プロダクション内でのファイル・アダプタの使用法

Version 2023.1
2024-01-02

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

プロダクション内でのファイル・アダプタの使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

ファイル受信アダプタの使用法

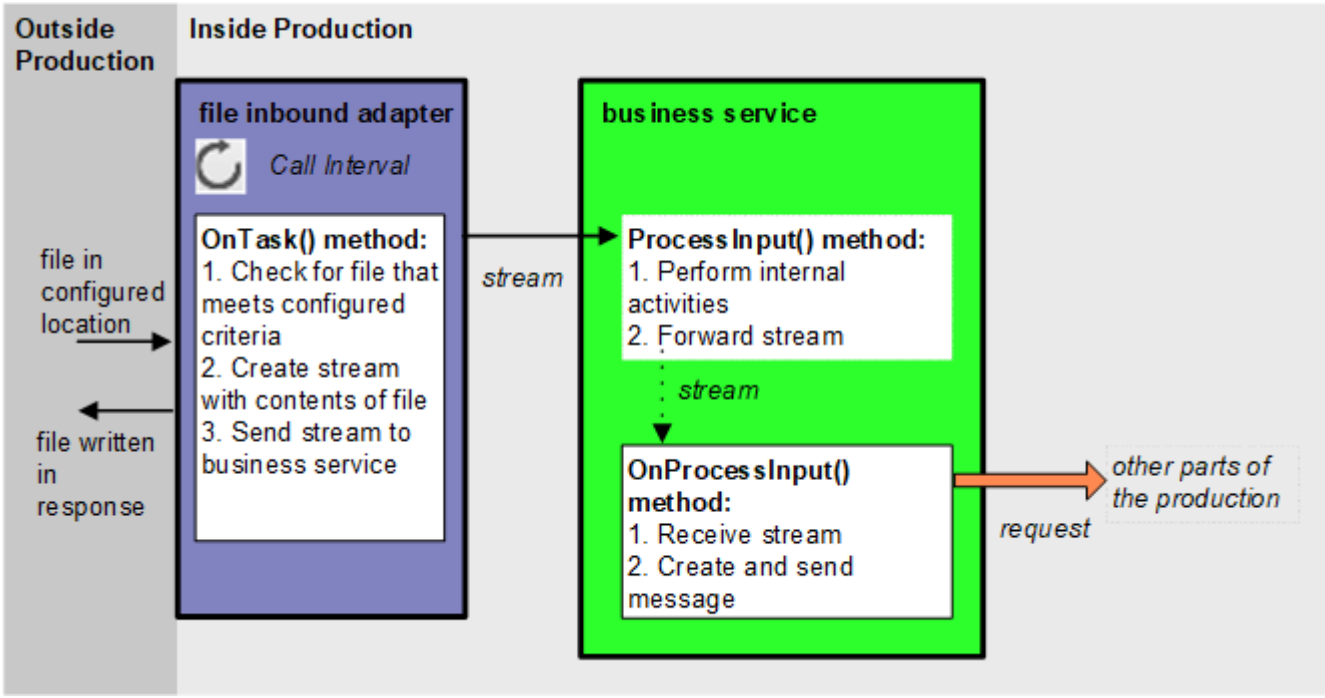
この章では、ファイル受信アダプタ (EnsLib.File.InboundAdapter) の使用方法について説明します。

Tipヒント InterSystems IRIS® では、このアダプタを使用する特殊なビジネス・サービス・クラスとユーザのニーズに適したビジネス・サービス・クラスの 1 つも提供されます。そのため、プログラミングの必要がありません。″相互運用プロダクションの概要″ の “接続オプション” のセクションを参照してください。

1.1 全般的な動作

EnsLib.File.InboundAdapter は、構成された場所でファイルを検索したり、入力を読み取ったり、入力をストリームとして関連するビジネス・サービスに送信したりします。ユーザが作成および構成するビジネス・サービスは、このストリームを使用してプロダクションの他の部分と通信します。受信ファイル・アダプタが構成された場所に複数のファイルを見つけた場合は、ファイルの最終変更時刻に基づいて、それが早いものから順番に処理します。ただし、アダプタは、時刻値の小数点以下の秒数は無視します。そのため、2 つ以上のファイルの変更日付と時刻が、時刻の小数点以下の秒数部分のみ異なる場合は、アダプタはこれらを任意の順序で処理します。

以下の図は、全体的なフローを示しています。



詳細は、以下のとおりです。

1. アダプタは、構成されたデータ・ソースからの入力を検出するたびに、ビジネス・サービス・クラスの内部 ProcessInput() メソッドを呼び出して、ストリームを入力引数として渡します。
2. ビジネス・サービス・クラスの内部 ProcessInput() メソッドが実行されます。このメソッドは、すべてのビジネス・サービスが必要とする内部情報の保持など、基本的なプロダクション・タスクを実行します。ビジネス・サービス・クラスが継承するこのメソッドは、カスタマイズや上書きを行いません。
3. 次に、ProcessInput() メソッドがカスタムの OnProcessInput() メソッドを呼び出し、ストリーム・オブジェクトを入力として渡します。このメソッドに関する要件については、後述の“OnProcessInput() メソッドの実装”で説明します。

応答メッセージは、同じパスを逆向きにたどります。

1.2 受信アダプタを使用するビジネス・サービスの作成

このアダプタをプロダクションで使用するには、ここに記載されているように新しいビジネス・サービスを作成します。後で、そのサービスをプロダクションに追加して構成します。存在しなければ、該当するメッセージ・クラスを作成する必要もあります。"プロダクションの開発" の“メッセージの定義”を参照してください。

ビジネス・サービス・クラスの基本要件を以下に列挙します。

- ビジネス・サービス・クラスは Ens.BusinessService を拡張するものでなければなりません。
- クラス内の ADAPTER パラメータは EnsLib.File.InboundAdapter と一致する必要があります。
- クラスは、“OnProcessInput メソッドの実装”に記載されているように、OnProcessInput() メソッドを実装する必要があります。
- その他のオプションと一般情報は、"プロダクションの開発" の“ビジネス・サービス・クラスの定義”を参照してください。

以下の例は、必要となる一般的な構造を示しています。

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
}
}
```

代わりに、予想されるファイルの内容に応じて、OnProcessInput() への最初の引数を %FileBinaryStream にすることができます。

注釈　スタジオには、上記のようなビジネス・サービス・スタブの作成に使用できるウィザードが用意されています。このウィザードにアクセスするには、[ファイル]→[新規作成] をクリックし、[プロダクション] タブをクリックします。次に [ビジネス・サービス] をクリックして [OK] をクリックします。このウィザードには、汎用入力引数が用意されています。ウィザードを使用する場合は、このアダプタに必要な特定の入力引数を使用するようにメソッド・シグニチャを編集することをお勧めします。入力引数の型は %FileCharacterStream または %FileBinaryStream にする必要があります。

1.3 OnProcessInput() メソッドの実装

ビジネス・サービス・クラス内では、OnProcessInput() メソッドに次のシグニチャを含める必要があります。

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

または：

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

説明：

- pInput は、アダプタがこのビジネス・サービスに送信するメッセージ・オブジェクトです。これは、予想されるファイルの内容に応じて、%FileCharacterStream と %FileBinaryStream のどちらかの型にすることができます。アダプタ設定 ([文字セット]) を使用して、入力ファイルが文字かバイナリかを指定します。"“ファイル受信アダプタに関する設定”" を参照してください。

  いずれの場合も、`pInput.Attributes("Filename")` はファイルの名前と一致します。

- pOutput は、メソッド・シグニチャに必要な汎用出力引数です。

OnProcessInput() メソッドは、以下の一部またはすべてを実行する必要があります。

1. 入力ファイル (pInput) を検査して、その使用方法を決定します。
2. ビジネス・サービスから送信されることになる要求メッセージのインスタンスを作成します。

   メッセージ・クラスの作成方法は、"プロダクションの開発" の “メッセージの定義” を参照してください。
3. 要求メッセージの場合は、必要に応じて、入力内の値を使用してそのプロパティを設定します。
4. ビジネス・サービスの適切なメソッドを呼び出して、要求をプロダクション内の宛先に送信します。具体的には、SendRequestSync()、SendRequestAsync()、または (あまり一般的ではない) SendDeferredResponse() を呼び出します。詳細は、"プロダクションの開発" の “要求メッセージの送信” を参照してください。

   これらの各メソッドは、ステータス (具体的には、%Status のインスタンス) を返します。

5. 必ず出力引数 (pOutput) を設定します。通常、受信した応答メッセージと同じように設定します。この手順は必須です。
6. 適切なステータスを返します。この手順は必須です。

1.3.1 アダプタ・メソッドの呼び出し

ビジネス・サービス内では、以下のアダプタのインスタンス・メソッドを呼び出すことができます。各メソッドはアダプタ設定に対応しています。これらのメソッドによって、設定の変更後に調整の機会が与えられます。各設定の詳細は、後述する“ファイル受信アダプタに関する設定”を参照してください。

ArchivePathSet()

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

pInVal は、アダプタがファイルの処理を完了した後に、各ファイルのコピーを格納するディレクトリです。

FilePathSet()

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

path は、ローカル・サーバ上のファイルの検索先ディレクトリです。

WorkPathSet()

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

WorkPath

path は、ファイルの処理中にファイルを格納するローカル・サーバ上のディレクトリです。

1.4 アダプタのアーカイブ動作の理解

ビジネス・サービスがプロダクション内の宛先に要求を送信した後、アダプタはその要求をトリガした入力ファイルをアーカイブまたは削除できます。次のテーブルは、さまざまな設定が指定されたアダプタのアーカイブ動作を示しています。

このテーブルを使用して、ご使用の環境に最適な設定の組み合わせを選択してください。例えば、プロダクションでメッセージ・バンク・オペレーションを使用して、ビジネス・サービスからのメッセージ・ボディを追跡する場合、1 つ目のシナリオを使用して、ファイル・ストリームが削除される前に、ファイルの内容がメッセージ・バンクにアーカイブされていることを確認できます。詳細は、″エンタープライズ・メッセージ・バンクの構成″を参照してください。3 つ目と 4 つ目のシナリオは、アーカイブされた入力ファイルを永続的に保持する場合に使用できます。6 つ目のシナリオは、競合状態が生じるため削除される可能性があることにより、入力ファイルの内容には依存していないターゲット・ホストでイベントをトリガする場合に使用できます。

3 つ目と 4 つ目以外のすべてのシナリオで、[メッセージ・ボディを含む] 設定が に設定されている場合、InterSystems IRIS は、手動またはスケジュールされた削除の際に入力ファイルを削除します。

注釈　アダプタがホストに送信されたファイルを名前変更または削除できるのは、メソッドがエラーを返さない場合のみです。

シナリオ	ArchivePath と WorkPath	リクエスト・タイプ	ホストに送信されたファイル	ファイルの場所
1	ArchivePath と WorkPath は同じだが、FilePath とは異なる	非同期	ArchivePath + ファイル名 (オプションでタイムスタンプ付き) に名前変更された入力ファイル	ArchivePath + ファイル名 (オプションでタイムスタンプ付き)
2	ArchivePath と WorkPath の設定なし	同期	入力ファイル	[サーバから削除] が の場合、なし [サーバから削除] が の場合、入力ディレクトリ 注釈 [サーバから削除] は、ビジネス・サービスをカスタマイズすることによってのみ、 に設定できます。詳細は、"設定の追加と削除" を参照してください。
3	ArchivePath は FilePath とは異なり、WorkPath の設定はなし	同期	入力ファイル	ArchivePath + ファイル名 (オプションでタイムスタンプ付き)
4	ArchivePath は WorkPath とは異なり、WorkPath は FilePath とは異なる	同期	WorkPath + ファイル名 (オプションでタイムスタンプ付き) に名前変更された入力ファイル	ArchivePath + ファイル名 (オプションでタイムスタンプ付き)
5	ArchivePath の設定はなし、WorkPath は FilePath とは異なる	同期	WorkPath + ファイル名 (オプションでタイムスタンプ付き) に名前変更された入力ファイル	[サーバから削除] が の場合、なし [サーバから削除] が の場合、WorkPath + ファイル名 (オプションでタイムスタンプ付き)
6	ArchivePath は FilePath と同じで、WorkPath の設定はなし	非同期	入力ファイル	[サーバから削除] が の場合、なし [サーバから削除] が の場合、入力ディレクトリ

1.5 ビジネス・サービス・クラスの例

1.5.1 例 1

EnsLib.File.InboundAdapter を参照するビジネス・サービス・クラスのコードの例を以下に示します。この例は、以下のように動作します。

1. ファイルにはヘッダがあります。ヘッダ情報が各トランザクションに追加されます。
2. ファイルには多数のトランザクションがあります。
3. ヘッダとトランザクション XML 構造が、LBAPP.Header クラスと LBAPP.Transaction クラスによって定義されます (これは表示されていません)。
4. エラー処理がいくつか表示されていますが、すべてではありません。
5. RejectBatch() メソッドは表示されていません。
6. トランザクションは非同期でビジネス・プロセスに送信されるため、ファイル内の順番どおりに処理されるという保証はありません。
7. トランザクション・オブジェクト全体が各メッセージのペイロードとしてビジネス・プロセスに渡されます。
8. 1 つのファイル内のすべてのトランザクションが、1 つの InterSystems IRIS セッションで送信されます。

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 例 2

EnsLib.File.InboundAdapter を使用するビジネス・サービス・クラスのコードの例を以下に示します。コードのコメントは、OnProcessInput() 内のアクティビティを説明しています。

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 例 3

EnsLib.File.InboundAdapter を参照するビジネス・サービス・クラスのコードの例を以下に示します。

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
}
}
```

この例では、tSource 変数を、受信ストリーム (pInput) が持つ Attributes プロパティの Filename 添え字に格納されている元のファイル名に設定します。

非同期の要求を送信するのは、入力ファイルを移動または削除するつもりがない場合のみにすることをお勧めします。詳細は、"アダプタのアーカイブ動作の理解" を参照してください。

1.6 ビジネス・サービスの追加と構成

ビジネス・サービスをプロダクションに追加するには、管理ポータルを使用して以下の操作を行います。

1. ビジネス・サービス・クラスのインスタンスをプロダクションに追加します。
2. ビジネス・サービスを構成します。設定の詳細は、“設定の参照先” を参照してください。
3. ビジネス・サービスを有効化します。

4. プロダクションを実行します。

2
ファイル送信アダプタの使用法

この章では、ファイル送信アダプタ (EnsLib.File.OutboundAdapter) の使用方法について説明します。

Tipヒント InterSystems IRIS® では、このアダプタを使用する特殊なビジネス・サービス・クラスとユーザのニーズに適したビジネス・サービス・クラスの 1 つも提供されます。そのため、プログラミングの必要がありません。"相互運用プロダクションの概要" の "接続オプション" のセクションを参照してください。

2.1 全般的な動作

プロダクション内で、送信アダプタは、ユーザが作成および構成するビジネス・オペレーションに関連付けられます。このビジネス・オペレーションはプロダクション内からメッセージを受信し、メッセージ・タイプを調べ、適切なメソッドを実行します。このメソッドは、通常、関連するアダプタのメソッドを実行します。

2.2 アダプタを使用するビジネス・オペレーションの作成

EnsLib.File.OutboundAdapter を使用するビジネス・オペレーションを作成するために、新しいビジネス・オペレーション・クラスを作成します。後で、そのクラスをプロダクションに追加して構成します。

存在しなければ、該当するメッセージ・クラスを作成する必要もあります。"プロダクションの開発" の "メッセージの定義" を参照してください。

ビジネス・オペレーション・クラスの基本要件を以下に列挙します。

- ビジネス・オペレーション・クラスは、Ens.BusinessOperation を拡張するものでなければなりません。
- クラスの ADAPTER パラメータは EnsLib.File.OutboundAdapter と一致する必要があります。
- クラスの INVOCATION パラメータは、使用する呼び出しスタイルを指定する必要があります。以下のいずれかを使用します。
  - Queue は、メッセージが 1 つのバックグラウンド・ジョブ内で作成され、元のジョブが解放された段階でキューに配置されます。後で、メッセージが処理されるときに、別のバックグラウンド・ジョブがこのタスクに割り当てられます。これは最も一般的な設定です。
  - InProc は、メッセージが、作成されたジョブと同じジョブで生成、送信、および配信されることを意味します。このジョブは、メッセージが対象に配信されるまで送信者のプールに解放されません。これは特殊なケースのみに該当します。

- クラスでは、少なくとも 1 つのエントリを含むメッセージ・マップを定義します。メッセージ・マップは、以下の構造を持つ XData ブロック・エントリです。

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

- クラスでは、メッセージ・マップ内で名前が付けられたすべてのメソッドを定義します。これらのメソッドは、メッセージ・ハンドラと呼ばれます。各メッセージ・ハンドラは、以下のシグニチャを持っている必要があります。

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

  ここで、Sample はメソッド名、RequestClass は要求メッセージ・クラスの名前、ResponseClass は応答メッセージ・クラスの名前です。通常、メソッド・コードは、ビジネス・オペレーションのプロパティおよび Adapter プロパティのメソッドを参照します。

  メッセージ・クラスの定義方法は、"プロダクションの開発" の "メッセージの定義" を参照してください。

  メッセージ・ハンドラ・メソッドの定義方法は、後述する "メッセージ・ハンドラ・メソッドの作成" を参照してください。

- その他のオプションと一般情報は、"プロダクションの開発" の "ビジネス・オペレーション・クラスの定義" を参照してください。

以下の例は、必要となる一般的な構造を示しています。

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>
}
}
```

注釈　スタジオには、上記のようなビジネス・オペレーション・スタブの作成に使用できるウィザードが用意されています。このウィザードにアクセスするには、**[ファイル]→[新規作成]** をクリックし、**[プロダクション]** タブをクリックします。 次に **[ビジネス・オペレーション]** をクリックして [OK] をクリックします。

2.3 メッセージ・ハンドラ・メソッドの作成

EnsLib.File.OutboundAdapter で使用されるビジネス・オペレーション・クラスを作成する場合の最大のタスクは、このアダプタで使用されるメッセージ・ハンドラ、つまり、プロダクション・メッセージを受信してファイルを書き込むメソッドの作成です。

各メッセージ・ハンドラ・メソッドは、以下のシグニチャを持っている必要があります。

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

ここで、Sample はメソッド名、RequestClass は要求メッセージ・クラスの名前、ResponseClass は応答メッセージ・クラスの名前です。

通常、このメソッドは以下の操作を実行します。

1. 受信要求メッセージを調べます。
2. 受信要求の情報を使用して、ビジネス・オペレーションの Adapter プロパティのメソッドを呼び出します。下の例は、EnsLib.File.OutboundAdapter メソッドの PutString() を呼び出します。

   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
   }
   ```

   同様の構文を使用して、“ビジネス・オペレーションからのアダプタ・メソッドの呼び出し”に記載された EnsLib.File.OutboundAdapter メソッドのいずれかを呼び出すことができます。
3. 必ず出力引数 (pOutput) を設定します。通常、応答メッセージと同じように設定します。この手順は必須です。
4. 適切なステータスを返します。この手順は必須です。

2.3.1 ビジネス・オペレーションからのアダプタ・メソッドの呼び出し

ビジネス・オペレーション・クラスは、以下の EnsLib.File.OutboundAdapter のインスタンス・メソッドを使用できます。

CreateTimestamp()

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

開始点に pFilename 文字列を使用し、pSpec で指定されているタイム・スタンプ指定子を組み入れて、結果文字列を返します。デフォルトのタイム・スタンプ指定子は _%C で、これはミリ秒までの完全な日時を表します。

タイム・スタンプ規則の詳細は、"プロダクションの構成" の“ファイル名に関するタイム・スタンプ指定”を参照してください。

Delete()

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

ファイルを削除します。

Exists()

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

ファイルが存在するときは 1 (真)、存在しないときは 0 (偽) を返します。

GetStream()

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

ファイルからストリームを取得します。

NameList()

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

FilePath 設定で指定したディレクトリ内のファイルのリストを取得します。ファイル名は %ListOfDataTypes オブジェクト内に返されます。各エントリは、以下のようにセミコロンで区切られた文字列の列挙で構成されます。

Filename;Type;Size;DateCreated;DateModified;FullPathName

PutLine()

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

文字列をファイルに書き込み、LineTerminator プロパティで指定されている文字を文字列に付加します。デフォルトでは、LineTerminator は復帰改行で、その後に改行が続きます (ASCII 13、ASCII 10)。

LineTerminator プロパティに異なる値を必要とするオペレーティング・システムの場合は、ビジネス・オペレーションの OnInit() メソッドに値を設定します。以下に例を示します。

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

また、プロパティ値がオペレーティング・システムに依存するように設定することもできます。

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

PutString()

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

ファイルに文字列を書き込みます。

PutStream()

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

ファイルにストリームを書き込みます。

Rename()

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

現在のパスにあるファイルの名前を変更します。または、pNewPath で指定されているパスにファイルを移動します。

2.4 ビジネス・オペレーション・クラスの例

EnsLib.File.OutboundAdapter を参照するビジネス・オペレーション・クラスのコードの例を以下に示します。このクラスは 2 つの操作を実行します。このクラスが有効な Person データを受信すると、クラスは Person のステータスに基づいて Person の情報を保管します。このクラスが無効な Person データを受信すると、その情報を個別にログに記録します。

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 ビジネス・オペレーションの追加と構成

ビジネス・オペレーションをプロダクションに追加するには、管理ポータルを使用して以下の操作を行います。

1. ビジネス・オペレーション・クラスのインスタンスをプロダクションに追加します。
2. ビジネス・オペレーションを構成します。設定の詳細は、“設定の参照先”を参照してください。
3. ビジネス・オペレーションを有効化します。
4. プロダクションを実行します。

3

ファイル・パススルー・サービス・クラスとファイル・パススルー・オペレーション・クラスの使用法

InterSystems IRIS® には、任意の形式のファイルを送受信するための 2 つの汎用クラスも用意されています。そのクラスは以下のとおりです。

- EnsLib.File.PassthroughService は、任意の形式のファイルを受信します。
- EnsLib.File.PassthroughOperation は、任意の形式のファイルを送信します。

EnsLib.File.PassthroughService には [**ターゲット構成名**] 設定があります。この設定を使用すれば、ビジネス・サービスがメッセージを中継するプロダクション内の他の構成項目のカンマ区切りリストを指定できます。通常、リストには、1 つの項目が含まれますが、それ以上の項目数の場合があります。[**ターゲット構成名**] には、ビジネス・プロセスまたはビジネス・オペレーションを含めることができます。

EnsLib.File.PassthroughOperation には [**ファイル名**] 設定があります。この設定を使用すれば、出力ファイル名を指定できます。[**ファイル名**] には、InterSystems IRIS Interoperability タイム・スタンプ指定子を含めることができます。詳細は、"プロダクションの構成" の "ファイル名に関するタイム・スタンプ指定" を参照してください。

ファイル・アダプタ設定

ここでは、ファイル受信アダプタおよびファイル送信アダプタに関する参照情報を提供します。

″プロダクションの管理″ の “すべてのプロダクションに含まれる設定” も参照してください。

ファイル受信アダプタに関する設定

ファイル受信アダプタ EnsLib.File.InboundAdapter の設定に関する参照情報を提供します。

概要

受信ファイル・アダプタには以下の設定があります。

グループ	設定
基本設定	[ファイルパス]、[ファイルスペック]、[アーカイブパス]、[ワークパス]、[呼び出し間隔]
追加設定	[サブディレクトリ・レベル]、[文字セット]、[タイムスタンプ追加]、[セマフォ指定]、[深刻なエラー]、[ヘッダ・カウント]、[完了を確認]、[ファイル・アクセス・タイムアウト]

残りの設定はすべてのビジネス・サービスに共通しています。詳細は、"プロダクションの構成" の "すべてのビジネス・サービスに含まれる設定" を参照してください。

[タイムスタンプ追加]

[アーカイブパス] と **[ワークパス]** のディレクトリ内のファイル名にタイム・スタンプを付加します。これにより、同じファイル名の繰り返し処理で発生する可能性のある名前の競合を避けることができます。

- この値が空か 0 の場合は、タイム・スタンプが付加されません。
- この設定が 1 の場合は、標準テンプレート '%f_%Q' が付加されます。
- その他の可能性のある値は、"プロダクションの構成" の "ファイル名に関するタイム・スタンプ指定" を参照してください。

[アーカイブパス]

ファイル内のデータの処理を完了した後に、アダプタが入力ファイルを格納するディレクトリの完全パス名です。このディレクトリは存在するディレクトリであること、また、ローカル InterSystems IRIS® Interoperability マシンのファイル・システムからアクセス可能なディレクトリであることが必要です。この設定が指定されていない場合は、アダプタが ProcessInput() への呼び出しが復帰した後に入力ファイルを削除します。

プロダクションで入力ファイル内のデータを処理している最中にそのファイルが削除されないことを保証するために、[ArchivePath] と [WorkPath] を同じディレクトリに設定することをお勧めします。または、データ処理に、ビジネス・サービスから同期呼び出しのみを使用する方法があります。

呼び出し間隔

このアダプタの秒単位のポーリング間隔。これは、アダプタが指定された場所で入力ファイルをチェックする時間間隔です。

ポーリング時にアダプタがファイルを検出すると、アダプタはファイルをストリーム・オブジェクトにリンクし、ストリーム・オブジェクトを関連ビジネス・サービスに渡します。一度に複数のファイルを検出すると、アダプタはファイルが検出されなくなるまで、個々のファイルのビジネス・サービスに対して 1 つの要求を送信します。

ビジネス・サービスが各ファイルを同期で処理した場合、ファイルは順次処理されます。ビジネス・サービスが各ファイルをビジネス・プロセスまたはビジネス・オペレーションに非同期で送信した場合、ファイルは同時処理される場合があります。

有効なファイルをすべて処理し終わると、アダプタはポーリング間隔が経過してから、再度ファイルをチェックします。プロダクションが実行中であり、ビジネス・サービスが有効化され、アクティブになるようにスケジュールされている場合、このサイクルは常に継続されます。

入力ファイル間の [**呼び出し間隔**] 期間だけアダプタが遅れるように、ビジネス・サービス内にコールバックを実装できます。詳細は、"プロダクションの開発" の "ビジネス・サービスの定義" を参照してください。

デフォルト [**呼び出し間隔**] は 5 秒です。最小値は 0.1 秒です。

Charset

入力ファイルの文字セットを指定します。InterSystems IRIS は、自動的に、文字をこの文字エンコーディングから変換します。この設定値は大文字と小文字が区別されません。`Binary` は、バイナリ・ファイル、新規行文字と改行文字が異なるデータ、または変更しないまま残す必要のあるデータに対して使用します。テキスト・ドキュメントを転送するときは、他の設定が便利な場合があります。選択肢は以下のとおりです。

- `Binary` ― バイナリ転送
- `Ascii` ― 文字エンコード変換以外の Ascii モードの FTP 転送
- `Default` ― ローカル InterSystems IRIS サーバのデフォルトの文字エンコード
- `Latin1` ― ISO Latin1 8 ビット・エンコード
- `ISO-8859-1` ― ISO Latin1 8 ビット・エンコード
- `UTF-8` ― Unicode 8 ビット・エンコード
- `UCS2` ― Unicode 16 ビット・エンコード
- `UCS2-BE` ― Unicode 16 ビット・エンコード (ビッグ・エンディアン)
- InterSystems IRIS に NLS (各国言語サポート) をインストールするための、国際文字エンコード規格に基づくその他のエイリアス。

ビジネス・サービス内の OnProcessInput() の実装と矛盾しない値を使用します。

- **Charset** 設定に `Binary` 値が含まれている場合は、OnProcessInput() の pInput 引数が **%FileBinaryStream** 型となり、バイトで構成されます。
- そうでない場合は、pInput が **%FileCharacterStream** 型となり、文字で構成されます。

InterSystems IRIS の文字変換に関する背景情報は、"サーバ側プログラミングの入門ガイド" の "ローカライズのサポート" を参照してください。

セマフォ仕様

セマフォ仕様では、セマフォとして使用する 2 番目のファイルを作成することで、データ・ファイルの作成完了と読み取り準備の完了を示すことができるようになります。受信ファイル・アダプタは、セマフォ・ファイルが存在するまで待機した後、[完了を確認] の要件で指定されたその他の条件をチェックしてから、データ・ファイルを処理します。これにより、データ・ファイルを作成するアプリケーションでは、データ・ファイルの完了までアダプタがそのファイルの処理をしないように待機させることができます。アダプタは、セマフォ・ファイルが存在することのみをテストし、セマフォ・ファイルの内容を読み取ることはありません。

セマフォ仕様が空の文字列の場合、アダプタはセマフォ・ファイルを待機せずに、[完了を確認] の要件で指定された条件が成立すると即座にデータ・ファイルを処理します。セマフォ・ファイルを使用して、アダプタがデータ・ファイルをいつ処理するかを制御する場合は、[完了を確認] フィールドを [なし] に設定することを検討してください。

セマフォ仕様では、データ・ファイルごとに個別のセマフォ・ファイルを指定できます。また、1 つのセマフォ・ファイルで複数のデータ・ファイルを制御することもできます。セマフォ・ファイルとデータ・ファイルのペアにワイルドカードを使用することができます。さらに一連のパターン・マッチング・セマフォ・ファイルをデータ・ファイルに指定することができます。アダプタは、常にデータ・ファイルと同じディレクトリで一致するセマフォ・ファイルを検索します。アダプタがサブディレクトリのデータ・ファイルを検索する場合、セマフォ・ファイルは対応するデータ・ファイルと同じサブディレクトリのレベルに存在する必要があります。

セマフォ仕様を使用する標準的な形式は次のとおりです。

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

例えば、次のセマフォ仕様があるとします。

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

ABCTest.SEM セマフォ・ファイルは、アダプタが ABCTest.TXT ファイルをいつ処理するかを制御し、ABCdata.SEM セマフォ・ファイルは、アダプタが ABCdata.txt ファイルをいつ処理するかを制御することを意味します。

注釈　セマフォ仕様では、*(アスタリスク) はドット以外のすべての文字に対応します。ファイル仕様では、アスタリスクはドットを含むすべての文字に対応します。

1 つのセマフォ・ファイルで複数のデータ・ファイルを制御できます。例えば、次のセマフォ仕様があるとします。

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

DATA.SEM セマフォ・ファイルは、同じディレクトリ内のすべての *.DAT ファイルについてアダプタがいつ処理するかを制御します。アダプタがデータ・ファイルと対応するセマフォ・ファイルを検索するとき、ポーリングの間隔ですべてのデータ・ファイルをループします。上のセマフォ仕様の場合、ABC.DAT ファイルの DATA.SEM から検索を開始し、検出できない場合は、別のファイルのセマフォ・ファイルの検索を続けます。このプロセスで XYZ.DAT の一致を検索していたときに DATA.SEM が作成された場合には、この対応するセマフォ・ファイルが検出されます。ただし、アダプタは XYZ.DAT の処理を次のポーリングまで延期します。これは前のデータ・ファイル ABC.DAT が同じセマフォ・ファイルを待機していたためです。

複数のペアを指定する場合は、それらを ; (セミコロン) で区切ります。例えば、次のセマフォ仕様があるとします。

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

セマフォ・ファイル MyData.SEM は、アダプタが MyData.TXT をいつ処理するかを制御しますが、セマフォ・ファイル MyData.READY は、アダプタが MyFile.DAT をいつ処理するかを制御します。

アダプタは、セマフォ仕様を左から右に読み取って、各データ・ファイルに対応するセマフォ・ファイルを検出します。対応するセマフォ・ファイルを特定すると、そのファイルのセマフォ仕様の読み取りを停止します。例えば、次のセマフォ仕様があるとします。

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

アダプタは、VIData.DAT を処理する前にセマフォ・ファイル Special.SEM を検索しますが、VIData.SEM は VIData.DAT のセマフォ・ファイルと見なされません。stuff.SEM は、stuff.DAT のセマフォ・ファイルと見なされます。これは stuff.DAT が以前の仕様と一致しなかったためです。したがって、同じファイルと一致する複数の仕様を含める場合は、より限定的な仕様を指定してから、より一般化された仕様を指定する必要があります。

データ・ファイルのターゲット・パターンは大小文字が区別され、セマフォ・パターンの大小文字の区別はオペレーティング・システムに依存します。つまり、*.TXT=*.SEM は、大文字の .TXT で終わるターゲット・ファイルのみに適用されますが、オペレーティング・システムは *.SEM と *.sem を区別できない場合があります。オペレーティング・システムで大小文字が区別されない場合、アダプタは、大小文字の任意の組み合わせで終わる *.SEM と *.sem のセマフォ・ファイルを同等に処理しますが、それらは名前が *.TXT のデータ・ファイルのみに対してセマフォとして使用されます。セマフォ・ファイルでは大小文字を区別できませんが、データ・ファイルでは区別が可能です。

1 つのファイル仕様のみを指定し、= (等号) 記号を省略すると、アダプタがこれを全データ・ファイルのセマフォ仕様として扱います。例えば、次のセマフォ仕様があるとします。

```
*.SEM
```

これは、1 つのワイルドカードを = (等号) 記号の左側に指定した場合と等価です。

```
*=*.SEM
```

この場合、セマフォ・ファイル MyFile.SEM はデータ・ファイル MyFile.txt を制御し、セマフォ・ファイル BigData.SEM はデータ・ファイル BigData.DAT を制御します。

セマフォ仕様でワイルドカードが使用されていない場合、その仕様はセマフォ・ファイルの完全な fileSpec です。例えば、次のセマフォ仕様があるとします。

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

この場合、DataDone.SEM セマフォ・ファイルは、アダプタが .DAT ファイル拡張子を持つデータ・ファイルをいつ読み取るかを制御します。

セマフォ仕様が指定され、データ・ファイルがいずれのパターンとも一致しない場合、対応するセマフォ・ファイルは存在せず、アダプタはこのデータ・ファイルを処理しません。この状況は、セマフォ仕様で最後のデータ・ファイルとして * を指定することで回避できます。例えば、次のセマフォ仕様があるとします。

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

SEM.LAST は、.DAT や .DOC で終了しないすべてのファイルのセマフォ・ファイルです。

FileSpec が * でアダプタが構成されている場合、そのアダプタは、通常はディレクトリ内にあるすべてのファイルをデータ・ファイルと見なします。ただし、アダプタにセマフォ仕様も存在し、特定のファイルをセマフォ・ファイルとして認識する場合、そのファイルがデータ・ファイルとして扱われることはありません。

アダプタは、すべてのデータ・ファイルを特定のポーリング・サイクルで処理し終わると、対応する全セマフォ・ファイルを削除します。

[深刻なエラー]

レコード・マップ・サービスにおいて、個々のレコードでの検証エラーなどのエラーが発生した場合に、システムによるメッセージの処理を停止するかどうかを指定します。アダプタを構成する際は、以下のいずれかのオプションを選択します。

- Any – 既定値。InterSystems IRIS で個々のレコードの保存中にエラーが発生した場合、メッセージの処理を停止します。
- ParseOnly – InterSystems IRIS で個々のレコードの保存中にエラーが発生した場合、エラーをログに記録し、そのレコードをスキップしてから、メッセージの解析を続行します。このログには、無効なレコードのストリーム内の位置が含まれます。また、[エラー時に警告] が有効になっている場合は、システムによってアラートが生成されます。

[ヘッダ・カウント]

レコード・マップ・サービスにおいて、受信ドキュメントで先頭行としてサービスが無視する行数を指定します。先頭行を無視することで、サービスは、列ヘッダが付いたレポートおよびコンマ区切り値 (CSV) ファイルを解析できるようになります。

[完了を確認]

ファイルの受領の完了を確認するための特別な手段を指定します。選択項目は、以下のとおりです。

リスト・オプション	製数値	説明
[なし]	0	ファイルを完全に受信したかどうかを確認するための特別な方法は行いません。
Size	1	[FilePath] ディレクトリ内のファイルの報告されたサイズが増加しなくなるまで待機します。ソース・アプリケーションの動きが遅いときは、このオプションは十分ではない場合があります。オペレーティング・システムによって [ファイルアクセスタイムアウト] 設定の期間に同じファイル・サイズが報告された場合に、ファイルの受信が完了したと見なされます。

リスト・オプション	製数値	説明
Rename	2	オペレーティング・システムからファイルの名前を変更する許可があるまで、ファイルのデータ読み込みを続けます。
Readable	4	読み込みモードでファイルを開くことが可能になった場合に、ファイルの受信が完了したと見なされます。
Writable	8	書き込みモードでファイルを開くことが可能になった場合に、ファイルの受信が完了したと見なされます (テストとしてのみ。ファイルへの書き込みは行われません)。

各オプションの有効性は、オペレーティング・システムと、ファイルを [ファイルパス] ディレクトリに格納するプロセスの詳細によって異なります。

[ファイルアクセスタイムアウト]

システムがファイル受領の完了を確認する前に、ソース・アプリケーションからの情報を待機する時間 (秒単位)。詳細は、"[完了を確認]" を参照してください。

10 進数値を指定すると、最も近い整数に値が切り上げられます。既定値は 2 です。

File Path

ファイルの検索先ディレクトリの完全パス名です。このディレクトリは存在するディレクトリであること、また、ローカル InterSystems IRIS Interoperability マシンのファイル・システムからアクセス可能なディレクトリであることが必要です。

ファイルスペック

取得するファイルのファイル名またはワイルドカードのファイル仕様です。ワイルドカード仕様では、ローカルの InterSystems IRIS Interoperability マシンのオペレーティング・システムに適した規則を使用します。

[サブディレクトリレベル]

ファイルを検索するディレクトリ下のサブディレクトリの階層数。

[ワークパス]

ファイル内のデータの処理中に、アダプタが入力ファイルを格納するディレクトリの完全パス名です。このディレクトリは存在するディレクトリであること、また、ローカル InterSystems IRIS Interoperability マシンのファイル・システムからアクセス可能なディレクトリであることが必要です。この設定は、ファイル送信の繰り返しで同一ファイル名が使用される場合に便利です。[WorkPath] を指定しない場合、アダプタはファイルの処理中にファイルを移動しません。

プロダクションで入力ファイル内のデータを処理している最中にそのファイルが削除されないことを保証するために、[ArchivePath] と [WorkPath] を同じディレクトリに設定することをお勧めします。または、データ処理に、ビジネス・サービスから同期呼び出しのみを使用する方法があります。

ファイル送信アダプタに関する設定

ファイル送信アダプタ EnsLib.File.OutboundAdapter の設定に関する参照情報を提供します。

概要

送信ファイル・アダプタには以下の設定があります。

グループ	設定
基本設定	File Path
追加設定	[上書き]、[文字セット]、[オープンタイムアウト]

残りの設定はすべてのビジネス・オペレーションに共通しています。詳細は、"プロダクションの構成" の "すべてのビジネス・オペレーションに含まれる設定" を参照してください。

Charset

出力ファイルに必要な文字セットを指定します。InterSystems IRIS® は、自動的に、文字をこの文字エンコーディングに変換します。"ファイル受信アダプタに関する設定" の "文字セット" を参照してください。

File Path

出力ファイルの書き込み先ディレクトリの完全パス名です。このディレクトリは存在するディレクトリであること、また、ローカル InterSystems IRIS Interoperability マシンのファイル・システムからアクセス可能なディレクトリであることが必要です。

[オープンタイムアウト]

各出力ファイルを書き込むためにファイルを開くまでアダプタが待機する時間です。

デフォルトは 5 秒です。

Overwrite

Overwrite 設定は、FilePath ディレクトリ内に同一名のファイルが存在する場合の動作を制御します。真のとき、ファイルを上書きします。偽のとき、既存ファイルに新しい出力を付加します。