



BPL プロセスの開発

Version 2023.1
2024-01-02

BPL プロセスの開発

InterSystems IRIS Data Platform Version 2023.1 2024-01-02

Copyright © 2024 InterSystems Corporation

All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, および TrakCare は、InterSystems Corporation の登録商標です。HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, および InterSystems TotalView™ For Asset Management は、InterSystems Corporation の商標です。TrakCare は、オーストラリアおよび EU における登録商標です。

ここで使われている他の全てのブランドまたは製品名は、各社および各組織の商標または登録商標です。

このドキュメントは、インターシステムズ社(住所: One Memorial Drive, Cambridge, MA 02142)あるいはその子会社が所有する企業秘密および秘密情報を含んでおり、インターシステムズ社の製品を稼働および維持するためにのみ提供される。この発行物のいかなる部分も他の目的のために使用してはならない。また、インターシステムズ社の書面による事前の同意がない限り、本発行物を、いかなる形式、いかなる手段で、その全てまたは一部を、再発行、複製、開示、送付、検索可能なシステムへの保存、あるいは人またはコンピュータ言語への翻訳はしてはならない。

かかるプログラムと関連ドキュメントについて書かれているインターシステムズ社の標準ライセンス契約に記載されている範囲を除き、ここに記載された本ドキュメントとソフトウェアプログラムの複製、使用、廃棄は禁じられている。インターシステムズ社は、ソフトウェアライセンス契約に記載されている事項以外にかかるソフトウェアプログラムに関する説明と保証をするものではない。さらに、かかるソフトウェアに関する、あるいはかかるソフトウェアの使用から起こるいかなる損失、損害に対するインターシステムズ社の責任は、ソフトウェアライセンス契約にある事項に制限される。

前述は、そのコンピュータソフトウェアの使用およびそれによって起こるインターシステムズ社の責任の範囲、制限に関する一般的な概略である。完全な参照情報は、インターシステムズ社の標準ライセンス契約に記載され、そのコピーは要望によって入手することができる。

インターシステムズ社は、本ドキュメントにある誤りに対する責任を放棄する。また、インターシステムズ社は、独自の裁量にて事前通知なしに、本ドキュメントに記載された製品および実行に対する代替と変更を行う権利を有する。

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

InterSystems Worldwide Response Center (WRC)

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: support@InterSystems.com

目次

1 BPL プロセスについて	1
1.1 ビジネス・プロセス・ウィザードの使用	1
1.2 ビジネス・プロセス・リスト	2
1.3 BPL 機能	3
1.4 ビジネス・プロセスのコンポーネントとしての使用法	3
1.5 ビジネス・プロセスの実行コンテキスト	4
1.5.1 context オブジェクト	5
1.5.2 request オブジェクト	5
1.5.3 response オブジェクト	5
1.5.4 callrequest オブジェクト	5
1.5.5 callresponse オブジェクト	6
1.5.6 syncresponses コレクション	6
1.5.7 synctimedout 値	6
1.5.8 status 値	6
1.5.9 process オブジェクト	7
1.6 BPL ビジネス・プロセスの例	8
2 ビジネス・プロセス・デザイナの使用	11
2.1 BPL デザイナ・ツールバー	11
2.2 BPL ダイアグラム	12
2.2.1 BPL ダイアグラムのシェイプ	14
2.2.2 BPL ダイアグラムでの接続	16
2.2.3 BPL ダイアグラムのレイアウト	17
2.2.4 BPL ダイアグラムのドリル・ダウン	17
2.3 BPL ダイアグラムへのアクティビティの追加	19
2.3.1 コール・アクティビティの追加	21
2.4 BPL デザイナのプロパティ・タブ	21
2.4.1 BPL ビジネス・プロセスの一般プロパティの設定	22
2.4.2 context オブジェクトの定義	22
2.4.3 BPL ダイアグラムのプリファレンスの設定	23
2.5 スタジオでの BPL 作成に関する注意事項	24
3 構文ルール	27
3.1 メッセージ・プロパティへの参照	27
3.2 リテラル値	27
3.2.1 XML 予約文字	28
3.2.2 仮想ドキュメント内のセパレータ文字	28
3.2.3 XML 予約文字がセパレータでもある場合	28
3.2.4 数字のコード	29
3.3 有効な式	29
3.4 間接指定	29
4 BPL 要素のリスト	31
4.1 ビジネス・プロセス	31
4.2 実行コンテキスト	31
4.3 制御フロー	32
4.4 メッセージング	32
4.5 スケジューリング	33
4.6 ルールおよび決定事項	33

4.7 データ操作	33
4.8 ユーザ記述コード	34
4.9 ロギング	34
4.10 エラー処理	34
5 BPL のエラー処理	37
5.1 システム・エラー — エラー処理なし	37
5.1.1 イベント・ログ・エントリ	38
5.1.2 この BPL 用の XData	38
5.2 システム・エラー — Catchall で対応	39
5.2.1 イベント・ログ・エントリ	41
5.2.2 この BPL 用の XData	42
5.3 フォールトをスロー — Catchall で対応	42
5.3.1 イベント・ログ・エントリ	44
5.3.2 この BPL 用の XData	45
5.4 フォールトをスロー — Catch で対応	45
5.4.1 イベント・ログ・エントリ	47
5.4.2 この BPL 用の XData	47
5.5 ネスト構造のスコープ — 内側のフォールト・ハンドラに Catchall を配置	48
5.5.1 イベント・ログ・エントリ	50
5.5.2 この BPL 用の XData	50
5.6 ネスト構造のスコープ — 外側のフォールト・ハンドラに Catchall を配置	51
5.6.1 イベント・ログ・エントリ	54
5.6.2 この BPL 用の XData	54
5.7 ネスト構造のスコープ — どのスコープでも一致しない場合	54
5.7.1 イベント・ログ・エントリ	56
5.7.2 この BPL 用の XData	57
5.8 ネスト構造のスコープ — 外側のフォールト・ハンドラに Catch を配置	58
5.8.1 イベント・ログ・エントリ	59
5.8.2 この BPL 用の XData	60
5.9 フォールトをスロー — 補償ハンドラで対応	60
5.9.1 イベント・ログ・エントリ	62
5.9.2 この BPL 用の XData	63

1

BPL プロセスについて

ビジネス・プロセス言語 (BPL) は、標準の XML ドキュメント内で実行可能なビジネス・プロセスを記述するために使用する言語です。BPL の構文は、ビジネス・プロセス・ロジックを定義するための推奨 XML 標準の複数の項目に基づいています。BPL ビジネス・プロセス・クラスは、`Ens.BusinessProcessBPL` から派生します。BPL ビジネス・プロセス・クラスは、BPL をサポートするという点を除いて、`Ens.BusinessProcess` から派生したクラスとすべての点で同じです。

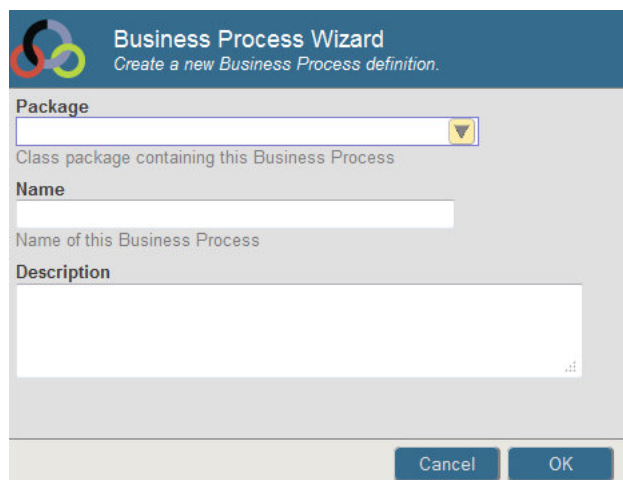
ビジネス・プロセス・デザイナーを使用して、管理ポータルで BPL クラスを作成します。ビジネス・プロセス・デザイナーにアクセスするには、[Interoperability]→[ビルド]→[ビジネス・プロセス] を選択します。(スタジオで BPL クラスを開くと、ビジネス・プロセス・デザイナーが起動します。“スタジオでの BPL 作成に関する注意事項”を参照してください)。

- ・ [新規作成] をクリックして、ビジネス・プロセス・ウィザードを使用して BPL ビジネス・プロセスを作成します。
- ・ [開く] をクリックして、ビジネス・プロセス・デザイナーを使用して既存の BPL ビジネス・プロセスを編集します。

ビジネス・プロセス、データ変換、およびビジネス・ルールで使用可能なオプションが重複していることに注意してください。違いを確認する場合は、“プロダクションの開発”の“ビジネス・ロジック・ツールの比較”を参照してください。

1.1 ビジネス・プロセス・ウィザードの使用

[ビジネス・プロセス・ウィザード] を使用すると、`Ens.BusinessProcessBPL` から継承した BPL ビジネス・プロセス・クラスをすばやく作成できます。このウィザードで表示される次のダイアログでは、作成する BPL ビジネス・プロセスの予備的特性を定義できます。



The image shows a 'Business Process Wizard' dialog box. At the top, it has a logo with three interlocking circles (red, green, blue) and the text 'Business Process Wizard' and 'Create a new Business Process definition.' Below this, there are four input fields: 'Package' (a dropdown menu), 'Name' (a text box), 'Description' (a text box), and 'Class package containing this Business Process' (a text box). At the bottom right, there are two buttons: 'Cancel' and 'OK'.

以下のフィールドの値を入力します。

[パッケージ]

このビジネス・プロセス・クラスを格納するパッケージの名前を入力するか、ネームスペース内のパッケージのリストから選択します。

[名前]

この BPL ビジネス・プロセス・クラスの名前を入力します。

[説明]

(オプション) データ変換の説明を入力します。これはクラスの説明として使用されます。

[OK] をクリックしてウィザードを完了すると、BPL ダイアグラムの開始点と終了点が **ビジネス・プロセス・デザイナー** に表示され、その BPL ビジネス・プロセスにアクティビティを追加できる状態になります。

1.2 ビジネス・プロセス・リスト

[**ビジネス・プロセス・リスト**] ページには、アクティブな相互運用対応ネームスペースで定義されているビジネス・プロセス・クラスのリストが表示されます。管理ポータルでこのページに移動するには、[**Interoperability**]→[**リスト**]→[**ビジネス・プロセス**] を選択します。

BPL ビジネス・プロセスは青色で表示されます。BPL ビジネス・プロセスをダブルクリックすると、その BPL ビジネス・プロセスが [**ビジネス・プロセス・デザイナー**] に表示されます。黒色で表示されるビジネス・プロセスは、スタジオで編集する必要のあるカスタム・クラスです。

ビジネス・プロセス・クラスを選択することで、そのビジネス・プロセス・クラスをリボン・バーにある以下のコマンドのいずれかのターゲットにできます。

- ・ **[新規作成]** – このトピック内で前述したように、ビジネス・プロセス・ウィザードが起動します。
- ・ **[開く]** (BPL クラスのみ) – 選択したビジネス・プロセスを編集します。
- ・ **[エクスポート]** – 選択したビジネス・プロセス・クラスを XML ファイルにエクスポートします。
- ・ **[インポート]** – XML ファイルにエクスポートされたビジネス・プロセスをインポートします。
- ・ **[削除]** – 選択したビジネス・プロセス・クラスを削除します。
- ・ **[インスタンス]** – 実行中のプロダクション内のビジネス・プロセスの現在のインスタンスを一覧表示します。ビジネス・プロセスがその処理を完了している場合は、このページにはエントリは表示されません。
“**プロダクションの監視**” を参照してください。
- ・ **[ルール・ログ]** – このビジネス・プロセスによって呼び出されたルールのビジネス・ルール・ログを表示します。
“**プロダクションの監視**” を参照してください。

InterSystems IRIS の他のクラスと同じように、ビジネス・プロセス・クラスをエクスポートおよびインポートすることもできます。管理ポータルの [**クラス**] ページを使用できます。アクセスするには、[**システムエクスプローラ**]→[**クラス**] を選択します。スタジオの [**ツール**] メニューにある [**エクスポート**] コマンドと [**インポート**] コマンドを使用することもできます。

1.3 BPL 機能

BPL は、標準の XML ドキュメント内で実行可能なビジネス・プロセスを記述するために使用する言語です。BPL 構文は、Web サービス向けビジネス・プロセス実行言語 (BPEL4WS または BPEL)、ビジネス・プロセス管理言語 (BPML または BPMI) など、ビジネス・プロセス・ロジックの定義を目的とする複数の推奨 XML 標準に基づいています。

BPL は、その他に指定されている XML ベースの標準のスーパーセットで、統合ソリューション構築の補助を目的とした追加要素を提供します。これらの追加要素でサポートされているのは以下のとおりです。

- ・ `<branch>`、`<if>`、`<switch>`、`<foreach>`、`<while>`、`<until>` など、実行フロー・コントロール要素。これらの要素やその他の BPL 構文要素の使用の詳細は、“[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](#)”を参照してください。
- ・ ビジネス・プロセス・ロジックからの実行可能コードの生成。
- ・ ビジネス・プロセス・ロジックへの SQL およびカスタム記述コードの埋め込み。
- ・ **ビジネス・プロセス・デザイナー** (ビジネス・プロセス・ロジックをグラフィカルに表示および編集するためのフル機能を備えたビジュアル・モデリング・ツール)。このツールは、ビジネス・プロセスのビジュアル表示および BPL 表示間での完全なラウンドトリップ・エンジニアリングを備えています。一方の表示で変更を行うと、変更内容が自動的に他方に反映されます。
- ・ ビジネス・プロセスと統合ソリューションのその他のメンバ間での、非同期および同期メッセージング両方の自動サポート。BPL を使用すると、このようなエラーの発生しやすい難しいタスクを合理化できます。
- ・ 永続的な状態。BPL では、長時間実行中のビジネス・プロセスが非アクティブであり、非同期の応答を待機しているような場合に、そのビジネス・プロセスに実行の自動中断を許可し、組み込み型の永続キャッシュに実行状態を効果的に保存できるようにします。InterSystems IRIS では自動的に、保存されているすべての状態が管理され、処理の再開が適切に管理されます。
- ・ リッチで多様性に富むデータ変換サービス。これには、ビジネス・プロセス内に埋め込まれた SQL クエリも含まれます。

BPL ビジネス・プロセスは、管理ポータルまたはスタジオを使用して作成できます。推奨される方法は、管理ポータルの **[ビジネス・プロセス・デザイナー]** ページから **ビジネス・プロセス・ウィザード** を使用することです。詳細は、このドキュメントの以降のトピックを参照してください。

1.4 ビジネス・プロセスのコンポーネントとしての使用法

ビジネス・プロセス・コンポーネントまたは BPL コンポーネントは、プログラマが BPL 言語のモジュール式の再利用可能な一連の手順として指定する BPL ビジネス・プロセスです。BPL コンポーネントは、他のプログラミング言語における関数、マクロ、サブルーチンと似ています。

別の BPL ビジネス・プロセスのみが BPL コンポーネントを呼び出すことができます。これは、BPL の `<call>` 要素を使用して呼び出します。BPL ビジネス・プロセス・コンポーネントは、タスクを実行すると、呼び出された BPL ビジネス・プロセスに制御を戻します。

プロダクション・アーキテクチャでは、ある BPL ビジネス・プロセスを使用して別の BPL ビジネス・プロセスを呼び出せるようになっています。オプションのコンポーネントを指定することにより、さらに便利になります。これにより、特定の BPL ビジネス・プロセスを、以下のようなより単純な下位レベルのコンポーネントに分類できます。

- ・ スタンドアロンのビジネス・プロセスとして実行することを目的としないコンポーネント (ただし、このアーキテクチャでその妨げになるものではありません)

- ・ (BPL 言語の関数、マクロ、サブルーチンという意味での) 再利用可能コンポーネント

コンポーネントでないビジネス・プロセスは、より複雑で特殊な目的で設計され、コンポーネントより高い概念レベルで動作するものと想定されます。BPL のコンポーネント以外は、BPL コンポーネントを呼び出してタスクを実行することが想定されます。

重要 任意の BPL ビジネス・プロセスに対して、コンポーネントの指定を使用するための要件はありません。BPL プログラマは、目的に合わせて適宜利用できます。

ビジネス・プロセスをコンポーネントに変換するには、その BPL ビジネス・プロセスに対して最上位レベルの `<process>` コンテナの属性を設定します。この属性は `component` という名前であり、その値は 1 (真) または 0 (偽) に設定できます。構文の詳細は、“[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](#)” を参照してください。

`component` 属性の値を設定するには、以下のいずれかを実行します。

- ・ **ビジネス・プロセス・デザイナー** の [一般] タブで、[コンポーネント] を選択して、このプロセスをコンポーネント・ライブラリに組み込みます。
- ・ スタジオを使用して、クラス・コードの XData BPL ブロック内の BPL `<process>` 要素を編集します。

BPL ビジネス・プロセスからコンポーネントに対する `<call>` をセットアップするには、後述する“[コール・アクティビティの追加](#)”を参照してください。

1.5 ビジネス・プロセスの実行コンテキスト

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。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。

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。BPL ビジネス・プロセスの実行が中断および再開されるたびに、実行コンテキスト内の変数が自動的に保存およびリストアされます。これらの変数は、すべての BPL ビジネス・プロセス、つまり、**Ens.BusinessProcessBPL** から継承したすべてのビジネス・プロセス・クラスに使用できます。

重要 **Ens.BusinessProcess** から継承したカスタム・ビジネス・プロセスは、組み込みの実行コンテキストにアクセスできないため、カスタム・コードを使用して同様の処理を実行する必要があります。

実行コンテキスト変数には、BPL ビジネス・プロセス内のすべてのアクティビティで利用可能なものがあります。その他の変数は通常利用可能ですが、ビジネス・プロセスが実行しているアクティビティのタイプによっては、スコープ内に含まれる場合もあれば、含まれない場合もあります。以下のトピックでは、実行コンテキスト変数と、その変数が BPL ビジネス・プロセスで使用される状況について説明します。変数には、以下のものがあります。

- ・ `context`
- ・ `request`
- ・ `response`
- ・ `callrequest`
- ・ `callresponse`
- ・ `syncresponses`
- ・ `synctimedout`
- ・ `status`

・ process

Tip ヒン ト [「<process>、<context>、<call>」などの BPL 要素の BPL 構文に関する詳細は、「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」を参照してください。](#)これには、コンテキスト変数の参照情報も記載されています。

1.5.1 context オブジェクト

context オブジェクトは、[<process>](#) 要素内の任意の箇所で BPL ビジネス・プロセスによって使用可能です。context は、ビジネス・プロセスのライフ・サイクル中に永続化される必要のある任意のデータ用の汎用コンテナです。BPL ビジネス・プロセスの作成時に、各データ項目を context オブジェクト上のプロパティとして定義します。推奨手順は、“[context オブジェクトの定義](#)”を参照してください。

context オブジェクト上のプロパティを定義したら、通常のドット構文と対象のプロパティ名を使用して、BPL 内のどこからでもそれらのプロパティを参照できます (例: `context.MyData`)。

1.5.2 request オブジェクト

request オブジェクトには、元の要求メッセージ・オブジェクトに含まれていたプロパティが格納されています。元の要求メッセージ・オブジェクトとは、このビジネス・プロセスを最初にインスタンス化させた受信メッセージです。これは基本要素と呼ばれます。

request オブジェクトは、[<process>](#) 要素内であれば、BPL ビジネス・プロセスによって使用できます。request オブジェクトのプロパティを参照するには、`request.OriginalThought` のようにドット構文とプロパティ名を使用します。

1.5.3 response オブジェクト

response オブジェクトには、ビジネス・プロセス・インスタンスが返す最終的な応答メッセージ・オブジェクトを構築するのに必要なプロパティが含まれます。ビジネス・プロセスは、ライフ・サイクルの最後に達したとき、または [<reply>](#) アクティビティを検出したときに、この最終的な応答を返します。

response オブジェクトは、[<process>](#) 要素内であれば、BPL ビジネス・プロセスによって使用できます。response オブジェクトのプロパティを参照するには、`response.BottomLine` のようにドット構文とパラメータ名を使用します。

1.5.4 callrequest オブジェクト

callrequest オブジェクトには、[<call>](#) で送信する要求メッセージ・オブジェクトを作成する際に必要となるすべてのプロパティが含まれます。

[<call>](#) アクティビティは、要求メッセージを送信し、必要に応じて応答を受信します。BPL [<call>](#) 要素には、要求メッセージ・オブジェクトのプロパティに値を挿入する [<request>](#) アクティビティが含まれている必要があります。そのため、[<request>](#) には、callrequest オブジェクトのプロパティに値を挿入する一連の [<assign>](#) アクティビティが用意されています。これらの値には、通常、元の request オブジェクトのプロパティ値が使用されますが、任意の値を割り当てることができます。

[<request>](#) 内の [<assign>](#) アクティビティが完了すると、すぐにメッセージが送信され、関連付けられた callrequest オブジェクトはスコープから外れます。callrequest は、関連付けられた [<request>](#) アクティビティの外部では意味を持たないため、関連付けられた [<call>](#) が次のアクティビティであるオプションの [<response>](#) の処理を開始するときには、既にスコープから外れています。

関連する [<request>](#) 要素のスコープ内であれば、`callrequest.UserData` のようにドット構文を使用して、callrequest のプロパティを参照できます。

1.5.5 callresponse オブジェクト

〈call〉アクティビティが完了すると、callresponse オブジェクトには、〈call〉に返された応答メッセージ・オブジェクトのプロパティが追加されます。〈call〉が応答を待たないアクティビティとして設計されていた場合、callresponse はありません。同様に、〈sync〉を使用して応答を待っていても、〈sync〉要素で指定されたタイムアウト期間内に応答が返ってこなかった場合は、callresponse が存在しません。

応答を待つ〈call〉はすべて、〈call〉内に〈response〉アクティビティが存在している必要があります。〈response〉アクティビティの目的は、応答値を取得し、それをビジネス・プロセス全体で使えるようにすることです。callresponse オブジェクトは、〈response〉アクティビティ内であればどこでも使用できます。ただし、〈response〉アクティビティが完了した時点で、関連する callresponse オブジェクトはスコープから外れます。そのため、ビジネス・プロセスの他の場所で callresponse の値を使用する場合は、その値に対して context オブジェクトまたは response オブジェクトのプロパティへの〈assign〉を実行し、値を受信した〈response〉アクティビティが完了するまでに、この処理を実行する必要があります。

callresponse のプロパティを参照するには、callresponse.UserAnswer のようにドット構文を使用します。

1.5.6 syncresponses コレクション

syncresponses は、〈sync〉によって同期化された〈call〉アクティビティの名前をキーとするコレクションです。

〈sync〉アクティビティが開始されると、新しい応答に備えて、syncresponses が消去されます。〈call〉アクティビティが返されると、応答がコレクションに格納されます。〈sync〉アクティビティが完了すると、syncresponses には、必要な応答のすべてまたは一部が格納されるか、何も格納されません (synctimedout を参照してください)。syncresponses は、関連する〈call〉および〈sync〉アクティビティが含まれた〈sequence〉の内部であれば使用できますが、〈sequence〉の外部ではスコープから外れます。

同期化された呼び出しの 1 つから応答値を参照するには、構文 syncresponses.GetAt ("name") を使用します。

ここで、関連する〈call〉は <call name="name"> と定義されます。

1.5.7 synctimedout 値

synctimedout は 0、1、または 2 の値を取る整数値です。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。

synctimedout は、関連する〈call〉アクティビティと〈sync〉アクティビティが含まれた〈sequence〉の内部であれば、BPL ビジネス・プロセスに使用できますが、〈sequence〉の外部では、スコープから外れます。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。synctimedout を参照するには、synctimedout のように、整数の変数名と同じ構文を使用します。

1.5.8 status 値

status は、成功または失敗を示す %Status 型の値です。

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。

`status` では、以下のいずれかの方法で失敗値を取得できます。

- ・ `status` は、ビジネス・プロセスが別のビジネス・ホストに対して実行した任意の `<call>` から返された `%Status` 値を自動的に受け取ります。この `%Status` の値が失敗を示す場合、`status` には自動的にその失敗値が入ります。これは、`status` を設定する最も一般的な方法で、BPL コードで特別な文を使用しなくても、自動的に処理されます。
- ・ `<assign>` アクティビティで、`status` を失敗値に設定できます。そのために、一般的には `<if>` 要素を使用して前のアクティビティの結果をテストし、`<true>` 要素または `<false>` 要素の中で `<assign>` を使用して失敗条件が存在すれば `status` を失敗値に設定します。
- ・ `<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。

`status` に失敗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 `$$$ISERR(status)` を使用します。このテストで `True` 値が返された場合、`status` には失敗値が格納されています。このテストは、メイン BPL ビジネス・プロセスに戻る前に `<code>` アクティビティの本体内でのみ実行できます。ビジネス・プロセスは任意の `<call>`、`<assign>`、または `<code>` アクティビティの後に `status` で失敗値を取得したことを検出すると、自動的にエラーで終了してしまいます。

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。

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、この予約語を使用しないでください。

1.5.9 process オブジェクト

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を表します。process オブジェクトの目的は、BPL ビジネス・プロセスのフロー内の任意のコンテキストから（例えば、`<code>` アクティビティのテキスト・ブロック内から）、`SendRequestSync()` や `SendRequestAsync()` などの任意のビジネス・プロセス・メソッドを呼び出せるようにすることです。

process オブジェクトは、`<process>` 要素内であれば、BPL ビジネス・プロセスで使用できますが、一般的には `<code>` アクティビティ内でのみ必要になります。process オブジェクトのメソッドを参照するには、`process.SendRequestSync()` や `process.ClearAllPendingResponses` のようにドット構文とメソッド名を使用します。

1.6 BPL ビジネス・プロセスの例

以下のビジネス・プロセスでは、3 つの異なる銀行にプライム・レートと信用承認情報を相談できます。

Class Definition

```

/// Loan Approval Business Process for Bank Soprano.
/// Bank Soprano simulates a bank with great service but
/// somewhat high interest rates.
Class Demo.Loan.BankSoprano Extends Ens.BusinessProcessBPL
{
  XData BPL
  {
    <process request="Demo.Loan.Msg.Application"
      response="Demo.Loan.Msg.Approval">

      <context>
        <property name="CreditRating" type="%Integer"/>
        <property name="PrimeRate" type="%Numeric"/>
      </context>

      <sequence>

        <trace value='"received application for " + _request.Name' />

        <assign name='Init Response'
          property="response.BankName"
          value='"BankSoprano"'>
          <annotation>
            Initialize the response object.

          </annotation>
        </assign>

        <call name="PrimeRate"
          target="Demo.Loan.WebOperations"
          async="1">
          <annotation>
            Send an asynchronous request for the Prime Rate.

          </annotation>
          <request type="Demo.Loan.Msg.PrimeRateRequest"/>
          <response type="Demo.Loan.Msg.PrimeRateResponse">
            <assign property="context.PrimeRate"
              value="callresponse.PrimeRate"/>
          </response>
        </call>

        <call name="CreditRating"
          target="Demo.Loan.WebOperations"
          async="1">
          <annotation>
            Send an asynchronous request for the Credit Rating.

          </annotation>
          <request type="Demo.Loan.Msg.CreditRatingRequest">
            <assign property="callrequest.TaxID" value='request.TaxID'/>
          </request>
          <response type="Demo.Loan.Msg.CreditRatingResponse">
            <assign property="context.CreditRating"
              value="callresponse.CreditRating"/>
          </response>
        </call>

        <sync name='Wait'
          calls="PrimeRate,CreditRating"
          type="all"
          timeout="10">
          <annotation>
            Wait for the response from the async requests.
            Wait for up to 10 seconds.

          </annotation>
        </sync>
      </process>
    </XData>
  }
}

```

```

<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
  name='Delay'
  duration="2+($zcrc(request.Name,4)#5)">
  <annotation>
    Wait for a random duration.

  </annotation>
</delay>

<trace value='"application is "
  _$$s(response.IsApproved:"approved for "_response.InterestRate_"",
  1:"denied")' />

</sequence>
</process>
}
}

```


2

ビジネス・プロセス・デザイナの使用

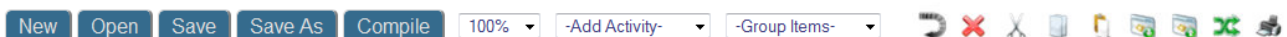
ここでは、ビジネス・プロセス・デザイナの使用法について説明します。ビジネス・プロセス・デザイナを使用すると、BPL ビジネス・プロセスをビジュアル・ダイアグラムとして作成できます。ビジネス・プロセス・デザイナでダイアグラムを保存すると、正しい BPL 構文で記述されたクラスの説明 (テキスト・ドキュメント) が生成されます。BPL ダイアグラムおよび BPL ドキュメントは、同じ BPL ビジネス・プロセス・クラスの等しく有効な説明です。

管理ポータルまたはスタジオで BPL ビジネス・プロセスを開くと、またはウィザードを使用して新しい BPL ビジネス・プロセスを作成すると、その BPL ダイアグラムがビジネス・プロセス・デザイナに表示されます。このダイアグラムの右側には、一連のプロパティ・タブを含むペインがあります。二重矢印のアイコンをクリックすることで、この右側のペインを必要に応じて展開および縮小できます。

重要 非アクティブ状態が続くと、インターシステムズの管理ポータルによってログアウトされ、保存されていない変更が破棄される可能性があります。非アクティブ状態の時間は、InterSystems IRIS サーバの呼び出しから次の呼び出しまでです。すべてのアクションがサーバの呼び出しを引き起こすわけではありません。例えば、[保存] をクリックした場合はサーバが呼び出されますが、テキスト・フィールドに入力した場合はサーバの呼び出しは生じません。このため、ビジネス・プロセスを編集していて、[セッションタイムアウト] のしきい値より長い間 [保存] をクリックしないと、セッションは期限切れとなり、保存されていない変更は破棄されます。ログアウトされた後、ログイン・ページが表示されるか、現在のページが更新されます。詳細は、“[管理ポータルの自動ログアウト動作](#)” を参照してください。










2.1 BPL デザイナ・ツールバー

[ビジネス・プロセス・デザイナ] ページのリボン・バーには、BPL デザイナ・ツールバーを構成する各種のオプションとコマンドが含まれています。



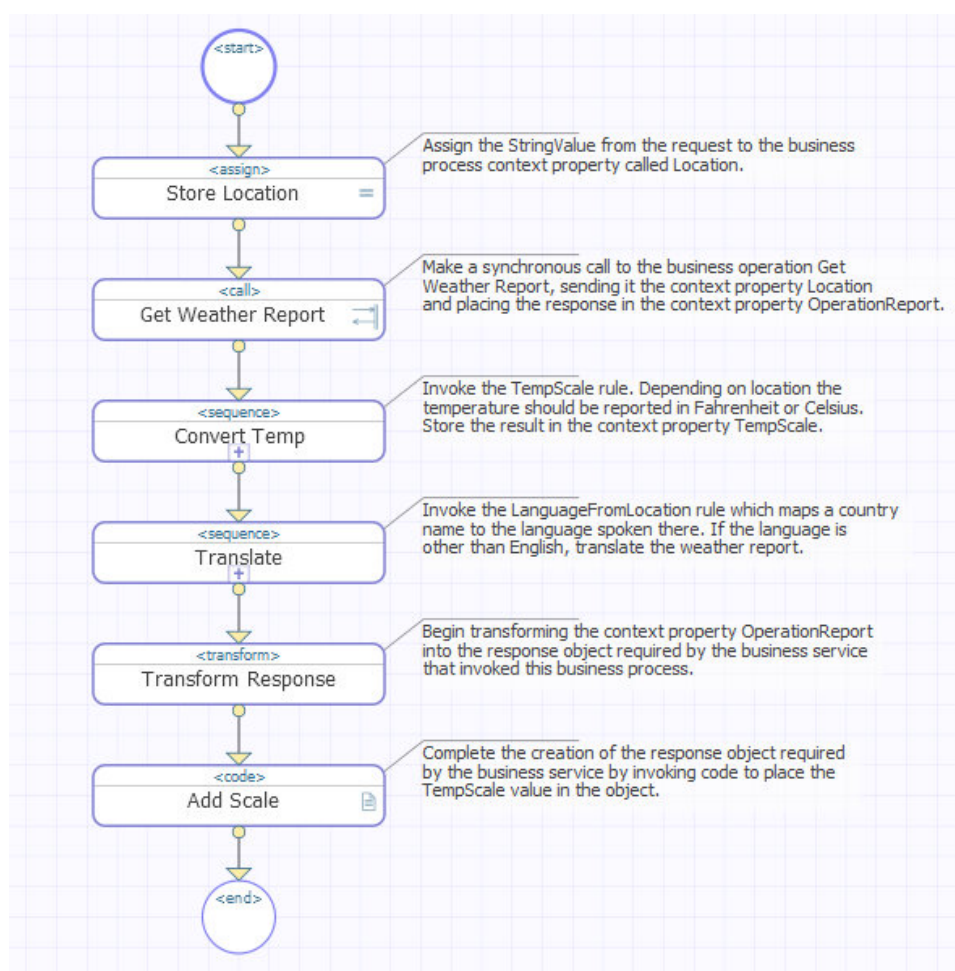
対象のダイアグラムに対して有効ではないコマンドは淡色表示されます。以下のテーブルでは、各コマンドで実行するアクションを説明しており、一部のコマンドについては、そのプロセスの詳細な説明へのリンクを示しています。

コマンド	説明
[新規作成]	ビジネス・プロセス・ウィザード を起動して、新しい BPL ビジネス・プロセスを作成します。
[開く]	[ファインダダイアログ] を起動してロードする既存の BPL ビジネス・プロセス・クラスを選択し、ビジネス・プロセス・デザイナを使用して編集を開始します。
[保存]	ビジネス・プロセス・ダイアグラムに加えた変更内容を保存します。

コマンド	説明
[名前を付けて保存]	変更内容を新しい BPL ビジネス・プロセス・クラスとして保存します。
[コンパイル]	BPL ビジネス・プロセス・クラスをコンパイルします。
100%	BPL ダイアグラムのサイズを縮小または拡大するためのパーセンテージ値のリストから選択します。詳細を表示するには大きな拡大率を、全体を表示するには小さな拡大率を選択します。
[アクティビティ追加]	“ BPL ダイアグラムへのアクティビティの追加 ” の節の説明に従って、BPL ダイアグラムに追加するアクティビティのリストから BPL 要素を選択します。
項目のグループ化	選択した項目をグループ化して、グループ・ダイアグラムを構成します。このグループ・ダイアグラム内で、詳細をドリル・ダウンできます。このグループは、上位レベル・ダイアグラム上の 1 つのシェイプとして表現されます。選択した要素を <sequence> としてグループ化するか、特定タイプのループ (<foreach>、<while>、または <until>) としてグループ化するか、または <scope> 要素や <flow> 要素としてグループ化するかを選択できます。
	直前のアクション（アクティビティの追加、移動、編集など）を取り消します。
	選択した項目をダイアグラムから削除します。一度に選択できるアクティビティは 1 つで、アクティビティをクリックして選択します。選択されたアクティビティは色が黄色になります。
	選択した項目を切り取って、BPL クリップボードに格納します。
	選択した項目を BPL クリップボードにコピーします。
	BPL クリップボード内の項目を選択した箇所に貼り付けます。
	グループ化したアクティビティ詳細の BPL ダイアグラムを表示します。これはネストされたダイアグラムに対してドリル・ダウンします。グループを表すシェイプに対してのみ使用できます（そのシェイプの下部にあるプラス記号をクリックすることもできます）。
	現在のグループを単一のシェイプとして上位レベルのダイアグラムに表示します。これはネストされたダイアグラムに対してドリル・アップします。以前にグループにドリル・ダウンしたことがある場合にのみ使用できます。
	ダイアグラム内の項目のレイアウトを整理します。これにより、基盤の BPL ドキュメントが変更されることなく、ダイアグラム内のシェイプが整列されます。
	印刷用のダイアグラムを新しいブラウザ・ページに表示します。

2.2 BPL ダイアグラム

リボン・バーの下側の左側ペインに表示される BPL ダイアグラムは、BPL ファイル内のアクティビティに対応するシェイプと、BPL ファイル内のロジックに対応するその他のシェイプおよび接続で構成されています。BPL ダイアグラムのサンプルを以下に示します。



編集のヒント

ビジネス・プロセス・デザイナーを使用して BPL ダイアグラムを操作する際は、以下のことを実行できます。

- ・ シェイプを選択できます。そのためには、そのシェイプをクリックします。
- ・ 複数のシェイプを選択できます。そのためには、**Ctrl** キーを押しながらそれらのシェイプを選択します。
- ・ 任意の要素を別の要素に接続できます。そのためには、その要素の入力または出力の接続ポイントをクリックして対象の要素までドラッグします。ビジネス・プロセス・デザイナーでは、許可されていない接続を行うことはできません。
- ・ 要素のプロパティを表示または編集できます。そのためには、その要素を選択して、そのプロパティを右側の【**アクティビティ**】タブに表示します。コネクタを選択してそのプロパティを表示することもできます。プロセス自体のプロパティを表示するには、他のプロパティ・タブをクリックします。
- ・ 1 回の操作で新しいシェイプを挿入して接続できます。新しいシェイプの挿入場所となる 2 つの要素の間のコネクタを選択してから、アクティビティを追加します。既存の要素間に新しいシェイプが表示され、接続が自動的に行われます。
- ・ アクティビティをダイアグラムに追加する際に、それらのアクティビティを自動的に検証できます。InterSystems IRIS® によって論理エラーのある要素が検知された場合は、その要素の【**アクティビティ**】タブに赤色の警告とエラーの理由が表示されます。












以下のトピックでは、ダイアグラムについて、およびダイアグラムで各種の BPL 要素がどのように表示されるのかについて詳述しています。

- ・ [BPL ダイアグラムのシェイプ](#)

- ・ BPL ダイアグラムでの接続
- ・ BPL ダイアグラムのレイアウト
- ・ BPL ダイアグラムのドリル・ダウン

2.2.1 BPL ダイアグラムのシェイプ

BPL ダイアグラムでは所定のシェイプを使用して、BPL 要素がコードの中にあることを示します。

BPL シェイプ	意味	例
	アクティビティ *	<assign>、<call>、<sync>、およびその他ほとんど。
	ループ	<foreach>、<while>、または <until>。ループの詳細を表示するには、シェイプの下部にある矢印をクリックするか、  をクリックします。
	シーケンス	<catch>、<catchall>、または <sequence>。シーケンスを表示するには、シェイプの下部にあるプラス記号をクリックするか、シェイプを選択するか、または  をクリックします。
	スコープ	エラー処理用の BPL <scope> の開始。濃色表示された長方形の背景の中にあるのが、この <scope> 内に入るすべての BPL 要素です。<scope> に <faulthandlers> 要素が含まれている場合は、長方形の中央を横切る水平点線が追加されます。この線の下側の領域に、<faulthandlers> の内容が表示されます。例については、“ BPL のエラー処理 ”を参照してください。
	判断	<if>、<switch>、または <branch> の開始。
	特殊	<alert>、<reply>、または <label>。
	分割	1 つのポイントからさまざまな論理パスが分岐する、BPL の <flow> 要素の開始点
	結合	すべての有効なパスが集結する、任意の分岐要素 (<if>、<flow>、<branch>、<scope>、または <switch>) の最後。
	開始/終了	BPL ダイアグラムの始まりまたは終わり。

* 上記の <call> アクティビティ・ボックスの右側部分に表示されているように、多くのアクティビティのシェイプではアイコンが表示されます。以下のテーブルは、これらのアイコンの意味を示しています。

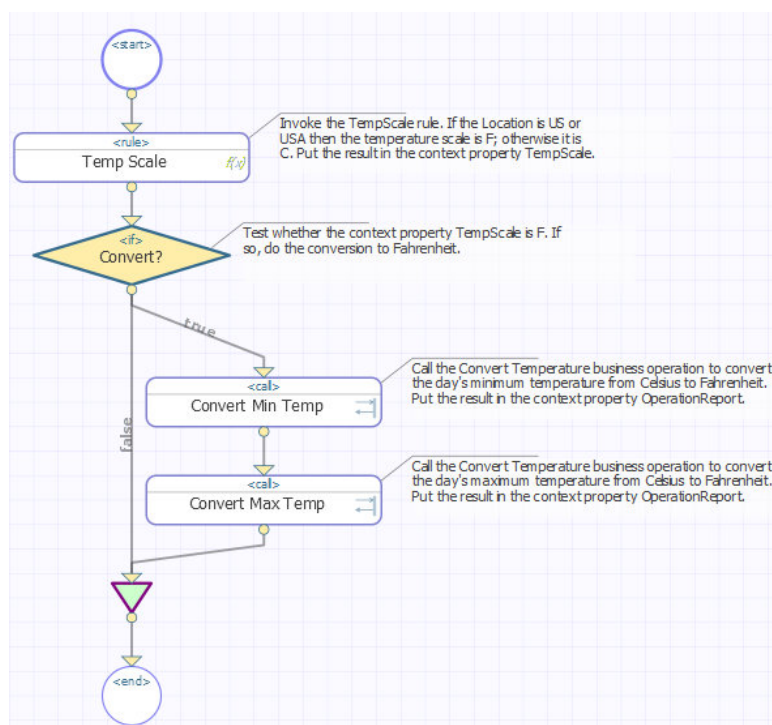
アイコン	BPL 要素	アイコン	BPL 要素	アイコン	BPL 要素
=	<assign>		<code>		<sql>
	非同期の <call>		<delay>		<sync>
	同期の <call>		<milestone>		<trace>
	<catch>	$f(x)$	<rule>		

通常、BPL ダイアグラム・シェイプの内部の色は白で、青いアウトラインで囲まれています。シェイプがエラーの場合は、アウトラインは赤色で表示されます。シェイプが無効の場合は、内部の色はグレーで、グレーのアウトラインで囲まれます。

BPL ダイアグラムでシェイプをクリックすると、そのシェイプが選択されます。そのシェイプの属性は【アクティビティ】タブに表示されて、このタブでこれらの値を編集できます。内部の色は黄色に変わります。シェイプにエラーがある場合、そのシェイプのアウトラインは赤色のままになり、エラーがない場合は、アウトラインはより太い幅の青色に変わります。無効なシェイプを選択すると、点線のアウトラインが表示されます。

複数のシェイプを選択するには、**Ctrl** キーを押しながらシェイプをクリックします。シェイプの選択を解除するには、選択されているシェイプをクリックします。

シェイプが <if> や <switch> などの複雑なアクティビティを表し、そのアクティビティに BPL ダイアグラムの他の場所に関連付けられた分岐や結合などのシェイプが複数含まれている場合は、それらのシェイプの 1 つをクリックすると、関連するシェイプが紫色のアウトラインで囲まれた緑色でハイライト表示されます。<sync> 要素をクリックすると、同期化する <call> 要素がハイライト表示されます。<if> シェイプをクリックすると、<true> と <false> の分岐が交わる結合がハイライト表示されます。その他についても同様です。以下に例を示します。



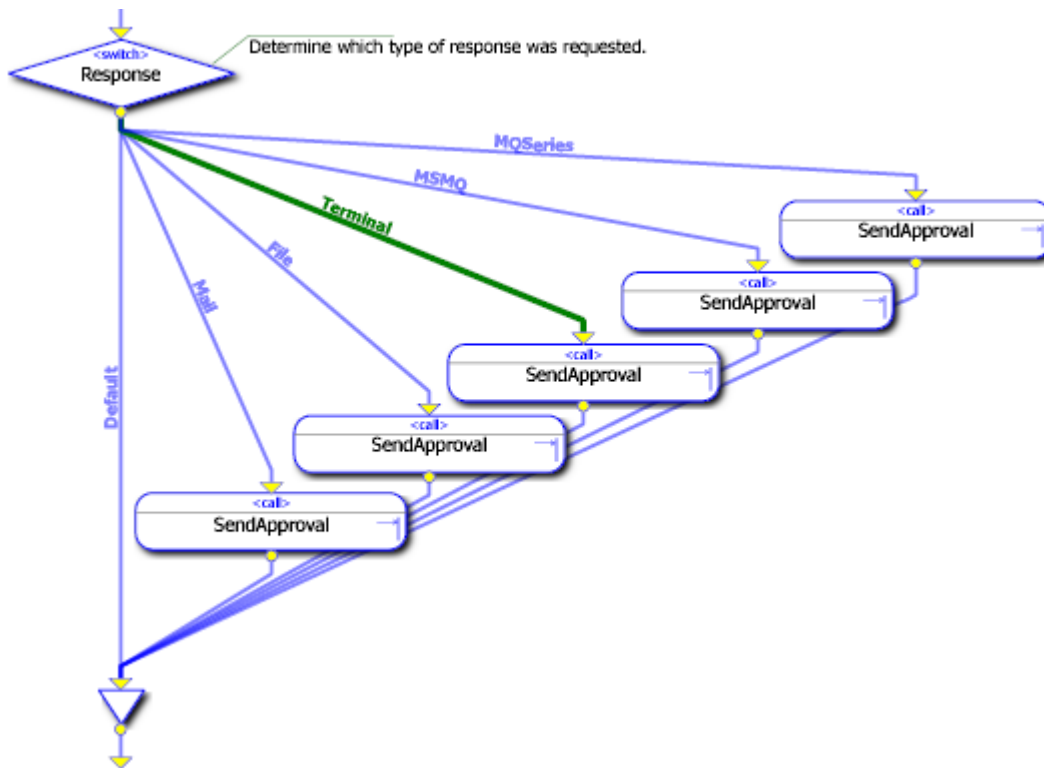
2.2.2 BPL ダイアグラムでの接続

BPL ダイアグラムでは、シェイプ間のラインによって要素間のロジック関係とシーケンスが指定されています。このようなラインは、接続と呼ばれます。各ライン開始点は円形の ナブ に、終了点は三角形の点となっています。BPL ダイアグラムに追加された各シェイプには、三角形の 入力 ナブと円形の 出力 ナブが 1 つずつ組み込まれています。

シェイプの入力または出力ナブをクリックして、希望するシェイプまでカーソルをドラッグすると、あるシェイプを別のシェイプに接続できます。マウスを放すと、接続が表示されます。シェイプを接続する別の方法として、1 つの手順で新しいシェイプの挿入と自動接続を行う方法もあります。新しいシェイプの挿入先の両側にある 2 つの要素を選択します。複数のシェイプを選択するには、**Ctrl** キーを押したままシェイプをクリックします。間に既存の接続がない 2 つの要素を選択した場合、新しいシェイプを追加すると、このシェイプが既存の要素間に表示され、接続は自動的に行われます。接続されている 2 つの要素間に新しいシェイプを追加するには、接続をクリックしてハイライト表示してから、新しい要素を追加します。既存の要素間に新しいシェイプが表示され、自動的に接続が行われます。


2 つのシェイプが接続された後は、それぞれのシェイプをどこにドラッグしても接続状態は保たれます。シェイプは、同じダイアログ内のどのレイアウト位置にでもドラッグできます。接続は自動的にリルートされますが、基盤となる BPL ドキュメントは変更されません。その一方で、接続のロジックを変更した場合は(呼び出し順序の変更、ループの作成、カット・アンド・ペーストなど)、ビジネス・プロセス・デザイナーで実行されたアクションを反映するために、基盤となる BPL ドキュメントが変更されます。

〈switch〉アクティビティ内では、実行可能な各パスに、対応する〈switch〉値のラベルが自動的に付けられます。〈switch〉アクティビティから分岐した実行可能なパスはすべて、1 つの矢印が結合シェイプから BPL ダイアログ内の次のアクティビティに接続される手前で、結合シェイプに収束されます。

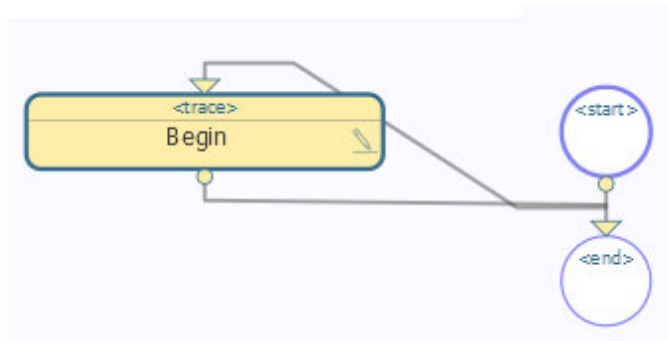



ビジネス・プロセス・デザイナーには、操作中のダイアグラムを対象にしたさまざまなタイプの検証機能があります。例えば、〈if〉、〈flow〉または〈switch〉要素の出力分岐がダイアグラム内の間違った結合シェイプに接続されている場合、エディタがそのエラーを検出する機能があります。その場合は、ダイアグラムを修正するまで、エラーのあるコネクタが赤で表示されます。

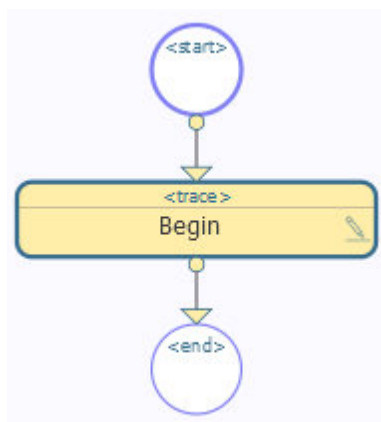
2.2.3 BPL ダイアグラムのレイアウト

シェイプの追加後や新しい接続の作成後に、ダイアグラムの内容を整えるには、ツール・バーの整列アイコン  をクリックします。

例えば、プリファレンスで自動整列機能を有効にしていない場合は、シェイプを BPL ダイアグラムに追加すると、次の図のようになります。



自動整列ツール  をクリックすると、これらのシェイプは次の図のように整列されます。

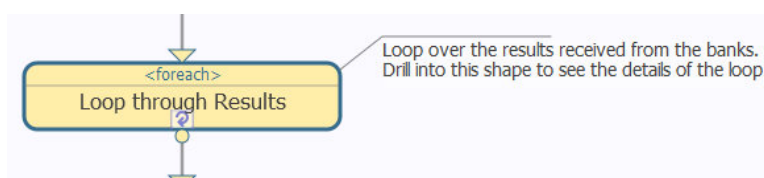




ダイアグラムで常にこのような整列されたレイアウトを使用するには、[プリファレンス] タブの [自動整列] チェックボックスにチェックを付けます。

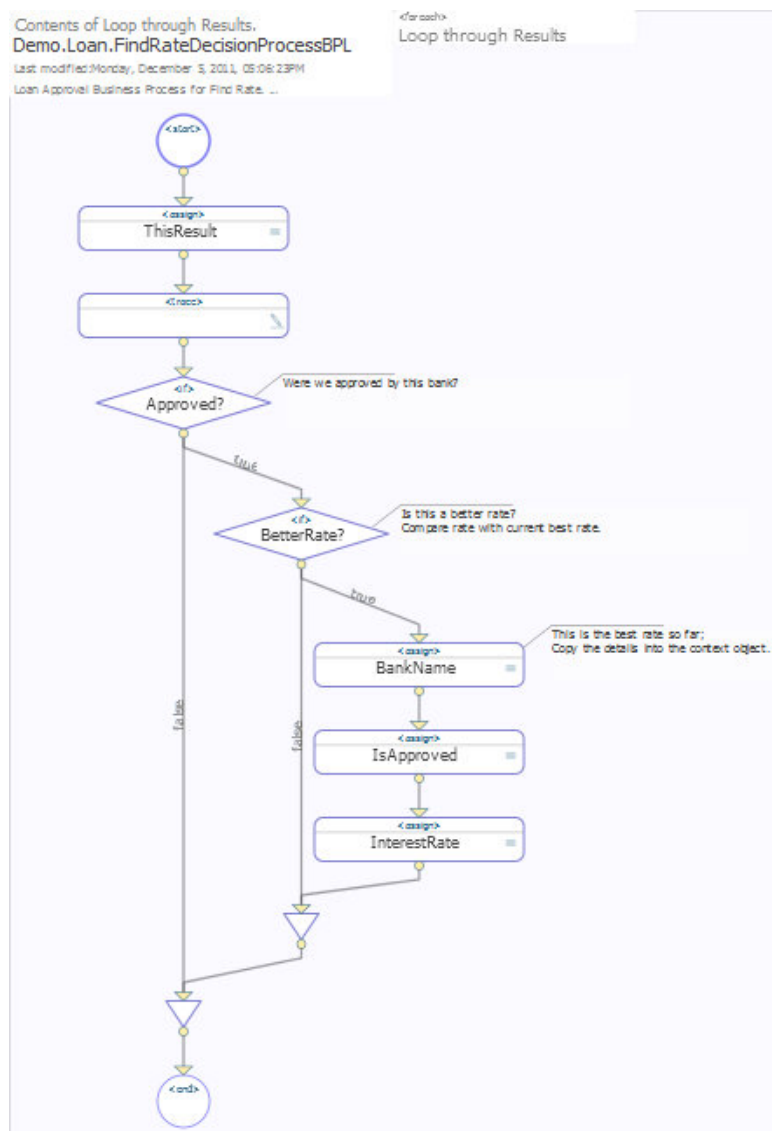
デフォルトでは、ビジネス・プロセス・デザイナーで BPL ダイアグラムを初めて開いたときは、自動整列機能は有効になっています。ただし自動整列機能は、すべてのダイアグラムに対して適しているとは限りません。自動整列機能を無効にして、ダイアグラムを常に希望どおりのレイアウトで表示させるには、[プリファレンス] タブの [自動整列] チェックボックスのチェックを外します。そうすることで、ビジネス・プロセス・デザイナーで表示されるダイアグラムは、指定したとおりのレイアウトで表示されるようになります。


2.2.4 BPL ダイアグラムのドリル・ダウン

ループ・アクティビティは、ドリルダウンの詳細を提供することを示す環状矢印を表示します。〈foreach〉 ループ・アクティビティの例を以下に示します。その他には、〈while〉 および 〈until〉 が含まれます。以下に例を示します。

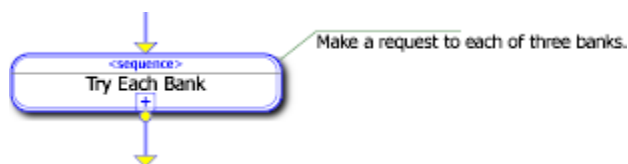



ループ・アクティビティを選択してから、 ツールをクリックするか、ループ・シェイプ内の  をクリックすると、ループの BPL ダイアグラムが表示されます。これは、ループの開始から終了までのすべてのロジックを表示する完全な BPL ダイアグラムです。以下に例を示します。

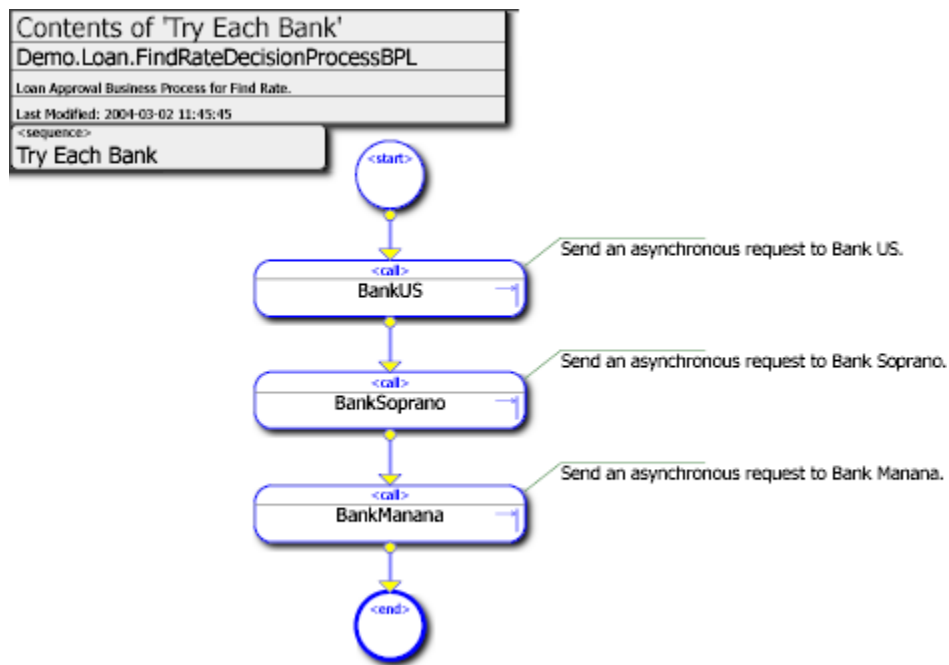


1 つのシェイプがグループ全体を表す上位の論理レベルに戻るには、 ツールをクリックします。

シーケンスも、ドリル・ダウンの詳細を提供できることを示すためにプラス・サインを表示します。以下に例を示します。



シーケンスにドリル・ダウンすると、結果として得られる BPL ダイアグラムでは、そのシーケンスの開始から終了までのすべてのロジックが表示されます。1 つのシェイプがシーケンス全体を表す上位の論理レベルに戻るには、 ツールをクリックします。以下に例を示します。



下位レベルのダイアグラムのどこかにエラーがあると、ビジネス・プロセス・デザイナーでは、グループ・シェイプ (<foreach>、<sequence>、<while>、または <until>) が赤色でハイライト表示されます。このエラーを修正するには、グループ・シェイプにドリル・ダウンして、下位レベルのダイアグラムで赤色でハイライト表示されているエラーがあるアクティビティを確認する必要があります。

2.3 BPL ダイアグラムへのアクティビティの追加

管理ポータルまたはスタジオで BPL ダイアグラムを開いているときは常に、ビジネス・プロセス・デザイナーのツールバーに [アクティビティ追加] リストが表示されます。このリストで項目をクリックすると、その項目のシェイプが BPL ダイアグラムに追加されます。このリストは、以下のカテゴリに分割されています。

- ・ アクティビティ

追加するアクティビティ	この要素に関する参照先ドキュメント
アラート	<alert>
割り当て	<assign>
ブレーク	<break>
コール	<call>
コード	<code>
継続	<continue>
遅延	<delay>
空	<empty>
応答	<reply>
ルール	<rule>

追加するアクティビティ	この要素に関する参照先ドキュメント
SQL	<sql>
同期	<sync>
トレース	<trace>
変換	<transform>
XPATH	<xpath>
XSLT	<xslt>

・ 決定とプレースホルダ

追加する決定	この要素に関する参照先ドキュメント
If	<if>
スイッチ	<switch>
分岐	<branch>
ラベル	<label>
マイルストーン	<milestone>

・ ロジック

追加するロジック	この要素に関する参照先ドキュメント
フロー	<flow>
結合	(ダイアグラムに必要。対応する BPL 要素はなし)
スコープ	<scope>
シーケンス	<sequence>

・ ループ

追加するループ	この要素に関する参照先ドキュメント
ForEach	<foreach>
While	<while>
Until	<until>

・ エラー処理

追加するループ	この要素に関する参照先ドキュメント
Throw	<throw>
Catch	<catch>
すべて Catch	<catchall>
Compensate	<compensate>
Compensation ハンドラ	<compensationhandlers>

ダイアグラムに要素を追加すると、[アクティビティ] タブにその要素に適用できるプロパティが表示されます。このタブの上部には、対応する BPL 要素と説明に加えて、その要素の BPL リファレンス・エントリへのアクティブなリンクが表示されます。これは、その後の設定に関する情報を取得するための最も正確な場所です。

以下の設定は、<start> シェイプと <end> シェイプを除くすべての要素に共通しています。

[名前]

シェイプ内のキャプションの名前を入力します。

[x]

ダイアグラムで選択されたシェイプの位置の X 軸座標。

[y]

ダイアグラムで選択されたシェイプの位置の Y 軸座標。

[無効]

そのアクティビティを無効にするには、このチェックボックスにチェックを付けて、有効にするにはチェックを外します。デフォルトでは有効になっています。

[アノテーション]

ダイアグラムでシェイプの横にコメントとして表示されるテキストを入力します。

注釈 <start> シェイプと <end> シェイプには x と y の座標しかないため、これらのシェイプは必要に応じて手動で移動できます。

2.3.1 コール・アクティビティの追加

BPL ビジネス・プロセス内の一般的なタスクは、コール・アクティビティを追加することです。プロダクション内で使用可能なビジネス・プロセスまたはビジネス・オペレーションのいずれかに対する新しい <call> を適切に作成するには、以下の情報が必要です。

- ・ 入力
- ・ 出力
- ・ 名前
- ・ ターゲット
- ・ 要求

2.4 BPL デザイナのプロパティ・タブ

BPL ダイアグラムの右側には、一連のプロパティ・タブを含むペインがあります。二重矢印のアイコンをクリックすることで、この右側のペインを必要に応じて展開および縮小できます。これらのタブうち、3 つはその BPL ビジネス・プロセス自体に関するものであり、1 つは選択したシェイプに関するものです。

- ・ [一般]—その BPL ビジネス・プロセスの全体的な定義に関する設定が含まれています。“[BPL ビジネス・プロセスの一般プロパティの設定](#)”を参照してください。

- ・ **[コンテキスト]** – その BPL ビジネス・プロセスの [context オブジェクトを定義](#)するためのインタフェースが用意されています。
- ・ **[アクティビティ]** – BPL ダイアグラムで選択された項目の設定が含まれています。このタブの内容に関する詳細は、“[BPL ダイアグラムへのアクティビティの追加](#)”の節を参照してください。
- ・ **[プリファレンス]** – BPL ダイアグラムの外観に関する設定が含まれています。詳細は、“[BPL ダイアグラムのプリファレンスの設定](#)”を参照してください。

2.4.1 BPL ビジネス・プロセスの一般プロパティの設定

[一般] タブには、その BPL ビジネス・プロセスに適用される以下の設定が含まれています。

- ・ **[言語]** – [ObjectScript] に設定する必要があります。
- ・ **[レイアウト]** – ダイアグラムのサイズについて、[自動] または [手動] を選択します。[手動] を選択した場合は、[幅] と [高さ] を入力できます。
- ・ **[アノテーション]** – クラスの説明に含めるテキストを入力します。
- ・ **[インクルード]** – インクルード・ファイル名のカンマ区切りリストを入力します (省略可能)。このリストを入力すると、`<code>` セグメントでマクロを使用できるようになります。
- ・ **[バージョン]** – その BPL ダイアグラムのバージョン番号を入力します (省略可能)。
- ・ **[コンポーネント]** – 真の場合、このプロセスがコンポーネント・ライブラリに組み込まれます。コンポーネント・ライブラリ内では、そのプロセスを他のプロセスによって呼び出すことができます。

これらのプロパティの詳細は、“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”で [process](#) のエントリを参照してください。

2.4.2 context オブジェクトの定義

BPL ビジネス・プロセスの context オブジェクトは、ビジネス・プロセス・デザイナの [コンテキスト] タブで定義できます。虫めがねアイコンをクリックすると、以下の各フィールドの [ファインダダイアログ] を起動できます。

- ・ **[要求クラス]** – このプロセスの受信要求のクラスを選択します。
- ・ **[応答クラス]** – このプロセスによって返される応答のクラスを選択します。
- ・ **[コンテキスト・スーパークラス]** – 次に説明するように、このオプションを使用すると [コンテキスト・プロパティ] リストへの追加とは異なる方法で、カスタム・コンテキスト・プロパティが提供されます。[コンテキスト・スーパークラス] を使用するには、`Ens.BP.Context` のカスタム・サブクラスを作成します。このサブクラスで、コンテキスト・プロパティとして使用するクラス・プロパティを定義します。このクラスの名前を、ビジネス・プロセスの [コンテキスト・スーパークラス] の値として使用します。こうすることで `<assign>` アクションを作成する際に、例えば、context オブジェクトの標準プロパティだけでなく、これらのカスタム・プロパティも選択できるようになります。

[コンテキスト・プロパティ] のリストを追加するには、プラス記号をクリックしてビジネス・プロセス・コンテキスト・ウィザードを起動します。次に、以下のフィールドに値を入力します。

- ・ **[プロパティ名]** – 有効な識別子である必要があります。
- ・ プロパティ・データが [単一値]、[リスト・コレクション]、または [配列コレクション] のいずれ科である場合に選択します。
- ・ **[プロパティタイプ]** – パラメータを含むこのプロパティのタイプ。

[タイプ] フィールドにデータ型クラス名を入力するか、虫めがねアイコンをクリックしてデータ型として使用するクラスを探して選択します。

- ・ **[デフォルト値]** (コレクションの場合は無視されます) – 単一値データ型の最初の式を入力します。
- ・ **[インスタンス化]** – オブジェクト値型のプロパティの場合に、そのオブジェクトが作成されたときにインスタンス化されるようにするには、このチェックボックスにチェックを付けます。
- ・ **[説明]** – コンテキスト・プロパティの説明を入力します (省略可能)。

作業内容を保存する場合は **[OK]** を、作業内容を破棄する場合は **[キャンセル]** をクリックします。ビジネス・プロセス・デザイナーによって、必要な `<context>` および `<property>` 要素が BPL コード内に生成されます。

ビジネス・プロセス・デザイナーで MINVAL、MAXVAL、MINLEN、MAXLEN などのプロパティ・パラメータを設定するには、そのプロパティを最初に追加するときに、またはそれ以降の任意の時点で、データ型パラメータを context プロパティに追加します。そのためには、データ型クラス名の後ろにパラメータのカンマ区切りリストをカッコで囲んで挿入します。つまり、単に `%String` や `%Integer` を入力する代わりに、以下のようにデータ型を入力できます。

```
%String(MAXLEN=256)
%Integer(MINVAL=0,MAXVAL=100)
%String(VALUELIST="Buy,Sell,Hold")
```

ビジネス・プロセス・デザイナーによって、必要な `<parameters>` 要素が BPL コード内に生成されます。

context オブジェクト上のプロパティを定義したら、通常のドット構文と対象のプロパティ名を使用して、BPL 内のどこからでもそれらのプロパティを参照できます (例: `context.MyData`)。


参照の詳細は、以下のトピックを参照してください。

- ・ 通常は、“クラスの定義と使用” の “データ型” に記載されたデータ型のシステム・ライブラリからプロパティ・タイプを選択します。`%String`、`%Integer`、`%Boolean` などがあります。
- ・ システム・データ型には、オプションのパラメータがあります。詳細は、“クラスの定義と使用” の “データ型” の “パラメータ” の節を参照してください。これらの中に、`%String` プロパティの有効な最大長と最小長を設定する MINLEN パラメータと MAXLEN パラメータが含まれています。`%String` のデフォルトの最大長は 50 文字です。これを再設定するには、その `%String` プロパティの MAXLEN を別の値に設定します。

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、タイプ `Ens.BusinessProcess` のプロパティ `%Process` 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。


2.4.3 BPL ダイアグラムのプリファレンスの設定

[プリファレンス] タブには、BPL ダイアグラムの外観に適用される以下の設定が含まれています。

- ・ **[グリッド線]** – ダイアグラム上のグリッド線の外観として、**[なし]**、**[明色]**、**[中間色]**、または **[暗色]** を選択します。
- ・ **[アノテーションを表示]** – 各シェイプについて説明したテキストの表示または非表示を選択します。注釈の表示を選択すると、BPL ドキュメント内の `<annotation>` 要素を持つ各シェイプの右上にコメントが表示されます。
- ・ **[自動整列]** – ダイアグラムに新しいシェイプを追加した後に  を選択しなくても、それぞれの新しいシェイプが自動的に整列されるようにします。


シェイプの位置を変更しても、基盤となる BPL コードは変わりません。接続ラインを変更した場合のみ、コードが変更されます。

2.5 スタジオでの BPL 作成に関する注意事項

ビジネス・プロセス・デザイナーでダイアグラムを保存すると、正しい BPL 構文で記述されたクラスの説明 (テキスト・ドキュメント) が生成されます。BPL ダイアグラムおよび BPL ドキュメントは、同じ BPL ビジネス・プロセス・クラスの等しく有効な説明です。スタジオでは、どちらの形式も同じように認識されます。一方の形式を変更すると、変更内容が自動的に他方に反映されます。したがって、スタジオでクラスを表示している最中に、 [他のコードを表示] をクリックするか、Ctrl+Shift+V を押すことで、BPL ドキュメントのダイアグラム・ビューとテキスト・ビューを切り替えることができます。

注釈 テキスト・ビューからグラフィック・ビューに切り替える場合は、再度グラフィック・ビューを表示するために、クラスを閉じてからもう一度開ける必要がある場合があります。

管理ポータルの一部のツールは、他のツールと共に表示されません。スタジオの既存のコマンドはそれらと同じ機能を提供します。

コマンド	スタジオの同等コマンド
[新規作成]	[ファイル] メニューで [新規作成] をクリックしてから [プロダクション] タブの [ビジネス・プロセス] をクリックします。
[開く]	[ファイル] メニューで [開く] をクリックしてファインダー・ダイアログを起動し、ロードする既存の BPL ビジネス・プロセス・クラスを選択し、ビジネス・プロセス・デザイナーを使用して編集を開始します。
[保存]	[ファイル] メニューで [保存] をクリックして、ビジネス・プロセス・ダイアグラムに加えた変更内容を保存します。
[名前を付けて保存]	[ファイル] メニューで [名前を付けて保存] をクリックして、変更内容を新しい BPL ビジネス・プロセス・クラスとして保存します。
[コンパイル]	[ビルド] メニューで [コンパイル] をクリックして、BPL ビジネス・プロセス・クラスをコンパイルします。
	[ファイル] メニューで [印刷] をクリックして、[印刷] ダイアログを開きます。

BPL ダイアグラムを XML ファイルにエクスポートして、その XML ファイルを後で別の InterSystems IRIS インストール環境にインポートすることができます。この操作を実行するためのルールは、他のクラスをファイルにインポートまたはエクスポートする場合と同じです。スタジオでは、[ツール]→[エクスポート] コマンドおよび [ツール]→[インポート] コマンドを使用します。

context オブジェクトの定義

context オブジェクトを定義する際に BPL コード内で実行できるいくつかのタスクがあります。

BPL コードでは、“[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](#)” に説明があるように、プロパティごとに 1 つの `<property>` 要素を `<context>` 要素の中に挿入します。

BPL コード内で MINVAL、MAXVAL、MINLEN、MAXLEN などのプロパティ・パラメータを設定するには、`<property>` 要素の type 属性でデータ型を指定することを許可して (クラス名のみ)、`<property>` 要素内に `<parameters>` 要素を組み込んで、組み込もうとしているすべてのデータ型パラメータを記述する必要があります。詳細は“[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](#)”を参照してください。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>
```


3

構文ルール

ここでは、プロパティを参照したり、さまざまな BPL アクティビティ内で式を作成したりするための構文ルールについて説明します。

3.1 メッセージ・プロパティへの参照

BPL プロセス内部のアクティビティで、メッセージのプロパティを参照しなければならない場合があります。プロパティを参照するためのルールは、操作するメッセージの種類によって異なります。

- ・ 仮想ドキュメント以外のメッセージの場合は、次のような構文を使用します。

```
message.propertyname
```

または

```
message.propertyname.subpropertyname
```

ここで、propertyname はメッセージ内のプロパティで、subpropertyname はそのプロパティのプロパティです。

- ・ XML 仮想ドキュメント以外の仮想ドキュメントの場合は、“[プロダクション内での仮想ドキュメントの使用法](#)”の“[仮想プロパティ・パスに関する構文ガイド](#)”に記載された構文を使用します。
- ・ XML 仮想ドキュメントについては、“[プロダクション内での XML 仮想ドキュメントのルーティング](#)”を参照してください。

3.2 リテラル値

値をプロパティに代入するときに、リテラル値を指定することがよくあります。リテラル値は、trace アクション内の値のような他の場所にも適している場合があります。

リテラル値は次のどちらかです。

- ・ 数値リテラルはただの数字です。例えば、42.3 などです。
- ・ 文字列リテラルは二重引用符で囲まれた文字列です。例えば、“ABD” などです。

注釈 この文字列には XML 予約文字を含めることができません。詳細は、“[XML 予約文字](#)”を参照してください。

仮想ドキュメントの場合は、この文字列に仮想ドキュメント形式で使用するセパレータ文字を含めることができません。“[仮想ドキュメント内のセパレータ文字](#)”と“[XML 予約文字がセパレータでもある場合](#)”を参照してください。

3.2.1 XML 予約文字

BPL プロセスは XML ドキュメントとして保存されるため、XML 予約文字の代わりに XML エンティティを使用する必要があります。

この文字を含めるには...	次の XML エンティティを使用します...
>	>
<	<
&	&
'	'
"	"

例えば、値の Joe’s “Good Time” Bar & Grill をプロパティに代入するには、**[値]** を次のように設定します。

```
"Joe&apos;s &quot;Good Time&quot; Bar &amp; Grill"
```

InterSystems IRIS® では、エディタに入力されたテキストの周りに自動的に CData ブロックが配置されるため、この制限は <code> アクティビティと <sql> アクティビティの内部に適用されません(XML 標準では、CData ブロックの間に XML と解釈すべきではないテキストが入ります。そのため、そのブロックに予約文字を含めることができます)。

3.2.2 仮想ドキュメント内のセパレータ文字

ほとんどの仮想ドキュメント形式で、セグメント間、フィールド間、サブフィールド間などで特定の文字がセパレータとして使用されます。メッセージ内に値を設定するときにこれらの文字をリテラル・テキストとして含める必要がある場合は、代わりに、そのドキュメント形式に適切なエスケープ・シーケンス (存在する場合) を使用する必要があります。

これらの文字は該当するドキュメント内で文書化されます。詳細は、以下を参照してください。

- ・ “[プロダクション内での EDIFACT ドキュメントのルーティング](#)” の参照節内の “[セパレータ](#)”
- ・ “[プロダクション内での X12 ドキュメントのルーティング](#)” の参照節内の “[セパレータ](#)”

3.2.3 XML 予約文字がセパレータでもある場合

- ・ 文字 (& など) がセパレータでそれをリテラル文字として含めたい場合は、仮想ドキュメント形式に適用されるエスケープ・シーケンスを使用します。
- ・ それ以外の場合は、“[XML 予約文字](#)” で示したような XML エンティティを使用します。

3.2.4 数字のコード

リテラル文字列には、10 進表現または 16 進表現を含めることができます。

文字列 `&#n;` は Unicode 文字を表します (n は Unicode 文字の 10 進数)。例えば、`é` は、揚音アクセント符号付きのラテン文字 e (é) を表します。

また、文字列 `&#xh;` も Unicode 文字を表します (h は 16 進の Unicode 文字番号)。例えば、`¿` は逆さの疑問符 (¿) を表します。

3.3 有効な式

値をプロパティに代入するときに、BPL プロセス用に選択した言語で式を指定できます。式は、`<if>` アクティビティの条件、`<trace>` アクティビティ内の値、`<code>` アクティビティ内の文などの他の場所でも使用します。

有効なすべての式を以下に示します。

- ・ [前節](#) で説明したリテラル値。
- ・ 関数コール (InterSystems IRIS はビジネス・ルールやデータ変換で使用するための一連のユーティリティ関数を提供しています。詳細は、“ビジネス・ルールの開発” の “[プロダクションで使用するユーティリティ関数](#)” を参照してください)。
- ・ “[プロパティへの参照](#)” で説明したプロパティへの参照。
- ・ BPL プロセス用に選択したスクリプティング言語の構文を使用してこれらを組み合わせた式。以下に留意してください。
 - － ObjectScript の場合、連結演算子は、以下のように `_` (アンダースコア) 文字です。


```
value="prefix"_source.{MSH:ReceivingApplication}_"suffix"
```
 - － `$CHAR` や `$PIECE` などの便利な ObjectScript 文字列関数については、“ObjectScript リファレンス” を参照してください。
 - － 概要は、“ObjectScript の使用法” を参照してください。

3.4 間接指定

InterSystems IRIS では、以下に示す BPL 要素と属性の組み合わせについてのみ、値を間接的に指定できます。

- ・ `<call name=`
- ・ `<call target=`
- ・ `<sync calls=`
- ・ `<transform class=`

アット・マーク記号 `@` は間接演算子です。

例えば、`<call>` 要素では、`name` 属性または `target` 属性の値を間接指定できます。`name` は呼び出しを識別し、その後、`<sync>` 要素で参照できます。`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">
```

このドキュメントでは、@ 間接構文をサポートしている各要素（[<call>](#)、[<sync>](#)、[<transform>](#)）で、この構文について説明しています。

重要 BPL と DTL は多くの点で似ていますが、DTL は間接指定をサポートしていません。

4

BPL 要素のリスト

ここでは、BPL 要素を機能グループに分類し、各要素の目的について説明します。

4.1 ビジネス・プロセス

BPL ドキュメントは、[〈process〉](#) 要素とそのさまざまな子要素で構成されます。[〈process〉](#) 要素はビジネス・プロセスのコンテナです。

4.2 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。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。

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。変数には、以下のものがあります。

- ・ context
- ・ request
- ・ response
- ・ callrequest
- ・ callresponse
- ・ syncresponses
- ・ synctimedout
- ・ status

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。この規則の例外は、context と呼ばれる汎用コンテナ・オブジェクトです。これは、BPL ドキュメントの先頭に [〈context〉](#) 要素、[〈property〉](#) 要素、および [〈parameters〉](#) 要素を配置することによって BPL 開発者が定義します。

これらの変数の詳細は、“[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](#)” を参照してください。

また、`<assign>` 要素のドキュメントと“[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。

4.3 制御フロー

ビジネス・プロセス内では、アクティビティは順次または並列で実行されます。

- ・ 順次実行は、`<sequence>` 要素で指定します。
- ・ 並列実行は、`<flow>` 要素と `<sequence>` 要素の組み合わせで指定します。

BPL には、BPL ビジネス・プロセス内での実行順序を制御する多数の制御フロー要素があります。

BPL 要素	目的	説明
<code><branch></code>	分岐	条件に基づいて、実行フローを直接変更します。
<code><break></code>	ループ	ループから抜けて、ループ・アクティビティを終了します。
<code><continue></code>	ループ	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。
<code><flow></code>	グループ	不定の順序でアクティビティを実行します。
<code><foreach></code>	ループ	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。
<code><if></code>	分岐	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。
<code><label></code>	分岐	条件分岐操作の分岐先を指定します。
<code><sequence></code>	グループ	他のビジネス・オペレーションやビジネス・プロセスへの 1 つ以上の呼び出しを編成します。BPL ダイアグラムを構成します。
<code><switch></code>	分岐	一連の条件を評価し、実行する操作を決定します。
<code><until></code>	ループ	条件が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。
<code><while></code>	ループ	条件が真である限り、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。

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。

4.4 メッセージング

BPL の要素では、ビジネス・オペレーションや他のビジネス・プロセスへの同期要求および非同期要求を作成できます。

BPL 要素	目的
<call>	要求を送信し、必要に応じてビジネス・オペレーションまたはビジネス・プロセスから応答を受信します。呼び出しは同期でも非同期でもかまいません。
<request>	他のビジネス・オペレーションまたはビジネス・プロセスへの呼び出しの要求を準備します。
<response>	他のビジネス・オペレーションまたはビジネス・プロセスへの呼び出しから返された応答を受信します。
<sync>	他のビジネス・オペレーションやビジネス・プロセスへの 1 つ以上の非同期呼び出しからの応答を待ちます。
<reply>	プロセスの実行が完了する前に、ビジネス・プロセスから基本応答を返します。

4.5 スケジュールリング

<delay> 要素を使用して、指定した期間または指定した時刻までビジネス・プロセスの実行を遅延できます。

4.6 ルールおよび決定事項

<rule> 要素はビジネス・ルールを実行します。この要素は、ビジネス・ルール名に加えて、判断結果とその理由 (オプション) を保持するパラメータを指定します。

<rule> 要素のパラメータには、context という名前の汎用的な実行コンテキスト変数内の任意のプロパティを含めることができます。したがって、ルールを呼び出すビジネス・プロセスに対する一般的な設計アプローチとしては、ビジネス・プロセスが以下の点を実現するようにします。

1. <property> 要素と <context> 要素を提供して、適切な名前とタイプを持つプロパティが context オブジェクトに含まれるようにします。
例えば、公的な教育ローンを受ける適格性をルールによって判定する場合、**Age**、**State**、および **Income** などのプロパティを追加することも考えられます。
2. 希望の方法で、プロパティの値を収集します。例えば、ビジネス・オペレーションやビジネス・プロセスに要求を送信し、応答が返されると、context のプロパティに値を割り当てる、などの方法があります。
3. <rule> 要素を提供して、これらの入力値に基づいて回答を返すビジネス・ルールを呼び出します。

実行コンテキスト変数の詳細は、前述した“[ビジネス・プロセスの実行コンテキスト](#)”を参照してください。

ビジネス・ルールの作成に関する情報は、“[ビジネス・ルールの開発](#)”を参照してください。

4.7 データ操作

BPL には、データを別の場所に移動するための要素がいくつか用意されています。例えば、一般的なビジネス・プロセスでは、ビジネス・オペレーションまたは他のビジネス・プロセスへの一連の呼び出しが作成されます。これらの呼び出しを設定し、それによって返されるデータを処理するため、ビジネス・プロセスでは、さまざまな実行コンテキスト変数 (context、request、response など) の間でデータがやり取りされます。この“データ移動”とその他のデータ操作は、以下の要素を使用して行われます。

BPL 要素	目的
<code><assign></code>	プロパティに値を割り当てます。
<code><sql></code>	埋め込まれた SQL SELECT 文を実行します。
<code><transform></code>	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。
<code><xpath></code>	ターゲットの XML ドキュメントで XPath 式を評価します。
<code><xslt></code>	XSLT 変換を実行して、データ・ストリームを変更します。

4.8 ユーザ記述コード

BPL だけでは特定の問題を解決できない場合に備え、InterSystems IRIS® には、自動生成されたビジネス・プロセス・コードにユーザ記述コードを埋め込むためのしくみが用意されています。

BPL 要素	目的
<code><code></code>	CDATA ブロックに追加するコードを指定できます。
<code><empty></code>	何も実行されません。コードを記述するまでのプレースホルダとして機能します。

4.9 ロギング

BPL には、情報メッセージやエラー・メッセージのログに使用できる要素があります。

BPL 要素	目的
<code><alert></code>	外部の警告メカニズムにテキスト・メッセージを書き込みます。
<code><milestone></code>	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。
<code><trace></code>	コンソール・ウィンドウおよびイベント・ログにテキスト・メッセージを書き込みます。

4.10 エラー処理

BPL には、エラーをスローおよびキャッチしたり、エラーやフォールトを補償するための要素があります。これらの要素は互いに密接に関連しています。詳細は、このトピックの“[BPL のエラー処理](#)”を参照してください。エラー処理関連の要素は以下のとおりです。

BPL 要素	目的
<code><catch></code>	<code><throw></code> 要素によって生成されたフォールトをキャッチします。
<code><catchall></code>	<code><catch></code> 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。
<code><compensate></code>	<code><catch></code> または <code><catchall></code> から <code><compensationhandler></code> を呼び出します。
<code><compensationhandler></code>	以前のアクションを元に戻す一連のアクティビティを実行します。
<code><compensationhandlers></code>	1 つ以上の <code><compensationhandler></code> 要素が含まれます。

BPL 要素	目的
<faulthandlers>	ゼロ個以上の <catch> と 1 つの <catchall> 要素を使用できます。
<scope>	フォールト・ハンドラおよび補償ハンドラで複数のアクティビティをラップします。
<throw>	指定された特定のフォールトをスローします。

5

BPL のエラー処理

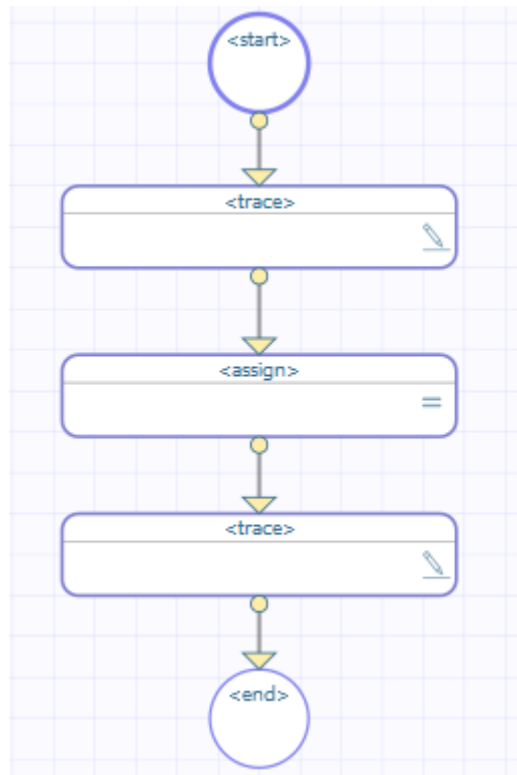
ここでは、BPL ビジネス・プロセスでのエラー処理について説明します。BPL は、ビジネス・プロセスがエラーをスロー/キャッチできるようにするためのエラー・ハンドラ、およびエラーの発生原因となったアクションを元に戻すことによって、エラーから復帰する方法を指定するための補償ハンドラを備えています。

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 `<catchall>` ロジックをトリガしません。

エラー処理で呼び出される BPL 要素は、`<scope>`、`<throw>`、`<catch>`、`<catchall>`、`<compensate>`、`<compensationhandlers>`、`<compensationhandler>`、および `<faulthandlers>` です。このトピックでは、これらの要素を紹介し、これらの要素が連携して、さまざまなエラー処理シナリオをサポートするしくみについて説明します。

5.1 システム・エラー — エラー処理なし

以下は、エラー状態を生成し、エラー処理を行わない BPL ビジネス・プロセスの例です。

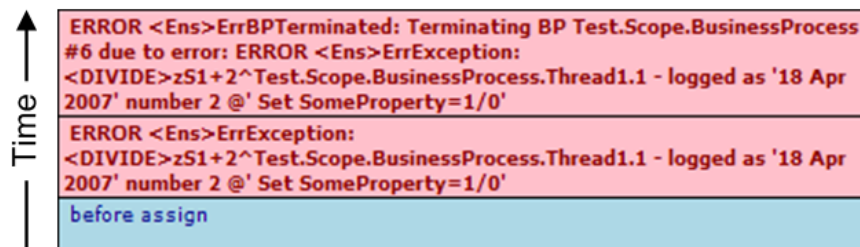


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの before assign を生成します。
2. <assign> 要素が SomeProperty を式 $1/0$ と同じに設定しようとします。この試みによって 0 による除算システム・エラーが発生します。
3. ビジネス・プロセスが終了し、メッセージがイベント・ログに送信されます。
2 つ目の <trace> 要素が使用されることはありません。

5.1.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



基礎的な情報は、“プロダクションの管理”の“[イベント・ログ](#)”を参照してください。

5.1.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

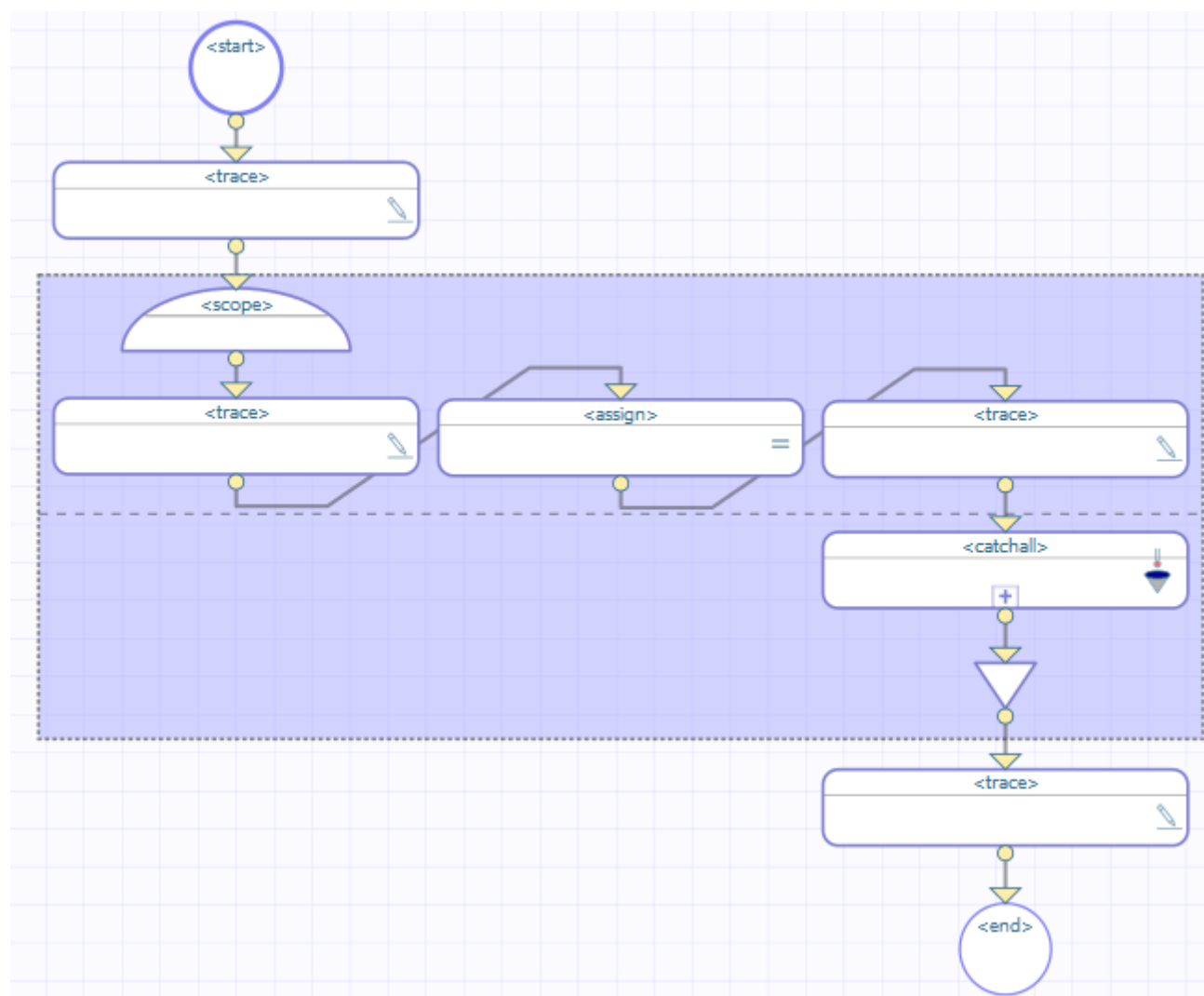
Class Member

```
XData BPL
{
<process language='objectscript'
    request='Test.Scope.Request'
    response='Test.Scope.Response' >
    <sequence>
        <trace value='before assign'/'>
        <assign property="SomeProperty" value="1/0"/>
        <trace value='after assign'/'>
    </sequence>
</process>
}
```

5.2 システム・エラー — Catchall で対応

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プです。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。

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。`<scope>` に `<faulthandlers>` 要素が含まれているため、長方形の中央を横切る水平点線が追加されます。この線の下側の領域に、`<faulthandlers>` の内容が表示されます。

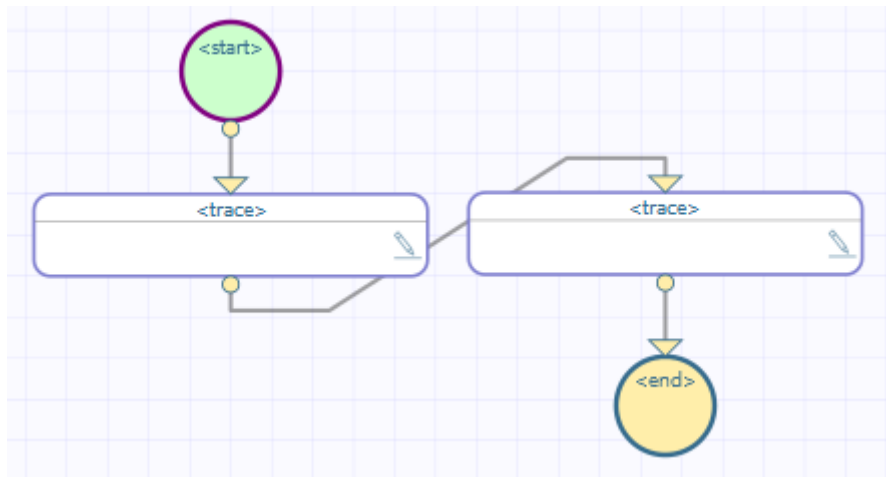


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの **before scope** を生成します。
2. <scope> 要素はスコープの始まりです。
3. 2 つ目の <trace> 要素がメッセージの **before assign** を生成します。
4. <assign> 要素が式 $1/0$ を評価しようとします。この試みによって 0 による除算システム・エラーが発生します。
5. これで、制御が <scope> 内で定義された <faulthandlers> に移ります。<scope> 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に <faulthandlers> 要素の内容が表示されます。この場合は、<catch> が存在しませんが、<catchall> 要素が存在するため、ここに制御が移ります。

InterSystems IRIS® は、<assign> 要素の直後の <trace> 要素メッセージを無視することに注意してください。

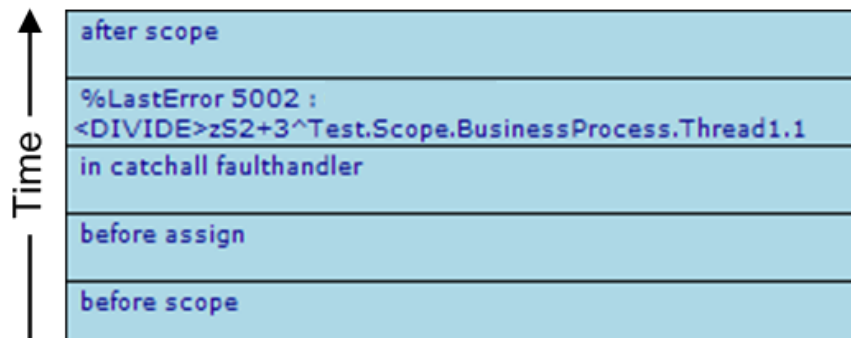
<catchall> を掘り下げる場合は、以下を参照してください。



6. <catchall> 内部で、<trace> 要素がメッセージの in catchall fault handler を生成します。
7. <catchall> 内部で、別の <trace> 要素が `System.Status` メソッドと特殊変数の `%Context` と `%LastError` を使用してエラーの特性を調査するメッセージを生成します。詳細は、“[イベント・ログ・エントリ](#)”を参照してください。
8. <scope> が終わります。
9. 最後の <trace> 要素がメッセージの after scope を生成します。

5.2.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下のようになります。



予期しないシステム・エラーが発生した場合、<scope> 内に <faulthandlers> ブロックがあるときは、“[システム・エラー — エラー処理なし](#)”の例とは異なり、イベント・ログにエントリが自動的に出力されません。ビジネス・プロセスが何を実行するかは、<faulthandlers> ブロックによって決定されます。この例では、エラーに関する情報を含む <trace> メッセージを出力します。実際のエラーを示すイベント・ログ・エントリは、<catchall> ブロック内の以下の文によって生成されます。

XML

```
<trace value=
  '%LastError %_
  $System.Status.GetErrorCodes(..%Context.%LastError)_
  ' : "%_
  $System.Status.GetOneStatusText(..%Context.%LastError)'
/>
```

BPL コンテキスト変数 `%LastError` には、常に `%Status` 値が含まれます。<UNDEF> などの予期せぬシステム・エラーが発生した場合は、この `%Status` 値がエラー “ObjectScript エラー” (コード 5002) および特殊変数 `$ZERROR` のテキストから作成されます。`%LastError` から対応するエラー・コードとテキストを取得するには、上に示すように、`System.Status` の `GetErrorCodes` メソッドと `GetOneStatusText` メソッドを使用して、それらを <trace> 文字列に連結します。

5.2.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

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>
}
```

5.3 フォールトをスロー — Catchall 対応

〈throw〉文を実行するとき、その fault 値は結果が文字列となる式です。Java のような他のオブジェクト指向言語と異なり、フォールトはオブジェクトではなく、文字列値です。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

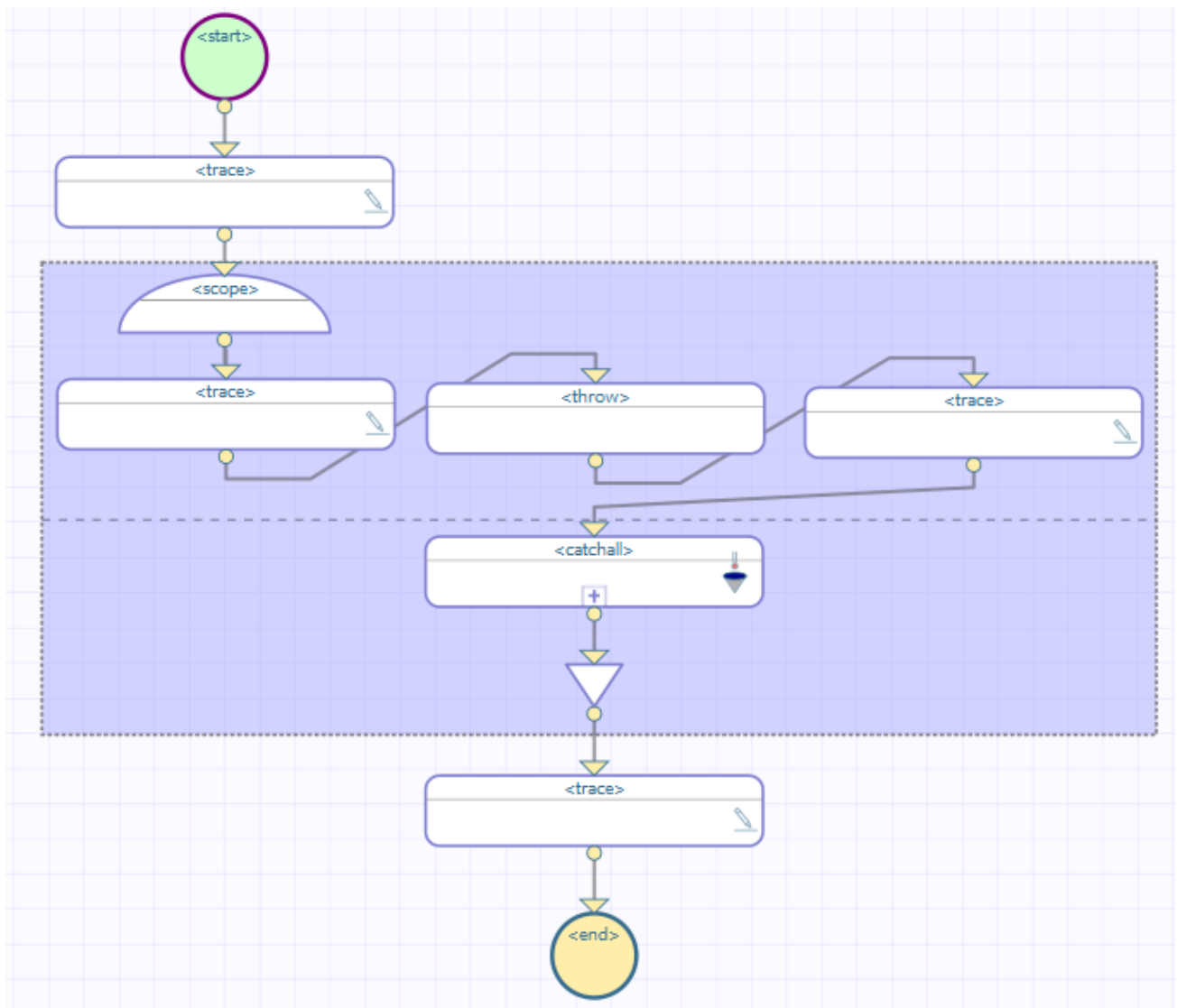
```
<throw fault=' "thrown" ' />
```

〈throw〉文が実行されると、即座に、同じ〈scope〉内の〈faulthandlers〉ブロックに制御が移り、〈throw〉以降の途中の文はすべて省略されます。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。

フォールトがスローされ、フォールト文字列と一致する〈catch〉ブロックが〈faulthandlers〉ブロック内に存在しない場合は、制御が〈throw〉文から〈faulthandlers〉内の〈catchall〉ブロックに移ります。〈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。

以下の BPL があるとします。スペースの関係で、〈start〉要素と〈end〉要素が表示されていません。

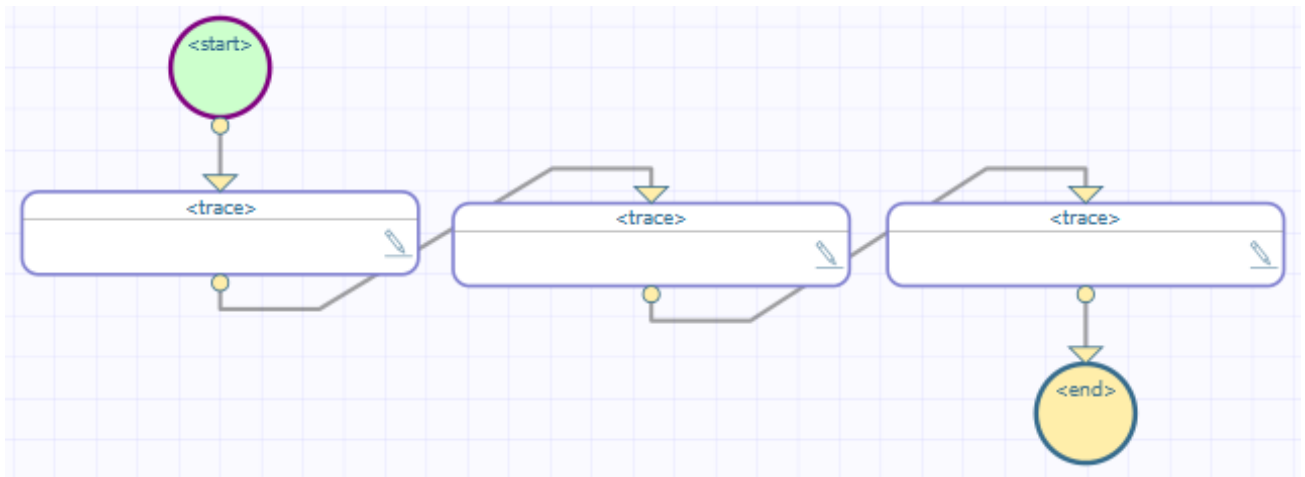


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの before scope を生成します。
2. <scope> 要素はスコープの始まりです。
3. 2 つ目の <trace> 要素がメッセージの before assign を生成します。
4. <throw> 要素が特定の名前付きフォールト ("MyFault") をスローします。
5. これで、制御が <scope> 内で定義された <faulthandlers> に移ります。<scope> 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に <faulthandlers> 要素の内容が表示されます。この場合は、<catch> が存在しませんが、<catchall> 要素が存在するため、ここに制御が移ります。

InterSystems IRIS は 3 つ目の <trace> 要素を無視することに注意してください。

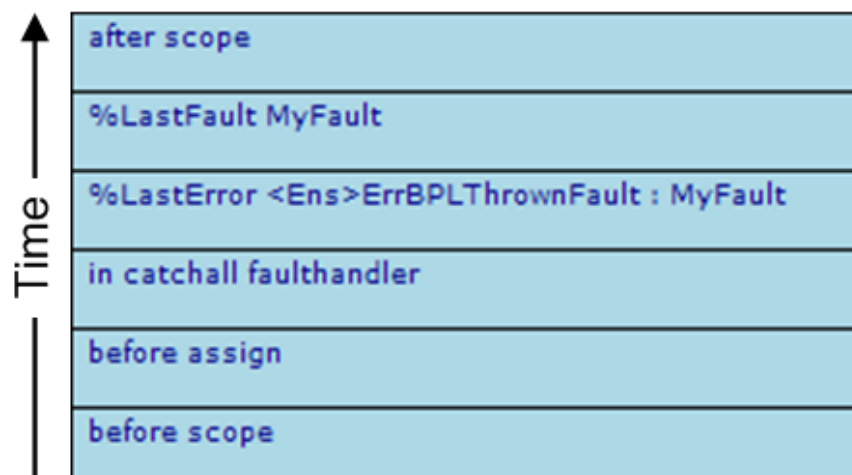
<catchall> を掘り下げる場合は、以下を参照してください。



6. `<catchall>` 内部で、最初の `<trace>` 要素がメッセージの `in catchall faulthandler` を生成します。
7. `<catchall>` 内部で、2 つ目の `<trace>` 要素が `$$System.Status` メソッドと特殊変数の `%Context` と `%LastError` を使用してフォールトに関する情報を提供するメッセージを生成します。フォールトがスローされた結果としての `%LastError` は、システム・エラーの結果としての値と異なります。
 - ・ `GetErrorCodes` は `<Ens>ErrBPLThrownFault` を返します。
 - ・ `GetOneStatusText` は、`<throw>` 文の `fault` 式から得られたテキストを返します。
8. `<catchall>` 内部で、3 つ目の `<trace>` 要素が BPL コンテキスト変数の `%LastFault` を使用してフォールトに関する情報を提供するメッセージを生成します。このメッセージには、`<throw>` 文の `fault` 式から得られたテキストが含まれます。
9. `<scope>` が終わります。
10. 最後の `<trace>` 要素がメッセージの `after scope` を生成します。

5.3.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



5.3.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

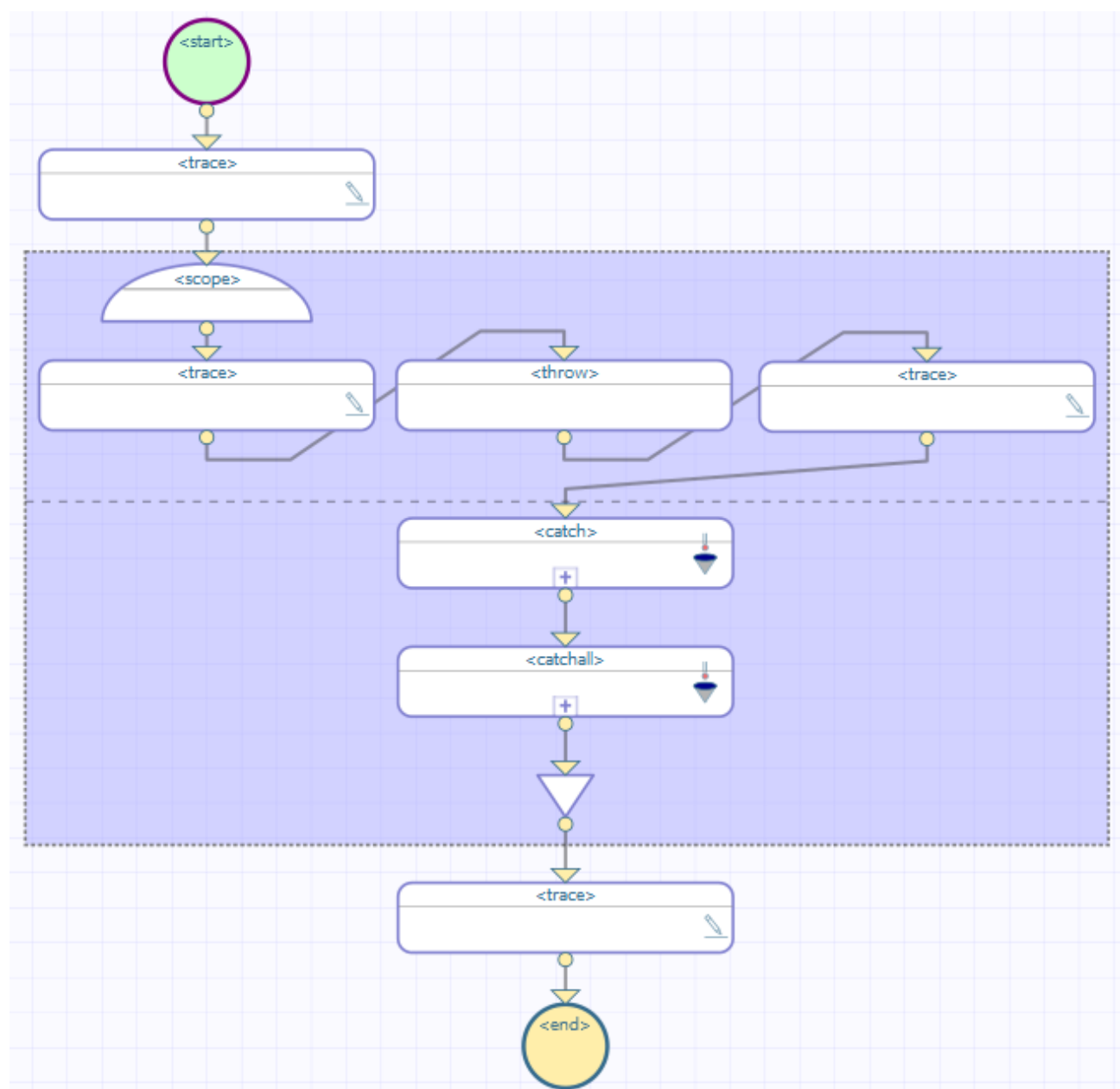
Class Member

```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before scope' />
    <scope>
      <trace value='before assign' />
      <throw fault='MyFault' />
      <trace value='after assign' />
      <faulthandlers>
        <catchall>
          <trace value='in catchall fault handler' />
          <trace value=
            '%LastError' _
            '$System.Status.GetErrorCodes(..%Context.%LastError)_
            " : "_
            '$System.Status.GetOneStatusText(..%Context.%LastError)'
            />
          <trace value='%LastFault' _..%Context.%LastFault' />
        </catchall>
      </faulthandlers>
    </scope>
    <trace value='after scope' />
  </sequence>
</process>
}
```

5.4 フォールトをスロー — Catch に対応

前述の例のように、スローされたフォールトを<catchall>で処理する以外に、特定の<catch>を使用することもできます。

以下の BPL があるとします。

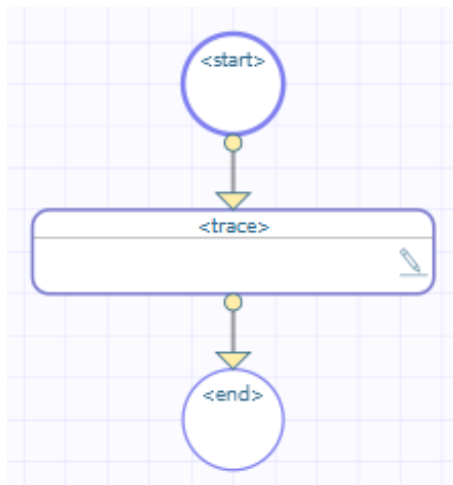


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの **before scope** を生成します。
2. <scope> 要素はスコープの始まりです。
3. 2 つ目の <trace> 要素がメッセージの **before throw** を生成します。
4. <throw> 要素が特定の名前付きフォールト ("MyFault") をスローします。
5. これで、制御が <scope> 内で定義された <faulthandlers> に移ります。<scope> 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に <faulthandlers> 要素の内容が表示されます。この場合は、fault 値が "MyFault" の <catch> 要素が存在するため、ここに制御が移ります。<catchall> 要素は無視されます。

InterSystems IRIS は、<throw> 要素以降の <trace> 要素メッセージを無視することに注意してください。

<catch> を掘り下げる場合は、以下を参照してください。

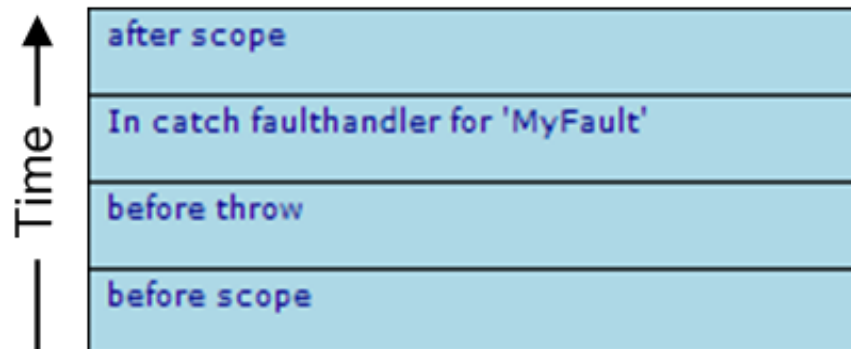


注釈 <catchall> を設ける場合は、それを <faulthandlers> ブロックの最後の文にする必要があります。必ず、<catchall> の前にすべての <catch> ブロックを配置します。

6. <catch> 内部で、<trace> 要素がメッセージの in catch faulthandler for 'MyFault' を生成します。
7. <scope> が終わります。
8. 最後の <trace> 要素がメッセージの after scope を生成します。

5.4.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



5.4.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

Class Member

```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before scope' />
    <scope>
      <trace value='before throw' />
      <throw fault='MyFault' />
      <trace value='after throw' />
    <faulthandlers>
      <catch fault='MyFault' >
```

```

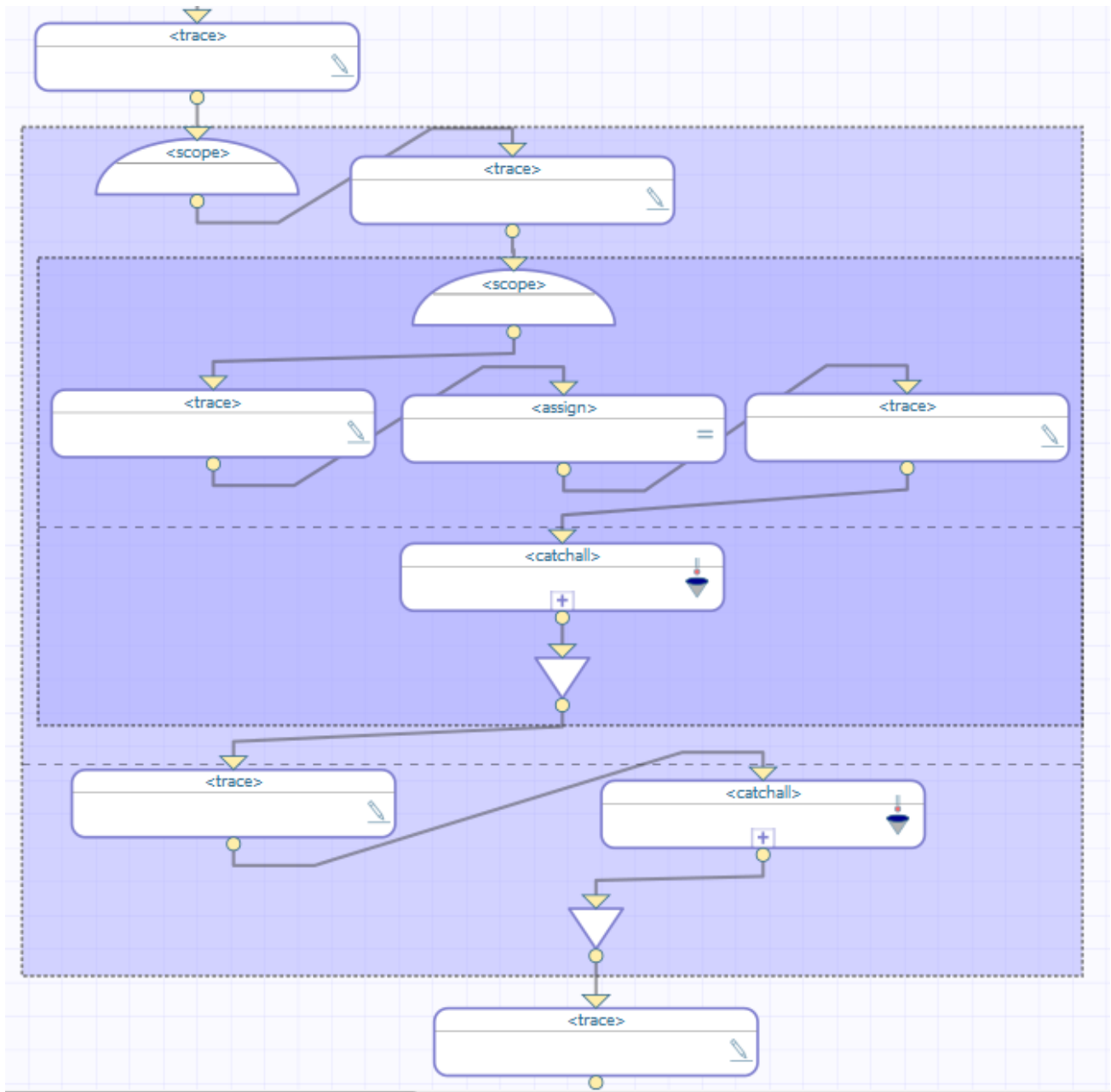
    <trace value='In catch fault handler for &apos;MyFault&apos;' />
  </catch>
  <catchall>
    <trace value='in catchall fault handler' />
    <trace value=
      '%LastError' _
      $System.Status.GetErrorCodes(..%Context.%LastError)_
      " : " _
      $System.Status.GetOneStatusText(..%Context.%LastError)'
    />
    <trace value='%LastFault' '..%Context.%LastFault' />
  </catchall>
</faulthandlers>
</scope>
<trace value='after scope' />
</sequence>
</process>
}

```

5.5 ネスト構造のスコープ – 内側のフォールト・ハンドラに Catchall を配置

〈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。この後のトピックでは、複数のスコープがネストされているとき、内側のスコープで発生したエラーやフォールトを BPL がどのように処理するかを説明します。

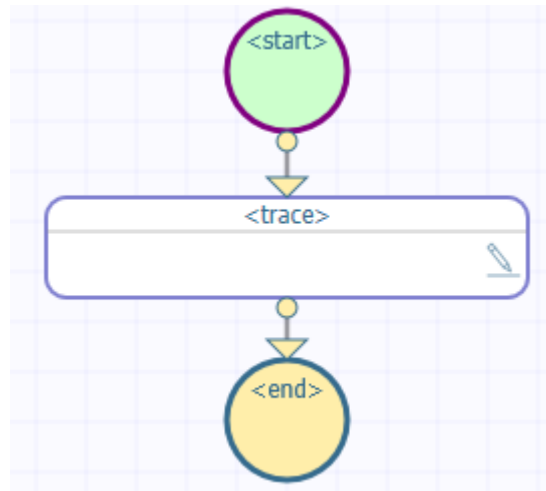
以下の BPL があるとします (ここでは、〈start〉要素と〈end〉要素が省略されています)。



この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの before outer scope を生成します。
2. 最初の <scope> 要素は外側の範囲の始まりです。
3. 2 つ目の <trace> 要素がメッセージの in outer scope, before inner scope を生成します。
4. 2 つ目の <scope> 要素は内側の範囲の始まりです。
5. 次の <trace> 要素がメッセージの in inner scope, before assign を生成します。
6. <assign> 要素が式 $1/0$ を評価しようとします。この試みによって 0 による除算システム・エラーが発生します。
7. これで、制御が内側の <scope> 内で定義された <faulthandlers> に移ります。この <scope> 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に <faulthandlers> 要素の内容が表示されます。この場合は、<catch> が存在しませんが、<catchall> が存在するため、ここに制御が移ります。

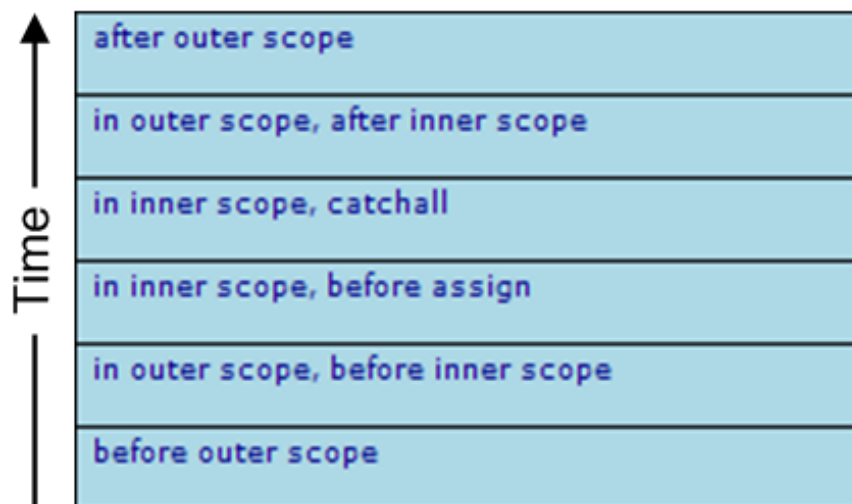
InterSystems IRIS は、`<assign>` 要素の直後の `<trace>` 要素を無視することに注意してください。
この `<catchall>` を掘り下げる場合は、以下を参照してください。



8. `<catchall>` 内部で、`<trace>` 要素がメッセージの `in inner scope, catchall` を生成します。
9. 内側の `<scope>` が終わります。
10. 次の `<trace>` 要素がメッセージの `in outer scope, after inner scope` を生成します。
11. 外側の `<scope>` 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に `<catchall>` を含む `<faulthandlers>` 要素の内容が表示されます。フォールトが存在しないため、この `<catchall>` は無視されます。
12. 外側の `<scope>` が終わります。
13. 最後の `<trace>` 要素がメッセージの `after outer scope` を生成します。

5.5.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下のようになります。



5.5.2 この BPL 用の XData

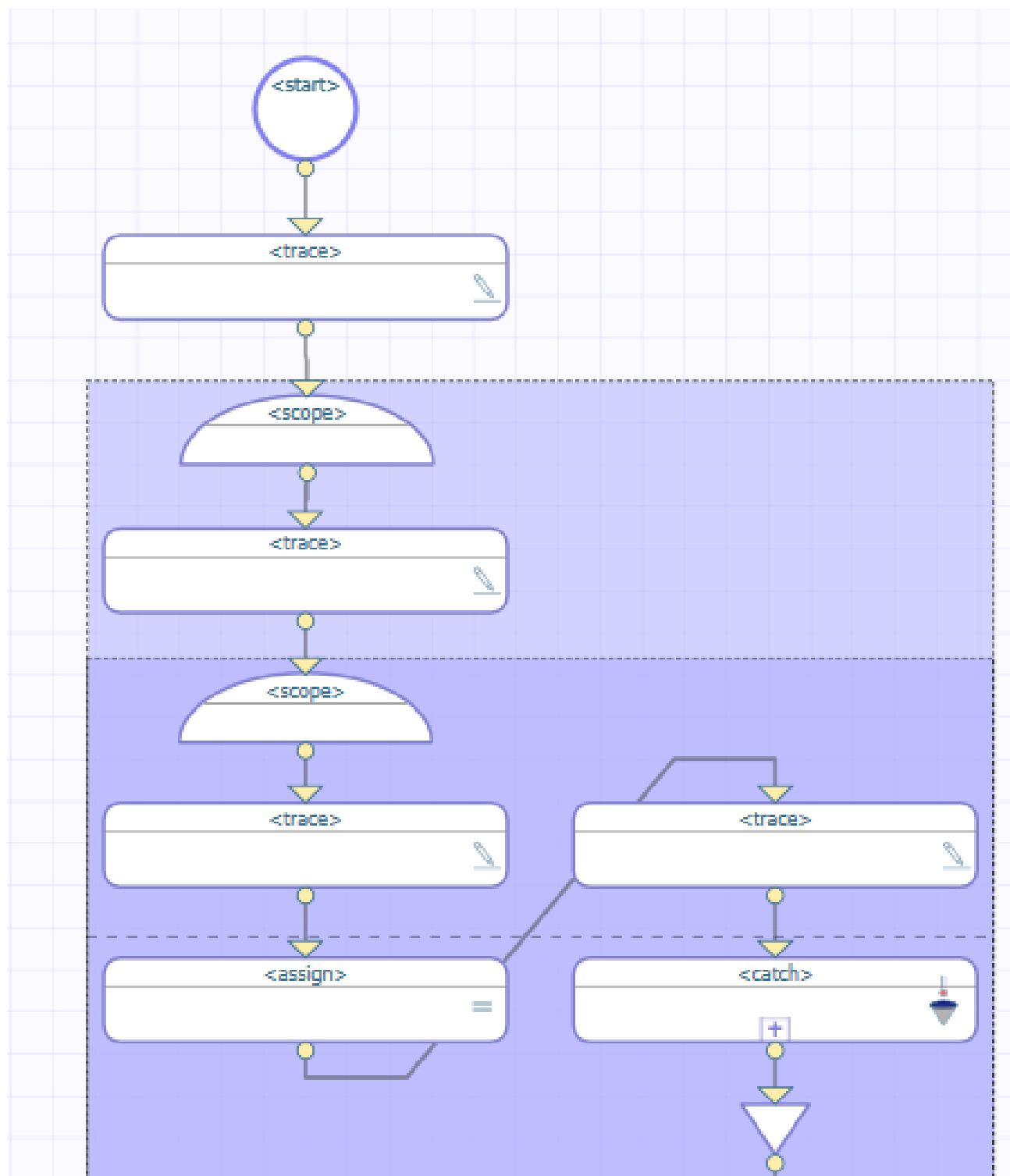
この BPL は以下の XData ブロックによって定義されます。

Class Member

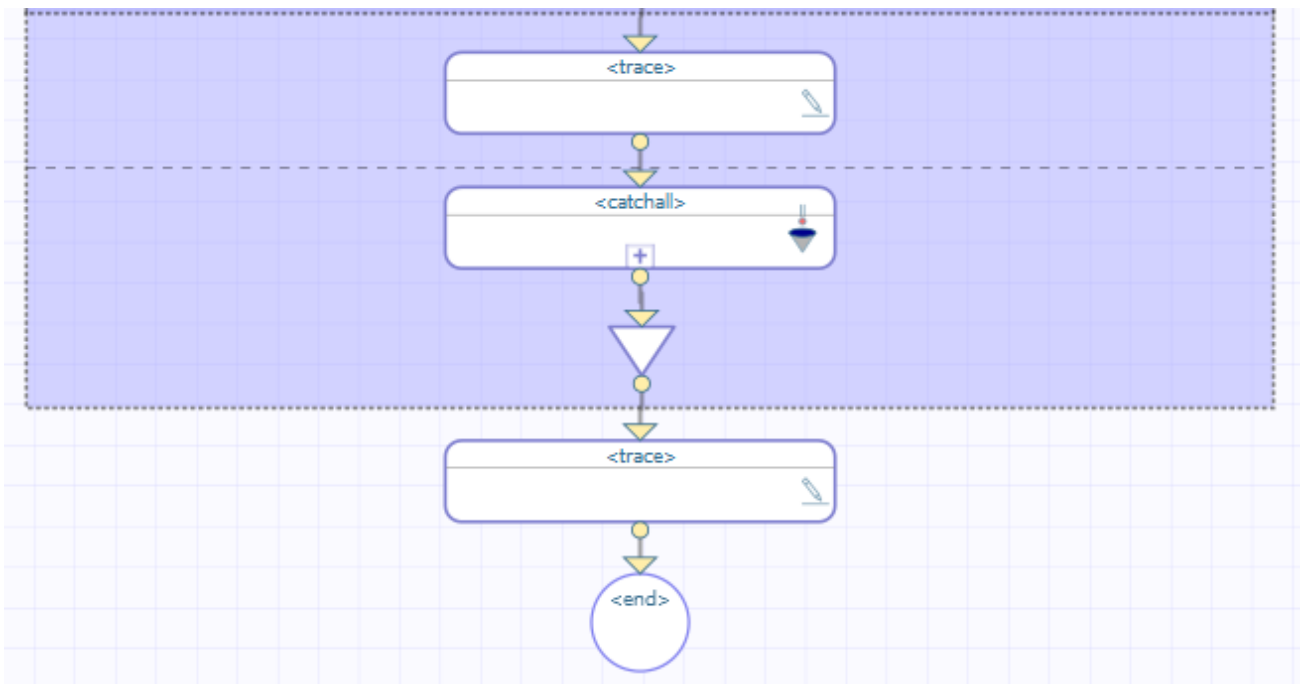
```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before outer scope'/'>
    <scope>
      <trace value='in outer scope, before inner scope'/'>
      <scope>
        <trace value='in inner scope, before assign'/'>
        <assign property='SomeProperty' value='1/0'/'>
        <trace value='in inner scope, after assign'/'>
        <faulthandlers>
          <catchall>
            <trace value='in inner scope, catchall'/'>
          </catchall>
        </faulthandlers>
      </scope>
      <trace value='in outer scope, after inner scope'/'>
      <faulthandlers>
        <catchall>
          <trace value='in outer scope, catchall'/'>
        </catchall>
      </faulthandlers>
    </scope>
    <trace value='after outer scope'/'>
  </sequence>
</process>
}
```

5.6 ネスト構造のスコープ – 外側のフォールト・ハンドラに Catchall を配置

以下の BPL があるとします (部分的に表示されています)。



この BPL の残りを以下に示します。



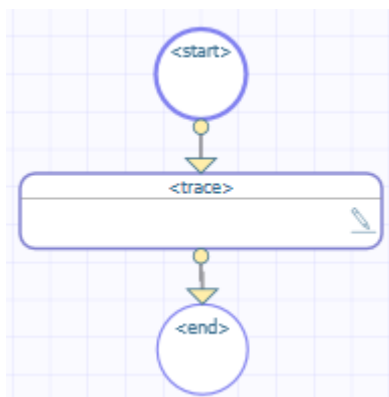
この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の <trace> 要素がメッセージの `before outer scope` を生成します。
2. 最初の <scope> 要素は外側の範囲の始まりです。
3. 次の <trace> 要素がメッセージの `in outer scope, before inner scope` を生成します。
4. 2 つ目の <scope> 要素は内側の範囲の始まりです。
5. 次の <trace> 要素がメッセージの `in inner scope, before assign` を生成します。
6. <assign> 要素が式 `1/0` を評価しようとします。この試みによって 0 による除算システム・エラーが発生します。
7. これで、制御が内側の <scope> 内で定義された <faulthandlers> に移ります。この <scope> 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に <faulthandlers> 要素の内容が表示されます。この場合は、<catch> が存在しますが、その `fault` 値が、スローされたフォールトと一致しません。内側の範囲には <catchall> がありません。

InterSystems IRIS は、<assign> の直後の <trace> 要素を無視することに注意してください。

8. これで、制御が外側の <scope> 内の <faulthandlers> ブロックに移ります。どの <catch> もフォールトと一致しませんが、<catchall> ブロックがあります。制御がこの <catchall> に移ります。

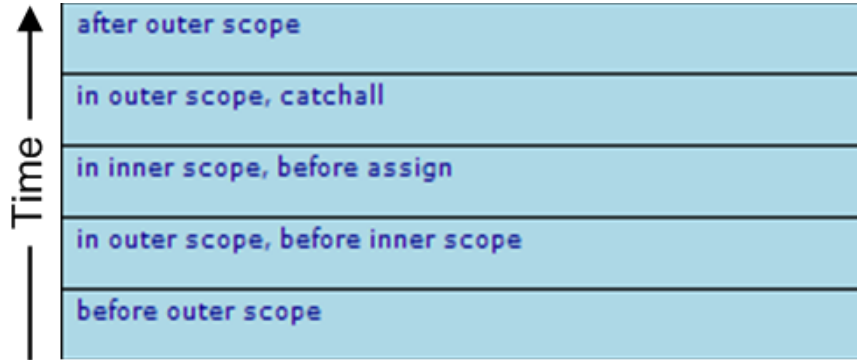
この <catchall> を掘り下げる場合は、以下を参照してください。



9. <catchall> 内部で、<trace> 要素がメッセージの in inner scope, catchall を生成します。
10. 外側の <scope> が終わります。
11. 最後の <trace> 要素がメッセージの after outer scope を生成します。

5.6.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



5.6.2 この BPL 用の XData

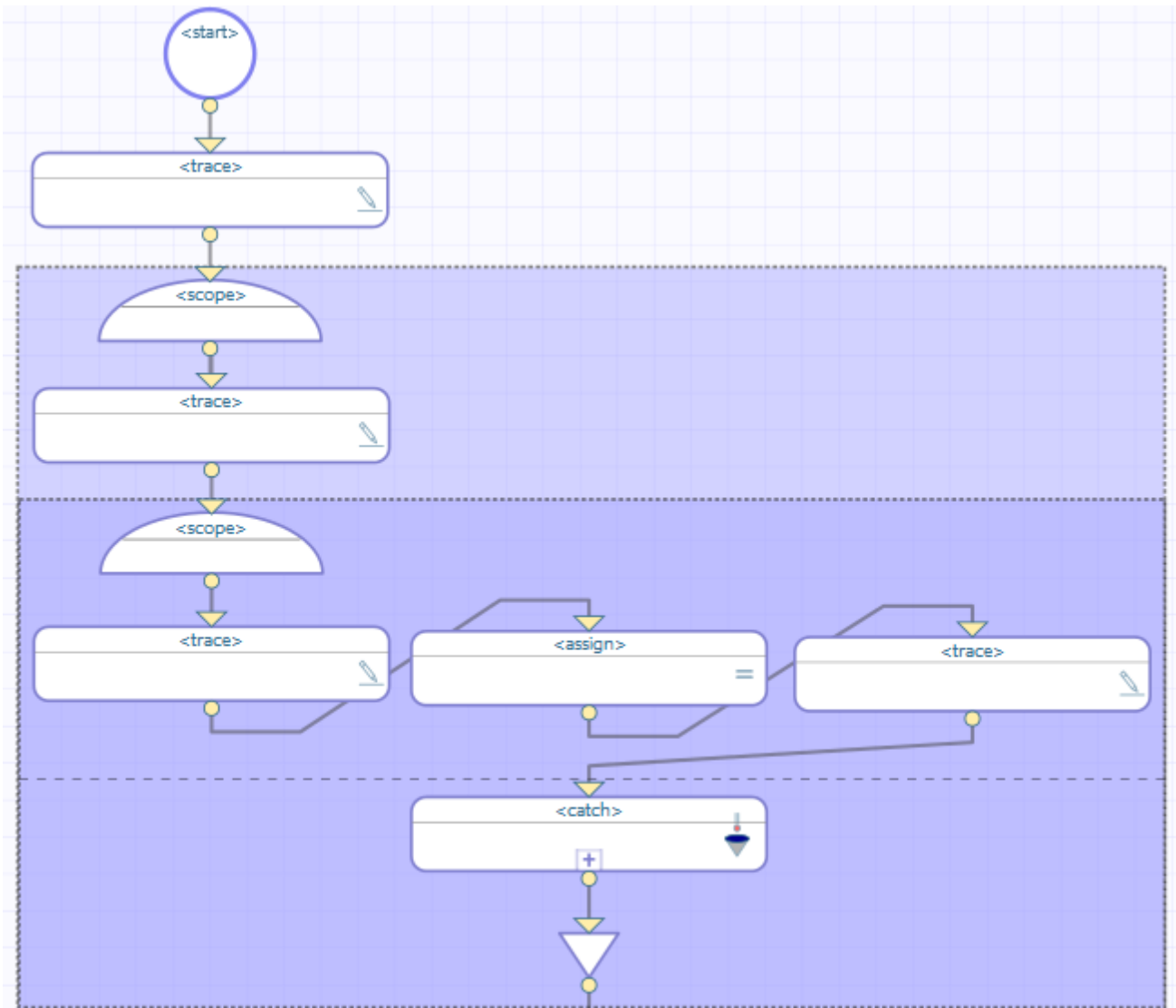
この BPL は以下の XData ブロックによって定義されます。

Class Member

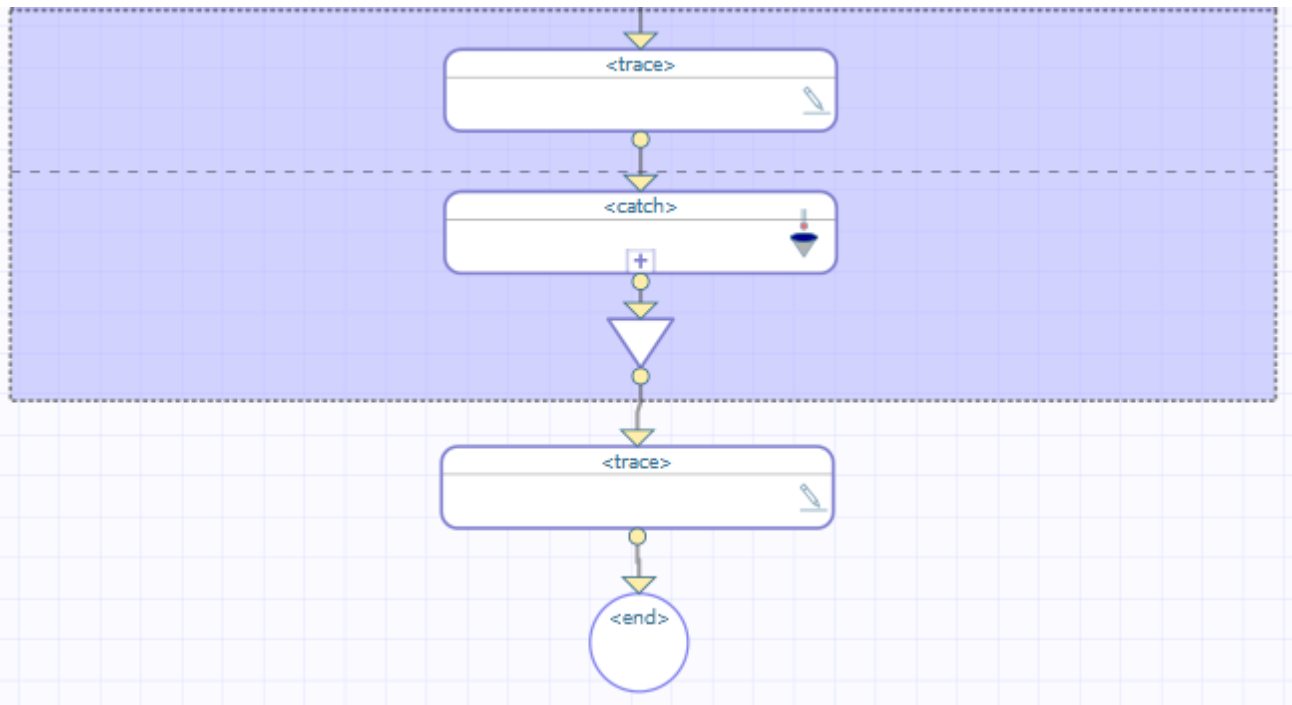
```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before outer scope' />
    <scope>
      <trace value='in outer scope, before inner scope' />
      <scope>
        <trace value='in inner scope, before assign' />
        <assign property='SomeProperty' value='1/0' />
        <trace value='in inner scope, after assign' />
        <faulthandlers>
          <catch fault='MismatchedFault'>
            <trace value='In catch fault handler for &apos;MismatchedFault&apos;' />
          </catch>
        </faulthandlers>
      </scope>
      <trace value='in outer scope, after inner scope' />
      <faulthandlers>
        <catchall>
          <trace value='in outer scope, catchall' />
        </catchall>
      </faulthandlers>
    </scope>
    <trace value='after outer scope' />
  </sequence>
</process>
}
```

5.7 ネスト構造のスコープ – どのスコープでも一致しない場合

以下の BPL があるとしたします (部分的に表示されています)。



この BPL の残りを以下に示します。

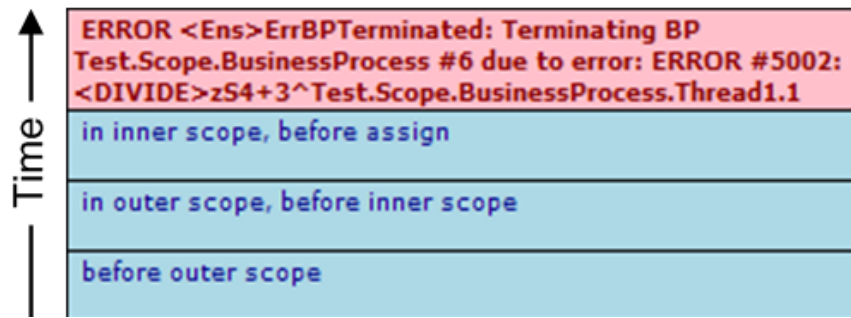


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の `<trace>` 要素がメッセージの `before outer scope` を生成します。
2. 最初の `<scope>` 要素は外側のスコープの始まりです。
3. 次の `<trace>` 要素がメッセージの `in outer scope, before inner scope` を生成します。
4. 2 目目の `<scope>` 要素は内側のスコープの始まりです。
5. 次の `<trace>` 要素がメッセージの `in inner scope, before assign` を生成します。
6. `<assign>` 要素が式 `1/0` を評価しようとします。この試みによって 0 による除算システム・エラーが発生します。
7. これで、制御が内側の `<scope>` 内の `<faulthandlers>` ブロックに移ります。`<scope>` 長方形に、中央を横切る水平点線が追加されます。この点線の下側の領域に `<faulthandlers>` 要素の内容が表示されます。この場合は、`<catch>` が存在しますが、その `fault` 値が、スローされたフォールトと一致しません。内側のスコープには `<catchall>` がありません。
8. これで、制御が外側の `<scope>` 内の `<faulthandlers>` ブロックに移ります。どの `<catch>` もフォールトと一致しません。また、`<catchall>` ブロック也没有ありません。
9. BPL が即座に停止して、メッセージがイベント・ログに送信されます。

5.7.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下のようになります。



このイベント・ログと、“システム・エラー — エラー処理なし” の例で紹介したイベント・ログとの間には重要な相違点があります。これら 2 つの例の共通点は、ゼロによる除算のエラーが発生した場合の適切なフォールト処理が設けられていないことです。

ただし、“システム・エラー — エラー処理なし” の例には <scope> がなく、<faulthandlers> ブロックもありません。このような状況では、最初の例で示したように、InterSystems IRIS は、自動的に、システム・エラーをイベント・ログに出力します。

一方、上記の例では、各 <scope> に <faulthandlers> ブロックが含まれています。このような場合は、“システム・エラー — エラー処理なし” の例と異なり、イベント・ログにシステム・エラーが自動的に出力されません。予期しないエラーが発生したとき <trace> メッセージをイベント・ログに出力するかどうかは、BPL ビジネス・プロセス開発者が決定します。この例には、フォールトをキャッチする <faulthandlers> ブロックがありません。ビジネス・プロセスの終了に関する自動メッセージには、システム・エラーのトレース情報のみが含まれます（上記の 4 番）。

システム・エラー・メッセージがターミナル・ウィンドウに表示されます。

```
ERROR #5002: ObjectScript error: <DIVIDE>zS4+3^Test.Scope.BusinessProcess.Thread1.1
```

5.7.2 この BPL 用の XData

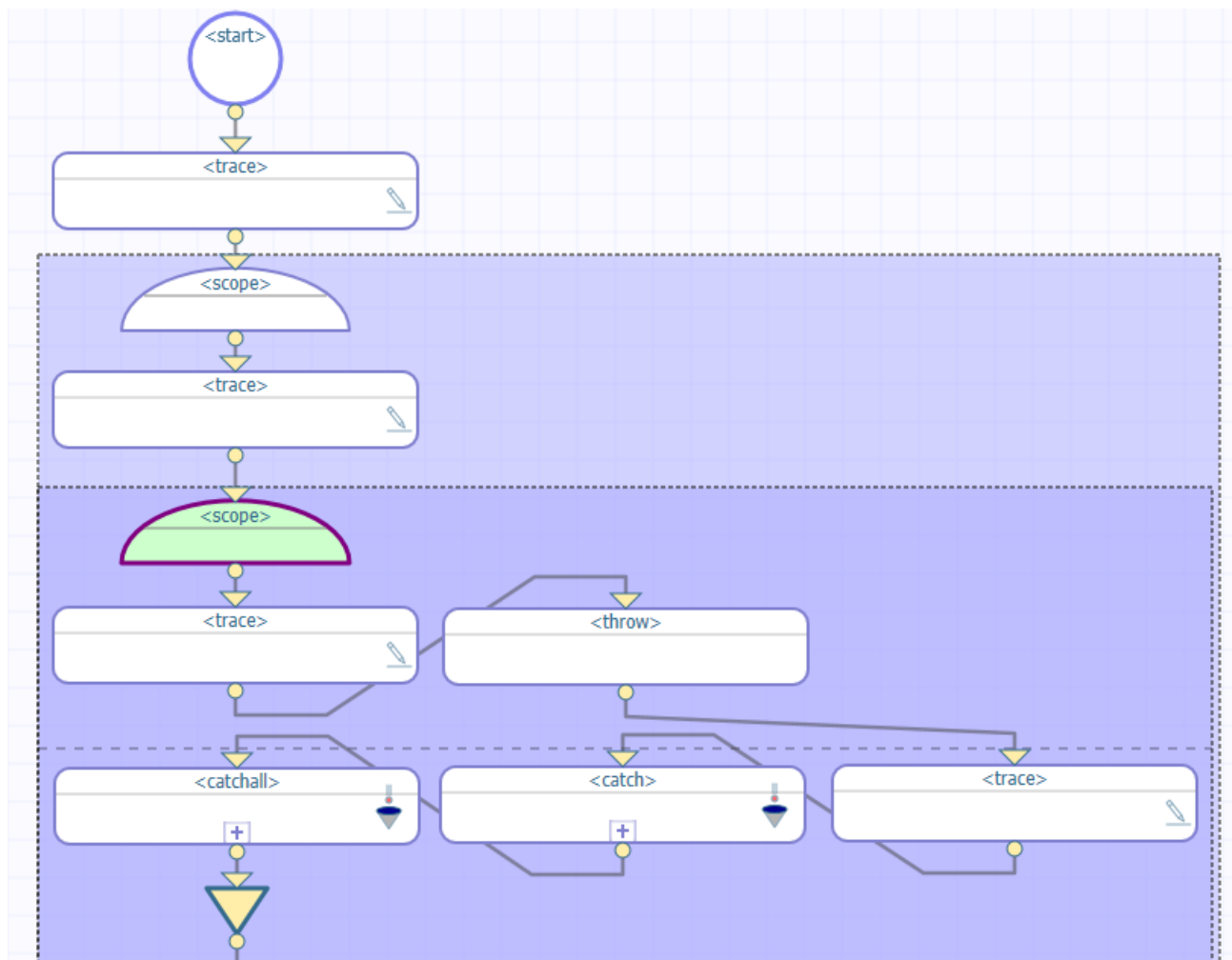
この BPL は以下の XData ブロックによって定義されます。

Class Member

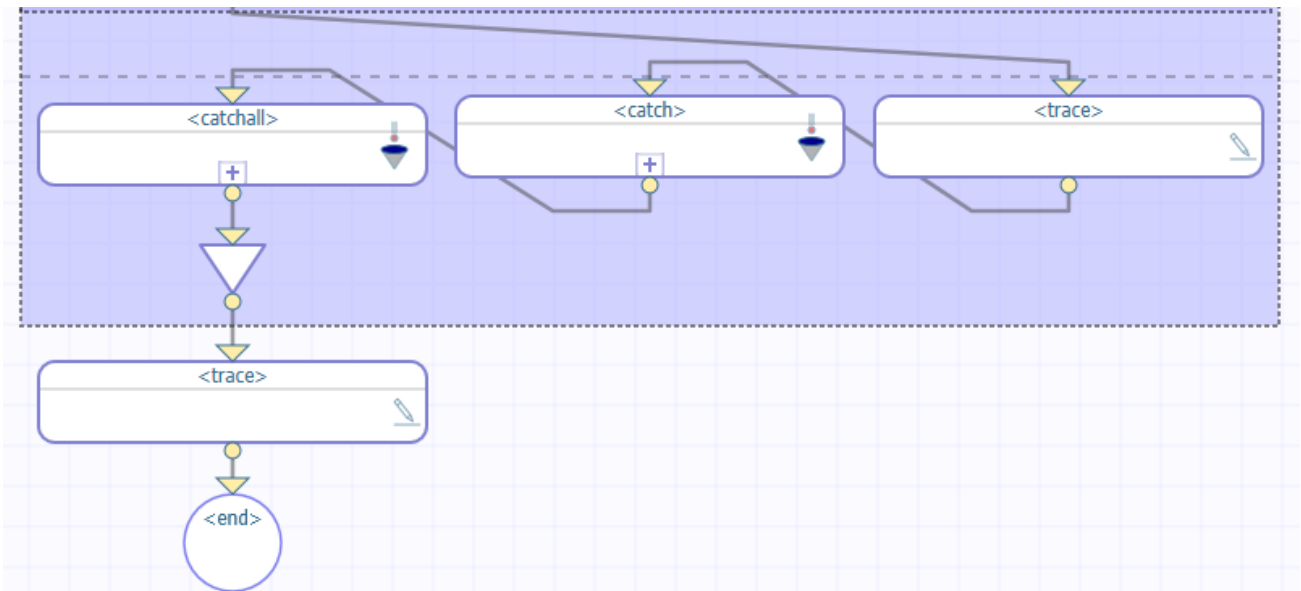
```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before outer scope' />
    <scope>
      <trace value='in outer scope, before inner scope' />
      <scope>
        <trace value='in inner scope, before assign' />
        <assign property='SomeProperty' value='1/0' />
        <trace value='in inner scope,after assign' />
        <faulthandlers>
          <catch fault='MismatchedFault'>
            <trace value=
              'In catch fault handler for &apos;MismatchedFault&apos;' />
          </catch>
        </faulthandlers>
      </scope>
      <trace value='in outer scope, after inner scope' />
      <faulthandlers>
        <catch fault='MismatchedFault'>
          <trace value=
            'In catch fault handler for &apos;MismatchedFault&apos;' />
        </catch>
      </faulthandlers>
    </scope>
    <trace value='after outer scope' />
  </sequence>
</process>
}
```

5.8 ネスト構造のスコープ – 外側のフォールト・ハンドラに Catch を配置

以下の BPL があるとしします (部分的に表示されています)。



この BPL の残りを以下に示します。

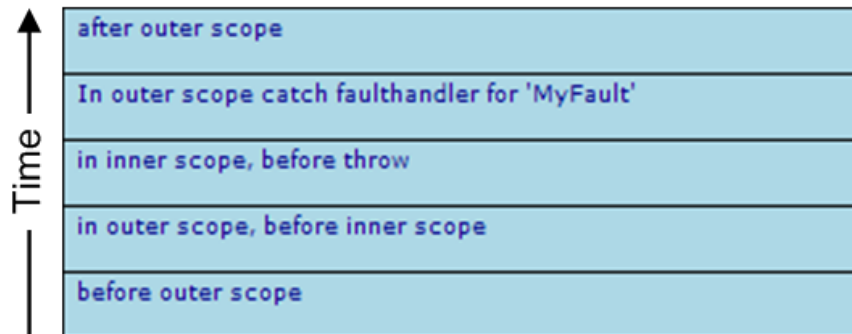


この BPL ビジネス・プロセスは以下の処理を実行します。

1. 最初の `<trace>` 要素がメッセージの `before outer scope` を生成します。
2. 最初の `<scope>` 要素は外側のスコープの始まりです。
3. 次の `<trace>` 要素がメッセージの `in outer scope, before inner scope` を生成します。
4. 2 つ目の `<scope>` 要素は内側のスコープの始まりです。
5. 次の `<trace>` 要素がメッセージの `in inner scope, before throw` を生成します。
6. `<throw>` 要素が特定の名前付きフォールト ("MyFault") をスローします。
7. これで、制御が内側の `<scope>` 内で定義された `<faulthandlers>` に移ります。`<catch>` は存在しますが、その `fault` 値は "MismatchedFault" です。内側のスコープには `<catchall>` がありません。
8. 制御が外側の `<scope>` 内の `<faulthandlers>` ブロックに移ります。このブロックには、`fault` 値が "MyFault" の `<catch>` が含まれます。
9. 次の `<trace>` 要素がメッセージの `in outer scope catch fault handler for 'MyFault'` を生成します。
10. 2 つ目の `<scope>` が終わります。
11. 最後の `<trace>` 要素がメッセージの `after outer scope` を生成します。

5.8.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



5.8.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

Class Member

```
XData BPL
{
<process language='objectscript'
  request='Test.Scope.Request'
  response='Test.Scope.Response' >
  <sequence>
    <trace value='before outer scope' />
    <scope>
      <trace value='in outer scope, before inner scope' />
      <scope>
        <trace value='in inner scope, before throw' />
        <throw fault='MyFault' />
        <trace value='in inner scope, after throw' />
        <faulthandlers>
          <catch fault='MismatchedFault'>
            <trace value=
              'In inner scope catch fault handler for &apos;MismatchedFault&apos;' />
          </catch>
        </faulthandlers>
      </scope>
      <trace value='in outer scope, after inner scope' />
      <faulthandlers>
        <catch fault='MyFault'>
          <trace value=
            'In outer scope catch fault handler for &apos;MyFault&apos;' />
        </catch>
      </faulthandlers>
    </scope>
    <trace value='after outer scope' />
  </sequence>
</process>
}
```

5.9 フォールトをスロー — 補償ハンドラで対応

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る必要があります。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

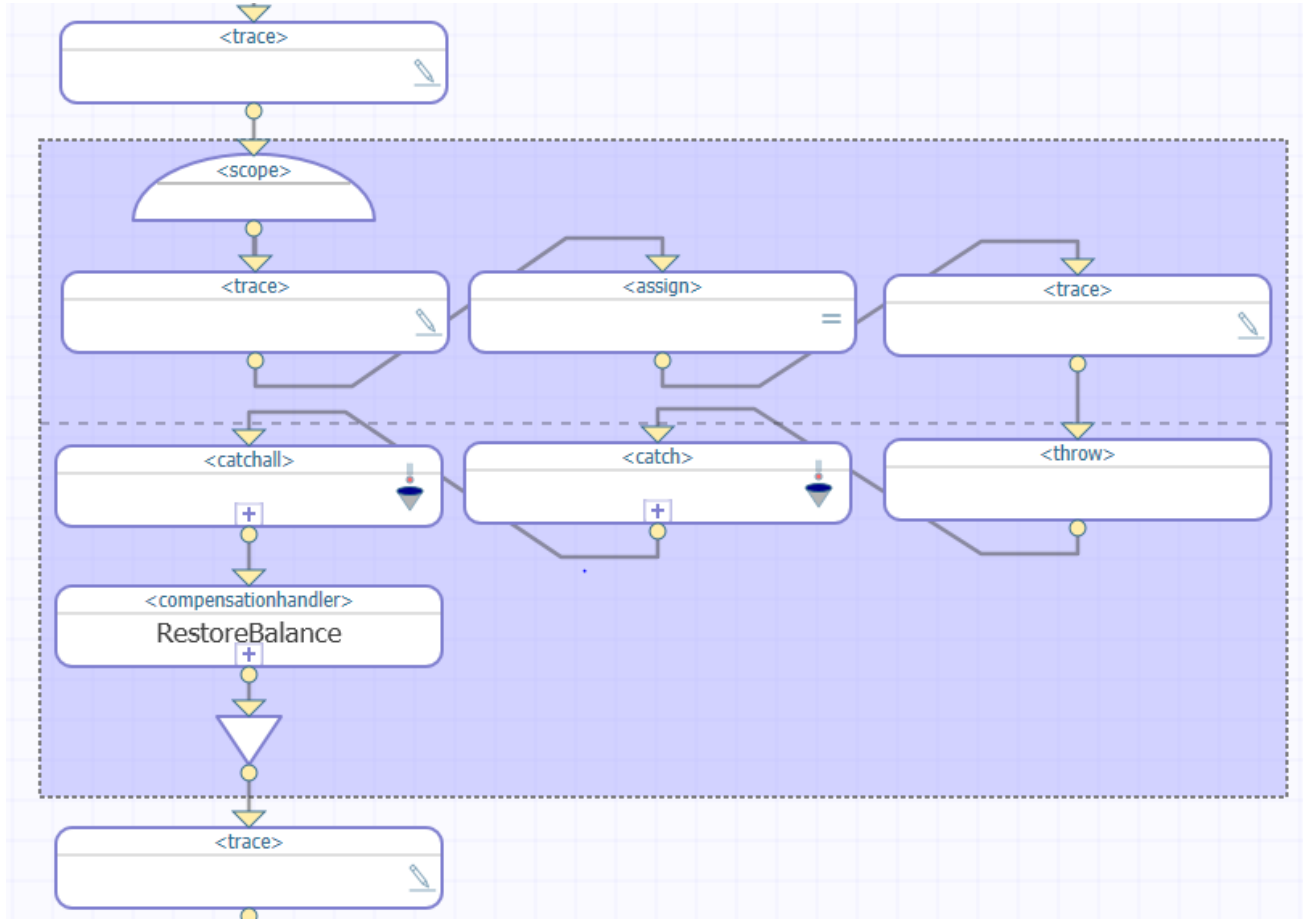
<compensationhandler> ブロックを呼び出すときは、ターゲットとしてその名前を指定します。この構文では、引用符を追加する必要がありません。

XML

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

補償ハンドラが役に立つのは、既に行わ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。

以下の BPL があるとします。

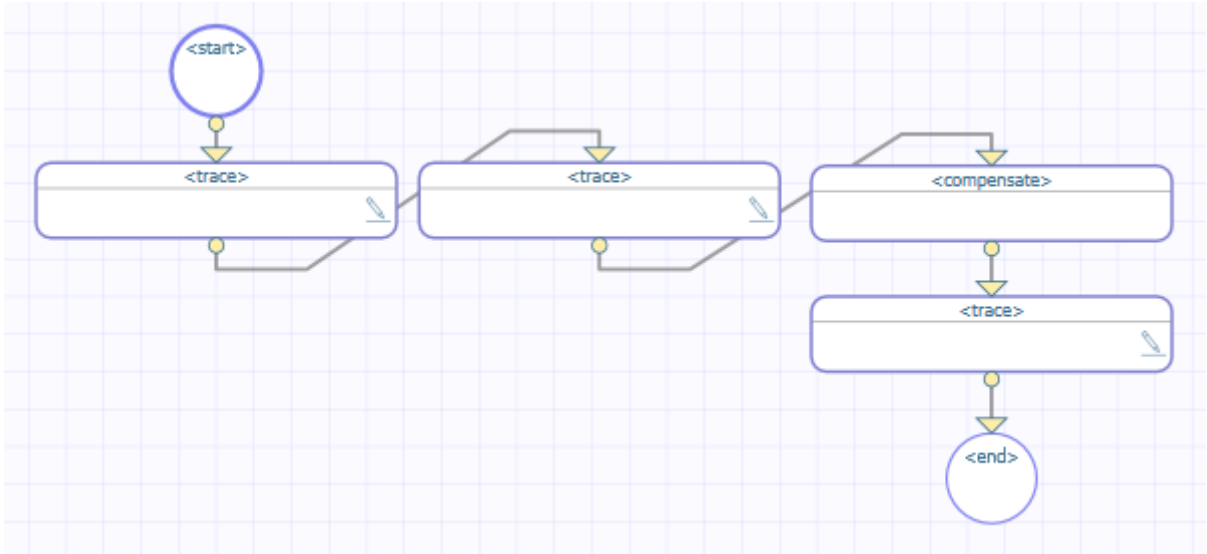


この BPL ビジネス・プロセスは以下の処理を実行します。

1. [コンテキスト] タブ (非表示) は、MyBalance という名前のプロパティを定義して、その値を 100 に設定します。
2. 最初の <trace> 要素がメッセージの before scope balance is を生成し、その後に MyBalance の値が続きます。
3. <scope> 要素はスコープの始まりです。
4. 次の <trace> 要素がメッセージの before debit を生成します。
5. <assign> 要素が MyBalance を 1 減らします。
6. 次の <trace> 要素がメッセージの after debit を生成します。
7. <throw> 要素が特定の名前付きフォールト ("BuyersRegret") をスローします。

8. これで、制御が <faulthandlers> に移ります。fault 値が "BuyersRegret" の <catch> が存在するため、ここに制御が移ります。

この <catch> 要素を掘り下げる場合は、以下を参照してください。



9. この <catch> 内部で、<trace> 要素がメッセージの `in catch faulthandler for 'BuyersRegret'` を生成します。
10. この <catch> 内部で、2 つ目の <trace> 要素がメッセージの `before restore balance is` を生成し、その後に MyBalance の現在値が続きます。
11. <compensate> 要素が使用されます。この要素の場合は、target が、name が RestoreBalance の <compensationhandler> です。この <compensationhandler> ブロックで以下の処理を実行します。
- ・ <trace> 文により、メッセージ “Restoring Balance” を出力します。
 - ・ <assign> 文を実行して、MyBalance を 1 増やします。

注釈 <compensationhandlers> と <faulthandlers> の順序を逆にはできません。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。

12. 次の <trace> 要素がメッセージの `after restore balance is` を生成し、その後に MyBalance の現在値が続きます。
13. <scope> が終わります。
14. 最後の <trace> 要素がメッセージの `after scope balance is` を生成し、その後に MyBalance の現在値が続きます。

5.9.1 イベント・ログ・エントリ

この場合、イベント・ログのエントリは以下ようになります。



5.9.2 この BPL 用の XData

この BPL は以下の XData ブロックによって定義されます。

Class Member

```
XData BPL
{
<process language='objectscript'
    request='Test.Scope.Request'
    response='Test.Scope.Response' >
    <context>
        <property name="MyBalance" type="%Library.Integer" initialexpression='100' />
    </context>
    <sequence>
        <trace value="before scope balance is "_context.MyBalance"/>
        <scope>
            <trace value="before debit"/>
            <assign property='context.MyBalance' value='context.MyBalance-1' />
            <trace value="after debit"/>
            <throw fault="BuyersRegret" />
            <compensationhandlers>
                <compensationhandler name="RestoreBalance">
                    <trace value="Restoring Balance"/>
                    <assign property='context.MyBalance' value='context.MyBalance+1' />
                </compensationhandler>
            </compensationhandlers>
            <faulthandlers>
                <catch fault="BuyersRegret">
                    <trace value="In catch fault handler for &apos;BuyersRegret&apos;"/>
                    <trace value="before restore balance is "_context.MyBalance"/>
                    <compensate target="RestoreBalance"/>
                    <trace value="after restore balance is "_context.MyBalance"/>
                </catch>
                <catchall>
                    <trace value="in catchall fault handler"/>
                    <trace value=
                        "%LastError " _
                        $System.Status.GetErrorCodes(..%Context.%LastError)_
                        " : "_
                        $System.Status.GetOneStatusText(..%Context.%LastError)'
                    />
                    <trace value="%LastFault " _..%Context.%LastFault"/>
                </catchall>
            </faulthandlers>
        </scope>
        <trace value="after scope balance is "_context.MyBalance"/>
    </sequence>
</process>
}
```

