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

Version 2023.1
2024-01-02

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

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
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

BPL 要素

このリファレンスでは、BPL 要素のそれぞれについて詳しく説明します。

Tipヒント BPL 用の XML を表示または編集する場合は、スタジオを使用して BPL を編集し、**[他のコードを表示]** をクリックします。

一般的な属性と要素

ほとんどの BPL 要素に存在する属性と要素について説明します。

一般的な属性

ほとんどの BPL 要素には、以下のリストに簡単にまとめてある属性を含めることができます。

属性	説明	値
name	通常はオプション。この要素の名前。	0 ～ 255 文字の文字列。
disabled	オプション。disabled 属性を 1 (真) に設定することにより、要素を一時的に無効にできます。要素を再度有効にするには、disabled 属性を削除するか、またはこの属性を 0 (偽) に設定します。	1 (真) または 0 (偽) のブーリアン値。
xpos	オプション。BPL ダイアグラムで、この要素を表すグラフィックの x 座標を設定します。BPL コンパイラでは無視されます。	正の整数。
ypos	オプション。y 座標。	正の整数。
xend	オプション。この要素を表すグラフィックに 2 つのアイコン (開始と終了) がある場合、xend は、終了アイコンの x 座標を設定します。BPL コンパイラでは無視されます。	正の整数。
yend	オプション。終了アイコンの y 座標。	正の整数。

一般の要素: <annotation>

ほとんどの BPL 要素には <annotation> 要素を含めることができ、BPL ダイアグラムのシェイプに説明テキストを関連付けることができます。この要素は、以下のとおりです。

```
<annotation>
    Gets the current Account Balance for a customer.

</annotation>
```

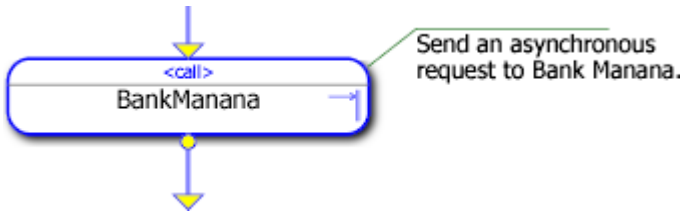
CDATA ブロック内のこのテキストは、関連するアクティビティの注釈として表示されます。以下の例は、<call> アクティビティの <annotation> です。

XML

```
<call name="BankManana">
  <annotation>
    Send an asynchronous
      request to Bank Manana.


  </annotation>
</call>
```

CDATA ブロックでは、改行を挿入したり、XML エスケープ・シーケンスを使用せずにアポストロフィ (') などの特殊文字を入力できます。上記の例では、`asynchronous` と `request` の間に改行が挿入されています。BPL ダイアグラムでも、下図のように、まったく同じように表示されます。



〈annotation〉 文字列の最大長は、CDATA エスケープ文字も含めて 32,767 文字です。

〈alert〉

ビジネス・プロセスの実行中、ユーザ・デバイスに警告メッセージを送信します。

構文

```
<alert value="The system needs service right away."/>
```

詳細

属性または要素	説明	値
value 属性	必須項目。警告メッセージのテキスト。	1 文字以上の文字列。式またはリテラル文字列を使用できます。式の場合は、この属性が含まれる 〈process〉 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
〈annotation〉 要素		

説明

〈alert〉 要素は、ユーザのデバイスにアラート・メッセージを送信します。

警告メッセージのテキストは、常に、アラート・タイプのエントリとして InterSystems IRIS® イベント・ログに書き込まれます。ただし、〈alert〉 要素の一番の目的は、携帯電話や電子メールなどの通知デバイスを介してユーザに連絡することです。そのため、〈alert〉 要素は、Ens.Alert と呼ばれる構成項目にメッセージのテキストを送信します。この構成項目には、InterSystems IRIS 外部のユーザ・デバイスに連絡するのに必要なすべての情報が設定されています。

重要　　プロダクションのメンバとして構成された Ens.Alert 項目が存在しない場合は、〈alert〉 がイベント・ログに書き込まれるだけです。

詳細は、"プロダクションの開発" の “アラート・プロセッサの定義” を参照してください。

〈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。

構文

```
<assign property="propertyname" value="expression" />
```

詳細

属性または要素	説明	値
property 属性	必須項目。この割り当てのターゲット。実行コンテキスト・オブジェクトのプロパティを指定する必要があります（通常は、context、request、response、callrequest、または callresponse）。詳細は、以下の "説明" トピックの表を参照してください。	1 文字以上の文字列。
value 属性	必須項目。プロパティの値。	プロパティに有効な値を返すリテラル値または式。式の場合は、この属性が含まれる〈process〉要素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。
action 属性	オプション。property がコレクション（リストまたは配列）の場合は、action を使用して、そのコレクションに対して行う割り当てのタイプを指定します。指定しない場合、“set” が実行されます。	リテラル文字列 “append”、“set”、“clear”、“insert”、または “remove”（この後の説明を参照）。
key 属性	property がコレクション（リストまたは配列）である一部の場合を除き、オプション。コレクションの場合は、このキーを使用して、この割り当てのターゲットであるコレクションのメンバを指定します。	キーを求める式。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
〈annotation〉要素		

説明

ここでは、BPL ビジネス・プロセスにおける実行コンテキストの重要性、および 〈assign〉 要素を使用して、ビジネス・プロセスの実行コンテキストの値を設定する方法を説明します。

ビジネス・プロセスでは、実行を中断するたびに状態情報をディスクに保存し、再開するときは、その状態情報をディスクからリストアする必要があります。この機能は、完了までに数日または数週間かかる可能性のある長期実行のビジネス・プロセスにおいては特に重要です。この要件に対応するため、InterSystems IRIS のすべての BPL ビジネス・プロセスには、実行コンテキストと呼ばれる一連のオブジェクトと変数があります。実行コンテキストの変数は、BPL ビジネス・プロセスが実行を中断および再開するたびに自動的に保存、リストアされます。BPL ビジネス・プロセスを適切に実行するには、これらの変数を正しく使用する必要があります。

実行コンテキストの変数には、それぞれ特定の名前と目的が与えられています。これらの変数には、〈assign〉 要素を使用して値を設定できます。以下は、実行コンテキストの変数一覧です。

変数	目的
callrequest	callrequest オブジェクトには、<call> が送信する要求メッセージ・オブジェクトの構築に必要となるすべてのプロパティが含まれます。対応する <request> アクティビティ内で一連の <assign> 要素を使用して、callrequest にプロパティ値を設定します。
callresponse	<call> アクティビティが完了すると、callresponse オブジェクトには、<call> に返される応答メッセージ・オブジェクトのプロパティが追加されます。対応する <response> アクティビティ内で一連の <assign> 要素を使用して、callresponse のプロパティから context または response のプロパティに戻り値をコピーします。
context	context オブジェクトは、ビジネス・プロセスの汎用データ・コンテナです。context は自動的には定義されません。このオブジェクトのプロパティを定義するには、<context> 要素を使用します。その場合、<process> 要素内でこれらのプロパティを参照するには、`context.Balance` のようにドット構文を使用します。
request	request オブジェクトには、このビジネス・プロセスをインスタンス化させた元の要求メッセージ・オブジェクトのプロパティが含まれます。<process> 要素内で request のプロパティを参照するには、`request.UserID` のようにドット構文を使用します。
response	response オブジェクトには、ビジネス・オブジェクトが返す最終的な応答メッセージ・オブジェクトの構築に必要となるすべてのプロパティが含まれます。<process> 要素内で response のプロパティを参照するには、`response.IsApproved` のようにドット構文を使用します。これらのプロパティに値を割り当てるには <assign> 要素を使用します。
status	status は、成功または失敗を示す %Status 型の値です。BPL ビジネス・プロセスの実行中に status が失敗値を受け取ると、そのビジネス・プロセスは即座に終了し、失敗の理由を示すテキスト・メッセージがイベント・ログに書き込まれます。通常、<call> アクティビティから不成功値が返された時点で、この処理が自動的に行われます。ただし、<assign> または <code> を使用して status 値を設定すれば、BPL ビジネス・プロセス・コードをいつでも正常に終了できます。このトピックの最後の説明を参照してください。
syncresponses	syncresponses は応答オブジェクトのコレクションであり、同期している <call> アクティビティの名前がキーとなります。完了した呼び出しのみが示されます。syncresponses から応答を取得できるのは、<sync> を実行してから、現在の <sequence> の終了までの間に限られます。該当する呼び出しが `<call name="MyName">` として定義されている `syncresponses.GetAt("MyName")` 構文を使用して応答を取得します。
synctimedout	synctimedout 値は整数です。synctimedout は、いくつかの呼び出し後の <sync> アクティビティの結果を示します。synctimedout の値をテストできるのは、<sync> を実行してから、該当する呼び出しと <sync> を含む <sequence> の終了までです。synctimedout の値は、以下の 3 つのいずれかになります。 ・ 0 の場合、どの呼び出しもタイムアウトになっていません。すべての呼び出しが時間内に完了しました。<sync> アクティビティに timeout が設定されていない場合も、この値が返されます。 ・ 1 の場合、1 つ以上の呼び出しがタイムアウトになりました。つまり、時間切れのため、完了できなかった <call> アクティビティがあります。 ・ 2 の場合、1 つ以上の呼び出しが中断されました。 通常、synctimedout を取得してステータスを確認し、その後、syncresponses コレクションの完了した呼び出しから応答を取得します。

注意　その他すべての実行コンテキスト変数名と同様、status は BPL の予約語です。上記の場合を除き、<assign> でこの予約語を使用しないでください。

BPL の <assign> 要素は、ターゲットと、そのターゲットに割り当てられる式を指定します。ターゲットとしては、ビジネス・プロセスの実行コンテキストのいずれかのオブジェクトのプロパティ、またはいずれかの単値変数 (status など) を使用できます。<assign> 要素に含まれるプロパティは、データ型、オブジェクト、またはこれらいずれかのコレクションです。コレクション・プロパティを宣言するには、対応する <property> 要素で、collection 属性を “array” または “list” に設定します。

上の表で説明したように、context と呼ばれるオブジェクトは、ビジネス・プロセスの汎用コンテキスト・オブジェクトとして機能します。context オブジェクトのプロパティを定義するには、<process> 環境の先頭に <context> 要素と <property> 要素を配置します。以下に例を示します。

XML

```
<process request="Demo.Loan.Msg.Application" response="Demo.Loan.Msg.Approval">
  <context>
    <property name="CreditRating" type="%Integer"/>
    <property name="PrimeRate" type="%Numeric"/>
  </context>
  ...
</process>
```

上記の BPL では、このビジネス・プロセスの 2 つの context プロパティ (context.CreditRating と context.PrimeRate) を定義しています。ただし、値は割り当てていません。<process> 環境内で、この <context> 要素の下に <assign> 要素を配置すれば、必要に応じてこれらのプロパティに値を割り当てることができます。以下に例を示します。

XML

```
<process request="Demo.Loan.Msg.Application" response="Demo.Loan.Msg.Approval">
  <context>
    <property name="CreditRating" type="%Integer"/>
    <property name="PrimeRate" type="%Numeric"/>
  </context>
  <sequence>
    <call name="PrimeRate" target="Demo.Loan.WebOperations" async="0">
      <request type="Demo.Loan.Msg.PrimeRateRequest">
      </request>
      <response type="Demo.Loan.Msg.PrimeRateResponse">
        <assign property="context.PrimeRate" value="callresponse.PrimeRate"/>
      </response>
    </call>
    ...
  </sequence>
  ...
</process>
```

上の BPL は最初の例の続きです。この例の <call> 要素は同期しており、<request> 要素と <response> 要素が両方とも含まれていることに注意してください。

この場合の <response> には、実行のコンテキストでオブジェクトの 2 つのプロパティを参照する <assign> 操作が記述されています。そのプロパティの 1 つは、汎用コンテキスト・オブジェクトのプロパティである context.PrimeRate です。もう 1 つのプロパティは、現在の <call> 要素に関連付けられた応答オブジェクト (ここでは Demo.Loan.Msg.PrimeRateResponse) のプロパティである callresponse.PrimeRate です。<assign> 操作は、<call> から返される PrimeRate プロパティの値を受け取り、その値を汎用コンテキスト・オブジェクトに配置します。

上に示す <sequence> 要素内の、<call> 要素の後は以下のようになります。

XML

```
<call name="CreditRating" target="Demo.Loan.WebOperations" async="0">
  <request type="Demo.Loan.Msg.CreditRatingRequest">
    <assign property="callrequest.SSN" value='request.SSN'/>
  </request>
  <response type="Demo.Loan.Msg.CreditRatingResponse">
    <assign property="context.CreditRating" value="callresponse.CreditRating"/>
  </response>
</call>
```

上記の文は、基本要求 (request.SSN) の SSN プロパティを、現在の <call> 要素 (callrequest.SSN) によって作成される要求の SSN プロパティに割り当てます。この割り当ての後で、<call> 要素は要求を発行します。これは、タイプ Demo.Loan.WebOperations の同期呼び出しです。応答を受け取った <call> 要素は、<call> (callresponse.CreditRating) から返された CreditRating プロパティの値を取得して、汎用コンテキスト・オブジェクトのプロパティ (context.CreditRating) に配置します。

次の文は、整数値 1 を、ビジネス・プロセス (response.IsApproved) の基本応答オブジェクト内の IsApproved プロパティに割り当てます。この例では、IsApproved は、InterSystems IRIS の規則に応じてブーリアン値 (真または偽) になります。つまり、整数値 1 は真 (申込者が承認されたこと) を示し、0 は偽 (申込者が否認されたこと) を示します。

XML

```
<assign name='IsApproved' property="response.IsApproved" value="1">
  <annotation>
    Copy IsApproved into the response object.


  </annotation>
</assign>
```

以下の文は、計算値 (汎用コンテキスト・オブジェクト内の 2 つのプロパティを使用した式の結果) を、ビジネス・プロセス (response.InterestRate) の基本応答オブジェクト内の InterestRate プロパティに割り当てます。

XML

```
<assign name='InterestRate'
        property="response.InterestRate"
        value="context.PrimeRate+1+(2*(1-(context.CreditRating/100)))">
  <annotation>
    Copy InterestRate into the response object.


  </annotation>
</assign>
```

<assign> オペレーションのタイプ

BPL の <assign> 要素の構文は以下のように機能します。

1. property 属性は、オブジェクトと、割り当て操作のターゲットであるプロパティを識別します。
2. value 属性は、ターゲット・プロパティの値を指定します。これは、割り当て値を指定するために、実行時に評価される式になる場合もあります。<assign> 要素内の式は、そのビジネス・プロセスの <process> 要素で指定された言語を使用する必要があります。
3. BPL の <assign> オペレーションにはいくつかのタイプがあります。これらは、オプションの action 属性で指定します。action 属性には、以下の値を指定できます。

値	説明
“append”	一覧の最後にターゲット要素を追加します。
“set”	(デフォルト) 新しい値にターゲット要素を設定します。
“insert”	集合に新しい値を挿入します。
“remove”	集合からターゲット要素を削除します。
“clear”	ターゲット集合の内容を消去します。

デフォルト値の “set” を除くと、このようなアクションは、ほとんどがコレクション・プロパティに関係した割り当てを処理するためのものです。以下のテーブルは、さまざまな割り当てタイプについてまとめたものです。

プロパティのタイプ	action 属性の値	key 属性の必要性	結果
非コレクション	“set”	なし	プロパティは新しい値に設定されます。
配列	“clear”	なし	配列が消去されます。
配列	“remove”	あり	key の要素が削除されます。
配列	“set”	あり	key の要素が新しい値に設定されます。
リスト	“append”	なし	リストの末尾に要素が追加されます。
リスト	“clear”	なし	リストが消去されます。
リスト	“insert”	あり	key で指定した位置に要素が挿入されます。
リスト	“remove”	あり	key の要素が削除されます。
リスト	“set”	あり	key の要素が置換されます。

以下に、BPL <assign> 操作の各タイプについて詳しく説明します。

append (追加) 操作

“append” 操作では、リスト・プロパティの末尾にターゲット要素が追加されます。

set (設定) 操作

“set” 操作では、指定したプロパティの値が value 属性の値に設定されます。value 属性は、以下のように、式を含んでおり、それ自体が実行コンテキスト内のオブジェクトまたはオブジェクトのプロパティを参照できます。

XML

```
<assign name='CopyResult' property='context.SSN' value='callresponse.SSN' />
```

ターゲット・プロパティが配列コレクションの場合、key 属性の値は配列内のアイテムを指定します。それ以外の場合、key 属性は無視されます。

ターゲット・プロパティがコレクションで、value 属性で以下のように同じタイプが指定されている場合は、コレクションの内容がターゲット・コレクションにコピーされます。

XML

```
<assign name='CopyResults' property='context.List' value='callresponse.List' />
```

assign 要素のデフォルトのアクションは set 操作です。action が指定されていない場合、assign では set 操作が指定されます。

clear (消去) 操作

この操作は、コレクション・プロパティにのみ適用されます。“clear” 操作では、指定したコレクション・プロパティが消去されます。value 属性と key 属性は無視されます。ただし、<assign> 要素の BPL スキーマでは必要なので、文に value 属性を含める必要があります。

例えば、以下の文では、コレクション・プロパティ List が消去されます。

XML

```
<assign name='ClearResults' property='context.List' action='clear' value='' />
```

insert (挿入) 操作

この操作は、リストのコレクション・プロパティにのみ適用されます。“insert”操作では、指定したコレクション・プロパティに値を挿入します。key 属性が存在する場合は、key で指定された位置 (整数) の後ろに新しい値が挿入されます。指定がない場合、新しい項目は末尾に追加されます。

例えば、以下の例では、キー“primary”を使用して、配列コレクション・プロパティ Array に値を挿入します。

XML

```
<assign name='Ins' property='context.Array'
        action='insert'
        key='primary'
        value='request.Primary' />
```

remove (削除) 操作

この操作は、コレクション・プロパティにのみ適用されます。“remove”操作では、指定したコレクション・プロパティからアイテムを削除します。value 属性は無視されます。ただし、<assign> 要素の BPL スキーマではこの属性が必須なので、文に value 属性を記述しておく必要があります。

ターゲット・プロパティが配列コレクションの場合、key 属性の値は配列内のアイテムを指定します。それ以外の場合、key 属性は無視されます。

例えば、以下では、キー“abc”を使用して、配列プロパティ Array から要素を削除します。

XML

```
<assign name='Remove' property='context.Array' action='remove'
        key='abc' value='' />
```

<assign> を使用したステータス変数の設定

status は %Status 型のビジネス・プロセスの実行コンテキスト変数であり、成功または失敗を表します。

注釈　BPL ビジネス・プロセスのエラー処理は自動的に実行されます。BPL ソース・コードで status 値をテストまたは設定する必要はありません。ここでは、特殊な状況下で BPL ビジネス・プロセスを終了しなければならない場合に備えて、status 値が設けられています。

BPL ビジネス・プロセスが開始されると、status には成功を示す値が自動的に割り当てられます。status に成功値が割り当てられているかどうかを確認するには、ObjectScript ではマクロ `$$$ISOK(status)` を使用します。このテストで True 値が返された場合、status には成功値が格納されています。

BPL ビジネス・プロセスの実行中に status が失敗値を受け取ると、そのビジネス・プロセスは即座に終了し、該当するテキスト・メッセージがイベント・ログに書き込まれます。この処理は、status がどのような状況で失敗値を受け取ったかにかかわらず実行されます。したがって、status に失敗値を設定すれば、BPL ビジネス・プロセスをいつでも正常に終了することができます。

<assign> 要素を使用すれば、status に失敗値を設定できます。その場合、通常は、まず <if> 要素を使用して以前のアクティビティの結果を確認します。次に、<true> 要素または <false> 要素内で、失敗条件が存在するとき、<assign> を使用して status に失敗値を設定します。

BPL ビジネス・プロセスでは、<process> 内であればどこでも status を使用できます。status を参照するための構文は、%Status 型の変数の場合と同じです。つまり `status` となります。

関連項目

<call>、<context>

<branch>

条件に基づいて、実行フローを直接変更します。

構文

```
<branch condition="myVar='1'" label="JumpToMe" />
```

詳細

属性または要素	説明	値
condition 属性	必須項目。この式が真の場合、指定した <label> に制御フローがジャンプします。	整数値 1（真の場合）または 0（偽の場合）を求める式。この式では、この属性が含まれる <process> 要素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。
label 属性	必須項目。ジャンプ先となる <label> の名前。	0 ～ 255 文字の文字列。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		

説明

<branch> 要素の condition 式が真の値の場合、実行フローが即座に変更されます。<branch> の label 属性に値として名前が指定されている <label> 要素に制御が渡されます。

以下の BPL 例では、condition 式が真の場合、condition 値が TraceSkipped である <branch> から、name 値が TraceSkipped である <label> に直接制御が渡されます。この間にある <trace> 要素は無視されます。

```
<branch condition="myVar='1'" label="TraceSkipped" />
<trace value="Ignore me when myVar is 1..." />
<label name="TraceSkipped" />
```

<branch> の condition 式が偽の場合は、<branch> の次に記述されている BPL 文に制御が渡されます。つまり、上記の例では <trace> に制御が移ります。

ジャンプ先の <label> は、これを参照する <branch> と同じスコープにあることが条件となります。例えば、以下のようになっているとします。

- <flow> 内の各 <sequence> 要素には、それぞれ異なる <label> スコープが割り当てられます。BPL 実行エンジンでは、現在の <sequence> コンテナの外部にある <label> への <branch> は阻止されます。
- 実行時に実行フローを制御するその他の BPL コンテナ要素についても、同様の制約が適用されます。各コンテナには、それぞれ異なる <label> スコープが割り当てられます。

これらの制約に加え、各 <label> name 値は、現在のスコープ内だけでなく、BPL ビジネス・プロセス全体で一意にする必要があります。

注意　すべてのプログラミング言語に言えることですが、BPL 分岐メカニズムの使用時には注意が必要です。BPL エディタは、無限ループや無効な分岐といった基本的なプログラミング・エラーを防ぐことができません。

<break>

ループから抜けて、ループ・アクティビティを終了します。

構文

```
<break/>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	

説明

BPL 構文では、必要に応じて、一連のアクティビティを含めることができるすべての要素 (<case>、<default>、<foreach>、<false>、<sequence>、<true>、<until>、または <while>) に <break> 要素を含めることができます。

<break> 要素を使用すれば、制御フローで、含まれているループ内部の処理を中断して、ループを即座に終了できます。以下に例を示します。

XML

```
<while condition="0">

  //...do various things...

  <if condition="somecondition">
    <true>
      <break/>
    </true>
  </if>

  //...do various other things...

</while>
```

上記の例では、<break> 要素は <true> 要素に含まれています。ただし、この <break> 要素が実際に影響を与えるループは、この <break> を含む <while> ループです。

この例は、次のように機能します。このループにおいて、<if> 要素の “somecondition” が真になる (整数 1 に等しくなる) と、<if> 内の <true> 要素に制御が渡されます。<break> 要素が見つかると、その <break> 要素を含む <while> ループが即座に終了し、</while> の後続文に移動します。

<break> 要素を使用して変更するループ・アクティビティには、<foreach>、<until>、<while> があります。

注釈　<assign> 文または <code> 文を使用して、ビジネス・プロセスの実行コンテキスト変数 status に失敗値を設定すれば、BPL ビジネス・プロセス・コードをいつでも正常に終了できます。

関連項目

<continue>

<call>

ビジネス・オペレーションまたは別のビジネス・プロセスに要求を送信します。

構文

```
<call name="Call" target="MyApp.MyOperation" async="1">
   <request type="MyApp.Request">
     ...
   </request>
   <response type="MyApp.Response">
     ...
   </response>
</call>
```

詳細

ここに別途指定のない限り、各属性は必須です。

属性または要素	説明	値
name 属性	必須項目。<call> 要素の名前。リテラル文字列として指定するか、または @ 間接演算子でコンテキスト変数の値を参照することによって指定します。 <sync> 要素を使用して非同期呼び出しから応答を取得する場合は、name を使用して <sync> 要素を参照します。	1 ～ 255 文字の文字列
target 属性	必須項目。要求の送信先であるビジネス・オペレーションまたはビジネス・プロセスの構成名。この値は、リテラル文字列として指定するか、または @ 間接演算子でコンテキスト変数の値を参照することによって指定します。	1 文字以上の文字列
async 属性	必須項目。作成する要求のタイプを指定します。1 (真) のとき、要求は非同期になります。0 (偽) のとき、要求は同期します。	1 (真) または 0 (偽)
timeout 属性	オプション。同期呼び出しのタイムアウトを設定します。<call> の async 属性が 0 (偽) に設定されている場合のみ、timeout 値を使用します。応答を待機する秒数を、XML の xsd:dateTime 型の値を求める式として指定します。	1 文字以上の文字列 (“2003:10:19T10:10” など)
disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
<annotation> 要素		
<request> 要素	必須項目。送信する要求のタイプ (クラス名) を指定します。	
<response> 要素	オプション。返す応答のタイプ (クラス名) を指定します。指定しない場合は、この <call> から応答が返されません。	

説明

<call> 要素は、ビジネス・オペレーションまたはビジネス・プロセスに要求 (同期または非同期) を送信します。<call> 要素には、要求の作成方法を決定する必須の属性 async があります。

- async が 0 (偽) の場合、要求が同期として作成されます。つまり、応答を受け取った後で、ビジネス・プロセスの実行が再開されます。

重要　〈call〉要素に `async='False'` と〈response〉ブロックを定義することにより、呼び出した操作が完了するまで、ビジネス・プロセスのスレッドの実行を保留できます。

〈call〉要素に `async='False'` を定義し、〈response〉ブロックを定義していない場合の動作は、`async='True'` を定義した場合と同じになります。同期要求は送信する必要はあるが、応答は必要ないという場合は、機能しない〈response〉ブロックを作成します。これにより、ターゲット・ホストの完了を待って〈call〉が実行を継続するようにします。

- async が 1 (真) の場合、要求が非同期として作成されます。つまり、要求を送信した後で、ビジネス・プロセスの実行が再開されます。複数の非同期呼び出しからの応答を後で受け取るには、〈sync〉要素を使用して、応答を待機している〈call〉要素のリストを指定します。詳細は、このドキュメントの〈sync〉トピックを参照してください。

〈call〉要素には、〈request〉と〈response〉とい子要素があります。これらは、呼び出しの作成で使用する要求オブジェクトと応答オブジェクトのクラスを指定します。どちらの要素にも、1 つ以上の〈assign〉要素を含めることができます。〈request〉要素では、〈assign〉要素を使用して、その呼び出しに必要な要求オブジェクトのプロパティを指定します。〈response〉要素では、結果として返された応答オブジェクトのプロパティを、ビジネス・プロセスの実行コンテキストの context 変数や response 変数など、別の場所に移動する必要がある場合に〈assign〉要素を使用します。

注釈　ビジネス・プロセスの実行コンテキストの詳細は、〈assign〉要素の説明を参照してください。"BPL プロセスの開発" の "BPL プロセスについて" の章で "ビジネス・プロセスの実行コンテキスト" も参照してください。

非同期要求の場合、〈response〉要素の本体に含まれる〈assign〉要素は、対応する要求を受け取ったときに実行されます。これがいつ実行されるかはわかりません。そのためビジネス・プロセスでは、通常、〈sync〉要素を使用して非同期応答を待ちます。〈sync〉要素で指定されている timeout 時間内に応答を受信しなかった場合、対応する〈response〉ブロックで定義された割り当ては実行されません。応答自体は [破棄] ステータスとしてマーキングされます。

呼び出しが同期の場合は、〈call〉要素自体の timeout 属性を使用してオプションのタイムアウトを指定できます。この属性は、非同期呼び出しでは使用できません。〈call〉要素で async が 1 (真) に設定されている場合、タイムアウト時間を設定するには、非同期応答を収集する〈sync〉要素の timeout 属性を使用するしか方法がありません。

下の例は、同期 Ens.StringRequest リクエストを Get Weather Report ビジネス・オペレーションに送信します。

```
<call name='Get Weather Report' target='Get Weather Report' async='0' >
   <request type='Ens.StringRequest' >
     <assign property="callrequest.StringValue" value="context.Location" action="set" />
   </request>
   <response type='Demo.Service.Msg.WeatherOperationResponse' >
      <assign property="context.OperationReport" value="callresponse" action="set" />
   </response>
</call>
```

以下の例では、〈call〉要素を使用して、非同期の MyApp.SalaryRequest 要求を MyApp.PayrollApp ビジネス・オペレーションに送信します。

XML

```
<call name="FindSalary" target="MyApp.PayrollApp" async="1">
  <request type="MyApp.SalaryRequest">
    <assign property="callrequest.Name" value="request.Name" />
    <assign property="callrequest.SSN" value="request.SSN" />
  </request>
  <response type="MyApp.SalaryResponse">
    <assign property="context.Salary" value="callresponse.Salary" />
  </response>
</call>
```

〈call〉要素を実行するたびに、その〈call〉要素の name がメッセージ・ヘッダに挿入されるので、その後も [メッセージ・ブラウザ] や [ビジュアル・トレース] で参照できます。

<assign> 要素の使用

上記の例には、context、request、callrequest、callresponse など、ビジネス・プロセスの実行コンテキストの変数のプロパティを操作する <assign> 要素が含まれています。これらの変数については <assign> 要素のドキュメントで詳しく説明されていますが、以下の表では、<call> アクティビティに関連する実行コンテキスト変数について説明します。

<call> 要素は以下の変数とそのプロパティを参照できます。このリストに示されていない変数を使用しないでください。

変数	目的
callrequest	<call> 要素には、ターゲットに送信するメッセージのタイプを指定する <request> 要素が含まれます。このタイプのメッセージに入力パラメータがある場合、<request> 要素で <assign> 要素を使用し、callrequest オブジェクトのプロパティに値を割り当てる必要があります。これらのプロパティは、このメッセージ・タイプの入力パラメータと一致している必要があります。<request> が完了すると、callrequest オブジェクトはスコープから外れます。
callresponse	要求メッセージ・タイプに対応する応答メッセージ・タイプがある場合、<call> 要素には <response> 要素が含まれます。応答を受け取ると、<response> 要素に制御が渡されます。応答メッセージからの出力パラメータは、callresponse オブジェクトのプロパティとなります。callresponse は <response> 要素内でのみ有効です。したがって、これらの値を保持するには、<response> 要素内で <assign> 要素を使用して、ビジネス・プロセスの実行コンテキストのより永続的なオブジェクト（通常は context または response）のプロパティに、callresponse の値を割り当てる必要があります。
context	ビジネス・プロセスの実行中、context オブジェクトは、保持する必要のあるビジネス・プロセス・データの汎用コンテナとして機能します。
request	ビジネス・プロセスの実行中、request オブジェクトには、ビジネス・プロセスに送信された元のプロパティが、そのビジネス・プロセスをインスタンス化した要求のパラメータとして格納されます。
response	ビジネス・プロセスの実行中、response オブジェクトにはそのスコープが保持されます。このオブジェクトは、このビジネス・プロセスの出力パラメータとして呼び出し元に返されるべきプロパティを保持します。ビジネス・プロセスの完了時または終了時には、response オブジェクト内のプロパティがそのビジネス・プロセスの戻り値として解釈されます。
status	status は、成功または失敗を示す %Status 値です。BPL ビジネス・プロセスが開始されると、status には成功を示す値が自動的に割り当てられます。BPL ビジネス・プロセスの実行中に status が失敗値を受け取ると、そのビジネス・プロセスは即座に終了し、対応するテキスト・メッセージがイベント・ログに書き込まれます。status は、BPL コードに特別な文がなくても、<call> アクティビティから返された %Status の値を自動的に受け取ります。そのため、いずれかの <call> アクティビティが失敗した場合は、BPL ビジネス・プロセスが即座に終了して、イベント・ログ・エントリが書き込まれます。
syncresponses	syncresponses は応答オブジェクトのコレクションであり、同期している <call> アクティビティの名前がキーとなります。完了した呼び出しのみが示されます。syncresponses から応答を取得できるのは、<sync> を実行してから、現在の <sequence> の終了までの間に限られます。該当する呼び出しが `<call name="MyName">` として定義されている `syncresponses.GetAt("MyName")` 構文を使用して応答を取得します。

変数	目的
synctimedout	synctimedout 値は整数です。synctimedout は、いくつかの呼び出し後の <sync> アクティビティの結果を示します。synctimedout の値をテストできるのは、<sync> を実行してから、該当する呼び出しと <sync> を含む <sequence> の終了までです。synctimedout の値は、以下の 3 つのいずれかになります。 · 0 の場合、どの呼び出しもタイムアウトになっていません。すべての呼び出しが時間内に完了しました。<sync> アクティビティに timeout が設定されていない場合も、この値が返されます。 · 1 の場合、1 つ以上の呼び出しがタイムアウトになりました。つまり、時間切れのため、完了できなかった <call> アクティビティがあります。 · 2 の場合、1 つ以上の呼び出しが中断されました。 通常、synctimedout を取得してステータスを確認し、その後、syncresponses コレクションの完了した呼び出しから応答を取得します。

注意　その他すべての実行コンテキスト変数名と同様、status は BPL の予約語です。このテーブルで説明されている以外の用途では使用しないでください。

name 属性または target 属性の間接指定 (コンテキスト変数へのアクセス)

name 属性または target 属性の値は文字列です。name は呼び出しを識別し、その後、<sync> 要素で参照することができます。target は、要求の送信先であるビジネス・オペレーションまたはビジネス・プロセスの構成名です。どちらの文字列もリテラル値にすることができます。

```
<call name="Call" target="MyApp.MyOperation" async="1">
```

また、次のように @ 間接演算子を使用して、適切な文字列を含むコンテキスト変数の値にアクセスすることもできます。

```
<call name="@context.nextCallName" target="@context.nextBusinessHost" async="1">
```

ループで複数の非同期 <calls> を使用し、その後に 1 つの <sync> を指定

この節では、ループで複数の非同期 <calls> を使用し、その後に 1 つの <sync> を指定する方法を説明します。

BPL で <call> を実行すると、その呼び出しの名前が記録されます。待機対象とする保留中の要求を指定するために、その同じ名前を <sync> に指定する必要があります。シナリオによっては、次の例で示しているように、1 つのループ内に複数の非同期呼び出しを使用する場合があります。

```
<sequence>
  <while condition='...'>
      <call name="A" async="1" />
   </while>
  ...
  <sync calls="A" type="all" timeout="3600"/>
</sequence>
```

BPL では、呼び出し名で、待機対象の呼び出しが追跡されるため、最初の応答を受け取ってすぐに sync が実行されます。複数の非同期呼び出しすべてが完了するまで sync の実行を待機する場合は、一連の固有の呼び出し名を生成し、その名前リストを使用する必要があります。それを実行する方法は、次のとおりです。

1. 数値イテレータ (以下の例では `i`) を追加することによって、呼び出しごとに変化する文字列が含まれるコンテキスト変数を作成します。次の例のように、呼び出しの前にこの変数を初期化します。

   ```
   set context.callname = "A" _ context.i
   ```

2. <call> の `Name` をこの変数と同じに設定します。

3. <call> の名前すべてをカンマ区切りで含む文字列を作成します。次に例を示します。"A1,A2,A3,A4,A5"。これを別個の変数 (以下の例では context.allCallNames) に保存します。
4. <sync> の calls 属性を、呼び出しのリストを含む変数と同じに設定します。

```
<sequence>
  <while condition='...'>
      .... code here to set up callname and allCallNames ...
      <call name="@context.callname" async="1" />
   </while>
  ...
  <sync calls="@context.allCallNames" type="all" timeout="3600"/>
</sequence>
```

関連項目

<assign>、<code>、<reply>、<sequence>、<sync>

〈case〉

〈switch〉 要素内で条件が一致した場合に、一連のアクティビティを実行します。

構文

```
<switch>
   <case>
     ...
   </case>
    ...
   <default>
     ...
   </default>
</switch>
```

詳細

属性または要素	説明	値
condition 属性	必須項目。この式の結果が真の場合、この 〈case〉 要素の内容が実行されます。偽の場合、この 〈case〉 要素は無視されます。	整数値 1 (真の場合) または 0 (偽の場合) を求める式。この式では、この属性が含まれる 〈process〉 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
〈annotation〉 要素		
その他の要素	オプション。〈case〉 には、〈alert〉、〈assign〉、〈branch〉、〈break〉、〈call〉、〈code〉、〈continue〉、〈delay〉、〈empty〉、〈flow〉、〈foreach〉、〈if〉、〈label〉、〈milestone〉、〈reply〉、〈rule〉、〈scope〉、〈sequence〉、〈sql〉、〈switch〉、〈sync〉、〈throw〉、〈trace〉、〈transform〉、〈until〉、〈while〉、〈xpath〉 および 〈xslt〉 をゼロ個以上、自由に組み合わせて使用できます。	

説明

〈switch〉 要素には、連続する 1 つ以上の 〈case〉 要素と、オプションの 〈default〉 要素が含まれます。

〈switch〉 要素が実行されると、さらに各 〈case〉 条件が評価されます。これらの条件は、それを含む 〈process〉 要素のスクリプト言語で記述された論理式です。いずれかの式の結果が整数値 1 (真) の場合、それに対応する 〈case〉 要素の内容が実行されます。それ以外の場合は、次の 〈case〉 要素の式が評価されます。

どの 〈case〉 条件も真でない場合は、〈default〉 要素の内容が実行されます。

いずれかの 〈case〉 要素が実行されるとすぐに、対応する 〈switch〉 文から実行制御が離れます。どの 〈case〉 条件も一致しない場合は、〈default〉 アクティビティを実行した後で 〈switch〉 から制御が離れます。

〈case〉 要素内では、いずれかの BPL アクティビティを使用できます。例えば、以下の例では 〈assign〉 要素が使用されています。

XML

```
<switch name='Approved?'>
  <case name='No PrimeRate' condition='context.PrimeRate=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <case name='No Credit' condition='context.CreditRating=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <default name='Approved' >
    <assign name='Approved' property="response.IsApproved" value="1"/>
    <assign name='InterestRate'
            property="response.InterestRate"
            value="context.PrimeRate+10+(99*(1-(context.CreditRating/100)))">
      <annotation>
        Copy InterestRate into response object.


      </annotation>
    </assign>
  </default>
</switch>
```

〈catch〉

〈throw〉 要素によって生成されたフォールトをキャッチします。

構文

```
<scope>
   <throw fault='"MyFault"'/>
         ...
   <faulthandlers>
      <catch fault='"MyFault"'>
            ...
     </catch>
   </faulthandlers>
</scope>
```

詳細

属性または要素	説明	値
fault 属性	必須項目。フォールトの名前。この値には、リテラル文字列または評価対象となる式を指定できます。	0 ～ 255 文字の文字列。式の場合は、この属性が含まれる 〈process〉 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
〈annotation〉 要素		
その他の要素	オプション。〈catch〉 には、〈alert〉、〈assign〉、〈branch〉、〈break〉、〈call〉、〈code〉、〈compensate〉、〈continue〉、〈delay〉、〈empty〉、〈foreach〉、〈if〉、〈label〉、〈milestone〉、〈reply〉、〈rule〉、〈scope〉、〈sequence〉、〈sql〉、〈switch〉、〈sync〉、〈throw〉、〈trace〉、〈transform〉、〈until〉、〈while〉、〈xpath〉、および 〈xslt〉 をゼロ個以上、自由に組み合わせて使用できます。	

説明

〈throw〉 文が実行されると、同じ 〈scope〉 内にある 〈faulthandlers〉 ブロックに即座に制御が移り、〈throw〉 以降の途中の文はすべてスキップされます。次にプログラムは、〈faulthandlers〉 ブロック内で、〈throw〉 文の fault 文字列式と value 属性が同じである 〈catch〉 ブロックを探します。この比較では大文字と小文字が区別されます。フォールト文字列を指定するときは、以下のように、1 組の引用符を余分に追加する必要があります。

XML

```
<catch fault='"thrown"'/>
```

フォールトと一致する 〈catch〉 ブロックが見つかった場合は、その 〈catch〉 ブロック内のコードを実行して、〈scope〉 から抜けます。終わりの 〈/scope〉 要素の次の文から実行を再開します。

フォールトが発生し、フォールト文字列に一致する 〈catch〉 ブロックが 〈faulthandlers〉 ブロック内に存在しない場合は、〈throw〉 文から、〈faulthandlers〉 内の 〈catchall〉 ブロックに制御が移ります。〈catchall〉 ブロック内のコードを実行した後で、〈scope〉 から抜けます。終わりの 〈/scope〉 要素の次の文から実行を再開します。予期しないエラーを確実にキャッチするため、すべての 〈faulthandlers〉 ブロック内に 〈catchall〉 ブロックを配置することをお勧めします。

詳細は、"BPL プロセスの開発" の "BPL のエラー処理" を参照してください。

注釈　<catchall> を設ける場合は、<faulthandlers> ブロックの最後の文にする必要があります。必ず、<catchall> の前にすべての <catch> ブロックを配置します。

関連項目

<catchall>、<compensate>、<compensationhandlers>、<faulthandlers>、<scope>、<throw>

<catchall>

<catch> に合致しないフォールトやシステム・エラーをキャッチします。

構文

```
<scope>
   <throw fault='"MyFault"'/>
        ...
     <faulthandlers>
         <catch fault='"MyFault"'>
          ...
         </catch>
        <catch fault='"OtherFault"'>
           ...
        </catch>
        <catchall>
           ...
        </catchall>
     </faulthandlers>
</scope>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	
その他の要素	オプション。<catchall> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<compensate>、<continue>、<delay>、<empty>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath> および <xslt> をゼロ個以上、自由に組み合わせて使用できます。

説明

<throw> 文が実行されると、同じ <scope> 内にある <faulthandlers> ブロックに即座に制御が移り、<throw> 以降の途中の文はすべてスキップされます。次にプログラムは、<faulthandlers> ブロック内で、<throw> 文の fault 文字列式と value 属性が同じである <catch> ブロックを探します。見つかった場合は、この <catch> ブロック内のコードが実行され、その後で <scope> から抜けます。終わりの </scope> 要素の次の文から実行を再開します。

フォールトが発生し、フォールト文字列に一致する <catch> ブロックが <faulthandlers> ブロック内に存在しない場合は、<throw> 文から、<faulthandlers> 内の <catchall> ブロックに制御が移ります。<catchall> ブロック内のコードを実行した後で、<scope> から抜けます。終わりの </scope> 要素の次の文から実行を再開します。予期しないエラーを確実にキャッチするため、すべての <faulthandlers> ブロック内に <catchall> ブロックを配置することをお勧めします。

詳細は、"BPL プロセスの開発" の “BPL のエラー処理”を参照してください。

重要　　このエラー処理システムを、他のビジネス・ホストと通信する <call> 文と共に使用するときは、エラーの場合にターゲット・ビジネス・ホストがエラー・ステータスを返すことを確認します。エラーの場合でもターゲット・コンポーネントが成功を返すと、BPL プロセスは <catchhall> ロジックをトリガしません。

注釈　　<catchall> を設ける場合は、それを <faulthandlers> ブロックの最後の文にする必要があります。必ず、<catchall> の前にすべての <catch> ブロックを配置します。

関連項目

<catch>、<compensate>、<compensationhandlers>、<faulthandlers>、<scope>、<throw>

〈code〉

カスタム・コード行を実行します。

構文

```
<code name='CodeWrittenInBasic'>
   'invoke custom method "MyApp.MyClass".Method(context.Value)

</code>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
〈annotation〉 要素	

説明

BPL 〈code〉 要素は、BPL ビジネス・プロセス内の 1 行または複数行のユーザ記述コードを実行します。〈code〉 要素を使用すると、BPL 要素では表現が難しい特殊なタスクを実行できます。〈code〉 要素で参照するプロパティは、ビジネス・プロセスの実行コンテキスト内のプロパティである必要があります。

BPL 〈code〉 要素のスクリプト言語は、この要素を含む 〈process〉 要素の language 属性で指定されます。これは `objectscript` である必要があります。詳細は、以下を参照してください。

- ObjectScript の使用法
- ObjectScript リファレンス

通常、開発者は、アポストロフィー (') や アンパサンド (&) などの特殊な XML 文字をエスケープしなくてすむように、〈code〉 要素の内容を CDATA ブロック内にラップします。以下に例を示します。

XML

```
<code name="MyCode" language="objectscript">
   callrequest.Name = request.FirstName & " " & request.LastName

</code>
```

ビジネス・プロセスの実行を適切に中断およびリストアできるようにするために、〈code〉 要素の使用時には以下のガイドラインに従ってください。

- 実行時間は短くします。カスタム・コードがビジネス・プロセスの実行を妨げないようにしてください。
- 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、必ず同じ 〈code〉 要素内でそのリソースを解放してください。
- 〈code〉 要素がトランザクションを開始する場合は、同じ 〈code〉 要素が考えられるすべてのシナリオでトランザクションを終了することを確認します。トランザクションを終了していない場合には、トランザクションは無期限に開いたままになる可能性があります。これにより他の処理が阻止されたり、重大なダウンタイムを発生することがあります。
- ビジネス・プロセスの実行コンテキストに含まれていない変数には依存しないでください。InterSystems IRIS は、ビジネス・プロセスが中断され、後から再開された場合は常に、実行コンテキストの内容を自動的にリストアします。他の変数はクリーンアップされます。

また、<code> 内にコードの複数行を挿入するのではなく、クラス・メソッドや必要なコードを含むルーチンを起動することを強くお勧めします。この方法は、実行した処理のテストおよびデバッグをはるかに簡単にします。

使用可能な変数

<call> 要素は、以下の実行コンテキスト変数とそのプロパティを参照できます。このリストに示されていない変数を使用しないでください。

変数	目的
context	context オブジェクトは、ビジネス・プロセスの汎用データ・コンテナです。context は自動的には定義されません。このオブジェクトのプロパティを定義するには、<context> 要素を使用します。その場合、<process> 要素内でこれらのプロパティを参照するには、`context.Balance` のようにドット構文を使用します。
request	request オブジェクトには、このビジネス・プロセスをインスタンス化させた元の要求メッセージ・オブジェクトのプロパティが含まれます。<process> 要素内で request のプロパティを参照するには、`request.UserID` のようにドット構文を使用します。
response	response オブジェクトには、ビジネス・オブジェクトが返す最終的な応答メッセージ・オブジェクトの構築に必要となるすべてのプロパティが含まれます。<process> 要素内で response のプロパティを参照するには、`response.IsApproved` のようにドット構文を使用します。これらのプロパティに値を割り当てるには <assign> 要素を使用します。
status	status は、成功または失敗を示す %Status 型の値です。BPL ビジネス・プロセスが開始されると、status には成功を示す値が自動的に割り当てられます。BPL ビジネス・プロセスの実行中に status が失敗値を受け取ると、そのビジネス・プロセスは即座に終了し、該当するテキスト・メッセージがイベント・ログに書き込まれます。通常、<call> アクティビティから不成功値が返された時点で、この処理が自動的に行われます。ただし、<assign> または <code> を使用して status 値を設定すれば、BPL ビジネス・プロセス・コードをいつでも正常に終了できます。このトピックの最後の説明を参照してください。
process	process オブジェクトは、BPL ビジネス・プロセス・オブジェクトの現在のインスタンス (BPL クラスのインスタンス) を表します。このオブジェクトには、そのクラスで定義されたプロパティごとに 1 つのプロパティがあります。`process.SendRequestSync()` などの process オブジェクトのメソッドを起動できます。

注意　その他すべての実行コンテキスト変数名と同様、status は BPL の予約語です。<code> ブロックを終了させる場合以外は、<code> ブロック内で使用しないでください。

<code> を使用したステータス変数の設定

status は %Status 型のビジネス・プロセスの実行コンテキスト変数であり、成功または失敗を表します。

注釈　BPL ビジネス・プロセスのエラー処理は自動的に実行されます。BPL ソース・コードで status 値をテストまたは設定する必要はありません。ここでは、特殊な状況下で BPL ビジネス・プロセスを終了しなければならない場合に備えて、status 値が設けられています。

BPL ビジネス・プロセスが開始されると、status には成功を示す値が自動的に割り当てられます。status に成功値が割り当てられているかどうかを確認するには、ObjectScript ではマクロ `$$$ISOK(status)` を使用します。このテストで True 値が返された場合、status には成功値が格納されています。

BPL ビジネス・プロセスの実行中に status が失敗値を受け取ると、そのビジネス・プロセスは即座に終了し、該当するテキスト・メッセージがイベント・ログに書き込まれます。この処理は、status がどのような状況で失敗値を受け取ったかにかかわらず実行されます。したがって、status に失敗値を設定すれば、BPL ビジネス・プロセスをいつでも正常に終了することができます。

〈code〉 アクティビティ内の文では、status を失敗値に設定できます。〈code〉 アクティビティがすべて完了するまで、BPL ビジネス・プロセスは status 値の変更を認識しません。したがって、status が失敗の場合に 〈code〉 アクティビティを即座に終了するには、status に失敗値を設定した直後に、〈code〉 アクティビティ内に終了コマンドを配置する必要があります。

BPL ビジネス・プロセスでは、〈process〉 内であればどこでも status を使用できます。status を参照するための構文は、**%Status** 型の変数の場合と同じです。つまり `status` となります。

関連項目

〈call〉、〈sql〉

〈compensate〉

〈catch〉 または 〈catchall〉 から 〈compensationhandler〉 を呼び出します。

構文

```
<scope>
   <throw fault='"BuyersRegret"'/>
   <faulthandlers>
     <catch fault='"BuyersRegret"'>
       <compensate target="RestoreBalance"/>
     </catch>
   </faulthandlers>
   <compensationhandlers>
     <compensationhandler name="RestoreBalance">
       <assign property='context.MyBalance' value='context.MyBalance+1'/>
     </compensationhandler>
   </compensationhandlers>
</scope>
```

詳細

属性または要素	説明	値
target 属性	必須項目。以前のアクションを取り消す一連のアクティビティを表す 〈compensationhandler〉 の名前。	0 ～ 255 文字の文字列。
〈annotation〉 要素	“一般的な属性と要素” を参照してください。	

説明

〈compensate〉 要素から 〈compensationhandler〉 ブロックを呼び出すときは、ターゲットとしてその名前を指定します。

XML

```
<compensate target="general"/>
```

〈compensate〉 は、〈catch〉 または 〈catchall〉 内にのみ配置できます。その target 値は、同じ BPL ビジネス・プロセス内の 〈compensationhandler〉 の name と一致している必要があります。

詳細は、″BPL プロセスの開発″ の “BPL のエラー処理” を参照してください。

〈compensationhandlers〉

以前の操作を元に戻すための一連のアクティビティを実行する補償ハンドラです。

構文

```
<scope>
   <throw fault='"BuyersRegret"'/>
   <faulthandlers>
     <catch fault='"BuyersRegret"'>
       <compensate target="RestoreBalance"/>
     </catch>
   </faulthandlers>
   <compensationhandlers>
     <compensationhandler name="RestoreBalance">
       <assign property='context.MyBalance' value='context.MyBalance+1'/>
     </compensationhandler>
   </compensationhandlers>
 </scope>
```

要素

要素	目的
〈compensationhandler〉	ゼロ個以上の 〈compensationhandler〉 要素を 〈compensationhandlers〉 コンテナ内に配置できます。各 〈compensationhandler〉 要素には、以前の処理を元に戻すための一連の BPL アクティビティが含まれます。

説明

ビジネス・プロセス管理では、一部のロジック・セグメントを元に戻す必要が頻繁に生じます。この処理を“補償”といいます。原則として、ビジネス・プロセスが何かを実行する場合、その操作を取り消すことができなければなりません。つまり、エラーが発生した場合、ビジネス・プロセスは失敗した操作を元に戻すことによって、そのエラーを補償できる必要があります。障害発生時点以降のすべての操作を取り消し、問題が発生しなかった場合と同じ状態に戻す必要があります。BPL では、補償ハンドラと呼ばれるしくみによってこれを実現します。

BPL 〈compensationhandler〉 ブロックはサブルーチンに似ていますが、一般的なサブルーチン・メカニズムを備えていません。これらのブロックを“呼び出す”ことは可能ですが、必ず 〈faulthandler〉 ブロックから呼び出すこと、かつ、必ずその 〈compensationhandler〉 ブロックと同じ 〈scope〉 内から呼び出すことが条件となります。〈compensate〉 要素から 〈compensationhandler〉 ブロックを呼び出すときは、ターゲットとしてその名前を指定します。この構文では、引用符を追加する必要がありません。

XML

```
<compensate target="general"/>
```

補償ハンドラが役に立つのは、既に実行されている処理を元に戻せる場合だけです。例えば、預金を間違った口座に振り替えた場合、元の口座に振り替え直すことは可能ですが、元どおりの状態にできない処理もあります。したがって、適切な補償ハンドラを計画し、どの程度後退するのかに応じて、それらの補償ハンドラを構成する必要があります。

詳細は、″BPL プロセスの開発″ の“BPL のエラー処理”を参照してください。

注釈　〈compensationhandlers〉 と 〈faulthandlers〉 の順序を逆にすることはできません。これら両方のブロックを記述する場合は、最初に 〈compensationhandlers〉、次に 〈faulthandlers〉 を配置する必要があります。

<compensationhandler> 属性と要素

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	

関連項目

<catch>、<catchall>、<compensate>、<faulthandlers>、<scope>、<throw>

〈context〉

ビジネス・プロセスの実行コンテキストの汎用プロパティを定義します。

構文

```
<context>
    <property name="P1" type="%String" />
    <property name="P2" type="%String" />
    ...
</context>
```

要素

要素	目的
〈property〉	**オプション。ゼロ個以上の 〈property〉 要素を使用できます。それぞれの 〈property〉 要素は、ビジネス・プロセスの実行コンテキストの 1 つのプロパティを定義します。**

説明

ビジネス・プロセスのライフ・サイクルでは、ビジネス・プロセスが実行を中断または再開するたびに、一定のステータス情報をディスクに保存し、またディスクからリストアする必要があります。BPL ビジネス・プロセスでは、実行コンテキストと呼ばれる、変数グループを使用したビジネス・プロセスのライフ・サイクルをサポートしています。

実行コンテキストの変数には、context、request、response、callrequest、callresponse、process の各オブジェクト、整数値 synctimedout、コレクション syncresponses、および **%Status** 値の status があります。その目的は変数によって異なります。詳細は、〈assign〉 要素、〈call〉 要素、〈code〉 要素、および 〈sync〉 要素の説明を参照してください。

ほとんどの実行コンテキスト変数は、ビジネス・プロセスに対して自動的に定義されます。ただし、context と呼ばれる汎用コンテナ・オブジェクトだけは、BPL 開発者が定義する必要があります。維持する必要のある値、およびビジネス・プロセスのあらゆる場所で使用する必要のある値は、context オブジェクトのプロパティとして宣言します。そのためには、BPL ドキュメントの冒頭に 〈context〉 要素と 〈property〉 要素を配置します。以下に例を示します。ビジネス・プロセス・デザイナを使用する場合も、BPL ドキュメントにコードを直接入力する場合も、生成される BPL コードは同じです。

- ビジネス・プロセス・デザイナを使用する場合は、BPL ダイアグラムの右側にある **[コンテキスト]** タブからさまざまなタイプのプロパティを context オブジェクトに追加できます。**[コンテキスト・プロパティ]** の横にあるプラス記号をクリックすることによって、必要な任意のプロパティを追加します。プロパティ名の横にあるアイコンを使用して、プロパティを編集したり、削除したりすることもできます。生成されたビジネス・プロセスの BPL に、適切な 〈context〉 要素と 〈property〉 要素が配置されます。
- 以下の例に示すように、〈context〉 要素と 〈property〉 要素の両方を 〈process〉 要素の冒頭に追加できます。

XML

```
<process request="Demo.Loan.Msg.Application" response="Demo.Loan.Msg.Approval">
  <context>
    <property name="BankName" type="%String"
              initialexpression="BankOfMomAndDad" />
    <property name="IsApproved" type="%Boolean"/>
    <property name="InterestRate" type="%Numeric"/>
    <property name="TheResults"
              type="Demo.Loan.Msg.Approval"
              collection="list"/>
    <property name="Iterator" type="%String"/>
    <property name="ThisResult" type="Demo.Loan.Msg.Approval"/>
  </context>
  ...
</process>
```

各 <property> 要素は、プロパティの名前とデータ型を定義します。使用可能なデータ型クラスのリストは、"クラスの定義と使用" の "データ型" の章で "パラメータ" を参照してください。<property> 要素に初期値を割り当てるには、initialexpression 属性を指定します。また、<assign> 要素を使用すれば、ビジネス・プロセスの実行中に値を割り当てることができます。

〈continue〉

ループは終了せず、ループ内の次の繰り返しにジャンプします。

構文

```
<continue/>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
〈annotation〉 要素	

説明

BPL 構文では、必要に応じて、一連のアクティビティを含めることができるすべての要素 (〈case〉、〈default〉、〈foreach〉、〈false〉、〈sequence〉、〈true〉、〈until〉、〈while〉) に 〈continue〉 要素を含めることができます。

〈continue〉 要素を使用すれば、現在の繰り返し内の処理をすべて実行しなくても、ループの次の繰り返しに制御を進めることができます。以下に例を示します。

XML

```
<foreach property="P1" key="K1">

  //...do various things...

  <if condition="somecondition">
    <true>
      <continue/>
    </true>
  </if>

  //...do various other things...

</foreach>
```

上記の例では、〈continue〉 要素は 〈true〉 要素に含まれています。ただし、この 〈continue〉 要素が実際に影響を与えるループは、この 〈continue〉 を含む 〈foreach〉 ループです。

この例は、次のように機能します。このループにおいて、〈if〉 要素の “somecondition” が真になる (整数 1 に等しくなる) と、〈if〉 内の 〈true〉 要素に制御が渡されます。〈continue〉 要素に達すると、〈foreach〉 ループの現在の処理が停止し、(次の項目がある場合は) このコレクションの次の項目に進み、ループの先頭から次の項目が処理されます。

〈continue〉 要素を使用して変更するループ・アクティビティには、〈foreach〉、〈until〉、〈while〉 があります。ループ要素の各タイプで 〈continue〉 を使用すると、ループの現在の処理が停止し、ループの条件テストにジャンプします。そのテストとループのタイプによって次の動作が決まります。つまり、そのタイプのループが持つ通常の動作に応じて、ループを続けるか、ループを終了します。以下に例を示します。

〈continue〉を含むループ	〈continue〉の動作
〈foreach〉	コレクションの次の項目をテストします。項目が見つかると、その項目がループの最初から処理されます。テスト条件と一致する項目がコレクションにない場合は、ループを終了します。
〈until〉	ループの最後の条件テストにジャンプします。条件が真の場合はループを終了し、偽の場合はループの文を実行します。
〈while〉	ループの最初の条件テストにジャンプします。条件が真の場合はループを終了し、偽の場合はループの文を実行します。

関連項目

〈break〉

<default>

<switch> 要素内に一致する条件が見つからない場合に、一連のアクティビティを実行します。

構文

```
<switch>
   <case>
      ...
   </case>
   ...
   <default>
      ...
   </default>
</switch>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	"一般的な属性と要素" を参照してください。"name のデフォルト値は "Default" です。"
<annotation> 要素	
その他の要素	オプション。<default> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath> および <xslt> をゼロ個以上、自由に組み合わせて使用できます。

説明

<switch> 要素には、連続する 1 つ以上の <case> 要素と、オプションの <default> 要素が含まれます。

<default> 要素を使用する場合は、<switch> 内の最後の要素として記述する必要があります。同様に、ビジネス・プロセス・デザイナのダイアログラムでは、<switch> 部分の右端のオプションとして <default> 要素を追加する必要があります。

<switch> 要素が実行されると、さらに各 <case> 条件が評価されます。これらの条件は、それを含む <process> 要素のスクリプト言語で記述された論理式です。いずれかの式の結果が整数値 1 (真) の場合、それに対応する <case> 要素の内容が実行されます。それ以外の場合は、次の <case> 要素の式が評価されます。

どの <case> 条件も真でない場合は、<default> 要素の内容が実行されます。

<default> 要素内では、前述のいずれかの BPL アクティビティを使用できます。以下の例では <assign> 要素を使用しています。

XML

```
<switch name='Approved?'>
  <case name='No PrimeRate' condition='context.PrimeRate=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <case name='No Credit' condition='context.CreditRating=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <default name='Approved' >
    <assign name='Approved' property="response.IsApproved" value="1"/>
    <assign name='InterestRate'
            property="response.InterestRate"
            value="context.PrimeRate+10+(99*(1-(context.CreditRating/100)))">
      <annotation>
        Copy InterestRate into response object.


      </annotation>
    </assign>
  </default>
</switch>
```

<delay>

指定された期間、または指定された時間までビジネス・プロセスの実行を遅延します。

構文

```
<delay duration='"PT60S"'/>
```

または、以下のようになります。

```
<delay until='"2020-10-19T10:10"'/>
```

詳細

属性または要素	説明	値
duration 属性	オプション。XML duration 値を求める式として、遅延時間を指定します。*	1 文字以上の文字列。例えば、“PT60S” は 60 秒を表し、“P1Y2M3DT10H30M” は1 年 2 か月 3 日 10 時間 30 分を表します。<delay> 要素は秒の小数部を無視します。時間が 1 秒未満の値の場合、0 秒として処理されます。
until 属性	オプション。将来の時刻による遅延の期限を、XML dateTime 値を求める式で指定します。*	1 文字以上の文字列 (“2003:10:19T10:10” など)
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
<annotation> 要素		

* 詳細は、W3C 勧告 ″XML Schema Part 2: Datatypes Second Edition″ の ″Primitive Datatypes″ の該当する項目を、以下の URL から参照してください。

https://www.w3.org/TR/xmlschema-2/#duration

https://www.w3.org/TR/xmlschema-2/#dateTime

説明

<delay> 要素は、指定した時間または特定の時刻まで、ビジネス・プロセス (または <flow> 内の現在のスレッド) の実行を中断します。以下に例を示します。

XML

```
<sequence>
  <annotation>
     Write the time now, and sixty seconds later.


  </annotation>
  <trace value='"The time is: " & Now' />
  <delay duration='"PT60S"' />
  <trace value='"The time is: " & Now' />
</sequence>
```

<delay> 要素は、duration 属性で指定した時間、または until 属性で指定した時刻まで、ビジネス・プロセスの実行を中断します。duration 属性または until 属性を指定する必要があります。いずれも指定しない場合は実行が遅延されません。

遅延期間中は、現在のビジネス・プロセスのスレッドの実行が中断され、その時点のビジネス・プロセスの状態がデータベースに保存されます。

duration 値および until 値のフォーマットについては、XML データ型に関する W3C (World Wide Web Consortium) のドキュメントを参照してください。詳細は、W3C 勧告 "XML Schema Part 2: Datatypes Second Edition" の "Primitive Datatypes" (https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes) を参照してください。以下に duration の例を示します。

- PT60S または PT1M (1分)
- PT219S または PT3M39S (3 分 39 秒)

<delay> 要素を実行するたびに、その <delay> 要素の name がメッセージ・ヘッダに挿入されるので、その後も [メッセージ・ブラウザ] や [ビジュアル・トレース] で参照できます。

<empty>

アクションを何も実行しません。

構文

```
<empty />
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	

説明

<empty> 要素は、操作を何も実行しません。この要素の目的は、BPL 定義内のプレースホルダとして機能すること、またはビジネス・プロセスの実行に影響を与えずに、追加の注釈を保持する場所として機能することです。例えば、以下のようになります。

XML

```
<empty>
  <annotation>This is an empty element.
  </annotation>
</empty>
```

<false>

<if> 要素の条件が偽の場合に、一連のアクティビティを実行します。

構文

```
<if condition="0">
   <true>
     ...
   </true>
   <false>
     ...
   </false>
</if>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	オプション。要素について記述するテキスト文字列。
その他の要素	オプション。<false> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath> および <xslt> をゼロ個以上、自由に組み合わせて使用できます。

説明

<false> 要素は、<if> 内で、条件が偽の場合に実行する必要がある要素を収容するのに使用されます。

<faulthandlers>

フォールトおよびシステム・エラーをキャッチする、ゼロ個以上の <catch> と 1 つの <catchall> 要素が含まれます。

構文

```
<scope>
   <throw fault='"MyFault"'/>
         ...
     <faulthandlers>
        <catch fault='"MyFault"'>
           ...
        </catch>
        <catch fault='"OtherFault"'>
           ...
        </catch>
        <catchall>
           ...
        </catchall>
   </faulthandlers>
</scope>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<catch> 要素	<faulthandlers> 内にはゼロ個以上の <catch> 要素を配置できます。それぞれ、<throw> 要素によって生成された特定のフォールトをキャッチします。
<catchall> 要素	<catch> に合致しないフォールトやシステム・エラーをキャッチします。<faulthandlers> 内に <catch> 要素がない場合は、<catchall> を含める必要があります。それ以外の場合、<catchall> はオプションです。

説明

エラー処理に対応するため、BPL には <scope> と呼ばれる要素があります。スコープは一連のアクティビティのラッパです。このスコープには、1 つ以上のアクティビティ、1 つ以上のフォールト・ハンドラ、ゼロ個以上の補償ハンドラを含めることができます。<faulthandlers> 要素の目的は、<scope> 内のアクティビティが生成するエラーをキャッチすることです。<faulthandlers> 内の <catch> 要素と <catchall> 要素では、<compensate> 文を使用して、キャッチしたエラーを補償する <compensationhandler> 要素を呼び出すことができます。

<scope> 内に <faulthandlers> ブロックがない場合は、システム・エラーがイベント・ログに自動的に出力されます。<scope> に <faulthandlers> ブロックが含まれている場合は、<trace> メッセージをイベント・ログに出力し、システム・エラー・メッセージがイベント・ログに表示されるようにする必要があります。いずれの場合も、システム・エラー・メッセージがターミナル・コンソールに表示されます。

詳細は、"BPL プロセスの開発" の “BPL のエラー処理”を参照してください。

注釈　<compensationhandlers> と <faulthandlers> の順序を逆にすることはできません。これら両方のブロックを記述する場合は、最初に <compensationhandlers>、次に <faulthandlers> を配置する必要があります。

関連項目

<catch>、<catchall>、<compensate>、<compensationhandlers>、<scope>、<throw>

<flow>

不定の順序でアクティビティを実行します。

構文

```
<flow>
   <sequence name="thread1">
     ...
   </sequence>
   <sequence name="thread2">
     ...
   </sequence>
   ...
</flow>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	
<sequence> 要素	オプション。ゼロ個以上の <sequence> 要素内に、<flow> に必要なすべてのアクティビティを記述します。<sequence> 要素が指定されていない場合、<flow> によって実行されるアクションはありません。

説明

<flow> 要素は、その中に含まれる各要素が不定の順序で実行されることを指定します。<flow> 要素には 1 つまたは複数の <sequence> 要素が含まれます。これらの各 <sequence> 要素はスレッドと呼ばれます。

生成される BPL コードを調べるとわかるように、ビジネス・プロセス・デザイナを使用してビジネス・プロセスに <flow> 要素を追加すると、<sequence> 要素が <flow> 内に自動的に挿入されます。

<flow> 内のいずれかの <sequence> 要素を一時的に無効にする必要がある場合は、生成された BPL コードを編集して、対応する <sequence> 要素の disabled 属性を設定します。

以下のコードは <flow> 要素の使用例を示しています。この手動でコード化された BPL 例では、<flow> 内に 2 つの <sequence> が同レベルで配置されています。それぞれ、`thread1` と `thread2` という別々のスレッドで実行されます。

XML

```
<process>
  <flow>
    <sequence name="thread1">
      <call name="A" />
      <call name="B" />
    </sequence>
    <sequence name="thread2">
      <call name="C" />
      <call name="A" />
    </sequence>
  </flow>
  <call name="E" />
</process>
```

この例では、<flow> 要素内に 2 つのスレッドが定義されており、それぞれ <sequence> 要素 `thread1` と `thread2` として指定されています。これら 2 つのスレッドが実行される順序は決まっていません (ただし、<sequence> 要素の内側にある <call> 要素は、この順序で実行されます)。

可能な場合、スレッドの実行は飛び越されます。例えば、あるスレッドの実行が中断された場合（非同期呼び出しからの応答を待機している状態など）は、可能な場合は他のスレッドのいずれかが実行されます。

厳密には、<flow> 要素内の各スレッドは同時に実行されません。これは、並行性とデータの整合性を正しく維持するために、ビジネス・プロセスの実行コンテキストにアクセスできるスレッドを一度に 1 つのみとしているからです。

注釈　ビジネス・プロセス実行コンテキストの詳細は、このドキュメントの <assign> 要素と、"BPL プロセスの開発" を参照してください。

<flow> 要素は、内側にあるすべてのスレッドが完了するのを待ってから、実行を再開します。上の例では、両方のスレッドが完了した時点で実行が再開され、<call> 要素 E が実行されます。

<flow> 要素内のスレッドには、さらにネストされた <flow> 要素を含めることもできます。

<flow> 要素での <sync> の使用方法については、<sync> 要素の説明を参照してください。

<foreach>

繰り返し実行される一連のアクティビティを定義します。

構文

```
<foreach property="P1" key="K1">
   ...
</foreach>
```

詳細

属性または要素	説明	値
property 属性	必須項目。繰り返しが行われるコレクション・プロパティ (リストまたは配列)。これは、実行コンテキスト内の有効なオブジェクト名およびプロパティ名である必要があります。	1 文字以上の文字列。
key 属性	必須項目。コレクション内で繰り返しを実行するために使用するインデックス。これは、実行コンテキスト内の有効なオブジェクト名およびプロパティ名である必要があります。コレクション内の要素ごとに値が割り当てられます。	1 文字以上の文字列。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		
その他の要素	オプション。<foreach> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath> および <xslt> をゼロ個以上、自由に組み合わせて使用できます。	

説明

<foreach> 要素では、繰り返し実行される一連のアクティビティを定義します。これらのアクティビティは、指定したコレクション・プロパティ内の各要素について 1 回ずつ実行されます。以下に例を示します。

XML

```
<foreach property="callrequest.Location" key="context.K1">
  <assign property="total"
          value="context.total+context.prices.GetAt(context.K1)"/>
</foreach>
```

<foreach> 要素は以下の変数とそのプロパティを参照できます。このリストに示されていない変数を使用しないでください。

変数	目的
context	context オブジェクトは、ビジネス・プロセスの汎用データ・コンテナです。context は自動的には定義されません。このオブジェクトのプロパティを定義するには、<context> 要素を使用します。その場合、<process> 要素内でこれらのプロパティを参照するには、`context.Balance` のようにドット構文を使用します。

変数	目的
request	request オブジェクトには、このビジネス・プロセスをインスタンス化させた元の要求メッセージ・オブジェクトのプロパティが含まれます。<process> 要素内で request のプロパティを参照するには、`request.UserID` のようにドット構文を使用します。
response	response オブジェクトには、ビジネス・オブジェクトが返す最終的な応答メッセージ・オブジェクトの構築に必要となるすべてのプロパティが含まれます。`response.IsApproved` と同様に、ドット構文を使用して、<process> 要素内部の response プロパティを参照できます。これらのプロパティに値を割り当てるには <assign> 要素を使用します。

注釈　　ビジネス・プロセスの実行コンテキストの詳細は、<assign> 要素の説明を参照してください。

ループの実行を微調整するには、<foreach> 要素内で <break> 要素と <continue> 要素を使用します。詳細は、各要素の説明を参照してください。

<if>

条件を評価し、真の場合の操作または偽の場合の操作を実行します。

構文

```
<if condition="1">
   <true>
     ...
   </true>
   <false>
     ...
   </false>
</if>
```

詳細

属性または要素	説明	値
condition 属性	必須項目。この条件式が真の場合、<true> 要素内のコードが実行されます。偽の場合は、<false> 要素内のコードが実行されます。	整数値 1（真の場合）または 0（偽の場合）を求める式。この式では、この属性が含まれる <process> 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		
<true> 要素	オプション。条件が真の場合、<true> 要素内のアクティビティが実行されます。	
<false> 要素	オプション。条件が偽の場合、<false> 要素内のアクティビティが実行されます。	

説明

<if> 要素は式を評価し、その値に応じて、2 つのアクティビティ・セット（式の結果が真の場合のアクティビティと偽の場合のアクティビティ）のいずれかを実行します。

<if> 要素には、<true> 要素と <false> 要素を含めることができます。これらの要素は、式の結果が真の場合および偽の場合に実行されるアクションをそれぞれ定義します。

<true> 要素と <false> 要素を両方指定する場合、<if> 要素ではどちらを先に配置してもかまいません。

条件が真で <true> 要素がない場合、または条件が偽で <false> 要素がない場合、その <if> 要素では何も実行されません。

次の例では、<if> 要素を使用して、<call> 要素と <sync> 要素を組み合わせた結果を調整しています。

XML

```
<sequence name="thread1">
  <call name="A" />
  <call name="B"  />
  <sync calls="A,B" type="all" timeout="10" />
  // Did the synchronization time out before it finished?
  <if condition='synctimedout="1"'>
    <true>
      <trace value="thread1 timeout: Call A or B did not return." />
    </true>
    // If not, then the calls came back, so assign the results.
      <false>
```

```
        <assign property="context.TheResultsFromEast"
                value='syncresponses.GetAt("A")'
                action="append"/>
        <assign property="context.TheResultsFromWest"
                value='syncresponses.GetAt("B")'
                action="append"/>
      </false>
    </if>
  </sequence>
```

この例の <if> アクティビティでは、実行コンテキスト変数 synctimedout が整数値 1 であるかどうかをテストする条件が指定されています。<call> に関するドキュメントで説明しているように、synctimedout の値は、0、1、または 2 となります。2 つの値が等しい場合、この <if> 条件は整数値 1 を受け取り、<true> 要素内の文が実行されます。それ以外の場合は、<false> 要素内の文が実行されます。

注釈　　ビジネス・プロセスの実行コンテキストの詳細は、<assign> 要素の説明を参照してください。

<label>

条件分岐操作の分岐先を指定します。

構文

```
<label name="JumpToMe" />
```

詳細

属性または要素	説明	値
name 属性	必須項目。このラベルの名前。この名前は、BPL ビジネス・プロセス全体で一意である必要があります。	0 ～ 255 文字の文字列。
disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		

説明

<label> 要素では、条件付き <branch> 要素の分岐先を指定します。

詳細は、<branch> の説明を参照してください。

〈milestone〉

ビジネス・プロセスが達成したステップを認知するためのメッセージを格納します。

構文

```
<milestone value='"The applicant has been notified of the interest rate."' />
```

詳細

属性または要素	説明	値
value 属性	必須項目。マイルストーン・メッセージのテキストです。この値には、リテラル文字列または評価対象となる式を指定できます。	1 文字以上の文字列。式またはリテラル文字列を使用できます。式の場合は、この属性が含まれる 〈process〉 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
〈annotation〉 要素		

説明

〈milestone〉 アクティビティは InterSystems IRIS データベースにメッセージを書き込みます。〈milestone〉 の機能は 〈trace〉 とよく似ていますが、〈trace〉 メッセージと異なる点として、〈milestone〉 メッセージは、関連するビジネス・プロセスが実行されている間だけ保持されます。ビジネス・プロセスが終了した後は、〈milestone〉 アクティビティによって生成されたすべてのメッセージが削除されます。

プログラマは、通常、診断目的で 〈trace〉 メッセージを使用します。一方、〈milestone〉 メッセージは、長時間にわたるビジネス・プロセスが正常に動作している際の進捗状況追跡に役立ちます。

〈milestone〉 メッセージを取得するには、Ens.Milestone グローバルの値を調べます。このグローバルは、プロダクションが 〈milestone〉 メッセージを発行した場合のみ定義されます。Ens.Milestone の値を取得する手順は次のとおりです。

- ″InterSystems IRIS 多次元ストレージの使用法″ の “多次元ストレージの使用法 (グローバル)” の章に従って、プログラムから取得します。
- 管理ポータルから、**[システムエクスプローラ]→[グローバル]** ページに移動し、**[ネームスペース]** オプションが選択されていることを確認して、プロダクションを実行するネームスペースの名前をクリックします。デフォルトでは、**[グローバル表示]** オプションが選択されます。

〈parameters〉

他の BPL 要素のパラメータを、名前と値のペアとして指定します。

構文

```
<parameters>
    <parameter name='MAXLEN' value='1024' />
    <parameter name='MINLEN' value='1' />
</parameters>
```

要素

要素	目的
〈parameter〉	ゼロ個以上の 〈parameter〉 要素を 〈parameters〉 コンテナ内に配置できます。各 〈parameter〉 要素は 1 つのパラメータを定義します。以下で説明するように、〈parameter〉 要素にはそれぞれ 2 つの属性 (name と value) があります。

説明

オプションの 〈parameters〉 要素は 〈property〉 または 〈xslt〉 内でのみ有効です。〈parameters〉 では、その内部の BPL 要素のパラメータが名前と値のペアとして定義されます。

- 〈context〉 内の場合、〈parameters〉 には、ビジネス・プロセスの実行コンテキストで定義する 〈property〉 のデータ型パラメータが含まれます。ビジネス・プロセスの実行コンテキストの詳細は、〈assign〉 要素の説明を参照してください。
- 〈xslt〉 内の場合、〈parameters〉 には、XSLT 変換を制御するスタイルシートに渡す XSLT 名前と値のペアが含まれます。

〈parameters〉 は BPL 属性をサポートしていません。〈parameters〉 は、ゼロ個以上の 〈parameter〉 要素を格納する単なるコンテナであり、パラメータごとに 1 つ使用します。〈parameter〉 要素は必要な数だけ使用できますが、すべて同じ 〈parameters〉 ブロック内に配置する必要があります。以下に例を示します。

XML

```
<context>
  <property name='Test' type='%Integer' initialexpression='342' >
    <parameters>
      <parameter name='MAXVAL' value='1000' />
    </parameters>
  </property>
  <property name='Another' type='%String' initialexpression='Yo' >
    <parameters>
      <parameter name='MAXLEN' value='2' />
      <parameter name='MINLEN' value='1' />
    </parameters>
  </property>
</context>
```

〈parameter〉の属性

属性	説明	値
name 属性	必須項目。このパラメータの名前。 ・ 〈property〉内では、name は対象プロパティのデータ型パラメータを指定します。有効な名前は、"クラスの定義と使用" の "データ型" の章で "パラメータ" を参照してください。 ・ 〈xslt〉内では、name には、有効な XSLT パラメータの名前を指定する必要があります。	1 文字以上の文字列。
value 属性	オプション。パラメータに割り当てる値。	1 文字以上の文字列。

関連項目

〈context〉、〈xslt〉

〈process〉

ビジネス・プロセスを定義します。

構文

```
<process request="MyApp.Request" response="MyApp.Response">
    <context>
     ...
    </context>
    <sequence>
     ...
    </sequence>
</process>
```

詳細

属性または要素	説明	値
request 属性	必須項目。このビジネス・プロセスに対する最初の要求のタイプを指定する、要求クラスの名前。	1 文字以上の文字列。
response 属性	オプション。このビジネス・プロセスから応答が返される場合にそのタイプを指定する、応答クラスの名前。	1 文字以上の文字列。
component 属性	オプション。この値を 1 (真) に設定すると、この 〈process〉 が再利用可能コンポーネントと見なされます。	1 (真) または 0 (偽) のブーリアン値。デフォルトは偽です。
contextsuperclass 属性	オプション。ビジネス・プロセス・コンテキストのスーパークラスを指定できます。異なる多数のビジネス・プロセスが同じコンテキスト変数を共有している場合に役立ちます。つまり、Ens.BP.Context のサブクラスを自分で生成し、独自のプロパティを追加した後、contextsuperclass でそのクラスを使用します。指定しない場合は、Ens.BP.Context がデフォルトとなります。	クラス名。
height 属性	オプション。ビジネス・プロセス・デザイナでのビジネス・プロセスのグラフィカルな表現に関連します。	正の整数。
includes 属性	オプション。 インクルード・ファイル名のカンマ区切りリスト。〈code〉 セグメントでマクロを使用できるようになります。	
language 属性	オプション。"objectscript" である必要があります。	"objectscript"
layout 属性	オプション。このビジネス・プロセスの BPL ダイアグラムで使用されるレイアウト・スタイルの名前。値 “automatic” を指定すると、ビジネス・プロセス・デザイナと BPL ビューワで自動的にダイアグラム要素のレイアウトが選択されます。値 “manual” を指定すると、指定したレイアウトがこれらのツールで使用されるようになります。	“manual” または “automatic” のいずれかの文字列。指定がない場合のデフォルト・レイアウトは、“automatic” です。
version 属性	オプション。バージョン番号を表す整数。値が大きいほど、新しいバージョンであることを示しています。式の場合、version 属性値では、ObjectScript を使用する必要があります。	正の整数。リテラル整数、または結果が整数になる式を指定できます。
width 属性	オプション。ビジネス・プロセス・デザイナでのビジネス・プロセスのグラフィカルな表現に関連します。	正の整数。

属性または要素	説明	値
<context> 要素	オプション。ビジネス・プロセスの実行コンテキストの汎用プロパティを定義します。ビジネス・プロセス実行コンテキストの詳細は、このドキュメントの <assign> 要素と、"BPL プロセスの開発" を参照してください。	
<sequence> 要素	オプション。ゼロ個以上の <sequence> 要素を使用できます。各要素で、ビジネス・プロセスが実行できるアクションを定義します。	

説明

<process> 要素は、BPL ドキュメントの最も外側の要素です。その他のすべての BPL 要素は、<process> 要素の中に含まれます。

ビジネス・プロセスは、実行コンテキスト (<context> 要素で定義) および一連のアクティビティ (<sequence> 要素で定義) で構成されます。

request 属性は、ビジネス・プロセスの最初の要求のタイプ (クラス名) を定義します。**response** 属性は、ビジネス・プロセスからの最終的な応答のタイプ (クラス名) を定義します。**request** 属性は必須項目ですが、**response** 属性はオプションです。これは、ビジネス・プロセスが応答を返さない場合があるためです。

実行コンテキスト

ビジネス・プロセスのライフ・サイクルでは、ビジネス・プロセスが実行を中断または再開するたびに、一定のステータス情報をディスクに保存し、またディスクからリストアする必要があります。BPL ビジネス・プロセスでは、実行コンテキストと呼ばれる、変数グループを使用したビジネス・プロセスのライフ・サイクルをサポートしています。

実行コンテキストの変数には、context、request、response、callrequest、callresponse、process の各オブジェクト、整数値 synctimedout、コレクション syncresponses、および **%Status** 値の status があります。その目的は変数によって異なります。詳細は、<assign> 要素、<call> 要素、<code> 要素、および <sync> 要素の説明を参照してください。

例

以下のサンプル・ビジネス・プロセスでは、<sync> 要素を使用して複数の <call> 要素を同期しています。このサンプルの最後では、<process> 要素内のアクティビティが省略記号 (...) で示されています。

XML

```
<process request="Demo.Loan.Msg.Application">
<context>
  <property name="BankName" type="%String"/>
  <property name="IsApproved" type="%Boolean"/>
  <property name="InterestRate" type="%Numeric"/>
  <property name="TheResults" type="Demo.Loan.Msg.Approval" collection="list"/>
  <property name="Iterator" type="%String"/>
  <property name="ThisResult" type="Demo.Loan.Msg.Approval"/>
</context>
<sequence>
  <trace value='"received application for "_request.Name'/>
  <call name="BankUS" target="Demo.Loan.BankUS" async="1">
    <annotation>
      Send an asynchronous request to Bank US.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.TheResults"
              value="callresponse"
              action="append"/>
    </response>
  </call>
```

```
  <call name="BankSoprano" target="Demo.Loan.BankSoprano" async="1">
    <annotation>
      Send an asynchronous request to Bank Soprano.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.TheResults"
              value="callresponse"
              action="append"/>
    </response>
  </call>

  <call name="BankManana" target="Demo.Loan.BankManana" async="1">
    <annotation>
      Send an asynchronous request to Bank Manana.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.TheResults"
              value="callresponse"
              action="append"/>
    </response>
  </call>

  <sync name='Wait for Banks'
        calls="BankUS,BankSoprano,BankManana"
        type="all"
        timeout="5">
    <annotation>
      Wait for responses. Wait up to 5 seconds.


    </annotation>
  </sync>
  <trace value='"sync complete"'/>
  ...
</sequence>
</process>
```

応答

ビジネス・プロセスからの基本応答は、そのビジネス・プロセス・インスタンスを最初に呼び出した要求に返す応答です。通常、ビジネス・プロセスを実行した時点で、その基本応答が自動的に返されます。ただし、〈reply〉 要素を使用すれば、基本応答をさらに早く返すことができます。これは、呼び出し側が要求している応答をすぐに返すことができるものの、その呼び出しの結果として、ビジネス・プロセスで実行しなければならない追加処理がある場合に役立ちます。

言語

〈process〉 要素では、language 属性に指定した値で、ビジネス・プロセスで使用するスクリプト言語を定義します。値は `"objectscript"` である必要があります。ビジネス・プロセス内のすべて式、および 〈code〉 要素内のコード行は、指定した言語で記述されている必要があります。

バージョン管理

開発者は、BPL ビジネス・プロセスの version 番号を更新し、その新機能が以前のバージョンと互換性がないことを通知できます。新しいバージョンになるほど、番号が大きくなります。BPL ビジネス・プロセスには自動バージョン機能が備わっていません。開発者は、BPL 〈process〉 要素内の version 属性の値を手動で更新し、新しいコードには、同じビジネス・プロセスの以前のバージョンと互換性のない変更が含まれていることを示します。例えば、ビジネス・プロセス 〈context〉 内のプロパティを追加または削除したり、ビジネス・プロセス 〈sequence〉 内のアクティビティ・フローを変更した場合などに、バージョン番号を更新します。

インスタンスが既に実行されている以前のバージョンの BPL ビジネス・プロセスは、当初のコンテキストで、当初のアクティビティを引き続き実行します。最新バージョンは、さらに別のコンテキストを使用して、別のアクティビティを実行します。そのため、InterSystems IRIS では、バージョンごとに新しいコンテキストとスレッド・クラスが生成されます。新しいバー

ジョンは、生成されたクラス階層のサブパッケージとして表示されます。例えば、クラス MyBPL がある場合、バージョン 3 によって MyBPL.V3.Context と MyBPL.V3.Thread1 が生成されます。

レイアウト

ビジネス・プロセス・デザイナで BPL ダイアグラムを開くと、デフォルトでは、自動レイアウト機能を使用してダイアグラムが表示されます。自動で選択されたレイアウトは、必ずしも個々の図に適しているとは限りません。作成するダイアグラムで自動選択が問題になる可能性がある場合は、ダイアグラムが必ず希望通りのレイアウトで正確に表示されるように、自動レイアウト機能を無効にできます。

ダイアグラムのレイアウトを最も直接的に制御する方法としては、**[プリファレンス]** タブの **[自動整列]** チェックボックスのチェックを外します。

[一般] タブをクリックして、レイアウトに対して [自動] または [手動] を選択することもできます。“手動”を選択した場合は、ダイアグラムを保存するたびに各要素の位置がそのまま保存されます。したがって、ダイアグラムをビジネス・プロセス・デザイナで表示したときには、自分で指定したレイアウト特性以外は適用されません。

ビジネス・プロセス・デザイナでのビジネス・プロセス・ダイアグラムのスクロールに関連する問題は、〈process〉 要素の **height** 属性または **width** 属性を調整することで解決できます。この操作は、**layout** 属性の場合と同じように [一般] タブで行えます。

<property>

ビジネス・プロセスの <context> 要素内にプロパティを定義します。

構文

```
<property name='Test' type='%Integer' initialexpression='342' >
   <parameters>
      <parameter name='MAXVAL' value='1000' />
   </parameters>
</property>
```

詳細

属性または要素	説明	値
name 属性	必須項目。このプロパティの名前。有効なプロパティ名である必要があります。	1 文字以上の文字列。
type 属性	オプション。このプロパティのタイプを指定するクラスの名前。データ型クラス (%String)、シリアル・クラス、または永続クラスを指定できます。	クラスの名前を表す 1 文字以上の文字列。
initialexpression 属性	オプション。この ObjectScript 式は、プロパティのデフォルト値を提供するために評価されます。	プロパティに有効な値を指定する式。以下の説明を参照してください。
instantiate属性	オプション。プロパティの “作成” フラグとして機能します。指定しない場合は、デフォルトの 0 (作成しない) が使用されます。	1 (作成する) または 0 (作成しない)
collection 属性	オプション。このプロパティが特定のタイプのコレクションであることを指定します。	次のいずれかのリテラル値。“list”、“array”、“binarystream” または “characterstream”。
<parameters>	オプションの <parameters> 要素を使用できます。<parameters> コンテナ内には、ゼロ個以上の <parameter> 要素を配置できます。各 <parameter> 要素では、パラメータの name と value を指定することによって、そのプロパティのデータ型パラメータを 1 つだけ定義します。有効な名前と値は、"クラスの定義と使用" の “データ型” の章で “パラメータ” の節を参照してください。	

説明

<property> 要素は、ビジネス・プロセスの実行コンテキスト内のプロパティを定義します。

ビジネス・プロセスのライフ・サイクルでは、ビジネス・プロセスが実行を中断または再開するたびに、一定のステータス情報をディスクに保存し、またディスクからリストアする必要があります。BPL ビジネス・プロセスでは、実行コンテキストと呼ばれる、変数グループを使用したビジネス・プロセスのライフ・サイクルをサポートしています。

実行コンテキストの変数には、context、request、response、callrequest、callresponse、process の各オブジェクト、整数値 synctimedout、コレクション syncresponses、および %Status 値の status があります。その目的は変数によって異なります。詳細は、<assign> 要素、<call> 要素、<code> 要素、および <sync> 要素の説明を参照してください。

ほとんどの実行コンテキスト変数は、ビジネス・プロセスに対して自動的に定義されます。ただし、context と呼ばれる汎用コンテナ・オブジェクトだけは、BPL 開発者が定義する必要があります。維持する必要のある値、およびビジネス・プロ

セスのあらゆる場所で使用する必要のある値は、context オブジェクトのプロパティとして宣言します。そのためには、BPL ドキュメントの冒頭に <context> 要素と <property> 要素を配置します。各 <property> 要素では、context オブジェクトのプロパティを 1 つ定義します。

<property> 要素には **name** を指定する必要があります。

非コレクション・プロパティの場合は、**initialexpression** 属性と **instantiate** 属性で、オブジェクトの初期化方法を指定します。**instantiate** 属性の値が整数 1 (真) である場合に “new” を呼び出すと、オブジェクトが生成されます。**initialexpression** 属性も指定されている場合は、この式の結果がオブジェクトに割り当てられます。

instantiate 属性は、インスタンス化可能なプロパティを初期化するときに使用します。一方、**initialexpression** 属性は、%String などのデータ型クラスを初期化するときに使用します。 値が文字列の場合、その文字列の引用符を別の引用符で囲んでください。つまり、`"hello"` という初期文字列値を設定する場合は、`initialexpression='"hello"'` と指定します。

collection 属性 (“list”、“array”、“binarystream”、または “characterstream”) を設定すると、プロパティが自動的にそのタイプのコレクションとしてインスタンス化されます。

以下の例は、ビジネス・プロセスの冒頭にある <context> 要素内に記述された一連の <property> 要素を示しています。

XML

```
<process request="Demo.Loan.Msg.Application" response="Demo.Loan.Msg.Approval">
  <context>
    <property name="BankName" type="%String"
              initialexpression="BankOfMomAndDad" />
    <property name="IsApproved" type="%Boolean"/>
    <property name="InterestRate" type="%Numeric"/>
    <property name="TheResults"
              type="Demo.Loan.Msg.Approval"
              collection="list"/>
    <property name="Iterator" type="%String"/>
    <property name="ThisResult" type="Demo.Loan.Msg.Approval"/>
  </context>
  ...
</process>
```

各 <property> 要素は、プロパティの名前とデータ型を定義します。使用可能なデータ型クラスのリストは、″クラスの定義と使用″ の “データ型” の章で “パラメータ” を参照してください。<property> で初期値を割り当てるには、initialexpression 属性を指定します。また、<assign> 要素を使用すれば、ビジネス・プロセスの実行中に値を割り当てることができます。

関連項目

<parameters>

<reply>

ビジネス・プロセスからの応答を、ビジネス・プロセスの実行が完了する前に送信します。

構文

```
<reply/>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。
<annotation> 要素	

説明

ビジネス・プロセスからの “基本応答” は、そのビジネス・プロセス・インスタンスを最初に呼び出した要求に返す応答です。通常、ビジネス・プロセスを実行した時点で、その基本応答が自動的に返されます。ただし、<reply> 要素を使用すれば、基本応答をさらに早く返すことができます。これは、呼び出し側が要求している応答をすぐに返すことができるものの、その呼び出しの結果として、ビジネス・プロセスで実行しなければならない追加処理がある場合に役立ちます。

<reply> 要素は、ビジネス・プロセスの実行コンテキストから response オブジェクトを返します。したがって、ビジネス・プロセスで <reply> を実行する前に、<assign> 要素を使用して、response オブジェクトのプロパティに値を割り当てる必要があります。

注釈　　ビジネス・プロセスの実行コンテキストの詳細は、<assign> 要素の説明を参照してください。

<request>

<call> 要素内で要求を準備します。

構文

```
<call name="Call" target="MyApp.MyOperation" async="1">
   <request type="MyApp.Request">
     ...
   </request>
   <response type="MyApp.Response">
     ...
   </response>
</call>
```

詳細

属性または要素	説明	値
type 属性	必須項目。要求メッセージ・クラスの名前。	1 文字以上の文字列。
name 属性	オプション。<request> 要素の名前。	0 ～ 255 文字の文字列。
その他の要素	オプション。<request> には、<assign>、<empty>、<milestone>、および <trace> をゼロ個以上、自由に組み合わせて使用できます。	

説明

<request> 要素は、<call> に必須の子要素です。<call> コンテキスト内で、<request> 要素は、送信する要求のタイプ (クラス名) を指定します。<request> 要素には、1 つ以上の <assign> 要素を含めることもできます。これの各要素は、ビジネス・プロセスの実行コンテキストのオブジェクトで使用するプロパティに値を割り当てます。以下に例を示します。

XML

```
<call name="FindSalary" target="MyApp.PayrollApp" async="1">
  <request type="MyApp.SalaryRequest">
    <assign property="callrequest.Name" value="request.Name" />
    <assign property="callrequest.SSN" value="request.SSN" />
  </request>
  <response type="MyApp.SalaryResponse">
    <assign property="context.Salary" value="callresponse.Salary" />
  </response>
</call>
```

通常、<request> 要素内の <assign> 要素は、callrequest オブジェクトのプロパティに値を割り当てるために使用します。このオブジェクトは、ビジネス・プロセスの実行コンテキストのメンバであり、呼び出しに使用される要求オブジェクトのプロパティを収容するコンテナとして機能します。ただし、context オブジェクト、request オブジェクト、response オブジェクトのプロパティも、<request> 内の <assign> 要素で設定や追加などを行えます。

ビジネス・プロセスの実行コンテキストの詳細は、<call> および <assign> の説明を参照してください。

関連項目

<process>、<reply>

〈response〉

〈call〉 要素内で受け取った応答を処理します。

構文

```
<call name="Call" target="MyApp.MyOperation" async="1">
   <request type="MyApp.Request">
     ...
   </request>
   <response type="MyApp.Response">
     ...
   </response>
</call>
```

詳細

属性または要素	説明	値
type 属性	必須項目。応答メッセージ・クラスの名前。	1 文字以上の文字列。
name 属性	オプション。〈response〉 要素の名前。	0 ～ 255 文字の文字列。
その他の要素	オプション。〈response〉 には、〈assign〉、〈empty〉、〈milestone〉、および 〈trace〉 をゼロ個以上、自由に組み合わせて使用できます。	

説明

〈response〉 要素は、〈call〉 のオプションの子要素です。〈call〉 コンテキスト内で、〈response〉 要素は、呼び出しから返される応答のタイプ (クラス名) を指定します。〈response〉 要素には、1 つ以上の 〈assign〉 要素を含めることもできます。以下に例を示します。

XML

```
<call name="FindSalary" target="MyApp.PayrollApp" async="1">
  <request type="MyApp.SalaryRequest">
    <assign property="callrequest.Name" value="request.Name" />
    <assign property="callrequest.SSN" value="request.SSN" />
  </request>
  <response type="MyApp.SalaryResponse">
    <assign property="context.Salary" value="callresponse.Salary" />
  </response>
</call>
```

呼び出し元のビジネス・プロセスに応答が返されると、〈response〉 要素で指定されたメッセージ・タイプからの出力パラメータが、ビジネス・プロセスの実行コンテキスト内の callresponse オブジェクトのプロパティになります。callresponse は 〈response〉 要素内でのみ有効です。したがって、これらの値を保持するには、〈response〉 要素内で 〈assign〉 要素を使用して、ビジネス・プロセスの実行コンテキストのより永続的なオブジェクト (通常は context または response) のプロパティに、callresponse の値を割り当てる必要があります。

詳細は、〈call〉 および 〈assign〉 の説明を参照してください。

〈request〉 要素はすべての 〈call〉 内に配置する必要がありますが、〈response〉 は必須ではありません。〈call〉 要素内で 〈response〉 要素を使用しない場合は、〈request〉 タイプが応答を返すように指定されている場合でも、その 〈call〉 からは応答が返されません。〈request〉 が非同期の場合、〈response〉 要素の本体に含まれる 〈assign〉 要素は、呼び出しの応答を受け取った後にのみ実行されます。これがいつ実行されるかはわかりません。そのためビジネス・プロセスでは、通常、〈sync〉 要素を使用して非同期応答を待ちます。

〈sync〉 要素で指定されたタイムアウト時間内に応答がなかった場合、対応する 〈response〉 ブロックで定義された割り当ては実行されません。応答自体には、"Discarded" ステータスが設定されます。

関連項目

〈process〉、〈reply〉

<rule>

プロダクション・ビジネス・ルール・クラスを参照してください。

構文

```
<rule name="ApproveLoan"
        rule="LoanApproval"
        resultLocation="context.Answer"
        reasonLocation="context.Reason">
</rule>
```

詳細

属性または要素	説明	値
name 属性	必須項目。<rule> 要素の名前。	1 文字以上の文字列。
rule 属性	必須項目。実行されるビジネス・ルールの名前。ネームスペース内で有効なルールを指定する必要があります。この後の“ルールの特定”を参照してください。ルールが定義されていない場合、または実行時に見つからない場合は、そのルールからデフォルト値 "" (空の文字列) が返されます。	1 文字以上の文字列。
ruleContext 属性	オプション。定義すると、ルール・エンジンに渡されるオブジェクトを特定する式となります。後述の“コンテキストの特定”を参照してください。以下に例を示します。 `context.MyObject` デフォルトでは、このルールは、ルール・エンジンにビジネス・プロセスの実行コンテキストを渡します。	1 文字以上の文字列。
resultLocation 属性	オプション。ルールの戻り値を格納する場所。通常、これはビジネス・プロセスの実行コンテキスト内のプロパティ、つまり context.MyValue です。	通常はビジネス・プロセスの実行コンテキスト内で有効なプロパティとオブジェクトの名前。
reasonLocation 属性	オプション。ルールから返された理由を格納する場所。ルール理由は、ビジネス・ルールがその決定を行った理由を示す文字列です。例えば、[ルール 1] や [デフォルト] などです。ビジネス・ルールが空である場合 (ルールを含まないルール・セットである場合など) は、決定に対して [ルールがありません] という理由が設定されます。	ゼロ文字以上の文字列。
disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		

説明

<rule> 要素は、ビジネス・プロセスからビジネス・ルールを呼び出します。<rule> が実行されると、関連するビジネス・ルール (rule 属性で指定) が呼び出され、その応答が直ちに返されます (<code> や <assign> アクティビティと同じ方法)。

ルールの特定

BPL で <rule> 要素を使用する場合、rule 属性の値は以下のいずれかになります。

- 単純な**ルール名**

  ```
  MyRule
  ```

- 完全な**パッケージ名**と**ルール名**の組み合わせ

  ```
  MyClassPackage.Organization.Levels.MyRule
  ```

<rule> 要素で単純な**ルール名**を指定した場合は、その先頭に**パッケージ名**が自動的に追加されます。パッケージ名は、その <rule> 要素を含む BPL ビジネス・プロセスの完全なパッケージ名とクラス名に相当します。つまり、以下のようになります。

BPLFullPackageAndClassName`.MyRule`

この組み合わせは、ネームスペース内の有効なルールを表している必要があります。有効でない場合は、<rule> から NULL 文字列が返されます。

コンテキストの特定

デフォルトでは、ルールに渡される ruleContext は、ビジネス・プロセスの実行コンテキストです。別のオブジェクトをコンテキストとして指定する場合、そのオブジェクトに対していくつかの制約が発生します。まず、Ens.BusinessProcess タイプの **%Process** というプロパティが必要です。これは、ルール・エンジンにビジネス・プロセスの呼び出しコンテキストを渡すために使用します。このプロパティに値を設定する必要はありませんが、存在することは必要です。次に、目的のオブジェクトが、ルールそのもので想定されているオブジェクトと一致している必要があります。これらの制約が守られていることを確認するためのチェックは行われません。開発者側で、オブジェクトを正しく設定することが必要です。

簡単な例

以下は、<rule> アクティビティを <switch> 要素と共に使用して、ルールからの結果を処理する BPL の例を示しています。

```
<sequence>
  <rule name="ExecuteRule"
        rule="MyRule"
        resultLocation="context.MyResult" />
  <switch>
    <case condition="context.MyResult=1">
      <!-- ...Rule is true... -->
    </case>
    <default>
      <!-- ... Rule is false... -->
    </default>
  </switch>
</sequence>
```

この例の <rule> アクティビティは、InterSystems IRIS の規則に従ってブーリアン値 (真または偽) を返します。つまり、整数値 1 は真、整数値 0 は偽を意味します。この例にあるように、すべてのルールは単一の値を返しますが、値の型がブーリアンである必要はありません。ルールから返される単一の値には、整数、小数、またはテキスト文字列などの任意のリテラル値を使用できます。

戻り値

結果および結果の理由は、それぞれ resultLocation 属性と reasonLocation 属性で指定した変数に格納されます。通常、これらの属性では、context 変数のプロパティ名を指定します。これは永続的な汎用変数で、<context> 要素と <property> 要素を使用して BPL ビジネス・プロセスの最初で定義します。

関連項目

ビジネス・ルールの開発

<sequence>

アクティビティを順次実行します。

構文

```
<sequence>
   ...
</sequence>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	
その他の要素	オプション。<sequence> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath>、および <xslt> をゼロ個以上、自由に組み合わせて使用できます。

説明

<process> 内または <flow> 内で <sequence> 要素を使用して、順次実行する必要のある要素を追加します。

<sequence> と <process>

各 BPL ドキュメントの <process> 要素内で少なくとも 1 つの <sequence> 要素を使用して、そのビジネス・プロセスのアクティビティのメイン・シーケンスを指定する必要があります。以下に例を示します。

XML

```
<process>
  <sequence>
    <call name="A" />
    <call name="B" />
  </sequence>
</process>
```

ビジネス・プロセス・デザイナで、上位レベルの新規の BPL ダイアグラムの <start> 要素と <end> 要素の間にアクティビティを追加すると、追加したアクティビティはすべて、BPL コード・ジェネレータが生成コードの <process> 要素内部に配置した単一の最上位レベルの <sequence> に含められます。上の例は、このような <sequence> を示しています。<call> 要素 A が最初に実行され、その後、<call> 要素 B が実行されます。

ビジネス・プロセス・デザイナの使用時に <process> 内の最上位レベルの <sequence> を一時的に無効にする必要がある場合は、スタジオでこの操作を行えます。生成された BPL コードで、対応する <sequence> 要素に disabled 属性を追加することによって、この要素を無効に設定できます。

ネストされた <sequence> 要素

<sequence> には他のシーケンスを含めることができます。ネストされた <sequence> 要素では追加の実行スレッドが開始されません。実行スレッドを開始するには、<flow> 要素が必要になります。ただし、<sequence> 要素を余分に追加してネストすることで、BPL ドキュメント内で項目をグループ化できます。以下に例を示します。

XML

```
<process>
  <sequence>
    <sequence>
      <call name="A" />
      <call name="B" />
    </sequence>
  </sequence>
</process>
```

ネストされた〈sequence〉要素は、ビジネス・プロセス用に生成されたコードに影響しません。しかし、BPL ダイアグラムには、このようなネストされたシーケンス全体が 1 つの〈sequence〉アイコンとして表示されます。この〈sequence〉アイコンをドリル・ダウンして、中に含まれている要素を表示できます。

〈sequence〉と〈flow〉

ビジネス・プロセス・デザイナを使用してビジネス・プロセスに〈flow〉要素を追加すると、〈flow〉内に〈sequence〉要素が自動的に挿入されます。生成された BPL コードを調べると、〈sequence〉要素が挿入されていることがわかります。このフローに新しい〈sequence〉要素を追加できます。その際、〈flow〉の各分岐を、それぞれの〈sequence〉要素で囲む必要があります。

〈flow〉内のいずれかの〈sequence〉要素を一時的に無効にする必要がある場合は、スタジオでこの操作を行えます。生成された BPL コードで、対応する〈sequence〉要素に disabled 属性を追加して値を true に設定します。

〈scope〉

一連のアクティビティのエラー処理手段を定義します。

構文

```
<scope>
   <throw fault='"MyFault"'/>
          ...
   <faulthandlers>
     <catch fault='"MyFault"'>
           ...
     </catch>
   </faulthandlers>
</scope>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
〈annotation〉 要素	
その他の要素	オプション。〈scope〉には、〈alert〉、〈assign〉、〈branch〉、〈break〉、〈call〉、〈code〉、〈continue〉、〈delay〉、〈empty〉、〈flow〉、〈foreach〉、〈if〉、〈label〉、〈milestone〉、〈reply〉、〈rule〉、〈scope〉、〈sequence〉、〈sql〉、〈switch〉、〈sync〉、〈throw〉、〈trace〉、〈transform〉、〈until〉、〈while〉、〈xpath〉、および 〈xslt〉 をゼロ個以上、自由に組み合わせて使用できます。

説明

エラー処理に対応するため、BPL には〈scope〉と呼ばれる要素があります。スコープは一連のアクティビティのラッパです。このスコープには、1 つ以上のアクティビティ、1 つ以上のフォールト・ハンドラ、ゼロ個以上の補償ハンドラを含めることができます。フォールト・ハンドラの目的は、〈scope〉内のアクティビティによって生成されるすべてのエラーをキャッチすることです。また、補償ハンドラを呼び出して、発生したエラーに対処することもできます。

以下の例では、〈scope〉内に〈faulthandlers〉ブロックを配置し、さらに〈catchall〉を追加しています。

Class Member

```
XData BPL
{
<process language='objectscript'
         request='Test.Scope.Request'
         response='Test.Scope.Response' >
  <sequence>
    <trace value='"before scope"'/>
    <scope>
      <trace value='"before assign"'/>
      <assign property="SomeProperty" value="1/0"/>
      <trace value='"after assign"'/>
      <faulthandlers>
        <catchall>
          <trace value='"in catchall faulthandler"'/>
          <trace value=
            '"%LastError "_
            $System.Status.GetErrorCodes(..%Context.%LastError)_
            " : "_
            $System.Status.GetOneStatusText(..%Context.%LastError)'
            />
        </catchall>
      </faulthandlers>
    </scope>
    <trace value='"after scope"'/>
```

```
  </sequence>
</process>
}
```

〈scope〉内に〈faulthandlers〉ブロックがない場合は、システム・エラーがイベント・ログに自動的に出力されます。〈scope〉に〈faulthandlers〉ブロックが含まれている場合は、〈trace〉メッセージをイベント・ログに出力し、システム・エラー・メッセージがイベント・ログに表示されるようにする必要があります。いずれの場合も、システム・エラー・メッセージがターミナル・コンソールに表示されます。

〈scope〉要素をネスト構造にできます。内側のスコープでエラーやフォールトが発生した場合、内側のスコープ内でキャッチする方法と、内側のスコープはそのエラーを無視し、外側スコープの〈faulthandlers〉ブロックでキャッチする方法があります。

詳細は、"BPL プロセスの開発" の "BPL のエラー処理" を参照してください。

関連項目

〈catch〉、〈catchall〉、〈compensate〉、〈compensationhandlers〉、〈faulthandlers〉、〈throw〉

<sql>

埋め込まれた SQL SELECT 文を実行します。

構文

```
<sql name="LookUp">

     SELECT SSN INTO :context.SSN
     FROM MyApp.PatientTable
     WHERE PatID = :request.PatID


</sql>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。
<annotation> 要素	

説明

<sql> 要素は、ビジネス・プロセスの実行中に、任意の埋め込み SQL SELECT 文を実行します。

<sql> 要素は、テーブルを使用した検索操作の実行に特に有効です。例えば、ビジネス・プロセスに送信された基本要求に、患者 ID 番号を示す PatId プロパティがあり、ビジネス・プロセスが作業を実行する前に、患者 ID 番号と一致する社会保障番号 (SSN) を検索する必要があるとします。PatId を SSN に関連付ける PatientTable テーブルが使用可能であれば、以下のように <sql> 要素を使用して、この検索を実行できます。

XML

```
<process>
  <sql name="LookUp">
    SELECT SSN INTO :context.SSN
    FROM MyApp.PatientTable
    WHERE PatID = :request.PatID


  </sql>
</process>
```

上記の例では、実行コンテキスト変数 context に、SQL のクエリ結果を取得できる SSN プロパティがあります。実行コンテキスト変数 request には、PatId プロパティが自動的に追加されます。この変数には、基本要求オブジェクトで取得したプロパティが必ず格納されるためです。

注釈　ビジネス・プロセス実行コンテキストの詳細は、このドキュメントの <assign> 要素と、"BPL プロセスの開発" を参照してください。

InterSystems IRIS データベース内に PatientTable のローカル・コピーを保持している場合、上記の例は、コストの高いネットワーク操作や追加のミドルウェアをまったく使用せずに実行できるため、特に効率的です。

<sql> 要素を効率的に使用するため、以下のヒントを参考にしてください。

- 常に、SQL スキーマ名とテーブル名の両方で構成された以下のような完全なテーブル名を使用します。

  `MyApp.PatientTable`

  上記の例の `MyApp` は SQL スキーマ名、`PatientTable` はテーブル名です。

- 〈sql〉 要素の内容には、有効な埋め込み SQL SELECT 文が含まれている必要があります。

  SQL クエリを CDATA ブロック内に配置すれば、特殊な XML 文字のエスケープに影響されることはありません。

- SQL クエリの FROM 節にリストされるテーブルは、ローカルの InterSystems IRIS データベースに格納されているか、SQL ゲートウェイを使用して外部リレーショナル・データベースにリンクされている必要があります。

- SQL クエリの INTO 節および WHERE 節内では、ビジネス・プロセスの実行コンテキスト内の 1 つの変数名の前に、“:”を付けることにより、そのプロパティを参照できます。以下に例を示します。

  XML

```
<sql name="LookUp">
  SELECT Name INTO :response.Name
  FROM MainFrame.EmployeeRecord
  WHERE SSN = :request.SSN AND City = :request.Home.City


</sql>
```

- 使用されるのは、クエリで返された最初の行のみです。WHERE 節では、必要な行を正確に指定してください。

<switch>

一連の条件を評価し、実行する操作を決定します。

構文

```
<switch>
   <case>
     ...
   </case>
   ...
   <default>
     ...
   </default>
</switch>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
<annotation> 要素	
<case> 要素	必須 (少なくとも 1 つは必要)。各 <case> 要素は、真またはそれ以外になる条件を定義します。
<default> 要素	オプション。どの <case> 条件も満たされない場合に実行する操作を指定します。指定する場合は、<switch> 要素の最後に配置する必要があります。

説明

<switch> 要素には、連続する 1 つ以上の <case> 要素と、オプションの <default> 要素が含まれます。

<switch> 要素が実行されると、さらに各 <case> 条件が評価されます。これらの条件は、それを含む <process> 要素のスクリプト言語で記述された論理式です。いずれかの式の結果が整数値 1 (真) の場合、それに対応する <case> 要素の内容が実行されます。それ以外の場合は、次の <case> 要素の式が評価されます。

どの <case> 条件も真でない場合は、<default> 要素の内容が実行されます。

いずれかの <case> 要素が実行されるとすぐに、対応する <switch> 文から実行制御が離れます。どの <case> 条件も一致しない場合は、<default> アクティビティを実行した後で <switch> から制御が離れます。

どの <case> も真ではなく、<default> も指定されていない場合、その <switch> 文では何も実行されません。

<case> 要素内では、いずれかの BPL アクティビティを使用できます。例えば、以下の例では <assign> 要素が使用されています。

XML

```
<switch name='Approved?'>
  <case name='No PrimeRate' condition='context.PrimeRate=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <case name='No Credit' condition='context.CreditRating=""'>
    <assign name='Not Approved' property="response.IsApproved" value="0"/>
  </case>
  <default name='Approved' >
    <assign name='Approved' property="response.IsApproved" value="1"/>
    <assign name='InterestRate'
            property="response.InterestRate"
            value="context.PrimeRate+10+(99*(1-(context.CreditRating/100)))">
      <annotation>
        Copy InterestRate into response object.


      </annotation>
    </assign>
  </default>
</switch>
```

<sync>

1 つ以上の非同期要求からの応答を待ちます。

構文

```
<sequence>
   <call name="A" async="1" />
   <call name="B" async="1" />
   ...
   <sync calls="A,B" type="all" timeout="3600"/>
</sequence>
```

詳細

属性または要素	説明	値
calls 属性	必須項目。<sync> が待機する 1 つ以上の非同期 <call> 要素の名前のリストです。	<call> 要素名のカンマ区切りリスト。この値は、リテラル文字列として指定するか、または @ 間接演算子でコンテキスト変数の値を参照することによって指定できます。この後の説明を参照してください。
allowresync 属性	オプション。真の場合、<sync> 要素は繰り返しポーリングを実行し、非同期呼び出しの完了を検出します。つまり、同じ呼び出しに対して <sync> を繰り返し実行することができます。この機能は、呼び出しが完了するまでの時間が明確でない場合に役立ちます。デフォルトの allowresync 値は偽です。	1 (真) または 0 (偽) のブーリアン値。
timeout 属性	オプション。応答を待機する秒数を、XML の xsd:dateTime 型の値を求める式として指定します。	1 文字以上の文字列 (“2003:10:19T10:10” など)。
type 属性	オプション。	文字列 “all” (デフォルト) または “any”。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
<annotation> 要素		

説明

多くのビジネス・プロセスは、外部システムに対して 1 つ以上の要求を発行します。外部システムの応答が遅い場合や、外部システムを利用できない場合などに対応するため、通常、これらの要求は非同期に発行されます。<sync> 要素を使用すると、1 つ以上の非同期呼び出しからの応答を待つ処理が簡単になります。この要素は、<call> 要素と共に使用します。

<sync> 要素の動作は、calls 属性、timeout 属性、および type 属性で指定します。type 属性には、値 “all” (すべての呼び出しからの応答を待機) または “any” (最初に受け取る応答のみを待機) を指定できます。“any” の場合、タイムアウト時間を過ぎた場合と同じように、残りの応答は破棄されます。

例えば、以下の BPL では、2 つの非同期要求 A と B を実行し、さらに、<sync> 要素を使用してその応答を待ちます (最長 1 時間)。

XML

```
<sequence>
  <call name="A" async="1" />
  <call name="B" async="1" />
  <sync calls="A,B" type="all" timeout="3600" />
</sequence>
```

タイムアウト時間後に受信した応答には "Discarded" ステータスが設定され、ビジネス・プロセスでは処理されません。〈sync〉要素でタイムアウト値を指定しない場合は、経過時間にかかわらず、すべての応答を受け取るまで待機し続けます。ただし、〈sync〉要素が待機している間はビジネス・プロセスがディスクに保存され、それを実行していたジョブが解放されるので、他のビジネス・プロセスを処理できます。

以下の BPL 例では、〈process〉が 2 つの呼び出しを発行し、5 秒間待機します。

XML

```
<process request="Demo.Loan.Msg.Application">
  <context>
    <property name="BankName" type="%String"/>
    <property name="IsApproved" type="%Boolean"/>
    <property name="InterestRate" type="%Numeric"/>
    <property name="Results" type="Demo.Loan.Msg.Approval" collection="list"/>
    <property name="Iterator" type="%String"/>
    <property name="ThisResult" type="Demo.Loan.Msg.Approval"/>
  </context>
  <sequence>
    <trace value='"received application for "_request.Name'/>
    <call name="BankUS" target="Demo.Loan.BankUS" async="1">
    <annotation>
      Send an asynchronous request to Bank US.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.Results"
              value="callresponse"
              action="append"/>
    </response>
  </call>

  <call name="BankSoprano" target="Demo.Loan.BankSoprano" async="1">
    <annotation>
      Send an asynchronous request to Bank Soprano.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.Results"
              value="callresponse"
              action="append"/>
    </response>
  </call>

  <call name="BankManana" target="Demo.Loan.BankManana" async="1">
    <annotation>
      Send an asynchronous request to Bank Manana.


    </annotation>
    <request type="Demo.Loan.Msg.Application">
      <assign property="callrequest" value="request"/>
    </request>
    <response type="Demo.Loan.Msg.Approval">
      <assign property="context.Results"
              value="callresponse"
              action="append"/>
    </response>
  </call>

  <sync name='Wait for Banks'
        calls="BankUS,BankSoprano,BankManana"
        type="all"
        timeout="5">
```

```
    <annotation>
      Wait for responses from the banks. Wait up to 5 seconds.


    </annotation>
  </sync>
  <trace value='"sync complete"'/>
  </sequence>
</process>
```

<sync> 要素を実行するたびに、その <sync> 要素の name がメッセージ・ヘッダに挿入されるので、その後も [メッセージ・ブラウザ] や [ビジュアル・トレース] で参照できます。

<call> 要素の一意名

別の <call> 要素と同じ name で <call> 要素を定義しようとすると、BPL エディタにエラー・メッセージが表示されます。この場合、一意の名前を指定する必要があります。

calls 属性の間接指定

calls 属性の値は文字列です。この文字列は、<call> 要素名のカンマ区切りリストとして指定する必要があります。以下のように、この文字列はリテラル値でもかまいません。

```
calls="BankUS,BankSoprano,BankManana"
```

また、次のように @ 間接演算子を使用して、適切な文字列を含むコンテキスト変数の値にアクセスすることもできます。

```
calls="@context.myListOfCalls"
```

syncresponses

<call> 要素の結果として受け取る応答を処理するための、さらに別のしくみが用意されています。

このしくみでは、<sync> 要素を使用すると、受信したさまざまな応答がコレクションに追加されます。このコレクションはビジネス・プロセスの実行コンテキストの変数であり、syncresponses と呼ばれます。実行コンテキストには、synctimedout という整数の変数もあります。2 つの変数 synctimedout と syncresponses は、次のように連動して機能します。

オブジェクト	説明
syncresponses	syncresponses は応答オブジェクトのコレクションであり、同期している <call> アクティビティの名前がキーとなります。完了した呼び出しのみが示されます。syncresponses から応答を取得できるのは、<sync> を実行してから、現在の <sequence> の終了までの間に限られます。該当する呼び出しが <call name="MyName"> として定義されている syncresponses.GetAt("MyName") 構文を使用して応答を取得します。
synctimedout	synctimedout 値は整数です。synctimedout は、いくつかの呼び出し後の <sync> アクティビティの結果を示します。synctimedout の値をテストできるのは、<sync> の後から、該当する呼び出しと <sync> を含む <sequence> の直前までです。synctimedout の値は、以下の 3 つのいずれかになります。 ・ 0 の場合、どの呼び出しもタイムアウトになっていません。すべての呼び出しが時間内に完了しました。<sync> アクティビティに timeout が設定されていない場合も、この値が返されます。 ・ 1 の場合、1 つ以上の呼び出しがタイムアウトになりました。つまり、時間切れのため、完了できなかった <call> アクティビティがあります。 ・ 2 の場合、1 つ以上の呼び出しが中断されました。 通常、synctimedout を取得してステータスを確認し、その後、syncresponses コレクションの完了した呼び出しから応答を取得します。

〈sync〉アクティビティを実行するとすぐに、新しい応答に備えて syncresponses コレクションが消去されます。呼び出しから制御が戻ると、それらの応答が syncresponses コレクションに追加されます。〈sync〉アクティビティが完了した時点で、待機していた応答の一部または全部が syncresponses に格納されます。

例えば、以下の構文を使用して、Call1 と Call2 に〈sync〉を実行した場合を考えてみます。

```
<sync type="all" timeout="60">
```

ここで、Call1 は 60 秒以内に返り、Call2 は 60 秒経過後も返らないものと仮定します。この時点の syncresponses には、Call1 に対する応答のみが含まれ、Call2 への応答は含まれません。synctimedout の値をテストして、該当する値が syncresponses に格納されているかどうかを確認できます。

〈sync〉アクティビティを実行した後、返された応答にアクセスするには、〈call〉アクティビティの名前をキーとして使用します。例えば、呼び出しが以下のように定義されているとします。

```
<call name="nameOfCall">
```

この場合、次の構文を使用して、応答にアクセスします。

```
syncresponses.GetAt("nameOfCall")
```

以下の〈sequence〉要素を実行するとします。

XML

```
<sequence>
  <call name="A" async="1" />
  <call name="B" async="1" />
  <call name="C" async="1" />
  <sync calls="A,B,C" type="all" />
</sequence>
```

〈sync〉要素が完了すると、syncresponses コレクションには、以下のように 3 つの応答オブジェクトへの参照が追加されます。

- `syncresponses.GetAt("A")` = A からの応答（存在する場合）
- `syncresponses.GetAt("B")` = B からの応答（存在する場合）
- `syncresponses.GetAt("C")` = C からの応答（存在する場合）

応答がない場合、syncresponses コレクションは空になります。

注釈　ビジネス・プロセス実行コンテキストの詳細は、このドキュメントの〈assign〉要素と、"BPL プロセスの開発" を参照してください。

マルチ・スレッドの syncresponses

〈sync〉要素を〈flow〉と共に使用する場合、〈process〉自体を実行するプライマリ・スレッドを含め、スレッドごとに別個の syncresponses コレクションが作成されます。そのため、ビジネス・プロセスの実行中、さまざまな syncresponses コレクションがスコープ内またはスコープ外になる可能性があります。各コレクションは、その直近の〈sequence〉にのみ関連し、その他の〈sequence〉とは関連性がありません。

以下の例は、3 つのスレッドにおける synctimedout と syncresponses の使用法を示しています。3 つのスレッドとは、プライマリ・ビジネス・プロセスのスレッドと、〈flow〉によって作成された 2 つのスレッドです。

Class Member

```
XData BPL
{
<process>
  <context>
    <property name="ResultsFromNorth" type="%String"/>
    <property name="ResultsFromSouth" type="%String"/>
    <property name="ResultsFromEast" type="%String"/>
```

```
    <property name="ResultsFromWest" type="%String"/>
  </context>
  <sequence>
  // In this context, syncresponses refers to the primary process thread
    <flow>
    // This flow runs two sequences (two threads) in parallel

      <sequence name="thread1">
      // In this context, syncresponses refers to results in thread1
        <call name="A" />
        <call name="B"  />
        <sync calls="A,B" type="all" timeout="10" />
        // Did the synchronization time out before it finished?
        <if condition='synctimedout="1"'>
          <true>
            <trace value='"thread1 timeout: Call A or B did not return."' />
          </true>
          // If not, then the calls came back, so assign the results.
          <false>
            <assign property="context.ResultsFromEast"
                    value='syncresponses.GetAt("A")'
                    action="append"/>
            <assign property="context.ResultsFromWest"
                    value='syncresponses.GetAt("B")'
                    action="append"/>
          </false>
        </if>
        </sequence>

        <sequence name="thread2">
        // In this context, syncresponses refers to results in thread2
          <call name="C" />
          <call name="A" />
          <sync calls="C,A" type="all"/>
          // Assign the results
          <assign property="context.ResultsFromNorth"
                  value='syncresponses.GetAt("C")'
                  action="append"/>
          <assign property="context.ResultsFromSouth"
                  value='syncresponses.GetAt("A")'
                  action="append"/>
        </sequence>
      </flow>

    // In this context, syncresponses refers to the primary process thread
    <call name="E" />
  </sequence>
</process>
}
```

この例の 〈if〉 アクティビティでは synctimedout が整数値 1 であるかどうかをテストする条件が指定されています。〈call〉 に関するドキュメントで説明しているように、synctimedout の値は、0、1、または 2 となります。2 つの値が等しい場合、この 〈if〉 条件は整数値 1 を受け取り、〈true〉 要素内の文が実行されます。それ以外の場合は、〈false〉 要素内の文が実行されます。

allowresync

BPL ビジネス・プロセスでは、〈call〉 を実行した後、その呼び出しに対して、タイムアウトを指定または指定せずに 〈sync〉 を複数回実行できます。〈sync〉 の allowresync 属性は、この動作を制御します。allowresync を 1 (真) に設定した場合、同じ 〈call〉 に対して引き続き 〈sync〉 を実行できます。その呼び出しが完了するまで、繰り返し実行できます。〈sync〉 の allowresync を値 0 (偽) に設定した場合、同じ呼び出しに対してさらに 〈sync〉 を実行できません。デフォルトの allowresync 値は 0 です。

実行に長時間かかり、いつまでも応答がない可能性のある 〈call〉 A を非同期で実行するとします。A に対する 〈sync〉 の timeout を 5 にします。この 〈sync〉 は、A が完了した時点で即座に制御を返します。また、A が完了しなくても、タイムアウト時間 (5 秒) が経過した時点で制御を返します。ここで、A の所要時間が不明確であるものの、エラーが発生する可能性は低いとします。つまり、通常は 5 秒以内に完了しますが、ときには 1 時間かかる可能性もあり、その場合の遅延は許容可能だとします。この場合、通常の時間に完了するが、同じ 〈call〉 に対してさらに 〈sync〉 アクティビティを実行できるケース、または完了に長時間かかるケースに備えて、A が完了したかどうかを頻繁にチェックする必要があります。

一般的な使用法は以下のとおりです。

XML

```
<sequence>
  <call name="A" async="1" />
  <sync call="A" timeout="5" allowresync="1" />
  <while condition='synctimedout=1'>
    <alert value="Waiting for call A to complete."/>
    <sync call="A" timeout="5" allowresync="1" />
  </while>
</sequence>
```

<sync> でタイムアウトを指定しない場合は、各 <sync> の前に synctimedout 変数をチェックすることが重要です。この変数をチェックしないと、既に完了している呼び出しを待機し続ける可能性があります。

連続する <sync> タイムアウト

同じ <call> 要素を参照する複数の <sync> 要素を連続して指定し、各 <sync> に timeout 値を設定したとします。<call> が返されるか、または <sync> の timeout 時間が経過したため、最初の <sync> が満たされると、2 番目の <sync> 要素は待機せずに即座に完了します。

XML

```
<sequence>
  <call name="A" async="1" />
  <sync name="Sync1" calls="A" type="all" timeout="60" />
  <sync name="Sync2" calls="A" type="all" timeout="300" />
</sequence>
```

〈throw〉

指定された特定のフォールトをスローします。

構文

```
<scope>
   <throw fault='"MyFault"'/>
         ...
   <faulthandlers>
     <catch fault='"MyFault"'>
           ...
     </catch>
   </faulthandlers>
</scope>
```

詳細

属性または要素	説明	値
fault	必須項目。フォールトの名前。この値には、リテラル文字列または評価対象となる式を指定できます。	0 ～ 255 文字の文字列。式の場合は、この属性が含まれる〈process〉要素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。
name、disabled、xpos、ypos、xend、yend	“一般的な属性と要素”を参照してください。	
〈annotation〉要素		

説明

〈throw〉文が実行されると、同じ〈scope〉内にある〈faulthandlers〉ブロックに即座に制御が移り、〈throw〉以降の途中の文はすべてスキップされます。次にプログラムは、〈faulthandlers〉ブロック内で、〈throw〉文の fault 文字列式と value 属性が同じである〈catch〉ブロックを探します。この比較では大文字と小文字が区別されます。フォールト文字列を指定するときは、以下のように、1 組の引用符を余分に追加する必要があります。

XML

```
<throw fault='"thrown"'/>
```

フォールトと一致する〈catch〉ブロックが見つかった場合は、その〈catch〉ブロック内のコードを実行して、〈scope〉から抜けます。終わりの〈/scope〉要素の次の文から実行を再開します。

フォールトが発生し、フォールト文字列に一致する〈catch〉ブロックが〈faulthandlers〉ブロック内に存在しない場合は、〈throw〉文から、〈faulthandlers〉内の〈catchall〉ブロックに制御が移ります。〈catchall〉ブロック内のコードを実行した後で、〈scope〉から抜けます。終わりの〈/scope〉要素の次の文から実行を再開します。予期しないエラーを確実にキャッチするため、すべての〈faulthandlers〉ブロック内に〈catchall〉ブロックを配置することをお勧めします。

詳細は、″BPL プロセスの開発″の“BPL のエラー処理”を参照してください。

関連項目

〈catch〉、〈catchall〉、〈compensate〉、〈compensationhandlers〉、〈faulthandlers〉、〈scope〉.

〈trace〉

フォアグラウンドのターミナル・ウィンドウにメッセージを書き込みます。

構文

```
<trace value='"The time is: " & Now' />
```

詳細

属性または要素	説明	値
value 属性	必須項目。トレース・メッセージのテキストです。この値には、リテラル文字列または評価対象となる式を指定できます。	1 文字以上の文字列。式またはリテラル文字列を使用できます。式の場合は、この属性が含まれる 〈process〉 要素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。{} 規則を使用して仮想プロパティ構文を作成することもできます。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
〈annotation〉 要素		

説明

〈trace〉 要素は、ターミナル・ウィンドウにメッセージを書き込みます。〈trace〉 メッセージが表示されるのは、生成元の BPL ビジネス・プロセスが [フォアグラウンドで実行] モードに設定されている場合のみです。

トレース・メッセージは、コンソールだけでなく、InterSystems IRIS のイベント・ログにも書き込むことができます。システム管理者がこの動作を制御するには、管理ポータルの [Interoperability]→[構成する]→[プロダクション] ページからプロダクションを構成します。BPL ビジネス・プロセスの [トレース・イベントを記録] オプションにチェックが付いている場合は、トレース・メッセージがイベント・ログに書き込まれると同時にコンソールにも表示されます。トレース・メッセージがログに書き込まれる場合は、そのイベント・ログ・エントリのタイプがトレースになります。

BPL の 〈trace〉 要素は、[ユーザ] の優先度を持つトレース・メッセージを生成します。結果は、ObjectScript から $$$TRACE ユーティリティを呼び出した場合と同じです。

注釈　詳細は、″プロダクションの開発″ の “InterSystems IRIS のプログラミング” の章で “トレース要素の追加” を参照してください。

〈transform〉

データ変換を使用して、あるオブジェクトを別のオブジェクトに変換します。

構文

```
<transform class="MyApp.SAPtoJDE" target="context.xform" source="request" />
```

詳細

属性または要素	説明	値
class 属性	必須項目。データ変換を実行するデータ変換クラスの名前。この値は、リテラル文字列として指定するか、または @ 間接演算子でコンテキスト変数の値を参照することによって指定できます。この後の説明を参照してください。	データ変換クラスの名前。
target 属性	必須項目。このデータ変換のターゲット（出力オブジェクト）。実行コンテキストのいずれかのオブジェクト、またはこれらのいずれかのオブジェクトのプロパティです。	実行コンテキスト内で有効なプロパティとオブジェクトの名前。
source 属性	必須項目。このデータ変換のソース（入力オブジェクト）。これは、現在の実行コンテキストに表示されるオブジェクトの 1 つ、またはそのオブジェクトのプロパティです。	実行コンテキスト内で有効なプロパティとオブジェクトの名前。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
〈annotation〉要素		

説明

〈transform〉要素では、ビジネス・プロセス内からデータ変換クラスを呼び出すことができます。

データ変換クラス（Ens.DataTransform のサブクラス）は、入力オブジェクトのインスタンスを出力オブジェクトのインスタンスに変換するメソッドを定義します。〈transform〉要素は、このメソッドを自動的に呼び出し、class 属性で指定されているデータ変換クラスを使用して、あるタイプのオブジェクトを別のタイプに変換します。

source 属性は、変換の入力オブジェクトを指定します。これは、ビジネス・プロセスの実行コンテキスト内で表示されるオブジェクト（またはそのオブジェクト値プロパティの 1 つ）であり、指定されたデータ変換クラスで予期される入力タイプであることが必要です。

target 属性は、出力オブジェクトの変換先を指定します。これも、ビジネス・プロセスの実行コンテキスト内で表示されるオブジェクト（またはそのオブジェクト値プロパティの 1 つ）であり、指定されたデータ変換クラスで予期される出力タイプであることが必要です。

実行コンテキストの変数

〈transform〉要素は以下の変数とそのプロパティを参照できます。このリストに示されていない変数を使用しないでください。

変数	目的
context	context オブジェクトは、ビジネス・プロセスの汎用データ・コンテナです。context は自動的には定義されません。このオブジェクトのプロパティを定義するには、<context> 要素を使用します。その場合、<process> 要素内でこれらのプロパティを参照するには、`context.Balance` のようにドット構文を使用します。
request	request オブジェクトには、このビジネス・プロセスをインスタンス化させた元の要求メッセージ・オブジェクトのプロパティが含まれます。<process> 要素内で request のプロパティを参照するには、`request.UserID` のようにドット構文を使用します。
response	response オブジェクトには、ビジネス・オブジェクトが返す最終的な応答メッセージ・オブジェクトの構築に必要となるすべてのプロパティが含まれます。`response.IsApproved` と同様に、ドット構文を使用して、<process> 要素内部の response プロパティを参照できます。これらのプロパティに値を割り当てるには <assign> 要素を使用します。

注釈　　ビジネス・プロセスの実行コンテキストの詳細は、<assign> 要素の説明を参照してください。

class 属性の値

<transform> 要素では、ビジネス・プロセスからデータ変換クラスを呼び出せますが、事前に BPL のサブセットであるデータ変換言語 (DTL) を使用してデータ変換クラスが定義されている必要があります。DTL の詳細は、"データ変換言語リファレンス" を参照してください。

class 属性の間接指定

class 属性の値は、DTL データ変換のパッケージとクラス名を表す文字列です。以下のように、この文字列はリテラル値でもかまいません。

```
<transform class="MyApp.SAPtoJDE" target="context.xform" source="request" />
```

また、次のように @ 間接演算子を使用して、適切な文字列を含むコンテキスト変数の値にアクセスすることもできます。

```
<call class="@context.nextTransform" target="context.xform" source="request"/>
```

〈true〉

〈if〉 要素の条件が真の場合に、一連のアクティビティを実行します。

構文

```
<if condition="1">
   <true>
     ...
   </true>
   <false>
     ...
   </false>
</if>
```

詳細

属性または要素	説明
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。
〈annotation〉 要素	
その他の要素	オプション。〈true〉 には、〈alert〉、〈assign〉、〈branch〉、〈break〉、〈call〉、〈code〉、〈continue〉、〈delay〉、〈empty〉、〈flow〉、〈foreach〉、〈if〉、〈label〉、〈milestone〉、〈reply〉、〈rule〉、〈scope〉、〈sequence〉、〈sql〉、〈switch〉、〈sync〉、〈throw〉、〈trace〉、〈transform〉、〈until〉、〈while〉、〈xpath〉、および 〈xslt〉 をゼロ個以上、自由に組み合わせて使用できます。

説明

〈true〉 要素は、〈if〉 内で、条件が真の場合に実行する必要がある要素を収容するのに使用されます。

<until>

条件が真になるまで、アクティビティを繰り返し実行します。

構文

```
<until condition='context.IsApproved="1"'>
   ...
</until>
```

詳細

属性または要素	説明	値
condition 属性	必須項目。<until> 要素内のアクティビティが実行された後で、毎回この式が評価されます。結果が真になると、<until> 要素の実行が停止されます。	整数値 1 (真の場合) または 0 (偽の場合) を求める式。この式では、この属性が含まれる <process> 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		
その他の要素	オプション。<until> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath>、および <xslt> をゼロ個以上、自由に組み合わせて使用できます。	

説明

<until> 要素は、論理式の結果が整数値 1 (真) になるまで繰り返し実行される一連のアクティビティを定義します。この式は、一連のアクティビティの実行が終了するたびに再評価されます。

ループの実行を微調整するには、<until> 要素内で <break> 要素と <continue> 要素を使用します。詳細は、各要素の説明を参照してください。

<while>

条件が真である限り、繰り返しアクティビティを実行します。

構文

```
<while condition='context.IsApproved="1"'>
   ...
</while>
```

詳細

属性または要素	説明	値
condition 属性	必須項目。この式は、<while> 要素内のアクティビティが実行される前に毎回評価されます。結果が偽になると、<while> 要素の実行が停止されます。	整数値 1（真の場合）または 0（偽の場合）を求める式。この式では、この属性が含まれる <process> 要素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。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
<annotation> 要素		
その他の要素	オプション。<while> には、<alert>、<assign>、<branch>、<break>、<call>、<code>、<continue>、<delay>、<empty>、<flow>、<foreach>、<if>、<label>、<milestone>、<reply>、<rule>、<scope>、<sequence>、<sql>、<switch>、<sync>、<throw>、<trace>、<transform>、<until>、<while>、<xpath>、および <xslt> をゼロ個以上、自由に組み合わせて使用できます。	

説明

<while> 要素は、論理式の結果が整数値 1（真）である限り、繰り返し実行される一連のアクティビティを定義します。この式は、一連のアクティビティの実行が開始される前に毎回再評価されます。

ループの実行を微調整するには、<while> 要素内で <break> 要素と <continue> 要素を使用します。以下に例を示します。

XML

```
<while condition="0">

  //...do various things...

  <if condition="somecondition">
    <true>
      <break/>
    </true>
  </if>

  //...do various other things...

</while>
```

〈xpath〉

ターゲットの XML ドキュメントで XPath 式を評価します。

構文

```
<xpath name='xpath'
        source="request.MetaDataXML"
        property="context.Result" context="/staff/doc"
        expression="name[@last='Marston']"/>
```

詳細

属性または要素	説明	値
source 属性	必須項目。XPath 式の実行対象とする XML を含むストリームを生成する式。通常、source 属性によりコンテキストまたは要求のプロパティを指定します。	1 文字以上の文字列。
property 属性	必須項目。評価結果の配置先とするプロパティ (通常はコンテキスト・プロパティ)。	1 文字以上の文字列。
context 属性	必須項目。ドキュメント・コンテキスト。	1 文字以上の文字列。
expression 属性	必須項目。XPath 式。	1 文字以上の文字列。
prefixmappings 属性	オプション。ドキュメントの接頭語マッピングを指定します。これは、接頭語とネームスペースのマッピングをカンマ区切りで記述したリストです。この後の説明を参照してください。	0 ～ 255 文字の文字列。
schemaspec 属性	オプション。スキーマ仕様。	0 ～ 255 文字の文字列。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素”を参照してください。	
〈annotation〉 要素		

説明

〈xslt〉 要素を使用するビジネス・プロセスでは、ターゲットの XML ドキュメントで XPath 式を評価できます。

〈xpath〉 要素を実行すると、source ストリームが処理されて XPath ドキュメントが生成され、XPath 式が順に評価されます。BPL ランタイム・エンジンは自動的にドキュメントの存続期間を管理し、処理が可能な限り効率的になるようこれらをキャッシュします。

各 prefixmappings エントリは、接頭語、空白、接頭語がマッピングされる URI として定義されます。これは特に、ドキュメントがデフォルトのネームスペースを `xmlns="http://somenamespaceuri"` 構文で定義しているにもかかわらず、明示的な接頭語マッピングを指定していない場合に有効です。次の prefixmappings 文字列は、`myprefix` という接頭語を `http://somenamespaceuri` という URI にマッピングします。文字列中の空白文字に注意してください。

```
prefixmappings="myprefix http://somenamespaceuri"
```

BPL の 〈xpath〉 要素は、スカラ値 (単一のテキスト、数値、データなど) を生成する XPath 式のサポートを目的としています。XPath DOM を生成する式を処理することが目的ではありません。つまり、式により DOM が生成される場合、ター

ゲットのプロパティは更新されません。DOM プログラミングは BPL の範囲外です。ビジネスでこのような処理を必要とする場合は、XPath をコード・ブロックまたはユーティリティ・クラスの呼び出しで実行する必要があります。

関連項目

〈xslt〉

〈xslt〉

埋め込まれた XSLT 変換を実行します。

構文

```
<xslt name='simon'
       xslurl="http://www.intersystems.com/transform.xsl"
       source="context.a" target="context.b">
   <parameters>
     <parameter name="surname" value="sez"/>
   </parameters>
</xslt>
```

詳細

属性または要素	説明	値
xslurl 属性	必須項目。変換を制御する XSLT 定義の URI。この URI は、文字列 “file:”、“http:”、“url:”、または “xdata:” で始まります。	0 ～ 255 文字の文字列。
source 属性	必須項目。ソース (ストリーム) オブジェクトの名前。	0 ～ 255 文字の文字列。
target 属性	必須項目。ターゲット (ストリーム) オブジェクトの名前。	0 ～ 255 文字の文字列。
name、disabled、xpos、ypos、xend、yend 属性	“一般的な属性と要素” を参照してください。	
〈annotation〉 要素		
〈parameters〉 要素	オプションの 〈parameters〉 要素を使用できます。〈parameters〉 コンテナ内には、ゼロ個以上の 〈parameter〉 要素を配置できます。各 〈parameter〉 要素は、XSLT 変換を制御するスタイルシートに渡す XSLT 名前-値ペアを定義します。	
xsltversion 属性	XSLT 変換が XSLT 1.0 を使用するか 2.0 を使用するかを指定します。	値 "1.0" または "2.0" を含む文字列。

説明

〈xslt〉 要素を使用すると、ビジネス・プロセスの実行中に XSLT 変換を適用できます。〈xslt〉 要素は、任意の XSLT 定義に基づいて入力ストリームを出力ストリームへ変換します。XSLT 定義は外部ファイルに記述できます。また、同じネームスペースのクラスで、BPL ビジネス・プロセスとして定義することもできます。

source ストリーム・オブジェクトと target ストリーム・オブジェクトは、ビジネス・プロセスの context オブジェクトのプロパティとして宣言する必要があります。context オブジェクトは、ビジネス・プロセスの汎用データ・コンテナです。context オブジェクトのプロパティを定義するには、〈process〉 要素の先頭に 〈context〉 要素と 〈property〉 要素を配置します。その場合、〈process〉 要素の中でこれらのプロパティを参照するには、`context.MyInputStream` や `context.MyOutputStream` のようにドット構文を使用します。

xslurl 文字列は、XSLT 定義の場所を表す URI です。xslurl 値は、以下のいずれかの文字列で始まります。

```
file:
http:
url:
xdata:
```

`file:`、`http:`、および `url:` の意味は通常と同じです。`xdata:` 文字列の形式は以下のとおりです。

`xdata://`PackageName.ClassName`:`XDataName

説明：

- PackageName.ClassName は、BPL ビジネス・プロセスと同じネームスペースにあるクラスを特定します。
- XDataName は、上記のクラス内にあり、この 〈xslt〉 文の XSLT 定義を含む XData ブロックの名前です。この規則に従って、XSLT 定義を InterSystems IRIS のクラス内に格納すれば、InterSystems IRIS 以外のローカル・ファイル・システムや Web 上に格納する場合より効率的に処理できます。

XSLT がパラメータを必要とする場合は、〈xslt〉 要素内に 〈parameters〉 ブロックを配置します。

関連項目

〈parameters〉 および 〈xpath〉

DTL 要素

このリファレンスでは、DTL 要素のそれぞれについて詳しく説明します。

Tipヒント DTL 用の XML を表示または編集する場合は、スタジオを使用して DTL を編集し、**[他のコードを表示]** をクリックします。

DTL <annotation>

DTL 要素についての詳しいコメントを指定します。

構文

```
<annotation>
    Sends patient data from lab to CRM system.


</annotation>
```

説明

<annotation> 要素により、DTL 要素に詳しいコメントを関連付けることができます。<annotation> は、注釈の対象となる要素の最初の子として挿入する必要があります。以下に例を示します。

XML

```
<transform targetClass='Demo.DTL.ExampleTarget'
           sourceClass='Demo.DTL.ExampleSource'
           create='new'
           language='objectscript'>
  <annotation>
    Implement current naming conventions.


  </annotation>
  <trace value='"Convert from lowercase to uppercase"'/>
  <assign property='target.UpperCase'
          value='$ZCONVERT(source.LowerCase,"U")'
          action='set'>
   <annotation>This is a comment for the assign element</annotation>
  </assign>
</transform>
```

上の例では、注釈テキストの前後に CDATA 構文を使用しています。この規則はオプションですが、これによって、XML エスケープ・シーケンスに影響されずに、アポストロフィ (') などの特殊文字を入力したり、改行を挿入したりできます。<annotation> 文字列の最大長は、CDATA エスケープ文字も含めて 32,767 文字です。

また、assign 要素に関する注釈が開始 assign タグの直後の子として挿入されている点にも注目してください。

DTL 内のほとんどの要素が <annotation> を子要素としてサポートしています。これにより、DTL 要素に詳しいコメントを関連付けることができます。すべての要素に対して位置属性を提供する BPL とは異なり、<annotation> は、ほとんどの DTL 要素が共通して持つ、唯一の子要素または属性です。<annotation> 要素を使用する場合は、注釈の対象となる要素の最初の子として挿入する必要があります。

DTL <assign>

オブジェクトのプロパティに値を割り当てます。

構文

```
<assign property="propertyname" value="expression" />
```

属性

属性	説明	値
property	必須項目。この割り当てのターゲットであるプロパティです。	1 文字以上の文字列。
value	必須項目。プロパティの値を指定します。	プロパティの有効な値を指定する ObjectScript 式。
action	オプション。value がコレクション・プロパティ (リストまたは配列) の場合は、action を使用して、実行する割り当てのタイプを指定します。デフォルトは set アクションです。	以下の値、set、clear、remove、append、insert のいずれか 1 つです。詳細は、"アクション" の節を参照してください。
key	value がコレクション・プロパティ (リストまたは配列) である一部の場合を除き、オプション。該当する場合は、このキーを使用して割り当ての実行対象となる要素を指定します。	1 文字以上の文字列。この文字列は、キーとして評価される式です。

要素

要素	目的
<annotation>	オプション。<assign> 要素について記述するテキスト文字列。

説明

DTL の <assign> 要素は、DTL <transform> 要素内から、ターゲット・プロパティとそのプロパティに値を割り当てる式を指定するために使用されます。一般に、この式はデータ変換のソース・オブジェクトから取得した値を含みますが、リテラル値も使用できます。DTL の <assign> アクティビティに含まれるプロパティはすべて、データ変換のソースまたはターゲット・オブジェクト内のプロパティでなければなりません。

"プロダクションの開発" の "メッセージ" の章で説明されているように、ソース・オブジェクトおよびターゲット・オブジェクトは、通常、プロダクション・メッセージ本文オブジェクトです。これらは、メッセージ・ヘッダとメッセージ本文オブジェクトで構成されます。

標準のプロダクション・メッセージ本文のプロパティは、データ型、オブジェクト、またはこれらいずれかのコレクションです。コレクション・プロパティは、クラス定義で `[ Collection = list ]` または `[ Collection = array ]` のいずれかで宣言されます。標準のプロダクション・メッセージ本文のプロパティは、任意のオブジェクト・プロパティについて、ドット構文を使用して参照できます。

仮想ドキュメントのプロパティには、ドキュメント "プロダクション内での仮想ドキュメントの使用法" の以下の節に説明がある、独自の構文が必要です。

- 仮想プロパティ・パス
- 仮想プロパティ・パスに関する構文ガイド

<assign> 要素のアクション

DTL の <assign> 操作には、オプションの action 属性で指定される、いくつかのタイプがあります。デフォルト値の set を除くと、このようなアクションは、標準のプロダクション・メッセージ本文内のコレクション・プロパティに関係した割り当てを処理するためのものです。以下の表に、<assign> 要素のアクションを説明します。

アクションの割り当て	説明	例
set	指定したプロパティの値を value 属性の値に設定します。value 属性は、式を含んでおり、それ自体がオブジェクトまたはオブジェクトのプロパティを参照できます。	以下の文では、ターゲットの BankName プロパティの値が次のように設定されます。 XML `<assign property='target.BankName' value='process.BankName' action='set'/>`
append	リスト・プロパティの最後にターゲット要素を追加します。	
clear	指定したコレクション・プロパティの内容を消去します。value 属性と key 属性は無視されます。(これは、コレクション・プロパティにのみ当てはまります。)	以下の文では、コレクション・プロパティ List の内容が消去されます。 XML `<assign property='target.List' action='clear' />`

insert	指定したコレクション・プロパティに値を挿入します。key 属性が存在する場合は、key で指定された位置(整数)の後ろに新しい値が挿入されます。指定がない場合、新しい項目は末尾に追加されます。(これは、リスト・コレクション・プロパティにのみ当てはまります。)	以下の文では、キー primary を使用して、値が配列コレクション・プロパティ Array に挿入されます。
		
		XML
		
		`<assign`
		`  property='target.Array'`
		`  action='insert'`
		`  key='primary'`
		`  value='source.Primary'`
		`  />`

remove	指定したコレクション・プロパティからアイテムを削除します。value 属性は無視されます。(これは、コレクション・プロパティにのみ当てはまります。)	

注釈　　仮想ドキュメントでは、set 以外の action 値は使用されません。

set アクションでは、指定したプロパティの値が value 属性の値に設定されます。value 属性は、以下のように、式を含んでおり、それ自体がオブジェクトまたはオブジェクトのプロパティを参照できます。

XML

```
<assign property='target.SSN' value='source.SSN' />
```

ターゲット・プロパティが配列コレクションの場合、key 属性の値は配列内のアイテムを指定します。それ以外の場合、key 属性は無視されます。

ターゲット・プロパティがコレクションで、value 属性で以下のように同じタイプが指定されている場合は、コレクションの内容がターゲット・コレクションにコピーされます。

XML

```
<assign property='target.List' value='source.List' />
```

assign 要素のデフォルトのアクションは set 操作です。action が指定されていない場合、assign では set 操作が指定されます。

オブジェクトとオブジェクト参照

ソースとして別のオブジェクトの任意のオブジェクト・プロパティまたは最上位のソース・オブジェクトから <assign> する場合、ターゲットは、オブジェクト自体ではなくそのオブジェクトの複製されたコピーを受け取ります。これにより、オブジェクト参照の不注意な共有を避け、ユーザ自身が複製オブジェクトを生成する手間を省きます。ただし、ソースとターゲットの間でオブジェクト参照を共有したい場合は、ソースから中間の一時的な変数に <assign> して、その後、その変数からターゲットに <assign> します。

一斉コピー

ソースの完全コピーであるターゲット・オブジェクトを作成するには、以下を使用しないでください。

```
<assign property='target' value='source' />
```

代わりに、含まれる <transform> 要素内で `create='copy'` 属性を使用します。

create オプションは、以下の値のいずれかを持つことができます。

- `new` ー データ変換内の要素を実行する前に、ターゲット・タイプの新しいオブジェクトを作成します。これがデフォルトです。
- `copy` ー 変換内の要素を実行する前に、ターゲット・オブジェクトとして使用するソース・オブジェクトのコピーを作成します。
- `existing` ー ターゲット・オブジェクトとしてデータ変換の呼び出し側により指定される既存のオブジェクトを使用します。

DTL <break>

For Each ループを終了するか、またはデータ変換処理を停止します。

構文

```
<break/>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<break> 要素について記述するテキスト文字列。

説明

<foreach> 要素に含まれている場合、<break> 要素は FOR Each ループを終了します。<break> が For Each ループの外にある場合、データ変換はすべて、break が実行されるとすぐに終了します。

DTL 〈case〉

指定された条件が満たされると、〈switch〉 要素内のアクションのブロックを実行します。

構文

```
<switch>
   <case condition="1">
   ...
   </case>
   <default>
   ...
   </default>
</switch>
```

属性

属性	説明	値
condition	必須項目。ObjectScript 条件式が真の場合、〈case〉 要素内のコードが実行されます。	整数値 1 (真の場合) または 0 (偽の場合) を求める式。

要素

要素	目的
〈annotation〉	オプション。〈case〉 要素について記述するテキスト文字列。

説明

〈switch〉 要素は 1 つ以上の 〈case〉 要素を含みます。〈case〉 要素内の要素は、条件が真に評価されると実行されます。

DTL <code>

カスタム・コード行を実行します。

構文

```
<code>
    target.Name = source.FirstName & " " & source.LastName

</code>
```

要素

要素	目的
<annotation>	**オプション。<code> 要素について記述するテキスト文字列。**

説明

DTL <code> 要素は、DTL データ変換内の 1 行または複数行のユーザ記述コードを実行します。<code> 要素を使用すると、DTL 要素では表現が難しい特殊な作業を実行できます。<code> 要素で参照するプロパティは、データ変換のソースまたはターゲット・オブジェクト内のプロパティである必要があります。

DTL <code> 要素のスクリプト言語は、含まれる <transform> 要素の **language** 属性によって指定されます。値は **objectscript** である必要があります。データ変換内のすべての式と <code> 要素内のコード行では、指定された言語を使用する必要があります。

詳細は、以下のドキュメントを参照してください。

- ObjectScript の使用法
- ObjectScript リファレンス

通常、開発者は、アポストロフィー (') や アンパサンド (&) などの特殊な XML 文字をエスケープしなくてすむように、<code> 要素の内容を **CDATA** ブロック内にラップします。以下に例を示します。

XML

```
<code>
   target.Name = source.FirstName & " " & source.LastName

</code>
```

データ変換の実行を中断およびリストアできるようにするには、<code> 要素の使用時に以下のガイドラインに従う必要があります。

- 実行時間は短くします。カスタム・コードがデータ変換の一般処理を妨げないようにしてください。
- 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、必ず同じ <code> 要素内でそのリソースを解放してください。
- <code> 要素がトランザクションを開始する場合は、同じ <code> 要素が考えられるすべてのシナリオでトランザクションを終了することを確認します。トランザクションを終了していない場合には、トランザクションは無期限に開いたままになる可能性があります。これにより他の処理が阻止されたり、重大なダウンタイムを発生することがあります。

DTL <comment>

DTL にコメントを追加します。

構文

```
<comment>
   <annotation>
   ...
   </annotation>
</comment>
```

属性

なし

要素

要素	目的
<annotation>	コメントを含むテキスト文字列。

説明

<comment> の <annotation> 要素の内容は、管理ポータルに表示され、DTL アクションについて説明します。

DTL <default>

<switch> 要素内のどの <case> 要素も真に評価されない場合、内容を実行します。

構文

```
<switch>
   <case condition="1">
   ...
   </case>
   <default>
   ...
   </default>
</switch>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<default> 要素について記述するテキスト文字列。

説明

<default> 要素は <switch> 要素の末尾に記され、どの <case> 要素も真に評価されない場合に実行されます。

DTL <false>

<if> 要素の条件が偽の場合に、一連のアクティビティを実行します。

構文

```
<if condition="0">
   <true>
      ...
   </true>
   <false>
      ...
   </false>
</if>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<false> 要素について記述するテキスト文字列。
ほとんどのアクティビティ	オプション。<false> には、<assign>、<code>、<foreach>、<if>、<sql>、<subtransform>、および <trace> をゼロ個以上、自由に組み合わせて使用できます。

説明

<false> 要素は、<if> 内で、条件が偽の場合に実行する必要のある要素を含めるために使用されます。

DTL <foreach>

繰り返し実行される一連のアクティビティを定義します。

構文

```
<foreach property="P1" key="K1">
   ...
</foreach>
```

属性

属性	説明	値
property	必須項目。繰り返しが行われるコレクション・プロパティ(リストまたは配列)。これは、実行コンテキスト内の有効なオブジェクト名およびプロパティ名である必要があります。	1 文字以上の文字列。
key	必須項目。コレクション内で繰り返しを実行するために使用するインデックス。これは、実行コンテキスト内の有効なオブジェクト名およびプロパティ名である必要があります。コレクション内の要素ごとに値が割り当てられます。	1 文字以上の文字列。

要素

要素	目的
<annotation>	オプション。<foreach> 要素について記述するテキスト文字列。
ほとんどのアクティビティ	オプション。<foreach> には、<assign>、<code>、<foreach>、<if>、<sql>、<subtransform>、および <trace> をゼロ個以上、自由に組み合わせて使用できます。

説明

<foreach> 要素では、繰り返し実行される一連のアクティビティを定義します。これらのアクティビティは、指定したコレクション・プロパティ内にある各要素について 1 回ずつ実行されます。要素が NULL の場合、このシーケンスは実行されません。このシーケンスは、要素に空の値がある場合に (つまり、セパレータはあるがその間に値がない場合に) 実行されます。しかし、NULL 値に対しては実行されません。つまり、フィールドが指定される前にメッセージが終了します。

以下に例を示します。

XML

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

または

XML

```
<foreach key='key' property='source.{PID:PatientIDInternalID()}'>
 <if condition='source.{PID:PatientIDInternalID(key).identifiertypecode}="PAS"'>
  <true>
   <assign property='target.{PID:PatientIdentifierList(key).identifiertypecode}'
           value='"MR"'
           action='set'/>
  </true>
 </if>
 <if condition='source.{PID:PatientIDInternalID(key).identifiertypecode}="GMS"'>
  <true>
   <assign property='target.{PID:PatientIdentifierList(key).identifiertypecode}'
           value='"MC"'
           action='set'/>
   <assign property='target.{PID:PatientIdentifierList(key).assigningfacility}'
           value='"AUSHIC"'
           action='set'/>
  </true>
 </if>
</foreach>
```

<foreach> 要素で参照するプロパティは、データ変換のソースまたはターゲット・オブジェクト内のプロパティである必要があります。

ネストされた <foreach>

<foreach> 要素をネスト構造にすることができます。

次のトピックでは、この <foreach> 構文をどのように簡素化できるかを説明します。

<foreach> のショートカット

ドキュメントベースのメッセージまたは“仮想ドキュメント”タイプを扱っている場合は、<assign> 文によって、ドキュメント構造内の反復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ショートカット表記が提供されます。したがって、繰り返しフィールドを処理するためだけに、'i' 'j' および 'k' を使用した <foreach> ループを実際に設定する必要はありません。代わりに、空のかっこを使用したより簡単な表記を使用できます。

このかっこを使用するショートカットは、ネスト構造の <foreach> にも使用できます。

ラージ・メッセージでの <STORE> エラーの回避

メッセージまたはオブジェクト・コレクションでセグメントのループ処理を行う際には、セグメントがメモリに書き込まれます。これらのオブジェクトによって現在のプロセスに割り当てられているすべてのメモリが消費されると、予期しないエラーが発生することがあります。

これを避けるために、不要になったオブジェクトはメモリから削除します。例えば、<foreach> ループで多数のセグメントを処理する場合、ループの最終ステップでソースとターゲットの両方に対して commitSegmentByPath メソッドを呼び出します。同様に、オブジェクト・コレクションの場合は %UnSwizzleAt メソッドを使用します。

コードを変更できない場合の一時的な回避策としては、各プロセスに割り当てられているメモリの量を増やします。この変更を行うには、管理ポータルの **[メモリ詳細設定]** ページで bbsiz パラメータを設定します。この場合、システムを再起動する必要がある点に注意してください。また、この変更を行う前に、必ずシステム管理者に相談してください。

DTL <group>

関連する要素を表示ユニットにまとめます。

構文

```
<group>
   ...
</group>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<group> 要素について記述するテキスト文字列。
すべて	どの要素も <group> 要素に追加できます。

説明

<group> 要素は、関連する要素を論理ユニットにまとめます。

DTL <if>

条件を評価し、真の場合の操作または偽の場合の操作を実行します。

構文

```
<if condition="1">
   <true>
     ...
   </true>
   <false>
     ...
   </false>
</if>
```

属性

属性	説明	値
condition	必須項目。この ObjectScript 条件式が真の場合、<true> 要素内のコードが実行されます。偽の場合は、<false> 要素内のコードが実行されます。	整数値 1（真の場合）または 0（偽の場合）を求める式。

要素

要素	目的
<annotation>	オプション。<if> 要素について記述するテキスト文字列。
<true>	オプション。条件が真の場合、<true> 要素内のアクティビティが実行されます。
<false>	オプション。条件が偽の場合、<false> 要素内のアクティビティが実行されます。

説明

<if> 要素は式を評価し、その値に応じて、2 つのアクティビティ・セット（式の結果が真の場合のアクティビティと偽の場合のアクティビティ）のいずれかを実行します。

<if> 要素には、<true> 要素と <false> 要素を含めることができます。これらの要素は、式の結果が真の場合および偽の場合に実行されるアクションをそれぞれ定義します。

<true> 要素と <false> 要素を両方指定する場合、<if> 要素ではどちらを先に配置してもかまいません。

条件が真で <true> 要素がない場合、または条件が偽で <false> 要素がない場合、その <if> 要素では何も実行されません。

DTL <sql>

データ変換内で、埋め込みの SQL SELECT 文を実行します。

構文

```
<sql>

      SELECT SSN INTO :context.SSN
      FROM MyApp.PatientTable
      WHERE PatID = :request.PatID


 </sql>
```

要素

要素	目的
<annotation>	**オプション。<sql> 要素について記述するテキスト文字列。**

説明

DTL <sql> 要素は、DTL <transform> 要素内から任意の埋め込み SQL SELECT 文を実行します。

<sql> 要素を効率的に使用するため、以下のヒントを参考にしてください。

- 次のように、必ず、SQL スキーマ名とテーブル名の両方を含むテーブルの完全修飾名を使用します。

  `MyApp.PatientTable`

  上記の例の `MyApp` は SQL スキーマ名、`PatientTable` はテーブル名です。

- <sql> 要素の内容には、有効な埋め込み SQL SELECT 文が含まれている必要があります。

  SQL クエリを CDATA ブロック内に配置すれば、特殊な XML 文字のエスケープに影響されることはありません。

- SQL クエリの FROM 節にリストされるテーブルは、ローカルの InterSystems IRIS データベースに格納されているか、SQL ゲートウェイを使用して外部リレーショナル・データベースにリンクされている必要があります。

- SQL クエリの INTO 節および WHERE 節内では、1 つの変数名の前にコロン (:) を付けることにより、ソース・オブジェクトまたはターゲット・オブジェクトのプロパティを参照できます。以下に例を示します。

  XML

  ```
  <sql>
    SELECT Name INTO :target.Name
    FROM MainFrame.EmployeeRecord
    WHERE SSN = :source.SSN AND City = :source.Home.City


  </sql>
  ```

- 使用されるのは、クエリで返された最初の行のみです。WHERE 節では、必要な行を正確に指定してください。

DTL <subtransform>

別のデータ変換を呼び出します。

構文

```
<subtransform class='class-name'
                    targetObj='target-value}'
                    sourceObj='source-value'/>
```

属性

属性	説明	値
class	必須項目。呼び出されるデータ変換を含むクラスの名前です。このクラスは、これを呼び出すクラスと同じネームスペースにあることが必要です。 多くの場合、このトピックの例で示すように、class は、DTL <transform> 要素を使用して定義される DTL データ変換です。 または、class で、Transform メソッドを実装して DTL を使用しない Ens.DataTransform のカスタム・サブクラスを指定することができます。	完全なパッケージとクラスの名前。
sourceObject	必須項目。変換されるプロパティを指定します。これには、オブジェクト・プロパティまたは仮想ドキュメント・プロパティを指定できます。これは通常、対応する <transform> 要素の sourceClass および (仮想ドキュメントの場合は) sourceDocType によって指定されるソース・オブジェクトのプロパティです。この場合、以下のようにドット構文を使用して参照されます。 source.property または source.{propertyPath}	プロパティ名。仮想ドキュメントおよびそのセグメントの場合は、仮想プロパティ構文を使用します。
targetObject	必須項目。変換された値を書き込むプロパティを指定します。これには、オブジェクト・プロパティまたは仮想ドキュメント・プロパティを指定できます。これは通常、対応する <transform> 要素の targetClass および (仮想ドキュメントの場合は) targetDocType によって指定されるターゲット・オブジェクトのプロパティです。この場合、以下のようにドット構文を使用して参照されます。 target.property または target.{propertyPath} [作成] がnew または copy として設定されている subtransform の場合は、既存のターゲット・オブジェクトを持っている必要がありません。	プロパティ名。仮想ドキュメントおよびそのセグメントの場合は、仮想プロパティ構文を使用します。

要素

要素	目的
<annotation>	オプション。<subtransform> 要素について記述するテキスト文字列。

説明

<subtransform> 要素は、別のデータ変換を呼び出します。<subtransform> を呼び出すことによって、対応する <transform> 要素が他のデータ変換を呼び出し、その作業のセグメントを完了させることができます。これにより、開発者は、再利用可能な一連の DTL 変換コードをより柔軟に保持できます。

<subtransform> 要素が利用できるようになる前は、すべての DTL <transform> が孤立していました。同じ一連のアクションを含む複数の DTL 変換を記述するには、コードの該当部分を、クラス間でコピーして貼り付ける必要がありました。現在、これらの DTL クラスではそれぞれ、繰り返される行を <subtransform> 要素に置き換えることにより、別のクラスを呼び出して必要なシーケンスを実行できるようになりました。

<subtransform> のソース・オブジェクトまたはターゲット・オブジェクトは、通常の InterSystems IRIS オブジェクト、仮想ドキュメントのメッセージ・オブジェクト、または仮想ドキュメント・メッセージ内の個々のセグメントを示す仮想ドキュメントのセグメント・オブジェクトです。<subtransform> は、Electronic Data Interchange (EDI) 形式を扱うインタフェース開発者にとって特に重要です。EDI 形式のメッセージやドキュメントにはそれぞれ、変換が必要な独自のセグメントが多くあるためです。<subtransform> を使用することにより、変換を呼び出すためにコードをコピーせずに、再利用可能なセグメント変換のライブラリを作成して必要に応じて呼び出すことができます。

仮想ドキュメントおよびそのセグメントの場合、以下の例のように、中かっこ {} の構文などの仮想プロパティ構文を使用する必要があります。かっこ内のプロパティ・パスは、セグメントやセグメントのグループ内にあるフィールドではなく、特定のセグメントを参照する必要があります。背景情報は、"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。詳しい説明は、"仮想プロパティ・パス" の節にあります。

DTL <switch>

<case> 要素を評価し、最初に真に評価されたものの内容を実行します。

構文

```
<switch>
   <case condition="1">
   ...
   </case>
   <default>
   ...
   </default>
</switch>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<switch> 要素について記述するテキスト文字列。
<case>	真に評価された最初の <case> 要素が実行されます。
<default>	オプション。どの <case> 要素も真に評価されない場合は、<default> 要素の内容が実行されます。

説明

<switch> 要素には、1 つ以上の <case> 要素と、オプションの <default> 要素が含まれます。<case> 要素の内容は、条件が真に評価されると実行されます。<case> 要素が真に評価されると、他の <case> 要素や <default> 要素は評価されません。<default> 要素の内容は、どの <case> 要素も真に評価されない場合に実行されます。

DTL 〈trace〉

フォアグラウンドのターミナル・ウィンドウにメッセージを書き込みます。

構文

```
<trace value='"The time is: " & Now' />
```

属性

属性	説明	値
value	必須項目。トレース・メッセージのテキストです。この値には、リテラル文字列または評価対象となる ObjectScript 式を指定できます。	1 文字以上の文字列。リテラル文字列または式を指定できます。

要素

要素	目的
〈annotation〉	オプション。〈trace〉 要素について記述するテキスト文字列。

説明

〈trace〉 要素は、ターミナル・ウィンドウにメッセージを書き込みます。〈trace〉 メッセージが表示されるのは、DTL データ変換の呼び出し元のビジネス・ホストが **[フォアグラウンドで実行]** モードに設定されている場合のみです。

トレース・メッセージは、コンソールだけでなく、InterSystems IRIS のイベント・ログにも書き込むことができます。システム管理者は、管理ポータルの [構成] ページでこの動作を制御します。DTL データ変換を呼び出すビジネス・ホストで **[トレース・イベントのログ]** オプションが選択されている場合、トレース・メッセージは、コンソールに表示されると共にイベント・ログにも書き込まれます。トレース・メッセージがログに書き込まれる場合、そのイベント・ログ・エントリのタイプは ″トレース″ になります。

DTL の 〈trace〉 要素は、[ユーザ] の優先度を持つトレース・メッセージを生成します。結果は、ObjectScript から $$$TRACE ユーティリティを呼び出した場合と同じです。

注釈　詳細は、″プロダクションの開発″ の “InterSystems IRIS のプログラミング” の章で “トレース要素の追加” の節を参照してください。

DTL 〈transform〉

あるタイプのオブジェクトを別のタイプのオブジェクトに変換します。

構文

```
<transform sourceClass="MyApp.SAPtoJDE"
           targetClass="AlsoMine.JDE"  />
```

属性

属性	説明	値
sourceClass	必須項目。データ変換用の入力オブジェクトのクラス名。	有効なオブジェクトとプロパティの名前。
targetClass	必須項目。データ変換用の出力オブジェクトのクラス名。	有効なオブジェクトとプロパティの名前。
sourceDocType	オプション。入力オブジェクトが仮想ドキュメントの場合、この文字列はその DocType を識別します。	文字列。
targetDocType	オプション。出力オブジェクトが仮想ドキュメントの場合、この文字列はその DocType を識別します。	文字列。
language	オプション。objectscript である必要があります。	objectscript
create	オプション。ターゲット・オブジェクトに必要な create オプションです。指定しない場合は、デフォルトの new が使用されます。	この値には new、copy、existing のいずれかを指定できます。以下に詳しく説明します。

要素

要素	目的
〈annotation〉	オプション。〈transform〉 要素について記述するテキスト文字列。
ほとんどのアクティビティ	オプション。〈transform〉 には、〈assign〉、〈code〉、〈foreach〉、〈if〉、〈sql〉、〈subtransform〉、および 〈trace〉 をゼロ個以上、自由に組み合わせて使用できます。

説明

〈transform〉 要素は、DTL ドキュメントの最も外側の要素です。その他のすべての DTL 要素は、〈transform〉 要素の中に含まれます。〈transform〉 内で、2 つのオブジェクトの名前はそれぞれ source および target です。以下に例を示します。

XML

```
<transform targetClass='Demo.DTL.ExampleTarget'
           sourceClass='Demo.DTL.ExampleSource'
           create='new'
           language='objectscript'>

       <trace value='"Convert from lowercase to uppercase"'/>
       <assign property='target.UpperCase'
          value='$ZCONVERT(source.LowerCase,"U")'
          action='set'/>

</transform>
```

ソース・オブジェクトとターゲット・オブジェクト

sourceClass と targetClass で、標準のプロダクション・メッセージ・クラスを指定できます。それぞれのクラスに一連のプロパティが含まれます。その場合、sourceDocType 属性および targetDocType 属性は必要ありません。

または、sourceClass および targetClass で、仮想ドキュメントを指定することができます。この場合、sourceDocType 属性および targetDocType 属性によって、仮想ドキュメントで期待されるメッセージ構造を InterSystems IRIS に伝える必要があります。

create オプションの値

ターゲット・オブジェクトの create オプションには、以下の値のいずれかを指定できます。

- `new` ー データ変換内の要素を実行する前に、ターゲット・タイプの新しいオブジェクトを作成します。これがデフォルトです。
- `copy` ー 変換内の要素を実行する前に、ターゲット・オブジェクトとして使用するソース・オブジェクトのコピーを作成します。
- `existing` ー ターゲット・オブジェクトとしてデータ変換の呼び出し側により指定される既存のオブジェクトを使用します。

DTL <true>

<if> 要素の条件が真の場合に、一連のアクティビティを実行します。

構文

```
<if condition="1">
   <true>
      ...
   </true>
   <false>
      ...
   </false>
</if>
```

属性

なし

要素

要素	目的
<annotation>	オプション。<true> 要素について記述するテキスト文字列。
ほとんどのアクティビティ	オプション。<true> には、<assign>、<code>、<foreach>、<if>、<sql>、<subtransform>、および <trace> をゼロ個以上、自由に組み合わせて使用できます。

説明

<true> 要素は、<if> 内で、条件が真の場合に実行する必要のある要素を含めるために使用されます。