

Windows Workflow Foundation

Hands-On Lab

Lab Manual

Lab 08 – Transactional Behavior in Workflow in C#

Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Microsoft Corporation.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user.  Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document.  Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2005 Microsoft Corporation.  All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, SQL Server, Visual C# and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**Contents**

[Overview 1](#_Toc218634340)

[Objectives 1](#_Toc218634341)

[System Requirements 1](#_Toc218634342)

[Setup 1](#_Toc218634343)

[Physical Folder Structure 1](#_Toc218634344)

[Code Snippets 2](#_Toc218634345)

[Starting Material 2](#_Toc218634346)

[Acronyms Used in this Lab 2](#_Toc218634347)

[Scenario 2](#_Toc218634348)

[Exercise 1 – Atomic Transactions in Workflows 3](#_Toc218634349)

[Task 1 – Set up the solution 3](#_Toc218634350)

[Task 2 – Create the Workflow 5](#_Toc218634351)

[Task 3 – Run the Application 11](#_Toc218634352)

[Exercise 2 – Compensation for Long-running Processes 13](#_Toc218634353)

[Task 1 – Modify the Workflow Project 13](#_Toc218634354)

[Task 2 - Test the application 18](#_Toc218634355)

[Lab Summary 19](#_Toc218634356)

# Overview

Estimated time to complete this lab: **40 minutes**

The project files for this lab are in the *C:\WF\WF 3.5 Labs\Lab08* folder.

This lab will familiarize the reader with atomic transactions, exceptions and compensation as used in workflow. We will be communicating with a database and deliberately throwing exceptions to demonstrate the rollback process of transactions.

## Objectives

The objective of this lab is to demonstrate exceptions, transactions and compensation features of Windows Workflow Foundation.

After completing this lab, you will be able to:

* Use atomic and long-running transactions with compensating behavior in the context of a workflow application
* Manage exceptions within a workflow

More information about Windows Workflow Foundation can be found at <http://msdn.microsoft.com/workflow>

## System Requirements

* Microsoft Visual Studio 2008

## Setup

Unzip the lab to your local hard drive. Everything needed to complete the lab is in the zip file.

### Physical Folder Structure

File paths referenced in this lab assume the lab is installed in the following folder:

*C:\WF\WF 3.5 Labs\Lab[Number]*

Within the ***Lab[Number]*** folder, several child folders are available:

* ***CSharp*** – The lab written for C#
* ***VB*** – The lab written for VB
* ***resources*** –Any files referenced in the lab can be found in the Resources subdirectory, including source code for custom assemblies referenced in the exercises.

Within each *[Language]* folder, several child folders are available:

* ***before*** – The work area for completing the HOL
* ***after*** – The fully completed HOL

### Code Snippets

All code required for this lab consisting of more than 2 lines is available as code snippets. To learn more about code snippets including how to install them and how to use them, see the snippet guide document for the language of your choice in the folder:

*C:\WF\WF 3.5 Labs\Snippets*

## Starting Material

### Acronyms Used in this Lab

* WF – Windows Workflow Foundation

### Scenario

A bank connects to a database crediting and debiting accounts. If money is withdrawn from one account then placed in another account and an exception occurs, such as being overdrawn, both the credit the debit must be rolled back correctly.

# Exercise 1 – Atomic Transactions in Workflows

In this exercise, you will learn how to use atomic transactions in a workflow. Atomic transactions exhibit ACID (Atomicity, Consistency, Isolation and Durability) properties that can enlist in transactions (say in the context of SQL databases). You will create a workflow to model moving money from one bank account to a second bank account. The workflow will consist of a sequence of activities. In the first activity money is withdrawn from bank account A. In the second activity money is deposited in bank account B. In the third activity an exception is thrown. All three activities are wrapped in a **TransactionScope**; this is functionally equivalent to a .Net 2.0 **System.Transactions.TransactionScope**. The two code activities perform the money transfer operation by using ADO.NET to access a SQL Server database that you create as part of the exercise.

You will observe that the database activities are rolled back at the end of the workflow and you will observe that when the exception activity is disabled, the database transaction does not get rolled back.

## Task 1 – Set up the solution

1. It is required that you have either SQL Server or SQL Server Express installed on your machine to continue this exercise. The database scripts provided work with the default SQL Server Express (SQLExpress) database named instance. Using other database instance names requires editing of the scripts.
2. Copy the *Exercise1* directory from *C:\WF\WF 3.5 Labs\Lab08\Resources\* to *C:\WF\WF 3.5 Labs\Lab08\*
3. Open a command window and change your directory to *C:\WF\WF 3.5 Labs\Lab08\Exercise1*
4. Run *setupdatabase.bat* script to create the Bank database we will use for this exercise. The script first creates a database with a name *“Bank”* (if the database is already created, it drops it first). The Bank database will have two tables, with two columns each. We will be recording banking transactions on those tables.

|  |  |  |
| --- | --- | --- |
| **ChequeAccount** |  | **SavingsAccount** |
| ChequeAccountId |  | SavingsAccountId |
| Balance |  | Balance |

1. Since we will be working with transactions, we will need a persistence mechanism to persist the workflow instance as part of the transaction. Please run the script *SetupPersistenceService.bat* in the same folder to setup the **SqlPersistenceService** service’s store. The database name used is *WorkflowStore*. The service is added by default in your host application *program.cs*.
2. Open the solution file *AtomicTransactionsWorkflow.sln* under *C:\WF\WF 3.5 Labs\Lab08\Exercise1\AtomicTransactionsWorkflow\*
3. **Right click** on the *AtomicTransactionWorkflow* project and select **Add | New Item**
4. Choose **Application Configuration File** and click the **Add** button
5. Between the *<configuration>* tags add the following xml:

<configuration>

<appSettings>

<add key="connectionString" value="Integrated Security=SSPI; Persist Security Info=False;Initial Catalog=WorkflowStore; Data Source=.\SQLExpress" />

</appSettings>

</configuration>

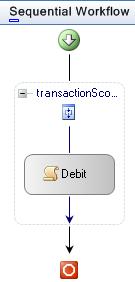
1. This connection string is specifically for a default SQL Express install. If you have a different instance name or are using SQL Server it will be modify the connection string to reflect this. If you are running a non-standard configuration it will be necessary to change the Source to point at the database you are using.

## Task 2 – Create the Workflow

1. Double click on *Workflow1.cs* from the **Solution Explorer** to show the workflow design surface.
2. Drag a **TransactionScope** activity from the **Toolbox** and drop it onto the workflow surface.
3. Drag a **Code** activity from the **Toolbox** and drop it into the box marked *“Drop Activities Here”* inside the **TransactionScope** activity.
4. In the **Properties** window for *codeActivity1* activity set the properties as follows:

* **(Name) –** *Debit*
* **ExecuteCode** – select *DebitCodeHandler*from the dropdown list.

1. Your workflow should look like this:

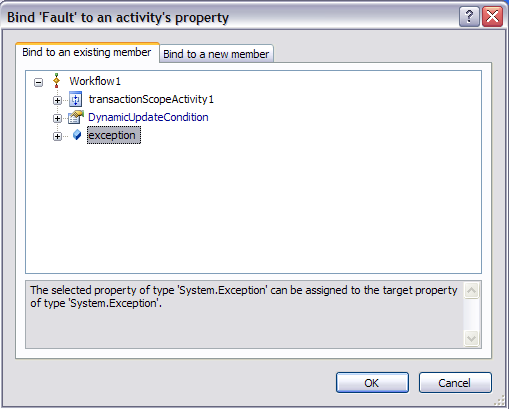


1. Go back to workflow design view and drag a **Code** activity from the **Toolbox**, and drop it underneath the *Debit*activity within the **TransactionScope** activity.
2. In the **Properties** Window for *codeActivity1* activity set the **Properties** as follows:

* **(Name) -** *Credit*
* **ExecuteCode** – select *CreditCodeHandler*from the dropdown list.

1. From the **Toolbox**, drag a **Throw** activity and drop it inside the **TransactionScope** activity underneath the *Credit* activity.
2. In the **Properties** window for *throwActivity1* set the **Properties** as follows:

* **Fault** – Click the ellipses button. In the property binding dialog, click the **Bind to existing member tab**, select exception and click **OK.**

****

* **FaultType** – Click on the ellipses button. In the **“Browse and select a .Net Type”** dialog, browse to find *mscorlib.System.Exception*. Then click **OK**.

1. Select **View Fault Handlers** by clicking the third tab at the base of the workflow designer.

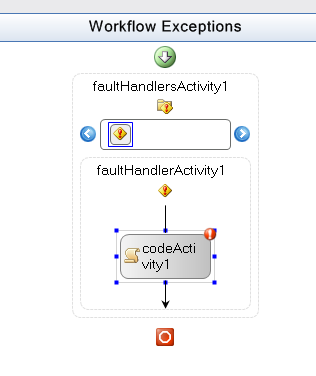
.*ViewFaults*

Add a **FaultHandler** to the workflow by dragging a **FaultHandler** activity from the **Toolbox**, and dropping it into the upper-most box marked *“Drop FaultHandlerActivity Here”.*

1. In the **Properties** window for *faultHandlerActivity1* set the **Properties** as follows:

* **FaultType** – Click on the ellipses button. In the **“Browse and select a .Net Type”** dialog, browse to find *mscorlib.System.Exception*. Then click **OK**.

1. From the **Toolbox**, drag a **Code** activity and drop it into the area marked *“Drop Activities Here”* for *faultHandlerActivity1*.



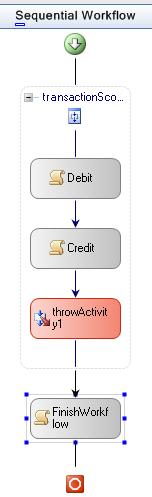
1. In the **Properties** window for *codeAcitivty1* in *faultHandlerActivity1* set the **Properties** as follows:

* **(Name)** –*WorkflowExceptionHandler*
* **ExecuteCode** –select *OnException*from the dropdown list.

1. Go back to the **SequentialWorkflow** view of the workflow by clicking on the first tab at the base of the workflow designer.
2. Drag a **Code** activity from the **Toolbox** and drop it underneath the *transactionScopeActivity1*.
3. In the **Properties** window for *codeActivity1* set the **Properties** as follows:

* **(Name)** –*FinishWorkflow*
* **ExecuteCode** – select *OnFinishWorkflow* from the dropdown list

1. Your workflow should look like this at the end of this step:



## Task 3 – Run the Application

1. Open a command window and change directory to *“C:\WF\WF 3.5 Labs\Lab08\Exercise1”*
2. Run the batch file *“dumptables.bat”.* This script dumps the contents of *ChequeAccount* and *SavingsAcount* tables. It should display the following before the workflow is run:

Checking balance

---------------------

100.0000

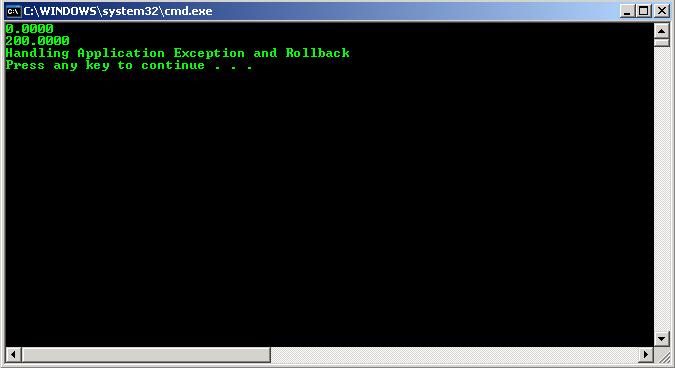
1 row affected)

Savings balance

---------------------

100.0000

1. Build and run the solution by pressing **CTRL + F5**.
2. You should see the following.



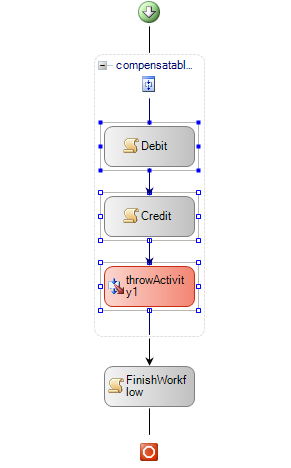
1. Press the **ENTER** key twice.
2. Run the batch file *“dumptables.bat”* again. You should see that the tables have not been modified since the exception caused a rollback because the **TransactionScopeActivity** is atomic.
3. Now go back to *Workflow1.cs*
4. Select the *throwActivity1* activity and change the enabled property to *“false”.*
5. Go to your Command window and run the script **setupdatabase.bat** to clean your database.
6. Run the application again by pressing **CTRL+F5** in Visual Studio 2005
7. You can observe that the rollback of the tables does not occur now by running *“dumptables.bat”* after the application completes.

# Exercise 2 – Compensation for Long-running Processes

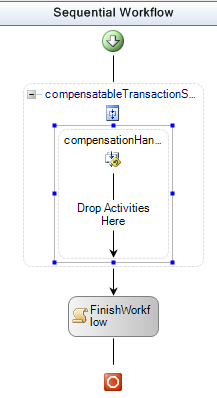
In this exercise, you will learn how to author compensation in Microsoft Windows Workflow Foundation. Compensation applies to completed transactions (Atomic or Long Running) and is the process of logically undoing the completed transactions (either Atomic or Long Running) in case of any business exceptions. For example this could be reversing a purchase order if there are insufficient funds on a credit card to complete the order.

## Task 1 – Modify the Workflow Project

1. Open the Visual Studio 2005 solution you have created for exercise 1 by navigating to *C:\WF\WF 3.5 Labs\Lab08\Exercise1\AtomicTransactionsWorkflow* if it is not already open.
2. Open the workflow *Workflow1.cs* by double clicking it in the **Solution Explorer**
3. To use the compensation capabilities of Windows Workflow Foundation we need to replace our **TransactionScopeActivity** *transactionScopeActivity1* with a**CompensatableTransactionScopeActivity.** Select the 3 activities *Debit*, *Credit* and *throwActivity1* by clicking on each while holding down the **CTRL** key. **Right click** on *Debit* and select **Copy** from the context menu.
4. Select *transactionScopeActivity1* and delete it by either selected **Delete** from the context menu or by pressing the **Delete key**.
5. Drag a **CompensatableTransactionScopeActivity** from the **Toolbox** and drop it on the workflow surface just above the **FinishWorkflow** activity.
6. **Right click** within *compensatableTransactionScopeActivity* and select **Paste** from the context menu to add our 3 activities we copied earlier, *Debit*, *Credit* and *throwActivity1*.



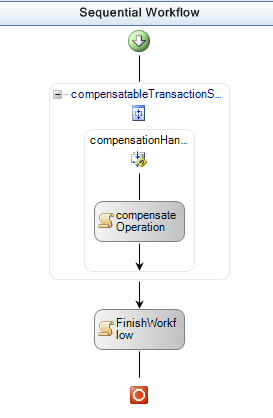
1. Right click on the *compensatableTransactionScopeActivity1* on the designer and select *‘****View Compensation Handler****”* from the context menu. The designer view should look like this:



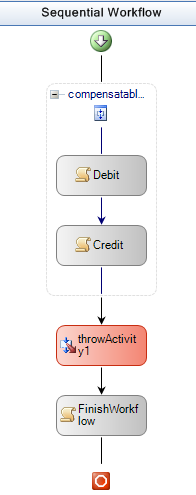
1. Add a **Code** activity onto the area labeled *“Drop Activities Here”.* In the **Properties** window for *codeActivity1* set the **Properties** as follows:

* **(Name)** –*compensateOperation*
* **ExecuteCode** – select *OnCompensation* from the dropdown list.

Your designer view should look like this:

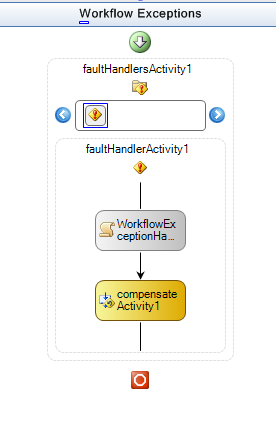


1. Go the **CompensatableTransactionScope** view of *compensatableTransactionScopeActivity1* by **right clicking** on *compensatableTransactionScopeActivity1* and selecting*“****View CompensatableTransactionScope****.”*
2. Move *throwActivity1* in the workflow so that it occurs after the *compensatableTransactionScopeActivity1* activity completes and set the **Enabled** property to *True*. Your workflow should look like this:



1. Switch to **View Faults** by clicking the third tab on the bottom of the design surface and add a **Compensate** activity to the exception handler just after the *WorkflowExceptionHandler* activity.
2. Set the **Properties** of the *compensate1* activity as follows from the drop down list

* **TargetActivityName**: *compensatableTransactionScopeActivity1*



## Task 2 - Test the application

1. Open a command window and navigate to *C:\WF\WF 3.5 Labs\Lab08\Exercise1.* Run the script *“setupdatabase.bat”* to refresh the database.
2. Build and run the application by pressing **CTRL+F5** in Visual Studio 2005.
3. Notice that even though the transaction was committed as seen in the output, the compensation operation logically restores the tables to the original values. This can be verified by running the *dumptables.bat* script again to display the contents of the tables. You should be able to see the transaction has been reversed by the values displayed.

# Lab Summary

Our objective in this lab was to introduce the concepts and topics related with transactions. We had two exercises:

The first exercise uses a **TransactionScopeActivity**. It has three activities within the **TransactionScopeActivity**, one for a debit operation, the other for a credit operation and the last one is simply there to initiate a rollback by throwing an exception.

The second exercise builds on the first. The throw activity is also taken out of the **CompensatableTransactionScopeActivity** and added to the workflow after the **CompensatableTransactionScopeActivity** for demonstrating that the compensation logic added to the **CompensatableTransactionScopeActivity** can be triggered as a part of the business logic from within the workflow with a compensation activity. The workflow starts, and enters the **CompensatableTransactionScopeActivity** (which contains the credit and debit operations) and exits the **CompensatableTransactionScopeActivity**. An exception is thrown after the **CompensatableTransactionScopeActivity** and the compensation is invoked as a part of the exception handler of the workflow. The compensation logic just reverses the operations done during the **CompensatableTransactionScopeActivity**.