Workflow Package Tutorial Sample

iAnywhere® Mobile Office 5.7

SYBASE® iAnywhere

Document ID: DC01174-01-0570-01
LAST REVISED: *September 2009*

Copyright and Trademarks

Disclaimer

This documentation, as well as the software described in it, is furnished under a license agreement. The content of this documentation is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this documentation. Any changes in the programs will be incorporated in a future edition of this publication.

Technical Support

For product-specific technical information, visit the iAnywhere Technical Support site at http://frontline.sybase.com/support/.

NOTE: Register at our technical support site for the latest information for your product. This site is available only to customers with a valid maintenance contract.

United States
+1 800 235 7576, menu options 2, 1
+1 208 322 7575, menu options 2, 1
6:00 a.m. to 6:00 p.m. Mountain Time (GMT -7)

United Kingdom
+44 (0) 117 333 9032
8:00 a.m. to 6:00 p.m. (GMT+1)

Germany
+49 (0) 7032 798-555
8:00 a.m. to 6:00 p.m. (GMT+1)

France
+33 (0) 825 826 835
8:00 a.m. to 6:00 p.m. (GMT+1)

Benelux
+31 (0) 302 478 455
8:00 a.m. to 6:00 p.m. (GMT+1)

Table of Contents

Introduction

Creating a Business Process Workflow Package with .NET Technology

This tutorial will guide readers through the process of creating a Workflow package using the .NET Framework. When completed, readers would have developed an end-to-end Workflow package that is able to collect data on a handheld device and asynchronously send it to Mobile Office for processing.

Server to client data transfer is demonstrated in this tutorial by sending a client module and list object to a mobile device. Once the user has selected an item from the list and filled out the various user-interface control items, the data is serialized and sent back to the server to be written to a text file.

A reference of the completed Workflow package (ActivitiesPackage.zip) can be found in the **C:\Program Files\iAnywhere Mobile Office\Tools\ Workflow Samples\Activities** directory.

This tutorial illustrates the five basic steps required to build and configure an end-to-end Workflow package. The five steps are:

1. Create the Server Module (WorkflowServer.dll)
2. Create the Client Module
    i. Create a .NET Compact Framework Client Module (WorkflowClient.dll)
    ii. Create a XML Client Module (ActivitiesWorkflow.xml)
3. Package the Workflow package
4. Deploy and Assign Users to Workflow package
5. Run the Workflow package on a Client Device

STEP ONE: Create the Server Module (WorkflowServer.dll)

The following instructions guide readers through the process of creating a server module. The server module is responsible for processing the e-mail message and supplementing it with context data so that the client module could be used to display it in a meaningful fashion. The server module also handles the response from the handheld device.

The server module used in this tutorial will consist of two .NET classes; both built to a single WorkflowServer.dll file.

A copy of the completed WorkflowServer.dll project is located in the **C:\Program Files\iAnywhere Mobile Office\Tools\ Workflow Samples\Activities\ WorkflowServer** directory. Before building this project, the AMPInterfaces.dll file located in **C:\Program Files\iAnywhere Mobile Office\Bin** must be added as a project reference.

1. Open Visual Studio.NET 2005

2. Start a **Windows C# Class Library** project named **WorkflowServer**.

3. In the Solution Explorer, rename file Class1.cs to **Handler.cs**. Click **Yes** if prompted to update all references of Class1 to Handler.

4. Right-click the **References** tab of the **WorkflowServer** project and add a reference to the **AMPInterface.dll** library. **AMPInterfaces.dll** is located in the **C:\Program Files\iAnywhere Mobile Office\Bin** directory.

5. Open **Handler.cs** and add the following code to import various namespaces used by the Workflow package component.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Collections.Generic;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using iAnywhere.MobileOffice.AMP;
```

6. Remove the **Handler** class definition that was generically added to the **WorkflowServer** namespace upon beginning the project.

```
public class Handler
{ }
```

7. Create a new public class in the **WorkflowServer** namespace named **TransformerComponent**. This class must inherit the **IMailProcessor** interface defined in the iAnywhere.MobileOffice.AMP namespace.

```
public class TransformerComponent : IMailProcessor
{ }
```

8. Add the following function to the **TransformerComponent** class.

```
    public void ProcessMail( ContextData oContextData,
                             MailData oMailData,
                             out string sData )
   {
      // Create a new message to be sent to the client module
      XmlWorkflowMessage _message = new XmlWorkflowMessage();

      // The message header can be used to transport state information (optional)
      _message.Header = Guid.NewGuid().ToString( "N" );

      // Demonstrate jumping to a different screen in the Workflow package.
      // In this example, jump to the comment screen if the subject
      // of the e-mail contains the text "skip me"
      if (oMailData.Subject.Contains("skip me"))
      {
         // The WorkflowScreen property can be set to show a particular screen
         // instead of the first screen.  (optional)
         // For a .NET Workflow package, this property should be a class name.
         // For a Xml Workflow package, this property should be the screen name.
         //
         // To target both types of Workflow packages, name the screen of the
XmlWorkflow
         // the same as the class name.
         _message.WorkflowScreen = "WorkflowClient.uicComment";
      }

      // Get the intial value from the Context Variables
      // Context variables are defined in the manifest.xml file and can be
      // edited by Administrators using the Mobile Office Admin console.
      string initialActivity = oContextData.Variables["Activity"];
      string initialLocation = oContextData.Variables["Location"];
      // Set the selected value for the "activity" combobox and
      // "location" combobox in the uicMain form
      // or the "activity" selectbox and "location" selectbox in the
      // ActivitiesWorkflow.xml
      _message.Values.Add( "activity", initialActivity );
      _message.Values.Add("location", initialLocation);

      // sData is available as the CustomContext in the uicMain.ApplyContext() method
      sData = _message.Serialize();
   }
```

ProcessMail is the first function called when the Workflow package is run and gives an opportunity for server modules to supplement the e-mail message with any context data to send to the device.

The above sample creates an instance of a XmlWorkflowMessage class used for storing all the data to be sent to the client module. The XmlWorkflowMessage class is available on both .NETCF and XML client modules and should be used when a server module's implementation is to be shared between a .NETCF and XML client implementation.

sData is assigned to the string serialization of the XmlWorkflowMessage object, and is deserialized in the client module automatically by Mobile Office when XML is used, or by the XmlWorkflowMessage.CreateFromXml method in .NETCF.

In *STEP 2: Create the Client Module*, sData is retrieved, deserialized and used by the client module. Therefore, if no data is required to be sent from the server to client, then sData need not be populated.

Steps 7 and 8 complete the required code to send data from the server to client.

The following instructions implement the server module response component for when data is sent from the client to server.

9. Create a new public class in the **WorkflowServer** namespace named **ResponderComponent**. This class must inherit the **IResponseProcessor and ISynchronousRequestProcessor** interface defined in the iAnywhere.MobileOffice.AMP namespace.

```
public class ResponderComponent : IResponseProcessor,
                                  ISynchronousRequestProcessor
{ }
```

10. Add the following method to the **ResponderComponent** class.

```
    ....
private static string USERNAME = "user1";
private static string PASSWORD = "password";

public void ProcessResponse( ContextData oContextData, string sData )
{
    if ( String.IsNullOrEmpty( sData ) )
    {
        // nothing to process
        return;
    }

    // sData was sent to us in the uicMain.Serialize() method
    XmlWorkflowMessage msg = XmlWorkflowMessage.CreateFromXml( sData );

      // Demonstrate requesting the client to enter new credentials by throwing
      // a CredentialRequestException
      string desc = Convert.ToString( msg.Values["description"] );
      if ( oContextData.BackEndUser.Equals( USERNAME ) &&
           desc.Contains( "discard" ) )
      {
         throw new CredentialRequestException( "Please refresh credentials for Activity
                                                Workflow package",
                                               " Workflow package Admin",
                                                string.Empty );
      }

     // Put the values from the client into an Activity object
      Activity activity = new Activity( msg.Header );
      activity.Name = Convert.ToString( msg.Values["activity"] );
      activity.Location = Convert.ToString( msg.Values["location"] );
      activity.Date = Convert.ToDateTime( msg.Values["date"] );
      activity.Summary = Convert.ToString( msg.Values["description"] );
      activity.Comments = Convert.ToString( msg.Values["comments"] );
      activity.RequestAction = msg.RequestAction;
      activity.WorkflowScreen = msg.WorkflowScreen;
      activity.BackendPassword = oContextData.BackEndPassword.TrimEnd();
      activity.BackendUsername = oContextData.BackEndUser;

    // Save the Activity
    _SaveResponse( oContextData, activity );
}
```

ProcessResponse is the first method called when control is returned from client to server. In this example, string variable sData, will contain the serialized data collected from the client module. The above ProcessResponse implementation deserializes sData and saves it via a helper function called _SaveResponse.

It is also possible to inform the Workflow package on the client that new credentials (if supported by the Workflow package) are required by throwing a CredentialRequestException (see the "iAnywhere.MobileOffice.AMP.CredentialRequestException Class" topic in the *iAnywhere Mobile Office Workflow Package Developer Guide*). The user is prompted with an e-mail in their Inbox when a CredentialRequestException is thrown and processing for the Workflow package on the user's device stops until credentials are received.

The credential information from the Workflow package on the client are available in ContextData as the BackEndUser and BackEndPassword properties (see the "iAnywhere.MobileOffice.AMP.ContextData Class" topic in the *iAnywhere Mobile Office Workflow Package Developer Guide*).

11. Add the following method to the **ResponderComponent** class.

```
private void _SaveResponse( ContextData oContextData, Activity activity )
{
    try
    {
        // Get the path from the context variables
        // Context variables are defined in the manifest.xml file and can be
        // edited by Administrators using the Mobile Office Admin console.
        string path = oContextData.Variables["OutputFolder"];
        // Set username.devicetype.deviceID.txt as the filename
        string filename = String.Format( "{0}.{1}.{2}.txt",
            oContextData.UserName,
           oContextData.DeviceType,
           oContextData.DeviceName );
        // Construct the absolute path and filename
        path = System.IO.Path.Combine( path, filename );

        // This will cause an exception if the absolute path is not valid on
        // the system
        using ( StreamWriter sw = File.AppendText( path ) )
        {
            sw.WriteLine( "ProcessReponse - " + DateTime.Now + " - " + activity.Identity );

            // get the activity object as an xml string
            XmlSerializer xmls = new XmlSerializer( typeof( Activity ) );
            StringWriter stringWriter = new StringWriter();
            xmls.Serialize( stringWriter, activity );

            sw.WriteLine( stringWriter.ToString() );
            sw.WriteLine();
        }// using
    }
    catch ( Exception ex )
    {
        // This exception will be logged in the Workflow package's Error List in the
        // Mobile Office Admin console.
        throw ex;
    }
}
```

_SaveResponse, given a ContextData and Activity type parameter, will output the data of the Activity parameter to a location specified in the manifest file (discussed in *STEP 3: Package the Workflow package*). The name of the file is decided by the username, device type and device name used by the client. Therefore, a device connected to iAnywhere Mobile Office with username *InboxAppUser*, on a Windows Mobile 5.0 Pocket PC device with device id 150006F0063006B006500740050004300000000 would produce a results file named *InboxAppUser.WindowsMobileProfessional.150006F0063006B006500740050004300000000.txt.*

The following instructions will implement the Activity class. An instantiation of an Activity object is used when data is serialized and sent from client to server.

12. Add the following method to the **ResponderComponent** class.

```
public string ProcessSyncRequest(ContextData oContextData, string sData)
{
   if ( String.IsNullOrEmpty( sData ) )
   {
      // nothing to process
      return sData;
   }

   // sData was sent to us in the uicMain.Serialize() method
   XmlWorkflowMessage msg = XmlWorkflowMessage.CreateFromXml(sData);

   string description = Convert.ToString(msg.Values["description"]);
   if ( description == null )
      description = "";

   string rmitype = Convert.ToString(msg.Values["rmitype"]);
   if ( rmitype == null )
      rmitype = "";

   if ( msg.WorkflowScreen == "main" )
   {
      //demonstrate the ability to update Workflow package screen control data
      msg.Values.Remove("description");
      msg.Values.Add("description", "Description data requested from server.");
   }

   if ( msg.WorkflowScreen == "WorkflowClient.uicComment" )
   {
      if ( rmitype.ToLower().Contains( "update" ) )
      {
         //demonstrate the ability to update the Workflow package screen controls
         msg.Values.Remove( "comments" );
         msg.Values.Add( "comments", "Comments data requested from server." );
      }
      else if ( rmitype.ToLower().Contains( "exception" ) )
      {
         //demonstrate throwing an exception request from server to client
         throw new Exception( "Server demonstrates throwing an exception to
                              device." );
      }
      else if ( rmitype.ToLower().Contains( "screen" ) )
      {
         //demonstrate the ability to change screens
         msg.WorkflowScreen = "main";
      }
   }

   return msg.Serialize();
}
```

The ProcessSyncRequest method handles a Remote Method Invocation (RMI) from the client Workflow package. Unlike the ProcessResponse method, the client Workflow package is waiting for the ProcessSyncRequest method to complete and a timeout may occur if the method does not complete in a timely fashion (see timeout attribute for the <action> tag in the Workflow Developer Guide). The return value from ProcessSyncRequest should be a serialized XmlWorkflowMessage if the client Workflow package is an Xml Workflow package. Values from the returned XmlWorkflowMessage are used to update values in the client Workflow package with the same key. It is also possible to redirect the client Workflow package to a different screen when the method returns by using the XmlWorkflowMessage.WorkflowScreen property.

13. In the Solution Explorer, right-click the WorkflowServer project and select **New Item…** from the **Add** menu. From the popup menu, select **Class** and name the file **Activity.cs**.

14. In the Solution Explorer, double-click the **Activity.cs** file to view the source code.

15. From the top of the Activity.cs source code, remove all `using` namespace statements except `using System;`

16. Modify the default Activity signature to be public

```
public class Activity
```

17. Copy the following code into the Activity class.

```
private string m_ident;
private string m_activityName;
private string m_locationName;
private DateTime m_activityDate;
private string m_summary;
private string m_comments;
private string m_requestAction;
private string m_workflowScreen;
private string m_backendUsername;
private string m_backendPassword;


public Activity()
{
    m_ident = "";
}// Activity()

public Activity( string ident )
{
    m_ident = ident;
}// Activity ( string ident )

public string Identity
{
    get
    {
        return m_ident;
    }
}// Identity

public string Comments
{
    get
    {
        return m_comments;
    }
    set
    {
        m_comments = String.IsNullOrEmpty(value) ? "" : value;
    }
}// Page Two

public string Name
{
```

```
        get
        {
            return m_activityName;
        }
        set
        {
            m_activityName = String.IsNullOrEmpty( value ) ? "" : value;
        }
    }// Name

    public string Location
    {
       get
       {
          return m_locationName;
       }
       set
       {
          m_locationName = String.IsNullOrEmpty(value) ? "" : value;
       }
    }// Location

    public DateTime Date
    {
        get
        {
            return m_activityDate;
        }
        set
        {
            m_activityDate = value;
        }
    } // Date

    public string Summary
    {
        get
        {
            return m_summary;
        }
        set
        {
            m_summary = String.IsNullOrEmpty( value ) ? "" : value;
        }
    }// Summary

    public string RequestAction
    {
       get
       {
          return m_requestAction;
       }
       set
       {
          m_requestAction = String.IsNullOrEmpty( value ) ? "" : value;
       }
    }// SubmitAction

    public string WorkflowScreen
    {
```

```
      get
      {
         return m_workflowScreen;
      }
      set
      {
         m_workflowScreen = String.IsNullOrEmpty( value ) ? "" : value;
      }
   }// WorkflowScreen

   public string BackendUsername
   {
      get
      {
         return m_backendUsername;
      }
      set
      {
         m_backendUsername = String.IsNullOrEmpty( value ) ? "" : value;
      }
   }// BackendUsername

   public string BackendPassword
   {
      get
      {
         return m_backendPassword;
      }
      set
      {
         m_backendPassword = String.IsNullOrEmpty( value ) ? "" : value;
      }
   }// BackendPassword

}// class Activity
```

The above implementation defines class fields used when collecting data from the client to be sent to the server. The get and set definitions are required by the Microsoft .NET Framework for XML serialization and deserialization.

18. Save the changes and build the project. Verify that the **WorkflowServer.dll** assembly is saved to the bin folder of the **WorkflowServer** project.

STEP 2: Create the Client Module

Workflow package developers have the flexibility of creating the client module either as a .NET Compact Framework project or formatted XML document. Both client module types are discussed in the following subsections. Furthermore, each client module type described in this tutorial will share the same user-interface format and functionality.

Client modules developed with .NETCF are only supported on Windows Mobile Professional (WM6), Windows Mobile Pocket PC (WM5), Windows Mobile Standard (WM6) and Windows Mobile Smartphone (WM5) devices. This option gives the widest degree of flexibility as the client module can basically be made to do anything that .NETCF supports.

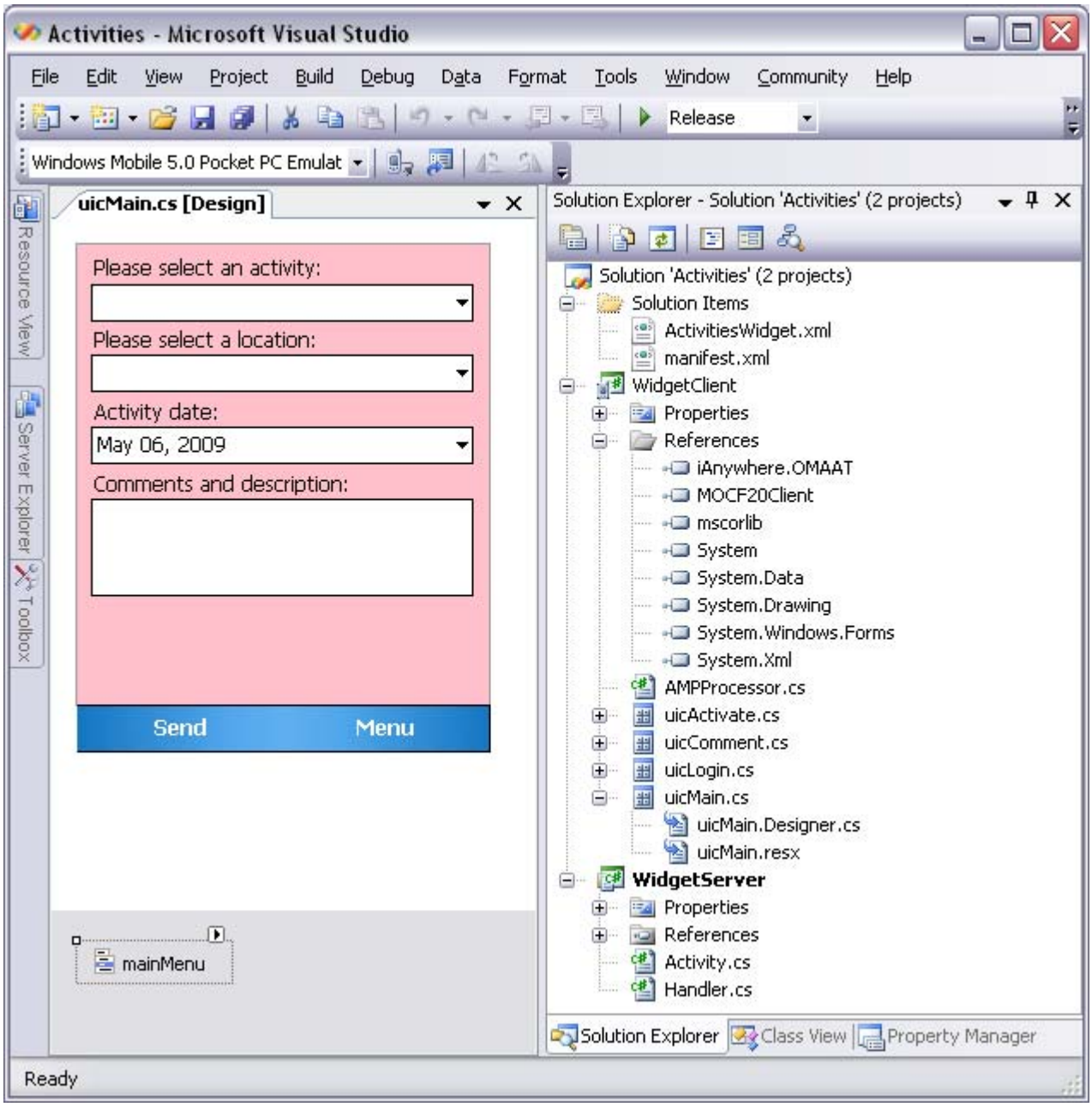
Client modules developed with a cross-platform XML description file are supported on all the above mentioned platforms in addition to Symbian and iPhone. Therefore, this option gives the widest range of platform support using a single code base.

To develop a .NET Compact Framework client module please reference subsection *STEP 2 (i): Create a .NET Compact Framework Client Module (WorkflowClient.dll)*; otherwise, subsection *STEP 2 (ii): Create a XML Client Module (ActivitiesWorkflow.xml)* will review the process of developing a XML client module.

STEP 2 (i): Create a .NET Compact Framework Client Module (WorkflowClient.dll)

The following instructions guide readers through the process of creating a .NET Compact Framework client module. The client module used in this subsection will consist of four user-controls (each corresponding to a single screen) and will be built to WorkflowClient.dll.

A copy of the completed project is located in the **C:\Program Files\iAnywhere Mobile Office\Tools\ WorkflowSamples\Activities\ WorkflowClient** directory. Before building this project, the iAnywhere.OMAAT.dll assembly located in **C:\Program Files\iAnywhere Mobile Office\Tools\MO_SDK\DotNet\WM5\PPC** and the MOCF20Client.dll assembly located in C:\Program Files\iAnywhere Mobile Office\Tools\MO_SDK\DotNet\WM5 must be added as project references.



1. Open Visual Studio.NET 2005.

2. Create a new project and select **Visual C# | Smart Device | Windows Mobile 5.0 Pocket PC | Control Library**. Name the project **WorkflowClient**.

3. In the Solution Explorer, rename UserControl1.cs to **uicMain.cs**.

4. In Design mode, right click on the user-control and select **Properties**. Assign the following values:

   BackColor: Pink

Name: uicMain
Size: 240, 268
confiscate it

5. Using the Toolbox, add a ComboBox named **cbxActivity**. In the properties of the ComboBox go to Items and add a collection of strings to populate the ComboBox. For example add “I was swimming”, “I was running”, “I was shopping” and “I was working”. Create a label beside the ComboBox; name the label **lblActivity** and change its Text setting in properties to **Please select an Activity:**

6. Using the Toolbox, add a ComboBox named **cbcLocation**. This ComboBox will be populated with named value pairs in the ApplyContext method. Create a label beside the ComboBox; name the label **lblLocation** and change its Text setting in properties to **Please select a location:**

7. Using the Toolbox, add a DateTimePicker named **dtpDate**. In the properties of the DateTimePicker, assign Format to **Custom**, and the CustomFormat property to **MMMMMMMM dd, yyyy**. Create a label beside the DateTimePicker named **lblDate** and change its Text setting in properties to **Activity date:**

8. Using the Toolbox, add a TextBox named **tbxSummary**. In the properties of the TextBox, adjust the following values:

   Text:
   Multiline: True
   Size: 223, 57

   Using the Toolbox, create a lable **lblSummary** above the TextBox named tbxSummary and change its Text property to **Comments and description:**

9. Using the Toolbox, add a new ContextMenu named **mainMenu**. In the Solution Explorer right click on uicMain.Designer.cs and select **View Code**. In InitializeComponent() change

   ```
   this.mainMenu = new System.Windows.Forms.ContextMenu();
   ```
   to
   ```
   this.mainMenu = new System.Windows.Forms.MainMenu();
   ```

   At the end of the file change

   ```
   private System.Windows.Forms.ContextMenu mainMenu;
   ```
   to
   ```
   private System.Windows.Forms.MainMenu mainMenu;
   ```

   Save and close the uicMain.Designer.cs file and return to the Design view of uicMain.cs.

10. Create a left menu button named **menuSend** and set the label property to **Send**. On the right of the main menu, create a submenu named **menuMenu** and set the label property to **Menu**. Add the following menu items to the submenu: **menuNext** with label property **Next** and **menuCancel** with label property **Cancel**.

11. Right-click the **References** tab of the **WorkflowClient** project and add the **iAnywhere.OMAAT.dll** library. iAnywhere.OMAAT.dll is located in the **C:\Program Files\iAnywhere Mobile Office\Tools\MO_SDK\DotNet\WM5\PPC** directory.

12. Right-click the **References** tab of the **WorkflowClient** project and add the **MOCF20Client.dll** library. MOCF20Client.dll is located in the **C:\Program Files\iAnywhere Mobile Office\Tools\MO_SDK\DotNet\WM5** directory.

13. Right-click on the user-control, and select **View Code**. The remaining instructions implement the functionality of the user-control in **uicMain.cs**.

At the top of the user-control source file, add the following code to import various namespaces used by the user-control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using iAnywhere.OMAAT;
using iAnywhere.MobileOffice.AMP;
```

14. Modify the **uicMain** class signature to inherit **UIComponent** instead of **UserControl**.

```
public partial class uicMain : UIComponent
{ }
```

15. Add the following class field definition just before the **uicMain()** constructor in the **uicMain** class.

```
private bool m_bCancel = false;
 private string m_strRequestAction = String.Empty;
XmlWorkflowMessage m_Msg;
```

16. Add the following **ApplyContext** method definition below the **uicMain()** constructor in the **uicMain** class.

```
protected override void ApplyContext()
{
    //reset the UI to a known good state
    Framework.BaseForm.Menu = this.mainMenu;
    this.m_bCancel = false;

    // Initialize the UI once when first loaded
    if ( !this.Valid )
    {
          // Set existing controls to default values.
          this.cbxActivity.Text = "";
          this.cbxLocation.Text = "";
          this.dtpDate.Value = DateTime.Today;
          this.tbxSummary.Text = "";

          // CustomContext is what was set in TransformerComponent.ProcessMail
          // on the server
          string strCustomContext = this.CustomContext as string;
          if ( !String.IsNullOrEmpty( strCustomContext ) )
          {
             this.m_Msg = XmlWorkflowMessage.CreateFromXml( strCustomContext );
             if ( this.m_Msg != null )
             {
                this.cbxActivity.Text = Convert.ToString( this.m_Msg.Values["activity"] );
```

```
                // parse the location context variable for a list of items
                // the user can select
                 this.cbxLocation.Items.Clear();
                 this.cbxLocation.Text = "";

                 string[] _values =
                         Convert.ToString( this.m_Msg.Values["location"] ).Split('|');

                 List<string> _keyCollection = new List<string>();

                 foreach ( string _val in _values )
                 {
                    string _key = _val;
                    string _value = _val;
                    int _index = _val.IndexOf( '$' );

                    if ( _index >= 0 )
                    {
                       _key = _val.Substring( 0, _index );
                       _value = _val.Substring( _index + 1 );
                    }

                    if ( !_keyCollection.Contains( _key ) )
                    {
                       _keyCollection.Add( _key );
                       this.cbxLocation.Items.Add( _value );
                    }
                 }

                 if ( this.cbxLocation.Items.Count > 0 )
                    this.cbxLocation.Text = this.cbxLocation.Items[0].ToString();
              }
           }
           else
           {
              this.m_Msg = new XmlWorkflowMessage();
           }

           this.Valid = true;
        }
     }
```

ApplyContext is the first method called when the client module about to be displayed on the mobile device. The implementation above overrides that defined in the UIControl base class; if the client module is being loaded for the first time, then default values will be assigned to each of its user-interface control items; otherwise, the user must be navigating back from the second page of the client module and the data previously entered to the first page is restored.

When the client module first loads to a mobile device, the cbxActivity ComboBox is populated with the values previously assigned by the ProcessMail method during instruction 8 of *STEP 1: Create Server Module (WorkflowServer.dll)*. The cbxLocation ComboBox is filled with named value pairs.

17. Add the following **Validate** function definition below the **ApplyContext** method in the **uicMain** class.

```
protected override bool Validate()
{
    //Check if the user selected to close the form
    //without submitting the entered data
    if ( this.m_bCancel )
    {
        this.Valid = false; // do not submit the data to server
        return true;        // the form is allowed to close
    }

    //perform simple validation: if an activity is not selected or summary
    //is blank
    //do not allow the form to close
    if ( String.IsNullOrEmpty( cbxActivity.SelectedItem.ToString() ) ||
         String.IsNullOrEmpty( cbxLocation.SelectedIndex.ToString() ) ||
         String.IsNullOrEmpty( tbxSummary.Text ) )
    {
        // the form is not allowed to close
        return false;
    }

    // All validation passed
    this.Valid = true;   // submit the data to server
    return true;         // the form is allowed to close
}
```

Validate is called as the client module is being closed. Similar to ApplyContext, Validate is also a function overridden from the UIControl base class. This function is responsible for determining whether the data entered into the user-control is valid. Invalid data entered should result in **this.valid** to be set to **false**; this will prevent any further processing of the entered data. Conversely, assigning **this.valid** to **true** will allow the submitted data to be relayed back to the server module.

The bool value returned from **Validate** is used to determine if the currently displayed client module should be allowed to close. If false is returned from the **Validate** method, the currently displayed client module will remain visible.

18. Add the following **Serialize** function definition below the Validate method in the **uicMain** class.

```
public override string Serialize()
{
    //gather all the information and return the result string.
    //result string is received as sData parameter of //ResponderComponent.ProcessResponse()
    if ( this.m_Msg != null )
    {
        if ( this.m_Msg.Values["activity"] != null )
            this.m_Msg.Values.Remove( "activity" );

        this.m_Msg.Values.Add( "activity", this.cbxActivity.Text );

       if (this.m_Msg.Values["location"] != null)
            this.m_Msg.Values.Remove("location");

       this.m_Msg.Values.Add("location", this.cbxLocation.Text );

        if ( this.m_Msg.Values["date"] != null )
            this.m_Msg.Values.Remove( "date" );

        this.m_Msg.Values.Add( "date", this.dtpDate.Value );

        if ( this.m_Msg.Values["description"] != null )
            this.m_Msg.Values.Remove( "description" );

        this.m_Msg.Values.Add( "description", this.tbxSummary.Text );

       this.m_Msg.RequestAction = this.m_strRequestAction;

       // Display a message box for user to dismiss via the OK button.
       MessageBox.Show( "The information has been sent.",
                         "Info",
                         MessageBoxButtons.OK,
                         MessageBoxIcon.Asterisk,
                         MessageBoxDefaultButton.Button1
                        );

        // The value that was set in uicComment on the second form
        // is already contained inside the XmlWorkflowMessage. The value
        // was set in the Validate() method when the uicComment form
        // was closed.
        return this.m_Msg.Serialize();
    }

    //The message and stack trace of this exception will be written to the
    //client log file at \Program Files\OneBridge\AMP\AMPHost.log.
    //The administrator can retrieve this log file via the Mobile Office
    //Admin console.
    throw new Exception("Unable to collect XmlWorkflowMessage data.");
}
```

Serialize is the last function called in the client module before returning control to the server module. Serialize is also a function overridden from the UIComponent base class. The implementation above serializes the activity selected from the cbxActivity and cbxLocaiton ComboBox, date selected from the dtpDate DateTimePicker and description entered in the tbxSummary TextBox into a single string. The menu item that was selected is also saved in the XmlWorkflowMessage. The XmlWorkflowMessage is serialized to a string is relayed to the **sData** parameter of the **ProcessResponse** method in the server module.

The XmlWorkflowMessage class is used in this method to allow the same server module to be shared between the XML and .NET Compact Framework client Workflow package. Thus, eliminating the redundant task of creating two server modules with the same logic; differing only by their associated client module type (.NET Compact Framework or XML).

19. Add the following **menuNext_Click** method definition below the **Serialize** method in the **uicMain** class.

```
private void menuNext_Click( object sender, EventArgs e )
{
    // Get a reference to the UIComponent that represents the second form.
    // The Framework keeps a cache of all UIComponents loaded via
    // GetUIComponent.
    UIComponent _uic = Framework.GetUIComponent( typeof( uicComment ) );

    // Now make _uic the top UIComponent in the Framework's stack of
    // UIComponents
    _uic.BringToFront( _uic.Text,    // set the title of the form for the second page
                       false,        // no minimize box
                       true,         // show OK button
                       this.m_Msg    // our XmlWorkflowMessage that stores all values
                      );
}
```

menuNext_Click will be triggered when the Next submenu item is selected. The above implementation gets a reference of the second page user-control and sends it a reference of the XmlWorkflowMessage class member this.m_Msg. The second page is displayed with a BringToFront call. this.m_Msg is passed as a parameter in the BringToFront call for containing the data entered to the second page screen; thus, all the data entered to the client module is contained in a single XmlWorkflowMessage instance.

20. Add the following menu method implementations below the **Serialize** function in the **uicMain** class.

```
private void menuCancel_Click( object sender, EventArgs e )
{
    this.m_bCancel = true;
    this.Close();     // this will cause the Validate() method to get called
}

private void menuSubmit_Click( object sender, EventArgs e )
{
    this.m_bCancel = false;
    this.m_strRequestAction = "send"; // Control that caused submit
                                      // (Text matches action key in XML)
    this.Close();     // this will cause the Validate() method to get called
}
```

21. To instruct the Workflow package to call the appropriate method implementations when the menu items are selected, the methods created in the previous steps have to be linked with the Click events of the menu items. Switch back to the Design view of uicMain.cs and select the mainMenu. Select the menu item Submit and link the menuSubmit_Click method implementation to the Click event in the Properties Windows by switching to the Events view (indicated by the small flash icon) and selecting the menuSubmit_Click method from the drop down for the Click event. Repeat this for the menuNext_Click and the menuCancel_Click methods.

    This concludes the construction of the first page for the client module.

The following instructions describe the process of creating a second page to the client module, accessible via the Next submenu item on the first page main menu.

22. In the Solution Explorer, right-click the **WorkflowClient** project and select **Add | New Item**. From the Add New Item menu select **User Control** and name the file **uicComment.cs**.

23. Double-click on the **uicComment.cs** file from the Solution Explorer. In Design mode, right-click on the user-control and select **Properties**. Assign the following values:

    BackColor: ControlLightLight
    Name: uicComment
    Size: 240, 268

24. Using the Toolbox, add a ComboBox named cmbrmiType. In the properties of the ComboBox go to Items and add a collection of strings to populate the ComboBox. “Update control data from server”, “Throw exception from server”, “and “Return to previous screen”. These strings are the same ones used in the Xml Workflow package to allow the same server component to be used.

    Create a label beside the ComboBox; name the label lblrmiType and change its Text setting in properties to **Please select a RMI demonstration type:**.

25. Using the Toolbox, add a TextBox named **tbxComments**. In the properties of the TextBox, adjust the following values:

    Text:
    Multiline: True
    Size: 227, 152

26. Using the Toolbox, create a label beside on top of the tbxComments Textbox; name the label **lblPageTwo** and change its Text property to **Page 2 Comments:**.

27. Using the Toolbox, add a new ContextMenu named **mainMenu1**. In the Solution Explorer right click on uicComment.Designer.cs and select **View Code**. In InitializeComponent() change

    ```
    this.mainMenu = new System.Windows.Forms.ContextMenu();
    ```
    to
    ```
    this.mainMenu1 = new System.Windows.Forms.MainMenu();
    ```

    At the end of the file change

    ```
    private System.Windows.Forms.ContextMenu mainMenu;
    ```
    to
    ```
    private System.Windows.Forms.MainMenu mainMenu1;
    ```

    Save and close the uicComment.Designer.cs file and return to the Design view of uicComment.cs.

28. Create a left menu button named **menuBack** and set its Text property to **Back**. Create a right menu button named **menuRMI** and set its Text property to **RMI**.

29. At the top of the user-control source file, add the following code to import various namespaces used by the user-control.

    ```
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    ```



```
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using iAnywhere.OMAAT;
using iAnywhere.MobileOffice.AMP;
using ExtendedSystems.OneBridge.MobileObject;
```

30. Modify the **uicComment** class signature to inherit **UIComponent** instead of **UserControl**.

```
public partial class uicComment : UIComponent
{ }
```

31. Add the following **ApplyContext** method definition below the **uicComment()** constructor in the **uicComment** class.

```
protected override void ApplyContext()
{
    //reset the UI to a known good state
    Framework.BaseForm.Menu = this.mainMenu;
    this.tbxComments.Text = "";

    // Load message passed from page one
    // This was passed to us in the BringToFront(...) call
    // from the uicMain form
    XmlWorkflowMessage _msg = this.CustomContext as XmlWorkflowMessage;
    if ( _msg != null )
    {
        // Load comments if provided within the context for this form.
        if ( _msg.Values["comments"] != null )
        {
            this.tbxComments.Text = Convert.ToString( _msg.Values["comments"] );
        }
    }
}
```

The above ApplyContext() method is called when the second page user-control is loaded. This implementation loads the XmlWorkflowMessage object passed from page one and adjusts the menu bar accordingly. Next a check is performed to see if data has previously been entered to the tbxComments box (possibly from switching back and forth between the first page user-control and second page user-control) if so, the Textbox is loaded with this data.

32. Add the following **menuBack_Click** method definition below the **ApplyContext()** method in the **uicComment** class.

```
private void menuBack_Click( object sender, EventArgs e )
{
    //Save the comments into the context.
    XmlWorkflowMessage _msg = this.CustomContext as XmlWorkflowMessage;
    if ( _msg != null )
    {
        // Load comments if provided within the context for this form.
        if ( _msg.Values["comments"] != null )
        _msg.Values.Remove( "comments" );

        _msg.Values.Add("comments", this.tbxComments.Text);
    }

    // This would cause the Validate() method to get called.
    // If there is no Validate() override defined, the default
    // returned value is true (i.e. the form is allowed to be closed).
    this.Close();
}
```

The above implementation of **menuBack_Click** saves any data entered to the **tbxComments** box and closes the second page user-control.

33. Add the following **menuRMI_Click** method definition to the the **uicComment** class.

```
private void menuRMI_Click( object sender, EventArgs e )
{
   XmlWorkflowMessage _msg = this.CustomContext as XmlWorkflowMessage;

   if ( _msg != null )
   {
      // Setup Mobile Objects for RMI call
      MOConnection _conn = new MOConnection();
      //Get User Name
      string strUserName = MobileOfficeConfig.UserName;
      //Get Server Name
      string strServer = MobileOfficeConfig.ServerName;
      //Get Port
      ushort usPortNumber = MobileOfficeConfig.ServerPort;
      //Get ActivationCode
      string strActivationCode = MobileOfficeConfig.ActivationCode;
      string strCompanyID = MobileOfficeConfig.CompanyID;
      _conn.Init( strServer, usPortNumber, strCompanyID, strUserName,
                  true, strActivationCode, "OBPIM", "AMPACTIVITYSAMPLE" );

      MORequestOptions _req = new MORequestOptions();
      _req.ClientTimeout = 60; // 1 minute timeout;

      // Construct the params for the call with the values from the on screen controls
      XmlWorkflowMessage _msgRequest = new XmlWorkflowMessage();
      _msgRequest.WorkflowScreen = "WorkflowClient.uicComment";
      _msgRequest.Values.Add("rmitype", this.cmbrmiType.Text);
      _msgRequest.Values.Add("comments", this.tbxComments.Text);
      string _request = _msgRequest.Serialize();
      byte[] _bufEntryID = null;

      try
      {
```

```
         // Get the Credentials for the Workflow package
         string _strUsername = String.Empty;
         byte[] _bufPassword = null;
         WorkflowUtils.GetCredentials( "activity_credentials",  // specified in the
                                                                // manifest.xml file
                                    ref _strUsername,
                                    ref _bufPassword );

         // ModuleName is specified in the manifest.xml file, get the corresponding
         // ModuleID
         int _moduleID = WorkflowUtils.GetModuleID( "SampleActivitiesModule" );

         // send the RMI request to server
         AMPResponseProcessor _processor = new AMPResponseProcessor( _conn );
         string strResponse =
             _processor.ProcessSyncRequest( _moduleID,   // server assigned module ID
                                            1,           // version of Workflow package
                                                         // specified in manifest.xml file
                                            _conn.DeviceID,
                                            _request,     // serialized params
                                               _bufEntryID,// ID of e-mail message if known
                                               _strUsername,// username if Workflow package
                                                            // supports credentials
                                               _bufPassword, // password if Workflow package
                                                             // supports credentials
                                               _req );

         // parse the server's response
         XmlWorkflowMessage _msgResponse = XmlWorkflowMessage.CreateFromXml( strResponse );

         // Server asked us to jump back to the main screen;
         if ( _msgResponse.WorkflowScreen.Equals("main") )
         {
            menuBack_Click( sender, e );
         }

         // Update the comments from the server
         this.tbxComments.Text = Convert.ToString( _msgResponse.Values["comments"] );
      }
      catch ( MOException ex )
      {
         // Show server thrown exception
         MessageBox.Show( ex.Message,
                          "Exception",
                          MessageBoxButtons.OK,
                          MessageBoxIcon.Exclamation,
                          MessageBoxDefaultButton.Button1);
      }

      _conn.Close();
   }// if
}// menuRMI_click
```

The above implementation of menuRMI_Click connects to the server and makes a Remote Method Invocation (RMI) call to the responder component's implementation of ProcessSyncRequest. This method call blocks the Workflow form execution until a response is obtained from the server, or the request timesout.

The response from the responder component is parsed and used to update the screen controls or used for navigation. Utility methods in the WorkflowUtils class are used to obtain the Workflow package properties needed for the method call.

34. To instruct the Workflow package to call the appropriate method implementations when the menu items are selected the method created in the previous steps, they have to be linked with the Click event of the menu item. Switch back to the Design view of uicComment.cs and select the mainMenu. Select the menu item Back and link the menuBack_Click method implementation to the Click event in the Properties Windows by switching to the Events view (indicated by the small flash icon) and selecting the menuBack_Click method from the drop down for the Click event. Repeat for the RMI menu item.

35. Save the changes and build the project. Verify that the **WorkflowClient.dll** assembly is saved to the bin folder of the **WorkflowClient** project.

36. In the Solution Explorer, right-click the **WorkflowClient** project and select **Add | New Item**. From the Add New Item menu select **User Control** and name the file **uicLogin.cs**.

37. Double-click on the **uicLogin.cs** file from the Solution Explorer. In Design mode, right-click on the user-control and select **Properties**. Assign the following values:

    BackColor: ControlLightLight
    Name: uicLogin
    Size: 240, 268

38. Using the Toolbox, add a TextBox named **txtUsername**. In the properties of the TextBox, adjust the following values:

    Text:
    Size: 223, 21
    Tag: username

39. Using the Toolbox, create a label beside on top of the txtUsername Textbox; name the label **lblUsername** and change its Text property to **Username:**.

40. Using the Toolbox, add a TextBox named **txtPassword**. In the properties of the TextBox, adjust the following values:

    Text:
    Size: 223, 21
    Tag: password

41. Using the Toolbox, create a label beside on top of the txtUsername Textbox; name the label **lblPassword** and change its Text property to **Password:**.

42. In the previous steps, the values **username** and **password** specified in the Tag property of the text boxes are important. They help the Workflow package framework identify which controls should be treated as credentials and automatically persisted.

43. Using the Toolbox, add a new ContextMenu named **mainMenu**. In the Solution Explorer right click on uicComment.Designer.cs and select **View Code**. In InitializeComponent() change

```
this.mainMenu = new System.Windows.Forms.ContextMenu();
```
to
```
this.mainMenu2 = new System.Windows.Forms.MainMenu();
```

At the end of the file change

```
private System.Windows.Forms.ContextMenu mainMenu;
```
to
```
private System.Windows.Forms.MainMenu mainMenu2;
```

Save and close the uicLogin.Designer.cs file and return to the Design view of uicLogin.cs.

44. Create a left menu button named **menuOk** and set its Text property to **OK**.

45. At the top of the user-control source file, add the following code to import various namespaces used by the user-control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using iAnywhere.OMAAT;
using iAnywhere.MobileOffice.AMP;
```

46. Modify the **uicLogin** class signature to inherit **UIComponent** instead of **UserControl**.

```
public partial class uicLogin : UIComponent
{ }
```

47. Add the following **ApplyContext** method definition below the **uicLogin()** constructor in the **uicLogin** class.

```
protected override void ApplyContext()
{
   //reset the UI to a known good state
   Framework.BaseForm.Menu = mainMenu2;
   this.Valid = true;
}
```

The above ApplyContext() method is called when the Login page is needed. A login page will be shown to the user if the Workflow package requires credentials (specified in the manifest.xml file), and its credentials are expired. The server can also inform the Workflow form to prompt for new credentials by throwing a CredentialsRequestException. See server response component for details.

48. Add the following **Validate** method definition to the **uicLogin** class.

```
protected override bool Validate()
{
   //perform simple validation: if the username or password textbox are blank
   //do not allow the form to close
   if ( String.IsNullOrEmpty(txtUsername.Text) ||
        String.IsNullOrEmpty(txtPassword.Text) )
   {
      // the form is not allowed to close
      return false;
   }

   // All validation passed
   this.Valid = true;   // submit the data to server
   return true;         // the form is allowed to close

}
```

The above implementation does simple validation on the username and password textbox fields to make sure that values are filled in.

49. Add the following **menuOk_Click** method definition to the **uicLogin** class.

```
private void menuOk_Click(object sender, EventArgs e)
{
   // This would cause the Validate() method to get called.
   // If there is no Validate() override defined, the default
   // returned value is true (i.e. the form is allowed to be closed).
   this.Close();
 }
```

50. To instruct the Workflow package to call the appropriate method implementations when the menu items are selected the method created in the previous steps, they have to be linked with the Click event of the menu item. Switch back to the Design view of uicLogin.cs and select the mainMenu. Select the menu item OK and link the menuOk_Click method implementation to the Click event in the Properties Windows by switching to the Events view (indicated by the small flash icon) and selecting the menuOk_Click method from the drop down for the Click event.

51. Save the changes and build the project. Verify that the **WorkflowClient.dll** assembly is saved to the bin folder of the **WorkflowClient** project.

52. In the Solution Explorer, right-click the **WorkflowClient** project and select **Add | New Item**. From the Add New Item menu select **User Control** and name the file **uicActivate.cs**.

53. Double-click on the **uicActivate.cs** file from the Solution Explorer. In Design mode, right-click on the user-control and select **Properties**. Assign the following values:

    BackColor: ControlLightLight
    Name: uicActivate
    Size: 240, 268

54. Using the Toolbox, add a TextBox named **txtUsername**. In the properties of the TextBox, adjust the following values:

    Text:
    Size: 223, 21

Tag: username

55. Using the Toolbox, create a label beside on top of the txtUsername Textbox; name the label **lblUsername** and change its Text property to **Username:**.

56. Using the Toolbox, add a TextBox named **txtPassword**. In the properties of the TextBox, adjust the following values:

    Text:
    Size: 223, 21
    Tag: password

57. Using the Toolbox, create a label beside on top of the txtUsername Textbox; name the label **lblPassword** and change its Text property to **Password:**.

58. In the previous steps, the values **username** and **password** specified in the Tag property of the text boxes are important. They help the Workflow package framework identify which controls should be treated as credentials and automatically persisted.

59. Using the Toolbox, add a ComboBox named **cmbLanguage**. In the properties of the ComboBox go to Items and add a collection of strings to populate the ComboBox. For example add “English”, “German” and “Chinese”. Create a label beside the ComboBox; name the label **lblLangauge** and change its Text setting in properties to **Language:**

60. Using the Toolbox, add a new ContextMenu named **mainMenu**. In the Solution Explorer right click on uicComment.Designer.cs and select **View Code**. In InitializeComponent() change

    ```
    this.mainMenu = new System.Windows.Forms.ContextMenu();
    ```
    to
    ```
    this.mainMenu = new System.Windows.Forms.MainMenu();
    ```

    At the end of the file change

    ```
    private System.Windows.Forms.ContextMenu mainMenu;
    ```
    to
    ```
    private System.Windows.Forms.MainMenu mainMenu;
    ```

    Save and close the uicLogin.Designer.cs file and return to the Design view of uicActivate.cs.

61. Create a left menu button named **menuActivate** and set its Text property to **Activate**.

62. At the top of the user-control source file, add the following code to import various namespaces used by the user-control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Serialization;
using System.IO;
using iAnywhere.OMAAT;
using iAnywhere.MobileOffice.AMP;
```

63. Modify the **uicActivate** class signature to inherit **UIComponent** instead of **UserControl**.

```
public partial class uicActivate : UIComponent
{ }
```

64. Add the following **ApplyContext** method definition below the **uicActivate()** constructor in the **uicActivate** class.

```
protected override void ApplyContext()
{
   //reset the UI to a known good state
   Framework.BaseForm.Menu = mainMenu;
   this.Valid = true;
}
```

The above ApplyContext() method is called when the Activation page is needed. An Activation page will be shown to the user if the Workflow package requires activation the first time it is used (specified in the manifest.xml file).

After the Activation page is dismissed, the default screen for the Workflow package will be shown. If the Workflow package also requires credentials and they have not been entered the credentials screen will be shown to collect the credentials. The Activation page automatically saves credentials if the screen contains credentials controls. Recall the Workflow package framework treats controls with the Tag values of **username** or **password** as credential controls.

65. Add the following **Validate** method definition to the **uicActivate** class.

```
protected override bool Validate()
{
   //perform simple validation: if the username or password textbox are blank
   //do not allow the form to close
   if ( String.IsNullOrEmpty(txtUsername.Text) ||
        String.IsNullOrEmpty(txtPassword.Text) )
   {
      // the form is not allowed to close
      return false;
   }

   // All validation passed
   this.Valid = true;   // submit the data to server
   return true;         // the form is allowed to close

}
```

The above implementation does simple validation on the username and password textbox fields to make sure that values are filled in.

66. Add the following **menuItemActivate_Click** method definition to the **uicActivate** class.

```
private void menuItemActivate_Click(object sender, EventArgs e)
{
   // This would cause the Validate() method to get called.
   // If there is no Validate() override defined, the default
   // returned value is true (i.e. the form is allowed to be closed).
   this.Close();
}
```

67. To instruct the Workflow package to call the appropriate method implementations when the menu items are selected the method created in the previous steps, they have to be linked with the Click event of the menu item. Switch back to the Design view of uicActivate.cs and select the mainMenu. Select the menu item **Activate** and link the **menuItemActivate_Click** method implementation to the Click event in the Properties Windows by switching to the Events view (indicated by the small flash icon) and selecting the **menuItemActivate_Click** method from the drop down for the Click event.

68. Save the changes and build the project. Verify that the **WorkflowClient.dll** assembly is saved to the bin folder of the **WorkflowClient** project.

STEP 2 (ii): Create a XML Client Module (ActivitiesWorkflow.xml)

The following instructions guide readers through the process of creating a XML client module. A copy of the completed ActivitiesWorkflow.xml client module is located in the **C:\Program Files\iAnywhere Mobile Office\Tools\ Workflow Samples\Activities** directory.

Once deployed to the client device, the frameworks implemented in iAnywhere Mobile Office will parse the XML, construct its graphical user-interface items and display them to the screen. XML client modules are supported on Windows Mobile , Symbian and iPhone platforms. The iAnywhere Mobile Office framework will contruct the graphical user-interface elements for each platform using the native controls on each platform.

1. Using a text editor, create a file named **ActivitiesWorkflow.xml**.

2. Copy the following code into the **ActivitiesWorkflow.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
  <workflow>
    <screens default="main">
           <screen key="main" text="%ACTIVITY%" okaction="cancel" backcolor="pink">
           <actions default="send">
           <action key="send" text="%SEND%" type="submit"
                  onsubmit="%ONSUBMITTEXT%" onresubmit="%ONRESUBMITTEXT%"/>

          <action key="rmi" text="%RMI%" type="rmi" timeout="60"
                                              onerror="%REQUESTERROR%">
               <param key="activity" />
               <param key="date" />
               <param key="description" />
          </action>

               <action key="next" text="%NEXT%" type="open" target="WorkflowClient.uicComment"
                  />

               <action key="cancel" text="%CANCEL%" type="close" />
            </actions>
        </screen>
    </screens>
    <resources>
        <strings default="en-us">
          <lang key="en-us">
                <string key="ACTIVITY" value="Activity" />
                <string key="ACTIVITY2" value="Activity - Page 2" />
                  <string key="LOGIN" value="Login" />
                <string key="SEND" value="Send" />
                <string key="CANCEL" value="Cancel" />
                <string key="SWIM" value="I was swimming" />
                <string key="RUN" value="I was running" />
                <string key="SHOP" value="I was shopping" />
                <string key="WORK" value="I was working" />
                <string key="DATE" value="Activity date:" />
                <string key="SELECTLABEL" value="Please select an activity:" />
                <string key="SELECTLABEL2" value="Please select a location:" />
                <string key="SELECTLABEL3" value="Please select a RMI
    demonstration type:" />
                 <string key="TEXTLABEL" value="Comments and description:" />
                 <string key="COMMENTSLABEL" value="Page 2 Comments:" />
```



```
                <string key="NEXT" value="Next" />
                <string key="BACK" value="Back" />
                <string key="INFO" value="Notice" />
                 <string key="ONSUBMITTEXT" value="The information has been sent." />
                <string key="ONRESUBMITTEXT" value="The information has already been
                                    sent." />
                 <string key="USERNAMELABEL" value="Username:" />
                <string key="PASSWORDLABEL" value="Password:" />
                <string key="OK" value="OK" />
                <string key="REQUESTERROR" value="Error requesting data from server." />
                <string key="RMI" value="RMI" />
                <string key="UPDATE" value="Update control data from server." />
                <string key="EXCEPTION" value="Throw exception from server." />
                <string key="SCREEN" value="Return to previous screen." />
                <string key="ACTIVATE" value="Activation Screen"/>
                <string key="ACTMNU" value="Activate"/>
                <string key="LANGLBL" value="Language:"/>

          </lang>
        </strings>
    </resources>
</workflow>
```

The XML definition above creates a new screen on the client-side device with four menu items, Send, RMI, Next and Cancel. The next menu button will allow for navigation to the second page of the client module. The code enclosed by the `<resources>` and `</resources>` tag map strings to keywords that are used throughout the user-interface implementation. Localized text can be supplied for different languages by specifying an additional <lang> …</lang> section for the desired language. (See Xml Workflow Resources in the Workflow Developer Guide and Step 5 below). The completed sample demonstrates localized German text.

The default attribute on the screen tag denotes which screen is displayed first when this client module is loaded by Mobile Office.

3. Copy the following code into the **ActivitiesWorkflow.xml** file between the **</actions>** and **</screen>** tags.

```
<controls>
    <selectbox key="activity" value="o4"
        required="true" label="%SELECTLABEL%" labelpos="top">
        <option key="" value="" />
            <option key="o1" value="%SWIM%" />
            <option key="o2" value="%RUN%" />
            <option key="o3" value="%SHOP%" />
            <option key="o4" value="%WORK%" />
    </selectbox>
   <selectbox key="location" value=""
           required="false" label="%SELECTLABEL2%" labelpos="top">
   </selectbox>

    <datepicker key="date" value="today" label="%DATE%" labelpos="top" minvalue="today-7"
        maxvalue="today+7">
    </datepicker>
    <textbox key="description" value="" required="false"
        multiline="3" label="%TEXTLABEL%" labelpos="top">
    </textbox>
</controls>
```

The code above adds four controls to the user-interface. These controls are a ComboBox populated with 4 values (each displaying their associated strings mapped from the `<resources>` section), a second

ComboBox used to demonstrate setting named value pairs for the <selectbox> tag, a DateTimePicker and a TextBox. The controls are labeled with strings specified in the `<resources>` section.

4. Copy the following code into the **ActivitiesWorkflow.xml** file between the **</screen>** and **</screens>** tags.

```
<screen key="WorkflowClient.uicComment" text="%ACTIVITY2%">
    <actions default="back">
        <action key="back" text="%BACK%" type="save" />
      <action key="rmi" text="%RMI%" type="rmi" timeout="60" onerror="%REQUESTERROR%">
          <param key="rmitype" />
          <param key="comments" />
      </action>
    </actions>
    <controls>
         <selectbox key="rmitype" value="update"
                  required="true" label="%SELECTLABEL3%" labelpos="top">
             <option key="update" value="%UPDATE%" />
             <option key="exception" value="%EXCEPTION%" />
             <option key="screen" value="%SCREEN%" />
         </selectbox>
        <textbox key="comments" value="" required="false" multiline="6" label="%COMMENTSLABEL%"
           labelpos="top">
        </textbox>
    </controls>
</screen>
```

The code above implements a second page for the client module. The second page consists of a SelectBox and TextBox control with menu options to demonstrate an RMI call to the server and also to navigate back to the first page.

5. Copy the following code into the **ActivitiesWorkflow.xml** file between the **</lang>** and **</strings>** tags.

```
<resources>
<strings default="en-us">
<lang key="en-us">
            <string key="ACTIVITY" value="Activity" />
            <string key="ACTIVITY2" value="Activity - Page 2" />
            <string key="LOGIN" value="Login" />
            <string key="SEND" value="Send" />
            <string key="CANCEL" value="Cancel" />
            <string key="SWIM" value="I was swimming" />
            <string key="RUN" value="I was running" />
            <string key="SHOP" value="I was shopping" />
            <string key="WORK" value="I was working" />
            <string key="DATE" value="Activity date:" />
            <string key="SELECTLABEL" value="Please select an activity:" />
            <string key="SELECTLABEL2" value="Please select a location:" />
            <string key="SELECTLABEL3" value="Please select a RMI demonstration type:" />
            <string key="TEXTLABEL" value="Comments and description:" />
            <string key="COMMENTSLABEL" value="Page 2 Comments:" />
            <string key="NEXT" value="Next" />
            <string key="BACK" value="Back" />
           <string key="INFO" value="Notice" />
            <string key="ONSUBMITTEXT" value="The information has been sent." />
           <string key="ONRESUBMITTEXT" value="The information has already been sent." />
            <string key="USERNAMELABEL" value="Username:" />
            <string key="PASSWORDLABEL" value="Password:" />
            <string key="OK" value="OK" />
            <string key="REQUESTERROR" value="Error requesting data from server." />
            <string key="RMI" value="RMI" />
            <string key="UPDATE" value="Update control data from server." />
            <string key="EXCEPTION" value="Throw exception from server." />
            <string key="SCREEN" value="Return to previous screen." />
            <string key="ACTIVATE" value="Activation Screen"/>
            <string key="ACTMNU" value="Activate"/>
            <string key="LANGLBL" value="Language:"/>
 </lang>
 <lang key="de-de">
          <string key="ACTIVITY" value="Aktivität" />
          <string key="ACTIVITY2" value="Aktivität - Seite 2" />
          <string key="LOGIN" value="Logon" />
          <string key="SEND" value="Speichern" />
          <string key="CANCEL" value="Schliessen" />
          <string key="SWIM" value="Ich war schwimmen" />
          <string key="RUN" value="Ich war laufen" />
          <string key="SHOP" value="Ich war einkaufen" />
          <string key="WORK" value="Ich habe gearbeitet" />
          <string key="DATE" value="Datum:" />
          <string key="SELECTLABEL" value="Bitte wählen Sie eine Aktivität:" />
          <string key="TEXTLABEL" value="Kommentar und Beschreibung:" />
          <string key="COMMENTSLABEL" value="Kommentare Seite 2:" />
          <string key="NEXT" value="Weiter" />
          <string key="BACK" value="Zurück" />
          <string key="INFO" value="Info" />
          <string key="ONSUBMITTEXT" value="Ihre Eingaben wurden versendet." />
          <string key="ONRESUBMITTEXT" value="Ihre Eingaben wurden bereits gesendet." />
          <string key="USERNAMELABEL" value="Username:" />
          <string key="PASSWORDLABEL" value="Passwort:" />
          <string key="OK" value="OK" />
          <string key="REQUESTERROR" value="Es ist eine Fehler in der Kommunikation
                 mit dem Server aufgetreten." />
```

```
            <string key="RMI" value="RMI" />
            <string key="UPDATE" value="Controldaten via Server aktualisieren." />
            <string key="EXCEPTION" value="Eine Ausnahme auf dem Server auslösen." />
            <string key="SCREEN" value="Zum vorherigen Screen zurückkehren." />
            <string key="ACTIVATE" value="Aktivierung"/>
            <string key="ACTMNU" value="Aktivieren"/>
            <string key="LANGLBL" value="Sprache"/>
    </lang>
    </strings>
    </resources>
```

The code above adds German support for the client module. Each of the keywords mapped earlier to an English string have now been mapped to their equivalent German expressions. Depending on the language set on the client device, the XML user-interface will display the text for the appropriate language accordingly.

5. Copy the following code into the **ActivitiesWorkflow.xml** file between the **</screen>** and **</screens>** tags.

```
<screen key="WorkflowClient.uicLogin" text="%LOGIN%">
   <actions>
      <action key="ok" text="%OK%" type="save" />
   </actions>
   <controls>
      <textbox key="username" value=""
               required="true"
                           credential="username"
               label="%USERNAMELABEL%" labelpos="top">
      </textbox>
      <textbox key="password" value=""
               required="true"
                           credential="password"
               password="true" label="%PASSWORDLABEL%" labelpos="top">
      </textbox>
    </controls>
 </screen>
```

The code above defines a login screen to prompt for credentials. If at anytime the credentials are expired, this screen will be shown to the user before the default screen. See <CredentialsCache> tag in the Manifest file reference and <textbox> control from the Workflow Developer Guide for more information.

6. Copy the following code into the **ActivitiesWorkflow.xml** file between the **</screen>** and **</screens>** tags.

```
<screen key="WorkflowClient.uicActivate" text="%ACTIVATE%">
   <actions>
      <action key="activate" text="%ACTMNU%" type="save" />
   </actions>
   <controls>
      <textbox key="username" value=""
               required="true"
           credential="username"
               label="%USERNAMELABEL%" labelpos="top">
      </textbox>
      <textbox key="password" value=""
               required="true"
           credential="password"
               password="true" label="%PASSWORDLABEL%" labelpos="top">
      </textbox>
```

```
        <selectbox key="lang" value="en"
                label="%LANGLBL%"
                required="true">
           <option key="" value="" />
           <option key="en" value="English" />
           <option key="de" value="German" />
           <option key="cn" value="Chinese" />
        </selectbox>
     </controls>
  </screen>
```

The code above defines an activation screen for the Workflow package. The package denoted it required Activation in the manifest file with the <RequiresActivation> tag. An activation screen is only shown the first time a Workflow package that requires activation is invoked. If the defined activation screen contains credential controls (<textbox> with the credential attribute defined), the credentials are automatically saved to the Credentials Cache.

7. Save and close the ActivitiesWorkflow.xml file.

STEP 3: Package the Workflow Package

A Workflow package is packaged as a compressed zip file. The **manifest.xml** file which must sit at the root of the zip package is a description of how the zip contents are organized. Inside the manifest.xml file, it specifies matching rules, the location of the server module implementation, and the client module to deploy onto the mobile device. Matching rules are used to differentiate between regular emails and Workflow package email requests. The following instructions document the construction of a manifest.xml file.

A copy of the completed manifest.xml file is located in the **C:\Program Files\iAnywhere Mobile Office \Tools\Workflow Samples\Activities** directory.

1. Using a text editor, create a file named **manifest.xml**.

2. Copy the following code into the **manifest.xml** file.

```
<?xml version="1.0" encoding="utf-16"?>
<Manifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AMPManifest.xsd">
   <ModuleName>SampleActivitiesModule</ModuleName>
   <ModuleVersion>1</ModuleVersion>
   <ModuleDesc>AMP Sample - Activities Collection</ModuleDesc>
   <ModuleDisplayName>Activities Sample</ModuleDisplayName>
   <ClientIconIndex>35</ClientIconIndex>
   <InvokeOnClient>1</InvokeOnClient>
   <MarkProcessedMessages>1</MarkProcessedMessages>
   <DeleteProcessedMessages>0</DeleteProcessedMessages>
   <CredentialsCache key="activity_credentials">1</CredentialsCache>
   <RequiresActivation>1</RequiresActivation>

   <MatchRules>
      <SubjectRegExp>^Activity:.*</SubjectRegExp>
  </MatchRules>
</Manifest>
```

   The code above uses the regular expression **Activity:** followed by any set of characters as its matching rule.

3. Copy the following code into the **manifest.xml** file between the **</RequiresActivation>** and **<MatchRules>** tags.

```
<TransformPlugin>
   <File>WorkflowServer.dll</File>
   <Class>WorkflowServer.TransformerComponent</Class>
</TransformPlugin>

<ResponsePlugin>
   <File>WorkflowServer.dll</File>
   <Class>WorkflowServer.ResponderComponent</Class>
</ResponsePlugin>
```

   The above tags specify that the server module responsible for the transform of the e-mail and processing its response is implemented in the **WorkflowServer.dll** file. The file paths are relative to the zip package, so in the above example, WorkflowServer.dll resides at the root of the zip package.

4. Copy the following code into the **manifest.xml** file between the **</ResponsePlugin>** and **<MatchRules>** tags.

```
<ClientWorkflows>
    <WindowsMobileProfessional>
        <CFWorkflow>
            <File>Client\WorkflowClient.dll</File>
            <Class>WorkflowClient.uicMain</Class>
          <CredentialScreen>WorkflowClient.uicLogin</CredentialScreen>
          <ActivationScreen>WorkflowClient.uicActivate</ActivationScreen>
        </CFWorkflow>
    </WindowsMobileProfessional>
    <WindowsMobileStandard>
        <XMLWorkflow>
            <File>Client\ActivitiesWorkflow.xml</File>
          <CredentialScreen>WorkflowClient.uicLogin</CredentialScreen>
          <ActivationScreen>WorkflowClient.uicActivate</ActivationScreen>
        </XMLWorkflow>
    </WindowsMobileStandard>
    <Symbian>
        <XMLWorkflow>
            <File>Client\ActivitiesWorkflow.xml</File>
          <CredentialScreen>WorkflowClient.uicLogin</CredentialScreen>
          <ActivationScreen>WorkflowClient.uicActivate</ActivationScreen>
        </XMLWorkflow>
    </Symbian>
   <iPhone>
      <XMLWorkflow>
         <File>Client\ActivitiesWorkflow.xml</File>
         <CredentialScreen>WorkflowClient.uicLogin</CredentialScreen>
         <ActivationScreen>WorkflowClient.uicActivate</ActivationScreen>
      </XMLWorkflow>
   </iPhone>
</ClientWorkflows>
```

The **<ClientWorkflows>** tag specifies the client module to deploy to the mobile device. In this tutorial, we associate the .NET Compact Framework client module with the Windows Mobile PocketPC device, and the XML client module with the Windows Mobile Smartphone , Symbian and iPhone platforms. The file paths are relative to the zip package, so in the above example, WorkflowClient.dll and ActivitiesWorkflow.xml reside in a folder named **Client** at the root of the zip package.

5. Copy the following code into the **manifest.xml** file between the **</ClientWorkflows>** and **<MatchRules>** tags.

```
<ContextVariables>
   <ContextVariable>
      <Name>OutputFolder</Name>
      <Value>C:\ActivitiesSampleOutput</Value>
      <Password>false</Password>
   </ContextVariable>
   <ContextVariable>
      <Name>Activity</Name>
      <Value>I was running</Value>
      <Password>false</Password>
   </ContextVariable>
    <ContextVariable>
        <Name>Location</Name>
        <Value>|Loc1$Location 1|Loc2$Location 2|Loc3$Location 3</Value>
        <Password>false</Password>
```

```
   </ContextVariable>
</ContextVariables>
```

The **<ContextVariables>** tag provides developers the freedom of sending string variables to a server module without having to recompile the project. In this tutorial, we specify the string value **C:\AMPSampleOutput** to be set to variable **OutputFolder**. This value was used by the **_SaveResponse** method in our **WorkflowServer.dll** for outputting our client module data to a text file. In addition, we specify the string value **I was running** to be set to variable **Activity**. This value is used by the ProcessMail method in the TransformerComponent class to populate the Activities ComboBox on the client module. Finally, we populate the Location ComboBox with named value pairs with the **Location** context variable. When a Workflow package is deployed, Administrators have the option of overwriting the values of the context variables via the Mobile Office Admin console.

6. Save and close the manifest.xml file.

   This concludes the configuration required in our manifest.xml file. Next, we will need to package our server module (WorkflowServer.dll) and client module (ActivitiesWorkflow.xml or WorkflowClient.dll) according to what was specified in the manifest.xml file.

   The manifest.xml file should be at the same directory level as WorkflowServer.dll.

   If during *STEP 2: Create the Client Module* a .NET Compact Framework client module was created then WorkflowClient.dll need be located in the directory **Client** where Client is a directory of equal level to the manifest.xml file.

   Similarly, if during *STEP 2: Create the Client Module* a XML client module was created then ActivitiesWorkflow.xml need be located in the directory **Client** where Client is a directory of equal level to the manifest.xml file.

7. Using the cursor, highlight the manifest.xml file, WorkflowServer.dll file and Client directory.

8. Right click the group of selected files, and select **Compressed (zipped) Folder** from the **Send To** menu. A .zip file will be generated; the name of this file can be assigned to any desired string. Let's call it **ActivitiesPackage.zip**.

STEP 4: Deploy and Assign Users to Workflow Package

1. Open iAnywhere Mobile Office Admin. In the **Workflows** tab, select the **Add...** button located on the left of the screen. Locate the **ActivitiesPackage.zip** file generated from *STEP 3: Package the Workflow Package* and select **Open**.

2. From the **Users** tab in the iAnywhere Mobile Office Admin console, highlight a user you wish to register to the Workflow package and select **Edit Settings...** followed by **Workflows...**

   From the checklist menu, select *Activities Sample*. Select **OK** from each submenu screen until you have returned to the iAnywhere Mobile Office Admin main menu screen.

STEP 5: Run the Tutorial on a Client Device

Running the Workflow package on a client device is the final task of this tutorial. Our server module, client module, and iAnywhere Mobile Office configurations have now been built and configured to run our Workflow package.

1. On the Mobile Office server, create a new directory on the C:\ drive named **ActivitiesSampleOutput**. Recall, this value was specified in the manifest.xml file as a context variable.

2. If a .NET Compact Framework client module was created during *STEP 2: Create the Client Module* then the following OneBridge Mobile Office cab file should be installed to the Windows Mobile 5.0 Pocket PC device.

   C:\Program Files\iAnywhere Mobile Office\Bin\Clients\OneBridgeSetup_ppc.cab

   Otherwise, the following OneBridge Mobile Office cab file should be installed to the Windows Mobile 5.0 Smart Phone device.

   C:\Program Files\iAnywhere Mobile Office\Bin\Clients\OneBridgeSetup_sp.cab

   When prompted for a server name and credentials, specify the server used in *STEP 4: Deploy and Assign Users to Workflow Packages* and the user registered to the Workflow package.

3. Send an email request to the iAnywhere Mobile Office user. This may be accomplished by logging into the email account of the iAnywhere Mobile Office user specified in instruction 2 and composing a new email. The email must be addressed to the iAnywhere Mobile Office user and the email subject must begin with **Activity:** followed by any set of characters.

   The client will shortly receive an email in their device's Inbox labeled with a unique inbox icon (specified in the Workflows tab of iAnywhere Mobile Office Admin console) and the subject entered in the server-side email client.

4. On the client device, select the Workflow package request email. The client module created in *STEP 2: Create the Client Module* will appear.

5. Populate the items on the client module as desired and click the **Send** button.

6. On the Mobile Office server, open the **C:\ActivitiesSampleOutput** directory. A newly generated text file containing the information sent from the client device will be present. Open this text file and compare its contents against those typed in the client module.