プロダクション内での仮想ドキュメントの使用法

Version 2023.1
2024-01-02

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

プロダクション内での仮想ドキュメントの使用法
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

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

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

目次

1

仮想ドキュメント

この章では、仮想ドキュメントの定義、有効性、および標準メッセージとの違いについて説明します。また、仮想ドキュメントを標準メッセージと全く同じように使用するために InterSystems IRIS から提供されているツールについても紹介します。

1.1 概要

仮想ドキュメントはメッセージの一種で、InterSystems IRIS では部分的にしか解釈されません。仮想ドキュメントの目的を理解するには、プロダクション・メッセージをもう少し詳しく調べる必要があります。すべてのプロダクション・メッセージが次の 2 つの部分から構成されています。

- メッセージ・ヘッダには、InterSystems IRIS 内でメッセージをルーティングするために必要なデータが含まれています。メッセージ・ヘッダは、常に同じタイプのオブジェクトです。これは、永続オブジェクトです。つまり、InterSystems IRIS データベース内のテーブルに保存されます。
- メッセージ・ボディには、メッセージ・データが含まれています。標準メッセージの場合は、メッセージ・ボディが永続オブジェクトです。仮想ドキュメントの場合は、後述するように、メッセージ・ボディが別の方法で実装されます。

オブジェクトは、データの各断片を別々のプロパティとして表現します。これには、オブジェクト内の任意の値に簡単にアクセスできるというメリットがあります。コードを記述するときは、値を取得するために、名前でクラス・プロパティを参照します。

EDI (Electronic Data Interchange) 形式の場合は、このアプローチが扱いにくく、不要なものになります。これは、大量のプロパティ (数百個になる場合もある) が必要となり、オブジェクトのインスタンスの作成プロセスに時間がかかる可能性があるためと、アプリケーションが多くても、ドキュメント内で実際に使用可能なフィールドのほとんどが使用されないためです。

これらの問題を解決するために、InterSystems IRIS には仮想ドキュメントと呼ばれる別の種類のメッセージ・ボディが用意されています。仮想ドキュメントを使用すれば、ドキュメントの内容を正式なプロパティのセットとして保持するためのオブジェクトを作成せずに、ドキュメントの内容そのままをプロダクション・メッセージのボディとして送信できます。仮想ドキュメント内のデータは内部使用グローバルに直接保存されるため、処理速度が向上します。

仮想ドキュメント・アプローチは XML ドキュメント (標準メッセージとして処理することもできる) に対しても有効です。

1.2 仮想ドキュメントの種類

InterSystems IRIS では、以下の種類のドキュメントを仮想ドキュメントとして処理できます。

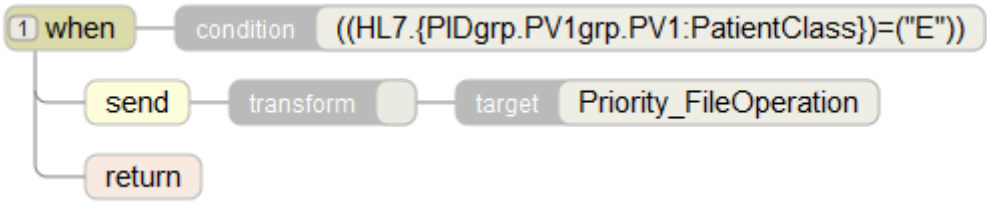
ドキュメントの種類	参照先
EDIFACT ドキュメント	プロダクション内での EDIFACT ドキュメントのルーティング
X12 ドキュメント	プロダクション内での X12 ドキュメントのルーティング
XML ドキュメント	プロダクション内での XML 仮想ドキュメントのルーティング

XML ドキュメントを標準メッセージとして処理することもできます。これを実現するには、対応する XML スキーマからクラスを生成する必要があります。詳細は、"XML ツールの使用法" を参照してください。

1.3 仮想ドキュメントの内容へのアクセス

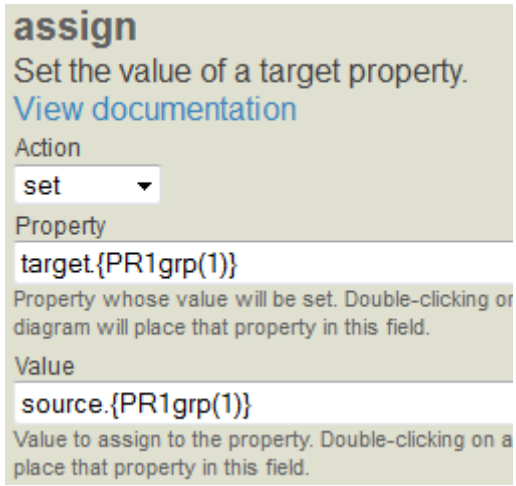
仮想ドキュメント内のデータを操作するには、その中の特定のデータ項目を識別できる必要があります。このデータ項目を仮想プロパティといいます。仮想プロパティ・パスは InterSystems IRIS で仮想プロパティの場所を指定するために使用される構文です。仮想プロパティ・パスは、特に、以下の場所で使用できます。

- ビジネス・ルール。以下に例を示します。



  仮想プロパティ・パスが中かっこで囲まれています。

- データ変換。以下に例を示します。



  [**プロパティ**] と [**値**] の両方に関して、仮想プロパティ・パスが中かっこで囲まれています。

結果的に、プロダクションでは、標準メッセージとほとんど同じように仮想ドキュメントを操作できます。

詳細は、後述する “仮想プロパティ・パス” の章を参照してください。

1.4 フィルタと検索のサポート

標準のメッセージ本文の場合、管理ポータルでメッセージの各プロパティを直接検索することができます。つまり、ユーザは、プロパティのパスを知らなくても、プロパティを検索やフィルタリングに使用することが可能です。

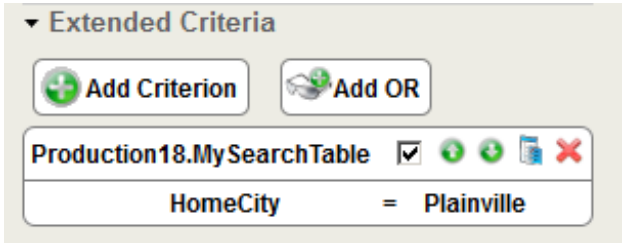
デフォルトで、メッセージ識別子を除いて、管理ポータルでは、仮想ドキュメント内のデータを直接検索することはできません。そのため、ユーザがデータ項目を検索やフィルタリングに使用するには、その項目のプロパティ・パスを知っている必要があります。

ユーザを支援するために、以下のメカニズムを使用して、仮想プロパティを直接検索可能にできます。

- 検索テーブル・クラスを定義します。このクラスでは、仮想プロパティ・パスを使用して検索可能プロパティを定義します。
- 検索テーブル・クラスを使用するうえで適用可能なビジネス・ホストを構成してください。

  ビジネス・ホストでメッセージが受信されると、InterSystems IRIS によってそれらのプロパティが標準メッセージ本文内のプロパティであるかのようにインデックスが割り当てられます。

これにより、ユーザは使用するプロパティ・パスを知らなくても、それらのプロパティを直接使用できるようになります。以下に例を示します。



検索テーブルの定義方法は、後述する“検索テーブルの定義”を参照してください。

2
スキーマ定義

この章では、InterSystems IRIS® で仮想ドキュメントを検証する (および “仮想プロパティ・パス” の章で説明されているように仮想ドキュメント内のデータにアクセスする) ために使用されるスキーマ定義について説明します。

2.1 スキーマ定義の概要

仮想ドキュメントはオブジェクト (対応するクラス定義を含む) として表現されないため、InterSystems IRIS には仮想ドキュメントを解釈したり、検証したりするための追加のツールが必要です。これらのツールではスキーマ定義から始まります。

InterSystems IRIS スキーマ定義は、特定の EDI 標準または XML スキーマを表現する一連の記述です。スキーマ定義は InterSystems IRIS の概念です。データベース・スキーマや XML スキーマなどの他の概念と混同しないようにしてください。ただし、スキーマ定義は対応する EDI または XML スキーマに基づいています。

スキーマ定義は、特定の相互運用対応ネームスペースに固有であり、そのネームスペースの InterSystems IRIS データベースに保存されます。

2.2 スキーマ・カテゴリ

各スキーマ定義によって、InterSystems IRIS は EDI 標準 (または XML スキーマ) の全体像を把握できます。例えば、X12 にはスキーマ定義が 1 つあります。実際には、1 つのスキーマ定義に、対象となる標準の 1 つのサブセットしか含めることができない場合があります。これは、対応する EDI スキーマがどのように InterSystems IRIS にインポートされたかによって異なります。

スキーマ定義では、スキーマ・カテゴリが分類規則です。各スキーマ定義は、1 つまたは複数のスキーマ・カテゴリを含みます。

2.3 ドキュメント構造

スキーマ定義では、各スキーマ・カテゴリに 1 つ以上のドキュメント構造が含まれています。ドキュメント構造ごとに 1 つずつのドキュメント・タイプが記述されます。

EDI 標準に応じて、スキーマ定義をスキーマ・カテゴリとドキュメント構造に整理するためのさまざまなアプローチが使用されます。例えば、X12 の場合は、スキーマ定義がもっとフラットな構造をしており、通常はスキーマ・カテゴリごとにドキュメント構造が 1 つずつしか含まれていません。

2.4 ドキュメント・タイプ (DocType)

仮想ドキュメントごとに 1 つずつのドキュメント・タイプ (通常は単に DocType (この情報が保存されるプロパティの名前) と表現される) が設定されます。これは、スキーマ定義の特定の部分に対応します。この部分で、仮想ドキュメントの期待される構造と含まれる値が記述されます。DocType を使用すれば、InterSystems IRIS でその仮想ドキュメントを検証したり、解釈したりできます。

仮想ドキュメントの **DocType** は、スキーマ・カテゴリとドキュメント構造の組み合わせで識別されます。具体的に、**DocType** プロパティの構文は次のようになります。

```
category:structure
```

説明 :

- category はスキーマ・カテゴリの名前です。
- structure は、参照される category 内のドキュメント構造の名前です。この断片は、スキーマ・カテゴリにドキュメント構造が 1 つしか含まれていない場合でも必須です。

2.5 ツール

管理ポータルには、[スキーマ構造] ページがあります。このページでは、EDI または XML スキーマをインポート (スキーマ定義を作成するため) したり、スキーマをエクスポートしたり、スキーマ定義を参照したりできます。後述する “ポータル・ツール” を参照してください。

3

仮想プロパティ・パス

この章では、InterSystems IRIS® で仮想ドキュメント内のデータにアクセスするために使用される仮想プロパティ・パスについて説明します。

3.1 概要

仮想ドキュメントを使用するには、仮想ドキュメント内の特定のデータ項目を識別できる必要があります。このデータ項目を仮想プロパティといいます。仮想プロパティ・パスは InterSystems IRIS で仮想プロパティの場所を指定するために使用される構文です。

XML 仮想ドキュメントを除いて、仮想プロパティ・パスは、該当する EDI スキーマが InterSystems IRIS にロードされている場合にしか使用できません。スキーマが InterSystems IRIS にロードされたら、InterSystems IRIS で対応するドキュメントを EDI スキーマによって意図されたように解釈するために必要なすべての情報が揃うことになります。

3.2 EDI ドキュメントの解釈：セグメントとフィールド

EDIFACT および X12 ドキュメントの場合は、未加工のデータ・ストリームがセグメントに分割され、さらにフィールドに分割されます。(詳細は、XML ドキュメントの場合と異なります。"プロダクション内での XML 仮想ドキュメントのルーティング" を参照してください。)

終端文字 (多くの場合はキャリッジ・リターン) は、セグメントの最後を示します。この場合は、各行が 1 つのセグメントです。

セグメント内では、セパレータ文字がフィールド間の境界を示します。この場合は、コロンがフィールド間のセパレータ文字で、セミコロンがサブフィールド間のセパレータ文字です。

フィールドには、サブフィールドとそのさらに下位区分を含めることができます。これらは他のセパレータ文字で区切られます。

ターミネータ文字とセパレータ文字の詳細は EDI 形式によって異なります。

- X12 の詳細は、"区切り文字" を参照してください。X12 のデフォルトのセパレータは以下のとおりです。

  ```
  * : \a ~ \r \n
  ```

- EDIFACT の詳細は、"区切り文字" を参照してください。EDIFACT のデフォルトのセパレータは以下のとおりです。

  ```
  : + ? ' \r \n
  ```

3.3 セグメント構造

一般に、EDI 標準は、構成要素として使用可能な多数のセグメントを定義します。ドキュメント構造ごとに特定のセグメントしか含まることができません。複数のドキュメント構造で別々の順番または数量のセグメントを組み合わせることもできます。

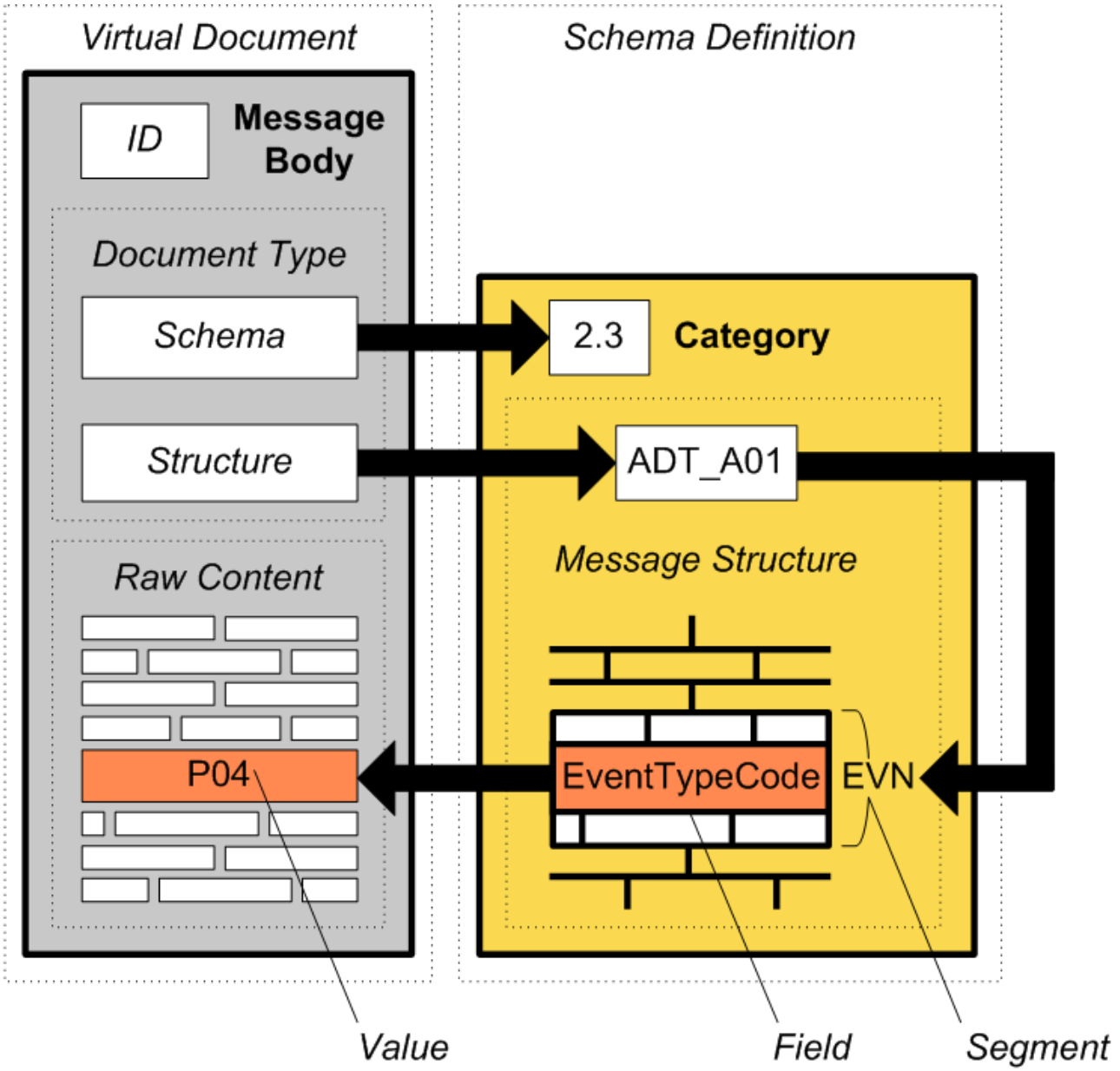
(詳細は、XML ドキュメントの場合と異なります。"プロダクション内での XML 仮想ドキュメントのルーティング" を参照してください。)

3.4 仮想プロパティ・パスの特定

概念上は、仮想プロパティ・パスに以下のすべての単位が含まれています。

- category : structure ー その **DocType**
- segment : field ー その **DocType** の仮想ドキュメント内のデータ値へのパス

以下の図は、この規則を示しています。



通常、パスの segment 部分で、セグメントのグループや繰り返しブロックを含む、階層的なドキュメント構造内にあるターゲット・セグメントを識別します。例えば、2.3:ORM_O01 メッセージの NTE セグメントは、以下のように識別されます。

```
ORCgrp(1).OBRuniongrp.OBXgrp(3).NTE(1)
```

同様に、パスの field 部分で、ターゲット・セグメント内のフィールド、サブフィールド、および繰り返しグループの階層的な構造内にある、ターゲット・フィールドを識別します。実際には、NTE 内の各 field は、以下のように、簡単です。

```
SourceofComment
```

したがって、segment:field の完全なパスは次のようになります。

```
ORCgrp(1).OBRuniongrp.OBXgrp(3).NTE(1):SourceofComment
```

複雑な階層構造を持つフィールドもあります。同じ 2.3:ORM_O01 メッセージ構造の PID セグメントを見てみます。segment では、以下のように識別されます。

```
PIDgrp.PID
```

以下のような field パスを使用できます。

```
PatientIDInternalID(1).identifiertypecode
```

segment パスと異なり、パスの field 部分では、通常、フィールドやサブフィールドの名前の代わりに数値を使用できます。例えば、上記の名前の代わりに以下の番号を使用できます。

```
3(1).5
```

管理ポータルには、正しい segment:field パスを特定するのに役立つページが用意されています(これらのページにアクセスするには、[Interoperability]、[相互運用] の順にクリックします)。DTL エディタにも、特定の変換に使用するドキュメント構造を表示する機能が用意されています。

(詳細は、XML ドキュメントの場合と異なります。"プロダクション内での XML 仮想ドキュメントのルーティング" を参照してください。)

3.5 仮想ドキュメント・クラス

仮想ドキュメントを操作する場合は、メッセージ・クラスを作成する必要がありません。InterSystems IRIS にはメッセージ・クラスが用意されています。例えば、X12 ドキュメント、XML ドキュメントを転送するためのクラスがそれぞれ 1 つずつあります。ビジネス・ホスト・クラスでは、自動的に、適切なメッセージ・クラスが使用されます。

これらのメッセージは、総称して、仮想ドキュメントと呼ばれています。

仮想ドキュメント・クラスには、InterSystems IRIS でメッセージを処理するために必要な情報を転送するためのプロパティが用意されています。これらのプロパティには以下が含まれます。

DocType プロパティ

次のように 2 つの部分からなる文字列であるドキュメント・タイプを指定します。

```
category:structure
```

説明 :

- category はスキーマ・カテゴリの名前です。
- structure は、参照される category 内のドキュメント構造の名前です。

このプロパティは管理ポータルで表示できます。例えば、フィルタリングに使用できます。

RawContent プロパティ

未加工のメッセージの最初の 32 KB が含まれています。

このプロパティは、管理ポータルにも表示できます。形式が不正なメッセージの分析や報告に役立ちます。

このプロパティにはインデックスが付けられていないことに注意してください。このプロパティを SQL 検索クエリ内で使用した場合は、そのクエリがあまり有効に機能しない可能性があります。このプロパティにプログラム (DTL など) からアクセスすることはお勧めできません。代わりに、仮想プロパティ・パスを使用して必要なデータにアクセスしてください。

BuildMapStatus プロパティ

未加工のメッセージの内容を特定の DocType にマップする最近の試みが成功したか失敗したかを示す %Status 値が含まれています。BuildMapStatus はコード内で以下のようにテストできます。

- ObjectScript ではマクロ `$$$ISOK(myX12Message.BuildMapStatus)`、Basic ではメソッド `$SYSTEM.Status.IsOK(myMessage.BuildMapStatus)` を使用します。このテストで真の値が返された場合、BuildMapStatus には成功値が格納されています。
- ObjectScript ではマクロ `$$$ISERR(myMessage.BuildMapStatus)`、Basic ではメソッド `$system.Status.IsError(myX12Message.BuildMapStatus)` を使用します。このテストで真の値が返された場合、BuildMapStatus にはエラー値が格納されています。

BuildMapStatus のエラー・コードの詳細を表示するには、“ポータル・ツール”の章で説明されている [スキーマ構造] ページを使用します。

仮想ドキュメント・クラスでは、InterSystems IRIS で特定の形式と DocType の仮想プロパティ・パスを解釈するために必要なロジックも提供されます。このクラスでは、仮想ドキュメント内で (状況に応じて) 値を取得または設定するために使用可能な以下のインスタンス・メソッドが提供されます。

GetValueAt() メソッド

仮想プロパティ・パスを所与として、メッセージ内の仮想プロパティの値を返します。

このメソッド (および次のメソッド) は、メッセージにアクセスしてカスタム・コードを実行可能なプロダクション内の任意の場所 (BPL <code> 要素内など) から呼び出すことができます。

SetValueAt() メソッド

仮想プロパティ・パスと値を所与として、メッセージ内に値を設定します。GetValueAt() に関するコメントを参照してください。

3.6 ターミナルでの仮想プロパティ・パスのテスト

特に、構文に精通している場合は、仮想ドキュメント・プロパティ・パスをビジネス・プロセスやデータ変換などで使用する前にターミナルでテストできると便利です。そのためには、以下のように操作します。

1. 対応するスキーマを InterSystems IRIS にロードします (まだロードされていない場合)。これを実現するには、“ポータル・ツール”の章を参照してください。
2. ターミナルまたはテスト・コード内:
   a. 適切なドキュメントのテキストを含む文字列を作成します。
   b. 該当する仮想ドキュメント・クラスの ImportFromString() メソッドを使用して、この文字列から仮想ドキュメントのインスタンスを作成します。

      このクラスの一覧表を以下に示します。

ドキュメント・タイプ	仮想ドキュメント・クラス
EDIFACT	EnsLib.EDI.EDIFACT.Document
XML	EnsLib.EDI.XML.Document

   c. このインスタンスの DocType プロパティを設定します。
   d. このインスタンスの GetValueAt() インスタンス・メソッドを使用します。

4

プロダクション内での仮想ドキュメントの使用法

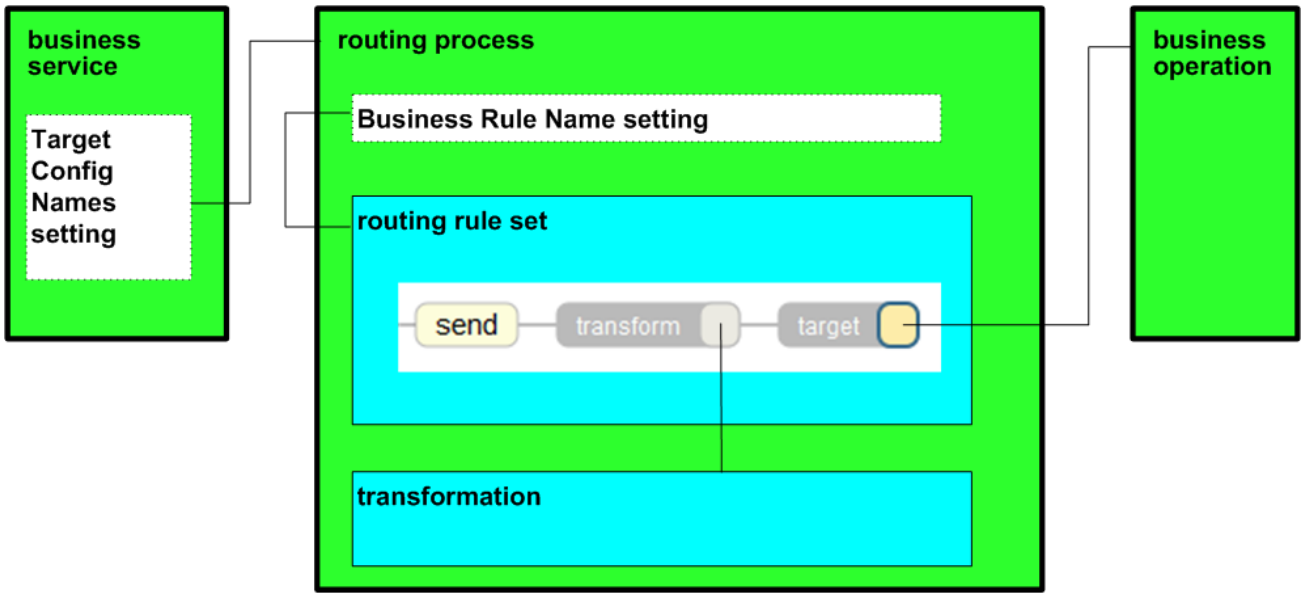
この章では、プロダクション内での仮想ドキュメントの使用方法について簡単に説明します。

4.1 概要

次のように、InterSystems IRIS® には、仮想ドキュメントの種類ごとに、ビジネス・ホストとして使用可能なクラスのセットが用意されています。

項目	[クラス]
ビジネス・サービス	InterSystems IRIS には、関連するアダプタが異なる 1 つ以上の特殊なビジネス・サービス・クラスが用意されています。これらのクラスを使用するためのコーディングは必要ありません。
ビジネス・プロセス	· 標準仮想ドキュメント・ルーティング・プロセス・クラスの EnsLib.MsgRouter.RoutingEngine · このルーティング・プロセス・クラスの特殊なサブクラス これらのクラスを使用するためのコーディングは必要ありません。
ビジネス・オペレーション	InterSystems IRIS には、関連するアダプタが異なる 1 つ以上の特殊なビジネス・オペレーション・クラスが用意されています。例えば、X12 ドキュメントの場合は、ファイル出力、TCP 出力、および FTP 出力専用のクラスがあります。これらのクラスを使用するためのコーディングは必要ありません。

ビジネス・ホスト・クラスには構成可能なターゲットが含まれています。それらのいくつかを下の図に示します。



InterSystems IRIS に該当するスキーマをロードしなければ、仮想ドキュメントの検証や解釈に使用できません。

4.2 仮想ドキュメント用のビジネス・サービス

InterSystems IRIS には、仮想ドキュメント形式ごとに、関連する入力アダプタが異なる 1 つ以上の特殊なビジネス・サービス・クラスが用意されています。これらのクラスはプロダクションにビジネス・サービスを追加するために使用します。いくつかの例外を除いて、これらのビジネス・ホストには以下の構成可能な設定が含まれています。

- [Docスキーマカテゴリ] ー 形式に忠実で、このビジネス・サービスに対して期待されるメッセージと矛盾しないスキーマ・カテゴリを指定します。この設定により、ビジネス・サービスでメッセージの DocType を設定するために必要な情報の一部が提供されます。具体的には、この設定で、DocType の category 部分が決定されます。

  ビジネス・サービスでは、DocType の structure 部分を決定するためにもドキュメントが解釈され、そのプロパティの値が設定されます。

- [テーブルクラス検索] ー このビジネス・サービスに対して期待されるメッセージと矛盾しない適切な検索テーブル・クラスを指定します。ビジネス・サービスでは、処理するメッセージごとにインデックスを付けるためにこれが使用されます。

- [ターゲット構成名] ー このビジネス・サービスからメッセージを送信するビジネス・ホストを指定します (複数可)。

- [検証] ー このサービスに対して実行する検証の種類を表す文字列を指定します。

  組み込み検証プロセスの詳細は、後述する “仮想ドキュメントのメッセージ検証” を参照してください。

  プロダクションの定義プロセスの一部として、適用可能なすべての EDI または XML スキーマが InterSystems IRIS にロードされ、仮想ドキュメントの検証に使用されます。

仮想ドキュメント・ビジネス・サービスによっては、他にも、構成可能なターゲットがあります。例えば、FTP を使用している場合は、常時接続経由で応答メッセージを送信できるため、一部のビジネス・ホストには **[リプライターゲット構成名]** 設定があります。

このようなビジネス・サービスには、他にも、EDI または XML 形式に固有の詳細を指定するための設定が数多くあります。

4.3 仮想ドキュメント用のビジネス・プロセス

InterSystems IRIS には、仮想ドキュメントで使用するための特殊なビジネス・プロセスが用意されています。これらのプロセスには多くの共通点があります。また、それぞれがルーティング・プロセスとして設計されています。ルーティング・プロセスは、以下のキー項目を使用して、メッセージを転送したり、変換したりします。

- ルーティング・ルールは、メッセージをその内容に基づいて宛先に転送します。
- スキーマ・カテゴリは、メッセージの内容にアクセスして検証する手段を提供します。
- データ変換は、宛先に合わせてメッセージを調整します。

これらのルーティング・プロセスには以下の構成可能な設定 (一部) があります。

- [ビジネスルール名] ー このプロセスで使用されるビジネス・ルール・セットの名前。

  プロダクションの定義プロセスの一部として、このビジネス・ルール・セットとそれに必要なデータ変換を作成します。
- [検証] ー このプロセスに対して実行する検証の種類を表す文字列を指定します。

  組み込み検証プロセスの詳細は、後述する“仮想ドキュメントのメッセージ検証”を参照してください。
- [不正なメッセージハンドラ] ー 検証プロセスの判断に従ってこのプロセスから不正なメッセージが送信されるビジネス・ホストの名前。

ルーティング・プロセスによっては、他にも、応答メッセージを処理するためなどの構成可能なターゲットがあります。そのターゲットには、仮想ドキュメント形式に固有の詳細を指定するための追加の設定が含まれます。

4.4 仮想ドキュメント用のビジネス・オペレーション

InterSystems IRIS には、仮想ドキュメント形式ごとに、関連する出力アダプタが異なる 1 つ以上の特殊なビジネス・オペレーション・クラスが用意されています。これらのクラスはプロダクションにビジネス・オペレーションを追加するために使用します。いくつかの例外を除いて、これらのビジネス・ホストには以下の構成可能な設定が含まれています。

- [テーブルクラス検索] ー このビジネス・サービスに対して期待されるメッセージと矛盾しない適切な検索テーブル・クラスを指定します。ビジネス・オペレーションでは、処理するメッセージごとにインデックスを付けるためにこれが使用されます。
- [検証] ー このオペレーションに対して実行する検証の種類を表す文字列を指定します。

  組み込み検証プロセスの詳細は、後述する“仮想ドキュメントのメッセージ検証”を参照してください。

このようなビジネス・オペレーションには、他にも、EDI または XML 形式に固有の詳細を指定するための設定が数多くあります。

5
検索テーブルの定義

この章では、仮想ドキュメントの検索テーブルを定義する方法について簡単に説明します。

これらのタスクはプロダクションが存在しているネームスペースで実行する必要があります。検索テーブルを作成するときに、予約パッケージ名を使用しないでください。"プロダクションの開発" の "予約パッケージ名" を参照してください。

注釈　InterSystems IRIS® では、検索テーブル・クラスを追加する前に受信したメッセージに対し、遡ってインデックスを割り当てることはありません。検索テーブル・クラスの `BuildIndex()` メソッドを使用すると、このような既存のメッセージにインデックスを割り当てることができます。

5.1 検索テーブル・クラスの定義

検索テーブル・クラスを定義するには、以下の一般的な手順を使用します。

- プロダクションが含まれているネームスペース内でクラスを作成します。ここでも、予約パッケージ名は使用しないでください。"プロダクションの開発" の "予約パッケージ名" を参照してください。
- 次のような仮想ドキュメントのタイプに使用されるデフォルト検索テーブル・クラスのサブクラス (または必要に応じてコピー) を作成します。

ドキュメント・タイプ	デフォルト検索テーブル・クラス	メモ
X12	EnsLib.EDI.X12.SearchTable	X12 ドキュメント ID に対応する `Identifier` プロパティにインデックスを付けます。
EDIFACT	EnsLib.EDI.EDIFACT.SearchTable	EDIFACT ドキュメント ID に対応する `Identifier` プロパティにインデックスを付けます。
XML	EnsLib.EDI.XML.SearchTable	XML ドキュメントのルート要素の名前にインデックスを付けます。

- 必要に応じて、このサブクラスに仮想プロパティを定義する XData ブロックを含めます。以降の項で詳細を説明します。

  この XData ブロックには、選択されたスーパークラスによってインデックスが付けられた仮想プロパティに関する詳細を含める必要がありません。例えば、検索テーブル・クラスの一部では、`Identifier` という名前のプロパティにインデックスが付けられます。これらのクラスのいずれかをサブクラス化する場合は、XData ブロックに `Identifier` を含める必要がありません。選択的でないプロパティを指定することで、検索の効率を向上させることができます。プロパティが選択的でないのは、多くのメッセージがそのプロパティに対して同じ値を持つ場合です。InterSystems

IRIS でメッセージを検索する場合、選択的でないプロパティの値をテストする前に検出されるメッセージの数を選択的プロパティによって制限すると、検索の効率を向上させることができます。プロパティが選択的でないことを指定するには、XData ブロックで `Unselective="true"` を指定します。

- 検索テーブル・クラスが複数のネームスペースにマッピングされている場合は、その検索テーブル・クラスをそれぞれのネームスペースでコンパイルして、各ネームスペースに対するローカルのメタデータが最新状態になるようにします。

  重要　検索テーブルのメタデータは、相互運用対応ネームスペースごとのデフォルト・グローバル・データベースに配置されます。そのため、検索テーブル・クラスを変更しても、クラスがマッピングされているすべてのネームスペース内のメタデータが更新されるわけではありません。そのため、これらのネームスペースごとに検索テーブル・クラスを再コンパイルする必要があります。

このクラスをコンパイルすると、検索テーブル・プロパティ別にローカル・メタデータが動的に取得され、そのプロセスが InterSystems IRIS ホストとして実行されている場合は、そのメタデータがキャッシュに格納されます。プロパティのメタデータが存在しない場合 (例えばマッピングされた検索テーブル・クラスに新しいプロパティのローカル・メタデータが保持されていない場合など)、このクラスは他のすべてのプロパティをインデックス化してから、メタデータが存在しなかったことを示すエラーを返します。同じように、メッセージ本文が削除されると、対応するエントリが検索テーブルから削除されます。ユーザ側での操作は必要ありません。

検索テーブル・クラスを使用する場合は、該当するビジネス・ホストの構成オプション ([**テーブルクラス検索**]) として指定します。そのビジネス・ホストがメッセージを処理する際に、構成済みの検索クラスを使用してこれらのメッセージをインデックス化します。

5.1.1 検索テーブル・クラスに関する XData 詳細

検索テーブル・クラスを作成する目的は、メッセージ・ブラウザ、ルール・エディタ、および管理ポータルのその他の部分で検索とフィルタの対象とする仮想プロパティごとに 1 つずつの検索テーブル・エントリを提供することです。これを実現するために、以下のスタブと同様の XData ブロックを検索テーブル・クラスに追加します。

Class Member

```
XData SearchSpec [ XMLNamespace="http://www.intersystems.com/EnsSearchTable" ]
{
<Items>
   <Item DocType="doctype1" PropName="name1" PropType="type1" StoreNulls="boolean"
     Unselective="true">path1</Item>
   <Item DocType="doctype2" PropName="name2" PropType="type2" StoreNulls="boolean">path2</Item>
<  <Item DocType="doctype3" PropName="name3" PropType="type3" StoreNulls="boolean">path3</Item>
</Items>
}
```

説明 :

- path1、path2、path3 などは仮想プロパティ・パスです。

  これらのそれぞれが文字列式になっています。文字列式には、以下の要素を任意に組み合わせたものを含めることができます。

  - 二重引用符で囲まれたリテラル文字。
  - {} または [] で囲まれた仮想プロパティ構文。これは、X12 ドキュメントの特定のフィールドの値に解決します。角かっこは中かっこことは異なり、segment:field の組み合わせを囲みます。この場合、含まれるドキュメント構造を識別する必要がありません。中かっこの構文で解決する場合は、ドキュメント構造を把握している必要があります。
  - 連結に使用するアンダースコア (_) などの ObjectScript 文字列演算子。
  - $PIECE または $EXTRACT などの関数。

- doctype1、doctype2、doctype3 などは DocType 識別子です。これらのそれぞれが次のようにコロンで区切られたスキーマ・カテゴリ名とドキュメント構造名になっています。

  ```
  category:structure
  ```

  category がない場合は、任意のスキーマが一致します。structure がない場合は、任意の構造が一致します。"" の値 (空白の文字列) は、任意のカテゴリ・スキーマおよび任意のドキュメント構造と一致します。

  この <Item> の部分で、特定のプロパティ・パスにインデックスを付ける DocType が指定されます (複数可)。

- name1、name2、name3 などは選択された仮想プロパティ名です。

  これは、管理ポータルに表示される名前です。リスト内で他の項目と一緒に表示されたときに区別しやすい文字列を選択します。

  複数の <Item> 要素に同じ名前を割り当てた場合は、その名前を検索で選択すれば、同じ名前を持つすべてのエントリが検索されるという便利な相加効果があります。

- type1、type2、type3 などはオプションのタイプ識別子です。以下のリテラル値のいずれかを指定します。

  - `String:CaseSensitive`
  - `String:CaseInsensitive`
  - `Integer`
  - `Numeric`
  - `Boolean`
  - `DateTime:ODBC`
  - `DateTime:HL7` (EDIFACT メッセージを転送する仮想ドキュメント専用にサポートされています)

  `String:CaseSensitive` がデフォルトです。

- boolean は、ドキュメント内で空のフィールドが見つかった場合の処理を制御するオプション・フラグです。このフラグが真の場合は、空の文字列に有効なポインタが返されます。このフラグが偽の場合は、[見つかりません] ステータスと NULL ポインタが返されます。

  1 (真を意味する) と 0 (偽を意味する) のどちらかを指定します。デフォルトは 0 です。

- `Unselective="true"` は、プロパティの値が少数のメッセージを通常選択しないことを示します。InterSystems IRIS では、この情報を使用して検索の効率を向上させます。デフォルト値は `Unselective="false"` です。

重要　InterSystems IRIS は、仮想ドキュメントをインデックス付け (つまり、検索テーブルに追加) すると、縦棒 (|) をプラス記号 (+) に置き換えます。検索テーブルを使用して内容を検索する際には、このことを考慮してください。例えば、`my|string` という文字列を含むメッセージを検索するには、検索条件として `my+string` を使用します。

5.2 カスタム検索テーブル・クラスの定義

この章に説明する検索テーブルの基本メカニズムでは、必要に応じたメッセージのインデックス化が実行できない場合もあります。そのような場合は、カスタム検索テーブル・クラスを定義して使用することができます。

このクラスでは、2 種類のプロパティを定義できます。このトピックでは、これらのプロパティを標準プロパティ (検索テーブルに格納されているもの) および仮想プロパティ (検索テーブルには格納されていない代わりに、実行時に取得され

るもの）と呼んでいます。どちらの種類のプロパティでも、インデックス化する場合とインデックス化しない場合があります。プロパティをインデックス化すると、ディスク容量がより多く消費されますが、そのプロパティに対するクエリ実行が高速化されます。管理ポータルではインデックス化したプロパティをインデックス化していないプロパティの上にグループで表示するため、ユーザは適切に選択できます。

カスタム検索テーブル・クラスを定義するには、クラスを以下のように定義します。

- Ens.CustomSearchTable を拡張します。

  このクラスによって 1 つの標準クラス・プロパティ、DocId が定義されます。これはインデックス化されています。

- 必要に応じて追加のクラス・プロパティを定義し、そのクラス・プロパティのインデックスを追加します。以下に例を示します。

```
Property Type As %String(COLLATION = "EXACT");

Index Type On Type [ Type = bitmap ];
```

  コレクション・プロパティは、現在はクエリ生成メカニズムによって直接サポートされていません。コレクション・プロパティに対しては、以下に説明する GetVirtualPropertyList() メソッド・メカニズムを使用します。

- オプションとして、必要に応じて GetPropertyList() メソッドを実装します。

```
classmethod GetPropertyList(Output pIndexedProperties As %List, Output pProperties As %List) as
%Status
```

  説明：

  - pIndexProperties は、インデックス化が必要な標準プロパティの $LISTBUILD リストです。
  - pProperties は、定義する標準プロパティの $LISTBUILD リストです。

  この手順の目的は、標準プロパティとして使用するクラス・プロパティを指定し（このリストの前に示した定義を参照してください）、その中でインデックス化が必要なものを指定することです。

  デフォルトでは、このメソッドが生成され、InterSystems IRIS では検索テーブル・クラスのすべてのクラス・プロパティが標準プロパティとして使用され、すべてがインデックス化されます。ただし、private、internal、transient、および multidimensional の各プロパティを除きます。

  仮想プロパティの場合は、代わりに（または追加として）GetVirtualPropertyList() を実装します。

- オプションとして、必要に応じて GetVirtualPropertyList() メソッドを実装します。

```
classmethod GetVirtualPropertyList(Output GetVirtualPropertyList As %List,
                                   Output pVirtualProperties  As %List)
                                   as %Status
```

  説明：

  - GetVirtualPropertyList は、インデックス化が必要な仮想プロパティの $LISTBUILD リストです。
  - pVirtualProperties は、定義する仮想プロパティの $LISTBUILD リストです。

  この手順の目的は、仮想プロパティとして使用するクラス・プロパティを指定し、その中でインデックス化が必要なものを指定することです。

  標準プロパティの場合は、代わりに（または追加として）GetPropertyList() を実装します。

- GetVirtualPropertyList() を実装した場合は、GetVirtualProperty() メソッドも実装します。このメソッドは、ドキュメント ID と仮想プロパティ名が指定されている場合、以下のように仮想プロパティの値を返す必要があります。

```
classmethod GetVirtualProperty(pDocID As %String,
                               pPropName As %String,
                               Output pPropValue As %String,
                               ByRef pUserArgs) as %Status
```

説明：

- pDocID は、カスタム検索テーブル内のドキュメントの ID です。
- pPropName は、仮想プロパティの名前です。
- pPropValue は、そのプロパティの値です。
- pUserArgs は任意の引数を指定します。

- OnIndexDoc() メソッドを実装します。

```
ClassMethod OnIndexDoc(pDocObj As %Persistent, pSearchTable As Ens.CustomSearchTable) As %Status
```

このメソッドによって、検索テーブル内でプロパティから指定した行を、指定されたメッセージに入力する方法が指定されます。

OnProcessCondition() および追加オプションは、**Ens.CustomSearchTable** のクラス・リファレンスを参照してください。

例については、**Demo.CustomSearchTable.Sample** を参照してください。

5.3 検索テーブルの管理

クラス **Ens.DocClassMap** は、(カスタム検索テーブルを含め) すべての検索テーブルを管理します。これは、グローバル (^Ens.DocClassMap) への書き込みと読み込みを行います。このグローバルは、データを含める検索テーブルをメッセージ・クラス別に指定します。このグローバルは、直接編集しないでください。

InterSystems IRIS Interoperability クラスは、メッセージ本文が削除されたときに、このクラスを使用して検索テーブルのエントリを削除します。

通常は、このクラスを直接使用する必要はありません。ただし、^Ens.DocClassMap 内のデータが失われたり壊れた場合は、このクラスの RebuildMap() メソッドを使用してグローバルを再作成します。詳細は、**Ens.DocClassMap** のクラス・リファレンスを参照してください。

5.4 管理ポータルで使用するクエリのカスタマイズ

ユーザが管理ポータルの **[メッセージ・ビューワ]** ページと **[メッセージ・バンクのメッセージ・ビューワ]** ページでメッセージを検索すると、クエリが生成されて使用されます。高度な手法として、InterSystems IRIS によるクエリ生成方法をカスタマイズできます。そのためには、以下の一般的な手順を実行します。

- **EnsPortal.MsgFilter.AbstractAssistant** のサブクラスを定義します。詳細は、そのクラスのクラス・リファレンスを参照してください。
- クラスの名前を、メッセージ・ビューワでは ^EnsPortal.Settings("MessageViewer","AssistantClass") に、メッセージ・バンク・ビューワでは ^EnsPortal.Settings("MsgBankViewer","AssistantClass") に設定します。

6

メッセージ検証の制御

ASTMドキュメント以外の仮想ドキュメントの場合は、InterSystems IRIS® では、不正なメッセージを処理するビジネス・ホストを追加するためのオプションを使用したメッセージ検証を利用できます。必要に応じて、検証ロジックを上書きすることもできます。この章では、その詳細について説明します。

6.1 メッセージ検証の概要

ASTM ドキュメント以外の仮想ドキュメントでは、1 つ以上の特殊なビジネス・ホスト・クラスに **[検証]** 設定が含まれており、ビジネス・ホストで受信したメッセージを処理する前に検証する方法を指定するために使用します。

メッセージが検証に合格した場合は、ビジネス・ホストがそのメッセージを指定された標準ターゲット (複数可) に送信します。

ドキュメントが検証に合格しなかった場合は、ビジネス・ホストの種類によって詳細が異なります。

- ビジネス・サービスまたはビジネス・オペレーションはどこにもメッセージを送信しません。
- ルーティング・プロセスには、ビジネス・ホストの名前にすべき [不正なメッセージ・ハンドラ] という追加の設定が含まれています。ドキュメントが検証に合格しなかった場合は、この設定の指定に従って、ルーティング・プロセスがそのドキュメントを不正メッセージ・ハンドラに転送します。不正メッセージ・ハンドラが存在しない場合は、ルーティング・プロセスがドキュメントを転送せずに、ログにエラーを書き込みます。ルーティング・プロセスには、アラートの送信を可能にする設定を含めることもできます。

6.2 基本検証オプションとロジック

このトピックでは、[検証] 設定の許容値と InterSystems IRIS によるメッセージの検証方法について説明します

値	意味
d	ドキュメントの DocType プロパティを調べ、値が指定されているかどうかを確認します。
m	ドキュメントのセグメント構造が適切に構築されており、ドキュメントの DocType プロパティで識別されたスキーマを使用して解析できることを確認します。
dm	d と m の両方が適用されます。

値	意味
(空白文字列)	ビジネス・ホストが検証を省略し、すべてのドキュメントをルーティングします。ほとんどの仮想ドキュメント形式において、これがデフォルト設定です。

6.2.1 d 検証フラグ

d フラグが [**検証**] 文字列に含まれている場合は、(ビジネス・サービスによって設定された) メッセージ内で指定された [**スキーマ・カテゴリ**] が検査され、メッセージの DocType と比較されます。メッセージ内の [**スキーマ・カテゴリ**] が空の場合は、自動的に、そのメッセージが不正として宣言されます。ただし、[**スキーマ・カテゴリ**] が空白ではないが、メッセージの DocType と一致しない場合は、m などの別の [**検証**] フラグが定義されていれば、検証が続行されます。

6.2.2 m 検証フラグ

m フラグが [**検証**] 文字列に含まれている場合は、メッセージを検証する方法と不正として宣言する方法のどちらかが検索されます。詳細は、仮想ドキュメント・タイプによって異なります。

6.3 検証ロジックの上書き

仮想ドキュメント・ルーティング・プロセス (EnsLib.MsgRouter.VDocRoutingEngine) とそのサブクラスを通して、デフォルトの検証ロジックが提供されます。特殊な仮想ドキュメントのビジネス・サービス・クラスとビジネス・オペレーション・クラスのほとんどでも検証ロジックが提供されます。この検証ロジックは上書きすることができます。そのために、該当するクラスのサブクラスを作成して使用します。

6.3.1 ルーティング・プロセス・クラス内の検証ロジックの上書き

InterSystems IRIS Interoperability クラスのサブクラスを作成してから、OnValidate() メソッドを上書きする場合は、次の手順を実行する必要があります。

- [**検証**] 設定で受け入れられた値のリストを拡張または置換すること。
- ルーティング・プロセスによるドキュメントの検証方法を決定すること。[**検証**] のオプションで制御できます。

OnValidate() メソッドをオーバーライドする場合は、同じサブクラス内の [**検証**] プロパティの定義もオーバーライドできます。以下のような詳細情報に注意してください。

- `InitialExpression` 値は、[**検証**] 構成設定のデフォルトを指定します。
- [**検証**] プロパティ定義に先行するコメントは、[**検証**] 設定のツール・ヒントとして使用されます。`///` 規則を使用し、最後のコメントとプロパティ定義の間には空白行を残さないでください。これにより、管理ポータル・ユーザは入力されたコメントをツール・ヒントとして表示できます。

6.3.2 ビジネス・サービス・クラスまたはビジネス・オペレーション・クラス内の検証ロジックの上書き

仮想ドキュメントのビジネス・サービス・クラスとビジネス・オペレーション・クラスのそれぞれで、上書きが可能な Validation プロパティと OnValidate() メソッドが提供されます。デフォルトで、このプロパティがいずれかのビジネス・サービスまたはビジネス・オペレーションの設定として公開されることはなく、これらのクラス内で OnValidate() アクティビティが実行されることはありません。ドキュメントを、いつものようにルーティング・エンジンで検証するのではなく、インタフェースの受信側または送信側で検証する場合は、この動作を変更できます。変更を実行するには、以下の手順に従います。

1. 上記の **[検証]** プロパティと OnValidate() メソッドをオーバーライドするための手順を実行します。
2. ユーザが検証のタイプを選択できるようにする場合は、Validation プロパティも設定として追加します。"プロダクションの開発" の "設定の追加と削除" を参照してください。

6.4 不正メッセージ・ハンドラの定義

ルーティング・プロセスには、**[不正なメッセージ・ハンドラ]** という設定があります。この設定の目的は、ビジネス・プロセスの不正メッセージ・ハンドラに従って、不正なことが判明したメッセージをプロセスから送信するビジネス・ホストを指定することです。不正メッセージ・ハンドラを定義するには、まず、不正なメッセージの処理方法を決定します。通常は、ビジネス・オペレーションを作成します。このビジネス・オペレーションは以下のどちらかまたは両方を実行できます。

- (ファイル・アダプタ経由) メッセージの内容をファイルに書き込みます。
- (構成設定によって異なる) 不正なメッセージが見つかったときにアラートをトリガします。

ビジネス・プロセスは、検証されたメッセージの通常のターゲットの代わりに、このビジネス・ホストに不正なメッセージを送信することに注意してください。

メッセージが不正で、**[不正なメッセージ・ハンドラ]** 設定が指定されていない場合は、ルーティング・プロセスが検証シーケンスを停止して、メッセージを送信しません。

7

カスタム・スキーマ・カテゴリの作成

InterSystems IRIS® でカスタム・スキーマ・カテゴリを作成しなければならない場合があります。この章では、その詳細について説明します。

重要　組み込みスキーマ・カテゴリ定義は絶対に編集しないでください。カスタム・スキーマが必要な場合は、組み込みスキーマ・カテゴリ定義をスキーマ・ベースとして使用するカスタム・スキーマ・カテゴリ定義を作成します。

これらのタスクはプロダクションが存在しているネームスペースで実行する必要があります。

7.1 カスタム・スキーマ・カテゴリが必要な場合

EDIFACT および X12 標準は拡張可能です。

例えば、一般的な方法は、他の標準のドキュメント構造にカスタム・セグメントを追加することです。

ロードした標準スキーマに従っていないドキュメントを操作する場合は、以下のどちらかを実行するために対応するカスタム・カテゴリを作成する必要があります。

- 構造的なドキュメントの検証の実行
- BPL、DTL、およびルーティング・ルール構文でのセグメントおよびフィールド・パス名の使用

7.2 カスタム・スキーマ・カテゴリの作成方法

カスタム・スキーマ・カテゴリを作成する一般的な方法が 4 つあります。

- 最も厳密にニーズを満たしているスキーマ・ファイルのコピーを作成します。そのファイルを編集します。その後で、“ポータル・ツール”の章の説明に従って、そのファイルをインポートします。
- スタジオで、最も厳密にニーズを満たしているスキーマ・カテゴリ定義を開きます。**[ファイル]→[名前を付けて保存]** を使用して新しい定義を作成します。その後で、そのコピーを編集します。“InterSystems IRIS 内でのスキーマ・カテゴリの構文”の節で、この定義の構文について詳しく説明します。
- スタジオ・ウィザードを使用して、新しいスキーマ・カテゴリ定義を作成してから、それを編集します。″プロダクション内での X12 ドキュメントのルーティング″の“カスタム X12 スキーマの作成”を参照してください。同じプロセスを EDIFACT に対しても使用できます。

7.3 InterSystems IRIS 内でのスキーマ・カテゴリの構文

InterSystems IRIS では、EDIFACT および X12 用のスキーマ・カテゴリ定義を表現するために同じ XML ベースの構文が使用されます。

カスタム・スキーマ・カテゴリを作成する際、カスタム・セグメント (Z セグメント) を定義し、これらのセグメントを含む可能性のあるメッセージ・タイプとメッセージ構造を記述することにより、既存のスキーマを補足します。これを行うために必要なことは、XML 要素 <MessageType>、<MessageStructure>、および <SegmentStructure> を使用することのみです。

カスタムのスキーマ・カテゴリ定義は、組み込みの定義より単純で、含まれる文は少なくなります。ベース・カテゴリ内のすべてがカスタムのスキーマ・カテゴリ定義に含まれます。標準メッセージ・タイプの定義を繰り返す必要はありません。カスタム・メッセージ・タイプを定義するだけです。これを行うための規則は以下のとおりです。

定義対象	定義方法
カスタムのスキーマ・カテゴリ定義	<Category>
カスタム・セグメント	<SegmentStructure>
カスタム・セグメントを含むメッセージ構造	<MessageStructure>
カスタム・セグメントのあるメッセージ構造を含むメッセージ・タイプ。メッセージ・タイプは以下を指定します。 ・ 送信するメッセージ構造 ・ 応答で予定されるメッセージ構造	<MessageType>

7.3.1 <Category>

<Category> 要素は、カスタム・スキーマ・カテゴリを記述する XML ドキュメントの最上位のコンテナです。

以下のテーブルは、<Category> の属性を示しています。

属性	説明	値
name	[スキーマ構造] ページの利用可能なスキーマ・カテゴリのリストに表示される名前。	文字列。便宜上、スキーマ・ファイルの名前を使用します。
std	1 (真) の場合、この <Category> ブロックは標準の HL7 スキーマ・カテゴリを示します。デフォルトは、0 (偽) です。	標準のスキーマ・カテゴリ定義専用。カスタム・スキーマには std を使用しないでください。
base	このカスタム・スキーマ・カテゴリのベースとなるスキーマ・カテゴリを指定します。スキーマ・ベース内の各定義が自動的にカスタム・カテゴリに含まれ、カスタム・スキーマ・カテゴリ内の文が単純にベースに追加されます。	別のスキーマ・ファイル内の <Category> ブロックを使用して定義された標準またはカスタム・スキーマ・カテゴリの name。

7.3.2 <SegmentStructure>

<Category> 要素には、1 つ以上の <SegmentStructure> 要素を含めることができます。各 <SegmentStructure> 要素では、カスタム・セグメントの構造を定義します。

以下のテーブルは、<SegmentStructure> の属性を示しています。

属性	説明	値
name	[スキーマ構造] ページの利用可能なセグメント構造のリストに表示される名前。	3-文字の文字列。.
description	[スキーマ構造] ページおよび [ドキュメント・ビューア] ページのツールのヒントに表示される、セグメントのコンテンツを説明するテキスト。	文字列

7.3.3 <SegmentSubStructure>

<SegmentStructure> 要素には、1 つ以上の <SegmentSubStructure> 要素を含めることができます。各 <SegmentSubStructure> 要素は、カスタム・セグメントの 1 つのフィールドを、上から下に順に説明します。以下は、<SegmentSubStructure> 要素の構文の例です。

XML

```
<SegmentSubStructure piece='6' description='Encounter Number' symbol='!'
                     length='12' required='R' ifrepeating='0'/>
```

以下のテーブルは、<SegmentSubStructure> の属性を示しています。

属性	説明	値
piece	ユーザがこのフィールドを含むセグメントの詳細を表示するよう要求したときに [スキーマ構造] ページに表示される数値。この数値を使用して、仮想プロパティ・パスのフィールドを指定できます。	整数。<SegmentStructure> 内の各 <SegmentSubStructure> は、1 から始まり、1 ずつ増加するよう順に piece 値を使用する必要があります。
codetable	このフィールドで有効な値を列挙するコード・テーブル。カスタム・スキーマでは通常この属性は使用されません。	<CodeTable> ブロックを使用して定義されたコード・テーブルの name。
datastruct	このフィールドの値の解釈方法を指定するデータ構造。カスタム・スキーマでは通常この属性は使用されません。	<DataStructure> ブロックを使用して定義されたデータ構造の name。
description	[スキーマ構造] ページおよび [ドキュメント・ビューア] ページのツールのヒントに表示される、フィールドのコンテンツを説明するテキスト。	文字列
symbol	セグメント内のこのフィールドの有無または繰り返しの要件を示す記号。 このフィールドはオプションです。これは [スキーマ構造] ページでインジケータとして機能します。フィールドの要件または繰り返しを実際に制御することはありません。required および ifrepeating を参照してください。	以下の単一文字 ・ ! は、1 のみを意味します。フィールドは表示する必要がありますが、1 回のみです。 ・ ? は、0 または 1 を意味します。フィールドは表示することができますが、最大 1 回です。 ・ + は、1 以上を意味します。フィールドは 1 回以上繰り返すことができます。 ・ * は、0 以上を意味します。フィールドは 0 回以上繰り返すことができます。 ・ & は、特定の条件においてのみ、フィールドを表示し、繰り返すことができることを意味します。

属性	説明	値
length	このフィールドに表示できる文字数の上限。	整数
required	このフィールドをセグメント内に表示する必要性。	以下の単一文字 ・ C は条件付きであることを意味します。 ・ O はオプションであることを意味します。 ・ R は必須であることを意味します。
ifrepeating	このフィールドをセグメント内で繰り返す可能性。	整数。0 は繰り返す可能性がないことを意味し、1 は繰り返す可能性があることを意味します。

7.3.4 <MessageStructure>

<Category> 要素には、1 つ以上の <MessageStructure> 要素を含めることができます。各 <MessageStructure> 要素は、メッセージ構造内のセグメントの数および配置についての仕様を指定します。以下は、<MessageStructure> 要素の構文の例です。

XML

```
<MessageStructure
  name='MFN_M03'
  definition='base:MSH~base:MFI~{~base:MFE~[~ZSI~]~base:OM1~[~base:Hxx~]~}'
  description='HNB MFN message'
  />
```

以下のテーブルは、<MessageStructure> の属性を示しています。

属性	説明	値
name	[スキーマ構造] ページの利用可能なメッセージ構造のリストに表示される名前。	3 文字の文字列、アンダースコア (_)、および 3 文字の文字列で構成。
definition	メッセージ構造内のセグメントの数および配置についての仕様。標準およびカスタムのメッセージ・セグメントの組み合わせを含む場合があります。以下の構文ルールを参照してください。	<SegmentStructure> を使用して定義された標準またはカスタムのメッセージ・セグメントの 3 文字の name 値を含む文字列。
description	[スキーマ構造] ページおよび [ドキュメント・ビューア] ページのツールのヒントに表示される、フィールドのコンテンツを説明するテキスト。	文字列

definition 文字列の構文規則は以下のとおりです。

- 文字列全体を 1 行に収めます。
- 各セグメントを左から右に向かって順に列挙します。
- セグメントを列挙する際、<SegmentStructure> で定義された name 値を使用します。
- セグメント間は ~ (チルダ) 文字で区切られます。
- セグメントまたはセグメントのブロックが繰り返される場合、繰り返し部分を {~ および ~} で囲みます。
- セグメントまたはセグメントのブロックがオプションの場合、オプション部分を [~ および ~] で囲みます。

definition 内では、name を単純にすることができます。この場合、InterSystems IRIS は、同じファイル内のカスタム <MessageStructure> ブロックの値を参照していると見なします。さらに、name は、スキーマ・ベースの標準メッセージ構

造を表すこともできます。これは、<Category> 要素に含まれる base 属性で指定されたファイルで定義されます。これを指定するには、name で以下の接頭語を使用する必要があります。

```
base:
```

これにより、InterSystems IRIS は別のファイルで適切な <SegmentStructure> を見つけることができます。他の外部ファイルを参照することはできません。スキーマ・ベースのみとなります。

7.3.5 <MessageType>

<Category> 要素には、1 つ以上の <MessageType> 要素を含めることができます。<MessageType> エントリは、カスタム・セグメントのあるメッセージ構造を含むメッセージ・タイプを定義します。<MessageType> 要素は、以下の 2 つの項目の単純なリストです。

- 送信するメッセージ構造
- 応答で予定されるメッセージ構造

以下は、<MessageType> 要素の構文の例です。

XML

```
<MessageType name='ADT_A31' structure='ADT_A01' returntype='base:ACK_A31'/>
```

以下のテーブルは、<MessageType> の属性を示しています。

属性	説明	値
name	[スキーマ構造] ページの利用可能なメッセージ・タイプのリストに表示される名前。	3-文字の文字列、アンダースコア (_)、および 3 文字の文字列で構成。
structure	送信するメッセージ構造。	<MessageStructure> を使用して定義された標準またはカスタムのメッセージ構造の name。
returntype	応答で予定されるメッセージ構造。これは、有効な ACK メッセージ構造である必要があります。returntype がスキーマ・ベースの値であることを確認してください。例 : `returntype="base:ACK"`	<MessageStructure> を使用して定義された標準またはカスタムのメッセージ構造の name。

structure 値は、現在のファイルの単純な name 値にすることができます。

代わりに、structure または returntype の値で、スキーマ・ベースの標準メッセージ構造を表すこともできます。これを指定するには、値に以下の接頭語を使用する必要があります。

```
base:
```

8

ポータル・ツール

管理ポータルには、スキーマ・カテゴリとその下位区分を表示したり、ドキュメントを表示したり、関連タスクを実行したりするためのページがあります。この章では、これらのページを使用する一般的な方法について説明します。

8.1 ツールへのアクセス

1. 管理ポータルで、該当するネームスペースに切り替えます。

   そのためには、タイトル・バーの **[切り替え]** をクリックして、対象のネームスペースを選択し、[OK] をクリックします。
2. [Interoperability] をクリックします。
3. **[相互運用]** をクリックします。
4. 目的の EDI 形式に対応したメニュー・オプションをクリックします。

8.2 スキーマ構造ページの使用法

InterSystems IRIS® には、X12、EDIFACT、および XML 用の各バージョンのスキーマ構造ページが用意されています。InterSystems IRIS for Health™ には、追加の形式用の各バージョンのスキーマ構造ページが用意されています。通常、スキーマの表示とインポートには、[スキーマ構造] ページを使用します。このページは、XML スキーマ以外のスキーマのエクスポートにも使用できます。

このページにアクセスするには:

1. **[相互運用性]→[相互運用]** をクリックしてから、目的の EDI 形式に対応するメニュー・オプションをクリックします。
2. **[スキーマ構造]** で終わるオプションをクリックしてから、**[実行]** をクリックします。

   InterSystems IRIS によって、このネームスペース内のこのタイプのスキーマが一覧表示されたページが表示されます。

ほとんどの場合、このページの左側にスキーマ・カテゴリのリストが表示されます。これらは、この形式に関連した、このネームスペース内のすべてのスキーマ・カテゴリです。

この領域を使用すれば、調査したいスキーマ・カテゴリを指定できます。スキーマ・カテゴリをクリックすると、右側のタブにそのカテゴリの詳細が表示されます。

EDI 形式によっては、ページの右側に以下のタブが表示される場合があります。

- [DocType構造] は、メッセージ構造内のセグメントの順番と分類を指定します。
- [**セグメント構造**] は、各セグメント内のフィールドを列挙します。
- [**データ構造**] は、複合データ・フィールドの内容を列挙します。
- [**コードテーブル**] は、列挙型フィールド内で使用可能な値を列挙します。

また、以下の操作の一部またはすべてを実行できるボタンが表示される場合があります。

- この EDI 標準の場合は、このネームスペースにスキーマをインポートします。そのためには、[**インポート**] をクリックして、[**参照**] でファイルを選択し、[OK] をクリックします。
- XML 以外のスキーマをファイルにエクスポートします。そのためには、対象のスキーマのカテゴリを選択し、[**エクスポート**] をクリックして、ファイル名を入力し、ファイル・タイプを選択して、[OK] をクリックします。
- このネームスペースからカスタム・スキーマを削除します。そのためには、カスタム・スキーマを選択し、[**削除**] をクリックして、[OK] をクリックします。InterSystems IRIS と共にインストールされている標準スキーマを削除することはできません。

  スキーマが直ちに削除されます。

  注意　　この操作を取り消すことはできません。
- 既存のスキーマに基づいて、新しいスキーマを作成します。そのためには、[**新規作成**] をクリックして、基本スキーマと新しいカスタム・スキーマ名を指定します。その後に、カスタム・スキーマの詳細を指定できます。

8.3 ドキュメント・ビューワ・ページの使用法

ドキュメント・ビューワの詳細は、"X12 ドキュメント・ビューワ・ページの使用法" を参照してください。これらのガイドで紹介されている情報は、XML、EDIFACT、または他のどの EDI 形式にも同様に適用できます。

仮想プロパティ・パスに関する構文ガイド

このリファレンスでは、仮想プロパティ・モデルをサポートしている構文の詳細について説明します。

下の表は、仮想プロパティ・パス構文を使用可能な場所を示しています。

構文	BPL	DTL	ビジネス・ルール	検索フィルタと検索テーブル
GetValueAt()	サポート対象	サポート対象	サポート対象外	サポート対象外
{} 構文	サポート対象 (<code> 要素と <sql> 要素内を除く)	サポート対象 (<code> 要素と <sql> 要素内を除く)	サポート対象	サポート対象
[] 構文	サポート対象外	サポート対象外	サポート対象	サポート対象
() 構文	サポート対象外	サポート対象	サポート対象	サポート対象外
<> 構文	サポート対象外	サポート対象外	サポート対象	サポート対象外

DocType が重要でない場合の仮想プロパティ・ショートカット

DocType が重要でない場合に使用可能な仮想プロパティ・ショートカットについて説明します。

詳細

仮想プロパティ・パスを識別する必要がある場合の多くは、コンテキストから特定の **DocType** は明白であるので、segment:field パスのみを識別する必要があります。また、特定の **DocType** は重要でなく、検索条件に一致するスキーマ定義の **DocType** が重要な場合があります。そのため、BPL、DTL、およびルーティングの各ルールに使用する仮想プロパティ構文には、以下のような、2 つの重要なショートカットがあります。

- 中かっこ

  {segment:field}

  中かっこの構文は、BPL、DTL、ビジネス・ルール (ルーティング・ルールを含む)、検索フィルタ、または検索テーブルで使用できます。中かっこ内の segment:field の組み合わせには、セグメント名とフィールド名、または数値による位置を使用できます。ただし、名前が使用できるのは、**DocType** (category:structure) が現在のコンテキストで明白に識別されている場合のみです。例えば、DTL データ変換では、ソースとターゲットの **DocType** が常に <transform> 要素で識別されます。

- 角かっこ

  [segment:field]

  角かっこの構文は、ビジネス・ルール、検索テーブル、および検索フィルタのみで使用できます。segment は名前でなければなりません。field は名前または数値です。名前を解決できるのは、**DocType** (category:structure) が実行時に既知である場合のみです。InterSystems IRIS は、特定のメッセージ構造またはスキーマが既知でなくても、数値 field を解決できます。角かっこ内に、パターンに一致する複数の結果がある場合、この構文は、一致するすべての値を含み、各値が <> 山かっこで囲まれた文字列を返します。

以下のショートカットは、ルーティング・ルールを定義する場合にのみ使用できます。

- 丸かっこ

  (multi-valued-property-path)

- 山かっこ

  <context|expression>

仮想プロパティ・パスの基本設定

仮想プロパティ・パスの作成方法について説明します。

概要

ほとんどの EDI 形式 (XML 仮想ドキュメントではなく) において、仮想プロパティ・パスは次のような構文になります。

```
segmentorsubsegmentID:fieldorsubfieldID
```

説明 :

- segmentorsubsegmentID は、セグメントまたはサブセグメントを参照します。この識別子は、“セグメント識別子とサブセグメント識別子”に記載された規則に従います。
- fieldorsubfieldID は、親セグメントまたはサブセグメント内のフィールドまたはサブフィールドを参照します。この識別子は、“フィールド識別子とサブフィールド識別子”に記載された規則に従います。

仮想プロパティ・パスは、特定のメッセージ構造に対して相対的で、その他のメッセージ構造では無効となります。管理ポータルには、パスを特定するのに役立つページが用意されています“ポータル・ツール”の章を参照してください。

重要　　XML 仮想ドキュメントの場合は、構文が異なります。″プロダクション内での XML 仮想ドキュメントのルーティング″を参照してください。

セグメント識別子とサブセグメント識別子

セグメントを参照するには、以下のどちらかを使用します。

- セグメントのシンボリック名。

  メッセージに同じ名前の複数のセグメントが含まれている場合は、識別子の最後に (n) を付加します。ここで、n は必要なセグメントの 1-ベースの位置です。
- メッセージに含まれているセグメントの数。この数は不明なことが多いため、この構文はあまり役に立ちません。

セグメントにはサブセグメントを追加できます。サブセグメントにはさらにサブセグメントを追加できます。サブセグメントを特定するには、以下の構文を使用します。

```
segmentID.subsegmentID
```

ここで、segmentID と subsegmentID はともに、セグメントを特定するための前述の規則に従います。シンボリック名と数値識別子を混在させることはできないことに注意してください。つまり、この構文の一部でシンボリック名を使用している場合は、構文全体でシンボリック名を使用する必要があります。同様に、この構文の一部で数値識別子を使用している場合は、構文全体で数値識別子を使用する必要があります。

フィールド識別子とサブフィールド識別子

フィールドを参照するには、以下のどちらかを使用します。

- フィールドのシンボリック名。

  親セグメントまたはサブセグメントに同じ名前の複数のフィールドが含まれている場合は、識別子の最後に (n) を付加します。ここで、n は必要なフィールドの 1-ベースの位置です。
- 親セグメントまたはサブセグメント内のフィールドの数

フィールドにはサブフィールドを追加できます。サブフィールドにはさらにサブフィールドを追加できます。サブフィールドを特定するには、以下の構文を使用します。

```
fieldID.subfieldID
```

ここで、fieldID と subfieldID はともに、フィールドを特定するための前述の規則に従います。シンボリック名と数値識別子を混在させることはできないことに注意してください。つまり、この構文の一部でシンボリック名を使用している場合は、構文全体でシンボリック名を使用する必要があります。同様に、この構文の一部で数値識別子を使用している場合は、構文全体で数値識別子を使用する必要があります。

例

下の仮想プロパティ・パスは、2 つ目の `NK1` セグメントの `Address` フィールドの `streetaddress` サブフィールドにアクセスします。

```
NK1(2):Address.streetaddress
```

下の仮想プロパティ・パスは、1 つ目の `PR1grp` セグメントの `AUTgrp.CTD` サブセグメントの `ContactAddress` フィールドの `streetaddress` サブフィールドにアクセスします。

```
PR1grp(1).AUTgrp.CTD:ContactAddress.streetaddress
```

繰り返しフィールドの特殊なバリエーション

この節では、繰り返しフィールドを参照するときに適用される仮想プロパティ・パスのバリエーションについて説明します。

繰り返しフィールドを通した繰り返し

BPL または DTL 内で中かっこ `{}` 表記が使用されている場合は、ショートカット `()` が繰り返しフィールドのすべてのインスタンスを通して繰り返されます。例えば、DTL の次の 1 行を見てください。

XML

```
<assign property='target.{PID:3().4}' value='"001"'/>
```

この行は、同じく有効な DTL の次の 3 行と等価です。

XML

```
<foreach key='i' property='target.{PID:3()}'>
   <assign property='target.{PID:3(i).4}' value='"001"'/>
</foreach>
```

同じ `()` 規則は、BPL でも利用できます。

フィールドのカウント

パスが繰り返しフィールドを参照している場合は、`(*)` を使用してフィールド数を返すことができます。

重要　　DTL 内のフィールド数をカウントするには、`(*)` の代わりに `("*")` を使用します。

この構文は、標準メッセージ内のコレクション・プロパティにも使用できます。

一連の繰り返しフィールド内の最後のフィールドへのアクセス

XML 仮想ドキュメントの場合のみ、パスが繰り返しフィールドを参照している際に、`.(-)` を使用して最後のフィールドを返すことができます。詳細は、"XML 要素の内容の取得または設定" を参照してください。

この構文は、標準メッセージ内のコレクション・プロパティにも使用できます。

一連の繰り返しフィールドへの追加

パスが繰り返しフィールドを参照している場合は、**.()** を使用して別のフィールドを追加できます。

この構文は、標準メッセージ内のコレクション・プロパティにも使用できます。

中かっこ {} の構文

中かっこ {} 構文を使用して仮想プロパティにアクセスする方法について説明します。

該当する場合

この構文は、ビジネス・ルール、検索テーブル、検索フィルタ、BPL 要素 (<code> と <sql> 以外)、および DTL 要素 (<code> と <sql> 以外) 内で使用できます。

中かっこの構文で解決する場合は、メッセージ構造を把握している必要があります。現在のコンテキストでメッセージ構造が不明の場合は、可能であれば、代わりに、角かっこ [] 構文を使用します。

詳細

中かっこ {} 構文を使用して仮想プロパティにアクセスするには、以下の構文を使用します。

```
message.{myVirtualPropertyPath}
```

説明 :

- message は、現在のメッセージを参照している変数です。この変数の名前はコンテキストによって異なります。
- myVirtualPropertyPath は、前述した仮想プロパティ・パスです。

前述した構文は、以下のメソッド呼び出しと等価です。

```
message.GetValueAt("myVirtualPropertyPath")
```

中かっこ **{}** 構文の方が単純なため、幅広く使用されています。

一斉コピー

BPL または DTL で中かっこ **{}** 表記を使用している場合、セグメント全体、セグメントのグループ、またはセグメント内の複合フィールド全体をコピーできます。これを実現するには、仮想プロパティ・パスのフィールド部分 (およびコロン・セパレータ) を削除します。したがって、以下の DTL <assign> 文はすべて有効です。

```
<assign property='target.{MSH}' value='source.{MSH}' />
<assign property='target.{DG1()}' value='source.{DG1()}' />
<assign property='target.{DG1(1):DiagnosingClinician}' value='"^Bones^Billy"'/>
```

注釈　ソースとターゲットのタイプが異なる場合は、構造が並列であるように見える場合でも、一斉コピーを使用してサブプロパティを割り当てることはできません。このようなコピーの場合は、構造の各リーフ・ノードを別々に割り当てて、For Each アクションをプロセス反復に追加する必要があります。

上の例の最後の行は、コンポーネントの区切り文字としてキャレット (^) を使用しています。各 EDI 形式の詳細は、以下を参照してください。

- "プロダクション内での EDIFACT ドキュメントのルーティング" の参照節内の "セパレータ"
- "プロダクション内での X12 ドキュメントのルーティング" の参照節内の "セパレータ"

角かっこ [] の構文

角かっこ [] 構文を使用して仮想プロパティにアクセスする方法について説明します。

該当する場合

この構文は、ビジネス・ルール、検索テーブル、および検索フィールド内で使用できます。また、この構文は、EDIFACT および X12 ドキュメントで使用できます。

詳細

角かっこ [] 構文を使用して仮想プロパティにアクセスするには、以下の構文を使用します。

```
[myVirtualPropertyPath]
```

ここで、myVirtualPropertyPath は、フィールド識別子を数値形式にしなければならないことを除いて、前述した仮想プロパティ・パスです。この構文は、メッセージ構造に関係なく、指定されたセグメント内で値を検索します。

ビジネス・ルールに角かっこ構文を使用する場合に、メッセージ内に segment タイプのインスタンスが複数あるときは、この構文は、一致するすべての値を含み、各値が <> 山かっこで囲まれた文字列を返します。例えば、構文が複数の値の a、b、および c を返す場合は、これらが次のように 1 つの文字列で表示されます。

```
<a><b><c>
```

角かっこ構文を検索テーブルと共に使用する場合に、メッセージ内に segment タイプのインスタンスが複数あるときは、この構文は、一致するフィールド/セグメントのリストをコンパイルしてからリストを反復処理し、フィールド/セグメントを 1 つずつ個別にテーブルに挿入します。

`IntersectsList` ユーティリティ関数を使用して、角かっこ構文によって返された文字列に特定の値があるかどうかをテストできます。`IntersectsList` の詳細は、"組み込み関数" を参照してください。

角かっこを使用した場合、InterSystems IRIS は特定のメッセージ構造またはスキーマを認識することなく、数値パスを解決することができます。これは、メッセージ構造を指定する必要のある中かっこ **{}** とは異なります。例えば、DTL データ変換では、中かっこ構文を使用できるよう、<transform> 要素 (sourceDocType および targetDocType) の属性を持つソースおよびターゲットのメッセージ構造を指定します。

角かっこの構文では、プロパティ・パス (segment:field) の field 部分でのみ、繰り返しフィールドのショートカット **()** をサポートしています。

例

以下の検索テーブル・クラスからの抜粋は、メッセージ内の FT1:12.1 フィールドと FT1:6 フィールドのすべての値を照合する方法を示しています。

XML

```
<Items>
  <Item DocType="" PropName="TransactionAmt">[FT1:12.1]</Item>
  <Item DocType="" PropName="TxType">[FT1:6]</Item>
</Items>
```

FindSegmentValues() との比較

この構文は、FindSegmentValues() のデフォルトの動作に似ていますが、全く同じではありません。代わりに、区切り文字が変更され、結果が山かっこで囲まれます。そのため、角かっこの構文は次のメソッド呼び出しに相当します。

```
"<"_msg.FindSegmentValues("segment:field",,"><")_">"
```

丸かっこ () 構文

丸かっこ () 構文を使用して仮想プロパティにアクセスする方法について説明します。

該当する場合

この構文は、ビジネス・ルールと DTL 変換内で使用できます。

詳細

丸かっこ () 構文を使用して仮想プロパティにアクセスするには、以下の構文を使用します。

```
message.(multi-valued-property-path)
```

説明：

- message は、現在のメッセージを参照している変数です。この変数の名前はコンテキストによって異なります。
- multi-valued-property-path は、前述したように、繰り返しフィールドのショートカット () を使用して繰り返しフィールドのすべてのインスタンスを通して繰り返される仮想プロパティ・パスです。

前述した構文は、以下のメソッド呼び出しと等価です。

```
message.GetValueAt("multi-valued-property-path")
```

構文により複数値、a、b、および c が返される場合、これらは以下のように <> 山かっこで囲まれ、1 つの文字列に表されます。

```
<a><b><c>
```

山かっこ <> 構文

山かっこ <> 構文を使用して、XPath 式で仮想プロパティにアクセスする方法について説明します。

該当する場合

この構文はビジネス・ルール内で使用できます。

詳細

山かっこ構文を使用して仮想プロパティにアクセスするには、以下の構文を使用します。

```
message<xpathexpression>
```

説明

- message は、現在のメッセージを参照している変数です。この変数の名前はコンテキストによって異なります。
- xpathexpression は、XPath 式です。

例えば、XPath 式を使用して、特定の文字列が XML ドキュメント内にあるかどうかを評価する条件は、以下のようになります。

```
Contains(Document.Stream.</|name(/*)>, "00UK")
```

前述した構文は、以下と等価です。

```
GetXPathValues(message.stream,"context|expression")
```

GetXPathValues() は、ルール・エンジンにおいて便利なメソッドです。これは、コンテンツが XML ドキュメントであるストリーム・プロパティを含むメッセージで実行されます。このメソッドは、XPath 式をストリーム・プロパティ内の XML ドキュメントに適用し、一致するすべての値を返します。XPath 引数の context| 部分がない場合、InterSystems IRIS は XML ドキュメント全体を検索します。

構文により複数値、a、b、および c が返される場合、これらは以下のように <> 山かっこで囲まれ、1 つの文字列に表されます。

```
<a><b><c>
```

共通設定

ここでは、仮想ドキュメントに関連する以下の参照情報を提供します。

ビジネス・サービスに関する設定

仮想ドキュメントを処理するビジネス・サービスの設定に関する参照情報を提供します。

概要

仮想ドキュメントを処理する複数のビジネス・サービスに以下の設定が含まれています。

グループ	設定
基本設定	[ターゲット構成名]、[Docスキーマカテゴリ]
追加設定	[テーブルクラス検索]、[検証]

[ドキュメント・スキーマ・カテゴリ]

完全な DocType 指定を生成するために受信ドキュメント・タイプ名に適用されるカテゴリ。ドキュメント・タイプ名と組み合わせると、DocType 割り当てを生成できます。この設定では、複数のタイプ名をカンマ区切りリストで指定することもできます。それぞれのタイプとして宣言したドキュメントに適用する DocTypeCategory の値または完全な DocType の値も、このリストでタイプ名の後に = で続けて指定できます。

タイプ名の一部分を指定してその末尾にアスタリスク(*)を付けると、その部分名で始まるすべてのタイプと一致します。

DocType 割り当ては、[検証] または [テーブルクラス検索] のインデックス付けに必要な場合があることに注意してください。

[検索テーブル・クラス]

受信ドキュメント内の仮想プロパティにインデックスを付けるために使用するクラスを指定します。デフォルトは EDI 形式によって異なりますが、独自の検索テーブル・クラスを作成して使用できます。“検索テーブルの定義”の章を参照してください。

いずれの場合も、[Docスキーマ・カテゴリ]で指定されたカテゴリに、検索テーブル・クラス内の DocType 値(存在する場合)が含まれていることを確認します。

[ターゲット構成名]

受信ドキュメントの送信先の構成項目。

[検証]

ビジネス・ホストが受信仮想ドキュメントを受信したときに実行する検証を決定します。

ドキュメントが検証に合格しなかった場合は、ビジネス・サービスまたはビジネス・オペレーションがそのドキュメントを送信しません(詳細は、ビジネス・プロセスの場合と異なります)。

下の表は、[検証] に指定可能な値を示しています。

値	意味
d	ドキュメントの DocType プロパティを調べ、値が指定されているかどうかを確認します。
m	ドキュメントのセグメント構造が適切に構築されており、ドキュメントの DocType プロパティで識別されたスキーマを使用して解析できることを確認します。
dm	d と m の両方が適用されます。
(空白文字列)	ビジネス・ホストが検証を省略し、すべてのドキュメントをルーティングします。これがデフォルト設定です。

詳細は、“基本検証オプションとロジック”を参照してください。

ルーティング・プロセスに関する設定

ほとんどの種類の仮想ドキュメントを転送可能な EnsLib.MsgRouter.RoutingEngine ビジネス・プロセスの設定に関する参照情報を提供します。

概要

EnsLib.MsgRouter.RoutingEngine には以下の設定が含まれています。

グループ	設定
基本設定	[検証]、[ビジネスルール名]
追加設定	[変換エラー時に作動]、[検証エラー時に作動]、[不正メッセージ警告]、[不正なメッセージ・ハンドラ]、[レスポンス From]、[リプライターゲット構成名]、[応答タイムアウト]、[強制同期送信]
開発とデバッグ	ルールのロギング

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

[変換エラー時に作動]

True の場合は、変換によってエラーが返された場合は、ルール評価が停止されて、そのエラーは [リプライコードアクション] 設定によって処理されます。

[検証エラー時に作動]

True の場合は、検証によって返されたエラーは [リプライコードアクション] 設定によって処理されます。

[不正メッセージに関するアラート]

真の場合は、検証に失敗したドキュメントにより、アラートが自動的にトリガされます。

[不正メッセージ・ハンドラ]

ドキュメントが検証で不合格となり、ルーティング・プロセスに **[不正メッセージ・ハンドラ]** が構成されている場合、その不正ドキュメントは、検証に合格したドキュメントの通常の宛先ではなく、このビジネス・オペレーションに送信されます。

前述した "不正メッセージ・ハンドラの定義" を参照してください。

[ビジネス・ルール名]

このルーティング・プロセスのルーティング・ルール・セットの完全名。

[強制同期送信]

真の場合、このルーティング・プロセスからのすべての "送信" アクションに対し、同期呼び出しを行います。偽の場合、これらの呼び出しを非同期に行うことができます。この設定の目的は FIFO の順序を保証することです。これが必要になるのは、このルーティング・プロセスとそのターゲット・ビジネス・オペレーションのすべてで **[プールサイズ]** が 1 に設定されており、このルーティング・プロセスから呼び出されるデータ変換またはビジネス・オペレーション内部から非同期で付帯的なビジネス・オペレーションが呼び出される場合です。

[強制同期送信] が真の場合、このルーティング・プロセスから同期的に呼び出されるターゲットにより別のビジネス・プロセスが呼び出されると、デッドロックが発生する可能性があります。

複数の“送信”ターゲットがある場合、**[強制同期送信]** では、前の呼び出しが完了すると次が呼び出され、これらのターゲットが 1 つずつ連続して呼び出されます。同期呼び出しは、[応答タイムアウト] 設定の影響を受けないことにも注意してください。

[応答ターゲット構成名]

ビジネス・サービスが受け取ったすべての応答ドキュメントを中継する必要があるプロダクション内で構成項目のカンマ区切りリストを指定します。通常、リストには、1 つの項目が含まれますが、それ以上の項目数の場合があります。リストには、ビジネス・プロセスまたはビジネス・オペレーション、あるいはその両方の組み合わせを含めることができます。

この設定は、[レスポンス From] 設定に値が指定されている場合にのみ有効です。

[応答]

プロダクション内の構成項目のカンマ区切りリスト。このリストは、呼び出し側が応答を要求した場合に、呼び出し側に応答を返すターゲットを指定します。

[応答] の文字列が指定されている場合は、呼び出し側に返される応答が、リスト内のターゲットから最初に返される応答です。応答がない場合は、空の“OK”応答ヘッダが返されます。

[応答] の文字列には、以下の特殊文字を使用することもできます。

- * 文字はそれ自体でプロダクション内のターゲットと一致します。したがって、ターゲットの最初の応答が返されます。応答がない場合は、空の“OK”応答ヘッダが返されます。
- ターゲットのリストが + 文字で始まる場合は、すべてのターゲットからの応答が、応答ヘッダ内のドキュメント・ヘッダ ID のリストとしてまとめて返されます。どのターゲットも応答しない場合は、空の OK 応答ヘッダが返されます。
- ターゲットのリストが - 文字で始まる場合は、エラー応答のみが応答ヘッダ内のドキュメント・ヘッダ ID のリストとして返されます。どのターゲットもエラーにより応答しない場合は、空の OK 応答ヘッダが返されます。

この設定値が指定されていない場合は、何も返されません。

[応答タイムアウト]

“タイムアウト・エラー”応答ヘッダが返されるまでの非同期応答の最長待機時間。値が -1 の場合は永久に待機します。0 の値は、すべての応答がタイムアウトすることになるため、有用ではありません。この設定は、**[応答]** フィールドに値がある場合のみ有効です。

[ルールのロギング]

ロギングが有効になっている場合は、ルール内のロギングのレベルを制御します。以下のフラグを指定できます。

- e – エラーのみをログに記録します。他のフラグとは無関係にすべてのエラーがログに記録されるため、値を 'e' に設定するか、値を空白のままにすると、エラーのみがログに記録されます。
- r – 戻り値をログに記録します。これがデフォルト値であり、'd' や 'c' のフラグが指定されている場合はこのフラグが自動的に指定されます。
- d – ルール内のユーザ定義のデバッグ・アクションをログに記録します。デバッグ・アクションの詳細は、″ビジネス・ルールの開発″ の “アクションの追加” を参照してください。このフラグには 'r' フラグの機能も含まれます。
- c – ルール内で評価される条件の詳細をログに記録します。このフラグには 'r' フラグの機能も含まれます。
- a – すべての取得可能な情報をログに記録します。このフラグは 'rcd' と指定するのと同じ意味です。

[検証]

許容値と基本情報については、ビジネス・サービスの [検証] 設定を参照してください。

ドキュメントが検証に合格しなかった場合は、[不正なメッセージハンドラ] 設定で指定されているように、ルーティング・プロセスがそのドキュメントを不正メッセージ・ハンドラに転送します。不正メッセージ・ハンドラが存在しない場合は、ルーティング・プロセスがドキュメントを転送せずに、ログにエラーを書き込みます。[不正メッセージ警告] も参照してください。

ビジネス・オペレーションに関する設定

仮想ドキュメントを処理するビジネス・オペレーションの設定に関する参照情報を提供します。

概要

仮想ドキュメントを処理する複数のビジネス・オペレーションに以下の設定が含まれています。

グループ	設定
追加設定	[テーブルクラス検索]、[検証]

[検索テーブル・クラス]

ビジネス・サービスの [テーブルクラス検索] 設定を参照してください。

[検証]

ビジネス・サービスの [検証] 設定を参照してください。