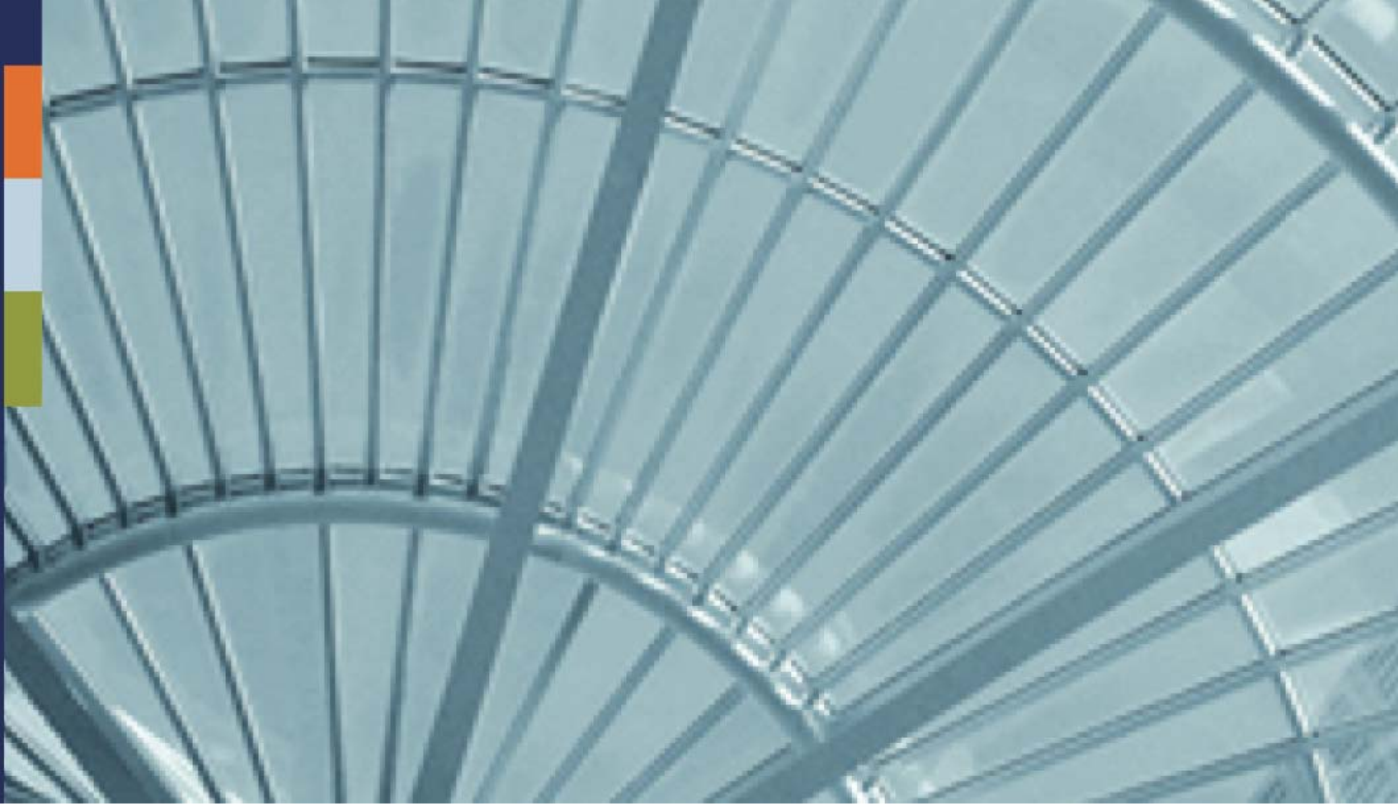


Workflow Package Developer Guide

iAnywhere® Mobile Office 5.7

SYBASE
iAnywhere

Document ID: DC01175-01-0570-01
LAST REVISED: *September 2009*

Copyright and Trademarks

Disclaimer

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

Copyright and Trademarks ii
Disclaimer ii
Technical Support ii
Table of Contents iii
Welcome 1
Solution Overview 1
Server-side Development 3
IMailProcessor Interface 3
IMailProcessor.ProcessMail Method 3
IResponseProcessor Interface 4
IResponseProcessor.ProcessResponse Method 4
ISynchronousRequestProcessor Interface 6
ISynchronousRequestProcessor.ProcessSyncRequest Method 6
iAnywhere.MobileOffice.AMP.TransformRetryException Class 7
iAnywhere.MobileOffice.AMP.ResponseRetryException Class 8
iAnywhere.MobileOffice.AMP.TransformIgnoreException Class 8
iAnywhere.MobileOffice.AMP.TransformDropException Class 9
iAnywhere.MobileOffice.AMP.CredentialRequestException Class 9
iAnywhere.MobileOffice.AMP.ContextData Class 12
iAnywhere.MobileOffice.AMP.MailData Class 13
iAnywhere.MobileOffice.AMP.VariableBag Class 14
iAnywhere.MobileOffice.AMP.eDeviceType Enumeration 14
iAnywhere.MobileOffice.AMP.XmlWorkflowMessage Class 15
XmlWorkflowMessage Constructor 15
XmlWorkflowMessage.CreateFromXml Method 16
XmlWorkflowMessage.Serialize Method 16
XmlWorkflowMessage.ToString Method 17
iAnywhere.MobileOffice.AMP.MessageValueCollection Class 17
MessageValueCollection.Add Method 18

MessageValueCollection.Clear Method 19
MessageValueCollection.Remove Method 19
Client-side Development 20
.NET Compact Framework Workflow Package for Windows Mobile 20
Debugging of .NET Compact Framework Workflow Packages 20
Workflow Package Signing Requirements 20
Methods to Implement 21
Using two or more screens in your .NETCF Workflow package 22
iAnywhere.OMAAT.Framework Class 22
iAnywhere.OMAAT.UIComponent Class 24
XML Workflow Package Overview 30
XML Reference 35
XML Workflow Package Screens 35
XML Workflow Package Actions 36
Xml Workflow Package Resources 39
Xml Workflow Form Controls 40
Workflow Package Packaging 48
Available Icons 48
The Manifest.xml File 54
Manifest.xml File Reference 55
Activities Tutorial 62
BodyOfText .... Error! Bookmark not defined.

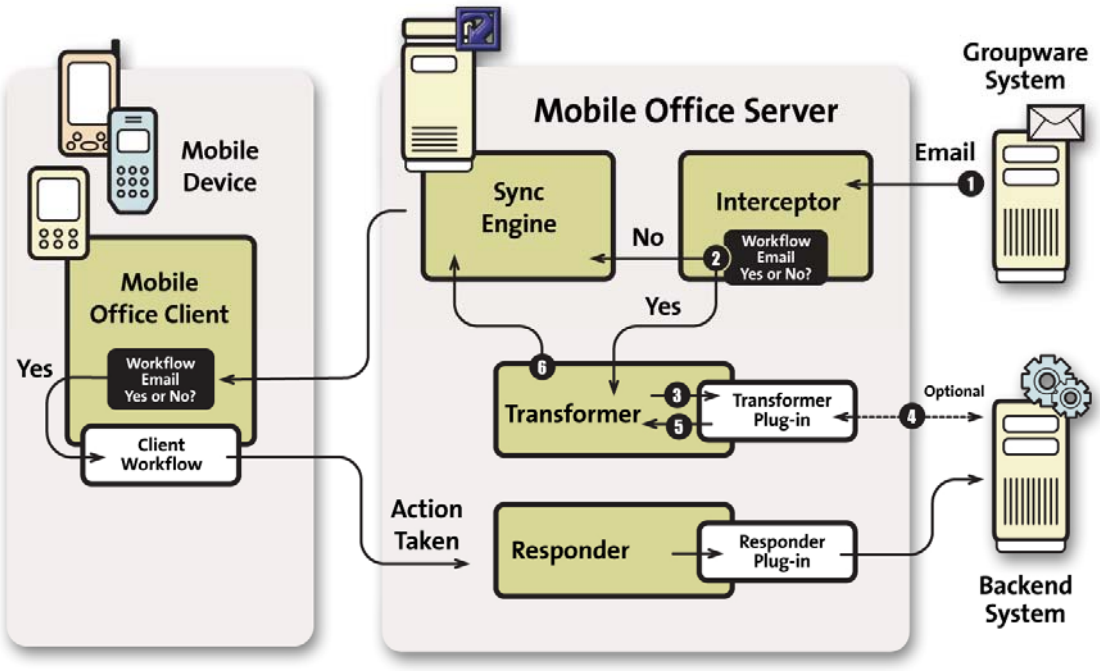
Welcome

Welcome to iAnywhere® Mobile Office. iAnywhere Mobile Office is specifically designed for today's mobile business workforce. It combines fully integrated wireless email and PIM with on-device security and business process mobilization.

This document will introduce you to the Business Process Workflow packages. After the study of this document, you will understand the general concept of Workflow packages and their development.

Solution Overview

iAnywhere Mobile Office supports the extension of the E-Mail Inbox with Business Process Workflow packages. The image below shows the Mobile Office server architecture and the interaction with the Mobile Office Client when e-mail delivered to a mobile device is transformed into a Workflow package.



Any e-mail delivered to a mobile device can be transformed into a business process request on the server; this allows mobile workers to directly interact with backend systems.

In addition, it is also possible to declare a Workflow package to be available on the device for user-initiated invocation. In this case no e-mail is transformed on the server side; the mobile worker is able to initiate a business process from within his E-Mail Inbox anywhere at anytime.

A Workflow package on the device is created using an easy to learn cross platform XML description. On Windows Mobile operated devices it is also possible to use the .NET Compact Framework technology for development.

On the server side the iAnywhere Mobile Office Server integrates with existing enterprise backend systems using Web Services or their native protocols. Any system can be integrated by using the .NET Framework technology from within the Mobile Office Server.

The deployment of a Workflow package is completely manageable through the iAnywhere Mobile Office administration console and does not require any manual device interaction from the administrator. Once a Workflow package is deployed into an existing installation, the administrator can configure the package and assign it to any active user in the system.

A Workflow package consists of the server components that are targeted towards the specific enterprise backend systems and components for the targeted device platforms.

The following chapters describe how to develop a Workflow package and how to prepare it for deployment. The configuration and deployment steps are described within the *Administrator Guide* of the iAnywhere Mobile Office Server under the Help button on the administration console and also located at Start > Programs > iAnywhere Mobile Office > Mobile Office Docs > Administrator Guide.

Server-side Development

A Workflow package requires an integration module on the server side. This module is responsible for processing the e-mails identified by configured matching rules (see <MatchRules>) and processing the responses sent to the server from the device. The developer must implement logic to process the data contained in the e-mail to enable the device to display the data.

The server-side integration module must be implemented as a .NET class library. The class library must contain a class that implements certain interfaces. These interfaces define the methods that are called during processing of an e-mail message and during the processing of the response coming back from the device.

IMailProcessor Interface

IMailProcessor.ProcessMail Method

C#

```
public void ProcessMail( ContextData oContextData,
                         MailData oMailData, out string sData )
```

Visual Basic

```
Sub ProcessMail(ByVal oContextData As ContextData,
                ByVal oMailData As MailData, ByRef sData As String)
```

By implementing this method, your server module will be able to preprocess the e-mail message before it is delivered to the device and add any contextual data to the message.

Parameters

oContextData

Additional information about the device, user, and application that this e-mail message is intended for plus any context variables from the manifest.xml file if defined.

oMailData

Information about the actual e-mail message

sData

Additional contextual data to send along with the e-mail message down to the device. This value is stored as a custom field in the e-mail message on the device and is made available to the client module in the `ApplyContext()`method.

Return Value

None

Remarks

Except for the special exceptions described below, exceptions thrown from this method are logged to the Application log and can be viewed from the iAnywhere Mobile Office administration console. If such an exception is thrown here during the processing of an e-mail, it will not be processed further and will not be delivered to the device.

The Transformer plugin may throw an exception of type TransformRetryException that specifies the number of seconds to wait before retrying the current item. Instead of logging an error for the item and removing it from the queue, the Transformer will call the plugin again for this item after the retry time expires. See also TransformDropException, TransformIgnoreException, and CredentialRequestException.

If a call to ProcessMail times out, the engine logs an error to the Workflow Error List. The engine then reschedules the item to retry after a designated delay period. This retry behaves the same as if a retry exception were thrown from the Transform plugin. By default, the timeout period is ten minutes, and the retry interval is double the timeout period. Both of these values may be overridden in the registry. For more information about configuring these values in the registry, see the *iAnywhere Mobile Office Administrator Guide* from the Start menu or from the Help button in the administration console.

IResponseProcessor Interface

IResponseProcessor.ProcessResponse Method

C#

```
public void ProcessResponse( ContextData oContextData, string sData )
```

Visual Basic

```
Sub ProcessResponse(ByVal oContextData As ContextData,
                    ByVal sData As String)
```

By implementing this method, your server module will have access to the information serialized from the client module. This information can be deserialized and entered into different back end systems and processes.

Parameters

oContextData

Additional information about the device, user, and application that this e-mail message is intended for, plus any context variables from the manifest.xml file if defined.

sData

The serialized string sent from the Workflow package deployed on the client.

Return Value

None

Remarks

Except for the special exceptions described below, exceptions thrown from this method are logged to the Application log and can be viewed from the iAnywhere Mobile Office administration console. If such an exception is thrown here during the processing of a response, it will not be processed further and will not be delivered to the device.

The Responder plugin may throw an exception of type ResponseRetryException that specifies the number of seconds to wait before retrying the current item. Instead of logging an error for the item and removing it from the queue, the Responder will call the plugin again for this item after the retry time expires.

See also CredentialRequestException type.

If a call to ProcessResponse times out, the engine logs an error to the Workflow Error List. The engine then reschedules the item to retry after a designated delay period. This retry will behave the same as if a retry exception were thrown from the Responder plugin. By default, the timeout period is ten minutes and the retry interval is double the timeout period. Both of these values may be overridden in the registry. For more information about configuring these values in the registry, see the *iAnywhere Mobile Office Administrator Guide* from the Start menu or from the Help button within the administration console.

ISynchronousRequestProcessor Interface

ISynchronousRequestProcessor.ProcessSyncRequest Method

C#

```
public string ProcessSyncRequest( ContextData oContextData, string sData )
```

By implementing this method, your server module will be able to process synchronous method calls from the Workflow package deployed on the client.

Parameters

oContextData

Information about the device, user, and application that this request is associated with plus any context variables from the manifest.xml file if defined.

sData

The serialized string sent from the Workflow package deployed on the client.

Return Value

When invoked from XML, the return value is a modified version of the XmlWorkflowMessage passed as an argument.

When invoked from a Compact Framework plugin, the return value is application-dependent content.

Remarks

Note that this interface must be implemented by the same plugin class that is registered as a Responder plugin. As a result, a Responder plugin class must implement IResponseProcessor, as before, and must implement ISynchronousRequestProcessor only if the synchronous message processing feature is being used by the Workflow form. If a Workflow form calls the synchronous processing mechanism and the corresponding Responder plugin does not implement ISynchronousRequestProcessor, an exception will be thrown back to the caller.

Any exception thrown by the ProcessSyncRequest will be passed to the caller on the device.

If a call to ProcessSyncRequest times out, the engine will throw an exception to the client. The timeout period is the same period used for ProcessResponse.

See also information on the client side about the <action> tag's type attribute "rmi".

iAnywhere.MobileOffice.AMP.TransformRetryException Class

Used to signal to the Transform engine that the current item should be retried after a designated period of time. See the Remarks section above.

C#

```
public class TransformRetryException
```

Inheritance Hierarchy

```
System.Exception
     TransformRetryException
```

The following table lists the members exposed by the `TransformRetryException` type.

Public Constructors

Name	Description
`TransformRetryException`	Takes a single argument of type int. This argument indicates the number of seconds the Transformer should wait before trying to process the current queue item again.

iAnywhere.MobileOffice.AMP.ResponseRetryException Class

Used to signal to the Responder engine that the current item should be retried after a designated period of time. See the Remarks section above.

C#

```
public class ResponseRetryException
```

Inheritance Hierarchy

```
System.Exception
     ResponseRetryException
```

The following table lists the members exposed by the `ResponseRetryException` type.

Public Constructors

Name	Description
`ResponseRetryException`	Takes a single argument of type int. This argument indicates the number of seconds the Responder should wait before trying to process the current queue item again.

iAnywhere.MobileOffice.AMP.TransformIgnoreException Class

An exception class thrown from the ProcessMail method that alters the way the current item from the transform queue is handled. If the TransformIgnoreException is thrown, the item currently being processed as a Workflow message is reverted to appear as a normal email on the device and is no longer associated with any Workflow package.

C#

```
public class TransformIgnoreException
```

Inheritance Hierarchy

```
System.Exception
     TransformIgnoreException
```

The following table lists the members exposed by the `TransformIgnoreException` type.

Public Constructors

Name	Description
`TransformIgnoreException`	Takes no arguments.

iAnywhere.MobileOffice.AMP.TransformDropException Class

An exception class thrown from the ProcessMail method that alters the way the current item from the transform queue is handled. If the TransformDropException is thrown, the item currently being processed as a Workflow message is not delivered to the device, and the device will not see the email.

C#

```
public class TransformDropException
```

Inheritance Hierarchy

```
System.Exception
     TransformDropException
```

The following table lists the members exposed by the `TransformDropException` type.

Public Constructors

Name	Description
`TransformDropException`	Takes no arguments.

iAnywhere.MobileOffice.AMP.CredentialRequestException Class

This exception is thrown either from the ProcessMail method, the ProcessResponse method, or the ProcessSyncRequest method. The exception signals the engine to send a request to the client user to enter new credentials for the item currently being processed. All items in the queue following this item for the same module and device are suspended until a reply is received from the client. When the reply is received, the plugin is called again for the same item with the new credentials provided in the ContextData object.

Note: This exception may be thrown only if the package configuration manifest shows SupportCredentialsCache as enabled. See Enable Credentials Support in manifest.xml *below.*

C#

```
public class CredentialRequestException
```

Inheritance Hierarchy

```
System.Exception
     CredentialRequestException
```

The following table lists the members exposed by the `CredentialRequestException` type.

Public Constructors

Name	Description	
CredentialRequestException	Takes 3 arguments of type string. These are as follows:	
	sRequestSubject	The Subject text to be associated with the request; appears in the email message to the user.
	sRequestFrom	The From text to be associated with the request; appears in email message to the user. Does not have to be legitimate email address; can be arbitrary text.
	sRequestTransformData	Serialized XmlWorkflowMessage. Optional.

Enable Credentials Support in manifest.xml

To enable credentials support, the following tags must be added to the manifest.xml:

- <CredentialsCache>, at the same level as <ModuleName>, must be added to indicate that Credentials support is enabled (value set to "1"). The attribute "key" must also be set, which allows all versions of the same module to share the same set of credentials, or related modules to share the same set of credentials.

  Example:
  `<CredentialsCache key="MyCredentialsKey">1</CredentialsCache >`

- Each Workflow package definition must also have a new element indicating the name or class of the form to be invoked on the device to request the credentials from the user. Inside each <XMLWorkflow> tag, a new <CredentialScreen> tag must be added whose value is the name of the credentials form within the XML Workflow. Inside each <CFWorkflow> tag, a new <CredentialScreen> tag must be added, whose value is the name of the class implementing the credentials form.

Example:

```
<WindowsMobileProfessional>
        <CFWorkflow>
            <File>ClientPlugins\PPC\plugin.dll</File>
            <Class>my.namespace.ClientPlugin</Class>
            <CredentialScreen>my.namespace.CredentialsForm</CredentialScreen>
        </CFWorkflow>
</WindowsMobileProfessional>
<WindowsMobileStandard>
        <XMLWorkflow>
             <File>ClientPlugins\Smartphone\plugin.xml</File>
            <CredentialScreen>MyCredentialsForm</CredentialScreen>
        </XMLWorkflow>
</WindowsMobileStandard>
```

iAnywhere.MobileOffice.AMP.ContextData Class

Represents additional information associated with a server module of a Workflow package.

The following tables list the members exposed by the ContextData type

Public Properties

Name	**Description**
DeviceName	This is a unique string identifying the device. This identifier is a hardware identifier that comes from the physical device and will not change over time for the same physical device.
DeviceType	An enumeration value indicating what type of device.
DeviceId	An integer identifying the device for which the current message is being processed. This is an internal ID generated by the Mobile Office system to identify a device during registration. A single physical device may change its internal DeviceId if it is re-registered.
MessageId	The unique identifier of the email being processed. MessageId is presented as a byte array. In a groupware context, this is the ID used by the sync engine to uniquely identify a PIM record in the groupware backend server (*e.g.,* the Domino UNID or Exchange ID). In the context of a message inserted by the Injection API, this is the MessageId specified by the caller of the InjectMail method. The byte array is the ASCII byte encoding of the MessageId string passed to InjectMail. This MessageID may be used by Responder and Transformer plugins to map any given Workflow message to a response for that message.
ModuleVersion	An integer identifying the version of the module as specified in the manifest file.
UserName	A string identifying the Mobile Office user name.
Variables	A collection of context variables defined in the file. Administrators may have edited the values of context variables from the Mobile Office administration console.
BackEndUser	If the Workflow package requires credentials, this field contains the backend user name entered by the package user in the user credential field.
BackEndPassword	If the Workflow package requires credentials, this field contains the backend user password entered by the package user in the password credential field.

iAnywhere.MobileOffice.AMP.MailData Class

Represents an e-mail message.

The following table lists the members exposed by the MailData type.

Public Properties

Name	**Description**
Body	A string representing the message text body.
CC	A string representing the carbon-copied recipients for the e-mail message.
CustomFields	Domino only. Provides values for fields XCSCustom1 through 10 if you are using a custom template or custom read formulas to populate those fields.
From	A string representing the recipient who sent the message.
ReceivedDate	The date that the message was received.
Subject	A string representing the message subject.
To	A string representing the primary recipients.

iAnywhere.MobileOffice.AMP.VariableBag Class

Represents a collection of context variables as specified in the manifest.xml file. Administrators may have edited the values of context variables from the Mobile Office Admin console.

The following tables list the members exposed by the VariableBag type

Public Properties

Name	**Description**
Item	Gets the value of the context variable associated with the specified key.

Public Methods

Name	**Description**
GetVariableNames	Gets a string array containing all the variable names in the VariableBag.

iAnywhere.MobileOffice.AMP.eDeviceType Enumeration

Specifies the type of device

Members

Member name	**Description**
WindowsMobileProfessional	Windows Mobile Professional (WM6) or Pocket PC (WM5) device.
WindowsMobileStandard	Windows Mobile Standard (WM6) or Smartphone (WM5) device.
Symbian	Symbian device
iPhone	iPhone device
Unknown	Unknown device type

iAnywhere.MobileOffice.AMP.XmlWorkflowMessage Class

Represents a common xml message format used between server and client modules. This class cannot be inherited.

XmlWorkflowMessage Constructor

Syntax

C#

```
public XmlWorkflowMessage()
```

Visual Basic

```
Public Sub New()
```

The following tables list the members exposed by the XmlWorkflowMessage type

Public Properties

Name	Description
Header	Defines a header for this message. The intention is this is used to transport data from the server to the device and vice versa, but the data is not modified on the device (e.g., used as UID for state).
Values	MessageValueCollection contains all values within this message.
RequestAction	The key value of the Workflow form's submit or rmi action.
WorkflowScreen	The key value of the Workflow form's screen. This property can be set by server components to direct the client to show a different Workflow package screen. This value is reported by the Workflow form during a submit or rmi action. For an XML Workflow package, WorkflowScreen should be set to the key attribute of the screen. For .NETCF Workflow packages, WorkflowScreen should be set to a classname in the .NETCF Workflow form that implements UIComponent.

Public Methods

Name	**Description**
CreateFromXml	Create an XmlWorkflowMessage instance from an XML string. Often the XML string is created with Serialize.
Serialize	Serializes this XmlWorkflowMessage instance into a string format recognized by both server and client module.
ToString	Overridden. Formats the string representation of this instance.

XmlWorkflowMessage.CreateFromXml Method

Create an XmlWorkflowMessage instance from a XML string. Often the XML string is created with Serialize.

Syntax

C#

```
public static XmlWorkflowMessage CreateFromXml(string sXml)
```

Visual Basic

```
Public Shared Function CreateFromXml(ByVal sXml As String) As
XmlWorkflowMessage
```

Parameters

sXml – The xml to create the XmlWorkflowMessage from.

Return Value

A newly created XmlWorkflowMessage instance.

XmlWorkflowMessage.Serialize Method

Serializes this XmlWorkflowMessage instance into a string format recognized by both server and client modules.

Syntax

C#

```
public string Serialize()
```

Visual Basic

```
Public Function Serialize() As String
```

XmlWorkflowMessage.ToString Method

Formats the string representation of this instance

Syntax

C#

```
public override string ToString()
```

Visual Basic

```
Public Overrides Function ToString() As String
```

iAnywhere.MobileOffice.AMP.MessageValueCollection Class

Represents a collection of MessageValues inside an XmlWorkflowMessage. This class cannot be inherited.

The following tables list the members exposed by the MessageValueCollection type

Public Properties

Name	Description
Count	The number of values in the collection.
Item	Returns the value based on key.
Keys	Returns an array of contained keys.
Values	Returns an array of contained objects.

Public Methods

Name	Description
Add	Overloaded. Adds a new value to the collection.
Clear	Removes all contained values from this collection.
Remove	Removes the value from this collection.

MessageValueCollection.Add Method

Overload List

Name	Description
Add (String, Boolean)	Adds a new Boolean value to the collection.
Add (String, DateTime)	Adds a new DateTime value to the collection.
Add (String, Decimal)	Adds a new Decimal value to the collection.
Add (String, Double)	Adds a new Double value to the collection.
Add (String, Int32)	Adds a new int value to the collection.
Add (String, String)	Adds a new string value to the collection.

Adds a new value to the collection

Syntax

C#

```
public void Add(string sKey, bool bValue)

public void Add(string sKey, DateTime dateValue)

public void Add(string sKey, decimal decValue)

public void Add(string sKey, double dblValue)

public void Add(string sKey, int iValue)

public void Add(string sKey, string sValue)
```

Visual Basic

```
Public Sub Add(ByVal sKey As String, ByVal bValue As Boolean)

Public Sub Add(ByVal sKey As String, ByVal dateValue As Date)

Public Sub Add(ByVal sKey As String, ByVal decValue As Decimal)

Public Sub Add(ByVal sKey As String, ByVal dblValue As Double)

Public Sub Add(ByVal sKey As String, ByVal iValue As Integer)

Public Sub Add(ByVal sKey As String, ByVal sValue As String)
```

sKey

The key associated with the value

bValue, dateValue, decValue, dblValue, iValue, sValue

The value to add to the collection

Remarks

Throws ArgumentException if key already exists.

Throws NotSupportedException if key is null or empty.

MessageValueCollection.Clear Method

Removes all contained values from this collection

Syntax

C#

```
public void Clear()
```

Visual Basic

```
Public Sub Clear()
```

MessageValueCollection.Remove Method

Removes the value associated with sKey from this collection.

Syntax

C#

```
public void Remove(string sKey)
```

Visual Basic

```
Public Sub Remove(ByVal sKey As String)
```

Parameters

sKey – The key used to lookup object

Remarks

If sKey is not found, this method does nothing to the collection.

Client-side Development

The client side component for a Workflow package is commonly implemented using a cross platform XML description. On Windows Mobile devices the client component can also be implemented as a .NET Compact Framework class library. Regardless of the type chosen for implementation of the client side component, iAnywhere Mobile Office defines two different types of Workflow packages. One type requires an e-mail message for the user to trigger the transformation on the server side (either via a groupware backend or the InjectionAPI); the other type allows the user to start the process on the device and submit data to the server. The implementation of the client side component is in either case the same.

.NET Compact Framework Workflow Package for Windows Mobile

A client side module implemented with .NET Compact Framework (.NETCF) technology is simply a custom user control inside a regular .NETCF assembly. Therefore, it is possible to drag and drop with the Visual Studio designer and toolbox to design your user interface. Any built-in device control that is part of Visual Studio can be used in the design of your UI. Since the client module is simply a .NET user control, normal .NETCF UI development practices still apply here. Tasks such as handling of the soft keys for five way navigation and screen rotation remain responsibilities of the developer.

Once the UI has been designed to your satisfaction, include iAnywhere.OMAAT.dll to your application and inherit your class from `iAnywhere.OMAAT.UIComponent` instead of `System.Windows.Forms.UserControl` which the Visual Studio wizard puts in by default.

The iAnywhere.OMAAT.dll can be found in your installation at \Program Files\iAnywhere Mobile Office\Tools\MO_SDK\DotNet. Follow the correct folder for your target platform.

.NET Compact Framework Workflow packages are hosted within a background process running on the device. Once a Workflow package is launched for the first time, it is cached to enhance its performance on subsequent requests by the user. Therefore multiple Workflow package versions with the same type name (e.g., namespace.type) can not be running concurrently on the same device. A previous version needs to be removed from the system before a later version can be deployed and executed successfully. A device restart might be necessary.

Debugging of .NET Compact Framework Workflow Packages

To debug a Workflow package on a device within the context of iAnywhere Mobile Office, the background process hosting all Workflow packages provides command line options that can be used to install and launch a Workflow package from within the Visual Studio development environment. The executable hosting all Workflow packages is called AMPHost.exe and is located in the installation directory of Mobile Office on the device (i.e., \Program Files\OneBridge\AMPHost.exe).

```
amphost.exe –debug "\workflow.dll" "full–type-name" [[Y|N] "context data"]
```

Y indicates if the specified Workflow package should be installed as a user visible Workflow package. If defined, the package will be listed within the Workflows option menu of the users e-mail inbox.

Workflow Package Signing Requirements

Client modules that are implemented as .NET Compact Framework Workflow packages need to be signed with a Privileged Certificate in order for them to function correctly with Mobile Office.

Methods to Implement

By overriding three methods defined in `iAnywhere.OMAAT.UIComponent`, your user control will be able to:

- Receive input from your server module
- Validate user input
- Asynchronously send collected input back to the server module

iAnywhere.OMAAT.UIComponent.ApplyContext method

```
protected override void ApplyContext()
```

Override `ApplyContext` method to receive input from server module.

Your client module will be able to get the value of the custom context data that your server module set in the `IMailProcessor.ProcessMail` method.

The value of the custom context is stored as part of the e-mail message on the device and provides context to your Workflow package for that particular e-mail. Therefore it is important to limit the size of this custom context data as it will increase the storage requirements of messages on the device.

This method is called before your UI is displayed.

iAnywhere.OMAAT.UIComponent.Validate method

```
protected override bool Validate()
```

Override `Validate` method to validate data before submitting to server.

Your client module can let Mobile Office know whether the info contained in the displayed UI should be submitted to back to the server module or not.

The boolean return value indicates to Mobile Office whether the screen currently being displayed should be closed or not. To close the screen, return true, otherwise return false.

To indicate the data in the UI is valid and should be sent back to the server module, set the `this.Valid` property to `true`. Otherwise if you don't wish for the data in the UI to be sent back to the server module, set this.`Valid` property to `false`.

E.g. if true is returned from the Validate method, but this. `Valid` is set to false, the screen will be allowed to close, but the data in the UI is not sent back to the server module.

This method is called before your UI is closed.

iAnywhere.OMAAT.UIComponent.Serialize method

```
public override string Serialize()
```

Override `Serialize` method to asynchronously submit collected input back to the server module.

Your client module will be able to asynchronously submit the collected data from your UI as a string back to the server module for processing. The string will be queued locally on the device until Mobile Office is able to obtain a network connection. Therefore, this operation can be performed while offline.

`Serialize` will only be called if the `this.Valid` property was set to `true` in the `protected override bool Validate()` method.

If only .NETCF Workflow packages are targeted, the format of the string to use for serialization and deserialization is determined by the individual developer, but the format needs to be agreed upon between server and client modules.

To enable XML and .NETCF Workflow packages to be able to share an implementation of a server module, developers should consider using the `iAnywhere.MobileOffice.AMP.XmlWorkflowMessage` class to serialize and deserialize strings between server modules and client modules.

The Activities sample Workflow package demonstrates the use of the `iAnywhere.MobileOffice.AMP.XmlWorkflowMessage` class to share a single server module implementation between an XML Workflow package and a .NETCF Workflow package. This sample project can be found in your installation at \Program Files\iAnywhere Mobile Office \Tools\ Workflow Samples\Activities.

This method is called after your UI is closed.

Using two or more screens in your .NETCF Workflow package

Each screen in your .NETCF client module should be a custom user control that inherits from `iAnywhere.OMAAT.UIComponent` instead of `System.Windows.Forms.UserControl`.

To navigate to another user control that inherits from `iAnywhere.OMAAT.UIComponent` and display it, use the methods from the `iAnywhere.OMAAT.Framework` class.

Example:

```
UIComponent _uicPage2 = Framework.GetUIComponent( typeof( uicPage2 ) );
_uicPage2.BringToFront();
```

iAnywhere.OMAAT.Framework Class

This class is the main entry point for every Workflow package. The class defines all base methods for controlling the screen flow of an application. This is a static class (Module in Visual Basic) and so cannot be inherited or instantiated.

The following tables list the members exposed by the Framework type.

Public Properties

Name	**Description**
BaseForm	The Framework uses only one Form during the whole runtime of an application. This property allows access to this Form.

Public Methods

Name	**Description**
GetCurrentUIComponent	Retrieves a reference to the UIComponent currently at the top of the stack of UIComponents shown to the user.

GetUIComponent	Overloaded. Retrieves a reference to the instance of the requested UIComponent out of the Framework cache.

Framework.GetCurrentUIComponent Method

Retrieves a reference to the UIComponent currently at the top of the stack of UIComponents shown to the user.

Syntax

C#

```
public static UIComponent GetCurrentUIComponent()
```

Visual Basic

```
Public Shared Function GetCurrentUIComponent() As UIComponent
```

Framework.GetUIComponent Method

Overload List

Name	**Description**
GetUIComponent (Type)	Retrieves a reference to the instance of the requested UIComponent out of the Framework cache.
GetUIComponent (Type, Boolean)	Retrieves a reference to the instance of the requested UIComponent out of the Framework cache and creates a new instance of the Type if specified.

Syntax

C#

```
public static UIComponent GetUIComponent(Type uicType)
public static UIComponent GetUIComponent(Type uicType, bool newInstance)
```

Visual Basic

```
Public Shared Function GetUIComponent(ByVal uicType As Type) As UIComponent

Public Shared Function GetUIComponent(ByVal uicType As Type,
                                      ByVal newInstance As Boolean) As UIComponent
```

Parameters

uicType

The System.Type of the UIComponent that is requested.

newInstance

Specifies whether the Framework will return a new instance instead of a previously cached instance. If a new instance is requested, the previously cached instance is removed and the new instance is cached instead.

iAnywhere.OMAAT.UIComponent Class

UIComponent is the base class for all .NET CF Workflow packages. The Framework maintains a stack of UIComponents.

Syntax

C#

```
public class UIComponent : UserControl
```

Inheritance Hierarchy

```
System.Object
    System.MarshalByRefObject
        System.ComponentModel.Component
            System.Windows.Forms.Control
                System.Windows.Forms.ScrollableControl
                    System.Windows.Forms.ContainerControl
                        System.Windows.Forms.UserControl
                            iAnywhere.OMAAT.UIComponent
```

Visual Basic

```
Public Class UIComponent Inherits UserControl
```

The following tables list the members exposed by the UIComponent type.

Public Constructors

Name	**Description**
UIComponent	Initializes a new instance of the UIComponent class.

Public Properties

Name	**Description**
AutoScroll	Overridden. The property gets interpreted whenever a UIComponents gets positioned on top of the stack of UIComponents shown to the user. Its value is applied to the Framework.BaseForm form.
Text	Overridden. Gets or sets the text associated with this UIComponent.
Valid	The value of this property is evaluated during the close of a UIComponent.

Protected Properties

Name	**Description**
CustomContext	Gets the context object defined in the BringToFront call to the UIComponent.

Public Methods

Name	**Description**
BringToFront	Overloaded. The method puts the UIComponent on top of the stack of UIComponents shown to the user.
Close	Calling the method will close the UIComponent.
Delete	Calling the method will close the UIComponent and removes it from the Framework cache.
GetNamedControls	This method returns a Dictionary containing all controls with a unique name of the UIComponent in a flatten hierarchy using their Name property as the key.
GetTaggedControls	This method returns a Dictionary containing all controls having a Tag specified not equal null of the UIComponent in a flatten hierarchy using their Tag value as the key.

SendToBack	This method closes the UIComponent.
Serialize	This method gets called by the Framework if Validate returned true.

Protected Methods

Name	Description
ApplyContext	The method gets called by the Framework after a screen orientation change happened.
Validate	The method gets called by the Framework before the UIComponent gets closed.

UIComponent.ApplyContext Method

The method gets called by the Framework whenever the UIComponent changes its context.

Syntax

C#

```
protected virtual void ApplyContext()
```

Visual Basic

```
Protected Overridable Sub ApplyContext()
```

Remarks

The method gets called before the UIComponent is shown to the user.

UIComponent.BringToFront Method

Overload List

Name	Description
BringToFront	The method puts the UIComponent on top of the stack of UIComponents shown to the user.
BringToFront (String, Boolean, Boolean)	The method puts the UIComponent on top of the stack of UIComponents shown to the user.
BringToFront (String, Boolean, Boolean, Object)	The method puts the UIComponent on top of the stack of UIComponents shown to the user.

BringToFront (String, Boolean, Boolean, Object, Boolean)	The method puts the UIComponent on top of the stack of UIComponents shown to the user.

Syntax

C#

```
public void BringToFront(
   string formTitle,
   bool minimizeBox,
   bool allowClosing
)

public void BringToFront(
   string formTitle,
   bool minimizeBox,
   bool allowClosing,
   object context
)

public void BringToFront(
   string formTitle,
   bool minimizeBox,
   bool allowClosing,
   object context,
   bool hideBelow
)
```

Visual Basic

```
Public Sub BringToFront()

Public Sub BringToFront(
   ByVal formTitle As String,
   ByVal minimizeBox As Boolean,
   ByVal allowClosing As Boolean
)

Public Sub BringToFront(
   ByVal formTitle As String,
   ByVal minimizeBox As Boolean,
   ByVal allowClosing As Boolean,
   ByVal context As Object
)
Public Sub BringToFront(
   ByVal formTitle As String,
   ByVal minimizeBox As Boolean,
   ByVal allowClosing As Boolean,
   ByVal context As Object,
   ByVal hideBelow As Boolean
)
```

Parameters

formTitle

Defines the Text property of the Framework.BaseForm form.

minimizeBox

Defines the MinimizeBox property of the Framework.BaseForm form.

allowClosing

Defines if it should be possible for the user to close this UIComponent using the upper right OK sign visible when minimizeBox is set false.

context

A context to be passed to the UIComponent and its delegates.

hideBelow

Set the hideBelow value to true to let the Framework remove all UIComponents currently shown to the user from the view. However, the hidden UIComponents are still in the stack of UIComponets shown to the user.

UIComponent.Close Method

Calling the method will close the UIComponent.

Syntax

C#

```
public void Close()
```

Visual Basic

```
Public Sub Close()
```

Remarks

After this method executes, the next UIComponent in the stack will be shown.

UIComponent.Delete Method

Calling the method will close the UIComponent and remove it from the Framework cache.

Syntax

Visual Basic (Declaration)

```
Public Sub Delete()
```

C#

```
public void Delete()
```

Remarks

After this method executes, the next UIComponent in the stack will be shown. In addition, the UIComponent will be removed from the cache of UIComponents maintained by the Framework.

UIComponent.GetNamedControls Method

This method returns a Dictionary containing all controls in the .NETCF Workflow package with a unique name, using their Name property as the key.

Syntax

Visual Basic (Declaration)

```
Public Function GetNamedControls() As Dictionary(Of String, Control)
```

C#

```
public Dictionary<string, Control> GetNamedControls()
```

UIComponent.GetTaggedControls Method

This method returns a Dictionary containing all the controls in the .NETCF Workflow package with the Tag property not equal to null. The Tag property's value is used as the key.

Syntax

C#

```
public Dictionary<object, Control> GetTaggedControls()
```

Visual Basic

```
Public Function GetTaggedControls() As Dictionary(Of Object, Control)
```

UIComponent.SendToBack Method

This method closes the UIComponent

Syntax

C#

```
public void SendToBack()
```

Visual Basic

```
Public Sub SendToBack()
```

UIComponent.Serialize Method

This method gets called by the Framework if Validate returned true

Syntax

C#

```
public virtual string Serialize()
```

Visual Basic

```
Public Overridable Function Serialize() As String
```

Return Value

A string that is asynchronously sent back to the server module for processing

Remarks

Server and client modules should agree on the format of the string used for communication. Consider using the XmlWorkflowMessage class to share an implementation between Xml and .NETCF Workflow packages.

UIComponent.Validate Method

This method gets called by the Framework before the UIComponent gets closed.

Syntax

C#

```
protected virtual bool Validate()
```

Visual Basic

```
Protected Overridable Function Validate() As Boolean
```

Return Value

If one decides to return false, the close operation gets interrupted and the Serialize method for the UIComponent is not called. i.e. When false is returned, the current .NETCF Workflow package remains displayed.

XML Workflow Package Overview

iAnywhere Mobile Office provides developers the ability to create client side components using an easy to learn XML based form description language. It is possible to describe multiple screens, their contained controls as well as the specific actions possible to execute from within a single screen. It is also possible to embed text resources into the xml description to allow the client side component to automatically change its language based on the language used by the operating system (see also Resources).

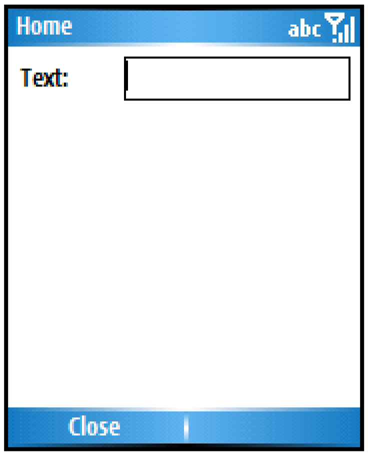
A basic example of such a component is shown below.

```
<?xml version="1.0" encoding="utf-8" ?>
<workflow>
  <screens>
    <screen key="screen1" text="Home">
```

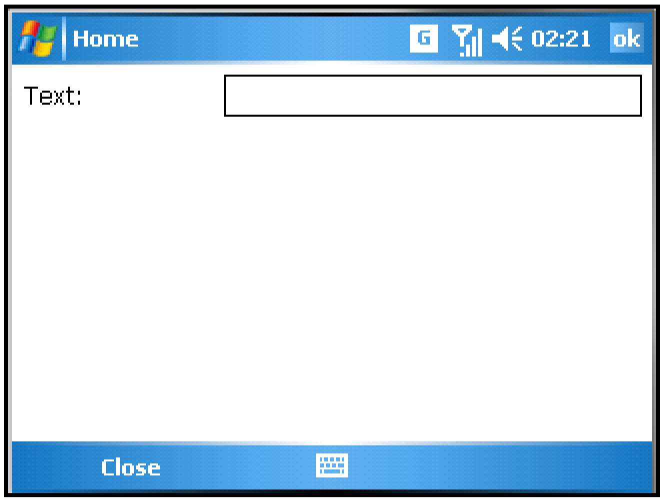
```
            <actions>
              <action key="action1" type="close" text="Close" />
            </actions>
            <controls>
              <textbox key="tbx1" label="Text:" />
            </controls>
          </screen>
        </screens>
      </workflow>
```

The example xml defines a single screen containing a single control, a text input control, and a single action. A client component can define a virtually unlimited number of screens, and each screen can contain a virtually unlimited number of actions. Depending on the type of control (see also Controls) a single screen can hold multiple controls or a single control. Each screen, action and control defined within the client component has to contain a unique key used to identify the item. If multiple screens or multiple actions within a single screen are specified, the default screen (the first screen shown to a user) and the default action need to be specified (see also Screens and Actions).

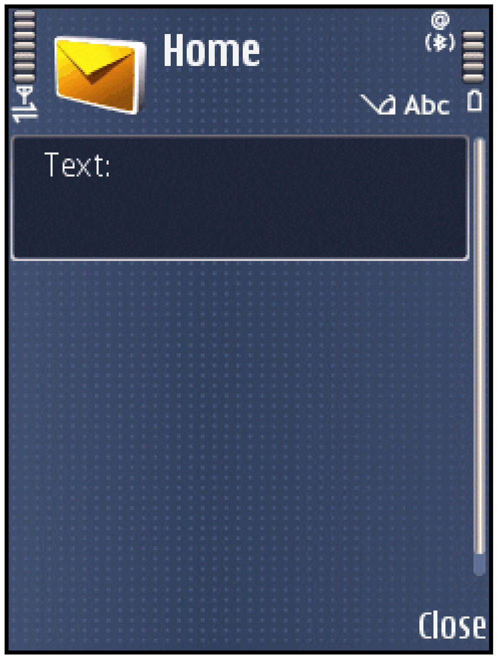
As shown in the screenshots below, the basic example displays a single textbox for user input on any supported target device platform. On Windows Mobile devices, the label for an input control is by default always on the left side and takes one third of the screen width, whereas the input control takes the other two thirds of the screen width. This default behavior can be overridden by using additional attributes to the specified control (see also Controls). On a Nokia S60 3rd Edition device the label is always displayed on top of the user input control.



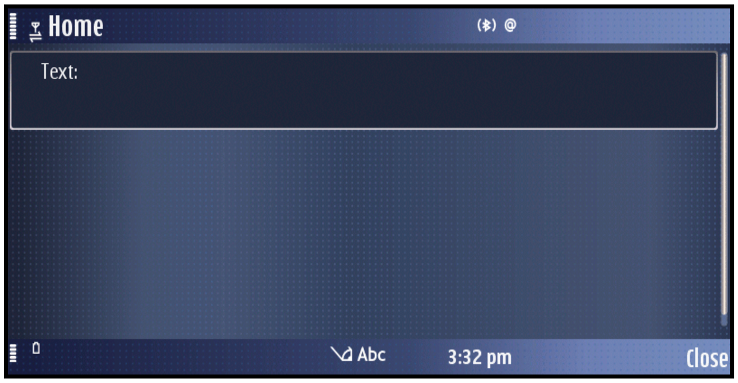
Windows Mobile Standard



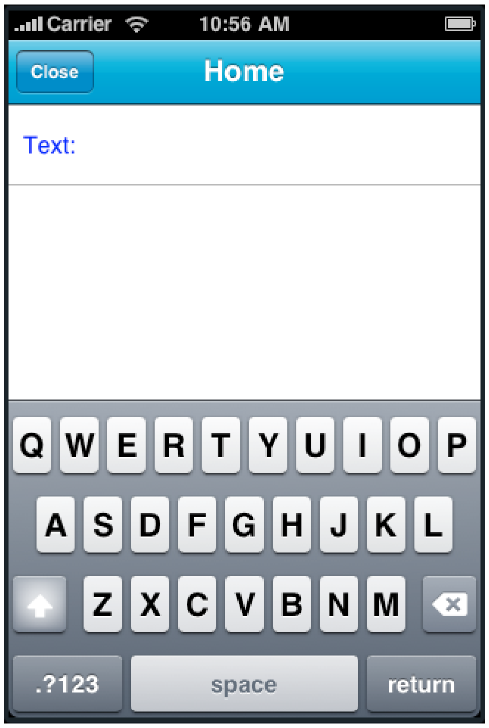
Windows Mobile Professional (landscape)



Nokia S60 3rd Edition (portrait)



Nokia S60 3rd Edition (landscape)



iPhone

In this example, selecting the *Close* menu item would result in a complete close of the form. Any user input would be disregarded. This behavior is specified using the type attribute of the action tag (see also Actions). Selecting [OK] in the upper right corner on the Windows Mobile Professional device would result in a submission of any user input to the server using a message as described within the XmlWorkflowMessage class (see also Screens on how to control this behavior). Using the concept of actions also enables the user to navigate between different screens defined in a client side component.

It is possible to define values as well as input validation criteria for controls using additional attributes (see also Controls). If a client component is bound to an e-mail message transformed by the server side, it is also possible to send such values to the client using the XmlWorkflowMessage class (see also XmlWorkflowMessage).

See below for an example that defines multiple screens, the navigation between them and a text input control with a predefined value and a validation criteria.

```
<?xml version="1.0" encoding="utf-8" ?>
<workflow>
  <screens default="screen1">
    <screen key="screen1" text="Home">
      <actions default="action1">
        <action key="action1" type="close" text="Close" />
        <action key="action2" type="open" text="Help"
                target="screen2" />
      </actions>
      <controls>
        <textbox key="num1" label="Number:" value="125"
                 required="true" numeric="true"
```

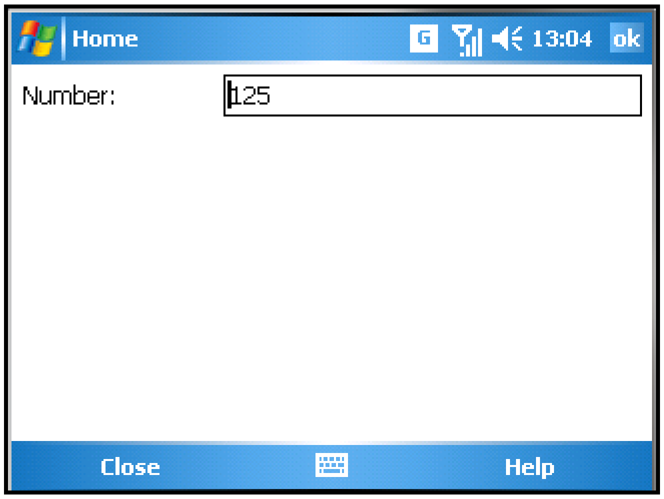
```
                    minvalue="0" maxvalue="250"
                    onerror="The value is out of range." />
            </controls>
          </screen>
          <screen key="screen2" text="Help">
            <actions>
              <action key="action3" type="cancel" text="Back" />
            </actions>
            <controls>
              <htmlview key="html">
                <![CDATA[
                  <html>
                  <b>This example</b> is used to demonstrate
                  the input validation critera’s of a
                  <i>textbox</i> control, as well as the
                  navigation between two dependent <i>screens
                  </i> using the <i>open</i> and <i>cancel
                  action types</i>.<br><br> The <i>textbox</i>
                  control contained in <i>screen1</i> only
                  allows <i>numeric</i> input in the range of
                  <i>0</i> to <i>250</i>.<br><br>The user can
                  navigate from <i>screen1</i> to <i>screen2
                  </i> using <i>action2</i> and go back using
                  <i>action3</i> on <i>screen2</i>.
                  </html>
                ]]>
              </htmlview>
            </controls>
          </screen>
        </screens>
      </workflow>
```

These screenshots show the results displayed to the user using above xml description on a Windows Mobile Professional device.

The default screen is shown to the user upon launch of the client side component.

The text input control is set to its defined value of *125*.

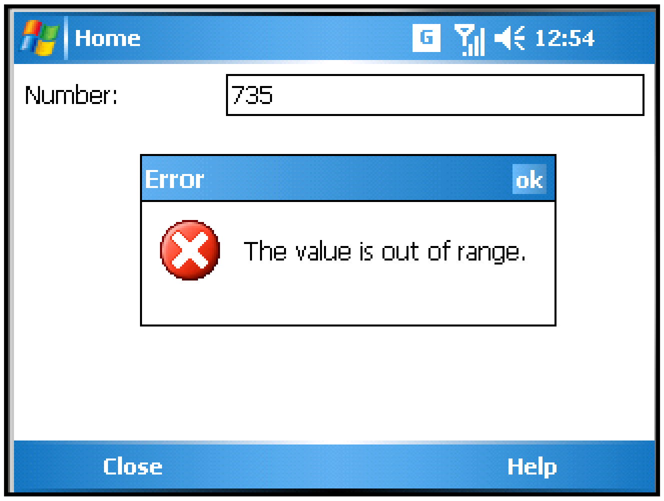
As defined two menu options are visible to the user, *Close* and *Help*.



Number: 125

The user changed the predefined value to the number *735*.

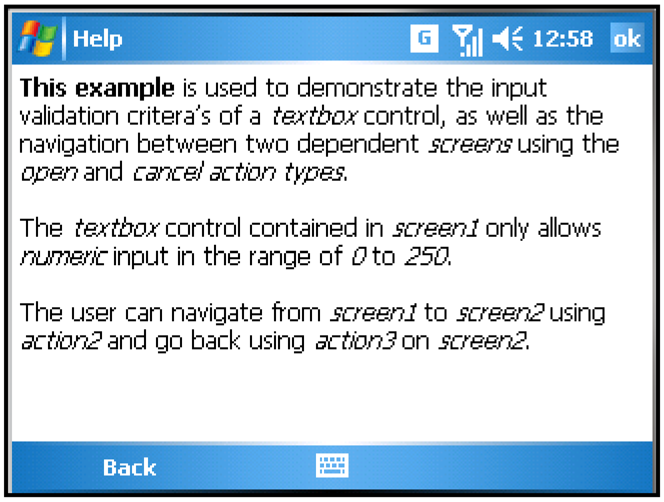
If the user selects [OK], the screen will be validated. As defined an error message is displayed to the user, containing the defined error message.



The value is out of range.

If the user selects the *Help* menu option, the second screen is displayed. In this case an html page is displayed.

Note: The htmlview control can not be combined with controls of different types on a single screen (see also [Controls](#)).



This example is used to demonstrate the input validation critera's of a *textbox* control, as well as the navigation between two dependent *screens* using the *open* and *cancel action types.*

The *textbox* control contained in *screen1* only allows *numeric* input in the range of *0* to *250.*

The user can navigate from *screen1* to *screen2* using *action2* and go back using *action3* on *screen2.*

XML Reference

XML Workflow Package Screens

Each client side component has to specify at least one screen. The navigation between different screens is described in the [Actions](#) section.

The available screens are described within the <screens>…</screens> section of the client-side xml description. If multiple screens are defined, the default screen needs to be specified using the default attribute of the <screens> tag, for example:

<screens default="screen1">…<screens>

A list of <screen></screen> sections within the <screens></screens> section describes the individual screens. Each <screen> has to contain a unique key defined using the key attribute and a title, defined using the text attribute (the text attribute allows the reference of a string resource, see also [Resources](#)), for example:

<screen key="screen1" text="Home">…</screen>

Note: On Windows Mobile Professional devices and Windows Mobile Standard devices with an [OK] hardware key, a special attribute on screen level can be used to specify the default behavior of the [OK] action, for example:

<screen key=”screen1” text=”Home” okaction=”cancel”>…</screen>

By default, if the user selects the [OK] option a screen performs a save action (see also Actions). This behavior can be changed for each screen by defining the okaction attribute.

Backcolor Attribute

For Windows Mobile devices only, the color of the screen can be specified by defining the backcolor attribute. The possible colors are black, blue, brown, gray, green, orange, pink, purple, red, white, and yellow.

```
<screen key=”screen1” text=”Home” backcolor=”blue”>
```

If a custom color is required, you can define one using the red, green, and blue values of the color. The format is #rrggbb where rr, gg, and bb are the hexadecimal values for the red, green and blue. The value 0 means no color while ff (255 in decimal) is the highest value.

```
<screen key="screen1" text="Home" backcolor="#999900">
```

XML Workflow Package Actions

Each screen has to specify its available actions to allow the user to submit data to the server, to close the client or to navigate between different screens of the client side component. A single screen can define an unlimited number of actions. The defined actions are displayed as menu options. Using the text attribute, the displayed menu item text can be defined (the text attributes allows the reference of a string resource, see also Resources).

There are six different action types defined.

```
<action key=”action1” type=”open” text=”Help” target=”help” save=”true”/>
```

The open action type allows the user to open a different screen; the screen to open must be specified using the target attribute of the action. The screen to open is placed on top of the current screen. A user can open an unlimited number of screens using this action type, but all open screens are kept open in a stack until the user navigates back using one of the following action types.

```
<action key=”action4” type=”save” text=”Save” />
```

The save action type allows the user to close the current open screen and save the input made to the collection of response values to be sent to the server using the submit action. Using the save action type a user can navigate back one single screen at a time. Before a screen is closed and the user gets returned back to the underlying screen, the data entered by the user is validated using the specified input validation criteria of the controls contained on the current screen. If the validation fails the user will not be able to navigate back using the save action type. If the option is available, the user can use a cancel action to discard all entered data and return to the previous screen without saving. The save action type operates similar to the submit action type and actually results in exactly the same behavior when executed on the default screen of a client component.

```
<action key=”action5” type=”cancel” text=”Cancel” />
```

The cancel action type allows the user to close the current open screen without any data validation occurring. The cancel action type navigates the user back to the underlying screen immediately. If executed on the default screen of a client component, the cancel action results in exactly the same behavior as the close action type.

```
<action key="action2" type="submit" text="Submit" onsubmit="" onresubmit=""/>
```

The submit action type validates the entered data of all open screens and saves it. The collected data is sent to the server side component as an XmlWorkflowMessage. If the user has multiple screens open at the time of the submit execution, and one of the underlying screens fails validation, the user is prompted with the configured warning message and the submit process stops. If the action type is submit and the onsubmit attribute is present and filled in, the value for the attribute is displayed in a message box with an OK button when the submit finishes (the onsubmit attribute allows the reference of a string resource. See also Resources). A title can be specified for the message box by defining the %INFO% resource (default: “Info”).

```
<action key="action3" type="close" text="Close" />
```

The close action type discards any changes to any open screens and allows the user to directly leave the client side component no matter which screen the client side component currently shows.

```
<action key="action6" type="rmi" text="RMI" timeout="60" onerror="Error Message">
      <param key="key1" />
      <param key="key2" />
      <param key="key3" />
</action>
```

The RMI action type allows users to send and receive Workflow package data to and from the server. In the example above, key1, key2, and key3 correspond to the key values of 3 unique client side Workflow form controls. During an RMI call, the value stored in each of the named controls in the current workflow screen or previously saved workflow screens is collected and sent to the server side component (via string representation of an XmlWorkflowMessage object) for processing. If a key does not have a named control in the current workflow screen or previously saved workflow screens, then the key will be ignored and not be included in the client value collection request string. The server side component should first check whether a key exists in the client request string before accessing the key’s values. Likewise, the data received on the server side can be manipulated and returned to the client side (via string representation of an XmlWorkflowMessage object) for processing. The RMI action is complete after the data sent from server side to client side has been processed.

The 'onerror' attribute specifies the error message to display in the event of an error during the RMI call. %REQUESTERROR% can be used to display the error message received from the server.

The 'timeout' attribute specifies the maximum number of seconds to wait for a server response before announcing a timeout error. %TIMEOUTERROR% resource can be overridden to display a custom timeout message.

See also on the server side, ISynchronousRequestProcessor.ProcessSyncRequest Method.

Action Positioning Rules for iPhone

Rule #1: If a default action is defined for the screen, it is positioned on the top right.

Rule #2: The first Cancel or Close action if any are defined, is positioned on the top left. The behavior varies slightly depending on whether the screen is the first one opened or a nested screen: a Close action takes precedence over a Cancel action on the first screen, while on the nested screen only Cancel actions are considered for the spot. This is because the top left button should work as a return-one-screen-back function according to the Apple guildelines and not an exit-all button.

Rule #3: If no default actions are specified for the screen, and the screen has Save or Submit actions defined, they are assigned to the top-right slot. Again, the behavior differs slightly depending on whether it is a first screen or a nested screen:

- On nested screens, Save actions take precedence over Submit actions.
- On the first screen, Submit takes precedence over Save.

Rule #4: If there is more than one action that is still not assigned to a slot, they are all put in the menu.

Rule #5: If after rules 1..3 there is only one action left, then there is no menu, and the action is put in place of the menu button.

Rule #6: If no action has been placed in the top left, a default Cancel button is placed there.

Xml Workflow Package Resources

Resources allow the developer to specify different variables and their values within the xml description of the client side component for use within all the different supported attributes. Using this concept, a developer can construct a single xml description for multiple languages to simplify the management of Workflow packages in larger global deployments.

```
<workflow>
  <screens>...<screens>
  <resources>
    <strings default="en-us">
      <lang key="en-us">
        <string key="INPUT" value="Input:" />
        <string key="OUTPUT" value=”Output:" />
        <string key="ERROR" value="Error" />
      </lang>
      <lang key="de-de”>
        <string key="INPUT" value="Eingabe:" />
        <string key="ERROR” value="Fehler" />
      </lang>
    </strings>
  </resources>
</workflow>
```

The above example shows how labels can be localized for U.S. English and German. To use a resource in a label text or a supported value, the resource needs to be specified with %KEYNAME% within the label text. E.g., to reference the above defined INPUT resource within a label text, the developer would use %INPUT%. Only numbers 0-9 and capital letters A-Z can be used for resource variable names/keys. There are six predefined resources. On Windows Mobile devices the Menu label (%MENU%)the Message Box title for validation errors (%ERROR%) and the Message Box title for submit actions with the attribute onsubmit (%INFO%) can be overwritten. On Symbian devices the check box values (%ON% and %OFF%) can be overwritten. In addition, iPhone devices define another resource (%MENUCANCEL%) which can also be overwritten. This is the label for the button to close the menu in Workflow forms.

Xml Workflow Form Controls

The following chapters describe the different controls available. The control behavior is generally controlled using attributes, whereas certain attributes are only available with certain controls or only make sense in conjunction with other attributes of a single control. For example, the min- and maxvalue attributes of the textbox control are only useful when the control also has the numeric attribute set to true).

The “key” attribute can contain only capital letters and numbers. No special characters are allowed.

textbox

```
<textbox key="tbx" value="" label="Textinput:" />
```

Attributes

key Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.

value The default value for this control (default: "")

- Allows resource reference, e.g. %VALUE%
- In the above example the value can also be specified as the innertext of the control tag.

label The label describing the control on the user interface (default: "")

- Allows resource reference, e.g. %VALUE%

labelpos The position of the label in regards to the associated control (possible values: left, right and top - default: left).

Note: This attribute is not supported on the Symbian platform.

required Specifies if user input for the control is required, a submission without a value is not possible (default: false).

maxlength The maximum length (number of characters) for the control. Default is 32767 characters. Assigning this attribute to '0' will remove the max length limitation.

readonly Specifies if the control is read only (default: false).

multiline Specifies the number of lines for the text control (default: 1).

Note: This attribute is not supported on the Symbian platform.

regex If the user enters text in the control, the input has to comply to provided regular expression (default: "").

Note: The “regular expression” matching rule offers the highest degree of flexibility. The regular expression syntax supported by Mobile Office is the *syntax based on that used by the programming language Perl. A reference for this syntax can be found at http://www.boost.org/doc/libs/1_36_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html.*

Note: This attribute is not supported on the Symbian platform.

numeric	Specifies if the displayed or entered text should be treated as a numeric type (can be integer or decimal), (default: false).
decimals	Specifies the number of decimals (the number of digits after the decimal point, valid range is 0 to 28),for a number, only valid in conjunction with the numeric attributes (default: 0).
minvalue	Specifies the minimal value for numeric input, only valid in conjunction with the numeric attributes (default: none).
maxvalue	Specifies the maximum value for numeric input, only valid in conjunction with the numeric attributes (default: none).
onerror	Specifies the error message displayed to the user in case the input validation criteria do not match. • Allows resource reference, e.g. %VALUE%
credential	If a Workflow package supports cached credentials, this attribute specifies that the textbox be treated as either the username or password. “username|password”. (Refer also to CredentialRequestException class.)
password	If this attribute is set to true, text entered in the textbox is replaced with asterisks. Default is false.

selectbox

```
<selectbox key="sel1" value="" label="Select:">
  <option key="" value="" />
  <option key="option1" value="displayvalue1" />
</selectbox>
```

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	Specifies a list of selectable items using a formatted string, e.g. key1$value 1\|key2$value 2|key3$value 3 will generate a select box with items: value 1, value 2, value 3. Duplicate Keys are discarded. The first item in the select box is selected by default. This value could be dynamically set by the server. • Allows resource reference, e.g. %VALUE%

label	The label describing the control on the user interface (default: "") • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
numeric	Specifies if the displayed text should be treated as a numeric type (default: false).
decimals	Specifies the number of decimals for a number, only valid in conjunction with the numeric attributes (default: 0).
onerror	Specifies the error message displayed to the user in case the input validation criteria's do not match. • Allows resource reference, e.g. %VALUE%

Innertags

```
<option key="oKey" value="val" />
```

Attributes

key	Unique identifier for the associated option within the scope of a single selectbox control. • Used when setting or getting a value from the selectbox control.
value	Specifies the value for the associated option. • Allows resource reference, e.g. %VALUE%

checkbox

```
<checkbox key="chk1" value="true" label="Checkbox:" />
```

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: ""). • Allows resource reference, e.g. %VALUE%
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%

labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies that the submission is possible only when the checkbox is checked. A submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
onerror	Specifies the error message displayed to the user in case the input validation criteria's do not match. • Allows resource reference, e.g. %VALUE%

datepicker

```
<datepicker key="dp1" value="today" label="Date:" />
```

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: "").
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
minvalue	Specifies the minimum value (default: none).
maxvalue	Specifies the maximum value (default: none).

Value expressions

One can specify absolute values or values relative to the current day using simple calculations in days, e.g.,

“2008-05-24”
"today"
"today+1"
"today-1"

All values are expected to be in UTC, values are displayed in local time and transferred in UTC between device and server.

datetimepicker

`<datetimepicker key="dtp1" value="now" label="Date:" />`

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: "").
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
minvalue	Specifies the minimum value (default: none).
maxvalue	Specifies the maximum value (default: none).
precision	Specifies the precision of the time portion displayed to the user and calculations (possible values: hours, minutes or seconds - default: seconds). *Note: iPhone Date and Time controls allow input for hour and minutes only. They do not allow the selection of seconds. The iPhone Workflow message has seconds always set to "00".*

Value expressions

One can specify absolute values or values relative to the current time or day using simple calculations in seconds. e.g.,

"2008-05-24" or "2008-05-24T14:30:15"
"today", "today+86400" or "today-86400" - today is interpreted as the current date, time 12:00 PM
"now", "now-86400" or "now+86400"

All values are expected to be in UTC, values are displayed in local time and transfered in UTC between device and server.

timepicker

```
<timepicker key="tp1" value="now" label="Time:" />
```

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: "").
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
minvalue	Specifies the minimum value (default: none).
maxvalue	Specifies the maximum value (default: none).
precision	Specifies the precision of the time used for display and calculations (possible values: hours, minutes or seconds - default: seconds).

Value expressions

Only absolute values are permitted, e.g. "14:30" or "14:30:15". All values are transferred as text in a non-localized form of "HH:mm:ss"

The valid values are 00:00:00 to 23.59:59.

Note: iPhone Date and Time controls allow input for hour and minutes only. They do not allow the selection of seconds. The iPhone Workflow message has seconds always set to "00".

slider

```
<slider key="slide1" value="5" label="Performance:" />
```

Note: The <slider> tag is not supported on SmartPhone platforms.

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.

value	The default value for this control (default: 0).
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).
minvalue	Specifies the minimum value (default: 0).
maxvalue	Specifies the maximum value (default: 10).
onerror	Specifies the error message displayed to the user in case the input validation criteria do not match. • Allows resource reference, e.g. %VALUE%

signature

```
<signature key="sig1" value="" label="Signature:" />
```

Note: The Signature control is only supported on the Windows Mobile Professional platform.

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: "").
label	The label describing the control on the user interface (default: ""). • Allows resource reference, e.g. %VALUE%
labelpos	The position of the label in regards to the associated control (possible values: left, right and top - default: left). *Note: This attribute is not supported on the Symbian platform.*
required	Specifies if user input for the control is required, a submission without a value is not possible (default: false).
readonly	Specifies if the control is read only (default: false).

onerror	Specifies the error message displayed to the user in case the input validation criteria's do not match. • Allows resource reference, e.g. %VALUE%

Value expressions

The value is a list of points that combined make up the signature that gets or was drawn to the control, e.g. -1x-1,5x7,5x8,-1x-1 whereas -1x-1 is interpreted as a interruption of a line drawn to the screen.

htmlview

```
<htmlview key="html1"><![CDATA[<html> html content </html>]]></htmlview>
```

Attributes

key	Unique identifier within the scope of a Workflow package, used when setting or retrieving a value, required.
value	The default value for this control (default: ""). • As shown in above example the value can also be specified as the innertext of the control tag. • Allows resource reference, e.g. %VALUE%

Note: The htmlview control always fills the whole screen of an Xml Workflow form; no other controls can be placed on a screen that hosts an htmlview control.

Value expressions

An alternative to specify the value of the htmlview is:

```
<htmlview key="html1" value="&lt;html&gt; html content &lt;/html&gt;" />
```

Workflow Package Packaging

A Workflow package is packaged as a compressed zip file. A developer can suggest an icon from a preselected list to associate with the packaged application; however, when an Administrator deploys the application, he or she has the ability to override the suggested icon with a different one. The selected icon appears beside the e-mail message on the mobile device instead of the default mail icon.

If the manifest.xml file specifies an unknown icon index for the Workflow package, the default icon used is icon number 48.

Available Icons

Icon Index	Name	Icon
30	Document	📄
31	User_Exclaim	👤
32	Chart Dot	📈
33	User_Male_2	👤
34	Internet	🌐
35	Clipboard	📋
36	User_Male_Chat	💬
37	Toolbox	🧰
38	Shopping_Cart	🛒

39	Notepad Edit
40	User_Male_Edit
41	Chart Bar
42	Management
43	Globe Chat
44	User_Male
45	Management Chat
46	Categories
47	Management Edit
48	New Workflow package **(this is the default AMP icon)**
49	Accounts
50	Airplane
51	Announcements
52	Barcode
53	Briefcase

54	Cargo Ship	🚢
55	Clock Alarm	⏰
56	Compass	🧭
57	Connected	🌐
58	Contact Card	🪪
59	Delivery Truck	🚚
60	Digital_Camera	📷
61	Document Edit	📝
62	Document Search	🔍
63	Fax Machine	📠
64	Find	🔭
65	First_Aid	⛑
66	Folder_32	📁
67	Health_Meter	📈
68	Home	🏠
69	Hospital	🏥

70	Insurance
71	Invoice
72	Manufacturing
73	Medical_Symbol
74	Newspaper
75	Pager
76	PDA
77	Phone
78	Prescription
79	Printer
80	Railcar
81	RSS_Newsfeed
82	Scanner
83	Shopping_Bag
84	Software_CD
85	SOS
86	Speedometer

87	Status Flag Black
88	Status Flag Blue
89	Status Flag Green
90	Status Flag Red
91	Status Flag Yellow
92	Support_Tube
93	Symbol Add
94	Symbol Delete
95	Symbol Security
96	Technology
97	Thumbs Down
98	Thumbs Up
99	Travel Suitcase
100	Under_Construction
101	Video_Document
102	Battery
103	Calculator

104	Calendar	📅
105	Cash Register	🧾
106	Configuration	⚙
107	Firewall	🧱
108	Ledger	📗
109	Planner	📕
110	Push Pin Note	📌
111	Safe	🔒
112	Search	🔍
113	Tools	🛠
114	Truck	🚚
115	UPS	🔋
116	Virus	🐞

The Manifest.xml File

The manifest.xml file describes how the contents of the zip package are organized. This file must sit at the root of the zip package.

The following is the skeleton outline of a manifest.xml file. For a complete description of each of the sections, please refer to the next section Manifest.xml File Reference.

```
<?xml version="1.0" encoding="utf-16"?>
<Manifest xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:noNamespaceSchemaLocation="AMPManifest.xsd">
   <ModuleName>…</ModuleName>
   <ModuleVersion>…</ModuleVersion>
   <ModuleDesc>…</ModuleDesc>
   <ModuleDisplayName>…</ModuleDisplayName>
   <ClientIconIndex>…</ClientIconIndex>
   <InvokeOnClient>…</InvokeOnClient>
   <MarkProcessedMessages>…</MarkProcessedMessages>
   <DeleteProcessedMessages>…</DeleteProcessedMessages>
   <CredentialsCache key="…">…</CredentialsCache>
   <RequiresActivation>…</RequiresActivation>
   <TransformPlugin>
     <File>…</File>
     <Class>…</Class>
   </TransformPlugin>
   <ResponsePlugin>
     <File>…</File>
     <Class>…</Class>
   </ResponsePlugin>
     <ClientWorkflows>
       <WindowsMobileProfessional>
        <XMLWorkflow>
          <!-- File>…</File>
             <Class>…</Class>
             <CredentialScreen>…</CredentialScreen  -->
          <File>…</File>
          <CredentialScreen>…</CredentialScreen>
          <ActivationScreen>…</ActivationScreen>
        </XMLWorkflow>
      </WindowsMobileProfessional>
      <WindowsMobileStandard>
         <XMLWorkflow>
            <File>…</File>
            <CredentialScreen>…</CredentialScreen>
            <ActivationScreen>…</ActivationScreen>
         </XMLWorkflow>
      </WindowsMobileStandard>
      <Symbian>
         <XMLWorkflow>
            <File>…</File>
            <CredentialScreen>…</CredentialScreen>
            <ActivationScreen>…</ActivationScreen>
         </XMLWorkflow>
     </Symbian>
     <iPhone>
        <XMLWorkflow>
          <File>…</File>
          <CredentialScreen>…</CredentialScreen>
          <ActivationScreen>…</ActivationScreen>
        </XMLWorkflow>
     </iPhone>
</ClientWorkflows>
   <ContextVariables>
      <ContextVariable>
         <Name>…</Name>
         <Value>…</Value>
         <Password>…</Password>
      </ContextVariable>
   </ContextVariables>
   <MatchRules>
      <SubjectRegExp>…</SubjectRegExp>
      <ToRegExp>…</ToRegExp>
      <FromRegExp>…</FromRegExp>
      <CCRegExp>…</CCRegExp>
      <BodyRegExp>…</BodyRegExp>
      <CustomFieldRegExp>…</CustomFieldRegExp>
  </MatchRules>
</Manifest>
```

Manifest.xml File Reference

ModuleName

`<ModuleName>SampleActivitiesModule</ModuleName>`

Defines the name of the Workflow package.

ModuleDesc

`<ModuleDesc>AMP Sample - Activities Collection</ModuleDesc>`

A short description of the Workflow package.

ModuleDisplayName

```
<ModuleDisplayName>Activities Sample</ModuleDisplayName>
```

The display name of a Workflow package. This string gets shown to the user in the Workflow List on the device for user invocable Workflow packages. When Administrators deploy a Workflow package, they have the ability to override the Display Name that is specified here with one of their own choosing.

ClientIconIndex

```
<ClientIconIndex>35</ClientIconIndex>
```

The index of the icon to associate with this Workflow package. See table of available icons for possible index values. This icon is shown beside the email message in the device's Inbox listing instead of the regular email icon. When Administrators deploy a Workflow package, they have the ability to override the icon that is specified here with one of their own choosing.

InvokeOnClient

```
<InvokeOnClient>1</InvokeOnClient>
```

States whether this Workflow package is suitable for use without an associated e-mail. 1 for yes, 0 for no. If 1 is specified, the Workflow package is shown in the Workflow List on the device and can be used without the context of an e-mail message.

MarkProcessedMessages

```
<MarkProcessedMessages>1</MarkProcessedMessages>
```

States whether a Workflow message is to show a visual indication in Inbox after it has been processed (1 = yes).

> *Note: When a Workflow message shows a visual indication that it has been processed, the visual indication will disappear if the device is reregistered, or if the device user does a Refresh All Data.*

DeleteProcessedMessages

```
<DeleteProcessedMessages>1</DeleteProcessedMessages>
```

States whether a Workflow message is to be deleted from the mobile device's Inbox after a the message has been processed (1 = yes).

> *Note: You cannot set both `DeleteProcessedMessages` and `MarkProcessedMessages` to yes (1). If you want to set `MarkProcessedMessages` to yes, then `DeleteProcessedMessages` must be set to no (0).*

CredentialsCache

```
<CredentialsCache key="activity_credentials>1</CredentialsCache>
```

Specifies whether a Workflow package requires credentials (1 = yes). Different packages can specify a credentials key, and packages with the same credentials key share the set of credentials.

RequiresActivation

```
<RequiresActivation>1</RequiresActivation>
```

Specifies whether a Workflow package requires activation (1 = yes). If enabled, the screen defined in the ActivationScreen tag is displayed the very first time the Workflow package is launched before the default screen is displayed.

If the Activation Screen contains credentials controls (and the Workflow package requires credentials), the values are updated to the Credentials Cache automatically as well without further prompting with the specified Credentials Screen.

TransformPlugin

```
<TransformPlugin>
   <File/>
   <Class/>
</TransformPlugin>
```

Describes the server module implemented as a .NET assembly that implements the IMailProcessor interface. This module is responsible for processing the intercepted e-mail message before it gets delivered to the device.

Innertags

```
<File>WorkflowServer.dll</File>
```

The path including filename of the assembly that implements the IMailProcessor interface. The path is relative to the zip package.

```
<Class>WorkflowServer.TransformerComponent</Class>
```

The .NET Type in the assembly that implements the IMailProcessor interface.

ResponsePlugin

```
<ResponsePlugin>
   <File/>
   <Class/>
</ResponsePlugin>
```

Describes the server module implemented as a .NET assembly that implements the IResponseProcessor interface**.** This module is responsible for processing the response from the device.

Innertags

<File>WorkflowServer.dll</File>

The path including filename of the assembly that implements the IResponseProcessor interface. The path is relative to the zip package.

```
<Class>WorkflowServer.ResponderComponent</Class>
```

The .NET Type in the assembly that implements the IResponseProcessor interface.

ClientWorkflows

```
<ClientWorkflows>
   <WindowsMobileProfessional>
    <XMLWorkflow>
      <!-- File>Client\WorkflowClient.dll</File>
         <Class>WorkflowClient.uicMain</Class>
         <CredentialScreen>WorkflowClient.uicLogin</CredentialScreen  -->
      <File>…</File>
      <CredentialScreen>…</CredentialScreen>
      <ActivationScreen>…</ActivationScreen>
   </XMLWorkflow>
  </WindowsMobileProfessional>
  <WindowsMobileStandard>
    <XMLWorkflow>
       <File>…</File>
       <CredentialScreen>…</CredentialScreen>
       <ActivationScreen>…</ActivationScreen>
    </XMLWorkflow>
  </WindowsMobileStandard>
  <Symbian>
    <XMLWorkflow>
       <File>…</File>
       <CredentialScreen>…</CredentialScreen>
       <ActivationScreen>…</ActivationScreen>
    </XMLWorkflow>
  </Symbian>
 <iPhone>
    <XMLWorkflow>
       <File>…</File>
       <CredentialScreen>…</CredentialScreen>
       <ActivationScreen>…</ActivationScreen>
    </XMLWorkflow>
 </iPhone>
</ClientWorkflows>
```

Describes the supported device platforms for this Workflow package and the corresponding client module to use for each platform.

Innertags

`<WindowsMobileProfessional>…</WindowsMobileProfessional>`

Denotes support for Windows Mobile Professional devices (WM6) and Pocket PC (WM5) devices.

`<WindowsMobileStandard>…</WindowsMobileStandard>`

Denotes support for Windows Mobile Standard devices (WM6) and Smartphone (WM5) devices.

`<Symbian>…</Symbian>`

Denotes support for Symbian devices.

`<iPhone>…</iPhone>`

Denotes support for iPhone devices.

CFWorkflow

```
<CFWorkflow>
   <File/>
   <Class/>
</CFWorkflow>
```

Describes the client module implemented as a .NET CF user control that inherits from the iAnywhere.OMAAT.UIComponent class. This module represents the device UI shown to the user for a Workflow package.

Note: The CFWorkflow tag is not supported on the Symbian and iPhone platforms.

Innertags

`<File>Client\WorkflowClient.dll</File>`

The path including filename of the .NETCF assembly. The path is relative to the zip package.

`<Class>WorkflowClient.uicMain</Class>`

`<CredentialScreen>WorkflowClient.uicLogin<CredentialScreen/>`

The screen that prompts for the credentials if they are not present on the device.

`<ActivationScreen>WorkflowClient.uicActivate</ActivationScreen>`

The screen that is shown to the user the first time the Workflow package is launched.

The .NET Type in the assembly that inherits from iAnywhere.OMAAT.UIComponent. This Type will be the first screen that is shown when the Workflow package is executed.

XMLWorkflow

```
<XMLWorkflow>
   <File/>
   <CredentialScreen/>

   <ActivationScreen/>
</XMLWorkflow>
```

Describes the client module implemented as a XML UI Description. This module represents the device UI shown to the user for a Workflow package.

Innertags

`<File>Client\ActivitiesWorkflow.xml</File>`

The path including filename of the XML UI Description file. The path is relative to the zip package.

`<CredentialScreen>credentials_screen_key<CredentialScreen/>`

The screen's key attribute value that prompts for the credentials if they are not present on the device.

`<ActivationScreen>activation_screen_key</ActivationScreen>`

The screen's key attribute value that is shown to the user the first time the Workflow package is launched.

ContextVariables

`<ContextVariables>...</ContextVariables>`

Describes the collection of context variables that will be made available to the methods in the IMailProcessor and IResponseProcessor interfaces.

When Administrators deploy a Workflow package, they have the ability to override the Display Name that is specified here with one of their own choosing.

ContextVariables

```
<ContextVariable>
   <Name/>
   <Value/>
   <Password/>
</ContextVariable>
```

Describes a context variable that will be made available to the methods in the IMailProcessor and IResponseProcessor interfaces. When Administrators deploy a Workflow package, they have the ability to override the value of the context variable that is specified here.

It is expected that sufficient documentation will be provided by developers of Workflow packages for Administrators to knowledgeably edit a context variable's value as necessary. Context variables are a good place to store configuration information which will likely change between development and production environments.

Innertags

`<Name>OutputFolder</Name>`

The name of the context variable. This is the key used to retrieve the value of the context variable in the methods defined in the IMailProcessor and IResponseProcessor interface.

> *Note: The value of <Name> tag supports single-byte characters only.*

`<Value>C:\ActivitiesSampleOutput</Value>`

The value of the context variable. When Administrators deploy a Workflow package, they have the ability to override the value of the context variable that is specified here.

> *Note: The value of <Value> tag supports double-byte characters.*

`<Password>false</Password>`

States whether this context variable is a password. If true, the value will be displayed as asterisks in the Mobile Office Admin console.

<MatchRules>

`<MatchRules>...</MatchRules>`

Describes the collection of match rules that will be used to determine if an e-mail message should be sent to a TransformPlugin server module for processing. When Administrators deploy a Workflow package, they have the ability to Add, Delete and /or override the Match Rules that are specified here.

<MatchRule>

`<MatchRule>...   </MatchRule>`

Describes a single match rule.

> *Note: The value of the `<MatchRule>` tag supports double-byte characters.*

Innertags

`<SubjectRegExp>…</SubjectRegExp>`

The value to test for against the subject line of an e-mail message.

`<ToRegExp>…</ToRegExp>`

The value to test for against the To line of an e-mail message

`<FromRegExp>…</FromRegExp>`

The value to test for against the From line of an e-mail message

`<CCRegExp>…</CCRegExp>`

The value to test for against the CC line of an e-mail message

`<BodyRegExp>…</BodyRegExp>`

The value to test for against the Body text of an e-mail message

`<CustomFieldRegExp CustomFieldNum="">…</CustomFieldRegExp>`

The value to test for against a custom Notes field of an e-mail message. This is valid only for Mobile Office Domino installations. A custom field number must be specified using the CustomFieldNum attribute.

> *Note: Only one custom field matching is supported.*

Activities Tutorial

A fully functional and completed tutorial which demonstrates all of the above concepts can be found in your installation at:

\Program Files\iAnywhere Mobile Office\Tools\Workflow Samples\Activities

A prebuilt and packaged version of the tutorial can be deployed using the ActivitiesPackage.zip file in the same location.

The sample Workflow package is a simple application that collects Activity information on a device and writes the collected information out to a text file on the server. Where the text file is created is configured via a context variable. The device application is implemented as a .NET Compact Framework Workflow package for Windows Mobile Professional devices, and as an XML description for Windows Mobile Standard, Symbian and iPhone devices.

Please refer to the Administration Guide for details on deploying and managing applications at Start > Programs > iAnywhere Mobile Office > Mobile Office Docs.