プロダクションの開発

Version 2024.1
2024-06-06

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

プロダクションの開発
InterSystems IRIS Data Platform　Version 2024.1　2024-06-06

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

プロダクションの開発の概要

このページでは、プロダクションの開発プロセスについて説明します。

InterSystems IRIS Interoperability の概念とオプションの説明は、"相互運用プロダクションの概要" を参照してください。InterSystems IRIS Interoperability の処理動作は、"プロダクションの監視" の最初の章を参照してください。

1.1 環境上の考慮事項

InterSystems IRIS Interoperability は、特定の Web アプリケーションが属している相互運用対応ネームスペース内でしか使用できません。クラスの作成では、予約パッケージ名を使用しないようにする必要があります。以降の項で詳しく説明します。

1.1.1 プロダクション対応ネームスペース

相互運用対応ネームスペースは、プロダクションをサポートするクラス、データ、およびメニューをネームスペースで使用可能にするグローバル・マッピング、ルーチン・マッピング、およびパッケージ・マッピングで構成されたネームスペースです。マッピングの一般情報は、"システム管理ガイド" の "ネームスペースの構成" を参照してください。(この節の情報を使用して、相互運用対応ネームスペースで行われる実際のマッピングを確認できます。詳細はリリースにより異なる場合がありますが、ユーザ側で特に作業する必要はありません)。

InterSystems IRIS のインストール時に作成されるシステム提供のネームスペースは、Community Edition で USER ネームスペースが相互運用対応ネームスペースである場合を除き、相互運用対応ではありません。ユーザが作成する新しいネームスペースはすべて、デフォルトで相互運用対応になります。ネームスペースを作成する際に **[ネームスペースを相互運用プロダクション対応にする]** チェック・ボックスのチェックを外すと、InterSystems IRIS は相互運用に対応しないネームスペースを作成します。

重要　システム提供のネームスペースはすべて、再インストールまたはアップグレードの際に上書きされます。このため、常に、ユーザが作成した新規ネームスペースで作業することをお勧めします。新規ネームスペースの詳細は、"システム管理ガイド" の "ネームスペースの構成" を参照してください。

1.1.2 Web アプリケーションの要件

また、プロダクションは、`/csp/`namespace (この場合の namespace はネームスペースの名前) と名付けられた Web アプリケーションが関連付けられている場合のみ、そのネームスペースで使用できます。(これは、ネームスペースに対するデフォルトの Web アプリケーション名です。)Web アプリケーションの定義の詳細は、"アプリケーション" を参照してください。

1.1.3 予約パッケージ名

相互運用対応ネームスペースでは、Demo、Ens、EnsLib、EnsPortal、または CSPX をパッケージ名として使用しないでください。これらのパッケージは、アップグレード・プロセス中に完全に置換されます。これらのパッケージ内でクラスを定義した場合は、アップグレード前にそれらのクラスをエクスポートして、アップグレード後にインポートする必要があります。

また、先頭に Ens (大文字と小文字の区別あり) が付くパッケージ名を使用しないことをお勧めします。これには次の 2 つの理由があります。

- 先頭に Ens が付く名前のパッケージにクラスをコンパイルすると、コンパイラは、生成されたルーチンを ENSLIB システム・データベースに書き込みます(コンパイラがそうするのは、名前の先頭に Ens が付くルーチンはすべて、そのデータベースにマッピングされるからです)。つまり、インスタンスをアップグレードすると、ENSLIB データベースが置換され、生成されたルーチンはアップグレードによって削除され、クラス定義のみが残ります。この時点で、クラスを使用するために、クラスのリコンパイルが必要になります。

  これに対し、名前の先頭が Ens ではないパッケージ内のクラスは、インスタンスのアップグレード時にリコンパイルする必要はありません。

- 名前の先頭に Ens が付くパッケージ内のクラスを定義すると、それらのクラスは相互運用対応のすべてのネームスペースで使用可能になりますが、このことは望ましい場合も、望ましくない場合もあります。1 つ言えることとして、パッケージの名前の先頭に Ens が付いている場合、名前が同じでコンテンツが異なる 2 つのクラスを、異なる相互運用対応ネームスペースで使用することはできなくなります。

1.2 プロダクション定義の確認

プロダクションは管理ポータルで作成して構成しますが、選択した IDE で既存のプロダクション・クラスの定義を確認することから始めることをお勧めします。以下にプロダクションの簡単な例を示します。

Class Definition

```
Class Demo.FloodMonitor.Production Extends Ens.Production
{

XData ProductionDefinition
{
<Production Name="Demo.FloodMonitor.Production">
  <ActorPoolSize>1</ActorPoolSize>
  <Item Name="Demo.FloodMonitor.BusinessService"
        ClassName="Demo.FloodMonitor.BusinessService"
        PoolSize="1" Enabled="true" Foreground="false" InactivityTimeout="0">
  </Item>
  <Item Name="Demo.FloodMonitor.CustomBusinessProcess"
        ClassName="Demo.FloodMonitor.CustomBusinessProcess"
        PoolSize="1" Enabled="true" Foreground="false" InactivityTimeout="0">
  </Item>
  <Item Name="Demo.FloodMonitor.GeneratedBusinessProcess"
        ClassName="Demo.FloodMonitor.GeneratedBusinessProcess"
        PoolSize="1" Enabled="true" Foreground="false" InactivityTimeout="0">
  </Item>
  <Item Name="Demo.FloodMonitor.BusinessOperation"
        ClassName="Demo.FloodMonitor.BusinessOperation"
        PoolSize="1" Enabled="true" Foreground="false" InactivityTimeout="0">
  </Item>
</Production>
}
}
```

以下の点に注意してください。

- プロダクションは 1 つのクラスで、具体的には、Ens.Production のサブクラスです。
- XData ProductionDefinition ブロックには、プロダクションの構成情報が記されています。

- 各 〈Item〉 はビジネス・ホストです。構成項目と呼ばれることもあります。
- ビジネス・ホストごとに 1 つずつのクラスを参照します。ClassName はこのホストの基本となるクラスを指しています。これは、プロダクションでこのビジネス・ホストのインスタンスを作成するときに、特定のクラスのインスタンスを作成する必要があることを意味します。
- ビジネス・ホスト名は任意の文字列です。この例のように、クラス名をホスト名にすると便利な場合があります。同じクラスを使用する多数のビジネス・ホストを作成する場合は、この方法ではうまくいきません。

  開発の早い段階で命名規則を設定することが重要です。"プロダクション作成のベスト・プラクティス" を参照してください。命名規則が存在しない場合は混乱を招く可能性があります。
- XData ブロック内のその他の値はすべて設定値です。一番上にある 〈ActorPoolSize〉 はプロダクションに関する設定です。ビジネス・ホスト定義内では、プール・サイズ、有効、フォアグラウンド、および非活動タイムアウトがビジネス・ホストに関する設定です。

1.3 開発ツールと開発タスク

プロダクションは、主に、クラス定義とサポート・エンティティで構成されています。プロダクションの作成プロセスでは、ニーズに応じて、少量のプログラミングが必要な場合と、大量のプログラミングが必要な場合があります。前述したように、InterSystems IRIS には、非技術系ユーザがビジネス・ロジックを視覚的に作成するためのグラフィカル・ツールが用意されています。これらのツールによって、必要なクラス定義が生成されます。

プロダクションを開発する際には、次のように、管理ポータルと選択した IDE の両方を使用します。

1.3.1 ポータル・タスク

管理ポータルでは、ウィザードを使用して、以下を定義してコンパイルします。

- プロダクション・クラス。"プロダクションの構成" を参照してください。

  プロダクションに変更を加えると、InterSystems IRIS が、自動的に、それらを保存してプロダクション・クラスをコンパイルします。
- BPL ビジネス・プロセス・クラス。"BPL プロセスの開発" を参照してください。
- DTL 変換クラス。"DTL 変換の開発" を参照してください。
- ビジネス・ルール・クラス。"ビジネス・ルールの開発" を参照してください。

以下の追加のタスクでも管理ポータルを使用します。

- 設定で使用する再利用可能項目の定義。これには、プロダクション認証情報やビジネス・パートナーなどが含まれます。"プロダクションの構成" を参照してください。
- プロダクションの開始と停止。"プロダクションの管理" を参照してください。
- テスト・プロセスとデバッグ・プロセスに含まれるメッセージ・フローの調査。"プロダクションの監視" を参照してください。
- テスト固有のビジネス・ホスト。"テストとデバッグ" を参照してください。

1.3.2 IDE タスク

IDE で以下のクラスを定義してコンパイルします。

- メッセージ・クラス。"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。"ビジネス・サービスの定義" を参照してください。InterSystems IRIS には、特定のアダプタを直接使用するビジネス・サービス・クラスがあることに注意してください。このクラスの 1 つを使用すれば、独自のクラスを作成する必要がありません。
- カスタム・ビジネス・プロセス・クラス。"ビジネス・プロセスの定義" を参照してください。
- ビジネス・オペレーション・クラス。"ビジネス・オペレーションの定義" を参照してください。InterSystems IRIS には、特定のアダプタを直接使用するビジネス・オペレーション・クラスがあることに注意してください。このクラスの 1 つを使用すれば、独自のクラスを作成する必要がありません。
- カスタム・データ変換クラス。"データ変換の定義" を参照してください。
- カスタム・アダプタ・クラス。"あまり一般的ではないタスク" を参照してください。

"あまり一般的ではないタスク" では、その他のトピックも参照してください。

1.4 利用可能な特殊なクラス

InterSystems IRIS には、特殊なアダプタ・クラスとビジネス・ホスト・クラスが多数用意されており、これにより開発時間とテスト時間を短縮することができます。最も一般的なオプションの概要は、"相互運用プロダクションの概要" の "その他のプロダクション・オプション" を参照してください。

2

ビジネス・サービス、ビジネス・プロセス、およびビジネス・オペレーションのプログラミング

このページでは、プロダクションのビジネス・サービス、ビジネス・プロセス、およびビジネス・オペレーションの開発時に一般的なプログラミング・タスクおよびトピックについて説明します。

2.1 概要

ビジネス・ホスト・クラスまたはアダプタ・クラスを作成する場合は、コールバック・メソッドを実装して、必要に応じて追加のメソッドを定義し、設定を追加または削除します。

ビジネス・ホストまたはアダプタ内では、メソッドが以下のタスクの一部または全部を実行します。

- ビジネス・ホストまたはアダプタのプロパティ値の取得または設定。
- コールバック・メソッドの定義。コールバック・メソッドは、デフォルトでは何もしない継承メソッドです。このメソッドを上書きして実装する必要があります。
- ビジネス・ホストまたはアダプタの継承ヘルパー・メソッドの実行。ヘルパー・メソッドは、他のメソッドによって使用されるメソッドの俗称です。

  ビジネス・オペレーションとビジネス・プロセス内のメソッドの多くが、プロダクション内の他のビジネス・ホストにメッセージを送信する継承メソッドを呼び出します。
- 発生したさまざまな状態に応じたログ、アラート、またはトレース通知の生成。以下の用語が使用されます。
  - ログ・エントリは、外部の物理的な問題 (ネットワーク接続の不具合など) のような項目を記録するためのものです。

    これらは自動的にイベント・ログに書き込まれます。
  - アラートは、定義してプロダクションに追加されたアラート・プロセッサ経由でユーザに警告するためのものです。

    これらも自動的にイベント・ログに書き込まれます。
  - トレース項目は、デバッグや診断の目的に使用するためのものです。これらは、プロダクションを展開する前にプログラム・エラーを特定するためなどに使用できます。イベント・ログとターミナルのどちらかまたは両方に書き込まれます。

すべてのタイプのエラーやアクティビティによってこれらの通知が生成されるわけではありません。どの事象をどのように記録するかは、開発者が決めることです。プログラム・エラーはイベント・ログに登録するべきではないことに注意してください。これらはプロダクションをリリースする前に解決すべき問題です。

2.2 基本原理

プロダクションに最適なプログラミング方法を理解しておくことが重要です。ビジネス・ホストは別々のプロセス内で動作するため、次の点を確認しておく必要があります。

- ビジネス・ホストがトランザクションを開始する場合は、同じビジネス・ホストがトランザクションを完了するか、ロールバックする必要があります。

  具体的には、ObjectScript TSTART や %COMMITMODE = EXPLICIT を設定した SQL 文などによってビジネス・ホスト・コードでトランザクションを開始する場合、そのビジネス・ホストには、そのトランザクションを完了またはロールバックするコードが必要です。トランザクションを入れ子にする場合、トランザクションのすべての部分がコミットまたはロールバックされるまで、プロセスですべてのロックが保持される点に注意が必要です。

- ビジネス・ホストがシステム・リソースを割り当てる (ロックの取得やデバイスのオープンなど) 場合は、同じビジネス・ホストがそれらのリソースを解放する必要があります。

- ビジネス・ホスト間で共有される情報はすべて、(パブリック変数を介してではなく) ビジネス・ホスト間で送信されるメッセージ内に含める必要があります。

重要　　この指針に従わないと、プロダクションが動作不能になることがあります。

ビジネス・ルールとデータ変換についても、同様の考え方が適用されます。

また、InterSystems IRIS メソッドから受け取ったエラー・コードを頻繁に処理する必要があります。InterSystems IRIS Interoperability 開発フレームワークは、エラー・コードに関してカスタム・コードを極力単純で直線的にできるように設計されています。例えば、以下のようなことです。

- ビジネス・サービスの OnProcessInput() メソッドと、ビジネス・オペレーションの OnMessage() メソッドとその他のユーザ作成 MessageMap メソッドがプロダクション・フレームワークでラップされるため、コードに新たなエラー・トラッピングを追加する必要がありません。

- カスタムのビジネス・オペレーション・コードで使用できるアダプタ・メソッドでは、SendAlert() や SendRequestSync() などのカスタム・コードに使用できるフレームワーク・メソッドのように、絶対にトラップ・アウトしないように作成されています。また、アダプタ・メソッドは、エラー状態が発生したときに [再試行] と [一時停止] の適切な値を自動的に設定します。

プロダクション・フレームワークにはこれらの予防措置が組み込まれているので、一般的な環境では、カスタム・コードは単に各呼び出しのエラー・コードをチェックするだけにとどめ、それがエラー値であったらその値で終了するようにすることをお勧めします。次に、このコーディング・スタイルの例を示します。

Class Definition

```
Class Test.FTP.FileSaveOperation Extends Ens.BusinessOperation
{
 Parameter ADAPTER = "EnsLib.File.OutboundAdapter";

 Method OnMessage(pRequest As Test.FTP.TransferRequest,
                  Output pResponse As Ens.Response) As %Status
 {
 Set pResponse=$$$NULLOREF
 Set tFilename=..Adapter.CreateTimestamp(pRequest.Filename,"%f_%Q")
 ; file with timestamp should not already exist
 $$$ASSERT('..Adapter.Exists(tFilename))
 Set tSC=..Adapter.PutStream(tFilename,pRequest.StreamIn) Quit:$$$ISERR(tSC) tSC
 Quit $$$OK
 }
}
```

SQL アダプタを使用してビジネス・オペレーションで多数の SQL 文を実行した後、そのいずれかが失敗したときに戻る前にロールバックを呼び出す場合など、複雑なシナリオの方が有用なこともあります。呼び出される API によっては、返されるステータス値に加え、パブリック変数をチェックする必要がある場合があります。例としては、埋め込み SQL の場合の SQLCODE チェックが挙げられます。しかし、単純な環境では、前の例のコーディング・スタイルが最適な慣行といえます。

2.3 参照によるまたは出力としての値の引き渡し

参照または出力による値の引き渡しに慣れていない場合は、この節を参考にしてください。

InterSystems IRIS の多くのメソッドで、少なくともステータス (%Status のインスタンス) と応答メッセージ (またはその他の戻り値) の 2 つが返されます。通常、応答メッセージは、参照によってまたは出力として返されます。値が参照によってまたは出力として返される場合は、次のような意味があります。

- メソッドを定義するときに、メソッドで対応する変数を設定する必要があります。
- メソッドを呼び出すときに、対応する引数の前に期間を含める必要があります。

下の例は、これらのポイントを示しています。

2.3.1 標準的なコールバック・メソッド

標準的なコールバック・メソッドのシグニチャを以下に示します。

```
method OnRequest(request As %Library.Persistent, Output response As %Library.Persistent) as %Status
```

キーワードの `Output` は、2 つ目の引数を出力として返す必要があることを意味します。このメソッドの実装では、以下のタスクを実行してメソッド・シグニチャを満たす必要があります。

1. `response` という名前の変数を適切な値に設定します。この変数には、メソッドが実行を完了したときに値を代入する必要があります。
2. %Status のインスタンスを参照している変数の名前が後に続く、終了コマンドで終了します。

以下に例を示します。

```
Method OnRequest(request As %Library.Persistent, Output response As %Library.Persistent) as %Status
{
   //other stuff
   set response=myObject
   set pSC=..MyMethod() ; returns a status code
   quit pSC
}
```

注釈　要求と同じ応答を設定する場合、必ず %ConstructClone() を使用します。そうしないと、応答を操作すると要求オブジェクトが変更されてしまい、ビジネス・ホストに送信されたメッセージのレコードが不正確になります。例えば、要求に対する応答を設定する場合、以下のように入力します。

```
Method OnRequest(request As %Library.Persistent, Output response As %Library.Persistent) as
%Status
{
  set response=request.%ConstructClone()
  // manipulate response without affecting the request object
}
```

この例は、値が出力として返される場合を示していますが、詳細は値が参照によって渡される場合と同じです。

2.3.2 標準的なヘルパー・メソッド

標準的な継承ヘルパー・メソッドのシグニチャを以下に示します。

```
method SendRequestSync(pTargetDispatchName As %String,
                       pRequest As %Library.Persistent,
                       ByRef pResponse As %Library.Persistent,
                       pTimeout As %Numeric = -1,
                       pDescription As %String = "") as %Status
```

キーワードの ByRef は、3 つ目の引数を参照によって返す必要があることを意味します。このメソッドを呼び出すには、以下を使用します。

```
 set sc=##class(pkg.class).SendRequestSync(target,request,.response,timeout,description).
```

3 つ目の引数の手前のピリオドに注目してください。

この例は、値が参照によって渡される場合を示していますが、詳細は値が出力として返される場合と同じです。

2.4 設定の追加と削除

プロダクション、ビジネス・ホスト、またはアダプタの新しい設定を入力するには、そのクラス定義を次のように修正します。

1. 定義する構成設定ごとにプロパティを追加します。
2. SETTINGS というクラス・パラメータをクラスに追加します。
3. SETTINGS の値として、定義したプロパティの名前のカンマ区切りリストを設定します。以下に例を示します。

```
Property foo As %String;

Property bar As %String;

Parameter SETTINGS = "foo,bar";
```

SETTINGS の詳細は、次の節を参照してください。

これで、該当クラスの項目が構成対象として選択されるたびに、[**プロダクション構成**] ページの構成画面に foo および bar の設定が自動的に表示されるようになります。

継承された構成設定を削除するには、ハイフン (-) を先頭に付加したプロパティを SETTINGS クラス・パラメータで指定します。以下に例を示します。

```
Parameter SETTINGS = "-foo";
```

2.5 設定に関するカテゴリと制御の指定

プロダクションを構成する場合は、デフォルトで、設定が **[詳細]** タブの **[追加設定]** カテゴリに表示されます。デフォルトでは、設定は、その基準となるプロパティに応じて、次のいずれかの入力メカニズムを提供します。

- 通常、設定は平文入力フィールドを伴って表示されます。
- プロパティが VALUELIST パラメータを指定している場合、設定はドロップダウン・リストを伴って表示されます。ドロップダウン・リストに表示される項目は、区切り文字としてコンマを使用して区切る必要があります。
- プロパティのタイプが **%Boolean** である場合、設定はチェックボックスとして表示されます。

いずれの場合も、プロパティの InitialExpression (指定されている場合) によって入力フィールドの初期状態が制御されます。

場所と制御タイプの両方をオーバーライドできます。デフォルト設定をオーバーライドするには、次の手順で SETTINGS に設定名を含めます。

```
Parameter SETTINGS = "somesetting:category:control";
```

または、次の手順を行います。

```
Parameter SETTINGS = "somesetting:category";

Parameter SETTINGS = "somesetting::control";
```

ここでは、category と control は、それぞれ使用する カテゴリ および コントロール を示します。次の項では、詳細を説明します。

次の手順により、複数の設定を含めることができます。

```
Parameter SETTINGS = "setting1:category:control1,setting2:control2:editor,setting3:category:control3";
```

次に、例を示します (許可されない改行が含まれています)。

```
Parameter SETTINGS = "HTTPServer:Basic,HTTPPort:Basic,SSLConfig:Connection:sslConfigSelector,
ProxyServer:Connection,ProxyPort:Connection,ProxyHTTPS:Connection,
URL:Basic,Credentials:Basic:credentialsSelector,UseCookies,ResponseTimeout:Connection"
```

2.5.1 設定のカテゴリ

category は、大文字小文字を区別する、次のリテラル値のいずれかです。

- `Info` ― 設定を **[情報設定]** カテゴリに配置します。
- `Basic` ― 設定を **[基本設定]** カテゴリに配置します。
- `Connection` ― 設定を **[接続設定]** カテゴリに配置します。
- `Additional` ― 設定を **[追加設定]** カテゴリに配置します。
- `Alerting` ― 設定を **[アラート・コントロール]** カテゴリに配置します。
- `Dev` ― 設定を **[開発とデバッグ]** カテゴリに配置します。

または、ユーザ自身のカテゴリ名を使用します。

2.5.2 設定のコントロール

control は、[プロダクション構成] ページでの設定の表示時および変更時に使用する、特定のコントロールの名前を指定します。selector を使用するか、パッケージ EnsPortal.Component 内のクラスの名前を使用します。"設定コントロールの例" を参照してください。

適切なオプションのセットを表示するためにコンポーネントに追加の値を指定するには、次のように、一連の名前-値のペアを追加します。

```
myControl?Prop1=Value1&Prop2=Value2&Prop3=Value3
```

説明：

- Prop1、Prop2、Prop3 などはコントロールのプロパティ名です。例:multiSelect
- Value1、Value2、Value3 などは対応する値です。

  JavaScript zenPage オブジェクトのプロパティを参照するには、構文 @propertyname を使用します。サポートされる変数は、次のとおりです。

  - currHostId は、現在の Ens.Config.Item の ID を示します。
  - productionId は、使用中の Ens.Config.Production の ID を示します。

  文字 @ をそのまま示すには、\@ を使用します。

  コントロールの context プロパティに値を渡すには、値を中括弧で囲みます。次の項を参照してください。

2.5.2.1 コントロールの context プロパティへの値の受け渡し

InterSystems IRIS 内の汎用セレクタ・コンポーネントを使用することもできます。そのためには、control を selector として指定してから、プロパティ/値のペアを追加します。

selector コンポーネントはほぼすべてのデータまたはオプションのリストをユーザに表示可能にするため、ユーザは必要に応じてデータを入力できます。このようにコンポーネントを構成するには、以下の構文を使用してコントロールの context プロパティを設定します。

```
context={ContextClass/ContextMethod?Arg1=Value1&Arg2=Value2&Arg3=Value3}
```

説明：

- ContextClass はコンテキスト・クラスの名前です。これは、%ZEN.Portal.ContextSearch または Ens.ContextSearch のサブクラスとして指定する必要があります。"設定コントロールの例" を参照してください。
- ContextMethod はこのクラスのメソッドです。このメソッドによって提供されるデータから、値を選択します。
- Arg1、Arg2、Arg3 などは、このメソッドの多次元引数です。
- Value1、Value2、Value3 などはこれらの引数の値です。値のルールは、前述のルールと同じです (例えば、@propertyname が使用できます)。

注釈　セレクタ・コンポーネントには、multiSelect というプロパティも設定されています。デフォルトでは、1 つのアイテムのみが選択可能です。複数のアイテムを使用可能にするには、プロパティ/値のペアを multiSelect=1 のように含めます。

引数の名前と値は URL エンコードされる必要があります。例えば、% は %25 で置換します。

2.5.2.2 設定コントロールの例

次のリストに control の例を示します。このリストは、ユーザが選択可能なデータの種類別に編成されています。

これらの例では、InterSystems IRIS クラスが使用されています。これらの例の多くは Ens.ContextSearch クラスを使用していますが、これは有用なメソッドを多数提供します。このリストに必要なシナリオが含まれていない場合は、Ens.ContextSearch のクラス・ドキュメントを参照して、既存のメソッドによって必要なデータが提供されるかどうかを判別してください。必要なシナリオがそのクラスで対応していない場合は、専用の Ens.ContextSearch のサブクラスを作成できます。

BPL

```
bplSelector
```

同じプロダクションのビジネス・ホスト

```
selector?multiSelect=1&context={Ens.ContextSearch/ProductionItems?targets=1&productionName=@productionId}
```

または、ユーザがビジネス・ホストを 1 つだけ選択しなければならない場合:

```
selector?context={Ens.ContextSearch/ProductionItems?targets=1&productionName=@productionId}
```

ビジネス・パートナー

```
partnerSelector
```

ビジネス・ルール

```
ruleSelector
```

文字セット

```
selector?context={Ens.ContextSearch/CharacterSets}
```

資格情報セット

```
credentialsSelector
```

ディレクトリ

```
directorySelector
```

DTL

```
dtlSelector
```

ファイル

```
fileSelector
```

構成

```
selector?context={Ens.ContextSearch/getDisplayList?host=@currHostId&prop=Framing}
```

ローカル・インタフェース

```
selector?context={Ens.ContextSearch/TCPLocalInterfaces}
```

特定のビジネス・ホストに適切なスキーマ・カテゴリ

```
selector?context={Ens.ContextSearch/SchemaCategories?host=classname}
```

ここで、classname はビジネス・ホストのクラス名です。以下に例を示します。

```
selector?context={Ens.ContextSearch/SearchTableClasses?host=EnsLib.MsgRouter.RoutingEngineST}
```

スケジュール

```
scheduleSelector
```

特定のビジネス・ホストに適切なテーブル・クラス検索

```
selector?context={Ens.ContextSearch/SearchTableClasses?host=classname}
```

ここで、classname はビジネス・ホストのクラス名です。以下に例を示します。

```
selector?context={Ens.ContextSearch/SearchTableClasses?host=EnsLib.EDI.EDIFACT.Service.Standard}
```

SSL 構成

```
sslConfigSelector
```

2.6 設定に関するデフォルト値の指定

ビジネス・ホスト・クラス (およびアダプタ・クラス) を定義するときに、これらの項目の設定に関するデフォルト値の制御方法を検討する必要があります。InterSystems IRIS は、次の 3 つのソースのいずれかから設定のデフォルト値を取得できます。

- プロダクション定義。
- 現在の InterSystems IRIS インスタンス用に定義されているが、プロダクションの外部に格納されている値。詳細は、"プロダクション・デフォルトの定義" を参照してください。
- ホスト・クラス内で定義されたプロパティのデフォルト値。この場合は、デフォルト値が InitialExpression プロパティ・キーワードによって決定されます。

設定の中には、TCP/IP アドレスやファイル・パスなどの環境に依存しているものがあります。これらの設定は、プロダクションの外部にソースを持つように構成することが普通です。一方、これ以外の ReplyCodeActions などの設定は設計段階で決定しておくものなので、多くの場合はプロダクション定義からその設定値を取得するようにアプリケーションを開発します。

さまざまなソースにある構成設定を使用するプロダクションを開発できます。その主な目的は、複数の InterSystems IRIS インスタンス間でプロダクションを容易に移動できるようにすることです。例えば、テスト環境からライブ環境への移動が考えられます。

プロダクションを定義したら、管理ポータルの [**プロダクション構成**] ページでプロダクション設定とビジネス・ホスト設定の両方のソースを変更できます。詳細は、"プロダクションの構成" を参照してください。

デフォルト設定を使用すると、プロダクション定義の外部でプロダクション設定およびビジネス・ホスト設定を定義でき、プロダクションを更新するときにはその場所に両方の設定を保持しておくことができます。プロダクションのアップグレードやシステム間でのプロダクションの移動を円滑に進めるために、設定処理を省略し、システムにインストールした構造から設定値を取得できます。プロダクション定義に設定が見つからない場合、プロダクション定義の外部にデフォルト設定があれば、InterSystems IRIS はそこから値を取得します。

プログラミングの詳細は、"クラス参照" の Ens.Director のクラス・エントリにある以下のメソッドの説明を参照してください。

- GetProductionSettingValue()
- GetProductionSettings()

2.7 ビジネス・ホストからのプロパティとメソッドへのアクセス

ビジネス・ホスト・クラス内でメソッドを定義する場合は、そのクラスまたは関連アダプタのプロパティまたはメソッドにアクセスする必要があります。この節では、その手順について簡単に説明します。

ビジネス・ホストのインスタンス・メソッド内では、以下の構文を使用できます。

- `..bushostproperty`

  ビジネス・ホストの設定またはその他のプロパティにアクセスします (すべての設定がそれぞれのクラスのプロパティであることを思い出してください)。

- `..bushostmethod()`

  ビジネス・ホストのインスタンス・メソッドにアクセスします。

- `..Adapter.adapterproperty`

  アダプタの設定またはその他のプロパティにアクセスします (すべてのビジネス・ホストにプロパティの Adapter があることに注意してください。このプロパティを使用してアダプタにアクセスしてから、ドット構文を使用してアダプタのプロパティにアクセスします)。

- `..Adapter.adaptermethod()`

  アダプタのインスタンス・メソッドにアクセスし、引数をそのメソッドに渡します。例えば、ビジネス・オペレーションから送信アダプタの `PutStream` メソッドを呼び出すには、以下のように入力します。

  ```
  ..Adapter.PutStream(pFilename,..%TempStream)
  ```

2.8 プロダクション設定へのアクセス

プロダクションの設定にアクセスしなければならない場合があります。そのためには、マクロの `$$$ConfigProdSetting` を使用します。例えば、`$$$ConfigProdSetting("mySetting")` は、`mySetting` という名前のプロダクション設定の値を取得します。安全のため、このマクロは $GET 呼び出しにラップすることをお勧めします。以下に例を示します。

```
set myvalue=$GET($$$ConfigProdSetting("mySetting"))
```

$GET の詳細は、"ObjectScript リファレンス" を参照してください。

"Ens.Director の使用による設定へのアクセス" も参照してください。

2.9 メッセージの送信方法の選択

ビジネス・オペレーションとビジネス・プロセス内のメソッドの多くが、プロダクション内の他のビジネス・ホストにメッセージを送信する継承メソッドを呼び出します。ここでは、そのオプションについて説明します。

2.9.1 同期送信と非同期送信

ビジネス・サービス・クラス、ビジネス・プロセス・クラス、およびビジネス・オペレーション・クラスを定義するときに、ビジネス・ホストからの要求メッセージの送信方法を指定します。次の 2 つの主要なオプションがあります。

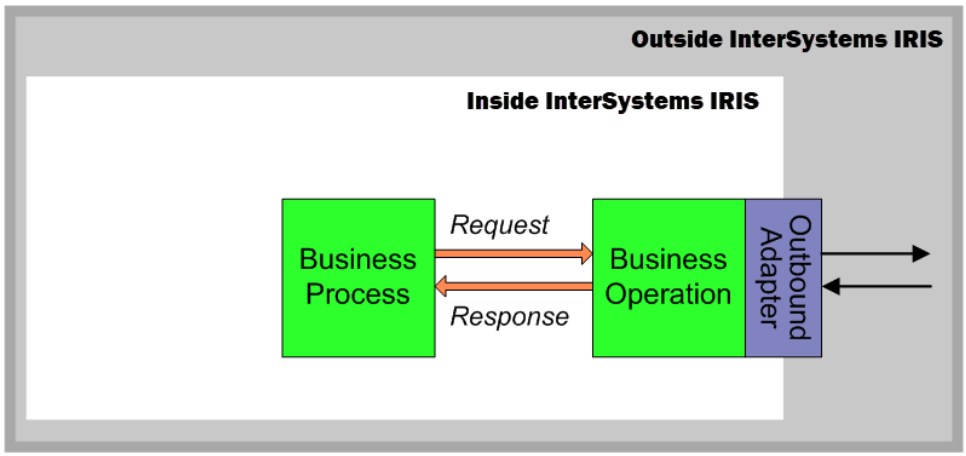
- 同期 ― 呼び出し側は、すべての処理を停止して応答を待機する必要があります。
- 非同期 ― 呼び出し側は待機しません。要求を送信した直後に、別の処理を再開します。要求を非同期で送信する場合、呼び出し側は、この要求に対する応答に関して以下の 2 つのオプションのうち 1 つを指定します。
  - 応答が到着したときに、それを受信することを求める。
  - 応答の可能性を無視する。

メッセージの送信方法の選択は、メッセージ自体に記録されず、メッセージ定義の一部でもありません。これはメッセージを送信するビジネス・ホスト・クラスによって決定されます。

2.9.2 遅延送信

同期（待つ）と非同期（待たない）という単純な選択肢のほかに、遅延応答と呼ばれているメカニズムを使用して InterSystems IRIS の外部にメッセージを送信できます。

例えば、あるビジネス・プロセスで、InterSystems IRIS の外部でアクションを起動する必要があるとします。このビジネス・プロセスはビジネス・オペレーションに要求を送信し、そのビジネス・オペレーションは呼び出しを実行して、受け取った応答をビジネス・プロセスに返します。このビジネス・プロセスはすべての応答が目的とする受信者です。ビジネス・オペレーションは、要求を送り出し、応答を受け取るための手段にすぎません。ビジネス・プロセスが同期的に要求を発行している場合でも、また非同期応答を要求して非同期的に要求を発行している場合でも、このビジネス・オペレーションにより応答が中継されて戻されます。下の図は、このメカニズムをまとめたものです。

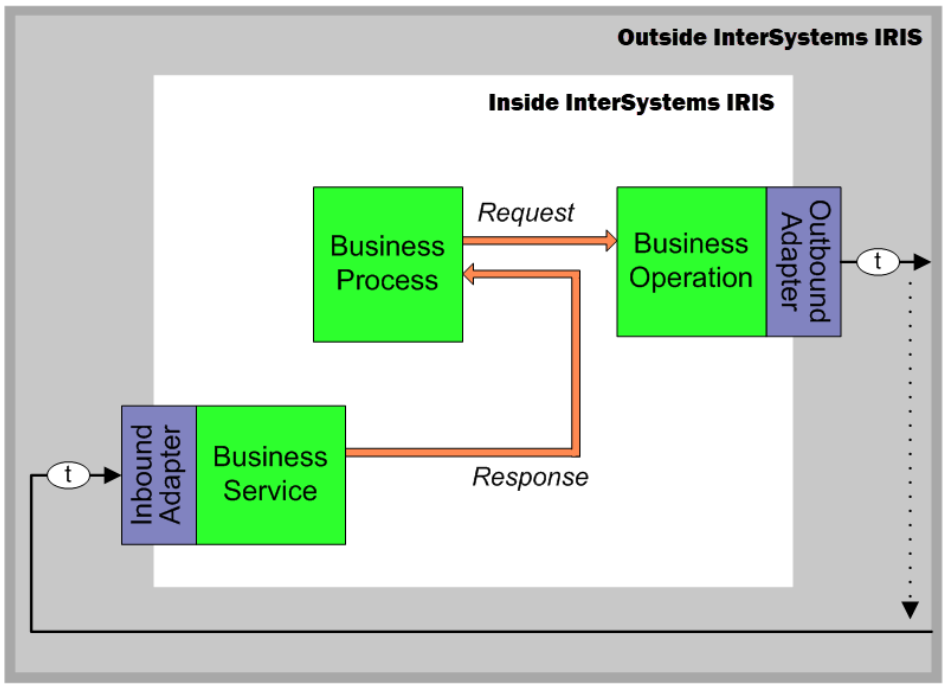


あるビジネス・プロセスから要求を受信するビジネス・オペレーションが、延期された応答機能を使用するように作成されているとします。元の送信者は、ビジネス・オペレーションによって応答が延期することを知りません。応答の延期は、ビジネス・オペレーションの開発者の判断で設計に取り入れます。実際に、ビジネス・オペレーションが応答を延期した場合、元の送信者は、延期期間の最後に応答を受信しても、応答が延期されていたことには気付きません。

ビジネス・オペレーションで応答を延期するには、このオペレーションの DeferResponse() メソッドを呼び出し、元の送信者と元の要求を表すトークンを生成します。ビジネス・オペレーションは、このトークンが InterSystems IRIS への延期された応答に含まれるように、このトークンを外部エンティティに伝達する方法も見つける必要があります。例えば、外部の宛先が電子メールである場合は、ビジネス・オペレーションから送信する電子メールの件名にこのトークン文字列を記述します。この電子メールを受信した外部エンティティは、要求の件名からこのトークンを抽出し、応答の件名に使用します。次のダイアグラムでは、項目 t がこのトークンを表しています。

ビジネス・オペレーションが要求を延期した時点から、その応答を最終的に元の送信者が受信する時点まで、その要求メッセージは延期ステータスになっています。対応する応答を元の送信者が受信すると、要求メッセージのステータスは遅延から完了に変わります。

プロダクション内のどのビジネス・ホストでも、要求に対する応答の受信イベントを判別して、元の送信者に送り返すことができます。InterSystems IRIS プロダクションのどこにイベントが到着するかは、プロダクションの設計によって異なります。通常、InterSystems IRIS 外部からの受信イベントを受け取るのはビジネス・サービスです。受信イベントを受け取るビジネス・ホストは、このイベントと共に、延期された応答のトークンを受信する必要があります。その後、このビジネス・ホストは SendDeferredResponse() メソッドを呼び出して、受信イベント・データから適切な応答メッセージを作成し、元の送信者に返します。元の送信者はこの応答を受信しますが、これがどのように返されたかについてはまったくわかりません。下の図は、要求とその遅延応答を示しています。



2.10 イベント・ログ・エントリの生成

イベント・ログは、特定のネームスペース内で実行中のプロダクションで発生したイベントが記録されたテーブルです。管理ポータルには、このログを表示するページがあります。このページは、主にシステム管理者向けですが、開発でも役に立ちます (このページの詳細は、"プロダクションの監視" を参照してください)。

イベント・ログの主な目的は、プロダクションの実行中に問題が発生した場合にシステム管理者に役立つ診断情報を提供することです。

InterSystems IRIS ではイベント・ログ・エントリが自動的に生成されますが、独自のエントリを追加することもできます。イベントは、アサート、情報、警告、エラー、およびステータスのいずれかです (イベント・ログには、次の節で説明するアラート・メッセージとトレース項目を含めることもできます)。

イベント・ログ・エントリを生成するには:

1. ログに記録するイベントを特定します。

   すべてのタイプのエラーまたはアクティビティによってイベント・ログ・エントリが生成されるわけではありません。注目すべき事象、使用するタイプ、および記録する情報を選択する必要があります。例えば、イベント・ログ・エントリは、ネットワークの接続不良などの、外部の物理的な問題が発生したときに生成する必要があります。

イベント・ログには、プログラム・エラーを登録すべきではありません。それらはプロダクションをリリースする前に解決すべき問題です。

2. 以降の項の説明に従って、ObjectScript でイベント・ログ・エントリを生成するようにプロダクションの該当部分 (通常はビジネス・ホスト・クラス) を修正します。

重要　特定の状態やイベントをユーザに積極的に通知する必要がある場合はアラートを使用します。アラートについては、次の節と "アラート・プロセッサの定義" を参照してください。

2.10.1 ObjectScript でのイベント・ログ・エントリの生成

プロダクションによって使用されるビジネス・ホスト・クラスまたはその他のコード内の ObjectScript でイベント・ログ・エントリを生成できます。そのためには、以下のマクロのいずれかを使用します。これらのマクロは、InterSystems IRIS システム・クラスに自動的にインクルードされる Ensemble.inc インクルード・ファイル内で定義されています。

マクロ	詳細
$$$LOGINFO(message)	[情報] タイプのエントリを書き込みます。このテーブルで登場するすべての message は、評価結果として文字列が得られる ObjectScript 式または文字列リテラルです。
$$$LOGERROR(message)	[エラー] タイプのエントリを書き込みます。
$$$LOGWARNING(message)	[警告] タイプのエントリを書き込みます。
$$$LOGSTATUS(status_code)	%Status のインスタンスにする必要のある特定の status_code の値に応じて、エラー・タイプまたは情報タイプのエントリを書き込みます。
$$$ASSERT(condition)	引数が偽の場合に、アサート・タイプのエントリを書き込みます。condition は真か偽かを評価する ObjectScript 式です。
$$$LOGASSERT(condition)	引数の値に関係なく、アサート・タイプのエントリを書き込みます。condition は真か偽かを評価する ObjectScript 式です。

以下では、静的テキストをクラス・プロパティの値と組み合わせた式を使用した例を示しています。

ObjectScript

```
$$$LOGERROR("Awaiting connect on port "_..Port_" with timeout "_..CallInterval)
```

以下の例では、ObjectScript 関数を使用しています。

ObjectScript

```
$$$LOGINFO("Got data chunk, size="_$length(data)_"/"_tChunkSize)
```

2.11 アラートの生成

アラートは、プロダクションの実行中にアラート・イベントが発生した場合に該当するユーザに通知を送信します。その目的は、システム管理者またはサービス技術者に問題の存在を警告することです。アラートは電子メール、携帯電話、またはその他のメカニズムにより送信されます。すべてのアラートが、アラート・タイプの InterSystems IRIS イベント・ログにもメッセージを書き込みます。

プロダクションのアラート・メカニズムは、次のように動作します。

- プロダクション用のビジネス・ホスト・クラスを作成するときに、次のようなコードを追加します。
  1. ユーザが解決しなければならない望ましくない状態やその他の環境を検出する。
  2. このような状況に対してアラートを生成する。
- ビジネス・ホストである Ens.Alert という名前のアラート・プロセッサを定義して構成します。このアラート・プロセッサは、イベントを解決するプロセスを追跡するアラートを必要に応じて管理できます。アラート・プロセッサの定義方法の詳細は、"アラート・プロセッサの定義" を参照してください。どのプロダクションにも、1 つのアラート・プロセッサしか組み込むことはできません。

ビジネス・ホスト・クラス (BPL プロセス・クラス以外) で、以下を実行してアラートを生成します。

1. Ens.AlertRequest のインスタンスを作成します。
2. このインスタンスの AlertText プロパティを設定します。このプロパティの値には、技術者が問題に適切に対処できるための十分な情報を提供する文字列を指定してください。
3. ビジネス・ホスト・クラスの SendAlert() メソッドを呼び出します。このメソッドは非同期的に実行されるため、ビジネス・ホストの通常の処理を遅延させません。

注釈　BPL 内でのアラートの生成方法は、"BPL プロセスの開発" を参照してください。

2.12 トレース要素の追加

トレースは、主に開発時に使用するツールです。トレース要素を追加することで、デバッグや診断のために、プロダクション内のさまざまな要素の動作を確認できるようになります。プロダクションにトレース要素を追加するには、実行時情報を表示するコード内の領域 (通常はビジネス・ホスト・クラス) を特定します。これらの領域で、トレース・メッセージを (状況に応じて) 書き込むコード行を追加します。トレース・メッセージは、大まかな意味でのメッセージにすぎません。すなわち、トレース・メッセージは単なる文字列であり、Ens.Message やそのサブクラスとは無関係です。

ほとんどの場合は、ユーザ要素とシステム要素という 2 種類のトレース要素を定義できます。ほとんどの場合は、ユーザ・トレース要素を定義する方が適切です。

注釈　BPL、DTL、またはビジネス・ルール内でのトレース要素の書き込みに関する情報は、"BPL プロセスの開発"、"DTL 変換の開発"、および "ビジネス・ルールの開発" を参照してください。

また、トレースの有効化については、"プロダクションの監視" の "トレースの有効化" を参照してください。

2.12.1 ObjectScript でのトレース・メッセージの書き込み

ObjectScript を使用してトレース・メッセージを書き込むには、以下のコード行を使用します。

- ユーザ・トレース・メッセージを書き込むには、以下のように記述します。

  ObjectScript

  ```
  $$$TRACE(trace_message)
  ```

  ここで trace_message は、このコード行を追加するコンテキストに関する有用な情報が含まれた文字列です。
- システム・トレース・メッセージを書き込むには (あまり一般的ではありません)、以下のように記述します。

ObjectScript

```
$$$sysTRACE(trace_message)
```

InterSystems IRIS システム・コードに $$$sysTRACE が含まれていることがありますが、独自のビジネス・ホスト・クラスでは一般的には $$$TRACE を使用するのが適切です。

例:

```
$$$TRACE("received application for "_request.CustomerName)
```

2.12.2 BPL または DTL でのトレース・メッセージの書き込み

BPL ビジネス・プロセスまたは DTL データ変換を使用してユーザ・トレース・メッセージを書き込むには、<trace> 要素を使用します。"ビジネス・プロセス言語およびデータ変換言語リファレンス" または "データ変換言語リファレンス" を参照してください。

3
メッセージの定義

このページでは、プロダクション・メッセージ・ボディを定義するクラスの定義方法について説明します。

3.1 概要

メッセージ・ボディは、任意の永続オブジェクトにすることができます。

実際には、Ens.Util.RequestBodyMethods または Ens.Util.ResponseBodyMethods のサブクラスを作成してプロパティを追加する方法が一般的です。これにより、標準のメッセージ・ボディが作成されます。これらのクラスを使用すれば、管理ポータルでメッセージの内容を表示するためのさまざまな組み込み機能に簡単にアクセスできます。これらの機能により、開発者や管理者がプロダクション実行時のエラーを見つけるのに役立ちます。特に、プロダクションでメッセージの内容を使用して送信先を決定する場合には有効です。

X12 などの Electronic Data Interchange (EDI) 形式の一部の電子ドキュメントには、長さがまちまちで、複雑なデータが含まれています。この場合は、代替クラスの InterSystems IRIS® 仮想ドキュメントを表すクラスを使用する方が適しています。またこの場合、メッセージの内容を含む一連のプロパティがメッセージ・ボディに含まれていません。詳細は、"プロダクション内での仮想ドキュメントの使用法" を参照してください。

このドキュメントのほとんどの例では、標準のメッセージ本文と比較的少数のメッセージ・プロパティが想定されています。

3.2 単純なメッセージ・ボディ・クラスの作成

メッセージ・クラス (メッセージ・ボディとして使用する) を作成するには、次のようなクラスを作成します。

- Ens.Util.RequestBodyMethods と Ens.Util.ResponseBodyMethods のどちらかを拡張する。
- 必要に応じて、メッセージ内で転送すべきデータの要素を表すプロパティを含む。

簡単な例を以下に示します。

Class Definition

```
Class Demo.Loan.Msg.CreditRatingResponse Extends Ens.Util.ResponseBodyMethods
{

Property TaxID As %String;

Property CreditRating As %Integer;

}
```

クラスにはメソッドを含めることもできます。以下に例を示します。

Class Definition

```
Class Demo.Loan.Msg.Application Extends Ens.Util.RequestBodyMethods
{

Property Amount As %Integer;
Property Name As %String;
Property TaxID As %String;
Property Nationality As %String;
Property BusinessOperationType As %String;
Property Destination As %String;

Method RecordNumber() As %String
{
  If ..%Id()="" Do ..%Save()
  Quit ..%Id()
}

Method GetRecordNumberText(pFormatAsHTML As %Boolean = 0) As %String
{
  Set tCRLF=$s(pFormatAsHTML:"<br>",1:$c(13,10))
  Set tText=""
  Set tText=tText_"Your loan application has been received,"_tCRLF
  Set tText=tText_"and is being processed."_tCRLF
  Set tText=tText_"Your record number is "_..RecordNumber()_"."_tCRLF
  Set tText=tText_"You'll receive a reply from FindRate"_tCRLF
  Set tText=tText_"within 2 business days."_tCRLF
  Set tText=tText_"Thank you for applying with FindRate."_tCRLF
  Quit tText
}

}
```

カスタムのメッセージ・クラスを作成する方法もありますが、その場合は、メッセージの検索を迅速にするために、そのクラス専用のテーブルにのみメッセージが格納されるようにクラスを定義することが重要です。そのためには、%Persistent をプライマリ・スーパークラスとして使用し、メッセージが要求であれば Ens.Request、応答であれば Ens.Response をそれに続けて記述します。例えば以下のようにします。

Class Definition

```
Class Demo.Loan.Msg.CreditRatingResponse Extends (%Persistent, Ens.Response)
{

Property TaxID As %String;

Property CreditRating As %Integer;

}
```

(クラス・パラメータの USEEXTENTSET と DEFAULTGLOBAL を使用して、クラス専用のテーブルにメッセージが格納されるようにすることもできます。詳細は、%Persistent のクラス・リファレンスを参照してください。)

重要　Ens.Request または Ens.Response をプライマリ・スーパークラスにした場合、このメッセージの保存先は、他のすべてのリクエストとレスポンスも格納されるテーブルになります。これにより、メッセージ・テーブルを問い合わせる際のパフォーマンスが低下することがあります。

永続クラスに基づくクラスを作成する方法もあり、必要に応じてそのクラスに別のスーパークラスとして %XML.Adaptor を設定できます (%XML.Adaptor を使用すると、管理ポータルにメッセージを XML 形式で表示できます)。

3.3 複雑なメッセージ・ボディ・クラスの作成

前の例では、メッセージ・ボディ・クラスに単純なプロパティしか含まれていませんでした。他のクラスを使用するプロパティを定義しなければならない場合があります。その場合は、メッセージ・ボディをパージするときにすべきことを慎重に検討する必要があります ("プロダクションの管理" を参照してください)。

メッセージ・ボディをパージすると、InterSystems IRIS では特定のメッセージ・ボディ・オブジェクトのみが削除されます。例えば、以下のメッセージ・クラスがあるとします。

```
Class MyApp.Messages.Person Extends Ens.Util.RequestBodyMethods
{
Property Name As %String;
Property MRN As %String;
Property BirthDate As %Date;
Property Address As MyApp.Messages.Address;
}
```

Address クラスは次のとおりです。

```
Class MyApp.Messages.Address Extends %Persistent
{
Property StreetAddress As %String;
Property City As %String;
Property State As %String;
Property ZIP As %String;
}
```

この場合は、メッセージ・ボディをパージすると、InterSystems IRIS によって MyApp.Messages.Person のインスタンスが削除されますが、MyApp.Messages.Address のインスタンスは削除されません。

メッセージ・ボディ・クラスで他のクラスがプロパティとして使用されている場合とアプリケーションで任意の参照オブジェクトもパージしなければならない場合は、以下のアプローチのいずれかを使用します。

- 参照クラスがシリアルであることを確認します。例えば、次のように、Address クラスを再定義します。

  ```
  Class MyApp.Messages.Address Extends %SerialObject
  {
  ...
  }
  ```

  この場合は、Address クラスのデータが Person クラスの一部として保存されます (そのため、自動的に同時にパージされます)。
- 適切なリレーションシップとしてプロパティを定義します。"リレーションシップ" を参照してください。
- メッセージ・クラスに削除トリガまたは %OnDelete() メソッドを追加して、参照クラス内の該当するレコードを削除できるようにします。
- オプションで、%XML.Adaptor をスーパークラスとして含めることによって、参照クラス内で定義されたプロパティが管理ポータルに表示されるようにします。

3.4 メッセージのパージ動作の設定

メッセージ・ボディ・クラスを定義する際には、ENSPURGE パラメータを含めて、パージ処理時にそのクラスのインスタンスを InterSystems IRIS でどのように処理するかを指定できます。このパラメータに指定できる値は以下の 2 つです。

- 0 — InterSystems IRIS は、メッセージ・ボディをパージするオプションが有効になっている場合でも、クラスに基づいてメッセージ・ボディをパージすることはありません。

- 1 – InterSystems IRIS は、メッセージ・ボディをパージするオプションが有効になっている場合、クラスに基づいてメッセージ・ボディをパージします。

ENSPURGE パラメータは、エンタープライズ・メッセージ・バンクでのパージ処理を除いて、管理ポータルからのすべてのパージ処理に影響を与えます。同様に、このパラメータは、Ens.MessageHeader クラスの Purge() メソッドを使用したプログラムによるパージにも影響を与えます。

例えば、以下の Sample.Person 永続データベース・クラスを見てみましょう。

```
Class Sample.Person Extends (%Persistent, %Populate, %XML.Adaptor)
{
Property Name As %String(POPSPEC = "Name()") [ Required ];
Property SSN As %String(PATTERN = "3N1""-""2N1""-""4N") [ Required ];
Property DOB As %Date(POPSPEC = "Date()");
...
}
```

患者情報を更新するビジネス・オペレーションに Sample.Person オブジェクトを送信するようプロダクションを構成する場合、そのオブジェクトの保持が重要になると考えられます。システムで Sample.Person メッセージ・ボディ・クラスのいずれのインスタンスもパージされないようにするために、以下のように、クラス定義に ENSPURGE パラメータを追加できます。

```
Class Sample.Person Extends (%Persistent, %Populate, %XML.Adaptor)
{
Parameter ENSPURGE As %Boolean = 0;
Property Name As %String(POPSPEC = "Name()") [ Required ];
Property SSN As %String(PATTERN = "3N1""-""2N1""-""4N") [ Required ];
Property DOB As %Date(POPSPEC = "Date()");
...
}
```

それ以降のパージでは、Sample.Person メッセージ・ボディ・クラスに基づいて、メッセージのヘッダのみが削除されるようになります。メッセージ・ボディは実質的に孤立して、プログラムでのみ削除可能となります。詳細は、"プロダクション・データのパージ" を参照してください。

ENSPURGE パラメータは、継承可能で、必須ではありません。既定値は 1 です。

4

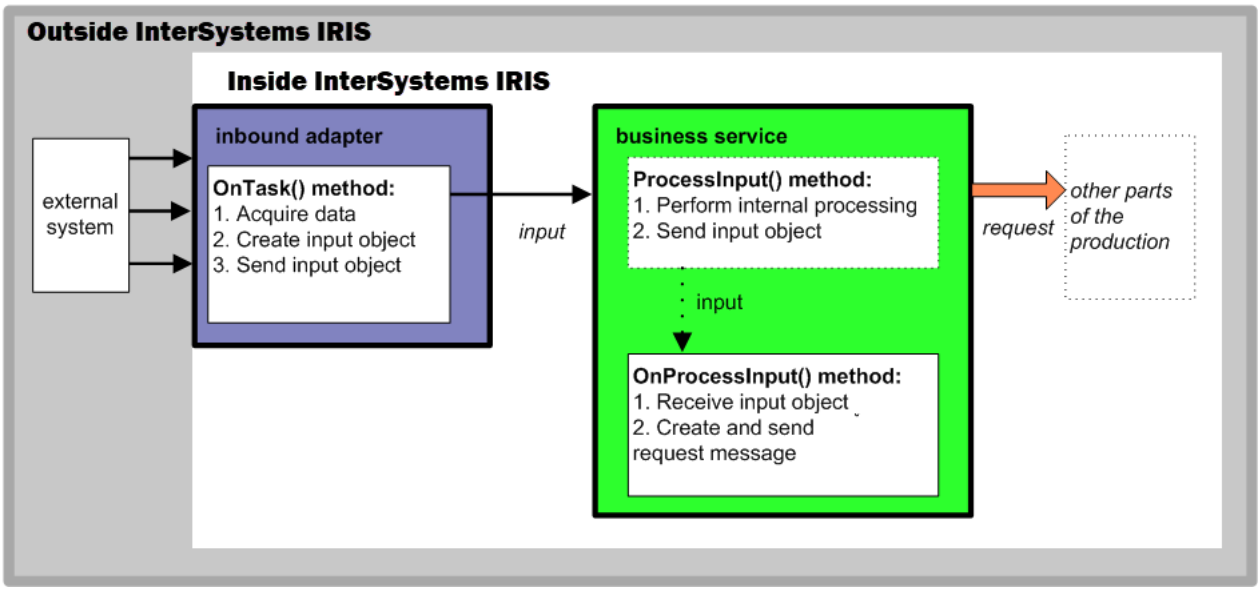
ビジネス・サービスの定義

このページでは、ビジネス・サービス・クラスの定義方法について説明します。

Tipヒント InterSystems IRIS® では、特定の受信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 つが提供されます。そのため、プログラミングの必要がありません。その一部は、"相互運用プロダクションの概要" の "接続オプション" を参照してください。

4.1 概要

ビジネス・サービスは、外部アプリケーションからの要求を受信します。下の図は、そのしくみを示しています。



この図に示されているのは、データの入力フローだけで、オプションの応答は含まれていないことに注意してください。

ビジネス・サービスは、以下のアクティビティを実行します。

- 特定の外部イベント (アプリケーションからの通知や TCP メッセージの受信など) を待ち受けます。
- イベントなどに付随するデータの読み取り、構文解釈、および検証を行います。
- 必要に応じて、イベントを受け取ったことを示す確認応答を外部アプリケーションに返します。

- 要求メッセージのインスタンスを作成して、それを処理するために適切なビジネス・プロセスまたはビジネス・オペレーションに転送します。

ビジネス・サービスの主な目的は、データ入力を受け取ることです。大半のケースでは、ビジネス・サービスにはビジネス・サービスと関連付けられた受信アダプタが存在します。ただし、アダプタが不要なケースもあります。アプリケーションが要求メッセージをサービス内に送り込める場合や、ビジネス・サービスが特定の種類の外部呼び出し（例えば複合アプリケーションからの呼び出し）を処理するように既に記述されている場合がこれに該当します。この種のビジネス・サービスは、アダプタ不要型 のビジネス・サービスと呼ばれます。

ビジネス・サービスが受信アダプタを備えている場合は、押し込みモードとは逆のデータ引き出しモードとなります。このモードでは、ビジネス・サービスはアダプタに対し、データを持っているかどうかを一定間隔で問い合わせます。一方、アダプタは、入力データを検出すると、それを処理するビジネス・サービスを呼び出します。

ビジネス・サービスがアダプタを備えていない場合は、データが引き出されません。代わりに、クライアント・アプリケーションがこのビジネス・サービスを呼び出して入力の処理を指示します（こちらはデータ押し込みモードです）。

4.2 基本原理

初めに、"InterSystems IRIS のプログラミング" を参照してください。

ビジネス・サービス内では、ビジネス・サービスの Adapter プロパティとして使用可能な関連アダプタのプロパティとメソッドにアクセスできます。これは、アダプタのデフォルト動作を変更できることを意味します。それが適切な場合と適切でない場合があります。カプセル化の原理を思い出せば役に立ちます。カプセル化のアイデアとは、アダプタ・クラスに技術固有のロジックを担当させ、ビジネス・サービス・クラスにプロダクション固有のロジックを担当させるというものです。

ビジネス・サービス・クラス内部からアダプタ・クラスの動作を大幅にまたは頻繁に変更する必要がある場合は、アダプタ・クラスのサブクラスをカスタマイズする方がより適切です。"あまり一般的ではないタスク" を参照してください。

この原理はビジネス・オペレーションにも適用されます。

4.3 ビジネス・サービス・クラスの定義

ビジネス・サービス・クラスを作成するには、次のようにクラスを定義します。

- クラスは、Ens.BusinessService（またはサブクラス）を拡張する必要があります。
- クラス内では、ADAPTER パラメータを、使用するビジネス・サービス用のアダプタ・クラスの名前と一致させる必要があります。

  Tipヒント InterSystems IRIS の外部のイベントを気にせず、定期的にビジネス・サービスを呼び出して実行させるだけの場合は、アダプタ・クラスの Ens.InboundAdapter を使用します。

- クラスには OnProcessInput() メソッドを実装する必要があります。これについては、"OnProcessInput() メソッドの実装" で説明します。
- クラスでは設定を追加または削除できます。"設定の追加と削除" を参照してください。
- クラスは任意のまたはすべてのスタートアップ・メソッドおよびティアダウン・メソッドを実装できます。"開始動作と停止動作の上書き" を参照してください。
- クラスにはその内部で作業を完了するためのメソッドを含めることができます。

ビジネス・サービス・クラスの例は、"アダプタ・ガイド" を参照してください。

4.4 OnProcessInput() メソッドの実装

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

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

ここで、pInput はアダプタがこのビジネス・サービスに送信する入力オブジェクトで、pOutput は出力オブジェクトです。

最初に、選択したアダプタ・クラスを確認してください。アダプタで必要な特定の入力引数を使用するように OnProcessInput() メソッド・シグニチャを編集することをお勧めします。

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. オプションで、ビジネス・サービス・クラスのプロパティを設定します (適時)。最も重要なビジネス・サービスのプロパティは、**%WaitForNextCallInterval** です。その値によって、InterSystems IRIS がアダプタの OnTask() メソッドを呼び出す頻度が制御されます。

   その他のプロパティについては、**Ens.BusinessService** に関する "クラス・リファレンス" を参照してください。
2. 必要に応じて、入力オブジェクトを検証します。
3. 入力オブジェクトを検査してその使用方法を決定します。
4. ビジネス・サービスから送信される要求メッセージ・クラスのインスタンスを作成します。

   メッセージの作成方法は、"メッセージの作成" を参照してください。
5. 要求メッセージの場合は、必要に応じて、入力オブジェクト内の値を使用してそのプロパティを設定します。
6. 要求メッセージの送信先を決定します。メッセージを送信するときに、プロダクション内のビジネス・ホストの構成名を使用する必要があります。
7. 要求メッセージをプロダクション内の宛先 (ビジネス・プロセスまたはビジネス・オペレーション) に送信します。次の節を参照してください。
8. 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。
9. 適切なステータスを返します。この手順は必須です。

4.5 要求メッセージの送信

ビジネス・サービス・クラスでは、OnProcessInput() の実装が要求メッセージをプロダクション内のいずれかの宛先に送信する必要があります。そのためには、必要に応じて、以下のビジネス・サービス・クラスのインスタンス・メソッドのいずれかを呼び出します。

- SendRequestSync() は、同期的にメッセージを送信します (応答を待ちます)。詳細は、"SendRequestSync() メソッドの使用法" を参照してください。
- SendRequestAsync() は、非同期的にメッセージを送信します (応答を待ちません)。詳細は、"SendRequestAsync() メソッドの使用法" を参照してください。
- SendDeferredResponse() は、以前に延期された応答を送信します。このメソッドは通常はあまり使用されません。詳細は、"SendDeferredResponse() メソッドの使用法" を参照してください。

これらのメソッドはそれぞれ、ステータス、つまり、**%Status** のインスタンスを返します。

また、これらのメソッドは、Ens.BusinessProcess と Ens.BusinessOperation 内で同じメソッド・シグニチャを使用して定義されますが、そのインターナルはクラス内とは異なります。これは、これらのインスタンス・メソッドをビジネス・プロセス・クラスとビジネス・オペレーション・クラスの内部から呼び出せることを意味します。

4.5.1 SendRequestSync() メソッドの使用法

同期の要求を送信するには、以下のように SendRequestSync() メソッドを使用します。

```
Set tSC = ..SendRequestSync(pTargetDispatchName, pRequest, .pResponse, pTimeout)
```

説明：

- pTargetDispatchName ー 要求の送信先となるビジネス・プロセスまたはビジネス・オペレーションの構成名。
- pRequest ー 要求メッセージ。"メッセージの定義" を参照してください。
- pResponse ー (参照渡し) 応答メッセージ。このオブジェクトは、応答によって返されるデータを受信します。
- pTimeout ー (オプション) 応答を待機する秒数。デフォルトは -1 (永久に待機) です。

このメソッドは、ステータス、つまり、%Status のインスタンスを返します。

応答を期待しない場合は、SendRequestSync() の代わりに SendRequestAsync() を使用できます。

4.5.2 SendRequestAsync() メソッドの使用法

非同期の要求を送信するには、以下のように SendRequestAsync() メソッドを使用します。

```
Set tSC = ..SendRequestAsync(pTargetDispatchName, pRequest)
```

説明：

- pTargetDispatchName ー 要求の送信先となるビジネス・プロセスまたはビジネス・オペレーションの構成名。
- pRequest ー 要求メッセージ。"メッセージの定義" を参照してください。

このメソッドは、ステータス、つまり、%Status のインスタンスを返します。

4.5.3 SendDeferredResponse() メソッドの使用法

SendDeferredResponse() メソッドはすべてのビジネス・ホストでサポートされています。このメソッドは、ビジネス・ホストのプロダクション遅延応答メカニズムへの参加を許可します。ビジネス・ホストは、1 つ前の遅延要求を識別して、実際の応答メッセージを作成し、要求を発信したビジネス・ホストに送信します。"InterSystems IRIS のプログラミング" の "遅延送信の使用法" を参照してください。

このトピックでは、このメカニズムにおけるビジネス・サービスの役割について説明します。例えば、延期された応答トークンと共に受信イベントがプロダクションに到着し、その到着ポイントがビジネス・サービスであるとします。このビジネス・サービスは、SendDeferredResponse() を呼び出して応答を作成し、要求の送信元に転送します。この SendDeferredResponse() 呼び出しは、以下のようになります。

ObjectScript

```
Set sc = ..SendDeferredResponse(token, pResponseBody)
```

説明：

- token ー 延期された要求を識別するための文字列。これにより、呼び出し元は、延期された要求を元の要求と照合できます。ビジネス・サービスは、プロダクション固有のメカニズムを通じて、このトークン文字列を取得します。

例えば、外部の宛先が電子メールである場合、延期された応答の受信が必要な要求を送信するときに、送信する電子メールの件名にビジネス・オペレーションでこのトークン文字列を記述します。この電子メールを受信した外部エンティティは、要求の件名からこのトークンを抽出し、応答の件名に使用します。これによりトークンが保持されるので、この応答メールを受信したビジネス・サービスで SendDeferredResponse() を呼び出すときにこのトークンを使用できます。

- pResponseBody – 応答メッセージ。このオブジェクトは、応答によって返されるデータを受信します。"メッセージの定義" を参照してください。

このメソッドは、ステータス、つまり、%Status のインスタンスを返します。

4.6 呼び出し間隔単位のイベント処理

ビジネス・サービスで呼び出し間隔ごとに 1 つのイベントしか処理しない場合は、OnProcessInput() の実装内で %WaitForNextCallInterval プロパティを 1 (真) に設定します。

ObjectScript

```
set ..%WaitForNextCallInterval=1
```

このように設定すると、複数の入力イベントが存在しても、ビジネス・サービスで CallInterval の期間に入力イベントが 1 つしか処理されないよう制限されます。

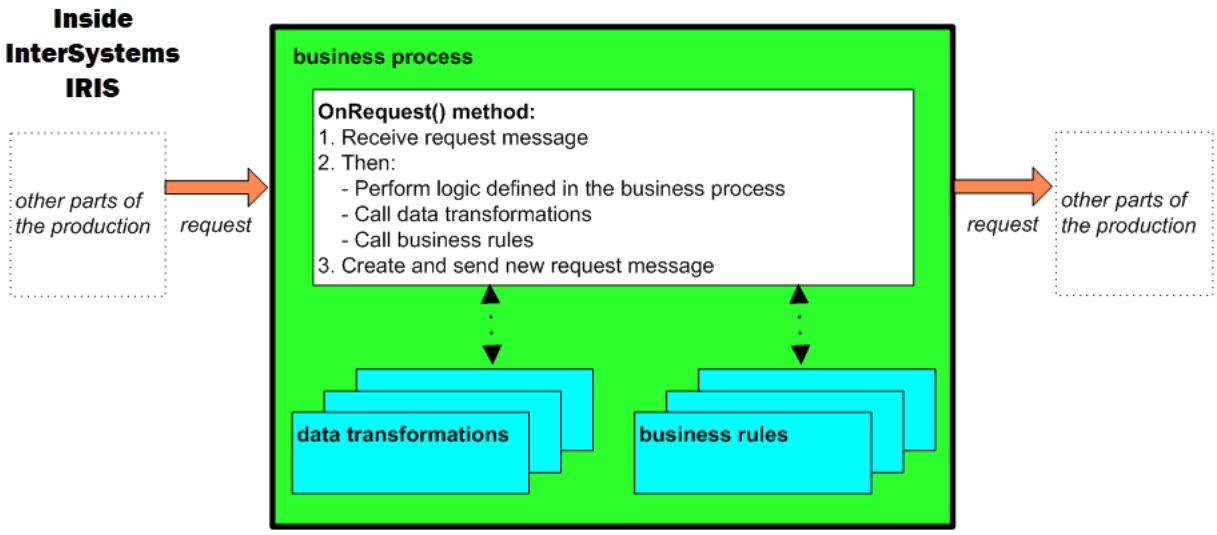
この情報は、CallInterval という名前のプロパティを持ち、そのプロパティをポーリング間隔として使用するアダプタを使用するビジネス・サービスに適用されます。

5
ビジネス・プロセスの定義

ビジネス・プロセスは、プロダクション内で高水準の処理を実行します。このページでは、ビジネス・プロセスの概要と、ビジネス・プロセス・クラスの設計方法と開発方法について説明します。

5.1 概要

慣例により、ビジネス・プロセスにはプロダクションのほとんどのロジックが含まれています。これらは、独自のロジックを含むことができ、同様にそれぞれが特殊なロジックを含むビジネス・ルールとデータ変換を呼び出すことができます。下の図は、この様子を示しています。



この図には要求メッセージしか描かれていないことに注意してください。

ビジネス・プロセスにはさまざまな用途があります。時には、1 つのビジネス・プロセスが 1 つ以上の外部アプリケーション内の一連のアクションを調整する場合があります。このプロセスは、処理を決定するロジックが組み込まれており、必要に応じて、ビジネス・オペレーションまたは他のビジネス・プロセスを呼び出します。ビジネス・プロセスにはユーザ操作を含めることもできます。詳細は、"ワークフローの開発" を参照してください。

InterSystems IRIS® には、以下の汎用タイプのビジネス・プロセスが用意されています。

- Ens.BusinessProcessBPL クラスを基本とする BPL プロセス。

  BPL プロセスのみがビジネス・プロセス実行コンテキストと、ロジックのグラフィック表示をサポートします。

- EnsLib.MsgRouter.RoutingEngine クラスまたは EnsLib.MsgRouter.VDocRoutingEngine クラスを基本とするルーティング・プロセス。

  InterSystems IRIS には、特定の種類のメッセージをルーティングする一連のクラスが用意されています。これらのサブクラスを使用するために、通常、コーディングの必要はありません。これらのクラスに関するドキュメントのリストは、"相互運用プロダクションの概要" の "ビジネス・プロセスのタイプ" を参照してください。

- Ens.BusinessProcess クラスを基本とする カスタム・ビジネス・プロセス。

プロダクションにはこれらのビジネス・プロセスを混在させることができます。

Ens.BusinessProcessBPL、EnsLib.MsgRouter.RoutingEngine、および EnsLib.MsgRouter.VDocRoutingEngine はすべて Ens.BusinessProcess を基本とすることに注意してください。

5.2 ビジネス・ロジック・ツールの比較

ビジネス・プロセスは、使用するデータ変換やビジネス・ルールと一緒に開発することになります。データ変換とビジネス・ルールの目的は、次のような特定のロジックを組み込むことです。

- データ変換でメッセージを変更する
- ビジネス・ルールで値を返したり、メッセージの送信先を指定したりする (この両方を行う場合もある)

ただし、ビジネス・プロセス、データ変換、およびビジネス・ルールで使用可能なオプションが重複しています。これらの項目の作成方法を判断しやすくするために、下の表にこれらの比較を示します。BPL (最も一般的なビジネス・プロセス)、DTL (最も一般的なデータ変換)、およびビジネス・ルールが比較されています。

オプション	BPL でのサポート	DTL でのサポート	ビジネス・ルールでのサポート
ビジネス・プロセスに関する情報の取得	あり(ビジネス実行コンテキスト変数)	なし	なし
値の割り当て	あり (<assign>)	あり (assign アクション)	あり (assign アクション)
データ変換の呼び出し	あり (<transform>)	あり (subtransform アクション)	あり (send アクション)
ビジネス・ルールの呼び出し	あり (<call>)	なし	あり (delegate アクション)
カスタム・コードの呼び出し	あり (<code>)	あり (code アクション)	なし
SQL の呼び出し	あり (<sql>)	あり (sql アクション)	なし
条件付きロジック	あり (<if>、<switch>、<branch>)	あり (if アクション)	なし
ループ処理	あり (<foreach>、<while>、<until>)	あり (for each アクション)	なし
アラートの送信	あり (<alert>)	なし	なし
トレース要素の追加	あり (<trace>)	あり (trace アクション)	あり (trace アクション)
ビジネス・オペレーションまたはプロセスへの要求メッセージの送信	あり (<call>)	なし	あり (send アクション)

オプション	BPL でのサポート	DTL でのサポート	ビジネス・ルールでのサポート
非同期要求からの応答の待機	あり (<sync>)	なし	なし
メッセージの削除	なし	なし	あり (delete アクション)
エラー処理の実行	あり (<throw>、<catch> など)	なし	なし
指定された期間または将来の特定の時刻までの実行の延期	あり (<delay>)	なし	なし
実行完了前の初期応答の送信	あり (<reply>)	なし	なし
XPATH と XSLT の使用	あり (<xpath>、<xslt>)	なし	なし
マイルストーンを認識するための一時的なメッセージの保存	あり (<milestone>)	なし	なし

DTL 変換とビジネス・ルールの詳細は、"DTL 変換の開発" と "ビジネス・ルールの開発" を参照してください。

5.3 基本原理

初めに、"InterSystems IRIS のプログラミング" を参照してください。

ビジネス・プロセスを開発する場合は、以下の基本原理を考慮してください。

- 応答オブジェクトは受信要求オブジェクトの変更バージョンにした方が望ましい場合があり、変更は段階的に行った方が有効な場合があります。ただし、受信要求オブジェクトは変更しないでください。代わりに、それをコンテキスト変数にコピーします (または、カスタム・ビジネス・プロセスの場合は、データをローカル変数にコピーします)。その後で、そのコピーを変更します。
- メッセージを同期的に送信する場合は注意が必要です (カスタム・ビジネス・プロセスまたは BPL 内の <code> でしか実行できません)。

  ビジネス・プロセス A がビジネス・プロセス B を同期的に呼び出す場合、プロセス A は応答を受け取るまで先に進めません。プロセス A がそれ自体を完了するために他のプロセス (B) への呼び出しを完了する必要がある場合とそれらのプロセスがアクター・ジョブのプールを共有している場合は、呼び出されたビジネス・プロセス (B) を処理する空きアクター・ジョブが存在しなければ、アクター・プールがデッドロックする可能性があります。

  この現象は、呼び出し元のビジネス・プロセスは呼び出し先のビジネス・プロセスが復帰するまでアクター・ジョブを完了して解放することができないが、実行する空きアクター・ジョブが存在しないことから、呼び出し先のビジネス・プロセスが実行できないために発生します。

  また、InterSystems IRIS は真の同期呼び出し中はシャットダウンできないことに注意してください。

  SendRequestAsync() を使用して、OnResponse() メソッド内の応答メッセージを処理するほうが適切です。同期的に呼び出す必要がある場合は、呼び出し先のビジネス・プロセス (B) で独自のジョブ・プールが使用されるように構成することで、この問題を回避できます。
- 単一ジョブ・ビジネス・プロセスが要求を発行して応答を待っている場合は、このプロセスから FIFO 機能が失われます。

BPL からの同期呼び出しはコンパイラによって非同期的に実装されるため、ビジネス・プロセスは呼び出し発行後にディスクに移動することになります。そうすれば、アクターは、新たなビジネス・プロセスまたは他のビジネス・プロセスへの応答をキューから除去し続けることができます。

FIFO 処理は、応答の必要な要求を呼び出さない単一ジョブ・ビジネス・プロセスにのみ許可されます。ビジネス・プロセスが呼び出しの実行後も FIFO を維持する場合は、SendRequestSync() を呼び出す <code> アクティビティを使用する必要があります。ただし、その場合は、前述の箇条書き項目が適用されます。

- BPL は、競合やデッドロックに伴う一般的な問題を回避する合理的な整然とした方法で同期メッセージングと非同期メッセージングのすべての側面を処理することにより、大きなメリットを提供します。同期を指定した呼び出しであっても、待機時間が発生すると、プロダクション・フレームワークは BPL ビジネス・プロセスのために呼び出しを実行したジョブを暗黙のうちに解放し、BPL が同期応答を待機している間にそのジョブで他の作業ができるようにします。その後、Ensemble フレームワークは暗黙のうちに同期応答の受信を調整し、BPL ビジネス・プロセスをアクティブ状態に戻して元の作業を再開させます。

  カスタム・コードを使用する場合は、デッドロックを起こしやすいプロダクションを (誤って) 設計する可能性が高くなります。必要なことは、同期要求を送信し、限られたアクター・プールを使用するビジネス・プロセスのシーケンスを作成することです。プロダクションがアクターと同じ数のメッセージを受信し、それらすべてのメッセージに対する同期送信が同時に発生した場合、すべてのアクター・プロセスが、それとは別のプロセスによってキューからメッセージが解放されるのを待機する状態となり、プロダクションでデッドロックが発生します。この問題で最も危険なことは、デッドロックの原因となる状態をテストでは必ずしも再現できない点です。導入した後で、初めてデッドロックが発生する可能性があります。この時点では、プロダクションが動かなくなる原因がわからないので、この予期できない問題では高いコストが発生することになります。

5.4 BPL ビジネス・プロセスの定義

BPL ビジネス・プロセスは Ens.BusinessProcessBPL を基本とするクラスです。この場合は、管理ポータルまたは IDE でプロセスを視覚的に作成および編集できます。詳細は、"BPL プロセスの開発" を参照してください。

5.5 カスタム・ビジネス・プロセスの定義

カスタム・ビジネス・プロセス・クラスを作成するには、次のようにクラスを定義します。

- クラスは、Ens.BusinessProcess (またはサブクラス) を拡張する必要があります。
- クラスでは、後述する 2 つの節の説明に従って、OnRequest() メソッドと OnResponse() メソッドを実装する必要があります。
- クラスでは設定を追加または削除できます。"設定の追加と削除" を参照してください。
- クラスは任意のまたはすべてのスタートアップ・メソッドおよびティアダウン・メソッドを実装できます。"開始動作と停止動作の上書き" を参照してください。
- クラスにはその内部で作業を完了するためのメソッドを含めることができます。

5.5.1 OnRequest() メソッドの実装

カスタム・ビジネス・プロセス・クラスは OnRequest() メソッドを実装する必要があります。プロダクションは、特定のビジネス・プロセスに関する最初の要求が適切なキューに到着して、実行すべきジョブが割り当てられるたびに、このメソッドを呼び出します。

このメソッドには、以下のシグニチャがあります。

```
method OnRequest(request As %Library.Persistent, Output response As %Library.Persistent) as %Status
```

説明：

- request ー 受信要求オブジェクトです。
- response ー このビジネス・プロセスによって返される応答です。

5.5.1.1 例

以下は OnRequest() メソッドの例です。

Class Member

```
Method OnRequest(request As Demo.Loan.Msg.Application, Output response As Demo.Loan.Msg.Approval)
       As %Status
{
  Set tSC=$$$OK
  $$$TRACE("received application for "_request.Name)
    #;
  If $zcrc(request.Name,2)#5=0 {
    Set tRequest = ##class(Demo.Loan.Msg.PrimeRateRequest).%New()
    Set tSC =..SendRequestAsync("Demo.Loan.WebOperations",tRequest,1,"PrimeRate")
    If $$$ISOK(tSC){
      Set tRequest = ##class(Demo.Loan.Msg.CreditRatingRequest).%New()
      Set tRequest.SSN = request.SSN
      Set tSC =..SendRequestAsync("Demo.Loan.WebOperations",tRequest,1,"CreditRating")
      If $$$ISOK(tSC){
        Set tSC = ..SetTimer("PT15S")
      }
    }
  } Else {
      Set response = ##class(Demo.Loan.Msg.Approval).%New()
      Set response.BankName = "BankUS"
      Set response.IsApproved = 0
      $$$TRACE("application is denied because of bank holiday")
    }
  Return tSC
}
```

5.5.2 OnResponse() メソッドの実装

カスタム・ビジネス・プロセス・クラスは OnResponse() メソッドを実装する必要があります。プロダクションは、特定のビジネス・プロセスに関する応答が適切なキューに到着して、実行すべきジョブが割り当てられるたびに、このメソッドを呼び出します。通常、これは、ビジネス・プロセスが生成した非同期の要求に対する応答です。

このメソッドには、以下のシグニチャがあります。

```
method OnResponse(request As %Library.Persistent,
                  ByRef response As %Library.Persistent,
                  callrequest As %Library.Persistent,
                  callresponse As %Library.Persistent,
                  pCompletionKey As %String) as %Status
```

このメソッドは、以下の引数を取ります。

- request ー このビジネス・プロセスに送信される最初の要求オブジェクトです。
- response ー このビジネス・プロセスによって最終的に返される応答オブジェクトです。
- callrequest ー 受信応答と関連付けられた要求オブジェクトです。
- callresponse ー 受信応答オブジェクトです。
- pCompletionKey ー 受信応答と関連付けられた完了を示すキー値です。この値は、要求を生成した SendRequestAsync() メソッドを呼び出すことで設定されます。

5.5.2.1 例

以下は OnResponse() メソッドの例です。

Class Member

```
/// Handle a 'Response'
Method OnResponse(request As Ens.Request,
                  ByRef response As Ens.Response,
                  callrequest As Ens.Request,
                  callresponse As Ens.Response,
                  pCompletionKey As %String) As %Status
{
    Set tSC=$$$OK
    If pCompletionKey="PrimeRate" {
        Set ..PrimeRate = callresponse.PrimeRate
    } Elseif pCompletionKey="CreditRating" {
        Set ..CreditRating = callresponse.CreditRating
    }
    Return tSC
}
```

5.5.3 OnRequest() と OnResponse() 内で使用するメソッド

OnRequest() と OnResponse() を実装するときに、Ens.BusinessProcess クラスの以下のメソッドを使用できます。

- SendDeferredResponse()
- SendRequestSync()

  ただし、"基本原理" を参照してください。
- SendRequestAsync()
- SetTimer()
- IsComponent()

これらのメソッドの詳細は、Ens.BusinessProcess に関する "クラス・リファレンス" を参照してください。

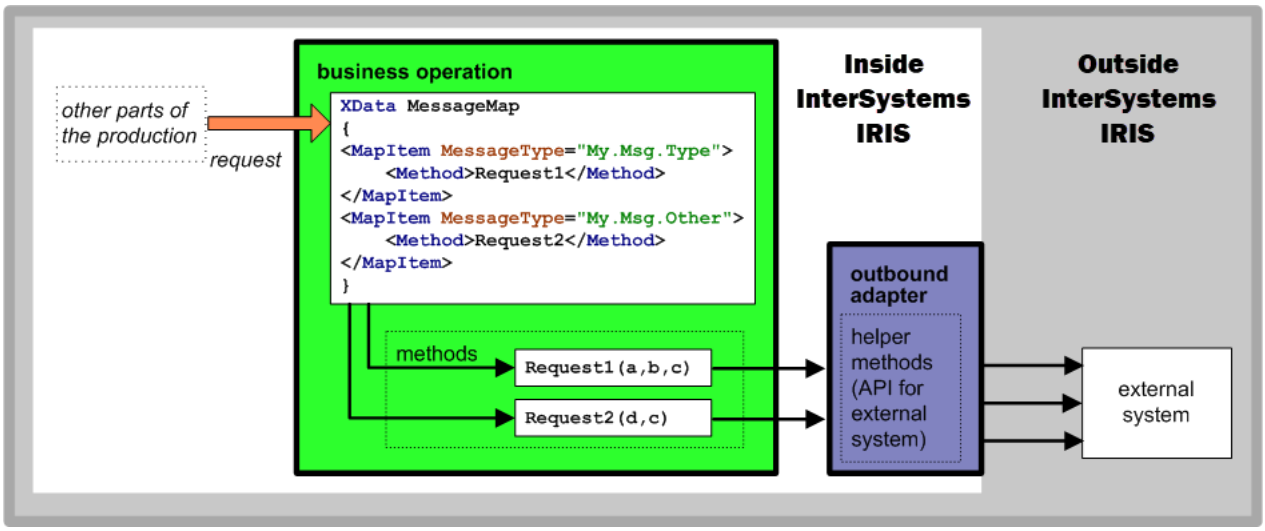
6
ビジネス・オペレーションの定義

このページでは、ビジネス・オペレーション・クラスの定義方法について説明します。

Tipヒント InterSystems IRIS® では、特定の送信アダプタを使用する特殊なビジネス・オペレーション・クラスが提供されており、これらのクラスのいずれかがユーザのニーズに適合している可能性があります。そのため、プログラミングの必要がありません。その一部は、"相互運用プロダクションの概要" の "接続オプション" を参照してください。

6.1 概要

ビジネス・オペレーションは、InterSystems IRIS から外部のアプリケーションまたはシステムに要求を送信します。下の図は、そのしくみを示しています。



この図は、データの入力フローだけで、オプションの応答は含まれていないことに注意してください。

ビジネス・オペレーションは、以下のアクティビティを実行します。

- ビジネス・サービスまたはビジネス・プロセスからの要求を待ちます。
- メッセージ・マップを通して、ビジネス・オペレーション内の特定のメソッドに要求をディスパッチします。ビジネス・オペレーション・クラス内の各メソッドは、外部アプリケーション内の特定のアクションを表しています。
- 要求オブジェクトを、関連する送信アダプタで使用できる形式に変換し、送信アダプタに対しては要求を外部アプリケーションに送信するように指示を出します。

- 必要に応じて、応答オブジェクトを呼び出し側に返します。

各ビジネス・オペレーションには、受信した要求メッセージのタイプに応じて実行すべき外部オペレーションを指定するメッセージ・マップが 1 つずつ含まれています。このメッセージ・マップは 1 つ以上のエントリで構成されており、それぞれのエントリは関連する送信アダプタの 1 回の呼び出しに対応しています。

6.2 基本原理

初めに、"InterSystems IRIS のプログラミング" を参照してください。

規則では、ビジネス・オペレーションは、非常に具体的なオペレーションで、ロジックはほとんど含まれておらず、別のオペレーションを呼び出したり、何らかの形で分岐したりせずに要求を処理します。プロダクションの設計にロジックが必要な場合は、ビジネス・プロセスに組み込まれます。

多くのプロダクションがきわめて単純なビジネス・オペレーションの大規模なセットを提供しています。このような場合は、ビジネス・プロセスに、各オペレーションの呼び出し時期を決定するロジックが組み込まれています。

"ビジネス・サービスの定義" の "基本原理" も参照してください。

6.3 ビジネス・オペレーション・クラスの定義

ビジネス・オペレーション・クラスを作成するには、次のようにクラスを定義します。

- ビジネス・オペレーション・クラスは、Ens.BusinessOperation (またはサブクラス) を拡張する必要があります。
- クラス内では、ADAPTER パラメータを、使用するビジネス・サービス用のアダプタ・クラスの名前と一致させる必要があります。

  または、関連する送信アダプタ・クラスなしでもビジネス・オペレーションを定義できます。この場合、ビジネス・オペレーション自体が、外部アプリケーションとの通信に必要なロジックを持つ必要があります。
- クラス内では、INVOCATION パラメータで、使用する呼び出しスタイルを指定する必要があります。このスタイルは以下のいずれかにする必要があります。
  - 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ることを意味します。その後、メッセージが処理された段階で、別のバックグラウンド・ジョブがそ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、メッセージがターゲットに配信されるまで、送信者のプール内で再び使用可能になることはありません。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>
}
```

  "メッセージ・マップの定義" を参照してください。

- クラスは、メッセージ・マップ内で指定されたすべてのメソッドを定義する必要があります。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、これらのメソッドは、ビジネス・オペレーションの Adapter プロパティのプロパティおよびメソッドを参照します。詳細は、"メッセージ・ハンドラ・メソッドの定義" を参照してください。
- クラスでは設定を追加または削除できます。"設定の追加と削除" を参照してください。
- クラスは任意のまたはすべてのスタートアップ・メソッドおよびティアダウン・メソッドを実装できます。"開始動作と停止動作の上書き" を参照してください。

  InterSystems IRIS は他の多くの異なるデバイスと通信する可能性のある統合プラットフォームであるため、プロパティ値は、サーバ・プラットフォーム、タイム・ゾーン、タイム・フォーマットなど、当てはまるおそれがあるローカリゼーションの問題に依存しません。このようなケースは、プロダクションの実装において処理することをお勧めします。プロパティ値に異なる初期設定を必要とするプロダクションの場合、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。"開始動作と停止動作の上書き" を参照してください。
- クラスにはその内部で作業を完了するためのメソッドを含めることができます。

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 MyProduction.NewOperation Extends Ens.BusinessOperation
{
Parameter ADAPTER = "MyProduction.MyOutboundAdapter";

Parameter INVOCATION = "Queue";

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

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

ビジネス・オペレーション・クラスの例は、"アダプタ・ガイド" を参照してください。

6.4 メッセージ・マップの定義

メッセージ・マップは XML ドキュメントであり、ビジネス・オペレーション・ホスト・クラスの XData MessageMap ブロック内に含まれています。以下に例を示します。

Class Definition

```
Class MyProduction.Operation Extends Ens.BusinessOperation
{

XData MessageMap
{
<MapItems>
  <MapItem MessageType="MyProduction.MyRequest">
    <Method>MethodA</Method>
  </MapItem>
  <MapItem MessageType="Ens.StringRequest">
    <Method>MethodB</Method>
  </MapItem>
</MapItems>
}

}
```

メッセージ・マップのオペレーションは単純です。ビジネス・オペレーションは、受信要求を受け取ると、受信メッセージのタイプと一致する MessageType 属性を持つ最初の MapItem が見つかるまで、メッセージ・マップの先頭から各 MapItem を検索します。次に、この MapItem に関連のあるオペレーション・メソッドを呼び出します。

メッセージ・マップについて注意する点は以下のとおりです。

- メッセージ・マップは先頭から最後にかけて検索されます。一致する項目が検出されたら、それ以上の検索は行われません。
- 受信要求オブジェクトが所定の MessageType のサブクラスである場合は、一致対象と見なされます。サブクラスを除外する場合は、サブクラスをメッセージ・マップ内でスーパー・クラスの上に配置する必要があります。
- 受信要求がどの MapItem エントリとも一致しない場合、OnMessage メソッドが呼び出されます。

6.5 メッセージ・ハンドラ・メソッドの定義

ビジネス・オペレーション・クラスを作成するときの最大のタスクは、このアダプタで使用するためのメッセージ・ハンドラ、つまり、プロダクション・メッセージを受け取ってから、プロダクションの外部のターゲットと通信するためのアダプタのメソッドを呼び出す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。

一般的には、メソッドで以下の一部または全部を実行する必要があります。

1. オプションで、ビジネス・オペレーション・クラスのプロパティを設定します (適時)。"ビジネス・オペレーションのプロパティ" を参照してください。
2. 入力オブジェクトを検査します。
3. 応答クラスのインスタンスを作成します。
4. アダプタの該当するメソッドを呼び出します (複数可)。これらのメソッドは、ビジネス・オペレーションの Adapter プロパティ経由で利用できます。以下に例を示します。

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

   このメソッドについては、後で説明します。

または、プロダクション内のターゲットにメッセージを送信する場合は、"プロダクション内のターゲットに対する要求の送信" を参照してください。

5. 応答を調べます。
6. 応答内の情報を使用して、応答メッセージ (Ens.Response またはサブクラスのインスタンス) を作成します。メソッドは出力としてこのメッセージを返します。

   メッセージ・クラスの定義方法は、"メッセージの定義" を参照してください。
7. 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。
8. 適切なステータスを返します。この手順は必須です。

6.6 ビジネス・オペレーションのプロパティ

オペレーションのメソッド内では、以下のようなビジネス・オペレーション・クラスのプロパティが使用できます。

プロパティ	説明
%ConfigName	このビジネス・オペレーションの構成名。
%SessionId	現在処理されているメッセージのセッション ID。
Adapter	このビジネス・オペレーションに関連のある送信アダプタ。
DeferResponse	このビジネス・オペレーションからの応答を延期して後で送信するには、DeferResponse プロパティを整数値の 1 (真) に設定し、ビジネス・オペレーションを終了する前に、延期された応答配信トークンを取得します。
FailureTimeout	再試行を実行できる時間 (秒)。この時間を経過すると、再試行は不可能になり、エラー・コードが返されます。"Retry" および "RetryInterval" を参照してください。
Retry	現在のメッセージを再試行する場合は、このプロパティを整数値の 1 (真) に設定します。通常、再試行機能は、外部アプリケーションから応答がなく、エラーを生成せずに再試行を行う場合に使用されます。"RetryInterval" および "FailureTimeout" を参照してください。
RetryInterval	このメッセージが再試行を行うようにマーキングされている場合に、出力システムに再試行アクセスを行う頻度 (秒単位)。"Retry" および "FailureTimeout" を参照してください。
SuspendMessage	ビジネス・オペレーションで現在処理中のメッセージを保留状態としてマーキングする場合は、このプロパティを整数値の 1 (真) に設定します。"保留中のメッセージ" の節を参照してください。

6.7 アダプタ・メソッドの呼び出し

通常、ビジネス・オペレーションには、外部システムとの通信に使用するロジックは含まれません。代わりに、このロジックを処理する送信アダプタが使用されます。ビジネス・オペレーションを送信アダプタに関連付けると、そのビジネス・オペレーションは、そのアダプタのメソッドを呼び出してデータを送受信します。アダプタ・メソッドの呼び出しの詳細は、"ビジネス・ホストからのプロパティとメソッドへのアクセス" を参照してください。

6.8 プロダクション内のターゲットに対する要求の送信

ビジネス・オペレーションは、主に、特定の外部アプリケーションへの要求の配信を担当しますが、必要に応じて、他のビジネス・オペレーションまたはビジネス・プロセスにメッセージを送信することもできます。プロダクション内のターゲットにメッセージを送信するには、SendRequestSync()、SendRequestAsync()、または SendDeferredResponse() を呼び出します。

これらのメソッドの詳細は、"ビジネス・サービスの定義" の "要求メッセージの送信" を参照してください。

Ens.BusinessOperation は、DeferResponse() を使用可能な追加のメソッドを定義します。

6.8.1 DeferResponse() メソッド

このメソッドは、成功または失敗を示す %Status 値を返します。これにより、1 つの参照渡しの引数、token が提供され、この引数は後で SendDeferredResponse() を呼び出すために必要な延期された応答配信トークンを返します。以下に例を示します。

ObjectScript

```
Set sc=..DeferResponse(.token)
// Send the token out somewhere...
Quit $$$OK
```

遅延送信の概要は、"InterSystems IRIS のプログラミング" の "遅延送信の使用法" を参照してください。

6.9 保留中のメッセージ

ビジネス・オペレーションで現在処理中のメッセージを一時停止ステータスとしてマーキングする場合は、ビジネス・オペレーション・プロパティの SuspendMessage を整数値の 1 (真) に設定します。通常、ビジネス・オペレーションでは、何らかの理由で外部システムから拒否されたメッセージに対して保留状態に設定します。

一時停止メッセージは特殊なキューに入れられるため、システム管理者は問題を診断して解決してからメッセージを再送できます。また、システム管理者は、単純な再送 (元のターゲット宛て) を実行することも、新しい宛先に送信することもできます。詳細は、"プロダクションの監視" を参照してください。

以下のサンプル・メソッドは、ビジネス・オペレーションからドキュメントを外部システムに送信します。送信直前のドキュメントを検証する Validate() の呼び出しからエラーが返されると、このメソッドは SuspendMessage プロパティを 1 に設定します。

Class Member

```
Method validateAndIndex(pDoc As MyX12.Document) As %Status
{
  If ""=..Validation||'$method($this,"OnValidate",pDoc,..Validation,.tSC) {
    Set tSC=##class(MyX12.Validator).Validate(pDoc,..Validation)
  }
  Set:'$D(tSC) tSC=$$$OK
  If $$$ISERR(tSC) {
    Set ..SuspendMessage=1
    Do ..SendAlert(##Class(Ens.AlertRequest).%New($LB(
        ..%ConfigName,"Suspended document "_pDoc.%Id()_
        " because it failed validation using spec '"
        _..Validation_"' with error "_
        $$$StatusDisplayString(tSC))))
    Quit tSC
  }
  If ""'=..SearchTableClass {
    TRY {
      Set tSCStore=$classmethod(..SearchTableClass,"IndexDoc",pDoc)
```

```
      If $$$ISERR(tSCStore)
        $$$LOGWARNING("Failed to create SearchTable entries")
    }
    CATCH errobj {
      $$$LOGWARNING("Failed to invoke SearchTable class")
     }
  }
  Quit $$$OK
}
```

7

アラート・プロセッサの定義

システム・アラートとユーザ生成アラートは、プロダクションの問題をユーザに通知する手段を提供します。アラート・プロセッサは、該当するユーザに電子メール、携帯電話、またはその他のメカニズムを介して修正すべき問題を通知するビジネス・ホストです。多くの場合、アラート・プロセッサはカスタム・コードを作成することなく定義できます。アラート・プロセッサをプロダクションへ追加する方法の詳細は、"アラートの監視" を参照してください。このページでは、アラート・プロセッサをカスタム・コードで作成する方法を説明します。

7.1 背景情報

ビジネス・ホストはアラートを送信できます。"InterSystems IRIS のプログラミング" の "アラートの生成" を参照してください。InterSystems IRIS® は、プロダクション内の設定値に応じて、特定の事象の発生時に自動的にアラートを送信することもできます。プロダクションに Ens.Alert という名前のビジネス・ホストが含まれている場合は、InterSystems IRIS は自動的にそのビジネス・ホストに特殊な要求メッセージ (Ens.AlertRequest) を送信します。このビジネス・ホストがプロダクションのアラート・プロセッサです。1 つのプロダクションにこのプロセッサを 2 つ以上含めることはできません。

これにより、アラート・プロセッサは、このメッセージ内の情報を使用して接続先を決定できます。一般的なシナリオをいくつか紹介します。

- すべてのアラートを同じ出力メカニズムを介して処理できる場合は、アラート・プロセッサを該当するアダプタを使用するビジネス・オペレーションにすることができます。"簡単な電子メール・アラート・プロセッサの使用" と "簡単な送信アダプタ・アラート・プロセッサの使用" を参照してください。
- アラートをルーティングする必要があるが、アラートの解決を追跡する必要がない場合は、"ルーティング・アラート・プロセッサの使用" を参照してください。
- アラートの解決プロセスを追跡する場合は、"アラート管理を使用したアラート解決の追跡" を参照してください。

すべてのケースにおいて、InterSystems IRIS は、アラート・タイプの InterSystems IRIS イベント・ログに情報を書き込みます。

注釈　Ens.Alert は、アラート・プロセッサとして機能するビジネス・ホストの必須名です。これとクラス名を混同しないでください。アラート・プロセッサは任意のクラス名を使用できます。

7.2 簡単な電子メール・アラート・プロセッサの使用

すべてのアラートを電子メール経由で送信することが望ましい場合は、Ens.Alert コンポーネントに EnsLib.EMail.AlertOperation クラスを使用します。この特殊なビジネス・オペレーションは以下の処理を実行します。

- ADAPTER パラメータは EnsLib.EMail.OutboundAdapter として指定されます。

  このアダプタは、電子メールの受信者だけでなく、SMTP 電子メール・サーバを使用するために必要な情報も指定する設定を提供します。

  また、電子メールを構成されたサーバ経由で送信するヘルパー・メソッドも提供します。

- OnMessage() メソッドが実装されます。このメソッドは入力として Ens.AlertRequest を期待しています。

  また、このメソッドは、以下の処理を実行します。

  1. Ens.AlertRequest からアラート・テキストを読み取ります。
  2. 電子メール・メッセージ (%Net.MailMessage のインスタンス) を作成して、それにアラート・テキストを書き込みます。
  3. すべての構成された受信者に電子メール・メッセージを送信します。

このクラスを変更せずに使用することもできます。または、そのサブクラスを作成して使用することもできます。

7.3 簡単な送信アダプタ・アラート・プロセッサの使用

すべてのアラートを同じ出力メカニズムを介して処理できるが、EnsLib.EMail.AlertOperation を使用できない場合は、次のように Ens.Alert というビジネス・オペレーションを作成します。

- ADAPTER パラメータを適切なアダプタ・クラスの名前として指定します。
- OnMessage() メソッドを実装します。このメソッドのシグニチャは次のようにする必要があります。

  ```
  Method OnMessage(pRequest As Ens.AlertRequest, Output pResponse As Ens.Response) As %Status
  ```

  実装内で、必要に応じて、アダプタのメソッドを呼び出します。

  "ビジネス・オペレーションの定義" と "アダプタのドキュメント" を参照してください。

電子メール・アドレスや電話番号などの詳細情報を構成できるようにクラスを定義することをお勧めします。"設定の追加と削除" を参照してください。

7.4 ルーティング・アラート・プロセッサの使用

ユーザに複数の出力メカニズムを介して連絡する必要がある場合は、アラート・プロセッサを、Ens.AlertRequest メッセージのルーティング方法を決定するビジネス・プロセスにする必要があります。この場合は、プロダクションに出力メカニズムごとに 1 つずつの新しいビジネス・オペレーションを追加する必要があり、アラート・プロセッサがメッセージをそれらのビジネス・オペレーションに転送します。

7.4.1 ルーティング・プロセスとしてのアラート・プロセッサの定義

アラート・プロセッサをルーティング・プロセスとして定義するには、Ens.AlertRequest メッセージを受信可能なビジネス・プロセス・クラスを作成します。

このビジネス・プロセスはこれらのメッセージを調べて、アラートの内容と組み込まれたロジックに応じて異なるビジネス・オペレーションに転送します。

作成するロジックでは、以下の要因が考慮されている必要があります。

- ユーザの種類ごとに異なる要件
- 時刻に応じて異なる要件
- 組織の問題解決のためのポリシーと手順

EnsLib.MsgRouter.RoutingEngine クラスを Ens.Alert のルーティング・プロセスとして使用できます。このクラスには、**[ビジネスルール名]** という設定があります。この設定をルーティング・ルール・セットの名前として指定した場合は、このビジネス・ホストがルール・セット内のロジックを使用して、受け取ったすべてのメッセージを転送します。

7.4.2 ビジネス・オペレーションの定義

"簡単な電子メール・アラート・プロセッサの使用" または "簡単な送信アダプタ・アラート・プロセッサの使用" の説明に従って、必要なビジネス・オペレーションのそれぞれを定義できます。

7.5 アラート管理へのカスタム・コードの追加

アラート管理によって、ユーザへのアラートの割り当て、アラートのステータスの追跡、およびアラートの解決の進捗を管理できます。アラート管理の概要は、アラート管理コンポーネントの構成方法と、アラート管理のルールおよびデータ変換の方法について説明する "アラート管理の構成" を参照してください。この節では、アラート管理コンポーネントへのカスタム・コードの追加方法について説明しています。

アラート管理フレームワークには、以下のアーキテクチャが備わっています。

- 管理対象アラートにはそのライフサイクルに渡って 1 つの永続オブジェクトが使用される。
- アラート・マネージャ、通知マネージャ、およびアラート・モニタの全体的な内部構造は同一。これらのコンポーネントの 1 つが呼び出されると、3 つのフェーズでその機能が実行されます。
  1. 最初にコンポーネントは、OnProcess メソッドがサブクラスによって実装されている場合に、このメソッドを実行します。このメソッドを実装することで、コンポーネントにカスタム・コードを含めることができます。OnProcess メソッドが処理が完了したことを示すフラグを設定すると、コンポーネントが終了します。
  2. 次に、コンポーネントは、コンポーネントのアクションを制御するためのパラメータを設定するルールを評価するか、通知マネージャの場合はデータ変換を評価します。
  3. 最後に、コンポーネントは、ルールによって設定されたパラメータか、またはコンポーネントの構成によって設定されたデフォルトを基にアクションを実行します。
- アラート通知オペレーションは、メッセージの形式を指定し、その宛先に転送する単純なコンポーネントです。

7.5.1 アラート・マネージャ

アラート・マネージャは Ens.Alerting.AlertManager クラスを持ち、Ens.Alert と命名される必要があります。アラート・マネージャは、すべてのプロダクション・コンポーネントからアラートを受信します。アラート・マネージャは、ルールで指定されている条件に基づいてアラートを管理対象アラートに昇格できます。アラート・マネージャは管理対象アラートを通知マネージャに送信します。

アラート・マネージャは 3 つのフェーズで実行されます。

1. コンポーネントのクラスが OnCreateManagedAlert() メソッドをオーバーライドしている場合は、オーバーライドを実行します。アラート要求を処理するカスタム・コードを使用して、このメソッドで管理対象アラートを作成することができます。ベースとなるアラート・マネージャ・コードで、ルールを評価すること、および管理対象アラートを作成して通知マネージャに送信することを希望しない場合は、tProcessingComplete パラメータを 1 に設定する必要があります。この場合、アラート・マネージャではさらなる処理は実行されません。
2. CreateManagedAlertRule ルールを評価します。このルールでは、tAlertContext へアクセスできます。真の値 (1) が返された場合、アラート・マネージャは管理対象アラートを作成します。偽が返された場合、アラート・マネージャは管理対象アラートを作成せず、アラートはログに書き込まれるだけです。アラートのコンテキストでは、以下へのアクセスが可能です。
   - 受信アラート
   - アラートを発信したコンポーネントに対して構成されたアラート・グループ
   - アラートを発信したコンポーネントに対して構成されたビジネス・パートナー
   - アラートの所有者

   ルールは、0 を返すことでアラートの管理対象アラートへの昇格を抑制できます。また、1 を返すことでアラートを管理対象アラートへ昇格させることができます。
3. ルールが、tCreateAler を 1 に設定した場合、アラート・マネージャは管理対象アラートを作成します。また、定義済みの CreateManagedAlertRule ルールがない場合、アラート・マネージャはデフォルトのアクションを実行し、管理対象アラートを作成します。アラート・マネージャは、OnCreateManagedAlert() メソッドを呼び出すことで管理対象アラートを作成します。このメソッドは、Ens.Alerting.AlertManager を拡張するクラスによってオーバーライド可能です。OnCreateManagedAlert() のデフォルトの実装では、管理対象アラートにプロダクション名が設定され、現在の所有者は値を空の文字列にすることで割り当てられません。アラート・マネージャが管理対象アラートを作成した場合、通知マネージャに送信します。

7.5.2 通知マネージャ

通知マネージャには、Ens.Alerting.NotificationManager クラスがあり、通知先のグループと使用する通知オペレーションを決定する役割があります。

通知マネージャは 3 つのフェーズで実行されます。

1. コンポーネントのクラスが OnProcessNotificationRequest() メソッドをオーバーライドしている場合は、オーバーライドを実行します。オーバーライドが pProcessingComplete パラメータを 1 に設定する場合、通知マネージャは変換を評価せず、デフォルトのアクションを適用します。
2. データ変換が構成されている場合には、実行します。データ変換の詳細は、"通知マネージャの追加とそのデータ変換の定義" を参照してください。
3. 変換によって target.Notify プロパティが 1 に設定された場合、またはデータ変換がない場合、通知マネージャは各ターゲットのリストにあるコンポーネントにアラート通知を送信し、アドレスのリストをターゲットに渡します。

通知マネージャは、管理対象アラート・オブジェクトを送受信せずに、永続管理対象アラート・オブジェクトへの参照を含んだ通知要求オブジェクトを使用します。

7.5.3 アラート・モニタ

アラート・モニタは、現在の時刻が NextActionTime の値を超えているすべての開かれた管理対象アラートをクエリします。以下の SQL クエリを発行します。

```
"SELECT ID FROM Ens_Alerting.ManagedAlert WHERE IsOpen = 1 AND NextActionTime <= ?"
```

ここで、$$$timeUTC によって返される現在の時刻はパラメータとして指定されます。

アラート・モニタは返された各管理対象アラート・メッセージを個別に処理します。各管理対象アラートは、3 つのフェーズで処理されます。

1. コンポーネントのクラスが OnProcessOverdueAlert() メソッドをオーバーライドしている場合は、オーバーライドを実行します。アラートを処理するカスタム・コードを使用することができます。ベースとなるアラート・モニタ・コードで、ルールを評価すること、および管理対象アラートを更新して通知マネージャに送信することを希望しない場合は、tProcessingComplete パラメータを 1 に設定する必要があります。この場合、アラート・モニタではさらなる処理は実行されません。
2. OverdueAlertRule ルールを評価します。このルールでは、tOverdueContext へアクセスできます。期限切れのコンテキストでは、以下へのアクセスが可能です。
   - 受信アラート
   - 現在の時刻
   - NewNextActionTime
   - NewEscalationLevel

   このルールは、0 を返すことでリマインダの送信を抑制するか、NewNextActionTime を設定することで管理対象アラートがアラート・モニタによって検出される次の時刻を設定するか、または NewEscalationLevel を設定することで管理対象アラートを昇格または解除できます。

   アラート・ルールのコンテキスト、およびアラート・モニタによる結果の処理方法はオーバーライドできます。
   - GetOnOverdueAlertContext() メソッドをオーバーライドすることで、アラート・ルールのコンテキストに情報を追加できます。
   - OnProcessOverdueRuleResult() メソッドをオーバーライドすることで、アラート・モニタによるルールの結果の処理方法をオーバーライドできます。オーバーライドしない場合は、このメソッドをベース・クラスで実行します。OnProcessOverdueRuleResult() メソッドは管理対象アラートの昇格を実行します。このオーバーライドでは、管理対象アラート、tOverdueContext、tSendNotification、および tNotificationType へアクセスできます。ベース・クラスの実装の機能を再現するか、##super() を呼び出すことでその機能を呼び出す必要があることに注意してください。
3. ルールが 1 を返す場合、アラート・マネージャは管理対象アラートを通知マネージャに送信します。

7.5.4 通知オペレーション

通知オペレーションはユーザのグループに通知を送信します。複数種類のメカニズムを使用して通知を送信する場合は、転送方法ごとに独立した通知オペレーションを使用する必要があります。

8

データ変換の定義

このページでは、データ変換について説明します。

8.1 概要

データ変換では、別のメッセージを変換して新しいメッセージが作成されます。メッセージを変換すると、データ変換によってオリジナルの変換である新しいメッセージ・ボディが送信されます。以下のような変換がこのプロセス中に行われます。

- ソース上のプロパティからターゲット上のプロパティへの値のコピー
- ソース上のプロパティの値を使用した計算の実行
- ターゲット上のプロパティへの計算結果のコピー
- ターゲット上のプロパティに対するリテラル値の割り当て
- ターゲットに関係しないソース上のプロパティの無視

8.2 DTL 変換の定義

DTL 変換は Ens.DataTransformDTL を基本とするクラスです。この場合は、管理ポータルまたは IDE からアクセス可能な DTL エディタで変換を視覚的に作成および編集できます。DTL エディタは非技術系ユーザ向けのエディタです。"DTL 変換の開発" を参照してください。

8.3 カスタム変換の定義

カスタム変換 は、以下の情報を指定する Ens.DataTransform のサブクラスです。

- 入力 (ソース) メッセージ・クラスの名前
- 出力 (ターゲット) メッセージ・クラスの名前
- 出力オブジェクトのプロパティに値を割り当てる一連の操作

各割り当て処理は、Ens.DataTransform クラス・メソッドの Transform() の呼び出しで構成されます。引数は単純な式で、それが評価されて、出力クラスのプロパティの 1 つに値が指定されます。式に含めることができる要素は以下のとおりです。

- リテラル値
- context という名前の汎用実行コンテキスト変数内のプロパティ
- ソース・オブジェクトのプロパティ
- 式言語の関数と式
- InterSystems IRIS® で用意されているメソッドの呼び出し
- ユーザが用意したメソッドの呼び出し

9
ビジネス・メトリックの定義

ビジネス・メトリックは、ダッシュボードまたはプロダクション・モニタで表示するための、プロダクションの性能に関連した 1 つ以上の値を測定または計算します。このページでは、ビジネス・メトリックの作成方法と表示方法について説明します。

注釈　別の方法として、サードパーティ・ビジネス・アクティビティ監視製品と InterSystems IRIS® を組み合わせて使用することもできます。このような製品は、Web サービス、JDBC、ODBC をはじめとする任意の接続テクノロジを介して、InterSystems IRIS と相互運用できます。

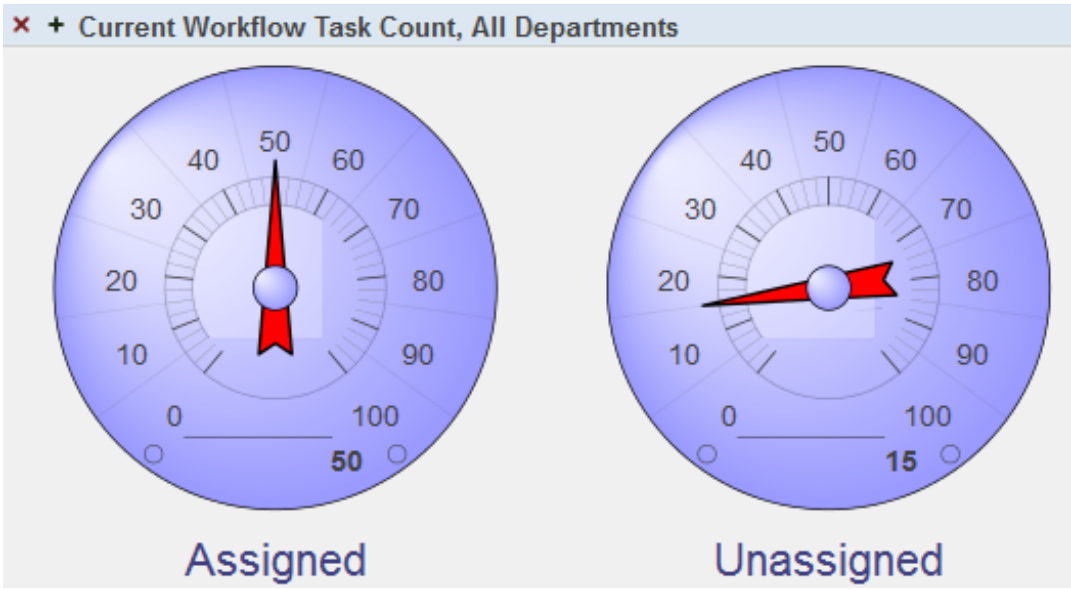
9.1 InterSystems IRIS ビジネス・メトリックの概要

ビジネス・メトリックは、プロダクションに含める特殊なビジネス・サービス・クラスです。プロダクションが動作中は、ビジネス・メトリック値を表示に使用できます。プロダクションが動作していない場合は、値が NULL になります。

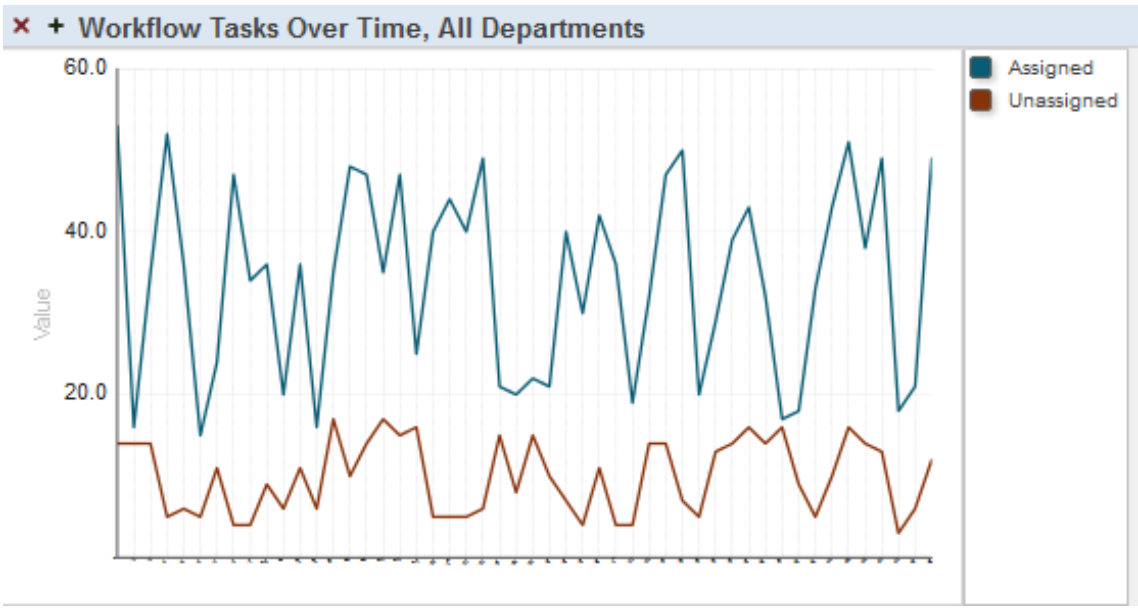
9.1.1 ビジネス・メトリックのプロパティ

ビジネス・メトリック内の値は、プロパティと呼ばれます。単純なプロパティと自動履歴付き多次元プロパティの 2 種類の一般的なビジネス・メトリック・プロパティがあります。

単純なプロパティは、常に、1 つの値しか保持しません。下の例は、2 つの単純なプロパティを持つビジネス・メトリックを示しています。



自動履歴付きプロパティは、複数の値 (時間単位ごとに 1 つずつの値) を保持し、一番最後が最も新しい値です。値を記録する回数を制御できます。下の例は、2 つの自動履歴付きのプロパティを持つビジネス・メトリックを示しています。



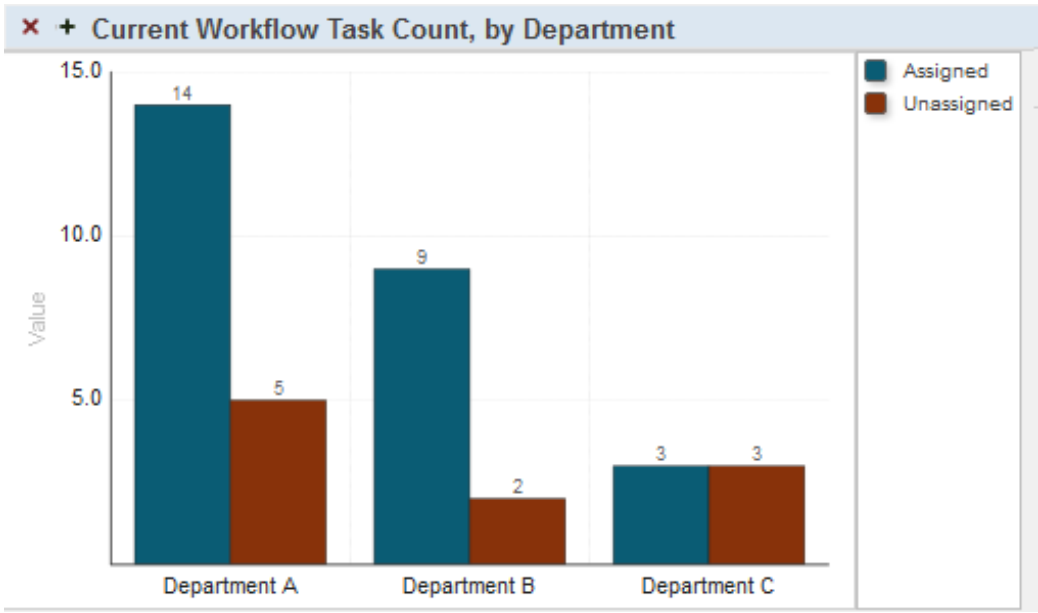
ビジネス・メトリック・クラスの開発時には、任意の方法でメトリック・プロパティに値を提供できます。例えば、プロダクション・メッセージやビジネス・プロセス・インスタンスに格納したデータに基づく値、ビジネス・プロセスによって InterSystems IRIS に保持しているデータに基づく値、外部のアプリケーションやデータベースに要求を送信して取得したデータに基づく値を指定できます。

9.1.2 単一インスタンス・ビジネス・メトリックと複数インスタンス・ビジネス・メトリック

一般的なビジネス・メトリックには、単一インスタンス・ビジネス・メトリックと複数インスタンス・ビジネス・メトリックの 2 種類があります。

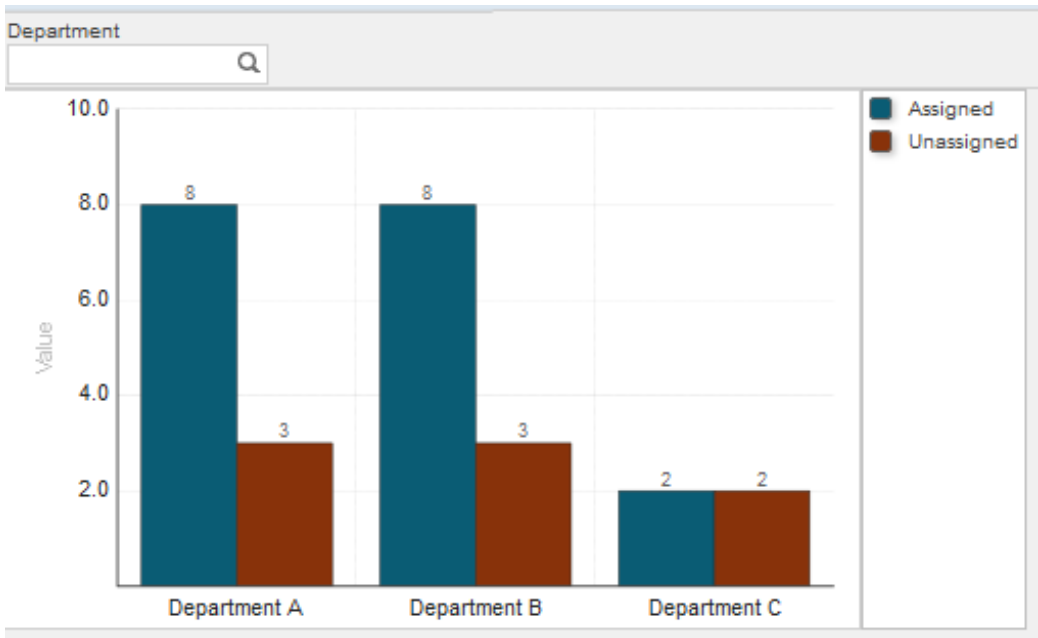
単一インスタンス・ビジネス・メトリックは、単一のメトリック値のセットを保持します。前の節の例は単一インスタンス・ビジネス・メトリックを示していました。

複数インスタンス・ビジネス・メトリックは、複数のメトリック値のセット（メトリックで定義されたインスタンスごとに 1 つずつのセット）を保持します。このメトリックは、複数の同様の項目のメトリックを比較する場合に役立ちます。各項目は別個ですが、他の項目と共通するプロパティを持っています。例えば、各部門で、割り当てられていないワークフロー・タスクの数と割り当てられたワークフロー・タスクの数をカウントする場合などです。1 つのビジネス・メトリックに、部門ごとに 1 つずつのインスタンスを割り当てることができます。2 つの単純なプロパティを持つ複数インスタンス・ビジネス・メトリックの例を以下に示します。

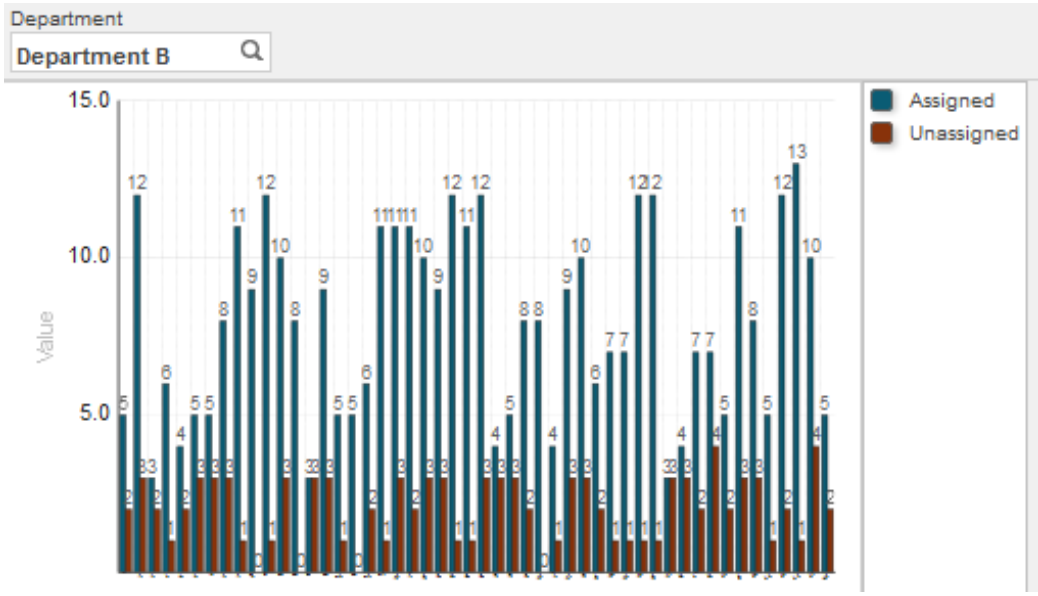


複数インスタンス・ビジネス・メトリックは、自動履歴付き多次元プロパティも持つことができます。ただし、実際には、インスタンスと履歴を同時に表示することはできません。このようなビジネス・メトリックを定義してダッシュボードに追加すると、デフォルトで、すべてのインスタンスの現在値がダッシュボードに表示されます。ユーザが単一のインスタンスを選択できるようにするフィルタを含めることができます。その場合は、ダッシュボードにそのインスタンスの履歴を表示できます。

以下に例を示します。



次に、ユーザがインスタンスを選択した場合：



9.1.3 ビジネス・サービスとしてのビジネス・メトリック

すべてのビジネス・メトリック・クラスは、それ自体が Ens.BusinessServiceクラスから派生した Ens.BusinessMetric クラスから派生しているため、ビジネス・サービスのすべての機能を備えています。例えば、プロダクション定義にビジネス・メトリックを追加したり、ビジネス・メトリックに論理名を割り当てたり、定期的に実行する (特定の時間間隔でメトリック・プロパティを計算し直す) ようにスケジュールしたり、ビジネス・オペレーションとビジネス・プロセスをメトリック値計算の一部として呼び出したりできます。

9.2 単一インスタンス・ビジネス・メトリックの定義

単一インスタンス・ビジネス・メトリックを定義するには、以下の要件を満たすクラスを定義します。

- Ens.BusinessMetric のサブクラスにする必要があります。

  このクラスの場合は、ADAPTER パラメータが Ens.InboundAdapter で、[CallInterval] 設定があります。これにより、ビジネス・メトリックが定期的に呼び出されることが保証されます。

- 1 つ以上のビジネス・メトリック・プロパティを定義する必要があります。詳細は、単純なプロパティと自動履歴付きプロパティで異なります。

- オプションで、例えば、値の範囲を制御するために、プロパティ・パラメータの値を指定できます。

- ビジネス・メトリック・プロパティに値を割り当てる必要があります。そのためには、OnCalculateMetrics() メソッドを実装する必要があります。

以降の項で詳しく説明します。

"複数インスタンス・ビジネス・メトリックの定義" も参照してください。

9.2.1 単純なビジネス・メトリック・プロパティの定義

単純なビジネス・メトリック・プロパティを定義するには、次のように、ビジネス・メトリック・クラスにプロパティを追加します。

Class Member

```
Property MetricProperty As Ens.DataType.Metric;
```

このプロパティは、数値と文字列値のどちらかを保持できます。

ここで、MetricProperty はビジネス・メトリック・プロパティの名前です。以下に例を示します。

Class Member

```
/// This metric tracks A/R totals
Property AccountsReceivable As Ens.DataType.Metric;
```

このプロパティは、数値と文字列値のどちらかを保持できます。

9.2.2 自動履歴付きビジネス・メトリック・プロパティの定義

自動履歴付きビジネス・メトリック・プロパティを定義するには、次のように、ビジネス・メトリック・クラスにプロパティを追加します。

Class Member

```
Property MetricProperty As Ens.DataType.Metric (AUTOHISTORY=50) [MultiDimensional];
```

AUTOHISTORY パラメータには、任意の正の整数を使用できます。以下に例を示します。

Class Member

```
/// Recent Sales History
Property SalesHistory As Ens.DataType.Metric (AUTOHISTORY = 50) [ MultiDimensional ];
```

一般に、この種のプロパティの目的は、一定期間に一定の間隔で値を収集し、収集した一連の数値をグラフ上にプロットすることです。そのため、割り当てられる値は数値がほとんどです。

収集の頻度は、構成されたビジネス・メトリックの **[呼び出し間隔]** 設定によって制御されます。

9.2.3 メトリック・プロパティのその他のパラメータの指定

他にも、ビジネス・メトリック・プロパティのプロパティ・パラメータを指定できます。これらのパラメータには、メトリックを表示するメーターの外観を制御する上限と下限のデフォルト値が含まれます。以下に例を示します。

Class Member

```
/// Total Sales for the current day.
Property TotalSales As Ens.DataType.Metric (RANGELOWER = 0, RANGEUPPER = 50, UNITS = "$US");
```

下の表は、(前の節で説明した AUTOHISTORY に加えて) Ens.DataType.Metric で使用可能なプロパティ・パラメータを示しています。これらのパラメータは、数値を保持するメトリック・プロパティに適用されます。メトリック・プロパティが文字列値を保持する場合には適用されません。

パラメータ	説明
LINK	オプション。メトリックに関連するブラウザ・ページを指定する URL。ダッシュボードのユーザがこのパラメータに関連付けられたメーターを右クリックして [ドリル・ダウン] オプションを選択すると、ブラウザにこのページが表示されます。
RANGELOWER	オプション。メトリックに想定される値の範囲の低い方を指定する数値。この値は、このメトリックに関連付けられたメーターの下位範囲のデフォルトになります。

パラメータ	説明
RANGEUPPER	オプション。メトリックに想定される値の範囲の高い方を指定する数値。この値は、このメトリックに関連付けられたメーターの上位範囲のデフォルトになります。
THRESHOLDLOWER	オプション。メトリックに想定される低い方のしきい値を指定する数値。この値は、このメトリックに関連付けられたメーターの下位しきい値のデフォルトになります。
THRESHOLDUPPER	オプション。メトリックに想定される高い方のしきい値を指定する数値。この値は、このメトリックに関連付けられたメーターの上位しきい値のデフォルトになります。
UNITS	オプション。メトリックの単位を指定する、二重引用符に囲まれたユーザ定義の文字列。例として "$US" または "リットル" などがあります。この文字列は、ダッシュボード画面で、ユーザが画面の上半分に表示された該当のメーターをクリックすると、画面の下半分に表示されます。

9.2.4 ビジネス・メトリック・プロパティへの値の割り当て

ここでは、単一インスタンス・ビジネス・メトリックのビジネス・メトリック・プロパティへの値の割り当て方法について説明します。複数インスタンス・ビジネス・メトリックの詳細は、このページの後半で説明します。

ビジネス・メトリック・プロパティに値を割り当てるには、ビジネス・メトリック・クラスの OnCalculateMetrics() インスタンス・メソッドを実装します。OnCalculateMetrics() の目的は、クラス内の任意のメトリック・プロパティの値を計算、検索、または設定することです。

以下の例では、2 つのメトリック・プロパティ、Counter および SalesHistory を定義しています。この例では、OnCalculateMetrics() メソッドが、単一値 Counter プロパティをインクリメントして、SalesHistory プロパティをランダム値に更新するだけです。

Class Definition

```
/// Example Business Metric class
Class MyProduction.MyMetric Extends Ens.BusinessMetric
{

/// Number of times these metrics have been calculated.
Property Counter As Ens.DataType.Metric
         (RANGELOWER = 0, RANGEUPPER = 100000, UNITS = "Events");

/// Total Sales for the current day.
Property SalesHistory As Ens.DataType.Metric
         (RANGELOWER = 0, RANGEUPPER = 50, AUTOHISTORY = 50, UNITS = "$US")
         [ MultiDimensional ];

/// Calculate and update the set of metrics for this class
Method OnCalculateMetrics() As %Status
{
  // set the values of our metrics
  Set ..Counter = ..Counter + 1
  Set ..SalesHistory = $GET(..SalesHistory) + $RANDOM(10) - 5

  Quit $$$OK
}

}
```

注：

- 単純なプロパティと自動履歴付きプロパティは値の指定方法が同じであることに注目してください

  (多次元プロパティに慣れている場合は、ここで示すように、添え字が付けられていないプロパティの最上位ノードの値しか指定しないことに注意してください。AUTOHISTORY パラメータに従って、InterSystems IRIS は、配列内の下位ノード (整数添え字が付けられた配列) を自動的に維持するコードを生成します。例えば、`SalesHistory(1)` は `SalesHistory` プロパティ内の最も古い値です)。

- このメソッドでは、オプションで、ビジネス・オペレーションとビジネス・プロセスを呼び出すことができます。値の計算に必要な API を呼び出すこともできます。
- 自動履歴付きプロパティに NULL 値を含めることはできません。NULL 値は表示されないため、棒グラフと折れ線グラフの棒やエントリの数が合わなくなります。これにより、軸ラベルとラベルで表示する項目が不一致になることがあります。このような問題を避けるには、すべての NULL 値を 0 に置き換えます。
- OnCalculateMetrics() メソッドの呼び出しの前に、すべてのメトリック・プロパティは、最後に計算された値に設定されます (そのような値がある場合)。OnCalculateMetrics() の呼び出し後、すべてのメトリック・プロパティの値は、ダッシュボードやそれらの値を必要とする関係者など (存在する場合) が後から使用できるように、ビジネス・メトリックのキャッシュに格納されます。

9.3 複数インスタンス・ビジネス・メトリックの定義

複数インスタンス・ビジネス・メトリックを定義するには:

- OnCalculateMetrics() の実装と詳細が少し異なる点を除いて、前の節の手順に従います。
- インスタンス名を定義します。そのためには、以下を実行します。
  - 静的なインスタンス名のセットを定義します。そのためには、配列に名前の固定リストを割り当てる OnGetInstances() メソッドを実装します。このアプローチはインスタンスのセットが静的な場合に有効です。
  - インスタンス名を動的に定義します。そのためには、SQL データベースの列から名前のリストを取得する MetricInstances() クエリを追加します。このアプローチは、項目数または項目名が時間の経過に伴って変化することを想定している場合に有効です。
  - これらのアプローチを組み合わせます。MetricInstances() クエリを使用して初期リストを取得してから、OnGetInstances() を使用して名前を追加または置換します (ビジネス・メトリック・インスタンスは、最初に MetricInstances() を呼び出してから、OnGetInstances() を呼び出します)。
- OnCalculateMetrics() の実装では、**%Instance** プロパティの値をチェックして、ビジネス・メトリック・プロパティにそのインスタンスに適切な値を割り当てます。
- 以下の原則を覚えておいてください。
  - インスタンス名は文字列です。
  - インスタンス名は一意である必要があります。
  - インスタンス名は、ユーザに対してダッシュボード上に表示される場合があるため、簡潔でわかりやすい適切な名前を使用してください。
  - インスタンス数は適度なものにしてください。インスタンスが多数あると、計算にコストがかかり、ユーザにとって理解しにくくなる可能性があります。

以降の項で詳しく説明します。

9.3.1 静的なインスタンス名のセットの定義

静的なインスタンス名のセットを定義するには、OnGetInstances() メソッドを上書きします。このメソッドには、配列が参照によって渡されます。OnGetInstances() メソッドは、1 から始まる序数を添え字として使用した名前をこの配列に格納する必要があります。単純な例を以下に示します。

Class Member

```
/// Return an array of metric instances
ClassMethod OnGetInstances(ByRef pInstSet As %String) As %Status
{
  Set pInstSet(1) = "Apple"
  Set pInstSet(2) = "Banana"
  Set pInstSet(3) = "Cherry"
  Quit $$$OK
}
```

9.3.2 動的なインスタンス名の定義

インスタンス名のセットを動的に定義するには、次のように、ビジネス・メトリック・クラスにクエリを追加します。

- クエリには MetricInstances() という名前を付ける必要があります。
- 引数は取らないようにする必要があります。
- クエリには SQL SELECT 文を含める必要があります。
- クエリから返される最初の列にインスタンス名が入っています。

以下に例を示します。

Class Member

```
/// Return current list of product names
Query MetricInstances() As %SQLQuery
{
  SELECT ProductName FROM MyApplication.Product
}
```

9.3.3 複数インスタンス・メトリック内のプロパティへの値の割り当て

複数インスタンス・ビジネス・メトリック内のプロパティに値を割り当てるには、OnCalculateMetrics() メソッドを実装します。この実装では、インスタンスごとに適切な値を設定します。そのためには、以下のように操作します。

1. %Instance プロパティの値をチェックします。このプロパティは、ビジネス・メトリック・インスタンスのいずれかの名前と一致します

   (InterSystems IRIS は、自動的に、一度に 1 つずつのインスタンスを順番に処理します。インスタンスごとに、OnCalculateMetrics() インスタンス・メソッドが実行されます)。
2. ビジネス・メトリック・プロパティの値をそのインスタンスの必要に応じて設定します。

以下に例を示します。

Class Member

```
Method OnCalculateMetrics() As %Status
{
  // get product name
  Set product = ..%Instance


  // find recent sales using SQL
  &SQL(SELECT SUM(Sales) INTO :sales
              FROM MyApplication.Product
              WHERE ProductName = :product)

  // update sales metric
  Set ..Sales = sales
  Quit $$$OK
}
```

この例では、SQL クエリ内の現在のインスタンス名 (%Instance) を使用して、そのインスタンスの最新の売上データを取得してから、それをクラスの現在のインスタンスの Sales プロパティに書き込みます。

注釈　%Instance プロパティは、現在のインスタンス名を置き換えたいとき、クラス内の別の場所で使用することもできます。

9.4 ビジネス・メトリック内のその他のオプション

この節では、ビジネス・メトリック・クラス内のその他のオプションについて説明します。

9.4.1 ダッシュボード内で使用するアクションの定義

ビジネス・メトリック・クラスでは、ダッシュボードにユーザ・オプションとして表示可能なアクションを定義できます。アクションはクライアント側のアクティビティ (ダッシュボードのフィルタリングや更新など) とサーバ側のアクティビティ (独自の API の呼び出しなど) の組み合わせを実行できます。アクション・メカニズムはごく一般的なものです。

アクションを定義するには、ビジネス・メトリック・クラスの %OnGetActionList() メソッドと %OnDashboardAction() メソッドを実装します。これらのメソッドの詳細は、"InterSystems Business Intelligence の実装" の "カスタム・アクションの定義" を参照してください。

9.4.2 OnInit() の実装

例えば、すべてのプロパティを初期化するように、ビジネス・メトリック・クラスの OnInit() コールバックを上書きすることもできます。そのためには、以下に示すように、スーパークラスの BusinessMetric から提供される OnInit() メソッドが明示的に呼び出されることを保証する必要があります。そうしないと、該当するダッシュボード要素が正しく表示されません。

Class Member

```
Method OnInit() As %Status
{
    // . . .

    // invoke superclass implementation
    Quit ##super()
}
```

9.5 ダッシュボードへのビジネス・メトリックの追加

ダッシュボードにビジネス・メトリックを追加するには、次の手順を実行します。

1. 他のビジネス・サービスを追加する場合と同じ方法で、適切なプロダクションにビジネス・メトリックを追加します。
2. ビジネス・メトリックごとに必要であれば、**[呼び出し間隔]** 設定を構成します。
3. ダッシュボードを作成して、それらにビジネス・メトリックを追加します。詳細は、"プロダクションの構成" を参照してください。
4. オプションで、ビジネス・メトリックからの情報が表示されるように [プロダクション・モニタ] ページを拡張します。次の節を参照してください。

9.6 プロダクション・モニタへのビジネス・メトリックの追加

ダッシュボードにビジネス・メトリックを表示することに加えて、ビジネス・メトリック・クラスからの情報を表示するように [プロダクション・モニタ] ページを拡張できます。そのためには、次のように、ネームスペース内の ^Ens.Monitor.Settings グローバルのノードを設定します。

ノード	値
^Ens.Monitor.Settings("MetricClasses",n,"Metric")	n 番目のビジネス・メトリックの構成名。[プロダクション・モニタ] ページには、ビジネス・メトリックが n で指定された順で表示されます。
^Ens.Monitor.Settings("MetricClasses",n,"Title")	このビジネス・メトリックの表示名。デフォルトは、ビジネス・メトリックの構成名です。
^Ens.Monitor.Settings("MetricClasses",n,"Instance")	このビジネス・メトリックのインスタンス名。メトリックにインスタンスが存在しない場合は、これを省略します。メトリックにインスタンスが存在せず、これを省略した場合は、定義された順で最初のインスタンスが使用されます。

例えば、ターミナルで以下を実行します。

ObjectScript

```
Set ^Ens.Monitor.Settings("MetricClasses",1,"Metric") = "MetricConfigName"
Set ^Ens.Monitor.Settings("MetricClasses",1,"Title") = "Title for Display"
Set ^Ens.Monitor.Settings("MetricClasses",1,"Instance") = "MetricInstanceName"
```

追加する各ビジネス・メトリックについて、[プロダクション・モニタ] ページには、そのメトリック情報の最終更新日時、そのメトリックやインスタンスのデータが存在するかどうか、およびそのメトリックが現在実行中であるかどうかが示されます。

[プロダクション・モニタ] ページの使用方法は、"プロダクションの監視" の "プロダクションの監視" を参照してください。

9.7 プログラムによる値の設定と取得

メトリック・プロパティにプログラムからアクセスしなければならない場合があります。例えば、メトリックのプロパティを直接読み込んだり、設定したりするために、ビジネス・プロセスが必要になる場合もあります。そのためには、Ens.BusinessMetric の GetMetric() クラス・メソッドと SetMetric() クラス・メソッドを使用します。

9.7.1 GetMetric() メソッドの使用法

GetMetric() クラス・メソッドは、ビジネス・メトリック・キャッシュから、指定されたメトリック・プロパティの現在値を読み取ります。このメソッドは次のように呼び出します。

ObjectScript

```
Set value = ##class(Ens.BusinessMetric).GetMetric(metric,property)
```

ここで、metric はビジネス・メトリックの名前 (クラス名ではなく、構成名) で、property はメトリック・プロパティの名前です。指定された値を GetMetric() で読み込めない場合は、空文字列が返されます。

多次元メトリックのプロパティの値を読み込む場合には、プロパティのどのサブノードを読み込むかを指定する、別のオプション・パラメータがあります。以下に例を示します。

ObjectScript

```
Set value(1) = ##class(Ens.BusinessMetric).GetMetric(metric,property,1)
```

9.7.2 SetMetric() メソッドの使用法

SetMetric() クラス・メソッドは、ビジネス・メトリック・キャッシュ内の指定されたメトリック・プロパティの値を設定します。このメソッドは次のように呼び出します。

```
Set tSC = ##class(Ens.BusinessMetric).SetMetric(metric,property,value)
```

ここで、metric はビジネス・メトリックの名前 (クラス名ではなく、構成名)、property はメトリック・プロパティの名前、value はメトリック・プロパティに設定する値です。

SetMetric() は、成功か失敗かを示す %Status コードを返します。そのため、SetMetric() が目的のビジネス・メトリックのロックを取得できずに失敗する場合があります。

多次元メトリック・プロパティのすべての値を設定するには、値の配列を作成してから、参照によって配列を渡します。以下に例を示します。

ObjectScript

```
For i=1:1:20 {
  Set data(i) = i*i
}
Set tSC = ##class(Ens.BusinessMetric).SetMetric("MyMetric","MyGraph",.data)
```

9.8 ビジネス・メトリック・キャッシュについて

InterSystems IRIS はできるだけ効率的にメトリック値を読み取ることができるように、その値をキャッシュに保存しています。このキャッシュが以下の構造を持つグローバル ^IRIS.Temp.EnsMetrics です。

```
^IRIS.Temp.EnsMetrics(Namespace,BusinessMetric,Instance,Property) = value
```

説明 :

- Namespace は、このメトリックが設定されたプロダクションが実行されるネームスペースです。
- BusinessMetric は、ビジネス・メトリックのプロダクション構成名です。
- Instance は、インスタンス番号です。インスタンスには、定義された順で番号が付けられます。
- Property は、ビジネス・メトリック・プロパティの名前です。

10

エンタープライズ・メッセージ・バンクの定義

このページでは、オプションで用意されているエンタープライズ・メッセージ・バンクと呼ばれる特殊なプロダクションの定義方法について説明します。

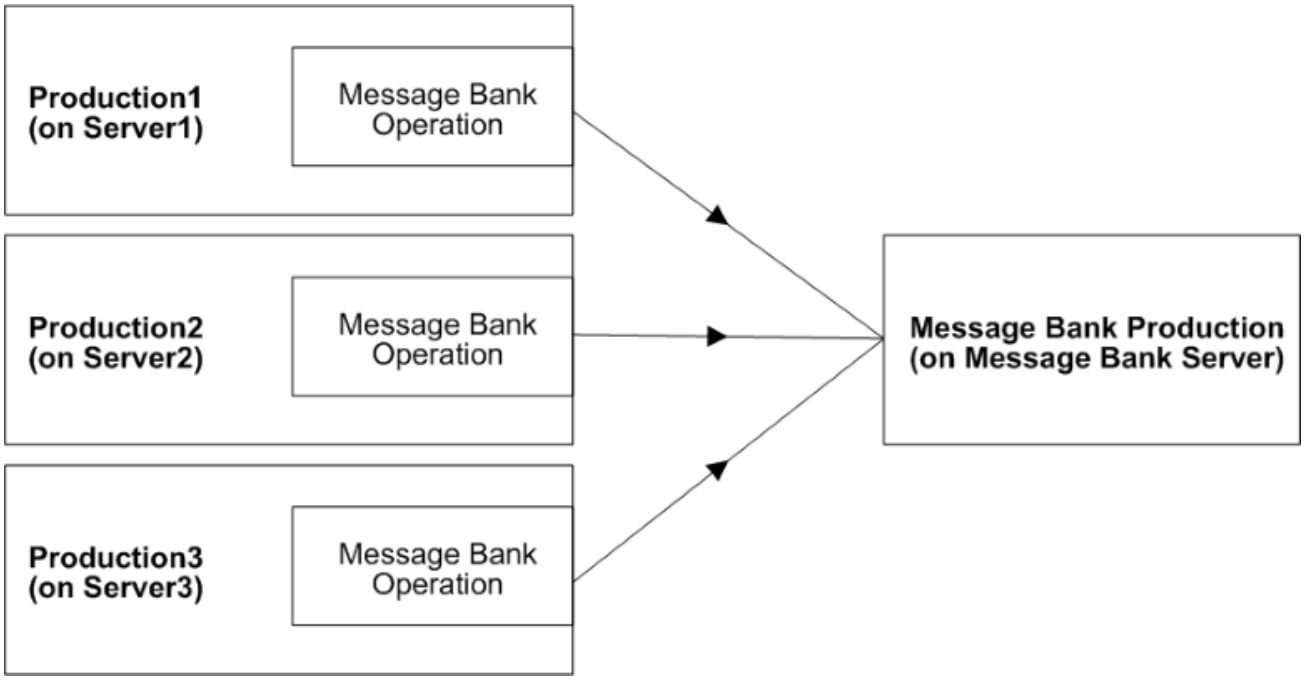
エンタープライズ・メッセージ・バンクの定義を完了した後、"エンタープライズ・メッセージ・バンクの構成"を参照してください。

10.1 概要

エンタープライズ・メッセージ・バンクはオプションのリモート・アーカイブ機能であり、複数のクライアント・プロダクションからメッセージ、イベント・ログの項目、および検索テーブルのエントリを収集することができます。これは、以下のコンポーネントから構成されます。

- メッセージ・バンク・サーバ。任意の数のクライアント・プロダクションからの送信を受信するメッセージ・バンク・サービスだけで成る単純なプロダクションです。
- プロダクションに追加して、メッセージ・バンク・サーバのアドレスで構成する、クライアント・オペレーション（メッセージ・バンク・オペレーション）。

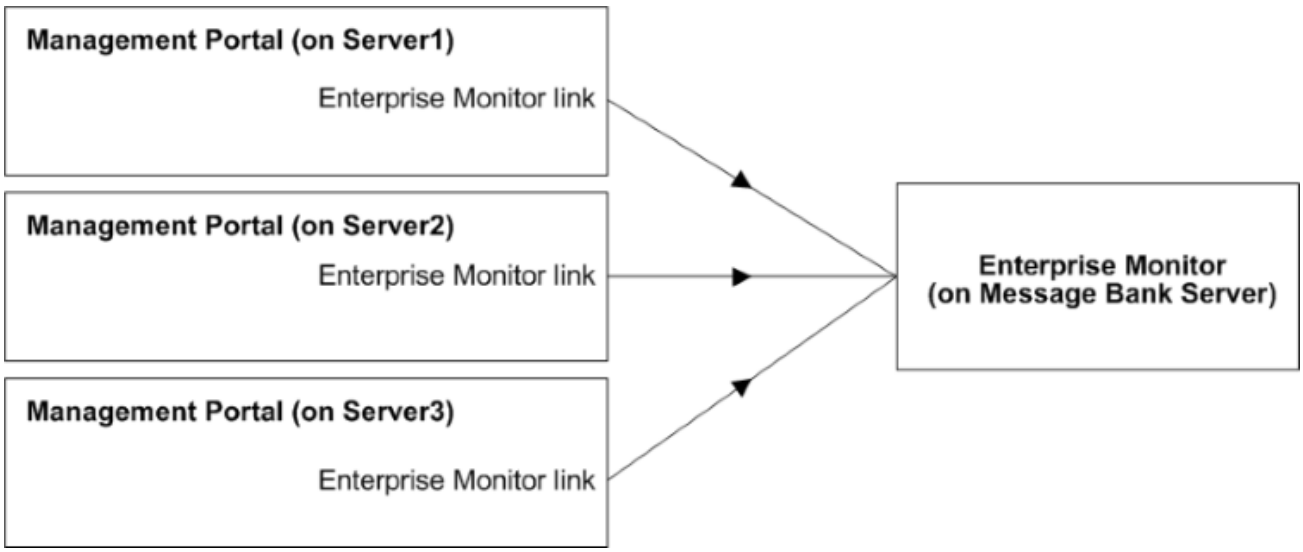
概念的な例を以下に示します。



メッセージ・バンクでメッセージを簡単に表示できるように、InterSystems IRIS® には以下の追加オプションが用意されています。

- メッセージ・バンク・インスタンスの場合は、管理ポータルに [エンタープライズモニタ] ページが自動的に追加されます。このページでは、クライアント・プロダクションのステータスを監視したり、メッセージ・バンクを参照したり、監視対象クライアントからメッセージの検索を実行したりできます。
- 各クライアントのインスタンスの場合、メッセージ・バンクのインスタンスにあるエンタープライズ・モニタへのリンクを構成します。

以下に例を示します。



10.2 メッセージ・バンク・サーバの定義

メッセージ・バンク・サーバを定義するには、相互運用対応ネームスペースにあるサーバ・マシンで以下を実行します。

1. Ens.Enterprise.MsgBank.Production 抽象クラスのサブクラスを作成します。
2. `ProductionDefinition` XData ブロックをクラスにコピーします。
3. 新しいクラスをコンパイルします。

以下に例を示します。

Class Definition

```
Class MyMessageBank Extends Ens.Enterprise.MsgBank.Production
{

XData ProductionDefinition
{
<Production Name="Ens.Enterprise.MsgBank.Production" TestingEnabled="false" LogGeneralTraceEvents="false">

  <Description>Production for receiving and collating message bank submissions from one or more client

   interoperability-enabled namespaces and for maintaining a local repository of production status
information about each
   client namespace, for display on the Enterprise Monitor page. Open the Monitor page on the same
   machine that that is hosting this Production.</Description>
  <ActorPoolSize>0</ActorPoolSize>
  <Setting Target="Production" Name="ShutdownTimeout">120</Setting>
  <Setting Target="Production" Name="UpdateTimeout">10</Setting>
  <Item Name="MonitorService" Category="" ClassName="Ens.Enterprise.MonitorService" PoolSize="1"
        Enabled="true" Foreground="false" InactivityTimeout="0" Comment="Populates global
        ^IRIS.Temp.Ens.EntMonitorStatus by polling namespaces from Systems List every CallInterval
seconds"
        LogTraceEvents="false" Schedule="">
    <Setting Target="Host" Name="AlertGracePeriod">0</Setting>
    <Setting Target="Host" Name="AlertOnError">0</Setting>
    <Setting Target="Host" Name="ArchiveIO">0</Setting>
    <Setting Target="Adapter" Name="CallInterval">10</Setting>
  </Item>
  <Item Name="MsgBankService" Category="" ClassName="Ens.Enterprise.MsgBank.TCPService" PoolSize="100"

        Enabled="true" Foreground="false" InactivityTimeout="20" Comment="" LogTraceEvents="true"
Schedule="">
    <Setting Target="Host" Name="AlertGracePeriod">0</Setting>
    <Setting Target="Host" Name="AlertOnError">0</Setting>
    <Setting Target="Host" Name="ArchiveIO">0</Setting>
    <Setting Target="Adapter" Name="Endian">Big</Setting>
    <Setting Target="Adapter" Name="UseFileStream">0</Setting>
    <Setting Target="Adapter" Name="JobPerConnection">1</Setting>
    <Setting Target="Adapter" Name="AllowedIPAddresses"></Setting>
    <Setting Target="Adapter" Name="QSize">100</Setting>
    <Setting Target="Adapter" Name="CallInterval">5</Setting>
    <Setting Target="Adapter" Name="Port">9192</Setting>
    <Setting Target="Adapter" Name="StayConnected">-1</Setting>
    <Setting Target="Adapter" Name="ReadTimeout">10</Setting>
    <Setting Target="Adapter" Name="SSLConfig"></Setting>
  </Item>
</Production>
}

}
```

このプロダクションには以下のサービスが含まれます。

- Ens.Enterprise.MsgBank.TCPService ― クライアント・プロダクションから受信データを受け取ります。このサービスの構成の詳細は、"サーバ上のメッセージ・バンク・サービスの構成" を参照してください。
- Ens.Enterprise.MonitorService ― クライアント・プロダクションからステータス情報を収集します。

このプロダクションは、このデータを受信して、ローカル・リポジトリを維持します。

テスト目的では、メッセージ・バンク・プロダクションを正規のプロダクションと同じマシンとインスタンス上に配置できますが、監視する予定のプロダクションとは別のネームスペースに入れる必要があります。

10.3 メッセージ・バンク・ヘルパ・クラスの追加

デフォルトでは、メッセージ本文をインデックス付けしていないため、検索できません。ヘルパ・クラスを追加すると、メッセージ・バンク内の検索機能を実装できます。そのためには、以下のように操作します。

1. Ens.Enterprise.MsgBank.BankHelperClass のサブクラスを作成し、OnBankMsg() メソッドを実装します。このメソッドの詳細は、"Intersystems クラス・リファレンス" を参照してください。

   OnBankMsg() メソッドは、カスタム処理を特定し、受信メッセージにメッセージ本文が含まれるときに実行されます。

   このメソッドの実装では、実行中にメッセージを処理し、必要に応じて入力容量を削減) するかどうか、または、メソッドが非同期でメッセージを二次プロセスに転送してより効率的に処理を分散するべきかどうかを判断しなければなりません。

2. メッセージ・バンクのプロダクションでは、Ens.Enterprise.MsgBank.TCPService ビジネス・サービスを選択し、[Bank ヘルパークラス] の設定を指定します。値には、ヘルパ・クラスの名前を使用します。

10.4 メッセージ・バンクに関する注意事項

メッセージ・バンクの以下の重要な特性に注意してください。

- メッセージ・バンクは、寄与しているプロダクションとのメッセージ・ボディ・クラスに関する同期依存性を持たないため、シリアル化された形式の各メッセージを受信します。仮想ドキュメント・メッセージ・ボディの場合も、メッセージ・バンクはシリアル化されたドキュメントをオブジェクトにリパースし、寄与しているプロダクションから検索テーブル・エントリを受信して格納します。

  メッセージ・バンク内をカスタム・スキーマのプロパティで検索できるようにするには、必ず、そのカスタム・スキーマ定義をメッセージ・バンク・プロダクションのネームスペースに格納することを確認してください。

- 一意性を確保するために、メッセージ・バンク・プロダクションは、クライアント・プロダクションの数値識別子をメッセージ ID の先頭に付加します。

- メッセージ・バンク・ページは、複数のクライアント・プロダクションの状態を表示したり、それらのクライアント・プロダクションに再送信サービスを起動するためのポータルとしても機能します。これらの追加機能が利用可能になるには、メッセージ・バンクがクライアント・プロダクションの Web アドレスを把握する必要があります。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。

- メッセージ・バンクのメカニズムでは、送信元プロダクションからのメッセージは削除されません。その機能は別の削除プロセスによって処理する必要があります。

- メッセージをメッセージ・バンクから送信元プロダクションまたは別のクライアント・プロダクションに再送信することも可能です。"プロダクションの監視" の "エンタープライズ・メッセージ・バンクの使用法" を参照してください。

11

レコード・マッパーの使用法

次の 2 つの技術文書では、複雑なレコード・マッピング (それぞれのレコードが一群の異種サブレコードで構成されている) と、バッチ処理 (複数のレコードをグループに分けて一括して処理する) について説明します。

- 複雑なレコード・マッパーの使用法
- レコードの効率的なバッチ処理

11.1 概要

レコード・マッパー ツールを使用すると、テキスト・ファイル内のデータを永続プロダクション・メッセージにマッピングしたり、元のデータに再びマッピングしたりする作業をすばやく効率的に実行できます。特に、管理ポータル・ユーザ・インタフェースでは、テキスト・ファイルの表現を視覚的に作成することによって、プロダクションの単一の永続メッセージ・オブジェクトにマッピングされる、そのデータの有効なオブジェクト表現を作成できます。ターゲット・オブジェクト構造と入出力パーサの両方を生成するプロセスは自動化されており、指定する必要があるのはオブジェクト・プロジェクションの永続構造に関する少数のオプションだけです。InterSystems IRIS® は単一の永続ツリーになるようにオブジェクトを生成するため、完全なカスケード式の削除操作が可能になります。

管理ポータルには、コンマ区切り形式 (CSV) のファイルをレコード・マップ構造に変換する **CSV ウィザード**もあります。このウィザードは、ヘッダ名を使用してサンプル・ファイル内の列に対応するレコード・マップ・プロパティを作成するため、特に、ファイルに列ヘッダが含まれている場合に有効です。

レコード・マッパーでは、区切りがあるか、または固定幅のフィールドを持つ単純なレコードが処理されます。レコード・マップは、レコード内のデータを識別する一連のフィールドと、フィールドを 1 つの構成単位に整理する複合で構成されます。区切りのあるレコードでは、複合の階層がフィールド間で使用されるセパレータを指定します。区切りのあるレコード内には、繰り返される単純なフィールドを含めることができます。繰り返される複合フィールドを含めることはできません。保存されたレコードでフィールドがスペースを無駄にしないように、必要に応じて受信テキスト・ファイルのフィールドを無視できます。

レコード・マッパーは、区切りのあるデータと固定幅のデータが混在したデータを処理できません。また、繰り返される単純なフィールドの処理以外に、受信レコードの内容に基づいてパーサまたはオブジェクト構造を動的に調整することもできません。

複雑なレコード・マッパーを使用すれば、さまざまなレコード・タイプを含む構造化レコードを処理できます。これには、繰り返しレコードを含む構造や区切りのあるレコードと固定フィールド・レコードの混在した構造を処理する能力も含まれます。

レコード・マッパー・パッケージの追加機能では、EnsLib.RecordMap.Batch の継承クラスを実装することによって、異種レコードをバッチ処理できます。このクラスは、特定のバッチに関連付けられたヘッダおよびトレーラの解析と書き出しを処理します。ヘッダおよびトレーラが単純な場合は、レコード・マッパー・ユーザ・インタフェースでバッチ・タイプ

EnsLib.RecordMap.SimpleBatch を作成できます。より複雑なヘッダおよびトレーラ・データを処理する必要がある場合は、これらの 2 つのバッチ実装のいずれかを拡張します。

RecordMap バッチ・オペレーションでは、個別のレコードからバッチを作成するときに、部分的に作成したバッチを中間ファイルに保存します。このファイルの場所は、バッチ・オペレーションの IntermediateFilePath プロパティを使用して指定できます。ミラー・システムでは、メイン・システムとフェイルオーバー・システムの両方にアクセスできるネットワーク・ドライブに、中間ファイルを保存できます。これにより、フェイルオーバーが発生した場合でも、フェイルオーバー・システムは引き続き未完了のバッチにレコードを追加できます。未完了のバッチは InterSystems IRIS データベースではなくファイルに保存されるため、ミラー・システムに自動的にコピーされることはありません。

11.2 レコード・マップの作成と編集

この節では、レコード・マップの作成方法と編集方法について説明します。この章は以下の節で構成されています。

- 概要
- はじめに
- 一般的な制御文字
- レコード・マップのプロパティの編集
- レコード・マップ・フィールドと複合の編集

11.2.1 概要

レコード・マップを作成するには、管理ポータルの **[レコード・マッパー]** ページを使用します。区切りのあるファイルの場合は、[CSV レコード・ウィザード] を使用して、プロセスをさらに自動化できます。レコード・マップの開発時には、サンプル・ファイルがレコード・マップにどのように表示されるかを確認できます。

重要　　レコード・マップ (CSVRecord マップも同様) の再生成では、生成済みコードの手動による変更が破棄されます。この原則に対応するために、生成されたパーサー・クラス・メソッド (GetObject、PutObject、GetRecord、PutRecord) にはコメント "DO NOT EDIT" が明確に付記されます。

[レコード・マッパー] ページには、レコード・マップ構造のビジュアル表現に加えて、レコード・マップ・コンポーネントで使用できる詳細な設定を入力および操作できる単純なインタフェースが含まれています。また、兄弟要素 (同じレベルにある要素) の配置を変更することもできます。このユーザ・インタフェースのより重要な機能の 1 つは、サンプル入力ファイルがある場合に、現在のレコード・マップの保存時にこのファイルのサンプル解析が試みられることです。レコード・マップ内の小さな問題には、管理ポータルから直接対処することができます。

XML とモデル・クラスを使用して、直接レコード・マップを作成することもできます。

11.2.2 はじめに

レコード・マッパーを起動するには、**[Interoperability]** > **[ビルド]** > **[レコード・マッパー]** を選択します。ここから、以下のコマンドを使用できます。

- **[開く]** – ファインダ・ダイアログ・ボックスが表示されます。このダイアログ・ボックスで、編集対象として開く既存のレコード・マップを選択できます。
- **[新規作成]** – ページが初期化され、新しいレコード・マップ構造を入力できるようになります。
- **[保存]** – 作業中のネームスペースに、クラスとしてレコード・マップ構造が保存されます。保存されると、オブジェクトはレコード・マップのリストに表示されます。

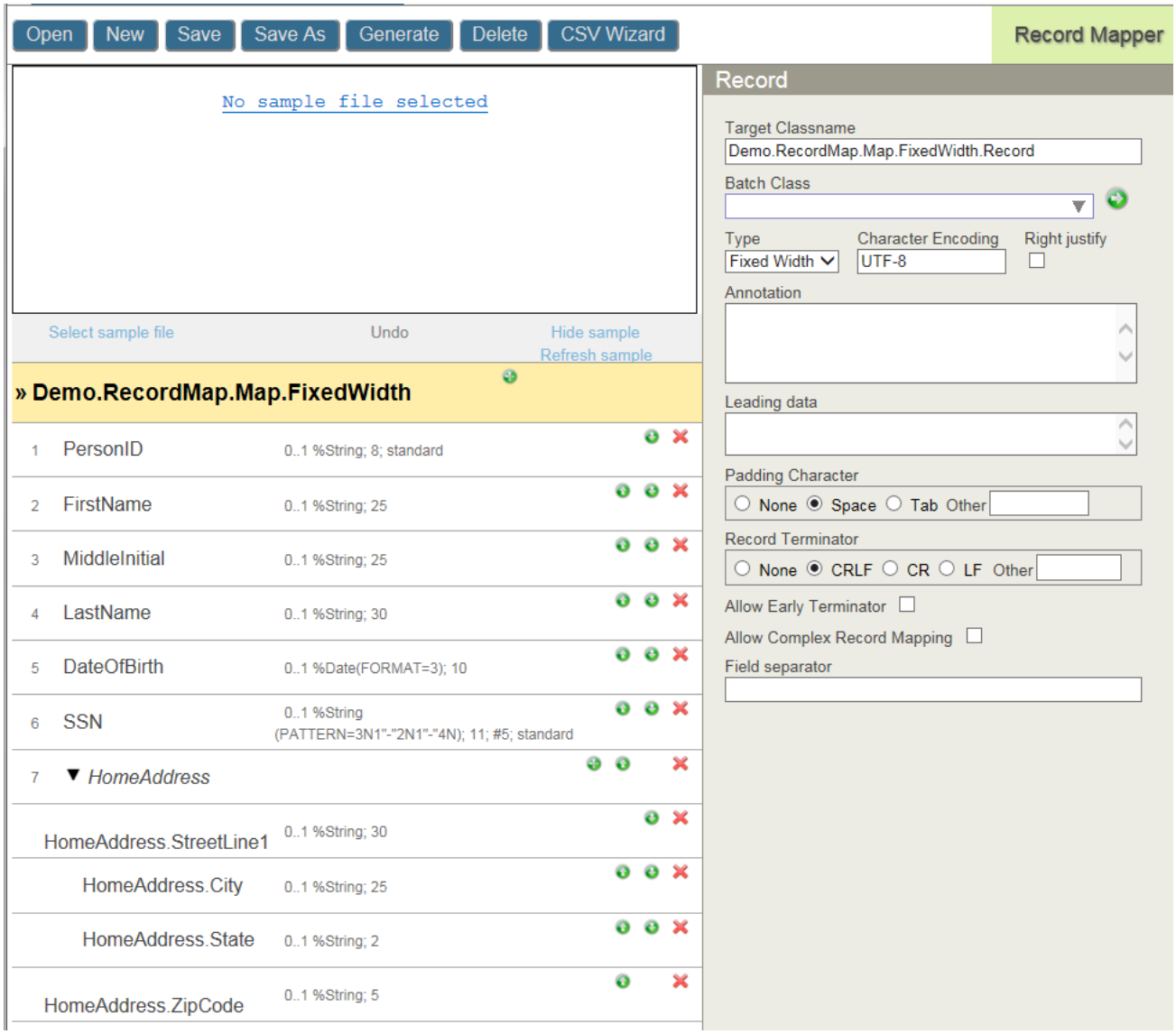
- **[名前をつけて保存]** – レコード・マップ構造が新しいクラスとして、作業中のネームスペースに保存されます。保存されると、オブジェクトはレコード・マップのリストに表示されます。
- **[生成]** – レコード・マップ・パーサ・コードと関連する永続メッセージ・レコード・クラス・オブジェクトが生成されます。

  オブジェクトを手動で生成するには、**EnsLib.RecordMap.Generator** の GenerateObject() クラス・メソッドを使用します。このメソッドでは、生成するオブジェクトの永続構造に関するいくつかのオプションを指定できます。これについては、このメソッドのコメントに記載があります。
- **[削除]** – 現在のレコード・マップが削除されます。オプションで、関連する永続メッセージ・レコード・クラスと保存されたすべてのクラスのインスタンスを削除できます。
- **[CSV ウィザード]** – カンマ区切り値 (CSV) を含むサンプル・ファイルからのレコード・マップの作成プロセスの自動化を支援する CSV レコード・ウィザードが開きます。

重要　**[保存]** 操作では、現在のレコード・マップのみがディスクに書き込まれます。対照的に、**[生成]** 操作では、基礎のオブジェクトに対してパーサ・コードと永続オブジェクト構造が生成されます。

ページの左側でレコード・マップ名を選択すると、右側にレコードの設定が表示され、そのレコード・マップのプロパティを編集できます。レコード・マップを保存する前に、レコード・マップに少なくとも 1 つのフィールドを追加する必要があります。以下の各節では、これらのプロセスについて説明します。

新しいレコード・マップを作成するか、または既存のレコード・マップを開くと、レコード・マッパーの左側のパネルにレコード・マップで定義したフィールドの要約が表示されます。右側のパネルでは、レコード・マップまたは選択したフィールドのプロパティを設定できます。サンプル・データ・ファイルを指定した場合は、左側のパネルの上に表示されます。例えば、以下は、右側のパネルにレコード・マップのプロパティが表示されたレコード・マッパーを示します。



レコード・マップをエクスポート、インポート、または削除するには、[Interoperability]、[リスト]、[レコード・マップ] の順にクリックして、[レコード・マップ・リスト] ページを表示します。

11.2.3 一般的な制御文字

1 つのレコード・マップ内の複数の場所で、リテラル制御文字だけでなく、印刷可能文字も使用できます。例えば、一般的な制御文字であるタブ文字に加えて、印刷可能文字であるカンマをセパレータとして指定できます。制御文字は、パディング文字またはレコード・ターミネータ文字の 1 つとして指定することもできます。これらのコンテキストのいずれかで制御文字を指定するには、その文字の 16 進エスケープ・シーケンスを指定する必要があります。レコード・マッパーで、パディング文字としてスペースまたはタブ文字を選択した場合やレコード・ターミネータ文字として CRLF (復改の後に行フィードが続く)、CR、または LF を選択した場合は、管理ポータルが自動的に 16 進表現を生成します。パディング文字としてまたはレコード・ターミネータ内で別の制御文字を指定する場合や、セパレータとして任意の制御文字を指定する場合は、対応するフォーム・フィールドに 16 進表現を入力する必要があります。以下の表に、一般的に使われている制御文字の 16 進のエスケープ・シーケンスを示します。

文字	16 進表記
タブ	\x09
行フィード	\x0A
復改	\x0D
スペース	\x20

その他の文字については、https://en.wikipedia.org/wiki/C0_and_C1_control_codes にアクセスするか、その他のリソースを参照してください。

注釈　RecordMap でレコード・ターミネータを指定する場合は、受信メッセージとそのレコード・ターミネータが正確に一致する必要があります。例えば、CRLF (¥x0D¥x0A) を指定する場合は、受信メッセージ・レコードとその順序が一致する必要があります。

11.2.4 レコード・マップのプロパティの編集

新しいレコード・マップのプロパティを入力する場合も、ウィザードで生成したマップから作業を始める場合も、また既存のマップを編集する場合も、プロセスは同じです。レコード自体に対して、以下のフィールドの値を入力または更新します。

[レコード・マップ名]

レコード・マップの名前。レコード・マップ名はパッケージ名で修飾する必要があります。パッケージ名を入力せず、非修飾レコード・マップ名を指定した場合は、デフォルトで、レコード・マップ・クラスがユーザ・パッケージ内に保存されます。

[ターゲット・クラス名]

このレコードを表すクラスの名前。デフォルトで、レコード・マッパーは、ターゲット・クラス名を、レコード・マップ名の後に “.Record” が付いた修飾名に設定しますが、このターゲット・クラス名は変更できます。 ターゲット・クラス名はパッケージ名で修飾する必要があります。パッケージ名を入力せず、非修飾ターゲット・クラス名を指定した場合は、デフォルトで、ターゲット・クラスがユーザ・パッケージ内に保存されます。

[バッチ・クラス]

このレコード・マップに関連付けるバッチ・クラスの名前 (存在する場合)。

[タイプ]

レコードのタイプ。以下のオプションがあります。

- [デリミタ区切り]
- [固定幅]

[文字エンコード]

インポートしたデータ・レコードの文字エンコード。

[右寄せ]

パディング文字をフィールド内のデータの左側に表示するように指定するフラグ。

[アノテーション]

このレコード・マップの目的と用途を記述するテキスト。

[先行データ]

実際のレコード・コンテンツのあらゆるデータの前に配置される静的文字。このレコード・マップを複雑なレコード・マップ内で使用している場合は、先行データが含まれたレコードを識別する必要があります。

[パディング文字]

値を埋めるために使用される文字。パディング文字はビジネス・サービスによって受信メッセージから削除され、ビジネス・オペレーションによって固定幅レコード・マップ内のフィールドを埋めるフィールド値として使用されます。

- [なし]
- [スペース]
- [タブ]
- [その他]

[レコード終端文字]

レコードの終端として使用される文字。

- [なし]
- [CRLF]
- CR
- [LF]
- [その他]

[早期ターミネータを許可 (固定幅レコード・マップのみ)]

レコードの終了前に終端が可能かどうかを指定するフラグ。可能な場合は、レコードがパディング文字で埋められているものとして処理されます。

[複雑なバッチ処理を許可]

レコード・マップを複雑なレコード・マップに使用できるかどうかを指定するフラグ。

[フィールド・セパレータ (固定幅レコード・マップのみ)]

固定幅フィールドをレコードに分割するために使用されるオプションの単一文字。指定された場合は、入力メッセージのフィールド間にこの文字を含める必要があり、ビジネス・オペレーションがフィールド間にこの文字を書き込みます。

[フィールド・セパレータ (区切りのあるレコード・マップのみ)]

フィールド・セパレータ文字のリスト。最初のセパレータがレコード内の最上位フィールドを区切ります。次のセパレータが最上位複合フィールド内のフィールドを区切ります。その他のセパレータがネストした複合フィールド内のフィールドを区切ります。

[繰り返し区切り文字 (区切りのあるレコード・マップのみ)]

繰り返されるすべてのフィールドで使用される 1 つの区切り文字。

[引用符処理 : なし (区切りのあるレコード・マップのみ)]

引用符スタイルのエスケープ処理がないことを指定するラジオ・ボタン。

[引用符エスケープ処理 (区切りのあるレコード・マップのみ)]

フィールド値内にセパレータ文字を出現可能にする引用符スタイルのエスケープ処理を有効にするラジオ・ボタン。入力フィールドは、引用符文字で囲むことができます。このフィールドでは、開始引用符と終了引用符の間のすべての文字が考慮されます。引用符内に出現するセパレータ文字は、リテラル文字として処理されます。セパレータとしては処理されません。出力では、値にセパレータを含むフィールドは、引用符文字で囲まれます。

[すべて引用符で囲む (区切りのあるレコード・マップのみ)]

フィールド値内にセパレータ文字を出現可能にする引用符スタイルのエスケープ処理を有効にするラジオ・ボタン。これには**引用符エスケープ**処理と同じ効果がありますが、出力のすべてのフィールドが、そこに区切り文字が使用されているかどうかに関係なく、引用符で囲まれる点が異なります。

[引用文字 (引用符エスケープ処理を含む区切りのあるレコード・マップのみ)]

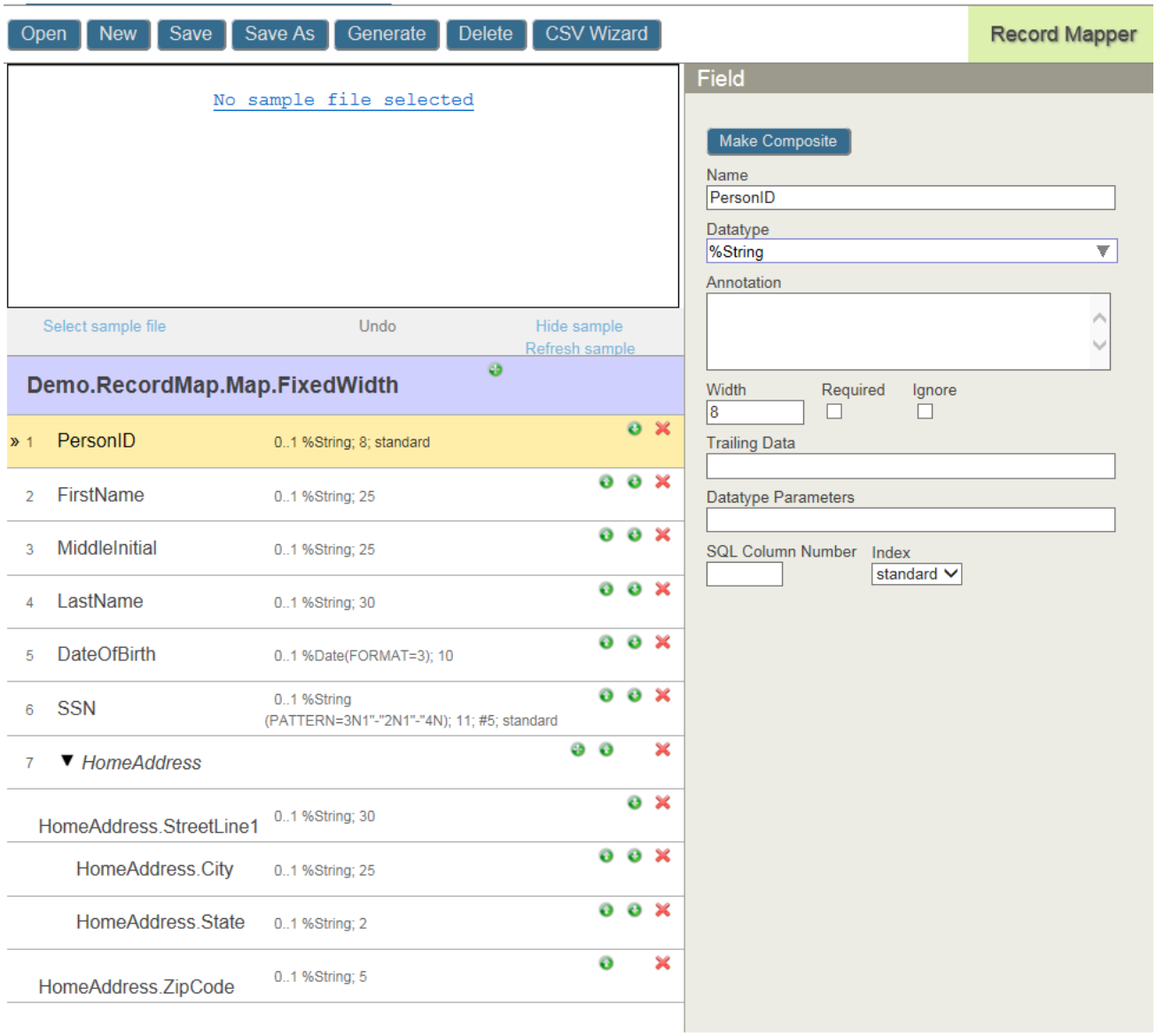
フィールド・コンテンツを囲む引用符として使用する文字。このフィールドは、**[引用符のエスケープ]** ラジオ・ボタンまたは **[すべてクォート]** ラジオ・ボタンを選択した場合に表示されます。引用符として制御文字を使用している場合、16 進数で入力する必要があります。″一般的な制御文字″ を参照してください。

[埋め込みレコードターミネータを許可]

引用符で囲まれたフィールドにレコード終端文字がある場合のレコード・マッパーの動作を指定します。チェックを付けると、レコード・マッパーによってレコード終端文字がエスケープ処理され、レコードの終了としてではなく、フィールド・データの一部として扱われます。

11.2.5 レコード・マップ・フィールドと複合の編集

レコード・マッパーの左側のパネルには、レコード・マップで定義したフィールドの要約が表示されます。フィールドを選択すると、右側のパネルにフィールドのプロパティが表示されます。以下に例を示します。



レコード・マップは、一連のフィールドと複合で構成されます。各複合は、一連のフィールドと複合で構成されます。**[複合の作成]** ボタンと **[フィールドの作成]** ボタンは複合フィールドとデータ・フィールドを切り替えます。複合フィールドでは、名前と、フィールドが必要なことを示すフラグを指定するだけです。レコード・マップ上の緑色のプラス記号アイコンをクリックすると、単一のフィールドまたは複合フィールドが最上位に追加されます。複合フィールドのプラス記号をクリックすると、そこに単一のフィールドまたは複合フィールドが追加されます。

レコード・マップへのフィールドの追加時には、サンプル・ファイルを開いて、作成するレコードにそのデータがどのようにマッピングされるかを確認できます。

区切りのあるレコード・マップでは、複合フィールド内のフィールドに異なるセパレータが使用されます。例えば、レコード内の最上位フィールドはカンマで区切られますが、複合内のフィールドはセミコロンで区切られます。固定幅レコード・マップでは、複合フィールドがデータを概念的に整理するのに役立ちますが、入力メッセージの処理には影響しません。

レコード・マッパーで複合フィールドを作成すると、複合フィールドによって、デフォルト名が複合構造と一致する修飾名として設定されます。修飾フィールド名によって、生成されたレコード・クラス内のフィールドの構造が決定されます。別の修飾名を含むようにフィールド名を変更しても、レコード・マップ内の複合フィールドのレベルは、生成されたレコード・クラス内のフィールドの構造に関係なく変化しません。

データ・フィールドごとに、以下のプロパティを入力します。

[名前]

フィールドの名前。

[データタイプ]

フィールドのデータ型。以下のリストから選択するか、カスタム・データ型を入力します。

- %Boolean
- %Date
- %Decimal
- %Double
- %Integer
- %Numeric
- %String
- %Time
- %Timestamp

[アノテーション]

このレコード・マップ内のフィールドの目的と用途を記述します。

[幅 (固定幅レコード・マップのみ)]

フィールドの幅。

[必須]

フィールドが必須であることを示すフラグ。

[繰り返し (区切りのあるレコード・マップのみ)]

レコード・マップの繰り返し区切り文字を使用して、フィールドに繰り返される値を含めることが可能なことを指定するフラグ。

[無視]

入力ではフィールドを無視し、保存されるレコードに含めないことを指定するフラグ。**[無視]** プロパティを使用すると、保存されたレコードのストレージ・スペースが節約されます。出力では、無視したフィールドに対して空の値が出力されます。固定幅のレコードでは、フィールドはスペースで埋められます。区切りのあるレコードでは、空のフィールドに連続する 2 つのセパレータが書き込まれます。

[後続データ (固定幅レコード・マップのみ)]

このフィールドの末尾に付ける必要がある文字。制御文字は 16 進で入力する必要があります。"一般的な制御文字" を参照してください。

[データタイプ・パラメータ]

セミコロンで区切られたデータ型に適用されるパラメータ。"一般的なプロパティ・パラメータ" を参照してください。

[SQL 列番号]

フィールドの SQL 列番号。この値は省略するか、SqlColumnNumber プロパティ・キーワードの値と同じように、2 ～ 4096 の範囲にする必要があります。SQL 表現を容易に複製できることから、列番号は CSV ファイルのデータまたは同様のデータ・ダンプをインポートするときに特に役立ちます。

[インデックス]

プロパティにインデックスを付けるかどうかを制御する列挙値。以下のいずれかを選択します。

- (空白) ー インデックスを作成しません。
- 1
- [bitmap]
- [idkey]
- [unique]

レコード・マッパーの左側のパネルは、フィールド定義の要約が入っているテーブルです。各列は以下を指定しています。

- 上位レベルのフィールド番号。
- フィールド名。
- フィールドのプロパティの要約。要約には、以下の情報が含まれており、; (セミコロン) で区切られています。
  - ignored ー **[無視]** チェック・ボックスにチェックが付いている場合、表示されます。
  - 0..1 または 1..1 と、それに続くデータタイプおよびデータタイプ・パラメータ ー それぞれ、オプションのフィールドか、必須のフィールドかを示します。
  - 固定幅のレコード・マップのフィールド幅。
  - #nnn ー SQL 列番号 (指定されている場合)。
  - standard、bitmap、idkey、または unique ー インデックスのタイプ (指定されている場合)。

  例えば、固定幅のレコード・マップの SSN フィールドの要約は `0..1 %String(PATTERN=3N1"-"2N1"-"4N); 11; #5; standard` になる場合があります。これは、このフィールドがオプションのフィールドで、データタイプおよびデータタイプ・パラメータが `%String(PATTERN=3N1"-"2N1"-"4N)`、フィールド幅が 11、SQL 列番号が 5、インデックスが標準インデックスであることを意味します。
- フィールドを上下に移動したり、フィールドを削除したりできるアイコン。複合フィールドでは、プラス・アイコンで新しいサブフィールドを追加できます。

11.3 CSV レコード・ウィザードの使用

InterSystems IRIS は、カンマ区切り値 (CSV) を含むサンプル・ファイルからのレコード・マップの作成プロセスの自動化を支援するウィザードを備えています。この CSV レコード・ウィザードを開始するには、InterSystems IRIS の **[ビルド]** サブメニューでこのウィザードを選択するか、または **[レコード・マッパー]** ページのリボン・バーにある **[CSV ウィザード]** をクリックします。ウィザードは、単一レベルのセパレータを含むファイルしか処理せず、先行データは処理しません。

ウィザードから、以下のフィールドの値を入力します。

[サンプル・ファイル]

サンプルの完全なパスとファイル名を入力するか、または [**ファイルを選択**] をクリックしてサンプル・ファイルの場所に移動し、ファイルを選択します。

[レコード・マップ名]

サンプル・ファイルから生成するレコード・マップの名前を入力します。

[区切り文字]

サンプル・ファイルで使用している区切り文字。制御文字は 16 進で入力する必要があります。"一般的な制御文字" を参照してください。

[レコード終端文字]

サンプル・ファイルでレコードを終了する方法を指定します。以下のいずれかを選択します。

- CRLF ー 各レコードは、後ろに行フィードが続く復改で終了します。
- CR ー 各レコードは、復改のみで終了します。
- LF ー 各レコードは、行フィードのみで終了します。
- その他 ー 各レコードは、制御文字で終了します。制御文字の値は 16 進で入力します。"一般的な制御文字" を参照してください。

[文字エンコード]

サンプル・ファイルで使用されている文字エンコードのタイプを選択します。

[ヘッダ行付きのサンプル]

使用するサンプル・ファイルにヘッダ行が含まれている場合は、このチェックボックスにチェックを付けます。

この場合は、ヘッダ行の句読点と空白がすべて削除され、その結果値がプロパティ名としてレコード・マップに使用されます (このオプションを選択しない場合、プロパティ名は `Property1`、`Property2`、などと指定されます)。

[SQL の列の順序を維持]

生成するオブジェクトで SQL の列の順序を維持する場合は、このチェックボックスにチェックを付けます。

[引用符形式のエスケープを使用]

サンプル・ファイルでセパレータの引用符スタイルのエスケープ処理が使用されている場合は、このチェック・ボックスにチェックを付けて、引用符文字を選択します。

ウィザード・フォームへの入力が完了したら、[RecordMap **の作成**] をクリックします。それにより、サンプル・ファイルから新しいレコード・マップが生成され、[**レコード・マッパー**] ページに戻ります。この後、レコード・マップの調整を行い、生成されたプロパティに詳細を追加します。詳細は、"レコード・マップのプロパティの編集" を参照してください。

11.4 レコード・マップ・クラスの構造

レコード・マップを記述するクラスが 2 つあります。

- レコードの外部構造を記述し、レコード・パーサとレコード・ライタを実装する RecordMap。

- データを含むオブジェクトの構造を定義する生成されたレコード・クラス。このオブジェクトを使用すれば、データ変換とルーティング・ルール条件内のデータを参照できます。

レコード・マップ・ビジネス・サービスは、受信データを読み取って解釈し、生成されたレコード・クラスのインスタンスであるメッセージを作成します。ビジネス・プロセスは、生成されたレコード・クラスのインスタンスを読み取り、変更、または生成することができます。最後に、レコード・マップ・ビジネス・オペレーションは、インスタンス内のデータを使用して、RecordMap を書式設定テンプレートとして送信データを書き込みます。RecordMap クラスと生成されたレコード・クラスは共に、データを記述する階層構造を持っていますが、生成されたオブジェクト構造を RecordMap 構造と同じにする必要はありません。

管理ポータルで新しいレコード・マップを作成してから保存すると、RecordMap クラスを拡張するためのクラスが定義されます。生成されたレコード・クラスを定義するには、管理ポータルで **[生成]** をクリックする必要があります。これにより、EnsLib.RecordMap.Generator クラス内の GenerateObject() メソッドが呼び出されます。RecordMap クラス定義をコンパイルするだけでは、生成されたレコード・クラス用のコードは作成されません。管理ポータルを使用するか、ターミナルまたはコードから Generator.GenerateObject() メソッドを呼び出す必要があります。

RecordMap は、一連のフィールドと複合で構成されます。

- フィールドは、指定されたタイプのデータ・フィールドを定義します。フィールド・タイプは、VALUELIST、MAXVAL、MAXLEN、FORMAT などのパラメータを指定できます。固定幅レコードでは、レコード・マッパーがフィールド幅を使用して、MAXVAL パラメータまたは MAXLEN パラメータのデフォルト値を設定します。
- 複合は、一連のフィールドと複合で構成されます。また、複合は RecordMap 内でネストすることができます。

デフォルトで、管理ポータル内のレコード・マッパーは、複合レベルを使用してフィールドの修飾名を設定します。区切りのあるレコードでは、次のように、複合要素のネスト・レベルによって、フィールド間で使用されるセパレータが決定します。

1. 複合内に含まれない RecordMap 内のフィールドは、最初のセパレータで区切られます。
2. RecordMap 内に存在する複合内に出現するフィールドは、2 つ目のセパレータで区切られます。
3. それ自体が複合内に存在する複合内に出現するフィールドは、3 つ目のセパレータで区切られます。
4. 複合のその他のネスト・レベルでは、フィールドを区切るためのセパレータが順にインクリメントされます。

固定幅レコード内の複合は、データ構造を記述したものですが、InterSystems IRIS によるメッセージの処理方法に影響しません。

各 RecordMap オブジェクトは、対応するレコード・オブジェクト構造を持っています。RecordMap を生成すると、レコード・マップのオブジェクト表現を規定したレコード・オブジェクトが定義され、コンパイルされます。デフォルトで、管理ポータル内のレコード・マッパーが “Record” レコードを RecordMap の名前で修飾しますが、**[ターゲット・クラス名]** フィールドで明示的にレコード・オブジェクトの名前を設定することができます。また、デフォルトで、レコード・マッパーは、複合内のフィールドをそれが含まれる複合の名前で修飾します。デフォルト修飾名を使用した場合は、レコード・オブジェクト・クラス・プロパティの構造が RecordMap フィールドと複合の構造と一致しますが、フィールドに他の名前を割り当てた場合は、レコード・オブジェクト・クラス・プロパティの構造が RecordMap フィールドと複合の構造と一致しません。

レコード・オブジェクト・クラスは、EnsLib.RecordMap.Base、%Persistent、%XML.Adaptor、および Ens.Request の各クラスを拡張します。既存のクラスの RECORDMAPGENERATED パラメータが 0 の場合は、ターゲット・クラスがレコード・マップ・フレームワークによって変更されません。したがって、すべての変更をプロダクション開発者が行う必要があります。生成されたレコード・クラス内のプロパティは、レコード・マップ内のフィールドの名前によって異なります。

レコード・オブジェクト・クラスのプロパティは、レコード・マップのフィールドに対応しており、以下の名前と型を持っています。

- RecordMap またはその中の複合内の任意の場所に出現する単純な非修飾名を含むフィールドの名前。これらのプロパティにはフィールドの型によって決定された型が設定されます。

- RecordMap またはその中の複合内の任意の場所に出現する修飾名を含むフィールドの最上位名。これらのプロパティには、同じ最上位修飾名を共有するフィールドによってクラスが定義されたオブジェクト型が設定されます。これらのクラスは、%SerialObject クラスと %XML.Adaptor クラスを拡張します。また、これらのクラスは、生成されたレコード・クラス名のスコープ内で定義されます。さらに、これらのクラスには、次のレベルの名前修飾に対応したプロパティが含まれています。

データに 3 つのレベルのセパレータが含まれている区切りのあるレコード・マップを定義しているとします。最上位セパレータ・フィールドが人に関する情報を区切り、次のレベルが識別番号、名前、および電話番号に関する情報を区切り、最後のレベルが住所と名前に含まれる要素を区切ります。例えば、メッセージの先頭を次のようにすることができます。

```
French Literature,TA,199-88-7777;Jones|Robert|Alfred;
```

これらのセパレータを処理する RecordMap を定義するには、人のレベルの複合と名前のレベルの複合が必要になります。つまり、FamilyName フィールドのデフォルト・フィールド名を Person.Name.FamilyName にすると、レコード・オブジェクト・クラス内に深いレベルのクラス名が作成されます。例えば、クラス NewRecordMap.Record.Person.Name には、NewRecordMap.Record.Person.Name.FamilyName などのプロパティが含まれます。フィールド名の先頭に $ (ドル記号) 文字を付けることによってこの深いレベルを回避できます。こうすることによって、クラスとプロパティのすべてがレコード・スコープ内で直接定義されます。同じ例を使用すると、クラス NewRecordMap.Record.Name に NewRecordMap.Record.FamilyName のようなプロパティが含まれることになります。

注釈　フィールド名の修飾に使用される名前は、オブジェクト型でプロパティを定義するために使用されます。そのため、フィールド名を修飾するための名前をフィールド名の最後の部分に使用することはできません。これは、データ型と名前が同じプロパティが定義されるためです。

11.5 RecordMap 構造のオブジェクト・モデル

クラス構造の動作は、XML を直接作成するか、EnsLib.RecordMap.Model.* クラスを使用して RecordMap のオブジェクト・プロジェクションを作成することによって実現できます。一般に好まれるアプローチでは管理ポータルが使用されますが、状況によっては、オブジェクト・モデルを使用して RecordMap 構造を作成する方が都合がいい場合もあります。これらのクラスの構造は RecordMap クラスの構造に従います。このレベルにおける詳細は、クラスリファレンスを参照してください。

11.6 プロダクションでのレコード・マップの使用

[レコード・マッパー] ページでオブジェクト・クラスの生成を選択した場合は、プロダクションのビジネス・サービスで使用できるクラスを作成します。

12

複雑なレコード・マッパーの使用法

メッセージ形式が複数の異種レコードで構成されている場合は、複雑なレコード・マッパーと、組み込みファイルまたはFTP サービスおよびオペレーションを使用して、これらの複雑なレコードを処理できます。これらの複雑なレコードの多くは、ヘッダ・レコードの後ろにレコードのパターンが続き、トレーラ・レコードで終わっています。これらのレコードは、固定フィールド・レコードまたは区切りのあるレコードのどちらにすることも、省略したり、繰り返したりすることもできます。通常は、レコードにはレコードの種類を識別する先行データが含まれています。

より簡潔なレコードとするには、プレーンなレコード・マッパー・ツールを使用します。"レコードの効率的なバッチ処理"も参照してください。

12.1 概要

複雑なレコード・マップは、以下を含む構造化レコードを記述することができます。

1. オプションのヘッダ・レコード。
2. 各要素を RecordMap またはシーケンスによって定義されたレコードにすることができる要素のシーケンス。シーケンスには、一連のレコードと他のシーケンスを含めることができます。
3. オプションのトレーラ・レコード。

シーケンス内のレコードは、区切りのあるレコードまたは固定幅レコードのどちらにもすることができます。複雑なレコード・マップ内では区切りのあるレコードと固定幅レコードを混在させることができますが、通常は、すべてのレコードが区切られるか、すべてのレコードが固定幅になるかのどちらかです。

以下に示す区切りのあるサンプル・データは、複雑なレコード・マップで記述することができます。このデータは、大学の学期を識別するヘッダと、学生と各学生が受講するクラスに関する情報で構成されています。

```
SEM|194;2012;Fall;20
STU|12345;Adams;John;Michael;2;john.michael.adams@example.com;617-999-9999
CLS|18.034;1;Differential Equations;4
CLS|21W.759;1;Writing Science Fiction;4
STU|12346;Adams;Jane;Michelle;3;jane.michelle.adams@example.com;
CLS|21L.285;1;Modern Fiction;3
CLS|7.03;1;Genetics;4
STU|12347;Jones;Robert;Alfred;1;bobby.jones@example.com;
CLS|18.02;1;Calculus;4
```

このデータを記述する複雑なレコード・マップは、以下で構成されます。

1. 先行データの “SEM|” で識別されるヘッダ・レコード。
2. 学生のシーケンス。ここでは、各学生が以下で構成されます。

a. 先行データの “STU|” で識別される学生レコード。

b. 先行データの “CLS|” で識別される繰り返しクラス・レコード。

学生のシーケンスは、複雑なレコードの繰り返し構造を定義したものですが、データ内のレコードには対応していません。

複雑なレコード・マップは、ファイル構造とオブジェクト構造の両方を定義します。複雑なレコード・マップ・ファイル・サービスは、複雑なレコード・マップによって定義されたファイル構造を使用してファイルを解釈してから、そのデータをオブジェクト構造によって定義されたオブジェクトに保存します。複雑なレコード・マップ・ファイル・オペレーションは、その逆を実行します。つまり、オブジェクト内のデータを取得して、それを複雑なレコード・マップによって定義されたファイル構造を使用してファイルに書き出します。

12.2 複雑なレコード・マップの作成と編集

管理ポータルで複雑なレコード・マッパーを使用するか、XML 定義をインポートして IDE で編集することで、新しい複雑なレコード・マップの作成や既存のレコード・マップの編集ができます。

複雑なマップには、フィールドを定義可能な最上位シーケンスが含まれています。複雑なレコードが、シーケンスが繰り返されない単一のレコード・シーケンスで構成されている場合は、レコードのレコード・マップを直接、複雑なレコード・マップに入力できます。ただし、ヘッダとトレーラ間でシーケンス全体を繰り返すことができる場合は、繰り返しシーケンス・レコードを最上位シーケンスの唯一の要素として入力する必要があります。

12.2.1 はじめに

管理ポータルから複雑なレコード・マッパーにアクセスするには、[Interoperability]、[**ビルド**]、[**複雑なレコード・マップ**] の順にクリックします。ここから、以下のコマンドを使用できます。

- [**開く**] – 既存の複雑なレコード・マップを開きます。
- [**新規作成**] – 新しい複雑なレコード・マップを作成します。
- [**保存**] – 複雑なレコード・マップ構造を、[**複雑な RecordMap 名**] で指定されたパッケージ内の作業中のネームスペース内の 1 つのクラスとして保存します。
- [**生成**] – 複雑なレコード・マップ・パーサ・コードと関連する永続メッセージの複雑なレコード・クラス・オブジェクトを生成します。

  オブジェクトを手動で生成するには、EnsLib.RecordMap.ComplexGenerator 内の Generate() クラス・メソッドを使用します。
- [**削除**] – 現在の複雑なレコード・マップを削除します。オプションで、関連する永続メッセージの複雑なレコード・クラスと保存されたすべてのクラスのインスタンスを削除できます。

重要　[**保存**] 操作では、現在の複雑なレコード・マップのみがディスクに書き込まれます。対照的に、[**生成**] 操作では、基礎のオブジェクトに対してパーサ・コードと永続オブジェクト構造が生成されます。複雑なレコード・マップを生成すると、生成済みのパーサー・コードに対する手動での変更が破棄されます。この原則に対応するために、生成されたクラスには、(ヘッダ、フッタ、バッチも対象として) コメント ″DO NOT EDIT″ が明確に付記されます。

12.2.2 複雑なレコード・マップ・プロパティの編集

新しい複雑なレコード・マップのプロパティを入力する場合も、ウィザード生成マップから作業を始める場合も、既存のマップを編集する場合も、プロセスは同じです。複雑なレコード自体に対して、以下のフィールドの値を入力または更新します。

[複雑な RecordMap 名]

複雑なレコード・マップの名前。複雑なレコード・マップ名はパッケージ名で修飾する必要があります。パッケージ名を入力せず、非修飾の複雑なレコード・マップ名を指定しなかった場合は、デフォルトで、複雑なレコード・マップ・クラスがユーザ・パッケージ内に保存されます。

[ターゲット・クラス名]

複雑なレコードを表すクラスの名前。 デフォルトで、複雑なレコード・マッパーは、ターゲット・クラス名を、複雑なレコード・マップ名の後に ".Batch" が付いた修飾名に設定しますが、このターゲット・クラス名は変更できます。ターゲット・クラス名はパッケージ名で修飾する必要があります。パッケージ名を入力せず、非修飾ターゲット・クラス名を指定した場合は、デフォルトで、ターゲット・クラスがユーザ・パッケージ内に保存されます。

[文字エンコード]

インポートしたデータ・レコードの文字エンコード。複雑なレコード・マップに対する指定は、複雑なレコード・マップに含まれているすべてのレコード・マップのエンコーディングと同じにする必要があります。これらが異なる場合は、複雑なレコード・マップの文字エンコーディングの方が、レコード・マップの文字エンコーディングより優先されます。

[アノテーション]

この複雑なレコード・マップの目的と用途を記述するテキスト。

新しい複雑なレコード・マップを作成すると、複雑なレコード・マッパーによって、以下の要素で構成された定義が作成されます。

- 複雑なマップの名前と型 – 複雑なマップの名前とクラスを入力します。
- ヘッダ – 複雑なレコードにヘッダが含まれている場合は、ヘッダを記述するレコード・マップの名前とクラスを入力します。
- トレーラ – 複雑なレコードにトレーラが含まれている場合は、トレーラを記述するレコード・マップの名前とクラスを入力します。

12.2.3 複雑なレコード・マップのレコードとシーケンスの編集

複雑なレコード・マップは以下で構成されます。

1. オプションのヘッダ・レコード。
2. 要素のシーケンス。ここでは、各要素を以下にすることができます。
   - RecordMap によって定義されたレコード。複雑なレコード・マップ内に以下のプロパティを含めることができます。
     - 必須項目。値 0 はレコードが省略可能なことを意味し、値 1 はレコードが必須であることを意味します。
     - 最小回数と最大回数の繰り返し。
   - ネストした要素のシーケンス。
3. オプションのトレーラ・レコード。

各レコードはレコード・マップによって定義されます。シーケンスは、複雑なレコード・マップ定義内で定義されます。これは、メッセージ内のデータの構造を記述しますが、それ自体がデータ内のフィールドに対応しているわけではありません。

ヘッダ・レコードとトレーラ・レコードはそれぞれレコード・マップによって定義されます。複雑なレコード・マップ定義にヘッダ・レコードまたはトレーラ・レコードを含めるかどうかは任意ですが、定義にヘッダ・レコードが含まれている場合は、データにヘッダ・レコードを含める必要があり、定義にトレーラ・レコードが含まれている場合は、データにトレーラ・レコードを含める必要があります。ヘッダ・レコードとトレーラ・レコードは繰り返すことができません。

すべてのシーケンスに 1 つ以上のレコードまたはシーケンスを含める必要があります。

レコードを編集しているときに、**[シーケンスの作成]** ボタンをクリックすると、レコードをシーケンスに置き換えることができます。シーケンスを編集しているときに、**[レコードの作成]** ボタンをクリックすると、シーケンスをレコードに置き換えることができます。

レコードに対して以下のプロパティを指定できます。

- レコード名。
- レコード形式を定義する RecordMap。RecordMap は、レコード、レコードが固定列と区切りのいずれを含んでいるか、セパレータ、およびレコードのターミネータを定義する **[先行データ]** を指定します。RecordMap の定義方法の詳細は、″レコード・マッパーの使用法″ を参照してください。
- レコードが必須かどうか。
- レコードを繰り返すことができるかどうか。レコードを繰り返すことができる場合は、以下も指定できます。
  - 最小繰り返し回数
  - 最大繰り返し回数
- 複雑なレコード・マップ内のレコードの目的と用途を記述するアノテーション。

シーケンスに対して以下のプロパティを指定できます。

- シーケンス名。
- シーケンスが必須かどうか。
- シーケンスを繰り返すことができるかどうか。シーケンスを繰り返すことができる場合は、以下も指定できます。
  - 最小繰り返し回数
  - 最大繰り返し回数
- 複雑なレコード・マップ内のシーケンスの目的と用途を記述するアノテーション。

12.3 複雑なレコード・マップ・クラスの構造

レコード・マップを記述する 2 つのクラスと同様の方法で複雑なレコード・マップを記述する 2 つのクラスがあります。この複雑なレコード・マップを記述する 2 つのクラスを以下に示します。

- 複雑なレコードの外部構造を記述し、複雑なレコード・パーサと複雑なレコード・ライタを実装する複雑なレコード・マップ。
- データを含むオブジェクトの構造を定義する生成された複雑なレコード・クラス。このオブジェクトを使用すれば、データ変換とルーティング・ルール条件内のデータを参照できます。

複雑なレコード・マップ・ビジネス・サービスは、受信データを読み取って解釈し、生成されたレコード・クラスのインスタンスであるメッセージを作成します。ビジネス・プロセスは、生成された複雑なレコード・クラスのインスタンスを読み取り、変更、または生成することができます。最後に、複雑なレコード・マップ・ビジネス・オペレーションは、インスタンス内のデータを使用して、複雑なレコード・マップを書式設定テンプレートとして送信データを書き込みます。複雑なレコード・マップ・クラスと生成された複雑なレコード・クラスの両方がデータを記述する階層構造を持っています。また、複雑なレコード・マップ・クラスと生成されたレコード・クラスは並列構造を持っています。これは、生成されたレコード・クラスが別の階層構造を持つことが可能な RecordMap クラスと違う点です。

管理ポータルで新しい複雑なレコード・マップを作成してから保存すると、EnsLib.RecordMap.ComplexMap クラスと Ens.Request クラスを拡張するためのクラスが定義されます。生成されたレコード・クラスを定義するには、管理ポータルで [生成] をクリックする必要があります。これにより、EnsLib.ComplexGenerator クラス内の Generate() メソッドが呼び出されます。ComplexMap クラス定義をコンパイルするだけでは、生成されたレコード・クラス用のコードは作成されません。管理ポータルを使用するか、ターミナルまたはコードから ComplexGenerator.Generate() メソッドを呼び出す必要があります。生成されたクラスは、RecordMap.ComplexBatch クラスと Ens.Request クラスを拡張します。

ComplexMap クラスは、RecordReference 要素によって指定されたレコードと RecordSequence 要素によって定義されたシーケンスを使用して ComplexBatch を定義する XData 定義内の複雑なレコード構造を定義します。既存のクラスの RECORDMAPGENERATED パラメータが 0 の場合は、ターゲット・クラスが複雑なレコード・マップ・フレームワークによって変更されません。したがって、すべての変更をプロダクション開発者が行う必要があります。

ComplexBatch クラスには、複雑なマップ定義内の以下の最上位要素に対応するプロパティが含まれています。

- 指定された場合のヘッダ・レコード。このプロパティには、指定されたレコード・マップ用の生成されたレコード・クラスに設定された型が設定されます。
- 指定されたレコード・マップ用の生成されたレコード・クラスに設定された型を持つレコード。または、レコードを繰り返すことが可能な場合は、型は生成されたレコード・クラスの配列に設定されます。
- シーケンス用に定義されたクラスに設定された型を持つシーケンス。または、シーケンスを繰り返すことが可能な場合は、型はそのクラスの配列に設定されます。
- 指定された場合のトレーラ・レコード。このプロパティには、指定されたレコード・マップ用の生成されたレコード・クラスに設定された型が設定されます。

クラスはシーケンスごとに定義されます。シーケンス・クラスは、ComplexSequence クラスと %XML.Adaptor クラスを拡張します。また、シーケンス・クラスは、ComplexBatch クラスに対して定義されたパッケージとネームスペース内で定義されます。すべてのシーケンス・クラスが、他のシーケンス内に含まれている場合でもこのレベルのネームスペース内で定義されます。

各シーケンスには、その中のレコードとシーケンスに対応するプロパティが含まれています。

12.4 プロダクションでの複雑なレコード・マップの使用

複雑なレコードを使用するプロダクションを作成するには、次の手順を実行します。

1. ヘッダとトレーラを含めて、複雑なレコードの部分ごとにレコード・マップを作成します。個別のレコード・マップの作成方法は、"レコード・マッパーの使用法"を参照してください。サンプル・ファイルを使用する場合は、個別のレコード・マップ内で定義している複雑なレコードの部分のみを含むサンプル・ファイルを作成する必要があることに注意してください。サンプル・ファイルには複雑なレコード全体を含めないようにする必要があります。
2. 複雑なレコード・マッパーを使用して、複雑なレコードの構造を定義します。
3. プロダクションを作成して、1 つ以上の組み込みの複雑なレコード・サービスおよびオペレーションを追加します。
4. プロダクションがアプリケーション間で複雑なレコードを受け渡すだけの場合は、単純なルーティング・エンジン・プロセスを使用できます。しかし、プロダクションが複雑なレコードを別の複雑なレコードに変換する場合は、ルーティ

ング・エンジン内にデータ変換を作成することになります。入力の複雑なレコードと出力の複雑なレコードの両方が同じ構造の場合は、ソース・フィールドとターゲット・フィールドをつなげる単純なデータ変換を作成できます。例えば、単純なデータ変換を使用して、区切りのあるレコードを含む複雑なレコードを、固定列レコードを含む複雑なレコードに変換することができます。ただし、入力の複雑なレコードと出力の複雑なレコードの構造が異なる場合は、データ変換とビジネス・プロセス言語 (BPL) プロセス内のどちらかにコードを追加する必要があります。

13

レコードの効率的なバッチ処理

RecordMap 機能は、一度に 1 つのレコードをインポートしますが、大量のレコードをインポートまたはエクスポートする場合は、RecordMap バッチを使用することで大幅に効率を高めることができます。RecordMap バッチ機能は同種のレコードを取り扱い、バッチのすべてのレコードを一度に処理します。バッチにはオプションで、先行するヘッダ・レコードと後続のトレーラ・レコードを含めることができます。

RecordMap バッチを作成するには、%Persistent および EnsLib.RecordMap.Batch から継承されるクラスを実装します。Batch クラスには、特定のバッチに関連付けられたヘッダおよびトレーラの解析と書き出しを処理するメソッドが含まれています。ヘッダを解析および書き出すためのコードを用意する必要があります。シンプルなヘッダとトレーラの場合は、EnsLib.RecordMap.SimpleBatch クラスを使用できます。このクラスは Batch クラスを継承し、シンプルなヘッダとトレーラを処理するためのコードを提供します。より複雑なヘッダおよびトレーラ・データを処理する必要がある場合は、これらの 2 つのバッチ実装のいずれかを拡張します。

バッチ処理では、X12 のような、プロダクションの他のメッセージ形式に使用されるアプローチに従って処理が行われます。これは特に、RecordMap バッチ・オブジェクトを処理する組み込みのビジネス・オペレーションに関連します。このようなビジネス・オペレーションは、 EnsLib.RecordMap.Base を拡張する RecordMap オブジェクトまたはバッチ・オブジェクト、あるいは BatchRolloverRequest タイプの要求のどちらかを受け入れます。特定のバッチのレコードを受け取ると、バッチが開かれ、バッチ・ヘッダが一時ファイルに書き込まれます。その後、オペレーションが受け取ったそのバッチ内のあらゆるオブジェクトが書き込まれます。同期的な要求の場合は、クラス名、ID、およびそのバッチに対してこれまでに書き込まれたレコード数が EnsLib.RecordMap.BatchResponse で返されます。バッチ・オブジェクト (これはデフォルトのバッチである場合があります) の受信によって、一時ファイルへのバッチ・トレーラの書き込みが開始され、その後そのビジネス・オペレーションのアダプタによってこのファイルが必要な宛先に送信されます。バッチ・オブジェクト自体を受け取った場合は、そのバッチ全体が一時ファイルに書き出され、その後このファイルが必要な場所に転送されます。

重要　　このプロセスでアーカイブ IO が有効な場合、受信ストリームのコピーはネームスペースの一時的なストリーム位置に保存されます (既定の位置は <install-dir>/mgr/GLOBAL_DB_DIRECTORY/stream/ です)。これらのストリームは自動的に削除され、手動で削除する必要はありません。

バッチ・オペレーションでは、デフォルト・バッチ・オプションもサポートしています。この場合、バッチにまだ属していないレコードがデフォルト・バッチに追加されます。このバッチの出力は、バッチ・オブジェクトをオペレーションに送るか、オペレーションに BatchRolloverRequest を送ることによって開始できます。デフォルト・バッチに対してスケジュール・ベースまたはカウント・ベースのロールオーバを使用するように、ビジネス・オペレーションを構成することもできます。これらのオプションはいずれもビジネス・オペレーションで構成します。それぞれのオプションは同時に使用できます。

サービスに関するオプションは、主に、ビジュアル・トレースでバッチ内のメッセージを表示する方法に影響します。

RecordMap バッチ・オペレーションは、最終的な出力ファイルの生成プロセスで一時ファイルを作成します。これらの一時ファイルの場所は、RecordMap バッチ・オペレーションの IntermediateFilePath 設定を指定することで制御できます。ネームスペースのデータベースがミラーリングされている場合は、RecordMap バッチ・オペレーション時に正常にフェイルオーバーできるように、すべてのミラー・メンバが一時ファイルにアクセスできることが重要です。ミラーリングの詳細は、"高可用性ガイド" を参照してください。

重要　RecordMap バッチ・メッセージでは一対多のリレーションシップが使用されているので、すべてのレコードを検索対象として目的の変換を容易に実現できます。ただし、このプロセスでは多くのメモリが消費されるので、〈STORE〉エラーが発生することがあります。プロセスで使用するメモリの増強、入力ファイルの分割、一対多のリレーションシップではなくSQLを使用するようにカスタマイズした変換の実装などが必要になることがあります。

14

あまり一般的ではないタスク

このページでは、あまり一般的ではない開発タスクについて説明します。

14.1 カスタム・ユーティリティ関数の定義

InterSystems IRIS® には、ビジネス・ルールと DTL から呼び出すことが可能なユーティリティ関数のセットが用意されています。これらの関数については、"ビジネス・ルールの開発" の "プロダクションで使用するユーティリティ関数" で説明します。独自の関数を追加することができ、ビジネス・ルール・エンジンとビジネス・ルール・エディタは自動的に拡張に適応します。

新しいユーティリティ関数を追加するには:

1. Ens.Rule.FunctionSet のサブクラスである新しいクラスを作成します。このクラスは Ens.Rule.FunctionSet 以外のスーパークラスを拡張してはなりません。
2. 定義する関数ごとに、クラス・メソッドを新しい関数セット・クラスに追加します。ポリモフィズムはサポートされていないため、正確さを期するには、これらのクラス・メソッドを最終的なものとしてマーク付けする必要があります。これは、既存の Ens.Util.FunctionSet メソッドで表示できます (Ens.Util.FunctionSet は Ens.Rule.FunctionSet のスーパークラスです)。
3. 新しいクラスをコンパイルします。これでルールの式で使用する新しい関数を利用できるようになりました。これらの関数を呼び出すには、サブクラスから `ClassMethod` の名前を使用します。Ens.Rule.FunctionSet で定義された関数と異なり、ユーザ定義のメソッド名は、それが属するクラスを含めて完全修飾で指定する必要があります。管理ポータル内のウィザードから名前を選択してメソッドを追加すると、自動的に完全修飾名で指定されます。

以下の関数セット・クラスでは、ビジネス・ルールで使用する日付と時刻の関数の例を示します。このクラス・メソッド DayOfWeek() および TimeInSeconds() で ObjectScript 関数の $ZDATE および $PIECE を呼び出して、ObjectScript の特殊変数 $HOROLOG から目的の日付と時刻の値を抽出します。

Class Definition

```
/// Time functions to use in rule definitions.
Class Demo.MsgRouter.Functions Extends Ens.Rule.FunctionSet
{

/// Returns the ordinal position of the day in the week,
/// where 0 is Sunday, 1 is Monday, and so on.
ClassMethod DayOfWeek() As %Integer [ CodeMode = expression, Final ]
{
$zd($H,10)
}

/// Returns the time as a number of seconds since midnight.
ClassMethod TimeInSeconds() As %Integer [ CodeMode = expression, Final ]
{
$p($H,",",2)
}

}
```

ObjectScript で使用可能な関数および特殊変数の完全なリストは、"ObjectScript リファレンス" を参照してください。

このトピックで説明があるように新しい関数を追加した場合、それを参照するための構文は組み込み関数の構文と多少異なります。以下のような Ens.Rule.FunctionSet から継承されるクラスの関数を定義すると仮定します。

```
ClassMethod normalizaSexo(value as %String) as %String
```

このクラスをコンパイルした後に、ルーティング・ルール・エディタやデータ変換ビルダなどの InterSystems IRIS Interoperability のビジュアル・ツールのいずれかを使用すると、関数の選択ボックスには、[Strip]、[In]、[Contains] などの組み込み関数と共に [normalizaSexo] という関数名が表示されます。

関数の選択ボックスから組み込み関数を選択して、生成されたコードを確認するとします。その結果、InterSystems IRIS がダブルドットのメソッド呼び出し構文を使用してその関数の呼び出しを生成したこと (DTL 内)、または単にその関数を名前で参照したことがわかります (ビジネス・ルール内) (これらの構文規則については、"ビジネス・ルールの開発" の "プロダクションで使用するユーティリティ関数" で説明します)。

次に、ダブルドット構文で組み込みの Strip 関数を参照する DTL の <assign> 文の例を示します。

XML

```
<assign property='target.cod'
        value='..Strip(source.{PatientIDExternalID.ID},"<>CW")'
        action='set'/>
```

ただし、独自のユーザ定義関数を作成する場合、DTL の構文は異なります。単に関数を指定するだけでは不十分で、関数のクラス・メソッドが含まれるクラスの完全なクラス名も指定する必要があります。独自の関数 normalizaSexo が HP.Util.funciones というクラスで定義されたと仮定します。その場合、関数の選択ボックスから関数を選択して生成されたコードを確認すると、以下の例のようなコードが表示されます。

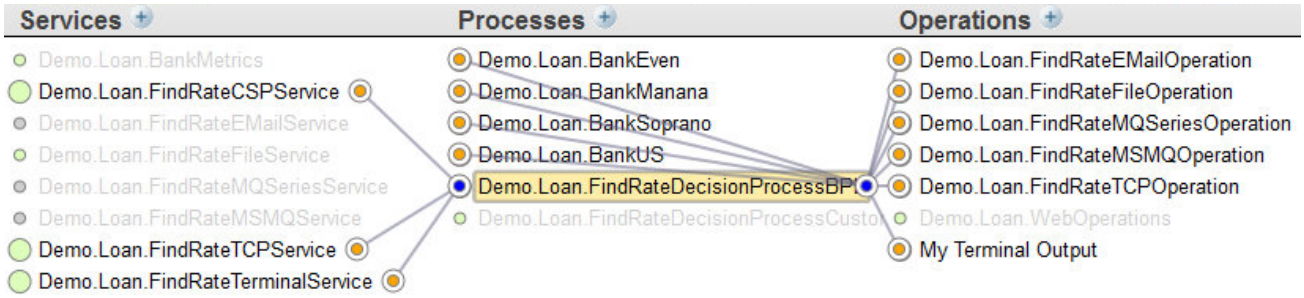
XML

```
<assign property='target.sexo'
        value='##class(HP.Util.funciones).normalizaSexo(source.{Sex})'
        action='set'/>
```

データ変換ビルダを使用してコードを生成するのではなく、このように直接 DTL コードに文を入力する場合は、この構文の違いを認識する必要があります。

14.2 ターゲットが動的な場合の接続のレンダリング

管理ポータルには、ユーザがビジネス・ホストを選択したときに、自動的に、特定のビジネス・ホストとの接続状態が表示されます。以下に例を示します。



これを実現するために、InterSystems IRIS がビジネス・ホストの構成設定を読み取って使用します。

ただし、ビジネス・サービス・ホストが実行時にそのターゲットを動的に決定する場合は、InterSystems IRIS がその接続を自動的に表示することはできません。この場合に、接続を表示するには、OnGetConnections() コールバック・メソッドを実装します。InterSystems IRIS は、構成ダイアグラムをレンダリングするときに、自動的に、このメソッド (デフォルトでは何もしない) を呼び出します。

注釈　ビジネス・ホストが接続に Ens.DataType.ConfigName またはそのサブクラスのいずれかを使用する場合、その接続は自動的にレンダリングされ、OnGetConnections() は必要ありません。

OnGetConnections() のシグニチャを以下に示します。

```
ClassMethod OnGetConnections(Output pArray As %String, item As Ens.Config.Item) [ CodeMode = generator
 ]
```

引数は以下のとおりです。

- pArray ー このビジネス・サービスからのメッセージ送信先となるアイテムに構成された名前を添え字とする多次元配列。例えば、それぞれ ABC と DEF を名前とするビジネス・ホストにメッセージを送信するには、次のようにコードに pArray を設定します。

  ```
  set pArray("ABC")=""
  set pArray("DEF")=""
  ```

- item ー このビジネス・サービスを示す Ens.Config.Item オブジェクト。

オーバーライドされた OnGetConnections() メソッドの例は、IDE を使用して、X12 などの Electronic Data Interchange (EDI) プロトコルで使用するために提供されている組み込みビジネス・サービスを確認してください。これらについては、"プロダクション内での X12 ドキュメントのルーティング" で詳しく説明されています。

14.3 Ens.Director の使用によるプロダクションの開始および停止

開発中は、管理ポータルを使用してプロダクションを開始または停止するのが一般的です。導入済みで稼働中のプロダクションでは、"プロダクションの構成" で説明されている自動開始オプションの使用をお勧めします。

プロダクションが定義されているネームスペースでプログラムからプロダクションを開始または停止するオプションもあります。そのためには、Ens.Director クラス内の以下のメソッドを呼び出します。

StopProduction()

現在実行中のプロダクションを停止します。

ObjectScript

```
Do ##class(Ens.Director).StopProduction()
```

StartProduction()

実行中のプロダクションが他にない限り、指定されたプロダクションを開始します。

ObjectScript

```
Do ##class(Ens.Director).StartProduction("myProduction")
```

RecoverProduction()

実行中のプロダクションで問題発生インスタンスをクリーンアップし、同じネームスペースで新しいインスタンスを実行できるようにします。

ObjectScript

```
Do ##class(Ens.Director).RecoverProduction()
```

RecoverProduction() を呼び出す前に、プロダクションが異常終了したかどうかを確認する目的で GetProductionStatus() を呼び出す必要はありません。プロダクションに問題が発生していない場合、このメソッドは何も実行せずに返ります。

GetProductionStatus()

このメソッドは、参照で渡される 2 つの出力パラメータを使用して、プロダクションのステータスを返します。それらのパラメータの 1 つは、ステータスが [実行中]、[中断中]、または [問題発生] の場合のみ、プロダクション名を返します。もう 1 つのパラメータは、プロダクションの状態を返します。これは次の定数のいずれかに相当する数値です。

- `$$$eProductionStateRunning`
- `$$$eProductionStateStopped`
- `$$$eProductionStateSuspended`
- `$$$eProductionStateTroubled`

以下に例を示します。

ObjectScript

```
Set tSC=##class(Ens.Director).GetProductionStatus(.tProductionName,.tState)
Quit:$$$ISERR(tSC)
If tState'=$$$eProductionStateRunning {
  $$$LOGINFO($$$Text("No Production is running.")) Quit
  }
```

一般的なクラスやルーチンなどの InterSystems IRIS プロダクション・クラスの外部で、`$$$eProductionStateRunning` など、プロダクションの状態を確認するマクロをコードに記述できます。このためには、目的のクラスに以下の文を追加する必要があります。

```
#include Ensemble
```

これをビジネス・ホストなどのプロダクション・クラスの内部で行う必要はありません。

Ens.Director には、InterSystems IRIS 内部フレームワーク専用のものを含む多くのクラス・メソッドが用意されています。このドキュメントに記述されている Ens.Director メソッドのみを、その説明のとおりに使用することをお勧めします。

注釈　^%ZSTART ルーチンを使用してプロダクションの起動を制御しないことをお勧めします。InterSystems IRIS の起動メカニズムの方がはるかに使いやすく、プロダクション自体に密接に関連付けられています。

14.4 Ens.Director の使用による設定へのアクセス

以下の各 Ens.Director クラス・メソッドでは、プロダクションを実行していなくても、そのプロダクションの設定を取得できます。

GetAdapterSettings()

特定された構成項目 (ビジネス・サービスまたはビジネス・オペレーション) のすべてのアダプタ設定の値を含む配列を返します。配列には、設定名の添え字が付けられます。InterSystems IRIS $ORDER 関数を使用して配列の要素へアクセスできます。このメソッドの 1 番目のパラメータは、プロダクション名と設定項目名を 2 本の垂直バー (||) で区切って記述した文字列です。戻り値はステータス値です。ステータス値が $$$OK ではない場合、指定したプロダクション名 (myProd) と設定項目名 (myOp) の組み合わせが見つからなかったことを示しています。

ObjectScript

```
Set tSC=##class(Ens.Director).GetAdapterSettings("myProd||myOp",.tSettings)
```

GetAdapterSettingValue()

特定された構成項目 (ビジネス・サービスまたはビジネス・オペレーション) の指定されたアダプタ設定の値を返します。1 番目のパラメータは、プロダクション名と設定項目名を 2 本の垂直バー (||) で区切って記述した文字列です。2 番目のパラメータは構成設定の名前です。3 番目の出力パラメータは、呼び出しで得られたステータス値を返します。以下に例を示します。

ObjectScript

```
Set val=##class(Ens.Director).GetAdapterSettingValue("myProd||myOp","QSize",.tSC)
```

返されたステータス値が $$$OK ではない場合、指定したプロダクション名 (myProd) と設定項目名 (myOp) の組み合わせが見つからなかったか、指定したプロダクションと設定項目の設定には指定した名前の設定 (QSize) が見つからなかったことを示しています。

GetCurrProductionSettings()

現在実行中のプロダクションまたは最後に実行されたプロダクションのすべてのプロダクション設定の値を含む配列を返します。配列には、設定名の添え字が付けられます。このメソッドの戻り値はステータス値です。ステータス値が $$$OK ではない場合、現在のプロダクションを識別できなかったことを示しています。

ObjectScript

```
Set tSC=##class(Ens.Director).GetCurrProductionSettings(.tSettings)
```

GetCurrProductionSettingValue()

現在実行中のプロダクションまたは最後に実行されたプロダクションの、指定されたプロダクション設定の文字列値を返します。2 番目の出力パラメータは、呼び出しで得られたステータス値を返します。このステータス値が $$$OK ではない場合、指定した名前の設定が現在のプロダクションの設定に見つからなかったか、現在のプロダクションを識別できなかったことを示しています。

ObjectScript

```
Set myValue=##class(Ens.Director).GetCurrProductionSettingValue("mySet",.tSC)
```

GetHostSettings()

特定された構成項目 (ビジネス・サービス、ビジネス・プロセス、またはビジネス・オペレーション) のすべての設定の値を含む配列を返します。配列には、設定名の添え字が付けられます。このメソッドの 1 番目のパラメータは、プロダクション名と設定項目名を 2 本の垂直バー (||) で区切って記述した文字列です。戻り値はステータス値です。ステータス値が $$$OK ではない場合、指定したプロダクション名 (myProd) と設定項目名 (myOp) の組み合わせが見つからなかったことを示しています。

ObjectScript

```
Set tSC=##class(Ens.Director).GetHostSettings("myProd||myOp",.tSettings)
```

GetHostSettingValue()

特定された構成項目 (ビジネス・サービス、ビジネス・プロセス、またはビジネス・オペレーション) の指定された設定の値を返します。1 番目のパラメータは、プロダクション名と設定項目名を 2 本の垂直バー (||) で区切って記述した文字列です。2 番目のパラメータは構成設定の名前です。3 番目の出力パラメータは、呼び出しで得られたステータス値を返します。以下に例を示します。

ObjectScript

```
Set val=##class(Ens.Director).GetHostSettingValue("myProd||myOp","QSize",.tSC)
```

返されたステータス値が $$$OK ではない場合、指定したプロダクション名 (myProd) と設定項目名 (myOp) の組み合わせが見つからなかったか、指定したプロダクションと設定項目の設定には指定した名前の設定 (QSize) が見つからなかったことを示しています。

GetProductionSettings()

指定されたプロダクションのすべてのプロダクション設定の値を含む配列を返します。配列には、設定名の添え字が付けられます。このメソッドの戻り値はステータス値です。ステータス値が $$$OK ではない場合、指定したプロダクションが見つからなかったことを示しています。

ObjectScript

```
Set tSC=##class(Ens.Director).GetProductionSettings("myProd",.tSettings)
```

GetProductionSettingValue()

指定されたプロダクションの指定されたプロダクション設定の値を返します。3 番目の出力パラメータは、呼び出しで得られたステータス値を返します。このステータス値が $$$OK ではない場合、指定したプロダクションが見つからなかったか、指定した名前の設定が、指定したプロダクションの設定に見つからなかったことを示しています。

ObjectScript

```
Set val=##class(Ens.Director).GetProductionSettingValue("prod","set",.tSC)
```

Ens.Director には、InterSystems IRIS 内部フレームワーク専用のものを含む多くのクラス・メソッドが用意されています。このドキュメントに記述されている Ens.Director メソッドのみを、その説明のとおりに使用することをお勧めします。

14.5 ビジネス・サービスの直接呼び出し

言語バインディング、CSP ページ、SOAP、またはオペレーティング・システム・レベルで呼び出されるルーチンなどの他のメカニズムで作成されたジョブから、ビジネス・サービスを直接呼び出す場合があります。これを実行できるのは、ADAPTER クラス・パラメータの値が NULL の場合だけです。この種のビジネス・サービスは、アダプタ不要型ビジネス・サービスと呼ばれています。

ビジネス・サービスを機能させるには、そのビジネス・サービス・クラスのインスタンスを作成する必要があります。%New() メソッドを呼び出してこのインスタンスを作成することはできません。代わりに、Ens.Director の CreateBusinessService() メソッドを使用する必要があります。以下に例を示します。

ObjectScript

```
Set tSC = ##class(Ens.Director).CreateBusinessService("MyService",.tService)
```

プロダクションは、起動時に、このビジネス・サービスにジョブを割り当てません。これは、[**プールサイズ**] の設定が 0 であることが想定されているためです。

CreateBusinessService() メソッドは、以下を実行します。

1. プロダクションが実行中であること、およびそのプロダクションが所定のビジネス・サービスを定義していることを確かめます。
2. 所定のビジネス・サービスが現在有効であることを確かめます。
3. ビジネス・サービスの構成名の名前解決を行い、正しい設定値を使って正しいビジネス・サービス・オブジェクトをインスタンス化します (プロダクションが、別の名前と設定を持つ同一のビジネス・サービス・クラスを使って、多くのビジネス・サービスを定義している可能性もあります)。

CreateBusinessService() メソッドが成功すると、ビジネス・サービス・クラスのインスタンスを参照によって返します。これに続けて、その ProcessInput() メソッドを直接呼び出すことができます。ProcessInput() メソッドに対し、このメソッドが期待する入力オブジェクトのインスタンスを提供する必要があります。以下に例を示します。

```
If ($IsObject(tService)) {
  Set input = ##class(MyObject).%New()
  Set input.Value = 22
  Set tSC = tService.ProcessInput(input,.output)
}
```

Ens.Director には、InterSystems IRIS 内部フレームワーク専用のものを含む多くのクラス・メソッドが用意されています。このドキュメントに記述されている Ens.Director メソッドのみを、その説明のとおりに使用することをお勧めします。

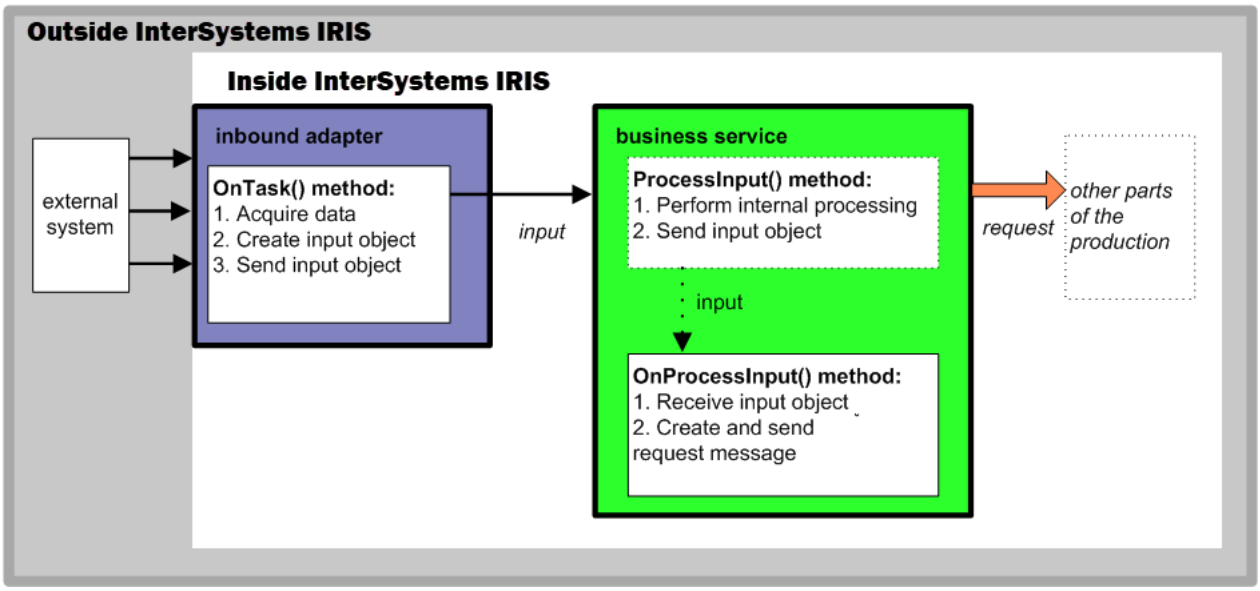
14.6 受信アダプタの作成またはサブクラス化

この節では、受信アダプタを作成またはサブクラス化する方法について説明します。

14.6.1 受信アダプタの概要

受信アダプタは、外部システムからの要求を受信し、検証します。

受信アダプタ・クラスは、ビジネス・サービス・クラスとの組み合わせで機能します。一般に、受信アダプタには汎用目的で再利用可能なコードが含まれていますが、ビジネス・サービスにはプロダクション固有のコード（特別な検証ロジックなど）が含まれています。通常は、InterSystems IRIS の組み込みアダプタ・クラスのいずれかを使用して受信アダプタ・クラスを実装します。下の図は、プロダクションでどのように受信要求が受け付けられるかを示しています。



一般に、外部アプリケーションが特定の処理の実行を要求すると、上の図のように、その要求は受信アダプタを経由して InterSystems IRIS 内に入ってきます。この要求する側のアプリケーションは、クライアント・アプリケーションと呼ばれます。プロダクションに何かをするように依頼したためです。このアプリケーションはプロダクションのクライアントです。この段階で機能する要素が受信アダプタです。受信アダプタは、クライアントのネイティブな要求形式をプロダクションが理解できる形式に変換するアダプタとして機能するコードの一部です。プロダクションに要求を出す各アプリケーションは、自分自身で受信アダプタを備える必要があります。クライアント・アプリケーションのコードを変更する必要はありません。それは、アダプタが、クライアント・アプリケーションでネイティブな呼び出しを処理するためです。

14.6.2 受信アダプタの定義

受信アダプタ・クラスを作成するには、次のようにクラスを作成します。

- クラスは、Ens.InboundAdapter（またはサブクラス）を拡張する必要があります。
- クラスは OnTask() メソッドを実装する必要があります。これについては、"OnTask() メソッドの実装" で説明します。
- クラスは設定を定義できます。"設定の追加と削除" を参照してください。
- クラスは任意のまたはすべてのスタートアップ・メソッドおよびティアダウン・メソッドを実装できます。"開始動作と停止動作の上書き" を参照してください。
- クラスにはプロダクション資格情報を追加できます。"アダプタ・クラスへの資格情報の追加" を参照してください。
- クラスにはその内部で作業を完了するためのメソッドを含めることができます。

以下に例を示します。

Class Definition

```
Class MyProduction.InboundAdapter Extends Ens.InboundAdapter
{

Parameter SETTINGS = "IPAddress,TimeOut";

Property IPAddress As %String(MAXLEN=100);
```

```
Property TimeOut As %Integer(MINVAL=0, MAXVAL=10);

Property Counter As %Integer;

Method OnTask() As %Status
{
  #; First, receive a message (note, timeout is in ms)
  Set msg = ..ReceiveMessage(..CallInterval*1000,.tSC)

  If ($IsObject(msg)) {
    Set tSC=..BusinessHost.ProcessInput(msg)
  }

  Quit tSC
}

}
```

14.6.3 OnTask() メソッドの実装

OnTask() メソッドでは、受信アダプタの処理が実際に行われます。このメソッドの用途を以下に示します。

1. Ensemble が受け取るイベントのチェック。受信アダプタは、この処理をさまざまな方法で行うことができます。例えば、Ensemble が受け取る I/O イベント (TCP ソケットの読み取りなど) を待機する方法や、外部データ (ファイルなど) が存在するかどうかを定期的にポーリングする方法などが考えられます。

   ほとんどの事前構築受信アダプタには、OnTask() の呼び出し間隔を制御する **CallInterval** という設定があります。このアプローチも使用できます。

2. このイベントから受け取った情報を、ビジネス・サービス・クラスが期待する型のオブジェクトにパッケージ化します。
3. ビジネス・サービス・オブジェクトの ProcessInput() メソッドを呼び出します。
4. 必要に応じて、イベントを受け取った外部システムに対して通知を返します。
5. さらに入力データがある場合は、OnTask() で以下のいずれかを実行できます。
   - すべてのデータが取得されるまで、ProcessInput() を繰り返し呼び出します。
   - 複数の入力イベントが存在する場合でも、**CallInterval** ごとに 1 回だけ ProcessInput() を呼び出します。

受信アダプタを設計する際は、OnTask() メソッドはビジネス・サービスに対して定期的にコントロールを返す必要があるということ、つまり、OnTask() メソッドは受信イベントをいつまでも待ち続けてはならないということに注意してください。いつまでも待つ代わりに、例えば 10 秒間だけ待機して、それからビジネス・サービスに制御を返します。ビジネス・サービス・オブジェクトは InterSystems IRIS 内のイベントを定期的にチェックしなければならないというのがその理由です。プロダクションの終了通知はこのイベントの例です。

逆に、OnTask() メソッドはイベントを効率的に待ち受けます。ーイベントを頻繁にポーリングして待ち受けるとCPU サイクルの浪費につながり、プロダクション全体のパフォーマンス低下を引き起こしかねません。OnTask() メソッドが待機する必要がある場合は、プロセスを休眠状態にするような方法 (I/O イベント待ちや Hang コマンドの使用など) で待機する必要があります。

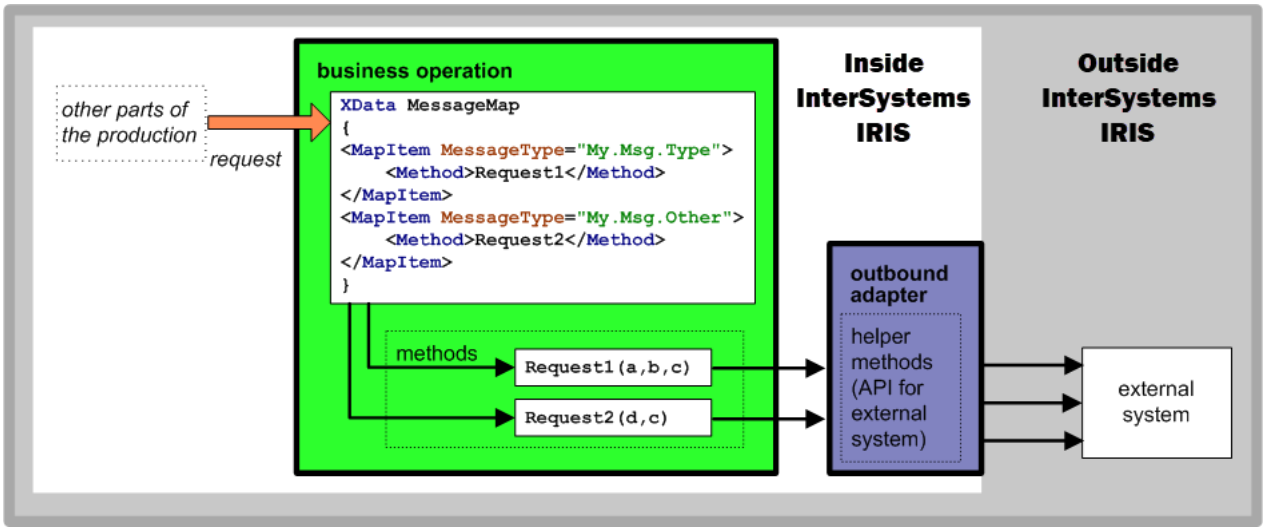
クラスがプロダクション・アダプタのサブクラスの場合は、OnTask() を実装している可能性があります。その場合は、受信アダプタ・クラスで指定されているように、サブクラスで別のメソッドを上書きする必要があります。

14.7 送信アダプタの作成またはサブクラス化

この節では、送信アダプタを作成またはサブクラス化する方法について説明します。

14.7.1 送信アダプタの概要

送信アダプタは、要求を外部システムに送信します。下の図は、プロダクションでどのように送信要求が中継されるかを示しています。



送信アダプタは、外部アプリケーションまたは外部データベースのネイティブ・プログラミング・インタフェースを、プロダクションで認識可能な形式に適合させるコードの一部です。ビジネス・オペレーションを通してプロダクションへ応答を行う各外部アプリケーションまたはデータベースには、独自の送信アダプタが必要です。ただし、外部アプリケーションまたはデータベース内のすべてのメソッドを送信アダプタにマッピングする必要はなく、プロダクションで必要とされるオペレーションだけをマッピングします。受信アダプタと同様、送信アダプタを作成するために外部アプリケーション自体を変更する必要はありません。また、アダプタ自体の概念も、プロダクションと特定のアプリケーション間またはプロダクション外のデータベース間での要求、応答、およびデータを中継するというシンプルなものです。

送信アダプタ・クラスは、ビジネス・オペレーション・クラスとの組み合わせで機能します。一般に、送信アダプタには汎用目的で再利用可能なコードが含まれていますが、ビジネス・オペレーションにはプロダクション固有のコード（特別な処理ロジックなど）が含まれます。通常は、InterSystems IRIS の組み込みアダプタ・クラスのいずれかを使用して、送信アダプタ・クラスを実装します。

14.7.2 送信アダプタの定義

送信アダプタ・クラスを作成するには、次のようにクラスを作成します。

- クラスは、Ens.OutboundAdapter（またはサブクラス）を拡張する必要があります。
- クラスは、対応するビジネス・オペレーションを呼び出すための 1 つ以上のメソッドを定義する必要があります。すべての送信アダプタで、関連するビジネス・オペレーション・クラスによって使用されるアダプタ独自の API（メソッドのセット）を自由に定義できます。
- クラスは設定を定義できます。"設定の追加と削除" を参照してください。
- クラスは任意のまたはすべてのスタートアップ・メソッドおよびティアダウン・メソッドを実装できます。"開始動作と停止動作の上書き" を参照してください。
- クラスにはプロダクション資格情報を追加できます。"アダプタ・クラスへの資格情報の追加" を参照してください。
- クラスにはその内部で作業を完了するためのメソッドを含めることができます。

14.8 アダプタ・クラスへの資格情報の追加

アダプタ・クラスにプロダクション認証情報を含めるには、クラス定義内で次の手順を実行します。

- Credentials という設定を追加します。
- 資格情報テーブル内のユーザ名とパスワードを検索するためのキーとして Credentials 設定の値を使用する CredentialsSet() というメソッドを定義します。このメソッドにより、ユーザ名とパスワードで構成された資格情報オブジェクトがインスタンス化されます。

14.9 プロダクション認証情報の上書き

プロダクション認証情報システムが管理を集中化してソース・コードの外部でログイン・データを維持しますが、別のソースから認証情報を取得するコードを作成しなければならない場合があります。例えば、お使いのコードは、Web フォームや Cookie からユーザ名とパスワードを取得してから、これらを HTTP 送信アダプタで使用して、他のサイトに接続する場合があります。

これを処理するには、アダプタのメソッドを呼び出す前に、ビジネス・サービス・コードまたはビジネス・オペレーション・コード内で、以下の両方を実行します。

- 資格情報オブジェクトをインスタンス化して、それにユーザ名とパスワードの値を割り当てるコードを入力します。
- その後、アダプタの Credentials プロパティを設定したり、アダプタの CredentialsSet() メソッドを呼び出さないでください。値がリセットされる可能性があります。

以下に例を示します。

ObjectScript

```
If ..Adapter.Credentials="" {
   Set ..Adapter.%CredentialsObj=##class(Ens.Config.Credentials).%New()
}
Set ..Adapter.%CredentialsObj.Username = tUsername
Set ..Adapter.%CredentialsObj.Password = tPassword
```

このコードには、EnsLib.HTTP.OutboundAdapter が使用できる資格情報オブジェクトがありますが、オブジェクト内の値は [認証情報] テーブルから取得しません。

14.10 開始動作と停止動作の上書き

InterSystems IRIS には、プロダクション、そのビジネス・ホスト、またはそのアダプタのライフ・サイクル期間の開始時点と終了時点でカスタム処理を追加するために上書き可能なコールバック・メソッドのセットが用意されています。デフォルトで、これらのメソッドは何もしません。

14.10.1 プロダクション・クラス内のコールバック

プロダクションの起動前に実行することが必要なコードがあり、そのコードを実行する前に InterSystems IRIS プロダクション・フレームワークを実行しておく必要がある場合は、プロダクション・クラスの OnStart() メソッドをオーバーライドする必要があります。OnStart() に目的のコード文を配置し、適切な順序で実行されるようにします。つまり、InterSystems IRIS

が起動した後、プロダクションが要求の受け付けを開始する前に実行されるようにします。また、OnStop() メソッドを使用すると、プロダクションがシャットダウンを終了する前に一連のタスクを実行できます。

14.10.2 ビジネス・ホスト・クラス内のコールバック

ビジネス・サービス、ビジネス・プロセス、ビジネス・オペレーションなどのビジネス・ホストは、Ens.Host のサブクラスです。これらのクラスのどれでも、OnProductionStart() メソッドをオーバーライドして、このホストのためにプロダクションの起動時に InterSystems IRIS が実行するコード文を指定できます。OnProductionStop() メソッドを実装することもできます。

例えば、プロダクションでプロパティ値の初期設定を変える必要がある場合は、ビジネス・オペレーションの OnInit() メソッドで値を設定します。例えば、LineTerminator プロパティの初期設定を、オペレーティング・システムに依存するように変更するには、以下のように設定します。

Class Member

```
Method OnInit() As %Status
 {
     Set ..Adapter.LineTerminator="$Select($$$isUNIX:$C(10),1:$C(13,10))"
     Quit $$$OK
 }
```

14.10.3 アダプタ・クラス内のコールバック

アダプタ・クラスは、OnInit() メソッドをオーバーライドできます。このメソッドは、アダプタ・オブジェクトが作成され、その構成可能なプロパティ値が設定された後に呼び出されます。OnInit() メソッドは、アダプタが特別な設定処理を実行するための方法を提供します。

例えば、次の OnInit() メソッドは、アダプタの起動時にデバイスとの接続を確立します。この場合は、このアダプタが ConnectToDevice() メソッドも実装していることを前提とします。

Class Member

```
Method OnInit() As %Status
{
  // Establish a connection to the input device
  Set tSC = ..ConnectToDevice()
  Quit tSC
}
```

アダプタ・クラスは、OnTearDown() メソッドもオーバーライドできます。このメソッドは、アダプタ・オブジェクトが破棄される前のシャットダウン中に呼び出されます。OnTearDown() メソッドは、アダプタが特別なクリーンアップ処理を実行するための方法を提供します。

例えば、次の OnTearDown() メソッドは、アダプタの停止時にデバイスとの接続を閉じます。この場合は、このアダプタが CloseDevice() メソッドも実装していることを前提とします。

Class Member

```
Method OnTearDown() As %Status
{
  // close the input device
  Set tSC = ..CloseDevice()
  Quit tSC
}
```

14.11 プログラムによるルックアップ・テーブルの操作

InterSystems IRIS には、ビジネス・ルールまたは DTL データ変換からテーブル検索を容易に実行できるように、Lookup() というユーティリティ関数が用意されています。この関数は、ルックアップ・テーブルが 1 つ以上作成され、各テーブルに適切なデータが入力されている場合にのみ機能します。

ルックアップ・テーブルの定義方法は、"データ・ルックアップ・テーブルの定義" を参照してください。

ルックアップ・テーブルを管理ポータルよりも直接的に操作する必要がある場合は、Ens.Util.LookupTable クラスを使用します。このクラスは、ルックアップ・テーブルをオブジェクトまたは SQL を介してアクセスできるようにします。さらに、このクラスは、テーブルの消去、XML としてのデータのエクスポート、および XML からのデータのインポートを行うクラス・メソッドを提供します。

Ens.Util.LookupTable には、以下の文字列プロパティが含まれています。

TableName

検索テーブルの名前。最大 255 文字です。ネームスペース内で定義されたルックアップ・テーブルを表示するには、InterSystems IRIS ポータルで [Interoperability]、[構成する]、[データ・ルックアップ・テーブル] の順に選択してから、[開く] を選択します。

KeyName

検索テーブルにあるエントリのキー。最大 255 文字です。これは、[Interoperability]→[構成する]→[データ・ルックアップ・テーブル] ページの [キー] フィールドの値です。

DataValue

検索テーブルでこのキーに関連付けられた値。最大 32,000 文字です。これは、[Interoperability]→[構成する]→[データ・ルックアップ・テーブル] ページの [値] フィールドの値です。

SQL クエリの例を以下に示します。

SQL

```
SELECT KeyName,DataValue FROM Ens_Util.LookupTable WHERE TableName = 'myTab'
```

Ens.Util.LookupTable には、以下のクラス・メソッドも含まれています。

%ClearTable()

指定されたルックアップ・テーブルの内容を削除します。

ObjectScript

```
do ##class(Ens.Util.LookupTable).%ClearTable("myTab")
```

%Import()

指定された XML ファイルからルックアップ・テーブル・データをインポートします。インポートを正常に行うには、このクラスの %Export() メソッドにより指定された XML 形式と同じ形式をファイルで使用する必要があります。

ObjectScript

```
do ##class(Ens.Util.LookupTable).%Import("myFile.xml")
```

%Export()

指定されたXMLファイルにルックアップ・テーブル・データをエクスポートします。指定されたファイルが既に存在している場合、そのファイルは新しいデータで上書きされます。ファイルが存在していない場合は、作成されます。以下の例では、指定した検索テーブル myTab の内容のみがエクスポートされます。

ObjectScript

```
do ##class(Ens.Util.LookupTable).%Export("myFile.xml","myTab")
```

以下の例では、ネームスペースにあるすべての検索テーブルの内容がエクスポートされます。

ObjectScript

```
do ##class(Ens.Util.LookupTable).%Export("myFile.xml")
```

得られるXMLファイルは、以下の例のようになります。すべてのテーブルにある各エントリは、単一の〈lookupTable〉要素にある兄弟要素の〈entry〉として記述されます。

XML

```
<?xml version="1.0"?>
<lookupTable>
  <entry table="myOtherTab" key="myKeyA">aaaaaa</entry>
  <entry table="myOtherTab" key="myKeyB">bbbbbbbbb</entry>
  <entry table="myTab" key="myKey1">1111</entry>
  <entry table="myTab" key="myKey2">22222</entry>
  <entry table="myTab" key="myKey3">333333</entry>
</lookupTable>
```

〈entry ごとに、そのエントリを含むテーブルを table 属性で特定します。key 属性は、キーの名前を指定します。〈entry〉要素のテキストは、そのエントリの値を表します。

前述の XML 形式のほか、SQL インポート・ウィザードを使用して、テーブルとキーで構成するカンマ区切り形式 (CSV) ファイルをインポートすることもできます。

14.12 カスタム・アーカイブ・マネージャの定義

InterSystems IRIS では、管理ポータルに、アーカイブ・マネージャと呼ばれるツールが用意されています。これについては、"プロダクションの管理" で説明します。カスタム・アーカイブ・マネージャを定義して使用できます。そのためには、次のようにクラスを作成します。

- スーパークラスとして Ens.Archive.Manager を使用できます。
- DoArchive() メソッドを定義する必要があります。このメソッドには、以下のシグニチャが設定されています。

```
ClassMethod DoArchive() As %Status
```

代替オプションとして、エンタープライズ・メッセージ・バンクを使用すれば、複数のプロダクションからのメッセージをアーカイブできます。概要は、"エンタープライズ・メッセージ・バンクの定義" を参照してください。

15
プロダクションのテストとデバッグ

このトピックでは、プロダクションのテストとデバッグで使用できる機能について説明します。ここで説明する内容は、既に企業で使用しているプロダクション・ソフトウェアのトラブルシューティングと調整にも役立ちます。

15.1 プロダクション問題状態の修正

プロダクションが一時停止またはトラブルの場合は、この節に目を通してください。

15.1.1 一時停止されたプロダクション

プロダクションの一時停止は、キュー内のすべての非同期メッセージが処理される前にプロダクションが停止された場合に発生します。これらの非同期メッセージを手動でクリアしない場合、これらはプロダクションのバックアップが開始されるときに自動的に処理されます。メッセージを処理する場合、一時停止されたプロダクションの起動前に他の手順は必要ありません。

15.1.2 トラブル・プロダクションの回復

InterSystems IRIS が停止しても、プロダクションが適切にシャットダウンしなかった場合に、そのプロダクションは [トラブル] ステータスになります。先にプロダクションを停止せずに InterSystems IRIS の再開またはマシンの再起動を実行すると、このステータスが発生する可能性があります。

この場合は、**[回復]** コマンドが **[プロダクション構成]** ページに表示されます。**[回復]** をクリックすると、障害が発生しているプロダクションのインスタンスが終了し、クリーンアップされるので、新しいインスタンスを実行できるようになります。

または、コマンド・ラインを使用してプロダクションを回復しなければならない場合があります。"Ens.Director の使用によるプロダクションの開始および停止" を参照してください。

15.1.3 ネームスペースでのプロダクションのリセット

開発時に、プロダクションのすべてのキューが確実にクリアされてから、またはプロダクションに関する情報をすべて削除してから、別のプロダクションを開始したい場合があります。CleanProduction() メソッドを実行すると、キューがクリアされます。

注意　導入済みで稼働中のプロダクションでは、この手順を使用しないでください。CleanProduction() メソッドを実行すると、キューにあるすべてのメッセージが消去され、プロダクションに関する現在の情報がすべて削除されます。この手順は、開発中のプロダクションでのみ使用してください。

CleanProduction() メソッドを使用するには、以下を実行します。

1. 以下のコマンドを入力して、該当のネームスペースに移動します。

   ObjectScript

   ```
   set $namespace = "EnsSpace"
   ```

   ここで EnsSpace は、プロダクションが実行される、プロダクション対応のネームスペースの名前です。

2. 以下のコマンドを入力します。

   ObjectScript

   ```
   do ##class(Ens.Director).CleanProduction()
   ```

15.2 管理ポータルからのテスト

プロダクションを開発、テスト、およびデバッグする際に、管理ポータルを使用して以下のようなタスクを実行できます。

- システム構成の表示と変更ができます。
- プロダクションの起動と停止ができます。
- キューとその内容、メッセージとその詳細、アダプタとアクターおよびそれらのステータス、ビジネス・プロセスとそのステータス、コード、構成項目のグラフィカル表現を表示できます。
- イベント・ログ・エントリの表示、ソート、および選択的パージができます。
- 接続が一時的に遮断されているメッセージを一時停止 (および後で再送) できます。
- カテゴリまたはメッセージ内容に基づいて、メッセージ・ウェアハウスの内容をフィルタリングして特定のメッセージを検索できます。この操作は、グラフィカル・ユーザ・インタフェースを使用して行うことも、SQL SELECT コマンドを入力して行うこともできます。
- グラフィカル・ユーザ・インタフェースを使用して、メッセージ・アクティビティをビジュアルにトレースできます。
- 統計レポートの作成と表示ができます。

開発者にとって最も役立つポータルの機能は、モニタ・サービス、テスト・サービス、およびイベント・ログの 3 つです。モニタ・サービスは実行時データを継続的に収集します。テスト・サービスを使用すると、開発中のプロダクションに対して、シミュレートした要求を発行できます。イベント・ログはビジネス・ホストによって発行されたステータス・メッセージのログを記録します。テスト・データの生成と結果の調査には、この 3 つの機能を一緒に使用してください。

ポータルの使用方法は、"プロダクションの管理" を参照してください。

管理ポータルの [テスト] メニューでは、ビジネス・ホストとデータ変換の両方をテストできます。このメニューには以下の項目が含まれています。

- [ビジネス・ホスト] – **[Interoperability]→[テスト]→[ビジネス・ホスト]** ページでは、ビジネス・プロセスとビジネス・オペレーションをテストできます。
- [データ変換] – このオプションは別のページに誘導します。このページで、データ変換を選択して [テスト] をクリックできます。詳細は、"データ変換のテスト" の節を参照してください。

15.2.1 テスト・サービスの使用

テスト・サービスを使用すると、アクティブなネームスペースで実行中のプロダクションのビジネス・プロセスまたはビジネス・オペレーションをテストできます。

ビジネス・プロセスまたはビジネス・オペレーションをテストする前に、次のことを確認してください。

- 適切なプロダクションが動作していることを確認します。"プロダクションの管理" を参照してください。
- そのプロダクションに対してテストが有効になっていることを確認します。**[プロダクション構成]** ページで次の操作を行います。
  1. **[プロダクション設定]** リンクを選択します。
  2. **[設定]** タブで、**[開発とデバッグ]** プロパティ・リストを開き、**[テスト使用可能]** チェック・ボックスにチェックを付けます。
  3. **[適用]** を選択します。

テスト・サービスには、管理ポータル内の以下の場所から移動できます。

- [Interoperability]→**[テスト]** を選択した後、**[ビジネス・ホスト]** または **[データ変換]** を選択します。
- **[プロダクション構成]** ページの構成ダイアグラムで、左ペインにあるビジネス・プロセスまたはビジネス・オペレーションを選択し、**[アクション]** タブの **[テスト]** を選択します。

ビジネス・プロセスまたはビジネス・オペレーションでテスト・サービスを使用する手順は、次のとおりです。

1. 管理ポータルで、[Interoperability]→**[テスト]**→**[ビジネス・ホスト]** を選択して、**[テスト・サービス]** ページを表示します。

   このページには、テストの対象として **[ビジネスプロセス]** と **[ビジネスオペレーション]** のどちらかを選択可能なオプションがあります。
2. 必要に応じて、**[ビジネスプロセス]** と **[ビジネスオペレーション]** のどちらかを選択します。
3. ドロップダウン・リストからテスト対象を選択します。
4. 送信するメッセージのタイプを選択します。ページに以下のフィールドが表示されます。
   - **[現在のプロダクション]** – 現在実行中のプロダクションの名前 (表示専用)。
   - **[ターゲット]** – 前の [テスト・サービス] ページで選択したビジネス・プロセスまたはビジネス・オペレーション (表示専用)。
   - **[要求タイプ]** – 要求メッセージのリストから選択します。指定されている**ターゲット**で有効な要求タイプのみが、サポートされているタイプのサブクラスも含め、表示されます。
5. 選択したメッセージのプロパティに関する値を入力します。

   要求メッセージにプロパティが含まれていない場合は、何も表示されません。

   仮想ドキュメント・メッセージをテストする場合、テキスト・メッセージを貼り付けることができる自由形式のボックスがあります。このボックスの下に、メッセージのオブジェクト・プロパティを入力できます。
6. **[実行]** を選択して、入力した値が含まれた要求を送信し、結果を表示します。

   テスト・サービスによる要求の試行に時間がかかる場合は、[待機中] のページに以下の表示専用情報が表示されます。
   - **[ターゲット]** – 要求に関連付けられているセッション ID
   - **[要求タイプ]** – 選択されたターゲットの要求タイプ。

- [セッション ID] ― 要求に関連付けられているセッション ID
- [送信済み要求] ― 要求が送信された日時。
- [受信済み応答] ― 待機中のステータスと、作業が進行中であることを示すグラフィック進行バー。

最後に、要求によって生成された応答の出力値が [結果] ページに表示されます。エラーがあれば、その完全なエラー・メッセージ・テキストと共に表示されます。

テストの完了後は、次のいずれかのコマンドを実行できます。

- [完了] を選択してホーム・ページに戻ります。
- [トレース] を選択して [ビジュアル・トレース] ページに移動し、プロダクション内でのメッセージの経路を視覚的にたどります。

EnsLib.Testing パッケージ内のクラスとメソッドを使用することもできます。詳細は、"クラス・リファレンス" の EnsLib.Testing.Service のエントリを参照してください。

15.3 プロダクションのコードのデバッグ

デバッグの最初の手順は、"プロダクションの監視" の説明に従ってトレースを有効にすることです。この手順で問題が発見されなければ、次のように、デバッガを使用してコードにステップインできます。

1. IDE でコードを編集して、デバッグを開始する箇所に BREAK コマンドを挿入します。
2. デバッグ対象のクラスを使用しているビジネス・ホストの [フォアグラウンド] 設定を有効にします。
3. プロダクションを起動します。手順 2 でマークしたジョブが、ターミナル内でフォアグラウンドで実行されます。
4. BREAK コマンドが検出されると、ターミナルがデバッグ・モードに入り、コードをステップスルーできるようになります。

詳細は、"コマンド行ルーチンのデバッグ" を参照してください。

15.4 %ETN ロギングの有効化

イベント・ログには、自動的に、システム・レベルの例外 (コード内の例外を含む) に関する部分的な情報が書き込まれます。これらのイベント・ログ・エントリの最後は、デフォルトで、次のようになります。

```
-- logged as '-' number - @' quit arg1/arg2 }'
```

このようなエラーのより完全な情報を入手するには:

1. `^Ens.Debug("LogETN")` グローバル・ノードを任意の値に設定します。

   これにより、InterSystems IRIS でシステム・レベルの例外に関する追加の詳細情報が記録されます。
2. 例外を引き起こしたと思われるコードを再実行します (メッセージの再送など)。
3. エントリの最後が次のようになっているイベント・ログを再チェックします。

   ```
   -- logged as '25 Sep 2012' number 15 @' quit arg1/arg2 }'
   ```

   この情報は、アプリケーション・イベント・ログ内のエントリを参照しています。具体的には、2012 年 9 月 25 日のアプリケーション・エラー・ログ内のエラー 15 を参照しています。

4. 次に、これらの例外を検査するために、以下のどちらかを実行できます。
   - **[システム処理]→[システムログ]→[アプリケーションエラーログ]** を選択します。
   - ^%ERN ルーチンを使用します。詳細は、"その他のデバッグ・ツール" を参照してください。

16
プロダクションの配置

通常は、プロダクションを開発システムで開発し、完成後に、テスト配置でプロダクションをテストしたら、ライブ・プロダクション・システムに配置します。このページでは、管理ポータルを使用して、開発システムからの配置用プロダクションをパッケージ化し、次に他のシステムに配置する方法について説明します。また、プロダクションへの変更の実施とテスト、およびライブ・ビジネス・データを実行するシステムへそれらの更新を配置する方法を説明します。

16.1 プロダクションの配置の概要

プロダクションは、管理ポータルまたは IDE を使用して配置できます。管理ポータルでは、IDE を使用して手動で実行する必要があるいくつかの手順が自動化されます。使用中のライブ・プロダクションがあり、そのプロダクションへの更新を開発中の場合は、ビジネス・データの処理を中断することなくライブ・プロダクションを更新する必要があります。プロダクションの配置の最も単純なレベルは、あるシステムからプロダクションの XML 定義をエクスポートして、ターゲット・システムでその XML をインポートし、コンパイルすることです。開発からライブ・システムへ正しく配置するために最も重要な事項は、以下のとおりです。

- XML 配置ファイルに必要なコンポーネントがすべて含まれていることを確認する。
- 配置ファイルをテスト・システムでテストしてから、ライブ・システムに配置する。
- ライブ・プロダクションを中断することなく、配置ファイルをターゲット・システムにロードする。

通常、プロダクションをライブ・システムに配置することは、次の手順による繰り返しプロセスになります。

1. 開発システムからプロダクションをエクスポートします。
2. テスト・システムに配置ファイルを配置します。
3. プロダクションに必要なコンポーネントがすべて含まれ、テスト・システムで適切に実行できることを確認します。なんらかの問題が見つかった場合は、修正して、手順 1 から繰り返します。
4. プロダクションをエラーなくテスト・システムに配置できたら、配置ファイルをライブ・システムに配置します。ライブ・システムを監視して、プロダクションが継続的に正しく実行されていることを確認します。

テスト・システムの環境は、できるだけライブ・システムの環境に合わせる必要があります。既存のプロダクションを更新する場合は、更新を適用する前のライブ・システムのプロダクションにテスト・システムのプロダクションを合わせる必要があります。新規の InterSystems IRIS® インストールにプロダクションを配置する場合は、テスト・システムを新規の InterSystems IRIS インストールにする必要があります。

実行中のプロダクションのコンポーネントを更新するには、以下の手順を行う必要があります。

1. 更新された XML をシステムにロードします。

2. XML をコンパイルします。
3. コンポーネントを無効化し再度有効化することで、コンポーネントの実行中のインスタンスを新しいコードに更新します。

ターゲット・システムでプロダクションのバージョンが既に実行中かどうかによって、配置プロセスは多少異なります。ターゲット・システムが古いバージョンのプロダクションを実行している場合、配置ファイルに必要なのは更新されたコンポーネントといくつかの構成項目のみで、多くの場合、プロダクション・クラスの定義を含める必要はありません。ターゲット・システムにプロダクションが含まれていない場合は、配置ファイルにすべてのプロダクション・コンポーネントと設定が含まれている必要があります。管理ポータルの **[Interoperability]→[管理]→[変更の配置]→[配置]** ページを使用して実行中のプロダクションの更新を配置する場合、このポータルによって自動的に以下が実行されます。

1. ロールバックおよびログ・ファイルが作成されます。
2. 配置ファイルに構成項目があるコンポーネントを無効化します。
3. XML をインポートおよびコンパイルします。コンパイル・エラーがある場合は、ポータルによって自動的に配置がロールバックされます。
4. 無効化されたコンポーネントが有効化されます。

いくつかの条件では、コンポーネントまたはプロダクション全体を明示的に停止し、再起動することが必要になります。IDE を使用する場合や、管理ポータルの **[システムエクスプローラ]** からクラスをインポートする場合は、これらの手順を手動で実行する必要があります。

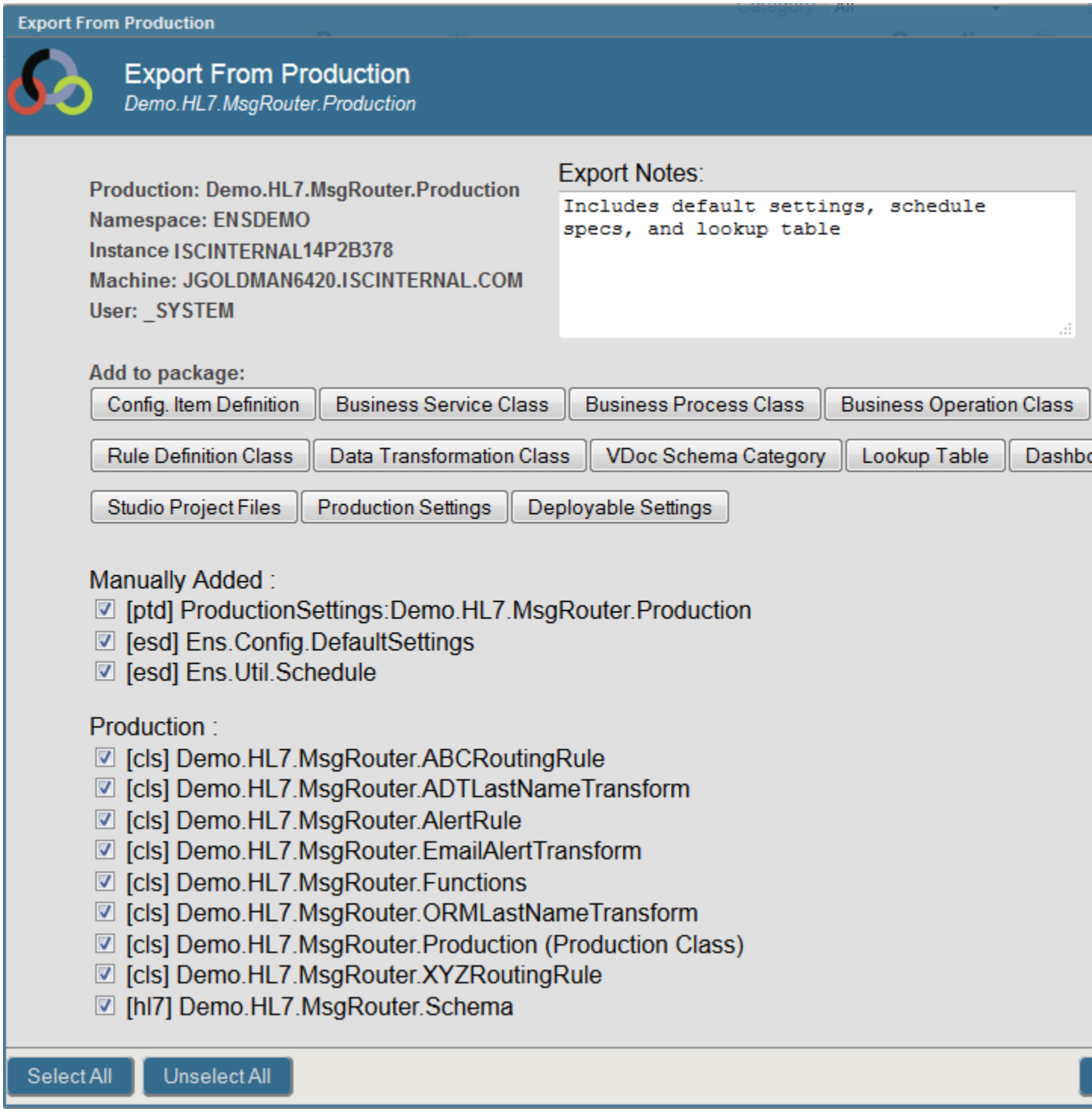
プロダクションをエクスポートして配置するには、以下のような適切な特権が必要です。

- `%Ens_Deploy:USE` : **[相互運用性]→[管理]→[変更のデプロイ]** ページにアクセスして配置アクションを実行する
- `%Ens_DeploymentPkg:USE` : XML をサーバにエクスポートする
- `%Ens_DeploymentPkgClient:WRITE` : Web ブラウザを使用して XML をローカルにエクスポートする
- `%Ens_DeploymentPkgClient:USE` : Web ブラウザを使用して XML を配置する

既定では、これらのリソースは、ロール `%EnsRole_Administrator` を持つユーザのみに自動的に付与されます。詳細は、"プロダクション関連のアクティビティを保護するリソース" を参照してください。

16.2 プロダクションのエクスポート

管理ポータルを使用してプロダクションの XML をエクスポートするには、プロダクションを開いて **[プロダクション設定]**、**[アクション]** タブの順にクリックしてから、**[エクスポート]** ボタンをクリックします。InterSystems IRIS によって、すべてのビジネス・サービス、ビジネス・プロセス、ビジネス・オペレーション、およびいくつかの関連するクラスが選択され、エクスポートの注意事項と追加コンポーネントを追加できるように以下のフォームが表示されます。



また、プロダクション構成でコンポーネントを選択し、[**アクション**] タブの [**エクスポート**] ボタンをクリックすることでも、ビジネス・サービス、プロセス、またはオペレーションをエクスポートできます。どちらの場合でも、いずれかのボタンをクリックし、コンポーネントを選択することで、コンポーネントをパッケージに追加できます。チェック・ボックスのチェックを外すことで、パッケージからコンポーネントを削除できます。

エクスポートの注意事項を使用することで、配置パッケージの内容を記述できます。例えば、パッケージに完全なプロダクションが含まれているか、プロダクションに対して更新となる一連のコンポーネントかを記述できます。エクスポートの注意事項は、管理ポータルを使用してターゲット・システムにパッケージを配置するときに表示されます。

配置パッケージをエクスポートする際、最初にする必要があるのは、ターゲット・システムで使用されているのが古いバージョンのプロダクションかどうかを判断することです。

新規インストールとしてプロダクションを配置する場合は、以下を行う必要があります。

- プロダクション・クラスの定義を含めます。
- プロダクション設定を含めます。
- プロダクションで使用されているすべてのコンポーネントの定義を含めます。
- 各コンポーネントのプロダクション設定 (ptd ファイル) を除外します。これは、プロダクション・クラスで重複して定義してしまいます。

ライブ・バージョンのプロダクションを更新するためにプロダクションを配置する場合は、以下を行う必要があります。

- プロダクション・クラスの定義を除外します。
- 変更があり、ローカル設定をオーバーライドする場合を除き、プロダクション設定を除外します。
- 更新されたすべてのコンポーネントの定義を含めます。
- XML をインポートおよびコンパイルする前に、設定が変更されたコンポーネントや無効にする必要があるコンポーネントのプロダクション設定 (ptd) ファイルを含めます。

デフォルトで多くのコンポーネントがパッケージに含められますが、その他のコンポーネントは、**[パッケージへ追加]** セクションのボタンのいずれかを選択することで手動で追加します。例えば、以下のいずれかがプロダクションで使用されている場合、それらを手動で追加する必要があります。

- レコード・マップ ー 定義され生成されたクラスが含まれます。
- 複合レコード・マップ ー 定義され生成されたクラスが含まれます。
- ルックアップテーブル
- コード内で参照されているユーザ・クラス
- 導入可能として設定されているシステム・デフォルト設定またはスケジュール指定

[プロダクション設定] ボタンを使用すると、プロダクションの ptd ファイルを追加できます。この XML は、以下を定義します。

- プロダクションのコメント
- 標準のプール・サイズ
- テストを使用可能にするかどうかと、トレース・イベントをログに記録する必要があるかどうか

チェック・ボックスのチェックを外すことで、リストのコンポーネントを選択解除できます。そのボックスにチェックを付けることでコンポーネントを選択できます。**[すべて選択]** ボタンはすべてのボックスにチェックを付け、**[すべての選択を解除]** ボタンはすべてのチェック・ボックスのチェックを外します。

配置パッケージのコンポーネントの選択が完了したら、**[エクスポート]** をクリックして配置パッケージを作成します。エクスポート・ファイルは、サーバに保存することも、ブラウザのダウンロード機能を使用してローカルに保存することもできます。サーバにエクスポートする場合は、ファイルの場所を指定できます。Web ブラウザ経由でエクスポートする場合は、ファイル名を指定できます。

配置パッケージには、作成方法に関する以下の情報が含まれています。

- InterSystems IRIS を実行するシステムの名前
- プロダクションが含まれるネームスペース
- ソース・プロダクションの名前

- プロダクションをエクスポートしたユーザ
- プロダクションがエクスポートされたときの UTC タイムスタンプ

配置ファイルのコピーは、開発システムで保管する必要があります。これは、コンポーネントへの最新の変更で新しい配置パッケージを作成するために使用します。配置ファイルのコピーを保管することで、配置ファイルに含めるコンポーネントを手動で選択する手間を減らせます。

既存の配置パッケージを使用してコンポーネントを選択し新規配置パッケージを作成するには、以下の手順を実行します。

1. 更新されたプロダクションが存在する開発システムで、**[プロダクション設定]**、**[アクション]** タブの順にクリックしてから、**[再エクスポート]** ボタンをクリックします。
2. 古い配置パッケージを含むファイルを選択します。
3. InterSystems IRIS によって、古い配置パッケージに含まれていた現在のプロダクションと同じコンポーネントが選択されます。
4. 古い配置パッケージに含まれていないコンポーネントが存在する場合や、新しいコンポーネントをプロダクションに追加した場合は、足りないコンポーネントを手動で追加します。
5. **[エクスポート]** ボタンをクリックして、更新されたコンポーネントを含んだ新しい配置パッケージを保存します。

注釈　プロダクションで XML ドキュメント用の XSD スキーマが使用されているか、X12 ドキュメント用の旧形式スキーマが使用されている場合は、これらのスキーマは XML 配置ファイルに含まれないため、別のメカニズムを通じて配置される必要があります。InterSystems IRIS では、現行形式と旧形式のどちらかまたは両方で X12 スキーマを保管できます。配置ファイルを作成する場合は、そのファイルには現行形式の X12 スキーマを含めることができますが、旧形式の X12 スキーマや XML ドキュメント用の XSD スキーマはまったく含まれません。プロダクションで XSD XML スキーマや旧形式の X12 スキーマを使用している場合は、これらのスキーマをプロダクションの配置とは別個に配置する必要があります。配置ファイルに含まれていないスキーマは、次のいずれかの方法でターゲット・システムに配置できます。

- XML スキーマまたは X12 スキーマが元々は XSD ファイルまたは SEF ファイルからインポートされたものであり、そのファイルをまだ使用可能な場合は、そのファイルをインポートすることでそのスキーマをターゲット・システムにインポートします。XSD ファイルを使用して XML スキーマをインポートでき、SEF ファイルを使用して X12 スキーマをインポートできます。
- スキーマが含まれている基盤の InterSystems IRIS グローバルをエクスポートしてから、このグローバルをターゲット・システムにインポートします。グローバルをエクスポートするには、**[システムエクスプローラ]→[グローバル]** を選択し、目的のグローバルを選択した後、**[エクスポート]** を選択します。X12 スキーマの格納先は、`EnsEDI.Description`、`EnsEDI.Schema`、`EnsEDI.X12.Description`、および `EnsEDI.X12.Schema` というグローバルです。XML スキーマは、`EnsEDI.XML.Schema` グローバルに格納されます。グローバルのエクスポートの詳細は、"グローバルの使用法" ガイドの "グローバルのエクスポート" を参照してください。

16.3 ターゲット・システムでのプロダクションの配置

管理ポータルによって、開発システムからライブ・システムへのプロダクションの配置プロセスが自動化されます。この節では、ライブ・システムで新しいバージョンのプロダクションをロードするときの InterSystems IRIS の動作について説明します。

配置パッケージの XML ファイルを入手したら、ターゲット・システムでロードできます。管理ポータルで、正しいネームスペースを選択して、**[Interoperability]**、**[管理]**、**[配置の変更]**、**[配置]** の順にクリックし、XML 配置パッケージがサーバと

ローカル・マシンのどちらにあるかに応じて、**[配置を開く]** ボタンまたは **[ローカルの配置を開く]** ボタンをクリックします。サーバ・マシンで作業している場合、**[ローカルの配置を開く]** ボタンはアクティブではありません。XML 配置パッケージ・ファイルを選択すると、フォームに、配置パッケージの新規および変更された項目が一覧で示され、パッケージの作成時に指定された配置の注意事項が表示されます。このフォームでは、以下の配置設定を指定できます。

- ターゲット・プロダクション ー コンポーネントを追加するプロダクションを指定します。配置パッケージにソース・プロダクションのプロダクション・クラスが含まれている場合、ターゲット・プロダクションがソース・プロダクションに設定され、変更できません。そうでない場合、InterSystems IRIS によってデフォルト・プロダクションが現在開いているプロダクションに設定されますが、これは変更可能です。
- ロールバック・ファイル ー ロールバック情報を含めるファイルを指定します。ロールバック・ファイルには、配置によって置換されるすべてのコンポーネントの現在の定義が含まれています。
- 配置ログ・ファイル ー 配置によって実行される変更のログを含んでいます。

パッケージを配置する過程で、InterSystems IRIS は、以下の手順でプロダクションを停止し、新しいコードをロードして、プロダクションを再起動します。

1. ロールバック・パッケージを作成および保存します。
2. 配置パッケージにプロダクション設定 (ptd) ファイルがあるプロダクションでコンポーネントを無効にします。
3. XML ファイルをインポートし、コードをコンパイルします。コンポーネントのコンパイルでエラーが発生した場合は、配置全体がロールバックされます。
4. プロダクション設定を更新します。
5. 配置の詳細を説明するログを記録します。
6. 無効にされているプロダクション・コンポーネントの現在の設定が有効になるように指定されている場合は、プロダクション・コンポーネントを有効にします。

この配置の変更の結果を元に戻すには、**[配置を開く]** ボタンを使用してロールバック・ファイルを選択し、**[配置]** ボタンをクリックします。

A

プロダクションとその構成要素のライフ・サイクル

このページでは、参考として、プロダクションとその構成要素のライフ・サイクルについて説明します。

A.1 プロダクションのライフ・サイクル

A.1.1 プロダクションの起動

プロダクションを起動すると、以下に示す一連のアクションが実行されます。

1. プロダクション・クラスがインスタンス化され、オプションの OnStart() メソッドが実行されます。
2. プロダクションにより、各ビジネス・オペレーションがインスタンス化され、オプションの OnProductionStart() メソッドが実行されます。
3. プロダクションにより、各ビジネス・プロセスがインスタンス化され、オプションの OnProductionStart() メソッドが実行されます。
4. プロダクションによって、前の実行で残されたメトリック値のビジネス・メトリック・キャッシュがクリアされます。
5. プロダクションにより、各ビジネス・サービスがインスタンス化され、オプションの OnProductionStart() メソッドが実行されます。
6. プロダクションによって、既にキューに入っている項目がすべて処理されます。これには、プロダクションが停止したときにキューに入れられた非同期メッセージが含まれます。
7. これでプロダクションは InterSystems IRIS® 外部からの入力を受け付けるようになります。

A.1.2 プロダクションのシャットダウン

プロダクションを停止すると、以下に示す一連のアクションが生じます。

1. プロダクションにより、各ビジネス・サービスがオフラインになり、オプションの OnProductionStop() メソッドが実行されます。このアクションによって、InterSystems IRIS 外部からのすべての要求が停止します。
2. すべてのビジネス・ホストが、静止状態になるための信号を受信します。

3. すべてのキューが静止状態になります。つまり、この時点以降、ビジネス・ホストが処理できるのはキューに入れられた、優先順位の高いメッセージ (同期メッセージ) のみになります。非同期メッセージはそれぞれのキューに残ります。
4. プロダクションは、その処理能力の及ぶ限りすべての同期メッセージの処理を終了させます。
5. プロダクションにより、各ビジネス・プロセスがオフラインになり、オプションの OnProductionStop() メソッドが実行されます。
6. プロダクションにより、各ビジネス・オペレーションがオフラインになり、オプションの OnProductionStop() メソッドが実行されます。
7. プロダクションがオフラインになります。InterSystems IRIS により、このプロダクション・クラスでオプションの OnStop() メソッドが実行されます。

A.2 ビジネス・サービスとアダプタのライフ・サイクル

A.2.1 プロダクションの起動

プロダクションを起動 (または特定のビジネス・サービスの設定を変更) すると、InterSystems IRIS は構成済みの各ビジネス・サービス・クラス (つまり、プロダクション定義に記載されたすべてのビジネス・サービス) に対して、以下の処理を自動的に実行します。

1. 定義に応じて、ビジネス・サービスの OnProductionStart() コールバック・メソッドが呼び出されます。

   OnProductionStart() メソッドは、プロダクション構成に記載されているビジネス・サービス・クラスごとに一度呼び出されるクラス・メソッドです。ビジネス・サービス・クラスでは、このコールバックを使用して必要とされるクラス全体の初期化を実行できます。ビジネス・サービスにアダプタがない場合、ビジネス・サービス・クラスでは、このコールバックを使用してエラーをチェックできます。

2. InterSystems IRIS は、ビジネス・サービスを実行する 1 つまたは複数のバックグラウンド・プロセスを生成します。

   バックグラウンド・プロセスの数は、プロダクション構成に記述されたビジネス・サービスの PoolSize プロパティによって決まります。各バックグラウンド・プロセスは、ビジネス・サービスのインスタンスとも呼ばれ、ビジネス・サービス・オブジェクトのインスタンスが含まれています。

   InterSystems IRIS では、以下の条件が満たされている場合のみ、ビジネス・サービスのバックグラウンド・プロセスが作成されます。

   - ビジネス・サービス・クラスに、ADAPTER クラス・パラメータで指定された、受信アダプタ・クラスが関連付けられていること。

     関連する受信アダプタを持たないビジネス・サービス・クラスは、“アダプタ不要型のサービス” と呼ばれます。外部イベントを待つ代わりに、そのようなサービスがプロセス内で呼び出されます (例えば、複合アプリケーションからの呼び出し)。

   - プロダクション構成内のビジネス・サービスの Enabled プロパティが 1 に設定されていること (そのように設定されていない場合、ビジネス・サービスは無効と見なされ、入力を受け付けません)。
   - プロダクション構成内のビジネス・サービスの PoolSize プロパティが 0 より大きい値に設定されていること。

   プロダクション構成内にあるビジネス・サービスの Foreground プロパティを 1 に設定している場合は、InterSystems IRIS はそのビジネス・サービス用にフォアグラウンド・プロセスを生成します (つまり、InterSystems IRIS はターミナル・ウィンドウを開きます)。この機能により、テストとデバッグは効率化されます。

3. InterSystems IRIS は、ビジネス・サービスのステータスおよびオペレーション履歴の監視に使用するシステム監視情報を初期化します。
4. 各バックグラウンド・プロセス内で、InterSystems IRIS は以下を実行します。
   a. ビジネス・サービス・クラスのインスタンスを作成します。
   b. すべてのビジネス・サービスの設定に最近構成された値を入力します。
   c. ビジネス・サービスの OnInit() コールバック・メソッド (存在する場合) を呼び出します。OnInit() メソッドは、ビジネス・サービスの任意の初期化ロジックを実行するのに役立つインスタンス・メソッドです。
   d. 関連するアダプタ・クラス (定義されている場合) のインスタンスを作成し、すべてのアダプタの設定に最近構成された値を入力します。

A.2.2 実行時

プロダクションの実行時に、ビジネス・サービスは受信アダプタの OnTask() メソッドを繰り返し呼び出します。この OnTask ループは、ビジネス・サービスの **CallInterval** 設定および **%WaitForNextCallInterval** プロパティにより、以下のように制御されます。

1. ビジネス・サービスで受信アダプタの OnTask() メソッドを呼び出します。
2. OnTask() で、ビジネス・サービスの対象となる入力イベントに関して、InterSystems IRIS プロダクション外部をチェックします。
   - 入力が検出されると、OnTask() は関連付けられたビジネス・サービス・オブジェクトの ProcessInput() メソッドを呼び出します。
   - 入力が検出されなかった場合、OnTask() は制御をビジネス・サービスに戻し、次の **CallInterval** が経過するのを待ってから手順 1 に戻ります。
   - 複数の入力イベントが存在する場合もあります。例えば、ビジネス・サービスで **File.InboundAdapter** が使用されている場合は、指定されたディレクトリで複数のファイルが待機していることがあります。

     複数の入力イベントがある場合、以下のような処理が行われます。
     - 一般的には、OnTask() メソッドにより、待機中の入力イベントがすべてなくなるまで必要なだけ ProcessInput() が呼び出されます。
     - また、複数の入力イベントが存在していても **CallInterval** ごとに 1 回だけ ProcessInput() を呼び出すように受信アダプタで OnTask() を制限できます。OnTask() は、すべての入力イベントは処理せず、最初に検出したイベントの処理を終えると、休止状態に入ります。

3. ProcessInput() は、ビジネス・サービスの **%WaitForNextCallInterval** プロパティを 0 (偽) に設定し、OnProcessInput() を呼び出して入力イベントを処理します。
4. 完了すると、ProcessInput() は OnTask() にコントロールを戻します。
5. この時点で、OnTask() は **%WaitForNextCallInterval** を 1 (真) に設定することもできます。このように設定すると、複数の入力イベントが存在しても、ビジネス・サービスで **CallInterval** の期間に入力イベントが 1 つしか処理されないよう制限されます。

   通常は、ビジネス・サービスで待機中の入力イベントをすべて遅延なく処理したいので、この時点で **%WaitForNextCallInterval** を変更しません。ProcessInput() で設定された 0 (偽) のままにしておきます。

   アダプタ基本クラス **Ens.InboundAdapter** には、ProcessInput() を呼び出し、**%WaitForNextCallInterval** を 1 に設定して返される OnTask() メソッドがあります。

Tipヒント InterSystems IRIS 外部のイベントを気にすることなく、ビジネス・サービスを呼び出して、その ProcessInput() メソッドを CallInterval ごとに 1 回ずつ実行するだけであれば、アダプタ・クラスの Ens.InboundAdapter を使用します。

6. OnTask() が返されます。
7. ビジネス・サービスは、その %WaitForNextCallInterval プロパティの値をテストします。
   - 1 (真) の場合、ビジネス・サービスは、CallInterval の期間が経過するのを待ってから、手順 1 に戻ります。
   - 0 (偽) の場合、ビジネス・サービスは直ちに手順 1 に戻ります。OnTask() でそれ以上入力がないと判断されるまで、CallInterval は使用されません (手順 2 を参照)。

A.2.3 プロダクションのシャットダウン

プロダクションが停止すると、ビジネス・サービスに関連する以下のイベントが発行されます。

1. InterSystems IRIS は各ビジネス・サービスを無効にします。以後、このプロダクションに対する要求は受け付けられません。
2. 各受信アダプタ内の OnTearDown() メソッドが呼び出されます。
3. すべての受信アダプタ・オブジェクトとビジネス・サービス・オブジェクトが破棄され、バックグラウンド・プロセスは強制終了されます。
4. プロダクション内の該当クラスの構成アイテムごとに一度、各ビジネス・サービスの OnProductionStop() クラス・メソッドが呼び出されます。

ビジネス・サービスがシステム管理者により無効化されたり、スケジュールの構成に応じて非アクティブになると、プロダクションは実行を継続しますが、関連付けられた受信アダプタは終了し、その OnTearDown() メソッドが実行されます。

A.3 ビジネス・プロセスのライフ・サイクル

プロダクションが起動するたびに、InterSystems IRIS によってそのプロダクションのパブリック・アクター・プールが作成されます。ActorPoolSize 設定の値によって、プール内のジョブ数が決定されます。

アクター・プールの各ジョブ内には、Ens.Actor オブジェクトのインスタンスがあり、このインスタンスによって、ビジネス・プロセスによるジョブの使用が管理されます。この Ens.Actor インスタンスは、アクターと呼ばれます。

プロダクションのビジネス・プロセスは、Ens.Actor と呼ばれるパブリック・メッセージ・キューを共有できます。このパブリック・キューは、独自の専用キューを持たないプロダクション内の任意のビジネス・プロセスに送信されるすべてのメッセージのターゲットになります。アクターは、ビジネス・プロセスのホスティングから解放されているときは、いつでも Ens.Actor キューをリッスンしています。Ens.Actor キューに要求が着信すると、解放されている任意のアクターが、その要求で指定されているビジネス・プロセスをホスティングするためのジョブを割り当てることがあります。Ens.Actor キューに着信した要求は、着信した順番で処理されます。連続した要求がある場合は、次に空いているアクターによって、次々に処理されていきます。

プールの詳細は、"プール・サイズとアクター・プール・サイズ" を参照してください。

ビジネス・サービスやビジネス・オペレーションのライフサイクルとは異なり、ビジネス・プロセスの `OnInit` メソッドおよび `OnTearDown` メソッドは、ビジネス・プロセスの開始時または停止時に呼び出されません。これらのメソッドは、要求がビジネス・プロセス (InProc または Queued) によって処理されるたびに実行されます。ビジネス・プロセス・クラスの `OnProductionStart` メソッドを実装すると、プロダクションの開始時またはビジネス・プロセスの再起動時にカスタム・コードを実行できます。このクラスの `OnProductionStop` メソッドは、プロダクションの停止時に呼び出されます。

A.3.1 パブリック・キュー内のライフ・サイクル

パブリック・キューを使用するビジネス・プロセスのライフ・サイクルを以下に示します。

1. ビジネス・プロセスに対する最初の要求が送信されます。要求メッセージが Ens.Actor キューに着信します。
2. いずれかのアクターが空き状態になると、そのアクターは直ちに Ens.Actor キューから要求メッセージを取り出します。
3. そのアクターは、適切なビジネス・プロセス・クラスの新しいインスタンスを作成します。ビジネス・プロセスの OnInit() メソッドが呼び出されます。アクターは、ビジネス・プロセスの設定に最新の値を入力し、そのインスタンスの OnRequest() メソッドを呼び出します。これで、ビジネス・プロセス・インスタンスが継続します。
4. ビジネス・プロセス・インスタンスが継続中でも、アクティブでない場合 (例えば、入力やフィードバックを待っている場合) は、通常、インスタンス化を行ったアクターに制御が戻されます。この場合、アクターは以下の処理を行います。
   a. アクターがビジネス・プロセス・インスタンスを中断します。
   b. アクターがインスタンスの現在の状態をディスクに保存します。
   c. アクターがそのジョブをアクター・プールに返します。
5. 継続中のビジネス・プロセスにアクターが割り当てられておらず、そのビジネス・プロセスに対して下位要求または応答が着信した場合は (例えば、予測している入力またはフィードバックが着信した場合は)、常に以下のシーケンスが発生します。
   a. いずれかのアクターが空き状態になると、そのアクターが Ens.Actor キューから新しい要求メッセージまたは応答メッセージを取り出します。
   b. アクターは、対応するビジネス・プロセス・オブジェクトをディスクからリストアし、そのすべてのステータス情報を復元します。
   c. アクターは、状況に応じて、OnRequest() または OnResponse() というインスタンス・メソッドを呼び出します。
6. ビジネス・プロセス・インスタンスが実行を完了したら、その現在のアクターがその OnComplete() メソッドを呼び出し、そのインスタンスを [完了したか] ステータスでマーキングします。また、アクターはそのジョブをアクター・プールに返します。これで、このビジネス・プロセス・インスタンスにイベントが送信されなくなります。ビジネス・プロセスの OnTearDown メソッドが呼び出されます。

A.3.2 プライベート・キュー内のライフ・サイクル

または、ビジネス・プロセスに専用キューを割り当てて、パブリック Ens.Actor キューを迂回するように構成できます。専用キューを持つビジネス・プロセスのライフ・サイクルは、パブリック・キューの説明にあるもの同様に実行されますが、以下の点が異なります。

- ビジネス・プロセスは、専用プールのみからジョブを実行します。
- このビジネス・プロセスに宛てたメッセージは、その専用キューに着信し、Ens.Actor キューには着信しません。

A.4 ビジネス・オペレーションとアダプタのライフ・サイクル

InterSystems IRIS は、各ビジネス・オペレーションのライフ・サイクルを自動で管理します。

A.4.1 プロダクションの起動

プロダクションを起動（または特定のビジネス・オペレーションの構成を変更）すると、InterSystems IRIS は構成済みの各ビジネス・オペレーション・クラス（つまり、プロダクション定義に記載されたすべてのビジネス・オペレーション）に対して、以下の処理を自動的に実行します。

1. 定義に応じて、クラスの OnProductionStart() コールバック・メソッドが呼び出されます。

   OnProductionStart() メソッドは、プロダクション構成にリストされているビジネス・オペレーション・クラスごとに一度呼び出されるクラス・メソッドです。ビジネス・オペレーション・クラスでは、このコールバックを使用して必要とされるクラス全体の初期化を実行できます。

2. ビジネス・オペレーションを実行する 1 つまたは複数のバックグラウンド・プロセスが生成されます。

   バックグラウンド・プロセスの数は、プロダクション構成に記述されたビジネス・オペレーションの PoolSize プロパティによって決まります。各バックグラウンド・プロセスは、ビジネス・オペレーションのインスタンスとも呼ばれ、ビジネス・オペレーション・オブジェクトのインスタンスが含まれています。

   InterSystems IRIS では、以下の条件が満たされている場合のみ、ビジネス・オペレーションのバックグラウンド・プロセスが作成されます。

   - ビジネス・オペレーション・クラスの INVOCATION クラス・パラメータが Queue に設定されていること。
   - プロダクション構成内のビジネス・オペレーションの Enabled プロパティが 1 に設定されていること（そのように設定されていない場合、ビジネス・オペレーションは無効と見なされます）。無効なビジネス・オペレーションでも受信メッセージ・キューは機能しています。ただし、このキューに送信した要求はビジネス・オペレーションを有効にしないと処理されません。
   - プロダクション構成内のビジネス・オペレーションの PoolSize プロパティが 0 より大きな値に設定されていること。

   プロダクション構成内にあるビジネス・オペレーションの Foreground プロパティが 1 に設定されている場合は、InterSystems IRIS はそのビジネス・オペレーション用にフォアグラウンド・プロセスを生成します（つまり、ターミナル・ウィンドウを開きます）。この機能により、テストとデバッグは効率化されます。

3. これにより、ビジネス・オペレーションのステータスとオペレーション履歴の監視に使用するシステム監視情報が初期化されます。

4. 各バックグラウンド・プロセスの中では、以下の処理が行われます。

   a. InterSystems IRIS が、ビジネス・オペレーション・クラスのインスタンスを作成して、すべてのビジネス・オペレーションの設定に最近構成された値を入力し、ビジネス・オペレーションの OnInit() コールバック・メソッド（存在する場合）を呼び出します。OnInit() メソッドは、ビジネス・オペレーションの任意の初期化ロジックを実行するのに役立つインスタンス・メソッドです。

   b. InterSystems IRIS が、関連するアダプタ・クラス（定義されている場合）のインスタンスを作成し、すべてのアダプタの設定に最近構成された値を入力します。

A.4.2 実行時

実行時は、ビジネス・オペレーションが以下の処理を行います。

1. 要求が（ビジネス・サービス、ビジネス・プロセス、およびその他のビジネス・オペレーションから）関連するメッセージ・キューに送信されるまで待機します。

2. ビジネス・オペレーションは、そのメッセージ・キューから要求を取得すると、メッセージ・マップで要求タイプに応じたオペレーション・メソッドを検索します。その後、そのオペレーション・メソッドを呼び出します。

3. オペレーション・メソッドは、要求オブジェクト内のデータを使用して、外部アプリケーションに要求を行います。通常、この処理は、関連する送信アダプタ・オブジェクトの 1 つ以上のメソッドを呼び出すことによって実行されます。

A.4.3 プロダクションのシャットダウン

プロダクションが停止すると、ビジネス・オペレーションに関連する以下のイベントが発行されます。

1. InterSystems IRIS は、各ビジネス・オペレーションが休止状態に至るのを待ちます (つまり、InterSystems IRIS は各ビジネス・オペレーションがすべての同期要求を完了するのを待ちます)。
2. 各送信アダプタ内の OnTearDown() メソッドが呼び出されます。
3. すべての送信アダプタ・オブジェクトとビジネス・オペレーション・オブジェクトが破棄され、バックグラウンド・プロセスは強制終了されます。
4. プロダクション内の該当クラスの構成アイテムごとに一度、各ビジネス・オペレーションの OnProductionStop() クラス・メソッドが呼び出されます。

ビジネス・オペレーションがシステム管理者により無効化されたり、スケジュールの構成に応じて非アクティブになると、プロダクションは実行を継続しますが、関連付けられた送信アダプタは終了し、その OnTearDown() メソッドが実行されます。