**Command Types**

WebSphere Commerce commands are Java beans that contain the programming logic associated with handling a particular request. Commands perform a specific business process, such as adding a product to the shopping cart, processing an order, updating a customer's address book, or displaying a specific product page.

Depending on its nature, a command can:

* Call enterprise beans to perform database operations.
* Call one or more task commands that are assigned to process tasks to process and write information to the database.
* Return a view task on completion of a controller command.

The WebSphere Commerce programming model defines the following command types:

**Controller commands**

Encapsulate the logic related to a particular business process. Examples of controller commands include the OrderProcessCmd command for order processing and the LogonCmd that allows users to log on. In general, a controller command contains the control statements (for example, if, then, else) and invokes task commands to perform individual tasks in the business process. Upon completion, a controller command returns a view name. Based upon the view name, the store identifier, and the device type, the solution controller determines the appropriate implementation class for the view and then invokes it.

**Task commands**

Implement a specific unit of application logic. In general, a controller command and a set of task commands together implement the application logic for a URL request. Task commands are executed in the same container as the controller command.

**Data bean commands**

Are invoked by the data bean manager when a JSP page needs to instantiate a data bean. The primary function of a data bean command is to populate the fields of a data bean with data from a persistent object.

**View commands**

View commands, used to compose a view as a response to a client request, are deprecated in this release of WebSphere Commerce. Since WebSphere Commerce is a Struts application the view command has been replaced by global forwards. For compatibility with previous releases, view command of previous releases will continue to work.

When creating new business logic for your e-commerce application, it is expected that you might need to create new controller and task commands.

The following table shows which implementation class a new command should extend and which interface it should implement:

| **Command type** | **Example command name** | **Extends** | **Implements example interface** |
| --- | --- | --- | --- |
| Controller command | MyControllerCmdImpl | com.ibm.commerce.command.ControllerCommandImpl | MyControllerCmd |
| Task command | MyTaskCmdImpl | com.ibm.commerce.command.TaskCommandImpl | MyTaskCmd |
| Data bean command | MyDataBeanCmdImpl | com.ibm.commerce.command.DataBeanCommandImpl | MyDataBean |

## Default commands and views

WebSphere Commerce provides default commands and views which you can use in your store. These default commands and views are listed in the Struts configuration file for the Web application.

Additionally, many of the views used in starter store were created specifically for the store. These views are listed in the struts-config-update.tpl.xml file packaged as part of the store archives. If a required command or view is not provided, you can add your own to the store archive's Struts configuration file.

**Command context**

Commands can obtain information using the command context. Examples of information available include the user's ID, the user object, the language identifier, and the store identifier.

When writing a command, you have access to the command context by calling the getCommandContext() method of the command's superclass. The command context is set to the controller command when the command is invoked by the component facade. A controller command should propagate the command context to any task or controller commands that are invoked during processing.

In previous releases, the information described in the following list was stored in the Command Context object. With the introduction of Business Context in this release, this information are now stored in various business contexts. Command Context becomes a helper class that wraps on top of these business contexts. One can directly retrieve the same piece of information by retrieving the appropriate business context by using the CommandContext.getContext(businessContextName) method. Information that is not available from business contexts, remains available and local to the Command Context object. A command can get the following key information from the command context:

**getUserId() and getUser()**

Gets the user ID or user object the current request should run against. The userId for the current session is stored in the BaseContext. Every BaseContext is associated with an activity identifier which is stored in the session. This session can be persisted in one of two ways: using WebSphere Commerce cookie or a WebSphere Application Server persistent session object. The command context hides the complexity of session management from a command. Instead of retrieving the user ID from the command context, it can also be retrieved from the following code:

BaseContext baseContext = (BaseContext)getCommandContext().getContext(BaseContext.CONTEXT\_NAME);

baseContext.getRunAsId();

**getStoreId(), getStore()**

Gets the store associated with the current request. For Web requests, this method will return the store ID in the URL. If the store ID is not specified in the URL, it can be retrieved from the BaseContext of the activity saved in the session of previous requests. For requests initiated from other channels such as the Sales Center, WebSphere Commerce runtime will first try to retrieve the store ID from the activity data the client passed in with the request. If none is found in the activity data, the store ID stored in the BaseContext of the activity will be used. An alternate way to retrieve the store ID is as follows:

BaseContext baseContext = (BaseContext) getCommandContext().getContext(BaseContext.CONTEXT\_NAME);

baseContext.getStoreId();

The command context also provides a convenience method, getStore(), that will return you the store object of the current store.

**getLanguageId()**

Returns the language ID that should be used for the current request. If the language ID is specified in the URL for a Web request or in the activity data for a Sales Center request, WebSphere Commerce runtime first validates whether this language ID is supported by the store. If it is, this language ID will be returned by this method. If it is not supported, it will then use the language ID stored in the GlobalizationContext of the activity in the session. However, if this is the first request and hence no language ID is available in the GlobalizationContext, of the activity and reuse for subsequent requests.

* The user's preferred language ID is first validated and checked whether it is supported by the store. If it is true, this method will return this language ID.
* If it is not supported, this method will return the store default language.

An alternate way to retrieve the language ID is from the GlobalizationContext:

GlobalizationContext globalizationContext = (GlobalizationContext)getCommandContext().getContext(GlobalizationContext.CONTEXT\_NAME);

globalizationContext.getLanguageId();

**getCurrency()**

Returns the currency to be used for the current request. The currency is stored in the GlobalizationContext and is derived from the languageId as part of the Globalization framework. Hence, the logic behind this method is similar to that of the getLanguageId() method. The SetCurrencyPreference command should be used to modify the currency in the session.

Another way to retrieve the currency to be used by the request is from the GlobalizationContext:

(GlobalizationContext)getCommandContext().getContext(GlobalizationContext.CONTEXT\_NAME);

globalizationContext.getCurrency();

Since currency is part of the Globalization Framework, logic behind this method is similar to that of the getLanguageId() method.

**getCurrentTradingAgreements()**

**getCurrentTradingAgreements(CommandContext)**

Returns the set of trading agreements that are used for the current session. This set may be all of the trading agreements to which the user is entitled, or it can be a subset that was defined by the ContractSetInSession command. A command should always get the trading agreement object from the command context to take advantage of the object cache. You can get the current trading agreement by calling the getCurrentTradingAgreements(CommandContext) method.

The command context should be used as a read-only object. You should not call its setter methods.