IBM Software Group

# WebSphere® Commerce V6

## *Developing and customizing store business logic*

business on demand.

© 2008 IBM Corporation
Updated June 11, 2008

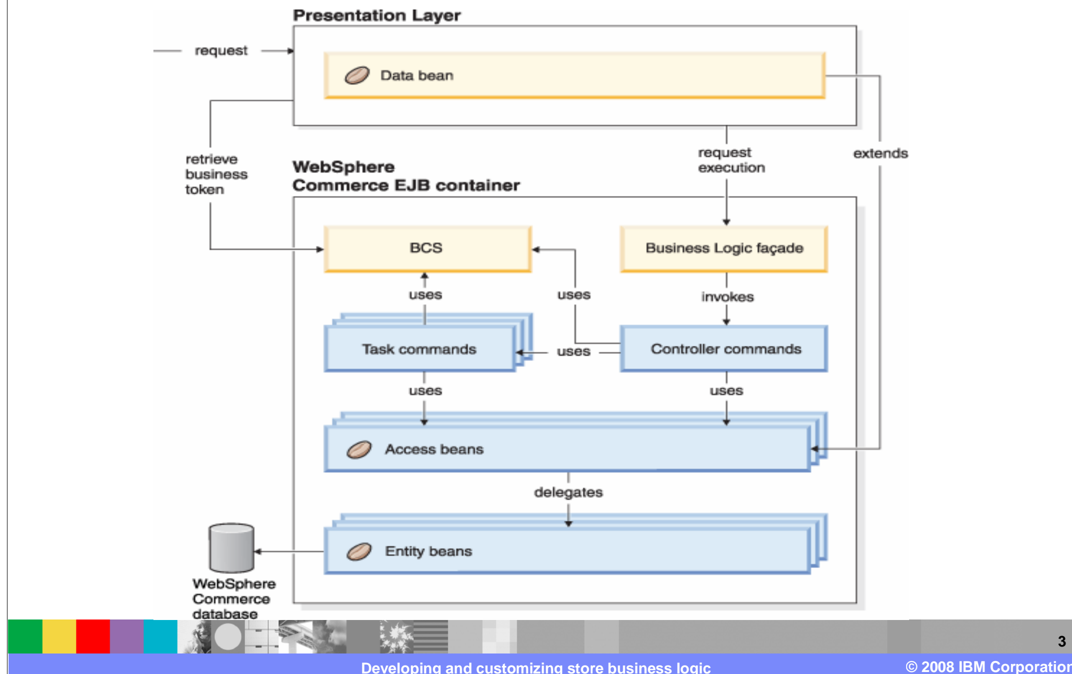
Welcome to the WebSphere Commerce V6 presentation. This presentation describes methods and best practices for developing and customizing store business logic.

# Agenda

- Describe the WebSphere Commerce command framework and the command API
- Examine how to register commands
- Describe the command programming model
- Introduce customizable components in WebSphere Commerce

© 2008 IBM Corporation

This presentation introduces you to the WebSphere Commerce command framework. Command registration and the command programming model is discussed. The presentation  concludes with a discussion of customizable business components in WebSphere Commerce.




This diagram summarizes the WebSphere Commerce functional architecture. If you are not familiar with the WebSphere Commerce architecture you should view the presentation **Programming architecture** before continuing with this presentation. The components that make up the business logic layer will now be examined in more detail.

The business logic façade is a generic interface implemented as a stateless session bean. The Struts controller calls the business logic façade to invoke controller commands. A controller command performs business process logic such as OrderProcess. It invokes task commands to accomplish different units of work in the business process. By default, access control is enabled for controller commands. A task command is an autonomous task that accomplishes a specific unit of application logic such as check inventory. A task command typically works with other task commands to complete processing of a controller command. By default, access control is not enabled for task commands.

Business context service, labeled BCS in the diagram, is new in version 6. It is a service that manages contextual information used by business components. The contexts include such information as globalization and entitlement.

Access beans are simple persistent objects with setters and getters. The access bean behaves like a Java™ bean and hides all the enterprise bean specific programming interfaces, like JNDI, home and remote interfaces from the clients. Rational® Application Developer provides tool support to generate access beans from the schema. Entity beans are used in the persistence layer within WebSphere Commerce. The architecture is implemented according to the EJB component architecture. The EJB architecture defines two types of enterprise beans: entity beans and session beans.

Finally, the presentation layer displays the result of command execution. The presentation layer can use JSP pages, or other rendering technologies.

## WebSphere Commerce commands

- Controller commands
- Task commands
- Data bean commands
- View commands

© 2008 IBM Corporation

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.

Controller commands encapsulate the logic related to a particular business process. 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. Examples of controller commands include the OrderProcessCmd command for order processing and the LogonCmd that allows users to log on.

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 run in the same container as the controller command. Examples of task commands are DoPayment and DoLuhnCheck used in the OrderProcess controller command.

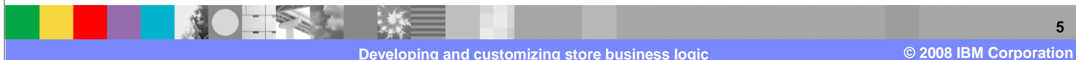
Data bean commands are invoked by the data bean manager when a JSP page needs to instantiate a data bean. The primary function of a data bean command is to populate the fields of a data bean with data from a persistent object.

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

IBM Software Group

## WebSphere Commerce command factory

- Design pattern
- Instantiates new command objects
- Defers instantiation away from the invoking class
- Factory class determines the correct implementation class
  - Two ways to declare implementation class of a command
    - defaultCommandClassName
    - Command registry
  - Command registry overrides defaultCommandClassName



5

Developing and customizing store business logic

© 2008 IBM Corporation

In order to create new command objects, the caller of the command uses the command factory. The command factory is a bean that is used to instantiate commands. It is based on the factory design pattern, which defers instantiation of an object away from the invoking class, to the factory class that understands which implementation class to instantiate.

The factory provides a smart way to instantiate new objects. In this case, the command factory provides a way to determine the correct implementation class when creating a new command object, based upon the individual store. The command interface name and the particular store identifier are passed into the new command object, upon instantiation.

There are two ways for the implementation class of a command to be specified. A default implementation class can be specified directly in the code for the command interface, using the defaultCommandClassName variable. The second way to specify the implementation class is to use WebSphere Commerce command registry. The command registry should always be used when the implementation class varies from one store to another. In the case where a default implementation class is specified in the code for the interface and a different implementation class is specified in the command registry, the command registry takes precedence.

# Nesting controller commands

- Pass the command context
- Pass the request properties
- Call the execute() method

```
YourControllerCmd ctrlCmd = null;
public void processAndCallOtherCommand()
  throws ECException
  {
    YourControllerCmd ctrlCmd = (YourControllerCmd)CommandFactory.createCommand(
      com.yourcompany.commands.yourControllerCmd, this.getStoreId());
    ctrlCmd.setCommandContext(this.getCommandContext());
    ctrlCmd.setRequestProperties(this.getRequestProperties());
    ctrlCmd.execute();
  }
```


© 2008 IBM Corporation

You will most often use the command factory to create instances of task commands. However, it can also be used within one controller command to create an instance of another controller command. The syntax for instantiating task commands and controller commands is the same. You specify the name of the command's interface and the store ID in both scenarios.

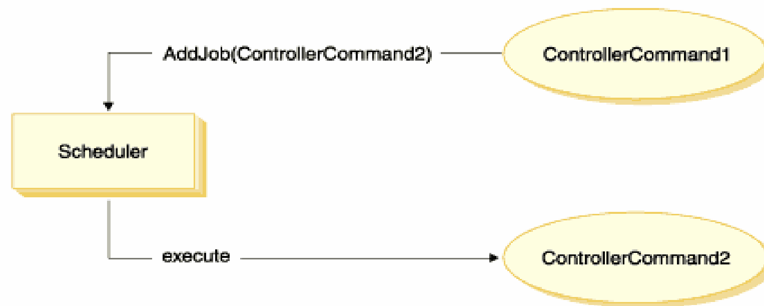
Once you have instantiated the nested command, call its setCommandContext method and pass in the current command context. Note that if you are passing a different set of request properties to the nested command and these parameters affect the command context, you should clone the command context before instantiating the nested command. This preserves the command context information for the outer command.

The preferred approach is to call the setRequestProperties method of the nested command and pass a TypedProperties object containing input properties. Otherwise, you can use the individual setter methods defined on the interface of the command to set the required properties.

After input properties have been set, call the **execute** method of the nested command. All controller commands must return a view when processing has completed. But in this case, the outer command does not need to do anything with the view returned by the nested command. The nested command is invoked within the transaction scope of the outer command.

The code snippet shows an example of a nested controller command. The example shows a method in the outer command and how it can use the command factory to instantiate a second controller command.

# Long-running controller commands





© 2008 IBM Corporation

If a controller command takes a long time to run, you can split the command into two commands. The first command, which is invoked as the result of a URL request, adds the second command to the Scheduler, so that it runs as a background job.

The flow is shown in the diagram. ControllerCommand1 is invoked as a result of a URL request. ControllerCommand1 adds a job to the Scheduler. The job is ControllerCommand2. ControllerCommand1 returns a view, immediately after adding the job to the Scheduler. The Scheduler invokes ControllerCommand2 as a background job.

In this scenario, the client typically polls the result from ControllerCommand2. ControllerCommand2 should write the job state to the database.

# WebSphere Commerce command registry

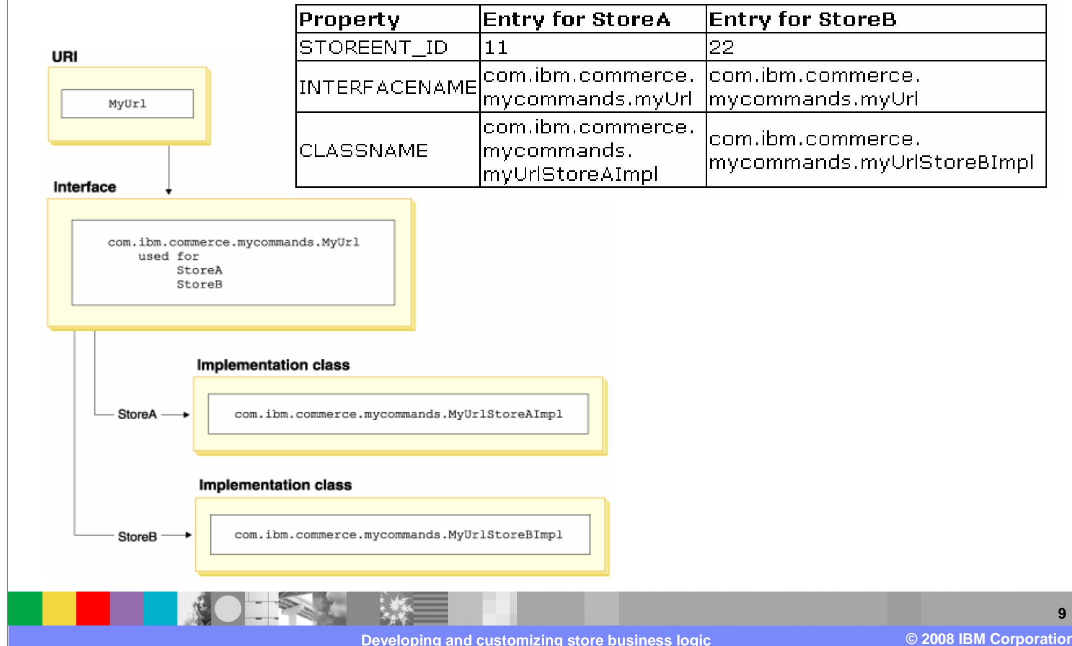
| Property | Description | Comments |
|---|---|---|
| STOREENT_ID | Store entity identifier | This can be set to 0 to use the command for all stores, or to a unique store identifier to indicate that the command is used only for a particular store. |
| INTERFACENAME | Command interface name | This defines the interface; use the same name as you did in the URL registry. |
| DESCRIPTION | Description of this command | For example, This command is used for testing purposes. |
| CLASSNAME | Command implementation class name | Typically the interface name with "Impl" appended to end. |
| PROPERTIES | Default name-value pairs set as input properties to the command | Format is same as URL query string. For example "parm1=val1&parm2=val2" |

© 2008 IBM Corporation

WebSphere Commerce controller and task commands are registered with the command registration framework. The command registry is implemented by means of the CMDREG database table. The command registry provides a mechanism for mapping the command interface to its implementation class. Multiple implementations of an interface allow for command customization on a per store basis. The table describes information contained in the command registry.

In general, when you create a new controller or task command, you should create a corresponding entry in the command registry. If the command you are writing always uses the same implementation class, you do not necessarily have to register the command in the command registry. In this case, you can use the defaultCommandClassName attribute in the interface to specify the implementation class. If you specify the implementation class in this manner, you cannot pass default properties to the implementation class and the same implementation class must be used for all stores.

# WebSphere Commerce command registry example



| Property | Entry for StoreA | Entry for StoreB |
|---|---|---|
| STOREENT_ID | 11 | 22 |
| INTERFACENAME | com.ibm.commerce. mycommands.myUrl | com.ibm.commerce. mycommands.myUrl |
| CLASSNAME | com.ibm.commerce. mycommands. myUrlStoreAImpl | com.ibm.commerce. mycommands.myUrlStoreBImpl |



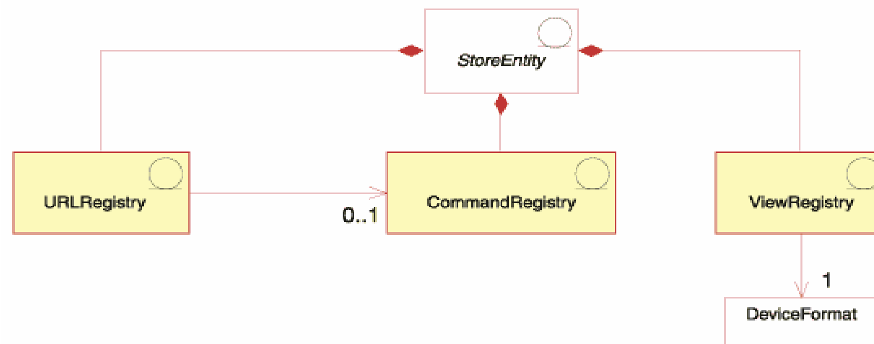
Consider a scenario in which your site has two stores: StoreA and StoreB. Each store has different implementations of the command. This slide shows how the command registry is used to enable this customization. Calling MyUrl will invoke the business logic that is associated with the com.ibm.commerce.mycommands.MyUrl interface. The interface is shared between the two stores. However the implementation for the business logic is controlled by the store specific configuration found in the Command Registry.

The diagram illustrates the flow of the solution controller. Using entries in the command registry, the solution controller determines the correct implementation class for the store being accessed.

IBM Software Group

IBM

# WebSphere Commerce command registry object model




10

Developing and customizing store business logic

© 2008 IBM Corporation

The command, view, and URL registries are part of the WebSphere Commerce command framework. The framework determines how a command runs and then returns a response based on the view returned by the command. The command execution and response is store dependent, which means that the same command can be implemented differently for each store and return different responses for each store. The diagram illustrates the command, view, and URL registry structure in the WebSphere Commerce Server.

The URL registry maps a command name to the actual interface of the command to be run. Each URL registry entry is store sensitive. Each store can define a different interface for the same URL value. If the store version of the URL registry cannot be found, then the URL registry defined for the site (store 0) is used. By default, all URL registries are defined for the site. URL registry mappings are stored in a Struts configuration file as action mappings.

Every command, whether it is a controller or task command, can be defined in the command registry. If a command is defined in the command registry, that definition is used as the command implementation when the command is invoked. If the command is not defined in the command registry, a default implementation is used instead.

After a command is run, in most cases, the requester of the command requires a response to be returned. Every view that returns a response must be defined in the view registry, either per store, or by default, by site. Each store normally defines the view for each possible device format of the incoming request. However, if a view is not defined by a store, the default view for the site is used. The adapter handling the request decides which device format to use when determining which view to call. There is no generic device format. Depending on the different types of requests that can be accepted, there might be a view defined for each device format. The view registry is stored as forwards in the Struts configuration file.

IBM Software Group

# WebSphere Commerce command context

- A handle to the solution controller
- Wraps the new business context objects
- Can be modified for specific command execution
- A container of common environment objects
  - User ID
  - User object (UserAccessBean)
  - Language identifier
  - Store Identifier
  - More …

11

Developing and customizing store business logic

© 2008 IBM Corporation

The Command Context is another key piece of the WebSphere Commerce command framework.

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 superclass. The command context is sent to the controller command when the command is invoked by the component façade. A controller command should propagate the command context to any task or controller commands that are invoked during processing.

In previous releases, the information was stored in the Command Context object. With the introduction of the Business Context Service in this release, this information is now stored in various business contexts. Command Context becomes a helper class that wraps on top of these business contexts. You can directly retrieve the same piece of information by retrieving the appropriate business context. Information that is not available from business contexts, remains available and local to the Command Context object.

IBM

# Business context services

- Business contexts
- Benefits
  - Enablement of generic components
  - Tailored content and experience
  - Precisely targeted offers
  - Enforcement of business policies
  - Appropriate prices, entitlements, and terms for a particular user
- Business context service components

© 2008 IBM Corporation

In older versions of WebSphere Commerce, runtime infrastructure is designed to serve customers from the Web channel. As WebSphere Commerce introduces support for different client types, a new infrastructure is needed to provide contextual information to the business logic independent of channel. Business context services track user session information from different channels by using different types of business contexts. A complete list of business contexts is provided in the next slide. Examples include the BaseContext and EntitlementContext. The first contains the basic attributes that an request needs, such as store ID, caller ID, and the run-as ID. The second holds information about entitlement criteria, such as reduced prices for gold club membership. You can also define custom context information to extend WebSphere Commerce predefined session attributes.

Abstracting user contextual information offers several benefits. Components can be created in a generic fashion with specific actions being triggered by the context information. Content and shopping experience can be tailored to the individual resulting in precisely targeted offers in addition to appropriate prices, entitlements and terms. Business policies can be easily enforced.

Multiple business context services can be logically grouped into business components. A component specializes in a function such as catalog management or tax calculation.

# Business contexts

- BaseContext
- EntitlementContext
- GlobalizationContext
- ContentContext
- TaskContext
- AuditContext
- PreviewContext
- ExperimentContext

© 2008 IBM Corporation

WebSphere Commerce predefines the contexts shown here.

BaseContext contains the basic attributes that an activity needs, such as store ID, caller ID, and the run-as ID.

EntitlementContext holds information about entitlement criteria, such as reduced prices for gold club membership.

GlobalizationContext helps components determine locale-specific information such as what language a message should be rendered in, or what currency should be used in the calculation of a price.

If Workspaces are enabled, ContentContext determines the content or business objects that can be displayed or edited based on workspace task group information and TaskContext determines which task an administrator is currently performing.

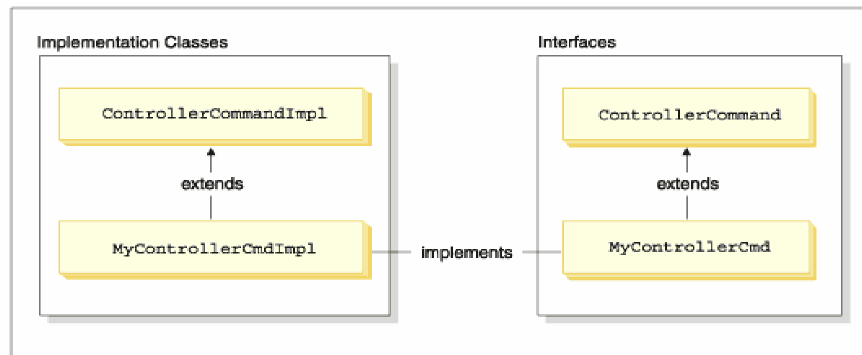
AuditContext is typically provided by vendor components. You might want to bridge the gap to the vendor interface instead of programming to it directly. This context enables you to connect to a different vendor's implementation of the service in the future without the need to rewrite your component.

The preview context allows you to validate independent content without influencing other users. The context object associated with the preview operation represents the state information that is used when deciding the content to display. The preview context contains the preview date that is used to render the content to be displayed.

ExperimentContext is used to store the result of all active experiments for individual users, where the result is a system-generated number which determines the control or test element to be selected in the experiment. This information is persisted throughout the user session, so the same result is used in the same session without re-generation of the number.

# Controller command programming model

- Implementation class extends from ControllerCommandImpl
- Interface extends from ControllerCommand
- Methods to implement and their usage





© 2008 IBM Corporation

Command beans follow a specific design pattern. Every command includes both an interface class and an implementation class. A new controller command should extend the abstract controller command class ControllerCommandImpl. When writing a new controller command, you should override six methods from the abstract class.

First, isGeneric. In the standard WebSphere Commerce implementation there are multiple types of users. These include generic, guest, and registered users. The generic user has a common user ID that is used across the entire system. This common user ID supports general browsing on the site in a manner that minimizes system resource usage. The isGeneric method returns a Boolean value which specifies whether the command can be invoked by the generic user.

Second, isRetriable. The isRetriable method returns a Boolean value which specifies whether the command can be retried on a transaction rollback exception. The isRetriable method of the new controller command superclass returns a value of false. You should override this method and return a value of true if your command can be retried on a transaction rollback exception.

Third, setRequestProperties. The setRequestProperties method is invoked by the Web controller to pass all input properties to the controller command. The controller command must parse the input properties and set each individual property explicitly within this method. This explicit setting of properties by the controller command itself promotes the concept of type safe properties.

Fourth, validateParameters. The validateParameters method is used to do initial parameter checking and any necessary resolution of parameters. This method is called before both the getResources and performExecute methods.

Fifth, getResources. This method is used to implement resource-level access control. It returns a vector of resource-action pairs upon which the command intends to act. If nothing is returned, no resource-level access control is performed.

Sixth, performExecute. The performExecute method contains the business logic for your command. It should invoke the performExecute method of the command's superclass before any new business logic is invoked. At the end, it must return a view name.

## Extending an existing controller command

```
public class ModifiedControllerCmdImpl extends
ExistingControllerCmdImpl
implements ExistingControllerCmd
{
     public void performExecute () throws
com.ibm.commerce.exception.ECException
     {
     /* Insert new business logic that must be executed before the
original command.*/
     // Execute the original command logic.
     super.performExecute();
     /* Insert new business logic that must be executed after the
original command. */
     }
}
```


© 2008 IBM Corporation

Suppose there is an existing WebSphere Commerce controller command, called ExistingControllerCmd. Following the WebSphere Commerce naming conventions, this controller command will have an interface class named ExistingControllerCmd and an implementation class named ExistingControllerCmdImpl. Now assume that a business requirement arises and you must add new business logic to this existing command. One portion of the logic must be run before the existing command logic and another portion must be run after the existing command logic.

The first step in adding the new business logic is to create a new implementation class that extends the original implementation class. In this example, you create a new ModifiedControllerCmdImpl class that extends the ExistingControllerCmdImpl class. The new implementation class should implement the original interface (ExistingControllerCmd).

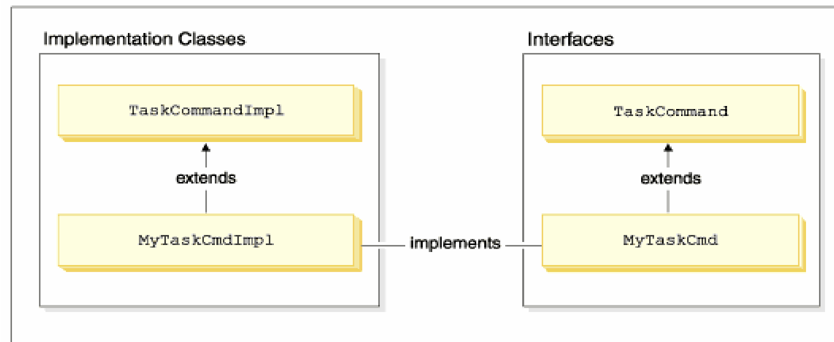
In the new implementation class, you must create a new performExecute method to override the performExecute of the existing command. Within the new performExecute method, there are two ways in which you can insert your new business logic. You can either include the code directly in the controller command, or you can create a new task command to perform the new business logic. If you create a new task command then you must instantiate the new task command object from within the controller command.

The code snippet demonstrates how to add new business logic to the beginning and end of an existing controller command by including the logic directly in the controller command.

Regardless of whether you include the new business logic in the controller command, or create a task command to perform the logic, you must also update the CMDREG table in the WebSphere Commerce command registry. This is required to associate the new controller command implementation class with the existing controller command interface.

# Task command programming model

- Implementation class extends fromTaskCommandImpl
- Interface extends from TaskCommand
- performExecute method
- Simple example





© 2008 IBM Corporation

Since controller commands encapsulate the logic for a business process, they frequently invoke individual task commands to perform specific units of work in the business process. A new task command should extend the abstract task command class TaskCommandImpl and implement an interface that extends the TaskCommand interface. The diagram illustrates the relationship between the implementation class and interface of a new task command with the existing abstract implementation class and interface.

All the input and output properties for the task command must be defined in the command interface, for example MyTaskCmd. The caller programs to the task command interface, rather than the task command implementation class. This enables you to have multiple implementations of the task command (one for each store), without the caller being concerned about which implementation class to call.

All the methods defined in the interface must be implemented in the implementation class. Since the command context should be set by the caller (a controller command), the task command does not need to set the command context. The task command can, however, obtain additional session information by using the command context. In addition to implementing the methods defined in the task command interface, you should override the performExecute method from the abstract task command class. The performExecute method contains the business logic for the particular unit of work that the task command performs. It should invoke the performExecute method of the task command's superclass, before performing any business logic.

The runtime framework calls the getResources method of the controller command to determine which protectable resources the command will access. It might be the case that a task command is invoked during the scope of a controller command and it attempts to access resources that were not returned by the getResources method of the controller command. If this is the case, the task command itself can implement a getResources method to ensure that access control is provided for protectable resources. Note that, by default, getResources returns null for a task command and resource-level access control checking is not performed. Therefore, you must override this if the task command accesses protectable resources.

# Command error handling

- Exception types
  - ECApplicationException – no retry
  - ECSystemException – retry allowed
- Customized code and exception handling
- Exception handling flow
- Exception logging
  - WebSphere Application Server's logs
  - WebSphere Commerce tracing facility

© 2008 IBM Corporation

WebSphere Commerce uses a well-defined command error handling framework that is simple to use in customized code. By design, the framework handles errors in a manner that supports multicultural stores. A command can create one of two exception types.

An ECApplicationException is created if the error is related to user input and will always fail. For example, when a user enters an incorrect parameter, an ECApplicationException is created. When this exception occurs, the solution controller does not retry the command, even if it is specified as a retriable command.

An ECSystemException is created if a runtime exception or a WebSphere Commerce configuration error is detected. Examples of this type of exception include create exceptions, remote exceptions, and other EJB exceptions. When the exception was caused by either a database deadlock or database rollback, the solution controller retries the command, if it is retriable.

When creating new commands, it is important to include appropriate exception handling. You can take advantage of the error handling and tracing provided in WebSphere Commerce, by specifying the required information when catching an exception.

The exception handling flow begins when the solution controller invokes a controller command. The command creates an exception that is caught by the solution controller. This can be either an ECApplicationException, or an ECSystemException. For a Web application, the struts framework determines the error global forward and invokes the specified error view. When invoking the view command, the solution controller composes a set of properties from the ECException object and sets it to the view. The ErrorDataBean passes the error parameters to the message helper object. The StoreErrorDataBean maps the error codes to messages.

Exception handling is tightly integrated with the logging system. When a system exception occurs, it is automatically logged. Exception messages can be written to both the WebSphere Application Server logs and the WebSphere Commerce trace files.

## WebSphere Commerce data beans

- Data beans are Java bean objects that extend access beans
- Access beans wrap the entity EJB objects to provide a simple interface to the database
- Data beans are grouped by subsystem
- Three types:
  - SmartDataBean
  - CommandDataBean
  - InputDataBean

© 2008 IBM Corporation

Extending business logic might also result in new or modified data beans. A data bean is a Java bean that is mainly used to provide dynamic data in JSP pages. There are two types of data beans: smart data beans and command data beans.

A smart data bean uses a lazy fetch method to retrieve its own data. This type of data bean can provide better performance in situations where not all data from the access bean is required, since it retrieves data only as required. Smart data beans that require access to the database should extend from the access bean for the corresponding entity bean. For example, the ProductDataBean data bean extends the ProductAccessBean access bean, which corresponds to the Product entity bean. Some smart data beans do not require database access. For example, the PropertyResource smart data bean retrieves data from a resource bundle, rather than the database.

A command data bean relies on a command to retrieve its data and is a more lightweight data bean. The command retrieves all attributes for the data bean at once, regardless of whether the JSP page requires them. As a result, for JSP pages that use only a selection of attributes from the data bean, a command data bean can be costly in terms of performance. While access control can be enforced on a data bean level when using the smart data bean, this is not true for command data bean. Only use command data beans if using a smart data bean is impractical.

A data bean implementing the InputDataBean interface retrieves data from the URL parameters or attributes set by the view. Attributes defined in this interface can be used as primary key fields to fetch additional data. When a JSP page is invoked, the generated JSP servlet code populates all the attributes that match the URL parameters, and then activates the data bean by passing the data bean to the data bean manager. The data bean manager then invokes the data bean's setRequestProperties() method to pass all the attributes set by the view.

## WebSphere Commerce data beans

- All changes to entity beans are reflected in the access beans and data beans immediately upon regenerating them in Rational Application Developer
- Used in JSP pages to deliver dynamic database content
- Two ways to activate:
  - wcbase:useBean tag (recommended)
  - Data bean manager activate method

© 2008 IBM Corporation

A data bean normally extends an access bean. The access bean, which can be generated by Rational Application Developer, provides a simple way to access information from an entity bean. When modifications are made to an entity bean, the update is reflected in the access bean as soon as the access bean is regenerated. An example of a modification would be adding a new field, a new business method, or a new finder method. Since the data bean extends the access bean, it automatically inherits the new attributes. As a result of this relationship, no coding is required to enable the data bean to use new attributes from the entity bean.

WebSphere Commerce data beans require activation before their use. WebSphere Commerce provides a Commerce-specific version of the useBean tag, wcbase:useBean, that performs data bean activation in a Java-free manner and is the recommended method of data bean activation in store JSP pages. The activate method of the data bean manager is a more generic method of activating a data bean and requires scriptlet code in your JSP.

# Access control policies in WebSphere Commerce

- Understanding access control policies
  - Users
  - Resources
  - Actions
  - Relationships
- Two levels of authorization control:
  - WebSphere Application Server
  - WebSphere Commerce (fine-grained)
- Access control is implemented as a means of supporting authorization for resources

© 2008 IBM Corporation

Access control in a WebSphere Commerce application is composed of four elements: users, actions, resources, and relationships.

Users are the people that use the system. For access control purposes, users must be grouped into relevant access groups. One common attribute that is used to determine membership of an access group is roles. Roles are assigned to users on a per organization basis. Some examples of access groups include registered customers, guest customers, or administrative groups like customer service representatives.

Actions are the activities that users can perform on the resource. For access control purposes, actions must also be grouped into relevant action groups. For example, a common action used in a store is a view. A view is invoked to display a store page to customers. The views used in your store must be declared as actions and assigned to an action group before they can be accessed.

Resources are the entities that are protected. For example, if the action is a view, the resource to be protected is the command that invoked the view, for example, ViewCommand. For access control purposes, resources are grouped into resource groups.

Relationships are the relationship between a user and the resource requested. Access control policies might require that a relationship between a user and the resource be satisfied. For example, users might only be allowed to display the orders that they have created.

These four elements are combined to create access policies. Access control is based on a user's context, action invoked, resources used, and the relationships within the stores and site implementation.

In a WebSphere Commerce application, there are two main levels of access control. The first level of access control is performed by the WebSphere Application Server. In this respect, WebSphere Commerce uses WebSphere Application Server to protect enterprise beans and servlets. The second level of access control is the fine-grained access control system of WebSphere Commerce.

The WebSphere Commerce access control framework uses access control policies to determine if a given user is permitted to perform a given action on a given resource. This access control framework provides fine-grained access control. It works in conjunction with, but does not replace, the access control provided by the WebSphere Application Server.

## Types of access control

- There are two types of access control
  - Command-level (broad)
    - Also known as role-based
    - Specifies that users assigned to a particular role can run certain commands
  - Resource-level (very fine)
    - Specifies the relationship that a user must have with a resource before a given action can be performed

© 2008 IBM Corporation

There are two types of access control, both of which are policy-based: command-level access control and resource-level access control.

Command-level, also known as role-based, access control uses a broad type of policy. You can specify that all users of a particular role can run certain types of commands. For example, you can specify that users with the Account Representative role can run any command in the AccountRepresentativesCmdResourceGroup resource group. All controller commands must be protected by command-level access control. In addition, any view that can be called directly, or that can be launched by a redirect from another command must be protected by command-level access control. Command-level access control determines whether a user is allowed to run the particular command within the store you have specified.

By contrast, resource-level access control uses a fine grain policy. If a command-level policy allows a user to run a command, a subsequent resource-level access control policy can be applied to determine if the user can access the resource in question. Resource-level access policies define the relationship a user must have with a resource in order to perform an action on it. For example, a seller administrator might be permitted to perform an administrative task but only on resources that are owned by the organization for which they are a seller administrator.

To summarize, in command-level access control the "resource" is the command itself and the "action" is to run the command within the current store. The access control check determines if the user is permitted to run the command within the current store. In resource-level access control the "resource" is any protectable resource that the command or bean accesses and the "action" is the command itself.

IBM Software Group

IBM

# Access control policy utilities

- Policies are defined in a specific XML format:
  - See examples in <WC_HOME>\xml\policies\xml
  - Create XML files that conform to DTDs in \xml\policies\dtd
- Can load through the SAR publish process
- Four tools to work with policies:
  - acugload  (Load access groups)
  - acpload  (Load policy groups)
  - acpnlsload  (Load NLS descriptions)
  - acpextract  (Extract policies and relationships)
- View policies in Organization Administration Console

22

Developing and customizing store business logic

© 2008 IBM Corporation

All access control policies are defined in XML format. You can look at any of the predefined WebSphere Commerce policies for an example of how to create your policy. DTD documents are provided to define the expected format.

When you publish a new store using the Store Archive (SAR) publish process, the access control policies defined for that store are loaded automatically.

If you create or update access control policies for an existing store, they need to be loaded into the WebSphere Commerce database to take effect. There are three utility tools that you can make use of when loading access control policies. The acugload utility loads the XML files containing the user (access) group definitions. The acpload utility loads the XML files containing the main access control policies and the acpnlsload utility loads the XML files containing the display names and descriptions. If you need to retrieve access control policies that have already been loaded you can use the acpextract utility. It extracts the access control policy and access group information in the database and generates files that capture the information in XML format. It uses an input filter XML file to specify the data to extract from the database.

Once your policies are loaded, you can view them using the Organization Administration Console. It is also possible to change existing policies through this tool.

## Sample components to customize

- Business policy framework
- WebSphere Commerce subsystems
- WebSphere Commerce customizable features

© 2008 IBM Corporation

The remaining slides introduce you to some of the commonly customized components and features of WebSphere Commerce. The business policy framework is discussed followed by an overview of the various WebSphere Commerce subsystems, and finally some additional customizable features. This presentation does not go into detail on specific customization strategies for each component however the techniques covered so far form the basis of most customization tasks. You can find detailed information and customization notes on each of the components discussed in the Information Center.

# Business policy framework (WebSphere Commerce Enterprise)

- Business policies
- Business accounts
- Contracts and service agreements
- Terms and conditions

© 2008 IBM Corporation

Business policies are sets of rules followed by a store or group of stores that define business processes, industry practices, and the scope and characteristics of a store or group of stores offerings. Business policies also define how the store or site interacts with customers and other business partners. For example, your site might have business policies determining when and how customers are allowed to return products to a store, or business policies that determine what payment methods your store accepts.

WebSphere Commerce provides a framework that allows you to implement your store's business policies in your online store or site. The business policy framework consists of business policies, business accounts, contracts and service agreements and terms and conditions.

In most instances, you will have predefined business policies for your business that you need to implement in your online store or site. WebSphere Commerce provides a set of business policies that you can use as is, or change to meet your needs. Business accounts define the relationship between a customer and your business. Business accounts track contracts and orders for customer organizations and configure how buyers from customer organizations shop in a store. Before a customer or business partner, for example resellers or distributors, can access your store, you must create a contract or service agreement that defines customer or business partner access to your store. In the WebSphere Commerce business policy framework, you create contracts for customers and service agreements for other types of business partners. A contract with a customer defines what areas of your store the customer can access, what prices the customer will see, and for how long the customer has access to your site and those prices. All stores must contain at least one contract, as without a contract no one but internal administrators can access your store. WebSphere Commerce provides a default contract that applies to all customers shopping at a store. A service agreement with a business partner (business partners can be resellers, distributors, manufacturers, suppliers, or other partners) defines your arrangement with the business partner. For example, a service agreement with a reseller might define what access the reseller has to your site, whether they can share your catalog, or whether you host a store for them. A service agreement with a distributor might define how customers to your site can receive quotes from a distributor, or how customers can access the distributors site from yours. Finally, terms and conditions define how contracts and service agreements are implemented for a particular customer or business partner. For contracts, terms and conditions define what is being sold under the contract; the price of the items being sold; how the items are shipped to the customer; and how the customer pays for the order. For service agreements with business partners, terms and conditions can restrict the products the business partner is allowed to sell. Terms and conditions typically reference business policies as most aspects of a site or stores operations are defined by business policies. Terms and conditions provide standard parameters for the business policies they reference. Providing parameters to the business policies allows you to modify the behavior of business policies for each contract.

IBM

# WebSphere Commerce subsystems

- Catalog subsystem
- Marketing subsystem
- Member subsystem
- Order management subsystem
- Payments subsystem
- Trading subsystem

© 2008 IBM Corporation

The catalog subsystem is a component of the WebSphere Commerce Server that provides online catalog navigation, partitioning, categorization, and associations. In addition, the catalog subsystem includes support for personalized interest lists and custom catalog display pages. The catalog subsystem contains all logic and data relevant to an online catalog, including catalog groups (or categories), catalog entries, and any associations or relationships among them.

The Marketing subsystem provides numerous marketing concepts to your site, designed to increase brand awareness, and to attract and retain customers. Components of the marketing subsystem provide functionality to create marketing campaigns, including customer segments and advertising; and e-mail activities. All of these components can be extensively customized to ensure that your site marketing strategy matches that of your brick and mortar store.

The Member subsystem is a component of the WebSphere Commerce Server that includes data for participants of the WebSphere Commerce system. A member can be a user, a group of users, or an organizational entity. Business logic in the Member subsystem provides member registration and profile management services. Other services which are closely related to the Member subsystem include access control, authentication, and session management.

The Order management subsystem supports order capture in addition to order, inventory and payment processing. Order capture provides functionality such as sales quotes and shopping carts and order submission. There are many ways to create shopcarts and submit orders. Order processing is responsible for the overall coordination of inventory allocation, payment processing, releasing the order to fulfillment, and tracking order status. WebSphere Commerce supports two inventory systems: Available to promise (ATP) and non-ATP. The interface to inventory is encapsulated by a single inventory task command, which in turn invokes either ATP or non-ATP task commands. WebSphere Commerce Payments supports the use of payment plug-ins for offline or online payment processing.

New in WebSphere Commerce version 6, the Payments subsystem is included in the WebSphere Commerce Server. The WebSphere Commerce instance contains payment information. The Payments subsystem connects to Payment Service Providers by using payment plug-ins. The new Payments subsystem is introduced in the **Payments Technical Overview** presentation.

The trading subsystem in WebSphere Commerce provides the logic, function and data relevant for negotiating the price and quantity of a product or set of products between the buyer and seller organization. The trading subsystem includes auctions, contracts, and Request for Quote components that are used to carry out specific transactions between organizations.

# WebSphere Commerce customizable features

- Currency
- Returns
- Tax codes
- Calculation framework

© 2008 IBM Corporation

You can display prices in your site in one currency, or you can use multiple currencies. For a site with multiple stores, you can use different currencies for the stores, or you can assign currencies to the store group. Depending on the type of site that you are creating, you can specify what currencies you want to use and how they are displayed. You can also allow customers to select a shopping currency. If a currency you want to support is not in the store by default you can add it to the CURLIST table to provide support.

Every product that is modified within a return goes through an automatic approval process. With the appropriate level of security, manual approval can be issued, but for a typical return, automatic approval is easier. Automatic approval consists of a series of tests on the specified product and its relationship to the rest of the return, other returns, and the original order. For each unsuccessful test, a denial reason is logged against the return item. The reasons can be presented to a Customer Service Representative who can override the system evaluation and issue a manual approval.

A tax calculation code indicates the tax calculation for order items. A store typically collects two type of taxes: sales or use tax, and shipping tax. The tax codes are unique within each tax type for a store. Only one tax calculation code of each tax type is applied to a particular order item. Tax calculation codes can be classified for convenience. A tax code scheme consists of a group of tax code classifications. A store typically uses a single tax code scheme. TaxCategory objects correspond to the different kinds of tax that a store might be required to collect, such as federal, state or provincial, and municipal. Taxes for each TaxCategory object are calculated in ascending sequence by their sequence attributes.

Web commerce systems need to calculate monetary amounts and apply them to business objects. Business rules and legal requirements specify how and when these monetary amounts should be calculated. When these rules and requirements change, a good Web commerce system can adapt to the changes with little or no programming changes. WebSphere Commerce provides a flexible, generic framework that can be used to implement different kinds of calculations and apply them to the business objects. The framework can handle a wide variety of business and legal requirements without programming. WebSphere Commerce provides many method implementations from which you can select to do the calculations. If business or legal requirements require a programming change, you can make many such changes just by programming additional method implementations, without having to change existing programming. These implementations can be overridden.

## Summary

- Command framework
- Business context service
- Command programming model
- Customizable components

© 2008 IBM Corporation

This presentation introduced you to developing and customizing WebSphere Commerce store business logic. The command framework was discussed in detail followed by an introduction to the new business context service. Customization methodology was discussed in the command programming model section and the presentation concluded with a discussion of customizable business components in WebSphere Commerce.

# Feedback

## Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_DevelopingAndCustomizingStoreBusinessLogic.ppt

This module is also available in PDF format at:
../wcs60_DevelopingAndCustomizingStoreBusinessLogic.pdf

© 2008 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.

# Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM Rational WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

EJB, Java, JSP, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2008. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.