Using File Adapters in Productions

Version 2019.4
2020-01-28

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using File Adapters in Productions
InterSystems IRIS Data Platform Version 2019.4 2020-01-28
Copyright © 2020 InterSystems Corporation
All rights reserved.

InterSystems, InterSystems IRIS, InterSystems Caché, InterSystems Ensemble, and InterSystems HealthShare are registered trademarks of InterSystems Corporation.

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 Memorial Drive, Cambridge, MA 02142, 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

About This Book **1**

1 Using the File Inbound Adapter **3**
- 1.1 Overall Behavior 3
- 1.2 Creating a Business Service to Use the Inbound Adapter 4
- 1.3 Implementing the OnProcessInput() Method 5
  - 1.3.1 Invoking Adapter Methods 6
- 1.4 Example Business Service Classes 6
  - 1.4.1 Example 1 6
  - 1.4.2 Example 2 7
  - 1.4.3 Example 3 8
- 1.5 Adding and Configuring the Business Service 9

2 Using the File Outbound Adapter **11**
- 2.1 Overall Behavior 11
- 2.2 Creating a Business Operation to Use the Adapter 11
- 2.3 Creating Message Handler Methods 13
  - 2.3.1 Calling Adapter Methods from the Business Operation 13
- 2.4 Example Business Operation Class 15
- 2.5 Adding and Configuring the Business Operation 16

3 Using the File Passthrough Service and Operation Classes **17**

Reference for Settings **19**
- Settings for the File Inbound Adapter 20
- Settings for the File Outbound Adapter 25

About This Book

This book describes how configure and use the simple file adapters that InterSystems IRIS® data platform provides (the adapters in the EnsLib.File package). This book contains the following sections:

- Using the File Inbound Adapter
- Using the File Outbound Adapter
- Using the File Passthrough Service and Operation Classes
- Reference for Settings

For a detailed outline, see the table of contents.

The following books provide related information:

- *Best Practices for Creating Productions* describes best practices for organizing and developing productions.
- *Developing Productions* explains how to perform the development tasks related to creating a production.
- *Configuring Productions* describes how to configure the settings for productions, business hosts, and adapters. It provides details on settings not discussed in this book.

1

Using the File Inbound Adapter

This chapter describes how to use the file inbound adapter (EnsLib.File.InboundAdapter). It contains the following sections:

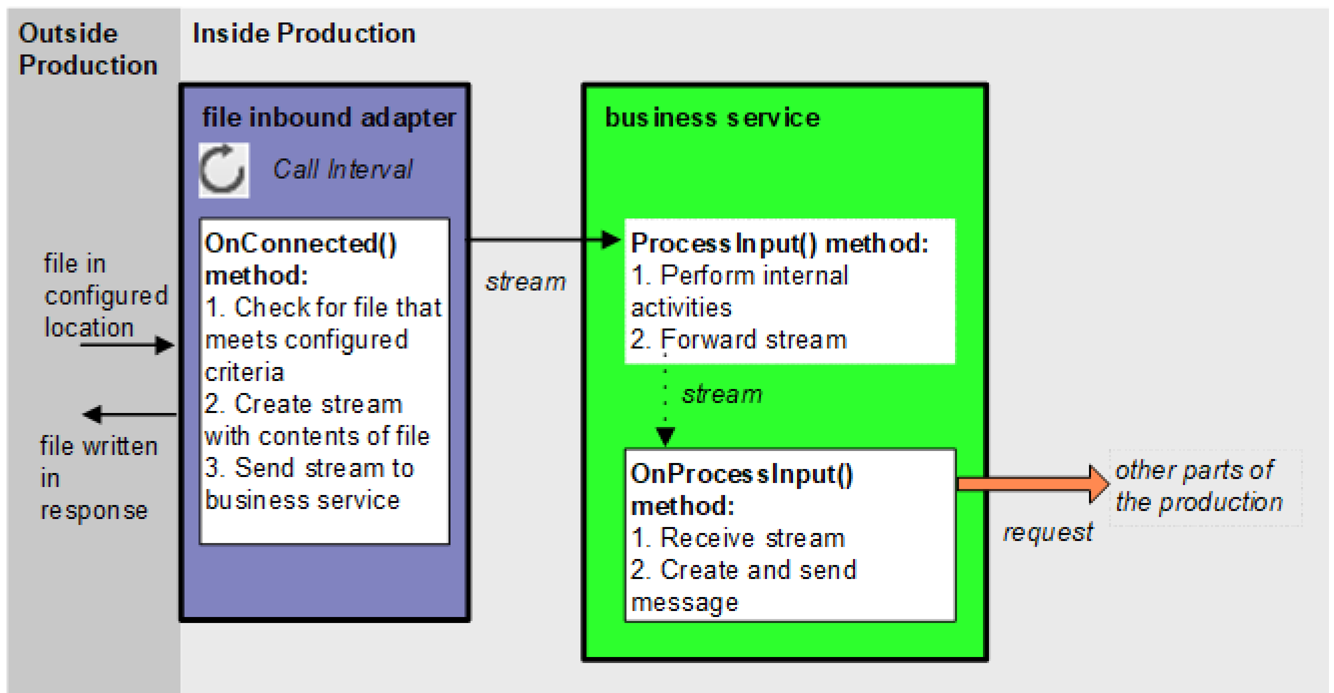
- Overall Behavior
- Creating a Business Service to Use the Inbound Adapter
- Implementing the OnProcessInput() Method
- Example Business Service Classes
- Adding and Configuring the Business Service

Tip: InterSystems IRIS® also provides specialized business service classes that use this adapter, and one of those might be suitable for your needs. If so, no programming would be needed. See the section "Connectivity Options" in *Introducing Interoperability Productions*.

1.1 Overall Behavior

EnsLib.File.InboundAdapter finds a file in the configured location, reads the input, and sends the input as a stream to the associated business service. The business service, which you create and configure, uses this stream and communicates with the rest of the production. If the inbound file adapter finds multiple files in the configured location, it processes them in order of the time, earliest first, based on when the file was last modified. But the adapter ignores any fractional seconds in the time value. Consequently, if two or more files have a modified date-time differing only in the fractional second part of the time, the adapter can process them in any order.

The following figure shows the overall flow:



In more detail:

1. Each time the adapter encounters input from its configured data source, it calls the internal **ProcessInput()** method of the business service class, passing the stream as an input argument.

2. 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.

3. The ProcessInput() method then calls your custom OnProcessInput() method, passing the stream object 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 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" in *Developing Productions*.

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.File.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" in *Developing Productions*.

The following example shows the general structure that you need:

```
Class EFILE.Service Extends Ens.BusinessService
{
Parameter ADAPTER = "EnsLib.File.InboundAdapter";

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

The first argument to OnProcessInput() could instead be %FileBinaryStream, depending on the contents of the expected file.

Note: Studio provides a wizard that you can use to create a business service stub similar to the preceding. To access this wizard, click **File —> New** and then click the **Production** tab. Then click **Business Service** and click **OK**. Note that the wizard provides a generic input argument. If you use the wizard, InterSystems recommends that you edit the method signature to use the specific input argument needed with this adapter; the input argument type should be %FileCharacterStream or %FileBinaryStream.

1.3 Implementing the OnProcessInput() Method

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

```
Method OnProcessInput(pInput As %FileCharacterStream,pOutput As %RegisteredObject) As %Status
```

Or:

```
Method OnProcessInput(pInput As %FileBinaryStream,pOutput As %RegisteredObject) As %Status
```

Where:

- *pInput* is the message object that the adapter will send to this business service. This can be of type %FileCharacterStream or %FileBinaryStream, depending on the contents of the expected file. You use an adapter setting (Charset) to indicate whether the input file is character or binary; see "Settings for the File Inbound Adapter."

  In either case, `pInput.Attributes("Filename")` equals the name of the file.
- *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 input file (*pInput*) and decide how to use it.
2. 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" in *Developing Productions*.
3. For the request message, set its properties as appropriate, using values in the input.
4. 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" in *Developing Productions*

   Each of these methods returns a status (specifically, an instance of %Status).
5. 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.
6. Return an appropriate status. This step is required.

1.3.1 Invoking Adapter Methods

Within your business service, you might want to invoke the following instance methods of the adapter. Each method corresponds to an adapter setting; these methods provide the opportunity to make adjustments following a change in any setting. For detailed descriptions of each setting, see "Settings for the File Inbound Adapter," later in this chapter.

ArchivePathSet()

```
Method ArchivePathSet(pInVal As %String) As %Status
```

pInVal is the directory where the adapter should place a copy of each file after processing.

FilePathSet()

```
Method FilePathSet(path As %String) As %Status
```

path is the directory on the local server in which to look for files.

WorkPathSet()

```
Method WorkPathSet(path As %String) As %Status
```

WorkPath

path is the directory on the local server in which to place files while they are being processed.

1.4 Example Business Service Classes

1.4.1 Example 1

The following code example shows a business service class that references the EnsLib.File.InboundAdapter. This example works as follows:

1. The file has a header. The header information is added to each transaction.
2. The file experiences a number of transactions.
3. The header and transaction XML structures are defined by the classes LBAPP.Header and LBAPP.Transaction (not shown).
4. Some error-handling is shown, but not all.
5. The method **RejectBatch()** is not shown.
6. The transactions are submitted to the business process asynchronously, so there is no guarantee they are processed in order as they appear in the file.
7. The entire transaction object is passed as the payload of each message to the business process.
8. All of the transactions in one file are submitted as a single InterSystems IRIS session.

```
Class LB.MarketOfferXMLFileSvc Extends Ens.BusinessService
{
Parameter ADAPTER = "EnsLib.File.InboundAdapter";

Method OnProcessInput(pInput As %FileCharacterStream,
                      pOutput As %RegisteredObject) As %Status
{
```

```
 // pInput is a %FileCharacterStream containing the file xml

 set batch=pInput.Filename // path+name.ext
 set batch=##class(%File).GetFilename(batch) // name.ext

 // Load the data from the XML stream into the database
 set reader = ##class(%XML.Reader).%New()

 // first get the header
 set sc=reader.OpenStream(pInput)
 if 'sc {
   do $this.RejectBatch("Invalid XML Structure",sc,pInput,batch)
   quit 1
   }
 do reader.Correlate("Header","LBAPP.Header")
 if (reader.Next(.object,.sc)) {set header=object}
 else {
   if 'sc {do $this.RejectBatch("Invalid Header",sc,pInput,batch)}
   else {do $this.RejectBatch("No Header found",sc,pInput,batch)}
   quit 1
   }

 // then get the transactions, and call the BP for each one
 do reader.Correlate("Transaction","LBAPP.Transaction")
 while (reader.Next(.object,.sc)) {
   set object.Header=header
   set sc=$this.ValidateTrans(object)
   if sc {set sc=object.%Save()}
   if 'sc {
   do $this.RejectTrans("Invalid transaction",sc,object,batch,tranct)
   set sc=1
   continue
   }

 // Call the BP for each Transaction
 set request=##class(LB.TransactionReq).%New()
 set request.Tran=object
 set ..%SessionId="" // make each transaction a new session
 set sc=$this.SendRequestAsync("LB.ChurnBPL",request)
 }

 do reader.Close()
 quit sc
}
}
```

1.4.2 Example 2

The following code example shows another business service class that uses the EnsLib.File.InboundAdapter. Code comments explain the activities within **OnProcessInput()**:

```
Class training.healthcare.service.SrvFilePerson Extends Ens.BusinessService
{

Parameter ADAPTER = "EnsLib.File.InboundAdapter";

Method OnProcessInput(pInput As %RegisteredObject,
                      pOutput As %RegisteredObject) As %Status
{

  //file must be formatted as set of lines, each field comma separated:
  //externalcode,
  //name, surname, dateBirth, placeBirth, provinceBirth
  //nationality, gender,
  //address, city, province, country,
  //fiscalCode
  //note:
  //fiscalCode may be optional
  //sso is an internal code so must be detected inside InterSystems IRIS Interoperability
  //operation must be detected as well:
  //if the group: name, surname, dateBirth, placeBirth, provinceBirth
  //point to a record then it's an UPDATE; if not it's a NEW
  //no DELETE via files

  Set $ZT="trap"

  set counter=1  //records read
  while 'pInput.AtEnd {
    set line=pInput.ReadLine()

    set req=##class(training.healthcare.message.MsgPerson).%New()
```

```
    set req.source="FILE"

    set req.externalCode=$piece(line,",",1)
    set req.name=$piece(line,",",2)
    set req.surname=$piece(line,",",3)
    set req.dateBirth=$piece(line,",",4)
    set req.placeBirth=$piece(line,",",5)
    set req.provinceBirth=$piece(line,",",6)
    set req.nationality=$piece(line,",",7)
    set req.gender=$piece(line,",",8)
    set req.address=$piece(line,",",9)
    set req.city=$piece(line,",",10)
    set req.province=$piece(line,",",11)
    set req.country=$piece(line,",",12)
    set req.fiscalCode=$piece(line,",",13)


    //call the process
    //res will be Ens.StringResponse type message
    set st=..SendRequestAsync(
           "training.healthcare.process.PrcPerson", req)
    if 'st
    $$$LOGERROR("Cannot call PrcMain Process for Person N°" _ counter)

    set counter=counter+1
  }

  $$$LOGINFO("Persons loaded : " _ (counter - 1))
  Set $ZT=""
  Quit $$$OK

trap
  $$$LOGERROR("Error loading for record N°" _ counter _ " - " _ $ZERROR)
  SET $ECODE = ""
  Set $ZT=""
  Quit $$$OK
}

}
```

1.4.3 Example 3

The following code example shows a business service class that references the EnsLib.File.InboundAdapter.

```
Class EnsLib.File.PassthroughService Extends Ens.BusinessService
{

Parameter ADAPTER = "EnsLib.File.InboundAdapter";

/// Configuration item(s) to which to send file stream messages
Property TargetConfigNames As %String(MAXLEN = 1000);

Parameter SETTINGS = "TargetConfigNames";

/// Wrap the input stream object in a StreamContainer message object
/// and send it. If the adapter has a value for ArchivePath, send async;
/// otherwise send synchronously to ensure that we don't return to the
/// Adapter and let it delete the file before the target Config Item is
/// finished processing it.

Method OnProcessInput(pInput As %Stream.Object,
                      pOutput As %RegisteredObject) As %Status
{
  Set tSC=$$$OK, tSource=pInput.Attributes("Filename"),
                 pInput=##class(Ens.StreamContainer).%New(pInput)
  Set tWorkArchive=(""'=..Adapter.ArchivePath)&&(..Adapter.ArchivePath=
    ..Adapter.WorkPath || (""=..Adapter.WorkPath &&
    (..Adapter.ArchivePath=..Adapter.FilePath)))
  For iTarget=1:1:$L(..TargetConfigNames, ",")
 {
    Set tOneTarget=$ZStrip($P(..TargetConfigNames,",",iTarget),"<>W")
    Continue:""=tOneTarget
    $$$sysTRACE("Sending input Stream ...")
    If tWorkArchive {
      Set tSC1=..SendRequestAsync(tOneTarget,pInput)
      Set:$$$ISERR(tSC1) tSC=$$$ADDSC(tSC,tSC1)
    } Else {
      #; If not archiving send Sync to avoid Adapter deleting file
      #; before Operation gets it
      Set tSC1=..SendRequestSync(tOneTarget,pInput)
      Set:$$$ISERR(tSC1) tSC=$$$ADDSC(tSC,tSC1)
```

```
    }
  }
  Quit tSC
}
}
```

This example sets the *tSource* variable to the original file name which is stored in the *Filename* subscript of the Attributes property of the incoming stream (*pInput*).

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 business service class to the production.
2. Configure the business service. For information on the settings, see "Reference for Settings."
3. Enable the business service.
4. Run the production.

2
Using the File Outbound Adapter

This chapter describes how to use the file outbound adapter (EnsLib.File.OutboundAdapter). It contains the following sections:

- Overall Behavior
- Creating a Business Operation to Use the Outbound Adapter
- Creating Message Handler Methods
- Example Business Operation Class
- Adding and Configuring the Business Operation

Tip: InterSystems IRIS® also provides specialized business service classes that use this adapter, and one of those might be suitable for your needs. If so, no programming would be needed. See the section "Connectivity Options" in *Introducing Interoperability 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.

2.2 Creating a Business Operation to Use the Adapter

To create a business operation to use EnsLib.File.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" in *Developing Productions*.

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.File.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 is 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, the method code will refer to properties and methods of the Adapter property of your business operation.

For information on defining message classes, see "Defining Messages" in *Developing Productions*.

For information on defining the message handler methods, see "Creating Message Handler Methods," later in this chapter.

- For other options and general information, see "Defining a Business Operation Class" in *Developing Productions*.

The following example shows the general structure that you need:

```
Class EHTP.NewOperation1 Extends Ens.BusinessOperation
{
Parameter ADAPTER = "EnsLib.File.OutboundAdapter";

Parameter INVOCATION = "Queue";

Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
{
  Quit $$$ERROR($$$NotImplemented)
}

XData MessageMap
{
<MapItems>
  <MapItem MessageType="RequestClass">
    <Method>Sample</Method>
  </MapItem>
</MapItems>
}
}
```

Note: Studio provides a wizard that you can use to create a business operation stub similar to the preceding. To access this wizard, click **File —> New** and then click the **Production** tab. Then click **Business Operation** and click **OK**.

2.3 Creating Message Handler Methods

When you create a business operation class for use with EnsLib.File.OutboundAdapter, typically your biggest task is writing message handlers for use with this adapter, that is, methods that receive production messages and then write files.

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. Using the information from the inbound request, call a method of the Adapter property of your business operation. The following example calls the EnsLib.File.OutboundAdapter method **PutString()**:

   ```
   /// Send an approval to the output file
   Method FileSendReply(pRequest As Demo.Loan.Msg.SendReply,
                        Output pResponse As Ens.Response) As %Status
   {
     $$$TRACE("write to file "_pRequest.Destination)
     Set tSC=..Adapter.PutString(pRequest.Destination, pRequest.Text)
     Quit tSC
   }
   ```

   You can use similar syntax to call any of the EnsLib.File.OutboundAdapter methods described in "Calling Adapter Methods from the Business Operation."
3. Make sure that you set the output argument (`pOutput`). Typically you set this equal to the response message. This step is required.
4. Return an appropriate status. This step is required.

2.3.1 Calling Adapter Methods from the Business Operation

Your business operation class can use the following instance methods of EnsLib.File.OutboundAdapter.

CreateTimestamp()

```
ClassMethod CreateTimestamp(pFilename As %String = "",
                            pSpec As %String = "_%C") As %String
```

Using the *pFilename* string as a starting point, incorporate the time stamp specifier provided in *pSpec* and return the resulting string. The default time stamp specifier is `_%C` which provides the full date and time down to the millisecond.

For full details about time stamp conventions, see "Time Stamp Specifications for Filenames" in *Configuring Productions*.

Delete()

```
Method Delete(pFilename As %String) As %Status
```

Deletes the file.

Exists()

```
Method Exists(pFilename As %String) As %Boolean
```

Returns 1 (True) if the file exists, 0 (False) if it does not.

GetStream()

```
Method GetStream(pFilename As %String,
                 ByRef pStream As %AbstractStream = {$$$NULLOREF})
                 As %Status
```

Gets a stream from the file.

NameList()

```
Method NameList(Output pFileList As %ListOfDataTypes,
                pWildcards As %String = "*",
                pIncludeDirs As %Boolean = 0) As %Status
```

Get a list of files in the directory specified by the FilePath setting. The filenames are returned in a %ListOfDataTypes object. Each entry in the list is a semicolon-separated string containing:

Filename ; *Type* ; *Size* ; *DateCreated* ; *DateModified* ; *FullPathName*

PutLine()

```
Method PutLine(pFilename As %String, pLine As %String) As %Status
```

Writes a string to the file and appends to the string the characters specified in the LineTerminator property. By default, the LineTerminator is a carriage return followed by a line feed (ASCII 13, ASCII 10).

If your operating system requires a different value for the LineTerminator property, set the value in the **OnInit**() method of the business operation. For example:

```
Method OnInit() As %Status
 {
     Set ..Adapter.LineTerminator="$C(10)"
     Quit $$$OK
 }
```

You can also make the property value to be dependent on the operating system:

```
Set ..Adapter.LineTerminator="$Select($$$isUNIX:$C(10),1:$C(13,10))"
```

PutString()

```
Method PutString(pFilename As %String, pData As %String) As %Status
```

Writes a string to the file.

PutStream()

```
Method PutStream(pFilename As %String,
                 pStream As %Stream,
                 ByRef pLen As %Integer = -1) As %Status
```

Writes a stream to the file.

Rename()

```
Method Rename(pFilename As %String,
              pNewFilename As %String,
              pNewPath As %String = "") As %Status
```

Renames the file in the current path or moves it to the path specified by *pNewPath*.

2.4 Example Business Operation Class

The following code example shows a business operation class that references the EnsLib.File.OutboundAdapter. This class can perform two operations: If it receives valid Person data, it files Person information based on Person status. If it receives invalid Person data, it logs this information separately.

```
Class training.operation.OpeFilePerson extends Ens.BusinessOperation
{

Parameter ADAPTER = "EnsLib.File.OutboundAdapter";

Parameter INVOCATION = "Queue";

/* write on log file wrong person records */
Method writeMessage(
       pRequest As MyData.Message,
       Output pResponse As Ens.StringResponse)
       As %Status
{
  $$$LOGINFO("called  Writer")

  set ..Adapter.FilePath="C:\Intersystems\test\ftp"

  set st=..Adapter.PutLine("person.log",message)

  Quit $$$OK
}

/* write on log file wrong person records */
Method logWrongPerson(
       pRequest As training.healthcare.message.MsgPerson,
       Output pResponse As Ens.StringResponse)
       As %Status
{
  $$$LOGINFO("called OpeFilePerson")

  set ..Adapter.FilePath="C:\Intersystems\test\errorparh"
  set message="some information are missing from record: " _
              pRequest.sso _ ", " _
              pRequest.name _ ", " _
              pRequest.surname

  set st=..Adapter.PutLine("Person.log",message)

  Quit $$$OK
}

/* write in xml format the list of active/inactive/requested Persons */
Method writeSSOList(
       pRequest As Ens.StringRequest,
       Output pResponse As Ens.StringResponse)
       As %Status
{
  set ..Adapter.FilePath="C:\Intersystems\test\ftp"
  set status=pRequest.StringValue

  if status="ACTIVE" set fileName="ActiveSSO.xml"
  if status="INACTIVE" set fileName="InactiveSSO.xml"
  if status="REQUESTED" set fileName="RequestedSSO.xml"

  set st=..Adapter.PutLine(fileName,"<Persons>")

  set rs=
  ##class(training.healthcare.data.TabPerson).selectPersons("",status)
  while rs.Next(){
    set st=..Adapter.PutLine(fileName,"<Person>")
    for i=1:1:rs.GetColumnCount() {
      set st=..Adapter.PutLine(fileName,
        "<"_ rs.GetColumnName(i)_">" _
        rs.GetData(i)_"</"_ rs.GetColumnName(i)_">")
    }
    set st=..Adapter.PutLine(fileName,"<Person>")
  }

  set st=..Adapter.PutLine(fileName,"<Persons>")
```

```
  set pResponse=##class(Ens.StringResponse).%New()
  set pResponse.StringValue="done"

  quit $$$OK
}

XData MessageMap
{
<MapItems>
  <MapItem MessageType="training.healthcare.message.MsgPerson">
    <Method>logWrongPerson</Method>
  </MapItem>
   <MapItem MessageType="Ens.StringRequest">
    <Method>writeSSOList</Method>
  </MapItem>
</MapItems>
}

}
```

2.5 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 business operation class to the production.
2. Configure the business operation. For information on the settings, see “Reference for Settings.”
3. Enable the business operation.
4. Run the production.

3

Using the File Passthrough Service and Operation Classes

InterSystems IRIS® also provides two general purposes classes to send and receive files in any format. These classes are as follows:

- EnsLib.File.PassthroughService receives files of any format
- EnsLib.File.PassthroughOperation sends files of any format

EnsLib.File.PassthroughService provides the setting, **Target Config Names**, which allows you to specify a comma-separated list of other configuration items within the production to which the business service should relay the message. Usually the list contains one item, but it can be longer. **Target Config Names** can include business processes or business operations.

EnsLib.File.PassthroughOperation provides the **File Name** setting, which allows you to specify an output file name. The **FileName** can include InterSystems IRIS Interoperability time stamp specifiers. For full details, see "Time Stamp Specifications for Filenames" in *Configuring Productions*.

Reference for Settings

This section provides the following reference information:

- Settings for the File Inbound Adapter
- Settings for the File Outbound Adapter

Also see “Settings in All Productions” in *Managing Productions Productions*.

Settings for the File Inbound Adapter

Provides reference information for settings of the file inbound adapter, EnsLib.File.InboundAdapter.

Summary

The inbound file adapter has the following settings:

Group	Settings
Basic Settings	File Path, File Spec, Archive Path, Work Path, Call Interval
Additional Settings	Subdirectory Levels, Charset, Append Timestamp, Semaphore Specification, Confirm Complete, File Access Timeout

The remaining settings are common to all business services. For information, see "Settings for All Business Services" in *Configuring Productions*.

Append Timestamp

Append a time stamp to filenames in the **Archive Path** and **Work Path** directories; this is useful to prevent possible name collisions on repeated processing of the same filename.

- If this value is empty or 0, no time stamp is appended.
- If this setting is 1, then the standard template '%f_%Q' is appended.
- For other possible values, see "Time Stamp Specifications for Filenames" in *Configuring Productions*.

Archive Path

Full pathname of the directory where the adapter should place the input file after it has finished processing the data in the file. This directory must exist, and it must be accessible through the file system on the local InterSystems IRIS® Interoperability machine. If this setting is not specified, the adapter deletes the input file after its call to **ProcessInput()** returns.

To ensure that the input file is not deleted while your production processes the data from the file, InterSystems recommends that you set **Archive Path** and **Work Path** to the same directory. Alternatively, you can use only synchronous calls from your business service to process the data.

Call Interval

The polling interval for this adapter, in seconds. This is the time interval at which the adapter checks for input files in the specified locations.

Upon polling, if the adapter finds a file, it links the file to a stream object and passes the stream object to the associated business service. If several files are detected at once, the adapter sends one request to the business service for each individual file until no more files are found.

If the business service processes each file synchronously, the files will be processed sequentially. If the business service sends them asynchronously to a business process or business operation, the files might be processed simultaneously.

After processing all the available files, the adapter waits for the polling interval to elapse before checking for files again. This cycle continues whenever the production is running and the business service is enabled and scheduled to be active.

It is possible to implement a callback in the business service so that the adapter delays for the duration of the **Call Interval** between input files. For details, see "Defining Business Services" in *Developing Productions*.

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

Charset

Specifies the character set of the input file. InterSystems IRIS automatically translates the characters from this character encoding. The setting value is not case-sensitive. Use `Binary` for binary files, or for any data in which newline and line feed characters are distinct or must remain unchanged. Other settings may be useful when transferring text documents. Choices include:

- `Binary` — Binary transfer
- `Ascii` — Ascii mode FTP transfer but no character encoding translation
- `Default` — The default character encoding of the local InterSystems IRIS server
- `Latin1` — The ISO Latin1 8-bit encoding
- `ISO-8859-1` — The ISO Latin1 8-bit encoding
- `UTF-8` — The Unicode 8-bit encoding
- `UCS2` — The Unicode 16-bit encoding
- `UCS2-BE` — The Unicode 16-bit encoding (Big-Endian)
- Any other alias from an international character encoding standard for which NLS (National Language Support) is installed in InterSystems IRIS

Use a value that is consistent with your implementation of **OnProcessInput()** in the business service:

- When the Charset setting has the value `Binary`, the *pInput* argument of **OnProcessInput()** is of type %FileBinaryStream and contains bytes.
- Otherwise, *pInput* is of type %FileCharacterStream and contains characters.

For background information on character translation in InterSystems IRIS, see "Localization Support" in the *Orientation Guide for Server-Side Programming*.

Semaphore Specification

The Semaphore Specification allows you to indicate that the data file is complete and ready to be read by creating a second file that is used as a semaphore. The inbound file adapter waits until the semaphore file exists before checking the other conditions specified by the Confirm Complete requirements and then processing the data file. This allows the application creating the data file to ensure that the adapter waits until the data file is complete before processing it. The adapter tests only for the existence of the semaphore file and does not read the semaphore file contents.

If the Semaphore Specification is an empty string, the adapter does not wait for a semaphore file and processes the data file as soon as the conditions specified by the Confirm Complete requirements are met. If you are using a semaphore file to control when the adapter processes the data file, you should consider setting the Confirm Complete field to None.

The Semaphore Specification allows you to specify individual semaphore files for each data file or a single semaphore file to control multiple data files. You can use wildcards to pair semaphore files with data files, and can specify a series of patterns matching semaphore files to data files. The adapter always looks for a matching semaphore file in the same directory as the data file. If the adapter is looking for data files in subdirectories, the semaphore file must be in the same subdirectory level as its corresponding data file.

The general format for specifying the Semaphore Specification is:

```
[DataFileSpec=] SemaphoreFileSpec [;[DataFileSpec=] SemaphoreFileSpec]...
```

For example, if the Semaphore Specification is:

```
ABC*.TXT=ABC*.SEM
```

It means that the ABCTest.SEM semaphore file controls when the adapter processes the ABCTest.TXT file and that the ABCdata.SEM semaphore file controls when the adapter processes the ABCdata.txt file.

Note: In a semaphore specification, the * (asterisk) matches any character except dot. In a file specification, the asterisk matches any character including the dot.

You can have one semaphore file control multiple data files. For example, if the Semaphore Specification is:

```
*.DAT=DATA.SEM
```

The DATA.SEM semaphore file controls when the adapter processes all *.DAT files in the same directory. When the adapter is looking for data files and corresponding semaphore files, it loops through all the data files at a polling interval. With the previous Semaphore Specification, if it started looking for DATA.SEM for the ABC.DAT file and does not find it, it continues looking for the semaphore files for the other files. But, if during this process DATA.SEM is created and it is looking for a match for XYZ.DAT, it finds the corresponding semaphore file. But the adapter defers processing XYZ.DAT until the next polling interval because a preceding data file, ABC.DAT, was waiting for the same semaphore file.

If you specify multiple pairings, separate them with a ; (semicolon). For example, if the Semaphore Specification is:

```
*.TXT=*.SEM;*.DAT=*.READY
```

The semaphore file MyData.SEM controls when the adapter processes MyData.TXT, but the semaphore file MyData.READY controls when it processes MyFile.DAT.

The adapter finds the corresponding semaphore file for each data file by reading the Semaphore Specification from left to right. Once it determines the corresponding semaphore file, it stops reading the Semaphore Specification for that file. For example, if the Semaphore Specification is:

```
VIData.DAT=Special.SEM; *.DAT=*.SEM
```

The adapter looks for the semaphore file Special.SEM before it processes VIData.DAT, but it does not consider VIData.SEM as a semaphore file for VIData.DAT. It does consider stuff.SEM as the semaphore file for stuff.DAT because stuff.DAT did not match an earlier specification. Consequently, if you are including multiple specifications that can match the same file, you should specify the more specific specification before the more general ones.

The data file target pattern is case-sensitive and the semaphore pattern case sensitivity is operating system dependent, that is *.TXT=*.SEM .is only applied to target files found ending with capitalized .TXT but the operating system may not differentiate between *.SEM and *.sem. If the operating system is not case-sensitive, the adapter treats semaphore files ending in any case combination of *.SEM and *.sem as equivalent but only uses them as the semaphore for data files named *.TXT. It cannot distinguish case in the semaphore files but can distinguish it in the data files.

If you only specify a single file specification and omit the = (equals) sign, the adapter treats that as the Semaphore Specification for all data files. For example, if the Semaphore Specification is:

```
*.SEM
```

This is equivalent to specifying a single wildcard to the left of the = (equals) sign:

```
*=*.SEM
```

In this case, the semaphore file MyFile.SEM controls the data file MyFile.txt and the semaphore file BigData.SEM controls the data file BigData.DAT.

If no wildcard is used in the Semaphore Specification then it is the complete fileSpec for the semaphore file. For example, if the Semaphore Specification is:

```
*.DAT=DataDone.SEM
```

Then the DataDone.SEM semaphore file controls when the adapter reads any data file with the .DAT file extension.

If a Semaphore Specification is specified and a data file does not match any of the patterns, then there is no corresponding semaphore file and the adapter will not process this data file. You can avoid this situation by specifying * as the last data file in the Semaphore Specification. For example, if the Semaphore Specification is:

```
*.DAT=*.SEM; *.DOC=*.READY; *=SEM.LAST
```

The `SEM.LAST` is the semaphore file for all files that do not end with `.DAT` or `.DOC`.

If an adapter configured with a FileSpec equal to `*`, the adapter usually considers all files in the directory as data files. But, if the adapter also has a Semaphore Specification and it recognizes a file as a semaphore file, it does not treat it as a data file.

After the adapter has processed through all the data files in a polling cycle, it deletes all the corresponding semaphore files.

Confirm Complete

Indicates the special measures that InterSystems IRIS should take to confirm complete receipt of a file. The options are:

List option	Integer value	Description
None	0	Take no special measures to determine if a file is complete.
Size	1	Wait until the reported size of the file in the **FilePath** directory stops increasing. This option may not be sufficient when the source application is sluggish. If the operating system reports the same file size for a duration of the **File Access Timeout** setting, then InterSystems IRIS Interoperability considers the file complete.
Rename	2	Read more data for a file until the operating system allows InterSystems IRIS to rename the file.
Readable	4	Consider the file complete if it can open it in *Read* mode.
Writable	8	Consider the file complete if it can open it in *Write* mode (as a test only; it does not write to the file).

The effectiveness of each option depends on the operating system and the details of the process that puts the file in the **File Path** directory.

File Access Timeout

The default is 10 seconds.

File Path

Full pathname of the directory in which to look for files. This directory must exist, and it must be accessible through the file system on the local InterSystems IRIS Interoperability machine.

File Spec

Filename or wildcard file specification for file(s) to retrieve. For the wildcard specification, use the convention that is appropriate for the operating system on the local InterSystems IRIS Interoperability machine.

Subdirectory Levels

Number of levels of subdirectory depth under the given directory that should be searched for files.

Work Path

Full pathname of the directory where the adapter should place the input file while processing the data in the file. This directory must exist, and it must be accessible through the file system on the local InterSystems IRIS Interoperability

machine. This setting is useful when the same filename is used for repeated file submissions. If no **WorkPath** is specified, the adapter does not move the file while processing it.

To ensure that the input file is not deleted while your production processes the data from the file, InterSystems recommends that you set **Archive Path** and **Work Path** to the same directory. Alternatively, you can use only synchronous calls from your business service to process the data.

Settings for the File Outbound Adapter

Provides reference information for settings of the file outbound adapter, EnsLib.File.OutboundAdapter.

Summary

The outbound file adapter has the following settings:

Group	Settings
Basic Settings	File Path
Additional Settings	Overwrite, Charset, Open Timeout

The remaining settings are common to all business operations. For information, see “Settings for All Business Operations” in *Configuring Productions*.

Charset

Specifies the desired character set for the output file. InterSystems IRIS® automatically translates the characters to this character encoding. See “Charset” in “Settings for the File Inbound Adapter.”

File Path

Full pathname of the directory into which to write output files. This directory must exist, and it must be accessible through the file system on the local InterSystems IRIS Interoperability machine.

Open Timeout

Amount of time for the adapter to wait on each attempt to open the output file for writing.

The default is 5 seconds.

Overwrite

If a file of the same name exists in the FilePath directory, the Overwrite setting controls what happens. If True, overwrite the file. If False, append the new output to the existing file.