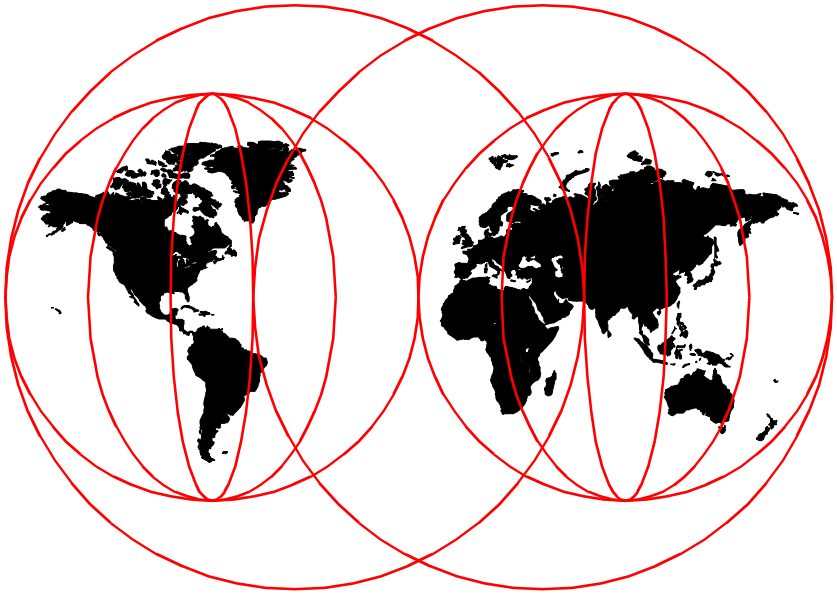
Payment Server V1.2 for AS/400: Secure Transactions in e-commerce

Suehiro Sakai, Masahiko Hamada, Rodney Nelson, Fant Steele



International Technical Support Organization

www.redbooks.ibm.com

SG24-5199-00

SG24-5199-00

International Technical Support Organization

Payment Server V1.2 for AS/400: Secure Transactions in e-commerce

March 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special notices” on page 145.

First Edition (March 2000)

This edition applies to Version 1, Release 2 of IBM Payment Server for AS/400, Program Number 5733-PY1, for use with the OS/400, Program Number 5769-SS1, Version 4, Release 3 and later.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

Welcome to the electronic commerce world! One of the most important parts of online shopping in electronic commerce is to ensure that a secured payment process is in place. As part of IBM Payment Suite, IBM Payment Server V1.2 for AS/400 offers you a way to secure the payment process over the Internet. This redbook will help you plan, install, and configure IBM Payment Server V1.2 for AS/400. IBM Payment Server V1.2 for AS/400 is based on the open standard secure protocol, SET Secure Electronic Transaction, so that it can be used to exchange transaction data with other SET-based payment systems, such as IBM Payment Gateway and IBM Payment Registry.

Prior to reading this redbook, you should have some knowledge of the concept of electronic commerce.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

Suehiro Sakai is an Advisory International Technical Support Specialist for the AS/400 system at the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide in all areas of AS/400 communications. Before joining the ITSO in 1995, he worked in the AS/400 Brand for IBM Japan as an AS/400 Solution Specialist.

Masahiko Hamada is an Advisory I/T specialist in IBM Japan. He has 12 years of experience with IBM mid-range systems. His areas of expertise include Object Oriented (OO) application development, AS/400 connectivity to Microsoft Windows 95/NT, and Client Access/400. He developed ToolBox/400 used in Japanese environments. Currently, his focus is on AS/400 Internet technologies. He has written several technical documents and taught classes in the USA, Europe, and Japan.

Fant Steele is a Senior I/T Technical Specialist on the e-business team in the AS/400 Advanced Technical Support Group located in Rochester, MN. Prior to joining this group, Fant spent two and a half years as an Advisory ITSO Specialist for AS/400 in the International Technical Support Organization (ITSO), Rochester Center. While in the ITSO, Fant wrote several redbooks on communication and e-business topics. Fant also taught classes and seminars worldwide on many areas of AS/400 communications technologies and

e-business. He spent eight years as an instructor and developer for the AS/400 communications and programming curriculum of IBM Education and Training. Prior to joining IBM in 1989, he worked on S/36 to AS/400 code conversion, VM/MVS systems programming, and applications programming for the manufacturing industry.

Rodney Nelson is an I/T Specialist with IBM Global Services in the United States. He has five years of experience in the information technology industry and has worked with IBM for three years. He holds a bachelor's degree in Management Information Systems from Lawrence Technological University and is a Microsoft Certified Systems Engineer. His areas of expertise include AS/400 e-business applications, TCP/IP, Internet security, Windows NT, and Lotus Domino.

Thanks to the following people for their invaluable contributions to this project:

Paul Chmielewski
Tim Fynskov
Thomas Grigoleit
Marty Keech
Jennifer La Rocca
Leah Stier
IBM Rochester Development

David Jackson
IBM US, e-business Payment Solution

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks evaluation” on page 157 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Introduction to Payment Server for AS/400

When you establish an electronic commerce site, you have to decide which payment process (method) to use. One of the preferred methods is to automate the process by using a software running on a merchant system.

The concern of using such software to exchange sensitive information, such as a credit card number, is security. The industry has been working to develop a protocol to secure the payment transactions. It came up with several options, including Secure Socket Layer (SSL) and SET Secure Electronic Transaction.

SET Secure Electronic Transaction is considered the major player of the payment process in the electronic commerce business. IBM Payment Server V1.2, which runs on AS/400 system, is the software product that implements SET.

IBM Payment Server for AS/400 does not need to be installed with Net.Commerce for AS/400. IBM Payment Server for AS/400 can be used with any merchant server. IBM Net.Commerce for AS/400 is one of the merchant servers. You can install and use IBM Payment Server for AS/400 independently from Net.Commerce. For more information, refer to the redbook *Net.Commerce V3.2 for AS/400: A Case Study for Doing Business in the New Millennium,* SG24-5198.

1.1 Typical payment process: The e-commerce way

In the transactions of purchasing goods through e-commerce stores, there are several important roles in the scene. They are:

Cardholder A person who has a valid payment card account and uses software that supports electronic commerce. They are also known as a shopper, online shopper, consumer, or buyer.

Consumer wallet
Software that enables a user to make approved SET payments to authenticated merchants over public networks using SET certificates.

Card issuer In the context of SET programs, a financial institution that issues payment cards to individuals. An issuer can act as its own cardholder certificate authority (CCA) or can contract with a third party for the service.

Merchant An organization that has goods or services to sell to the cardholders.

Acquirer This is an organization that provides card authorization and payment capture services for merchants. A merchant normally wants to accept more than one credit card brand, but does not want to have to deal with multiple bankcard associations. They achieve this by using the services of an acquirer. These services may include verbal or electronic telephone authorization support and electronic transfer of payments to the merchant's account. The services can now include SET protocol support as well. Acquirer services are paid for by the merchant in the form of a small percentage charge on each transaction.

A typical online Web shopping process looks like the example shown in Figure 1.

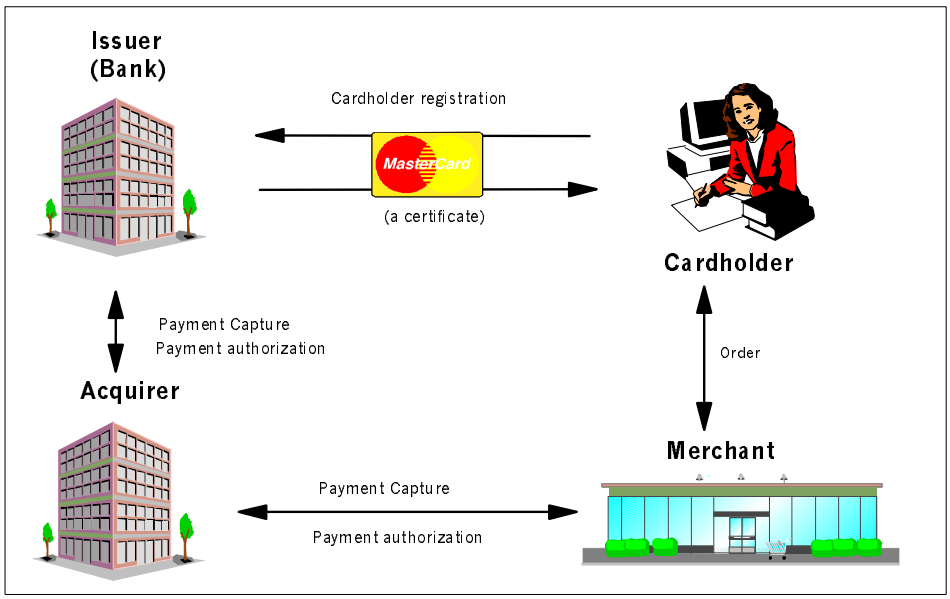


Figure 1. Online payment process

The following steps describe the process of a typical transaction using the electronic payment system:

1. Before purchasing, a cardholder needs to receive the certificate for the card in the electronic wallet software. It needs to be certified with the issuer.
2. The cardholder selects the items to be purchased and the means of payment. The cardholder sends the merchant a completed order along with payment instructions.
3. The merchant requests payment authorization from the cardholder's issuer through the acquirer. The merchant sends confirmation of the order. A cardholder decides to make a purchase.
4. The merchant requests payment from the cardholder's financial institution.

For a more detailed process, see 1.3.1, "A Payment Server transaction" on page 6.

1.2 Security in transactions

On the Internet, there are security features, including SSL and VPN. The most popular and widely used security feature is SSL. Is SSL appropriate for e-business? Is there another way to secure the transactions? The following discussions of SSL and the e-business oriented security feature, SET, can help answer these questions,

1.2.1 Secure Socket Layer (SSL)

SSL is developed by Netscape. Many software vendors support the protocol as the security feature for communicating over the Internet. SSL is designed to encrypt data for data integrity between the sender and receiver, and authenticate the other party. If there are more parties participating in a process (such as the payment process), SSL is not appropriate. For example, when a cardholder sends his credit card number to a merchant in an online purchase, SSL does not guarantee if it is secured while the number is stored in the merchant.

SSL basically uses two different key lengths for its encryption. The shorter and weaker encryption key length is 40 bits, which is not as secure as people expect. The stronger encryption key, which is 128 bits, is regulated not to be used outside the US by the US government.

1.2.2 SET Secure Electronic Transaction

SET Secure Electronic Transaction is an open-network payment-card protocol that provides greater confidentiality, greater transaction integrity, and less opportunity for fraud at all transaction points than any other existing secure payment system. SET was designed by MasterCard and VISA to address the security issues in the online payment process. The process involves a series of security checks performed using digital certificates, which are issued to participating purchasers, merchants, banks, and payment brands.

There are five main parties involved in a SET transaction:

- The cardholder
- The merchant
- The issuer: The customer's financial institution, which provides the payment card to the customer and the payment to the merchant
- The acquirer: The merchant's financial institution, which enables the merchant to accept a payment card brand and issues the captured payment to the merchant
- The certificate authority: A trusted third party that can certify the identities of the customer, the merchant, and the acquiring institution to each other

Four of these parties require their own SET software. The issuer communicates with the acquirer over a secure network or other communications channel, and therefore, does not need a secure Internet implementation.

SET has four components:

Cardholder wallet
: Run by an online consumer enabling secure payment card transactions over a network. SET Cardholder Wallet components must generate SET protocol messages that can be accepted by SET Merchant and Certificate Authority components.

Merchant server
: (Payment Server) Run by an online merchant to process payment card transactions and authorizations. It communicates with the Cardholder Wallet, Payment Gateway, and Certificate Authority components.

Payment gateway
: Run by an acquirer or a designated third party that processes merchant authorization and payment messages (including

payment instructions from cardholders) and interfaces with private financial networks.

Certificate authority
Run by a certificate authority that is authorized to issue and verify digital certificates as requested by Cardholder Wallet components, Merchant Server components, or Payment Gateway components over public and private networks.

Some benefits to merchants for implementing SET are:

- Increased sales from existing online shoppers who can now more confidently expand the number of merchant sites where they shop
- Additional sales from consumers who were traditionally constrained from electronic shopping due to their concerns about security on the Internet
- Increased savings through a reduction of exception handling
- Reduced costs associated with fraud

The SET logo or SET mark is a visible symbol signifying that software complies with the SET specification (Figure 2).

Figure 2. SET logo

For more information about SET, read the redbook *Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*, SG24-4978. You can also refer to the site on the Web at:
`http://www.setco.org`

1.3 Payment Server

The Payment Server provides payment services on the Internet by taking credit card payments from consumers. The Payment Server runs at a merchant, and is used in conjunction with online shopping software, such as Net.Commerce. It supports the SET protocol developed by Visa, MasterCard, IBM, and others. The SET protocol uses strong cryptographic techniques to ensure transaction data is kept private and not improperly modified.

The Payment Server can obtain credit card approvals and capture funds by communicating with a payment gateway, which runs at a bank (typically called an *acquirer*). In addition, it can process deposits and credits or perform reversals.

Figure 3 shows the other servers with which the Payment Server to interacts.

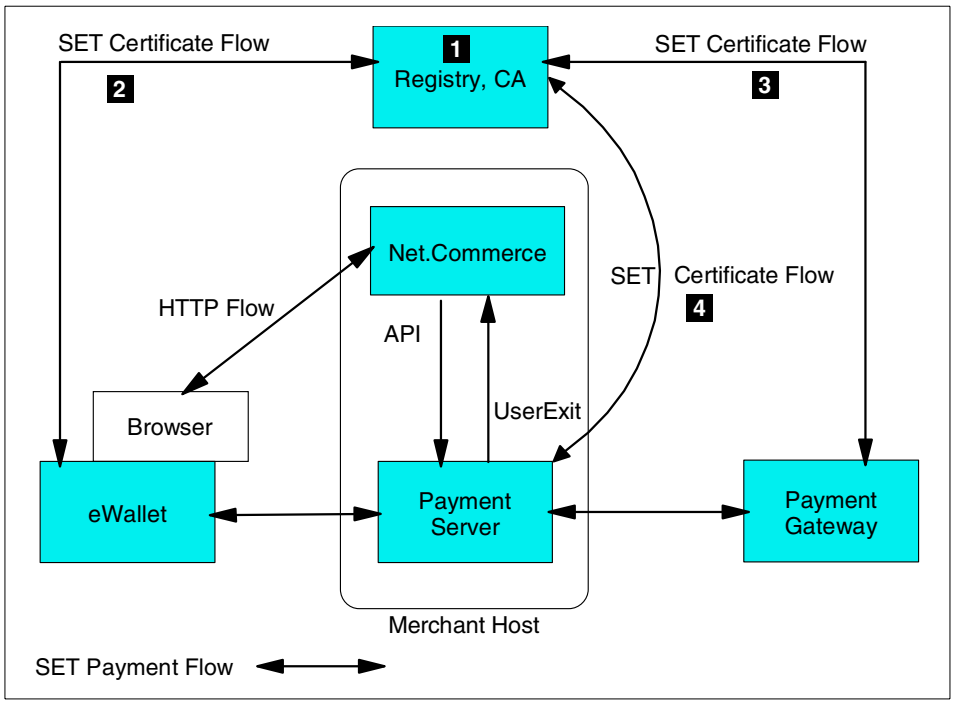


Figure 3. Payment Server

The certificate authorities (CAs) are divided into brands and server types. For more information, see 1.3.2, “SET certificate” on page 9.

1.3.1 A Payment Server transaction

The Payment Server interacts with other applications and servers to achieve SET Secure Electronic Transaction.

Figure 4 illustrates the process of a typical transaction using the IBM Payment Server. Each step of the process is outlined in the list that follows.

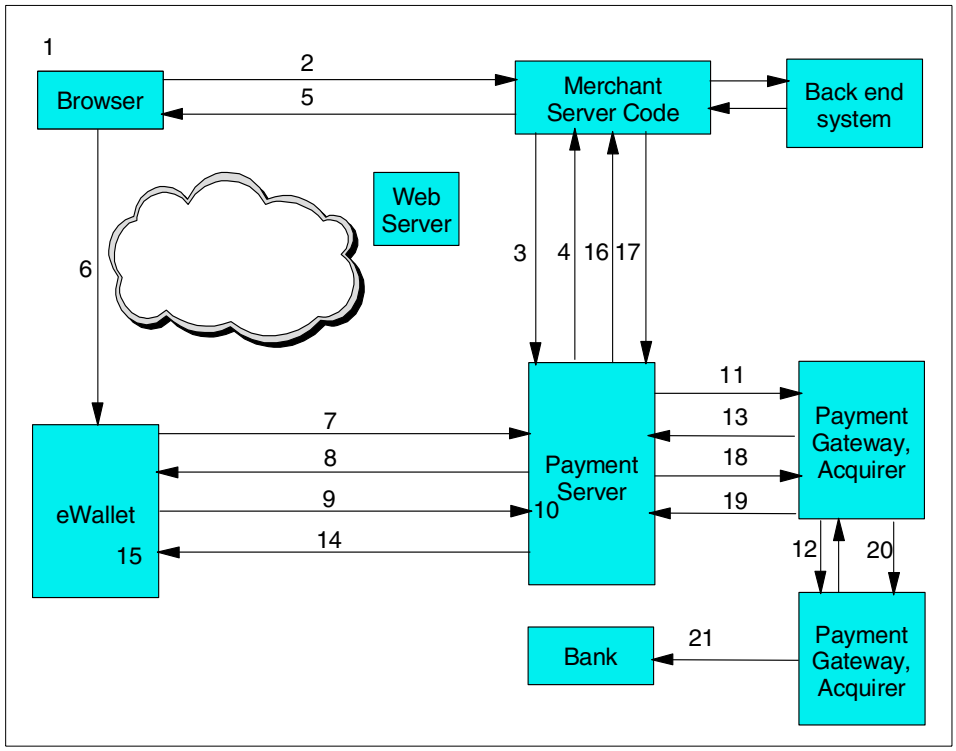


Figure 4. Payment Server transactions

1. A cardholder decides to make a purchase.
2. When the cardholder clicks the Buy button, a command is sent to the Merchant Server.
3. The Merchant Server calls the Payment Server etReceivePayment() API.
4. The command causes the Payment Server to send a payment initiation message, known as a *wakeup message,* back to the Merchant Server.
5. The Merchant Server passes the message back to the cardholder's browser.
6. This message starts the cardholder's wallet software.
7. The wallet software sends a Payment Initialization Request (`PInitReq`) to the Payment Server.
8. A Payment Initialization Response (`PInitRes`) message is generated by the Payment Server and sent back to the cardholder.

9. The wallet displays the Verify Merchant dialog box to the cardholder, who clicks OK. This causes the wallet software to send Purchase Request (`PReq`) message to the merchant. This message includes the order information, the payment card information, and the cardholder's certificate, all encrypted and digitally signed.
10. The Payment Server checks to see if authorization should be done at this point. For example, if the merchant's acquirer is closed on that day, the process may be delayed until the acquirer is available.
11. When authorization can be done, the server generates an Authorization Request (`AuthReq`), sends it to the acquirer, and waits for an Authorization Response (`AuthRes`) message.

> **Note**
>
> This scenario shows the automatic approval process that is achieved by etReceivePayment() API. You can also approve manually with the etApprove() API. You can see the automatic approve and manual approve processes in 3.3, "Sample API sources" on page 76, in more detail.

12. The acquirer software or Payment Gateway receives the request. Using a normal back-end network or other communication channels, the acquiring institution contacts the cardholder's issuing institution. It checks that the payment card is valid and that the cardholder has sufficient funds or credit to make the purchase.
13. The `AuthRes` message is received by the Payment Server and processed. Information is stored in the database for record-keeping and further order processing.
14. A Purchase Response (`PRes`) message is generated by the Payment Server and sent to the cardholder's wallet application.
15. Assuming the Payment Gateway has confirmed authorization, `PRes` tells the wallet software that the order has been authorized.
16. The merchant can now fulfill the order.
17. When the goods are shipped, the merchant requests payment.
18. The merchant now begins the capture process by sending a Capture Request to the Payment Gateway. *Capture* is the transfer of funds from the cardholder's issuing institution to the merchants acquirer and onward to the merchant.

19. The acquirer software receives the capture request and sends it a capture response message.
20. The acquirer uses the closed (back-end) network to contact the cardholder's issuing institution and requests the transfer of payment.
21. The acquirer deposits the payment to the merchant's bank account.

1.3.2 SET certificate

The digital certificates that are used in SET are not the same digital certificates that are used during the SSL. The Payment Server is not using SSL. The Payment Server performs its own encryption of the payment data in a manner which is much more secure than SSL. The SET process uses special certificates and 128-bit encryption for the credit card information, even outside of North America.

The SET specification requires a hierarchy of trust that is similar to today's global payment system model. Cardholders and merchants have trusted relationships with their financial institutions. The financial institutions have existing relationships with one or more payment card brands. Because of the open network environment, SET requires an additional level of trust to authenticate the individual brands. An overall industry entity defined as the SET root certificate authority (CA) authenticates the brands within the SET trust hierarchy (Figure 5 on page 10).

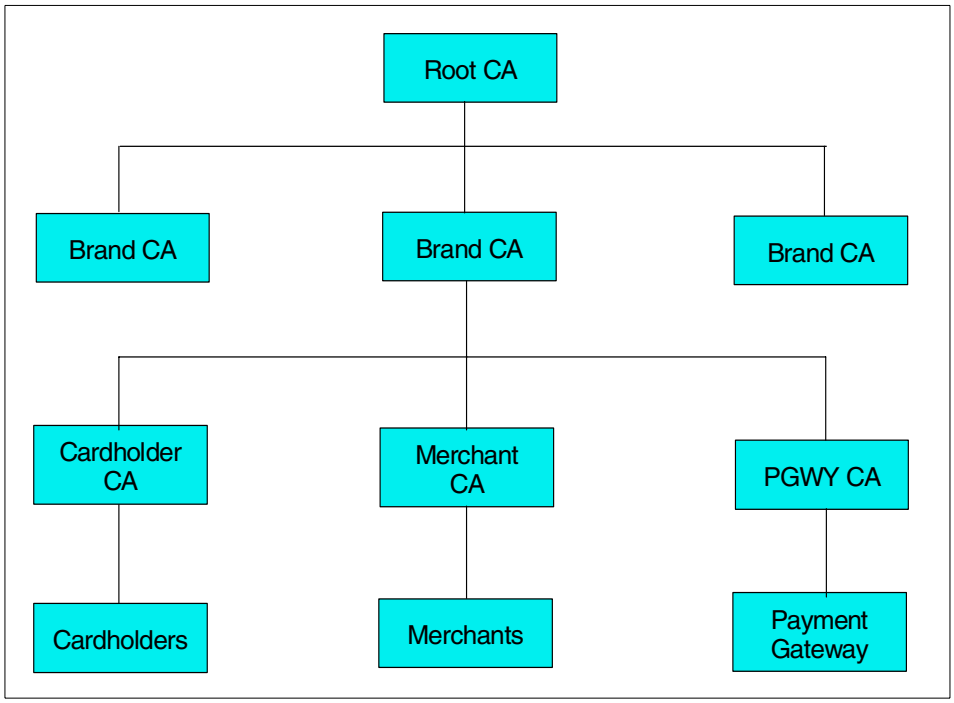


Figure 5. Brand CA

SET represents and verifies each of these trusted relationships by issuing digital certificates. The SET root CA issues digital certificates to brands that meet SETCo's brand requirements. A *brand* is an entity that issues payment cards with its own distinct logo. Once a brand issues its certificates, the brand can then sign the certificates and issue them to cardholder CAs, merchant CAs, and payment gateway CAs.

To use SET as a merchant, you must register with a certificate authority (CA) before you can receive SET payment instructions from cardholders or process SET transactions through a payment gateway. You also need a copy of the registration form from your financial institution. Your software must identify the acquirer to the CA. For the latest information about CA for SET, go to the Web site at: `http://www.setco.org`

1.4 Payment process without a wallet

The certificate needed for the cardholder may be an inhibitor to implement the SET network. Installing the wallet software on a PC and receiving the certificate from a CA is a cumbersome process for cardholders. To use the

SET protocol in a more practical environment, IBM came up with the idea of not using the wallet and the cardholder certificate. This method lacks one security feature compared to a pure SET method, which is the cardholder identification. The merchant basically trades the cardholder identification for business capability. This method is called Merchant Originated Payment (MOP). The MOP is the IBM proprietary implementation. SETCo extended the SET protocol to accommodate this scenario. It is called Merchant Initiated Authorization (MIA). The MOP and MIA extension permit a merchant to use SET messages for authorization and capture for orders that were placed by the cardholder using a transmission method other than SET. Since these messages will not carry a cardholder certificate, the gateway certificate must indicate that non-certificate purchases are supported.

In a typical SET implementation, the merchant customizes order forms to allow shoppers to request the payment initiation message, also known as the *wakeup message*, from a merchant server, such as the Net.Commerce server. When the shopper's Web browser receives this payment initiation message, the browser launches a wallet application, such as the IBM Consumer Wallet, which must already be installed on the shopper's machine and certified by a CA. Shoppers specify payment card information and select the payment card to use from the wallet application window.

With MOP or MIA, the wallet is not necessary, so the cardholder does not need a certificate. In the Figure 3 on page 6, the SET certificate flow (represented as 2) is not needed. Also, in the same figure, the CA (represented as 1) can be any CA, but not a SET certified CA. Merchant Originated Payment is an IBM extension based on the SET protocol that allows the merchant to receive credit card information through any mechanism, such as over the phone or through the store's online order forms. If the shopper submits credit card information through the store's online order form, when the form is submitted, the credit card information is encrypted using SSL. It is then passed to the acquirer, using regular SET messages, through the IBM Payment Gateway, or a similar method that supports MOP. However, Merchant Originated Payment does not perform cardholder authentication the way that a wallet application does. It is an attractive payment method because shoppers do not need to download and install the wallet software.

Payment flows which involve certificates for all three parties to the transaction (that is, with a wallet) are often referred to as "3KP". Payments which do not involve a cardholder certificate (without a wallet) are referred to as "2KP". 2KP is also known as a "certless purchase" (non-certificate purchase). All merchant initiated authorizations (MIA or MOP) are considered 2KP flows.

1.4.1 MOP and MIA transactions

Since the SET protocol starts from the merchant server, you need to change the way to process the transactions and APIs compared to the process with a wallet. Figure 4 on page 7 shows a typical flow of the 3KP. The 2KP flow looks like the example shown in Figure 6. The process is explained in the list that follows.

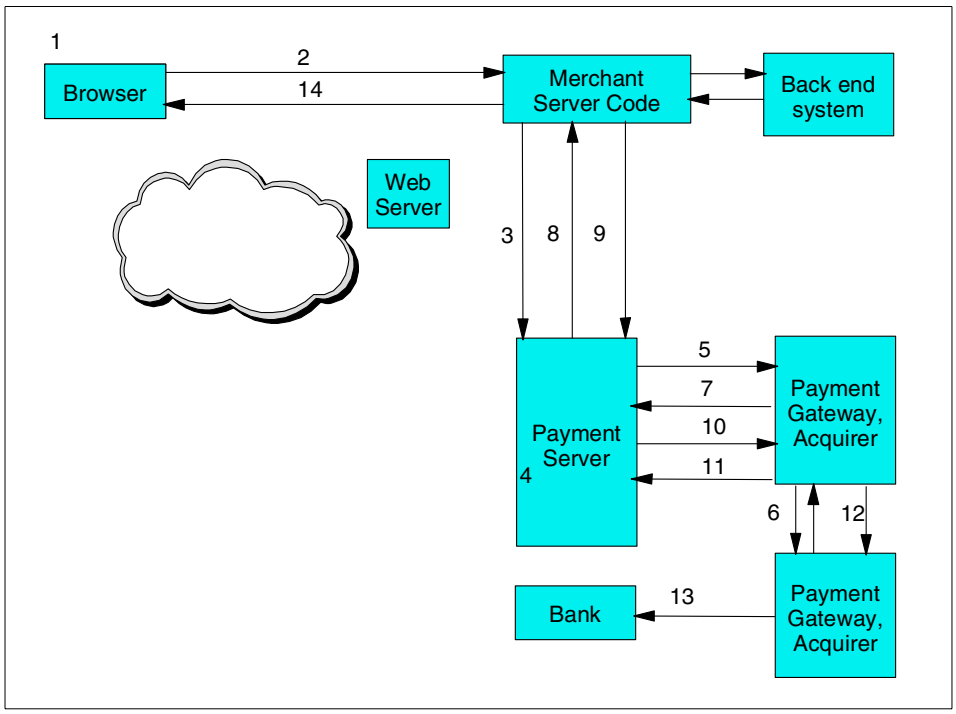


Figure 6. MIA or MOP transactions

1. A cardholder decides to make a purchase.
2. When the cardholder clicks the Buy button, a command is sent to the merchant server.
3. The merchant server calls the Payment Server etAcceptPayment() API.
4. The Payment Server checks to see if authorization should be done at this point. For example, if the merchant's acquirer is closed on that day, the process may be delayed until the acquirer is available.
5. When authorization can be done, the server generates an Authorization Request (`AuthReq`) sends it to the acquirer, and waits for an Authorization Response message (`AuthRes`).

Note

This scenario shows the automatic approval process that is achieved by the etReceivePayment() API. You can also approve manually with the etApprove() API. You can see the automatic and manual approve processes in more detail in 3.3, “Sample API sources” on page 76.

6. The acquirer software or Payment Gateway receives the request. Using a normal back-end network or other communication channels, the acquiring institution contacts the cardholder’s issuing institution. It checks that the payment card is valid and that the cardholder has sufficient funds or credit to make the purchase.
7. The `AuthRes` message is received by the Payment Server and processed. Information is stored in the database for record-keeping and further order processing.
8. The merchant can now fulfill the order.
9. When the goods are shipped, the merchant requests payment by calling the etDeposit() API.
10. The Payment Server now begins the capture process by sending a Capture Request to the Payment Gateway. *Capture* is the transfer of funds from the cardholder’s issuing institution to the merchant’s acquirer and onward to the merchant.
11. The acquirer software receives the capture request and sends it a capture response message.
12. The acquirer uses the closed (back-end) network to contact the cardholder’s issuing institution and requests the transfer of payment.
13. The acquirer deposits the payment to the merchant’s bank account.
14. The merchant server sends the confirmation to the cardholder.

Chapter 2. Planning for SET Secure Electronic Transaction

This chapter contains the information you must know to be able to use different payment methods. The major advantage of SET over existing security systems is the addition of digital certificates that associate the cardholder and merchant with their financial institutions and the respective payment brands, for example, MasterCard, Visa, and so on. Digital certificates reinforce existing trusted business relationships and protect against fraud at a level that existing systems do not. For example, SSL provides security in the transmission of sensitive data, but does not guarantee the identity of the parties involved in the transaction.

Note

As mentioned in 1.4, “Payment process without a wallet” on page 10, when you use MOP or MIA in SET, you lose the security capability to some degree.

2.1 Payment Server planning tables

The Payment Server has three categories of information, which are shown in Figure 7 on page 16. The first category contains information provided by your acquirer to set up the communication to them. The second category contains information that is given by the certificate authority for the certification. The third category includes information for your Payment Server’s unique configuration. You need to set up the third category of information by yourself. In the Payment Server configuration steps, you can simply use the values in the tables shown on the following pages.

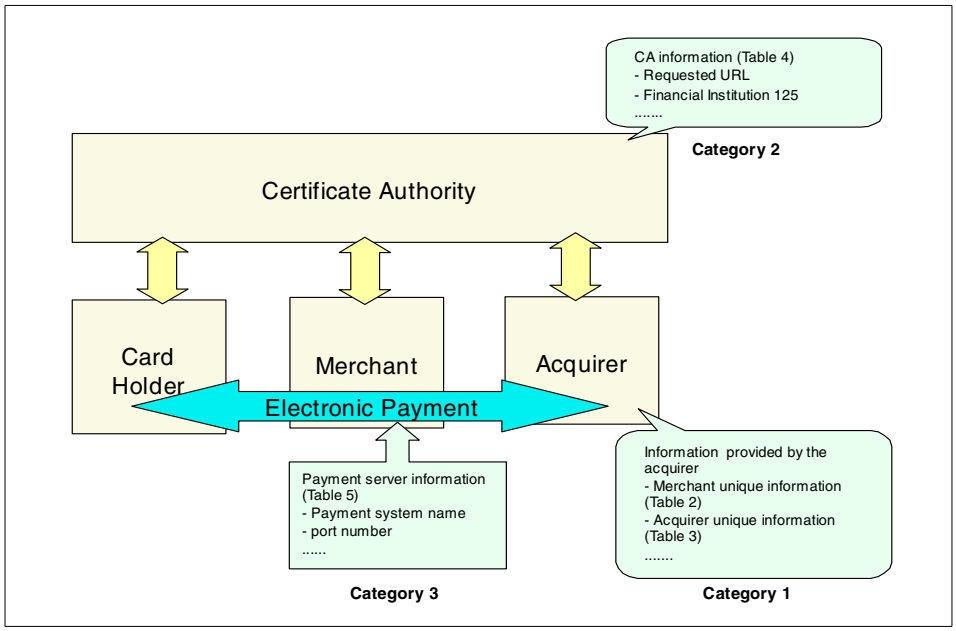


Figure 7. The group for the Payment Server configuration information

There are five tables in this section. Table 1 shows the configuration check list. Table 2 through Table 5 on page 20 show the configuration values and their brief descriptions. The field numbers in the tables are placed there to match the information in the field with other references in this book. These numbers will not be seen on the screens of the Payment Server configuration.

Table 1. Payment Server check table

Information needed to install and configure the Payment Server	Answer
Is the information in Table 2, Table 3, Table 4, and Table 5 filled in?	
Is the firewall configuration updated?	
Is the Payment Server installed and created?	

2.1.1 Information provided by the acquirer

You should obtain the information in Table 2 from your acquirer, except for the merchant number. This information is used to configure your Payment Server.

Table 2. User dependant information provided by the acquirer

Field Name and Number		Value	Description
Merchant number	01		**Required field** — Numeric character string, 1 to 9 characters, Valid values: 0-999999999. This field can only be chosen from the list of merchants previously configured as payment systems.
Account number	02		**Required field** — Numeric character string. Obtain this value from your acquirer.
Signing brand ID	03		**Required field** — Text string, 1 to 40 characters. The Brand ID from the certificate of the signing brand. **Note**: A merchant may support multiple brands, but only one signing brand ID per acquirer is allowed.
SET profile	04		**Required field** — Integer: The numeric representation for the acquirer profile that the merchant is using for batch transactions. Obtain this value initially from your acquirer. If you want to know more information about the SET Profile, refer to the Web address: `http://www.ibm.com/software/webservers/paymgr/support/acqprofile.html` If you have the latest PTFs installed on your AS/400 system, you can also find the same information at: `http://as400:2001/QIBM/PymSvr/Admin/readme.ndm/readme` (as400 is your AS/400 TCP/IP host name).
Brand ID	05		**Required field** — Text string 1 to 40 characters long and case sensitive. Payment card brand. Obtain this value from your acquirer.
Acquirer bank ID (BIN)	06		**Required field** — Numeric string, 1 to 6 characters long. Unique identifier for this acquirer for the brand. Obtain this value from your acquirer.
Acquirer business ID	07		**Required field string** — 1 to 32 characters long. The business identification number of this acquirer. Obtain this value from your acquirer.
Merchant ID	08		**Required field** — Alpha-numeric (if PTF SF57438 is installed. If the PTF is not installed, only numeric is allowed.) character string, 1 to 30 characters long. The SET Merchant ID. Obtain this value from your acquirer.
Have certificate	09		**Required field** — Default=No. If set to No, the Payment Gateway's certificate is requested from the acquirer when the Payment Server is started. The Payment Server then changes the setting to Yes.

Terminal ID	10		**Optional** to SET message, but may be required by the acquirer; numeric character string.
Chain number	11		**Optional** to SET message, but may be required by the acquirer; numeric character string.
Store number	12		**Optional** to SET message, but may be required by the acquirer; numeric character string.
Agent number	13		**Optional** to SET message, but may be required by the acquirer; numeric character string.

You need the following information shown in Table 3 to complete the acquirer information on your Payment Server.

Table 3. Acquirer unique information

Field name and number		Value	Description
Start time	14		**Optional field** — Expressed as the number of minutes after midnight in the merchant's local time; 0=midnight: Time of day the acquirer opens for business.
Stop Time	15		**Optional field** — Expressed as the number of minutes after midnight in the merchant's local time; 0=midnight: Time of day the acquirer closes for business.
Gateway host name	16		**Required field** — Text or numeric string, 1 to 40 characters: Internet host name or IP address of this Payment Gateway.
Gateway port	17		**Required field** — Numeric character string, default=8888: Port number where this acquirer accepts messages from this particular merchant.
Gateway protocol	18		**Required field** — HTTP must be in uppercase: Protocol used by the acquirer (HTTP).
Maximum number of connections	19		**Required field** — Numeric character string, > or = to 1; default=1: Maximum number of sockets or connections on the channel.
Read timeout	20		**Required field** — Number in seconds, 1 to 65535; default=30: Number of seconds to wait for a Payment Gateway to read a SET message from a socket before timing out on a socket. It is also the time between replays.
Number of immediate retries	21		**Required field** — Numeric character string, 0 to 65535, default=0: Maximum number of replays in a row before waiting for the interval specified on Delayed retry interval.

Field name and number		Value	Description
Delayed retry interval	22		**Required field** — Number of minutes, 0-65535; default=0: Number of minutes to wait after the maximum number of replays has been reached before trying again.
Confirm delay code	23		**Optional field** — Numeric character string, default=blank: Set this field to blank unless your acquirer gives you a value.
Confirm delay time	24		**Optional field** — Number in minutes; default=blank: Set this field to blank unless your acquirer gives you a value.
Pending delay code	25		**Optional field** — Numeric character string; default=blank: Set this field to blank unless your acquirer gives you a value.
Pending delay time	26		**Optional field** — Number in minutes; default=blank: Set this field to blank unless your acquirer gives you a value.

2.1.2 CA information

You need the information in Table 4 to request the Payment Server SET certificate.

Table 4. Information provided by the Certificate Authority

Field name and number		Value	Description
Request URL	27		**Required field** — Case sensitive; provided by the acquirer.
Root hash	28		**Required field** —A string of characters that allows you to verify the correctness of the Root Public Key you are using; provided by the acquirer.
Financial Institution 125	29		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.
Merchant Name	30		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.
Merchant City	31		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.
Merchant State	32		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.
Merchant Postal Code	33		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.

Field name and number		Value	Description
Merchant Country	34		**Required field** — Questions depending on the CA being used. Review the information you type on this screen. Errors in this information will result in errors on your certificate.

2.1.3 Payment Server information

You need the information in Table 5 to configure the Payment Server on your AS/400 system.

Table 5. Payment Server information

Field name and number		Value	Description
Payment System name	35	SET	**Required field** — Type of payment system being used. Only valid value is SET.
Administration port number	36		**Required field** — A valid, available TCP port where the Payment Server accepts Administration API messages. Valid values: 1000 to 65535. The default value is 8614.
Payment port number	37		**Required field** — A valid, available TCP port where the Payment Server accepts Payment API messages. Valid values: 1000 to 65535. The default value is 8611.
Error logging port number	38		**Required field** — A valid, available TCP port where the Payment Server accepts error logging and trace messages. Internal use only. Valid values: 1000 to 65535. The default value is 8617.
Log path	39		**Optional field** — The path name in the root file system where all trace, error, and dump files are written. Ensure the log path exists and that the Payment Server has read/write access to this path. The default value is /QIBM/UserData/PymSvr/QUSRPYMSVR/Logs.
SET payment port number	40		**Optional field** — A valid, available TCP port where the Payment Server accepts SET payment messages. Valid values: 1000 to 65535. The default value is 8620.
SET inquiry port number	41		**Required field** — A valid, available port where the Payment Server accepts SET inquiry messages. Valid values: 1000 to 65535. The default value is 8621.

After you gather the necessary information using Table 2 on page 17 through Table 5, you can configure the Payment Server easily and flawlessly. Use the field numbers to Ensure that you transfer the information between the correct fields in the tables.

As you read the following sections, note the numbers that appear in reverse bold type. These numbers indicate the numbers (in the second column) from the planning tables (Table 2, Table 3, Table 4, and Table 5).

2.2 Installing and configuring the Payment Server

Fundamental to Payment Server configuration is the notion that certain fields in the Payment Server configuration tables must exactly match the values in both your merchant certificate and in your acquirer's payment gateway certificate. Both of these certificates are issued by a certificate authority (CA). To have the best chance of successfully configuring your Payment Server the first time, you should not obtain a merchant certificate or configure the Payment Server until your acquirer has received certificates and has agreed with the CA on the merchant and acquirer Business Identification Numbers (BINs), Acquirer Business ID, and Brand IDs.

2.2.1 Before you start

If you did not already write down the following data on the worksheet in 2.1, “Payment Server planning tables” on page 15, be sure to collect the following data about the merchants, the brands the merchants accept, and the acquirers who are providing banking services for the merchants (Table 6).

Table 6. Minimum configuration information

	Field name	Description
1	Merchant Number	Select a unique 1 to 9-digit number for each merchant for whom you are creating a profile.
2	Acquirer BIN	A unique identifier for the acquirer for the brand being configured. Obtain this value from your acquirer.
3	Acquirer Business ID	The Business Identification Number of the acquirer. It is specified as a unique identifier for the acquirer's Payment Gateway. Obtain this value from your acquirer.
4	Merchant ID	The SET merchant ID. Obtain this value from your acquirer.
5	Optional Acquirer Information	You need the following information if it is required by your acquirer: -Terminal ID -Chain Number -Store Number -Agent Number The acquirer will provide this information if it is required.

	Field name	Description
6	Acquirer Business Hours and Off-Days	Obtain a list of any hours of the day, days of the week, or holidays when the acquirer is not open for business.
7	Gateway Information	You need to know the IP addresses or host names for any payment gateways that will communicate with the Payment Server and the port numbers the gateways will use to listen for transactions.

2.2.1.1 Hardware requirements

To run the Payment Server, you must meet the following hardware requirements:

- Any AS/400 RISC system that is capable of running OS/400 with a CPW of 45 or more, and 96 MB of RAM

  You will need additional space for the databases where the Payment Server configuration and transaction data is stored. The transaction data in particular may require a significant amount of space, if the transaction rate is high or if you do not frequently prune (delete old records) the transaction database or the logs.

  If more than one language feature is installed, then additional space is required.

- Communications adapter

  Any communications adapters that run TCP/IP are supported.

> **Note**
>
> Payment Server for AS/400 does not support hardware cryptography.

2.2.1.2 Software requirements

To run the Payment Server, you must meet the following software requirements:

- OS/400 (5769SS1) Version 4 Release 3, or later
- AS/400 Developer Kit for Java for AS/400 (5769JV1) Version 4 Release 3, or later
- IBM HTTP Server for AS/400 (5769DG1) Version 4 Release 3, or later

Note

JV1 and DG1 both come with OS/400 but are installed separately.

- QShell Interpreter, OS/400 option 30 of Version 4 Release 3, or later

2.2.2 Installing a Payment Server

Payment Server is installed in the standard manner using the Restore LIC Program (RSTLICPGM) command. The objects created during installation are:

- QPYMSVR library (the library contains programs, service programs, commands, a message file, and a header file)
- QPYMSVR user profile
- /QIBM/ProdData/HTTP/Protect/PymSvr directory and subdirectories
- /QIBM/ProdData/PymSvr directory and subdirectories

The product can be deleted by using the Delete LIC Program (DLTLICPGM) command. The QPYMSVR user profile is not deleted.

2.2.3 Time and date system values

The Payment Server checks the date, time, and time zone when you use it for online transactions. It is important that the system values for the date, time, and time zone in your AS/400 system are correctly set. The Payment Server also checks the date, time, and time zone that is sent to it from eWallet, and compares it with its own values.

To display your date- and time-related system values, complete these steps:

1. On an AS/400 command line, type:

   ```
   DSPSYSVAL SYSVAL(QDATE)
   ```

   Press Enter to see the system date value.

   If the value is incorrect, you have to change it. On an AS/400 command line, type:

   ```
   CHGSYSVAL SYSVAL(QDATE) VALUE('mm/dd/yy')
   ```

   In this statement, mm/dd/yy occurs in the command, where mm = month, dd = day, and yy = year.

Note

The format of the date field is not the same in all countries. Verify your format before you change it.

2. On an AS/400 command line, type:

   ```
   DSPSYSVAL SYSVAL(QTIME)
   ```

   Press Enter to see the system time. If the value is incorrect, you have to change it. On an AS/400 command line, type:

   ```
   CHGSYSVAL SYSVAL(QTIME) VALUE('hh:mm:ss')
   ```

   In this statement, `hh` = hour, `mm` = minute, and `ss` = second.

3. On an AS/400 command line, type:

   ```
   DSPSYSVAL SYSVAL(QUTCOFFSET)
   ```

   Press Enter to see the Coordinated Universal Time Offset system value. The value specifies the difference in hours and minutes between the Universal Time, Coordinated (UTC), also known as Greenwich Mean Time (GMT), and the current system time.

   If the value is incorrect, you must change it. On an AS/400 command line, type:

   ```
   CHGSYSVAL SYSVAL(QUTCOFFSET) VALUE('-hh:mm')
   ```

   In this statement, the symbol "-" can be either "+" or "-" depending on where your time zone is in relation with GMT. In this statement, `hh` = hour and `mm` = minute. Some examples are:
   "-6:00" for Central Standard Time
   "+9:00" for Japan Standard Time

2.2.4 Creating a Payment Server instance

You have to create a Payment Server instance before you can request a digital certificate from a CA. To do so, follow these steps:

1. From the AS/400 Task page, as shown in Figure 8, click **IBM Payment Server for AS/400**.

Figure 8. AS/400 Tasks page

2. On the IBM Payment Server for AS/400 page, click **Administration** in the left-hand frame to display an extended list of server tasks. This takes you to the display shown in Figure 9 on page 26.

Figure 9. Payment Server for AS/400

3. Click on **Create** to go to the Create Payment Server page.
4. On the Create Payment Server page, type a key database password (Figure 10), and click **Create**.

> **Note**
>
> If you lose your password, there is no way to recover your certificates. You will need to request them again. This is different than other AS/400 passwords, where the security officer can reset the password for you and nothing is lost. We recommend that you back up your data just prior to changing the password.

Write down the password, and save it in a secure place. You will need it when you request a digital certificate for your SET merchant, or every time you start your Payment Server.

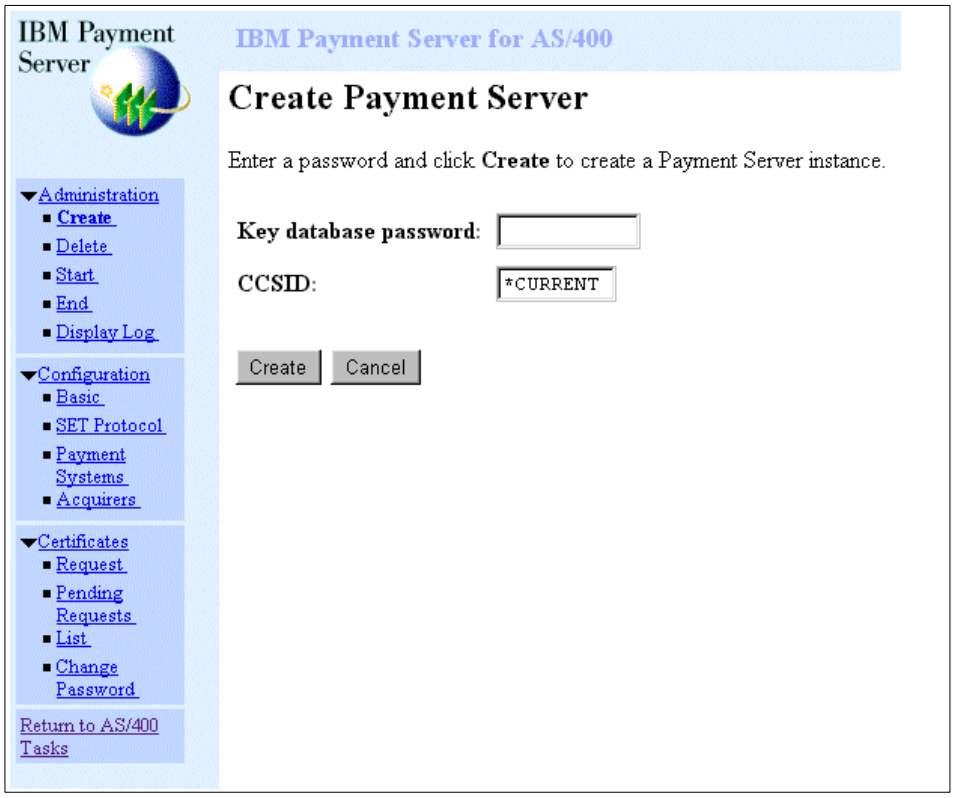


Figure 10. Payment Server administration create page

The Payment Server is created (Figure 11 on page 28).

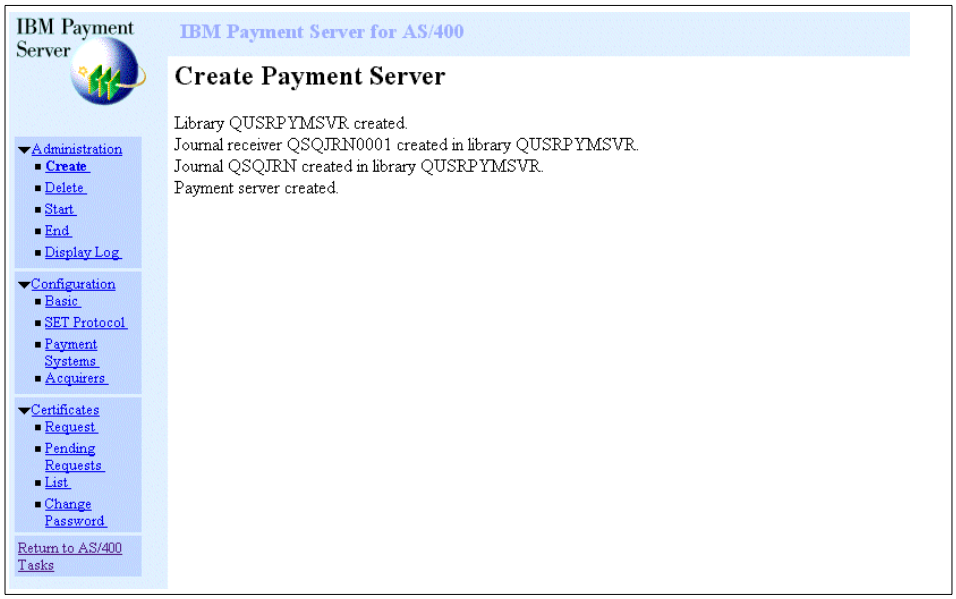


Figure 11. Payment Server administration creation done page

Note

Creating the Payment Server can be performed via the command line using the Create Payment Server (`CRTPYMSVR`) CL command.

The objects created during the creation of Payment Server instance are:

- Transaction and operations tables
- Configuration tables
- Trace directory

2.2.4.1 Transaction and operations tables

Both the transaction and operations tables that store order and transaction information, and the configuration tables, are placed in the QUSRPYMSVR library. Table 7 shows a brief description of the transaction and operations tables. For more detailed information about the transaction and operation

tables, please refer to the manual *IBM Payment Server for AS/400, Version 1.2 Programmer's Guide and Reference,* SC31-8697.

Table 7. Payment Server generic transaction data

Table name	Description
ETORDER	Generic information about each order the Payment Server processes.
ETPAYMENT	Generic information about each payment the Payment Server processes.
ETCREDIT	Generic information about each credit the Payment Server processes.
ETBATCH	Information about batches that were open and closed.
ETBATxxxx	Details about individual items in each batch.

2.2.4.2 Configuration tables

The configuration tables store the Payment Server configuration information. You do not need to manipulate them directly, because all tables are updated through the AS/400 Task browser interface.

In this section, we describe the role of the major tables to understand how the Payment Server works. The configuration tables can be further split into two categories: the generic tables and the protocol specific tables. The generic tables configure the values the Payment Server needs, independent from the payment protocol being used. The protocol-specific tables contain the information specific to a particular payment protocol. For Payment Server V1.2, the tables specific to the SET protocol have been included.

Table 8 lists the generic tables. Table 9 on page 30 lists the SET-specific tables.

Table 8. Generic configuration profiles

Payment Server profile	Corresponding database table name	Description
Payment Server information	ETILL0000x	Contains configuration information about the Payment Server. The table also sets API and error log ports, log file path, pass through data for user exits. ETILL0000x table is created for each Payment Server. The AS/400 system supports only one Payment Server at the same time.

Payment Server profile	Corresponding database table name	Description
Payment System Configuration	ETPAY0000x	Each payment system is identified by the merchant number and the payment type. One ETPAY0000x table is created for each payment system.

Table 9. SET-specific configuration profiles

Payment Server profile	Corresponding database table name	Description
SET Configuration	ETSETCFG	Contains information specific to the SET protocol and ports that will be used for sending and receiving data. One ETSETCFG table is created for each Payment Server that uses the SET protocol cassette.
Acquirer Configuration	ETACQCFG	SET protocol-specific. It contains information about a specific acquirer and the acquirer's payment gateway. Each profile describes one acquirer-merchant relationship and includes payment processing options, business hours, and communication settings. One Acquirer profile is created for each acquirer. There can be multiple Acquirer Configuration profiles for each merchant because a merchant may use several acquirers.
Brand Configuration	ETBRANDCFG	The Brand profile is specific to the SET protocol. One Brand profile is created for each brand used by a particular merchant. The Brand profile corresponds to the ETBRANDCFG table in the database.
Acquirer Off-Days Configuration	ETACQ0000x	The Acquirer Off-Days profile is specific to the SET protocol. The Acquirer Off-Days Configuration profile (ETACQ0000x) indicates when an acquirer is closed. Because the acquirer may have off-days, this profile is optional. You must enter one profile for each off-day for a particular acquirer. For example, if an acquirer is closed every December 25 and is not available Sundays, you would configure two profiles for that acquirer. Note that business hours for an acquirer are handled in the Acquirer profile. Special closings, such as a particular day of the week or a holiday, are handled in the Off-Days profile.

Each row in the table contains profile-specific information. For example, when you have three Brand profiles, they are stored as three rows in the ETBRANDCFG table. Most of the fields are not initialized with default values, so you need to configure all required fields before you connect to an acquirer system and successfully complete transaction processing.

The profiles for the payment system, the acquirer, and the brands of cards supported are all closely related. You create a set of each profile for a particular merchant. The numeric merchant name you selected for the merchant associates the profiles with a particular merchant. The fields in the Acquirer and Brand profiles must match the information in the merchant certificates you request for each of the card brands.

2.2.4.3 Trace directory

The /QIBM/UserData/PymSvr/QUSRPYMSVR/Logs directory is created to contain the trace and dump information of the Payment Server process. The data is used to analyze and isolate a problem when it occurs.

2.2.5 Configuration scenarios

When you use an AS/400 system for the Payment Server, you do not need to create the tables for the Payment Server that are described in 2.2.4, “Creating a Payment Server instance” on page 24. The tables are created when the Payment Server is created. You can use the AS/400 Task browser interface to create those tables as shown in Figure 12 on page 32.

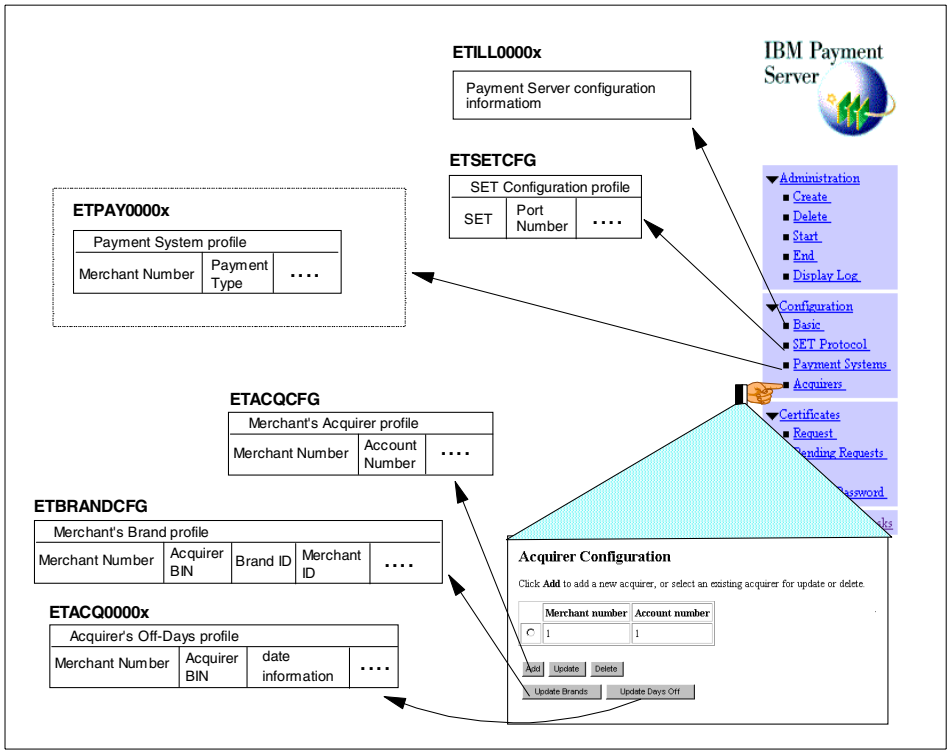


Figure 12. Relationship of the Payment Server tables to the configuration interface

This section shows some configuration scenarios based on your merchant configurations and your acquirer requirements:

- **Scenario 1**: Minimum configuration

  At minimum, your Payment Server connects to one acquirer and supports one brand (credit card). You create and populate each of the following items for this requirement:

  – One Payment Server information profile
  – One SET configuration profile
  – One Payment System profile
  – One acquirer profile
  – One brand profile

- **Scenario 2**: One merchant (multiple brands)

  For a particular merchant, you configure more than one card brand that the merchant accepts and one acquirer with which the merchant communicates. Suppose you are a merchant who accepts Visa and

MasterCard and you use a single bank as your acquirer. You would configure the following items:

  – One Payment Server information profile
  – One SET configuration profile
  – One Payment System profile
  – One acquirer profile
  – Two brand profiles (one for Visa, one for MasterCard)

- **Scenario 3**: Two merchants (multiple brands)

  If more than one merchant uses a particular Payment Server, you must configure separate Payment System profile for each merchant. Suppose you have the following configuration: merchant 1 accepts two brands and merchant 2 accepts one brand. Both merchants use the same acquirer for all brands they accept. You configure the following profiles:

  – One Payment Server information profile
  – One SET configuration profile
  – Two Payment System profiles (one for each merchant)
  – Two acquirer profiles (one for each merchant-acquirer relationship)
  – Three brand profiles (one for each merchant-brand relationship)

- **Scenario 4**: One merchant (one acquirer that closes for holidays)

  Acquirers may have “off” days when they are not open for business. The Acquirer Off-Days profile handles these situations. Suppose an acquirer in the United States closes for the Fourth of July and Christmas holidays. You would configure the following profiles:

  – One Payment Server information profile
  – One SET configuration profile
  – One Payment System profile
  – One acquirer profile
  – One brand profile
  – Two Acquirer Off-Days profiles (one to handle July 4, one to handle December 25)

In the following section, we describe the configuration of scenario 1. In scenario 1, we configure the following profiles:

- One Payment Server information profile
- One SET configuration profile
- One Payment System profile
- One acquirer profile
- One brand profile

On the AS/400 Payment Server, you configure the profiles through the AS/400 Task browser interface using the following configurations:

1. *Basic* configuration for creating the Payment Server Information profile.
2. *SET protocol* configuration for creating the SET configuration profile.
3. *Payment System* configuration for creating the Payment System profile and the Cassette Configuration profiles.
4. *Acquirer* configuration for creating a merchant's Acquirer profiles and a merchant's Brand profile.

Each step is described in more detail in the following sections.

2.2.6 Basic configuration of the Payment Server

To use your Payment Server, you have to complete its basic configuration. To do this, follow these steps:

1. From the IBM Payment Server for AS/400 page, click **Configuration** in the left-hand frame to display an extended list of server tasks.
2. Click on **Basic** from the list to perform the basic configuration.
3. On the Basic Configuration page (Figure 13), only make changes if the ports are used by other applications.

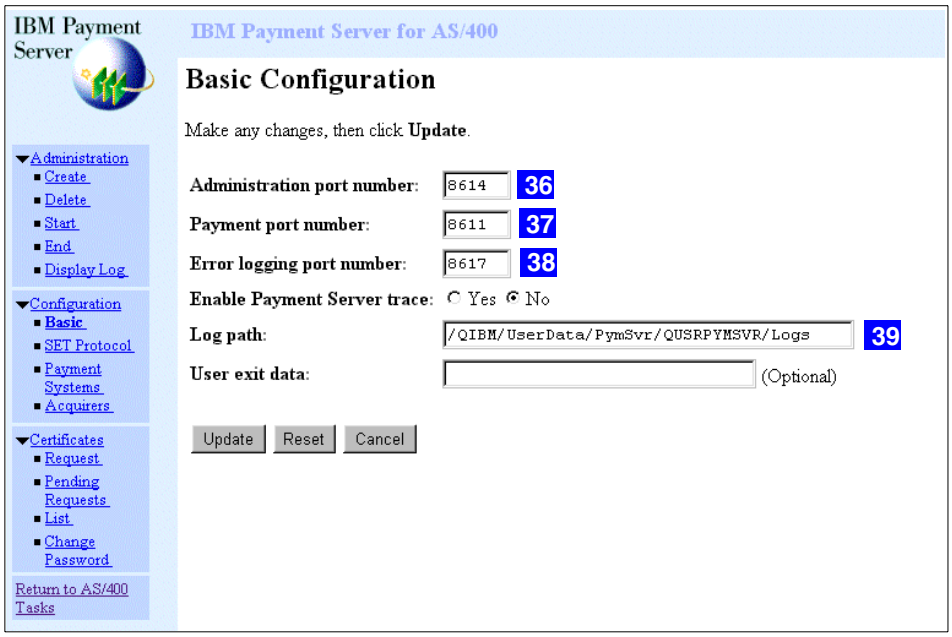


Figure 13. Payment Server configuration basic page

4. Click **Update**.

   You receive a message, which indicates that the configuration update is successful (Figure 14).

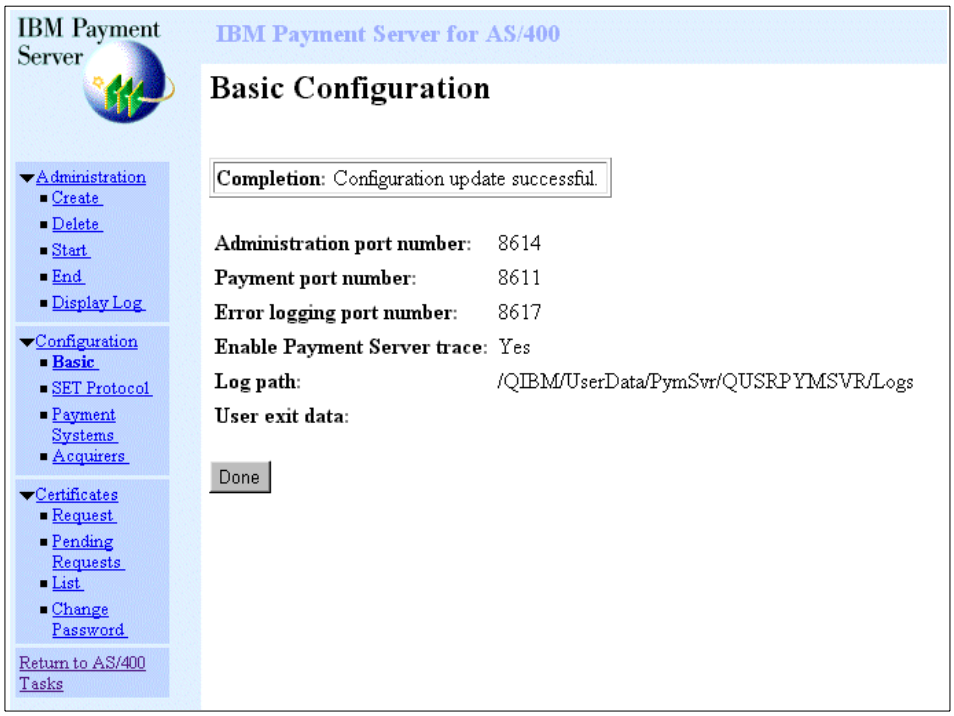


Figure 14. Payment Server basic configuration complete page

2.2.7 SET Protocol configuration of the Payment Server

The next step in the configuration of the Payment Server is to configure the SET Protocol. Complete the following steps:

1. Click **SET Protocol** from the list to perform the SET Protocol configuration.
2. On the SET Protocol Configuration page (Figure 15 on page 36), only make changes if your acquirer tells you to make them.

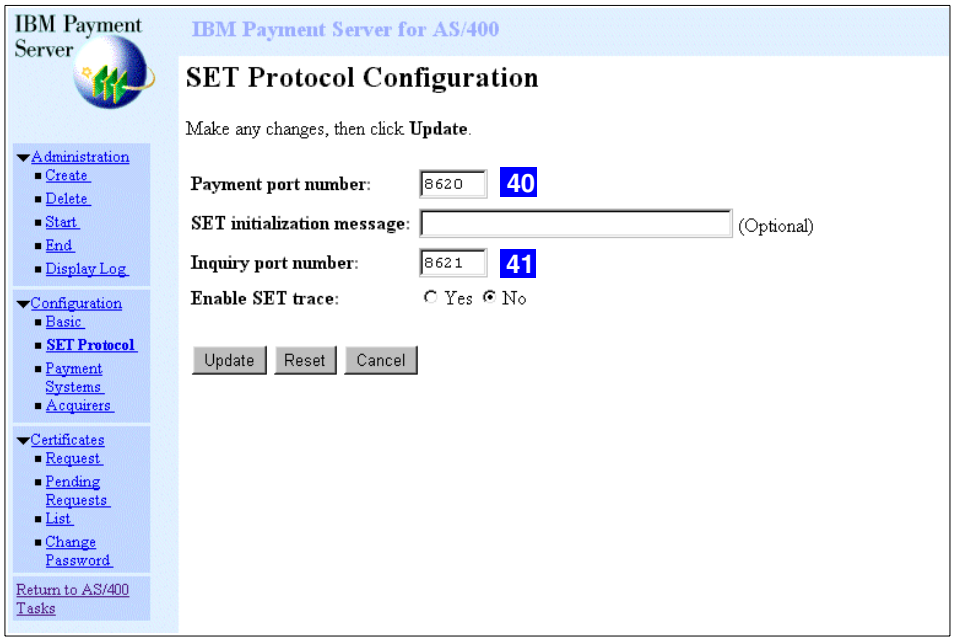


Figure 15. Payment Server SET protocol configuration page

3. Click **Update**.

   You receive a message, indicating that the configuration update is successful (Figure 16).

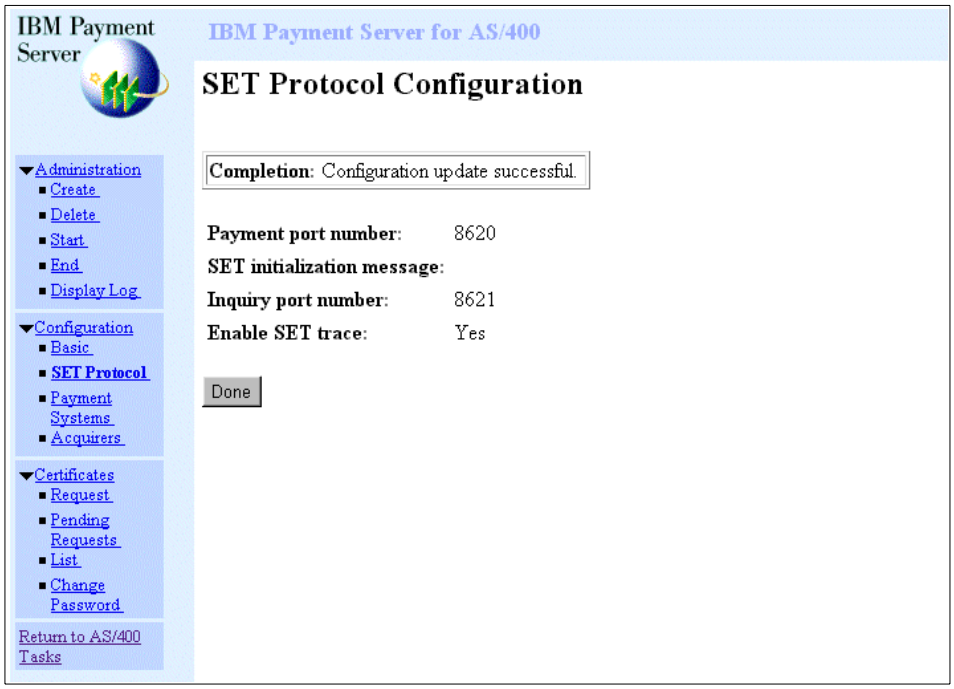


Figure 16. Payment Server SET protocol configuration complete page

2.2.8 Payment System configuration of the Payment Server

The next step in the configuration of the Payment Server is to configure the Payment System. Follow these steps:

1. Click on **Payment Systems** from the list to perform the Payment System configuration.
2. On the Payment System Configuration page (Figure 17 on page 38), click **Add**.

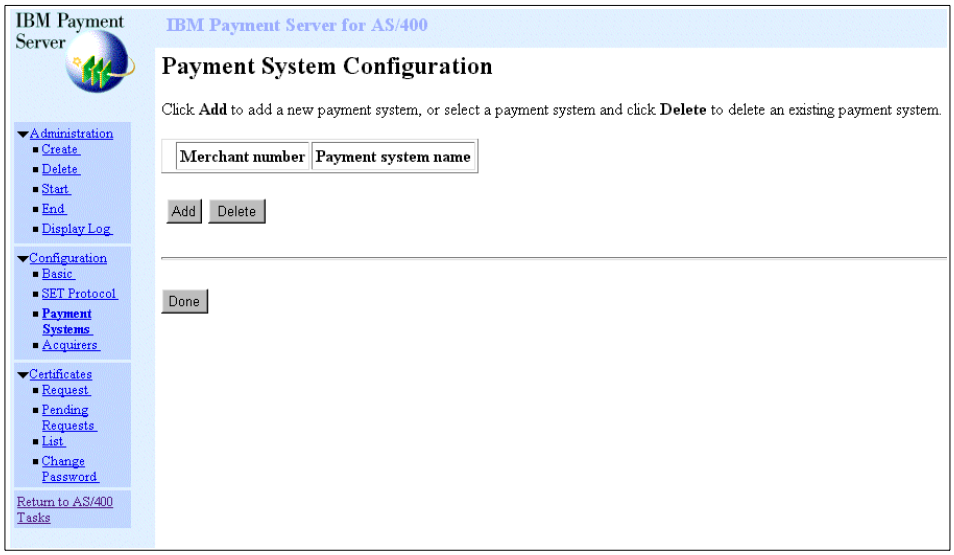


Figure 17. Payment Server payment system configuration page

3. On the Payment System Configuration page (Figure 18), add your merchant number. If you only have one shop, enter a number, such as 1. You will use this merchant number later during the Acquirer configuration, as described in 2.2.9, “Acquirer configuration of the Payment Server” on page 40.

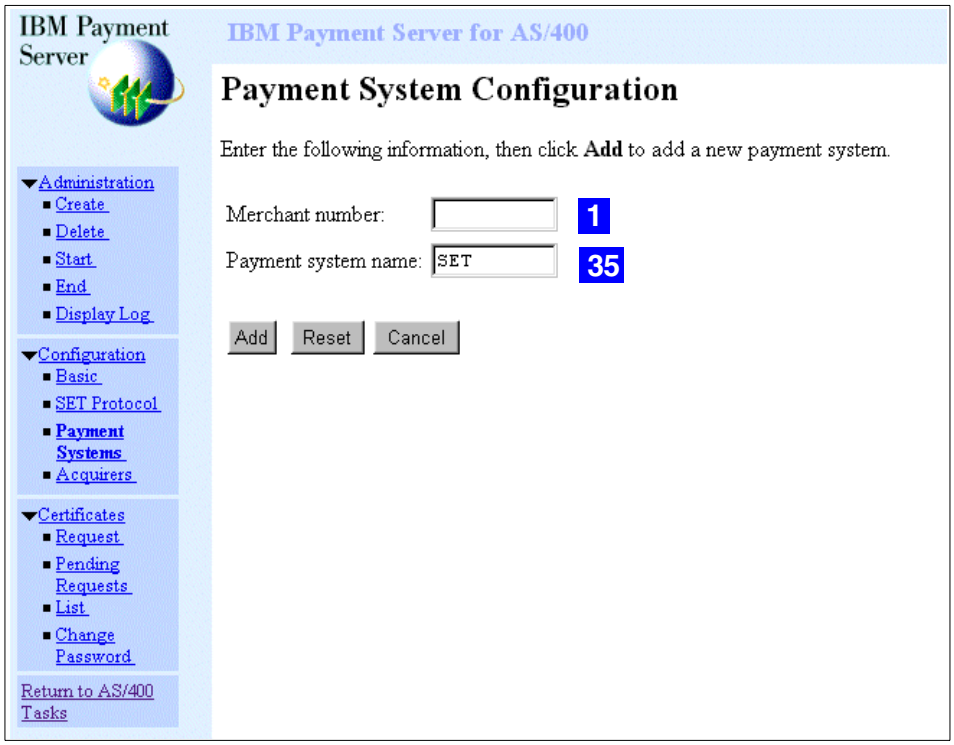


Figure 18. Payment Server payment system configuration form page

4. Click **Add.**

   You receive a message that the configuration update is successful (Figure 19 on page 40).

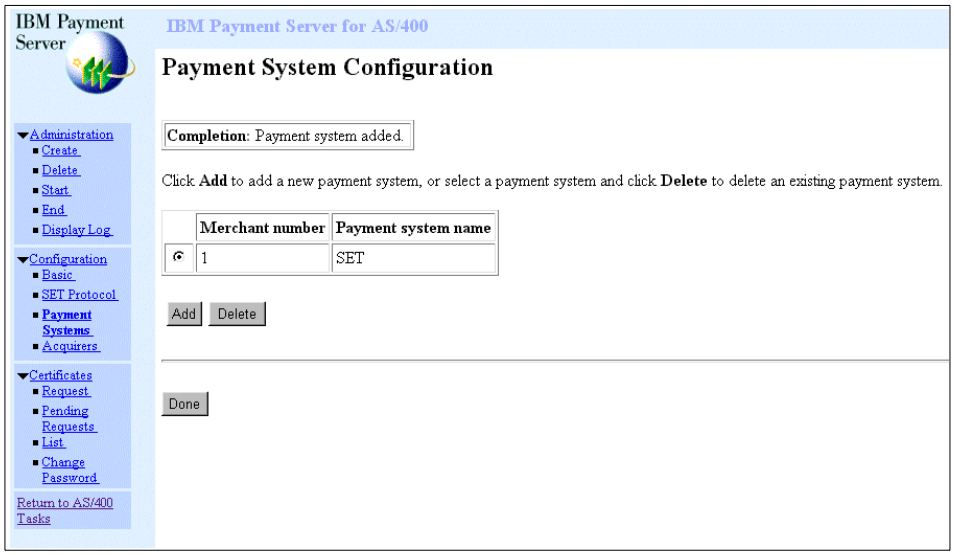


Figure 19. Payment Server payment system configuration complete page

2.2.9 Acquirer configuration of the Payment Server

The next step in the configuration of the Payment Server is to configure the Acquirers. Follow these steps:

1. Click **Acquirers** from the list to perform the Acquirers configuration.
2. On the Acquirer Configuration page (Figure 20), click **Add**.

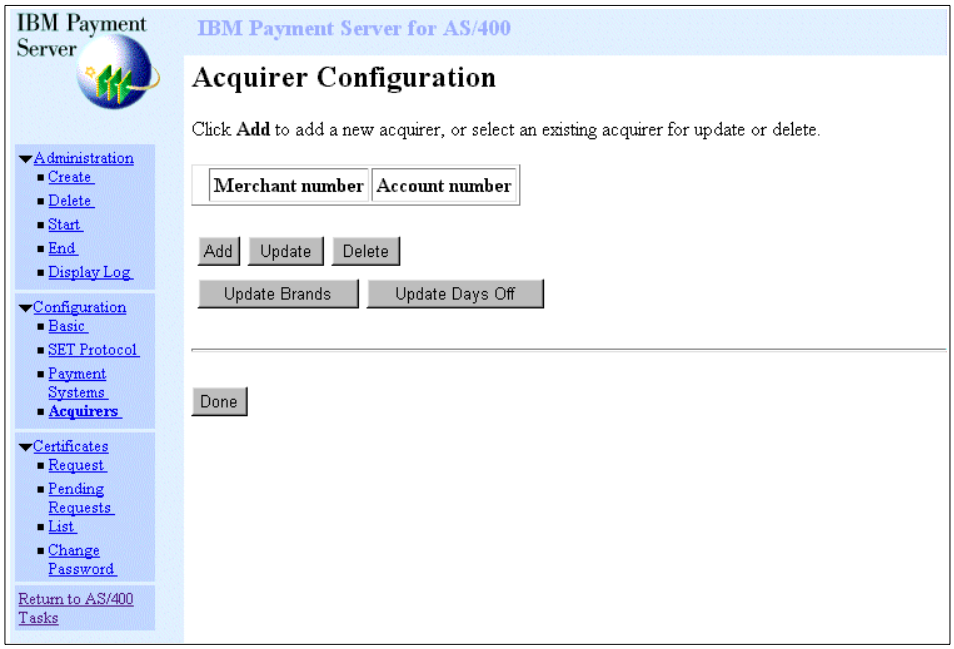


Figure 20. Payment Server acquirer configuration page

3. On the Acquirer Configuration form (Figure 21 on page 42), complete the page by using the information that you received from your acquirer. The Merchant number is the merchant number you selected in 2.2.8, “Payment System configuration of the Payment Server” on page 37. Use your values from Table 2 on page 17 and Table 3 on page 18 as the input values to the fields shown in Figure 21.

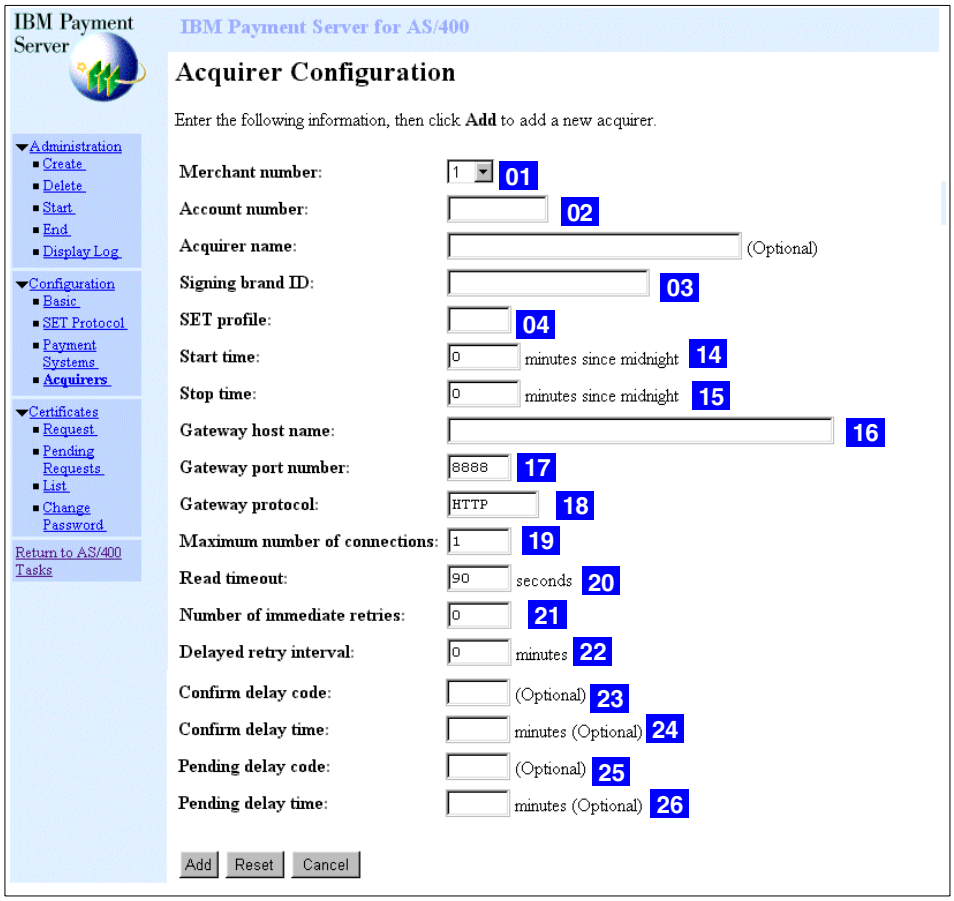


Figure 21. Payment Server acquirer configuration form page

Click **Add**.

The SET profile field defines how the Payment Server uses the SET protocol or messages. Depending on the value to this field, the Payment Server uses different optional parts of the SET messages.

Acquirers use the SET protocol in a variety of ways. In some cases, acquirers require merchants to use particular optional fields of a message or a different message. In other cases, acquirers do not allow merchants to use particular optional fields or some types of messages. In addition, different acquirers interpret the SET specification differently. Payment Server needs to be able to work with each of these different environments. The SET profile field encodes these differences in the Payment Server's

configuration. To determine what value should be used for your Payment Server configuration, you must consult your acquirer.

If you need more information about the SET profile, see the readme file at:
`http://www.ibm.com/software/webservers/paymgr/support/acqprofile.html`

If you have the latest PTFs installed on your AS/400 system, you can also find the same information at:
`http://as400:2001/QIBM/PymSvr/Admin/readme.ndm/readme`

Here, `as400` is your AS/400 TCP/IP host name.

4. You receive the message showing that the Acquirer profile is added (Figure 22).

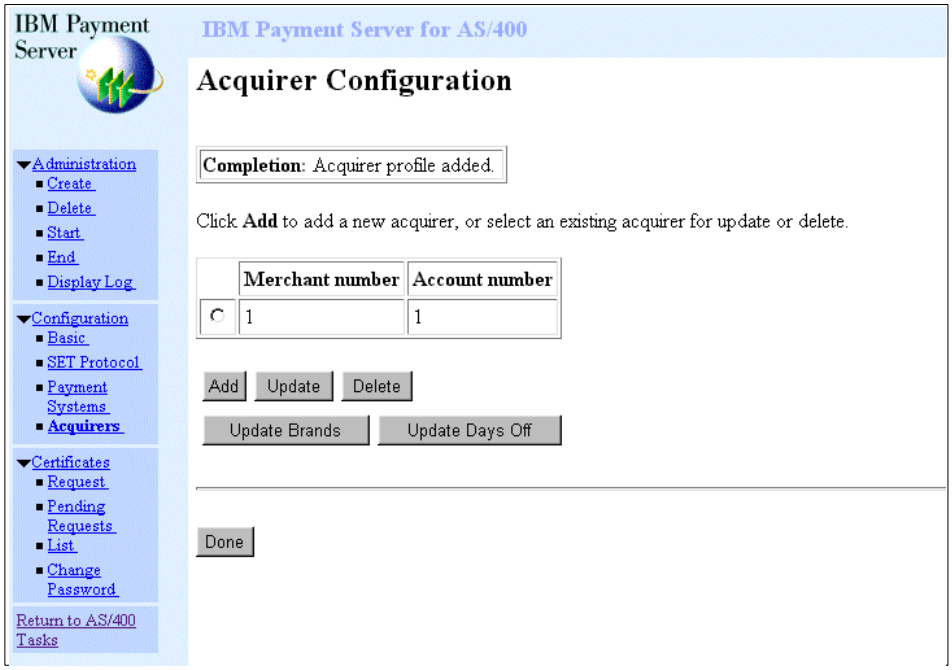


Figure 22. Payment Server acquirer profile added page

Select the Merchant radio button, and click **Update**.

5. On the Acquirer Brand Configuration page (Figure 23 on page 44), click **Add**.

Figure 23. Payment Server acquirer add brand page

6. On the Acquirer Brand Configuration form (Figure 24), enter the information you received from your acquirer. Use the values from Table 2 on page 17 as input to the fields.

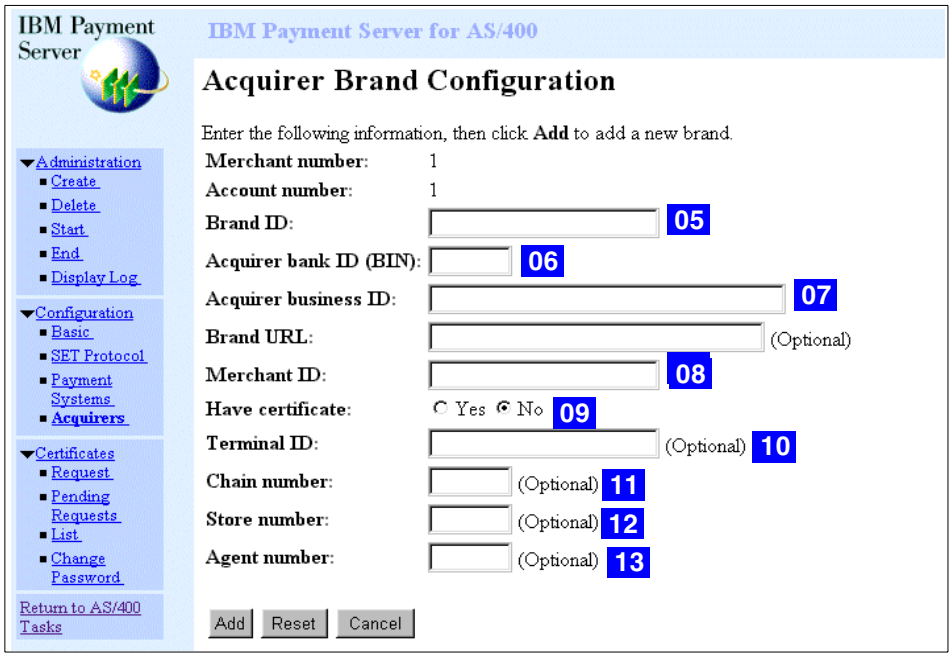


Figure 24. Payment Server acquirer brand configuration form

Click **Add**. The Payment Server Acquirer Brand Configuration page (Figure 25 on page 46) shows a completed status that the brand profile was added.

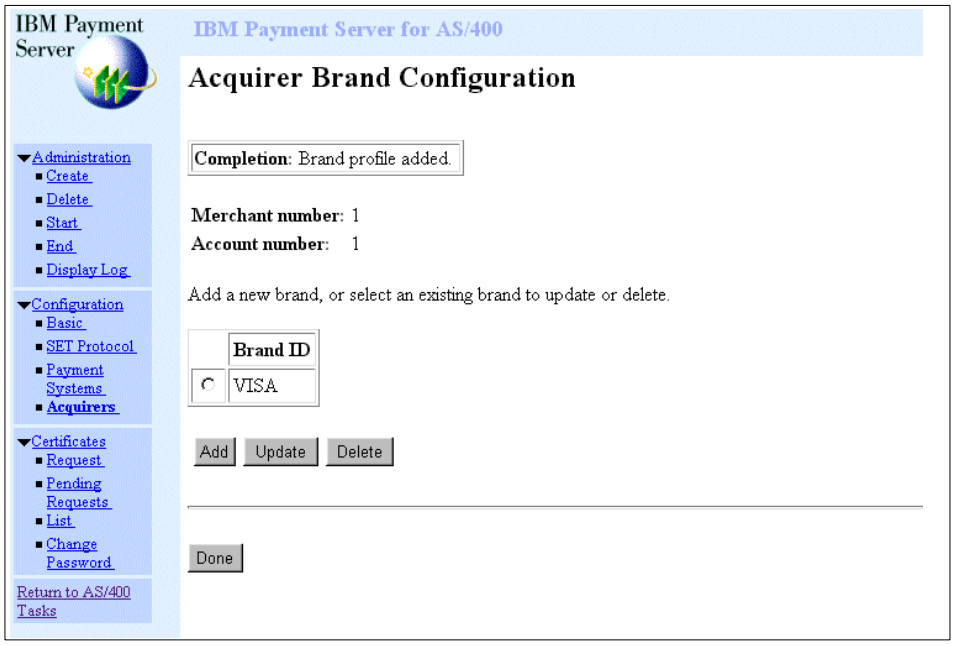


Figure 25. Payment Server acquirer brand profile added page

2.2.10 Requesting a SET merchant certificate from a CA

To conduct commercial business on the Internet, you should request your SET merchant certificate from a certificate authority (CA), such as VeriSign or Globeset. This section describes how to obtain a SET merchant certificate from a certificate authority. To use SET for secure payments, your server must have a digital certificate.

When you choose to use a CA to issue a server certificate, you must first request the certificate. To request the certificate, follow these steps:

Note

Before you start requesting the certificate, make sure you have all information ready for the request. Leaving any windows in the Request Certificate for a certain time will cause the session time out, and then you have to start all over again from the beginning.

1. From the IBM Payment Server for AS/400 page, click **Certificates** in the left-hand frame to display an extended list of server tasks.

2. Click **Request** from the list to request a certificate.
3. On the Certificate Management Login page (Figure 26), enter your Key database password that you selected when you created your Payment Server in 2.2.4, “Creating a Payment Server instance” on page 24.

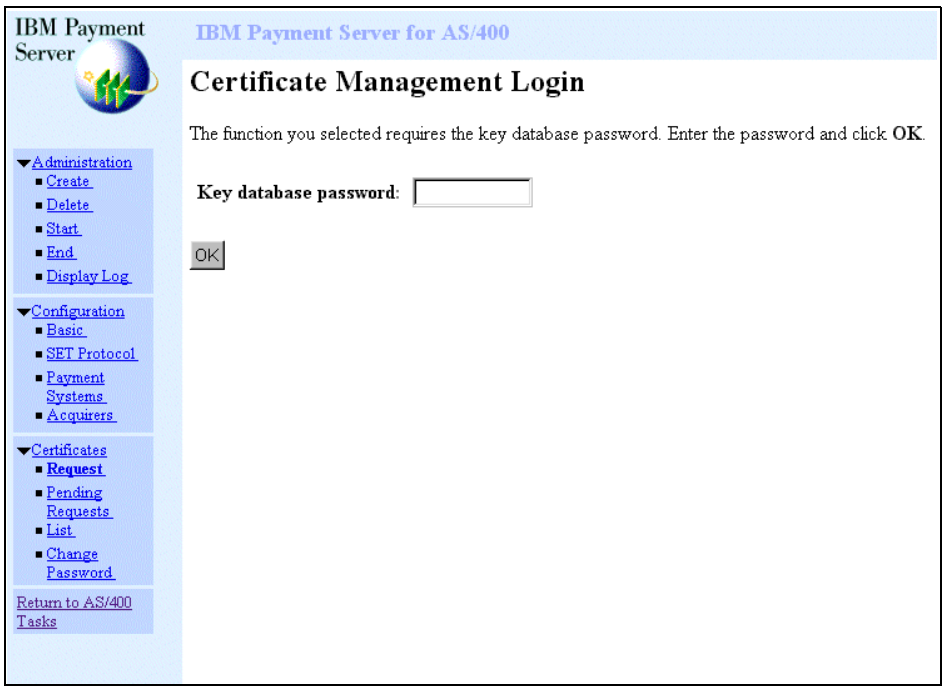


Figure 26. Payment Server Certificate Management Login page

Click **OK** to display the Request Certificate page.

4. Enter the URL of the CA in the Request URL field. Click **Continue**. The Payment Server Request Certificate page (Figure 27 on page 48) is displayed.

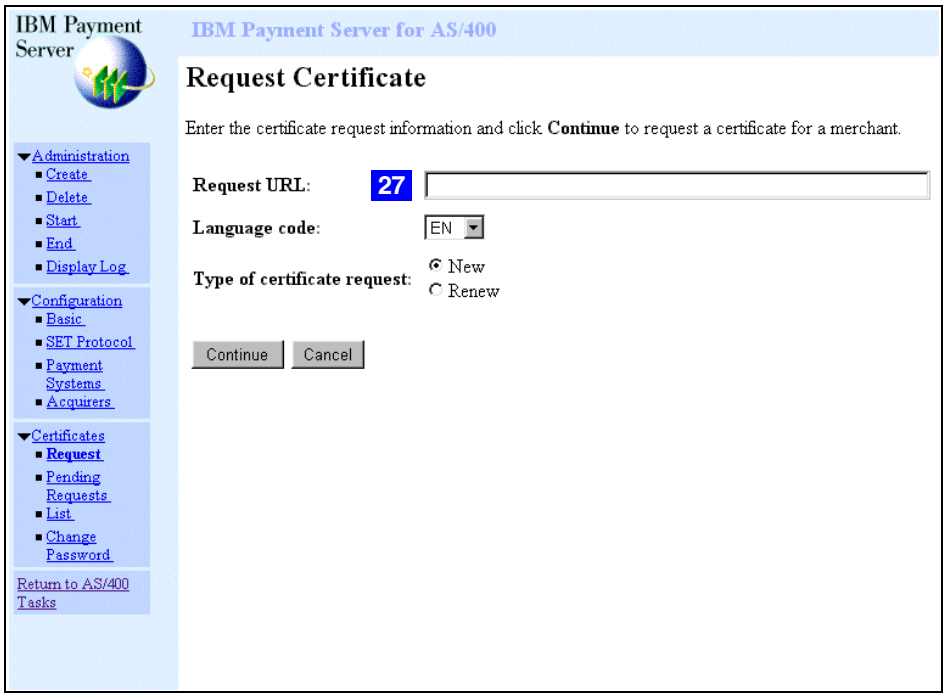


Figure 27. Payment Server certificate request

Click **Continue** to display the Payment Server Request Certificate Brand page.

5. On the Payment Server Request Certificate brand page, select the brand for the certificate, and click **Continue** (Figure 28).

> **Attention**
>
> It is important that you configure acquirers and brands with the information received from the acquirer before submitting the request.

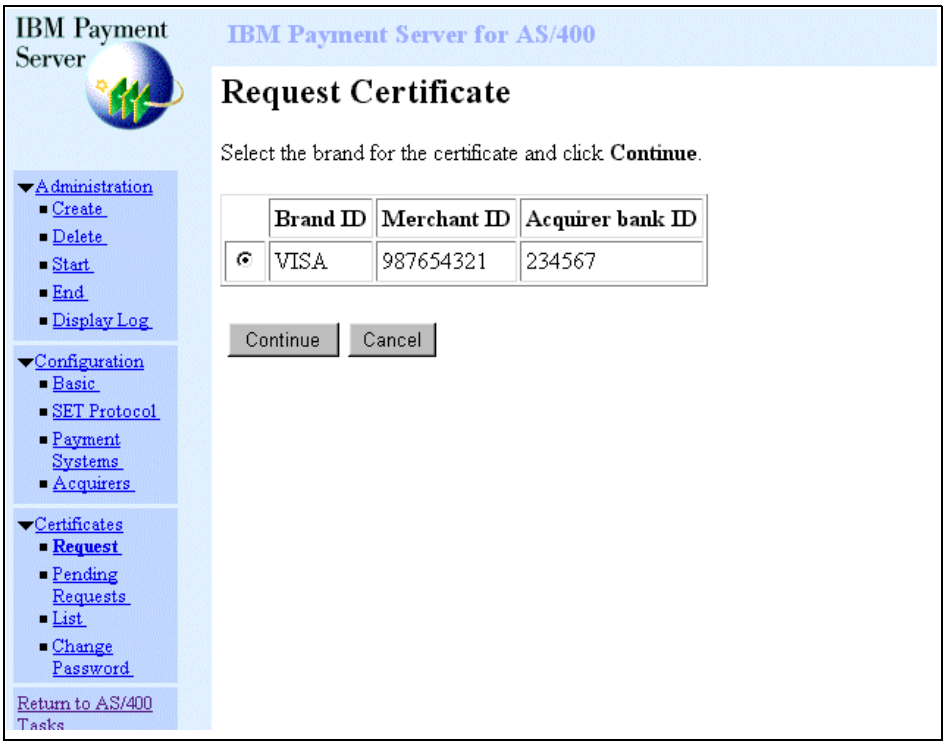


Figure 28. Payment Server request certificate brand

6. If you do not already have a valid SET certificate installed, you will be asked to enter the root hash, which is a 40-character string. On the Payment Server Request Certificate root hash page, enter the root hash code that you received from your acquirer (Figure 29 on page 50).

   Click **Continue**.

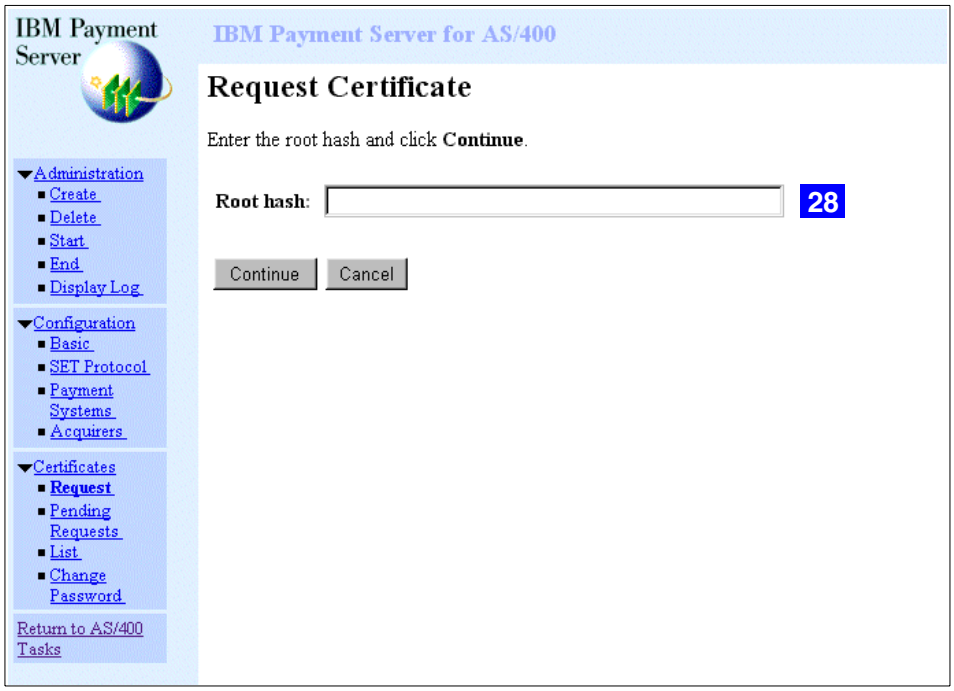


Figure 29. Payment Server certificate root hash

7. On the Payment Server Request Certificate policy page, you have to read and agree with the policy. Figure 30 shows a sample policy statement that depends on the CA. Click **Continue**.

Figure 30. Payment Server certificate request policy

8. On the Payment Server Request Certificate information page (Figure 31 on page 52), enter the certificate information requested by the certificate authority. Use your values from Table 4 on page 19 as input to the fields.

   Click **Continue**.

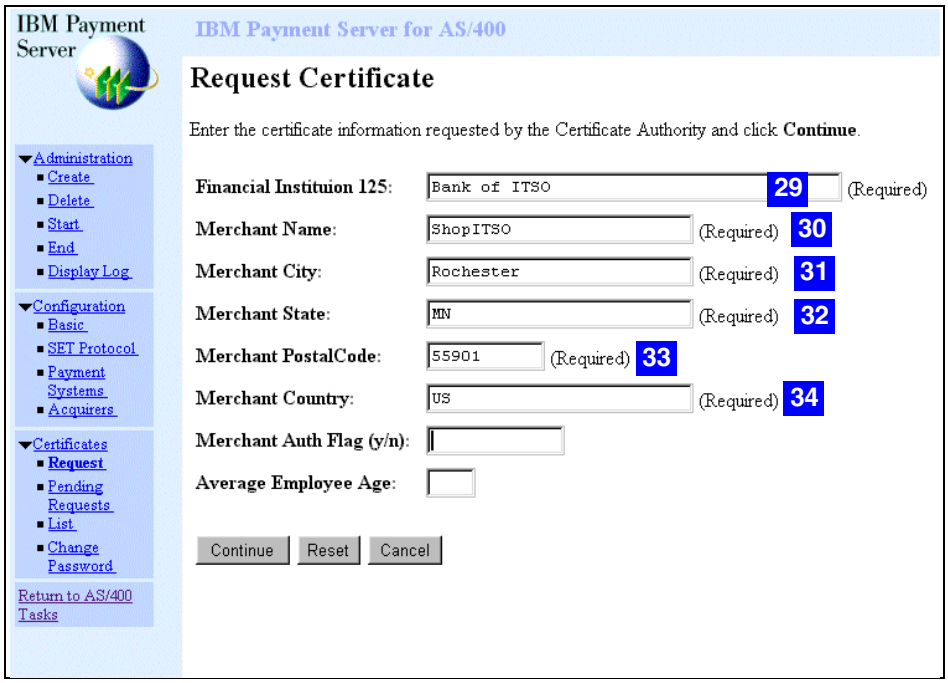


Figure 31. Payment Server certification request information page

9. The Payment Server Request Certificate complete page is displayed (Figure 32), and the certification receive process is finished. Click **Done**.

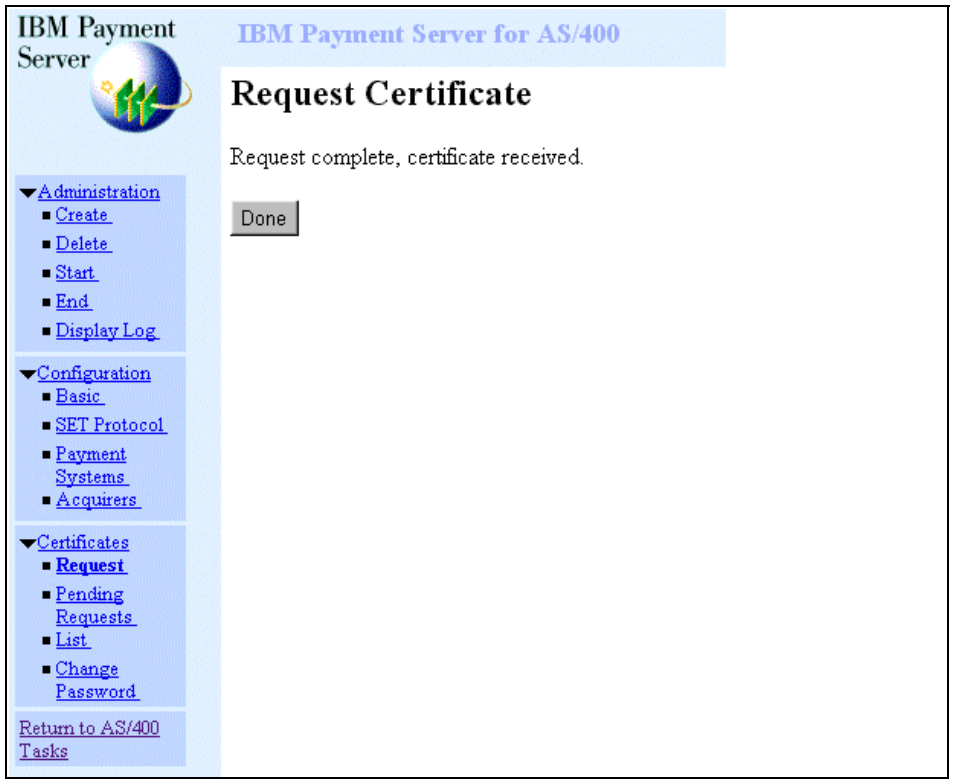


Figure 32. Payment Server certificate request complete page

You have now installed your brand's digital SET certificate on your Payment Server.

2.2.11 Starting and ending the Payment Server

When you have completed the previous tasks, you are ready to start and end the Payment Server.

2.2.11.1 Starting the Payment Server

If you use Net.Commerce with Payment Server, the Payment Server starts automatically when you start your Net.Commerce instance. There are two ways to start your Payment Server independent from Net.Commerce.

One way is to start the Payment Server from the IBM Payment Server for AS/400 page (Figure 33 on page 54). To do so, follow these steps:

1. Click **Administration->Start**.

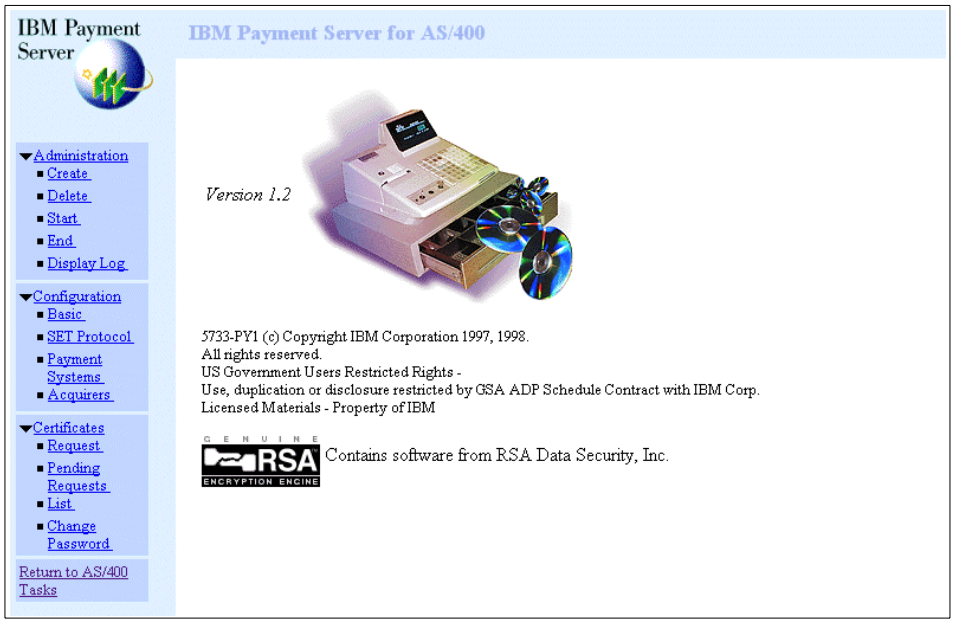


Figure 33. Payment Server page

2. On the Start Payment Server page (Figure 34), enter the Key database password. Click **Start**.

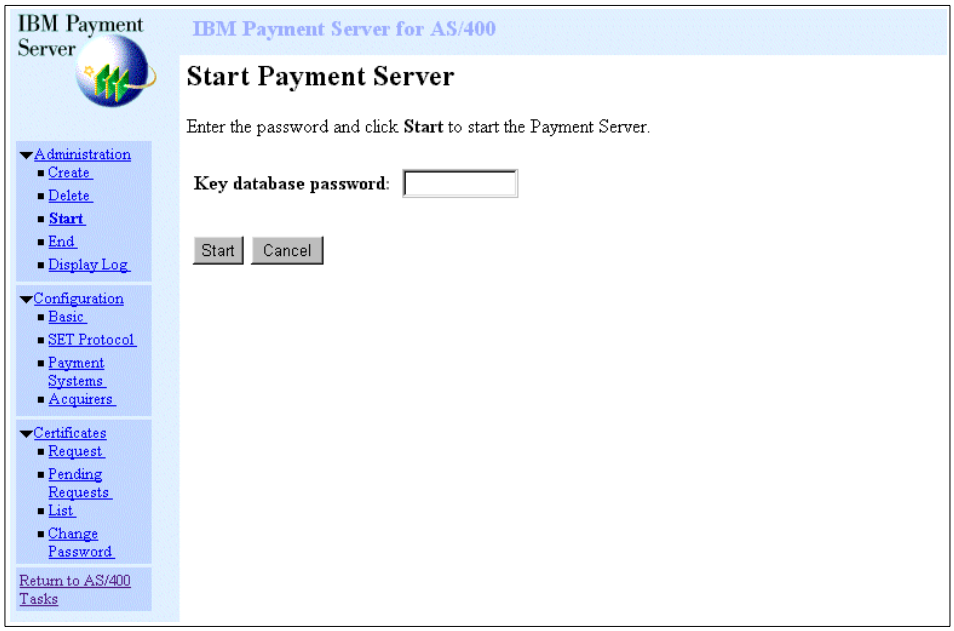


Figure 34. Start Payment Server

The Payment Server starts as shown in Figure 35 on page 56.

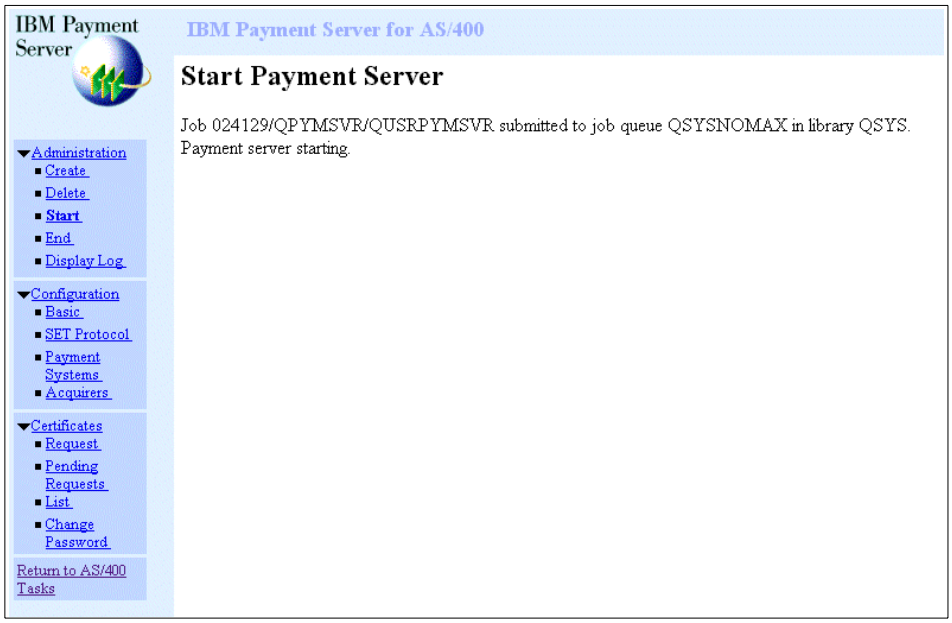


Figure 35. Payment Server starting page

The other way to start the Payment Server is to use an AS/400 command. To do so, follow these steps:

1. On an AS/400 command line, type:

   `STRPYMSVR KEYPWD(password)`

   Here, `password` in the command is the Key database password.

2. Press Enter. The following message appears: `Payment server starting`.

The Payment Server job QUSRPYMSVR is starting. To make sure the server job is fully started, look for the message PYM1101 in the QSYSOPR message queue.

2.2.11.2 Ending the Payment Server

There are two ways to end your Payment Server. The first way is to end it from the IBM Payment Server for AS/400 page (Figure 36). The steps for this process are described here:

1. Click **Administration->End**.

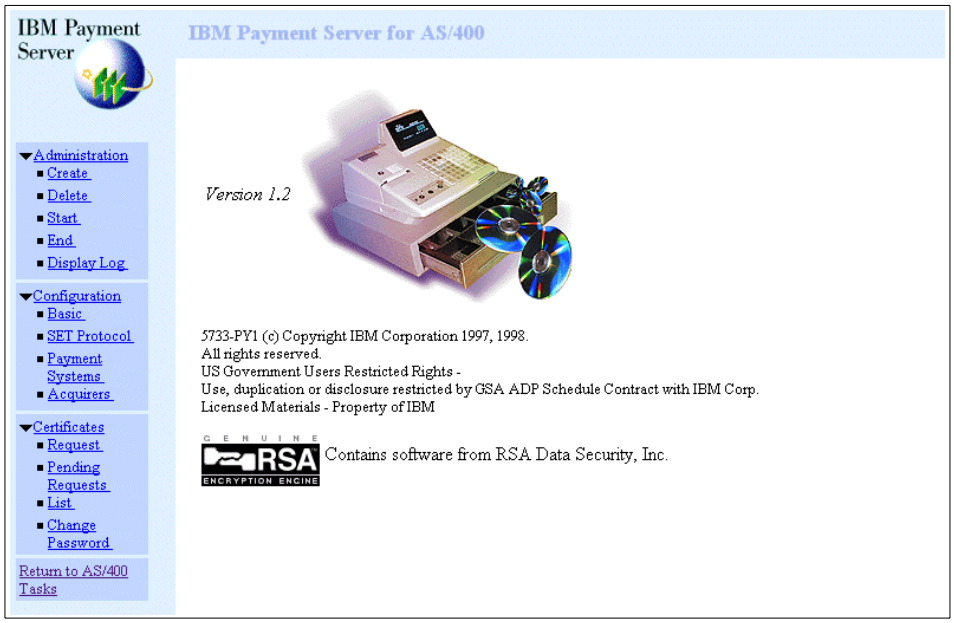


Figure 36. Payment Server page

2. On the End Payment Server page (Figure 37 on page 58), click **End**.

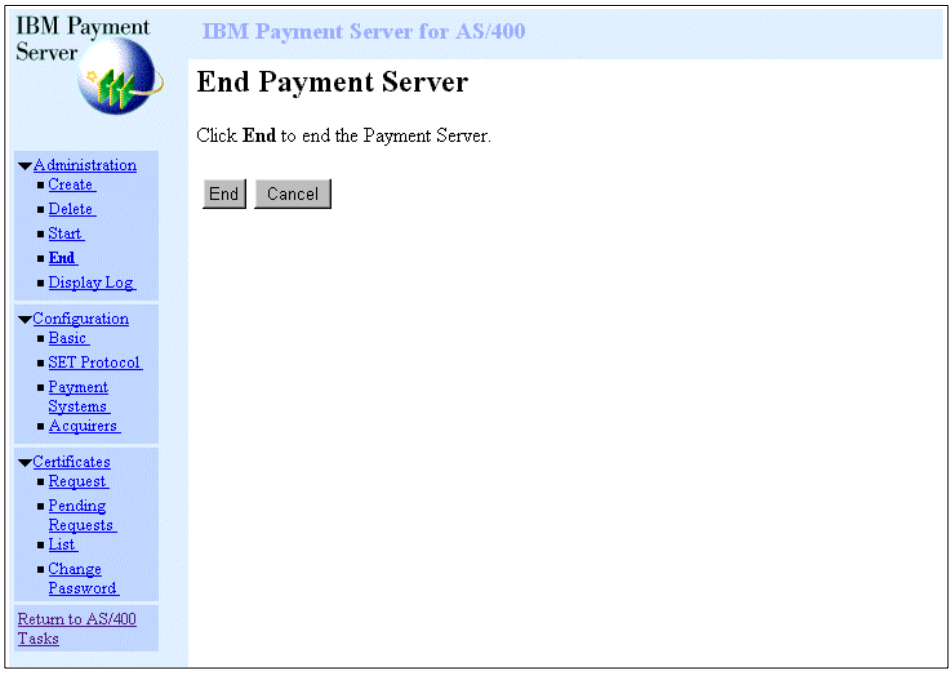


Figure 37. End Payment Server

The Payment Server ends, as shown in Figure 38.

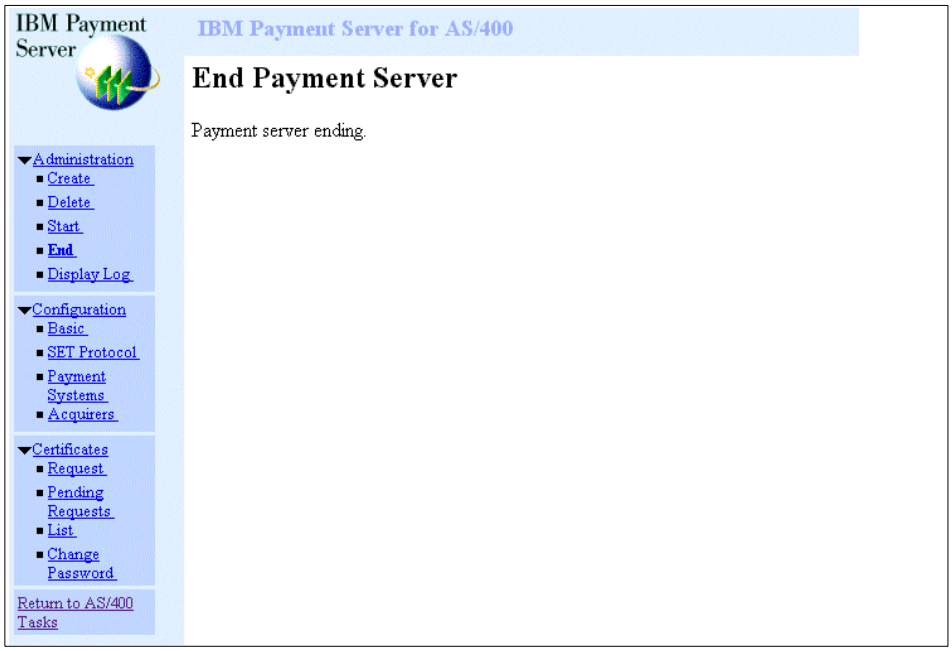


Figure 38. Payment Server ending

The other way to end your Payment Server is to use an AS/400 command. This process is described here:

1. On an AS/400 command line, type:

   ```
   ENDPYMSVR
   ```

2. Press **Enter**. The following message appears: `Payment Server ending`.

The Payment Server job QUSRPYMSVR is now ending in the subsystem QSYSWRK.

Chapter 3. Payment Server APIs

This chapter describes the relationship between Payment Server API commands and SET messages in SET transactions. It presents different business scenarios that illustrate how C API calls are invoked at different stages in the SET transaction process. This chapter also discusses how to process transactions that require special handling in a SET environment.

After completing this chapter, you should be able to:

- Define the basic terms used in the credit card industry and associate these terms with corresponding Payment Server API commands and SET messages.
- Use the Payment Server and the payment gateway by using C APIs in typical SET business scenarios.

3.1 Payment Server API and credit card transactions

The Payment Server provides several APIs for processing payments. These APIs can be implemented in a variety of ways using a number of different programming languages. The examples and scenarios in this redbook use the C API interface, referred to as the C API. This interface uses the C language functions and table definitions to communicate with the merchant server. The Payment Server supplies source code for this interface in the C API library. When writing programs and applications using the C API, a programmer does not have to understand the communications protocol between the merchant server and Payment Server because the C API library handles all of the protocol details.

This section presents business scenarios illustrating how Payment Server APIs can be invoked by your merchant applications. It also explains important concepts in the electronic commerce environment. These concepts include approving and depositing purchases, reversing a deposit, refunding a payment, and batch administration. These are generic scenarios that can apply to any payment protocol.

The terms *authorize*, *capture*, and *credit* are often associated with credit card transactions and relate to specific processing functions. See Table 10 on page 62 for a description of how they are used in the credit card industry.

Figure 39 on page 62 shows the relationship between the card transaction (purchase) and the processes incurred by the transaction, such as Authorize and Capture.

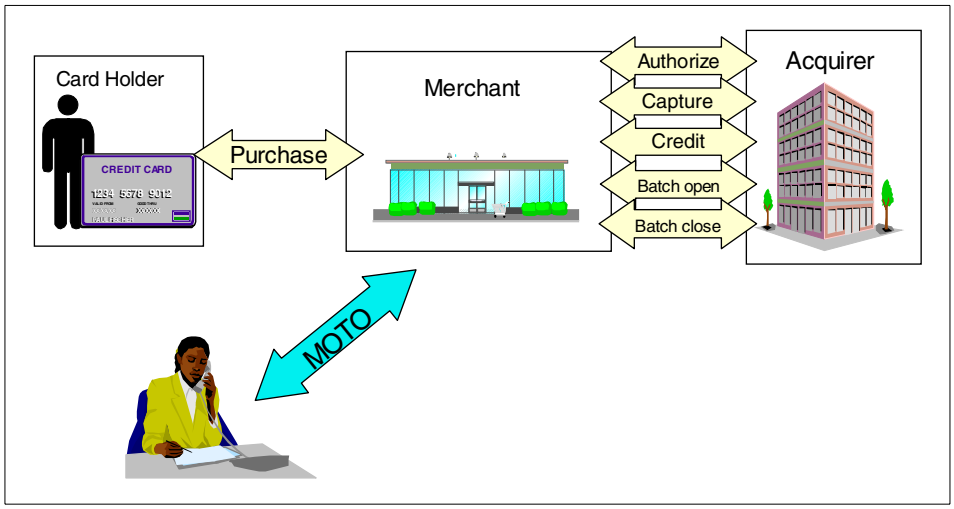


Figure 39. Credit card transaction

Table 10. Description for the terms of credit card transactions

Term	Description
Purchase	The purchase process is initiated when the cardholder is ready to pay for the goods and services ordered from the merchant.
Authorize	The cardholder is given permission to make a purchase. The process involves assessing transaction risk, confirming that a given transaction does not raise the account holder's debt above the account's credit limit, and reserving the specified amount of credit.
Capture	Funds can be moved or deposited to the merchant's account.
Credit	The merchant needs to return money to the cardholder following a valid capture transaction. For example, if goods are returned or are defective, the cardholder receives a credit.
Open Batch	Either the merchant or the acquirer opens a log to group together transactions and process them at the same time.
Close Batch	The merchant or acquirer closes the open log. When the batch is closed, no more transactions are added to the log.
MOTO	Mail order/telephone order. The cardholder contacts the merchant by telephone, over the Internet, or by mail, for example, without wallets.

These typical credit card processing functions relate to specific Payment Server API and SET messages, which are shown in Table 11. The Payment Server API column lists the C API functions that a merchant server would invoke to communicate with the Payment Server.

Table 11. Payment server APIs and credit card functions

Payment Server API	Credit card functions	SET messages
etReceivePayment()	Card request	SET initiation, PInitReq, PInitRes, PReq, PRes
etAcceptPayment	MOTO request	-
etApprove()	Authorize	AuthReq, AuthRes
etApproveReversal()	Authorize reversal	AuthRevReq, AuthRevRes
etDeposit()	Capture	CapReq, CapRes
etDepositReversal()	Capture reversal	CapRevReq, CapRevRes
etRefund()	Credit	CreditReq, CreditRes
etRefundReversal()	Credit reversal	CreditRevReq, CreditRevRes
etOpenBatch()	Open a new batch	BatchAdminReq, BatchAdminRes
etCloseBatch()	Close an existing batch	BatchAdminReq, BatchAdminRes

You should now understand the correlation between the credit card functions and the Payment Server API commands.

3.2 Basic payment processing functions with the Payment Server API

In this section, business scenarios are provided to illustrate the flows for SET transactions and the Payment Server API. Each scenario begins with an online shopper who has finished browsing and is ready to pay for the goods and services ordered. The merchant server then presents the shopper with a Web page summarizing the purchases.

These scenarios assume:

- The phrases Buy Button and Buy Page refer to functions normally included in the merchant's online catalog.
- All parties involved (merchant, acquirer, and shopper) have SET certificates.

- If the shopper is using a browser-based SET wallet in the scenario, the shopper has already installed the eWallet software.
- The payment type is credit card (SET).
- The Payment Server is started, and the etInitializeAPI() function has been called.

3.2.1 Before starting

A handle is used by the C API to identify sessions between the merchant server and the Payment Server. A session and its handle are created as part of the etInitializeAPI() call when the merchant server defines all of the attributes for that session (such as the Payment Server host name and the Payment and Admin API port numbers). The etInitializeAPI() call returns a handle for the new session. This handle is passed as a field for subsequent API calls for that session. Sessions are terminated and resources are released using the etCloseAPI() call.

After calling etInitializeAPI(), you prepare the protocoldata structure that specifies the SET data the Payment Server requires to compose a message. The information shown in Table 12 must be included in the protocoldata structure.

Table 12. Protocol data

Keyword	Value	Type
$BRAND	Cardholder's brand matching a brand configured in the Payment Server.	String Example value: VISA
$PAN	Cardholder's Personal Account Number (PAN), used as PAN in the payment initiation.	String Example value: 2222333344445555
$EXPIRY	Cardholder's card expiration, used in the payment initiation.	String Example value: 200004 (for April 2000)

This protocoldata structure is created by using the etAddProtocolData() or etAddProtocolDataString() API as shown in Figure 40.

```
BRAND = "VISA";
PAN = "1122334455";         /* sample "test" CC Number      */
EXPIRY = "199912";

    memset(&protocoldata, 0, sizeof(protocoldata));
    rc = etAddProtocolDataString(&protocoldata, "$BRAND          ",
                                 BRAND,strlen(BRAND), &rc2);
   if (rc > 0)
   {
    printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
    etCloseAPI(&handle);
    exit(0);
    }

    rc = etAddProtocolDataString( &protocoldata, "$PAN            ",
                                 PAN,strlen(PAN), &rc2);
   if (rc > 0)
   {
    printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
    etCloseAPI(&handle);
    exit(0);
    }

    rc = etAddProtocolDataString( &protocoldata, "$EXPIRY         ",
                                 EXPIRY,strlen(EXPIRY), &rc2);
   if (rc > 0)
   {
    printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
    etCloseAPI(&handle);
    exit(0);
    }
```

Figure 40. etAddProtocolDataString() API sample

Because the AS/400 system uses the EBCDIC data, and wallet programs use ASCII data, the order description must be converted to ASCII in order to be sent.

The code shown in Figure 41 on page 66 converts the order description from the job CCSID to ASCII. This is appropriate when the order description is passed in as a parameter or retrieved from a database text field. In both those cases, the order description is coded in the job CCSID.

```
/* Get a handle to convert from EBCDIC to US ASCII. */
 toCCSID.CCSID = 819;
 fromCCSID.length_option = 1; /* iconv determines input length */
 jobCCSIDtoAscii = QtqIconvOpen( &toCCSID, &fromCCSID );
 if (jobCCSIDtoAscii.return_value != 0 ) {
   /* Error handling here. */
   printf("Cannot get conversion handle, rc = %d",
          jobCCSIDtoAscii.return_value);
   etCloseAPI(&handle);
   exit(0);
 }

 /* Now convert the characters to ASCII. */
 inBytesLeft = 0;  /* iconv determines input length */
 outBytesLeft = sizeof(orderDescriptionString);
 inBuffer = orderDescriptionString;
 outBuffer = convertedString;
 iconvrc = iconv(jobCCSIDtoAscii, &inBuffer, &inBytesLeft,
            &outBuffer, &outBytesLeft);

 /* Close and release code conversion descriptor. */
 iconv_close(jobCCSIDtoAscii);

 if (iconvrc >= 0) {
   orderdescription.contenttypecharset = "text/plain";
   orderdescription.description = convertedString;
   orderdescription.length = strlen(convertedString);
 }
 else {
   /* Error handling here. */
   printf("EBCDIC to ASCII conversion failed, rc = %d", iconvrc);
   etCloseAPI(&handle);
   exit(0);
 }
```

Figure 41. EBCDIC to ASCII conversion sample

Figure 42 illustrates the generic flow of the transaction processing using the Payment Server APIs.

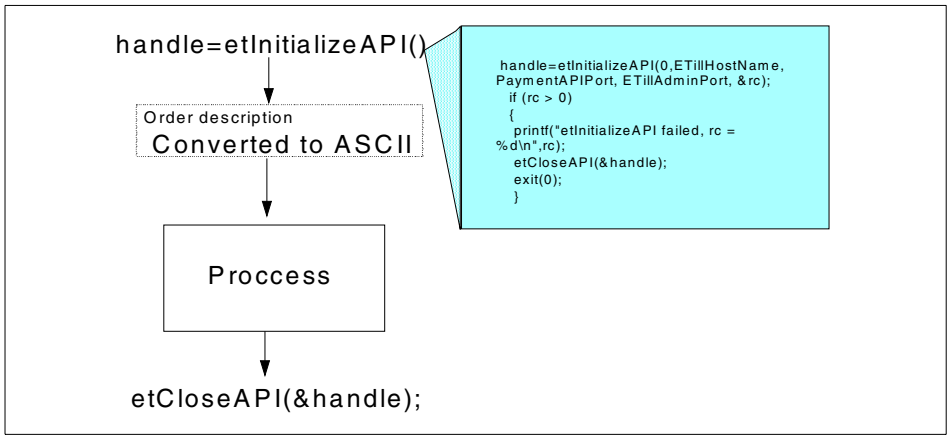


Figure 42. Payment server API flow

In the following sections, we discuss how to use the APIs depending on the process of the credit card transactions. We have two types of credit card transactions: the transaction using the SET wallet and the transaction without the SET wallet. There is one difference in terms of the APIs used in the two types. When you use the SET wallet, you need to call the etReceivePayment() API. On the other hand, when you do not use the SET wallet, you need to call the etAcceptPayment() API. Figure 43 on page 68 outlines the process *with* the wallet on the left side and the process *without* the wallet in the right side. It also refers you to the sections that describe each API.

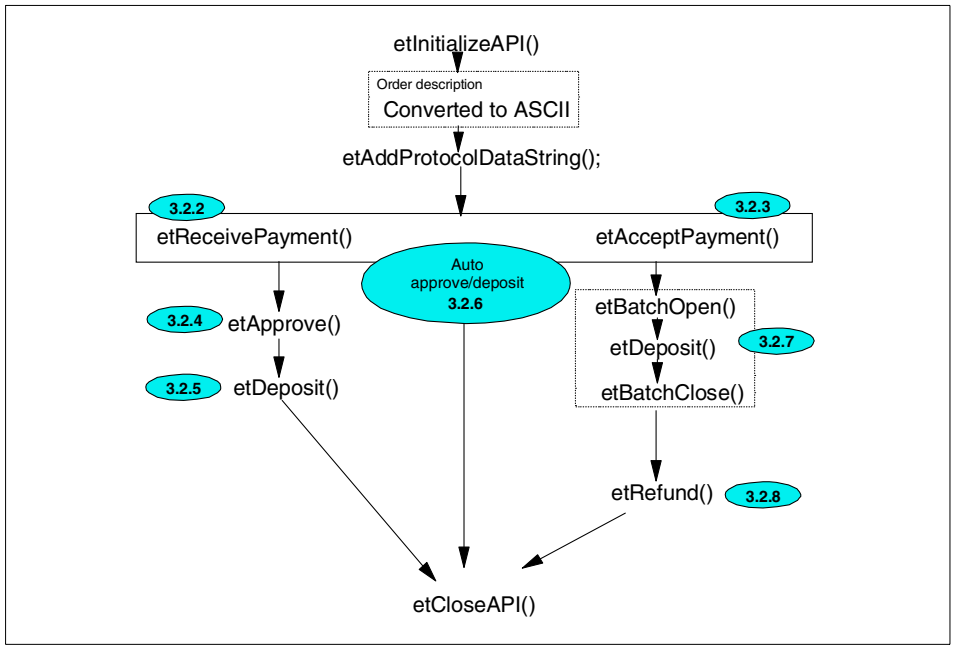


Figure 43. API flow

Note

In a real environment, the sample logic for etRefund() API, shown as 3.2.8 in Figure 43, also needs the etBatchOpen() and etBatchClose() APIs. The sample above it, shown as 3.2.7, also uses the APIs. The reason why the sample does not have the batch APIs is that, in this redbook, we try to show both cases, with and without batch processes.

3.2.2 Shopper and Payment Server interaction with a SET wallet

The example in Figure 44 illustrates the flow of data between the online shopper and the merchant, using the C API interface. In this business situation, you can assume:

- No transactions are flowing to or from the acquirer at the time the shopper is placing an order.
- Authorizations and captures can be performed manually at a later time.

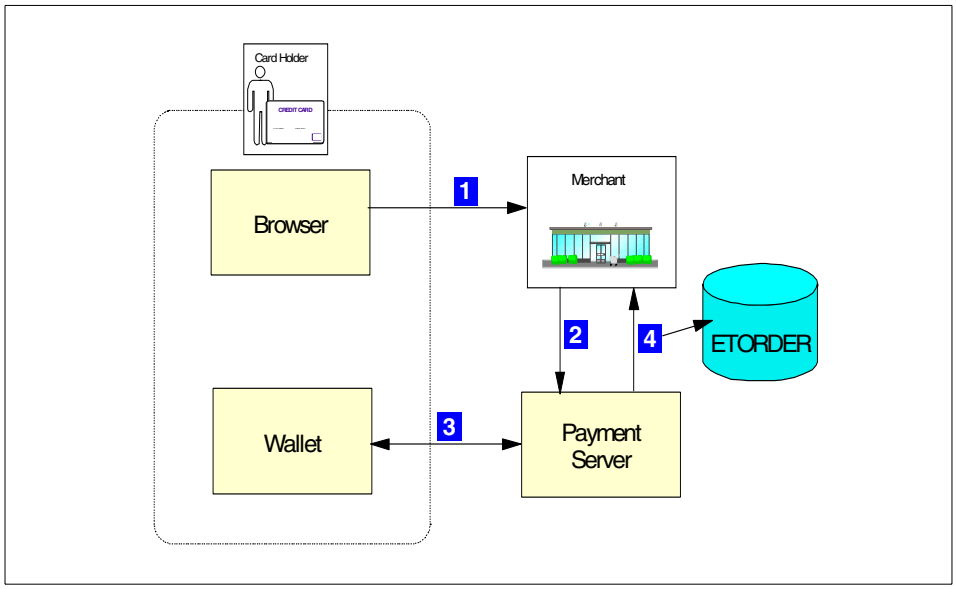


Figure 44. Interaction with a SET wallet

The data flow is explained here:

1. A shopper contacts an Internet merchant and places an order.
2. The merchant server calls the etReceivePayment() API. The Payment Server creates an order in the Requested state with the specified order number.
3. The Payment Server exchanges payment protocol-specific messages with the shopper's wallet. The order remains in the Requested state until the exchange of data is complete.
4. If the request is successful, the order moves into an Ordered state. At this point, the order information is stored in the order information database (QUSRPYMSVR/ETORDER). Then, the merchant server can create payments for this order. If there is a financial problem with the request, the order moves into a Rejected state.

3.2.3 Shopper and Payment Server interaction without a SET wallet

The example shown in Figure 45 on page 70 illustrates the flow between the shopper and the merchant for a purchase without a wallet, using the C API interface. The points are explained in the list following the figure. In this scenario, you can assume:

- No transactions are flowing to or from the acquirer at the time the shopper is placing an order.
- Authorizations and captures can be performed manually at a later time, such as when the batch is closed.

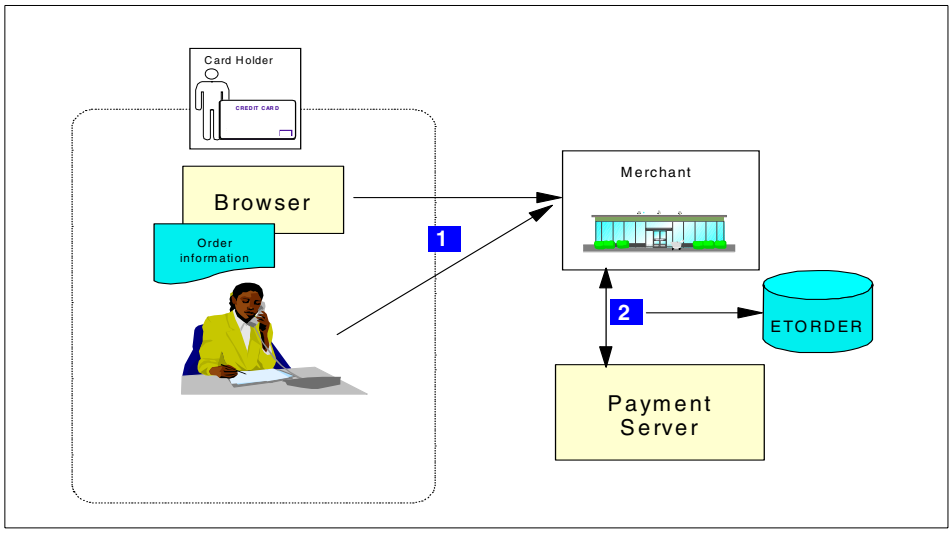


Figure 45. Interaction without a wallet

1. A shopper contacts a merchant and places an order. For example, they may contact the merchant by telephone, over the Internet, or by mail. We call this MOTO (mail order/telephone order).
2. The merchant server calls the etAcceptPayment() API. The Payment Server creates an order with the Ordered state in ETORDER with the specified order number. The order is put in the Ordered state rather than in a Requested state (as it was with the previous etReceivePayment scenario), because the etAcceptPayment() call contains all of the shopper's information. At this point, the order is “viable”, and the merchant server can create payments for this order.

3.2.4 Approving an order

This scenario is a logical followup to the first and second scenarios, in which the purchase is initiated and an order is created. After a customer decides to make a purchase online as described in 3.2.2, “Shopper and Payment Server interaction with a SET wallet” on page 68, or 3.2.3, “Shopper and Payment Server interaction without a SET wallet” on page 69, the next step in the payment cycle is to have the transaction approved.

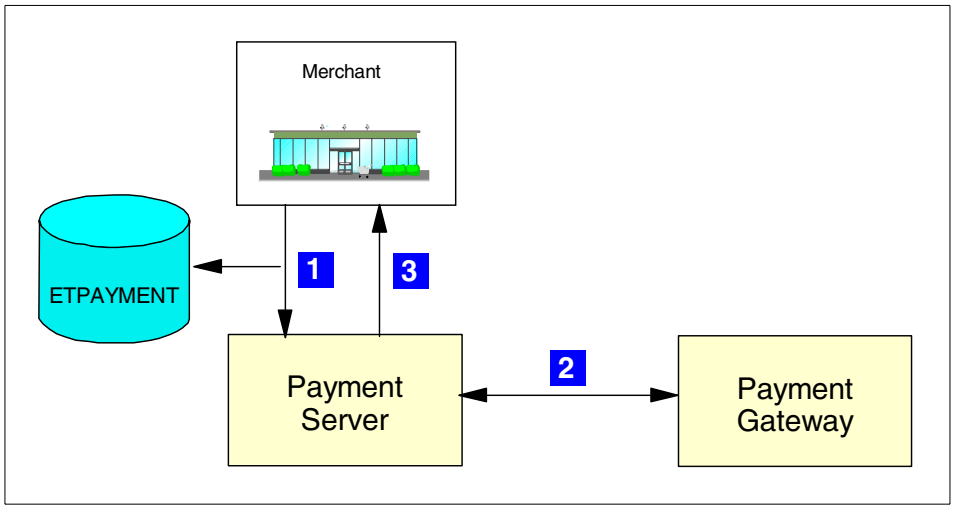


Figure 46. Approving an order

The approval process shown in Figure 46 is explained here:

1. After an order has been created, the merchant server calls the etApprove() function to approve monies for the order.
2. The Payment Server creates a payment object associated with the order and assigns it the payment number specified by the merchant server. As part of the process of creating the payment, the Payment Server may need to interact with the acquirer.
3. If the etApprove() request is successful, the Payment Server moves the payment object into an Approved state. If the acquirer rejects the etApprove() request for financial reasons, the payment object goes into the Declined state.

The merchant has the option of automatically approving an order at the time it is initiated or using an explicit approval request at a later time. For more detailed information, see 3.2.6, “Automatic approval and deposit” on page 73.

3.2.5 Depositing an approved purchase

This scenario illustrates the flow of data between a merchant server, a Payment Server, and an acquirer's payment gateway. This scenario is a logical followup to the scenario described in the previous section.

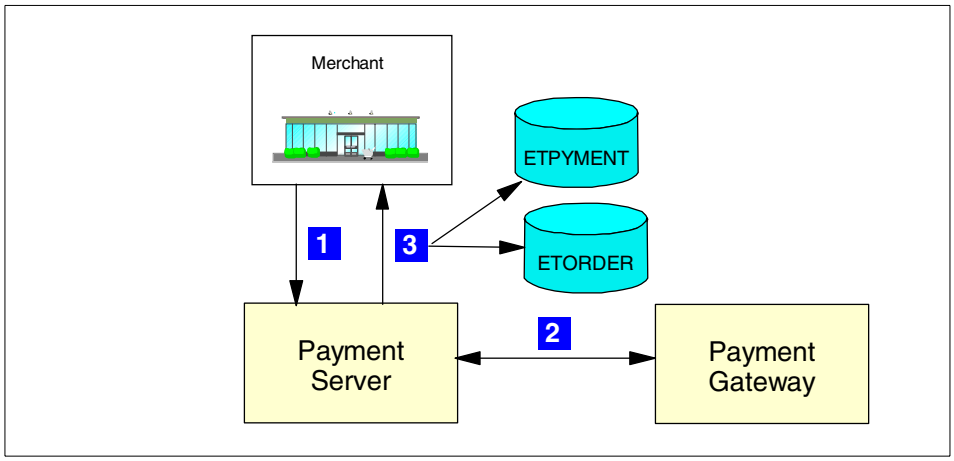


Figure 47. Depositing an approved purchase

The process flow shown in Figure 47 is explained here:

1. After a payment has been approved, the merchant server calls the etDeposit() function to deposit the approved funds.
2. The Payment Server issues a request to deposit the specified amount against the payment. As part of the process of depositing the payment, the Payment Server may need to interact with the acquirer.
3. If the etDeposit() request is successful, the Payment Server moves the payment object into the Deposited state (ETPAYMENT database). The order either remains in the Ordered state or moves into the Refundable state (ETORDER database), depending on the payment cassette being used. If the acquirer rejects the etDeposit() request for financial reasons, the payment object remains in the Approved state (ETPAYMENT database).

The merchant has the option of automatically depositing a payment at the time it is approved or using an explicit deposit request at a later time. To automatically deposit a payment through the C API interface, include the autoDeposit data structure in the etApprove() API. To do an explicit deposit request, use the etDeposit() C API.

See the following section for additional information about automatically depositing a payment.

3.2.6 Automatic approval and deposit

You can automatically approve an order and deposit a payment for the order at the same time. To automatically approve an order through the C API interface, include the Approve data structure in the etReceivePayment() or etAcceptPayment() functions. If an acquirer is closed when a purchase takes place, and you request automatic approval and deposit on the etReceivePayment() call or etAcceptPayment(), the approval request is queued until the acquirer is open.

To combine this process using the C APIs, include an AutoDeposit data structure in the Approve data structure on the etReceivePayment() or etAcceptPayment() functions as shown in Figure 48.

```
autodeposit.batchID = &batchid;
approveinfo.paymentnumber = paymentnumber;
approveinfo.paymentamount = orderamount.amount;
approveinfo.splitallowed = splitallowed;
approveinfo.deposit = &autodeposit;

/*************************************************************************
** Perform the Merchant Originated Payment (etAcceptPayment)
** with APPROVE and DEPOSIT flags on
*************************************************************************/
    rc = etAcceptPayment(handle,
                         &keys,
                         &orderamount,
                         PAYMENTTYPE,
                         &orderdescription,
                         &approveinfo,   /* approveinfo structure     */
                         &protocoldata,  /* protocoldata structure    */
                         &rc2,
                         &apiReturnValues);
```

Figure 48. Auto approve and deposit sample

3.2.7 Batch administration

This scenario illustrates the flow of data between a merchant server, a Payment Server, and an acquirer's payment gateway in a batch processing environment (Figure 49 on page 74). You need to check the SETProfile parameter. SETProfile is a field in the acquirer configuration. Its value tells the Payment Server how to use the optional parts of the SET Protocol when sending messages to the acquirer. It is appropriate when an acquirer allows merchants to perform their own batch administration. In this scenario, it is assumed that the merchant's policy is to submit batches to the acquirer on a weekly basis. Usually, you use etDeposit() and etRefund() APIs in the batch environment.

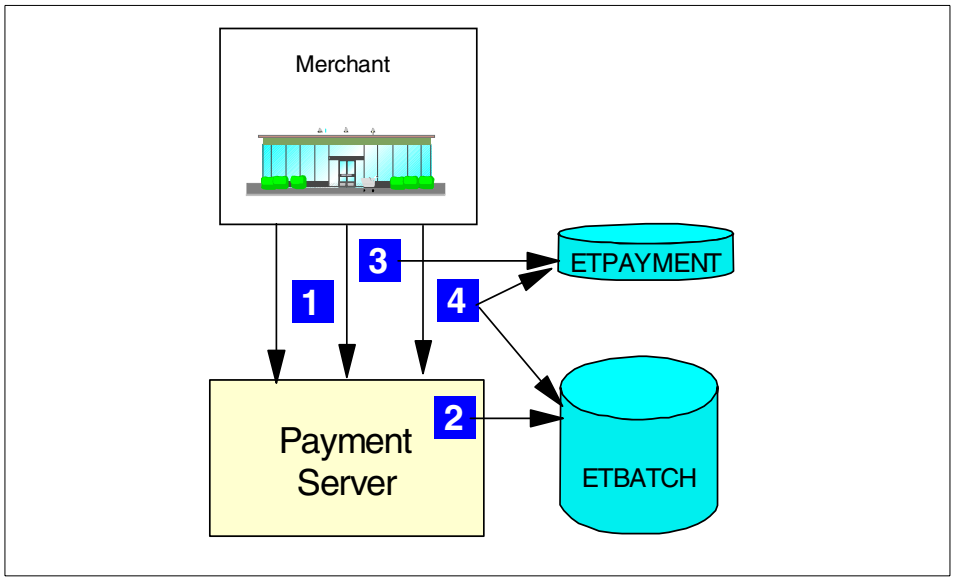


Figure 49. Batch administration

The Batch administration flow is explained here:

1. On Sunday morning (the start of the business week for this particular merchant), the merchant server starts a program to do batch administration. For each acquirer with whom the merchant has a business relationship, the merchant server calls etBatchOpen() API to the Payment Server. It passes a numeric value to be used as the batch ID for all payments and credits assigned to this batch.
2. The Payment Server creates a new batch with the assigned batch ID. If the etBatchOpen() function succeeds, the batch moves to an Open state (QUSRPYMSVR/ETBATCH database).
3. The merchant server calls the etDeposit() API with the batch ID to the Payment Server for all payments in the Approved state (ETPAYMENT database).
4. The merchant server calls etBatchClose() API with the batch ID. The batch moves to a Closed state (ETBATCH database). Then, all associated payments go into a Closed state. No further actions may be taken against those objects.

3.2.8 Credit (After batch closed)

There may be a case where a customer decides to cancel the purchase after the purchase is deposited to the merchant. To credit the deposited purchase to the customer, use the scenario shown in Figure 50.

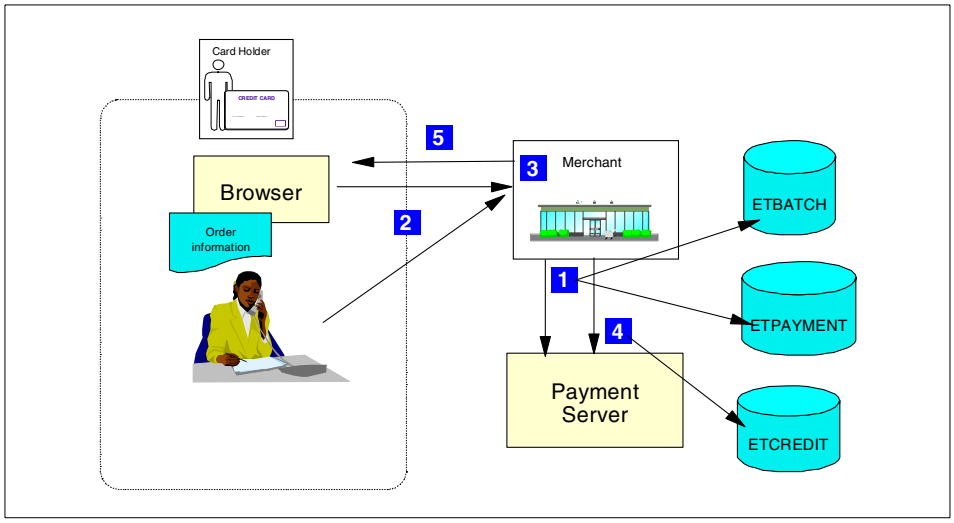


Figure 50. Credit flow

The credit flow is explained here:

1. The merchant server runs a program that issues etDeposit() API calls for all payments in an Approved state. On the etDeposit() calls, the ID of the batch opened at the beginning of the week is specified, and all of the payment deposits are included in the batch. As a result of eetDeposit() API, the Payment Server then sends the corresponding protocol-specific message to the payment gateway requesting deposit. Then, the merchant server makes an etBatchClose() API call to the Payment Server. The batch is closed.
2. The next day, the customer contacts the merchant and cancels the order.
3. Because the merchant controls the batch, they know that this week's batch is closed. The payment object is in a Closed state and no further actions can be performed on it. To refund the customer, they must create a credit.
4. The merchant server makes an etRefund() API call to the Payment Server. Within the etRefund() call is information about the order for which this refund is being made.

5. Because the original payment was deposited according to the acquirer's and merchant's records, the original deposit and the refund shows up in the customer's bill or monthly statement.

3.3 Sample API sources

The following sections list the sample programs with APIs as a reference.

3.3.1 Manual approve and deposit without a batch sample

This section shows an example of merchant-initiated authorization processing, in which the shopper *does not* have wallet software. This example applies to MOP and MIA. The merchant approves the order and then deposits it. This example shows that an acquirer controls batch, so that the sample does not issue batch operation APIs, such as etBatchOpen() and etBatchClose().

```
/*****************************************************************************
** Source File Name: sample1a.C
**
**
** Source File Description:
**   Provides SET(tm) payment processing function using the IBM Payment
**   Server for AS/400.
**
** To Compile: ADDLIBLE QPYMSVR
**             CRTCMOD MODULE(yourlib/SAMPLE1a) SRCFILE(yourlib/QCCSRC)
**             CRTPGM PGM(yourlib/SAMPLE1a) MODULE(yourlib/SAMPLE1a)
**                          BNDSRVPGM(QPYMSVR/QZETLPAY)
**
** To Run:     CALL PGM(yourlib/SAMPLE1a) PARM('1' '10' )
**
** Parameters: Merchant Number, Order Number
**
** Action:
**
**     Accept Payment
**     Approve
**
**     Deposit
**
**
** Payment Server APIs used: etInitializeAPI
**                           etAddProtocolDataString
**                           etAcceptPayment
**                           etApprove
**                           etDeposit
**                           etCloseAPI
**
** Disclaimer:
**   This sample is furnished by IBM as a simple example to
**   illustrate the use of the IBM Payment Server for AS/400.
**   This example has not been thoroughly tested under all
**   conditions.  IBM, therefore cannot guarantee or imply
**   reliability, serviceability, or function of this program.
**   This program is provided to you "AS IS."  All implied
```

```
**   warranties, including but not limited to the implied
**   warranties of merchantability and fitness for a particular
**   purpose, are expressly disclaimed.
**
** SET Secure Electronic Transaction, Secure Electronic Transaction,
** SET, and the SET Secure Electronic Transaction design mark are
** trademarks and service marks owned by SET Secure Electronic
** Transaction LLC. Use of the trademarks without a written license
** from SET Secure Electronic Transaction LLC is strictly prohibited.
**
*****************************************************************************/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <qtqiconv.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include "etillapi.h"


/*****************************************************************************
                              main()
*****************************************************************************/

int main(int argc, char *argv[])
{


  long           rc            = -1; /* primary return code         */
  long           rc2           = -1; /* secondary return code       */
  long           merchantnumber;    /* Parameter                   */
  long           ordernumber;       /* Parameter                   */
  long           paymentnumber = 1; /* used on et_Approve and      */
                                    /* et_Deposit                  */
  long           splitallowed  = 1; /* allow split payments        */
  long           accountnumber = 1; /* used on etBatchOpen,        */
                                    /* etBatchClosed               */
  unsigned long  batchid;           /* used on etBatchOpen,        */
                                    /* etBatchClosed               */
  char *          PAYMENTTYPE;
  char *          BRAND = NULL;
  char *          PAN = NULL;
  char *          EXPIRY = NULL;

  char *          ETillHostName;
  unsigned long   ETillAdminPort;
  unsigned long   PaymentAPIPort;

  /***************************************************************************
   ** These variables are the fields and structures used by the APIs
***************************************************************************/
  PaymentServerHandle handle;
  Key               keys;              /* Payment Server key structure */
  Amount            orderamount;       /* order amount structure */
  OrderURLs         orderURLs;         /* order URL structure */
  OrderDescription  orderdescription;  /* order description structure */
  ProtocolData      protocoldata;      /* protocol data structure */
  ReturnData        apiReturnValues;   /* PaymentAPI returned data */
  Approve           approveinfo;
  AutoDeposit       autodeposit;
```

```
  /*****************************************************************************
   ** Variables for converting order description to ASCII
 ****************************************************************************/
  static QtqCode_T fromCCSID = { 0, 0, 0, 0, 0, 0 };
  static QtqCode_T toCCSID   = { 0, 0, 0, 0, 0, 0 };
  iconv_t jobCCSIDtoAscii;

  char orderDescriptionString[] = "Sample Order Description";
  int orderDescriptionLength = strlen(orderDescriptionString);
  char convertedString[100];
  char* inBuffer;
  char* outBuffer;
  size_t inBytesLeft, outBytesLeft;

  int finalrc;
  int iconvrc;

  /*****************************************************************************
  ** Input parameters are:
  **
  **   Merchant Number
  **   Order Number
  **   Batch ID
  *****************************************************************************/
  if (argc <= 3)
  {
  printf("\nUsage: CALL SAMPLE1a "
         "PARM('Merchant_number' 'Order_number' )");
    exit(0);
  }
  merchantnumber = atol(argv[1]);
  ordernumber = atol(argv[2]);
  batchid = atol(argv[3]);

 /*****************************************************************************
  **
  ** Set the customization values:
  **
  ** !!! THESE MAY BE UPDATED TO REFLECT THE LOCAL INSTALLATION
  **
  *****************************************************************************/

  PaymentAPIPort = 8611;
  ETillAdminPort = 8614;
  ETillHostName = "your AS/400 name";

 /*****************************************************************************
  ** Initialize the API
  *****************************************************************************/

   handle=etInitializeAPI(0,ETillHostName, PaymentAPIPort,
                          ETillAdminPort, &rc);

   if (rc > 0)
   {
    printf("etInitializeAPI failed, rc = %d\n",rc);
    exit(0);
    }
 /*****************************************************************************
  ** Setup the transaction data
  *****************************************************************************/
```

```
keys.merchantNumber = merchantnumber;
keys.orderNumber = ordernumber;

orderamount.amount = 10000;                    /*$100.00 dollars US */
orderamount.currency = 840;                    /* US Dollars */
orderamount.amountExponent10 = -2;             /* 10000 = 100.00 */

PAYMENTTYPE = "SET";              /* Secure Electronic Transactions */

/*************************************************************************
** The Order description must be converted to ASCII because it needs
** to be sent to the wallet on the browser in some cases.
**
** The following code converts the order description from the job CCSID
** to ASCII.  This is appropriate when the order description is passed
** in as a parameter, or retrieved from a database text field.  In both
** those cases, the order description will be coded in the job CCSID.
**
** In this sample, the orderDescriptionString was initialized in the
** program itself.  In this case, orderDescriptionString is coded in
** the CCSID of the program source file at compile time, and iconv
** converts from the job CCSID (at program run time) to ASCII.  For
** this sample to work correctly, the CCSID of the program source file
** must match the CCSID of the job at program run time.
*************************************************************************/

/* Get a handle to convert from EBCDIC to US ASCII. */
toCCSID.CCSID = 819;
fromCCSID.length_option = 1;  /* iconv determines input length */
jobCCSIDtoAscii = QtqIconvOpen( &toCCSID, &fromCCSID );
if (jobCCSIDtoAscii.return_value != 0 ) {
  /* Error handling here. */
  printf("Cannot get conversion handle, rc = %d",
         jobCCSIDtoAscii.return_value);
  etCloseAPI(&handle);
  exit(0);
}

/* Now convert the characters to ASCII. */
inBytesLeft = 0;                  /* iconv determines input length */
outBytesLeft = sizeof(orderDescriptionString);
inBuffer = orderDescriptionString;
outBuffer = convertedString;
iconvrc = iconv(jobCCSIDtoAscii, &inBuffer, &inBytesLeft,
           &outBuffer, &outBytesLeft);

/* Close and release code conversion descriptor. */
iconv_close(jobCCSIDtoAscii);

if (iconvrc >= 0) {
  orderdescription.contenttypecharset = "text/plain";
  orderdescription.description = convertedString;
  orderdescription.length = strlen(convertedString);
}
else {
  /* Error handling here. */
  printf("EBCDIC to ASCII conversion failed, rc = %d", iconvrc);
  etCloseAPI(&handle);
  exit(0);
}

paymentnumber = 1;           /* increment for split payments      */
accountnumber = 201355;           /* Used on etBatchOpen,etBatchClosed */
```

```
BRAND = "VISA";
PAN = "1233333333333333";   /*       sample "test" CC Number      */
EXPIRY = "199912";

  memset(&protocoldata, 0, sizeof(protocoldata));
  rc = etAddProtocolDataString(&protocoldata, "$BRAND          ",
                               BRAND,strlen(BRAND), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$PAN            ",
                               PAN,strlen(PAN), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$EXPIRY         ",
                               EXPIRY,strlen(EXPIRY), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

/*************************************************************************
 ** Perform the MOP or MIA (etAcceptPayment)
 *************************************************************************/
  rc = etAcceptPayment(handle,
                       &keys,
                       &orderamount,
                       PAYMENTTYPE,
                       &orderdescription,
                       NULL,          /* approveinfo structure      */
                       &protocoldata, /* protocoldata structure     */
                       &rc2,
                       &apiReturnValues);

 if (rc > 0)
 {
  printf("etAcceptPayment failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etAcceptPayment for Order number %d complete\n",
         keys.orderNumber);
 }
/*************************************************************************
 **                  Perform the APPROVE
 *************************************************************************/
rc = etApprove( handle,
                &keys,                      /* key structure       */
                paymentnumber,              /* payment number      */
                &orderamount,               /* amount structure    */
```

```
                splitallowed,    /* further accept_payments allowed */
                NULL,                    /* auto deposit structure */
                NULL,                    /* protocoldata structure */
                &rc2);
   if (rc > 0)
   {
    printf("etApprove failed, rc = %d %d\n",rc, rc2);
    etCloseAPI(&handle);
    exit(0);
    }
   else
   {
    printf("etApprove for Order number %d complete\n",
           keys.orderNumber);
   }

   /*************************************************************************
   **                  DEPOSIT
   *************************************************************************/
       rc = etDeposit( handle,
          &keys,                              /* key structure       */
          paymentnumber,                      /* payment number      */
          &orderamount,                       /* amount structure    */
          NULL,                               /* batch ID            */
          NULL,                              /* protocoldata structure */
          &rc2);                             /* secondary return code  */


   if (rc > 0)
   {
    printf("etDeposit failed, rc = %d %d\n",rc, rc2);
    etCloseAPI(&handle);
    exit(0);
    }
   else
   {
    printf("etDeposit for Order number %d complete\n",
           keys.orderNumber);
    }


    etCloseAPI(&handle);

}
```

3.3.2 Manual approve and deposit with a batch sample

This section shows an example of merchant-initiated authorization processing, in which the shopper *does not* have wallet software. This example applies to MOP and MIA. The merchant approves the order and then performs its own batch administration: opening a batch, depositing the order, and closing the batch.

```
/*************************************************************************
** Source File Name: sample1.C
**
**
** Source File Description:
**   Provides SET(tm) payment processing function using the IBM Payment
**   Server for AS/400.
```

```
**
** To Compile: ADDLIBLE QPYMSVR
**             CRTCMOD MODULE(yourlib/SAMPLE1) SRCFILE(yourlib/SAMPLE1)
**             CRTPGM PGM(yourlib/SAMPLE1) MODULE(yourlib/SAMPLE1)
**                         BNDSRVPGM(QPYMSVR/QZETLPAY)
**
** To Run:     CALL PGM(yourlib/SAMPLE1) PARM('1' '10' '10')
**
** Parameters: Merchant Number, Order Number, Batch ID
**
** Action:
**
**     Accept Payment
**     Approve
**     Batch Open
**
**     Deposit
**
**     Batch Close
**
** Payment Server APIs used: etInitializeAPI
**                           etAddProtocolDataString
**                           etAcceptPayment
**                           etApprove
**                           etBatchOpen
**                           etDeposit
**                           etBatchClose
**                           etCloseAPI
**
** Disclaimer:
**   This sample is furnished by IBM as a simple example to
**   illustrate the use of the IBM Payment Server for AS/400.
**   This example has not been thoroughly tested under all
**   conditions.  IBM, therefore cannot guarantee or imply
**   reliability, serviceability, or function of this program.
**   This program is provided to you "AS IS."  All implied
**   warranties, including but not limited to the implied
**   warranties of merchantability and fitness for a particular
**   purpose, are expressly disclaimed.
**
** SET Secure Electronic Transaction, Secure Electronic Transaction,
** SET, and the SET Secure Electronic Transaction design mark are
** trademarks and service marks owned by SET Secure Electronic
** Transaction LLC. Use of the trademarks without a written license
** from SET Secure Electronic Transaction LLC is strictly prohibited.
**
*****************************************************************************/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <qtqiconv.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include "etillapi.h"


/*****************************************************************************
                                 main()
*****************************************************************************/
```

```
int main(int argc, char *argv[])
{


  long           rc            = -1; /* primary return code         */
  long           rc2           = -1; /* secondary return code       */
  long           merchantnumber;    /* Parameter                   */
  long           ordernumber;       /* Parameter                   */
  long           paymentnumber = 1; /* used on et_Approve and      */
                                    /* et_Deposit                  */
  long           splitallowed  = 1; /* allow split payments        */
  long           accountnumber = 1; /* used on etBatchOpen,        */
                                    /* etBatchClosed               */
  unsigned long  batchid;           /* used on etBatchOpen,        */
                                    /* etBatchClosed               */
  char *          PAYMENTTYPE;
  char *          BRAND = NULL;
  char *          PAN = NULL;
  char *          EXPIRY = NULL;

  char *          ETillHostName;
  unsigned long   ETillAdminPort;
  unsigned long   PaymentAPIPort;

  /*************************************************************************
   ** These variables are the fields and structures used by the APIs
   *************************************************************************/
  PaymentServerHandle handle;
  Key               keys;               /* Payment Server key structure */
  Amount            orderamount;        /* order amount structure */
  OrderURLs         orderURLs;          /* order URL structure */
  OrderDescription  orderdescription;   /* order description structure */
  ProtocolData      protocoldata;       /* protocol data structure */
  ReturnData        apiReturnValues;    /* PaymentAPI returned data */
  Approve           approveinfo;
  AutoDeposit       autodeposit;

  /*************************************************************************
   ** Variables for converting order description to ASCII
   *************************************************************************/
  static QtqCode_T fromCCSID = { 0, 0, 0, 0, 0, 0 };
  static QtqCode_T toCCSID   = { 0, 0, 0, 0, 0, 0 };
  iconv_t jobCCSIDtoAscii;

  char orderDescriptionString[] = "Sample Order Description";
  int orderDescriptionLength = strlen(orderDescriptionString);
  char convertedString[100];
  char* inBuffer;
  char* outBuffer;
  size_t inBytesLeft, outBytesLeft;

  int finalrc;
  int iconvrc;

  /*************************************************************************
  ** Input parameters are:
  **
  **   Merchant Number
  **   Order Number
  **   Batch ID
  *************************************************************************/
  if (argc <= 3)
  {
```

```
 printf("\nUsage: CALL SAMPLE1 "
        "PARM('Merchant_number' 'Order_number' 'Batch_ID')");
   exit(0);
 }
 merchantnumber = atol(argv[1]);
 ordernumber = atol(argv[2]);
 batchid = atol(argv[3]);

/*****************************************************************************
 **
 ** Set the customization values:
 **
 ** !!! THESE MAY BE UPDATED TO REFLECT THE LOCAL INSTALLATION
 **
 *****************************************************************************/

 PaymentAPIPort = 8611;
 ETillAdminPort = 8614;
 ETillHostName = your AS/400 name;

/*****************************************************************************
 ** Initialize the API
 *****************************************************************************/

  handle=etInitializeAPI(0,ETillHostName, PaymentAPIPort,
                         ETillAdminPort, &rc);

  if (rc > 0)
  {
   printf("etInitializeAPI failed, rc = %d\n",rc);
   exit(0);
   }
/*****************************************************************************
 ** Setup the transaction data
 *****************************************************************************/

 keys.merchantNumber = merchantnumber;
 keys.orderNumber = ordernumber;

 orderamount.amount = 10000;                    /*$100.00 dollars US */
 orderamount.currency = 840;                    /* US Dollars */
 orderamount.amountExponent10 = -2;             /* 10000 = 100.00 */

 PAYMENTTYPE = "SET";              /* Secure Electronic Transactions */

/*****************************************************************************
 ** The Order description must be converted to ASCII because it needs
 ** to be sent to the wallet on the browser in some cases.
 **
 ** The following code converts the order description from the job CCSID
 ** to ASCII.  This is appropriate when the order description is passed
 ** in as a parameter, or retrieved from a database text field.  In both
 ** those cases, the order description will be coded in the job CCSID.
 **
 ** In this sample, the orderDescriptionString was initialized in the
 ** program itself.  In this case, orderDescriptionString is coded in
 ** the CCSID of the program source file at compile time, and iconv
 ** converts from the job CCSID (at program run time) to ASCII.  For
 ** this sample to work correctly, the CCSID of the program source file
 ** must match the CCSID of the job at program run time.
 *****************************************************************************/

 /* Get a handle to convert from EBCDIC to US ASCII. */
```

```
toCCSID.CCSID = 819;
fromCCSID.length_option = 1;  /* iconv determines input length */
jobCCSIDtoAscii = QtqIconvOpen( &toCCSID, &fromCCSID );
if (jobCCSIDtoAscii.return_value != 0 ) {
  /* Error handling here. */
  printf("Cannot get conversion handle, rc = %d",
         jobCCSIDtoAscii.return_value);
  etCloseAPI(&handle);
  exit(0);
}

/* Now convert the characters to ASCII. */
inBytesLeft = 0;                  /* iconv determines input length */
outBytesLeft = sizeof(orderDescriptionString);
inBuffer = orderDescriptionString;
outBuffer = convertedString;
iconvrc = iconv(jobCCSIDtoAscii, &inBuffer, &inBytesLeft,
           &outBuffer, &outBytesLeft);

/* Close and release code conversion descriptor. */
iconv_close(jobCCSIDtoAscii);

if (iconvrc >= 0) {
  orderdescription.contenttypecharset = "text/plain";
  orderdescription.description = convertedString;
  orderdescription.length = strlen(convertedString);
}
else {
  /* Error handling here. */
  printf("EBCDIC to ASCII conversion failed, rc = %d", iconvrc);
  etCloseAPI(&handle);
  exit(0);
}

paymentnumber = 1;           /* increment for split payments      */
accountnumber = 201355;           /* Used on etBatchOpen,etBatchClosed */
BRAND = "VISA";
PAN = "1233333333333333";   /*       sample "test" CC Number      */
EXPIRY = "199912";

  memset(&protocoldata, 0, sizeof(protocoldata));
  rc = etAddProtocolDataString(&protocoldata, "$BRAND          ",
                               BRAND,strlen(BRAND), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$PAN            ",
                               PAN,strlen(PAN), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$EXPIRY         ",
                               EXPIRY,strlen(EXPIRY), &rc2);
 if (rc > 0)
 {
```

```
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

/*************************************************************************
 ** Perform the MOP or MIA (etAcceptPayment)
 ************************************************************************/
  rc = etAcceptPayment(handle,
                       &keys,
                       &orderamount,
                       PAYMENTTYPE,
                       &orderdescription,
                       NULL,          /* approveinfo structure      */
                       &protocoldata, /* protocoldata structure     */
                       &rc2,
                       &apiReturnValues);

 if (rc > 0)
 {
  printf("etAcceptPayment failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etAcceptPayment for Order number %d complete\n",
         keys.orderNumber);
 }
/*************************************************************************
 **                  Perform the APPROVE
 ************************************************************************/
rc = etApprove( handle,
                &keys,                        /* key structure       */
                paymentnumber,                /* payment number      */
                &orderamount,                 /* amount structure    */
                splitallowed,    /* further accept_payments allowed */
                NULL,                      /* auto deposit structure */
                NULL,                      /* protocoldata structure */
                &rc2);
 if (rc > 0)
 {
  printf("etApprove failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etApprove for Order number %d complete\n",
         keys.orderNumber);
 }
/*************************************************************************
 **                  BATCH OPEN prior to DEPOSIT
 ************************************************************************/
 rc = etBatchOpen( handle,
    merchantnumber,               /* merchant number for this batch */
    accountnumber,                /* account number                 */
    batchid,                      /* batch ID                       */
    PAYMENTTYPE,             /* payment type for this batch e.g. SET */
    NULL,                    /* protocoldata structure               */
    &rc2);                   /* secondary return code                */
```

```
  if (rc > 0)
  {
   printf("etBatchOpen failed, rc = %d %d\n",rc, rc2);
   etCloseAPI(&handle);
   exit(0);
   }
  else
  {
   printf("etBatchOpen for Batch ID %d complete\n",
          batchid);
  }
  /*************************************************************************
  **                    DEPOSIT
  *************************************************************************/
      rc = etDeposit( handle,
         &keys,                                /* key structure       */
         paymentnumber,                        /* payment number      */
         &orderamount,                         /* amount structure    */
         &batchid,                             /* batch ID            */
         NULL,                               /* protocoldata structure */
         &rc2);                              /* secondary return code  */


  if (rc > 0)
  {
   printf("etDeposit failed, rc = %d %d\n",rc, rc2);
   etCloseAPI(&handle);
   exit(0);
   }
  else
  {
   printf("etDeposit for Order number %d complete\n",
          keys.orderNumber);
   }
 /*************************************************************************
  **                 BATCH CLOSE
  *************************************************************************/
  rc = etBatchClose( handle,
     merchantnumber,               /* merchant number for this batch */
     accountnumber,                /* account number                 */
     batchid,                      /* batch ID                       */
     NULL,                         /* protocoldata structure         */
     &rc2);                        /* secondary return code          */

  if (rc > 0)
  {
   printf("etBatchClose failed, rc = %d %d\n",rc, rc2);
   etCloseAPI(&handle);
   exit(0);
   }
  else
  {
   printf("etBatchClose for Batch ID %d complete\n",
          batchid);
  }

   etCloseAPI(&handle);

}
```

3.3.3 Auto approve and deposit sample

This section shows an example of merchant-initiated authorization processing, in which the shopper *does not* have wallet software, using auto APPROVE and the DEPOSIT. This example applies to MOP and MIA and shows that an acquirer controls the batch, so that the sample does not issue batch operation APIs, such as etBatchOpen() and etBatchClose().

```
/*****************************************************************************
** Source File Name: sample2.C
**
**
** Source File Description:
**   Provides SET(tm) payment processing function using the IBM Payment
**   Server for AS/400.
**
** To Compile: ADDLIBLE QPYMSVR
**             CRTCMOD MODULE(yourlib/SAMPLE2) SRCFILE(yourlib/QCSRC)
**             CRTPGM PGM(yourlib/SAMPLE2) MODULE(yourlib/SAMPLE2)
**                          BNDSRVPGM(QPYMSVR/QZETLPAY)
**
** To Run:     CALL PGM(yourlib/SAMPLE2) PARM('1' '10' '10')
**
** Parameters: Merchant Number, Order Number, Batch ID
**
** Action:
**
**     Batch Open
**     Accept Payment (with approve and deposit flags on)
**     Batch Close
**
**
** Payment Server APIs used: etInitializeAPI
**                           etAddProtocolDataString
**                           etAcceptPayment
**                           etBatchOpen
**                           etBatchClose
**                           etCloseAPI
**
** Disclaimer:
**   This sample is furnished by IBM as a simple example to
**   illustrate the use of the IBM Payment Server for AS/400.
**   This example has not been thoroughly tested under all
**   conditions.  IBM, therefore cannot guarantee or imply
**   reliability, serviceability, or function of this program.
**   This program is provided to you "AS IS."  All implied
**   warranties, including but not limited to the implied
**   warranties of merchantability and fitness for a particular
**   purpose, are expressly disclaimed.
**
** SET Secure Electronic Transaction, Secure Electronic Transaction,
** SET, and the SET Secure Electronic Transaction design mark are
** trademarks and service marks owned by SET Secure Electronic
** Transaction LLC. Use of the trademarks without a written license
** from SET Secure Electronic Transaction LLC is strictly prohibited.
**
*****************************************************************************/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <stdarg.h>
#include <qtqiconv.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include "etillapi.h"



/*****************************************************************************
                              main()
 *****************************************************************************/

int main(int argc, char *argv[])
{


  long           rc            = -1; /* primary return code         */
  long           rc2           = -1; /* secondary return code       */
  long           merchantnumber;    /* Parameter                   */
  long           ordernumber;       /* Parameter                   */
  long           paymentnumber = 1; /* used on et_Approve and      */
                                    /* et_Deposit                  */
  long           splitallowed  = 1; /* allow split payments        */
  long           accountnumber = 1; /* used on etBatchOpen,        */
                                    /* etBatchClosed               */
  unsigned long  batchid;           /* used on etBatchOpen,        */
                                    /* etBatchClosed               */

  char *          PAYMENTTYPE;
  char *          BRAND = NULL;
  char *          PAN = NULL;
  char *          EXPIRY = NULL;

  char *          ETillHostName;
  unsigned long   ETillAdminPort;
  unsigned long   PaymentAPIPort;

  /*****************************************************************************
   **These variables are the fields and structures used by the APIs
   *****************************************************************************/
  PaymentServerHandle handle;
  Key               keys;               /* Payment Server key structure */
  Amount            orderamount;        /* order amount structure */
  OrderURLs         orderURLs;          /* order URL structure */
  OrderDescription  orderdescription;   /* order description structure */
  ProtocolData      protocoldata;       /* protocol data structure */
  ReturnData        apiReturnValues;    /* PaymentAPI returned data */
  Approve           approveinfo;
  AutoDeposit       autodeposit;

  /*****************************************************************************
   **Variables for converting order description to ASCII
   *****************************************************************************/
  static QtqCode_T fromCCSID = { 0, 0, 0, 0, 0, 0 };
  static QtqCode_T toCCSID   = { 0, 0, 0, 0, 0, 0 };
  iconv_t jobCCSIDtoAscii;

  char orderDescriptionString[] = "Sample Order Description";
  int orderDescriptionLength = strlen(orderDescriptionString);
  char convertedString[100];
  char* inBuffer;
  char* outBuffer;
```

```
size_t inBytesLeft, outBytesLeft;

int finalrc;
int iconvrc;

/*********************************************************************
** Input parameters are:
**
**   Merchant Number
**   Order Number
**   Batch ID
*********************************************************************/
if (argc <= 3)
{
printf("\nUsage: CALL SAMPLE2 "
       "PARM('Merchant_number' 'Order_number' 'Batch_ID')");
  exit(0);
}
merchantnumber = atol(argv[1]);
ordernumber = atol(argv[2]);
batchid = atol(argv[3]);

/*********************************************************************
**
** Set the customization values:
**
** !!! THESE MAY BE UPDATED TO REFLECT THE LOCAL INSTALLATION
**
*********************************************************************/
PaymentAPIPort = 8611;
ETillAdminPort = 8614;
ETillHostName = your AS/400;

/*********************************************************************
** Initialize the API
*********************************************************************/

 handle=etInitializeAPI(0,ETillHostName, PaymentAPIPort,
                        ETillAdminPort, &rc);
 if (rc > 0)
 {
  printf("etInitializeAPI failed, rc = %d\n",rc);
  etCloseAPI(&handle);
  exit(0);
  }

/*********************************************************************
** Setup the transaction data
*********************************************************************/

keys.merchantNumber = merchantnumber;
keys.orderNumber = ordernumber;

orderamount.amount = 10000;                    /*$100.00 dollars US */
orderamount.currency = 840;                    /* US Dollars */
orderamount.amountExponent10 = -2;             /* 10000 = 100.00 */

PAYMENTTYPE = "SET";              /* Secure Electronic Transactions */

/*********************************************************************
** The Order description must be converted to ASCII because it needs
** to be sent to the wallet on the browser in some cases.
**
```

```
** The following code converts the order description from the job CCSID
** to ASCII.  This is appropriate when the order description is passed
** in as a parameter, or retrieved from a database text field.  In both
** those cases the order description will be coded in the job CCSID.
**
** In this sample, the orderDescriptionString was initialized in the
** program itself.  In this case, orderDescriptionString is coded in
** the CCSID of the program source file at compile time, and iconv
** converts from the job CCSID (at program run time) to ASCII.  For
** this sample to work correctly, the CCSID of the program source file
** must match the CCSID of the job at program run time.
*********************************************************************/

/* Get a handle to convert from EBCDIC to US ASCII. */
toCCSID.CCSID = 819;
fromCCSID.length_option = 1;  /* iconv determines input length */
jobCCSIDtoAscii = QtqIconvOpen( &toCCSID, &fromCCSID );
if (jobCCSIDtoAscii.return_value != 0 ) {
  /* Error handling here. */
  printf("Cannot get conversion handle, rc = %d",
         jobCCSIDtoAscii.return_value);
  etCloseAPI(&handle);
  exit(0);
}

/* Now convert the characters to ASCII. */
inBytesLeft = 0;
outBytesLeft = sizeof(orderDescriptionString);
inBuffer = orderDescriptionString;
outBuffer = convertedString;
iconvrc = iconv(jobCCSIDtoAscii, &inBuffer, &inBytesLeft,
           &outBuffer, &outBytesLeft);

/* Close and release code conversion descriptor. */
iconv_close(jobCCSIDtoAscii);

if (iconvrc >= 0) {
  orderdescription.contenttypecharset = "text/plain";
  orderdescription.description = convertedString;
  orderdescription.length = strlen(convertedString);
}
else {
  /* Error handling here. */
  printf("EBCDIC to ASCII conversion failed, rc = %d", iconvrc);
  etCloseAPI(&handle);
  exit(0);
}


paymentnumber = 1;         /* increment for split payments        */
accountnumber = 123456789;         /* Used on Batch_Open, Batch_Closed    */
BRAND = "VISA";
PAN = "1122334455";         /* sample "test" CC Number      */
EXPIRY = "199912";

  memset(&protocoldata, 0, sizeof(protocoldata));
  rc=etAddProtocolDataString(&protocoldata, "$BRAND          ",
                             BRAND,strlen(BRAND), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
```

```
 }

 rc=etAddProtocolDataString( &protocoldata, "$PAN            ",
                             PAN,strlen(PAN), &rc2);
if (rc > 0)
{
 printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
 etCloseAPI(&handle);
 exit(0);
 }

 rc=etAddProtocolDataString( &protocoldata, "$EXPIRY         ",
                             EXPIRY,strlen(EXPIRY), &rc2);
if (rc > 0)
{
 printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
 etCloseAPI(&handle);
 exit(0);
 }

 autodeposit.batchID = &batchid;

 approveinfo.paymentnumber = paymentnumber;
 approveinfo.paymentamount = orderamount.amount;
 approveinfo.splitallowed = splitallowed;
 approveinfo.deposit = &autodeposit;

/*************************************************************************
 **        Batch Open prior to autoapprove and autodeposit
 ************************************************************************/
 rc=etBatchOpen( handle,
    merchantnumber,          /* merchant number for this batch      */
    accountnumber,           /* account number                      */
    batchid,                 /* batch ID                            */
    PAYMENTTYPE,         /* payment type for this batch e.g. SET(tm)*/
    NULL,                /* protocoldata structure                  */
    &rc2);               /* secondary return code                   */


 if (rc > 0)
 {
  printf("etBatchOpen failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etBatchOpen for Batch ID %d complete\n",
         batchid);
 }
/*************************************************************************
 ** Perform the MOP and MIA (etAcceptPayment)
 ** with APPROVE and DEPOSIT flags on
 ************************************************************************/
  rc = etAcceptPayment(handle,
                       &keys,
                       &orderamount,
                       PAYMENTTYPE,
                       &orderdescription,
                       &approveinfo,   /* approveinfo structure     */
                       &protocoldata,  /* protocoldata structure    */
                       &rc2,
                       &apiReturnValues);
```

```
   if (rc > 0)
   {
    printf("etAcceptPayment failed, rc = %d %d\n",rc, rc2);
    etCloseAPI(&handle);
    exit(0);
    }
   else
   {
    printf("etAcceptPayment for Order number %d complete\n",
           keys.orderNumber);
   }
  /*************************************************************************
   **               BATCH CLOSE
   *************************************************************************/
   rc=etBatchClose( handle,
      merchantnumber,              /* merchant number for this batch */
      accountnumber,               /* account number                 */
      batchid,                     /* batch ID                       */
      NULL,                        /* protocoldata structure         */
      &rc2);                       /* secondary return code          */

   if (rc > 0)
   {
    printf("etBatchClose failed, rc = %d %d\n",rc, rc2);
    etCloseAPI(&handle);
    exit(0);
    }
   else
   {
    printf("etBatchClose for Batch ID %d complete\n",
           batchid);
   }

    etCloseAPI(&handle);

}
```

3.3.4 Credit sample

This section shows an example of merchant-initiated authorization processing, in which the shopper *does not* have wallet software. This example applies to MOP and MIA. The merchant approves the order and then performs its own batch administration, including opening a batch, depositing the order, and closing the batch. Then, the merchant opens a batch (for refund), performs a refund, and closes the batch.

```
/*****************************************************************************
** Source File Name: sample3.C
**
**
** Source File Description:
**   Provides SET(tm) payment processing function using the IBM Payment
**   Server for AS/400.
**
** To Compile: ADDLIBLE QPYMSVR
**             CRTCMOD MODULE(yourlib/sample3) SRCFILE(yourlib/QCSRC)
**             CRTPGM PGM(yourlib/sample3) MODULE(yourlib/sample3)
**                         BNDSRVPGM(QPYMSVR/QZETLPAY)
**
```

```
** To Run:     CALL PGM(yourlib/sample3) PARM('1' '10' '10')
**
** Parameters: Merchant Number, Order Number, Batch ID
**
** Action:
**
**     Accept Payment
**     Approve
**     Batch Open
**      Deposit
**     Batch Close
**
**     Batch Open
**      Refund
**     Batch Close
**
** Payment Server APIs used: etInitializeAPI
**                           etAddProtocolDataString
**                           etAcceptPayment
**                           etApprove
**                           etBatchOpen
**                           etDeposit
**                           etBatchClose
**                           etRefund
**                           etCloseAPI
**
** Disclaimer:
**   This sample is furnished by IBM as a simple example to
**   illustrate the use of the IBM Payment Server for AS/400.
**   This example has not been thoroughly tested under all
**   conditions.  IBM, therefore cannot guarantee or imply
**   reliability, serviceability, or function of this program.
**   This program is provided to you "AS IS."  All implied
**   warranties, including but not limited to the implied
**   warranties of merchantability and fitness for a particular
**   purpose, are expressly disclaimed.
**
** SET Secure Electronic Transaction, Secure Electronic Transaction,
** SET, and the SET Secure Electronic Transaction design mark are
** trademarks and service marks owned by SET Secure Electronic
** Transaction LLC. Use of the trademarks without a written license
** from SET Secure Electronic Transaction LLC is strictly prohibited.
**
*****************************************************************************/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <qtqiconv.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include "etillapi.h"


/*****************************************************************************
                                 main()
 *****************************************************************************/

int main(int argc, char *argv[])
{
```

```
long           rc            = -1; /* primary return code         */
long           rc2           = -1; /* secondary return code       */
long           merchantnumber;    /* Parameter                   */
long           ordernumber;       /* Parameter                   */
long           paymentnumber = 1; /* used on et_Approve and      */
                                  /* et_Deposit                  */
long           creditnumber  = 1; /* used on etRefund            */
long           splitallowed  = 1; /* allow split payments        */
long           accountnumber = 1; /* used on etBatchOpen,        */
                                  /* etBatchClosed               */
unsigned long  batchid;           /* used on etBatchOpen,        */
                                  /* etBatchClosed               */
char *          PAYMENTTYPE;
char *          BRAND = NULL;
char *          PAN = NULL;
char *          EXPIRY = NULL;

char *          ETillHostName;
unsigned long   ETillAdminPort;
unsigned long   PaymentAPIPort;

/*************************************************************************
 **These variables are the fields and structures used by the APIs
 *************************************************************************/
PaymentServerHandle handle;
Key               keys;               /* Payment Server key structure */
Amount            orderamount;        /* order amount structure */
OrderURLs         orderURLs;          /* order URL structure */
OrderDescription  orderdescription;   /* order description structure */
ProtocolData      protocoldata;       /* protocol data structure */
ReturnData        apiReturnValues;    /* PaymentAPI returned data */
Approve           approveinfo;
AutoDeposit       autodeposit;

/*************************************************************************
 ** Variables for converting order description to ASCII
 *************************************************************************/
static QtqCode_T fromCCSID = { 0, 0, 0, 0, 0, 0 };
static QtqCode_T toCCSID   = { 0, 0, 0, 0, 0, 0 };
iconv_t jobCCSIDtoAscii;

char orderDescriptionString[] = "Sample Order Description";
int orderDescriptionLength = strlen(orderDescriptionString);
char convertedString[100];
char* inBuffer;
char* outBuffer;
size_t inBytesLeft, outBytesLeft;

int finalrc;
int iconvrc;

/*************************************************************************
** Input parameters are:
**
**   Merchant Number
**   Order Number
**   Batch ID
*************************************************************************/
if (argc <= 3)
{
printf("\nUsage: CALL sample3 "
       "PARM('Merchant_number' 'Order_number' 'Batch_ID')");
```

```
  exit(0);
 }
 merchantnumber = atol(argv[1]);
 ordernumber = atol(argv[2]);
 batchid = atol(argv[3]);

/*************************************************************************
 **
 ** Set the customization values:
 **
 ** !!! THESE MAY BE UPDATED TO REFLECT THE LOCAL INSTALLATION
 **
 *************************************************************************/

 PaymentAPIPort = 8611;
 ETillAdminPort = 8614;
 ETillHostName = "as05";

/*************************************************************************
 ** Initialize the API
 *************************************************************************/

  handle=etInitializeAPI(0,ETillHostName, PaymentAPIPort,
                         ETillAdminPort, &rc);

  if (rc > 0)
  {
   printf("etInitializeAPI failed, rc = %d\n",rc);
   exit(0);
   }
/*************************************************************************
 ** Setup the transaction data
 *************************************************************************/

 keys.merchantNumber = merchantnumber;
 keys.orderNumber = ordernumber;

 orderamount.amount = 10000;                    /*$100.00 dollars US */
 orderamount.currency = 840;                    /* US Dollars */
 orderamount.amountExponent10 = -2;             /* 10000 = 100.00 */

 PAYMENTTYPE = "SET";              /* Secure Electronic Transactions */

/*************************************************************************
 ** The Order description must be converted to ASCII because it needs
 ** to be sent to the wallet on the browser in some cases.
 **
 ** The following code converts the order description from the job CCSID
 ** to ASCII.  This is appropriate when the order description is passed
 ** in as a parameter, or retrieved from a database text field.  In both
 ** those cases, the order description will be coded in the job CCSID.
 **
 ** In this sample, the orderDescriptionString was initialized in the
 ** program itself.  In this case, orderDescriptionString is coded in
 ** the CCSID of the program source file at compile time, and iconv
 ** converts from the job CCSID (at program run time) to ASCII.  For
 ** this sample to work correctly, the CCSID of the program source file
 ** must match the CCSID of the job at program run time.
 *************************************************************************/

 /* Get a handle to convert from EBCDIC to US ASCII. */
 toCCSID.CCSID = 819;
 fromCCSID.length_option = 1; /* iconv determines input length */
```

```
jobCCSIDtoAscii = QtqIconvOpen( &toCCSID, &fromCCSID );
if (jobCCSIDtoAscii.return_value != 0 ) {
  /* Error handling here. */
  printf("Cannot get conversion handle, rc = %d",
         jobCCSIDtoAscii.return_value);
  etCloseAPI(&handle);
  exit(0);
}

/* Now convert the characters to ASCII. */
inBytesLeft = 0;  /* iconv determines input length */
outBytesLeft = sizeof(orderDescriptionString);
inBuffer = orderDescriptionString;
outBuffer = convertedString;
iconvrc = iconv(jobCCSIDtoAscii, &inBuffer, &inBytesLeft,
           &outBuffer, &outBytesLeft);

/* Close and release code conversion descriptor. */
iconv_close(jobCCSIDtoAscii);

if (iconvrc >= 0) {
  orderdescription.contenttypecharset = "text/plain";
  orderdescription.description = convertedString;
  orderdescription.length = strlen(convertedString);
}
else {
  /* Error handling here. */
  printf("EBCDIC to ASCII conversion failed, rc = %d", iconvrc);
  etCloseAPI(&handle);
  exit(0);
}

paymentnumber = 1;           /* increment for split payments      */
accountnumber = 123456789;           /* Used on etBatchOpen,etBatchClosed */
BRAND = "VISA";
PAN = "1122334455";         /* sample "test" CC Number      */
EXPIRY = "199912";

  memset(&protocoldata, 0, sizeof(protocoldata));
  rc = etAddProtocolDataString(&protocoldata, "$BRAND          ",
                               BRAND,strlen(BRAND), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$PAN            ",
                               PAN,strlen(PAN), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
  exit(0);
  }

  rc = etAddProtocolDataString( &protocoldata, "$EXPIRY         ",
                               EXPIRY,strlen(EXPIRY), &rc2);
 if (rc > 0)
 {
  printf("etAddProtocolDataString failed, rc = %d %d\n",rc,rc2);
  etCloseAPI(&handle);
```

```
  exit(0);
  }

/*************************************************************************
 ** Perform the MOP and MIA (etAcceptPayment)
 ************************************************************************/
  rc = etAcceptPayment(handle,
                       &keys,
                       &orderamount,
                       PAYMENTTYPE,
                       &orderdescription,
                       NULL,          /* approveinfo structure      */
                       &protocoldata, /* protocoldata structure     */
                       &rc2,
                       &apiReturnValues);

 if (rc > 0)
 {
  printf("etAcceptPayment failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etAcceptPayment for Order number %d complete\n",
         keys.orderNumber);
 }
/*************************************************************************
 **                  Perform the APPROVE
 ************************************************************************/
rc = etApprove( handle,
                &keys,                       /* key structure       */
                paymentnumber,               /* payment number      */
                &orderamount,                /* amount structure    */
                splitallowed,    /* further accept_payments allowed */
                NULL,                     /* auto deposit structure */
                NULL,                     /* protocoldata structure */
                &rc2);
 if (rc > 0)
 {
  printf("etApprove failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etApprove for Order number %d complete\n",
         keys.orderNumber);
 }
/*************************************************************************
 **                  BATCH OPEN prior to DEPOSIT
 ************************************************************************/
 rc = etBatchOpen( handle,
    merchantnumber,               /* merchant number for this batch */
    accountnumber,                /* account number                 */
    batchid,                      /* batch ID                       */
    PAYMENTTYPE,             /* payment type for this batch e.g. SET */
    NULL,                    /* protocoldata structure              */
    &rc2);                   /* secondary return code               */


 if (rc > 0)
 {
```

```
 printf("etBatchOpen failed, rc = %d %d\n",rc, rc2);
 etCloseAPI(&handle);
 exit(0);
 }
else
{
 printf("etBatchOpen for Batch ID %d complete\n",
        batchid);
}
/*************************************************************************
**                    DEPOSIT
*************************************************************************/
    rc = etDeposit( handle,
       &keys,                                /* key structure       */
       paymentnumber,                        /* payment number      */
       &orderamount,                         /* amount structure    */
       &batchid,                             /* batch ID            */
       NULL,                               /* protocoldata structure */
       &rc2);                              /* secondary return code  */


if (rc > 0)
{
 printf("etDeposit failed, rc = %d %d\n",rc, rc2);
 etCloseAPI(&handle);
 exit(0);
 }
else
{
 printf("etDeposit for Order number %d complete\n",
        keys.orderNumber);
 }
/*************************************************************************
**                   BATCH CLOSE
*************************************************************************/
rc = etBatchClose( handle,
   merchantnumber,               /* merchant number for this batch */
   accountnumber,                /* account number                 */
   batchid,                      /* batch ID                       */
   NULL,                         /* protocoldata structure         */
   &rc2);                        /* secondary return code          */

if (rc > 0)
{
 printf("etBatchClose failed, rc = %d %d\n",rc, rc2);
 etCloseAPI(&handle);
 exit(0);
 }
else
{
 printf("etBatchClose for Batch ID %d complete\n",
        batchid);
}

/*************************************************************************
**                   BATCH OPEN prior to refund
*************************************************************************/
batchid++;                                  /* Use the next batchid */
rc = etBatchOpen( handle,
   merchantnumber,               /* merchant number for this batch */
   accountnumber,                /* account number                 */
   batchid,                      /* batch ID                       */
   PAYMENTTYPE,         /* payment type for this batch e.g. SET(tm) */
```

```
   NULL,               /* protocoldata structure                  */
   &rc2);              /* secondary return code                   */


 if (rc > 0)
 {
  printf("etBatchOpen failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etBatchOpen for Batch ID %d complete\n",
         batchid);
 }
 /*************************************************************************
 **                   REFUND
 *************************************************************************/
 rc = etRefund(       handle,
                      &keys,
                      creditnumber,           /* credit number */
                      &orderamount,           /* amount structure */
                      &batchid,               /* batch ID */
                      NULL,                   /* protocoldata structure */
                      &rc2);                  /* secondary return code, */


 if (rc > 0)
 {
  printf("etRefund failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etRefund for Order number %d complete\n",
         keys.orderNumber);
  }
/*************************************************************************
 **                  BATCH CLOSE
 *************************************************************************/
 rc = etBatchClose( handle,
    merchantnumber,              /* merchant number for this batch */
    accountnumber,               /* account number                 */
    batchid,                     /* batch ID                       */
    NULL,                        /* protocoldata structure         */
    &rc2);                       /* secondary return code          */

 if (rc > 0)
 {
  printf("etBatchClose failed, rc = %d %d\n",rc, rc2);
  etCloseAPI(&handle);
  exit(0);
  }
 else
 {
  printf("etBatchClose for Batch ID %d complete\n",
         batchid);
 }

  etCloseAPI(&handle);

}
```

Chapter 4. Setting up the network

This chapter contains the information that you must know to implement the infrastructure for your merchant server with Payment Server. In this chapter, we assume that a Payment Server is installed in the same system in which the merchant server is installed. Before you implement your merchant server solution, you must carefully plan how you are going to connect to the Internet, protect your recourses, and connect your merchant server with your back-end system. Plus, you must install all of the necessary program products.

4.1 Security

It is important that your implementation follows your company security policy. Here are some examples from different parts of a security policy.

4.1.1 General I/T security policy statement

Normally, an I/T security policy has several pages. Here, we provide only small parts from an example I/T security policy. In the following example, we call the company *Mycompany*.

- A Mycompany Information System (IS) is any automated information or telecommunications system owned, leased, or operated by Mycompany.
- Mycompany will implement at least the minimum security requirements, as identified in this policy, to protect IS resources and information (non-sensitive and sensitive data) processed, stored, or transmitted by a Mycompany IS. Based on risk management, they may apply additional safeguards to provide the most restrictive set of controls (privileges) that permit the performance of authorized tasks (principle of least-privilege).
- Sensitive information in any Mycompany IS must be safeguarded against unauthorized disclosure, modification, access, use, destruction, or delay in service.
- Any IS that processes, stores, or transmits sensitive information must be accredited.
- Connectivity is prohibited between Mycompany IS and any other systems or networks not under Mycompany authority, unless formally approved by an appropriate Company Accrediting Authority.
- Any Mycompany IS is for Mycompany business only, and users have no expectation of privacy while using these resources.
- All persons who use, manage, operate, maintain, or develop a Mycompany IS, applications, or data must comply with these policies.

4.1.2 Internet services policy

Any Mycompany owned or controlled IS may only access the Internet through Mycompany approved gateways. This limitation means that Mycompany owned, controlled, or authorized computer equipment, regardless of its location or means of connection to any network or system, may not be used to access the Internet, directly or indirectly. The only way is if the connection is through a Mycompany-approved Internet gateway (firewall).

While the configuration of some networks make it technically possible to access the Internet without going through an approved gateway, such access is not authorized. Exceptions to this policy must be approved in writing by the Director of the Telecommunications Department.

4.2 Server placement

When implementing a merchant server solution, you have to follow your company's security policy. In our example, we only allow a connection to the Internet through a firewall. Since we must protect our data on both the backend system and the merchant server, we have to protect it with a firewall.

We use private IP addresses on our example network. Private IP addresses cannot be used on the Internet. Internet routers do not route private IP addresses. The easiest way to make our merchant server on our private network visible on the Internet is to use Network Address Translation (NAT) in the firewall. We are not going to let our merchant server be directly attached to the Internet.

By placing the merchant server behind the firewall, you can:

- Allow Internet clients to access the merchant server
- Allow the backend system to update the merchant server database

4.3 Firewall

Since we must protect our data on the merchant server and on our backend system, we must implement a firewall solution. In our example, we use Firewall for AS/400 to protect us and our network.

Because the merchant server is a public server and accessed from the Internet, it needs a public address. We use NAT to map a private address to a public address, for use over the Internet. We use the non-secure port of the firewall as the public address for the merchant server.

This section describes the tasks that you must perform to install and configure a firewall using NAT. Figure 51 shows our network configuration for this scenario.

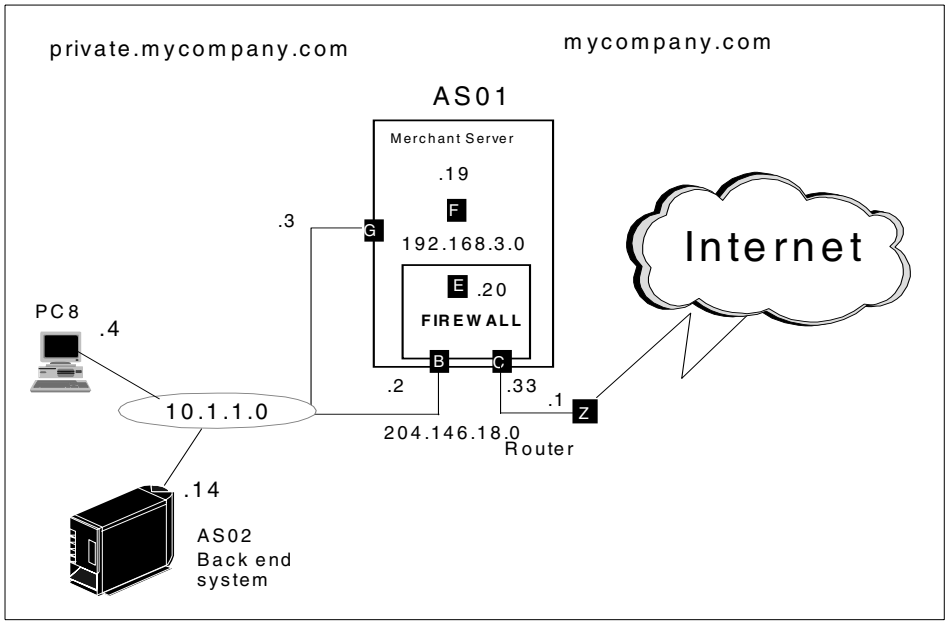


Figure 51. Scenario network configuration

Our scenario configuration includes two AS/400 servers in the mycompany.com network. AS01 houses the firewall, as well as a merchant server, behind the firewall. Internet clients access the public Web server by using the same IP address as the non-secure port of the firewall, 204.146.18.33. The Web server IP address is actually the AS/400 system *INTERNAL port IP address 192.168.3.19 (F). You can use NAT to map the private address to the public one. AS02 is the backend system in the secure network.

4.3.1 Task summary

The following list summarizes the tasks used to implement this NAT environment:

1. Install the firewall and start it successfully.
2. Perform basic configuration for the local firewall. Select the services that you want your internal users to have on the Internet (HTTP and mail, for

example). Select a public HTTP server behind the firewall. In our scenario, we do not allow internal users to connect to the Internet.

3. Start NAT.
4. Add a default route to AS01 TCP/IP configuration pointing to the *INTERNAL port of the firewall as the next hop to enable responses from the HTTP server behind the firewall to the Internet clients.
5. Restart the filters.
6. Verify the following items:
   – An Internet user can open a Web page on the merchant server behind the firewall.
   – Internal clients can use SOCKS and Proxy to open a Web page on the Internet (this is optional).

4.3.2 Installing the AS/400 Firewall

Install the firewall at the local site using the instructions in the manual *Getting Started with IBM Firewall for AS/400*, SC41-5424. A summary of the installation parameters is shown in Figure 52.

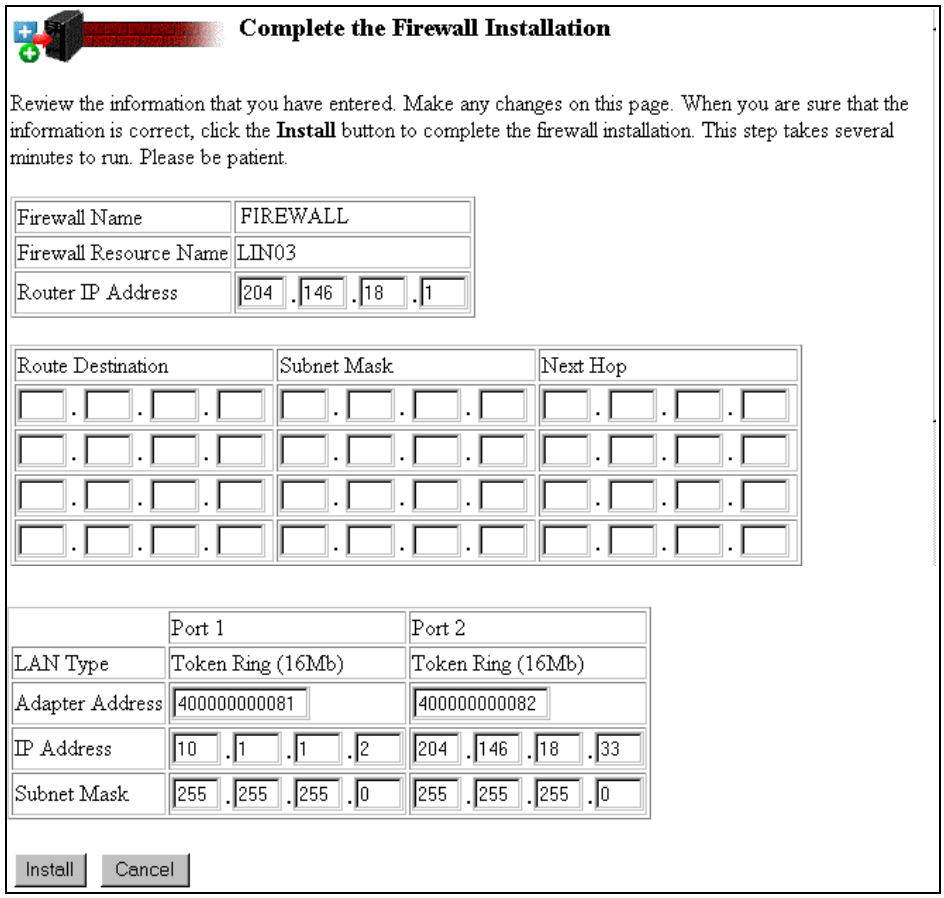


Figure 52. Firewall installation summary page

Start the firewall (Figure 53) by clicking **Start**.

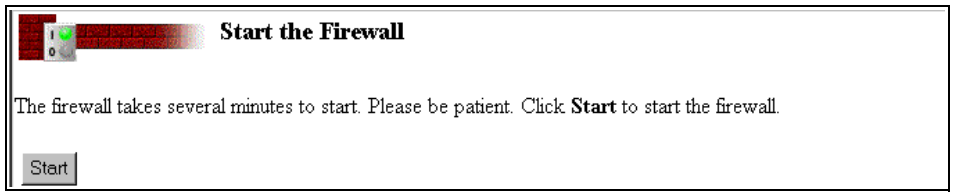


Figure 53. Starting the firewall

4.3.3 Performing basic configuration

Perform the basic configuration of the local firewall (FIREWALL). For further information, refer to *Getting Started with IBM Firewall for AS/400*, SC41-5424, and the redbooks *AS/400 Internet Security: IBM Firewall for AS/400,* SG24-2162, and *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376.

The Review Configuration page, shown in Figure 54 and Figure 55 on page 108, show our configuration on the local system, AS01. Refer to Figure 51 on page 103 for the scenario network configuration.

Notice that we entered the name of the merchant server and its *public* IP address. In this example, the public IP address of the merchant server is the same as the non-secure port of the firewall. The next section of the page asks if the public server is behind the firewall. If so, enter the public ports that will be used for HTTP and HTTPS. We selected the well-known ports of 80 and 443 for HTTP and HTTPS, respectively. We also entered the private IP address of the merchant server, which is the home AS/400 *INTERNAL port (F in Figure 51 on page 103) of the firewall. Remember that the merchant server is on the same AS/400 system that houses the firewall. Information about the merchant server is used to automatically generate the appropriate NAT settings and filter rules for accessing the merchant server behind the firewall.

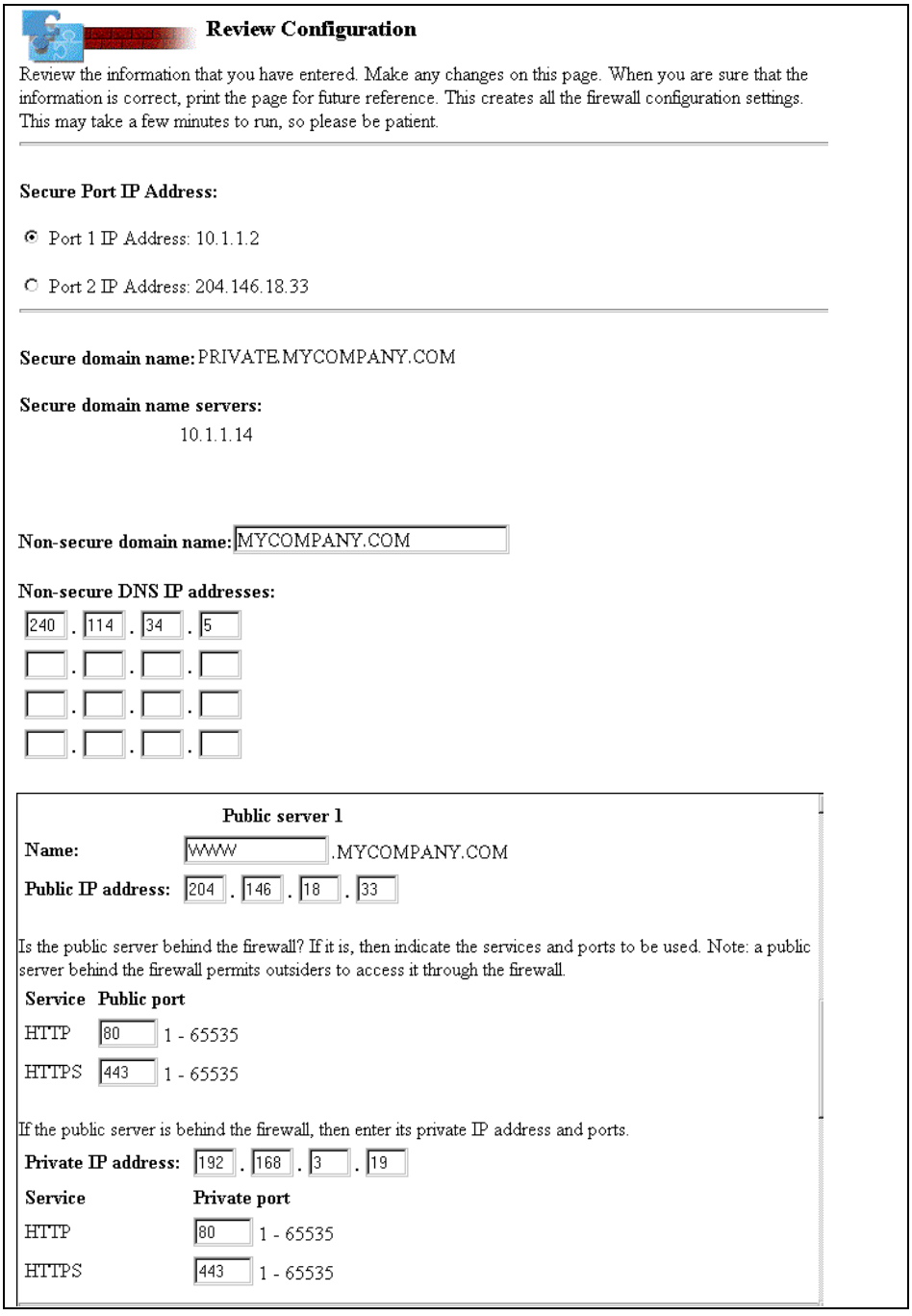
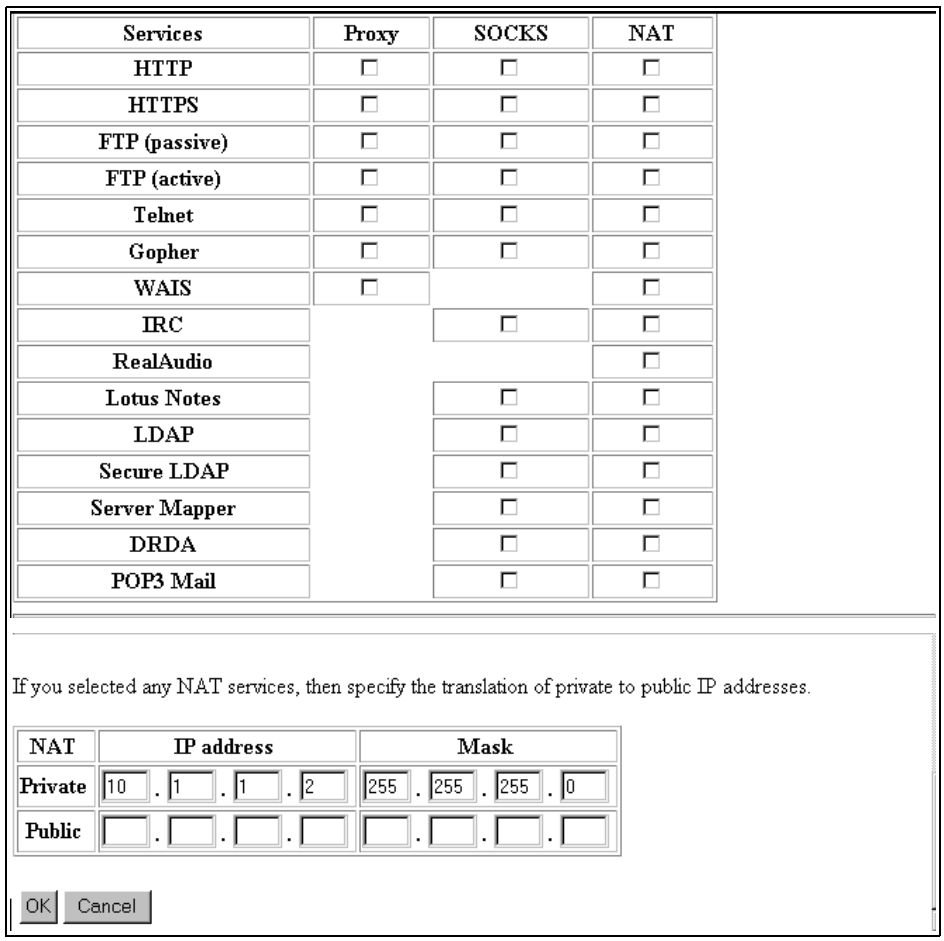


Figure 54. Firewall basic configuration summary page (Part 1 of 2)



Services	Proxy	SOCKS	NAT
HTTP	☐	☐	☐
HTTPS	☐	☐	☐
FTP (passive)	☐	☐	☐
FTP (active)	☐	☐	☐
Telnet	☐	☐	☐
Gopher	☐	☐	☐
WAIS	☐		☐
IRC		☐	☐
RealAudio			☐
Lotus Notes		☐	☐
LDAP		☐	☐
Secure LDAP		☐	☐
Server Mapper		☐	☐
DRDA		☐	☐
POP3 Mail		☐	☐

Figure 55. Firewall basic configuration summary page (Part 2 of 2)

Complete the following steps:

1. Click **OK**. A confirmation page (Figure 56) is shown. It indicates that the firewall is configured. It is not necessary to restart the firewall at this time because we have more configuration work to do.
2. Click **No** on the Firewall is Configured page.

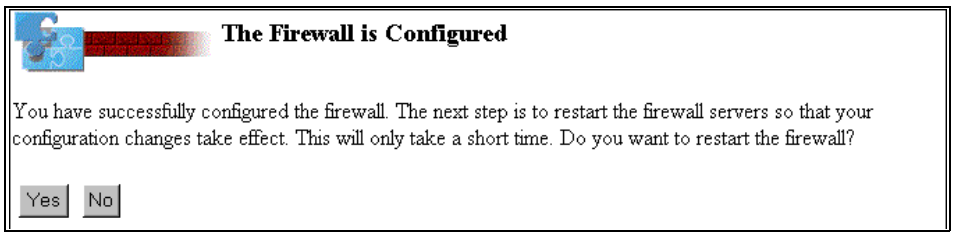


Figure 56. Confirmation that the firewall is configured

4.3.4 Changing NAT rules

Basic configuration automatically creates the NAT filter rules to allow HTTP and HTTPS traffic. However, if you want to use SET Secure Electronic Transaction, you must create the additional rules that we have to communicate with eWallet on the shopper's PC and with the Payment Gateway. The default ports for eWallet communication are 8620 and 8621 and, for the Gateway Server, it is 8888.

Note

Ports 8620 and 8621 are used to communicate with eWallet on a PC. If your Payment Server supports only MOP or MIA, you do not have to configure the NAT and filter rules for the two ports. The only port that needs to be configured in the firewall is the port to the payment gateway in MOP and MIA transactions.

If your SET certifying authority is using some special port for the SET certificate, you have to consider that too.

For further information about NAT, refer to *Getting Started with IBM Firewall for AS/400*, SC41-5424, and the redbook *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376.

For information regarding SET, refer to the Web site at: `http://www.setco.org`

In this example, we add the eWallet rule (port 8620) to NAT. You have to add a rule for each port that you want to use on your server. We also change the NAT filter rules to translate all ports.

To begin, perform the following steps:

1. Click **NAT** on the Configuration Menu page (Figure 57 on page 110).

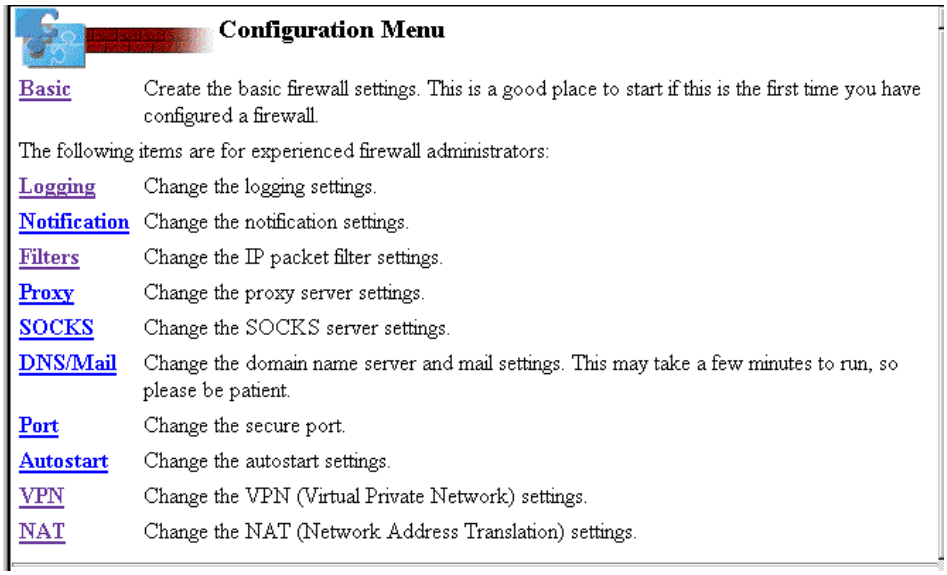


Figure 57. Selecting NAT from the Configuration menu

The Network Address Translation Settings page appears as shown in Figure 58. Notice that IBM Firewall for AS/400 already generated two MAP settings for us. They are based on the information we provided in the Public server 1 section of Basic configuration (Figure 54 on page 107).

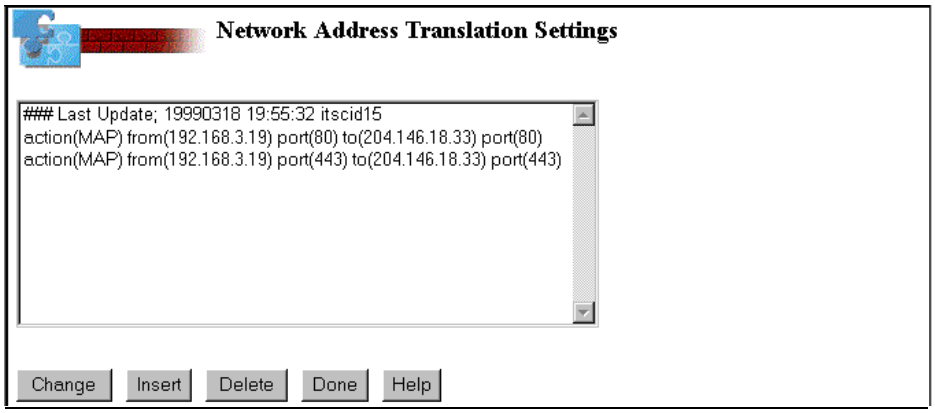


Figure 58. Network Address Translation Settings page

2. Select the last MAP setting in the list (Figure 58). Click **Delete**.

3. Select the remaining MAP setting in the list (Figure 58). Click **Change**. The Change Network Address Translation page is shown.

   Figure 59 shows the Change Network Address Translation page. Change the *From port* to “0” and the *To port* to “0”. Port 0 tells NAT to pass all communication on all ports. Changing the NAT port to 0 makes us depend on the filter rules for protection of the merchant server. Therefore, it is important that you only allow the IP packets that you want to get through the firewall filters. During the SET payment, the Payment Server uses randomly selected ports above 1023 to communicate to the Payment Gateway.

> **Tip**
>
> Remember that the *From* port is always the secure (hidden) address and the *To* address is the registered address that you want to publish.

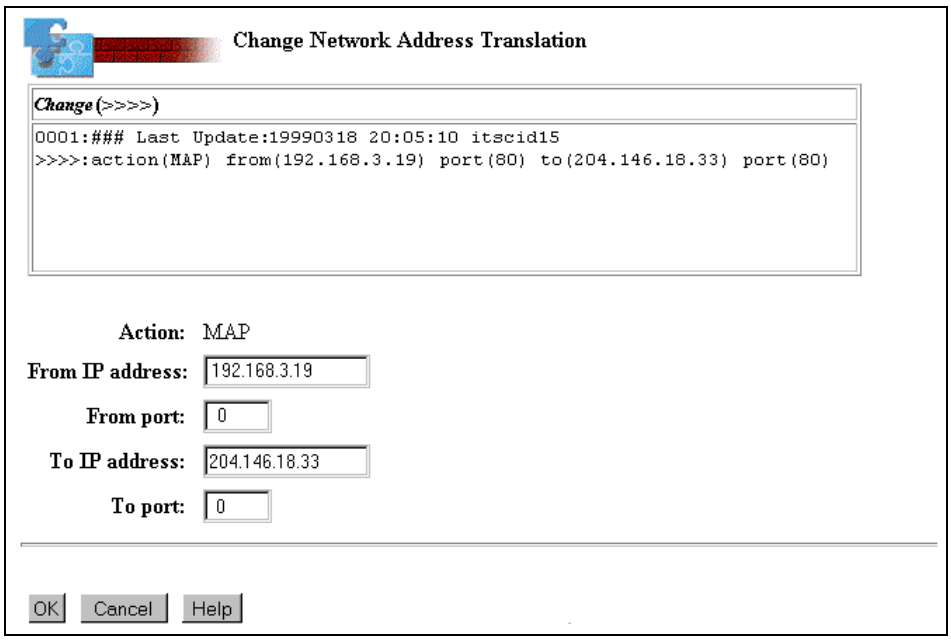


Figure 59. Changing the NAT MAP setting

4. The *From IP address* is always the secure (hidden) address (in our environment, this is 192.168.3.19) and the port to map is 0 (all port). The *To* IP address is the address that we want to publish, which is

204.146.18.33 (the non-secure port of the firewall), also using port 0 (all port). After entering the required information, click **OK** to continue.

5. The resulting NAT setting is shown for confirmation (Figure 60). If you have more settings to add, you can do so now. In this scenario, this is the only NAT setting we need to add. Click **Done**.

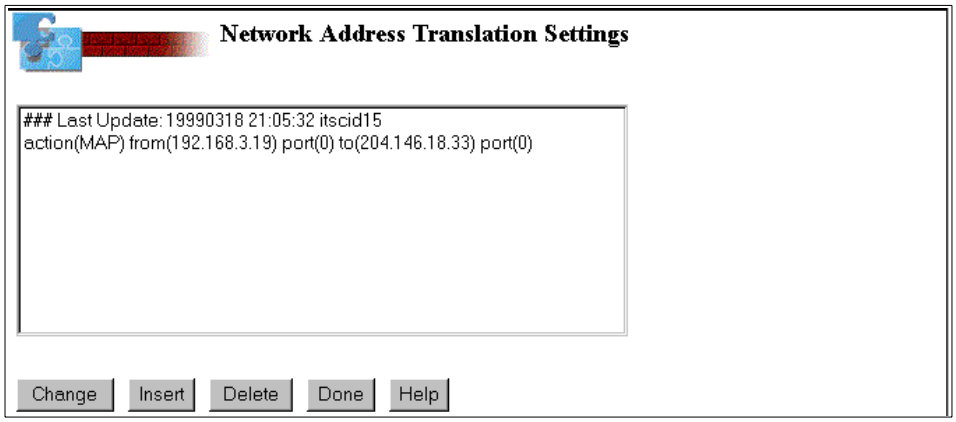


Figure 60. Displaying NAT settings

You are returned to the Firewall Installation Tasks page (Figure 52 on page 105).

4.3.5 Starting NAT

Click the **Administration** icon. Then, click **Status** from the Administration Menu page. Start NAT as shown in Figure 61.

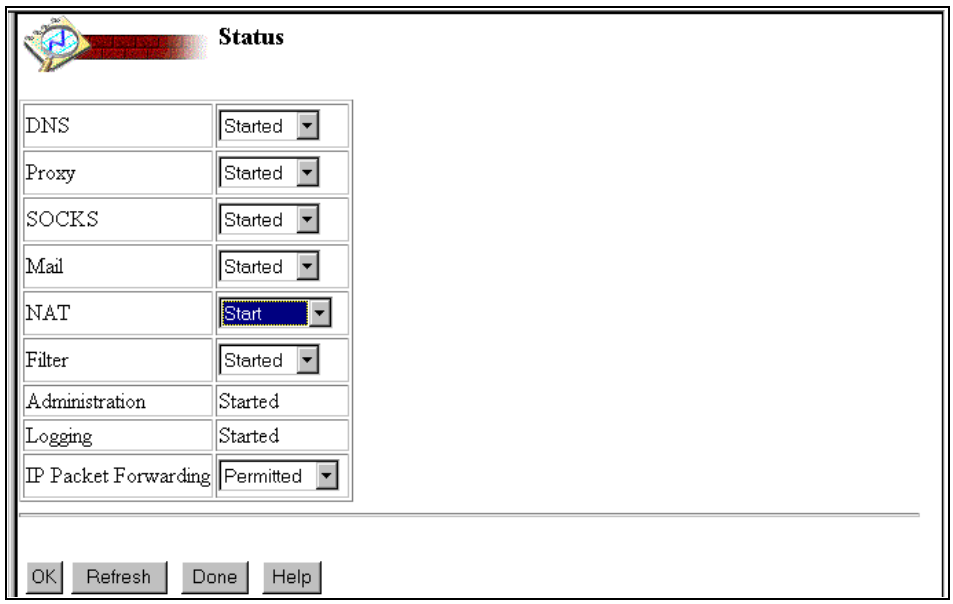


Figure 61. Starting NAT from the Status page

4.3.6 Adding filter rules for SET

Basic configuration automatically creates the filter rules to allow HTTP and HTTPS traffic. However, you must create the additional rules for SET. We have to communicate with the eWallet and with the Payment Gateway. The default ports for eWallet are 8620 and 8621, and 8888 for the payment gateway.

Note

Ports 8620 and 8621 are used to communicate with eWallet on a PC. If your Payment Server supports only MOP or MIA, you do not have to configure the NAT and filter rules for the two ports. The only port that needs to be configured in the firewall is the port to the payment gateway in MOP and MIA transactions.

You must be sure that you do not override the rules that Basic configuration created. This will make it easier to recognize rules that you manually add after the initial configuration of the firewall. We recommend that you create a section at the bottom of the filter rules just before the *Ending defense* section. Enter a title, such as `Custom Rules`.

Important

Always test your filter rules before you use them. One simple misspelling can open your filter.

4.3.7 Filter rules for requesting a certificate

You need to add the rules listed in this section to allow the Payment Server to request a digital SET certificate from a certificate authority that uses port 5065 for a certificate request. If your certificate authority is using port 80 or 443 for certificate requests, you do not need to add any filter rules for the certificate request. Those filters were created when you selected to have the public server behind the firewall during the basic configuration of the firewall. See Figure 54 on page 107.

Add the following four filter rules to your firewall:

```
permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp ge 1024 eq 5065
secure   route inbound f=y l=y t=0 # Permit SET Cert. request 1/2

permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp ge 1024 eq 5065
non-secure route outbound f=y l=n t=0 # Permit SET Cert. request 2/2

permit 0.0.0.0 0.0.0.0 204.146.18.33 255.255.255.255 tcp eq 5065 ge 1024
non-secure both inbound f=y l=n t=0 # Permit SET Cert. request response 1/2

permit 0.0.0.0 0.0.0.0 192.168.3.19 255.255.255.255 tcp eq 5065 ge 1024
secure   both outbound f=y l=n t=0 # Permit SET Cert. request response 2/2
```

You should consider removing or changing these filter rules to deny when you have received your digital SET certificate.

For more information about adding and implementing filter rules, refer to *IBM Internet Security: IBM Firewall for AS/400*, SG24-2162.

4.3.8 Setting up SOCKS for a certificate request

Before configuring the SOCKS support for Payment Server for the AS/400 system, you must consider the following network environments:

- The network that is directly connected to the AS/400 system. A SOCKS server is not needed to reach the network.
- The network that requires the use of a SOCKS server for access and the SOCKS server to use to access the network.

If you are going to request a digital SET certificate from VeriSign, you may have to configure SOCKS client support on your AS/400 system. You have to use Operations Navigator to configure the SOCKS on your AS/400 system.

Select **TCP/IP Properties**, and click the **SOCKS** tab. The SOCKS information window appears (Figure 62).

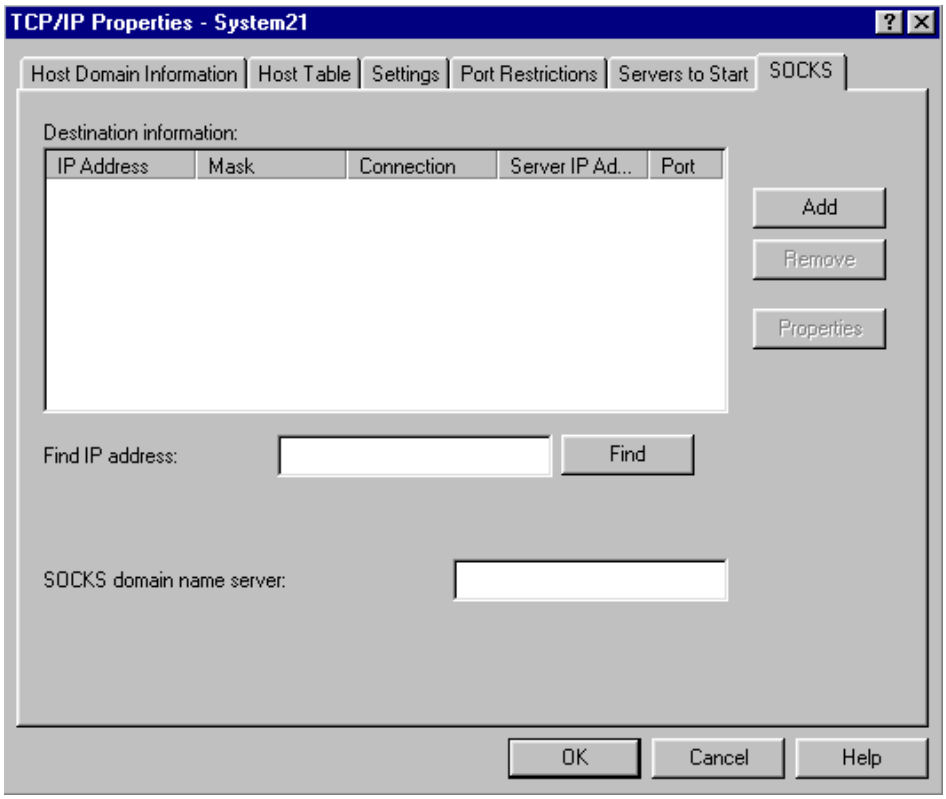


Figure 62. Operations Navigator: TCP/IP properties SOCKS before configuration

You are now ready to configure SOCKS information for your AS/400 system.

4.3.8.1 Defining the direct network

Do not use the SOCKS server to connect to any network that is directly connected to the system. To prevent the AS/400 system from connecting through the SOCKS server, the directly connected network should be defined.

To define the directly connected network, complete the following steps:

1. In the SOCKS information window, click **Add**. The Add SOCKS Destination window appears (Figure 63).

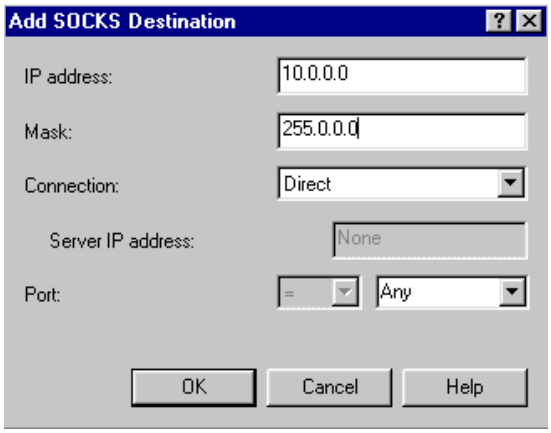


Figure 63. Add SOCKS Destination with direct connection information

2. Type the network address of the secure network in the IP address field. In our sample network, we use 10.0.0.0.
3. Type the subnet mask that describes your secure network in the Mask field. In our sample network, we use a subnet mask of 255.0.0.0.
4. Click the down arrow in the Connection field, and select **Direct** from the list of options.
5. Click **OK** to add the destination information.

You have now defined the “10.” network as a direct network. SOCKS does not access any host with an address that starts with “10.”

4.3.8.2 Defining the network connection using SOCKS

Now, you must define the network for use with the SOCKS server. In this example, we use the SOCKS server to access all networks except the direct connection.

To define the network for use with SOCKS, follow these steps:

1. In the SOCKS information window, click the **Add** button. The Add SOCKS Destination window appears (Figure 64).

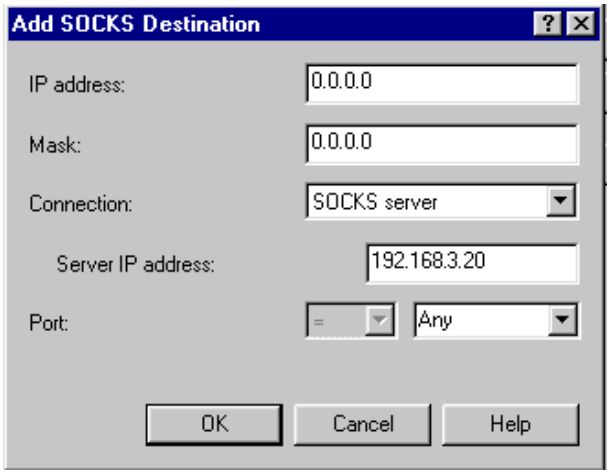


Figure 64. Add SOCKS Destination with SOCKS server connection

2. Type the address `0.0.0.0` in the IP address field.
3. Type the subnet mask `0.0.0.0` in the Mask field.

   When a destination address is “ANDed” with a mask of 0.0.0.0, the result is 0.0.0.0. By specifying a mask and address of all zeros, all IP addresses match this destination description.
4. Click the down arrow in the Connection field, and select **SOCKS Server** from the list of options.
5. Type the IP address of the SOCKS server in the Server IP Address field. On the AS/400 system with the firewall installed, this is the IP address of the *INTERNAL port of the firewall. On other AS/400 systems in the secure network, this is the IP address of the secure port of the firewall.
6. Verify that the Port field is set to **Any**. This specifies the remote ports for which this connection can be used.
7. Click **OK** to add the destination information.

You have now defined the destination information for SOCKS. You may also need to configure the SOCKS domain name server.

4.3.8.3 Defining the SOCKS domain name server

The SOCKS domain name server field specifies the IP address of a DNS server that can resolve names or IP addresses that reside on a non-secure network. Leave this field blank if the domain name servers configured with TCP/IP resolve the addresses.

For name or IP address resolution, the system queries the DNS servers configured with TCP/IP first. If they cannot resolve the name or address, the system queries the DNS server that you specify.

Note

At least one DNS server must be configured by using CFGTCP option 12 before SOCKS checks the domain name server configured for SOCKS.

If you do not have an internal DNS server, point the AS/400 system to the firewall for DNS services. If the internal DNS server cannot resolve external information, type the IP address of the firewall in the SOCKS domain name server field. On the AS/400 system with the firewall installed, this is the IP address of the *INTERNAL port of the firewall. On the other AS/400 systems in the secure network, this is the IP address of the secure port of the firewall.

After you enter all of your SOCKS information, your SOCKS information window should appear similar to the one shown in Figure 65. Click **OK** to save the configuration. The Operations Navigator window appears.

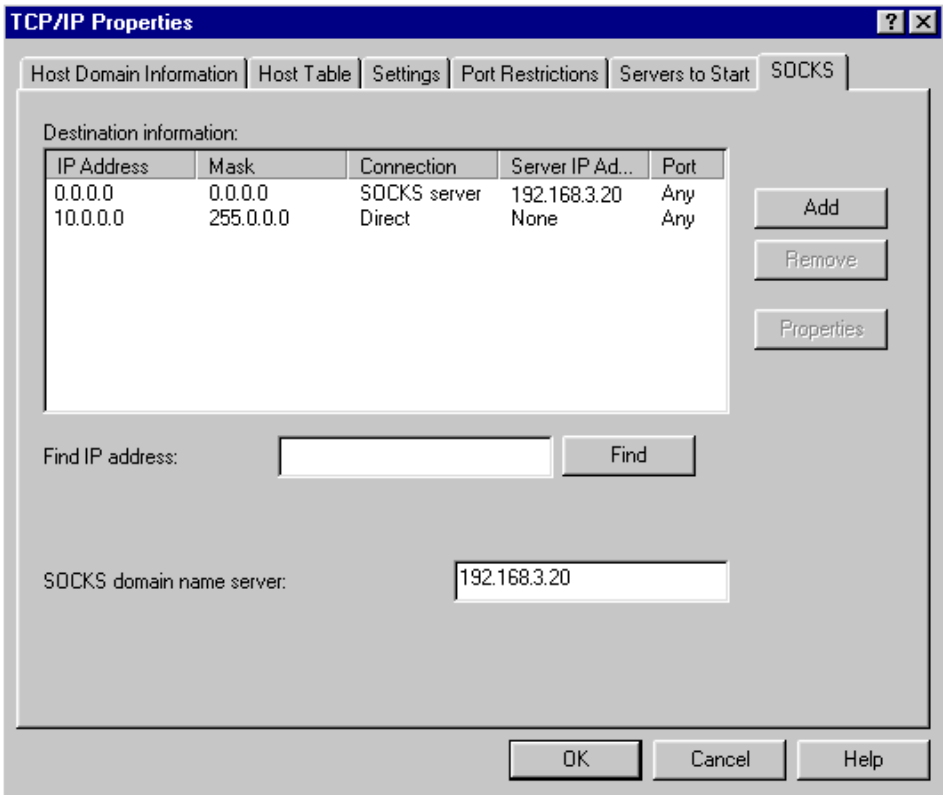


Figure 65. Pointing to the SOCKS domain name server

4.3.9 Filter rules for SET communication

You need to add the following rules to allow the Card Holder Requests to and from the eWallet, and Pay Authorizations and Payment Capture to the Payment Gateway. Here are the filter rules to allow communication between the Payment Server and the eWallet. In our example, they are ports 8620 and 8621:

```
permit 0.0.0.0 0.0.0.0 204.146.18.33 255.255.255.255 tcp ge 1024 eq 8620
non-secure both inbound f=y l=y t=0

permit 0.0.0.0 0.0.0.0 192.168.3.19 255.255.255.255 tcp ge 1024 eq 8620
secure route outbound f=y l=y t=0

permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp eq 8620 ge 1024
secure route inbound f=y l=y t=0
```

```
permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp eq 8620 ge 1024
non-secure route outbound f=y l=y t=0

permit 0.0.0.0 0.0.0.0 204.146.18.33 255.255.255.255 tcp ge 1024 eq 8621
non-secure both inbound f=y l=y t=0

permit 0.0.0.0 0.0.0.0 192.168.3.19 255.255.255.255 tcp ge 1024 eq 8621
secure route outbound f=y l=y t=0

permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp eq 8621 ge 1024
secure route inbound f=y l=y t=0

permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp eq 8621 ge 1024
non-secure route outbound f=y l=y t=0
```

Here are the filter rules to allow communication between the Payment Server and the Payment Gateway. In our example, it is port 10010. The default port for communication with an acquirer (Payment Gateway) is 8888.

```
permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp ge 1024 eq 10010
secure route inbound f=y l=y t=0

permit 192.168.3.19 255.255.255.255 0.0.0.0 0.0.0.0 tcp ge 1024 eq 10010
non-secure route outbound f=y l=y t=0

permit 0.0.0.0 0.0.0.0 204.146.18.33 255.255.255.255 tcp eq 10010 ge 1024
non-secure both inbound f=y l=y t=0

permit 0.0.0.0 0.0.0.0 192.168.3.19 255.255.255.255 tcp eq 10010 ge 1024
secure route outbound f=y l=y t=0
```

Important

These filter rules may not apply to your system. The ports used by your acquirer may differ from the ports used in our example.

For more information about how to add and implement filter rules, please refer to *IBM Internet Security: IBM Firewall for AS/400*, SG24-2162.

4.3.10 Configuring a default route to route Web server responses

Because you have the merchant server on the same AS/400 system that houses the firewall (AS01 in our scenario), you must add a default route that specifies the *INTERNAL IP address of the firewall (interface E, Figure 51 on page 103) as the next hop. This allows the Internet clients to receive responses from the server (which must be routed through the firewall). Refer

to Figure 67 on page 122 for an example of the default route configuration on AS01 entry.

4.3.11 Restarting the filters

To restart the filters, click the firewall **Administration** icon, and click **Status** from the Administration Menu page. Select **Restart** for the filters, and click **OK**. Refer to Figure 61 on page 113 for an example of the Status page.

4.3.12 Verifying access to the Web server and Internet

After completing the steps in our scenario, we perform the following verification testing. We successfully open a Web page:

- On the merchant server behind the firewall (AS01) from the Internet
- On the Internet from an internal client in the secure network using SOCKS as well as Proxy (this is optional)

For more information about filter rules and NAT, refer to *IBM Internet Security: IBM Firewall for AS/400*, SG24-2162, and *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376.

4.3.13 Additional configuration information

This section shows the TCP/IP configuration and network server descriptions for the firewall configuration FIREWALL on system AS01 (Figure 66 through Figure 74 on page 124).

```
                      Work with TCP/IP Interfaces
                                                            System:   AS01
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display   9=Start   10=End

     Internet         Subnet             Line      Line
Opt  Address          Mask            Description  Type

     10.1.1.3         255.255.255.0    TRLAN2       *TRLAN
     127.0.0.1        255.0.0.0        *LOOPBACK    *NONE
     192.168.3.19     255.255.255.0    FIREWAL00    *TRLAN
```

Figure 66. AS/400 system TCP/IP interfaces

```
                          Work with TCP/IP Routes
                                                              System:  AS01
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display

     Route            Subnet           Next             Preferred
Opt  Destination      Mask             Hop              Interface

     *DFTROUTE        *NONE            192.168.3.20     *NONE
```

Figure 67. AS/400 system routing configuration

```
                        Display Network Server Desc                    AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *BASIC



Resource name  . . . . . . . . . . :   LIN03
Network server type  . . . . . . . :   *BASE
Online at IPL  . . . . . . . . . . :   *YES
Vary on wait . . . . . . . . . . . :   *NOWAIT
Language version . . . . . . . . . :   2924
Country code . . . . . . . . . . . :   1
Code page  . . . . . . . . . . . . :   850
NetBIOS description  . . . . . . . :   QNTBIBM
Start NetBIOS  . . . . . . . . . . :   *NO
Start TCP/IP . . . . . . . . . . . :   *YES
```

Figure 68. Network Server Description (Part 1 of 7)

```
                        Display Network Server Desc                    AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *BASIC



Configuration file . . . . . . . . :   *NONE
  Library  . . . . . . . . . . . . :
Synchronize date and time  . . . . :   *YES
Text . . . . . . . . . . . . . . . :   *FIREWALL
```

Figure 69. Network Server Description (Part 2 of 7)

```
                      Display Network Server Desc                   AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *PORTS
Ports  . . . . . . . . . . . . . . :



-----Attached lines------
Port           Attached
number         line
1              FIRNAT101
2              FIRNAT102
*INTERNAL      FIRNAT100
```

Figure 70. Network Server Description (Part 3 of 7)

```
                      Display Network Server Desc                   AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *STGLNK
Storage space links  . . . . . . . :



---------------------------Storage space links---------------------------
Network
server
storage        Drive     Text
FIRNAT100       K
```

Figure 71. Network Server Description (Part 4 of 7)

```
                      Display Network Server Desc                   AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *TCPIP
TCP/IP port configuration  . . . . :



--------------------TCP/IP port configuration--------------------
                                                          Maximum
               Internet           Subnet              transmission
Port           address            mask                    unit
1              10.1.1.2           255.255.255.0             1500
2              204.146.18.33      255.255.255.0             1500
*INTERNAL      192.168.3.20       255.255.255.0            15400
```

Figure 72. Network Server Description (Part 5 of 7)

```
                      Display Network Server Desc                  AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *TCPIP
TCP/IP route configuration . . . . :



--------------TCP/IP route configuration---------------
Route              Subnet              Next
destination        mask                hop
*DFTROUTE          *NONE               204.146.18.1
```

Figure 73. Network Server Description (Part 6 of 7)

```
                      Display Network Server Desc                  AS01
                                                          04/05/99  11:37:45
Network server description . . . . :   FIREWALL
Option . . . . . . . . . . . . . . :   *TCPIP



TCP/IP local host name . . . . . . :   *NWSD

TCP/IP local domain name . . . . . :   *SYS



TCP/IP name server system  . . . . :   *SYS
```

Figure 74. Network Server Description (Part 7 of 7)

For more detailed information about firewall implementation, read the redbook *AS/400 Internet Security: IBM Firewall for AS/400*, SG24-2162. For detailed information about Network Address Translation (NAT), read the redbook *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376.

4.3.14 OS/400 TCP/IP configuration

Because you have a merchant server on the same AS/400 system that houses the firewall (AS01), you must add a default route specifying the *INTERNAL IP address of the firewall (interface E in Figure 51 on page 103) as the next hop. This allows Internet clients to receive responses (which must be routed through the firewall) from the server. Refer to Figure 67 on page 122 for an example of the default route configuration on AS01.

It is not necessary to make manual changes to the firewall network server description for this scenario. Figure 75 and Figure 76 show the TCP/IP interface and route configuration on system AS01.

```
                        Work with TCP/IP Interfaces
                                                                 System:   AS01
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display   9=Start   10=End

     Internet         Subnet             Line      Line
Opt  Address          Mask            Description  Type

     10.196.5.3       255.255.255.0    TRLAN2       *TRLAN
     127.0.0.1.15     255.0.0.0        *LOOPBAK     *NONE
     192.168.3.2      255.255.255.0    FIREWALL00   *TRLAN
```

Figure 75. AS/400 system TCP/IP interfaces: AS01

```
                          Work with TCP/IP Routes
                                                                 System:   AS01
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display

     Route            Subnet           Next             Preferred
Opt  Destination      Mask             Hop              Interface

     *DFTROUTE        *NONE            192.168.3.2      *NONE
```

Figure 76. AS/400 system routing configuration: AS01

4.3.15 eTill.Hostname

When the SET Payment Initiation (Wakeup) message is sent to eWallet, Payment Server retrieves the TCP/IP hostname in the environment variable called eTill.Hostname and embeds it in the wakeup message. When Payment Server starts, it uses a Java function to retrieve the fully qualified TCP/IP host name of the machine on which Payment Server is running and sets it in the environment variable. If your Payment Server is behind a firewall and you want to hide the host name from Internet users, you can use the hostname column in the ETJVMPARMS table to override the real host name. An example use is:

```
INSERT INTO QUSRPYMSVR.ETJVMPARMS (PARMGROUPNAME, TYPE, VALUE)
VALUES('Hostname', 1, 'eTill.Hostname=host.your.domain.com')
```

4.4 Backend system connection

As a part of your merchant server implementation, you must have a connection between your merchant system and your backend system. We do not discuss security on the backend system or any other security on the local network.

If you are interested in IP filtering on the AS/400 system, read the redbook *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376.

To communicate between the merchant server and the back-end system, you must have some sort of communication. We assume that you have a local area network (LAN) that connects your two systems.

Chapter 5. Sample Payment Server configuration

The information presented in this chapter is a Payment Server configuration performed at a real customer location. It is provided here to give you a solid idea of what is actually involved in configuring Payment Server in a real life scenario.

5.1 Overview

The configuration presented here uses the Merchant Originated Payment (MOP) implementation discussed earlier. Web users (cardholders) are not required to have signed certificates from a cardholder certificate authority to make a purchase from the merchant. Therefore, this implementation uses the SET protocol between the merchant and the acquirer, not SET itself because the root CA is not SETCo endorsed or SETCo themselves. The merchant shown in this chapter is called Worldshare.

The acquirer, NOVA Information Systems, also known as a *processor*, is providing the payment gateway services for Worldshare. NOVA is called a processor because it is not a financial institution, but simply provides acquirer services and works with all government recognized financial institutions. When most financial institutions provide acquirer services, they usually require the merchant to setup a bank account with them, where a processor will work with the merchant's current bank.

Worldshare (merchant) accepts Visa, MasterCard, and American Express card brands from cardholders. NOVA (processor) accepts and will process all three brands that Worldshare needs for authorizations. This agreement was set up when Worldshare established an account for acquirer services from NOVA. If you are a U.S. merchant, it is likely you will follow this configuration model. Many merchants think that MOP or MIA is secure enough to do their business even though they recognize that pure SET environments are more secure, They also think that getting certificates by the cardholders for their eWallet is a cumbersome process to keep away from their business.

Note

This scenario demonstrates getting Payment Server up and running. You still need to interface with a merchant server, sockets interface, or C API to test and perform transactions from your Payment Server to the acquirer.

In NOVA's MOP implementation, they currently use CyberTrust as their root CA. The root CA within CyberTrust is called SPS. SPS is equivalent to the SETCo Root CA in a pure SET and MIA environment. Figure 77 illustrates the MOP hierarchical trust relationships.

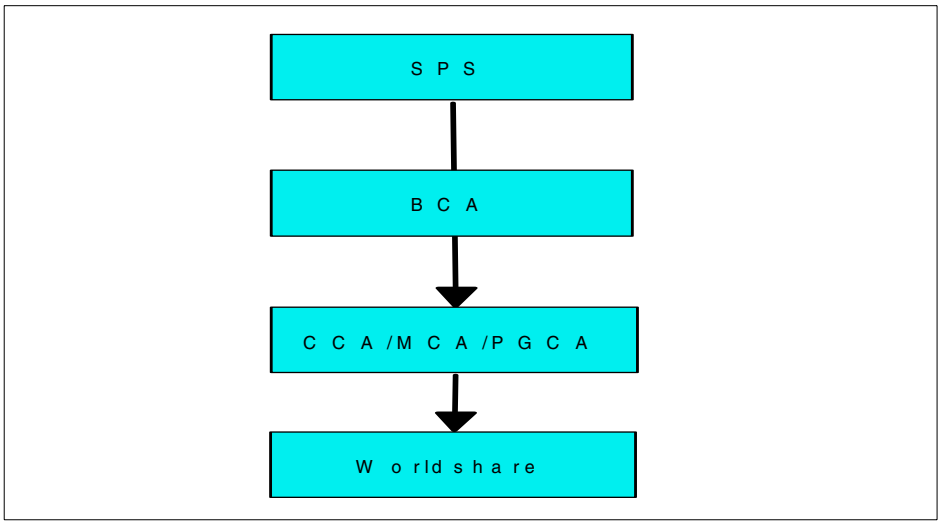


Figure 77. MOP implementation trust hierarchy with CyberTrust SPS

Notice in Figure 77 that CyberTrust is using a combined BCA/MCA/PGCA. SPS, in this case, is acting as a root CA and a card brand CA for each card brand, and a merchant CA for each merchant that sets up a services agreement with NOVA for acquirer services. SPS is also taking on the role of the Payment Gateway CA, which establishes (certifies) the gateway that NOVA provides for merchants that use its services. The environment is set up this way because of the implementation requirements of the SET protocol.

5.2 Transactions flow

Figure 78 shows a conceptual flow of how Worldshare, NOVA, and cardholders interface with one another.

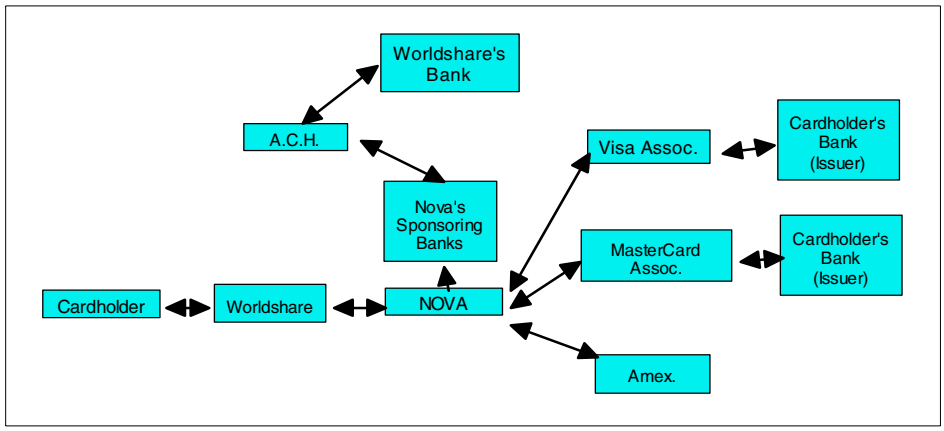


Figure 78. Transaction flow between participating parties

In this scenario, a cardholder using Visa, MasterCard, or American Express makes a purchase request on Worldshare's site. Worldshare then generates a authorization request from NOVA over the payment gateway. NOVA takes the request and passes this along to the Visa or MasterCard association who then communicates with the cardholder's bank who issued the Visa and MasterCard. The corresponding bank (issuer's bank) then generates the authorization code and passes this back through the network to the association and back to NOVA, who then passes this on to Worldshare, and thus the order is placed. Once this happens, NOVA initiates a capture request for the deposits into Worldshare's bank account through their sponsoring banks. The sponsoring banks go through the Automated Clearing House, and the funds are deposited into Worldshare's account. On Worldshare's monthly statement from their bank, they will see deposits from NOVA when using Visa or MasterCard. For American Express transactions, NOVA has a direct connection to this entity because American Express issues cards directly to cardholders and do not go through associations. American Express issues the authorization code and Worldshare will see deposits from American Express on their monthly bank statements.

5.3 Tasks performed before Payment Server configuration

The following tasks were performed before configuring Payment Server:

- Worldshare contacted NOVA and established an acquirer account with NOVA.
- NOVA sent an e-mail regarding its acquirer information, firewall configuration, and information for requesting certificates to Worldshare all at the same time.
- Worldshare installed the Payment Server software and verified the latest PTFs at: `http://as400service.rochester.ibm.com`

Note

You must have this information available before you configure the Payment Server. *Do not* arrange to configure the server until you have received this information from your acquirer.

5.4 Configuration steps

The following configuration steps are used to configure the Payment Server with NOVA's information. These are the same steps that are illustrated in Chapter 2, "Planning for SET Secure Electronic Transaction" on page 15. Here, we use NOVA and Worldshare's information.

1. Access the AS/400 Task page at:

   `http://hostname:2001`

   The window shown in Figure 79 appears.

2. Select **Payment Server for AS/400**.

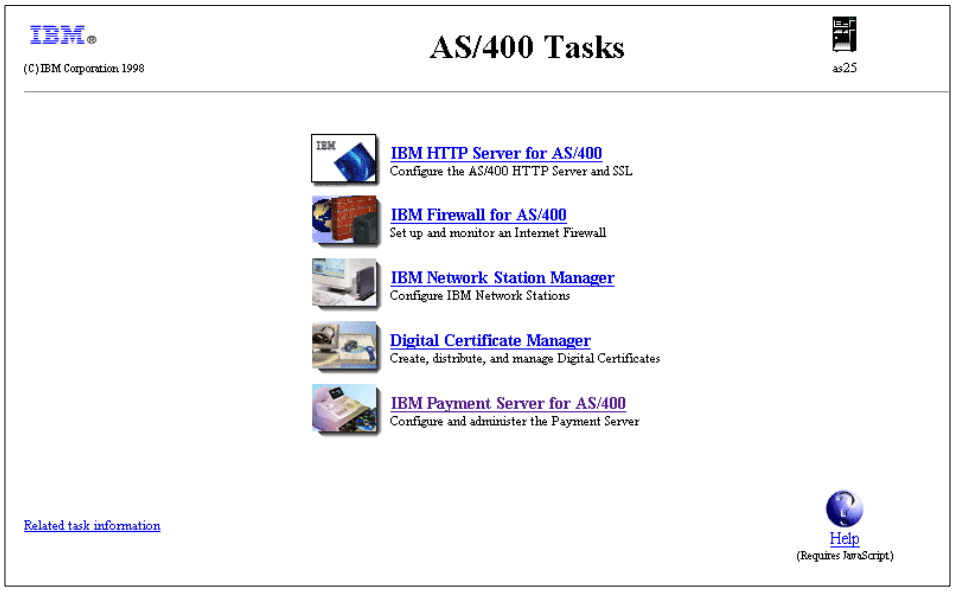


Figure 79. AS/400 Task page

3. Create a Payment Server instance. This is done by clicking **Create** under the **Administration** link (Figure 80).

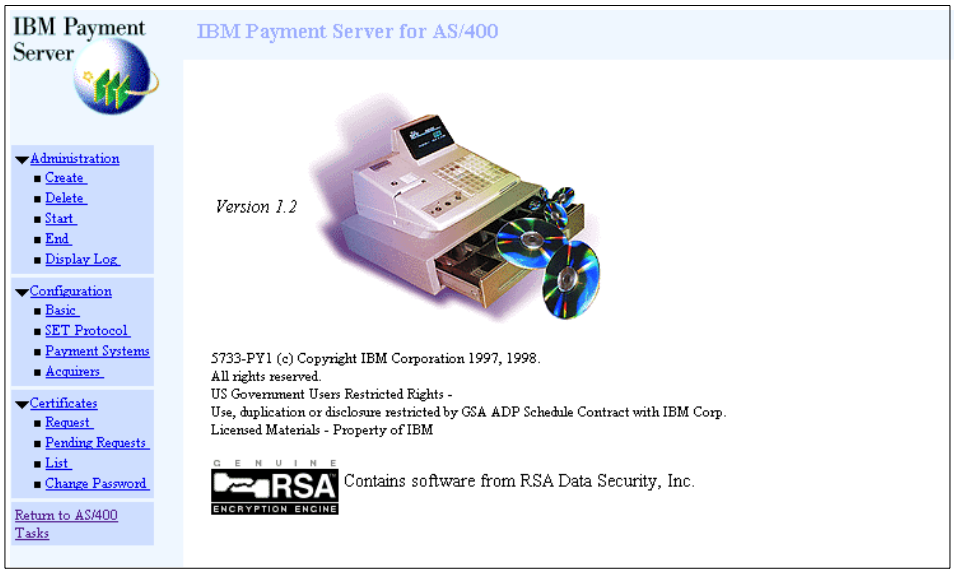


Figure 80. Payment Server for AS/400

4. The window shown in Figure 81 appears. Enter a key database password. This can be any password you choose. This protects access to the database with your certificates.

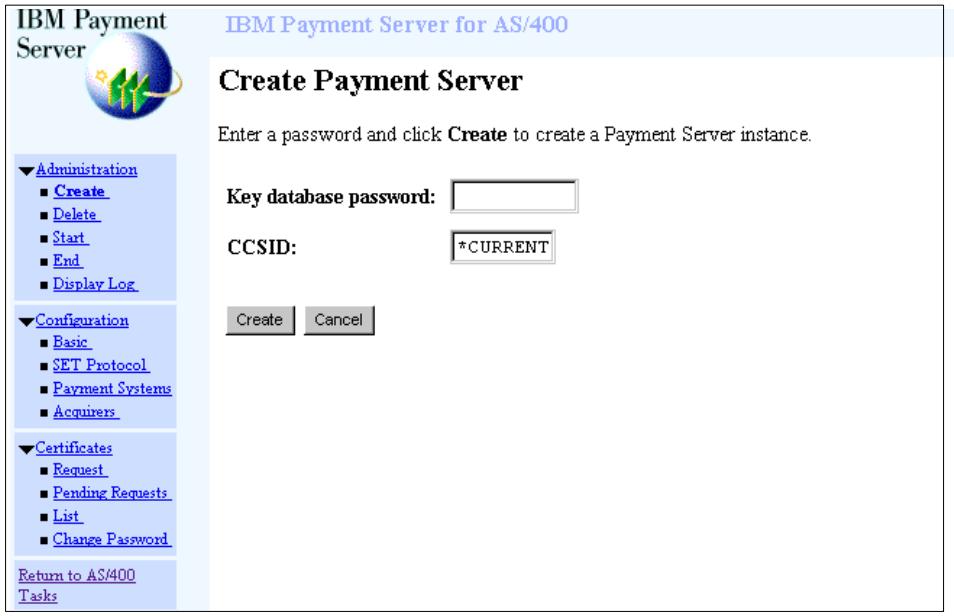


Figure 81. Create Payment Server screen

5. After you enter a password, click **Create**. The window shown in Figure 82 appears.

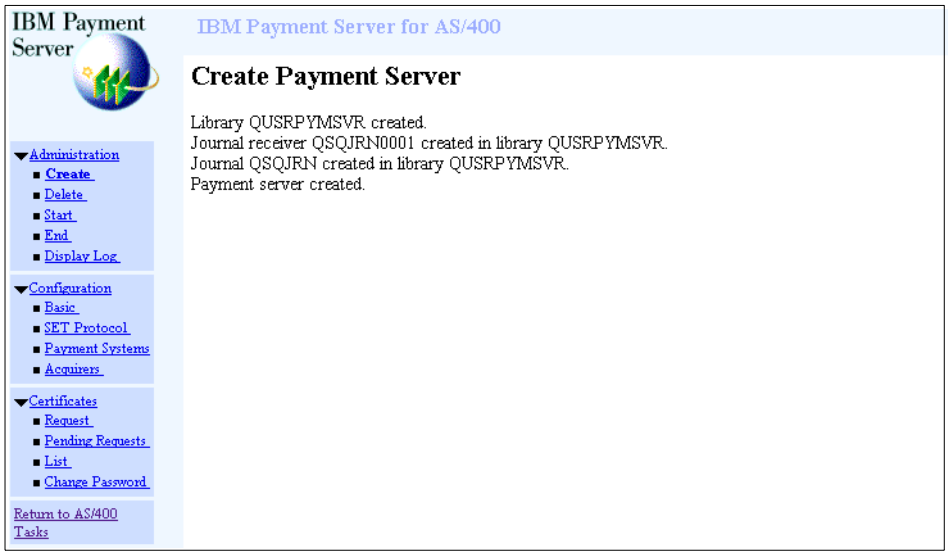


Figure 82. Confirmation screen for successful creation of Payment Server

6. You are ready to begin the configuration. From the option on the left, click **Basic**. The window shown in Figure 83 appears.

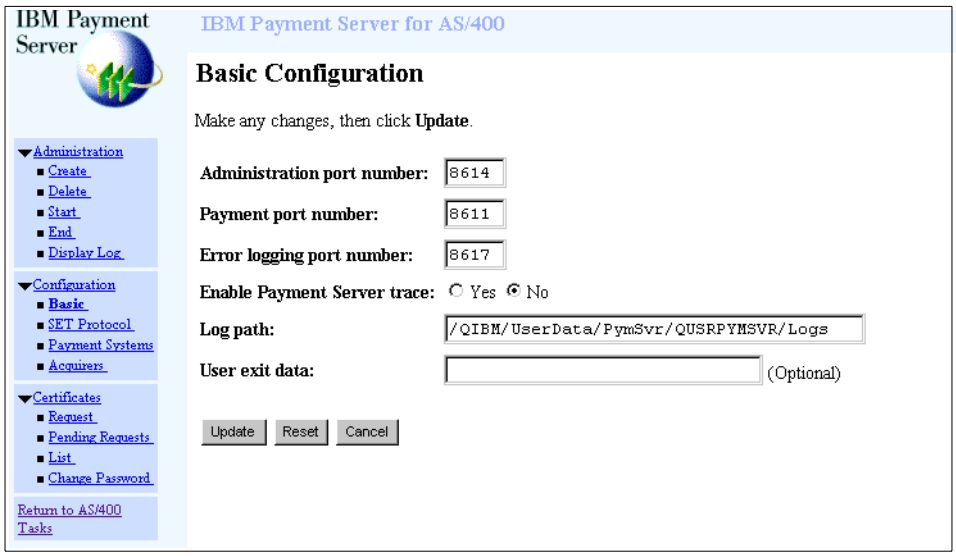


Figure 83. Basic Configuration screen

7. On the Basic Configuration page, check the **Yes** radio button for Enable Payment Server trace. If you experience problems, this allows you to go and look at the specified log directory for trace files. Click **Update** to accept the rest of the defaults. The window shown in Figure 84 appears.

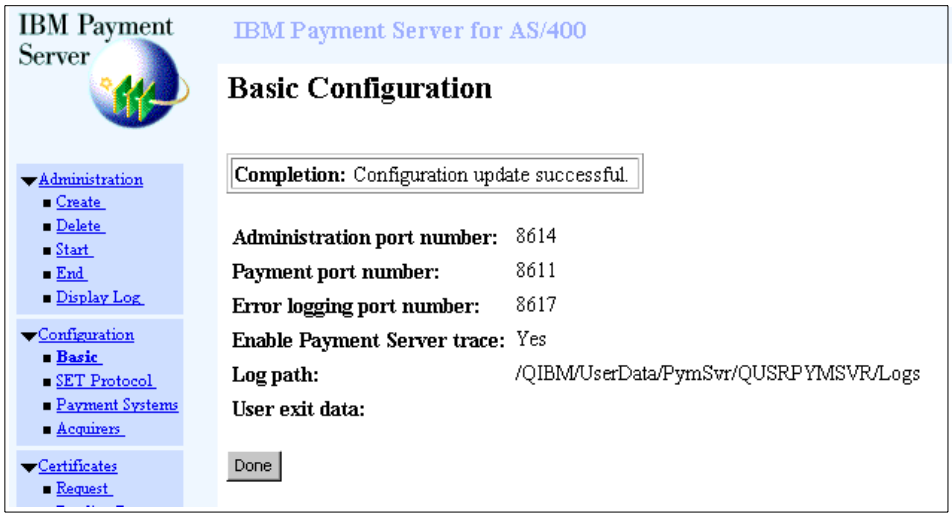


Figure 84. Confirmation screen for basic configuration

8. Click **Done**. You return to the Payment Server for AS/400 screen.
9. Select **SET protocol** from the options on the left. The SET Protocol Configuration screen appears as shown in Figure 85. If you want to enable trace, select the **Yes** radio button.

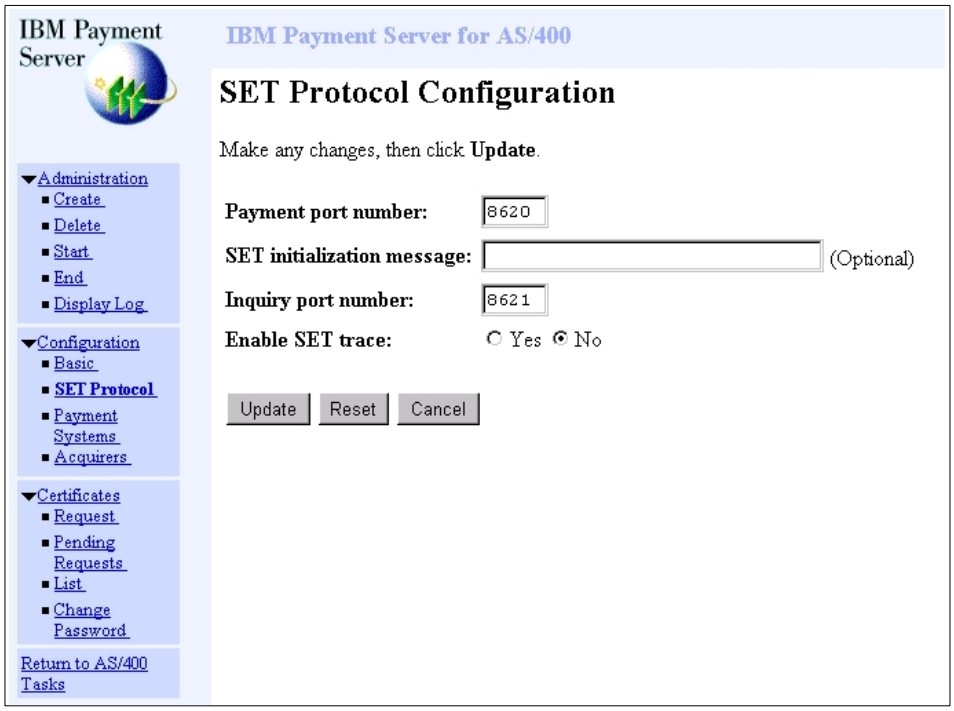


Figure 85. SET Protocol Configuration screen

10. Click **Update** to accept the defaults.

11. You receive a confirmation screen. Click **Done**. This takes you to the Payment Server for AS/400 screen again. Click **Payment Systems** on the left side of the page. The window shown in Figure 86 on page 136 appears.

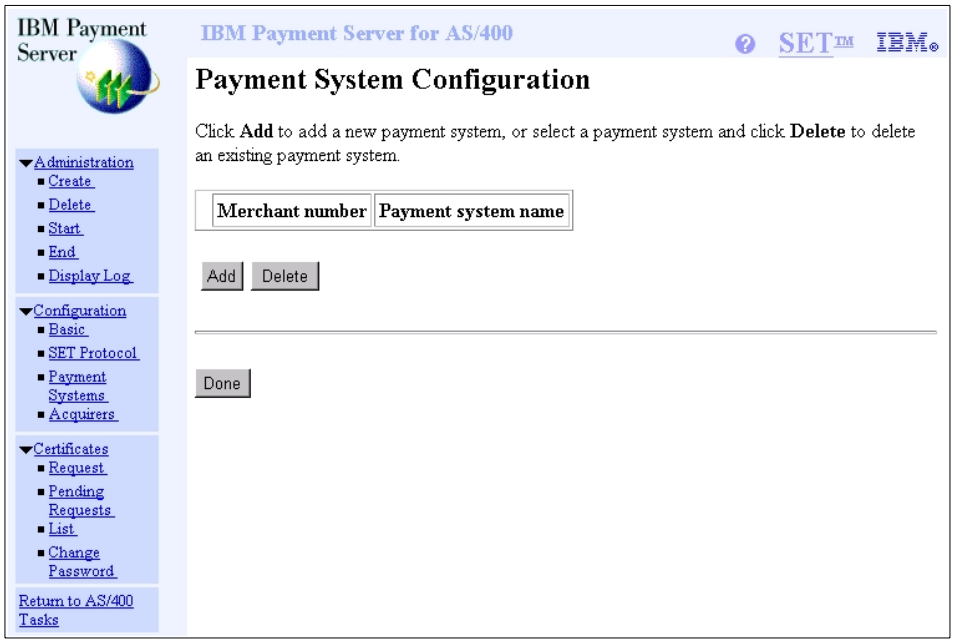


Figure 86. Payment System Configuration screen

12.Click **Add**. The window shown in Figure 87 appears.

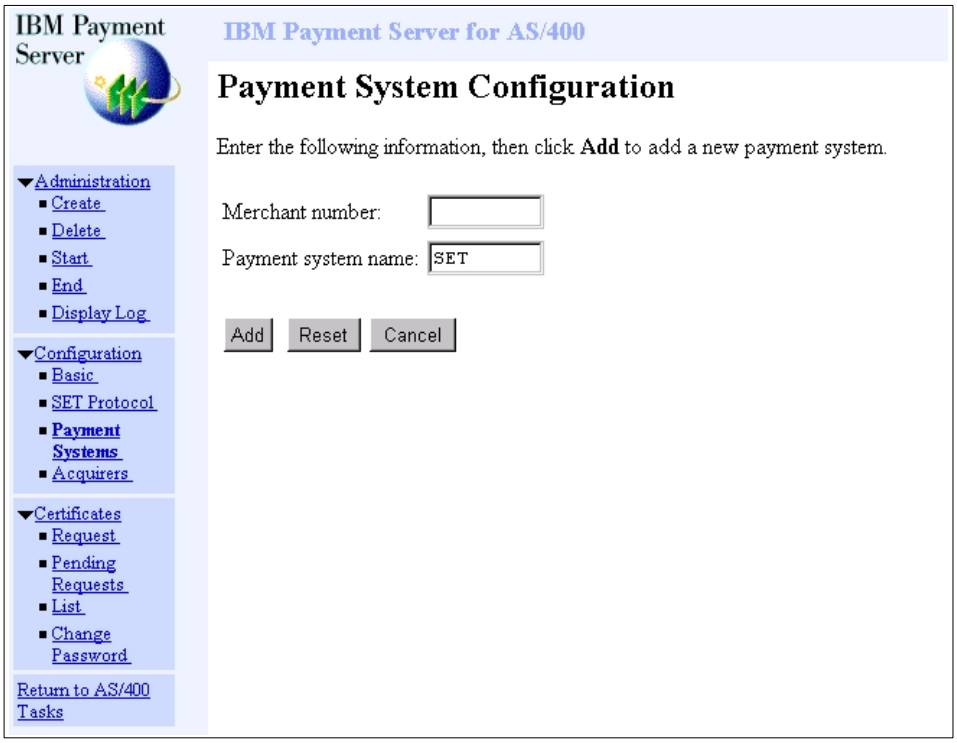


Figure 87. Add payment system configuration

13. In the Merchant number field, enter the merchant number that you want to assign to a particular merchant. This number is used by Payment Server. Leave the Payment System name to SET. Click **Add**. The window shown in Figure 88 on page 138 appears.

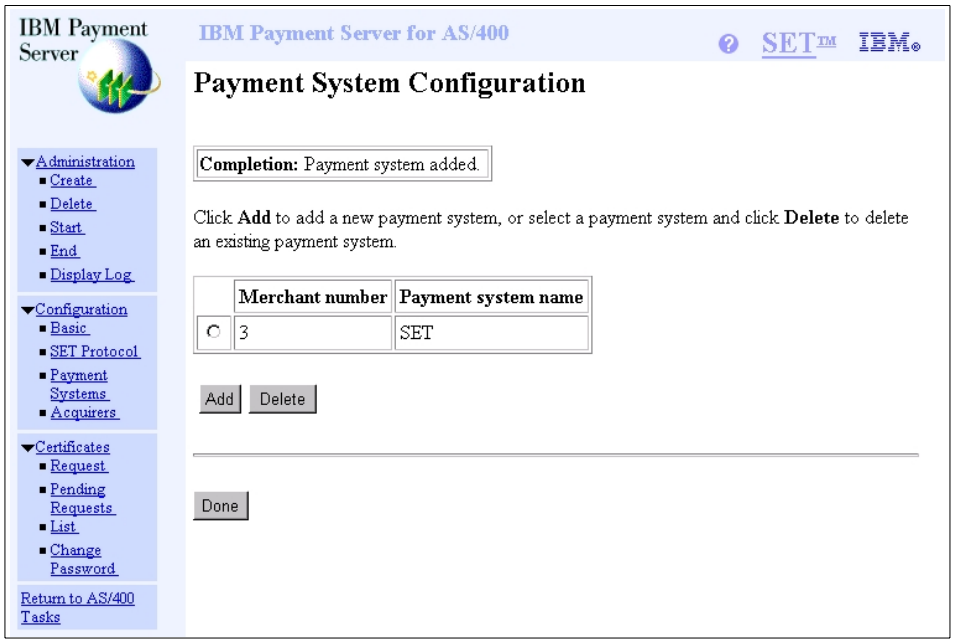


Figure 88. Payment System Confirmation screen

14. Click **Done**. You must configure the acquirer information (in our case, this is NOVA).

15. Click **Acquirers** under Configuration on the left side of the page. The window shown in Figure 89 appears.

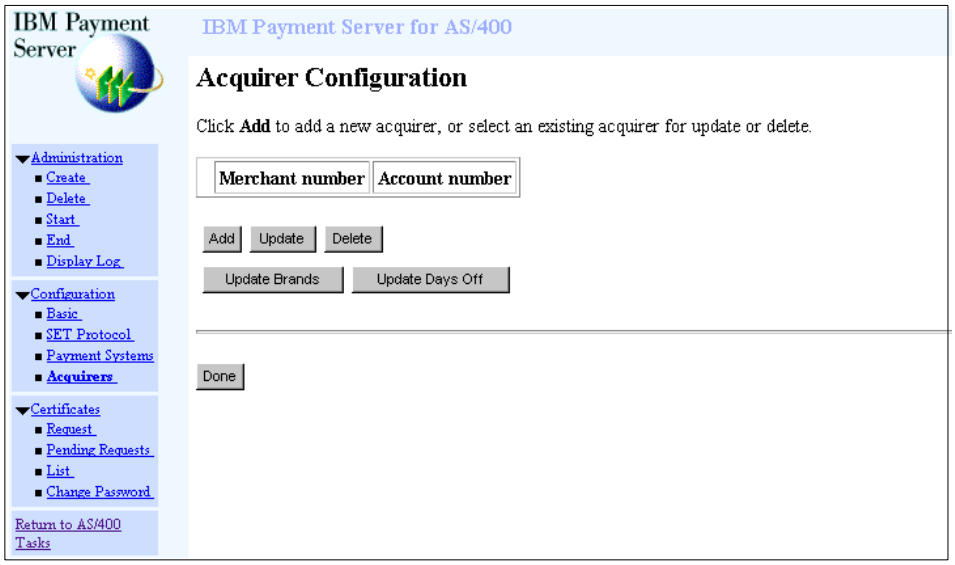


Figure 89. Adding an acquirer configuration screen

16. On this page, select **Add**. The window shown in Figure 90 on page 140 appears.

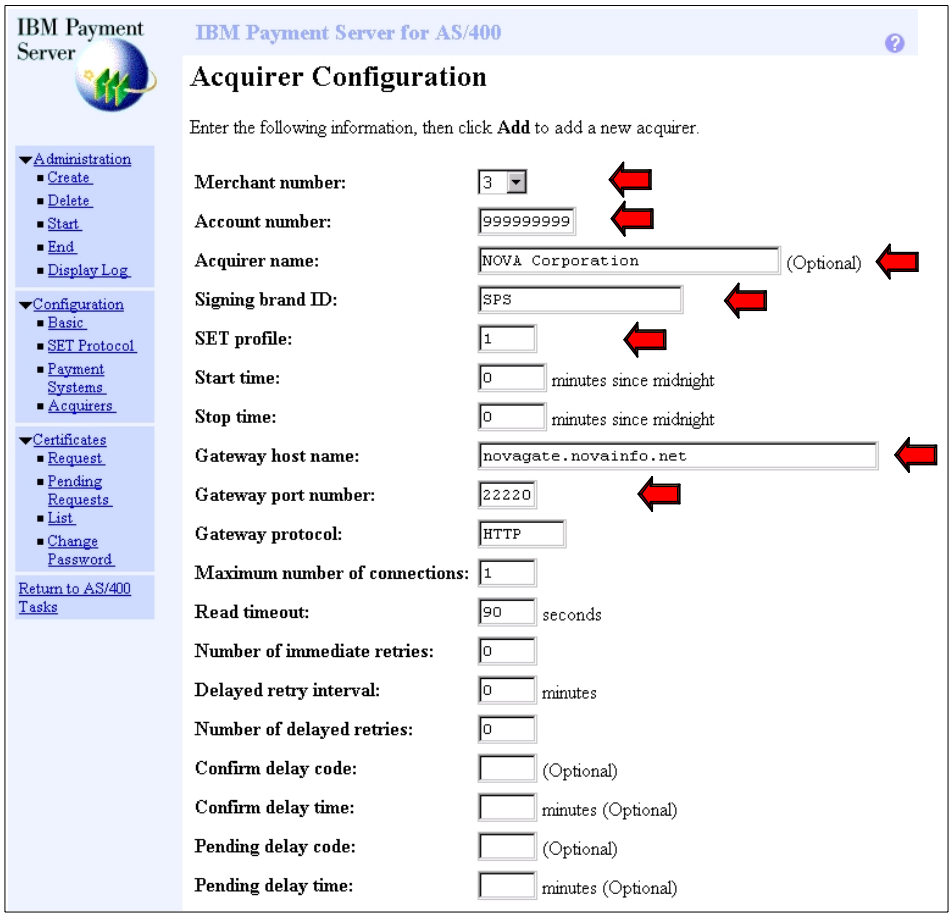


Figure 90. Adding an acquirer profile screen

17. The parameters with the arrows are the ones that need to be changed from the defaults. Change the parameters that are shown in Figure 90. Leave the rest as the defaults, and click **Add**. The window shown in Figure 91 appears.

> **Note**
>
> The “9s” represented here are for demonstration purposes only. If you use NOVA as your acquirer, your account number will be the last 9 digits of your merchant ID number that they provide you.

You see the confirmation screen with your merchant number that you assigned to the merchant and the account number from NOVA for this particular acquirer-merchant relationship.

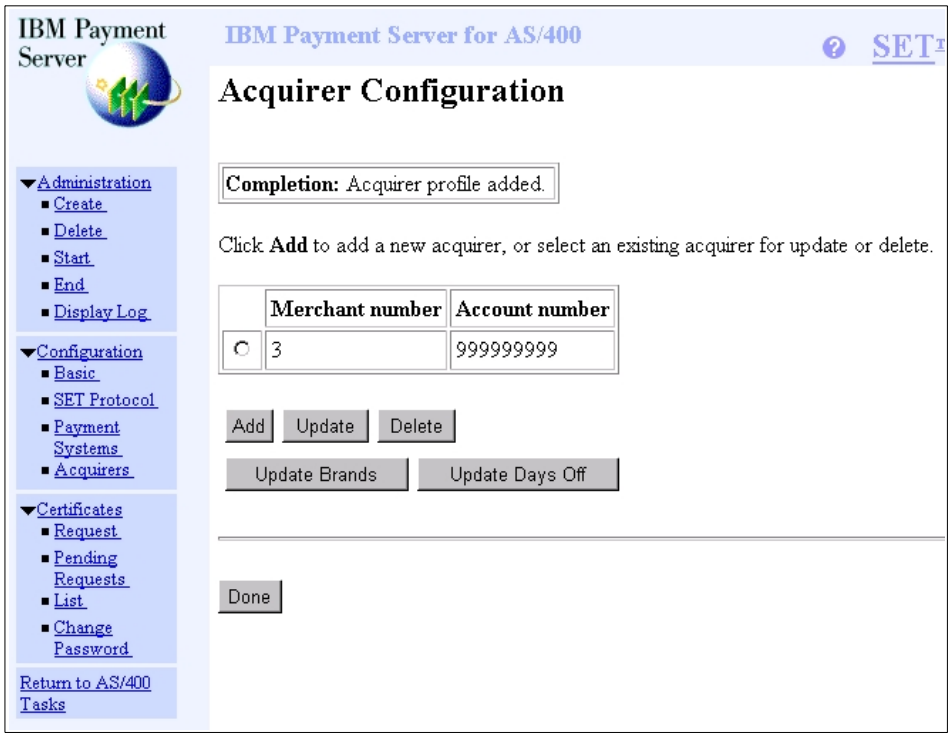


Figure 91. Confirmation screen addition of the acquirer profile

18. Configure the brands that are going to be used by this particular merchant (Worldshare) with this acquirer (NOVA). Select the radio button next to the merchant number, and then click **Update Brand**. The window shown in Figure 92 on page 142 appears.

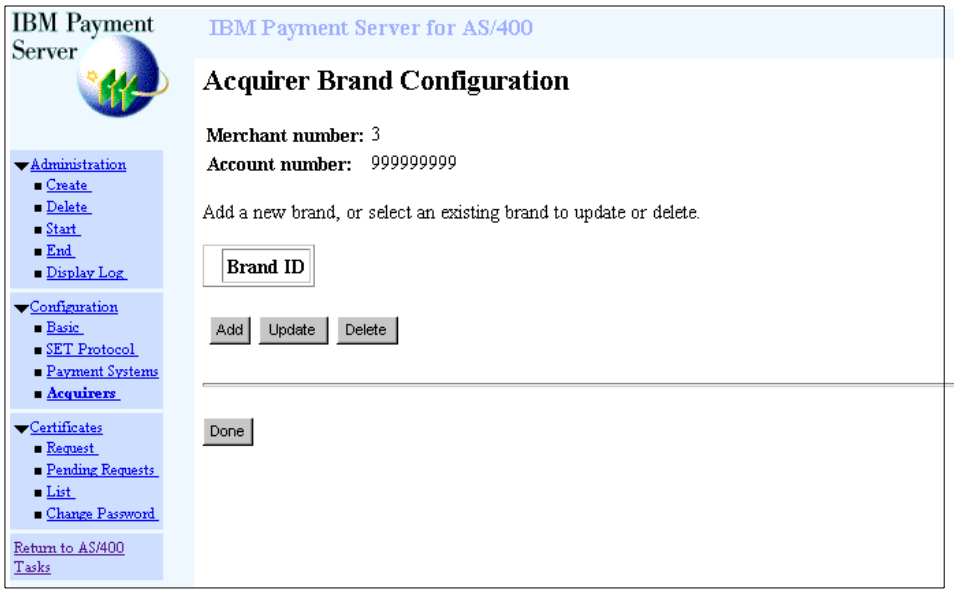


Figure 92. Acquirer Brand Configuration screen

19.On this screen, click **Add**. The window shown in Figure 93 appears.

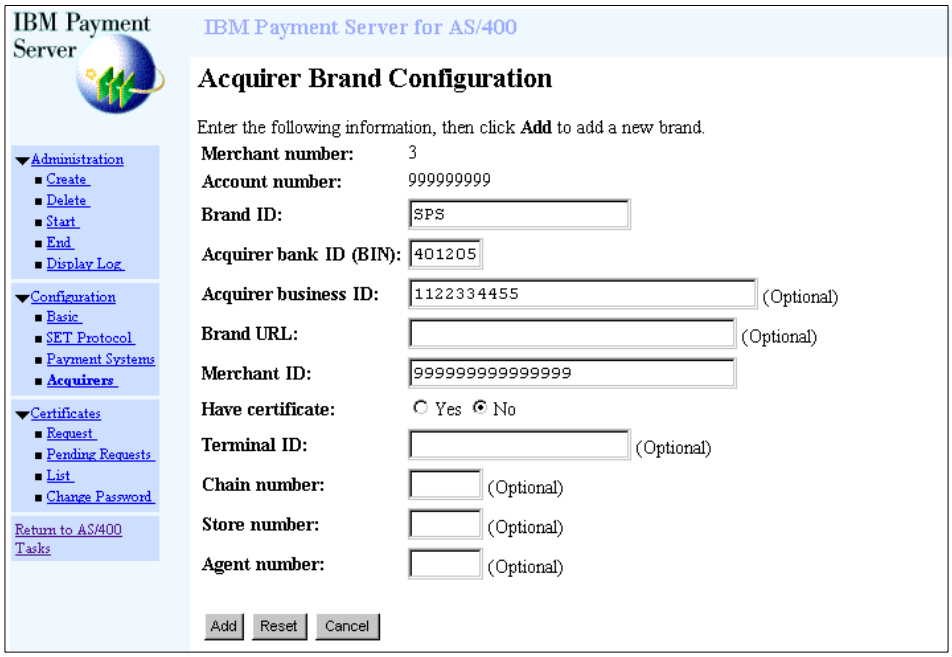


Figure 93. Acquirer Brand Configuration profile

20. Enter all the values obtained from NOVA. Select **Add**. The window shown in Figure 94 on page 144 appears.

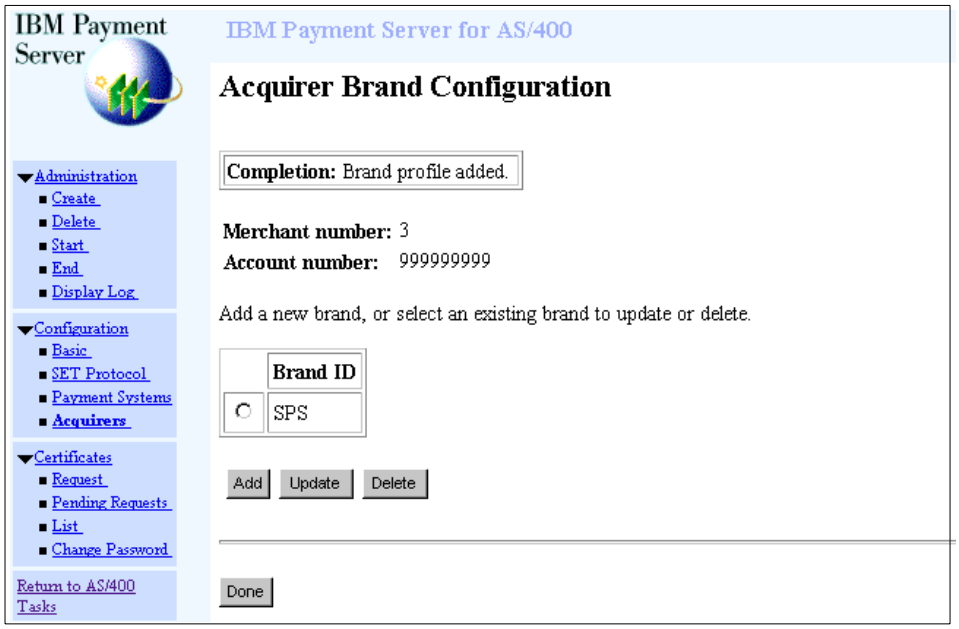


Figure 94. Brand profile addition confirmation screen

Next, you have to request certificates from CyberTrust. This is exactly the same process that is described in 2.2.10, “Requesting a SET merchant certificate from a CA” on page 46. The factors that are different are that the Brand ID will be SPS and, obviously, you will see the policy statement from CyberTrust. In addition to these differences, you will be asked for an e-mail address. There is an automatic turnaround with CyberTrust SETCo certificates, so you see the same Request complete screen as in Figure 32 on page 53. This completes the Payment Server configuration using NOVA as an acquirer.

Appendix A. Special notices

This publication is intended to help system administrators and programmers to plan, install, and configure IBM Payment Server for AS/400 Version 1 Release 2. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM Payment Server for AS/400 V1.2. See the PUBLICATIONS section of the IBM Programming Announcement for IBM Payment Server for AS/400 V1.2, for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate

them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AS/400	AT
CT	IBM
IBM Consumer Wallet	IBM Payment Gateway
IBM Payment Registry	IBM Payment Server
Netfinity	NetView
OS/400	RS/6000
SP	System/390
400	

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix B. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 IBM Redbooks publications

For information on ordering these ITSO publications, see “How to get IBM Redbooks” on page 151.

- *AS/400 Internet Security: IBM Firewall for AS/400,* SG24-2162
- *Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*, SG24-4978
- *Net.Commerce V3.2 for AS/400: A Case Study for Doing Business in the New Millennium,* SG24-5198
- *IBM Firewall for AS/400 V4R3: VPN and NAT Support*, SG24-5376

B.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

B.3 Other resources

These publications are also relevant as further information sources:

- *Getting Started with IBM Firewall for AS/400*, SC41-5424
- *IBM Payment Server for AS/400, Version 1.2 Administrator's Guide*, SC31-8696. This book is only available online as a PDF at:
  `http://www-4.ibm.com/software/webservers/commerce/payment/download/lepl0mst.pdf`
- *IBM Payment Server for AS/400, Version 1.2 Programmer's Guide and Reference,* SC31-8697. This book is only available online as a PDF at:
  `http://www-4.ibm.com/software/webservers/commerce/payment/download/leol0mst.pdf`

B.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- Visit the SETCo home page at: `http://www.setco.org`
- The IBM Payment Suite home page is located at:
  `http://www.ibm.com/software/commerce/payment/`
- If you want more information about the SET Profile, refer to the Web address at:
  `http://www.ibm.com/software/webservers/paymgr/support/acqprofile.html`
- If you need more information about the SET profile, see the following Web site:
  `http://www.ibm.com/software/webservers/paymgr/support/acqprofile.html`
- Worldshare installed the Payment Server software and verified the latest PTFs on the Web at the AS/400 Technical Support home page at:
  `http://as400service.rochester.ibm.com`

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

T

V

W

IBM Redbooks evaluation

Payment Server V1.2 for AS/400: Secure Transactions in e-commerce
SG24-5199-00

Your feedback is very important to help us maintain the quality of IBM Redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other Redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5199-00
Printed in the U.S.A.

Payment Server V1.2 for AS/400: Secure Transactions in e-commerce

SG24-5199-00