

Windows Workflow Foundation

Hands-On Lab

Lab Manual

Lab 09 – Modifying Workflow Models at Runtime with Workflow Change 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, 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](#_Toc218634391)

[Objectives 1](#_Toc218634392)

[System Requirements 1](#_Toc218634393)

[Setup 1](#_Toc218634394)

[Physical Folder Structure 1](#_Toc218634395)

[Code Snippets 2](#_Toc218634396)

[Starting Material 2](#_Toc218634397)

[Acronyms Used in this Lab 2](#_Toc218634398)

[Scenario 2](#_Toc218634399)

[Exercise 1 – WorkflowChange from an Existing Workflow 3](#_Toc218634400)

[Task 1 – Create a new Workflow Project 3](#_Toc218634401)

[Task 2 – Create the First Workflow 3](#_Toc218634402)

[Task 3 – Add a Parameter to the Workflow 9](#_Toc218634403)

[Task 4 – Create the Second Workflow 12](#_Toc218634404)

[Task 5 – Add the Program Code 13](#_Toc218634405)

[Task 6 – Run the Workflow 14](#_Toc218634406)

[Exercise 2 – WorkflowChange from the Host on an Executing Workflow 15](#_Toc218634407)

[Task 1 – Modify the Workflow 15](#_Toc218634408)

[Task 2 – Modify workflow at Runtime 17](#_Toc218634409)

[Task 4 – Run the Workflow 19](#_Toc218634410)

[Exercise 3 – Changing Conditions in an IfElse Activity of a Running Workflow 21](#_Toc218634411)

[Task 1 – Create a new Workflow Project 21](#_Toc218634412)

[Task 2 – Modify project to run DynamicRulesWorkflow 22](#_Toc218634413)

[Task 3 – Modify the project to change the conditions of the DynamicRulesWorkflow 25](#_Toc218634414)

[Task 4 – Run the Code 27](#_Toc218634415)

[Lab Summary 28](#_Toc218634416)

# Overview

Estimated time to complete this lab: **60 minutes.** The last two exercises are optional.

The project files for this lab are in the *C:\WF\WF 3.5 Labs\Lab09* folder.

## Objectives

The objective of this lab is to demonstrate how to author workflow through code, and apply dynamic changes to a workflow at run-time. After completing this lab, you will be able to:

Make dynamic changes to the workflow while it is running, from the host and from the workflow.

Change a declarative condition from the host while the instance is running.

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

# Exercise 1 – WorkflowChange from an Existing Workflow

In this exercise, you will make dynamic changes to a workflow while it is running, from within the workflow itself. The workflow will be able to create activities later in the process based on some external input or learning. You will build this sample from scratch and create a workflow which receives a purchase order request. The workflow creates the PO for amounts less than $1000. For amounts greater than $1000, the workflow dynamically invokes another workflow to approve the purchase order.

## Task 1 – Create a new Workflow Project

1. Open Microsoft Visual Studio 2005 by going to **Start | All Programs | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005**.
2. In Visual Studio 2005, select **File** | **New** | **Project**.
3. In the **New Project** dialog box, expand **Visual C# | Workflow** in the **Project** types tree on the left.
4. Select the template **Sequential Workflow Console Application** from the **Visual Studio installed templates** and type the following for the name and location and click **OK**.

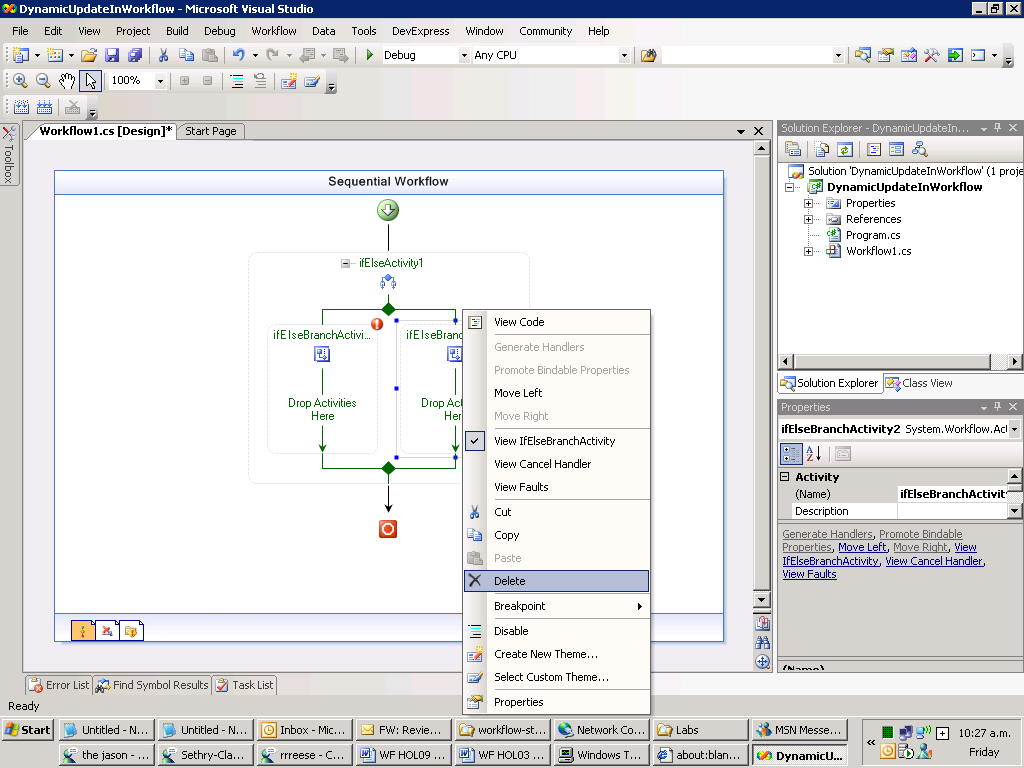
**Name:** *DynamicUpdateInWorkflow.*

**Location:** *C:\WF\WF 3.5 Labs\Lab09.*

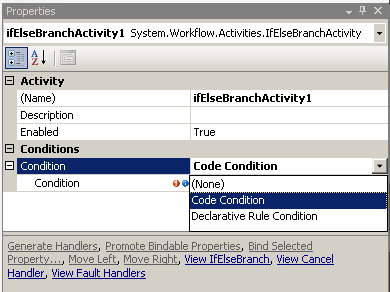
## Task 2 – Create the First Workflow

You will start by creating the workflow which creates a purchase order and checks to see if the amount supplied is less than or greater than $1000.

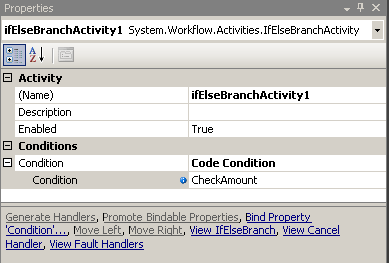
1. Double click on *Workflow1.cs* from **Solution Explorer** if it is not already displayed.
2. From the **Toolbox**, select an **IfElse** activity and drop it into the area marked **“Drop Activities** **to create a Sequential Workflow”**.
3. Highlight the branch marked *ifElseBranchActivity2*, right click on this branch and choose **Delete**.



1. Right Click select **Properties** on the *ifElseBranchActivity1* branch to bring up the **Properties** window.
2. Select *Code Condition* from the drop-down list in the **Condition** box in the **Properties** window.



1. Click on the plus sign to the left of the **Condition** box and type *CheckAmount* in the second **Condition** property text box.



1. In the code view Insert the gray highlighted code into the *CheckAmount* method.

(Snippet: “WFLab09\_Ex01\_Task02\_CheckAmount”).

public void CheckAmount(object sender, ConditionalEventArgs e)

{

if (Amount >= 1000)

e.Result = true;

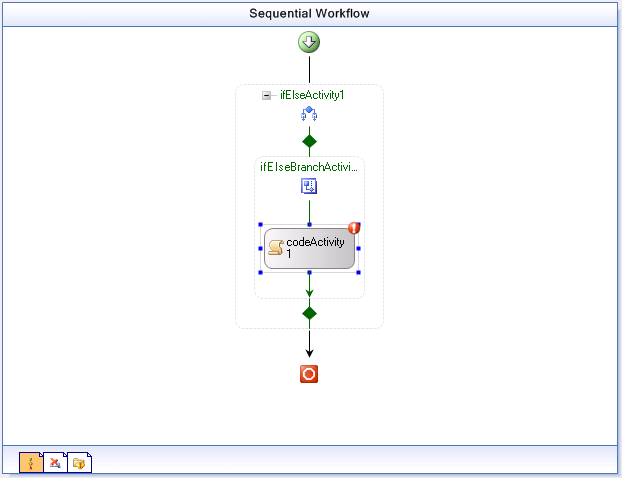
else

e.Result = false;

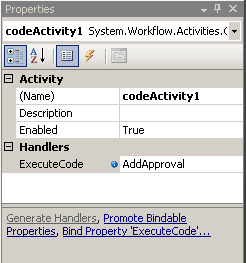
Console.WriteLine("\tCheckAmount : amount \'{0}\' result \'{1}\'", Amount, e.Result);

}

1. Click the *Workflow1.cs[Design]* tab to return to the Sequential Workflow design view.
2. From the **Toolbox**, select a **Code** activity and drop it into the area marked “**Drop Activities here”**, as pictured below.



1. Right Click select **Properties** to bring up the **Properties** window.
2. Type *AddApproval* into the **ExecuteCode** box.



In this **Code** activity, we will create a new **InvokeWorkflow** activity that will invoke the second workflow (which we will create in Task 4) and perform a dynamic update to add it to the currently running workflow instance. Add the following code in the *AddApproval()* method.

(Snippet: "WFLab09\_Ex01\_Task02\_AddApproval").

InvokeWorkflowActivity invokeNewStepWorkflow = new InvokeWorkflowActivity();

// use WorkflowChanges class to author dynamic change

WorkflowChanges changes = new WorkflowChanges(this);

// setup to invoke NewStepWorkflow type

Type type = typeof(DynamicUpdateInWorkflow.Workflow2);

invokeNewStepWorkflow.Name = "Workflow2";

invokeNewStepWorkflow.TargetWorkflow = type;

// insert invokeNewStepWorkflow in ifElseApproval

// transient activity collection

CompositeActivity ifElse = changes.TransientWorkflow.Activities["ifElseActivity1"] as CompositeActivity;

CompositeActivity branch1 = ifElse.Activities["ifElseBranchActivity1"] as CompositeActivity;

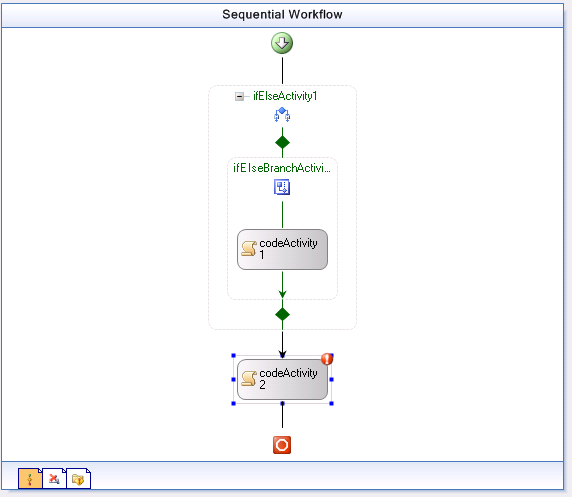
branch1.Activities.Add(invokeNewStepWorkflow);

// apply transient changes to instance

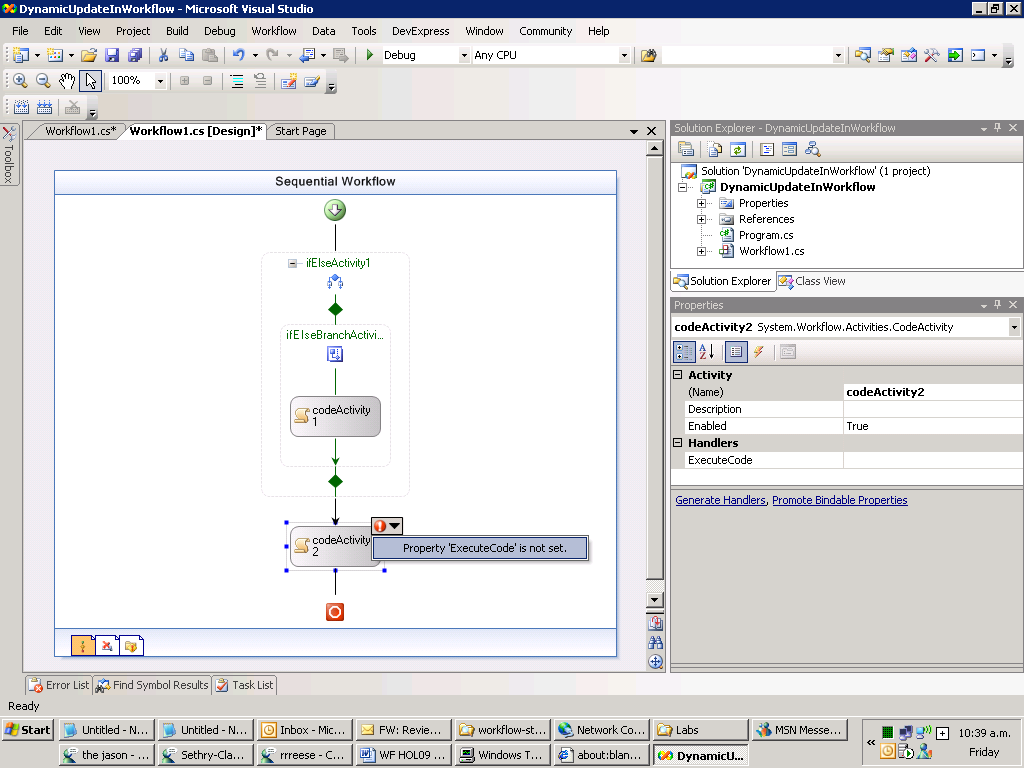
this.ApplyWorkflowChanges(changes);

Console.WriteLine("\tAdded a new step from within workflow");

1. Click the *Workflow1.cs[Design]* tab to return to the Sequential Workflow design view.
2. From the **Toolbox**, select a **Code** activity and drop it underneath the **ifElse** activity right at the bottom of the workflow.



1. Click the white exclamation mark in a red circle and select **Property 'ExecuteCode' is not set** to bring up the **Properties** window.

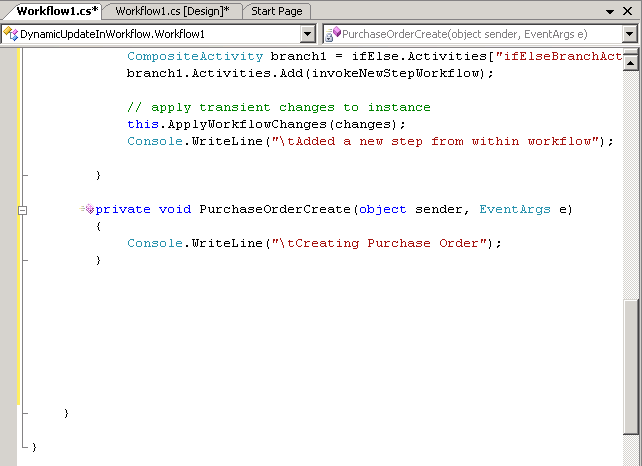


1. Type *PurchaseOrderCreate* into the **ExecuteCode** box.
2. Insert the following code inside the **PurchaseOrderCreate** method:

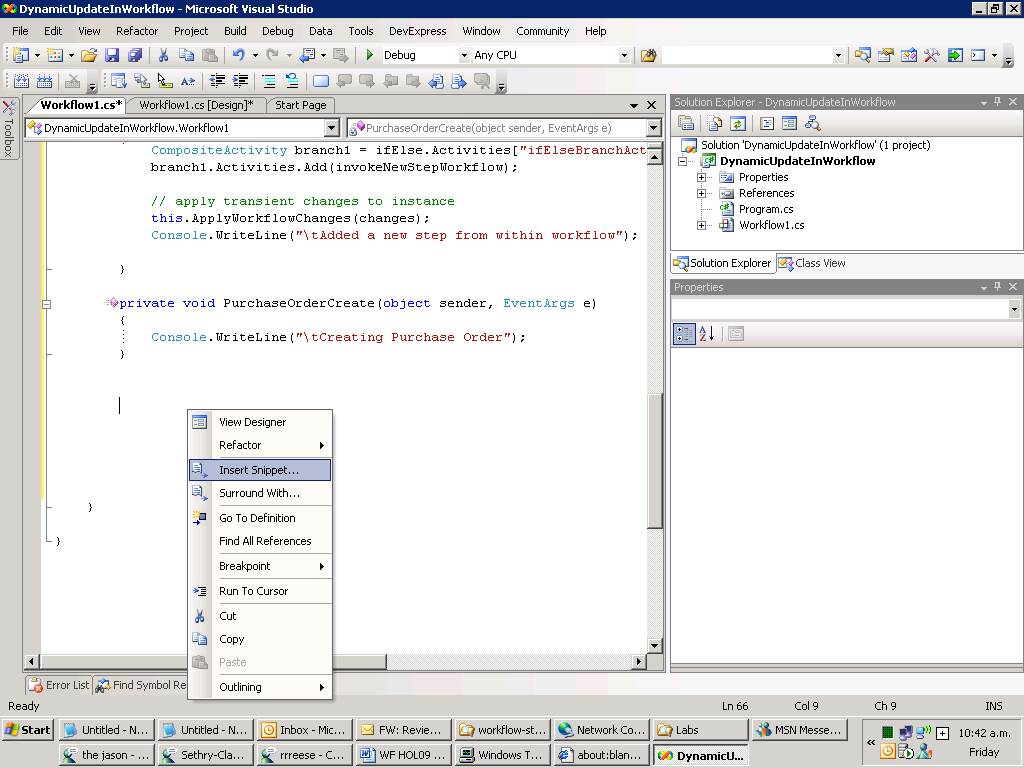
Console.WriteLine("\tCreating Purchase Order");

## Task 3 – Add a Parameter to the Workflow

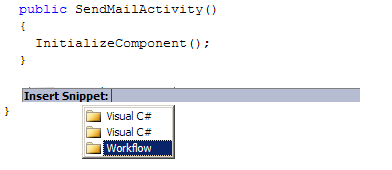
1. Place the insertion pointer after the *PurchaseOrderCreate* method and press enter several times to create some space.



1. Right Click on the space you created and select **Insert Snippet.**



1. Select **Workflow**.



1. Select **DependencyProperty – Property**.

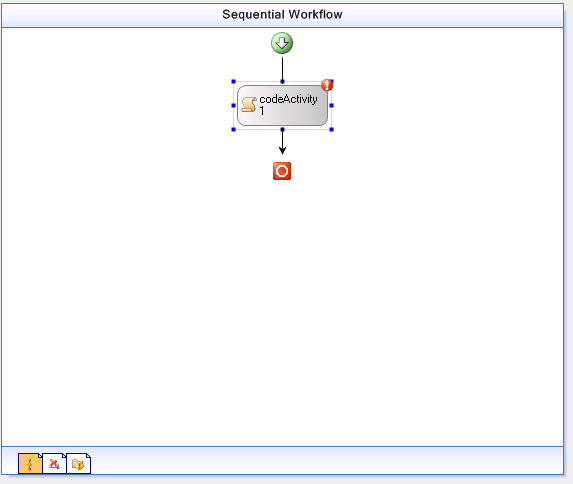
New Picture (3)

1. The text area will be populated with some boilerplate code that you can customize.
2. Change **MyProperty** to *Amount*, notice that replacements are made from ‘*MyProperty’* to ‘*Amount*”.
3. Press **Tab**, the contents of the return type will be highlighted enter *System.Int32* notice that replacements are made from ‘**string’** to ‘**System.Int32’** then press **Enter** to finish entering the snippet.

## Task 4 – Create the Second Workflow

In this task, you will create the second workflow to handle approvals for purchase orders greater than $1000.

1. From **Solution Explorer** right-click the project *DynamicUpdateInWorkflow* and select **Add | New Item**.
2. Select **Sequential Workflow (code)** and accept the default file name *Workflow2.cs* and click **Add**.
3. From the **Toolbox**, select a **Code** activity and drop it in the area marked **“Drop Activities to create a Sequential Workflow”**.



1. Double-click on the **Code** activity and add the following code to handler *codeActivity1\_ExecuteCode*:

Console.WriteLine("\tAdditional Approver Executing");

## Task 5 – Add the Program Code

1. In the **Solution Explorer** double-click on *Program.cs* to open it in the code window.
2. Now we need to create a simple console host that will start our main workflow and pass the required parameter to it.
3. Replace the following lines:

WorkflowInstance instance = workflowRuntime.CreateWorkflow(typeof(DynamicUpdateInWorkflow.Workflow1));

instance.Start();

with this code.

(Snippet: "WFLab09\_Ex01\_Task05\_TestWorkflow").

//start PO approval workflow with purchase less than $1000

Console.WriteLine("Workflow test 1 (purchase less than $1000) :");

int workflow1Amount = 750;

Dictionary<string, object> workflow1NamedValues = new Dictionary<string, object>();

workflow1NamedValues.Add("Amount", workflow1Amount);

WorkflowInstance workflow1Instance = workflowRuntime.CreateWorkflow(typeof(DynamicUpdateInWorkflow.Workflow1), workflow1NamedValues);

workflow1Instance.Start();

waitHandle.WaitOne();

//waiting for the workflow to complete so the console output makes sence

System.Threading.Thread.Sleep(new TimeSpan(0, 0, 0, 10, 0));

Console.Write("\r\n\r\n");

// start PO approval workflow with purchase greater than $1000

Console.WriteLine("Workflow test 2 (purchase greater than $1000) :");

int workflow2Amount = 1200;

Dictionary<string, object> workflow2NamedValues = new Dictionary<string, object>();

workflow2NamedValues.Add("Amount", workflow2Amount);

WorkflowInstance workflow2Instance = workflowRuntime.CreateWorkflow(typeof(DynamicUpdateInWorkflow.Workflow1), workflow2NamedValues);

workflow2Instance.Start();

waitHandle.WaitOne();

//waiting for the workflow to complete so the console output makes sence

System.Threading.Thread.Sleep(new TimeSpan(0,0,0,10,0));

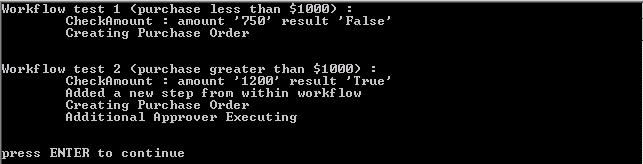
Console.Write("\r\n\r\n");

Console.WriteLine("press ENTER to continue");

Console.ReadLine();

## Task 6 – Run the Workflow

1. Build the Solution.
2. Press **CTRL + F5** to run the solution without debugging.
3. You should see output resembling the following.

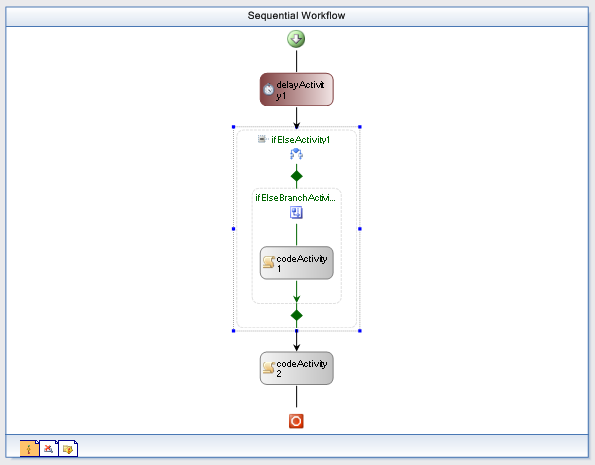


# Exercise 2 – WorkflowChange from the Host on an Executing Workflow

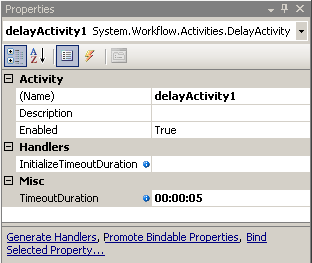
Starting with the workflow created in Exercise 1 you will locate a running instance of the workflow and make a change to the workflow from the host *program.cs*. You will modify *ifElseActivity1* to insert an **InvokeWorkflow** to the host so that all purchase order requests will require approval.

## Task 1 – Modify the Workflow

1. Double Click on *Workflow1* to bring up the designer.
2. After the **Green Arrow** but before *ifElseActivity1* drop a **DelayActivity**, Your Sequential workflow should look like this:



1. Right click on *delayActivity1,* select **Properties**, and set the **TimeOutDuration** property to ‘*00:00:05*’.



## Task 2 – Modify workflow at Runtime

1. Double Click on *Program.cs* to bring it into the code window.
2. Add the following **using** statements.

(Snippet: "WFLab09\_Ex02\_Task02\_UsingStatements").

using System.Workflow.ComponentModel;

using System.Workflow.Activities;

1. Add the following method to the *Program* class below the *Main* method, this code will modify the running workflow .

(Snippet: "WFLab09\_Ex02\_Task02\_ModifyWorkflowFromHost").

static void modifyWorkflowFromHost(WorkflowInstance workflowInstance)

{

if (!workflowUpdated)

{

workflowUpdated = true;

Activity workflowInstanceDefinition = workflowInstance.GetWorkflowDefinition();

WorkflowChanges workflowChangesToMake = new WorkflowChanges(workflowInstanceDefinition);

// remove ifelse

Console.WriteLine("\tHost change to always require approval for");

Console.WriteLine("\tInstanceId: " + workflowInstance.InstanceId);

CompositeActivity ifelse = workflowChangesToMake.TransientWorkflow.Activities["ifElse1Activity"] as CompositeActivity;

workflowChangesToMake.TransientWorkflow.Activities.Remove(ifelse);

// setup to invoke NewStepWorkflow type

InvokeWorkflowActivity invokeNewStepWorkflow = new InvokeWorkflowActivity();

invokeNewStepWorkflow.Name = "AddNewStepWorkflow";

invokeNewStepWorkflow.TargetWorkflow = typeof(DynamicUpdateInWorkflow.Workflow2);

// insert approval workflow

workflowChangesToMake.TransientWorkflow.Activities.Insert(1, invokeNewStepWorkflow);

// apply transient changes to instance

workflowInstance.ApplyWorkflowChanges(workflowChangesToMake);

} else {

Console.WriteLine("\tWorkflow has already been updated by the Host");

}

}

#### Add the following delegates to the *Main* method after the existing *WorkflowTerminated* delegate.

#### (Snippet: "WFLab09\_Ex02\_Task02\_WorkflowDelegates").

workflowRuntime.WorkflowIdled +=

delegate(object sender, WorkflowEventArgs e) {

//idling this workflow so we can modify it

e.WorkflowInstance.Suspend("suspending to modify workflow");

modifyWorkflowFromHost(e.WorkflowInstance);

e.WorkflowInstance.Resume();

};

workflowRuntime.WorkflowSuspended +=

delegate(object sender, WorkflowSuspendedEventArgs e) {

string reason = e.Error;

string workflowID = e.WorkflowInstance.InstanceId.ToString();

Console.WriteLine("\tWorkflow \'{0}\' Suspended, reason \'{1}\'", workflowID, reason);

};

workflowRuntime.WorkflowResumed +=

delegate(object sender, WorkflowEventArgs e) {

string workflowID = e.WorkflowInstance.InstanceId.ToString();

Console.WriteLine("\tWorkflow \'{0}\' Resumed", workflowID);

};

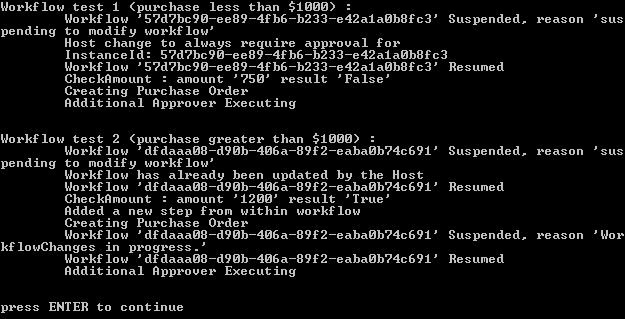
1. Inside the *Program* class add a private static member, This will let us know when we have modified the workflow.

static bool workflowUpdated = false;

The above step will now invoke the method *modifyWorkflowFromHost*when the *Workflow1* goes idle. The above method *modifyWorkflowFromHost*will replace *ifElseActivity1* activity by the mandatory approval step in the form of invoking a workflow (defined in *Workflow2.cs*).

## Task 4 – Run the Workflow

1. Build the solution.
2. Press **CTRL + F5** to run the solution without debugging.
3. The following output should be displayed:



# Exercise 3 – Changing Conditions in an IfElse Activity of a Running Workflow

In this exercise, you will locate a running instance of the workflow and make a change to the workflow. The workflow has two branches. Upon receipt of a value less than $10,000 it will execute the branch labeled *getManagerApproval*, If the value exceeds $10,000 it will execute the branch labeled *getVPApproval*. You will create the workflow and change the condition dynamically, so that it alters who approves the purchase order.

## Task 1 – Create a new Workflow Project

1. Open Microsoft Visual Studio 2005 by going to **Start | All Programs | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005**.
2. In Visual Studio 2005, select **File** | **New** | **Project**.
3. In the **New Project** dialog box, expand **Visual C# | Workflow** in the **Project** types tree on the left.
4. Select the template **Sequential Workflow Console Application** from the **Visual Studio installed templates** and type the following for the name and location and click **OK**.

**Name:** *DynamicUpdateChangingRules*

**Location:** *C:\WF\WF 3.5 Labs\Lab09\*

1. Navigate using Explorer to the directory:

*C:\WF\WF 3.5 Labs\Lab09\Resources\Exercise3*

1. Copy **All Files** to:

*C:\WF\WF 3.5 Labs\Lab09\DynamicUpdateChangingRules\DynamicUpdateChangingRules*

1. Right Click on the *DynamicUpdateChangingRules* project file.
2. Select **Add | Existing Item**…
3. Browse to and select the items you copied in step 6.
4. Right Click on *Workflow1.cs*and select **Delete**.

## Task 2 – Modify project to run DynamicRulesWorkflow

1. Double click on *Program.cs* to bring it into the code editor.
2. Replace the lines.

WorkflowInstance instance = workflowRuntime.CreateWorkflow(typeof(DynamicUpdateChangingRules.Workflow1));

Instance.Start();

waitHandle.WaitOne();

1. With these lines – this will make the *DynamicRulesWorkflow* execute with the parameters provided.

(Snippet: "WFLab09\_Ex03\_Task02\_MainLogic").

// The "Amount" parameter is used to determine which branch of the IfElse should be executed

// a value less than 10,000 will execute branch 1 - Get Manager Approval; any other value will execute branch 2 - Get VP Approval

int workflowAmount = 9000;

Dictionary<string, object> workflowNamedValues = new Dictionary<string, object>();

workflowNamedValues.Add("Amount", workflowAmount);

WorkflowInstance workflowInstance = workflowRuntime.CreateWorkflow(typeof(DynamicUpdateChangingRules.DynamicRulesWorkflow), workflowNamedValues);

workflowInstance.Start();

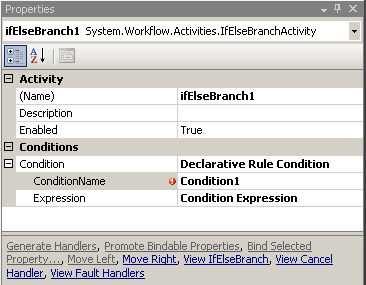
waitHandle.WaitOne();

Console.Write("\r\n\r\n");

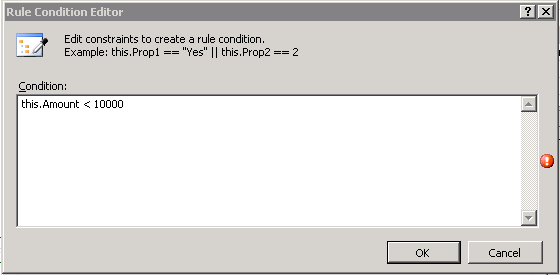
Console.WriteLine("press ENTER to continue");

Console.ReadLine();

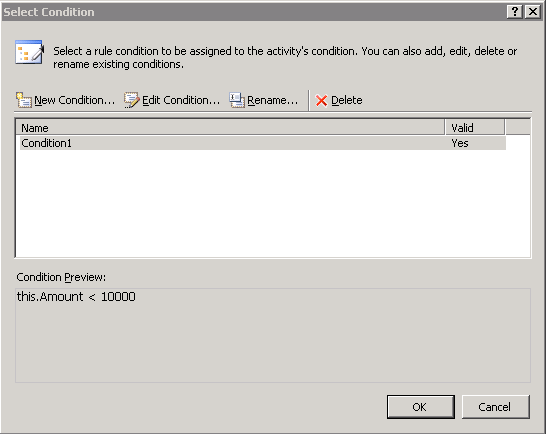
1. Double Click on *DynamicRulesWorkflow* to bring it into the designer.
2. Right Click on *ifElseBranch1* and select **Properties** to bring up its properties.
3. Click the plus beside **Condition** and Click the elipses beside **ConditionName**.



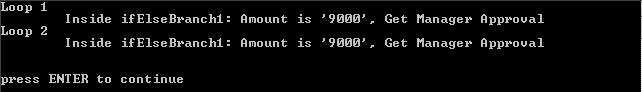
1. In the dialog click **New…** and enter the following rule condition.



1. Click **OK**.
2. Ensure that the name of the condition is *Condition1* as pictured below.



1. Press **OK**.
2. Press **F5**, and you should see the following text outputted to the console.



## Task 3 – Modify the project to change the conditions of the DynamicRulesWorkflow

1. Double click on *Program.cs* to bring it into the code window
2. Add the following **using** **statements** at the top of the file.

(Snippet: "WFLab09\_Ex03\_Task03\_UsingStatements").

using System.Workflow.ComponentModel;

using System.Workflow.Activities.Rules;

using System.CodeDom;

1. Inside the *Program* class add the following private static member, this will let us know when we have modified the rules.

static bool rulesChanged= false;

1. Add the following method *changeRulesForWorkflow* into the *Program* class below the *Main* method.

(Snippet: "WFLab09\_Ex03\_Task03\_ChangeRules").

static void changeRulesForWorkflow(WorkflowInstance workflowInstance)

{

if (!rulesChanged) {

rulesChanged = true;

// our new validation amount

Int32 newAmount = 8000;

Console.WriteLine("\tDynamically change approved amount to {0}", newAmount);

// Dynamic update of order rule

WorkflowChanges workflowchanges = new WorkflowChanges(workflowInstance.GetWorkflowDefinition());

CompositeActivity transient = workflowchanges.TransientWorkflow;

RuleDefinitions ruleDefinitions = (RuleDefinitions)transient.GetValue(RuleDefinitions.RuleDefinitionsProperty);

RuleConditionCollection conditions = ruleDefinitions.Conditions;

RuleExpressionCondition condition1 = (RuleExpressionCondition)conditions["Condition1"];

(condition1.Expression as CodeBinaryOperatorExpression).Right = new CodePrimitiveExpression(newAmount);

// Apply our changes to the workflow

workflowInstance.ApplyWorkflowChanges(workflowchanges);

} else {

Console.WriteLine("\tRules for Workflow already changed");

}

}

1. Modify the *Main* method by pasting the following event handlers after the existing *WorkflowTerminated* handler.

(Snippet: "WFLab09\_Ex03\_Task03\_WorkflowDelegates").

workflowRuntime.WorkflowIdled +=

delegate(object sender, WorkflowEventArgs e) {

//idling this workflow so we can modify it

e.WorkflowInstance.Suspend("suspending to modify IfElse1");

changeRulesForWorkflow(e.WorkflowInstance);

e.WorkflowInstance.Resume();

};

workflowRuntime.WorkflowSuspended +=

delegate(object sender, WorkflowSuspendedEventArgs e) {

string reason = e.Error;

string workflowID = e.WorkflowInstance.InstanceId.ToString();

Console.WriteLine("\tWorkflow \'{0}\' Suspended, reason \'{1}\'", workflowID, reason);

};

workflowRuntime.WorkflowResumed +=

delegate(object sender, WorkflowEventArgs e) {

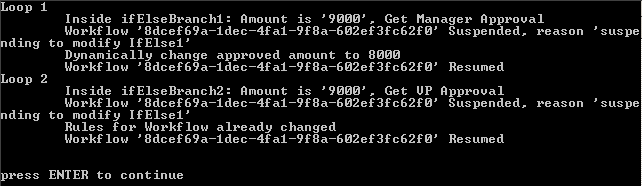
string workflowID = e.WorkflowInstance.InstanceId.ToString();

Console.WriteLine("\tWorkflow \'{0}\' Resumed", workflowID);

};

## Task 4 – Run the Code

1. Run the solution without debugging by pressing **CTRL + F5**.
2. The following console output should be displayed.



1. With an approved amount of 10,000 an order amount of 9000 requires a Manager Approval as the first branch is executed.
2. After the host changes the approved amount to 8000, running the same order amount of 9000 should require VP Approval.

# Lab Summary

In this lab, you learned how to apply dynamic changes to a workflow at run-time.

In this lab you performed the following exercises.

Made dynamic changes to a workflow while it is running, from within the workflow itself.

Made dynamic changes to the workflow while it is running, from the host

Changing a declarative condition from the host while the instance is running.