IBM WebSphere Commerce

Payments Cassette for KitCash Supplement

Version 5.5

IBM WebSphere Commerce

Payments Cassette for KitCash Supplement

Version 5.5

Note

Before using this information and the product it supports, be sure to read the general information under "Notices", on page 37.

Sixth Edition (July 2003)

This edition applies to a sample payment cassette, the KitCash Cassette, that can be used with version 5.5 of IBM WebSphere Commerce. Make sure you are using the correct edition for the level of the product.

Contents

Welcome!

This book describes the Cassette for KitCash, a sample payment cassette that is intended to illustrate the use of various features of the Payments component of IBM® WebSphere®Commerce (hereafter referred to as WebSphere Commerce Payments).

Note: IBM WebSphere Commerce Payments for Multiplatforms was previously known as IBM WebSphere Payment Manager for Multiplatforms. Starting with version 3.1.3, the payments application was renamed to WebSphere Commerce Payments and references to the product were changed throughout this document. References to the former product may still appear in this document and apply to earlier releases of the product.

This book is for programmers who develop payment cassettes for WebSphere Commerce Payments. The intended audience is experienced Java™ programmers with a strong background in the field of electronic payment processing.

Before reading this book or writing a payment cassette, you should be very familiar with the following information:

- *IBM WebSphere Commerce Payments Programming Guide and Reference*
- *IBM WebSphere Commerce Payments Cassette Kit Programming Guide*, provided in the WebSphere Commerce Payments Cassette Kit Developer's Toolkit.

If you are not familiar with WebSphere Commerce Payments programming interfaces, you should learn about them now.

In addition, the following documents may be referenced in this document:

- *IBM WebSphere Commerce Installation Guide*
- *IBM WebSphere Commerce Studio Installation Guide for WebSphere Commerce Studio*
- *IBM WebSphere Commerce Store Development Guide*
- *IBM WebSphere Commerce Administration Guide*

Conventions in this book

This book uses the following highlighting conventions:

- **Boldface** type indicates commands or graphical user interface (GUI) controls such as names of fields, icons, or menu choices.
- `Monospace` type indicates examples of text you enter exactly as shown, file names, and directory paths and names.
- *Italic* type is used to emphasize words. Italics also indicate names for which you must substitute the appropriate values for your system. When you see the following names, substitute your system value as described.

Windows indicates information specific to the Windows® operating environment.

AIX indicates information specific to AIX®.

Solaris indicates information specific to the Solaris Operating Environment.

400 indicates information specific to the IBM iSeries™ 400® (formerly called AS/400®).

Linux indicates information specific to Linux.

WC_installdir represents the following default installation paths for WebSphere Commerce:

AIX `/usr/lpp/WebSphere/CommerceServer`*nn*

Linux Solaris `/opt/WebSphere/CommerceServer`*nn*

Windows *drive*`:\WebSphere\CommerceServer`*nn*

400 `/QIBM/ProdData/CommerceServer`*nn*

Payments_installdir represents the following default installation paths for WebSphere Commerce Payments:

AIX `/usr/lpp/WebSphere/CommerceServer`*nn*`/payments`

Linux Solaris `/opt/WebSphere/CommerceServer`*nn*`/payments`

Windows *drive*`:\WebSphere\CommerceServer`*nn*`\payments`

400 `/QIBM/ProdData/CommercePayments/V`*nn*

WAS_installdir represents the following default installation path for WebSphere Application Server:

AIX `/usr/WebSphere/AppServer/`

Linux Solaris `/opt/WebSphere/AppServer/`

400 `/QIBM/ProdData/WebAS5/Base/`*WAS_instancename*`/`

Windows *drive*`:\Program_Files\WebSphere\AppServer\`

WAS_userdir represents the following default directory (for data that is used by WebSphere Application Server which can be modified or needs to be configured by the user):

400 `/QIBM/UserData/WebAS5/Base/`*WAS_instancename*`/installedApps`

Additional information

In addition to the information specified above, this book can also be used in conjunction with the sample cassette LDBCard, and its associated documentation. LDBCard is a fully-functioning cassette, and is meant to serve as a skeleton you can use to build your cassettes. You can download the LDBCard package from the same Web site you downloaded this book from.

Chapter 1. Overview of Cassette for KitCash

The Cassette for KitCash is a sample payment cassette that implements a fictional electronic cash (stored value) protocol. The Cassette for KitCash is designed to show you how to use WebSphere Commerce Payments to support an internet payment system.

KitCash is an example for cassette writers that covers several aspects of cassette writing not covered by the LDBCard skeleton cassette. KitCash is **not** intended to serve as a skeleton upon which cassette writers will typically build new cassettes. You should use the LDBCard cassette for that purpose.

KitCash illustrates these unique features:

- An example of a payment protocol that is not oriented towards credit cards. The fictional KitCash stored value protocol supports a limited set of financial transactions. For example, the KitCash protocol does not support reversals or credits. As a result, the cassette supports a limited subset of the WebSphere Commerce Payments API set to match the needs of the payment protocol.
- How to support the ReceivePayment command, including:
  - The definition of protocol messages representing a set of associated protocol flows from a buyer's wallet.
  - The use of ComPoints to manage the protocol messages.
- How to use CassetteWorkItems and the framework's service thread queue.
- How to use the Finite State Machine (FSM) tools.
- An alternative cassette design. The alternative cassette design is possible because the framework uses Java interfaces instead of abstract methods to define some of the key cassette objects. Specifically, KitCash implements the CassetteOrder and CassetteTransaction (representing a payment) interfaces in a single object named KitCashPurchase.

KitCash provides full support for WebSphere Commerce Payments administration commands and user interface according to the needs of the payment protocol. A test harness is provided for the Cassette for KitCash. The test harness shows how the cassette interacts with other components in an internet payment system.

Protocol overview

A bank that supports the KitCash payment scheme can issue KitCash cards to both consumers and merchants. The bank assigns an account number for each card. When a consumer uses a card to make an internet payment from a merchant, money transfers from the consumer's card onto the card the merchant selected. The merchant can decide which KitCash card the money is transferred to by designating an account number for each available product. At any time, the merchant can transfer the money on their cards to the bank. The bank then credits the merchant's account with the appropriate funds.

When a consumer decides to use his KitCash card to buy a product, the payment must be for the full purchase price of the product. Partial payments and refunds are not supported by the KitCash payment scheme.

The following diagram illustrates the key components involved in KitCash transactions.

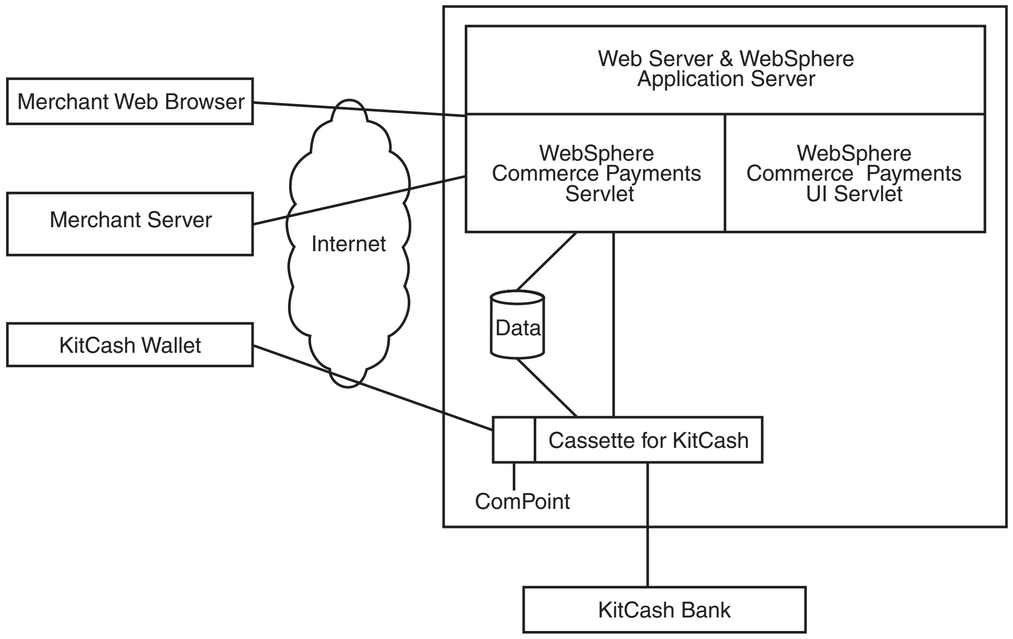

Web Server & WebSphere Application Server
Merchant Web Browser
WebSphere Commerce Payments Servlet
WebSphere Commerce Payments UI Servlet
Merchant Server
Internet
KitCash Wallet
Data
Cassette for KitCash
ComPoint
KitCash Bank

Figure 1. Components in KitCash transactions

Chapter 2. KitCash and WebSphere Commerce Payments Concepts

WebSphere Commerce Payments provides a unified interface through which merchants can use multiple payment protocols in a common way. Each WebSphere Commerce Payments cassette attempts to extract protocol-specific differences so that merchants can ignore disparities between protocols.

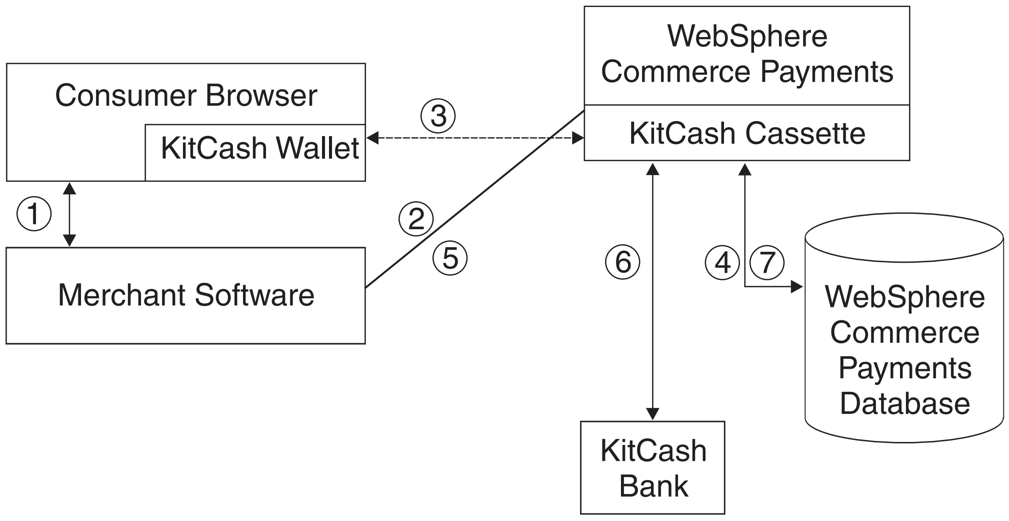
This section describes how the Cassette for KitCash presents the fictional electronic cash protocol through the WebSphere Commerce Payments object model and API set.

The Cassette for KitCash implements the payment commands and the payment processing model of the WebSphere Commerce Payments framework, using the processing services of the KitCash test harness described in “Test harness” on page 7. This implementation supports:

- Wallet-driven purchases only, ReceivePayment
- Traditional payment-oriented commands (see “Cassette for KitCash payment command summary” on page 5 for more information)
- Multiple accounts per merchant
- A single batch for an account

A KitCash example

The following is an example of how a typical consumer purchase and merchant acquisition of funds would take place using the Cassette for KitCash. For low level details, refer to “Cassette sequence diagrams” on page 25.



1. A consumer has been shopping online at a merchant Web site. After choosing several items to purchase, the consumer initiates a purchase, typically by pressing a ″Buy″ button on the shopping page.
2. The merchant software creates an order and requests that the cassette and consumer exchange additional information to confirm payment using the consumer’s KitCash card.

3. The Cassette for KitCash receives the consumer’s KitCash card information via protocol message exchanges with the consumer’s KitCash wallet.
4. The Cassette for KitCash marks the payment as approved and deposits the payment in the appropriate batch. In addition, the Cassette for KitCash and WebSphere Commerce Payments update status in the database.
5. At some point in time, the merchant settles open batches by sending a BatchClose request to WebSphere Commerce Payments.
6. The Cassette for KitCash exchanges protocol messages with the KitCash bank.
7. The Cassette for KitCash closes the batch and updates the appropriate financial objects. The Cassette for KitCash along with WebSphere Commerce Payments updates status in the database and returns a success response to the merchant.

WebSphere Commerce Payments object model implementation

This section describes how the Cassette for KitCash supports the administrative and financial object models that the WebSphere Commerce Payments framework provides.

Administrative objects

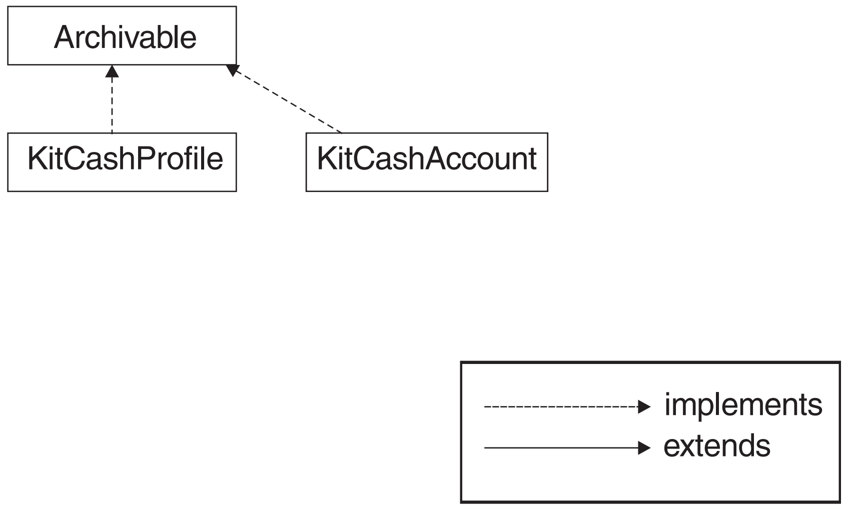
WebSphere Commerce Payments administration objects are the entities that comprise the system and merchant configuration under which all financial transactions will be performed. Refer to the *IBM WebSphere Commerce Payments Cassette Kit Programming Guide* for a description of WebSphere Commerce Payments administration objects. The Cassette for KitCash augments two of the framework administration objects with its own attributes.

Cassette Admin object

The CassetteAdmin object represents the cassette itself and contains attributes that apply globally across the cassette. The Cassette for KitCash extends this object with KitCashProfile. The KitCashProfile object contains a protocol port attribute used when listening for incoming protocol messages from the consumer’s wallet.

Account Admin object

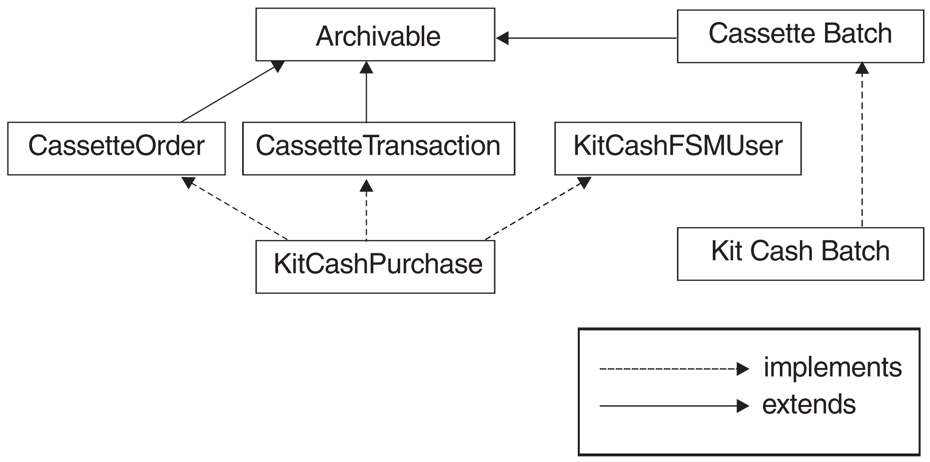
In the WebSphere Commerce Payments object model, the AccountAdmin object represents a relationship between a given merchant and a given financial institution. This is exactly the type of relationship that each KitCash merchant account represents. The cassette extends the WebSphere Commerce Payments AccountAdmin with KitCashAccount. The KitCashAccount object contains attributes that identify and describe the corresponding merchant account, for example Bank hostname and Bank port.



Financial objects

WebSphere Commerce Payments financial objects are used to represent the financial transactions executed by merchants. As mentioned in the introduction, the Cassette for KitCash implements a fictitious electronic cash protocol. The KitCash protocol doesn’t support the concept of reversals or refunds. Since the protocol doesn’t support refunds, the Credit object is not needed. The Cassette for KitCash uses an alternative design in that it provides a single object, KitCashPurchase, to provide extensions to the following financial objects:

- Order objects
- Payment objects



Cassette for KitCash payment command summary

Table 1 summarizes the way the Cassette for KitCash handles each of the WebSphere Commerce Payments payment commands (commands that carry out financial transactions). Specifically, for each payment command, the table shows:

- How the cassette processes the command:
  – ″Not supported by cassette″ means the cassette does not support that particular command. These commands will always receive return codes PRC_CASSETTE_ERROR, RC_NONE.

– ″Handled by WebSphere Commerce Payments; no message sent″ means that the command is processed completely within WebSphere Commerce Payments without communicating with the KitCash bank.

Table 1. Cassette for KitCash. Summary of Payment API Commands

API command	KitCash message
AcceptPayment	Not supported by cassette.
Approve	Not supported by cassette.
ApproveReversal	Not supported by cassette.
BatchClose	KitCash transaction with bank.
BatchOpen	Not supported by cassette (cassette opens batches internally as needed).
BatchPurge	Not supported by cassette.
CancelOrder	Not supported by cassette.
CloseOrder	Handled by WebSphere Commerce Payments; no message sent.
DeleteBatch	Handled by WebSphere Commerce Payments; no message sent.
Deposit	Not supported by cassette (cassette implicitly deposits when a ReceivePayment is processed).
DepositReversal	Not supported by cassette.
ReceivePayment	Cassette and KitCash Wallet exchange protocol messages.
Refund	Not supported by cassette.
RefundReversal	Not supported by cassette.

Summary of state changes

The following table summarizes the state changes that Order, Payment, and Batch objects undergo as a result of successful completion of each payment command. Only those objects whose states actually change as a result of the given operation are shown. Any other existing object states remain unchanged.

API Command	Object States
ReceivePayment	ORDER_ORDERED PAYMENT_DEPOSITED
BatchClose	BATCH_CLOSED PAYMENT_CLOSED
CloseOrder	ORDER_CLOSED

Chapter 3. Installing the Cassette for KitCash

This chapter describes how to install the Cassette for KitCash on AIX, Linux, Solaris, Windows, and iSeries platforms.

Test harness

The KitCash card is implemented in software by a KitCashDriver class. No special hardware is required to run the sample code. The KitCash test harness allows you to see how the cassette interacts with other internet payment software. The KitCash test harness includes:

- KitCash wallet (Java applet)
- KitCash bank (Java application) to receive deposits from a merchant

Before installing Cassette for KitCash

Before installing the Cassette for KitCash, ensure that you have done the following:

- Installed WebSphere Commerce Version 5.5 with the WebSphere Commerce Payments component
- Created a WebSphere Commerce Payments instance

WebSphere Commerce and the Payments component must be installed before the Cassette for KitCash can be installed. The minimum Payments framework level supported by the cassette is 5.5. For detailed information on the installation of WebSphere Commerce and the Payments component, refer to the *WebSphere Commerce Installation Guide, Version 5.5*, for your platform.

The Payments instance you intend to use with the Cassette for KitCash should *not* be started or running when you install this cassette.

Installing the Cassette for KitCash

You can install the Cassette for Kitcash as a sample cassette for use with:

- WebSphere Commerce: WebSphere Commerce Professional Edition or Business Edition
- WebSphere Commerce Studio: WebSphere Commerce Studio Professional Edition and Business Developer Edition for Windows 2000 (for development use and modification)

Use the correct installation instructions for your environment.

Installing on WebSphere Commerce

To install the Cassette for KitCash on WebSphere Commerce Professional Edition or Business Edition, do the following:

1. Uncompress the `kitCashCassette55.zip` file to the following directory:

   Windows AIX Solaris Linux

   WC_installdir/payments/cassettes

   400

   Payments_installdir/cassettes
2. Start the WebSphere Commerce Configuration Manager. Because you uncompressed the zip file in the cassettes directory shown previously, the Configuration Manager will locate the appropriate files to add to your Payments configuration. For information about how to start the Configuration Manager, refer to the *WebSphere Commerce Installation Guide*.
3. Select the Payments instance to which you want to add the Cassette for KitCash (select **WebSphere Commerce** > *host_name* > **Payments** >**InstanceList**.
4. Use the **Cassettes** page of the Configuration Manager to add the Cassette for KitCash to the Payments instance.
5. Restart the Payments instance (right-click on the Payments instance and select **Start Payments Instance**). For complete instructions on starting and stopping a WebSphere Commerce Payments instance, refer to the *WebSphere Commerce Installation Guide*.

See "Installing the KitCash wallet" for additional mandatory installation tasks.

Installing the KitCash wallet

The Cassette for KitCash supports order creation through a wallet, and therefore uses the Payments ReceivePayment API command. In the Cassette for KitCash, the wallet is implemented through an applet (`KitCashWallet.jar`) which is downloaded into a the Web browser when the ReceivePayment transaction occurs. As part of your cassette installation, you must copy the `KitCashWallet.jar` file into the merchant directory in the `PaymentRuntime.war` directory to be able to create orders through a wallet.

To install a KitCash wallet in WebSphere Commerce, follow these procedures:

1. Create a folder called 'merchant' in the following directory:

   WAS_installdir/installedApps/*host_name*/*payments_instance*_Commerce_Payments_App.ear/
   Payments.war

   400

   WAS_userdir/installedApps/*node_name*/*payments_instance*_Commerce_Payments_App.ear/
   Payments.war
2. Copy the following file:

   `WC_installdir/payments/cassettes/KitCash/lib/KitCashWallet.jar`

   to the following directory:

   WAS_installdir/installedApps/*host_name*/*payments_instance*_Commerce_Payments_App.ear/
   Payments.war/merchant

   400 Copy the following file:

   Payments_installdir/cassettes/KitCash/lib/KitCashWallet.jar to the following directory:

```
WAS_userdir/installedApps/node_name/payments_instance_Commerce_Payments_App.ear/
   Payments.war/merchant
```

The KitCash wallet is now installed in the proper location.

Proceed with “Using the Cassette for KitCash with a sample store” on page 14 for optional post-installation tasks before moving on to Chapter 5, “Getting Started”, on page 19.

Installing on WebSphere Commerce Studio

To install the Cassette for KitCash on WebSphere Commerce Studio Professional Edition or Business Developer Edition for use in a test environment, follow these procedures.

▶ Windows

1. Uncompress the `kitCashCassette55.zip` file to the following directory:

   ```
   WC_installdir/payments/cassettes
   ```

   After the zip file is uncompressed, you should see the following in the cassettes directory:

   ```
   cassette_properties.xml
   KitCashFSM.prj
   readme.kitcash.html
   WC55KitCashSupplement.pdf
   /bin
   /javadoc
   /lib
   /Payments-KitCashCassette
   /pspl
   /SampleCheckout
   /schema
   ```

   The `Payments-KitCashCassette` directory contains the project file for the Cassette for KitCash.
2. Import the .project file into the WebSphere Commerce workspace containing the Payments component:
   a. Start WebSphere Commerce Studio.
   b. Open the J2EE perspective. Select **Window > Open Perspective > J2EE**. Go to the J2EE Navigator view.
   c. Import the KitCash project into the WebSphere Commerce workspace by doing the following:
      1) Select **File > Import**. The Import Wizard starts.
      2) Select **Existing Project into Workspace** and click **Next**.
      3) On the Import Project page of the wizard, click **Browse**.
      4) In the Browse for Folder dialog, locate the folder with the KitCash project. Select the folder and click **OK**.

         The default workspace directory for WebSphere Commerce is:

         ```
         c:\WebSphere\workspace_db2      or
         c:\WebSphere\workspace_oracle
         ```

         depending on the target database type you chose in the WebSphere Commerce Studio installation wizard.
      5) Click **Finish**.

After the project is imported, you must complete additional steps to add the Cassette for KitCash assets to the WebSphere Studio Application Developer environment as described in the following sections:

- Add the `eTillKitCashClasses.jar` file to the list of modules.
- Set JAR dependencies.
- Add the `KitCash.PSPL` file to the PSPL folder.
- Import the KitCash Cashier profile.
- Import the `SampleCheckoutKitCash.properties` file.
- Enable the SampleCheckout application to run the Cassette for KitCash.
- Import the KitCash wallet

After assets are added, you can then add the cassette to a Payments instance.

Adding the eTillKitCashClasses.jar file

To add the `eTillKitCashClasses.jar` file to WebSphere Commerce Studio, do the following:

1. Start WebSphere Commerce Studio.
2. Open the J2EE perspective. Select **Window > Open Perspective > J2EE**. Go to the J2EE Hierarchy view.
3. Expand **Enterprise Applications** and right-click on **WebSphereCommercePaymentsServer**.
4. Select **Open with > Deployment Descriptor Editor** in the pop-up menu.
5. Click the **Module** tab.
6. In the Project Utility JARs section, click **Add**.
7. Click **Payments-KitCashCassette**. In the URI field, enter:

   ```
   lib/eTillKitCashClasses.jar
   ```

   and click **Finish**.

The `eTillKitCashClasses.jar` file is added to the project utility JARs list. Proceed with the next section.

Setting JAR dependencies

In the J2EE Hierarchy view of WebSphere Commerce Studio, do the following to set the JAR dependencies for WebSphere Commerce Payments and the Sample Checkout application:

1. Expand **Web Modules** and right-click on **WebSphereCommercePaymentsServerRuntime**.
2. Select **Open with > JAR Dependency Editor** in the pop-up menu.
3. Select the check box for `lib/eTillKitCashClasses.jar` and save the editor contents.
4. Expand **Web Modules** and right-click on **WebSphereCommercePaymentsSampleCheckout**.
5. Select **Open with > JAR Dependency Editor** in the pop-up menu..
6. Select the check box for `lib/eTillKitCashClasses.jar` and save the editor contents.

The JAR dependencies are now defined. Proceed with the next section.

Adding the KitCash PSPL file

In the J2EE Navigator view of WebSphere Commerce Studio, do the following:

1. Expand **WebSphereCommercePaymentsServerRuntime**.
2. Expand **Web Content**.
3. Right-click the **pspl** folder.
4. Click **File system** and then click **Next**.
5. In the Browse for Folder dialog, select **WebSphere > Commerce Studio 55> Commerce > Payments > Cassettes > KitCash > pspl**.
6. On the Import dialog, click **Finish**.

The `KitCash.PSPL` is added to the pspl folder. Proceed with the next section.

Importing the KitCash Cashier profile

In the J2EE Navigator view of WebSphere Commerce Studio, do the following:

1. Select **WebSphereCommercePaymentsSampleCheckout > Web Content > profiles**.
2. Right-click the **profiles** folder and then click **import**.
3. In the Import dialog, click **File system**, then **Next**.
4. Browse to find `*WC_installdir*/payments/cassettes/KitCash/SampleCheckout/profiles.`
5. Select the checkbox for **select the profile**.
6. Click **Finish**.

The KitCash Cashier profile (`SampleCheckoutKitCash.profile`) is added. Proceed with the next section.

Importing the SampleCheckoutKitCash.properties file

In the J2EE Navigator view of WebSphere Commerce Studio, do the following:

1. Select **WebSphereCommercePaymentsSampleCheckout > WEB-INF**.
2. Right-click the **classes** folder and then click **import**.
3. In the Import dialog, click **File system**, then **Next**.
4. Browse to find `*WC_installdir*/payments/cassettes/KitCash/SampleCheckout/properties`.
5. Select the checkbox for **`SampleCheckoutKitCash.properties`**.
6. Click **Finish**.

The `SampleCheckoutKitCash.properties` file is added. Proceed with the next section.

Enabling the SampleCheckout application to run the Cassette for KitCash

In the J2EE Navigator view of WebSphere Commerce Studio, do the following:

1. Select **WebSphereCommercePaymentsSampleCheckout > Web Content**.
2. Open the `SampleCheckout.xml` file.
3. Edit the `SampleCheckout.xml file` and add the following line to the PaymentOptionList element:
   ```
   <PaymentOption id="KitCash" profile="SampleCheckoutKitCash">KitCash</PaymentOption>
   ```
4. Save the updates to the XML file (save editor contents).

The SampleCheckout application is now enabled to run the Cassette for KitCash.

Importing the KitCash wallet to the merchant directory

The Cassette for KitCash supports order creation through a wallet, and therefore uses the Payments ReceivePayment API command. In the Cassette for KitCash, the wallet is implemented through an applet (`KitCashWallet.jar`) which is downloaded into a the Web browser when the ReceivePayment transaction occurs. As part of your cassette installation, you must copy the `KitCashWallet.jar` file into the merchant directory in the `PaymentRuntime.war` file to be able to create orders through a wallet.

To install a KitCash wallet in WebSphere Commerce Studio, follow these procedures. In the J2EE Navigator view of WebSphere Commerce Studio, do the following:

1. Expand **WebSphereCommercePaymentsServerRuntime**.
2. Right-click **Web Content**.
3. Select **New > Folder**.
4. Enter the folder name: `merchant`, and then click **Finish**.
5. Right-click the merchant folder.
6. Select **Import**.
7. Click **File system** and then click **Next**.
8. In the Browse for Folder dialog, select **WebSphere > Commerce Studio 55 > Commerce > Payments > Cassettes > KitCash > lib > KitCashWallet.jar**.
9. Click **Finish**.

The KitCash wallet is now installed in the proper location.

After you complete this last step, you have completed adding code assets to WebSphere Commerce Studio. You can proceed with adding the cassette to a Payments instance.

Adding the cassette to a Payments instance

After all Cassette for KitCash assets have been added to WebSphere Commerce Studio, you can add the Cassette for KitCash to a Payments instance and start the Payments server. Follow these procedures to add the cassette to a Payments instance:

1. In WebSphere Commerce Studio, select **Windows > Open Perspective > Server**.
2. Select **Windows > Show View > Other**. In the Show View dialog, expand **Other**.
3. Select **WebSphere Commerce** and click **OK**.
4. In the WebSphere Commerce view, right-click **Configuration Manager Server**, and then click **Start Server** from the pop-up menu.
5. After the Configuration Manager server is started, right-click the server again and click **Run Client** from the pop-up menu. The Configuration Manager client displays.
6. Expand **WebSphere Commerce Payments > Instance List > wpm > Instance Properties> Cassettes**.
7. Select the **KitCash** Cassette in the list of **Available Cassettes** and add it. Click **Apply**.
8. Close the Configuration Manager.
9. In the Server view, start the WebSphere Commerce Payments server.

The WebSphere Commerce Payments server should start successfully with the addition of the Cassette for KitCash.

Post-installation optional tasks

Optional tasks for WebSphere Commerce Studio

Compiling KitCash code in WebSphere Commerce Studio

A sample Java project is provided in the Cassette for KitCash zip file for cassette writers to use with WebSphere Commerce Studio Professional Edition or Business Developer Edition. As previously described, you can install the Cassette for KitCash in WebSphere Commerce Studio and import the KitCash project into the WebSphere Commerce workspace so that you can:

- Understand how payments cassettes can be used with WebSphere Commerce Payments in an internet payment system.
- Learn about the structure of the cassette.
- Experiment with and manipulate cassette assets.

If you are building a new cassette, it is recommended that you not base your new cassette on the KitCash model; but rather, review the KitCash Cassette to understand its unique features. (Features were described in Chapter 1, "Overview of Cassette for KitCash", on page 1.)

If you are installing and using the Cassette for KitCash in WebSphere Commerce Studio, and make changes to any of the cassette assets in Studio, you must compile the cassette code before attempting to run the cassette in WebSphere Commerce. To compile the cassette code, do the following:

1. In the J2EE Navigator view of WebSphere Commerce Studio, select the KitCash cassette project.
2. Select **Project > Build** project to compile the cassette. A successful build should result in no messages being displayed. If the build was unsuccessful, refer to the resulting error messages to investigate and resolve the error.

After the project builds successfully, you can then add the revised cassette to the Payments instance. If you have not already added the cassette to a Payments instance, follow the procedures outlined in "Adding the cassette to a Payments instance" on page 12. If the cassette has already been added to the instance, it is not necessary to delete it from the instance before re-adding it.

Exporting the cassette from WebSphere Commerce Studio for use on other WebSphere Commerce platforms

You can export the Cassette for KitCash resources from WebSphere Commerce Studio, and add them to the WebSphere Commerce Professional Edition or Business Edition if desired by following this procedure:

1. In WebSphere Commerce Studio, select the Java view.
2. Select the **PaymentsKitCashCassette** project.
3. Right-click the project and select **Export** from the pop-up menu.
4. Select **Jar file**, and then click **Next**.
5. In the JAR Export dialog, select the resource to export to the library and save the resources to the folllowing location if the cassette was not already added to WebSphere Commerce Professional Edition or Business Edition:

   WC_installdir\payments\cassettes\KitCash\lib\eTillKitCashClasses.jar

   If the cassette was already added, export the resources to this location:

   WAS_installdir\installedApps*host_name**payments_instance*_Commerce_Payments_App.ear\lib

After the cassette resources are exported to the JAR file, you can then add the JAR file to the `WC_installdir/payments/cassettes` directory and follow procedures in "Installing on WebSphere Commerce" on page 8 (starting with step 2) to add the cassette to your Payments instance.

Other optional tasks

Using the Cassette for KitCash with a sample store

If you would like to use the Cassette for KitCash with one of the sample stores WebSphere Commerce provides, you must customize the WebSphere Commerce environment for the Cassette for KitCash. Customization tasks include:

- Including a payment asset file (`paymentinfo.xml`) in the store archive.
- Modifying the sample store .jsp file to include the name of the Cassette for KitCash.
- Modifying the Cassette for KitCash Cashier profile (optional).

Refer to the Payments instruments chapter in the *WebSphere Commerce Store Development Guide* for complete instructions on how to do this customization. More information about the Cassette for KitCash cashier profile is also provided in Chapter 4, "Cassette for KitCash Cashier profiles", on page 17.

When modifying the store's .jsp file, use the following name for the payment policy:

```
KitCash
```

For example:

```
 if (info[i].getPolicyName().trim().equals("KitCash"))
```

The Cashier profile which is used to create orders in the Payments component for the Cassette for KitCash is called

```
SampleCheckoutKitCash.profile
```

and can be found in this location:

```
WC_installdir/payments/cassettes/KitCash/SampleCheckout/profiles
```

400

```
Payments_installdir/cassettes/KitCash/SampleCheckout/profiles
```

If the sample store you are using supports Quick Checkout, there are other files to update besides this .jsp file. Follow the instructions in the Payments instruments chapter of the *WebSphere Commerce Store Development Guide* to update other possible files, and to store the profile in the proper directory location in WebSphere Commerce.

Updating the Payments port number in the SampleCheckoutKitCash profile

If you are installing the Cassette for KitCash on WebSphere Commerce and intend to use the SampleCheckout application to place an order, you must update the `SampleCheckoutKitCash.profile` file with the correct value of the port on which Payments will run on. If the port is not set correctly, the result page is not displayed properly after the "buy" process completes in SampleCheckout.

By default, the value for the $PORT parameter is set to 9081 in the profile; however, 9081 is the default port for Payments in WebSphere Commerce Studio. This value must be changed to use the SampleCheckout application in WebSphere Commerce.

To update the port value, locate the SampleCheckoutKitCash.profile. If the Cassette for KitCash has already been added to WebSphere Commerce Professional Edition or Business Edition, the `SampleCheckoutKitCash.profile` is in the following location:

```
WAS_installdir/installedApps/host_name/payments_instance_Commerce_Payments_App.ear/
   SampleCheckout.war/profiles/SampleCheckoutKitCash.profile
```

▶ 400

```
WAS_userdir/installedApps/node_name/payments_instance_Commerce_Payments_App.ear/
   SampleCheckout.war/profiles/SampleCheckoutKitCash.profile
```

If the Cassette for KitCash has not been added already, the profile is in this location:

```
WC_installdir/payments/cassettes/KitCash/SampleCheckout/profiles/
   SampleCheckout.profile
```

▶ 400

```
Payments_installdir/cassettes/KitCash/SampleCheckout/profiles/SampleCheckout.profile
```

Edit the file and change the port value from 9081 to 5432 or the correct port value for Payments in your environment:

```
<Parameter name="$PORT"><CharacterText>5432</CharacterText></Parameter>
```

If you are using the cassette in WebSphere Commerce Studio, there should be no need to change the default port value in the profile.

Chapter 4. Cassette for KitCash Cashier profiles

The Cashier is WebSphere Commerce Payments software that can be invoked by client applications (such as merchant software) to simplify the process of creating WebSphere Commerce Payments orders and payments. The Cashier uses XML documents called profiles that describe how orders should be created for a given cassette. This allows the client code writer to concentrate on integrating with WebSphere Commerce Payments in a generic way rather than having to write code that deals with cassette-specific information.

It is still possible to create WebSphere Commerce Payments orders without using the Cashier; programs can use the client access library or the HTTP/XML interface to use the API commands. However, the use of the Cashier is preferred since it allows the potential for new cassettes to be introduced to the system without the need for rewriting any code. For more information on the Cashier, see the *WebSphere Commerce Payments Programming Guide and Reference*.

A Cashier profile represents a description of how WebSphere Commerce Payments orders should be created for a particular payment method. Profiles are XML documents that contain all the information needed by the Cashier to create WebSphere Commerce Payments API requests to create orders for a cassette supporting that payment method. All profiles must include the following data:

- An indication of whether a wallet is used. This flag will be used to determine whether the Cashier should use the AcceptPayment or ReceivePayment command.
- Required WebSphere Commerce Payments parameters.
- Required cassette parameters.
- Specifications for how the Cashier should supply values for each of the above parameters.

In addition, profiles may also contain the following optional data:

- An indication of which WebSphere Commerce Payments instance to use for each profile.
- Optional WebSphere Commerce Payments parameters.
- Optional cassette parameters.
- Buy page information that specifies how client code should build buy pages to collect buyer information. For example, the buy page information might contain an HTML form that collects credit card information required by a specific cassette.
- An indication of whether diagnostic information is to be enabled for the profile.

Cashier profiles allow parameter values to be specified in four different ways:

1. Hard-coded as constants in the profile.
2. Passed as an environment variable on the CollectPayment() call.
3. Specified as originating from a relational database field.
4. Specified as being calculated by Cashier extension code.

Sample KitCash cashier profile

A sample cashier profile, `SampleCheckoutKitCash.profile`, is provided with the Cassette for KitCash. This profile can be used by the SampleCheckout order entry system (sample application) provided with WebSphere Commerce Payments. For details on designing and tailoring Cashier profiles, see the Cashier chapter of the *WebSphere Commerce Payments Programming Guide and Reference*.

If you are installing and using the Cassette for KitCash in either the WebSphere Commerce Professional or Business Edition, or the WebSphere Commerce Studio environment, you must edit the `SampleCheckout.xml` file provided with the Payments component. Add the KitCash payment option as a SampleCheckout payment option before using the SampleCheckout application.

To add the KitCash payment option, follow these procedures:

- (In a WebSphere Commerce environment) Edit the following XML file:

  ```
  WAS_installdir/installedApps/host_name/payments_instance_Commerce_Payments_App.ear/
      SampleCheckout.war/SampleCheckout.xml
  ```

  400 For iSeries, the path is:

  ```
  WAS_userdir/node_name/payments_instance_Commerce_Payments_App.ear/
     SampleCheckout.war/SampleCheckout.xml
  ```

- Find the PaymentOptionList element and add the following to the list:

  ```
  <PaymentOption id="KitCash" profile="SampleCheckoutKitCash">KitCash</PaymentOption>
  ```

- To use the Sample Checkout application, point your browser to **http://<***host_name:port***>/webapp/SampleCheckout**.

Chapter 5. Getting Started

In this chapter, you'll configure the Cassette for KitCash and execute transactions through the test harness by purchasing items from the test merchant. At this point, you should have completed the following:

- Created a Payments instance
- Installed the Cassette for KitCash
- Configured the Cassette for KitCash using Configuration Manager (added the cassette to the Payments instance)
- Started the Payments instance (refer to the *WebSphere Commerce Installation Guide* for instructions)
- Defined a WebSphere Commerce Payments user with Merchant Administrator authority. For more information on performing this task, refer to the tutorial in the *WebSphere Commerce Administration Guide*.

Starting the KitCash sample test harness

After you have installed and configured the Cassette for KitCash, you can see the cassette interacting with WebSphere Commerce Payments, your Web server, the test wallet, and KitCash bank. To do this, you need to run WebSphere Commerce Payments and sample KitCash bank programs as follows:

1. Start the WebSphere Commerce Payments instance if it is not already started.
2. Start the sample KitCash bank program. From the `WC_installdir/payments/cassettes/KitCash/bin` directory, enter the following at a command prompt:
   - AIX Solaris Linux `./KitCashBank.sh`
   - Windows `KitCashBank.bat`
   - 400 Install and run the sample KitCash bank program on your workstation (Windows, AIX, etc.). Uncompress the KitCash zip file to a directory on your workstation and start the KitCash bank program at a command prompt from the KitCash/bin directory, as directed above.

The sample bank program will open a Java console window and show a startup value of five thousand KitCash dollars. The bank will listen for deposit requests from the Cassette for KitCash.

Configuring WebSphere Commerce Payments for the test merchant

Before your test merchant can receive and process orders, you must configure the merchant in WebSphere Commerce Payments. It is important for the KitCash test harness that Web pages are generated every time they are displayed so that you can be sure that you are seeing up-to-date data. To do this, you should turn off the caching support of your browser while you are using the test harness.

Then, in WebSphere Commerce Payments, do the following:

- Add a new merchant:
  - Merchant Name: Intangible Incorporated
  - Merchant Number: 123

  - KitCash Cassette
- Add an account for the new merchant:
  - Account Name: Complements department
  - Account Number: 457
  - Financial Institution Name: ACME Bank
  - hostname: *your_server_hostname*
  - Bank port: 47820
- Add a second account.
  - Account Name: Inspirations department
  - Account Number: 456
  - Financial Institution Name: ACME Bank
  - hostname: *your_server_hostname*
  - Bank port: 47820

400 *your_server_hostname* should be the host name of the workstation where you are running the sample KitCash bank program.

Open for business

To make purchases, use the SampleCheckout sample application program to enter an order. The Sample Checkout tool provides a user interface you can use to create sample orders to test your cassette implementation. (The *WebSphere Commerce Payments Programming Guide and Reference* provides more information about SampleCheckout.)

To access the WebSphere Commerce Payments Sample Checkout and create orders, do the following:

1. Open the `SampleCheckout.xml` file in the following directory:
   ```
   WAS_installdir/installedApps/host_name/payments_instance_Commerce_Payments_App.ear/
   SampleCheckout.war
   ```
   400 For iSeries, the directory path is
   ```
   WAS_userdir/installedApps/node_name/payments_instance_Commerce_Payments_App.ear/
   SampleCheckout.war
   ```
2. At the `SampleCheckout` element, change the following attribute values:
   ```
   pmHostName="fully_qualified_host_name"
   pmPort="port"
   default userid="wc_userid"
   password="wc_password"
   ```
   For `pmHostName`, enter the fully qualified host name for the WebSphere Commerce Payments Web server. For `pmPort`, enter the port number WebSphere Commerce Payments is running on as shown in the Configuration Manager WebServer information for your Payments instance. For the userid and password, enter the user ID and password associated with the WebSphere Commerce user.
3. Save the file.
4. Point your browser to http://*host_name:port*/webapp/SampleCheckout/, where *host_name* is the host name of the machine running the Web Server for Payments, and *port* refers to the port number Payments is running on as shown in the Configuration Manager WebServer information for your Payments instance.

5. At the Sample Checkout page enter the following:

Table 1. Sample Checkout fields for Cassette for KitCash

Field	Description
Merchant number	Enter 123, the number used when creating the merchant, to represent the merchant number. (Required)
Order number	Enter any unique number to represent an Order number. (Required)
Amount	Enter 5 to represent the total numeric amount of the order. (Required)
Currency	Select US dollar. The currency used to place this order. (Required)
Payment method	Select KitCash as the payment method. (Required)

6. Click **Buy**.

Merchant Settlement

- Point your browser at the following URL:
  http://*your_server:port*/webapp/PaymentManager
- Select the **Settle** link on the Navigation pane. Assuming you only made the one purchase, you should see a single batch to settle.
- Select the **Batch Number** link to view batch information.
- Select the **Settle** button to officially transfer funds from the consumer's bank account to the merchant's bank account. If everything is working correctly, the settlement page should display `The Batch was successfully settled` message. The KitCash bank screen should also display the conversation between the Cassette for KitCash and itself.

Chapter 6. Cassette for KitCash Design

As mentioned in the introduction, the Cassette for KitCash was created to illustrate the following features:

- An example of a payment protocol that is not oriented towards credit cards.
- How to support the ReceivePayment command, including:
  - The definition of protocol messages representing a set of associated protocol flows from a buyer's wallet.
  - The use of ComPoints to manage the protocol messages.
- How to use CassetteWorkItems and the Framework's service thread queue.
- Use of the Finite State Machine.
- An alternative cassette design.

This chapter is intended to provide enough detail for you to understand high level flows and the source that implements the key features. All of the source is located in `KitCash/Payments-KitCashCassette/java/com/ibm/etill/kitcashcassette`.

WebSphere Commerce Payments provides a finite state machine (FSM) editor and code generator you can use to help design your cassette. You should start by reviewing the Cassette for KitCash finite state machine (FSM) section that follows to understand cassette input and transition states. A detailed description of the FSM editor and code generator is provided in Chapter 7, "Tools", on page 31.

The Framework's FSM Editor should be used to identify actions that are called in KitCashPurchase. Once you understand the high level states and actions, you should review the "Cassette sequence diagrams" on page 25 to understand object interaction. The source code and Javadoc can be referenced for lower level details.

Finite state machine

The KitCash finite state machine (FSM), which is contained in *`WC_installdir`*`/payments/cassettes/KitCash/kitcashfsm.prj`, provides a model with a set of inputs that define what action the Cassette for KitCash should take for its current state and what the next state should be. The table below shows the six KitCash states and the corresponding inputs, actions, and next states.

Table 2. KitCash Finite State Machine

Project	
Name	KitCash
ProjectFileName	*`WC_installdir`*`/payments/cassettes/KitCash/kitcashfsm.prj`
ProjectDescription	Cash based protocol for Cassette Kit V2.2
PackageName	com.ibm.etill.kitcashcassette
Imports	com.ibm.etill.framework.supervisor.FSM com.ibm.etill.framework.payapi.ETillAbortOperation com.ibm.etill.framework.payapi.PaymentAPIConstants com.ibm.etill.kitcashcassette.test.card.KitCashConstants

Matrix	
Name	KitCash
MatrixDescription	Purchase flow for KitCash

Table 3. KitCash finite state machine. When viewing the KitCash finite state machine in kitcashfsm.prj, a Condition column appears between Input and Start. Because no data appears in that column, it was deleted here to facilitate printing.

Input	Start	Payment Requested	Payment Pending	Payment Received	Payment Added To Batch	Payment Complete	Error State
Receive Payment	Send Initiation Msg Payment Requested						
KitCash Msg		Start Payment Payment Pending	Continue Payment Payment Pending				
End Of Consumer Flow			Mark Payment As Received Payment Received				
Deposit				Mark PaymentFor Deposit Payment Added To Batch			
Batch Closed					Mark Payment As Complete Payment Complete		
Order Closed						CloseOrder Payment Complete	
Error	Report Error Error State	Report Error Error State	Report Error Error State	Report Error Error State	Report Error Error State	Report Error Error State	Report Error Error State

Javadoc

Javadoc for KitCash is provided in `WC_installdir/payments/cassettes/KitCash/javadoc`.

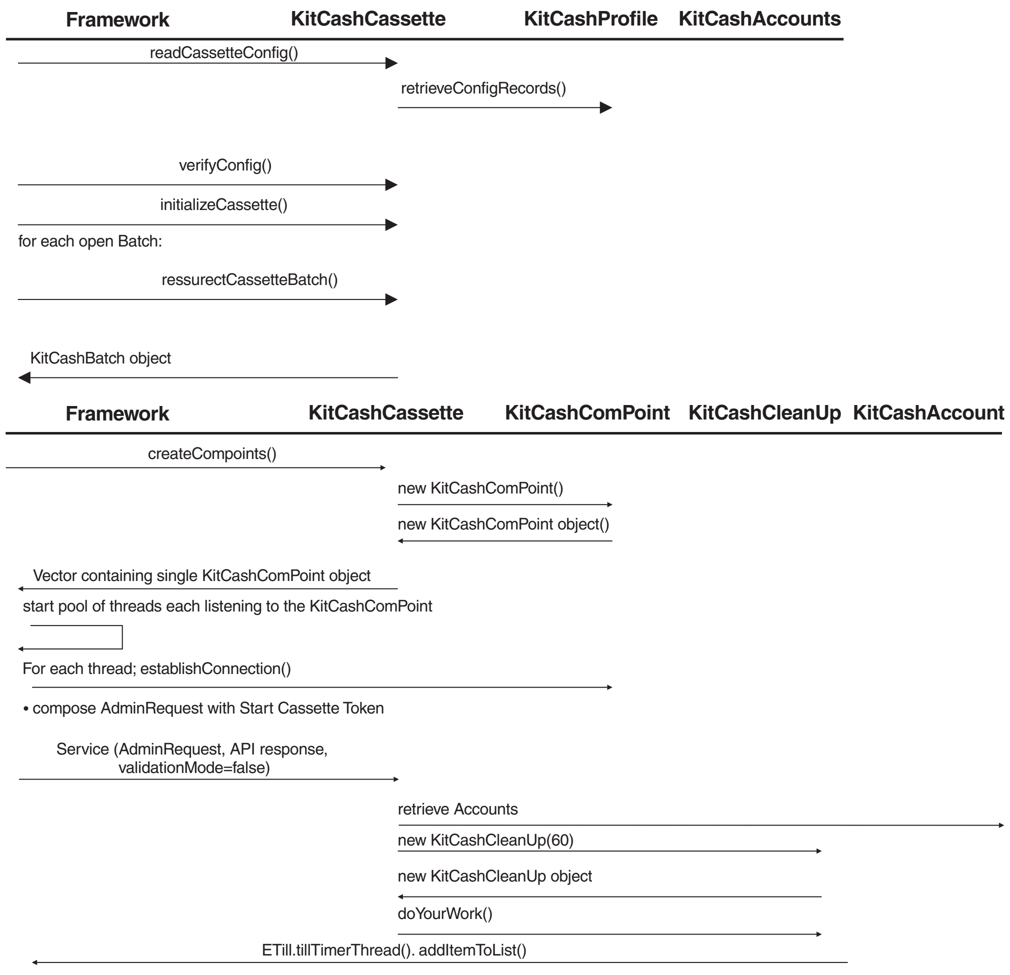
Cassette sequence diagrams

WebSphere Commerce Payments responds to events from the outside world. Scenarios describe the processing that occurs as a result of the receipt of a particular event. Sequence diagrams visually describe the sequence of interactions between the major players for a particular scenario. The sequence diagrams here describe success scenarios for the sample Cassette for KitCash.

The details in the sequence diagrams are not intended to be absolutely precise. For instance, method calls will use the real name of the Java method, but will not precisely define the parameters to that method. The diagrams are intended to give a logical idea of the responsibilities of the framework and the Cassette for KitCash for each scenario.

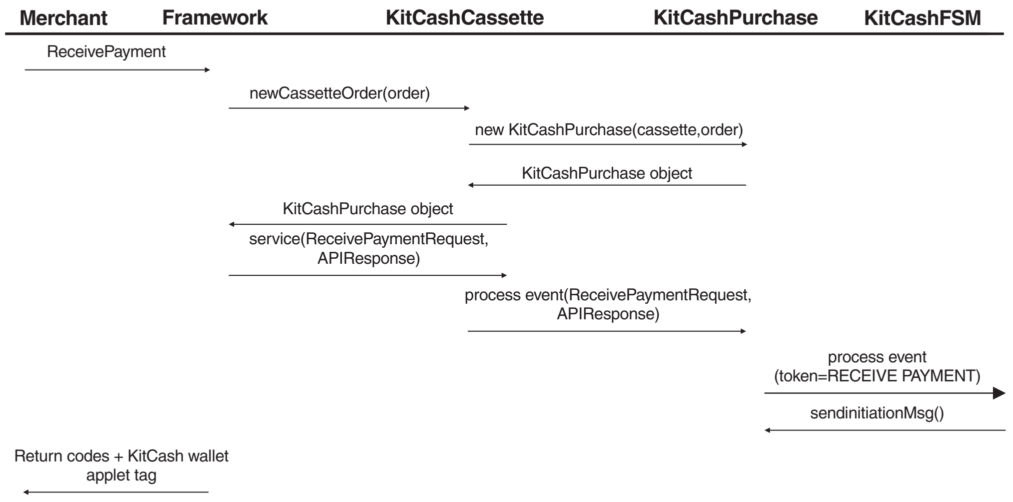
Startup API sequence

The KitCash startup API sequence diagram shows the interactions that occur when the Cassette for KitCash is started.



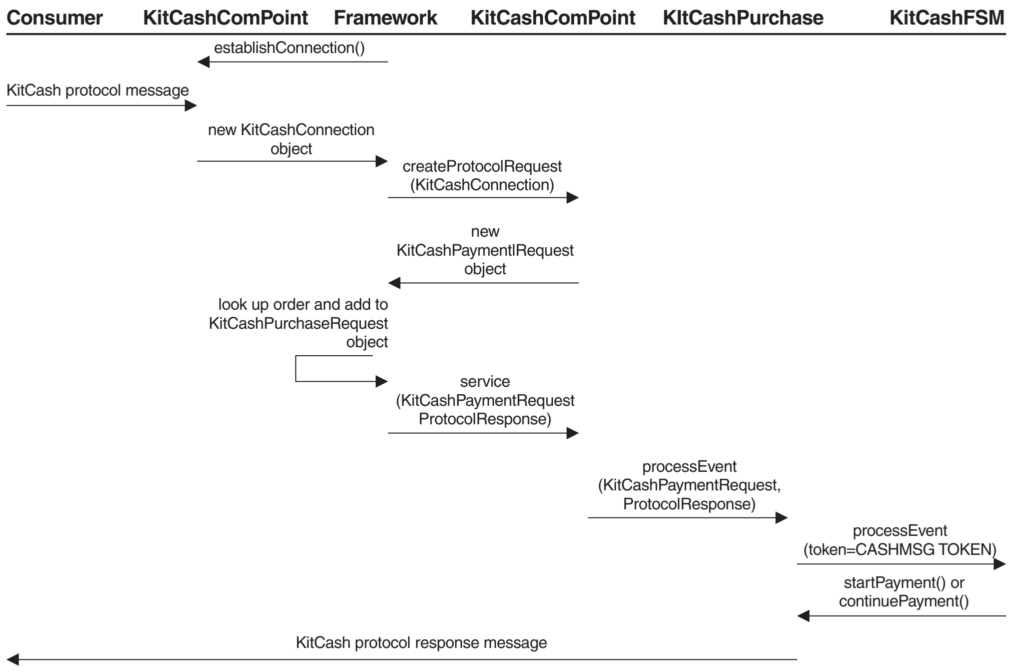
ReceivePayment API sequence

The ReceivePayment sequence diagram shows the interactions between the WebSphere Commerce Payments framework and the Cassette for KitCash when a `ReceivePayment` API command is sent to WebSphere Commerce Payments.


Merchant
Framework
KitCashCassette
KitCashPurchase
KitCashFSM
ReceivePayment
newCassetteOrder(order)
new KitCashPurchase(cassette,order)
KitCashPurchase object
KitCashPurchase object
service(ReceivePaymentRequest, APIResponse)
process event(ReceivePaymentRequest, APIResponse)
process event (token=RECEIVE PAYMENT)
sendinitiationMsg()
Return codes + KitCash wallet applet tag

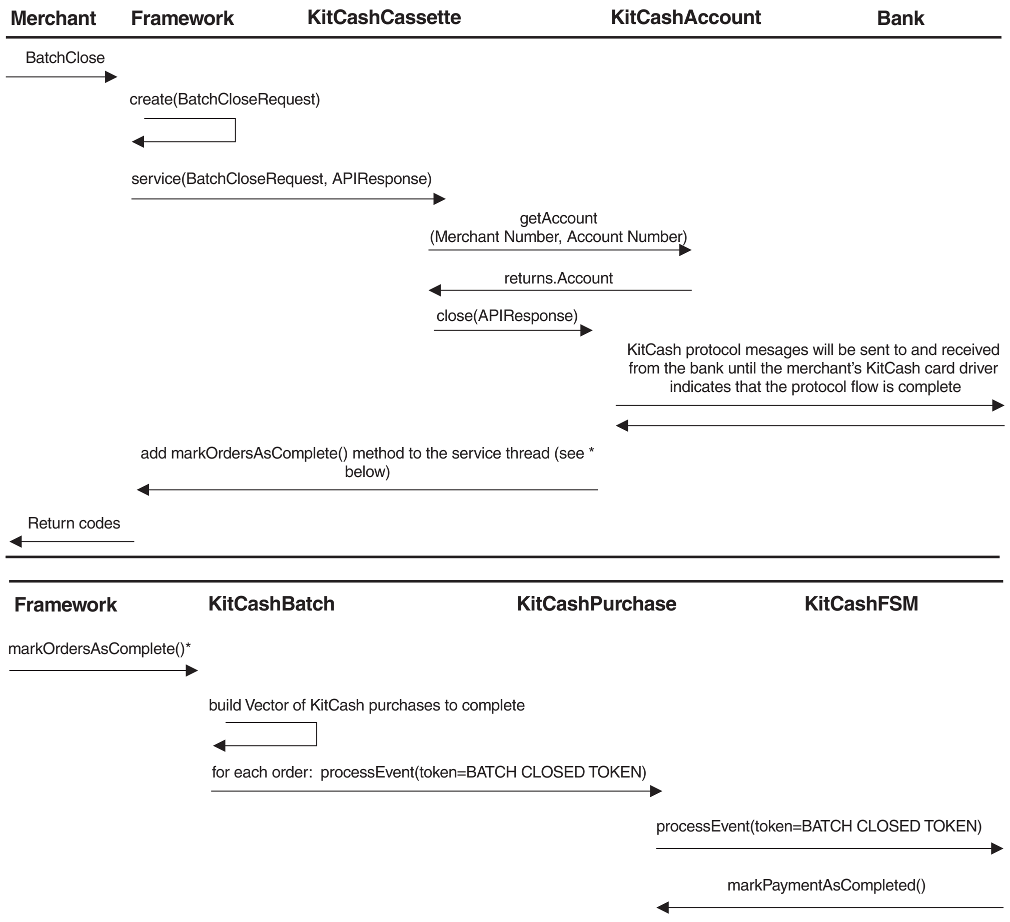
Protocol message API sequence

This sequence diagram shows the interaction between the WebSphere Commerce Payments framework and the Cassette for KitCash when a protocol message specific to the cassette is received from the outside world.


Consumer
KitCashComPoint
Framework
KitCashComPoint
KItCashPurchase
KitCashFSM
establishConnection()
KitCash protocol message
new KitCashConnection object
createProtocolRequest (KitCashConnection)
new KitCashPaymentIRequest object
look up order and add to KitCashPurchaseRequest object
service (KitCashPaymentRequest ProtocolResponse)
processEvent (KitCashPaymentRequest, ProtocolResponse)
processEvent (token=CASHMSG TOKEN)
startPayment() or continuePayment()
KitCash protocol response message

BatchClose API sequence

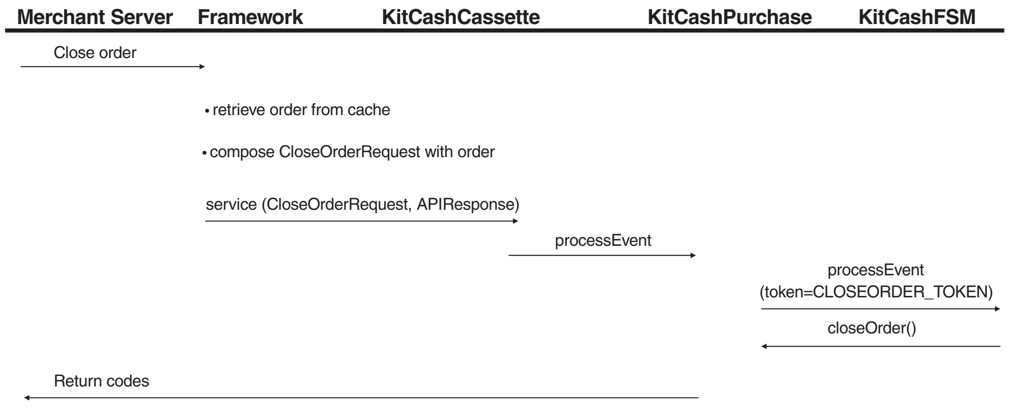
The BatchClose sequence diagram shows the interactions between the WebSphere Commerce Payments framework and the Cassette for KitCash when a `BatchClose` command is sent.


Merchant
Framework
KitCashCassette
KitCashAccount
Bank
BatchClose
create(BatchCloseRequest)
service(BatchCloseRequest, APIResponse)
getAccount (Merchant Number, Account Number)
returns.Account
close(APIResponse)
KitCash protocol mesages will be sent to and received from the bank until the merchant's KitCash card driver indicates that the protocol flow is complete
add markOrdersAsComplete() method to the service thread (see * below)
Return codes
Framework
KitCashBatch
KitCashPurchase
KitCashFSM
markOrdersAsComplete()*
build Vector of KitCash purchases to complete
for each order: processEvent(token=BATCH CLOSED TOKEN)
processEvent(token=BATCH CLOSED TOKEN)
markPaymentAsCompleted()

Note: If the number of orders in a batch exceeds fifty (50), a service thread (as shown above) is spawned or else ″markOrdersAsComplete″ is handled in the same thread.

CloseOrder API sequence

The CloseOrder API sequence diagram shows the interactions between the WebSphere Commerce Payments framework and the Cassette for KitCash when a `CloseOrder` API command is sent to WebSphere Commerce Payments.


Merchant Server
Framework
KitCashCassette
KitCashPurchase
KitCashFSM
Close order
• retrieve order from cache
• compose CloseOrderRequest with order
service (CloseOrderRequest, APIResponse)
processEvent
processEvent (token=CLOSEORDER_TOKEN)
closeOrder()
Return codes

Chapter 7. Tools

This section describes the FSM editor and code generator.

FSM editor and code generator

WebSphere Commerce Payments provides a finite state machine (FSM) editor and code generator you can use to help design your cassette.

Finite state machine overview

A finite state machine is a model of a system whereby a set of inputs defines what action the system should take for its current state and what the next state should be. It is recommended that you use finite state machines to control the processing performed on payment transactions for any given input from the merchant or elsewhere. There are several benefits of using finite state machines:

- Because it is not possible to guarantee what state a transaction will be in on receiving a given input, using a finite state machine forces you to be rigorous about defining actions for *all* inputs in *all* states (including exception conditions).
- Separating the state transaction logic from the payment method processing allows changes to be made to one without necessarily affecting the other.
- Debugging is much easier when the state transition logic and payment logic are separated.

Finite state machines can be represented by state transition diagrams, such as:

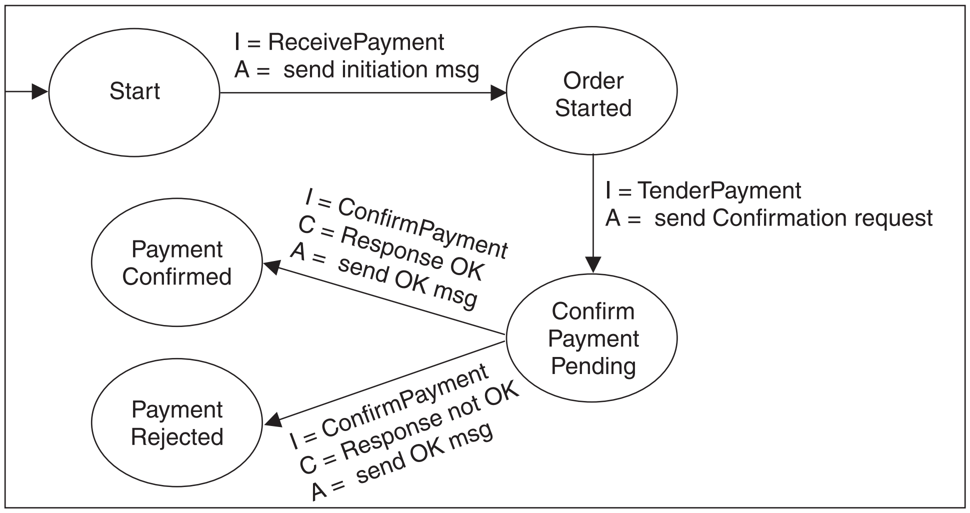


Figure 2. Finite state transitions

The nodes represent all possible states of the machine. The arrows indicate which inputs (I=) will trigger which actions (A=) and what the next state of the machine will be. You can also specify conditions (C=) to qualify each input.

An alternative representation of the finite state machine uses a matrix representation. In this table, each row indicates what effect the input specified in the first two columns has on each state. The top half of each cell indicates the action (A) to be taken; the bottom half indicates the next state (NS) of the machine.

Empty cells indicate a ″should not occur″ state. Error processing should be performed.

Table 4. Finite state machine matrix

Input	**Condition**	**Start**	**OrderStarted**	**ConfirmPayment Pending**
Receive Payment		Action: send init message Next State: OrderStarted		
Tender Payment			Action: send confirm request Next State: ConfirmPayment Pending	
Receive Response	**Response OK:** *true*			Action: send OK message Next State: PaymentConfirmed
Confirm Payment	**Response OK:** *false*			Action: send reject message Next State: PaymentRejected

FSD Data Model

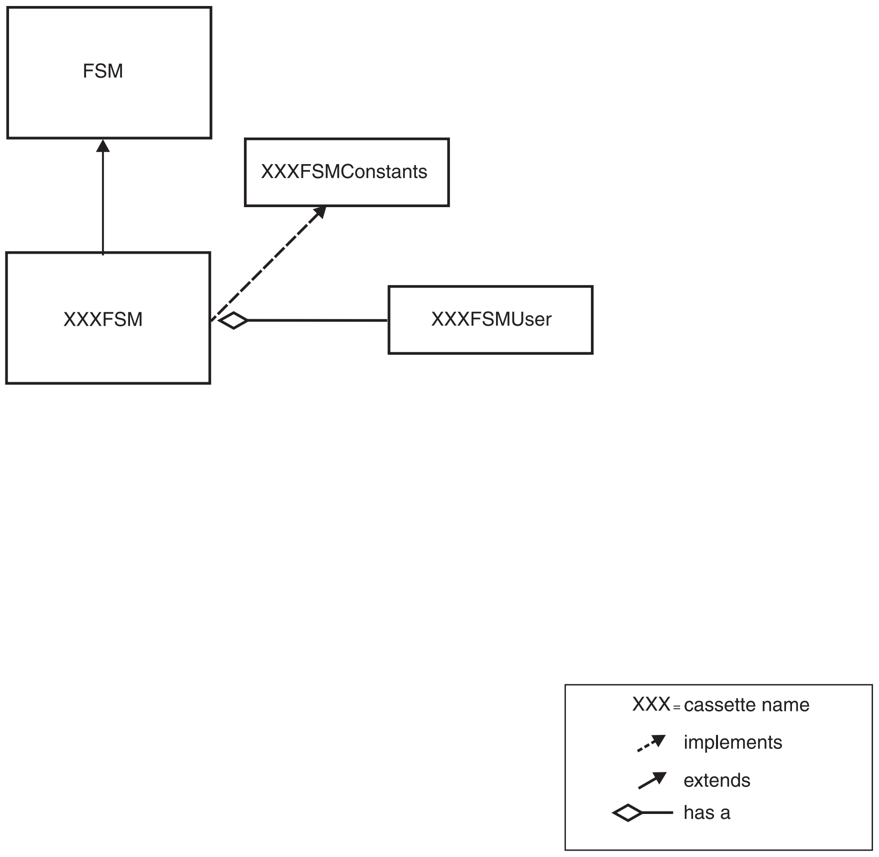


Figure 3. FSD data model

Finite state machine editor

The Cassette Kit contains an editor that allows you to create finite state machines for your cassette. You can start the editor using this command:

```
c:\CassetteKit\tools> FSMEdit [project_name]
```

where `project_name` is the name of an existing `.prj` file.

Creating a new finite state machine

To create a new finite state machine project:

1. Select **New** from the File menu.
2. Select **Project** from the Properties menu to enter details of the new finite state machine:

Field	Description
Name	the name of the finite state machine. The names of the generated Java class files will be based on this name, so it should contain no white space characters and follow Java naming conventions (for example, `Test`)
Description	a description for the finite state machine
Filename	the name of the file that will hold the finite state machine (for example, `c:\eTill\testfsm.prj`)
Import	the package name of your cassette (for example, `com.acme.ibmetill.acmecardcassette`).

3. You should also set the name of the matrix to be the same as the project name above by selecting Matrix from the Properties menu. Currently only one matrix per project file is supported.

Creating HTML from your FSM matrix

To document your FSM matrix, you can create an HTML table from your finite state machine project:

1. Select **Save Project as HTML** from the File menu.

Adding inputs, conditions, actions and states

From the Windows menu, select from Inputs, Conditions, Actions and States to add, edit or delete the inputs, conditions, actions and states of the finite state machine. As an example, for the above state machine, the following values would be set:

Inputs	Conditions	Actions	States
ReceivePayment	ResponseOK	SendInitiationMsg	Start
TenderPayment		SendConfirmRqst	OrderStarted
ConfirmPayment		SendOKMsg	ConfirmPaymentPending
		SendRejectMsg	PaymentConfirmed
			PaymentRejected

For each setting, you can also provide a description and a code fragment. The code has different meanings depending on which of the inputs, conditions, actions and states it is associated with.

For inputs, the code specifies a Java constant uniquely identifying the token for the input. For example, `PaymentAPIConstants.RECEIVEPAYMENT_TOKEN` for ReceivePayment and `ACMECardCassette.TenderPayment_Token` for TenderPayment.

The code fragment is not applicable for states.

For conditions, the code fragment specifies the Java method name that will be invoked to determine whether the condition is true or false. This method must be defined by the Java class that implements the FSMUser interface, for example: `responseOK` for ResponseOK.

For actions, the code fragment specifies the Java method name that will be invoked to perform the action. This method must be defined by the Java class that implements the FSMUser interface. For example `sendOKMsg` for SendOKMsg.

Building the finite state machine matrix

The editor displays the matrix representation of the finite state machine. To add rows and columns to this matrix, select *Add InputRow* and *Add StateColumn* from the *Matrix* menu, and select the inputs and states that you have defined.

Setting inputs and input conditions

When you added a new row to the matrix, you were asked to choose an input name. If you need to change the input name, choose a different input from the list available in the input name field.

There can be multiple matrix rows with the same input name. Input conditions are used to further qualify an input. An input condition is usually a condition that cannot be checked until a new input has been received. For instance, when the input represents an Approve API command, the amount of the request may affect the action to be performed. The amount would be checked as an input condition.

Note that it is perfectly legitimate to check multiple conditions for a single input. But the FSMEdit tool only allows a single input condition to be specified per row. If you need to use multiple inputs conditions per row, create a single method that checks all the conditions for the input row. Use this method as the condition.

Setting actions and next states

Having defined the matrix rows and columns, you can specify the actions and next states by clicking one of the buttons in each and selecting the action and next state that you want for the given input and current state.

Saving the finite state machine

To save the finite state machine you have built, select **Save** from the File menu. This will create a `.prj` file that contains the definitions of all inputs, conditions, actions and states you have defined, as well as a representation of the finite state machine matrix itself.

You can also build an HTML representation of the matrix by selecting **Save HTML** from the File menu.

Generating the Java source files

To generate Java source code from the finite state machine `.prj` file, use this command:

```
c:\CassetteKit\tools> FSMGen [project_name]
```

where *`project_name`* is the `.prj` file name of your finite state machine.

Assuming the names of the project and matrix are both Test, the following three files will be created:

File	Description
`TestFSMConstants.java`	a Java interface containing constant definitions for the states and inputs used in the finite state machine.
`TestFSM.java`	a Java class containing the controlling logic for the finite state machine.
`TestFSMUser.java`	a Java interface containing signatures for the action methods and conditions methods that must be implemented by a cassette using this finite state machine.

Using the finite state machine

Having created the Java finite state machine code, you can then use this in a class in your cassette by implementing the FSMConstants and FSMUser interfaces you have created, and by including an FSM object as a member variable of the class:

```
public class TestOrder implements CassetteOrder, TestFSMConstants, TestFSMUser {
  private TestFSM fsm;
  ...
```

Implementing the TestFSMConstants interface allows you to access the constant values for the inputs and states of the finite state machine. Implementing the TestFSMUser interface ensures that your class includes all the methods to handle the actions and conditions specified in the finite state machine.

The FSM object must be initialized in your constructor as follows:

```
fsm = new TestFSM(this, Start);
```

where Start is the state in which the finite state machine should begin.

To use the finite state machine you have created, you can write code as follows:

```
  while (token != null) {
    token = fsm.processEvent(token.intValue());
  }
```

This code assumes that token is an Integer identifying a valid input accepted by the finite state machine. The processEvent() method determines what the next state should be and which action is dictated by the finite state machine. It invokes the action by calling the transaction’s performAction() method passing the relevant action constant as its parameters.

The performAction() method can be coded in your class as follows:

```
public Integer performAction(int action) throws ETillAbortOperation {
  Integer token = null;
  switch (action) {
    case SendInitiationMsg:
      token = sendInitiationMsg();
      break;
    case SendOKMsg:
      token = sendOKMsg();
      break;
    case SendRejectMsg:
      token = sendRejectMsg();
      break;
    case SendConfirmRqst:
      token = sendConfirmRqst();
      break;
    default:
  }
  return token;
}
```

Note that for future versions of WebSphere Commerce Payments, this method may be generated automatically rather than needing to be written by hand as at present.

Appendix. Notices

Trademarks

Index

Printed in U.S.A.