



Using Email Adapters in Productions

Version 2025.1
2025-06-03

Using Email Adapters in Productions

PDF generated on 2025-06-03

InterSystems IRIS® Version 2025.1

Copyright © 2025 InterSystems Corporation

All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™, HealthShare® Health Connect Cloud™, InterSystems® Data Fabric Studio™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Congress Street, Boston, MA 02114, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: support@InterSystems.com

Table of Contents

1 Using the Email Inbound Adapter	1
1.1 Overall Behavior	1
1.2 Creating a Business Service to Use the Email Inbound Adapter	3
1.3 Implementing the OnProcessInput() Method	3
1.4 Using Email Messages	4
1.4.1 Message Contents	4
1.4.2 Message Headers	5
1.4.3 Other Message Information	5
1.5 Adding and Configuring the Business Service	6
1.5.1 Specifying How to Log in to a POP3 Server	6
1.5.2 Specifying the Messages to Retrieve	6
2 Using the Email Outbound Adapter	9
2.1 Overall Behavior	9
2.2 Creating a Business Operation to Use the Adapter	9
2.3 Creating Message Handler Methods	11
2.3.1 Available Methods	11
2.3.2 Example	12
2.4 Creating Email Messages	13
2.4.1 Creating an Email Message	13
2.4.2 Specifying the Content and Headers of a Message Part	14
2.4.3 Specifying the Message Headers	14
2.4.4 Adding Attachments	15
2.4.5 Example	16
2.5 Example	17
2.6 Adding and Configuring the Business Operation	18
2.6.1 Specifying How to Log Into the SMTP Server	18
2.6.2 Specifying Additional Email Addresses	19
Email Adapter Settings	21
Settings for the Email Inbound Adapter	22
Settings for the Email Outbound Adapter	24

1

Using the Email Inbound Adapter

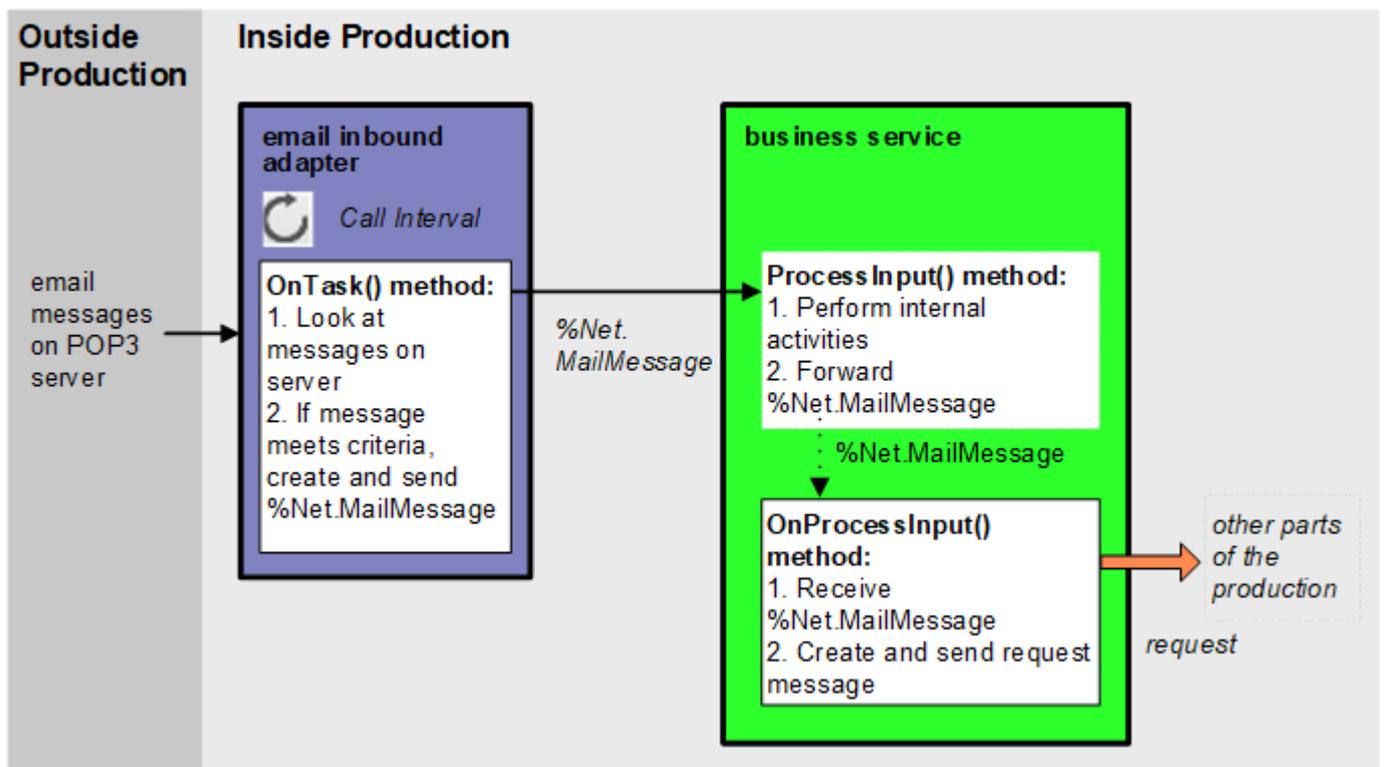
This topic describes the default behavior of the email inbound adapter (`EnsLib.Email.InboundAdapter`) so that the production can send and receive email and describes how to use this adapter in your productions. You should be familiar with the requirements and limitations of the SMTP or POP3 server with which you are working.

1.1 Overall Behavior

First, it is useful to understand the details that you specify for the adapter. The `EnsLib.Email.InboundAdapter` class provides runtime settings that you use to specify items like the following:

- The POP3 server to use and login details for the mailbox from which to read messages
- Matching criteria that indicate the messages of interest
- A polling interval, which controls how frequently the adapter checks for new input

In general, the inbound email adapter (`EnsLib.Email.InboundAdapter`) periodically checks the mailbox, finds matches, sends the messages (as instances of `%Net.MailMessage`) to the associated business service, and deletes the messages from the email server. The business service, which you create and configure, uses these messages and communicates with the rest of the production. The following figure shows the overall flow:



More specifically:

1. The adapter regularly executes its **OnTask()** method, which connects to the POP3 server and logs on, using a specific username and password. The polling interval is determined by the CallInterval setting.
2. The adapter looks at all the messages in this mailbox and compares them against the match criteria.
3. When the adapter finds a message that meets the criteria, it does the following:
 - a. The adapter creates an instance of the %Net.MailMessage class and puts the email data into it.
 - b. The adapter calls the internal ProcessInput() method of the associated business service class, passing the %Net.MailMessage instance as input.
 - c. The adapter deletes the mail message from the server.
4. The internal ProcessInput() method of the business service class executes. This method performs basic production tasks such as maintaining internal information as needed by all business services. You do not customize or override this method, which your business service class inherits.
5. The ProcessInput() method then calls your custom OnProcessInput() method, passing the %Net.MailMessage instance as input. The requirements for this method are described later in [Implementing the OnProcessInput\(\) Method](#).

The response message follows the same path, in reverse.

1.2 Creating a Business Service to Use the Email Inbound Adapter

To use this adapter in your production, create a new business service class as described here. Later, [add it to your production and configure it](#). You must also create appropriate message classes, if none yet exist. See [Defining Messages](#).

The following list describes the basic requirements of the business service class:

- Your business service class should extend `Ens.BusinessService`.
- In your class, the `ADAPTER` parameter should equal `EnsLib.EMail.InboundAdapter`.
- Your class should implement the `OnProcessInput()` method, as described in [Implementing the OnProcessInput\(\) Method](#).
- For other options and general information, see [Defining a Business Service Class](#).

The following example shows the general structure that you need:

Class Definition

```
Class EEMA.EmailService Extends Ens.BusinessService
{
Parameter ADAPTER = "EnsLib.EMail.InboundAdapter";

Method OnProcessInput(pInput As %Net.MailMessage,
                    pOutput As %RegisteredObject) As %Status
{
    set tsc=$$$OK
    //your code here
    Quit tsc
}
}
```

1.3 Implementing the OnProcessInput() Method

Within your custom business service class, your `OnProcessInput()` method should have the following signature:

```
Method OnProcessInput(pInput As %Net.MailMessage,
                    pOutput As %RegisteredObject) As %Status
```

Here *pInput* is the email message object that the adapter will send to this business service; this is an instance of `%Net.MailMessage`. Also, *pOutput* is the generic output argument required in the method signature.

The `OnProcessInput()` method should do some or all of the following:

1. Examine the email message and decide how to use it.

Email messages can have a variety of different structures, which would need different handling. You may want to start by making sure that the message has the structure you expect. That is, you would check whether it is a multipart message and whether the parts are binary. For more information, see [Using Email Messages](#).
2. Once you are sure of the message structure, use the other properties of `%Net.MailMessage` to access the message body or any headers. See [Using Email Messages](#).
3. Create an instance of the request message, which will be the message that your business service sends.

For information on creating message classes, see [Defining Messages](#).
4. For the request message, set its properties as appropriate, using values in the email message.

5. Call a suitable method of the business service to send the request to some destination within the production. Specifically, call **SendRequestSync()**, **SendRequestAsync()**, or (less common) **SendDeferredResponse()**. For details, see [Sending Request Messages](#).

Each of these methods returns a status (specifically, an instance of %Status).

6. Make sure that you set the output argument (pOutput). Typically you set this equal to the response message that you have received. This step is required.
7. Return an appropriate status. This step is required.

The following shows a simple example:

Class Member

```
Method OnProcessInput(pInput As %Net.MailMessage,
pOutput As %RegisteredObject) As %Status
{
    //Check if mail message has multiple parts
    Set multi=pInput.IsMultiPart
    If multi
        {$$$TRACE("This message has multiple parts; not expected")}
        Quit $$$ERROR($$$GeneralError,"Message has multiple parts")
    }

    //Check if mail message is binary
    Set bin=pInput.IsBinary
    If bin
        {$$$TRACE("This message is binary; not expected")}
        Quit $$$ERROR($$$GeneralError,"Message is binary")
    }

    //Check if mail message is HTML
    Set html=pInput.IsHTML
    If html
        {$$$TRACE("This message is HTML not expected")}
        Quit $$$ERROR($$$GeneralError,"Message is HTML")
    }

    //now safe to get text of message
    Set pReq=##class(EEMA.EmailContents).%New()
    Set pReq.MessageText=pInput.TextData

    Set tSc=..SendRequestSync("EEMA.EmailProcessor",pReq,.pResp)
    Set pOutput=pResp

    Quit tSc
}
```

1.4 Using Email Messages

As noted earlier, after you retrieve an email message (%Net.MailMessage), you generally start by determining what kind of message it is and how to read it; that is, whether it is a multipart message and whether the parts are binary.

In this step, you can use the ContentType property, which is equivalent to the Content-Type property. Or you can use the IsBinary, IsHTML, and IsMultiPart properties, which indirectly provide the same information as ContentType.

If the message is a multipart message, each part is an instance of %Net.MailMessagePart.

A message has *message headers*; each part of the message can also have message headers. You can access the headers via various properties of %Net.MailMessage and %Net.MailMessagePart.

1.4.1 Message Contents

Once you know what the general message structure is, use the following techniques to retrieve the contents:

- For a multipart message, use the `Parts` property, which is an array of the parts. Use the **Count()** method to get the number of parts. Use the **GetAt()** method to retrieve a given part; the key for each part is an integer, starting with 1 for the first part.
- For a binary message (or message part), use the `BinaryData` property.
- For a text message (or message part), use the `TextData` property.
 - If `IsHTML` is 0, the `TextData` property is an ordinary text string.
 - If `IsHTML` is 1, the `TextData` property is an HTML text string.

Note that the email client that sends a message determines any wrapping in the message. The mail server has no control over this; nor does InterSystems IRIS.

1.4.2 Message Headers

The message itself and each part of the message has a set of headers.

The `%Net.MailMessage` class provides properties that give you the most commonly used headers.

- **To**—The list of email addresses to which this message was sent. This property is a standard InterSystems IRIS list; to work with it, you use the standard list methods: **Insert()**, **GetAt()**, **RemoveAt()**, **Count()**, and **Clear()**.
- **From**—The email address from which this message was sent.
- **Date**—The date of this message.
- **Subject**—A string containing the subject for this message.
- **Sender**—The actual sender of the message.
- **Cc**—The list of carbon copy addresses to which this message was sent.
- **Bcc**—The list of blind carbon copy addresses to which this message was sent. If the receiver of the message was on the blind carbon copy list, this property contains the address of the receiver; otherwise, it is null.

Both `%Net.MailMessage` and `%Net.MailMessagePart` provide the following properties, which give you other headers:

- **ContentType**—The `Content-Type` header of the message or message part.
- **ContentTransferEncoding**—The `Content-Transfer-Encoding` header of the message or message part.
- **Headers**—A multidimensional array that gives access to any additional headers.

Also, you can use the **GetAttribute()** method. Given a header name and an attribute, this method returns the value of that attribute.

1.4.3 Other Message Information

The following properties and methods provide additional information about the message:

- The `MessageSize` property indicates the total length of the message, apart from any attached email messages.
- The **GetLocalDateTime()** method returns the date and time when the message was retrieved, converted to local time in **\$HOROLOG** format.
- The **GetUTCDateTime()** method returns the date and time when the message was retrieved, converted to UTC in **\$HOROLOG** format.
- The **GetUTCSeconds()** method returns the date and time when the message was retrieved, in seconds since 12/31/1840.

- The **HToSeconds()** class method converts a date/time in **\$HOROLOG** format to seconds since 12/31/1840.
- The **SecondsToH()** class method converts seconds since 12/31/1840 to a date/time in **\$HOROLOG** format.

1.5 Adding and Configuring the Business Service

To add your business service to a production, use the Management Portal to do the following:

1. Add an instance of your custom business service class to the production.
2. Enable the business service.
3. Configure the adapter to access a POP3 mail server and download messages. Specifically:
 - [Specify a POP3 mail server and login details needed to access a mailbox](#)
 - [Specify criteria to determine which messages to download](#)
 - Use [Call Interval](#) to specify how often the adapter looks for email
4. Run the production.

1.5.1 Specifying How to Log in to a POP3 Server

Specify values for the following settings to indicate the POP3 server to log onto, as well as the security information to access a mailbox:

- [POP3 Server](#)
- [POP3 Port](#)
- [Credentials](#)
- [SSL Configuration](#)

For example, you could use values like the following:

Setting	Value
POP3 Server	pop.hotpop.com
POP3 Port	110
Credentials	hotpop

In this example, hotpop is the ID of the production credentials that consist of the username `isctest@hotpop.com` and the corresponding password.

1.5.2 Specifying the Messages to Retrieve

Specify values for the following settings to control which messages to retrieve. Only messages that match *all* the given criteria are used. The matching is case-sensitive.

- [Match From](#)
- [Match To](#)

- [Match Subject](#)

If you change these criteria, the adapter will examine and possibly process messages that did not match the criteria before.

2

Using the Email Outbound Adapter

This topic describes the behavior of the email outbound adapter (`EnsLib.Email.OutboundAdapter`) and describes how to use this adapter in your productions.

2.1 Overall Behavior

Within a production, an outbound adapter is associated with a business operation that you create and configure. The business operation receives a message from within the production, looks up the message type, and executes the appropriate method. This method usually executes methods of the associated adapter.

The email outbound adapter (`EnsLib.Email.OutboundAdapter`) provides settings that you use to specify the following:

- The SMTP server to connect to and login details needed for that server, if any.
- The default address to send email from (the `From:` header).
- Recipients to add to any messages sent by the adapter, in addition to any hardcoded recipients.

It provides methods to do the following three actions:

- Add a recipient to the `To:` list.
- Add a recipient to the `Cc:` list.
- Send a message.

2.2 Creating a Business Operation to Use the Adapter

To create a business operation to use the `EnsLib.Email.OutBoundAdapter`, you create a new business operation class. Later, [add it to your production and configure it](#).

You must also create appropriate message classes, if none yet exist. See [Defining Messages](#).

The following list describes the basic requirements of the business operation class:

- Your business operation class should extend `Ens.BusinessOperation`.
- In your class, the `ADAPTER` parameter should equal `EnsLib.Email.OutboundAdapter`.

- In your class, the *INVOCATION* parameter should specify the invocation style you want to use, which must be one of the following.
 - **Queue** means the message is created within one background job and placed on a queue, at which time the original job is released. Later, when the message is processed, a different background job will be allocated for the task. This is the most common setting.
 - **InProc** means the message will be formulated, sent, and delivered in the same job in which it was created. The job will not be released to the sender's pool until the message is delivered to the target. This is only suitable for special cases.
- Your class should define a *message map* that includes at least one entry. A message map is an XData block entry that has the following structure:

```
XData MessageMap
{
  <MapItems>
  <MapItem MessageType="messageclass">
    <Method>methodname</Method>
  </MapItem>
  ...
</MapItems>
}
```

- Your class should define all the methods named in the message map. These methods are known as *message handlers*. Each message handler should have the following signature:

```
Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
```

Here *Sample* is the name of the method, *RequestClass* is the name of a request message class, and *ResponseClass* is the name of a response message class. In general, these methods will refer to properties and methods of the Adapter property of your business operation.

- For other options and general information, see [Defining a Business Operation Class](#).

The following example shows the general structure that you need:

Class Definition

```
Class EMA.NewOperation1 Extends Ens.BusinessOperation
{
  Parameter ADAPTER = "EnsLib.Email.OutboundAdapter";

  Parameter INVOCATION = "Queue";

  Method SampleCall(pRequest As Ens.Request,
                   Output pResponse As Ens.Response) As %Status
  {
    Quit $$$ERROR($$$NotImplemented)
  }

  XData MessageMap
  {
  <MapItems>
  <MapItem MessageType="Ens.Request">
    <Method>SampleCall</Method>
  </MapItem>
  </MapItems>
  }
}
```

2.3 Creating Message Handler Methods

When you create a business operation class for use with `EnsLib.Email.OutboundAdapter`, typically your biggest task is writing message handlers for use with this adapter, that is, methods that receive production messages and then send email messages via an SMTP server.

Each message handler method should have the following signature:

```
Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
```

Here *Sample* is the name of the method, *RequestClass* is the name of a request message class, and *ResponseClass* is the name of a response message class.

In general, the method should do the following:

1. Examine the inbound request message.
2. Create an email message to send; for information, see [Creating Email Messages](#).
3. Optionally call the **AddRecipients()**, **AddCcRecipients()**, and **AddBccRecipients()** methods of the Adapter property of your business operation. These methods add email addresses to the `To:`, `Cc:`, and `Bcc:` headers when you send the email message. These methods are discussed after these steps.
4. Call the **SendMail()** method of the Adapter property of your business operation:

```
Set tSc=..Adapter.SendMail(email,.pf)
```

This method is discussed after these steps.

5. Examine the response.
6. Use information in the response to create a response message (an instance of `Ens.Response` or a subclass), which the method returns as output.

For information on defining message classes, see [Defining Messages](#).

7. Make sure that you set the output argument (`pOutput`). Typically you set this equal to the response message. This step is required.
8. Return an appropriate status. This step is required.

2.3.1 Available Methods

The adapter provides the following methods:

SendMail

```
Method SendMail(pMailMessage As %Net.MailMessage,
                Output pFailedRecipients As %ListOfDataTypes) As %Status
```

Given an email message, this method sends the message by means of the configured SMTP server. It returns, as output, as list of failed recipients, if the SMTP server returns this information.

AddRecipients

```
Method AddRecipients(pMailMessage As %Net.MailMessage,
                    pRecipients As %String)
```

Given an email message, this method adds the listed email addresses to the `To:` header of the message.

AddCcRecipients

```
Method AddCcRecipients(pMailMessage As %Net.MailMessage,
                      pRecipients As %String)
```

Given an email message, this method adds the listed email addresses to the Cc : header of the message.

AddBccRecipients

```
Method AddBccRecipients(pMailMessage As %Net.MailMessage,
                       pRecipients As %String)
```

Given an email message, this method adds the listed email addresses to the Bcc : header of the message. When sending an email there must be at least one address in the To : or Cc : header.

ContinueAfterBadSendSet

```
Method ContinueAfterBadSendSet(%val As %Integer) as %Status
```

If %val is true, the adapter continues to send the message if one or more of the recipients have an invalid address. The default is true.

2.3.2 Example

A method might look like the following:

Class Member

```
Method SendMultipartEmailMessage(pRequest As EEMA.MultipartEmailMsg,
Output pResponse As Ens.Response) As %Status
{
  Set part1=##class(%Net.MailMessage).%New()
  Do part1.TextData.Write(pRequest.Message1)
  Set part2=##class(%Net.MailMessage).%New()
  Do part2.TextData.Write(pRequest.Message2)
  Set part3=##class(%Net.MailMessage).%New()
  Do part3.TextData.Write(pRequest.Message3)

  Set email=##class(%Net.MailMessage).%New()
  Set email.Subject=pRequest.Subject
  Set email.IsMultiPart=1
  Do email.Parts.SetAt(part1,1)
  Do email.Parts.SetAt(part2,2)
  Do email.Parts.SetAt(part3,3)

  Set tSc=..Adapter.SendMail(email,.pf)
  Set pResponse=##class(EEMA.EmailFailedRecipients).%New()
  Set pResponse.FailedRecipients=pf

  if pf.Count()>0
  {
    set count=pf.Count()
    for i=1:1:count
    {
      $$$TRACE("Failed recipient:"_pf.GetAt(i))
    }
  }

  Quit tSc
}
```

For information on creating email messages, see the next section.

2.4 Creating Email Messages

A message can have one or more parts; each part contains content and can include headers. The message itself also has headers. The following sections discuss the topics:

- [How to create an email message in general](#)
- [How to specify the contents and headers of a given message part](#)
- [How to specify headers of the message and its parts](#)
- [How to add attachments](#)
- [A simple example](#)

Note: You should be aware of the requirements of the SMTP server that you are using. For example, some SMTP servers require that you include a `Subject:` header. Similarly, some SMTP servers do not permit arbitrary `From:` headers.

2.4.1 Creating an Email Message

To create an email message, do the following:

1. Create an instance of `%Net.MailMessage`.

Tip: You can specify a character set as the argument to `%New()`; if you do so, that sets the `Charset` property for the message.

For information on character sets and translation tables, see [Translation Tables](#).

2. Set the `To`, `From`, and `Subject` properties of your instance. These properties are required.
 - `To`—The list of email addresses to which this message will be sent. This property is a standard InterSystems IRIS list; to work with it, you use the standard list methods: **`Insert()`**, **`GetAt()`**, **`RemoveAt()`**, **`Count()`**, and **`Clear()`**.
 - `From`—The email address this message is sent from.
 - `Subject`—The subject of the message.
3. Optionally set `Date`, `Cc`, `Bcc`, and other properties. For details, see [Basic Headers](#).
4. If you are creating a single-part message, specify the contents and headers as described in [Specifying the Content and Headers of a Message Part](#).
5. Or, if you are creating a multipart message, do the following:
 - a. Set the `IsMultiPart` property to 1.
 - b. Set the `MultiPartType` property to one of the following: "related", "alternative", or "mixed".
 - c. For each part that the message should contain, create an instance of `%Net.MailMessagePart`. Then specify the contents and headers as described in [Specifying the Content and Headers of a Message Part](#).
 - d. For the parent email message, set the `Parts` property, which is an array. Insert each child message part into this array.

2.4.2 Specifying the Content and Headers of a Message Part

1. If the message is not plain text, set the following properties to indicate the kind of message you are creating:
 - If this is an HTML message, set the `IsHTML` property to 1.
 - If this is a binary message, set the `IsBinary` property to 1.

2. To specify the desired character set of the message and its headers, set the `Charset` property as needed.

Important: It is important to specify the character set *before* you add the contents of the message.

For information on character sets and translation tables, see Translation Tables.

3. Add the contents of the message:

- For plain text or HTML, use the `TextData` property, which is an instance of `%FileCharacterStream`. You do not need to specify the `TranslateTable` property of this stream; that occurred automatically when you specified the character set of the mail message.

The system automatically translates the character encoding to the encoding that you specified in the previous step.

- For binary data, use the `BinaryData` property, which is an instance of `%FileBinaryStream`.

Note that `%FileCharacterStream` and `%FileBinaryStream` are deprecated but are still supported for use in this way. It is not recommended to substitute different stream classes for this use case.

Tip: When you specify the `Filename` property of the stream, be sure to use a directory to which the users will have write access.

To work with these properties, use the standard stream methods: **Write()**, **WriteLine()**, **Read()**, **ReadLine()**, **Rewind()**, **MoveToEnd()**, and **Clear()**. You can also use the `Size` property of the stream, which gives you the size of the message contents.

2.4.3 Specifying the Message Headers

As noted previously, both the message itself and each part of a message has a set of headers.

The `%Net.MailMessage` and `%Net.MailMessagePart` classes provide properties that give you easy access to the most commonly used headers, but you can add any header you need. This section provides information on all the headers as well as how to create custom headers.

Note: The headers of a given message part are in the character set specified by the `Charset` property of that part.

2.4.3.1 Basic Headers

Set the following properties (only in `%Net.MailMessage`) to specify the most commonly used headers of the message itself:

- **To**—(Required) The list of email addresses to which this message will be sent. This property is a standard InterSystems IRIS list; to work with it, you use the standard list methods: **Insert()**, **GetAt()**, **RemoveAt()**, **Count()**, and **Clear()**.
- **From**—(Required) The email address this message is sent from.
- **Date**—The date of this message.
- **Subject**—A string containing the subject for this message.
- **Sender**—The actual sender of the message.

- **Cc**—The list of carbon copy addresses to which this message will be sent.
- **Bcc**—The list of blind carbon copy addresses to which this message will be sent.

Note: You should be aware of the requirements of the SMTP server that you are using. For example, some SMTP servers require that you include a `Subject:` header. Similarly, some SMTP servers do not permit arbitrary `From:` headers.

2.4.3.2 Content-Type Header

When you send the message, the `Content-Type` header for the message and for each message part is automatically set as follows:

- If the message is plain text (`IsHTML` equals 0 and `IsBinary` equals 0), the `Content-Type` header is set to `"text/plain"`.
- If the message is HTML (`IsHTML` equals 1 and `IsBinary` equals 0), the `Content-Type` header is set to `"text/html"`.
- If the message is binary (`IsBinary` equals 1), the `Content-Type` header is set to `"application/octet-stream"`.
- If the message is multipart, the `Content-Type` header is set as appropriate for the value of the `MultiPartType` property.

Both `%Net.MailMessage` and `%Net.MailMessagePart` provide the `ContentType` property, which gives you access to the `Content-Type` header.

2.4.3.3 Content-Transfer-Encoding Header

Both `%Net.MailMessage` and `%Net.MailMessagePart` provide the `ContentTransferEncoding` property, which provides an easy way to specify the `Content-Transfer-Encoding` header of the message or the message part.

This property can be one of the following: `"base64"` `"quoted-printable"` `"7bit"` `"8bit"`

The default is as follows:

- For a binary message or message part: `"base64"`
- For a text message or message part: `"quoted-printable"`

2.4.3.4 Custom Headers

With both `%Net.MailMessage` and `%Net.MailMessagePart`, you can set or get custom headers by accessing the `Headers` property, which is an array with the following structure:

Array Key	Array Value
Name of the header, such as <code>"Priority"</code>	Value of the header

You use this property to contain additional headers such as priority. For example:

```
Do msg.Headers.SetAt("Urgent", "Priority")
```

2.4.4 Adding Attachments

You can add attachments to an email message or message part. To do so, use the following methods of `%Net.MailMessagePart` or `%Net.MailMessage`.

Each of these methods adds the attachment to the `Parts` array of the original message (or message part), and automatically sets the `IsMultiPart` property to 1.

AttachFile

```
method AttachFile(Dir As %String,
                 File As %String,
                 isBinary As %Boolean = 1,
                 charset As %String = "",
                 ByRef count As %Integer) as %Status
```

Attaches the given file to the email message. By default the file is sent as a binary attachment, but you can specify instead that it is text. You can also specify the character set that the file uses if it is text. (For information on character sets and translation tables, see [Translation Tables](#).)

Specifically, this method creates an instance of `%Net.MailMessagePart` and places the contents of the file in the `BinaryData` or `TextData` property as appropriate, and sets the `Charset` property and `TextData.TranslateTable` properties if needed. The method returns, by reference, an integer that indicates the position of this new message part within the `Parts` array.

This method also sets the `Dir` and `FileName` properties of the message or message part.

AttachNewMessage

```
method AttachNewMessage() as %Net.MailMessagePart
```

Creates a new instance of `%Net.MailMessage`, adds it to the message, and returns the newly modified parent message or message part.

AttachEmail

```
method AttachEmail(mailmsg As %Net.MailMessage)
```

Given an email message (an instance of `%Net.MailMessage`), this method adds it to the message.

Note: This method sets `ContentType` to `"message/rfc822"`. In this case, you cannot add any other attachments.

For examples, see the class reference for the `%Net.MailMessagePart` class.

2.4.5 Example

For example:

Class Member

```
ClassMethod CreateTextMessage() As %Net.MailMessage
{
  Set msg = ##class(%Net.MailMessage).%New()
  set msg.From = "test@test.com"
  Do msg.To.Insert("xxx@xxx.com")
  Do msg.Cc.Insert("yyy@yyy.com")
  Do msg.Bcc.Insert("zzz@zzz.com")
  Set msg.Subject="subject line here"
  Set msg.IsBinary=0
  Set msg.IsHTML=0
  Do msg.TextData.Write("This is the message.")
  Quit msg
}
```

2.5 Example

This section presents an example that allows you to send simple single-part or multipart email messages that contain string content. In a production system, you would probably use streams instead of strings. Because this example uses strings, however, you can easily test it by means of the production testing page.

First, the request message class for the single-part message is as follows:

Class Definition

```
Class EEMA.SimpleMsg Extends Ens.Request
{
Property Subject As %String(MAXLEN = 100);
Property Message As %String(MAXLEN = 5000);
}
```

The request message class for the multipart message is next. This class can contain a three-part message.

Class Definition

```
Class EEMA.MultipartMsg Extends Ens.Request
{
Property Subject As %String(MAXLEN = 100);
Property Part1 As %String(MAXLEN = 5000);
Property Part2 As %String(MAXLEN = 5000);
Property Part3 As %String(MAXLEN = 5000);
}
```

The following business operation can accept either of these messages, create the appropriate email message, and send that via the configured SMTP server:

Class Definition

```
Class EEMA.EmailOperation Extends Ens.BusinessOperation
{
Parameter ADAPTER = "EnsLib.EMail.OutboundAdapter";
Parameter INVOCATION = "Queue";

Method SendSimpleMessage(pRequest As EEMA.SimpleMsg,
Output pResponse As Ens.Response) As %Status
{
Set email=##class(%Net.MailMessage).%New()

//Get info from pReq
Do email.TextData.Write(pRequest.Message)
Set email.Subject=pRequest.Subject

//simple case--do not check for failed recipients
Set tSc=..Adapter.SendMail(email)

//send an empty response message after message is sent
Set pResponse=##class(Ens.Response).%New()

;Quit tSc
}

Method SendMultipartEmailMessage(pRequest As EEMA.MultipartMsg,
Output pResponse As Ens.Response) As %Status
{
Set part1=##class(%Net.MailMessage).%New()
Do part1.TextData.Write(pRequest.Part1)
Set part2=##class(%Net.MailMessage).%New()
```

```

Do part2.TextData.Write(pRequest.Part2)
Set part3=##class(%Net.MailMessage).%New()
Do part3.TextData.Write(pRequest.Part3)

Set email=##class(%Net.MailMessage).%New()
Set email.Subject=pRequest.Subject
Set email.IsMultiPart=1
Do email.Parts.SetAt(part1,1)
Do email.Parts.SetAt(part2,2)
Do email.Parts.SetAt(part3,3)

//simple case--do not check for failed recipients
Set tSc=..Adapter.SendMail(email)

//send an empty response message after message is sent
Set pResponse=##class(Ens.Response).%New()

Quit tSc
}

XData MessageMap
{
<MapItems>
  <MapItem MessageType="EEMA.SimpleMsg">
    <Method>SendSimpleMessage</Method>
  </MapItem>
  <MapItem MessageType="EEMA.MultipartMsg">
    <Method>SendMultipartEmailMessage</Method>
  </MapItem>
</MapItems>
}
}

```

2.6 Adding and Configuring the Business Operation

To add your business operation to a production, use the Management Portal to do the following:

1. Add an instance of your custom business operation class to the production.
2. Enable the business operation.
3. [Specify an SMTP mail server and credentials needed to access it.](#)
4. [Optionally specify additional addresses for the email messages.](#)
5. Run the production.

2.6.1 Specifying How to Log Into the SMTP Server

To specify the SMTP server to use and any associated login credentials, specify values for the following settings of `EnsLib.Email.OutboundAdapter`:

- [SMTP Server](#)
- [SMTP Port](#)
- [Credentials](#)
- [SSL Configuration](#)

For example, you could use values like the following:

Setting	Value
SMTP Server	smtp.hotpop.com
SMTP Port	25
Credentials	hotpop

In this example, hotpop is the ID of the production credentials that consist of the username isctest@hotpop.com and the corresponding password.

2.6.2 Specifying Additional Email Addresses

You can use the following settings of `EnsLib.EMAIL.OutboundAdapter` to specify email addresses for email messages sent by this adapter:

- [Recipient](#)
- [Cc](#)
- [From](#)

For any settings not listed here, see [Configuring Productions](#).

Email Adapter Settings

This section provides reference information for settings for the email adapters.

Settings for the Email Inbound Adapter

Provides reference information for settings of the email inbound adapter.

Summary

The inbound email adapter has the following settings:

Group	Settings
Basic Settings	POP3 Server , POP3 Port , Credentials , Call Interval
Connection Settings	SSL Configuration , SSL Check Server Identity
OAuth2	OAuth2 Authorization Properties , OAuth2 Authorization Workflow Role , OAuth2 Callback Handler , OAuth2 Client Application Name , OAuth2 Grant Type , OAuth2 Scope
OAuth2 Grant Specific	OAuth2 JWT Subject
Additional Settings	Match From , Match To , Match Subject

The remaining settings are common to all business services. For information, see [Settings for All Business Services](#).

Call Interval

Number of seconds that the adapter will wait before checking again for new email, before checking for a shutdown signal from the production framework.

If the adapter finds input, it acquires the data and passes it to the business service. The business service processes the data, and then the adapter immediately begins waiting for new input. This cycle continues whenever the production is running and the business service is enabled and scheduled to be active.

The default Call Interval is 5 seconds. The minimum is 0.1 seconds.

Credentials

ID of the production credentials that contain the username and password of a valid mailbox on the given POP3 server. See [Defining Production Credentials](#).

Match From

A list of strings to look for in the `From:` field of incoming email messages, separated by semicolons (;). If this setting is null, the `From:` field is ignored when selecting messages to retrieve. For details, see [Specifying the Messages to Retrieve](#).

Match Subject

A list of strings to look for in the `Subject:` field of email messages, separated by semicolons (;). If this setting is null, the `Subject:` field is ignored when selecting messages to retrieve. For details, see [Specifying the Messages to Retrieve](#).

Match To

A list of strings to look for in the `To:` field of email messages, separated by semicolons (;). If this setting is null, the `To:` field is ignored when selecting messages to retrieve. For details, see [Specifying the Messages to Retrieve](#).

OAuth2 Authorization Properties

Extra properties in the grant flow authorization process.

Comma separated key value pairs such as `access_type=offline,prompt=consent`.

OAuth2 Authorization Workflow Role

The Workflow Role to which authorization requests will be sent depending on Grant Type flow. For more information on workflow roles, see [Workflow Roles and Users](#).

OAuth2 Callback Handler

If **OAuth2 Client Application Name** is specified, this class is used to handle obtaining the access token. The default is `Ens.Util.XOAuth2.Handler` which can be subclassed for access token retrieval customization.

OAuth2 Client Application Name

The OAuth2 Client Configuration Application name to use. This is the client configuration created in InterSystems IRIS® OAuth 2.0 settings. If specified, sub classes can use this as an indication that OAuth 2.0 is to be used and the name is used in the Authorization and Access Token retrieval process.

OAuth2 Grant Type

This is the grant type flow that the OAuth2 Callback Handler will follow. Ability to follow the grant flow type will depend on the OAuth2 Callback Handler implementation, as well as InterSystems IRIS® and the external OAuth2 server's support for the grant type flow.

OAuth2 JWT Subject

This is the JWT Subject when using the JWT Authorization Grant Type flow. For more information on JWTs, please see [Introduction to JSON Web Tokens](#).

OAuth2 Scope

This specifies the scope included in the authorization request. If not specified, it uses the default scope specified in the OAuth2 Client Application Name.

POP3 Port

TCP port on the POP3 email server to get mail from. The default value is 110.

POP3 Server

Address of the POP3 email server to get mail from.

SSL Configuration

The name of an existing TLS configuration to use to authenticate this connection. Choose a client TLS configuration, because the adapter initiates the communication.

To create and manage TLS configurations, use the Management Portal. See InterSystems TLS Guide. The first field on the **Edit SSL/TLS Configuration** page is **Configuration Name**. Use this string as the value for the **SSLConfig** setting.

SSL Check Server Identity

When connecting to a POP3 or SMTP server via TLS, the server name in the certificate must match the DNS name used to connect to the server. This match is based on the rules in section 3.1 of RFC 2818.

Settings for the Email Outbound Adapter

Provides reference information for settings of the settings outbound adapter.

Summary

The outbound email adapter has the following settings:

Group	Settings
Basic Settings	SMTP Server , SMTP Port , Credentials
Connection Settings	SSL Configuration , SSL Check Server Identity
OAuth2	OAuth2 Authorization Properties , OAuth2 Authorization Workflow Role , OAuth2 Callback Handler , OAuth2 Client Application Name , OAuth2 Grant Type , OAuth2 Scope
OAuth2 Grant Specific	OAuth2 JWT Subject
Additional Settings	Recipient , Cc , Bcc , From , ContinuelInvalidRecipient

The remaining settings are common to all business services. For information, see [Settings for All Business Services](#).

Bcc

Specifies a comma-separated list of email addresses to *add* to the `Bcc`: list of each mail message sent.

Cc

Specifies a comma-separated list of email addresses to *add* to the `Cc`: list of each mail message sent.

ContinuelInvalidRecipient

If selected, the adapter continues to send the message if one or more of the recipients have an invalid address.

Credentials

The ID of the production credentials that can authorize a connection to the given server. See [Defining Production Credentials](#).

From

The default `From`: address to put in mail messages. May be overridden by the business operation implementation code.

OAuth2 Authorization Properties

Extra properties in the grant flow authorization process.

Comma separated key value pairs such as `access_type=offline,prompt=consent`.

OAuth2 Authorization Workflow Role

The Workflow Role to which authorization requests will be sent depending on Grant Type flow. For more information on workflow roles, see [Workflow Roles and Users](#).

OAuth2 Callback Handler

If **OAuth2 Client Application Name** is specified, this class is used to handle obtaining the access token. The default is `Ens.Util.XOAuth2.Handler` which can be subclassed for access token retrieval customization.

OAuth2 Client Application Name

OAuth2 Client Configuration Application name to use. This is the client configuration created in InterSystems IRIS® OAuth 2.0 settings. If specified, sub classes can use this as an indication that OAuth 2.0 is to be used and the name is used in the Authorization and Access Token retrieval process.

OAuth2 Grant Type

This is the grant type flow that the OAuth2 Callback Handler will follow. Ability to follow the grant flow type will depend on the OAuth2 Callback Handler implementation, as well as InterSystems IRIS® and the external OAuth2 server's support for the grant type flow.

OAuth2 JWT Subject

This is the JWT Subject when using the JWT Authorization Grant Type flow. For more information on JWTs, please see [Introduction to JSON Web Tokens](#).

OAuth2 Scope

This specifies the scope included in the authorization request. If not specified, it uses the default scope specified in the OAuth2 Client Application Name.

Recipient

Specifies a comma-separated list of email addresses to *add* to the `To`: list of each mail message sent.

SMTP Port

The port on the SMTP server to send mail to. The default value is 25.

SMTP Server

The IP address of the SMTP server to send mail to. (Note: the timeouts for connecting and sending mail can be more than 10 minutes).

SSL Configuration

The name of an existing TLS configuration to use to authenticate this connection. Choose a client TLS configuration, because the adapter initiates the communication.

To create and manage TLS configurations, use the Management Portal. See InterSystems TLS Guide. The first field on the **Edit SSL/TLS Configuration** form is **Configuration Name**. Use this string as the value for the **SSLConfig** setting.

Generally, when you specify a value for this setting, outbound email opens a socket on default port 465 and uses SMTP over TLS. However, the **SSL Config** setting also supports the server interaction described in the RFC3207 standard. Specifically, you can enable the system to initiate the connection by opening a standard TCP socket and then switching to a TLS connection on the same port. The system achieves the switch by issuing the issue the `STARTTLS` command. To enable this special type of connection, you include an asterisk at the end of the **SSL Config** value, for example, `MySSLItem*`. The default SMTP port in this case is 25.

For further information, see the description of the `SSLConfig` property in the Class Reference entry for `EnsLib.Email.OutboundAdapter`.

SSL Check Server Identity

When connecting to a POP3 or SMTP server via TLS, the server name in the certificate must match the DNS name used to connect to the server. This match is based on the rules in section 3.1 of RFC 2818.