
# ビジネス・ルールの開発

Version 2024.1
2024-06-06

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

ビジネス・ルールの開発
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 ビジネス・ルールについて ..... 1
1.1 クラスとしてのルール ..... 2

2 はじめに ..... 5
2.1 旧 Zen ルール・エディタの使用 ..... 5
2.2 ルール定義について ..... 6
2.2.1 エクスポートとインポートのルール ..... 6
2.3 ルール・セットについて ..... 7

3 ルールの操作 ..... 9
3.1 if 節と else 節について ..... 9
3.2 アクションについて ..... 10
3.2.1 foreach アクションの使用法 ..... 10
3.3 式の定義 ..... 12
3.3.1 If 節の条件プロパティの編集 ..... 13
3.3.2 式の演算子 ..... 14
3.3.3 式の関数 ..... 15
3.3.4 式の例 ..... 16
3.4 制約の定義 ..... 17
3.5 ルールの無効化 ..... 17
3.6 データ変換へのデータの引き渡し ..... 18
3.7 ビジネス・ルール通知の追加 ..... 18

4 ルーティング・ルールのデバッグ ..... 19
4.1 ルーティング・ルールのテスト ..... 19
4.1.1 メッセージの未加工のテキストによるテスト ..... 19
4.1.2 テスト結果 ..... 20
4.1.3 セキュリティ要件 ..... 20
4.2 ルーティング・ルールのデバッグ方針 ..... 20

付録A: プロダクションで使用するユーティリティ関数 ..... 25
A.1 組み込み関数 ..... 25
A.2 関数の呼び出し構文 ..... 30

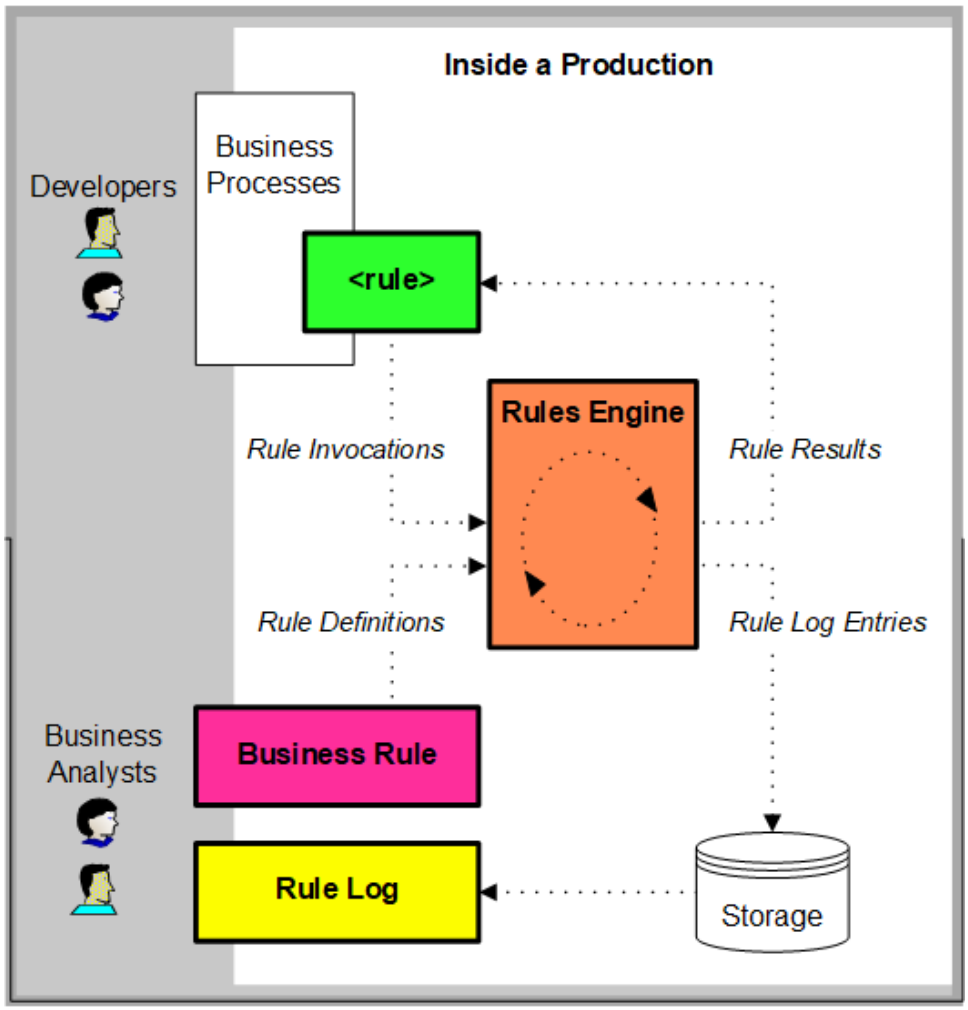
# 図一覧

図 4-1: ルーティング・ルールに関する問題の解決 (図 A) ..... 21
図 4-2: ルーティング・ルールに関する問題の解決 (図 B) ..... 22
図 4-3: ルーティング・ルールに関する問題の解決 (図 C) ..... 23
図 4-4: ルーティング・ルールに関する問題の解決 (図 D) ..... 24

# 1

# ビジネス・ルールについて

ビジネス・ルールを使用すれば、非技術系ユーザでも、特定の判断ポイントにおけるビジネス・プロセスの動作を変更できます。ルールのロジックは、管理ポータルでルール・エディタを使用してすぐに変更できます。ルールの変更には、プログラミング・スキルやダイアグラミング・スキルは必要ありません。また、変更内容を有効化するために、プロダクション・コードを変更したり、コンパイルする必要はありません。以下の図は、ビジネス・ルールが機能するしくみについて示しています。



ある国際企業が、ローン申し込み処理するプロダクションを実行しているとします。判断のプロセスは全世界で共通していますが、企業内の各銀行にはそれぞれ独自の受け入れ基準があり、それは国によって異なる可能性があります。以下で説明するように、ビジネス・ルールは、この地域レベルでの責任範囲をサポートします。

1. ビジネス・プロセスの開発者は、ビジネス・プロセスの代わりに判断を行うビジネス・ルールに名前を付けて、この判断ポイントを識別します。開発者が、ビジネス・プロセス言語 (BPL) コード内のそのビジネス・ルールのプレースホルダを活用するには、ビジネス・プロセス言語 (BPL) 要素 〈rule〉 を呼び出します。〈rule〉 要素は、ビジネス・ルール名、および判断結果とその結果の理由 (オプション) を保持するパラメータを指定します。ここでは、このルールを `LoanDecision` と呼びます。

2. BPL ビジネス・プロセス内に 〈rule〉 要素が出現するたびに、プロダクション内に対応するルール定義が存在する必要があります。企業のユーザ (通常はビジネス・アナリスト) は、ルール・エディタというブラウザベースのオンライン・フォームを使用してルールを定義できます。このフォームでは、ユーザにメッセージが表示され、`LoanDecision` と呼ばれるビジネス・ルールの定義で必要とされる簡単な情報を入力するように促します。この情報は、InterSystems IRIS® の構成データベースに保存されます。

   ルール・エディタに習熟している企業ユーザは、管理ポータルにあるこのエディタへのアクセス権を与えられていれば、ルール定義を変更できます。変更は単なるデータベースの更新であるため、実行中のプロダクションに即座に適用できます。したがって、各地域のビジネス・アナリストは、ルール・エディタを実行してルールのコピーを変更することによって、それぞれの地域に適したその地域固有の基準を別途作成できます。

3. 実行時には、BPL 〈rule〉 文に到達すると、ビジネス・プロセスは、`LoanDecision` という名前のルールを呼び出します。ルールは、構成データベースから、地域ごとに異なる判断基準を取得します。これらの基準に基づいて、ルールは、ビジネス・プロセスに回答を返します。ビジネス・プロセスは、この回答に基づく実行パスをリダイレクトします。

4. 進行中の保守が目的である場合については、ルールを変更する必要があっても、ビジネス・プロセス開発者が参加する必要はありません。どのルール定義もビジネス・プロセス・コードとは完全に分離されています。ルール定義は構成データベース内にクラスとして格納され、実行時に評価されます。また、ルール定義は、InterSystems IRIS インストール間でエクスポートおよびインポートできます。

   同様に、ビジネス・アナリストなどの企業ユーザは、ビジネス・プロセスの BPL またはクラス・コードの修正に必要なプログラミングの専門知識がなくても、判断ポイントでビジネス・プロセスのオペレーションを変更できます。

ルール定義は 1 つ以上のルール・セットの集まりです。ルール・セットは 1 つ以上のルールの集まりです。各ルール・セットには、有効期間開始日時と有効期間終了日時があります。ビジネス・プロセスがルール定義を呼び出すと、常に 1 つのルール・セットのみが実行されます。

ビジネス・プロセス、データ変換、およびビジネス・ルールを使用してルール・ワークフローを指定するための、複数のオプションが用意されています。これらのオプションについては、"ビジネス・ロジック・ツールの比較" を参照してください。

以降のトピックでは、ビジネス・ルールの定義方法について取り上げ、ルール・エディタを使用したルール・セットの作成および使用方法などについて説明すると共に、BPL とビジネス・プロセス・ルーティング・エンジンのそれぞれを使用したルールの呼び出し方法について説明します。

# 1.1 クラスとしてのルール

企業のビジネス・アナリストは、**[相互運用性]**→**[ビルド]**→**[ビジネス・ルール]** (または **[ルール・エディタ]**) ページを使用することによって、ビジネス・プロセスで行う論理的な決定を体系的に構築できます。この構築には、プログラミングのスキルは不要です。

さらに、ビジネス・プロセス開発者は、スタジオを使用してビジネス・ルール定義をクラスとして使用することができます。

### Class Definition

```
/// Business rule responsible for mapping an input location
///
Class Demo.ComplexMap.Rule.SemesterBatchRouting Extends Ens.Rule.Definition
{

Parameter RuleAssistClass = "EnsLib.MsgRouter.RuleAssist";

XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ]
{
<ruleDefinition alias="" context="EnsLib.MsgRouter.RoutingEngine"
production="Demo.ComplexMap.SemesterProduction">
<ruleSet name="" effectiveBegin="" effectiveEnd="">
<rule name="" disabled="false">
<constraint name="source" value="Semester_Data_FileService"></constraint>
<constraint name="msgClass" value="Demo.ComplexMap.Semester.Batch"></constraint>
<when condition="1">
<send transform="" target="Semester_Data_FileOperation"></send>
<send transform="Demo.ComplexMap.Transform.SemesterBatchToSemesterSummaryBatch"
target="Semester_Summary_FileOperation"></send>
<send transform="Demo.ComplexMap.Transform.SemesterBatchToFixedClassBatch"
target="Semester_FixedClassBatch_FileOperation"></send>
<send transform="Demo.ComplexMap.Transform.SemesterBatchToFixedStudentBatch"
target="Semester_FixedStudentBatch_FileOperation"></send>
<send transform="" target="Semester_FixedStudent_BatchCreator"></send>
<return></return>
</when>
</rule>
</ruleSet>
</ruleDefinition>
}

}
```

スタジオでビジネス・ルールをクラスとして開き、ドキュメントを編集して、変更内容を保存できます。スタジオで保存した変更内容は、すぐに **[ルール・エディタ]** ページで使用できます。ただし、これらの変更内容を表示するには、ページの更新が必要になることがあります。

### パッケージ・マッピング・ルール・クラス

ルールがクラスである場合は、他のネームスペースにマッピングできます。マッピングする場合は、マッピングしたすべてのルール・クラスを使用先の各ネームスペースでリコンパイルすることによって、各ネームスペースでローカル・メタデータを確実に使用できるようにする必要があります。

詳細は、"パッケージ・マッピング" を参照してください。

# 2
# はじめに

InterSystems IRIS 内の各ビジネス・ルールは、ルール・セットの一部です。さらに、各ルール・セットは、より大きなルール定義の一部です。ルールとルール定義という用語は、場合によっては同じ意味で使用されますが、それぞれのルールは、最終的にはルール定義の下でグループ化されます。

ルール定義、ルール・セット、およびルールは、ルール・エディタを使用して編集します。これは、管理ポータルで **[相互運用性]→[ビルド]→[ビジネス・ルール]** に移動することによりアクセスできます。

ルール・エディタ内の既存のルールは、**[相互運用性]→[リスト]→[ビジネス・ルール]** に移動して開くことができます。

重要　ルール・エディタには、`/ui/interop/rule-editor` と `/api/interop-editors` の 2 つの Web アプリケーションが含まれています。`/api/interop-editors/` Web アプリケーションは、管理ポータルと同じ認証方法を使用するように構成できます。この Web アプリケーションに代わりの認証方法が構成されていない場合、ユーザはルール・エディタを開いたときに InterSystems IRIS ユーザとして再認証する必要があります。`/ui/interop/rule-editor` Web アプリケーションは、認証なしのままにする必要があります。

## 2.1 旧 Zen ルール・エディタの使用

Zen ベースのルール・エディタは、新しいルール・エディタに置き換えられました。新しいルール・エディタは同じ機能を維持していますが、ユーザ・インタフェースが更新されています。

古いルール・エディタは新しいルール・エディタから引き続きアクセスできます。これにアクセスするには、新しいルール・エディタを開き、右上のメニュー（現在のユーザのユーザ名とプロファイルのアイコンで識別）をクリックして、**[Zen ルール・エディタで開く]** をクリックします。

古いルール・エディタを既定のエディタとして使用するように設定することもできます。以下はその方法です。

1. 管理ポータルで、**[システム管理]** → **[セキュリティ]** → **[アプリケーション]** → **[ウェブ・アプリケーション]** に移動します。
2. [/ui/interop/rule-editor] Web アプリケーションを選択します。
3. **[アプリケーション有効]** チェックボックスのチェックを外します。
4. **[保存]** をクリックします。

この Web アプリケーションを無効にすると、管理ポータルのリンクで古いルール・エディタが開くようになります。[/ui/interop/rule-editor] Web アプリケーションを無効にした場合、新しいルール・エディタにアクセスするには、再度このアプリケーションを有効にする必要があります。

# 2.2 ルール定義について

個々のビジネス・ルールは、ルール定義の下でグループ化されます。これは、ルール・エディタを使用して構築および編集します。ルール定義には、以下の設定が含まれます。

**パッケージ**

ルール定義クラスのパッケージ。

**名前**

ルール定義クラスの名前。

**説明**

ルール定義とその目的を示すユーザ指定の説明。

**ルール・タイプ**

ルール定義のタイプ。そのルール定義に属するルールを定義する際に、有効なアクションを決定します。

**コンテキスト・クラス**

ルールの編集時に変更できるオブジェクト・プロパティを決定するクラス。汎用ビジネス・ルールの場合、コンテキスト・クラスは BPL プロセスに関連付けられているビジネス・プロセス・クラスから生成され、末尾に .Context が付きます。BPL プロセスに関連付けられていないルーティング・ルールの場合、コンテキスト・クラスは通常、ルーティング・エンジンによって使用されるビジネス・プロセス・クラスです。

ルール定義の作成時に、**[フィルタ]** オプションを使用すると、**[コンテキスト・クラス]** ドロップダウン・リストに表示されるクラスのリストを縮小できます。

**プロダクション名**

(オプション) ルーティング・ルールでは、ルールを編集する際にルール・エディタが事前定義したオプションを提供できるように、ルールが使用されるプロダクションの名前を指定します。例えば、プロダクションを指定してから制約を変更した場合、プロダクションの構成項目は、制約の **[ソース]** フィールドのオプションとして表示されます。

プロダクションの構成時にルールを指定した場合を除き、プロダクションではルーティング・ルールは自動的には使用されません。

## 2.2.1 エクスポートとインポートのルール

ルール・セットおよびルールを含むルール定義は、**[相互運用性]**→**[リスト]**→**[ビジネス・ルール]** に移動し、**[エクスポート]** を選択することによりエクスポートできます。以前エクスポートしたルール定義をインポートするには **[インポート]** オプションを使用します。

あるいは、ルール・クラスを、管理ポータルの **[システムエクスプローラ]**→**[グローバル]** ページ、またはスタジオの **[ツール]** メニューからエクスポートまたはインポートすることもできます。

# 2.3 ルール・セットについて

新しいルール定義を作成すると、新しいルール・セットが自動的に作成されます。ルール・セット名および有効な日付範囲を定義する場合、または新しいルール・セットを作成する場合は、ルール・セット名の横にある  アイコンを選択します。

一般に、ルール・セットには以下に示す 2 つのタイプがあります。

- 汎用ビジネス・ルール・セット – いずれか 1 つが真であることが判明するまで順次評価されるルールのリスト。真と判明したルールは、そのルールを呼び出したビジネス・プロセスの次のアクションを決定します。どのルールも真でない場合は、ルール・セットからデフォルト値が返されます。このタイプのルール・セットを呼び出すには、BPL <rule> 要素を使用します。
- ルーティング・ルール・セット – メッセージ・ルーティング・プロダクションで使用するルール・セット。ルーティング・ルール・セットでは、受信メッセージ (制約として指定した) のタイプと内容に基づいて、各メッセージの正しい宛先と、送信前にメッセージの内容を変換する方法が決定されます。このタイプのルール・セットを呼び出すには、ルーティング・エンジン・ビジネス・プロセスを使用します。

すべてのルール・セットには、2 つのプロパティがあります。

- **[ルール・セット名]** – ルール・セットの識別子。
- **[有効期間]** – ルール・セットが有効な期間、つまりそのルールが実行される期間を定義します。

一般に、ルール定義には、常に有効なルール・セットが 1 つだけ含まれます。ただし、ルール定義には、異なる時間に有効なルール・セットであれば、複数のルール・セットを含めることができます。ビジネス・プロセスからのルールの呼び出し時には、常に 1 つのルール・セットのみが実行されます。

# 3
# ルールの操作

ルール・セットには、ビジネス・プロセスの特定の機能を満たすために定義された 1 つ以上のルールが含まれます。ルール・エディタ ([**相互運用性**]→[**ビルド**]→[**ビジネス・ルール**]) で新しいルール定義を作成すると、ルール・セットにルールを追加する準備が整います。

各ルールに名前を付けることはできますが、必須ではありません。既定では、InterSystems IRIS® では、**rule#**n の形式のルール名が使用され、順番に番号が割り当てられます。ルールにユーザ定義の名前を付与すると、その名前がクラス定義に表示されると共に、ルール・ログで内部ルール名の横に括弧で囲んで示されます。ルール・セット内のルールの順序を変えると、n の値が変わります。

## 3.1 if 節と else 節について

ルールには 1 つ以上の **if** 節と 1 つの **else** 節を含めることができます。各節には、**assign** や **return** などのアクションを含めることができます。

**if** 節のロジックを実行できるのは、節に関連付けられている **condition** プロパティが真の場合のみです。**else** 節のロジックを実行できるのは、前の if 節に関連付けられているいずれの **condition** プロパティも真でない場合のみです。ルールに複数の if 節が含まれる場合、節に関連付けられている **condition** プロパティが真である、最初の if 節のロジックのみが実行されます。条件の詳細は、"If 節の条件プロパティの編集" を参照してください。

ルールを開発する際は、以下の点に留意してください。

- ルール・セットの実行で return アクションが検出されると、ルール・セットの実行は終了し、そのルール定義クラスを呼び出したビジネス・プロセスに戻ります。
- return を省略することによって、ルール・セット内の複数のルールの実行を制御できます。つまり、すべてのルールをチェックする必要がある場合は、どのルール節でも return アクションを使用しないようにします。そして、どのルール節も真として評価されなかった場合のために、ルール・セットの最後に return アクションで値を提供します。
- if 節ごとに条件プロパティを 1 つ指定します。汎用ビジネス・ルール・セットの一般的な設計は、一連の if 条件で構成されるルールを 1 つ組み込み、どの条件が真であるかに応じて値を返す、というものです。どの条件も真でない場合にデフォルト値を返すには、else 節を return と共に使用します。
- ルーティング・ルール・セットの一般的な設計は、複数のルールを使用してルールごとに異なる制約を定義し、制約に一致したメッセージのルーティング方法とルーティング先を示す if 節を各ルールに 1 つ組み込むというものです。
- "仮想プロパティ・パスの基本設定" に説明されている構文を使用して、仮想ドキュメントのプロパティ・パスにアクセスできます。

# 3.2 アクションについて

ルールの各 if 節または else 節にはアクションを含めることができますが、必須ではありません。節のアクションが実行されるのは、節に関連付けられている条件が真の場合かつその場合のみです。以下のアクションがサポートされています。

| ルール・セットのタイプ | アクション | 説明 |
|---|---|---|
| すべて | assign | ビジネス・プロセスの実行コンテキストのプロパティに値を割り当てます。 |
| すべて | return | ルールをそれ以上実行せずにビジネス・プロセスに戻ります。また、汎用ルールの場合、示された値を結果ロケーションに返します。 |
| すべて | trace | このルールの特定の部分が実行されたときに、入力された情報がイベント・ログに追加されます。詳細は、"<trace>" を参照してください。 |
| すべて | debug | ルールの当該部分が実行されたときに、式のテキストと値がルール・ログに追加されます。debug アクションが実行されるのは、ルータ・ビジネス・プロセスの RuleLogging プロパティで d フラグが指定されている場合のみです。RuleLogging プロパティの詳細は、"ルールのロギング" を参照してください。 |
| セグメント化された仮想ドキュメントのルーティング・ルールまたは HL7 メッセージのルーティング・ルール | foreach | 繰り返しセグメントをループします。繰り返しセグメントとして指定されている場合、繰り返しループ内にある場合、またはその両方の場合、セグメントを繰り返すことができます。詳細は、"foreach アクションの使用法" を参照してください。 |
| ルーティング・ルール | send | ルーティング・エンジン・ビジネス・プロセスによって評価されると、このアクションは必要に応じて変換処理を行った後に、特定のターゲットにメッセージを送信します。データをデータ変換に渡す機能については、"データ変換へのデータの引き渡し" を参照してください。 |
| ルーティング・ルール | delete | ルーティング・エンジン・ビジネス・プロセスによって評価されると、このアクションは現在のメッセージを削除します。 |
| ルーティング・ルール | delegate | ルーティング・エンジン・ビジネス・プロセスによって評価されると、このアクションはメッセージを別のルールに委任します。 |

BPL <rule> 内では、send、delete、および delegate アクションは使用できません。これらを含めると、そのアクションは実行されず、指定したアクションを含む文字列値が返されます。

論理的に適切で、意図したとおりに実行されるようにルール・セットを構築する必要があります。例えば、既定の返り値を設定することは、ルール・セット内のどのルールも実行されない場合には意味がありますが、1 つのルールが常に実行されるようなルール・セットを作成した場合には意味がありません。一般に、ほとんどのアクションは、ルールの if 節にあります。

## 3.2.1 foreach アクションの使用法

foreach アクションでは、繰り返しセグメントをループし、そのセグメント内の任意のフィールドを参照できます。

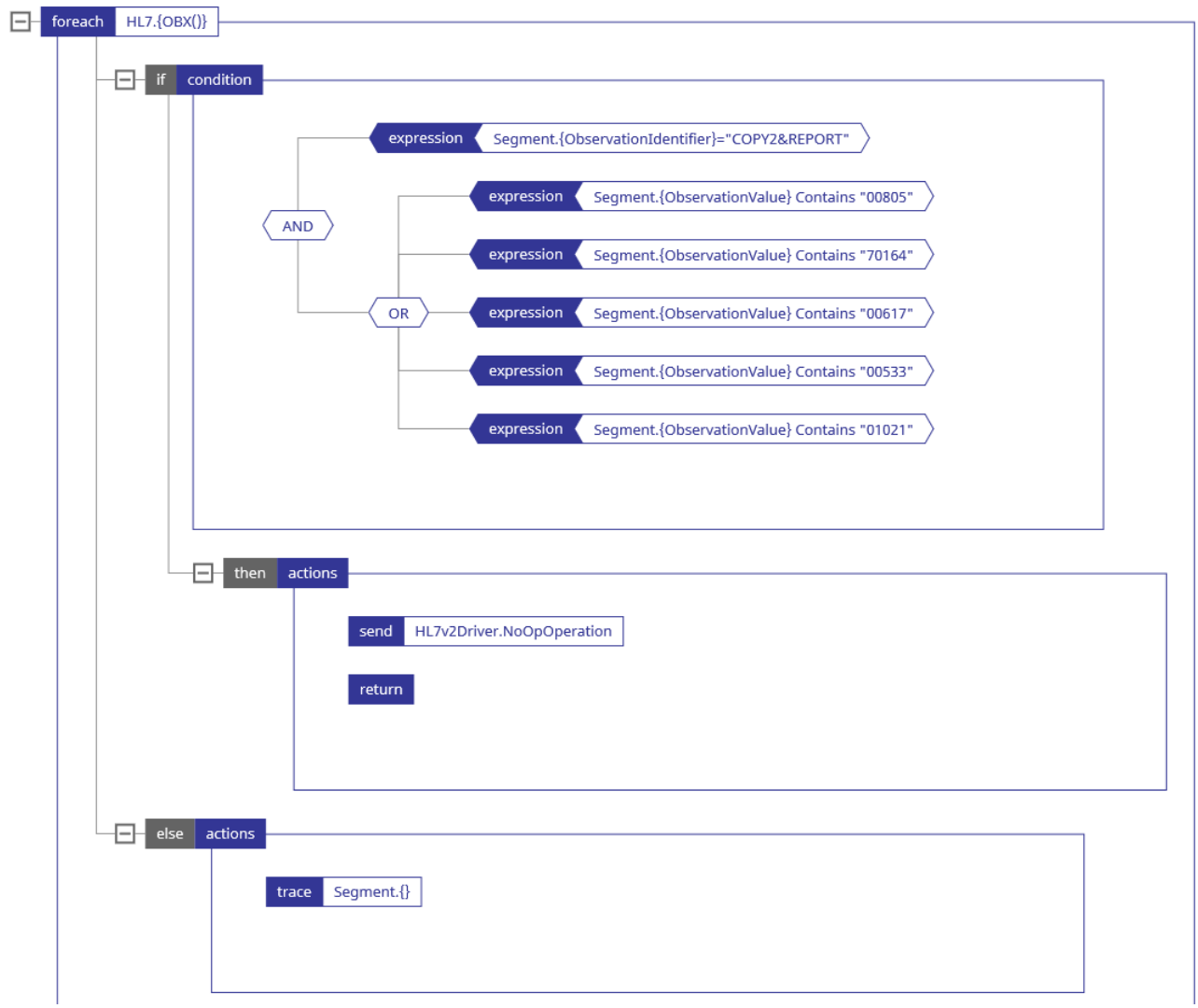
繰り返しセグメントは、"仮想プロパティ・パスの基本設定" で説明されている構文を使用して、foreach アクションの propertypath プロパティで指定します。例えば、HL7 ドキュメントの繰り返しの OBXgrp 内の OBX セグメントにアクセスするには、HL7.{OBXgrp().OBX} を指定します。ここで、空のかっこは繰り返しグループを示します。foreach アクションには 1 つ以上の if 節と 1 つの else 節を含めることができます。節内では、節の条件が真の場合に実行するアクションを指定します。

例えば、foreach アクションを使用して、繰り返しセグメントのフィールドに、いつ特定の値が含まれるかを特定し、値が存在する場合にメッセージをルーティングする send アクションを指定できます。特定のフィールドを参照するには、Segment.{<field-name>} を使用できます (例 : Segment.{ObservationIdentifier})。

foreach アクションの if 節と else 節には、1 つ以上の rule ノードを含めることができます。ただし、foreach アクションを入れ子にすることはできません。

ルールが foreach ループ内で return アクションを実行する場合、そのループまたはルールだけではなく、ルール・セット全体を終了させます。

以下の例は、ビジネス・ルール内での foreach の使用を示しています。このアクションは、HL7 ドキュメントの繰り返しの OBX セグメントを反復して、ObservationIdentifier フィールドに、いつ特定の文字列値が含まれるかを特定します。値が見つかった場合、ルールはドキュメントをファイル操作に送信します。値が見つからない場合、ルールは trace アクションを使用して、イベント・ログにエントリを記録します。



# 3.3 式の定義

式を使用して、以下の 4 つのプロパティの値を変更できます。

- if 節の condition プロパティ ー if 節でロジックを実行する条件を指定します。詳細は、"If 節の条件プロパティの編集" を参照してください。
- assign アクションの value プロパティ ー 割り当てる値を指定します。
- 汎用ビジネス・ルールの return アクションの value プロパティ ー ルール・セットを実行したプロセスに返す値を指定します。
- trace アクションの value プロパティ ー トレース・メッセージに含めるテキストを指定します。リテラル・テキスト文字列または評価対象となる式を指定できます。式は、対応する <process> 要素の language 属性で指定されているスクリプト言語を使用する必要があります。

各プロパティを、以下のいずれかのサポートされている値に設定できます。

- 数値 (整数または小数)。1.1 や 23000 など。
- 二重引用符で囲まれた文字列値。例 :

  ```
  "NY"
  ```

  重要　　二重引用符は必須です。

- コンテキスト・プロパティの値。BPL ビジネス・プロセスには、context と呼ばれる汎用の永続変数を含めることができることを思い出してください。context 変数は、BPL の <context> 要素および <property> 要素を使用して定義します。context オブジェクトのプロパティには、ビジネス・プロセス内の任意の場所からアクセスできます。したがって、<rule> 要素を使用してビジネス・プロセスからルールを呼び出した場合、そのルール内からコンテキスト・プロパティにアクセスできます。

  リストや配列などのコレクションが含まれるコンテキスト・プロパティの場合、InterSystems IRIS では、ビジネス・ルール内からさまざまな取得メソッドをサポートします。それらのメソッドとしては、Count()、Find()、GetAt()、GetNext()、GetPrevious()、IsDefined()、Next()、Previous() などがあります。詳細は、"コレクションを使用した作業" を参照してください。

  注釈　　プロパティ名では大文字と小文字が区別され、引用符で囲むことはできません (例 : PlaceOfBirth)。

- サポートされている演算子、リテラル値、汎用のプロパティ、および永続変数 context を使用する式。以下に例を示します。

  ```
  ((2+2)*5)/154.3
  "hello" & "world"
  Age * 4
  (((x=1) || (x=3)) && (y=2))
  ```

- Min()、Max()、Round(n,m)、SubString() などの組み込み関数。関数名にはかっこが含まれている必要があります。また、Round の数値 n や m などの入力パラメータも含める必要があります。関数に入力値がない場合は、中身が空の左右のかっこを含める必要があります。
- メッセージ・オブジェクトを表す Document 変数。
- foreach アクション内のセグメント。詳細は、"アクションについて" を参照してください。

詳細は、"式の例" を参照してください。

## 3.3.1 If 節の条件プロパティの編集

ルール定義では、1 つの条件は 2 つの値とこれらの値間の比較演算子で構成されます。以下に例を示します。

```
Amount  <=  5000
```

**条件**が  でない場合は、 となります。**条件**プロパティにはその他の可能な値はありません。真または偽のみの結果を、ブーリアン値の結果と呼びます。InterSystems IRIS では、ブーリアン値の結果を整数値として格納します。1 は真で、0 は偽です。ほとんどの場合、この内部表現を使用する必要はありません。ただし、ルーティング・ルールの場合、ルールの**制約**が真の場合はいつでも、**条件**プロパティに対応する If 節を実行できます。この場合、**条件**プロパティを 1 に設定できます。

**条件**プロパティには、複数の条件を含めることができます。InterSystems IRIS は、対応するルールを実行するかどうか決定する前に、プロパティのすべての条件を評価および比較します。条件どうしのロジックは、AND 演算子または OR 演算子によって決定されます。例えば、以下の値の**条件**プロパティを考えてみます。

```
IF  Amount  <=  5000
AND  CreditRating > 5
OR  CurrentCustomer = 1
```

同じ値が、**ルール・エディタ**では次のように表示されます。


値には、Amount <= 5000、CreditRating > 5、CurrentCustomer = 1 という 3 つの条件があります。各条件は真か偽のいずれかになります。InterSystems IRIS は、これらの条件を個別に評価してから、AND および OR 演算子によって定義された、これらの条件の間の関係を評価します。

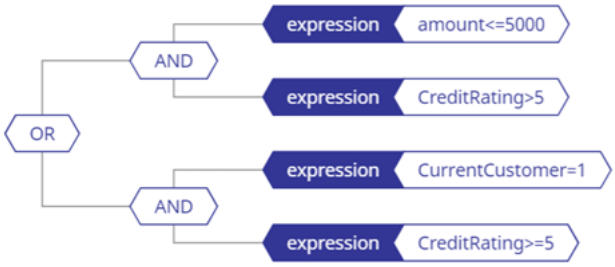
AND および OR 演算子は、真の値および偽の値に対してのみ機能します。すなわち、演算子は 2 つのブーリアン値の間に配置し、次のように 1 つのブーリアン値の結果を返す必要があります。

| 演算子 | 結果が真となる状況 |
|---|---|
| AND | 両方の値が真の場合。 |
| OR | 少なくとも 1 つの値が真であるか、両方の値が真の場合。1 つの値が偽でもう一方の値が真の場合でも、全体としての結果は真になります。 |

**条件**プロパティに複数の AND または OR 演算子が含まれる場合、AND 演算子が OR 演算子よりも優先されます。また、すべての AND 演算子が先に実行され、次に OR 演算子が実行されます。例えば、以下の条件セットを考えてみます。

```
IF  Amount  <=  5000
AND  CreditRating > 5
OR  CurrentCustomer = 1
AND  CreditRating >= 5
```

同じ条件セットが、**ルール・エディタ**では次のように表示されます。




InterSystems IRIS は、条件を次のように評価します。

```
IF  (Amount  <=  5000  AND  CreditRating > 5)
OR  (CurrentCustomer = 1  AND  CreditRating >= 5)
```

以下のいずれかまたは両方の文が真の場合、全体としての条件セットは真となります。

- 誰かが 5,000 未満の金額を要求し、さらにその人の信用等級が平均よりも優れている。
- 現在の銀行顧客の誰かが任意の金額を要求し、さらにその人の信用等級が平均以上である。

両方の文が偽の場合、全体としての条件セットは偽となります。

別の方法で説明すると、InterSystems IRIS は次の手順を実行して条件セットを評価します。

1. 以下の AND 式の結果が真か偽かを判別します。

   ```
   IF  Amount  <=  5000
   AND  CreditRating > 5
   ```

   この結果を“SafeBet”と呼ぶとします。

2. 以下の AND 式の結果が真か偽かを判別します。

   ```
   IF  CurrentCustomer = 1
   AND  CreditRating >= 5
   ```

   この結果を“KnownEntity”と呼ぶとします。

3. 以下の OR 式の結果が真か偽かを判別します。

   ```
   IF  SafeBet is true
   OR  KnownEntity is true
   ```

   SafeBet が真で KnownEntity が偽の場合、条件セットは真となります。同様に、SafeBet が偽で KnownEntity が真の場合でも、条件セットは真となります。最後に、SafeBet と KnownEntity の両方が真の場合、条件セットは真となります。

## 3.3.2 式の演算子

式を定義する際は、以下の算術演算子のいずれかを選択できます。

| 演算子 | 意味 |
|---|---|
| + | 加算 (2 進および単項) |
| - | 減算 (2 進および単項) |
| * | 乗算 |
| / | 除算 |

さらに、以下の論理演算子がサポートされています。論理演算子は、1 (真) または 0 (偽) の整数値を返します。

| 演算子 | 意味 | 式が真となる状況 |
|---|---|---|
| AND (&&) | および | 両方の値が真の場合。 |
| OR (\|\|) | または | 少なくとも 1 つの値が真の場合。両方の値が真であるか、1 つの値のみが真である必要があります。 |
| ! | 偽 (単項) | 値が偽の場合。 |
| = | 等しい | 2 つの値が等しい場合。 |
| != | 等しくない | 2 つの値が等しくない場合。 |
| > | より大 | 演算子の左側の値が演算子の右側の値よりも大きい場合。 |
| < | より小 | 左側の値が演算子の右側の値よりも小さい場合。 |
| >= | 以上 | 左側の値が右側の値よりも大きいか、または 2 つの値が等しい場合。 |
| <= | 以下 | 左側の値が右側の値よりも小さいか、または 2 つの値が等しい場合。 |
| [ | Contains | 文字列は右側のサブ文字列を含みます。Contains のパターン・マッチングは、完全一致です。左側の値が “Hollywood, California” で、右側の値が “od, Ca” の場合は一致と見なされますが、値が “Wood” の場合は一致と見なされません。 |

最後に、以下の文字列演算子を使用できます。

| 演算子 | 意味 |
|---|---|
| & | 文字列の連結演算子。 |
| _ | リテラル文字列、式、または変数を結合するバイナリの連結。 |

複数の演算子が式に含まれているとき、演算子は先頭から後方に向かって以下のような優先順位で評価されます。

1. 次の論理演算子すべて：! = != < > <= >= [
2. 乗算と除算：* /
3. 加算と減算：+ -
4. 文字列の連結：& _
5. 論理 AND：&&
6. 論理 OR：||

## 3.3.3 式の関数

ルール定義内では、式には InterSystems IRIS ユーティリティ関数の 1 つへの呼び出しを含めることができます。このような関数には、他のプログラミング言語に存在するものと類似する数学的な処理関数または文字列処理関数などがあります。式を定義するには、ドロップダウン・リストから関数を選択するだけです。

使用可能なユーティリティ関数のリストとそれらをビジネス・ルールまたは DTL データ変換で使用するための適切な構文については、"プロダクションで使用するユーティリティ関数" を参照してください。

## 3.3.4 式の例

ルール定義内では、式は、値とプロパティを組み合わせて値を返すための公式です。以下の表に、式とその算出値の例を示します。

| 式 | 算出値 |
|---|---|
| `((2+2)*5)/154.3` | `0.129617628` |
| `"hello" & "world"` | `"helloworld"` |
| `Age * 4` | Age が context プロパティ (BPL で <context> および <property> 要素を使用して定義できる汎用永続変数 context のプロパティ) で、その数値が 30 である場合、この式の値は 120 です。 |
| `1+2.5*2` | 6 |
| `2*5` | 10 |
| `Min(Age,80,Limit)` | この式では、組み込みの Min() 関数が使用されます。Age が値 30 を持つ context プロパティであり、同じくプロパティである Limit の値が 65 であるとき、この式の値は 30 になります。 |
| `Round(1/3,2)` | この式では、組み込みの Round() 関数が使用されます。結果は 0.33 です。 |
| `x<65&&A="F"||x>80` | この式では、演算子の優先順位規則が使用されます (詳細は、"式の演算子" で説明します)。A が F の文字列値を持つ context プロパティであり、同じくプロパティである x が 38 の整数値を持っている場合、この式は 1 の整数値を持ちます。InterSystems IRIS では、整数値 1 は真、整数値 0 は偽を意味します。 |
| `Min(10,Max(X,Y))` | この式では、Min() および Max() 関数が使用されます。X が数値 9.125 を持つ context プロパティであり、(同じくプロパティである) Y の数値が 6.875 であるとき、この式の値は 9.125 になります。 |
| `(((x=1) || (x=3)) && (y=2))` | この式では、複雑な論理リレーションシップ内での優先順位を明確に示すため、かっこが使用されます。 |

その値として式を取るプロパティを選択すると、ルール・セット・ダイアグラムの上部に空白のテキスト・フィールドが表示されます。テキスト・フィールドではどんな文字列でも指定できるため、プロパティに適切な構文が使用されていることを確認する必要があります。式を生成する際には、以下のルールを考慮します。

- 式には、前述のように、数値、文字列、context プロパティ、その他の式、関数、またはこれらの有効な組み合わせを使用して、任意の値を記述できます。
- 式の中の空白は無視されます。
- 式には、サポートされる任意の演算子を使用できます。
- 既定の演算子優先順位を上書きするか、式を読みやすくする場合は、かっこを使用して式の部分をグループにまとめ、優先順位を示すことができます。例えば、結果が 6 になる以下の式を考えてみます。

  ```
  1+2.5*2
  ```

  式を次のように変更すると、結果は 7 になります。

  ```
  (1+2.5)*2
  ```

- ビジネス・ルールでは、`(((x=1) || (x=3)) && (y=2))` のように、かっこを使用して複雑な論理式をグループ化できます。

# 3.4 制約の定義

ルール・セットにルーティング・ルールが含まれる場合、ルール・セットを通過するメッセージが、そのルールに対して定義されている制約と一致すると、そのルール・ロジックが実行されるように、制約を定義することができます。フィールドを空白のままにすると、すべての値と一致することになります。ルールの制約を設定するには、そのルールをダブルクリックし、以下の設定を定義します。

**ソース**

以下の項目のいずれかの構成名。

- ビジネス・サービス (ルーティング・インタフェース用)
- メッセージ・ルーティング・プロセス (別のルールがこのルーティング・ルール・セットに連鎖している場合)

**メッセージ・クラス**

このルールによってルーティングするプロダクション・メッセージ・オブジェクトを指定します。このフィールドの値は、以下のルーティング・ルール・タイプに応じて決まります。

- 汎用メッセージ・ルーティング・ルールの場合、**[メッセージ・クラス]** フィールドの横にある省略記号 (...) をクリックして **[ファインダダイアログ]** を呼び出し、該当するメッセージ・クラスを選択できます。メッセージ・クラスのカテゴリを選択して、選択肢を絞り込むことができます。
- 仮想ドキュメントのメッセージ・ルーティング・ルールの場合、定義されている仮想ドキュメント・クラスのリストから選択できます。

**スキーマ・カテゴリ**

仮想ドキュメントのルーティング・ルールの場合、メッセージ・クラスのカテゴリを識別して、その構造を指定します。選択した仮想ドキュメント・クラスに定義されているカテゴリ・タイプのリストから選択できます。タイプは、組み込みのタイプか、カスタム・スキーマからインポートされたタイプです。

**ドキュメント名**

仮想ドキュメントのルーティング・ルールの場合、メッセージ構造を識別します。許容値はメッセージ・クラスに応じて異なります。選択した仮想ドキュメント・クラスに定義されているカテゴリ・タイプのリストから選択できます。タイプは、組み込みのタイプか、カスタム・スキーマからインポートされたタイプです。

**[ドキュメント名]** フィールドで複数の値を指定した場合、ルールは指定したいずれかの **[ドキュメント名]** 値と一致し、他の値とは一致しません。

# 3.5 ルールの無効化

ルール・セットがルールを実行しないようにしたいが、ルールは削除したくない場合、これを無効にすることができます。ルールをダブルクリックして、**[無効]** を選択するだけです。

# 3.6 データ変換へのデータの引き渡し

Send アクションは、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前に、データ変換を呼び出すことができます。このデータ変換では、その aux 変数を使用して、ルールから情報を取得できます。ルールの名前やルールが起動された理由など、このデータの一部は、ルール・クラスを変更することなく、この変換に使用できます。

変換に追加情報を渡すには、IDE でルール・クラスを編集し、クラスのプロパティに値を割り当てる必要があります。ルール・クラスの RuleUserData プロパティに割り当てられた値は、変換が aux.RuleUserData 変数にアクセスする場合に使用できます。ルール・クラスの RuleActionUserData プロパティに割り当てられた値は、aux.RuleActionUserData として変換で使用できます。

変換での aux 変数へのアクセスの詳細は、DTL の構文ルールの有効な式のリストを参照してください。

# 3.7 ビジネス・ルール通知の追加

InterSystems IRIS では、あるルールを実行するたびにシステムで特定のアクションが実行されるように、ルール通知を設定できます。ルールに関連する大半のアクティビティとは異なり、通知の設定にはプログラミングが必要です。Ens.Rule.Notification クラスのサブクラスを作成して、そのサブクラスの %OnNotify メソッドを上書きする必要があります。%OnNotify メソッドのシグニチャは、以下のとおりです。

```
ClassMethod %OnNotify(pReason As %String,
                      pRule As Ens.Rule.RuleDefinition)
                      As %Status
```

使用可能な pReason の値は次のとおりです。

- BeforeSave
- AfterSave
- Delete

実行時にプロダクション・フレームワークは自動的に Ens.Rule.Notification のサブクラスを検索し、%OnNotify 内のコードを使用してルールの実行時に行う操作を判断します。

# 4

# ルーティング・ルールのデバッグ

この章では、プロダクション全体でメッセージを送信することなく、ルーティング・ルールをテストする方法について説明します。また、プロダクションで EDI メッセージについて定義されているルーティング・ルールの問題をデバッグするのに役立つフロー・ダイアグラムも用意しています。

## 4.1 ルーティング・ルールのテスト

ルール・エディタの **[テスト]** ボタンを使用すると、プロダクション全体でメッセージを送信しなくても、メッセージがいずれかのルーティング・ルールをトリガするかどうかを確認できます。このテストを実行しても、メッセージの変換や送信は行われませんが、メッセージがプロダクションで実行されたかのように条件内の関数が実行されます。

メッセージのソースに基づいたルールの制約をテストするには、**[プロダクション・ソース]** フィールドを使用して、メッセージを送信しているプロダクション内のビジネス・ホストを指定します。ドロップダウン・メニューを使用して、リストからビジネス・ホストを選択できます。

**[コンテキス]** フィールドを使用して、次の 3 つのいずれかの方法でメッセージの内容を指定できます。

- [ユーザ入力] を指定してから [次へ] をクリックし、メッセージの未加工テキストを貼り付けます。
- 既存のメッセージのドキュメント本文 ID を指定します。メッセージ・ビューワの **[本文]** タブの **[<オブジェクト ID>]** フィールドで検索することにより、メッセージのドキュメント本文 ID を見つけることができます。
- 既存のメッセージのメッセージ・ヘッダ ID を指定します。**[プロダクション・ソース]** フィールドが空白の場合、メッセージ・ヘッダのソース構成名がソースとして使用されます。メッセージ・ビューワの **[ヘッダ]** タブの **[<オブジェクト ID>]** フィールドで検索することにより、メッセージのメッセージ・ヘッダ ID を見つけることができます。

### 4.1.1 メッセージの未加工のテキストによるテスト

仮想ドキュメントのメッセージ・ルーティング・ルールでは、メッセージの未加工のテキストでルールをテストできます。そのためには、以下のように操作します。

1. オプションで、**[プロダクション・ソース]** フィールドに、メッセージを送信しているビジネス・ホストを入力します。
2. **[コンテキスト]** ドロップダウン・リストから **[ユーザ入力]** を選択します。
3. **[次へ]** ボタンをクリックします。
4. **[コンテンツ]** テキスト・フィールドにメッセージの未加工のテキストを貼り付けます。

   これは、HL7 メッセージまたは X12 メッセージのテキストです。

5. テストしたいその他の制約については、[DocType] フィールドに情報を入力するか、[**カテゴリー**] または [**名前**] ドロップダウンから選択して入力します。
6. [**実行**] をクリックします。

### 4.1.2 テスト結果

テスト結果により、メッセージがルールの制約を満たしたかどうか、および if 節または else 節がトリガされたかどうかがわかります。

### 4.1.3 セキュリティ要件

ユーザがルーティング・ルールをテストするには、適切なセキュリティ特権を持っている必要があります。`%Ens_RuleLog` と `%Ens_TestingService` の USE 許可が必要です。さらに、`Ens_Rule.log` テーブルと `Ens_Rule.DebugLog` テーブルに対する Select SQL 特権も必要です。

## 4.2 ルーティング・ルールのデバッグ方針

ここでは、EDI メッセージ・ルーティング・プロダクションのルーティング・ルールのデバッグ方針について説明します。

ルーティング・ルールで発生する大きな問題として、メッセージが宛先に届かないという症状があります。これは多くの場合、ルーティング・プロダクション内部のビジネス・オペレーションやルーティング・プロセスなど、途中のポイントにはメッセージが到達しているものの、InterSystems IRIS® 外部のアプリケーション・サーバに存在することが普通のターゲットの宛先に届かない現象です。この場合は、次の“ルーティング・ルールに関する問題の解決”の図 A、図 B、図 C、および図 D に示す問題解決シーケンスに従うことができます。

図 4-1: ルーティング・ルールに関する問題の解決 (図 A)

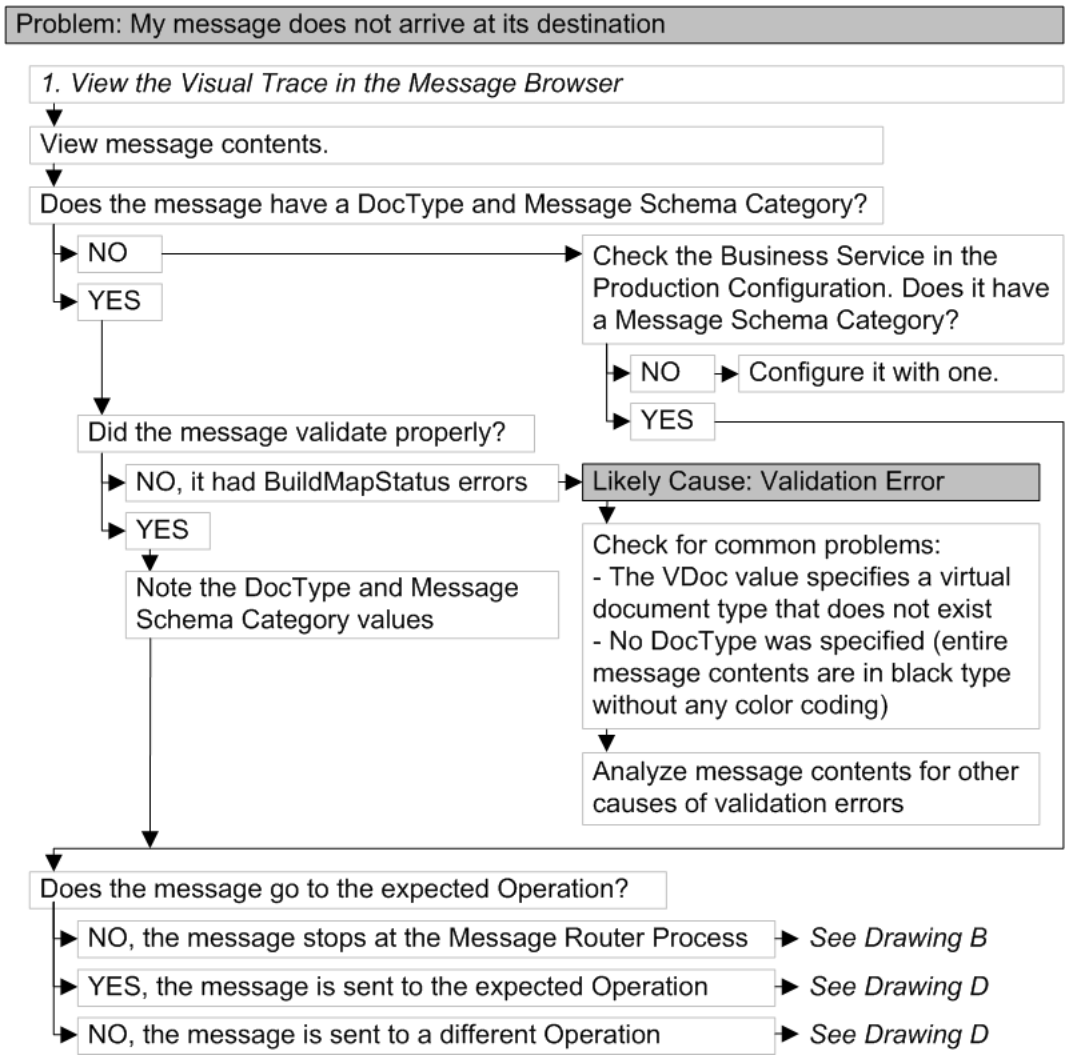


図 4-2: ルーティング・ルールに関する問題の解決 (図 B)

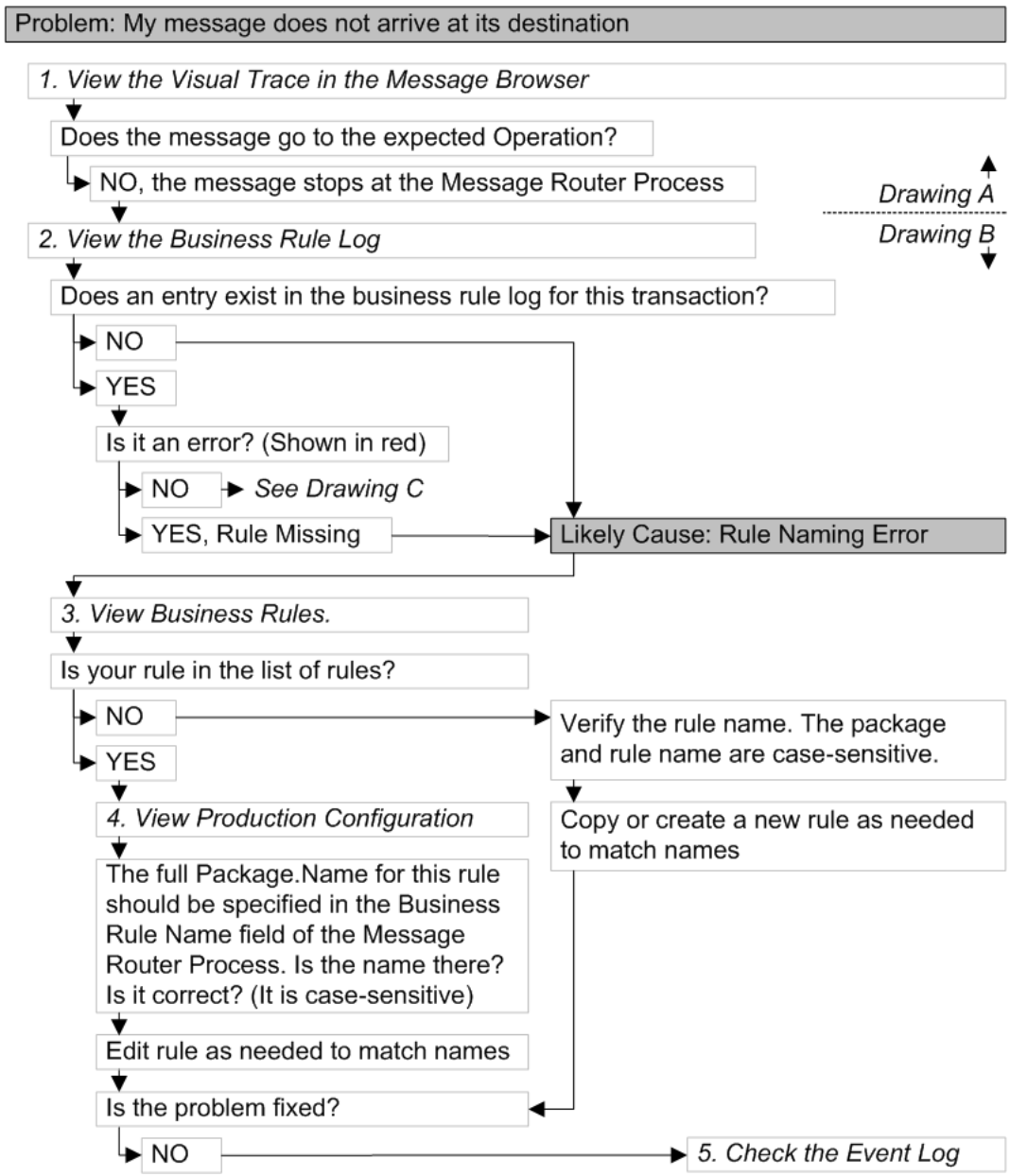


図 4-3: ルーティング・ルールに関する問題の解決 (図 C)

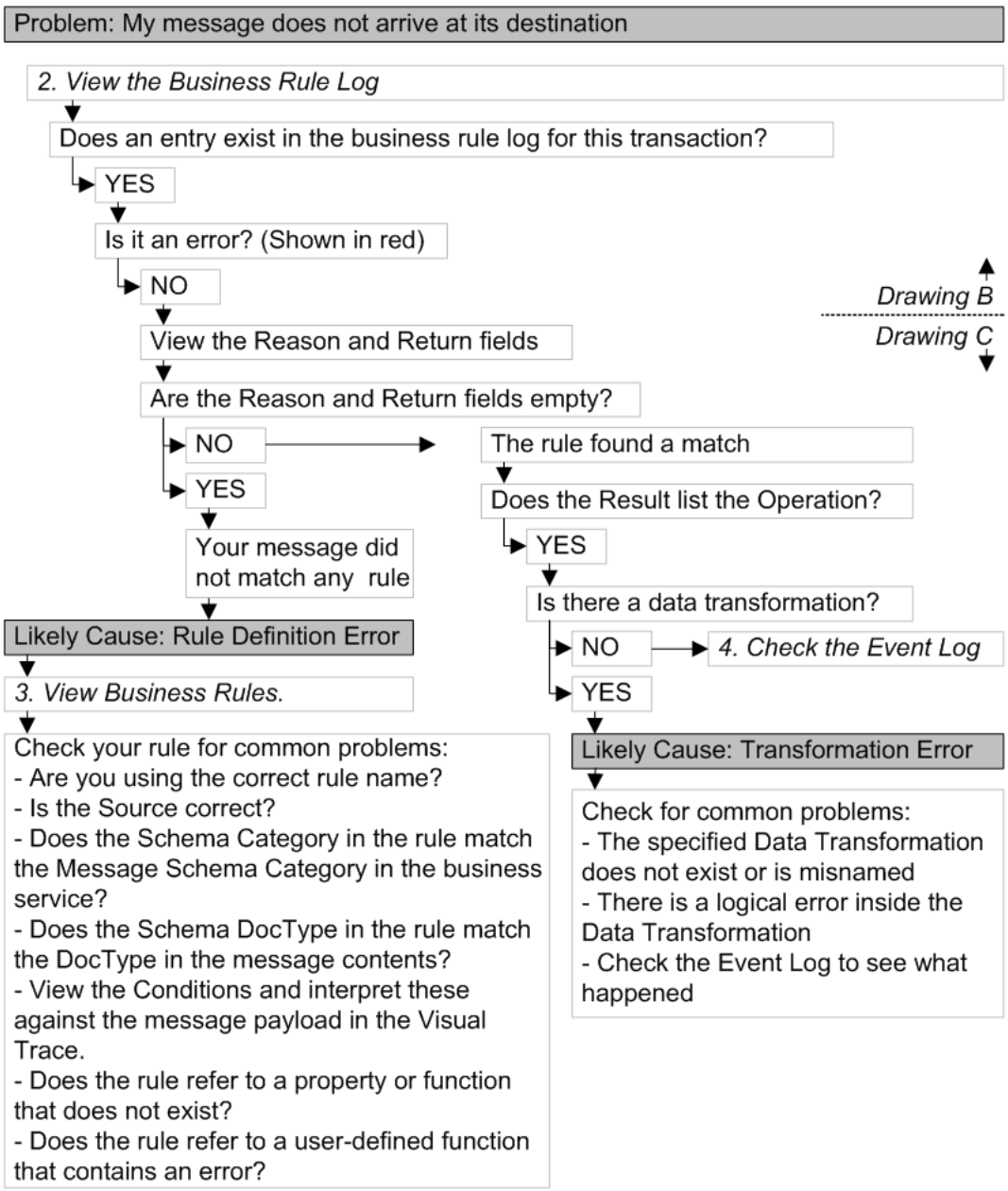
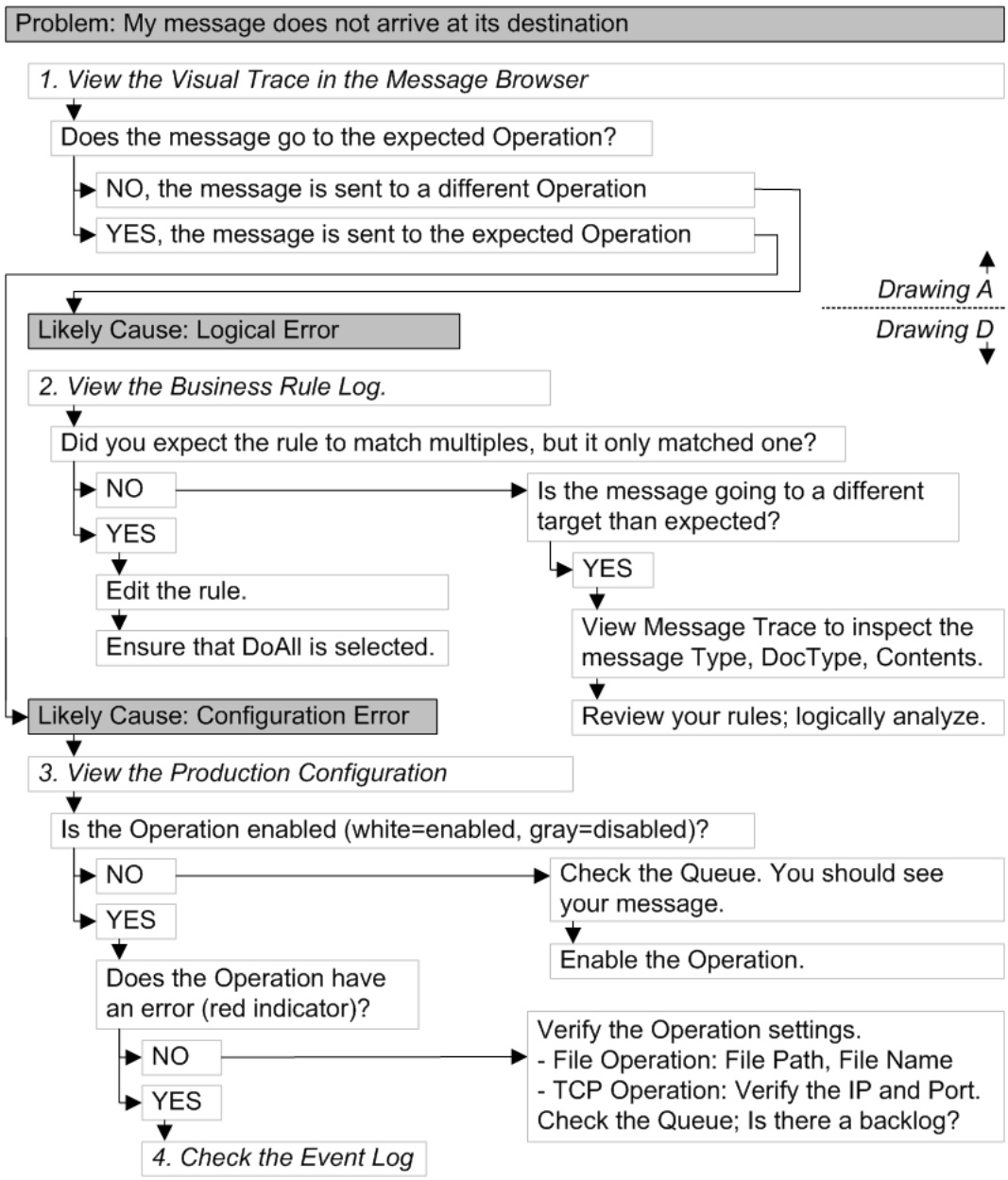


図 4-4: ルーティング・ルールに関する問題の解決 (図 D)



# A
# プロダクションで使用するユーティリティ関数

ここでは、ビジネス・ルールと DTL データ変換で使用可能な InterSystems IRIS® ユーティリティ関数について説明します。このような関数には、他のプログラミング言語で使い慣れている数学的な処理関数または文字列処理関数などがあります。

独自の関数を定義するには、"カスタム・ユーティリティ関数の定義" を参照してください。

## A.1 組み込み関数

以下のリストに、InterSystems IRIS に組み込まれているユーティリティ関数を示します。

注釈　　ブーリアン値の場合、1 は真、0 は偽を示します。

### Contains(val,str)

val に文字列 str が含まれる場合は 1 (真) を返します。それ以外は 0 (偽) を返します。

### ConvertDateTime (val,in,out,file)

in 形式のタイム・スタンプとして入力文字列 val を読み取り、その値を out 形式のタイム・スタンプに変換して返します。"ファイル名に関するタイム・スタンプ指定" を参照してください。

in および out のデフォルトは %Q です。out 引数の中の任意の %f 要素は、file 文字列に置き換えられます。val が in 形式に適合しない場合、out は無視され、val は変換されずに返されます。

### CurrentDateTime(format)

指定されたフォーマットで日付/時間値を表す文字列を返します。利用可能な形式のリストは、"FormatDateTime" メソッドのクラス・リファレンスの "Date and Time Expansion" のセクションを参照してください。例えば、`CurrentDateTime("%H")` は、24 時間形式での現在の時刻を 2 桁の数字で返します。デフォルト・フォーマットは、サーバのローカル・タイムゾーンでの ODBC フォーマット (%Q) です。

### DoesNotContain(val,str)

val に文字列 str が含まれない場合は 1 (真) を返します。

### DoesNotIntersectList(val,items,srcsep,targetsep)

指定されたソース・リスト (val) のアイテムがターゲット・リスト (items) に表示されない場合、1 (真) を返します。引数の詳細は、IntersectsList を参照してください。

### DoesNotMatch(val,pat)

val のパターンが、pat で指定されるパターンと一致しない場合は 1 (真) を返します。pat は、ObjectScript のパターン・マッチング演算子 ? に適した構文を使用した文字列とする必要があります。詳細は、"パターン・マッチング (?)" を参照してください。

### DoesNotStartWith(val,str)

val が文字列 str で始まらない場合は 1 (真) を返します。

### Exists(tab,val)

Exists 関数は、Lookup 関数の結果を予測する手段を提供します。Exists は、val が tab で指定されたテーブルの中で定義されたキーであれば 1 (真) を返します。それ以外は、0 (偽) を返します。

tab 値は二重引用符で囲まなければなりません。以下に例を示します。

```
Exists("Alert","Priority_FileOperation")
```

### If(val,true,false)

引数 val が 1 (真) に評価されると、If 関数は true 引数の文字列値を返します。それ以外は、false 引数の文字列値を返します。

### In(val,items)

カンマ区切り文字列 items の中に val がある場合は 1 (真) を返します。

### InFile(val,file)

特定された file の中に val がある場合は 1 (真) を返します。

### InFileColumn(...)

関数 InFileColumn can は、最大 8 個の引数を指定できます。完全な関数のシグニチャは、以下のとおりです。

InFileColumn(val, file, columnId, rowSeparator, columnSeparator, columnWidth, lineComment, stripPadChars)

InFileColumn は、テーブル形式のテキスト・ファイル file の中の指定された列に val がある場合、1 (真) を返します。引数の詳細は、以下のとおりです。

- val (必須) は値です。
- file (必須) はテキスト・ファイルです。
- デフォルトの columnId は 1 です。
- デフォルトの rowSeparator は ASCII 10 です。rowSeparator に負の値を指定すると、列の長さを示します。
- デフォルトの columnSeparator は ASCII 9 です。 columnSeparator が 0 の場合、ファイルのフォーマットを “位置基準” といいます。この場合、columnId は文字の位置、columnWidth は文字カウントを示します。
- デフォルトの columnWidth は 0 です。
- デフォルトの lineComment は空文字列です。
- デフォルトの stripPadChars は、1 つの空白文字とその後の ASCII 9 で構成されます。

### IntersectsList(val,items,srcsep,targetsep)

指定されたソース・リスト (val) のアイテムがターゲット・リスト (items) に表示される場合、1 (真) を返します。引数 srcsep および targetsep は、ソース・リストとターゲット・リストにそれぞれリスト・セパレータを指定します。各リスト・セパレータのデフォルトは `"><"` であり、リストのフォームが `"<item1><item2><item3>"` であると想定されます。

`IntersectsList` ユーティリティは、仮想ドキュメント・プロパティの値と照合する角かっこ [] 構文で適切に機能します。メッセージ内にそのセグメント・タイプのインスタンスが複数ある場合、角かっこ構文は複数の値を `<ValueA><ValueB><ValueC>` のような 1 つの文字列で返します。

ターゲット・リストのアイテムが 1 つだけの場合、この関数は基本的に Contains 関数と同じです。ソース・リストのアイテムが 1 つだけの場合、この関数は基本的に In 関数と同じです。

### Length(string,delimiter)

指定された文字列の長さを返します。delimiter を指定した場合、この関数はそのデリミタに基づいて文字列数を返します。

### Like(string,pattern)

指定された値 (string) が指定されたパターン文字列 (pattern) によって SQL Like 比較を満たす場合、1 (真) を返します。SQL Like パターンでは、% は 0 または複数文字に一致し、_ は単一文字に一致します。パターンに "%%" を追加することによってエスケープ文字を指定して (例: . "#%SYSVAR_#_%%%#")、"%SYSVAR" で始まり、後ろに単一文字、アンダースコア、その他が続く任意の値文字列に一致させることができます。

### Lookup(table,keyvalue,default, defaultOnEmptyInput)

Lookup() 関数は、table で指定されたテーブル内の keyvalue で指定されたキー値を検索して、関連する値を返します。この戻り値は以下のグローバルと等価です。

`^Ens.LookupTable(`table,keyvalue`)`

table 値は二重引用符で囲まなければなりません。以下に例を示します。

```
Lookup("Gender",source.{PID:Sex},,"U")
```

キーがテーブル内で見つからなかった場合は、default パラメータで指定されたデフォルト値が返されます。default パラメータは省略可能なため、それが指定されず、Lookup で一致するキーが見つからなかった場合は、空の文字列が返されます。例外として、ルックアップ・テーブルまたはキー値が空の場合に、Lookup() 関数によって、以下のテーブルに示された defaultOnEmptyInput パラメータの値に従ってデフォルト値または空の文字列のいずれかが返されます。defaultOnEmptyInput パラメータの既定値は 0 です。

| defaultOnEmptyInput の値 | キー値とルックアップ・テーブル | Lookup() の戻り値 |
|---|---|---|
| 0 | キー値またはルックアップ・テーブルが空 | 空の文字列 |
| 1 | キー値が空 | 空の文字列 |
| | ルックアップ・テーブルは空だが、キー値は空ではない | デフォルト値 |
| 2 | ルックアップ・テーブルが空 | 空の文字列 |
| | キー値は空だが、ルックアップ・テーブルは空ではない | デフォルト値 |
| 3 | キー値またはルックアップ・テーブルが空 | デフォルト値 |

Exists() 関数は、同じパラメータを使用する Lookup() 関数がルックアップ・テーブルでキー値が検出する場合に、真を返します。

### Matches(val,pat)

val のパターンが、pat で指定されたパターンと一致する場合は 1 (真) を返します。pat は、ObjectScript のパターン・マッチング演算子 ? に適した構文を使用した文字列とする必要があります。詳細は、"パターン・マッチング (?)" を参照してください。

### Max(...)

最高 8 つの値を含むリスト内の最大値を返します。リストのエントリはカンマで区切られています。

### Min(...)

最高 8 つの値を含むリスト内の最小値を返します。リストのエントリはカンマで区切られています。

### Not(val)

val が 1 (真) の場合、0 (偽) を返します。val が 0 (偽) の場合、1 (真) を返します。

### NotIn(val,items)

カンマ区切り文字列 items の中に val がない場合、1 (真) を返します。

### NotInFile(val,file)

特定された file の中に val がない場合、1 (真) を返します。

### NotLike(string,pattern)

指定された値 (string) が指定されたパターン文字列 (pattern) によって SQL Like 比較を満たさない場合、1 (真) を返します。"Like" を参照してください。

### Pad(val,width,char)

入力文字列 val を読み取り、この文字列が width の文字列長になるまで char の文字を追加します。width に正の値を指定すると、val 文字列の右側に文字が追加されます。width に負の値を指定すると、val 文字列の左側に文字が追加されます。

### Piece(val,char,from,to)

区切り文字 char が文字列 val の中にあると、ここで文字列が分断されます。文字列が複数に分断される場合、返される分断文字列の範囲を from と to で指定します。最初の分断文字列が 1 になります。複数の分断文字列が返される場合、返される文字列には入力文字列と同じ区切り文字が使用されます。以下に例を示します。

Piece("A,B,C,D,E,F") は "A" を返します。

Piece("A!B!C!D!E!F","!",2,4) は "B!C!D" を返します。

デフォルトの char はカンマ、デフォルトの from は 1、デフォルトの to は from です (1 ピースのみを返します)。詳細は、"$PIECE (ObjectScript)" を参照してください。

### ReplaceStr(val,find,repl)

入力文字列 val を読み取り、文字列 find があったらすべて repl に置き換えて、その文字列を返します。

注釈　Replace 関数は廃止されたため、代わりに ReplaceStr を使用してください。

RegexMatch(string,regex)

入力文字列 val と正規表現 regex が指定されている場合、val がその正規表現に一致すれば 1 を返し、一致しなければ 0 を返します。

Round(val,n)

val の値を小数点以下 n 桁に丸めて返します。n が指定されていない場合 (Round(val)) は、小数点以下の桁を切り捨て、1 の位に丸めることで、整数が生成されます。

Rule(rulename,context,activity)

ルール・ログに対して指定された context オブジェクトと指定された activity ラベルを使用して、rulename に指定されたルールを評価します。

Schedule(ScheduleSpec, ODBCDateTime)

指定された ScheduleSpec 文字列の状態、ODBCDateTime により指定された時点で指定された Schedule または Ruleの状態を評価します。ScheduleSpec が '@' で始まる場合は、スケジュール名またはルール名を示します。そうでない場合は、生の Schedule 文字列を示します。ODBCDateTime が空白の場合は、現在の時刻に対して評価が行われます。

StartsWith(val,str)

val が文字列 str で始まる場合、1 (真) を返します。

Strip(val,act,rem,keep)

入力文字列 val を読み取り、act テンプレートおよび rem 文字列で指定したカテゴリに適合する文字をすべて削除し、keep 文字列にある文字はすべて残します。その結果の文字列を返します。 詳細と例は、"$ZSTRIP (ObjectScript)" を参照してください。

SubString(str,n,m)

文字列 str の部分文字列を返します。開始位置 n と終了位置 m を数字で指定します。数字の 1 は文字列の最初の文字を示します。m が指定されていない場合 (SubString(str,n)) は、位置 n から文字列の最後までの部分文字列を返します。

ToLower(str)

文字列 str を小文字に変換して返します。

ToUpper(str)

文字列 str を大文字に変換して返します。

Translate(val,in,out)

入力文字列 val を読み取り、文字列 in にある 1 文字を、文字列 out の同じ位置にある文字にすべて変換して返します。

注釈　これらの関数は、Ens.Util.FunctionSet クラス内のメソッドで定義されます。

# A.2 関数の呼び出し構文

ビジネス・ルールまたは DTL データ変換で関数を参照する場合、構文には必ずかっこが含まれます。また、数値関数 Min、Max、Round の数値などの入力パラメータも含める必要があります。関数に入力値がない場合は、中身が空の左右のかっこを含める必要があります。

次に、有効な関数の構文の例を示します。

| 式 | 算出値 |
|---|---|
| `Min(Age,80,Limit)` | `Age` が値 `30` を持つプロパティであり、同じくプロパティである `Limit` の値が `65` であるとき、この式の値は `30` になります。 |
| `Round(1/3,2)` | `0.33` |
| `Min(10,Max(X,Y))` | `X` が数値 `9.125` を持つプロパティであり、同じくプロパティである `Y` の数値が `6.875` であるとき、この式の値は `9.125` になります。 |

関数への値の入力が、数値で始まる文字列である場合、文字列内の数値以外の文字は破棄され、数値部分のみが使用されます。文字列 `"3a"` は、`3` の数値として取り扱われるため、関数 `Min("3a","2OfThem")` からは `2` の値が返されます。`"a123"` のような数字以外の文字で始まる文字列の場合は、数値 `0` となります。

ビジネス・ルールのユーティリティ関数の構文は、以下のように大幅に DTL 構文と異なります。

- ビジネス・ルールは、ユーティリティ関数を単純に名前で参照します。

  ```
  ToUpper(value)
  ```

- DTL では関数名の直前に 2 つのドットを使用して、関数をメソッドのように扱います。

  ```
  ..ToUpper(value)
  ```

以下の DTL データ変換では、ユーティリティ関数 ToUpper() を使用して文字列をすべて大文字に変換します。この <assign> 文は、クラスのメソッドのように、ダブルドット構文を使用して ToUpper() を参照します。

### Class Definition

```
Class User.NewDTL1 Extends Ens.DataTransformDTL
{

XData DTL
{
<?xml version="1.0" ?>
<transform targetClass='Demo.Loan.Msg.Approval'
           sourceClass='Demo.Loan.Msg.Approval'
           create='new' language='objectscript'>
<assign property='target.BankName'
        value='..ToUpper(source.BankName)' action='set'/>
<trace value='Changed all lowercase letters to uppercase!' />
</transform>
}

}
```