IBM® WebSphere® Commerce Multiple Approvals Reference Application

---

IBM® WebSphere® Commerce
Business Edition

Version 5.4

Last Updated: June 12, 2002

Introduction:

This reference application demonstrates an extended WebSphere Commerce 5.4 BE Approvals flow for the 'User Self-Registration' business process. This functionality would allow the User Registration transaction to be approved by multiple approvers, prior to being successful.

E.g. A user Self-Registers, depending on the Approvals role setup, his transaction may need to be approved by ApproverA, ApproverB, ... and ApproverN. (For implementation details refer to latter sections in this document).

Prerequisite Environment:

Operating System	Windows NT / 2000
Commerce Server	IBM WebSphere Commerce Business Edition 5.4
Application Server	WAS 4.x
WCS Database	IBM DB2 (with required fix-patches for WCS 5.4)

Installing the Reference Application:

To install this sample, you must first install and configure IBM WebSphere Commerce V5.4. Next, unzip the *MultipleApprovalsRefApp.zip* into a temp directory. Then locate the *RefApp*`driver-yyyymmdd.jar` file in the `packagedDriver` directory and copy it to the system that has IBM WebSphere Commerce installed. Open a command prompt and ensure that `java` executable is in your path. Issue `java -jar` *RefAppD*`river-yyyymmdd.jar` and provide the information that is requested.

The install utility is part of the driver. It is packaged into the driver jar file and can be run by issuing java -jar *RefAppD*`river-`yyyymmdd.jar. The install utility source can be found in java/com/ibm/commerce/multipleapprovals/install/InstallDriver.java. At a high level the utility performs the following tasks:

- Unpackages the driver assets into their appropriate directories.
- Updates the commerce database with configuration information.

Examine this document for further information about what it does.

Restart your Commerce Server for the changes to take effect. You should also refer to the log files that were created in the logs directory in the IBM WebSphere Commerce root directory for any problems during the install.

Uninstalling the Reference Application:

To uninstall the Reference Application complete the following steps:

- Run the *uninstallrefapp.bat*, with the *DatabaseName*, *DatabaseAdminID*, *DatabasePassword*. E.g. *uninstallrefapp mall db2admin myPassword*. This will reset your database to original configuration.
- Manually delete the compiled class files under the "…\ *AppServer\installedApps\WC_Enterprise_App_demo.ear\lib\ com\ibm\commerce\multipleapprovals\commands"* directory.
- Manually delete the *MultipleApprovals_en_US.properties* file under the "*…\ \AppServer\installedApps\WC_Enterprise_App_demo.ear\properties"* directory.

Building the sample:

Unzipping the *MultipleApprovalsRefApp.zip* would create the following directory structure for the build tree:

```
WCS_Ref_Apps
  buildjars
  MultipleApprovalsRefApp
    driver
      ear
        lib
      META-INF
      wcs
        bin
        schema
          xml
    java
      com
        ibm
          commerce
            multipleapprovals
              install
              commands
    Uninstall
    packagedDriver
```

Directory	Description	Contents
WCS_Ref_Apps	This is the root directory.	buildjars directory MultipleApprovalsRefApp directory Readme.htm
buildjars	This directory contains	ivjejb35.jar

	dependency JAR files that are used to build the Java™ source in this project. These JARs are taken from the current version of IBM WebSphere Application Server and IBM WebSphere Commerce.	j2ee.jar wcsejbimpl.jar wcsejsclient.jar wcslogging.jar wcsmcruntime.jar wcsruntime.jar wcssfc.jar wcsbusinessflow.jar
MultipleApprovalsRefApp	This is the root directory for the sample application.	driver directory java directory Uninstall directory build.xml
driver	This directory contains the assets that will be built into the installable archive.	ear directory META-INF directory wcs directory
ear	This directory contains the assets that will be copied to the installed commerce enterprise archive directory.	lib directory properties directory
lib	The compiled class files are contained in this directory.	Compiled class files for Commands to be used by this application, in a qualified Directory structure: ApprovalMultipleSetUpRecordsCmdImpl.class HandleMultipleApprovalsCmdImpl.class
properties	This directory contains the property file that will be copied to the installed commerce enterprise archive directory.	MultipleApprovals_en_US.properties
driver/META-INF	This directory contains the MANIFEST.MF file for the driver JAR file.	MANIFEST.MF - indicates the Java class that should be invoked when the driver JAR is executed.
wcs	This directory contains assets that need to be installed in the root directory of IBM WebSphere Commerce.	bin directory schema directory
bin	Executables required to install and configure this application.	*multipleApprovalsPopulatedb.db2.bat* - This script populates the commerce database with configuration information. This script is used if the database is DB2 and the operating system is Windows.
schema	Assets required to populate the database.	xml directory.
schema/xml	Massloader XML document.	*multipleApprovals.xml* - Massloader import document used to populate the commerce database with configuration information for this application.
java	This directory contains the Java source.	Java source files are located in directories that model their package names.

Uninstall	The directory contains batch files to uninstall the reference application	uninstall.bat uninstall.sql.bat
packagedDriver	This directory contains the packaged driver.	The *RefAppDriver-yyyymmdd.jar* file is located in this directory.

The contents of this source tree are available in the attached *MultipleApprovalsRefApp.zip* file. This zip file excludes dependency jars found in the buildjars directory. In order to run the build, you will need to copy these jars from your installation of IBM WebSphere Commerce Business Edition and IBM WebSphere Application Server and place them in the buildjars directory. The first two can be found in the IBM WebSphere Application Server lib directory. The other jars can be found in the lib and wc.ear/lib directories in the IBM WebSphere Commerce root directory:

- ivjejb35.jar
- j2ee.jar
- wcsejbimpl.jar
- wcsejsclient.jar
- wcslogging.jar
- wcsmcruntime.jar
- wcsruntime.jar
- wcssfc.jar
- wcsbusinessflow.jar

Special attention should be paid to the build.xml document. This example uses Apache Ant to perform the build. Ant is a Java-based build tool like the make tool. The build.xml document contains the instructions that Ant uses to build the application. To run this build you will need to download a copy of Ant and ensure that the ant/bin directory is in your path. Then make the *MultipleApprovalsRefApp* directory your current directory and issue ant. (For more information about Ant refer to http://jakarta.apache.org/ant/index.html.) The build.xml document contains additional documentation on the build process. The basic build process is to compile the entire Java source into a lib directory in the driver/ear directory, then finally package the driver up into a jar file and place the jar in a directory called packagedDriver. Before you start the build you will need to ensure that java executable is in your path. You should ensure that the JDK you are using is the same as the one shipped with your version of IBM WebSphere Application Server.

Design and Implementation Overview

Objective:

To design and implement a reference demo application, demonstrating an extended WCS 5.4 BE Approvals Flow.

Current Business Flow:

Transactions may require that an individual approve some electronic marketplace actions before they proceed. This individual, called an approver, can accept or reject requests to perform a specific action. Approvals can be activated for the following WCS 5.4 BE business processes:

- Order Process
- User Registration
- Contract Submit
- RFQ Response

The current business flow is as follows (for User Self Registration):

- Respective Controller Command for the business process is invoked (e.g. *UserRegistrationAdd*).
- A Default implementation intervenes and starts the approval flow.
- An invoked PreApproval Task Command (e.g. *UserRegistrationAddPreApproval*) creates records, sets state to *pending_approval* and returns control to the default implementation.
- An invoked CheckApproval (e.g.. UserRegistrationAddCheckApproval) Task Command checks to determine if approval is required and returns control to the default implementation.
- Assuming CheckApproval returns Approval Required, relevant approvers are searched:
- If no approver group is found, an approval is not required; the state is changed from *pending_approval* to *approved*.
- If an approver group is found but is empty, exception is thrown.
- Otherwise the approvers are notified.
- Approvers logon to either approve or reject, this would invoke *HandleApprovalsCmd*. This command starts the default implementation flow, and this would invoke either:
    - A *PostApproval* (e.g. *UserRegistrationAddPostApproval*) command, in which case the state is changed to *approved*; or
    - A *PostReject* (e.g. *UserRegistrationAddPostReject*) command, in which case the state is changed to *rejected*.
- Control is returned to *HandleApprovalsCmd*.

Feature Scenario:

This demo application in addition to the 'out-of-the-box' approval scenarios, shall demonstrate Multiple Approvals functionality for the User Self-Registration process. This functionality would allow a transaction to be approved by multiple approvers, prior to being successful. E.g. A user Self-Registers, depending on the Approvals role setup, his transaction may need to be approved by ApproverA, ApproverB, ... and ApproverN.

Design:

This Demo Approval application is designed and packaged on top of WCS 5.4 BE, and incorporates Multiple Approval functionality.

The *APRVSTATUS* table for approvals Stores approval requests and their status. It contains multiple rows for each action awaiting approval, one row for every potential approver. Each entry is unique per approver, entity tuplet. FLOWTYPE_ID tells us which type of ENTITY_ID is pending approval.

Column Name	*Column Type*	*Column Description*
ACTIONTIME	TIMESTAMP NULL	When the approval record was acted on (that is, the time it was approved or rejected).
APPROVER_ID	BIGINT NULL	The ID of the user eligible to approve or reject the requested action.
APRVSTATUS_ID	BIGINT NOT NULL	Primary key.

COMMENTS	VARCHAR (254) NULL	Comments entered by approver during approval or rejection.
ENTITY_ID	BIGINT NOT NULL	Reference to the business object -- that is, which particular business object instance within the business object type specified by the FLOWTYPE_ID.
FLOW_ID	BIGINT NOT NULL	Reference to the flow for the steps in the approval process. Foreign key FLOW_ID.
FLOWTYPE_ID	BIGINT NOT NULL	Foreign key reference to the type of flow which determines the type of business object included in the approval. There are different IDs for each RFQ, Orders, User Registration, Contracts, Organization Registration and so on. Foreign key to FLOWTYPE_ID.
FLSTATEDCT_ID	BIGINT NOT NULL	Current state of the flow instance.
MBRGRP_ID	BIGINT NOT NULL	The ID of the Member Group to which the approver belongs making her an approver for this record.
STATUS	INTEGER NOT NULL	Approval status of this action:0 = pending, 1 = approved, 2 = rejected.
SUBMITTER_ID	BIGINT NOT NULL	ID of the user requesting the action which needs approval.
SUBMITTIME	TIMESTAMP NULL	The time that the action requiring approval was requested.

The default implementation incorporates a feature that allows for the flexible creation and alteration of business processes, this flow allows developers to modify the business flows across the various subcomponents of WebSphere Commerce. The default implementation manages the Approval business flow, by making use of a State-Machine; this business flow, however, need not be modified to incorporate for the new multiple Approvals functionality.

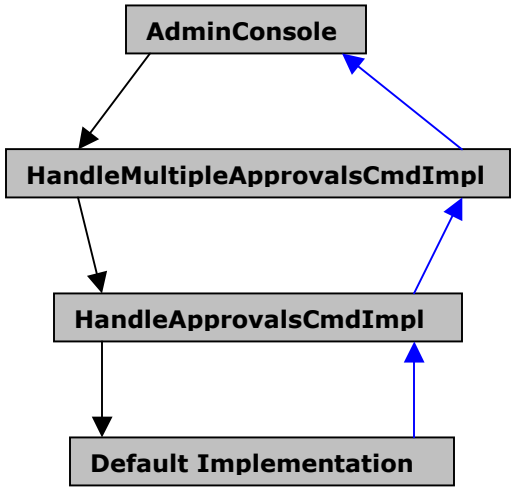
A new controller command ***'HandleMultipleApprovalsCmdImpl'*** is created, this command extends from the current *HandleApprovalsCmdImpl* and implements the *HandleApprovalsCmd* interface (mapped in the *CMDREG* table). This command would be triggered from within the WCS AdminConsole when an Approver either rejects or approves the request(s), and would incorporate the following business logic:

- *Reject*, in this case the HandleMultipleApproversCmd is called, it checks for the reject flag, and runs a super.performExecute() (HandleApprovalsCmd). This would then invoke the default implementation, set the request status to *rejected = 2* (*UserRegitrationAddPostRejectCmd*) and delete other pending submissions sent to Approvers. Also the default implementation reaches completion and its *State = rejected* is set. Control is returned to HandleApprovals, success is judged, and control returned to *HandleMultipleApprovalsCmd*, this command then completes execution.

- *Approve*, in this case the HandleMultipleApproversCmd is called, it checks for the Approve flag. It fetches the ***multipleApprovalContant*** constant from the *MultipleApprovals_en_US.properties* file and checks if this is the last approval required for the submission. If it is not the last approval required, the command sets the Status of the Approval (for the particular Approver) as approved. However, if it is the last required approval, HandleApprovalsCmd is invoked with appropriate parameters. The default implementation is invoked, it sets the request status to *approved = 1* (*UserRegitrationAddPostApproveCmd*) and deletes other pending submisions sent to Approvers. Also as the default implementation reaches completion and it's *State = rejected*

is set. Control is returned to HandleApprovals, success is judged, and control returned to *HandleMultipleApprovalsCmd*, this command then completes execution, and the finally the submitter is Approved.

Call-stack snapshot:



Also a task command ***'ApprovalMultipleSetupRecordsCmdmpl'*** is created, this command extends from the current *ApprovalSetupRecordsCmdmpl* and implements the *ApprovalSetupRecordsCmd* interface. This command in addition to checking validity of the Approval request, and creates the approval record in the *APRVSTATUS* table. New added functionality includes the check for enough approvers being assigned. User self-registration is not allowed if enough approvers (=> *multipleApprovalContant* parameter) are not assigned*,* and an ECSystemException *_ERR_NOT_ENOUGH_APPROVERS* shall be thrown by the *ApprovalMultipleSetupRecordsCmdImpl*. The text for this exception message resides in the MultipleApprovals_en_US.properties file.

Note: This reference application is implemented for the **User Self Registration** business process only; however the code itself contains comments within logic to activate the multiple approval flow for other approvable business processes also.

Customizations to HandleMultipleApprovalsCmdImpl:

HandleMultipleApprovalsCmdImpl is the name of the controller command that processes approvals and rejections. This controller command was modified by extending the original implementation class and registering the new implementation class in the command registry. Refer to java/com/ibm.commerce/multipleapprovals/commands/HandleMultipleApprovalsCmdImpl.java for details on how the controller command was customized. Refer to schema/xml/ multipleApprovals.xml for details on how the new implementation was registered in the command registry.

The approver may either approve or reject one or more approval requests. The command will handle the updating of the APRVSTATUS table as well as any other activities required. The status of the approval requests record in the APRVSTATUS table will be updated to 1 for approved or 2 for rejected. The command treats each approval or rejection as a separate transaction so it is possible that if the command is invoked to process a batch of approvals or rejections, some may succeed and others may fail. Upon the last required approval or rejection *super.performExecute()* will be called hence raising a business flow event to handle each approval or rejection. The business flow

event will handle the updating of the APRVSTATUS table as well as any other activities which have been defined for this transition.

The performExecute() method of the command contains the business logic for multiple approval handling. The current logic enables multiple approval flow for the User Registration business process, however by minor customization the functionality could be enabled for the flowing business processes:

- Order Process
- Contract Submit
- RFQ Response

Modifying conditions for the following 'if' block would allow for these business processes to be Multiple Approval enabled:

```
if (aprvFlag == 2 ||
!flowTypeIdentifier.equalsIgnoreCase(ApprovalConstants.EC_APPROVAL_FLOWTYPE_USER_REGIST
RATION))
{
        …..
        …..
}
```

Customizations to ApprovalMultipleSetUpRecordsCmdImpl:

ApprovalMultipleSetUpRecordsCmdImpl is the name of the task command that creates an approval record for each potential approver of the action. This controller command was modified by extending the original implementation class and registering the new implementation class in the command registry. Refer to java/com/ibm.commerce/multipleapprovals/commands/ ApprovalMultipleSetUpRecordsCmdImpl.java for details on how the controller command was customized. Refer to schema/xml/ multipleApprovals.xml for details on how the new implementation was registered in the command registry.

This command incorporates added logic to check for enough approvers being assigned. User self-registration is not allowed if enough approvers (*multipleApprovalContant* parameter) are not assigned and an ECSystemException *_ERR_NOT_ENOUGH_APPROVERS* shall be thrown by the *ApprovalMultipleSetupRecordsCmdImpl*. The text for this exception message resides in the MultipleApprovals_en_US.properties file.

To enable this check for other business processes modify conditions to the following 'if' block found in the performExecute () method of the class:

```
if (approvers.length < multipleApprovalConstant &&
aRequest.getRequestName().equalsIgnoreCase(ApprovalConstants.EC_APPROVAL_FLOWTYPE_USER
_REGISTRATION))
{
        throw new ECSystemException(
              notEnoughApproversEx,
              getClass().getName(),
               methodName,
               ECMessageHelper.generateMsgParms(approverGroupName, orgId));
}
```

Customizations to the MultipleApprovals_en_US.properties file:

The MultipleApprovals_en_US.properties files is the resource bundle for the Multiple Approvals Application. The file contains the following two keys:

- ***HandleApprovals.multipleApprovalContant*:** the value of this is set to '*2*', hence an approval request would require two approvers to approve the request. Changing the value of this variable would alter the number of approvals required for the reference application.
- ***_ERR_NOT_ENOUGH_APPROVERS:*** This holds the message to the error key corresponding to the exception, not enough approvers assigned. This message could be customized as per requirements, additional keys could be added to this file to handle exception conditions.

Customization to WCS published store:

The multiple approval reference application is independent to store logic, a published store is not required. However, a published *Tooltech* B2B store would help in understanding and utilizing the multiple approval flow functionality. The *UserRegistrationNew.jsp* provided with the *Tooltech* SAR handles exceptions thrown during user self registration; it provides with an excellent exception handling logic and does not have to be modified in order to make use of new functionality.

Trademarks and Service Marks

End of document.
