



# **Merchant Services Guide**

**Taking Advantage of the CyberCash Payment Services**

**Version 1.0**

---

© 1998 CyberCash, Inc. All rights reserved.

CyberCoin is a registered trademark of CyberCash, Inc. CyberCash, CashRegister, Secure Internet Payment System, and the CyberCash logo are trademarks of CyberCash, Inc. All other products mentioned in this document are trademarks of their respective companies.

March 1998

Draft

---

# Table of Contents

---

<b>List of Tables</b> .....	ix
-----------------------------	----

## **Introduction**

Purpose .....	1
Audience .....	1
Style Conventions .....	2
What's in this Document .....	3
Getting Help .....	4

## **Part I: Understanding I-Commerce**

### **CHAPTER 1**

### **What Is I-Commerce?**

Providing a Trusted Shopping and Payment Environment	7
Competing on a Level Playing Field .....	8
Using a New Delivery Channel .....	8
Enhancing Consumer Relationships .....	8
Understanding Payment Issues .....	9
Meeting Consumer Needs .....	9

Watching the Shop . . . . .	9
Taking a Cue From the Physical World. . . . .	10
Creating a Trusted Online Payment Infrastructure . . . . .	10
Understanding How the CyberCash System Works . . . . .	11
Understanding the CashRegister Setup . . . . .	11
Understanding the Gateway Server . . . . .	12
Following a Credit Transaction . . . . .	13
Capturing and Completing a Transaction . . . . .	14
Monitoring Transactions . . . . .	14
Understanding Security Details . . . . .	14
Understanding New Security Initiatives . . . . .	14
Understanding Payment Services . . . . .	15
Promoting Secure Online Payments . . . . .	16
What's the Bottom Line . . . . .	17

## CHAPTER 2

### Enabling Your I-Commerce Store

Building Your Storefront . . . . .	19
Making Choices About Your Web Server . . . . .	20
Getting Connected . . . . .	21
Choosing a Location for Your Web Server . . . . .	23
Using Enabling Software Applications . . . . .	25
Choosing a Database . . . . .	26
Choosing Payment Software . . . . .	26
Deciding What Goes on Your Web Site . . . . .	27
Understanding Firewalls . . . . .	27
Understanding Shipping and Fulfillment. . . . .	28
Integrating Your Storefront with the CashRegister . . . . .	28
Setting Up a Merchant Account with a Financial Institution . . . . .	29
What Does the Financial Institution Do?. . . . .	29
Why Do You Need a Merchant Account? . . . . .	29
Processing Payments When the Card Isn't Present. . . . .	30
Understanding Credit Card Processing Models . . . . .	30
Understanding Credit Card Processing Models . . . . .	32
Registering with CyberCash . . . . .	34

---

---

**CHAPTER 3**

**I-Commerce Concepts and Planning**

Understanding a Merchant Offer . . . . .	35
Collecting Credit Card Data. . . . .	36
Understanding Fulfillment. . . . .	38
Soft Goods. . . . .	38
Hard Goods . . . . .	39
Understanding Receipts . . . . .	40
Static Receipts . . . . .	41
Dynamic Receipts . . . . .	41
Understanding Transaction Flows and Settlement. . . . .	41
Understanding the Terminal Capture Processing Flow . . . . .	41
Understanding the Host Capture, AuthCapture Processing Flow . . . . .	45
Understanding the Host Capture, PostAuthCapture Processing Flow . . . . .	47
Performing Settlement Tasks . . . . .	50
Understanding Pending Transactions and Timeouts . . . . .	51
Planning Your MCK Implementation . . . . .	52

**CHAPTER 4**

**Part II:  
Using CyberCash Payment Services**

**Getting a Merchant Account**

What Do You Do with Your Account Information? . . . . .	57
---	----

**CHAPTER 5**

**Using Integrated Merchant Registration (IMR)**

Understanding the Registration Process . . . . .	60
Understanding the IMR . . . . .	60
Downloading, Installing, and Configuring the MCK . . . . .	61
Accessing the IMR. . . . .	62
Updating Your IMR Information . . . . .	63
Understanding the IMR API . . . . .	63

**CHAPTER 6**

**Installing the MCK**

Understanding the MCK . . . . .	66
---------------------------------	----

Understanding MCK Installation . . . . .	66
Installing the MCK in Stand-Alone Mode . . . . .	67
Setting Up a Merchant in Stand-Alone Mode . . . . .	67
Installing the MCK in Shared-Server Mode . . . . .	68
Understanding Common Files . . . . .	70
Setting Up a Merchant in Shared-Server Mode . . . . .	70
Understanding the MCK Development Environment	73
MCK System Requirements . . . . .	73
Operating Systems . . . . .	73
Programming Languages . . . . .	74
Understanding the MCK Data Structure . . . . .	75

**CHAPTER 7**

**Customizing the MCK**

Before You Begin . . . . .	87
Understanding Browser Connect and Direct Connect	88
Understanding Browser Connect Transactions . . . . .	88
Understanding Direct Connect Transactions . . . . .	89
Understanding Browser Connect and Direct Connect Scripts . . . . .	90
Customizing Your MCK Scripts . . . . .	92
Customizing sslformpay.* . . . . .	93
Customizing mswalletpay.* . . . . .	95
Customizing ccwalletpay.* . . . . .	97
Customizing directpaycheck.* . . . . .	99
Understanding directpaycredit.* . . . . .	102
Customizing fulfillment.* . . . . .	105
Customizing notification.* . . . . .	106
Customizing the CCMerchant.* and CCMerchantCustom.* Modules . . . . .	106
Customizing Settlements and Returns . . . . .	109

**CHAPTER 8**

**Testing Your Storefront Integration . . . . .111**

**CHAPTER 9**

**Networking and Security**

Understanding Direct Connect Security . . . . .	113
Using HTTP Proxies . . . . .	114
Understanding Browser Connect Security . . . . .	114

	Understanding IMR Security . . . . .	115
	Storing Merchant Data . . . . .	116
<b>CHAPTER 10</b>	<b>Advanced Topics</b> . . . . .	<b>117</b>
	<b>Appendixes</b>	
<b>APPENDIX A</b>	<b>Using the CyberCash Payment Services API</b>	
	Understanding Standard Output Fields . . . . .	123
	Using batch-commit . . . . .	127
	Using batch-prep . . . . .	132
	Using batch-query . . . . .	140
	Using batch-unroll . . . . .	145
	Using card-query . . . . .	153
	Using checkauth . . . . .	156
	Using checkreturn . . . . .	158
	Using mauthcapture . . . . .	160
	Using mauthonly . . . . .	162
	Using postauth . . . . .	165
	Using query . . . . .	167
	Using retry . . . . .	174
	Using return . . . . .	177
	Using void . . . . .	180
	Understanding Error Codes . . . . .	182
<b>APPENDIX B</b>	<b>Name-Value Pair Variables</b> . . . . .	<b>193</b>

---

---

**Draft**

---

# List of Tables

---

<b>TABLE 1:</b>	Payment Options . . . . .	15
<b>TABLE 2:</b>	Comparing Physical and Online Stores . . . . .	20
<b>TABLE 3:</b>	Comparison of Dedicated Full Links . . . . .	22
<b>TABLE 4:</b>	Location Options . . . . .	24
<b>TABLE 5:</b>	Overview of Processing Models . . . . .	32
<b>TABLE 6:</b>	Processors and Processing Models . . . . .	33
<b>TABLE 7:</b>	Linking Payment to Fulfillment (Soft Goods) . . . . .	39
<b>TABLE 8:</b>	Linking Payment to Fulfillment (Hard Goods) . . . . .	40
<b>TABLE 9:</b>	Merchant Task Checklist . . . . .	50
<b>TABLE 10:</b>	Methods Used to Settle Transactions . . . . .	51
<b>TABLE 11:</b>	Proxy Parameters . . . . .	114
<b>TABLE 12:</b>	CyberCash Payment Services Messages . . . . .	121
<b>TABLE 13:</b>	Standard Merchant Output Fields . . . . .	124
<b>TABLE 14:</b>	Standard Gateway Output Fields . . . . .	125
<b>TABLE 15:</b>	batch-commit Input Fields . . . . .	127
<b>TABLE 16:</b>	batch-commit Output Fields . . . . .	129
<b>TABLE 17:</b>	batch-prep Input Fields . . . . .	133
<b>TABLE 18:</b>	batch-prep Output Fields . . . . .	136
<b>TABLE 19:</b>	batch-query Input Fields . . . . .	140

<b>TABLE 20:</b>	batch-query Output Fields . . . . .	141
<b>TABLE 21:</b>	batch-unroll Input Fields . . . . .	145
<b>TABLE 22:</b>	batch-unroll Output Fields . . . . .	148
<b>TABLE 23:</b>	Card-query Input Fields . . . . .	153
<b>TABLE 24:</b>	card-query Output Fields . . . . .	154
<b>TABLE 25:</b>	checkauth Input Fields . . . . .	156
<b>TABLE 26:</b>	checkauth Output Fields . . . . .	157
<b>TABLE 27:</b>	checkreturn Input Fields . . . . .	158
<b>TABLE 28:</b>	checkreturn Output Fields . . . . .	159
<b>TABLE 29:</b>	mauthcapture Input Fields . . . . .	160
<b>TABLE 30:</b>	mauthcapture Output Fields . . . . .	162
<b>TABLE 31:</b>	mauthonly Input Fields . . . . .	162
<b>TABLE 32:</b>	mauthonly Output Fields . . . . .	164
<b>TABLE 33:</b>	postauth Input Fields . . . . .	165
<b>TABLE 34:</b>	postauth Output Fields . . . . .	166
<b>TABLE 35:</b>	query Input Fields . . . . .	167
<b>TABLE 36:</b>	query Output Fields . . . . .	170
<b>TABLE 37:</b>	retry Input Fields . . . . .	175
<b>TABLE 38:</b>	retry Output Fields . . . . .	177
<b>TABLE 39:</b>	return Input Fields . . . . .	177
<b>TABLE 40:</b>	return Output Fields . . . . .	179
<b>TABLE 41:</b>	void Input Fields . . . . .	180
<b>TABLE 42:</b>	void Output Fields . . . . .	181
<b>TABLE 43:</b>	Common Error Codes . . . . .	182
<b>TABLE 44:</b>	Less Common error Codes . . . . .	190
<b>TABLE 45:</b>	Name-Value Pair Variables . . . . .	194

---

# Introduction

---

---

## Purpose

This guide describes provides a detailed overview of Internet commerce, the options you can choose in setting up an online store of your own, and advanced setup scenarios.

---

## Audience

This guide is intended for Merchant Development Partners (MDPs), Tech Partners (TPs), and merchants. To use this guide, you need computer programming experience. Specifically, you should be able to write programs in C, C++, Perl, or ASP, using the following concepts:

- data types and data structures (for example: integers, strings, arrays, and name/value pairs)
- control structures (for example: if/then/else, while)
- functions, procedures, and parameters

- structured or object-oriented design
- CGI programming using the HTTP POST method

If you are unfamiliar with these terms and audience requirements, you should consult MDP to help you with your store setup or use *Merchant Connection Kit: User Guide*.

---

## Style Conventions

This section describes the styles that are used in this document.

**Bold** type indicates items such as file names, window names, and buttons.

*Italics* type indicates a reference to another section of this document, a reference to another document, or terms that are defined within the text.

< > (angle brackets) indicate variables in user input. Information inside the angle brackets should actually reflect information specific to your setup.

Information that you will see as output on a screen, information that you must input, and system prompts are indented and in a smaller typeface than the document's body text.

```
xcopy autoexec.bat autoexec.bak
```

**NOTE:** Indicates suggestions or additional, detailed, information.

**!!!** Indicates actions you must take or avoid for the system to operate properly.

---

## What's in this Document

The chapters in this document contain the following information:

Part I, *Understanding I-Commerce*, describes what I-Commerce is and how it works, and the benefits the CyberCash system brings to your setup.

- Chapter 1, *What Is I-Commerce?*, defines I-Commerce and explains how you can provide a trusted shopping and payment environment for your customers.
- Chapter 2, *Enabling Your I-Commerce Store*, discusses the steps you need to take to get started as a CyberCash merchant.
- Chapter 3, *I-Commerce Concepts and Planning*, provides detailed information about a number of I-Commerce concepts. It further discusses using the MCK for implementation.

Part II, *Using CyberCash Payment Services*, discusses the actions you take to enable your store to use CyberCash Payment Services, and how to use the API to CyberCash Payment Services.

- Chapter 4, *Getting a Merchant Account*, explains how to get a merchant account, along with the information you will need.
- Chapter 5, *Using Integrated Merchant Registration (IMR)*, describes how to use the online Integrated Merchant Registration (IMR) web site to register to become an officially recognized CyberCash Merchant.
- Chapter 6, *Installing the MCK*, explains what the Merchant Connection Kit is, how to install it, and how to set up multiple merchants.
- Chapter 7, *Customizing the MCK*, describes different ways to build CyberCash payment capabilities for your electronic storefront using the Merchant Connection Kit (MCK).
- Chapter 8, *Testing Your Storefront Integration*, is a placeholder for information about running test transactions against your storefront.
- Chapter 9, *Networking and Security*, discusses networking and security issues in relation to the Merchant Connection Kit (MCK) and Integrated Merchant Registration (IMR). It also presents tips and issues for multiple merchant environments.
- Chapter 10, *Advanced Topics*, is a placeholder for a discussion of advanced topics. Please review the topic suggestions listed in this chapter, and send us your ideas!

- Appendix A, *Using the CyberCash Payment Services API*, describes the API messages you can use to perform the functions available on the administrative interface to CyberCash Payment Services.
- Appendix B, *Name-Value Pair Variables*, describes the merchant offer (mo), credit payment information (cpi), and proof of purchase (pop) variables.

---

## Getting Help

If you have questions during installation, configuration, or web store integration, refer to the web sites:

**MDPs:** [mdpnet.cybercash.com](http://mdpnet.cybercash.com)

**TPs and Merchants:** [www.cybercash.com](http://www.cybercash.com)

---

**Part I:  
Understanding  
I-Commerce**

**Draft**



# What Is I-Commerce?

---

This chapter defines I-commerce and explains how you can provide a trusted shopping and payment environment for your consumers.

---

## Providing a Trusted Shopping and Payment Environment

Today, businesses of all sizes—from Fortune 500-class corporations to the online equivalent of Mom and Pop specialty stores—are setting up shop in cyberspace. Most of the credit goes to the explosive popularity of the graphically oriented World Wide Web portion of the Internet. In the words of Forrester Research, Inc., "The World Wide Web has almost single-handedly transformed the Net from a members-only sandbox into a gigantic crossroads with strip malls, nouveau info-publishers, and EDI depots."

Yet, over its brief history, Internet, or Web, commerce has been characterized more by "window shopping" than by real buying. For the most part, the reluctance of consumers to buy goods and services online has been due to the lack of a secure and convenient Internet payment structure—a structure that consumers can trust, and that supports the interactive nature of online shopping. Fortunately, for both Internet merchants and their growing numbers of consumers, this problem is being resolved.

### Competing on a Level Playing Field

For merchants, one of the major attractions of the Internet—which knows no time zones or physical boundaries—is its global reach. *Fortune* magazine has declared that the Internet is already so large that it is almost "a force of nature." There is no need for brick and mortar investments or immense payrolls to achieve a global presence. On the level playing field of the Internet, entrepreneurial startups compete successfully with the largest conglomerates.

### Using a New Delivery Channel

Existing businesses, as well as startups, are moving onto the Internet in droves. Here is a chance for established merchants to extend their businesses, develop new markets, provide consumers with added convenience, and sidestep rapidly rising marketing expenses, including the costs of newsprint, advertising space, and face-to-face sales encounters. I-Commerce also provides you the opportunity to keep your store open 24 hours a day, seven days a week.

### Enhancing Consumer Relationships

In addition to reaching large numbers of potential consumers, the Internet allows you to market and sell to any consumer as an individual. Today's Web servers capture every detail of a consumer's visit to a Web site, and you can then use this information to tailor a unique, interactive marketing experience for each consumer at every visit. For example, if a consumer bought jazz CDs on his last visit, you can show the same consumer new jazz selections on his next visit.

Not since the days when a merchant could know every consumer by name has there been such an opportunity for getting close to consumers in order to meet their specific needs.

## Understanding Payment Issues

The universality, interactivity, and convenience of Internet shopping traditionally breaks down at payment time. Consumers have been justifiably nervous about sending credit card numbers over the Internet—card numbers might be intercepted, or the merchant might be a "fake store front" simply collecting card numbers. Thus, the typical method of payment has been for the consumer to pick up the telephone, call the merchant's toll-free number, and then verbally communicate the card number. Spontaneity and convenience, and their positive effect on sales, are lost in this process. In response, various electronic solutions have sprung up. However, they typically rely on special financial institution accounts and/or a pre-determined relationship between a consumer and a merchant. This severely limits the pool of available consumers and has done little to advance the state of universal Internet commerce.

## Meeting Consumer Needs

To successfully engage in online selling, you must provide for the following consumer needs:

- a fast, easy payment process that puts convenience and spontaneity back into the equation
- a trusted payment environment that guarantees security and privacy
- a variety of payment options (credit cards, electronic checks, and electronic cash)

## Watching the Shop

While you need the ability to accept a range of payment options, you also need the assurance that you will get paid. Can the consumer's identity be validated? Is this really the consumer's credit card? You also need links between your payment function and your various back-office functions such as inventory and accounting systems. You need ways to easily handle chargebacks, returns, and end-of-day reconciliation.

## Taking a Cue From the Physical World

All the merchant and consumer needs mentioned in this section are being successfully met in the physical world of in-store sales by today's automated point-of-sale (POS) systems. These systems, many of them consumer activated, are trusted by hundreds of millions of consumers as they purchase from supermarkets, retail stores, restaurants, gas stations and so on. The Internet merchants that extend this type of universal payment functionality, convenience, and trust to their online consumers will be the ones who will prosper from the Internet commerce revolution.

---

## Creating a Trusted Online Payment Infrastructure

The CyberCash Secure Internet Payment Service enables you to finally provide your consumers with a viable online payment mechanism. It does this by extending the well-trusted automated POS paradigm to online transactions, with fees that are competitive with existing in-store electronic payment systems. Like today's automated POS systems, the CyberCash system represents a front end to the existing infrastructure of financial networks, financial institutions, and payment processors. Unlike physical POS systems, CyberCash substitutes an online payment mechanism for an in-store payment terminal.

To meet the your needs and those of your consumers, the CyberCash system is:

- **Universal**—Any consumer with a valid credit card can take advantage of the CyberCash system to make payments using that card to buy from any certified CyberCash merchant without having any prior relationship. Consumers can also make payments with electronic checks and electronic cash.
- **Convenient**—Consumer and merchant transactions are performed online, and are typically concluded in less than 20 seconds.
- **Automated**—The CyberCash system supports hands-off, lights-out operations to meet the needs of continuous, around-the-clock commerce.
- **Secure**—Card information is automatically encrypted using triple DES encryption.
- **Exportable**—The CyberCash system can be used around the world and is approved for export by the United States government.

## Understanding How the CyberCash System Works

The CyberCash system gives you the option of collecting payment information through SSL-enabled links. You can also enter credit card information manually. This latter option is helpful if you want to use the CashRegister to take orders placed over the telephone.

The CyberCash system uses the metaphors of physical payment: consumer wallets, merchant cash registers/POS terminals, and gateways to private payment networks. Following are the main components:

- any SSL-based browser (optionally CyberCash Wallet and Microsoft Wallet)
- Merchant Connection Kit (MCK) and CashRegister for Cyber-merchants
- CyberCash Gateway servers linked to existing financial networks

You can use SSL-forms to collect payment information, or allow customers to use the CyberCash Wallet or Microsoft Wallet. The CyberCash Wallet uses a Web-based interface that provides consumers the option to securely store credit card information to purchase goods or services from stores on the World Wide Web. Consumers can also make purchases without storing credit card information. The Wallet application is not local to the consumer's PC, but is dynamic through the use of Java.

No matter what method of payment collection you use, you have a ready pool of potential consumers who can quickly and easily purchase goods and services from a CyberCash-enabled Web site. Just like a physical store, you can allow a choice of payment options—credit cards, electronic checks, or electronic cash.

## Understanding the CashRegister Setup

The MCK is a set of scripts that merchants use to connect their storefronts with the CashRegister administration interface. The MCK can be downloaded from the CyberCash Web site at:

[www.cybercash.com](http://www.cybercash.com)

You can integrate SSL forms or Wallets (CyberCash or Microsoft) with your storefront, which interfaces with CyberCash Payment Services and the Gateway. In effect, CyberCash Payment Services assumes the familiar role of your electronic cash register/POS system for Internet transactions. You can use it to perform the following tasks:

- submit sales, voids, and returns
- submit credit card authorization requests
- submit batched transactions for capture (terminal capture merchants)
- submit post-authorization capture requests (host capture merchants)
- query the database
- query for credit card information

Besides enabling secure credit card processing, CyberCash Payment Services provides you with important administrative functions, including manually processing credit card payments, checking transaction status, and using database functions to support balancing, accounting, and tracking inventory.

You can also process credit card payments for toll-free telephone orders as well as faxed and emailed orders. This means that a single program can handle all of your "card-not-present" transactions. Importantly, CyberCash Payment Services is designed to do the following functions:

- handle multiple concurrent payment transactions as your business expands
- easily integrate with merchant web sites—meaning that your applications don't have to be forced into an inflexible framework
- support multiple online stores

### **Understanding the Gateway Server**

Bridging the Internet and the financial institution world, the Gateway server is the software and hardware platforms operated by CyberCash to provide the secure link between you and (1) your consumers residing on the Internet and (2) your financial institution, which uses existing financial payment networks. From a financial institution's point of view, CyberCash transactions arrive looking and behaving exactly like traditional POS terminal transactions.

The Gateway server provides firewall protection, message translation between Internet and financial network protocols, the maintenance and authentication of CyberCash IDs, merchant-originated charges (toll-free number, fax, or email orders), and comprehensive message tracking.

### Following a Credit Transaction

To better understand how the CyberCash system benefits you, you should follow a consumer and a merchant through a typical credit card transaction— automated from end to end, and completed in just 15 to 20 seconds. Figure 1 illustrates this flow.

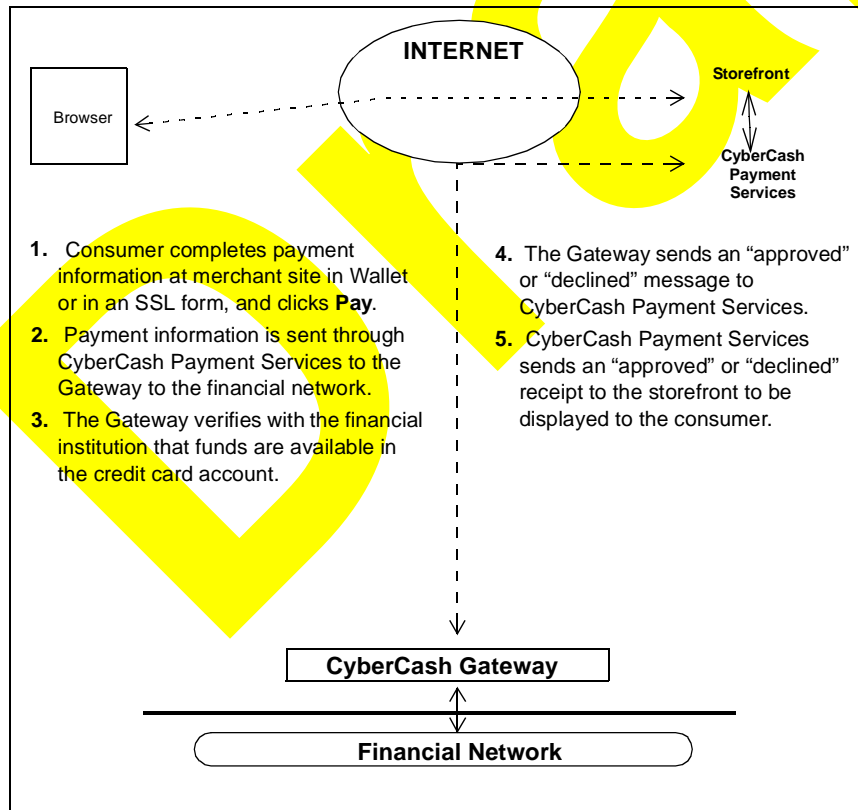


FIGURE 1: MAKING A CREDIT CARD PURCHASE

## Capturing and Completing a Transaction

To complete the transaction, CyberCash Payment Services passes an acknowledgment of payment to the storefront, where it is generally displayed as a web page on the storefront. A transaction can be captured and posted to your account while your consumer is still online, or later if you cannot ship the purchased product immediately (if you have questions regarding the management of your transactions around shipment of goods, contact your financial institution about the rules and regulations for fulfillment). The CyberCash system supports host and terminal capture processing methods (along with a full complement of exception handling capabilities for either method), depending on your individual financial institution's systems. For a complete description of processing methods, see *I-Commerce Concepts and Planning*, page 35.

## Monitoring Transactions

CyberCash Payment Services track your transactions. You can search for transactions individually or grouped according to specified criteria. For example, you can search for transaction within a date range, or for particular card types (MasterCard, Visa, Discover, and so on).

You can also query for additional transaction details, such as information you will need for chargebacks. The transaction records in this database are encrypted and made available only under password control. This password protection feature further protects you, your consumers, and your financial institution because it helps prevent a merchant server from being compromised.

## Understanding Security Details

CyberCash transactions are protected by triple DES encryption. This powerful encryption makes it difficult for hackers to unscramble or crack into information.

## Understanding New Security Initiatives

CyberCash is actively working with the credit card associations and is committed to complying with standardized security initiatives for Internet commerce as they evolve and are deployed. This includes the Secure Electronic Transaction specification (SET) defined by Visa and MasterCard.

## Understanding Payment Services

CyberCash offers a variety of payment services. You can accept credit cards, electronic cash, and electronic checks. [Table 1](#) outlines these payment services.

**TABLE 1: PAYMENT OPTIONS**

OPTION	DESCRIPTION
Credit Card	Allows you to accept credit card payments over the Internet. You can accept payments through the Wallet or SSL forms, or you can manually enter payments using CyberCash Payment Services administrative interface. This eliminates the need to purchase any other point of sale hardware or software to complete your telephone or fax orders.
PayNow	Allows you to present bills to your consumers, which can be paid with electronic checks. Electronic checks offer security to the merchant because an electronic check cannot be accepted if funds are not available in the consumer's checking account.
CyberCoin	Allows you to sell inexpensive items (typically \$.025 to \$5.00) over the Internet. This service is ideal for electronic information or graphics files.

## Promoting Secure Online Payments

The number of potential consumers flocking to the Internet and the World Wide Web is phenomenal. International Data Corporation (IDC), a market research firm, projects that the number is growing at a compound annual rate of 39%, and that it will "reach just shy of 200 million users in 1999." The commercial online sites that provide a convenient and trusted payment environment to as much of this audience as possible obviously possess a competitive advantage—an advantage akin to that enjoyed by retail stores and restaurants that display MasterCard, Visa, Discover, and American Express stickers in their shop windows. The advantage is actually greater in cyberspace because there are so few viable payment options. The CyberCash system is designed to be as universal as possible, keeping the following points in mind:

- You can design your own SSL-form (or use the CyberCash Wallet or Microsoft Wallet).
- Anyone surfing the Web can shop at your site at any time, using the payment methods you allow them to use.
- Consumers do not need a prior relationship with you or with CyberCash to use an SSL-form or a Wallet.
- Consumers can use electronic check and electronic cash payment options.
- The powerful encryption techniques used by CyberCash have been licensed by the U.S. government for export.

The result is the Internet's largest ready-made pool of consumers. Authorized merchants typically place the CyberCash logo in their sites' "shopping pages"—the same way physical sites place credit card stickers—to assure consumers that their payments will be fast, easy, and secure.

---

## What's the Bottom Line

Ultimately, the online revolution will be a consumer revolution. Meeting consumers' needs for goods and services is only one half of the battle. Meeting their needs for trusted payment is the other. Consequently, online payment systems must meet the following criteria:

- Provide the same type of conveniences (including payment options) that draw consumers to the Internet and other online networks to begin with
- Foster the same, if not greater, degree of trust as physical payment systems—both in the minds of the consumer and the merchant
- Support the interactive nature and universal reach of cyberspace and online commerce

While there is little doubt that commerce will flow freely over the Internet by the end of this decade, the CyberCash system is alone in providing merchants the unrestricted ability to enjoy convenient and secure payments today.

Draft

# Enabling Your I-Commerce Store

---

This chapter discusses the steps you need to take to get started as a CyberCash merchant. There are four main tasks:

- building your storefront
- integrating your storefront with CyberCash Payment Services using the Merchant Connection Kit (MCK)
- setting up a merchant account to accept credit cards
- registering with CyberCash using the Integrated Merchant Registration (IMR)

---

## Building Your Storefront

Building a store on the Web is the online equivalent of establishing a physical storefront on Main Street or in a mall—just a great deal easier and less expensive.

Essentially, the components of an online store all have their counterparts in the physical world. [Table 2, page 20](#), shows this correlation.

**TABLE 2: COMPARING PHYSICAL AND ONLINE STORES**

PHYSICAL STORE	ONLINE STORE
Stores are built with raw materials like bricks and mortar.	Stores are built with computer hardware and software.
Shelves and cash registers physically differentiate retail stores from each other.	HTTP and payment software are the shelves and cash registers of the online world
Promotions and store displays bring in the consumers, get them to reach for shopping carts, and clinch the sales.	Applications software and Web content promote goods and services, and get consumers to make their purchases.

Of course, there's also the street or parking lot in front of a physical store to provide consumers easy access to the good on sale inside. In the online world, high bandwidth or fast access to the Internet itself fulfills that need. And then there are security devices—the physical locks, mirrors, and even security guards. These are present in the online world too, in the form of firewall software and data encryption schemes. Firewall software protects your back office from intruders. Encryption uses algorithms to scramble your sensitive data, including payment information, so no one can read it.

### **Making Choices About Your Web Server**

The first thing you need to do as an aspiring Internet merchant is to put up your store's walls and roof. In the Internet world, this means making choices about your Web server. *Server* is one of those ambiguous computer terms that takes its meaning from its context. In this case, it is essentially the combined hardware, operating system, and Web server software you web site's content will reside.

**What Kind of Hardware and Operating System Do You Need?** The hardware portion of your Web server is simply the computer or "the box." This can be a single processor machine, such as a PC or a high-end workstation. It can also be a multi-processor, or even multi-processor system, designed for high-performance, mission-critical computing.

Historically, Web servers have been high-end, engineering workstations that run some version of the UNIX operating system. This is not surprising, given the traditionally technical nature of the Internet and its first citizens. High end UNIX workstations are still the most popular choice because of the abundance of development tools and programming talent.

Recently, Microsoft Windows NT has gained popularity as an operating system for Web servers. Among its advantages are the ability to run on Intel-based PCs, its graphical interface, its familiar configuration process, and the large number of consultants versed in Windows NT who can help implement an NT solution.

**What Kind of Web Server Software Do You Need?** Web server software resides on top of the hardware and operating system, and provides the very important HTTP component—the communications language of the World Wide Web. It is HTTP that provides you with the ability to click on key words or items and then be automatically transported to information that may reside on a completely different server thousands of miles away.

Web server software ranges in price from free to several thousands of dollars. Generally, the more expensive packages provide more features. A few of the more popular Web server software packages are listed below:

- Netscape
- NCSA
- O'Reilly & Associates (for Windows NT)
- Microsoft (for Windows NT)

## Getting Connected

Once you have built your store and installed your Web server, it's time to get on the Internet, typically through an Internet Service Provider (ISP), and decide where to physically locate your Web server. These two steps are closely intertwined, and both have very much to do with the data communication issue of bandwidth.

*Bandwidth* refers to the size of your link, or data pipeline, to the Internet. Basically, the greater the bandwidth of your link, the faster your consumers can access your services, and the more concurrent consumers you can handle without slowing down your system. The amount of interactive traffic (back and forth communication with consumers) you anticipate has a bearing on the size of the link, as does your type of business. For example, if your consumers will download detailed images or large sound files, you probably want a high-speed, high-bandwidth link.

You need to be concerned about the bandwidth of two distinct data pipelines:

- the link between your Web server and your ISP
- the link between your ISP and the Internet itself

**From Your Web Server to Your ISP .** There are several options you can choose when considering this link. Your first choice is deciding between a link that is available on demand or a dedicated full link. An on-demand link is activated only when someone calls you or when you call out—just like your standard telephone service. A dedicated full link, on the other hand, is always active (and you pay for it, whether there is traffic on it or not). If you are expecting a lot of consumers, a dedicated full link is the best choice. Options for high-bandwidth dedicated full links include:

- ISDN—56 Kbits per second (Kbps) up to 128 Kbps
- Frame Relay—56 Kbps up to T1 (1.55 Mps)
- Dedicated Point-to-Point—56 Kbps up to T3 (45 Mps)

Table 3 discusses the pros and cons of these links.

**TABLE 3: COMPARISON OF DEDICATED FULL LINKS**

LINK	DESCRIPTION
ISDN	ISDN was designed as a dial-up or on-demand connection, but can be used as a dedicated connection. It is really only cost-effective if your ISP is located within 12 miles of one of your telephone company's central offices. Otherwise, you will be paying long distance phone rates.
Frame Relay	Frame Relay is more cost-effective than ISDN. Rates are not tied to distance; instead, you pay a flat fee. You can also enjoy much higher bandwidth connections, up to T1.
Dedicated Point-to-Point	Dedicated Point-to-Point provides the highest bandwidth of all, up to T3. Like Frame Relay, you pay a flat rate, but you are charged a set number of dollars per mile depending on how far your ISP is located from you.

**From your ISP to the Internet .** The other half of your connection lies between your ISP and the Internet itself. To avoid traffic slowdowns, it is critical that your ISP have a robust connection to the Internet.

You should be prepared to ask your ISP the following questions:

- How fat is your pipeline to the Internet—T1 or T3? (T3 is optimal.)
- How close are you to a Network Access Point (NAP)?

Only major backbone carriers—including Sprint, MCI, Agis, UUNet, PSInet, and WillTel—connect to NAPs. ISPs contract with these carriers for NAP access, which automatically puts them one hop away from a NAP. There may be other middlemen involved, adding additional hops. Essentially, when you select an ISP, you want to minimize the number of hops involved in connecting to a NAP.

### Choosing a Location for Your Web Server

While you sort out connection issues, you need to decide where to locate your Web server. You have three basic options:

- Your own site
- A co-location
- A Web hosting service

[Table 4, page 24](#), discusses these options.

**TABLE 4: LOCATION OPTIONS**

LOCATION OPTION	DESCRIPTION
Onsite	<p>You typically own the hardware and manage it. If you are a sizable organization, you might have an MIS department to help with management issues.</p> <p>Contact an ISP to connect you to the Internet. You also have to address both the issue of your link to the ISP, and the ISP's link to the Internet.</p>
Co-location	<p>This is an option typically offered by an ISP that allows you to still own the Web server. However, it actually resides at the ISP's site. You can manage the server yourself, either remotely or onsite if you want.</p> <p>The main advantage of co-location is that your Web server can be directly plugged into a very high-bandwidth connection to the Internet, meaning you don't have to worry about the link between you and the ISP.</p>
Web hosting service	<p>This option grew out of the worlds of the ISP and of commercial online services. Some systems integrators also offer hosting services. With Web hosting, you buy space on a server belonging to a third party, such as CompuServe. The server is at the host's location, but you are responsible for your Web site and its content.</p> <p>The major benefit is that you don't have to buy your own Web server. Presumably, the server that you rent space on is fairly powerful. Also, as with co-location, you can be plugged directly into a high-bandwidth connection to the Internet.</p>

When choosing a co-location service, ask the following questions:

- How many other Web sites are on the same server?  
A large number of sites might slow its performance.
- What kind of content do your "neighbors" provide?  
If you are sharing a server with a site that sells music or video clips, your available bandwidth may be drastically affected.

On the other hand, some of these services can provide added value in terms of content development and technical support that can outweigh performance and bandwidth issues.

### Using Enabling Software Applications

Another issue you need to address is enabling software applications. These Web applications are what really define your Internet storefront: differentiating it from the competition, presenting your wares, tracking consumers, and so on.

There are a variety of applications, from essential shopping cart applications that allow consumers to make selections, to sophisticated one-to-one marketing and access control capabilities. For example, Web servers capture every detail of a consumer's visit to your Web store. You can use this information to tailor a unique shopping experience for every consumer. For example, if a consumer bought jazz CDs on his last visit, you can show the same consumer new jazz CDs on his next visit. You could also allow valued, longtime consumers access to special sales areas.

Until recently, merchants had to build their own Web applications to perform these types of functions. These applications were developed behind the scenes and integrated into Web sites. Fortunately, tool kits are increasingly available to help. Nevertheless, thoroughly developing and integrating your Web site can still be time consuming and require intensive programming.

Recently, highly functional, off-the-shelf packages have appeared, such as BroadVision, Mercator, E-Shop and Connect Inc.'s OneServer. These integrated packages can help alleviate much of the work and expense of application integration and database programming.

## Choosing a Database

A database is the foundation for almost all enabling Web applications. A database is your store's filing cabinet. In it, you store consumer, order, and product information. Your consumers query it to quickly find the products they want to buy. Thus, a high performance database is critical to success.

Oracle, Informix, and Sybase all develop leading databases for UNIX systems. Microsoft Access is a leading choice for Windows NT.

**Search Engines** . Search engines work with databases to allow consumers to quickly find their way through your offerings (not to be confused with public services offered by companies such as Yahoo!). For example, your search engine can help a consumer find that size eight blouse, in teal, made of silk, and with French cuffs and faux pearl buttons. Leading search engines include:

- WAIS
- Verity
- Personal Librarian
- Excaliber

## Choosing Payment Software

Internet commerce is electronic. That means payment must also be electronic—in the form of credit cards, electronic check, or electronic cash.

CyberCash provides electronic payment software—the Internet equivalent of wallets and cash registers. We've created the Secure Internet Payment Service to make payments over the Internet convenient and secure. With our software, consumers can buy goods online with full confidentiality and protection; merchants can sell goods and services and receive assured instantaneous payment.

As a CyberCash-enabled merchant, you gain the following abilities:

- Dealing with any financial institution or payment processor you choose
- Operating your store in a fully unattended mode
- Accepting secure payments from the Internet's largest pool of potential consumers

The MCK was designed to easily integrate your storefront with the CashRegister. This means that your applications and processes don't have to be shoehorned into an inflexible framework.

## Deciding What Goes on Your Web Site

Content creation is typically an ongoing process. Information providers, for example, are always adding new information and updating old. Hard good merchants are always adding new products and devising new ways to promote and merchandise them.

Consequently, content creation is becoming a rather large Internet-related business, involving both creative and technical personnel. The process can encompass:

- Converting existing text and multimedia content into HTML
- Copywriting to market new or to repurpose existing content
- Designing graphics

To find this kind of help, you can turn to advertising or graphics art agencies, or to consultants who can pull together the creative people and the right programmers. CyberCash, through its Merchant Development program, can help merchants find the content creation help they need.

## Understanding Firewalls

Firewall software allows you to lock your Web store and keep intruders out. In the online world, intruders can take the shape of mischief makers, thieves, and viruses. Generally, with commercially oriented Web sites, the Web server sits outside of the firewall (after all, you don't want to filter out legitimate consumers and messages). The firewall protects the critical company data that may be part of your Web application scheme but that doesn't need to be made available to visitors. Your back office accounting system, for example, should be secured against intrusion.

For more information about security, see [Networking and Security](#), page 113.

## Understanding Shipping and Fulfillment

If you sell hard goods, as opposed to information or services, then you need to consider shipping and fulfillment. Often, merchandise is shipped from a third party's location, such as a fulfillment house's warehouse. Direct computer-to-computer connections over private Electronic Data Interchange (EDI) networks can streamline the fulfillment process. On a simpler and less costly level, you can use encrypted email—for example, using a package such as Pretty Good Privacy (PGP)—to expedite fulfillment.

For more information about fulfillment, see *Understanding Fulfillment*, page 38.

---

## Integrating Your Storefront with the CashRegister

The Merchant Connection Kit (MCK) is a set of scripts that allows you to integrate your storefront with the CashRegister at CyberCash. The MCK installation builds a very basic web page that accepts and processes transactions. If you want the CashRegister to work with your existing web site, you must customize the MCK scripts to match the needs of your web site, and integrate the CashRegister functionality into your storefront.

The MCK is discussed in further detail in *Installing the MCK*, page 65, and *Customizing the MCK*, page 87.

The following list outlines the three main tasks you need to perform to integrate your storefront with the CashRegister:

1. Customize your store pages.
2. Configure your web server.
3. Modify your HTML files.
4. Customize your MCK scripts.

For basic setup instructions, refer to *Merchant Connection Kit: User Guide*.

## Setting Up a Merchant Account with a Financial Institution

As you build your web site, you need to contact your financial institution to set up a merchant account. The types of products sold in cyberspace are diverse, ranging from physical goods to online deliverable information and services. What you deliver, and how and when you deliver, affect the type of payment options that consumers will most often use, as well as how payments are captured and settled. For example, sellers of information may find their consumers using electronic cash, while larger ticket items may be paid for by credit cards in most cases. Also, if you are dealing with goods that will not ship immediately, refer to your financial institution about transaction settlement and fulfillment rules and regulations. Financial institutions can help you sort through these and other issues and deploy appropriate processing and settlement solutions in conjunction with CyberCash.

### What Does the Financial Institution Do?

Once your financial institution certifies you for online commerce, the financial institution will typically perform the following tasks:

- Issue you a terminal ID (TID) (a Terminal ID is the same type issued to a physical merchant's POS terminals)
- Issue you a Merchant ID (MID)

Your processor will communicate your MID and TID to CyberCash. This enables you to begin accepting secure credit card payments using the CyberCash system.

**NOTE:** CyberCash accepts merchant account information from processors only. This ensures that your information is protected.

While the financial institution is working on these tasks, you should integrate your storefront with the CashRegister using the MCK, and register with CyberCash using the Integrated Merchant Registration (IMR).

### Why Do You Need a Merchant Account?

Just as in the physical world, if you are going to accept credit card payments over the Internet, you have to have a merchant financial institution account and be authorized as a credit card merchant. You have to have a merchant account to accept credit cards.

**NOTE:** The CyberCash system will support phone and fax transactions as well as online transactions, allowing you to consolidate all your “card-not-present” transactions into one system. CyberCash Payment Services administrative features allow you to enter phone, fax, and mail transactions directly into the system quickly and easily.

## Processing Payments When the Card Isn't Present

There are many rules and regulations around accepting credit cards, whether you are a physical world merchant or an Internet merchant. As the Internet evolves, the credit card associations are creating new regulations related to Internet based transactions.

Internet-based credit card transactions are treated very similarly to mail order/telephone order (MO/TO) transactions. Internet transactions are “card-not-present” transactions, where you are not face-to-face with the consumer and are unable to physically check the consumer’s card for validity (for example, you cannot read the magnetic stripe, verify the signature, view the hologram, or take other measures). In response, the credit card associations suggest that Internet merchants use the address verification service (AVS). This tool was created to assist merchants in their decision process of whether to ship the goods. For more details on AVS, contact your financial institution.

The CyberCash system complies with the particular transaction information requirements of the major credit card associations, including supporting AVS.

## Understanding Credit Card Processing Models

There are two parts to a credit card transaction: the authorization and the capture of the transaction. The authorization is the electronic request message that is sent to the consumer’s credit card issuing financial institution for approval of the sale. The credit card issuer will return one of the following responses to the authorization request:

- Approval—transaction was approved
- Decline—transaction was not approved
- Referral—approval pending more info; call card issuer for assistance

If a transaction receives an approval (authorization number), the merchant can capture or submit the sale for financial settlement. The action of submitting a credit card sale for financial settlement is called the capture of the transaction. When you capture a transaction, the sale amount will be credited to the your deposit account through your acquiring financial institution and posted to the cardholder's credit card account.

Your acquiring financial institution will determine which capture method you will use based on your type of business. Contact your acquiring financial institution if you have questions about your designated capture method.

There are two types of capture methods: terminal capture (formerly known as authonly) and host capture (formerly known as authcapture). The difference lies in where the authorized transactions are stored awaiting capture and the action that is required to capture them.

Under the terminal capture method, the authorized transactions are stored at CyberCash until the you use the administrative interface to CyberCash Payment Services to create a batch and submit it for capture and settlement.

Under the host capture method, the authorized transactions are stored at the acquiring financial institutions or third party processor's host computer awaiting capture. However, there are actually two types of host capture. In the first type, the capture happens automatically at the end of the day without action by you. This type is known as host capture (AuthCapture). In the other, capture occurs only after action by the you. This second type is known as host capture (PostAuthCapture).

[Table 5, page 32](#), provides a description of each processing model and a sample business model that aligns with each type of processing.

**NOTE:** Your CyberCash-affiliated financial institution determines which processing model and processor you will use. [Table 5, page 32](#), is provided for informational purposes only.

**TABLE 5: OVERVIEW OF PROCESSING MODELS**

PROCESSING MODEL	DESCRIPTION	SAMPLE BUSINESS MODEL
Terminal Capture	Payments are authorized immediately and stored in a batch file at your store. Batch files are sent to the processor, through CyberCash, for settlement.	Merchant offers services or sells digital or physical goods for either immediate or future fulfillment.
Host Capture, (AuthCapture)	Payments are authorized and captured at the same time, and then stored in a batch file at the processor.	Merchant offers Internet connectivity services that consumers pay for each month using a credit card, or merchant sells digital goods, such as software, that are delivered to the consumer on the same day as the purchase (immediate fulfillment).
Host Capture, (PostAuthCapture)	Payments are authorized and captured in two different interactions, and then stored in a batch file at the processor.	Merchant sells physical goods and may not always ship the ordered goods on the same day the order was received (delayed or future fulfillment).

### Understanding Credit Card Processing Models

A credit card processor is a company that provides credit card processing, billing, reporting, authorization, and settlement services to merchants on behalf of acquiring financial institutions. CyberCash works with six processors including First Data Corporation (FDC), Global Payment Systems (GPS), Nova, Paymentech, Vital, and Wells Fargo. Each CyberCash merchant is serviced by one of these processors as designated by their acquiring financial institution.

Table 6, page 33, lists CyberCash-supported processors and the processing models they use.

**TABLE 6: PROCESSORS AND PROCESSING MODELS**

PROCESSOR	PROCESSING MODEL
First Data Corporation (FDC)	Terminal Capture
First USA Paymentech (FUSA or FUSAP)	Terminal Capture
Global Payment Systems Atlanta (GPS-ATL, formerly NDC)	Host Capture (AuthCapture)
	Terminal Capture
Global Payment Systems St. Louis (GPS-STL, formerly MAPP)	Host Capture (AuthCapture)
	Host Capture (PostAuthCapture)
NOVA	Host Capture (AuthCapture) (immediate fulfillment)
	Host Capture (PostAuthCapture) (future fulfillment)
Vital (VisaNet)	Terminal Capture
Wells Fargo	Host Capture (AuthCapture)
	Host Capture (PostAuthCapture)

Regardless of the processing model, each credit card payment transaction goes through three phases:

- authorization (of the purchase amount)
- capture (of the authorization to a batch file)
- settlement

For a detailed discussion of each processing model and their associated tasks, see [I-Commerce Concepts and Planning, page 35](#).

---

## Registering with CyberCash

The fourth task you need to perform to become an Internet merchant using the CyberCash system is to register with CyberCash using the IMR. The IMR is a web based interface that you use to provide CyberCash with your setup information.

You can also use the IMR to change your current setup. For example, if you use host capture processing and later switch to terminal capture processing, use the IMR to inform CyberCash of this switch.

You can use the IMR online or through an API. For a detailed discussion of each method, see [Using Integrated Merchant Registration \(IMR\)](#), page 59.

# I-Commerce Concepts and Planning

---

This chapter provides detailed information about a number of I-Commerce concepts. It further discusses planning your implementation. This chapter should help you answer the following questions:

- What is a merchant offer?
- How do you collect credit card data?
- What is meant by fulfillment?
- What are receipts, and how do you issue them?
- What is the payment transaction flow, and how do you settle payments?
- What are pending items and timeouts?
- What are your choices for implementing the MCK?

---

## Understanding a Merchant Offer

For all payments over the Internet, you need to present a *merchant offer* to your consumers. Simply explained, a merchant offer outlines the terms of your agreement with your consumers: what you will provide to them at what price. A merchant offer is presented as a web page and is the electronic commerce equivalent to an invoice.

The consumer needs to see only the terms of the deal (for example, T-shirt, \$10.00, and order ID123). You also need to provide the option to pay at this point, or give a time limit by which the purchase must be made.

Once the consumer clicks **Pay**, CyberCash, will receive a description of the transaction, including your signature, which contains your secret and/or other hashed information.

---

## Collecting Credit Card Data

Collecting credit card data is an important concept for you to understand for the following reasons:

- You need it to get paid.
- It provides a record of the payments that are due to you.

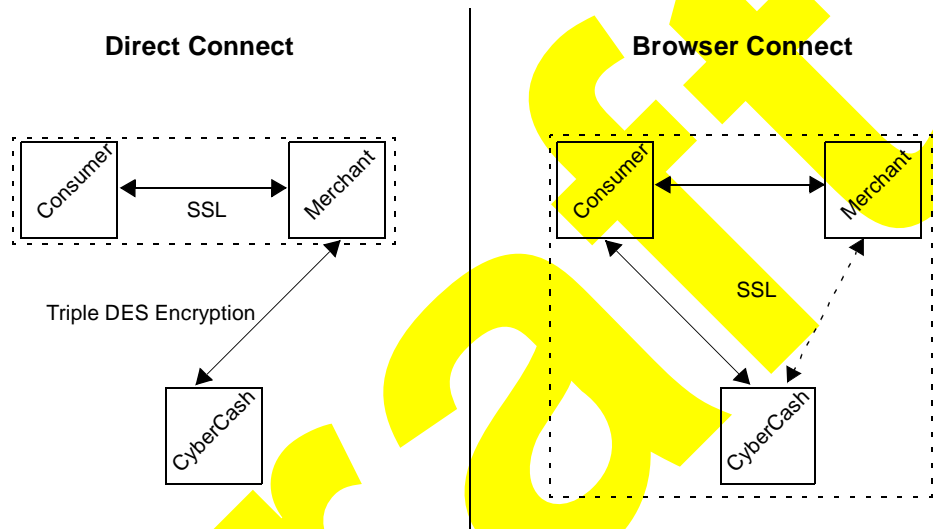
It is up to you whether you want to store payment data. CyberCash tracks your transactions, and you can access your records at any time by using the CyberCash Payment Services administrative interface, or by downloading your information from CyberCash using an API call. However, CyberCash collects a limited amount of information, and you might want to keep more detailed records. For example, you could store the names and addresses of your consumers, along with their buying habits. This would allow you to inform them of future sales promotions.

For secure electronic commerce, you should present your payment collection page (an SSL form or a wallet) from an SSL-enabled server. Your consumers should also use an SSL-enabled browser for their own security. You can collect payment data in the following ways:

- through a form (Direct Connect or Browser Connect)
- through the CyberCash Wallet (Direct Connect or Browser Connect)
- through the Microsoft Wallet (Direct Connect only)

**NOTE:** If you choose to collect payment through either Wallet, you can use a form to collect additional information from the consumer. Any additional information you collect is not needed by CyberCash to process a payment, but it might be useful to you for other purposes.

In each setup, the information you collect is handled differently. [Figure 2](#) illustrates the difference in information flow. For a detailed discussion about direct connect versus browser connect environments, see [Customizing the MCK](#), page 87.



**FIGURE 2:** DIRECT CONNECT VS BROWSER CONNECT

In the Direct Connect model, payment information flows from the consumer, to the merchant, and to CyberCash, and then back in a reverse flow. Using this model, you can use an API to perform the tasks you would do on the CyberCash Payment Server administration interface. In the Browser Connect model, the merchant requests payment information from the consumer, but the information is passed and processed between the consumer and CyberCash. You can check your transactions using the CyberCash Payment Services administration interface.

**NOTE:** In the illustration, note that “Consumer” represents the consumer’s web browser and “Merchant” represents you, using the MCK.

---

## Understanding Fulfillment

*Fulfillment* is providing goods or services to a consumer, fulfilling your end of the deal. CyberCash provides various mechanisms for linking a successful payment to the systems used to fulfill orders. You need to choose which mechanisms to use (depending on the kinds of goods being sold), when fulfillment will occur, and the systems available to fulfill orders and keep financial records.

In general, CyberCash returns a response indicating that the payment was accepted or declined. An accepted response is also known as a receipt or a proof-of-purchase. The response is returned via the Internet to a server known as a fulfillment center. Most often the fulfillment center is the same as the web storefront. However, the fulfillment center may be a web server operated by a different company on a different continent. The fulfillment center should check that the receipt is valid, using software provided in the MCK.

### Soft Goods

Digital goods and services provided over the Internet are often referred to as *soft goods*. Some examples include software, graphics, and other files downloaded to the consumer's computer, and subscriptions to entertainment and information services. Soft goods are normally provided to the consumer immediately after the payment is approved.

When a payment is approved, CyberCash Payment Services returns a receipt, also known as a proof-of-purchase. The receipt can be returned directly to your server, or the receipt can be hidden in an HTML page returned to the consumer. The consumer submits this page with the receipt to complete fulfillment. Fulfillment can come from the same server that was used for shopping, or from any other web server.

The scripts provided by CyberCash include examples of how to deliver a file or other digital payload to the consumer after payment is approved. [Table 7, page 39](#), shows the common ways that payment can be linked to fulfillment.

**!!!** Payload files must have a suffix. CyberCash does not support payloads that do not have suffixes.

**TABLE 7: LINKING PAYMENT TO FULFILLMENT (SOFT GOODS)**

SOFT GOODS	MERCHANT FULFILLMENT	THIRD PARTY FULFILLMENT
You get receipt in response to payment request and fulfills the order.	✓	
Fulfillment center receives receipt redirected from CyberCash by the consumer's browser.	✓	✓
Consumer gets the receipt hidden in an HTML page. The consumer clicks on items in the page to access his or her purchase.	✓	✓

**Hard Goods**

Physical goods, such as disk drives, wine, and hot tubs, are often referred to as *hard goods*. They may be delivered immediately upon payment or at a later time, shipped from one or more locations, or built after order is placed and shipped upon assembly.

As with soft goods, the receipt from CyberCash can be returned directly to your server, or the receipt can be sent to another web server that accepts fulfillment instructions. Either way, a web page should be returned to the consumer thanking him or her for the order and providing any other useful information. Fulfillment can be tracked using the same software used for your web store front, or data can be transferred to another system to trigger fulfillment.

With hard goods, your system should normally ensure that goods are shipped if payment is approved, even if the consumer leaves the web site prematurely. [Table 8, page 40](#), shows the common ways that payment can be linked to fulfillment.

**TABLE 8: LINKING PAYMENT TO FULFILLMENT (HARD GOODS)**

HARD GOODS	MERCHANT FULFILLMENT	THIRD PARTY FULFILLMENT
You get receipt in response to payment request and marks the order for delivery.	✓	
Fulfillment center receives receipt redirected from CyberCash by the consumer's browser.	✓	✓
CyberCash notifies the fulfillment center via HTTP in a reliable queuing process.	✓	✓
The fulfillment center communicates off the Internet with another system to initiate fulfillment.	✓	✓

## Understanding Receipts

The concept of a receipt is universal, in the virtual and the physical worlds alike. A receipt represents a record of the purchase from a merchant to a customer. A typical receipt includes one or more of the following items:

1. Thank you note
2. Business Name/Customer Support contact (phone number, email address, web site, and so on)
3. Order ID
4. Amount of transaction
5. Item(s) purchased

6. Date and time of purchase
7. Expected Delivery date

Receipts can be either static or dynamic. For example, items 1, 2, and 7 above can be static—they do not need to change for each transaction. Items 3, 4, 5, and 6 are dynamic—they change for each transaction.

### Static Receipts

A static receipt consists of a text-only template since there are no variables that need to change. Once the template is created, it needs to change very little. Although it might be easier to set up static receipts, they tend to be impersonal and not as useful as dynamic receipts.

### Dynamic Receipts

A dynamic receipt consists of a mixture of text and variables. The text is the static part of the receipt, such as your business name, and the variables are the dynamic part, such as the amount of the transaction.

---

## Understanding Transaction Flows and Settlement

Settlement is the last step in the payment process, and is the step that allows you to be paid for your goods or services.

This section explains what it takes to settle a transaction using different processing models. For an overview discussion of the different processing models, see [Understanding Credit Card Processing Models, page 30](#). The most efficient way to use this section is to know what type of processor (and fulfillment, if applicable) you have, and simply read the sections that apply to your situation.

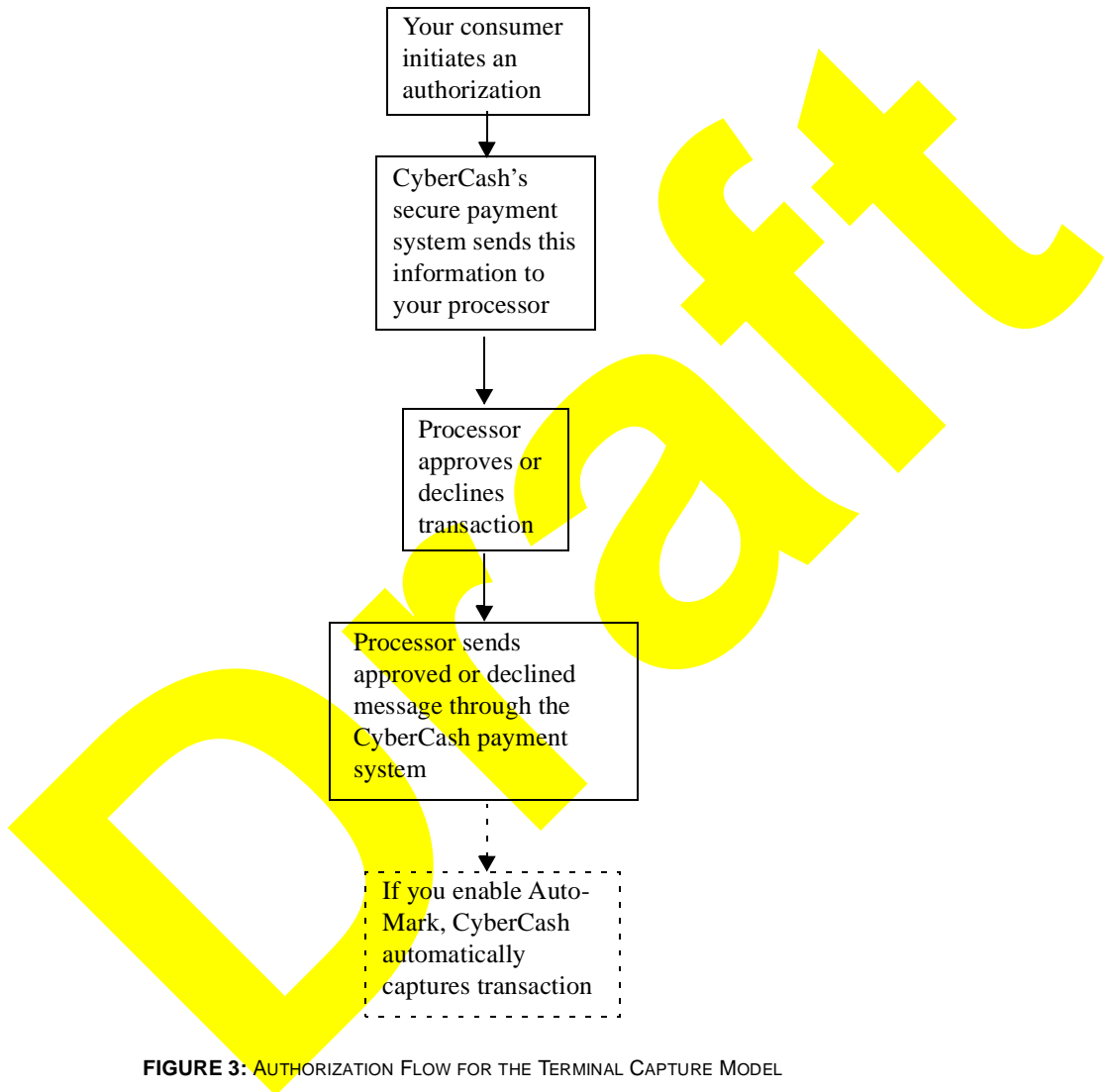
### Understanding the Terminal Capture Processing Flow

In the terminal capture processing environment, authorization for a credit card purchase is obtained. Then it is captured in a batch by marking the transaction. The process of marking transactions is also referred to as *assembling a batch*. Once a batch is assembled, it can be submitted to the processor for settlement.

In this model, the transactions are stored in a batch file at CyberCash. The processor settles the batched transactions when you submit the batch to the processor. If you use this processing model, you must get authorization for the purchase amount, capture of the authorization into the batch by marking the transaction, and submit the batch to the processor for settlement.

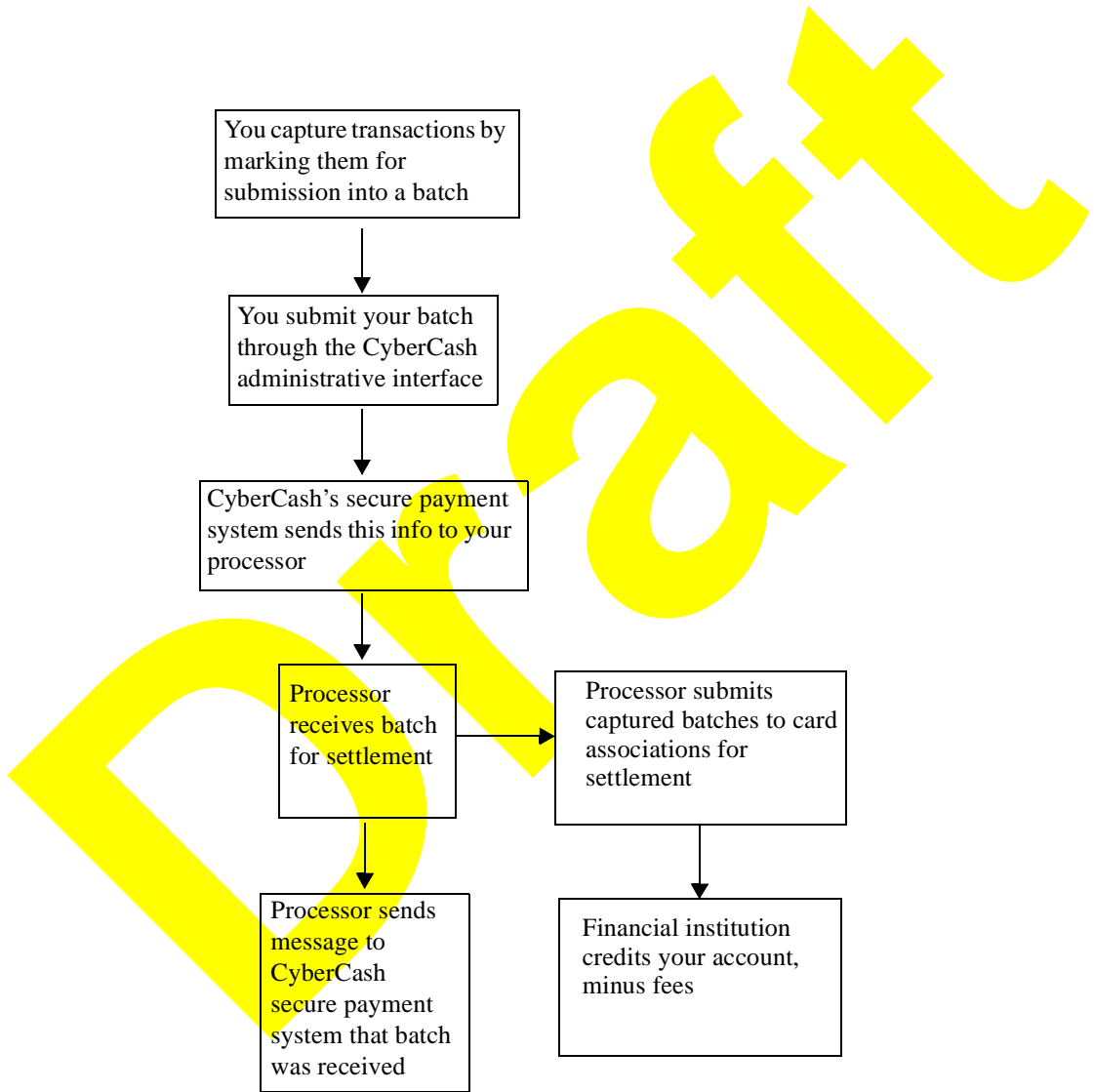
If you use terminal capture, your authorized transactions are stored at CyberCash until you capture them. In order to financially settle the transactions, you must capture the transactions into a batch, and then submit the batch for capture. You can configure the CashRegister to automatically mark transactions by specifying to Auto-Mark during registration. For more information about Auto-Mark, see [Using Auto-Mark and Auto-Settle, page 45](#).

[Figure 3, page 43](#), shows how a transaction is authorized using a terminal capture processor. This figure is meant to give you an overview of the flow; it omits background components and processes.



**FIGURE 3:** AUTHORIZATION FLOW FOR THE TERMINAL CAPTURE MODEL

Figure 4, page 44, shows how a transaction is captured and settled in a batch using a terminal capture processor. This figure is meant to be an overview of the flow; it omits background components and processes.



**FIGURE 4:** CAPTURE AND SETTLEMENT FLOW FOR A TERMINAL CAPTURE PROCESSOR

**Using Auto-Mark and Auto-Settle.** If you provide immediate fulfillment, you can enable the Auto-Mark feature during merchant registration. Auto-Marking automatically captures all transactions into batch files. Any transactions that you need to remove from the batch prior to submitting the batch for settlement will have to be manually unmarked using the administrative interface. This feature does not submit the batch for settlement. You must manually do this, set the Auto-Settle option, or use the API.

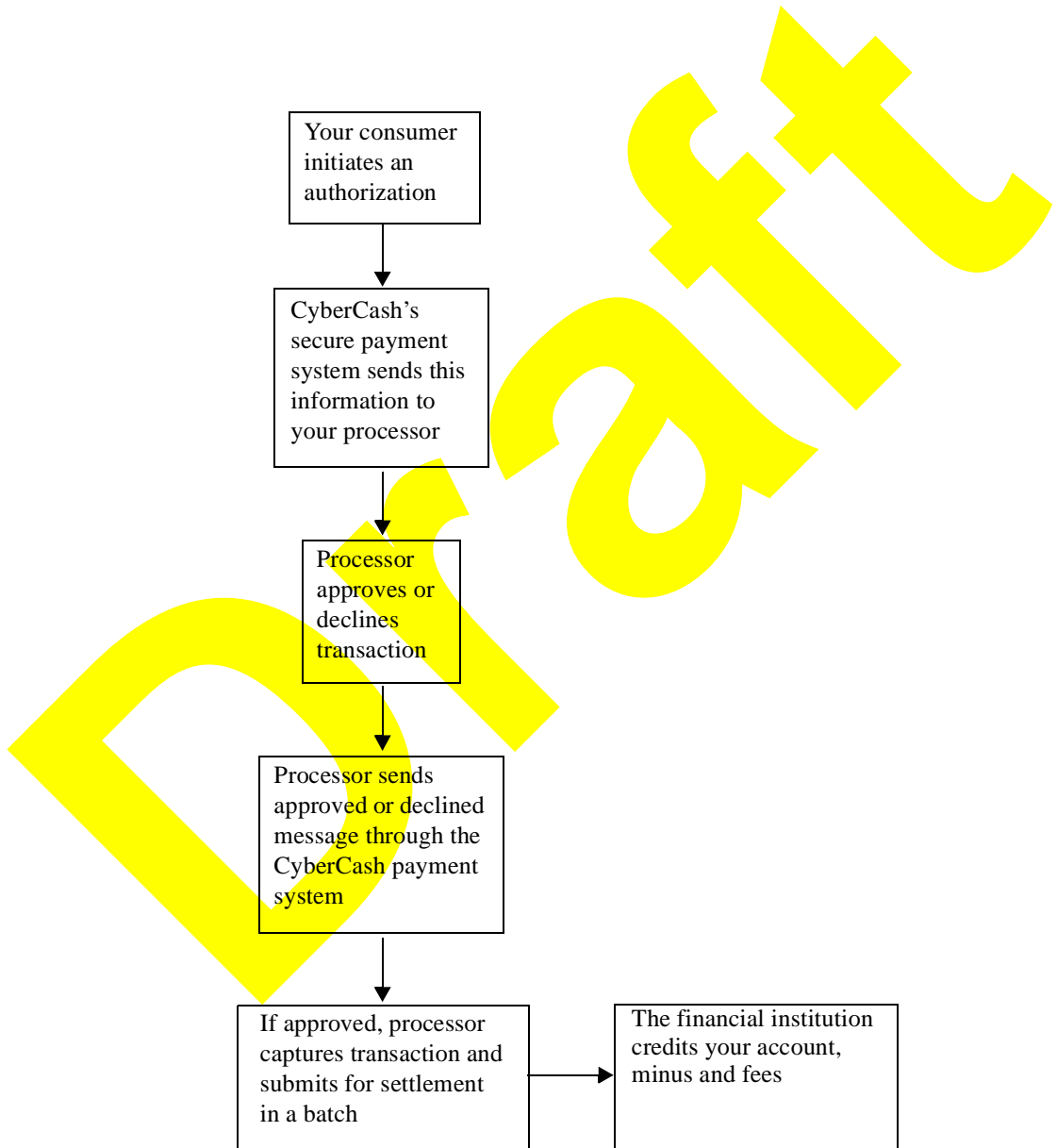
If you enable the Auto-Settle feature during merchant registration, your batches will be submitted for settlement once a day at a time determined by CyberCash. You can also manually submit batches for settlement during the day using the administrative interface. You do not need to use Auto-Mark to use Auto-Settle. If you do not use Auto-Mark, you need to capture your transactions into a batch manually using the administrative interface, or use the API.

Auto-Mark and Auto-Settle are independent features. You can choose to use one or both. Be aware of the tasks associated with using just one of these features. If you use both, capturing transactions into a batch and submitting it for settlement is a hands-off operation.

### Understanding the Host Capture, AuthCapture Processing Flow

If your financial institution has assigned you to a processor using this model, you probably offer a service, sell digital goods, or ship orders of physical goods within 24 hours of when they are placed (immediate fulfillment). In this model, the payment is captured at the same time the credit card amount is authorized. [Figure 5, page 46](#), shows the flow for a simultaneous authorization and capture. This figure is meant as an overview of the process; it omits details about background components and processes.

Under host capture, the processor's host computer stores the authorized transactions. With host capture (AuthCapture), the capture happens automatically at the end of the day without action by you. In this model, the transactions are stored in a batch file at the processor. The processor settles the batched transactions during certain times of the day, known as *processing windows*.



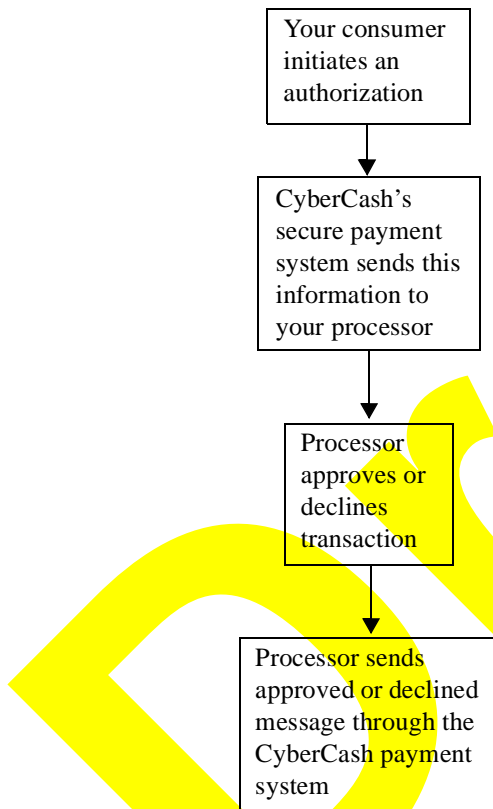
**FIGURE 5:** PROCESSING FLOW FOR THE HOST CAPTURE, AUTHCAPTURE MODEL

## Understanding the Host Capture, PostAuthCapture Processing Flow

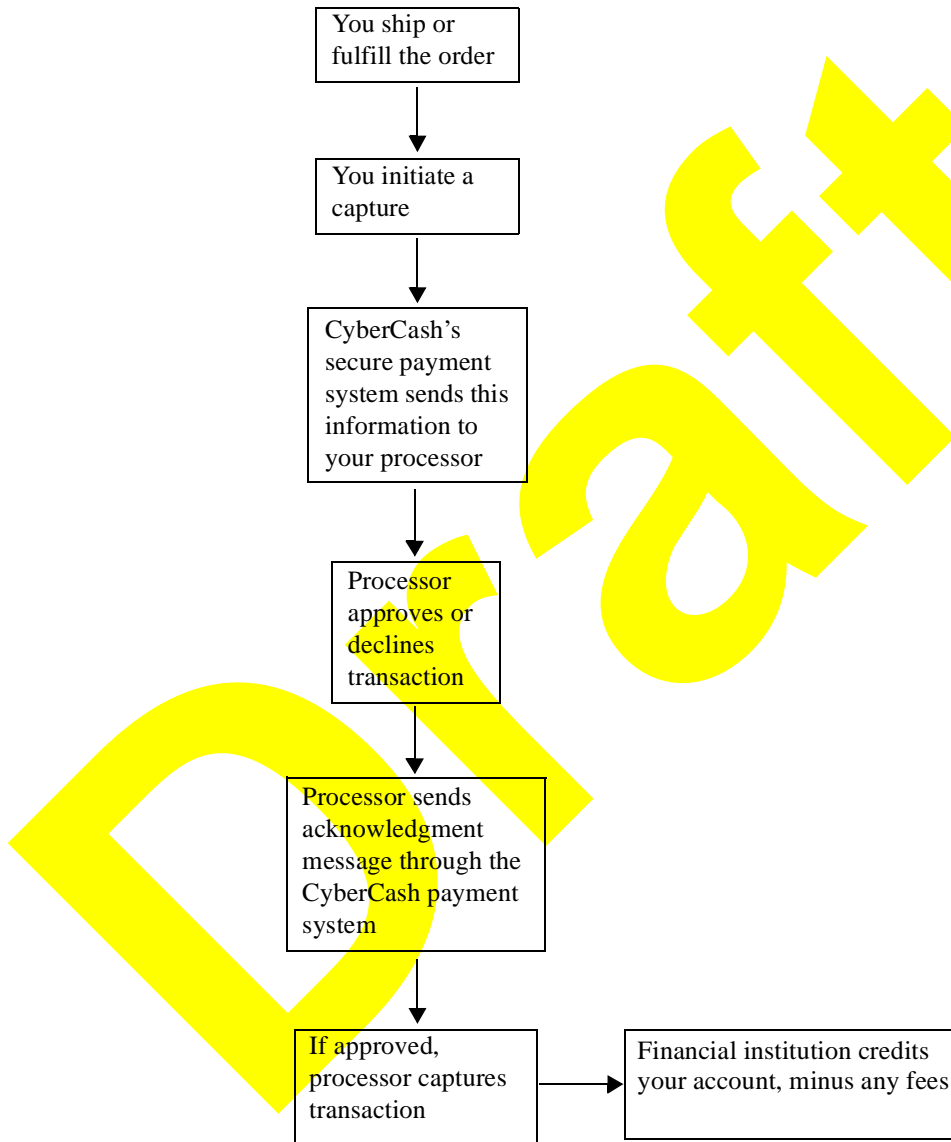
When you fulfill orders more than one day after receiving them, you must authorize and capture transactions separately. Authorization is performed at the time the consumer wants to make the purchase. Capture is performed when you ship the order.

Under host capture, the processor's host computer stores the authorized transactions. With host capture (PostAuthCapture), the capture occurs only after action you take. In this model, the transactions are stored in a batch file at the processor. The processor settles the batched transactions during certain times of the day, known as *processing windows*. If you use this processing model, you must get authorization for the purchase amount and subsequently capture the authorization when you are ready to ship the goods. The settlement of the transactions is done for you by the processor.

[Figure 6, page 48](#), shows the authorization flow. [Figure 7, page 49](#), shows the capture flow. This figure is meant as an overview of the process; it omits details about background components and processes.



**FIGURE 6:** AUTHORIZATION FLOW FOR THE HOST CAPTURE, POSTAUTHCAPTURE MODEL



**FIGURE 7:** CAPTURE FLOW FOR THE HOST CAPTURE, POSTAUTHCAPTURE MODEL

## Performing Settlement Tasks

The tasks you perform and the tasks your processor performs on your behalf vary by processing model. The checklist shown in [Table 9](#) identifies the functions that you need to perform depending on your model. Your tasks are indicated by checkmarks.

**TABLE 9: MERCHANT TASK CHECKLIST**

PROCESSING MODEL	AUTHORIZATION	CAPTURE	SETTLEMENT
Host Capture, AuthCapture	✓		
Host Capture, PostAuthCapture	✓	✓	
Terminal Capture	✓	✓	✓

There are three methods you can use to perform these tasks:

- using the CyberCash payment services administration interface
- using auto features
- using the Direct Connect API

[Table 10, page 51](#), describes these methods.

**TABLE 10: METHODS USED TO SETTLE TRANSACTIONS**

METHOD	DESCRIPTION
administration interface	Use the administrative interface to CyberCash payment services. Depending on your processor, the tasks you need to perform are displayed on the interface. For instructions, see <a href="#">CashRegister: Administration Guide</a> .
auto features	Use the Auto-Mark and/or Auto-Settle features. These features automatically capture transactions in a batch and submit batches for settlement. For details, see <a href="#">Using Auto-Mark and Auto-Settle</a> , page 45.
API	Use the Direct Connect API to perform your tasks. For example, you can use the API to authorize your payments without having to access the administration interface. For API details, see <a href="#">Using the CyberCash Payment Services API</a> , page 121.  You cannot use the API if you use Browser Connect.

## Understanding Pending Transactions and Timeouts

Purchase transactions (authorizations) normally are either accepted or declined. *Accepted* means that the consumer information is valid, and you can get paid through the normal settlement process. *Declined* means that the information provided is invalid, or that the consumer's account does is insufficient to complete the transaction.

Sometimes transactions go into a *pending* state. This happens under various circumstances. Most commonly, the Internet connection is lost between CyberCash and either you or the consumer, or you may not get a response from the CyberCash Payment Services within a time limit. Your application may continue processing, commonly on the assumption that the transaction will be declined. This event is called a timeout. In these cases, the real transaction may be accepted or declined and this result is recorded in with CyberCash Payment Services.

Sometimes, the CyberCash does not get a response from the processor, or the CyberCash may not get a response from the processor within its time limit. CyberCash Payment Services will then record that transaction is pending and return this state to you.

Your application should resolve pending transactions by sending a query to CyberCash Payment Services. If the transaction was accepted, you should either complete fulfillment or issue a credit. If transaction was declined, you should consider aborting fulfillment if possible. Regardless, the final result needs to be recorded in your financial records.

---

## Planning Your MCK Implementation

The MCK is very versatile. It is important to understand the concepts and options that you have before implementing the MCK. Once you have familiarized yourself with the concepts of I-commerce covered in this chapter, you can begin selecting the options you want to use.

In summary, the decisions that you need to make are:

- How you want to display your merchant offer?
- Which payment date collection methods you wish to support (SSL-based forms, Microsoft Wallet, and/or CyberCash Wallet)?
- Do you need to store the payment data?
- How do you want to fulfill your orders (in-house, third-party, online, and so on)?
- How do you want to display your receipts (static vs. dynamic)?
- Which settlement option you want to use and which one does your financial institution support?
- Do you want to automate the settlement or use a manual interface to do it?

Draft

Draft

---

**Part II:  
Using CyberCash  
Payment Services**

**Draft**



# Getting a Merchant Account

---

Now that you understand the concepts and issues involved with setting up a store on the Internet, it's time for you to get started. Remember, for the quickest implementation, you should work on several tasks simultaneously. Getting a merchant account is just one of these tasks. You also need to build your storefront, integrate it with the CashRegister, and register with CyberCash.

This chapter explains how to get a merchant account, along with the information you will need. For an in-depth discussion of merchant accounts, see *Setting Up a Merchant Account with a Financial Institution*, page 29. You can use that section to help determine what kind of setup is best for you.

---

## What Do You Do with Your Account Information?

Getting your merchant account is relatively easy. All you need to do is tell your financial institution that you want to open a merchant account that can accept credit card payments. Be prepared to discuss your business practices with your financial institution. For example, your financial institution will need to know what you are selling, and how and when your products or services are delivered. Based on your setup, your financial institution will help you determine what credit card processing model and credit card processor are best for you.

Once your merchant account is set up, your financial institution's processor will communicate your merchant account information to CyberCash. The processor will provide CyberCash your merchant account information so CyberCash can ensure the credit card service payments are handled properly.

Once the following steps are complete, your store should be ready to go live:

- Get a merchant account
- Register with CyberCash using the IMR
- Create your storefront with goods and/or services for sale
- Download the MCK (if you are a self hosted merchant)
- Integrate the MCK with your store
- Run test transactions
- Go live

# Using Integrated Merchant Registration (IMR)

---

This chapter describes how to use the online Integrated Merchant Registration (IMR) web site to register to become an officially recognized CyberCash merchant.

**NOTE:** While using the IMR, do not download the MCK if you are a merchant using software provided by a third party company that comes equipped to connect to CyberCash Payment Services. Simply use the IMR to submit the registration information, and then setup an account with a financial institution that accepts CyberCash Payment Services. Follow the instructions provided by the third party company.

If you do not know if you're are using third party software that is designed to connect with CyberCash Payment Services, check the Tech Partner listing on the CyberCash web site at [www.cybercash.com](http://www.cybercash.com).

---

## Understanding the Registration Process

The merchant registration process is outlined below:

- Establish a merchant account with a financial institution that lets you accept credit cards. You can ask your current bank to set up your account to process credit card payments through CyberCash. Or you can visit [www.cybercash.com](http://www.cybercash.com) to access a list of financial institutions that specialize in setting up merchants to use CyberCash services.
- After you open a merchant account, have your financial institution send your merchant ID, terminal ID, and other payment setup information directly to CyberCash. When CyberCash receives the information from your financial institution, CyberCash notifies you to go to the IMR web site to pick up your CyberCashID (CCID), hash secret or merchant key, and the host location assigned to your store.
- Go to the IMR web site, and submit your registration information.
- Download and install the MCK.
- Integrate the MCK with your storefront.
- Run test transactions.
- Open your web store for business (go live).

---

## Understanding the IMR

The IMR lets you register as a CyberCash merchant. When you go to the IMR web site, you must set your username and password. After you set your username and password, you can begin entering your registration information. The information that you supply helps CyberCash to provide first quality technical support to you.

If you are unclear about the information that you need to type into the fields, use the context-sensitive Help by clicking on the question mark icon located next to the fields. The red star icons denote fields that require you to enter information.

The information that the IMR requests is listed below:

- The legal name of your business, and the name that your store uses (if different than the legal name).
- Your address, telephone number, and contact person information.
- The store's URL.
- The name of the web integration company that you use, and the contact person.
- The name of the company that hosts your store, and the contact person.
- A brief description of the services provided by your web store.
- The credit card processor that your company uses.
- Your store configuration information, including the operating system type and version, the web server software, and the storefront software.
- The transaction settlement method that you use.
- The type of CyberCash Payment Service that you want to use (credit card or PayNow).

**NOTE:** The Originating Depository Financial Institution (ODFI) name, your financial institution (bank) name, routing/transit number, and account number are also requested if you want to use PayNow service.

It is possible for you to host and integrate your own store. Enter your own information if you host and/or integrate your store.

After you submit your information, the IMR brings you to a page that lets you download the version of the MCK that you need for your store.

Upon notification by your financial institution, CyberCash will notify you to retrieve your CyberCash ID (CCID), secret keys, and other parameters. You will need to enter these data items into your merchant registration.

## Downloading, Installing, and Configuring the MCK

The MCK contains software and example scripts that enable your web server to connect to the CyberCash Payment Services (see [Installing the MCK](#), page 65, for more information). These scripts enable communication among your web server, your customer, and CyberCash.

There are several versions of the MCK available for downloading onto your computer. You must select the version that matches your computer setup.

After you download the MCK, you must install it and configure it. As you go through the installation, you will be prompted for information such as your CCID, secret, and host location. The MCK installation script automatically sets up a configuration file based on the information that you enter.

When you complete MCK installation and configuration, you must integrate your storefront. For detailed information about installing and configuring the MCK, see *Installing the MCK*, page 65, and *Customizing the MCK*, page 87.

---

## Accessing the IMR

If you are accessing the IMR for the first time, you must set a username and password, and then enter your registration information. It is important that you remember your username and password for future use. They are used to access your IMR record.

To set your username and password:

1. Go to the IMR web site at the following location:  
<http://amps.cybercash.com>
2. Click **get a username and password**.  
The **Register as a New User** page is displayed.
3. Type your email address and password; then click **Register**.  
The **Succeeded** page is displayed.

**NOTE:** Your username is the email address that you entered.

4. Click **log in** to begin the registration process.  
The **Username and Password Required** dialog is displayed.
5. Type your email address in the **Username** field, and type your password in the **Password** field.
6. Click **OK**.  
The IMR **Welcome** page is displayed. You can now begin entering your registration information.

---

## Updating Your IMR Information

Updating your IMR information requires your username and password that you set when you first accessed the IMR.

To update your IMR information:

1. Go to the IMR web site at the following location:  
`http://amps.cybercash.com`
2. Click **log in**.  
The **Username and Password Required** dialog is displayed.
3. Type your email address in the **Username** field, and type your password in the **Password** field.
4. Click **OK**.

The IMR **Welcome** page is displayed. You can now begin entering your registration information.

---

## Understanding the IMR API

The IMR Application Program Interface (API) allows submission of more than one item under a programming interface. This draft does not contain information about using IMR API. This information will be available in the final release of this document.

Draft

# Installing the MCK

---

You can install the Merchant Connection Kit (MCK) to support a single merchant or multiple merchants. The type of installation that you choose depends on how you want to use the MCK.

This chapter covers the following topics:

- understanding the MCK
- installing the MCK to support multiple merchants
- understanding common files, the development environment, and data structure

**NOTE:** You do not need the MCK if you are using third party software that is designed to seamlessly integrate your web store with CyberCash Payment Services.

---

## Understanding the MCK

The MCK is a software toolkit provided by CyberCash to assist you in connecting your storefront to CyberCash Payment Services. The MCK includes a collection of scripts, HTML template files, software libraries, and configuration files, which you can customize to accommodate your storefront integration.

You need to register as a CyberCash merchant to download the MCK. To register as a merchant, access the Integrated Merchant Registration (IMR) web site (see *Using Integrated Merchant Registration (IMR)*, page 59) at <http://amps.cybercash.com>.

After you register and download the MCK, you must need to install, configure, and integrate the MCK with your storefront. For more information, see *Customizing the MCK*, page 87.

---

## Understanding MCK Installation

You can install the MCK in two different modes:

- stand-alone
- shared-server

The way you choose to install the MCK depends on your needs and the environment in which you are working.

A stand-alone installation is ideal if you are a single merchant, or if you are a developer who needs to take the MCK apart and integrate it into your own application, as in the case of a Tech Partner or MDP.

A shared-server installation is ideal if you are hosting multiple merchants on a shared server. The shared-server installation option optimizes the MCK setup for a shared-server environment by allowing you to complete the following tasks:

- specify a common directory for MCK shared files (see [Understanding Common Files](#), page 70)
- select supported language(s)
- specify interpreter/compiler location
- customize the default merchant templates and scripts
- specify the location of the **build-merchant** script

---

## Installing the MCK in Stand-Alone Mode

To install the MCK in stand-alone mode:

1. Run the installation program without using any parameters. For example:  

```
./install-mck-3.2-solaris-sparc
```

Once the MCK is installed, you have the option of setting up a merchant with real configuration information, or you can set up a merchant using test configuration information.

### Setting Up a Merchant in Stand-Alone Mode

After you install the MCK, you need to configure the MCK with the configuration variables that you received from CyberCash when you registered as a merchant. Those configuration variables should include your CyberCash ID (CCID), hash secret or merchant key, and host location. See [Using Integrated Merchant Registration \(IMR\)](#), page 59, for more information.

If you have your configuration variables, you can run the **express** or **customized configure** MCK command. Express configuration uses default values, only prompting you for the CyberCash configuration variables. Customize configuration prompts you for all configuration variables.

If you have not yet received your configuration variables, you can run the **test configure** MCK command. Test configuration installs test variables that let you complete test transactions. To process real transactions, you must replace the test configuration variables with the configuration variables assigned to your store by CyberCash.

- To setup a merchant in stand-alone express mode, type the following command:  
`./configure`
- To setup a merchant in stand-alone customized mode, type the following command:  
`./configure -c`
- To setup a merchant in stand-alone test mode, type the following command:  
`./configure -T`

---

## Installing the MCK in Shared-Server Mode

To install the MCK in shared-server mode:

1. Run the installation program using the **-mdp** parameter. For example:  
`./install-mck-3.2-solaris-sparc -mdp`

Once the MCK is installed on the shared server, you customize the default templates and scripts available in the **master-merchant** directory to fit your needs. For example, you might want to modify the receipt templates to display a certain logo, or modify templates to contain certain information, static or dynamic. All new merchants set up using the MCK will use the customized files in the **master-merchant** directory.

Once the MCK is installed, you can also begin adding multiple merchants to your server. See [Setting Up a Merchant in Shared-Server Mode, page 70](#), for more information

Before the MCK is stalled in your server, you must accept the license agreement. [Figure 8, page 69](#), shows shared-server installation script.

```

Extracting distribution data ... done.
Configuring the Merchant Connection Kit
Version mck-3.2.r3-solaris-sparc
>>>MDP Shared-Server Installation<<<

The Merchant Connection Kit supports two types of scripts to integrate with
a storefront and/or fulfillment center.
You can choose the language(s) that you want to allow your merchants to
use.
Perl - Requires Perl minimum version 5.003
C - Requires an ANSI compatible C compiler
Both - Requires both language compilers above.

What type of CGI scripts would you like to support? (P/C/B) [Perl]: P
Checking Perl 5.003...

Installation using '/usr/local/bin/perl5'.
As an MDP, you have the option of allowing your merchants to install their
own MCK or you can build it for them yourself. If you which to allow
merchants to install their own MCK, the build-merchant script needs to be
copied to a directory that is in the execution path of all users. This
script will give merchants the ability to setup their own configurations,
and optionally install the customized configurations to their WEB server
directories.

Do you wish to copy the build-merchant script to a public location? (y/n):

Where do you want to install the build-merchant script? [/usr/local/bin]:

The Merchant Connection Kit Perl scripts reference libraries and utilities
to perform common functions. These files must be installed in a direct that
is accessible to the MCK CGI scripts.

Where do you want to install the Perl utilities? [/home/mck/mck-3.2.r3-
solaris-sparc/perl-api]

Modifying shared-server utilities ...
You do not have sufficient privileges to write to the following
directories: /usr/local/bin
Please supply a user id and password for a UNIX user that can write to the
above directories. Username: Password:

Perl libraries and utilities already installed.
Installing build-merchant script to /user/local/bin

MDP Shared-Server configuration complete.

```

FIGURE 8: SHARED-SERVER INSTALLATION SCRIPT

## Understanding Common Files

When installing the shared-server mode, the installation program prompts you for a common directory for shared files. The MCK shares these files among all supported merchants.

The following files are the shared files for Perl:

- CCMckDirectLib3\_2.pm
- CCMckErrno3\_2.pm
- CCMckLib3\_2.pm
- MCKdecrypt
- MCKencrypt
- computeMD5hash

The following files are the shared files for C:

- CCBlock.c
- CCBlock.h
- CCConfig.c
- CCConfig.h
- CCErrno.c
- CCErrno.h
- CCHttpSocket.c
- CCHttpSocket.h
- CCMckDirect.c
- CCMckDirect.h
- CCMckDirectPayment.c
- CCMckDirectPayment.h
- CCMckLib.c
- CCMckLib.h
- CCNVlist.c
- CCNVlist.h
- CCSocket.c
- CCSocket.h
- CCString.c
- CCString.h
- base64.h
- mckcrypt.h
- myassert.h
- lib/libCCMck.a
- lib/libmd5hash.a

## Setting Up a Merchant in Shared-Server Mode

To set up a merchant, you use the **build-merchant** command each time you want to add a merchant on your server.

If you have the CyberCash configuration variables for the merchant that you want to add to your server, you can run the express or customized **build-merchant** command. The express **build-merchant** command uses default values, only prompting you for the CyberCash configuration variables. The customize **build-merchant** command prompts you for all configuration variables.

If you have not yet received the configuration variables for the store that you want to add to your server, you can run the test **build-merchant** command. The test **build-merchant** command installs test variables that let the merchant complete test transactions. To process real transactions, you must replace the test variables with the configuration variables assigned to the store by CyberCash.

- To add a merchant to your server in express mode, type the following command:

```
Install/build-merchant
```

- To add a merchant to your server in customize mode, type the following command:

```
Install/build-merchant -c
```

- To setup a merchant in stand-alone test mode, type the following command:

```
Install/build-merchant -T
```

[Figure 9, page 72](#), shows the information that you must enter when you run the express **build-merchant** command.

```
What type of center would you like to support?

  1) Storefront with Fulfillment Center
  2) Storefront only
  3) Fulfillment Center only
Enter center type: [1]: 1

Enter your CyberCash ID:
Enter your Hash Secret:
Enter your Store Front Name:
Enter your Customer Service Phone Number:

Has CyberCash provided you a Merchant Key during registration?
(y/n) [n]:
Enter your Merchant Key:

Enter the full URL of your cgi-bin alias:

Enter the full URL of your documents alias:

Enter the full Unix path to your SECURE http server cgi-bin
directory]:

Enter the full Unix path to your SECURE http server documents
directory:
Enter the full Unix path to your UN-SECURE http server cgi-bin
directory:

Do you want to support Credit Card purchases using the Microsoft
wallet? (y/n):

Will you be selling Soft Goods (payloads)? (y/n):

Do you want to accept forwarded POPs? (y/n):

Enter the URL of the script that will accept the POP:

Would you like to install the configured files (y/n):
```

**FIGURE 9:** BUILD-MERCHANT EXPRESS COMMAND QUESTIONS

---

## Understanding the MCK Development Environment

The MCK development environment is very flexible. During installation, you are asked what programming language you want to use. Individual merchants can choose **C** or **PERL**. Merchant Development Partners (MDPs) can choose **C** or **PERL**, or both. **ASP/VB** scripts available for NT users. You use the programming language of your choice to customize the scripts that the MCK provides.

---

## MCK System Requirements

To use the MCK and CyberCash payment services, you need the following items:

- a connection to the Internet
- a UNIX or NT operating system
- PERL 5.003 or later **OR** a C compiler
- an SSL-compliant web server with a registered, valid certificate from an established certificate authority
- an SSL-compliant web browser

### Operating Systems

This section lists specific operating systems and requirements.

#### UNIX:

- Solaris (Sparc)
- FreeBSD
- RedHat Linux
- IRIX
- HPUX
- BSDI
- Solaris (Intel)
- Digital UNIX
- SCO
- AIX (PowerPC)

System administrator rights.

Minimum of 10 MB of free disk space.

**Windows NT 4.0:**

System administrator rights.

Minimum of 10 MB of free disk space.

## Programming Languages

This section lists the programming languages you can use under each platform.

**UNIX:**

- C—requires ANSI C compiler such as VC++, GNU G++, G++, or GCC
- PERL—must be PERL 5.003 or later, available at [www.activestate.com](http://www.activestate.com) or [www.perl.com](http://www.perl.com)

**NT:**

- C—requires ANSI C compiler such as VC++, GNU G++, G++, or GCC
- PERL—must be PERL 5.003 or later, available at [www.activestate.com](http://www.activestate.com) or [www.perl.com](http://www.perl.com)
- ASP—requires Microsoft Internet Information Server

## Understanding the MCK Data Structure

This section lists the directories and files that are downloaded into your **mck.cgi** directory when you install the MCK on your computer.

**NOTE:** For a description of the directory files, read the README files. The information in the README is relative to the directory in which the README is located.

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
CHANGES Install	DIST		
	cc-setup config.in fix-perms.skell install.sh shell-funcs		cc-setup fix-perm.skell install.sh perl-setup
LICENSE README			

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
c-api	CCBlock.c CCBlock.h CCConfig.c CCConfig.h CCErrno.c CCErrno.h CCHttpSocket. CCHttpSocket.h CCMckDirect.c CCMckDirect.h CCMckDirectPayment .c CCMckDirectPayment .h CCMckLib.c CCMckLib.h CCNVlist.c CCNVlist.h CCSocket.c CCSocket.h CCString.c CCString.h Makefile README base64.h lib mckcrypt.h myassert.h		libCCMck.a libmckcrypto.a libmd5hash.a
configure configure.log configure.log.hist master-merchant	README README.c-cgi README.mswallet README.perl		

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
	c-scripts		
		CCMerchantCustom	
		.c	
		CCMerchantCustom	
		.h	
		CCMerchantTest.c	
		CCMerchantTest.h	
		README	
		browser	
			CCmsw.c
			CCmsw.h
			Makefile
			README
			cardin.c
			ccwalletpay.c
			checkin.c
			directpaycheck
			.c
			directpaycredit
			.c
			fulfillment.c
			mwalletpay.c
			notification.c
			sslformpay.c
			testdriver.c
		cr21api	
			CCArgs.h
			CCLib.c
			CCLib.h
			Makefile
			README
			example-auth.c
			example-capture.c
		direct	
			CCArgs.h
			Makefile
			README
			example-auth.c
			example-capture.c
			example-query.c

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
tools			Makefile README get-
notification.c	conf	README merchant_conf mime.types	
	html	README cyberlogo.gif empty.wlt safefail.html test-mck.html	
	payload	README sample.html	
	perl-scripts	CCMerchantCustom.pm CCMerchantTest.pm README browser	README cardin.cgi ccwalletpay.cgi checkin.cgi directpaycheck. cgi directpaycredit .cgi fulfillment.cgi msw.pl mswalletpay.cgi notification. cgi sslformpay.cgi testdriver.cgi
		cr2api	CCLib.pm README example-auth example-capture

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
direct			README example-auth example-capture
		tools	README get-notification.pl
	templates		
			README autoPayment.tem cardPayment.tem cardsale.tem ccwPayment.tem ccwalletsale.tem ccwallet.tem checksale.tem customFailureResponse.tem customReceipt.tem customReceipt.tem customRedirectResponse.tem directchecksale.tem directcreditsale.tem failFulfillment.tem manualPayment.tem mswPayment.tem mswalletsale.tem mswjs.tem scriptError.tem sslformsale.tem tempDifficulties.tem thanks.tem
mcktest-20			
			README README.c-cgi README.mswallet README.perl c-scripts
			CCMerchantCustom.c CCMerchantCustom.h CCMerchantTest.c CCMerchantTest.h README

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
		browser	CCmsw.c CCmsw.h Makefile README cardin.c ccwalletpay.c checkin.c
directpaycheck.c			
directpaycredit.c			
directpaycredit.c			fulfillment.c mswalletpay.c notification.c sslformpay.c testdriver.c
		cr21api	CCArgs.h CClib.c CClib.h Makefile README example-auth.c example-
capture.c		direct	CCArgs.h Makefile README example-auth.c example- capture.c example-query.c
		tools	Makefile README get- notification.c
	conf		README merchant_conf mime.type

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
config.sh			
		README cyberlogo.gif empty.wlt safefail.html test-mck.html	
	html	README sample.html	
	payload	README sample.html	
	perl-scripts	CCMerchantCustom.pm CCMerchanTest.pm README browser	
			README cardin.cgi ccwalletpay.cgi checkin.cgi directpaycheck .cgi directpaycredit .cgi fulfillment.cgi msw.pl mswalletpay.cgi notification .cgi sslformpay.cgi testdriver.cgi
		cr21api	CCLib.pm README example-auth example-capture
		direct	README example-auth example-capture

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
		tools	README get- notification .pl
	templates	README autoPayment.tem cardPayment.tem cardsale.tem ccwPayment.tem ccwalletsale.tem ccwapplet.tem checksale.tem checksale.tem customFailure Response.tem customReceipt.tem customRedirect Response.tem directchecksale.tem directcreditsale .tem failFulfillment.tem manualPayment.tem mswPayment.tem mswalletsale.tem mswjs.tem scriptError.tem sslformsale.tem tempDifficulties .tem thanks.tem	
perl-api	CCMckDirectLib3_2 .pm CCMckDirectLib3_2 .pm.bak CCLib3_2.pm CCLib3_2.pm.bak DIST	CCMckDirectLib3 _2.pm CCMckLib3_2.pm	

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
test-mck	MCKdecrypt MCKencrypt computeMD5hash mcktest		
	README README.c-cgi README.mswallet README.perl c-scripts	CCMerchantCustom .c CCMerchantCustom .h CCMerchantTest.c CCMerchantTest.h README browser	CCmsw.c CCmsw.h Makefile README cardin.c ccwalletpay.c checkin.c directpaycheck.c directpaycredit.c fulfillment.c mswalletpay.c notification.c sslformpay.c testdriver.c
		cr21api	CCArgs.h CCLib.c CCLib.h Makefile README example-auth example-capture



MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
		direct	
capture.c			CCArgs.h Makefile README example-auth.c example- example-query.c
		tools	Makefile README get- notification.c
	conf		README merchant_conf mime.types
	config.sh		README cyberlogo.gif empty.wlt safefail.html test-mck.html
	html		README sample.html
	payload		CCMerchantCustom.pm CCMerchantTest.pm README browser
	perl-scripts		README cardin.cgi ccwalletpay.cgi checkin.cgi directpaycheck .cgi directpaycredit .cgi fulfillment.cgi msw.pl mswalletpay.cgi notification .cgi sslformpay.cgi testdriver.cgi

MCK.CGI Base Directories & Files	Directories & Files	Sub-Directories & Files	Files
		cr21api	CCLib.pm README example-auth example-capture
		direct	README example-auth example-capture
		tools	README get-notification.pl
	templates		README autoPayment.tem cardPayment.tem cardsale.tem ccwPayment.tem ccwalletsale.tem ccwapplet.tem checksale.tem customFailureResponse.tem customReceipt.tem customRedirectResponse.tem directchecksale.tem directcreditsale.tem failFulfillment.tem manualPayment.tem mswPayment.tem mswalletsale.tem mswjs.tem scriptError.tem sslformsale.tem tempDifficulties.tem thanks.tem



Draft

# Customizing the MCK

---

This chapter describes different ways to build CyberCash payment capabilities for your electronic storefront using the Merchant Connection Kit (MCK). At this point, you should understand how commerce is transacted on the Internet and how you want to plan your implementation. Before integrating your storefront with CyberCash Payment Services, you need to make sure that you understand the business requirements outlined below, and that your development environment is within the parameters described in *Installing the MCK*, page 65. You also need to get a merchant account with a financial institution (see *Getting a Merchant Account*, page 57) and register with CyberCash using the Integrated Merchant Registration (see *Using Integrated Merchant Registration (IMR)*, page 59).

---

## Before You Begin

Before you begin your integration, review *Planning Your MCK Implementation*, page 52, to make sure you know all of your store's requirements.

You will also need your CyberCash ID (CCID), secret keys, and other parameters that you received as part of your registration with CyberCash. For more information about registration, see *Using Integrated Merchant Registration (IMR)*, page 59.

Review the system requirements and install the MCK as described in *Installing the MCK*, page 65. You should know what programming environment you want to use to customize the MCK. If you are using UNIX, you can choose between PERL and C. If you are using Windows NT, you can choose to integrate your storefront using Active Server Pages (ASP), PERL, or C.

---

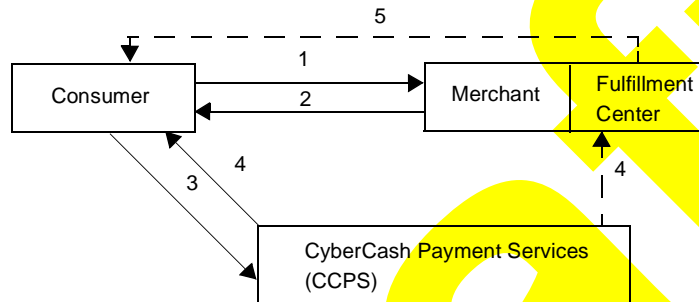
## Understanding Browser Connect and Direct Connect

The MCK offers two ways to interface your web store with CyberCash Payment Services: Browser Connect, and Direct Connect. Browser Connect supports SSL forms, the CyberCash Agile Wallet, and the Microsoft Wallet.

Direct Connect lets you use the CyberCash Application Programming Interface(API) to write programs in C and/or PERL for purchase authorizations, settlements, queries, and reports, letting you integrate payment processing into any I-Commerce application. Direct Connect lets you execute server-to-server commands (from your server to the CyberCash server), which means that you have the capability to send purchases and administrative transactions directly to the CyberCash Payment Services without browser interference.

### Understanding Browser Connect Transactions

Figure 10, page 89, shows how transactions are processed using the Browser Connect method. Browser Connect is recommended for merchants who are using the Agile Wallet or Microsoft Wallet. In the Browser Connect model, all communication is secured using SSL.



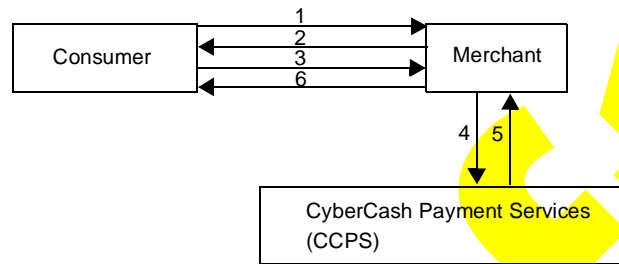
**FIGURE 10:** BROWSER CONNECT TRANSACTIONS

1. The consumer clicks Pay on your web page, which contacts your web store.
2. You send an offer/confirmation back to the consumer.
3. The consumer accepts the offer, and the transaction is sent to CCPS, where it is approved or denied.
4. An approval or denial message is sent to the consumer. The message is also sent simultaneously to your fulfillment center, if you have one. (For more information about fulfillment centers, see [Understanding Fulfillment, page 38](#). If you do not have a fulfillment center, you can check the status of your purchases by accessing your administrative interface.
5. The fulfillment center downloads the purchase to the consumer if the purchase is an electronic good. If the purchase is a hard good, the fulfillment center sends a receipt to the consumer.

### Understanding Direct Connect Transactions

Figure 11, page 90, shows how transactions are processed using the Direct Connect method. Direct Connect is recommended for advanced merchants with a high volume of transactions who want to make to produce customized reports programming modifications to the MCK.

In the Direct Connect model, communication between you and the consumer is secured using SSL. Communication between you and CCPS uses triple DES encryption.



**FIGURE 11:** DIRECT CONNECT TRANSACTIONS

1. The consumer clicks Pay on your web page, which contacts your web store.
2. You send an offer/confirmation back to the consumer.
3. The consumer accepts the offer, and the transaction is sent back to your web store.
4. Your web store send the transaction to CCPS, where it is approved or denied.
5. CCPS sends an approval or denial back to your web store.
6. Your web store sends the message to the consumer.

### Understanding Browser Connect and Direct Connect Scripts

There are two classes of CGI scripts that CyberCash provides for you to use in your storefront integration: Browser Connect and Direct Connect. These scripts are examples of how to interface a web site with CyberCash Payment Services, and should be viewed as starting points in your development of the CGI scripts used in your storefront design. All these scripts except **notification**.\* CGI generate HTML output.

The following scripts are used for Browser Connect:

- `sslformpay.*`
- `ccwalletpay.*`
- `mswalletpay.*`
- `cardin.*`
- `checkin.*`
- `fulfillment.*`
- `notification.*`

The following scripts are used for Direct Connect:

- `directpaycredit.*`
- `directpaycheck.*`

If you want the consumer to enter credit card payment information into an HTML form, CyberCash provides the **sslformpay.\*** script as an example of how this is implemented. The advantage of using the payment collection method in this script is that you do not have to worry about collecting or storing credit card data. If you want to collect additional information besides credit card data you can customize this script and modify the payment page appropriately.

If you prefer to collect the consumer's credit card payment data, CyberCash provides the **directpaycredit.\*** script as an example of how this is implemented. This method is preferable when your consumer is previously known, or if you are collecting a consumer's payment information at only one place in your storefront.

If you want to support the CyberCash Wallet as a payment option, CyberCash provides the **ccwalletpay.\*** script as an example of how this is implemented. This script also demonstrates how to collect a consumer's credit card information if the consumer does not have a CyberCash Wallet.

If you want to support the Microsoft Wallet as a payment option, CyberCash provides the **mswalletpay.\*** script as an example of how this is implemented. This script also demonstrates how to collect a consumer's credit card information if the consumer does not have an Microsoft Wallet.

If you are collecting check payment information, CyberCash provides the **directpaycheck.\*** script as an example of how this is implemented. With this method, you must collect the check payment information by enhancing this CGI or by designing your own to work with custom HTML forms.

The scripts **cardin.\*** and **checkin.\*** are provided for backward compatibility with existing MCK storefront integration solutions.

The scripts **fulfillment.\*** and **notification.\*** are scripts provided for customizing fulfillment and notification of your consumer's order.

---

## Customizing Your MCK Scripts

The MCK uses several CGI scripts (**cardin.\***, **ccwalletpay.\***, **checkin.\***, **directpaycheck.\***, **directpaycredit.\***, **mwalletpay.\***, **sslformpay.\***, **notification.\*** and **fulfillment.\***) to perform different methods of electronic payment. These scripts are used to communicate consumer information from your storefront to CyberCash, from CyberCash to you (through your consumers), from CyberCash directly to your consumers, and in specific instances, from CyberCash to your fulfillment center. This section outlines a description of these scripts, and the points of customization within these scripts.

!!! CyberCash recommends that you make (backup) copies of your source scripts before modifying them.

The MCK builds the scripts described below using information that you have provided during the MCK configuration process. The scripts are commented in the areas where you can add your own custom code. CyberCash recommends that you modify the way your scripts collect payment information from your consumers. These scripts encapsulate information about the merchant offer, the consumer order, and the consumer payment information, and send that information securely to CyberCash. The messaging uses name-value pair variables; you are expected to modify some of these variables, but several of these variables should not be altered in the scripts. A description of these variables and information on whether they should be modified or not are included in [Name-Value Pair Variables, page 193](#). The scripts also include messages handled by the consumer's web browser, which are fully customizable.

## Customizing sslformpay.\*

The **sslformpay.\*** script performs the following tasks:

- initializes certain MCK configuration parameters
- references the templates modified and assigned by you that collect payment information and notify the consumer of transaction failures
- retrieves the merchant offer price information from the **CCMerchantCustom.\*** file (this is likely to have been modified by you)
- retrieves the identifier for this order (offer ID)
- generates a payment page for the consumer to complete the transaction

**NOTE:** The page allows the consumer to capture his or her credit card information. This page is either automatically submitted to the CashRegister to process the transaction (if JavaScript is supported), or it is manually submitted

In this script, the following points of customization are suggested (in order of their placement in the script):

- If you want CyberCash system default response templates to be integrated into your payment page, you should assign the local variable **useCustomResponses** to **0**.

For PERL:

```
$useCustomResponses = 0;
```

**NOTE:** This scenario is unlikely.

- You can also collect other information on the payment page (to pass to the fulfillment center). To do this you should assign **mo.signed.cpi** the value **yes**, and change the payment template to **manualPayment.tem**. You must also customize **manualPayment.tem** to collect the additional information required.

For PERL:

```
$paymentNVList {'mo.signed-cpi'} = "yes";
$paymentTemplate = $templateDir . " manualPayment.tem";
```

**NOTE:** For security reasons, you should never read **mo.signed.cpi** from an HTML form.

- You must decide how you want the CashRegister to communicate with the consumer after the order is processed. You may generate a page that thanks the consumer (possibly with additional information on the page), or you may choose to generate a page that redirects the consumer to another web page/site. With a redirection page, it is possible to send the consumer to a fulfillment center. In this scenario, the HTML form will contain a POP and additional data from the Merchant Fulfillment (MF) fields of the message. The redirect URL is usually set up in your merchant configuration when you configure your MCK; however, in some specific circumstances you might want to define this in the script. For example, you may want to redirect particular consumers (frequent shoppers, for instance) to another web site. Assigning the URL value to **mo.redirect-url** does this.

For PERL:

```
$paymentNVList{'mo.redirect-url'} = "your URL";
```

- In either case, you can customize the HTML success and failures pages, or you can use the CyberCash default pages. If you want to customize your pages, you must make sure that the local variable **useCustomResponses** is set to **0**. You should then change these custom templates: **customReceipt.tem**, **customRedirectResponse.tem**, **customFailureResponse.tem**, and **tempDifficulties.tem**.

- Depending on your choices, the script may collect additional information on the payment page. Since you already have the order data, you might want to perform some verification on the order data before the payment page is generated. For example, this may involve database lookups (to check order price).

- At this point you can choose to add additional data to the message; this will be sent as is to the fulfillment center. To do this, you must assign **mf.your-field-here** your desired value. You can sign this data, but you must know how to read the signature at the fulfillment center.

For PERL:

```
$paymentNVList {'mf.your-field-here'} = "some value";
```

- You can also change your payment page URL by replacing the token **#PAYMENT\_URL#** in your payment page template. You can also add to the name-value pairs (some of which are described earlier) defined by adding to the token **#MANUAL\_ARGS#** in your payment page template.

## Customizing mswalletpay.\*

The **mswalletpay.\*** script performs the following tasks:

- initializes certain MCK configuration parameters
- references templates modified and assigned by you that collect payment information and notify the consumer of transaction failures
- retrieves the merchant offer price information from the **CCMerchantCustom.\*** file (this is likely to have been modified by you)
- retrieves the identifier for this order (offer ID)
- generates a payment page for the consumer to complete the transaction with MicroSoft Wallet. If the consumer does not have the Microsoft Wallet, the payment page allows the consumer to enter credit card information. This page is automatically submitted to CyberCash Payment Services to process the transaction.

In this script, the following points of customization are suggested (in order of their placement in the script):

- If you want CyberCash system default response templates to be integrated into your payment page, then you should assign the local variable **useCustomResponses** to **0**.

For PERL:

```
$useCustomResponses = 0;
```

**NOTE:** This scenario is unlikely.

- You must decide how you want CyberCash Payment Services to communicate with the consumer after the order is processed. You can generate a page that thanks the consumer (with possibly additional information on the page), or you can choose to generate a page that redirects the consumer to another web page/site. With a redirection page, it is possible to send the consumer to a fulfillment center. In this scenario, the HTML form will contain a POP and additional data from the MF fields of the message. The redirect URL is usually set up in your merchant configuration when you configure your MCK; however, you can assign this in the script directly for specific circumstances. Assigning to the URL value **mo.redirect-url** does this.

For PERL:

```
$paymentNVList{'mo.redirect-url'} = "your URL";
```

- In either case, you can customize the HTML success and failures pages, or you can use the CyberCash default pages. If you want to customize your pages, you must make sure that the local variable **useCustomResponses** is set to **0**. You should then change these custom templates: **customReceipt.tem**, **customRedirectResponse.tem**, **customFailureResponse.tem**, and **tempDifficulties.tem**.
- Since you already have the order data, you might want to perform some verification on the order data before the payment page is generated. For example, this may involve database lookups (to check order price).
- You can add additional data to the message. This will be sent as is to the fulfillment center. To do this, you must assign **mf.your-field-here** your desired value. You may sign this data, but you must know how to read the signature at the fulfillment center.

For PERL:

```
$paymentNVList {'mf.your-field-here'} = "some value";
```

- You can also change your payment page URL by replacing the token **#PAYMENT\_URL#** in your payment page template. You can also add to the name-value pairs (some of which are described earlier) defined by adding to the token **#MANUAL\_ARGS#** in your payment page template.

You should also perform your bookkeeping tasks after the payment page is processed. This might involve saving order and consumer data.

## Customizing ccwalletpay.\*

The **ccwalletpay.\*** script performs the following tasks:

- initializes certain MCK configuration parameters
- references templates modified and assigned by you that collect payment information and notify the consumer of transaction failures
- retrieves the merchant offer price information from the **CCMerchantCustom.\*** file (this is likely to have been modified by you)
- retrieves the identifier for this order (offer ID)
- generates a payment page for the consumer to complete the transaction with CyberCash Wallet

**NOTE:** If the consumer does not have the CyberCash Wallet, the payment page allows the consumer to enter credit card information. This page is either automatically submitted to the CashRegister to process the transaction.

In this script, the following points of customization are suggested (in order of their placement in the script):

- If you want CyberCash system default response templates to be integrated into your payment page, then you should assign the local variable **useCustomResponses** to **0**.

For PERL:

```
$useCustomResponses = 0;
```

**NOTE:** This scenario is unlikely.

- You must decide how you want CyberCash Payment Services to communicate with the consumer after the order is processed. You can generate a page that thanks the consumer (with possibly additional information on the page), or you can choose to generate a page that redirects the consumer to another web page/site. With a redirection page, it is possible to send the consumer to a fulfillment center. In this scenario, the HTML form will contain a POP and additional data from the MF fields of the message. The redirect URL is usually set up in your merchant configuration when you configure your MCK; however, you can assign this in the script directly for specific circumstances. Assigning the URL value to **mo.redirect-url** does this.

For PERL:

```
$paymentNVList { 'mo.redirect-url' } = "your URL";
```

- In either case, you can customize the HTML success and failures pages, or you can use the CyberCash default pages. If you want to customize your pages, you must make sure that the local variable **useCustomResponses** is set to **0**. You should then change these custom templates: **customReceipt.tem**, **customRedirectResponse.tem**, **customFailureResponse.tem**, and **tempDifficulties.tem**.
- Since you already have the order data, you might want to perform some verification on the order data before the payment page is generated. For example, this may involve database lookups (to check order price).
- At this point you can choose to add additional data to the message; this will be sent as is to the fulfillment center. To do this, you must assign **mf.your-field-here** your desired value. You can sign this data, but you must know how to read the signature at the fulfillment center.

For PERL:

```
$paymentNVList { 'mf.your-field-here' } = "some value";
```

- You can also change your redirect page URL by replacing the token **# REDIRECT\_URL#** in your payment page template. You may also add to the name-value pairs (some of which are described earlier) defined by adding to the token **#MANUAL\_ARGS#** in your payment page template. You can also replace the token **#CCWJS#** (which is the name of the Java script used when processing a return from the CyberCash Wallet server), and the token **#CCWCODEBASE#** (which represents the URL used to download the CyberCash Wallet applet).

You should also perform your bookkeeping tasks after the payment page is processed. This might involve saving order and consumer data.

## Customizing directpaycheck.\*

The **directpaycheck.\*** script performs the following tasks:

- initializes certain MCK configuration parameters
- references templates modified and assigned by you that collect payment information and notify the consumer of transaction failures
- retrieves the merchant offer price information from the **CCMerchantCustom.\*** file (this is likely to have been modified by you)
- retrieves the identifier for this order (offer ID)
- retrieves the consumer's checking account information from an HTML form, or from another source (no payment page is generated for this script)
- sends out a payload if it is configured to do so

In this script, the following points of customization are suggested (in order of their placement in the script):

- This script requires that you must customize your templates. These include: **customReceipt.tem**, **customRedirectResponse.tem**, **customFailureResponse.tem**, and **tempDifficulties.tem**.
- You must decide how you want CyberCash Payment Services to communicate with the consumer after the order is processed. You can generate a page that thanks the consumer (with possibly additional information on the page), or you may choose to generate a page that redirects the consumer to another web page/site. With a redirection page, it is possible to send the consumer to a fulfillment center. In this scenario, the HTML form will contain a POP and additional data from the MF fields of the message. The redirect URL is usually set up in your merchant configuration when you configure your MCK; however, in some specific circumstances you might want to define this in the script. Assigning the URL value to **mo.redirect-url** does this.

For PERL:

```
$paymentNVList{'mo.redirect-url'} = "your URL";
```

- You must provide the consumer's checking account and order information. The checking account information should be collected from an HTML form (or from a database if the consumer is known to you). If you use a form, you should use the `#MANUAL_ARGS#` token to build your name-value pairs list, and define at least the following arguments: **`cpi.check-account-number`**, **`cpi.check-bank-routing-number`**, **`cpi.check-name`**, **`cpi.check-use`**, and **`cpi.account-paid`**. If you are retrieving the information from some other place, you should incorporate that into the script, and define the above name-value pairs.
- You should verify the order information and checking account information before the payment message is signed.
- At this point you can choose to add additional data to the message. This will be sent as is to the fulfillment center. To do this, you must assign **`mf.your-field-here`** your desired value. You may sign this data, but you must know how to read the signature at the fulfillment center.

For PERL:

```
$paymentNVList {'mf.your-field-here'} = "some value";
```

You must customize the fulfillment of the consumer's order in this script. If you are redirecting the consumer to another web page/site, you must do so in the script. If not, you must generate your own receipt method, and dispense a payload (if a payload is configured). Do not rely on the default logging built into the script for order tracking. Instead it is suggested that you track sales using more sophisticated methods.

In PERL, a sample might be (this is just an adaptation of the stub in the script):

```

if ($POP {'pop.status'} !~ /^success/)
{
    # failed...respond using the customFailureResponse page
    &PrintTemplate ($failTemplate, $TemplateFailRef, %tokenList);
    trackOrder (...) # function you must implement
}
elseif ($redirectURL)
{
    trackOrder (...) # function you must implement
    # Use customRedirectResponse.tem to redirect browser to
    fulfillment site...
    &PrintTemplate($successTemplate, $TemplateFailRef, %tokenList);
}
else
{
    trackOrder (...) # function you must implement
    # Here we generate a receipt or directly dispense a payload.
    # everything checks out, so check for payload
    if ($POP{'pop.payload'})
    {
        $filename = $POP{'pop.payload'};
        $pStatus = &SendPayload($filename);
        if ($pStatus ne "")
        {
            $tokenList{'MESSAGE'} = "Unable to open payload
$filename";
            $tokenList{'ORDERID'} = $orderId;
            &PrintTemplate ($problemTemplate, $TemplateFailRef,
                %tokenList);
            die ("$0: Payload failed\n.");
        }
    }
    if ($logStatus eq "failure")
    {
        $tokenList{'MESSAGE'} = $logMessage;
        $tokenList{'ORDERID'} = $orderId;
        &PrintTemplate ($problemTemplate, $TemplateFailRef,
            %tokenList);
    }
    else
    {
        &PrintTemplate ($problemTemplate, $TemplateFailRef,
            %tokenList);
    }
}
}

```

## Understanding `directpaycredit.*`

The `directpaycredit.*` script performs the following tasks:

- initializes certain MCK configuration parameters
- references templates modified and assigned by you that collect payment information and notify the consumer of transaction failures
- retrieves the merchant offer price information from the `CCMerchantCustom.*` file (this is likely to have been modified by you)
- retrieves the identifier for this order (offer ID)
- retrieves the consumer's credit card information from an HTML form, or from another source (no payment page is generated for this script)
- sends out a payload if it is configured to do so

In this script, the following points of customization are suggested (in order of their placement in the script):

- This script requires that you must customize your templates. These include: `customReceipt.tem`, `customRedirectResponse.tem`, `customFailureResponse.tem`, and `tempDifficulties.tem`.
- You must decide how you want CyberCash Payment Services to communicate with the consumer after the order is processed. You can generate a page that thanks the consumer (with possibly additional information on the page), or you can choose to generate a page that redirects the consumer to another web page/site. With a redirection page, it is possible to send the consumer to a fulfillment center. In this scenario, the HTML form will contain a POP and additional data from the MF fields of the message. The redirect URL is usually set up in your merchant configuration when you configure your MCK; however, in some specific circumstances you might want to define this in the script. Assigning to `mo.redirect-url` the URL value does this.

For PERL:

```
$paymentNVList{'mo.redirect-url'} = "your URL";
```

- You must provide the consumer's credit card and order information. The credit card information should be collected from an HTML form (or from a database if the consumer is known to you). If you use a form, you should use the `#MANUAL_ARGS#` token to build your name-value pairs list, and define at least the following arguments: **`cpi.card-number`**, **`cpi.card-exp`**, **`cpi.card-name`**, **`cpi.card-address`**, **`cpi.card-city`**, **`cpi.card-state`**, **`cpi.card-zip`**, and **`cpi.card-country`**. If you are retrieving the information from some other place, you should incorporate that into the script, and define the above name-value pairs.
- You should verify the order information and checking account information before the payment message is signed.
- You can choose to add additional data to the message. This will be sent as is to the fulfillment center. To do this, you must assign **`mf.your-field-here`** your desired value. You may sign this data, but you must know how to read the signature at the fulfillment center.

For PERL:

```
$paymentNVList {'mf.your-field-here'} = "some value";
```

You must customize the fulfillment of the consumer's order in this script. If you are redirecting the consumer to another web page/site, you must do so in the script. If not, you must generate your own receipt method, and dispense a payload (if a payload is configured). You must not rely on the default logging built into the script for order tracking. Instead it is suggested that you track sales using more sophisticated methods.

In PERL, a sample might be (this is just an adaptation of the stub in the script):

```

if ($POP {'pop.status'} !~ /^success/)
{
    # failed...respond using the customFailureResponse page
    &PrintTemplate ($failTemplate, $TemplateFailRef,
    %tokenList);
    trackOrder (...) # function you must implement
}
elseif ($redirectURL)
{
    trackOrder (...) # function you must implement
    # Use customRedirectResponse.tem to redirect browser to
    fulfillment site...
    &PrintTemplate($successTemplate, $TemplateFailRef,
    %tokenList);
}
else
{
    trackOrder (...) # function you must implement
    # Here we generate a receipt or directly dispense a payload.
    # everything checks out, so check for payload
    if ($POP{'pop.payload'})
    {
        $filename = $POP{'pop.payload'};
        $pStatus = &SendPayload($filename);
        if ($pStatus ne "")
        {
            $tokenList{'MESSAGE'} = "Unable to open payload
            $filename";
            $tokenList{'ORDERID'} = $orderId;
            &PrintTemplate ($problemTemplate, $TemplateFailRef,
            %tokenList);
            die ("$0: Payload failed\n");
        }
    }
    if ($logStatus eq "failure")
    {
        $tokenList{'MESSAGE'} = $logMessage;
        $tokenList{'ORDERID'} = $orderId;
        &PrintTemplate ($problemTemplate, $TemplateFailRef,
        %tokenList);
    }
}
else
{
    &PrintTemplate ($problemTemplate, $TemplateFailRef,
    %tokenList);
}
}
}

```

## Customizing fulfillment.\*

The **fulfillment.\*** script performs the following tasks:

- initializes certain MCK configuration parameters
- references templates modified and assigned by you that notify the consumer of fulfillment success and failure
- retrieves the order ID from a order transaction log or from another mechanism you implement (if this script is run at the fulfillment center, this step may not be necessary)
- sends out a payload if it is configured to do so
- keeps a receipt of the transaction and other bookkeeping aspects

In this script, the following points of customization are suggested (in order of their placement in the script):

- You can customize the fulfillment failure and success HTML templates. This is done by assigning the local variables **problemTemplate** and **thankYouTemplate** to your desired values.

In PERL:

```
$problemTemplate = $templateDir."myProblemFile.tem";
$thankYouTemplate = $templateDir."myThankYouFile.tem";
```

- If you sent fulfillment data in the payment script, you must retrieve it here and process it appropriately. If it was signed, you must unsign it.
- You should check in this script (especially if you as the merchant are doing the fulfillment) for the validity of the order ID received before you complete fulfillment. This might involve checking an orders database.
- If a payload is configured for this order, you should attempt to process and deliver it. In addition to modifying the script, you might want to modify **SendPayload ()** in **CCMerchantCustom.\***.
- You should enhance the basic receipt and logging features in this script by adding your own. This is especially critical when attempting to track orders. You must not rely on the logging features in this script to track your orders.

## Customizing notification.\*

The **notification.\*** script performs the following tasks:

- initializes some MCK configuration parameters
- retrieves the order ID from a order transaction log, or from another mechanism you customize (if this script is run at the fulfillment center, this step may not be necessary)
- stores the fulfillment information in a notification log, or to a database if you customize it do so

In this script, the following points of customization are suggested (in order of their placement in the script):

- You should check in this script (especially if you as the merchant are doing the fulfillment) for the validity of the order ID received before you complete fulfillment. This might involve checking an orders database.
- You should enhance the basic notification feature in this script by adding your own. This can be done by modifying this file, or by modifying the routine **LogNotification ()** in **CCMerchantTest.\***.

## Customizing the CCMerchant\*. \* and CCMerchantCustom.\* Modules

These modules contain subprograms and parameters used in all the CGI scripts (for example, **sslformpay.\***) that are invoked from the customer's HTML forms. The subprograms or parameters in these modules are not referenced directly from the customer's HTML page.

**CCMerchantTest.\*** contains subprograms and parameters that you are expected to modify to support your storefront implementation.

**CCMerchantCustom.\*** contains subprograms and parameters that you may wish to modify to support your storefront. It is important that you modify **CCMerchantTest.\***.

CCMerchantTest.\* contains the following subprograms:

- **templateFailure**

This subprogram generates a default HTML template that is used if there is an error in parsing a custom template or a default CyberCash template. This subprogram should be modified if you want this page to be different.

- **SaveOrderInfo**

This subprogram accepts order information as its input parameters, and writes that order information in a formatted manner to a flat ASCII file. This is a very simple way to track orders, and CyberCash strongly recommends that you modify this. Though the flat file order database is easy to implement, it is very inefficient and slow for a large and busy merchant storefront. You might also be interested in tracking other parameters of the order besides those currently saved in the flat file. One way to modify this subprogram would be to implement a real-time relational database management system (RDMS) solution.

- **GetOrderInfo**

This subprogram retrieves information about a particular order by using an order ID key. The CGI scripts call this routine at the points at which fulfillment is handled. Currently this routine does a linear search on the flat file produced by **SaveOrderInfo**, and its performance degrades considerably after a large number of orders are processed. If you modify **SaveOrderInfo** to support a different implementation of an order transaction database, then you must modify this subprogram accordingly.

- **CheckOrderId**

This subprogram verifies that a particular order ID exists. The CGI scripts call this routine at the points at which notification or fulfillment is handled. Unlike **GetOrderInfo**, this routine only validates that an order ID exists. Again, this subprogram should be modified if you modify **SaveOrderInfo**.

- **LogNotification**

This subprogram writes the POP from a successful order to a log file. This is a very simple implementation, and you should modify this routine if you want to handle notifications in a more complex way.

- **ExtractNotification**

This subprogram retrieves the POP for a particular order ID. If you modify **LogNotification**, you must modify this routine appropriately.

**CCMerchantCustom.\*** contains the following subprograms:

- **GenerateOrderId**

This subprogram generates unique and valid order IDs. You can generate any order ID you like as long as the ID is unique, has 32 characters or less, and does not contain the characters ':', '<', '>', '=', '+', '@', '"', '%', '=', and '&'. This routine generates unique order IDs based on time, date, and process ID variables. You may customize this routine if you want to replace the default algorithm, or want to enhance this routine.

- **LoadPaymentInfo**

This subprogram collects order-related information, and builds the vector of payment name-value variables. This routine retrieves payment information by using **GetOrderInfo**; if you modify **GetOrderInfo** you may have to modify this routine. You can also modify this routine if you want to alter the order price (for example, provide a discount), or any of the other merchant order (mo) fields. This is also a good place to validate the order payment information since it is usually invoked at the CGI scripts before the signed order message is built.

- **SetResponseCustomization**

This subprogram modifies the location and names of the custom templates and template directories. This can be done within the CGI scripts, but you may want to localize template/directory name changes in one file.

- **ValidatePrice**

This subprogram validates the syntax and currency format of a price. You can customize this if your storefront handles multiple currencies.

- **SendPayload**

This subprogram provides a payload file for fulfillment. You can customize this routine if your payload is more complicated than a file in the payload directory.

**NOTE:** You can find the syntax, input parameters, and output parameters of these subprograms by examining **CCMerchantCustom.\*** and **CCMerchantTest.\***. You can also view their typical usage by checking the payment CGI scripts and the **testdriver.\*** CGI scripts.

---

## Customizing Settlements and Returns

Credit card payments can be authorized at when the consumer agrees to the purchase, and settled later when the goods are delivered. After a purchase is settled, a merchant may issue a credit because the goods are returned or for other reasons. This section describes how to process settlements and returns using the MCK.

If your payment processor uses the host capture processing method, the processor settles all transactions automatically. If you don't always fulfill orders within 24 hours of purchase, you should normally be assigned to a different mode by the processor.

If your payment processor uses the terminal capture processing method, or if you have chosen batch settlement, you have several different options for settlement. Using the IMR, you can specify that you want to perform auto-mark and/or auto-settlement of your authorized transactions.

The auto-mark feature directs CyberCash to capture all your authorized transactions in a batch.

The auto-settle feature directs CyberCash to submit all your batches for settlement in a timely manner.

You can emulate a host capture processor by specifying both options. CyberCash will automatically mark all your transactions, combine them in batches, and send the batches to your processor for settlement.

If you want to handle marking transactions yourself, you can specify just the auto-settle option during the IMR process. You will then have to capture the transactions into a batch yourself. If you specify the auto-settle option, CyberCash will attempt to settle your batches in 24 hours. You might want to use this option if the orders you process are not fulfilled immediately, or if the order price is not the same as the authorization amount.

If you want to handle settlement yourself, you can specify just the auto-mark option during the IMR process. Cybercash will capture all your authorized transactions into a batch automatically, but you must submit the batch for settlement yourself. You might want to use this configuration if your storefront offers goods that will be shipped after 24 hours.

If you don't specify either option, you will have to mark and request settlement of your transactions yourself. You should normally do this with goods that will be shipped after 24 hours, and when you are trying to integrate your storefront with fulfillment and settlement systems that are already in place.

You can capture transactions in a batch and settle it by developing PERL or C scripts that invoke the messages described in [Using the CyberCash Payment Services API, page 121](#). You will probably be most interested in the following messages: **batch-commit**, **batch-prep**, **batch-unroll**, **postauth**, **query**, **return**, and **void**. These messages allow you to access CyberCash Payment Services through the MCK.

Merchants can also process settlements and returns using the web-based administrative interface hosted by CyberCash. If your merchant application provides complete processing facilities for settlements and returns, you should advise your merchants to avoid using the administrative interface to eliminate inconsistencies between the CyberCash Payment Services and your application.

Draft

# Testing Your Storefront Integration

---

**Draft**

Draft

# Networking and Security

---

This chapter discusses networking and security issues in relation to the Merchant Connection Kit (MCK) and Integrated Merchant Registration (IMR). It also presents tips and concerns for multiple merchant environments. After reading this chapter, you should understand the following:

- how direct connect security works
- how browser connect security works
- how IMR security works
- how to securely store and separate merchant data

In the first three topics listed above, *security* refers to the connection between you and the CyberCash Payment System or IMR, and how your identity is authenticated.

---

## Understanding Direct Connect Security

If you use the direct connect method, the connection between you and the CyberCash Payment System is secured with triple DES encryption.

Messages encrypted by the MCK can only be decrypted by the CyberCash Payment System, and vice versa.

Using this encryption also ensures your authentication. Without your key, no one can pose as you to communicate with the CyberCash Payment System. This is why it is important to keep your key secure, and to never give it to anyone.

## Using HTTP Proxies

A *proxy* allows information to pass through a firewall to a server. An HTTP proxy does the same thing, transferring information using HTTP.

You can set up your system to receive information through an HTTP proxy. To do this, you must set two parameters in your merchant configuration file. [Table 11](#) explains these parameters.

**TABLE 11: PROXY PARAMETERS**

PARAMETER	DESCRIPTION
HTTP_PROXY_HOST	The hostname of the computer that is used as the HTTP proxy. In other words, the hostname of the computer that will receive information through the proxy.
HTTP_PROXY_PORT	The port number that the computer will use to receive information through the proxy.

You are **not** required to use a proxy. If you do not define proxy parameters, your communication will be sent through socket using a standard HTTP request.

---

## Understanding Browser Connect Security

If you use the browser connect method, your connection to the CyberCash Payment System is secured with an SSL link. This authenticates your identity to the CyberCash Payment System because you need a valid web server certificate to use SSL.

CyberCash recommends that you use SSL to communicate sensitive data if you are not using encryption. Most people use 40 bit SSL, although you can use 128 bit SSL in the United States. It is generally safe to use 40 bit SSL to transfer credit card information.

Your messages are protected with MD5 hashing, which both protects your message from tampering by the consumer and authenticates your identity. To use a hash, you have to have a hash secret and use the correct hash protocol. The CyberCash Payment System won't accept a message if your hash secret is incorrect. For this reason, it is important that you do not disclose your hash secret to anyone.

**NOTE:** The **notification.cgi** script is not called through SSL, nor does it use encryption. Normally, messages going to **notification.cgi** do not contain sensitive information. However, these messages do contain data derived from your Merchant Fulfillment (MF) data fields. Make sure you do not store any sensitive information in your MF data fields.

---

## Understanding IMR Security

When you use the IMR to register with CyberCash, or to change any registration information for your setup (for example, to register for another service), your information is protected by an SSL link.

The IMR authenticates your identity by requiring you to type your username and password when you use the system. Your username is your email address. You can choose any password. CyberCash does not verify that you are whom you say you are with your email address. However, no one can log on using your email address without your correct password.

Once you register, CyberCash sends assigns and sends you a secret. Do not share this secret with anyone. The security of your system could be compromised if you do.

CyberCash protects your registration data behind a firewall. It is available for you to access using the IMR.

---

## Storing Merchant Data

If you are storing data for multiple merchants on your server, read this section.

You might want to store all your merchant data in a single database, rather than setting up a database for each individual merchant. If you choose to do this, CyberCash recommends the following “good housekeeping” measures:

- Separate merchant data appropriately in the database. Do not mix merchant data.
- Give merchants access to their own information only. Do not allow merchants access to each other’s data.
- Make sure each merchant has a unique secret. Do not give more than one merchant the same secret.
- Limit access to log files. Do not allow everyone unlimited access to log files. Log files should be accessed on an individual, case-by-case basis.

These tips help to ensure the integrity of your system and your merchants’ data.

# Advanced Topics

---

Advanced topics are not available for this draft release. Advanced topics will be available in the production release of this document. Possible topics are listed below:

- using the CashRegister API (introduction)
- logging database support
- integrating with legacy systems (overview)
- using unsupported languages\*
- using unsupported platforms\*

**NOTE:** \*You can use [Using the CyberCash Payment Services API](#), page 121.

Please evaluate these topics to determine whether this information will be useful to you. Other topic suggestions are welcome. If you would like to suggest other topics, send an email message to [doc-suggest@cybercash.com](mailto:doc-suggest@cybercash.com). Use **MSG-Advanced** for your subject line.

Draft

---

## Appendixes

**Draft**



# Using the CyberCash Payment Services API

This appendix describes the API messages you can use to perform the functions available on the administrative interface to CyberCash Payment Services.

[Table 12](#) describes the messages you can use to interface with CyberCash Payment Services.

**TABLE 12:** CYBERCASH PAYMENT SERVICES MESSAGES

USE THIS MESSAGE	TO PERFORM THIS FUNCTION
batch-commit	Commit transactions assembled in a batch. For details about this message, see <a href="#">Using batch-commit, page 127</a> .
batch-prep	Query for transactions that are marked as ready to be sent to the processor in a batch. For details about this message, see <a href="#">Using batch-prep, page 132</a> .
batch-query	Query for a batch. For details about this message, see <a href="#">Using batch-query, page 140</a> .

TABLE 12: CYBERCASH PAYMENT SERVICES MESSAGES (CONT.)

USE THIS MESSAGE	TO PERFORM THIS FUNCTION
batch-unroll	<p>Query for transactions that were sent in a batch.</p> <p>For details about this message, see <i>Using batch-unroll</i>, page 145.</p>
card-query	<p>Request credit card data from the Gateway for a given order.</p> <p>For details about this message, see <i>Using card-query</i>, page 153.</p>
checkauth	<p>Validate and authorize a merchant-initiated check payment. This message is available only for the FirstUSA Paymentech processor (using the ECP option).</p> <p>For details about this message, see <i>Using checkauth</i>, page 156.</p>
checkreturn	<p>Return money to a consumer's checking account.</p> <p>For details about this message, see <i>Using checkreturn</i>, page 158.</p>
mauthcapture	<p>Authorize and capture a merchant-originated credit card sale. This message is for host capture processors only.</p> <p>For details about this message, see <i>Using mauthcapture</i>, page 160.</p>
mauthonly	<p>Authorize a merchant-originated credit card sale. This message is for terminal capture processors only.</p> <p>For details about this message, see <i>Using mauthonly</i>, page 162.</p>

**TABLE 12: CYBERCASH PAYMENT SERVICES MESSAGES (CONT.)**

USE THIS MESSAGE	TO PERFORM THIS FUNCTION
postauth	Capture a credit card payment previously authorized with mauthonly, or checkauth. For details about this message, see <i>Using postauth</i> , page 165.
query	Query the transactions database. For details about this message, see <i>Using query</i> , page 167.
retry	Retry a pending transaction for a given order. For details about this message, see <i>Using retry</i> , page 174.
return	Return money to the consumer's credit card. For details about this message, see <i>Using return</i> , page 177.
void	Void a transaction. For details about this message, see <i>Using void</i> , page 180.

### Understanding Standard Output Fields

Every message mentioned above returns three standard merchant output fields in addition to message-specific fields. [Table 13, page 124](#), describes these fields.

**TABLE 13:** STANDARD MERCHANT OUTPUT FIELDS

FIELD	DESCRIPTION
Mstatus	The returned status code of the command issued. This field is always present. It can assume the following values: success—successful transaction success-duplicate—result of a previously successful transaction partial success—batch has failed transactions failure-hard—transaction failed; subsequent retry will not help failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried failure-swversion—failure because of old or non-existent software (version) being used failure-bad-money—failure because of a credit problem with the financial institution
MErrLoc	The location of the error that occurred in the transaction. This field is present only if the transaction did not succeed (in other words, Mstatus was not "success"). Values for this field that may be returned are: smps—failure occurred at the payment server ccsp—failure occurred at the CyberCash Gateway financial institution—failure occurred at the payment processor
MErrMsg	The text message of the error that occurred in the transaction. This field is present only if the transaction did not succeed (in other words, Mstatus was not "success").

Some messages return standard Gateway output fields. These fields appear if the message was processed by the CyberCash Gateway. [Table 14, page 125](#), describes these messages.

**TABLE 14: STANDARD GATEWAY OUTPUT FIELDS**

FIELD	DESCRIPTION
merch-txn	The number that the Gateway uses to refer a transaction that was performed or attempted. This field is always present.
order-id	The order ID of the order whose transaction was just processed. This field is always present.
cust-txn	The transaction number generated by the CyberCash Wallet to refer to a just-processed transaction. This field is may not always be present.
aux-msg	The Gateway merchant message may contain verbiage from the Gateway or payment server. This field may be present only for successful transactions.
MSWErrMsg	The error message when an outdated version of the Wallet or the payment server is being used. This field may not always be present.
card-type	The type of card used in the transaction. This field may not always be present. Valid values for this field are: ot—other ax—American Express cb—Carte Blanche dc—Diners Club ds—Discover mc—MasterCard vs—Visa ecp-savings—Paymentech check service, savings account ecp-checking—Paymentech check service, checking account
card-number	The card number used in the transaction. This field may not always be present. If the transaction involved a Paymentech check (service), this number represents the MICR line of the check.

**TABLE 14:** STANDARD GATEWAY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-exp	The expiration of the card used in the transaction. This field may not always be present, and is of the format MM/YY (for example, "6/98" for June 1998).
auth-code	The authorization code returned from the payment processor. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
action-code	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often means that the transaction was authorized and captured. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
avs-code	The Address Verification System (AVS) code returned from the processor for the transaction. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
ref-code	The retrieval reference number for the transaction. This data is processor-specific and may not always be present.

**NOTE:** In addition to the input fields described for each message, there are other fields that are required for authenticating and securing a message sent to CyberCash Payment Services. These input fields do not change for the same merchant. There are functions available in the Merchant Connection Kit (MCK) that initialize these global configuration variables: `CCMckDirectSetup ()`, and `CCMckDirectShutdown ()` for C, and `InitConfig ()` for PERL. It is recommended that these functions be used.

## Using batch-commit

The batch-commit message is used to submit transactions assembled in a batch for settlement. [Table 15](#) describes the fields you should customize in the batch-commit message.

**TABLE 15:** BATCH-COMMIT INPUT FIELDS

FIELD	DESCRIPTION
num-txns	The number of transactions in the batch you are submitting. This field is required.
order-id	The order ID(s) for the transaction(s) you are submitting. This field is required.
txn-type	The transaction type(s) for the transaction(s) in the batch that you are submitting. Valid entries are "marked" and "markret." This field is required.
amount	The amount to capture of each transaction in the batch that you are submitting. Use the format currency dollar.cents (for example, usd 12.50). If this value is not present, the original authorization amount is used. This value may be equal to or less than the authorization amount. This field is optional.
lid-header	The line item detail header for the order(s) in the batch you are submitting. It consists of values concatenated together with back slashes used to separate the fields. For example, "header_value1\\header_value2\\header_value3" may be the lid-header value for order 1 of the batch. This field is optional.
lid-detail	The line item detail (LID) row for the order(s) in the batch you are submitting. For example, if order 1 has three LID rows, you should pass "lid-detail-1-1", "lid-detail-1-2", and "lid-detail-1-3". Each row consists of values concatenated together with backslashes used to separate the fields. This field is optional.

**NOTE:** You must differentiate the transactions in a batch. Assign a transaction number to each transaction and use it consistently in each field. This differentiates the transactions, and tells the system to which transaction the various field entries belong. CyberCash recommends that you specify this number by typing a dash and the number after the field entry. For example, you could assign the following for an order ID: 6787-ix where "ix" is the number you assigned to that transaction.

Then, in the txn-type field, you could assign the following: marked-ix where "ix" is the number you assigned to that transaction. CyberCash recommends this convention, but you can differentiate transactions in any way.

The following example shows how to use this messages in Perl.

```
$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$num-txns = 2;
$order-id1 = savedOrderId1;
$txn-type1 = 'marked';
$amount1   = 'usd 50.25'
$order-id2 = savedOrderId2;
$txn-type2 = 'markret';
$amount1   = 'usd 25.00'
# send the message to the CashRegister to
# commit this batch of 2 orders
$result = &SendCC2_lServer ('batch-commit',
                            'num-txns',    $num-txns,
                            'order-id-1',  $order-id1,
                            'txn-type-1',  $txn-type1,
                            'amount-1',    $amount1,
                            'order-id-2',  $order-id2,
                            'txn-type-2',  $txn-type2,
                            'amount-2',    $amount2);
# print all the name-value pairs returned by the message
for each (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the batch ID and status of the batch just submitted
print ("batch $batch-id status $result {'batch-status'}\n");
```

Table 16 describes the output fields that you should expect from the batch-commit message.

**TABLE 16:** BATCH-COMMIT OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These fields are listed in <a href="#">Table 13, page 124</a> .
batch-status	The batch settlement status as reported by the Gateway. This field is always present, and valid values returned: unknown—unknown batch-queued—batch is queued at Gateway to be submitted to payment processor batch-sent—batch was sent to payment processor, status pending batch-accepted—batch was accepted by processor (success) batch-rejected—batch was rejected by processor (failure) batch-error-at-cc—error occurred while processing the batch at CyberCash
batch-id	The batch ID of the batch. This field is always present.
gw-batch-id	The Gateway batch ID of the batch. This field is always present.
aux-msg	The Gateway merchant message may contain verbiage from the Gateway or payment server. This field may be present only for successful transactions.
action-code	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often implies that the transaction was authorized and captured. This field is not always present.
order-id-ix	The order ID of the transaction that was processed. This field is always present.

**TABLE 16:** BATCH-COMMIT OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
merch-txn-ix	The created ID for a transaction to be settled that was previously marked for capture or for return. This field is always present.
response-code-ix	The status of the transaction. This field is always present. It can assume the following values: success—successful transaction success-duplicate—result of a previously successful transaction partial success—batch has failed transactions failure-hard—transaction failed; subsequent retry will not help failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried failure-swversion—failure because of old or non-existent software (version) being used failure-bad-money—failure because of a credit problem with the financial institution
exception-message-ix	The exception message for the processed transaction. This field is not always present unless the transaction status is not successful (in other words, response-code-ix is not "success").
total-amount	The total currency amount for the whole batch. This field is always present, and is in the currency format (for example, usd 543.34).  <b>Note:</b> Sales are counted as positive amounts, while credits are counted as negative amounts.
sales-amount	The total sales amount for the whole batch. This field is always present, and is in the currency format.
sales-amount-amex	The total sales for American Express for the whole batch. This field is always present, and is in the currency format.

**TABLE 16:** BATCH-COMMIT OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
sales-amount-carte	The total sales for Carte Blanche for the whole batch. This field is always present, and is in the currency format.
sales-amount-diners	The total sales for Diners Club for the whole batch. This field is always present, and is in the currency format.
sales-amount-discover	The total sales for Discover for the whole batch. This field is always present, and is in the currency format.
sales-amount-ecp	The total sales for ECP (Paymentech) checks for the whole batch. This field is always present, and is in the currency format.
sales-amount-jcb	The total sales for Japanese Credit Bureau (JCB) for the whole batch. This field is always present, and is in the currency format.
sales-amount-mc	The total sales for MasterCard for the whole batch. This field is always present, and is in the currency format.
sales-amount-visa	The total sales for Visa for the whole batch. This field is always present, and is in the currency format.
sales-amount-others	The total sales for all other card types for the whole batch. This field is always present, and is in the currency format.
credit-amount	The total credit amount for the whole batch. This field is always present, and is in the currency format.
credit-amount-amex	The total credit for American Express for the whole batch. This field is always present, and is in the currency format.
credit-amount-carte	The total credit for Carte Blanche for the whole batch. This field is always present, and is in the currency format.
credit-amount-diners	The total credit for Diners Club for the whole batch. This field is always present, and is in the currency format.

**TABLE 16:** BATCH-COMMIT OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
credit-amount-discover	The total credit for Discover for the whole batch. This field is always present, and is in the currency format.
credit-amount-ecp	The total credit for ECP (Paymentech) checks for the whole batch. This field is always present, and is in the currency format.
credit-amount-jcb	The total credit for JCB for the whole batch. This field is always present, and is in the currency format.
credit-amount-mc	The total credit for MasterCard for the whole batch. This field is always present, and is in the currency format.
credit-amount-visa	The total credit for Visa for the whole batch. This field is always present, and is in the currency format.
credit-amount-others	The total credit for all other card types for the whole batch. This field is always present, and is in the currency format.

## Using batch-prep

The batch-prep message is used to query for transactions that are marked as ready to be sent to the processor in a batch. [Table 17, page 133](#), describes the fields you should use to customize the batch-prep message.

**TABLE 17:** BATCH-PREP INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for the transaction you are querying. Order IDs can contain letters, numbers, dashes, underscores, and periods. This field is optional.
txn-type	The transaction types for the transaction you are querying. This field is optional. Special values used only for querying are: anc—authorized not captured anm—authorized not marked msn—marked not settled rns—marked return, return not settled
txn-status	The status of the transaction you are querying. This field is optional. It can assume the following values: success—successful transaction. success-duplicate—result of a previously successful transaction partial success—batch has failed transactions failure-hard—transaction failed; subsequent retry will not help failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried failure-swversion—failure because of old or non-existent software (version) being used failure-bad-money—failure because of a credit problem with the financial institution
origin	The origin of the transaction you are querying. This field is optional and valid values are: c—consumer m—merchant blank—any (in other words not specified)

**TABLE 17:** BATCH-PREP INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-type	<p>The type of card used for the transaction you are querying. This field is optional and valid values are:</p> <p>ot—other  ax—American Express  cb—Carte Blanche  dc—Diners Club  ds—Discover  mc—MasterCard  vs—Visa  ecp-savings—Paymentech check service, savings account  ecp-checking—Paymentech check service, checking account</p>
start-time	<p>The earliest time that the transaction you are querying could have been created. Transactions will be returned that were started at this time or later. This field is optional.</p>
end-time	<p>The latest time that the transaction you are querying could have been created. Transactions will be returned that were created any time prior to this time. This field is optional.</p>
low-amount	<p>The lowest amount for a transaction you are querying. Transactions will be returned that are for this amount or higher. Use the format currency dollar.cents. (For example, usd 12.50). This field is optional.</p>
high-amount	<p>The highest amount for a transaction you are querying. Transactions will be returned that are for this amount or lower. Use the format currency dollar.cents. (For example, usd 12.50). This field is optional.</p>

**TABLE 17:** BATCH-PREP INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
maxcount	The highest number of transactions that you want returned from the query. This field is optional.
batch-id	The batch ID of the transaction(s) you are querying. This field is optional.

The following example show how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$txnType = 'anc';
$loAmount = 'usd 100.00'
# send the message to the CashRegister to query the
# transactions database for transactions that were authorized
# and not captured, and were greater than $100

$result = &SendCC2_1Server ('batch-prep',
                           'txn-type',    $txnType,
                           'low-amount',  $loAmount);
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}

```

[Table 18, page 136](#), describes the output fields that you should expect from the batch-prep message. The notation "-ix" represents one of many transactions that may be returned from the query. For example, if five transactions were returned for the query, then "order-id-1", "order-id-2", "order-id-3", "order-id-4", and "order-id-5" would be the five order IDs of these transactions.

**TABLE 18:** BATCH-PREP OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
order-id-ix	The order ID of the transaction. This field always appears.
merch-txn-ix	The number that the Gateway uses to refer a transaction that was performed or attempted. This field always appears.
cust-txn-ix	The transaction number generated by the CyberCash Wallet to refer to a just-processed transaction. This field is sometimes present.
origin-ix	The origin of the transaction. This field is optional and valid values are: c—consumer m—merchant blank—any (in other words not specified)

**TABLE 18:** BATCH-PREP OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
txn-type-ix	<p>The transaction types for the transaction. This field is optional. Expected values are:</p> <ul style="list-style-type: none"> <li>invoice—invoice consumer (Wallet)</li> <li>auth—credit card authorize (this applies to terminal capture payment processors only)</li> <li>postauth—credit card capture (this applies to terminal capture payment processors only)</li> <li>capture—credit card capture and authorize (applies to host capture payment processors)</li> <li>return—return money to credit card</li> <li>marked—mark capture for batching (batch processors only)</li> <li>markret—mark return for batching (batch processors only)</li> <li>settled—settle previously marked capture (batch</li> </ul>
txn-status-ix	<p>The status of the transaction. This field is always present. It can assume the following values:</p> <ul style="list-style-type: none"> <li>success—successful transaction</li> <li>success-duplicate—result of a previously successful transaction</li> <li>partial success—batch has failed transactions</li> <li>failure-hard—transaction failed; subsequent retry will not help</li> <li>failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried</li> <li>failure-swversion—failure because of old or non-existent software (version) being used</li> <li>failure-bad-money—failure because of a credit problem with the financial institution</li> </ul>
cust-id-ix	<p>The consumer (Wallet) ID. This ID identifies the Wallet to the Gateway. This field is sometimes present.</p>

**TABLE 18:** BATCH-PREP OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-type-ix	The type of card used for the transaction. This field is optional and valid values are: ot—other ax—American Express cb—Carte Blanche dc—Diners Club ds—"Discover mc—MasterCard vs—Visa ecp-savings—Paymentech check service, savings account ecp-checking—Paymentech check service, checking account
card-number-ix	The number of the credit card used for the transaction. This field is always present.
card-exp-ix	The expiration date of the credit card used for the transaction. This field is sometimes present, and has the format MM/YY (for example, 7/99 for July 1999)
amount-ix	The amount of the transaction. This field is always present.
time-ix	The time the transaction occurred. This field is always present.
auth-code-ix	The authorization code returned from the payment processor for the transaction. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
ref-code-ix	The retrieval reference number for the transaction. This data is processor-specific and may not always be present.

**TABLE 18:** BATCH-PREP OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
action-code-ix	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often implies that the transaction was authorized and captured. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
batch-id-ix	The batch ID of the transaction. This field is always present.
avs-code-ix	The Address Verification System (AVS) code returned from the processor for the transaction. This field may only be present if the transaction type (txn-type-ix) of the transaction is "batch" and is processor-dependent; consult your payment processor for details of these codes.
gw-batch-id-ix	The Gateway batch ID of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch".
batch-status-ix	<p>The batch status of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch". This field is optional, and valid values are:</p> <ul style="list-style-type: none"> <li>unknown—unknown</li> <li>batch-queued—batch is queued at Gateway to be submitted to payment processor</li> <li>batch-sent—batch was sent to payment processor, status pending</li> <li>batch-accepted—batch was accepted by processor (success)</li> <li>batch-rejected—batch was rejected by processor (failure)</li> <li>batch-error-at-cc—error occurred while processing the batch at CyberCash</li> </ul>

## Using batch-query

The batch-query message is used to update and reconcile the status of transactions committed in a given batch. This message is used to determine if a particular batch item has failed, and is essential for payment processors like VISA where the status of an item cannot be determined correctly from the output fields of a batch-commit message. [Table 19](#) describes the fields you should use to customize the batch-prep message.

**TABLE 19: BATCH-QUERY INPUT FIELDS**

FIELD	DESCRIPTION
batch-id	The batch ID of the batch; this would have been returned from a previous batch-commit message. This field is required.

The following example shows how to use this message in Perl.

```
$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely..
    die ("$0: unable to initialize configuration!\n");
}
$batchID = '4589';
# send the message to the CashRegister to reconcile
# and update the transactions database for batch ID 4589

%result = &SendCC2_lServer ('batch-query',
                           'batch-id, $batchID);
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the status of the query
print ("The batch query returned with a status $result
{'MStatus'}\n");
# print the total sales and credit amounts in this batch
print ("\tsales   = $result {'sales-amount'}");
print ("\tcredits = $result {'credit-amount'}\n");
```

[Table 20](#) describes the output fields that you should expect from the batch-query message. The notation "-ix" represents one of many transactions that may be returned from the query. For example, if five transactions were returned for the query, then "order-id-1", "order-id-2", "order-id-3", "order-id-4", and "order-id-5" would be the 5 order IDs of these transactions. These detailed line-item data are only returned for transactions that are reported by the Gateway as having failed.

**TABLE 20:** BATCH-QUERY OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
batch-id	The batch ID of the batch. This field is always sent.
order-id-ix	The order ID of the transaction that was processed. This field is only present for transactions that failed at the Gateway.
merch-txn-ix	The created ID for a transaction to be settled that was previously marked for capture or for return. This field is only present for transactions that failed at the Gateway.

**TABLE 20:** BATCH-QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
response-code-ix	<p>The status of the transaction. This field is only present for transactions that failed at the Gateway. It can assume the following values:</p> <p>success—successful transaction</p> <p>success-duplicate—result of a previously successful transaction</p> <p>partial success—batch has failed transactions</p> <p>failure-hard—transaction failed; subsequent retry ill not help</p> <p>failure-q-or-cancel", "failure-q-or-discard—transaction failed due to a communications failure; this may be retried</p> <p>failure-swversion—failure because of old or non-existent software (version) being used</p> <p>failure-bad-money—failure because of a credit</p>
exception-message-ix	The exception message for the processed transaction. This field is only present for transactions that failed at the Gateway.
action-code-ix	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often implies that the transaction was authorized and captured. This field is only present for transactions that failed at the Gateway.
total-amount	<p>The total currency amount for the whole batch. This field is always present, and is in the currency format (for example, usd 543.34).</p> <p><b>Note:</b> Sales are counted as positive amounts, while credits are counted as negative amounts.</p>
sales-amount	The total sales amount for the whole batch. This field is always present, and is in the currency format.

**TABLE 20:** BATCH-QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
sales-amount-amex	The total sales for American Express for the whole batch. This field is always present, and is in the currency format.
sales-amount-carte	The total sales for Carte Blanche for the whole batch. This field is always present, and is in the currency format.
sales-amount-diners	The total sales for Diners Club for the whole batch. This field is always present, and is in the currency format.
sales-amount-discover	The total sales for Discover for the whole batch. This field is always present, and is in the currency format.
sales-amount-ecp	The total sales for ECP (Paymentech) checks for the whole batch. This field is always present, and is in the currency format.
sales-amount-jcb	The total sales for JCB for the whole batch. This field is always present, and is in the currency format.
sales-amount-mc	The total sales for MasterCard for the whole batch. This field is always present, and is in the currency format.
sales-amount-visa	The total sales for Visa for the whole batch. This field is always present, and is in the currency format.
sales-amount-others	The total sales for all other card types for the whole batch. This field is always present, and is in the currency format.
credit-amount	The total credit amount for the whole batch. This field is always present, and is in the currency format.

**TABLE 20:** BATCH-QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
credit-amount-amex	The total credit for American Express for the whole batch. This field is always present, and is in the currency format.
credit-amount-carte	The total credit for Carte Blanche for the whole batch. This field is always present, and is in the currency format.
credit-amount-diners	The total credit for Diners Club for the whole batch. This field is always present, and is in the currency format.
credit-amount-discover	The total credit for Discover for the whole batch. This field is always present, and is in the currency format.
credit-amount-ecp	The total credit for ECP (Paymentech) checks for the whole batch. This field is always present, and is in the currency format.
credit-amount-jcb	The total credit for JCB for the whole batch. This field is always present, and is in the currency format.
credit-amount-mc	The total credit for MasterCard for the whole batch. This field is always present, and is in the currency format.
credit-amount-visa	The total credit for Visa for the whole batch. This field is always present, and is in the currency format.
credit-amount-others	The total credit for all other card types for the whole batch. This field is always present, and is in the currency format.

## Using batch-unroll

The batch-unroll message is used to query for transactions that were sent in a batch. [Table 21](#) describes the fields you should use to customize the batch-unroll message.

**TABLE 21:** BATCH-UNROLL INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for the transaction you are querying. Order IDs can contain letters, numbers, dashes, underscores, and periods. This field is optional.
txn-type	The transaction types for the transaction you are querying. This field is optional. Special values used only for querying are: anc—authorized not captured anm—authorized not marked msn—marked not settled rns—marked return, return not settled
txn-status	The status of the transaction you are querying. This field is optional. It can assume the following values: success—successful transaction success-duplicate—result of a previously successful transaction partial success—batch has failed transactions failure-hard—transaction failed; subsequent retry will not help failure-q-or-cancel", "failure-q-or-discard—transaction failed due to a communications failure; this may be retried failure-swversion—failure because of old or non-existent software (version) being used failure-bad-money—failure because of a

**TABLE 21:** BATCH-UNROLL INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
origin	<p>The origin of the transaction you are querying. This field is optional and valid values are:</p> <p>c—consumer                      m—merchant                      blank—any (in other words not specified)</p>
card-type	<p>The type of card used for the transaction you are querying. This field is optional and valid values are:</p> <p>ot—other                      ax—American Express                      cb—Carte Blanche                      dc—Diners Club                      ds—"Discover                      mc—MasterCard                      vs—Visa                      ecp-savings—Paymentech check service, savings account                      ecp-checking—Paymentech check service, checking account</p>
start-time	<p>The earliest time that the transaction you are querying could have been created. Transactions will be returned that were started at this time or later. This field is optional.</p>
end-time	<p>The latest time that the transaction you are querying could have been created. Transactions will be returned that were created any time prior to this time. This field is optional.</p>

**TABLE 21:** BATCH-UNROLL INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
low-amount	The lowest amount for a transaction you are querying. Transactions will be returned that are for this amount or higher. Use the format currency dollar.cents. For example, usd 12.50. This field is optional.
high-amount	The highest amount for a transaction you are querying. Transactions will be returned that are for this amount or lower. Use the format currency dollar.cents. For example, usd 12.50. This field is optional.
maxcount	The highest number of transactions that you want returned from the query. This field is optional.
batch-id	The batch ID of the transaction(s) you are querying. This field is optional.

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$txnType = 'msn';
$hiAmount = 'usd 200.00'
# send the message to the CashRegister to query the
# transactions database for transactions that were marked and
# not settled, and that were less than $200

%result = &SendCC2_1Server ('batch-unroll',
                           'txn-type',    $txnType,
                           'high-amount', $hiAmount);
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the status of the query
print ("The batch unroll returned with status $result
{'MStatus'}\n");

```

Table 22 describes the output fields that you should expect from the batch-unroll message. The notation "-ix" represents one of many transactions that may be returned from the query. For example, if five transactions were returned for the query, then "order-id-1", "order-id-2", "order-id-3", "order-id-4", and "order-id-5" would be the 5 order IDs of these transactions.

**TABLE 22:** BATCH-UNROLL OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
order-id-ix	The order ID of the transaction. This field always appears.
merch-txn-ix	The number that the Gateway uses to refer a transaction that was performed or attempted. This field always appears.

**TABLE 22:** BATCH-UNROLL OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
cust-txn-ix	The transaction number generated by the CyberCash Wallet to refer to a just-processed transaction. This field is sometimes present.
origin-ix	The origin of the transaction. This field is optional and valid values are: c—consumer m—merchant blank—any (in other words not specified)
txn-type-ix	The transaction types for the transaction. This field is optional. Expected values are: invoice—invoice consumer (Wallet) auth—credit card authorize (this applies to terminal capture payment processors only) postauth—credit card capture (this applies to terminal capture payment processors only) capture—credit card capture and authorize (applies to host capture payment processors) return—return money to credit card marked—mark capture for batching (batch processors only) markret—mark return for batching (batch processors only) settled— settle previously marked capture

**TABLE 22:** BATCH-UNROLL OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
txn-status-ix	<p>The status of the transaction. This field is always present. It can assume the following values:</p> <p>success—successful transaction</p> <p>success-duplicate—result of a previously successful transaction</p> <p>partial success—batch has failed transactions</p> <p>failure-hard—transaction failed; subsequent retry will not help</p> <p>failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried</p> <p>failure-swversion—failure because of old or non-existent software (version) being used</p> <p>failure-bad-money—failure because of a</p>
cust-id-ix	<p>The consumer (Wallet) ID. This ID identifies the Wallet to the Gateway. This field is sometimes present.</p>
card-type-ix	<p>The type of card used for the transaction. This field is optional and valid values are:</p> <p>ot—other</p> <p>ax—American Express</p> <p>cb—Carte Blanche</p> <p>dc—Diners Club</p> <p>ds—"Discover</p> <p>mc—MasterCard</p> <p>vs—Visa</p> <p>eep-savings—Paymentech check service, savings account</p> <p>eep-checking—Paymentech check service, checking account</p>

**TABLE 22:** BATCH-UNROLL OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-number-ix	The number of the credit card used for the transaction. This field is always present.
card-exp-ix	The expiration date of the credit card used for the transaction. This field is sometimes present, and has the format MM/YY (for example, 7/99 for July 1999).
amount-ix	The amount of the transaction. This field is always present.
time-ix	The time the transaction occurred. This field is always present.
auth-code-ix	The authorization code returned from the payment processor for the transaction. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
ref-code-ix	The retrieval reference number for the transaction. This data is processor-specific and may not always be present.
action-code-ix	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often implies that the transaction was authorized and captured. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
batch-id-ix	The batch ID of the transaction. This field is always present.

**TABLE 22:** BATCH-UNROLL OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
avs-code-ix	The Address Verification System (AVS) code returned from the processor for the transaction. This field may only be present if the transaction type (txn-type-ix) of the transaction is "batch" and is processor-dependent; consult your payment processor for details of these codes.
gw-batch-id-ix	The Gateway batch ID of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch".
batch-status-ix	<p>The batch status of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch". This field is optional, and valid values are:</p> <ul style="list-style-type: none"> <li>unknown—unknown</li> <li>batch-queued—batch is queued at Gateway to be submitted to payment processor</li> <li>batch-sent—batch was sent to payment processor, status pending</li> <li>batch-accepted—batch was accepted by processor (success)</li> <li>batch-rejected—batch was rejected by processor (failure)</li> <li>batch-error-at-cc—error occurred while processing the batch at CyberCash</li> </ul>

---

## Using card-query

The card-query message is used to query for a given card/check transaction (order) from the Gateway. Using this message, it is possible to get data about a particular transaction that might not usually be available (for example, the credit card number used in a particular transaction). This message will only be applicable for credit card or check transactions. [Table 23](#) describes the fields you should use to customize the card-query message.

**TABLE 23: CARD-QUERY INPUT FIELDS**

FIELD	DESCRIPTION
order-id	The order ID for the card transaction you are querying. This field is required.
passwd	The merchant "secret" that is needed to perform this query. This secret is given to the merchant at time of registration. This field is required.

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID = $savedOrderID; # this variable was previously
assigned
$secret      = $Config {"HASH_SECRET"};
# $Config is initialized when we call InitConfig
# send the message to the CashRegister to query
# for a particular card transaction
%result = &SendCC2_lServer ('card-query,
                           'order-id', $orderID,
                           'passwd',  $secret);

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the card type and the cardholder's name
print ("got back type $card-type for holder $result {'card-
name'}\n");

```

[Table 24](#) describes the output fields that you should expect from the card-query message.

**TABLE 24:** CARD-QUERY OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
aux-msg	The Gateway merchant message may contain verbiage from the Gateway or payment server. This field may not always be present.
merch-txn	The number (ID) that the Gateway uses to refer to a transaction that was performed or attempted. This field always appears.

**TABLE 24:** CARD-QUERY OUTPUT FIELDS

FIELD	DESCRIPTION
MSWErrMsg	The error message when an outdated version of the Wallet or the payment server is being used. This field may not always be present.
card-type	The type of card used in the transaction. This field is always present. Valid values for this field are: ot—other ax—American Express cb—Carte Blanche dc—Diners Club ds—"Discover mc—MasterCard vs—Visa ecp-savings—Paymentech check service, savings account ecp-checking—Paymentech check service, checking account
card-number	The card number used in the transaction. This field is always present. If the transaction involved a Paymentech check (service), this number represents the MICR line of the check
card-exp	The expiration of the card used in the transaction. This field is present for card transactions but not for Paymentech check transactions, and is of the format MM/YY (for example, "6/98" for June 1998).
card-name	The name of the cardholder or checking/savings account holder. This field is always present.

## Using checkauth

The checkauth message is used to validate and authorize a merchant-initiated check payment. This message is available only for the FirstUSA Paymentech processor. [Table 25](#) describes the fields you should use to customize the checkauth message.

**TABLE 25:** CHECKAUTH INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize for the check transaction. Use the format currency dollar.cents (for example, usd 12.50). This field is required.
card-number	The numeric component of the MICR line on the check used for this transaction. This number includes the financial institution routing number, account number, and check number. This field is required.
card-name	The name of the check used for this transaction. This field is required.
card-type	The type of account used in this transaction. This field is required. Valid values are: ecp-savings—FirstUSA Paymentech check service, savings account ecp-checking—FirstUSA Paymentech check service, checking account

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID = &GenerateOrderId ();
$authAmt = 'usd 54.35'
$cardNumber = '123456789009876543213456'
$cardName = 'John Doe'
# send the message to the CashRegister to authorize
# and validate a check transaction
%result = &SendCC2_lServer ('checkauth,
                            'order-id', $orderID,
                            'card-number', $cardNumber,
                            'card-name', $cardName,
                            'card-type', 'ecp-checking');
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print an error message if transaction didn't succeed
if (($MStatus != "success") && ($MStatus != "success-duplicate"))
{
    print ("check transaction failed at $result {'MErrLoc'}...\n");
}

```

[Table 26](#) describes the output fields that you should expect from the checkauth message.

**TABLE 26:** CHECKAUTH OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14, page 125</a> .

## Using checkreturn

The checkreturn message is used to return money to a consumer's checking or savings account. Prior authorization of the amount to be returned is not required. If only an order ID and amount are specified for this message and the order ID is found at CyberCash, money is returned for an existing order for the amount specified. If other input fields are specified, the order return is processed with no need for a previous authorization. This message is available only for the FirstUSA Paymenttech processor. [Table 27](#) describes the fields you should use to customize the checkreturn message.

**TABLE 27:** CHECKRETURN INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize (that is, the amount to be returned) for the check transaction. Use the format currency dollar.cents (for example, usd 12.50). This field is required.
card-number	The numeric component of the MICR line on the check used for this transaction. This number includes the financial institution routing number, account number, and check number. This field is required only for returns with no prior authorization.
card-name	The name of the check used for this transaction. This field is required only for returns with no prior authorization.
card-type	The type of account used in this transaction. This field is required only for returns with no prior authorization. Valid values are: ecp-savings—FirstUSA Paymenttech check service, savings account ecp-checking—FirstUSA Paymenttech check service, checking account

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID      = $savedOrder      # from a prior check authorization
$refund       = 'usd 35.00'
# send the message to the CashRegister to credit
# a checking/savings account with a return
%result = &SendCC2_1Server ('checkreturn',
                           'order-id',    $orderID,
                           'amount',     $refund);

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}

# print the type of account credited if successful
if (($MStatus == "success") || ($MStatus == "success-duplicate"))
{
    print ("account type $result {'card-type'}\n");
}

```

[Table 28](#) describes the output fields that you should expect from the checkreturn message.

**TABLE 28: CHECKRETURN OUTPUT FIELDS**

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14, page 125</a> .

## Using mauthcapture

The mauthcapture message is used to authorize and capture a merchant-originated credit card sale. Use this message only for credit card transactions made through host capture payment processors. [Table 29](#) describes the fields you should use to customize the mauthcapture message.

**TABLE 29:** MAUTHCAPTURE INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize (that is, the amount to be returned) for this transaction. Use the format currency dollar.cents (for example, usd 12.50). This field is required.
card-number	The card number for the credit card used for this transaction. This field is required.
card-exp	The expiration date of the credit card used for this transaction. Use the format month/year. For example, 2/97 for February 1997. This field is required.
card-name	The name of the credit card owner. This field is optional.
card-address	The street address of the credit card owner. This field is optional.
card-city	The city of the credit card owner. This field is optional.
card-zip	The postal zip code of the credit card owner. The following examples are all valid entries: 22091, 20191-1448, NW3 5RJ, 113 192. This field is optional.

**TABLE 29:** MAUTHCAPTURE INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-state	The state of the credit card owner. This field is optional.
card-country	The country of the credit card owner. This field is optional.

The following message shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID = &GenerateOrderId ();
$authAmt = 'usd 29.95'
$cardNumber = '4111111121111111'
$cardExpr = '5/98'
$cardName = 'Jane Doe'
# send the message to the CashRegister to authorize
# and capture a card transaction
$result = &SendCC2_1Server ('mauthcapture',
                           'order-id',    $orderID,
                           'amount',     $authAmt,
                           'card-number', $cardNumber,
                           'card-name',   $cardName,
                           'card-exp',   $cardExpr);

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print an error message if transaction didn't succeed
if (($MStatus != "success") && ($MStatus != "success-duplicate"))
{
    print ("card transaction failed at $result {'MErrLoc'}...\n");
}

```

Table 30 describes the output fields that you should expect from the mauthcapture message. If Gateway communications fails on this message, you can retry the authorization/capture transaction by using the *retry* message.

**TABLE 30:** MAUTHCAPTURE OUTPUT FIELDS

FIELDS	DESCRIPTION
Standard Merchant Output Fields	These are described in Table 13, page 124.
Standard Gateway Output Fields	These are described in Table 14, page 125.

---

## Using mauthonly

The mauthonly message is used to authorize a merchant-originated credit card sale. Use this message only for credit card transactions made through terminal capture payment processors. Table 31 describes the fields you should use to customize the mauthonly message.

**TABLE 31:** MAUTHONLY INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize (that is, the amount to be returned) for this transaction. Use the format currency dollar.cents (for example, usd 12.50). This field is required.
card-number	The card number for the credit card used for this transaction. This field is required.

**TABLE 31:** MAUTHONLY INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-exp	The expiration date of the credit card used for this transaction. Use the format month/year. For example, 2/97 for February 1997. This field is required.
card-name	The name of the credit card owner. This field is optional.
card-address	The street address of the credit card owner. This field is optional.
card-city	The city of the credit card owner. This field is optional.
card-zip	The postal zip code of the credit card owner. The following examples are all valid entries: 22091, 20191-1448, NW3 5RJ, 113 192. This field is optional.
card-state	The state of the credit card owner. This field is optional.
card-country	The country of the credit card owner. This field is optional.

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$savedOrder = &GenerateOrderId ();
$authAmt    = 'usd 49.95'
$cardNumber = '511116721111111'
$cardExpr   = '11/99'
# send the message to the CashRegister to authorize
# a card transaction
%result = &SendCC2_1Server ('mauthonly',
                           'order-id',    $savedOrder,
                           'amount',     $authAmt,
                           'card-number', $cardNumber,
                           'card-exp',    $cardExpr);
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the authorization code if successful
if (($MStatus == "success") || ($MStatus == "success-
duplicate"))
{
    print ("card authorization code $result {'auth-
code'}...\n");
}

```

[Table 32](#) describes the output fields that you should expect from the `mauthonly` message. If Gateway communication fails on this message, you can retry the authorization transaction by using the `retry` message.

**TABLE 32:** MAUTHONLY OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14, page 125</a> .

## Using postauth

The postauth message is used to capture a previously authorized merchant-initiated card payment; this payment must have been authorized by a mauthonly or a checkauth message. [Table 33](#) describes the fields you should use to customize the postauth message.

**TABLE 33:** POSTAUTH INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize for the check transaction. Use the format currency dollar.cents (for example, usd 12.50). This may be less than the authorized amount; it may not be more. This field is required.

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderId = $savedOrder      # from a prior authorization
$captureAmt = 'usd 44.95'
# send the message to the CashRegister to capture
# a previously authorized card transaction
%result = &SendCC2_lServer ('postauth',
                           'order-id',    $orderId,
                           'amount',     $captureAmt);

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the reference code if successful
if (($MStatus == "success") || ($MStatus == "success-
duplicate"))
{
    print ("processor reference code $result {'ref-
code'}...\n");
}

```

[Table 34](#) describes the output fields that you should expect from the postauth message. If Gateway communication fails on this message, you can retry the capture transaction by using the *retry* message.

**TABLE 34:** POSTAUTH OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14, page 125</a> .

## Using query

The query message is used to query the transactions database. [Table 35](#) describes the fields you should use to customize the query message.

**TABLE 35:** QUERY INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for the transaction you are querying. Order IDs can contain letters, numbers, dashes, underscores, and periods. This field is optional.
txn-type	The transaction types for the transaction you are querying. This field is optional. Special values used only for querying are: anc—authorized not captured anm—authorized not marked msn—marked not settled rns—marked return, return not settled
txn-status	The status of the transaction you are querying. This field is optional. It can assume the following values: success—successful transaction success-duplicate—result of a previously successful transaction partial success—batch has failed transactions failure-hard—transaction failed; subsequent retry will not help failure-q-or-cancel, failure-q-or-discard—transaction failed due to a communications failure; this may be retried failure-swversion—failure because of old or non-existent software (version) being used failure-bad-money—failure because of a

**TABLE 35:** QUERY INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
origin	<p>The origin of the transaction you are querying. This field is optional and valid values are:</p> <ul style="list-style-type: none"> <li>c—consumer</li> <li>m—merchant</li> <li>blank—any (in other words not specified)</li> </ul>
card-type	<p>The type of card used for the transaction you are querying. This field is optional and valid values are:</p> <ul style="list-style-type: none"> <li>ot—other</li> <li>ax—American Express</li> <li>cb—Carte Blanche</li> <li>dc—Diners Club</li> <li>ds—"Discover</li> <li>mc—MasterCard</li> <li>vs—Visa</li> <li>ecp-savings—Paymentech check service, savings account</li> <li>ecp-checking—Paymentech check service, checking account</li> </ul>
start-time	<p>The earliest time that the transaction you are querying could have been created. Transactions will be returned that were started at this time or later. This field is optional.</p>
end-time	<p>The latest time that the transaction you are querying could have been created. Transactions will be returned that were created any time prior to this time. This field is optional.</p>

**TABLE 35:** QUERY INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
low-amount	The lowest amount for a transaction you are querying. Transactions will be returned that are for this amount or higher. Use the format currency dollar.cents. For example, use 12.50. This field is optional.
high-amount	The highest amount for a transaction you are querying. Transactions will be returned that are for this amount or lower. Use the format currency dollar.cents. (For example, use 12.50). This field is optional.
maxcount	The highest number of transactions that you want returned from the query. This field is optional.
batch-id	The batch ID of the transaction(s) you are querying. This field is optional.

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely..
    die ("$0: unable to initialize configuration!\n");
}
$txnType = 'anm';
$cardType = 'ax';
# send the message to the CashRegister to query the
# transactions database for AmEx transactions that were
# authorized
# and not marked.
%result = &SendCC2_1Server ('query',
                           'txn-type', $txnType,
                           'card-type', $cardType);
# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}

```

Table 36 describes the output fields that you should expect from the query message. The notation "-ix" represents one of many transactions that may be returned from the query. For example, if five transactions were returned for the query, then "order-id-1", "order-id-2", "order-id-3", "order-id-4", and "order-id-5" would be the 5 order IDs of these transactions.

**TABLE 36:** QUERY OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13, page 124</a> .
order-id-ix	The order ID of the transaction. This field always appears.
merch-txn-ix	The number that the Gateway uses to refer a transaction that was performed or attempted. This field always appears.
cust-txn-ix	The transaction number generated by the CyberCash Wallet to refer to a just-processed transaction. This field is sometimes present.
origin-ix	The origin of the transaction. This field is optional and valid values are: c—consumer m—merchant blank—any (in other words not specified)

TABLE 36: QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
txn-type-ix	<p>The transaction types for the transaction. This field is optional. Expected values are:</p> <ul style="list-style-type: none"> <li>invoice—invoice consumer (Wallet)</li> <li>auth—credit card authorize (this applies to terminal capture payment processors only)</li> <li>postauth—credit card capture (this applies to terminal capture payment processors only)</li> <li>capture—credit card capture and authorize (applies to host capture payment processors)</li> <li>return—return money to credit card</li> <li>marked—mark capture for batching (batch processors only)</li> <li>markret—mark return for batching (batch processors only)</li> <li>settled— settle previously marked capture</li> </ul>
txn-status-ix	<p>The status of the transaction. This field is always present. It can assume the following values:</p> <ul style="list-style-type: none"> <li>success—successful transaction</li> <li>success-duplicate—result of a previously successful transaction</li> <li>partial success—batch has failed transactions</li> <li>failure-hard—transaction failed; subsequent retry will not help</li> <li>failure-q-or-cancel", "failure-q-or-discard—transaction failed due to a communications failure; this may be retried</li> <li>failure-swversion—failure because of old or non-existent software (version) being used</li> <li>failure-bad-money—failure because of a</li> </ul>

**TABLE 36:** QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
cust-id-ix	The consumer (Wallet) ID. This ID identifies the Wallet to the Gateway. This field is sometimes present.
card-type-ix	The type of card used for the transaction. This field is optional and valid values are: ot—other ax—American Express cb—Carte Blanche dc—Diners Club ds—"Discover mc—MasterCard vs—Visa ecp-savings—Paymentech check service, savings account ecp-checking—Paymentech check service, checking account
card-number-ix	The number of the credit card used for the transaction. This field is always present.
card-exp-ix	The expiration date of the credit card used for the transaction. This field is sometimes present, and has the format MM/YY (for example, 7/99 for July 1999).
amount-ix	The amount of the transaction. This field is always present.
time-ix	The time the transaction occurred. This field is always present.
auth-code-ix	The authorization code returned from the payment processor for the transaction. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.

**TABLE 36:** QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
ref-code-ix	The retrieval reference number for the transaction. This data is processor-specific and may not always be present.
action-code-ix	The three-digit action code returned by the financial institution. If you do an authcapture and get back a 000, this indicates that the transaction was authorized but not captured; 007 often implies that the transaction was authorized and captured. This field may not always be present and is processor-dependent; consult your payment processor for details of these codes.
batch-id-ix	The batch ID of the transaction. This field is always present.
avs-code-ix	The Address Verification System (AVS) code returned from the processor for the transaction. This field may only be present if the transaction type (txn-type-ix) of the transaction is "batch". This field is processor-dependent; consult your payment processor for details of these codes.

**TABLE 36:** QUERY OUTPUT FIELDS (CONT.)

FIELD	DESCRIPTION
gw-batch-id-ix	The Gateway batch ID of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch".
batch-status-ix	The batch status of the transaction. This field is only present if the transaction type (txn-type-ix) of the transaction is "batch". This field is optional, and valid values are: unknown—unknown batch-queued—batch is queued at Gateway to be submitted to payment processor batch-sent—batch was sent to payment processor, status pending batch-accepted—batch was accepted by processor (success) batch-rejected—batch was rejected by processor (failure) batch-error-at-cc—error occurred while processing the batch at CyberCash

---

## Using retry

The `retry` message is used to retry a transaction that is pending for a given order. [Table 37, page 175](#), describes the fields you should use to customize the `retry` message.

**TABLE 37:** RETRY INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for the pending transaction to be retried. This field is required.
txn-type	The transaction type to retry. This field is only required if it is required that we are retrying in order to void an order. Permissible values are: invoice—invoice consumer (Wallet) auth—credit card authorize (this applies to terminal capture payment processors only) postauth—credit card capture (this applies to terminal capture payment processors only) capture—credit card capture and authorize (applies to host capture payment processors) return—return money to credit card marked—mark capture for batching (batch processors only) markret—mark return for batching (batch processors only)

The following example shows how to use this message in Perl.

```
$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$savedOrder = &GenerateOrderId ();
$authAmt    = 'usd 49.95'
$cardNumber = '511116721111111'
$cardExpr   = '11/99'
# send the message to the CashRegister to authorize
# a card transaction
%result = &SendCC2_1Server ('mauthonly',
                            'order-id',    $savedOrder,
                            'amount',      $authAmt,
                            'card-number', $cardNumber,
                            'card-exp',    $cardExpr);
if (($MStatus == "failure-q-or-cancel") ||
    ($MStatus == "failure-q-or-discard"))
{
    $orderId = $savedOrder # from a prior authorization
    attempt
    # send the message to the CashRegister to retry
    # a prior failed transaction
    %result = &SendCC2_1Server ('retry',
                                'order-id',    $orderId);
    # print all the name-value pairs returned by the message
    foreach (keys (%result))
    {
        print (" $_ -> $result {$_}\n");
    }
}
```

[Table 38, page 177](#), describes the output fields that you should expect from the retry message.

**TABLE 38:** RETRY OUTPUT FIELDS

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14</a> .

## Using return

The return message is used to return money to a consumer's credit card. Prior authorization of the amount to be returned is not required. If only an order ID and amount are specified for this message, and the order ID is found in the transaction database, money is returned for an existing order for the amount specified. If other input fields are specified, the order return is processed with no need for a previous authorization. This message is available only for credit card returns. [Table 39](#) describes the fields you should use to customize the return message.

**TABLE 39:** RETURN INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for this transaction. This field is required.
amount	The amount to authorize (that is, the amount to be returned) for the credit card transaction. Use the format currency dollar.cents (for example, usd 12.50). This field is required.
card-number	The card number for the credit card used for this transaction. This field is required only if no prior authorization exists.

**TABLE 39:** RETURN INPUT FIELDS (CONT.)

FIELD	DESCRIPTION
card-exp	The expiration date of the credit card used for this transaction. Use the format month/year. (For example, 2/97 for February 1997). This field is required only if no prior authorization exists.
card-name	The name of the credit card owner. This field is optional.
card-address	The street address of the credit card owner. This field is optional.
card-city	The city of the credit card owner. This field is optional.
card-zip	The postal zip code of the credit card owner. The following examples are all valid entries: 22091, 20191-1448, NW3 5RJ, 113 192. This field is optional.

The following example shows how to use this example in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID      = $savedOrder      # from a prior authorization
attempt
$refund       = 'usd 35.00'
# send the message to the CashRegister to credit
# a card with a refund/return
$result = &SendCC2_1Server ('return',
                           'order-id',    $orderId,
                           'amount',     $refund);

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the action code if successful
if (($MStatus == "success") || ($MStatus == "success-duplicate"))
{
    print ("processor action code $result {'action-code'}...\n");
}

```

[Table 40](#) describes the output fields that you should expect from the return message.

**TABLE 40: RETURN OUTPUT FIELDS**

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14</a> .

## Using void

The void message is used to void a transaction. [Table 41](#) describes the fields you should use to customize the retry message.

**TABLE 41:** VOID INPUT FIELDS

FIELD	DESCRIPTION
order-id	The order ID for the transaction to be made void. This field is required.
txn-type	The transaction type to void. This field is required. Valid values are: invoice—invoice consumer (Wallet) auth—credit card authorize (this applies to terminal capture payment processors only) postauth—credit card capture (this applies to terminal capture payment processors only) capture—credit card capture and authorize (applies to host capture payment processors) return—return money to credit card marked—mark capture for batching (batch processors only) markret—mark return for batching (batch processors only) settled—settle previously marked capture

The following example shows how to use this message in Perl.

```

$status= &InitConfig ($ConfigFile)
if ($status != 0)
{
    # failed to initialize configuration completely...
    die ("$0: unable to initialize configuration!\n");
}
$orderID      = $savedOrder      # from a prior authorization
attempt
# send the message to the CashRegister to void
# a previous authorized transaction
%result = &SendCC2_lServer ('void',
                           'order-id',    $orderID,
                           'txn-type',    'auth');

# print all the name-value pairs returned by the message
foreach (keys (%result))
{
    print (" $_ -> $result {$_}\n");
}
# print the card type if successful
if (($MStatus == "success") || ($MStatus == "success-duplicate"))
{
    print ("return on card of type $result {'card-type'}...\n");
}

```

[Table 42](#) describes the output fields that you should expect from the void message.

**TABLE 42: VOID OUTPUT FIELDS**

FIELD	DESCRIPTION
Standard Merchant Output Fields	These are described in <a href="#">Table 13</a> .
Standard Gateway Output Fields	These are described in <a href="#">Table 14</a> .

## Understanding Error Codes

Table 43 lists the errors that might be received if you are using the Merchant Connect Kit (MCK) and cybercash Payment Services to provide electronic credit card and check payment solutions. If you are using the C programming language to implement your storefront interface to CyberCash, you can use the C constants named below; these are found in *CCErrno.h* in the *c-api* directory. If you are using PERL, you may use the PERL variables described below by using the library *CCMckErrno3\_2* in your PERL script; the file *CCMckErrno3\_2.pm* is located in the *perl-api* directory.

Some errors are only specific to the PERL programming environment, and some only to the C environment; the errors that may be referenced by a PERL/C variable or constant occur at the MCK layer of the programming interface. Errors are also generated at Cybercash Payment Services, at the Gateway, and at the payment processors; some of these error codes and descriptions are outlined below, but no variable or constant names are used to refer to it. Also, the following list of error codes that occur outside the MCK is not an exhaustive compilation.

**NOTE:** If you are using the PERL or C API, the name-values pairs *MErrLoc* and *MErrMsg* describe the location of the error and a description of it. For information about these standard output fields, please refer to Appendix A. CyberCash suggests that error handling for a consumer-initiated browser (HTTP) request should be handled by modifying the error templates that accompany the MCK. For details on modifying the error templates, please consult *Customizing the MCK*, page 87.

**TABLE 43: COMMON ERROR CODES**

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-300	CC_ERROR	E_ERROR	Generic error
-301	CC_ENOPOST	E_No_Post	Expecting HTTP POST message
-302	CC_ENOCONT_LEN	E_No_Cont_Len	No content-length
-303	CC_EBADCONT_LEN	E_BadCont_Len	Bad content-length

**TABLE 43:** COMMON ERROR CODES (CONT.)

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-304	CC_ENULL_MSG	E_Null_Msg	POST message body empty
-305	CC_ENO_METHOD	E_No_Method	No HTTP request method specified
-306	CC_ENOGET	E_No_Get	Expecting HTTP GET request
-307	CC_EBAD_METHOD	E_BadMethod	Unexpected HTTP request method
-308	CC_EBROWSER	E_Browser	Unsupported browser
-310	CC_ENO_AMOUNT	E_No_Amount	No price/amount specified
-311	CC_EBAD_AMOUNT	E_Bad_Amount	Invalid price/amount specified
-312	CC_EBIG_AMOUNT	E_Big_Amount	Invalid price/amount specified
-313	CC_ENEG_AMOUNT	E_Neg_Amount	Price/amount specified is negative or zero
-314	CC_EBAD_CURRENCY	E_Bad_Currency	Currency specified is invalid
-315	CC_EREJECT_CURRENCY	E_Reject_Currency	Currency specified is not accepted
-316	CC_ENO_ORDERID	E_No_Orderid	No Order ID specified
-317	CC_EBAD_ORDERID	E_Bad_Orderid	Invalid Order ID specified
-318	CC_ENO_PAYLOAD	E_No_Payload	Cannot open/read payload

TABLE 43: COMMON ERROR CODES (CONT.)

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-319	CC_EBAD_PAYLOAD	E_Bad_Payload	Invalid payload
-330	CC_EBAD_SSL	E_No_SSL	Strong-Secure-Layer (SSL) is unsupported for direct connect
-331	CC_EBAD_URL	E_Bad_Url	URL is malformed
-332	CC_ENOT_HTTP	E_Not_HTTP	Invalid HTTP response
-333	CC_EFAILED_HTTP	E_Failed_HTTP	HTTP request failed
-334	CC_ENO_CASHREG	E_No_CashReg	Missing CashRegister host or port
-335	CC_ENO_MESSAGE	E_No_Message	Missing message for <i>CCSocketSend</i>
-340	CC_ESOCKET_FAILED	E_SocketFailed	System <i>socket ()</i> failed
-341	CC_ECONNECT_FAILED	E_ConnectFailed	System socket <i>connect ()</i> failed
-342	CC_EDNS_FAILED	E_DNS_Failed	Unable to resolve host name
-343	CC_EALARM_FAILED	E_Alarm_Failed	UNIX Alarm failure
-344	CC_ESOCKOPT_FAILED	E_Sockopt_Failed	System <i>setsockopt ()</i> failed
--345		E_SyswriteFailed	Socket <i>write ()</i> failed
-350	CC_ENO_CONFIG	E_No_Config	Cannot open MCK configuration file

**TABLE 43:** COMMON ERROR CODES (CONT.)

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-351	CC_ENULL_CONFIG	E_Null_Config	MCK configuration is not initialized
-352	CC_EBAD_CONFIG	E_Bad_Config	MCK configuration file has syntax errors
-353	CC_EBAD_TIMEOUT	E_Bad_Timeout	Invalid timeout value ( $\leq 0$ )
-360	CC_EGO_AWAY	E_GoAway	Server rejected request
-361	CC_ETIMED_OUT	E_TimedOut	Server request timed out
-362	CC_ENO_POP	E_No_POP	No Proof-of-Purchase (POP) record returned for payment request
-363	CC_EPOP_SIGNATURE	E_POP_Signature	Proof-of-Purchase (POP) signature error
-364	CC_EMF_SIGNATURE	E_MF_Signature	Merchant data (MF) signature error
-365	CC_ECASH_REGISTER	E_Cash_Register	Problem at the CashRegister
-366		E_SignatureFailed	Signature error
-367		E_Payment_Failed	Payment failed
-370	CC_ENO_MEMORY	E_No_Memory	Out of memory
-371	CC_ESTAT_FAILED	E_Stat_Failed	<i>stat ()</i> on file failed

**TABLE 43: COMMON ERROR CODES (CONT.)**

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-380	CC_ENO_KEY	E_No_Key	No merchant encryption key
-381	CC_ECRYPTO	E_Crypto	Failure in cryptographic library
-382	CC_EMAC_COMPARE	E_MAC_Compare	Message Authentication Codes (MACs) are not identical
-383		E_EncryptFailed	<i>MCKencrypt</i> failed
-384		E_DecryptFailed	<i>MCKdecrypt</i> failed
-385		E_No_Skey	Missing session key
-386		E_No_MAC	Missing MAC
-390	CC_ENO_TEMPLATE	E_No_Template	Cannot open template
-391	CC_EREAD_TEMPLATE	E_Read_Template	Failure while reading template
-392	CC_ENO_ORDERLOG	E_No_OrderLog	Cannot open the order log
-393	CC_EORDER_NOTFOUND	E_Order_NotFound	Order not found
-394	CC_EORDER_NOTSAVED	E_Order_NotSaved	Order not saved
-395	CC_ENULL_RESULT	E_Null_Result	Null pointer passed in for result parameter
-396	CC_EBAD_PARAMS	E_Bad_Params	Invalid parameters for function call

**TABLE 43: COMMON ERROR CODES (CONT.)**

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-397	CC_ENO_ NOTIFLOG	E_No_NotifLog	Cannot open notification log
-398	CC_ENO_PAY_ PAGE	E_No_Pay_Page	Failed to generate a payment page
-399	CC_EFAIL_ NOTIF	E_Fail_Notif	Unable to log this transaction
-401		E_No_File	Unable to open file
-402		E_No_Receipt	Payment successful, failed to generate receipt
-410	CC_ENV_NO_ NAME		No name specified in name-value (NV) list pair
-411	CC_ENV_NO_ LIST		NV list is Null
-412	CC_ENV_NO_ PLIST		NV list is empty
-413	CC_ENV_NOT_ FOUND		NV pair not found
-420	CC_ENO_ CCMSG		No CyberMessage
-421	CC_EBAD_ CCMSG		Bad CyberMessage
-6001	kCoinFail		Failure in coin transaction
-6002	kInvalidHash		Invalid hashed message/secret

**TABLE 43: COMMON ERROR CODES (CONT.)**

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-6004	kGetInvoiceFail		Failure in credit/coin transaction
-6006	kNullField		Missing required field
-6007	kInvalidPr1		Corrupt field or message
-6009	kNoUrl		Missing URL
-6010	kSystemFailure		System failure
-6011	kInternalFailure		Internal software failure
-6012	kUnknownFailure		Unspecified software failure
-6020	kPriceErr		Invalid Price
-6021	kDuplicate		Offer has previously been presented to the CashRegister
-6022	kFileErr		File I/O error
-6024	kInvalidDate		Date format is incorrect
-6027	kFailedPr1		Failure in coin/credit transaction
-6031	kPayloadCCARC Fail		Failure in coin transaction
-6034	kInvalidNum		Expected and did not get a number
-6036	kUnknownBrowser		Unsupported browser

**TABLE 43:** COMMON ERROR CODES (CONT.)

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-6037	kNoSeedFile		Failure in coin transaction
-6040	kNoMerchantId		Missing/Invalid merchant ID
-6041	kNoOfferId		Missing/Invalid offer ID
-6042	kNoVersion		Missing/Invalid version information
-6043	kNoPrice		Missing price
-6044	kNoNote		Missing/Invalid note
-6045	kNoOvt		Missing/Invalid order valid till date
-6046	kNoSignature		Missing/Invalid signature
-6047	kNoconsumerId		Missing/Invalid consumer ID (for Wallet transactions)
-6050	kNoMcNotification		Missing URL for merchant notification
-6051	kNoFcNotification		Missing URL for fulfillment center notification
-6052	kNoResponse		CashRegister did not respond
-6053	kFailedNotification		Notification attempted but failed

**TABLE 43: COMMON ERROR CODES (CONT.)**

CODE	C CONSTANT	PERL VARIABLE	DESCRIPTION
-6060	kBadUrl		Invalid URL
-6061	kBadAuthMode		Invalid authorization mode (for credit card transactions)
-6062	kFailedCardInput		Invalid credit card information
-6063	kFailedCheckInput		Invalid check information
-6070	kRepudiated		Notification to merchant rejected because order ID not recognized
-6071	kUnknownOfferFailure		Notification to merchant rejected because order ID not recognized. Applies to coin and credit transactions.

In addition to the errors described above, it is sometimes possible to get other error messages. [Table 44, page 190](#), outlines these errors.

**TABLE 44: LESS COMMON ERROR CODES**

ERROR CODE/RANGE	DESCRIPTION
0 ->199	UNIX system error messages. This is usually related to insufficient memory problems.
200 - 220	HTTP parsing error detected at the CashRegister.

**TABLE 44: LESS COMMON ERROR CODES (CONT.)**

ERROR CODE/RANGE	DESCRIPTION
998	CashRegister did not return any output fields on a transaction request. This indicates that the CashRegister was not able to validate the communication mechanism used for the transaction.
999	CashRegister timed out on a transaction request.
less than or equal to -6000	Error at the CashRegister. Most of these errors are described above.

Draft

# Name-Value Pair Variables

---

This appendix describes the merchant offer (mo), credit payment information (cpi), and proof of purchase (pop) variables. Use this chapter in conjunction with [Customizing the MCK, page 87](#). [Table 45, page 194](#), describes these variables.

Draft

**TABLE 45: NAME-VALUE PAIR VARIABLES**

VARIABLE	DESCRIPTION
mo.cybercash-id	<p>The reference used by CyberCash to determine your identity. This is defined during installation.</p> <p><b>!!! DO NOT MODIFY THIS VARIABLE.</b></p>
mo.order-id	<p>The unique identifier used by CyberCash to identify the order. The order ID must be 32 characters or fewer, and it must be unique. The scripts that are included in the MCK automatically generate unique order IDs using the date, time, and process number of the payment. If you want to generate or provide your own order IDs, you can modify this variable in your scripts.</p> <p><b>NOTE:</b> Do not use any of the following characters: : &lt; &gt; = + @ " % &amp;</p> <p>Make sure that you modify your scripts to save order IDs to your order log. The order log is defined during installation, but can be modified if you want to change the location of your order log.</p>
mo.version	<p>The version of the MCK you are using. This is set automatically.</p> <p><b>!!! DO NOT MODIFY THIS VARIABLE.</b></p>
mo.order-valid-till	<p>The date the bill or sale expires. This is required, but can be any type of data string. The <b>mo.order-valid-till</b> variable is included in the notification log.</p> <p>This variable is optional.</p>

**TABLE 45: NAME-VALUE PAIR VARIABLES (CONT.)**

VARIABLE	DESCRIPTION
mo.customer-id	The ID assigned to the customer by you or CyberCash. This variable is optional.
mo.signed-cpi	The setting required for signed CPI blocks. If a CPI is signed, this variable is either not present, or set to <b>no</b> . If the CPI was not signed and collected on the payment page, then this variable is set to <b>yes</b> . This variable is optional.
mo.redirect-url	The URL used in redirecting CyberCash response to a remote script. The MCK provides the <b>fulfillment.*</b> script as an example of what the script <b>mo.redirect-url</b> is sent to should do. This variable is optional.
mo.payload	The name of the payload file being sold if you want to sell soft goods from online files. This variable is optional.
mo.success-page	The HTML template used as a consumer response page if a payment succeeds. This can act as a simple receipt, or it can incorporate POP redirection using <b>mo.redirect-url</b> . This variable is optional.
mo.failure-page	The HTML template used as a consumer response page if a payment fails. This variable is optional.
mo.product-descr	The description of an item being sold. This information is forwarded in the POP. This variable is optional.

TABLE 45: NAME-VALUE PAIR VARIABLES (CONT.)

VARIABLE	DESCRIPTION
mo.failure-url	The URL of an HTML file, with a link to customer support information, that is returned to the customer in the event of a failure.  If you provide a failure page, you do not need this.
mo.price	The price of the purchase sent in the following format: CCC 999999.99 where CCC is the currency code, such as USD for U.S. Dollars, and 999999.99 is the value with leading zeros trimmed.  The scripts included in the MCK automatically generate random prices for you using the GetCart subroutine. You must modify this variable to create your own prices. You may want this information to be obtained from a database.
mo.sign	The signature on the order. This signature contains the hashed information needed by CyberCash and your secret. This is generated and sent automatically.  <b>!!! DO NOT MODIFY THIS VARIABLE.</b>
cpi.card-number	The number on your customer's credit card. This variable should not be modified.
cpi.card-exp	The expiration date on your customer's credit card. This variable should not be modified.
cpi.card-name	The name on your customer's credit card. This information is retrieved from your HTML form. This variable should not be modified.
cpi.card-address	The street address where your credit card customer resides. This variable should not be modified.

**TABLE 45: NAME-VALUE PAIR VARIABLES (CONT.)**

VARIABLE	DESCRIPTION
cpi.card-city	The city where your credit card customer resides. This variable should not be modified.
cpi.card-state	The state where your credit card customer resides. This variable should not be modified.
cpi.card-zip	The zip code under which your credit card customer resides. This variable should not be modified.
cpi.card-country	The country where your credit card customer resides. This variable should not be modified.
cpi.check-account-number	The account number of your customer's checking account. This information is retrieved from your HTML form. This variable should not be modified.
cpi.check-bank-routing-number	The routing number of your customer's bank. This information is retrieved from your HTML form. This variable should not be modified.
cpi.check-name	The name on your customer's check. This information is retrieved from your HTML form. This variable should not be modified.
cpi.check-use	The nature of the checking account. This variable can be set to PERSONAL or BUSINESS. This information is retrieved from your HTML form. This variable should not be modified.  This variable is optional. You can modify this to be defaulted.

TABLE 45: NAME-VALUE PAIR VARIABLES (CONT.)

VARIABLE	DESCRIPTION
cpi.account-paid	The account or billing number of the customer's account with you. This number is assigned by you when a customer signs up for your online services. This information is retrieved from your HTML form. This variable should not be modified.  This variable is optional. You can modify this to be defaulted.
cpi.sign	The signature of your customer's checking account.  <b>!!! DO NOT MODIFY THIS VARIABLE.</b>
pop.cybercash-id	The reference used by CyberCash to determine your identity.
pop.order-id	The ID used by CyberCash to identify the transaction.
pop.customer-id	The customer ID assigned by you to CyberCash. CyberCash uses customer IDs to track customers.
pop.sale-date	The date and time of the sale. It uses the following format:  yyyymmddhhmmss:0000
pop.payload	The name of the payload file being purchased.
pop.price	The price of the purchase sent in the following format:  CCC 999999.99  where CCC is the currency code, such as USD for U.S. Dollars, and <b>999999.99</b> is the value with leading zeros trimmed.
pop.product-descrip	The description of the payment if it is present in the merchant order.

**TABLE 45: NAME-VALUE PAIR VARIABLES (CONT.)**

VARIABLE	DESCRIPTION
pop.order-valid-till	The date the order expires. It is copied from your order information. CyberCash does not look at this value except to include it in the message signature. The format is up to you.
pop.auth-code	The authorization code for credit card payments.
pop.avv-code	The address verification information for credit card payments.
pop.fee	The fee the user is charged for check processing.
pop.status	The outcome of the transaction. The status can be either <b>failure</b> or <b>success</b> . If this is not <b>success</b> , you will see the following: <ul style="list-style-type: none"><li>• <b>pop.error-code</b>: optional error code assigned by failure</li><li>• <b>pop.error-message</b>: optional text of error message</li></ul>
pop.ref-code	The authorization code assigned to the credit payment.
pop.sale-date	The date of service.
pop.txn-id	The transaction ID assigned to the authorization by CyberCash.
pop.sign	The signature on the POP. It is signed with your secret.

Draft