Email Injection Developer Guide

iAnywhere® Mobile Office 5.7

SYBASE® iAnywhere

Document ID: DC01169-01-0570-01
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

Welcome

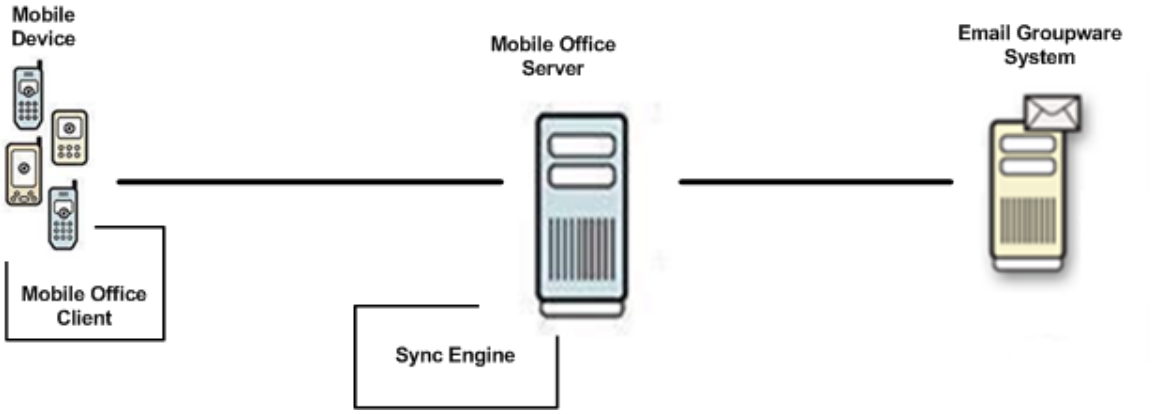
Welcome to iAnywhere® Mobile Office. iAnywhere Mobile Office is specifically designed for today's mobile business workforce. It combines fully integrated wireless email and PIM with on-device security and business process mobilization.

This document introduces you to the iAnywhere Mobile Office Email Injection API. After the study of this document you will understand the general concept of email injection and its development.

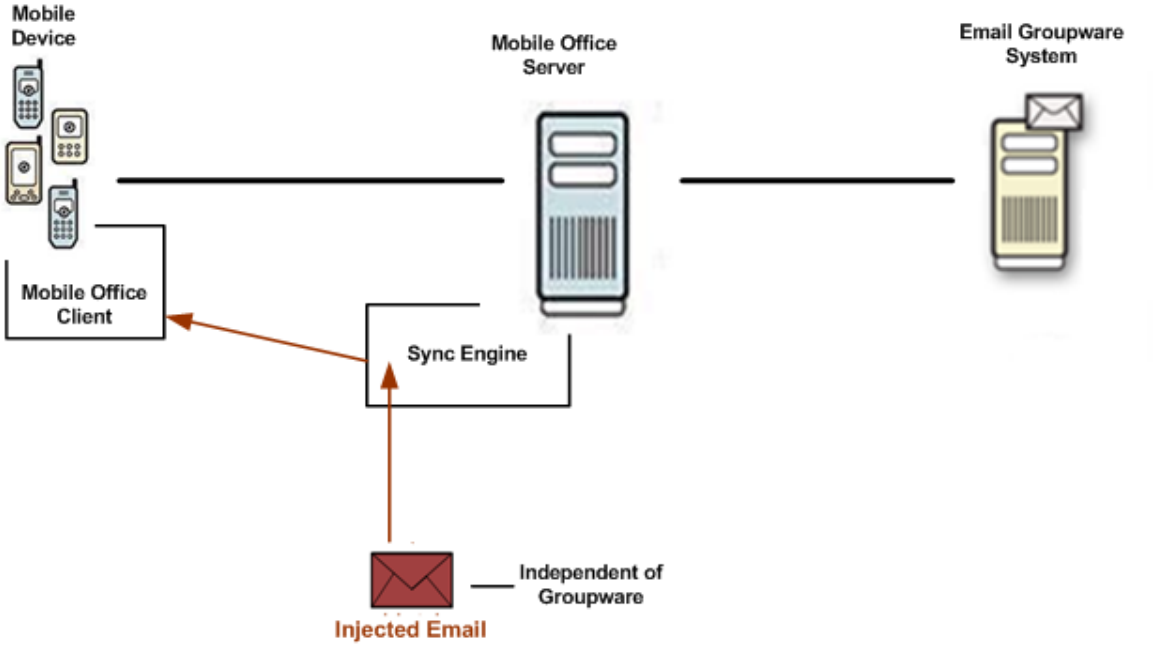
Solution Overview

The iAnywhere Mobile Office Injection API allows third party developers to "inject" emails into the Mobile Office system. An injected email behaves much like an email introduced into the system by the conventional groupware listeners. On the device, it is indistinguishable from a groupware email. However, it never appears on the groupware server. The images below compare the conventional email path with the injected email path.

Conventional email path:



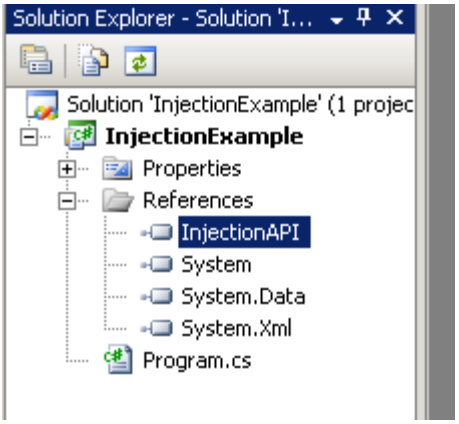
Injected email path:



The Injection API is implemented as a .NET assembly. To use the API, a programmer must build a .NET application that runs on the iAnywhere Mobile Office server and loads the Injection API assembly.

Project Setup

To use the Injection API, you must create a .NET project. A console application is used as an example here, but you could also use a form-based application. After creating your project, you must add a reference to *InjectionAPI.dll*, found in the C:\Program Files\iAnywhere Mobile Office\Bin folder.



Your program must be run on a Mobile Office server and have access to *InjectionAPI.dll*. *InjectionAPI.dll* should *not* be moved from its installation folder.

All objects in the API are in the iAnywhere.MobileOffice namespace, so it is convenient to include a "using iAnywhere.MobileOffice" statement at the top of any source file that calls the API.

Injecting an Email

Injecting an email requires that you call one method and use two objects, one indicating the content (see next section, MailContent class) and the other indicating the destination of the email (see MailDestination class).

MailContent class

The MailContent class encapsulates the content of an email message. The object has a constructor which takes the values of commonly used email fields as arguments. Each constructor argument corresponds to a read/write property of the class.

It has the following interface:

```
namespace iAnywhere.MobileOffice
{
   [Serializable]
   public class MailContent
   {
      public MailContent( string sSubject, string sTo, string sFrom,
                          string sCC, string sBCC, string sBody,
                          DateTime dtReceivedDate, bool bIsRead,
                          bool bIsPriority );

      public string BCC { get; set; }
      public string Body { get; set; }
      public string CC { get; set; }
      public string From { get; set; }
      public bool IsPriority { get; set; }
      public bool IsRead { get; set; }
      public DateTime ReceivedDate { get; set; }
      public string Subject { get; set; }

      public string GetCustomField( int iNum );
      public void SetCustomField( int iNum, string sVal );

   }
}
```

Available Properties

Property	Type	Description
Subject	string	Subject line of email
To	string	“To” line that will appear in the email on the device. If an empty string is passed to the API, the Mobile Office user name of the recipient will be substituted for the empty string when the email is delivered. Note that this string will not be interpreted by the system, so it need not be an actual email address.
From	string	“From” line that will appear in the email on the device. Note that this string will not be interpreted by the system, so it need not be an actual email address, although this may be advisable if you wish for the user to have the ability to reply to the email.
CC	string	“CC” line that will appear in the email on the device. Note that this string will not be interpreted by the system, so it need not be an actual email address, although this may be advisable if you wish for the user to have the ability to reply to the email. The system will *not* send the injected email to any addresses in the CC

		line.
BCC	string	"BCC" line that will be associated with the email. Note that this string will not be interpreted by the system, so it need not be an actual email address. The system will *not* send the injected email to any addresses in the BCC line. Since the BCC line is not visible on the device, the most likely use for this field is for server processing, such as if you are using it in matching rules for Mobile Office Business Process Workflow packages.
Body	string	Body of the email that will appear on the device. The API does not support formatted (rich text, HTML) email bodies or attachments.
ReceivedDate	DateTime	Received date that will be displayed for the email on the client. This field is time zone aware, so if you specify it using local time on the server, it will be appropriately transformed into the local time zone of the device when it is displayed.
IsRead	bool	This flag indicates whether the email will appear in a "read" or "unread" state when it first arrives on the device. Pass true for "read" or false for "unread."
IsPriority	bool	This flag indicates whether the email will appear with a high priority flag, which is usually represented with a red exclamation point ("!") icon, on the device. Pass true for high priority.

The SetCustomField method is used to set one of ten custom fields that can be attached to the email. The custom fields are indexed using the integers 1 through 10. The first argument is the index and the second is the string value to which that field should be set. The GetCustomField method can be used to retrieve the set value. Currently, the custom fields are only used during the match process for Business Process Workflow packages, so you need only set these fields if you are using match rules based on the fields.

Note: Only one custom field matching is supported.

MailDestination class

The MailDestination class is used to specify the recipient of an email. Its only public methods are its constructors. It has an overloaded constructor that allows the specification of the recipient in different ways.

The following constructors are available:

```
public MailDestination( int iDeviceId );
public MailDestination( string sDeviceName );
public MailDestination( string sDeviceName, string sUserName );
```

The iDeviceId argument is the internal Mobile Office numeric identifier associated with the destination device. This is the only constructor that can be used to send emails to devices that have been registered, but not yet activated (the device has not yet connected). This value may have been acquired by the third party application through the Web Services API or through a registration notification.

The sDeviceName argument is the external string identifier for the device that is received from the device during activation. This is effectively the unique hardware device identifier for the device and is the value that is displayed in the Admin application on the main screen in the device list.

The sUserName argument is the Mobile Office user name of the intended recipient. This value is used for validation only, to verify that the sDeviceName that is specified is currently associated with the specified user. This comparison is case-insensitive.

If the DeviceId, DeviceName, or UserName is not valid, an Exception will be thrown. This check is done at the time of construction of the object.

InjectionAPI class

This class contains two public static methods for sending and deleting emails.

Sending Emails

```
public static void InjectMail( MailDestination oDest, string sMessageId,
                               int iVersion, MailContent oMail )
```

This method allows the injection of a new email. The arguments are as follows:

Argument	Description
oDest	Destination of the email, specified through the constructor of the object
sMessageId	Arbitrary identifier for the email. This identifier is associated with the email in the Mobile office system as a server identifier, analogous to the identifier provided by a groupware server for a conventional email. This identifier must be unique across all emails the device receives. If you are using this API within an environment that is also delivering groupware emails, it must not conflict with the identifiers generated by the groupware server for emails sent to the same device. The MessageId may be a maximum of 32 characters in length and may use only single-byte (ASCII) characters. Note that a GUID satisfies all the above conditions on the MessageId. We recommend that callers use a GUID for this field.
iVersion	An integer indicating the record version for the email. Use “1” for a new email. This field is relevant if you are sending updates to an existing email (see below).
oMail	Content of the email.

This method may also be used to update an existing email. To send an update, use the same MessageId as in the original creation of the email. Send a Version number that is greater than any sent so far for this email (*e.g.*, send “1” for the original creation and “2” for a subsequent update). The destination should be the same device as the original email. The MailContent object should contain the same values as the original email, with the IsRead property setting changed. If properties other than IsRead are modified in the MailContent during an update, *they will only be seen if, at the time of the update, the email has not yet been delivered to the device*. Once the email is delivered to the device, only the IsRead property will be transmitted and changes to other properties will be ignored.

> *Note: Email filter properties such as Past Days and Preview Size in the Inbox setting of the template do not apply to injected emails. Injected emails are independent of backend groupware systems.*

Deleting Emails

```
public static void DeleteMail( MailDestination oDest, string sMessageId )
```

This method is used for deleting an email sent with InjectMail. Simply pass the same destination and MessageId as was sent in the call to InjectMail. This will remove the email from the device as if the user had deleted it. If the email has not yet been delivered to the device, it will be removed from the queue and never delivered.

Error Handling

All methods in the API indicate error conditions by throwing an instance of the standard Exception class with a descriptive error message. The caller should be prepared to handle such exceptions.

Sample API Code

The following code demonstrates the use of the API. It simply sends an email to a device, waits 15 seconds, updates the read flag, waits another 15 seconds, and then deletes the email.

```
using System;
using iAnywhere.MobileOffice;

namespace InjectionExample
{
   class InjectionExample
   {
      static void Main( string[] args )
      {
         try
         {
            MailContent oContent = new MailContent(
               "My subject",  // subject
               "MyTo",        // To
               "MyFrom",      // From
               "MyCC",        // CC
               "MyBCC",       // BCC
               "My body",     // Body
               DateTime.Now,  // Delivered Date
               false,         // Unread
               true           // High priority
               );


            // use hardware device name to specify destination
            MailDestination oDest = new MailDestination(
               "150006F0063006B006500740050004300000044455649434545 4D00"
               );

            // use GUID for message ID
            string sMesssageId = Guid.NewGuid().ToString( "N" );

            // send the email (use version = 1 )
            InjectionAPI.InjectMail( oDest, sMesssageId, 1, oContent );
```

```
            // sleep a while so we can see it delivered to the device
            System.Threading.Thread.Sleep( 15000 );

            // update the read/unread flag on the message
            // (use version = 2)
            oContent.IsRead = true;
            InjectionAPI.InjectMail( oDest, sMesssageId, 2, oContent );

            // sleep a while so we can see the change
            // delivered to the device
            System.Threading.Thread.Sleep( 15000 );

            // delete the email from the device
            InjectionAPI.DeleteMail( oDest, sMesssageId );
         }
         catch ( Exception x )
         {
            Console.Error.WriteLine( x.Message );
         }
      }
   }
}
```