InterSystems™
IRIS Data Platform

Using File Adapters in Productions

Version 2025.1
2025-06-03

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

Using File Adapters in Productions
PDF generated on 2025-06-03
InterSystems IRIS® Version 2025.1
Copyright © 2025 InterSystems Corporation
All rights reserved.

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 File Inbound Adapter

This topic describes how to use the file inbound adapter (EnsLib.File.InboundAdapter).

Tip: InterSystems IRIS® data platform 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 Business Host Classes That Use File Adapters.

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 adapter can look for a file by its exact name, or you can use a wildcard specification. 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.

Note that 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.

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 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.

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.

The following example shows the general structure that you need:

Class Definition

```
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: The classes %FileCharacterStream and %FileBinaryStream are both deprecated except for use in product APIs such as this one (the EnsLib.File.InboundAdapter).

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.
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.

   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.

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 Understanding the Adapter Archiving Behavior

After the business service sends a request to some destination within the production, the adapter may archive or delete the input file that triggered the request. The following table describes the archiving behavior of the adapter given various settings.

You can use the table to choose the combination of settings that best suits your environment. For example, if the production uses a Message Bank operation to track message bodies from the business service, you can use the first scenario to ensure that the contents of files are archived to the Message Bank before the file streams are removed. For more information, see Configuring the Enterprise Message Bank. You can use the third and fourth scenarios to permanently retain archived input files. You can use the sixth scenario to trigger an event on the target host that is independent of the contents of the input file since the file may be deleted, causing a potential race condition.

Note: The adapter can rename or delete a file sent to host only if the method does not return an error.

In all the scenarios except the third and fourth scenarios, InterSystems IRIS purges the input file during a manual or scheduled purge if the Include message bodies setting is set to `true`.

CAUTION: Be careful with message purges in the scenario where:

- The file specification (the File Spec setting) does not include a wildcard.
- And your settings do not cause the system to move a file when processing it.

In this scenario, unless you time the purge appropriately, it would be possible for the system to purge a file that has not yet been processed.

Scenario	ArchivePath and WorkPath	Request Type	File Sent to Host	File Location
1	**ArchivePath** and **WorkPath** are the same, but different from **FilePath**	Async	Input file renamed **ArchivePath** + filename (with optional timestamp)	**ArchivePath** + filename (with optional timestamp)
2	**ArchivePath** and **WorkPath** are not set	Sync	Input file	If **Delete From Server** is `true`, none If **Delete From Server** is `false`, input directory
3	**ArchivePath** is different from **FilePath**, and **WorkPath** is not set	Sync	Input file	**ArchivePath** + filename (with optional timestamp)
4	**ArchivePath** is different from **WorkPath**, which is different from **FilePath**	Sync	Input file renamed **WorkPath** + filename (with optional timestamp)	**ArchivePath** + filename (with optional timestamp)
5	**ArchivePath** is not set, and **WorkPath** is different from **FilePath**	Sync	Input file renamed **WorkPath** + filename (with optional timestamp)	If **Delete From Server** is `true`, none If **Delete From Server** is `false`, **WorkPath** + filename (with optional timestamp)
6	**ArchivePath** is the same as **FilePath**, and **WorkPath** is not set	Async	Input file	If **Delete From Server** is `true`, none If **Delete From Server** is `false`, input directory

1.5 Example Business Service Classes

1.5.1 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 Definition

```
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
}
}
```

1.5.2 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 Definition

```
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
}

}
```

1.5.3 Example 3

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

Class Definition

```
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 you move the input file to the ArchivePath or delete the file
/// after sending, send the message object synchronously. Doing so prevents
/// a race condition, that is, a situation where the adapter attempts to
/// delete or modify the file while the target Config Item is still processing it.
/// Alternatively, send the object asynchronously.
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 {
      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*).

InterSystems recommends sending an asynchronous request only if you do not intend to move or delete the input file. For additional guidance, see Understanding the Adapter Archiving Behavior.

1.6 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 Settings for the File Inbound Adapter.
3. Enable the business service.

4. Run the production.

1.7 See Also

- Business Host Classes That Use File Adapters
- Using the File Outbound Adapter
- Using the File Passthrough Service and Operation Classes
- Settings for the File Inbound Adapter

2

Using the File Outbound Adapter

This topic describes how to use the file outbound adapter (EnsLib.File.OutboundAdapter).

Tip: InterSystems IRIS® data platform 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 Business Host Classes That Use File Adapters.

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.

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.

  For information on defining the message handler methods, see Creating Message Handler Methods.

- 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 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>
}
}
```

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()**:

   Class Member

   ```
   /// 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.

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 Definition

```
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 Settings for the File Outbound Adapter.
3. Enable the business operation.
4. Run the production.

2.6 See Also

- Business Host Classes That Use File Adapters
- Using the File Inbound Adapter
- Using the File Passthrough Service and Operation Classes
- Settings for the File Outbound Adapter

3 Using the File Passthrough Service and Operation Classes

InterSystems IRIS® data platform also provides two general purpose 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.

3.1 See Also

- Business Host Classes That Use File Adapters
- Using the File Inbound Adapter
- Using the File Outbound Adapter
- File Adapter Settings

File Adapter Settings

This section provides reference information for the file inbound and outbound adapters:

Also see Settings in All 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, Fatal Errors, Header Count, Confirm Complete, File Access Timeout

The remaining settings are common to all business services. For information, see Settings for All Business Services.

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.

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, use this adapter in a custom business service and send only synchronous calls from this business service.

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.

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

Note: The classes %FileCharacterStream and %FileBinaryStream are both deprecated except for use in product APIs such as EnsLib.File.InboundAdapter.

Semaphore Specification

The Semaphore Specification allows you to indicate that the data file or files are complete and ready to be read by creating an associated second file in the same directory 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 or files. 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 files as soon as the conditions specified by the Confirm Complete requirements are met.

If you are using the Semaphore Specification feature, consider setting the Confirm Complete field to None.

Syntax

Semaphore Specification can be an empty string or can be a series of pairs, each of which associates a data filename specification with a semaphore filename pattern. The pairs are separated by semicolons:

```
DataFileSpec=SemaphorePattern;DataFileSpec=SemaphorePattern;...
```

DataFileSpec is either a plain filename or a filename specification that includes the * wildcard (which matches any character). *SemaphorePattern* directly or indirectly specifies the name of the associated semaphore file; it can be either of the following:

- A plain filename (such as `SemaphoreFile.SEM`). In this case, when the system finds a file that matches *DataFileSpec*, the system looks for a file with that exact name.

- A string of the form `*.`*extension* such as `*.sem`. In this case, when the system finds a file that matches *DataFileSpec*, the system looks for a file with the same name, but with the `sem` extension instead. For example, if the filename `ABCDEF.txt` matches *DataFileSpec*, the system looks for a semaphore file named `ABCDEF.sem`

  When looking for a semaphore file based on a data filename, the system looks only at the part of the data filename before the first period. For example, if the filename `test.txt.data.zip.tar` matches *DataFileSpec*, if the semaphore filename pattern is `*.sem` the system looks for a semaphore file named `test.sem`

Notes:

- The semaphore file associated with a given data file (or multiple data files) must be in the same directory as those files.
- *DataFileSpec* and *SemaphorePattern* do not include the directory name.
- *DataFileSpec* is always case-sensitive.
- *SemaphorePattern* is case-sensitive if the operating system is case-sensitive and is not case-sensitive otherwise.
- The pairs are processed left-to-right, and the first matching pair is used; see How the Inbound File Adapter Uses Semaphore 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.
- If an adapter configured with a FileSpec equal to `*`, the adapter usually considers all files in the directory as data files. But if Semaphore Specification is also specified, the adapter can recognize a file as a semaphore file and not treat it as a data file.

For example, consider the following Semaphore Specification, consisting of a single pair:

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

In this case, 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`.

In the simplest case, Semaphore Specification can consist only of a single *SemaphorePattern*, which may or may not include a wildcard. This means that the presence or absence of that semaphore file controls whether the adapter processes any files.

Files That Do Not Match a Pattern

If a Semaphore Specification is specified and a given data file does not match any of the patterns, then the adapter will not process this data file. If this is undesirable, specify a final pair that will match any file and that uses its own semaphore file. For example, consider this Semaphore Specification:

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

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

How the Inbound File Adapter Uses Semaphore Specification

Within each polling cycle, the inbound file adapter examines all the files found within the configured directory (and any subdirectories). Then for each file:

1. The adapter reads the Semaphore Specification from left-to-right, finding the first specification whose *DataFileSpec* matches the given filename. This indicates the name of the semaphore file to look for.
2. The adapter looks for the semaphore file in the same directory as the file being examined.

   Then:

   - If it does not find the semaphore file, the adapter skips the file and sets an internal flag that causes the adapter to wait until the next polling cycle.

- If it does find the semaphore file, the adapter processes the file.

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

Fatal Errors

For record map services, determines whether the system stops processing a message when it encounters an error such as a validation error in an individual record. When configuring the adapter, choose one of the following options:

- `Any`—This is the default. If InterSystems IRIS encounters an error when saving an individual record, it stops processing the message.
- `ParseOnly`—If InterSystems IRIS encounters an error when saving an individual record, it logs the error, skips the record, and then continues parsing the message. The log includes the position in the stream of the invalid record. Additionally, if **Alert On Error** is enabled, the system generates an alert.

Header Count

For record map services, determines the number of lines that the service ignores as prefix lines in incoming documents. Ignoring prefix lines enables the services to parse reports and comma-separated values (CSV) files with column headers.

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

Amount of time in seconds that the system waits for information from the source application before confirming the complete receipt of a file. For more information, see Confirm Complete.

If you supply a decimal value, the system rounds the value up to the nearest whole number. The default value is 2.

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.

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.