PNMSOFT
Business Processes in Motion™

PNMsoft Knowledge Base

Sequence Best Practices

Sequence and ASP.NET Applications

Decmeber 2013
Product Version 7.x

PNMsoft UK 38 Clarendon Road Watford Hertfordshire WD17 1JJ

Tel: +44(0)192 381 3420 • Email: info@pnmsoft.com • Website: www.pnmsoft.com

Microsoft Partner
Gold Application Development

TABLE OF CONTENTS

General Document Information

Purpose

This document is designed to instruct you on how to integrate Sequence Flowtime Workflow and Task execution with an independent ASP.NET web application, taking advantage of the Sequence .NET API. This document provides step-by-step information on how to set this up.

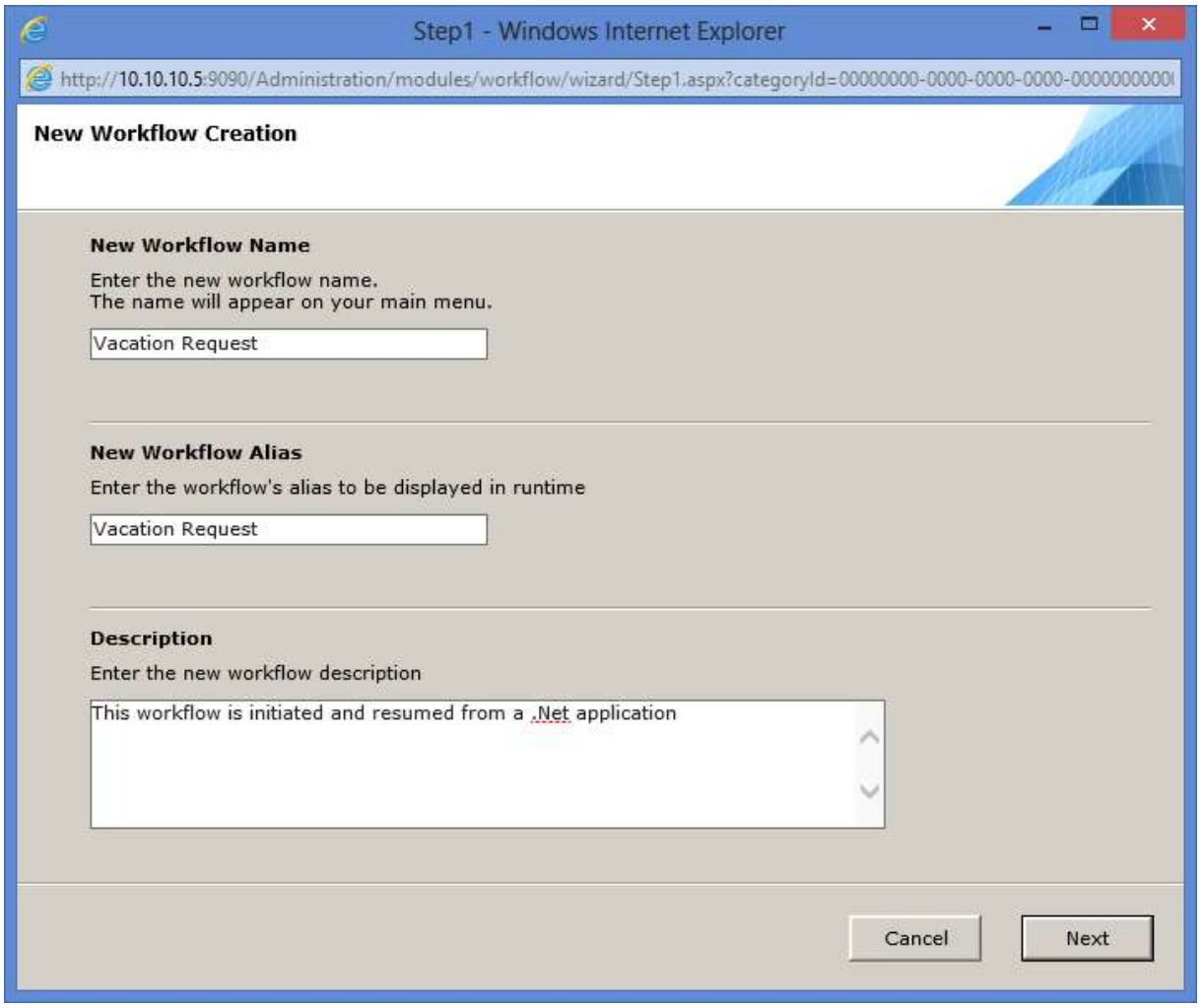
Prerequisites

- Knowledge:
  - Sequence App Studio
  - .NET development using C#
- Environments:
  - A Sequence server environment
  - Visual Studio

Note: It is highly advised to involve standard development practices such as error handling while utilising these code samples in any software development project.

Step 1: Creating a Request Process

1. Open Sequence Administration and create a new process. Give your process a meaningful name (e.g. Vacation Request).

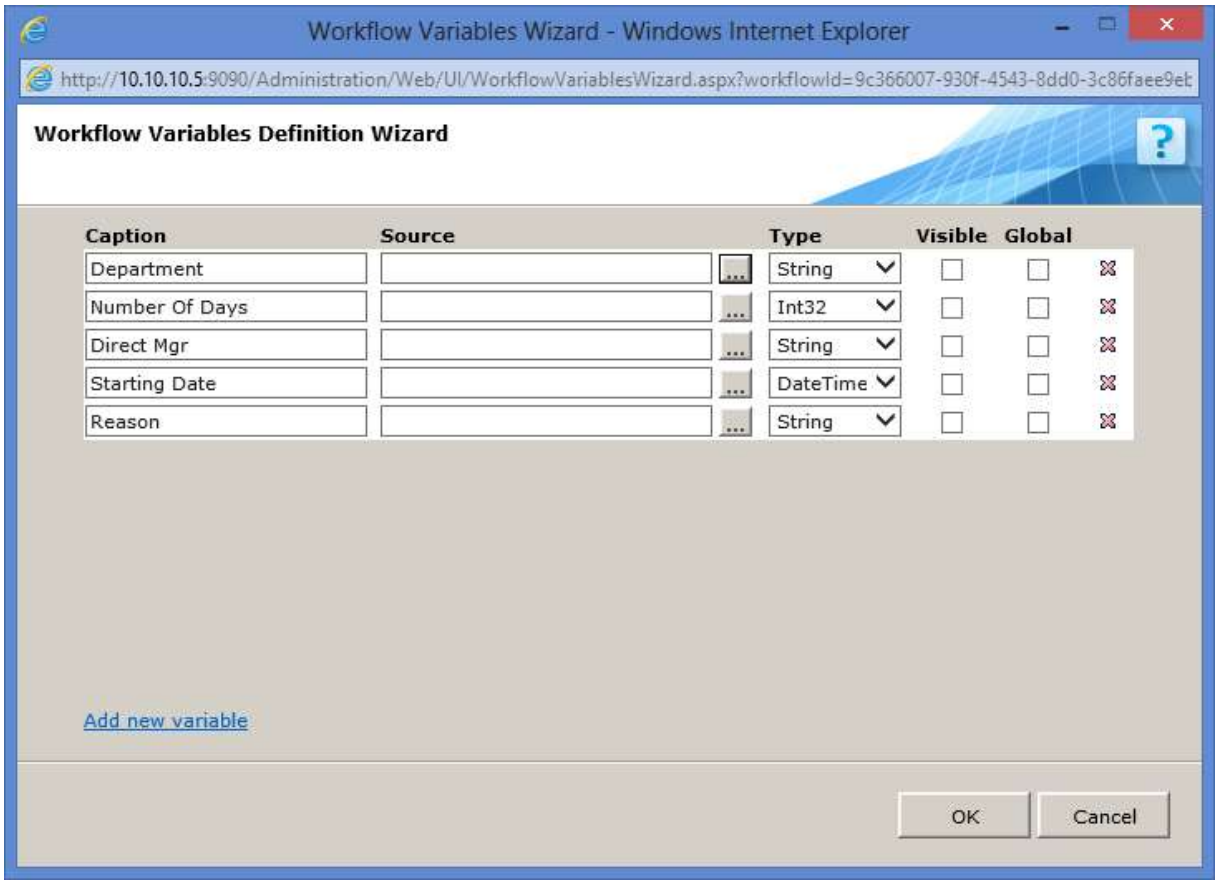


Create New Workflow

Step 2: Creating the Workflow Variables

1. Once the Sequence App Studio is open, click **Variables** in the workflow properties window.
2. Add these variables to your workflow (the source should be left empty):

№	Name	Data type
1	Department	String
2	Number Of Days	int
3	Direct Mgr	String
4	Starting Date	DateTime
5	Reason	String

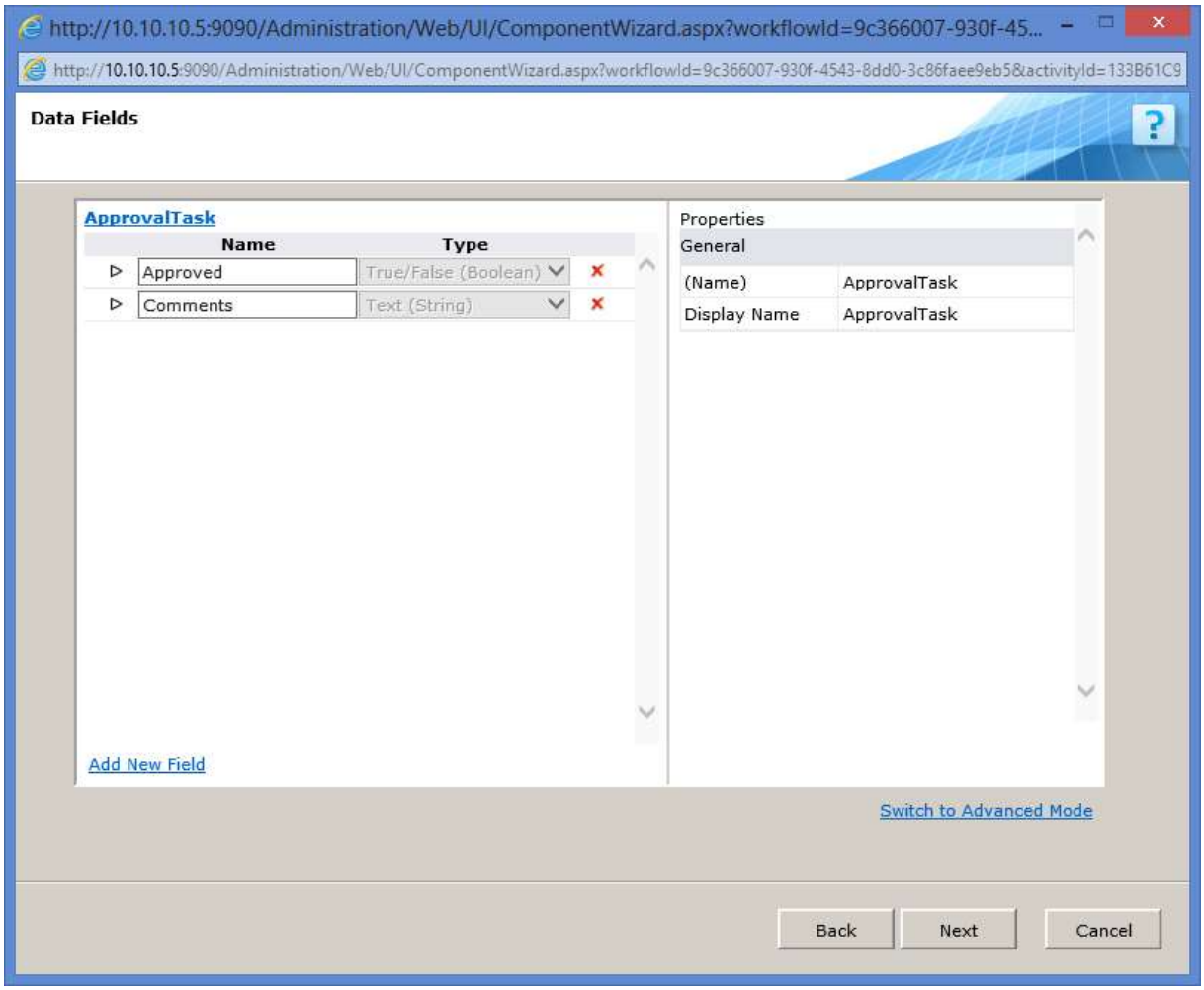


Workflow Variables Wizard

Step 3: Creating an Approval Task

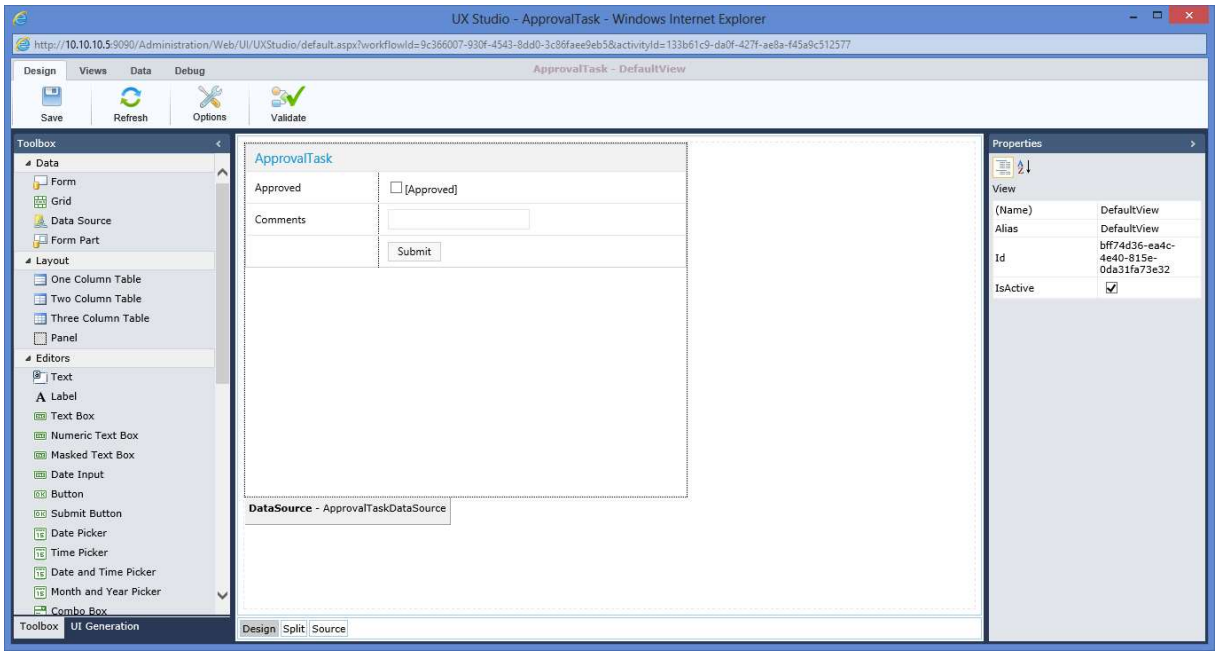
1. Click **Task Activity** from the Activity toolbox.
2. Double-click the Task to define the activity details.
3. Give the activity a meaningful name, e.g. Approval Task.
4. Select **First define data** and keep the Fast Track checkbox checked.
5. In the Data Fields screen, add the following fields:

№	Name	Type
1	Approved	True/False (Boolean)
2	Comments	Text (String)



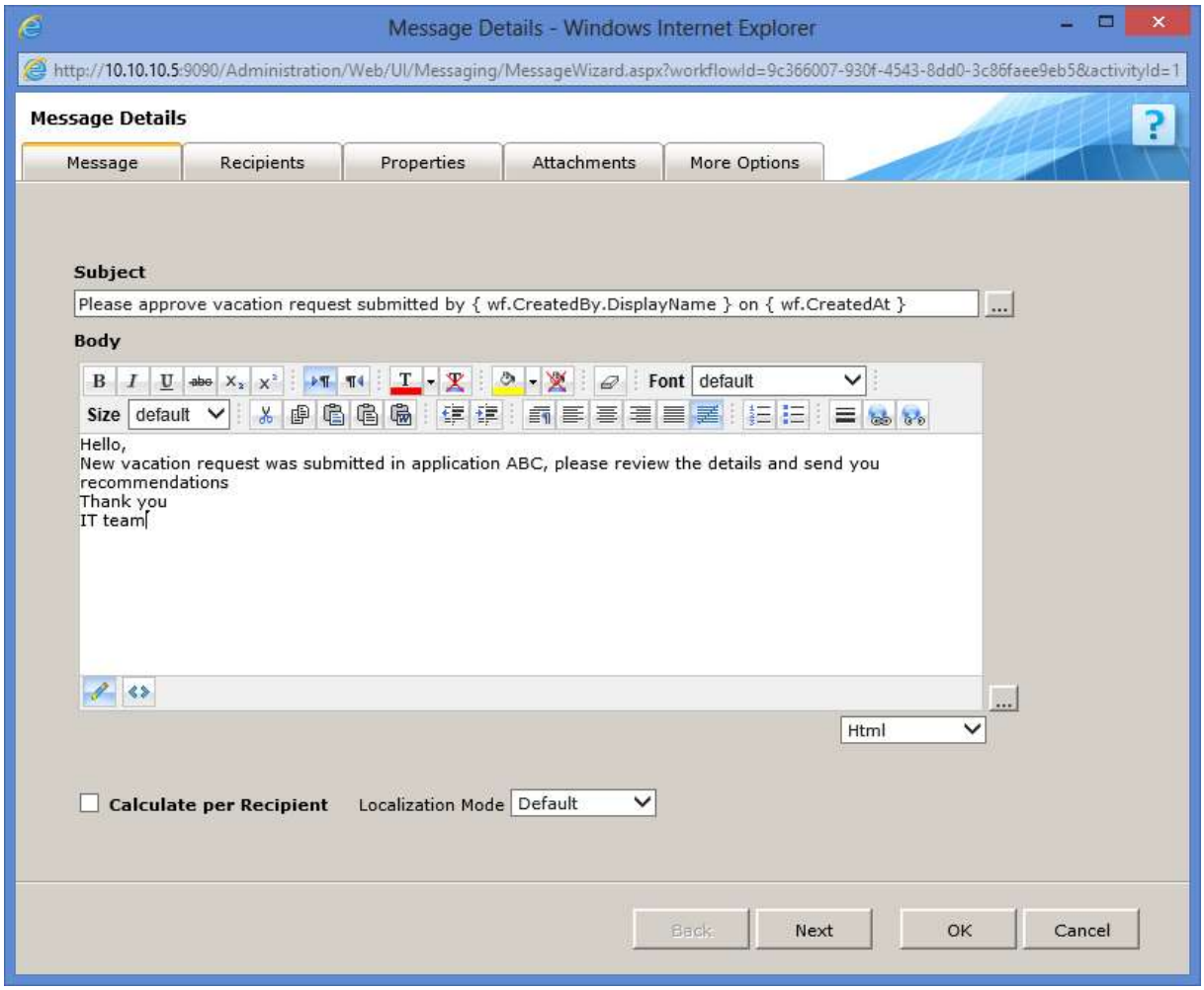
Data Fields Screen

6. Click **Next**, select **Form,** and click **Finish**. The form is created.



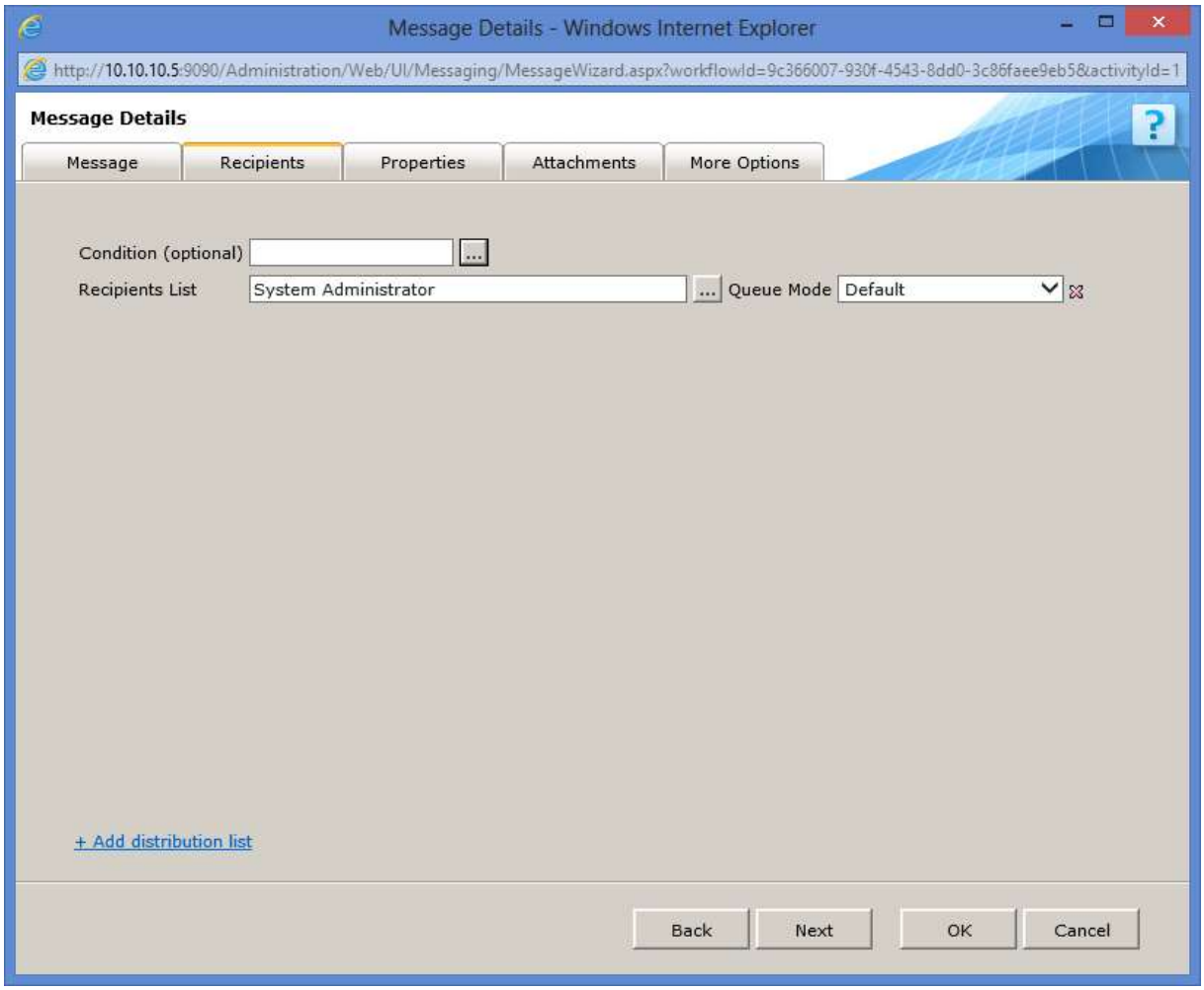
Form is Created

7. Close the UX Studio.
8. Click **…** beside Message in the Task activity Properties pane.
9. Define the **Subject** and **Body** of the message and click **Next**.



Message Properties

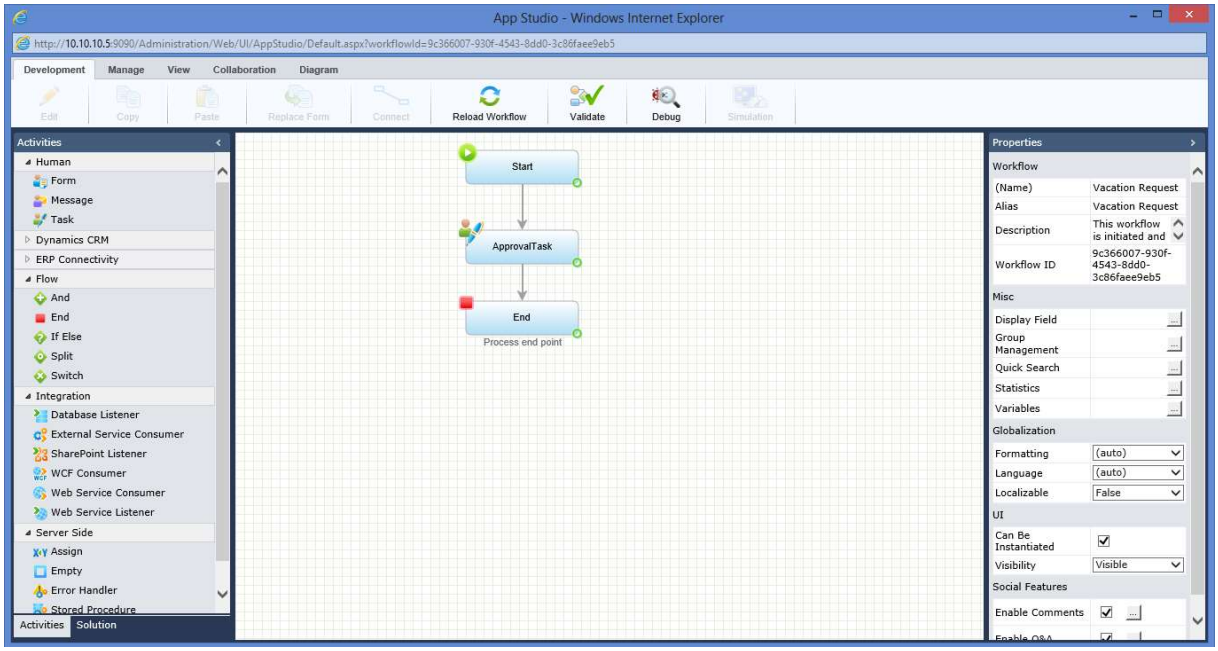
10. Select a **Recipient**, e.g. System Administrator, and click **OK**.



Message Recipients

Step 4: Finalising the Workflow

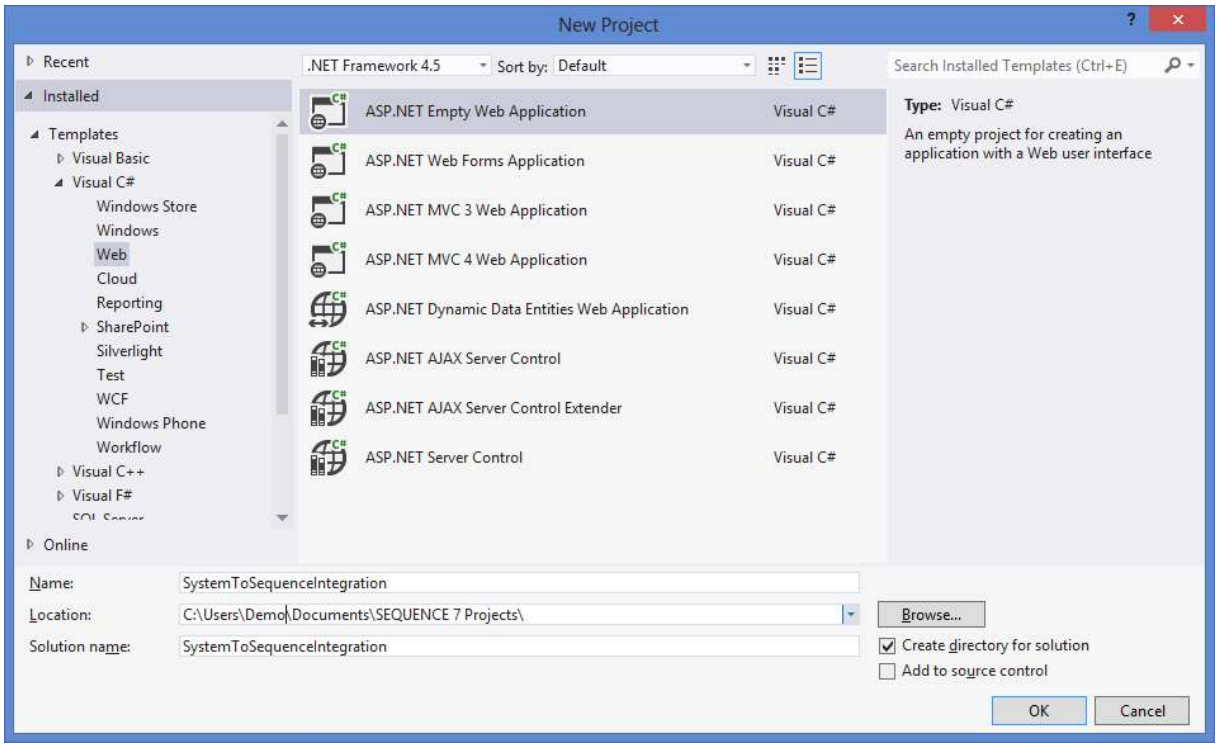
1. Quick Bind the three Activities together on the workflow canvas.
2. Set Workflow Permissions for the user who will be calling the workflow from the ASP.NET application.
3. Keep the UX Studio window open as you will use it to retrieve entities' GUIDs, as well as opening the instance you created using the debugger.



App Studio

Step 5: Creating a .NET Application Project

1. Open Visual Studio and create a new ASP.NET web application or any other project that meets your needs. Give the project a meaningful name (e.g. systemToSequenceIntegration).



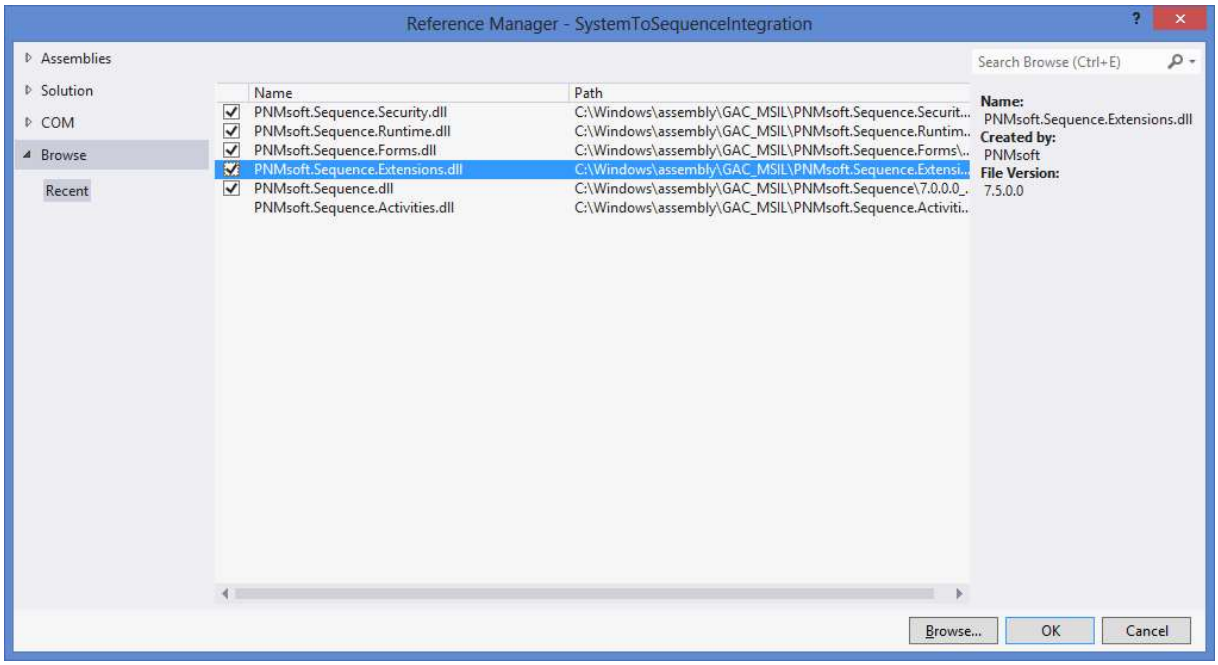
ASP.NET Web Application

In an ASP.NET solution, the Sequence Http Module will handle the authentication for you. If you build a console or window application, you must take care of the authentication yourself.

Step 5.1: Adding a Reference to the Sequence API

1. Add the following references to your project:
   - PNMsoft.Sequence
   - PNMsoft.Sequence.Runtime
   - PNMsoft.Sequence.Security
   - PNMsoft.Sequence.Forms
   - PNMsoft.Sequence.Extensions

   Refer to Appendix A – Setting the Developer Environment.



Add Reference

Note: It is highly advised to include all Sequence assemblies in the Global assembly cache on your development machine.

Step 5.2: Modifying the config File

1. Merge these sections from the web.config found in the Flowtime site to the target config:
   - Sequence.engine Config section
   - Sequence.engine section
   - Handlers section - merge all keys which include PNMsoft elements, exclude any element of `Rad.Web.UI`
   - Modules section - merge all keys which include PNMsoft element Assemblies, exclude any element of `Rad.Web.UI`
   - Diagnostics – copy the entire section
2. Your config should look as follows:

```
<?xml version="1.0"?>
<configuration>
<configSections>
  <sectionGroup name="sequence.engine"
type="PNMsoft.Sequence.Configuration.WorkflowEngineConfigurationSectionGroup,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1">
    <section name="authentication"
type="PNMsoft.Sequence.Security.Configuration.AuthenticationConfigurationSection,
PNMsoft.Sequence.Security, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
    <section name="workflowExecution"
type="PNMsoft.Sequence.Configuration.WorkflowExecutionServiceConfigurationSection,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
allowDefinition="MachineToApplication"/>
```

```
    <section name="notifications"
type="PNMsoft.Sequence.Configuration.NotificationServiceConfigurationSection,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
allowDefinition="MachineToApplication"/>
    <section name="dynamicProxyRuntime"
type="PNMsoft.Sequence.Dynamic.Configuration.DynamicProxyRuntimeConfigurationSection,
PNMsoft.Sequence.Dynamic, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
    <section name="services"
type="PNMsoft.Sequence.Configuration.WorkflowEngineServiceConfigurationSection,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
allowDefinition="MachineToApplication"/>
    <section name="persistence"
type="PNMsoft.Sequence.Configuration.WorkflowEnginePersistenceConfigurationSection,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
allowDefinition="MachineToApplication"/>
    <section name="obs" type="PNMsoft.Sequence.Configuration.ObsConfigurationSection,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
    <sectionGroup name="activities"
type="PNMsoft.Sequence.Configuration.ActivitiesConfigurationSection, PNMsoft.Sequence,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1">
    </sectionGroup>
    <section name="forms"
type="PNMsoft.Sequence.Configuration.FormsConfigurationSection, PNMsoft.Sequence,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
    <section name="mimeTypes"
type="PNMsoft.Sequence.Configuration.MimeTypesConfigurationSection, PNMsoft.Sequence,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
    <section name="resources"
type="PNMsoft.Sequence.Configuration.ResourcesConfigurationSection, PNMsoft.Sequence,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
  </sectionGroup>
</configSections>
  <sequence.engine>
    <authentication impersonate="true">
      <providers>
        <add type="PNMsoft.Sequence.Security.WindowsAuthenticationProvider,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
        <add type="PNMsoft.Sequence.Security.UsernameAuthenticationProvider,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
      </providers>
      <!--<forms enabled="true">
        <web loginUrl="~/_layouts/FormLogin.aspx" />
      </forms>-->
    </authentication>
    <workflowExecution redirectMode="Manual"/>
    <notifications/>
    <!-- Clear; Encrypted; Hashed -->
    <obs userCacheEnabled="true" userCacheCapacity="500" groupCacheEnabled="true"
groupCacheCapacity="500"/>
    <services>
      <add type="PNMsoft.Sequence.Runtime.AuthorizationService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Data.Sql.SqlNotificationService,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
```

```
      <add type="PNMsoft.Sequence.Runtime.WorkflowPersistenceService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.WorkflowDefinitionService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.WorkflowDesignService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.WorkflowExecutionService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.ObsService, PNMsoft.Sequence.Runtime,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.WorkflowMessagingService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.AuthenticationService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.WorkflowDataPropagationService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Dynamic.DynamicProxyRuntime,
PNMsoft.Sequence.Dynamic, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Runtime.FormDefinitionService,
PNMsoft.Sequence.Runtime, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      <add type="PNMsoft.Sequence.Data.EntityModelWorkspaceService,
PNMsoft.Sequence.Extensions, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
    </services>
    <persistence>
      <database provider="System.Data.SqlClient"
credentials="dOtg/dWG6Q1TJDAvqHLX6YzTx1z6qrQoBRzyFXAUK7+0Tw/Fuwb21OH3uMQnJeleocvDumai9
rty4WUPq/n7WQ==" connectionString="packet size=4096;data source=SQLSERVER;persist
security info=False;initial catalog=Sequence;MultipleActiveResultSets=true;"
commandTimeout="1200"/>
      <providers>
        <add type="PNMsoft.Sequence.Data.Sql.SqlAuthenticationDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
        <add type="PNMsoft.Sequence.Data.Sql.SqlAuthorizationDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
        <add type="PNMsoft.Sequence.Data.WorkflowDefinitionDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
        <add type="PNMsoft.Sequence.Data.Sql.SqlWorkflowExecutionDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
        <add type="PNMsoft.Sequence.Data.DbSystemDataSourceProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
        <add type="PNMsoft.Sequence.Data.Sql.SqlObsDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
```

```
        <add type="PNMsoft.Sequence.Data.Sql.SqlMessagingDataProvider,
PNMsoft.Sequence.Data, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"></add>
      </providers>
    </persistence>
    <dynamicProxyRuntime>
      <providers>
        <add name="WebServiceConsumerProvider"
type="PNMsoft.Sequence.Web.Services.WebServiceConsumerProxyFactoryProvider,
PNMsoft.Sequence.Web.Services, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
        <add name="WebServiceListenerProvider"
type="PNMsoft.Sequence.Web.Services.WebServiceListenerProxyFactoryProvider,
PNMsoft.Sequence.Web.Services, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
        <add name="ExternalServiceConsumerProvider"
type="PNMsoft.Sequence.Externals.ExternalServiceProxyFactoryProvider,
PNMsoft.Sequence.Externals, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
        <add name="WcfServiceConsumerProvider"
type="PNMsoft.Sequence.ServiceModel.ServiceConsumerProxyFactoryProvider,
PNMsoft.Sequence.ServiceModel, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
      </providers>
    </dynamicProxyRuntime>
    <forms>
      <validationExtensionTypes>
        <add type="PNMsoft.Sequence.Forms.FileUploadFieldMimeTypeValidationExtension,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
      </validationExtensionTypes>
    </forms>
    <mimeTypes fileLocation="C:\Program Files\PNMsoft\Shared
Resources\Configuration\MimeTypes.config" reloadOnChanges="false"/>
    <resources resXFilesLocation="C:\Program Files\PNMsoft\Shared
Resources\Resources\"/>
    <activities>
    </activities>
  </sequence.engine>
    <system.web>
      <compilation debug="true" targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />
    </system.web>
  <system.webServer>
    <validation validateIntegratedModeConfiguration="false"/>
    <modules>
      <remove name="ScriptModule"/>
      <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      <add name="SequenceAuthenticationModule"
type="PNMsoft.Sequence.Web.AuthenticationModule, PNMsoft.Sequence.Web,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
      <add name="SequenceModule" type="PNMsoft.Sequence.Web.WorkflowEngineHttpModule,
PNMsoft.Sequence.Web, Version=7.0.0.0, Culture=neutral,
PublicKeyToken=0a1a1b90c1c5dca1"/>
      <add name="SequenceFormsModule"
type="PNMsoft.Sequence.Forms.Web.AspFormsHttpModule, PNMsoft.Sequence.Forms,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
    </modules>
```

```
    <handlers accessPolicy="Read, Script">
      <remove name="WebServiceHandlerFactory-Integrated"/>
      <remove name="ScriptHandlerFactory"/>
      <remove name="ScriptHandlerFactoryAppServices"/>
      <remove name="ScriptResource"/>
      <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add name="ScriptHandlerFactoryAppServices" verb="*" path="*_AppService.axd"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add name="ScriptResource" preCondition="integratedMode" verb="GET,HEAD"
path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add name="MemberInfo" verb="GET, HEAD, POST" path="MemberInfo.axd"
type="PNMsoft.Sequence.Web.MemberInfoDetailsHttpHandler, PNMsoft.Sequence.Web,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"/>
    </handlers>
    <defaultDocument>
      <files>
        <clear/>
        <add value="Default.asp"/>
        <!--<add value="FormLogin.aspx"/>-->
      </files>
    </defaultDocument>
    <directoryBrowse enabled="false"/>
    <httpErrors>
      <clear/>
    </httpErrors>
  </system.webServer>
  <system.diagnostics>
    <trace autoflush="true" indentsize="4"/>
    <sources>
      <source name="sequence.runtime" switchName="SourceSwitch"
switchType="System.Diagnostics.SourceSwitch">
        <listeners>
          <remove name="Default"/>
          <add name="RuntimeLog"
type="PNMsoft.Sequence.Diagnostics.SvcFormatWorkflowRuntimeTraceListener,
PNMsoft.Sequence, Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
traceDirectory="C:\temp\Logs"/>
          <add name="EventLogListener"
type="PNMsoft.Sequence.Diagnostics.EventLogTraceListenerEx, PNMsoft.Sequence,
Version=7.0.0.0, Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1"
initializeData="Panam">
            <filter type="System.Diagnostics.EventTypeFilter"
initializeData="Warning"/>
          </add>
        </listeners>
      </source>
      <source switchType="System.Diagnostics.SourceSwitch"
switchName="ObsTracingSwitch" name="sequence.obsTracing">
        <listeners>
          <remove name="Default"/>
```

```
          <add traceData="tblTraceObs"
type="PNMsoft.Sequence.Diagnostics.DbTraceListener, PNMsoft.Sequence, Version=7.0.0.0,
Culture=neutral, PublicKeyToken=0a1a1b90c1c5dca1" name="DbListener"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="SourceSwitch" value="All"/>
      <add value="All" name="ObsTracingSwitch"/>
    </switches>
  </system.diagnostics>
</configuration>
```

3. Change the settings for the database key if required (see the highlighted area above).
4. Change the trace folder to match your environment setting (see the highlighted area above).

Note: It is advisable to adjust the services section to meet your application functionality.

Step 8: Creating the Workflow Utility Class

The workflow utility class deals with the Sequence API and introduces several methods to achieve the interaction between the ASP.NET application and the workflows.

1. Add a new class to your project. Give your class a meaningful name (e.g. WorkflowUtility).
2. Add the following statements to the 'using' statements section:

```
using PNMsoft.Sequence;
using PNMsoft.Sequence.Runtime;
using PNMsoft.Sequence.Messaging;
using PNMsoft.Sequence.Forms.Activities;
using PNMsoft.Sequence.Data;
using PNMsoft.Sequence.Linq;
```

3. Copy and paste the code from the following section into the class body:

```
 //since we are in a web application, the Sequence HttpModule handles the engine
initiation
        static IWorkflowEngine engine = WorkflowRuntime.Engine;

        //Get the workflow execution service the engine
        static IWorkflowExecutionService oExecution =
engine.GetServiceWithCheck<IWorkflowExecutionService>();

        //Starting a new workflow instance for a given workflow , passing dictionary
of parameters
//Starting a new workflow instance for a given workflow , passing dictionary of
parameters
        public static Int32 StartNewWorkflow(Guid workflowId,
Dictionary<string,object> variables)
        {
            WorkflowInstance wfInstance = oExecution.StartWorkflow(workflowId,
variables);
            return  wfInstance.Id;
        }

        public static void UpdateTask(Dictionary<string,object> values,int
taskInstanceId)
        {
            //Get the Task executor from the engine Static method based on Task
Instance id, if Engine is null the system will use WorkflowRuntime.Engine
            TaskExecutor taskExecutor =
TaskExecutor.GetExecutor(taskInstanceId,engine);

            using (DataContext ctx = taskExecutor.CreateDataContext())
            {
                IDataTable tbl = ctx.GetDataTable("ApprovalTask"); //ApprovalTask is
the name of the query we need to update
                IDataRow row = tbl.CreateRow();

                //Iterate trough the Dictionary, and set fields to the row.
                foreach (KeyValuePair<string, object> entry in values)
                {
                    row.SetField(entry.Key, entry.Value);
                }
                //add the row to the table
                tbl.InsertOnSubmit(row);
                bool forceCloseTask = true;
                bool forceCloseActivity = true;

                ctx.SubmitChanges(forceCloseTask, forceCloseActivity);
            }

        }

        public virtual void FetchTaskInstance(int activityInstanceId, int userId)
        {
           //Get the Activity Instance
            ActivityInstance activityInstance =
oExecution.GetActivityInstance(activityInstanceId);

            //Check that this activity is not already fetched
```

```
            if (activityInstance.FetchedById.HasValue &&
activityInstance.FetchedById.Value > 0)
            {
                return;
                //Add Code that deals with this case
            }

            //Get the Task instance from the activity instance, where the user id
match the user id.
            //The user that execute this code must have full control on the activity
or be the fetched user.
            TaskInstance taskInstance =
activityInstance.MessagesInstances.FirstOrDefault(
                t => t.ToId == userId) as TaskInstance;

            //If there was no task to fetch
            if (taskInstance == null)
            {
                return;
                //Add Code that deals with this case
            }

            TaskExecutor taskExecutor = new TaskExecutor(taskInstance);
            taskExecutor.Fetch();
        }

        public static Dictionary<string, object> GetFormData(int workflowInstanceId,
int ActivityId, int taskInstanceId)
        {
            Dictionary<string, object> formValues = new Dictionary<string,object>;
            TaskExecutor taskExecutor = TaskExecutor.GetExecutor(taskInstanceId,
engine);

            using (DataContext ctx = taskExecutor.CreateDataContext())
            {
                IDataTable tbl = ctx.GetDataTable("ApprovalTask"); //ApprovalTask is
the name of the query we need to update
                foreach (var dataRow in tbl)
                {
                    foreach (EntityPropertyDescriptor property in
dataRow.GetProperties())
                    {
                        if (!property.EntityProperty.IsAssociation)
                        {
                            formValues[property.Name] = dataRow[property.Name];
                        }
                    }
                }
            }
            return formValues;
        }
```

Step 9: Understanding the Workflow Utility class

- **oExecution**: a static object of PNMsoft.Sequence.Runtime.IWorkflowExecutionService type that is used to initiate new workflows and get existing ones.

- **StartNewWorkflow**: a method which takes a workflow unique identifier and returns the id of a new workflow instance that was created for this workflow.

- **UpdateTask**: a method which takes a Dictionary of keys and values (we assume the keys match the fields name), and the task id (message instance id). It updates the task form with the values and closes the task.

  Method logic:

  - Get the context from the task executor.
  - Get the Table, e.g. the query.
  - For each pair in the dictionary, set the field in the row.
  - Submit the changes.

- **FetchTaskInstance**: a method which takes a task activity id and user id and fetches the task to that user. This method is required in case a task was sent to a queue.

  Method logic:

  - Get the activity instance from the execution service.
  - Make sure the task is not already fetched.
  - Find the task instance that is associated with the user.
  - Fetch the task using a task executor.

- **GetFormData**: a method which takes a task instance id, and return a dictionary of fields and values for a query of the task.

  Method logic:

  - Get the context from the task executor.
  - Get the Table e.g. the query.
  - For each row, take EntityPropertyDescriptor and fill in the data.

  *** this sample assumes there is one record. For multiple records, the method should be changed to return a data table.

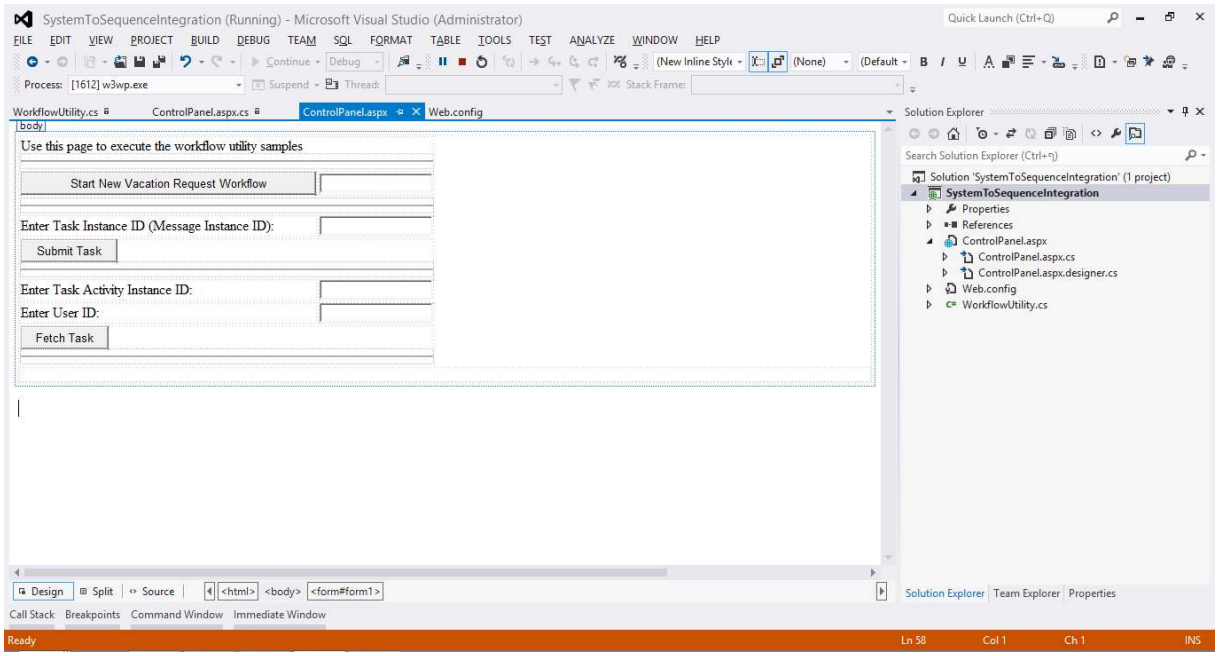
Step 10: Building a Sample Form to Execute the Code

1. Add a web form to your project and give it a meaningful name, e.g. ControlPanel.
2. Paste this form in your aspx source.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ControlPanel.aspx.cs"
Inherits="SystemToSequenceIntegration.ControlPanel" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                 <tr>
                    <td colspan="2"> <label>Use this page to execute the workflow
utility samples</label>
                        <hr />
                    </td>
                </tr>

                <tr>
                    <td > <asp:Button ID="RunWorkflow" runat="server" Text="Start New
Vacation Request Workflow" OnClick="StartVacationWorkflow_Click" />

                    </td>
                    <td><asp:TextBox runat="server"
ID="txtWorkflowInsatnceID"></asp:TextBox></td>
                </tr>
                <tr><td colspan="2"><hr /></td></tr>

                <tr>
                    <td><label>Enter Task Instance ID (Message Instance ID):
</label></td>
                    <td><asp:TextBox runat="server"
ID="txtTaskInstanceID"></asp:TextBox> </td>
                </tr>
                <tr>
                    <td colspan="2"> <asp:Button ID="SubmitTask" runat="server"
Text="Submit Task" OnClick="UpdateApprovalTask_Click" />
                        <hr />
                    </td>
                </tr>

                 <tr>
                    <td><label>Enter Task Activity Instance ID: </label></td>
                    <td><asp:TextBox runat="server" ID="txtActivityID"></asp:TextBox>
</td>
                </tr>
                <tr>
                    <td><label>Enter User ID: </label></td>
```

```
                    <td><asp:TextBox runat="server" ID="txtUserID"></asp:TextBox>
</td>
                </tr>
                <tr>
                    <td colspan="2"> <asp:Button ID="FetchTask" runat="server"
Text="Fetch Task" OnClick="FetchTask_Click" />
                        <hr />
                    </td>
                </tr>
            </table>
        </div>
    <div>

    </div>
    </form>
</body>
</html>
```

Your form will look like this:



3. Add this code to your code behind (cs) file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace SystemToSequenceIntegration
{
    public partial class ControlPanel : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void StartVacationWorkflow_Click(object sender, EventArgs e)
```

```
        {
            Dictionary<string,object> values = new Dictionary<string,object>();
            //Populate the dictionary based on your solution needs
            values.Add("Department","Product");
            values.Add("Number Of Days",5);
            values.Add("Direct Mgr","Paul");
            values.Add("Starting Date",new DateTime(2013,12,24));
            values.Add("Reason","none");
            int workflowInstanceId =  WorkflowUtility.StartNewWorkflow(new
Guid("46ebd98e-2d4f-4e02-b27d-29245a06dd77"), values); // the GUID here is the
workflow id.
            txtWorkflowInsatnceID.Text = workflowInstanceId.ToString();

        }

        protected void UpdateApprovalTask_Click(object sender, EventArgs e)
        {
            Dictionary<string, object> values = new Dictionary<string, object>();
            int taskInstanceId;

            Int32.TryParse(txtTaskInstanceID.Text, out taskInstanceId);

            //Populate the dictionary based on your solution needs
            values.Add("Approved", true);
            values.Add("Comments", "Auto Approval Comments");


            WorkflowUtility.UpdateTask(values, taskInstanceId);

        }

        protected void FetchTask_Click(object sender, EventArgs e)
        {
            int userId;
            int activityId;
            Int32.TryParse(txtUserID.Text, out userId);
            Int32.TryParse(txtActivityID.Text, out activityId);

            WorkflowUtility.FetchTaskInstance(activityId, userId);

        }
    }
}
```

Note: in this code, we have used hardcoded dictionary values. In a real scenario you will fill in the values based on you solution logic.

Note: All the values like workflow id and task id can be retrieved using the API execution service. This sample does not cover this code.

Step 11: Authentication and Execution

This sample assumes you are using Windows-based authentication. The Sequence `SequenceAuthenticationModule` handles the authentication and impersonation for you.

If your scenario is different:

- Add this method to your workflow utility:

```
public static void Authenticate(string userName, string password)
        {
            AuthenticatedUser authenticatedUser =
WorkflowRuntime.Engine.GetService<IAuthenticationService>().Authenticate(userName,
password);
            SecurityManager.Impersonate(authenticatedUser);
 }
```

- Add calls to this method prior to any other method call. For example:

```
WorkflowUtility.Authenticate("administrator","e");
int workflowInstanceId =  WorkflowUtility.StartNewWorkflow(new Guid("46ebd98e-2d4f-
4e02-b27d-29245a06dd77"), values); // the GUID here is the workflow id.
```

1. Set the controlpanel.aspx as your starting page.
2. Click **Run** or browse to your web application.
3. Click the Start New Vacation Request Workflow. The instance id is displayed on the text box to the right.
4. Enter the task Instance id. You can get the id from the query string when opening the task from the Flowtime Inbox, or from the view message window in the debugger.

   Click **Enter** to submit the task with the predefined values.

Appendix A – Setting the Developer Environment

The quickest way to set your developer environment with all the required functionality is to install the “Sequence SharePoint-less Flowtime”.

Before you launch the set up verify that:

- You have the correct privileges on your machine.
- You have installed the IIS feature on your machine.
- You have created a website to host the Flowtime (it can be the default web site).

After installation, you will get all the relevant dlls in your GAC and a valid web config in the website folder.

If you are using other components from the Sequence product, you can add your solution virtual directory to the Shared Resources folder.