



Informatica® Process Developer
April 2021

5. Process Developer

Informatica Process Developer 5. Process Developer

April 2021

2021 年 4 月

© 著作権 Informatica LLC 1993, 2021

発行日: 2021-08-10

目次

序文.....	18
第 I 部 : Process Developer の使用.....	19
第 1 章 : Process Developer のインストールおよびアクセス.....	20
第 2 章 : Process Developer の基本操作.....	21
Process Developer コンポーネント.....	21
ヒント.....	22
ソース管理システムの使用.....	32
Process Developer のヘルプへのアクセス.....	33
第 3 章 : ビジネスプロセス実行言語の概要.....	34
BPEL プロセスについて.....	34
BPEL プロセス定義要素.....	36
WS-BPEL 2.0 に対する Informatica 拡張機能.....	37
第 4 章 : Process Developer の設定.....	39
タスクタグの追加.....	40
B ユニットの設定.....	40
キャッシュとタイムアウトの設定.....	41
色とフォントの設定.....	41
コントリビューションの設定.....	42
ID 選択の設定.....	42
レイアウトの設定.....	43
[リレーション] ビューの設定.....	45
タスクと問題の設定.....	46
追加の設定.....	46
第 5 章 : BPMN 中心および BPEL 中心の編集スタイル.....	47
BPMN 中心のスタイルについて.....	48
BPEL 中心のスタイルについて.....	48
BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較.....	50
どちらの編集スタイル (BPMN 中心または BPEL 中心) を選択するかについて.....	52
スイムレーンの使用.....	52
Process Developer Classic スタイルについて.....	54
特定の編集スタイルから別の編集スタイル (BPMN およびクラシック) への保存.....	58
第 6 章 : Process Developer のユーザーインターフェース.....	59
Process Developer のメニューとツールバー.....	59

リファクタリング	60
アクティビティに移動 (Ctrl + l)	61
操作を開く (Ctrl + Shift + o)	61
ポートタイプを開く (Ctrl + Shift + t p)	61
Web タイプを開く (Ctrl + Shift + t w)	61
パラメータの最適化	61
プロセスの検証	62
プロセスレポートの生成	62
プロセスイメージの生成	62
デプロイメントイメージの生成	63
Process Editor のキーボードショートカット	63
Process Developer のファンクションキー	64
ウィンドウ、パースペクティブ、ビュー、およびエディタ	65
Process Developer のパースペクティブ	65
プロジェクトエクスプローラ	66
パーティシパント	67
[イメージの場所] ダイアログ	67
インタフェース	67
[アウトライン] ビュー	67
[プロパティ] ビュー	68
プロセス変数	68
エラーログ	69
[問題] ビュー	70
[タスク] ビュー	71
[サムネイル] ビュー	71
ブックマークビュー	72
[リレーション] ビュー	72
[サーバー] ビューと [コンソール] ビュー	73
ステータスバー	73
Process Developer パースペクティブのカスタマイズ	73
Process Developer の [デバッグ] パースペクティブ	73
Process Developer Process Editor の使用	73
Process Editor の [プロセスアクティビティ] タブ	74
Process Editor の [フォールトハンドラ] タブ	75
Process Editor の [イベントハンドラ] タブ	75
Process Editor の [補償] タブと [終了ハンドラ] タブ	75
Process Editor の [ソース] タブ	75
視覚的なプロパティの設定と独自のイメージライブラリの使用	76
プロセスへのタスクとブックマークの追加	78
プロセスへのコメントの追加	79
プロセスへのドキュメントの追加	79
Process Editor キャンバスで設計するためのヒント	80

アクティビティの表示と非表示.	81
第 7 章 : ワークスペース、オーケストレーション、プロセス.	83
Eclipse 環境の使用.	83
Process Developer の起動.	83
ワークスペースを使用したプロジェクトの保存.	84
Orchestration プロジェクトの作成.	85
ワークスペースプロジェクトのデフォルトの場所の変更.	86
Orchestration プロジェクトのテンプレート.	86
プロジェクト Orchestration ネーチャーの追加または削除.	88
プロジェクトのオーケストレーションおよび検証ビルダについて.	88
プロジェクト参照の使用.	89
新しいプロセスの作成.	90
新しいプロセスの開始.	90
BPMN 中心のパレットまたは BPEL 中心のパレットの選択.	93
既存の BPEL プロセスのインポート.	94
Visio XML 図面ファイルのインポート.	94
Process Developer を利用した最初のプロジェクトの開始.	95
組み込みプロセスサーバーのセットアップ.	96
第 II 部 : プロセスの作成と変更.	98
第 8 章 : インタフェース、サービス参照、ローカル WSDL.	99
インタフェース、サービス参照、ローカル WSDL について.	99
ローカル WSDL のインポート.	100
WSDL ツリーのキー要素の表示.	101
WSDL エディタでの WSDL の編集.	101
プロジェクトからの WSDL の削除.	101
サービス参照のインポート.	101
新しいインタフェースの作成.	102
Java インタフェースの作成.	103
Process Developer での Java プロジェクトのセットアップ.	104
Java プロジェクトの制約.	105
Java インタフェースからの WSDL とスキーマの生成.	105
スキーマ要素の引数名の生成.	107
Java プロジェクトと BPEL プロセスの同時更新.	108
Java (POJO) エンドポイントのデプロイメント要件.	109
Java (EJB) エンドポイントのデプロイメント要件.	109
POJO EJB エンドポイントを使用した BPEL プロセスの実行.	110
POJO/EJB インタフェースとカスタム Java 呼び出しハンドラの比較.	110
[インタフェース] ビューを使用したアクティビティの作成.	111
インタフェースツールバーオプション.	111
[インタフェース] ビューのフィルタリング.	111

システムサービスインタフェース.	112
BPEL 用の拡張機能を持つ WSDL ファイルの作成.	113
WSDL メッセージのサンプルデータの作成.	113
単純型のメッセージパートへのサンプルデータ値の追加または編集.	113
サンプルデータファイルの生成.	114
WSDL メッセージへのサンプルデータファイルの追加.	115
デフォルトのサンプルデータファイルの選択.	115
サンプルデータファイルの XML 構造の表示.	115
サンプルデータファイルの削除.	115
プロセス変数のサンプルデータの選択.	116
WSDL コンポーネントの使用箇所の検索.	116
プロセス検索の使用.	117
検証エラーメッセージダイアログ.	118
サンプルデータファイルのエラーの表示.	119
第 9 章 : BPEL プロセス.	120
トップダウンまたはボトムアップのプロセス設計の使用.	120
効率的な設計のための WSDL 参照の使用.	121
BPEL 用の WSDL 拡張機能の作成.	121
Process Editor への操作のドロップによるプロセスの開始.	121
WSDL、スキーマ、およびその他のリソースのインポート.	122
WSDL およびスキーマの場所の自動インポート.	122
WSDL、スキーマ、およびその他のリソースの手動でのインポート.	122
インポートの更新.	123
インポートの削除.	124
名前空間のプレフィックスと宣言.	124
名前空間の削除.	124
拡張機能の宣言.	124
Process Developer の XPath 作成拡張機能の使用.	125
Process Developer の [選択の無効化エラー] 拡張機能の使用.	126
拡張要素と属性の宣言.	126
BPEL プロセスの構造とプロパティの理解.	129
プロセス要素とプロパティ.	130
アクティビティ.	132
BPEL XML ソースと暗黙的に追加されたアクティビティ.	132
補償ハンドラ.	133
式言語.	133
フォールトハンドラ.	133
パートナーリンク.	133
変数.	133
BPEL プロセスのライフサイクルの理解.	134
実行可能なプロセスと抽象的なプロセス.	134
抽象プロセスの作成.	134

抽象プロセスを使用するためのヒント.	135
別の BPEL プロセスのサービスとしての BPEL プロセスの作成.	135
メッセージ交換の宣言.	136
第 10 章: パーティシパント.	138
パーティシパントについて.	138
[パーティシパント] ビューの使用.	139
新しいプロセスサービスコンシューマインタフェースの作成.	140
新しいパートナーサービスインタフェースの作成.	141
新しいコールバックインタフェースの作成.	142
パーティシパントからのサービスインタフェースのクリア.	143
[パーティシパント] ビューからの新しいアクティビティの作成.	143
アクティビティの [プロパティ] ビューからの新しい変数の作成.	144
パートナーリンクタイプとパートナーリンクについて.	144
パートナーリンクタイプ.	145
プロジェクトエクスプローラでの WSDL からの新しいパートナーリンクタイプの追加.	146
サービス参照 WSDL を使用した新規 WSDL への新たなパートナーリンクタイプの追加.	147
[インタフェース] ビューからの新しいパートナーリンクタイプの追加.	147
パートナーリンク.	149
スコープされたパートナーリンクの使用.	151
パートナーリンクとエンドポイント参照.	151
抽象プロセスへのパートナーリンクのエクスポート (非推奨)	151
第 11 章: BPMN タスクおよびイベント.	153
BPMN から BPEL へのタスクとイベントの実装.	153
BPEL アクティビティの概要.	155
アクティビティとそのプロパティの定義.	156
アクティビティのプロパティの値の選択.	158
アクティビティラベルの選択.	158
アクティビティの標準プロパティ.	159
背景色の追加.	160
WSDL インタフェースから開始するアクティビティの作成.	160
カスタムアクティビティの作成.	164
アクティビティシーケンスおよびフローの理解と使用.	165
BPMN の設計のヒント.	166
割り当て.	173
コピー操作のヒント.	177
コピー操作のクエリと式の例.	177
コピー操作のリテラルコンテンツの例.	178
コピー操作の動的エンドポイントの参照例.	181
ソース要素名属性を維持する要素間のコピー操作.	182
[見つからない開始データを無視] の属性を使用したコピー操作.	183
Break.	184

Compensate.	187
Compensate Scope.	189
Continue.	191
Empty.	192
終了.	193
Invoke.	194
開始パートから変数.	197
変数からパート.	197
入力変数.	197
出力変数.	199
あいまい.	200
Receive.	201
Reply.	203
Rethrow.	206
シグナル.	208
Start/End/None.	209
Suspend.	209
Throw.	210
検証.	212
Wait.	214
第 12 章 : ゲートウェイと制御フロー.	217
BPMN から BPEL へのゲートウェイの実装と制御フロー.	217
アクティビティを構造化するさまざまな方法.	218
制御フロー項目の概要.	219
選択した構造化アクティビティのグループ化解除.	219
ゲートウェイ.	219
ゲートウェイタイプについて.	220
相互に排他的な遷移.	221
ゲートウェイの構築.	221
Flow.	222
For Each.	223
Fork Join.	227
If.	228
Pick.	231
Repeat Until.	233
Sequence.	234
Scope.	235
スコープ内で分離を「はい」にする設定.	238
スコープへの終了ハンドラの使用.	238
スコープのライフサイクル.	239
While.	239

第 13 章 : 変数	241
変数の概要	241
変数の追加	243
WSDL メッセージタイプ	243
XML スキーマタイプ	244
XML スキーマ要素	244
変数プロパティとプロパティエイリアスの追加	245
変数の初期化	246
変数の表示	246
アクティビティで使用される変数のクイックビュー	247
[プロセス変数] ビューのオプションの使用	247
定義を表示するために変数を開く	248
変数プロパティの表示	248
アイコンの記号の理解と変数パートの説明	248
変数の削除	249
[プロセス変数] ビューでのサンプルデータの使用	249
単純型のメッセージパート用の単一のサンプルデータ値の編集	250
[プロセス変数] ビューでのサンプルデータファイルのロード	250
[プロセス変数] ビューでのサンプルデータの保存と表示	251
XML データウィザードの使用	251
変数の使用箇所の検索	255
コピー操作での変数の使用	256
コンテキストメニューを使用したコピー操作の作成	256
ドラッグアンドドロップを使用したコピー操作の作成	256
編集するコピー操作の選択	257
WSDL フォールトメッセージに基づく変数の使用	257
Web サービスインタラクションアクティビティでの WSDL メッセージパートのマッピング	257
変数の検証	258
変数の添付ファイルの使用	258
第 14 章 : 添付ファイル	259
添付ファイルの追加	259
シミュレーション用の添付ファイルの追加	259
添付ファイルを操作するためのカスタム関数	260
添付ファイルのカスタム関数の例	263
第 15 章 : リンク	264
リンクについて	264
リンクに対する Process Developer 拡張機能	265
アクティビティ間へのリンクの追加	266
遷移のないリンクの追加	266
遷移条件付きのリンクの追加	267

[リンクの追加] ダイアログの使用.	268
リンクの例.	269
リンクの実行ルール.	270
リンクを使用した設計と構造化されたアクティビティ.	270
リンクと結合条件.	271
プロパティのリンク.	271
第 16 章 : データ操作.	272
BPEL でのデータ操作の概要.	272
式構築のための XPath または XQuery の選択.	273
XQuery 式の例.	273
XPath 式の例.	275
式ビルダの使用.	275
コンテンツアシストの使用.	277
BPEL 関数.	277
Process Developer のカスタム関数全般.	279
添付ファイル関数.	281
ブール関数.	281
カタログ関数.	282
フォールト関数.	284
I18N 関数.	284
JSON 関数.	285
ノードセット関数.	286
数値関数.	287
文字列関数.	288
条件カウンタおよびその他の値に想定される式.	289
クエリビルダの使用.	290
入力リンクに対する結合条件の作成.	292
期限と期間の式.	293
第 III 部 : 関数、イベント、エラー、および関連.	294
第 17 章 : POJO および XQuery カスタム関数.	295
カスタム関数の概要.	295
関数コンテキストの実装と注釈の追加.	297
手順 1: 関数コンテキストの作成.	297
手順 2: カスタム関数の作成.	298
手順 3: パッケージの作成.	298
カスタム関数のサンプル.	298
プロセスサーバーへのグローバルカスタム関数の追加.	299
XQuery 関数の記述.	299
XQuery エディタの使用.	301
XQuery 関数を作成するためのヒント.	302

XQuery エディタでの XQuery 関数のテスト	302
第 18 章: システムサービス	303
システムサービスベースのプロセスへの BPEL テンプレートの使用	303
警告サービス	304
データアクセスサービス	305
データソースでステートメントを実行する BPEL プロセスの作成	305
データアクセス呼び出し出力のシミュレーション	307
例 - Insert Update または Delete ステートメントからの応答	310
データアクセスサービスのシミュレーション	311
電子メールサービス	311
ID サービス	313
移行サービス	315
マップ作成の入力と出力	315
移行のための入力と出力	316
プロセスインスタンスを新しいバージョンに移行する BPEL プロセスの作成	316
移行プロセスのテスト	317
プロセスサーバー移行 Web サービスの使用	317
シェルコマンドを呼び出す BPEL プロセスの作成	318
BPEL プロセスを使用した警告の監視サービスの使用	318
BPEL プロセスを使用した再試行ポリシーサービスの使用	319
BPEL プロセスを使用したサービスログサービスの使用	321
詳細情報	322
第 19 章: マイロールのエンドポイントおよびプロセスコンシューマ	323
SOAP とポリシー駆動型バインディング	323
Java からのプロセスの呼び出し	325
[匿名アクセスを許可] ポリシーの設定	328
第 20 章: カスタムサービスのインタラクション	329
REST ベースのサービスの使用	329
REST ベースの受信または呼び出しの作成	330
BPEL REST メッセージ	332
マルチパート HTTP メッセージの処理	333
REST ベースのプロセスのデプロイメントに関する詳細の指定	333
OAuth REST ベースのシステムサービスの使用	334
OAuth サービスの作成	335
OAuth サービスプロバイダのデプロイメントに関する詳細の指定	336
Java Messaging Service 呼び出しハンドラの使用	338
第 21 章: カスタム呼び出しハンドラ	342
[カスタム呼び出しハンドラ] インタフェースの参照	343
PDD へのカスタム呼び出しハンドラ属性の追加	344

EJB としてのカスタム呼び出しハンドラのパッケージ化.	345
カスタム呼び出しハンドラファイルのデプロイ.	346
第 22 章 : ビジネスイベント処理.	347
プロセスデプロイメント記述子でのイベントの定義.	347
システム定義のイベントの使用.	348
Event-Action BPEL プロセスの作成.	350
アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ. . . .	351
第 23 章 : イベント処理.	355
イベント処理について.	355
イベントハンドラの追加.	356
onEvent イベントハンドラの追加.	356
onAlarm イベントハンドラの追加.	358
イベントの処理ルール.	361
境界イベントの追加.	361
Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow.	365
Catch イベントまたは中断 OnEvent 境界イベントからの変数の使用.	365
第 24 章 : 相関.	367
相関について.	367
相関セットについて.	368
プロパティ.	368
プロパティエイリアス.	369
プロパティ名およびエイリアスの WSDL 構文と例.	369
グローバル相関セットとローカル相関セット.	370
メッセージプロパティとプロパティエイリアスの作成.	371
プロパティ定義の作成.	371
プロパティエイリアスの作成.	374
相関セットの追加.	376
相関セットの削除.	378
アクティビティへの相関の追加.	378
相関のパターンの開始と設定.	378
Receive、OnMessage、OnEvent、または Reply への相関の追加.	380
Invoke アクティビティへの相関の追加.	381
欠落した相関の追加.	382
相関セットを宣言および使用するためのルール.	382
相関セットおよびエンジンによって管理される相関.	382
第 25 章 : フォールト処理.	383
BPEL フォールト処理について.	383
Catch および CatchAll フォールトハンドラの定義.	384
サービス呼び出しのフォールト処理.	386

フォールト処理の処理ルール.	387
フォールトハンドラの追加.	388
プロセスへのフォールトハンドラの追加.	389
スコープへのフォールトハンドラの追加.	390
フォールト変数定義の追加.	392
フォールト名の選択.	392
呼び出しアクティビティの境界イベントとしてのフォールトハンドラの追加.	393
Catch アクティビティでフォールトをキャッチするためのルール.	393
フォールト処理に関するヒント.	394
宣言されていないフォールトと SOAP フォールトのキャッチ.	395

第 26 章 : プロセス例外管理. 398

プロセス例外管理について.	398
キャッチされないフォールト時のプロセスの一時停止.	399
キャッチされないフォールト時のすべてのプロセスの一時停止の対象化.	399
キャッチされないフォールト時の個々のプロセスの一時停止の対象化.	399
Suspend アクティビティによる、プログラムでのプロセスの一時停止.	399

第 27 章 : 補償. 400

補償について.	400
補償ハンドラと Compensate アクティビティ.	402
デフォルトの順序の補償の例.	402
指定した補償の例.	403
スコープへの補償ハンドラの追加.	403
呼び出しアクティビティの補償.	405

第 IV 部 : テストとデプロイメント. 406

第 28 章 : シミュレーションとデバッグ. 409

Process Developer の [デバッグ] パースペクティブについて.	409
Process Developer の [デバッグ] パースペクティブを開く.	410
Process Developer のパースペクティブの切り替え.	410
Process Developer の [デバッグ] パースペクティブビューとメニュー.	410
Process Developer の [デバッグ] ビューの使用.	411
BPEL プロセスシミュレーションでのブレークポイントの使用.	413
Process Developer デバッグコンソールの使用.	415
BPEL プロセスの実行のシミュレーション.	416
シミュレーションの前提条件.	417
BPEL プロセスのシミュレーションの開始と終了.	417
BPEL プロセスでのブレークポイントまでの実行.	418
BPEL シミュレーションの次のアクティビティへのステップ実行.	419
アクティビティまたはリンクの実行状態の表示.	419
シミュレーション中の BPEL プロセスの変更.	420

BPEL プロセスシミュレーションの終了と削除.	420
プロセス実行状態のクリア.	421
シミュレーション中のサンプル変数データの入力および検査.	421
入出力メッセージおよびフォールトメッセージのサンプルデータ値の設定.	422
シミュレーション中のプロセス変数の検査.	423
シミュレーションパスとプロパティの選択.	424
シミュレーション用の呼び出しサブプロセスの選択.	426
イベントハンドラのシミュレーション.	426
フォールトハンドラのシミュレーション.	426
シミュレーション中の標準フォールトの検査.	427
シミュレーションの設定.	427
bpel:selectionFailure フォールトの無効化の例.	428
Copy/To のターゲットパスの自動作成の例.	429
bpel:selectionFailure フォールトの無効化の例と Copy/To のターゲットパスの自動作成の例	430
デバッグの設定.	431

第 29 章 : BPEL ユニットテスト. 433

BPEL ユニットテストについて.	433
BPEL ユニットテストファイルの作成.	434
Process Developer での BPEL ユニットテストの実行.	435
B ユニット Ant スクリプトの作成と実行.	437
前提条件.	437
B ユニット Ant スクリプトの作成.	437
B ユニット Ant スクリプトの変更にに関する注意事項.	438
Ant スクリプトへの複数の B ユニットと B スイートの追加.	438
B ユニット Ant スクリプトの実行.	438
コード範囲レポートの生成.	438
B ユニットファイルの編集.	440
BPEL ユニット（ルート）.	441
拡張機能と拡張アクティビティ.	441
呼び出し.	442
アラーム.	443
コマンド.	444
B ユニットテストのデバッグ.	445
BPEL ユニットテストスイートの作成と実行.	445
アサーションの使用に関するヒント.	447
入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント.	448
パートナーリンクデータの指定に関するヒント.	448
B ユニットファイルの例.	449

第 30 章 : プロセスデプロイメント. 451

デプロイメントの準備.	451
デプロイメント用の BPEL ファイルの準備.	452

デプロイされたプロセス用のサーバープラットフォームの選択.	452
エンドポイント参照のアドレス指定の考慮事項.	452
エンドポイント参照と WS-Addressing の考慮事項.	452
エンドポイント参照と Informatica Cloud.	453
アクセス時に資格情報が必要なエンドポイント参照.	454
エンドポイント参照の置換可能な URN と URL の指定.	455
エンドポイント参照と WS-Policy.	455
プロセスデプロイメントの手順の概要.	455
プロセスデプロイメント記述子ファイルの作成.	456
全般的なデプロイメントオプション.	457
パートナーロールの呼び出しハンドラ.	461
パートナーロールのエンドポイントタイプ.	463
プロセスサービスの選択.	464
EJB/Java 呼び出しハンドラのプロパティダイアログ（オンプレミスのみ）.	464
マイロールのバインディングサービス名と許可されるロールオプション.	465
デプロイメント記述子パートナーリンク用のサービスの選択.	466
SOAP 1.1 または SOAP 1.2 のポートバインディングの使用.	467
ポリシーアサーションの追加.	468
インデックス付きプロパティの追加.	482
参照の表示.	482
PDD のイベントタブ.	482
PDD の [ユーザー] タブ.	482
PDD エディタの [ソース] ビューの使用.	483
ビジネスプロセスアーカイブコントリビューションの作成とデプロイ.	483
プロジェクトの依存関係のデプロイと除外された依存関係の表示.	485
追加リソースのデプロイ.	487
Informatica Cloud へのデプロイメント.	488
デプロイメントコントリビューションの管理.	492
BPRD スクリプトを使用した BPR コントリビューションの再生成およびデプロイ.	493
Process Developer 内からの BPRD Ant スクリプトの実行.	493
コマンドラインからの BPRD Ant スクリプトの実行.	494
デプロイメントの完了.	495
サーバーの起動とプロセスの実行.	496
BPEL プロセスをインスタンス化する方法.	496
プロセスのバージョンングについて.	497
第 31 章: エンドポイントと URL.	499
サービスエンドポイント.	499
プロセスサーバーの URL.	501
プロセスサーバーの API と SDK.	501
第 32 章: セキュリティ構成.	503
SAML で保護されたサービスの認証の設定.	504

プロセスサーバーコンポーネントの保護.	504
プロセスサーバーのセキュリティロールを操作するためのアプリケーションサーバーの設定.	506
Process Developer のオーケストレーションファイルリソース.	406
BPEL 標準フォールト.	407

第 V 部: プロセスセントラルとプロセスサーバー (オンプレミス) ... 510

第 33 章: プロセスセントラルフォームおよび設定. 511

プロセスセントラルについて.	511
プロセスセントラルプロセスフォームの作成.	512
プロセス要求フォームテンプレートの理解.	513
プロセスセントラルフォームでの HTML の編集.	514
フォームまたはタスクへの新しいサービス操作の追加.	515
タスクおよびフォームスクリプトのカスタマイズの概要.	516
フォームをカスタマイズするための Informatica Business Process Manager SDK の使用.	519
プロセスセントラルフォームのテストとデバッグ.	519
プロセスセントラル構成ファイルの作成.	520
フォームフィルタの設定.	521
レポートフィルタの設定.	522
プロセスセントラル用の独自のスタイルのスクリプトとメタデータを含める場合.	523
タスクロールフィルタの設定.	524
基本的なタスクロールのフィルタ構成.	525
タスクロールのフィルタリング用のカスタムカラムの設定.	526
WhereClause を使用した、GetMyTasks フィルタの設定.	528
getTasks と orderBy 要素の使用.	530
RSS または Atom フィードフィルタの設定.	530
フォームレポートと構成ファイルのデプロイ.	531
プロセスセントラルへの多言語サポートの追加.	532
Web ブラウザでの優先言語によるプロセスセントラルの表示.	533
フォームとタスクへの多言語サポートの追加.	533
.avcconfig ファイルへの多言語サポートの追加.	534
レポートへの多言語サポートの追加.	535
.properties ファイルの命名規則.	536
プロセスセントラルの詳細設定.	536

第 34 章: プロセスサーバーおよびプロセスセントラルレポート. 538

ユーザーレポート Orchestration プロジェクトの作成.	539
Process Developer レポートテンプレートの使用.	539
Process Developer データソースの使用.	541
Process Developer データソースからのデータセットの作成.	544
Process Developer データモデルの理解.	545
Process Developer レポートのデプロイ.	545
デプロイされたレポートの更新または削除.	546

Reporting Service. 546

序文

このガイドでは、Eclipse ベースのビジネスプロセスオーケストレーションツールである Process Developer を使用して、BPEL プロセスを作成、シミュレート、デプロイ、実行する方法について説明します。

パート I: Process Developer の使用

このセクションのトピックでは、Process Developer のユーザーインターフェースとその中で実行可能なアクションについて説明します。

第 1 章

Process Developer のインストールおよびアクセス

次のいずれかのオプションを使用して、Process Developer をダウンロードします。

- Process Developer をスタンドアロンの実行可能ファイルとして使用する場合は、[Informatica Process Developer \(Microsoft Windows\)](#) をクリックします。ダウンロード後にファイルを解凍し、.exe ファイルをダブルクリックして Process Developer をインストールします。
- Process Developer を Eclipse のプラグインとして使用する場合は、[Informatica Process Developer Eclipse Plug-ins](#) をクリックします。ダウンロード後、通常 Eclipse プラグインをインストールするときの手順に従います。

次の資格情報を使ってファイルをダウンロードします。

- ユーザー名: **Hotfix**
- パスワード: **Hotfix1!**

Informatica Process Developer for Windows のインストール後、システム上の Process Developer を起動します。Process Developer を Eclipse のプラグインとしてインストールした場合は、Eclipse を開いてから Process Developer にアクセスします。。

第 2 章

Process Developer の基本操作

Process Developer の詳細については、Informaticaネットワークの [Process Developer Tutorials](#) を参照してください。

次のトピックでは、Process Developer の基本について説明します。

Process Developer コンポーネント

Process Developer には、次のコンポーネントが含まれています。

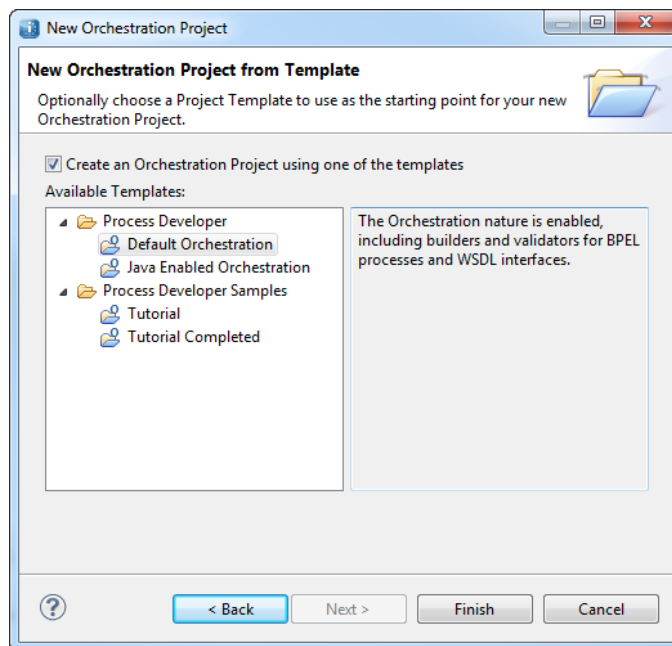
- **Process Developer**
Process Developer を使用すると、BPEL プロセスを作成、シミュレート、デプロイ、および実行できます。
- **Orchestration プロジェクトサンプル**
Process Developer チュートリアルおよびその他のオーケストレーションのサンプルには、基本的なプロセスと特別なプロセスを開始するためのファイルが含まれています。
- **デプロイメントおよびテストスクリプトのコマンドライン実行用の Ant ランタイム**
インストールウィザードを使用すると、B ユニットランタイムを選択して、Process Developer の外部で Process Developer スクリプトを実行するために必要な Eclipse プラグインをインストールできます。これらのスクリプトには、BPRD ファイルと B ユニット Ant ファイルが含まれます。BPRD スクリプトには、プロセスサーバーへのデプロイメントのターゲットが含まれています。B ユニットスクリプトには、BPEL プロセスを単体テストするためのターゲットが含まれています。

ヒント

すべてのリソースを保持する Orchestration プロジェクトの作成

Orchestration プロジェクトには、Process Developer がプロセスの構築に使用するフォルダのコレクションが含まれています。ビルダーとバリデータによって、プロセスを作成する際に正しい方向に進んでいるかどうかを確認します。

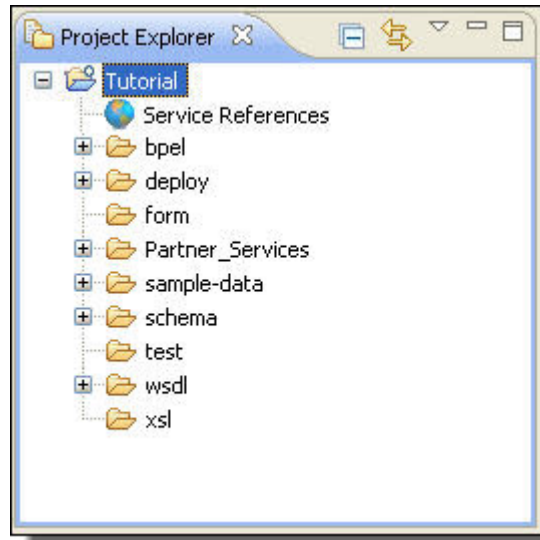
Process Developer サンプルプロジェクトから開始します。



サービス参照とパーティシパントを使用した新しい BPEL プロセスの迅速な開始方法

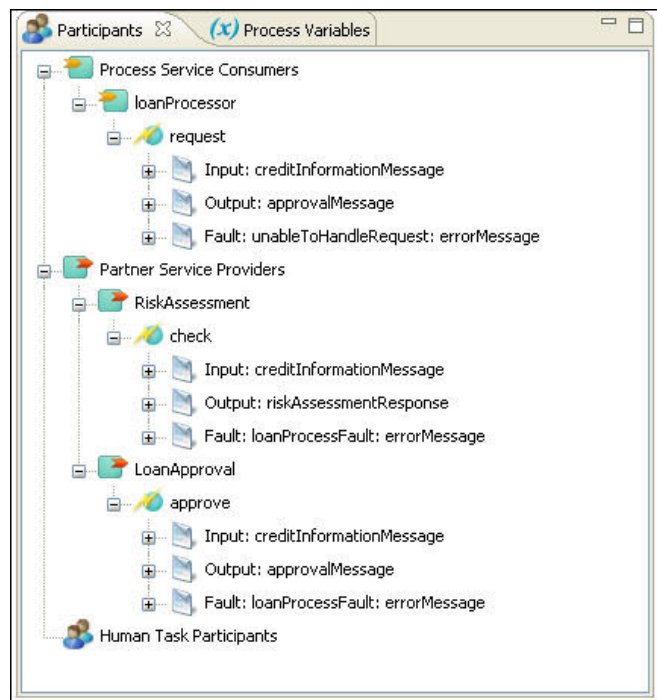
BPEL アクティビティを作成する前に、WSDL ファイルをサービス参照に追加するか、Orchestration プロジェクトに直接追加します。

Process Developer で、自動化されたアクティビティ作成のために WSDL ファイルがカタログ化されます。



パーティシパント

プロセスを使用する Web サービスとプロセスが使用するパートナーサービスを定義します。これは、プロセスを開始するための迅速かつ直感的な方法です。



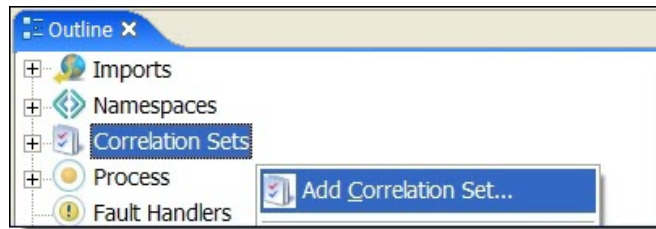
パーティシパント用のアクティビティの作成

パーティシパントの操作を Process Editor キャンバスにドラッグして、アクティビティを作成します。アクティビティはシーケンス内に含まれているため、新しいアクティビティを順番に追加するのが容易になります。



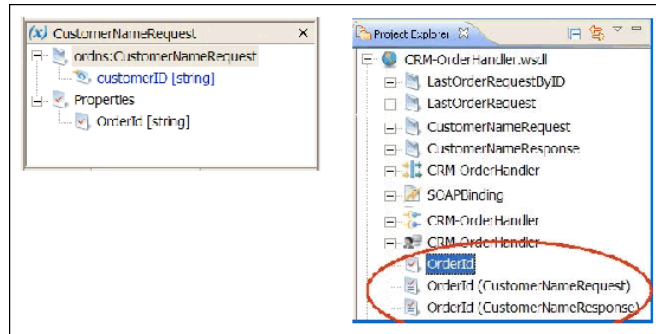
メッセージプロパティとプロパティエイリアスの作成

メッセージプロパティは、BPEL の WSDL 拡張機能で、長時間実行される非同期プロセスのメッセージを相互に関連付けることができます。[相関セットの追加] を選択して、メッセージプロパティとプロパティエイリアスを作成します。



メッセージプロパティとプロパティエイリアスを作成すると、プロセス変数と WSDL ファイルに追加されます。

次に、相関セットを作成し、メッセージを相関させる必要がある各アクティビティにそのプロパティとエイリアスを追加します。

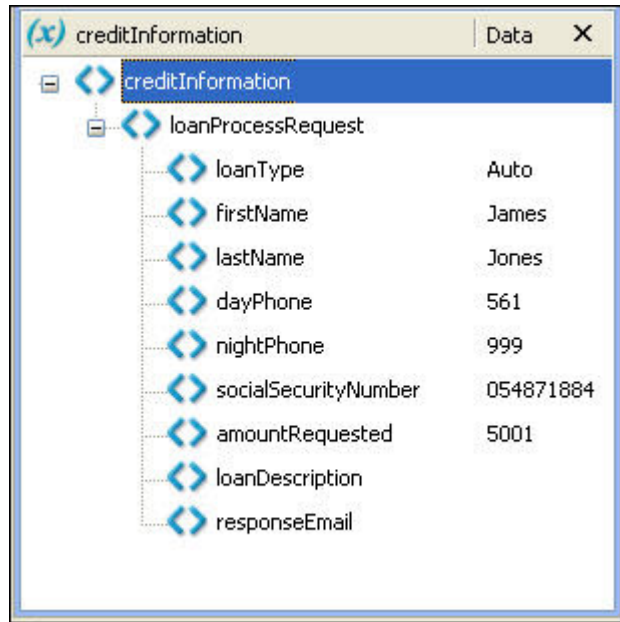


複合タイプ用の自動生成によるサンプルデータの便利なりポジトリの使用

[変数] ビューを表示してから、メッセージパートを右クリックして、サンプルデータ、複合型のインスタンスドキュメント、または単純型の値を生成するか、値を追加します。複合的なパートに対して、自動的にサンプルが生成されます。

プロセスの実行をシミュレートする際に、このデータはプロセス変数に自動的にロードされます。

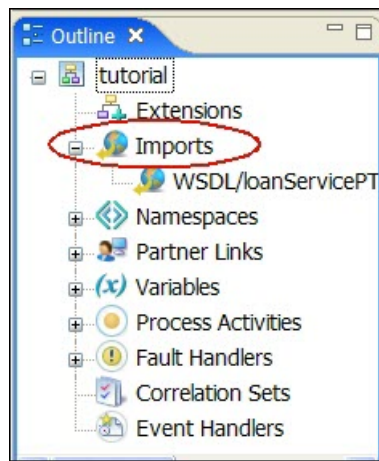
データは他のプロセスで再利用されます。



ターゲット名前空間またはその他のインポートの BPEL プロセスへの手動での追加

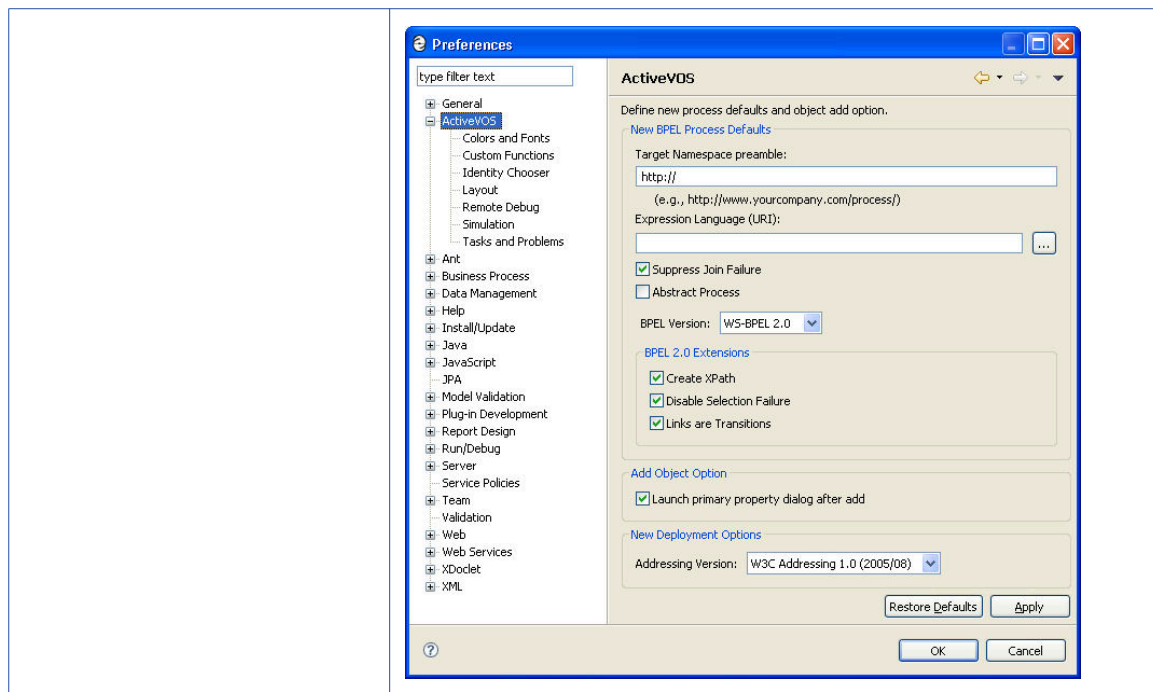
操作を使用してプロセスを開始すると、ターゲット名前空間が自動的に追加されます。

マウスを右クリックして【インポート】をクリックし、【インポートの追加】を選択して、[アウトライン]ビューにターゲット名前空間を手動で追加します。



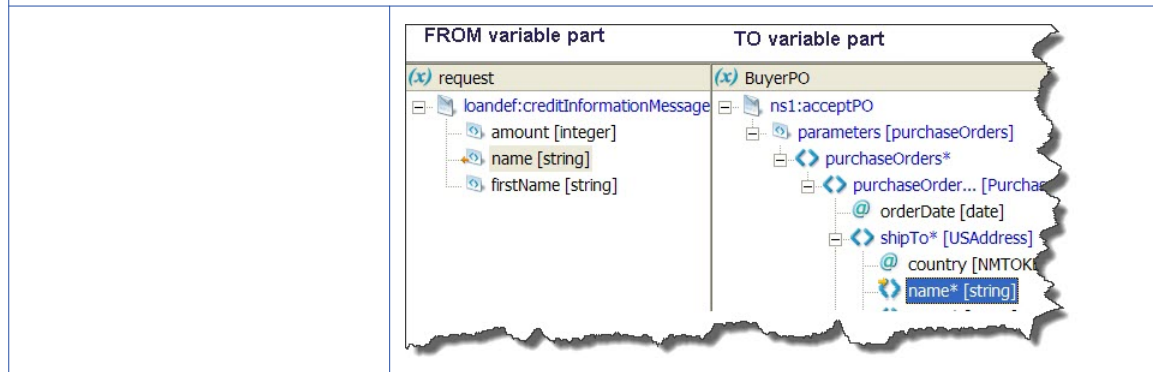
プロセス設定のカスタマイズ

Process Developer には、BPEL プロセスをグローバルにカスタマイズするためのさまざまな設定があります。結合の失敗の非表示やデフォルトのターゲット名前空間の指定など、すべてのプロセスのプリファレンスを設定します。必要に応じて、個々のプロセスのデフォルトを上書きします。



変数の割り当ての自動作成

同様のタイプのプロセス変数または変数部分を別のプロセス変数にドラッグして、コピー操作を自動的に作成します。青い色および From と To 矢印によって、可変部分が区別されます。

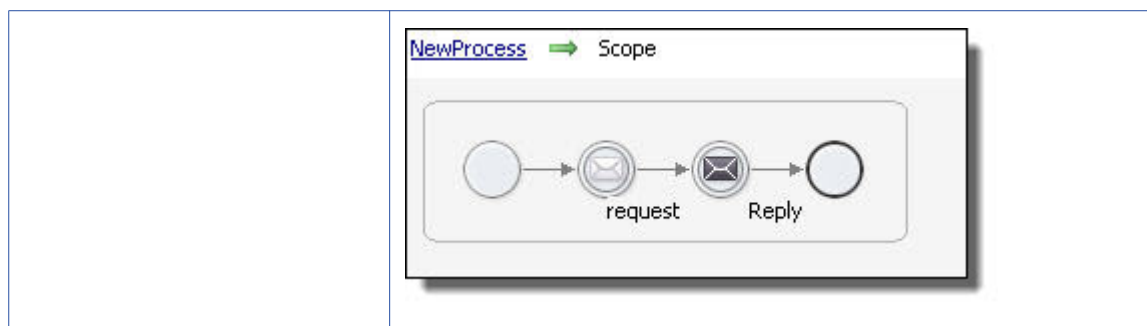


プロセス表示のヒント

コンテナの折りたたみ

Scope、While、Sequence などのコンテナを折りたたむことで、画面の領域を管理します。ダブルクリックして、展開したアクティビティをドリルダウンウィンドウに表示します。

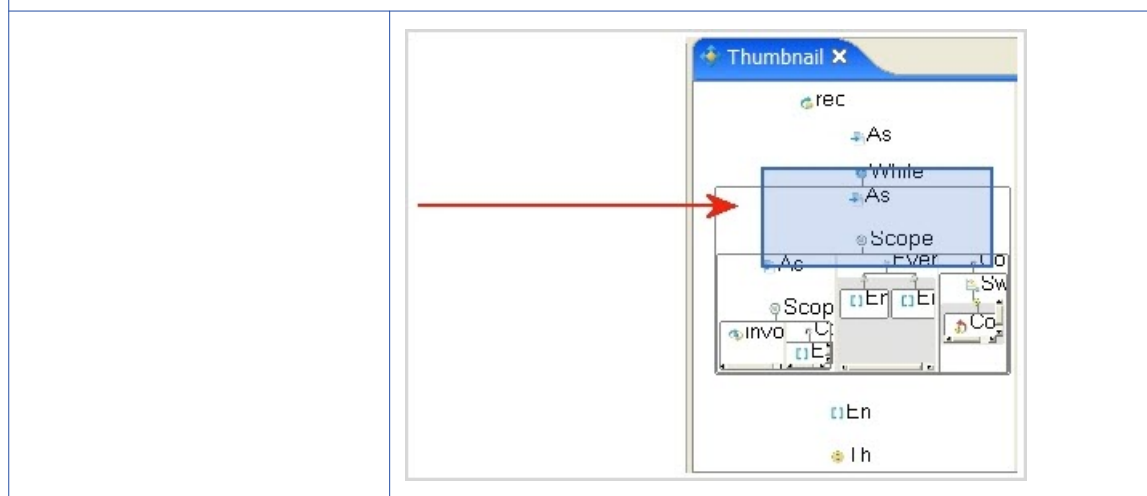
アクティビティを横方向または縦方向にします。



[サムネイル] ビュー

[サムネイル] ビューを使用して、プロセスのセクションを選択してパンニングします。

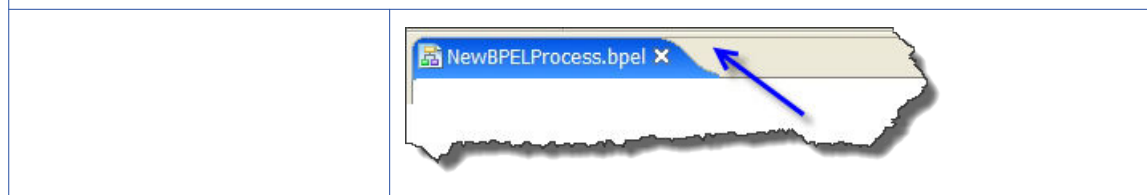
[サムネイル] ビューを表示するには、[ウィンドウ] > [ビューの表示] > [サムネイル] を選択します。



全画面の表示

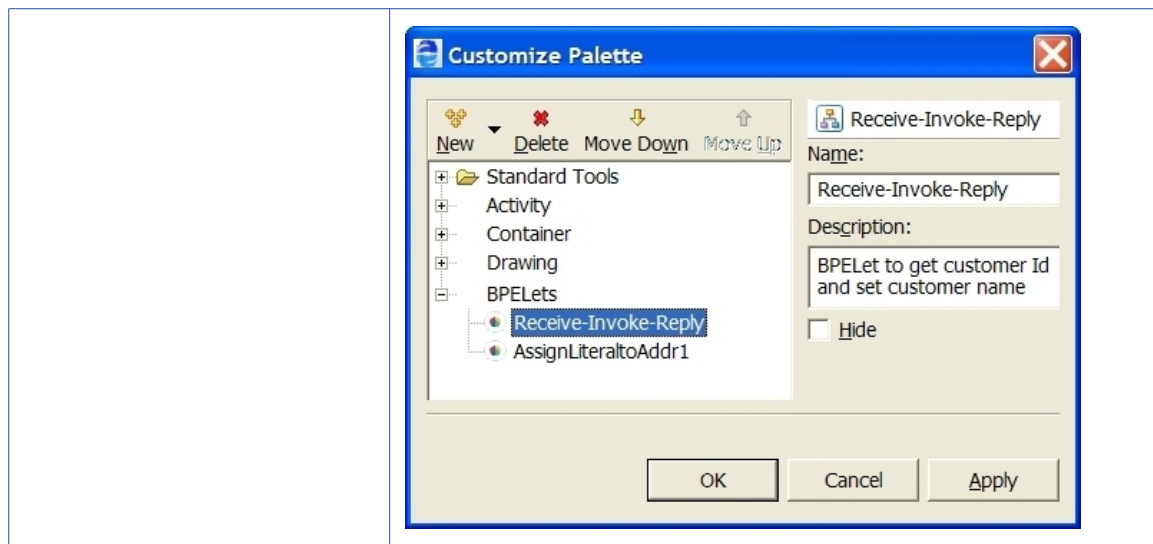
プロセスのタイトルバーをダブルクリックして、プロセスの全画面を表示します。

プロセスのプロパティを表示してメインのツールバーをアクティブにするには、Process Editor キャンバスの任意の場所をクリックしてフォーカスを合わせます。



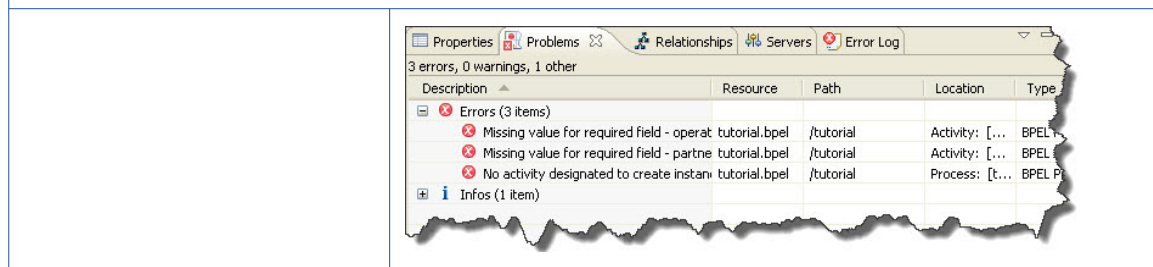
BPELet による設計時間の大幅な短縮

BPEL アクティビティまたはアクティビティのセットを BPELet (カスタムアクティビティ) として保存します。BPELet を [カスタム] パレットから任意のプロセスのキャンバスにドラッグします。



自動検証の問題の発見と修正

エラー、警告、および無効なアクティビティに関する情報のリストを表示します。問題を修正すると、この情報は自動的に表示されなくなります。

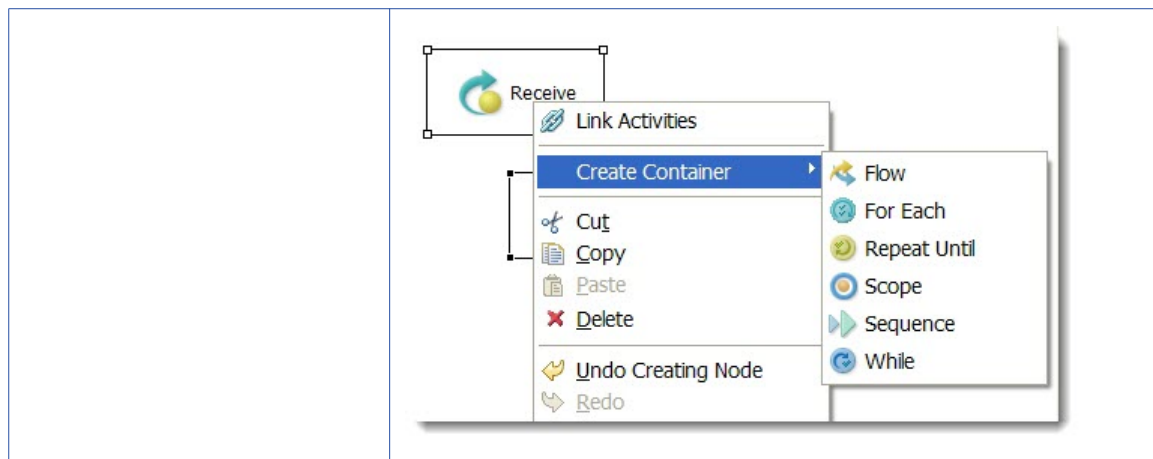


アクティビティ作成のヒント

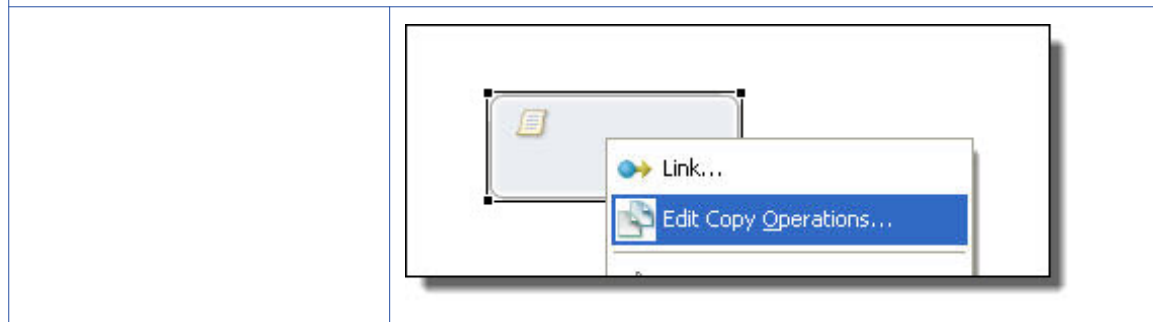
ソースアクティビティを選択してからターゲットを選択することで、アクティビティ間にリンクを追加します。マウスの右クリックメニューから「アクティビティのリンク」を選択するか、ツールバーの「エッジ」アイコンを使用します。



アクティビティのグループを選択し、マウスを右クリックして「コンテナの作成」をクリックすると、アクティビティのグループがコンテナに含まれます。

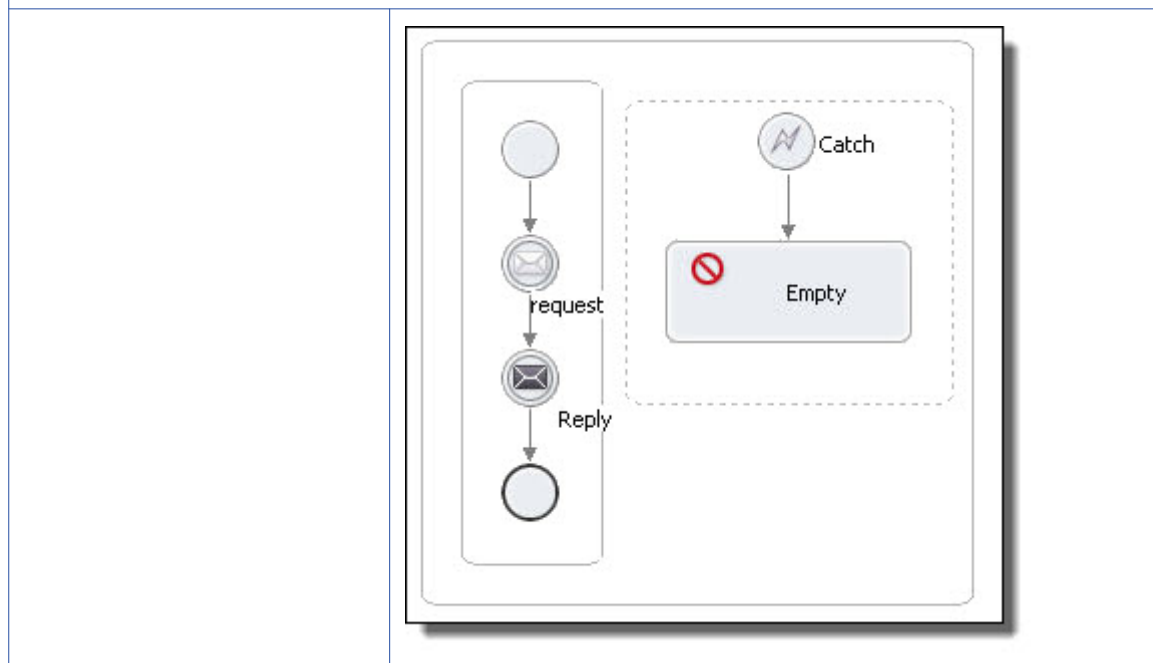


Receive や Reply などのアクティビティにデータマッピングを追加します。または、アクティビティをダブルクリックするか、アクティビティのマウスの右クリックメニューから選択して、[Assign] を追加し、[Assign] にコピー操作を追加/編集します。



パレットからスコープに Catch をドラッグして、フォールト処理（または他のハンドラ）をスコープに追加します。

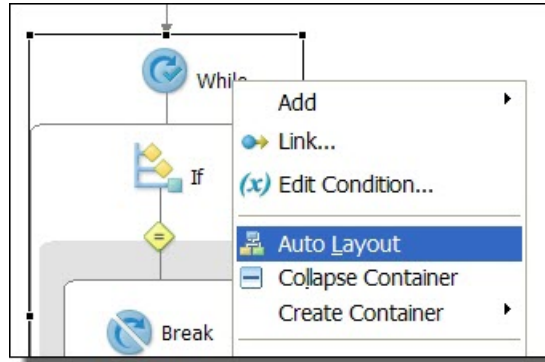
フォールトアクティビティが Catch コンテナに追加されることに注意してください。



プロセス設計のヒント

While アクティビティや If アクティビティなどの任意のコンテナで自動レイアウトを使用して、表示領域を最適化します。また、自動レイアウト用のオブジェクトのグループを選択することもできます。

オブジェクトはグリッドに自動整列されます。Alt キーとマウスの組み合わせを使用して、オブジェクトを一度に 1 ピクセルずつ移動します。



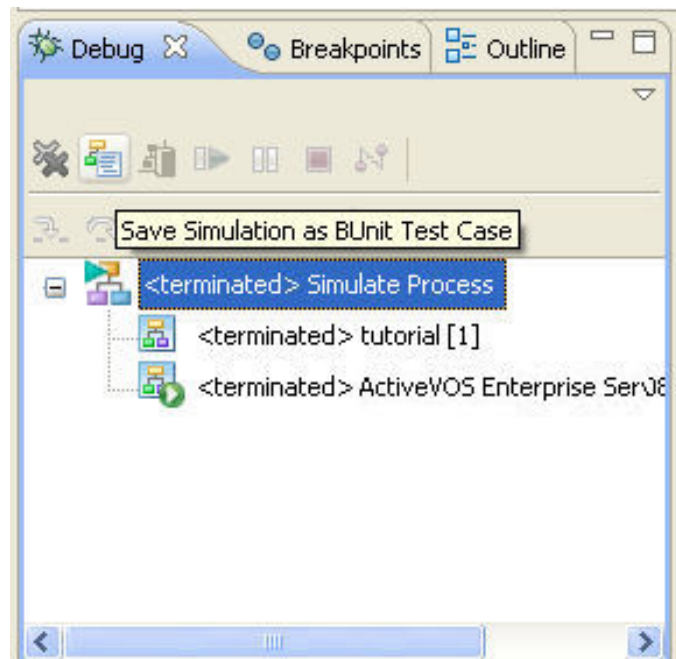
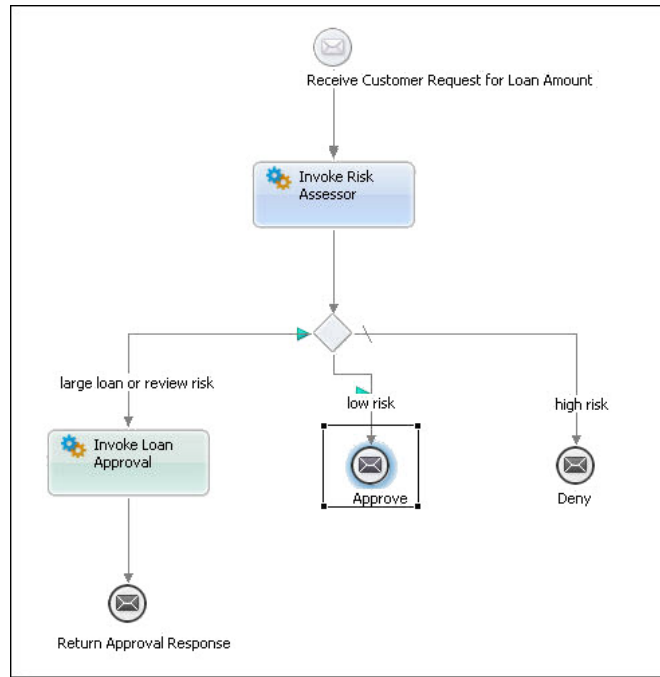
アクティビティ名を短いものではなく、わかりやすい名前にすることができます。



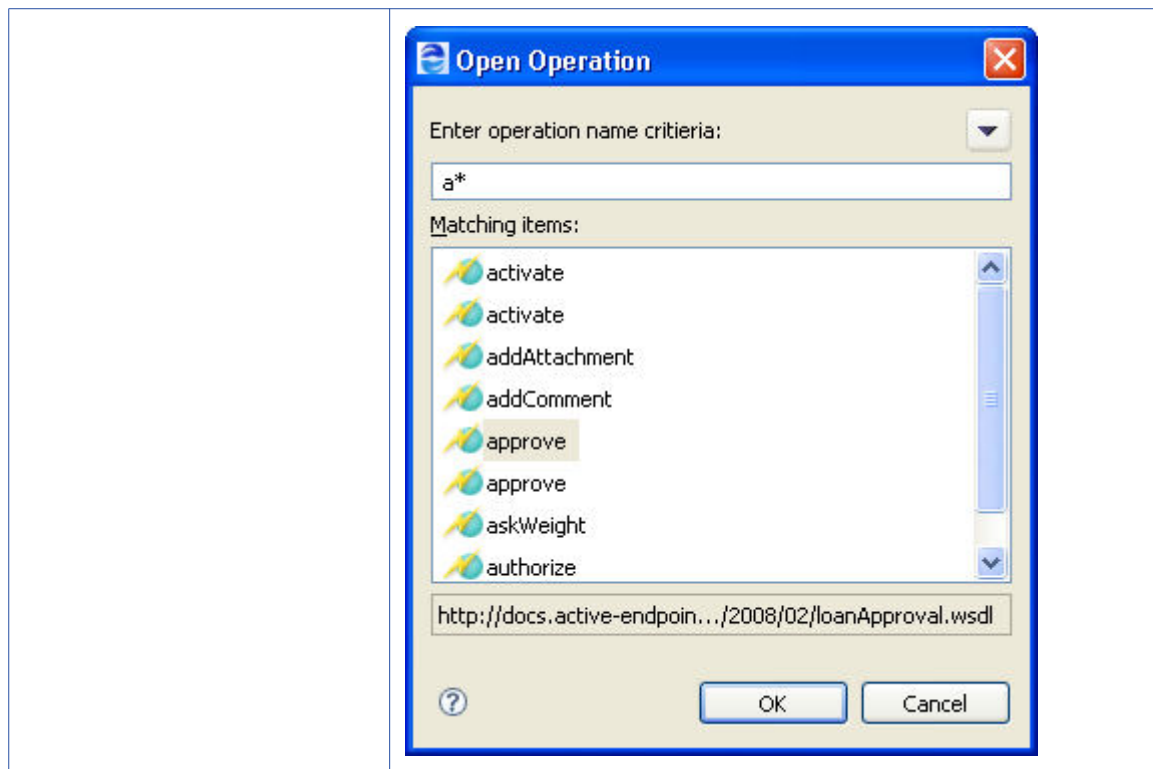
包括的なシミュレーション、ユニットテスト、およびリモートデバッグ機能の使用

メッセージのサンプルデータ値を生成し、さまざまな実行パスをステップスルーすることで、プロセスの実行をシミュレートします。Process Developer の内部実行エンジンは、テストのためのランタイム動作を提供します。

シミュレーションを B ユニットテストとして保存します。すべての B ユニットテストまたはスイート、あるいは失敗した B ユニットテストまたはスイートのみを再実行して、通常のプロセス実行を確認します。



「ナビゲート」メニューの各種の「開く」ダイアログを使用して、ワークスペース内の宣言、タイプ、およびデータファイルを簡単に見つけることができます。



Process Developer 内からのプロセスサーバーの起動（オンプレミスのみ）

プロセスをリモートでデバッグするには、サーバーエンジンを起動し、デプロイメント手順を完了します。ブラウザにコンソール URL を入力して、プロセスコンソールを表示します。

サーバーを開始する:

- ワークスペースの右下にある [サーバー] ビューを選択します。
- 右クリックして、[新規] > [サーバー] を選択します。[プロセスサーバー] を選択し、[完了] をクリックします。
- [サーバーの起動] ボタンを選択します。サーバーが起動すると、プロセスコンソールで起動タスクがスクロールします。サーバーを起動するたびに、ファイルが組み込みサーバーにデプロイされます。

[プロセスコンソール] ツールバーボタンを選択して、プロセスコンソールを表示します。次の URL でブラウザが開きます。

`http://localhost:8080/activevos`

このポートがすでにコンピュータで使用されている場合は、別のポートに変更します。

ソース管理システムの使用

Process Developer は、ソース管理システムで使用できます。Process Developer インストールパッケージには、CVS が組み込まれています。Subversion のソースコード管理を使用する場合は、次の方法でインストールすることができます。

1. [ヘルプ] > [新しいソフトウェアのインストール] をクリックします。
2. [追加] をクリックします。

Subclipse の更新サイトの名前を入力します。

場所 (http://subclipse.tigris.org/update_1.6.x) を入力します

3. **【OK】** をクリックします。
4. リスト内のすべての項目を選択します。
5. **【次へ】** をクリックします。
6. **【次へ】** を再度クリックします。
7. **【同意する】** ラジオボタンを選択します。
8. **【完了】** をクリックします。

Eclipse で SVN がインストールされます。

警告が表示された場合は、それらを承認してください。また、プロンプトが表示された場合は、Process Developer を再起動する必要があります。

Process Developer のヘルプへのアクセス

Process Developer には、次のようなさまざまなオンラインヘルプ機能があります。

ヘルプ機能	説明
ヘルプ メニュー	[ヘルプ] メニューを選択して、ようこそページ、ヒント、およびヘルプのコンテンツを表示します
ようこそページ	新規ユーザー向けの基本的なヘルプが記載されています
ヘルプのコンテンツ	[ヘルプ] メニューから、展開可能/折りたたみ可能なリストにあるすべてのヘルプトピックを表示します
チートシート	[ヘルプ] メニューから、ステップバイステップ形式の一般的なセルフガイド基本タスクを含むビューを開きます
F1 ヘルプ	状況依存ヘルプトピックを表示するには、任意のビューまたはダイアログボックスで F1 キーを押します
ホバーヘルプ	リスト、ツールバーボタン、またはグラフィックディスプレイの項目にマウスを置くと、プロパティの詳細が表示されます
ヘルプ検索	[検索] メニューまたはツールバーアイコンを選択して、ヘルプトピックを検索します

第 3 章

ビジネスプロセス実行言語の概要

Web サービス用のビジネスプロセス実行言語（BPEL）は、Web サービスプロトコルに基づいてプロセスのオーケストレーションを定義するための XML 表記です。BPEL 仕様は、Web サービスアーキテクチャの重要な標準です。BPEL により、プロセス固有の言語構造について説明し、複数の Web サービスを一貫した情報システムとして構成する方法を定義します。BPEL は、Web サービスアーキテクチャの他の標準、特に Web サービス記述言語（WSDL）に基づいて構築されています。

BPEL のリソース

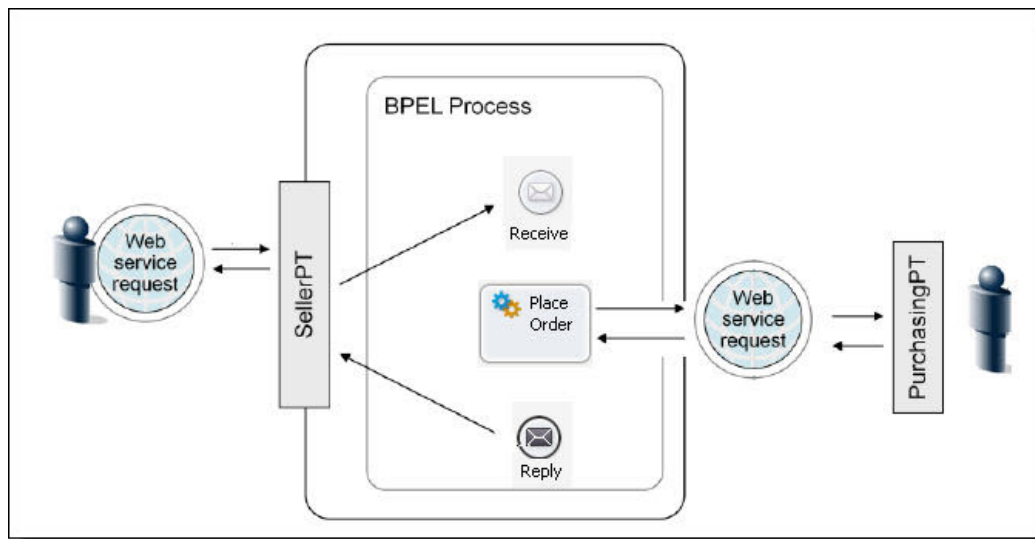
WS-BPEL 2.0 仕様は、Informatica の Web サイトから入手できます。WS-BPEL は、次の XML ベースの仕様に依存しています。

- Web サービス記述言語バージョン 1.1。
- XPath 式言語
- XML スキーマ WS アドレス指定。
- IBM、BEA システムズ、Microsoft など、仕様のコントリビュータの Web サイトにある WS-Addressing の仕様を参照してください。

BPEL プロセスについて

BPEL プロセスは、定義した方法でインタラク션을振り付けるための Web サービスのコレクションです。それぞれのサービスは、特定のタイプの処理を実行するパーティシパントです。サービスは、非常に細かく（例えば、レートを計算するようになど）設定されていたり、または非常に広い範囲（例えば、注文を処理するようになど）で設定されていたりすることがあります。

次の図では、プロセスは売り手のパーティシパントであり、Web サービスの入力メッセージを介して顧客からの注文書を受け入れています。次に、売り手は購入パーティシパントに注文を行います。注文が履行された場合、プロセスを終了するために、プロセスは顧客に対して確認応答を返します。



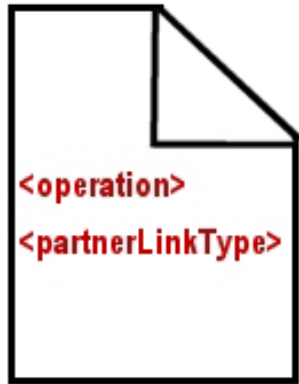
パーティシパントが確認応答を送信した後、出荷通知や請求書などの他のメッセージが顧客に送信されます。

発注を行った多くの顧客や他の顧客とこのようなやり取りをする場合は、1つのビジネスプロセスを構築し、それがビジネスプロセスインスタンスのテンプレートとして機能するようにします。

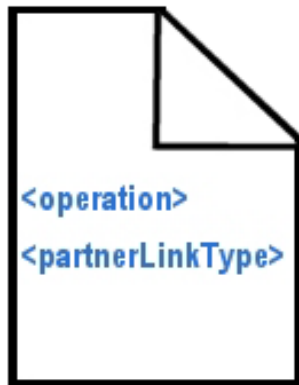
構築するビジネスプロセスに「インスタンスを作成する」という設定を行うと、新しい発注書が到着するたびに新しいプロセスが作成されます。新たなプロセスによって、関連するすべてのインタラクションが処理され、それぞれの買い手の注文書ごとにインタラクションが分離されます。

BPEL プロセス定義は、Web サービス記述言語（WSDL）ファイルにある定義を入力として使用します。これらのファイルには、外部と共有可能なインタフェース情報が含まれています。次の図に示すように、Process Developer は、パートナーリンクの種類や操作などの情報を選択して、プロセスステップを定義します。

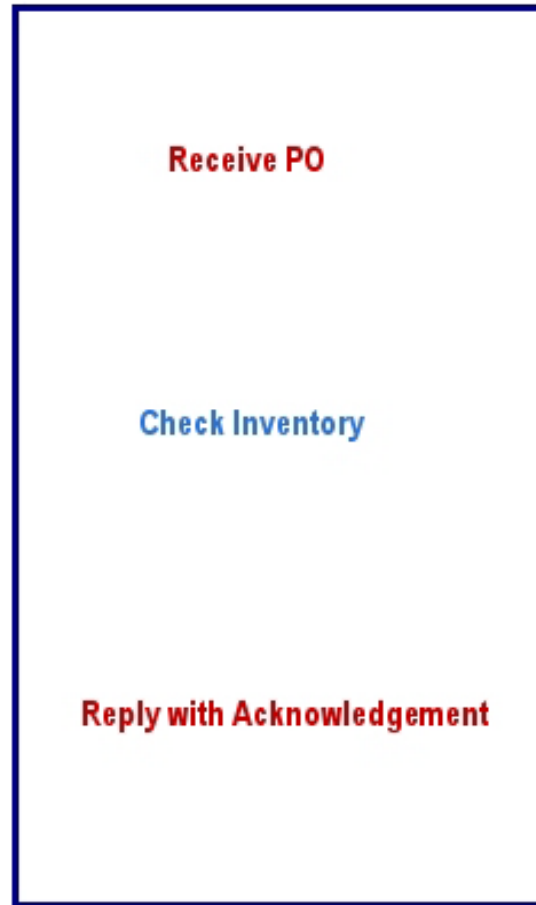
Seller's WSDL File



Supplier's WSDL File



Receive PO_CheckInventory_Reply BPEL Process



BPEL プロセスは、これらのインタラクションを調整し、ストレートスルーまたは長期実行フローとして構成します。例えば、プロセスの実行中に例外が発生した場合、アクティビティが取り消されたり元に戻されたりする可能性があります。BPEL プロセスは、相関、補償、フォールト、およびイベントを処理するための手法を提供します。

BPEL プロセス定義要素

次のコードサンプルに、ビジネスプロセス実行言語（BPEL）定義の主なセクションを示します。

```
<process>
  <!-- Definition of roles of process participants -->
  <partnerLinks> ... </partnerLinks>
  <!-- Data and state variables used within the process -->
  <variables> ... </variables>
  <!-- Correlation comment -->
  <correlationSets> ... </correlationSets>
  <!-- Exception management -->
  <faultHandlers> ... </faultHandlers>
  <!-- Message and timeout event handler -->
  <eventHandlers> ... </eventHandlers>
  <!-- Processing steps -->
```

```
activities*
</process>
```

WS-BPEL 2.0 に対する Informatica 拡張機能

Informatica Business Process Manager は、WS-BPEL 2.0 仕様を完全にサポートするだけでなく、このトピックで説明する BPEL 拡張機能を備えています。

プロセスレベルの拡張機能

- 補償ハンドラと終了ハンドラ
詳細については、このヘルプの「プロセス要素とプロセスのプロパティ」を参照してください。
この拡張機能の名前空間は、http://www.activebpel.org/2006/09/bpel/extension/process_coordination です。
- [XPath の作成] や [選択エラーの無効化] などのクエリ処理
詳細については、このヘルプの「プロセス要素とプロセスのプロパティ」を参照してください。
この拡張機能の名前空間は、http://www.activebpel.org/2006/09/bpel/extension/query_handling です。

アクティビティ

拡張アクティビティには、このヘルプの他の場所に記載されている「Suspend」、「Break」、および「Continue」があります。

この拡張機能の名前空間は、<http://www.activebpel.org/2006/09/bpel/extension/activity> です。

リンク

リンクは WS-BPEL で同期構造として使用され、1 つのアクティビティを別のアクティビティのターゲットまたはソースにすることができます。しかし、両方のアクティビティを同時に使用することはできません。リンク拡張機能を使用することで、ループバック機能が可能になります。このヘルプの他の場所にある「リンクの拡張機能」を参照してください。

この拡張機能の名前空間は、<http://www.activebpel.org/2009/06/bpel/extension/links> です。

カスタム XPath 関数

このヘルプの「式ビルダの使用」に記載されているように、複数のカスタム関数があります。

拡張プレフィックス abx は、名前空間 <http://www.activebpel.org/2006/09/bpel/extension> 用です。

暗黙的なスコープの変数

プロセスの視覚的な表示を合理化するために、プロセスサーバーでは、データのマッピングに使用される Assign アクティビティとプロセス変数は表示されません。代わりに、Receive、Reply、Invoke、およびユーザーアクティビティ内にデータマッピングを追加することができます。これらのアクティビティ内でデータをマッピングを実行すると、マッピングデータ用のスコープされた変数が内部的に追加されます。このヘルプの他の場所にある「入力変数」と「出力変数」を参照してください。

この拡張機能の名前空間は、<http://www.activebpel.org/2009/02/bpel/extension/ignorable> です。

BPMN での中断境界イベント

イベントハンドラを BPMN 境界イベントとして扱い、メインアクティビティを終了するように設定できます。このヘルプの他の場所にある「境界イベントの追加」を参照してください。

この拡張機能の名前空間は、<http://www.activebpel.org/2009/02/bpel/extension/ignorable> です。

宣言されていない SOAP フォールト (Java 名)

Informatica Business Process Manager には、宣言されていないフォールトをキャッチするカスタム関数が含まれており、Java 名でフォールトをキャッチできます。詳細については、このヘルプの他の場所にある「宣言されていないフォールトと SOAP フォールトのキャッチ」を参照してください。

必要な拡張名前空間は、<http://www.active-endpoints.com/2004/06/bpel/extensions/>です。プロセスサーバーもまた、この名前空間を使用してシステムエラーをキャッチします。

第 4 章

Process Developer の設定

すべてのプロセスのデフォルト設定を構成した後に、個々のプロセスの設定を上書きできます。

Process Developer の設定を行うには、**[Window] > [設定] > [Process Developer]** を選択します。

メインページの設定は次のとおりです。

新しいプロセスファイルのデフォルト設定:	
ターゲット名前空間のプリアンブル	すべての BPEL プロセスにはターゲット名前空間が必要です。必要に応じて、デフォルトの名前空間プリアンブルを入力します。名前空間を変更して、作成する新しいプロセスごとに名前空間を追加します。
式言語 (URI)	Process Developer は、さまざまな表現言語をサポートしています。デフォルトの式言語は XPath 1.0 ですが、他にも 2 つの表現言語が組み込まれています。他の言語は XQuery 1.0 と JavaScript 1.5 で、いずれかの言語をデフォルトとして選択できます。デフォルト言語とは、特定のプロセスで式言語が明示的に宣言されていない場合にそのプロセスで (検証とシミュレーションに) 使用される式言語を指します。設定した式言語を表示するには、 [ダイアログ (...)] を選択します。
結合の失敗の非表示	デフォルトで有効になっています。この設定の説明については、「プロセス要素とプロパティ」を参照してください。
抽象プロセス	すべての新規プロセスを、実行可能なプロセスではなく、抽象的なプロセスとして作成します。デフォルトでは無効になっています。この設定に関する説明については、「抽象プロセスの作成」を参照してください。
BPEL バージョン	Process Developer の起動時および新しい BPEL プロセスの作成時に、使用する BPEL 仕様バージョンを選択します。サポートされているバージョンは、WS-BPEL 2.0 (現在のバージョン) または BPEL4WS 1.1 です。選択したバージョンによって、新しいプロセスの正しいメニュー項目、パレット、およびその他の詳細が決定されます。作成する新しいプロセスは、選択したバージョンに準拠します。開いたプロセスはすべて、正しい BPEL バージョンに準拠します。
BPEL 拡張	
XPath の作成	WS-BPEL 2.0 プロセス用の Process Developer 拡張機能。この設定の説明については、「Process Developer の XPath 作成拡張機能の使用」を参照してください。BPEL4WS 1.1 プロセスのシミュレーション中にこの設定を使用する方法については、「シミュレーションの設定」を参照してください。
選択の無効化エラー	WS-BPEL 2.0 プロセス用の Process Developer 拡張機能。この設定の説明については、「Process Developer の無効な選択エラーのフォールト拡張機能の使用」を参照してください。BPEL4WS 1.1 プロセスのシミュレーション中にこの設定を使用する方法については、「シミュレーションの設定」を参照してください。

リンクは遷移です	WS-BPEL 2.0 プロセス用の Process Developer 拡張機能。完了したアクティビティを対象とするリンクを許可します。「リンクに対する <i>Process Developer</i> 拡張機能」を参照してください。
オブジェクト追加オプション	
追加後にプライマリプロパティダイアログを起動	<p>［アウトライン］ ビューで、インポート、パートナーリンク、変数などのノードを右クリックして、新しい項目を追加できます。このオプションを有効にすると、項目を右クリックしてプライマリプロパティを編集できます。</p> <p>［プロパティ］ ビューで（プライマリプロパティだけでなく）すべてのプロパティを編集する場合は、このオプションを無効にします。</p>
新規デプロイメントオプション	
Addressing バージョン	<p>デフォルトの WS Addressing 仕様は W3C 1.0 です。この仕様は、新しいプロセスデプロイメント記述子ファイルのプレアンプルで参照されます。</p> <p>新しいデプロイメント記述子に適用可能な場合は、以前のバージョンのいずれかの仕様を選択できます。</p>

タスクタグの追加

Process Editor を使用すると、プロセス要素のテキストタグおよびタスクタグの追加やタスクの優先度の設定ができます。

名前を入力し、タグの優先度を設定します。タグは次のように使用します。

1. Process Editor キャンバスで、*Receive* などのアクティビティを選択する。
2. ［プロパティ］ ビューで、**コメント** を選択し、行の最後で **ダイアログ** を選択する。
3. **コメント** ダイアログで、タグとテキストを入力する。例:
入力メッセージ用の@todo サンプルデータファイルの作成
4. **OK** をクリックすると、新しいタスクがタスクビューに追加されることに注意してください。また、タスクは、アクティビティのホバーヘルプの一部として、およびソースビューのコメントタグ内に表示されます。

ヒント: アクティビティの右クリックメニューから **タスクの追加** を選択して、タスクを追加することもできます。タスクはタスクビューに追加されますが、ホバーヘルプまたはプロセスコメントタグには表示されません。

B ユニットの設定

BPEL ユニット（B ユニット）テストは、Process Developer エンジンで BPEL プロセスをテストするフレームワークです。B ユニットテストを作成する場合、プロセスをシミュレートし、シミュレーションを B ユニットテストとして保存する方法が最も簡単です。

プロセスをシミュレートするには、Receive と Invoke のサンプルデータを追加します。

B ユニットテストのデフォルトのサンプルデータの作成を、インラインデータまたはインポートされたデータファイルのどちらから行うかを設定します。デフォルトでは、データはサンプルデータファイルとしてインポ

ートされます。この設定により、B ユニットテストの管理が容易になります。インラインを選択すると、シミュレーションのデータが XML ドキュメントとして表示されます。

B ユニットエディタで、[Invoke] > [シミュレート済み応答] > [メッセージ] > [パート] を選択し、データをインラインまたはワークスペースからインポートするように設定します。

詳細については、[第 29 章, 「BPEL ユニットテスト」 \(ページ 433\)](#)を参照してください。

キャッシュとタイムアウトの設定

ワークスペースの構築完了の待機時に問題が発生する場合は、プロセス検証のキャッシュ値とタイムアウト値を設定できます。

WSDL やスキーマを使用する BPEL プロセスを開く際または検証の実行中など、WSDL やスキーマファイルがロードされる状況にはさまざまなパターンがあります。プロセスに含まれる WSDL とスキーマの数が多い場合、サイズが大きい場合、またはリモートである場合、ロード時間と構築時間が遅くなる場合があります。これは特に、存在しない参照がある場合やロードに時間がかかる場合に当てはまります。

これらの設定を表示するには、[ウィンドウ] > [設定] > [Process Developer] > [キャッシュとタイムアウト] を選択します。

- **不正な WSDL キャッシュタイムアウト。**この値は 120 秒に設定されています。プロセスの検証中に WSDL の URL に接続できない場合は、WSDL の場所がキャッシュされ、後で接続が試行されます。
- **リソースロードタイムアウト。**このタイムアウトによって、外部 WSDL ファイルのロードを待機する時間を制御します。また、このタイムアウトは、WSDL（およびそのすべてのインポート）のロードにかかる時間も制御します。デフォルトでは、5 秒に設定されています。マシンまたはネットワークが遅い場合は、この値を増やします。

検証ビルダーの詳細については、[「プロジェクトのオーケストレーションおよび検証ビルダについて」 \(ページ 88\)](#)を参照してください。

色とフォントの設定

色とフォントの設定を表示するには、[ウィンドウ] > [設定] > [Process Developer] > [色とフォント] を選択します。

一部のアクティビティや遷移条件、クエリでは、データ操作式を作成するための式ビルダを使用できます。式ビルダ内では、Process Developer の式テキストのフォントプロパティを設定できます。

[BPEL ユニットテスト] (B ユニット) ビューには、B ユニット XML 比較ウィンドウを開くツールバーボタンが含まれています。このウィンドウに表示されるテキストの色とフォントの設定を行うことができます。

コントリビューションの設定

コントリビューションには、サーバー上のファイルの単位として管理されるすべてのプロセスリソースを含めます。

コントリビューションの設定を表示するには、**[ウィンドウ] > [設定] > [Process Developer] > [コントリビューション]** を選択します。

プロセスをプロセスサーバーにデプロイする場合は、提供されたデプロイメント内にすべてのプロセスリソースを含めることをお勧めします。デフォルトでは、エクスポートウィザードによって、サンプルデータとテストを除くすべてのプロジェクトフォルダ内のリソースがすべて選択されます。ただし、**[除外されていないすべてのプロジェクトアーティファクトを自動選択]** を無効にすると、リソースを除外できます。

BPR コントリビューションデプロイメントウィザードのエクスポート内で、選択または除外が必要なプロジェクトサブフォルダを一覧表示できます。エクスポートウィザードを開始するための関連フォルダのリストを作成します。

また、エクスポートウィザードで除外するフォルダのリストに、独自のフォルダ名を追加することもできます。例えば、バックアップフォルダを作成した場合は、そのフォルダをリストに追加できます。

詳細については、「**デプロイメントコントリビューションの管理**」を参照してください。

ID 選択の設定

勤めている企業のディレクトリからグループを選択します。組織のグループ名を識別するサービスの詳細を入力します。Process Developer はそのサービスと通信して、特定のアクティビティ内のグループ名を提示します。

[ウィンドウ] > [設定] > [Process Developer] > [ID の選択] を選択して、設定ページを表示します。

ID サービスの場所の指定が可能なこの設定は、現在、プロセス内にユーザーによるインタラクションを組み込んだアクティビティであるユーザーアクティビティでのみ使用されます。詳細については、「*Process Developer* のヘルプ」の「**ヒューマンタスク**」を参照してください。

グループメンバーの選択に使用する **[ID 選択]** と、Process Deployment 記述子エディタ内でのユーザークエリの作成に使用する **[管理サービス]** という 2 つの ID サービスにエンドポイントと資格情報を設定できます。

ID 選択

プロセスサーバーの実行中に Process Deployment 記述子エディタを開くことができ、**[ユーザー]** タブに **[ID 選択]** ダイアログの選択肢があります。**[更新]** を選択すると、このアクションによってプロセスサーバーで有効な ID サービスに要求が送信されます。組織のディレクトリまたはファイルベースの ID サービスから、グループ名のリストが取得されます。このグループ名は、デプロイ中に論理ユーザーグループ用に選択できます。

Process Developer の設定の [ID 選択] ページで、ID 選択に必要な ID サービスにアクセスするための通信の詳細を次のように定義します。

エンドポイント	プロセスサーバーのエンドポイント URL のデフォルトの host:port セクションを変更します。デフォルトの URL は、localhost:8080 で実行されているサーバーを指しています。使用するサーバーを指すようにホストとポートの値のみを変更します。
ユーザー名	必要に応じて、ユーザー名の資格情報を入力して、Process Developer の aeldentityService にアクセスします。プロセスサーバーの管理者は、Process Developer からの ID サービスのセキュリティを有効にする abldentityListConsumer ロールに関連付けられたユーザー名とパスワードを指定できます。このロールに関する情報は、このヘルプの「サーバー、インストール、構成、およびデプロイ」セクションに記載されています。 「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 (ページ 483)に記載されているように、ユーザー名とパスワードは、BPR ファイルを Web サービスとしてデプロイする際に入力したものと同一である可能性があることに注意してください。
パスワード	上記のユーザー名のパスワードを入力します

ID 選択の使用

ID 選択を使用して、ユーザーアクティビティで論理ユーザーグループを選択するための組織内のグループのリストが返されるようにします。

[ID 選択] を選択する前に、次のことを行う必要があります。

- [「組み込みプロセスサーバーのセットアップ」](#) (ページ 96)に記載されているように、プロセスサーバーが実行されていることを確認します。
- サーバーで Informatica プロセス ID サービスの設定ページが正しく入力されていることを確認します

ID 選択の詳細については、「*ヒューマンタスク (Process Developer) ガイド*」を参照してください。

管理サービス

管理サービスはサーバーと通信して、論理ユーザーグループにユーザークエリを作成するための構成設定を取得します。通信の詳細を追加するための ID 選択については、上記の説明を参照してください。

プロセスサーバーの管理者は、ActiveBpelAdmin サービスに必要な abAdmin ロールに関連付けられたユーザー名とパスワードを指定できます。

レイアウトの設定

Process Editor キャンバスにグラフィック要素をレイアウトするためのデフォルト値を設定します。

これらの設定を表示するには、[ウィンドウ] > [設定] > [Process Developer] > [レイアウト] を選択します。

[レイアウトの設定] ページでは、次のことができます。

- デフォルトのビジュアルプロセスプロパティを定義する編集スタイルを選択する。
- [自動レイアウト] モードのデフォルトを定義する。
- [コンテナの展開/折りたたみ] レイアウトのデフォルトを定義する。

編集スタイルとスタイルシート

このパネルでは、新しい BPEL プロセスのデフォルトの視覚的なプロパティを指定します。デフォルトの編集スタイルは、ビジネスプロセスモデルおよび表記法言語で BPEL プロセスをモデル化する BPMN です。BPEL プロセスは、視覚的に BPMN 図に似ています。

レガシーの Process Developer のアクティビティの視覚的表現には、[Classic] を選択します。[「Process Developer Classic スタイルについて」 \(ページ 54\)](#)を参照してください。

編集スタイルに BPMN を選択した場合は、[BPMN] または [BPEL] を設定します。このパネルでは、標準の BPMN 表記を使用するか、BPEL 中心のバージョンの BPMN を使用するかを設定できます。つまり、アイコンはほとんどが標準の BPMN であり、レガシーの Process Developer バージョンのパレットと一致します。

詳細については、[「BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較」 \(ページ 50\)](#)および [「どちらの編集スタイル \(BPMN 中心または BPEL 中心\) を選択するかについて」 \(ページ 52\)](#)を参照してください。

[スタイルシート] で、スタイルシートに [その他] を選択し、テキストエディタでスタイルシートファイルを編集してデフォルト値を変更します。変更はすべてのプロセスに適用されます (例えば、アクティビティの受信アイコンに境界線を設定する、など)。任意の BPEL プロセスで、必要に応じて、個々の Receive アクティビティの境界を削除できます。クラシックスタイルシートには、WS-BPEL 2.0 または BPEL4WS 1.1 のいずれかの BPEL の 1 つのバージョンにのみ適用される複数のプロパティが含まれていることに注意してください。BPMN スタイルシートは、WS-BPEL 2.0 にのみ適用されます。

視覚的なプロパティの説明については、[「視覚的なプロパティの設定と独自のイメージライブラリの使用」 \(ページ 76\)](#)を参照してください。

自動レイアウト

メインの Process Developer ツールバーには、プロセスキャンバス上のグラフィック要素を表示領域に合わせてグリッド形式で編成する **【自動レイアウト】** ボタンがあります。

このパネルで、グラフィック要素 (アクティビティやコンテナなど) をレイアウトするためのデフォルト値を設定します。

自動レイアウトは、**【自動レイアウト】** ツールバーボタンから利用できます。詳細については、[「Process Developer のメニューとツールバー」 \(ページ 59\)](#)を参照してください。

次のように、追加の自動レイアウト設定を行います。

方向: 横方向または縦方向	プロセス図の主軸を選択します。
コンテナのマージンのサイズ	スコープ、シーケンス、およびその他のコンテナの内部マージンのサイズをピクセル単位で設定します。
レベルとノード間の間隔	アクティビティ間の縦方向 (レベル) および横方向 (ノード) 間隔のサイズをピクセル単位で設定します。

コンテナの展開/折りたたみ

このパネルで、コンテナアクティビティ (Sequence、If、While、Pick、Flow、Scope、For Each、および Repeat Until) のオブジェクト移動とリンクスタイルのデフォルト値を設定します。詳細については、[「アクティビティの表示と非表示」 \(ページ 81\)](#)を参照してください。

自動移動	コンテナを折りたたむと、オブジェクトが空のスペースに表示されます。コンテナを展開すると、オブジェクトは空いている他のスペースに移動されます。
自動移動の展開	コンテナを展開すると、オブジェクトは空いている他のスペースに移動されます

自動レイアウト	コンテナを折りたたんだり展開したりすると、オブジェクトは非表示のグリッドに沿って配置されます
移動なし	コンテナを折りたたんだり展開したりしても、オブジェクトは移動されません
折りたたまれたリンクスタイル	折りたたまれたコンテナ内のアクティビティとの間のリンクに、実線、破線、または点線のスタイルを設定します。他のリンクスタイルとは異なるスタイルを設定することで、非表示のアクティビティまたはコンテナを視覚的に区別できるようにします。

アクティビティコンテキストメニュー

デフォルトでは、関連するすべての Eclipse コンテキスト項目は、Process Editor キャンバス上のアクティビティの右クリックメニューに表示されます。

これらの項目を非表示にするには、**[Process Developer 以外のコンテキストメニュー項目を表示する]**のチェックをオフにします。

次の項目を非表示にすることができます。

- 比較
- デバッグ
- プロファイル
- 置換後の文字列
- 実行
- チーム
- 検証
- Wiki テキスト

Process Editor のタブ

Process Editor の [フォールトハンドラ]、[イベントハンドラ]、および [ソース] タブを非表示にできます。**[プロセスアクティビティ以外のタブを表示する]**のチェックマークをオフにして、これらの項目を非表示にします。

[リレーション] ビューの設定

[リレーション] ビューのデフォルトのレイアウトを設定します。これにより、エディタで開いているファイルに関連する BPEL、WSDL、XSD、およびその他のアーティファクトの依存関係と参照がグラフィカルに表示されます。

[リレーション] ビューには、エディタで開いているファイルに関連する BPEL、WSDL、XSD、およびその他のアーティファクトの依存関係と参照がグラフィカルに表示されます。

次のような設定を行います。

- **深度制限。**ファイルにインポートがあり、インポートにインポートがある場合、これらをさまざまなレベルで表示できます。デフォルトの深度制限は 2 で、別のファイルをインポートする 1 つのファイルが表示されます。
- **レイアウトオプション。**複数のグラフィカルなレイアウトスタイルが利用できます。エディタで開いているファイルに最適なものを選択してください。デフォルトは有向グラフです。これはツリーに似ていますが、横方向の制御に優れています。

タスクと問題の設定

タスクに対する独自のタグの有効化や作成ができます。

これらの設定を表示するには、[ウィンドウ] > [設定] > [Process Developer] > [タスクと問題] を選択します。

ユーザー定義のタスクタグの有効化	タスクの作成時に独自のタグを使用するには、このチェックボックスを選択します。
タスクタグの追加	タグを追加してから、アクティビティの [コメント] ダイアログでタグを使用します。テキストは、[タスク] ビューにタスクとして追加されます。また、これはアクティビティのホバーヘルプの一部として表示されます。詳細については、「 タスクタグの追加 」(ページ 40)を参照してください。Process Developer には、通常の優先度を持つ @todo と高い優先度を持つ TODO という 2 つのデフォルトタグが用意されています。
エラーの問題の有効化	ファイルの保存時にすべての検証エラーを表示するには、このチェックボックスを選択します。検証エラーは、無効な BPEL 要素定義を参照します。例えば、パートナーリンク、操作、および変数を指定せずに Receive アクティビティをキャンバスにドラッグすると、欠落した属性ごとにエラーの問題が生成されます。問題を通知しない場合は、このチェックボックスをオフにします。実行をシミュレートする前には、必ずエラーの問題を有効化し、修正してください。
警告の問題を有効化	ファイルの保存時にすべての検証の警告を表示するには、このチェックボックスを選択します。検証の警告は、変数やパートナーリンク、サポートされていない式言語、その他の潜在的なエラーなど、未使用のリソースを参照している場合があります。警告を通知しない場合は、このチェックボックスをオフにします。 プロセスで相関セットを使用する場合は、警告の問題を有効にすることを強くお勧めします。相関メッセージを処理するアクティビティに相関がない場合、警告の問題が表示されます。
情報の問題の有効化	ファイルの保存時にすべての情報メッセージを表示するには、このチェックボックスを選択します。情報メッセージを通知しない場合は、このチェックボックスをオフにします。

追加の設定

Process Developer の設定に記載されている一般的なワークスペースの設定に加えて、他のセクションに記載されている次の設定ページを参照してください

- [リモートデバッグの設定](#)
- [シミュレーションの設定](#)

第 5 章

BPMN 中心および BPEL 中心の編集スタイル

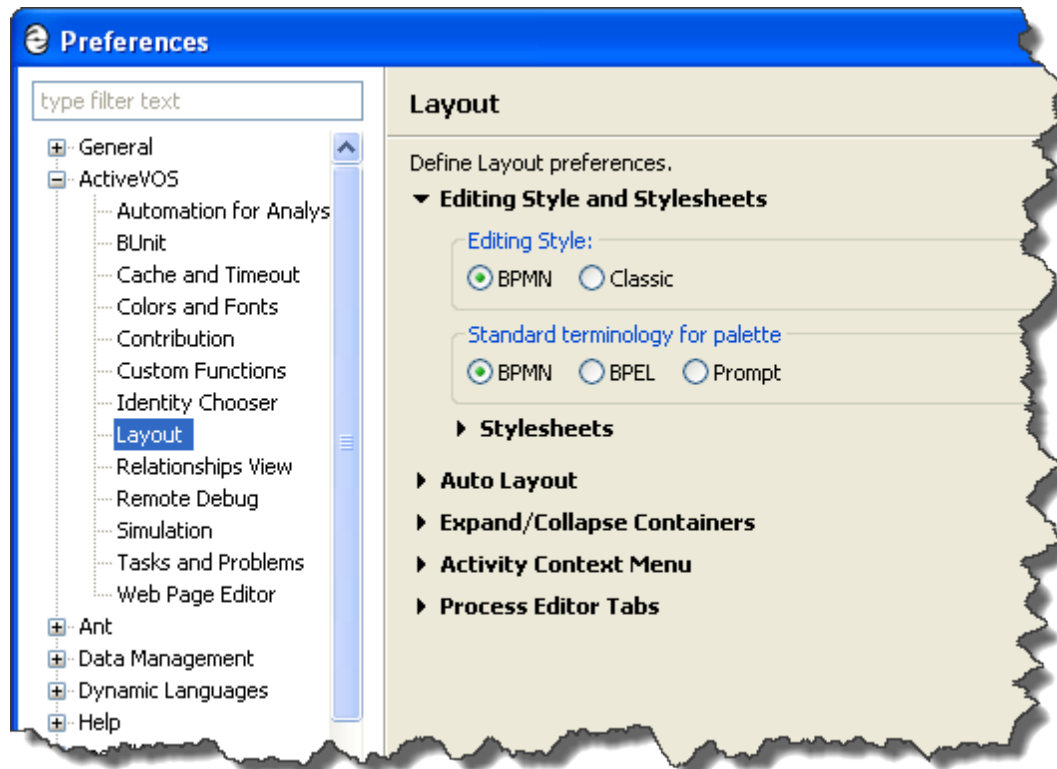
BPEL プロセスを設計するために、Process Developer には「ビジネスプロセスモデリング表記法 (BPMN)」および「BPEL 中心バージョンの BPMN」という 2 つの編集スタイルが用意されています。

Process Editor キャンバスの編集スタイルは、「[レイアウトの設定](#)」(ページ 43)で選択できます。

- **BPMN 中心**の編集スタイルには、ビジネスプロセスモデリング表記法 V2.0 標準に記載されているグラフィカルな表記法要素が含まれます。
- **BPEL 中心**の編集スタイルには、すべての BPEL 構成および BPMN モデリングで一般的に使用される一部のグラフィック要素が含まれます。

どちらの編集スタイルにおいてもすべての BPMN 構成が BPEL 構成にシリアル化されるため、編集したプロセスは BPEL で完全に実行可能です。

新しいプロセスは、レイアウト設定で指定した編集スタイルに関連付けられます。レガシーのプロセスは、作成したスタイルで開かれます。



レガシーのプロセスの場合、元の **Process Developer Classic** の編集スタイルがサポートされます。詳細については、[「Process Developer Classic スタイルについて」](#) (ページ 54)を参照してください。

関連事項:

- [「BPMN 中心のスタイルについて」](#) (ページ 48)
- [「BPEL 中心のスタイルについて」](#) (ページ 48)
- [「BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較」](#) (ページ 50)
- [「どちらの編集スタイル \(BPMN 中心または BPEL 中心\) を選択するかについて」](#) (ページ 52)

BPMN 中心のスタイルについて

ビジネスプロセスモデリング表記法 (BPMN) とは、分析、文書化、および実行のためのアクティビティのフローをモデル化するためにビジネスアナリストや設計者が通常使用するグラフィック指向の表記法標準です。この標準に従って、BPMN は、ビジネスプロセス設計とプロセス実装の間の隔たりを埋める標準化された表記法を提供します。また、BPMN は、BPEL をビジネス指向の表記法によって視覚化できるようにすることを目的としています。

BPMN 2.0 標準は、Object Management Group (OMG) によって管理されています。

図形と記号の使用規則

BPMN では、図形、記号、およびコネクタを使用して構成が表されます。こうした図形、記号、およびコネクタには、標準の使用規則と標準化されたオペレーショナルセマンティクスがあります。BPMN には、広範な注釈機能および文書化機能も含まれています。

主な図形は次のとおりです。

- 角丸長方形 (アクティビティ)
- ひし形 (ゲートウェイ)
- イベント駆動型の境界線スタイル (実線、細線、太線、点線) を持つ円 (イベント)

BPMN 要素の完全な説明については、[「BPMN から BPEL へのタスクとイベントの実装」](#) (ページ 153)および [「BPMN から BPEL へのゲートウェイの実装と制御フロー」](#) (ページ 217)を参照してください。

BPEL 中心のスタイルについて

BPEL 中心のスタイルには、BPEL 構成を表す BPMN グラフィック表記が含まれます。BPEL パレットには、次のような表記が含まれます。

- フローとシーケンスを除いて、各 BPEL 構成はパレットに表示されます。これらの構成は、プロセス設計を効率的に行えるようにするため、キャンバス上に暗黙的に表されます。例えば、Receive をキャンバスにドラッグすると、自動的にシーケンスにエンクローズされます。次のアクティビティを追加すると、Receive に自動的にリンクされます。
- 一部の BPMN 要素が含まれます。プロセスに視覚的な明瞭性を加えるため、start、end、none イベント、および fork-join と gateway という制御フローを使用します。
- Receive などの一部の BPEL アクティビティは、複数の BPMN 構成で表示されます。例えば、Receive はメッセージキャッチイベントまたは Receive タスクとして表示されます。Receive の [プロパティ] ビューで、[次として表示] リストから [BPMN] 表示スタイルを選択します。

BPEL および BPMN 構成の特別な使用法

次の表は、非表示の BPEL 構成と、BPEL 編集スタイルで使用される BPMN 構成を示しています。

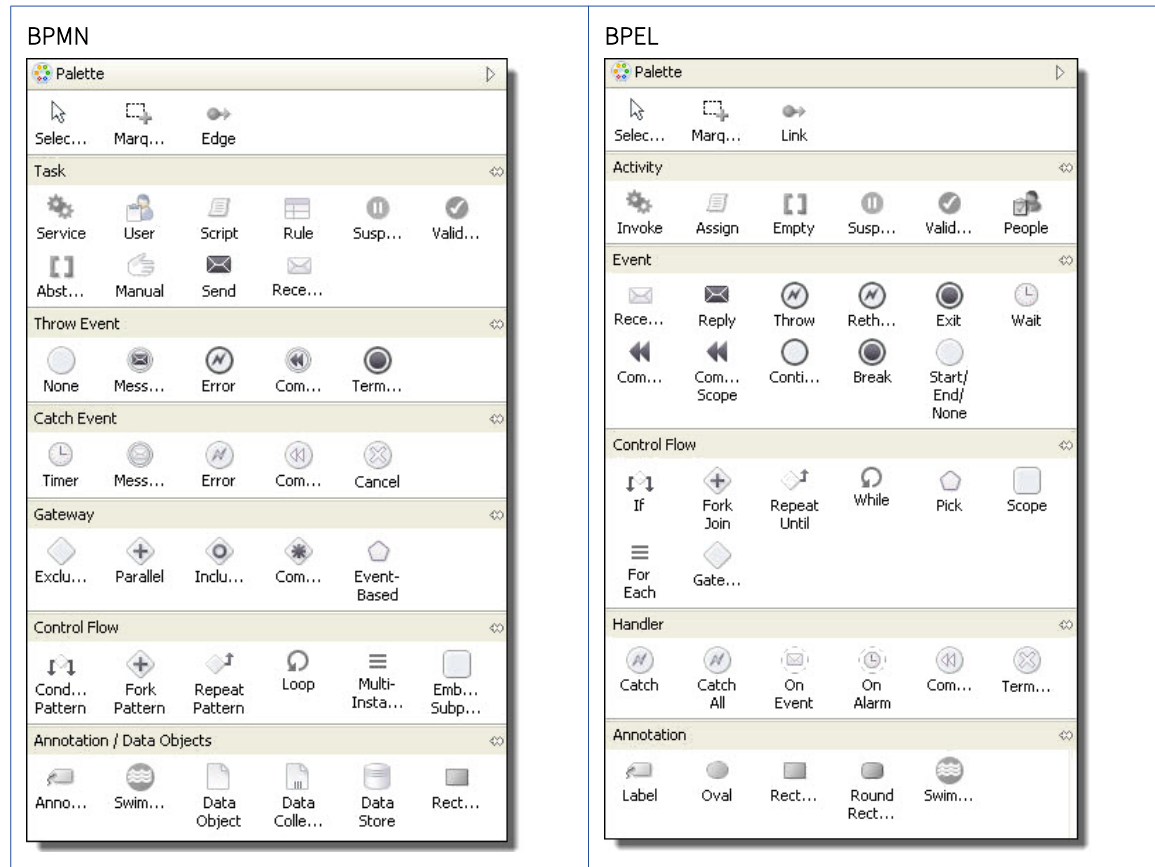
BPMN で非表示の BPEL 構成	BPMN 構成 (BPEL の用語の一部ではないもの)
フロー (必要に応じてプロセス設計に自動的に組み込まれます)	「Fork Join」 (ページ 227)
シーケンス (アクティビティのグループを構成するために自動的に組み込まれます)	「ゲートウェイ」 (ページ 219)
あいまい (使用されません)	「Start/End/None」 (ページ 209)

関連事項:

- [「BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較」 \(ページ 50\)](#)
- [「どちらの編集スタイル \(BPMN 中心または BPEL 中心\) を選択するかについて」 \(ページ 52\)](#)

BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較

以下の図は、2つのパレットを示しています。また、その後に各パレットの説明を記載しています。



次の表では、パレットの違いについて説明しています。

BPMN 中心のパレット

BPMN 中心のパレットは、次のグループで編成されています。

エッジと選択	エッジを使用して 2 つのアクティビティを接続し、それらを順番に実行します。キャンバス上の項目セットを選択するには、他の選択ツールを使用します。
タスク	プロセスを構築するための基本的なプロセスアクティビティ。例えば、Web サービスエンドポイントを表すサービスタスクを選択します。
Throw イベント	Reply、Invoke、Throw、Rethrow、Compensate、Exit、および Break など、(結果に対して) 影響のあるイベント。これらはアウトバウンドイベントです。
Catch イベント	トリガによって引き起こされた、フォールト、終了、イベント、および補償を含む Wait、Receive、およびスコープハンドラなどのイベント。これらはインバウンドイベントです。

ゲートウェイ	パスのブランチ、分岐、マージ、および結合の処理といった、シーケンスフローの分岐と収束を制御します。イベントベースのゲートウェイとして BPEL の Pick アクティビティが含まれます。
制御フロー	アクティビティのグループを構成するためのコンテナ。
注釈/データオブジェクト	ラベル、図形、スイムレーンをキャンバスに追加します。これらは XML コードでは無視されますが、印刷時に出力されます。ラベルはアクティビティに固定できます。
カスタム	BPEL 図面のアクティビティ、またはアクティビティのセットをカスタムアクティビティとして保存します。カスタムアクティビティは他の BPEL 図面で再利用できます。デフォルトでは、カスタムパレットは空の場合に非表示になります。詳細については、 「カスタムアクティビティの作成」 (ページ 164) を参照してください。

BPEL 中心のパレット

BPEL 中心のパレットは、次のグループで編成されています。

リンクと選択	リンクを使用して 2 つのアクティビティを接続し、それらを順番に実行します。キャンバス上の項目セットを選択するには、他の選択ツールを使用します。
アクティビティ	プロセスを構築するための基本的なプロセスアクティビティ。例えば、Web サービスエンドポイントを表す呼び出しを選択します。
イベント	Receive などのイベントをトリガするアクティビティを選択して、プロセスを開始します。
制御フロー	コンテナまたはゲートウェイを選択して、アクティビティのグループを構成します。
ハンドラ	スコープにフォールトまたはイベントハンドラを選択します。
注釈	ラベル、図形、スイムレーンをキャンバスに追加します。これらは XML コードでは無視されますが、印刷時に出力されます。ラベルはアクティビティに固定できます。
カスタム	BPEL 図面のアクティビティ、またはアクティビティのセットをカスタムアクティビティとして保存します。カスタムアクティビティは他の BPEL 図面で再利用できます。デフォルトでは、カスタムパレットは空の場合に非表示になります。詳細については、 「カスタムアクティビティの作成」 (ページ 164) を参照してください。

関連事項: [「どちらの編集スタイル \(BPMN 中心または BPEL 中心\) を選択するかについて」](#) (ページ 52)

パレットの使用に関するヒント:

- デフォルトでは、パレットは自動非表示モードになっています。パレットにマウスを置くとパレットが開きます。
- パレットの自動非表示機能を解除するには、**[パレットの表示]** または **[パレットの非表示]** 矢印を選択します。
- カスタマイズオプションのリストを表示するには、パレットグループのタイトルを右クリックします。
- グループを開いたり閉じたりするには、パレットグループのタイトルをクリックします。
- BPEL プロセスに注釈を付けるには、図形オブジェクトを使用します。
- 小さなアイコンを表示する場合、またはアイコンのみを表示する場合は、パレットをカスタマイズします。アイコンのみを表示するには、任意のパレットエントリを右クリックして、**[レイアウト]** > **[アイコンのみ]** を選択します。

どちらの編集スタイル（BPMN 中心または BPEL 中心）を選択するかについて

どちらの編集スタイル（BPMN 中心または BPEL 中心）を選択するかについて

BPMN および BPEL の編集スタイルによって、完全な検証を行った実行可能な BPEL XML コードを生成します。どちらのスタイルにおいても、プロセス記述の表記を図として表現します。ユーザーやユーザーのチームにとってより適切な視覚スタイルを選択することができます。

BPMN 中心の注目すべき点

- この編集スタイルでは、標準の BPMN 用語を使用します。
- Process Developer の Break 拡張アクティビティは、スコープ内の *Terminate* アクティビティを使用して、For Each または While 構造化アクティビティに実装できます。Break の実装により、最も近いスコープまたはループから抜け出すことができます。
- BPMN の注釈には正確な Continue セマンティクスがないため、Process Developer の Continue 拡張アクティビティはサポートされていません。
- エラーキャッチイベントは、Catch および Catch All フォールトハンドラに使用します。
- Error スローイベントは、Throw および Rethrow アクティビティに使用します。
- 注釈には、データオブジェクト、コレクション、およびストアのアイコンが含まれます。
- 注釈はアクティビティにリンクできます。
- 条件付きパターンや繰り返しパターンなどの制御フロー構造は、個別のアクティビティとしてグループ化を解除できます。

BPEL 中心の注目すべき点

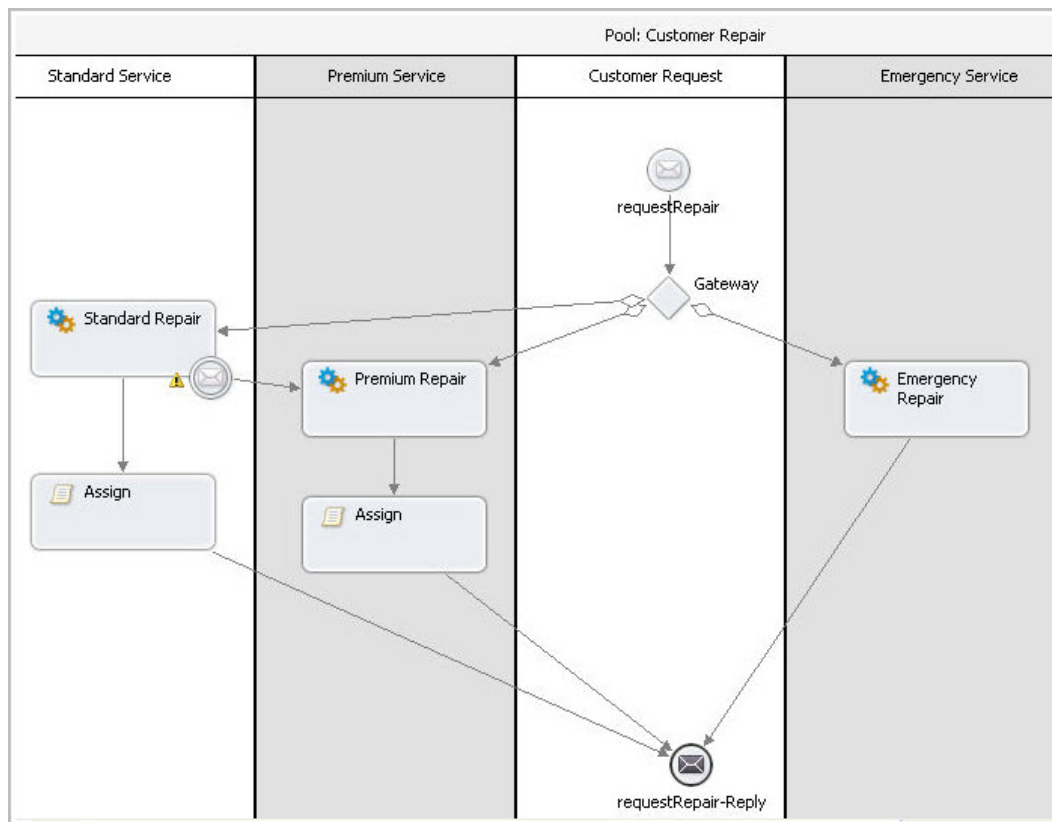
- このスタイルでは、BPEL 用語を使用して構成を行います。また、一部の一般的な BPMN 構成が含まれます。すべての BPEL 構成が表示されるわけではありません。より見やすく明確な図を表示するため、一部の構成は非表示になっています。
- このスタイルには、Process Developer 拡張アクティビティの Break と Continue が含まれます
- Catch および Catch All フォールトハンドラは別々に表されます
- Throw および Rethrow アクティビティは別々に表されます

スイムレーンの使用

スイムレーンのあるプールの追加することで、プロセスのパーティシパントをグラフィカルに表示できます。プールとスイムレーンの概念は、BPMN 仕様の一部です。プールはプロセスを表し、スイムレーンはパーティシパントを表します。

Process Developer では、通常、各パーティシパントのアクティビティを表示する場合に、Process Editor キャンバスにスイムレーンを追加します。しかし、スイムレーンアクティビティ間の関係を示すリンクを使用して、アクティビティを別のレーンにドラッグアンドドロップすることができます。プロセス全体のプールは 1 つだけです。

次の図は、プロセスのパーティシパントごとに 1 つずつ、4 つのスイムレーンがあるプロセスプールの例を示しています。



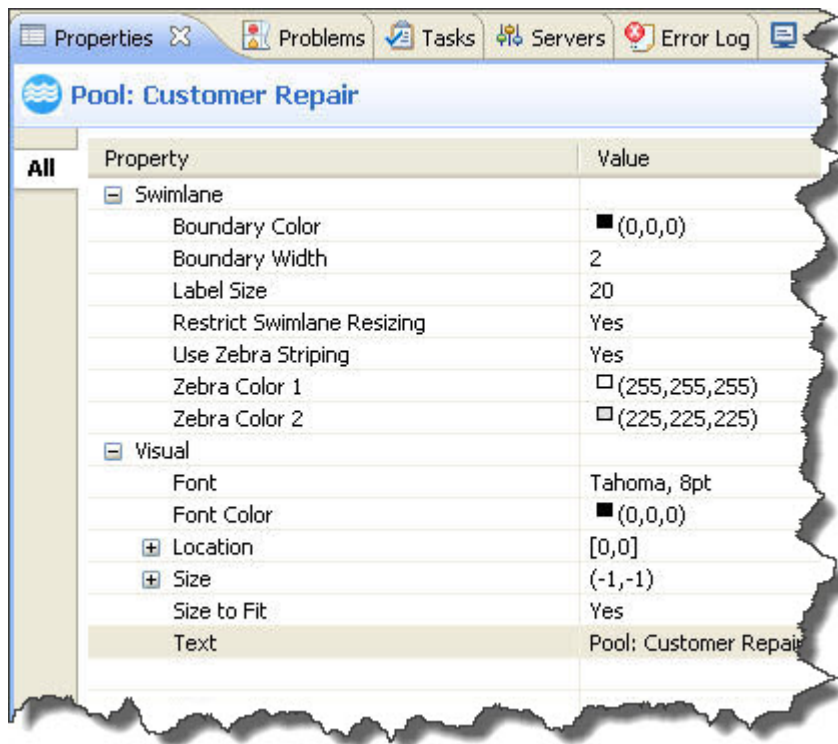
スイムレーンを追加して使用する手順

1. BPMN エディタのキャンバスを右クリックし、[追加] > [注釈] > [スイムレーン] を選択します。
2. プールと 1 つのスイムレーンが作成されることに注意してください。プール名はプロセス名であり、スイムレーンは無題です。
3. 次のいずれかを実行します。
 - 最初のスイムレーンの左側（または上部）にスイムレーンを追加するには、プロセスの左側（または上部）を右クリックし、[追加] > [注釈] > [スイムレーン] を選択します。
 - 最初のスイムレーンの右（または下）にスイムレーンを追加するには、最初のスイムレーンの右（または下）を右クリックし、[追加] > [注釈] > [スイムレーン] を選択します。
4. スイムレーンに名前を付けるには、スイムレーンのタイトルバーを選択し、[プロパティ] ビューを表示します。[テキスト] フィールドで、[無題] を適切なテキストに置き換えます。

スイムレーンの色を変更する場合は、最初に縞模様をオフにする必要があります。

スイムレーンの操作に関するヒント:

- デフォルトでは、スイムレーンは縞模様と呼ばれる白とグレーの帯で表示されます。縞模様を無効にし、各スイムレーンに個別の色を設定するには、プールのタイトルバーを選択し、[プロパティ] ビューを開きます。必要に応じてプロパティを設定します。



- スイムレーンを削除するには、スイムレーンを右クリックして【削除】を選択します。プールを削除するには、すべてのスイムレーンを削除します。
- スイムレーンには視覚的なプロパティしかないため、スイムレーン内のすべてのアクティビティを手動で配置する必要があります。
- パレットの【注釈】ドロワーからスイムレーンを追加できます
- スイムレーンはメインキャンバスで使用できますが、折りたたまれたコンテナの【ドリルダウン】ビューや、フォールトまたはイベントハンドラのタブでは使用できません。

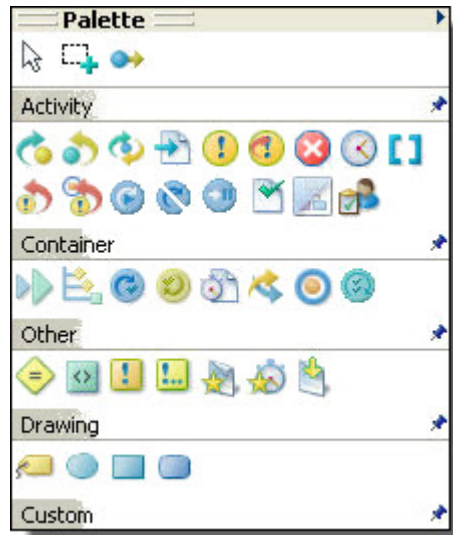
スイムレーンのサイズ変更

スイムレーンを拡大または縮小するには、追加または削除するアクティビティに合わせて、スイムレーンの境界線を水平方向に左または右に移動します（垂直方向の場合は上または下に移動します）。デフォルトでは、スイムレーンの幅が小さくなりすぎてアクティビティが見えなくなならないように、スイムレーンにはサイズ変更の制限が設定されています。プールの【スイムレーンのサイズ変更の制限】プロパティは【はい】に設定されています。スイムレーンのサイズを自由に変更できるようにするには、このプロパティを【いいえ】に設定します。

Process Developer Classic スタイルについて

Process Developer Classic スタイルは、主にレガシープロセスでサポートされています。

Process Developer Classic は、Process Developer のバージョン 1.0 以降に使用されているパレットと同じアイコンパレットを使用します。WS-BPEL 2.0 仕様で指定されているように、各 BPEL 構成は、Informatica が設計したアイコンで表されます。BPEL 仕様にはグラフィック表現の要件は含まれないため、この仕様を実装するベンダ各社では独自の外観を作成しています。



Process Developer Classic では、パレットドローの名称は Activity、Container、および Other です。

BPMN スタイルと Process Developer Classic スタイルの例

次の例は、BPMN スタイルと Process Developer Classic スタイルの主な違いを示しています。

例 1: 図形と記号

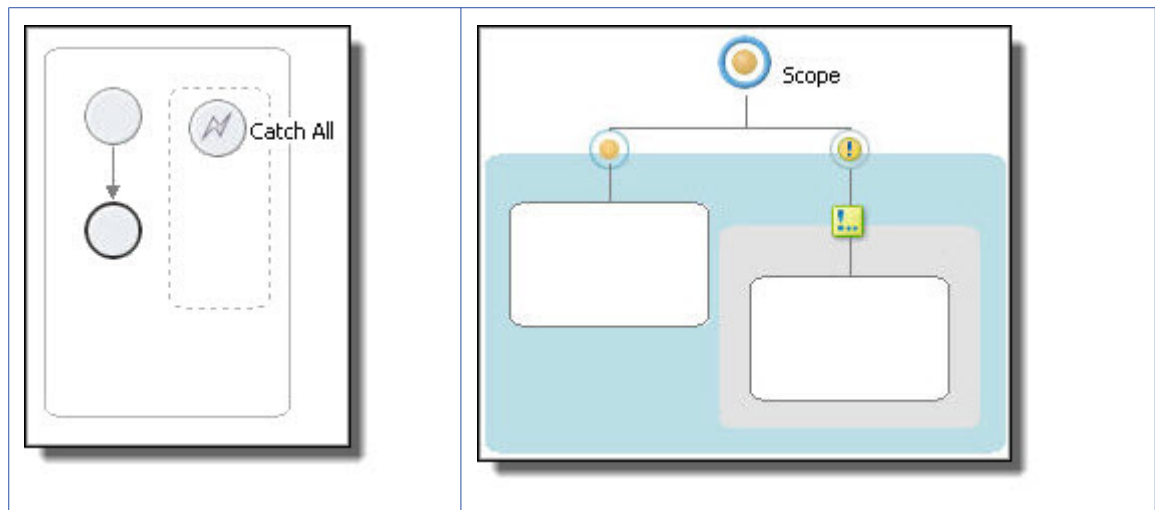
BPMN の図形には意味があります。左下の Receive と Reply に示されているように、円はイベントを示します。開始イベントと終了イベントには実線の境界線があり、中間イベントには太線の境界線があります。Process Developer Classic では、各アクティビティは基本アクティビティと構造化アクティビティに分類されます。Receive と Reply は基本アクティビティであり、右側に示されている Informatica 独自のアイコンで表されます。



例 2: 開始、終了、および例外イベント

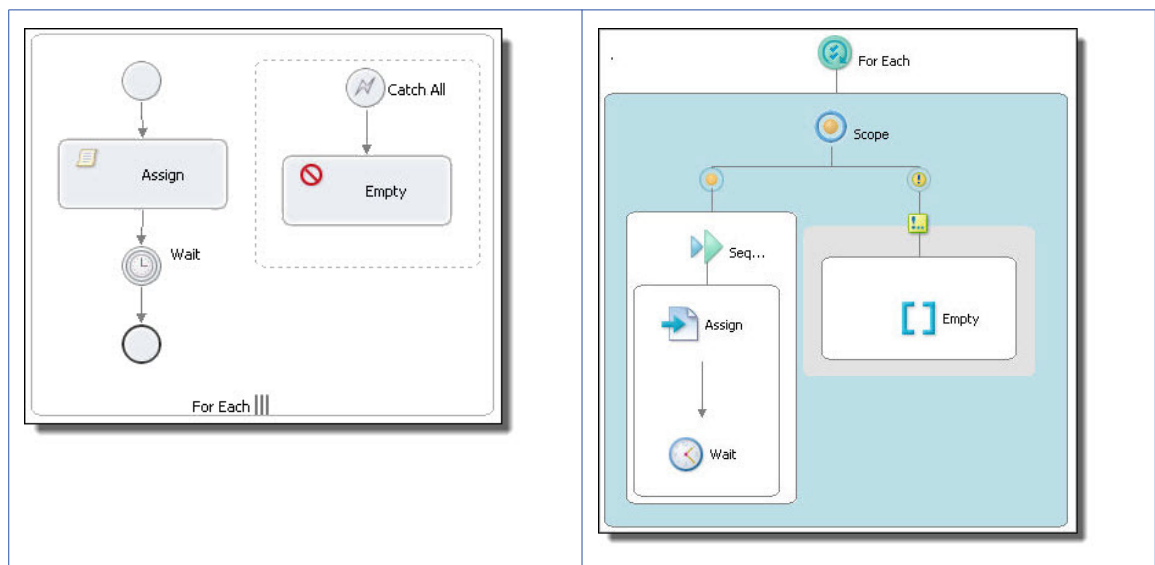
BPMN は、左下のスコープに示すように、イベントアイコンによって、ワークフローの開始と終了（BPMN の用語ではサブプロセス）を示します。開始イベントと終了イベントは、非表示の Sequence コンテナ内にあります。アクティビティは開始と終了の間に追加します。スコープに Catch フォールトハンドラを追加すると、このフォールトハンドラはスコープ内に点線の長方形として表示されます。Process Developer Classic では、

スコープは空のコンテナとして開始されます。[フォールトハンドラの表示] オプションを使用してフォールトハンドラを表示します。次に、このフォールトハンドラに Catch を追加します。



例 3: 制御フローとコンテナ

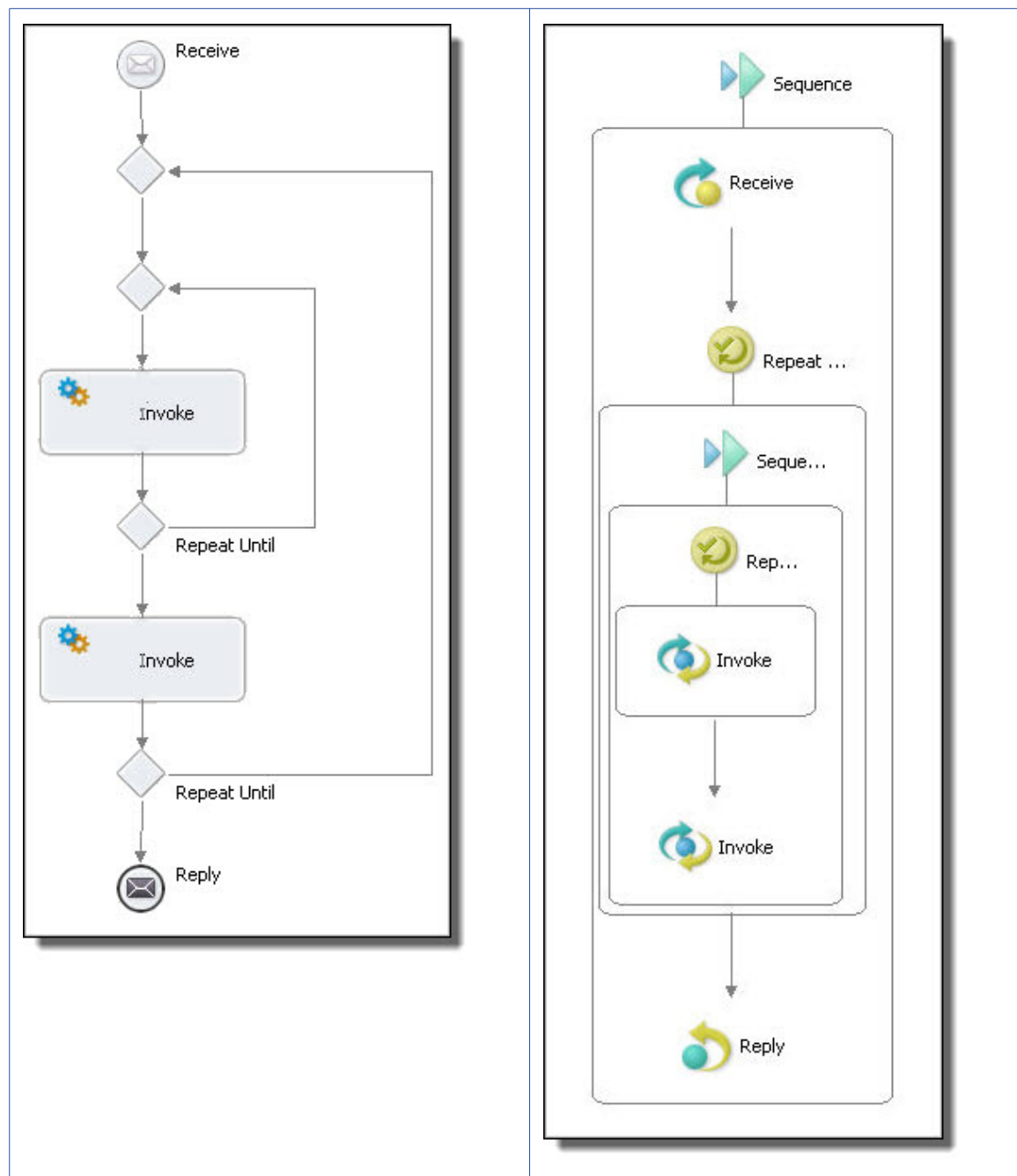
BPEL では、forEach、If、While、RepeatUntil などのネストされたアクティビティが必要となることがよくあります。BPMN（左下）は、制御フローの矢印を使用したネストを表しています。forEach の例では、制御フローに埋め込まれた（非表示の）シーケンスが含まれていることにも注意してください。シーケンスには開始イベントと終了イベントが含まれており、その間にアクティビティを追加できます。開始イベントと終了イベントは BPMN では適切なスタイルと見なされますが、厳密には必須ではないため、必要に応じて削除することもできます。Process Developer Classic では、すべてのアクティビティを手動で追加する、表示可能なスコープが埋め込まれた forEach コンテナが表示されます。



例 4: ネストされたアクティビティ

BPMN では、矢印は制御フローを表します。ネストされた RepeatUntil アクティビティの例に示すように、矢印を使用して表すと、コンテナを使用するよりも制御フローが見やすくなります。この図では、親の RepeatUntil

には、RepeatUntil と Invoke という 2 つの子があります。子の RepeatUntil には、Invoke という 1 つの子があります。BPMN の図は、よりわかりやすい形式であると言えます。



特定の編集スタイルから別の編集スタイル（BPMN およびクラシック）への保存

Process Developer バージョン 7.0 では、BPMN 編集スタイルが導入されています。このスタイルは、ビジネスプロセスモデリング表記法仕様の一般的なアイコンと記号に基づいています。ユーザーやユーザーのビジネスパートナーは、ビジネスアナリストや開発者によって広く使用されているワークフロースタイルのアイコンや記号を使い慣れているかもしれません。

必要に応じて、特定の編集スタイルから別の編集スタイルにプロセスを変換することができます。Process Developer Classic 編集スタイルで作成したプロセスを BPMN スタイルとして保存することができ、またその逆も可能です。

特定の編集スタイルから別の編集スタイルに変換するには、Process Editor でプロセスを開き、**[ファイル] > [名前を付けて保存]** を選択します。**[名前を付けて保存]** ダイアログで、必要な編集スタイル（BPMN またはクラシック）を選択します。詳細については、[第 5 章, 「BPMN 中心および BPEL 中心の編集スタイル」](#)（[ページ 47](#)）を参照してください。

第 6 章

Process Developer のユーザーインターフェース

この章では、以下の項目について説明します。

- [Process Developer のメニューとツールバー, 59 ページ](#)
- [ウィンドウ、パースペクティブ、ビュー、およびエディタ, 65 ページ](#)
- [Process Developer のパースペクティブ, 65 ページ](#)
- [Process Developer の \[デバッグ\] パースペクティブ, 73 ページ](#)
- [Process Developer Process Editor の使用, 73 ページ](#)

Process Developer のメニューとツールバー

Eclipse の基本メニューとツールバーに加えて、Process Developer の一部として、次のメニュー項目とツールバーボタンが表示されます。

[ファイル] メニュー

- [「リファクタリング」 \(ページ 60\)](#)

[ナビゲーション] メニュー

- [「アクティビティに移動 \(Ctrl + I\)」 \(ページ 61\)](#)
- **宣言を開く (F3)**。キャンバスでフォーカスされている BPEL 構成に関連付けられた WSDL または XSD ファイルに移動します。例えば、Invoke を選択してから、WSDL を開いて、強調表示された操作を確認する場合に F3 キーを押します。
- [「操作を開く \(Ctrl + Shift + o\)」 \(ページ 61\)](#)
- [「ポートタイプを開く \(Ctrl + Shift + t p\)」 \(ページ 61\)](#)
- [「Web タイプを開く \(Ctrl + Shift + t w\)」 \(ページ 61\)](#)

[プロセス] メニュー

[プロセス] メニューの項目は、Process Editor キャンバスでフォーカスされているプロセスに関連付けられています。詳細については、以下のトピックを参照してください。

- [「Process Developer Process Editor の使用」 \(ページ 73\)](#)
- [「プロセスレポートの生成」 \(ページ 62\)](#)
- [「プロセスイメージの生成」 \(ページ 62\)](#)

- [「デプロイメントイメージの生成」](#) (ページ 63)
- [「プロセスの検証」](#) (ページ 62)
- [「パラメータの最適化」](#) (ページ 61)

ツールバーボタン

標準のツールバーボタンに加えて、次のグループは BPEL プロセスに固有のボタンです。

- プロセス表示支援: **[自動レイアウト]** (以下の詳細を参照)、**[整列]**、**[拡大]**
- シミュレーション: **[シミュレーションの開始]**、**[実行状態のクリア]**
- プロセス検証 (保存なし): **[プロセスの検証]**

自動のレイアウト方法

次のように **[レイアウト方法]** を選択します。

方法	説明
グリッド (デフォルト)	オブジェクトは右/左 (または上/下) に分岐しますが、使用可能な余白に対してレイアウトは最適化されません。行またはカラムは、その行またはカラム内のオブジェクトの最大幅となります。
ツリー	オブジェクトは右/左 (または上/下) に分岐し、使用可能な余白を使用するようにレイアウトが最適化されます。
階層	1 つのメインのトランクとアウトラインでオブジェクトが整列され、分岐は右側に向かって進みます

単純なプロセスでは、各レイアウト方法に差異がない場合もあります。

リファクタリング

リソースを選択して名前を変更すると、そのリソースをインポートするファイルまたはそのリソースに依存するファイルも更新されます。例えば、プロセス名を変更すると、関連する B ユニットテストおよびプロセスデプロイメント記述子の参照が自動的に更新されます。

[リファクタリング] を選択すると **[リソースの名前変更]** ダイアログが開き、**[プレビュー]** を選択することで、変更される内容を確認できます。

リソースは移動することもできます。例えば、WSDL を移動した場合、他のすべての WSDL、BPEL、BPRD、および PDD は、移動した WSDL を指すように更新されます。

以下のリソースの名前を変更 (または移動) します。

- **プロセス名**。名前を変更した場合でも、.bpel ファイル名は変更されないことに注意してください。名前の変更によって、BPEL プロセスの <process name> 要素が変更されます。
- **WSDL のファイル名**
- **スキーマのファイル名**
- **スキーマの要素またはタイプ**
- **ターゲット名前空間**

WSDL エディタでは、ターゲット名前空間をリファクタリングできます。このアクションによって、WSDL または XSD を参照するすべてのリソースの名前空間のグローバルリファクタリングがトリガされます。WSDL でポートタイプを強調表示している場合、この要素を参照するリソースを更新するには、マウスを右クリックして **[リファクタリング]** > **[名前の変更]** を選択します。複合型用のスキーマエディタでも同様の操作が可能です。

アクティビティに移動 (Ctrl + I)

【アクティビティに移動】ダイアログを使用して、フォーカスされている BPEL プロセスの名前付きアクティビティを選択します。

【ナビゲート】 > 【アクティビティに移動】を選択して、Process Editor キャンバスでフォーカスされている BPEL プロセスの名前付きアクティビティを選択します。【BPEL アクティビティの検索】テキストボックスに入力を開始すると、名前に一致するようにアクティビティリストがフィルタリングされます。選択したアクティビティのエンクロージングスコープとプロセスの場所が表示されます。

操作を開く (Ctrl + Shift + o)

【操作を開く】ダイアログを使用して、WSDL 内の操作を見つけます。

操作とは、Web サービスアクティビティの実行時に実行するコマンドです。関連する一連の操作は、WSDL のポートタイプにバンドルされています。Receive、Reply、Invoke などの BPEL アクティビティには操作が含まれます。

【ナビゲート】 > 【操作を開く】を選択し、【操作を開く】ダイアログを使用して WSDL 内の操作を見つけることができます。ワークスペース WSDL から操作の名前の入力を開始すると、自動入力された名前のリストが表示されます。WSDL のターゲット名前空間がダイアログに表示されます。操作をダブルクリックすると WSDL が開き、操作が強調表示されます。

ポートタイプを開く (Ctrl + Shift + t p)

【ポートタイプを開く】ダイアログを使用して、WSDL 内のポートタイプを見つけます。

WSDL のポートタイプとは、関連する操作のセットです。Receive、Reply、Invoke などの BPEL アクティビティには操作が含まれます。

【ナビゲート】 > 【ポートタイプを開く】を選択し、【ポートタイプを開く】ダイアログを使用して、WSDL 内のポートタイプを見つけることができます。ワークスペース WSDL からポートタイプの名前の入力を開始すると、自動入力された名前のリストが表示されます。WSDL のターゲット名前空間がダイアログに表示されます。ポートタイプをダブルクリックすると WSDL が開き、ポートタイプが強調表示されます。

Web タイプを開く (Ctrl + Shift + t w)

【ナビゲート】 > 【Web タイプを開く】を選択し、【Web タイプを開く】ダイアログを使用して、WSDL メッセージ、スキーマ要素、またはスキーマタイプを見つけることができます。ワークスペース WSDL または XSD から任意の Web タイプの名前の入力を開始すると、自動入力された名前のリストが表示されます。WSDL または XSD のターゲット名前空間がダイアログに表示されます。メッセージ、要素、またはタイプをダブルクリックして WSDL または XSD を開き、項目を強調表示します。

パラメータの最適化

【パラメータの最適化】コマンドにより、次の条件に基づいてプロセスに変更を加えます。

- 変数は、単一の割り当てアクティビティ内でのみ割り当てられ、初期化式を持つ可能性があります。
- 割り当ては、変数を入力として（シーケンスまたはリンクを介して）使用するメインアクティビティの直前にあります。
- 割り当ては、XQuery 割り当てタイプに変換される資格を持ちます。つまり、コピー式は 1 つだけで、コピーには XQuery を使用する from 式があり、変数の初期化式はありません。

割り当ては、from 句が XPath を使用する式、リテラル、プロパティを持つ変数、オプションとしてパートまたはクエリを持つ変数である場合、XQuery 割り当てタイプに変換される資格を持ちます。

実行される変更

実行される可能性のある変更は次のとおりです。

- 変数が要素タイプである場合、メッセージタイプに変更され、変数を設定するすべてのコピーが新しい「パート」属性を取得します。
- 変数が要素タイプであり、初期化式を持つ場合、この初期化式は、Assign アクティビティの最初の「コピー」になるように移動されます。
- この変数が、Assign ステートメントの設定を行う、割り当てられた唯一のステートメントである場合は、割り当て全体を暗黙的なスコープ内に移動し、割り当てのすべてのインバウンドリンクとアウトバウンドリンクを暗黙的なスコープに移動します（既存のインバウンドリンクまたはアウトバウンドリンクに追加します）。それ以外の場合は、割り当てから暗黙的なスコープ内の新しい Assign アクティビティにコピーのみを移動します。

実行可能なアクションは次のとおりです。

- 変数を暗黙的なスコープに移動し、名前（およびそのすべての使用法）を「parameters」に変更します。
- コピー「from」コンストラクトを変換して式を使用するようにします（式がリテラルからのものでない場合）。`query="/foo/bar"`を持つ `variable.part` から `$parameters.part/foo/bar` に変更し、`property="ns:foo"` を持つ変数から `abx.getVariableProperty('parameters','ns:foo')` に変更します。また、`variable.part (query?)` からコピーから、式からコピーに変更します。

プロセスの検証

Eclipse Validation Builder を使用してプロセスを検証するには、このコマンドを選択します。プロセスを保存すると、プロセスの検証が自動的に実行されます。ただし、大規模なプロセスの場合は、保存に長い時間がかかる場合があります。このコマンドを使用することで、保存せずに検証を行うことができます。**【プロセスの検証】** 用のツールバーボタンとショートカットキーシーケンス（Ctrl + Shift + V）があることに注意してください。

検証の詳細については、[「プロジェクトのオーケストレーションおよび検証ビルダについて」](#)（ページ 88）を参照してください。

プロセスレポートの生成

プロセスレポートの生成を使用して、プロセス内のビジネスパーティシパントとアクティビティを要約したファイルを作成およびエクスポートできます。

レポートを生成する手順

1. [プロセス] メニューから、**【プロセスレポートの生成】** を選択します。
2. ダイアログで、ファイルを保存するファイル名と場所を入力します。

ヒント: レポートを生成する前に、プロセスの [プロパティ] ビューおよび任意のプロセス構成（特にアクティビティ）の [プロパティ] ビューにドキュメントを追加します。詳細については、[「プロセスへのドキュメントの追加」](#)（ページ 79）を参照してください。

プロセスイメージの生成

[プロセス] メニューの **【プロセスイメージの生成】** コマンドを使用して、キャンバスにフォーカスされているプロセスの単一の JPG イメージを作成します。ファイルは、選択したファイルシステムの場所に保存されます。

デプロイメントイメージの生成

プロセスを保存すると、バックグラウンドでプロセスのスクリーンショットが自動的に保存されます。プロジェクトをデプロイすると、プロセスコンソールはこのイメージを使用して、プロセスの [詳細グラフ] ビューを表示します。プロセスコンソールに表示されるプロセスは、Process Editor キャンバスのプロセスと一致します。

プロセスイメージの自動保存を無効にして、手動でイメージを生成することができます。[デプロイメントイメージの生成] オプションを有効にするには、プロセスの [プロパティ] ビューに移動し、[すべて] タブを選択して、[イメージの生成] オプションを ([自動] から) [手動] に変更します。次に、サーバーでレイアウトを完全に一致させる場合は、デプロイする前に [デプロイメントイメージの生成] を選択する必要があります。

イメージを生成しない場合は、BPEL プロセスのデフォルトのレイアウトが使用されますが、これは必ずしもキャンバス内の正確なレイアウトと一致するとは限りません。

[手動] オプションは、大規模なプロセスのファイル保存のパフォーマンスを向上させるために使用します。長時間にわたるファイルの保存が発生した場合は、プロセスの [イメージの生成] プロパティを [手動] に設定できます。また、[非永続的] としてデプロイしたプロセスの自動イメージ生成を無効にすることもできます (プロセス情報は、プロセスの終了時に保存されません)。非永続的なプロセスは、プロセスコンソールに表示されません。

Process Editor のキーボードショートカット

次の表は、マウスを使用せずに *Process Editor* キャンバス上のオブジェクトを選択して移動する方法を示しています。

ナビゲーションキー

- Process Editor で BPEL ファイルに移動する: CTRL + F6。
- [アクティビティに移動] ショートカットを使用して、選択したアクティビティに移動する: CTRL + L。

[「Process Developer のメニューとツールバー」 \(ページ 59\)](#) も参照してください。

オブジェクトの選択

アクティビティまたはコンテナを選択した状態で、次のキーを使用して別のオブジェクトに移動できます。

キー	アクション
Space	選択
左矢印	左側のオブジェクトに移動します
右矢印	右側のオブジェクトに移動します
上矢印	上のオブジェクトに移動します
下矢印	下のオブジェクトに移動します
/または?	次のリンクに移動します
\または	前のリンクに移動します
Alt + 下矢印	コンテナに移動します
Alt + 上矢印	コンテナの外に移動します

CTRL キーを押すと、選択範囲ではなくフォーカスが移動します。ナビゲーションキーの 1 つを使用している場合に SHIFT キーを押すと、選択範囲を拡大することができます。

オブジェクトの移動

オブジェクトを選択した状態で、以下のキーを使用してオブジェクトを移動できます。隣接する 2 つの矢印キーを同時に選択すると、オブジェクトを斜めに移動できます。

キー	アクション
Alt + マウス	選択したオブジェクトを一度に 1 ピクセルずつ移動します
ピリオド	次のハンドルを選択します
>	前のハンドルを選択します
左矢印	左にドラッグします
右矢印	右にドラッグします
上矢印	上にドラッグします
下矢印	下にドラッグします
Enter	ドラッグ操作を確定します
Esc	ドラッグ操作を取り消します

パレットキーボードアクション

キー	アクション
左矢印	アクティビティグループなどの展開されたパレットグループにフォーカスがある場合は、そのグループを折りたたみます。それ以外の場合は、グループにフォーカスが設定されます。
右矢印	折りたたまれたパレットグループにフォーカスがある場合は、そのグループを展開します。グループが展開されている場合は、グループにフォーカスが移動します。
上矢印	グループ内にフォーカスがある場合、グループタイトルに移動します。
下矢印	次の項目に移動します

Process Developer のファンクションキー

以下のファンクションキーが使用可能です。

場所	キー	機能
デバッグビュー	F6	ステップオーバー
デバッグビュー	F8	レジューム

ウィンドウ、パースペクティブ、ビュー、およびエディタ

Process Developer では、Eclipse Workbench の基本的なユーザーインターフェースコンポーネント（ウィンドウ、パースペクティブ、ビュー、およびエディタ）を使用します。

Process Developer を起動するたびに、[ワークベンチ] ウィンドウが開き、デフォルトのパースペクティブが表示されます。パースペクティブは、一連のタスクをサポートするビュー、エディタ、メニュー、およびツールバーで構成されます。

それぞれのパースペクティブには複数のビューがあります。ビューは、XML ツリーの表示、タスクの一覧表示、オブジェクトプロパティの表示など、1つのタスク用に設計された一意のウィンドウです。それぞれのビューには独自のツールバーがあります。

また、それぞれのパースペクティブには、ファイルを開くための編集領域があります。エディタはファイルタイプに関連付けられており、適切なファイルアクションを提供します。最も一般的なファイルタイプは編集可能なドキュメントですが、一部のファイルタイプでは、BPEL XML コードの自動生成などの特別なアクションが必要です。それぞれのエディタは同じ領域を共有し、複数のファイルが開いている場合は、各エディタがスタックされます。多くのエディタには、必要に応じてブックマーク、タスク、およびブレークポイントを追加するためのマーカーバーが表示されます。

作業環境をカスタマイズするためのヒントを次に示します。

エディタのヒント	ビューのヒント	パースペクティブのヒント
エディタを全画面表示にするには、タイトルバーをダブルクリックします	ビューを非表示にするには、ビューのタイトルバーを右クリックして、[高速ビュー] を選択します。ビューはアイコンとしてショートカットバーに移動します。ビューを開くには、アイコンをクリックします。	ビューを表示および非表示にするには、[ウィンドウ] > [パースペクティブのカスタマイズ] を選択します。
2つ以上のエディタを並べて表示するには、1つのタイトルバーを編集領域の左端にドラッグします	ビューを並び替えるには、ビューを移動してドッキングし、タブを並び替えます。	2つのウィンドウで同じパースペクティブを開くには、[ウィンドウ] > [新規ウィンドウ] を選択します。

Process Developer のパースペクティブ

Process Developer のパースペクティブはデフォルトの製品パースペクティブで、BPEL プロセス定義を設計および生成するためのタスクをサポートするビュー、エディタ、メニュー、およびツールバーが含まれています。

Process Developer を最初に起動すると、パースペクティブと呼ばれるデフォルトのウィンドウレイアウトが表示されます。パースペクティブは、一連のタスクをサポートするビュー、エディタ、メニュー、およびツールバーで構成されます。

ビューは、ファイルの XML ツリービューの表示、エラーやタスクの一覧表示、オブジェクトプロパティの表示など、1つのタスク用に用意されたパースペクティブ内の一意のウィンドウです。

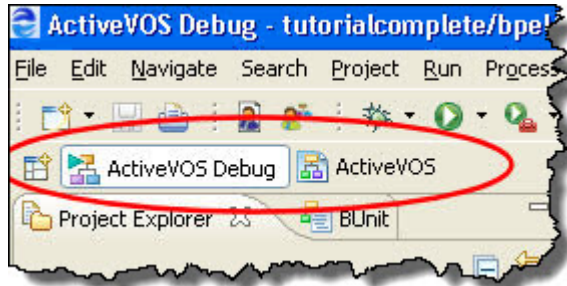
Process Developer には、次の2つのデフォルトのパースペクティブが含まれています。

- [「Process Developer のパースペクティブ」 \(ページ 65\)](#)

- [「Process Developer の \[デバッグ\] パースペクティブ」 \(ページ 73\)](#)

パースペクティブは、カスタマイズ可能な多用途のレイアウトです。詳細については [「Process Developer パースペクティブのカスタマイズ」 \(ページ 73\)](#) を参照してください。

[パースペクティブ (高速) / ビュー] バーのアイコンを使用して、パースペクティブを開いたり閉じたりすることができます。パースペクティブバーのデフォルトの位置は右上です。次の図に示すように、別の場所（ここでは左上）にバーをドッキングできます。

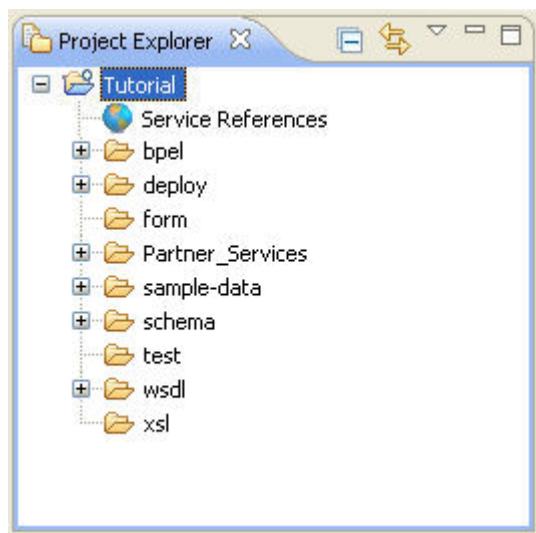


プロジェクトエクスプローラ

プロジェクトエクスプローラには、Process Developer ワークスペースフォルダにあるプロジェクト、フォルダ、およびファイルが表示されます。BPEL プロセスを作成する前に、プロジェクトを作成しておく必要があります。BPEL の拡張プロジェクトタイプは、Orchestration プロジェクトです。詳細については、[「Orchestration プロジェクトの作成」 \(ページ 85\)](#) を参照してください。

プロジェクトエクスプローラの Orchestration プロジェクトには、BPEL プロセスと関連する WSDL、スキーマ、サンプルデータ、テスト、およびデプロイメントファイルで作業するための各種詳細とビルダが含まれています。

プロジェクトエクスプローラは、ファイルシステムのワークスペースフォルダのミラーです。



すべてのファイルを Orchestration プロジェクト内に保存することをお勧めします。

プロジェクトエクスプローラの使用に関するヒント:

- プロジェクトとファイルをファイルシステムからドラッグしてプロジェクトエクスプローラに追加し、[「プロジェクト Orchestration ネーチャーの追加または削除」 \(ページ 88\)](#) に記載されているように、Orchestration ネーチャーを追加します。

- プロジェクトとファイルをインポートします。詳細については、「ワークベンチユーザーガイドのヘルプ」にある「[既存のプロジェクトのインポート](#)」および「[ファイルシステムからのリソースのインポート](#)」を参照してください。
- ファイルシステムからファイルをコピーして貼り付けます。
- ファイルシステムのプロジェクトワークスペースにファイルを追加してから、**[更新]** コマンドを使用してプロジェクトエクスプローラを更新します。詳細については、「ワークベンチユーザーガイドのヘルプ」にある「[ワークベンチ](#)」および「[ワークベンチ外のファイルの編集](#)」を参照してください。
- プロジェクトエクスプローラのさまざまなツールを使用して、ファイルのフィルタ処理、表示、非表示、およびナビゲートを行います。詳細については、「ワークベンチユーザーガイドのヘルプ」にある「[\[プロジェクトエクスプローラ\] ビュー](#)」を参照してください。

パーティシパント

詳細については、「[パーティシパントについて](#)」 ([ページ 138](#))を参照してください。

[イメージの場所] ダイアログ

[イメージの場所] ダイアログでは、ファイルシステムからアクティビティの表示イメージを選択することや、アクティビティのデフォルトのイメージを選択することができます。

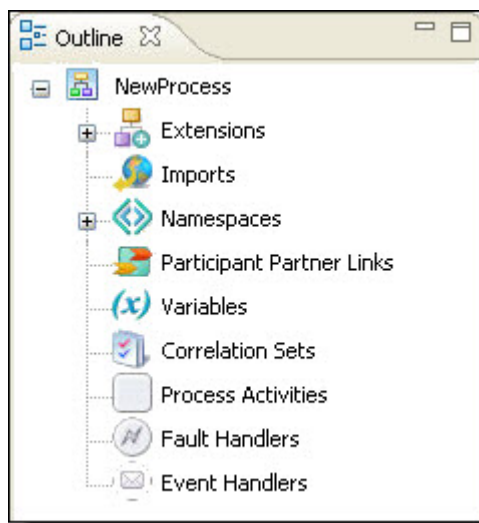
アクティビティアイコンの場所とイメージのファイル名を選択します。

インタフェース

デフォルトでは、このビューは表示されません。代わりに、[パーティシパント] ビューが表示されます。**[ウィンドウ] > [ビューの表示] > [インタフェース]** を選択すると、インタフェースが表示されます。詳細については、「[パーティシパント](#)」 ([ページ 67](#))を参照してください。

[アウトライン] ビュー

[アウトライン] ビューには、以下のように、BPEL プロセスのすべての主要コンポーネントが表示されます。



アウトライン内のノードは、Process Editor および [プロパティ] ビューと同期します。例えば、[プロセス アクティビティ] ノードでアクティビティを選択すると、強調表示されたアクティビティが Process Editor キャンバスに表示されます。

[アウトライン] ビューの使用に関するヒント:

- 同じタイプの新しい項目を追加するには、項目を右クリックします。
- 項目を上下に動かして並べ替えを行います。
- [プロセス変数] ビューで変数を並べ替えて、順序を変更します。
- 使用していない項目は削除します。
- 変数およびその他のコンポーネントをダブルクリックして、**[定義]** ダイアログなどのメインプロパティを開きます。

アクティビティは、作成した順序でアウトラインに追加され、BPEL XML コードにはこの順序が反映されます。[プロセスアクティビティ] ノードをアクティビティの実行順序に並べ替えておくことをお勧めします。

[プロパティ] ビュー

[プロパティ] ビューには、選択したリソースのプロパティの名前と値が表示されます。プロパティタイプに必要な内容（名前の入力、選択リストからのエントリの選択、または [ダイアログ (...) ボタン] の選択による式の作成）に応じて、プロパティ値を入力します。

次の表に、[プロパティ] ビューのさまざまな使用法を示します。

選択されたリソース	プロパティ
プロジェクトエクスプローラファイル	サイズや日付などのファイルプロパティが表示されます
プロセス (Process Editor で開くか、[アウトライン] ビューから選択)	プロセスプロパティ。これらのプロパティは、BPEL XML ファイルの <process> 要素の属性です。
プロセスオブジェクト: アクティビティ、リンク、フォールトハンドラ、イベントハンドラ、変数、関連セット	ビジュアルプロパティとシミュレーションプロパティを含む、必須およびオプションの BPEL プロパティ。各プロパティは編集可能です。
パーティシパントのパートナーリンク	必須およびオプションのプロパティ。各プロパティは編集可能です。
インポート	必須およびオプションのプロパティ
名前空間	必須およびオプションのプロパティ
サービス参照	なし。
エラーログ、[問題] ビュー	なし。詳細は [プロパティ] ビューに表示されません。詳細を表示するには、エラーまたは問題をダブルクリックします。

プロセス変数

[プロセス変数] ビューには、作成した変数、または WSDL 操作を Process Editor キャンバスにドラッグアンドドロップしたときに作成される変数のリストが表示されます。

BPEL プロセスは、XML 変数を介してデータを受信、操作、および送信します。変数は、ビジネスパートナー間で交換するメッセージやプロセス内のみで使用するデータを保持します。

[プロセス変数] ビューを使用して個々の変数を開き、それらのプロパティを表示して、シミュレーション中に使用するサンプルデータを追加します。マウスの右クリックメニューを使用して変数を開き、編集します。

開いた変数は、ステータスを示すアイコン、記号、色といったさまざまな表示で表されます。例えば、青いテキストで示された可変パートは、そのパートがコピー操作で使用されていることを示します。

[アウトライン] ビューを使用して、グローバル変数またはローカル変数を作成します。グローバル変数はプロセス全体で使用でき、ローカル変数はその変数が定義されているプロセススコープ内でのみ使用できます。

エラー処理と内部変数

フォールトハンドラからエラー処理変数を作成すると、エラー変数のアイコンは青と黄色の二色表示になります。これは、フォールト処理でのみ使用できる特別な変数です。

Receive、Reply、Invoke、またはユーザーアクティビティを作成する場合は、新しい変数を作成せずにデータを操作できます。パラメータと呼ばれる内部変数は、これらのアクティビティのいずれかに追加したデータマッピングを含めるために作成します。

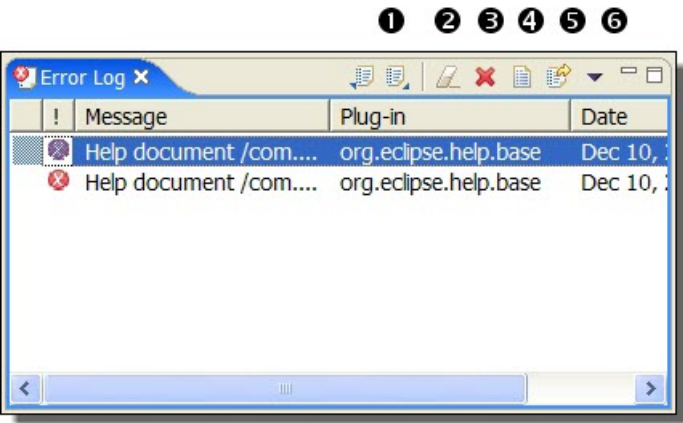
詳細については、[第 13 章, 「変数」 \(ページ 241\)](#)を参照してください。

エラーログ

エラーログには、システムエラーの詳細が表示されます。Informatica は、この情報をトラブルシューティングに使用します。

デフォルトのパースペクティブでは、エラーログは表示されません。エラーログを開くには、**[ウィンドウ] > [ビューの表示] > [エラーログ]** を選択します。

次の図は、エラーログの例を示しています。

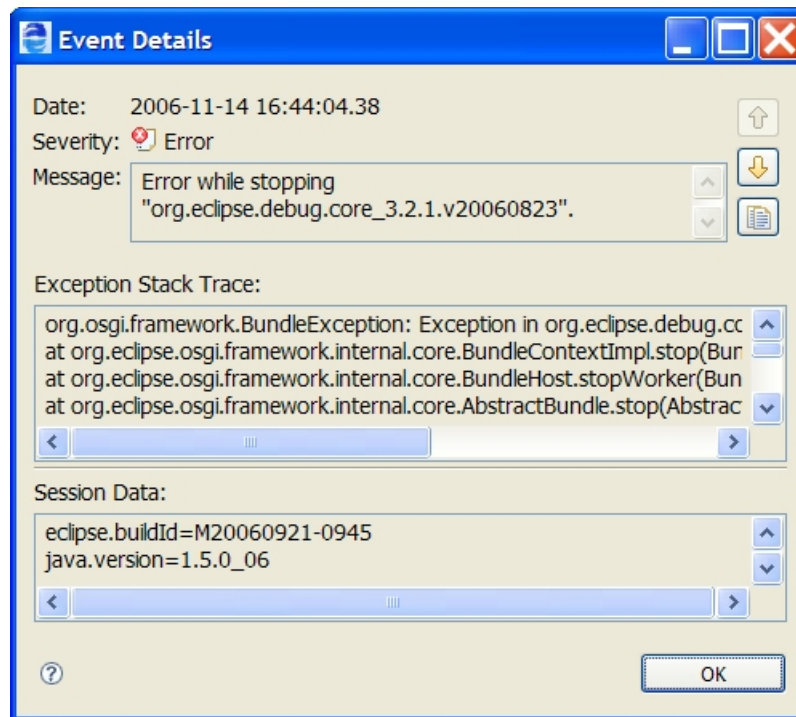


1	このログをテキストファイルにエクスポートします。古いエラーログをインポートします。
2	ログファイルの内容を削除せずに、エラーログビューをクリアします
3	ログファイルの内容を削除します
4	このログをテキストエディタで開きます
5	ログファイルの表示を復元します
6	イベントタイプやイベントの数を表示するためのフィルタ、およびその他の条件を選択します

エラーログを使用するためのヒント:

- ログとセッション情報を表示するには、エラーをダブルクリックします。
- ログとセッション情報をコピーするには、エラーを右クリックします。

次の図は、エラーのイベント詳細ビューを示しています。Informatica で問題のトラブルシューティングを行うために、このダイアログの情報をコピーするように求められる場合があります。

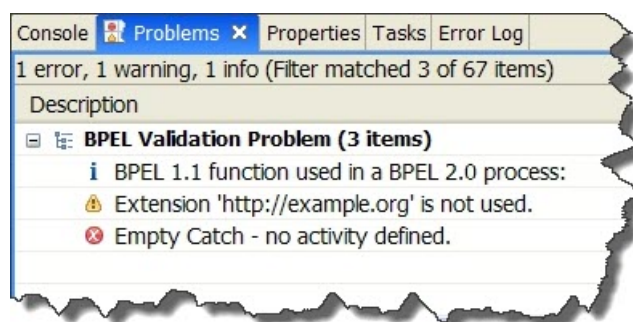


[問題] ビュー

[問題] ビューには、BPEL プロセスの検証に関連するエラー、警告、および情報が表示されます。ファイルを保存すると、Process Developer は BPEL ファイルに対して静的な分析を実行します。この分析によって、プロセスの BPEL 定義、式言語構文、および WSDL 参照が検証され、無効なアクティビティの項目が [問題] ビューに追加されます。プロセスをテストする前に、無効なアクティビティを修正することができます。

エラーおよび警告を修正すると、[問題] ビューの関連する項目が消えます。また、[問題] ビューに表示する情報のレベルを設定することもできます。

次の図は、Process Developer の BPEL プロセスに関する複数のエラーと 1 つの警告を示しています。



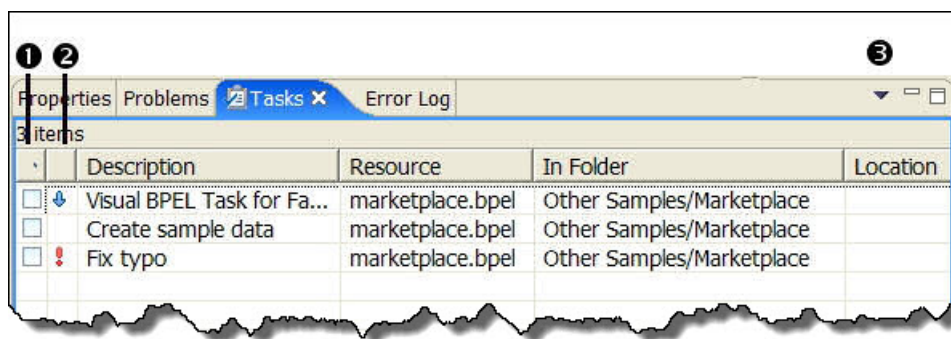
【問題】 ビューの使用に関するヒント:

- 関連するリソースに移動するには、項目をダブルクリックします。
- プロパティの詳細を表示したり、リソースに【移動】を選択したりするには、項目を右クリックします。
- タイプまたは重要度で問題を【グループ化】できます。
- 各種の【コンテンツの構成】機能を使用できます。

[タスク] ビュー

[タスク] ビューには、追加した実行予定の項目が表示されます。このビューから項目を追加するか、Process Editor キャンバス上のアクティビティを右クリックします。

次の図は、[タスク] ビューを開いた状態の例を示しています。



1	完了または未完了を示す、実行予定の項目のチェックボックス。項目の完了後に、チェックボックスをオンにします。カラムヘッダーをクリックすると、完了順に並べ替えることができます。
2	「高」、「通常」、「低」など、タスクを実行するための優先度。カラムヘッダーをクリックすると、優先度順に並べ替えることができます。
3	表示条件、並べ替え条件、およびフィルタ条件を選択します

【タスク】 ビューの使用に関するヒント:

- タスクをダブルクリックすると、関連するリソースに移動します。
- タスクを右クリックすると、プロパティの詳細を表示したり、リソースに【移動】を選択したりすることができます。

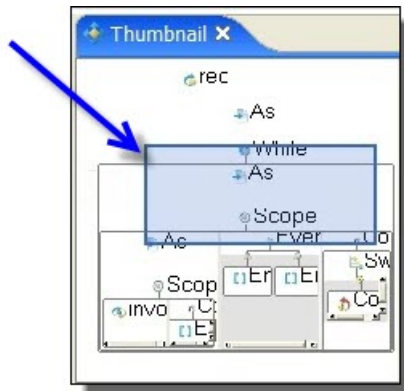
[サムネイル] ビュー

複雑なプロセスを設計する場合、表示するプロセスがキャンバスよりも大きくなってしまいます。[サムネイル] ビューを使用すると、大規模なプロセスでのナビゲーションを簡単に行えるようになります。

[サムネイル] ビューには、プロセスの縮小版が表示されます。Process Editor に現在表示されているプロセスのセクションは、青いボックスで網掛けされた状態になります。複雑なプロセスの全体を表示できない場合は、[サムネイル] ビューの青いボックスを移動して、選択した領域を表示します。

[サムネイル] ビューを表示するには、[ウィンドウ] > [ビューの表示] > [サムネイル] を選択します。

プロセスの別のセクションを表示するには、[サムネイル] ビューの網掛けボックスをクリックして、ボックスを移動します。キャンバス上のプロセスの対応するセクションが表示されます。次の画像は、この網掛けのボックスを示しています。



ブックマークビュー

【ブックマーク】ビューには、BPEL プロセスまたはサービスファイルに追加したブックマークのリストが表示されます。

BPEL アクティビティまたはサービスファイルにブックマークを追加すると、情報へ簡単にリンクバックできるようになります。リストからブックマークを選択することで、ブックマークを追加した場所に移動できます。

デフォルトのパースペクティブでは、【ブックマーク】ビューは表示されません。このビューを開くには、【ウィンドウ】>【ビューの表示】>【ブックマーク】を選択します。

次の図は、【ブックマーク】ビューの例を示しています。

Description	Resource	Path	Location
Link: [low risk]	loanProcess...	humanCompleted...	Link: [low risk]
Activity: [AssignApprove]	loanProcess...	humanCompleted...	Activity: [Assi...

ブックマークをダブルクリックすると、Process Editor キャンバス上のブックマークの場所が強調表示されます。ブックマークの詳細については、「ワークベンチユーザーガイド」のヘルプを参照してください。

【リレーション】ビュー

このビューを開くには、【ウィンドウ】>【ビューの表示】>【リレーションビュー】を選択します。

このビューには、エディタで開いているファイルに関連する BPEL、WSDL、XSD、およびその他のアーティファクトの依存関係と参照がグラフィカルに表示されます。このビューを使用して、ワークスペース内にある WSDL や XSD インポート、およびその他のアーティファクトの場所を追跡することができます。

例えば、BPEL ファイルが開いている場合は、WSDL、XSD、HTML などの BPEL ファイルが依存するファイルの関係図が表示されます。別のファイルを選択して、その依存関係と参照を確認することもできます。

ツールバーの【依存関係の表示】および【参照の表示】ボタンを使用して、ビューを切り替えることができます。

[サーバー] ビューと [コンソール] ビュー

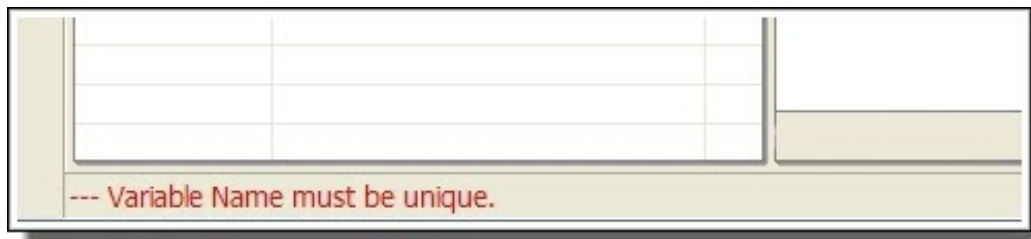
[サーバー] ビューでは、Process Developer インストールの一部であるプロセスサーバーを設定して起動できます。プロセスをサーバーにデプロイしてから実行し、リモートデバッグを行うことができます。

このビューで **[サーバーの起動]** を選択すると、コンソールに起動メッセージが表示されます。[コンソール] ビューには、プロセスサーバーが実行されている Tomcat サーブレットからのメッセージが表示されます。

組み込みエンジンの起動と使用に関する詳細については、[「組み込みプロセスサーバーのセットアップ」](#) (ページ 96) を参照してください。

ステータスバー

ステータスバーには、現在のアクションとエラーメッセージに関する情報が表示されます。次の図は、ステータスバーのエラーメッセージの例を示しています。



Process Developer パースペクティブのカスタマイズ

BPEL プロセスのさまざまな部分で作業する際に、一部のビューを非表示にしたり、他のビューを拡大したり、他のビューのタブの順序を変更したりする必要がある場合があります。Process Developer が構築されたワークベンチ環境は非常に柔軟であるため、これらの操作をすべて実行することができます。

詳細については、[「ウィンドウ、パースペクティブ、ビュー、およびエディタ」](#) (ページ 65) を参照してください。

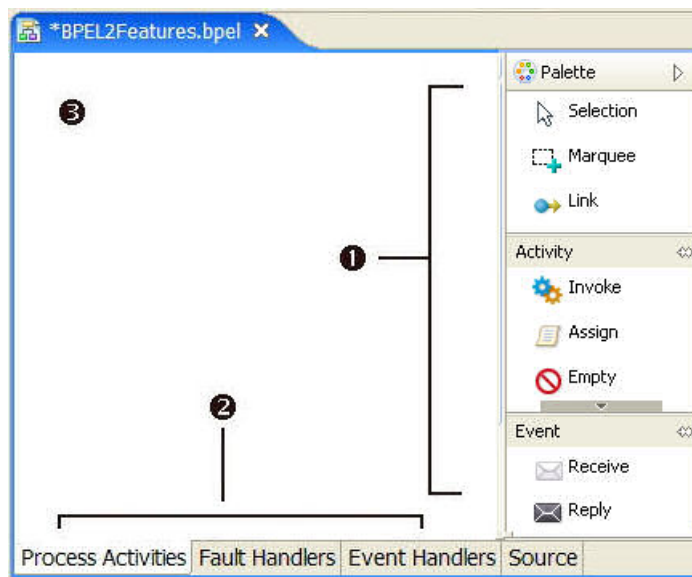
Process Developer の [デバッグ] パースペクティブ

詳細については、[第 28 章, 「シミュレーションとデバッグ」](#) (ページ 409) を参照してください。

Process Developer Process Editor の使用

Process Editor は、BPEL プロセスの視覚的な表現を作成するためのキャンバスとパレットで構成されています。[「新しいプロセスの作成」](#) (ページ 90) に記載されているように、BPEL ファイルを作成するとエディタの [プロセスアクティビティ] タブでファイルが開き、プロセスを作成するためのキャンバスが作成されます。

次の図は、プロセスエディタの各パートを示しています。



1	パレット 。構成を選択して、プロセス定義を作成します。図面ラベルと図形でプロセスに注釈を付けます。他のプロセスで再利用するカスタムアクティビティを作成します。
2	タブ 。タブを選択して、プロセスの特定のコンポーネントを独自のページで設計するか、生成された XML コードを [ソース] タブで表示します。各ページの内容は印刷することができます。
3	キャンバス 。パレット項目をキャンバスにドラッグします。項目とキャンバスの視覚的なプロパティを設定します。

詳細については、以下のトピックを参照してください。

- [「Process Editor の \[プロセスアクティビティ\] タブ」 \(ページ 74\)](#)
- [「Process Editor の \[フォールトハンドラ\] タブ」 \(ページ 75\)](#)
- [「Process Editor の \[イベントハンドラ\] タブ」 \(ページ 75\)](#)
- [「Process Editor の \[補償\] タブと \[終了ハンドラ\] タブ」 \(ページ 75\)](#)
- [「Process Editor の \[ソース\] タブ」 \(ページ 75\)](#)
- [「BPMN 中心のツールパレットと BPEL 中心のツールパレットの比較」 \(ページ 50\)](#)
- [「視覚的なプロパティの設定と独自のイメージライブラリの使用」 \(ページ 76\)](#)
- [「プロセスへのタスクとブックマークの追加」 \(ページ 78\)](#)
- [「プロセスへのコメントの追加」 \(ページ 79\)](#)
- [「プロセスへのドキュメントの追加」 \(ページ 79\)](#)
- [「Process Editor キャンバスで設計するためのヒント」 \(ページ 80\)](#)
- [「アクティビティの表示と非表示」 \(ページ 81\)](#)
- [「Process Editor のキーボードショートカット」 \(ページ 63\)](#)

Process Editor の [プロセスアクティビティ] タブ

プロジェクトエクスプローラからプロセスファイル (.bpel ファイル) を開くと、プロセスのグラフ表現が Process Editor の [プロセスアクティビティ] タブに表示されます。

このタブを使用する一般的なアクティビティは次のとおりです。

- [プロセスアクティビティ] タブを使用して、すべてのアクティビティ、リンク、構造などのプロセスを設計します。パレットからオブジェクトを選択して、プロセスを設計し、注釈を付けます。
- [フォールトハンドラ] タブと [イベントハンドラ] タブを使用して、プロセスレベルで例外イベントの処理を設計します。
- 任意の数の .bpel ファイルを開くことができます。Process Editor のタイトルバーでファイル名をクリックして、特定のプロセスを表示します。
- このページでグラフィック要素を印刷できます。

[「Process Editor キャンバスで設計するためのヒント」 \(ページ 80\)](#) も参照してください。

Process Editor の [フォールトハンドラ] タブ

Process Editor の [フォールトハンドラ] タブを使用して、プロセスレベルのフォールト処理を設計します。このタブでは、catch および catchAll コンテナとアクティビティを追加できます。これらのフォールトハンドラは、BPEL ファイルの <faultHandlers> セクションに追加されます。

このタブを使用する一般的なアクティビティは次のとおりです。

- フォールトハンドラへの図面ラベルの追加。
- このページのグラフィック要素の印刷。

Process Editor の [イベントハンドラ] タブ

Process Editor の [イベントハンドラ] タブを使用して、プロセスレベルのメッセージおよびタイムアウトイベントを設計します。詳細については、[第 23 章、「イベント処理」 \(ページ 355\)](#) を参照してください。

このタブを使用する一般的なアクティビティは次のとおりです。

- 図面ラベルをイベントハンドラに追加できます。
- このページでグラフィック要素を印刷できます。

Process Editor の [補償] タブと [終了ハンドラ] タブ

1 つの BPEL プロセスで別の BPEL プロセスを呼び出すことができます。この場合、呼び出されたプロセスはサブプロセスと呼ばれ、特別な方法で処理されます。サブプロセスは、プロセススコープと同様の方法による補償処理および終了処理の対象となります。詳細については、「[別の BPEL プロセスのサービスとしての BPEL プロセスの作成](#)」を参照してください。

これらのタブは次のとおりです。

- **[補償ハンドラ] タブ。** プロセスのインスタンスの補償を有効にする場合は、このタブを使用します。詳細については、「[プロセス要素とプロパティ](#)」を参照してください。
- **[終了ハンドラ] タブ。** プロセスの終了処理を有効にする場合は、このタブを使用します。詳細については、「[プロセス要素とプロパティ](#)」を参照してください。

Process Editor の [ソース] タブ

Process Editor の [ソース] タブを使用して、プロセス用に生成された XML コードを表示します。表示されるコードは、WS-BPEL 2.0 仕様に従って検証されています。詳細については、「[BPEL XML ソースと暗黙的に追加されたアクティビティ](#)」 (ページ 132) を参照してください。

視覚的なプロパティの設定と独自のイメージライブラリの使用

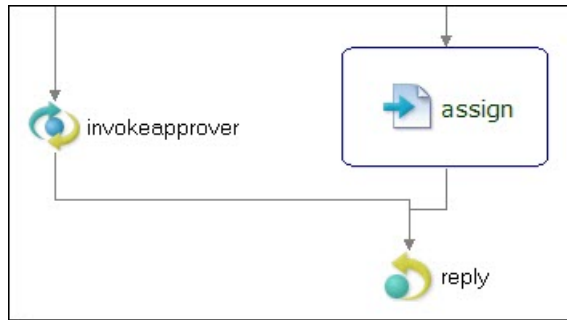
Process Developer のすべてのアクティビティアイコンは、フォントサイズ、色、境界線、およびその他の視覚的なプロパティを変更できます。

さらに、アクティビティアイコンを変更して、構築中のプロセスのタイプによりわかりやすい視覚的表現を加えることができます。

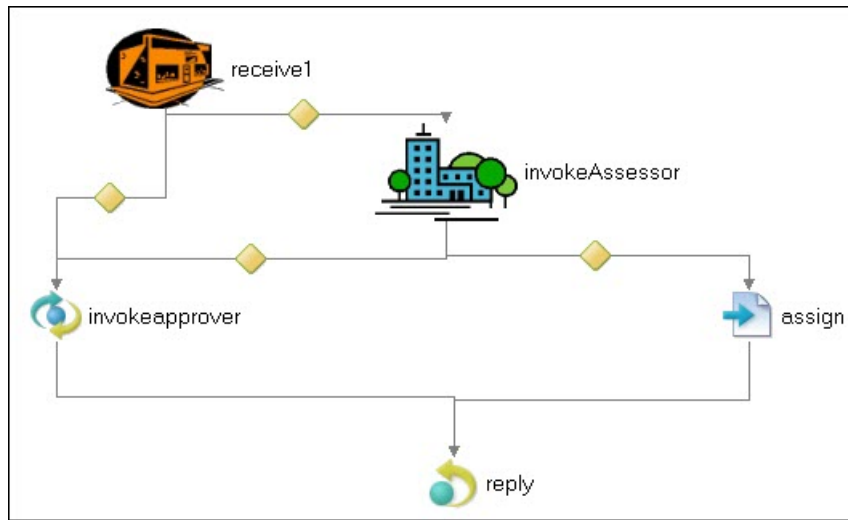
[「レイアウトの設定」 \(ページ 43\)](#)に記載されているように、すべてのプロセスにデフォルトを設定することや、個々のプロセスおよびアクティビティの視覚的なプロパティを変更することもできます。

一部の例を以下に示します。

例 1: 境界線を追加し、アクティビティのフォントサイズを変更する

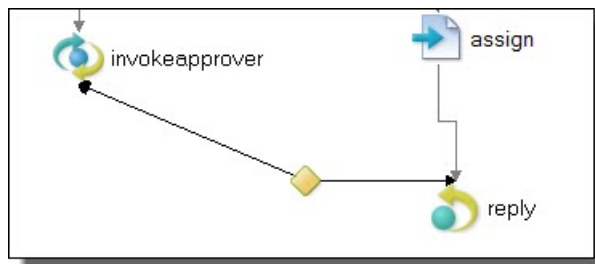


例 2: アクティビティごとにイメージの場所プロパティを設定して、アクティビティ用に独自のイメージを選択する



例 3: リンクルーターのスタイルを [マンハッタン] から [手動] に変更する

手動スタイルでは、階段スタイルのリンクではなく直線のリンクが描画され、リンク遷移の角度を設定できます。最短経路スタイルのリンクは、オブジェクトの境界を移動して 2 つのアクティビティをリンクします。



次の表に、さまざまなアクティビティに設定できる視覚的なプロパティを示します。プロパティの中にはすべてのアクティビティに適用されないものもあります。

プロパティ	説明
背景色	透明プロパティが [いいえ] の場合、色を塗りつぶします
枠線	図面アイテムに境界線を表示するための [はい] / [いいえ] の選択
境界の色	図面アイテムの境界線の基本色またはカスタム色を選択するためのダイアログを開きます。
拡張	スコープなどのコンテナの場合、コンテナの内容が表示されているか非表示になっているかを示す [はい] / [いいえ] が表示されます。この設定は、コンテナの右クリックメニューから [展開] または [折りたたみ] を選択することで変更できます。
フォント	ラベルに使用するフォントを設定します。
フォント色	ラベルの色を設定します。
イメージの表示	描画するイメージ（存在する場合）
イメージの場所	イメージファイルのファイルシステムの場所と名前を設定します。イメージを変更するための複数選択アクティビティについては、以下のヒントのセクションを参照してください。
イメージのテキストギャップ	アクティビティアイコンとラベル間のギャップのサイズを選択します。
ラベル	アクティビティラベルの表示形式を選択します。 「アクティビティラベルの選択」 （ページ 158）を参照してください。
ラベルの整列	テキストの位置揃えのプロパティを設定します: 左、右、中央、上、下
ラベルの配置	描画オブジェクトに対して、イメージに対するテキストラベルの配置を設定します。
場所	オブジェクトのキャンバス上の XY 位置を設定します
印刷方向	コンテナに対して、シーケンス内のアクティビティの縦方向または横方向を設定します。
一次アライメント	シーケンスアクティビティに対して、アクティビティのグループの配置を設定します。
二次アライメント	シーケンスアクティビティに対して、シーケンス内のアクティビティの配置位置を設定します。

プロパティ	説明
サイズ	キャンパスの現在のサイズを設定します
サイズの自動調整	選択した図面オブジェクトのサイズ変更を制限する場合は、[はい] に設定します。図面オブジェクトのサイズ変更を許可するには、[いいえ] を選択します。
テキスト	図面オブジェクトに対するテキストラベル
透明度レベル	描画オブジェクトに対して、透明度の値を設定します。値を大きくすると透明度が低下します。
トランスペアレント	[はい] / [いいえ] を設定。[いいえ] を選択すると、背景色を設定できます。

視覚的なプロパティを設定するためのヒント:

- 視覚的なプロパティファイルにデフォルト値を設定します。詳細については、[「レイアウトの設定」 \(ページ 43\)](#)を参照してください。
- 特定のプロセスに対するアクティビティのグループのイメージを変更するには、アクティビティを複数選択してから、イメージの場所を選択します。例えば、すべての Receive を選択して、そのアクティビティタイプのアイコンを変更します。
- If アクティビティなどのコンテナアクティビティのサイズを変更するには、[サイズの自動調整] オプションを [いいえ] に設定します。

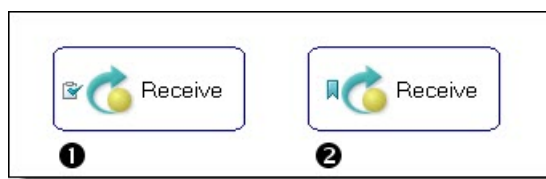
プロセスへのタスクとブックマークの追加

Process Editor キャンパスのグラフィック要素にブックマークを追加してリンクを作成することで、プロセスのその場所に簡単に戻ることができます。また、ブックマークに似た働きを持つ、タスクを追加することもできます。タスクを追加するには、グラフィック要素を強調表示し、ジャンプしてその場所まで簡単に戻るためのリンクを作成します。

タスクとブックマークは、リンクの目的が異なります。タスクは通常、その場所に単純にリンクするのではなく、その場所で実行する必要があることを示します。また、タスクには、重要度に応じてタスクを簡単に並べ替えることができる優先順位があります。

グラフィカル要素にタスクまたはブックマークを追加するには、要素を選択して右クリックします。[ブックマークの追加] または [タスクの追加] を選択します。タスクまたはブックマークに名前を付けます。

次の図は、タスクまたはブックマークを追加したグラフィック要素を示しています。



1	タスクを追加した Receive アクティビティ
2	ブックマークを追加した Receive アクティビティ

詳細については、[「\[タスク\] ビュー」 \(ページ 71\)](#)を参照してください。

プロセスへのコメントの追加

Process Developer では、プロセスに注釈を追加する場合にコメントとドキュメントという 2 つの方法を使用できます。

コメントは、タスクや実行する項目を追加することができるため、設計時の注釈として使用すると便利です。これらは、BPEL ソースコードで HTML 形式のコメントとして表示されます。ドキュメント要素を BPEL プロセス、アクティビティ、およびリンクに追加することもできます。詳細については、[「プロセスへのドキュメントの追加」 \(ページ 79\)](#)を参照してください。

アクティビティ、リンク、変数、プロセス自体など、プロセスの任意の要素にコメントを追加できます。

1. [アウトライン] ビューまたは Process Editor キャンバスから項目を選択します。例:
 - [アウトライン] ビューで、プロセスまたはパートナーリンク、変数、あるいはコピー操作を選択します
 - Process Editor キャンバスでアクティビティまたはリンクを選択します
2. [プロパティ] ビューの [コメント] 行の最後にある **[ダイアログ]** をクリックします。
3. コメントダイアログにコメントを入力し、**[OK]** をクリックします。
ヒント: コメントの先頭にタスクタグを追加すると、コメントが [タスク] ビューに表示されます。詳細については、[「タスクと問題の設定」 \(ページ 46\)](#)を参照してください。

図のように、Process Editor キャンバス上でカーソルを上にとくとアクティビティとリンクのコメントが表示されます。



コメントは、フォーマット済みのタグとしてプロセスの [ソース] ビューに表示されます。

名前空間に関するコメントを追加する場合は、プロセス自体にコメントを追加します。

プロセスへのドキュメントの追加

プロセスまたはアクティビティには、1 つ以上のドキュメント項目を追加できます。言語サポートが必要な場合は、言語識別子を追加します。ソースフィールドは、アプリケーション固有の参照用のフィールドです。

Process Developer では、プロセスに注釈を追加する場合にコメントとドキュメントという 2 つの方法を使用できます。<documentation>要素を BPEL プロセス、アクティビティ、およびリンクに追加することができます。タスクや実行する項目を追加することができるため、設計時にはコメントを使用すると便利です。詳細については、[「プロセスへのコメントの追加」 \(ページ 79\)](#)を参照してください。

アクティビティ、リンク、変数、プロセス自体など、プロセスの任意の要素にドキュメントを追加できます。

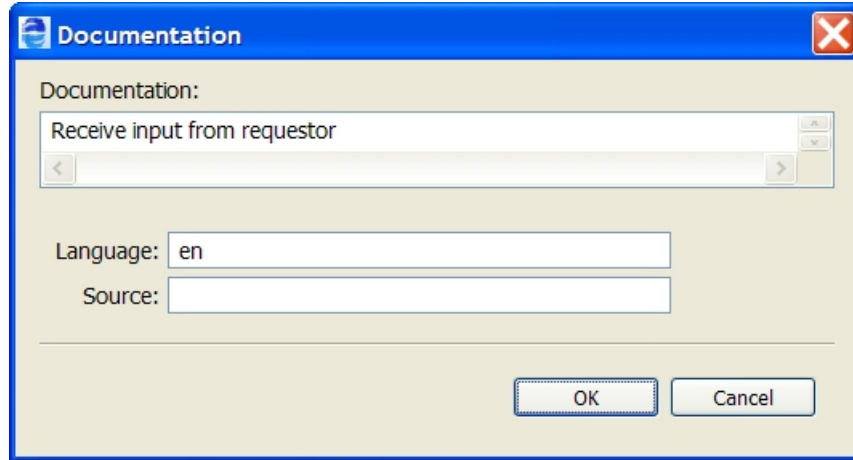
プロセスレポートを生成する場合、ドキュメントを使用すると非常に便利です。詳細については、[「プロセスレポートの生成」 \(ページ 62\)](#)を参照してください。

1. [アウトライン] ビューまたは Process Editor キャンバスから項目を選択します。例:
 - [アウトライン] ビューで、プロセスまたはパートナーリンク、変数、あるいはコピー操作を選択します

- *Process Editor* キャンバスでアクティビティまたはリンクを選択します
2. [プロパティ] ビューで、次のいずれかを実行します。
 - [ドキュメント] タブにテキストを直接入力します。
 - [すべて] ビューの [ドキュメント] 行の最後にある **【ダイアログ (...)]** をクリックします。以下に示すようにダイアログに入力します。

ドキュメントの詳細を入力する手順

1. 例に示すように、**【新規】** を選択して **【ドキュメントの詳細】** ダイアログを開きます。



2. 次のようにダイアログに入力します。
 - **ドキュメント**。プレーンテキスト、HTML、XHTML など、選択した形式でコンテンツを入力します。
 - **言語** (オプション)。コンテンツが記述されている自然言語または形式言語の言語識別子を入力します。言語識別子の参照に関する詳細については、XML 1.0 の仕様を参照してください。
 - **ソース** (オプション)。この属性は、アプリケーション情報のソースを URI 形式で示します (属性は XSD スキーマで記述されており、通常はドキュメント要素に含まれます)。
3. **【OK】** をクリックし、**【新規】** を選択してドキュメント要素を追加します。
4. [ドキュメントエントリ] リストで、**【上】** ボタンまたは **【下】** ボタンを選択してエントリを再編成します。リストの最初のエントリが、アクティビティ、リンク、またはパートナーリンクの [プロパティ] ビューに表示されます。

ドキュメントは、フォーマット済みのタグとしてプロセスの [ソース] ビューおよび生成可能なレポートに表示されます。

Process Editor キャンバスで設計するためのヒント

Process Editor キャンバスを使用して、BPEL プロセスの視覚的な表現を作成します。

プロセスを設計するためのヒントを次に示します。

- Process Editor のタイトルバーをダブルクリックすると、キャンバス全体が表示されます。
- アクティビティとリンクのフォントサイズと色、境界線、イメージ、その他のプロパティを変更します。[「視覚的なプロパティの設定と独自のイメージライブラリの使用」 \(ページ 76\)](#) を参照してください。
- **【自動レイアウト】** ツールバーボタンを使用して、[「レイアウトの設定」 \(ページ 43\)](#) の設定に従ってプロセスアクティビティを整列します。任意のコンテナまたは選択範囲のマウスの右クリックメニューにある **【自動レイアウト】** オプションを使用します。
- プロセスの方向を変更するには、**【縦にする】** および **【横にする】** を使用します。

- アクティビティまたはコンテナを移動すると、オブジェクトはグリッドに自動整列されます。Alt キーとマウスボタンを使用して、オブジェクトを一度に 1 ピクセルずつ移動します。
- キャンバスの倍率を変更するには、[プロセス] メニューの [ズームイン] と [ズームアウト] を使用します。
- ツールバーの [ズーム] ドロップダウンから [ズーム] または [ページに合わせる] を選択します。
- コンテナの [展開] または [折りたたみ] を使用して、コンテナを表示または非表示にします。折りたたまれたコンテナにドリルダウンして、その内容を一時的に表示します。[「アクティビティの表示と非表示」\(ページ 81\)](#)を参照してください。
- 2 つ以上のアクティビティを選択し、[リンク] ツールバーボタンを使用してアクティビティをリンクします。
- プロセスの規模が大きく複雑な場合は、[「\[サムネイル\] ビュー」\(ページ 71\)](#)を使用してプロセスの各部分を表示します。
- 2 つ以上のグラフィック要素を選択し、ツールバーの [リストの整列] を使用してグラフィック要素を整列します。
- 図面ラベルとアクティビティを選択し、マウスの右クリックメニューから [アンカーの注釈] を選択して、図面ラベルをアクティビティにアンカーします。

アクティビティの表示と非表示

Process Editor の領域は、次の方法で保存できます。

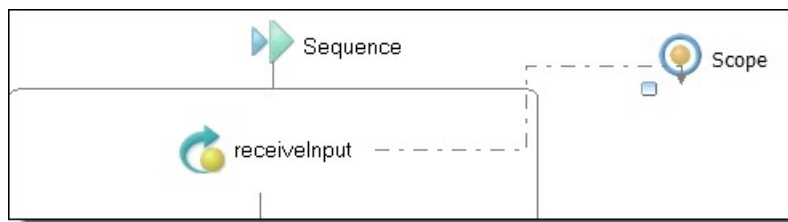
- コンテナアクティビティの展開と折りたたみ
- 折りたたまれたコンテナへのドリルダウンによる一時的な内容の確認

コンテナアクティビティの展開、折りたたみ、およびコンテナアクティビティへのドリルダウン

コンテナの表示を折りたたんで、Flow アクティビティを制御できます。これらのアクティビティには、Sequence と Flow (Process Developer Classic のみ)、Fork Join (BPMN のみ)、If、While、Pick、Scope、repeat Until、および For Each が含まれます。

アクティビティが折りたたまれている場合は、+アイコンを選択して一時的にアクティビティにドリルダウンし、その内容を表示します。ブレードक्रम表示または [ナビゲート] > [戻る] コマンドを使用して、アクティビティを折りたたみます。

Scope などのアクティビティを折りたたむには、右マウスメニューから [コンテナの折りたたむ] (または [制御フロー]) を選択します。下の折りたたまれたスコープに示されている小さなアイコンは、コンテナが折りたたまれていることを示します。コンテナを展開するには、マウスの右メニューから [コンテナの展開] (または制御フロー) を選択します。



展開、折りたたみ、およびドリルダウンの使用に関するヒント:

- アクティビティが折りたたまれている場合は、+アイコンを選択してコンテンツにドリルダウンします。または、マウスの右メニューから [アクティビティに移動] (Ctrl + Alt + L) を選択します。
- 上記の例に示されるように、デフォルトでは、折りたたまれたコンテナ内に 1 つ以上のターゲットがあるリンクのリンクスタイルが点線になります。

- [プロセス] メニューまたは Process Editor の空白の領域から、**[すべてのコンテナを展開]** して **[すべてのコンテナを折りたたむ]** ことができます。Process Editor から、右マウスを使用してこれらのオプションを選択します。
- 設定を変更して、他のアクティビティの位置に対する、展開されたコンテナと折りたたまれたコンテナの位置を制御することもできます。この設定は、プロセス設計の全体的な外観に影響します。詳細については、[「レイアウトの設定」 \(ページ 43\)](#)を参照してください。
- 結果が適切でない場合は、展開/折りたたみの移動を **[元に戻す]** ことができます

第 7 章

ワークスペース、オーケストレーション、プロセス

この一連のトピックでは、Process Developer、ワークスペース、オーケストレーション、およびプロセスの起動に関する情報を示します。

Eclipse 環境の使用

Process Developer は、Eclipse 統合開発環境に対する一連のプラグインです。Eclipse は、ツールプロバイダのオープンコミュニティによって構築された、ツール統合のためのオープンプラットフォームです。Process Developer は、Eclipse Workbench の機能を利用して、BPEL の構築およびオーケストレーション機能を提供します。

Process Developer は、次の方法でインストールできます。

- Process Developer をスタンドアロンアプリケーションとしてインストールする
インストールアーカイブには、Process Developer が Eclipse アプリケーションとして抽出されます。Process Developer の実行可能ファイルを起動して、アプリケーションを起動します。
- Process Developer プラグインを Eclipse 環境にインストールする
ダウンロードしたインストールアーカイブで、Process Developer_plugins.zip（または Process Developer_plugins.tar.gz）を使用して必要な機能とプラグインを抽出し、Eclipse インストールにコピーします。Eclipse で、Process Developer パースペクティブを開きます。

インストールの詳細については、Process Developer インストールファイルを抽出したフォルダのルートにある readme ファイルを参照してください。

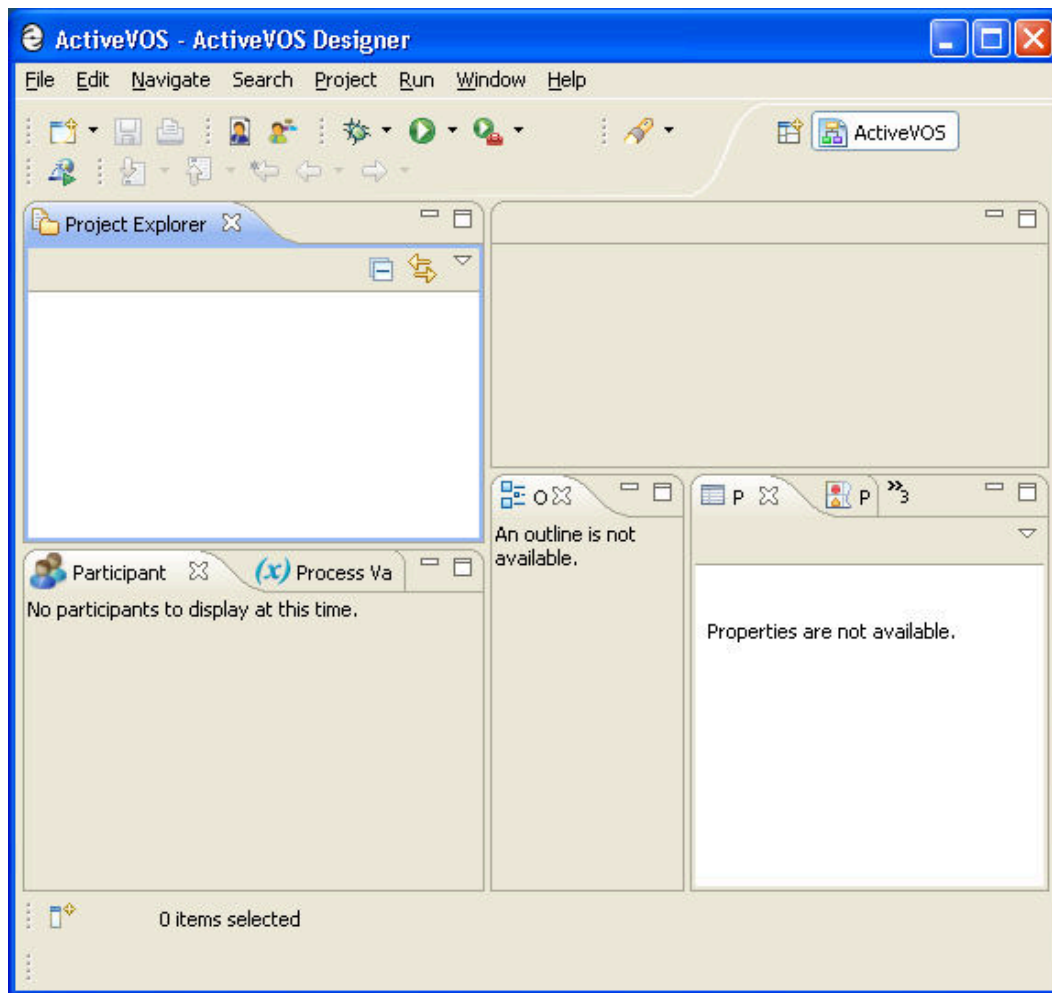
Eclipse コミュニティの詳細については、<http://www.eclipse.org> を参照してください。

Process Developer の起動

Process Developer を起動するには、インストール中に追加されるショートカットボタンを使用します。または、Process Developer インストールフォルダを開き、Process Developer フォルダで development.exe (Windows)、または designer (Linux) を見つけます。

Process Developer を Eclipse プラグインとしてインストールした場合は、Eclipse を起動し、[ウィンドウ] メニューから **[Open Perspective] > [Process Developer]** を選択します。

Process Developer を初めて起動した場合、ワークスペースは次のようになります。



開始するには、Tutorial Orchestration プロジェクトまたは新しい Orchestration プロジェクトを作成するか、既存のプロジェクトをインポートします。デフォルトの場所であるワークスペースフォルダに新しいプロジェクトを作成することをお勧めします。

詳細については、「[Orchestration プロジェクトの作成](#)」(ページ 85)および「[ワークスペースを使用したプロジェクトの保存](#)」(ページ 84)を参照してください。

ワークスペースを使用したプロジェクトの保存

ワークスペースは、すべてのプロジェクトファイルや、カスタマイズした設定などのメタデータを含むディスク上のフォルダです。インストール中に、ワークスペースを作成する場所を指定する必要があります。

作成したすべてのプロジェクトとファイルがワークスペースフォルダに追加され、Process Developer の「プロジェクトエクスプローラ」ビューに表示されます。Process Developer の外部でファイルに変更を加えると、その変更はプロジェクトエクスプローラに反映されます。

詳細については、以下を参照してください。

- 「[Orchestration プロジェクトの作成](#)」(ページ 85)
- 「[既存の BPEL プロセスのインポート](#)」(ページ 94)

Orchestration プロジェクトの作成

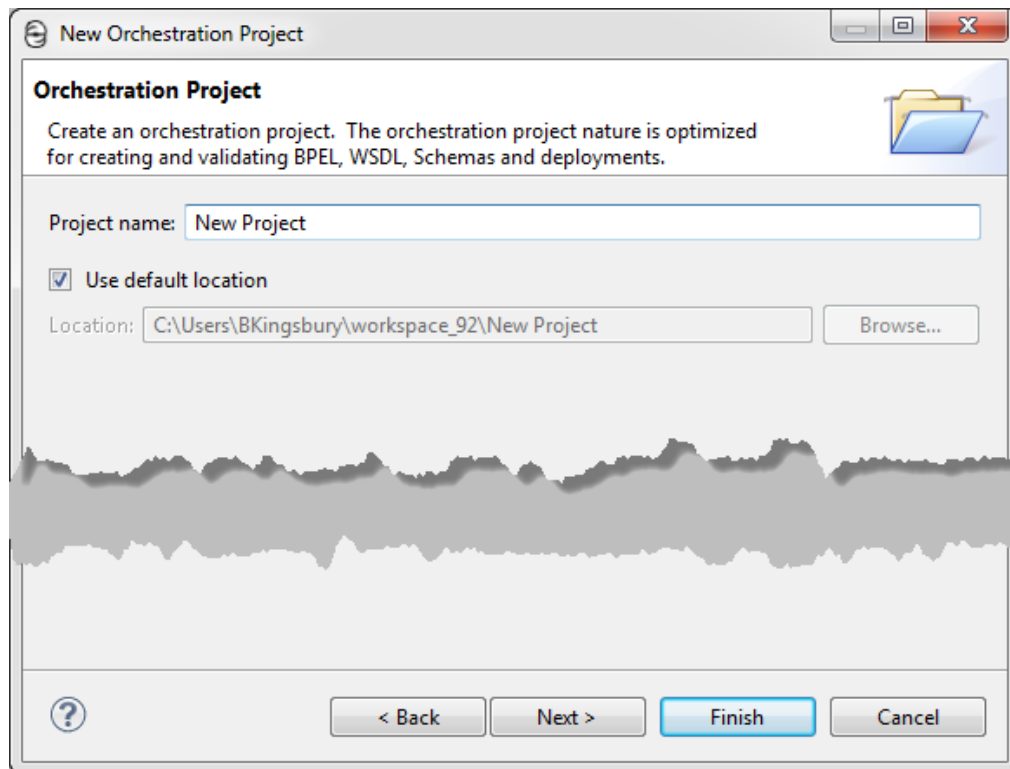
BPEL プロセスおよび関連ファイルを格納するプロジェクトフォルダを作成する必要があります。プロジェクトは、ファイルシステム内のフォルダにマッピングされます。デフォルトの Orchestration プロジェクトには、BPEL オーケストレーションをサポートする次のものが含まれます。

- **サービス参照**。詳細については、[「インタフェース、サービス参照、ローカル WSDL について」](#) (ページ 99)を参照してください。
- **フォルダ**。オーケストレーションをサポートする、整理された便利なリソースのレジストリとして、複数のフォルダが追加されています。
- **ビルダ**。詳細については、[「プロジェクトのオーケストレーションおよび検証ビルダについて」](#) (ページ 88)を参照してください。

さらに、プロジェクトの開始に役立つ複数のテンプレートが用意されています。[「Orchestration プロジェクトのテンプレート」](#) (ページ 86)を参照してください。

新しい Orchestration プロジェクトを作成する手順

1. **[ファイル] > [新規] > [Orchestration プロジェクト]** を選択し、**[次へ]** をクリックします。



2. プロジェクト名を入力します。
3. 次のいずれかを実行します。
 - **[完了]** をクリックして新しいプロジェクトを作成します。
 - 必要に応じて、**[デフォルトの場所を使用]** チェックボックスをオフにします。詳細については、[「ワークスペースプロジェクトのデフォルトの場所の変更」](#) (ページ 86)を参照してください。
 - **[次へ]** をクリックして、特別なタイプの Orchestration プロジェクトのテンプレートを選択します。詳細については、[「Orchestration プロジェクトのテンプレート」](#) (ページ 86)を参照してください。

4. デフォルトでは、プロセスファイルおよび関連のあるリソースを保存するための複数のフォルダを作成できることに注意してください。これらのフォルダは、Orchestration プロジェクトの性質に応じて提供されます。
5. 新しいプロジェクトがプロジェクトエクスプローラに表示され、ファイルシステムのフォルダに *.project* ファイルが含まれます。

[「新しいプロセスの作成」 \(ページ 90\)](#)に記載されているように、新しい BPEL プロセスファイルを作成できるようになりました。

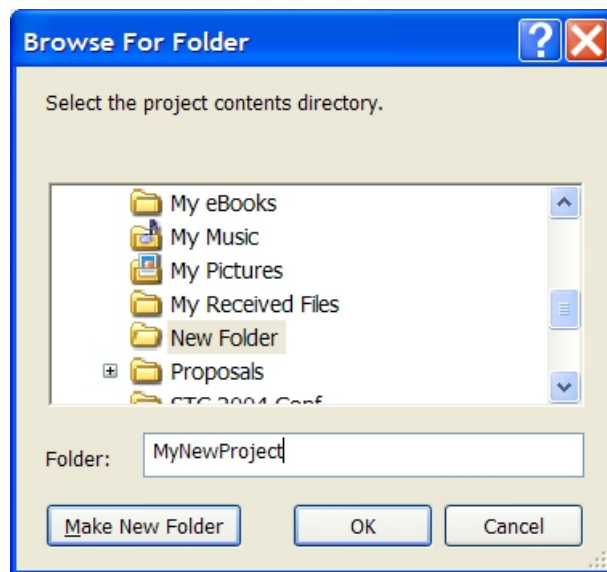
[「プロジェクト Orchestration ネーチャーの追加または削除」 \(ページ 88\)](#)も参照してください。

ワークスペースプロジェクトのデフォルトの場所の変更

新規 Orchestration プロジェクトを作成する場合、[デフォルトの場所を使用] の横にあるチェックボックスの選択を解除し、ファイルシステムの場所を参照してプロジェクトを作成できます。

デフォルト以外の場所を選択する場合:

- 選択したファイルシステムフォルダにサブフォルダが含まれる場合、すべてのサブフォルダがプロジェクトの一部になります。空のプロジェクトが必要な場合は、図に示すように、必ず **「新しいフォルダを作成」** を選択してフォルダ名を入力してください。

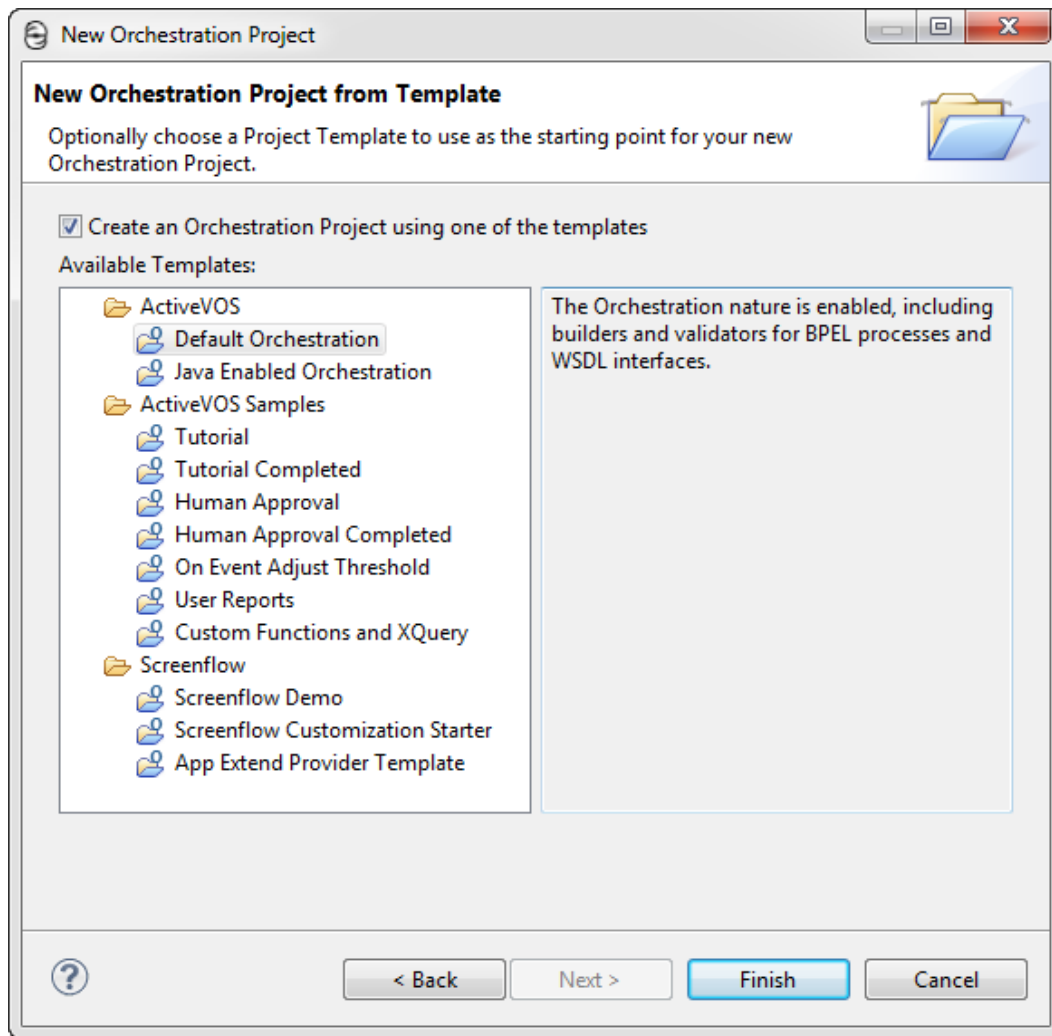


- ネストされたプロジェクトを作成することはできません。プロジェクトは重複しないようにしてください。

Orchestration プロジェクトのテンプレート

Orchestration プロジェクトのテンプレートには、プロジェクトの開始ポイントを選択するための組み込みのリソースとネーチャーが用意されています。

[ファイル] > [新規] > [Orchestration プロジェクト] を選択して新しいプロジェクトに名前を付け、**[次へ]** をクリックして特別な Orchestration プロジェクトテンプレートを選択します。



テンプレートには、特別なプロジェクトを開始する場合に役立つファイル、リソース、および Orchestration ネーチャー（詳細については「[プロジェクト Orchestration ネーチャーの追加または削除](#)」（ページ 88）を参照）が含まれています。例:

- **Java 対応 Orchestration。** 詳細については、「[Java インタフェースの作成](#)」（ページ 103）を参照してください。
- **チュートリアル。** チュートリアルには、WSDL とサンプルが含まれています。これは、最初の BPEL プロセスを作成するための開始ポイントです。
- **完全なチュートリアル。** 完全な BPEL プロセスを含む、シミュレーションとデプロイメント用のすべての関連リソース

他のプロジェクトテンプレートは、Process Developer の使用方法を確認する場合に役立ちます。

プロジェクト Orchestration ネーチャーの追加または削除

プロジェクトエクスプローラーでプロジェクトをインポートまたは作成すると、プロジェクト *Orchestration* ネーチャーを追加できます。Orchestration ネーチャーによって、オーケストレーションの作成と検証が強化されます。

Orchestration ネーチャーにより、次のことができます。

- サービス参照をインポートし、WSDL ポートタイプ、パートナーリンクタイプ、メッセージ、およびサンプルデータを表示します。詳細については、「[サービス参照のインポート](#)」(ページ 101)を参照してください。
- 必要なすべてのリソースに対して、自動的に作成されたプロジェクトフォルダを使用します。
- 「[プロジェクトのオーケストレーションおよび検証ビルダについて](#)」(ページ 88)に記載されているように、ファイルを検証します。

Orchestration ネーチャーを追加するには、プロジェクト上でマウスを右クリックし、**[Orchestration ネーチャーの追加]** を選択します。

Orchestration ネーチャーがすでに存在している場合は、**[Orchestration ネーチャーの削除]** を選択できます。プロジェクトが Eclipse プロジェクトに変換されます。

プロジェクトのオーケストレーションおよび検証ビルダについて

BPEL プロセスを追加および変更すると、検証ビルダがバックグラウンドで働き、プロセス、インポートされた WSDL、およびその他のリソース内のエラーと問題を確実に特定します。エラー、警告、および情報は、**[問題]** タブに表示されます。

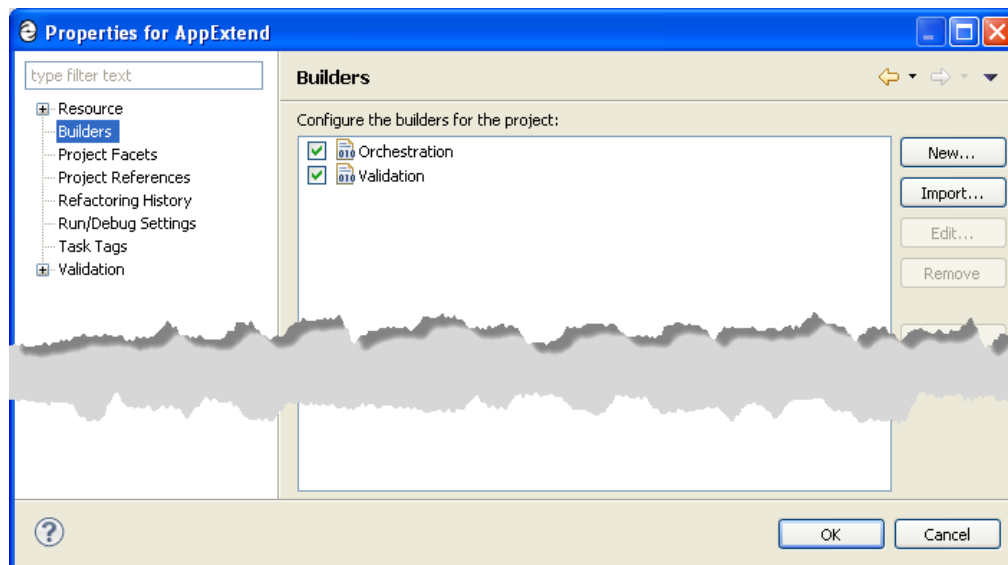
それぞれの Orchestration プロジェクトには、2 つのビルダがあります。

- **[オーケストレーションビルダ]** は、BPEL、B ユニット、PDD、および PDEF ファイルの検証を行います。また、このビルダは、ファイル間の依存関係を追跡します。例えば、WSDL ファイルが変更された場合、その WSDL ファイルをインポートしている BPEL ファイルが自動的に検証されます。
- **[検証ビルダ]** は、Process Developer に付属する Eclipse Web Tools Project の一部です。このビルダは、WSDL およびスキーマファイルを検証し、WS-I 準拠をチェックします。

Orchestration プロジェクトのビルダを無効または有効にする手順

1. プロジェクトエクスプローラーで、Orchestration プロジェクトを右クリックし、**[プロパティ]** を選択します。

2. [プロパティ] ダイアログで、[ビルダ] を選択します。



「オーケストレーション」ビルダと「検証」ビルダが有効になっていることに注意してください。これらは必要に応じて無効にすることができます。

コンピュータまたはネットワークの速度に応じて、WSDL キャッシュとタイムアウト値を設定して、構築が高速または低速で実行されるようにすることができます。[「キャッシュとタイムアウトの設定」 \(ページ 41\)](#)を参照してください。

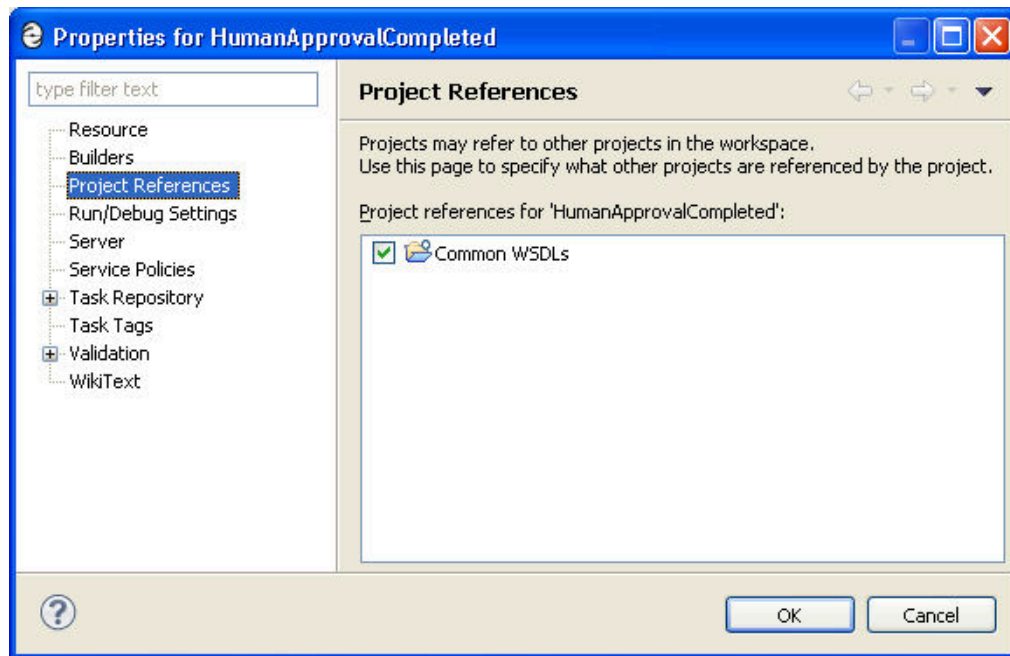
プロジェクト参照の使用

プロジェクトを作成する場合に、共通の WSDL ファイルとスキーマファイルを別のワークスペースプロジェクトに保持できます。この方法を使用すると、一般的に使用される WSDL を個別に保守およびデプロイできます。これにより、共通のリソースを最大限に再利用することができます。

複数のプロセス間でさまざまなリソースを共有できます。最も効率的な方法としては、それらを分離して、1 つ以上のリソースのみのプロジェクトに配置することが挙げられます。これらは、独自のコントリビューションとして別々にデプロイできます。プロセスプロジェクトでは、これらのリソースプロジェクトへの依存関係を宣言でき、依存するリソースのみのプロジェクトの後にデプロイします。

プロジェクト参照を使用することで、現在のプロジェクトの依存関係として別のプロジェクトを定義できます。

1. 新しい BPEL プロセスを開始する前に、共通の WSDL とスキーマを格納するための新しい Orchestration プロジェクトを作成します。
2. プロジェクトエクスプローラで BPEL ファイルを含むプロジェクトのプロジェクト名を右クリックし、**[プロパティ]** を選択します。
3. **[プロジェクト参照]** を選択します。



4. 参照するリソースを含むプロジェクトをリストから選択します。この例は、「CommonWSDL」という名前のプロジェクトを示しています。
 5. 新たなパーティシパントを作成すると WSDL が使用可能になり、[プロジェクト参照] フォルダに表示されます。詳細については、「[パーティシパント] ビューの使用」を参照してください。
- また、「プロジェクトの依存関係のデプロイと除外された依存関係の表示」も参照してください。

新しいプロセスの作成

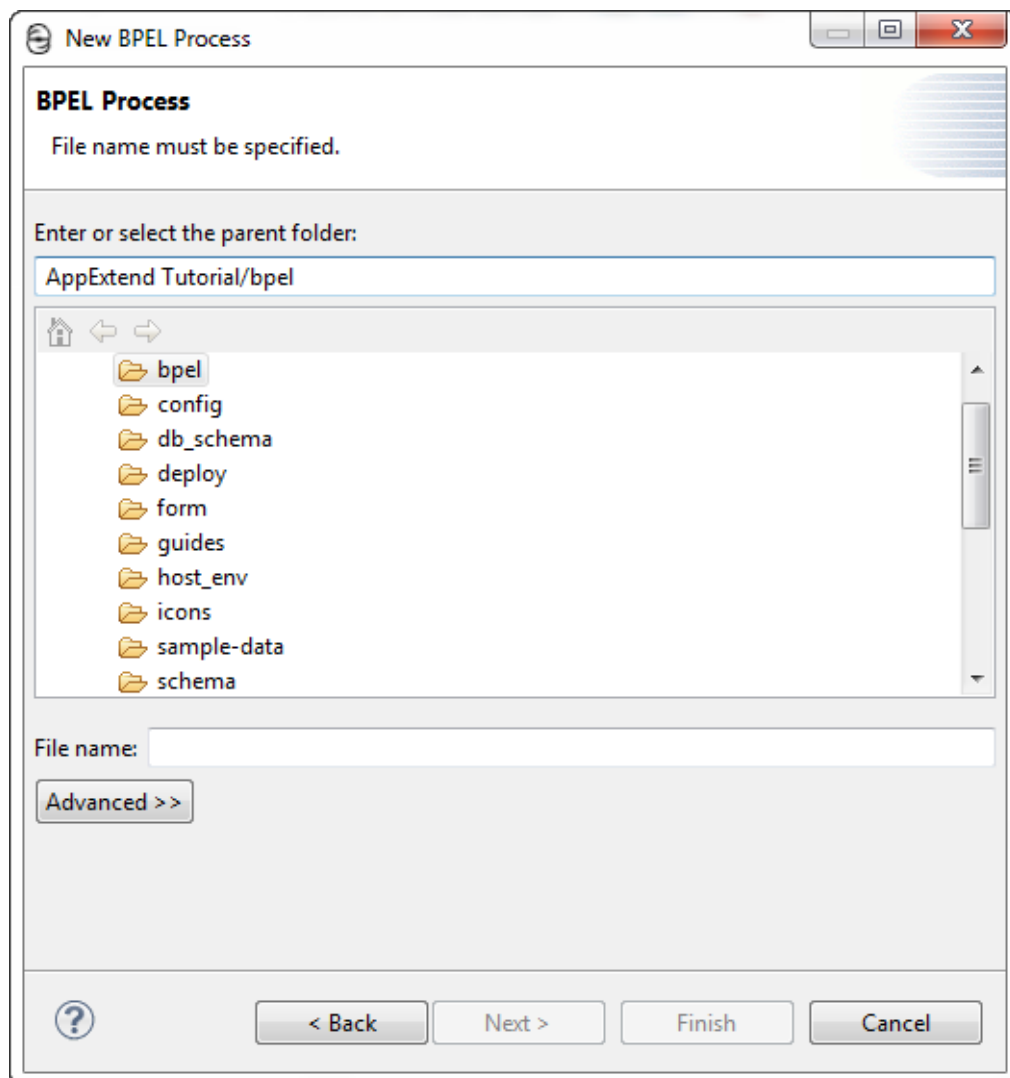
新しいプロセスを作成する際に、BPEL ファイルをワークスペースプロジェクトに追加します。

新しい BPEL プロセスを作成する前に、少なくとも 1 つの Orchestration プロジェクトがプロジェクトエクスプローラに表示されていることを確認してください。詳細については、[「Orchestration プロジェクトの作成」\(ページ 85\)](#)を参照してください。

新しいプロセスの開始

次の手順は、新しいプロセスを開始する方法を示しています。

1. [ファイル] > [新規] > [BPEL プロセス] を選択します。
2. より簡単な方法として、Orchestration プロジェクトフォルダで bpe1 フォルダを選択し、[ファイル名] フィールドに BPEL ファイルの名前を入力します。.bpe1 という拡張子が自動的に追加されます。



3. このプロセスに設定可能なプロパティを表示するには、【詳細】を選択します。これらのプロパティは、この【詳細】ボタンの下に表示されます。

File name:

<< Advanced

Process Name:

Target Namespace:

Expression Language: ...

☒ Suppress Join Failure

☐ Abstract Process

BPEL Version:

BPEL 2.0 Extensions

☒ Create XPath ☒ Disable Selection Failure ☒ Links are Transitions

Editing Style

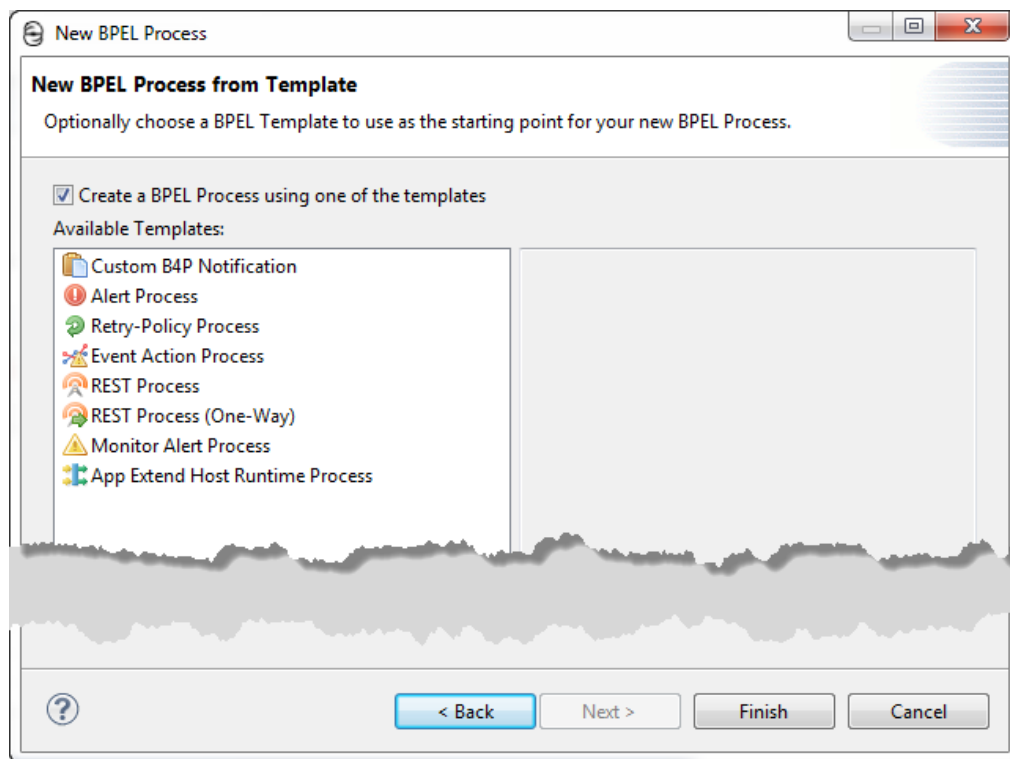
☒ BPMN ☐ Classic

? < Back Next > Finish Cancel

デフォルト値（チェックボックスのプロパティ）が表示され、*Process Developer* の設定内のすべてのプロセスに設定されます。なお、このプロセスのプロセス名、ターゲット名前空間、およびその他のプロパティは、後ほど設定できます。詳細については、「プロセス要素とプロパティ」を参照してください。

このダイアログに入力した後に、**[次へ]** をクリックします。

4. [BPEL テンプレート] ページを選択して、特別な目的のプロセスを作成します。詳細については、「システムサービスベースのプロセスへの BPEL テンプレートの使用」を参照してください。



5. **【完了】** をクリックします。

BPEL ファイルが Process Editor の **【プロセスアクティビティ】** タブで開きます。

フォルダの横にエラーアイコンが表示されることに注意してください。このアイコンは、BPEL ファイルの検証が実行され、現在はプロセスが無効であることを示します。プロセスに必要なアクティビティを追加すると、このアイコンは表示されなくなります。

検証の詳細については、「[プロジェクトのオーケストレーションおよび検証ビルダーについて](#)」を参照してください。

BPMN 中心のパレットまたは BPEL 中心のパレットの選択

BPMN パレットには、ビジネスプロセスモデリング表記法でも使用される標準のアイコンが表示されます。BPEL パレットでは、BPEL 用語コンテキスト内に多くの BPMN アイコンが使用されています。いずれのパレットを使用しても、同じ BPEL ソースコードが生成されます。

BPMN パレットには、ビジネスプロセスモデリング表記法でも使用される標準のアイコンが表示されます。BPEL パレットでは、BPEL 用語コンテキスト内に多くの BPMN アイコンが使用されています。いずれのパレットを使用しても、同じ BPEL ソースコードが生成されます。

エディタパレットに表示するアイコンのセットを選択できます。

- **BPMN 中心**
標準のビジネスプロセスモデルと表記要素を使用します
- **BPEL 中心**
BPEL ネーミングコンテキスト内で BPMN 表記を使用します

Process Developer で **【新規 BPEL プロセス】** ダイアログの **【編集スタイル】** 領域内にアイコンを表示する方法を設定します。[「レイアウトの設定」](#) (ページ 43) のパレットスタイルにワークスペース全体のプリファレンスを設定することもできます。

詳細については、[第 5 章, 「BPMN 中心および BPEL 中心の編集スタイル」 \(ページ 47\)](#)および「[どちらの編集スタイル \(BPMN 中心または BPEL 中心\) を選択するかについて」 \(ページ 52\)](#)を参照してください。

既存の BPEL プロセスのインポート

Process Developer の外部で BPEL プロセスを作成した場合は、BPEL および関連ファイルを Orchestration プロジェクトにインポートできます。BPEL プロセスがすでに Eclipse プロジェクトに含まれている場合は、[「プロジェクト Orchestration ネーチャーの追加または削除」 \(ページ 88\)](#)を参照してください。

これを行う手順は次のとおりです。

1. **[ファイル] > [新規] > [Orchestration プロジェクト]** を選択し、**[次へ]** をクリックします。
2. プロジェクト名を入力し、**[完了]** をクリックします。
3. XSD ファイルがある場合は、スキーマフォルダを選択し、マウスを右クリックして **[インポート]** を選択します。
4. **[全般] > [ファイルシステム]** を選択します。
5. XSD ファイルが存在するディレクトリを参照します。XSD ファイルをスキーマフォルダにインポートします。
6. [「サービス参照のインポート」 \(ページ 101\)](#)に記載されているように、WSDL を wsdl フォルダにインポートするか、サービス参照を使用します。
7. BPEL ファイルを bpeL フォルダにインポートします。
8. BPEL ファイルをダブルクリックして、Process Editor で開きます。
ファイルが、BPEL バージョンに関連付けられたモード (WS-BPEL 2.0 または BPEL4WS 1.1) で開きます。これは、作成時に使用したスタイルシートに関連付けられた、BPMN、BPEL、または Process Developer Classic のいずれかのモデリングスタイルで開かれます。
別のスタイルシートでファイルを保存する方法の詳細については、[第 5 章, 「BPMN 中心および BPEL 中心の編集スタイル」 \(ページ 47\)](#)を参照してください。
9. **[問題]** ビューを使用して、発生した検証エラーを修正します。

参照される WSDL とスキーマを追加する前に BPEL プロセスをインポートした場合、プロセスの名前空間が見つかるまで Process Developer では検証エラーが表示されます。

WS-BPEL 2.0 プロセスに、WS-BPEL 2.0 仕様の拡張機能である要素と属性が含まれている場合は、それらが自動的に宣言されます。詳細については、[「拡張要素と属性の宣言」 \(ページ 126\)](#)を参照してください。

Visio XML 図面ファイルのインポート

ワークスペースで使用する Visio 図面は、BPMN 形状に基づきます。Visio 図面ファイルをワークスペースプロジェクトにインポートします。

BPMN テンプレートまたは BPMN 図形を使用したフローチャートで作成された Visio 2010 Premium (またはそれ以降) から Visio XML 図面をインポートできます。他の種類の Visio 図面は BPEL に変換できません。

ファイルをインポートする手順

1. .vdx という拡張子の XML 図面としてファイルを保存します。
2. **[ファイル] > [インポート] > [オーケストレーション] > [Visio XML 図面]** を選択します。

3. ワークスペースまたはファイルシステムから Visio ファイルを選択します。ファイルの拡張子は.vdx である必要があります。
4. ファイルのプロジェクトフォルダの場所を選択します。例えば、Orchestration プロジェクトの「bpel」フォルダを選択します。
5. 必要に応じて、BPEL に変換されるファイルの名前を入力します（またはデフォルトのままにします）。ファイルに複数のプールが含まれている場合は、名前を指定する必要がある場合があります。Visio ファイルに複数のプールが含まれている場合、各プールは 1 つの BPEL ファイルに変換されます。1 つの BPEL ファイル内での複数のプールはサポートされていません。Process Developer によって、.bpel 拡張子が自動的に追加されます。
6. 必要に応じて、既存の BPEL ファイルを上書きできます。

Visio ファイルを変換するためのヒント:

- Visio ファイルはプロジェクトにインポートされません。インポートウィザードによって、Visio ファイルを BPEL に変換します。
- スケール係数は、アクティビティ間のデフォルトのスペースを使用して BPEL プロセスを作成します。スケール係数は、Process Developer でピクセルに変換する Visio の測定単位（デフォルトではインチ単位）です。デフォルトの 150 で作成される BPEL プロセスのアクティビティ間のスペースが広すぎる場合は、値を 100 に設定します。または、Process Developer で **【自動レイアウト】** を選択して、キャンバス上のアクティビティの配置を整えます。
- プール間に存在するメッセージングデータはサポートされていません。Process Developer では、プロセスデプロイメント記述子で process-to-process または process-to-subprocess 呼び出しハンドラを作成する必要があります。
- 変換時に生成されたメッセージを確認するには、[エラー] ビューと [問題] ビューを表示します

Process Developer を利用した最初のプロジェクトの開始

Process Developer には、最初のプロセスを構築するための多くのエントリポイントが用意されています。開始方法を決定する際に役立つ一部の指針をここに示します。

ユーザーの知識、ユーザーのリソース	開始ポイント
知識やリソースがない	http://www.activevos.com の「Process Developer Developer's Education Center」にある <i>Hello World</i> の例を使用します。
Process Developer をエンドツーエンドで使用方法を学びたい	Process Developer のチュートリアルを閲覧します。
WSDL はないが、プロセスで実行したい内容を把握している	プロセス、サービスロール、およびパーティシパントを生成するには、「 「[パーティシパント] ビューの使用」 (ページ 139) 」を参照してください

ユーザーの知識、ユーザーのリソース	開始ポイント
構築するプロセスの入力と出力を表すサンプルデータファイル（XML ファイル）がある	サービスとサービスパーティシパントを生成するには、 「[パーティシパント] ビューの使用」 （ページ 139）を参照してください
WSDL がある	WSDL をインポートし、[パーティシパント] ビューを使用してサービスパーティシパントとアクティビティを構築します

組み込みプロセスサーバーのセットアップ

プロセスサーバーは、Apache Tomcat で実行中のサーバーエンジンで構成されています。Tomcat は、Java Servlet および Java Server Pages テクノロジーの公式リファレンス実装で使用するサーブレットコンテナです。

BPEL プロセスとそのデプロイメント記述子を作成すると、実行中のサーバーにプロセスリソースを直接デプロイできます。これにより、プロセスを実行してリモートでデバッグできます。

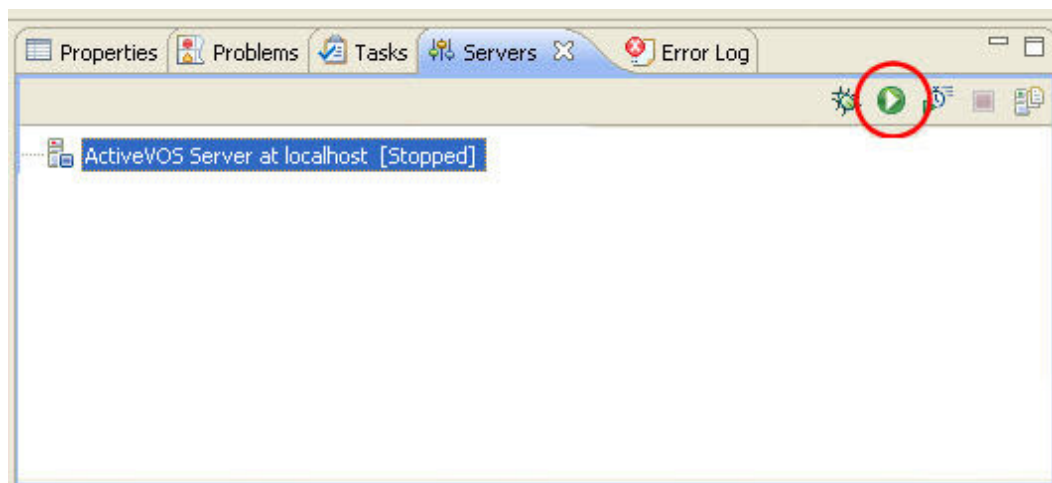
サーバーをセットアップする手順

1. ワークスペースの右下にある [サーバー] ビューを選択します。
2. [サーバー] ビュー内でマウスを右クリックし、**[新規] > [サーバー]** を選択します。
3. [サーバータイプ] リストで、**[プロセスサーバー]** を選択してから、**[次へ]** を選択します。
4. デフォルトの HTTP ポートとコネクタポートがコンピュータですでに使用されている場合は、[ポート] ページで変更し、**[完了]** をクリックします。

コンソールに Tomcat の起動アクティビティが表示されます。起動の最後の部分で、Tomcat に「server started」というメッセージが表示されます。

サーバーを起動する手順

Tomcat の開始後に、図のようにプロセスサーバーの [サーバー] ビューで **[サーバーの起動]** アイコンを選択します。



プロセスコンソールを表示するには、Process Developer のツールバーにある **【プロセスコンソール】** アイコンを選択します。

プロセスコンソールはブラウザで表示することもできます。ブラウザでプロセスコンソールを表示するには、コンソールの URL に移動します。デフォルトの URL は <http://localhost:8080/activevos> です。

パート II: プロセスの作成と変更

このセクションのトピックでは、プロセスを作成および変更する方法を説明します。

第 8 章

インタフェース、サービス参照、ローカル WSDL

「インタフェース」ビューで WSDL、スキーマ、および XML データファイルを管理して、設計をより迅速かつ簡単に行うことができます。

ヘルプのこの部分に記載された内容は次のとおりです。

- [「インタフェース、サービス参照、ローカル WSDL について」 \(ページ 99\)](#)
- [「ローカル WSDL のインポート」 \(ページ 100\)](#)
- [「サービス参照のインポート」 \(ページ 101\)](#)
- [「新しいインタフェースの作成」 \(ページ 102\)](#)
- [「\[インタフェース\] ビューを使用したアクティビティの作成」 \(ページ 111\)](#)
- [「Java インタフェースの作成」 \(ページ 103\)](#)
- [「WSDL メッセージのサンプルデータの使用」 \(ページ 113\)](#)
- [「WSDL コンポーネントの使用箇所の検索」 \(ページ 116\)](#)
- [「プロセス検索の使用」 \(ページ 117\)](#)

インタフェース、サービス参照、ローカル WSDL について

すべての BPEL プロセスには、Receive や Invoke といった Web インタラクションアクティビティ用のインタフェースが必要です。このインタフェースは、オーケストレーションで交換されるサービスとメッセージを記述する Web サービス記述言語 (WSDL) ファイルです。Process Developer には、組み込み WSDL ファイルを追加、インポート、作成、または使用するための次のような簡単な方法が用意されています。

- ローカル WSDL ファイルをプロジェクトにインポートする。
- リモート WSDL への参照をインポートする。
- 新しいパーティシパントに基づいて新しい WSDL ファイルを作成する。
- サンプルデータ、スキーマ、または Java インタフェースに基づいて新しい WSDL ファイルを作成する。
- システムサービスの使用。

新しい WSDL ファイルを Orchestration プロジェクトに追加すると、すべてのプロセスで使用できるようになります。

WSDL ファイルとスキーマを追加することで、名前空間、メッセージ、型定義、サンプルデータ、および BPEL プロセス全体で使用するその他の要素の便利なレジストリが得られます。作業を迅速かつ簡単にするために、ウィザードによって必要な WSDL 要素が自動的に含まれるため、プロセス設計をすぐに開始することができます。

さらに、WSDL を追加する際にメッセージ変数のサンプルデータファイルを追加して、プロセスのテストとデバッグでデータを使用できるようにすることができます。

WSDL インタフェースを表示するには、[パーティシパント] ビューまたは [インタフェース] ビューを選択します。

WSDL ファイルは、名前空間宣言、型宣言、メッセージ、およびビジネスプロセスの作成に使用可能なその他の要素定義を含む XML ベースのファイルです。WSDL ファイルは、ワールドワイドウェブコンソーシアムによって作成された Web サービス記述言語標準に準拠しています。

Process Developer は、2001 年 3 月 15 日付けの W3C 勧告である Web サービス記述言語 (WSDL) 1.1 をサポートしています。この標準の詳細については、<http://www.w3.org/TR/wsdl> を参照してください。

詳細については、次を参照してください:

- [「ローカル WSDL のインポート」 \(ページ 100\)](#)
- [「サービス参照のインポート」 \(ページ 101\)](#)
- [「新しいインタフェースの作成」 \(ページ 102\)](#)
- [「システムサービスインタフェース」 \(ページ 112\)](#)

ローカル WSDL のインポート

[「Orchestration プロジェクトの作成」 \(ページ 85\)](#)に記載されているように、プロジェクトエクスプローラでは、プロセスリソースを管理するためのフォルダをプロジェクトに追加できます。これらのフォルダの 1 つは WSDL 用です。WSDL をこのフォルダまたはプロジェクトエクスプローラの任意の場所にインポートして、プロセスで使用するためにローカルで使用可能な WSDL を作成できます。

WSDL とそのスキーマをプロジェクトエクスプローラにインポートすると、ポートタイプが [パーティシパント] ビューと [インタフェース] ビューに追加されます。

WSDL をインポートする手順

1. プロジェクトエクスプローラで、wsdl フォルダを右クリックします (またはプロジェクトを選択します)。WSDL でスキーマまたは他の WSDL をインポートする場合は、検証警告メッセージが表示されることを避けるために、最初にファイルをインポートします。
2. [インポート] を選択します。
3. [インポート] ダイアログで、インポートソースとして [ファイルシステム] を選択します。
4. WSDL を参照して選択します。
5. WSDL ポートタイプが [パーティシパント] ビューに追加されることに注意してください。

WSDL ファイルは次のように管理できます。

- [「WSDL ツリーのキー要素の表示」 \(ページ 101\)](#)
- [「プロジェクトからの WSDL の削除」 \(ページ 101\)](#)

WSDL ツリーのキー要素の表示

プロジェクトエクスプローラには、インポートされた WSDL ファイルの名前が表示されます。WSDL ファイルは XML ファイルであるため、ツリー構造でファイルビューを展開するか折りたたんで要素を表示できます。

WSDL ファイルを XML ツリー構造として表示するには、ファイル名の横にあるプラス記号をクリックして、ファイル内の次のような要素を表示します。

- バインディングと操作
- メッセージ
- パートナーリンクの種類、ロール、および操作
- ポートタイプと操作
- プロパティとプロパティエイリアス
- サービス名とポート

WSDL エディタでの WSDL の編集

WSDL エディタで表示および編集するには、プロジェクトエクスプローラで WSDL ファイルをダブルクリックします。エディタで強調表示するには、操作またはポートタイプをダブルクリックします。

WSDL を変更すると BPEL ファイルの参照に影響を与える可能性があるため、これらの影響を解決する必要があります。例えば、変更したパートナーリンクのロール名が BPEL プロセスで参照されている場合、このロール名は [未解決の参照] としてマークされます。プロセスが有効であることを確認するには、[プロパティ] ビューから新しい名前を選択する必要があります。

プロジェクトからの WSDL の削除

不要になった WSDL を削除できます。プロジェクトエクスプローラから WSDL ファイルを選択し、マウスを右クリックして、**[削除]** を選択します。プロジェクトの場所が無効になるため、WSDL を使用するプロセスはすべて無効になります。パートナーリンクタイプとポートタイプのエントリは、[インタフェース] ビューから削除されます。

サービス参照のインポート

サーバーまたはリポジトリからリモート WSDL をインポートします。リモート WSDL がローカルコピーと同期されるようにします。

サービス参照とは、外部 WSDL サービスのプロジェクトベースのカタログです。外部の場所への接続を保持しながら、外部サービスをプロジェクトにインポートできます。サービス参照は元の場所として扱われますが、ローカルで解決されます。

サービス参照を使用すると、元のファイルに加えられた変更の認識を保持しながら、WSDL とスキーマのローカルコピーを保持できます。この方法によって、ローカルコピー内で更新を処理する最適な方法を決定できます。

サービス参照をインポートすることで、次のことが可能です。

- ローカルの WSDL ファイルを元のソースと比較する。
- 変更を行っても WSDL ファイルをインポートするプロセスに影響がない場合に、ローカルコピーを手動で編集して、元のソースから更新する。

- WSDL フォルダにインポートされた WSDL を使用方法と同様に、WSDL を使用する。

ローカルに接続された外部 WSDL のコピーをインポートする手順

1. Orchestration プロジェクトの [サービス参照] フォルダを右クリックし、[インポート] を選択します。
2. ネットワークサーバー、イントラネット、またはインターネット上にある WSDL ファイルの URL を貼り付けます。
3. インポートしたサービスを [サービスグループ] フォルダに関連付けます。[サービスグループ] フォルダには、インポートしたサービスと、WSDL ファイルや XSD ファイルなどのすべてのインポートが含まれます。推奨されるサービスグループ名として、元の WSDL の場所と名前が表示されます。[インポート] ダイアログで [完了] を選択すると、[サービス参照] フォルダ内にサービスグループが表示されます。
4. XML を読みやすくするために、[インポート後にファイルをフォーマットする] チェックボックスをオンにします。インポートウィザードは、子要素のインデントと改行の作成を試行します。

サービス参照を元のソースと比較する手順

ローカル WSDL ファイルを元のソースと比較するには、[サービスグループ] フォルダを右クリックし、[比較] > [ソースのインポート] を選択します。ファイルが同期されていない場合、[比較] ウィンドウに WSDL ファイルが表示されます。ファイルをクリックして、ローカルコピーと元のコピーの差異を表示します。差異のみを表示することができます。編集は許可されていません。

サービス参照の使用に関するヒント:

- 新しいポートタイプあるいはスキーマ、または変更されたポートタイプあるいはスキーマの変更によって元のファイルが更新された場合に、元のファイルと一致するようにローカルコピーを編集すると、WSDL のインポートプロセスでエラーが表示されます
- 元のファイルが使用できなくなった場合は、WSDL またはスキーマをプロジェクトにコピーしてから、インポートの場所を変更してプロジェクトの場所を指すようにする必要があります。これにより、リモート検証の発生による長い待機時間が回避されます。インポートの [プロパティ] ビューでインポートの場所を変更します。インポートは [アウトライン] ビューに一覧表示されます。
- WSDL 検証のタイムアウト値の設定に関する詳細については、[「キャッシュとタイムアウトの設定」\(ページ 41\)](#) を参照してください。

新しいインタフェースの作成

WSDL ファイルがない場合は、新規インタフェースウィザードをクイックスタートとして使用して、基本的な WSDL ファイルを生成できます。

WSDL ファイルがない場合は、新規インタフェースウィザードを使用して、WSDL ファイルの基本を生成できます。WSDL ファイルを生成するためには、まず最初にポートタイプおよび操作の入出力メッセージの基礎として、次のうちの 1 つまたは複数を設定します。

- XML スキーマ
- サンプル XML データ
- Java インタフェース

新しいインタフェースを作成する手順

1. 次のいずれかを実行します。
 - XSD または XML ファイルを Orchestration プロジェクト内のプロジェクトフォルダに追加します。これらのファイルでデータ型を説明し、名前空間の定義を含める必要があります。

- [「Java インタフェースの作成」 \(ページ 103\)](#)に記載されている手順に従います
 - 2. パーティシパントにインタフェースを作成するか（[パーティシパント] ビュー）、または [インタフェース] ビューのツールバーにある **【新規インタフェース】** アイコンを選択して、新規インタフェースウィザードを表示します。
 - 3. 新規インタフェースウィザードで、新しい操作が同期か非同期かを選択します。
 - 要求 - 応答操作
 - 一方向操作
 - 4. 操作の各メッセージを説明するために使用する、次のいずれかのデータ型を選択します。
 - XML スキーマ[XML スキーマ]
 - サンプル XML
 - 5. スキーマまたはサンプルを参照して選択します。スキーマの場合、メッセージに使用する要素を選択します。
ウィザードでエラーが報告された場合は、データファイルに有効な定義、宣言、および構文が含まれていることを確認してください。例については、チュートリアルプロジェクトにある「サンプルデータファイルとスキーマ」を参照してください。
 - 6. プレビューでメッセージを表示します。
 - 7. 入力と出力に選択した名前に基づいて、ウィザードは必要な WSDL 要素を作成します。ポートタイプ、操作、およびターゲット名前空間要素をよりわかりやすい名前に変更する必要があります。
ターゲット名前空間は、WSDL ドキュメントがそのドキュメント自体を参照できるようにする XML スキーマの規則です。これは一意の値であり、定義された他のすべての名前空間とは異なります。
 - 8. 提案された名前で新しい WSDL ファイルを保存するか、新しい名前を入力してプロジェクトの場所にファイルを保存します。
 - 9. 生成された WSDL を WSDL エディタで表示して、修正と追加を行います。
- 生成された WSDL は、プロセスインタフェースの開始ポイントとして使用できます。WSDL は次のように変更できます。
- スキーマの場所を修正または追加する。
 - フォールト名とメッセージを追加する。
 - 別の操作を追加し、生成されたパーティシパント（またはパートナーリンクタイプ）に追加する。（パートナーリンクタイプには 2 つのロールがあり、これらは非同期で通信する 2 つの異なるサービスを表します）。または、[パーティシパント] ビューでコールバックインタフェースを追加します。[「新しいコールバックインタフェースの作成」 \(ページ 142\)](#)を参照してください。
 - tns プレフィックスを意味のあるものに変更する。操作ウィザードを使用してアクティビティを作成すると、プロセスにプレフィックスが自動的に追加されます。

Java インタフェースの作成

Process Developer には、Java 開発者が既存の Java プロジェクトを使用して BPEL プロセスで Java エンドポイントを構築するための簡単な方法が用意されています。プロジェクトには、POJO（プレーンオールド Java オブジェクト）または EJB を含めることができます。（JavaBean は、Apache Tomcat を除くすべてのアプリケーションサーバーで使用できます）。インタフェース、およびインタフェースが実装された Java クラスを含むパッケージから開始するか、パッケージを構築することができます。

Process Developer に用意された方法を使用して、Java インタフェースから WSDL とスキーマを自動的に生成できます。WSDL には、Receive、Reply、Invoke を作成するためのポートタイプ、操作、およびメッセージが含まれています。

Process Developer には、サーバーへのデプロイに必要なすべての JAR およびその他のファイルを自動的に含める組み込み機能も用意されています。

ステートレスまたはステートフルな呼び出し

Java インタフェースを使用すると、BPEL で Invoke アクティビティを簡単に作成できます。これらの呼び出しは、ステートレスまたはステートフルにすることができます。

- ステートレスな呼び出しとは、Java コードが BPEL プロセスから呼び出されるたびに、Java クラスの新しいインスタンスがインスタンス化されることを意味します。
- ステートフルな呼び出しは、`java.io.Serializable` マーカーインタフェースの実装としてマークされる Java クラスに基づいています。ステートフルな呼び出しを使用すると、プロセスは、その呼び出しを使用する `partnerLink` の存続期間にわたって同一の Java インスタンスを使用します。同じ POJO パートナーリンクで複数の呼び出しを行うと、同一の POJO インスタンスでメソッドが呼び出されます。

詳細については、次を参照してください:

- [「Process Developer での Java プロジェクトのセットアップ」 \(ページ 104\)](#)
- [「Java プロジェクトの制約」 \(ページ 105\)](#)
- [「Java インタフェースからの WSDL とスキーマの生成」 \(ページ 105\)](#)
- [「スキーマ要素の引数名の生成」 \(ページ 107\)](#)
- [「Java プロジェクトと BPEL プロセスの同時更新」 \(ページ 108\)](#)
- [「Java \(POJO\) エンドポイントのデプロイメント要件」 \(ページ 109\)](#)
- [「Java \(EJB\) エンドポイントのデプロイメント要件」 \(ページ 109\)](#)

Process Developer での Java プロジェクトのセットアップ

BPEL プロセスの Java インタフェースを作成するには、既存の Java プロジェクトから開始するか、新しい Java 対応の Orchestration プロジェクトを作成します。

Java プロジェクトのセットアップ:

1. **【ファイル】 > 【新規】 > 【Orchestration プロジェクト】** を選択します。
2. プロジェクトに名前を付けて、**【次へ】** をクリックします。
3. Java 対応オーケストレーションを選択します。
4. Orchestration プロジェクトで、新しい Java プロジェクトをインポートまたは作成します。Java プロジェクトには次のものが含まれている必要があります。
 - 1 つ以上のメソッドを持つ Java インタフェース。
 - Java インタフェースの具体的な実装。

ヒント: Eclipse で Java プロジェクトを開いている場合は、[Eclipse Java] パースペクティブから Process Developer のパースペクティブに切り替えることができます。次に、プロジェクトエクスプローラでプロジェクトを右クリックし、**【オーケストレーションネーチャーの追加】** を選択します。

Java プロジェクトにオーケストレーションネーチャーを追加すると、バックグラウンドの検証ビルダにより、プロセス、インポートされた WSDL、およびその他のリソースのエラーと問題を確実に把握できるようになります。また、各ビルダによって、生成された WSDL が、ソースの Java に加えられた変更と自動的に同期されます。

Java プロジェクトを Process Developer に追加した後で、アクティビティの作成に必要な BPEL 指向のインタフェースである WSDL を生成できます。詳細については、[「Java インタフェースからの WSDL とスキーマの生成」 \(ページ 105\)](#)を参照してください。

関連事項:

- [「Java プロジェクトの制約」 \(ページ 105\)](#)
- [「Java インタフェースの作成」 \(ページ 103\)](#)

Java プロジェクトの制約

Java インタフェースの実装で使用する Java クラスには、次の要件があります。

- Java インタフェースのメソッドパラメータタイプは、JavaBeans および次のプリミティブ型の Java に制限されています。
 - byte、short、int、long、float、double、boolean、char
 - Byte、Short、Integer、Long、Float、Double、Boolean、Character
 - String
 - BigInteger
 - Date
- Java でチェックされた例外はフォールトとしてモデル化されますが、このフォールトが適切にスローされるためには、実行時に Java メソッドが同じ例外をスローする必要があります。
- WSDL ジェネレータへの Java インタフェースは JAXB によって実行され、JavaBeans のサポートが JAXB から継承されます。
- 配列とコレクションは現在、Java メソッドの引数としてサポートされていません。ただし、コレクションが JavaBeans でラップされている場合、コレクションはサポートされます。
- インタフェースが実装されたクラスには、デフォルトのコンストラクタが必要です。
- インタフェースが実装されたクラスは、ステートレスまたはステートフルになります。ステートフルクラスとは、`java.io.Serializable` マーカーインタフェースの実装としてマークされるクラスです。

関連事項:

- [「Java インタフェースの作成」 \(ページ 103\)](#)

Java インタフェースからの WSDL とスキーマの生成

WSDL がない場合、WSDL の基本を生成するためのクイックスタートとして新規インタフェースウィザードを使用できます。

Java インタフェースからの WSDL とスキーマの生成

Process Developer のプロジェクトエクスプローラで Java プロジェクトを設定した後に、WSDL とスキーマを生成します。BPEL プロセスには WSDL インタフェースが必要で、Process Developer では Java インタフェースから自動的に生成されます。

さらに、Process Developer で生成された WSDL は、ユーザーがソース Java に加えた変更と自動的に同期されます。

WSDL とスキーマを生成する手順

1. Java Orchestration プロジェクトで、`bpel` フォルダに新しい BPEL プロセスを作成します。
2. [パーティシパント] ビューで [パートナーサービスプロバイダ] を右クリックし、[新規パートナーサービスプロバイダ] を選択します。

3. **【パートナーサービスプロバイダ】** ダイアログで、**【インタフェースの生成】** を選択します。
4. 新規インタフェースウィザードで、**【Java インタフェースから】** を選択します。
5. Java インタフェースを参照して選択します。
6. 新しい WSDL がプロジェクトの `wsdl` フォルダに生成されることに注意してください。
7. **【完了】** を選択し、**【パートナーサービスプロバイダ】** ダイアログで **【OK】** を選択します。

WSDL の生成

Java インタフェースから WSDL とスキーマを生成すると、以下のアクションが発生します。

- Java インタフェースのすべてのメソッドが、WSDL 操作としてマニフェストされます。
- すべてのメソッド引数が、最上位の XML スキーマ要素宣言としてマニフェストされます。引数が、「`argl_string`」などの一般的な名前で作成されます。実際の引数名を生成する方法については、[「スキーマ要素の引数名の生成」 \(ページ 107\)](#) を参照してください。
- メソッドの戻り値の型が、最上位の XML スキーマ要素宣言としてマニフェストされます。
- メソッドで宣言されたすべての例外が、WSDL フォールトとしてマニフェストされます。
- 自動的に更新される派生ファイルであるため、WSDL には「編集しないでください」というコメントが表示されます。

結果の WSDL には、次の構成が含まれます。

- `[JavaInterfaceName]PLT` という名前の 1 つのパートナーリンクタイプ
- `[JavaInterfaceName]` という名前の 1 つのポートタイプ
- Java インタフェースのメソッドごとにポートタイプ内で 1 つの操作
- Java インタフェースメソッドごとに次の 2 つのメッセージ:
 - 同じ名前の単一のパートを持つ `[methodName]` という名前のメソッドの引数に対する 1 つのメッセージ。
 - 同じ名前の単一のパートを持つ `[methodName]Response` という名前のメソッドの戻り値に対する 1 つのメッセージ。
- Java インタフェースメソッドごとに次の 2 つの組み込みスキーマ要素:
 - `methodName` という名前付きの Java インタフェースメソッドの引数をモデル化するための 1 つの要素。
 - `[methodName]Response` という名前付きの Java インタフェースメソッドの戻り値をモデル化するための 1 つの要素。
- インタフェースで使用する各 Java クラスのスキーマ `complexType` とスキーマ要素。

スキーマの生成

- 最上位の要素宣言は、明示的なクラス参照ごとに作成されます。
- 暗黙的な宣言は最上位の `complexTypes` として明示され、スキーマの他のパート全体で再利用されます (該当する場合)。

WSDL とスキーマを生成するその他の方法:

WSDL を生成する方法	結果
[インタフェース] ビューで、[新規インタフェース] ツールバーボタンを選択する。	Java インタフェースを選択する新規インタフェースウィザードが開きます
プロジェクトエクスプローラで、Java プロジェクトからインタフェースを選択し、右クリックメニューから、[WSDL の生成] を選択する	WSDL とスキーマが、選択した場所に生成されます
Java プロジェクトから、Java インタフェースメソッドを Process Editor キャンバスにドラッグアンドドロップする	WSDL とスキーマの生成だけでなく、新しい Web サービスアクティビティ (Receive など) が作成可能な、アクティビティの作成ウィザードが開きます。

関連事項:

- [「Java プロジェクトと BPEL プロセスの同時更新」 \(ページ 108\)](#)
- [「Java インタフェースの作成」 \(ページ 103\)](#)

スキーマ要素の引数名の生成

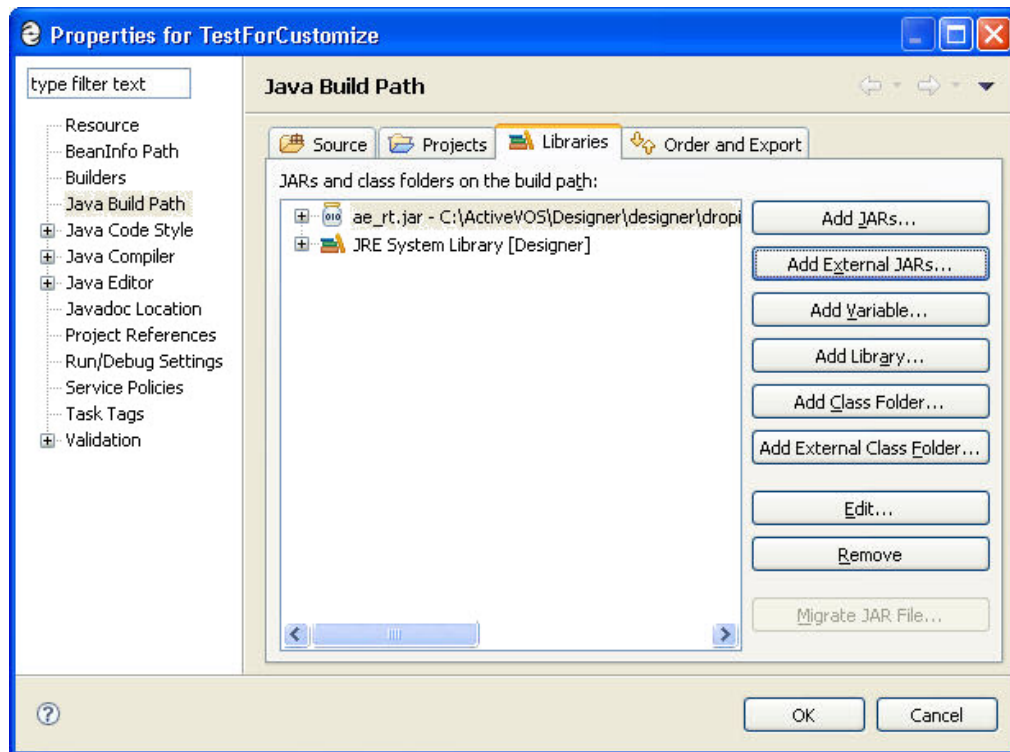
[「Java インタフェースからの WSDL とスキーマの生成」 \(ページ 105\)](#)に記載されているように、WSDL ファイルを生成すると、メソッドの引数がスキーマ要素として生成されます。次のように、要素には arg1_string などの一般的な名前が付いています。

```
<xs:element form="unqualified" name="arg1_string" type="xs:string"/>
```

WSDL を生成する前に ae_rt.jar という JAR ファイルをプロジェクトの外部 JAR の Java ビルドパスリストに追加し、インタフェースに注釈を追加することで、実際の引数名を使用してスキーマ要素を生成できます。

引数名を生成する手順

1. プロジェクトエクスプローラで、プロジェクトを右クリックし、[プロパティ] > [Java ビルドパス] を選択します。
2. [ライブラリ] タブを選択します。
3. [外部 JAR の追加] を選択します。
4. [Designer インストール]\designer\plugins\org.activebpel.enginep_[version_number]\server\shared\lib に移動します
5. 次のように、ae_rt.jar を選択します。



6. Java インタフェースファイルを開き、@AeWsdParam 注釈を各引数に追加します。以下に例を示します。

```
Public interface IMyInterface {
    public void execute(
        @AeWsdParam("someValue") String someValue);
}
```

7. Java インタフェースを右クリックし、**[WSDL の生成]** を選択して、WSDL を生成（または再生成）します。

Process Developer で、引数名を使用してスキーマ要素が生成されます。例:

```
<xs:element form="unqualified" name="someValue" type="xs:string"/>
```

注:

- 生成された WSDL は編集しないでください。Java インタフェースを更新する場合は、WSDL を再生成する必要があります。
- ae_rt.jar をデプロイする必要はありません。これはすでにサーバーの classpath 上にあります。
- Process Developer を新しいバージョンにアップグレードする場合は、古いバージョンの ae_rt.jar を削除して、新しいバージョンを再度追加する必要があります。

Java プロジェクトと BPEL プロセスの同時更新

呼び出されている Java コードとそのコードを呼び出している BPEL をアクティブに開発できます。Process Developer は、ソースの Java インタフェースまたはクラスが変更されるたびに WSDL とスキーマを再生成します。例えば、Java インタフェースに新しいメソッドを追加すると、Process Developer は、生成された WSDL と生成されたすべてのスキーマを自動的に再生成します。

WSDL が変更されると、BPEL プロセスに未解決の WSDL 参照インジケータが表示されます。エラーメッセージには、各アクティビティに対して WSDL を使用して行う必要のある修正の内容が示されます。

[「Java インタフェースの作成」 \(ページ 103\)](#)も参照してください。

Java (POJO) エンドポイントのデプロイメント要件

実行時に作成および呼び出しを行う Java クラスを選択します。

EJB サービスの詳細については、[「Java \(EJB\) エンドポイントのデプロイメント要件」 \(ページ 109\)](#)を参照してください。

Java インタフェースに基づく WSDL を使用して BPEL プロセスを開発した後に JAR ファイルを作成することで、デプロイメントの準備が整います。次に、Java インタフェースに固有のデプロイメントの詳細を入力します。プロセスのプロセスデプロイメント記述子 (PDD) ファイルの作成時に、実行時に作成および呼び出される Java クラスの詳細を指定する必要があります。

必要な POJO デプロイメントの詳細を指定する手順

1. [「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#)に記載されているように、PDD エディタを開きます。
2. [パートナーリンク] タブで、Java インタフェースを呼び出すパートナーロールを選択します。Invoke ハンドラの Java サービスが選択されることに注意してください。
3. **[Java 呼び出しハンドラのプロパティ]** ダイアログで、実行時に作成および呼び出される Java クラスを選択します。
4. Java クラスをインスタンス化および呼び出すときに使用する Java クラスパスを選択します。クラスパスには、呼び出されるクラスを含んだ JAR (またはローカルの「src」ワークスペースディレクトリ) だけでなく、そのクラスによって直接的または間接的に参照されるクラスを含んだ JAR が含まれます。デフォルトでは、[サーバークラスローダーの継承] チェックボックスが選択されていることに注意してください。このオプションを無効にすると、クラスパス上の JAR にのみアクセスでき、サーバーにデプロイされた JAR にはアクセスできなくなります。プロセスサーバーのクラスローダーを継承する際に、最後に親クラスローダーを選択することもできます。

デプロイメント詳細

Java インタフェースが実装されたパートナーロールに Java 呼び出しハンドラを選択すると、Java リソースがリソースカタログに追加され、プロセスとともにデプロイされます。これには、クラスパスに含まれるすべての JAR が含まれます。

また、Java サービスに必要なエンドポイント参照情報はありません。

[「Java インタフェースの作成」 \(ページ 103\)](#)も参照してください。

Java (EJB) エンドポイントのデプロイメント要件

実行時に作成および呼び出しを行う Java クラスを選択します。

EJB サービスを使用した BPEL プロセスの設定と設計に関する詳細については、[「Java インタフェースの作成」 \(ページ 103\)](#)を参照してください。

EJB として実装された Web サービスを作成した場合は、EJB サービスと呼ばれるパートナーロール呼び出しハンドラを使用して、JAR ファイルおよびその他のリソースをデプロイできます。PDD エディタでは、呼び出しハンドラによって EJB プロジェクトが POJO として検出され、Java サービスが自動的に選択されることに注意してください。必ず EJB サービスを選択するようにしてください。

また、EJB サービスは Apache Tomcat では使用できないことに注意してください。EJB サービスはその他のアプリケーションサーバーでサポートされています。

EJB サービスハンドラは、JNDI ルックアップを実行して EJB ホームを取得し、メソッドの呼び出しに使用されるオブジェクトをインスタンス化します。

[EJB サービス呼び出しハンドラのプロパティ] ダイアログに入力する手順

1. JNDI 名には、JBoss ejb-jar.xml ファイル、WebLogic weblogic-ejb-jar.xml ファイル、または WebSphere ibm-ejb-jar-bnd.xml ファイルなど、アプリケーションサーバーの EJB ファイルで定義された JNDI 名を指定します。
2. EJB ホームには、アプリケーションサーバーの ejb-jar.xml ファイルで定義されたホームインタフェースを指定します。
このプロパティは WebSphere には必須ですが、JBoss および WebLogic ではオプションです。
3. [クラスパス] ボックスで、Java クラスのインスタンス化および呼び出しの際に使用する Java クラスパスを選択します。クラスパスには、呼び出されるクラスを含んだ JAR（またはローカルの「src」ワークスペースディレクトリ）だけでなく、そのクラスによって直接的または間接的に参照されるクラスを含んだ JAR が含まれます。
デフォルトでは、[サーバークラスローダーの継承] チェックボックスが選択されていることに注意してください。このオプションを無効にすると、クラスパス上の JAR にのみアクセスでき、サーバーにデプロイされた JAR にはアクセスできなくなります。

EJB インタフェースが実装されたパートナーロールに EJB 呼び出しハンドラを選択すると Java リソースがリソースカタログに追加され、プロセスとともにデプロイされます。これには、クラスパスに含まれるすべての JAR が含まれます。

また、EJB サービスに必要なエンドポイント参照情報はありません。

POJO EJB エンドポイントを使用した BPEL プロセスの実行

実行時に、Java エンドポイントに基づいて Invoke アクティビティが実行されると、Process Developer は次のように Java メソッドを呼び出します。

- WSDL メッセージを Java オブジェクトにアンマーシャルします。(POJO の場合、リフレクションが使用されます。EJB の場合、JNDI ルックアップが使用されます。)
- 具象 Java クラスをインスタンス化します。
- WSDL 操作名に基づいて、適切なメソッドを呼び出します。
- メソッドの戻り値を WSDL メッセージにマーシャルします (または Java 例外を WSDL フォールトにマーシャルします)。

POJO/EJB インタフェースとカスタム Java 呼び出しハンドラの比較

Java インタフェースは操作しやすいように設計されています。WSDL が生成され、JAR およびその他のリソースが BPR ファイルに自動的にデプロイされます。ただし、[「Java プロジェクトの制約」 \(ページ 105\)](#)で概説されている要件に従って、Java プロジェクトを単純性を保つ必要があります。

BPEL プロセスで、非同期コールバック、ポリシーアサーション、およびその他の詳細を処理するカスタム Java コードが必要な場合は、[「EJB/Java 呼び出しハンドラのプロパティダイアログ \(オンプレミスのみ\)」 \(ページ 464\)](#)に記載されているように、Java ラッパーのカスタム呼び出しハンドラを使用します。

[インタフェース] ビューを使用したアクティビティの作成

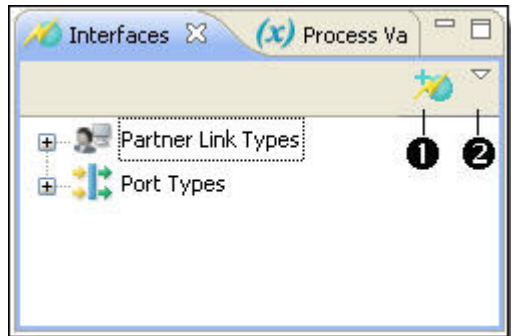
アクティビティを作成するための簡単な開始方法（[インタフェース] ビューを使用する代わりにオプションを使用する方法）については、[第 10 章, 「パーティシパント」 \(ページ 138\)](#)を参照してください。パートナリンクタイプの BPEL 要件に精通していない場合は、[パーティシパント] ビューから開始することもできます。

[インタフェース] ビューの主要な生産性機能の 1 つとして、プロセスの設計で使用するために WSDL 操作にアクセスすることが挙げられます。Orchestration プロジェクトを作成し、WSDL をインポートしてから、ポートタイプの操作をインタフェースから Process Editor キャンバスにドラッグして開始します。詳細については、[「WSDL インタフェースから開始するアクティビティの作成」 \(ページ 160\)](#)を参照してください。

- 作成する必要があるアクティビティの [インタフェース] ビューに WSDL 要素が存在しない場合は、[インタフェース] ツールバーから **[新規インタフェース]** を選択します。新しい WSDL の作成に役立つウィザードが表示されます。詳細については、[「新しいインタフェースの作成」 \(ページ 102\)](#)を参照してください。
- 電子メールの送信や ID サービスからのユーザーの取得などといったシステムサービスアクティビティの場合は、組み込み WSDL からの操作を使用します。詳細については、[「システムサービスインタフェース」 \(ページ 112\)](#)を参照してください。
- Java インタフェースを使用するには、[「Java インタフェースの作成」 \(ページ 103\)](#)を参照してください。

インタフェースツールバーオプション

新しいインタフェースを追加したり、インタフェースをフィルタリングして現在必要なインタフェースのみを表示したりするには、ツールバーオプションを使用します。



1	「新しいインタフェースの作成」 (ページ 102)
2	「[インタフェース] ビューのフィルタリング」 (ページ 111)

[インタフェース] ビューのフィルタリング

参照セットを作成または編集します。また、参照セットを選択して、[Web インタフェース] ビューをフィルタリングします。

デフォルトでは、Process Developer には、ワークスペース内にあるすべてのプロジェクトのすべてのインタフェースが表示されます。複数の BPEL ファイルを同時に開いている場合は、[インタフェース] ビューをフィルタリングして、フォーカスされた BPEL ファイルまたはフォーカスされたプロジェクトに関連するインタフェースのみを表示することができます。

フィルタは次の要素で使用できます。

- **ワークスペース。**すべてのインタフェース。
- **プロジェクト。**現在開いているプロジェクトのインタフェース。
- **プロセス。**フォーカスされた、BPEL ファイルからのインタフェース。
- **システムサービス。**呼び出しアクティビティのインタフェース。

システムサービスインタフェース

新しい【パーティシパント】ダイアログまたは【インタフェース】ビューのフィルタからシステムサービスを選択すると、Process Developer WSDL 内で定義したインタフェースが表示されます。これらの WSDL は公開されているため、プロセスサーバーと通信可能なアクティビティを作成できます。

パートナーリンクのタイプ/操作	説明
プロセスコンシューマサービス	
警告サービス	
イベントアクション	
警告の監視サービス	
REST サービス	詳細については、「REST ベースのサービスの使用」を参照してください
呼び出されたサービスの再試行	詳細については、
「Guide Designer のサービスコールステップ サービス」を参照してください。	詳細については、「 <i>Process Developer Guide Designer</i> 」のドキュメントを参照してください。
タスクのカスタム通知	ユーザーアクティビティの通知期限に実行するカスタムプロセスを作成します。詳細については、「 <i>Process Developer</i> ヒューマンタスクのヘルプ」を参照してください。
パートナーサービスプロバイダサービス	
Process Developer の管理操作	この操作は、Process Developer の管理機能に API を提供します
REST サービス	詳細については、「REST ベースのサービスの使用」を参照してください
電子メールの送信	詳細については、「電子メールサービス」を参照してください
サーバーログコメント	
シェルコマンド実行	

BPEL 用の拡張機能を持つ WSDL ファイルの作成

一部の WSDL ファイルには、実行可能な BPEL プロセスの作成に必要な拡張子が含まれていないものもあります。この拡張機能は、パートナーリンクタイプ、プロパティ、およびプロパティエイリアスです。

ヒント: パートナーリンクタイプを自動的に作成するには、[パーティシパント] ビューを使用します。

Process Developer ウィザードを使用して、実行可能な BPEL プロセスの作成に必要なパートナーリンクタイプ定義を作成できます。

新しいパートナーリンクタイプを作成すると、Process Developer には、それらを既存の WSDL に追加するか、新しい WSDL を作成するかを確認するメッセージが表示されます。新しい WSDL を作成するとプロジェクトに自動的に追加され、インタフェースにカタログ化されます。WSDL には、パートナーリンクタイプの定義のみが含まれています。

さらに、ウィザードを使用すると、相関セットに使用するプロパティとプロパティエイリアスを作成できます。これらの拡張機能は、既存の WSDL または新しい WSDL に追加できます。

詳細については、「[パートナーリンクタイプ](#)」および「[変数プロパティとプロパティエイリアスの追加](#)」を参照してください。

WSDL メッセージのサンプルデータの使用

追加するサンプルのソースの場所を選択します。サンプルデータ、WSDL、XSL ファイルなどのワークスペースリソースを選択します。

WSDL ファイルによって、入力メッセージを受信する操作、および必要に応じて出力メッセージとフォールトメッセージを返す操作を定義します。[パーティシパント] ビューまたは [インタフェース] ビューで、メッセージパートのサンプルデータ値を追加できます。プロセスのテストを行う準備を整えてサンプルのメッセージパート値を変数にロードすると、Web サービスで交換される実際のメッセージをシミュレートできます。プロセスを実行およびデバッグして、入力データ、出力データ、およびフォールトデータの値が処理される際にこれらのデータを調べることができます。

サンプルデータ値を WSDL メッセージに追加することで、すべてのプロジェクトにわたるデフォルトのデータセットが得られます。プロジェクトエクスプローラは、すべてのサンプルファイルとその場所を追跡し、メッセージ変数に値をロードする際にこれらの情報を表示します。

詳細は、

- [「単純型のメッセージパートへのサンプルデータ値の追加または編集」](#) (ページ 113)
- [「サンプルデータファイルの生成」](#) (ページ 114)
- [「WSDL メッセージへのサンプルデータファイルの追加」](#) (ページ 115)
- [「デフォルトのサンプルデータファイルの選択」](#) (ページ 115)
- [「サンプルデータファイルの XML 構造の表示」](#) (ページ 115)
- [「サンプルデータファイルの削除」](#) (ページ 115)

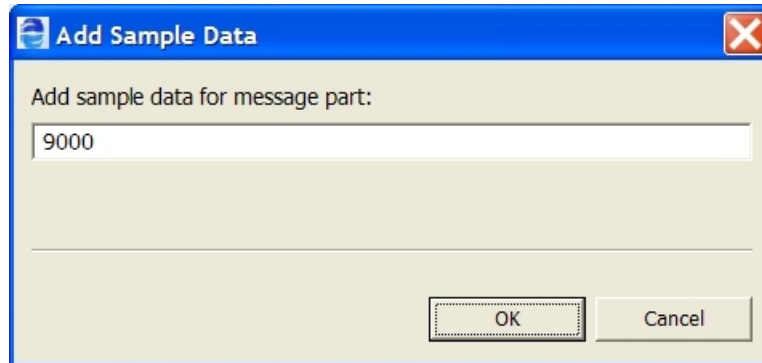
単純型のメッセージパートへのサンプルデータ値の追加または編集

[インタフェース] ビューで WSDL、スキーマ、および XML データファイルを管理して、設計をより迅速かつ簡単に行うことができます。このダイアログを使用して、WSDL 単純型メッセージにサンプルデータを追加するか、単純型の変数のサンプルデータを編集します。

[パーティシパント] ビューまたは [インタフェース] ビューで、単純型のメッセージパートにサンプルデータ値を追加できます。プロセスをテストする準備が整うと、サンプルメッセージパートの値が変数へ自動的にロードされます。また、[プロセス変数] ビューで値を変更したり、プロセスシミュレーション中に値を上書きしたりすることもできます。

単一のサンプルデータ値を追加する手順

1. [パーティシパント] ビューまたは [インタフェース] ビューで、xsd:int などの単純型のパートを使用してメッセージを展開します。
2. 図のように、変数をダブルクリックして **【サンプルデータの追加】** ダイアログを開きます。



3. 値を入力して **【OK】** をクリックします。メッセージパートの下に値が子ノードとして表示されていることに注意してください。
 4. 必要に応じて、サンプルをダブルクリックして編集します。
- 複数のデータ値を追加してから、1つのサンプルデータ値を右クリックして、デフォルトとして選択できます。不要な値は削除できます。

サンプルデータファイルの生成

追加するサンプルのソースの場所を選択します。これは、サンプルデータ、WSDL、XSL ファイルなどのワークスペースリソースです。

複合的なメッセージを含むプロセスのプロセス実行をシミュレートする場合は、メッセージのサンプルデータファイルを生成します。サンプルデータファイルは、WSDL 名前空間で宣言されたスキーマで定義済みのメッセージパートの要素またはタイプで構造化された XML ファイルです。以下は、発注書出力メッセージの部分的なサンプルデータファイルの例です。この例のようなコードが、スキーマタイプまたは要素に基づいて自動的に生成されます。

```
<purchaseOrders
  xmlns="http://samples.cxdn.com/po"
  xmlns:tns="http://samples.cxdn.com/po">
  <purchaseOrder tns:orderDate="2005-09-05">
    <shipTo tns:country="US">
      <name>Jay Lor Jr.</name>
      <street>100 My Street</street>
      <city>My City</city>
      <state>CT</state>
      <zip>06484</zip>
    </shipTo>
    ...
  </purchaseOrder>
</purchaseOrders>
```

サンプルデータを生成する手順

1. [パーティシパント] ビューまたは [インタフェース] ビューで、複合パートを含むメッセージを展開し、サンプル XML データファイルを生成するメッセージパートを選択します。

2. メッセージ部分を右クリックして、**【サンプルの生成】** を選択します。
3. XML データウィザードで、サンプルデータを生成するための設定を入力します。詳細については、[「XML データウィザードの使用」 \(ページ 251\)](#) を参照してください。**【次へ】** をクリックします。
4. ファイルに名前を付けて、Orchestration プロジェクトの sample-data フォルダに保存します。

同じメッセージ部分に対して複数のサンプルデータファイルを生成できます。ファイルに一意の名前を付けることや、同じファイル名を使用してファイルを別のフォルダに保存することができます。Process Developer は、ファイルの場所を追跡します。

WSDL メッセージへのサンプルデータファイルの追加

サンプルデータ、WSDL、XSL ファイルなどのワークスペースリソースを選択します。

サンプルデータファイルを Orchestration プロジェクトに追加してから、サンプルデータファイルを複合型の WSDL メッセージに追加できます。Process Developer で作成するすべてのプロセスにデフォルトのサンプルを選択できます。プロセスごとに、プロジェクトまたは **【インタフェース】** ビューに追加したサンプルを使用することに加えて（またはサンプルを使用する代わりに）、プロセス変数にサンプルデータを追加できます。

サンプルデータファイルをメッセージ変数に追加する手順

1. **【パーティシパント】** ビューまたは **【インタフェース】** ビューで、複合パートを含むメッセージを展開し、サンプル XML データファイルを追加するメッセージ部分を選択します。
2. 複合パートをダブルクリックして、**【プロジェクト】** ダイアログを開きます。
3. ファイルのソースプロジェクトの場所を選択します。
4. ファイルを参照して選択し、**【OK】** をクリックします。

最初に追加したファイルがデフォルトのサンプルとして設定されますが、ファイルを追加して、そのうちの 1 つをデフォルトとして選択することもできます。

デフォルトのサンプルデータファイルの選択

【パーティシパント】 ビューまたは **【インタフェース】** ビューで、メッセージ部分ごとに複数のサンプルデータファイルを追加できます。プロセス変数にロードされるデフォルトファイルとして、1 つのサンプルファイルを選択できます。

【パーティシパント】 ビューまたは **【インタフェース】** ビューで、WSDL メッセージを展開します。デフォルトのサンプルデータファイルを選択するには、サンプルファイルを右クリックして **【デフォルトのサンプル】** を選択します。

サンプルデータファイルの XML 構造の表示

【インタフェース】 ビューで WSDL メッセージに追加されたサンプルデータファイルの XML テキストを表示するには、データファイル名を右クリックして **【開く】** を選択します。

サンプルデータファイルの削除

【パーティシパント】 ビューまたは **【インタフェース】** ビューの WSDL メッセージからサンプルデータファイルを削除するには、データファイルを右クリックして **【サンプルの削除】** を選択します。

デフォルトファイルとして設定されているファイルを削除すると、リストの最初にあるファイルがデフォルトになります。

プロセス変数のサンプルデータの選択

このリストは、選択したメッセージパートのインタフェースにユーザー自身または別のユーザーが追加したすべてのサンプルデータファイルで構成されます。

プロセス変数のサンプルデータの選択

【サンプルのロード】 ダイアログには、表示されたプロセス変数のメッセージパートのサンプルデータ値を含む XML ファイルのリストが表示されます。

このリストは、選択したメッセージパートの「パーティシパント」ビューまたは「インタフェース」ビューに追加したすべてのサンプルデータファイルで構成されます。詳細については、「[サンプルデータファイルの生成](#)」(ページ 114)および「[WSDL メッセージへのサンプルデータファイルの追加](#)」(ページ 115)を参照してください。

表示される各ファイルのパス名によって、使用するファイルを識別できます。

リストからサンプルファイルを選択します。変数のデータカラムにデータ値が表示されます。

WSDL コンポーネントの使用箇所の検索

Process Developer では、「使用した場所の検索」オプションを使用して WSDL コンポーネントを検索できます。

「使用した場所の検索」は、Process Developer プロジェクトファイル内の WSDL コンポーネント参照をすばやく見つけるために役立ちます。

Process Developer で BPEL プロセスを構築する際、ユーザーは WSDL ファイルのコンポーネントを参照します。特定の WSDL コンポーネントを検索することで、そのコンポーネントを参照している BPEL ファイルを見つけることができます。この検索により、参照を検索して更新することができるため、WSDL ファイルが変更された場合に役立ちます。

使用した場所を検索する手順

1. WSDL コンポーネントを選択し、マウスを右クリックして、「**使用した場所の検索**」を選択します。
2. 検索の範囲を選択します:
 - **ワークスペース:** 「ワークスペース」フォルダ内のすべてのプロジェクトファイル
 - **ワーキングセット:** 名前付きの、カスタマイズされたファイルとフォルダのグループ
3. 検索条件に一致するプロジェクトファイルのリストが「**検索**」ビューに表示されます。このビューが表示されていない場合（これは通常、Process Developer ウィンドウの右下隅にあるタブのグループにあります）、「**ウィンドウ**」>「**ビューの表示**」メニューから選択します。
4. ファイルをダブルクリックして開きます。

「**検索**」ビューでは、再検索、以前の検索の使用、検索結果内の検索など、より多くの検索機能を実行できます。詳細を表示するには、「**検索**」ビューで F1 キーを押してください。

プロセス検索の使用

プロセス検索は、ActiveVOS プロジェクトファイルで参照されている WSDL コンポーネントを見つける場合に役立ちます。

Process Developer では、BPEL ファイル内の WSDL コンポーネントの使用について検索できます。

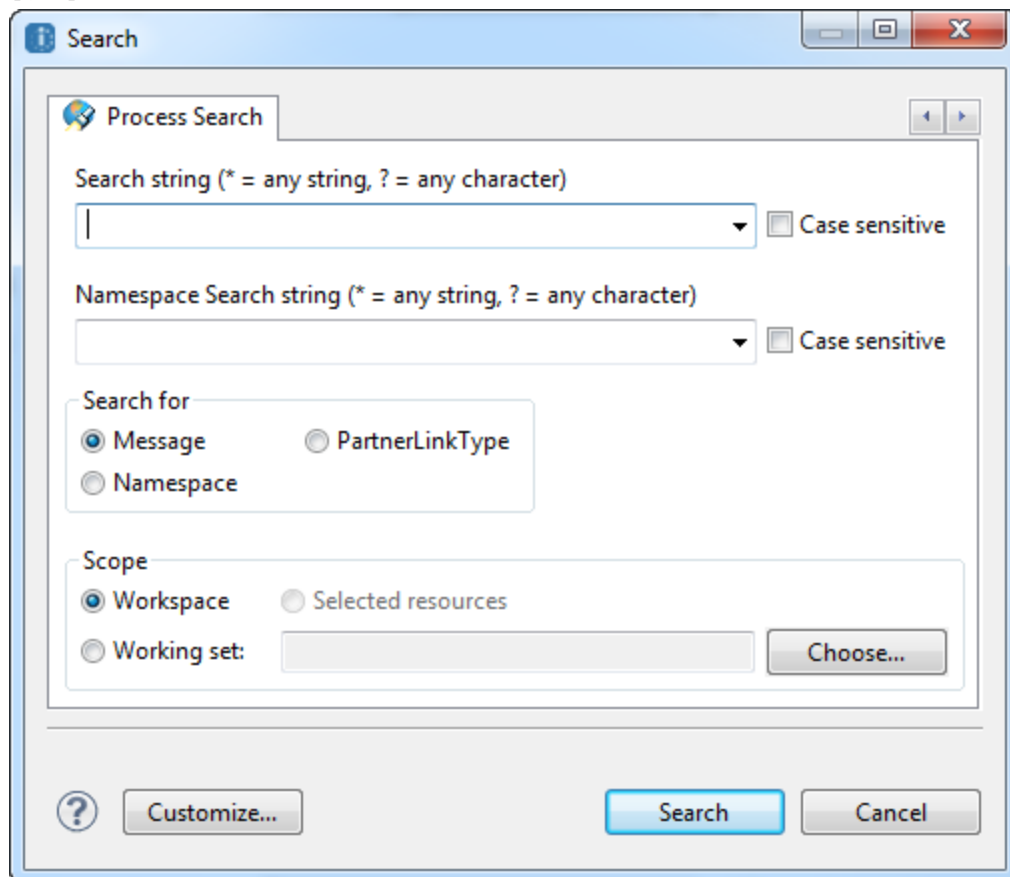
次の 2 種類の検索を実行できます。

- **【検索】** ダイアログの [プロセス検索] タブを使用して、文字列またはパターンを検索します。
- **【使用した場所の検索】** オプションを使用すると WSDL コンポーネントをすばやく検索できます。詳細については、[「WSDL コンポーネントの使用箇所の検索」 \(ページ 116\)](#)を参照してください。

プロセス検索は、BPEL ファイルで参照されている WSDL コンポーネントを見つける場合に役立ちます。コンポーネントの完全な名前がわからない場合は、ワイルドカードを使用して検索できます。

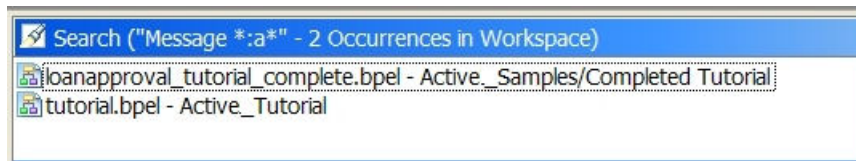
ワークスペース全体またはワーキングセットでプロセス検索を実行する手順

1. **【検索】** メニューから [プロセス検索] を選択します。



2. **【検索】** パネルで、WSDL コンポーネントのタイプを選択します。
 - メッセージ
 - パートナーリンクタイプ
 - 名前空間

3. メッセージまたはパートナーリンクタイプの検索については、次の手順を実行します。
 - 必要に応じて、オプションでワイルドカードを使用して検索文字列を入力します。
 - 必要に応じて、名前空間情報を入力して検索をさらに絞り込みます。
4. 名前空間検索の場合は、必要に応じて、オプションでワイルドカードを使用して検索文字列を入力します。
5. 検索の範囲を選択します：
 - ワークスペース、つまり、ワークスペースの外部からインポートされたものではないプロジェクトエクスプローラ内のすべての BPEL ファイル。
 - ワーキングセット。これは、プロジェクトファイルとフォルダの名前付きのカスタマイズ済みのグループです。
 - 選択したリソースの使用に関する詳細については、次の手順を参照してください。
6. **【検索】** を選択します。
7. プロジェクトファイルのリストが **【検索結果】** ビューに表示されます。**【検索】** ビューが表示されていない場合は、**【ウィンドウ】 > 【ビューの表示】** メニューから選択します。
次の例では、検索のタイトルバーに示されているように、「ワークスペース全体のすべての名前空間内にある文字列「a」で始まるメッセージ」が検索条件に含まれています。

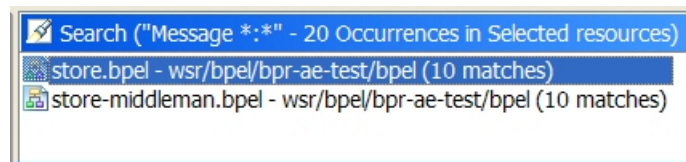


8. ファイルをダブルクリックして開きます。

選択したリソースでプロセス検索を実行する手順

1. **【プロジェクトエクスプローラ】** ビューを表示します。
2. 1つ以上のファイルを選択します。**【検索】** ダイアログを開くと、**【リソースの選択】** オプションが選択済みになっています。
3. 手順 1 から開始して、上記の手順を実行します。

次の例では、**【検索】** ダイアログを開く前に、プロジェクトエクスプローラで 2 つのファイルが選択されています。検索条件は、すべての名前空間のすべてのメッセージに対するものです。



検証エラーメッセージダイアログ

検出されるエラーは、追加する XML ファイル内の 1 つ以上の欠落、あるいは無効な宣言または無効な定義が原因で発生します。

追加しようとしているファイルは、選択した WSDL ファイルで宣言されたスキーマに照らし合わせて検証され、1 つ以上のエラーが見つかりました。エラーは、ファイル内の 1 つ以上の無効なファイル、あるいは欠落した宣言または欠落した定義が原因で発生します。

エラーが重大でない場合は、サンプルファイルを追加して、後でエラーを修正できます。エラーの説明を確認するには、「**エラーを表示**」をクリックしてください。

Process Developer で XML ファイルを解析できない場合、サンプルファイルを追加することはできません。エラーはエラーログで確認できます。エラーログが表示されない場合は、**[ウィンドウ] > [ビューの表示] > [エラーログ]** を選択します。

サンプルデータファイルのエラーの表示

宣言されたスキーマに従って、サンプルデータファイルの 1 つ以上の XML 要素が無効になりました。

[エラーの表示] ダイアログには、サンプルデータファイルの XML テキストと、ファイルで見つかった検証エラーが表示されます。

選択した WSDL ファイルで宣言されたスキーマに対するファイルの検証後にエラーが生成されました。

ダイアログの下半分にあるエラーメッセージをクリックして、検証エラーを含むファイル内の行を強調表示します。

サンプルファイルを追加して、後でエラーを修正できます。

第 9 章

BPEL プロセス

BPEL プロセスを作成するには、以下のものがが必要です。

- 自動化されたデータ交換に関するアイデア。
- 交換するデータを持つビジネスパートナー。
- データの受信方法、割り当て時のデータのマッピング方法、操作の呼び出し方法、およびビジネスパートナーへの返信方法を説明するフローチャートタイプのプラン。
- プロセスによって参照されるメッセージ、操作、およびプロパティを記述した WSDL ファイル。
- 存在するプロセス内、またはプロセス用に作成するプロセス内の各パートナーの WSDL 定義。

BPEL プロセスのプランニングに関するトピックは次のとおりです。

- トップダウンまたはボトムアップのプロセス設計の使用
- 効率的な設計のための WSDL 参照の使用
- WSDL、スキーマ、およびその他のリソースのインポート
- 拡張機能の宣言
- 拡張要素と属性の宣言
- BPEL プロセスの構造とプロパティの理解
- BPEL プロセスのライフサイクルの理解
- 実行可能なプロセスと抽象プロセスの作成
- 別の BPEL プロセスのサービスとしての BPEL プロセスの作成
- メッセージ交換の宣言

トップダウンまたはボトムアップのプロセス設計の使用

Process Developer を使用して、トップダウン、ボトムアップ、またはこれらを組み合わせて用いてプロセスを設計できます。

トップダウンを用いる場合は、アクティビティをキャンバスにドロップダウンし、それらの間にリンクを作成することで、プロセスをスケッチします。次に、アクティビティを実装にバインドする情報を追加し、意思決定を追加して、スケッチから作業プロセスを組み立てます。

ボトムアップを用いる場合は、プロセスの作成を開始する場合に利用可能な実装の定義を使用します。ボトムアップを用いる場合には、完全かつ有効な Web サービス記述言語（WSDL）ファイルが必要です。詳細については「[効率的な設計のための WSDL 参照の使用](#)」（[ページ 121](#)）を参照してください。

トップダウンとボトムアップの組み合わせを用いる場合は、プロジェクトにすでにインポートされた WSDL ファイルを使用してプロセスを開始できます。追加の WSDL ファイルが必要な場合は、インタフェースウィザードを使用して追加します。詳細については「[新しいインタフェースの作成](#)」(ページ 102)を参照してください。

効率的な設計のための WSDL 参照の使用

BPEL プロセスは、Web サービス記述言語 (WSDL) ファイルの定義を使用します。WSDL ファイルには、プロセスアクティビティを定義するために必要な名前空間、パートナーリンクタイプ、操作、およびメッセージが含まれ、有効かつ実行可能な BPEL 定義を作成するには WSDL ファイルが必要です。

WSDL ファイルは Process Developer で生成できます。または、プロセス定義の作成を開始する前に Process Developer の「プロジェクトエクスプローラ」ビューに追加することもできます。

- WSDL ファイルは、次のような複数の重要な生産性機能を提供します。
- WSDL 操作を Process Editor にドロップすることで、Web サービスのインタラクティブアクティビティ (Receive、Receive/Reply、Invoke、OnMessage、または OnEvent) を自動的に作成できます。
- BPEL プロセスに必要な WSDL 拡張機能 (パートナーリンクタイプ、プロパティ、プロパティエイリアス) が存在しない場合は、必要な拡張機能を自動的に作成することができます。
- WSDL 定義はプロセス全体で使用できます。

以下のトピックを参照

- [第 8 章, 「インタフェース、サービス参照、ローカル WSDL」](#) (ページ 99)
- [「BPEL 用の WSDL 拡張機能の作成」](#) (ページ 121)
- [「Process Editor への操作のドロップによるプロセスの開始」](#) (ページ 121)

BPEL 用の WSDL 拡張機能の作成

BPEL プロセスは、BPEL に固有の WSDL 拡張機能に依存しています。Process Developer では、WSDL ファイルにこれらの拡張機能が存在しない場合、こうした拡張機能を自動的に作成できます。この拡張機能は、パートナーリンクタイプ、プロパティ、およびプロパティエイリアスです。

- パートナーリンクタイプによって、サービスのロールとサービスが提供するポートタイプを定義します。詳細については、「[パートナーリンクタイプ](#)」を参照してください。
- プロパティとプロパティエイリアスは、プロセス内のメッセージのグループを相互に関連付ける場合に重要となり、任意のプロセス変数にも使用できます。詳細については、「[変数プロパティとプロパティエイリアスの追加](#)」を参照してください。

Process Editor への操作のドロップによるプロセスの開始

Process Developer の主要な生産性機能を使用して、BPEL プロセスの設計を開始できます。次の開始点のいずれかを使用して、Process Editor キャンバスへのドラッグが可能な WSDL 操作を公開し、アクティビティを自動的に作成します。

- プロセスで使用される Web サービスとパートナーを追加して、BPEL 拡張機能を自動的に生成します。[第 10 章, 「パーティシパント」](#) (ページ 138)を参照してください。
- 既存の WSDL を使用して、BPEL 拡張機能を作成します。「[サービス参照のインポート](#)」(ページ 101)を参照してください。

WSDL、スキーマ、およびその他のリソースのインポート

ターゲット名前空間を含むリソースファイルの場所を定義します。

BPEL プロセスには、WSDL とスキーマの場所を指す名前空間が含まれています。1 つの名前空間には多数の WSDL またはスキーマファイルを関連付けることができるため、リソースをインポートすることで Process Developer で正しいファイルが識別されます。

式の作成に使用する XSL スタイルシートやリソースカタログのカスタム関数で使用する XML ファイルなど、BPEL プロセスに必要な他のリソースをインポートできます。

インポートは手動または自動で追加できます。

以下のトピックを参照してください:

- [「WSDL およびスキーマの場所の自動インポート」 \(ページ 122\)](#)
- [「WSDL、スキーマ、およびその他のリソースの手動でのインポート」 \(ページ 122\)](#)
- [「インポートの更新」 \(ページ 123\)](#)
- [「インポートの削除」 \(ページ 124\)](#)

WSDL およびスキーマの場所の自動インポート

WSDL ファイルとスキーマファイルは多くの場所に配置することができ、同じ名前空間を共有します。WSDL またはスキーマの場所を指定すると、Process Developer では名前空間宣言が BPEL プロセスに追加され、その名前空間が正しい参照を指していることが確認されます。

Process Developer は、WSDL ファイルをプロジェクトエクスプローラに追加し、既存の操作または新しい操作から開始して受信アクティビティあるいは呼び出しアクティビティを作成する際に、WSDL の場所と名前空間の宣言を処理します。インタフェースの利便性に関する基本情報については、[第 10 章, 「パーティシパント」 \(ページ 138\)](#) および [第 8 章, 「インタフェース、サービス参照、ローカル WSDL」 \(ページ 99\)](#) を参照してください。

[「WSDL インタフェースから開始するアクティビティの作成」 \(ページ 160\)](#) も参照してください。

WSDL、スキーマ、およびその他のリソースの手動でのインポート

ターゲット名前空間を含むリソースファイルの場所を定義します。サンプルデータ、WSDL、XSL ファイルなどのワークスペースリソースを選択します。

ヒント: WSDL ファイルをプロジェクトエクスプローラに追加する前に Process Developer で BPEL プロセスをインポートして開いた場合、WSDL が不明であるため、Process Developer ではポートタイプへの未解決の参照が表示されます。

WSDL を追加する場合はパーティシパントを使用することをお勧めします。詳細については、[「\[\[パーティシパント\] ビューの使用」 \(ページ 139\)](#) を参照してください。

WSDL の場所と名前空間を手動で追加できます。また、次のようにスキーマやその他のリソースを追加することもできます。

1. [アウトライン] ビューから、[インポート] を選択します。
2. マウスを右クリックして、[追加] > [宣言] > [インポート] を選択します。
3. [インポートタイプ] リストから、タイプを選択します。

4. XSL を選択すると、ファイルの URI タイプが追加されます。このタイプは、ファイルで使用するエンコード言語を次のように識別します。
<http://www.w3.org/1999/XSL/Transform>

5. [場所] グループで、次のいずれかを実行します。

- **[プロジェクト]** を選択し、ファイルを含むプロジェクトエクスプローラのフォルダを参照します。

- URL を選択し、次のようにファイルの Web の場所を入力します。

<http://www.tempuri.org/myFile.wsdl>

または

<http://www.tempuri.com/service.asmx?WSDL>

プロセスをデプロイして実行する場合は、この URL が使用可能である必要があることに注意してください。

WSDL またはスキーマファイルのターゲット名前空間が表示されることに注意してください。パス名または URL を [WSDL (またはスキーマ) ファイルの場所] フィールドに貼り付けると、Tab キーを押してターゲットの名前空間を表示できます。完了後に、**[OK]** をクリックします。

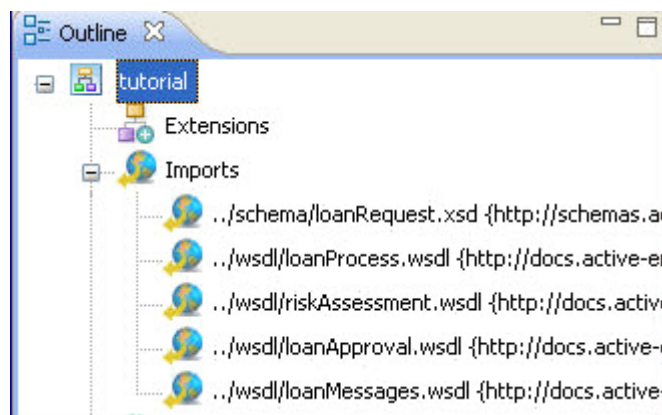
6. [アウトライン] ビューで [インポート] ノードを右クリックし、**[インポートの更新]** を選択します。

インポートを変更するには、インポートをダブルクリックするか、[プロパティ] ビューで [場所] の横にある [ダイアログ (...)] ボタン をクリックします。

Process Developer でターゲット名前空間が BPEL プロセスに追加され、名前空間がデフォルトのプレフィックスに関連付けられます。また、WSDL で定義されたポートタイプ、操作、およびパートナーリンクタイプをプロセスで使用できます。

次の図は、Process Developer で WSDL の場所と名前空間がどのように表示されるかを示しています。

Process Developer では、デフォルト名 ns1 で新しい名前空間プレフィックスが追加されることに注意してください。詳細については、[「名前空間のプレフィックスと宣言」](#) (ページ 124) を参照してください。



[「インポートの更新」](#) (ページ 123) も参照してください。

インポートの更新

[アウトライン] ビューの [インポート] ノードには、プロセスで参照されているすべての WSDL、スキーマ、およびその他のリソースの場所が表示されます。インポートを変更した場合、または新しいリソースを作成した場合は、インポートを更新して参照を更新できます。

インタフェースを更新しても、インポート情報は更新されません。

ヒント: WSDL ファイルをプロジェクトエクスプローラに追加する前に Process Developer で BPEL プロセスをインポートして開くと、WSDL が不明であるため、Process Developer はポートタイプ、操作、および変数への参照が未解決であると報告します。これらのエラーは次のように修正できます。

1. [「ローカル WSDL のインポート」 \(ページ 100\)](#)、[「新しいインタフェースの作成」 \(ページ 102\)](#)、または [「サービス参照のインポート」 \(ページ 101\)](#)に記載されているように、WSDL ファイルをプロジェクトエクスプローラに追加します
2. [アウトライン] ビューで [インポート] を右クリックし、[インポートの更新] を選択します。

インポートの削除

この BPEL プロセスからインポートを削除します。関連する名前空間のプレフィックスおよび URI は削除されません。必要に応じて、名前空間を手動で削除します。

インポートされた WSDL、スキーマ、またはその他のリソースの場所を削除するには、[アウトライン] ビューでインポートを右クリックし、[削除] を選択します。メニューバーから、[プロセス] > [削除] > [インポート] を選択することもできます。

関連する名前空間のプレフィックスおよび URI は削除されません。必要に応じて、名前空間を手動で削除します。

名前空間のプレフィックスと宣言

BPEL プロセスには、WSDL の場所を指す名前空間を含める必要があります。WSDL の場所を追加するには、[「WSDL、スキーマ、およびその他のリソースのインポート」 \(ページ 122\)](#)に記載されているように、インポート機能を使用します。

インポートの追加後、名前空間のデフォルトのプレフィックスの名前を次のように変更します。

1. [アウトライン] ビューで、[名前空間] を選択します。
2. [プロパティ] ビューで、名前空間のショートカット定義として使用するプレフィックスを入力します。表示された URI 参照にプレフィックスがマッピングされます。

インポートを削除する場合は、インポートに関連付けられている名前空間を手動で削除する必要があります。

名前空間の削除

この BPEL プロセスから名前空間を削除します。

[アウトライン] ビューで [名前空間] を右クリックして [削除] を選択すると、名前空間を削除できます。メニューバーから、[プロセス] > [削除] > [名前空間] を選択することもできます。[タスク] ビューに示されているように、名前空間内に変数およびその他の参照が存在する場合は、未解決の参照になります。

インポートを削除した場合でも、関連する名前空間のプレフィックスおよび URI は削除されません。必要に応じて、名前空間を手動で削除します。

拡張機能の宣言

拡張宣言を使用すると、BPEL プロセスに新しい構成を追加できます。Process Developer ではこの宣言が認識されないため、無視されます。

拡張を宣言することにより、BPEL 言語の定義を拡張できます。拡張宣言を使用すると、WS-BPEL 2.0 仕様の範囲外の新しいアクティビティタイプや新しい処理命令などの構造を追加できます。

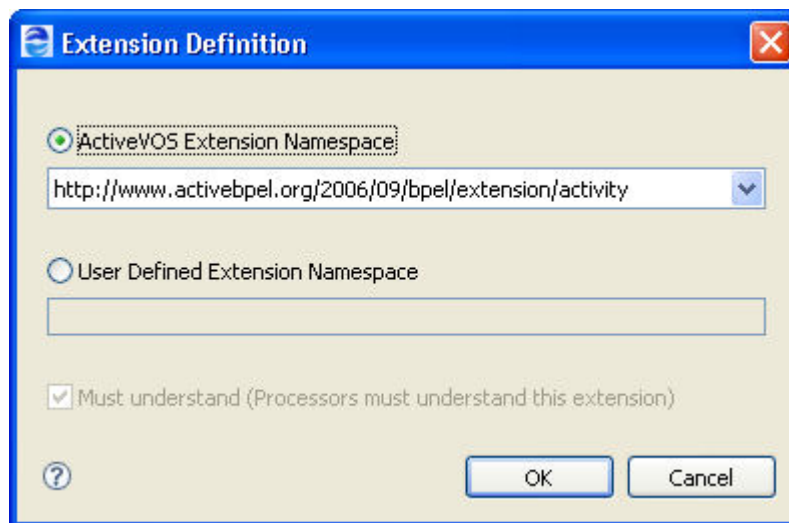
Process Developer とプロセスサーバーは拡張機能を使用します。Process Developer の拡張機能は次のとおりです。

- [コピー先] ターゲットの XPath を自動的に作成し、bpel:selectionFailure フォールトを無効にするためのクエリ処理（「*Process Developer の XPath 作成拡張機能の使用*」および「*Process Developer の無効な選択エラーのフォールト拡張機能の使用*」を参照）。
- アクティビティの一時停止、中断、継続（「*BPEL アクティビティの概要*」を参照）。
- プロセスレベルの補償/終了（「*別の BPEL プロセスのサービスとしての BPEL プロセスの作成*」を参照）。
- 暗黙的なスコープ変数。
- ループバックリンク（「*リンクの使用*」を参照）。

これらの拡張機能のいずれかを使用すると、拡張機能の名前空間が [アウトライン] ビューに自動的に追加されます。ただし、必要に応じて、拡張機能を使用する前に、拡張機能の名前空間を手動で追加することができます。

独自の拡張定義を追加する手順

1. [アウトライン] ビューから [拡張] を選択し、右クリックして [追加] > [宣言] > [拡張] を選択します。
2. [拡張定義] ダイアログで、[ユーザー定義の拡張名前空間] を選択します。
3. BPEL 拡張を参照する名前空間を入力します。
4. 必要に応じて、[認識必須] をオンにします。このオプションを選択してプロセスで拡張機能を使用すると、プロセスの保存時に Process Developer でエラーが表示されることに注意してください。プロセスをシミュレートまたはデプロイすることはできません。[認識必須] を選択しない場合、拡張機能はプロセスサーバーで無視されます。



Process Developer の XPath 作成拡張機能の使用

Process Developer の XPath 作成拡張機能を使用すると、実行中のプロセスで、複合的な変数に存在しないノードのロケーションパスを自動的に作成できます。この拡張機能は、プロセスごとに使用するか、すべてのプロセスに含めるように設定することができます。拡張機能を使用すると、拡張名前空間とプロセス属性が次のようにプロセスに追加されます。

```
<bpel:process ...  
  xmlns:ext=  
    "http://www.activebpel.org/2006/09/bpel/extension/query_handling" ...  
  ext:createTargetXPath="yes" ...>  
<bpel:extensions>
```

```

    <bpel:extension mustUnderstand="yes" namespace=
      "http://www.activebpel.org/2006/09/bpel/
      extension/query_handling"/>
  </bpel:extensions>
  ...

```

プロセス拡張は、Process Developer でのシミュレーション中に有効になり、プロセスサーバーでの実行が有効になります。

個々のプロセスに対してこの拡張機能を有効化または無効化する手順

1. プロセスの [プロパティ] ビューで、[すべて] タブを選択します。
2. [XPath の作成] 行で、選択矢印から [はい] または [いいえ] を選択します。

拡張機能が [アウトライン] ビューに追加され、その要素と属性が BPEL ソースコードに追加されます。

Process Developer の [選択の無効化エラー] 拡張機能の使用

Process Developer の [選択の無効化エラー] 拡張機能を使用すると、割り当てコピー操作の From 句の XPath クエリで、空のノードセットが返されるようになります。この機能を使用した場合、割り当てのターゲットノードが削除されます。この拡張機能は、プロセスごとに使用するか、すべてのプロセスに含めるように設定することができます。拡張機能を使用すると、拡張名前空間とプロセス属性が次のようにプロセスに追加されます。

```

<bpel:process ...
  xmlns:ext=
    "http://www.activebpel.org/2006/09/bpel/extension/query_handling" ...
  ext:disableSelectionFailure="yes" ...>
  <bpel:extensions>
    <bpel:extension mustUnderstand="yes" namespace=
      "http://www.activebpel.org/2006/09/bpel/
      extension/query_handling"/>
  </bpel:extensions>
  ...

```

プロセス拡張は、Process Developer でのシミュレーション中に有効になり、プロセスサーバーでの実行が有効になります。

個々のプロセスに対してこの拡張機能を有効化または無効化する手順

1. プロセスの [プロパティ] ビューで、[すべて] タブを選択します。
2. [選択の無効化エラー] 行で、選択矢印から [はい] または [いいえ] を選択します。

拡張機能が [アウトライン] ビューに追加され、その要素と属性が BPEL ソースコードに追加されます。

拡張要素と属性の宣言

拡張要素宣言を使用すると、BPEL 構成に新しい属性を追加できます。Process Developer ではこの宣言が認識されないため、無視されます。

WS-BPEL 2.0 仕様では、拡張要素や BPEL 構成の拡張属性など、いくつかの方法で基本的な BPEL 言語を拡張できます。

例えば、仕様で定義されていない要素と属性を含むプロセスを開いた場合、その拡張機能の詳細は Process Developer によって保持されます。

拡張機能の詳細で [認識必須] 属性が [はい] に設定されている場合、Process Developer はプロセスの実行方法を認識しないため、プロセスをシミュレートまたはリモートデバッグすることはできません。この場合、Process Developer は不明な拡張子をエラーとして宣言します。

[認識必須] 属性が [いいえ] に設定されている場合、プロセスをシミュレートおよびデバッグする際に拡張機能は無視されます。

拡張要素と属性は、任意の WS-BPEL 2.0 プロセスに追加できます。ただし、[認識必須] 属性を有効にすると、プロセスをシミュレートしたり、プロセスサーバーにデプロイしたりすることはできません。

拡張属性を追加または変更する手順

1. [アウトライン] ビューで、拡張属性を使用する BPEL 構成を選択します。プロセス要素、パートナーリンク、アクティビティ、相関セット、フォールトハンドラ、イベントハンドラなど、ほぼすべての構成には、[プロパティ] ビューに [拡張属性] プロパティが含まれていることに注意してください。
2. [プロパティ] ビューで **[詳細プロパティを表示]** ツールバーのボタンをクリックして、すべてのプロパティを表示します。
3. [拡張属性] 行の最後にある [ダイアログ (...)] ボタン] を選択します。
4. インポートしたプロセスに拡張属性が含まれている場合、それらの属性が **[拡張属性]** ダイアログに表示されます。次のいずれかを実行します。
 - リストから属性を選択し、**[編集]** を選択します。
 - **[追加]** を選択して、新しい属性を作成します。
5. **[拡張属性]** ダイアログで、次の手順を実行します。
 - 名前空間を入力するか、既存の拡張名前空間のリストから選択します。
 - 属性 [名前] を追加または編集します。
 - 属性 [値] を追加または編集します。
 - 新しい拡張名前空間である場合は、[この拡張機能を宣言] チェックボックスをオンにします。
 - プロセスが正常に実行されるように BPEL エンジンでの属性値の認識を有効にする場合は、**[認識必須]** をオンにします。Process Developer エンジンでは属性の定義または値を認識せず、プロセスをシミュレートしたり、プロセスサーバーにデプロイしたりできないことに注意してください。
 - プロセスが正常に実行されるように BPEL エンジンでの属性値の認識を有効にする場合は、**[認識必須]** をオンにします。Process Developer エンジンでは属性の定義または値を認識せず、プロセスをシミュレートしたり、プロセスサーバーにデプロイしたりできないことに注意してください。

次の図は、例を示しています。

Extension Attribute

Edit extension attribute

Namespace: http://tempuri.org

Name: seqNum

Value: 12

Extension Status

This extension is not declared in the process.

☒ Declare this extension

☒ Must understand (Processors must understand this extension)

OK Cancel

6. **[OK]** をクリックすると、属性が **[拡張属性]** ダイアログに表示され、リストから属性を編集および削除できるようになることに注意してください。

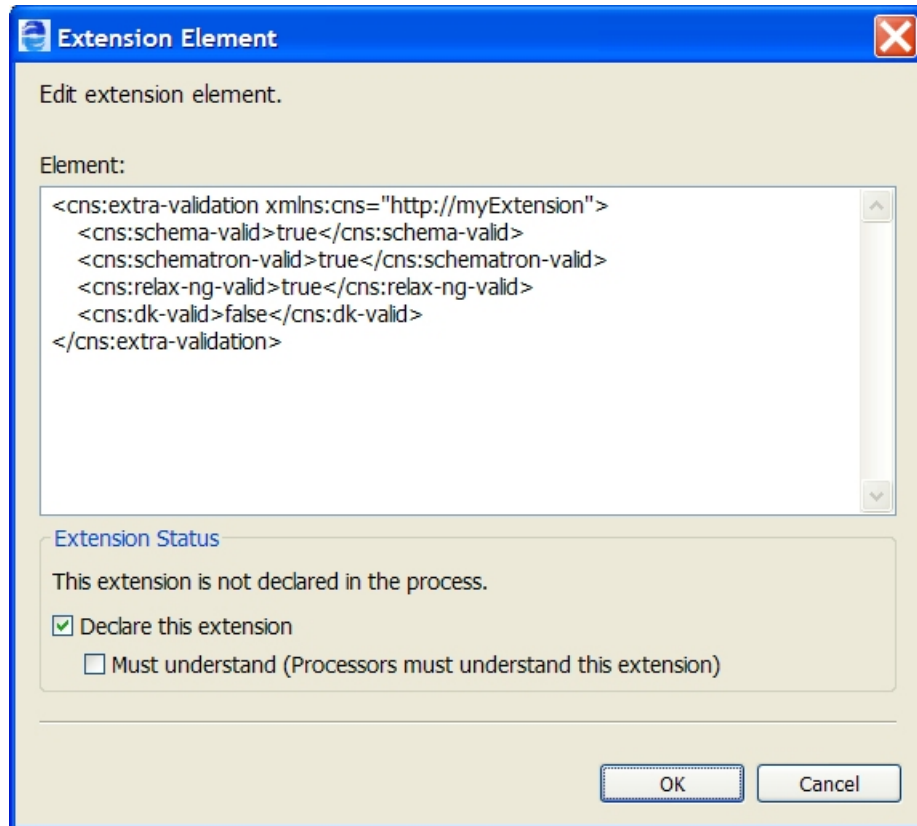
ヒント:

- 拡張属性を追加すると、Process Developer によって名前空間にプレフィックスが追加されます。必要に応じて、[アウトライン] ビューから [名前空間] を選択し、プレフィックスの名前をよりわかりやすいものに変更します。
- 拡張属性は、BPEL 構成の [プロパティ] ビューに、作成した順序で表示されます。
- [認識必須] 属性をリセットするには、[アウトライン] ビューから [拡張名前空間宣言] を選択し、[プロパティ] ビューで値を変更します。

拡張要素を追加または変更する手順

1. Process Editor または [アウトライン] ビューで、拡張要素を使用する BPEL 構成を選択します。プロセス要素、パートナーリンク、アクティビティ、相関セット、フォールトハンドラ、イベントハンドラなど、ほぼすべての構成には、[プロパティ] ビューに [拡張要素] プロパティが含まれていることに注意してください。
2. [プロパティ] ビューで **[詳細プロパティを表示]** ツールバーのボタンをクリックして、すべてのプロパティを表示します。
3. *[拡張要素]* 行の最後にある **[ダイアログ (...)]** を選択します。
4. インポートしたプロセスに拡張要素が含まれている場合、それらの属性が **[拡張要素]** ダイアログに表示されます。次のいずれかを実行します。
 - リストから要素を選択し、**[編集]** を選択します。
 - **[追加]** を選択して、新しい要素を作成します。
5. **[拡張要素]** ダイアログで、次の手順を実行します。
 - [要素] テキストボックスに完全修飾の有効な XML 構文を入力します。Process Developer で XML の検証が完了すると、**[OK]** ボタンが有効になります。
 - 新しい拡張名前空間である場合は、[この拡張機能を宣言] チェックボックスをオンにします。

- プロセスが実行されるように BPEL エンジンでの要素値の認識を有効にする場合は、**「認識必須」** をオンにします。Process Developer エン진은要素の定義または値を認識せず、プロセスをシミュレートしたり、プロセスサーバーにデプロイしたりできないことに注意してください。
次の図に例を示します。



6. **「OK」** をクリックすると、要素が **「拡張要素」** ダイアログに表示され、リストを再編成してリストから要素を削除できるようになることに注意してください。

ヒント:

- 拡張要素を実行する順序を反映させるには、要素のエントリを再編成します
- 「認識必須」** 属性をリセットするには、**「アウトライン」** ビューから **「拡張名前空間宣言」** を選択し、**「プロパティ」** ビューで値を変更します。

BPEL プロセスの構造とプロパティの理解

BPEL プロセスには、次の構造部分があります。

- [「プロセス要素とプロパティ」 \(ページ 130\)](#)
- [「変数」 \(ページ 133\)](#)
- [「アクティビティ」 \(ページ 132\)](#)
- [「フォールトハンドラ」 \(ページ 133\)](#)
- [「補償ハンドラ」 \(ページ 133\)](#)

- [「メッセージ交換の宣言」 \(ページ 136\)](#)
- [「WSDL、スキーマ、およびその他のリソースのインポート」 \(ページ 122\)](#)
- [「拡張機能の宣言」 \(ページ 124\)](#)
- [第 24 章, 「相関」 \(ページ 367\)](#)

BPEL プロセスは、[「Process Developer Process Editor の使用」 \(ページ 73\)](#)に記載されているように、Process Editor でグラフィカルに表されます。さらに、[「BPEL XML ソースと暗黙的に追加されたアクティビティ」 \(ページ 132\)](#)に記載されているように、BPEL ソースコードを表示することができます。

プロセス要素とプロパティ

Process Developer で新しい BPEL ファイルを作成し、Process Editor キャンバスにアクティビティを追加してファイルを保存すると、Process Developer では基礎となる XML ソースファイルが作成されます。XML ソースは<process>要素で始まります。

実行可能なプロセス要素の XML 構文は次のようになります。

```
<process name="NCName" targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
```

Process Developer 拡張機能を使用する場合は、[「拡張機能の宣言」 \(ページ 124\)](#)に記載されているように、他の名前空間とプロセス属性を追加できます。

Process Editor キャンバスの空白部分をクリックして、プロセス全体にフォーカスを合わせます。[プロパティ] ビューで、必要に応じてプロセスのプロパティを設定します。

プロパティ	説明	デフォルト値
プレフィックス付きソースの生成	BPEL XML ソースコード要素が<bpel:...>プレフィックス付きで生成されることを指定します。このプロパティを有効にすると、属性 xmlns:bpel が NULL にならないようにすることができます。	はい
書き込みポートタイプ	Receive、Reply、onEvent、onMessage、および Invoke のアクティビティ定義にポートタイプ属性を追加します。	いいえ
抽象プロセス	プロセスが抽象プロセスであるか実行可能なプロセスであるかを指定します。詳細については、「 実行可能プロセスと抽象プロセスの作成 」を参照してください。この設定は、すべての新規プロセスに対する設定として指定できます。「 Process Developer の設定 」を参照してください。	×
抽象プロセスプロファイル	WS-BPEL 2.0 仕様で参照されるデフォルトプロファイルのリスト	http://docs.oasis-open.org/wsbpel/2.0/process/abstract/simple-template/2006/08
BPEL 名前空間	実行可能なプロセスまたは抽象プロセスの BPEL 言語の名前空間を指定します。デフォルトは実行可能なプロセス用です。	http://docs.oasis-open.org/wsbpel/2.0/process/executable

プロパティ	説明	デフォルト値
コメント	<process>要素に HTML タグ付きの注釈を追加するためのオプションのプロパティ。アクティビティ、リンク、またはコンテナにコメントを追加することもできます。	なし
XPath の作成	このプロパティは、WS-BPEL 2.0 の Process Developer 拡張機能です（「 <i>拡張機能の宣言</i> 」に記載）。	はい
選択の無効化エラー	このプロパティは、WS-BPEL 2.0 の Process Developer 拡張機能です（「 <i>拡張機能の宣言</i> 」に記載）。	はい
マニュアル	<process>要素に注釈を追加するためのオプションのプロパティ。アクティビティ、リンク、またはコンテナにドキュメントを追加することもできます。「 <i>プロセスへのドキュメントの追加</i> 」を参照してください。	なし
標準フォールトで終了	このプロパティを「はい」に設定した場合、bpel:joinFailure 以外の WS-BPEL 標準フォールトが発生すると、終了アクティビティに到達した場合と同様にプロセスが即座に終了します。このプロパティを「いいえ」に設定した場合、プロセスはフォールトハンドラを使用して標準フォールトを処理します。 キャッチされないフォールトが発生した場合は、プロセスが中断されます。この動作は、「標準フォールトで終了」の設定よりも優先されます。	×
Expression Language	変数やその他の式の作成に使用する言語を指定します。デフォルトは XPath 1.0 ですが XQuery 1.0 および JavaScript 1.5 のサポートも組み込まれており、また、他の言語も追加できます。	urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0
拡張	Process Developer で認識されない WS-BPEL 2.0 仕様の要素および属性の拡張機能がプロセスに存在することを示します。拡張機能は、パートナーリンクやアクティビティといった他の多くの BPEL 構造にも存在する可能性があります。詳細については、「 <i>拡張要素と属性の宣言</i> 」を参照してください。	なし
リンクは遷移です	デフォルトでは、完了したアクティビティへのリンクが許可されています。詳細については、「 <i>リンクに対する Process Developer 拡張機能</i> 」を参照してください。	はい
メッセージ交換	Receive または onMessage の属性として選択可能なプロセスまたはスコープのプロパティとそれに一致する Reply	(なし)
プロセスレベルの補償/終了	これは、WS-BPEL 2.0 への Process Developer の拡張機能で、プラットフォーム固有の方法でプロセス全体を補償および終了できるかどうかを指定します。プロセスインスタンスは、通常の完了後に補償されます。詳細については、「 <i>別の BPEL プロセスのサービスとしての BPEL プロセスの作成</i> 」を参照してください。 これははいに設定した場合、Process Editor に新しいタブが表示されます。詳細については、「 <i>Process Editor の [補正] タブと [終了ハンドラ] タブ</i> 」を参照してください。	×

プロパティ	説明	デフォルト値
プロセス名	BPEL ファイル名とは異なるプロセス名を指定できます。これは、同じプロセスに異なるバージョンまたは更新されたバージョンを作成する場合に役立ちます。バージョン情報を使用して BPEL ファイルに意味のある名前を付けることができますが、内部プロセス名をそのまま使用することもできます。BPEL ファイル名ではなく、プロセス名のプロパティがサーバーエンジンで認識されます。	BPEL ファイルの名前
クエリ言語	XML クエリ言語を指定します。デフォルトは XPath1.0 ですが、新たな標準が使用可能な場合は、それらを利用できます。	urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0
結合の失敗の非表示	結合条件はすべてのアクティビティタイプのプロパティで、入力リンクに関するステータスが表示されます。結合の失敗は、リンクの実行が失敗したことを示します。プロセスレベルで、この属性によってプロセス全体で結合の失敗を非表示にするかどうかを指定します。アクティビティレベルでの上書きも可能で、すべての新規プロセスの設定として指定できます。「 <i>Process Developer の設定</i> 」を参照してください。	はい
ターゲット名前空間	必要な WSDL 情報を参照する XML 名前空間。これは必須のプロパティです。	なし
イメージの生成	「 <i>デプロイメントイメージの生成</i> 」を参照してください	自動

アクティビティ

アクティビティとは、Process Developer で設計したフロー図の順序で実行される処理ステップです。

Process Developer には、次のような複数のタイプのアクティビティがあります。

- データの交換および割り当て、実行ステップの定義を行うための基本的なアクティビティ
- アクティビティを構造化し、フォールトと補償を処理するためのコンテナ
- イベントハンドラ

詳細については、[第 11 章, 「BPMN タスクおよびイベント」 \(ページ 153\)](#)を参照してください。

BPEL XML ソースと暗黙的に追加されたアクティビティ

Process Developer は、Process Editor に追加する BPEL 構成ごとに BPEL XML ソースコードを生成します。ソースを表示するためにファイルを保存する必要はありません。Process Editor の [ソース] タブを選択すると、ソースコードを表示または編集できます。

プロセスレベルのアクティビティを Flow コンテナにラップしない場合、およびシーケンスや他のコンテナが存在しない場合、Process Developer によって、そのアクティビティが Flow コンテナに自動的にラップされることがあります。Process Developer により最上位レベルのコンテナが追加された場合、そのコンテナは ProcessEditor キャンバスまたは [アウトライン] ビューには表示されません。

補償ハンドラ

補償は、特にフォールトが発生した場合に、正常に完了したアクティビティを元に戻すか、代替となる手段を提供するプロセスです。詳細については、[第 27 章, 「補償」 \(ページ 400\)](#)を参照してください。

式言語

プロセスの [プロパティ] ビューで、ツールバーの [詳細] ボタンをクリックして、式言語のプロパティを表示します。[ダイアログ (...)] をクリックして、[式言語] ダイアログを開きます。

設定済みの言語ボックスから言語を選択します。

すべての式ビルダーが、選択した言語を使用するように再構成されます。

すべてのプロセスのデフォルト言語に対して設定を行うことができます。詳細については、「*Process Developer の設定*」を参照してください。

XQuery 1.0 および XQuery 3.0 がサポートされています。

フォールトハンドラ

パートナーリンク

BPEL プロセスは、プロセスとサービス間のインタラクションのフローを表します。それぞれのインタラクションは、フローのそのステップにおいてプロセスとサービスが果たすロール、およびそれらのロールのユーザーが操作可能なデータを表します。

ロールを定義するために使用される構成は次のとおりです。

- パートナーリンクタイプ
- パートナーリンク

これらのロールは、Process Developer で定義および追加できます。詳細については、[「パートナーリンク」 \(ページ 149\)](#)を参照してください。

変数

BPEL プロセスは、XML 変数を介してデータを受信、操作、および送信します。例えば、プロセスは、パイヤーからの発注書メッセージを受信し、そのメッセージを入力 XML 変数に組み込んで、そこから別の操作にメッセージをコピーします。

変数は、次のいずれかの方法で定義します。

- WSDL メッセージタイプ
- XML スキーマタイプ
- XML スキーマ要素

変数を使用することで、プロセスは、交換されたメッセージに基づいて、状態データ（タイムアウトなど）およびプロセス履歴を維持します。

BPEL プロセスのライフサイクルの理解

BPEL プロセスインスタンスには、開始、中間、および終了があり、これらはすべてアクティビティによって定義されます。プロセスは、次のいずれかのアクティビティで開始されます。

- Receive アクティビティ
- Pick アクティビティ (onMessage イベントを使用)
- 1 つ以上の Receive または Pick アクティビティを含む Flow あるいは Sequence アクティビティ

Receive または Pick では、*[インスタンスの作成]* プロパティが *[はい]* に設定されている必要があります。プロセスの中間は、順番に実行されるアクティビティまたは同時に実行されるアクティビティで構成されます。

プロセスインスタンスは、次のいずれかの方法で終了します。

- プロセスが完了したことを定義するアクティビティ
- プロセススコープに到達したフォールト、およびプロセスの終了

実行可能なプロセスと抽象的なプロセス

Process Developer では、次の 2 種類のビジネスプロセスを構築できます。

- *抽象プロセス*は、実行時にプロセスがどのような形を取るかをビジネスパートナーに伝えるためのビジネスプロトコル記述ドキュメントです。実行中のプロセスに必要なすべてのアクションとデータを実際に入力せずに、プロセスのステップの概要に関する情報を記述します。

抽象プロセスを使用することで、サービスから渡されるデータをどのように使用するかは公表せずに、パートナーと情報を共有することができます。これにより、パートナーは、プロセスで機能する WSDL ファイルの作成方法を理解し、プロセスの実行時に送信される情報を確認することができます。抽象プロセスは、交換するデータのタイプ、アクティビティ、制限時間、エラー処理、およびプロセスに対するその他の意味のある情報の概要を示します。

[「抽象プロセスの作成」 \(ページ 134\)](#) および [「抽象プロセスを使用するためのヒント」 \(ページ 135\)](#) を参照してください。

抽象プロセスの作成

抽象プロセスを作成する手順

1. **【ファイル】** > **【新規】** > **【BPEL プロセス】** を選択します。
2. プロジェクトフォルダを選択し、**【ファイル名】** フィールドに BPEL ファイルの名前を入力します。*.bpel という拡張子が自動的に追加されます。
3. このプロセスに設定可能なプロパティを表示するには、**【詳細】** を選択します。
4. **【抽象プロセスとして作成】** を選択します。
5. **【完了】** をクリックします。

プロセスの **【抽象プロセス】** プロパティを **【いいえ】** から **【はい】** に変更することで、実行可能なプロセスを抽象プロセスに変更することができます。また、その逆も可能です。

[「抽象プロセスを使用するためのヒント」 \(ページ 135\)](#) も参照してください。

抽象プロセスを使用するためのヒント

「[実行可能なプロセスと抽象的なプロセス](#)」(ページ 134)に記載されているように、実行をすぐに必要としないプロセスを作成できます。抽象プロセスを使用する場合は、次のヒントを考慮してください。

- 抽象プロセスの場合、名前空間は次のようになります: <http://docs.oasis-open.org/wsbpel/2.0/process/abstract>
- プロセスの [プロパティ] ビューで [抽象プロセスのプロファイル] を設定できます。リストされたデフォルトのプロファイルは、WS-BPEL 2.0 仕様で参照されています。
- 抽象プロセスには固有のアクティビティを使用できます。詳細については、「[あいまい](#)」(ページ 200)を参照してください。
- コピー操作では、*あいまいな from-spec* 割り当てを使用できます。詳細については、「[割り当て](#)」(ページ 173)を参照してください。
- プロセスの [抽象プロセス] プロパティを [はい] から [いいえ] に変更することで、抽象プロセスを実行可能プロセスにすることができます。プロセスにあいまいなアクティビティが含まれている場合は、別のアクティビティに置き換えるように警告が表示されます。
- 必要に応じて、WS-BPEL 2.0 仕様のガイダンスに従って、プロセス内の構文要素を非表示にすることもできます。以下の例を参照してください。

WS-BPEL XML フラグメントの例

例 1 - 変数の宣言

```
<variable name="commonRequestVar" element="##opaque" />
```

例 2 - Invoke アクティビティの使用

```
<invoke partnerLink="homeInfoVerifier"
  operation="##opaque" inputVariable="##opaque"
  ext:uniqueUserFriendlyName="request verification" />
```

例 3 - あいまいアクティビティの使用

```
<opaqueActivity template:createInstance="yes">
  <documentation>...</documentation>
</opaqueActivity>
```

別の BPEL プロセスのサービスとしての BPEL プロセスの作成

BPEL プロセスは、Invoke アクティビティを使用して、タスクを実行するための Web サービスを呼び出します。この目的のために BPEL プロセスを作成することができます。

次の異なる 2 つのプロセスの呼び出しのいずれかを選択できます。

- 通常のエンドポイント参照のように動作する標準 **プロセス**
プロセスデプロイメント記述子を作成する場合に、パートナーロールの静的エンドポイント参照として任意のプロセスを選択できます。これにより、標準の呼び出しレイヤーを介してディスパッチを行うための SOAP メッセージを作成するサーバーのオーバーヘッドを回避できます。代わりに、メッセージはエンジンに直接送信されます。
- プロセスレベルの補償と終了処理の対象となる呼び出しプロセスの **サブプロセス**
プロセスデプロイメント記述子を作成する場合に、パートナーロールの静的エンドポイント参照として任意のプロセス:サブプロセスを選択できます。これにより、プロセスレベルの補償と終了処理にサブプロセスを加えることができます。

サブプロセスの詳細

サブプロセスは、呼び出される親プロセスエンクロージングスコープのエラーおよび補償処理ルールの対象となります。サブプロセスは、Invoke アクティビティをエンクローズするスコープのフォールトハンドラまたは補償ハンドラのいずれかが実行されるたびに補償されます。この作業は暗黙的に実行されます。ただし、サブプロセスのプロセスレベルの補償と終了を有効にして、実行する補償と終了のタスクを選択できます。補償の説明については、「[補償](#)」を参照してください。

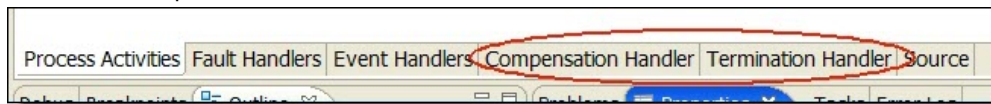
Invoke アクティビティのエンクロージングスコープは、サブプロセスが完了するまで完了しません。サブプロセスの完了時に補償可能である場合は、エンクロージングスコープによって、補償中にサブプロセスが補償されます。

サブプロセスがフォールトをスローすると、親プロセスの Invoke アクティビティがフォールトをスローします。

サービスとしてのプロセスの作成

BPEL プロセスをサービスとして作成するには、以下の手順を実行する必要があります。

1. BPEL プロセスの WSDL ファイルをプロジェクトエクスプローラに追加します。WSDL ファイルは、複数の方法で作成できます。「[インタフェース](#)、[サービス参照](#)、および[ローカル WSDL](#)」を参照してください。
2. サービスとして使用する BPEL プロセスを作成します。
3. 必要に応じて、プロセスプロパティ [プロセスレベルの補償/終了] を [はい] に設定します。Process Editor に 2 つのタブが追加され、それらに補償および終了アクティビティを追加できます。次の図は、Process Developer に追加されるタブを示しています。



4. プロセスデプロイメントウィザードで、他のプロセスの場合と同じように、[マイロールパートナー] リンクのサービス名を指定します。
5. 「[プロセスのデプロイ](#)」に記載されているように、ビジネスプロセスアーカイブ (.BPR) を作成します。
6. プロセスをサーバーにデプロイします。
7. Invoke アクティビティを使用して BPEL プロセスを呼び出すメインプロセスを作成します。
8. メインプロセスのプロセスデプロイメントウィザードで、パートナーロールのパートナーリンクに対して次の手順を実行します。
 - a. Invoke ハンドラとして、プロセスまたはサブプロセスを選択します。
 - b. エンドポイント参照として静的を選択します。
 - c. **【ダイアログ】** ボタンを選択して、サブプロセス情報を入力します。
9. メインプロセスをデプロイします。

メイン BPEL プロセスが実行されている場合に別の BPEL プロセスを呼び出すと、呼び出されたプロセスは、元々の状態情報とともにサブプロセスのステータスを持つようになります。メインプロセスは、サブプロセスが完了するまで待機してから、元々の実行を続行します。アクティブな状態で実行されているプロセスの詳細については、「[プロセスコンソールのヘルプ](#)」を参照してください。

メッセージ交換の宣言

[メッセージ交換の宣言] ダイアログを使用して、プロセス全体または現在のスコープのメッセージ交換の宣言を追加、編集、または削除します。

メッセージ交換値は、同期受信とその応答をバインドします。メッセージ交換値によって、受信と応答を明示的に一致させ、同じパートナーリンク、ポートタイプ、および操作で同期受信を同時に実行する際のあいまいさを排除します。

メッセージ交換は、Receive、onMessage、または onEvent とそれらのアクティビティに一致する応答の属性として選択可能なプロセスまたはスコープのプロパティです。このダイアログでメッセージ交換値を宣言します。次に、必要に応じて、適切な Receive/Reply ペア、onEvent/Reply ペア、または onMessage/Reply ペアの値を選択します。

この宣言を使用するタイミング

並列 forEach アクティビティのプロセスと子スコープは、デフォルトのメッセージ交換値を暗黙的に宣言します。明示的に宣言されたメッセージ交換値が Receive または Reply アクティビティに存在しない場合、Receive または Reply がネストされた場所に応じて、プロセスあるいは並列 forEach のスコープのいずれかによって提供されるデフォルト値を持ちます。ユーザーは、並列 forEach 内の Receive、Reply、および onMessages での明示的なメッセージ交換値を引き続き使用できます。

メッセージ交換値は、同じパートナーリンク、ポートタイプ、および操作を持つフロー内の複数の Receive/Reply、onEvent/Reply、または onMessage/Reply のペアでも役立ちます。ペアにメッセージ交換値を追加することで、bpel:conflictingRequest フォールトの受信を回避できます。

プロセスは、実行中に受信する同期メッセージ交換（Receive または onMessage）に応答するよう求められます。応答がない場合、プロセスフォールトには bpel:missingReply フォールトが発生します。メッセージ交換値を宣言する各スコープは、アクティブなメッセージ交換を追跡して、スコープが完了する前に応答が実行されるようにします。

新しいメッセージ交換値を追加した場合、デフォルト名は MessageExchange_1 です。必要に応じて、この値を編集できます。

内部スコープで宣言されたメッセージ交換が外部スコープまたはプロセスで宣言されたものと同じ値である場合、内部スコープの宣言によって外部スコープまたはプロセスの宣言が上書きされます。異なるスコープでの Receives、onEvents、onMessages、および Reply は、最も外側のスコープのメッセージ交換宣言を使用することによってのみ照合することができます。

第 10 章

パーティシパント

パーティシパントに関するトピックは次のとおりです

- パーティシパントについて
- パートナーリンクタイプとパートナーリンクについて
- パートナーリンクタイプ
- パートナーリンク

パーティシパントについて

パーティシパントとは、実行の過程でプロセスがインタラクションを行うユーザーとサービスです。パーティシパントとの通信は、常に Web サービスインタフェースを介して行われます。

パーティシパントは、インタフェースに提供する必要のある BPEL の詳細を単純化します。パーティシパントを作成することで、Process Developer が詳細を生成する場合に、BPEL の技術的な詳細をスキップできます。

パーティシパントのタイプは次のとおりです。

- **プロセスサービスコンシューマ**

それぞれのプロセスは、少なくとも 1 人のパーティシパント（プロセスを開始するサービス要求メッセージを送信するパーティシパント）とインタラクションを行います。このパーティシパントは、最初のプロセスサービスコンシューマです。プロセスが提供するサービスを使用する追加のパーティシパントがいる場合は、追加のコンシューマを作成することもできます。また、コンシューマはコールバックサービスを提供することもあります。これらによって、プロセスはコンシューマに操作の実行を要求できます。BPEL では、プロセスサービスコンシューマは、マイロールのプロパティが設定されているパートナーリンクと同等です。

- **パートナーサービスプロバイダ**

プロセスがプロセスの作業を行うためにサービスを呼び出す必要がある場合、プロセスはパートナーサービスプロバイダによって提供されるサービスを呼び出します。このプロセスは、パートナーサービスプロバイダがコールバックサービスを利用できるようにすることもあります。このようなコールバックのインタフェースは、そのインタフェースを使用できるパートナーサービスプロバイダにリストされます。BPEL では、このパーティシパントは、パートナーロールのプロパティが設定されたパートナーリンクに相当します。

- **ヒューマンタスクパーティシパント**（オンプレミスのみ）

プロセスで複数のステップを達成するためにユーザーが必要である場合、そのプロセスではユーザーアクティビティが使用されます。ヒューマンタスクパーティシパントは、プロセスのアクティビティの一部を完了するユーザーアクティビティを表します。それぞれのヒューマンタスクパーティシパントには、そのロールが機能するタスクが一覧表示されます。BPEL4People では、*論理ユーザーグループ*がヒューマンタスクパーティシパントごとに作成されます。

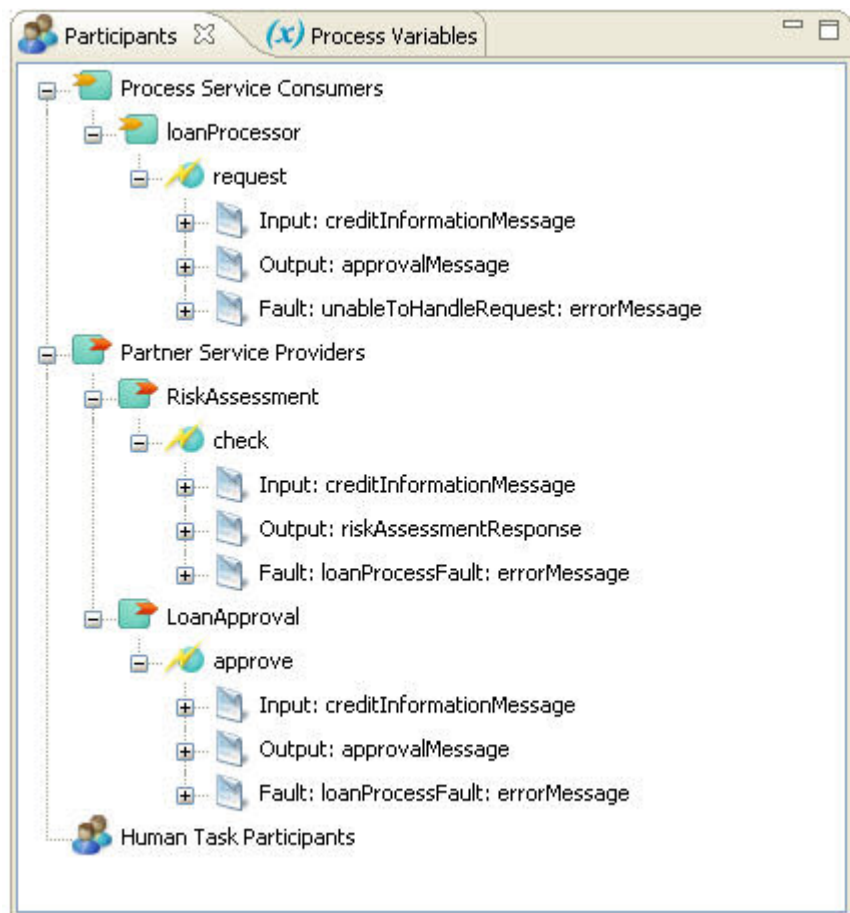
詳細については、[「\[パーティシパント\] ビューの使用」](#)（ページ 139）を参照してください。

HTML フォームの開発と組み合わせた [パーティシパント] ビューの使用に関する詳細については、[「フォームまたはタスクへの新しいサービス操作の追加」](#)（ページ 515）を参照してください。

[パーティシパント] ビューの使用

[パーティシパント] ビューは、BPEL プロセスを開発するための重要な出発ポイントです。このビューは、プロセスの主要な要素であるプロセスと、パートナーサービスが果たすロールおよび各ロールのインタフェースを提供するためのガイドとなります。さらに、インタフェースを定義した後に、その操作を Process Editor キャンバスにドラッグして、プロセスアクティビティを作成できます。例えば、Receive アクティビティは、プロセスサービスコンシューマに関連付けられたインタフェース操作から作成します。

次の図は、パーティシパントとそのインタフェースの例を示しています。



プロセスが開いていない場合、[パーティシパント] ビューは空です。ビューはプロセスベースです。つまり、フォーカスされているプロセスのパーティシパントのみが表示されます。

このビューでは、次のことを実行できます（オンプレミスのみ）。

- 新しいプロセスおよびサービスインタフェースを作成する。[「新しいプロセスサービスコンシューマインタフェースの作成」](#)（ページ 140）および [「新しいパートナーサービスインタフェースの作成」](#)（ページ 141）を参照してください。
- ユーザーアクティビティで使用するための新しいヒューマンタスクパーティシパントとそのタスクを作成する。
- 操作（またはタスク）を Process Editor にドラッグして、適切なアクティビティを作成する。[「\[パーティシパント\] ビューからの新しいアクティビティの作成」](#)（ページ 143）を参照してください。

新しいプロセスサービスコンシューマインタフェースの作成

プロセスサービスコンシューマは、Receive および/または Reply アクティビティの構築に使用するパーティシパント Web サービスです。このコンシューマは、名前とインタフェースで構成されます。

サービスコンシューマは段階的に作成できます。

- 名前を入力するか、デフォルト名の Process_Consumer を使用します。
- インタフェースリソースが利用可能な場合は、インタフェースを定義できます。利用できない場合は、後で [プロパティ] ビューで追加できます。
- パーティシパントの詳細の入力を完了すると、プロセスの Receive および Reply アクティビティに必要なほぼすべての要素を作成することができます。

新しいプロセスサービスコンシューマの作成

新しいプロセスサービスコンシューマを作成する方法を説明した一連の手順は次のとおりです。

1. Process Editor でプロセスが開いていることを確認します。
2. [パーティシパント] ビューで、[プロセスサービスコンシューマ] を右クリックし、[新規プロセスサービスコンシューマ] を選択します。
3. デフォルトのコンシューマ名は「Process_Consumer」です。必要に応じて、意味のある、よりわかりやすい名前に変更します。例えば、Web サービスでプロセスに対して在庫アイテムのチェックを要求するような場合には、サービスにコンシューマ「inventoryRequestor」という名前を付けます。
4. 必要に応じて **[OK]** をクリックして名前を受け入れ、[プロパティ] ビューでの詳細の入力は後で行います。準備ができてから、次のいずれかを実行します。
 - 詳細を追加する準備ができている場合は、インポート済みのインタフェースを選択するか、新しいインタフェースを作成します。
 - ワークスペースにインポート済みのインタフェースを選択するには、インタフェースツリーを使用します。このサービスコンシューマの WSDL 操作を選択します。

ツリーには、プロセスにインポートされた WSDL、現在のプロジェクトと参照されたプロジェクトの WSDL、WSDL を含む他のワークスペース Orchestration プロジェクト、およびシステムサービス WSDL の複数のカテゴリが表示されます。システムサービスの詳細については、[「システムサービスインタフェース」](#)（ページ 112）を参照してください。

新しいインタフェースを作成するには、**[インタフェースの生成]** を選択します。詳細については [「新しいインタフェースの作成」](#)（ページ 102）を参照してください。

プロセスサービスコンシューマの [プロパティ] ビュー

プロセスサービスコンシューマの [プロパティ] ビューには、パーティシパント名などの一部の詳細が表示されます。

また、選択したサービスインタフェースも表示されます。WSDL を表示するには、[サービスインタフェース] リンクを選択します。[「パーティシパントからのサービスインタフェースのクリア」 \(ページ 143\)](#)に記載されているように、このインタフェースをクリアすることができます。

このビューには、[「新しいコールバックインタフェースの作成」 \(ページ 142\)](#)に記載されているコールバックインタフェースを追加するためのプロパティも含まれています。

[パートナーリンク] タブには、このパーティシパント用に作成された BPEL 構成が表示されます。このパーティシパント用に自動的に作成された WSDL を表示するには、パートナーリンクタイプ（またはマイロールのリンク）を選択します。WSDL には、必要なパートナーリンクタイプの定義が含まれています。これは、プロジェクトの WSDL フォルダに表示されないプライベート WSDL として作成されます。ただし、WSDL は必要なプロジェクトリソースであるため、デプロイ中に Process Developer によってこのリソースがデプロイメントパッケージに含まれます。

[「\[パーティシパント\] ビューからの新しいアクティビティの作成」 \(ページ 143\)](#)も参照してください。

新しいパートナーサービスインタフェースの作成

パートナーサービスプロバイダは、Invoke アクティビティの構築に使用するパーティシパント Web サービスです。このコンシューマは、名前とインタフェースで構成されます。

パートナーサービスプロバイダは段階的に作成できます。

1. 名前を入力するか、デフォルト名のプロバイダを使用します。
2. インタフェースリソースが利用可能な場合は、インタフェースを選択できます。利用できない場合は、後で [プロパティ] ビューで追加できます

パーティシパントの詳細の入力を完了すると、プロセスの Invoke アクティビティに必要なほぼすべての要素が作成されます。

新しいパートナーサービスプロバイダの作成

新しいパートナーサービスプロバイダを作成する手順は次のとおりです。

1. Process Editor でプロセスが開いていることを確認します。
2. [パーティシパント] ビューで、[パートナーサービスプロバイダ] を右クリックし、[新規パートナーサービスプロバイダ] を選択します。
3. デフォルトのプロバイダ名は「Provider」です。必要に応じて、意味のある、よりわかりやすい名前に変更します。例えば、注文内のアイテムの価格を設定するために Web サービスを呼び出すような場合には、サービスプロバイダに *orderPricingSystem* という名前を付けます。
4. 詳細を入力する準備ができていない場合は、**[OK]** をクリックして名前を受け入れます。準備ができてから、次の段落に記載されているようにインタフェースを選択、インポート、または作成して、この手順を完了します。

実行する必要があるタスクは次のとおりです。

- ワークスペースにインポート済みのインタフェースを選択するには、インタフェースツリーを使用します。このサービスプロバイダの WSDL 操作を選択します。
 - 現在の BPEL プロセスと、他のパーティシパント（存在する場合）で現在使用されている WSDL インタフェース。

- プロジェクトにインポートされた WSDL、および参照されたプロジェクト。「[プロジェクト参照の使用](#)」を参照してください。
- WSDL を含む他のワークスペース Orchestration プロジェクト。
- システムサービス WSDL。システムサービスの詳細については、「[システムサービスインタフェース](#)」を参照してください。
- インタフェースをインポートするには、**[サービス参照のインポート]** を選択します。詳細については、「[サービス参照のインポート](#)」を参照してください。
- 新しいインタフェースを作成するには、**[インタフェースの生成]** を選択します。詳細については、「[新しいインタフェースの作成](#)」を参照してください。

パートナーサービスプロバイダの [プロパティ] ビュー

パートナーサービスプロバイダの [プロパティ] ビューには、パーティシパント名などの一部の詳細が表示されます。

また、選択したサービスインタフェースも表示されます。WSDL を表示するには、[サービスインタフェース] リンクを選択します。「[パーティシパントからのサービスインタフェースのクリア](#)」 (ページ 143) に記載されているように、このインタフェースをクリアすることができます。

このビューには、「[新しいコールバックインタフェースの作成](#)」 (ページ 142) に記載されているコールバックインタフェースを追加するためのプロパティも含まれています。

[パートナーリンク] タブには、このパーティシパント用に作成された BPEL 構成が表示されます。このパーティシパント用に自動的に作成された WSDL を表示するには、パートナーリンクタイプ (またはパートナーロールのリンク) を選択します。WSDL には、必要なパートナーリンクタイプの定義が含まれており、これはプロジェクトの WSDL フォルダに表示されないプライベート WSDL として作成されます。ただし、WSDL は必要なプロジェクトリソースであるため、Process Developer ではこのリソースがデプロイメントパッケージに含まれます。

[「\[パーティシパント\] ビューからの新しいアクティビティの作成」](#) (ページ 143) も参照してください。

新しいコールバックインタフェースの作成

パーティシパントがコールバックするポートタイプと操作を選択または作成します。

Receive、Reply、および Invoke アクティビティで表されるプロセス内の各 Web サービスには、サービスインタフェースが必要です。インタフェースにより、サービスによって提供される操作を記述します。

コールバックインタフェースを追加することを選択できます。これにより、サービスプロバイダはサービスコンシューマで操作を呼び出すことができます。例えば、サービスインタフェースでは、プロセスが注文を送信し、パートナーが注文 ID で応答します。コールバックインタフェースでは、パートナーは注文出荷通知を送信できます。

コールバックインタフェースを追加すると、Process Developer はそのパーティシパントの標準 BPEL パートナーリンクタイプを生成し、それを WSDL に格納します。これにより、WSDL の将来のユーザーにもパーティシパントのリレーションが明確になり、サービスインタフェースを使用する新しいプロセスがコールバックインタフェースを自動的に確認できるようになります。

コールバックインタフェースの作成の詳細は、サービスインタフェースの作成と同様です。以下を参照してください。

- [「新しいプロセスサービスコンシューマインタフェースの作成」](#) (ページ 140)
- [「新しいパートナーサービスインタフェースの作成」](#) (ページ 141)

ヒント:

- Process Developer は、BPEL に必要な詳細、つまりパートナーリンクタイプ定義を新しい WSDL に追加します。[process].public.wsdl という名前のファイルに、プロセスのクライアントが表示する必要のある定義を追加します。プロセス専用の定義（つまり、パートナーサービスのパートナーリンクタイプ）を [process].public.wsdl という名前のファイルに配置します。
- コールバックのパーティシパントが、[パーティシパント] ビューに子パーティシパントとして表示されます。

[「\[パーティシパント\] ビューからの新しいアクティビティの作成」 \(ページ 143\)](#)も参照してください。

パーティシパントからのサービスインタフェースのクリア

パーティシパントからのサービスインタフェースを削除します。

[プロパティ] ビューから、パーティシパントからのサービスインタフェースをクリアできます。

- [パーティシパント] ビューまたは [アウトライン] ビューから、パーティシパントを選択します。
- [プロパティ] ビューで、[サービスインタフェース] 行の [ダイアログ] ボタンを選択します。
- [サービスインタフェース] ダイアログで、[インタフェース] 行の [クリア] ボタンを選択します。

[パーティシパント] ビューからの新しいアクティビティの作成

パーティシパントとパーティシパントインタフェースを作成した後に、操作またはタスクを Process Editor にドラッグすることで、アクティビティを作成できます。これらのアクティビティには、必要なパーティシパント名と操作が含まれています。その後、アクティビティの [プロパティ] ビューを使用して、データの割り当てを行うことができます。

次の表は、Process Editor キャンバスに操作をドロップした場合に、Process Developer によって作成されるアクティビティを示しています。

このパーティシパントインタフェースからの操作のドラッグ	作成されるアクティビティまたはアクティビティペア
プロセスサービスコンシューマの一方向の操作	Receive、onMessage (Pick を選択した場合)、または onEvent (イベントハンドラを選択した場合)
プロセスサービスコンシューマの双方向の操作	Receive-Reply、onMessage-Reply、onEvent-Reply
プロセスサービスコンシューマのコールバック	Invoke
パートナーサービスプロバイダ	Invoke
パートナーサービスプロバイダのコールバック	一方向および双方向の操作のプロセスサービスコンシューマと同一のアクティビティ

関連事項:

- [「新しいプロセスサービスコンシューマインタフェースの作成」 \(ページ 140\)](#)
- [「新しいパートナーサービスインタフェースの作成」 \(ページ 141\)](#)
- [「新しいコールバックインタフェースの作成」 \(ページ 142\)](#)

アクティビティの「プロパティ」ビューからの新しい変数の作成

Receive またはその他のアクティビティ用の新しいプロセス変数を作成します。

次のような方法で、Receive、onMessage、onEvent、Reply、または Invoke アクティビティに新しいプロセス変数を作成します。

1. アクティビティの「プロパティ」ビューを選択する。
2. アクティビティの「データ」、[入力]、または「出力」タブを開く。
3. 「割り当てタイプ」で、「単一の変数」を選択する。
4. 「変数」リストで、「新規変数」を選択します。

提案される変数名は、次のいずれかです。

- 単一パート要素の WSDL メッセージのスキーマ要素の修飾名
- 単一パート要素が文字列などの XSD タイプの場合のメッセージのローカル名
- マルチパート WSDL メッセージの場合のメッセージのローカル名

「[スコープ] フィールドで、プロセスまたはスコープを選択して、変数をグローバルまたはローカルにします。

変数タイプは、WSDL メッセージ定義から派生します。

パートナーリンクタイプとパートナーリンクについて

パートナー定義に含めるパートナーリンクを選択します。パートナーリンクは、1 つのパートナー定義でのみ使用できます。

パートナーリンクタイプとパートナーリンクを作成して使用する簡単な方法については、「パーティシパントについて」を参照してください。

BPEL プロセスは、プロセスとサービス間のインタラクションのフローを表します。それぞれのインタラクションは、フローのそのステップにおいてプロセスとサービスが果たすロール、およびそれらのロールのユーザーが操作可能なデータを表します。

BPEL は、インタラクションで使用されるロールと関係を識別するための構成を定義します。

この構成は、パートナーリンクとパートナーリンクタイプです。パートナーリンクによって、プロセスとサービスが果たすロールと、そのロールで操作できるデータを記述します。次の図に示すように、パートナーリンクはそのパートナーリンクによって定義されます。

BPEL プロセスでのパートナーの定義と使用法は、Web サービスの詳細を参照しません。BPEL プロセスは、さまざまな方法でデプロイできる再利用可能な定義であるため、定義が抽象的です。Web サービスのアドレス指定、セキュリティ、およびその他の詳細については、デプロイ時に処理します。プロセスをデプロイすると、BPEL プロセスインスタンスのパートナーリンク内のすべてのパートナーロールに一意的なエンドポイント参照が割り当てられます。



パートナーリンクタイプ

Receive、Reply、Invoke などの標準の BPEL アクティビティを作成します。パートナーリンクタイプの定義を新しい WSDL ファイルまたは既存の WSDL ファイルに追加します。

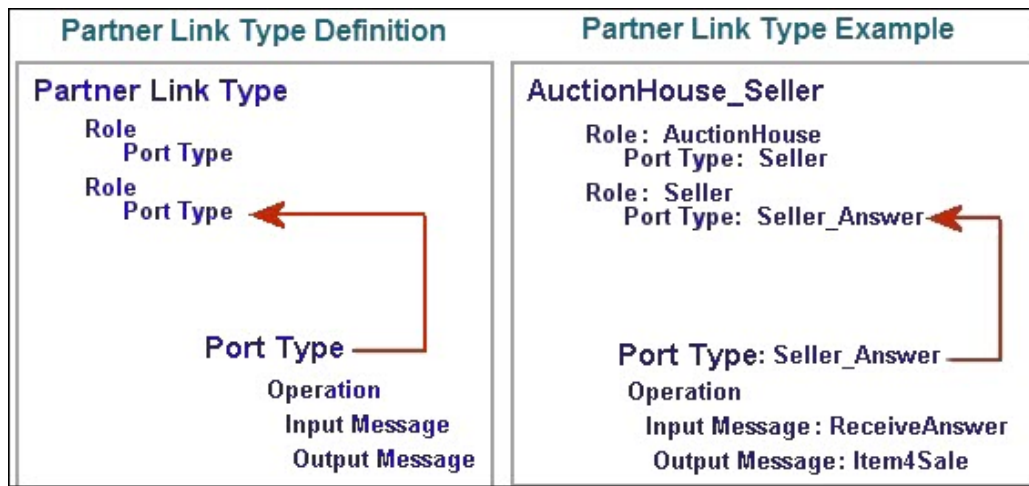
パートナーリンクタイプによって、2 つの WSDL サービスが実行しようとしているメッセージのやりとりの種類を記述します。パートナーリンクタイプにより、各サービスが果たすロールを定義し、やりとりに適したメッセージを受信するためにサービスが提供するポートタイプを指定することで、このやりとりの特徴を定義します。

パートナーリンクタイプとパートナーリンクを作成して使用する簡単な方法については、「パーティシパントについて」を参照してください。

パートナーリンクタイプには、1 つまたは 2 つのロールを含めることができます。

- パートナーリンクタイプに 2 つのロールが含まれている場合、それぞれのサービスは、指定したポートタイプを入力することでそのロールを実装します。2 つのロールを使用するということは、通信の過程で、呼び出し側サービスがターゲットサービスから何らかのタイプのコールバックを受信する必要があることを示します。例えば、サービスが一方向の操作で呼び出されたとします（入力メッセージのみ）。返信する準備が整うと、プロセスの Receive アクティビティにメッセージを返送します。
- パートナーリンクタイプにロールが 1 つしかない場合は、ロールに関する Web サービスの呼び出しに制限はありません。単一のロールで記述されたサービスは、コールバックを発行する必要なく、その操作を呼び出すだけで通信を完了します。

次の図は、2 つのロールを定義するパートナーリンクタイプの例を示しています。



例えば、AuctionHouse_Seller という名前のパートナーリンクタイプには、AuctionHouse と Seller という 2 つのロールが記述されています。AuctionHouse ロールは、販売する項目の入力メッセージを予期したポートタイプ Seller をサポートします。Sell ロールは、項目が販売されたかどうかに関する AuctionHouse からの入力メッセージを予期したポートタイプ Seller_Answer をサポートします。出力メッセージは、販売する項目の名前です。

この例では、ポートタイプは同じサービスからのものですが、ポートタイプは異なるサービスからのものである可能性があります。

パートナーリンクタイプは WSDL 拡張機能です。1 つまたは 2 つのロールが指定されます。ポートタイプは、同じ WSDL ファイルまたは異なる WSDL ファイルから取得できます。

パートナーリンクタイプの定義は、次のソースから取得されます。

- すでに定義された WSDL ファイル、または新しい定義を追加できる WSDL ファイルから。
- 2 つのポートタイプがあり、それぞれが異なるサービスからのものである場合は、独自の名前空間を持つ別の WSDL ファイルから。
- 新しい定義を作成して WSDL ファイルに追加できる、Process Developer の BPEL プロセスから。

次の方法で、新しいパートナーリンクタイプを WSDL に追加できます。

- プロジェクトエクスプローラの WSDL から
- サービス参照から
- [インタフェース] ビューの WSDL から

プロジェクトエクスプローラでの WSDL からの新しいパートナーリンクタイプの追加

次の手順に従って、プロジェクトエクスプローラの WSDL から新しいパートナーリンクタイプを追加します。

1. Orchestration プロジェクトに WSDL がインポート済みであることを確認します。通常、WSDL ファイルは標準の wsdl フォルダにインポートします。
2. WSDL を展開して、ポートタイプを表示します。
3. ポートタイプを右クリックし、*[PartnerLink タイプに追加]* を選択します。
4. 表示されるウィザードを完了します。詳細については、[「\[インタフェース\] ビューからの新しいパートナーリンクタイプの追加」 \(ページ 147\)](#)の手順 6 を参照してください。

サービス参照 WSDL を使用した新規 WSDL への新たなパートナーリンクタイプの追加

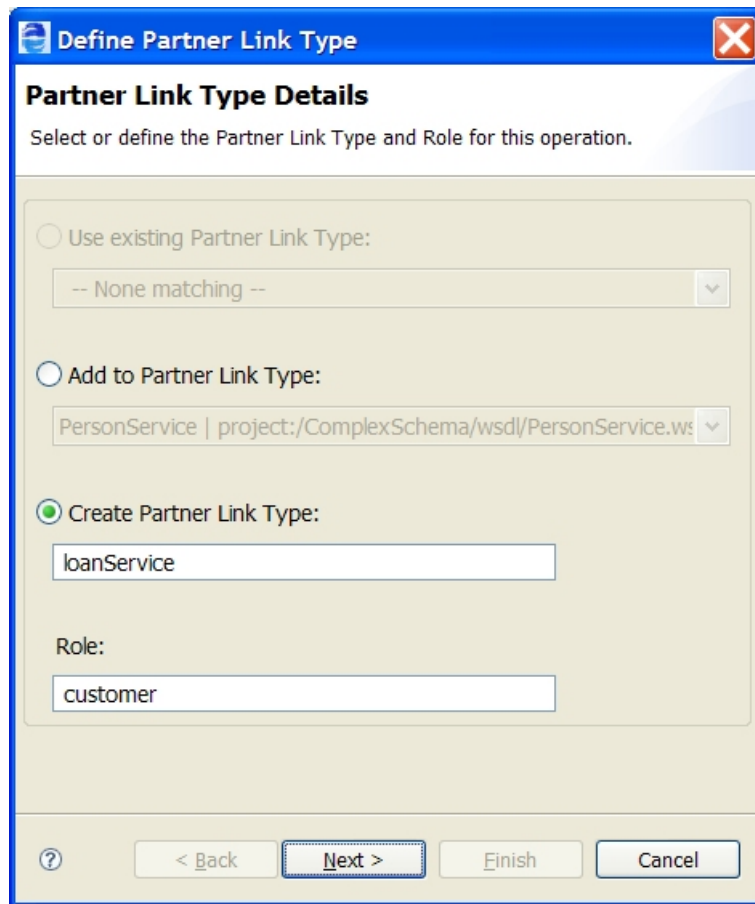
次の手順に従って、サービス参照 WSDL を使用して新規 WSDL に新たなパートナーリンクタイプを追加します。

1. Orchestration プロジェクトの [サービス参照] フォルダに WSDL がインポート済みであることを確認します。この WSDL は、サーバー上ですでに実行されているサービスへの参照です。
2. WSDL を展開して、ポートタイプを表示します。
3. ポートタイプを右クリックし、**[PartnerLink タイプに追加]** を選択します。
4. 表示されるウィザードを完了します。既存のリモート WSDL にはパートナーリンクタイプを追加できないことに注意してください。このウィザードを使用することで、サービス参照のインポートを行う新規 WSDL を作成できます。

[インタフェース] ビューからの新しいパートナーリンクタイプの追加

次の手順を使用して、[インタフェース] ビューから新しいパートナーリンクタイプを追加します。

1. [インタフェース] ビューを表示します。このビューは、デフォルトのパーспекティブの一部ではありません。[インタフェース] ビューを開くには、**[ウィンドウ] > [ビューの表示] > [インタフェース]** を選択します。
2. 参照する WSDL ファイルのポートタイプがリストに表示されていることを確認します。
3. **[ポートタイプ]** を展開して、使用する操作を表示しますが、関連付けられたパートナーリンクタイプはまだありません。
4. **[BPEL アクティビティ]** を選択する必要があります。
5. 図のように、**操作**を Process Editor キャンバスにドラッグして、パートナーリンクタイプウィザードを開きます。



Define Partner Link Type

Partner Link Type Details

Select or define the Partner Link Type and Role for this operation.

☐ Use existing Partner Link Type:

-- None matching --

☐ Add to Partner Link Type:

PersonService | project:/ComplexSchema/wsd/PersonService.ws

☒ Create Partner Link Type:

loanService

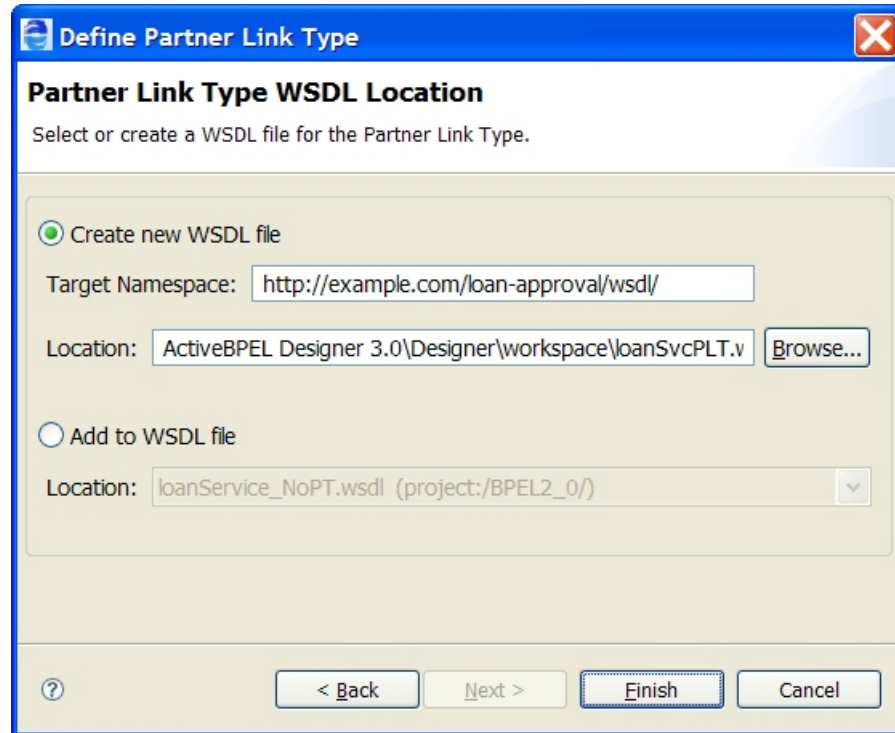
Role:

customer

? < Back Next > Finish Cancel

6. 次のいずれかを実行します。
 - *〔既存のパートナーリンクタイプを使用する〕* を選択してリストから選択し、**〔完了〕** をクリックします。
 - *〔パートナーリンクタイプに追加〕* から *〔パートナーリンクタイプの定義〕* を選択し、現在のパートナーリンクタイプに 2 番目のロール（関連付けられたポートタイプを含む）を追加します。パートナーリンクタイプには 2 つのロールがあります。これらのロールは、非同期で通信する 2 つの異なるサービスを表します。選択リストには、現在 1 つのロールだけを持つインタフェースのパートナーリンクタイプが表示されます。*〔ロール〕* テキストボックスに新しいロール名を入力し、**〔完了〕** をクリックします。
 - 新しいパートナーリンクタイプ名および *〔ロール〕* 名として *〔パートナーリンクタイプの作成〕* を選択し、**〔次へ〕** をクリックします。
7. 手順 5 で新しいパートナーリンクタイプを作成した場合は、作成したリンクタイプを既存の WSDL ファイルに追加するか、定義用の新しいファイルを作成できます。次のいずれかを実行します。
 - *〔新しい WSDL ファイルの作成〕* フィールドに URL を入力して、このパートナーリンクタイプの定義のすべての名前とメッセージタイプを修飾します。次に、次の図に示す例のように、*〔場所〕* フィールドに WSDL ファイル名とパス名を入力するか、**〔参照〕** をクリックして新しいファイル名と場所を作成します。**〔開く〕** ダイアログの *〔ファイル名〕* フィールドにファイル名を入力します。

- [WSDL ファイルに追加] から WSDL ファイルを選択して、新しいパートナーリンクタイプの定義を追加します。



8. [完了] をクリックします。

パートナーリンクタイプの定義を新しい WSDL に追加すると、Process Developer によって WSDL がプロジェクトエクスプローラと [アウトライン] ビューの [インポート] ノードに自動的に追加されます。

パートナーリンク

パートナーリンクタイプとパートナーリンクを作成して使用する簡単な方法については、主要な Process Developer の機能、[「パーティシパントについて」](#) (ページ 138) を参照してください。

パートナーリンクとは、2つのパートナー間で行われる通信のやりとりです。最も基本的な形式であるプロセスは、外部サービスのパートナーリンクとして外部サービスからの要求を受け取ります。パートナーリンクによって、特定のやりとりでプロセスが果たすロール（存在する場合）とパートナーサービスが果たすロール（存在する場合）を定義します。パートナーリンクは、グローバルに宣言することやスコープ内で宣言することができます。

パートナーリンクの定義は次のもので構成されます。

- 名前
- パートナーリンクタイプ
- 少なくとも 1 つのロールを持ち、2 つのロールを含めることができます
- プロセスロールである MyRole のロール名と PartnerRole のロール名
- パートナーロールの初期化と呼ばれるオプションのプロパティ

- [「プロセスへのコメントの追加」 \(ページ 79\)](#)および [「プロセスへのドキュメントの追加」 \(ページ 79\)](#)のオプションのプロパティ
- [「拡張要素と属性の宣言」 \(ページ 126\)](#)のオプションのプロパティ

BPEL プロセスでは、Web サービスのインタラクションアクティビティを定義する場合にパートナーリンクを使用します。例えば、Receive アクティビティによって、パートナーリンク、そのポートタイプ、および操作を指定します。ポートタイプは、関連付けられたパートナーリンクタイプから決定されます。これは、プロセスが、パートナーリンクで定義したパートナーのロールを果たすサービスから受信メッセージデータを受け入れることができることを意味します。

プロセスには、関与する特定の通信のやりとりごとに少なくとも 1 つのパートナーリンクが必要です。BPEL では、同じパートナーリンクタイプの複数のパートナーリンクをプロセス内に存在させることができます。

デフォルトでは、パートナーリンクはプロセスレベルで宣言されます。スコープレベルでパートナーリンクを宣言できます。パートナーリンク名は、プロセスまたはスコープで定義されたパートナーリンクのセット内で一意である必要があります。

パートナーロールの初期化プロパティは、プロセスのデプロイを行うユーザーへのヒントとして指定します。このプロパティは、プロセスの実行時の動作には影響を及ぼしません。プロセスにパートナーリンクの初期化ロジックが含まれている場合は、このプロパティを [いいえ] に設定する必要があります。このタイプの初期化は、Assign アクティビティを使用して実行します。デプロイメント時の静的エンドポイントまたは WS-Addressing などのエンドポイント参照スキームを介してパートナーリンクを初期化するデプロイメント環境がプロセスに必要な場合は、このプロパティを [はい] に設定する必要があります。このプロパティを設定しない場合、パートナーリンクが初期化される方法についてデプロイを行うユーザーに伝達される情報はありません。

プロセスへのパートナーリンクの自動的な追加

WSDL 操作を Process Editor キャンバスにドロップし、ウィザードを完了して Receive、Invoke、Reply、onMessage、または onEvent を作成すると、パートナーリンクがプロセスに自動的に追加されます。新しい操作を作成する必要がある場合は、操作ウィザードを使用します。これらの手順は、パートナーリンクを作成する場合に推奨される方法です。詳細については、[「WSDL インタフェースから開始するアクティビティの作成」 \(ページ 160\)](#)を参照してください。

プロセスへのパートナーリンクの手動での追加

1. [アウトライン] ビューで、[パーティシパントのパートナーリンク] を右クリックし、[プロセスサービスコンシューマの追加] または [パートナーサービスプロバイダ] を選択します。
2. 詳細については、[「新しいプロセスサービスコンシューマインタフェースの作成」 \(ページ 140\)](#)を参照してください。

XML 構文

```
<partnerLinks>
  <partnerLink name="NCName" partnerLinkType="QName"
    myRole="NCName"? partnerRole="NCName"?
    initializePartnerRole="yes|no"?>+
  </partnerLink>
</partnerLinks>
```

例

```
<partnerLink name="seller"
  partnerLinkType="AuctionHouse_SellerLT"
  myRole="AuctionHouse" partnerRole="Seller"
</partnerLink>
```

スコープされたパートナーリンクの使用

パートナーリンクは、プロセスレベルまたはスコープレベルで宣言できます。パートナーリンク名は、同一である場合でも一意であると見なされます。プロセスレベルでパートナーリンクを宣言すると、スコープレベルで同じ名前のパートナーリンクがある場合を除いて、それらをグローバルに使用できるようになります。スコープレベルのパートナーリンクは、宣言されたスコープでのみ使用できます。詳細については、[「Scope」 \(ページ 235\)](#)を参照してください。

スコープレベルでパートナーリンクを宣言すると、スコープの複数のインスタンスが同時に実行される並列 forEach アクティビティに使用できます。

スコープされたパートナーリンクは、コードのモジュール化およびコードの再利用にも使用できます。さまざまな BPEL プロセスでコードを再利用するための方法に関する詳細については、[「カスタムアクティビティの作成」 \(ページ 164\)](#)を参照してください。

パートナーリンクとエンドポイント参照

BPEL プロセスでは、ビジネスパートナー間のインタラクションをパートナーリンクによって定義します。この定義は抽象的であり、実際に実行されているサービスに関するアクセス情報は含まれません。実際のアクセス情報を省略する理由は、プロセス定義を多くのサービスで再利用できるようにするためです。

実際のサービスにはどのようにアクセスしますか? サービスは、デプロイメント時のプロセス外、または Assign アクティビティのプロセス内という 2 つの方法で識別されます。

プロセスをデプロイすると、パートナーリンクで定義した各ロールにエンドポイント参照が割り当てられます。エンドポイントは、特定のプロトコルとデータ形式を使用して Web サービスにアクセスするための特定の場所を示します。エンドポイント参照は、Web サービスエンドポイントにアクセスするために必要な情報を伝達します。デプロイメントでエンドポイント参照を使用することで、サービスパートナーの動的な選択が可能になります。

プロセス内で、Assign アクティビティ内でターゲットエンドポイント参照を設定することで、サービスを呼び出すことができます。詳細については、[「割り当て」 \(ページ 173\)](#)を参照してください。また、パートナーリンクは、プロセス内またはスコープアクティビティ内でも宣言できます。詳細については、[「Scope」 \(ページ 235\)](#)を参照してください。

抽象プロセスへのパートナーリンクのエクスポート (非推奨)

バージョン 9.0 以降、この機能は非推奨になりました。

サービスとのインタラクションを行う操作をパートナーが実装できるように、その操作を説明するプロセス定義をパートナーに提供できます。このプロセス定義は、プロセス図のパートナーリンクからエクスポートできます。

パートナーリンク情報を含む BPEL ファイルを作成する手順

1. Receive、Invoke、Pick アクティビティ、または onEvent ハンドラの onMessage ブランチなどの WSDL 操作に基づいて、少なくとも 1 つの完了済みのアクティビティを使用して BPEL プロセスを作成します。
2. [プロセス] メニューから、[抽象プロセスのエクスポート] を選択します。

3. [パートナーリンク] を選択します。
4. リストからパートナーリンクを選択します。
5. **[抽象プロセスのエクスポート]** ダイアログで、作成する BPEL ファイルの場所を選択します。
6. デフォルト名を使用するか、BPEL ファイルに新しい名前を入力し、必要に応じて [エクスポート時にファイルを開く] のチェックボックスをオンにします。
7. **[OK]** をクリックします。

Process Developer で Abstract Process 属性が [はい] に設定され、BPEL プロセスが作成されます。ファイルがプロジェクトエクスプローラに一覧表示されます。

第 11 章

BPMN タスクおよびイベント

アクティビティは、BPEL プロセス内のステップです。アクティビティによって、Web サービスとのインタラクションを記述したり、内部プロセス機能を実行したりできます。プロセスには、少なくとも 1 つのアクティビティを含める必要があります。

BPMN では、BPEL アクティビティとしてタスクおよびイベントを実装します。ただし、タスクとイベント（タスクが作業を実行し、イベントがトリガまたは結果であるもの）との間には相違があります。

- [「BPMN から BPEL へのタスクとイベントの実装」](#) (ページ 153)
- [「BPEL アクティビティの概要」](#) (ページ 155)
- [「アクティビティとそのプロパティの定義」](#) (ページ 156)
- [「カスタムアクティビティの作成」](#) (ページ 164)
- [「WSDL インタフェースから開始するアクティビティの作成」](#) (ページ 160)

BPMN から BPEL へのタスクとイベントの実装

多くの場合、BPMN 要素は、以下の表に示した 1 つ以上の BPEL アクティビティ（タスク、Throw イベント、および Catch イベントなどの BPMN タイプ）として実装できます。

追加の実装については、[「BPMN から BPEL へのゲートウェイの実装と制御フロー」](#) (ページ 217)を参照してください。

タスク

BPMN タスク	BPEL 実装
サービス	Invoke
ユーザー	ユーザーアクティビティ、タスクタイプ
スクリプト	Assign
ルール	Invoke
中断	中断
検証	検証
抽象型	Empty

BPMN タスク	BPEL 実装
手動	<ul style="list-style-type: none"> - Empty - ユーザーアクティビティ、タスクタイプ
送信	<ul style="list-style-type: none"> - Invoke - Reply - ユーザーアクティビティ、通知タイプ
Receive	Receive

Throw イベント

BPMN Throw イベント	BPEL 実装
なし/開始/終了	<ul style="list-style-type: none"> - Empty - Assign
メッセージ	<ul style="list-style-type: none"> - Reply - Invoke - ユーザーアクティビティ、通知タイプ
エラー	<ul style="list-style-type: none"> - Throw - Rethrow
補償	<ul style="list-style-type: none"> - Compensate - Compensate Scope
終了	<ul style="list-style-type: none"> - Exit - ブレーク（forEach で実装）

Catch イベント

任意の catch イベントをアクティビティの境界にドロップすることで境界イベントを作成することができます。または、スコープ（イベントサブプロセス）にドロップすることでハンドラを作成することができます。

BPMN Catch イベント	BPEL 実装
タイマー	<ul style="list-style-type: none"> - Wait - Pick またはイベントハンドラの OnAlarm
メッセージ	<ul style="list-style-type: none"> - Receive - Pick の OnMessage - イベントハンドラの OnEvent
エラー	<ul style="list-style-type: none"> - Catch - Catch All
補償	スコープ補償ハンドラ
キャンセル	スコープ終了ハンドラ

BPEL アクティビティの概要

デフォルト（BPMN）の Process Developer 編集スタイルでは、BPMN タイプ（アクティビティとイベント）に従って基本的なアクティビティが編成されます。BPEL ではアクティビティとイベントの区別がなく、どちらもアクティビティと見なされます。

次の表に、簡単な定義と詳細へのリンクを示します。

アクティビティ名	説明
invoke	Web サービスに操作を実行するように指示します
割り当て	プロセス変数とパートナーリンクエンドポイント参照のコピー元またはコピー先操作を作成することで、それらを実行します
空	実行時に何らかの操作が発生しないアクティビティ。アクティビティは必要であっても実際には操作を実行させる必要がない状況で役立ちます。例えば、Empty を Catch の子にすることでフォールトを抑制します。
サスペンド。	プロセスを中断します。Catch または catchAll イベントで、予期しないエラーをキャッチする場合に役立ちます。
検証	関連する XML および WSDL データ定義に照らし合わせて変数の値を検証します
あいまい	(Process Developer Classic のみ)。抽象プロセス用。実行可能なプロセスで使用されるアクティビティのプレースホルダとして機能します。
受信	サービスパートナーからのメッセージデータを受け入れます。必要に応じて、プロセスのインスタンスを作成してプロセスを開始します。
リプライ	一致する Receive アクティビティで識別されたパートナーに応答を送信します
シグナル	Signal は Throw に似ていますが、フォールトがないという点が異なります。Signal は、この情報を待機する Signal によって受信された情報をスローします。
Throw	フォールトを通知します。標準またはカスタムのフォールトを指定します
Rethrow	直ちにエンクローズするフォールトハンドラによって最初にキャッチされたフォールトを親スコープに渡します
終了	実行可能なプロセスを直ちに停止し、フォールトプロセスを発生させます。
待機	指定した時間、または期限に達するまでプロセスの実行を停止します
Compensate Scope	すでに正常に完了している指定された内部スコープで補償を開始します
補償	名前付きスコープで補償ハンドラを実行するか、スコープが指定されていない場合はデフォルトの補償を実行します
Break	一時的またはアクティビティごとに、スコープまたはループから抜け出します。通常は、次のアクティビティから処理が続行されます。

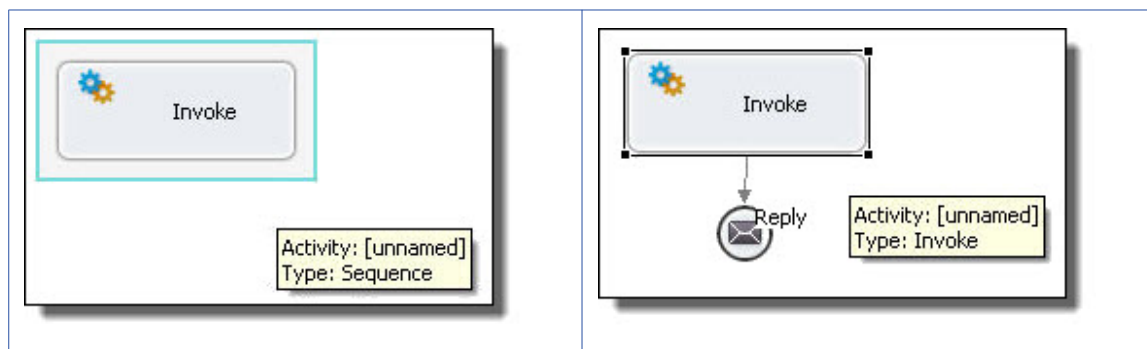
アクティビティ名	説明
続行	(BPEL 中心パレットのみ)。ループ (While、Repeat Until、または For Each) 内の次の反復を続行します。特定の条件が満たされた場合に通常の処理を続行するために必要なすべての条件を指定することなく処理を簡素化することができる便利なアクティビティです。
Start/End/None	上記の BPEL の Empty または Assign アクティビティと同等です。

構造化されたアクティビティについては、「活動を構造化するさまざまな方法」を参照してください。

アクティビティとそのプロパティの定義

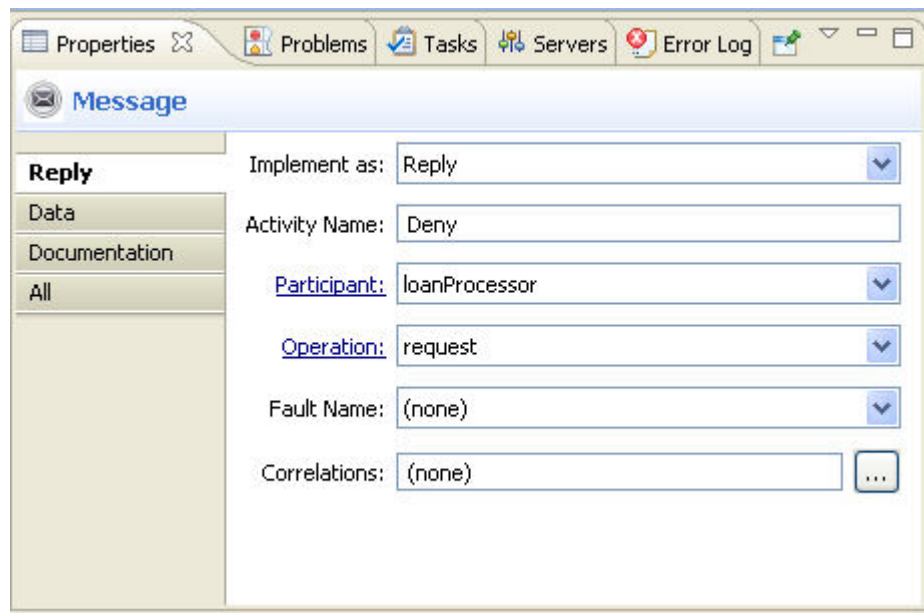
任意のパレット項目を選択して、Process Editor キャンバスにドラッグできます。

Process Editor キャンバスでフォーカスされているアクティビティを使用して、[プロパティ] ビューでプロパティ値を設定できます。左に示した境界線のシーケンスではなく、右に示すようにアクティビティを選択する必要があります。

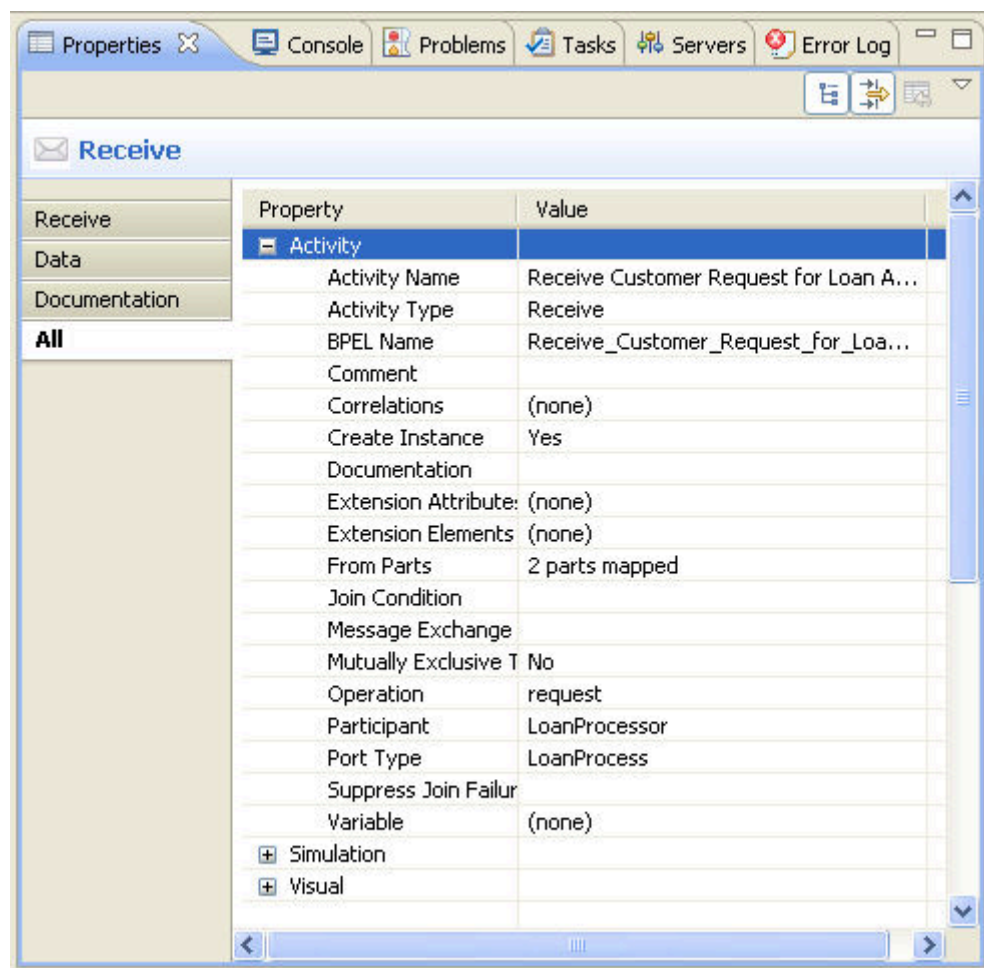


この図は、基本的なアクティビティのプロパティの例を示しています。例として示したアクティビティは、Reply です。

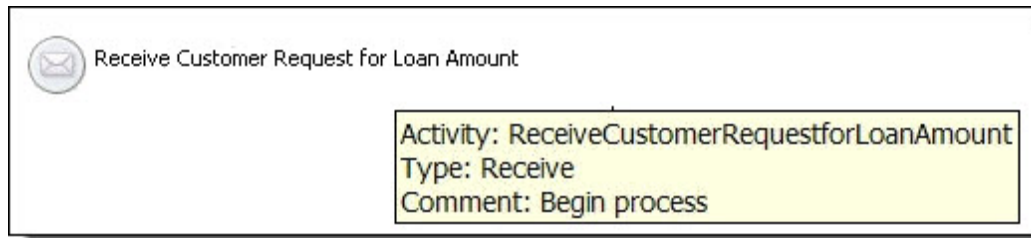
選択されている BPMN 構成は、Reply として実装された Message Throw イベントであることに注意してください。



[プロパティ] ビューで [すべて] タブを選択して、アクティビティのすべてのプロパティを表示します。[すべて] タブには、すべてのアクティビティの標準プロパティが含まれています。次の図では、基本プロパティと詳細プロパティが表示されています。



マウスを [アクティビティ] アイコンに合わせると、アクティビティのプロパティの詳細が表示されます。次の図は、例を示しています。



詳細については、以下を参照してください。

- [「アクティビティのプロパティの値の選択」 \(ページ 158\)](#)
- [「アクティビティラベルの選択」 \(ページ 158\)](#)
- [「アクティビティの標準プロパティ」 \(ページ 159\)](#)

アクティビティのプロパティの値の選択

Process Editor キャンバスにアクティビティを追加すると、[プロパティ] ビューにそのアクティビティの有効なプロパティがすべて表示されます。一部のプロパティはアクティビティに固有のものであり、その他の一部はすべてのアクティビティに標準のものであります。

例えば、[インスタンスの作成] プロパティは、Receive または Pick アクティビティに対してのみ有効です。コピー操作は、Assign アクティビティに対してのみ有効です。

次のようにプロパティに値を割り当てます。

- アクティビティ名の場合は、いくつかの選択肢があります。[「アクティビティラベルの選択」 \(ページ 158\)](#) を参照してください。
- 変数などのリスト値のプロパティの場合は、リストから選択します。
- 相関セットなどの複合的なプロパティを作成するには、[ダイアログ (...) ボタン] をクリックします。

アクティビティラベルの選択

Process Editor キャンバスにアクティビティを追加すると、アクティビティタイプに応じたラベルが付けられます。例えば、Receive アクティビティには *Receive* というラベルが付けられます。名前または説明を追加することで、ラベルの意味をよりわかりやすくすることができます。また、名前を追加して、画面のスペースを確保するためにタイプのみを表示することもできます。

次の例は、一部のラベル付けの方法を示しています。必要に応じてラベルの長さを短くし、詳細はホバーヘルプを使用して表示できることに注意してください。

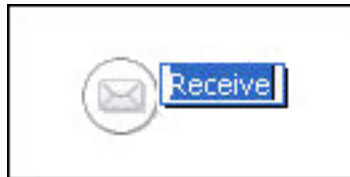


次のいずれかの方法でラベルを作成します。

- Process Editor キャンバスでアクティビティラベルを直接編集するか、[プロパティ] ビューにアクティビティ名を入力する
- [ラベル] プロパティからラベルタイプを選択する

アクティビティラベルを編集する手順:

1. Process Editor キャンバスで、アクティビティラベルをクリックします。図のように、ラベルフィールドが編集フィールドに変わります。



2. アクティビティ名と同じラベルを入力します。スペースまたは特殊文字を含めることができます。

次のように、[プロパティ] ビューのラベルフィールドからラベルタイプを選択します。

アクティビティ名またはアクティビティタイプ	デフォルトでは、タイプが表示されます。名前を追加すると、その名前が表示され、キャンバス上で編集できるようになります。
アクティビティ名	これは表示のみを目的としています。スペースや特殊文字を含めることができ、キャンバス上で編集できます。BPEL XML ではエクスポートされません。
アクティビティタイプ	画面のスペースを確保する場合に適しています。タイプはキャンバス上では編集できません。
アクティビティタイプ: 名前	わかりやすい意味が書かれたラベルですが、キャンバス上で編集することはできません。
カスタム	[カスタムラベル] フィールドでテキストを入力します

アクティビティの標準プロパティ

次の表は、すべてのアクティビティの標準プロパティを示しています。この表には、各アクティビティの説明に記載されたアクティビティ固有のプロパティは含まれていません。

属性	説明
結合の失敗の非表示	第9章, 「BPEL プロセス」 (ページ 120) を参照してください。
結合条件	「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
コメント	「プロセスへのコメントの追加」 (ページ 79) を参照してください。
ドキュメント	「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
実行状態	「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
拡張属性と要素	「拡張要素と属性の宣言」 (ページ 126) を参照してください。

[プロパティ] ビューで [すべて] タブを選択すると、[結合の失敗の非表示]、[結合条件]、および [拡張] プロパティが [プロパティ] ビューに表示されます。

背景色の追加

図のわかりやすさを向上するために、次のようなアクティビティに色を追加できます。

- 境界のあるアクティビティ。境界のあるアクティビティは、BPMN パレットの **[タスク]** ドロワーにあります。
- **折りたたまれたスコープ**。マウスを右クリックして [コンテナを折りたたむ] を選択し、スコープを折りたたみます。スコープは、[制御フロー] ドロワーの BPMN パレットでは組み込みサブプロセスと呼ばれます。
- 折りたたまれたフォールト、イベント、または補償ハンドラ

WSDL インタフェースから開始するアクティビティの作成

作成するアクティビティタイプを選択します。[Receive-Reply] を選択して、Receive に一致する Reply を作成します。OnMessage アクティビティをアクティブにするには、[操作ウィザード] アイコンを Pick アクティビティにドロップします。OnEvent の場合は、ウィザードをプロセスまたはスコープのイベントハンドラにドロップします。参照される WSDL ファイルで宣言されたフォールト名を選択します。既存のフォールト変数を選択するか、新しいフォールト変数を作成します。アクティビティにメッセージパートを使用するには、[変数なし] を選択します。

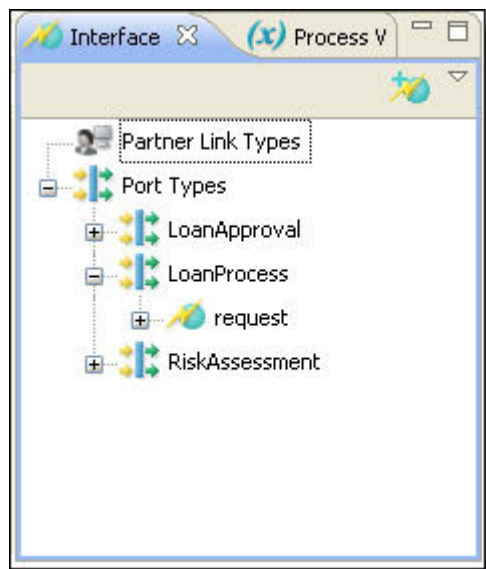
Process Developer には、実行可能プロセスのアクティビティを作成するためのショートカットがあります。[パーティシパント] ビューまたは [インタフェース] ビューから操作を選択し、ウィザードを完了することで、Web インタラクションアクティビティを作成できます。

主要な生産性の開始ポイントについては、「*[パーティシパント] ビューの使用*」を参照してください。

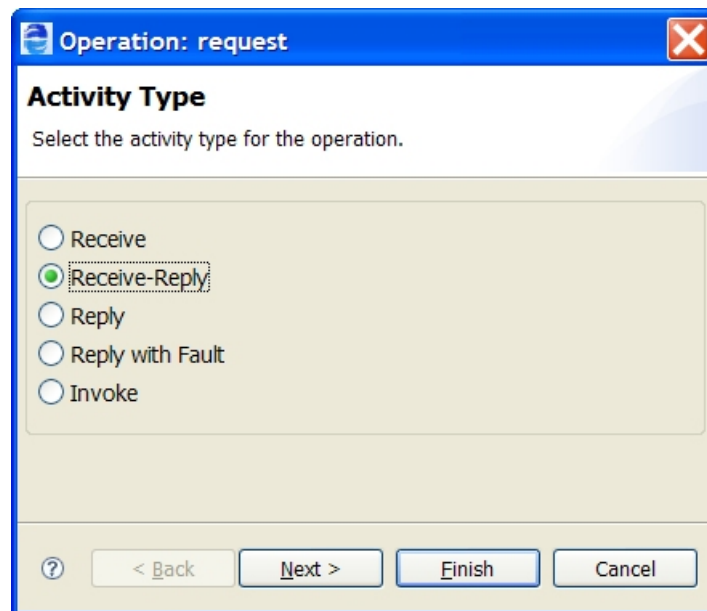
パートナーリンクタイプの定義からの開始

WSDL ファイルにパートナーリンクタイプの定義が含まれていない場合は、ポートタイプから操作を選択できます。

1. WSDL でパートナーリンクタイプ要素を表示します。この図は、パートナーリンクタイプを作成する前の loanProcess ポートタイプの例を示しています。



2. 受信元、返信先、または呼び出す操作を選択します。
 3. 操作を次のいずれかの場所にドラッグします。
 - Receive、Reply、フォールトが発生した Reply、または Invoke を作成する Process Editor キャンバス上の任意の場所
 - スコープレベルのパートナーリンクまたは変数を作成する場合は、Process Editor キャンバスのスコープ内
 - onMessage および/または Reply を作成する Pick アクティビティ
 - onEvent および/または Reply を作成するイベントハンドラ
- [アクティビティタイプ] ページに表示されるリストは、操作をドロップした場所に依じて異なります。次の図は、Pick アクティビティまたはイベントハンドラではなく、プロセスまたはスコープレベルの場所のアクティビティを示しています。WSDL 操作には入力メッセージと出力メッセージがあるため、Receive-Reply の組み合わせがリストされています。同様に、WSDL 操作に対してフォールトが宣言されているため、フォールトが発生した Reply がリストされます。



4. アクティビティタイプを選択します。ショートカットとして [Receive-Reply] を選択することで、一致するアクティビティを同時に設定できます。[次へ] をクリックします。フォールトが発生した Reply を作成する場合は、「フォールトが発生した Reply の作成」を参照してください。
5. 可能な場合は、プロセスですでに定義された既存のプロセスレベルまたは上位スコープのパートナーリンクを使用するか、新しいリンクを入力します。パートナーリンクは、プロセスとサービスのロールを特定します。スコープ内に新しいパートナーリンクを作成する場合は、ロケーションの宣言に [エンクロージングスコープ] を選択して、プロセスでグローバルに使用されるのではなく、ローカルでのみパートナーリンクが使用されるようにします。[次へ] をクリックします。

Operation: request

Partner Link

Select or create the Partner Link for the operation.

☐ Use existing Partner Link:

-- None matching --

☒ Create new Partner Link:

Name:

Location:

Role Assignment

My Role:

? < Back Next > Finish Cancel

ウィザードが終了した後に、パートナーリンクやその他のウィザード情報を編集できます。

6. 入力変数として次のいずれかを選択し、**次へ** をクリックします。
 - 操作の入力メッセージに関連付けられた既存の変数名を使用する
 - プロセスの入力変数に新しい名前を入力する。[プロセス] または [エンクロージングスコープ] から宣言の場所を選択します。[メッセージ] または [要素] から変数タイプを指定します。要素によって定義された単一のパートを持つメッセージタイプの変数である場合は、要素を選択できます。
 - 変数が空のメッセージ用である場合、またはアクティビティの fromPart または toPart を作成する場合は、[変数なし] を選択する。詳細については、「変数からパート」を参照してください。

Operation: request

Input Variable

Add or create a variable for the input message associated with the operation.

☐ Use existing Variable:

-- None matching --

☒ Create new Variable:

Name: request

Location: Process

Type: Message

☐ No Variable

< Back Next > Finish Cancel

7. Invoke または Reply アクティビティの場合は、出力変数名に対して手順 6 を繰り返します。**【完了】** をクリックします。

フォールトが発生した Reply の作成

操作ウィザードで「フォールトが発生した Reply の作成」を選択した場合は、次の手順を実行します。

1. 操作に対して宣言されているフォールトを選択し、**【次へ】** をクリックします。

Operation: approve

Fault Name

Select the name of the fault to use with the new reply activity.

Declared Faults:

loanProcessFault

< Back Next > Finish Cancel

2. フォールト変数に対して次のいずれかを選択します。

- 操作フォールト変数に関連付けられた既存の変数名を使用する
- プロセスのフォールト変数に新しい名前を入力する。[プロセス] または [エンクロージングスコープ] から宣言の場所を選択します。[メッセージ] または [要素] から変数タイプを指定します。要素によって定義された単一のパートを持つメッセージタイプの変数である場合は、要素を選択できます。
- 変数が空のメッセージ用である場合、またはアクティビティの fromPart または toPart を作成する場合は、[変数なし] を選択する。詳細については、「[開始パートから変数](#)」(ページ 197)を参照してください。

パートナーリンクタイプの定義を使用しない開始

選択した操作がパートナーリンクタイプに関連付けられていない場合は、ポートタイプから操作を選択できます。表示されるウィザードでは、パートナーリンクタイプ定義を作成して既存の WSDL ファイルに追加するか、新しい WSDL ファイルを作成することができます。詳細については、「[パートナーリンクタイプ](#)」を参照してください。

カスタムアクティビティの作成

以前にカスタムパレットに保存したアクティビティまたはアクティビティグループを選択します。

BPEL ファイルから、任意のアクティビティまたはアクティビティのセットをカスタムアクティビティとして保存できます。他の BPEL ファイルでカスタムアクティビティを再利用することにより、設計時間を短縮できます。

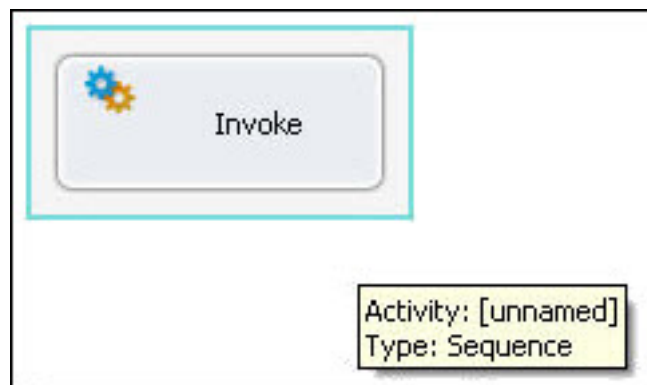
カスタムアクティビティを作成する手順

1. Process Editor キャンバスで 1 つ以上のアクティビティまたはコンテナを選択します。

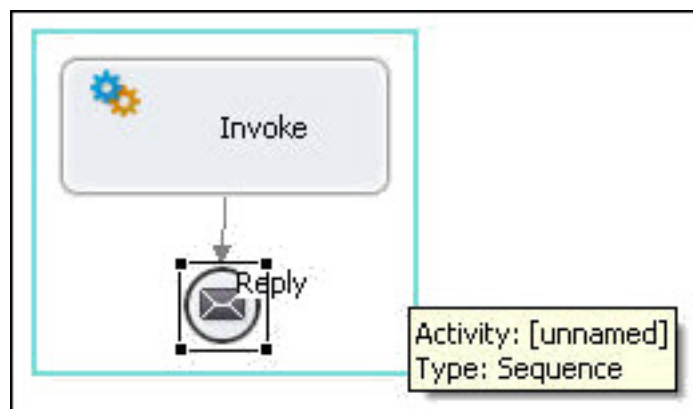
2. マウスを右クリックして、[パレットに追加] を選択します。
空であった場合は、カスタムパレットが表示されます。選択範囲は、カスタムパレットに *Custom1* として追加されます。
3. カスタムアイコンを右クリックして、[カスタマイズ] を選択します。
4. [名前] フィールドに入力して、カスタムアイコンの名前を変更します。
5. 小（最大 20 x 20）、または大（最大 32 x 32）のいずれかを使用できます。項目のアイコンの画像。
6. [OK] をクリックします。

アクティビティシーケンスおよびフローの理解と使用

Process Editor キャンバスに追加する各アクティビティは、非表示の Sequence アクティビティに含まれています。図に示すように、シーケンスは、マウスカーソルを近くに置くと青い境界線として表されます。



シーケンスが非表示になっている理由は、アクティビティを簡単にリンクできるようにするためです。アクティビティを別のアクティビティの上、下、左、または右に近づけて配置すると、アクティビティは自動的にシーケンスにリンクされます。



BPMN の設計のヒント

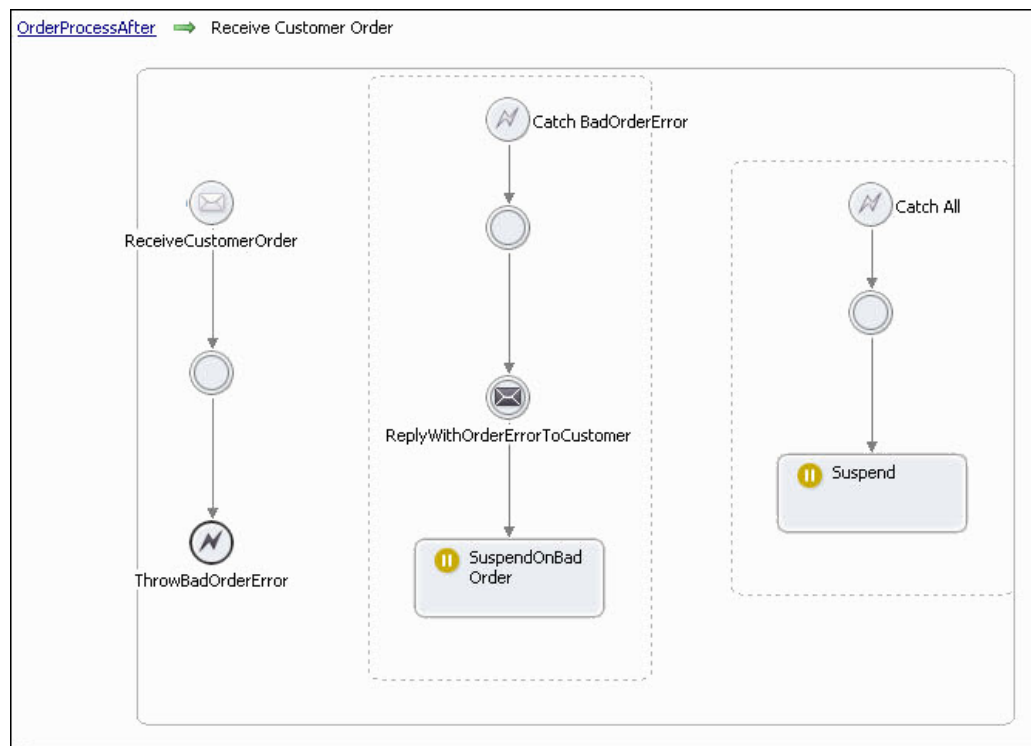
ビジネスプロセスモデリング表記法（BPMN）とは、分析、文書化、および実行のためのアクティビティのフローをモデル化するためにビジネスアナリストや設計者が通常使用するグラフィック指向の表記法標準です。Process Developer は、実行可能な BPEL プロセスを設計するために、デフォルトでこの標準を使用します。

プロセスの上位レベルのビューと詳細ビューの表示

プロセスをスコープで表した作業単位に編成します。スコープとは、任意の数のアクティビティおよび制御フロー用のコンテナです。右クリックメニュー項目を使用して、スコープを折りたたみます。例に示すように、スコープを折りたたむと、上位レベルのプロセスのビューが表示されます。



以下に示すように、プラス記号 (+) または右クリックメニューオプションの「アクティビティに移動」を選択して、スコープに含まれるアクティビティを表示します。ウィンドウの上部にあるナビゲーショントレイルを選択して、プロセスの全体像に戻ります。

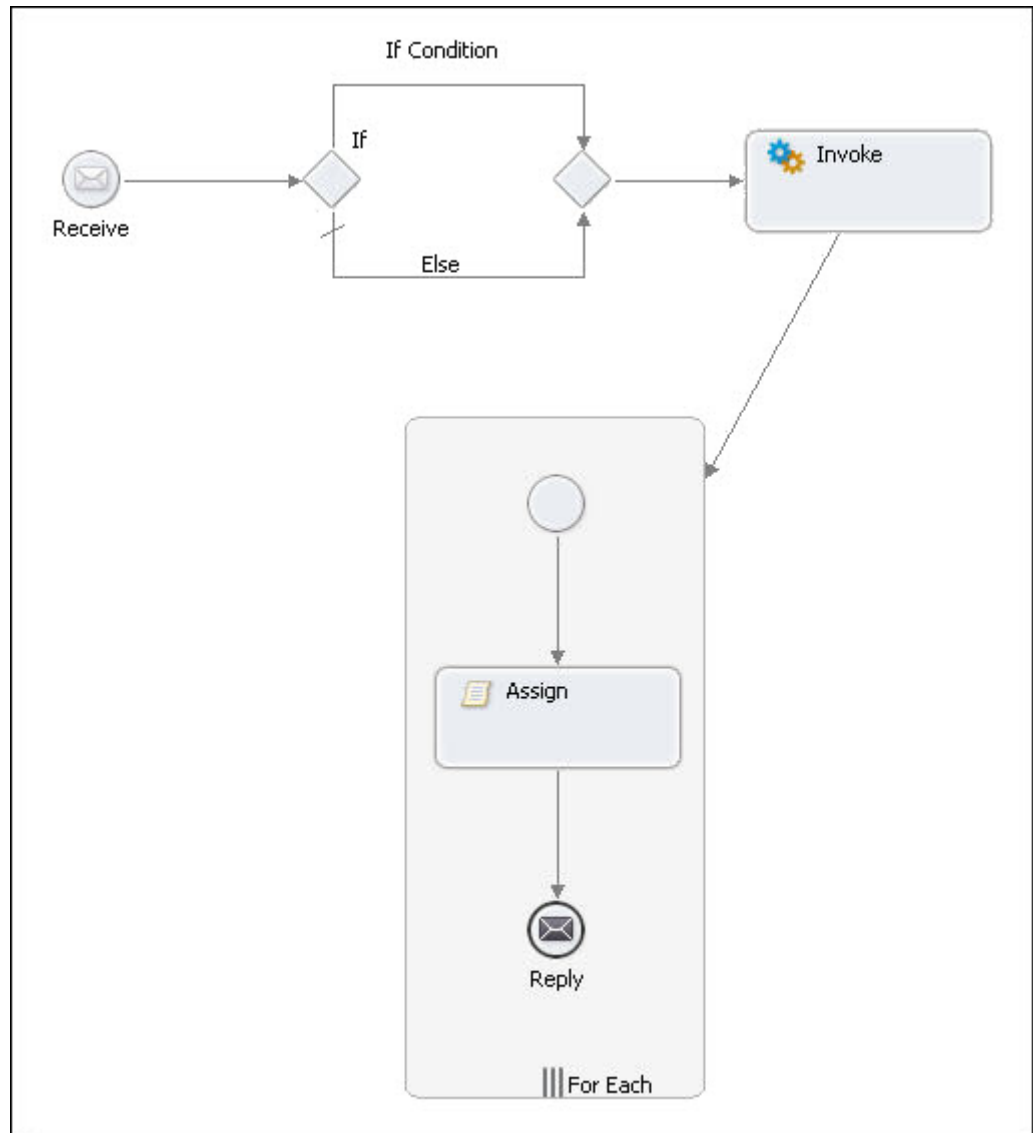


横方向または縦方向レイアウトの選択

プロセスおよびプロセス内の個々のアクティビティに適したレイアウトを選択します。

シーケンス内などにアクティビティが含まれている場合は、親コンテナによってレイアウトが設定されます。アクティビティがリンクで接続されている場合は、接続されたアクティビティごとに異なるレイアウトを選択

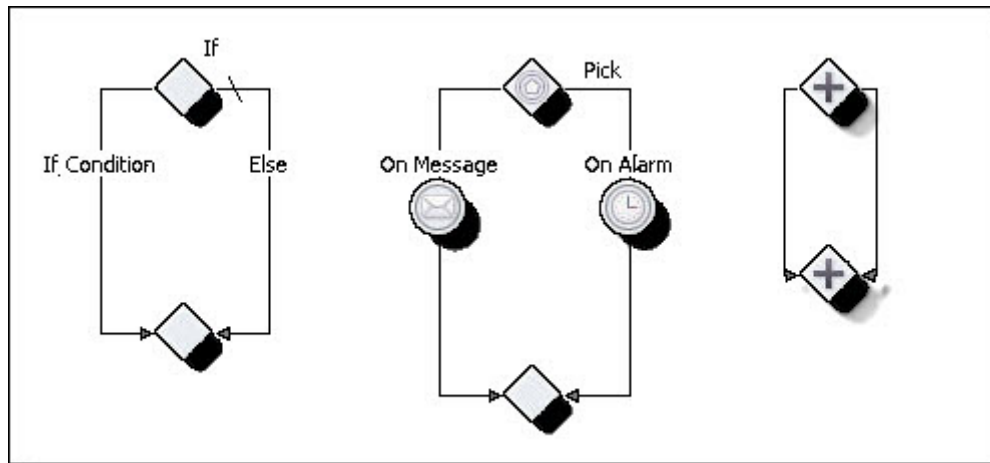
できます。以下の例では、3つのアクティビティを含む横方向シーケンスが、縦方向に整列されたアクティビティにリンクされています。



構造化された制御フローの使用

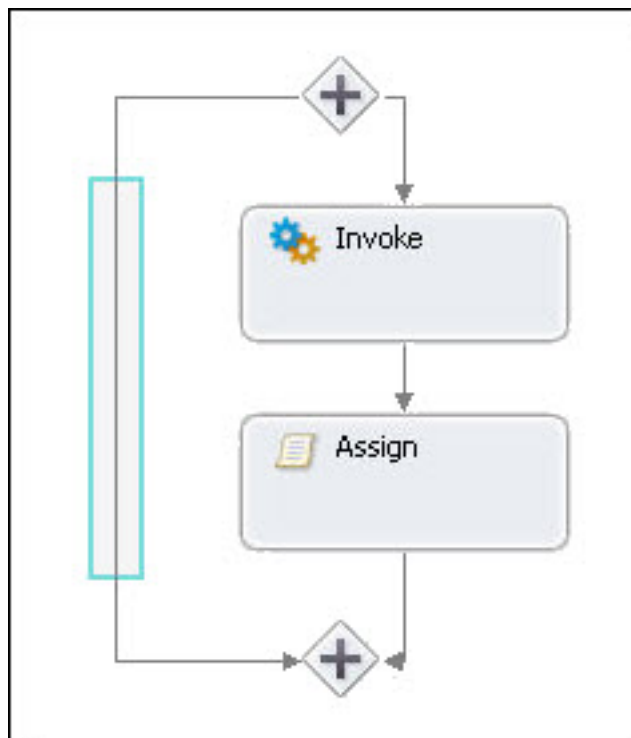
アクティビティに組み込まれた構造を利用します。

例に示すように、If、Pick、および Fork Join アクティビティが、それぞれ2つのパスで事前に構築されています。必要に応じて、パスを追加したり、既存のパスを削除したりできます。例えば、Fork Join に複数のパスを追加して、複数のアクティビティを並列で実行します。



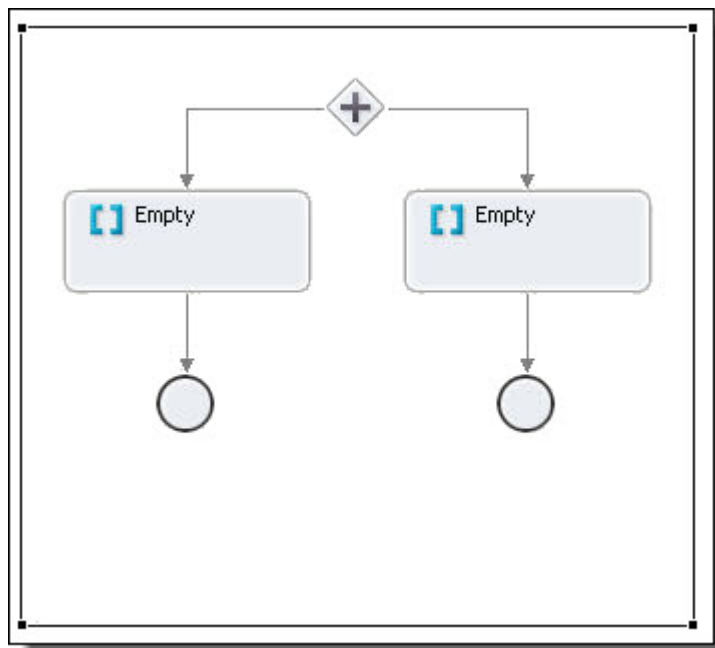
If、Pick、Fork Join のパスへのアクティビティの追加

Fork Join の例の中で青い境界線で示されているように、パレットから暗黙的なシーケンスにアクティビティをドラッグします。パスに追加する新しいアクティビティはそれぞれ、右側のパスに示されているような順番で並んでいます。



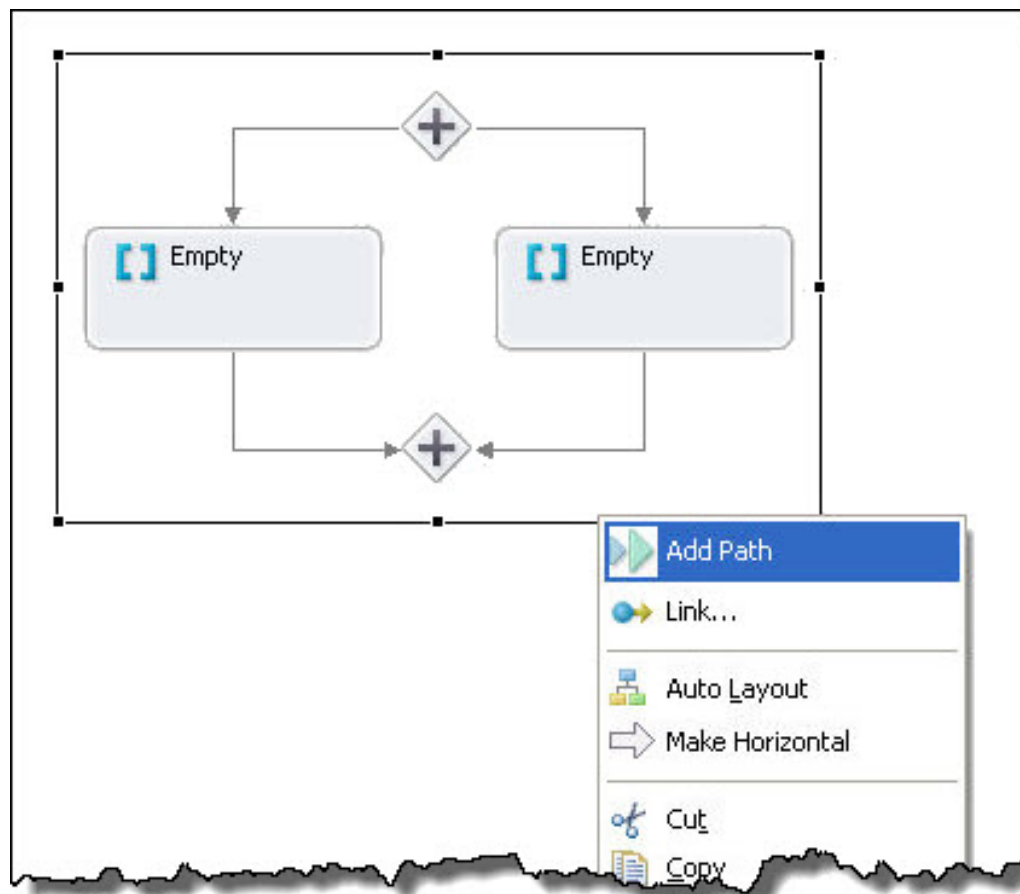
If、Pick、Fork Join パスへの終了イベントの追加

Fork Join、If、または Pick の後にアクティビティがない場合、Fork Join の例に示すように、終了イベントを追加することでマージダイヤモンドを削除できます。



If、Pick、Fork Join のパスの追加および削除

Fork Join の例に示すように、右クリックメニューを使用してパスを追加または削除します。



ダブルクリックによる式と条件の追加

アクティビティラベルをダブルクリックして、If、While、または Repeat Until に条件を追加します。適切なビルダーが開きます。ダブルクリックによって、リンクの Assign アクティビティと遷移ビルダーも開きます。

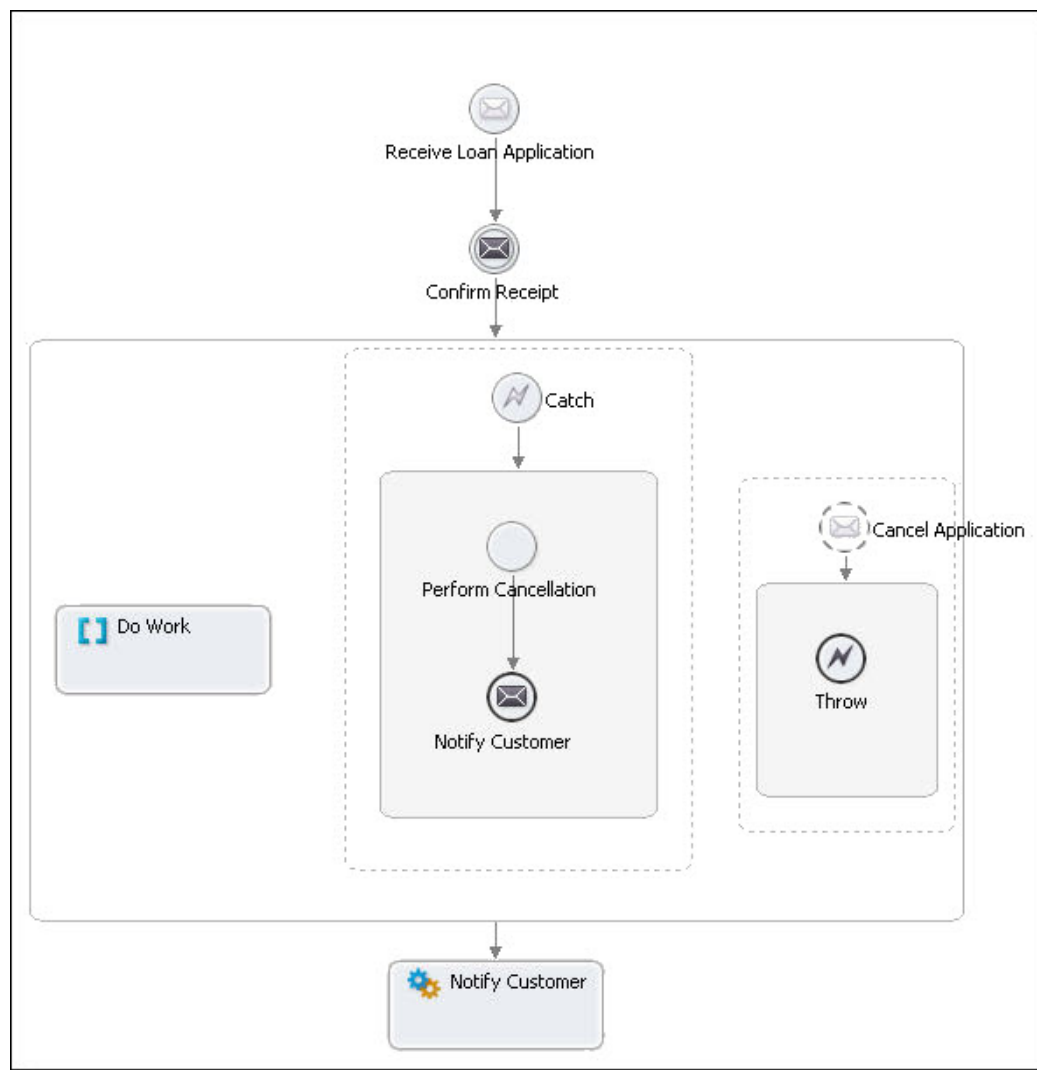
イベント処理のビューを最適化するための境界イベントの使用

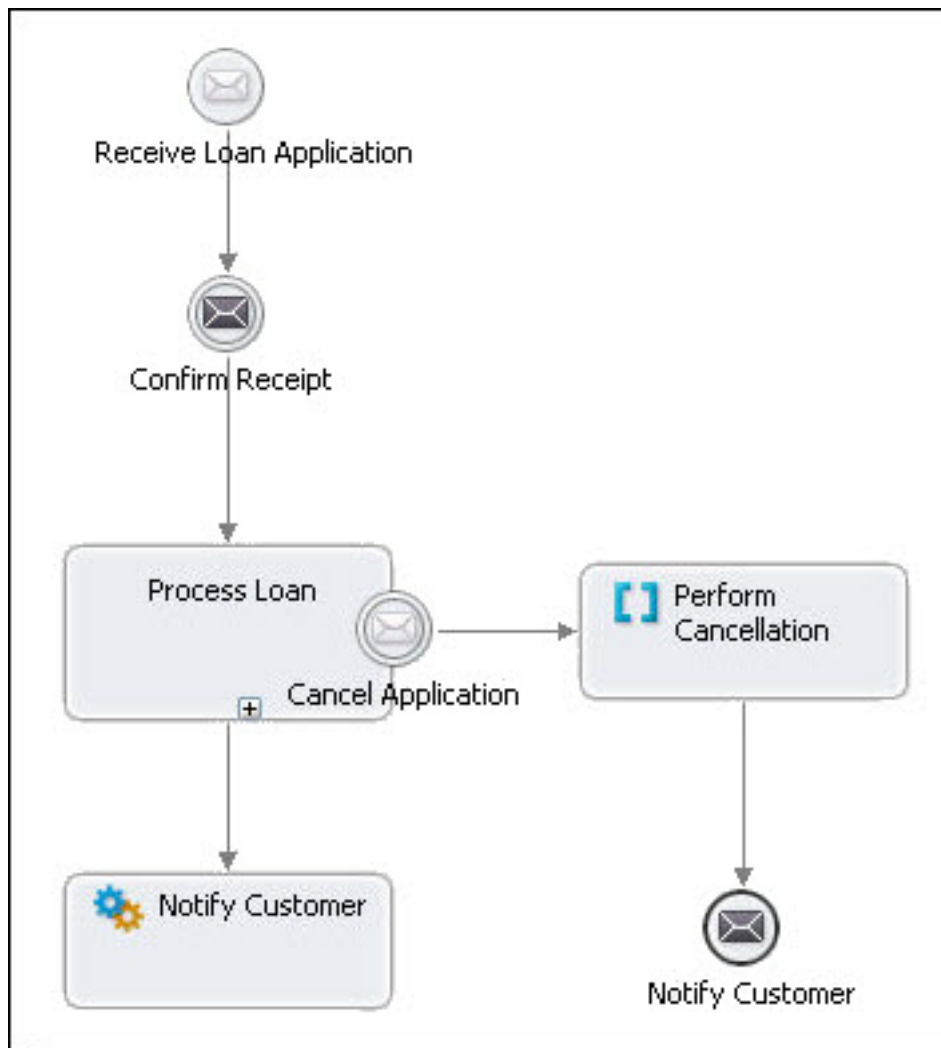
通常のスコープハンドラの代わりに境界イベントを使用して、見やすいプロセスイベントのビューを作成できます。

次の例では、上の図に、Catch と onEvent ハンドラを持つスコープが表示されています。onEvent は、ローン申請をキャンセルするために Catch にフォールトをスローします。下の図では、キャンセル作業にリンクする中断 onEvent の使用によって、同じイベントが発生しています。

下の図は、より理解しやすい構造を持ちます。onEvent および onAlarm イベントの中断は、WS-BPEL 2.0 に対する Informatica Business Process Manager の拡張機能であることに注意してください。

図 1: 通常のキャッチハンドラとイベントハンドラを使用したスコープ:

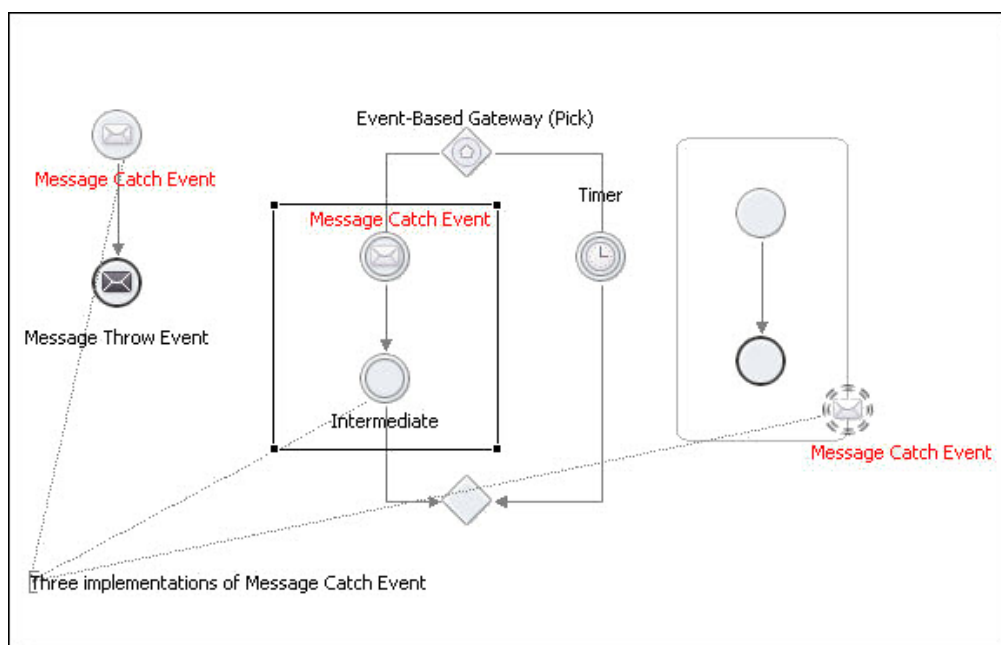




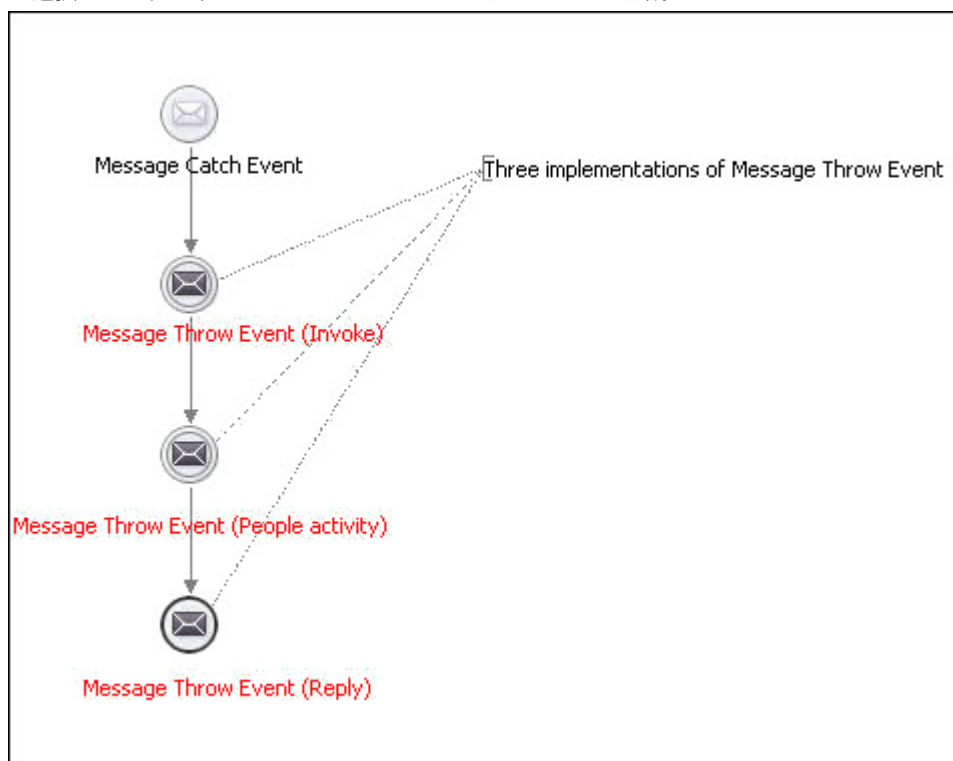
その他の例

BPMN の使用に関する注意事項は次のとおりです

- これまでは、プロセスのマイルストーンを記録するために None 中間イベントが使用されていました。例えば、見積もり要求が現在進行中であるとします。このようなマイルストーンでは、ステータス変数に割り当てができるようにしたいと考えるのが一般的です。したがって、「進行中」とラベル付けされた中間 None イベントによって、内部的に、進行中の文字列がステータス変数に割り当てられます。または、None 中間イベントが空のまま実装されることもあります。
- メッセージ Catch イベントを境界のあるアクティビティの境界にドラッグすると、OnEvent イベントハンドラになります。同じイベントをイベントベースのゲートウェイにドラッグすると、OnMessage になります。イベントを他の場所にドラッグすると、Receive になります。メッセージ Catch イベントは、何らかのアクティビティによってトリガされるのを待つ待機イベントです。



- メッセージ Throw イベントをキャンパスにドラッグし、必要な実装（Invoke、ユーザー、または Reply）を選択します。メッセージ Throw イベントは、イベントを生成するアウトバウンドイベントです。



割り当て

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

BPMN 実装: スクリプトタスク、None/Start/End スローイベント

Assign アクティビティは、変数の内容を更新します。このアクティビティは、次のような複数の方法で変数を更新します。

- コピー操作のために、ある変数から別の変数にデータをコピーする
- 定義のコピー元またはコピー先の指定を求めずに、スクリプト操作でスクリプトを実行する
- XPath（または他の言語）の式と関数を使用して新しいデータを構築する
- WS-BPEL およびその他の拡張関数を使用して新しいデータを構築する

Assign を使用して、エンドポイント参照をパートナーリンクとの間でコピーし、サービスパートナーを動的に選択することもできます。

必要に応じて、XML または WSDL 定義に照らし合わせて変数の値を検証できます。

ヒント: Receive、Invoke、Reply、およびユーザーアクティビティ内でデータの割り当てを作成すると、Assign を作成するためのショートカットを使用できます。詳細については、[「入力変数」 \(ページ 197\)](#)を参照してください。

必須のプロパティ	オプションのプロパティ
コピー オプションの属性: ソース要素名の維持。 「ソース要素名属性を維持する要素間のコピー操作」 (ページ 182) を参照してください。 見つからない開始データを無視。 「[見つからない開始データを無視]」の属性を使用したコピー操作」を参照してください。	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。

必須のプロパティ	オプションのプロパティ
	検証。 「変数の検証」 (ページ 258)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)の要素と属性を参照してください。

プロセスに Assign アクティビティを追加する手順

1. **【タスク】** パレットから、スクリプトタスクを Process Editor キャンバスにドラッグします。
スクリプトタスクには背景色を追加できます。
2. Assign をダブルクリックして、**【操作】** ダイアログを開きます。
または、**【プロパティ】** ビューで **【コピー操作】** の **【値】** フィールドをクリックして、**【ダイアログ (...) ボタン】** を選択します。
3. **【新規コピー】** または **【新規スクリプト】** を選択します。
4. **【新規スクリプト】** を選択した場合、実行する式を作成するスクリプトビルダーが開きます。スクリプト操作を作成することで、コピー操作で使用するコピー元/コピー先の仕様を定義する必要がなくなります。
5. **【新規コピー】** を選択した場合は、**【コピー元】** セクションで **【タイプ】** を選択します。

FROM タイプ	説明	このタイプを選択すると...
式	変数とプロパティの計算を実行するための XPath (または他の言語) 式。文字列、数値、またはブール値を返します。	式は式ビルダーで作成できます。詳細については、 「式ビルダの使用」 (ページ 275)を参照してください。
リテラル	XML スキーマの定義に準拠する定数。タイプは、xsi:type を使用してインラインで示されます	次のいずれかを実行します。 リテラル値 (例えば、100) を入力します より大きな編集領域を開くには、リテラルコンテンツの 【ダイアログ (...) ボタン】 を選択します 「XML データウィザードの使用」 (ページ 251)および「コピー操作のリテラルコンテンツの例」に記載されているように、複合型の変数にリテラルコンテンツを生成します。 動的エンドポイント参照を割り当てます。「コピー操作の動的エンドポイントリファレンスの例」を参照してください。
あいまい	プライベートアプリケーションデータを表す名前。このオプションは、抽象プロセスにのみ使用してください。詳細については、 「実行可能なプロセスと抽象的なプロセス」 (ページ 134)を参照してください。	他の選択肢は許可されていません
パートナー	これを選択すると、パートナーリンクとロールのリストが生成されます	エンドポイント参照を定義するパートナーリンクとロールを選択します

FROM タイプ	説明	このタイプを選択すると...
変数	WSDL メッセージ変数とオプションのパート名 WSDL メッセージプロパティ XML スキーマの単純な型または要素	変数名を選択し、オプションでパート名とクエリを選択します。詳細については、「 クエリビルダの使用 」(ページ 290)を参照してください。クエリは、実行可能なプロセスでのみ許可されます。
変数プロパティ	WSDL メッセージプロパティ	変数名とプロパティを選択します

6. [コピー先] セクションで、タイプを選択します。以下に示すように、[TO タイプ] は [FROM タイプ] と互換性を持ちます。

この FROM タイプ	この TO タイプと一致します
メッセージタイプの変数	一致するタイプを持つメッセージタイプの変数
XML スキーマタイプの変数	スキーマタイプを含む変数
XML 要素タイプの変数	同じ要素を含む変数
リテラル	一致する XML スキーマタイプを持ちます
Expression	式または変数
パートナーリンク	パートナーロール

7. 必要に応じて、要素ベースの変数にのみ適用される【**ソース要素名の維持**】チェックボックスを有効にします。ソース要素名は、結果のコピー先要素の名前として使用されます。「[ソース要素名属性を維持する要素間のコピー操作](#)」(ページ 182)を参照してください。
8. 必要に応じて、「見つからない開始データを無視」チェックボックスを有効にして、ソース変数にオプションのデータが見つからない場合の選択エラーを回避します。「[見つからない開始データを無視]」の属性を使用したコピー操作」を参照してください。
9. **[OK]** をクリックします。Assign には複数の操作を追加できます。これらの操作は、**[操作]** ダイアログおよび [アウトライン] ビューに表示される順序で実行されます。
10. 必要に応じて、[プロパティ] ビューで [検証] プロパティを [はい] に設定して、Assign のコピー操作で使用されるすべての変数を検証します。

新しい Assign を作成したり、既存の Assign にコピー操作を追加したりするためのショートカットとして、[プロセス変数] ビューでドラッグアンドドロップ機能を使用できます。詳細については、「[コピー操作での変数の使用](#)」(ページ 256)を参照してください。

コピー操作の XML 構文

```
<assign validate="yes|no"? standard-attributes>
  standard-elements
  (<copy keepSrcElementName="yes|no"?
    ignoreMissingFromData="yes|no"?>
    from-spec (see below)
    to-spec (see below)
  </copy> |
  <extensionAssignOperation>
    ...assign-element-of-other-namespace...
  </extensionAssignOperation>) +
</assign>
```

From-Spec XML 構文

FROM 要素は、次のいずれかの形式にすることができます。

```
<from variable="BPELVariableName" part="NCName"?
  <query queryLanguage="anyURI"?>?
    queryContent
  </query>
</from>
<from partnerLink="NCName"
  endpointReference="myRole|partnerRole"/>
<from variable="BPELVariableName" property="QName"/>
<from expressionLanguage="anyURI"?>expression</from>
<from><literal>literal value</literal></from>
```

抽象プロセスでは、次の行も使用できます。

```
<from opaque="yes">
```

あいまいデータは、ビジネスパートナーに公開しないプライベートアプリケーションデータです。これは、実行可能なプロセスで使用する実際のデータの抽象的なプロセスプレースホルダです。

To-Spec XML 構文

TO 要素は、次のいずれかの形式にすることができます。

```
<to variable="BPELVariableName" part="NCName"?
  <query queryLanguage="anyURI"?>?
    queryContent
  </query>
</to>
<to partnerLink="NCName"/>
<to variable="BPELVariableName" property="QName"/>
<to expressionLanguage="anyURI"?>expression</to>
```

例:

例 1:

```
<assign>
  <copy>
    <from>'yes'</from>
    <to part="accept" variable="approval"/>
  </copy>
</assign>
<assign>
  <copy>
    <from>$po.lineItem[@prodCode=$myProd]/amt * $exchangeRate
    </from>
    <to>$convertedPO.lineItem[@prodCode=$myProd]/amt</to>
  </copy>
</assign>
```

例 2: スクリプト操作の XML

```
<bpel:assign>
  <bpel:extensionAssignOperation>
    <bpel:documentation>new script operation
    </bpel:documentation>
    <ext3:script>abx:createAttachment('request',
      'text/plain',
      abx:base64Encode("Test attachment"))</ext3:script>
  </bpel:extensionAssignOperation>
</bpel:assign>
```

関連事項:

- [「コピー操作のヒント」 \(ページ 177\)](#)
- コピー操作のクエリと式の例
- コピー操作のリテラルコンテンツの例

- コピー操作の動的エンドポイントの参照例
- [「ソース要素名属性を維持する要素間のコピー操作」 \(ページ 182\)](#)
- 「見つからない開始データを無視」の属性を使用したコピー操作

コピー操作のヒント

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

コピー操作は次のような場合に使用します。

- Assign アクティビティのコピー操作は、[操作] ダイアログにリストされた順序で実行されます。この順序は、[アウトライン] ビューにも表示されます。
- 操作は、[操作] ダイアログまたは [アウトライン] ビューで並べ替えることができます。
- コピー操作ごとにコメントやドキュメントを追加することもできます。[アウトライン] ビューから操作を選択します。操作のプロパティが表示され、プロパティにコメントやドキュメントを追加できます。詳細については、[「プロセスへのコメントの追加」 \(ページ 79\)](#)を参照してください。

コピー操作のクエリと式の例

選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。

例 1: クエリを選択

Define the From and To parts of the Copy Operation.

From

Type: Variable

Variable: inputVariable

Part: payload

Query: string(\$inputVariable.payload/ns5:Age)

To

Type: Variable

Variable: Invoke_1_getPerson_InputVariable

Part: Person_getPerson_getPersonInput

Query: ns2:PersonAgeDescriptionText

☐ Keep source element name

☐ Ignore missing "From" data

OK Cancel

例 2: 組み込みの BPEL 関数を使用した式

Add, edit or reorganize Copy operations.

Copy operation:

Copy Expression(bpel:getVariableProperty(inputVar, prop1)) TO Variable(Invoke_1_getPerson_Inp

詳細については、「[式ビルダの使用](#)」(ページ 275)を参照してください。

コピー操作のリテラルコンテンツの例

「FROM タイプ」を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。複合型または要素の場合、XML データファイルを生成できます。「生成」ボタンがグレー表示されている場合、選択した変数は互換性のあるタイプではありません。任意の変数について、このエディタを使用して適切なリテラル値を入力します。ドロップダウンリストから一致するタイプまたは要素を選択して、ルート要素の XML データファイルを作成します。

コピー操作のリテラルコンテンツの例

例 1 に示すように、単純な変数、変数プロパティ、式、またはパートナーリンクにコピーするリテラル値を入力します。例 2 に示すように、複合型の変数の XML データを生成します。

例 1: リテラル値

The screenshot shows a 'Copy Operation' dialog box with the following configuration:

- From:**
 - Type: Literal
 - Literal Contents: yes
- To:**
 - Type: Variable
 - Variable: approval
 - Part: accept
 - Query: (empty)
- ☐ Keep source element name
- ☐ Ignore missing "From" data
- Buttons: OK, Cancel

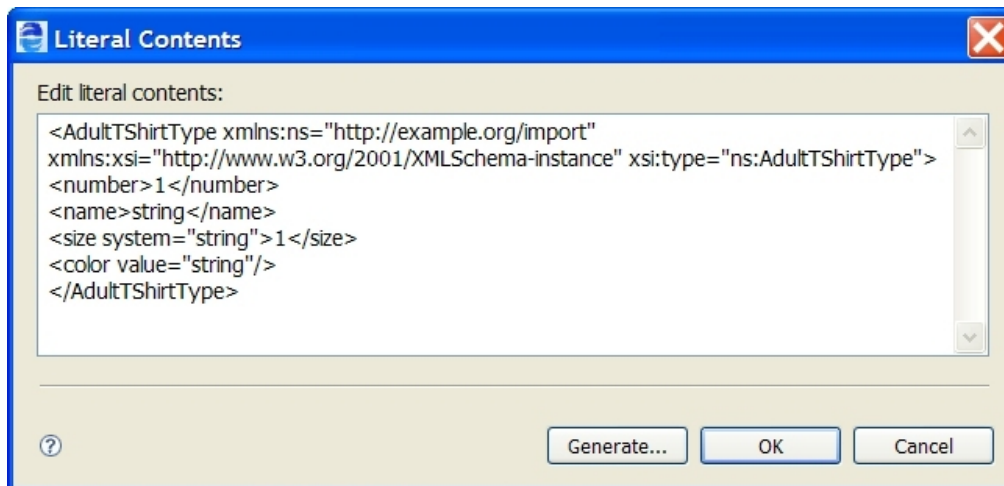
上記の式によって、文字列である変数パートに文字列を割り当てます。変数パートが整数の場合、Process Developer はエラーを報告せず、ゼロ値を生成します。Process Developer は、数値型変換用に不正な値をゼロに変換します。

例 2: 複合型の変数を初期化するために使用されるリテラルコンテンツの生成

以下の手順を実行してください。

1. TO パネルで、XML データを生成する複合型の変数（および、必要に応じてそのパートの 1 つ）を選択します。
最初に TO ターゲットを選択しない場合は、プロセスがインポートしている最上位のスキーマ要素または複合型のフィルタリングされていないリストを使用できます。使用可能な要素またはタイプがない場合、**[生成]** ボタンは使用できません。
2. より大きな編集ボックスを開くには、[リテラルコンテンツ] テキストボックスの下にある [ダイアログ (...)] ボタンを選択します。
3. **[生成]** を選択します。**[生成]** ボタンは、インポートされた WSDL に最上位のスキーマ要素または複合型がある場合にのみ使用できます。TO パネルで既知のタイプまたは要素を選択した場合、リストはフィルタリングされます。

4. 「XML データウィザードの使用」 (ページ 251)に記載されているように、データを生成するための詳細を入力します
5. 例に示すように、必要に応じて、生成されたデータの詳細を [リテラルコンテンツ] 編集ボックスで編集します。



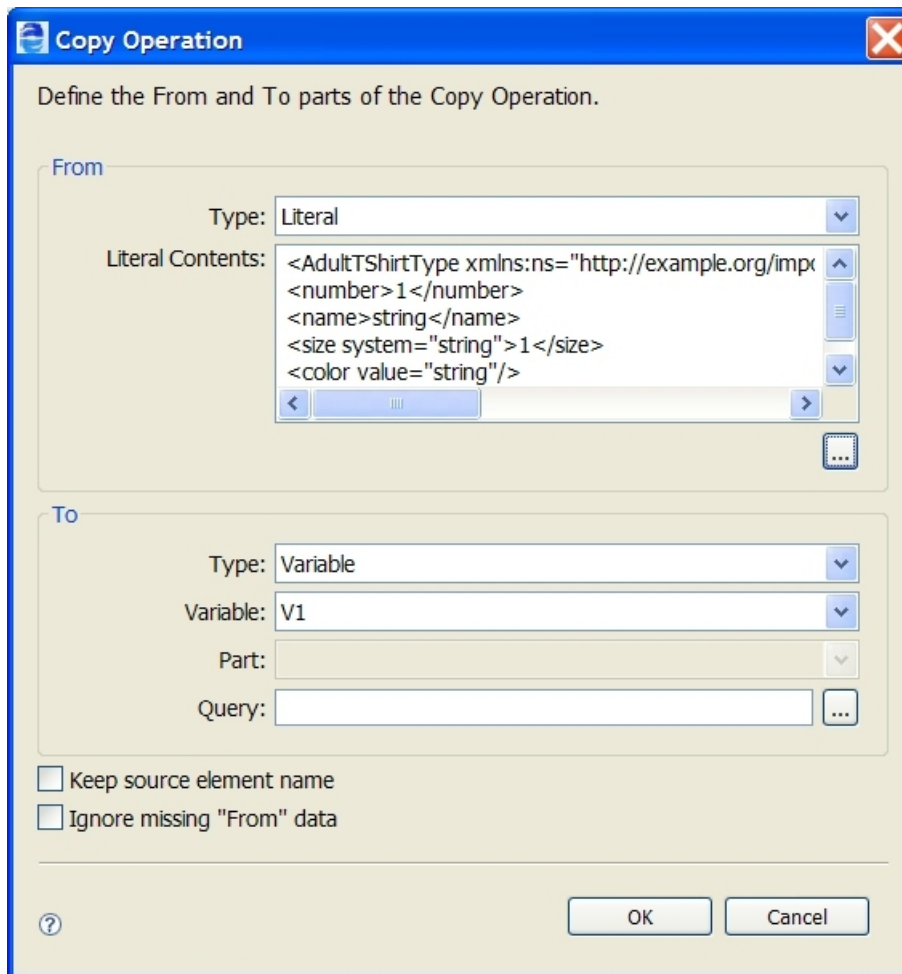
Literal Contents

Edit literal contents:

```
<AdultTShirtType xmlns:ns="http://example.org/import"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns:AdultTShirtType">
<number>1</number>
<name>string</name>
<size system="string">1</size>
<color value="string"/>
</AdultTShirtType>
```

? Generate... OK Cancel

6. 例に示すように、[OK] をクリックしてコピー操作を表示します。



Copy Operation

Define the From and To parts of the Copy Operation.

From

Type: Literal

Literal Contents: <AdultTShirtType xmlns:ns="http://example.org/import" <number>1</number> <name>string</name> <size system="string">1</size> <color value="string"/> </AdultTShirtType>

To

Type: Variable

Variable: V1

Part:

Query:

☐ Keep source element name
☐ Ignore missing "From" data

? OK Cancel

コピー操作の動的エンドポイントの参照例

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

サービスの動的割り当てのコピー操作を指定する方法の例を以下に示します。サービス名は、Assign アクティビティへの入力として指定します。

エンドポイント参照を参照する場合は、必ず WS-Addressing 名前空間を宣言してください。Process Developer は、[「エンドポイント参照のアドレス指定の考慮事項」](#) (ページ 452) に記載されているように、WS-Addressing 仕様の一部のバージョンをサポートしています。

この割り当てを実装するには、[リテラルコンテンツ] テキストボックスに有効なエンドポイント参照を入力します。必要に応じて、テキストボックスの下にあるダイアログ (...) ボタンをクリックし、より大きな編集ボックスを開きます。次のように、リテラルからパートナーリンクにコピーします。

Copy Operation

Define the From and To parts of the Copy Operation.

From

Type: Literal

Literal Contents: `<wsa:EndpointReference xmlns:s="http://www.act
<wsa:Address>anyURI</wsa:Address>
<wsa:ServiceName>tns:Service</wsa:ServiceNan
</wsa:EndpointReference>`

To

Type: Partner

Partner Link: assessor

Query:

☐ Keep source element name

☐ Ignore missing "From" data

OK Cancel

例:

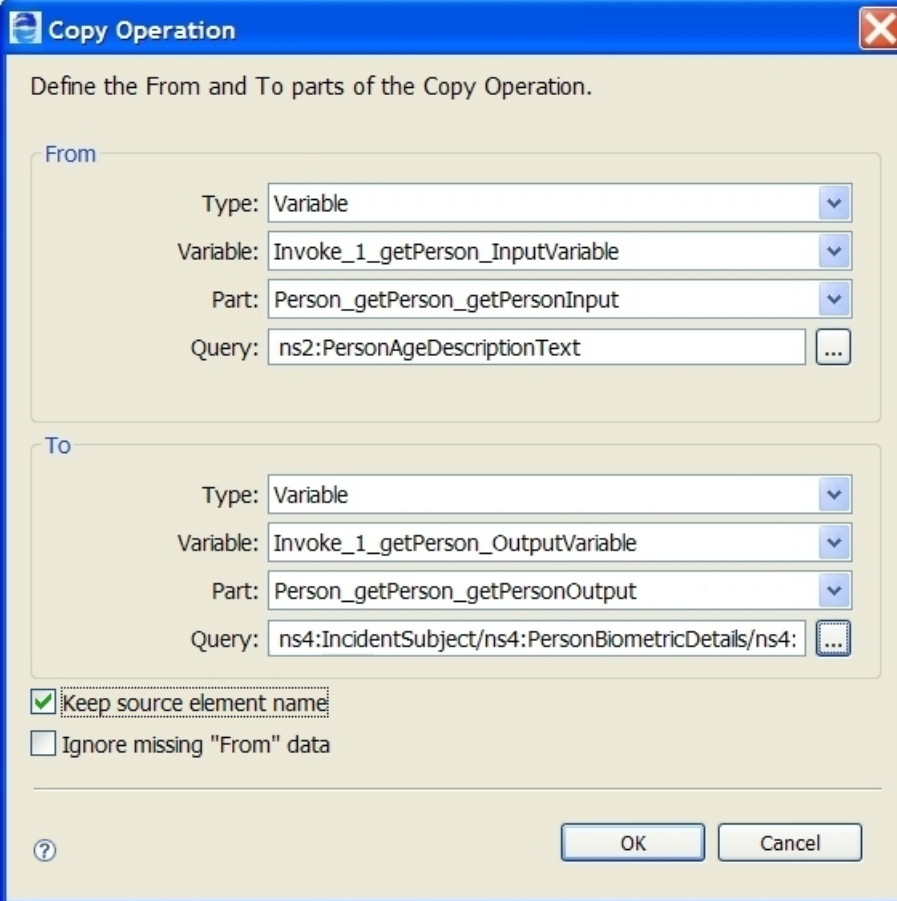
```
<assign>
  <copy>
    <from>
      <wsa:EndpointReference
        xmlns:s="http://www.active-endpoints.com/wsd/bpr-epr"
        xmlns:wsa=
          "http://www.w3.org/2005/08/addressing">
        <wsa:Address>anyURI</wsa:Address>
        <wsa:ServiceName>tns:Service</wsa:ServiceName>
      </wsa:EndpointReference>
    </from>
    <to partnerLink="mypartnerLink"/>
  </copy>
</assign>
```

</copy>
</assign>

ソース要素名属性を維持する要素間のコピー操作

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

[ソース要素名の維持] を使用して、コピー先の要素を、子と属性プロパティを含むソースの要素全体のコピーに置き換えます。この属性は、XSD 置換グループと選択をサポートします。

The image shows a 'Copy Operation' dialog box with a blue title bar and a close button. The main text says 'Define the From and To parts of the Copy Operation.' There are two sections: 'From' and 'To'. Each section has four fields: 'Type' (set to 'Variable'), 'Variable' (set to 'Invoke_1_getPerson_InputVariable' for From and 'Invoke_1_getPerson_OutputVariable' for To), 'Part' (set to 'Person_getPerson_getPersonInput' for From and 'Person_getPerson_getPersonOutput' for To), and 'Query' (set to 'ns2:PersonAgeDescriptionText' for From and 'ns4:IncidentSubject/ns4:PersonBiometricDetails/ns4:' for To). Below these sections are two checkboxes: 'Keep source element name' (checked) and 'Ignore missing "From" data' (unchecked). At the bottom are 'OK' and 'Cancel' buttons, and a help icon on the left.

[ソース要素名の維持] が無効な場合、コピー先（つまり、TO 側）の要素ベースの変数の結果の値には、元の名前空間とローカル名のプロパティが使用されます。

[ソース要素名の維持] が有効な場合、コピー先の要素ベースの変数の結果の値には、ソース（つまり、FROM 側）の要素ベースの変数の名前が使用されます。

次のスキーマスニペットがある場合:

```
<xsd:element name="poHeader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="shippingAddr"
          type="tns:AddressType" />
        <xsd:element name="USshippingAddr"
          type="tns:USAddressType" />
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```

        type="tns:AddressType" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

コピー操作の前に、次のプロセス変数値がある場合:

poHeaderVar1	poHeaderVar2
<tns:poHeader>	<tns:poHeader>
<tns:USshippingAddr verified="true">	
<tns:street>123 Main Street</tns:street>	
<tns:city>SomeWhere City</tns:city>	<tns:shippingAddr pobox="true" />
<tns:country>USA</tns:country>	
<tns:zipcode>98765</tns:zipcode>	
</tns:USshippingAddr>	
</tns:poHeader>	</tns:poHeader>

次のコピー操作がある場合:

```

<assign>
  <copy keepSrcElementName="yes">
    <from>$poHeaderVar1/tns:USshippingAddr</from>
    <to>$poHeaderVar2/tns:shippingAddr</to>
  </copy>
</assign>

```

poHeaderVar2 の結果の値:

```

<tns:poHeader>
  <tns:USshippingAddr verified="true">
    <tns:street>123 Main Street</tns:street>
    <tns:city>SomeWhere City</tns:city>
    <tns:country>USA</tns:country>
    <tns:zipcode>98765</tns:zipcode>
  </tns:USshippingAddr>
</tns:poHeader>

```

[見つからない開始データを無視] の属性を使用したコピー操作

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

ソース変数に存在しないデータをコピーした場合の選択エラーを回避するには、**[見つからない開始データを無視]** を使用します。コピー操作が有効である場合、ゼロの要素、属性、またはテキスト項目を選択する from-spec によって、コピー操作が無効になります。コピー操作の to-spec は評価されず、ターゲット変数は変更されません。この属性が有効ではない場合、from-spec によってゼロの項目が選択されると、通常の bpel:selectionFailure フォールトが発生します。

顧客レコードに小包配達の特典の指示が含まれている可能性がある以下のサンプルについて考えてみましょう。レコードにこれらの指示が存在する場合は、配送サービスに送信するデータにコピーする必要があります。指示が存在しない場合、「NONE」のデフォルトの指示はそのまま残ります。

ケース 1: オプションのデータが存在する

コピー操作前のサンプルデータ:

```
<CustomerRecord>
  <Id>100256</Id>
  <ShippingAddress>... </ShippingAddress>
  <DeliveryInstructions>
    Please leave at the back door
  </DeliveryInstructions>
</CustomerRecord>
<ShippingRecord>
  ...
  <SpecialHandling>NONE</SpecialHandling>
</ShippingRecord>
```

コピー操作:

```
<copy ignoreMissingFromData="yes">
  <from>$customerRecord/DeliveryInstructions/text()</from>
  <to>$shippingRecord/SpecialHandling</to>
</copy>
```

コピー操作後の結果:

```
<ShippingRecord>
  ...
  <SpecialHandling>Please leave at the back door
</SpecialHandling>
</ShippingRecord>
```

ケース 2: オプションのデータが存在しない

コピー操作前のサンプルデータ:

```
<CustomerRecord>
  <Id>100256</Id>
  <ShippingAddress...>... </ShippingAddress>
</CustomerRecord>
<ShippingRecord>
  ...
  <SpecialHandling>NONE</SpecialHandling>
</ShippingRecord>
```

コピー操作:

```
<copy ignoreMissingFromData="yes">
  <from>$customerRecord/DeliveryInstructions/text()</from>
  <to>$shippingRecord/SpecialHandling</to>
</copy>
```

コピー操作後の結果:

```
<ShippingRecord>
  ...
  <SpecialHandling>NONE
</SpecialHandling>
</ShippingRecord>
```

選択エラー設定の詳細については、「*Process Developer* の無効な選択エラーのフォールト拡張機能の使用」を参照してください。

Break

BPMN 実装: Terminate Throw イベント

これは、Process Developer の拡張アクティビティです。プロセスを保存すると、BPEL 仕様がないアクティビティがそのプロセスに含まれていることを通知するメッセージが表示されます。

デフォルトでは、Break アクティビティは、最も近いエンクローズされた Scope、While、または forEach アクティビティを終了します。コンテナ内で Break を使用することで、処理を中断できます。

Break を実行すると、Break アクティビティのターゲットに制御が移ります。ターゲットではそれ以上の反復が実行されず、正常に完了します。複数の Scope、While、または forEach アクティビティが相互にネストされている場合、Break アクティビティは最も内側のアクティビティにのみ適用されます。並列 forEach の場合、Break アクティビティにより、実行中のすべての反復によって実行中のすべての子アクティビティが終了され、正常に完了します。Break アクティビティにより、forEach のルートスコープが正常に完了し、補償が可能になります。Break アクティビティによる早期終了はフォールト状態とは見なされないため、補償が妨げられることはありません。

BPMN の Terminate プロパティを [いいえ] に設定することでループ（While、for Each、Repeat Until）から抜け出すための Break の動作を変更することはできますが、Scope から抜け出すことはできません。

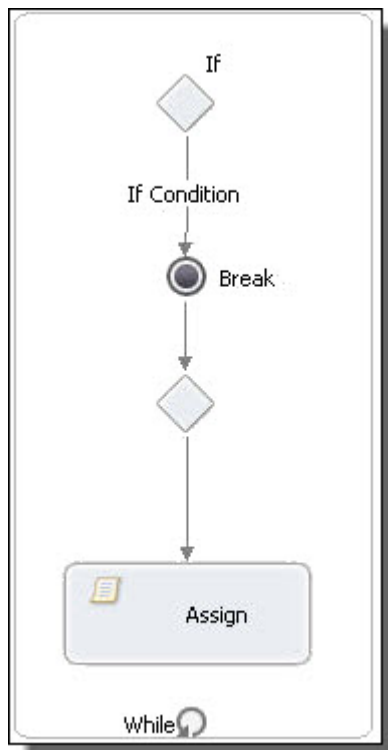
Break を使用すると、フォールトをスローしてプロセスを強制的に終了する代わりに通常どおり終了できます。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。
	<p>BPMN の終了。[はい]（デフォルト）と設定した場合、最も近い Scope ループ、While ループ、または for-Each ループのいずれか早いもので実行が停止します。Repeat-Until ループは（Scope のように）境界で表されておらず、BPMN の終了によってループとみなされないため、スキップされます。Scope またはループは正常に完了したと見なされます（フォールトをスローしてからキャッチする場合と同様です）。</p> <p>[いいえ] に設定した場合、ループ（While、for Each、Repeat Until）から抜け出しますが、Scope ではループから抜け出しません。</p>

Break アクティビティを追加する手順

1. Process Editor キャンバスで、Scope、While、または forEach コンテナを選択します。
2. **【Terminate Throw イベント】** をそのコンテナにドラッグします。

次の図は、例を示しています。



XML 構文

```

<break standard-attributes>
  standard-elements
</break>

```

例

```

<while>
  <targets><target linkName="Link2"/></targets>
  <sources><source linkName="Link5"/></sources>
  <condition>($counter < 5)</condition>
  <flow>
    <links>
      <link name="Link5"/>
    </links>
    <if>
      <sources><source linkName="Link5"/></sources>
      <condition>($counter = 3)</condition>
      <extensionActivity>
        ext:break name="Break"/>
      </extensionActivity>
    </if>
    <assign>
      <targets><target linkName="Link5"/></targets>
      <copy>
        <from>($counter + 1)</from>
        <to variable="counter"/>
      </copy>
    </assign>
  </flow>
</while>

```

Compensate

BPMN 実装: Compensate Throw イベント

Compensate アクティビティは、すでに正常に完了したすべての内部スコープで補償の開始をトリガします。補償ハンドラの説明については、[第 27 章, 「補償」 \(ページ 400\)](#)を参照してください。

Compensate アクティビティはスコープ名を指定しないため、すべての適格なスコープで、完了までとは逆の順序で補償ハンドラを実行する明示的なデフォルト順序の補償が提供されます。

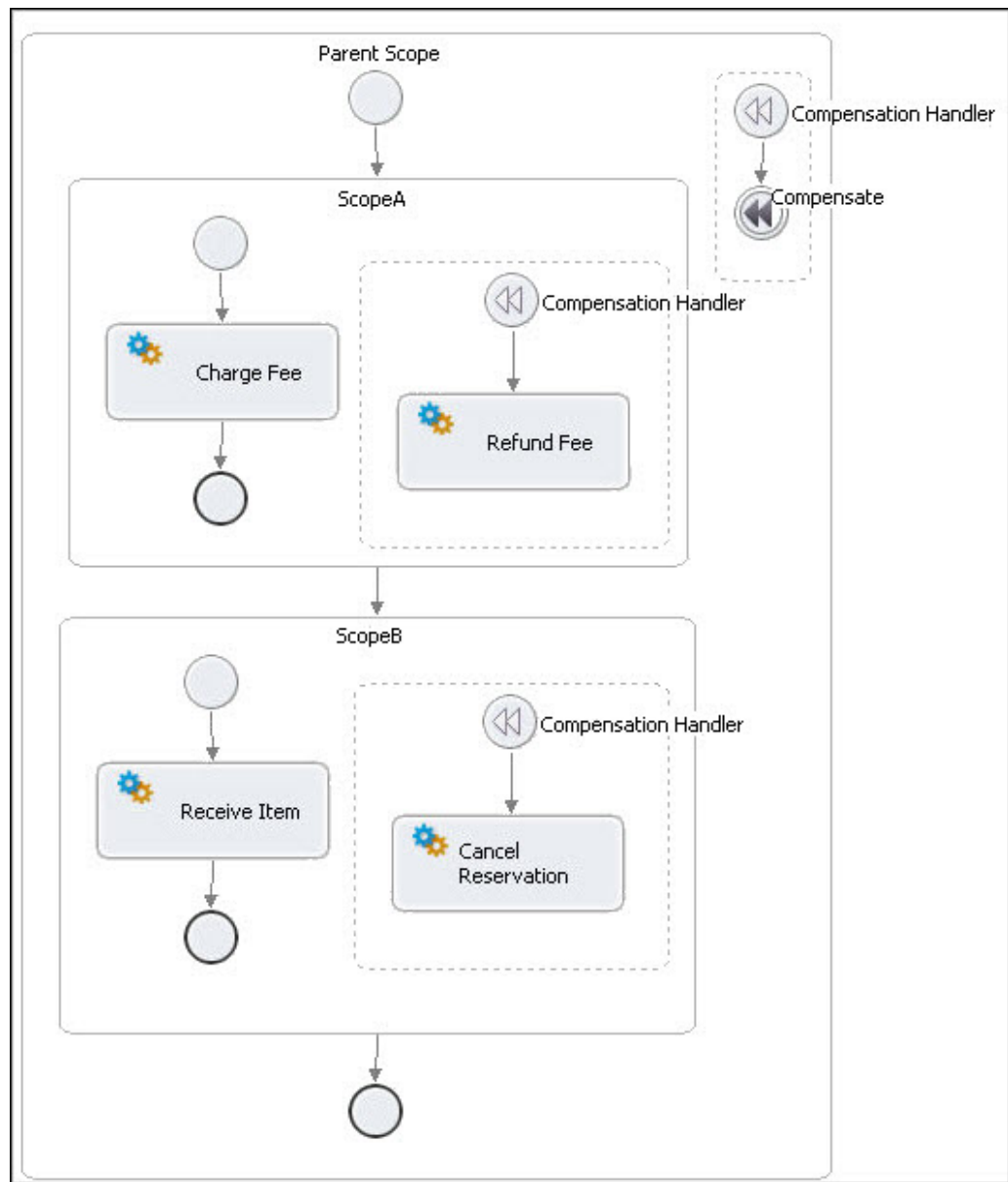
Compensate アクティビティは、補償ハンドラ、フォールトハンドラ、または終了ハンドラ内で定義します。また、Compensate アクティビティは補償境界イベントのターゲットになることもあります。

Compensate アクティビティは、2 つの補償アクティビティの 1 つです。[「Compensate Scope」 \(ページ 189\)](#) アクティビティは、補償用にエンクローズされたスコープに名前を付けます。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

Compensate アクティビティを作成する手順

1. Process Editor キャンバスで、補償ハンドラコンテナ、フォールトハンドラコンテナ、または終了ハンドラコンテナを持つスコープを選択します。
フォールトハンドラの場合、フォールトハンドラに Catch コンテナが含まれていることを確認します。
2. 次の例に示すように、**[Compensate Throw イベント]** をコンテナにドラッグします。



または、BPMN 補償境界イベントを使用して、Compensate アクティビティをそのターゲットにします。詳細については、[「境界イベントの追加」](#)（ページ 361）および [「Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow」](#)（ページ 365）を参照してください。

XML 構文

```

<compensate standard-attributes>
  standard-elements
</compensate>

```

例

```

<compensate/>

```

Compensate Scope

BPMN 実装: Compensate Throw イベント

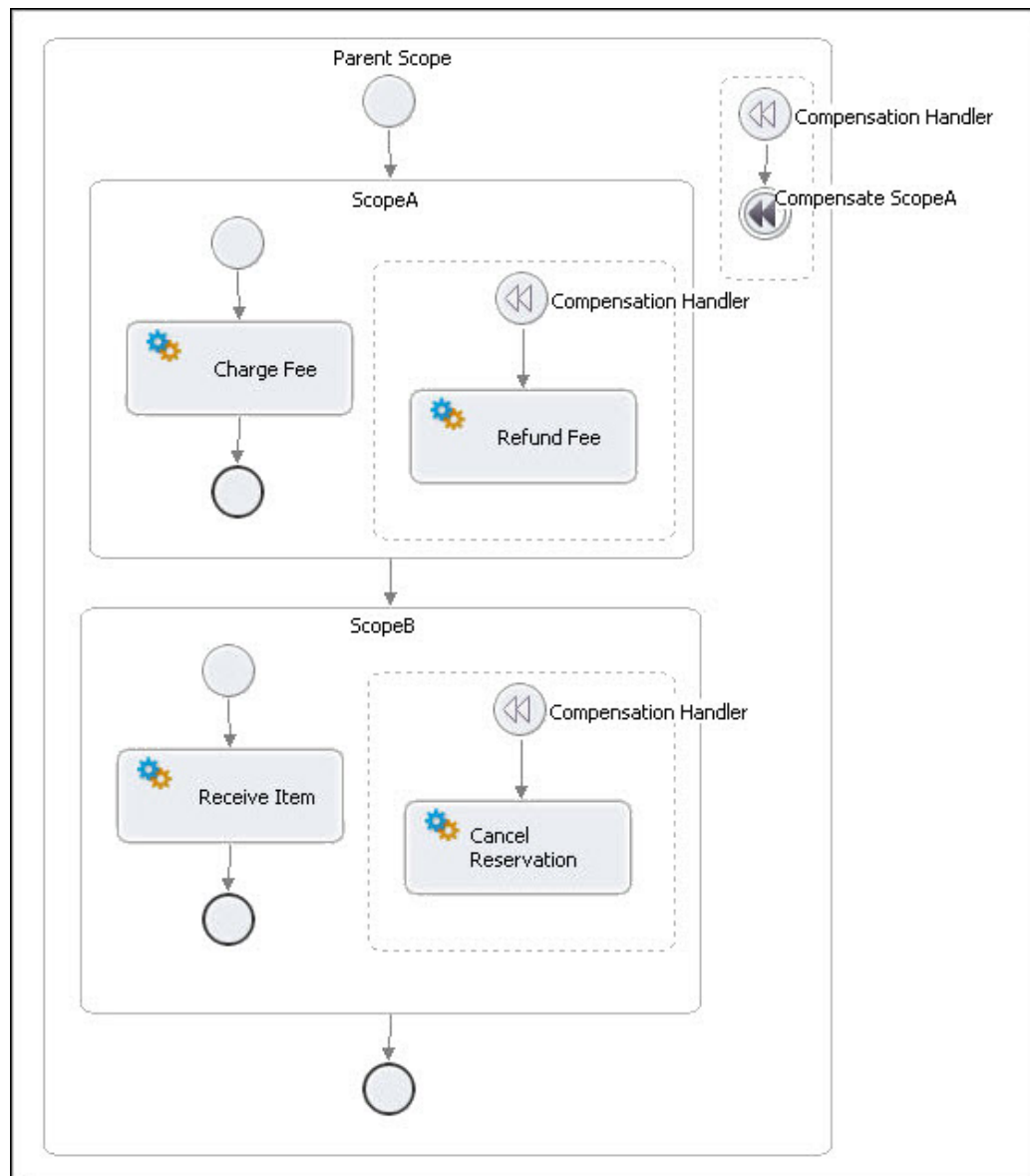
Compensate Scope アクティビティは、補償用にエンクローズされたスコープに名前を付け、スコープの補償ハンドラの開始をトリガします。補償ハンドラの説明については、[第 27 章, 「補償」 \(ページ 400\)](#)を参照してください。

Compensate Scope アクティビティは、2 つの補償アクティビティの 1 つです。[「Compensate」 \(ページ 187\)](#) アクティビティは、エンクローズされたスコープに名前を付けることはありませんが、すべての適格なエンクローズ済みのスコープで、完了までとは逆の順序で補償ハンドラを実行します。

必須のプロパティ	オプションのプロパティ
ターゲット	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

Compensate Scope アクティビティを作成する手順

1. Process Editor キャンバスで、補償ハンドラ、フォールトハンドラ、または終了ハンドラを持つスコープを選択します。
フォールトハンドラの場合、フォールトハンドラに Catch または Catch All コンテナが含まれていることを確認します。
2. 次の例に示すように、**[Compensate Throw イベント]** を Catch ハンドラ、Catch All ハンドラ、補償ハンドラ、または終了ハンドラにドラッグします。



「プロパティ」ビューで、「ターゲット」行の矢印を選択し、リストからスコープ名を選択します。このスコープは、Compensate Scope アクティビティに関連する内部スコープである必要があります。

使用可能なスコープ名がない場合、現在のスコープに Compensate Scope アクティビティを追加することはできません。

XML 構文

```

<compensateScope target="NCName" standard-attributes>
  standard-elements
</compensateScope>

```

例

```

<compensateScope target="assessorScope"/>

```

Continue

BPMN 実装: なし

これは、Process Developer の拡張アクティビティです。プロセスを保存すると、BPEL 仕様がないアクティビティがそのプロセスに含まれていることを通知するメッセージが表示されます。

Continue アクティビティは、While、Repeat Until、または forEach コンテナ内で使用できます。Continue アクティビティを実行すると、すぐ近くのエンクローズされたループアクティビティによって現在の反復の実行が停止され、次の反復に進みます。While の場合は、While の条件式が評価され、別の反復が可能かどうかを確認されます。シリアル forEach の場合、シーケンス内の次の反復が実行されるか、現在の反復が最後の反復である場合はループが完了します。並列 forEach の場合、Continue はエンクローズされた反復にのみ影響し、並列の反復は正常に完了します。

Continue アクティビティにより、forEach のルートスコープが正常に完了し、補償が可能になります。Continue アクティビティによる早期終了はフォールト状態とは見なされないため、補償が妨げられることはありません。

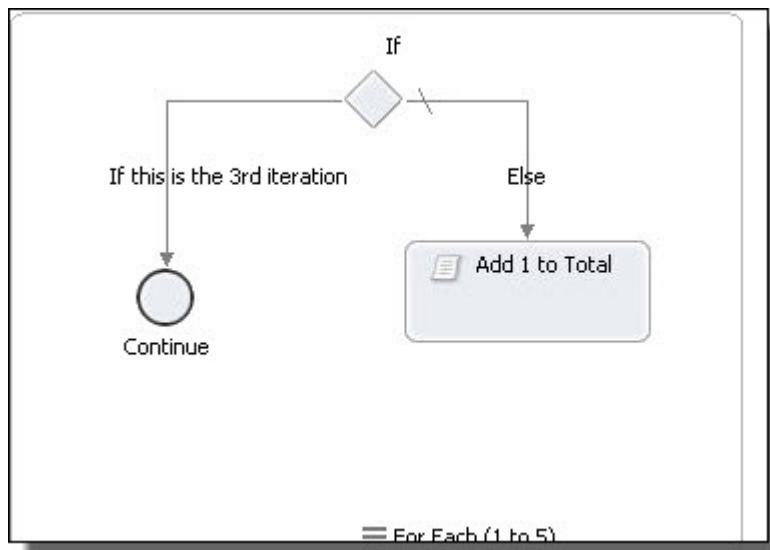
必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Continue アクティビティを追加する手順

Continue アクティビティを使用するには、BPEL 中心のパレットを使用する必要があります。

1. Process Editor キャンバスで、While、Repeat Until、または forEach コンテナを選択します。
2. **【イベント】** パレットから、Continue アクティビティをそのコンテナにドラッグします。

次の図は、例を示しています。



XML 構文

```

<continue standard-attributes>
  standard-elements
</continue>

```

Empty

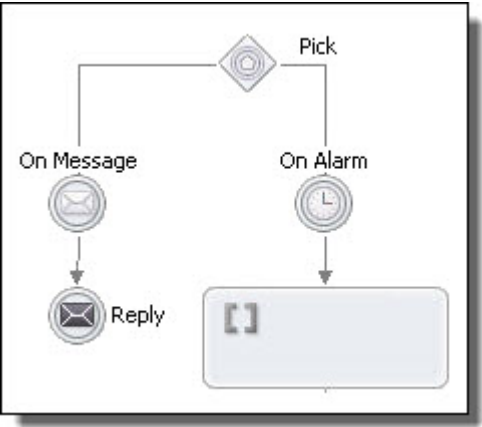
BPMN 実装: 抽象タスク、Start/End/None Throw イベント

Empty アクティビティとは、ビジネスプロセスにおける「操作なし」の命令です。このアクティビティは、同時アクティビティの同期に役立ちます。例えば、Empty アクティビティを使用して、リンクがどのように影響を受けるか、フォールトがキャッチされたかどうか、フォールトがどのスコープレベルでキャッチされたかなどを表示することができます。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Empty アクティビティを作成する手順

- 1. 抽象タスク（または Start/End/None Throw イベント）を Process Editor キャンバスにドラッグします。
背景色を抽象タスクに追加することはできませんが、Throw イベントに追加することはできません。
 - 2. アクティビティを適切なコンテナに追加するか、別のアクティビティをそのコンテナにリンクします。
- 次の図は、抽象タスクとして実装された Empty アクティビティの使用例を示しています。



XML 構文

```
<empty standard-attributes>  
  standard-elements  
</empty>
```

終了

BPMN 実装: Terminate Throw イベント

Exit アクティビティにより、ビジネスプロセスが停止します。これは、抽象的なプロセスではなく、実行可能なプロセス用に設計されています。現在実行中のすべてのアクティビティは、フォールト処理や補償なしで直ちに終了する必要があります。

Exit 終了アクティビティを伴うプロセスの終了は、異常な終了です。プロセスは、Process Developer のシステムフォールト processTerminated で終了します。

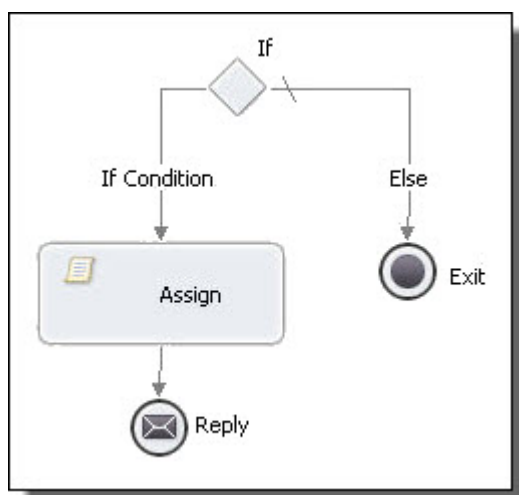
必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)

必須のプロパティ	オプションのプロパティ
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Exit アクティビティを作成する手順

1. **Terminate Throw** イベントを Process Editor キャンバスにドラッグします。
2. アクティビティを適切なコンテナに追加するか、別のアクティビティをそのコンテナにリンクします。

次の図は、Exit アクティビティの使用例を示しています。



XML 構文

```

<exit standard-attributes
      standard-elements
/>

```

Invoke

BPMN 実装: 送信タスク、サービスタスク、ルールタスク、メッセージスローイベント

Invoke アクティビティは、Web サービスに一方向の操作または要求/応答操作を実行するように指示します。Invoke アクティビティは、サービスを提供するパーティシパントおよび呼び出す操作を指定します。Invoke アクティビティは、送信するメッセージのデータを指定する必要があり、同期要求/応答 Web サービス呼び出しの場合は、出力変数または変数部分が指定されます。このアクティビティに対する重要な概念の説明については、[第 10 章, 「パーティシパント」](#) (ページ 138)を参照してください。

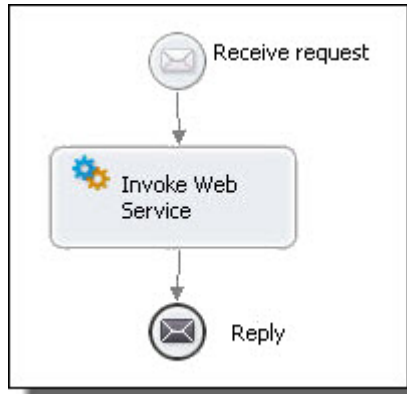
Invoke アクティビティは、補償したり、元に戻したりすることができます。詳細については、[第 27 章, 「補償」 \(ページ 400\)](#)を参照してください。

必須のプロパティ	オプションのプロパティ
パーティシパント (パートナーリンク)	名前。 「アクティビティラベルの選択」 (ページ 158) 「アクティビティラベルの選択」 (ページ 158) を参照してください。
操作	ポートタイプ
入力変数 「入力変数」 (ページ 197) を参照してください。 または toPart fromVariable 「変数からパート」 (ページ 197) を参照してください。	相関。 第 24 章, 「相関」 (ページ 367) を参照してください。
	出力変数 「出力変数」 (ページ 199) および 「開始パートから変数」 (ページ 197) を参照してください。
	入力割り当てと出力割り当て
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

プロセスに Invoke アクティビティを手動で追加する手順

1. **【送信タスク】** (または上記の他の BPMN 実装) を Process Editor キャンパスにドラッグします。
2. 送信タスク、サービスタスク、およびルールタスクには、背景色を追加できます。
[プロパティ] ビューで、次の値を選択します。
 - a. 必要に応じて名前を入力します。
 - b. [パーティシパント] ドロップダウンで、**【新規パートナーサービスプロバイダ】** を選択します。[「新しいパートナーサービスインタフェースの作成」 \(ページ 141\)](#)を参照してください。
 - c. 選択リストからパーティシパントの [操作] を選択します。

3. [入力] タブで、次のいずれかを実行します。
 - **[割り当てタイプ]** で [単一変数] を選択してから、変数を選択します。
 - **[XPath]** または **[XQuery]** を選択します。詳細については、[「入力変数」 \(ページ 197\)](#)を参照してください。
 4. 必要に応じて、出力変数または終了パートを選択します。詳細については、[「出力変数」 \(ページ 199\)](#)を参照してください。
 5. 必要に応じて、他のオプションのプロパティを選択します。
- 次の図に、プロセスで Invoke アクティビティを使用する簡単な例を示します。



Invoke アクティビティを作成するための他のショートカットについては、[「WSDL インタフェースから開始するアクティビティの作成」 \(ページ 160\)](#)を参照してください。

XML 構文

```

<invoke partnerLink="NCName" portType="QName"? operation="NCName"
  inputVariable="NCName"? outputVariable="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"?
      pattern="request|response|request-response"/>+
  </correlations>
  <catch faultName="QName"? faultVariable="NCName"?
    faultMessageType="QName"? faultElement="QName"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
  <toParts>?
    <toPart part="NCName" fromVariable="BPELVariableName"/>+
  </toParts>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName"/>+
  </fromParts>
</invoke>
  
```

例:

```

<invoke name="invokeapprover" partnerLink="approver"
  portType="lns:loanApprovalPT" operation="approve"
  inputVariable="request" outputVariable="approval">
  
```

開始パートから変数

変数パートを選択し、プロセス変数に割り当てます。[新規] を選択して、リストに追加する開始パートまたは終了パートの仕様を作成します。

Receive、onMessage (Pick アクティビティ)、onEvent (イベントハンドラ)、Invoke、およびユーザーアクティビティといった各アクティビティは、メッセージパートをプロセス変数にコピーすることができるため、コピーを実行するために Assign アクティビティを作成する必要はありません。Receive、onMessage、および onEvent アクティビティの変数の代わりに、fromPart を指定できます。また、Invoke またはユーザーアクティビティの出力変数の代わりに fromPart を指定することもできます。

WSDL 操作で、要素を使用して定義した 1 つのパートのみを含むメッセージが使用される場合、Web サービスインタラクションアクティビティでは、アクティビティの変数に fromPart 要素が使用されます。アクティビティの変数宣言の代わりに、fromPart 宣言が使用されます。Invoke またはユーザーアクティビティの場合、fromPart 宣言で出力変数が置き換えられます。

スコープ内の変数のリストからパートと変数を選択します。onEvent アクティビティの場合、変数名を指定すると変数が暗黙的に作成されます。

次の例は、fromPart 割り当ての XML ソースを示しています。

```
<onEvent operation="asyncOp" partnerLink="requestPLT">
  <fromParts>
    <fromPart part="one" toVariable="myNewVariable"/>
  </fromParts>
  <scope/>
</onEvent>
```

変数からパート

[新規] を選択して、リストに追加する開始パートまたは終了パートの仕様を作成します。または、プロセス変数を選択して、変数パートに割り当てます

Reply および Invoke アクティビティは、プロセス変数をメッセージパートにコピーすることができるため、コピーを実行するために Assign アクティビティを作成する必要はありません。Invoke の場合は、入力変数の代わりに toPart を指定でき、Reply の場合は、変数の代わりに toPart を指定できます。

WSDL 操作で、要素を使用して定義した 1 つのパートのみを含むメッセージが使用される場合、Web サービスインタラクションアクティビティでは、出力変数に toPart 要素が使用されます。アクティビティの出力変数宣言の代わりに、toPart 宣言が使用されます。

次の例は、toPart 割り当ての XML ソースを示しています。

```
<invoke outputVariable="twoWayRequest"
  name="InvokeWithManagedCorrelation" operation="asyncOp"
  partnerLink="requestPLT" portType="ns1:asyncRequestPT">
  <toParts>
    <toPart fromVariable="V1" part="one"/>
  </toParts>
</invoke>
```

入力変数

割り当てタイプ

- **単一の変数。** Assign アクティビティを介した入力への準備が必要ない場合は、変数を選択します。既存のプロセス変数を選択するか、[新規変数] を選択します。変数は操作の入力メッセージタイプである必要があります。または、操作に 1 つのパートのみが必要な場合は、変数のタイプを 1 つのパートの要素タイプにすることができます。
- **XPath。** テーブルを使用して既存の変数またはリテラル内のデータから入力メッセージの内容にマッピングする場合は、XPath を選択します。このテーブルで XPath 式を使用することで、変数からノードを選択し

て入力変数に割り当てたり、変数の内容から値（文字列、数値、またはブール値）を計算したりすることができます。以下の「**XPath の追加**」を参照してください。

- **XQuery**。それぞれの入力メッセージパートに XQuery 式言語で記述された単一のクエリを入力する場合は、XQuery を選択します。この言語を使用すると、XML ドキュメントのような外見を持つクエリを作成できますが、ドキュメントの一部は XPath 式構文のスーパーセットを使用して作成されます。これは、結合を実行するための SQL のような「FLWOR 式」（FOR、LET、WHERE、ORDER BY、および RETURN）で XPath を補足します。以下の「**XQuery の追加**」を参照してください。

XPath および XQuery 式の例については、「[式構築のための XPath または XQuery の選択](#)」（ページ 273）を参照してください。

ヒント: Javascript を使用して入力メッセージの一部またはすべてを準備する場合は、[入力] タブを使用する代わりに、Assign アクティビティを作成します。割り当ての各コピー操作では、異なる表現言語を使用することができます。

XPath の追加

1. テーブル内で、**[追加]** を選択して **b** カラムに入力します。
2. 必要に応じて、[パス] カラムで、パートの子ノードを選択します。
3. [E/L] カラムで、[From] タイプに [式] または [リテラル] を選択します。
4. [From] カラムで、テーブルセルの最後にある [ダイアログ (...)] ボタン] を選択します。リテラルの場合は、**[生成]** を選択し、選択した [終了パート] および [終了パス] に必要なタイプに基づいてリテラル XML ドキュメントが生成されることに注意してください。式を追加するには、[式] テキストボックスに入力します。ショートカットについては、「[コンテンツアシストの使用](#)」（ページ 277）を参照してください。

式言語の詳細については、「[式ビルダの使用](#)」（ページ 275）を参照してください。

リテラルマッピングの例を以下に示します。

Invoke	Assignment Type: XPath			
Input				
Output	E/L	From	To Part	To Path
All		<div> <div><ns2:reportMessageInput xmlns:ns2="http://schemas.active-endpoints.com/reporting/2009/05/reporting.xsd"></div> <div><ns2:adminConsole host="localhost" port="8080"/></div> <div><ns2:report>string</ns2:report></div> <div><ns2:format>pdf</ns2:format></div> <div><ns2:parameters></div> <div><ns2:param name="string" value="string"/></div> <div></ns2:parameters></div> <div><ns2:otherOptions></div> </div>	reportInputPart	

5. 必要に応じて、**[添付のコピー]** で、コピー先またはコピー元とする添付ファイルを持つ変数を選択します。すべての添付ファイルをコピーする必要があることに注意してください。一部の添付ファイルのみが必要な場合は、アクティビティの前に Assign アクティビティを追加できます。詳細については「[添付ファイルの追加](#)」（ページ 259）を参照してください。
6. XPath を使用すると、パラメータと呼ばれる、各パートが含まれた一時変数が作成されます。詳細については、以下の「[コンテナ変数の表示](#)」を参照してください。

XQuery の追加

XQuery の開発を簡単に行う方法については、「[XQuery 関数の記述](#)」（ページ 299）を参照してください。

1. Invoke、Reply、またはユーザーアクティビティの [入力（またはデータ）] タブで、[割り当てタイプ] が [XQuery] であることを確認します。
メッセージパートに対して XML テンプレートが生成されることに注意してください。必要に応じて、ドキュメントの編集後に、テンプレートを再生成できます。

2. 各要素に、必要な XQuery 式を入力します。必要に応じて、**ビルダ** を選択してクエリビルダを開き、式または関数をクエリに挿入します。
XQuery マッピングの例を以下に示します。

Invoke	Assignment Type: XQuery
Input	
Output	Part: emailPart Regenerate Builder...
All	<pre> <aem:emailMessage xmlns:aem="http://schemas.active-endpoints.com/email/2007/01/email.xsd"> <aem:from>{ \$fill_in_here }</aem:from> <aem:replyTo>{ \$fill_in_here }</aem:replyTo> { for \$to in \$fill_in_here return <aem:to>{ \$fill_in_here }</aem:to> } { for \$cc in \$fill_in_here return <aem:cc>{ \$fill_in_here }</aem:cc> } { for \$bcc in \$fill_in_here </pre>

3. 必要に応じて、上記の XPath の説明に従って、添付ファイルをコピーします。

コンテナ変数の表示

XPath および XQuery を使用している場合、パラメータと呼ばれるコンテナ変数が [プロセス変数] ビューに追加されます。同様に、添付ファイルを入力変数にコピーしている場合は、コンテナ変数 attachmentCopyResult が表示されます。これらの変数は、実行時に完全な変数の一部を処理するために一時的に使用されるものであるため、デフォルトでは非表示になっています。これらの変数を表示するには、**内部変数の表示** を選択します。

出力変数

割り当てタイプ

- **単一の変数**。リストから変数を選択して、アクティビティの出力をその変数に加えます。変数は、操作の出力メッセージタイプになります。または、操作の応答に 1 つのパートしかない場合、変数のタイプはその 1 つのパートの要素タイプになります。既存のプロセス変数を選択するか、**新規変数** を選択します。
- **パートから変数**。それぞれの応答メッセージパートを異なる変数にマッピングする場合は、**パートから変数** を選択します。詳細については、**「開始パートから変数」 (ページ 197)** を参照してください。
- **XPath**。テーブルを使用して既存の変数内のデータから受信メッセージの内容にデータをマッピングする場合は、**XPath** を選択します。このテーブルで XPath 式を使用することで、変数からノードを選択して入力変数に割り当てたり、変数の内容から値 (文字列、数値、またはブール値) を計算したりすることができます。**「入力変数」 (ページ 197)** のトピックにある **「XPath の追加」** を参照してください。

コンテナ変数の表示

XPath を使用する場合、result と呼ばれるコンテナ変数が [プロセス変数] ビューに追加されます。同様に、入力変数に添付ファイルをコピーすると、コンテナ変数 attachmentCopyResult が表示されます。これらの変数は、実行時に完全な変数の一部を処理するために一時的に使用されるものであるため、デフォルトでは非表示になっています。これらの変数を表示するには、**内部変数の表示** を選択します。

あいまい

Process Developer Classic のみ

このアクティビティは、抽象プロセスでのみ使用されます。抽象プロセスの作成に関する詳細については、[「抽象プロセスの作成」 \(ページ 134\)](#)を参照してください。

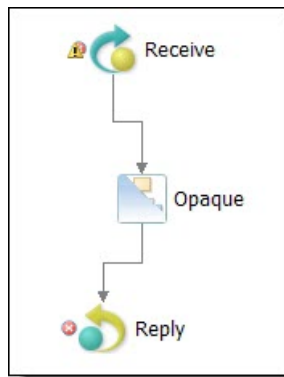
このアクティビティは、実行可能なプロセスで使用されるアクティビティを表します。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

あいまいなアクティビティを作成する手順

1. BPEL プロセスが抽象プロセスの名前空間を参照していることを確認します。デフォルトでは、BPEL プロセスは実行可能です。抽象プロセスを使用するためのヒントを参照してください。
2. **【アクティビティ】** パレットから、あいまいなアクティビティを Process Editor キャンバスにドラッグします。
3. 必要に応じて、アクティビティを適切なコンテナに追加するか、別のアクティビティをそのコンテナにリンクします。

次の図は、あいまいなアクティビティの使用例を示しています。



XML 構文

```
<opaqueActivity standard-attributes>  
  standard-elements  
</opaqueActivity>
```

例:

```
<opaqueActivity/>
```

Receive

BPMN 実装: 受信タスク、メッセージキャッチイベント

BPEL 処理エンジンはメッセージを受信すると、一致するパートナーリンクと操作を使用して Receive (または Pick) アクティビティを検索します。実行可能プロセスの場合、Receive によって、受信したメッセージデータの変数または変数パートが指定されます。このアクティビティに対する重要な概念の説明については、[「パートナーリンク」 \(ページ 149\)](#)を参照してください。

[インスタンスの作成] プロパティを [はい] に設定することで、Receive アクティビティでビジネスプロセスインスタンスを開始できます。操作が要求/応答である場合、Receive アクティビティを Reply アクティビティに関連付けることができます。さらに、Receive にはメッセージ交換属性を含めることができます。

同時初期受信のセットを作成できます。この場合、セットの任意の 1 つのメッセージで BPEL プロセスを初期化できます。複数のアクティビティの開始ポイントを作成するには、Receive を Flow コンテナに追加します。また、[「アクティビティへの関連の追加」 \(ページ 378\)](#)に記載されているように、プロセスを開始するすべての同時受信に関連セットを提供する必要があります。

Pick アクティビティの onMessage 句や onEvent イベントハンドラといった他の 2 つのアクティビティは、Receive に類似しています。詳細については、[「Pick」 \(ページ 231\)](#)および [第 23 章, 「イベント処理」 \(ページ 355\)](#)を参照してください。

必須のプロパティ	オプションのプロパティ
パーティシパント (パートナーリンク)	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
操作	ポートタイプ
変数 (抽象プロセスではなく、実行可能プロセスにのみ必要) または 開始パートから変数。 「開始パートから変数」 (ページ 197) を参照してください。	関連。 第 24 章, 「関連」 (ページ 367) を参照してください。
	インスタンスの作成。これが Start アクティビティである場合は必須です。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 第 9 章, 「BPEL プロセス」 (ページ 120) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。

必須のプロパティ	オプションのプロパティ
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	メッセージ交換。 「メッセージ交換の宣言」 (ページ 136) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

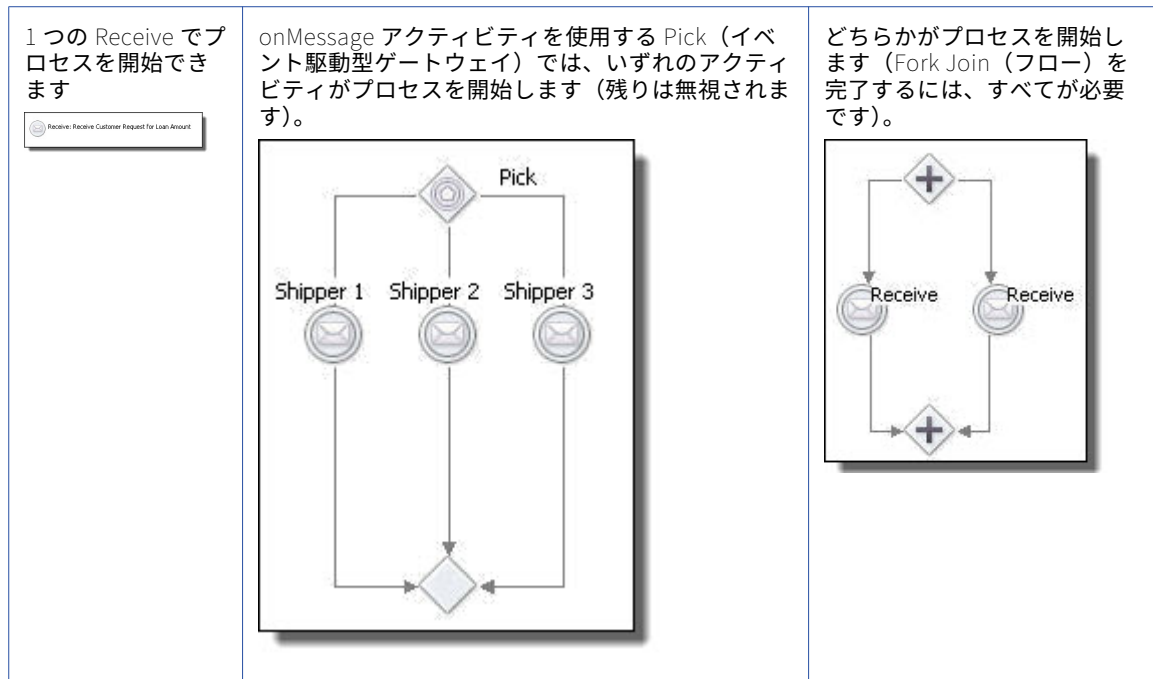
Receive アクティビティをプロセスに手動で追加する手順

ショートカットと推奨される手法については、[「WSDL インタフェースから開始するアクティビティの作成」 \(ページ 160\)](#)を参照してください。

1. **【受信タスク】** または **【メッセージキャッチイベント】** アクティビティを Process Editor キャンバスにドラッグします。
受信タスクに背景色を追加することはできますが、メッセージキャッチイベントに背景色を追加することはできません。
2. **【プロパティ】** ビューで、次の値を選択します。
 - a. 必要に応じて名前を入力します。
 - b. **【パーティシパント】** ドロップダウンで、**【新規プロセスサービスコンシューマ】** を選択します。[「新しいパートナーサービスインタフェースの作成」 \(ページ 141\)](#)を参照してください。
 - c. 選択リストから **【操作】** を選択します。
3. **【データ】** タブで、次のいずれかを実行します。
 - 割り当てタイプから単一の変数を選択し、新しい変数を作成するか、プロセス変数を選択する。
 - **【パートから変数】** 仕様を作成する。詳細については、[「開始パートから変数」 \(ページ 197\)](#)を参照してください。
 - XPath 仕様を作成する。詳細については、[「入力変数」 \(ページ 197\)](#)を参照してください。
4. 必要に応じて、他のオプションのプロパティを選択します。

Receive アクティビティは、実際にはスコープ内での Receive とそれに続く Assign という 2 つのアクティビティです。これらの 2 つのアクティビティはアトミックではなく、例えば、イベントハンドラで onAlarm を初期化する場合、Assign が完了するまで変数を使用できません。

使用例



XML 構文

```
<receive partnerLink="NCName" portType="QName"?
  operation="NCName"
  variable="BPELVariableName"? createInstance="yes|no"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"?>+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName"/>+
  </fromParts>
</receive>
```

XML の例:

```
<receive name="ReceiveCustomerRequest"
  partnerLink="customer"
  portType="lns:loanServicePT"
  operation="request"
  variable="request"
  createInstance="yes"/>
```

Reply

BPMN 実装: 送信タスク、メッセージキャッチイベント

Reply アクティビティは、受信したメッセージへの応答として、またはフォールト処理を通じて、メッセージを Web サービスに送り返します。受信したメッセージは、Receive アクティビティ、onMessage ハンドラ、onEvent ハンドラ、または Invoke に表示されます。要求/応答操作の場合、Reply アクティビティには、対応する Receive、onMessage、または onEvent と同じパートナーリンクおよび操作があります。

Receive には、Receive アクティビティを参照する複数の Reply アクティビティが含まれる場合がありますが、実行できるのはそのうちの 1 つだけです。例えば、通常の応答とフォールト処理の応答が含まれている場合があります。両方の応答はパートナーリンクと操作を受信と共有しますが、実行できるのはそのうちの 1 つだけです。

実行可能なプロセスの場合、Reply アクティビティでは、送信されるメッセージデータの変数を指定する必要があります。

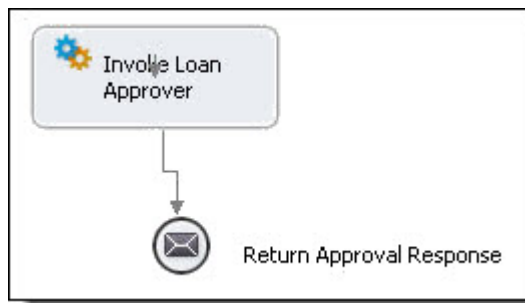
必須のプロパティ	オプションのプロパティ
パーティシパント（パートナーリンク）	名前。 「アクティビティラベルの選択」 （ページ 158）を参照してください。
操作	ポートタイプ
変数 「入力変数」 （ページ 197）を参照してください。 または toPart fromVariable 「変数からパート」 （ページ 197）を参照してください。	第 24 章, 「関連」 （ページ 367）
	フォルト名
	結合条件。 「入力リンクに対する結合条件の作成」 （ページ 292）を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 （ページ 130）を参照してください。
	コメント。 「プロセスへのコメントの追加」 （ページ 79）を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 （ページ 79）を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 （ページ 76）
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 （ページ 419）を参照してください。
	メッセージ交換。 「メッセージ交換の宣言」 （ページ 136）を参照してください。
	拡張属性と拡張要素。 「メッセージ交換の宣言」 （ページ 136）を参照してください。

Reply アクティビティをプロセスに手動で追加する手順

ショートカットと推奨される手法については、[「WSDL インタフェースから開始するアクティビティの作成」](#)（ページ 160）を参照してください。

1. **【送信タスク】** または **【メッセージスローイベント】** を Process Editor キャンバスにドラッグします。
送信タスクに背景色を追加することはできますが、メッセージスローイベントに背景色を追加することはできません。

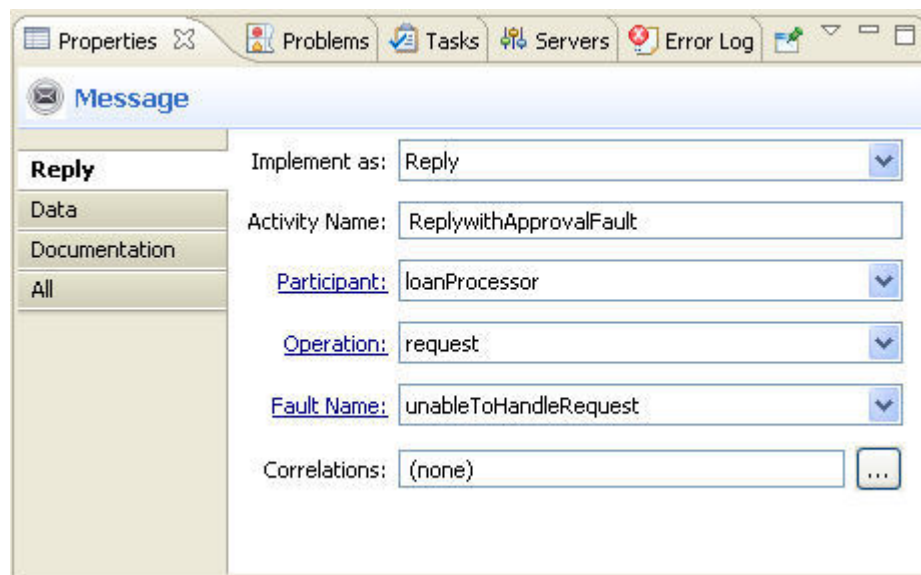
2. [プロパティ] ビューで、次の値を選択します。
 - a. 必要に応じて名前を入力します。
 - b. [パーティシパント] ドロップダウンで、**[新規プロセスサービスコンシューマ]** を選択します。[「新しいプロセスサービスコンシューマインタフェースの作成」 \(ページ 140\)](#)を参照してください。
 - c. 選択リストから **[操作]** を選択します。
 3. [データ] タブで、次のいずれかを実行します。
 - [割り当てタイプ] から [単一の変数] を選択して変数を選択するか、[新規変数] を選択します。
 - XPath または XQuery を選択します。詳細については、[「入力変数」 \(ページ 197\)](#)および [「開始パートから変数」 \(ページ 197\)](#)を参照してください。
 4. 必要に応じて、他のオプションのプロパティを選択します。
- 次の図に、プロセスで Reply アクティビティを使用する簡単な例を示します。



フォールト応答を含む Reply アクティビティを追加する手順

1. Catch コンテナに Reply を追加します。
2. 上記の手順の説明に従って、プロパティと名前を選択します。
3. リストからフォールト名を選択します。フォールト名は WSDL ファイルで定義されています。
4. 変数リストからフォールト変数を選択します。

次の図は、フォールトが発生した応答の例を示しています。フォールト名はメインタブに表示されます。変数は [データ] タブで指定します。



次の図は、Catch ハンドラでフォールトが発生した応答を使用する簡単な例を示しています。



XML 構文

```
<reply partnerLink="NCName" portType="QName"?
  operation="NCName"
  variable="BPELVariableName"? faultName="QName"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"?>+
  </correlations>
  <toParts>?
    <toPart part="NCName" fromVariable="BPELVariableName"/>+
  </toParts>
</reply>
```

例 1 通常の応答:

```
<reply name="reply" partnerLink="customer"
  portType="lns:loanServicePT" operation="request"
  variable="approval">
```

例 2 フォールト処理:

```
<reply name="reply" partnerLink="customer"
  portType="lns:loanServicePT" operation="request"
  variable="error" faultName="unableToHandleRequest">
```

Rethrow

BPMN 実装: Error Throw イベント

Rethrow アクティビティは、直ちにエンクロージングするフォールトハンドラによって最初にキャッチされたフォールトを親スコープに渡します。Rethrow アクティビティは、フォールトハンドラの Catch および Catch All 要素内でのみ使用できます。このアクティビティは常に元のフォールトデータをスローし、そのタイプを保持します。

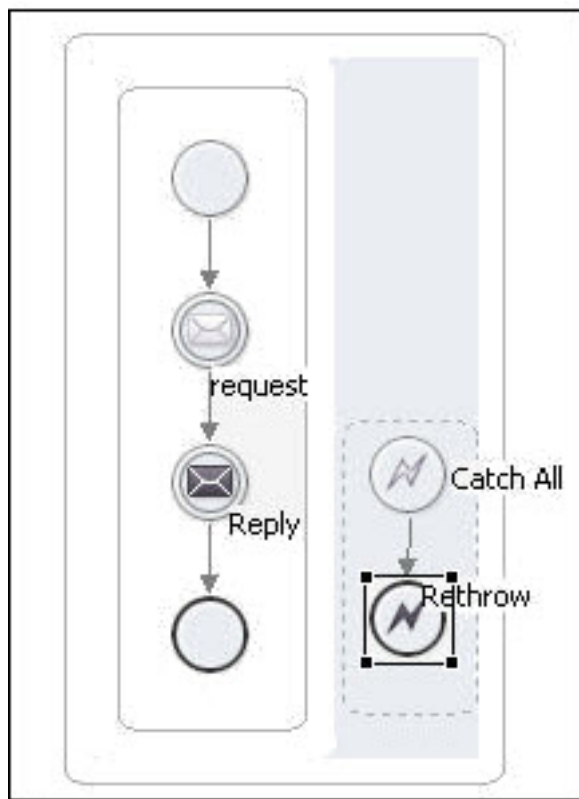
必須のプロパティ	オプションのプロパティ
なし	名前。『 アクティビティラベルの選択 』 (ページ 158)を参照してください。
	結合条件。『 入力リンクに対する結合条件の作成 』 (ページ 292)を参照してください。
	結合の失敗の非表示。『 プロセス要素とプロパティ 』 (ページ 130)を参照してください。

必須のプロパティ	オプションのプロパティ
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

プロセスに Rethrow アクティビティを追加する手順

1. Process Editor の [フォールトハンドラ] タブまたはスコープのフォールトハンドラで、**Catch** または、**Catch All** コンテナを追加します。
2. **[Error Throw イベント]** を Catch または Catch All コンテナにドラッグします。

次の図は、スコープの Catch All フォールトハンドラ内で Rethrow アクティビティを使用する簡単な例を示しています。



ヒント: フォールトハンドラを折りたたむと、[プロパティ] ビューで背景色を追加できます。

XML 構文

```
<rethrow standard-attributes
          standard-elements
>/rethrow>
```

例:

```
<rethrow/>
```

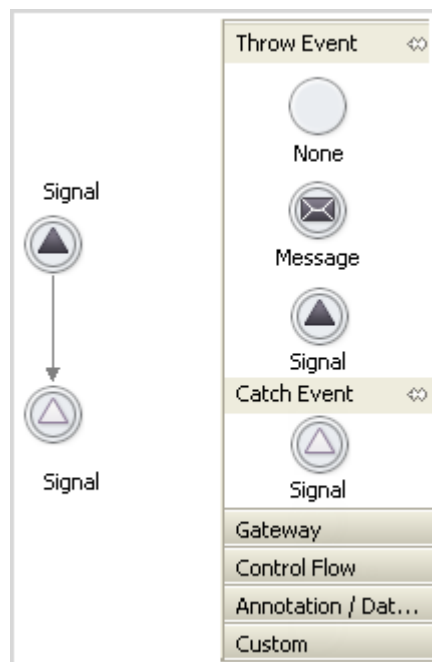
シグナル

シグナルはユーザーが作成するイベントで、プロセス内で情報をスローし、プロセス内の場所に関わらず受信できます。シグナルイベントのキャッチは、シグナルの名前のみを指定する必要があることを除き、メッセージまたはアラームイベントをキャッチすることと同様です。これは、インライン（待機など）、Pick の代替手段（onAlarm など）、ハンドラ（onAlarm など）、または境界イベントの使用によってキャッチできることを意味します。

注: これは、Process Developer の拡張アクティビティです。

メッセージ（シグナルと類似）とは異なり、シグナルは複数のアクティビティで受信されます。また、メッセージの送信先のアクティビティがメッセージによって認識されている必要があるという点が異なります。メッセージとは対照的に、キャッチシグナルはブロードキャストされ、ブロードキャストされたシグナルのうち、どれを処理するかを決定します。

プロセスにシグナルを追加するには、[シグナル] アイコンを Process Editor キャンバスにドラッグします。シグナルは、Throw および Catch イベントパレットにあります。



スローシグナルをキャンバスにドラッグした後に、シグナル名を割り当てる必要があります。シグナル名はアクティビティ名とは異なるため、混同しないようにしてください。キャッチシグナルのプロパティを定義する際に、キャッチするシグナルをこの名前で識別します。プロセスは複数のシグナルを持つことができるため、この名前が必要となります。

シグナルがスローされると、その名前のシグナルを待機しているアクティブなすべてのシグナルハンドラが実行され、現在のトランザクションが終了する前にすべてのシグナルでこれが実行されます。

シグナルのプロパティの定義については、[「アクティビティとそのプロパティの定義」](#)（ページ 156）を参照してください。

既存のワークスペースを使用しており、パレットをカスタマイズしている場合は、シグナルイベントがそのワークスペースに表示されない場合があります。このような問題が発生した場合は、シグナルイベントエントリをワークスペースにコピーするか、ワークスペースのカスタマイズされたパレットを削除してください。これは、パレットに表示される新しいワークスペースのシグナルイベントを作成する場合には発生しません。

Start/End/None

BPEL 実装: Empty または Assign

None イベントは、制御フローの開始時と終了時にフローの境界を示す場合に役立ちます。Start イベントにはトリガはありません。End イベントには結果はありません。

If アクティビティの Else If ブランチなど、中間イベントを制御フロー内に追加することができます。

Suspend

これは、Process Developer の拡張アクティビティです。プロセスを保存すると、BPEL 仕様がないアクティビティがそのプロセスに含まれていることを通知するメッセージが表示されます。

Suspend アクティビティは、実行中のプロセスを一時停止します。このアクティビティは、予期しないフォールトをキャッチして、エラーを表示および修正するために手動で介入できるようにする場合に役立ちます。オプションの変数属性を使用することで、自動およびプログラムによる一時停止でトリガ可能なアラートにデータを添付できます。

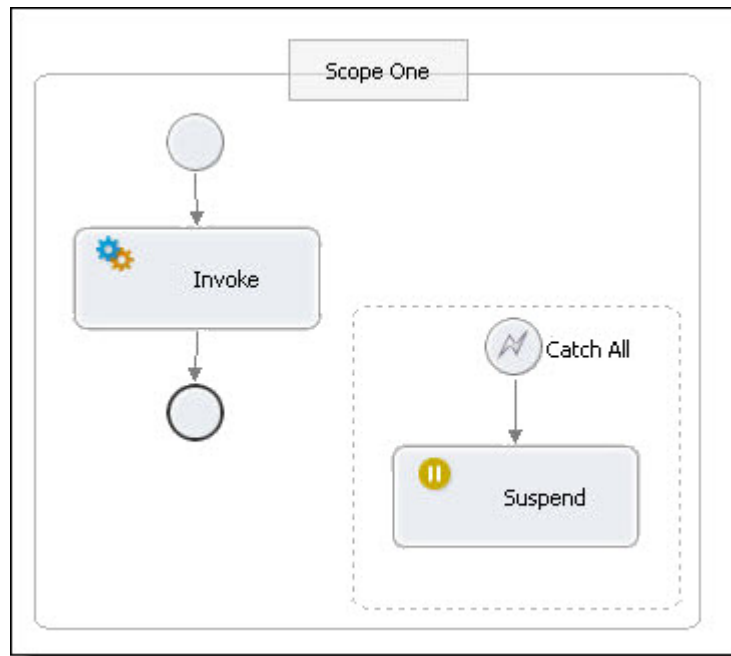
一時停止されたプロセスでは、プロセス例外管理を実行できます。プロセスコンソールを使用すると、フォールトが発生したアクティビティの場所を簡単に表示できます。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	変数
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Suspend アクティビティを作成する手順

1. **【タスクの一時停止】** を Process Editor キャンバスにドラッグします。
タスクの一時停止には背景色を追加できます。
2. フォールトハンドラの Catch または Catch All アクティビティなどの適切なコンテナにアクティビティを追加するか、別のアクティビティをそのアクティビティにリンクします。

次の図は、Suspend アクティビティの使用例を示しています。



XML 構文

```
<suspend variable="NCName"? standard-attributes>  
  standard-elements  
</suspend>
```

例

```
<catchAll>  
  <extensionActivity>  
    <ext:suspend/>  
  </extensionActivity>  
</catchAll>
```

Throw

BPMN 実装: Error Throw イベント

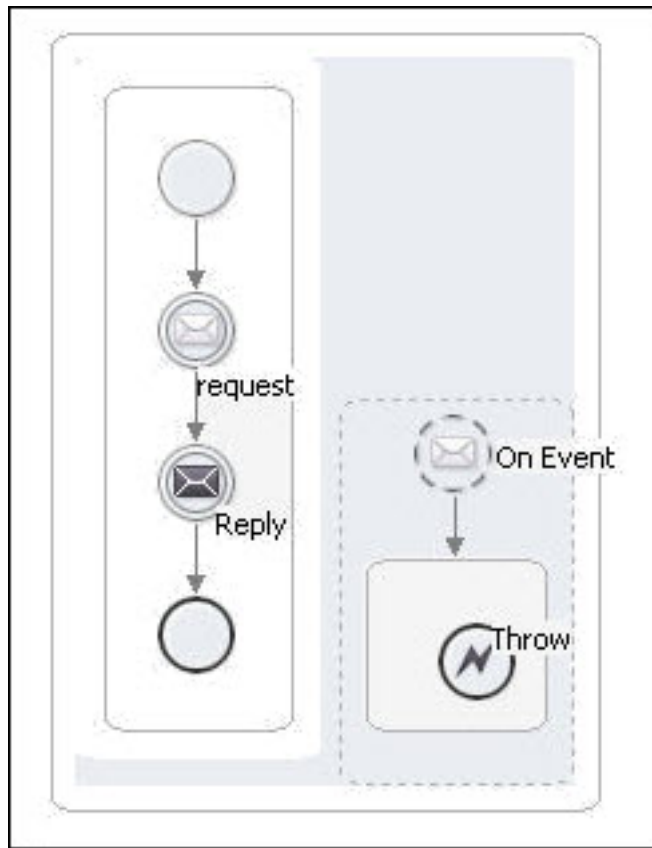
BPEL 仕様に記載されているフォールトのリストについては、このヘルプの他の場所にある「*BPEL 標準フォールト*」を参照してください。

必須のプロパティ	オプションのプロパティ
フォールト名	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	フォールト変数
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Throw アクティビティを作成する手順

1. **「Error Throw イベント」** を Process Editor キャンバスにドラッグします。
2. **「プロパティ」** ビューで、スローするフォールトを指定します。

次の図は、Throw アクティビティの使用例を示しています。



XML 構文

```
<throw faultName="QName"
      faultVariable="BPELVariableName"? standard-attributes
      standard-elements
/>
```

例:

```
<throw xmlns:FLT="http://example.com/faults"
      faultName="FLT:OutOfStock"/>
```

検証

Validate アクティビティの変数リストの変数を選択します。

Validate アクティビティは、関連する XML および WSDL データ定義に照らし合わせてプロセス変数の値を検証します。変数が BPEL プロセス内で使用されているかどうか、またはどのように使用されているかに関係なく、検証のためにこのアクティビティに変数のリストを追加できます。

シミュレーション中に変数に無効な値が含まれていることが判明した場合、プロセスは `bpel:invalidVariables` フォールトで終了します。

Assign アクティビティ内で使用される特定の変数を検証することもできることに注意してください。詳細については、「[割り当て](#)」(ページ 173)を参照してください。

また、コピー操作で [ソース要素名の維持] 属性が有効になっていると、ターゲット変数が無効になる可能性があることにも注意してください。

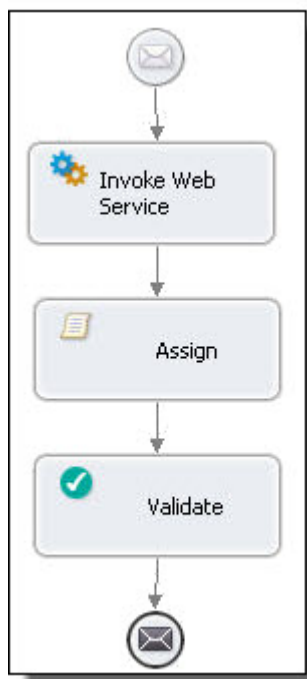
WS-BPEL 検証機能の概要については、「[変数の検証](#)」 (ページ 258) を参照してください。

必須のプロパティ	オプションのプロパティ
変数	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。

プロセスに Validate アクティビティを追加する手順

1. **【タスクの検証】** を Process Editor キャンバスにドラッグします。
Validate タスクの一時停止には背景色を追加できます。
2. 検証をダブルクリックして、**【検証の選択】** ダイアログを開きます。
または、**【プロパティ】** ビューで、**【変数】** フィールドをクリックし、**【ダイアログ (...)]** をクリックします。
3. **【変数の選択】** ダイアログで、XML または WSDL 定義に照らし合わせて検証する 1 つ以上のプロセス変数を選択します。

次の図は、Validate アクティビティを使用して、Assign アクティビティの後に変数の値を検証する簡単な例を示しています。



XML 構文

```

<validate variables="BPELVariableName-list"
  standard-attributes>
  standard-elements
</validate>

```

Wait

BPMN 実装: タイマーキャッチイベント

Wait アクティビティは、特定の期間、または特定の時間が経過するまで待機するようにビジネスプロセスに指定します。

Wait 式は式として記述し、値は dateTime、date、または duration の XML スキーマタイプとなります。

必須のプロパティ	オプションのプロパティ
Wait 式	「アクティビティラベルの選択」 (ページ 158)
待機タイプ	「入力リンクに対する結合条件の作成」 (ページ 292)
	「プロセス要素とプロパティ」 (ページ 130)
	「プロセスへのコメントの追加」 (ページ 79)
	「プロセスへのドキュメントの追加」 (ページ 79)
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)

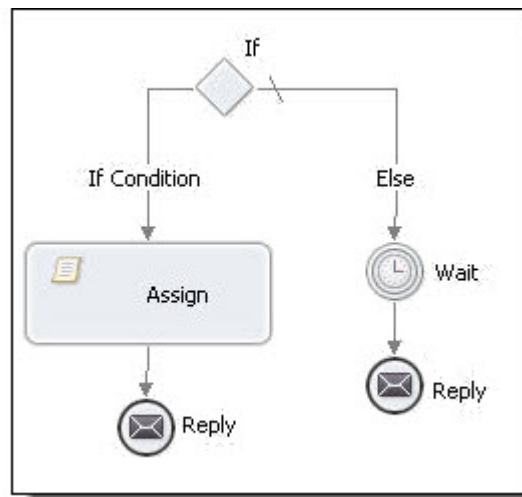
必須のプロパティ	オプションのプロパティ
	「アクティビティまたはリンクの実行状態の表示」 (ページ 419)
	「拡張要素と属性の宣言」 (ページ 126)。

Wait アクティビティを作成するには

1. **【タイマーキャッチイベント】** を Process Editor キャンバスにドラッグします。
2. [プロパティ] ビューで、[待機タイプ] に [期間] または [期限] を選択します。静的な期間または期限を選択するか、式を作成できます。
3. 期間または期限の静的コントロールを選択するか、期限または期間の式の横にある [ダイアログ (...)] ボタンをクリックします。
4. [式] ボックスで、次のいずれかを指定します。
 - 期限には、'2010-12-12T12:00'などの日付と時刻を指定します。必ず一重引用符を含めるようにしてください。
 - 期間には、'P1DT10S'などの時間を指定します。必ず一重引用符を含めるようにしてください。

以下の期限式と期間式の一部の例に加えて、XML スキーマ仕様に示されている例を示します。

次の図は、Wait アクティビティの使用例を示しています。



以下を参照してください。

- <http://www.w3.org/TR/xmlschema-2/#duration>
- <http://www.w3.org/TR/xmlschema-2/#datetime>
- <http://www.w3.org/TR/xmlschema-2/#date>

XML 構文

```

<wait (for="duration-expr" |
      until="deadline-expr")standard-attributes>
  standard-elements
</wait>

```

例:

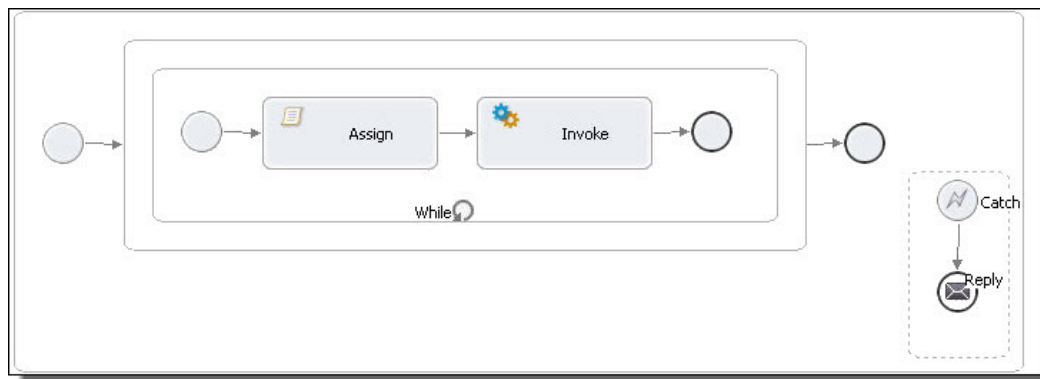
- 1 秒という期間:
`<wait for='PT1S' />`
- 1 年 2 か月 3 日 10 時間 30 分という期間:
`<wait for= "P1Y2M3DT10H30M' " />`
- 日時による期限:
`<wait until="'2010-12-12T12:00' " />`

第 12 章

ゲートウェイと制御フロー

Process Developer は、アクティビティを構造化するためのコンテナを提供します。それぞれのコンテナには、子アクティビティの実行方法に関する独自のルールがあります。

次の図は、ネストされた構造の使用方法を確認できるように、ネストされた一部の構造を使用したプロセスの例を示しています。



トピック

- [「BPMN から BPEL へのゲートウェイの実装と制御フロー」](#) (ページ 217)
- [「アクティビティを構造化するさまざまな方法」](#) (ページ 218)

BPMN から BPEL へのゲートウェイの実装と制御フロー

BPMN から BPEL へのゲートウェイの実装と制御フロー

多くの場合、BPMN 要素は、以下の表に記載された 1 つ以上の BPEL アクティビティとして実装できます。

次の表は、BPMN タイプ（ゲートウェイと制御フロー）に分類されています。

追加の実装については、[「BPMN から BPEL へのタスクとイベントの実装」](#) (ページ 153) を参照してください。

ゲートウェイ

BPMN ゲートウェイ	BPEL 実装
Exclusive (排他)	詳細については、「 ゲートウェイタイプについて 」 (ページ 220)を参照してください。
Parallel (並列)	詳細については、「 ゲートウェイタイプについて 」 (ページ 220)を参照してください。
Inclusive (包括)	詳細については、「 ゲートウェイタイプについて 」 (ページ 220)を参照してください。
Complex (複合)	詳細については、「 ゲートウェイタイプについて 」 (ページ 220)を参照してください。
Event-based (イベントベース)	詳細については、「 Pick 」 (ページ 231)を参照してください。

制御フロー

BPMN 制御フロー	BPEL 実装
Conditional Pattern (条件パターン)	「If」 (ページ 228)
「Fork Join」 (ページ 227)	リンクで結合された 2 つの空のアクティビティ (並列ゲートウェイ) (BPEL Flow アクティビティに相当)
Repeat Pattern (繰り返しパターン)	「Repeat Until」 (ページ 233)
ループ	「While」 (ページ 239)
Multi-Instance (複数インスタンス)	「For Each」 (ページ 223)
組み込みサブプロセス	「Scope」 (ページ 235)

アクティビティを構造化するさまざまな方法

BPEL 言語は、アクティビティを構造化してネストする方法を提供します。Process Developer は、構造化されたアクティビティを Process Editor のゲートウェイおよび制御フローパレットにグループ化します。基本的なアクティビティは、親コンテナで設定されたルールと条件に基づいて実行されます。

構造化コンテナの使用に加えて、エッジツールとリンクすることでアクティビティを構造化することもできます。詳細については、[第 15 章, 「リンク」](#) (ページ 264)を参照してください。

制御フロー項目の概要

次の表に、制御フローアクティビティの簡単な定義を示します。

コンテナ名	説明
「ゲートウェイ」 (ページ 219)	タイプに応じて、入力リンクと出力リンク、およびそれらのさまざまな条件を許可します。BPEL の空のアクティビティに相当します。
「Pick」 (ページ 231)	入力メッセージまたはタイムアウトイベントに基づいて、選択したアクティビティを実行します。プロセスを開始するための [インスタンスの作成] プロパティが含まれています。
「Fork Join」 (ページ 227)	上記の BPEL フローアクティビティに相当する BPMN。
「If」 (ページ 228)	ブランチ条件に基づいて、選択したアクティビティを実行します
「While」 (ページ 239)	条件が false と評価されるまで、アクティビティを繰り返し実行します
「Repeat Until」 (ページ 233)	アクティビティを少なくとも 1 度実行し、その条件が true と評価されるまで繰り返します
「Scope」 (ページ 235)	相関、補償、フォールト処理、イベント処理、終了処理、およびローカルパートナーのリンクと変数を使用して、一連のアクティビティのコンテキストを提供します
「For Each」 (ページ 223)	ドキュメント要素のリストを反復処理します
「Sequence」 (ページ 234)	(Process Developer Classic のみ)。順序付きリストでアクティビティを実行します
「Flow」 (ページ 222)	(Process Developer Classic のみ)。すべてのアクティビティを並列で実行します

選択した構造化アクティビティのグループ化解除

If または Repeat Until アクティビティがシーケンスに含まれていない場合は、これらのアクティビティのグループ化を解除できます。アクティビティをマウスで右クリックし、**【グループ解除】** を選択します。グループ解除を行うと、構造化されたコンテナとそのすべてのコンテンツが、同じ位置に表示される個々のアクティビティ、リンク、およびゲートウェイに分割されます。シーケンス（アクティビティを囲む青い境界線）の **【グループ解除】** を選択することもできます。

If または Repeat Until のグループ化を解除すると、その中に含まれていたアクティビティにリンクできるようになります。グループ化を解除しない場合は、If または Repeat Until のみにリンクできます。

ゲートウェイ

ゲートウェイ制御フローの表記は BPMN のみです。この制御フローは、BPEL の [「Empty」 \(ページ 192\)](#) アクティビティと同等です。

ゲートウェイは、プロセスの制御点を表します。これは、シーケンスフローを結合したり、それらを分割したりするという 2 つの働きを持ちます。ゲートウェイは、このいずれかの働きまたは両方の働きを同時に持ちます。結合および分割は、入力リンクと出力リンクによって実行されます。

ゲートウェイタイプについて

以下で説明しているとおり、ゲートウェイには4つのタイプがあります。タイプを選択すると、そのタイプの受信リンクと送信リンクの結合条件がゲートウェイの「プロパティ」ビューで自動的に生成されます。複合ゲートウェイの場合は、結合条件を手動で追加できます。デフォルトのゲートウェイタイプは「排他」です。

受信リンクまたは送信リンクの結合条件に基づいて、ゲートウェイは次のいずれかに分類されます。

- 包含（O 付きのひし形）



1つ以上の受信リンクが true と評価され、遷移条件によって残りの受信リンクが設定されるまでアクティビティは待機します。いずれのアウトバウンドリンクも遷移条件を持つことがあり、また、それらの条件が true と評価されることもあります。

- 排他



デフォルトのタイプ。1つ以上の受信リンクが true と評価され、遷移条件によって残りの受信リンクが設定されるまでアクティビティは待機します。アウトバウンドリンクのうちの1つが true と評価される必要があります。実行時に複数のアウトバウンドリンクが true と評価された場合は、最初のリンクが実行されます。

- 並列（+付きのひし形）



アクティビティは、すべての受信リンクが true と評価されるまで待機します。すべてのアウトバウンドリンクが true です（並列ゲートウェイからの移行条件は許可されません）。

- 複合（アスタリスク付きのひし形）



アクティビティは、すべての受信リンクが設定され（true または false）、複合ゲートウェイ内で指定された結合条件が true と評価されるまで待機します。いずれのアウトバウンドリンクも遷移条件を持つことがあり、また、それらの条件が true と評価されることもあります。

- イベントベース。[「Pick」 \(ページ 231\)](#)を参照してください。

If、While、Repeat Until などの BPEL 構造化アクティビティ、またはゲートウェイを使用して意思決定を構築するための使いやすい代替案。

排他ゲートウェイは、ループバックリンクのソースまたはターゲットとして適しています。詳細については、[「相互に排他的な遷移」](#) (ページ 221)を参照してください。

必須のプロパティ	オプションのプロパティ
ゲートウェイタイプ	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。
	「相互に排他的な遷移」 (ページ 221)

相互に排他的な遷移

「相互に排他的な遷移」はすべてのアクティビティのプロパティですが、このプロパティは直接設定しないことをお勧めします。排他ゲートウェイにより、このプロパティは自動的に [はい] に設定されます。

このプロパティを [はい] に設定すると、アウトバウンドリンクが相互に排他的であることが宣言されます。これにより、ループバックリンクを作成して、構造化されたアクティビティから抜け出すリンクを作成することができます。2 つ以上のリンク条件によってターゲットアクティビティが同時に実行されることがないかどうかを検証する方法はないため、このプロパティを設定する必要があります。詳細については、[「リンクに対する Process Developer 拡張機能」](#) (ページ 265)を参照してください。

[相互に排他的な遷移] プロパティは、[はい] に設定した排他的ゲートウェイを除くすべてのアクティビティのデフォルトとして [いいえ] に設定されています。

ゲートウェイの構築

次の手順に従って、ゲートウェイを構築します。

1. **【ゲートウェイ】** パレットから、Process Editor キャンバスにゲートウェイをドラッグします。
2. Invoke などのアクティビティをゲートウェイの下または上にドラッグして、シーケンスに自動的にリンクさせます。
3. 必要に応じて、キャンバスに別のアクティビティを追加し、2 つのアクティビティをリンクさせます。
4. リンクを使用した設計に関するヒントについては、[「遷移条件付きのリンクの追加」](#) (ページ 267)を参照してください。
5. 必要に応じて、遷移条件をリンクに追加し、対応するゲートウェイタイプを [プロパティ] ビューから選択します。

Flow

Flow コンテナは、Process Developer Classic レイアウトスタイルで表示されます。BPMN スタイルが有効な場合、パレットに Flow コンテナはありません。同等のコンテナは「[Fork Join](#)」(ページ 227)と呼ばれます。

Flow コンテナは、すべてのアクティビティを並列で実行します。これは、2 つの Receive アクティビティなど、2 つ以上のアクティビティを定義して同時に開始することができることを意味します。フローを開始すると、アクティビティが開始されます。コンテナ内のすべてのアクティビティが完了すると、フローが完了します。

例えば、売り手と買い手の Receive アクティビティをフローに追加することで売り手および買い手から承認を受けられるため、プロセスによって注文出荷サービスの呼び出しなどの別のアクティビティが実行される前に、両方の受信が完了します。

フローには、アクティビティ間の依存関係を指定して、アクティビティが実行される順序を制御するリンクを含めることもできます。詳細については、[第 15 章、「リンク」](#) (ページ 264)を参照してください。

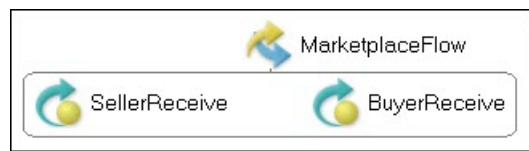
Flow は、アクティビティを同時に実行する場合に最も役立ちます。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

フローを構築する手順

- レイアウトスタイル設定が [Process Developer Classic] に設定されていることを確認します。
- [コンテナ]** パレットから、**[Flow]** を Process Editor キャンバスにドラッグします。
- Receive などのアクティビティを Flow 内にドラッグします。
- 他のアクティビティを Flow 内にさらにドラッグして、同時に実行されるようにします。
- Flow 内の各アクティビティのすべてのプロパティを指定します。

次の図は、Flow アクティビティの例を示しています。



XML 構文

```
<flow atandard-attributes>
  standard-elements
  <links>?
    <link name="ncname">+
  </links>
  activity+
</flow>
```

例:

```
<flow name="MarketplaceFlow">
  <receive name="SellerReceive" partnerLink="seller"
    portType="tns:sellerPT" operation="submit"
    variable="sellerInfo" createInstance="yes">
  </receive>
  <receive name="BuyerReceive" partnerLink="buyer"
    portType="tns:buyerPT" operation="submit"
    variable="buyerInfo" createInstance="yes">
  </receive>
</flow>
```

For Each

BPMN 実装: マルチインスタンス制御フロー

For Each アクティビティには Scope アクティビティが含まれ、指定したカウントで実行されます。実行の反復は、並列で実行するか、順番に実行することができます。実行する反復回数は、式の開始値と最終値の評価によって決定されます。これらの値は包含的であるため、開始値が 1 で終了値が 10 の場合、エンクローズされたスコープは 10 回実行されます。

開始値と最終値の式は、xs:unsignedint と評価される必要があります。このように評価されない場合は、bpel:forEachCounterError のランタイム例外が発生します。最終値が開始値よりも大きい場合、For Each は実行されません。

For Each のカウンタ名は、子スコープ内の暗黙的な変数として表されます。変数の初期値は、開始値から最終値までの範囲にある For Each の反復の現在の値に設定されます。この変数は、For Each のスコープ内にネストされたすべてのアクティビティからアクセス可能で、子スコープ内の変数として表示されます。

M 個のブランチのうちの N 個を処理するための完了条件の使用

For Each アクティビティには、オプションの *[完了条件式]* プロパティを含めることができます。また、このプロパティには、オプションの *[成功したブランチのみをカウント]* プロパティを含めることができます。

完了条件式を使用することで、M 個のブランチのうちの N 個以上という処理を実行できます。式は、M 個のうちの N 個という条件を定義するために使用される xs:unsignedint 値と評価されます。これは、For Each が実行を開始するときに 1 度評価されます。completionCondition が xs:unsignedint でない場合、または条件を満たすことがないと実行前に判断された場合、For Each は bpel:invalidBranchCondition で失敗します。この例は、それぞれの反復が 3 回しかない場合の completionCondition 値 5 です。

プロパティ *[成功したブランチのみをカウント]* は [はい] または [いいえ] に設定できます。[はい] に設定した場合、Process Developer エンジン、正常に完了した完了条件に向けてブランチをカウントすることを意味します（つまり、各反復の子スコープはフォールトをキャッチしておらず、補償の対象となります）。デフォルト値である [いいえ] は、For Each で、完了条件に向けて子の反復の完了をカウントします。

各反復の完了時に、For Each は、完了したスコープの数とその完了条件をテストします（上記の *[成功したブランチのみをカウント]* 設定が考慮されます）。完了条件を満たすための十分な反復が残っていない場合、For Each は bpel:completionConditionFailure で失敗し、残りの反復は実行されません。または、それぞれの反復が並列の場合は終了します。

完了条件は、作業範囲を完了するために必要な最小限の項目のみを処理する場合に使用するのが一般的です。例えば、BPEL プロセスでは、さまざまなパートナーに並列で呼び出しを行い、項目の承認を求めることができます。このプロセスでは、3つの承認のみで項目の処理が続行されるようになる場合があります。プロセスで多くの承認が要求されたとしても、3つの承認しか受け取られないような場合、作業の承認範囲を超えて実行されることがあります。その後、さらに後続の処理を続行します。

並列 For Each

並列 For Each、またはシーケンシャル For Each を作成できます。並列 ForEach では、エンクローズされたアクティビティのインスタンスが同時に発生します。それぞれのカウンタ変数は、カウント内の整数値の1つに初期化されます。

ドキュメントパーツの同時処理には、並列 For Each を使用します。例えば、発注書の明細など、複合的なメッセージ変数の一部を処理できます。

Receive/Reply または onMessage/Reply の正しいペアをまとまりとして扱うデフォルトの MessageExchange プロパティは、スコープ内で自動的にかつ暗黙的に宣言されます。詳細については、[「メッセージ交換の宣言」 \(ページ 136\)](#)を参照してください。

シーケンシャル For Each

シーケンシャル For Each では、エンクローズされたアクティビティのインスタンスが一度に1つずつ実行されます。

必須のプロパティ	オプションのプロパティ
カウンタ名	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
開始カウンタ値	完了条件式
最終カウンタ値	成功したブランチのみをカウント
並列実行フラグ	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76) を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

For Each を構築するには

1. **【制御フロー】** パレットから、**【マルチインスタンス】** アクティビティを Process Editor キャンバスにドラッグします。

2. For Each の [プロパティ] ビューで、必要なプロパティを入力します。
 - 並列実行フラグ。この設定を有効にすると、アクティビティのすべてのインスタンスが同時に実行されます。このタイプの実行に関する要件については、上記の「並列 For Each」の説明を必ずお読みください。この設定を無効にすると、スコープのアクティビティが順番に実行されます。
 - エンクローズされたスコープのローカル変数を識別するカウンタ名を追加します。デフォルト名は counter です。
 - 開始カウンタ値。値に符号なし整数を追加します。値を入力するか、行の最後にある [ダイアログ (...) ボタン] を選択して、式ビルダを開きます。詳細については、[「式ビルダの使用」 \(ページ 275\)](#)を参照してください。
 - 最終カウンタ値。上記の開始カウンタ値を参照してください。
 - 必要に応じて、オプションのプロパティ **【完了条件】** の値を追加し、上記のように [成功したブランチをカウント] を有効にして、成功したブランチのみをカウントできるようにします。
3. Scope コンテナのプロパティを入力します。
4. Scope コンテナ内の各アクティビティのすべてのプロパティを指定します。並列実行が有効になっている場合は、スコープ内で Receive/Reply および onMessage/Reply と照合するためのメッセージ交換プロパティを必ず設定してください。

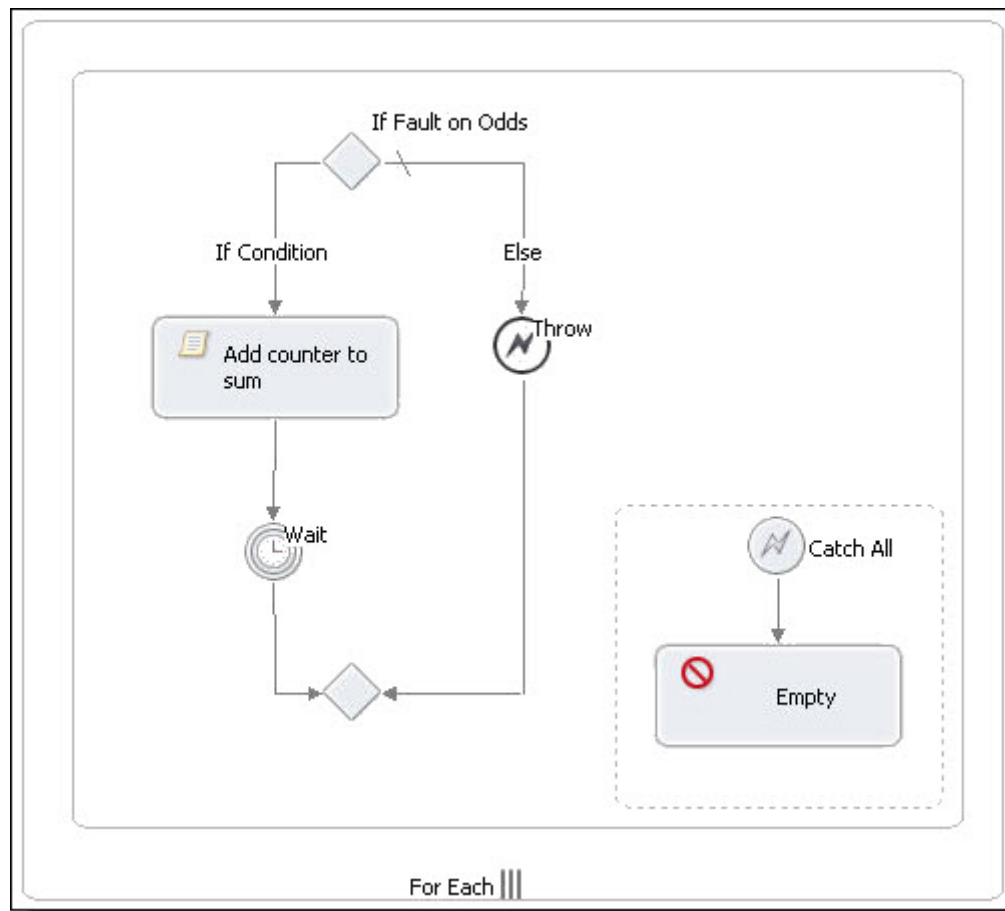
XML 構文

```
<forEach counterName="NCName" parallel="yes|no">
  standard-attributes>
  standard-elements
    <startCounterValue expressionLanguage="anyURI">
    </startCounterValue>
    <finalCounterValue expressionLanguage="anyURI">
    </finalCounterValue>
    <completionCondition extension-attribute
      extension-element
        <branches expressionLanguage="URI"?
          countSuccessfulBranchesOnly="yes|no"?>
          an-integer-expression
        </branches>
      </completionCondition>
  activity
</forEach>
```

例 1: 100 回の実行、10 回成功した場合の実行

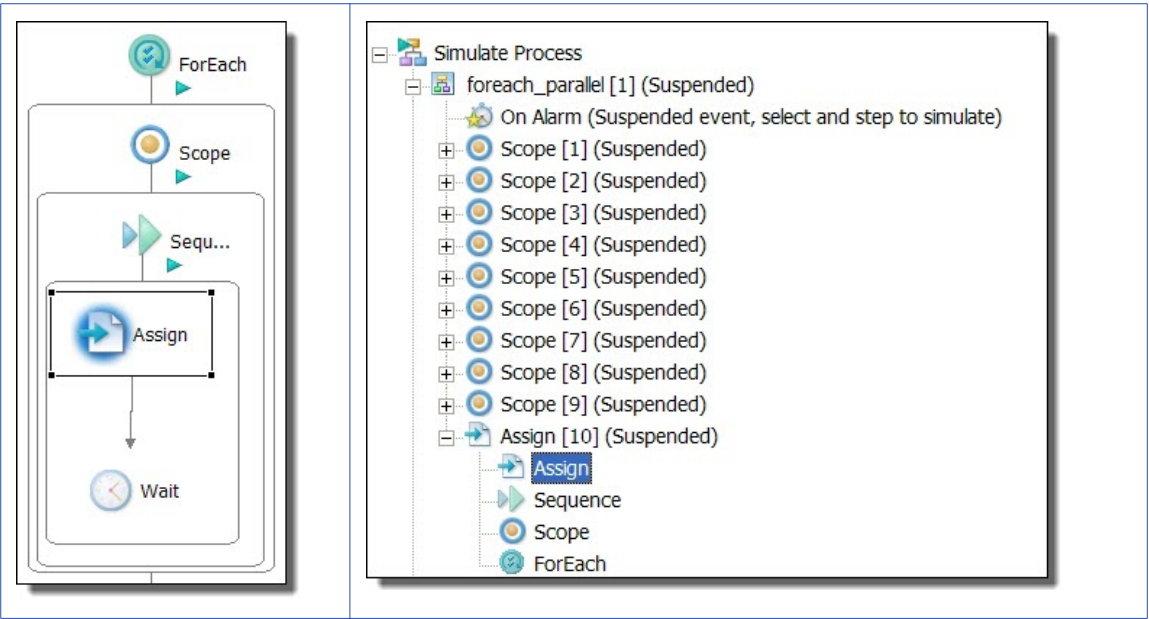
```
<forEach counterName="counter" parallel="no">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>100</finalCounterValue>
  <completionCondition>
    <branches countSuccessfulBranchesOnly="yes">
      10</branches>
    </completionCondition>
  <scope>...</scope>
</forEach>
```

例 2: 完了条件を持つそれぞれの連続



例 3: 各アクティビティの並列の実行スレッド

次の図は、カウンタ値が 10 の並列 For Each の実行スレッドを示しています。エンクローズされたスコープの 10 個のインスタンスが並行して実行されていることに注意してください。



For Each アクティビティに関連するフォールト

For Each アクティビティでは、次のフォールトが発生する可能性があります。

- 実行中のプロセスでカウンタ変数のエラーが発生すると、bpel:forEachCounterError フォールトがスローされます。
- For Each アクティビティでは、完了条件が true になることがない場合、bpel:completionConditionFailure がスローされます。
- [成功したブランチのみをカウント] 式で評価された整数値がカウンタよりも大きい場合、bpel:invalidBranchCondition フォールトがスローされます。

Fork Join

Fork Join コンテナの表記は BPMN のみにあります。この制御フローは、BPEL の [「Flow」 \(ページ 222\)](#) アクティビティと同等です。

Fork Join をキャンバスにドラッグすると、デフォルトで 2 つのパスが含まれます。必要に応じてパスを追加できます。それぞれのパスには、1 つ以上のアクティビティが含まれています。各パスは並列で実行されます。

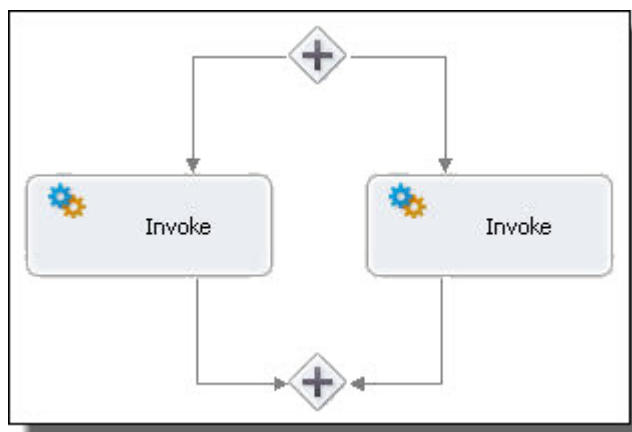
必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。

必須のプロパティ	オプションのプロパティ
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Fork Join を構築する手順

1. **【制御フロー】** パレットから、**【Fork Join】** を Process Editor キャンバスにドラッグします。
2. Receive タスクなどのアクティビティを 1 つのパスにドラッグします。
3. その他のアクティビティを、同じパスまたは他のパスにさらにドラッグします。
4. Fork Join を右クリックし、**【パスの追加】** を選択して、新しいパスを追加します。

次の図は、並列で実行される 2 つの呼び出しを含む Fork Join アクティビティの例を示しています。



If

BPMN 実装: 条件パターン

If コンテナは、If 要素とオプションの Else If 要素で定義された 1 つ以上の条件に基づいてアクティビティを実行し、その後にはオプションの Else 要素が続きます。条件は順番に評価され、最初に true と評価された条件でアクティビティが実行されます。すべての条件が true または false の条件と評価されるような場合は、If コンテナが適しています。

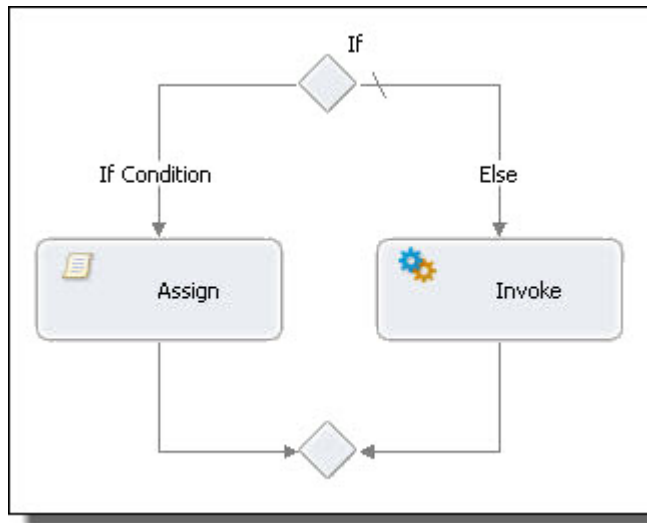
ブランチを定義しない場合、暗黙の Else は空のアクティビティを実行します。

[「選択した構造化アクティビティのグループ化解除」](#) (ページ 219)も参照してください。

必須のプロパティ	オプションのプロパティ
If 条件	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。
	拡張属性と要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

If を構築する手順

1. **【制御フロー】** パレットから、**【条件パターン】** アクティビティを Process Editor キャンバスにドラッグします。
 2. **【If 条件】** コンテナをダブルクリックして、式ビルダを開きます。
または、**【プロパティ】** ビューで、**【If 条件】** の横にある **【ダイアログ (...) ボタン】** をクリックします。
 3. 変数、関数、および演算子を選択して、ブール式を作成します。詳細については、[「式ビルダの使用」](#) (ページ 275)を参照してください。
 4. **【アクティビティ】** パレットから、アクティビティを **【If 条件】** コンテナにドラッグし、アクティビティのプロパティを選択します。
 5. 必要に応じて、**【Else If】** または **【Else】** ブランチに対して手順 2 から 5 を繰り返します。
- 次の図は、If アクティビティの例を示しています。



XML 構文

```

<if standard-attributes>
  standard-elements
  <condition expressionlanguage="anyURI"?>
    bool-expr
  </condition>
  activity
  <elseif*>
    <condition expressionlanguage="anyURI"?>
      bool-expr
    </condition>
    activity
  </elseif>
  <else?>
    activity
  </else>
</if>

```

例:

```

<if xmlns:inventory="http://supply-chain.org/inventory"
  xmlns:FLT="http://example.com/faults">
  <condition>
    bpel:getVariableProperty('stockResult',
      'inventory:level') > 100
  </condition>
  <flow>
    <!-- perform fulfillment work -->
  </flow>
  <elseif>
    <condition>
      <bpel:getVariableProperty('stockResult',
        'inventory:level') >= 0
    </condition>
    <throw faultName="FLT:OutOfStock"
      variable="RestockEstimate" />
  </elseif>
  <else>
    <throw faultName="FLT:ItemDiscontinued" />
  </else>
</if>

```

Pick

BPMN 実装: イベントベースのゲートウェイ

Pick アクティビティには、1 つ以上のメッセージまたはメッセージパートがあり、必要に応じて 1 つ以上のアラームを含めることができます。Pick を実行すると、メッセージの 1 つが受信されるか、アラームの 1 つが鳴るまでアラームがブロックされます。メッセージの受信またはアラームの発生により他のすべてのブランチがすぐに停止されるため、Pick で実行されるメッセージまたはアラームは 1 つだけです。

例えば、サービスで、承認メッセージ、拒否メッセージ、およびタイムアウトを指定した Pick を使用して、見積みりへの応答を待機するプロセスを実装するといったことができます。タイムアウト期間内に受け入れメッセージまたは拒否メッセージが受信されない場合、Alarm アクティビティが実行されます。

Pick アクティビティは、非同期アクティビティに適しています。

また、Pick アクティビティを使用して、一連のメッセージを持つ、アラームのないプロセスを開始することもできます。この特殊な形式の Pick は、[インスタンスの作成] プロパティを [はい] に設定した場合に、最初のメッセージが到着した際に実行されます。

必須のプロパティ	オプションのプロパティ
onMessage の場合: パートナーリンク 操作 変数または開始パーツ 「開始パートから変数」 (ページ 197) を参照してください。	名前。 「アクティビティラベルの選択」 (ページ 158) を参照してください。 インスタンスの作成 相関。 第 24 章, 「相関」 (ページ 367) を参照してください。 結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292) を参照してください。 結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130) を参照してください。 メッセージ交換。 「メッセージ交換の宣言」 (ページ 136) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
onAlarm の場合: アラーム式 アラームのタイプ	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。

Pick の onAlarm ブランチを構築する手順

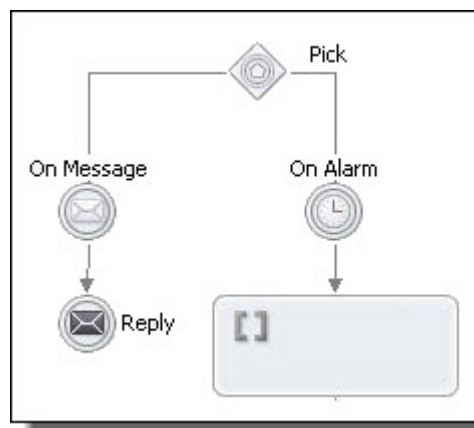
1. [ゲートウェイ] パレットから、[イベントベースのゲートウェイ] アクティビティを Process Editor キャンバスにドラッグします。
2. onAlarm ブランチを選択します。
3. [プロパティ] ビューで、アラームタイプ（期間または期限）を選択します。
4. アラーム式の横にある [ダイアログ (...) ボタン] をクリックします。
5. [式] ボックスで、次のいずれかを指定します。
 - a. 期限には、「2010-12-12T12:00」などの xsd:datetime 式を入力します

- b. 期間には、「PT1S」などの xsd:duration 式を入力します
 - c. 詳細および例については、「[期限と期間の式](#)」(ページ 293)を参照してください。
6. Start、End、または None などのアクティビティを [アラーム時] アイコンにドラッグし、必要に応じてアクティビティのプロパティを選択して、アラームが鳴った際に実行するアクションを指定します。

Pick の onMessage ブランチを構築する手順

1. **【ゲートウェイ】** パレットから、**【イベントベースのゲートウェイ】** アクティビティを Process Editor キャンバスにドラッグします。
2. OnMessage ブランチを選択します。
3. OnMessage の [プロパティ] ビューで、必要に応じて新しいパーティシパントを作成するか、リストからパーティシパントを選択します。
4. [スクリプトタスク] (割り当て) などのアクティビティを [On Message] ブランチにドラッグし、アクティビティのプロパティを選択して、メッセージが到着した際に実行するアクションを指定します。

次の図は、Pick アクティビティの例を示しています。



XML 構文

```
<pick createInstance="yes|no"? standard-attributes>
  standard-elements
  <onMessage partnerLink="NCName"
    portType="QName"? operation="NCName"
    variable="BPELVariableName"?>+
    messageExchange="NCName"?>+
    <correlations?
      <correlation set="NCName" initiate="yes|join|no"?>+
    </correlations>
    <fromParts?
      <fromPart part="NCName" toVariable="BPELVariableName" />+
    </fromParts>
    activity
  </onMessage>
  <onAlarm>
    (
      <for expressionLanguage="anyURI"?>duration-expr</for>
      <until expressionLanguage="anyURI"?>deadline-expr
    </until>
    )
    activity
  </onAlarm>
</pick>
```

例:

```
<pick name="pick1" createInstance="yes">
  <onAlarm> <for>'PT30S'</for>
  </empty>
</onAlarm>
  <onMessage partnerLink="customer"
    portType="lns:loanServicePT" operation="request"
    variable="request" >
    <reply activity>
  </onMessage>
</pick>
```

Repeat Until

BPMN の実装: Repeat Pattern

Repeat Until アクティビティは、条件が true と評価されるまでアクティビティを繰り返し実行します。While アクティビティとは対照的に、Repeat Until ループは、含まれているアクティビティを少なくとも 1 度実行します。

[「選択した構造化アクティビティのグループ化解除」](#) (ページ 219)も参照してください。

必須のプロパティ	オプションのプロパティ
条件	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Repeat Until を構築する手順

1. **【制御フロー】** パレットから、**【Repeat Pattern】** アクティビティを Process Editor キャンバスにドラッグします。
2. [Repeat Until] をダブルクリックして条件ビルダを開き、ブール式を作成します。詳細については、[「式ビルダの使用」](#) (ページ 275)を参照してください。
3. [Scope] や [Invoke] などのアクティビティを [Repeat Until] 内にドラッグします。
4. [Repeat Until] 内でそれぞれのアクティビティのすべてのプロパティを指定します。

XML 構文

```
<repeatUntil standard-attributes>
  standard-elements
  activity
</repeatUntil>
```

```

<condition expressionLanguage="anyURI"?>
  bool-expr
</condition>
</repeatUntil>

例:
<repeatUntil>
  <condition
    expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:
      xpath1.0">$counter > 0
  </condition>
  <sequence>
    <assign name="IncrementCounter">
      <copy>
        <from>$counter + 1</from>
        <to variable="counter"/>
      </copy>
    </assign>
    <wait name="WaitTwoSeconds">
      <for expressionLanguage=
        "urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
        'PT2S'
      </for>
    </wait>
  </sequence>
</repeatUntil>

```

Sequence

Sequence コンテナは、Process Developer Classic レイアウトスタイルで表示されます。BPMN スタイルが有効な場合、パレットに Sequence コンテナは表示されません。キャンバスでの設計を簡素化するために、すべてのアクティビティは非表示の Sequence に含まれています。

Sequence コンテナは、アクティビティを整列し、順序付きリストでアクティビティを実行しますこれは、Sequence の最初のアクティビティが実行され、それが終了すると 2 番目のアクティビティが開始されることを意味します。

必須のプロパティ	オプションのプロパティ
なし	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

Sequence を構築する手順

レイアウトスタイル設定が [Process Developer Classic] に設定されていることを確認します。

1. **【コンテナ】** パレットから、**【Sequence】** アクティビティを Process Editor キャンバスにドラッグします。
2. 他のアクティビティを Sequence にドラッグして、上から下または左から右に並べます。
ヒント: Process Editor キャンバスでアクティビティを選択し、右クリックして **【コンテナの作成】** > **【Sequence】** を選択します。
3. Sequence 内の各アクティビティのプロパティを設定します。

XML 構文

```
<sequence standard-attributes>  
  standard-elements  
  activity+  
</sequence>
```

例:

```
<sequence name="SequenceExample">  
  <receive name="receive1" partnerLink="customer"  
    portType="lns:loanServicePT" operation="request"  
    variable="request" createInstance="yes"/>  
  <reply name="reply" partnerLink="customer"  
    portType="lns:loanServicePT" operation="request"  
    variable="approval"/>  
</sequence>
```

Scope

BPMN 実装: 組み込みサブプロセス

Scope アクティビティは、アクティビティのサブセットに対するコンテキストを提供します。Scope アクティビティには、その中にネストされたアクティビティの Fault、Event、および Compensation の処理や、定義した変数のセットおよび相関セットを含めることができます。

スコープは論理的な作業単位を包含するため、作業の実行の管理が容易になり、また必要に応じてアクティビティを元に戻すことができます。例えば、顧客が有料の旅行予約をキャンセルした場合、返金を行って、他の予約に影響を与えずに予約をキャンセルする必要があるとします。アクティビティをスコープでエンクローズすることにより、アクティビティを 1 つの単位として管理するための構造と条件を作成することができます。

各スコープには、複合的な構造化アクティビティである可能性を持ったプライマリアクティビティがあります。

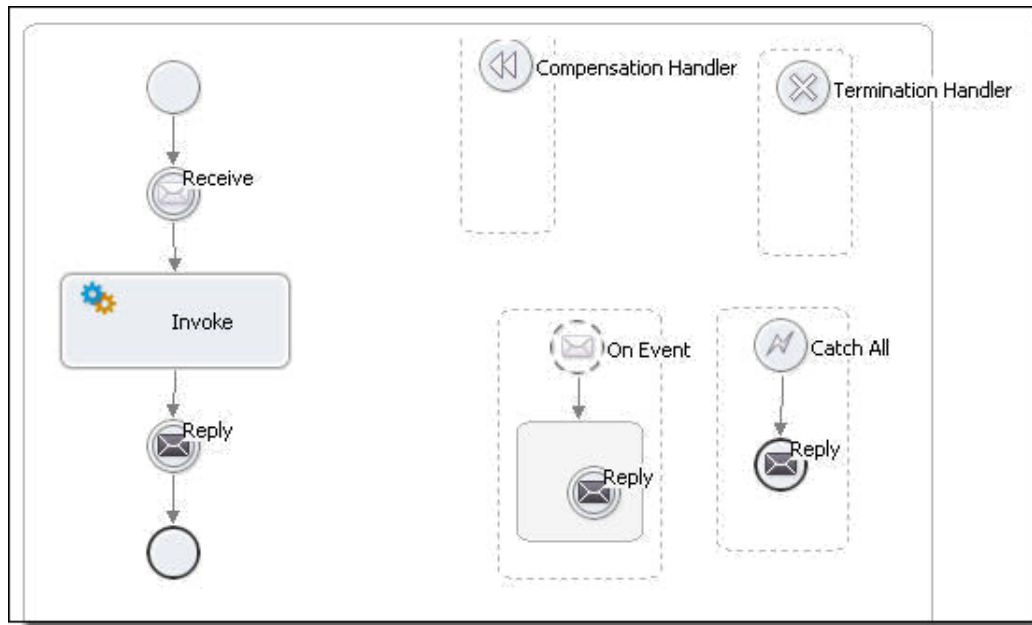
必須のプロパティ	オプションのプロパティ
分離。「 スコープ内で分離を「はい」にする設定 」 (ページ 238)を参照してください。	名前。「 アクティビティラベルの選択 」 (ページ 158)を参照してください。
	結合条件。「 入力リンクに対する結合条件の作成 」 (ページ 292)を参照してください。
	結合の失敗の非表示。「 プロセス要素とプロパティ 」 (ページ 130)を参照してください。
	コメント。「 プロセスへのコメントの追加 」 (ページ 79)を参照してください。

必須のプロパティ	オプションのプロパティ
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

スコープを作成する手順

1. **【制御フロー】** パレットから、**【埋め込みサブプロセス】** アクティビティを Process Editor キャンバスにドラッグします。
埋め込みサブプロセスが折りたたまれている場合は、このサブプロセスに背景色を追加できます。埋め込みサブプロセスを右クリックして、**【コンテナの折りたたみ】** を選択します。色のプロパティが **【プロパティ】** ビューに表示されます。
2. **【プロパティ】** ビューで、**【分離】** を **【はい】** または **【いいえ】** に設定します。
3. Process Editor キャンバスで、**【Scope】** アクティビティを右クリックして、次のような宣言を追加します。
 - 相関セット。詳細については、[第 24 章, 「相関」](#) (ページ 367)を参照してください。
 - 変数。詳細については、[第 13 章, 「変数」](#) (ページ 241)を参照してください。
 - パーティシパント。詳細については、[「【パーティシパント】ビューの使用」](#) (ページ 139)を参照してください。
4. **【ハンドラ】** パレットから、イベントフォールトハンドラと補償ハンドラを追加します。
 - 補償ハンドラ。詳細については、[「Compensate」](#) (ページ 187)および [第 27 章, 「補償」](#) (ページ 400)を参照してください。
 - イベントハンドラ。詳細については、[第 23 章, 「イベント処理」](#) (ページ 355)を参照してください。
 - 終了ハンドラ。詳細については、[「スコープへの終了ハンドラの使用」](#) (ページ 238)を参照してください。
5. **【タスク】** パレットから、アクティビティを Scope コンテナにドラッグし、アクティビティのプロパティを選択します。

次の図は、すべてのハンドラが表示された Scope アクティビティのサンプルを示しています。



アクティビティまたはアクティビティのグループを [カスタム] パレットに保存して再利用できます。スコープは複雑になる可能性があるため、再利用に適しています。詳細については、[「カスタムアクティビティの作成」 \(ページ 164\)](#)を参照してください。

XML 構文

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
  standard-attributes
  standard-elements
  <variables>?
  ...
</variables>
  <partnerLinks>?
  ...
</partnerLinks>
  <correlationSets>?
  ...
</correlationSets>
  <messageExchanges>?
  ...
</messageExchanges>
  <faultHandlers>?
  ...
</faultHandlers>
  <compensationHandler>?
  ...
</compensationHandler>
  <eventHandlers>?
  ...
</eventHandlers>
  <terminationHandler>?
  ...
</terminationHandler>
  activity
</scope>
```

例:

```
<scope isolated="no">
  <faultHandler>
    <reply>...</reply>
  </faultHandler>
  <flow>
```

```

        <invoke partnerLink="Seller" portType="Sell:Purchasing"
            operation="SyncPurchase"
            inputVariable="sendPO"
            outputVariable="getResponse"/>
        <invoke partnerLink="Shipper" portType="Ship:Orders"
            operation="OrderShipment"
            inputVariable="sendShipOrder"
            outputVariable="shipAck"/>
    </flow>
</scope>

```

関連事項:

- [「スコープ内で分離を「はい」にする設定」 \(ページ 238\)](#)
- [「スコープのライフサイクル」 \(ページ 239\)](#)

スコープ内で分離を「はい」にする設定

2つのスコープを同時に実行できます。例えば、年末に、ある銀行が利息の追加とクレジットカードの費用の請求という2つのアクティビティを口座ごとに実行するとします。これらのアクティビティは、任意の順序で実行することができます。ただし、どちらも口座情報を更新するため、同時に実行することはできません。

各スコープ内で参照される変数が競合しないようにするため、分離したプロパティを有効にすることができます。このプロパティによって変数へのシリアル化されたアクセスが有効化されるため、1つのスコープのみが共用変数にアクセスでき、同時に実行されたスコープでそれらの変数が使用される前に変数の使用が終了するようになります。

分離は、補償中には有効化されません。詳細については、[第 27 章, 「補償」 \(ページ 400\)](#)を参照してください。

スコープへの終了ハンドラの使用

BPMN 実装: キャンセルキャッチイベント

プロセス内のアクティビティは、フォールト処理、並列 `forEach` の完了条件による早期終了、または Process Developer 拡張機能の Break アクティビティの実行を通じて終了されます。スコープは、アクティビティの終了時に何が発生するかを制御するための手段を提供する唯一のアクティビティです。スコープに対する終了ハンドラは、スコープがそのメインアクティビティまたはイベントハンドラを実行している場合に有効になります。スコープが実行を完了した場合、またはスコープのフォールトハンドラが実行された場合、終了ハンドラは直ちに無効になります。スコープが終了の対象である場合、スコープの終了は、そのプライマリアクティビティと実行中のすべてのイベントハンドラインスタンスの終了から始まります。これに続いて、スコープに対するカスタム `<terminationHandler>` が実行されます。スコープに対する終了ハンドラがない場合は、デフォルトの終了ハンドラが実行されます。

デフォルトの終了ハンドラは次のとおりです。

```

<terminationHandler>
    <compensate/>
</terminationHandler>

```

スコープのプロパティに関する主な説明については、[「Scope」 \(ページ 235\)](#)を参照してください。

スコープのライフサイクル

Scope アクティビティを実行すると、子アクティビティの実行前に、エンジンにイベントメッセージまたはアラームがすぐに登録されます。アラームが登録されると、これらのメッセージとアラームは、スコープの子アクティビティが完了するまで実行可能な状態となります。

- 子アクティビティまたはスコープのイベントの 1 つの実行中にフォールトが発生した場合、スコープのフォールトハンドラはそのフォールトをキャッチしようとします。
- スコープでフォールトを検出できない場合、そのフォールトは再スローされます。スコープにフォールトが発生したと見なされ、補償の対象にはなりません。
- 子アクティビティが正常に完了すると、イベントメッセージとタイマーをアンインストールした後にスコープは完了します。また、実行中である場合は完了させることができますようになります。

スコープは、フォールトが検出または再スローされずに完了した場合、正常に完了したと見なされます。スコープが正常に完了すると、補償の対象になります。

While

BPMN 実装: Loop

While アクティビティは、条件が false と評価されるまで、アクティビティを繰り返し実行します。

Repeat Until アクティビティとは対照的に、While ループでは、含まれるアクティビティがまったく実行されない場合もあります。

必須のプロパティ	オプションのプロパティ
条件	名前。 「アクティビティラベルの選択」 (ページ 158)を参照してください。
	結合条件。 「入力リンクに対する結合条件の作成」 (ページ 292)を参照してください。
	結合の失敗の非表示。 「プロセス要素とプロパティ」 (ページ 130)を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

While を構築する手順

1. **【制御フロー】** パレットから、**【Loop】** アクティビティを Process Editor キャンバスにドラッグします。
2. **【While】** をダブルクリックして、条件ビルダを開きます。詳細については、[「式ビルダの使用」](#) (ページ 275)を参照してください。
3. **【Scope】** や **【Invoke】** などのアクティビティを **【While】** 内にドラッグします。
4. **【While】** 内の各アクティビティのすべてのプロパティを指定します。

XML 構文

```
<while standard-attributes>
  <condition expressionLanguage="anyURI"?>
    bool-expr
  </condition>
  standard-elements
  activity
</while>
```

例:

```
<while>
  <condition>
    $orderDetails > 100
  </condition>
  <scope>
    ...
  </scope>
</while>
```

第 13 章

変数

Process Developer には、BPEL プロセス内で変数を定義および使用し、サンプルデータを処理するための複数のツールとショートカットが用意されています。

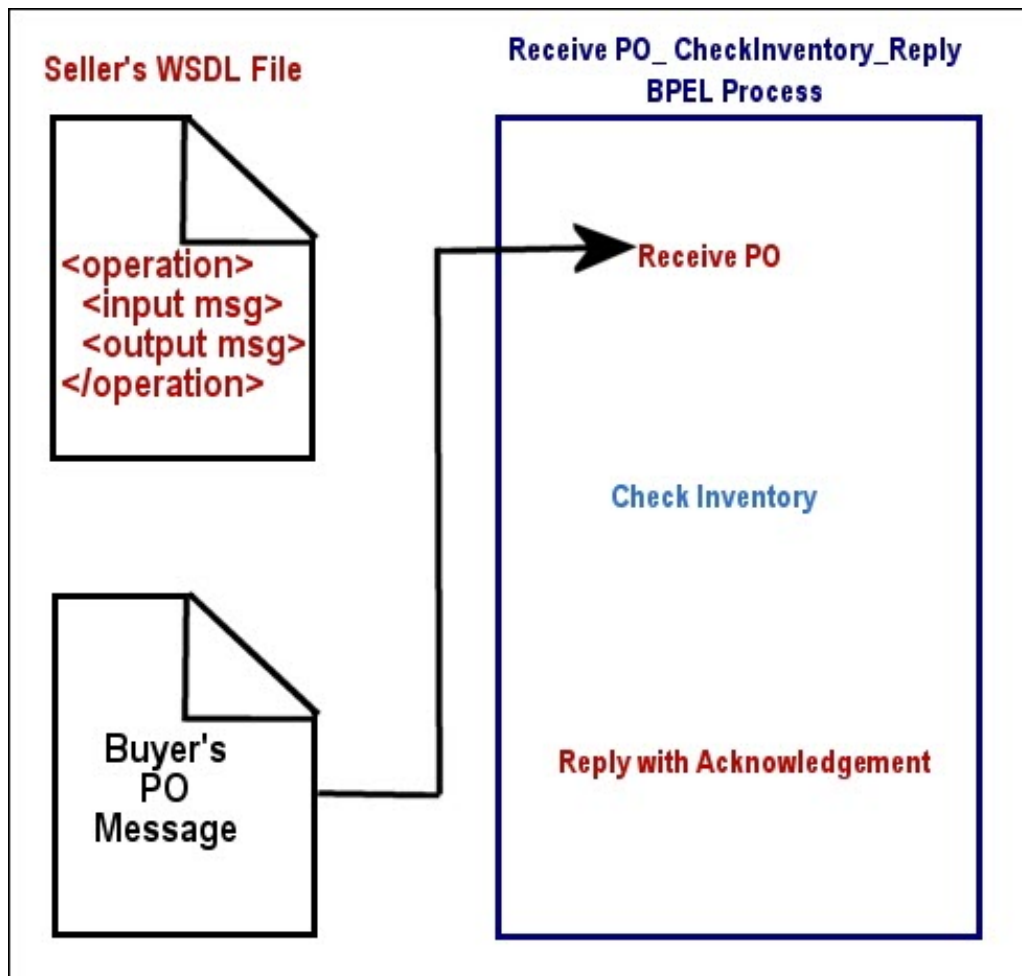
変数については、以下を参照してください。

- [「変数の概要」 \(ページ 241\)](#)
- [「変数の追加」 \(ページ 243\)](#)
- [「変数プロパティとプロパティエイリアスの追加」 \(ページ 245\)](#)
- [「変数の初期化」 \(ページ 246\)](#)
- [「変数の表示」 \(ページ 246\)](#)
- [「変数の削除」 \(ページ 249\)](#)
- [「\[プロセス変数\] ビューでのサンプルデータの使用」 \(ページ 249\)](#)
- [「XML データウィザードの使用」 \(ページ 251\)](#)
- [「変数の使用箇所の検索」 \(ページ 255\)](#)
- [「コピー操作での変数の使用」 \(ページ 256\)](#)
- [「Web サービスインタラクションアクティビティでの WSDL メッセージパートのマッピング」 \(ページ 257\)](#)
- [「変数の検証」 \(ページ 258\)](#)
- [「変数の添付ファイルの使用」 \(ページ 258\)](#)

変数の概要

BPEL プロセスは、XML 変数を介してデータを受信、操作、および送信します。変数は、ビジネスパートナー間で交換するメッセージやプロセス内で使用するデータを保持します。

例えば、バイヤーから注文書メッセージを受信するようなプロセスでは、メッセージに入力変数が含まれます。これにより、他の操作で変数の内容にアクセスできるようになります。次の図は、例を示しています。



変数は、次のいずれかの方法で定義します。

- [「WSDL メッセージタイプ」](#) (ページ 243)
- [「XML スキーマタイプ」](#) (ページ 244)
- [「XML スキーマ要素」](#) (ページ 244)

変数の使用に関するヒント:

- 変数を宣言する際に変数を初期化できます。詳細については、[「変数の初期化」](#) (ページ 246)を参照してください。
- Assign アクティビティに Validate アクティビティまたは検証属性を追加することで、変数の値を定義に照らし合わせて検証できます。詳細については、[「検証」](#) (ページ 212)および [「割り当て」](#) (ページ 173)を参照してください。
- [第 14 章, 「添付ファイル」](#) (ページ 259)に記載されているように、変数には添付ファイルを含めることができ、この添付ファイルを操作できます。

XML 構文

```
<variable name="BPELVariableName"
  messageType="QName"? type=QName? element=QName?>+ from-spec?
</variable>
```

変数名には「.」という文字を含めることはできません。この文字は、WS-BPEL の XPath 1.0 へのデフォルトのバインディングで区切り文字として使用するために予約されています。

変数の追加

タイプに基づいて新しいプロセス変数を定義します。

WSDL から Receive、Pick、Invoke、イベントハンドラ、または Reply アクティビティを作成する際に、メッセージタイプ変数をプロセスに自動的に追加できます。詳細については、[「WSDL インタフェースから開始するアクティビティの作成」](#) (ページ 160)を参照してください。

変数をプロセスに手動で追加することもできます。変数を手動で追加する前に、適切な WSDL 名前空間が定義されていることを確認してください。詳細については、[「WSDL、スキーマ、およびその他のリソースのインポート」](#) (ページ 122)を参照してください。

プロセス全体からアクセス可能なグローバル変数の宣言、およびスコープ内で使用するローカル変数の宣言ができます。ローカル変数にはグローバル変数と同じ名前を持たせることが可能で、名前が同一である場合、スコープ内ではローカル変数にのみアクセスできます。

グローバル変数を定義する手順

1. [アウトライン] ビューで [変数] を右クリックし、[追加] > [宣言] > [変数] を選択します。
2. 新しい変数をダブルクリックするか、[プロパティ] ビューで [ダイアログ (...)] をクリックして [定義] ダイアログを開きます。
3. ダイアログで、次の変数タイプを選択します。
 - WSDL メッセージ ([「WSDL メッセージタイプ」](#) (ページ 243)を参照)
 - スキーマタイプ ([「XML スキーマタイプ」](#) (ページ 244)を参照)
 - スキーマ要素 ([「XML スキーマ要素」](#) (ページ 244)を参照)
4. 表示されたリストから必要な名前を選択し、[OK] をクリックします。

変数に初期値を追加する手順については、[「変数の初期化」](#) (ページ 246)を参照してください。

ローカル変数を定義する手順

1. [アウトライン] ビューからスコープを選択します。
2. 上記の手順 2~4 を完了します。

変数は、作成した順序でリストされます。リストの順序を変更するには、[アウトライン] ビューで変数を上下に移動します。

WSDL メッセージタイプ

WSDL メッセージタイプ変数は、プロセス内で参照される WSDL 名前空間で宣言されたメッセージタイプを使用します。

次の図は、WSDL メッセージタイプとして定義された変数の例を示しています。メッセージタイプの名前空間は、BPEL プロセスで宣言されます。

BPEL Process

```
<variable name = "BuyerPO"  
  messageType = "Ins:POMsg"/>
```

WSDL File

```
<portType name = "poPT">  
  <operation name = "PO">  
    <input message = "Ins:POMsg"/>  
  </operation>  
</portType>
```

XML スキーマタイプ

XML スキーマタイプ変数は、現在定義されている名前空間のスキーマに組み込まれた単純型または複合型の要素を使用します。以下に例を示します。

```
<variable xmlns:props="http://example.com/shipProps"  
  name="itemsShipped" type="props:itemCountType"/>
```

XML スキーマ要素

XML スキーマ要素変数は、次の例に示すように、現在定義されている名前空間のスキーマで定義された要素を使用します。

BPEL Process

```
<variable name="buyer_PO">  
  <element name="pOrd"/>  
</variable>
```

WSDL Schema

```
<xsd:element name="pOrd">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element ref="pO"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

変数プロパティとプロパティエイリアスの追加

相関セットで使用する WSDL で定義されたプロパティを選択します。

プロパティは、メッセージ以外の特定の 변수に含まれる定義の詳細からプロセスのロジックを分離する方法として役立ちます。プロセスは、プロパティを使用して、変数の初期化ロジックを 1 つの場所に分離し、その変数を操作できるようにするために変数のプロパティを設定および取得します。変数の定義が後から変更された場合、その変数を操作するための残りのプロセス定義は変更されず、元のままになります。

変数には複数のプロパティを含めることができ、それぞれのプロパティはプロパティエイリアスとして定義します。

プロパティ

プロパティは以下のいずれかになります。

- xsd:int や xsd:string などのスキーマタイプ
- 要素

例えば、注文書番号や顧客 ID などがプロパティに該当します。プロパティは、Web サービスのインタラクション中に送信される WSDL メッセージ内、または任意のプロセス変数内に存在します。1 つの変数には複数のプロパティが含まれることがあり、また、1 つのプロパティが複数の変数に存在する場合もあります。

プロパティエイリアス

変数とプロパティのペアごとにプロパティエイリアスが存在し、これは、変数部分が単純型、スキーマ要素、または複合型であるかどうかに関係なく、任意の変数からプロパティの値を抽出する方法を特定します。パートが要素または複合型である場合、要素または型の中にあるプロパティの場所を特定するためには、XPath (または他の言語) クエリが必要です。

詳細については、以下のトピックを参照してください。

- [「プロパティ名およびエイリアスの WSDL 構文と例」 \(ページ 369\)](#)
- [「プロパティ定義の作成」 \(ページ 371\)](#)
- [「プロパティエイリアスの作成」 \(ページ 374\)](#)

変数の初期化

すべての変数は、使用する前に初期化する必要があります。変数または変数のパートは、アクティビティでのメッセージの受信、割り当てまたは入力マッピングでのデータの割り当て、変数定義への初期値の追加など、複数の方法で初期化できます。

Receive、onMessage、onEvent、および Inbound Invoke アクティビティの変数は自動的に初期化されます。他のプロセス変数については、宣言を行う際に初期化できます。宣言時に初期化することで、変数を初期化する Assign アクティビティをスキップすることができます。

変数の初期値は、プロセスの実行中にスキーマまたは WSDL 定義に照らし合わせて検証されます。また、変数を検証アクティビティに追加して、変数を検証することもできます。詳細については、[「変数の検証」 \(ページ 258\)](#)を参照してください。

変数に初期値を追加する手順

1. [「変数の追加」 \(ページ 243\)](#)に記載されているように、新しいプロセス変数を追加します。
2. [アウトライン] ビューまたは [プロセス変数] ビューから、変数を選択します。
3. [プロパティ] ビューで、[初期値] の横にある [ダイアログ (...)] ボタンを選択します。
4. **[変数の初期化]** ダイアログで、変数に値を割り当てるための詳細を選択します。この選択内容は、コピー操作の *From* 側の選択と同様です。*From Type* および関連する属性の説明については、[「割り当て」 \(ページ 173\)](#)を参照してください。

Receive などの開始アクティビティを含むスコープで宣言されている変数の場合、日時関数などといった変更される可能性のある値で変数を初期化しないでください。この制限に従うことで、すべてのプロセスインスタンスに対する事前に計算された値が保証されます。

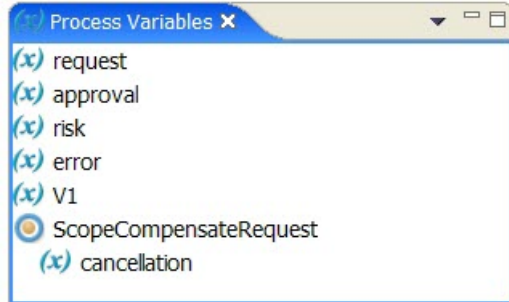
変数の表示

[プロセス変数] ビューには、変数を簡単に定義して使用するために役立つ、次のような複数の機能があります。

- [アウトライン] または Process Editor キャンバスでアクティビティを選択して、使用中の変数をすばやく表示する。[「アクティビティで使用する変数のクイックビュー」 \(ページ 247\)](#)を参照してください。
- [\[使用した場所の検索\]](#) を使用して、変数のすべての使用箇所を検索する。[「変数の使用箇所の検索」 \(ページ 255\)](#)を参照してください。
- 完全な変数定義（メッセージとパート、パス、プロパティ、要素とその属性、および単純型）を表示する。[「アイコンの記号の理解と変数パートの説明」 \(ページ 248\)](#)を参照してください。
- 変数をドラッグアンドドロップして、新規または現在の割り当ての [\[コピー\]](#) 操作を作成する。[「\[プロセス変数\] ビューのオプションの使用」 \(ページ 247\)](#) および [「コピー操作での変数の使用」 \(ページ 256\)](#) を参照してください。

- メッセージ変数と要素変数のサンプルデータを追加、ロード、生成、編集する。[「\[プロセス変数\] ビューでのサンプルデータの使用」 \(ページ 249\)](#)を参照してください。

次の図は、[プロセス変数] リストの例を示しています。



アクティビティで使用される変数のクイックビュー

Process Editor キャンバスまたは [アウトライン] ビューでアクティビティを選択すると、そのアクティビティで使用されている変数が、選択された状態で [プロセス変数] ビューに表示されます。

[プロセス変数] ビューのオプションの使用

[プロセス変数] ツールバーのオプションの選択リストは便利な機能です。次の表にオプションの一覧を示します。

オプション	説明
変数の表示	変数リストを表示し、変数を開くことができます。リストが非表示の場合に便利です。 「定義を表示するために変数を開く」 (ページ 248) を参照してください。
変数リストの表示	デフォルトで選択されています。選択を解除すると変数のリストが非表示になり、ビューで開いた変数用のスペースが増えます。
内部変数の表示	デフォルトで選択解除されています。選択すると、Invoke、Receive、および Reply アクティビティ内のデータマッピング用に作成された暗黙的なスコープに関連付けられたコンテナ変数が表示されます。コンテナ変数は、名前付きの parameters および attachmentCopyResult です。詳細については、 「入力変数」 (ページ 197) を参照してください。
変数データの表示	サンプルデータビューに変数を表示します
変数定義の表示	定義ビューに変数を表示します
横方向のレイアウト	変数を左から右に開きます
縦方向のレイアウト	変数を上から下に開きます
変数データのロード	最後に保存されたサンプルデータのバージョンを表示します

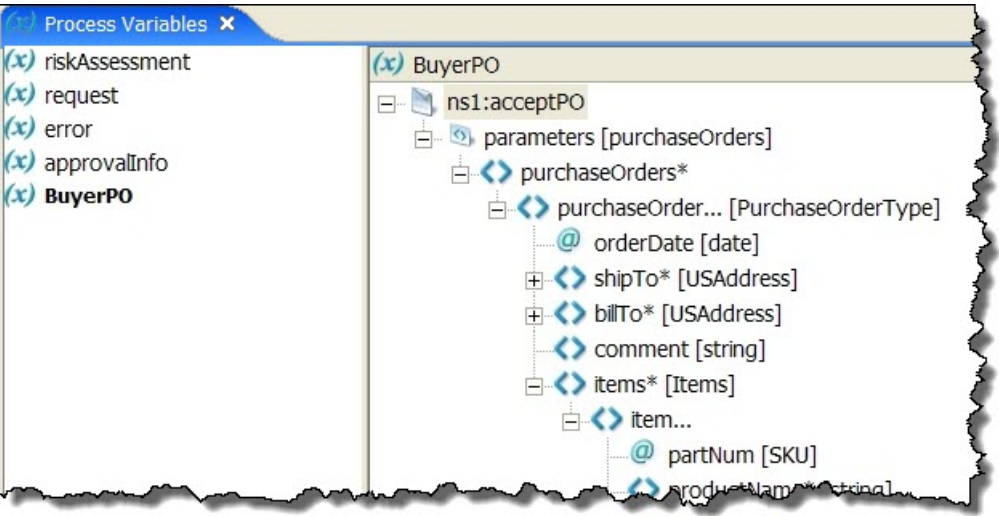
ヒント: [プロセス変数] ビューで変数のリストの順番を並び替えるには、[アウトライン] ビューに移動して、変数を上下に移動します。

定義を表示するために変数を開く

[プロセス変数] ビューで、1 つの変数、または選択した変数を開くことができます。

- 1 つの変数を開くには、変数をダブルクリックします
- 選択した変数を開くには、変数を選択してから、右マウスメニューの【すべて開く】を選択します
- リスト内のすべての変数を開くには、【すべて開く】を選択します

次の図は、BuyerPO 変数を開いた状態の例を示しています。



変数プロパティの表示

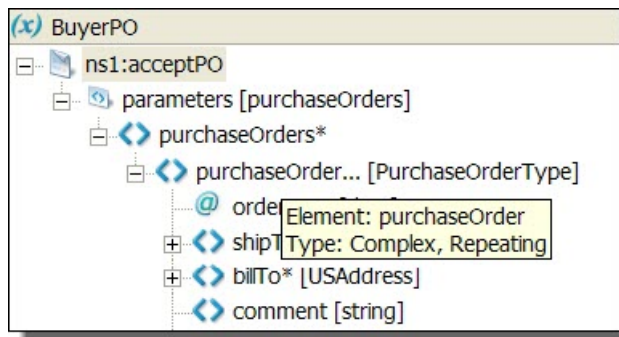
[プロセス変数] ビューまたは [アウトライン] から変数を選択して、[プロパティ] ビューにフォーカスします。

次の表に、変数に必須のプロパティとオプションのプロパティを示します。

必須のプロパティ	オプションのプロパティ
名前	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
定義	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	初期値。 「変数の初期化」 (ページ 246)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

アイコンの記号の理解と変数パートの説明

変数の各パートを理解するには、図のように、マウスカーソルを変数の上に置いて説明を表示します。



変数定義には次の内容を含めることができます。

- メッセージパートとプロパティ
- WSDL スキーマタイプ
- WSDL スキーマ要素と属性

変数パート、プロパティ、要素、または属性の左側にはアイコンが表示されます。角かっこのペア (<>) などのアイコンは、要素などの構造エンティティに関連付けられています。

[「ドラッグアンドドロップを使用したコピー操作の作成」 \(ページ 256\)](#)に記載されているように、コピー操作で使用する変数には追加の視覚情報があります。

さらに、パートの右側には次のような記号が表示されます。

- **アスタリスク** (必須フィールドを示します)
- **省略記号** (繰り返し要素を示します)

変数の削除

プロセスから変数を削除できます。

1. Process Developer のメインメニューから **【プロセス】** を選択します。
2. **【削除】** > **【変数】** を選択します。
3. **【変数の削除】** ダイアログから変数を選択し、**【OK】** をクリックします。

【プロセス変数】 ビューでのサンプルデータの使用

追加するサンプルのソースの場所を選択します。サンプルデータ、WSDL、XSL ファイルなどのワークスペースリソースを選択します。

WSDL メッセージに追加するサンプルデータ値とファイルが、プロセス変数に自動的にロードされます。シミュレーション中に使用するために、**【プロセス変数】** ビューでメッセージ変数のサンプルデータをロードできます。サンプル値は、プロセスシミュレーションの入力、出力、またはフォールト変数を初期化する場合に役立ちます。これらの変数は、プロセスの実行をシミュレートする前に初期化する必要があります。

サンプル値を変数に追加する場合、次のような複数の方法があります。

- [「WSDL メッセージへのサンプルデータファイルの追加」 \(ページ 115\)](#)
[パーティシパント] ビューまたは [インタフェース] ビューで WSDL メッセージにサンプルを追加した場合は、メッセージのデフォルトのサンプル値がメッセージタイプ変数に自動的にロードされます。
- [「単純型のメッセージパート用の単一のサンプルデータ値の編集」 \(ページ 250\)](#)
WSDL メッセージの単純型メッセージパートにサンプルデータ値を追加した場合は、メッセージのデフォルトのサンプル値がメッセージ型変数に自動的にロードされます。その値を編集するか、[プロセス変数] ビューで新しい値を追加できます。
- [「サンプルデータファイルの生成」 \(ページ 114\)](#)
複合型の変数の場合は、変数のルート部分または要素に基づいて XML ドキュメントを生成できます。
- [「\[プロセス変数\] ビューでのサンプルデータファイルのロード」 \(ページ 250\)](#)
複合的な変数の場合、ロードする XML ドキュメントを選択できます。サンプルファイルのデータを編集して保存するか、新しいファイルとして保存することができます。

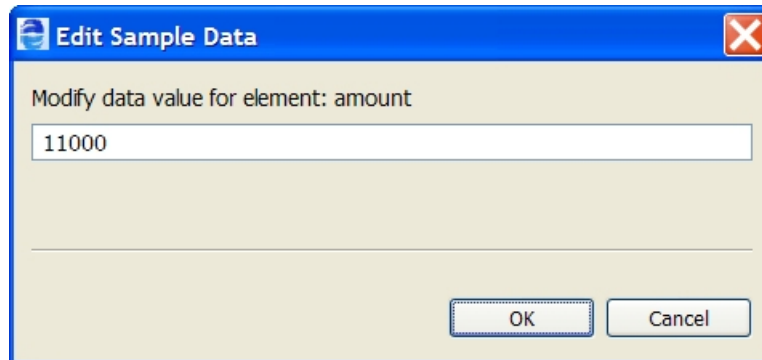
[データ] ビューで変数を開くたびに、Process Developer では以前にロードしたデータ値が表示されます。

単純型のメッセージパート用の単一のサンプルデータ値の編集

[プロセス変数] ビューで、単純型のメッセージパートにサンプルデータ値を追加できます。プロセスをテストする準備が整うと、サンプルメッセージパートの値が変数へ自動的にロードされます。また、[パーティシパント] ビューまたは [インタフェース] ビューに値を追加したり、プロセスシミュレーション中に値を上書きしたりすることもできます。詳細については、[「単純型のメッセージパートへのサンプルデータ値の追加または編集」 \(ページ 113\)](#)を参照してください。

単一のサンプルデータ値を追加する手順

1. [プロセス変数] リストで変数を開きます。
2. 変数名またはパート名を右クリックし、[変数] > [データの表示] を選択します。
3. 図のように、変数をダブルクリックして [サンプルデータの編集] ダイアログを開きます。



4. 値を入力して [OK] をクリックします。開いた変数の [データ] カラムに値が表示されていることに注意してください。

[プロセス変数] ビューでのサンプルデータファイルのロード

効率的な方法として、複合変数用の XML データファイルを準備した後に、[インタフェース] ビューにサンプルを追加することができます。サンプルデータの詳細については、[「WSDL メッセージのサンプルデータの使用」 \(ページ 113\)](#)を参照してください。

サンプルデータファイルをロードする手順

1. WSDL 名前空間からのメッセージまたは要素構造を表すサンプルデータの XML ファイルを作成します。XML ファイルは、メッセージタイプまたは要素のスキーマに準拠している必要があります。
2. 効率的な方法として、Orchestration プロジェクトにサンプルデータを追加します。サンプルは、BPEL プロセス全体で使用することができます。詳細については、「[サンプルデータファイルの生成](#)」(ページ 114)および「[WSDL メッセージへのサンプルデータファイルの追加](#)」(ページ 115)を参照してください。
3. [プロセス変数] リストで複合変数を開きます。
4. 変数名またはパーツ名を右クリックし、[データの表示] を選択します。
5. 変数パートを右クリックして、[データのロード] を選択します。
6. 次のいずれかを選択します。
 - **登録済みのサンプル (デフォルト)**。サンプルデータファイルを追加してデフォルトファイルを再ロードした場合や複数のデータファイルが存在する場合、またはデフォルトを変更した場合は、これを選択します。
 - **プロジェクトファイルから**。プロジェクトエクスプローラプロジェクトにサンプル XML ファイルがある場合は、これを選択します。
 - **登録済みのサンプルから**。[インタフェース] ビューにメッセージのサンプル XML ファイルがある場合は、これを選択します。
7. ファイルの選択後に、複雑な部分を展開してデータ値を表示できます。

[プロセス変数] ビューでのサンプルデータの保存と表示

複合プロセス変数のサンプルデータをロードまたは生成した後に、変数部分を右クリックして、データをファイルに保存するか、ダイアログで表示することができます。

可変部分のコンテキストメニューで、[名前を付けてデータを保存] を選択して新しい XML ファイルを作成します。

[XML データの表示] を選択して、変数の内容を簡単に確認することが可能なダイアログを開きます。

XML データウィザードの使用

[設定] ページの詳細については、以下を参照してください。

概要

XML データウィザードは、次のタイプのデータを自動的に生成します。

- 複合型である WSDL メッセージの**サンプルデータファイル**
「[サンプルデータファイルの生成](#)」(ページ 114)に記載されているように XML データウィザードを開きます
- コピー操作の From 側の複合変数の**リテラルコンテンツ**
「[コピー操作のリテラルコンテンツの例](#)」(ページ 178)に記載されているように XML データウィザードを開きます。
- 変数の**初期値**のタイプとして [リテラル] が選択されている場合の複合変数のリテラルコンテンツ
「[変数の初期化](#)」(ページ 246)に記載されているように XML データウィザードを開きます。次のダイアログが表示されます。

XML Data Wizard

Preferences

Select a root element and preferences for sample data generation.

Select Root Element

☐ Complex Type ☒ Element

loan:errorMessage

☒ Generate optional attributes and elements

☒ Generate data for attributes and elements

Repeating Elements: 1

Maximum Recursion Depth: 2

Choices, Abstract Elements and Abstract Types

☒ Generate first available only

☐ Generate first available with others commented out

< Back Next > Finish Cancel

[設定] ページで次のように選択します。

選択	説明
ルート要素の選択	
複合型または要素	複合型または要素は、選択した変数に基づいて事前に選択されています。選択リストから、一致するタイプまたは要素を選択できます。
オプションの属性と要素を生成する	スキーマ定義にオプションの属性（つまり、use="optional"）と要素（つまり、minOccurs="0"）が含まれている場合は、必要に応じてそれらの属性と要素をデータファイルに含めることができます。 属性と要素を含めない場合、要素の選択に使用するクエリによっては、プロセスの実行時に選択エラーフォールトが発生する可能性があります。

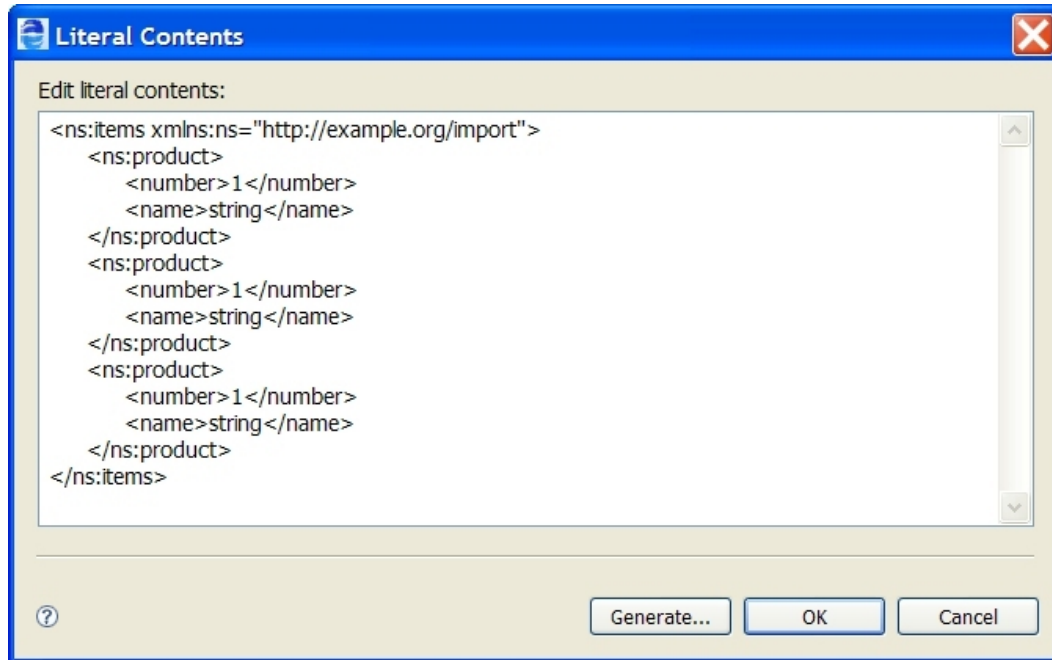
選択	説明
属性と要素のデータを生成する	<p>要素と属性に適切な値を生成する場合は、これを選択します。正規表現パターンを持つ派生単純型など、一部のケースでは、値が有効なスキーマではないことがあります。多くの場合、この値は、必要に応じて編集を開始するための適切な開始点となります。</p> <p>要素または属性に固定値またはデフォルト値が含まれる場合は、その値が自動的に使用されます。参照されるタイプに列挙された制限が含まれる場合は、最初の値がこの制限から選択されます。</p> <p>空の要素と属性を生成するには、この項目の選択を解除します。データがない状態で要素を生成した場合、有効なスキーマを持つサンプルは生成されません。変数の初期化およびコピー操作の場合、データの生成後に [リテラルコンテンツ] 編集ボックスに独自の値を追加できます。インタフェースとプロセス変数のサンプルデータの場合、生成された XML ファイルを開いて編集できます。</p>
反復する要素	複数の項目を含む発注書など、繰り返される要素を持つ変数の場合、その要素に対して生成する項目の数を指定できます。デフォルトは 1 です。
最大再帰深度	<p>再帰的なスキーマタイプの再帰の深さを設定します。デフォルトの深さは 2 で、生成されるデータは次のようになります。</p> <pre><foo> <child /> <foo> <child /> <foo /> </foo> </foo></pre>
選択肢は、抽象要素と抽象タイプです	
最初に使用可能なもののみを生成する	<p>スキーマタイプに、結果のドキュメントに表示される選択肢の 1 つのみを指定する選択肢構造を含めることができます。最初に使用可能な要素は、最初の子要素です。</p> <p>スキーマ要素には、1 つ以上の抽象要素を含めることができます。これらの要素は抽象的であるため、直接作成することはできません。代わりに、最初に使用可能な要素がその置換グループから選択されます。最初に使用可能な要素は、インポートされたスキーマ内で最初に検出された要素です。</p>
最初に使用可能なものを生成し、他のものはコメント化する	使用可能なすべての要素を生成し、最初の要素以外のすべての要素が <!-- --> タグ内に生成されるようにする場合は、これを選択します。

例 1: 反復する要素

element=ns:items に対して次のスキーマ定義があるとして。

```
<xs:element name="items" type="imp:ItemsType" />
<xs:complexType name="ItemsType">
  <xs:sequence>
    <xs:element ref="imp:product" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:element name="product" type="imp:ProductType" />
<xs:complexType name="ProductType">
  <xs:sequence>
    <xs:element name="number" type="xs:integer" />
    <xs:element name="name" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

[反復する要素] が 3 と設定されている場合、次のリテラルコンテンツ（またはサンプルデータファイル）が生成されます。

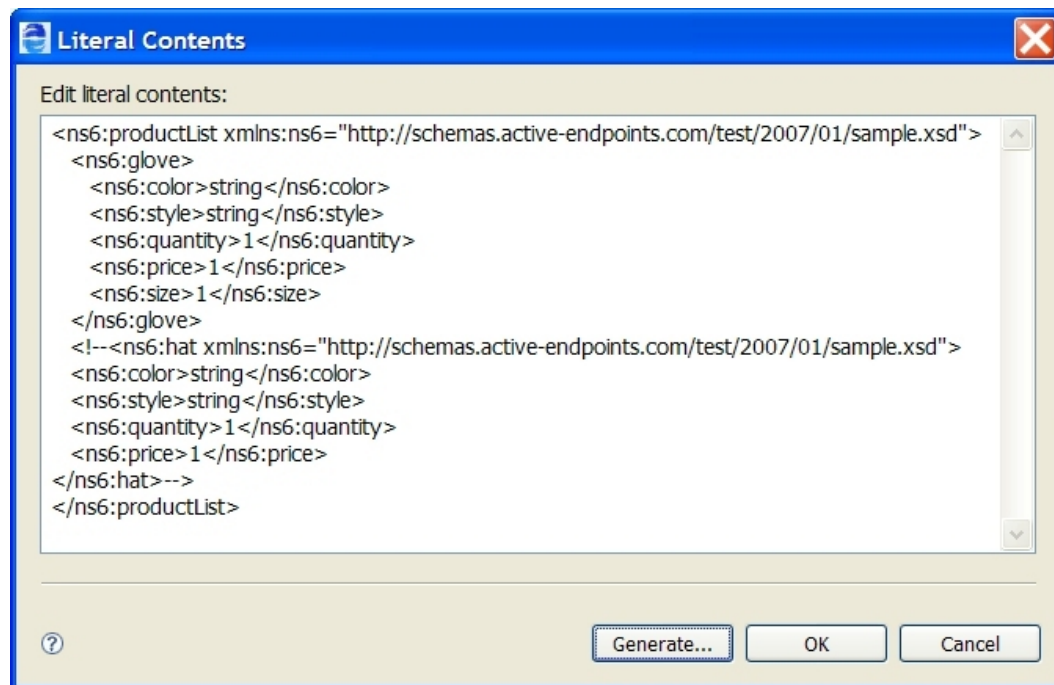


例 2: 最初に使用可能なものを生成し、他のものはコメント化する

element=ns6:productList に対して次のスキーマ定義があるとします。

```
<xs:element name="productList">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:product"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="product" type="tns:ProductType"
  abstract="true"/>
<xs:element name="hat" substitutionGroup="tns:product"
  type="tns:ProductType"/>
<xs:element name="glove" substitutionGroup="tns:product"
  type="tns:SubProductType"/>
<xs:complexType name="ProductType">
  <xs:sequence>
    <xs:element name="color" type="xs:string"/>
    <xs:element name="style" type="xs:string"/>
    <xs:element name="price" type="xs:decimal"/>
    <xs:element name="quantity" type="xs:positiveInteger"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SubProductType">
  <xs:complexContent>
    <xs:extension base="tns:ProductType">
      <xs:sequence>
        <xs:element name="size" type="xs:int" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

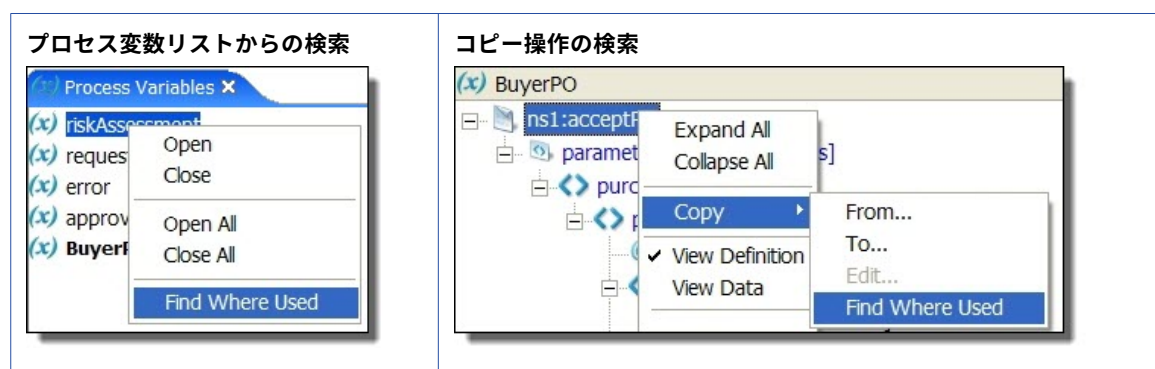
次のリテラルコンテンツ（またはサンプルデータファイル）が生成されます。



変数の使用箇所の検索

〔プロセス変数〕 リストの変数からマウスの右クリックメニュー **〔使用した場所の検索〕** を使用して、アクティビティおよびリンク内の変数のすべての出現箇所を検索できます。

また、コピー操作で使用するすべての変数を検索することもできます。コピー操作での変数の出現箇所を検索するには、図のように、変数上のマウスの右クリックメニュー **〔コピー〕** から **〔使用した場所の検索〕** を選択します。



Process Developer に検索結果が表示され、Process Editor キャンバスと **〔アウトライン〕** ビューで最初の結果が強調表示されます。**〔アウトライン〕** ビューから、編集するアクティビティ、リンク、またはコピー操作を選択できます。

コピー操作での変数の使用

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

[プロセス変数] ビュー内では、現在の Assign のコピー操作を作成および編集したり、新しい Assign を自動的に作成したりすることができます。

変数のコンテキストメニューを使用するか、変数パーツを別の変数パーツにドラッグアンドドロップすることで、コピー操作を作成できます。

[FROM タイプ] を選択し、他の適切な選択を行ってコピー操作を完了します。選択した Assign アクティビティのコピー操作およびスクリプトを追加、編集、コピー、削除、および再編成します。

詳細については、以下を参照してください。

- [「コンテキストメニューを使用したコピー操作の作成」 \(ページ 256\)](#)
- [「ドラッグアンドドロップを使用したコピー操作の作成」 \(ページ 256\)](#)
- [「編集するコピー操作の選択」 \(ページ 257\)](#)

コンテキストメニューを使用したコピー操作の作成

ヒント: この手順に従って、任意のタイプのコピー操作を作成します。変数から変数へのコピー操作をすばやく作成する方法については、[「ドラッグアンドドロップを使用したコピー操作の作成」 \(ページ 256\)](#)を参照してください。

1. 次のいずれかを実行します。
 - コピー操作を追加する場合は、Process Editor キャンバスから Assign アクティビティを選択します。コピーは、現在のコピー操作の後に追加されます。
 - 新しい Assign アクティビティを自動的に作成する場合は、手順 2 に進みます。
2. [プロセス変数] リストで、コピー操作の *From* 句または *Copy* 句の変数を開きます。
3. マウスを右クリックして、[コピー] > [コピー元] を選択して *From* 句を作成するか、[コピー] > [コピー先] を選択して *Copy* 句を作成します。
4. [コピー操作] ダイアログで、残りの句を完了し、[OK] をクリックします。

Process Editor キャンバスと [アウトライン] ビューで、現在の Assign アクティビティまたは新しい Assign アクティビティが選択されます。

ドラッグアンドドロップを使用したコピー操作の作成

ドラッグアンドドロップで、次のタイプのコピー操作を作成できます。

From Variable [Part] To Variable [Part]
From Variable Property To Variable Property

詳細については、[「割り当て」 \(ページ 173\)](#)を参照してください。

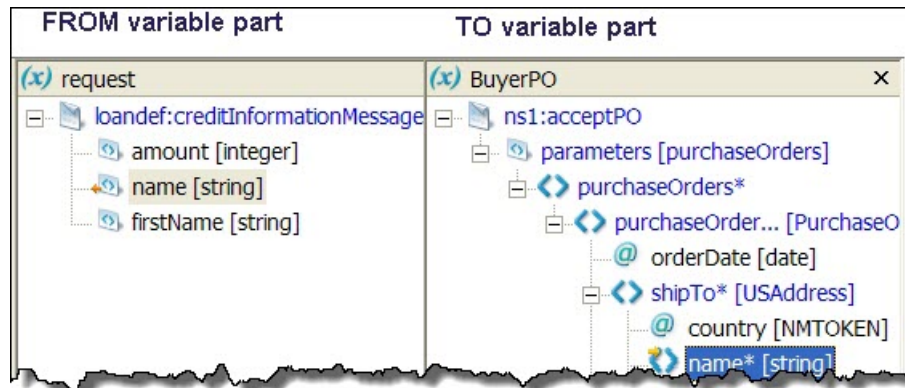
ドラッグアンドドロップでコピー操作を作成する手順

1. 次のいずれかを実行します。
 - コピー操作を追加する場合は、Process Editor キャンバスから Assign アクティビティを選択します。コピーは、現在のコピー操作の後に追加されます。
 - 現在選択されているスコープで新しい Assign アクティビティを自動的に作成する場合は、手順 2 に進みます。

2. [プロセス変数] リストで、コピー操作の *From* 句と *To* 句の 2 つの変数を開きます。
3. *From* 変数の変数、パーツ、またはプロパティを *To* 変数の適切な変数、パーツ、またはプロパティにドラッグします。

ヒント: マウスボタンを離す前に、Process Developer ステータスバーでコピーの仕様を表示できます。

マウスボタンを離すと、使用中のパーツが強調表示された状態で Process Developer に結果が表示されます。また、図に示すように、*From* パーツアイコンには外向きの矢印があり、*To* パーツアイコンには内向き矢印があります。



Process Editor キャンバスと [アウトライン] ビューで、現在の Assign アクティビティまたは新しい Assign アクティビティが選択されます。

編集するコピー操作の選択

変数の右マウスメニューから [コピー] > [編集] を選択することで、コピー操作を編集できます。

変数に複数のコピー操作がある場合は、[コピー操作] ダイアログが開き、編集する操作を選択できます。

WSDL フォールトメッセージに基づく変数の使用

Web サービスインタラクションアクティビティでの WSDL メッセージパートのマッピング

receive、onMessage、onEvent、invoke、reply アクティビティなどの Web サービスインタラクションアクティビティでは、通常、入力変数または出力変数に WSDL メッセージタイプ変数を選択します。ただし、WSDL 操作で、要素を使用して定義された 1 つのパートのみを含むメッセージを使用する場合、Web サービスインタラクションアクティビティでは、入力変数に<fromPart>要素が使用され、出力変数に<toPart>要素が使用されます。アクティビティの入力または出力変数宣言の代わりとして、*fromPart*または *toPart* 宣言が使用されます。

変数の検証

Process Developer には、シミュレーションまたは実行時に変数の値を検証するための複数の明示的な方法が用意されています。変数の値を検証するための方法は次のとおりです。

- **Validate アクティビティ**

Validate アクティビティにはプロセス変数のリストが含まれ、関連する XML または WSDL データ定義に照らし合わせて各プロセス変数の値を検証します。詳細については、[「検証」 \(ページ 212\)](#)を参照してください。

- **Assign アクティビティの検証属性**

この属性は、Validate アクティビティと同じように機能します。Assign アクティビティで検証が有効になっている場合、Process Developer は、割り当てのすべてのコピー操作で使用されるすべての変数を検証します。

- **コピー操作のコピー先側で要素ベースの変数の値を検証するために、コピー操作でソース要素名の属性を保持します。**

ソース要素名の保持属性は、要素間のコピー操作でのみ機能し、他のタイプのコピー操作には適用されません。Process Developer では変数全体に対する検証は実行されず、ルート要素が XML スキーマまたは WSDL 定義に対して有効であることが検証されます。これは、コピー先の要素が変数内の多くの要素の 1 つにすぎないためです。このタイプのコピー操作では、ソース要素とコピー先の要素が同じスキーマ置換グループの一部である必要があります。置換グループを使用することで、ある要素を他の要素に置き換えることができます。例については、[「コピー操作のクエリと式の例」 \(ページ 177\)](#)を参照してください。

- **入出力メッセージの変数を検証するための設定**

詳細については、[「シミュレーションの設定」 \(ページ 427\)](#)を参照してください。

変数値が無効な場合は、`bpel:invalidVariables` フォールトがスローされます。

変数の添付ファイルの使用

変数には、添付ファイルとしてプロセスに加わる非構造化データを含めることができます。変数の添付ファイルを使用した追加、削除、シミュレーション、およびデバッグの詳細については、[第 14 章, 「添付ファイル」 \(ページ 259\)](#)を参照してください。

第 14 章

添付ファイル

ビジネスプロセスで使用される多くのメッセージには、非構造化データへの参照が含まれています。例えば、補償請求を行う保険のビジネスプロセスでは、損傷した車両の写真が参照されることがあります。こうした非構造化データを Process Developer に含めるため、BPEL プロセスの変数に画像、ドキュメント、スプレッドシート、またはその他のファイルタイプを添付できます。

添付ファイルは特定の変数から別の変数に受信、送信、およびコピーできます。また、カスタム関数を使用して、添付ファイルまたは添付ファイルプロパティの内容をプロセス変数にコピーすることもできます。

添付ファイルに関連するトピックは次のとおりです。

- [「添付ファイルの追加」 \(ページ 259\)](#)
- [「シミュレーション用の添付ファイルの追加」 \(ページ 259\)](#)
- [「添付ファイルを操作するためのカスタム関数」 \(ページ 260\)](#)
- [「添付ファイルのカスタム関数の例」 \(ページ 263\)](#)

添付ファイルの追加

[「添付ファイルを操作するためのカスタム関数」 \(ページ 260\)](#)に記載された一連の関数を使用して、任意のプロセス変数に添付ファイルを追加したり、添付ファイルを操作したりすることができます。

現在、Process Developer で添付ファイルを表示するには、[「シミュレーション用の添付ファイルの追加」 \(ページ 259\)](#)に記載されたシミュレーションプロパティとして添付ファイルを追加します。

プロセス変数が初期化されていない場合は、添付ファイルをコピーする前に、必ず初期化してください。詳細については[「変数の初期化」 \(ページ 246\)](#)を参照してください。

シミュレーション用の添付ファイルの追加

変数に添付ファイルを追加して、実行中のプロセスが通常送受信する添付ファイルをシミュレートできます。また、添付ファイルにプロパティを追加してから、getAttachmentProperty 関数を使用してプロパティを操作することもできます。詳細については、[「添付ファイルを操作するためのカスタム関数」 \(ページ 260\)](#)を参照してください。

シミュレーションの説明については、[「BPEL プロセスの実行のシミュレーション」 \(ページ 416\)](#)を参照してください。

シミュレーション用の変数に添付ファイルを追加する手順

1. Process Editor キャンバスから、Receive、Invoke、onMessage、または onEvent アクティビティを選択します。
2. [プロパティ] ビューの [シミュレーション] で、次のいずれかを実行します。
 - Receive、onMessage、onEvent の場合は、[添付の入力] 行の最後にある [ダイアログ (...)] ボタンをクリックします。
 - Invoke の場合は、[出力添付ファイル] または [フォールト添付ファイル] 行の最後にある [ダイアログ (...)] ボタンをクリックします。
3. [入力/出力/フォールト添付ファイル] ダイアログで、[追加] を選択します。
4. [添付ファイル] ダイアログで、添付ファイルの場所として [ファイルシステム] または [プロジェクト] を選択してから、添付ファイルを参照して選択します。
5. 既知のコンテンツプロパティが識別されることに注意してください。例えば、GIF 画像を添付する場合、コンテンツタイプは *image/gif* です。
添付ファイルのサイズをバイト単位で示すために追加される *X-Size* プロパティに注意してください。
6. 必要に応じて、既知のプロパティを編集するか、次のように新しいプロパティを追加します。
 - a. [追加] を選択して、[添付プロパティ] ダイアログを開きます。
 - b. プロパティの名前と値を入力します。プロパティは、添付ファイルの既知のプロパティである必要があることに注意してください。
 - c. [OK] をクリックします。
7. [OK] をクリックして、[添付ファイル] ダイアログを閉じます。
8. 別の添付ファイルを追加するには、[入力/出力/フォールト] ダイアログで [追加] を選択し、添付ファイルの追加を続行します。

添付ファイルを追加した後、[「BPEL プロセスのシミュレーションの開始と終了」 \(ページ 417\)](#)に記載されているようにシミュレーションを開始します

シミュレーション中に、添付ファイルのあるプロセス変数を開きます。プロセスが添付ファイルを受信、送信、またはコピーすると、[プロセス変数] ビューで結果を表示できます。

添付ファイルのプロパティは、既知のプロパティに準拠している必要があります。詳細については、[「添付ファイル进行操作するためのカスタム関数」 \(ページ 260\)](#)の「`abx:getAttachmentProperty`」を参照してください。

添付ファイル进行操作するためのカスタム関数

式、クエリ、およびその他のビルダでは、添付ファイル进行操作するために次の関数を選択することができます。これらの機能は、[BPEL] > [Process Developer] > [添付ファイル] カテゴリにあります。

注: ネストされた添付ファイルについては、SOAP メッセージを使用した送信はサポートされていません（ネストされた添付ファイルをサポートしない複数の添付ファイルを含む SOAP メッセージを送信する場合）。テスト中に、SoapUI Disable のマルチパートプロパティを `false` に設定することをお勧めします。

関数の使用法を確認するには、[「添付ファイルのカスタム関数の例」 \(ページ 263\)](#)を参照してください。

`abx:base64EncodeAttachment(variableName, attachmentNumber)`

名前付き変数の Base64 でエンコードされた添付ファイルを返します。Base64 は、MIME コンテンツ転送エンコーディングを参照します。

パラメータ:

- `variableName`: 添付ファイルを含む変数の名前
- `attachmentNumber`: 1 から始まるソース変数の添付ファイル項目番号

abx:getAttachmentCount(variableName)

名前付き変数に関連付けられた添付ファイル項目の数である整数を返します。

abx:copyAttachment(fromVariableName, fromItemNumber, toVariableName)

添付ファイルの 1 つをソース変数からターゲット変数にコピーします。戻り型は `xs:boolean` です。コピーが失敗した場合、添付ファイルのフォールトがスローされます。

パラメータ:

- `fromVariableName`: ソース変数の名前
- `fromItemNumber`: 1 から始まるソース変数の添付ファイル項目番号
- `toVariableName`: ターゲット変数の名前

abx:replaceAttachment(fromVariableName, fromItemNumber, toVariableName, toItemNumber)

ターゲット変数の添付ファイルの 1 つを、ソース変数の添付ファイルに置き換えます。戻り型は `xs:boolean` です。置き換えが失敗した場合、または項目番号が範囲外の場合、添付ファイルのフォールトがスローされます。

パラメータ:

- `fromVariableName`: ソース変数の名前
- `fromItemNumber`: 1 から始まるソース変数の添付ファイル項目番号
- `toVariableName`: ターゲット変数の名前
- `toItemNumber`: 1 から始まるターゲット変数の添付ファイル項目

abx:removeAttachment(variableName, itemNumber)

`toItemNumber` で識別される名前付き変数の添付ファイルを削除します。戻り型は `xs:boolean` です。削除が失敗した場合、または項目番号が範囲外の場合、添付ファイルのフォールトがスローされます。

パラメータ:

- `variableName`: 添付ファイルを含む変数の名前
- `itemNumber`: 1 から始まる添付ファイルの項目番号

abx:createAttachment(variableName, contentType, encodedContent [, contentId])

コンテンツタイプとコンテンツを指定する名前付き変数に添付ファイルを追加します。戻り型は `xs:boolean` です。作成が失敗した場合、添付ファイルのフォールトがスローされます。

パラメータ:

- `variableName`: 添付ファイルを追加する変数の名前
- `contentType`: `text/xml` や `image/jpg` などの添付ファイルの MIME タイプ
- `encodedContent`: Base64 でエンコードされた添付ファイルのデータコンテンツを含む文字列
- `contentId`: (オプション)。添付ファイルの内容を参照する文字列。このパラメータは、メッセージに複数の添付ファイルがある場合に特定の添付ファイルへアクセスする際に役立ちます。

abx:copyAllAttachments(fromVariableNames, toVariableName)

`fromVariableNames` (`xsd:string` リスト) で識別される変数のリストから、`toVariableName` で識別される変数にすべての添付ファイルをコピーします。戻り型は `xs:integer` です。コピーが失敗した場合、添付ファイルのフォールトがスローされます。

パラメータ:

- fromVariableNames: コピー元の変数の名前。スペースで区切った単一の名前または複数の名前を入力できます。スコープ内のすべての変数を指定するには、*を使用します。
- toVariableName: コピー先の変数の名前

abx:removeAllAttachments(variableNames)

変数のリストからすべての添付ファイルを削除します。戻り型は xs:integer です。

パラメータ:

- fromVariableNames: コピー元の変数の名前。スペースで区切った単一の名前または複数の名前を入力できます。スコープ内のすべての変数を指定するには、*を使用します。

abx:getAttachmentType(variableName, itemNumber)

名前付き変数に関連付けられた添付ファイルの MIME タイプを返します。

パラメータ:

- variableName: 添付ファイルを含む変数の名前
- itemNumber: 1 から始まる添付ファイルの項目番号

abx:getAttachmentProperty(variableName, itemNumber, propertyName)

添付ファイルプロパティに関連付けられた値を返します。

パラメータ:

- variableName: 添付ファイルを含む変数の名前
- itemNumber: 1 から始まる添付ファイルの項目番号
- propertyName: 添付ファイルのプロパティの名前

添付ファイルのプロパティには、添付ファイル付きの *W3C SOAP* メッセージ仕様 (<http://www.w3.org/TR/SOAP-attachments>)、マルチパート/関連 MIME メディアタイプ (RFC 2387)、および Informatica で指定したプロパティが含まれます。

添付ファイル付きの SOAP プロパティには、*Content-Type*、*Content-Transfer-Encoding*、*Content-ID*、および *Content-Location* が含まれます。

Informatica で指定した添付ファイルのプロパティには、*Attached-By* (添付ファイルを送信した認証済みユーザー)、*X-Size* (添付ファイルのサイズ (バイト単位))、*Attachment-Created-At* (添付ファイルがプロセスサーバーによって受信された際のサーバー生成の時間値 (ミリ秒単位)) が含まれます。

abx:getAttachmentSize(variableName, itemNumber)

名前付き変数の変数添付ファイル項目のコンテンツサイズを返します。

パラメータ:

- variableName: 添付ファイルを含む変数の名前
- itemNumber: 1 から始まる添付ファイルの項目番号

abx:setAttachmentProperty(variableName, itemNumber, propertyName, propertyValue)

名前付き変数に関連付けられた添付ファイルの名前付きプロパティを設定して返します。

パラメータ:

- variableName: 添付ファイルを含む変数の名前
- itemNumber: 1 から始まる添付ファイルの項目番号
- propertyName: 添付ファイルのプロパティの名前

- `propertyValue`: 添付ファイルのプロパティの値

この関数は、BIRT レポートが電子メールに添付された PDF である場合に、このレポートに意味のある名前を付ける際に特に役立ちます。

abx:xmlAttachmentToElement(variableName, attachmentNumber)

添付ファイルを返し、添付ファイルを要素スタイル変数に変換します。添付ファイルは XML ドキュメントである必要があります。

パラメータ:

- `variableName`: 添付ファイルを含む変数の名前
- `attachmentNumber`: 1 から始まるソース変数の添付ファイル項目番号

添付ファイルのカスタム関数の例

次のカスタム関数の使用例は、割り当てアクティビティのコピー操作での添付ファイルの操作を示しています。各関数の詳細については、[「添付ファイルを操作するためのカスタム関数」 \(ページ 260\)](#)を参照してください。

例 1:

最初の添付ファイルを `incomingMessage` 変数から `resultMessage` 変数にコピーします。`copyAttachment` の戻り値が、未使用の `tempVar` にコピーされます。

```
<copy>
  <from>abx:copyAttachment
    ("incomingMessage",1,"resultMessage")
  </from>
  <to variable="tempVar"/>
</copy>
```

例 2:

(N - 1) の添付ファイルを削除します。つまり、最後から 2 番目の添付ファイルを `resultMessage` 変数から削除します。この場合、N は `resultMessage` の添付ファイルの数です。`removeAttachment` の戻り値は、未使用の `tempVar` にコピーされます。

```
<copy>
  <from>abx:removeAttachment("resultMessage",
    abx:getAttachmentCount("resultMessage") - 1 )
  </from>
  <to variable="tempVar"/>
</copy>
```

例 3:

最初の添付ファイルの *Content-Transfer-Encoding* プロパティの値を `resultMessage` 変数にコピーします。

```
<copy>
  <from>abx:getAttachmentProperty("resultMessage" , 1 ,
    "Content-Transfer-Encoding")
  </from>
  <to part="propResult" variable="resultMessage"/>
</copy>
```

第 15 章

リンク

リンクに関連するトピックは次のとおりです。

- [「リンクについて」 \(ページ 264\)](#)
- [「リンクに対する Process Developer 拡張機能」 \(ページ 265\)](#)
- [「アクティビティ間へのリンクの追加」 \(ページ 266\)](#)
- [「リンクの実行ルール」 \(ページ 270\)](#)
- [「リンクを使用した設計と構造化されたアクティビティ」 \(ページ 270\)](#)
- [「リンクと結合条件」 \(ページ 271\)](#)

リンクについて

リンクとは、2つのアクティビティを接続し、2つのアクティビティの実行順序を制御するための構造です。BPMN 中心のパレットでは、リンクはエッジと呼ばれます。

リンクには条件を含めることができ、アクティビティがいつ実行されるか、または実行されるかどうかをさらに詳細に制御できます。条件は XPath（または他の言語）式です。

基本アクティビティ間および構造化アクティビティ間には、複数のリンクを追加することができます。また、構造化されたアクティビティに含まれる基本的なアクティビティへのリンクを追加することもできます。

リンクはフロー内で機能します。2つのアクティビティ間にリンクを追加すると、Process Developer では、次に示すようにリンク定義を含むフローアクティビティが作成されます。

XML 構文のリンク

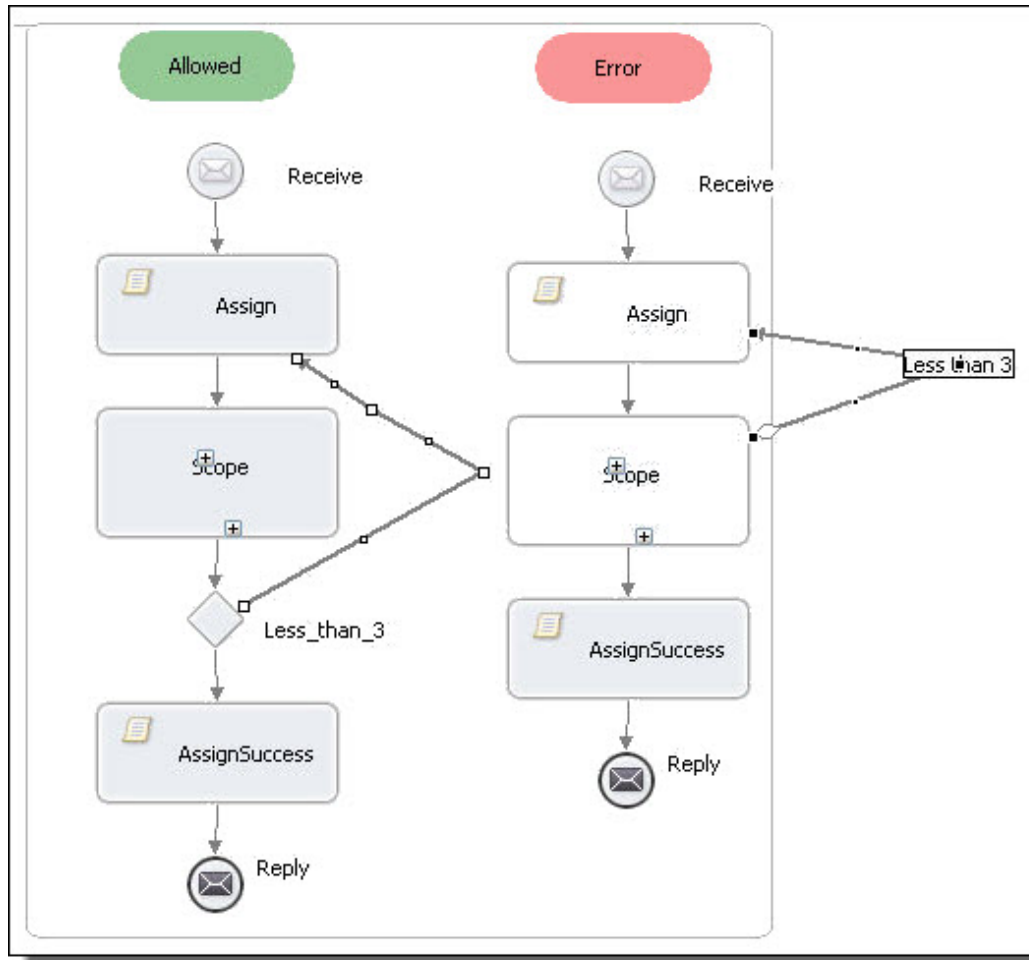
```
<flow>
  <link name="L1" />
  activity1
  <source linkName="L1" />
  ...
  activity2
  <target linkName="L1" />
  ...
</flow>
```

リンクを使用する場合、またはシーケンス、スコープ、およびその他の構造化されたアクティビティを使用する場合に関する詳細については、[「リンクを使用した設計と構造化されたアクティビティ」 \(ページ 270\)](#)を参照してください。

リンクに対する Process Developer 拡張機能

リンクの WS-BPEL 2.0 実行ルールに定められているように、完了したアクティビティにリンクを設定しても、このリンクによって制御サイクルが作成されることはありません。さらに、リンクは、While アクティビティなどの構造化されたアクティビティの境界を越えることはできません。ただし、プロセスサーバーには、特定の条件下でこれらの動作を可能にする拡張機能が含まれています。

次の図は、完了したアクティビティへのループバックリンクを使用した 2 つのケースを示しています。左側は、ループバックリンクの有効な使用例です。右側は、以下で説明する Fork Join 制限に違反しているため、静的解析でエラーが発生するケースです。



デフォルトでは、リンク拡張機能はオンになっており、ループバックリンクを使用することができます。プロセスの [すべて] タブで [リンクは遷移です] プロパティを設定することにより、プロセスごとに拡張機能をオフにできます。

リンク拡張機能の実行ルール

リンクへの Process Developer 拡張機能の基本的な実行ルールは次のとおりです。

- リンクのソースが完了すると、遷移条件が true と評価されるか、条件がない場合、遷移が発生します。リンクのターゲットが他の受信遷移のないアクティビティである場合、アクティビティが実行されます。
- アクティビティの [プロパティ] ビューで、リンクのソースの **相互に排他的な遷移** プロパティが [はい] に設定されている場合に Process Developer 拡張機能のリンクが Repeat Until アクティビティからクロスアウトできることを除いて、リンクがコンテナの境界を越える方法に関する BPEL の制約は引き続き保持され

ます。デフォルトでは、このプロパティは、排他的ゲートウェイの場合には *[はい]* に設定され、それ以外の場合には *[いいえ]* に設定されています。

- アクティビティで「[相互に排他的な遷移](#)」(ページ 221)が *[はい]* に設定されている場合、アクティビティを含むコンテナの通常の制御フローは、アクティビティからのアウトバウンドリンクが実行できない場合にのみ続行されます。動作は標準の BPMN セマンティクスに従うため、プロセス図に基づいて動作を明確にする必要があります。

BPMN は、境界のあるアクティビティの境界をリンクが越えることを許可しません。ただし、BPEL ではリンクがスコープの境界を越えることが許可されるため、有効な BPMN でない場合でも、Process Developer でこの動作が許可されます。

並列実行でのアクティビティの使用制限

2 つ以上のアクティビティが並列で実行され、並列処理の前にループバックするリンクが作成される場合は、並列処理を Fork Join コンテナ内にカプセル化する必要があります。Fork Join によって並列パス上のすべてのアクティビティが完了し、次のループの実行前に結合されるため、この制限により、ループアクティビティが同時に実行されることはありません。

排他タイプのゲートウェイアクティビティのソースリンクとして、相互に排他的なリンクを使用することをお勧めします。排他ゲートウェイからリンクを引き出すと、アウトバウンドリンクの意図された動作が明確になります。Process Developer 拡張機能のリンクをループバックリンクとして使用することは、すでに構造化された While または Repeat Until アクティビティを使用することと類似していることに注意してください。

リンクに対する Process Developer 拡張名前空間

プロセスには、アウトラインの拡張機能ノードのリンクに対する Process Developer 拡張機能が含まれることがあります。

アクティビティ間へのリンクの追加

2 つのアクティビティ間にリンクを追加して、アクティビティの実行順序を制御します。遷移条件のあるリンクおよび遷移条件のないリンクを追加できます。

詳細については、以下を参照してください。

- [「遷移のないリンクの追加」](#) (ページ 266)
- [「遷移条件付きのリンクの追加」](#) (ページ 267)
- [「リンクに対する Process Developer 拡張機能」](#) (ページ 265)
- [「リンクの例」](#) (ページ 269)

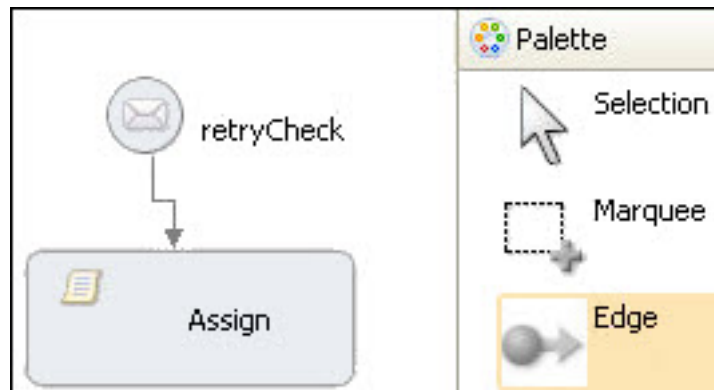
遷移のないリンクの追加

ソースアクティビティからターゲットアクティビティへの単純なリンクを追加して、ターゲットが実行される前にソースが正常に終了するように要求します。

1. リンクする Process Editor キャンバスに 2 つのアクティビティがあることを確認します。
2. ソースアクティビティを選択してから、**[Shift + 選択]** を使用してターゲットアクティビティを選択します。
3. **[リンク]** ツールバーボタンを選択するか、いずれかのアクティビティのマウスの右メニューから **[アクティビティのリンク]** を選択します。

4. Process Developer によって、リンクに L1 という名前が付けられます。リンクをダブルクリックすると、**「遷移ビルダ」** ダイアログが開きます。
5. **「プロパティ」** ビューでは、必要に応じてリンクの名前を変更できます。

ヒント: 必要に応じて、パレットの **「BPMN エッジ」** ツールを使用できます。このツールを使用するには、ツールを選択してからソースアクティビティを選択します。図の左側に示すように、リンクをターゲットに接続します。**「選択」** ツールを選択して、**「エッジ」** ツールをオフにします。



遷移条件付きのリンクの追加

ソースアクティビティの完了に基づいて、遷移条件を持つリンクを追加します。ターゲットアクティビティを開始する前に、条件がブール値 true と評価される必要があります。

1. **「遷移のないリンクの追加」** ([ページ 266](#))に記載されているように、リンクを描画するための手順を完了します。
2. リンクを選択してダブルクリックします。または、**「プロパティ」** ビューで、**「遷移条件」** の横にある **「ダイアログ (...)」** ボタンをクリックします。
3. **「式ビルダの使用」** ([ページ 275](#))に記載されているように、XPath (または他の言語) 式を作成します。
次の図は、回数が多すぎる場合にアクティビティが実行されないようにするための条件の例を示しています。

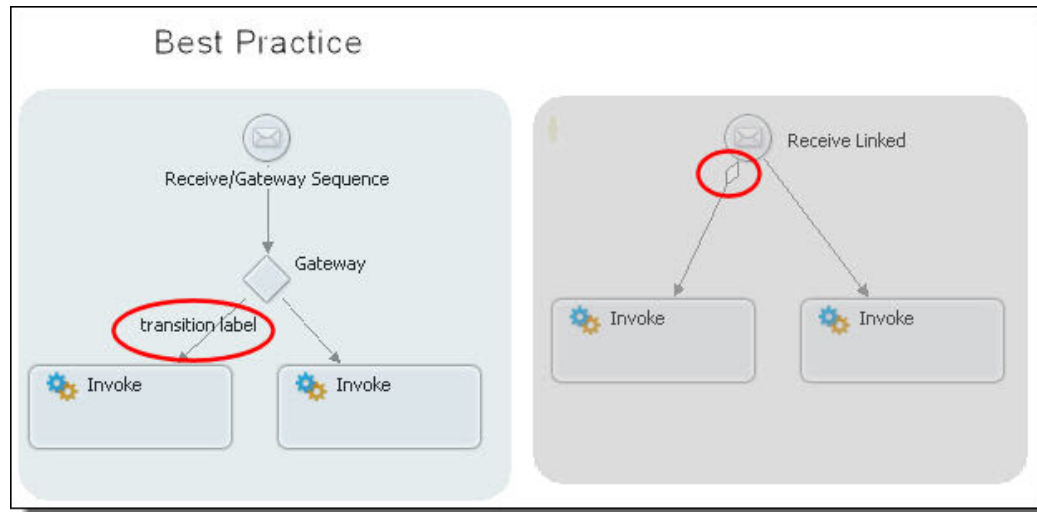


この例では、各パラメータは、WSDL メッセージからのデータを保持する BPEL プロセス変数を参照しています。詳細については、[第 13 章, 「変数」 \(ページ 241\)](#)を参照してください。

アクティビティのリンク

任意の2つのアクティビティをリンクできます。ただし、ゲートウェイを使用して、リンクの目的を明確に示す必要があります。次の図の左側の構成では、リンクの遷移条件を示すラベルを、ゲートウェイを使用して追加した場合が示されています。

右側には、2つのアクティビティにリンクされた受信があります。小さなゲートウェイのひし形によって遷移条件が示されていることに注意してください。ただし、ここにラベルを追加することもできます。



[リンクの追加] ダイアログの使用

リンクのターゲットアクティビティを選択し、必要に応じて遷移条件を追加します。

2つのアクティビティ間にリンクを追加するには、次のいずれかを実行します。

- ソースアクティビティを選択してから、ターゲットアクティビティを選択します。**[リンク]** ツールバーボタンを選択して、リンクを作成します。複数のアクティビティを選択することも可能で、リンクは選択した順番で作成されます。詳細については「[アクティビティ間へのリンクの追加](#)」(ページ 266)を参照してください。
- パレットの**[リンクツール]**を使用します。詳細については、「[アクティビティ間へのリンクの追加](#)」(ページ 266)のヒントを参照してください。
- **[リンクの追加]** ダイアログを使用します

[リンクの追加] ダイアログを使用する手順

1. Process Editor キャンバスでアクティビティを選択します。例えば、Receive またはシーケンスを選択します。
2. 右マウスメニューから、**[リンク]** (または **[追加] > [リンク]**) を選択します。
3. 次のように、**[リンクの追加]** ダイアログに入力します。
 - a. **[名前]** フィールドで、デフォルトの名前を使用するか、よりわかりやすいリンク名を入力します。
 - b. **[ターゲット]** フィールドで、**[ダイアログ (...)]** ボタンを選択します。
 - c. **[リンクターゲットノードの選択]** ダイアログで、リンクのターゲットアクティビティを選択し、**[OK]** をクリックします。
 - d. 必要に応じて、リンクをダブルクリックするか、**[プロパティ]** ビューの **[ダイアログ]** ボタンを選択して遷移ビルダを開き、リンクの **[遷移条件]** を追加します。詳細については「[遷移条件付きのリンクの追加](#)」(ページ 267)を参照してください。
 - e. **[リンクの追加]** ダイアログで **[OK]** を選択します。

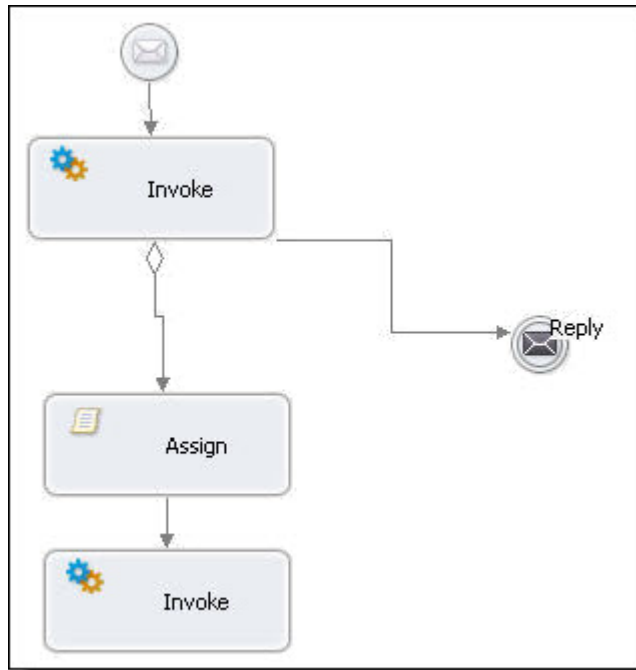
リンクの例

次の図は、リンクを使用する方法を示しています。

また、「[リンクを使用した設計と構造化されたアクティビティ](#)」(ページ 270)も考慮してください。

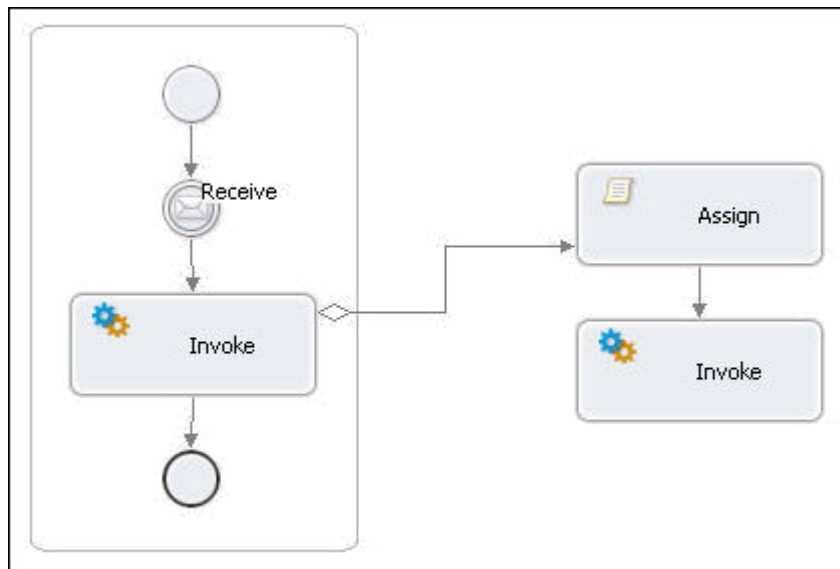
例 1

最初の例では、リンクが true と評価された場合に 2 番目のシーケンスが実行され、それ以外の場合は応答が実行されます。



例 2

次の例では、スコープがシーケンスと並行して実行されています。一方のシーケンスのアクティビティには、もう一方のシーケンスのアクティビティへのリンクがあります。



リンクの実行ルール

アクティビティが完了すると、出力リンクの状態が評価されます。条件が定義されておらず、アクティビティが正常に完了した場合、すべての条件が true と評価され、次のアクティビティが開始されます。ただし、アクティビティには、実行を制限するための結合条件をさらに設定することもできます。詳細については、[「リンクと結合条件」 \(ページ 271\)](#)を参照してください。

アクティビティにフォールトが発生した場合、またはアクティビティが実行されなかった場合、すべての条件は false と評価されます。

アクティビティは、複数の入力リンクを持つことがあります。アクティビティは、入力リンクの少なくとも 1 つが true と評価されるまで実行を待機します。

すべての入力リンクが false の場合、*suppressJoinFailure* プロパティが true とならない限り、アクティビティは実行されません。

以下の場合を除いて、リンクは構造化されたアクティビティの境界を越えることができます。

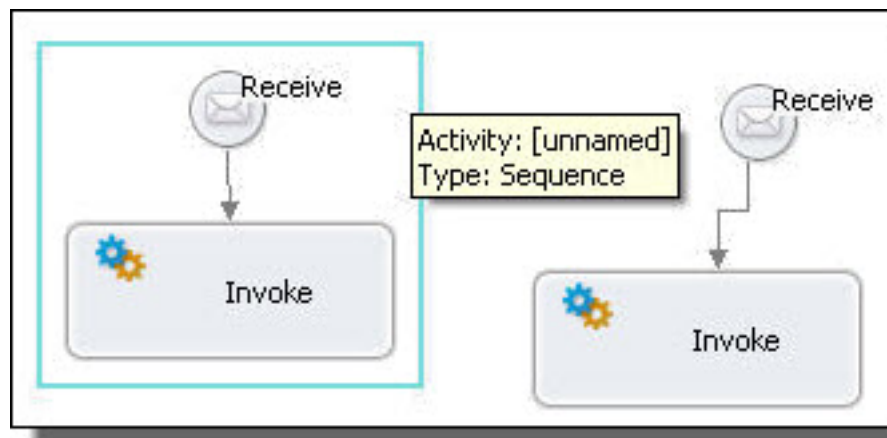
- While アクティビティ
- 分離スコープ
- イベントハンドラ
- 補償ハンドラ

フォールトハンドラの境界を越えるリンクは、フォールトハンドラ内にソースを持ち、フォールトハンドラスコープを囲むスコープ内にターゲットを持つ必要があります。

WS-BPEL では、完了したアクティビティにリンクしても、リンクによって制御サイクルが作成されることはありません。ただし、Process Developer では、[「リンクに対する Process Developer 拡張機能」 \(ページ 265\)](#)に記載されているように、この動作は許可されています。

リンクを使用した設計と構造化されたアクティビティ

リンクまたはシーケンスを使用してアクティビティに同じ実行順序を設定することはよくあります。次の例は、同一の 2 つの実行の設計を示しています。



多くの場合、リンクのみを使用して実行パスを設計できます。例としては、コンテナ内のアクティビティを別のコンテナ内の別のアクティビティにリンクする場合などが挙げられます。

リンクと結合条件

結合条件プロパティの式を記述することで、リンクを対象とするアクティビティの開始を制御できます。詳細については、[「入力リンクに対する結合条件の作成」](#) (ページ 292)を参照してください。

プロパティのリンク

Process Editor キャンバスまたは「アウトライン」ビューでリンクを選択し、「プロパティ」ビューを表示します。

次の表に、リンクに必須のプロパティとオプションのプロパティを示します。

必須のプロパティ	オプションのプロパティ
リンク名	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419)を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126)を参照してください。

第 16 章

データ操作

次のトピックでは、リテラル式、XPath、および XQuery を使用して BPEL で XML データを操作する方法について説明します。

- [「BPEL でのデータ操作の概要」 \(ページ 272\)](#)
- [「式構築のための XPath または XQuery の選択」 \(ページ 273\)](#)
- [「式ビルダの使用」 \(ページ 275\)](#)
- [「条件カウンタおよびその他の値に想定される式」 \(ページ 289\)](#)
- [「クエリビルダの使用」 \(ページ 290\)](#)
- [「入力リンクに対する結合条件の作成」 \(ページ 292\)](#)
- [「期限と期間の式」 \(ページ 293\)](#)

BPEL でのデータ操作の概要

BPEL プロセスでデータを操作するにはさまざまな方法がありますが、データコピータスクを簡単または効率的に実行する方法については、常に明確であるとは限りません。

また、データ構造およびデータ構造から作成する式の選択に応じて、使用する式言語を式ごとに選択することもできます。詳細については、[「式構築のための XPath または XQuery の選択」 \(ページ 273\)](#) を参照してください。

次の表を使用し、特定の BPEL 手法を利用してデータを操作する一部の例を検討してください。

割り当てにおけるコピー操作の使用	「コピー操作のヒント」 (ページ 177) 「コピー操作のクエリと式の例」 (ページ 177) 「コピー操作のリテラルコンテンツの例」 (ページ 178) 「コピー操作の動的エンドポイントの参照例」 (ページ 181) 「ソース要素名属性を維持する要素間のコピー操作」 (ページ 182) 「[見つからない開始データを無視] の属性を使用したコピー操作」 (ページ 183)
	「入力変数」 (ページ 197) 「出力変数」 (ページ 199) 「開始パートから変数」 (ページ 197) 「変数からパート」 (ページ 197)
変数の初期化	「変数の初期化」 (ページ 246)

XPath の例	「XPath 式の例」 (ページ 275)
XQuery の例	「XQuery 式の例」 (ページ 273)
添付ファイルのカスタム関数	「添付ファイルのカスタム関数の例」 (ページ 263)
時間と日付の形式	「期限と期間の式」 (ページ 293)

式構築のための XPath または XQuery の選択

デフォルトの言語は XQuery で、これは XPath のスーパーセットです。Process Developer プロジェクトには、カスタム関数 XQuery モジュールを作成するための XQuery エディタを含む XQuery の性質が含まれています。

次の表に、BPEL データマッピングで使用される XPath と XQuery の簡単な比較を示します。

XPath	XQuery
移植性が重要となる場合に推奨されます。言語は WS-BPEL 2.0 でサポートされています	BPEL に対する Process Developer 拡張機能です
複数のパートから構成されるメッセージ変数に、より適している可能性があります	単一パートの変数から大きなドキュメントを簡単に作成できます
一度に 1 つのドキュメントノードの式評価のみを行うことができます	XML ドキュメント全体からのデータの操作が可能です
	各種の日付関数による日付処理に、より適しています
ロケーションパス式が文字列に解決されます	クエリによって XML 結果が生成されます
	SQL に似たクエリ言語です。結合を実行する場合、「FLWOR 式」(FOR、LET、WHERE、ORDER BY、および RETURN) を使用して要素の繰り返しを簡単に反復することができます

関連事項:

- [「XQuery 式の例」 \(ページ 273\)](#)
- [「XQuery 関数の記述」 \(ページ 299\)](#)
- [「XPath 式の例」 \(ページ 275\)](#)

XQuery 式の例

XQuery 式言語を選択すると、すべての XPath 式の作成に加えて、XQuery でサポートされている多くの関数を使用できるようになります。追加の XQuery 関数カテゴリには、Date、QName、Misc、および Constructor が含まれます。

Invoke	Assignment Type: XQuery
Input	
Output	Part: emailPart Regenerate Builder...
All	<pre> <aem:emailMessage xmlns:aem="http://schemas.active-endpoints.com/email/2007/01/email.xsd"> <aem:from>{ \$fill_in_here }</aem:from> <aem:replyTo>{ \$fill_in_here }</aem:replyTo> { For \$to in \$fill_in_here return } <aem:to>{ \$fill_in_here }</aem:to> { For \$cc in \$fill_in_here return } <aem:cc>{ \$fill_in_here }</aem:cc> { For \$bcc in \$fill_in_here return } } </pre>

次に、以下の例に示すように、ドキュメントの式を作成します。

Part: emailPart	Regenerate Builder...
<pre> <aem:emailMessage xmlns:aem="http://schemas.active-endpoints.com/email/2007/01/email.xsd"> <aem:from>{\$runtimeParameters/param:emailAddress/text()}</aem:from> <aem:replyTo>{\$runtimeParameters/param:emailAddress/text()}</aem:replyTo> <aem:to>{\$claimRequest/claim:contact/claim:email/text()}</aem:to> <aem:subject>VOSton Mutual Auto Claim #: (Ref# { \$refNumber })</aem:subject> <aem:body mimeType="text/html">Dear { \$claimRequest/claim:contact/claim:name/text() }, <html> <body> <p> Claim #: { \$refNumber } has been { \$ClaimStatus}. </p> <p>Please check the status of your claim at http://{ \$runtimeParameters/param:server/text() }/AVInsuranceWebApp/claimStatus.jsp, and call us at the number below if you have any questions.</p> <p> Thank You for insuring your automobile with VOSton Mutual.
 Regards, </p> <p> VOSton Mutual
 Web: http://www.vostonmutual.com
 Tel: +1.866.555.1212
 Email: {\$runtimeParameters/param:emailAddress/text()}</p> </body> </html> </aem:body> </aem:emailMessage> </pre>	

上記の例から続く以下の部分のような、要素のコンテンツを生成するために使用される構文に注意してください。

```
<aem:replyTo>{$runtimeParameters/param:emailAddress/text()}</aem:replyTo>
```

パスの最後に text()関数を追加する必要があります。XQuery 式の結果が要素になる場合は、必ず text()を追加して要素のコンテンツを取得する必要があります。

/text()のないパスを使用した場合は、次のようになります。

```

<aem:replyTo>
  <emailAddress>reply@example.org</emailAddress>
</aem:replyTo>

```

想定される結果の代わりに:

```
<aem:replyTo>reply@example.org</aem:replyTo>
```

XML 要素ではなく単純型の値のみが式に含まれる場合、式はコンテンツになります。例えば、以下の式は文字列になります。

Please use the Claim #: {\$refNumber}

XPath 式の例

以下は、メッセージ、要素、および複合スキーマタイプ変数の式の例です。

- **要素タイプ変数:**
例: `$AStatus/e:statusDescription`
`AStatus` 要素の `statusDescription` という子ノードを選択します
- **メッセージタイプ変数:**
例: `$StatusVariable.StatusPart2/e:statusDescription`
`StatusVariable` の `StatusPart2` 部分の `statusDescription` という子ノードを選択します
- **スキーマ複合型変数:**
例: `$Results/e:AuctionResult[2]/@AuctionID`
2 番目の `AuctionResult` という子ノードの `AuctionID` 属性を選択します
- **式ステートメント** `itemsShipped = itemsShipped + itemsCount:`
`$itemsShipped + $shipnotice.props/itemsCount`
- 変数とパーツを次のような文字列に変換します。
`string($input.parameters)`
- **ローカル名のみの子ノードの参照**
これは、次のような `xsd:any` 要素がスキーマに含まれている場合に役立ちます。

```
<xsd:any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
```

式ビルダでは、変数ツリービューに構造は表示されません。この要素を使用するには、次の例のように名前空間プレフィックスを無視します。

```
$creditInformation/*[local-name()=
"previousLoanApplicationAttempts"][1]
/*[local-name()="applicationDate"]
```

以前のローン申請に関する情報が `creditInformation` メッセージの追加コンテンツとして提供されている場合は、前回のローン申請の最初の日付を選択します。

式ビルダの使用

[変数]、[関数]、および [演算子] 選択リストから適切なものを選択して、[式/条件] テキストボックスに式を作成します。

さまざまな関数タイプから適切なものを選択して、コピー操作、リンク遷移、およびその他の BPEL 構成の式を作成できます。式の作成の詳細については、以下の手順を参照してください。

[「式構築のための XPath または XQuery の選択」 \(ページ 273\)](#) も参照してください。

例

- [「条件カウンタおよびその他の値に想定される式」 \(ページ 289\)](#)
- [「XPath 式の例」 \(ページ 275\)](#)
- [「XQuery 式の例」 \(ページ 273\)](#)

関数

- [「ブール関数」 \(ページ 281\)](#)
- [「BPEL 関数」 \(ページ 277\)](#)
- [「ノードセット関数」 \(ページ 286\)](#)
- [「数値関数」 \(ページ 287\)](#)
- [「Process Developer のカスタム関数全般」 \(ページ 279\)](#)
- [「文字列関数」 \(ページ 288\)](#)

カスタマイズ

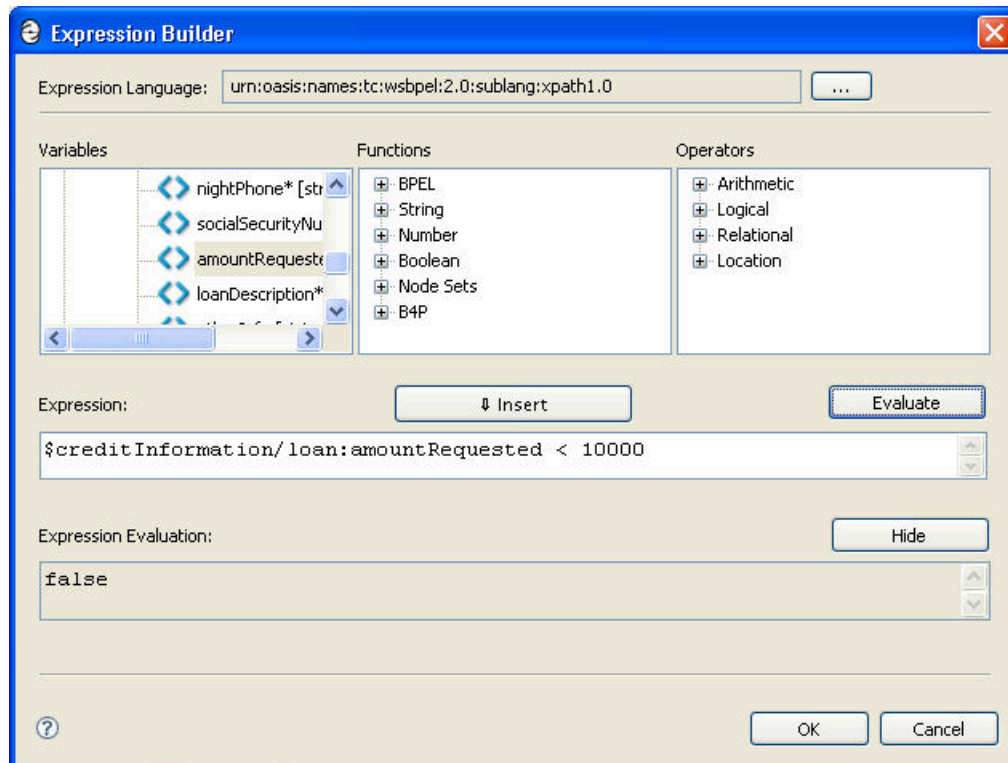
- [「カスタム関数の概要」 \(ページ 295\)](#)
- [「XPath 関数の記述」 \(ページ 299\)](#)
- [「式言語」 \(ページ 133\)](#)

Assign アクティビティのコピー操作、およびリンクやその他の設定の場合、文字列、数値、またはブール値になる XPath (または別の選択された言語) 式を作成できます。

詳細については、<http://www.w3.org/TR/xpath> で「XPath 1.0 仕様」を参照してください。

式を作成する手順

1. Pick アクティビティの割り当て、待機、アラームブランチなどの式を使用するアクティビティを選択します。
2. アクティビティの [プロパティ] ビューで、適切なプロパティを選択します。例えば、Assign アクティビティで [コピー操作] を選択し、[コピー先/コピー元としての式] を選択して、次のダイアログを表示します。



3. 次のヒントに従って、3つのボックスから変数、関数、および演算子を選択して式を作成します。
 - ショートカットを使用すると、式をよりすばやく作成できます。[「コンテンツアシストの使用」 \(ページ 277\)](#)を参照してください。
 - パーツとプロパティを表示および使用するには、変数を展開します。変数の詳細については、[第 13 章, 「変数」 \(ページ 241\)](#)を参照してください。
 - 変数、パーツ、関数、または演算子を選択してから、**[挿入]** を選択します
 - 変数添付ファイルの操作については、[「添付ファイルを操作するためのカスタム関数」 \(ページ 260\)](#)を参照してください。
 - [第 17 章, 「POJO および XQuery カスタム関数」 \(ページ 295\)](#)に記載されているように、独自のカスタム関数を追加します。
4. 変数パーツのサンプル値を置き換え、式を簡略化してその値を表示するには、**[評価]** を選択します。

コンテンツアシストの使用

[変数]、[関数]、および [演算子] 選択リストから適切なものを選択して、[式/条件] テキストボックスに式を作成します。

コンテンツアシストは、式の作成に使用可能な変数と関数から選択を行うためのショートカットです。変数を展開してパーツにドリルダウンし、式のテキストボックスに挿入する代わりに、式のテキストボックス内でパーツを選択できます。

コンテンツアシストを使用する手順

1. ビルダ (式、クエリ、またはその他) で、**[挿入]** ボタンの下にある式のテキストボックスにカーソルを合わせます。
2. 関数を選択するには、Ctrl + Space を選択します。選択可能な機能のリストが表示されます。
3. 変数、パーツ、または要素を選択するには、次のように Ctrl + Space を選択します。
 - a. カーソルが\$の後にある場合は、使用可能な変数のリストが表示されます (該当する場合)。
 - b. カーソルがピリオド (.) の後にあり、コンテキストが XPath 式内にある場合は、有効なパーツ名のリストが表示されます (該当する場合)。
 - c. カーソルがスラッシュ (/) の後にあり、コンテキストが XPath 式内にある場合は、有効な要素名のリストが表示されます (該当する場合)。

BPEL 関数

このセクションでは、次の関数について説明します。

- [「bpel:doXslTransform\(style-sheet-uri node-set\)」 \(ページ 277\)](#)
- [「bpel:getVariableProperty\(VariableName propertyName\)」 \(ページ 279\)](#)

bpel:doXslTransform(style-sheet-uri node-set)

指定したスタイルシートを使用して、単一要素のノードセットの XSL 変換の結果を返します。

パラメータ:

- style-sheet-uri: XSL ファイルの URI。
- node-set: 変換のソースドキュメントを提供する XPath ノードセット。

構文例:

```
bpel:doXslTransform("project:/myStylesheets/A2B.xsl", $A)
```

オプションのパラメータ（指定した場合はペアで表示されます）

- string: XSLT パラメータの修飾名を提供する XPath 文字列パラメータ。
- object: 名前付き XSLT パラメータの値を提供する XPath オブジェクトパラメータ。

例

次の例は、複合的なドキュメント変換と反復的なドキュメント構築を示しています。

例 1. 複合的なドキュメント変換。

WS-BPEL プロセスの一般的なパターンには、特定のサービスから XML ドキュメントを受信して、それを別のスキーマに変換して新しい要求メッセージを形成し、新しい要求を別のサービスに送信することが含まれます。このようなドキュメント変換は、次の例に示すように、`bpel:doXsltTransform` 関数を介して XSLT を使用して実行することができます。

```
<variables>
  <variable name="A" element="foo:AElement" />
  <variable name="B" element="bar:BElement" />
</variables>
...
<sequence>
  <invoke ... inputVariable="..." outputVariable="A" />
  <assign>
    <from>
      bpel:doXsltTransform("urn:stylesheets:A2B.xsl", $A)
    </from>
    <to variable="B" />
  </assign>
  <invoke ... inputVariable="B" ... />
</sequence>
```

シーケンスでは、サービスが呼び出され、結果（`foo:AElement`）が変数 A にコピーされます。Assign アクティビティは、変数 A の内容を `bar:BElement` に変換し、結果を変数 B にコピーします。変数 B は別のサービスを呼び出すために使用されます。スタイルシート `A2B.xsl` には、スキーマ `foo:AElement` のドキュメントをスキーマ `bar:BElement` に変換するための XSL ルールが含まれています。

例 2. 反復的なドキュメント構築。

次の例に示すように、サービスを繰り返し呼び出し、結果を変数に累積することによってドキュメントが作成されたとします。

```
<variables>
  <variable name="P0" element="foo:P0Element" />
  <variable name="OutVar" element="foo:ItemElement" />
</variables>
<!-- ... P0 is initialized ... -->
<!-- Iteratively add more items to P0 until complete -->
<while>
  <condition>...</condition>
  <sequence>
    <!-- Fetch next chunk into OutVar -->
    <invoke ... inputVariable="..." outputVariable="OutVar"/>
    <assign>
      <copy>
        <from>
          <expression>
            bpel:doXsltTransform
              ( "urn:stylesheets:AddToP0.xsl",
                $P0, "NewItem", $OutVar)
          </expression>
        </from>
        <to variable="P0" />
      </copy>
    </assign>
  </sequence>
</while>
```

doXsltTransform 呼び出しで指定したオプションのパラメータによって、NewItem という名前の XSLT パラメータが、プロセス変数 OutVar の値で設定されるように指定します。XSLT スタイルシートがこの値にアクセスできるようにするために、この値には、上記の関数呼び出しの 3 番目のパラメータで指定した名前と一致する名前のグローバル（トップレベル）パラメータが含まれています。

```
<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ...>
  <!-- NewItem variable set by WS-BPEL process;
        defaults to empty item -->
  <xsl:param name="NewItem">
    <foo:itemElement />
  </xsl:param>
  ...
</xsl:transform>
```

スタイルシートには、グローバルパラメータ NewItem の値（プロセスインスタンスからの OutVar の値）を PO 変数内の既存の項目リストに追加するテンプレートが含まれています。

```
<!-- line 1 --> <xsl:template match="foo:itemElement">
<!-- line 2 --> <xsl:copy-of select="." />
<!-- line 3 -->   <xsl:if test="position()=last()">
<!-- line 4 -->     <xsl:copy-of select="$NewItem" />
<!-- line 5 -->   </xsl:if>
<!-- line 6 --> </xsl:template>
```

このテンプレートは、ソースドキュメント内の既存のすべての項目をコピーし（1 行目と 2 行目）、XSLT パラメータ NewItem の内容を項目のリストに追加します（3 行目と 4 行目）。現在のノードが項目リストの最後にあるかどうかをテストし（3 行目）、XSLT パラメータ NewItem から result-tree フラグメントをコピーして最後の項目を追跡します（4 行目）。

PO が次の値である場合:

```
<foo:poElement>
  <foo:itemElement>item 1</foo:itemElement>
</foo:poElement>
```

While ループの反復の開始時に Invoke アクティビティが<foo:itemElement>item 2</foo:itemElement>の値を返すと、from 式の評価は次のような値になります。

```
<foo:poElement>
  <foo:itemElement>item 1</foo:itemElement>
  <foo:itemElement>item 2</foo:itemElement>
</foo:poElement>
```

コピー操作が完了すると、その値が PO 変数の新しい値になります。

プロセスサーバーへの XSL スタイルシートの追加

doXsltTransform 関数で参照するスタイルシートは、Process Developer によって自動的に認識され、デプロイメント用に BPR アーカイブに追加されます。

bpel:getVariableProperty(VariableName propertyName)

BPEL プロセスで選択された変数からプロパティデータを返します。

パラメータ:

- VariableName: データを取得する変数の名前
- propertyName: データを含む変数内のプロパティの名前

Process Developer のカスタム関数全般

次のトピックでは、Process Developer のカスタム関数について説明します。

- [「abx:base64Encode\(stringValue charSet\)」](#) (ページ 280)

- [「abx:elementToXMLString\(element\)」 \(ページ 280\)](#)
- [「abx:getAllHTTPHeaders\(partnerLinkName\)」 \(ページ 280\)](#)
- [「abx:getHTTPHeader\(partnerLinkName headerName\)」 \(ページ 280\)](#)
- [「abx:getInboundSOAPHeader\(partnerLinkName\)」 \(ページ 280\)](#)
- [「abx:getProcessId\(\)」 \(ページ 281\)](#)
- [「abx:getProcessInitiator\(\)」 \(ページ 281\)](#)
- [「abx:getProcessName\(\)」 \(ページ 281\)](#)
- [「abx:isVariableInitialized\(variable\)」 \(ページ 281\)](#)
- [「abx:resolveURN\(string\)」 \(ページ 281\)](#)
- [「abx:setProcessTitle\(titleString\)」 \(ページ 281\)](#)
- [「abx:xmlStringToElement\(string\)」 \(ページ 281\)](#)

abx:base64Encode(stringValue charSet)

Base64 でエンコードされた文字列を返します。

パラメータ:

- stringValue: エンコードする文字列
- charSet: エンコードに使用する、オプションの名前付き文字セット。デフォルトは UTF-8 です。

abx:elementToXMLString(element)

指定した要素に基づいて XML 文字列を返します。

abx:getAllHTTPHeaders(partnerLinkName)

ビジネスプロセスにおいて PartnerLink で受信された最後のメッセージから HTTP ヘッダー値のマップを返します。ペイロードが SOAP、XML、JSON のいずれかであっても、あらゆるタイプの受信 HTTP 要求に適用されます。

パラメータ:

- partnerLinkName: データを取得する PartnerLink の名前

abx:getHTTPHeader(partnerLinkName headerName)

ビジネスプロセスにおいて PartnerLink で受信された最後のメッセージから HTTP ヘッダーの値を返します。ペイロードが SOAP、XML、JSON のいずれかであっても、あらゆるタイプの受信 HTTP 要求に適用されます。

パラメータ:

- partnerLinkName: データを取得する PartnerLink の名前。
- headerName: ヘッダーの名前

abx:getInboundSOAPHeader(partnerLinkName)

ビジネスプロセスにおいて PartnerLink で受信された最後のメッセージから SOAP メッセージのヘッダーブロックを返します。

パラメータ:

- partnerLinkName: データを取得するパートナーリンクの名前

abx:getProcessId()

プロセスサーバーで現在実行中のプロセスまたは完了したプロセスのプロセス ID を返します。

abx:getProcessInitiator()

createInstance メッセージに添付されているプリンシパルを返します。メッセージに関連付けられた資格情報またはプリンシパルがない場合は、匿名を返します。

abx:getProcessName()

現在実行中のプロセスのプロセス名（ローカルの部分のみ）を返します。

abx:isVariableInitialized(variable)

変数が初期化されている場合は true を返します。

abx:resolveURN(string)

サーバーの URN/URL マッピング機能を使用して、URN の解決された場所を返します。単一の文字列パラメータを入力として使用します。例: abx:resolveURN('urn:localhost:AssessRisk')

abx:setProcessTitle(titleString)

プロセスコンソールの「アクティブなプロセス」ページで、アクティブなプロセスの表示タイトルを設定します。プロセス名が置き換えられることはありません。

abx:xmlStringToElement(string)

指定した XML 文字列に基づいて要素を返します。

添付ファイル関数

これらの関数については、「[添付ファイルを操作するためのカスタム関数](#)」 (ページ 260)に記載されています。

ブール関数

次のブール関数を使用できます。

- [「boolean\(object\)」](#) (ページ 282)
- [「not\(boolean\)」](#) (ページ 282)
- [「true\(\)」](#) (ページ 282)
- [「false\(\)」](#) (ページ 282)
- [「lang\(string\)」](#) (ページ 282)

boolean(object)

ブール関数によって、次のように引数がブールに変換されます。

- 正または負のゼロではない場合あるいは NaN ではない場合にのみ、数値は true になります。
- 空でない場合にのみ、ノードセットは true になります。
- 長さがゼロでない場合にのみ、文字列は true になります

パラメータ:

- object: 4 つの基本タイプ以外の型のオブジェクトは、その型に依存する方法でブール値に変換されます。

false()

false を返します。

lang(string)

lang 関数は、xml:lang 属性で指定したコンテキストノードの言語が引数文字列で指定した言語と同じであるか、またはサブ言語であるかに応じて、true あるいは false を返します。コンテキストノードの言語は、コンテキストノードの xml:lang 属性の値によって決定されます。または、コンテキストノードに xml:lang 属性がない場合は、xml:lang 属性を持つコンテキストノードの最も近い祖先の xml:lang 属性の値によって決定されます。このような属性がない場合、lang は false を返します。このような属性があり、大文字と小文字を区別を無視した引数とこの属性値が等しい場合、または属性値のサフィックスおよび大文字と小文字を区別を無視した引数と等しくなるような-で始まるサフィックスがこの属性値にある場合、lang は true を返します。

not(boolean)

引数が false の場合は true を返し、それ以外の場合は false を返します。

true()

true を返します。

カタログ関数

これらのカスタム関数によって、プロセスサーバーのカタログから任意のリソースをロードし、カタログ内のリソースを更新することができます。ロードしたリソースは、XML ドキュメントとして式言語に返されます。例えば、アラームやエンドポイント参照などの構成情報をプロセス定義から抽出し、それをカタログに含めることで、プロセス定義間で変更および共有されるようにすることができます。カタログを使用すると、そのデプロイメント環境に基づいて、プロセスに異なる構成情報を持たせることもできます。例えば、ある開発環境には特定のタイプの構成情報を含め、実稼働環境には別のタイプの構成情報を含めるといったことが可能です。構成情報は XML 形式です。リソース XML ファイルをプロジェクトにインポートし、シミュレーションや BPR アーカイブへの自動エクスポートに使用できるようにする必要があります。詳細については、[「WSDL、スキーマ、およびその他のリソースのインポート」 \(ページ 122\)](#)および [「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#)を参照してください。

使用可能なカタログ関数は次のとおりです。

- [「abx:getCatalogResource\(location\)」 \(ページ 283\)](#)
- [「abx:putCatalogResource\(location typeURI document\)」 \(ページ 283\)](#)

abx:getCatalogResource(location)

リソースコンテンツを含むドキュメントを返します。location（場所）は、カタログの場所の URL を含む文字列です。以下の例では、putCatalogResource() を使用して保存された要素またはドキュメントが場所ごとに返されます。例えば、この関数をコピー操作で使用すると、要素を変数に戻すことができます。

要求ヘッダーで application/json が使用されている場合、またはリソースが .tojson で終わる場合（例えば、/avccatalog/project:/path/mydoc.xml.tojson など）、エンドポイントが /avccatalog であれば、要求をプロキシするプロセスを作成する必要はありません。また、要求ヘッダーに application/json がある場合、エンドポイントが /catalog であれば、プロキシを作成する必要はありません。

例:

```
abx:getCatalogResource("project:/myProject/resources/resource.xml")
```

コピー操作で、この関数から（変数に）コピーを行う例を次に示します。式の言語は XQuery であることに注意してください。

```
<ns1:catalogEntry xmlns:ns1='http://activevos/samples/catalog/functions'>
  <ns1:catalogLocation>
    <ns1:location>{data(
      $requestOperation.catalogOp/
      ns1:catalogLocation/ns1:location )
    }</ns1:location>
    <ns1:typeURI>{data( $requestOperation.catalogOp/
      ns1:catalogLocation/ns1:typeURI )}</ns1:typeURI>
  </ns1:catalogLocation>
  <ns1:content>{abx:getCatalogResource( data(
    $requestOperation.catalogOp/ns1:
    catalogLocation/ns1:location))}</ns1:content>
</ns1:catalogEntry>
```

abx:putCatalogResource(location typeURI document)

渡された場所でカタログリソースを更新または追加します。この関数はブール値を返すため、コピー操作のコピー元側で使用するにより、ブール型の変数にブール値をコピーできます。以下の例では、ドキュメントまたは要素を、識別子 project:/myProject/resources/resource.xml を持つ、タイプ http://www.example.org/2009/03/として定義されたカタログの変数\$V1 に配置しています。タイプが WSDL の場合は、WSDL 名前空間を使用できます。タイプを指定しない場合、\$V1 の要素の名前空間が自動的に使用されます。

例:

```
abx:putCatalogResource( "project:/myProject/resources/resource.xml" , "http://www.example.org/2009/03/", $V1)
```

コピー操作で、この関数から（ブール変数に）コピーする例は次のとおりです。

コピー元:

```
abx:putCatalogResource( string($requestOperation.catalogOp/ns1:catalogLocation/ns1:location/text()), $requestOperation.catalogOp/ns1:content/*)
```

コピー先

funcResponse

パラメータ:

- location: カタログの場所の URL を含む文字列。場所はヒントとなります
- typeURI: リソースの名前空間タイプ（渡されない場合に使用される要素名前空間）を含むオプションの文字列。
- document: カタログリソースに配置される xml コンテンツ（ドキュメントまたは要素）。

この関数は、ブール値 true または false の値を返します。

フォールト関数

フォールトカスタム関数は、パートナーサービスでフォールトメッセージの宣言によってインタフェースが適切に実装されていない場合に役立ちます。一部の WSDL 操作ではフォールトメッセージが宣言されないため、サービスでフォールトがスローされても、catch または catchAll ハンドラによる通常の方法ではキャッチできません。宣言されていないフォールトは、これらの関数を使用してキャッチします。catch または catchAll フォールトハンドラによってトリガされるアクティビティ内のプロセス変数内にある関数を使用します。例: \$soapFault/faultstring/text() という XPath 式を介して。

以下のフォールトについては、下記の戻り値の例を参照してください。

使用可能なフォールト関数は次のとおりです。

- [「getFaultCode\(\)」 \(ページ 284\)](#)
- [「getFaultDetail\(\)」 \(ページ 284\)](#)
- [「getFaultString\(\)」 \(ページ 284\)](#)
- [「getSOAPFault\(\)」 \(ページ 284\)](#)

getFaultCode()

フォールトコード QName がある場合、フォールトハンドラをトリガしたイベントからこのコードを返します。

getFaultDetail()

フォールト詳細要素がある場合、フォールトハンドラをトリガしたイベントからこの要素を返します。

getFaultString()

フォールトメッセージ文字列がある場合、フォールトハンドラをトリガしたイベントからこの文字列を返します。

返されるフォールトメッセージの例を以下に示します。

```
<SOAP-ENV:Fault xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>the_fault_string</faultstring>
  <detail/>
</SOAP-ENV:Fault>
```

getSOAPFault()

SOAP フォールト要素がある場合、フォールトハンドラをトリガしたイベントからこの要素を返します。

I18N 関数

getLocalizedMessage(key, locationHint, locale, [arg1, argN])

指定されたロケールの外部化された文字列のキーの値を返します。このキーは、プロセスセントラルプロセス要求フォーム用に作成されたメッセージプロパティファイルから返されたものです。この関数を REST ベースのプロセスで使用して、ローカライズされたデータをプロセス要求フォームの出力メッセージに送り返します。REST プロセスでは、ブラウザロケールを使用できるため、返されたデータをローカライズすることができます。

パラメータ:

- key: 必須の文字列 - Process Developer で作成されたメッセージプロパティファイルのキー名。キーの値は関数によって返されます。

- `locationHint`: 必須の文字列 - デフォルトのプロパティファイルに対するカタログの場所の URL。場所はヒントとなります。
- `locale`: 必須の文字列 - `en_US`、`fr_CA`、`fr` などのメッセージのロケール。デフォルトのロケールに対する空の文字列。ロケールはブラウザのロケールに基づいています。
- `arg1`: オプションの文字列またはオブジェクト - キー値文字列内のテキストを置き換える引数
- `argN`: オプションの文字列またはオブジェクト - `arg1` と同じように、引数の数は、キー値に含まれるプレースホルダの数に応じて異なります。

例:

```
abx:getLocalizedMessage('loan.description','project:/myLoanApprovalProject/deploy/messages.properties','fr',
$loanAmount)
```

`arg1`、`argN` オブジェクトの構文は Java messageformat に基づいていることに注意してください。

JSON 関数

これらの JavaScript Object Notation (JSON) 関数を REST エンドポイントと組み合わせて使用します (この場合、要求や応答は JSON となります)。詳細については、[「REST ベースのサービスの使用」 \(ページ 329\)](#) を参照してください。

使用可能な JSON 関数は次のとおりです。

- [「jsonToXml\(jsonString\)」 \(ページ 285\)](#)
- [「xmlToJson\(element\)」 \(ページ 286\)](#)

jsonToXml(jsonString)

JSON 文字列を XML 要素に変換します。この文字列は配列を表します。

次の例は、4 つの異なる配列表現を XML に変換する方法を示しています。

- ルート要素と値は JSON オブジェクトです。
JSON:

```
{ "foo": { "name": "foo", "desc": "foo desc" } }
```

XML:

```
<foo desc="foo desc" name="foo"/>
```
- ルート要素と値は配列です。配列の値を表す要素のシーケンスを含むラッパーオブジェクト要素が作成されます。
JSON:

```
{ "foo": [ { "name": "foo", "desc": "foo desc" }, { "name": "foo2", "desc": "foo2 desc" } ] }
```

XML:

```
<jjson:object xmlns:jjson="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd">
  <foo desc="foo desc" name="foo"/>
  <foo desc="foo2 desc" name="foo2"/>
</jjson:object>
```
- ルートオブジェクトは存在しますが、ルート要素はありません。JSON オブジェクトはオブジェクト要素にラップされます。
JSON:

```
{ "name": "foo", "desc": "foo desc" }
```

XML:

```
<jjson:object xmlns:jjson="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd"
  desc="foo desc"
  name="foo"/>
```

- ルート配列は存在しますが、ルート要素はありません。配列は配列要素にラップされ、各オブジェクトはオブジェクト要素のシーケンスになります。

JSON:

```
[{"name":"foo","desc":"foo desc"}, {"name":"foo2","desc":"foo2 desc"}]
```

XML:

```
<json:array xmlns:json="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd">
  <json:object desc="foo desc" name="foo" />
  <json:object desc="foo2 desc" name="foo2" />
</json:array>
```

xmlToJson(element)

要素を JSON 文字列として返します。この要素は、XML ドキュメントまたは変換するための要素です。

ノードセット関数

次のノードセット関数を使用できます。

- [「count\(node-set\)」 \(ページ 286\)](#)
- [「id\(object\)」 \(ページ 286\)](#)
- [「last\(\)」 \(ページ 286\)](#)
- [「local-name\(node-set\)」 \(ページ 286\)](#)
- [「name\(node-set\)」 \(ページ 287\)](#)
- [「namespace-uri\(node-set\)」 \(ページ 287\)](#)
- [「position\(\)」 \(ページ 287\)](#)

count(node-set)

引数 node-set 内のノードの数を返します。

id(object)

id 関数は、一意の ID によって要素を選択します。id の引数が node-set 型の場合、その結果は、引数 node-set 内の各ノードの文字列値に id を適用した結果の和集合になります。id の引数が他のタイプの場合、引数は文字列関数の呼び出しのように文字列に変換され、この文字列は、空白で区切られたトークンのリストに分割されます（空白は、プロダクション S に一致する文字のシーケンスです）。その結果は、リスト内のいずれかのトークンと等しい一意の ID を持つコンテキストノードと同一のドキュメント内の要素を含むノードセットになります。

last()

式の評価コンテキストからコンテキストサイズに等しい数を返します。

local-name(node-set)

local-name 関数は、ドキュメント順で最初にある引数 node-set 内のノードの展開名のローカル部分を返します。引数 node-set が空である場合、または最初のノードに拡張名がない場合は、空の文字列が返されます。引数を省略すると、コンテキストノードを唯一のメンバーとする node-set にデフォルト設定されます。

name(node-set)

name 関数は、ドキュメント順で最初にある引数 node-set 内のノードの展開名を表す QName を含む文字列を返します。QName は、拡張名が表されているノードで有効な名前空間の宣言に関する拡張名を表している必要があります。通常、これは XML ソースで発生した QName です。複数のプレフィックスを同じ名前空間に関連付けたノードに有効な名前空間の宣言がある場合、これは当てはまりません。ただし、実装では、ノードの表現に元のプレフィックスに関する情報を含めることができます。この場合、実装によって、返される文字列が常に XML ソースで使用される QName と同じ文字列になります。引数 node-set が空であるか、最初のノードに展開された名前がない場合、空の文字列が返されます。引数を省略すると、コンテキストノードを唯一のメンバーとする node-set にデフォルト設定されます。

namespace-uri(node-set)

namespace-uri 関数は、ドキュメントの順序の最初にある引数 node-set 内のノードの拡張名の名前空間 URI を返します。引数 node-set が空である場合や最初のノードに拡張名がない場合、あるいは拡張名の名前空間 URI が nul である場合は、空の文字列が返されます。引数を省略すると、コンテキストノードを唯一のメンバーとする node-set にデフォルト設定されます。

position()

式評価コンテキストからコンテキスト位置に等しい数を返します。

数値関数

使用可能な数値関数は次のとおりです。

- [「ceiling\(number\)」 \(ページ 287\)](#)
- [「number\(object\)」 \(ページ 287\)](#)
- [「sum\(node-set\)」 \(ページ 288\)](#)
- [「round\(number\)」 \(ページ 287\)](#)

ceiling(number)

引数以上の整数である最も小さい（負の無限大に最も近い）数を返します。

floor(number)

引数以上ではない、整数である最も大きい（正の無限大に最も近い）数を返します。

number(object)

オブジェクトを数値に変換します。

round(number)

引数に最も近く、整数である数を返します。

- そのような数が 2 つある場合、正の無限大に最も近い方が返されます。
- 引数が NaN の場合、NaN が返されます。
- 引数が正の無限大の場合、正の無限大が返されます。
- 引数が正のゼロの場合、正のゼロが返されます。

- 引数が負の無限大の場合、負の無限大が返されます。
- 引数が負のゼロの場合、負のゼロが返されます。
- 引数がゼロ未満であっても-0.5 以上の場合、負のゼロが返されます。

sum(node-set)

引数ノードセット内の各ノードについて、ノードの文字列値を数値に変換した結果の合計を返します。

文字列関数

使用可能な文字列関数は次のとおりです。

- [「concat\(string1 string2 string3...\)」 \(ページ 288\)](#)
- [「contains\(string string\)」 \(ページ 288\)](#)
- [「normalize-space\(string\)」 \(ページ 288\)](#)
- [「starts-with\(mainString lookForString\)」 \(ページ 288\)](#)
- [「string\(object\)」 \(ページ 288\)](#)
- [「string-length\(string\)」 \(ページ 288\)](#)
- [「substring\(string number number\)」 \(ページ 289\)](#)
- [「substring-after\(string string\)」 \(ページ 289\)](#)
- [「substring-before\(string string\)」 \(ページ 289\)](#)
- [「translate\(string string string\)」 \(ページ 289\)](#)

concat(string1 string2 string3...)

渡された文字列を連結します。連結する任意の数の文字列を渡すことができます。

contains(string string)

最初の文字列に 2 番目の文字列が含まれている場合に true を返します。

normalize-space(string)

先頭と末尾のスペースを削除した、渡された文字列を返します。

starts-with(mainString lookForString)

最初の文字列が 2 番目の文字列で開始されている場合に、true を返します。

string(object)

渡されたオブジェクトを文字列に変換します。

string-length(string)

渡された文字列の長さを返します。

substring(string number number)

最初の数字（最初の文字は 1）で始まり、必要に応じて 2 番目の数字で示される長さの文字列の部分を返します。

substring-after(string string)

2 番目の文字列が最初に出現した後に現れる文字列の最初の部分を返します

substring-before(string string)

2 番目の文字列が最初に出現する前の最初の文字列の部分を返します。

translate(string string string)

2 番目の引数文字列に出現する文字を、3 番目の引数文字列内の対応する位置の文字に置き換えた、最初の引数文字列を返します。例えば、translate("bar", "abc", "ABC") は文字列 BAr を返します。2 番目の引数文字列に文字があり、3 番目の引数文字列内の対応する位置に文字がない場合（2 番目の引数文字列が 3 番目の引数文字列より長い場合など）は、最初の引数文字列でのその文字の出現は削除されます。例えば、translate("--aaa--", "abc-", "ABC") は "AAA" を返します。ある文字が 2 番目の引数文字列に複数回出現する場合は、最初の出現箇所によって置換する文字が決定されます。3 番目の引数文字列が 2 番目の引数文字列よりも長い場合、余分な文字は無視されます。

条件カウンタおよびその他の値に想定される式

次の表は、条件、カウンタ、およびその他の値に想定される式を示しています。

式のタイプ	想定される値
Assign Copy From/To 式	有効な from-spec 式
Assign Copy From/To クエリ	メッセージパートと場所を含む有効な from-spec 式
完了条件式ごと	M から N の条件を定義するために使用される符号なし整数
開始カウンタ値ごと	符号なし整数
最終カウンタ値ごと	符号なし整数
If 条件、If 式	ブール値
Else If 条件、Else If 式	ブール値
リンク遷移条件	リンクステータス（ブール値）
イベントハンドラ On Alarm 期限	正しい形式の値については、 「期限と期間の式」 （ページ 293）を参照してください。
イベントハンドラ On Alarm 期間	正しい形式の値については、 「期限と期間の式」 （ページ 293）を参照してください。

式のタイプ	想定される値
結合条件	ブール値
Repeat Until 条件	ブール値
Wait 期限	正しい形式の値については、 「期限と期間の式」 (ページ 293)を参照してください。
Wait 期間	正しい形式の値については、 「期限と期間の式」 (ページ 293)を参照してください。
While 条件	ブール値
変数の初期化式またはクエリ	有効な from-spec 式

クエリビルダの使用

[パーツ]、[変数]、[関数]、および [演算子] 選択リストから適切なものを選択して、[クエリ] テキストボックスにクエリを作成します。現在の式に使用する式言語を選択します。

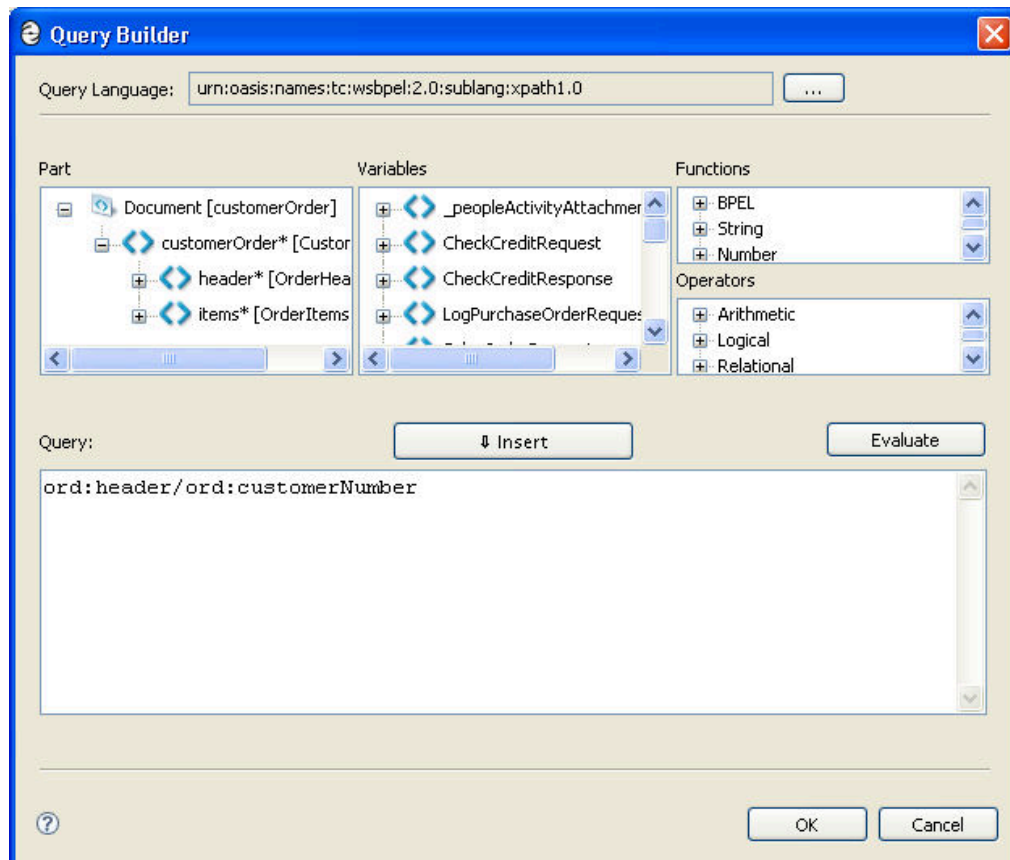
Assign アクティビティのコピー操作には、文字列、数値、またはブール値になる XPath（または選択した他の言語）のクエリ属性を含めることができます。クエリ属性の値は、ソースまたはターゲットの変数パート内の単一の値を識別するクエリ文字列です。

[クエリビルダ] ダイアログは [式ビルダ] ダイアログに似ていますが、パーツを簡単に選択できるように [変数パート] ツリーが追加されています。

XPath 構文の詳細については、<http://www.w3.org/TR/xpath> で「XPath 仕様」を参照してください。

クエリを作成する手順

1. Assign アクティビティの [プロパティ] ビューで、[コピー操作] を選択します。
2. [コピー操作] ダイアログで、[タイプ] フィールドから [変数] を選択します。
3. 必要に応じて [パート] を選択することで、クエリをすばやく作成できます。
4. [クエリビルダ] をクリックします。



5. 必要に応じて、[クエリ言語] フィールドの [ダイアログ (...)] ボタン] を選択して、Process Developer 以外のエンジンでサポートされる別のクエリ言語の URI を選択します。以下の手順を実行してください。
 - [クエリ言語] ダイアログ（または BPEL プロセスの [プロパティ] ビュー）で、Process Developer 以外のエンジンに設定された言語に対する別の URI を入力します。この URI は Process Developer では無視されます。
 - デフォルトに戻すには、URI をクリアして **[OK]** をクリックします。
6. 次のヒントに従って、ボックスから変数パート、関数、および演算子を選択してクエリを作成します。
 - ショートカットについては、[「コンテンツアシストの使用」 \(ページ 277\)](#)に記載されているように、コンテンツアシストを使用します
 - 変数、パーツ、関数、または演算子を選択してから、**[挿入]** を選択します
 - クエリの位置情報を生成するには、変数パートをダブルクリックします
 - \$variableReference 関数をクエリボックスに貼り付けるには、変数をダブルクリックします
 - 場所をクエリボックスに貼り付けるには、パーツの下をブランチをダブルクリックします
 - [「Process Developer のカスタム関数全般」 \(ページ 279\)](#)に記載されているように、カスタム関数を追加します
 - [第 17 章, 「POJO および XQuery カスタム関数」 \(ページ 295\)](#)に記載されているように、独自のカスタム関数を追加します
7. 変数パーツのサンプル値を置き換え、式を簡略化してその値を表示するには、**[評価]** を選択します。

入力リンクに対する結合条件の作成

結合条件を設定して、入力リンクの成功または失敗を評価します。結合条件は、アクティビティを対象とするリンクのステータスを示すブール式です。

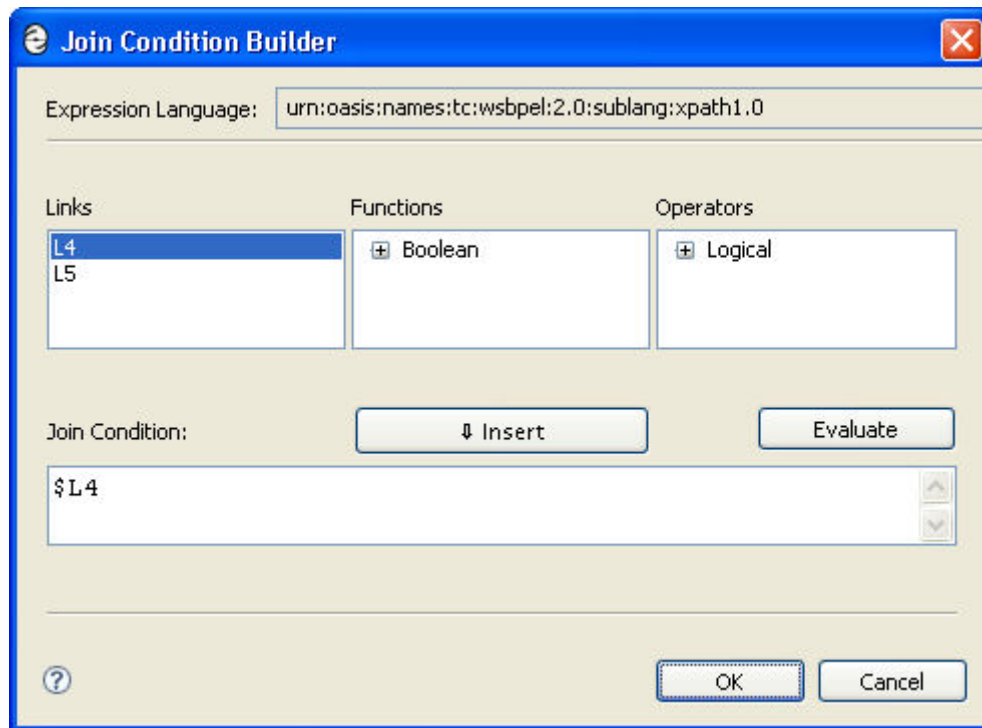
1 つ以上のリンクがアクティビティに接続されている場合、結合条件を設定することで、入力リンクの成功または失敗を評価できます。結合条件はすべてのアクティビティタイプのプロパティであるため、リンクステータスに基づいて複雑な実行条件を構築することができます。詳細については、[第 15 章, 「リンク」 \(ページ 264\)](#)を参照してください。

結合条件が true と評価された場合、プロセスは正常に続行されます。結合条件が false と評価された場合、プロセスは joinFailure フォールトをスローする可能性があります。

結合条件のあるアクティビティに対して *suppressJoinFailure* プロパティを [はい] に設定した場合、結合条件が false と評価されても joinFailure フォールトはスローされません。

結合条件を作成する手順

1. 入力リンクのあるアクティビティの [プロパティ] ビューを表示します。
2. [詳細プロパティの表示] ボタンを選択して、アクティビティのすべてのプロパティを表示します。
3. [結合条件] の横にある [ダイアログ (...)] ボタン をクリックして、次のダイアログを表示します。



4. リンクをダブルクリックして、BPEL リンクステータス関数を [結合条件] ボックスに貼り付けます。この関数は、渡されたリンクのステータスを示すブール値を返します。
5. [OK] をクリックします。

期限と期間の式

一部のアクティビティには、期限または期間を設定するためのプロパティが含まれています。期限と期間の式については、XML スキーマの仕様に示されている例を参照してください。

以下の例に示すように、期限と期間の値は必ず一重引用符で囲んでください。

以下を参照してください。

- <http://www.w3.org/TR/xmlschema-2/#duration>
- <http://www.w3.org/TR/xmlschema-2/#datetime>
- <http://www.w3.org/TR/xmlschema-2/#date>

また、[「XPath 関数を作成するためのヒント」 \(ページ 302\)](#)に記載されているフォーマットオプションも参照してください。

例:

- 1 秒という期間:
`<wait for=" 'PT1S' "/>`
- 1 年 2 か月 3 日 10 時間 30 分という期間:
`<wait for=" 'P1Y2M3DT10H30M' "/>`
- 日時による期限:
`<wait until=" '2010-12-12T12:00' "/>`

パート III: 関数、イベント、エラー、および相関

このセクションのトピックでは、関数の作成方法、イベントとエラーへの応答方法、およびユーザーの操作を相互に関連付ける方法について説明します。

第 17 章

POJO および XQuery カスタム関数

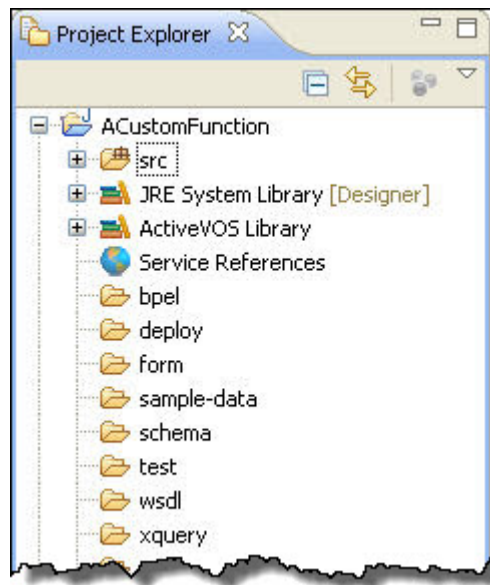
Process Developer では、POJO (Plain Old Java Object) または XQuery 関数としてカスタム関数を記述できます。組み込み機能を使用することで、式ビルダでのカスタム関数の作成と利用を簡単に行うことができます。

カスタム関数の概要

BPEL プロセスでは、カスタム関数を作成して使用できます。カスタム関数は、Process Developer ライブラリの関数コンテキストインタフェースが実装された POJO です。また、式ビルダとクエリビルダの関数リストにカスタム関数を登録して表示するための Process Developer の注釈も含まれています。

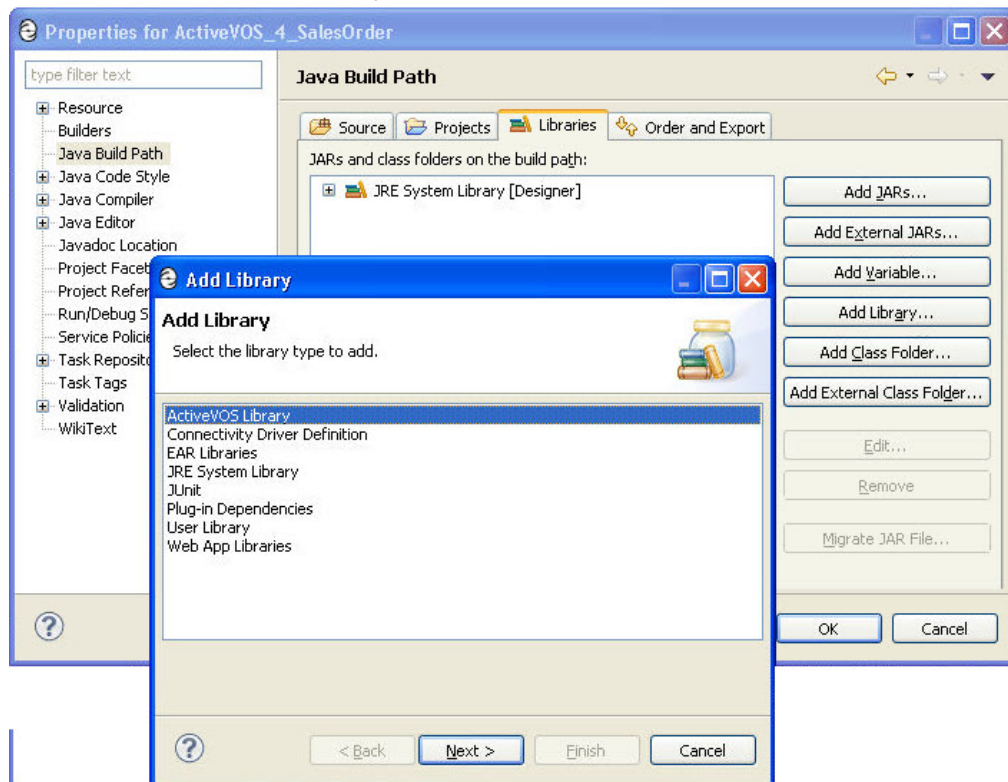
1 つ以上のプロジェクトで、カスタム関数を作成して使用できます。カスタム関数は、デプロイメントコントリビューション内のプロジェクトリソースとしてデプロイされます。カスタム関数は、BPEL ファイルと同じプロジェクトまたは参照されるプロジェクトに含めることができます。カスタム関数をサポートする JAR ファイルが、プロジェクトのクラスパスに含まれている必要があります。

関数コンテキストインタフェースを実装する簡単な方法として、[「Orchestration プロジェクトのテンプレート」 \(ページ 86\)](#)に記載されている新しい Java 対応の Orchestration プロジェクトを作成します。図に示すように、必要な Process Developer ライブラリが自動的にプロジェクトへ追加されます。



既存の Java 対応 Orchestration プロジェクトがある場合は、次のように Process Developer ライブラリを追加します。

1. プロジェクト名を右クリックし、**プロパティ** を選択します。
2. **Java ビルドパス** を選択します。
3. **ライブラリ** タブから、**ライブラリを追加** を選択します。
4. 例に示すように、**Process Developer ライブラリ** を選択します。



[「関数コンテキストの実装と注釈の追加」](#) (ページ 297) も参照してください。

関数コンテキストの実装と注釈の追加

プロセスサーバーは、カスタム関数拡張機能をサポートする Java フレームワークを使用します。このフレームワークは、利用可能な Java インタフェースとクラスのセットです。IAeFunction および IAeFunctionContext は、実行時にエンジンがアクセス可能なオブジェクトを作成するために実装します。AeFunctionCallException および AeUnresolvableException は、フォールト状態を通知するために使用します。これらのタイプは、ae_rtbpel.jar の org.activebpel.rt.bpel パッケージを使用して解決されます。さらに、AeUnresolvableException は、ae_rt.jar で提供される AeException を拡張します。

カスタム関数は、任意の式言語で使用できます。

手順 1: 関数コンテキストの作成

最初の作業は、IAeFunctionContext インタフェースが実装された新しいクラスである関数コンテキストを作成することです。

このクラスは、getFunction(String)という 1 つのパブリックメソッドを公開します。IAeFunction インタフェースを実装するタイプのオブジェクトを返します。このメソッドは、String 引数を使用して関数のローカル名を示し、そのオブジェクトを選択します。ユーザーは、1 つの関数コンテキストクラスを介して複数のカスタム関数を実装できます。getFunction()は、名前での関数を見つけることができない場合に、AeUnresolvableException をスローします。

注釈

@AeFunctionContext 注釈を追加して、クラスレベルでクラスに注釈を付けます。

式ビルダの関数リストに表示する関数単位ごとに、メソッドレベルで@AeFunctionUnit 注釈を追加します。

次の例は、関数コンテキストを示しています。

```
@AeFunctionContext(name="myContext", namespace="myns")
public class CustomContext implements IAeFunctionContext
{
    @AeFunctionUnit(
        prefix = "ld",
        display = "MyLDAP",
        hoverText = "My LDAP Functions",
        functions = {
            @AeFunction(
                syntax = "${prefix}:getOrgUnit(${caret})",
                display = "getOrgUnit(id)",
                hoverText = "Returns OU details"),
            @AeFunction(
                syntax = "${prefix}:getSiblings(${caret})",
                display = "getSiblings(id)",
                hoverText = "Returns siblings nodes"),
        }
    )
    public IAeFunction getFunction(String aFunctionName)
        throws AeUnresolvableException
    {
    }
}
```

ここで、

- @AeFunctionContext name は、プロセスコンソールカタログ内の関数コンテキストの表示名です。
- @AeFunctionUnit は関数のグループを表します
 - prefix は、ユニット内のすべての機能に使用されます
 - display は、式ビルダの関数リストのユニット名です。

- hoverText はユニットのマウスオーバーヘルプです
- functions は関数のリストです
- @AeFunction は関数を表します
 - syntax は、関数をダブルクリックしたときに [式] テキストボックスに追加される関数の構文です。構文には \${prefix} と \${caret} が含まれることに注意してください。キャレットトークンで設定したカラムにカーソルが表示されます。
 - display は、式ビルダの関数リストの関数名です。
 - hoverText は、関数のマウスオーバーヘルプです

手順 2: カスタム関数の作成

2 番目の作業として、それぞれのカスタム関数に `IAeFunction` インタフェースを実装した、実際のカスタム関数を作成します。これらの作業では、`call(IAeFunctionExecutionContext, List)` という 1 つのパブリックメソッドも公開し、これは作成した関数のタスクを実行します。関数の引数リスト（存在する場合）は、`List` 引数を使用して呼び出しメソッドに渡されます。この関数は、必要に応じて、オプションの関数実行 `IAeFunctionExecutionContext` を使用します。個々のカスタム関数は、タスクを完了できないような問題が発生した場合に、ルート例外メッセージを含むように構築された `AeFunctionCallException` をスローします。

手順 3: パッケージの作成

関数コンテキストと関数クラスを作成し、実際のカスタム関数の動作を実行するクラスをテストした後に、コントリビューションを作成します。デプロイメントのコントリビューションを作成すると、プロジェクト内のすべての関数コンテキストが自動的にパッケージ化されます。

カスタム関数は、コントリビューション内の BPEL ファイル、またはこのプロジェクトを参照する別のプロジェクトのコントリビューションにのみ使用できます。カスタム関数は、プロジェクトリソースの依存関係または参照された依存関係として、あるいは参照されていない場合は追加のリソースとして、[ビジネスプロセスアーカイブのエクスポート] の [カタログリソース] ページに一覧表示されることに注意してください。詳細については、[「プロジェクトの依存関係のデプロイと除外された依存関係の表示」](#) (ページ 485) を参照してください。

カスタム関数のサンプル

以下に、サンプルのカスタム関数を示します。

```
package com.acme.functions;
import java.util.List;
import org.activebpel.rt.bpel.function.AeFunction;
import org.activebpel.rt.bpel.function.AeFunctionCallException;
import org.activebpel.rt.bpel.function.AeFunctionContext;
import org.activebpel.rt.bpel.function.AeFunctionUnit;
import org.activebpel.rt.bpel.function.AeUnresolvableException;
import org.activebpel.rt.bpel.function.IAeFunction;
import org.activebpel.rt.bpel.function.IAeFunctionContext;
import org.activebpel.rt.bpel.function.IAeFunctionExecutionContext;
@AeFunctionContext(name = "AcmeContext", namespace = "http://acme.com/functions")
public class GenericFunctionContext implements IAeFunctionContext {
    private IAeFunction echoFunction = new EchoFunction();
    private IAeFunction companyNameFunction = new CompanyNameFunction();
    @Override
    @AeFunctionUnit(prefix = "ac", display = "ACME",
        hoverText = "Acme company functions", functions = {
            @AeFunction(syntax = "${prefix}:echo(${caret})", display = "echo(param)", hoverText = "echo"
```

```

function that returns the given string"),
    @AeFunction(syntax = "${prefix}:companyName()", display = "companyName()", hoverText = "Returns
company name") })
    public IAeFunction getFunction(String aFunctionName)
        throws AeUnresolvableException {
        if (aFunctionName.equals("echo"))
            return echoFunction;
        else if (aFunctionName.equals("companyName"))
            return companyNameFunction;
        return null;
    }
    private class EchoFunction implements IAeFunction {
        @Override
        public Object call(IAeFunctionExecutionContext aContext, List aArgs)
            throws AeFunctionCallException {
            if (aArgs.size() == 1)
                return aArgs.get(0);
            return null;
        }
    }
    private class CompanyNameFunction implements IAeFunction {
        @Override
        public Object call(IAeFunctionExecutionContext arg0, List arg1)
            throws AeFunctionCallException {
            return "acme";
        }
    }
}

```

プロセスサーバーへのグローバルカスタム関数の追加

Java ベースのカスタム関数を Process Developer 組み込みエンジンまたはプロセスサーバーに追加して、これらの関数がデプロイされたコントリビューションだけでなく、すべてのプロセスでグローバルに使用できるようにすることができます。

最初に、サーバーをホストするマシンに jar ファイルをコピーします。

次に、カスタム関数情報を追加して、関数をサーバーに認識させます。これは、プロセスコンソール内で **【管理】** > **【グローバル機能】** を選択して行います。

詳細を追加するためのヘルプについては、**【ヘルプ】** リンクを選択し、**グローバル関数のヘルプトピック**を表示してください。

XQuery 関数の記述

Process Developer には、XQuery Development Tool (XQDT) プロジェクトが統合されています。XQDT を使用して、式ビルダの関数リストに自動的に含まれる XQuery カスタム関数を Orchestration プロジェクトに作成できます。

XQDT には、構文の強調表示、検証、コンテンツ支援、および実行サポートを備えた XQuery エディタが含まれています。Process Developer の式ビルダ内で使用する前に、エディタで XQuery 関数を簡単に作成およびテストできます。また、関数に変更を加えて保存することもできます。更新は自動的に BPEL ファイルに含まれます。

式ビルダに含めるには、各関数がライブラリモジュールの一部である必要があります。1 つの XQuery ファイルで関数のモジュールを作成し、import module ディレクティブを使用して、作成したモジュールで定義した関数を他の XQuery で使用できます。

XQDT は、XQuery 1.0 および多くの XQuery 3.0 構成をサポートします。

式ビルダの関数セクションをダブルクリックして XQuery 関数を追加すると、XQuery ファイルがインポートとして追加されます。BPEL プロセスのインポートとして参照される XQuery ファイルに注意を払い、そのリストと、インポートモジュールを使用する XQuery ファイルにインポートとして明示的に追加されるものの間に重複がないことを確認する必要があります。

既存のプロジェクト（バージョン 9.0 より前）をお持ちの場合は、プロジェクトエクスプローラでプロジェクトを右クリックし、**[XQuery ネーチャーの追加]** を選択します。

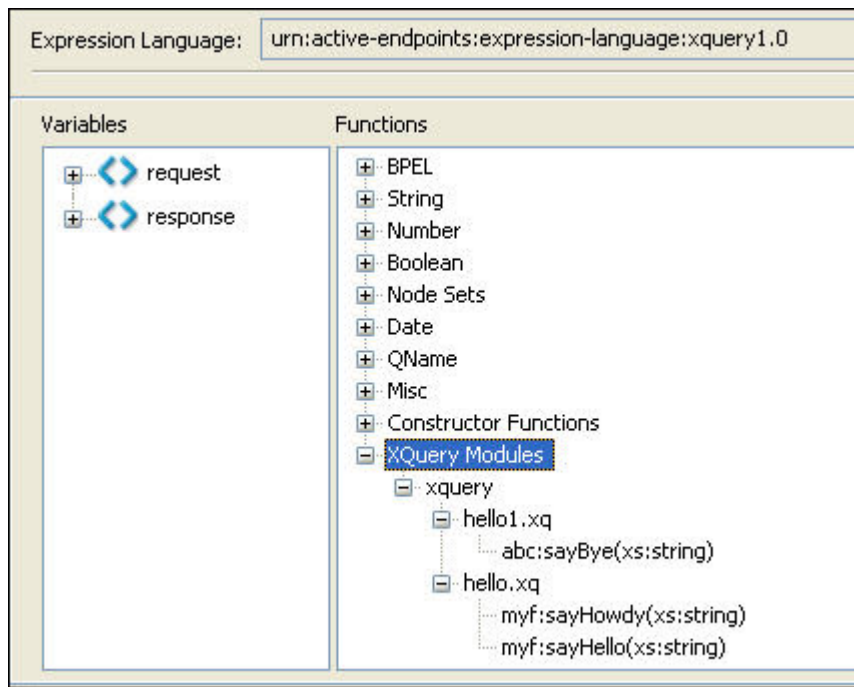
XQuery 関数の作成を開始する手順

1. Orchestration プロジェクトの XQuery フォルダ（または別のフォルダ）を右クリックし、**[新規] > [XQuery モジュール]** を選択します。
2. myNewFunctionModule.xq などのように、.xq という拡張子を付けてファイルの名前を設定します。
3. 次のいずれかを選択します。
 - メインモジュール。評価されるクエリ本文が含まれています
 - ライブラリモジュール。モジュール宣言が含まれていますが、クエリ本体は含まれていません。モジュール宣言は、インポートするモジュールを識別する URI を提供します。モジュール宣言にプレフィックスと名前空間を入力します。
4. **[完了]** を選択すると、XQuery エディタが開きます。コンテンツアシスト (Ctrl + Space キー) を使用して、新しい機能を開始します。

次の例は、ライブラリモジュールのモジュール宣言を示しています。この宣言は、次のように XQuery 関数が関数リストに表示されるようにするために必要です。

```
(: The version declaration in line one is optional.
   Version 1.0 does not allow any use of version 3.0 constructs. :)
xquery version "1.0" encoding "utf-8";
module namespace xqf= "TechSupportLevel1";
declare function xqf:pingModemResult ((: $param as type, ... :)) as item() {
    insert_an_expression_here
};
```

XQuery ファイルを保存すると、図に示すように、関数が自動的に式ビルダで使えるようになります。



関数を使用すると、対応する XQuery モジュールがプロセスのインポートノードに自動的に追加されます。

ヒント: 関数のプロパティを表示するには、[アウトライン] ビューで [インポート] の下の関数を選択します。B ユニットテストを実行する場合は、XQuery モジュールをリソースとして追加し、図のようにタイプ URI を指定します。

関連事項:

- [「XQuery エディタの使用」 \(ページ 301\)](#)
- [「XQuery 関数を作成するためのヒント」 \(ページ 302\)](#)
- [「XQuery エディタでの XQuery 関数のテスト」 \(ページ 302\)](#)

XQuery エディタの使用

拡張子が.xq であるファイルを作成するか開いた場合、XQuery エディタが開きます。次のような機能があることに注意してください。

- このエディタにはコンテンツアシスト機能が含まれており、CTRL + Space を選択することで、開いているファイルにインポートされたモジュールに存在する XQuery 構成、システム式、および独自の関数のリストを開くことができます。
- 構文の強調表示は、XQuery 式のさまざまな要素を示します
- 検証は自動的に行われ、エラーが報告されます
- 式をテストするためのローカルスクリプトを実行する場合に、[実行] 機能を使用できます。[「XQuery エディタでの XQuery 関数のテスト」 \(ページ 302\)](#)を参照してください。
- [設定] の [XQuery] セクションでは、構文の強調表示やその他の機能を設定できます。

[「XQuery 関数を作成するためのヒント」 \(ページ 302\)](#)も参照してください。

XQuery 関数を作成するためのヒント

Process Developer は、次のようなすべての XQuery 1.0 構成と一部の XQuery 3.0 構成をサポートします。

- Group by
- Try-Catch
- date、dateTime、time、および number の format 関数

XQuery 3.0 構成を使用するには、ファイルに次の宣言を必ず追加してください。

```
xquery version "3.0";
```

Process Developer では、XQuery プロセッサとして Saxon を使用します。特定の XQuery 3.0 構成のサポートについて不明な点がある場合は、Saxon の Web サイト (<http://www.saxonica.com/documentation9.3/changes/intro.xml>) を参照してください。

XQuery エディタでの XQuery 関数のテスト

関数のテストスクリプトを記述し、XQuery エディタでそのスクリプトを実行できます。Process Developer には、スクリプト実行用のデフォルトのインタプリタが用意されています。

例えば、新しい.xq ファイルを作成し、以下の *Kitty* 関数のように関数引数をハードコーディングする簡単なテストスクリプトを記述します。

```
xquery version "1.0" encoding "utf-8";
import module namespace xyz="myFunctions" at "hello.xq";
xyz:sayHello('Kitty')
```

このスクリプトを実行するには、エディタでマウスを右クリックし、**[実行] > [XQuery]** を選択します。

テストスクリプトに外部値を渡す

テストスクリプトに外部変数が定義されている場合、それらの変数の値は、**[実行]** の起動設定から引数として渡すことができます。以下は、外部変数を取り込む簡単なスクリプトで、その後にスクリプト引数の作成方法を説明しています。

```
xquery version "1.0" encoding "utf-8";
import module namespace xyz="myFunctions" at "hello.xq";
declare variable $ip as xs:string external;
xyz:sayHello($ip)
```

[実行] 設定の作成:

1. エディタで右クリックして、**[実行] > [実行設定]** を選択します。
2. **[新規]** を選択し、最初のページにプロジェクト名とテストスクリプト名を追加します。
3. **[引数]** ページで、ip="Kitty"などのスクリプト引数を追加します。
4. 名前を付けた設定を実行します。

インタプリタの動作のカスタマイズ

Process Developer には、デフォルトの Saxon インタプリタが含まれています。**[設定] > [XQuery] > [インタプリタ]** を選択すると、このインタプリタの引数を変更できます。

Process Developer のデフォルトのインタプリタをダブルクリックすると、設定を行うことができます。インタプリタの引数は、開発の要件に基づいて変更できます。引数のドキュメントは、<http://www.saxonica.com/documentation/using-xquery/commandline.xml> で参照できます。

最初の引数であるクラス名 `org.activebpel.rt.bpel.ext.expr.impl.xquery.AeQuery` は変更しないでください。

第 18 章

システムサービス

テンプレートを選択して、システムサービスベースのプロセスの構築を開始します。

システムサービスを使用したプロセスの構築に関する説明が記載されたトピックは次のとおりです。

- システムサービスベースのプロセスへの BPEL テンプレートの使用
- 警告サービス
- 再試行 - ポリシーサービス
- ID サービス
- 電子メールサービス
- 警告の監視サービス
- レポートサービス
- サーバログサービス
- シェルコマンド呼び出しサービス
- データアクセスサービス
- 移行サービス

システムサービスベースのプロセスへの BPEL テンプレートの使用

システムサービスベースのプロセスは、サーバーイベントがトリガされた場合にアラートを送信するなど、さまざまな方法でプロセスサーバーとのインタラクションを行います。プロセスは、プロセスコンシューマおよびパートナーサービスプロバイダのパーティシパントを含むインタフェースに基づいています。

プロセスコンシューマインタフェースの場合、新規 BPEL プロセスウィザードを使用してプロセスの構築を開始できます。**【ファイル】 > 【新規】 > 【BPEL】** を選択して **【次へ】** をクリックすると、**【テンプレートに基づく新しい BPEL プロセス】** ページが表示されます。

このウィザードページには、システムにデプロイされた WSDL に基づいて作成可能なプロセスのリストが表示されます。BPEL プロセスを作成するために WSDL ファイルをインポートする必要はありません。WSDL リソースからの操作に基づいて、Receive アクティビティを使用してプロセスを開始します。

テンプレートによって、次のいずれかの WSDL に基づいて BPEL プロセスが生成されます。

- アラートプロセス。詳細については、「アラートサービス」を参照してください。
- ポリシープロセスを再試行します。詳細については、「再試行 - ポリシーサービス」を参照してください。

- カスタム B4P 通知。詳細については、「*Process Developer のヘルプ*」の「ヒューマンタスク」を参照してください。
- REST ベースのプロセス。詳細については、「*REST ベースのサービスの使用*」を参照してください。
- イベントアクションプロセス。詳細については、「*Event-Action BPEL プロセスの作成*」を参照してください。
- 監視アラート。詳細については、「*警告の監視サービス*」を参照してください。

これらのプロセスのいずれかをデプロイする場合、BPR には WSDL または XSD が存在しないことに注意してください。これらはすでにデプロイされており、サーバーで使用可能です。

上記のサービスに加えて、[パーティシパント] ビューから開始して構築可能なその他のサービスがあります。パートナーサービスプロバイダまたはプロセスコンシューマウィザードでシステムサービスを使用すると、電子メールの送信や ID サービスからのユーザーの取得といった特別な目的で操作を使用できます。詳細については、「システムサービスインタフェース」を参照してください。

警告サービス

さまざまな理由により、アクティビティはキャッチされないフォールトをスローする場合があります。プロセスサーバーには、キャッチされないフォールトでプロセスを一時停止する機能があり、必要に応じて警告をトリガすることもできます。

キャッチされないフォールトが発生すると、プロセスサーバーは警告サービスをインスタンス化し、フォールトの発生したアクティビティをステップオーバーしてプロセスを再開したり、フォールトの原因であるデータ値を自動的に修正したりします。または、単に処理でフォールトが発生したことを管理者に通知するといったアクションを呼び出すこともあります。

警告サービスとして機能する BPEL プロセスを設計することができます。このプロセスは、alert.wsdl という名前の WSDL ファイルを基にして設計します。以下の手順で説明するように、この WSDL に基づた BPEL プロセステンプレートが用意されています。また、このファイルはシステムサービスとして [パーティシパント] ビューに自動的に読み込まれるため、プロセスの受信と応答を追加するプロセスコンシューマパーティシパントを作成できます。

基本的には、WSDL を使用して、プロセス ID、名前空間、名前、変数の場所のパス、およびフォールトが発生したプロセスのステータスに関する情報をサーバーから受信する警告操作を定義します。

警告の監視 BPEL プロセスに送信される入力メッセージには、次の例に示すようなパートがあります。

```
<ns1:alert xmlns:ns1=
"http://www.active-endpoints.com/services/alert"
  processId= "101"
  processNamespace="http://tempuri.org"
  processName="myProcess"
  locationPath="/process/variables
    /variable[@name='testMessage']"
  status="suspended"
  faultName="someFault"
  faultNamespace="someFaultNamespace"
/>
```

警告の監視 BPEL プロセスを作成する際に考慮する必要がある、一般的な一部の手順を次に示します。

警告の監視 BPEL プロセスを作成する手順

1. プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。
2. [ファイル] > [新規] > [BPEL プロセス] を選択します。
3. プロセスに名前を付けて、[次へ] をクリックします。

4. [プロセステンプレート] ページで、警告のプロセステンプレートを選択します。
Process Editor キャンバスで、*ReceiveAlert* アクティビティが存在する新しいファイルが開きます。
Receive アクティビティは、警告操作に基づいています。
5. プロセスの構築を完了し、実際の使用時にプロセスでフォールトが発生した場合に実行するプログラミングロジックを追加します。例えば、「[電子メールサービス](#)」(ページ 311)に記載されているようにメールを送信します。
6. [デプロイメント記述子] ウィザードを起動して、[パートナーリンク] タブで次の手順を実行します。
 - a. AlertPL マイロールのパートナーリンクを選択します。
 - b. マイロールのパートナーリンクのサービス名を入力します。
7. プロセスサーバーを起動して、警告サービスプロセスをサーバーにデプロイします。BPR には、WSDL や XSD はありません。これらはすでにデプロイされており、サーバー上で利用できます。
8. サーバーのプロセスコンソールで、[警告サービス] ページにサービスを追加します。詳細については、「[プロセスコンソールのヘルプ](#)」を参照してください。

ヒント: ダウンロード可能なさまざまな種類の警告サービスのサンプルについては、<http://www.activevos.com> 内にある「[電子メールおよび ID サービスを使用したプロセスサーバーの警告の処理](#)」を参照してください。

データアクセスサービス

データアクセスシステムサービスを使用して、BPEL プロセス内のデータソースを直接操作できます。このサービスを使用して、指定したデータソースで 1 つ以上の SQL ステートメントを実行し、応答として結果セットを受け取る呼び出しアクティビティを作成します。

データアクセスサービスは、execSQL と execMultiSQL という 2 つの操作を提供します。これらの操作により、単一の SQL ステートメントまたは複数の SQL ステートメントを実行できます。データまたはストアドプロシージャをデータソースから選択、挿入、更新、削除するかどうかに関わらず、任意の SQL ステートメントを実行できます。

デプロイメント中に、ターゲットデータソースを指定する必要があります。

データソースでステートメントを実行する BPEL プロセスの作成

以下の手順を使用して、データソースでステートメントを実行する BPEL プロセスを作成します。

プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。

1. [ファイル] > [新規] > [BPEL プロセス] を選択します。
2. プロセスに名前を付けて、[完了] をクリックします。
3. [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
4. [インタフェース] ツリーで、[システムサービス] を選択します。
5. [データアクセス] を選択します。
6. 作成したパートナーサービスプロバイダから execSQL または execMultiSQL 操作をキャンバスにドラッグして、Invoke アクティビティを作成します。
7. 表の下に例に示した選択肢を選び、入力データとして入力します。

次の表に、データアクセス要求メッセージ要素を示します。

この表の後に、以下のような例を示します。

- 単一ステートメントに対する要求の例とその応答

- 複数ステートメントを含む要求の例とその応答
- 挿入、更新、または削除ステートメントからのパラメータベースの要求と応答
- バイナリデータの処理
- フォールト応答の例

ストアドプロシージャの呼び出しの例（MySQL、SQL Server、および Oracle の例を示します）。

execSQL の入力要素	説明
sqlStatement	必須の<statement>とオプションの属性を含む要素
includeMetadata	実際の戻り値データを返す前に、クエリカラムに関するメタデータの行を返します（デフォルトは false です）。メタデータには、データ型と表示サイズが含まれます。
maxRows	返される結果行の最大数（デフォルトは 0 または無制限）。
maxWaitSeconds	結果の処理を待機する最大時間（秒単位）。このオプションでは、クエリの実行にかかる時間は考慮されません。このオプションを使用して、大きなクエリ結果にかかる長い処理を中断します。（デフォルトは 0 または無制限）。
columnCase	結果セットのカラムの大文字と小文字の形式を指定します（デフォルトはサーバーから「変更なし」です）。この要素に対して定義したスキーマ列挙には、小文字、大文字、および変更なしの文字が含まれます。
statementId	結果を識別するために、要求の一部として指定します（デフォルトでは、statement-1、statement-2 といった ID が自動生成されます）
statement	必須。実行する SQL ステートメント。
parameter	データソースに挿入、更新、または削除するデータを示します。パラメータ値は、ステートメントに関連付けられた<parameterBatch>要素内に順番で表示されます。
sqlType	パラメータ属性。タイプには、string（デフォルト）、byte、short、int、long、float、double、date、binary、または clob などがあります。
attachmentId	パラメータ属性。結果の添付ファイルがある場合は、それらを識別するために要求の一部として指定します。
hasResultSet	ストアドプロシージャがデータを返すことを示します。ストアドプロシージャがデータを返す場合にのみ、このパラメータを使用してください。

単一ステートメントの要求の例:

```
<das:dataAccessRequest xmlns:das="http://schemas.active-endpoints.com/data-access/2010/04/data-access.xsd">
  <das:sqlStatement includeMetadata="true" maxRows="100" maxWaitSeconds="120" columnCase="lowercase">
    <das:statement>SELECT EngineId, Name, State FROM AeEngine ORDER BY 1</das:statement>
  </das:sqlStatement>
</das:dataAccessRequest>
```

上記の要求からの応答の例:

```
<dataAccessResponse xmlns="http://schemas.active-endpoints.com/data-access/2010/04/data-access.xsd"
  statementId="statement-1" >
  <metadata>
    <engineid dataType="INT UNSIGNED" displaySize="10" xmlns=""/>
    <name dataType="VARCHAR" displaySize="255" xmlns=""/>
    <state dataType="TINYINT" displaySize="4" xmlns=""/>
  </metadata>
  <row>
```

```

        <engineid xmlns="">1</engineid>
        <name xmlns="">machine1</name>
        <state xmlns="">1</state>
    </row>
</dataAccessResponse>

```

複数のステートメントを含む要求の例:

```

<das:multiDataAccessRequest xmlns:das="http://schemas.active-endpoints.com/data-access/2010/04/data-access.xsd">
  <das:sqlStatement includeMetadata="false" statementId="num-processes">
    <das:statement>SELECT COUNT(*) AS ProcessCount FROM AeProcess</das:statement>
  </das:sqlStatement>
  <das:sqlStatement includeMetadata="false" statementId="num-plans" sqlcode='0' sqlstate='0'>
    <das:statement>SELECT COUNT(*) AS PlanCount FROM AePlan</das:statement>
  </das:sqlStatement>
  <das:sqlStatement includeMetadata="false" statementId="metainfo">
    <das:statement>SELECT * FROM AeMetaInfo</das:statement>
  </das:sqlStatement>
</das:multiDataAccessRequest>

```

上記の複数のステートメント要求からの応答の例:

```

<multiDataAccessResponse xmlns="http://schemas.active-endpoints.com/data-access/2010/04/data-access.xsd">
  <result statementId="num-processes">
    <row>
      <ProcessCount xmlns="">3</ProcessCount>
    </row>
  </result>
  <result statementId="num-plans">
    <row>
      <PlanCount xmlns="">18</PlanCount>
    </row>
  </result>
  <result statementId="metainfo">
    <row>
      <PropertyName xmlns="">DatabaseType</PropertyName>
      <PropertyValue xmlns="">mysql</PropertyValue>
    </row>
    <row>
      <PropertyName xmlns="">SiteId</PropertyName>
      <PropertyValue xmlns="">0</PropertyValue>
    </row>
    <row>
      <PropertyName xmlns="">SiteName</PropertyName>
      <PropertyValue xmlns="">mySite1</PropertyValue>
    </row>
    <row>
      <PropertyName xmlns="">Version</PropertyName>
      <PropertyValue xmlns="">1.2.3 ActiveVOS Enterprise</PropertyValue>
    </row>
  </result>
</multiDataAccessResponse>

```

データアクセス呼び出し出力のシミュレーション

シミュレーションを開始する前に一部の追加手順を実行することで、シミュレーション中の実際のデータアクセス要求と応答をテストできます。この手順には、Process Developer にバンドルされている Eclipse Data Source Explorer の使用が含まれます。

Data Source Explorer を使用すると、実際のステートメントの実行からサンプルデータを生成し、そのデータを正しいスキーマ形式で Process Developer にエクスポートできます。これにより、結果セットから返された実際のデータを、データのマッピング先のプロセス変数で使用するようになります。

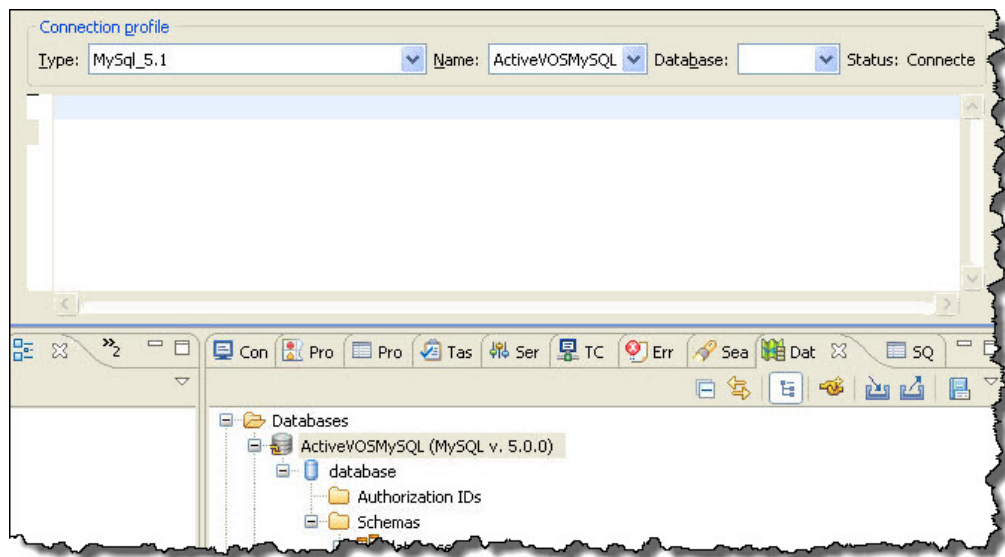
手順 1. データベース接続を設定します。

1. **【ウィンドウ】 > 【ビューの表示】 > 【その他】 > 【データ管理】 > 【Data Source Explorer】** を選択します。

2. [データベース接続] を右クリックして、[新規] を選択します。
3. リストから [接続プロファイルタイプ] を選択します。例えば、[MySQL] を選択します。
4. 新しいプロファイルに名前を付け、[次へ] を選択します。
5. [ドライバ] ページで、リストからデータベースドライバを選択します。ドライバがリストにない場合は、[新規ドライバ] を選択し、ドライバの定義を指定します。
6. データベースに接続するための URL を入力します。例えば、MySQL のデフォルト URL は jdbc:mysql://localhost:3306/[データベース名] です。
7. [接続のテスト] を選択し、成功した場合はウィザードを終了します。
8. Data Source Explorer に新しいデータベースがリストされることに注意してください。

手順 2。データベースに接続し、次のようにステートメントを実行します。

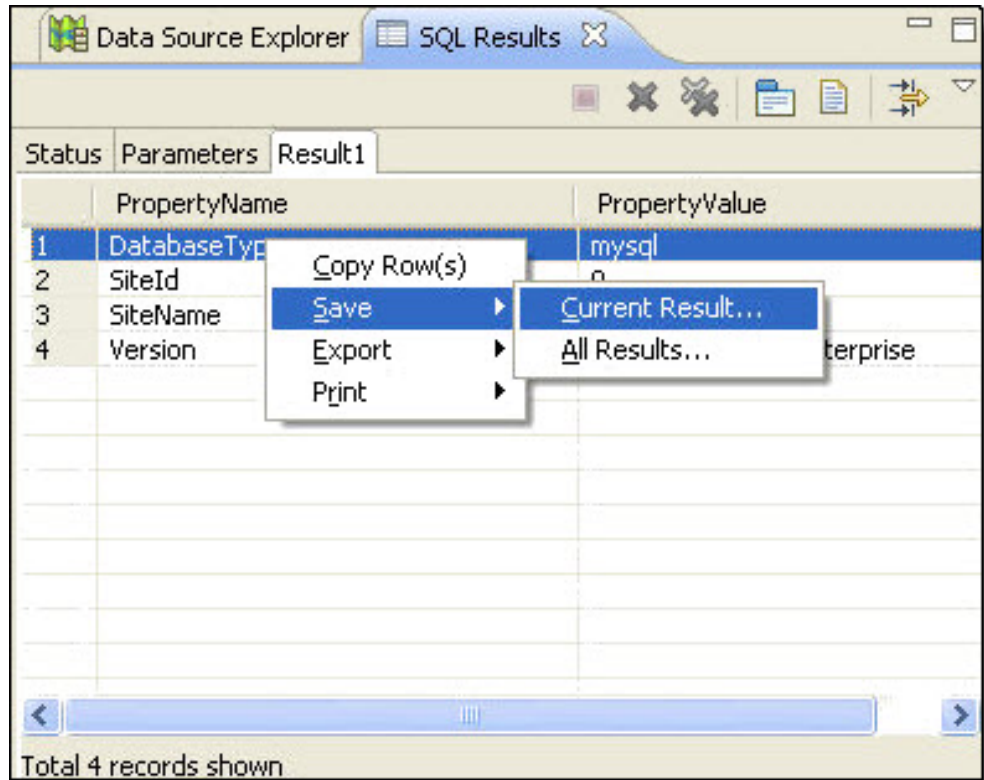
1. Data Source Explorer で、データベースを右クリックし、必要に応じて [接続] を選択します。ツリーを展開すると、データベース内のすべてのテーブルの概要が表示されます。
2. データベースを右クリックして、[Open SQL Scrapbook] を選択します。
3. [名前] リストからデータベースを選択します。
4. [Data Source Explorer] ウィンドウと [SQL スクラップブック] ウィンドウは、次の例のようになります。



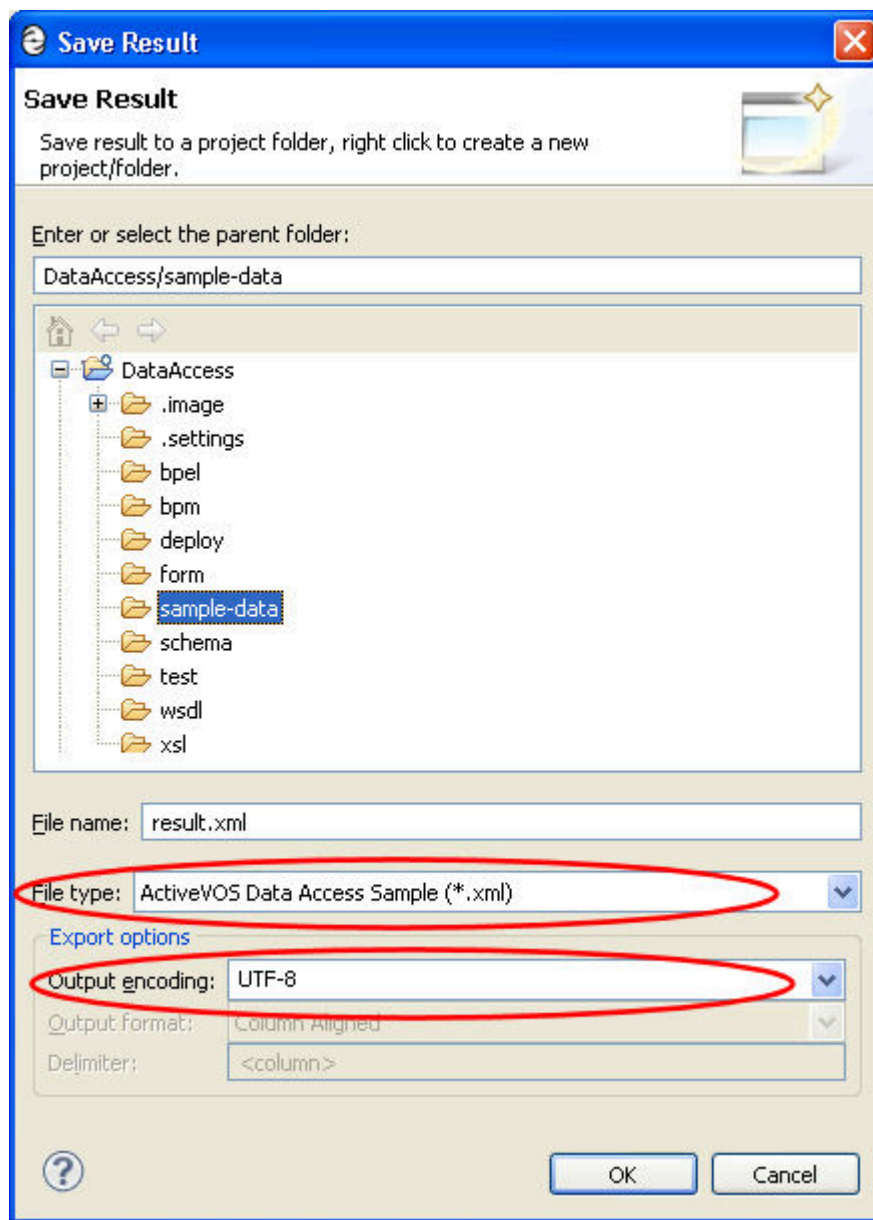
5. データアクセスの呼び出しが execSQL 操作に基づく場合は、単一の実行ステートメントをコピーして SQL スクラップブックに貼り付けます。
6. マウスを右クリックして、[Execute Selected Text] を選択します。[SQL 結果] タブに結果が返されることに注意してください。
7. execMultiSQL 操作の場合は、ステートメントをコピーして SQL スクラップブックに貼り付けます。マウスを右クリックして、[すべて実行] を選択します。

手順 3. 次のように、SQL 結果をサンプルデータとして保存します。

1. [SQL 結果] ビューで、マウスを右クリックし、[保存] > [現在の結果] を選択します。
単一のステートメントの実行に対しては現在の結果を選択し、複数のステートメントの実行に対してはすべての結果を選択する必要があります。データの形式は、選択した内容に応じて異なります。



2. [結果を保存] ダイアログで、ワークスペース内のデータアクセスプロジェクトを参照し、sample-data フォルダを選択します。ファイルに名前を付けて、この場所に保存します。
3. [ファイルの種類] ボックスで、[Process Developer データアクセスのサンプル] を選択します。次に、出力エンコーディングを [UTF-8] に設定します。次の例は、すべての設定を示しています。



4. [OK] を選択します。

例 - Insert Update または Delete ステートメントからの応答

非クエリステートメントから返されるデータはありません。この例に示すように、応答には操作の影響を受ける行数のみが含まれます。

```
<dataAccessResponse
  xmlns="http://schemas.active-endpoints.com/data-access/
    2010/04/data-access.xsd"
  statementId="statement-1">
  <row updatedRows="3"/>
</dataAccessResponse>
```

データアクセスサービスのシミュレーション

データアクセスサービスを含むプロセスのプロセスデプロイメント記述子を作成する場合は、データベースの場所を指定する必要があります。次のようにデプロイメント情報を入力します。

1. [パートナーリンク] ページで、データアクセス用のパートナーサービスプロバイダの呼び出しハンドラが自動的にシステムサービスに設定されることに注意してください。
2. クラウドプロセスを使用している場合は、編集フィールドで、テキスト `FILL_IN_DATASOURCE_JNDI` を次のいずれかに置き換えます。
 - データソースにバインドされているアプリケーションサーバーで定義された JNDI 名。JNDI プレフィックスを含める必要はありません。以下に例を示します。
`jdbc/myDatabase`
 - 同様に、WebLogic および Websphere の場合も、プレフィックスは必要ありません。
`jdbc:/myDatabase`
 - プロセスコンソールの [URN マッピング] ページで JNDI 名または URL にマッピング可能な URN。このオプションを使用すると、プロセスをリファクタリングおよび再デプロイせずにデータベース接続情報を変更できます。

注: オンプレミスでデプロイする場合は、次の例に示すように、JNDI プレフィックスも含める必要があります。

```
java:comp/env/jdbc/myDatabase
```

予備テストで、Process Developer に組み込まれたサーバー内でデータベース接続をセットアップする場合は、次の手順に従ってください。

1. Process Developer で、組み込みサーバーが実行されている場合は停止します。詳細については、[「組み込みプロセスサーバーのセットアップ」 \(ページ 96\)](#) を参照してください。
2. ファイルシステムの Process Developer インストールディレクトリで、組み込みサーバープラグインに移動します。例えば、`[デザイナのインストールディレクトリ]\designer\plugins\org.activebpel.enginep_[versionNNN]` (ここで、versionNNN はプラグインのバージョンです) など。
3. `org.activebpel.enginep` プラグインで、`server\conf\Catalina\localhost` に移動し、`active-bpel.xml` ファイルのバックアップコピーを作成します。次に、ファイルを開きます。
4. 次の例に示すように、`active-bpel.xml` ファイルで、`<Context>` 要素内に新しい `<Resource>` 要素を追加します。

```
<Resource
  driverClassName="driver_class_name"
  maxActive="100"
  maxIdle="30"
  maxWait="10000"
  name="jdbc/anyName"
  password="password"
  type="javax.sql.DataSource"
  url="url_of_your_db"
  username="db_login_name"/>
```

すべての属性値を、データベースに接続するための値に必ず置き換えてください。

5. ファイルを保存し、組み込みサーバーを再起動します。
6. BPR を組み込みサーバーにデプロイする前に、JNDI 名を必ずプロセスの PDD に追加してください。

電子メールサービス

プロセスサーバーは電子メールサービスをサポートしています。これは、Invoke アクティビティを介して受信者のリストに電子メールメッセージを送信するプロセスを作成できることを意味します。電子メール配信アクティビティは、[「警告サービス」 \(ページ 304\)](#) を使用するなど、さまざまな種類のプロセスで利用されます。

電子メールベースのアクティビティを実装する BPEL プロセスは、プロセスサーバーで提供される WSDL をインポートします。WSDL の名前は email.wsdl で、[パーティシパント] ビューウィザードでは、*send email* という名前のシステムサービスです。WSDL には、次のように Invoke アクティビティで使用可能な送信操作が含まれています。

```
<invoke
  inputVariable="emailMessage"
  operation="send"
  outputVariable="resultMessage"
  partnerLink="emailProvider"/>
```

電子メールを送信する BPEL プロセスを作成する手順

1. プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。
2. **[ファイル] > [新規] > [BPEL プロセス]** を選択します。
3. プロセスに名前を付けて、[完了] をクリックします。
4. [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
5. [システムサービス] を選択します。
6. **[電子メール]** を選択します。
7. [パーティシパント] ビューから、**[送信]** 操作をキャンバスにドラッグして、Invoke アクティビティを作成します。このアクティビティは、プロセスサーバーで有効になっている ID サービスのユーザーに電子メールメッセージを送信します。
8. プログラミングロジックを追加します。以下に例を示します。
 - a. 電子メールに関連付けられる入力を受信する Receive アクティビティを追加します。
 - b. 呼び出しの入力タブを使用して、変数にデータを割り当てます。以下の例を参照してください。
9. 送信操作のフォールトメッセージを使用して、電子メールサービスとの通信エラーをキャッチします。

emailMessage 変数を初期化する例:

1. 呼び出しの **[入力]** タブで、[割り当てタイプ] として [XPath] を選択します。
2. **[追加]** を選択します。
3. **[E/L]** パネルで、[リテラル] を選択します。
4. [From] カラムで、ダイアログボタンを選択し、リテラルコンテンツを生成します。Process Developer は、選択した単一パートの要素ベースの変数に対する正しいスキーマを自動的に見つけ、次のようなリテラルテンプレートを生成します。

生成されたテンプレート

```
<aem:emailMessage xmlns:aem="http://schemas.active-endpoints.com/email/2007/01/email.xsd">
  <aem:from>string</aem:from>
  <aem:replyTo>string</aem:replyTo>
  <aem:to>string</aem:to>
  <aem:cc>string</aem:cc>
  <aem:bcc>string</aem:bcc>
  <aem:subject>string</aem:subject>
  <aem:body mimeType="string"></aem:body>
</aem:emailMessage>
```

エラーを回避するために、必ず mimeType を *text/plain* などの有効なタイプに設定してください。手順 5 の例を参照してください。

5. 次のヒントのいずれかを使用して、emailMessage 要素に適切な文字列を入力します。
 - 要素ごとに 1 つずつ、XPath 式を作成します。例えば、<aem:to>要素にデータを入力するには、FROM\$quoteRequest/quote:contact/quote:name() TO aem:to などの式を作成します。
 - doXsltTransform 関数を使用して式を作成します。この方法では、電子メール要素のすべての変換の詳細を含む XSL ファイルを作成する必要があります。例: doXsltTransform ('monitorEmail.xsl', \$handleAlertRequest)

- 割り当てタイプを XQuery に設定します。次に、{expressions}または文字列をそれぞれの emailMessage 要素に直接追加します。

例

```
<ns:emailMessage
  xmlns:ns="http://schemas.active-endpoints.com/email/2007/01/email.xsd">
  <ns:from>Dave Verd &lt; DVerd@mycompany.com &gt;</ns:from>
  <ns:replyTo>sender@mycompany.org</ns:replyTo>
  <ns:to>{$quoteRequest/quote:contact/quote:name()}</ns:to>
  <ns:subject>{ $quoteRequest/quote:contact/quote:description}</ns:subject>
  <ns:body mimeType="plain/text">Message goes here.</ns:body>
</ns:emailMessage>
```

電子メールサービスとしてのプロセスのデプロイ

プロセスをデプロイし、マイロールパートナーリンクの標準ドキュメントリテラルサービスとして利用できるようことができます。BPR には、WSDL や XSD はありません。これらはすでにデプロイされており、サーバー上で利用できます。

メーリングリストに無効なアドレスが含まれている場合、プロセスによってエラーがログに記録されます。電子メールエラーを表示するには、プロセスコンソールのサーバーログを確認してください。

ID サービス

ID サービスは、定義した属性のセットに基づいてユーザーとグループを検索する方法を提供します。プロセスサーバーでの ID サービスのサポートは、Lightweight Directory Access Protocol (LDAP)、JDBC、またはファイルベースのサービスに基づいています。

ID ベースのアクティビティを含むプロセスを作成できます。前提条件として、プロセスコンソールで、ディレクトリサービスにアクセスするための通信の詳細を指定する必要があります。プロセスを実行すると、プロセスで指定されたユーザーまたはグループがディレクトリサービスで検索されます。

ID ベースのアクティビティを実装した BPEL プロセスは、プロセスサーバーによって提供される WSDL をインポートします。WSDL の名前は identity.wSDL で、[パーティシパント] ビューウィザードでは、*identity search* という名前のシステムサービスです。

WSDL には、次の操作が含まれています。

操作名	説明
findRolesByPrincipal	名前付きプリンシパルのロールのリストを返します（例えば、 <i>User1</i> は <i>Administration</i> および <i>Finance</i> のメンバーなど）。
findRoles	ディレクトリサービスで定義された <i>Marketing</i> 、 <i>Finance</i> 、 <i>DnsAdmins</i> などのロールのリストを返します。
findIdentitiesByRole	名前付きロールの ID のリストを返します

操作名	説明
findIdentities	<p>ユーザー名と電子メールアドレスを含む ID のリストを返します</p> <p>結果に含めたり、結果から除外したりするロールとプリンシパルを指定します</p> <p>クエリは、include ステートメントと exclude ステートメントで構成されます。結果セットには、include 要素の下にリストされたロールまたはプリンシパルが含まれ、exclude 要素の下にリストされたものは除外されます。</p>
assertPrincipalInQueryResult	<p>指定したプリンシパルについて、クエリの最終結果にユーザーが存在することを確認します。</p> <p>この操作は、フォールト処理に使用できます。権限を確認する場合に適しています。</p>
assertPrincipalInQueryResultwithResponse	上記と同じ働きを持ち、応答が含まれます
countIdentities	要求したユーザーとグループの数を受け取ります

アイデンティティベースのアクティビティの構築

警告サービスで ID を使用して、例えば、キャッチされないフォールトでプロセスが一時停止された場合に電子メールをグループに送信します。

BPEL プロセスでのブランチ処理用に、グループに基づいて ID を使用することもできます（例えば、マネージャ用のアクティビティのブランチと、顧客サービス担当者用のアクティビティのブランチを作成するなど）。

ID サービスを呼び出すために必要な Assign アクティビティを作成します。aeid:identityQuery 要素に基づいて、変数のリテラルコンテンツを生成できます。

次に例を示します。

```
<aeid:IdentityQuery xmlns:aeid="http://schemas.active-endpoints.com/identity/2007/01/identity.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="aeid:IdentityQuery">
<aeid:include>
<aeid:group>Development</aeid:group>
<aeid:user>user1</aeid:user>
<aeid:id>CN=Kim Pan,CN=Users,DC=aedomain,DC=active-endpoints,DC=local</aeid:id>
</aeid:include>
</aeid:IdentityQuery>
```

上記の<aeid:id>要素については、LDAP ディレクトリ内のユーザーを識別名（DN）で検索する例を示していることに注意してください。JDBC の場合、ルックアップはデータベースのプライマリーキーです。tomcat-users.xml の場合、ID はユーザー名と同じです。

必要に応じて、他のプログラミングロジックを追加します。

PDD では、パートナーロールに対して、カスタム呼び出しハンドラとしてシステムサービスが事前に選択されています。PDD エントリは次のようになります。

```
<partnerLink name="provider">
  <partnerRole endpointReference="dynamic"
    invokeHandler="system"/>
</partnerLink>
```

ID サービスとしてのプロセスのデプロイ

プロセスをデプロイし、マイロールパートナーリンクの標準ドキュメントリテラルサービスとして利用できるようにすることができます。BPR には、WSDL や XSD はありません。これらはすでにデプロイされており、サーバー上で利用できます。

移行サービス

移行サービスを使用すると、実行中のプロセスインスタンスを新しい（または異なる）バージョンのプロセス定義へ移行する際に詳細な制御が可能になります。実行中のプロセスを移行する最も簡単な方法として、[「全般的なデプロイメントオプション」](#)（ページ 457）ページにあるように、デプロイした新しいバージョンのプロセスデプロイメント記述子で移行を指定することが挙げられます。

ただし、この新しいプロセスバージョンに新しいアクティビティ、変数、パーティシパントなどの重要な変更がある場合は、移行する 1 つまたは複数のプロセスインスタンスを選択することで、移行をテストする移行サービスを作成できます。次に、サーバーの「アクティブなプロセスの詳細」ページで結果を分析します。エラーが発生した場合、プロセスは一時停止され、サーバーログにエラーが報告されます。プロセスサーバーにより、既存のプランと新しいプランの間の受信および発信メッセージ構造への変更は検証されないことに注意してください。

移行サービスのもう 1 つの使用例としては、移行を詳細に制御することが挙げられます。例えば、インスタンスを同じプロセス定義のオンラインバージョンに移行する必要はありません。サーバー上にある他のプロセス定義に移行できます。また、移行を行う実行中のプロセスインスタンスのみを特定できます。

移行を処理するために作成するプロセスは、次の 2 つの操作を順番に使用する移行システムサービスを利用します。

- **【マップの作成】 操作**

この操作は、プロセスインスタンスが実行されているプロセス定義のプラン ID を入力として受け取ります。サーバー上の各プラン ID は一意であり、プロセス定義の各バージョンのプロパティです。すべてのプロセスインスタンスは、プラン ID に対して実行されます。

【マップの作成】 操作は、移行マップ XML ドキュメントを文字列として返します。マップにはプラン間の移行構造が含まれており、これには、あるプロセス定義から別のプロセス定義へのアクティビティ、リンク、および変数の変換が記述されています。

- **移行操作**

この操作は、移行マップと、移行する実行中のプロセスインスタンスのプロセス ID (PID) を入力として受け取ります。

移行操作は、移行の警告のリスト（存在する場合）とバックアッププロセス ID を含む XML ドキュメントを返します。デフォルトでは、プロセスインスタンスは同じプロセス定義のオンラインバージョンのプラン ID に移行されますが、サーバー上の任意のプラン ID を指定することもできます。

マップ作成の入力と出力

マップの作成操作の入力は、次の例のようになります。実行中のプロセスインスタンスを移行するには、プラン ID を指定する必要があります。この操作にデータを渡すプロセス要求メッセージについては、「アクティブなプロセス詳細」ページでプラン ID を検索するか、データベースクエリや適切な管理 API 呼び出しを使用するなど、さまざまな方法でプログラマ的に取得できます。

次の例は、サンプルデータを示しています。toPlanId はオプションであることに注意してください。削除した場合、デフォルトの動作では、同じプロセス定義のオンラインバージョンにマッピングされます。

```
<migration:createMapRequest xmlns:migration="http://docs.active-endpoints/wsd/migration/2011/04/migration.xsd">
  <migration:fromPlanId>144</migration:fromPlanId>
  <!--Optional. If not used, defaults to the online plan of the pid:-->
  <migration:toPlanId>145</migration:toPlanId>
</migration:createMapRequest>
```

次の例は、応答用に作成された移行マップの最初の部分を示しています。

```
<createMapResponse xmlns="http://docs.active-endpoints/wsd/migration/2011/04/migration.xsd">
  <migrationMap><![CDATA[
    <processMigration xmlns="http://schemas.activevos.com/processMigration/2010/05/processMigrationMapper.xsd">
```

```

    <migrationInfo>
      <locationInfo>
        <locationId>1</locationId>
        <locationPath>/process</locationPath>
        <idsToRoot />
      </locationInfo>
      <mapForVariable>false</mapForVariable>
      <migrationMapper>
        <oldLocationInfo>
          <locationId>1</locationId>
          <locationPath>/process</locationPath>
          <idsToRoot />
        </oldLocationInfo>
        <activityStrategy>
          <stateMapType>0</stateMapType>
          <defaultLocationInfo>
            <locationId>1</locationId>
            <locationPath>/process</locationPath>
            <idsToRoot />
          </defaultLocationInfo>
        </activityStrategy>
      </migrationMapper>
    </migrationInfo>
  ...
</createMapResponse>

```

移行のための入力と出力

移行操作の入力には PID が必要です。この操作にデータを渡すプロセス要求メッセージについては、アクティブプロセスリストで PID を検索するか、プログラマ的に取得できます。基本的に、getProcessList API 呼び出し（またはレポート）を使用して、移行するプロセス ID のセットに一致するプロセス ID のセットを戻します。移行する数が多い場合は、多数のインスタンスを並行して移行できます。

次の例は、サンプルデータを示しています。

```

<migration:migrateRequest xmlns:migration="http://docs.active-endpoints/wsd/migration/2011/04/migration.xsd">
  <!--The string comes from the Create Map operation: -->
  <migration:migrationMap>string</migration:migrationMap>
  <migration:pid>850</migration:pid>
  <!--Optional. If not used, defaults to the online plan of the pid. If used, must match with toPlan in
  migrate operation:-->
  <migration:toPlanId>145</migration:toPlanId>
  <!--Optional. Defaults to false:-->
  <migration:createBackup>false</migration:createBackup>
</migration:migrateRequest>

```

以下に、移行応答メッセージの例を示します。

```

<migrateResponse xmlns="http://docs.active-endpoints/wsd/migration/2011/04/migration.xsd">
  <response>MIGRATION_WARNING Errorcode:3001. Migration Warning: The empty activity is being marked as
  Finished when it is not. This might result in invalid output for the pid 850. (location path /process/
  sequence/empty)</response>
  <backupPid>851</backupPid>
</migrateResponse>

```

プロセスインスタンスを新しいバージョンに移行する BPEL プロセスの作成

次の手順に従います。

1. プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。
2. **【ファイル】 > 【新規】 > 【BPEL プロセス】** を選択します。
3. プロセスに名前を付けて、**【完了】** をクリックします。

4. 作成した WSDL に基づいて Receive/Reply を作成します。プロセスの要求メッセージには、実行中のプロセスインスタンスのプラン ID と 1 つ以上のプロセス ID が含まれます。
5. [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
6. [システムサービス] を選択します。
7. [移行] を選択します。
8. [パーティシパント] ビューから、[マップの作成] 操作をキャンバスにドラッグして、Invoke アクティビティを作成します。このアクティビティは、プロセスプラン ID を入力として受け取り、プロセス構造のマップを作成します。
9. [パーティシパント] ビューから、[移行] 操作をマップ作成呼び出しの下にドラッグして、2 番目の Invoke アクティビティを作成します。このアクティビティは、移行マップとプロセス ID を入力として受け取り、デフォルトで、実行中のプロセスインスタンスを同じプラン ID のオンラインバージョンに移行させます。プロセスの構造に応じて、任意のプラン ID を指定することができます。

移行プロセスのテスト

移行プロセスをサーバーにデプロイした後に、次のようにテストを実行できます。

1. プロセスをデプロイし、実行時間の長いプロセスインスタンスをインスタンス化します。
2. 2 番目のバージョンのプロセスをデプロイします。PDD では移行オプションを選択しないでください。
3. 元のバージョンのプラン ID と実行中のプロセスインスタンスの PID を含む要求メッセージを使用して、移行プロセスをインスタンス化します。

BPEL プロセスの外部でプログラマ的に移行を行う方法については、[「プロセスサーバー移行 Web サービスの使用」 \(ページ 317\)](#)を参照してください。

プロセスサーバー移行 Web サービスの使用

BPEL プロセスの外部でプログラマ的に移行を行う場合は、プロセスサーバー SOAP ベースの移行 Web サービスを使用できます。

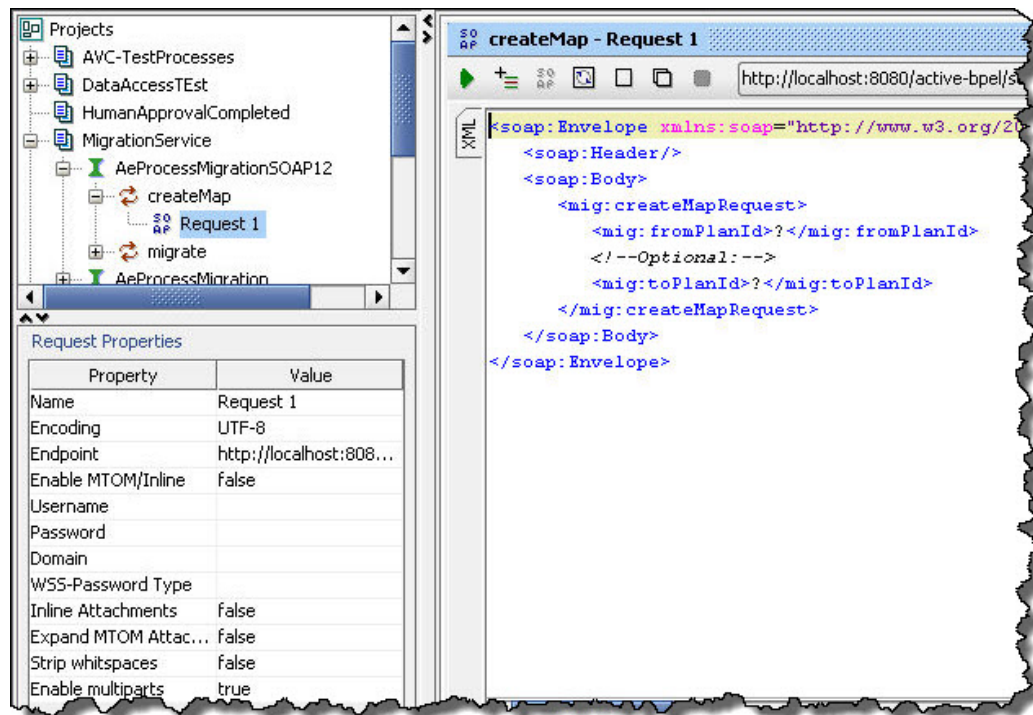
サーバーで以下のページを開きます。

`http://localhost:8080/active-bpel/services`

このページで下にスクロールして、次の AeProcessMigrationService を見つけます。

`http://localhost:8080/active-bpel/services/AeProcessMigrationService?wsdl`

次の例に示すように、SOAP UI などのツールを使用して、1 つのプロセスインスタンスを移行する要求を作成します。



シェルコマンドを呼び出す BPEL プロセスの作成

1. プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。
2. [ファイル] > [新規] > [BPEL プロセス] を選択します。
3. プロセスに名前を付けて、[完了] をクリックします。
4. [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
5. [インタフェース] ツリーで、[システムサービス] を選択します。
6. [シェルコマンド] を選択します。
7. 作成したパートナーサービスプロバイダから **execCommand** 操作をキャンバスにドラッグして、Invoke アクティビティを作成します。
8. 上記の選択を入力データに入力します。

詳細については、「[コネクタ] > [シェルコマンド呼び出しサービスコネクタ]」を参照してください。

BPEL プロセスを使用した警告の監視サービスの使用

BPEL プロセスに送信される入力メッセージには、次の例に示すようなパートがあります。

```
<ns:handleAlertRequest xmlns:ns="http://www.activebpel.org/monitor/2007/08/monitorAlert.wsdl">
  <ns:engineDetail>
    <ns:name>string</ns:name>
    <ns:state>created</ns:state>
  </ns:engineDetail>
</ns:handleAlertRequest>
```

```

<ns:monitoringStatus>normal</ns:monitoringStatus>
<ns:errorMessage>string</ns:errorMessage>
<ns:startTime>2007-09-10T19:15:15.868Z
  </ns:startTime>
<ns:id>1</ns:id>
<ns:numCPU>1</ns:numCPU>
<ns:deploymentGroupId>1</ns:deploymentGroupId>
<ns:deploymentGroup>string</ns:deploymentGroup>
</ns:engineDetail>
<ns:troubleItems>
  <ns:troubleItem>
    <ns:monitorName>string</ns:monitorName>
    <ns:level>1</ns:level>
    <ns:statistic>avg</ns:statistic>
    <ns:operator>eq</ns:operator>
    <ns:threshold>1.0</ns:threshold>
    <ns:displayName>string</ns:displayName>
    <ns:value>1.0</ns:value>
    <ns:timestamp>2007-09-10T19:15:15.868Z</ns:timestamp>
  </ns:troubleItem>
</ns:troubleItems>
</ns:handleAlertRequest>

```

警告の監視 BPEL プロセスを作成するときに考慮すべき一般的なステップを示します。

警告の監視 BPEL プロセスの作成手順

1. Project Explorer で、新しいプロジェクトを作成します。
2. [ファイル] > [新規] > [BPEL プロセス] を選択します。
3. プロセスに名前を付けて、[次へ] をクリックします。
4. [プロセステンプレート] ページで、警告の監視のプロセステンプレートを選択します。
5. ReceiveAlertMessage アクティビティが存在する新しいファイルが Process Editor キャンバスで開きます。Receive アクティビティが handleAlert 操作に基づいているのです。
6. 監視対象のプロパティが警告またはエラーレベルに到達したときに発生させるプログラムロジックを追加して、プロセスの作成を完了します。
7. [デプロイメント記述子] ウィザードを起動して、[パートナーリンク] ページで次を実行します。
 - a. MonitorAlertPL マイロールのパートナーリンクを選択します。
 - b. マイロールのパートナーリンクのサービス名を入力します。
8. プロセスサーバーを起動して、サーバーにプロセスをデプロイします。BPR には、WSDL や XSD はありません。これらは既にデプロイされ、サーバー上で利用できます。
9. プロセスコンソールで、警告の監視サービスを追加します。詳細については、プロセスコンソールのヘルプを参照してください。

BPEL プロセスを使用した再試行ポリシーサービスの使用

再試行ポリシー BPEL プロセスに送信された入力メッセージには、次のパートがあります。

```

<retryCheckInput
  xmlns="http://www.activebpel.org/services/retry">
  <faultName>{namespace}localpart</faultName>
  <processId>process id</processId>
  <processName>process name</processName>
  <invokePath>path of activity which is faulted</invokePath>
  <attempts>retry attempts already executed</attempts>

```

```

    <partnerLinkName>partner link used by invoke
  </partnerLinkName>
  <partnerLink>endpoint reference of partner link
</partnerLink>
</retryCheckInput>

```

記述したプロセスは、次のパートを付加してサーバーにメッセージを返送します。

```

<retryCheckOutput
  xmlns=http://www.activebpel.org/services/retry>
  <retry>yes or no indicating whether or not to retry</retry>
  <interval>number of seconds to delay if retrying</interval>
</retryCheckOutput>

```

返送するメッセージには、試用する代替サービスが含まれることもあります。サーバーに返送するメッセージは、次のようになります。

```

<retryCheckOutput
  xmlns=http://www.activebpel.org/services/retry>
  <retry>yes or no indicating whether or not to retry</retry>
  <interval>number of seconds to delay if retrying</interval>
  <wsa:EndpointReference xmlns="http://
    schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:ext="http://www.activebpel.org/bpel/extension"
    xmlns:ns1="http://www.activebpel.org/services
      /retrytests"
    xmlns:s="http://www.activebpel.org/
      services/retrytests"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03
      /addressing"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03
      /business-process/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsa:Address>ae:localhost:alternateRetryService
  </wsa:Address>
  <wsa:ServiceName PortName=
    "retryTesterServicePort">s:retryTesterService
  </wsa:ServiceName>
  </wsa:EndpointReference
</retryCheckOutput>

```

プロセスでは、システムのダウンタイムの説明など、その他多数のプログラマ的なタスクを実行できます。

システムのダウンタイム

サービスでは、メンテナンスや予定外のイベントによりダウンタイムが発生する場合があります。どちらのケースでも、再試行ポリシーサービスを使用して、BPEL アクティビティでプログラマ的に呼び出しを再スケジュールすることが適切である場合があります。

再試行ポリシーサービスを作成するときに考慮すべき一般的なステップを示します。

再試行ポリシーサービスとして使用するための BPEL プロセスの作成手順

1. Project Explorer で、新しいプロジェクトを作成します。
2. **【ファイル】 > 【新規】 > 【BPEL プロセス】** を選択します。
3. プロセスに名前を付けて、**【次へ】** をクリックします。
4. **【プロセステンプレート】** ページで、再試行ポリシーのプロセステンプレートを選択します。
5. *ReceiveRetryCheck* アクティビティが存在する新しいファイルが Process Editor キャンバスで開きます。Receive アクティビティが *retryCheck* 操作に基づいているのです。
6. 必要に応じた Assign アクティビティでデフォルトの再試行および間隔のデータを置き換えて、プロセスの作成を完了します。

7. [プロセスデプロイメント記述子] ウィザードを起動して、[パートナーリンク] ページで次を実行します。
 - a. マイロールのパートナーリンクを選択します。
 - b. マイロールのパートナーリンクのサービス名を入力します。
8. プロセスサーバーを起動して、サーバーに再試行ポリシープロセスをデプロイします。BPR には、WSDL や XSD はありません。これらは既にデプロイされ、サーバー上で利用できます。
9. 再試行ポリシーサービス名は覚えておく必要があります。再試行ポリシー BPEL プロセスの対象となる Invoke アクティビティを含むプロセスの PDD ファイルでは、再試行ポリシーのアサーションで再試行ポリシーサービス名を参照する必要があるためです。[再試行ポリシーのアサーション] ダイアログでは、次の例の太字に示すように、再試行ポリシープロセスの QName を指定するようにします。

```
<wsa:EndpointReference
  xmlns:s="http://www.activebpel.org/services/retrytests">
  <wsa:ServiceName PortName="retrytesterServicePort">
    s:retryTesterService</wsa:ServiceName>
  <wsp:Policy
    xmlns:wsp="http://schemas.xmlsoap.org/ws/
      2002/12/policy"
    xmlns:abp="http://www.activebpel.org/policies">
    <abp:retry service="retryCheckerService"
      faultList="{http://xml.apache.org/axis/}*
        {http://www.active-endpoints.com/2004/06/bpel
          /extensions/}*
        {http://www.activebpel.org/services
          /retrytests}*">
    </wsp:Policy>
  </wsa:EndpointReference>
```

10. このプロセスをサーバーにデプロイして、リモートデバッグを使用することにより、呼び出しが失敗して再試行ポリシーサービスを実行する場合のテストをします。

BPEL プロセスを使用したサービスログサービスの使用

プロセスサーバーログと対話する BPEL プロセスの作成手順

1. Project Explorer で、新しいオーケストレーションプロジェクトを作成します。
2. [ファイル] > [新規] > [BPEL プロセス] を選択します。
3. プロセスに名前を付けて、[完了] をクリックします。
4. [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
5. [インタフェース] ツリーで、[システムサービス] を選択します。
6. [サーバーロギング] を選択します。
7. 作成したパートナーサービスプロバイダから送信操作をキャンバスにドラッグして、Invoke アクティビティを作成します。
8. 上記の選択内容を使用して、入力データのリテラルを生成します。

詳細情報

システムサービスの詳細については、「設計」の「システムサービス、リスナ、およびコネクタの使用」を参照してください。

第 19 章

マイロールのエンドポイントおよびプロセスコンシューマ

プロセスコンシューマのサービスエンドポイントを使用すると、新しいプロセスインスタンスを作成するメッセージ、または既存のプロセスインスタンスにルーティングされるメッセージをプロセスサーバーに送信できます。BPEL の用語では、エンドポイントはマイロールサービスエンドポイントと呼ばれます。Process Developer では、マイロールエンドポイントはプロセスコンシューマと呼ばれます。

サービスエンドポイントのデプロイメントオプションは、プロセスデプロイメント記述子（PDD）内の myRole 要素として構成され、ビジネスプロセスアーカイブ（.bpr）をデプロイする際に Process Developer によって自動的に公開されます。

各サービスは、SOAP Web サービスまたは REST-ful HTTP エンドポイントとして、あるいは内部クライアントのみがアクセス可能な非公開サービスとしてデプロイすることができます。プロセスサーバーは、myRole サービスのバインディング属性を使用して、エンドポイントのデプロイ方法およびエンドポイントをアクセス可能にする方法を決定します。

プロセスデプロイメント記述子を作成する際に、myRole サービスの標準バインディングの 1 つを選択できます。また、必要に応じて、Java 実装を作成することもできます。

SOAP とポリシー駆動型バインディング

プロセスデプロイメント記述子（PDD）を作成する場合に、マイロールサービスの標準バインディングの 1 つを選択できます。この選択肢は次のとおりです。

- SOAP バインディング
- ポリシー駆動型バインディング
- パブリッシュ解除済み

SOAP バインディング

サービスエンドポイントを Web サービスとしてデプロイする最も簡単な方法としては、次のような標準の SOAP バインディングの 1 つを選択することが挙げられます。

- ドキュメントリテラル
- RPC リテラル
- エンコードされた RPC

この場合、バインディングスタイルとサービス名を選択するだけで済みます。選択したスタイルの SOAP バインディング要素は、PDD の myRole 要素に関連付けられたポートタイプとポリシーに基づいて生成されます。

デプロイメント後、プロセスサーバーは、サービスエンドポイント用に指定したメッセージの受信と応答を処理します。各メッセージは、バインディングに基づいて、プロセスサーバーによってフォーマットおよび処理されます。

サービスは、エンジンの内部 Web サービスプロバイダにデプロイされます。Web サービスは、Metro OAP スタックによってサポートされています（Metro は、Sun が開発した Glassfish プロジェクトのコアコンポーネントです）。

SOAP1.1 エンドポイントおよび SOAP 1.2 エンドポイントが、JAX-WS に準拠したサービス（ドキュメントリテラルと RPC リテラル）にデプロイされます。エンコードされた RPC などのレガシーバインディングは、SOAP 1.1 としてのみデプロイされます。

要求を受信すると、URL によってターゲットサービスと SOAP バージョンが決定されます。パスの最後のセグメントは、次のように PDD で指定されたサービス名に対応します。

- SOAP 1.1 の場合、URL は次のようになります。
`http://host:port/active-bpel/services/<MyRoleServiceNameInPDD>`
- SOAP 1.2 の場合、URL は次のようになります。
`http://host:port/active-bpel/services/soap12/<MyRoleServiceNameInPDD>`

デプロイした Web サービスエンドポイントの WSDL には、次の場所からアクセスできます。

`http://host:port/active-bpel/services/<MyRoleServiceNameInPDD>?wsdl`

プロセスサーバーエンジン内でホストされているすべてのデプロイ済みの Web サービスのリストは、次の URL で確認できます。

`http://host:port/active-bpel/services`

プロセスコンソールで、[サービス定義] リンクを選択します。

サービス用にパブリッシュされた WSDL には、WS-Addressing や WS-Security などのポリシーアサーション用の標準の WS-Policy 添付ファイルが含まれています。

ポリシー駆動型バインディング

ポリシー駆動型バインディングは、myRole 要素で定義されたポリシーアサーションを使用してサービスを公開する方法を定義します。

さまざまなポリシーアサーションがありますが、バインディングとして使用できるのは次の 3 つだけです。

- **参照先の WSDL バインディング:** 特定のスタイルに基づいてエンジンで SOAP バインディングを生成する代わりに、PDD の参照先の WSDL バインディングポリシーアサーションを使用して、特定の SOAP バインディングを WSDL で参照できます。この機能を使用して、SOAP ヘッダーとしてマッピングされたメッセージパートを使用してサービスをデプロイするか、空でない soapAction 属性を指定します。
- **WS-Reliable メッセージング:** WSRM ポリシーの存在は、WS-Reliable メッセージプロトコルをサポートする Web サービスとしてサービスがデプロイされていることを示します。
- **REST:** REST ポリシーが使用されている場合、REST-ful サービスエンドポイントは、URL を使用して HTTP クライアントに公開されます。

ポリシーアサーションの詳細については、「エンドポイント参照と WS-Policy」を参照してください。

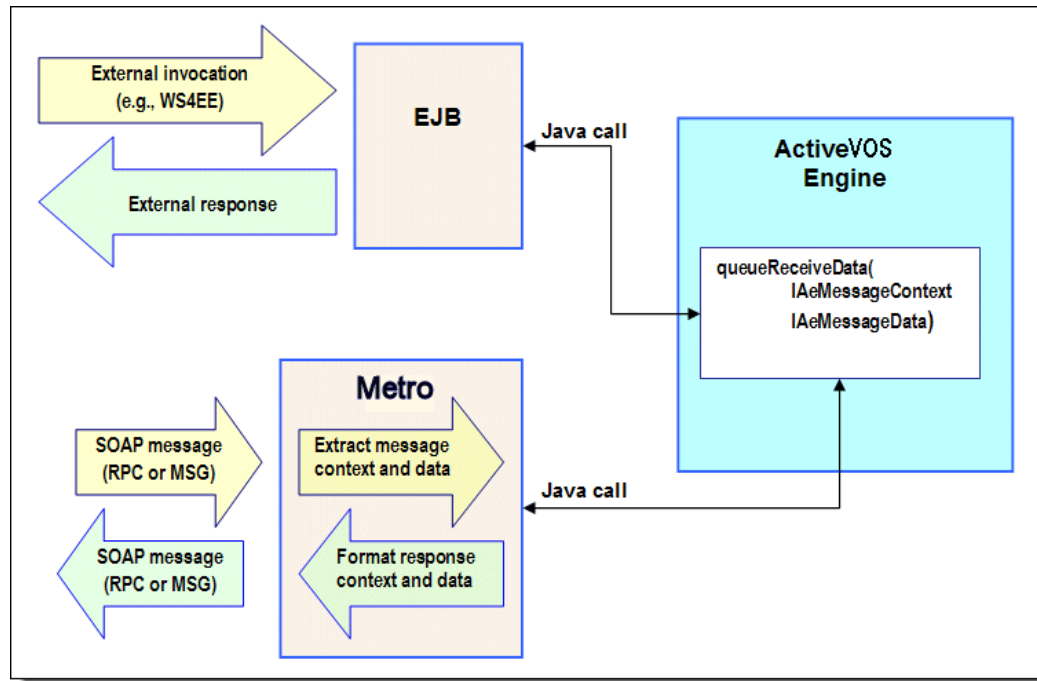
パブリッシュ解除済みのバインディングスタイル

[パブリッシュ解除済み] がバインディングとして選択されている場合、エンドポイントは SOAP スタックに公開されず、プロセス呼び出しを使用するか、カスタム Java クライアントを介してのみ内部クライアントにアクセスできます。詳細については、「Java からのプロセスの呼び出し」を参照してください。

Java からのプロセスの呼び出し

プロセスサーバーは、メッセージをエンジンに直接ディスパッチするための API を提供します。この API は、EJB をサポートするプラットフォームでは Enterprise Java Bean (EJB) として、または、EJB コンテナのないデプロイメントプラットフォームではプレーン JavaAPI として利用できます。

次の図は、マイロールのサービスエンドポイントの外部実装と標準実装の概要を示しています。



プロセスサーバーエンジンの Enterprise Java Bean (EJB) バインディングは、次のインタフェースを実装しています。

```
public interface org.activebpel.wsio.receive.IAeMessageQueue
{
    /**
     * Delivers the message to the BPEL engine's message queue.
     *
     * @param aData
     * @param aContext
     * @return IAeWebServiceResponse
     * @throws RemoteException
     * @throws AeRequestException
     */
    queueReceiveData
    (org.activebpel.wsio.IAeWebServiceMessageData aData,
     org.activebpel.wsio.receive.IAeMessageContext aContext)
    throws RemoteException,
     org.activebpel.wsio.receive.AeRequestException;
}
```

ここで、

- IAeMessageQueue はインタフェースの名前です。パッケージは org.activebpel.wsio.receive です。
- IAeWebServiceResponse は、呼び出し元のサービスから返される可能性のあるデータまたはフォールトのインタフェースです。
- queueReceiveData は、コンテキストとデータの受け渡しをサービスエンドポイントが呼び出すことができるメソッドです。

- IAeWebServiceMessageData には、受信メッセージのデータが含まれています。
- IAeMessageContext には、要求のコンテキスト情報が含まれています。

プレーン Java または EJB API を使用してプロセスを呼び出す手順

手順 1: Java プロジェクトのクラスパスに必要な.jar ファイルを含める

API は、次の.jar ファイルに含まれています。プレーン Java API を使用する場合は、ae_wsio.jar のみが必要です。また、EJB 実装を使用する場合は、クラスパスに両方の API が必要となります。

- ae_wsio.jar には、メッセージをエンジンに送信するために使用されるコアインタフェースが含まれています。
- ae_awfwsio.jar には、プロセスサーバーエンジンの EJB とデータのやり取りをするために必要なインタフェースが含まれています。

手順 2: EJBHome を検索する

Java コードで、JNDI コンテキストから AeMessageQueueBean に対する EJBHome を検索し、Bean のインスタンスを作成します。以下に例を示します。

```
InitialContext ctx = new InitialContext();
object home = ctx.lookup("ejb/AeMessageQueueBean");
Home = (AeMessageQueueHome)
portableRemoteObject.narrow(home, AeMessageQueueHome.class);
eMessageQueueRemote msgQueueBean = mHome.create();
```

手順 3: AeMessageContext のインスタンスを作成する

AeMessageContext オブジェクトには、BPEL プロセスからのマイロールパートナーリンクに関する詳細が含まれています。これらの詳細は、エンジンに送信されるインバウンドメッセージのコンテキストを構成します。これらの詳細を以下に示します。

AeMessageContext のパラメータ	説明
setProcessName	Qname を処理します。Qname には、名前空間とプロセス名のローカル部分が含まれます。BPEL ソースファイルでは、これはターゲット名前空間です。 注: プロセス/パートナーリンク名の代わりにサービス名を指定できます。サービス名の説明については、 第 19 章, 「マイロールのエンドポイントおよびプロセスコンシューマ」 (ページ 323) を参照してください。
setPartnerLinkName	BPEL プロセスからのパートナーリンク名
setServiceName	オプション。プロセス名とパートナーリンク名を設定する代わりに、マイロール (プロセスコンシューマ) サービス名を指定できます。この名前は、[パートナーリンク] ページのプロセスデプロイメント記述子エディタ、およびプロセスコンソールの [サービス定義] ページに表示されています。 このプロパティはプロセス名と相互に排他的であり、コンテキストにはどちらか一方のみを設定できます。
setOperation	マイロール用に定義されたパートナーリンクタイプの portType からの操作
setProcessVersion	オプション。プロセスバージョンが指定されていない場合、メッセージは現在のプロセスに送信されます。
setPrincipal	オプション。この値は、「プリンシパル」エンドポイントタイプを使用する場合に、デプロイされたパートナー定義でルックアップを実行するためにエンジンで使用されます。インバウンドメッセージのプリンシパル名の値を識別します。また、ヒューマンタスクを含むプロセスも、この値を使用してプロセスイニシエータを識別します。

新しいメッセージコンテキストを作成するためのサンプルコード:

```
// Create the message context object for the request
// containing the routing information for the process
eMessageContext context = new AeMessageContext();
context.setProcessName(new QName(aProcessNamespace,aProcessName));
context.setPartnerLink(aPartnerLink);
context.setOperation(aOperation);
```

手順 4: AeWebServiceMessageData のインスタンスを作成する

コンテキストの詳細は次のとおりです。

AeWebServiceMessageData のパラメータ	説明
Constructor QName	コンストラクタで指定された QName は、入力メッセージに関連付けられた WSDL メッセージの QName に対応します。
setData	それぞれのメッセージパートは、このパラメータで定義されます。メッセージパートは、文字列、整数、または DOM ドキュメントオブジェクトとして渡される XML などのプリミティブ型である必要があります。複合型はドキュメントとして渡されます。XML ドキュメントは、複合型のスキーマ定義に準拠している必要があります。メッセージに相関プロパティが含まれる場合、エンジンはプロパティエイリアスを使用してこれらのプロパティを抽出し、メッセージを正しいプロセスインスタンスにルーティングします。

メッセージデータオブジェクトの WSDL メッセージ:

```
<wsdl:message name="creditInformationMessage">
  <wsdl:part name="part1" element="tns:LoanProcessRequest" />
  <wsdl:part name="part2" element="tns:LoanProcessAddress" />
</wsdl:message>
```

メッセージデータオブジェクトを作成するためのサンプルコード:

```
AeWebServiceMessageData data = new AeWebServiceMessageData (new
    QName("http://tempuri.org/services/loandefinitions","creditInformationMessage"));
data.setData("part1", domOfLoanProcessRequest);
data.setData("part2", domOfLoanProcessAddress);
```

Bean で queueReceiveData メソッドを呼び出します。

```
IAeWebServiceResponse response = msgQueueBean.queueReceiveData(data,context);
```

コンテキストの詳細は次のとおりです。

IAeWebServiceResponse のパラメータ	説明
getMessageData()	呼び出しから返されたメッセージデータ。詳細については、AeWebServiceMessageData の説明を参照してください。
isFaultResponse()	応答がフォールトをラップする場合は true を返します
getErrorCode()	エラーコード QName のアクセサ
getErrorMessage()	フォールトに関連するエラーメッセージを返し、フォールトがない場合は null を返します
getErrorDetail()	フォールトに関連するスタックトレースまたはその他の詳細情報を返し、フォールトがない場合は null を返します。

IAeWebServiceResponse のパラメータ	説明
getRootCause()	エラーコードの根本原因である Throwable を返します
getBusinessProcessProperties()	ビジネスプロセスから（文字列の）名前/値のペアのマッピングを返します

[匿名アクセスを許可] ポリシーの設定

パブリックコンテキストを介して受信要求を許可するには、PDD の「ユーザー定義」ポリシーとして `<abp:AllowAnonymousAccess/>` ポリシーを適用する必要があります。

例えば、wsse:UsernameToken を使用してサービスを認証するには、次のパブリックコンテキストを使用します。

`https://[host]/active-bpel/public/services/[tenant]/[ServiceName]`

アクセスを許可するには、PDD のマイロールのサービスエンドポイント（`[ServiceName]`）に次のポリシーを設定します。

```
{
  {
    <wsp:Policy
      xmlns:abp="http://schemas.active-endpoints.com/ws/2005/12/policy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <abp:AllowAnonymousAccess/>
      <abp:Authentication direction="in">
        <abp:UsernameToken/>
      </abp:Authentication>
    </wsp:Policy>
  }
}
```

第 20 章

カスタムサービスのインタラクション

次のトピックでは、カスタムサービスのインタラクションについて説明します。

- [「REST ベースのサービスの使用」 \(ページ 329\)](#)
- [「OAuth REST ベースのシステムサービスの使用」 \(ページ 334\)](#)
- [「Java Messaging Service 呼び出しハンドラの使用」 \(ページ 338\)](#)

REST ベースのサービスの使用

WSDL 操作ではなく、Representational State Transfer (REST) アーキテクチャに基づいてメッセージを処理可能な BPEL で、Web サービスインタラクションアクティビティを作成できます。これは、特定のタイプの BPEL プロセスを作成するための独自の WSDL ファイルが不必要であることを意味します。次の例のような要求を使用して、プロセスを簡単にインスタンス化することができます。

`http://localhost:8080/active-bpel/services/REST/lookUpWeather/mass/waltham`

REST は、World Wide Web などのシステムでリソースがどのように定義およびアドレス指定されるかを記述するためのアーキテクチャのスタイルです。REST を使用すると、SOAP などの追加のメッセージングレイヤを使用せずに、HTTP を介して REST ベースのメッセージを送受信できます。

また、BPEL プロセスからパートナー REST サービスにアクセスすることもできます。

REST は、クライアントとサービスが HTTP を介して通信を行い、リソースの表現を交換できるようにするためのものです。リソースとは、ネットワーク上の URL を介してアクセス可能な関心項目です。BPEL では、表現とは、異なるコンテンツタイプの添付ファイルをオプションとして含む XML ドキュメントです。

BPEL プロセスは、REST 要求を送受信します。REST 要求がプロセスに送信されると、プロセスサーバーはその要求を標準の WSDL メッセージに変換します。プロセスで REST サービスが呼び出されると、標準の WSDL メッセージが REST 要求に変換されます。応答とは、要求されたリソースを表すドキュメントです。返されるドキュメントタイプは、HTTP 応答の Content-Type ヘッダーによって定義されます。

REST の使用を開始するためには、次のものがが必要です。

- `aeREST.wsdl` と `aeREST.xsd` をインポートする BPEL プロジェクト
- 呼び出しには、次のものがが必要です。
 - アドレス指定する REST リソースの URL
 - 呼び出しの対象となる HTTP メソッド

- 返されるようにするリソース表現のコンテンツタイプ
- 返される XML コンテンツの場合は、BPEL プロセス内のコンテンツをアドレス指定するための、返されるメッセージ（例えば、Atom スキーマ、ベンダー固有のメッセージ）に対するオプションのスキーマ
- アドレス指定する REST サービスによって指定されたクエリパラメータ

関連事項:

- [「REST ベースの受信または呼び出しの作成」 \(ページ 330\)。](#)
- [「BPEL REST メッセージ」 \(ページ 332\)](#)
- [「マルチパート HTTP メッセージの処理」 \(ページ 333\)](#)
- [「REST ベースのプロセスのデプロイメントに関する詳細の指定」 \(ページ 333\)](#)
- [「I18N 関数」 \(ページ 284\)](#)

REST ベースの受信または呼び出しの作成

REST ベースのサービスでは、Web ブラウザまたは HTTP クライアントのアドレスフィールドを介して送信された URL 要求を実行する開始アクティビティ（Receive、Pick、イベントハンドラ）を作成できます。また、HTTP 経由で Web ベースのサービスに要求を送信する Invoke アクティビティを作成することもできます。

Receive および Invoke アクティビティは、aeREST.wsdl という名前の WSDL ファイルに基づいている必要があります。この WSDL のポートタイプとパートナーリンクタイプは、[パーティシパント] ビューの **【インタフェース】** ダイアログの [システムサービス] ツリーにあります。

基本的には、WSDL を使用して、REST 要求を送受信するための RequestResponse 操作とメッセージを定義します。Process Developer は、受信 REST 要求を WSDL メッセージに変換し、送信要求に対してはその逆の変換を行います。

REST ベースのプロセスを対象とする要求は次の例のようになります。

`http://localhost:8080/active-bpel/services/REST/lookUpWeather/mass/waltham`

REST 要求メッセージ

例に示すように、プロセスサーバーは、要求を変換して以下のようなパートを持つ入力メッセージにします。

RestRequest の例（すべての要素が示されているわけではありません。以下の表を参照してください）：

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:RESTRequest xmlns:ns2="http://schemas.activebpel.org
/REST/2007/12/01/aeREST.xsd">
  <ns2:subdomain>Search</ns2:subdomain>
  <ns2:method>GET</ns2:method>
  <ns2:pathInfo>V1</ns2:pathInfo>
  <ns2:params>
    <ns2:param name="temp" value="50"/>
    <ns2:param name="windchill" value="40"/>
  </ns2:params>
  <ns2:payload contentType="text/xml">lookUpWeather/mass/waltham?temp=50&windchill=40</ns2:payload>
</ns2:RESTRequest>
```

メッセージパートは次のように定義されています。

subdomain	パートナーサービス（呼び出し）のオプション。ドメインが提供する REST ベースのサービスの 1 つを識別するパート。例えば、Yahoo は検索、ニュース、その他のサービスを提供しています。サブドメインパートを使用して、使用するサービスを指定できます。パートナーリンクのエンドポイント参照で、指定したサブドメインに置き換えることが可能な論理サブドメイン値を追加できます: <WS-Address>nnnn.yahooapis.com</WS-Address>。
method	実行する HTTP メソッド。以下の HTTP メソッドの表を参照してください。
pathInfo	パートナーサービス（呼び出し）のオプション。WS-Address で定義したリソースの詳細を拡張するパート。例えば、search.yahooapis.com では、検索サービスのパス情報、バージョン番号、および特定の種類の検索を指定できます: NewsSearchService/V1/newsSearch
params	(オプション) アドレス指定する REST サービスによって指定されたクエリパラメータ。
headers	(オプション)。接続、ホスト、コンテンツタイプ、その他の要求の詳細を含む HTTP ヘッダー。
payload	(オプション) リソースの表現。このデータは、文字データまたは XML データとすることができます。文字データの場合は、 <i>contentType</i> 属性を含める必要があります。
queryString	(オプション) 要求内の形式で表示される文字列。指定された順序で並んだパラメータが含まれます。
authType	(オプション) インバウンド要求（受信）に入力される値。この値は、 <code>javax.servlet.ServletRequest</code> の <code>getAuthType()</code> メソッドによって返される値と同等です。値は、BASIC、FORM、CLIENT_CERT、または DIGEST のいずれかになります。
ssl	(オプション) ブール値。要求が https として送信される場合は true。
contextPath	(オプション) 要求で呼び出されたアプリケーション。
requestURI	(オプション) 要求が行われるサービスの場所。
locales	(オプション)。国と言語に関する ISO 639 コード。例: <locale country="US" lang="en">en_US</locale>

HTTP メソッド:

リソースに対して実行されるアクションを示す 8 つのメソッドを定義する HTTP。

HEAD	GET 要求に対応する応答と同じ応答を要求しますが、応答本文はありません。これは、コンテンツ全体を転送せずに、応答ヘッダーに書き込まれたメタ情報を取得する場合に役立ちます。
GET	指定したリソースの表現を要求します
POST	処理するデータ（例えば、HTML フォームからのデータなど）を特定のリソースに送信します。データは要求の本文に含まれています。これにより、新しいリソースの作成または既存のリソースの更新、あるいはその両方が発生する可能性があります。
PUT	指定されたリソースの表現をアップロードします
DELETE	特定のリソースを削除します。

TRACE	(受信に対してはサポートされていません) 受信した要求をエコーバックして、クライアントが要求で追加または変更しようとしている中間サーバーを確認できるようにします。
OPTIONS	サーバーがサポートする HTTP メソッドを返します。これは、Web サーバーの機能を確認するために使用できます。
CONNECT	(現在サポートされていません) 通常、暗号化されていない HTTP プロキシを介した SSL 暗号化通信 (HTTPS) を容易にするために、要求接続を透過的な TCP/IP トンネルに変換します。

REST 応答メッセージ

REST 応答メッセージには、以下に示すパートが含まれています。

応答のサンプルデータを生成する場合は、必ず `statusCode` 属性を有効な HTTP ステータスコードに変更してください。HTTP 要求が成功した場合の標準的な応答は `200` です。生成された整数「1」により、フォールトがスローされます。

```
<rest:RESTResponse
  xmlns:rest="http://schemas.activebpel.org/REST/2007/12/01/aeREST.xsd"
  statusCode="1">
<!--Optional:-->
<rest:headers>
  <!--Optional:-->
  <rest:header name="string" value="string"/>
</rest:headers>
<!--Optional:-->
<rest:payload contentType="string">
  <!-- Payload information -->
</rest:payload>
</rest:RESTResponse>
```

`rest:payload` が修飾されていない場合、9.2 より以前の Process Developer バージョンとは異なり、`aeRest.xsd` 名前空間は継承されません。この継承に依存していた場合は、プロセスを更新するまで XPath 式が機能しなくなる可能性があります。

REST フォールトメッセージ

サンプルの REST フォールトは次のようになります。

```
<rest:RESTFault xmlns:rest="http://schemas.activebpel.org/REST/2007/12/01/aeREST.xsd" errorType="transport">
  <rest:errorCode>400</rest:errorCode>
  <rest:message>Bad Request</rest:message>
  <rest:messageDetails>The request cannot be fulfilled due to bad syntax</rest:messageDetails>
</rest:RESTFault>
```

- 上記のサンプルで *transport* として定義されている `errorType` は、*transport* または *system* です。
- `errorCode` は、上記のサンプルで `400` エラーとして定義された HTTP ステータスコードになります。

BPEL REST メッセージ

`aeREST.xsd` スキーマは、次の 3 つのメッセージタイプを定義します。

- **RESTRequest**。これは、サーバーが My Role エンドポイントから受信したメッセージ、およびサーバーが REST パートナーエンドポイントに配信したメッセージです。
- **\$RESTResponse**。これは、REST パートナーエンドポイントからサーバーに返される成功した応答メッセージです。
- **RESTFault**。これは、HTTP エラー応答またはその他のシステム関連のフォールトが発生した REST パートナーエンドポイントからサーバーに返されるフォールトです。

各メッセージタイプの例については、[「REST ベースの受信または呼び出しの作成」 \(ページ 330\)](#)を参照してください。

マルチパート HTTP メッセージの処理

Receive と Invoke は、マルチパート HTTP メッセージをサポートします。

マルチパートメッセージには、multipart/form-data とその他すべて（例えば、mixed、related、digest、parallel）という 2 つの異なるクラスがあります。multipart/form-data は、パラメータとファイルを持つことができる HTML フォーム構成からのデータを表します。

Receive

マルチパートメッセージは解析され、ペイロードおよび添付ファイルとして RESTRequest メッセージに入力されます。最初のテキストメッセージパートはペイロードで、他のすべてのパートは添付ファイルです。

multipart/form-data の場合、パラメータは RESTRequest メッセージパラメータに変換されます。最初のテキストフォームデータファイルがペイロードになり、他のすべてのファイルはメッセージの添付ファイルになります。テキストファイルがない場合、ペイロードのみの添付ファイルというものは存在しません。

Invoke

マルチパートメッセージは、ペイロードと添付ファイルの任意の組み合わせが 2 つ以上ある場合に、暗黙的に作成されます。マルチパート Content-Type ヘッダーが RESTRequest メッセージで定義されている場合を除き、送信されるデフォルトの Content-Type は multipart/mixed です。タイプ multipart/form-data の明示的な Content-Type ヘッダーによって、HTML フォーム送信をエミュレートするように動作が変更されます。この場合、ペイロード内のすべての RESTRequest パラメータは HTML INPUT パラメータとして処理され、添付ファイルはファイルとして処理されます。

REST ベースのプロセスのデプロイメントに関する詳細の指定

REST ベースのプロセス用の PDD ファイルを作成する場合は、マイロールとパートナーロールのエンドポイント参照のリソースに関する詳細とポリシーを追加する必要があります。

パートナーロール

パートナーロールのエンドポイント参照については、呼び出しハンドラに REST を選択します。HTTP トランスポートポリシーアサーションを追加して、サービスタイムアウト値を指定することもできます。

aeREST.wsdl にはサービスバインディングがないため、次のようにリソースに URL を指定することでエンドポイント参照を指定できます。

```
<wsa:Address>http://search.yahooapis.com:80/NewsSearchService/V1/newsSearch</wsa:Address>
```

ドメインの特定のサービスに対して変数の置換を使用するには、次のようなアドレスのパラメータを使用します。

```
<wsa:Address>http://nnnn.yahooapis.com:80/NewsSearchService/nn/newsSearch</wsa:Address>
```

次に、[「REST ベースの受信または呼び出しの作成」](#)（ページ 330）に記載されているように、パラメータを要求メッセージで指定します。

マイロール

マイロールのエンドポイント参照については、プロセスが REST 要求を受信している場合、REST 対応サービスのポリシーアサーションを追加する必要があります。

REST サービスのコンテキストポリシーの例は次のとおりです。

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:aep="http://schemas.active-endpoints.com/ws/2005/12/policy">
  <aep:RESTenabled
    description="short description of the service"/>
  <aep:usage>
    This demo returns search results from multiple
```

```

search engines.
The accepted url format is:
  http://{host[:port]}/active-bpel/services/REST/
  searchDemo?query=[keyword]
keyword - the phrase to be searched for
</aep:usage>
</aep:RESTenabled>
</wsp:Policy>

```

OAuth REST ベースのシステムサービスの使用

ビジネスプロセスで、OAuth 認証を使用してリソースにアクセスすることが可能なシステムが必要となる場合は、OAuth システムサービスを使用して、プライベートリソースへの委任アクセスを許可できます。

OAuth は、シンプルかつ標準的な方法で安全な API 認証を可能にするオープンプロトコルです。ユーザーは、パスワードを共有せずにリソースへのサードパーティアクセスを許可することができます。また、範囲または期間内で制限付きのアクセスを許可することもできます。

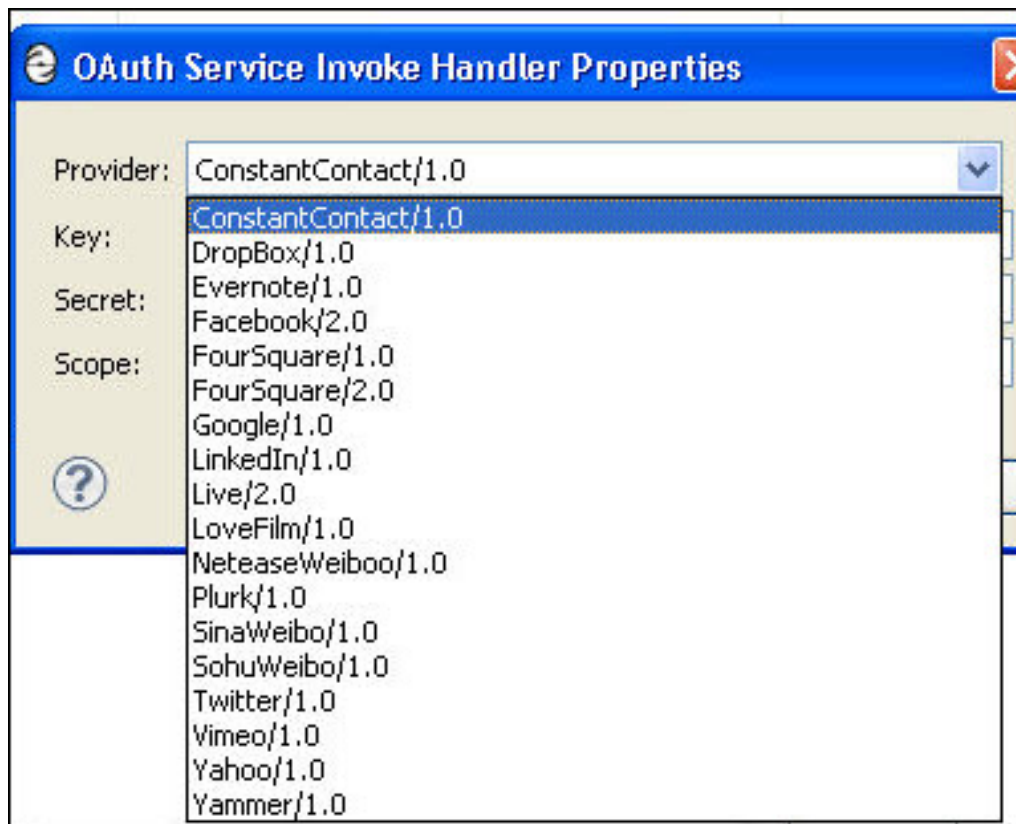
プライベートリソースへの委任アクセスという OAuth のメソッドでは、2 セットの資格情報を使用します。プロセスコンシューマ（この場合は BPEL プロセス）は識別子とシークレットを使用して自身を識別し、リソース所有者はアクセストークンとトークンシークレットによって識別されます。それぞれのセットは、ユーザー名とパスワードのペア（1 つはアプリケーション用、もう 1 つはエンドユーザー用）と考えることができます。

一部の例を以下に示します。

- ソーシャルネットワークでは、ユーザーのアドレス帳を電子メールプロバイダから取得して、ユーザーがそのソーシャルネットワークに友人を簡単に招待することができます。
- 写真印刷サービスでは、ユーザーの写真を写真ホスティングサービスから取得して、選択した写真を印刷することができます。
- 財務集計サービスでは、銀行やクレジットカード会社からユーザーの財務詳細を取得して、その情報を組み合わせたビューを表示できます。

OAuth システムサービスは、認証や将来の使用のためのアクセストークンストレージ、および認証後にプロバイダと通信する方法を提供します。

以下に示すように、サポートされる OAuth サービスプロバイダが PDD に一覧表示されます。



前提条件として、アクセスするプロバイダごとにサービスプロバイダへ登録し、クライアント ID キーとシークレットを取得する必要があります。これらのパラメータは、PDD のパートナーロールの設定に必要です。

OAuth システムサービスは、OAuth プロトコルのクライアント側コンポーネントで、ユーザー ID を使用して OAuth トークンと要求トークン（OAuth プロトコルの要素）を抽象化します。各操作は、既存のトークンを識別するための入力要素の 1 つとして userId を取得します。userId は、OAuth プロバイダごとに任意のタイミングで 1 つの承認済み OAuth トークンを持ちます。

OAuth サービスの作成

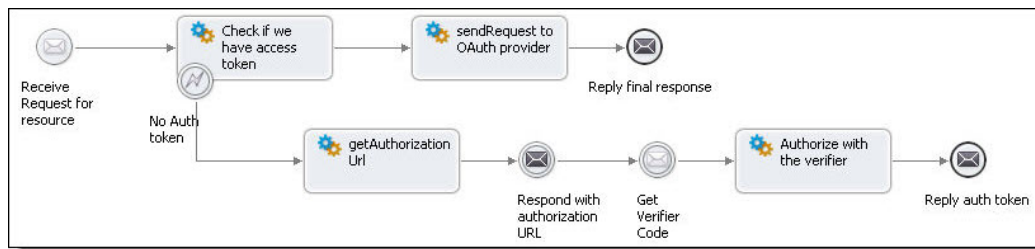
次の手順を使用して、OAuth サービスを作成します。

1. プロセスの [パーティシパント] ビューで、新しいパートナーサービスプロバイダを作成します。
2. [インタフェース] ツリーで、[システムサービス] を選択します。
3. [OAuth] を選択します。
4. 作成したパートナーサービスプロバイダから、getAccessToken 操作を開始して Invoke アクティビティを作成します。操作に関する説明は、以下を参照してください。

OAuth の操作は次のとおりです。

getAccessToken	指定されたユーザーの OAuth アクセストークンを返します。アクセストークンは、承認が完了した後も保持されます。
getAuthorizationURL	<p>OAuth プロバイダ用に設定した認証 URL を返します。Process Developer は、PDD にリストされている OAuth プロバイダ設定ごとにこの URL を維持します。この authorizationURL は（Process Developer の外部の）ユーザーによって呼び出され、このプロセス中に OAuth プロバイダは、Process Developer がアカウントにアクセスしようとしていることをユーザーに通知し、ユーザーのコンテンツを取得します。</p> <p>ユーザーがアクセスを承認すると、プロバイダは検証者の文字列/コードをユーザーに返します。OAuth 2.0 の場合、一部のプロバイダでは、要求がリダイレクトされるリダイレクト URL と、要求パラメータとしての検証用文字列が必要となります。リダイレクト URL を必要とするプロバイダ（通常は OAuth 2.0 プロバイダ）の場合、他の資格情報とともに PDD でリダイレクト URL を指定できます。</p>
authorize	<p>検証文字列（authorizationURL への応答として OAuth プロバイダによって指定された文字列）を受け取り、OAuth プロバイダで承認します。また、プロバイダから返された OAuth アクセストークンを保存します。このトークンは、プロバイダ上の保護されたリソースにアクセスするために使用されます。</p> <p>このトークンが生成されると、OAuth の初期設定ワークフローは終了します。これは通常、1 度だけ実行されます。ただし、OAuth アクセストークンの有効期限が切れた場合には、この操作を実行する必要があります（有効期間は OAuth プロバイダによって決定されます）。</p>
sendRequest	特定の OAuth プロバイダとユーザーの OAuth トークンが Process Developer に存在する場合、この操作は OAuth プロバイダシステム上の保護されたリソースにアクセスするための要求を送信します。これは、Process Developer でトークンを生成して保存する <i>authorize</i> の呼び出しを行ってから実行する必要があります。

次の図は、OAuth サービスのサンプル実装を示しています。



デプロイメントの詳細については、[「OAuth サービスプロバイダのデプロイメントに関する詳細の指定」](#)（ページ 336）を参照してください。

OAuth サービスプロバイダのデプロイメントに関する詳細の指定

このトピックでは、OAuth 呼び出しハンドラの入力パラメータについて説明します。

概要については、[「OAuth REST ベースのシステムサービスの使用」](#)（ページ 334）を参照してください。

OAuth プロバイダの PDD ファイルを作成する場合は、パートナーロールエンドポイント参照に適切なパラメータを追加する必要があります。

パートナーロールエンドポイント参照の場合は、呼び出しハンドラに OAuth サービスを選択します。

サービスに対して次の設定パラメータを入力します。

OAuth プロバイダ	リストからプロバイダを選択します
Key (必須)	クライアントの承認に使用される OAuth クライアント ID を表します。キーを取得するには、プロバイダ指定の手順に従う必要があります。例えば、LinkedIn の場合、キーとシークレットを受け取るためにはオンラインでフォームに入力する必要があります。
Secret (必須)	OAuth プロバイダから提供されたシークレットテキスト。認証時に使用します。
Scope (オプション)	スコープは、OAuth クライアントがアクセスする OAuth プロバイダのサブドメインを表します。例: <code>https://docs.google.com/feeds/</code> は、Google ドキュメントフィードにアクセスするためのスコープ値です。これは、OAuth システムサービスの観点からはオプションですが、ターゲットの OAuth プロバイダによっては必須となります。
RedirectURL (オプション)	これは通常、OAuth 2.0 プロバイダで必要です。検証文字列を使用した認証の応答は、この URL にリダイレクトされます。通常、検証文字列はクエリパラメータです (<code>url?code=verifierString</code> の形式)。

カスタムプロバイダの追加

カスタム OAuth プロバイダ (デフォルトのリストに含まれないプロバイダ) を統合して、PDD のリストに追加することができます。カスタムプロバイダを追加するには、プロバイダの 1 つ以上のエンドポイントが記述された設定ファイル (.oap ファイル) を、プロジェクトのデプロイフォルダに作成します。

例えば、*ACMEOAuthProvider* を追加する手順

```
<?xml version='1.0' encoding='UTF-8'?>
<oaconfig:oauthProviders xmlns:oaconfig="http://schemas.activebpel.org/OAuth/2011/08/01/aeoauthProviderConfig.xsd"
targetNamespace="http://schemas.activebpel.org/OAuth/2011/08/01/aeoauthProviderConfig.xsd">
  <!-- name of the provider and version. Any text and oauth
        version -->
  <oaconfig:provider name="ACMEOAuthProvider"
    oauthVersion="1.0">
    <oaconfig:requestTokenEndpoint
      verb="POST"> https://oauth.acme.com/ws/oauth/request_token
    </oaconfig:requestTokenEndpoint>
    <oaconfig:accessTokenEndpoint verb="POST">
      https://oauth.acme.com/ws/oauth/access_token
    </oaconfig:accessTokenEndpoint>
    <oaconfig:authUrl verb="GET">
      https://oauth.acme.com/ws/oauth/confirm_access?oauth_token=%s
    </oaconfig:authUrl>
  </oaconfig:provider>
</oaconfig:oauthProviders>
```

OAuth トークンが OAuth システムサービスで置き換えられるように、プレースホルダ (%s) を URL テンプレートに指定する必要があることに注意してください。

設定ファイルをプロジェクトの一部として追加すると、そのファイルが検出され、PDD の OAuth プロバイダリストがローカルの設定で更新されます。ファイルの拡張子は .oap とする必要があることに注意してください。このファイルには、任意の数のカスタム OAuth プロバイダ設定を含めることができます。

すぐに使用可能な OAuth プロバイダの設定ファイルは、場所のヒント `project:/com.activeee.rt.oauth.services/config/ae-oauth-providers.oap` にあるプロセスサーバーのカタログから取得できます。

Java Messaging Service 呼び出しハンドラの使用

Process Developer は、エンドポイントを呼び出すための Java Messaging Service (JMS) 呼び出しハンドラをサポートしています。これは、SOAP over HTTP の標準の Process Developer エンジン呼び出しフレームワークを、SOAP over JMS 呼び出しまたはプレーン XML メッセージ over JMS でバイパスできることを意味します。JMS 呼び出しを使用すると、プロセスは、サーバーがサポートする任意のメッセージ指向ミドルウェア (MOM) アプリケーションと通信できるようになります。これは、パートナーサービスを呼び出すことで、次の内容が実行可能であることを意味します。

- MOM キューへの接続
- サーバーが応答を返す MOM キューの名前の設定
- SOAP エンベロープを含む MOM メッセージの作成
- SOAP エンベロープを含む MOM メッセージの削除

JMS 呼び出しハンドラのデプロイメント記述の指定

BPEL プロセスのプロセスデプロイメント記述子ファイルでは、JMS トランスポートメカニズムを呼び出すために必要な詳細を Metro に提供する、JMS 呼び出しハンドラの詳細を指定できます。

PDD エディタの [パートナーリンク] ページで、パートナーロールの呼び出しハンドラとして [JMS] を選択します。

[エンドポイント参照] テキストボックスで、JMS トランスポートに固有の詳細を追加します。

指定可能な詳細は次のとおりです。

- キューまたはトピックのルックアップ名
- 要求/応答サービスの場合、応答をルーティングするための ReferenceParameter である、wsa:ReplyTo ヘッダーとしての応答キュー（以下の例を参照）
- パートナーエンドポイントに wsa:ReferenceParameters として含まれる追加の JMS メッセージプロパティ
- オプションで、パートナーロールに JMS ポリシーアサーションを含めることができます（以下の例を参照）

エンドポイント参照の詳細の指定

プロセスサーバーでは、JMS を介して配信される SOAP メッセージにより、WS-Addressing ヘッダーを使用して、要求が適切なサービスにルーティングされます。

ターゲットキューおよび（必要に応じて）JMS 呼び出しのサービス名は、エンドポイント参照の Address 要素で指定され、<queue JNDI name>?<service name>という形式になります。

例 1: JNDI 名を使用してキューを検索する

```
<wsa:EndpointReference>
  <wsa:Address>queue/com.activeee.jms.bpel.queue
    ?WSSReceiverService</wsa:Address>
</wsa:EndpointReference>
```

これは、次の JNDI 名を使用してキューを検索することを意味します。

queue/com.activeee.jms.bpel.queue

ここでは、ターゲットサービス名 WSSReceiverService が使用されます。このサービス名は、呼び出し中のターゲットサービスを決定するために、相手側のメッセージレシーバで使用されます。

例 2: 双方向の操作で<wsa:ReplyTo>を使用する

永続的な応答が必要とされない同期操作の場合、一時キューの JMS メカニズムによって、呼び出されたサービスのメッセージが処理されます。フェイルオーバーが発生するため、リカバリ中のノードまたは別のノードでは一時キューにアクセスできないことに注意してください。

アクティビティが長時間実行されている場合、またはフェイルオーバーが必要な場合は、エンドポイント参照定義の<wsa:ReplyTo>要素で応答の場所を指定します。以下に例を示します。

```
<wsa:EndpointReference>
  <wsa:Address>...</wsa:Address>
  <wsa:ReferenceProperties>
    <wsa:ReplyTo>
      <wsa:Address>queue/com.activeee.jms.reply.queue/
    </wsa:Address>
    </wsa:ReplyTo>
  </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

応答アドレスでは、サービスのないキュー名のみが指定されていることに注意してください。双方向の応答メッセージには、サービス名は必要ありません。

例 3: コールバック操作で<wsa:ReplyTo>を使用する

コールバックを伴う一方方向の操作である場合は、<wsa:ReplyTo>エンドポイントを次のように指定します。

```
<wsa:ReplyTo>
  <wsa:Address>queue/com.activeee.jms.reply.queue
    ?JMSCallbackService
  </wsa:Address>
</wsa:ReplyTo>
```

例 4: エンドポイント参照への JMS プロパティの追加

呼び出し時に追加の JMS 文字列プロパティを設定する必要がある場合は、パートナーエンドポイントに wsa:ReferenceParameters としてこの文字列プロパティを含める必要があります。これらのプロパティは、SOAP エンベロープとして送信する場合に、SOAP ヘッダーとしても含まれます。

```
<wsa:ReferenceParameters>
  <wsa:ReplyTo></wsa:Address>...</wsa:Address>
  </wsa:ReplyTo>
  <abx:param name="someProperty" value="value"
    xmlns:abx="http://www.activebpel.org/bpel/extension" />
</wsa:ReferenceParameters>
```

パートナーロールへの JMS 配信オプションのポリシーアサーションの追加

メッセージ送信者（パートナーロール）は、JMS で使用されるメッセージングオプションを制御します。プロセスサーバーが双方向の要求を受信すると、要求に使用されたオプションを反映して応答が送信されます。例えば、要求メッセージがプレーン XML としてフォーマットされたバイトメッセージである場合、応答も同様にプレーン XML としてフォーマットされたバイトメッセージになります。JMS 要求の送信に使用されるオプションは、エンドポイントでの WS-Policy アサーションによって制御されます。パートナーロールのポリシーアサーションとして、JMS メッセージ配信に追加の属性を指定できます。

基本的なポリシー属性は、[JMS 配信オプションのポリシーアサーション] ダイアログに表示されます。このポリシーアサーションを使用する手順については、[「ポリシーアサーションの追加」 \(ページ 468\)](#)を参照してください。

メッセージタイプ: テキスト (デフォルト) またはバイト	テキストメッセージの内容は XML テキストです。 バイトメッセージの内容はバイト配列です。
メッセージフォーマット: SOAP (デフォルト) または XML	デフォルトのフォーマットは SOAP です。SOAP では、SOAP エンベロープはメッセージペイロードと呼ばれます。SOAP メッセージの場合、すべての WS-Security 機能を使用して、メッセージレベルの認証、暗号化、および署名のサポートを提供することができます。 XML は、SOAP エンベロープのないプレーン XML としての要求または応答ドキュメントのいずれかとなります。プレーン XML メッセージの場合はメッセージレベルのセキュリティを使用できないため、アプリケーションサーバーの管理コンソールを介して宛先の JNDI ルックアップに承認制限を設定することで、アクセスを制御する必要があります。
優先度 (整数)	必要に応じて、メッセージ処理の優先度に負ではない整数を指定します。デフォルト値は 0、つまりプロバイダのデフォルト値が使用されます。 JMS では、0 (最低) から 9 (最高) までの 10 段階の優先度の値が定義されます。クライアントでは、0 から 4 ままで標準的な優先度と見なされ、5 から 9 ままで上位の優先度と見なされます。デフォルトでは、優先度は 4 に設定されています。
永続配信モード	この設定により、JMS プロバイダが、すべてのプロセスのメッセージをストレージへ永続的に保持するかどうかを制御します。デフォルトは <i>[永続的]</i> です。 <i>[永続的]</i> を選択することで、JMS プロバイダで障害が発生した場合でも転送中にメッセージが失われないようにする指示を JMS プロバイダに与えます。この配信モードで送信されたメッセージは、送信時に安定したストレージへ記録されます。永続的な配信を行うには、JMS プロバイダにストレージが設定されている必要があります。また、メッセージを永続的に保持する場合、通常は負荷が増加します。 必要に応じて <i>[非永続的]</i> を選択します。このモードでは、プロバイダ側のストレージ設定の方法に関する知識は必要ありません。
有効期間 (ミリ秒)	この設定により、コンシュームされないメッセージがキューに残る存続期間 (ミリ秒) を指定できます。一定の期間が経過した後にメッセージが期限切れとなるようにするには、有効期限を設定します。期限切れとなったメッセージを破棄すると、ストレージおよびコンピューティングリソースを節約できます。 デフォルト設定は 0、つまりプロバイダのデフォルト値が使用されます。通常、これはメッセージが有効期限切れとはならず、時間が経過してもキューに残ることを意味します。
JMS マネージャ ID	プロセスコンソールでは、複数の JMS プロバイダ設定を追加できます。デフォルト以外の設定を使用する場合は、設定済みの JMS プロバイダのマネージャ名を指定できます。

相関 ID や有効期限などのその他の属性については、パートナーエンドポイントへのコピー操作を通じて実行時に動的に設定が可能です。

jmsCorrelationID によって、双方向の操作の応答メッセージを相互に関連付けます。デフォルトでは、プロセスサーバーでは要求の wsa:MessageID が一意の相関 ID として使用されます。コンシューマが JMSCorrelationID を要求からプロパゲートする場合は、外部コンシューマとのデフォルトの相関 ID を使用します。呼び出しタイムアウトの問題が発生した場合は、相関 ID の欠落/不正が原因である可能性があります。

コンシューマが要求から相関 ID をプロパゲートしない場合は、このポリシーのオーバーライドとして明示的な jmsCorrelationID を設定します。同じキューで複数の呼び出しが応答を待機している可能性がある場合に静的な値を使用すると問題が発生する可能性があるため、注意してください。すべての応答で同じ相関 ID を使用している場合、応答がどこに呼び出されるかを判別する方法はありません。回避策として、応答を受信する場合に一時キューを使用します。応答に一時キューを使用すると、要求と応答のペアごとに一意の一時キューが存在するようになるため、相関 ID は必要なくなります。

[有効期限] とは、生成されたメッセージがメッセージシステムによって保持される、ディスパッチ時間からのミリ秒単位の経過時間です。

コピー操作によってパートナーエンドポイントに動的に追加されるポリシーアサーションの例を次に示します。

```
<partnerRole endpointReference=
  "static" invokeHandler="jms:Address">
  <wsa:EndpointReference>
    <wsa:Address>queue
      /com.activeee.jms.bpel.queue?JMSOneWayService
    </wsa:Address>
    <wsp:Policy xmlns:abp="http://schemas.active-endpoints.com
      /ws/2005/12/policy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <abp:JMSDeliveryOptions
        jmsManagerID="JMSMessagingManagerName"
        jmsMessageType="bytes"
        jmsMessageFormat="xml"
        jmsPriority="1"
        jmsCorrelationID="12345678"
        jmsExpiration = "9999999999"/>
      </wsp:Policy>
    </wsa:EndpointReference>
  </partnerRole>
```

プロセスコンソールでのメッセージサービスの設定

JMS メッセージを利用するには、プロセスサーバーの管理者がプロセスコンソール内でメッセージサービスを有効にする必要があります。詳細については、「プロセスコンソールのヘルプ」を参照してください。

第 21 章

カスタム呼び出しハンドラ

Java カスタムハンドラを記述して、BPEL プロセスでサービスエンドポイントを呼び出すことができます。

プロセスサーバーの標準呼び出しフレームワークは、BPEL プロセスの標準デプロイメントで機能します。この場合、サービスエンドポイントは、標準のパートナーロール属性を使用してプロセスデプロイメント記述子 (PDD) で識別されます。プロセスサーバーは、エンドポイントを呼び出すために、Metro (Apache SOAP エンジン) に基づく内部 Web サービスフレームワークを使用します。Metro は、データ構造からのデータを SOAP メッセージにマッシュアップし、SOAP メッセージをサービスエンドポイントに送信します。

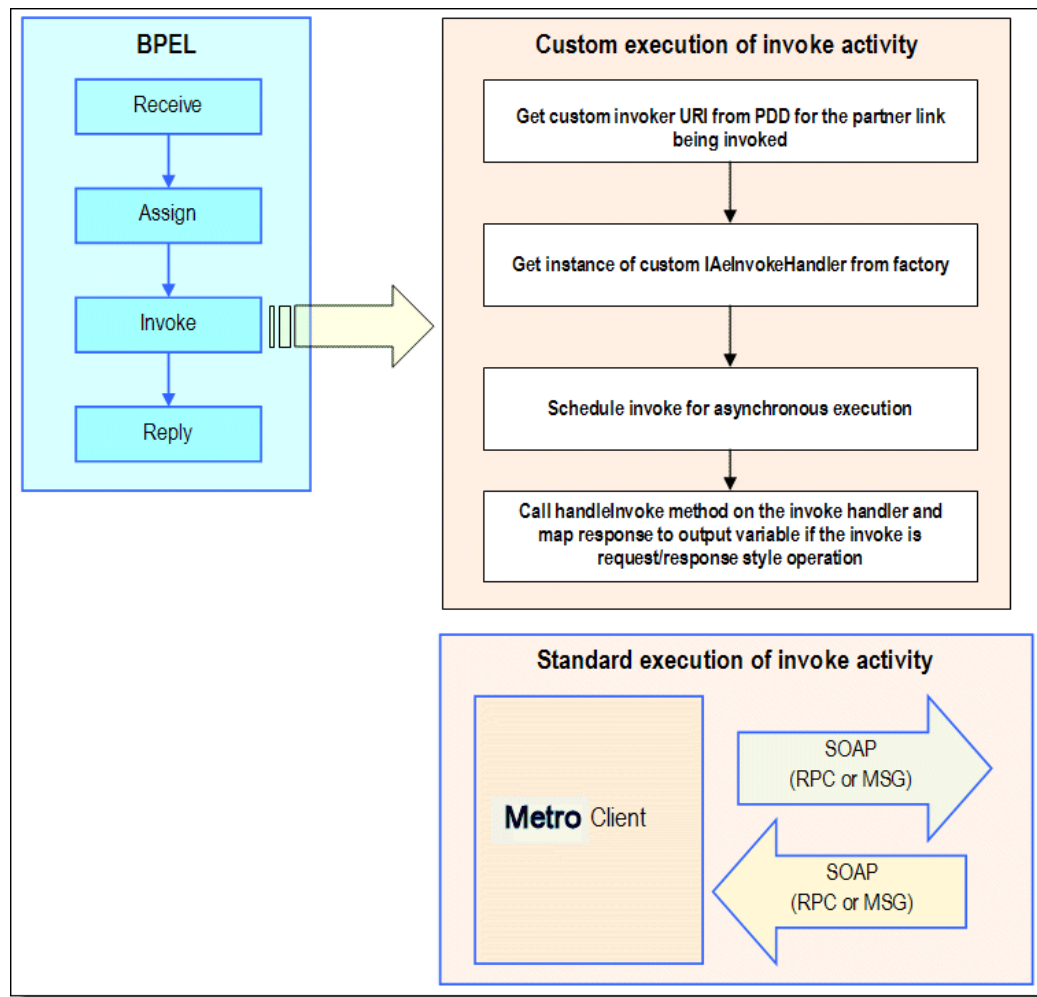
標準フレームワークを使用する代わりにカスタム呼び出しハンドラを実装することで、Metro を介した SOAP メッセージの作成と転送をバイパスできます。これは、Invoke アクティビティが実行される際に、アウトバウンドメッセージをサービスに直接ルーティングできることを意味します。

カスタム呼び出しハンドラは、プロセスサーバーによってロードされる Java クラスまたは EJB コンテナで、サービスが BPEL プロセス内で呼び出された際にパートナーロールサービスエンドポイントを呼び出します。ハンドラは、プロセスサーバーエンジンのカスタム呼び出しインタフェースである `IAeInvokeHandler` を実装します。BPEL プロセスのデプロイメント記述子には、パートナーロールサービスのカスタム呼び出しハンドラ属性が含まれます。

例えば、次のような設定をする場合があります。

- 同じマシン上のプロセスとサービス間でメッセージを送受信するためのより効率的な手法を提供する場合。SOAP メッセージではなく Java オブジェクトを作成します。
- Web サービスとして公開されていないアプリケーションにメッセージングを提供する場合
- 独自の SOAP エンジンを活用する場合

次の図は、パートナーロールサービスのエンドポイントのカスタム呼び出しと標準の呼び出しの概要を示しています。



[カスタム呼び出しハンドラ] インタフェースの参照

プロセスサーバーのカスタム呼び出しハンドラは、次のインタフェースを実装しています。

```

package org.ActiveVOS.wsio.invoke;
import org.ActiveVOS.wsio.IAeWebServiceResponse;
public interface org.ActiveVOS.wsio.invoke.IAeInvokeHandler
{
    /**
     * Handles the invoke call. Query data will be null if
     * none was specified on the customInvokerUri.
     * @param aInvoke
     * @param aQueryData
     */
    public IAeWebServiceResponse
        handleInvoke
        (IAeInvoke aInvoke, String aQueryData);
}
  
```

ここで、

- `IAeInvokeHandler` はインタフェースの名前です。パッケージは `org.ActiveVOS.wsio.invoke` です。このインタフェースは、`org.ActiveVOS.wsio.IAeWebServiceResponse` をインポートします。
- `IAeWebServiceResponse` は、呼び出し元のサービスから返されたデータまたはフォールトのインタフェースです。
- `handleInvoke` は、サービスへのエントリポイントです。
- `aInvoke` には、入力メッセージから返されたサービスのメッセージデータおよびエンドポイント参照アドレスデータが含まれます。
- `aQueryData` には、プロセスデプロイメント記述子のパートナーロールの `invokeHandler` 属性から渡される追加のパラメータが含まれます

コードの例

次の Java コードスニペットは、インタフェースでの `handleInvoke` メソッドの実装を示しています。この例では、ローン承認サービスにおいてローンが承認された分の金額を受け取り、許可された最大のローン金額に基づいて「承認済み」または「拒否済み」のメッセージが返されます。

```
public IAeWebServiceResponse handleInvoke(IAeInvoke aInvoke,
    String aQueryData)
{
    // Parse the query data section of the uri to create the
    // message type QName,
    // then create the appropriate response and return.
    Map data = parse( aQueryData );
    Integer maxLoan = new Integer(
        (String)data.get(URI_PARM_MAXLOAN));
    String ns = (String)data.get(URI_PARM_MSG_NAMESPACE);
    String lp = (String)data.get(URI_PARM_MSG_LOCALPART);
    QName messageType = new QName(ns, lp);

    // Get input message data.
    IAeWebServiceMessageData input = aInvoke.getInputMessageData();
    Map input_data = input.getMessageData();
    Integer amount = (Integer)input_data.get(MSG_AMOUNT);

    // Determine response.
    String approval = APPROVER_DENY_RESP;
    if ( validateLoan(amount, maxLoan) )
        approval = APPROVER_ACCEPT_RESP;

    // Build and return response message. Note that in this example,
    // the implementation is for a request/response invocation.
    // For a request-only invocation, you would not need to
    // setMessageData.
    Map output_data = new HashMap();
    output_data.put(MSG_ACCEPT, approval);
    AeWebServiceMessageData msgData = new
        AeWebServiceMessageData(messageType, output_data);
    AeInvokeResponse response = new AeInvokeResponse();
    response.setMessageData( msgData );
    return response;
}
```

PDD へのカスタム呼び出しハンドラ属性の追加

プロセスサーバーの標準呼び出しフレームワークをバイパスするには、カスタム呼び出しハンドラ属性をプロセスデプロイメント記述子 (PDD) ファイルのパートナーロールに追加する必要があります。

カスタム呼び出しハンドラ属性は、パートナーサービスのカスタム呼び出しを使用するようにエンジンに指定します。属性とオプションのメッセージパラメータを追加してサービスに渡し、ビジネスプロセスアーカイブ (BPR) ファイルをデプロイします。

呼び出しの実装に応じて、Java クラスまたは EJB コンテナの URI 情報を指定する必要があります。

Java クラスの実装

Process Developer の PDD ウィザードを使用して、Java 呼び出しハンドラを指定します。次の構文は、Java クラスファイルの PDD ファイルにカスタム呼び出し属性を含める方法を示しています。

```
<partnerLink name="qname">
  <partnerRole endpointReference="type"
    invokeHandler="java:org.ActiveVOS.rt.axis.bpel.
      .MyInvokeHandler?parameters+
    ...
  </partnerRole>
</partnerLink>
```

例:

```
invokeHandler="java:org.ActiveVOS.rt.axis.bpel.
  AeApproverInvokeHandler?type-namespace=
    http://tempuri.org/services/loanapprover&
    type-localpart=approvalMessage&maxLoan=1000000">
```

EJB 実装

次の構文は、EJB の PDD ファイルにカスタム呼び出し属性を含める方法を示しています。

```
<partnerLink name="qname">
  <partnerRole endpointReference="type"
    invokeHandler="ejb:jndiLocation/
      MyInvokeHandler?parameters+
    ...
  </partnerRole>
</partnerLink>
```

jndiLocation は、カスタム呼び出しハンドラを探す場所を指定するコンテキストを定義します。

例:

```
invokeHandler="ejb:ejb/AeApproverInvokeHandler?type-namespace=
  http://tempuri.org/services/loanapprover&
  type-localpart=approvalMessage&maxLoan=1000000">
```

EJB としてのカスタム呼び出しハンドラのパッケージ化

カスタム呼び出しハンドラが EJB である場合、EJB およびその他のファイルを EAR アーカイブにパッケージ化する必要があります。

パッケージには以下が含まれます。

- カスタム呼び出しハンドラの JAR ファイル
- デプロイメント記述子

サービス実装 Bean クラスとその依存クラス。これらのファイルは、ae_wsio.jar、ae_awfwsio.jar、および qname.jar です。

サンプル EAR パッケージ

```
META-INF
  Manifest.mf
  jboss-app.xml
  application.xml
  name.jar
  customInvokeSample.jar
  ae_awfwsio.jar
```

次の表に、サンプル EAR ファイルの内容を示します。

Web アプリケーションファイル	説明
META-INF Manifest.mf	EAR ファイルに関するメタ情報が含まれます。また、アーカイブにパッケージ化された他のファイルに関する情報を含めることもできます。EAR ファイルには、このファイル名とパス名が必要です。
jboss-app.xml	J2EE サーバー（この場合は JBoss）からの代表的な設定ファイル
application.xml	EAR ファイルに必要な記述子
customInvokeSample.jar	IAeInvokeHandler 実装を参照するカスタム呼び出しハンドラ
ae_wsio.jar	エンジンにメッセージを送信するために使用されるコアインタフェースが含まれています。このファイルは、プロセスサーバーの EAR ファイルにあります。
ae_awfwsio.jar	プロセスサーバーの EJB とデータのやり取りをするために必要なインタフェースが含まれています。このファイルは、プロセスサーバーの EAR ファイルにあります。
qname.jar	WSDL の処理に必要です。このファイルは、プロセスサーバーの EAR ファイルにあります。

カスタム呼び出しハンドラファイルのデプロイ

カスタム呼び出しハンドラをプロセスサーバーにデプロイするには、以下の手順に従ってください。

Apache Tomcat 用のプロセスサーバー:

1. JAR ファイルを [Tomcat インストールフォルダ]/lib にコピーします。
2. プロセスコンソールまたは Process Developer を使用して BPR ファイルをデプロイします。
3. Tomcat を再起動します。

EAR をサポートするプロセスサーバーのアプリケーションサーバー:

1. EAR ファイルを J2EE アプリケーションサーバーのデプロイサーバーにコピーします。例えば、JBoss では、[JBoss インストールフォルダ]/server/ActiveVOS server/deploy にデプロイします。
2. プロセスコンソールまたは Process Developer を使用して BPR ファイルをデプロイします。

注: プロセスサーバーに Java クラスをデプロイすることはできますが、サーバーの EAR ファイルのクラスパス上にそのクラスが存在していることを確認する必要があります。

第 22 章

ビジネスイベント処理

ビジネスイベント処理を使用することで、実行中の BPEL プロセスまたは完了した BPEL プロセスのアクティビティを監視できます。例えば、「サービス X によって注文が処理される平均時間」を確認するとします。ここで、サービス X はプロセス内の Invoke アクティビティで、特定の期間に完了したプロセスの特定の数を評価します。

また、もう 1 つの例として、「タスクの要求にかかる平均時間が N 分を超えた場合に警告する」というビジネスイベントを処理するとします。この場合、特定のビジネス条件が満たされたときにアラートが送信されるようにできます。

ビジネスイベント処理に必要な基本的な手順は次のとおりです。

- イベントを定義します。[「プロセスデプロイメント記述子でのイベントの定義」 \(ページ 347\)](#)を参照してください。
- イベントベースのプロセスをデプロイします
- 一致するビジネス条件に基づいて実行されるイベントアクションプロセスを作成します
- イベントを管理および監視するためのイベントアクションプロセスを設計およびデプロイします
- イベントを定義したプロセスをデプロイして実行します

イベントを使用することで、以下のことが可能です。

- 動作中のプロセスを監視し、異常を特定する
- アクティビティのレポートを確認する
- 特定の問題の原因を確認するために、すでに完了したプロセスの分析を実行する

次のトピックでは、ビジネスイベントの処理について説明します。

- [「プロセスデプロイメント記述子でのイベントの定義」 \(ページ 347\)](#)
- [「システム定義のイベントの使用」 \(ページ 348\)](#)
- [「Event-Action BPEL プロセスの作成」 \(ページ 350\)](#)
- [「アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ」 \(ページ 351\)](#)

プロセスデプロイメント記述子でのイベントの定義

プロセスデプロイメント記述子でのイベントの定義

必要に応じて、イベント処理で 사용되는ようにプロセス固有のイベントを定義できます。このタイプのイベントは、特定のプロセスアクティビティまたはリンクに基づいています。イベント定義には、BPEL オブジェ

クトに加えて、*実行済み*や*フォールト*などの1つ以上のライフサイクル状態が含まれ、オプションとして、イベントの発生時に使用可能なインデックス付きプロパティのデータを含めることができます。

プロセスイベントは、プロセスデプロイメント記述子エディタの「イベント」タブで定義します。

プロセスに関連するイベントの定義はオプションです。必要に応じて、システム定義のイベントを使用できます。詳細については、「[システム定義のイベントの使用](#)」(ページ 348)を参照してください。

プロセスに関連するイベントを定義する手順

1. 「[プロセスデプロイメント記述子ファイルの作成](#)」(ページ 456)に記載されている新しいプロセスデプロイメント記述子を作成します。
2. 必要に応じて、1つ以上のインデックス付きプロパティを定義します。プロセス変数のインデックス付きプロパティは、イベントの定義で使用できます。詳細については、「[インデックス付きプロパティの追加](#)」(ページ 482)を参照してください。
3. PDD エディタで「イベント」タブを選択します。
4. 「追加」を選択します。
5. 「イベント定義」ダイアログで、イベントのサービス名を入力します。これは、このイベントに応答するイベントアクションサービスの名前です。「[Event-Action BPEL プロセスの作成](#)」(ページ 350)を参照してください。
6. 「場所」選択リストで、アクティビティやリンクなどの BPEL オブジェクトを選択します。例えば、などのアクティビティを選択します
[ダイアログ] ボタンを選択してから「ビジネスイベントソースの検索」ダイアログを開くと便利です。このダイアログには、一般的なラベル (Receive や I2 など) が意味のある名前に置き換えられた、名前付きのアクティビティとリンクが表示されます。アクティビティまたはリンク名を見つける場合は、ワイルドカードを使用できます。
7. 1つ以上のアクティビティ状態を選択して、イベントをトリガします。状態の説明については、「[アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ](#)」(ページ 351)を参照してください。例えば、「*実行済み*」を選択します。
8. 必要に応じて、既存のインデックスプロパティにマッピングされたビジネスプロパティを追加します。ビジネスプロパティを使用することで、イベント用に選択したアクティビティ状態の変数に存在するデータ値を返すことができます。例えば、*customerId*が 101であるすべてのフォールト状態のプロセスを見つけることができます。ビジネスプロパティを追加する手順
 - a. 「ビジネスプロパティ」パネルで、「追加」を選択します。
 - b. 「インデックス付きプロパティ」カラムで、選択矢印を使用して、PDD の「インデックス付きプロパティ」タブで作成したインデックス付きプロパティを選択します。
 - c. 「ビジネスエイリアス」カラムで「プロパティ名」を選択すると、インデックス付きのプロパティ名が表示されます。

システム定義のイベントの使用

システム定義のイベントを使用して、イベント処理で使用する組み込みシステムイベントを選択できます。このタイプのイベントは、既知のプロセスサーバーイベントに基づいています。

システムイベントの定義は、プロセスの PDD でのイベントの定義とは異なります。システムイベントには、PDD の「イベント」タブは使用しません。代わりに、イベントアクションシステムサービスに基づいてプロセスを作成し、PDD の「パートナーリンク」タブで、必要なサービス名をマイロール *partnerlink* に指定します。詳細は以下のとおりです。

次のシステムイベントが利用可能です

- AeEngineEvent

このイベントの説明については、以下のエンジンイベントの表を参照してください。AeEngine イベントを受信するには、Event-Action システムサービスに基づいてプロセスをデプロイする必要があります。このプロセスの PDD には、必要なサービス名 AeEngineEvent（および portType EventActionPT）が含まれている必要があります。詳細については、「[Event-Action BPEL プロセスの作成](#)」（ページ 350）を参照してください。

このイベントを使用する際、別のノードがイベントを一時停止する可能性がある場合は、getProcessState 管理 API 呼び出しの使用に注意してください。例えば、デフォルトの 10 秒というプロセスアイドルタイムアウト内にノードが 2 番目のノードからプロセスをロードしようとした場合、エンジンによって、プロセスを実行しているノードのスレッドが一時停止され、要求側ノード（ノード 1）がスレッドをロードできるようになります。ただし、一時停止されると、プロセス状態を取得しようとするエンジンイベントプロセスがトリガされます。しかしながら、ノード 1 からのプロセス状態の取得の試行によって一時停止もまたトリガされるため、エンジンイベントプロセスが再びトリガされます。この試行によって、ノード 1 からのプロセス状態の取得を試みます。これらのアクションにより、親プロセスは繰り返し何度も一時停止されます。つまり、プロセスは実行状態に戻りません。

- タスクベースのエンジンイベント

説明については、以下のタスクベースのエンジンイベントの表を参照してください。タスクベースのイベントを受信するには、Event-Action システムサービスに基づいて、次の必須のサービス名のいずれかを含むプロセスをデプロイする必要があります。

- *AeHumanTaskCreated*。このサービス名は、AeHumanTaskCreated イベントにのみ使用します

- *AeHumanTaskEvent*。この名前は、AeHumanTaskEvent イベントにのみ使用します

Engine Event

イベント名	説明	イベントのプロパティ
AeEngineEvent	<p>エンジンイベントが発生した場合に発生します。エンジンイベントは以下の内容で構成されます。エンジンイベントをフィルタリングするには、その整数値を参照してください。</p> <ul style="list-style-type: none">- PROCESS_CREATED (0)- PROCESS_COMPLETED (1)- PROCESS_SUSPENDED (2)- PROCESS_RESUMED (3)- PROCESS_STARTED (4)- PROCESS_RECREATED (5)- PROCESS_UNDER_COORD (6)- PROCESS_FAULTED (7)- ENGINE_STARTED (100)- ENGINE_STOPPED (101)- ENGINE_FAILED (102)	<p>processId (long)</p> <p>state (int)。整数値は左側に表示されます。</p> <p>processName (QName)（「比較」ステートメントでは使用できません）</p> <p>processLocalName(string)</p> <p>processTargetNamespace</p> <p>timeStamp (Date)</p> <p>invokerLocationPath (String) (ProcessUnderCoordination のみ)</p> <p>parentProcessId (long) (ProcessUnderCoordination のみ)</p>

タスクベースのエンジンイベント

イベント名	説明	イベントのプロパティ
AeHumanTaskCreated	WS-HT タスクが作成された際に発生します。このイベントが発生するまでに、タスクが作成され、クエリの実行に使用できるようになります。	TaskPAProcessId (Double) TaskPALocationId (Double) TaskId (String) TaskStatus (String) 「アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ」 (ページ 351) にリストされているプロパティも参照してください。
AeHumanTaskEvent	要求、開始、追加、コメント、出力の送信、または完了などの理由で WS-HT タスクの状態が変化した場合に発生します。	TaskEventType (String) TaskEventPrincipal (String) TaskPAProcessId (Double) TaskPALocationId (Double) TaskId (String) TaskStatus (String) 「アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ」 (ページ 351) にリストされているプロパティも参照してください。

Event-Action BPEL プロセスの作成

イベントアクションプロセスは、他のプロセスで定義したイベントを受信して処理します。

Event-Action BPEL プロセスを作成する手順

1. プロジェクトエクスプローラで、[ファイル] > [新規] > [BPEL プロセス] を選択します。
2. ワークスペースフォルダを選択してファイル名を入力し、[次へ] を選択します。
3. [イベントアクションプロセス] という BPEL テンプレートを選択します。

新しい BPEL ファイルが開き、イベントトリガを入力として受け取る Receive アクティビティが表示されます。

イベントトリガは、次のいずれかに基づいています。

- 別のプロセスの PDD のイベントタブで定義したイベント。この場合、イベント定義を含むプロセスは、イベントアクションプロセスのサービス名を参照している必要があります。
- AeEngineEngine イベント。この場合、マイロールの PDD で選択するサービス名は *AeEngineEvent* にする必要があります。
- AeHumanTaskCreated イベント。この場合、サービス名は *AeHumanTaskCreated* にする必要があります。
- AeHumanTaskEvent イベント。この場合、サービス名は *AeHumanTaskEvent* にする必要があります。

システムイベントは、実行中のすべてのプロセスのイベントアクションサービスによって消費されます。ただし、次のようなプロセスをプログラムで参照することで、イベント処理を特定のプロセスに絞り込むことができます。

```
$trigger/AeEngineEvent/processLocalName/text()='myProcess'
```

以下に、定義されたイベントを使用したイベントアクションプロセスの入力メッセージの例を示します。

```
<evt:trigger xmlns:evt="http://docs.active-endpoints.com/wsd1/eventing/2008/06/eventing.wsd1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <orderitem xmlns:ext="http://dinner2">
    <processId type="xsd:long">601</processId>
    <eventId type="xsd:string">/process/flow
      /extensionActivity
      /peopleActivity[@name='PlaceOrder']:PID_601</eventId>
    <timeStamp type="xsd:dateTime">2008-07-28T19:03:33.464Z
    </timeStamp>
    <planId type="xsd:int">19</planId>
    <nodePath type="xsd:string">/process/flow
      /extensionActivity/peopleActivity[@name='PlaceOrder']
    </nodePath>
    <processName type="xsd:QName">ext:dinner</processName>
    <state type="xsd:int">2</state>
    <sessionId type="xsd:int">2</sessionId>
    <locationId type="xsd:int">75</locationId>
    <instanceNodePath type="xsd:string">/process/flow
      /extensionActivity/peopleActivity[@name='PlaceOrder']
    </instanceNodePath>
    <myOrderItemProp type="xsd:string">sandwiches
    </myOrderItemProp>
  </orderitem>
</evt:trigger>
```

アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ

アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプ

アクティビティの状態、イベントのプロパティ、タスクの状態、およびタスクイベントタイプのパラメータの定義については、次の表を参照してください。

アクティビティの状態

実行中に、アクティビティまたはリンクは処理状態に関連付けられます。[「プロセスデプロイメント記述子でのイベントの定義」 \(ページ 347\)](#)に記載されているイベントを定義する場合は、1 つ以上の処理状態を選択し、定義に含める必要があります。

アクティビティまたはリンクに関連付けることができる各状態の説明を以下に示します。

アクティビティの状態	説明
非アクティブ	アクティビティまたはリンクが次に実行されるオブジェクトではなく、また、実行されていません
実行完了	アクティビティの正常な完了
パスの停止ステータス	ブランチまたはリンクが原因でアクティビティが実行されていません
サスペンド状態	このアクティビティではプロセスが一時停止されています
実行準備完了	アクティビティまたはリンクが次に実行されるオブジェクトです
実行フォールト	アクティビティの異常な完了

アクティビティの状態	説明
終了	プロセスが終了したか、アクティビティの上位のコンテナの 1 つでフォーカスが発生したため、アクティビティが終了しました
フォールト状態	一時停止されたプロセスが、フォールトが発生したアクティビティで停止しています
実行	アクティビティが正常に実行されています
リンクのステータス	リンクが実行された場合は、true

イベントのプロパティ

次の表に、入力トリガメッセージの一部である使用可能なプロパティを示します。

イベントのプロパティ	タイプ	説明
processId	long	イベントを生成した BPEL プロセスのプロセス ID
state	int	ReadyToExecute、Executing、ExecuteComplete など、IAeProcessEvent にある状態定数の 1 つ。 注: 整数値については、以下の表を参照してください。
eventId	string	イベントを生成したアクティビティのロケーションパスとプロセス ID の組み合わせである一意のイベント ID。これは例えば、同じアクティビティによって発生した開始イベントと終了イベントを関連付ける場合に役立ちます。
instanceNodePath	string	イベントを生成したアクティビティへのパス。このパスには、forEach または他のコンテナ内に含まれた場合のインスタンス情報が含まれます
nodePath	string	イベントを生成したアクティビティへのパス。このパスにはインスタンス情報は含まれていません。これは、プロセスインスタンスではなく、プロセス定義へのパスと見なすことができます。
locationId	int	イベントを生成したアクティビティの ID (nodePath と同様)
processName	QName	イベントを生成した BPEL プロセスの名前
timeStamp	Date	イベントが発生した時間
sessionId	int	イベントが発生したときのプロセスのセッション ID。セッション ID は、プロセスが何らかの理由でメモリから離れるたびに増加します。
faultName	string	現在のプロセスフォールトの名前 (存在する場合)
planId	int	このプロセスインスタンスに関連付けられているプロセスプランの ID

イベントプロパティの 1 つは、*state* プロパティです。次の表に、状態および状態を参照するためにオプションで使用可能な整数値を示します。

イベントの状態	説明	整数値
READY_TO_EXECUTE	アクティビティを実行する準備が整ったことを示します	0
EXECUTING	アクティビティの実行が開始されました	1
EXECUTE_COMPLETE	アクティビティが正常に完了しました	2
EXECUTE_FAULT	アクティビティにフォールトが発生しました	3
LINK_STATUS	リンクが評価されました	4
DEAD_PATH_STATUS	アクティビティがパスの停止になりました（実行されません）	5
TERMINATED	Exit アクティビティが実行されたか、上位のコンテナにフォールトが発生したため、アクティビティが終了しました	6
SUSPENDED	アクティビティが一時停止されました	13
FAULTING	アクティビティにフォールトが発生しています	14
INACTIVE	アクティビティが非アクティブです	-1

タスクの状態

TaskStatus イベントプロパティで文字列として使用可能な各タスク状態の説明を以下に示します。

タスクの状態 (TaskStatus (string))	説明
READY	潜在的な所有者の 1 人がタスクを要求できます
RESERVED	潜在的な所有者がタスクを要求すると、タスクは <i>Reserved</i> 状態に移行し、その潜在的な所有者が実際の所有者になります
IN_PROGRESS	タスクが要求され、作業中です
SUSPENDED	アクティブな状態（ <i>Ready</i> 、 <i>Reserved</i> 、 <i>InProgress</i> ）のいずれかでタスクを一時停止して、一時停止状態に移行できます。
EXITED	最終状態の 1 つ（完了、失敗、エラー、終了、期限切れ）。要求中のアプリケーションは、タスクによって生成された結果を必要としなくなります。
FAILED	タスクの異常な完了
COMPLETED	作業が正常に完了しました

タスクイベントタイプ

次のリストには、タスクイベントタイプに対して有効な文字列が含まれています。

activate	deleteFault	resume
addAttachment	deleteOutput	setFault
addComment	escalated	setGenericHumanRole
cancel	expire	setOutput
claim	fail	setPriority
complete	finalize	skip
create	forward	start
delegate	nominate	stop
deleteAttachmentById	reassign	suspend
deleteAttachments	release	suspendUntil
deleteComment	remove	updateComment

第 23 章

イベント処理

次のトピックでは、イベント処理について説明します。

- [「イベント処理について」 \(ページ 355\)](#)
- [「イベントハンドラの追加」 \(ページ 356\)](#)
- [「イベントの処理ルール」 \(ページ 361\)](#)
- [「境界イベントの追加」 \(ページ 361\)](#)

イベント処理について

イベント処理は、スコープと同時に実行されるアクティビティです。イベントは、メッセージイベントまたはアラームの 2 つのタイプのいずれかになります。メッセージイベントまたはアラームが発生すると、それに関連付けられたイベントハンドラがアクティビティを呼び出します。

イベントハンドラは、メインアクティビティに関連したスケジュールを設定できない、想定外のタイミングで発生する可能性があるイベントや要求に対して特に有効です。これは例えば、顧客が処理中の注文をキャンセルした場合などが考えられます。

イベントハンドラは、通常の処理を引き継ぐフォールトハンドラや補償ハンドラとは異なり、スコープの通常の動作の一部と見なされます。

イベントハンドラは、プロセス全体またはスコープにアタッチすることができます。対応するスコープがアクティブである間は、1 つ以上のイベントが発生する可能性があり、任意のタイミングで処理されます。

イベントハンドラは、プロセスインスタンスが作成されるまで有効化できません。つまり、プロセスインスタンス自体を作成することはできません。

イベントタイプ

イベントは、WSDL での要求/応答または一方向の操作に対応する受信メッセージである可能性があります。例えば、ステータスクエリは要求/応答操作である可能性が高い一方で、キャンセルは一方向の操作である可能性があります。

イベントは、設定された時間に鳴るアラーム、または指定された時間継続するアラームである場合もあり、必要に応じて繰り返されます。

イベントハンドラアクティビティの例

イベントをトリガするメッセージを受信した場合、次のうちのいずれかがイベントハンドラによる応答である可能性があります。

- 応答を送信する。

- プロセスインスタンスを終了する（例えば、注文のキャンセルが発生してプロセスインスタンスをキャンセルする必要があり、元に戻して、補償する必要が生じるような進行中の作業がない場合など）。
- 進行中の作業を元に戻して、補償する必要が生じるようなフォールトをスローする。

イベントハンドラの追加

イベントハンドラは、プロセス全体またはスコープに追加できます。次の2種類のイベントハンドラを追加できます。

- [「onEvent イベントハンドラの追加」 \(ページ 356\)](#)
- [「onAlarm イベントハンドラの追加」 \(ページ 358\)](#)

概念情報については、[「イベント処理について」 \(ページ 355\)](#)を参照してください。

XML 構文

```
<eventHandlers>?
  <!-- there must be at least one onEvent
       or onAlarm handler -->
  <onEvent partnerLink="NCName" portType="QName"?
    operation="NCName"
    (messageType="QName" | element="QName")?
    variable="BPELVariableName"?
    messageExchange="NCName"?>*
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"/>+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName"/>
  </fromParts>
  <scope...>...</scope>
</onEvent>
<onAlarm>*
  (
    <for expressionLanguage="anyURI"?>duration-expr</for>
    |
    <until expressionLanguage="anyURI"?>deadline-expr</until>
  )
  <repeatEvery expressionLanguage="anyURI"?>?
    duration-expr
  </repeatEvery>
  <scope...>...</scope>
</onAlarm>
</eventHandlers>
```

onEvent イベントハンドラの追加

onEvent 要素により、メッセージが到着するまで特定のイベントが待機するように指定します。この要素は、メッセージイベントによって新しいプロセスインスタンスが作成されないことを除き、Receive アクティビティと非常に類似しています。また、onEvent には Scope アクティビティが含まれ、必要に応じて、パートナーから受信したメッセージを含むイベントハンドラのローカル変数が使用されます。

受信メッセージは、WSDL の要求/応答または一方向の操作に対応します。操作が要求/応答タイプである場合、イベントハンドラは通常、Reply を使用して応答を送り返します。

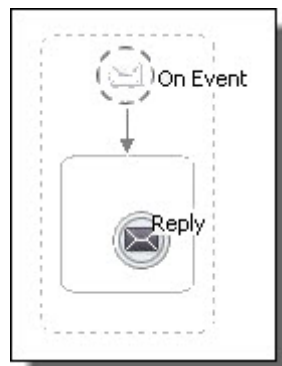
必須のプロパティ	オプションのプロパティ
パーティシパント	変数
操作	ポートタイプ
	関連。第 24 章、「関連」 (ページ 367) を参照してください。
	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76) を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	メッセージ交換。 「メッセージ交換の宣言」 (ページ 136) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

プロセスに onEvent イベントハンドラを追加する手順

イベントハンドラのプロパティの入力を行う前に、イベントハンドラにパーティシパントを追加したことを確認してください。

1. Process Editor の [イベントハンドラ] タブをクリックします。
2. [メッセージキャッチイベント] を [イベントハンドラ] キャンバスにドラッグします。
3. [プロパティ] ビューから、パーティシパント、操作を選択し、必要に応じて変数定義を選択して、変数名を入力します。この変数は、他のアクティビティで使用する暗黙的なスコープ変数です。
4. アクティビティを onEvent のスコープにドラッグして、応答や終了などのイベントに応答するようにします。
5. イベントを処理するアクティビティのプロパティを入力します。

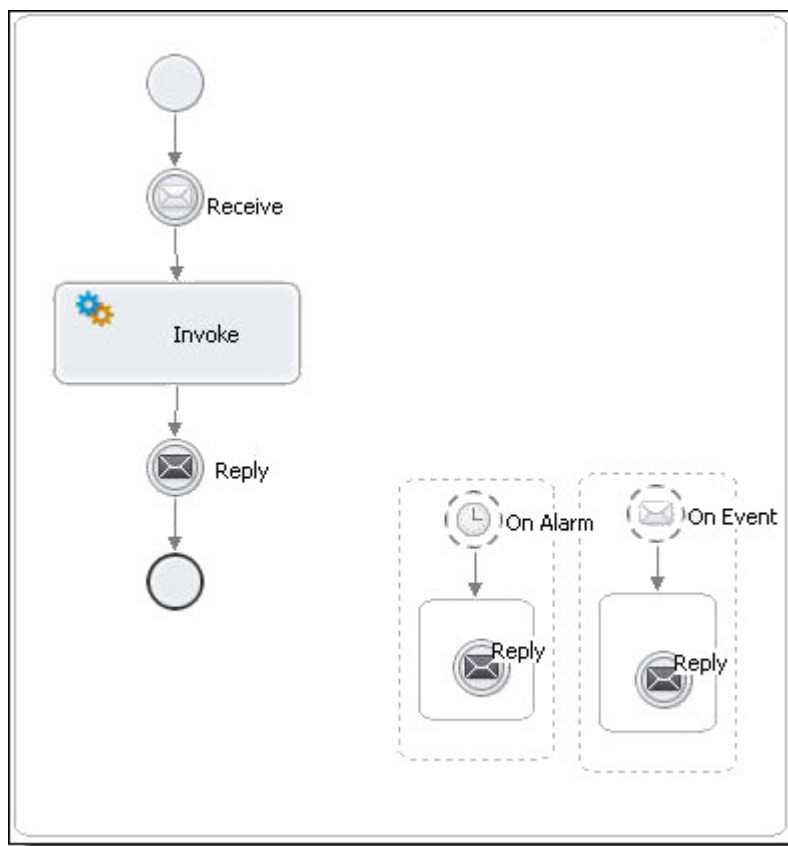
次の図は、プロセスに追加された onEvent イベントハンドラの例を示しています。イベントハンドラを折りたたむと、[プロパティ] ビューで背景色を追加できます。



スコープに onEvent イベントハンドラを追加する手順

1. 必要なパーティシパント情報を含む WSDL ファイルがプロジェクトエクスプローラに追加されていることを確認してください。
2. Process Editor で BPEL ファイルを表示し、スコープを選択します。
3. **【メッセージキャッチイベント】** をスコープにドラッグします。
4. パーティシパント、操作などの onEvent のプロパティを入力し、必要に応じて変数を入力します。この変数は、他のアクティビティで使用される暗黙的なスコープ変数です。
5. アクティビティを onEvent のスコープにドラッグして、応答や終了などのイベントに応答するようにします。
6. イベントを処理するアクティビティのプロパティを入力します。

次の図は、スコープに追加された onEvent および onAlarm イベントハンドラの例を示しています。各イベントに
応答するアクティビティは Reply です。



onAlarm イベントハンドラの追加

onAlarm 要素によって、**期間**（待機期間）または**期限**（待機期限）属性を使用して、タイムアウトイベントを定義します。**期間**属性では、イベントが通知されるまでの時間を指定します。継続時間は、関連するスコープが開始されると同時にカウントが開始されます。もう一方の**期限**属性では、アラームが発生した特定の時点を設定します。必要に応じて、**繰り返す単位**属性を設定して、間隔期間が終了するたびにアラームを繰り返し発生させることもできます。**繰り返す単位**属性は、単独で使用するか、**期限**属性または**期間**属性とともに使用します。

例えば、アラームを 5 分で鳴るように設定してから、30 秒ごとにアラームを繰り返すようにすることができます。

次の例に示すように、プロセスインスタンスを作成するメッセージ内に表示されるデータを使用して、プロセスのアラーム時間を設定できます。

WSDL ファイルでは、*processDuration* を指定する部分で複合的なメッセージが定義されています。

```
<wsdl:definitions
  targetNamespace="http://www.example.com/wsdl/exmple"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  ...>
  <wsdl:message name="orderDetails">
    <part name="processDuration"
      type="xsd:duration"/>
  </wsdl:message>
</wsdl:definitions>
```

上記のメッセージタイプは、BPEL プロセスの *onAlarm* 値で使用されます。

```
<process name="orderCar"
  xmlns:def="http://www.example.com/wsdl/example" ...>
  ...
  <eventHandlers>
    <onAlarm
      <for>$orderDetails.processDuration</for>
    </onAlarm>
  </eventHandlers>
  ...
  <variable name="orderDetails"
    messageType="def:orderDetails"/>
  </variable>
  ...
  <receive name="getOrder"
    partnerLink="buyer"
    operation="order"
    variable="orderDetails"
    createInstance="yes"/>
  ...
</process>
```

上記の例では、*onAlarm* 要素によって、*orderDetails* 変数の *processDuration* フィールドで指定した期間を超えた場合に発生するタイマーイベントを指定しています。フィールドの値は、注文の詳細を含むメッセージを受信し、その注文のプロセスインスタンスを作成する *getOrder* アクティビティを介して入力されます。

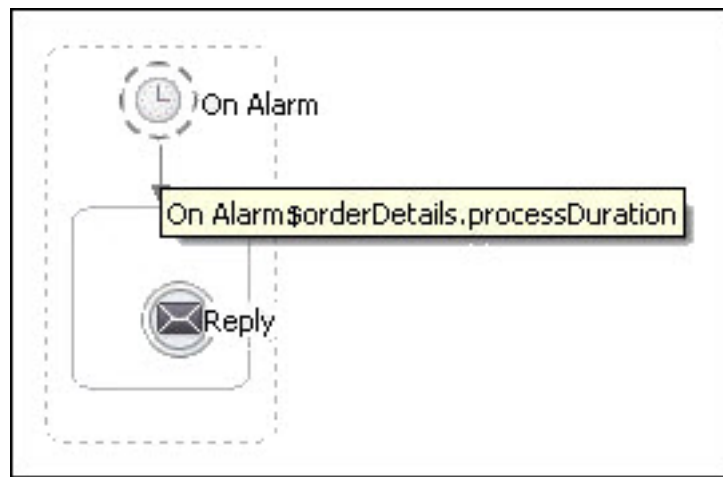
必須のプロパティ	オプションのプロパティ
アラーム式	繰り返す単位（アラームタイプで使用可能）
アラームタイプ または繰り返す 単位	コメント。 「プロセスへのコメントの追加」 （ページ 79）を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 （ページ 79）を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 （ページ 76）を参照してください。
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 （ページ 419）を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 （ページ 126）を参照してください。

プロセスに *onAlarm* イベントハンドラを追加する手順

1. Process Editor の [イベントハンドラ] タブをクリックします。

2. **【タイマーキャッチイベント】** を **【イベントハンドラ】** キャンバスにドラッグします。
3. **【プロパティ】** ビューで、必要に応じて **【繰り返す単位】** 行の最後にある **【ダイアログ】** ボタンを選択します。間隔の式を入力するか、作成します。この属性は、単独で使用するか、または待機タイプと組み合わせて使用します。
4. **【待機タイプ】** を次のいずれかに設定します。
 - a. **【期限式】** を選択して、アラームの期限（待機期間）を表す式を入力するか、作成します。
 - b. **【期間式】** を選択して、アラームの期間（待機期間）を表す式を入力するか、作成します。
 - c. **【期間】** には、待機する日数、時間数、分数、または秒数を選択します。
 - d. 期限には、X 日単位の T 時間、つまり X 日プラス T 時間を設定します。この式は次のような式に解決されます。
`xsd.dateTime(current-date()) + xsd.dayTimeDuration("P3DT15H20S")`
5. アクティビティをイベントハンドラにドラッグして、応答や終了などのイベントに応答するようにします。
6. イベントを処理するアクティビティのプロパティを入力します。

次の図は、プロセスに追加された onAlarm イベントハンドラの例を示しています。

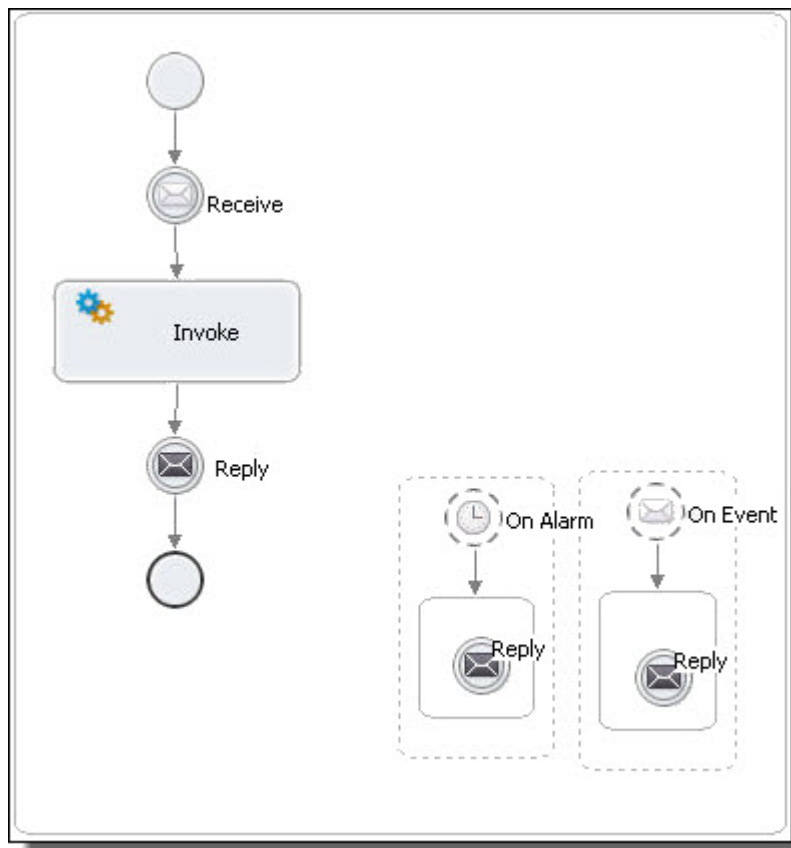


ヒント: イベントハンドラを折りたたむと、**【プロパティ】** ビューで背景色を追加できます。

スコープに onAlarm イベントハンドラを追加する手順

1. Process Editor で BPEL ファイルを表示し、スコープを選択します。
2. **【タイマーキャッチイベント】** をスコープにドラッグします。
3. **【プロパティ】** ビューで、上記のようにプロセスのプロパティを入力します。
4. アクティビティをイベントハンドラにドラッグして、応答や終了などのイベントに応答するようにします。
5. イベントを処理するアクティビティのプロパティを入力します。

次の図は、スコープに追加された onAlarm および onEvent イベントハンドラの例を示しています。各イベントに応答するアクティビティは Reply です。



イベントの処理ルール

アラームイベントは、*[繰り返す単位]* 属性によって変更されない限り、1 度だけ発生します。

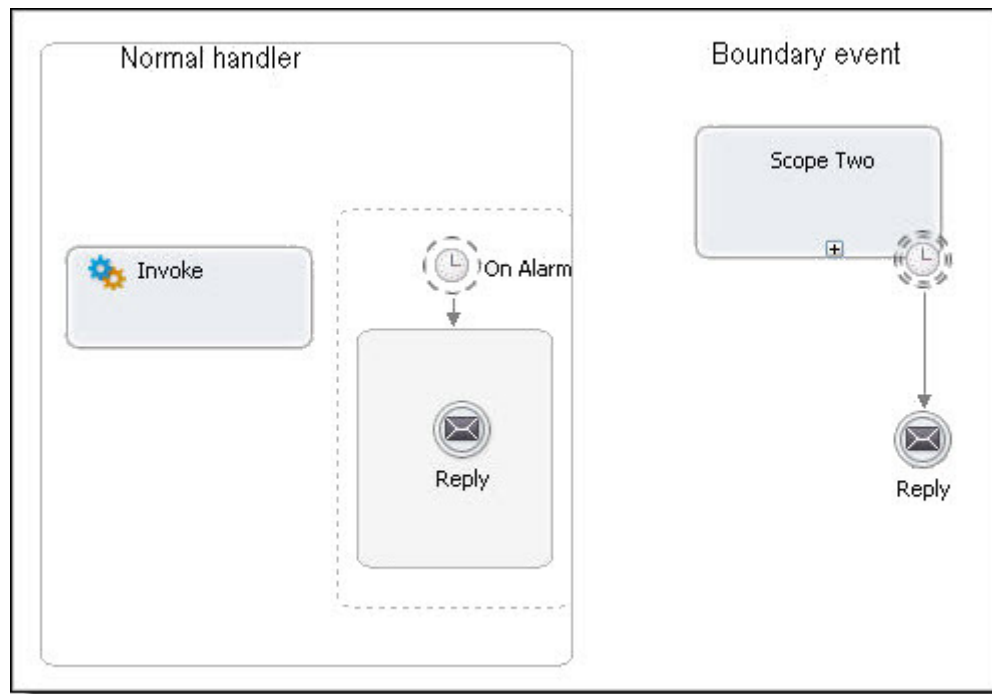
メッセージイベントは複数回発生する可能性があります。例えば、ある注文がスコープ内で処理されている間、この注文に対する各変更はメッセージイベントを介して受け入れられます。

複数のメッセージイベントとアラームイベントが同時に発生する可能性があります。共用変数とイベントハンドラを持つ 2 つの同時スコープがプロセスに含まれている場合は、スコープに対して *[分離]* を有効にしてください。*[分離]* を有効にすると、共用変数への一貫したアクセスが保証されます。

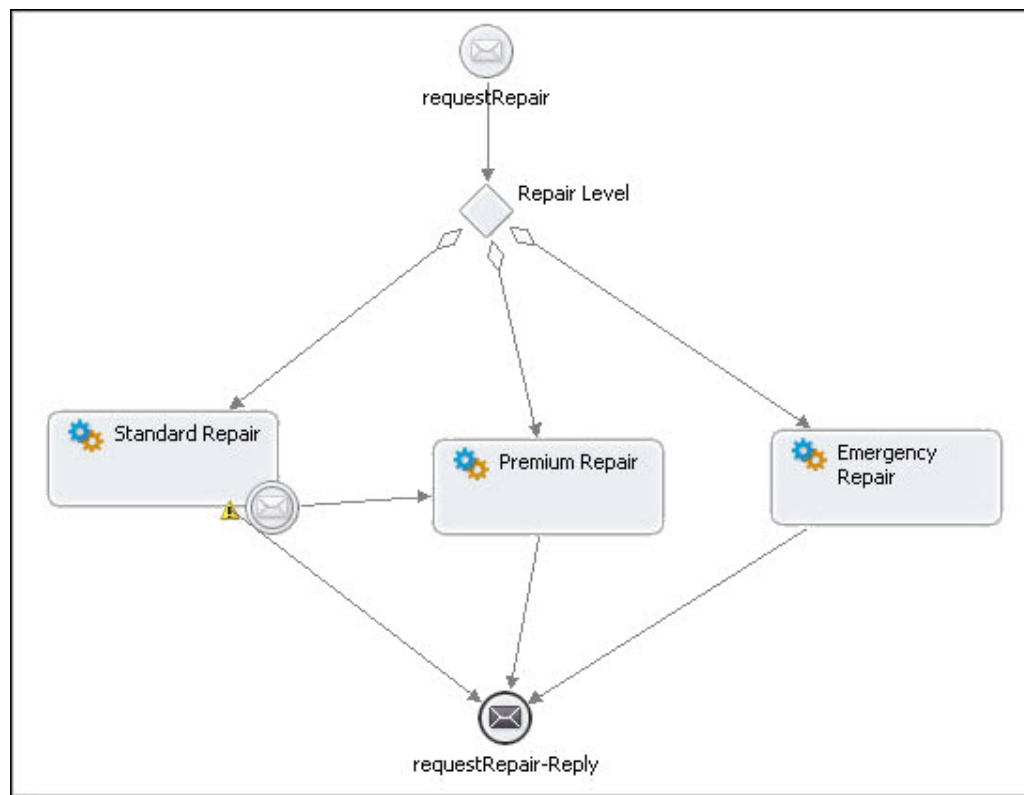
境界イベントの追加

ビジネスプロセスモデリング表記法 (BPMN) 2.0 仕様には、境界イベントの概念が含まれています。Process Developer での BPMN 境界イベントとは、境界のあるアクティビティの境界上にドロップされた BPEL スコープハンドラ (イベント、フォールト、または補償) を指します。スコープとその内部のスコープハンドラを作成するためのショートカットとして境界イベントを使用することで、設計領域を整理して作業を行うことができます。イベント処理は、メインアクティビティと同じレベルで表示されます。Process Developer はバックグラウンドで BPEL コードを作成し、有効な実行を行うためにアクティビティをスコープにラップします。

次の図は、通常のイベントハンドラと比較したスコープの境界イベントを示しています。



境界イベントは、境界のあるアクティビティに追加できます。次の例は、修正要求を「標準」から「プレミアム」にエスカレートする onEvent ハンドラを使用した呼び出しを示しています。



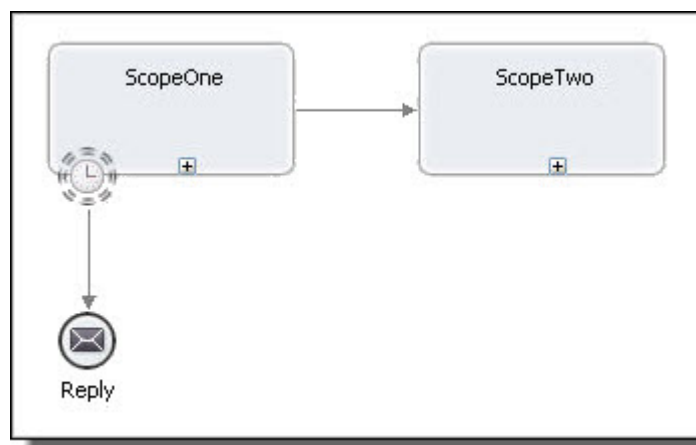
境界イベントのプロパティは次のとおりです。

- onEvent、onAlarm、Catch、Catch All、または Compensation のいずれかとなります。
- Scope や Invoke などは、丸い枠を持つ境界アクティビティの境界（境界）にのみドロップする必要があります。
- 実行されていないアクティビティ（ダウストリームアクティビティ）、またはパスの停止となっていないアクティビティ（if コンテナのブランチ）へのアウトバウンドリンクが使用されます。
- アウトバウンドリンクに条件を設定することはできません。
- 中断または非中断として定義できます（以下で説明）。
- 中断境界イベントは、BPEL に対する Process Developer 拡張機能です。
- クラシックエディタではなく、BPMN エディタでのみ使用できます。

境界イベントの例を以下に示します。

非中断 onAlarm または onEvent

次の図では、折りたたまれたスコープの境界に onAlarm ハンドラが表示されています。ハンドラは、スコープ内のアクティビティと並列で実行されます。境界イベントには、スコープにドロップされる onAlarm や onEvent ハンドラのようなコンテナはありません。しかし、境界イベントは、実行する 1 つ以上のアクティビティにリンクされます。リンクに条件を設定することはできません。



上記の例では、onAlarm 境界イベントはスコープの onAlarm ハンドラと同じように動作します。つまり、同様の処理ルールが適用され、アラームがトリガされた場合、メインスコープのアクティビティは onAlarm アクティビティと並列で実行されます。これにより、境界イベントは非中断となります。リンクされたアクティビティ（上記の例では Reply）を、メインプロセスの一部にすることはできません。

図に示すように、中断境界イベントのプロパティはデフォルトで無効になっています。

中断 onAlarm または onEvent

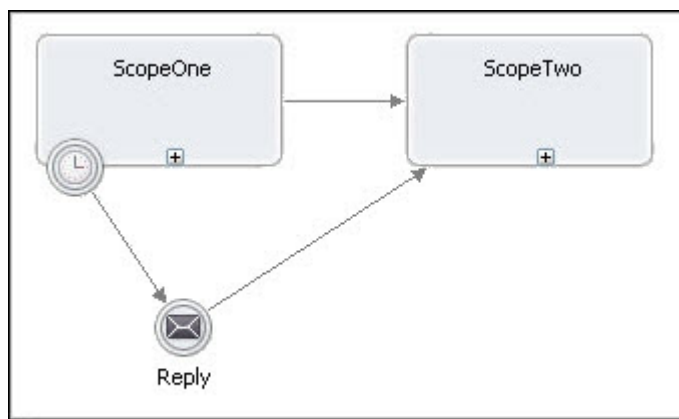
必要に応じて、上記の中断プロパティを有効にすることで、onAlarm または onEvent に新しい処理ルールを追加できます。この場合、イベントがトリガされるとメインスコープは終了します。

中断境界イベントを使用する場合の例を以下に示します。

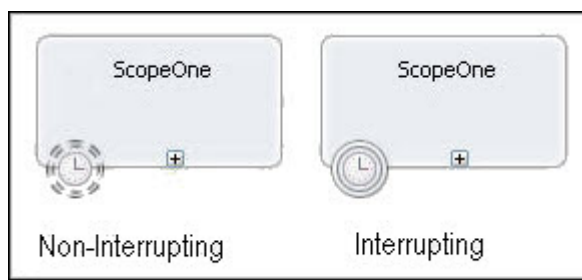
- onEvent: オーダーをキャンセルする場合
- onAlarm: 要求の関連性が失われ、必要なくなった場合

中断境界イベントは、メインプロセスのダウストリームアクティビティに再度接続し直すことができます。この動作は WS-BPEL 2.0 の拡張機能であることに注意してください。Process Developer は、[「相互に排他的な遷移」](#) (ページ 221) の拡張機能を使用して、中断境界イベントを実行します。

次の図では、折りたたまれたスコープに、トリガされるとメインスコープを終了する中断 onAlarm が含まれています。1 つのパスのみが実行されます。



非中断境界イベントと中断境界イベントのアイコンの外観の違いに注目してください。



フォールトハンドラまたは補償ハンドラに境界イベントを追加することもできます。[「Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow」](#) (ページ 365)に記載されているように、重要性の低い一部の例外があります。

関連事項:

- [「スコープへの補償ハンドラの追加」](#) (ページ 403)
- [「呼び出しアクティビティの補償」](#) (ページ 405)
- [「Catch イベントまたは中断 OnEvent 境界イベントからの変数の使用」](#) (ページ 365)

Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow

一部の BPEL アクティビティには、Catch または Catch All 境界イベントのターゲットとして使用できないものがあります。

Catch または Catch All にリンクするターゲットとして、Compensate、Compensate Scope、または Rethrow アクティビティを追加することはできません。Catch および Catch All イベントは、それらのイベントが接続されたスコープの終了によってトリガされる中断イベントです。そのため、このイベントはスコープ外で実行されます。Compensate、Compensate Scope、および Rethrow アクティビティは、スコープ内で実行されます。これらのアクティビティは、スコープの Catch または Catch All ハンドラ内に追加する必要があります。

詳細については、次を参照してください:

- [「Compensate」](#) (ページ 187)
- [「Compensate Scope」](#) (ページ 189)
- [「Rethrow」](#) (ページ 206)

Catch イベントまたは中断 OnEvent 境界イベントからの変数の使用

中断 onEvent 境界イベントおよび Catch 境界イベントは、Process Developer の拡張機能です。そのため、これらのイベントの変数の処理は、通常の BPEL フォールトおよびイベント処理とは異なります。

Catch 境界イベントの変数の処理

Catch 境界イベントの変数を定義する場合は、既存のプロセス変数を選択するか、[フォールト変数] 選択リストで新しいプロセス変数を作成する必要があります。

これは、特別なフォールト変数名を入力可能な通常のスコープ Catch ハンドラでの変数の定義とは異なります。

通常の Catch ハンドラでは、スコープでエンクローズされたアクティビティ内でフォールト変数が使用されるため、このような違いが発生します。中断境界イベントでは、イベントハンドラ内にアクティビティはありません。代わりに、境界イベントはプロセスまたは別のスコープのアクティビティにリンクします。

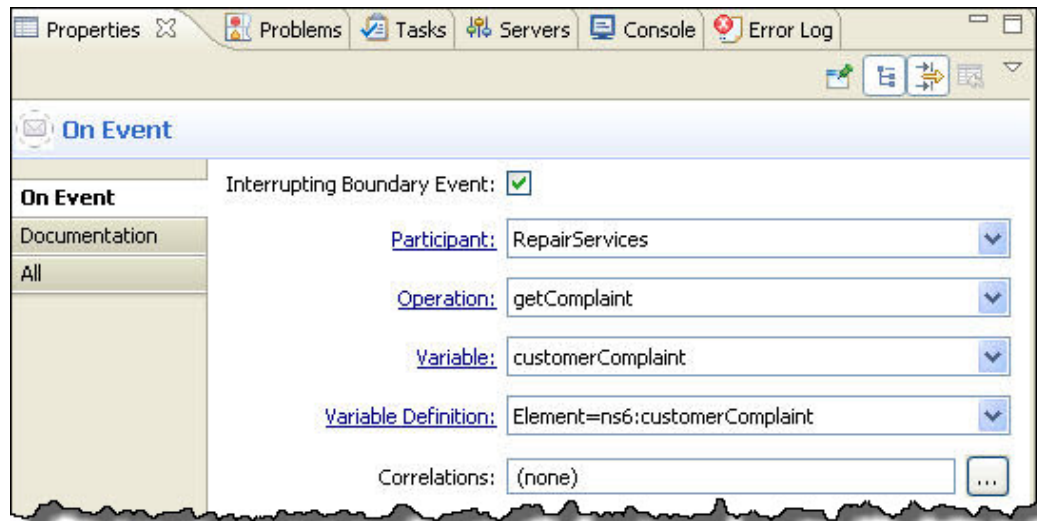
Process Developer は、境界イベント変数を使用するために必要なアクティビティをバックグラウンドで生成します。Process Developer は、Catch 境界イベントを含む非表示スコープ内でのみ使用可能な Catch の暗黙的な parameters 変数を作成します。したがって、フォールトのある応答など、プロセスの別の部分で使用する変数を定義するには、プロセス変数を選択するか、新しいプロセス変数を作成する必要があります。Process Developer は、暗黙的な parameters 変数をプロセス変数にマッピングします。以下の例を参照してください。

中断 OnEvent 境界イベント

変数の処理は、基本的に Catch 境界イベントの場合と同様です。中断 OnEvent 境界イベントの [プロパティ] ビューで、既存のプロセス変数を選択するか、[変数] ドロップダウンで新しい変数を作成します。

例

次の図は、中断 OnEvent 境界イベントのプロパティを示しています。



BPEL ソースコード内に、割り当てのあるスコープが追加されていることに注意してください。この割り当てによって、暗黙的なスコープ変数、parameters をプロセス変数 *customerComplaint* にコピーします。

```
<bpel:eventHandlers>
  <bpel:onEvent aei:boundaryEvent="interrupting" element="ns6:customerComplaint"
    ext1:handleIMAINEnclosingScope="true"
    operation="getComplaint"
    partnerLink="RepairServices"
    variable="parameters">
    <bpel:scope>
      <bpel:assign ext1:muxTransitions="yes">
        <bpel:sources>
          <bpel:source linkName="L5"/>
        </bpel:sources>
        <bpel:copy>
          <bpel:from variable="parameters"/>
          <bpel:to variable="customerComplaint"/>
        </bpel:copy>
      </bpel:assign>
    </bpel:scope>
  </bpel:onEvent>
</bpel:eventHandlers>
```

第 24 章

相関

相関に関するトピックは次のとおりです。

- [「相関について」 \(ページ 367\)](#)
- [「相関セットについて」 \(ページ 368\)](#)
- [「メッセージプロパティとプロパティエイリアスの作成」 \(ページ 371\)](#)
- [「相関セットの追加」 \(ページ 376\)](#)
- [「相関セットの削除」 \(ページ 378\)](#)
- [「アクティビティへの相関の追加」 \(ページ 378\)](#)
- [「相関セットを宣言および使用するためのルール」 \(ページ 382\)](#)
- [「相関セットおよびエンジンによって管理される相関」 \(ページ 382\)](#)

相関について

相関は、特定の 1 つのビジネスパートナーのインタラクションに属するメッセージのグループを追跡するための構造です。相関によって、各メッセージとそれらのメッセージの対象となるビジネスプロセスインスタンスとのインタラクションを一致させます。

メッセージを受信すると、BPEL エンジンはプロセスインスタンスを作成するか、すでに実行中のプロセスと一致させるかを決定します。相関セット内のデータは、あるメッセージをそのメッセージが想定されるプロセスに対してエンジンで一致させるための署名です。このための操作として、イベントハンドラのアクティビティを受信、選択、応答、呼び出し、および調整するための相関を追加します。

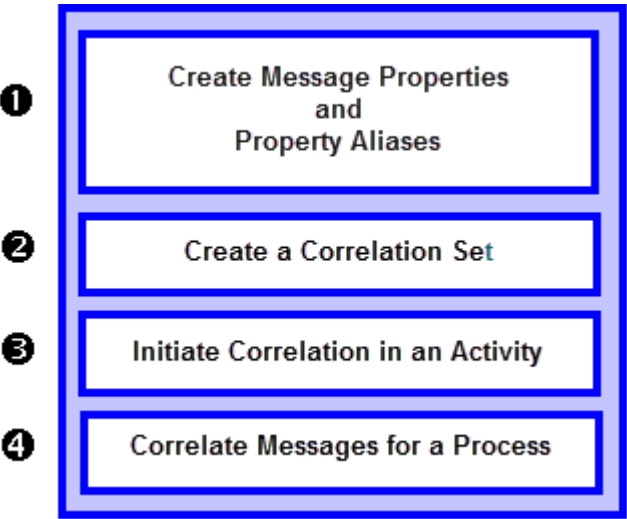
相関を使用する場合は、以下の内容を考慮してください。

- 相関させる必要のあるアクティビティを決定する。これらのアクティビティは、1 つ以上の共通データを共有している必要があります。
- 共通データを識別する 1 つのプロパティを定義する。
- プロパティエイリアスを定義する。これは、相関させる必要のある各アクティビティの共通データに応じて異なります。
- プロパティとプロパティエイリアスを含む相関セットを作成する。
- 相関させるアクティビティに相関セットを追加する。

例として、相関セットごとに、customerId などのデータプロパティを作成します。相関させるメッセージごとに、CustID という名前の Receive のメッセージパートや customerNum という名前の Invoke の入力メッセージパートなどのメッセージパートにプロパティエイリアスを作成します。

または、「[エンジンによって管理される相関](#)」 (ページ 473)に記載されているように、エンジンによって管理される相関ポリシーアサーションを使用することもできます。

次の図に、プロセスに相関を追加する手順を示します。



それぞれのステップの詳細については、リンクを選択してください	
1	「メッセージプロパティとプロパティエイリアスの作成」 (ページ 371)
2	「相関セットについて」 (ページ 368)
3	「相関のパターンの開始と設定」 (ページ 378)
4	「アクティビティへの相関の追加」 (ページ 378)

相関セットについて

相関セットで使用する WSDL で定義されたプロパティを選択します。

相関セットとは、メッセージによって共有されるプロパティのセットで、セット内の他のすべてのメッセージと同じ相関メッセージデータです。各メッセージの要素名は異なる可能性があるため、メッセージ間で同じデータを照合するメカニズムが必要です。

このメカニズムを機能させる方法について相関セットごとに、customerID などのデータプロパティを作成します。相関させるメッセージごとに、custID という名前の Receive のメッセージパートや CustomerNumber という名前の Invoke の入力メッセージパートなどのメッセージパートにプロパティエイリアスを作成します。

プロパティ

メッセージ相関に使用するメッセージプロパティは、xsd:int や xsd:string のようなスキーマ単純型である必要があります。

プロパティエイリアス

プロパティは、WSDL メッセージ、スキーマ要素、およびスキーマタイプに存在します。プロパティエイリアスによって、メッセージパートまたはクエリを指定します。

例えば、注文発注プロセスに OrderId プロパティがあるとします。メッセージには、OrderId の値が異なる名前さまざまなパートに含まれます。プロパティエイリアスはそのパートを識別します。特定のプロセスインスタンスは、2 つのプロセスが OrderId に対して同じ値を持つことがないように、一意の OrderId で識別されます。これにより、メッセージが到着した場合に、正しいプロセスインスタンスにディスパッチされます。

次の図は、メッセージのプロパティエイリアスがプロパティ名にどのようにマッピングされるかを示しています。プロパティ名は相関セットにマッピングされます。必要な数の相関セットを作成できます。

Correlation Set	Property	Property Alias
CS 1	OrderId	CustomerId
		OrderRequest
		OrderResponse
CS 2	InvoiceId	POOrder Invoice CustomerId
	ShipNoticeId	OrderRequest OrderResponse

詳細については、以下を参照してください。

- [「メッセージプロパティとプロパティエイリアスの作成」](#) (ページ 371)
- [「プロパティ名およびエイリアスの WSDL 構文と例」](#) (ページ 369)
- [「グローバル相関セットとローカル相関セット」](#) (ページ 370)

プロパティ名およびエイリアスの WSDL 構文と例

Process Developer ウィザードを使用して、変数プロパティとプロパティエイリアスを自動的に作成します。(これについては [「変数プロパティとプロパティエイリアスの追加」](#) (ページ 245) に記載されています)。次の例に示すように、このウィザードは WSDL コードを生成します。

プロパティ名の WSDL 構文は次のとおりです。

```
<wsdl:definitions name="NCName"
  xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
  <vprop:property name="NCName"
    type="QName"?
    element="QName"?/>
  ...
</wsdl:definitions>
```

プロパティエイリアスの WSDL 構文は次のとおりです。

```
<wsdl:definitions name="NCName"
  xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
  <vprop:propertyAlias propertyName="QName"
    messageType="QName"?
    part="NCName"?
    type="QName"?
    element="QName"?/>
  <vprop:query queryLanguage="anyURI"?>
    queryContent
  </vprop:query>
</wsdl:definitions>
```

```

    </vprop:query>
  </vprop:propertyAlias>
  ...
</wsdl:definitions>

```

プロパティとプロパティエイリアスの例

次のようなメッセージ定義について考えてみてください。

```

<wsdl:definitions name="messages"
  targetNamespace="http://example.com/taxMessages.wsdl"
  xmlns:txtyp="http://example.com/taxTypes.xsd"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <!-- define a WSDL application message -->
  <wsdl:message name="taxpayerInfoMsg">
    <wsdl:part name="identification"
      element="txtyp:taxPayerInfoElem" />
  </wsdl:message>
  ...
</wsdl:definitions>

```

次の WSDL フラグメントは、メッセージの特定のフィールドでのプロパティの定義とその場所を示しています。

```

<wsdl:definitions name="properties"
  targetNamespace="http://example.com/properties.wsdl"
  xmlns:tns="http://example.com/properties.wsdl"
  xmlns:txtyp="http://example.com/taxTypes.xsd"
  xmlns:txmsg="http://example.com/taxMessages.wsdl" ...>
  <!-- define a correlation property -->
  <vprop:property name="taxpayerNumber" type="txtyp:SSN" />
  ...
  <vprop:propertyAlias propertyName="tns:taxpayerNumber"
    messageType="txmsg:taxpayerInfoMsg"
    part="identification">
    <vprop:query>txtyp:socialsecnumber</vprop:query>
  </vprop:propertyAlias>
  <vprop:propertyAlias propertyName="tns:taxpayerNumber"
    element="txtyp:taxPayerInfoElem">
    <vprop:query>txtyp:socialsecnumber</vprop:query>
  </vprop:propertyAlias>
</wsdl:definitions>

```

最初の<vprop:propertyAlias>は、名前付きプロパティ tns:taxpayerNumber を、メッセージタイプ txmsg:taxpayerInfoMsg の識別部分の場所のエイリアスとして定義します。

2 番目の<vprop:propertyAlias>は、同じ名前のプロパティ tns:taxpayerNumber の 2 番目の定義を指定しますが、ここでは要素 txtyp:taxPayerInfoElem 内の場所のエイリアスとして定義を指定します。

両方のエイリアスが存在するということは、txmsg:taxpayerInfo のメッセージを保持する変数、および txtyp:taxPayerInfoElem を使用して定義された要素から社会保障番号を取得できることを意味します。

グローバル相関セットとローカル相関セット

相関セットは、プロセス全体またはスコープに対して宣言できます。スコープに対してローカルで相関セットを宣言する場合、宣言はスコープ内のアクティビティにのみ適用されます。スコープ内で同じ名前の相関セットを宣言することで、グローバル相関セットを非表示にすることができます。

メッセージプロパティとプロパティエイリアスの作成

プロパティ定義を作成し、既存の WSDL に追加します。必要に応じて、スキーマの単純型に基づいてプロパティの新しい WSDL を作成します。相関セットで使用する WSDL で定義されたプロパティを選択します。

WSDL ファイルにメッセージプロパティ定義が含まれていない場合は、それらの定義を既存の WSDL に自動的に追加するか、新しい WSDL を作成します。プロパティは、対応するプロセス変数に自動的に追加されます。

メッセージプロパティとプロパティエイリアスを使用して、相関セットを作成します。概要については、[「相関について」](#) (ページ 367) および [「相関セットについて」](#) (ページ 368) を参照してください。

メッセージプロパティとプロパティエイリアスを作成する準備として、次のことを考慮します。

- どのアクティビティを関連付ける必要があるか。
- これらのアクティビティには、共通のデータを含むメッセージが含まれているか。

WSDL ファイルにメッセージプロパティ定義が含まれていない場合は、それらの定義を既存の WSDL に自動的に追加するか、新しい WSDL を作成します。プロパティは、対応するプロセス変数に自動的に追加されます。

メッセージプロパティを作成する前に、BPEL プロセスによって [アウトライン] ビューの [インポート] セクションで WSDL ファイルが参照されていることを確認します。詳細については、[「WSDL、スキーマ、およびその他のリソースのインポート」](#) (ページ 122) を参照してください。

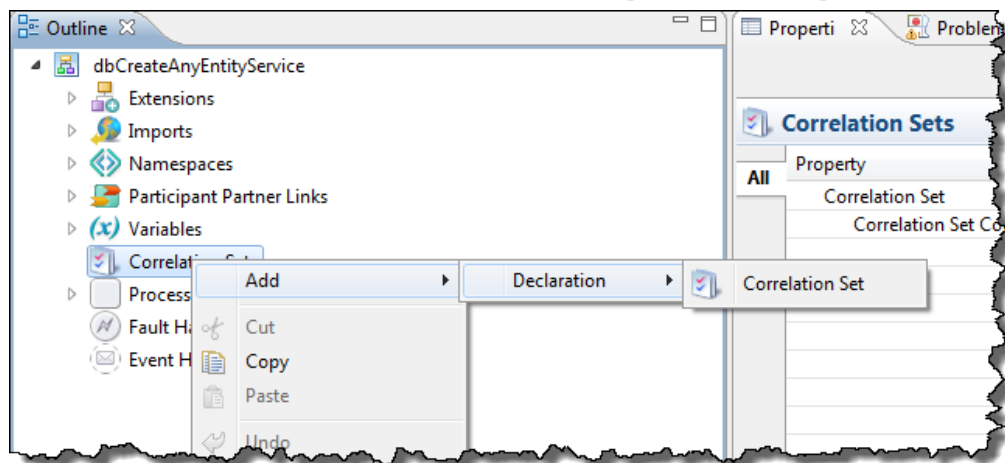
メッセージのプロパティとエイリアスを追加する場合は、以下を参照してください。

- [「プロパティ定義の作成」](#) (ページ 371)
- [「プロパティエイリアスの作成」](#) (ページ 374)

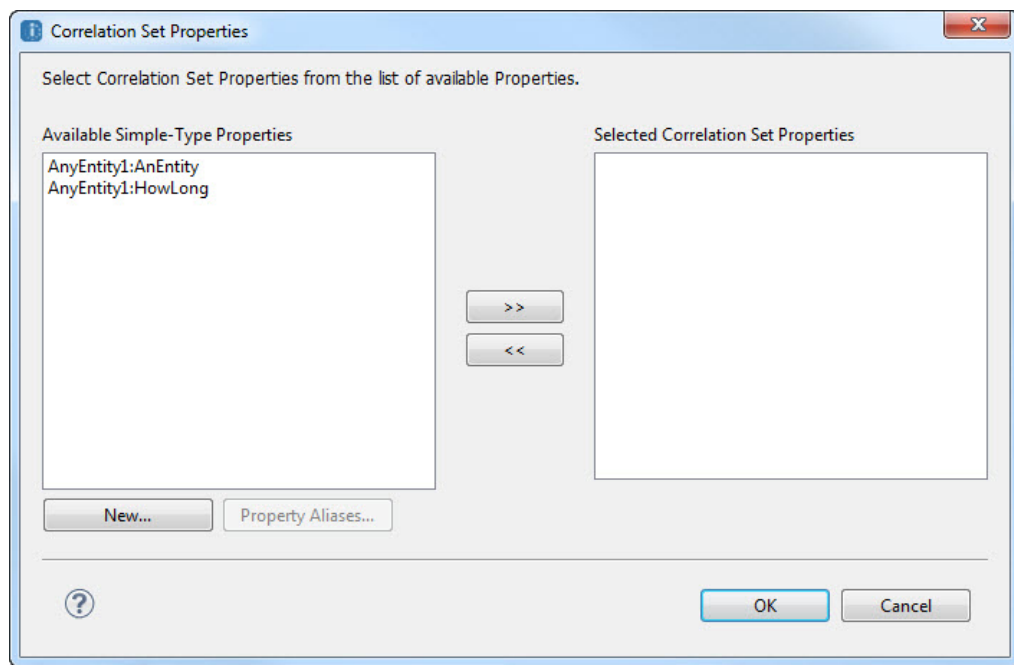
プロパティ定義の作成

プロパティ定義を作成し、既存の WSDL に追加します。必要に応じて、スキーマの単純型に基づいてプロパティの新しい WSDL を作成します。次のような方法でプロパティを作成できます。

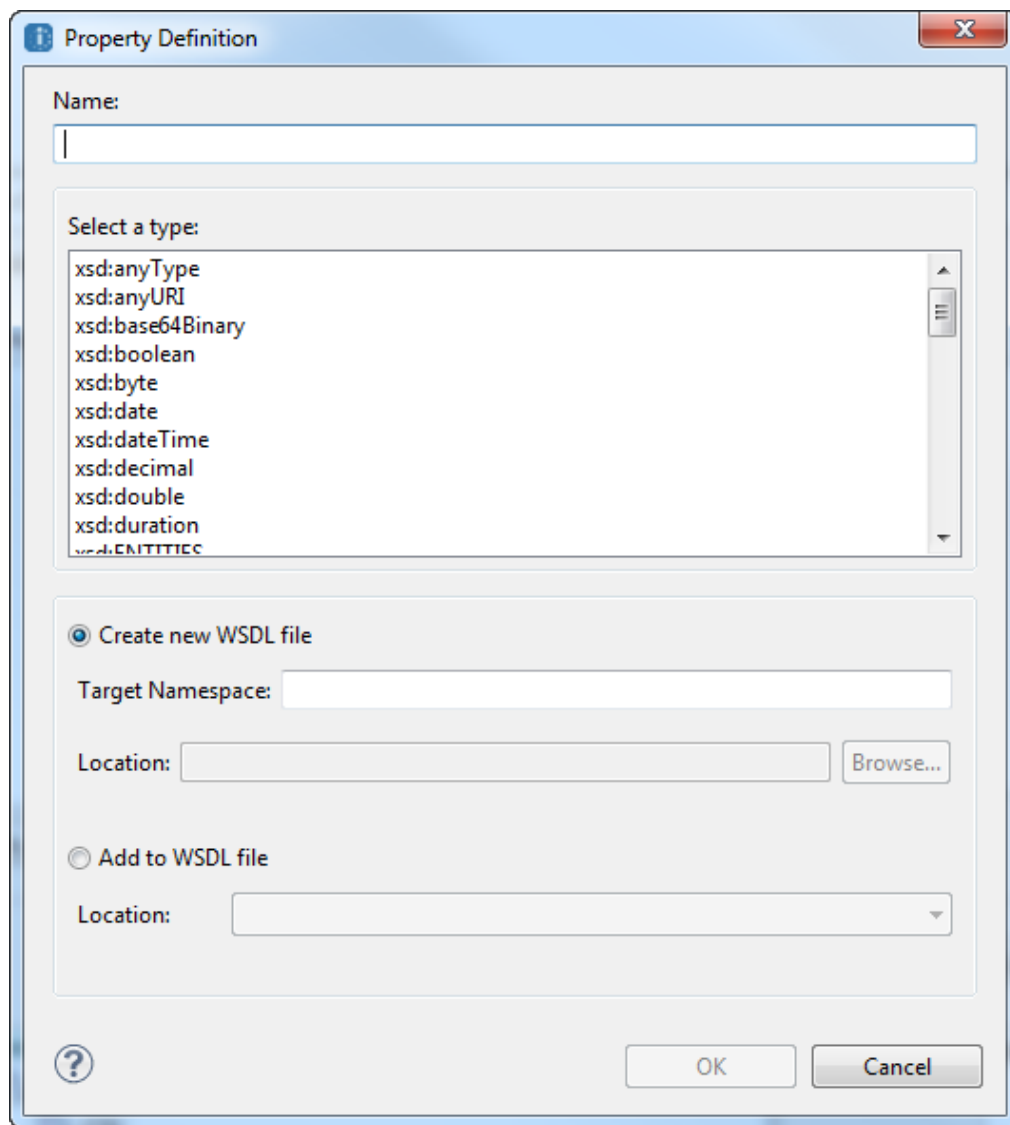
1. [アウトライン] ビューで [相関セット] を右クリックし、[「相関セットの追加」](#) を選択します。



2. [相関セットプロパティ] ダイアログで、[「新規」](#) を選択します。



3. 「プロパティ定義」ダイアログで、プロパティの名前を選択します（例: `OrderId`）。



4. **【スキーマタイプ】** または **【スキーマ要素】** のいずれかのプロパティタイプを選択します。
プロパティはスキーマ単純型である必要があります。
5. プロパティ定義を既存の WSDL に追加するか、新しい WSDL を作成するかを選択します。プロセスで参照される WSDL が URL である場合、または多くのパートナーによって共有されている場合は、新しい WSDL を作成することをお勧めします。
6. 新しい WSDL を作成するには、次の手順を実行します。
 - a. WSDL のターゲット名前空間を入力します（例: urn:MyNewNamespace）。
 - b. **【参照】** を選択し、**【開く】** ダイアログで、プロジェクトエクスプローラのプロジェクトまたはファイルシステムの場所を参照します。
 - c. WSDL ファイルのファイル名を入力し、**【開く】** をクリックします。
 - d. **【OK】** をクリックして、手順 8 に進みます。
7. 既存の WSDL に追加するには、選択リストから WSDL を選択します。このリストは、プロセスに追加された WSDL で構成されています。**【OK】** をクリックします。
8. **【関連セットプロパティ】** ダイアログの **【利用可能なプロパティ】** リストで、新しいプロパティが一覧表示されることに注意してください。

9. 次のいずれかを実行します。

- [「プロパティエイリアスの作成」 \(ページ 374\)](#)を続行する。
- **[OK]** をクリックして、ダイアログを閉じる。プロパティエイリアスは後で追加できます。これらのプロパティエイリアスは、メッセージ相関セットを追加する前に必要です。

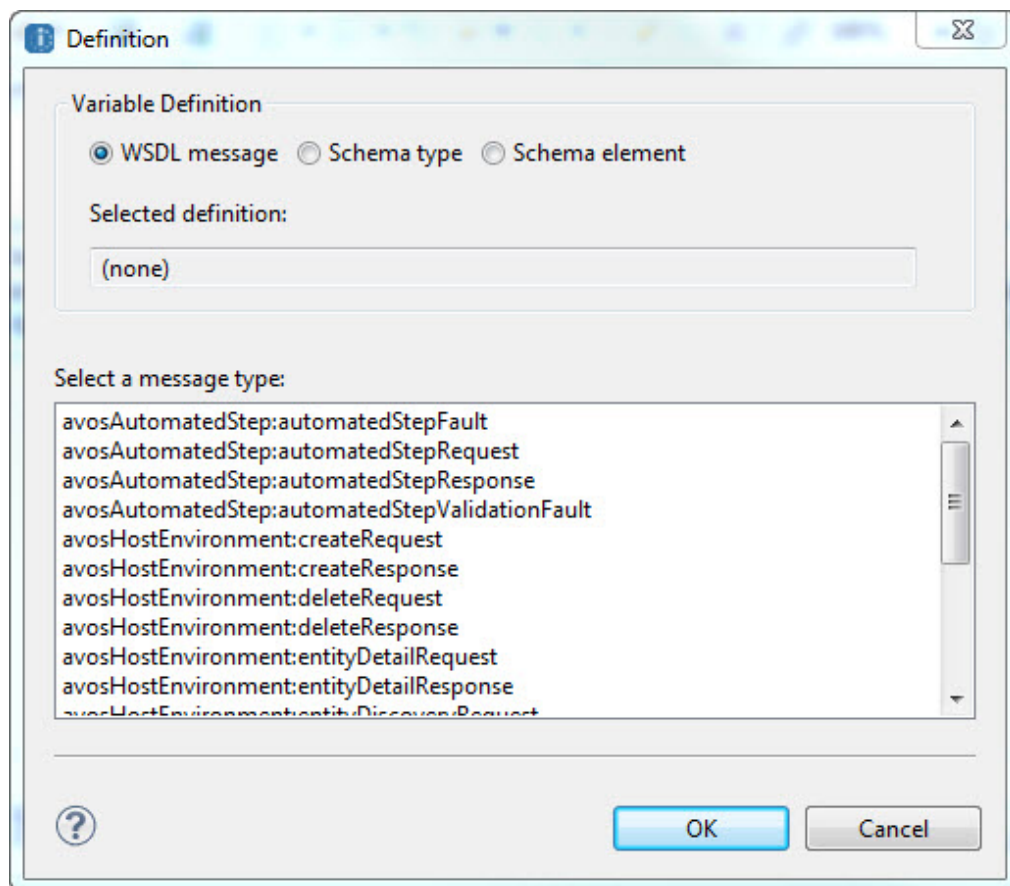
新しい WSDL を作成する場合、Process Developer で次のことを行います。

- [アウトライン] ビューの [インポート] セクションに WSDL を追加する。
- 新しい名前空間をプロセスに追加する。
- <import>要素を新しい WSDL に追加して、プロセスの WSDL ファイルを参照する。これにより、プロパティエイリアスに適切なメッセージ、タイプ、または要素を選択できるようになります。
- 新しい WSDL を [プロジェクトエクスプローラ] ビュー、[パーティシパント] ビュー、および [インタフェース] ビューに追加します。

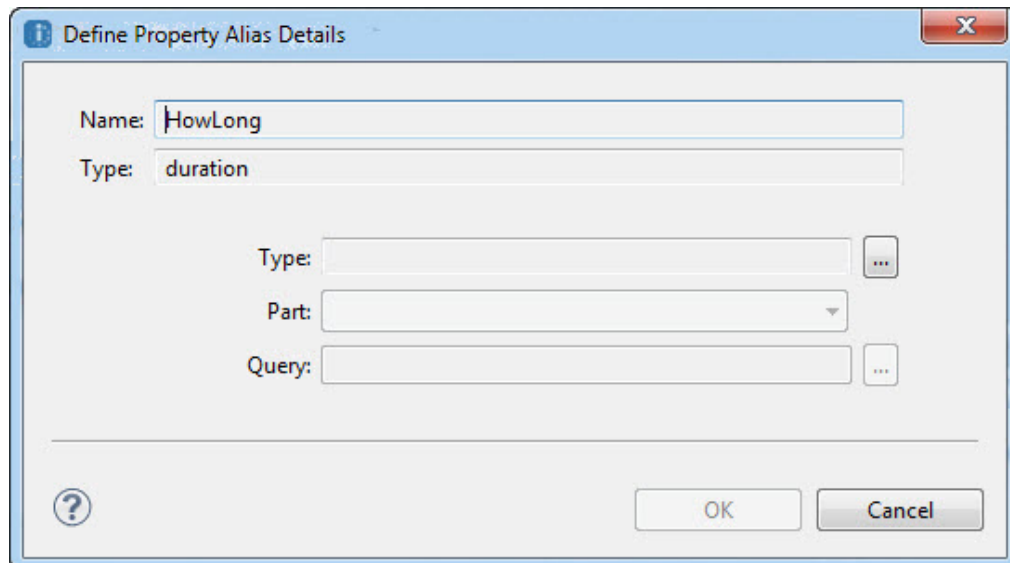
プロパティエイリアスの作成

選択したプロパティのプロパティエイリアスを追加、編集、または削除します。プロパティエイリアスのメッセージパートを選択します。複合型の場合は、クエリを含めます。

1. **[相関セットの追加]** ダイアログの [使用可能なプロパティ] リストからプロパティを選択します。
2. **[プロパティエイリアス]** を選択します。
3. **[プロパティエイリアス]** ダイアログで、**[新規]** を選択します。
4. **[プロパティエイリアスの詳細情報の定義]** ダイアログで、[タイプ] フィールドの横にある [ダイアログボタン (...)] を選択して、**[定義]** ダイアログを開きます。このダイアログには、例に示すように、プロセスにインポートされた名前空間からのメッセージ、スキーマタイプ、および要素が含まれています。

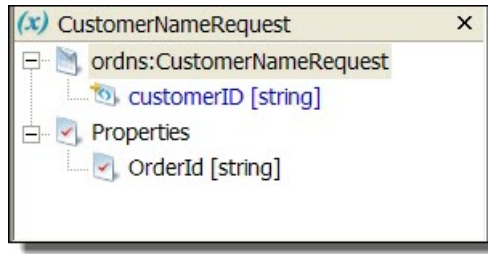


5. 変数定義（WSDL メッセージ、スキーマタイプ、またはスキーマ要素）を選択します。選択した定義タイプに応じて、使用可能なタイプのリストが表示されます。
6. リストから項目を選択し、[OK] をクリックします。
7. 例に示すように、[プロパティエイリアスの詳細情報の定義] ダイアログで、プロパティエイリアスの変数パートを選択します。



8. パートが複合タイプの場合は、クエリを入力するか、[ダイアログボタン (...)] を選択してクエリビルダーを開き、パートから選択ノードを追加します。
9. **[OK]** をクリックします。
10. これらの手順を繰り返して、プロパティに含まれるすべてのメッセージ、スキーマタイプ、およびスキーマ要素のプロパティエイリアスを追加します。選択する内容は、関連のためにメッセージをグループ化する方法に応じて異なります。

プロパティエイリアスを追加すると、例に示すように、Process Developer ではプロセス変数のプロパティが表示されます。



また、Process Developer では、プロパティとプロパティエイリアスがプロジェクトの WSDL フォルダにも追加されます。

プロパティエイリアスを作成するためのヒント:

- プロパティエイリアスは一度に 1 つずつ追加できます。
- 必要に応じて、プロパティエイリアスを編集および削除できます。WSDL ファイルと関連するプロセス変数が正しく更新されます。

関連セットの追加

関連セットをアクティビティに追加します。

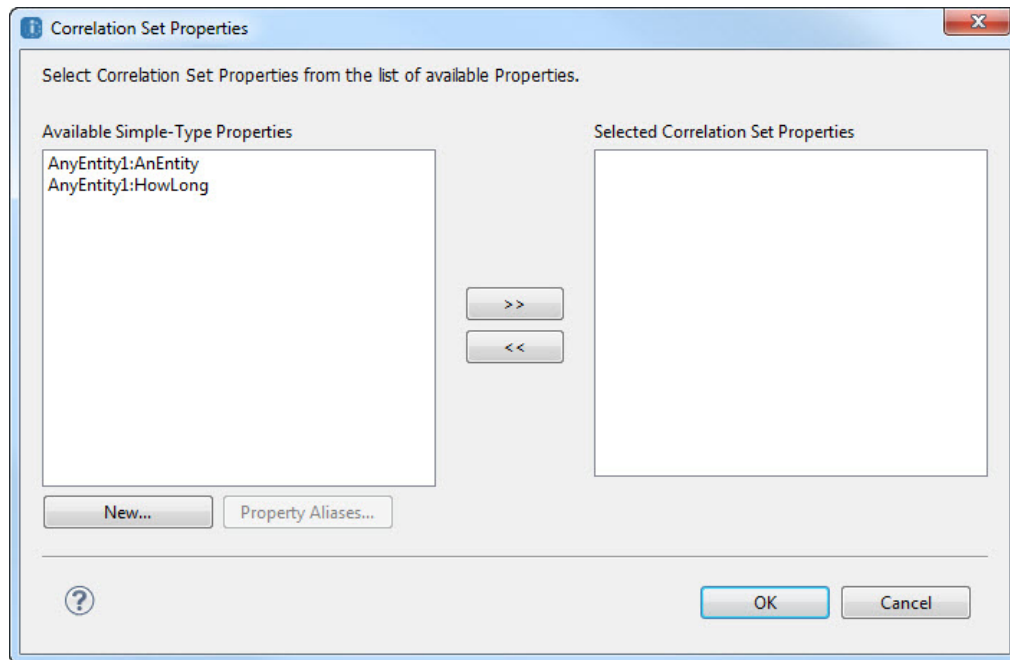
関連をアクティビティに追加する前に、関連セットを作成する必要があります。

関連セットを作成するには、以下で説明するように、関連プロパティにマッピングする WSDL ファイルからメッセージ、要素、またはスキーマのプロパティを特定します。

関連セットは、プロセスアクティビティまたはスコープに追加できます。詳細については、[「グローバル関連セットとローカル関連セット」 \(ページ 370\)](#) を参照してください。

関連セットを追加する手順

1. 定義したプロパティとプロパティエイリアスを含む WSDL ファイルが使用可能であることを確認します。詳細については、[「関連セットについて」 \(ページ 368\)](#) を参照してください。
2. 次のいずれかを実行します。
 - プロセスに関連セットを追加するには、[アウトライン] ビューで [関連セット] を右クリックし、[追加] > [宣言] > [関連セット] を選択します。
 - スコープに関連セットを追加するには、スコープを右クリックして、[追加] > [宣言] > [関連セット] を選択します。
3. 利用可能なプロパティリストからプロパティを選択して、選択された関連セットプロパティリストに移動します。



4. さらにプロパティが必要な場合は、それらのプロパティを適切なリストに移動します。

相関セット: 必須およびオプションのプロパティ

次の表に、相関セットに必須のプロパティとオプションのプロパティを示します。

必須のプロパティ	オプションのプロパティ
Set Name (セット名)	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79) を参照してください。
プロパティ。 「メッセージプロパティとプロパティエイリアスの作成」 (ページ 371) を参照してください。	コメント。 「プロセスへのコメントの追加」 (ページ 79) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

XML 構文

相関セットの XML 構文は次のとおりです。

```
<correlationSets?
  <correlationSet name="NCName" properties="QName-list"/>+
</correlationSets>
```

例

相関セットの宣言の例を次に示します。

```
<correlationSet name="PurchaseOrder"
  properties="cor:customerID cor:orderNumber"/>
```

相関セットの削除

この BPEL プロセスから相関セットを削除します。

プロセスまたはスコープから相関セットを削除できます。

プロセスから相関セットを削除する手順

1. Process Editor キャンバスの空白部分をクリックします。
2. Process Developer のメインメニューから **【プロセス】** を選択します。
3. **【削除】 > 【相関セット】** を選択します。
4. 相関セットを選択して、**【OK】** をクリックします。

スコープから相関セットを削除する手順

1. [アウトライン] ビューで、[スコープ] ノードを展開して相関セットを表示します。
2. 相関セットを右クリックして、**【削除】** を選択します。

アクティビティへの相関の追加

エンジンによって管理される相関と相関セットの使用に関する詳細については、[「相関セットおよびエンジンによって管理される相関」 \(ページ 382\)](#)を参照してください。

次の種類の Web サービスインタラクションアクティビティに相関を追加できます。

- 受信
- ピック (onMessage ブランチ)
- イベントハンドラ (onEvent バリエーション)
- invoke
- リプライ

アクティビティに相関を追加してプロセスファイルを保存すると、[問題] ビューで、相関が必要な関連するアクティビティに警告メッセージが表示されます。

設定が必要な相関属性の説明については、[「相関のパターンの開始と設定」 \(ページ 378\)](#)を参照してください。

相関を追加する方法については、[「Receive、OnMessage、OnEvent、または Reply への相関の追加」 \(ページ 380\)](#)および [「Invoke アクティビティへの相関の追加」 \(ページ 381\)](#)を参照してください。

相関のパターンの開始と設定

相関セットをアクティビティに追加します。

アクティビティで相関セットを使用する場合は、必ず開始属性を設定する必要があります。さらに、Invoke アクティビティにはパターン属性があります。

開始属性

開始属性には、*【はい】*、*【いいえ】*、または *【結合】* を指定できます。

[はい] は、アクティビティの実行時に、送信または受信されたメッセージの値で相関セットのプロパティエリアスが開始されることを意味します。WS-BPEL では、相関セットがすでに開始されている場合、標準の `bpel:correlationViolation` フォールトでフォールトするアクティビティが必要です。

複数のアクティビティが同時に有効になり、それらのアクティビティのいずれかによって相関セットが開始される可能性がある場合は、[結合] 属性を使用する必要があります。

[結合] は、アクティビティによって相関セットの開始が試行されることを意味します。[結合] は、複数開始アクティビティ、または複数のアクティビティによって相関セットが開始されるような場合に使用できます。

次のセクションでは、さまざまなアクティビティタイプについて説明します。

OnMessage と OnEvent の受信

これらのアクティビティはメッセージを受信します。

相関開始属性が [はい] で、相関セットが現在空の場合、プロセスは、これらのアクティビティのパーティシパントと操作に一致するすべてのメッセージを受け入れます。メッセージを受信した後に、相関セットが開始されます。相関セットが以前の Web サービスインタラクションからすでに開始されている場合、プロセスは、パーティシパントと操作に加えて相関プロパティと一致する場合にのみ、このアクティビティのメッセージを受け入れます。

Reply

Reply はデータを送信します。

- 相関開始属性が [はい] に設定されている場合、相関セットは、送信されたメッセージの値で開始されます。
- [いいえ] に設定されている場合は、現在相関セットにある値に照らし合わせてメッセージが検証され、このメッセージが正しいデータで送信されていることが確認されます。

Invoke

Invoke はデータを送信し、また、ターゲット Web サービスからの応答を受信することもできます。

Invoke アクティビティは、その二重の性質が示すように、相関での使用において追加の属性を持ちます。`pattern` 属性は、相関セットがいつ使用されるかを示します。

- `request` に設定されている場合、これは送信メッセージにのみ適用されます。この場合、Invoke の入力変数は、正しいデータが送信されていることを確認するために相関セットに照らし合わせて検証されます。
- `response` に設定されている場合、呼び出された Web サービスから入力された出力変数が相関セットのデータと一致することを確認します。

`request-response` に設定されている場合、データは出力時と入力時に検証されます。

開始前に相関セットが使用されている場合（つまり、開始属性が `no` の場合）、BPEL エンジン は `bpel:correlationViolation fault` フォールトをスローします。このエラーは、相関セットが一致しないデータの送信を検証しようとした場合にもスローされます。

相関セットの構文は次のとおりです。

```
<correlation set="NCName"
  initiate="yes|join|no"?
  pattern="request|response|request-response"? />
```

詳細については、「[Receive、OnMessage、OnEvent、または Reply への相関の追加](#)」（ページ 380）および「[Receive、OnMessage、OnEvent、または Reply への相関の追加](#)」（ページ 380）を参照してください。

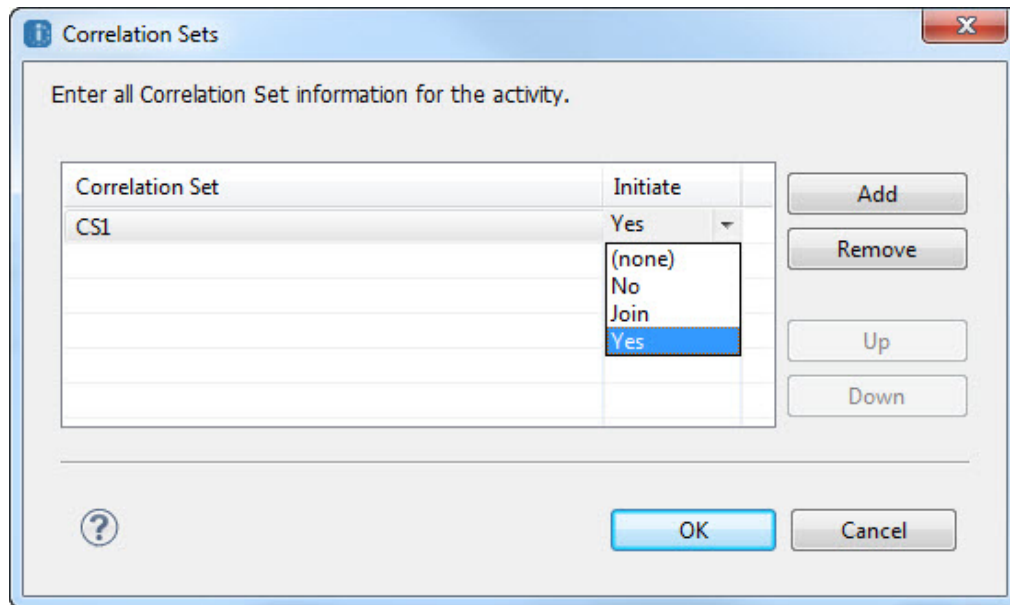
Receive、OnMessage、OnEvent、または Reply への相関の追加

相関セットをアクティビティに追加します。

エンジンによって管理される相関と相関セットの使用に関する詳細については、「[相関セットの追加](#)」(ページ 376)を参照してください。相関を開始するためのヘルプについては、「[相関セットおよびエンジンによって管理される相関](#)」(ページ 382)を参照してください。

相関をアクティビティに追加する前に、相関セットを作成します。詳細については、「[相関セットの追加](#)」(ページ 376)を参照してください。

1. Process Editor キャンバスで、Pick アクティビティの Receive、OnMessage ブランチ、イベントハンドラの OnEvent バリエーション、または Reply を選択します。
2. [プロパティ] ビューで、[ダイアログ (...)] をクリックします。
3. リストから相関セットを選択します。
4. [開始] リストから次のいずれかを選択します。
 - [はい] を選択すると、このアクティビティで相関が開始されます。
 - [結合] を選択すると、まだ開始されていない場合に、このアクティビティで相関が開始されます。これは、フローまたはスコープ内の複数の Receive など、複数開始アクティビティに役立ちます。受信ごとに [結合] を選択します。
 - [いいえ] は、このアクティビティで相関を開始しない場合に選択します。



5. 必要に応じて [追加] をクリックして、このアクティビティに別の相関セットを追加します。
6. 必要に応じて、相関セットをリスト内で上下に移動して再編成します。相関セットの順序は、処理において重要ではありません。必要に応じてリストを整理します。

例

```
<receive createInstance="yes" name="SellerReceive"
  operation="submit" partnerLink="seller"
  variable="sellerInfo" >
  <correlations>
    <correlation initiate="yes" set="negotiationIdentifier"/>
  </correlations>
</receive>
```

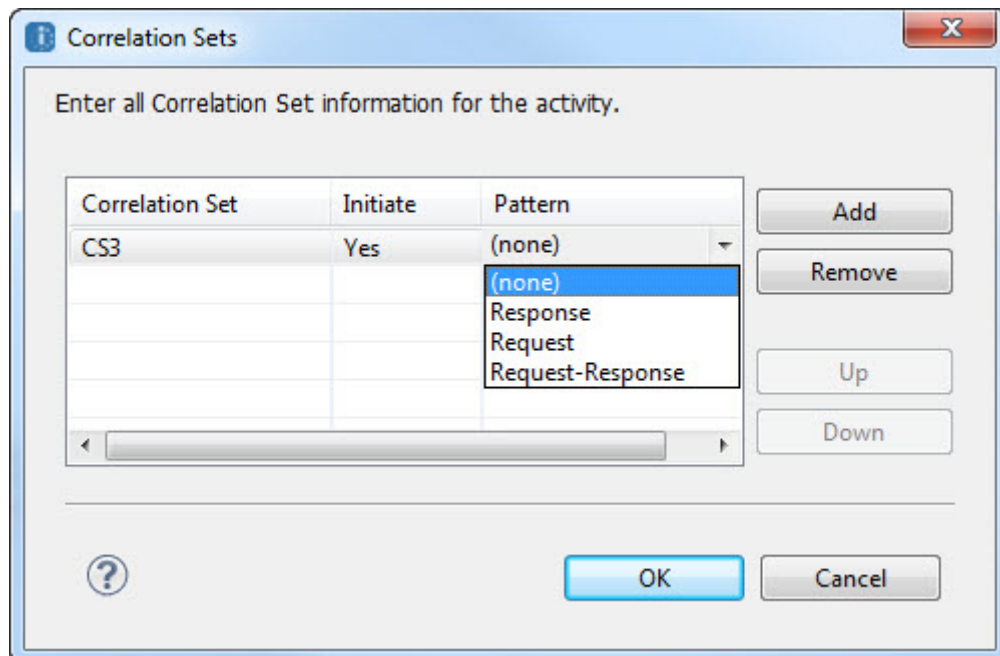
Invoke アクティビティへの相関の追加

相関セットをアクティビティに追加します。

相関をアクティビティに追加する前に、相関セットを作成します。詳細については、「[相関セットの追加](#)」(ページ 376)を参照してください。追加のヘルプについては、相関に関する「[相関のパターンの開始と設定](#)」(ページ 378)と「[Invoke](#)」(ページ 379)を参照してください。

Invoke アクティビティに入力メッセージおよび出力メッセージがある場合は、相関セットにパターンを選択する必要があります。Invoke アクティビティに入力メッセージのみがある場合、相関はアウトバウンドメッセージを検証するためだけに使用されます。

1. Process Editor キャンバスで、Invoke アクティビティを選択します。
2. [プロパティ] ビューで、[ダイアログ (...)] をクリックします。
3. リストから相関セットを選択します。
4. [開始] リストから次のいずれかを選択します。
 - [はい] を選択すると、このアクティビティで相関が開始されます。
 - [結合] を選択すると、まだ開始されていない場合に、このアクティビティで相関が開始されます。これは、フローまたはスコープ内の複数の Receive など、複数開始アクティビティに役立ちます。受信ごとに [結合] を選択します。
 - [いいえ] は、このアクティビティで相関を開始しない場合に選択します。
5. パターンを次のいずれかに設定します。
 - [要求] を選択すると、入力メッセージが相互に関連付けられます。
 - [応答] を選択すると、出力メッセージが相互に関連付けられます。
 - [要求-応答] を選択すると、両方のメッセージが相互に関連付けられます。これは、入力メッセージと出力メッセージが同じやりとりの一部であり、どちらにも同じプロパティのプロパティエイリアスが含まれている場合に役立ちます。



6. 必要に応じて [追加] をクリックして、このアクティビティに別の相関セットを追加します。

例

```
<invoke partnerLink="Seller" operation="SyncPurchase"
  inputVariable="sendPO"
  outputVariable="getResponse">
  <correlations>
    <correlation initiate="yes" set="PurchaseOrder"
      pattern="request">
    <correlation initiate="yes" set="Invoice"
      pattern="response">
  </correlations>
</invoke>
```

欠落した相関の追加

欠落した相関セットをアクティビティに追加します。

相関を開始して BPEL ファイルを保存すると、Process Developer によって、相関が必要なプロセス内の各アクティビティの問題ビューに警告が追加されます。

警告を表示するには、**[ウィンドウ] > [設定] > [Process Developer] > [タスク]** で警告タスクを有効にする必要があります。

相関セットを宣言および使用するためのルール

相関セットを宣言して使用するための一部のルールを以下に示します。

- 相関セットは、変数の宣言と同様に、スコープ内で宣言します。グローバル相関セットおよびローカル相関セットを宣言することができます。
- 同じ名前の相関セットを内側のスコープで宣言することにより、外側のスコープで相関セットを非表示にできます。スコープに対してローカルで相関セットを宣言する場合、宣言はスコープ内のアクティビティにのみ適用されます。
- 相関セットは、その相関セットが属するスコープの存続期間中に 1 度だけ開始できます。
- 相関セットは開始する必要があります。
- 相関セットの開始後、相関セットのプロパティの値は、スコープ内にあるすべての操作のすべてのメッセージで同一である必要があります。

相関セットおよびエンジンによって管理される相関

インバウンドメッセージを相互に関連付けるには、2 つの方法があります。以下を実行できます。

- 相関セットを作成して使用するか、Process Developer エンジンがメッセージ相関を自動的に管理できるようにすることができます。
- プロセスデプロイメント記述子ファイルのマイロールパートナーリンクにポリシーを追加することで、receive、onMessage、または onEvent を対象とするインバウンドメッセージが Process Developer エンジンで自動的に相互に関連付けられるように指定します。

詳細については、[「エンジンによって管理される相関」 \(ページ 473\)](#)を参照してください。

第 25 章

フォールト処理

次のトピックでは、フォールト処理について説明します。

- [「BPEL フォールト処理について」 \(ページ 383\)](#)
- [「Catch および CatchAll フォールトハンドラの定義」 \(ページ 384\)](#)
- [「サービス呼び出しのフォールト処理」 \(ページ 386\)](#)
- [「フォールトハンドラの追加」 \(ページ 388\)](#)
- [「フォールト処理の処理ルール」 \(ページ 387\)](#)
- [「Catch アクティビティでフォールトをキャッチするためのルール」 \(ページ 393\)](#)
- [「フォールト処理に関するヒント」 \(ページ 394\)](#)
- [「宣言されていないフォールトと SOAP フォールトのキャッチ」 \(ページ 395\)](#)

BPEL フォールト処理について

BPEL プロセスでのフォールト処理とは、通常の処理中に例外または予期しない状態が発生した場合に実行する手順を指します。実行した作業の状態が復元されるため、予測可能な結果が常に発生します。フォールト処理は補償処理とは異なり、スコープ内でフォールトが発生した場合に生じるのがフォールト処理であるのに対し、正常に完了したスコープの作業を元に戻すための処理を行うのが補償処理です。これは、フォールトハンドラがスコープ内に含まれる情報にアクセスすることを意味します。

次のような場合に、フォールトが発生します。

- Web サービス操作を正常に完了できず、サービスがフォールトを返した場合
- 内部プロセスエラーが発生し、標準の BPEL フォールトがスローされた場合。詳細については、「*Process Developer* のヘルプ」にある「*BPEL 標準フォールト*」のリストを参照してください。
- <throw>または<rethrow>アクティビティがフォールトをスローした場合
- 通信障害などのプラットフォーム固有のフォールトが BPEL プロセスインスタンスで発生した場合

フォールト処理はグローバルまたはローカルに設定することができます。つまり、プロセス全体またはプロセス内のスコープにフォールトハンドラを追加できます。

フォールトが発生すると通常の処理が終了し、プロセスまたはスコープの<faultHandlers>セクションで定義したように、対応するフォールトハンドラに制御が移ります。

このプロセスでは、フォールトハンドラが呼び出されたスコープの補償は有効にならないことに注意してください。

フォールトハンドラは、ネストされたスコープの状態に依存せず、正常に完了したかどうかを判断します。

スコープのフォールト処理と補償処理の説明については、[「スコープのライフサイクル」 \(ページ 239\)](#)を参照してください。

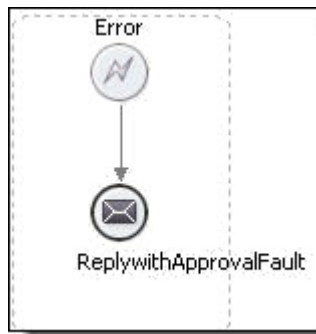
Catch および CatchAll フォールトハンドラの定義

次の2種類のフォールトハンドラを定義できます。

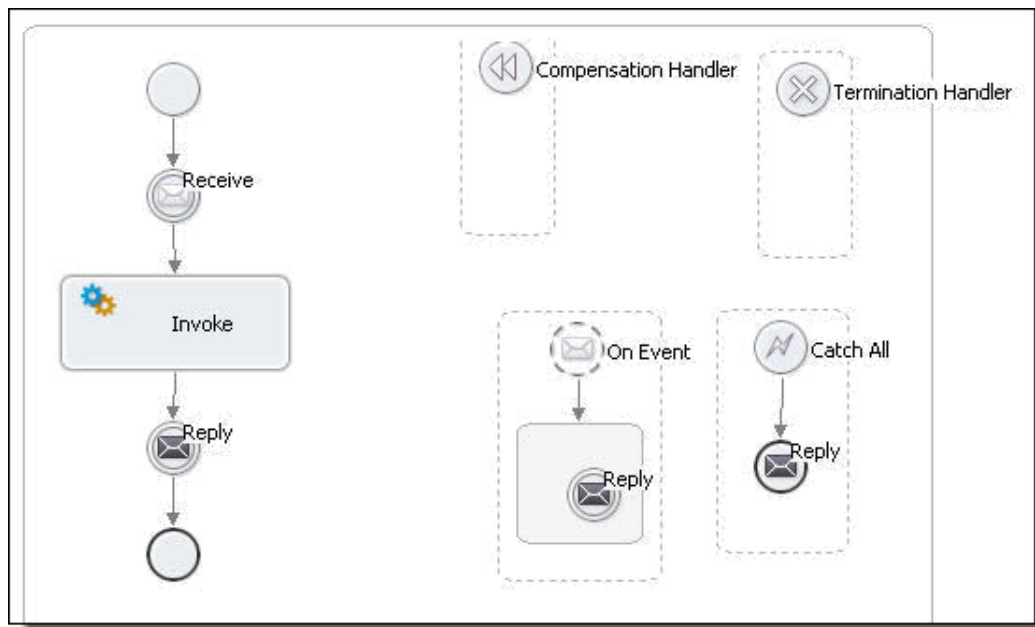
- オプションのフォールト名やフォールト変数に基づいて実行される一連のカスタムフォールト処理アクティビティを定義する<catch>フォールトハンドラ。フォールト名が欠落している場合、Catch は同じタイプのフォールトデータを持つすべてのフォールトをインターセプトします。
- スローされたフォールトが<catch>フォールトハンドラによってキャッチされない場合に実行される<catchAll>フォールトハンドラ。CatchAll は Catch の派生形です。CatchAll では、キャッチするフォールト名または変数は指定されません。

[「フォールトハンドラの追加」 \(ページ 388\)](#)に記載されているように、Process Editor の [フォールトハンドラ] タブで、プロセスに<catch>ハンドラと<catchAll>ハンドラを定義できます。

次の図は、Process Editor キャンバスで作成されたフォールトハンドラの例を示しています。



次の図に示すように、*Catch All*というラベルの付いたエラーキャッチイベントのように、スコープにフォールトハンドラを定義することもできます。



それぞれの Catch アクティビティを定義することで、グローバルな一意のフォールト名およびフォールトに関連付けられた変数によって定義された特定の種類のフォールトをインターセプトできます。必要に応じて、メッセージタイプまたは要素タイプのいずれかのフォールト変数に、定義済みのタイプを関連付けることができます。

XML 構文

```
<faultHandlers>?
  <catch faultName="QName"?
    faultVariable="BPELVariableName"?
    (faultMessageType="QName" | faultElement="QName")?*>
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
</faultHandlers>
```

必須のプロパティ	オプションのプロパティ
Catch のみ: フォールト名のみ、フォールト変数のみ、またはフォールト名とフォールト変数	Catch のみ: フォールト名、フォールト変数の定義、フォールト変数
	コメント。 「プロセスへのコメントの追加」 (ページ 79)を参照してください。
	ドキュメント。 「プロセスへのドキュメントの追加」 (ページ 79)を参照してください。
	「視覚的なプロパティの設定と独自のイメージライブラリの使用」 (ページ 76)を参照してください。

必須のプロパティ	オプションのプロパティ
	実行状態。 「アクティビティまたはリンクの実行状態の表示」 (ページ 419) を参照してください。
	拡張属性と拡張要素。 「拡張要素と属性の宣言」 (ページ 126) を参照してください。

フォールト名とフォールト変数をキャッチする

フォールト名とフォールト変数/変数定義はオプションの属性ですが、<catch>には少なくともそれらのうちの 1 つを指定する必要があります。こうした属性は、フォールトがスローされたときに Catch または Catch All のどちらのフォールトハンドラを実行するかを決定します。

フォールト名および変数を定義しない場合、Catch は<catchAll>として実行されます。

多く場合、これらの属性の値は、呼び出されているサービスの WSDL からの値です。フォールトの名前は通常、対象の名前空間の値としてターゲット名前空間を使用する操作のフォールトの名前です。フォールトの変数は通常、WSDL での操作によってフォールトがスローされたときに送信されるメッセージです。キャッチにフォールト変数が含まれている場合、その変数はキャッチ内で定義する必要があります。この変数には、Catch の実行時にフォールトからのデータが入力されます。さらに、Process Developer では、宣言されていないフォールトに対して特別な処理を使用できます。

フォールト名と変数の使用に関する詳細については、次のトピックを参照してください。

- [「フォールト処理の処理ルール」](#) (ページ 387)
- [「フォールト処理に関するヒント」](#) (ページ 394)
- [「宣言されていないフォールトと SOAP フォールトのキャッチ」](#) (ページ 395)

CatchAll ハンドラの例

<catchAll>は、既存の Catch ブロックによってキャッチされなかったフォールトをキャッチします。この構成の利点として、フォールトがスローされた場合に、フォールトハンドラがフォールト処理コードを確実に実行するようにできることが挙げられます。欠点としては、<catchAll>自体が使用可能なフォールト名またはフォールト変数はないため、スローされたフォールトのタイプに関する詳細は不明であるということが挙げられます。ただし、Process Developer では、<catchAll>で宣言されていないフォールトを処理するためのカスタム関数を使用できます。この構成は、スローされるフォールトの種類が重要ではない場合、またはフォールトが既存のフォールトハンドラロジックを通過しないことを確認する場合に適しています。

デフォルトの<catchAll>フォールトハンドラは次のとおりです。

```
<catchAll>
  <sequence>
    <compensate/>
    <rethrow/>
  </sequence>
</catchAll>
```

サービス呼び出しのフォールト処理

Invoke アクティビティへのフォールト応答は、WSDL 操作でのフォールトの定義を使用します。WSDL フォールトは、ポートタイプのターゲット名前空間とフォールト名を使用して定義します。

例: 発注書の BPEL プロセス

通常の発注書の BPEL プロセスでは、以下のステップが発生します。

1. 顧客が発注書を送信する。
2. BPEL プロセスが発注書を受け取り、送料と製造コストを計算するためのタスクを実行する。
3. BPEL プロセスが、顧客に請求書を返す。

何らかの問題が発生した場合、プロセスではエラーが返されます。フォールトメッセージは WSDL ファイルで定義し、次の例で説明するように BPEL プロセスで使します。

WSDL フォールト定義	サービス呼び出しの BPEL フォールト処理
<pre><portType name="POrderPT"> <operation name="sendPOrder"> <input message="..." /> <output message="..." /> <fault name="cannotCompletePO" message="pos:orderFaultType"/> </operation> </portType></pre>	<pre><faultHandlers> <catch faultMessageType="ns1:errorMessage" faultName=="lns:cannotCompletePO" faultVariable="POFault"> <reply partnerLink="purchasing" operation="sendPOrder" variable="POFault" faultName="lns:cannotCompletePO"/> </catch> </faultHandlers></pre>

上記の WSDL 定義では、sendPOrder という操作によって、発注書を完了できない場合に BPEL プロセスによって使用される cannotCompletePO という名前のフォールトを指定しています。BPEL プロセスのフォールトハンドラセクションの Catch アクティビティには、フォールトが受信された場合に POFault 変数へメッセージを返すように定義された応答が含まれています。

呼び出し操作でフォールトメッセージが返されると、現在のスコープでフォールトが発生します。対応する <catch> のフォールト変数は、受信したフォールトメッセージで初期化されます。

フォールトメッセージで Web サービス操作が定義されていない場合は、Process Developer カスタム関数を使用して、宣言されていないフォールトをキャッチできます。

詳細については、「[呼び出しアクティビティの境界イベントとしてのフォールトハンドラの追加](#)」 (ページ 393) および「[呼び出しアクティビティの境界イベントとしてのフォールトハンドラの追加](#)」 (ページ 393) を参照してください。

フォールト処理の処理ルール

フォールト処理の処理ルール

プロセスでフォールトが発生すると、実行フローは、フォールトを生成したアクティビティ（例えば、実行中にフォールトが発生した Ivoke や Throw アクティビティ）から、エンクローズされたスコープのフォールトハンドラにすぐに移動します。エンクローズされたスコープがすぐ近くにはない場合、実行はプロセスのフォールトハンドラに移動します。

フォールトハンドラに到達すると、単一の <catch> または <catchAll> が実行のために照合されます。フォールトを照合するためのルールは次のとおりです。

- を実行するための複数のルール、およびキャッチの一致の優先順位スキームがあります。<catch> 詳細については、「[Catch アクティビティでフォールトをキャッチするためのルール](#)」 (ページ 393) を参照してください。
- 上記のルールを使用してフォールトが <catch> に一致しない場合は、<catchAll> が実行されます。

- フォールトハンドラ内に<catchAll>がない場合、暗黙的なフォールト処理ロジックが実行されます。このロジックは、エンクローズされたすべてのスコープに対してデフォルトの補償ルーチンを実行する単一の補償アクティビティを含む<catchAll>が存在するように機能します。実行されると、元のフォールトは、次の包含スコープ、または使用可能なものがない場合はプロセスに再スローされます。ハンドラがすでにプロセスレベルである場合、プロセスはフォールトで終了します。

一致した<catch>または<catchAll>が実行される前に、スコープ内のすべてのアクティビティが終了します。スコープにフォールトが発生するとスコープは正常に完了しなかったと見なされるため、その実行に対する補償の対象にはなりません。スコープがフォールトを再スローせずにフォールトをキャッチした場合、通常のプロセス実行はスコープのポイントから再開されます。これがプロセスレベルで発生した場合、プロセスは正常に完了しますが、プロセスインスタンスの補償の対象にはなりません。

次の例では、フォールト名とフォールト変数が Catch アクティビティ全体で一意ではないことに注意してください。

例

```
<faultHandlers>
  <!-- catch all faults with a matching name, but no data -->
  <catch faultName="x:foo">
    <empty/>
  </catch>
  <!-- catch all faults with the matching variable type,
        whose name is not "x:foo"-->
  <catch faultVariable="bar"
        faultMessageType="tns:barType">
    <empty/>
  </catch>
  <!-- catch the fault specified by the name
        and variable type -->
  <catch faultName="x:foo" faultVariable="bar"
        faultMessageType="tns:barType">
    <empty/>
  </catch>
  <!-- catch all faults not caught by a specific handler -->
  <catchAll>
    <empty/>
  </catchAll>
</faultHandlers>
```

Rethrow アクティビティとフォールト処理

BPEL プロセスは、<rethrow>アクティビティを使用して、エンクローズされた最も近くにあるフォールトハンドラによってキャッチされた元のフォールトを再スローします。<rethrow>アクティビティは、任意のフォールトハンドラ内で使用できます。フォールトのキャッチ方法やフォールトハンドラがフォールトデータを変更するかどうかに関係なく、<rethrow>アクティビティは常に元のフォールトデータをスローし、そのタイプを保持します。

リンクとフォールト処理

フォールトハンドラの境界を越えるリンクはアウトバウンドである必要があります。つまり、ソースアクティビティがフォールトハンドラ内にあり、ターゲットはフォールトハンドラに関連付けられたスコープをエンクローズするスコープ内にある必要があります。

フォールトハンドラの追加

メッセージタイプまたは要素のいずれかの変数定義を選択します。次に、リストから変数を選択します。

BPEL プロセスは、標準フォールトだけでなく、Throw または Rethrow アクティビティまたはサービス呼び出しからのフォールトもキャッチします。キャッチされたフォールトは、エラーメッセージを含む Reply または

Compensate アクティビティなど、指定した Catch アクティビティによって処理されます。<catch>または <catchAll>は単一の子アクティビティのみを持つことができますが、このアクティビティは、独自の子を持つ可能性のある構造化されたアクティビティである場合もあります。

フォールトハンドラは、次のような複数の方法で追加できます。

- [「プロセスへのフォールトハンドラの追加」](#) (ページ 389)
- [「スコープへのフォールトハンドラの追加」](#) (ページ 390)
- [「呼び出しアクティビティの境界イベントとしてのフォールトハンドラの追加」](#) (ページ 393)

背景情報については、[第 25 章, 「フォールト処理」](#) (ページ 383)を参照してください。

プロセスへのフォールトハンドラの追加

プロセスには複数の Catch ハンドラと 1 つの Catch-All ハンドラを追加できます。

これらのハンドラの説明については、[「Catch および CatchAll フォールトハンドラの定義」](#) (ページ 384)および [「宣言されていないフォールトと SOAP フォールトのキャッチ」](#) (ページ 395)を参照してください。

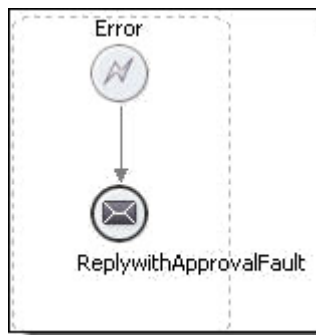
プロセスに Catch ハンドラを追加する手順

1. Process Editor キャンバスの [フォールトハンドラ] タブをクリックします。
2. [エラーキャッチイベント] をキャンバスにドラッグします。
Catch ハンドラが追加され、選択された状態になります。
3. [プロパティ] ビューで、次のオプション値を選択します。
 - a. フォールト名。 [「フォールト処理の処理ルール」](#) (ページ 387)を参照してください。
 - b. フォールト変数。変数定義で記述されている変数の名前を入力します。フォールト変数を追加するかどうかに関する説明については、[「Catch アクティビティでフォールトをキャッチするためのルール」](#) (ページ 393)を参照してください。フォールト名を選択した場合、フォールト変数の選択リストには互換性のある変数のみが表示されます。
 - c. 変数定義。行の最後にある [ダイアログ (...)] ボタンを選択して、変数定義ダイアログを開きます。
[「フォールト処理の処理ルール」](#) (ページ 387)を参照してください。
4. Reply や Compensate などのアクティビティを Catch ハンドラにドラッグします。例については、[「サービス呼び出しのフォールト処理」](#) (ページ 386)を参照してください。
5. フォールトを処理するアクティビティのプロパティを入力します。

プロセスに Catch All ハンドラを追加する手順

1. Process Editor キャンバスの [フォールトハンドラ] タブをクリックします。
2. [エラーキャッチイベント] をキャンバスにドラッグします。
3. [すべてのフォールトをキャッチ] チェックボックスを選択します。
4. Catch の [プロパティ] ビューでは、フォールト名または変数を選択しないでください。
5. Assign や Compensate といったアクティビティを Catch All ハンドラにドラッグします。
6. フォールトを処理するアクティビティのプロパティを入力します。

次の図は、プロセスに追加したフォールトハンドラの例を示しています。



ヒント: フォールトハンドラを折りたたむと、[プロパティ] ビューから背景色を追加できます。ハンドラを右クリックして、[コンテナの折りたたみ] を選択します。

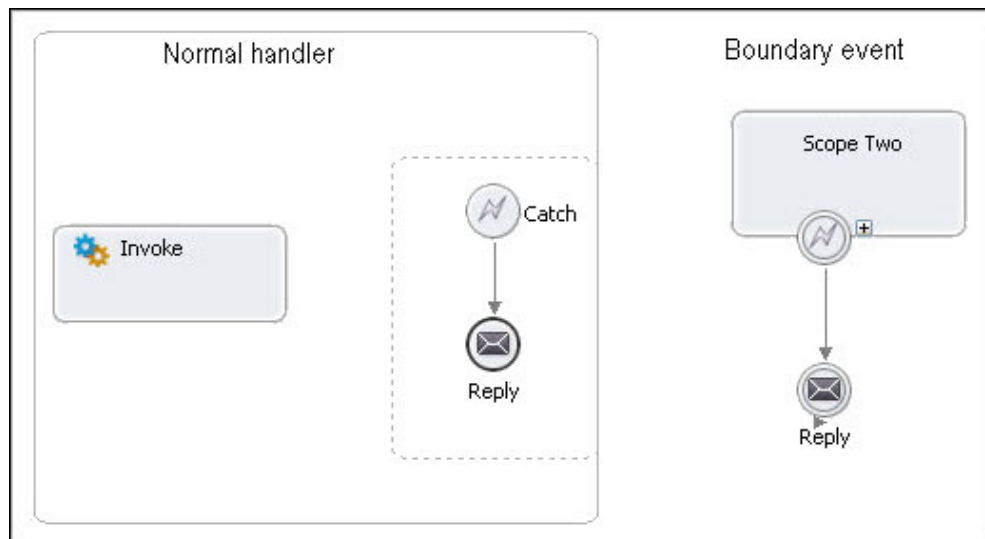
スコープへのフォールトハンドラの追加

プロセス内のスコープごとに、複数の Catch ハンドラと 1 つの Catchall ハンドラを追加できます。

これらのハンドラの説明については、[「Catch および CatchAll フォールトハンドラの定義」 \(ページ 384\)](#) および [「Catch および CatchAll フォールトハンドラの定義」 \(ページ 384\)](#) を参照してください。

スコープに Catch ハンドラを追加する手順

1. Process Editor キャンバスの [プロセスアクティビティ] タブで Scope コンテナを選択します。
2. 図に示すように、[エラーキャッチイベント] をスコープ内または折りたたまれたスコープの境界近くにドラッグします。



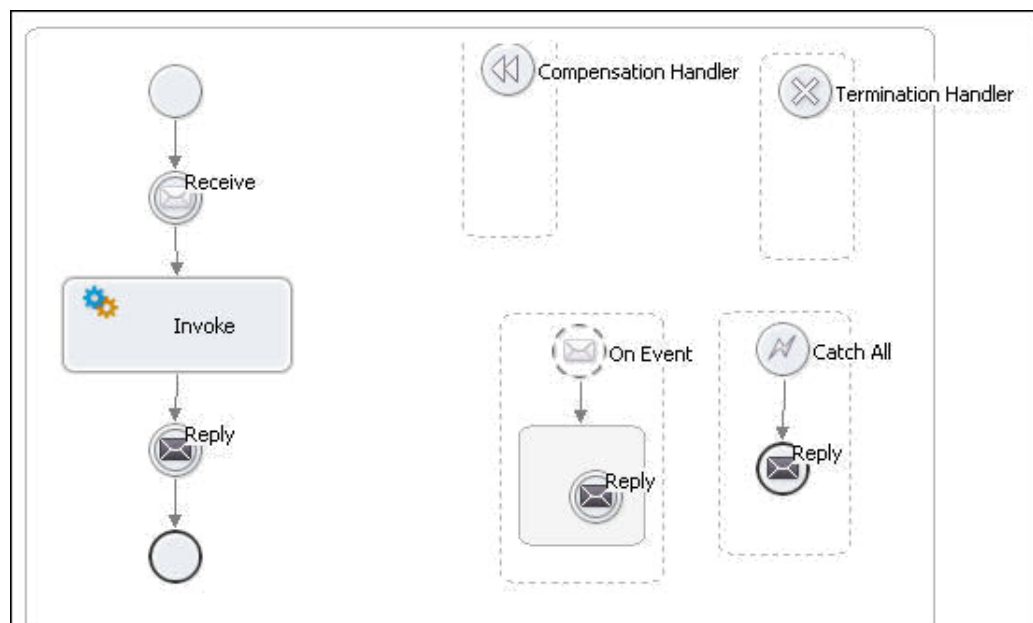
3. [プロパティ] ビューで、次のオプション値を選択します。
 - a. **【すべてのフォールトをキャッチ】** チェックボックスを選択し、他のプロパティは空白のままにします。
 - b. **フォールト名**。詳細については [「フォールト名の選択」 \(ページ 392\)](#) を参照してください。フォールト名を追加するかどうかに関する説明については、[「フォールト処理の処理ルール」 \(ページ 387\)](#) および [「フォールト名の選択」 \(ページ 392\)](#) を参照してください。

- c. **フォールト変数**。変数定義で記述されている変数の名前を入力します。フォールト変数を追加するかどうかに関する説明については、[「フォールト処理の処理ルール」](#) (ページ 387) および [「フォールト処理の処理ルール」](#) (ページ 387) を参照してください。フォールト名を選択した場合、フォールト変数の選択リストには互換性のある変数のみが表示されます。
 - d. **変数定義**。行の最後にある [ダイアログ (...)] ボタンを選択して、変数定義ダイアログを開きます。詳細については [「フォールト変数定義の追加」](#) (ページ 392) を参照してください。フォールト変数を追加するかどうかに関する説明については、[「フォールト処理の処理ルール」](#) (ページ 387) および [「フォールト変数定義の追加」](#) (ページ 392) を参照してください。
4. Reply などのアクティビティを Catch ハンドラにドラッグします。境界イベントの場合、実行するアクティビティにハンドラをリンクする必要があります。このアクティビティは、まだ実行されていないアクティビティである必要があります。
 5. 境界イベントを使用している場合、特定のアクティビティは使用できません。[「Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow」](#) (ページ 365) を参照してください。
 6. フォールトを処理するアクティビティのプロパティを入力します。

スコープに Catch All ハンドラを追加する手順

1. Process Editor キャンバスの [プロセスアクティビティ] タブでスコープを選択します。
2. **エラーキャッチイベント** をスコープ内またはその境界近くにドラッグします。
3. [プロパティ] ビューで、**すべてのフォールトをキャッチ** チェックボックスを選択します。
4. Assign や Compensate といったアクティビティを Catch All ハンドラにドラッグします。境界イベントを使用している場合、特定のアクティビティは使用できません。[「Catch および Catch All 境界イベント、Compensate、Compensate Scope、Rethrow」](#) (ページ 365) を参照してください。
5. フォールトを処理するアクティビティのプロパティを入力します。

次の図は、スコープに追加した Catch All フォールトハンドラの例を示しています。



ヒント: フォールトハンドラを折りたたむと、[プロパティ] ビューから背景色を追加できます。ハンドラを右クリックして、**コンテナの折りたたみ** を選択します。

フォールト変数定義の追加

メッセージタイプまたは要素のいずれかの変数定義を選択します。次に、リストから変数を選択します。

Catch コンテナの [プロパティ] ビューから [フォールト変数定義の追加] ダイアログを開きます。

1. 使用可能なすべてのメッセージのリストを表示するには、[メッセージタイプ] を選択します。
2. メッセージに単一のパートがあり、そのパートが要素タイプで定義されている場合は、[要素] を選択します。
3. 変数定義を選択した後に、Catch の [プロパティ] ビューに戻って、ローカルのキャッチ変数に変数名を入力します。

インバウンドフォールトメッセージの受信後、Catch でエンクローズされたアクティビティの実行に進む前に、フォールトハンドラによってインバウンドフォールトメッセージが変数に割り当てられます。この変数はフォールトハンドラ内で宣言されているため、ローカル変数が宣言スコープの外部にアクセスできないのと同様に、フォールトハンドラの外部にはアクセスできません。

詳細については、次のトピックを参照してください。

- [「フォールトハンドラの追加」 \(ページ 388\)](#)
- [「フォールト処理の処理ルール」 \(ページ 387\)](#)
- [「Catch アクティビティでフォールトをキャッチするためのルール」 \(ページ 393\)](#)
- [「フォールト処理に関するヒント」 \(ページ 394\)](#)
- [「宣言されていないフォールトと SOAP フォールトのキャッチ」 \(ページ 395\)](#)

フォールト名の選択

フォールトタイプと変数に基づいてフォールト名を定義します。

Catch または Throw アクティビティには、[プロパティ] ビューで開くことができるフォールト名プロパティがあります。[フォールト名] ダイアログでは、次のような選択を行うことができます。

- **スコープのフォールト**
スコープのフォールト名は通常、呼び出されるサービスの WSDL から決定されます。フォールトの名前は通常、対象の名前空間の値としてターゲット名前空間を使用する操作のフォールトの名前です。プロセスでユーザーアクティビティ（オンプレミスのみ）を使用する場合、複数の標準的なヒューマンタスクフォールトが含まれています。リストからフォールトを選択します。
- **標準フォールト**
標準フォールトは、WS-BPEL 2.0 仕様で定義されているフォールトです。リストからフォールトを選択します。このヘルプの他の場所にある「BPEL 標準フォールト」も参照してください。
- **カスタムフォールト**
カスタムフォールトはユーザーが作成したフォールトで、必要に応じてプレフィックスを付けたり、名前空間を含めたりすることができます。宣言されていないフォールトをキャッチする場合は、カスタムフォールト名を使用する必要があります。

フォールト名は、Throw アクティビティでは必須のプロパティで、Catch アクティビティではオプションのプロパティです。詳細については、次のトピックを参照してください。

- [「フォールトハンドラの追加」 \(ページ 388\)](#)
- [「フォールト処理の処理ルール」 \(ページ 387\)](#)
- [「フォールト処理に関するヒント」 \(ページ 394\)](#)
- [「宣言されていないフォールトと SOAP フォールトのキャッチ」 \(ページ 395\)](#)

呼び出しアクティビティの境界イベントとしてのフォールトハンドラの追加

呼び出しアクティビティに入力メッセージおよび出力メッセージが含まれる場合は、フォールトハンドラを追加して、出力メッセージエラーを取得できます。

フォールトハンドラを呼び出しアクティビティに追加する手順

1. Process Editor キャンバスの「プロセスアクティビティ」タブを表示します。境界イベントを使用するには、BPMN Process Editor を使用している必要があります。
2. 「サービス」タスクを図面キャンバスにドラッグします。
3. アクティビティの境界にフォールトハンドラ（またはイベントあるいは補償ハンドラ）を追加します。
4. [「スコープへのフォールトハンドラの追加」](#)（ページ 390）の手順に従います。

Catch アクティビティでフォールトをキャッチするためのルール

Catch アクティビティには、フォールト名やフォールト変数を含めることができます。フォールト変数は、WSDL メッセージタイプまたはスキーマ要素として定義できます。スキーマ要素は、WSDL のスキーマで定義された置換グループのメンバーにすることや、要素として定義された単一パートのメッセージにすることができます。

WS-BPEL のフォールト照合ロジックは、最良の一致方式を使用します。これは、使用可能なキャッチ要素を分析して、特定のフォールトに最適なキャッチが決定されることを意味します。これを、最初の一致方式を使用する Java などの言語のフォールト照合ロジックと比較してみましょう。「最良の一致」という方式であるため、フォールトハンドラ内の要素の順序とは関係なく、一致する Catch 要素が照合されます。

複数のキャッチが特定のフォールトに一致する可能性があります。この場合、優先度の高いルールに一致するキャッチが選択されます。同じルールによって 2 つの Catch 要素が選択された場合、最良の一致では、置換グループの階層という観点からフォールト要素に近い要素変数を宣言する Catch 要素によってキャッチが決定されます。

優先システムは、最初に、キャッチにフォールト名のみがあるのか、データ付きのフォールト名があるのかを確認します。最優先されるのは、フォールト名のみでデータを含まないキャッチです。

フォールトデータを含むキャッチの場合、Process Developer は、次の表に記載されている順序でキャッチを実行します。

キャッチの順序	選択したフォールトハンドラオプション	変数タイプの定義	フォールト変数からのデータ
フォールト名でフォールトをキャッチ(変数データがない場合):			
1	フォールト名のみ	なし	なし
変数データでフォールトをキャッチ:			
2	フォールト名と変数	メッセージまたは要素	メッセージまたは要素
3	フォールト名と変数	要素	要素（置換グループ）

キャッチの順序	選択したフォールトハンドラオプション	変数タイプの定義	フォールト変数からのデータ
4	フォールト名と変数	要素	単一パートの要素メッセージ
5	フォールト名と変数	要素	単一パートの要素メッセージ（置換グループ）
6	フォールト名のみ	なし	メッセージまたは要素は無視されますが、スコープをエンクローズするように再スローされます
7	変数のみ	メッセージまたは要素	メッセージまたは要素
8	変数のみ	要素	要素
9	変数のみ	要素	単一パートの要素メッセージ
10	変数のみ	要素	単一パートの要素メッセージ（置換グループ）

Catch/Catch All の実行ルールに関する詳細については、[「フォールト処理の処理ルール」](#)（ページ 387）を参照してください。

フォールト処理に関するヒント

次のヒントに注意してください。

- 宣言された WSDL フォールトの名前およびそのフォールトに関連するデータは、<catchAll>では使用できません。この情報にアクセスする必要がある場合は、必ず特定の<catch>にフォールト名とフォールト変数を指定してください。
- <rethrow>アクティビティを使用して、<catch>または<catchAll>に一致するフォールトを再スローできます。フォールト名および変更されていないフォールトデータを含む、元のフォールトデータが再スローされます。
- サービスの WSDL で定義したフォールト以外のフォールトも使用できます。<throw>アクティビティを定義して、独自に作成したフォールト名と変数を使用できます。
- 標準のランタイム BPEL フォールトは、プロセスの設計のエラー（例えば、2 つの受信が同じパートナーリンクで同時に実行され、操作によって bpel:conflictingReceive が発生する場合など）、および標準のランタイムエラーが原因で発生する可能性があります。通常、これらのフォールトは、プロセスのシミュレーション中またはテスト中にキャッチされる必要のあるエラーを表すため、一般に、これらのフォールトのキャッチは適切な設計手法であるとは言えません。これにはいくつかの例外があります。例えば、bpel:joinFailure は、入力リンクが false であるためにアクティビティが実行されないという状況を識別する場合に役立ちます。

宣言されていないフォールトと SOAP フォールトのキャッチ

標準のフォールト処理では、フォールトメッセージは WSDL 操作の一部として宣言されています。この宣言はベストプラクティスであり、BPEL フォールト処理において想定される動作です。ただし、Web サービスにはベストプラクティスが常に存在するとは限らず、フォールトメッセージが欠落している可能性があります。サービスにフォールトが発生した場合、プロセスでは、原因に関する情報がないフォールトが発生します。こうした動作を改善するために、Process Developer では、宣言されていないフォールトに対するフォールト処理が提供されます。

宣言されていないフォールトと SOAP フォールトをキャッチするには、独自のフォールトを宣言してから、Process Developer のフォールトカスタム関数を使用して WSDL 以外のフォールトを処理します。

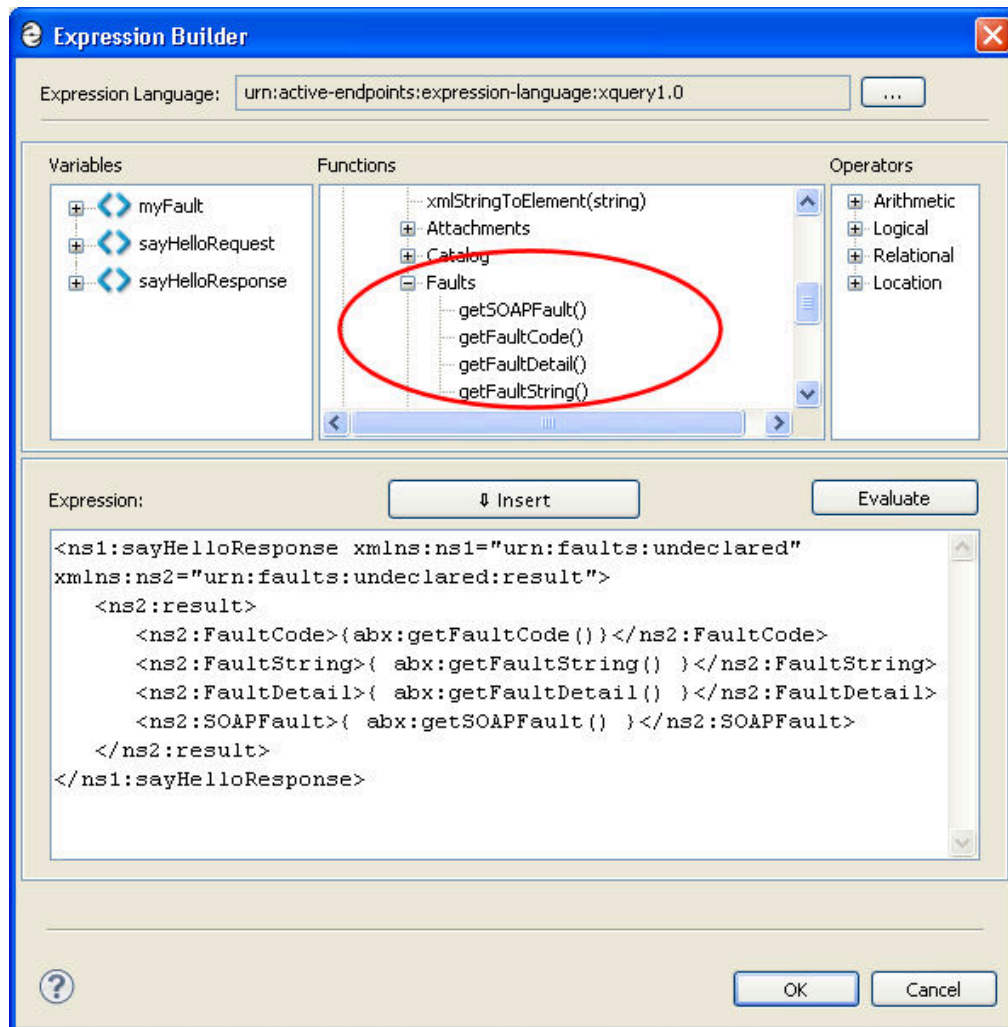
Process Developer のカスタム関数を使用することで、プロセスの終了を防止し、次のようなフォールト情報にアクセスすることができます。

- `getFaultCode()`: フォールトのカテゴリを返します
- `getFaultString()`: フォールトコードに関連する簡単な説明を返します
- `getFaultDetail()`: フォールトの原因に関する情報を返します。クラス名、メッセージ、スタックトレースなど、特定の Java フォールトをキャッチできます。以下の例 4 で説明しているように、キャッチ内でカスタムフォールト名を使用してこの機能を実装します。Java フォールトに対する修飾されたカスタムフォールト名に必要な名前空間に注意してください。
- `getSOAPFault()`: `soap:Fault` 要素全体を返します (SOAP 呼び出しによって発生したフォールトである場合)。

宣言されていないフォールトのフォールト処理を作成する方法の例を次に示します。

例 1: 宣言されていないフォールトと SOAP フォールトの *CatchAll* ハンドラを追加します

1. [「Catch および CatchAll フォールトハンドラの定義」 \(ページ 384\)](#)に記載されているように、CatchAll をスコープまたはプロセスに追加します (または、ハンドラを BPMN 境界イベントとして追加します)。
2. ハンドラのターゲットアクティビティとして Assign アクティビティを追加します。
3. 次の例に示すように、フォールトカスタム関数を使用して式をプロセス変数にコピーします。



4. 作成するコピー操作は、次の XQuery の例のようになります。

コピー元:

```
<ns1:myFaultInfo xmlns:ns1="urn:faults:undeclared" xmlns:ns2="urn:faults:undeclared:result">
  <ns2:result>
    <ns2:FaultCode>{ abx:getFaultCode() }</ns2:FaultCode>
    <ns2:FaultString>{ abx:getFaultString() }</ns2:FaultString>
    <ns2:FaultDetail>{ abx:getFaultDetail() }</ns2:FaultDetail>
    <ns2:SOAPFault>{ abx:getSOAPFault() }</ns2:SOAPFault>
  </ns2:result>
</ns1:myFaultInfo>
```

コピー先:

myFaultInfo

例 2: キャッチに追加するプロセス変数を宣言します

- スキーマ要素をプロジェクト WSDL に追加します。これは、フォールト変数のキャッチまたはスローの定義で使用されます。以下に例を示します。

```
<xs:element name="undeclaredFaultElement" type="tns:undeclaredFaultType"/>
<xs:complexType name="undeclaredFaultType">
  <xs:sequence>
    <xs:element name="faultMessage" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

2. キャッチを追加し、次のようにスキーマ要素をフォールト変数の定義として使用します。

```
Element = ns1:undeclaredFaultElement
```

3. `myFault` などの Catch フォールト変数名を追加します
4. キャッチに Assign アクティビティを追加し、例 1 のように手順を完了します。

例 3: `Throw` アクティビティに追加するプロセス変数を宣言します

1. 例 2 のように、スキーマ要素をプロジェクトの WSDL に追加します。
2. スキーマ要素として定義された `myFault` などの新しいプロセス変数を追加します。
3. 次の例のように、初期値をリテラルとして変数に追加します。

```
<ns1:undeclaredFaultElement xmlns:ns1="urn:faults:undeclared">
  <faultMessage>string</faultMessage>
</ns1:undeclaredFaultElement>
```

4. プロセスにスローを追加します。
5. **【フォールト変数】** として `[myFault]` を選択します。
6. `ns1:myFault` などのような、スローのカスタムフォールト名を作成します。

例 4: Java フォールトを名前でキャッチします

1. 次のように、必要な **【名前空間】** をプロセスに追加します。
`aex="http://www.active-endpoints.com/2004/06/bpel/extensions/"`
これは、システムエラーをキャッチするために使用される Process Developer の名前空間です。
2. プロセスにキャッチを追加します。
3. 次のようにカスタム **【フォールト名】** を定義します。
 - 手順 1 に示した、必要な名前空間を指す `aex` のプレフィックスまたはその他のプレフィックス。
 - Java 例外の非修飾クラス名。この名前はクラスの NCName であり、パッケージの詳細は含まれません。例: `UnknownHostException`。
例: `aex:UnknownHostException`。
4. 割り当てを追加してキャッチを完了します。例 1 を参照してください。
5. 例えば、無効な URL が原因で `java.net.UnknownHostException` が呼び出しによってスローされた場合、フォールトの詳細要素は次のようになります。

```
<ext:UnknownHostException xmlns:ext="http://www.active-endpoints.com/2004/06/bpel/extensions/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <Class xmlns="http://www.active-endpoints.com/2004/06/bpel/
extensions/">java.net.UnknownHostException</Class>
  <Message xmlns="http://www.active-endpoints.com/2004/06/bpel/extensions/">unknown-host</
Message>
  <StackTrace xmlns="http://www.active-endpoints.com/2004/06/bpel/
extensions/">java.net.UnknownHostException: unknown-host
at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:177)
at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:366)
at java.net.Socket.connect(Socket.java:525)
at java.net.Socket.connect(Socket.java:475)
at java.net.Socket.<init>(Socket.java:372)
...
at org.activebpel.work.AeDelegatingWork.run(AeDelegatingWork.java:62)
at org.activebpel.work.AeExceptionReportingWork.run(AeExceptionReportingWork.java:58)
at org.activebpel.work.AeWorkerThread.run(AeWorkerThread.java:142)
</StackTrace>
</ext:UnknownHostException>
```

第 26 章

プロセス例外管理

プロセス例外管理は、プロセスサーバーで使用可能な機能です。この機能を使用すると、キャッチされないフォールトが発生した場合にプロセスを一時停止したり、こうしたフォールトの発生時にアラートを発するようになりことができます。Process Developer では、この機能を複数の方法で使用することができます。

次のトピックでは、プロセス例外管理について説明します。

- [「プロセス例外管理について」 \(ページ 398\)](#)
- [「キャッチされないフォールト時のプロセスの一時停止」 \(ページ 399\)](#)
- [「警告サービス」 \(ページ 304\)](#)

プロセス例外管理について

プロセス例外管理には、実行中のプロセスで発生する例外の管理が含まれます。例外は一般的に、通常のフォールトハンドラによってキャッチされないフォールトとして現れます。つまり、それらは想定されたものではありません。これらのフォールトは、不良データや外部システムの問題など、プロセス実行における予期しないエラーの結果です。

プロセス例外管理により、プロセスを異常終了させるのではなく、フォールトが発生したアクティビティで一時停止することができます。キャッチされなかったフォールトの自動一時停止に加えて、BPEL でプログラム的に例外ケースをチェックし、Suspend アクティビティを使用してプロセスを一時停止することができます。いずれの場合も、問題のアラートを受け取ったユーザーは、その場でエラーを修正してプロセスを再開できます。

プロセスコンソールでプロセス例外管理を実行できます。詳細については、[「キャッチされないフォールト時のプロセスの一時停止」 \(ページ 399\)](#)を参照してください。

詳細については、次を参照してください:

- [「キャッチされないフォールト時のプロセスの一時停止」 \(ページ 399\)](#)
-
- [「警告サービス」 \(ページ 304\)](#)

キャッチされないフォールト時のプロセスの一時停止

キャッチされないフォールト時のプロセスの一時停止は、プロセスサーバーで実行中のプロセスに適用されません。

次の方法で、プロセスサーバーですべてのフォールトプロセス、または個別に選択されたフォールトプロセスを一時停止します。

- [「キャッチされないフォールト時のすべてのプロセスの一時停止の対象化」](#) (ページ 399)
- [「キャッチされないフォールト時の個々のプロセスの一時停止の対象化」](#) (ページ 399)
- [「Suspend アクティビティによる、プログラムでのプロセスの一時停止」](#) (ページ 399)

「キャッチされないフォールト時に中断」設定には、[全体] または [永続] というプロセスの永続性が必要です。永続性に関する情報については、[「全般的なデプロイメントオプション」](#) (ページ 457) のトピック内の「プロセスの永続性」セクションを参照してください。

キャッチされないフォールト時のすべてのプロセスの一時停止の対象化

キャッチされないフォールト時のプロセスの一時停止は、プロセスサーバーで実行中のプロセスに適用されません。

WS-BPEL 2.0 仕様に従って、キャッチされないフォールトのあるプロセスは、終了ハンドラによって終了します。ただし、プロセスサーバーでは、フォールトに対応するキャッチがない（つまり、キャッチされていない）場合、アクティビティでのフォールトの発生時にプロセスの一時停止が許可されます。ユーザーがアクティビティを再試行または完了することができるように、アクティビティの実行を完了する前に、サーバーはプロセスを停止します。

この設定は、プロセスコンソールで有効化できます。詳細については、「[プロセスコンソールのヘルプ](#)」を参照してください。

必要に応じて、個々のプロセスのサーバー設定を上書きできます。詳細については、[「キャッチされないフォールト時の個々のプロセスの一時停止の対象化」](#) (ページ 399) を参照してください。

キャッチされないフォールト時の個々のプロセスの一時停止の対象化

キャッチされないフォールト時のプロセスの一時停止は、プロセスサーバーで実行中のプロセスに適用されません。

サーバーには、キャッチされないフォールト時にすべてのプロセスを一時停止の対象とする構成設定があります。ただし、個々のプロセスのプロセスデプロイメント記述子 (PDD) ファイルのサーバー設定を上書きすることができます。詳細については、[「全般的なデプロイメントオプション」](#) (ページ 457) を参照してください。

[「キャッチされないフォールト時のすべてのプロセスの一時停止の対象化」](#) (ページ 399) も参照してください。

Suspend アクティビティによる、プログラムでのプロセスの一時停止

詳細については、「[中断](#)」を参照してください。

第 27 章

補償

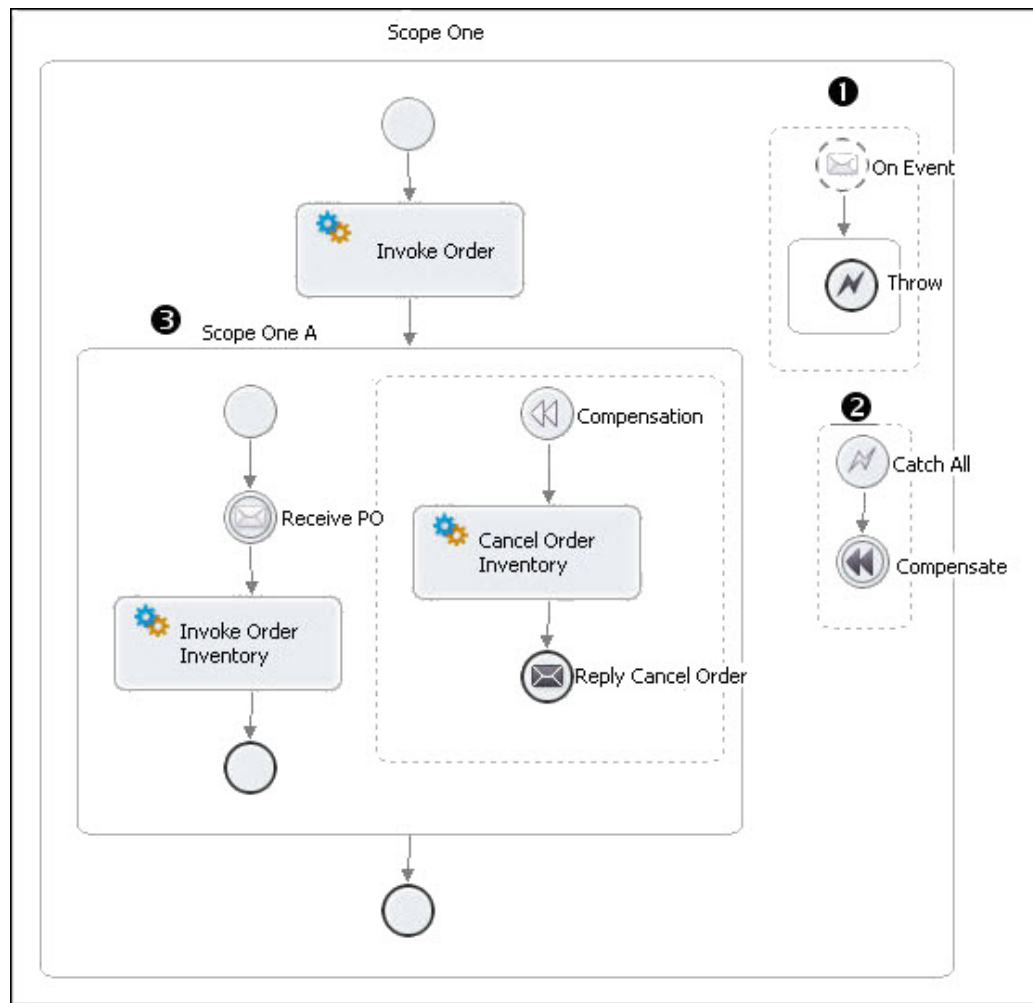
補償に関するトピックは次のとおりです。

- [「補償について」 \(ページ 400\)](#)
- [「補償ハンドラと Compensate アクティビティ」 \(ページ 402\)](#)
- [「スコープへの補償ハンドラの追加」 \(ページ 403\)](#)
- [「呼び出しアクティビティの補償」 \(ページ 405\)](#)

補償について

補償を使用することで、ビジネスプロセスにおけるスコープまたはプロセスレベルでのアクティビティの定義が可能になります。このアクティビティの実行は、以前に実行されたアプリケーションロジックを元に戻す場合に役立ちます。補償中にデータが自動的に復元されることはありません。独自の補償動作の定義については、アプリケーションに依存します。補償の例としては、予約のキャンセル、注文の保留、クレジットカードへの請求の取り消しなどが挙げられます。

ビジネスプロセスの一部を取り消すような場合でも、ビジネスプロセスが継続して実行されるようにすることができます。次の図は、ビジネスプロセスにおいて在庫注文をキャンセルし、注文キャンセルの通知を送信するイベントハンドラの onEvent 部分の例を示しています。補償の対象となるアクティビティは、すでに完了したスコープのアクティビティです。



1	イベントハンドラが注文キャンセルメッセージを受信すると、フォールトハンドラにフォールトをスローします
2	以前に完了したリンク先のスコープに対して、フォールトハンドラが補償アクティビティを実行します
3	完了したスコープに対する補償ハンドラが、InvokeOrderInventory サービスの作業をロールバックします

補償処理はスコープに関連付けられています。スコープは、プロセスの他の部分を正常に続行する一方で、取り消しや再試行が必要となる可能性のある一連のアクティビティを含んだ、論理的な作業単位と考えることができます。補償アクションは、元のアクションと同じ順序で実行される必要がある場合もあります。

補償と変数の状態

補償ハンドラが呼び出されると、補償の対象となるスコープが完了した場合と同じように、ローカルで定義したスコープ変数がフリーズ状態で表示されます。スコープがループ内で複数回実行された場合、各補償インスタンスには、スコープの完了時にローカル変数の値が記録されます。

補償ハンドラと Compensate アクティビティ

補償ハンドラと Compensate アクティビティ

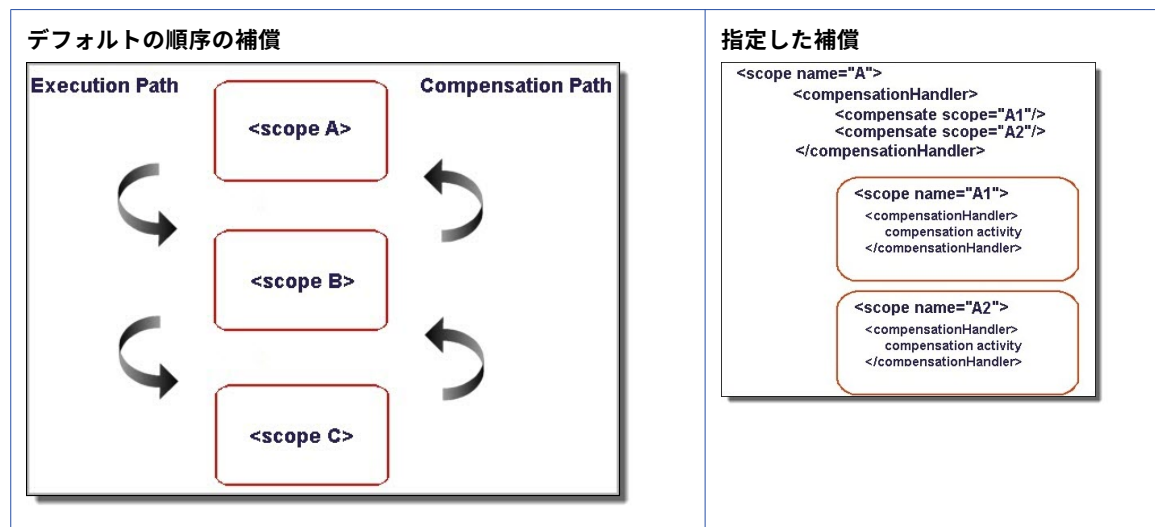
補償処理は、次のような手順で追加します。

- スコープに補償ハンドラを定義します。補償ハンドラは単一の子アクティビティのみを持つことができますが、このアクティビティとして、子アクティビティを持つシーケンスなどのような構造化されたアクティビティを設定することもできます。補償ハンドラの内容は、アプリケーションロジックに応じて異なります。
- すでに正常に完了したすべての内部スコープの補償をデフォルトの順序で指定するには、フォールトハンドラ、補償ハンドラ、または終了ハンドラ内で Compensate アクティビティを使用します。
- 名前付きの内部スコープの補償を指定するには、フォールトハンドラ、補償ハンドラ、または終了ハンドラ内で Compensate Scope アクティビティを使用します。この方法によって、スコープ内の Compensate アクティビティの順序と選択を制御できます。

XML 構文

```
<compensationHandler>  
  activity  
</compensationHandler>
```

次の図に、デフォルトの補償と指定した補償の比較を示します。



暗黙的な補償

スコープで補償ハンドラが定義されていない場合、実行エンジンによって暗黙的な補償ハンドラが提供されます。このハンドラには、単一の<compensate>アクティビティが含まれており、完了して補償の対象となった、エンクローズされたすべてのスコープが補償されます。通常は、以前に実行されたロジックを元に戻すために実行するアプリケーション固有のロジックがある場合にのみ、補償ハンドラを指定する必要があります。

関連事項:

- [「デフォルトの順序の補償の例」 \(ページ 402\)](#)
- [「指定した補償の例」 \(ページ 403\)](#)

デフォルトの順序の補償の例

デフォルトの補償処理は、<compensate/>という形式の compensate アクティビティから利用できます。このアクティビティは、補償の対象となるエンクローズされたすべてのスコープで補償ハンドラを呼び出します。ス

コープの補償ハンドラは、完了までとは逆の順序で呼び出されます。スコープが<while>ループで複数回実行された場合、このハンドラは、フォールトが発生することなく完了した各実行インスタンスに対して適格となります。

次の例は、確認済みの注文書がキャンセルされた場合を示しています。CancelPurchase 操作によって、すでに完了した Invoke アクティビティの SyncPurchase 操作が補償されます。

```
<scope>
  <compensationHandler>
    <invoke partnerLink="Seller" portType="SP:Purchasing"
      operation="CancelPurchase"
      inputVariable="getResponse"
      outputVariable="getConfirmation">
      <correlations>
        <correlation set="PurchaseOrder" pattern="request"/>
      </correlations>
    </invoke>
  </compensationHandler>
  <invoke partnerLink="Seller" portType="Sell:Purchasing"
    operation="SyncPurchase"
    inputVariable="sendPO"
    outputVariable="getResponse">
    <correlations>
      <correlation set="PurchaseOrder" initiate="yes"
        pattern="request"/>
    </correlations>
  </invoke>
</scope>
```

[「指定した補償の例」 \(ページ 403\)](#)も参照してください。

指定した補償の例

```
<faultHandlers>
  <catch faultName="lms:loanProcessFault"
    faultVariable="error">
    <sequence name="fault-sequence">
      <compensate scope="assessor-scope"/>
    </sequence>
  </catch>
</faultHandlers>
```

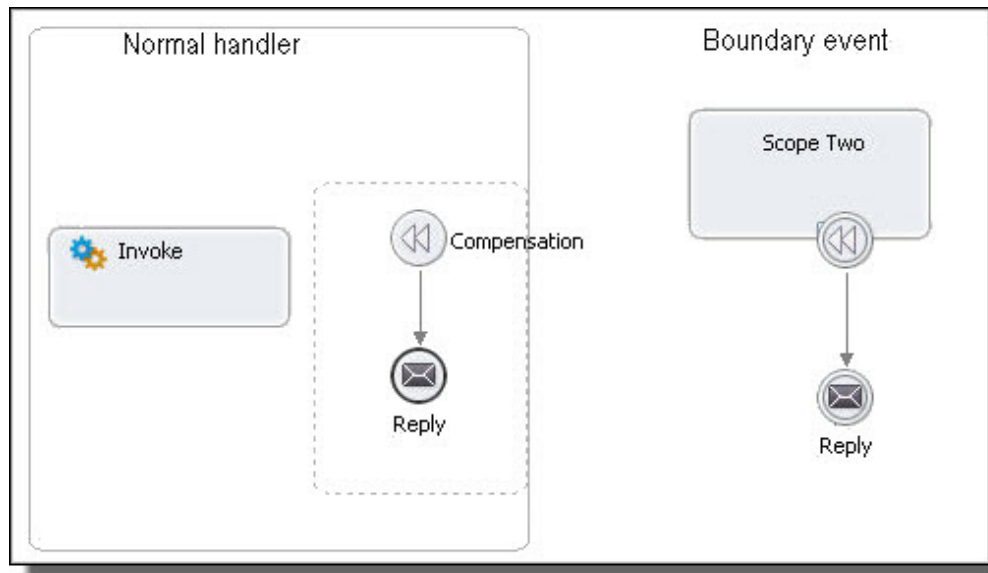
[「デフォルトの順序の補償の例」 \(ページ 402\)](#)も参照してください。

スコープへの補償ハンドラの追加

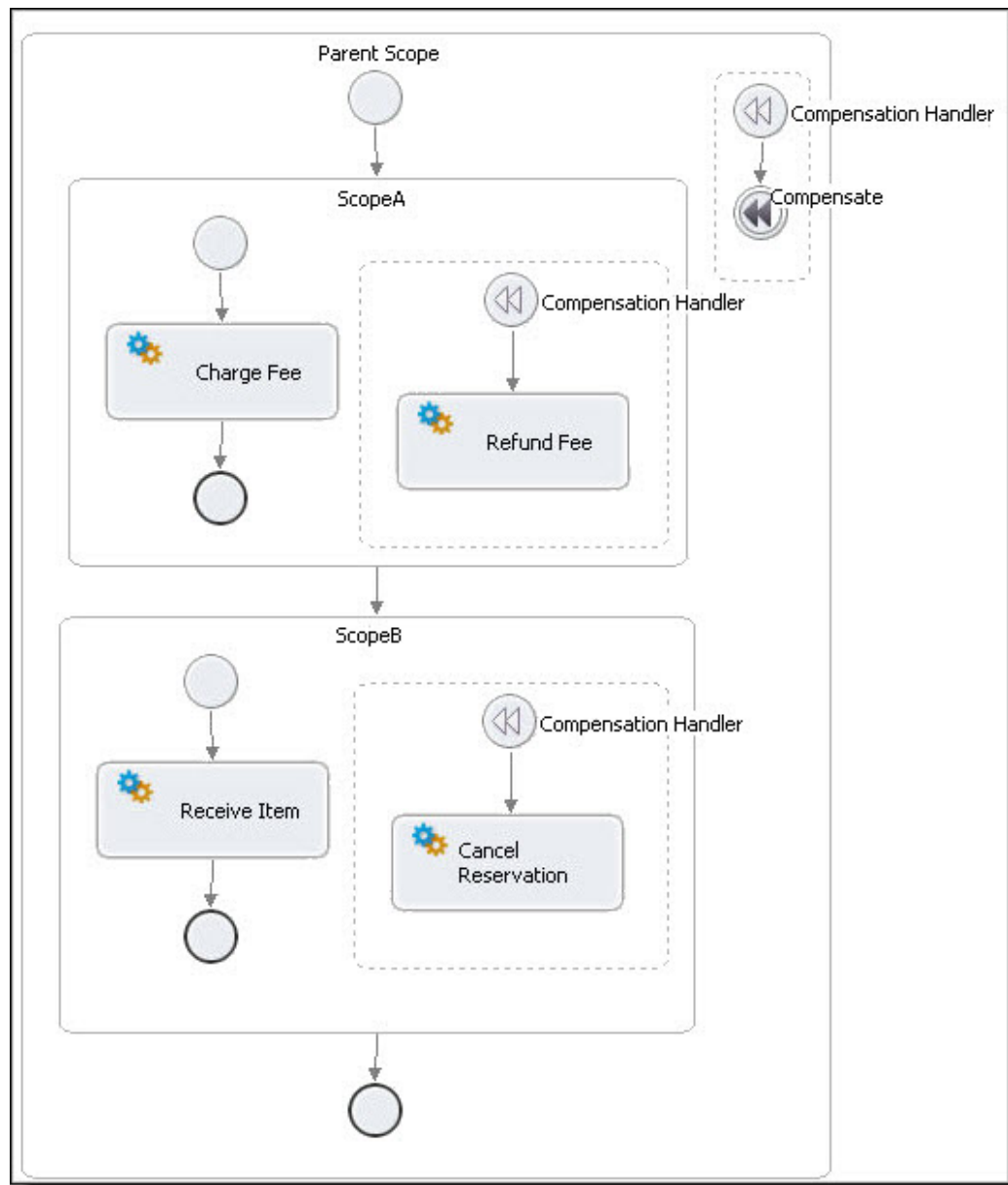
スコープのアクティビティは、スコープが完了した場合、および別のアクティビティによって補償が開始された場合に、補償するか元に戻すことができます。

スコープに補償ハンドラを追加する手順

1. Process Editor キャンバスから、スコープを選択します。
2. 図に示すように、スコープ内または折りたたまれたスコープの境界近くに Compensation catch イベントをドラッグします。



3. スコープのメインアクティビティの作業を元に戻す補償ハンドラにアクティビティをドラッグします。境界イベントの場合、実行するアクティビティにハンドラをリンクする必要があります。このアクティビティは、まだ実行されていないダウンストリームアクティビティである必要があります。
 4. 別のスコープから、完了したスコープの補償ハンドラをトリガするアクティビティを作成します。
- 次の図は、親スコープ、子スコープ、およびこれらのスコープの補償ハンドラを示しています。



呼び出しアクティビティの補償

WS-BPEL 2.0 仕様には、オプションの補償ハンドラおよびフォールトハンドラを呼び出しに追加するための簡易的な構成が用意されています。これらのハンドラは、[「スコープへの補償ハンドラの追加」](#) (ページ 403)に記載されている境界イベントとして追加できます。

パート IV: テストとデプロイメント

このセクションのトピックでは、プロセスをシミュレート、テスト、およびデプロイする方法を説明します。

Process Developer のオーケストレーションファイルリソース

Orchestration プロジェクトを開発すると、BPEL、WSDL、XSD、HTML、XML などのさまざまな種類のファイルが蓄積されます。これらに加えて、Process Developer は、デフォルトで非表示になっている複数のプロジェクトファイルを生成します。以下は、Process Developer によってプロジェクトに追加されるリソースのリストです。これらのファイルの説明に基づいて、それらのリソースをソース管理リポジトリに追加するかどうかを決定できます。

Process Developer のリソース	説明
.image	プロセスを保存すると、プロセスのスクリーンショットが .image という名前のフォルダに複数のイメージとして自動的に保存されます。デプロイを行うと、プロセスコンソールではこれらのイメージを使用して、プロセスの [詳細グラフ] ビューが表示されます。この情報が無い場合、デフォルトのプロセスグラフが表示されますが、これはプロセスの外観と一致しない可能性があります。 .image フォルダとその中のすべてのファイルを保持することをお勧めします。
.archive_settings.properties	BPR エクスポートウィザードでは、Web サービスの URL やその他の設定を選択できます。これらの設定は .archive_settings.properties に保存されます。プロジェクト内の BPR エクスポートウィザードのその後の工程で、Process Developer はこのファイルの情報を使用します。

Process Developer のリソース	説明
merge_mappings.xml	(ヒューマンタスク) ユーザーアクティビティのタスクフォームを作成する際に、タスク入力データを出力データフィールドにマッピングして、マージ可能な入力フィールドと出力フィールドのフォームにフィールドの統合リストを表示できます。入出力マッピングは、この XML ファイルに保存されます。通常、マッピングは 1 度きりの使用を通して作成されます。ただし、このファイルをプロジェクトとともに保存することをお勧めします。
.BPR、.BPRD、.project、.classpath	ビジネスプロセスアーカイブファイル (BPR) は、エクスポートウィザードを使用して再生成できます。ただし、このファイルを保存している場合は、自己完結型であるため、チームメンバー間で簡単に共有できます。一方で、生成する Ant ファイル (.BPRD ファイル) には、他のチームメンバーによる変更が必要となるローカルホストとポートを指す Web の URL を含めることができます。 .project および .classpath (使用する場合) を含む作成された Eclipse ファイルは、プロジェクトとともに保存する必要があります。

BPEL 標準フォールト

次のテーブルに、WS-BPEL 仕様内で定義されている標準のフォールトを示します。これらのフォールトはすべて、WS-BPEL 名前空間内で次の URI に対応する標準のプレフィックス `bpel:` が付いた名前となります。

<http://docs.oasis-open.org/wsbpel/2.0/process/executable>

フォールト名	説明
ambiguousReceive	ビジネスプロセスインスタンスによって、関連セットが異なる同一の partnerLink、portType、または操作に対して 2 つ以上の IMA が同時に有効化された場合にスローされ、これらのアクティビティの複数の相関が受信要求メッセージと一致します
completionConditionFailure	<forEach>アクティビティ内に直接エンクローズされた<scope>アクティビティが完了したときに、完了条件が true にならないと判断される場合にスローされます
conflictingReceive	同じパートナーリンク、ポートタイプ、操作、および関連セットに対して複数のインバウンドメッセージアクティビティが同時に有効になった場合にスローされます。
conflictingRequest	同じパートナーリンク、操作、およびメッセージ交換に対して複数のインバウンドメッセージアクティビティが開いている場合にスローされます
correlationViolation	<invoke>、<receive>、<reply>、<onMessage>、または<onEvent>で処理されるメッセージの内容が、指定した相関情報と一致しない場合にスローされます。
invalidBranchCondition	<forEach>の<branches>完了条件で使用する整数値が、直接エンクローズされた<scope>アクティビティの数よりも大きい場合にスローされます。

フォールト名	説明
invalidExpressionValue	WS-BPEL 構成 (<assign>を除く) 内で使用される式が、想定される XML スキーマタイプに関連する無効な値を返した場合にスローされます。
invalidVariables	変数値の XML スキーマ検証 (暗黙的または明示的) が失敗した場合にスローされます
joinFailure	アクティビティの結合条件が false と評価され、suppressJoinFailure 属性の値が 【はい】 である場合にスローされます
mismatchedAssignmentFailure	<assign>アクティビティで、互換性のないタイプまたは互換性のない XML 情報セット構造が検出された場合にスローされます。
missingReply	インバウンドメッセージアクティビティが実行され、対応する<reply>アクティビティが実行されずに、プロセスインスタンスまたはスコープインスタンスが実行の最後まで到達した場合にスローされます。
missingRequest	パートナーリンク、操作、およびメッセージ交換タプルを照合した際に、開いている受信メッセージアクティビティに<reply>アクティビティを関連付けることができない場合にスローされます。
scopeInitializationFailure	スコープの初期化の一部として定義されたオブジェクトの作成に問題がある場合にスローされます。このフォールトは、フォールトが発生したスコープの親スコープによって常にキャッチされます。
selectionFailure	bpel:getVariableProperty などの関数または割り当てで実行された選択操作でエラーが発生した場合にスローされます。
subLanguageExecutionFault	式の実行により、式言語またはクエリ言語で未処理のフォールトが発生した場合にスローされます。
uninitializedPartnerRole	<invoke>または<assign>アクティビティが、partnerRole エンドポイント参照の初期化が実行されていないパートナーリンクを参照している場合にスローされます。
uninitializedVariable	初期化されていない変数の値、またはメッセージタイプ変数の場合は初期化されていないいずれかのパートにアクセスしようとした場合にスローされます。
unsupportedReference	WS-BPEL 実装が、reference-scheme 属性と content 要素の組み合わせの解釈、または content 要素の解釈に失敗した場合にスローされます。
xsltInvalidSource	bpel:doXsltTransform 関数呼び出しで指定されたトランスフォーメーションソースが不正であった場合にスローされます。
xsltStylesheetNotFound	bpel:doXsltTransform 関数呼び出しで、指定のスタイルシートが見つからなかった場合にスローされます。

第 28 章

シミュレーションとデバッグ

シミュレーションとデバッグに関するトピックは次のとおりです。

- [「Process Developer の \[デバッグ\] パースペクティブについて」 \(ページ 409\)](#)
- [「Process Developer の \[デバッグ\] パースペクティブビューとメニュー」 \(ページ 410\)](#)
- [「BPEL プロセスの実行のシミュレーション」 \(ページ 416\)](#)
- [「シミュレーション中のサンプル変数データの入力および検査」 \(ページ 421\)](#)
- [「シミュレーションパスとプロパティの選択」 \(ページ 424\)](#)
- [「シミュレーション中の標準フォールトの検査」 \(ページ 427\)](#)
- [「シミュレーションの設定」 \(ページ 427\)](#)
- [「デバッグの設定」 \(ページ 431\)](#)

Process Developer の [デバッグ] パースペクティブについて

Process Developer の [デバッグ] パースペクティブには、Process Developer に作成またはインポートする BPEL プロセスの実行をサポートするビュー、エディタ、メニュー、およびツールバーが含まれています。

シミュレーション中に、ブレークポイントまたはステップを使用してプロセスをデバッグできます。シミュレート/実行モードではプロセスを実行できますが、実行の一時停止や検査はできません。シミュレーション/ステップモードでは実行を一時停止および再開したり、変数を検査および変更したりすることができます。

パースペクティブの使用に関する概念とヒントについては、[「ウィンドウ、パースペクティブ、ビュー、およびエディタ」 \(ページ 65\)](#)を参照してください。

[デバッグ] パースペクティブの詳細については、以下を参照してください。

- [「Process Developer の \[デバッグ\] パースペクティブを開く」 \(ページ 410\)](#)
- [「Process Developer のパースペクティブの切り替え」 \(ページ 410\)](#)

Process Developer の [デバッグ] パースペクティブを開く

シミュレーションまたはリモートデバッグを開始すると、Process Developer は自動的に [デバッグ] パースペクティブに切り替わります。ただし、次のいずれかの方法でこのパースペクティブを手動で表示することもできます。

- プロジェクトエクスプローラの右上にある [パースペクティブ] バーから、[Process Developer デバッグ パースペクティブ] アイコンを選択します。
- [ウィンドウ] メニューから、[パースペクティブを開く] > [その他] > [Process Developer のデバッグ] を選択します。

ヒント: 同じウィンドウまたは新しいウィンドウでパースペクティブを開くことができます。[ウィンドウ] > [新規ウィンドウ] メニューから、新しいウィンドウでパースペクティブを開きます。常に新しいウィンドウで新しいパースペクティブを開くためのワークベンチ設定もあります。

Process Developer のパースペクティブの切り替え

Process Developer には、[設計] と [デバッグ] という 2 つのパースペクティブがあります。各パースペクティブには、一連の関連タスクをサポートするビューとメニューのセットがあります。

次のいずれかの方法で、パースペクティブを切り替えます。

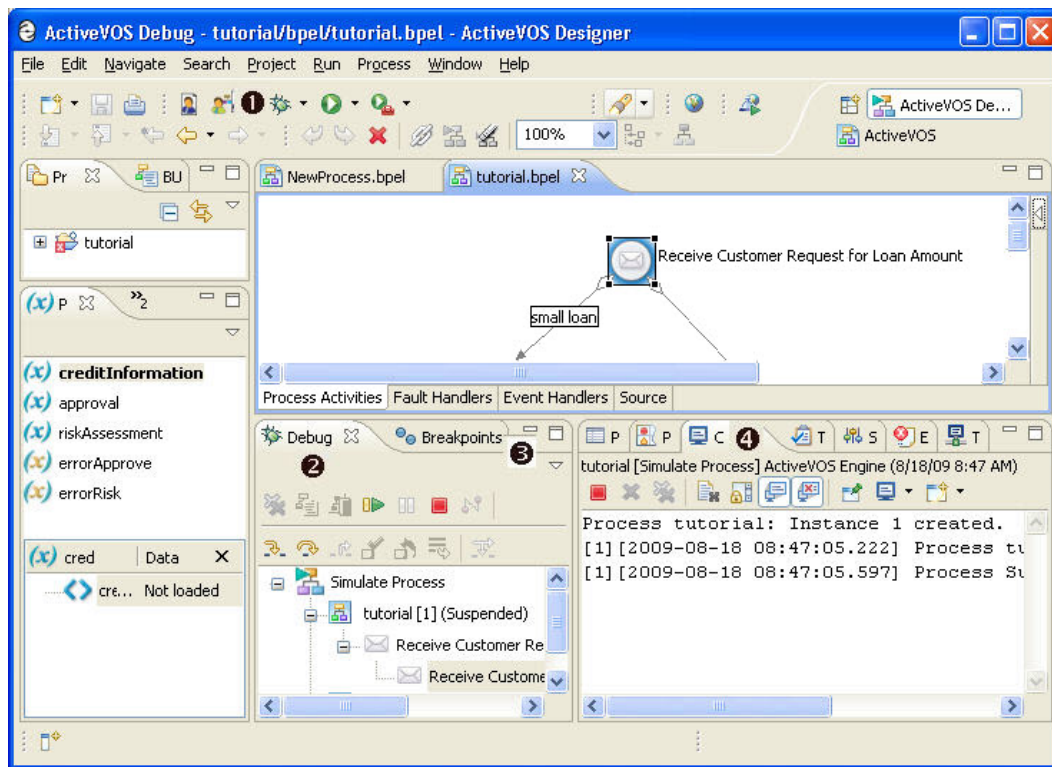
- [パースペクティブ] バーのアイコンをクリックする。
- [ウィンドウ] > [パースペクティブを開く] を選択し、パースペクティブを選択する。
- 両方のパースペクティブが開いている場合は、次のパースペクティブ (Ctrl + F8) と前のパースペクティブ (Ctrl + Shift + F8) のキーボードショートカットを使用します。

パースペクティブの表示に関するヒント:

- ウィンドウのタイトルバーには、現在アクティブなパースペクティブの名前が表示され、開いているパースペクティブを管理する場合に役立ちます。
- ビューを閉じたり再配置したりした場合は、[ウィンドウ] > [パースペクティブのリセット] を選択して、パースペクティブをデフォルトの表示にリセットできます。
- [デバッグ] パースペクティブの色とフォントの設定を選択します。[「デバッグの設定」 \(ページ 431\)](#) を参照してください。

Process Developer の [デバッグ] パースペクティブビューとメニュー

次の図は、デバッグパースペクティブを構成するビューとメニューを示しています。



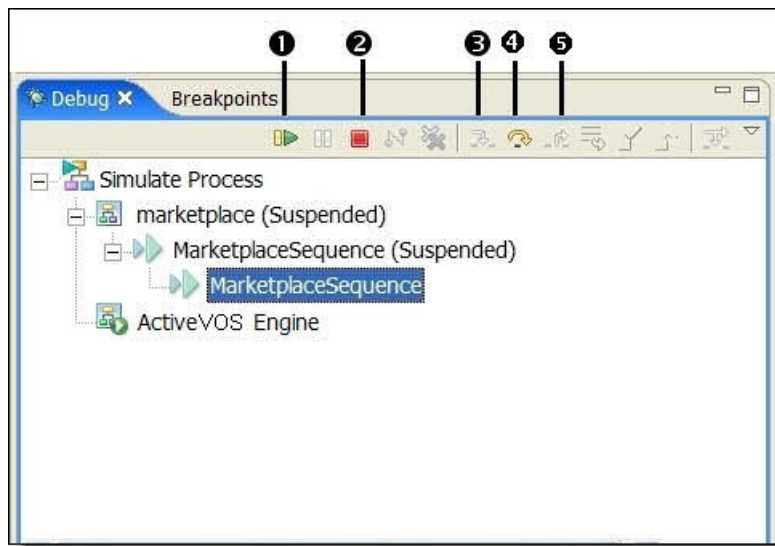
1	[プロセスのシミュレート] と [実行状態のクリア] ツールバーアイコン。『 BPEL プロセスのシミュレーションの開始と終了 』（ページ 417）および『 BPEL プロセスのシミュレーションの開始と終了 』（ページ 417）を参照してください。
2	[デバッグ] ビュー。『 Process Developer の [デバッグ] ビューの使用 』（ページ 411）を参照してください。
3	[ブレークポイント] ビュー。『 BPEL プロセスシミュレーションでのブレークポイントの使用 』（ページ 413）を参照してください。
4	コンソール。『 Process Developer デバッグコンソールの使用 』（ページ 415）を参照してください。

Process Developer の [デバッグ] ビューの使用

Process Developer の [デバッグ] パースペクティブには、Process Developer で作成した BPEL プロセス、または Process Developer にインポートした BPEL プロセスの実行あるいはステップスルーを管理するための [デバッグ] ビューが含まれています。[デバッグ] ビューには、デバッグ中のプロセスに関連付けられた実行ツリーが表示されます。

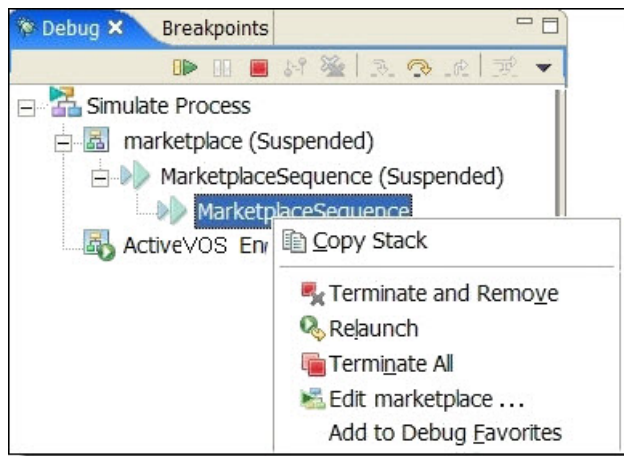
Process Developer の [デバッグ] ビューは Eclipse [デバッグ] ビューに基づいていますが、Process Developer ですべての機能が使用できるわけではありません。アイコン、オプション、および設定といった一部の機能は、Process Developer では使用できません。

次の図は、サンプルの [デバッグ] ビューを示しています。この表は、使用可能なファンクションキーを示しています。



1	終了 (F8 キー)	一時停止されたスレッドを再開します。次のブレークポイントが検出されるまで、またはプロセスが完了するまで、プロセスを実行します。
2	終了	選択したデバッグターゲットを終了します
3	ステップイン (F5 キー)	プロセス内の次のアクティビティに到達するまでプロセスを実行します。アクティビティが Scope、If、While、For Each などのコンテナである場合、コンテナ内の各アクティビティが実行されます。 ヒント: サブプロセスである Invoke アクティビティにステップインすることもできます。詳細については、 「シミュレーション用の呼び出しサブプロセスの選択」 (ページ 426) を参照してください。
4	ステップオーバー (F6 キー)	プロセス内の次のアクティビティに到達するまでプロセスを実行します。ブレークポイントが検出されると、実行はブレークポイントで一時停止されます。
5	ステップリターン (F7 キー)	ステップインされたコンテナから戻るには、このキーを使用します。ステップがコンテナから戻った後も、コンテナ内の残りのアクティビティは引き続き実行されます。

ツールバーオプションに加えて、シミュレーションスレッドの項目の右マウスメニューにはさらに多くのオプションがあります。次の図は、マウスの右メニューで使用可能なすべてのオプションではなく、追加のオプションのみを示しています。

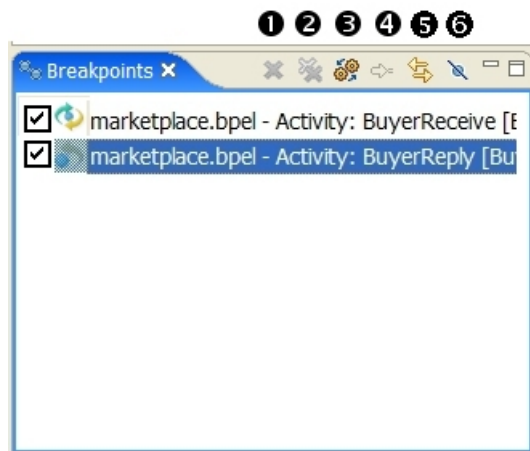


スタックのコピー	選択済みの一時停止されたスレッドのスタックと、実行中のスレッドの状態をクリップボードにコピーします
終了して削除	選択したデバッグターゲットを終了し、ビューから削除します
再起動	新たなシミュレーションを開始します
すべて終了	ビュー内のすべてのアクティブなシミュレーションを終了します

BPEL プロセスシミュレーションでのブレークポイントの使用

〔ブレークポイント〕ビューには、BPEL プロセスで設定したすべてのブレークポイントが一覧表示されます。ブレークポイントをダブルクリックすると、Process Editor にその場所が表示されます。〔ブレークポイント〕ビューでは、ブレークポイントを有効化、無効化、スキップ、または削除することもできます。

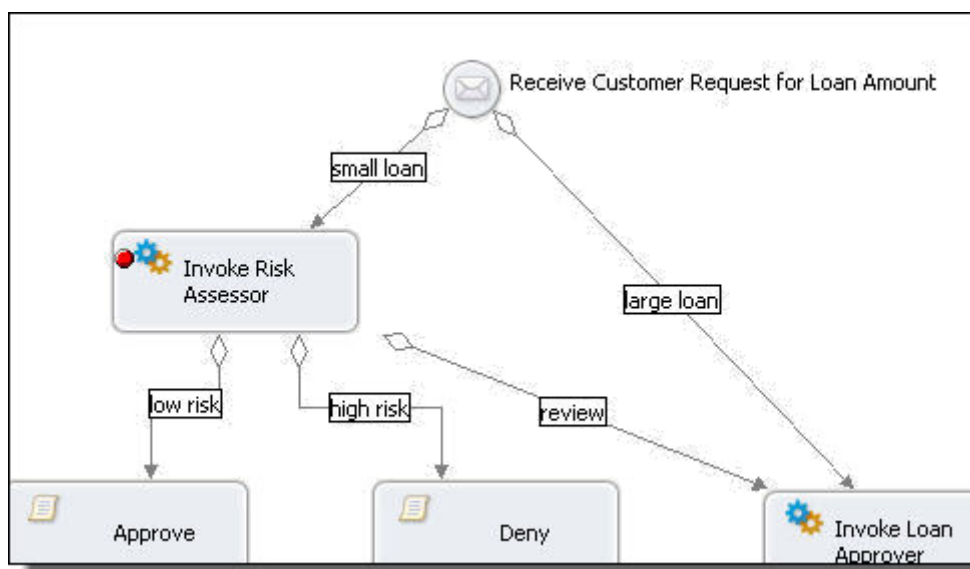
次の図は、サンプルの〔ブレークポイント〕ビューとプロセスを示しています。アクティビティの横にブレークポイントが追加され、ブレークポイントのリストが〔ブレークポイント〕ビューに表示されています。



1	選択したブレークポイントを削除します
2	すべてのターゲットのすべてのブレークポイントを削除します
3	選択したターゲットでサポートされるブレークポイントを表示します
4	ブレークポイントのファイルに移動します
5	ビューを関連付けて、デバッグ中の現在のファイルを常に表示します
6	すべてのブレークポイントをスキップします

ブレークポイントの追加

1. プロジェクトエクスプローラから、BPEL ファイルを開きます。
2. ブレークポイントを設定するアクティビティを右クリックし、**[ブレークポイントの追加]** を選択します。
図のように、ブレークポイントを示す赤い円がアクティビティの横に表示されます。



[ブレークポイント] ビューで作業するためのヒント

- 表示されているすべてのブレークポイントを削除するには、ツールバーから **[すべてのブレークポイントを削除]** を選択します。
- Process Editor でブレークポイントの場所を強調表示するには、ブレークポイントをダブルクリックします。
- ブレークポイントが設定されている Process Editor のアクティビティに移動するには、ブレークポイントを右クリックし、**[ファイルへ移動]** を選択します。ファイルが開いていない場合は、このオプションを選択するとファイルが開き、ブレークポイントの場所が示されます。
- それぞれのブレークポイントには、BPEL ファイルの名前とブレークポイントが設定されたアクティビティが表示されます。デバッグ中のすべてのファイルのすべてのブレークポイントが一覧表示されます。
- ブレークポイントの横にあるボックスのチェックマークを選択/選択解除することで、ブレークポイントを **[有効化]** または **[無効化]** することができます。無効化された状態のブレークポイントは、アクティビティアイコンの横に灰色の円として表示されます。

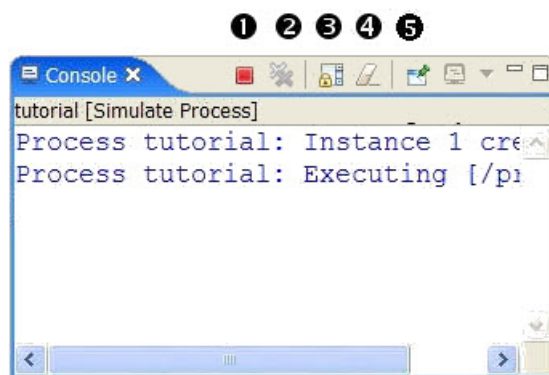
[「BPEL プロセスでのブレークポイントまでの実行」 \(ページ 418\)](#) も参照してください。

Process Developer デバッグコンソールの使用

[コンソール] ビューは、Process Developer の [デバッグ] パースペクティブの一部です。

このビューには実行イベントが表示され、式と結果の評価に関するビューが提供されます。

次の図は、コンソールのサンプルを示しています。



1	終了。現在のシミュレーションを停止します
2	終了したすべての起動を削除します
3	スクロールロック。水平スクロールが発生しないように、長い行を自動的に折り返します。
4	コンソールのクリア。
5	コンソールの固定。該当なし。

プロセスが実行されると、次のタイプのイベントが表示されます。

アクティビティタイプ、名前、パス	実行中、正常に完了、またはフォールトが発生して完了 例: Receive:Executing </process/flow/receive>
リンク名、リンク遷移条件、パス	式の状態と評価 例: Link receive-to-assess Condition true : bpws:getVariableData('request','amount') < 10000 </process/flow/links/link[@name='receive-to-assess']>
On Alarm	期間または期限
Wait	待機値とシミュレートされた待機値
結合条件	式の評価
While、If、Repeat Until 条件	式の評価
未処理のイベント	補助的なイベント情報

コンソールの使用に関するヒント:

- すでに実行されているアクティビティに移動するには、右マウスメニューを使用します。
- コンソールでシミュレーションイベントを見つけるには、右マウスメニューを使用します。

BPEL プロセスの実行のシミュレーション

設計後、BPEL プロセスをサーバーにデプロイする前に、Process Developer で実行をシミュレートできます。

シミュレーションを行うには有効なプロセスが必要で、実行中のプロセスが通常送受信するデータをシミュレートするために、特定の入力および出力サンプルデータ値を指定する必要があります。

プロセスの実行をシミュレートする準備ができていることを確認します。[「シミュレーションの前提条件」 \(ページ 417\)](#)を参照してください。Process Developer または [デバッグ] パースペクティブからシミュレーションを開始できます。

詳細については、以下のトピックを参照してください。

- [「BPEL プロセスのシミュレーションの開始と終了」 \(ページ 417\)](#)
- [「BPEL プロセスでのブレークポイントまでの実行」 \(ページ 418\)](#)
- [「BPEL シミュレーションの次のアクティビティへのステップ実行」 \(ページ 419\)](#)
- [「アクティビティまたはリンクの実行状態の表示」 \(ページ 419\)](#)
- [「シミュレーション中の BPEL プロセスの変更」 \(ページ 420\)](#)
- [「BPEL プロセスシミュレーションの終了と削除」 \(ページ 420\)](#)
- [「プロセス実行状態のクリア」 \(ページ 421\)](#)

シミュレーションの前提条件

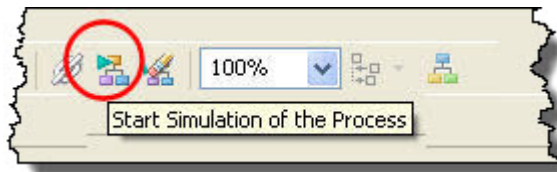
シミュレーションを開始する前に、次の手順を実行します。

- Receive、Pick、Invoke（または toPart）、および Reply に対するパートナーリンク、操作、変数（または fromPart）など、各アクティビティに必要なすべてのプロパティが入力されていることを確認します。
- プロセスの [抽象プロセス] プロパティが [いいえ] に設定されていることを確認します。デフォルトは [いいえ] です。
- 実行中のプロセスが送受信する入力変数、フォールト変数、および出力変数のサンプルデータを利用できるようにします。詳細については、「[シミュレーション中のサンプル変数データの入力および検査](#)」(ページ 421)を参照してください。
- プロセスに Pick アクティビティが含まれている場合は、実行するブランチを選択できます。詳細については、「[シミュレーションパスとプロパティの選択](#)」(ページ 424)を参照してください。

BPEL プロセスのシミュレーションの開始と終了

プロセスをシミュレートする手順

1. BPEL ファイルを開きます。
2. Process Editor キャンバスでマウスをクリックして、エディタツールバーをアクティブにします。
3. 図のように、ツールバーの **[プロセスのシミュレート]** アイコンを選択します。



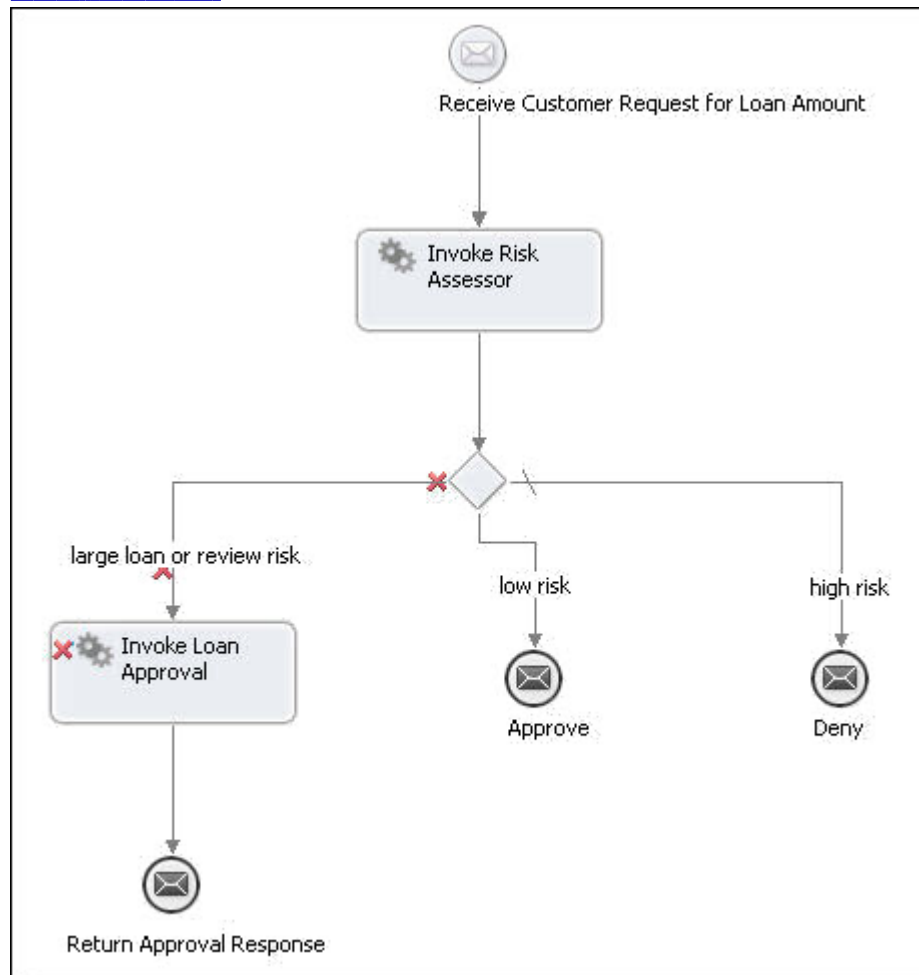
シミュレーションターゲットは、プロセスの最初のアクティビティで停止し、実行が一時停止されます。サンプルデータを追加していない場合は、追加するように求められます。

4. [デバッグ] ビューで、次のいずれかを実行します。
 - 完了するか、またはブレークポイントまで実行するには、**[再開]** アイコンをクリックします
 - 次のアクティビティまたはブレークポイントに進むには、**[ステップオーバー]** アイコン (F6 キー) をクリックします
 - While、If、For Each などのコンテナにステップインするには、**[ステップイン]** アイコン (F5 キー) をクリックします。

プロセスを実行すると、次のイベントが発生します。

- それぞれのアクティビティが、実行の準備として強調表示されます
- 必要に応じて、不足しているサンプルデータを追加するように促すメッセージが表示されます
- [プロセス変数] ビューで、シミュレーションの開始時にサンプルデータがクリアされ、対応するアクティビティの実行時にそのデータが表示されます
- [コンソール] ビューにシミュレーションイベントが表示されます
- [アウトライン] ビューに、すべてのアクティビティの実行が表示され、一部のアクティビティはキャンバスで非表示となります

- 次の図に示すように、エラーが発生するとエラーメッセージがポップアップし、シミュレーションが終了して、アクティビティにXのマークが付きます。詳細については、[「シミュレーション中の標準フォルトの検査」](#)（ページ 427）を参照してください。



シミュレーションスレッドを終了するには、最後までステップ実行するか、ステップを終了します。

[デバッグ] ビューにあるプロセスのスレッドのコンテキストメニューから **【終了】** を選択すると、シミュレーションを終了できます。

シミュレーションが終了すると、プロセスダイアグラムにシミュレーションパスが表示されます。上の図は、ステップが実行されたパス（強調表示）とステップが実行されていないパス（グレー表示）を示しています。

BPEL プロセスでのブレークポイントまでの実行

BPEL プロセスをデバッグしている場合、最初のアクティビティが実行される前にシミュレーションが一時停止します。

実行を再開して、プロセス内のアクティビティの横に設定したブレークポイントまで実行することができます。ブレークポイントの設定については、[「BPEL プロセスシミュレーションでのブレークポイントの使用」](#)（ページ 413）を参照してください。

ブレークポイントまで実行する手順

1. [デバッグ] ビューでプロセスを選択します。

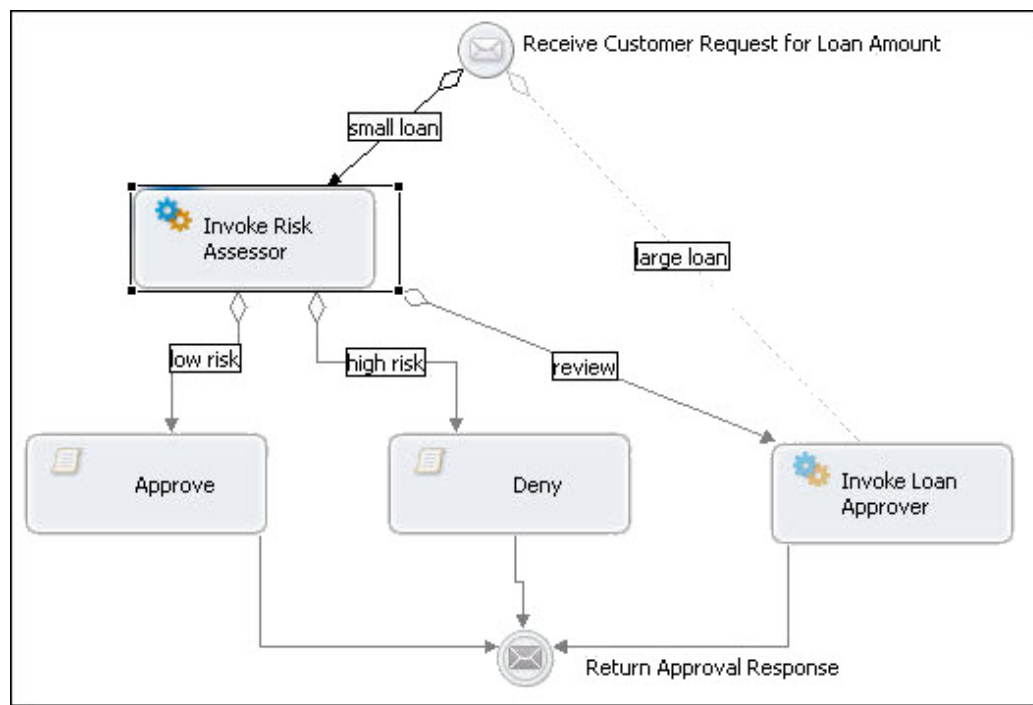
2. [デバッグ] ビューツールバーの[再開] ボタンをクリックします（または F8 キーを押します）。プロセスによる実行が再開されます。

BPEL シミュレーションの次のアクティビティへのステップ実行

一度に 1 つのアクティビティで、BPEL プロセスの実行をステップスルーできます。シミュレーションを進めている間に、コンソールで実行結果を確認できます。また、キャンバス上の実行パスとアウトラインをたどることもできます。

[デバッグ] ビューツールバーの[ステップオーバー] ボタンをクリックするか、F6 キーを押します。現在選択中のアクティビティが実行され、次の実行可能アクティビティで一時停止します。または、[ステップイン] (F5 キー) をクリックします。これにより、while、if、for Each などのコンテナにステップインします。

次の図は、実行パスの一度に 1 つのステップを示しています。



ヒント:

- ステップ操作中にブレークポイントが検出された場合、実行はブレークポイントで中断され、ステップ操作は終了します。
- Receive、Invoke、Reply アクティビティの入力、出力、またはデータタブを使用してデータをマッピングしている場合は、[アウトライン] ビューを使用してステップインの進行状況を表示できます。これらのアクティビティによって、キャンバスには表示されずにアウトラインにのみ表示される Assign アクティビティを持つスコープが生成されます。

アクティビティまたはリンクの実行状態の表示

- アクティブでない
- アクティブ
- 実行準備完了

- 実行中
- パスの停止（リンクまたは条件付き実行があるため、実行されません）
- 実行完了（アクティビティのみ）
- フォールトになった実行（アクティビティのみ）

アクティビティまたはリンクの「プロパティ」ビューで実行状態を確認できます。

シミュレーション中の BPEL プロセスの変更

シミュレーション中に BPEL プロセスに変更を加えることができます。例えば、サンプルデータ値、式、関連プロパティの値、パートナーリンクのアドレスなどを変更できます。

詳細については、次を参照してください:

- [「シミュレーション中のサンプル変数データの入力および検査」](#)（ページ 421）
- [Updating Correlation Property Data](#) コンソールに相関違反が表示された場合は、[関連プロパティの値を更新します。](#)
- [Updating Partner Link Address Information](#) シミュレーションまたはデバッグ中に、[パートナーリンクアドレスを変更する必要がある場合があります。](#)これは、[プロセスデプロイメント記述子（PDD）ファイルで指定するか、プロセスで動的に割り当てることができます。](#)
- [Correcting, Retrying, or Completing Activities](#)

シミュレータでは、この変更によって、プロセスモデルが現在実行中のプロセスと一致しなくなっていないかどうかを判断します。

BPEL プロセスシミュレーションの終了と削除

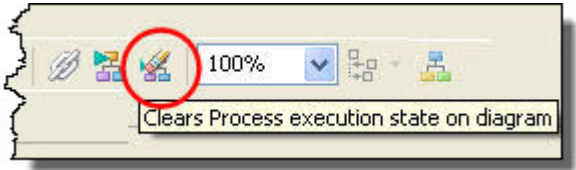
次のいずれかのオプションを使用して、シミュレーションを停止し、[デバッグ] ビューからシミュレーションを削除します。

オプション	場所	説明
終了	[デバッグ] ビューツールバーと [実行] メニュー シミュレーションスレッドの右マウスメニュー コンソールツールバー	関連するプロセスのシミュレーションを終了します
すべて終了	シミュレーションスレッドの右マウスメニュー	アクティブなシミュレーションをすべて終了します
終了して再起動	シミュレーションスレッドの右マウスメニュー	関連するプロセスのシミュレーションを終了し、再起動します
終了して削除	シミュレーションスレッドの右マウスメニュー	関連するプロセスのシミュレーションを終了し、ビューから削除します
終了したすべてを削除	[デバッグ] ビューツールバー シミュレーションスレッドの右マウスメニュー	終了したシミュレーションをビューからすべて削除します

プロセス実行状態のクリア

BPEL プロセスシミュレーションを終了すると、アクティビティとリンクに追加された強調表示が Process Editor に表示されたままになり、各アクティビティとリンクのシミュレーション状態が [プロパティ] ビューに表示されます。

強調表示とシミュレーションの状態を削除するには、図のように **【実行状態のクリア】** アイコンをクリックします。



シミュレーション中のサンプル変数データの入力および検査

シミュレーションを開始する前に、特定の入力および出力変数をサンプルデータの値で初期化する必要があります。プロセスを実行する実際のデータでシミュレーションすると、通常どおり受信および送信される、ということです。具体的には、次の変数のサンプルデータ値を追加します。

- プロセスを開始する Receive アクティビティまたは onMessage アクティビティの変数
- Invoke アクティビティの出力変数。これは、呼び出された Web サービスからプロセスに戻されるデータです。
- イベントハンドラの OnEvent 変数
- シミュレーションする必要があるフォールト変数

Process Developer には、プロセス変数のサンプルデータ値を入力する方法がいくつか用意されています。プロセスの実行をシミュレートするときは、さまざまなデータ値を使用してプロセス内のさまざまな実行パスをテストします。

次のように、サンプルデータを追加、ロード、変更します。

メッセージデータのサンプルデータファイルの追加	[パーティシパント] ビューで、WSDL メッセージを表示したら、メッセージごとに 1 つ以上のサンプルデータファイルを追加できます。メッセージごとに 1 つのデータファイルで、メッセージパートのデフォルト値を定義します。サンプルデータの追加については、 「WSDL メッセージのサンプルデータの使用」 (ページ 113) を参照してください。
メッセージデータのデフォルト値の選択	メッセージに複数のデータファイルを追加する場合、デフォルトとして 1 つのファイルを指定できます。 「デフォルトのサンプルデータファイルの選択」 (ページ 115) を参照してください。

Receive、Invoke、onMessage、および onEvent アクティビティで使用するプロセス変数のデフォルト値の変更	プロセスの実行をシミュレートするときに、入力、出力、フォールトメッセージのデフォルト値をオーバーライドできます。詳細については、「 入出力メッセージおよびフォールトメッセージのサンプルデータ値の設定 」(ページ 422)を参照してください。
サンプルデータ値のロードによるプロセス変数の初期化	特定の BPEL プロセスの場合、プロセス変数ごとにデータ値を入力するか、またはデータファイルをロードできます。詳細については、「 [プロセス変数] ビューでのサンプルデータの使用 」(ページ 249)を参照してください。

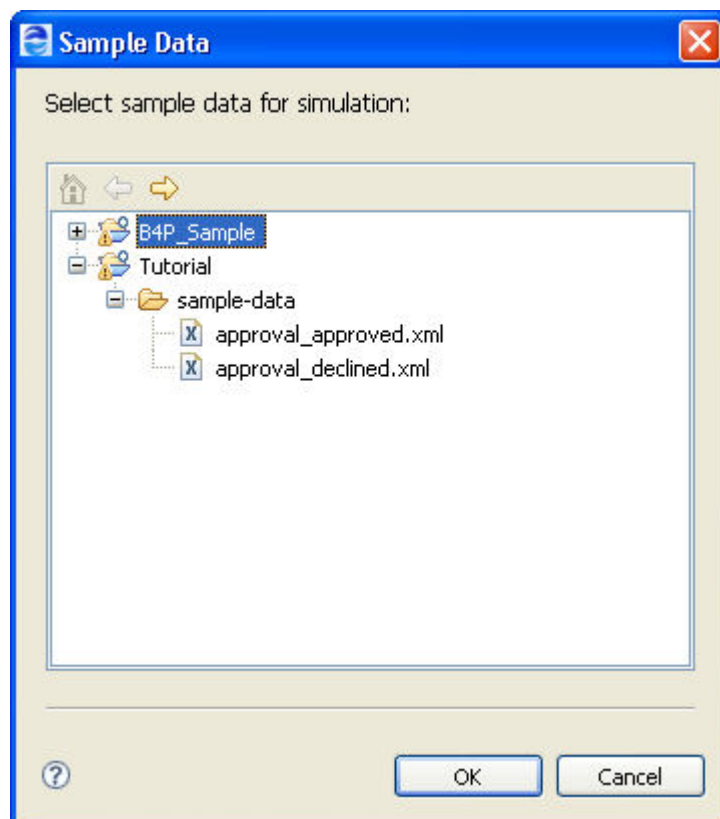
プロセスのシミュレーション中、変数の割り当てを表示できます。詳細については、「[シミュレーション中のプロセス変数の検査](#)」(ページ 423)を参照してください。

入出力メッセージおよびフォールトメッセージのサンプルデータ値の設定

追加するサンプルのソースの場所を選択します。

Receive、Invoke、onMessage、および onEvent アクティビティからプロセスによって受信される可能性のあるさまざまなデータ値を使用して実行をシミュレートできます。サンプル値は、アクティビティの [パーティシパント] ビュー、[プロセス変数] ビュー、または [プロパティ] ビューからロードできます。

プロジェクトエクスプローラの sample-data フォルダでサンプルデータファイルを作成した場合は、シミュレーション中にデータファイルを選択できます。サンプルデータがないメッセージの場合は、例に示すように、[サンプルデータ] ダイアログに適切な選択肢が表示されます。

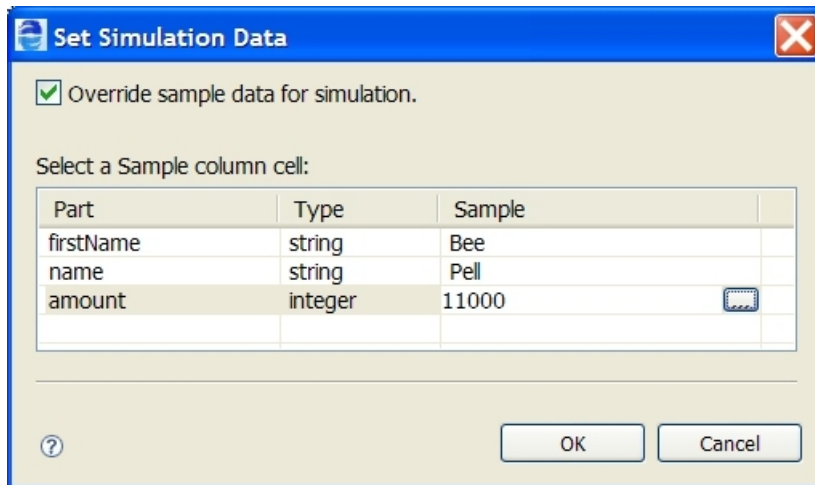


シミュレーションを開始する前に、サンプルデータをロードできます。[パーティシパント] ビューまたは [プロセス変数] ビューでのサンプルデータのロードと保存については、[「\[プロセス変数\] ビューでのサンプルデータの使用」 \(ページ 249\)](#)を参照してください。

シミュレーション中に、実行されていないアクティビティのデータ値をオンザフライで上書きできます。

ロードしたサンプルデータを入力メッセージと出力メッセージに上書きする手順

1. Process Editor キャンバスから、Receive、Invoke、onMessage、または onEvent アクティビティを選択します。
2. [プロパティ] ビューで、[すべて] タブを選択し、次のいずれかを実行します。
 - Receive、onMessage、または onEvent の場合は、[入力メッセージ] または [開始パート] 行の **ダイアログ (...)** をクリックします。
 - Invoke の場合は、[出力メッセージ] または [終了パート] 行の **ダイアログ (...)** をクリックします。
3. **[シミュレーションデータの設定]** ダイアログで、次の手順を実行します。
 - [シミュレーション用のサンプルデータを上書きします] の横にあるチェックボックスを選択します。
 - 図のように、行の最後にあるダイアログボタンを選択します。



4. 単純なデータ型の場合は、[サンプル] ダイアログでデータ値を編集し、**[OK]** をクリックします。複合的なメッセージタイプの場合は、データファイルをロードします。
5. 他のメッセージパートの追加の値を編集して、**[OK]** をクリックします。

ロードしたサンプルデータを呼び出しフォールトメッセージに上書きする手順

1. Process Editor キャンバスから、出力とフォールトメッセージが定義された Invoke アクティビティを選択します。
2. [プロパティ] ビューで、[結果] プロパティを [フォールト] に設定します。
3. リストからフォールト名を選択します。
4. フォールトメッセージを選択し、**[ダイアログ (...)]** をクリックします。
5. **[サンプルデータの設定]** ダイアログで、フォールトメッセージの値を入力します。

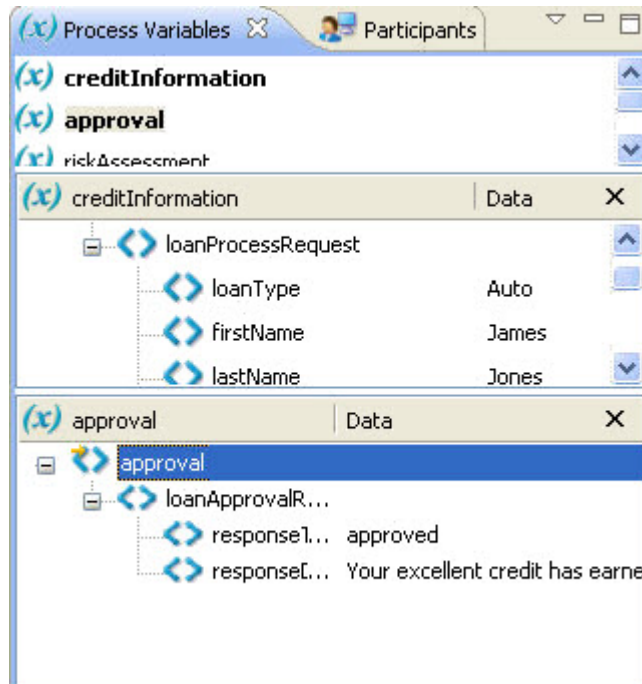
シミュレーション中のプロセス変数の検査

シミュレーション中に、各アクティビティが実行された場合に、[プロセス変数] ビューで変数値を検査できます。これらの値は、プロセス変数、または Pick の Receive、Invoke、onMessage ブランチ、およびイベント

ハンドラの onEvent ブランチのシミュレーションプロパティにロードした値です。データ値のロードと変更については、「[シミュレーション中のサンプル変数データの入力および検査](#)」(ページ 421)を参照してください。

必要に応じて変数を開いたり閉じたりするなど、変数値を検査するために推奨される方法を以下に示します。

1. [プロセス変数] ビューを表示します。
ヒント: すべての変数を閉じた状態で変数のリストを表示します。すべての変数を閉じるには、リストから変数を選択し、右マウスメニューから **[すべて閉じる]** を選択します。
2. シミュレーションを開始します。詳細については、「[BPEL プロセスの実行のシミュレーション](#)」(ページ 416)を参照してください。
3. Pick アクティビティの最初の Receive または onMessage ブランチに移動します。詳細については、「[BPEL シミュレーションの次のアクティビティへのステップ実行](#)」(ページ 419)を参照してください。
4. [プロセス変数] ビューで、強調表示された変数をダブルクリックして開きます。
5. 変数を右クリックして、**[データの表示]** を選択します。
6. 使用中の開いている変数のデータビューに現在のデータが表示されることに注意してください。



7. 次のアクティビティに進み、現在のアクティビティを実行します。ステップスルーしながら、データビューで変数を開いて、現在の値を調べます。

シミュレーションパスとプロパティの選択

シミュレートするエンドポイント参照アドレスを追加します。

特定の BPEL 要素を変更して、さまざまなシミュレーションイベントを発生させることができます。

例えば、受信メッセージ変数のデータ値を変更して、リンク遷移を true または false にすることができます。出力メッセージとフォールトメッセージのサンプルデータ値を変更することもできます。

同様に、必要に応じて相関プロパティの値を更新したり、パートナーリンクのアドレス情報を更新したりすることができます。

次の表に、変更可能な各 BPEL 要素のシミュレーションプロパティを示します。

アクティビティ	シミュレーション プロパティ	説明
Receive	入力メッセージ 添付の入力	シミュレーションで受信するメッセージのインスタンスデータを設定します
Pick	実行	シミュレーションのために実行する onMessage または onAlarm プランチを選択します。Pick をシミュレートするには、オプションを選択する必要があります。
Pick または onEvent イベント ハンドラの onMessage プラ ンチ	入力メッセージ	シミュレーションで受信するメッセージのインスタンスデータを設定します
Invoke	出力メッセージ 出力添付ファイル	シミュレーションで返されるメッセージのインスタンスデータを設定します
	結果	出力メッセージの結果を正常またはフォールトに設定します
	フォールト名	結果がフォールトに設定されている場合、リストからフォールト名を選択できます。フォールト名は、関連する WSDL ファイルで定義されています。
	フォールトメッセ ージ フォールト添付フ ァイル	シミュレートされたフォールトメッセージのインスタンスデータを設定します。
	サブプロセス	詳細については、「 シミュレーション用の呼び出しサブプロセスの選択 」(ページ 426)を参照してください。
Wait	待機時間（秒単 位）	秒単位でのシミュレーションの実際の待機時間。アクティビティの待機時間は、シミュレーション中の待機時間よりも長くなる可能性があります。これはその時間を短縮するための方法です。
パートナーリン ク	マイロールとパー トナーロール	コピー操作内の動的割り当てで使用されるエンドポイント参照アドレス

次の表に、変更可能な各 BPEL 要素の実行プロパティを示します。実行プロパティはシミュレーションまたはリモートデバッグ中にのみ使用可能で、通常は「プロパティ」ビューには表示されないことに注意してください。

要素	実行プロパティ	説明
相関セット	相関セットデータ	相関プロパティの現在の値を更新します
パートナーリンク	アドレス	パートナーリンクのアドレス情報を追加または更新します

相関セットを使用するシナリオの詳細については、[第 26 章、「プロセス例外管理」](#) (ページ 398)を参照してください。

シミュレーション用の呼び出しサブプロセスの選択

別のプロセスを選択して、サーバーにデプロイ済みのプロセスと同期します。この Invoke アクティビティが BPEL プロセスで定義されている場合は、シミュレーション中にステップインできます。

サブプロセスは、別の BPEL プロセスによって呼び出される BPEL プロセスです。1 つの BPEL プロセスを作成してデプロイし、結果のサービスの操作を使用して Invoke アクティビティを作成できます。

シミュレーション中に、Invoke アクティビティのシミュレーションプロパティであるサブプロセスを使用して、BPEL プロセスを選択できます。この選択を行うことで、サブプロセスにステップインし、Invoke アクティビティにステップ処理する際にサブプロセスをシミュレートできます。サブプロセスの作成の詳細については、「別の BPEL プロセスのサービスとしての BPEL プロセスの作成」を参照してください。

シミュレーション用の呼び出しサブプロセスを選択する手順

1. BPEL プロセスを呼び出す Invoke アクティビティを選択します。
2. [プロパティ] ビューで、[すべて] を選択します。
3. [シミュレーション] カテゴリで、[サブプロセス] を選択します。
4. 行の最後にある [ダイアログ] ボタンをクリックして、[プロセスの選択] ダイアログを開きます。
5. ワークスペースからプロセスを選択します。

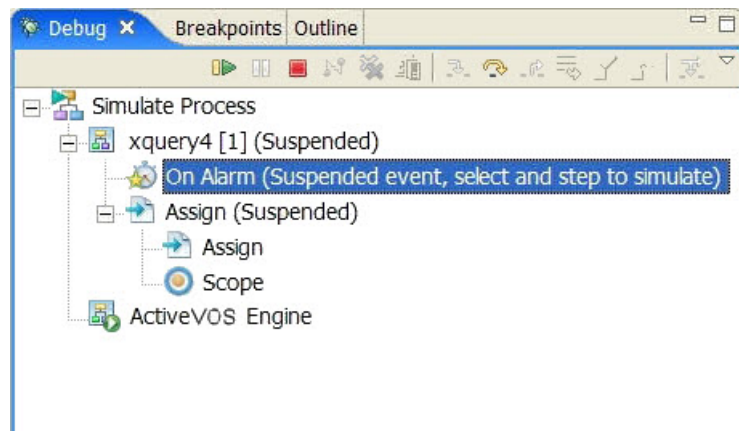
サブプロセスにプロセスレベルの補償ハンドラがある場合は、必ず補償ハンドラのアクティビティにブレークポイントを設定します。補償ハンドラにブレークポイントが設定されていない場合、シミュレーションはブレークポイントにステップインせず、ブレークポイントをステップオーバーします。

シミュレーション中は、プロセスロジックに応じて、メインプロセスとサブプロセスの間で実行ステップを進めたり戻したりできることに注意してください。[デバッグ] ビューで、次に実行する実行スレッドを選択します。

イベントハンドラのシミュレーション

シミュレーション中、メインプロセスとイベントハンドラには異なるデバッグターゲットが存在します。スコープのシミュレーションの任意の時点でイベントハンドラにステップインし、メインターゲットを選択して、メインプロセスでスコープのシミュレーションを続行できます。

次の図は、シミュレーションターゲットを示しています。



フォールトハンドラのシミュレーション

シミュレーション中に、関連する Invoke アクティビティのシミュレーションプロパティを設定することで、フォールトハンドラを実行できます。

[プロパティ] ビューで、[結果] プロパティを [フォールト] に設定し、適切なフォールト名と変数を選択します。

詳細については、[「入出力メッセージおよびフォールトメッセージのサンプルデータ値の設定」](#) (ページ 422) を参照してください。

シミュレーション中の標準フォールトの検査

WS-BPEL 仕様では、一部の標準的なフォールトが定義されています。シミュレーション中にこれらのフォールトが発生した場合、シミュレーションは終了し、フォールトが発生したアクティビティの横に赤い X が表示されます。また、コンソールにフォールトのリストが表示されます。以下に例を示します。

```
Assign AssignYestoAccept:
  Completed with fault: mismatchedAssignmentFailure :
    </process/flow/assign[@name='AssignYestoAccept']>
```

シミュレーションの設定

シミュレーション中の変数処理に関する設定を行います。

Process Developer でプロセスの実行をシミュレートすると、シミュレーションエンジンは、デプロイされたプロセスが実行されるサーバーエンジンと同じように動作します。設定を行ってサーバーエンジンの動作を変更したり、シミュレーションエンジンに同じ内容を設定したりすることができます。

サーバーエンジンの動作の設定に関する詳細については、プロセスサーバーのドキュメントを参照してください。

次の表で説明するシミュレーション設定を表示するには、[ウィンドウ] > [設定] > [Process Developer] > [シミュレーション] を選択します。

シミュレーションの設定	説明
入力/出力メッセージをスキーマと照合して検証	プロセス変数にロードされたサンプルデータを WSDL スキーマに照らし合わせて検証します。複合的なメッセージにサンプルデータファイルを追加し、そのデータファイルが無効な場合、Process Developer では警告が表示され、エラーがあってもデータファイルを追加できます。このオプションを有効にすると、シミュレーションを開始する前にデータを検証できます。シミュレーションを高速化するには、このオプションを無効にします。このオプションはデフォルトで有効になっています。
bpws:selectionFailurefault の無効化 BPEL バージョン 1.1 のみ	このオプションを有効にすると、XPath（または他の言語）のクエリ文字列を含む関数または割り当てから null 値が返されるようにすることができます。オプションの要素を使用してデータサンプルを処理する場合、このオプションを有効にして動作を上書きできます。 デフォルトでは、このオプションは有効ではありません。クエリ文字列が assign copy FROM から空の選択を返すと、プロセスは bpws:selectionFailure フォールトをスローします。これは、BPEL4WS 仕様で説明されている標準の応答です。 すべての WS-BPEL 2.0 プロセスのプロセス拡張機能としてこのオプションを含めるように設定を変更する場合は、「 <i>Process Developer の設定</i> 」を参照してください。 詳細については、「 <i>bpel:selectionFailure フォールトの無効化の例</i> 」および「 <i>bpel:selectionFailure フォールトの無効化の例と Copy/To のターゲットパスの自動作成の例</i> 」を参照してください。
Copy/To のターゲットパスの自動作成 BPEL バージョン 1.1 のみ	Process Developer が、プロセスインスタンスドキュメントの複合変数に存在しないノードのロケーションパスを作成できるかどうかを決定します。割り当てが、存在しないノード（または複数のノード）を参照している場合、標準の BPEL フォールト (bpws:selectionFailure) が BPEL 仕様に従ってスローされます。 このオプションを有効にすると、選択が即座に作成されます。つまり、assign copy TO 操作は、存在しないノードを参照し、それに値を割り当てることができます。このオプションはデフォルトで無効になっています。 WS-BPEL 2.0 プロセスのプロセス拡張機能としてこのオプションを含めるように設定を変更する場合は、「 <i>Process Developer の設定</i> 」を参照してください。 「 <i>Copy/To のターゲットパスの自動作成の例</i> 」および「 <i>bpel:selectionFailure フォールトの無効化の例 Copy/To のターゲットパスの自動作成の例</i> 」を参照してください。また、詳細については、「 <i>bpel:selectionFailure フォールトの無効化の例と Copy/To のターゲットパスの自動作成の例</i> 」を参照してください。

bpel:selectionFailure フォールトの無効化の例

次の例は、[bpel:selectionFailure フォールトの無効化] (BPEL4WS 1.1 プロセス: bpws:selectionFailure) オプションを有効または無効にした場合の影響を示しています。

WS-BPEL 2.0 プロセスの詳細については、「*Process Developer の無効な選択エラーのフォールト拡張機能の使用*」を参照してください。

この BPEL4WS 1.1 設定の説明については、「*シミュレーションの設定*」を参照してください。

コードサンプル:

```
<assign>
  <copy>
    <from part="OrderInfo" query="/ns1:OrderInfo
      /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
      variable="var1"/>
    <to part="OrderInfo" query="/ns1::Addr1"
      variable="var2"/>
  </copy>
</assign>
```

このコードサンプルでは、以下のスキーマスニペットに示すように、[Assign From/To] クエリはオプションの要素に対するものです。

Var1 および Var2 のスキーマスニペット:

```
<xs:complexType
  name="AddressInfoType">
  <xs:sequence>
    <xs:element ref="ord:Name"/>
    <xs:element ref="ord:Addr1" minOccurs="0"/>
    <xs:element ref="ord:Addr2"
      />
    <xs:element ref="ord:City"/>
    <xs:element ref="ord:St"/>
    <xs:element ref="ord:Zip"/>
    <xs:element ref="ord:Cntry"
      />
  </xs:sequence>
</xs:complexType>
<xs:element name="BillToInfo"
  type="ord:AddressInfoType"/>
```

Var1 のサンプルデータ	Var2 の初期化
<pre><ns1:OrderInfo xmlns:ns1="http://temp.com"> <ns1:OrderHeader> <ns1:OrdId>78</ns1:OrdId> <ns1:BillToInfo> <ns1:Name>Name1</ns1:Name> (Addr1 is missing) <ns1:Addr2>1 Main St </ns1:Addr2> <ns1:City>Albany</ns1:City> <ns1:St>NY</ns1:St> <ns1:Zip>12012</ns1:Zip> <ns1:Cntry>USA</ns1:Cntry> </ns1:BillToInfo> ...</pre>	<pre><ns1:OrderInfo xmlns:ns1="http://temp.com"> <ns1:OrderHeader> <ns1:OrdId/> <ns1:BillToInfo> <ns1:Name/> <ns1:Addr1/> <ns1:Addr2/> <ns1:City/> <ns1:St/> <ns1:Zip/> <ns1:Cntry/> </ns1:BillToInfo> ...</pre>

[bpel:selectionFailure フォールトの無効化] を無効にすると、Var1 にクエリ選択ノードがなく、Var2 クエリ選択ノードに割り当てることができないため、シミュレーションは選択エラーフォールトで終了します。

[bpel:selectionFailure フォールトの無効化] を有効にすると、空の選択ノードが許可されるため、プロセスは正常に終了します。割り当て中に Addr1 に NULL 値が追加されます。

Copy/To のターゲットパスの自動作成の例

次の例は、[Copy/To のターゲットパスの自動作成] オプションを有効にした場合、または無効にした場合の影響を示しています。

WS-BPEL 2.0 のプロセスの詳細については、[「Process Developer の XPath 作成拡張機能の使用」 \(ページ 125\)](#)を参照してください。

この BPEL4WS 1.1 の設定の説明については、[「シミュレーションの設定」 \(ページ 427\)](#)を参照してください。

コードサンプル:

```
<assign>
  <copy>
    <from part="OrderInfo" query="/ns1:OrderInfo
      /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
      variable="var1"/>
    <to part="OrderInfo" query="/ns1:OrderInfo
      /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
      variable="var2"/>
  </copy>
</assign>
```

このコードサンプルでは、以下のスキーマスニペットに示すように、[Assign From/To] クエリはオプションの要素に対するものです。

Var1 および Var2 のスキーマスニペット:

```
<xs:complexType name="AddressInfoType">
  <xs:sequence>
    <xs:element ref="ord:Name"/>
    <xs:element ref="ord:Addr1" minOccurs="0"/>
    <xs:element ref="ord:Addr2"/>
    <xs:element ref="ord:City"/>
    <xs:element ref="ord:St"/>
    <xs:element ref="ord:Zip"/>
    <xs:element ref="ord:Cntry"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="BillToInfo" type="ord:AddressInfoType"/>
```

Var1 のサンプルデータ	Var2 の初期化
<pre><ns1:OrderInfo xmlns:ns1="http:// temp.com"> <ns1:OrderHeader> <ns1:OrdId>78</ns1:OrdId> <ns1:BillToInfo> <ns1:Name>Name1</ns1:Name> <ns1:Addr1>Apt 12</ns1:Addr1> <ns1:Addr2>1 Main St </ns1:Addr2> <ns1:City>Albany</ns1:City> <ns1:St>NY</ns1:St> <ns1:Zip>12012</ns1:Zip> <ns1:Cntry>USA</ns1:Cntry> </ns1:BillToInfo> ...</pre>	<pre><ns1:OrderInfo xmlns:ns1="http:// /temp.com"> <ns1:OrderHeader> <ns1:OrdId/> <ns1:BillToInfo> <ns1:Name/> (Addr1 is missing) <ns1:Addr2/> <ns1:City/> <ns1:St/> <ns1:Zip/> <ns1:Cntry/> </ns1:BillToInfo> ...</pre>

[Copy/To のターゲットパスの自動作成] を無効にした場合、Var1 にはクエリ選択ノードが含まれるため、シミュレーションはフォールトで終了しますが、ロケーションパスがないため、Var1 は Var2 クエリ選択ノードには割り当てられません。

[Copy/To のターゲットパスの自動作成] を有効にした場合、Addr1 にロケーションパスが作成され、割り当て中に値が Addr1 へ追加されるため、プロセスは正常に終了します。

bpel:selectionFailure フォールトの無効化の例と Copy/To のターゲットパスの自動作成の例

次の例は、Disable bpel:selectionFailure フォールト (BPEL4WS 1.1 プロセス: bpws:selectionFailure) および [Copy/To のターゲットパスの自動作成] オプションを示しています。

WS-BPEL 2.0 プロセスの詳細については、「*Process Developer の XPath 作成拡張機能の使用*」および「*Process Developer の無効な選択エラーのフォールト拡張機能の使用*」を参照してください。

これらの BPEL4WS 1.1 の設定に関する説明については、「*シミュレーション設定*」を参照してください。

コードサンプル:

```
<assign>
  <copy>
    <from part="OrderInfo" query="/ns1:OrderInfo
      /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
      variable="var1"/>
    <to part="OrderInfo" query="/ns1:OrderInfo
      /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
      variable="var2"/>
  </copy>
</assign>
```

このコードサンプルでは、以下のスキーマスニペットに示すように、[Assign From/To] クエリはオプションの要素に対するものです。

Var1 のサンプルデータ	Var1 および Var2 のスキーマスニペット
<pre><ns1:OrderInfo xmlns:ns1="http:// temp.com"> <ns1:OrderHeader> <ns1:OrdId>78</ns1:OrdId> <ns1:BillToInfo> <ns1:Name>Name1</ns1:Name> (Addr1 is missing) <ns1:Addr2>1 Main St </ns1:Addr2> <ns1:City>Albany</ns1:City> <ns1:St>NY</ns1:St> <ns1:Zip>12012</ns1:Zip> <ns1:Cntry>USA</ns1:Cntry> </ns1:BillToInfo> ...</pre>	<pre><xs:complexType name="AddressInfoType"> <xs:sequence> <xs:element ref="ord:Name"/> <xs:elementref="ord:Addr1"minOccurs="0"/> <xs:element ref="ord:Addr2" /> <xs:element ref="ord:City"/> <xs:element ref="ord:St"/> <xs:element ref="ord:Zip"/> <xs:element ref="ord:Cntry" /> </xs:sequence> </xs:complexType> <xs:element name="BillToInfo" type="ord:AddressInfoType"/></pre>

[*bpws:selectionFailure* フォールトを無効化] および [*Copy/To* のターゲットパスの自動作成の有効化] を使用すると空の選択ノードが許可されるため、プロセスは正常に終了し、空の選択へのロケーションパスが作成されます。割り当て中に Addr1 に NULL 値が追加されます。

デバッグの設定

メインツールバーから [ウィンドウ] > [設定] を選択すると、[デバッグ] パースペクティブの設定のページが表示されます。

多くの設定はデフォルトの Eclipse の設定であるため、Process Developer には適用されません。

次の設定が、Process Developer に適用されます。

実行/デバッグ設定ページ	適用される設定
コンソール	このページのすべての設定が、プロセスコンソールに適用されます
外部ツール	適用されません

実行/デバッグ設定ページ	適用される設定
起動中	新しい起動の作成時に終了した起動が削除されます
文字列置換	適用されません
ビュー管理	適用されません

第 29 章

BPEL ユニットテスト

次のトピックでは、BPEL ユニットテストについて説明します。

- [「BPEL ユニットテストについて」 \(ページ 433\)](#)
- [「BPEL ユニットテストファイルの作成」 \(ページ 434\)](#)
- [「Process Developer での BPEL ユニットテストの実行」 \(ページ 435\)](#)
- [「B ユニット Ant スクリプトの作成と実行」 \(ページ 437\)](#)
- [「B ユニットファイルの編集」 \(ページ 440\)](#)
- [「B ユニットテストのデバッグ」 \(ページ 445\)](#)
- [「BPEL ユニットテストスイートの作成と実行」 \(ページ 445\)](#)
- [「アサーションの使用に関するヒント」 \(ページ 447\)](#)
- [「入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント」 \(ページ 448\)](#)
- [「パートナーリンクデータの指定に関するヒント」 \(ページ 448\)](#)
- [「B ユニットファイルの例」 \(ページ 449\)](#)

BPEL ユニットテストについて

個々の B ユニットテストケースを手動で作成するか、実行するそれぞれのシミュレーションパスを B ユニットテストファイルとして保存します。

BPEL ユニット (B ユニット) テストは、プロセスサーバーエンジンで BPEL プロセスをテストするフレームワークです。B ユニットテストにより、Invoke、Receive、および Reply のプロセス実行を検証します。B ユニットテストを使用して、プロセスに対する自動回帰テストを実行できます。

B ユニットテストを実行すると、次の手順が実行されます。

- SendMessage コマンドが、XML データを処理に送信します。ユーザーは、B ユニットテストの編集または生成の一部としてこのデータを入力します。
- 応答がプロセスから返されます
- アサート式によって、応答に必要なものを決定します
- 結果がアサートと比較されます

テストを効果的に行うには、XML のチャンク全体を比較して、一致しない部分を確認します。

個々の B ユニットテストケースを手動で作成することもできますが、実行する各シミュレーションパスを B ユニットテストファイルとして保存することをお勧めします。その後、プロセスをリファクタリングしながら、

必要に応じてファイルを再実行できます。すぐに利用可能な B ユニットテストを使用することで、プロセスのコンポーネントが正しく機能しているかどうかを簡単に確認できます。

単体テストを設計して、プロセス全体のすべてのパスをカバーするテストケースを作成し、さまざまなシナリオをテストできます。ファイルのグループをスイートまたはスイートのスイートに保存して、再実行することができます。自動ビルド手順の一部として、Process Developer の外部でも B ユニットテストを実行できます。

詳細については、次を参照してください:

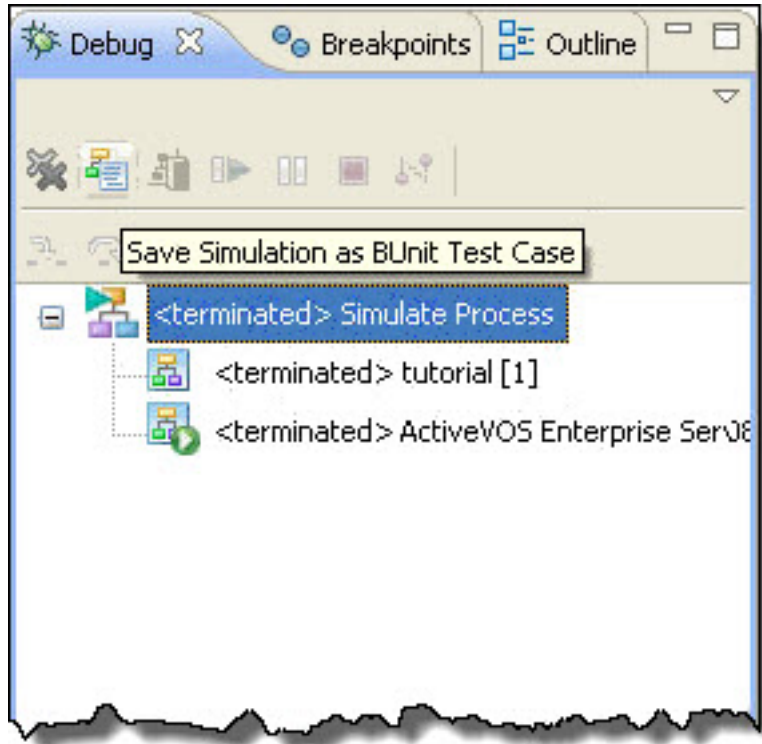
- [「BPEL ユニットテストファイルの作成」 \(ページ 434\)](#)
- [「Process Developer での BPEL ユニットテストの実行」 \(ページ 435\)](#)
- [「B ユニット Ant スクリプトの作成と実行」 \(ページ 437\)](#)
- [「B ユニットファイルの編集」 \(ページ 440\)](#)
- [「B ユニットテストのデバッグ」 \(ページ 445\)](#)
- [「BPEL ユニットテストスイートの作成と実行」 \(ページ 445\)](#)
- [「アサーションの使用に関するヒント」 \(ページ 447\)](#)
- [「入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント」 \(ページ 448\)](#)
- [「パートナーリンクデータの指定に関するヒント」 \(ページ 448\)](#)

BPEL ユニットテストファイルの作成

次のいずれかの方法を使用して、B ユニットテストファイルを作成します。

- (推奨) BPEL プロセスをシミュレートする。[デバッグビュー] ツールバーで、図に示すように **「シミュレーションを B ユニットテストケースとして保存」** を選択します。

- BPEL ファイルを右クリックして、**[新規] > [B ユニットテストケース]** を選択する。B ユニットエディタが開きます。このエディタでは、正常なプロセスおよびフォールトが発生したプロセスをテストするためのデータ値と要素属性を指定するファイルを基礎から手動で編集できます。詳細については、[「B ユニットファイルの編集」 \(ページ 440\)](#)を参照してください。



ヒント: B ユニットファイルは Orchestration プロジェクト内のテストフォルダに保存できます。ファイルを保存すると、保存場所に関係なく、B ユニットエディタでファイルが開きます。

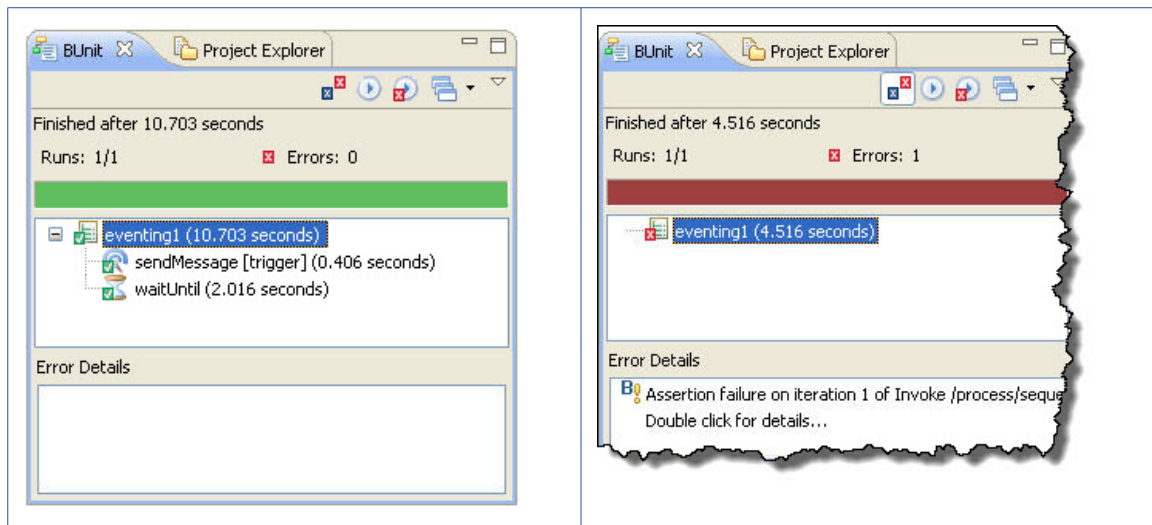
[「Process Developer での BPEL ユニットテストの実行」 \(ページ 435\)](#)および [「BPEL ユニットテストスイートの作成と実行」 \(ページ 445\)](#)を参照してください。

Process Developer での BPEL ユニットテストの実行

B ユニットテストは、Process Developer の外部の Ant スクリプトから、または Process Developer 内の [B ユニット] ビューから実行できます。コマンドライン実行の詳細については、[「B ユニット Ant スクリプトの作成と実行」 \(ページ 437\)](#)を参照してください。

プロジェクトエクスプローラで B ユニットテストまたはテストスイートを右クリックし、**[実行] > [B ユニット]** を選択するなど、Process Developer では複数の場所から [実行] コマンドを使用して B ユニットテストを実行できます。

[B ユニット] ビューが開き、テストが実行されます。次の例は、テストが完了し、テストが失敗した B ユニットを示しています。



「[BPEL ユニットテストスイートの作成と実行](#)」 (ページ 445)に記載されているように、B ユニットスイートから 1 つのテストを実行することもできます。

テストを実行すると、次のイベントが発生します。

- [B ユニット] ビューが開き、テストの進行状況が表示されます。
- コンソールにテストの出力が表示されます。デフォルトでは、トレースイベントとプロセスの実行が含まれます。これらのイベントの詳細については、[「B ユニットファイルの編集」](#) (ページ 440)を参照してください。
- テストが失敗すると、[B ユニット] ビューにエラーメッセージが表示されます。特定のテストエラーについては、失敗したテストをダブルクリックすると、テキスト比較エディタが開き、実際の結果と想定される結果が表示されます。
- テキスト比較エディタでは、想定される結果と実際の結果の違いが表示されます。[「正規化されたビューをオン/オフします」](#)を使用すると、ファイルの差異を簡単に見つけることができます。[「色とフォントの設定」](#) (ページ 41)に記載されているように、テキスト比較エディタの色設定を変更することもできます。

テストが完了した後に [B ユニット] ビューでイベントをクリックすると、B ユニットエディタ内の B ユニットテストの該当するセクションに直接移動できます。

[B ユニット] ビューでは、次のようなツールバーアイコンが使用できます。

- **失敗の表示。** 失敗したテストのみを表示する場合に選択し、表示される項目をフィルタリングします。
- **テストを再実行。** 1 つのテストを実行した後に、[B ユニット] ビューから再実行できます。
- **すべての失敗を再実行。** 失敗した複数のテストを [B ユニット] ビューから再実行します。
- **履歴。** 実行した 15 件までの単体テストが記憶されます。B ユニットファイルの名前に基づいて、リストからテストを選択することができます。

B ユニット Ant スクリプトの作成と実行

個々の B ユニットテストケースを手動で作成するか、実行するそれぞれのシミュレーションパスを B ユニットテストファイルとして保存します。

自動ビルド環境に B ユニットテストを含める場合は、コマンドラインで、Process Developer の外部にある複数の Orchestration プロジェクトからすべての B ユニットテストとスイートを実行する Ant スクリプトを作成します。この機能は、コマンドライン BPRD スクリプトを使用して BPR アーカイブをサーバーにデプロイする機能と類似しています。

前提条件

Ant スクリプトを実行するには、Process Developer の **B ユニット Ant ランタイム**コンポーネントをインストールする必要があります。このインストールにより、Process Developer の外部で、コマンドラインから B ユニット Ant スクリプトを実行するために必要なランタイムファイルを含むディレクトリが作成されます。Ant を個別にインストールする必要はなく、Apache Ant アプリケーションは Process Developer の一部としてインストールされます。

注: バージョン 1.7.0（またはそれ以降）の Ant が必要です。

B ユニット Ant スクリプトの作成

次の手順に従って、B ユニット Ant スクリプトを作成します。

1. [「BPEL ユニットテストファイルの作成」 \(ページ 434\)](#)に記載されているように、Orchestration プロジェクトに 1 つ以上の B ユニットテストまたはスイートが含まれていることを確認します。
2. 1 つの B ユニットテストまたはスイートを選択し、マウスを右クリックして **【新規】 > 【B ユニット Ant スクリプト】** を選択します。
3. **【B ユニット Ant スクリプトのエクスポート】** ウィザードで、**【テスト】** フォルダなどのフォルダを選択し、ファイル名を入力します。**【次へ】** を選択します。
4. スクリプトに含める B ユニットテストとスイートを選択します。複数のプロジェクトから複数のテストを選択することができます。また、スクリプトの作成後にスクリプトを編集して、テストのフォルダをさらに追加することもできます。
5. 必要に応じて、次のような B ユニットのオプションを選択します。
 - **トレースの有効化。** B ユニット固有の実行コマンドのコンソールログ出力を制御します。
 - **プロセスログの有効化。** ログに記録するプロセス実行データの数を制御します。デフォルトは **【なし】** です。このチェックボックスを有効にすると、**【実行】** レベルがオンになります。必要に応じて Ant スクリプトを手動で編集して、**【実行データ】**（データを使用した実行）または **【完全】** なログを有効にすることができます。
6. **【完了】** を選択して、`filename.ant.xml` ファイルを生成します。

Ant スクリプトがエディタで開き、表示または編集が可能になります。このスクリプトを Process Developer の外部で実行する場合は、Ant スクリプトの指示に従って B ユニットのランタイムクラスパスを設定します。

コマンドラインで B ユニット Ant スクリプトを実行すると、コード範囲レポートを作成できます。詳細については、[「コード範囲レポートの生成」 \(ページ 438\)](#)を参照してください。

B ユニット Ant スクリプトの変更に関する注意事項

Ant スクリプト内で、個々の B ユニットファイルのプロパティを上書きできます。

- 必要に応じて、`<config file="engine-configuration.xml" />`などのファイルを追加して、default-B-unit-engine の設定を上書きできます。
- ネストされた要素`<engineProperties>`は、`java.util.Properties` で必要な形式のエンジン `config path=value` のペアを持つプロパティファイルへのファイル相対パスまたは絶対パスを受け入れます。ファイル属性はオプションで、代わりにネストされた`<engineProperty name="..." value="..." />`要素を使用してエンジンプロパティを設定できます。ファイル属性および `engineProperty` 要素が使用されている場合、同じ構成パスが 2 回指定されていると、要素によってプロパティファイルの値が上書きされます。
- 複数のエンジンプロパティの例:
 - `<engineProperty name="Logging" value="urn:ae:execution" />`
 - `<engineProperty name="CustomFunctions/MyCustomFunction" value="com.test.my.custom.Function" />`

Ant スクリプトへの複数の B ユニットと B スイートの追加

Ant には、B ユニットファイルや B スイートファイルのグループなど、リソースコレクションを指定するための多くの方法が用意されています。一つの例を以下に示します。

```
<path>
  <fileset dir="${test.dir}" includes="**/*.bunit"/>
</path>
```

例については、Ant 1.7（またはそれ以降）のドキュメントの *リソースコレクションのセクション* を参照してください。

B ユニット Ant スクリプトの実行

必要に応じて、Process Developer 内から `filename.ant.xml` ファイルを実行できます。ファイルを右クリックして、**[実行] > [Ant ビルド]** を選択します。コンソールでは、トレースとログの詳細を表示できます。

コマンドラインで Ant スクリプトを実行するには、次の手順を実行します。

1. コマンドウィンドウを開きます。
2. Ant スクリプトが配置されているファイルシステムフォルダに移動します。
3. コマンドラインプロンプトで、`ant -f filename.ant.xml` と入力します。

コード範囲レポートの生成

B ユニットの Ant スクリプトを使用して、コード範囲レポートを生成できます。このレポートは実行パスを追跡し、実行されたアクティビティに関して、1 回以上実行されたアクティビティのパーセンテージ（すべてのプロセスのアクティビティに対する比率）として表示します。

コード範囲レポートを生成する手順

1. [「B ユニット Ant スクリプトの作成と実行」 \(ページ 437\)](#) の説明に従って、B ユニットの Ant スクリプトを作成します。
2. `<target>` という名前の `instrument` で、コメントを削除し、`bpel` ファイルのリソースコレクションに入ります。次の例は、単純なリソースコレクションを示しています。

```
<target depends="init" name="instrument">
  <!-- optional
  <coverage activitycoverage="true" linkcoverage="true" />
  -->
  <bunit-instrument>
```

```

    <filelist>
      <file name="../bpel/myFile.bpel"/>
    </filelist>
  </bunit-instrument>

```

デフォルトでは、アクティビティのコード範囲レポートが生成されます。リンクを含める場合は、オプションの<coverage>要素を追加します。

3. test という名前のターゲットで、<engine properties>セクションをコメント解除します。
4. ターゲットの名前付き範囲レポートで、次の手順を実行します。
 - レポートの出力先ディレクトリを指定します。
 - レポートタイプを指定します。タイプは、HTML（デフォルト）、XML（HTML の作成に使用される Raw XML）、または CONSOLE（コマンドラインで指定した Ant Logger に結果の要約を書き込みます）です。
 - HTML レポートを作成する場合は、xsltPath 属性を追加できます。
 - 必要に応じて、filePrefix の名前を変更します。Ant プロジェクト名はデフォルトの filePrefix です。レポートを生成すると、日付と時刻がプレフィックスに追加されます。
5. コマンドラインで B-unitant スクリプトを実行して、レポートを生成します。例えば、コマンドウィンドウで、ant -f filename.ant.xml と入力します。

コード範囲レポートに表示される内容

- アクティビティの範囲（デフォルトの範囲）：プロセス内のアクティビティの総数に対して、1 回以上実行されたアクティビティの総数。
- リnkの範囲（オプション）：条件付きのリンクの総数の 2 倍以上 true と評価されたアクティビティの総数と false と評価されたアクティビティの総数。

$$(totalTrue + totalFalse) / (totalLinksWithConditions * 2).$$
 これにより、条件付きの各リンクが true または false と評価される可能性があっても、ユニットテスト時にリンクの実行が完全に行われるようになります。

既存の B ユニット Ant スクリプトへのコード範囲の追加

既存の B ユニット Ant スクリプトがある場合は、B ユニット Ant スクリプトに正しい Ant タスクを入力するだけでコード範囲が追加されます。

- 次の typedef を Ant スクリプトに登録すると、属性 loaderref が bunit に設定されることに注意してください。すべての Ant タスクで同じクラスローダーが使用されるようにするには、この属性を設定する必要があります。3 つの typedef の値がすべて一致する場合、属性の値は重要ではありません。

```

<typedef classname="com.activee.ant.bunit.tasks.AeBUnitTask"
  loaderref="bunit" classpathref="bunit.classpath" name="bunit"/>
<typedef classname="com.activee.ant.bunit.coverage.AeBUnitInstrumentTask"
  loaderref="bunit" classpathref="bunit.classpath" name="bunit-instrument"/>
<typedef classname="com.activee.ant.bunit.coverage.AeBUnitCoverageReportTask"
  loaderref="bunit" classpathref="bunit.classpath" name="bunit-report"/>

```

- コード範囲の一部であるプロセスをインストルメントします。


```

<bunit-instrument>
  <coverage activitycoverage="true" linkcoverage="true" />
  <filelist>
    <file name="../bpel/myBEPL.bpel"/>
  </filelist>
</bunit-instrument>

```
- 以下のエンジンプロパティを、B ユニットテストを実行する B ユニットタスクに次のように追加します。

```

<bunit processLogging="urn:ae:full" trace="on">
  <engineProperties>
    <engineProperty name="CustomManagers/AeBUnitCoverageManager/Class"
      value="com.activee.rt.bunit.coverage.AeBUnitCoverageManager"/>
  </engineProperties>
  <filelist>

```

```

        <file name="CancelT0.bunit"/>
        <file name="CreateT0.bunit"/>
        <file name="SubmitPO_UNI.bunit"/>
        <file name="SubmitUNIPO_AIFallOut.bunit"/>
    </filelist>
</bunit>

```

- 最後に、bunit-report Ant タスクを使用してレポートを出力する必要があります。

```

<bunit-report reportType="HTML"
    destDir="/Users/User1/Downloads/bunit/html" />

```

B ユニットレポートタスクに使用可能なパラメータ (filePrefix、xsltPath、destDir、および reportType) の詳細については、上記の手順 4 を参照してください。

B ユニットファイルの編集

B ユニットエディタを使用して、独自の B ユニットテストを設計できます。プロジェクトエクスプローラで B ユニットファイルをダブルクリックして、B ユニットエディタを開きます。

B ユニットエディタには、[アウトライン] ペインと [詳細] ペインを含む [デザイン] タブが表示されます。[アウトライン] ペインには、[詳細] ペインで編集するために選択可能な B ユニットファイルのさまざまなセクションが表示されます。

B ユニットアウトラインの各ノードに関する説明を以下に示します。詳細については、ノード名を選択してください。

- [「BPEL ユニット \(ルート\)」 \(ページ 441\)](#)

テストで使用する BPEL、WSDL、スキーマ、およびその他のリソースを指定します。リソースは、プロセスサーバーエンジン用にデプロイされたプロセスとリソースのセットを作成するデータを提供します。B ユニットテストを開始すると、これらのリソースはすべて、標準のエンジンを起動するときと同じようにエンジンにデプロイされます。Informatica Business Process Developer の標準静的分析ルールは、B ユニットで識別された各 BPEL プロセスに対して実行することで、BPEL または環境にエラーがないことを確認します。

[トレース] オプションで、コンソールのログ出力を制御します。これはデフォルトで有効になっており、[B ユニット] ビューに適切なデバッグ情報が表示されない場合にアサーションの失敗を追跡する際に役立ちます。

- [「拡張機能と拡張アクティビティ」 \(ページ 441\)](#)

拡張機能で、WS-HT (ヒューマンタスク) の拡張機能である論理ユーザーグループを指定します。拡張アクティビティで、ユーザー拡張アクティビティ用の BPEL であるユーザーアクティビティを指定します。

- [「呼び出し」 \(ページ 442\)](#)

呼び出しアクティビティを指定します。

- [「アラーム」 \(ページ 443\)](#)

Pick およびイベントハンドラの Wait アクティビティおよび onAlarm ハンドラを指定します。アラーム要素と呼び出し要素には、テスト中に実行されることが想定されるプロセス内の各アラームと呼び出しに関する情報が含まれています。これらの要素は、アラームの実行に反応したり、プロセス内で呼び出したりするという意味では、リアクティブな要素であると考えする必要があります。こうした要素は、アラームまたは呼び出しを実行するのではなく、アラームまたは呼び出しが実行されたことをエンジンが通知するのを待ってから、プロセス内のそのアクティビティを B ユニット内の宣言と照合します。一致した場合、B ユニット宣言によってアサーション、シミュレートされた応答/フォールト/アラームが提供されます。

- [「コマンド」 \(ページ 444\)](#)

B ユニットテストエンジンにメッセージを送信してテストロジックを実行します。コマンド要素には、B ユニットテストの実行時にエンジンに送信されるすべてのコマンドが含まれています。これらの要素は、B ユニットがテストを開始するときに実行されるアクションであるという意味では、アクティブな要素であると考えする必要があります。各コマンドは順番に実行されます。コマンドに *asynch="true"* 属性がある場合、このコマンドは別のスレッドで実行されます。すべてのコマンドが完了すると、B ユニットテストは完了したと見なされます。最も一般的なコマンドは `sendMessage` コマンドで、これは、各 B ユニットに必要な唯一のコマンドです。

BPEL ユニット (ルート)

B ユニットエディタで B ユニットファイルを開き、[アウトライン] ビューから BPEL ユニットを選択します。

詳細ビューには、B ユニットテストで使用された BPEL、WSDL、およびその他のリソースが表示されます。

リソースを追加するには、名前、場所、およびタイプ URI を指定します。XQuery モジュールを使用している場合、タイプ URI は次のとおりです。

`http://modules.active-endpoints.com/2009/07/xquery`

パネルの下部には、[トレース] の選択があります。トレースをオンまたはオフにして、コンソールのログ出力を制御できます。

BPEL ユニットの子要素は次のとおりです。

- 拡張機能
- 拡張アクティビティ
- 呼び出し
- コマンド
- アラーム

子要素を追加するには、BPEL ユニートを右クリックして、[追加] を選択します。

[「B ユニットファイルの編集」 \(ページ 440\)](#) も参照してください。

拡張機能と拡張アクティビティ

B ユニットエディタで B ユニットファイルを開き、[アウトライン] ビューから **[拡張機能]** または **[拡張アクティビティ]** を選択します。拡張機能を追加するには、BPEL ユニートを右クリックします。

拡張機能

拡張機能により、B ユニットを実行するために必要なコンテキスト情報を指定します。これは通常、プロセスデプロイメント記述子で定義されます。現在、拡張機能は WS-HT 拡張機能である論理ユーザーグループを指定しているため、ユーザーアクティビティ内で使用できます。

論理ユーザーグループの値は、プロセスデプロイメント記述子エディタの [ユーザー] タブで定義したとおり に定義します (静的、動的、およびユーザーエラー値の詳細について)。

拡張アクティビティ

拡張アクティビティとは、ユーザーアクティビティを指します。ユーザーアクティビティに何が含まれるかを表明し、タスクが含まれる場合 (通知ではない場合) は、ユーザーアクティビティからの応答を指定することもできます。

B ユニットテストのユーザーアクティビティは、次のように定義できます。

1. 拡張アクティビティ（ユーザーアクティビティ）を選択または追加します。このアクティビティがグループ内にある場合（例えば、For Each など）、アクティビティが実行される回数を指定できます。
2. デフォルトの反復である反復を少なくとも 1 つ追加します。これは、それぞれの反復のデフォルトでの実行方法です。
3. 各反復ごとに、タスクの結果である応答タイプが追加されます。
 - **完了。** タスクから返される出力データを指定します。データはインラインでインポートまたは作成できます。デフォルトでは、データはインポートされるように設定されています。必要に応じて、インラインの設定を変更できます。
データがインポートされるように設定した場合、必要に応じて、タスクから返された XML データを変換するための独自の XSL ファイルを追加できます。これにより、データの一部の要素のみを変更しつつ、データ内の共通部分をさまざまな反復で再利用することができます。[「入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント」](#)（ページ 448）を参照してください。
必要に応じて、コンテキストデータ（タスクのイニシエータ、実際のタスクの所有者、およびタスクの他の割り当ての名前）を指定します。
 - **失敗。** 返すフォールトを選択します。
 - **期限切れ。** 有効期限が設定されている場合に、タスクが有効期限に達しました。
 - **スキップ。** タスクで [スキップ可能] オプションが選択されている場合に、タスクがスキップされました。
4. **アサートを追加する**
 - **Assert Match。** このアサーションのドキュメントであるラベル、アサーションのメッセージパートとクエリ、および一致するパターンを指定します。例として、B4P カスタム関数の 1 つから返されたデータを照合するといった場合が挙げられます。
 - **Assert Equals。** 返されるデータと一致する必要があるデータを指定します。
 - **タスクのアサート。** タスクの詳細（例えば、サブジェクト、プレゼンテーション、エスカレーションなど）をアサートします。

[「B ユニットファイルの編集」](#)（ページ 440）および [「入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント」](#)（ページ 448）を参照してください。

呼び出し

B ユニットエディタで B ユニットファイルを開き、[アウトライン] ビューから [呼び出し] を選択します。
[呼び出し] を追加するには、BPEL ユニットを右クリックします。

呼び出しの一般的な使用例としては、プロセスが呼び出しているサービスに対してシミュレートされた応答を提供することが挙げられます。プロセス内の Invoke アクティビティが実行されると、B ユニットエンジンは B ユニットファイル内の一致する呼び出し要素を探します。呼び出し要素が見つかった場合、この要素は、Invoke に渡された入力データに対してアサーションを実行する機能、およびサービスの Invoke からの応答をシミュレートするメッセージを提供します。

B ユニットの呼び出し要素は、一方向および双方向のサービス呼び出し用に設定することができます。どちらのスタイルの呼び出しであっても入力データにアサーションを設定できますが、応答が許可されるのは双方向のサービス呼び出しのみです。

B ユニットテストの呼び出しは次のように定義します。

1. [呼び出し] を選択または追加します。BPEL ユニット上でマウスを右クリックして、[呼び出し] を追加します。
2. 呼び出しの名前または **【場所のパス】** を指定します。このアクティビティがグループ内にある場合（例えば、For Each など）、アクティビティが実行される回数を指定できます。応答を返す前に遅延させる場合は、

遅延を指定します。この遅延により、onEvent イベントと onAlarm イベントをより適切に処理することができます。

3. デフォルトの反復を追加します。
4. 次のような想定される応答タイプを追加します。
 - **シミュレートされた応答**。出力データを指定します。
 - **サブプロセス**。呼び出しは BPEL プロセスであるため、サブプロセスとしてデプロイし、メインプロセスが終了および補償処理の対象となります。
 - **プロセス**。呼び出しは BPEL プロセスであるため、「プロセス」呼び出しハンドラとしてデプロイします。
 - **シミュレート済みフォールト**。呼び出しはフォールトを返します。
5. アサートを追加します。
 - **Assert Match**。このアサーションのドキュメントであるラベル、アサーションのメッセージパートとクエリ、および一致するパターンを指定します。
 - **Assert Equals**。呼び出しに渡される入力データを指定します。

メッセージパートのデータについて

次のような特定のアクションを実行すると、メッセージとメッセージに関連するパートに関する情報が B ユニットエディタで自動的に生成されます。

- メッセージの送信のプロセス名、パートナーリンク、および操作の値を入力します。
- 呼び出しアサーション（または拡張アサーション）に対して [シミュレート済み応答] を選択します。
- シミュレート済みフォールトのフォールト名を選択します。

[「B ユニットファイルの編集」 \(ページ 440\)](#)、[「アサーションの使用に関するヒント」 \(ページ 447\)](#)、および [「入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント」 \(ページ 448\)](#) も参照してください。

アラーム

[B ユニット] エディタで [B ユニット] ファイルを開き、[アウトライン] ビューから [アラーム] を選択します。アラームを追加するには、BPEL ユニットの右クリックします。

アラームは、Pick とイベントハンドラの Wait アクティビティおよび onAlarm イベントを参照します。アラームの期限または期間にアサートできます。

B ユニットテストのアラームは次のように定義できます。

1. アラームを選択または追加します。BPEL ユニットの右クリックしてアラームを追加します。
2. アラームベースのアクティビティのプロセスパスとロケーションパスを指定します。
 - アラームのシミュレート済み期間を追加します。これは、アラームがトリガされるまで B ユニットテストが待機する時間です。
 - このアクティビティがループ内にある場合（例えば、For Each など）、アクティビティが実行される回数を指定できます。
3. デフォルトのアラームを追加し、B ユニットテストで使用する期間を追加します。
4. アサートを追加します。
 - アサート期限。B ユニットテストの実際の期限を期間に変換して指定します。期限が発生する期間である、精度値を指定します。この値は秒単位で指定します。
 - アサート期間。期限と同様です。

5. 必要に応じて、アサートの反復を追加します。アラームベースのアクティビティがグループ内にある場合は、この反復にインデックスを追加します。インデックスは、アラームに指定された、カウント内の n 番目の反復となります。

[「B ユニットファイルの編集」 \(ページ 440\)](#)および[「B ユニットファイルの編集」 \(ページ 440\)](#)[「アサーションの使用に関するヒント」 \(ページ 447\)](#)を参照してください。

コマンド

B ユニットエディタで B ユニットファイルを開き、[\[アウトライン\]](#) ビューから [\[コマンド\]](#) を選択します。コマンドを追加するには、BPEL ユニットの右クリックします。

コマンドは、B ユニットテストエンジンにメッセージを送信して、テストロジックを実行します。

追加できるコマンドは次のとおりです。

コマンド	説明
メッセージの送信	WSDL メッセージをテストエンジンに送信します。 指定したパートナーリンクの操作を選択します。 配信オプション。 <ul style="list-style-type: none">- 非同期。最初の受信に関連付けられた応答が完了するのを待たずに、2 つの受信が連続して実行されるようにするかどうかを制御します- 遅延。タイムアウトのチェックを開始するまでに B ユニットが待機する待機値。- タイムアウト。テストエンジンがプロセスの完了を待機する時間を設定します。この設定により、Invoke を適切な回数で完了させることができます。- メッセージコンテキスト。必要に応じて、認証されたプリンシパルを含む受信メッセージのメタデータ、およびその他のメッセージヘッダーデータを指定します。
待機期間	プロセスの実行が完了したかどうかを確認します。 <ul style="list-style-type: none">- 遅延。タイムアウトのチェックを開始するまでに B ユニットが待機する待機値。- タイムアウト。テストエンジンがプロセスの完了を待機する時間を設定します。この設定により、Invoke を適切な回数で完了させることができます。- タイムアウト時に失敗。プロセスがタイムアウト値の範囲内で完了しない場合にテストが失敗するようにするには、このチェックボックスを有効にします。
プロセスの終了	複数のプロセスを実行している場合は、B ユニットテストのプロセスの 1 つのプロセスインスタンスを終了します。
アサートプロセス状態	指定したプロセスの実行状態をアサートします。 <ul style="list-style-type: none">- プロセス状態。実行中、一時停止、完了、フォールト、最終。完了状態とフォールト状態をアサートするには、[最終] を選択します。- プロセス ID。複数の BPEL ファイルでテストを実行している場合にのみ使用します。- 遅延。タイムアウトのチェックを開始するまでに B ユニットが待機する待機値。- 非同期待機。同じプロセスの [メッセージの送信] コマンドで [非同期] を選択している場合、このプロパティを使用すると、すべての非同期待機アクティビティが完了するまでプロセス状態のアサートを待機できるようになります。

[「B ユニットファイルの編集」 \(ページ 440\)](#)も参照してください。

B ユニットテストのデバッグ

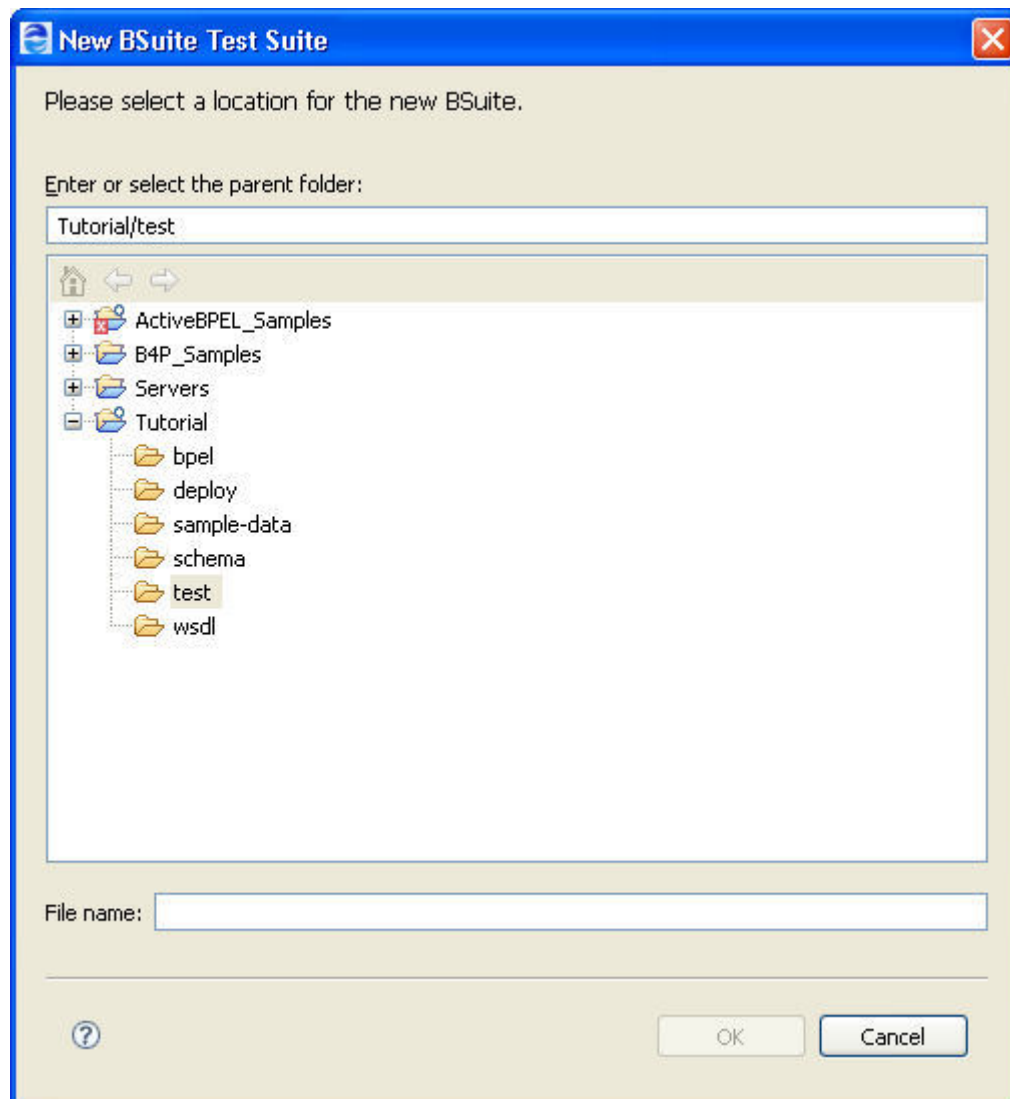
B ユニットテストが失敗した場合は、[B ユニット] ビューでテスト名を右クリックして、**[デバッグ]** を選択できます。[デバッグ] ビューが開き、テストが実行されます。コンソールには、テストからのログ出力が表示されます。

[デバッグ] モードでは、リモートデバッグと同じように、BPEL ファイルにブレークポイントを設定してデバッグできます。

BPEL ユニットテストスイートの作成と実行

プロジェクトエクスプローラで複数の B ユニットファイルを選択して、一連のテストを行うことができます。

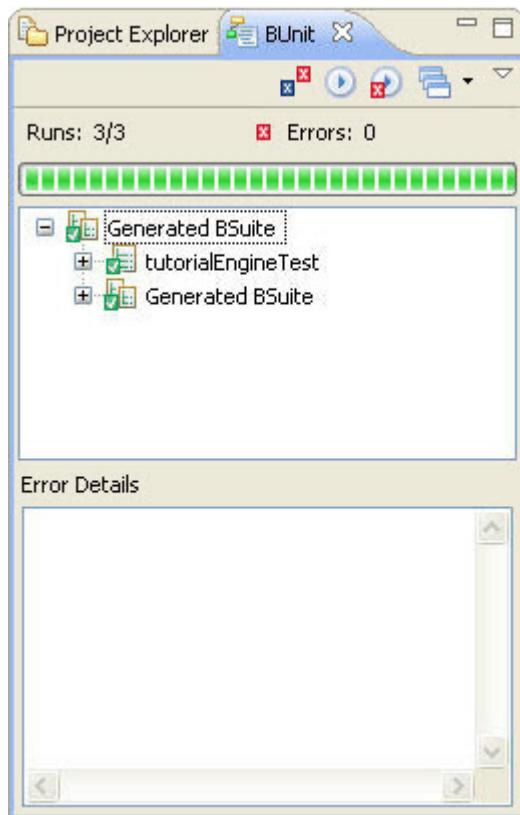
例えば、Orchestration プロジェクト内のテストフォルダから 2 つ以上の .bunit ファイルを選択します。マウスを右クリックして、**[新規] > [BSuite テストスイート]** を選択します。次のように、BSuite ダイアログが開きます。



BSuite の名前を入力します。.bsuite という拡張子が自動的に追加されます。

スイート内のすべてのテストが実行されます。

次に示すように、複数の BSuite を BSuite に追加することができます。



アサーションの使用に関するヒント

B ユニットテストフレームワークの重要なコンポーネントは、アサーションを実行する機能です。アサーションによって、実行時のコード単位の状態について、想定される正確性を記述します。アサーションには次のものが含まれます。

- Invoke アクティビティの入力データ
- B ユニットのメッセージ送信コマンドを介してエンジンに送信された同期メッセージからの応答
- アラームの期限と期間の値

アサーションには想定されるデータが含まれ、プロセス実行からのデータに対して実行されます。アサーションに必要なデータは、テキストノード、要素、またはインポートされた要素です。アサーションは、メッセージデータ全体に対して実行することができます。また、XPath クエリを使用してメッセージデータの一部を選択することもできます。

B ユニットエンジンは、Receive/Reply/Invoke のメッセージを検証します。そのため、メッセージ全体に対してアサーションを実行する必要はありません。代わりに、プロセス内のロジックをテストするメッセージの部分に焦点を絞ったアサーションを作成できます。このタイプの集中アサーションにより、ユニットテストが読みやすくなり、メッセージ定義が変更された場合に必要となる作業量が削減されます。例えば、`rm:ProductId` の値が 100 であることをアサートする場合、次のようなクエリを使用することで最も高い柔軟性が得られます。

```
<abu:assertEquals part="p"
  query="//rm:ProductId/text()">100</abu:assertEquals>
```

この例では、クエリでの子孫軸の使用に注意してください。これにより、メッセージ内の任意の場所で `ProductId` 要素が選択されます。こうした特異性の欠如は必ずしも良い側面のみを持つわけではありませんが、B ユニットエンジンによってすでにメッセージの構造が検証されているため、要素の位置が間違っていた場合にはスキーマ検証で報告されることに注意してください。

アサーションの適切な使用例の一部を次に示します。

- プロセス内の 1 つ以上の複雑なデータマッピングによって入力生成された場合に、呼び出しの入力をアサートする
- 遷移条件または if/else アクティビティの形式でプロセスに条件付きロジックがある場合に、アサーションによって、プロセス内の特定のパスでステップが実行されたことを証明する

アサートする価値のあるメッセージがごく一部しかない場合、スキーマ検証や大きなリテラルアサーションで得られる利点を考慮すると、アサーションを行う価値がない場合もあります。

入力およびアサートデータに対するパラメータ化された XSL の使用に関するヒント

複数のテスト間で同じデータにわずかな違いがあることをアサートする場合は、B ユニットテストに効率を組み込むことができます。効率は、パラメータ化された XSL 形式で指定します。

次の例を検討します。

XML データファイルには、次のような要素があります。

```
<ns:status>CREATED</ns:status>
```

それぞれが異なったステータス値を持つ別の XML ファイルを作成する代わりに、次のような XSL ファイルを作成します。

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ns="http://schemas.active-endpoints.com/b4p
    /wshumantask/2007/10/aeb4p-task-rt.xsd">
  <xsl:import href="common.xsl"/>
  <xsl:param name="state"/>
  <xsl:template match="ns:status/text()" priority="2">
    <xsl:value-of select="$state" />
  </xsl:template>
</xsl:stylesheet>
```

次に、B ユニットファイルの [ソース] ビューで、次のようなパラメータを追加できます。

```
<abu:part href="../../data/getTaskInstanceResponse-minimal.xml"
  name="taskInstance"
  xsl="../../xsl/change-task-instance-state.xsl">
  <abu:params>
    <abu:param name="state" value="COMPLETED"/>
  </abu:params>
```

B ユニットテストごとに、同じインポートデータと同じ XSL ファイルを使用します。別の結果を得るには、パラメータ値を変更します。

パートナーリンクデータの指定に関するヒント

一部のプロセスでは、テストの一部としてパートナーリンクデータの設定が必要です。パートナーリンクは、partnerRole の wsdl:EndpointReference 情報、または myRole のサービス名とともに指定できます。これは、計算時にプロセスがパートナーリンク内のデータを使用する場合にのみ必要です。

パートナーリンクデータを B ユニットテストに追加する手順

1. [アウトライン] ビューで、[パートナーリンク] ノードを選択してから、パートナーリンクを選択します。
2. [プロパティ] ビューで、[すべて] タブを選択してから、[シミュレーション] カテゴリを選択します。
3. 適切なパートナーリンク、マイロール、またはパートナーロールを選択します。
4. 行の最後にある [ダイアログ] ボタンを選択して、[パートナーリンクデータ] ダイアログを開きます。

5. エンドポイント参照の定義を入力します。
 6. プロセスをシミュレートし、シミュレーションを B ユニットテストとして保存します。
- 次の例は、テストに追加する B ユニット要素を示しています。

```
<!-- Controls deployment of partner links to the test engine. -->
<abu:partnerLinks>
  <abu:partnerLink xmlns:ns="http://docs.active-
    endpoints.com/sample
    /bpel/loanprocess/2008/02/loanProcessCompleted.bpel"
    name="LoanApproval"
    processName="ns:loanProcessCompleted">
    <abu:partnerRole endpointReference="static">
      <wsa:EndpointReference
        xmlns:s="http://docs.active-endpoints.com/
        sample/wSDL/riskAssessment/2008/
        02/riskAssessment.wSDL"
        xmlns:wsa="http://www.w3.org/2005
        /08/addressing">
        <wsa:Address>urn:x-vos:loancompany:
        RiskAssessmentServiceNEW
        </wsa:Address>
        <wsa:ServiceName
          PortName="RiskAssessmentServicePort">
            s:RiskAssessmentServiceNEW
          </wsa:ServiceName>
        </wsa:EndpointReference>
      </abu:partnerRole>
    </abu:partnerLink>
  </abu:partnerLinks>
```

B ユニットファイルの例

B ユニットファイルのほとんどのセクションを示す B ユニットソースのテンプレートを以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<abu:bpelUnit xmlns:abu="http://schemas.active-endpoints.com
/activebpelunit/2008/10/activebpelunit.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <abu:trace>on</abu:trace>
  <abu:engineProperties>
    <abu:documentation>Controls engine configuration
      options, including process logging.
    </abu:documentation>
    <abu:engineProperty path="Logging"
      value="urn:ae:execution"/>
  </abu:engineProperties>
  <abu:bpels>
    <abu:documentation>Controls deployment of the BPEL
      process to the test engine.</abu:documentation>
    <abu:bpel location=" ../bpel
      /loanProcessHumanCompleted.bpel"/>
  </abu:bpels>
  <abu:wsdls>
    <abu:documentation>Controls deployment of the WSDL files
      to the test engine.</abu:documentation>
    <abu:wSDL location=" ../wSDL/loanProcess.wSDL"/>
  </abu:wsdls>
  <abu:schemas>
    <abu:documentation>Controls deployment of the XML Schema
      files to the test engine.</abu:documentation>
    <abu:schema location=" ../schema/loanRequest.xsd"/>
  </abu:schemas>
  <abu:resources>
    <abu:documentation>Controls deployment of resource
```

```

        mappings to the test engine.</abu:documentation>
<abu:resource location="../xsl-renderings/
  loan_taskdetail.xml" locationHint=
    "project:/humantaskcompleted/xsl-renderings/
    loan_taskdetail.xml"
    type="http://www.w3.org/1999/XSL/Transform"/>
<abu:resource location="../xsl-renderings/
  loan_param2commands.xml" locationHint=
    "project:/humantaskcompleted/xsl-renderings/
    loan_param2commands.xml"
    type="http://www.w3.org/1999/XSL/Transform"/>
</abu:resources>
<abu:invokes>
  <abu:documentation>Defines the simulated output and
    assertions to perform on invoke activities
  </abu:documentation>
  <abu:invoke count="1" name="NotifyCRMOfExpiration">
    <abu:invokeIteration index="0"/>
  </abu:invoke>
</abu:invokes>
<abu:commands>
  <abu:documentation>Sends messages to the B-unit
    test engine
    to execute the test logic.</abu:documentation>
  <abu:sendMessage async="true"
    operation="request" partnerLink="LoanProcess">
</abu:sendMessage>
  <!-- Tells the test to wait before making final assertions -->
  <abu:waitUntil xmlns:
    ns="http://docs.active-endpoints.com/sample/bpel
    /loanprocess/2008/02/loanProcessHumanCompleted.bpel"
    processName="ns:loanProcessHumanCompleted"
    timeout="5000"/>
</abu:commands>
</abu:bpelUnit>

```

第 30 章

プロセスデプロイメント

デプロイメントとは、実行可能なプロセスサーバーに BPEL プロセスをパブリッシュする操作です。

Process Developer で、プロセスサーバーに必要な BPEL プロセスおよび関連リソースを作成します。次に、プロセスデプロイメント記述子を作成します。デプロイメント記述子と関連するすべてのリソースをビジネスプロセスアーカイブにパッケージ化し、サーバーにエクスポートします。

プロセスが実行されると、プロセスサーバーによって、正しいパートナーサービスおよびメッセージがプロセスと相互にやり取りすることが保証されます。

プロセスサーバーは、WS Addressing プロトコルを使用して、プロセスの通信先となるパートナーエンドポイントを識別します。

次のトピックでは、プロセスのデプロイについて説明します。

- [「デプロイメントの準備」 \(ページ 451\)](#)
- [「エンドポイント参照のアドレス指定の考慮事項」 \(ページ 452\)](#)
- [「プロセスデプロイメントの手順の概要」 \(ページ 455\)](#)
- [「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#)
- [「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#)
- [「デプロイメントコントリビューションの管理」 \(ページ 492\)](#)
- [「BPRD スクリプトを使用した BPR コントリビューションの再生成およびデプロイ」 \(ページ 493\)](#)
- [「BPEL プロセスをインスタンス化する方法」 \(ページ 496\)](#)
- [「プロセスのバージョンングについて」 \(ページ 497\)](#)

デプロイメントの準備

プロセスサーバーへのデプロイメント用に BPEL プロセスを準備するには、次の 2 つの作業が必要です。

- [「デプロイメント用の BPEL ファイルの準備」 \(ページ 452\)](#)
- [「デプロイされたプロセス用のサーバープラットフォームの選択」 \(ページ 452\)](#)

さらに、プロセスデプロイメント記述子ファイルを作成する前に、[「エンドポイント参照と WS-Addressing の考慮事項」 \(ページ 452\)](#)を読んでおくことをお勧めします。このトピックでは、必須のデプロイメント情報を入力するために必要となる詳細について説明します。

パートナーのサービスで認証済みアクセスが必要な場合は、[「アクセス時に資格情報が必要なエンドポイント参照」 \(ページ 454\)](#)を参照してください。

デプロイメント用の BPEL ファイルの準備

〔問題〕 ビューにエラーが表示されておらず、シミュレートされたプロセスの実行が正常に終了すると、BPEL ファイルをプロセスサーバーにデプロイする準備が整ったことになります。

デプロイメント用にプロセスの準備が整っていることを確認するには、次の確認項目をチェックします。

- パートナーリンク、操作、Receive (Pick とイベントハンドラ)、Invoke、Reply の変数など、各アクティビティに必要なすべてのプロパティが入力されていることを確認します。
- 1 つ以上の Receive または Pick アクティビティの [インスタンスの作成] プロパティが [はい] に設定されていることを確認します。
- プロセスの [抽象プロセス] プロパティが [いいえ] に設定されていることを確認します。
- [第 28 章, 「シミュレーションとデバッグ」 \(ページ 409\)](#) に記載されているように、シミュレートされたプロセスの実行をステップスルーします。

デプロイされたプロセス用のサーバープラットフォームの選択

BPEL プロセスは、任意の BPEL サーバーで実行できます。ただし、BPEL プロセスをプロセスサーバーにデプロイするには、[「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#) に記載されているように、デプロイ用のファイルのパッケージ (.bpel、.wsdl、.pdd) を作成します。

Process Developer では、インストールフォルダにプロセスサーバーが含まれています。プロセスを設計およびテストする場合に、この開発サーバーにプロセスをデプロイすることができます。このサーバーにデプロイする手順については、[「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#) に記載されています。

エンドポイント参照のアドレス指定の考慮事項

プロセスサーバーは、BPEL プロセスで呼び出された Web サービスに到達する方法、タイミング、および場所を指定するためのさまざまなアドレス指定オプションをサポートしています。プロセスデプロイメント記述子 (PDD) ファイルの作成中に、アドレスに関する各種の詳細を追加できます。

エンドポイント参照と WS-Addressing の考慮事項

デプロイ時に、各パートナーリンクのエンドポイント参照の場所に関する情報を指定する必要があります。また、「コピー操作の動的エンドポイント参照の例」に詳細が記載されているように、エンドポイント参照を動的に割り当てることができます。

エンドポイント参照の場所に関する情報を指定するための主要なプロトコルの 1 つは、Web Services Addressing (WS-Addressing) 仕様に基づいています。

WS-Addressing 仕様は、Web サービスを参照するための形式を標準化し、HTTP などの特定のトランスポートメカニズムにバインドされていない一意に識別された Web サービスおよび Web サービスインスタンスを作成できるようにします。WS-Addressing を使用すると、SOAP 本文で指定された URL ではなく、SOAP 要求のヘッダーにエンドポイント情報が追加されます。

SOAP ヘッダーにエンドポイント参照情報があるということは、次のことができることを意味します。

- Web サービスインスタンス ID を指定する
- *Reply To* アドレスおよび *Fault To* アドレスを選択する
- 確認応答を送信者に送り返す必要がある場合の *From* エンドポイント参照を選択する

プロセスサーバーでは、SOAP over HTTP 以外の他のメカニズムを指定できることに注意してください。詳細については、「[カスタムサービスのインタラクション](#)」を参照してください。

基本的な WS-Addressing 構文は次のとおりです。

```
<wsa:EndpointReference>
  <wsa:Address>anyURI</wsa:Address>
  <wsa:ServiceName PortName="portname">Service QName
</wsa:ServiceName>
</wsa:EndpointReference>
```

ここで、

- Address はエンドポイントを識別する必須要素です。これは、ネットワークアドレスまたは論理アドレスにすることができます。
- ServiceName は、呼び出されるサービスの修飾名または QName です。このサービスは、プロセスとともにデプロイされる WSDL ファイル、またはプロセスサーバーのリソースカタログで利用可能な WSDL ファイルで定義する必要があります。
- PortName は、呼び出されているサービスポートを識別します。SOAP 1.1 または SOAP 1.2 のポートバインディングを指定できることに注意してください。

「[アクセスに資格情報を必要とするエンドポイント参照](#)」に記載されているように、資格情報の詳細をエンドポイント参照に追加できます。

注: Informatica Cloud にデプロイする際の設定とエンドポイント参照については、「」を参照してください。

次のバージョンの WS-Addressing 仕様がサポートされています。

- <http://www.w3.org/2005/08/addressing>
これは、PDD で使用するデフォルトバージョンです。デフォルトの変更については、「*Process Developer* の設定」を参照してください。
- <http://schemas.xmlsoap.org/ws/2004/03/addressing>
- <http://schemas.xmlsoap.org/ws/2004/08/addressing>
- <http://schemas.xmlsoap.org/ws/2003/03/addressing>

WS-Addressing の仕様は次の URL で確認できます: <http://www.w3.org/2005/08/addressing/>。

エンドポイント参照と Informatica Cloud

注: WSA アドレス指定の一般的な情報については、「[エンドポイント参照と WS-Addressing の考慮事項](#)」(ページ 452)を参照してください。

Informatica Cloud にプロセスをデプロイし、エージェントを使用してプロセスにアクセスする場合は、PDD の [パートナーリンク] タブで次のようにフィールドを設定する必要があります。

- 呼び出しハンドラ: [プロセス] に設定します。
- エンドポイントタイプ: [静的] に設定します。

- エンドポイント参照: エージェントを指すように EndpointReference 要素を設定します。次に例を示します。

このアドレスの構造は `agent:agent_name:agent_service` です。
agent

"agent"という語。これにより、Process Developer はどのような情報が後に続くのかを把握します。

agent_name

エージェントの名前。通常、これはエージェントが実行されるシステムです。

agent_service

インストールされているサービスの名前。

アクセス時に資格情報が必要なエンドポイント参照

デプロイ時に、各パートナーリンクのエンドポイント参照の場所に関する情報を指定する必要があります。パートナーのサービスでアクセス時に認証が必要な場合は、プロセスデプロイメント記述子ファイルのパートナ

ーリンク定義に資格情報の詳細を追加できます。詳細については、「[ポリシーアサーションの追加](#)」(ページ 468)を参照してください。

エンドポイント参照の置換可能な URN と URL の指定

PDD ファイルで、静的エンドポイント参照の論理アドレスまたは物理アドレスを指定できます。論理アドレス (URN) を指定した場合は、プロセスコンソールの「URN マッピング」ページで URN を物理アドレスにマッピングできます。静的エンドポイントアドレスの URL を指定した場合は、サーバー上の別の URL にマッピングすることで URL を置き換えることができます。

例えば、PDD エディタでは、次のアドレスを指定できます。

```
<wsa:Address>urn:localhost:AssessRisk</wsa:Address>
```

プロセスコンソールでプロセスをデプロイした後で、次のように URN を URL にマッピングできます。

```
urn:localhost:AssessRisk = http://localhost:8080/active-bpel/services/AssessRisk
```

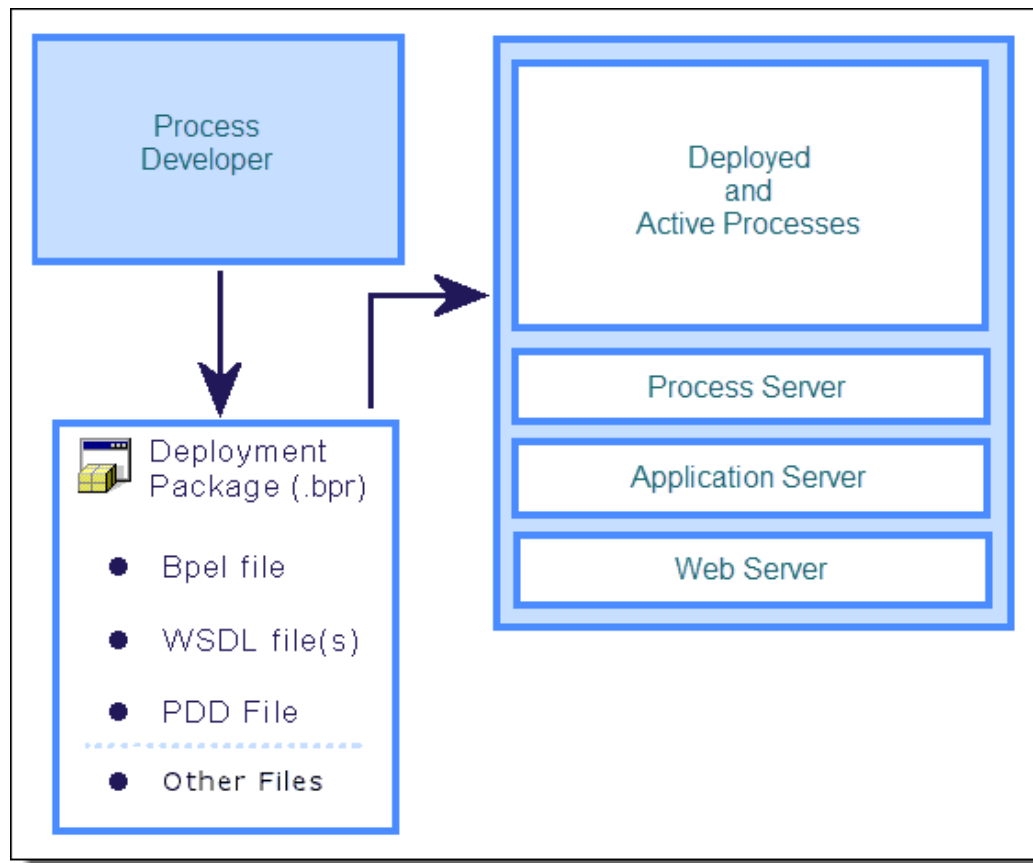
URN 構文のその他の例については、「プロセスコンソールのヘルプ」の「URN のマッピング」トピックを参照してください。

エンドポイント参照と WS-Policy

プロセスサーバーは、Web サービスポリシーフレームワーク (WS-Policy) をサポートしています。ポリシーによって、さまざまなサービス要件、設定、および機能を記述することができます。プロセスデプロイメント記述子ファイルのエンドポイント参照セクションにポリシー情報を追加できます。詳細については、「[ポリシーアサーションの追加](#)」を参照してください。

プロセスデプロイメントの手順の概要

次の図は、BPEL プロセスを設計からデプロイメントに移す方法を示しています。



1. Process Developer で、必要に応じて WSDL またはサービス参照をプロジェクトにインポートし、BPEL ファイルおよびプロセスデプロイメント記述子 (.pdd) ファイルを作成します。詳細については「[プロセスデプロイメント記述子ファイルの作成](#)」(ページ 456)を参照してください。
2. Process Developer で、プロセスデプロイメントパッケージ (.bpr) ファイルを作成してエクスポートします。詳細については「[ビジネスプロセスアーカイブコントリビューションの作成とデプロイ](#)」(ページ 483)を参照してください。

プロセスをデプロイすると、最初の Receive アクティビティ（または Pick あるいはイベント）でメッセージが受信された場合にアクティブになります。

プロセスデプロイメント記述子ファイルの作成

プロセスデプロイメント記述子 (.pdd) ファイルによって、プロセスサーバー環境でプロセスを実行するために必要な情報を記述します。プロセスをプロダクションサーバーにデプロイする場合は、プロセスのバージョンやその他の詳細を指定して、サーバーのデフォルト値を上書きできます。

プロセスデプロイメント記述子 (.ppd) ファイルによって、プロセスサーバーでプロセスを実行するために必要な情報を記述します。この情報には、パートナーリンクの詳細、永続性とバージョン管理の情報、プロセスディレクティブ、インデックス付きプロパティ、およびその他の詳細が含まれます。以下の手順の各ステップには、デプロイメントへの詳細の追加に関連するヘルプトピックへのリンクがあります。

プロセスデプロイメント記述子ファイルを作成する手順

1. Process Developer で、[ファイル] > [新規] > [デプロイメント記述子] を選択します。

2. デフォルトを変更する場合は、フォルダとプロセスデプロイメント記述子のファイル名を選択します。
3. 記述する BPEL ファイルを選択し、[完了] を選択して PDD エディタを開きます。
4. [全般オプション] を選択します。詳細については、「[全般的なデプロイメントオプション](#)」を参照してください。
5. 次のように、各パートナーリンクの [パートナーリンク] タブに入力します。
 - a. リストからパートナーリンクを選択します。
 - b. パートナーリンクにパートナーロールが定義されている場合は、パートナーリンクのエンドポイントタイプを選択し、必要に応じて呼び出しハンドラを選択します。エンドポイントタイプの説明と例については、「[パートナーロールのエンドポイントタイプ](#)」を参照してください。呼び出しハンドラの説明については、「[パートナーロールの呼び出しハンドラ](#)」を参照してください。WS-Policy の詳細については、「[エンドポイント参照と WS-Policy](#)」を参照してください。
 - c. エンドポイントタイプが [静的] の場合は、以下の注を参照してください。
 - d. パートナーリンクにマイロールが定義されている場合は、バインディングスタイルとサービス名を選択します。必要に応じて、許可されたロールを指定します。これらのプロパティの説明については、「[マイロールバインディング、サービス名、および許可されるロールオプション](#)」を参照してください。WS ポリシーアサーションをマイロールパートナーリンクに追加するには、「[ポリシーアサーションの追加](#)」を参照してください。
6. プロセスサーバーの場合、必要に応じてインデックス付きプロパティを定義します。インデックス付きプロパティは、プロセスの実行中に変数（または変数パート）の値が入力される変数のプロパティです。プロセスコンソールでは、プロセスインスタンスの検索フィルタとしてインデックス付きプロパティを選択できます。さらに、イベントの追跡にインデックス付きプロパティを使用できます。詳細については、「[インデックス付きプロパティの追加](#)」を参照してください。

ヒューマンタスクの論理ユーザーグループの追加に関する詳細については、このヘルプの他の場所にある「[ヒューマンタスク](#)」を参照してください。

静的エンドポイントタイプに関する注意:

- 柔軟性を高めるために、URL ではなく URN としてアドレスを指定し、呼び出しハンドラに WSA アドレスを指定します。詳細については、「[エンドポイント参照の置換可能な URN と URL の指定](#)」を参照してください。
- 次の例に示すように、WSDL にバインディングが含まれていない場合、WS Addressing 要素はプレースホルダとともに追加されます。[パートナーリンク] タブでプレースホルダを編集できます。詳細については、「[PDD エディタの \[ソース\] ビューの使用](#)」を参照してください。

```
Endpoint Reference:
<wsa:EndpointReference xmlns:s='FILL_IN_NAMESPACE'>
  <wsa:Address>s:FILL_IN_ADDRESS_URI</wsa:Address>
  <wsa:ServiceName PortName='FILL_IN_PORT_NAME'>s:FILL_IN_SERV
</wsa:EndpointReference>
```

- 別のサービスを選択する場合は、サービスを選択します。詳細については、「[デプロイメント記述子パートナーリンク用のサービスの選択](#)」を参照してください。
- WS-Policy ポリシーアサーションを追加する方法については、「[ポリシーアサーションの追加](#)」を参照してください。

全般的なデプロイメントオプション

プロセスをプロダクションサーバーにデプロイする場合は、プロセスのバージョンやその他の詳細を指定して、サーバーのデフォルト値を上書きできます。

プロセスデプロイメント記述子ファイルの作成中に、新しいバージョンと実行中のバージョンのデプロイメントオプションを選択できます。設定を変更しない場合、以下に記載されているようにサーバーはデフォルトを使用します。

プロセスのロギング

デフォルトでは、プロセスサーバーは実行中のプロセスに関する実行ログを生成します。この設定を有効にしてエンジンのデフォルトを上書きしてから、実行中または完了したプロセスの実行ログを表示またはダウンロードできます。実行ログは、アクティビティ実行の開始/終了時間やその他の詳細を提供し、フォールトが発生したプロセスのトラブルシューティングする場合に役立ちます。ロギングレベルは次のとおりです。

- システムのデフォルト。プロセスログの上書きなしですべてのプロセスに適用される、プロセスコンソールで有効な設定。サーバーはデフォルトで「実行」に設定されています。
- なし。プロセスの実行時にロギングは発生しません。ロギングを無効にすると、パフォーマンスが向上します。
- 完全。デッドパスアクティビティの *Will Not Execute* ステートメントを含む、すべての実行ステートメントがログに記録されます。例えば、実行されないすべてのフォールト処理ステートメントがログに記録されます。
- 実行（デフォルト）。*Will Not Execute* ステートメントを除くすべての実行ステートメントがログに記録されます。「完全」の代わりにこの設定を使用すると、ログファイルのサイズを大幅に減らすことができます。
- データを使用した実行。変数、式、およびパートナーリンクデータを含んだ *Will Not Execute* ステートメントを除く、すべての実行ステートメントがログに記録されます。この設定を使用すると、ログファイルのサイズが大幅に増加する可能性があります。

プロセスの永続性

永続性とは、サーバーにデプロイされたアクティブなプロセスのストレージを指します。プロセスサーバーでプロセスが実行されると、すべての状態データと変数データがデフォルトでサーバーデータベースに保存されます。ただし、パフォーマンス上の理由およびデータベースサイズの考慮事項をふまえて、より低いレベルの永続性を設定することをお勧めします。

プロセスで待機アクティビティまたは再試行ポリシーを使用する場合は、「永続」レベルまたは「完全」レベルを選択する必要があります。

永続性設定の選択は、次のように行うことができます。PDD を保存すると、選択した内容が検証されます。

設定	説明	制限
完全	デフォルト設定。プロセスインスタンスごとに、実行中のプロセス、フォールトが発生したプロセス、完了したプロセスのすべての状態情報が保存されます。サーバーエラーが発生した場合、実行中のプロセスは完全にリカバリされます。プロセスサーバーには、この設定のジャーナリング（データベースを対象とした変更のジャーナル）が保持されているため、リカバリが可能です。	プロセスがパートナーロールに WS-RM 呼び出しハンドラを使用している場合、またはマイロールに WS-Reliable メッセージングのポリシーアサーションを使用している場合は、この設定を選択する必要があります。詳細については、「 パートナーロールの呼び出しハンドラ 」（ページ 461）および「 WS-Reliable メッセージング 」（ページ 481）を参照してください。
持続	[完全] と同じストレージ設定ですが、ジャーナリングはありません。実行中のプロセスが中断されます。システムがダウンした場合、プロセスはリカバリ可能ですが、ジャーナリングが行われておらず、調査が必要であるため、リカバリの際にこのタイプは中断としてマークされます。これは、キャッチされないフォールト時に中断をサポートするための最小の永続性レベルです（「 Suspend アクティビティによる、プログラムでのプロセスの一時停止 」（ページ 399）を参照）。	プロセスが WS-Reliable メッセージングを使用している場合、この設定を選択することはできません。 テナントの詳細で設定したアクティビティ実行制限に達して永続性がこの最小レベルに設定されない場合、[キャッチされないフォールト時に中断] フラグは無視され、プロセスは終了します。
最終	プロセスの最終状態（完了またはフォールト）とプロセス変数のみを格納します。サーバーエラーが発生すると、実行中のプロセスが終了します。この設定を使用すると上記の設定よりもデータベースへの書き込み頻度が少なくなりますが、プロセス変数の実行パスと最終値が確認可能な、プロセスコンソールの「アクティブなプロセス」の詳細ページでプロセスのグラフを表示できます。プロセスはメモリ内でのみ実行され、[プロセスアイドルタイムアウト] というサーバープロパティがこの永続レベルに影響を及ぼすことはありません。	[最終]、[簡易]、および [なし] の場合、プロセスに次の構成を含めることはできません。 <ul style="list-style-type: none"> - Wait アクティビティ - 複数の Receive アクティビティ - ユーザーアクティビティ（オンプレミスのみ） - プロセスイベントハンドラ - プロセス:サブプロセス呼び出しハンドラ - WS-RM 呼び出し
簡易	これはプロセスロギングの最小レベルです（上記のセクションで説明）が、アクティブなプロセスのグラフを表示することはできません。開始時刻と完了時刻、および最終状態（完了またはフォールト）のみを格納します。プロセスにフォールトが発生した場合にのみ、状態変数とプロセス変数を格納します。プロセスはメモリ内でのみ実行され、[プロセスアイドルタイムアウト] というサーバープロパティがこの永続レベルに影響を及ぼすことはありません。	[最終] を参照してください。
なし	プロセス情報は、プロセスの終了時にサーバーデータベースに保存されません。プロセスインスタンスは、プロセスコンソールの「アクティブなプロセス」ページに表示されません。	[最終] を参照してください。

グループ

グループ名は 1 つ以上のプロセスに割り当てることができます。グループ名は、プロセスコンソールでデプロイしたプロセスまたはアクティブなプロセスのリストを表示するための選択フィルタです。

キャッチされないフォールト時に中断

WS-BPEL 2.0 仕様に従って、キャッチされないフォールトがあるプロセスは終了します。ただし、必要に応じて、キャッチされないフォールトでこのプロセスを中断して、プロセス例外管理を実行できます。

次のように選択できます。

システムのデフォルト	すべてのプロセスの現在のエンジン設定。デフォルトのエンジン設定では、キャッチされないフォールト時の中断が無効です。
False	キャッチされないフォールトでこのプロセスを中断させません。プロセスは異常終了します。この設定によって、エンジン設定が上書きされます。
True	キャッチされないフォールトでこのプロセスを中断して、中断されたフォールト状態にします。その後、フォールトが発生したプロセスに、プロセス例外管理（フォールト状態のアクティビティやスコープの再試行や完了など）を実行できます。この設定によって、エンジン設定が上書きされます。詳細については、 第 26 章、「プロセス例外管理」 （ ページ 398 ）を参照してください。

リカバリの呼び出し時に中断

ノードエラーが発生して完了しない呼び出しアクティビティの場合、リカバリ時にプロセスを中断できます。プロセスは保留中の呼び出しで中断され、必要に応じて、プロセス例外管理を実行できます。

次のように選択できます。

システムのデフォルト	すべてのプロセスの現在のエンジン設定。デフォルトのエンジン設定では、保留中の呼び出しがある場合のプロセスのリカバリ復時の中断が無効です。
False	プロセスのリカバリ後の保留中の呼び出しでこのプロセスを中断しません。プロセスは異常終了します。
True	プロセスを中断状態にする保留中の呼び出しがある場合は、リカバリ時にこのプロセスが中断されます。その後、プロセスでプロセス例外管理を実行してから、呼び出しを再試行または完了することができます。この設定によって、エンジン設定が上書きされます。

プロセスインスタンスの保持期間

プロセスをプロセスサーバーにデプロイし、実行を開始すると、完了したプロセスインスタンスとフォールトが発生したプロセスインスタンスが Informatica Business Process Developer のデータベースに保存されます。プロセスサーバーの管理者は、古いプロセスを削除する前に、保持する日数、時間、または分に基づいて、データベースから古いプロセスを自動スケジュールで削除できます。

PDD に保持番号を追加することで、完了プロセスまたはフォールトが発生したプロセスを Informatica Business Process Developer のデータベースに保持する日数（あるいは、時間または分）を指定できます。この数は、プロセスコンソールでプロセスに合わせて変更できます。

保持する値を指定しない場合、デフォルトのエンジン値がプロセスに適用されます。この値は、自動データベース保守が有効な場合にのみ必要です。

バージョンニング

プロセスの 1 つのバージョンのみに新しいプロセスインスタンスを作成できることに注意してください。

プロセスのバージョン	デフォルトでは、プロセスサーバーでは新しいバージョンのバージョン番号が自動的に増分されます。自動増分では、マイナーバージョン番号が削除され、メジャーバージョン番号が 1 つ増分されます。例えば、1.5 は 2.0 になります。 このプロセスに独自のバージョン番号を定義するには、1.5 などの番号を入力します。
オンラインの日付	オンライン（有効）日付が空白または過去の場合、プロセスはすぐにオンライン（現在）バージョンになります。特定のプロセスのオンラインバージョンは、新しいプロセスインスタンスを作成可能なバージョンです。将来の日付を設定した場合、有効日が到来すると、このバージョンがサーバーでオンラインバージョンになります。 プロセスを別のタイムゾーンのサーバーにデプロイする場合は、必ずデプロイメント記述子ファイルを編集してタイムゾーン式を含めるようにしてください。詳細については、「 プロセスのバージョンニングについて 」（ ページ 497 ）を参照してください。
オフラインの日付	オフライン（有効期限）の日付が空白の場合、プロセスはオフラインになりません。 限られた期間でプロセスを実行する場合には、オフラインの日付を指定すると便利です。オフラインプロセスバージョンでは新しいプロセスインスタンスを作成することはできませんが、実行中のプロセスインスタンスは正常に完了します。 オフラインの日付を指定できます。プロセスを別のタイムゾーンのサーバーにデプロイする場合は、必ずデプロイメント記述子ファイルを編集してタイムゾーン式を含めるようにしてください。詳細については、「 プロセスのバージョンニングについて 」（ ページ 497 ）を参照してください。
実行中のプロセスの処理	デフォルトでは、すべてのプロセスインスタンスが保持されます。デフォルト値の「バージョンの保持」では、このバージョンがオンラインになった場合に、以前のバージョンで作成されたプロセスインスタンスが最後まで実行されます。 「バージョンの移行」を選択して、実行中のプロセスインスタンスを以前のバージョンから変換し、新しいバージョンを使用することができます。互換性の問題が発生した場合に生成される警告については、サーバーログを確認してください。 プロセスインスタンスが完了しているかどうかにかかわらず、新しいバージョンのオフライン日に以前のバージョンで実行されていたプロセスを終了するには、「終了」を選択します。 詳細については、「 プロセスのバージョンニングについて 」（ ページ 497 ）を参照してください。

デフォルトを変更しない場合、.pdd ファイルにはバージョンの詳細は含まれません。バージョンの詳細を追加または変更する方法については、「[PDD エディタの「ソース」ビューの使用](#)」（[ページ 483](#)）を参照してください。

パートナーロールの呼び出しハンドラ

プロセスデプロイメント記述子（.pdd）ファイルによって、プロセスサーバー環境でプロセスを実行するために必要な情報を記述します。

デフォルトでは、プロセスサーバーは、WSDL でサービス名を検索してエンドポイント（例えば、<soap:address>）を見つけ、標準のエンジン呼び出しフレームワークを使用して、エンドポイントを呼び出します。次のいずれかの選択を行うことで、別の呼び出し方法を選択できます。

呼び出しハンドラ	以下に基づいてエンドポイントを呼び出します。
WSA アドレス	PDD ファイルの<wsa.Address>フィールドの URL。詳細については、「 デプロイメント記述子パートナーリンク用のサービスの選択 」を参照してください。プロセスコンソールで URN を指定して後に、URN を URL にマッピングできることに注意してください。詳細については、「 エンドポイント参照の置換可能な URN と URL の指定 」を参照してください。
WSDL サービスポート	WSDL ファイルのサービスのポートセクションの URL。サービスアドレスをそのまま使用するにはこれを選択します（実行時にエンジンは WSDL のサービスポートのみを検索することを意味します）。サービスアドレスの変更は許可されていません。「 デプロイメント記述子パートナーリンク用のサービスの選択 」を参照してください。
WSRM:アドレス	メッセージが WS-Reliable メッセージングプロトコルを使用して送信されることを除き、上記のアドレスと同じです。「 WS-Reliable メッセージング 」に記載されているように、信頼されたメッセージングポリシーアサーションをアタッチする場合は、このハンドラを選択します。アドレス指定が可能な ReplyTo エンドポイントが必要です。
EJB サービス（オンプレミスのみ）	デプロイされた JAR、WSDL、および EJB インタフェースから作成されたその他のリソース。EJB サービスは Apache Tomcat では使用できないことに注意してください。EJB サービスはその他のアプリケーションサーバーでサポートされています。詳細については、「 Java インタフェースの作成 」を参照してください。
Java サービス	デプロイされた JAR、WSDL、Java インタフェースから作成されたその他のリソース。詳細については、「 Java インタフェースの作成 」を参照してください。
JMS サービス	詳細については、「 Java メッセージングサービス呼び出しハンドラの使用 」を参照してください。アドレス指定が可能な ReplyTo エンドポイントが必要であることに注意してください。
REST サービス	詳細については、「 REST ベースのサービスの使用 」を参照してください。
システムサービス	サーバーログサービス、電子メールサービス、ID サービス、レポートサービス、シェルコマンド呼び出しサービス、データアクセスサービス、REST ベースのサービスの使用などのシステムサービスのいずれかに基づくパートナーのロール。
プロセス	ワークスペースプロジェクトに配置され、呼び出しプロセスと同じサーバーにデプロイされた BPEL プロセスと PDD。詳細については、「 別の BPEL プロセスのサービスとしての BPEL プロセスの作成 」を参照してください。 プロセスを選択するには、「 プロセスサービスの選択 」を参照してください。 注: Informatica Cloud にデプロイする場合は、このオプションを選択します。
プロセス: サブプロセス	ワークスペースプロジェクトに配置され、呼び出しプロセスと同じサーバーにデプロイされた BPEL プロセスと PDD。プロセスはサブプロセスであり、呼び出しプロセスのエンクロージングスコープ内で補償の対象となります。詳細については、「 別の BPEL プロセスのサービスとしての BPEL プロセスの作成 」を参照してください。 プロセスを選択するには、「 プロセスサービスの選択 」を参照してください。

呼び出しハンドラ	以下に基づいてエンドポイントを呼び出します。
EJB ラッパー	作成し、パッケージ化して、EJB をサポートするプロセスサーバーにデプロイしたカスタム EJB。EJB の JNDI の場所と、エンドポイントを呼び出すために必要なクエリ文字列を指定できます。詳細については、「 <i>EJB/Java 呼び出しハンドラのプロパティダイアログ</i> 」を参照してください。
Java ラッパー	作成し、パッケージ化して、サーバーにデプロイしたカスタム Java クラス。Java クラスの完全修飾クラス名と、エンドポイントの呼び出しに必要なクエリ文字列を指定します。詳細については、「 <i>EJB/Java 呼び出しハンドラのプロパティダイアログ</i> 」を参照してください。

パートナーロールのエンドポイントタイプ

プロセスデプロイメント記述子 (.pdd) ファイルによって、プロセスサーバー環境でプロセスを実行するために必要な情報を記述します。pdd ファイルを作成する場合は、パートナーリンクに定義された各パートナーロールのエンドポイントタイプを選択する必要があります。

プロセスデプロイメント記述子ファイルの作成中に、パートナーリンクに定義された各パートナーロールのエンドポイントタイプを選択する必要があります。エンドポイントタイプとは、プロセスがインタラクションを行う Web サービスと通信する方法を示すバインディングプロパティです。さまざまなタイプを使用して、現在および将来使用するサービスの指定を制御できます。例えば、サービスバインディングを静的にすることで「ハードコーディング」を行ったり、プロセスを再デプロイせずに新しいサービスパートナーを動的に変更および追加するためのより多用途なオプションを選択したりすることができます。

エンドポイントタイプについて、次の表に定義を示します。

エンドポイントタイプ	説明
動的	サービスの場所（つまり、エンドポイント参照）が BPEL プロセス内で動的に提供されることを示します。場所は、Assign アクティビティのコピー操作内で動的に割り当てられます。
プリンシパル	<p>この選択は、バージョン 9 で非推奨になりました。パートナー定義ファイルを使用するレガシープロセスの選択はそのまま使用できます。</p> <p>エンドポイント参照が、BPEL プロセスに要求を送信する認証済みプリンシパルの値に基づいて決定されることを示します。この構成は、BPEL プロセスのサービスエンドポイントが保護されていることを前提としています。これらのエンドポイントの 1 つにメッセージが到着すると、認証済みのユーザーのプリンシパル（通常はユーザー名）がプロセスサーバーでのルックアップとして使用されます。サーバーでは、呼び出されたプリンシパルとパートナーリンクタイプに基づいて、使用する適切なエンドポイントが決定されます。</p> <p>プリンシパルエンドポイントタイプは、特定のユーザーとやりとりを行うためのセキュリティと、プロセスおよびそのデプロイメントとは独立してエンドポイントの構成が可能な柔軟性を提供します。</p> <p>プリンシパルは、コールバックに使用されるパートナーロールを設定するために使用するものであるため、非同期のやりとりであることを意味します。パターンは Receive-Invoke で、パートナーリンクは両方のアクティビティで同一です。プロセスは、パートナーリンクの myRole 属性および partnerRole 属性を宣言します。Receive アクティビティのデータを受信すると、パートナーリンクのパートナーエンドポイント参照がパートナー定義ファイルからのコールバックで初期化されます。</p>
静的	<p>エンドポイント情報がデプロイメント記述子ファイルで定義されており、パートナーリンクのすべての呼び出しに使用されることを示します。</p> <p>注: Informatica Cloud にデプロイする場合は、このオプションを選択します。</p>

パートナーのサービスで認証済みアクセスが必要な場合は、[「アクセス時に資格情報が必要なエンドポイント参照」 \(ページ 454\)](#)を参照してください。

プロセスサービスの選択

このダイアログを使用してサービスを選択します

エンドポイントを BPEL プロセスまたはサブプロセスとして呼び出す場合は、次のように [プロセスサービスの選択] ダイアログを使用することができます。

1. 選択する BPEL プロセスの PDD ファイルがワークスペースプロジェクトにあることを確認します。
2. PDD ファイルを参照して選択します。
3. サービス名を選択します。これは、PDD ファイル内の *マイロールパートナーリンク*に関連付けられたいずれかのサービスの名前です。

プロセスまたはサブプロセスの選択については、「*別の BPEL プロセスのサービスとしての BPEL プロセスの作成*」を参照してください。

EJB/Java 呼び出しハンドラのプロパティダイアログ (オンプレミスのみ)

カスタム呼び出しハンドラは、標準の Informatica Business Process Manager 呼び出しフレームワークをバイパスします。標準フレームワークは、メッセージを SOAP 要求にパッケージ化し、HTTP プロトコルを介して指定したサービスエンドポイントアドレスに送信することで、パートナーサービスを呼び出します。

カスタム呼び出しハンドラは、標準の Business Process Manager 呼び出しフレームワークをバイパスします。標準フレームワークは、メッセージを SOAP 要求にパッケージ化し、HTTP プロトコルを介して指定したサービスエンドポイントアドレスに送信することで、パートナーサービスを呼び出します。Web サービスの入出力ライブラリにインタフェースを実装することで、異なる呼び出しメカニズムを作成できます。

カスタムの呼び出しハンドラを作成した場合は、次のいずれかの詳細を入力することで、PDD エディタで呼び出しハンドラのプロパティの詳細を入力できます。

- **EJB**

EJB 呼び出しは、EJB の処理が可能なプロセスサーバーにのみ適しています。EJB 呼び出しの場合は、EJB の JNDI の場所と、エンドポイントを呼び出すために必要なクエリ文字列を指定します。例:

名前: ejb/jndiLocation/MyInvokeHandler

クエリ文字列:

```
invokeHandler=
  "ejb:AeApproverInvokeHandler?type-namespace=
    http://tempuri.org/services/loanapprover&
    type-localpart=approvalMessage&maxLoan=1000000"
```

- **Java**

Java クラス呼び出しの場合は、Java クラスの完全修飾クラス名と、エンドポイントの呼び出しに必要なクエリ文字列を指定します。例:

名前: java.org.ActiveVOS.rt.axis.bpel.MyInvokeHandler

クエリ文字列: parameter=someValue

```
invokeHandler=
  "java.org.activebpel.rt.axis.bpel.AeTestInvokeHandler?
    type-namespace=
    http://localhost:8080/active-bpel/services/OrderIdService&
    type-localpart=getNextOrderIdResponse&
    getNextOrderIdReturn=wsio-109"
```

必ず呼び出しハンドラをパッケージ化してサーバーにデプロイするようにしてください。

マイロールのバインディングサービス名と許可されるロールオプション

プロセスデプロイメント記述子 (.pdd) ファイルによって、ActiveVOS サーバー環境でプロセスを実行するために必要な情報を記述します。 .pdd ファイルを作成する場合は、my_role タイプの各サービスパートナーリンクのバインディングスタイルとサービス名を選択する必要があります。

プロセスデプロイメント記述子ファイルの作成中に、my_role タイプの各サービスパートナーリンクのバインディングスタイルとサービス名を選択する必要があります。

バインディングスタイルとは、RPC (リモートプロシージャコール) とドキュメントの標準の SOAP スタイルで、未パブリッシュおよびポリシー依存と呼ばれるカスタムスタイルがあります。

サービスの操作によって操作を行う単純な変数、スキーマ変数、または複合的なメッセージ変数に適したスタイルを選択します。

- **【ドキュメントリテラル】** 呼び出しとは、SOAP メッセージ全体が XML ドキュメントである単一のエンティティで構成されていることを意味します。この呼び出しは、ドキュメント/リテラルのバインディングスタイルと同等です。
- **【RPC エンコード】** 呼び出しとは、SOAP 要求の本文に操作名と一致する外部要素があり、それぞれが操作のパラメータにマッピングされる内部要素が含まれていることを意味します。
- **【RPC リテラル】** 呼び出しとは、SOAP 要求の本体に操作名と一致する外部要素があり、それぞれの内部パートがその部分のコンテンツを説明するスキーマタイプ定義を指していることを意味します。RPC はバインディングスタイルであり、リテラルは使用法です。
- **【未パブリッシュ】** 呼び出しは、標準の呼び出しフレームワークをバイパスします。未パブリッシュのバインディングを使用してデプロイする主な理由としては、アプリケーションサーバーによって提供される機能を Web サービスのセキュリティ保護または管理に利用できることが挙げられます。Process Developer が提供するサービスパブリケーション用の組み込みのメカニズムを使用する代わりに、J2EE for Web Services (WS4EE) などの呼び出しを介してサービスのパブリケーションを制御できます。また、Process Developer システムサービスは未パブリッシュとして指定されています。
- **【ポリシー駆動型】** 呼び出しとは、SOAP 要求の処理に使用される特定の機能が、myRole エンドポイントにアタッチされたポリシーアサーションによってデプロイメント時に決定されることを意味します。例えば、WS-Reliable メッセージングプロトコルをサポートするサービスには、バインディングスタイルとして [ポリシー駆動型] を選択し、WS-Reliable メッセージングのポリシーを含めることができます。デプロイメント時に、サービスがポリシーアサーションからバインディングスタイルを決定できない場合は、エラーがスローされます。
- **【サービス名】** は、プロセスサーバーで BPEL プロセス用に作成されるエンドポイントの名前です。この名前は、WSDL ファイルのバインディングセクションのアドレス、または任意の名前と一致します。デフォルトでは、サービス名はパートナーリンクに関連付けられたポートタイプに基づいています。
- **【許可されたロール】** はオプションとして使用し、認証されたユーザーが BPEL プロセスと通信するために必要な 1 つ以上のセキュリティロールを指定します。アプリケーションサーバーがセキュリティで構成されていない場合、[許可されたロール] フィールドは無視されます。個々の Web アプリケーションを保護する方法については、アプリケーションサーバーのドキュメントを参照してください。

注: プロセスを Informatica Cloud にデプロイする場合は、このフィールドに入力する必要があります。ロールを 1 つも指定しない場合、プロセスにアクセスできなくなります。

また、[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、ポリシーアサーションをマイロールパートナーリンクに追加することもできます。

デプロイメント記述子パートナーリンク用のサービスの選択

エンドポイント参照のバインディング情報を含むサービスを選択します。[ポリシーアサーション] タブを選択して、エンドポイント参照に 1 つまたは複数のポリシーアサーションを追加します。

デプロイメント記述子パートナーリンク用のサービスの選択

プロセスデプロイメント記述子ファイルの作成中に、パートナーリンクに定義された各パートナーロールのエンドポイントタイプを選択する必要があります。静的タイプを選択すると、サービスバインディングを「ハードコーディング」するために使用するサービスを選択できます。

一致するサービスバインディングが 1 つしかない場合、デフォルトでは、インポートされた WSDL からのバインディング情報が使用されます。ただし、別のサービスにバインドする場合は、[...] をクリックして別のサービスを選択します。

Name	Location	Status
LoanApproval		Missing a partner role endpoint reference definition.
LoanProcess		Missing a partner role endpoint reference definition.
RiskAssessment		Missing a partner role endpoint reference definition.

Partner Role

Invoke Handler: WSA Address

Endpoint type: static

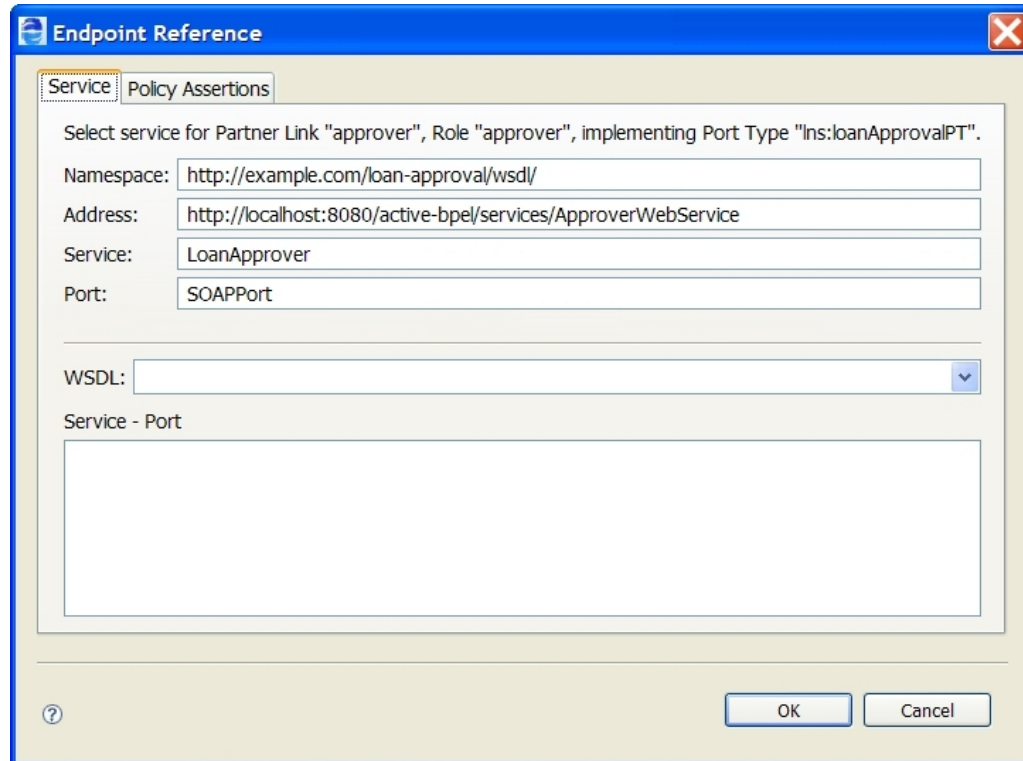
Endpoint Reference:

...

選択する WSDL は、ワークスペースレジストリにあり、バインディング情報が含まれている必要があります。

表示される [エンドポイント参照] ダイアログで、次のいずれかを実行します。

- 選択リストから WSDL を選択します。一致するサービスが表示された後にそのサービスを選択して、上のテキストボックスに名前空間、アドレス、サービス、およびポートを入力します。次の例は、選択されたサービスを示しています。



- バインディング情報が存在しない場合は、サービスの名前空間、アドレス、サービス、およびポートを手動で入力します
- 呼び出しハンドラに WSDL サービスポートを選択した場合は、サービス情報を編集することはできません。実行時には、WSDL サービス定義が常に使用されます。

[「SOAP 1.1 または SOAP 1.2 のポートバインディングの使用」 \(ページ 467\)](#)も参照してください。

[新規デプロイメント記述子] ダイアログを開く方法に関する詳細については、[「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#)を参照してください。

SOAP 1.1 または SOAP 1.2 のポートバインディングの使用

プロセスサーバーバージョン 7 では、SOAP 1.1 および SOAP 1.2 がサポートされています。SOAP 1.2 のポートバインディングを含むサービスを選択できますが、このサービスでポートが指定されていない場合、バインディングはデフォルトでドキュメントリテラルおよび SOAP 1.1 になります。

すべてのマイクロールサービスは、SOAP 1.1 および SOAP 1.2 のバインディングでデプロイされることに注意してください。クライアント側で SOAP 1.2 を利用する場合は、`http://[ホスト:ポート]/active-bpel/services/soap12/[myRoleService.wsdl]` を指定してください。

ポリシーアサーションの追加

エンドポイント参照のバインディング情報を含むサービスを選択します。[ポリシーアサーション] タブを選択して、エンドポイント参照に 1 つまたは複数のポリシーアサーションを追加します。

WS-Policy の仕様により、Web サービスプロバイダとクライアントは、*ポリシーアサーション*を通じて標準的な方法で幅広い機能、要件、および設定を表現できます。必要に応じて、プロセスデプロイメント記述子ファイルのパートナーリンクのエンドポイント参照セクションに 1 つまたは複数のポリシーアサーションを追加できます。

デプロイメント記述子ウィザードでは、**パートナーロール**のパートナーリンクおよびマイロールのパートナーリンクにポリシーアサーションを追加できます。

多くのポリシーアサーションは、*インバウンド*および*アウトバウンド*の方向を指定します。

- インバウンドアサーションは、呼び出されたサービスからの受信および応答のメッセージなど、BPEL プロセスが受信するメッセージを管理します。
- アウトバウンドアサーションは、呼び出されたサービスへの要求や受信に一致する応答など、プロセスが送信するメッセージを管理します。

ポリシーアサーションは、PDD ウィザードのパートナーロールの静的エンドポイント参照に適用されます。動的エンドポイント参照に対しては、Assign アクティビティの適切なコピー操作に詳細を手動で追加します。

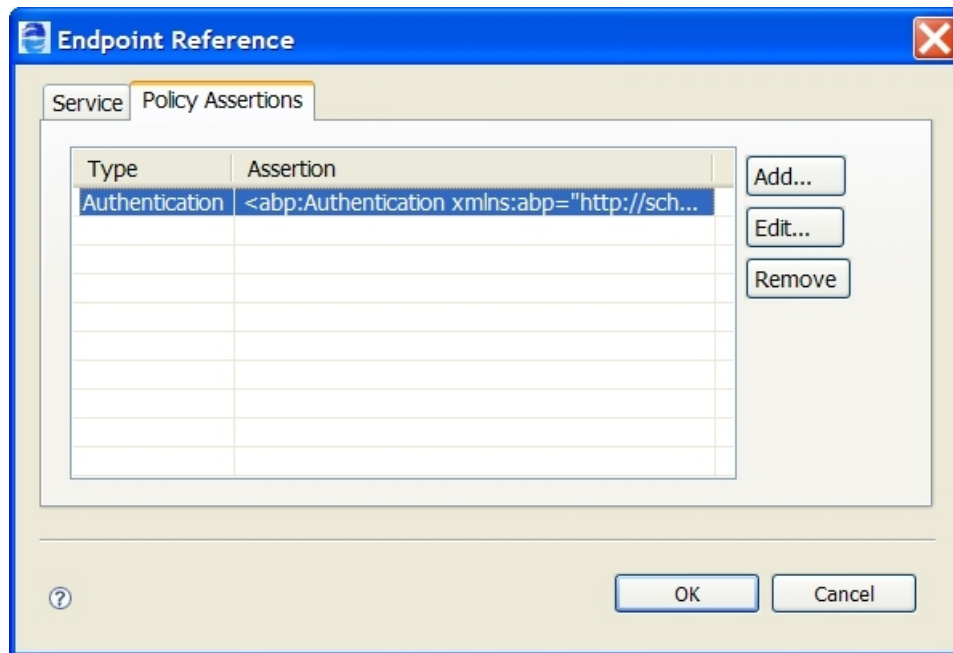
次の表に、サポートされるポリシーアサーションを示します。

ポリシー	説明
WS-Security ポリシーアサーション	
「認証」 (ページ 470)	サービスへのアクセスに必要な資格情報
「暗号化」 (ページ 471)	暗号化する SOAP メッセージのパートを表します
「署名」 (ページ 479)	X.509 証明書トークンを使用して XML 署名で署名する SOAP メッセージのパートを表します
「タイムスタンプ」 (ページ 480)	メッセージの SOAP ヘッダーに<Timestamp>要素を追加します
その他のポリシーアサーション:	
「再試行」 (ページ 476) (パートナーロール)	呼び出したサービスが応答しない場合に再試行するタイミングと回数を表します
「エンジンによって管理される相関」 (ページ 473) (マイロール)	パートナーロールのパートナーリンクへの送信中に、WS-Addressing を使用して replyTo エンドポイント参照を送信するようにプロセスサーバーに指定するマイロールポリシーアサーション
「WS-Reliable メッセージング」 (ページ 481)	保証されたメッセージの配信をサポートする業界標準のプロトコルにパートナーリンクが準拠していることを示します
「JMS 配信オプション」 (ページ 475) (パートナーロール - <i>Informatica Process Developer</i> のみ)	詳細については、 「Java Messaging Service 呼び出しハンドラの使用」 (ページ 338)を参照してください。
「HTTP トランスポート」 (ページ 474)	REST ベースの呼び出しに使用します

ポリシー	説明
WS-Security ポリシーアサーション	
「REST 対応」 (ページ 476) (マイロール)	REST ベースの呼び出しに使用します
「SAML」 (ページ 478)	Security Assertions Markup Language (SAML) は、疎結合およびフェデレーション ID 統合を可能にする OASIS 標準です。
「メッセージ検証」 (ページ 475)	パートナーリンクの WSDL メッセージのきめ細かい検証を提供して、処理を高速化します
「Web サービスタイムアウト」 (ページ 480)	特定の Web サービスのタイムアウトを待機する時間を設定します
「リカバリの呼び出し」 (ページ 475) (パートナーロール)	障害が発生したサーバーからプロセスがリカバリしたときに、呼び出しが保留されているプロセスを中断するかどうかを選択します
「WS-Addressing ヘッダーの送信」 (ページ 479) (パートナーロール)	呼び出しにアドレス指定を明示的に追加します
「WSDL バインディング参照」 (ページ 481) (マイロール)	RPC またはドキュメントの代わりに WSDL バインディングを参照するマイロールのパートナーリンクに適用できます
「xsi:type の非表示」 (ページ 480)	SOAP メッセージのスキーマ検証を抑制するための回避策で、xsi:type 属性を処理できないレガシーサービスを処理する場合に役立ちます
「実行ユーザー名」 (ページ 478)	サブプロセスの呼び出しのみ。プロセスイニシエータを表すユーザー名を追加します。プロセスイニシエータの名前の要求が可能なユーザーアクティビティを含むサブプロセスに役立ちます。

1 つ以上のポリシーをパートナーリンクに追加する手順

1. デプロイメント記述子ウィザードで、[パートナーリンク] ページに移動します。[新規デプロイメント記述子] ダイアログを開く方法に関する詳細については、[「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#)を参照してください。
2. パートナーロールのパートナーリンクに対して、エンドポイント参照タイプとして静的を選択します。
3. [エンドポイント参照] テキストボックスの最後にある [ダイアログ] ボタンを選択します。
4. 次に示すように、[ポリシーアサーション] タブを選択します。



5. **【追加】** ボタンを選択して、**【ポリシーアサーション】** ダイアログを開きます。
6. **【ポリシーテンプレート】** リストから、追加するポリシーを選択して、必要なすべての情報を入力します。詳細については、このトピックの上部のリストにある、サポートされるポリシーの名前を参照してください。
7. **【OK】** を選択します。
8. ポリシーを追加するには、もう一度 **【追加】** を選択します。
9. 次の手順を実行して、マイロールポリシーアサーションを追加します。
 - a. マイロールのパートナーリンクを選択します。
 - b. **【マイロールポリシー】** タブを選択します。
 - c. 上に記載されているように、マイロールサービスに適用可能なポリシーを選択します。

認証

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

このポリシーで、サービスへのアクセスに必要な HTTP 資格情報を表します。

受信認証は、マイロールサービスで受信したメッセージと、呼び出されたパートナーロールサービスからの応答用です。送信認証は、マイロールサービスからの応答と、呼び出されたパートナーロールサービスに送信されるメッセージ用です。

以下のパラメータを設定します。

送信認証ポリシー	
ユーザー名	サービスへのアクセスを許可されたユーザーの名前
パスワード	許可されたユーザーのパスワード

ブリエンプティブな HTTP 資格情報	ブリエンプティブ認証は、リモートホストからの要求を受けることなく、基本認証資格情報を送信します。このオプションは安全なチャネルを介した通信にのみ使用し、ユーザー資格情報の不必要な開示を防ぐように注意する必要があります
ヘッダーで wsse:UsernameToken を送信	オンにすると、SOAP ヘッダーの OASIS WS-Security UsernameToken 要素として資格情報が送信されます。UsernameToken に nonce（nonce は暗号通信で 1 回だけ使用される任意の番号）を含めるには、ウィザードの終了後に PDD エディタの PDD ファイルに<abp:HashPassword/>子要素を追加します。nonce は、文字列の繰り返しを禁止することで、認証のセキュリティを強化するために使用します。以下の例を参照してください。 wsse:UsernameToken ヘッダーは、呼び出しに対する有効な唯一のセキュリティポリシーである場合、暗号化および署名なしで送信できます。それ以外の場合、UsernameToken ヘッダーは署名および暗号化されているため、証明書キーストアを構成する必要があります。
クリアテキストパスワードの使用	許可されたユーザーのパスワードを PDD にクリアテキストで保存するには、このオプションを選択します。パスワードは PDD ファイルで表示および読み取り可能です。
受信認証ポリシー	
メッセージ受信時に UsernameToken が必要	マイロールまたはパートナーロールサービスによって受信されたメッセージの認証を要求するには、これを選択します

例:

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        xmlns:ns5="http://www.example.org/Hello/"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <wsa:Address>http://localhost:8081/active-bpel/services/Hello</wsa:Address>
  <wsa:Metadata>
    <wsa:ServiceName PortName="HelloSOAP">ns5:Hello</wsa:ServiceName>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
               xmlns:abp="http://schemas.active-endpoints.com/ws/2005/12/policy">
      <abp:Authentication direction="out">
        <abp:User>aeadmin</abp:User>
        <abp:Password>yH6CJei+D+s</abp:Password>
        <abp:HashPassword/>
      </abp:Authentication>
    </wsp:Policy>
  </wsa:Metadata>
</wsa:EndpointReference>
```

暗号化

「[ポリシーアサーションの追加](#)」 (ページ 468)に記載されているように、このポリシーアサーションを選択します。

暗号化ポリシーでは、XML 暗号化仕様[XMLENC]の処理ルールに従って、暗号化する SOAP メッセージのパートを記述します。

メッセージ内の指定した元の要素または要素の内容はそれぞれ削除され、暗号化された要素に置き換えられます。

- **【受信】** 暗号化は、マイロールサービスで受信したメッセージと、呼び出されたパートナーロールサービスからの応答用です。これは、マイロールパートナーロールが暗号化を受け入れ、受信したメッセージの暗号化を解除することを示します。

- **【送信】** 暗号化は、マイロールサービスからの応答と、呼び出されたパートナーロールサービスに送信されるメッセージ用です。

以下のパラメータを設定します。

暗号化パートの属性	
エイリアス	暗号化用のキーを取得するために使用されるオプションのキーストアエイリアス。デフォルトは、暗号プロパティファイルで指定されたエイリアスです。
コンテンツ名	暗号化するメッセージパートまたはメッセージ要素
コンテンツ名前空間	メッセージパートまたはメッセージ要素のターゲット名前空間

例

```
<abp:EncryptionParts alias="keystore_alias">
  <abp:Element
    namespace="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-secext-1.0.xsd"
    name="UsernameToken"/>
</abp:EncryptionParts>
```

メッセージコンシューマとして、プロセスサーバーのサービスエンドポイントは、WS-I ガイドラインに適合すると見なされるオプションに準拠したメッセージを受け入れ、コンシュームします。メッセージプロデューサとして、プロセスサーバーは、推奨されるアルゴリズム、参照、および識別子のみをサポートします。

以下のアルゴリズムは、SOAP メッセージのデータ暗号化で使用されます。WS-I の推奨事項と顧客の要求に基づいて、将来のリリースではアルゴリズムがさらに追加され、サポートされる可能性があります。

サポートされるトークンタイプは次のとおりです。

- **X.509 トークン**
 - 直接バイナリ参照（送信および受信）：可能な場合に使用が推奨される方法。
 - 発行者シリアル（送信および受信）：直接不可能な場合に推奨される外部参照方法。
 - X509 識別子（受信のみ）
 - サブジェクトキー識別子（受信のみ）
 - 埋め込みトークン参照（受信のみ）
- **対称型データ暗号化アルゴリズム:**
 - <http://www.w3.org/2001/04/xmlenc#tripledes-cbc>（送信および受信）
 - <http://www.w3.org/2001/04/xmlenc#aes128-cbc>（受信のみ）
 - <http://www.w3.org/2001/04/xmlenc#aes256-cbc>（受信のみ）
- **非対称型キー転送アルゴリズム:**
 - http://www.w3.org/2001/04/xmlenc#rsa-1_5（送信および受信）
 - <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>（受信のみ）
- **署名ダイジェストアルゴリズム:**
 - <http://www.w3.org/2000/09/xmlsig#sha1>（送信および受信）
- **署名アルゴリズム:**
 - <http://www.w3.org/2000/09/xmlsig#rsa-sha1>（送信および受信）
- **Canonical XML 変換アルゴリズム:**
 - <http://www.w3.org/2001/10/xml-exc-c14n#>（送信および受信）

エンジンによって管理される相関

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

BPEL プロセスで受信されたメッセージをプロセスサーバーで相関させるようにするには、相関セットと、エンジンによって管理される相関ポリシーアサーションの 2 つの方法があります。相関セットは、WSDL および BPEL プロセス内で定義されたメッセージプロパティを使用します。エンジンによって管理される相関は、メッセージの SOAP ヘッダー内の WS-Addressing 参照に依存して、受信したメッセージを正しいプロセスインスタンスと相関させます。

エンジンによって管理される相関関係を使用する場合は、マイロールパートナーリンクでポリシーアサーションを指定します。同じパートナーリンク内のパートナーロールに関連付けられたマイロールパートナーリンクを選択します。エンジンによって管理される相関ポリシーアサーションは、WS-Addressing を使用して、パートナーへの送信中に replyTo エンドポイント参照を送信するようにプロセスサーバーに指示します。

ポリシーアサーションの例

```
<myRole allowedRoles="" binding="RPC"
  service="ManagedCorrelationServiceA">
  <wsa:Metadata>
    <wsp:Policy xmlns:abp="http://schemas.
      active-endpoints.com/ws/2005/12/policy"
      xmlns:wsp="http://schemas.xmlsoap.org
        /ws/2004/09/policy">
      <abp:engineManagedCorrelationPolicy/>
    </wsp:Policy>
  </wsa:Metadata>
</myRole>
```

パートナーサービスがエンジンによって管理される相関メッセージを処理する方法

多くの場合、パートナーサービスでは、エンジンによって管理される相関メッセージを自動的に処理するために WSA SOAP バインディングの詳細を実装しています。このサービスは、SOAP ヘッダーから情報を抽出し、それを使用してプロセスサーバーに返信します。

プロセスサーバーがサービスに送信するメッセージは、次の例のように作成されます。SOAP ヘッダーには、相関の conversationId を含む <wsa:ReferenceProperties> 要素を含んだ <wsa:ReplyTo> 要素が含まれていることに注意してください。

プロセスサーバーからの送信メッセージ

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
  /soap/envelope/" xmlns:xsd="http://www.w3.org/2001
  /XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    ...
    <wsa:ReplyTo xmlns:abx="http://www.activebpel.org/bpel/extension"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      <wsa:Address>http://localhost:8080/active-bpel/services
        /ManagedCorrelationServiceA</wsa:Address>
      <wsa:ReferenceProperties>
        <abx:conversationId>1:/process/partnerLinks
          /partnerLink[@name='requestPLT']
        </abx:conversationId>
      </wsa:ReferenceProperties>
    </wsa:ReplyTo>
    ...
  </soapenv:Header>
  <soapenv:Body>
    ...
  </soapenv:Body>
</soapenv:Envelope>
```

サービスでメッセージが受信されると、SOAP ヘッダー <wsa:ReferenceProperties> が抽出され、同様の形式の SOAP ヘッダーで応答します。呼び出しているサービスが想定どおりに応答しない場合は、次の例を使用して、

エンジンによって管理される関連メッセージを受信するためのプロセスサーバーで想定される内容を理解してください。

プロセスサーバーで想定されるインバウンドメッセージ

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
/soap/envelope/" xmlns:xsd="http://www.w3.org/2001
/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
...
<abx:conversationID
xmlns:abx="http://www.activebpel.org/bpel/extension">
1:/process/partnerLinks/partnerLink[@name='requestPLT']
</abx:conversationId>
...
</soapenv:Header>
<soapenv:Body>
...
</soapenv:Body>
</soapenv:Envelope>
```

関連セットの詳細については、[第 24 章, 「関連」 \(ページ 367\)](#)を参照してください。

HTTP トランスポート

HTTP トランスポート

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択し
 ます。

このポリシーはパートナーロールに追加できます。これは、マイロールには適用されません。プロセスからの
 REST サービス要求に応じて、1 つ以上の HTTP トランスポート送信者のタイムアウト値を設定することができ
 ます。

注: このポリシーは現在、REST 対応プロセスとの組み合わせでのみ機能します。

クライアント接 続タイムアウト	接続が確立されるまでのタイムアウト。ゼロの値は、タイムアウトが使用されないこと を意味します。デフォルトはゼロです。
接続プールマネ ージャタイムア ウト	接続の取得までのタイムアウト値を設定します。デフォルトは定義されていません。
ソケットタイム アウト	データを待機するためのタイムアウトであるミリ秒単位のデフォルトのソケットタイム アウト (SO_TIMEOUT)。ゼロのタイムアウト値は、タイムアウトが無限であるとみなさ れます。この値は、HTTP メソッドパラメータにソケットタイムアウトが設定されてい ない場合に使用されます。
TCP No Delay	使用する場合は有効にします。 TCP No Delay パラメータは、TCP (Transmission Control Protocol) パケットのバッチ 処理を制御します。デフォルト値は 1 で、これは TCP パケットがバッチ処理されないこ とを意味します。 Nagle アルゴリズムを使用するかどうかを指定します (Nagle アルゴリズムは、送信され るセグメントの数を最小限に抑えることで帯域幅を節約します)。ネットワーク遅延を減 らしてパフォーマンスを向上させるために、プロセスサーバーは (TCP_NODELAY を有 効にすることで) Nagle アルゴリズムを無効にします。これにより帯域幅の消費量とパ ケット数は増加しますが、データの送信がより高速化されます。

GET によるリダイレクト	HTTP GET 要求を使用して別のページをロードするようにブラウザに指示します。
XML MIME タイプ	データが正しく解釈されるようにするには、image/svg+xml などの MIME タイプを追加します。

リカバリの呼び出し

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

サーバーがダウンした後にオンラインに戻った場合、プロセスサーバーによってアクティブなプロセスが自動的にリカバリされ、再開します。ただし、保留中の Invoke アクティビティの場合はプロセスを一時停止して、サービスによる操作が部分的または完全に完了したかどうかを調べることができます。プロセスコンソールで、入力メッセージデータを使用して、中断されたプロセスを手動で再開することができます。

サーバーが予期せず終了した場合は、プロセスが実行中である可能性があります。プロセスの状態がデータベースに保存されていない可能性もあるため、サーバーはプロセスの一部を実行して、プロセスを元の状態に戻します。ジャーナルエントリには、プロセス状態が保存されるまでのステップが記録され、リカバリ中に再生されます。障害が発生すると、クラスタ内の別のノードが、指定した猶予期間後にプロセスのリカバリを試みる場合があります。

Invoke アクティビティに関連付けられたパートナーリンクのパートナーロールのエンドポイント参照にポリシーアサーションを追加して、プロセスのリカバリ中にその Invoke アクティビティが保留（実行）状態にある場合にプロセスを一時停止するかどうかを指定できます。

次のように、プロセスの一時停止の設定を選択します。

- **デフォルト**。このオプションの属性が存在しない場合、リカバリの呼び出し処理については、[エンジン設定] ページの [エンジンプロパティ] タブにある [リカバリの呼び出し時のプロセスの一時停止] の設定によって指定します。
- **True**。プロセスのリカバリ中に保留中の呼び出しが見つかった場合、プロセスを一時停止します
- **False**。プロセスのリカバリ中に保留中の呼び出しが見つかった場合、プロセスを一時停止しません

JMS 配信オプション

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

JMS メッセージング、配信オプション、および配信オプションに属性を追加する例の詳細については、[「Java Messaging Service 呼び出しハンドラの使用」 \(ページ 338\)](#)を参照してください。

メッセージ検証

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

デフォルトでは、プロセスサーバーエンジンはプロセスの実行前に、プロセス変数にロードされたデータを WSDL スキーマに照らし合わせて検証します。この検証プロパティは、プロセスコンソールで無効/有効にすることが可能なグローバル設定です。

必要に応じて、メッセージ検証をより細かく制御するために、エンドポイント参照にポリシーアサーションを追加します。ポリシーアサーションによって、エンジンのグローバル設定が上書きされます。

メッセージ検証を無効にすることで、より高速な処理が可能になります。ただし、フォールト処理またはアクティビティの検証を有効化しない場合、プロセスに無効なデータが渡されるため、フォールトがキャッチされ

ない可能性が高まることに注意してください。メッセージ検証を有効にすると、無効なデータが検出された場合、プロセスはフォールトをスローします。

次のいずれかを選択します。

- **なし**。メッセージ検証が無効になります。
- **両方**。メッセージ検証が、入力メッセージと出力メッセージに対して有効になります。
- **入力メッセージ検証**が、入力メッセージに対して有効になります。
- **出力メッセージ検証**が、出力メッセージに対して有効になります。

REST 対応

「[ポリシーアサーションの追加](#)」(ページ 468)に記載されているように、このポリシーアサーションを選択します。このポリシーを「マイロール」のパートナーリンクに設定して、REST バインディングを指定できます。

このポリシーは、パートナーリンクのコンテキストパスを提供します。REST 対応ポリシーのコンテキスト値は、デプロイメントマネージャに登録されているプロセス名とパートナーリンク名を検索するために使用される値です。

デプロイメント時に、REST 対応ポリシーを含む「マイロール」パートナーリンクによって、REST パスとプロセスおよびパートナーリンク名間のマッピングが登録されます。

〔用途〕フィールドおよび〔説明〕フィールドはオプションです。〔説明〕フィールドには BPEL プロセス名を追加でき、〔用途〕フィールドには REST サービスのパラメータを追加できます。

REST 対応プロセスのこれらのフィールドに入力すると、BPEL プロセスごとにリストされたデプロイ済みの REST サービスのリスト、および〔用途〕フィールドに追加した詳細を取得できます。

例えば、次の要求 URL は実行時に、説明および用途のパラメータ（オプション）を含むサービスのリストを返します。

`http://localhost:8080/active-bpel/services/REST?`

REST: 次の HTTP リダイレクトの無効化

REST サービスから受信した次の HTTP リダイレクトを無効にするには、以下のポリシーディレクティブを追加します。

```
followRedirects="false"
```

これを行うには、この属性を PDD のポリシー要素に追加するか、動的エンドポイント参照を次のように設定します。

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <wsa:Address>http://localhost:8080/path</wsa:Address>
  <wsa:Metadata>
    <wsp:Policy
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
      xmlns:abp="http://schemas.active-endpoints.com/ws/2005/12/policy">
        <abp:HTTPTransportOptions followRedirects="false"/>
      </wsp:Policy>
    </wsa:Metadata>
  </wsa:EndpointReference>
```

このディレクティブを設定するための UI サポートはありません。

再試行

「[ポリシーアサーションの追加](#)」に記載されているように、このポリシーアサーションを選択します。

このポリシーによって、呼び出しに応答しないサービスを再試行するタイミングと回数を記述します。

プロセスの実行中は、呼び出されたサービスが応答することはありません。サービスがフォールトで応答すると想定して、フォールトが発生した実行をどのように処理するかを指定できます。

特定の回数だけサーバーが呼び出しを自動的に再試行するという比較的単純なポリシーを指定できます。

または、BPEL プロセスを作成して再試行を処理することもできます。詳細については、「[再試行 - ポリシーサービス](#)」を参照してください。

次のように再試行属性を追加します。

アサーションパラメータ	説明	デフォルト
間隔	再試行の間に待機する秒単位の時間	なし。値を追加する必要があります。サービス名を指定している場合は無視されます。
試行回数	サービスを再試行する回数	なし。値を追加する必要があります。サービス名を指定している場合は無視されます。
サービス	プロセスによって実装されたサービスの名前。このサービスは、必要な WSDL 操作に基づく BPEL プロセスです。	サービスを指定すると、試行と間隔の値は無視されます。
faultList	再試行に含めるフォールト	すべてのフォールトが再試行されます
faultExclusionList	再試行から除外するフォールト	
失敗時	再試行が終了したときの呼び出しアクティビティの状態: フォールト（デフォルト）または中断。	フォールト

XML 構文

```
<abp:retryPolicy attempts="xsd:int"?
  interval="xsd:int"?
  service="NCName"?
  faultList="QNameList"?
  faultExclusionList="QNameList"?
  onFailure="fault"/>
```

例 1: サービスとフォールトのリストを含むポリシー:

```
<wsa:EndpointReference
  xmlns:s="http://www.activebpel.org/services/retrytests">
  <wsa:ServiceName PortName="retrytesterServicePort">
    s:retryTesterService</wsa:ServiceName>
  <wsp:Policy
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
    xmlns:abp="http://schemas.active-endpoints.com/ws
      /2005/12/policy">
    <abp:retry service="retryCheckerService">
      faultList="{http://xml.apache.org/axis/*}
        {http://www.active-endpoints.com/2004/06/bpel
          /extensions/*}
        {http://www.activebpel.org/services
          /retrytests}*/>
    </wsp:Policy>
  </wsa:EndpointReference>
```

例 2: 試行とフォールト除外のリストを含むポリシー:

```
<wsa:EndpointReference
  xmlns:s="http://www.activebpel.org/services/retrytests">
  <wsa:ServiceName PortName="retrytesterServicePort">
    s:retryTesterService</wsa:ServiceName>
  <wsa:Metadata>
    <wsp:Policy
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
      xmlns:abp="http://schemas.active-endpoints.com/ws/
        2005/12/policy">
      <abp:retry interval="1" attempts="3">
        faultExclusionList="{http://xml.apache.org/axis/*}">
      </wsp:Policy>
    </wsa:Metadata>
  </wsa:EndpointReference>
```

実行ユーザー名

「[ポリシーアサーションの追加](#)」に記載されているように、このポリシーアサーションを選択します。

このポリシーは、「[パートナーロールの呼び出しハンドラ](#)」に記載されているように、プロセスおよびサブプロセスの呼び出しハンドラにのみ適用されます。このポリシーアサーションを使用して、プロセスイニシエータのデフォルト値を上書きします。

デフォルトでは、プロセスインスタンスはプロセスイニシエータと呼ばれるユーザーロールに関連付けられています。サーバーが保護されている場合、プロセスサーバーでは呼び出し側プロセスのプロセスサーバーの資格情報を使用してプロセスイニシエータの ID が入力されます。また、サーバーが保護されていない場合は、匿名でプロセスイニシエータの ID が入力されます。

[実行ユーザー名] の値は文字列です。ID サービスの実際のユーザーである必要はありません。

SAML

「[ポリシーアサーションの追加](#)」 ([ページ 468](#))に記載されているように、このポリシーアサーションを選択します。

Security Assertions Markup Language (SAML) は、疎結合およびフェデレーション ID 統合を可能にする OASIS 標準です。SAML は、ポリシードメイン間で ID 関連のセキュリティ情報を通信する方法を標準化します。

SAML アサーションは通常、ID プロバイダ（マイロールパートナーリンク）からサービスプロバイダ（パートナーロールパートナーリンク）に転送されます。アサーションには、サービスプロバイダがアクセス制御の決定を行うために使用するステートメントが含まれています。

現在サポートされているバージョンは、SAML 1.1 および 2.0 です。SAML の詳細については、www.oasis-open.org の OASIS Security Services (SAML) TC を参照してください。

方向	<ul style="list-style-type: none">- 送信。通常、パートナーロールのパートナーリンクに対して選択します。パートナーサービスに送信されるメッセージは信頼済みメッセージになります- 受信。通常、マイロールのパートナーリンクに対して選択します。パートナーサービスからプロセスに返送されたメッセージは、信頼済みメッセージとして受け入れられます- トランスポートメカニズムが SOAP over JMS などの SOAP over HTTP 以外のメカニズムである場合は、この両方が必要になる場合もあります。信頼済みのメッセージを送受信します。
バージョン	使用する SAML のバージョン

サブ ジェ クト 名	(オプション) 送信メッセージの場合、ID 情報に関連付けられたユーザーを示すサブジェクト名を追加します。例えば、LDAP サービスから識別名を入力することもできます。
確認 方法	送信メッセージの場合は、次の方法を選択します。 <ul style="list-style-type: none"> - sender-vouches: SSL 証明書で信頼がすでに確立されている場合、デジタル署名は不要で、sender-vouches を使用できます。 - holder of key: 信頼が確立されていない場合は、キーの所有者を選択して、信頼の証明がアサーション自体のデジタル署名を介して送信されるということを示すことができます。
認証 方法	送信メッセージの場合、サブジェクト名の認証に使用する方法を選択します（アサーションの情報が現在の要求を送信している当事者を参照しているかどうかを判別するため）。 デフォルトは urn:oasis:names:tc:SAML:1.0:am:unspecified です。 他のオプションの使用に関する詳細については、このトピックの概要に記載されている「アドレスでの SAML 仕様」を参照してください。

WS-Addressing ヘッダーの送信

「[ポリシーアサーションの追加](#)」（[ページ 468](#)）に記載されているように、このポリシーアサーションを選択します。

このポリシーアサーションは、パートナーロール partnerlink に追加できます。このポリシーは、要求を行う場合にアドレス指定を使用する必要があるかどうかをエンドポイントが明示的に宣言できるようにするための拡張要素です。このポリシーを追加すると、アドレス指定が呼び出しに自動的に追加されます。

詳細については、<http://www.w3.org/TR/ws-addr-metadata/#indicatinguse> を参照してください。

署名

「[ポリシーアサーションの追加](#)」（[ページ 468](#)）に記載されているように、このポリシーアサーションを選択します。

署名ポリシーによって、署名された情報の検証と信頼を可能にする X.509 証明書トークンを使用して、XML 署名で署名する SOAP メッセージのパートを記述します。

- **【受信】** 署名は、マイロールサービスで受信したメッセージと、呼び出されたパートナーロールサービスからの応答用です。これは、マイロールパートナーロールが、署名されたメッセージコンテンツを受け入れて署名を検証することを示します。
- **【送信】** 署名は、マイロールサービスからの応答と、呼び出されたパートナーロールサービスに送信されるメッセージ用です。

以下のパラメータを設定します。

SignatureParts エイリアス	署名するキーを取得するために使用されるオプションのキーストアエイリアス。デフォルトは、暗号プロパティファイルで指定されたエイリアスです。
コンテンツ名	署名するメッセージパートまたはメッセージ要素
コンテンツ名前空間	メッセージパートまたはメッセージ要素のターゲット名前空間

例

```
<abp:SignatureParts alias="keystore_alias">
  <abp:Element
    namespace="http://docs.oasis-open.org/wss
      /2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
```

```
name="UsernameToken"/>
</abp:SignatureParts>
```

メッセージコンシューマとして、プロセスサーバーのサービスエンドポイントは、WS-I ガイドラインに適合すると見なされるオプションに準拠したメッセージを受け入れ、コンシュームします。メッセージプロデューサとして、プロセスサーバーは、推奨されるアルゴリズム、参照、および識別子のみをサポートします。

次のアルゴリズムは、SOAP メッセージパートの署名に使用されます。

- **X.509 トークン**

直接バイナリ参照（送信および受信）：可能な場合に使用が推奨される方法。

発行者シリアル（送信および受信）：直接不可能な場合に推奨される外部参照方法。

X509 識別子（受信のみ）

サブジェクトキー識別子（受信のみ）

埋め込みトークン参照（受信のみ）

- **署名ダイジェストアルゴリズム:**

<http://www.w3.org/2000/09/xmldsig#sha1>（送信および受信）

- **署名アルゴリズム:**

<http://www.w3.org/2000/09/xmldsig#rsa-sha1>（送信および受信）

- **Canonical XML 変換アルゴリズム:**

<http://www.w3.org/2001/10/xml-exc-c14n#>（送信および受信）

xsi:type の非表示

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

このポリシーアサーションをパートナーロールのパートナーリンクまたはマイロールのパートナーリンクに追加して、SOAP メッセージの xsi:type 属性を非表示にすることができます。

このポリシーアサーションを有効にした場合、Type 属性がないと検証できないため、Invoke または Receive のメッセージに対するスキーマの検証が事実上無効になります。

このポリシーアサーションは、xsi:type 属性の処理ができず、修正またはアップグレードができないレガシーサービスの回避策としてのみ使用する必要があります。

タイムスタンプ

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

このアサーションは、OASIS WS-Security 1.0 仕様で定義されているとおりに、メッセージの SOAP ヘッダーに<Timestamp>要素を追加します。通常、タイムスタンプは、[「署名」 \(ページ 479\)](#)のポリシーアサーションに記載されているように署名されます。

Time To Live 属性により、タイムスタンプが検証される受信側の制限が示されます。

Direction 属性は、タイムスタンプが、呼び出されたサービスまたはサーバーでのみ検証されるか、あるいはその両方で検証されるかを示します。値は、in（サーバー）、out（呼び出されたサービス）、またはその両方になります。

Web サービスタイムアウト

[「ポリシーアサーションの追加」 \(ページ 468\)](#)に記載されているように、このポリシーアサーションを選択します。

パフォーマンス上の理由により、受信に対応する応答アクティビティおよび同期呼び出しは、プロセスサーバーで 10 分以内に実行されない場合、タイムアウトになります。タイムアウトエラーが発生する場合は、応答または同期呼び出しアクティビティが実行されずにプロセスがタイムアウトになるまでの 10 分間という待機時間を延長します。デフォルト値は 600 秒です。

タイムアウト値は秒単位で入力します。ゼロの値は、待機時間が無制限であることを示します。

WS-Reliable メッセージング

「[ポリシーアサーションの追加](#)」 (ページ 468)に記載されているように、このポリシーアサーションを選択します。

WS-Reliable メッセージングポリシーアサーションによって、保証されたメッセージの配信をサポートする業界標準のプロトコルにパートナーリンクが準拠していることを示します。

OASIS ビジネス標準コンソーシアムの WS-Reliable メッセージング標準は、メッセージの配信を保証するためのプロトコルを提供します。プロセスサーバーはこの標準をサポートしているため、BPEL プロセスでプロトコルを使用して、信頼性の高いメッセージの配信を識別、追跡、および管理することができます。

信頼性の高いメッセージングのプロセスサーバーへの実装については、次のような説明に基づいています。

メッセージの送信後、プロセスサーバーは [確認応答間隔] で設定した時間待機して、確認応答メッセージを取得します。プロセスサーバーがタイムアウトし、確認応答を受信しなかった場合、プロセスサーバーは、[基本再送信間隔] で設定した時間待機してから再送信を行います。[非アクティブのタイムアウト] で設定した時間内にシーケンスが完了しない場合、プロセスサーバーではそのシーケンスが終了したと見なされます。

次のように、WS-Reliable メッセージングの属性を追加します。

アサーションパラメータ	説明	デフォルト
確認アドレス	確認応答に使用されるアドレス	<code>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</code> このアドレスは、WS-Addressing で定義されている匿名のロールを表し、確認応答が同期的に送信されることを示します。
マイロールのバインディング	このパートナーリンクで使用される SOAP バインディングスタイル。PDD ウィザードの [パートナーリンク] ページに表示されるバインディングを指定します。	ドキュメントリテラル
非アクティブのタイムアウト	シーケンスが終了したと見なされるまでにプロセスサーバーが待機する時間。これは、比較的長い期間で設定することができます。	240000 (ミリ秒単位)
基本送信間隔	未確認の要求を再送信する前にプロセスサーバーが待機する時間。	20000 (ミリ秒単位)
確認間隔	メッセージが未確認と見なされるまでにプロセスサーバーが待機する時間。これは通常、比較的短い値となります。	10000 (ミリ秒単位)

WSDL バインディング参照

「[ポリシーアサーションの追加](#)」 (ページ 468)に記載されているように、このポリシーアサーションを選択します。

サービスバインディングの詳細について WSDL バインディングを参照するには、マイロールパートナーリンクのポリシー駆動型のバインディングスタイルを選択してから、このオプションを選択します。

名前空間の URI および関連するバインディングの詳細を入力します。または、バインディングの詳細が一致するワークスペースにある WSDL を選択します。

インデックス付きプロパティの追加

サーバーで実行中の特定のプロセスをすばやく見つけるために使用するインデックス付きプロパティを追加します。

インデックス付きプロパティは、プロセスの実行中に変数（または変数パート）の値が入力されるプロセス変数のプロパティです。インデックス付きプロパティを PDD に追加すると、プロセスコンソールで、プロセスインスタンスの検索フィルタとしてインデックス付きプロパティを選択できるようになります。

また、インデックス付きプロパティを使用して、ビジネスプロセスのイベントを定義することもできます。詳細については、[第 22 章, 「ビジネスイベント処理」 \(ページ 347\)](#)を参照してください。

例えば、customerId が 101 であるすべてのフォールト状態のプロセスを見つけることができます。この例では、インデックス付きプロパティの名前は customerId で、文字列型のプロセス変数に基づいています。「101」は、プロセスが中断された時点でのデータ値です。

インデックス付きプロパティを使用すると、実行時に重要なデータ項目をすばやく検索できます。

次のように、インデックス付きプロパティの詳細を追加します。

名前	インデックス付きのプロパティ名。この名前は、プロセスコンソールの [インデックス付きプロパティ] リストに表示されます。
タイプ	プロパティタイプを Boolean、Date、Double、String から選択します。タイプを変数、パート、またはクエリ値と一致させます
変数パス	完全なプロセス変数パス。選択リストから選択するか、[ダイアログ] ボタンを選択して [BPEL 変数の検索] ダイアログを開きます。変数のリストが長い場合は、ダイアログが便利です。
パート	メッセージ型の変数のプロセス変数パート。
クエリ	プロセス変数パートの詳細（オプション）。

参照の表示

PDD エディタの [参照] タブには、デプロイメント記述子で使用されているリソースが表示されます。PDD をデプロイすると、BPEL プロセスとこの [参照] タブにリストされたすべてのリソースがサーバーにデプロイされます。

[「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#)も参照してください。

PDD のイベントタブ

詳細については、[第 22 章, 「ビジネスイベント処理」 \(ページ 347\)](#)を参照してください。

PDD の [ユーザー] タブ

詳細については、このヘルプの「ヒューマンタスク」セクションを参照してください。

PDD エディタの [ソース] ビューの使用

プロセスデプロイメント記述子エディタは、スキーマに照らし合わせて .pdd ファイルを検証します。このエディタでエラーが検出されると、ファイルの保存時にエラーログにエラーが生成されます。

このファイルは、テキストエディタである [ソース] タブで編集するのが便利です。例えば、各 Web サービスの実際のエンドポイントの場所を指定したり、URN を入力したりすることができます。

静的エンドポイント参照を追加する手順

1. パートナーリンクごとに、次の手順を実行します。
 - `FILL_IN_NAMESPACE` を、パートナーリンクタイプが定義されている WSDL ファイルのターゲット名前空間に置き換えます。
 - `FILL_IN_ADDRESS_URI` を、サービスの WSDL ファイルの <service> セクションで指定されているサービスの URI に置き換えます。または、[「エンドポイント参照の置換可能な URN と URL の指定」 \(ページ 455\)](#) に記載されているように、URN を指定することもできます。
 - `FILL_IN_PORT_NAME` を、サービスの WSDL ファイルの <service> セクションで指定されているサービスポート名に置き換えます。
 - `FILL_IN_SERVICE_NAME` を、サービスの WSDL ファイルの <service> セクションで指定されているサービス名に置き換えます。
2. ファイルを保存します。

PDD エディタの [ソース] タブの使用に関するヒント:

- エラーと警告のリストについては、必ずエラーログを確認してください。
- 必要に応じて、エディタで行番号を表示できます。**[設定]** を選択するには、[ソース] ビューを右クリックします。必要に応じて、[行番号とその他のテキストエディタの設定を表示する] を有効にします。

[「プロセスデプロイメント記述子ファイルの作成」 \(ページ 456\)](#) も参照してください。

ビジネスプロセスアーカイブコントリビューションの作成とデプロイ

選択したデプロイメント記述子ファイルと、デプロイするその他のプロジェクトリソースを表示します。[カタログリソース] ページで、選択および除外されたリソースのリストを表示して、1 つの単位として管理されるすべてのファイルがコントリビューションに含まれていることを確認します。エクスポート時にファイルをデプロイし、スクリプトを保存して .bpr ファイルを再デプロイします。

プロセスをプロセスサーバーにデプロイするには、プロセスとリソースのコントリビューションを、Web アーカイブファイルと同様のビジネスプロセスアーカイブファイル (.bpr ファイル) にパッケージ化します。

.bpr ファイルを作成する前に、「[プロセスデプロイメント記述子ファイルの作成](#)」に記載されている .pdd ファイルを作成する必要があります。必要に応じて、プロセスセントラルフォームやタスクなどの追加リソースを使用できます。

プロジェクトごとに 1 つの BPR ファイルのみを作成することをお勧めします。プロセスとリソースは、サーバー上のファイルの単位として管理されます。詳細については、「[デプロイメントコントリビューションの管理](#)」を参照してください。

.bpr ファイルを作成する手順

1. 現在のプロジェクトのプロジェクト参照を作成して使用している場合は、必ず最初にそのコントリビューションをデプロイしてください。プロジェクトの依存関係は、最初にサーバーにデプロイする必要があります。
2. [ファイル] > [エクスポート] > [オーケストレーション] > [コントリビューション-ビジネスプロセスアーカイブ] を選択します。
3. 次の例に示すように、すべてのプロセスデプロイメント記述子ファイル (.pdd ファイル) がアーカイブに含まれるように事前に選択されていることに注意してください。

Export Business Process Archive

Contribution Specification

Select deployment descriptor files to be contributed, location of the contribution (business process archive, BPR) and deployment method.

Select the deployment descriptors to contribute:

- ☒ Tutorial
 - ☒ deploy
 - ☒ tutorial.pdd

Select the export destination:

BPR file: /Tutorial/deploy/tutorial.bpr Browse...

Server Deployment Option

Type: Web Service

Deployment URL: http://localhost:8080/active-bpel/services/ActiveBpelDeployBPR

Username:

Password:

Options

Group: Tutorials

Description: ActiveVOS Tutorials

☒ Save the contribution specification as an Ant script in the workspace (.bprd)

BPRD file: /Tutorial/deploy/tutorial.bprd Browse...

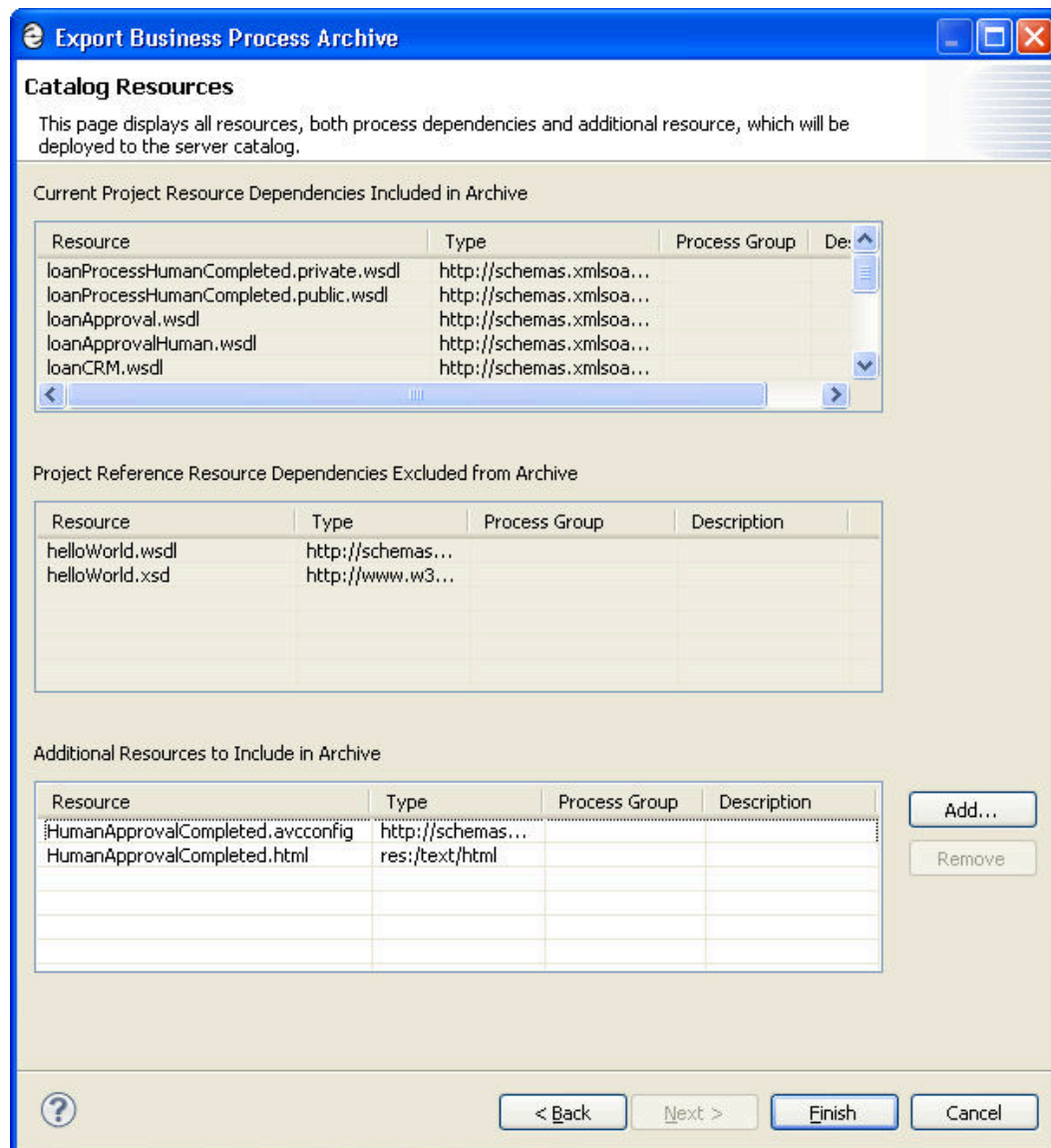
? < Back Next > Finish Cancel

4. BPR ファイルをエクスポートする場所とファイル名を選択します。

5. 次のように【**サーバーデプロイメントタイプ**】を選択します。
 - **なし**。エクスポートの完了時に BPR アーカイブをデプロイしない場合は、このオプションを選択します。
 - **ファイル**。ファイルシステムの場所にエクスポートするときにアーカイブを保存する場合は、このオプションを選択します。【**デプロイメントの場所**】フィールドで、任意のフォルダを参照します。
 - **Web サービス**。サーバーが実行中で、そのサーバーに自動的にデプロイする場合は、このオプションを選択します。サーバーが保護されている場合は、サーバーにアクセスするためのユーザー名とパスワードを入力します。
6. 必要に応じて、このコントリビューションの【**グループ**】名と説明を入力します。プロセスコンソールで、ファイルを一覧表示するための選択フィルタとしてグループ名を使用できます。説明は、コンソールの【**コントリビューションの詳細**】ページに表示されます。
7. 必要に応じて、【**ワークスペースの Ant スクリプトとしてコントリビューションの仕様を保存**】オプションを選択し、オーケストレーションフォルダのデプロイフォルダなど、このファイルの場所を選択します。このオプションの詳細については、「*BPRD スクリプトを使用した BPR コントリビューションの再生成とデプロイ*」を参照してください。
8. 【**次へ**】をクリックして、【**カタログリソース**】のリストを表示します。以下のトピックを参照してください。
 - コントリビューションの設定
 - プロジェクトの依存関係のデプロイと除外された依存関係の表示
 - 追加リソースのデプロイ
9. 【**完了**】をクリックして、.bpr ファイルとオプションの.bprd ファイルを作成し、エクスポートします。
10. プロジェクトを更新して、新しい BPR ファイルを表示します。
「デプロイメントコントリビューションの管理」も参照してください。

プロジェクトの依存関係のデプロイと除外された依存関係の表示

「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」に記載されているように、プロジェクトにプロセスのプロセスデプロイメント記述子（PDD）を含めることで、デプロイメントをプロジェクトレベルで開始します。次に、【**ビジネスプロセスアーカイブのエクスポート**】ウィザードで、例に示すように、事前に選択したすべてのプロジェクトリソースと、デプロイメントから除外されたプロジェクト参照を表示します。



このダイアログの各部分に関する説明は以下のとおりです。

プロジェクトリソースの依存関係

現在のプロジェクトリソースの依存関係がデプロイされます。次のタイプが含まれます。

- このプロジェクトからの WSDL とスキーマ
- プロジェクト参照を指定していない場合に、別のワークスペースプロジェクトからインポートされた WSDL とスキーマ
- ヒューマンタスク HTML フォーム
- (表示されません) 関数のパラメータとして指定したファイルの場所に基づいて、doXSLTransform 関数で使用する XSL ファイル
- PDD で指定した Java Jar

アーカイブから除外されたプロジェクト参照の依存関係 (デプロイされていないもの)

このコントリビューションをデプロイする前に、これらのリソースをコントリビューションにデプロイする必要があります。

プロジェクトプロパティで指定した、プロセス内で使用されるプロジェクト参照が「カタログリソース」ページに一覧表示されます。例えば、参照された WSDL に基づいてパーティシパントを作成すると、WSDL が除外リストに表示されます。この方法を使用する場合の利点に関する詳細については、「[プロジェクト参照の使用](#)」を参照してください。

追加リソース

追加リソースは、「[コントリビューションの設定](#)」に基づいて、事前に自動的に選択されます。

「[追加リソースのデプロイ](#)」に記載されているように、さらに他のリソースをデプロイできます。

追加リソースのデプロイ

サーバーにデプロイするリソースを追加します。選択したデプロイメント記述子ファイルと、デプロイするその他のプロジェクトリソースを表示します。「カタログリソース」ページで、選択および除外されたリソースのリストを表示して、1つの単位として管理されるすべてのファイルがコントリビューションに含まれていることを確認します。エクスポート時にファイルをデプロイし、スクリプトを保存して.bpr ファイルを再デプロイします。

デプロイメントのベストプラクティスの概要については、「[プロジェクトの依存関係のデプロイと除外された依存関係の表示](#)」 ([ページ 485](#))を参照してください。

プロセスの開発時に追加可能なリソースは、依存リソースと非依存リソースの2種類です。依存リソースには、WSDL、スキーマ、ヒューマンタスクフォーム、doXSLTransform 用の XSL、および PDD で指定した Java Jar が含まれます。依存リソースは、プロセス PDD を含む同じ BPR コントリビューション、または参照されるプロジェクトにデプロイできます。

XQuery モジュールは別の BPR にデプロイすることをお勧めします。

別の BPR にデプロイする可能性のあるリソースは次のとおりです。

- レポート。詳細については、[第 34 章、「プロセスサーバーおよびプロセスセントラルレポート」](#) ([ページ 538](#))を参照してください。
- 一元的な設定。プロセスセントラルのクライアントアプリケーションをカスタマイズします。詳細については、「[プロセスセントラルについて](#)」 ([ページ 511](#))を参照してください。
- バイナリファイルと HTML ファイル。プロセス要求フォームの場合、HTML、言語ファイル、画像、スタイルシートをエクスポートできます。詳細については、「[プロセスセントラルについて](#)」 ([ページ 511](#))を参照してください。
- XQuery モジュール。「[XQuery 関数の記述](#)」 ([ページ 299](#))を参照してください。

サーバーに追加のリソースをデプロイする手順

1. サーバーが実行中ではないことを確認し、リソースファイルが Workspace プロジェクトにあることを確認します。
2. **[ファイル] > [エクスポート] > [オーケストレーション] > [コントリビューション - ビジネスプロセスアーカイブファイル]** を選択します。
3. リソースをプロセスに直接インポートしていない場合は、プロセスデプロイメント記述子ファイル (.pdd ファイル) をスキップして、.bpr アーカイブに含めることができます。
4. .bpr アーカイブをエクスポートする場所とファイル名を選択します。
5. [デプロイメント] セクションで、[タイプ] に [Web サービス] を選択します。
6. [カタログリソース] ページで、必要に応じて **[追加]** を選択します。
7. リソースのタイプと場所を選択します。プロジェクトエクスプローラからリソースを選択します。それ以外の場合は、URI を追加します。可能な場合、URI は、リソースタイプに関連する任意の参照にします。例えば、XML リソースの URI は、XML ドキュメントのルート名前空間である必要があります。

「[フォームレポートと構成ファイルのデプロイ](#)」 ([ページ 531](#))も参照してください。

8. [リソース] リストで、次の手順を実行します。
 - 必要に応じて、新規または既存のグループ名を入力します。[サーバー] に、グループごとのリソースが表示されます。
 - 必要に応じて、説明を入力します。この説明は、サーバーの [カタログリソース] リストに表示されます。
9. 必要に応じてリソースを追加します。

Informatica Cloud へのデプロイメント

ビジネスプロセスアーカイブ (.bpr ファイル) には、プロジェクトリソースを Informatica Cloud Secure Agent にデプロイするために必要なすべてのリソースが含まれています。

Informatica Cloud のクラウドサーバーまたは Secure Agent のいずれかにデプロイする場合、オンプレミスでデプロイする際には適用されない次のような 2 つの要件があります。

1. 許可されたロールを指定する。
2. サーバーデプロイメントオプションをファイルに設定する。

このトピックでは、Process Developer でこれらのオプションを設定する方法および以下の方法について説明します。

- Informatica Cloud サーバーにデプロイする方法
- Secure Agent にデプロイする方法

許可されたロールを指定する

次のように、.pdd でのプロセスの実行が許可されたロールを指定します。

1. ロールを指定できるプロセスを選択します。例えば、[マイロール] パネルを開き、次のように [バインディングスタイル] で [ドキュメントリテラル] を選択します。

The screenshot shows the 'Partner Links' configuration window. It has a table with columns 'Name', 'Location', and 'Status'. The 'Name' column contains 'LoanApproval', 'RiskAssessment', and 'loanProcessor'. The 'loanProcessor' row is selected. Below the table are two panels: 'Partner Role' and 'My Role'. The 'Partner Role' panel has fields for 'Invoke Handler' (set to 'WSDL Service Port'), 'Endpoint type' (empty), and 'Endpoint Reference' (empty). The 'My Role' panel has fields for 'Binding' (set to 'Document Literal'), 'Service' (set to 'TutorialService'), 'Allowed Roles' (set to 'aeAdmin'), and 'Policy' (empty). At the bottom, there are tabs: 'General', 'Partner Links', 'Indexed Properties', and 'Event'.

Name	Location	Status
LoanApproval		
RiskAssessment		
loanProcessor		

Partner Role

Invoke Handler: WSDL Service Port

Endpoint type:

Endpoint Reference:

My Role

Binding: Document Literal

Service: TutorialService

Allowed Roles: aeAdmin

Policy:

General Partner Links Indexed Properties Event

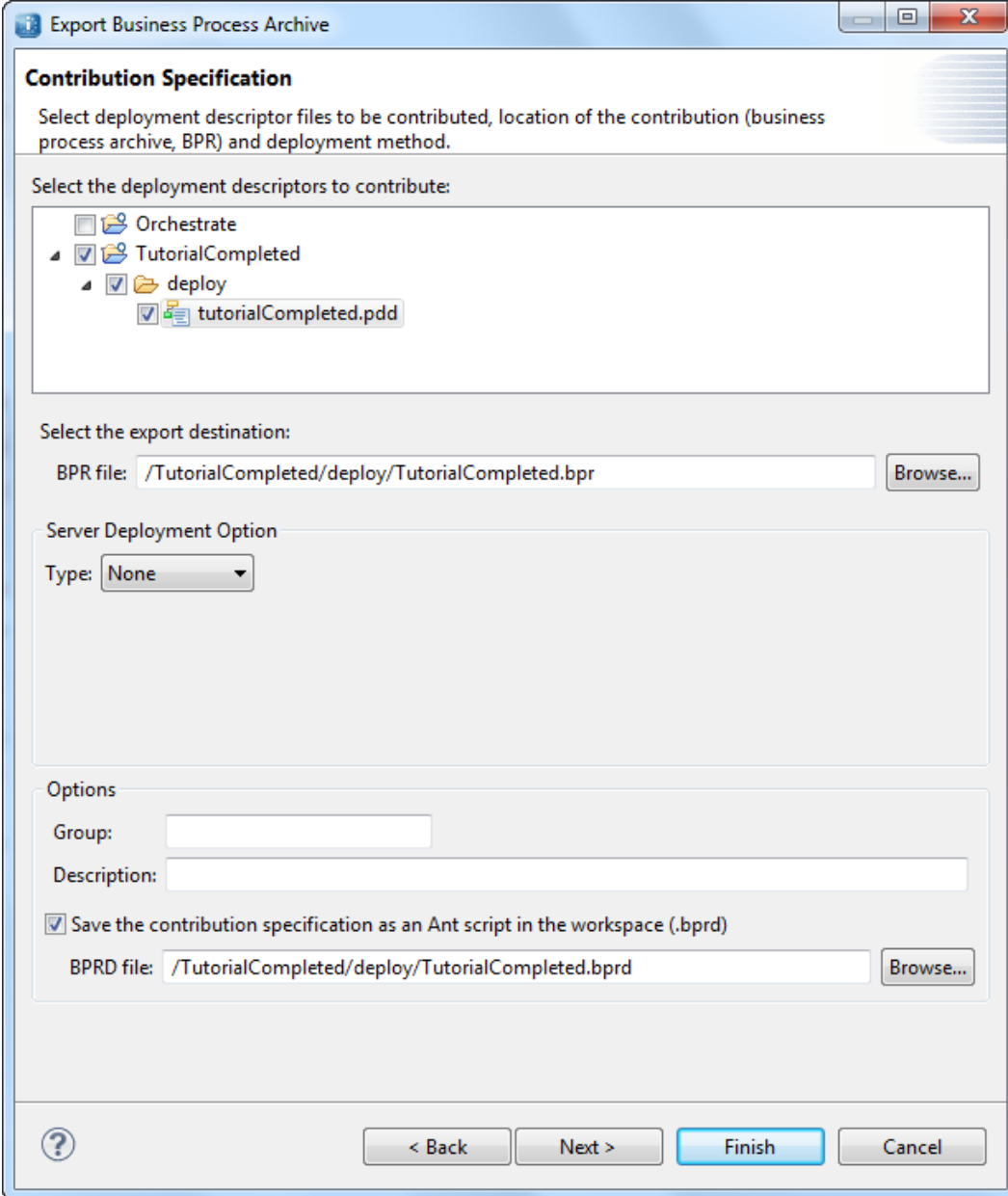
2. [サービス] フィールドに、ロールを識別するための名前を入力します。
3. プロセスにアクセス可能な、すべての許可されたロールを追加します。

注: ユーザーがアプリケーションにアクセスできるように、許可されたロールを少なくとも 1 つ追加する必要があります。

サーバーデプロイメントオプションをファイルに設定する

プロジェクトを BPR ファイルにエクスポートすると、次のようなダイアログが表示されます。デフォルトでは、サーバーデプロイメントオプション（タイプ）は [なし] に設定されています。

Informatica Cloud にデプロイするには、この値を変更する必要があります。タイプリストから **[ファイル]** を選択します。



The image shows a Windows-style dialog box titled "Export Business Process Archive". It contains several sections for configuring the export process.

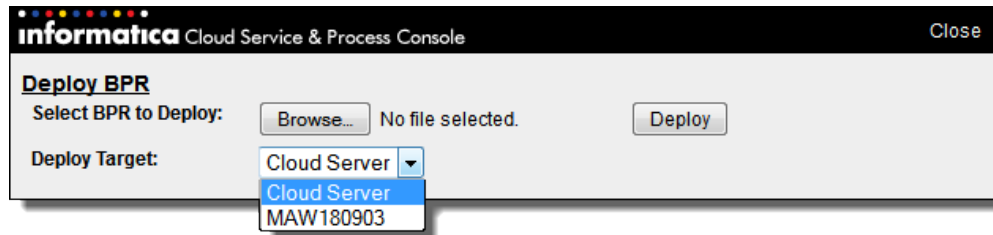
- Contribution Specification:** A section with a description: "Select deployment descriptor files to be contributed, location of the contribution (business process archive, BPR) and deployment method."
- Select the deployment descriptors to contribute:** A tree view showing a hierarchy of folders and files. The "TutorialCompleted" folder is expanded, showing a sub-folder "deploy" which contains the file "tutorialCompleted.pdd". All items are checked with checkboxes.
- Select the export destination:** A section with a label "BPR file:" followed by a text box containing the path "/TutorialCompleted/deploy/TutorialCompleted.bpr" and a "Browse..." button.
- Server Deployment Option:** A section with a label "Type:" followed by a dropdown menu currently set to "None".
- Options:** A section with two text boxes labeled "Group:" and "Description:". Below them is a checked checkbox labeled "Save the contribution specification as an Ant script in the workspace (.bprd)". Below the checkbox is another "BPRD file:" text box with the same path as the one above and a "Browse..." button.
- Navigation:** At the bottom, there are four buttons: a help icon (?), "< Back", "Next >", and "Finish". The "Finish" button is highlighted in blue.

Informatica Cloud にデプロイする

Informatica Cloud サーバーへのデプロイ

.bpr ファイルを作成した後にプロセスを Informatica Cloud にデプロイするには、次の手順を実行します。

1. Informatica Cloud にログインします。ユーザーには管理者権限が必要です。
2. **【ホーム】** > **【プロセス監視】** を選択します。
3. プロセスコンソールページの右上隅にある **【デプロイ】** をクリックします。
4. ダイアログで、**【参照】** をクリックして .bpr ファイルを選択します。
5. **【ターゲットをデプロイ】** リストから、次のように **【クラウドサーバー】** を選択します。



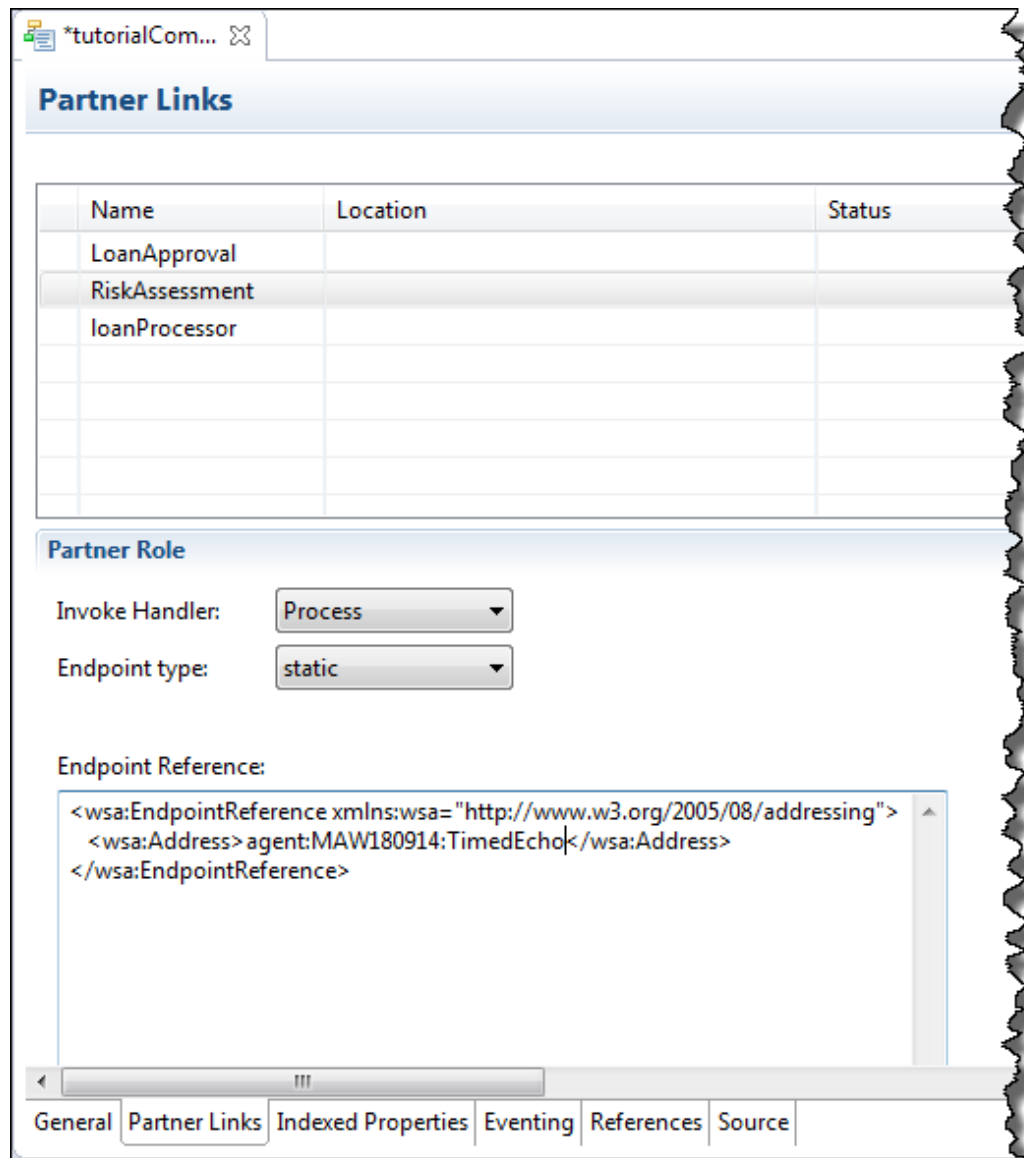
6. **【デプロイ】** をクリックします。
7. プロセス監視内で **【カタログ】** > **【プロセス定義】** を選択して、プロセスがデプロイされたことを確認します。

Secure Agent にデプロイするための要件

プロセスを Secure Agent にデプロイする場合、そのプロセスはクラウドサーバーへのデプロイと同様となります。さらに、次の手順を実行します。

1. 以下に示すように、エンドポイント参照を指定します。
2. 次のオプションを選択します。
 - 呼び出しハンドラ: プロセス
 - エンドポイントタイプ: 静的

以下に例を示します。



このアドレスの構造は `agent:agent_name:agent_service` です。

agent

これにより、Process Developer はどのような情報が後に続くのかを把握します。

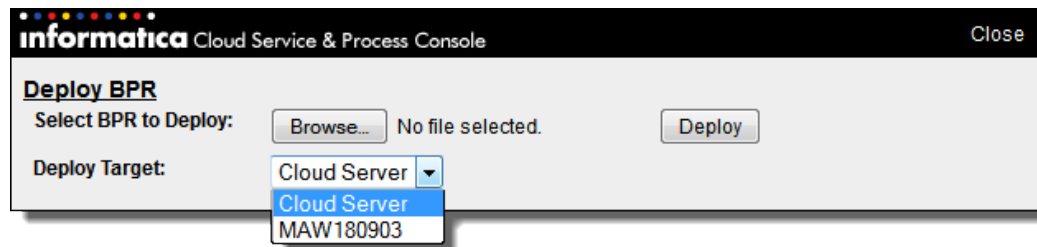
agent_name

通常、これはエージェントが実行されるシステムです。

agent_service

インストールされているサービスの名前。

プロセスコンソールでプロセスをデプロイする場合は、エンドポイント参照で使用したものと同一エージェント名をデプロイターゲットとして選択します。



デプロイメントコントリビューションの管理

サーバーにデプロイするリソースを追加します。

コントリビューションとは、デプロイされたビジネスプロセスアーカイブ（BPR）で、ファイルの単位としてサーバー上で管理されます。個々の BPEL プロセスとリソースをデプロイして置き換えるのではなく、現在のファイルおよび更新されたファイルを、サーバー上で簡単に管理可能なユニットとしてデプロイします。

いずれかの Process Developer のサンプルプロジェクトをデプロイする場合、サンプルベースのコントリビューションは 1 つだけ設定する必要があります。

コントリビューション名

各コントリビューションには、プロジェクトの場所に基づく識別子があります。プロセスコンソールの [コントリビューション] ページには、project:/myProject ファイルのリストが表示されます。各コントリビューションには、ワークスペースプロジェクト名に従って名前が付けられています。.BPRD ファイルでは、プロジェクト名に基づく contributionURI という名前の属性を表示できます: project:/myProject。場所に関するこのヒントは、サーバー上のデプロイメントを追跡する場合に役立ちます。

コントリビューションの状態

コントリビューションには、オンライン、オフライン、オフライン保留中という状態があります。

最初のデプロイメントはバージョン 1.0 で、オンラインです。ファイルを更新して再デプロイすると、古いコントリビューションはオフライン保留中となり、新しいコントリビューションがオンラインになります。

1 つのバージョンのみがオンラインとなります。オフライン保留中のバージョンに関連付けられた実行中のすべてのプロセスが完了すると、古いバージョンはオフラインになります。

コントリビューションのサーバー管理

コントリビューションを使用することで、次のタスクをサーバーで簡単に実行できます。

- コントリビューションを削除して、古いプロセスインスタンスと古いリソースをすべて削除する。
- 他の開発者のリソースに干渉しない独自のリソースを管理します。
- 現在の（オンライン）コントリビューションを以前のバージョンにロールバックします。また、その逆も可能です。
- [コントリビューションの詳細] ページで、コントリビューションのデプロイメントログに簡単にアクセスできます。

コントリビューションのデプロイメントのヒント

コントリビューションの使いやすさを維持するには、次のデプロイメントのヒントに必ず従ってください。

- 親コントリビューションをデプロイする前に、必ずプロジェクト参照コントリビューションをデプロイしてください。詳細については、「プロジェクト参照の使用」を参照してください。

- プロジェクトごとに1つのBPRを管理します。デプロイ可能なすべてのファイルをBPRに含めます。エクスポートウィザードを使用して再デプロイします。
- プロジェクトリソースを更新する場合は、プロジェクト全体を再デプロイします
- 新しいリソースを追加した後に、プロジェクト全体を再デプロイします。
- リソースを削除した後に、プロジェクト全体を再デプロイします。

BPRD スクリプトを使用した BPR コントリビューションの再生成およびデプロイ

ワークスペースプロジェクトを選択して、将来の.bpr アーカイブの再デプロイ時に実行するための Ant スクリプトを保存します。

に記載されているように、.bpr [「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#) ファイルを作成する場合に、.bpr ファイルをすばやく再生成してデプロイするために使用可能な Ant スクリプト (.bprd ファイル) を作成するというオプションがあります。

BPRD Ant スクリプトを使用して次の手順を実行します。

- BPEL またはプロセスデプロイメント記述子ファイルを変更した後に、.bpr ファイルを更新します
- .bpr ファイルを自動的にデプロイするためにデプロイメントパス情報を追加します
- XSL スタイルシートなど、ワークスペースプロジェクトに存在しないリソースに対して、適切なエントリを追加します

BPRD Ant スクリプトは、Process Developer 内または Process Developer 外部のコマンドラインから実行できます。

詳細については、次を参照してください:

- [「Process Developer 内からの BPRD Ant スクリプトの実行」 \(ページ 493\)](#)
- [「コマンドラインからの BPRD Ant スクリプトの実行」 \(ページ 494\)](#)

Process Developer 内からの BPRD Ant スクリプトの実行

[「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#)に記載されているように、.bpr ファイルを作成する場合に、.bpr ファイルをすばやく再生成してデプロイするために使用可能な Ant スクリプト (.bprd ファイル) を作成するというオプションがあります。

.bpr ファイルの自動デプロイメント

.bpr アーカイブの作成時にデプロイメントタイプを選択していない場合は、デプロイメント情報を BPRD スクリプトに追加して、スクリプトを実行することができます。

1. プロジェクトエクスプローラで、.bprd ファイルをダブルクリックして開きます。
2. スクリプトのコメントに従って、ファイルまたは Web サービスにデプロイメント情報を追加します。アーカイブの有効なデプロイメント先を指定し、デプロイメントターゲットを有効にする必要があります。例えば、archive.deployment 値は、次のように入力します。
`http://localhost:8080/active-bpel/services/ActiveBpelDeployBPR`
3. ファイルを保存します。

4. サーバーが実行中であることを確認してから、プロジェクトエクスプローラで、.bprd ファイルの右マウスメニューを選択し、**【実行】**を選択します。
プロセスマネージャでは、新しい.bpr がデプロイされた際に通知が表示されます。

.bpr ファイルの再生成

元のアーカイブ内の 1 つ以上のファイルを変更した場合は、.bpr ファイルを更新する必要があります。アーカイブに多くの BPEL ファイルや.pdd ファイルが含まれている場合は、.bpr ファイルを最初に作成した際に作成した Ant スクリプトを使用してアーカイブを更新する方が迅速かつ便利な場合があります。

新しいプロセスとそのリソースをアーカイブに追加する場合は、**【ファイル】 > 【エクスポート】** オプションを使用して新しい.bpr を作成するか、.bprd ファイルを手動で変更して、構文的に正しい参照を新しいファイルに追加する必要があります。.bprd ファイルを手動で変更するには、Ant スクリプト言語の使用方法をよく理解している必要があります。

.bpr ファイルを再生成する手順

1. プロジェクトエクスプローラで、ビジネスプロセスアーカイブ (.bpr) ファイルを最初に作成した際に作成した.bprd ファイルを選択します。
2. マウスを右クリックして、**【実行】**を選択します。
Process Developer では、新しい.bpr が生成された際に通知が表示されます。

コマンドラインからの BPRD Ant スクリプトの実行

[「ビジネスプロセスアーカイブコントリビューションの作成とデプロイ」 \(ページ 483\)](#)に記載されているように、.bpr ファイルを作成する場合、Ant スクリプト (.bprd ファイル) を作成するというオプションがあります。このファイルは、Process Developer 内またはコマンドラインで実行できます。

コマンドラインから BPRD Ant スクリプトを実行するには、Ant 環境をセットアップし、Process Developer の Ant ランタイムコンポーネントをインストールする必要があります。

JAVA 7 および IPv6 を使用している場合、ネットワークの問題が発生することがあります。BPR のデプロイメントまたはクラスタ通信で問題が発生した場合は、サーバーの起動時に `-Djava.net.preferIPv4Stack=true` フラグを使用する必要があります。

Ant を設定するには、Ant 環境でファイルを利用できるようにする必要があります。次のいずれかの方法を選択してください。

方法 1: JAR を Ant インストールにコピーして Ant 環境をセットアップする

次のファイルが `{${ant.home}}\lib` ディレクトリにあることを確認してください

- axis.jar
- bpr_tasks.jar
- jaxrpc.jar
- commons-logging.jar
- commons-discovery.jar
- saaj.jar
- wsdl4j.jar

方法 2: B ユニットのランタイムをインストールし、BPRD ファイルを編集して、以下に説明する変更を加える

次の行を変更します:

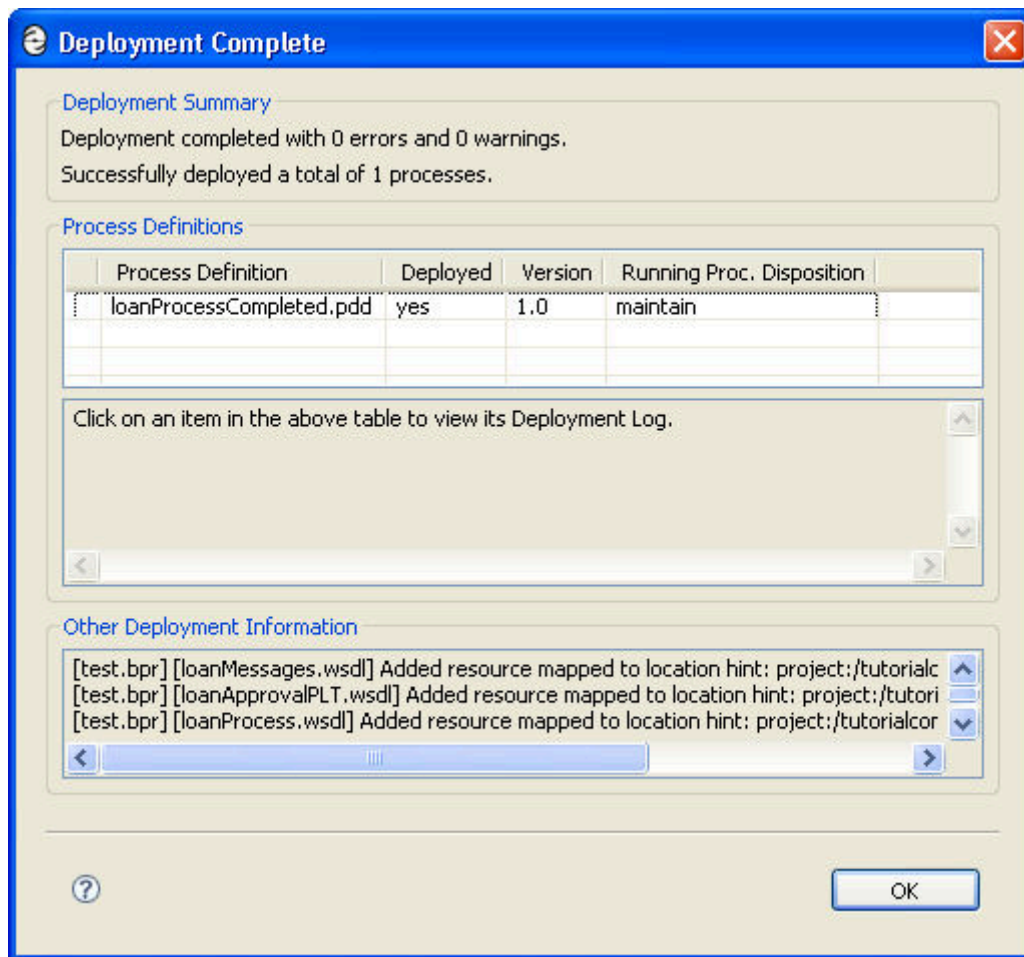
変更した行は次のようになります:

デプロイメントの完了

このダイアログでは、プロセスファイルのデプロイメントの結果をすばやく確認できます。

【デプロイメントの完了】 ダイアログには、.bpr ファイルをプロセスサーバーにデプロイした結果が表示されます。

次の図は、このダイアログの例を示しています。



このダイアログでは、プロセスファイルのデプロイメントの結果をすばやく確認できます。次の点に注意してください。

- 正常にデプロイされたそれぞれの Process Developer プロセスとリソースには、バージョン番号があります。バージョンは、デプロイメントのコントリビューションによって提供されます。
- プロセス定義を選択して、簡単なデプロイメントログを表示できます。完全なデプロイメントログについては、プロセスコンソールの [コントリビューション詳細] ページを確認してください。
- WSDL ファイルは検証されず、自動的にデプロイされます
- すべてのリソースのデプロイメントの詳細は、このダイアログの下部に表示されます

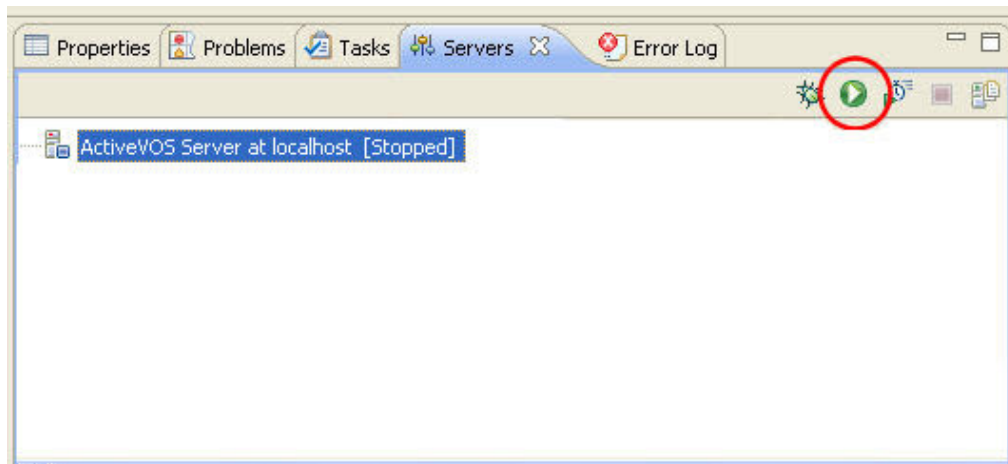
サーバーの起動とプロセスの実行

「[ビジネスプロセスアーカイブコントリビューションの作成とデプロイ](#)」(ページ 483)に記載されているプロセスをデプロイした後に、サーバーを起動し、適切なメッセージを送信してプロセスをインスタンス化することで、プロセスを実行できます。

Process Developer に付属するプロセスサーバーは、Apache Tomcat で実行中のプロセスサーバーエンジンで構成されています。Tomcat は、Java Servlet および Java Server Pages テクノロジーの公式リファレンス実装で使用されるサーブレットコンテナです。

プロセスをデプロイする前に、組み込みエンジンを起動する必要があります。組み込みエンジンを起動すると、このエンジンによって新規の.bpr ファイル、更新済みの.bpr ファイル、および削除された.bpr ファイルがスキャンされます。

1. 組み込みエンジンを起動していない場合は、「[組み込みプロセスサーバーのセットアップ](#)」(ページ 96)に記載されているようにサーバーをセットアップします。
2. ワークスペースの右下にある [サーバー] ビューを選択します。
3. 例に示すように、[サーバーの起動] ボタンを選択します。



サーバーが起動すると、コンソールで起動タスクがスクロールします。組み込みサーバーを起動するたびに、一部のファイルが組み込みサーバーにデプロイされます。サーバーの起動が完了すると、上記のように、サーバーが起動していることが [状態] に表示されます。

プロセスコンソールを使用して、デプロイしたアクティブなプロセスを管理できます。コンソールを表示するには、[プロセスコンソール] ツールバーボタンを選択し、ブラウザのアドレスフィールドに次の URL が表示されていることを確認します。

`http://localhost:8080/activevos`

コンピュータでポート 8080 がすでに使用されている場合は、ポート番号を変更します。

詳細を参照する場合は、コンソール内の [ヘルプ] を選択します。

BPEL プロセスをインスタンス化する方法

BPEL プロセスは、Web サービスと同じように呼び出します。プロセスサービスは、`http://[マシン名:ポート]/active-bpel/services/[サービス名]` という URL でデプロイされます (ここで、[サービス名] は、プロセスデプロイメント記述子ファイル内のマイロール要素のサービス属性の値です)。

また、プロセスは `http://[マシン名:ポート]/active-bpel/services/soap12/[サービス名]` という URL でもデプロイされるため、プロセスに要求を送信するクライアントは、SOAP 1.2 バインディングを使用して送信を行うことができます。デフォルトのバインディングは SOAP 1.1 です。

プロセスのバージョンングについて

プロセスサーバーでは、プロセスのバージョンングを管理できます。バージョンングにより、異なるバージョンを持った特定のプロセスの存在が許可されます。2 度のデプロイメントを行った場合、BPEL ファイル内のターゲット名前空間と名前が同じであれば、同じプロセスの異なるバージョンであると見なされますが、一方のデプロイメントともう一方のデプロイメントには何らかの差異があります。

プロセスのバージョンングを使用すると、プロセスをオンライン（有効）にするタイミングや期間を制御できます。また、新しいバージョンがオンラインになった場合に、古いバージョンで作成されたプロセスをどのように扱うかを制御することもできます。複数のバージョンのプロセスが同時に存在する可能性があります、新しいプロセスインスタンスの作成が可能なのは最新のオンラインバージョンのみです。

最新のバージョンはオンラインの状態です。他の状態としては、将来の有効な日付があるバージョンを示す *オンライン保留中*、有効期限を迎えたバージョンまたは有効期限が設定されたバージョンを示す *オフライン保留中*、および実行中のプロセスインスタンスを持たない期限切れのバージョンを示す *オフライン* があります。

プロセスデプロイメント記述子は、デプロイメントのバージョンング方法を説明するための選択肢を提供します。これらの選択はすべてオプションです。また、以下に説明するようなデフォルト値を持ちます。

必要に応じて、[「プロセスデプロイメント記述子ファイルの作成」](#)（ページ 456）に記載されているようにバージョン情報を指定できます。

次の例は、.pdd ファイルのバージョン情報の構文を示しています。

```
<version effectiveDate="2005-12-12T00:00:00-05:00"
  expirationDate="2007-12-12T00:00:00-05:00" id="1.5"
  runningProcessDisposition="maintain"/>
```

ここで、

- **オンラインの日付**は、新しいバージョンが現在のバージョンになる日付で、新しいプロセスインスタンスはすべてこれに対して実行されます。実行中のプロセスに対して選択した処理によっては、一部は古いバージョンの使用が終わるまで引き続き実行されます。オンラインの日付は、XML スキーマの日時値です。時間は、コンピュータのタイムゾーンに協定世界時（UTC）の前後の時間数をプラスまたはマイナスした深夜時間として示されます。上の例では、コンピュータのタイムゾーンは、UTC から 5 時間遅れの東部標準時です。日付/時刻の詳細については、[「期限と期間の式」](#)（ページ 293）を参照してください。オンラインの日付を指定しない場合は、プロセスがサーバーにデプロイされた日時にデフォルト設定されます。
- **オフラインの日付**は、オンライン日より先、現在のバージョンの期限が切れる日付です。オフラインバージョンは、新しいプロセスインスタンスを作成できません。オフライン保留バージョンに結び付けられた実行中のすべてのプロセスが完了すると、バージョンはオフラインになります。オフラインバージョンのすべてのプロセスインスタンスは、完了するまで実行されます。オフラインの日付は、XML スキーマの日時値です。オフラインの日付を指定しない場合、プロセスコンソールで手動で設定するか、新しいバージョンをデプロイするまで、バージョンはオフラインになりません。
- **id** は、major.minor 形式で表すプロセスバージョン番号です。バージョン番号を指定する必要はありません。プロセスサーバーは、新しいバージョンを自動増分します。サーバーは、minor 値を減らして major 値を 1 つ増やすことで、バージョン番号を増分します。例えば、バージョン 1.5 はバージョン 2.0 に増分されます。

- **runningProcessDisposition** は、このバージョンの有効日を迎えた後に、プロセスサーバーが、実行中のプロセスを持つ同じプロセスの他のバージョンに対して実行するアクションです。
[移行] を指定した場合は、プロセスコンソールでサーバーログを表示して、実行中のプロセスとそれらのプロセスが実行されている新しいバージョンとの間の非互換性を説明する警告メッセージを確認してください。詳細については、「プロセスコンソールのヘルプ」を参照してください。

有効な処理は次のとおりです。

- **Maintain**。以前のバージョンのすべてのプロセスインスタンスは、完了するまで実行される必要があることを示します。これがデフォルト値です。
- **Terminate**。以前のバージョンで実行中のすべてのプロセスインスタンスは、プロセスインスタンスが完了したかどうかに関係なく、新しいバージョンの有効日に終了する必要があることを示します。
- **Migrate**。以前のバージョンで作成された実行中のすべてのプロセスインスタンスは、状態情報を移行し、オンライン日を迎えたときに新たにデプロイされたプロセス定義を使用します。バージョン間で互換性のない変更がある場合、説明的な警告メッセージがプロセスコンソールのサーバーログに生成されます。詳細については、「プロセスコンソールのヘルプ」を参照してください。

第 31 章

エンドポイントと URL

プロセスサーバーには、ビジネスプロセス実行言語（BPEL）プロセスを実行および管理するための次のコンポーネントが含まれています。

- プロセスサーバーエンジンおよび関連するデータベース構成ファイル
- プロセスコンソール
- プロセスセントラルは、要求、レポート、およびタスクを管理するアプリケーションです。
- カスタムタスク管理クライアントアプリケーションのサポート
- ビジネス分析のためのビジネスイベント処理
- ユーザー定義のレポート
- ビジネスプロセスロジックと HTML フォームを更新するためのカタログリソース

プロセスサーバーエンジンは、プロセスのバージョン管理（有効日と有効期限）、パートナーエンドポイントの呼び出しハンドラ、エンドポイントの再試行、セキュリティ、その他の多くのポリシー、およびプロセス例外管理などのプロセスを管理します。

プロセスコンソールは、デプロイされたアクティブなプロセス、アラートシステム、およびエンドポイントの場所を管理するためのページを提供します。また、プロセスコンソールには、パフォーマンスの監視と管理のためのエンジン構成設定があります。

サービスエンドポイント

プロセスをプロセスサーバーにデプロイすると、サービスエンドポイントとして使用できるようになります。ユーザーおよび自動化されたビジネスプロセスは、サービスエンドポイントに要求を送信することでプロセスインスタンスを起動します。

プロセスコンソールで [サービス定義] を選択することで、サービスのリストを表示できます。

デフォルトのサービスエンドポイント: SOAP 1.1

デフォルトでは、REST サービスを除くすべてのサービスは次の場所にリストされています。

`http://localhost:8080/active-bpel/services`

この URL には、SOAP 1.1 バインディングを持つすべてのサービスが表示されます。SOAP 1.1 プロセスを開始する要求には、次のものが含まれます。

- ドキュメントリテラル、RPC リテラル、エンコードされた RPC、またはポリシー依存のバインディングスタイル（ポリシーアサーションによって処理されます）
- マルチパートメッセージ

- WS-I に準拠していない可能性のあるプロセス
- SOAP ヘッダー
- HTTP GET および POST メソッドへのバインディング

追加のサービスエンドポイント

SOAP 1.1 に加えて、以下のサービスエンドポイントを使用できます。

- `http://active-bpel/services/SOAP12`
SOAP 1.2 サービス。REST サービスと ActiveBpelAdmin サービスを除くすべてのサービスを呼び出すことができます。上記の SOAP 1.1 サービスと同じように、要求はサービスに送信されます。
- `http://active-bpel/services/REST`
REST サービスのみが呼び出されます。次の条件に従って要求を送信する必要があります。
 - 要求が、`aeRest.xsd` で定義した入力メッセージに準拠している
 - HTTP メソッド、HTTP GET、POST、PUT、および OPTIONS
- `http://active-bpel/services/XML`
REST と ActiveBpelAdmin サービスを除くすべてのサービスを呼び出すことができます。次の条件に従って要求を送信する必要があります。
 - バインディングスタイルがドキュメントリテラルである
 - 入力メッセージと出力メッセージが単一部分の要素である
 - プロセスが WS-I に準拠している
 - 要求は、`text/xml` というヘッダーコンテンツタイプを持つ HTTP POST です。

詳細については、[「プロセスサーバーの API と SDK」 \(ページ 501\)](#)を参照してください。

- `http://active-bpel/services?JSON`
REST と ActiveBpelAdmin サービスを除くすべてのサービスを呼び出すことができます。次の条件に従って要求を送信する必要があります。
 - バインディングスタイルがドキュメントリテラルである
 - 入力メッセージと出力メッセージが単一部分の要素である
 - 入力メッセージが JSON で表されている
 - プロセスが WS-I に準拠している
 - 要求は、`application/json` というヘッダーコンテンツタイプを持つ HTTP POST です。

詳細については、[「プロセスサーバーの API と SDK」 \(ページ 501\)](#)を参照してください。

- `http://host:port/active-bpel/catalog/project:/relative_path_to_resourceFolder`
例:
`http://localhost:8080/active-bpel/catalog/project:/AutoStep/wsd1/telco.wsd1`
`http://localhost:8080/active-bpel/catalog/project:/AutoStep/schema/telco.xsd`

プロセスサーバーの URL

次の表は、プロセスサーバーの実行中に使用可能な URL のリストを示しています。

http://<host>:<port>/	説明
activevos	プロセスコンソール
activevos-central	プロセスセントラル
activevos-central/inbox	B4P プロセス用の古いスタイルのタスククライアント
active-bpel/services	詳細については、「 サービスエンドポイント 」 (ページ 499)を参照してください。
active-bpel/catalog/project:/ path/to/catalog/ resource-type/ resource-name	次のような WSDL、スキーマ、またはその他のリソースを開きます。 http://localhost:8080/active-bpel/catalog/ project:/ HumanTaskDemo/schema/loanRequest.xsd
active-bpel/i18n/ name_of_props.properties	プロセスセントラルの多言語サポート用の外部化された文字列
active-bpel/avccatalog/ project:/path/to/catalog / name_of_form.html	プロセスセントラル要求フォームまたはタスクフォームを開きます。 abTaskClient セキュリティロールでのログインが必要です。
activevos/process/ processview.jsp? pid=101&readonly=1&view=2	[アクティブなプロセス詳細] ビューを読み取り専用にするには、 readonly=1 パラメータを使用します。[アクティブなプロセス詳細] ビ ューのさまざまなコンポーネントを表示および非表示にするには、次 の引数を指定して view パラメータを使用します。 1: デフォルト。すべてのコンポーネント（アウトライン、キャンバス、 プロパティ）を表示 2: キャンバスのみ 3: アウトラインおよびキャンバス 4: キャンバスおよびプロパティ

プロセスサーバーの API と SDK

開発者は、複数のプロセスサーバー API を利用できます。API については、*Process Developer* のヘルプに記載されています。また、このヘルプにはサンプルコードおよびアプリケーションも記載されています。

次の表は、プロセスサーバーのインタフェースの部分的なリストです。

名前	説明	サービスエンドポイント http://<host>:<port>/
プロセスセントラル	主に AvosCentralApi サービスに構成が依存するプロセスセントラル	active-bpel/services/JSON/ AvosCentralApi
WS-HT に対する Informatica Business Process Manager の拡張機能	<i>authorize</i> や <i>getInstance</i> などのヒューマンタスクとやり取りするための操作	active-bpel/services/ AeB4PTaskClient- aeTaskOperations
管理サービス	プロセスコンソールと通信するための操作	active-bpel/services/ ActiveBpelAdmin
WS-HT	OASIS 標準委員会によって示されている Web サービス - ヒューマンタスク操作	active-bpel/services/ AeB4PTaskClient- taskOperations
WS-HT タスクフィード	RSS または ATOM シンジケーションフィードとして表される <i>getMyTasks()</i> 操作	active-bpel/services/REST/ AeB4PTaskFeed
XML バインディングと JSON バインディング	単純な XML バインディングおよび JavaScript Object Notation (JSON) バインディングを介してすべてのプロセスとサービスを公開するプロセスサーバー	詳細については、「 プロセスサーバーの URL 」(ページ 501)を参照してください。

第 32 章

セキュリティ構成

OASIS WS-Security V1.0 (WSSE) 標準は、SOAP メッセージの整合性と機密性を保証するためのフレームワークを確立します。メッセージの整合性はデジタル署名によって保証され、メッセージデータの機密性は暗号化によって実現されます。WS-Security V1.0 で参照される標準は、プロセスサーバーでサポートされています。

WS-Security 機能を使用するには、プロセスサーバーをデプロイしたコンテナに、標準の SSL 処理で頻繁に使用される必要な証明書マネージャまたはキーストアをセットアップする必要があります。また、プロセスサーバーがプラットフォームの暗号化および証明書管理との間でデータをやり取りするために必要な設定を含むプロパティファイルを指定する必要があります。このプロパティファイルの名前は `crypto.properties` です。

プラットフォームに合わせて、`crypto.properties` で次のプロパティを設定する必要があります。

- **`org.apache.ws.security.crypto.provider=<provider>`**
ここで、`apache` 暗号インタフェースを実装するカスタムプロバイダを指定しない場合、`<provider>` はデフォルトとなります。デフォルトは `org.apache.ws.security.components.crypto.Merlin` です
- **`org.apache.ws.security.crypto.merlin.keystore.type=<type>`**
`<type>` は、キーストアの形式（通常は `jks` または `pks12`）に基づいて指定します。
- **`org.apache.ws.security.crypto.merlin.keystore.alias=<alias>`**
`<alias>` は、プライベートキーと証明書を識別するための名前です。
- **`org.apache.ws.security.crypto.merlin.keystore.password=<password>`**
ここで、`<password>` はオプションのプロパティです。キーストアにパスワードが必要な場合は、パスワードを含めます。
- **`org.apache.ws.security.crypto.merlin.file=<keystore filename>`**

キーストアは、`crypto.properties` の `org.apache.ws.security.crypto.merlin.file` で指定したパスを使用して、サーバーがファイルシステムからアクセスできるようにする必要があります。また、`crypto.properties` ファイルはサーバーのクラスパスで使用可能である必要があります。これらのファイルのターゲットとなる場所は、ターゲットプラットフォームによって異なります。

以下に `crypto.properties` の例を示します。

```
org.apache.ws.security.crypto.merlin.keystore.password=pw
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.file=ae.keystore
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.alias=myadmin
```

[「SAML で保護されたサービスの認証の設定」 \(ページ 504\)](#) も参照してください。

SAML で保護されたサービスの認証の設定

プロセスサーバーは、ID プロバイダ（アサーションのプロデューサ）とサービスプロバイダの間で認証および認証データを交換するための Security Assertions Markup Language（SAML）標準をサポートしています。

SAML ポリシーアサーションを使用するようにプロセスサーバーを設定するには、次の手順を実行する必要があります。

- BPEL プロセスのプロセスデプロイメント記述子（PDD）で、SAML 認証されたメッセージを送受信するマイロールやパートナーロールの SAML ポリシーアサーションを追加します。このポリシーは、サービスがアクセス制御の決定を行うために使用するパラメータについて説明します。詳細については、「*Process Developer* のヘルプ」の「SAML」を参照してください。
- [第 32 章、「セキュリティ構成」 \(ページ 503\)](#)に記載されているように、SAML プロパティセクションをプロセスサーバーの **crypto.properties** ファイルに追加します。

追加する SAML プロパティの例を以下に示します。

```
org.apache.ws.security.saml.issuer.key.name=aeadmin
org.apache.ws.security.saml.issuer.key.password=password
org.apache.ws.security.saml.issuer=http://www.abe-saml-demo.com/saml
org.apache.ws.security.saml.subjectNameId.qualifier=http://www.abe-saml-demo.com/saml
```

キー名とパスワードは、プロセスサーバーのキーストアで使用しているものと一致する必要があることに注意してください。

プロセスサーバーコンポーネントの保護

用意されたセキュリティロールを設定することで、ユーザーのグループにプロセスサーバーにアクセスするためのアクセス許可を指定できます。

注: 1 つのロールの設定が必要になります。プロセスセントラルへのアクセスに必要となるのは abTaskClient です。残りのロールは、オプションとして config-deploy のプロセス中に構成することができます。ただし、マルチテナント機能のライセンスをお持ちの場合は、セキュリティを設定する必要があります。

プロセスサーバーコンポーネントを保護する手順

1. config-deploy ユーティリティ（インストールユーティリティ）を実行し、アプリケーションをすでにインストールしている場合は、[セキュリティ] ページに移動します。
2. [セキュリティ] ページで、保護するコンポーネントのチェックボックスを選択します。
 - 管理機能。この設定により、プロセスコンソールおよびデプロイ済みのプロセスにアクセスするための 3 つのレベルの認証済みユーザーを構成します。
 - プロセスサービス。この設定により、デプロイ済みのプロセスのプロセスインスタンスを開始するように認証されたユーザーを構成します。
 - プロセスサーバー ID サービスコンシューマ。この設定により、Process Developer の認証済みユーザーが、プロセスサーバーコンソールで設定した ID サービス（メンバーディレクトリ）を開いて使用できるように構成します。ID サービスは、さまざまな種類のプロセスで 사용되는リソースです。
3. config-deploy のインストールを完了します。セキュリティのみを設定する場合は、以前のインストールにあった他のすべての設定がそのまま使用されることに注意してください。
4. 次の表で、セキュリティロールの定義を確認してください。

5. 必要に応じてユーザーとグループにロールを割り当て、アプリケーションサーバーに対してそれらのロールを使用するように指定します。[「プロセスサーバーのセキュリティロールを操作するためのアプリケーションサーバーの設定」](#) (ページ 506) を参照してください。

プロセスサーバーのセキュリティロール

次のセクションでは、プロセスセントラル、プロセスコンソール、およびデプロイ済みのプロセスを保護するために使用するロールについて説明します。

管理機能

次のロールを設定し、これらの機能によって ActiveVOS コンソールおよびサービスにセキュリティパラメータを追加します。

abAdmin

このロールに関連付けられたユーザーは、ActiveVOS サーバーに対する完全な管理者権限を持ちます。

abBusinessManager

このロールに関連付けられたユーザーは、プロセスインスタンスの詳細にアクセスできます（ただし、インスタンスを操作することはできません）。このユーザーは、アクティブなプロセスとタスク、および作業キューを監視できます。読み取り専用の「プロセスインスタンスの詳細」ビューを使用できます。

abDeployer

このロールに関連付けられたユーザーは、ActiveVOS サーバーに対するビジネスプロセスアーカイブファイルのデプロイの実行のみが可能な権限を持ちます。

abDeveloper

このロールに関連付けられたユーザーは、サービスアーティファクト、エンドポイント情報、およびデプロイ後にコンシュームや公開をするサービス（つまり、プロセス）のサンプルメッセージのみに制限された権限を持ちます。開発者には、プロセスのデプロイメントアーカイブをデプロイし、プロセスの実行を開始して分析する能力が必要とされます。また、開発者は、カスタム関数、URN マッピング、およびプロセスの実行をスケジュールする機能のグローバル関数コンテキストを設定する必要があります。具体的には、このユーザーは、[アクティブプロセス] リスト、[プロセスインスタンス] ビュー、[アクティブタスク] リストと [ワークキュー] リスト、サーバーログ、ダッシュボードとすべてのレポート、およびカタログのコンテンツにアクセスできます。

abOperator

このロールに関連付けられたユーザーには、システムの操作のみが可能な権限を持ちます。これには、プロセスの機能の監視、[プロセスインスタンスの詳細] ビューを使用したプロセスインスタンスの管理、レポートの実行、ロギング、例外の表示、サービス操作に関する情報の取得、テナントの追加と削除、およびスケジュールされたデータベースの削除スケジュールの管理が含まれます。

abTaskClient

必須。すべてのユーザーに対して ActiveVOS Central にアクセスするための権限を設定する必要があります。さらに、ヒューマンタスク (WS-HT) API を操作するユーザーには、このロールが必要です。

ActiveVOS Central では、ログインページが表示されます。

プロセスサービス

プロセスサービスは、ロールを持つすべてのデプロイ済みの BPEL サービスの Web サービスハンドラにセキュリティパラメータを追加します。http://[ホスト]:[ポート]/active-bpel/services にリストされるサービスは保護されています。プロセスサービス（ロール）は次のとおりです。

abRestrictedServiceConsumer

このロールに関連付けられたユーザーは、許可されたロール（PDD で指定）でデプロイされ、ユーザーがこれらのロールに属さない場合はサービスにアクセスできません。PDD でロールが指定されていない場合、

PDD でロールが指定されていないサービスへのアクセスも拒否されます。このロールのユーザーは、**abServiceConsumer** などの他のサービスの wsdl ファイルを表示できます（ただし、実行時にブロックされます）。

abServiceConsumer

このロールに関連付けられたユーザーは、ActiveVOS Central、Eclipse Web Tools Project、または SOAPUI などの他のクライアントアプリケーションからのプロセスを含め、デPLOYされたプロセスのプロセスインスタンスの開始のみが可能な権限を持ちます。

abTenantAdmin

（マルチテナントライセンスサーバーの場合のみ。）このロールに関連付けられたユーザーは、サーバー上で設定されたテナントにコントリビューションをデPLOYおよび管理することがのみが可能な権限を持ちます。

ActiveVOS サーバー管理者（abAdmin ロールを使用）によって設定されたテナント定義に基づいて、テナント管理者ユーザーは、サーバー上のテナントコンテキストにログインできます。サービスコンシューマユーザーは、テナントコンテキストにデPLOYされたプロセスのプロセスインスタンスを作成できます。

ID サービスコンシューマ

ID サービスコンシューマは、以下のロールを使用して、ActiveVOSCentral アプリケーションによって使用されるプロセス ID サービスの Web サービスハンドラにセキュリティパラメータを追加します。

abIdentityListConsumer

このロールまたは **abAdmin** に関連付けられたユーザーのみが、Process Developer から ID サービスに Web サービス要求を送信する権限を持っています。

プロセスサーバーのセキュリティロールを操作するためのアプリケーションサーバーの設定

プロセスサーバーへのセキュアなアクセスの設定には、次の手順が含まれます。

- [「プロセスサーバーコンポーネントの保護」](#)（ページ 504）に記載されているように、プロセスサーバーの config-deploy ユーティリティを実行して、セキュリティオプションを選択します。プロセスサーバーをデPLOYすると、セキュリティロールが使用できるようになります。
- ユーザーとグループにロールをマッピングすることで、プロセスサーバーのセキュリティロールを使用するようにアプリケーションサーバーを設定します（以下で説明）。
- 認証および承認方法に関して、アプリケーションサーバーのセキュリティの詳細を設定します（以下で説明）。

ユーザーとグループへのロールのマッピング

各アプリケーションサーバーには、デPLOYしたアプリケーションのセキュリティを設定するためのさまざまな手順が用意されています。必要なセキュリティメソッドを設定する方法を理解するためには、アプリケーシ

ョンサーバー環境に関する知識が必要です。ただし、ここでは一般的な一部のガイドラインおよびドキュメントへのリンクを記載しています。

アプリケーションサーバー	設定のガイドライン
Tomcat	<p>server.xml に 領域を追加し、ユーザー名、パスワード、およびユーザーロールの既存の「データベース」に接続します。</p> <p>以下の例を参照してください。</p> <p>LDAP ベースのデータベースの場合、プロセスサーバーのセキュリティロールをデータベースに追加し、それらのロールをグループにマッピングします。</p> <p>次の URL (Tomcat 6.0) にある「<i>Realm Configuration HOW-TO</i>」を参照してください。 http://tomcat.apache.org/tomcat-7.0-doc/realms-howto.html</p>
JBoss	<p>https://docs.jboss.org/author/display/AS71/Admin+Guide を参照してください</p> <p>選択した構成ファイルにセキュリティドメインとログインモジュールを追加します。セキュリティドメインには、プロセスサーバーの config-deploy ユーティリティで選択したものと同名を付ける必要があります。ユーティリティで名前を変更していない場合、この名前はデフォルトで「ActiveVOS」になります。次のいずれかにロールを追加します: ユーザー/ロールファイルのセット、LDAP ベースのデータベース、または JDBC ベースのデータベース。</p>
WebLogic	<p>まだ行っていない場合は、WebLogic コンソールにログインし、LDAP プロバイダを設定します (セキュリティ領域に移動します。デフォルトの領域で、[プロバイダ] > [認証] > [デフォルトの認証システム] を選択し、制御フラグを [必須] から [オプション] に変更します。LDAP プロバイダを追加し、設定します)。</p> <p>詳細については、http://download.oracle.com/docs/cd/E13222_01/wls/docs103/secmanage/atn.html の 認証プロバイダの設定を参照してください。</p> <p>セキュリティ領域で、[ロールとポリシー] に移動します。[グローバルロール]を展開し、[ロール] を選択して、プロセスサーバーのセキュリティロールを入力します。</p> <p>詳細については、http://download.oracle.com/docs/cd/E12840_01/wls/docs103/secwlrsecroles.html の「ユーザー、グループ、およびセキュリティロール」を参照してください。</p>
WebSphere	<p>WebSphere コンソールにログインし、次の例のようなリンクをたどります。</p> <p>[アプリケーション] > [アプリケーションタイプ] > [WebSphere エンタープライズアプリケーション] > [プロセスサーバー]</p> <p>ユーザー/グループマッピングへのセキュリティロールグループを探します。既存のプロセスサーバーのセキュリティロールをグループにマッピングします。</p> <p>注: WebSphere アプリケーションのセキュリティが正しくセットアップされていることを確認する必要があります。WebSphere Console のナビゲーション領域で、[セキュリティ] > [セキュアな管理、アプリケーション、およびインフラストラクチャ] を選択します。次に、[アプリケーションセキュリティを有効にする] の横にあるチェックボックスを選択し、[適用] を選択します。</p>

Tomcat の例

tomcat\conf\server.xml (ファイルベースの設定)

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase"/>
    tomcat\conf\tomcat-users.xml file:
    <role rolename="abTaskClient"/>
    <role rolename="abServiceConsumer"/>
    <role rolename="abAdmin"/>
    <user username="admin" password="admin"
        roles="abAdmin, abTaskClient, abServiceConsumer"/>
```

tomcat\conf\server.xml (LDAP ベースの設定)

```
<Realm className="org.apache.catalina.realm.JNDIRealm" debug="99"
  connectionName="uid=ldapadmin,ou=system"
  connectionPassword="password"
  connectionURL="ldap://<LDAP_Server_name>:<Port>"
  userSubtree="true"
  userBase="ou=users,o=xyzuser"
  userSearch="(uid={0})"
  roleSubtree="true"
  roleBase="ou=groups,o=xyzrole"
  roleName="cn"
  roleSearch="(uniqueMember={0})"
/>
```

JBoss の例

注: 次の 2 つの例は、JBoss 7.1.1 に固有のもので (他のバージョンでは、異なる構成が必要です)。また、JBoss を設定する場合の例を示すためにここに表示しています。実際に宣言する内容は、状況に応じて異なります。JBoss の各バージョンについては、JBoss のドキュメントを確認する必要があります。

[パス]\configuration\file.xml (ファイルベースの設定)

入力するファイル名は、インストールに固有のものになります。この中に含まれる唯一の名前部分は「configuration」です。例を示します: C:\servers\jboss-as-7.1.1.Final\standalone\configuration\standalone-full.xml

```
<security-domain name="ActiveVOS" >
  <authentication>
    <login-module code="RealmUsersRoles" flag="required">
      <module-option name="usersProperties"
        value="${jboss.server.config.dir}/application-users.properties"/>
      <module-option name="rolesProperties"
        value="${jboss.server.config.dir}/application-roles.properties"/>
      <module-option name="realm" value="ApplicationRealm"/>
      <module-option
        name="unauthenticatedIdentity"> anonymous
      </module-option>
    </login-module>
  </authentication>
</security-domain>
```

```
${jboss.server.config.dir}/application-roles.properties
admin=abTaskClient,abAdmin,abServiceConsumer
```

```
${jboss.server.config.dir}/application-users.properties
admin=admin
```

[パス]\configuration\[パス] (LDAP ベースの設定)

入力するファイル名は、インストールに固有のものになります。この中に含まれる唯一の名前部分は「configuration」です。例を示します: C:\servers\jboss-as-7.1.1.Final\standalone\configuration\standalone-full.xml。

```
<security-domain name="ActiveVOS">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
      flag="optional">
      <module-option name="java.naming.provider.url"
        value="ldap://myserver:3268"/>
      <module-option name="bindDN"
        value="CN=admin,CN=Users,DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="bindCredential" value="admin"/>
      <module-option name="baseCtxDN"
        value="DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="baseFilter"
        value="(sAMAccountName={0})"/>
      <module-option name="rolesCtxDN"
        value="DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="roleFilter" value="(member={1})"/>
    </login-module>
  </authentication>
</security-domain>
```

```
<module-option name="roleAttributeID" value="memberOf"/>
<module-option name="roleAttributeIsDN" value="true"/>
<module-option name="roleNameAttributeID" value="cn"/>
<module-option name="roleRecursion" value="2"/>
<module-option name="searchScope" value="SUBTREE_SCOPE"/>
<module-option name="allowEmptyPasswords" value="true"/>
<module-option name="java.naming.referral" value="follow"/>
<module-option name="unauthenticatedIdentity" value="aeadmin"/>
</login-module>
</authentication>
</security-domain>
```

パート V: プロセスセントラルとプロセスサーバー（オンプレミス）

このセクションのトピックでは、オンプレミスのプロセスサーバーで実行され、プロセスセントラルで表示されるレポートとフォームを作成する方法を説明します。

第 33 章

プロセスセントラルフォームおよび設定

次のトピックでは、プロセスセントラルのフォームと設定について説明します。

- [「プロセスセントラルについて」 \(ページ 511\)](#)
- [「プロセスセントラルプロセスフォームの作成」 \(ページ 512\)](#)
- [「プロセスセントラルフォームのテストとデバッグ」 \(ページ 519\)](#)
- [「プロセスセントラル構成ファイルの作成」 \(ページ 520\)](#)
- [「フォームレポートと構成ファイルのデプロイ」 \(ページ 531\)](#)
- [「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)
- [「プロセスセントラルの詳細設定」 \(ページ 536\)](#)

プロセスセントラルについて

プロセスセントラルは、プロセス要求フォーム、タスク、およびレポートを含むクライアントアプリケーションです。プロセスセントラルのユーザーは、プロセスを開始し、(ユーザープロセス用の BPEL から) ヒューマンタスクを操作して、タスクに関連するレポートを表示できます。Process Developer では、フォーム、タスク、およびレポートを作成してデプロイします。

サーバーの実行中は、次の URL で Process Developer のプロセスセントラルにアクセスできます。

`http://localhost:8080/activevos-central`

Process Developer のインストール中に host:port を変更した場合、host:port が異なるものとなっている可能性があることに注意してください。

【ログイン】 ページ (Process Developer で実行中のサーバーのみ) で、次のユーザー名とパスワードを入力します。

ユーザー名: **manager**

パスワード: **manager**

このユーザー名は、Process Developer の組み込みサーバーで実行中の有効な ID サービスで定義されています。デフォルトでは tomcat-users.xml ファイルが有効になっており、この名前またはリストされている *abTaskClient* ロールを持つ別のユーザー名でログインすることができます。tomcat-users.xml ファイルはプロセスサーバーに対して有効化されていません。

ユーザーは開発者として、プロセスセントラルで次のような操作を実行できます。

- プロセスセントラルのナビゲーション領域をカスタマイズし、デプロイされるフォーム、タスク、およびレポートを記述した設定ファイルを作成する。
[「プロセスセントラル構成ファイルの作成」 \(ページ 520\)](#)を参照してください。
- ユーザーがフォームを送信してプロセスを開始できるように、プロセス要求フォームを作成する。
[「プロセスセントラルプロセスフォームの作成」 \(ページ 512\)](#)を参照してください。
- レポートを作成する。
[第 34 章, 「プロセスサーバーおよびプロセスセントラルレポート」 \(ページ 538\)](#)を参照してください。
- ユーザーアクティビティのタスクに対して HTML レンダリングを作成する。
このヘルプの「ヒューマンタスク」セクションにある「タスククライアントに対するレンダリングの作成」を参照してください。

次の図は、プロセスセントラルの例を示しています。左側の [タスク]、[フォーム]、および [レポート] バーを上下にスライドして、表示を変更できることに注意してください。

The screenshot displays the Process Central web application interface. On the left is a navigation sidebar with tabs for 'Home', 'Tasks', 'Forms', 'Tutorial and Samples', and 'Reports'. The 'Forms' tab is currently selected. The main content area shows a table of forms with columns 'Name' and 'Description'. Two forms are listed: 'Human Approval Completed Form' and 'Tutorial Request Form'. Below the table, there are pagination controls showing 'Showing 1 to 2 of 2 entries' and buttons for 'First', 'Previous', '1', 'Next', and 'Last'. Below the table, a section titled 'Request Process Request' is expanded, showing a detailed form titled 'Loan Process Request'. This form contains several input fields: 'Loan Type', 'First Name', 'Last Name', 'Day Phone', 'Night Phone', 'Social Security Number', 'Amount Requested', and 'Loan Description'. Each field has a red asterisk indicating it is required. At the bottom of this section is a 'Send Request' button.

プロセスセントラルプロセスフォームの作成

プロセスへの要求メッセージの送信時に使用される HTML フォームを作成するための操作を選択します。プロセス要求フォームのデフォルトの要求フォルダとデフォルトのファイル名を使用するか、新しい名前を入力できます。

このトピックの概要については、[「プロセスセントラルについて」 \(ページ 511\)](#)を参照してください。

プロセス要求フォームは、Receive または Pick アクティビティの開始（インスタンスの作成）に対する入力メッセージから生成された HTML ファイルです。フォームをサーバーにデプロイすると、ユーザーはフォームを送信してプロセスセントラルからプロセスを開始できます。

プロセス要求フォームを作成する手順

1. プロセス要求フォームを作成する前に、プロセスデプロイメント記述子 (PDD) を作成して、ターゲットとするサービスを事前に確認できるようにすることをお勧めします。サービス名 (つまり、PDD のマイロールのサービス名) が不明な場合は、[「プロセス要求フォームテンプレートの理解」 \(ページ 513\)](#)に記載されているように、サービス名をフォームに手動で追加する必要があります。
2. **[ファイル] > [新規] > [プロセス要求フォーム]** を選択します。
3. プロセスの開始アクティビティで定義されているポートタイプと操作を選択します。
4. フォームのデフォルトの場所とファイル名を使用するか、それらを変更します。デフォルトのファイル名は操作名です。
5. フォームに多言語サポートを設定するは、**[詳細]** を選択します。詳細については、[「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)を参照してください。

関連事項：

- [「プロセス要求フォームテンプレートの理解」 \(ページ 513\)](#)
- [「プロセスセントラルフォームでの HTML の編集」 \(ページ 514\)](#)
- [「フォームまたはタスクへの新しいサービス操作の追加」 \(ページ 515\)](#)
- [「タスクおよびフォームスクリプトのカスタマイズの概要」 \(ページ 516\)](#)
- [「フォームをカスタマイズするための Informatica Business Process Manager SDK の使用」 \(ページ 519\)](#)
- [「プロセスセントラル用の独自のスタイルのスクリプトとメタデータを含める場合」 \(ページ 523\)](#)

プロセス要求フォームテンプレートの理解

ユーザーがプロセス要求フォームを送信する場合、サーバーはその要求を PDD のサービス名と照らし合わせます。フォームに表示されたサービス名が、PDD (プロセスの開始アクティビティ用) のマイロールサービス名と一致していることを確認してください。

生成されたフォームでサービス名のコード行を見つけて、名前を確認します。

```
<script type="text/javascript">
//
  // request form to submit requests and handle response
  var AeRequestForm$ID = function() {
    // this
    var mFormInstance = this;
    // the service name
    var mServiceName = "EstimationService";</pre></div><div data-bbox="188 671 320 686" data-label="Section-Header"><h3>追加の変更を加える</h3></div><div data-bbox="188 693 892 724" data-label="Text"><p>プロセス要求フォームに追加の変更を加える場合は、フォームの作成に使用するテンプレートと、フォームから生成される HTML ファイルについて理解しておくことが便利です。</p></div><div data-bbox="188 731 362 747" data-label="Section-Header"><h3>フォームの視覚的なパート</h3></div><div data-bbox="188 754 901 901" data-label="List-Group"><ul><li>• <b>タイトル</b><br/>デフォルトでは、タイトルは Receive アクティビティのインタフェースの操作名に基づいています。これは、HTML ページの一番上のテーブルにあります。</li><li>• <b>入力メッセージパート</b><br/>メッセージが単一パートの要素である場合、子要素ごとに 1 つの行を含む 1 つのテーブルがあります。要素ごとに個別のテーブルを持ちます。要素が繰り返されている場合は、<b>[追加]</b> と <b>[削除]</b> という 2 つのコマンドボタンが表示されます。</li><li>• <b>[要求コマンドの送信] ボタン</b><br/>フォームの下部に、要求を送信してプロセスを開始するためのコマンドボタンが表示されます。</li></ul></div><div data-bbox="573 947 903 963" data-label="Page-Footer"><p>プロセスセントラルプロセスフォームの作成 513</p></div>
```

フォームを変更する手順については、[「プロセスセントラルフォームでの HTML の編集」 \(ページ 514\)](#)および[「フォームまたはタスクへの新しいサービス操作の追加」 \(ページ 515\)](#)を参照してください。

多言語サポートについては、[「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)を参照してください。

フォームのスク립トセクション

スク립トには、プロセスセントラルでフォームを動的にフォーマットおよび表示するためのコマンドが含まれています。スク립トは JavaScript であり、JavaScript Object Notation (JSON) と jQuery 構文を含み、フォーム内およびトランスポートプロトコルで機能を提供します。詳細については、[「タスクおよびフォームスク립トのカスタマイズの概要」 \(ページ 516\)](#)を参照してください。

プロセスセントラルフォームでの HTML の編集

プロセスセントラルの HTML 要求とタスクを作成するには、Eclipse Web ページエディタを使用します。

プロセスセントラルフォームを作成すると、作成したフォームが Web ページエディタに表示されます。次のようにフォームを簡単に編集することができます。

- エンドユーザー向けの手順を追加します。
<p> </p>や などの HTML タグを追加して、フォームの使用方法に関する詳細を含めます。
- インラインスタイル、画像、リンクを使用して、使いやすいフォームの外観と機能を整えます。
<href>やなどの一般的な HTML リンクを追加します。画像は、ビジネスプロセスアーカイブ (BPR) ファイル内のリソースとしてプロセスサーバーにデプロイすることができます。詳細については、[「フォームレポートと構成ファイルのデプロイ」 \(ページ 531\)](#)を参照してください。
- Web ページエディタに追加されていない多言語サポート (国際化) を追加します。[「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)を参照してください。
- 複数の入力および/または出力のフォームにサービス操作を追加します。[「フォームまたはタスクへの新しいサービス操作の追加」 \(ページ 515\)](#)を参照してください。

Web ページエディタについて

Process Developer には、要求フォームとタスクフォームを編集するための Eclipse Web ページエディタが含まれています。Process Developer の設定では、Web ページエディタでプロセスセントラルのスタイルシートを含めるという設定が有効になっていることに注意してください。スタイルシートには、プロセスセントラルのフォントと背景のスタイルが含まれています。

Web ページエディタでの編集

Web ページエディタには、[デザイン] と [ソース] という 2 つの編集パネルが含まれています。[デザイン] パネルにはフォームがグラフィカルモードで表示され、[ソース] ペインには HTML タグと JavaScript 関数が表示されます。

ツールバーボタンを使用すると、パネルの表示/非表示の変更や、パネルの水平または垂直分割ができます。さらに、テキスト要素に太字や斜体などの単純な書式を適用できます。

[アウトライン] ビューは、編集パネルで構成を選択する場合に役立ちます。

プロセスセントラルでフォームがどのように表示されるかを確認するには、[プレビュー] タブを選択します。プレビューには、段落、表、コントロールボタン、およびその他の構成要素の配置と外観が表示されますが、コマンドボタンは表示専用です。

[デザイン] モードでは、HTML パレットを使用して、テキストフィールド、画像、線などの構成を追加します。

関連事項:

- [「フォームまたはタスクへの新しいサービス操作の追加」 \(ページ 515\)](#)

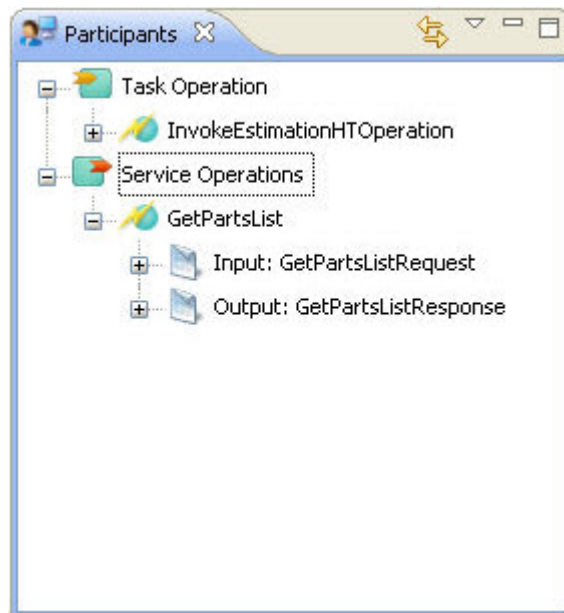
- [「タスクおよびフォームスクリプトのカスタマイズの概要」 \(ページ 516\)](#)
- [「フォームをカスタマイズするための Informatica Business Process Manager SDK の使用」 \(ページ 519\)](#)
- [「プロセスセントラルフォームのテストとデバッグ」 \(ページ 519\)](#)

フォームまたはタスクへの新しいサービス操作の追加

プロセスへの要求メッセージの送信時に使用される HTML フォームを作成するための操作を選択します。また、プロセス要求またはタスクフォームに新しいサービスを追加するための操作を選択することもできます。

必要に応じて、サービス操作から生成されたフィールドと関数を追加することで、既存のフォームまたはタスクフォームを変更できます。サービス操作とは、プロセスサーバーで実行中の BPEL プロセスのエンドポイントを表すワークスペース WSDL からの WSDL 操作です。この操作には、サービス名が関連付けられている必要があります。

タスクまたはフォームを Web ページエディタで開くと、通常はプロセスパーティシパントが表示される [パーティシパント] ビューには、図に示すように、フォームで使用可能なサービス操作のみが表示されます。



新しいサービス操作は、複数の方法で追加できます。一部の例を以下に示します。

- 既存のタスクまたはフォームがあり、タスクまたはサービス操作のスキーマが更新されている場合は、フィールドをフォームにドラッグアンドドロップすることで、既存のフォームを変更したり、既存のフォームにフィールドを追加することができます。
- サービスを呼び出すことで、フォームフィールドの機能を拡張します。例えば、フォームフィールドが「国の選択」である場合、国名のリストを提供するサービスを呼び出します。

使用する操作を含んだデプロイ済みプロセスがある場合は、WSDL サービス定義をワークスペースにインポートする必要があります。


ワークスペースに WSDL を追加する手順

1. プロセスコンソールで、[カタログサービスの定義] ページを表示します。
2. デプロイ済みプロセスのサービス定義を選択して、サービスの WSDL を表示します。
3. URL をコピーします。例: `http://localhost:8080/active-bpel/services/loanservice?wsdl`
4. プロジェクトで、[サービス参照] を右クリックします。

5. **【インポート】** を選択します。
6. **【インポート元の URL を入力します】** のフィールドに URL を貼り付け、**【完了】** を選択します。

新しいサービス操作を追加する手順

1. Web ページエディタでタスクまたはフォームを開きます。
2. **【パーティシパント】** ビューで、**【サービス操作】** を右クリックし、**【サービス操作の追加】** を選択します。
3. **【サービス操作の選択】** ダイアログで、ポートタイプと操作を選択します。
4. プロセス PDD ファイルにサービス名が存在しない場合、サービス名が不明である可能性があることに注意してください。[「プロセス要求フォームテンプレートの理解」 \(ページ 513\)](#)に記載されているように、必ずサービス名を追加してください。名前が不明な場合は、次のようにダイアログに通知が表示されます。



Namespace: http://VintageOldStock.com
Interface: InvestigateResponseTimes
Operation: Investigate
Service: [Replace with a service name]

新しいサービス操作を追加したときに発生するイベント。

- 新しい操作用に、HTML フォームの下部に **【要求の送信】** ボタンが追加されます
- 新しい JavaScript セクションがソースファイルに追加されます。これは、サービス名を指定する必要があるセクションです。
- 操作および入出力メッセージが **【パーティシパント】** ビューに追加されます

フォームで新しいサービス操作を使用する手順

1. Web ページエディタでフォームがフォーカスされていることを確認します。
2. **【パーティシパント】** ビューで、操作を展開して、入力メッセージまたは出力メッセージを表示します。
3. 操作の入力/出力メッセージ、メッセージのパート、またはパートの要素を **【デザイン】** ビューの場所にドラッグします。

フォームに多言語サポートを追加する場合は、新しいフィールドに多言語サポートを手動で追加し、プロパティファイルに言語値を追加する必要があります。フォームを手動で更新する方法に関する詳細については、[「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)を参照してください。

関連事項：

- [「プロセスセントラルフォームでの HTML の編集」 \(ページ 514\)](#)
- [「タスクおよびフォームスクリプトのカスタマイズの概要」 \(ページ 516\)](#)
- [「フォームをカスタマイズするための Informatica Business Process Manager SDK の使用」 \(ページ 519\)](#)

タスクおよびフォームスクリプトのカスタマイズの概要

タスクおよびフォーム用に生成された HTML フォームは、XML メッセージデータを HTML にレンダリングする場合の基本的な要件を満たしています。さらに、フォームを使用すると、Process Developer がフォームの作成に使用する複数の HTML ベースのテクノロジーを利用して、より詳細なカスタマイズが可能になります。

- JavaScript、非同期 JavaScript、および XML (AJAX)
- JavaScript Object Notation (JSON)
- jQuery と jQuery の UI

これらのテクノロジーを組み合わせることで、動的なフォームとタスクが提供されます。フォームには、次のような開発機能があることに注意してください。

- 標準ベースのソリューションを使用します
 - 任意の HTML エディタで作業できます
 - 日付選択画面、タブ、チェックボックス、コンボボックスなど、タスクおよびフォームのメッセージデータに対するさまざまなレンダリング機能が提供されます
 - 追加のサービス、画像、および Web アプリケーションプロジェクト用としてスクリプト化が可能なほぼすべての呼び出しを含む、各種の追加操作が可能です
 - 大規模な技術リソース、ナレッジベース、および Web アプリケーション開発者へのアクセスを提供します
- タスクとフォームの作成に使用される各テクノロジーの詳細を以下に示します。

JavaScript と AJAX

JavaScript はフォームのバインディングで、JSON と jQuery のサブセットが含まれています。AJAX は、フォームとタスクの表示と動作を妨げることなく、バックグラウンドで非同期的にサービスを呼び出します。

JSON

JavaScript Object Notation (JSON) は、データ交換形式です。フォームまたはタスクからの XML メッセージは JSON オブジェクトに変換されます。Process Developer による JSON の実装は、Google Data Protocol 内にあります。JSON の詳細については、<http://code.google.com/apis/gdata/json.html> を参照してください。

Google Data Protocol には次のようなルールがあります。

- XML は JSON オブジェクトとして表されます。
- XML のバージョン属性とエンコーディング属性は、それぞれルート要素のバージョン属性とエンコーディング属性に変換されます。
- 要素に名前空間エイリアスがある場合、エイリアスと要素は「\$」を使用して連結されます。例えば、`ns:element` は `ns$element` になります。
- ネストされた各要素や各属性は、オブジェクトの名前/値プロパティとして表されます。
- 属性は文字列プロパティに変換されます。
- 子要素はオブジェクトプロパティに変換されます。
- 複数回出現する可能性のある要素は、配列プロパティに変換されます。
- タグのテキスト値は `$t` プロパティに変換されます。

XML から JSON への変換の例を以下に示します。

XML メッセージ:

```
<?xml version="1.0" encoding="UTF-8"?>
<types1:EstimationRequest
  xmlns:ns="http://example.org/QuoteRequest.xsd"
  xmlns:ns1="http://example.org/QuoteRules.xsd"
  xmlns:types1="http://example.org/Estimation.xsd">
  <ns:refNumber>1</ns:refNumber>
  <ns:quoteRequest>
    <ns:contact>
      <ns:name>JZ</ns:name>
      <ns:email>activevos@gmail.com</ns:email>
    </ns:contact>
    <ns:model>500</ns:model>
    <ns:year>1983</ns:year>
    <ns:description>a fine car</ns:description>
  </ns:quoteRequest>
</types1:EstimationRequest>
```

JSON オブジェクト:

```
{ "EstimationRequest": { "xmlns": "http://\\example.org\\  
/Estimation.xsd",  
  "refNumber": { "xmlns": "http://\\example.org\\  
/QuoteRequest.xsd", "t": "1"},  
  "quoteRequest": { "xmlns": "http://\\example.org\\  
/QuoteRequest.xsd",  
    "contact": { "name": { "t": "JZ"},  
      "email": { "t": "activevos@gmail.com"} },  
    "model": { "t": "500"},  
    "year": { "t": "1983"},  
    "description": { "t": "a fine car" } } } }
```

ヒント: フォームがプロセスセントラル (JSON + JavaScript) で機能するようにするには、フォーム要素名が JavaScript 変数の定義に使用されるルールと同じルールに従っている必要があります。例えば、変数には、文字、数字、またはアンダースコアのみを含めることができます。また、要素名には予約済みの JavaScript キーワードを使用することはできません。

jQuery と jQuery の UI

jQuery は、HTML ドキュメントのトラバース、イベント処理、アニメーション化、および Ajax インタラクションを簡素化する JavaScript ライブラリです。jQuery は、ページ上の特定の要素を選択して操作するために構築されています。jQuery の UI には、コアインタラクションプラグインやさまざまな UI ウィジェットなどのビジュアルコントロールが含まれています。

詳細については、<http://jquery.com/>を参照してください

jQuery の例

jQuery の選択プロセスはフィルタリングプロセスで、ページ上のすべての要素はコマンドで指定したフィルタを通過し、このコマンドは一致する単一のオブジェクトまたは一致するトラバース可能なオブジェクトの配列を返します。

例えば、ページ内の一致するすべての HTML 要素の配列を取得するには、次のように HTML タグを jQuery 検索フィールドに渡します。

```
// Create a red background for every <p> tag in the page.  
$("p").css("background", "#ff0000");
```

フォームでは、プロセスサーバーが jQuery を使用してデータ要素を選択します。次の例では、jQuery コマンドが要素を繰り返すようにするためのテーブルを作成します。ユーザーが新しい行を追加する場合は、次のようなコードを使用します。

```
// Handles the adding of a new repeating field  
$("#processRequestForm$ID").find("table.avc_repeating_field_data_table  
input[type='button'].avc_add_field_element").click(function() {  
  var table = $(this).parents("table:first");  
  var rows = $(table).find("tbody tr").length;  
  var lastRow = $(table).find("tbody tr:last");  
  var newRow = $(lastRow).clone(true);  
  $(lastRow).after(newRow);  
});
```

上記の例の (#processRequestForm\$ID) のように、HTML 要素 ID 属性は \$ID サフィックスで終わる必要があります。\$ID サフィックスは、他のフォームとの競合を回避するための一意の HTML 要素 ID を提供します。

関連事項:

- [「フォームをカスタマイズするための Informatica Business Process Manager SDK の使用」 \(ページ 519\)](#)
- [「プロセスセントラルフォームのテストとデバッグ」 \(ページ 519\)](#)

フォームをカスタマイズするための Informatica Business Process Manager SDK の使用

フォームとタスクのカスタマイズを最大限に活用するには、Informatica Business Process Manager SDK パッケージをダウンロードし、使用方法について理解することをお勧めします。このパッケージには、Process Developer XML-JSON およびプロセスセントラル用の Process Developer XML-JSON の詳細な説明が含まれており、フォームの開発方法と使用方法が示されています。（この説明もそのヘルプの一部です。）

Process Developer SDK の入手先

SDK パッケージおよび SDK ドキュメントキットは、Informatica InfoCenter で入手できます。最新の InfoCenter は <http://infocenter.activevos.com> から参照できます。使用する製品バージョンで利用可能なバージョンの SDK を使用する必要があることに注意してください。

Process Developer オンラインヘルプから、InfoCenter の SDK ドキュメントにリモートでリンクできます。リモートヘルプを表示する手順については、「Process Developer のヘルプの設定」を参照してください。

プロセスセントラルフォームのテストとデバッグ

1 つ以上のプロセス要求フォームとタスクフォームをデプロイした後に、ブラウザの JavaScript デバッガを使用してプロセスセントラルでデバッグを有効にすることで、JavaScript をテストおよびデバッグできます。また、JSON Converter ユーティリティを使用することもできます。

JavaScript を HTML フォームでデバッグするには、次の手順を実行します。

- Firefox 用の Firebug、Internet Explorer の組み込みデバッグ、Chrome の Developer JavaScript デバッガなど、JavaScript デバッグツールをブラウザにインストールするか、有効にします。
- プロセスセントラルでデバッグを有効にするには、次のコマンドをプロセスセントラルの URL に次のように追加します。
`http://localhost:8080/activevos-central/avc/avc.jsp?debug=true`
- JSON Converter** ツールを使用して、JSON から XML への変換およびその逆の変換を確認します。JSON 要求フォームが XML でどのように表示されるかを確認するか、サンプル XML データを生成して JSON に変換します。次の URL でコンバータにアクセスします。
`http://localhost:8080/active-bpel/jsonConverter.html`

フォームに変更を加えて再デプロイする場合は、プロセスセントラルで **【フォームのリセット】** を選択して、更新されたフォームを表示します。

デバッグを有効にすると、jQuery 関数のデバッグが容易になります。これにより、デバッグコンソールに通常表示される 1 行の長いコードではなく、読み取り可能なコード行に jQuery 関数が展開されます。

タスクフォームの場合、**【開発モード】** を使用して、フォームを再デプロイせずにレイアウトをテストすることもできます。

フォームをテストするためのヒント

- HTML フォームが整形形式の XML であることを確認してください。プロセスセントラルはフォームを検証し、無効なタグがある場合は警告が表示されます。
- スキーマで有効とみなされる小文字の要素名を使用してください。
- すべてのタグが終了タグを使用して適切に閉じられていることを確認してください。
- HTML マークアップでは、混合型はサポートされていません。例えば、XML メッセージ要素内に、`<contactName>Georgia Jane</contactName>` などの HTML タグを追加することはできません。

- HTML 要素 ID 属性は、processRequestForm\$ID などの\$ID サフィックスで終わるようにする必要があります。
\$ID サフィックスは、他のフォームとの競合を回避するための一意の HTML 要素 ID を提供します。
- in や out などの JavaScript の予約語は、JSON データパートとして許可されていません。スキーマに JavaScript の予約語の名前の要素が含まれる場合、一部のブラウザで空白のフォームが表示されることがあります。この問題を修正するには、例に示すようにフォームの JavaScript パートを変更する必要があります。
不正: `getText(parametersTaskInputPart.NewOperation.in)`。ここで、*in* は NewOperation の入力メッセージです。
修正: `getText(parametersTaskInputPart.NewOperation["in"])`

プロセスセントラル構成ファイルの作成

プロセスセントラルの Web アプリケーションには、ナビゲーション用に構成したフォーム、タスク、およびレポートが含まれています。

基本構成ファイルはプロセスサーバーにあり、ナビゲーションフィルタの初期構成を提供します。プロセスコンソールの [カタログリソース] ページからこのファイルを表示できます。

ユーザーごとまたはグループごとにナビゲーションフィルタをカスタマイズする追加の構成ファイルを作成し、デプロイするフォームとレポートを記述することもできます。

基本構成ファイルをプロセスサーバー内の独自のファイルに置き換えることもできます。基本構成ファイルのすべての要素が必要で、デフォルトでは基本ファイルがデプロイされます。変更した場合、基本設定が上書きされます。

プロセスセントラル構成 (.avcconfig) ファイルを作成する手順

- [ファイル] > [新規] > [一元的な設定] を選択します。
- ファイルのデフォルトの場所とファイル名を使用するか、それらを変更します。デフォルトの場所はデプロイフォルダで、ファイル名はプロジェクト名に基づきます。
ファイルが XML エディタで開き、ファイルの主要なセクションがコメント化されます。ファイルの編集については、以下の編集のヒントを参照してください。

構成ファイルセクションの概要:

セクション	追加可能な構成フィルタ
タスクロールフィルタ: 「タスクロールフィルタの設定」 (ページ 524) を参照してください	<ul style="list-style-type: none">- [ロール] ドロップダウンの項目- [表示] ドロップダウンの項目- [タスク] リストのフォルダ名- ユーザーとグループ (許可されたロール) に基づいてフォルダを表示または非表示にする- タスクのカスタムカラム名- [タスク] リストのフィルタリング
フォーム: 「フォームフィルタの設定」 (ページ 521) を参照してください レポート: 「レポートフィルタの設定」 (ページ 522) を参照してください	<ul style="list-style-type: none">- [フォーム] リストまたは [レポート] リストのフォルダ名- ユーザーとグループ (許可されたロール) に基づいてフォルダを表示または非表示にする

セクション	追加可能な構成フィルタ
ローカリゼーション	「プロセスセントラルへの多言語サポートの追加」 (ページ 532)を参照してください。
プロセスセントラル、フォーム、およびタスクへのコントリビューション	独自のスクリプト、CSS、およびその他の要素を作成します。 <centralIncludes>セクションでそれらを指定します。詳細については、 「プロセスセントラル用の独自のスタイルのスクリプトとメタデータを含める場合」 (ページ 523)を参照してください。

編集のヒント:

- [タスクロールフィルタ] セクションなど、作業するセクションのコメント化を解除します。
- ファイルを読みやすくするために、このファイルにデプロイしないセクションを削除し、セクションごとに個別の構成ファイルを作成します。
- フォームは有効な XML である必要があるため、必ず HTML タグを閉じてください。
- コンテンツアシスト (CTRL + スペースバー) を使用して、プロセスセントラルの構成スキーマに基づいてファイルに新しい要素を追加します。
- [アウトライン] ビューで、要素を右クリックして、状況依存の属性と子を追加します
- Process Developer インストールの avoscentral-config.xsd スキーマは次の場所に表示できます。
<installdir>designer\plugins\com.activee.avoscentral.services_N.N\schema
- ベースの.avccconfig ファイルは、プロセスコンソールの [カタログリソース] ページから表示できます (このページから、セクションをコピーしてファイルに含めることができます)。

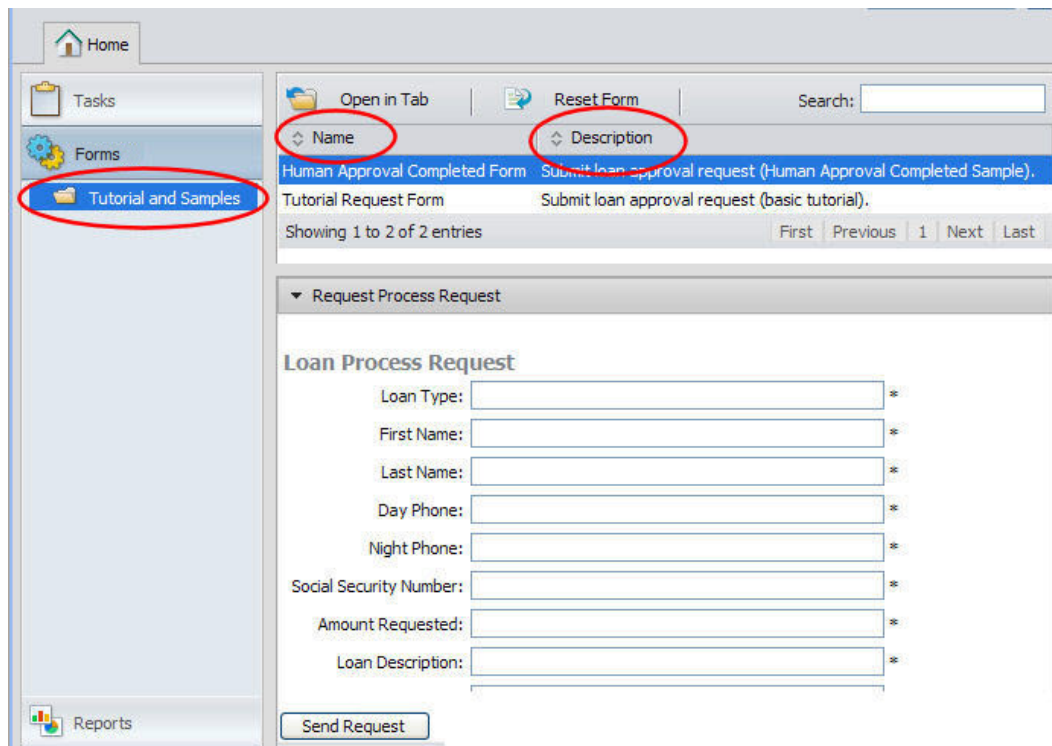
フォームフィルタの設定

フォームの概要については、[「プロセスセントラルプロセスフォームの作成」](#) (ページ 512)を参照してください。

プロセスセントラル構成ファイルの [フォーム] セクションでは、以下を設定します。

- [フォーム] カテゴリの下に追加するフォルダ名
- フォルダに追加するフォーム名
- 作業領域に表示するフォームの説明
- デプロイするフォームの場所と名前
- 名前付きユーザーロールのフォームへの制限付きアクセス

2つのフォームを含むカスタムフォルダを示すプロセスセントラルの構成の例を以下に示します。



次の例に示すように、値を入力します。

```
<tns:requestCategoryDefs>
  <tns:requestCategoryDef id="loans_category"
    name="Loan Approval">
    <avccom:requestDef id="loan_request"
      name="Customer Request Form">
    <!-- Use this section only to restrict access to requests.
      Delete it if not used.
    <avccom:allowedRoles>
      <avccom:role>FILL_IN_ROLE_1</avccom:role>
      <avccom:role>FILL_IN_ROLE_2</avccom:role>
    </avccom:allowedRoles>
    -->
    <avccom:description>
      Submit a request for loan approval.
    </avccom:description>
    <avccom:formLocation>
      project:/tutorial/form/request/loanRequest.html
    </avccom:formLocation>
    </avccom:requestDef>
  </tns:requestCategoryDef>
</tns:requestCategoryDefs>
```

レポートフィルタの設定

Process Developer でビジネスインテリジェンスとレポートツール（BIRT）レポートを設計し、プロセスセントラルおよびプロセスコンソールでそれらを表示するためにデPLOYできます。詳細については、[第 34 章](#)、[「プロセスサーバーおよびプロセスセントラルレポート」](#)（ページ 538）を参照してください。

プロセスセントラル構成ファイルの「レポート」セクションでは、以下を設定します。

- 「レポート」カテゴリの下に追加するフォルダ名。
- フォルダに追加するレポート名。
- 作業領域に表示するレポートの説明。

- デプロイするレポートの場所と名前。
ヒント: 標準のレポートを表示する場合は、プロセスコンソールを確認してください。
- [プレビュー] 領域用に作成されたオプションの HTML ファイルの場所と名前。
- 名前付きユーザーロールのレポートへの制限付きアクセス。

次の例に示すように、値を入力します。

```
<tns:reportCategoryDefs>
  <tns:reportCategoryDef id="loans_category"
    name="Loan Approval">
    <avccom:reportDef id="loan_report"
      name="Loan Approval Report">
<!-- Use this section only to restrict access to reports.
      Delete it if not used.
      <avccom:allowedRoles>
        <avccom:role>FILL_IN_ROLE_1</avccom:role>
        <avccom:role>FILL_IN_ROLE_2</avccom:role>
      </avccom:allowedRoles>
-->
      <avccom:description>
        Loan Application Approvals By Month
      </avccom:description>
      <avccom:formLocation>
        project:/tutorial/reports/loansReport.rptdesign
      </avccom:formLocation>
      <avccom:formPreview>
        project:/tutorial/form/preview/loanRptPreview.html
      </avccom:formPreview>
    </avccom:reportDef>
  </tns:reportCategoryDef>
</tns:reportCategoryDefs>
```

プロセスセントラル用の独自のスタイルのスクリプトとメタデータを含める場合

新しい avcconfig ファイルには、プロセスセントラルで使用する独自の CSS ファイル、スクリプト、およびメタデータを指定できるセクションがあります。

フォームやタスクで CSS、スクリプト、メタデータをリンクする代わりに、<centralIncludes>セクションの avcconfig ファイルからそれらを参照します。

<centralIncludes>セクションには、XHTML ファイルの一般的な<head>要素である次の要素を含めることができます。

- **<script>**。要求フォームとタスクフォームで使用する関数を含む JavaScript またはその他のスクリプトへの参照を追加します。<centralIncludes>の構文は次のようになります。
<script xmlns="http://www.w3.org/1999/xhtml" | type="text/javascript" src="jquery.script.js"/>
また、<script xmlns="http://www.w3.org/1999/xhtml" type="text/javascript"> var my_String = "Hello";などのインラインスクリプトを使用することもできます。
- **<meta>**。メタデータのキーワードを追加します。構文は次のようになります。
<meta xmlns="http://www.w3.org/1999/xhtml" name="keywords" content="some keyword,another keyword" />
- **<style>**。<style xmlns="http://www.w3.org/1999/xhtml" type="text/css">p {background-color: yellow}</style>などのインラインスタイルを使用します
- **<link>**。スタイルシートのリンク。以下の例を参照してください。

新しい avcconfig ファイルを作成すると、デフォルトでは、ファイルの 1 つのセクションのみが生成されます。

```
<tns:centralIncludes>
<link xmlns="http://www.w3.org/1999/xhtml" href="../example.css" rel="stylesheet" type="text/css">
</tns:centralIncludes>
```

独自のスタイルシートを使用するに:

1. 必要に応じて、デフォルトのプロセスセントラル CSS ファイルを確認します。CSS ファイルを確認するには、プロセスセントラルを開き、ページソースを表示します。CSS ファイルにリンクしているソースファイルのセクションを見つけて、リンクを選択します。フォントと色に対する基本的な UI スタイルのスタイルシートは *ae-avc.css* です。
2. 必要に応じて、ae-avc.css または他の CSS からスタイルをコピーし、独自の変更を加えることができます。
3. フォームとタスクでは、必要に応じて、CSS ファイルのクラス名を使用します。
4. CSS ファイルをデプロイします。
5. avcconfig ファイルで <tns:centralIncludes> セクションに CSS の名前を追加し、config ファイルをデプロイします。以下に例を示します。

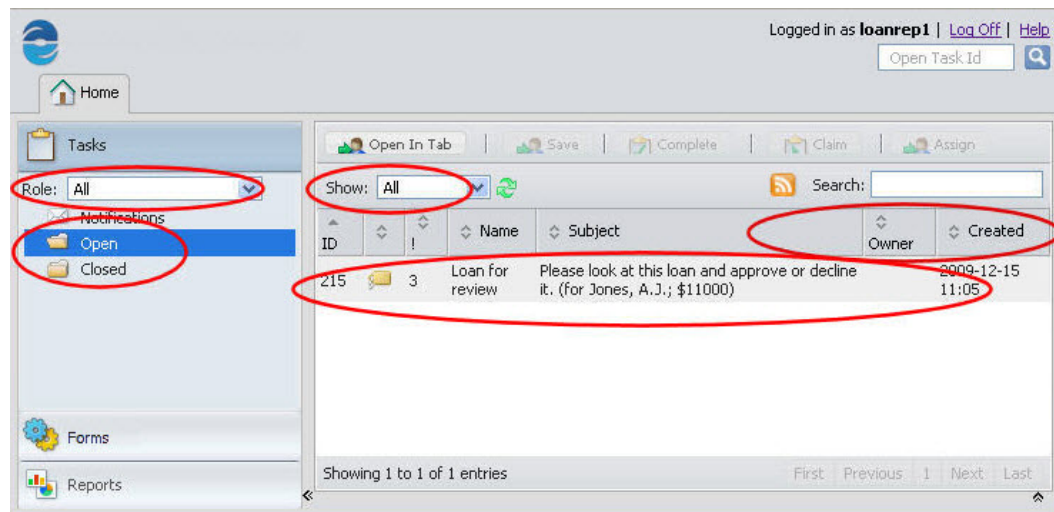
```
<link xmlns="http://www.w3.org/1999/xhtml"
      href="../example.css" rel="stylesheet" type="text/css">
```

ここで href パスは次のいずれかになります。

- プロジェクト内のこの avcconfig ファイルへの相対パス
- 完全な形式の project:/path/to/file
- http://domain:port/myCSS.css などの絶対 URL

タスクロールフィルタの設定

次の図の丸で囲んだ領域に示すように、プロセスセントラル構成ファイルの [タスクロールフィルタ] セクションでは、[タスク] ビューを包括的に設定できます。



基本構成と詳細構成を追加することができます。

基本設定

- 詳細については、「[基本的なタスクロールのフィルタ構成](#)」(ページ 525)を参照してください。
- 次の場合、基本設定では、.avcconfig ファイルの文字列の置き換えのみが必要となります。
- ユーザー、管理者、イニシエータ、および関係者などといった基本的な WS-HT 名を置き換えるための、[タスク] リストの [ロール] リストに追加する新しいロール名

- [通知]、[開いているタスク]、および [終了したタスク] のリストを置き換えるフォルダカテゴリ、またはこのリストに追加する新しいフォルダカテゴリ
- ID サービスのユーザーとグループは、新しいカテゴリを表示できます（必ずしもユーザーアクティビティのロールの割り当てに関連したカテゴリとは限りません）
- [すべて]、[要求済み]、[要求なし] を置き換えるか、これらに追加するために [表示] リストに追加する新しいタスクのリスト

詳細設定

詳細設定には、カスタマイズ可能な次のようなタスクの表示方法や検索方法があります。

- タスクのカスタムカラム。詳細については、[「タスクロールのフィルタリング用のカスタムカラムの設定」](#)（ページ 526）を参照してください。
- タスクリストの包括的なフィルタリング。詳細については、[「WhereClause を使用した、GetMyTasks フィルタの設定」](#)（ページ 528）を参照してください。
- タスクリストの RSS フィードフィルタ。[「RSS または Atom フィードフィルタの設定」](#)（ページ 530）を参照してください。

基本的なタスクロールのフィルタ構成

このトピックの概要については、[「タスクロールフィルタの設定」](#)（ページ 524）を参照してください。

プロセスセントラル構成ファイル（.avcconfig ファイル）で、[タスクロールフィルタ] セクションのデフォルト構成を次のように変更できます。

【ロール】 リストに新しいロールを追加する手順

ロールの基本セットには、ユーザー（潜在的な所有者）、管理者（ビジネス管理者）、イニシエータ（タスクイニシエータ）、および関係者という WS-HT ロールが含まれます。WS-HT のロールに基づいて、独自のロール名を追加することができます。

ファイルの次の行に、選択した ID を追加して、表示するロール名の名前属性を入力します。

```
<tns:taskRoleFilterDef id="FILL_IN_ROLE_NAME" name="FILL_IN_ROLE_NAME">
```

【表示】 リストに新しい項目を追加する手順

表示される [表示] 項目の基本セットには、[すべて]、[要求済み]、[要求なし] が含まれます。

ファイルの次の行に、ID を追加し、[表示] リストに表示されるフィルタ名の名前属性を入力します。

```
<avccom:taskFilterDef id="FILL_IN_TASK_FILTER_ID" name="FILL_IN_TASK_FILTER_NAME">
```

【タスク】 リストに新しいカテゴリフォルダを追加する手順

カテゴリの基本セットには、[通知]、[開いているタスク]、および [終了したタスク] が含まれます。

ファイルの次の行に、ID を追加し、[タスク] リストに表示されるカテゴリ名の名前属性を入力します。

```
<tns:taskFilterCategoryDef id="FILL_IN_FILTER_CATEGORY_ID" name="FILL_IN_FILTER_CATEGORY_NAME">
```

指定したユーザーおよびグループのフォルダを表示または非表示にする手順

追加した新しいフォルダを表示することができるユーザーとグループを指定できます。プロセスサーバーで有効になっている ID サービスを使用して、ユーザーとグループに関連付けられているロール名をコピーします。すべてのユーザーに新しいフォルダが表示される場合は、<role>要素を削除します。

デフォルトで表示される<role>要素は変更してください。また、要素を空にするとユーザーによるフォルダの表示が許可されない状態となるため、要素は空のままにしないでください。

```
<avccom:allowedRoles>
  <avccom:role>FILL_IN_ROLE_1</avccom:role>
```

```
<avccom:role>FILL_IN_ROLE_2</avccom:role>
</avccom:allowedRoles>
```

ユーザーロールを指定しない場合は、<roles>を削除する必要があります。デフォルト値は、フォルダを表示するにはユーザーが FILL_IN_ROLE_1 グループに属している必要があることを意味します。

構成の例:

```
<tns:taskRoleFilterDefs>
  <!-- for All users in the Role picklist -->
  <tns:taskRoleFilterDef id="All_Role" name="All">
    <!-- New folder -->
    <tns:taskFilterCategoryDef id="Loan_Apps" name="New Car Loans">
      <!-- Display the folder to these users and groups -->
      <avccom:allowedRoles>
        <avccom:role>group_Auto</avccom:role>
        <avccom:role>gDriver</avccom:role>
      </avccom:allowedRoles>
      <!-- Add this filter to the Show picklist -->
      <avccom:taskFilterDef id="auto_loans" name="InProgress">
        </tns:taskFilterCategoryDef>
      </tns:taskFilterDef>
    </tns:taskFilterCategoryDef>
  </tns:taskRoleFilterDef>
</tns:taskRoleFilterDefs>
```

詳細構成については、以下を参照してください。

- [「タスクロールのフィルタリング用のカスタムカラムの設定」 \(ページ 526\)](#)
- [「WhereClause を使用した、GetMyTasks フィルタの設定」 \(ページ 528\)](#)
- [「RSS または Atom フィードフィルタの設定」 \(ページ 530\)](#)

タスクロールのフィルタリング用のカスタムカラムの設定

.avccom:taskColumns ファイルの [タスクロールフィルタ] セクションで、タスクのカテゴリ（フォルダ）に表示するタスクカラム名を追加できます。カラム名にはデフォルトの設定があり、表示されるカラムを追加したり、置き換えたりすることができます。

- 標準の WS-HT カラム名を追加するには、このヘルプの他の場所にある「WS-HT タスクプロパティのリスト」を参照してください。
- カスタムカラム名を追加するには、このヘルプの他の場所にある「カスタムタスクプロパティの作成」を参照してください。
- カラム名をローカライズするには、以下の例を参照してください

デフォルトの設定

taskColumn のデフォルトの設定では、標準の WS-HT タスクプロパティ（すべてに接頭辞 *Task* が付きます）が使用されます。

```
<avccom:taskColumns>
  <avccom:taskColumn property="Task.Id">Id</avccom:taskColumn>
  <avccom:taskColumn property="Task.Status"></avccom:taskColumn>
  <avccom:taskColumn property="Task.Priority"></avccom:taskColumn>
  <avccom:taskColumn property="Task.Name"></avccom:taskColumn>
  <avccom:taskColumn property="Task.PresSubject"></avccom:taskColumn>
  <avccom:taskColumn property="Task.CreatedOn"></avccom:taskColumn>
</avccom:taskColumns>
```

これらのカラムを置き換えたり追加したりするには、次のようなカスタムタスクプロパティを使用して、独自のカラム仕様を追加します。

```
<avccom:taskColumn property="first">First Name</avccom:taskColumn>
```

ユーザーアクティビティのタスク定義の一部として、カスタムカラム名を作成する必要があります。

カラム名をローカライズするには、作成したメッセージプロパティファイルからキーを指定します。例えば、以下のカラム名は\${bundle.key}と指定されています。

```
<avccom:taskColumn property="Task.CreatedOn"
  type="dateTime">${bundle.key}</avccom:taskColumn>
```

メッセージプロパティファイルの作成に関する詳細については、[「フォームとタスクへの多言語サポートの追加」 \(ページ 533\)](#)を参照してください。

次のサンプルの [タスクロールフィルタ] セクションは、プロセスセントラルでのカスタムカラムの使用方法を示しています。

```
<!--
  Example - adding a new category/folder called Hello Tasks.
  This folder will appear in the All role.
-->
<tns:taskRoleFilterDefs>
  <tns:taskRoleFilterDef
    id="ae-avc-role-all" name="{ae.avc.role.user.name}">
    <tns:taskFilterCategoryDef id="custom-cat"
      name="Hello Tasks">
      <avccom:allowedRoles>
      </avccom:allowedRoles>
    <!--
      Define custom columns that appear in Process Central
      (or remove/comment out root element to use
      default column list provided out of the box).
    -->
    <avccom:taskColumns>
      <!-- Task.Id, Task.Status and Task.Name are
        standard WS-HT columns.
      -->
      <avccom:taskColumn
        property="Task.Id">Id</avccom:taskColumn>
      <avccom:taskColumn
        property="Task.Status"></avccom:taskColumn>
      <avccom:taskColumn property="Task.Name">Task Name</avccom:taskColumn>
      <avccom:taskColumn property="Task.PresName">
        Display Name</avccom:taskColumn>
      <!-- You can also specify the column data
        formatting via the type attribute. Useful
        values are date, dateTime and boolean. The
        column display name can also be externalized
        via ${18n.key} method.
      -->
      <avccom:taskColumn property="Task.CreatedOn" type="dateTime">
        ${bundle.key}</avccom:taskColumn>
      <!-- The property names 'first' and 'last' are
        WSH task presentation parameters defined in
        your tasks (custom properties).
      -->
      <avccom:taskColumn property="first">First Name</avccom:taskColumn>
      <avccom:taskColumn property="surname">Surname (Last Name)</avccom:taskColumn>
    </avccom:taskColumns>
    <avccom:taskFilterDefRef ref="ae-avc-user-open-all"/>
    <avccom:taskFilterDefRef
      ref="ae-avc-user-open-unclaimed"/>
    <avccom:taskFilterDefRef
      ref="ae-avc-user-open-claimed"/>
    <avccom:taskFilterDefRef
      ref="ae-avc-user-open-suspended"/>
  </tns:taskFilterCategoryDef>
</tns:taskRoleFilterDef>
</tns:taskRoleFilterDefs>
```

WhereClause を使用した、GetMyTasks フィルタの設定

.avcconfig ファイルの [タスクロールフィルタ] セクションで、[Pick の表示] エントリ内の特定のタスクに適用するフィルタを追加できます。例えば、[要求なしのタスク] にフィルタを作成して、優先度 0（最高の優先度）のみを表示することができます。

表示フィルタを使用した、タスクリストのフィルタリング

<avccom:taskFilterDef/>セクションの<avccom:filter/>セクションでデフォルトの設定を変更します。

次のコードスニペットは、デフォルトの設定を示しています。

```
<avccom:filter>
  <tsst:getTasks xmlns:tsst="http://schemas.active-endpoints.com/b4p/wshumantask/2007/10/aeb4p-task-state-
wsdl.xsd">
    <htdt:getMyTasks
      xmlns:htdt="http://www.example.org/WS-HT/api/xsd">
      <htdt:taskType>TASKS</htdt:taskType>
      <htdt:genericHumanRole>POTENTIAL_OWNERS</htdt:genericHumanRole>
      <htdt:status>READY</htdt:status>
      <htdt:whereClause>Task.Name =
        'ApproveLoan'</htdt:whereClause>
      <htdt:maxTasks>20</htdt:maxTasks>
    </htdt:getMyTasks>
    <tsst:taskIndexOffset>0</tsst:taskIndexOffset>
  </tsst:getTasks>
</avccom:filter>
```

WS-HT の whereClause を使用した、パラメータによるタスクリストのフィルタリング

whereClause では次の演算子を使用できます。

- >
- >=
- <
- <=
- =
- LIKE
- および
- または
- ()

次の例は、さまざまな whereClause 構文を示しています。ブール値 and および or では大文字と小文字が区別されることに注意してください。

タスクカラム名、タスクタイプ、一般的なヒューマンロール、およびその他の WS-HT 仕様の完全なリストについては、Informatica Business Process Manager SDK を使用してください。

例 1

次の例は、標準の WS-HT タスクプロパティ名を使用した whereClause のサンプルを示しています。

```
<htdt:whereClause>Task.Name = 'ApproveLoan'</htdt:whereClause>
```

例 2

次の例では、フィルタプロパティを組み合わせています。

```
Task.Priority &lt; 3 and Task.Owner = 'loanrep1'
```

例 3

次の例でも、フィルタプロパティを組み合わせています。

```
Task.ProcessId = 10
and(Task.Escalated = true
or Task.Priority = 0)
```

例 4: プレゼンテーションパラメータの使用（カスタムタスクプロパティ）

次の例は、プレゼンテーションパラメータの使用を示しています。

```
Task.Name = 'LoanApproval'
and loanAmount > 10000
and zipCode = '06484'
```

プレゼンテーションパラメータ

プレゼンテーションパラメータをクエリすると、追加情報が返されます。

```
<tsst:propertyValues
  xmlns:tsst=
    "http://schemas.active-endpoints.com/b4p/wshumantask/
      2007/10/aeb4p-task-state-wsdl.xsd">
  <tsst:propertyValue
    name="parentprocessid" type="ns:string"
    xmlns:ns="http://www.w3.org/2001/XMLSchema">5013
  </tsst:propertyValue>
</tsst:propertyValues>
```

プレゼンテーションパラメータを使用してタスクがデプロイされていない場合、エラーメッセージが返されます。

```
<htdt:illegalArgument
  xmlns:htdt="http://www.example.org/WS-HT/api/xsd">
  Task parameter property 'parentProcessId1' does not
  exist.
</htdt:illegalArgument>
```

ただし、このパラメータを持つタスクが存在しなくても有効な名前である場合は、空の情報が返されます。こうしたタイプのパラメータの例を次に示します。

```
<aetgt:getMyTasksResponse
  xmlns:aetgt="http://www.example.org/WS-HT/api/xsd"
  xmlns:htdt="http://www.example.org/WS-HT/api/xsd"
  xmlns:result="http://saxon.sf.net/xquery-results"
  xmlns:tss="http://www.activebpe1.org/b4p/2007/10/wshumantask/aeb4p-task-storage.wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
```

この要素は、parentProcessId パラメータであるタスクをリストし、その値を返します。

```
<xsd:getMyTasks>
  <xsd:taskType>TASK</xsd:taskType>
  <xsd:whereClause>parentProcessId >'</xsd:whereClause>
</xsd:getMyTasks>
```

次の表は、プレゼンテーションパラメータを使用した場合に発生する可能性のある内容をまとめたものです。

プレゼンテーションパラメータ	デプロイされるかどうか	実際の値	クエリ	結果
MyString	はい	"a"、"b"	<WHERE 句>	<パラメータの結果>
MyStringB	○	× 値	<WHERE 句>	<パラメータの結果>
MyStringC	いいえ	適用されません	<WHERE 句>	<パラメータの結果>

getTasks と orderBy 要素の使用

orderBy 要素を使用して、getTasks によって返された結果を並べ替えることができます。使用可能なフィールド名は次のとおりです。

- Created
- Expiration
- Modified
- Name
- Owner
- PresentationName
- PresName
- PresSubject
- Priority
- State
- Summary

次の例では、最新のものから順に（つまり、降順で）結果を並べ替え、次に優先度順に並べ替えています。

```
OrderBy orderBy = new OrderBy();
orderBy.getFieldId().add("-Created");
orderBy.getFieldId().add("-Created");
```

降順を設定するためのマイナス記号（“-”）が使用されていることに注意してください。

XML では、これと同じ例は次のようになります。

```
<ns:orderBy>
  <ns:fieldId>-Created</ns:fieldId>
  <ns:fieldIdPriority</ns:fieldId>
</ns:orderBy>
```

RSS または Atom フィードフィルタの設定

このトピックの概要については、[「タスクロールフィルタの設定」 \(ページ 524\)](#)を参照してください。

.avcconfig ファイルの [タスクロールフィルタ] セクションには、RSS または Atom Web フィードのデフォルトの URL 詳細を含む要素があります。プロセスセントラルには、タスクの更新に関する通知をユーザーに送信するように設定可能な Web フィードサブスクリプションサービスが含まれています。

.avcconfig ファイルでは、デフォルトのクエリと形式は次のように表示されます。

```
<avccom:feedQueryString format="rss" />
```

実行時に、Web フィードの URL は<getMyTasks>に基づいて自動的に生成されます。例えば、プロセスセントラルで、開いているタスクがタスクリストに表示されている場合、フィードでは新しいタスクが到着した際に更新が送信されます。

デフォルトの Web フィード形式は RSS ですが、必要に応じて Atom に変更することもできます。

avcconfig ファイルをデプロイする場合に [タスクロールフィルタ] セクションをコメントアウトしたままにすると、デフォルトの avoscentral.avcconfig ファイルからデフォルトの Web フィード URL が生成されます。このファイルは、プロセスコンソールの [カタログ] セクションで確認できます。

Web フィードクエリフィルタは次のように変更できます。

- taskFilterDef セクションを設定します。この設定の詳細は、フィードクエリが生成される際に自動的に使用されます。次の例は、feedQueryString がフィルタ定義の一部としてエンクロードされている場合を示しています。文字列を変更する必要はありません。

```
<avccom:taskFilterDef >
  <avccom:filter>
    <tsst:getTasks ...>
      <htdt:getMyTasks>... </htdt:getMyTasks>
    </tsst:getTasks>
    <avccom:feedQueryString format="rss" />
  </avccom:filter>
</avccom:taskFilterDef>
```

- (非推奨)。デフォルトの feedQueryString を変更することはお勧めしません。プロセスセントラルの各表示によって、ページのフィルタに基づいて Web フィード URL が自動的に生成されます。このフィード URL を変更する必要はありません。ただし、WS-HT getMyTasks 操作で既知のパラメータを追加することにより、feedQueryString を変更することができます。以下に例を示します。

```
<avccom:feedQueryString format="rss">filter=reserved</avccom:feedQueryString>
```

- パラメータの詳細なリストについては、「*Process Developer WS-HumanTask API*」ヘルプの「RSS または Atom フィードとしてのタスクリストの取得」を参照してください。

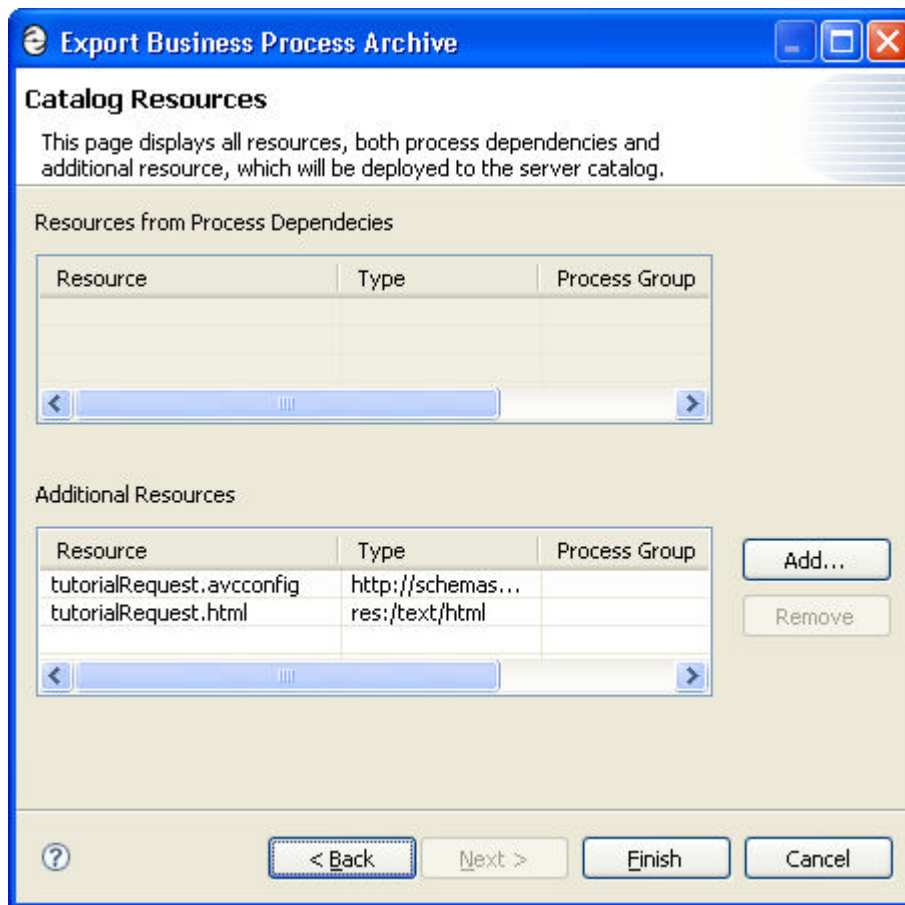
フォームレポートと構成ファイルのデプロイ

プロセスセントラルで使用するためにプロセスサーバーにデプロイするファイルには、プロセスセントラル構成ファイル、フォームの HTML ファイル、BIRT レポートデザイン、および画像や多言語プロパティファイルなどの関連リソースが含まれます。

これらのリソースのみを含むビジネスプロセスアーカイブ (.BPR) ファイルを作成することができます。

BPR を作成するには、[ファイル] > [エクスポート] > [Orchestration] > [コントリビューション-ビジネスプロセスアーカイブ] を選択します。[カタログリソース] ページで、次のリソースタイプを選択します。

- プロセスセントラル構成ファイルの一元的な設定
- フォームの HTML
- 画像のバイナリファイル
- 英語以外の言語の外部化された文字列を含む**プロパティファイル**。これらのファイルが表示されていない場合は、[その他] というタイプとして追加します。



関連事項:

- [「追加リソースのデプロイ」 \(ページ 487\)](#)。
- [「プロセスセントラル構成ファイルの作成」 \(ページ 520\)](#)
- [「プロセスセントラルプロセスフォームの作成」 \(ページ 512\)](#)
- ヒューマンタスクフォーム（タスクレンダリング）の作成に関する詳細については、「*Process Developer* のヘルプ」内の「ヒューマンタスク」を参照してください。

プロセスセントラルへの多言語サポートの追加

さまざまな言語バージョンのプロセスセントラル、およびその言語バージョンにデプロイするフォームおよびレポートを作成できます。プロセスセントラルアプリケーションのユーザーインターフェースは、Web ブラウザの優先言語で表示されます。また、このインターフェースには、要求、タスク、およびレポートの言語とロケールのサポートがあります。

多言語サポートの基本的なアプローチとしては、文字列を外部化します。サポートされる言語ごとに、外部化された文字列に既存のデフォルトプロパティファイルを使用するか、新しいデフォルトプロパティファイルを作成します。言語ごとに 1 つずつ、追加のプロパティファイルを使用します

多言語サポートの構成に関する詳細については、次のトピックを参照してください。

- [「Web ブラウザでの優先言語によるプロセスセントラルの表示」 \(ページ 533\)](#)

- [「フォームとタスクへの多言語サポートの追加」 \(ページ 533\)](#)
- [「.avconfig ファイルへの多言語サポートの追加」 \(ページ 534\)](#)
- [「レポートへの多言語サポートの追加」 \(ページ 535\)](#)
- [「プロセスセントラルの詳細設定」 \(ページ 536\)](#)
- [「I18N 関数」 \(ページ 284\)](#)

Web ブラウザでの優先言語によるプロセスセントラルの表示

必要に応じて、選択した言語でプロセスセントラルを表示できます。プロセスセントラルアプリケーションに表示される各テキスト文字列は、avoscentral.properties という名前のファイルに外部化されています。

これは、ログインページの表記、すべてのコマンドボタン、タイトル、エラーメッセージ、およびその他の表記に対して、外部化された文字列によるプロパティファイルを指定できることを意味します。

<http://www.activevos.com> から、必要な言語のプロパティファイルを取得できます。言語 BPR は、zip アーカイブ内の instructions.html ファイルに含まれているため、この zip ファイルをダウンロードしてプロセスコンソールにデプロイします。詳細については、<http://www.activevos.com> のソフトウェア開発キット (SDK) ページにアクセスし、「プロセスセントラル言語パックでの手順」を参照してください。

<http://www.activevos.com> から取得できない言語については、外部化された文字列の言語値を指定できます。独自の言語ファイルを追加するための手順は、言語パックのアーカイブに含まれています。

プロセスセントラルのユーザーインターフェイスに多言語サポートを設定する手順

1. Web ブラウザで、<http://www.activevos.com/developers/sdks> にアクセスします。
2. プロセスセントラル言語パックのバージョン 8 以降をダウンロードします。
3. ファイルを解凍し、instructions.html の指示に従って言語パッケージを変更します。

[「.properties ファイルの命名規則」 \(ページ 536\)](#)も参照してください。

フォームとタスクへの多言語サポートの追加

さまざまな言語をサポートする要求フォームとタスクフォーム（ユーザーアクティビティ用）を作成できます。このようなフォームを作成するには、新しいフォームに加えて、サポートされる各言語のデフォルトのプロパティファイルと追加のプロパティファイルを作成します。プロパティファイルには、スキーマベースの keyIdentifiers の外部化された文字列が含まれています。要求に対してこのサポートを設定するための手順を以下に示します。

タスクフォームの詳細については、このヘルプの他の場所にある「ヒューマンタスク」トピックの「タスクフォームへの多言語サポートの追加」を参照してください。

新しい要求フォームとデフォルトのプロパティファイルを作成します。

1. [「プロセスセントラルプロセスフォームの作成」 \(ページ 512\)](#)に記載されているように、新しい要求フォームを作成します。
2. [フォームの作成] ダイアログで、[詳細] ボタンを選択します。
3. [文字列の外部化] チェックボックスを選択します。
4. プロパティファイルのデフォルトの名前と場所を受け入れるか、[プロパティファイル] の横にあるダイアログボタンを選択して、ファイルに名前を付けて保存します。名前にはアンダースコアを使用しないでください。

文字列を外部化すると、次のようなイベントが発生します。

- フォーム内のスキーマ要素が、名前ではなく識別子によって生成され、プロパティファイルで見つかった文字列が実行時に置き換えられます。

- 多言語サポートを示す新しいリンクがフォーム内に追加されます。

```
<link rel="i18n" charset="UTF-8" type="text/plain"
      href="myForm.properties" />
```

- スキーマ要素のリストは、キーと値のペアとしてプロパティファイルに追加されます。
- 作成するプロパティファイルは、外部化されたデフォルトの文字列ファイルです。

サポートする追加の言語ごとに、次の手順を実行します。

1. デフォルトのプロパティファイルのコピーを作成し、フォームフォルダに貼り付けます。
2. [「.properties ファイルの命名規則」 \(ページ 536\)](#)に記載されている構文で、コピーしたファイルに名前を付けます。基本的に、各ファイルにはまったく同じ名前を使用し、その後にアンダースコアを付け、その後に言語名の表現用の ISO 639 コードを使用する必要があります。
3. ファイルをテキストエディタで開き、既存の値を別の言語に置き換えます。

プロパティをテストするには、プロセスセントラルにログインする際に言語を指定します。以下に例を示します。

<http://localhost:8080/activevos-central/login.jsp?lang=fr>

[「I18N 関数」 \(ページ 284\)](#)も参照してください。

.avcconfig ファイルへの多言語サポートの追加

このトピックの概要については、[「プロセスセントラルへの多言語サポートの追加」 \(ページ 532\)](#)を参照してください。

プロセスセントラル構成ファイルで、次の属性に外部化された文字列を指定できます。

- タスク、フォーム、およびレポートに追加するフォルダのカテゴリ名（例えば、*Tutorial and Samples* という名前のフォームカテゴリに、Process Developer チュートリアル要求とサンプル要求フォームが含まれている場合など）。
- レポートとフォームの説明（例えば、チュートリアルフォームに、「ローン承認リクエストの送信（基本チュートリアル）」という説明が含まれている場合など）。
- タスクロールと表示選択リストの名前

多言語サポートを追加する手順

1. [「プロセスセントラル構成ファイルの作成」 \(ページ 520\)](#)の説明に従って、新しい avcconfig ファイルを作成します。deploy フォルダに保存します。例: tutorial.avcconfig など。
2. 外部化するファイル内の名前と説明の属性ごとに、`${keyIdentifier}`を参照するようにプレースホルダテキストを変更する必要があります。以下に例を示します。

```
<tns:requestCategoryDef id="education" name="${tutorial.and.samples}">
```
3. テキストエディタで、.avcconfig ファイルのデフォルトの外部化文字列プロパティファイルである新しい.properties ファイルを作成します例: tutorialConfig.properties など。
4. avcconfig ファイルはこのファイルを参照する必要があるため、avcconfig ファイルに多言語サポートを示す次の行を追加する必要があります。

```
<tns:i18n location="/path/to/tutorialConfig.properties" />
```
5. デフォルトのプロパティファイルで、各`${keyIdentifier}`とその英語の値を追加します。以下に例を示します。
tutorial.and.samples=チュートリアルとサンプル
6. 他の外部化された文字列を追加する際に、次のような行をプロパティファイルに追加します。
tutorialRequest.description=Submit loan approval request (basic tutorial)

7. 追加の言語ごとに、デフォルトのプロパティファイルのコピーを作成し、その言語の keyIdentifier の値を変更します。
tutorial.and.samples=Tutoriel et échantillons
8. 各ファイルにアンダースコアと言語名を付けます。以下に例を示します。
tutorialConfig_fr.properties
9. コピーが UTF-8 でエンコードされていることを確認します。デフォルトでは、Process Developer でのエンコーディングが ISO-8859-1 に設定されています。

デプロイ後、プロセスセントラルでログインする際に言語を指定して、プロパティをテストします。以下に例を示します。

`http://localhost:8080/activevos-central/login.jsp?lang=fr`

構成ファイルのスニペットの例:

```
<tns:requestCategoryDef id="education"
  name="{tutorial.and.samples}">
<avccom:description>
  {tutorialRequest.description}
</avccom:description>
<tns:il8n location="/path/to/tutorialConfig.properties" />
```

デフォルトのプロパティファイルの例:

```
tutorial.and.samples=Tutorial and Samples
tutorialRequest.description=Submit loan approval request (basic tutorial).
```

[「.properties ファイルの命名規則」 \(ページ 536\)](#)も参照してください。

レポートへの多言語サポートの追加

プロセスセントラルには、ビジネスインテリジェンスレポートツール (BIRT) デザイナーで作成されたレポートが表示されます。プロセスセントラル内では、レポートは BIRT ビューアに表示されます。

プロセスセントラルは、次の言語のサポートします。

- ドイツ語
- スペイン語
- フランス語
- 日本語
- 韓国語
- 簡体字中国語

ブラウザの優先言語が上記のいずれかに設定されている場合、BIRT ビューアでは、その言語でレポートが自動的に表示されます。

言語をテストするには、言語を指定する URL を入力します。以下に例を示します。

`http://localhost:8080/activevos-central/login.jsp?lang=zh_CN`

他の言語の多言語サポートを有効にする場合は、次の考慮事項に注意してください。

- レポートデザインのプロパティファイルを作成するには、<http://www.birt-exchange.org/wiki/BIRT:Internationalization/>を参照してください。
- プロセスセントラルでサポートされていない言語については、BIRT ビューアのテキスト文字列とエラーメッセージのプロパティを作成する場合に、複数の messages.properties ファイルを手動で検索し、変更する必要があります。BIRT ビューアのテキスト文字列のメインファイルは、`server\webapps\activevos\WEB-INF\lib\ae_birtviewer.jar`にあります。

.properties ファイルの命名規則

プロセスセントラルの設定やフォームおよびタスクに対する多言語サポートを追加する場合は、サポートする言語ごとにプロパティファイルを作成します。1つのデフォルトファイルと言語コードを持つその他のファイルがあります。

MyPropertiesFile.properties:

これは通常、en-US のデフォルトファイルです。

MyPropertiesFile_ISO639Name.properties:

言語名を表すため、言語ごとにアンダースコアと ISO 639.2 コードを追加する必要があります。_language コードとオプションの国コードをデフォルトのファイル名に追加します。国コードは、ISO-3166 で定義されている大文字の2文字のコードです。

例:

- requestFormOperation.properties
- requestFormOperation_fr.properties
- requestFormOperation_de.properties
- MyConfigFile.properties
- MyConfigFile_zh_CN.properties

プロセスセントラルの詳細設定

プロセスセントラルのフォーム、タスク、およびレポートの設定と多言語サポートの提供に加えて、より高度な設定を追加できます。

このトピックでは、次のような追加の設定を実装する基本的な方法について説明します。

- プロセスセントラルのロゴとアイコンの画像の置き換え。
- Informatica Business Process Manager SDK およびドキュメントとプロセスセントラルのサンプルへのポインタ。

詳細な例については、<http://www.activevos.com> を確認してください。

独自のロゴを追加する手順

1. プロジェクトで、デフォルトの avoscentral.properties ファイルに基づいた名前を持つテキストファイルを作成します（例えば、avoscentral_en.properties など）。
2. サーバーを起動し、プロセスコンソールを開きます。
3. avoscentral.properties を検索します。
このファイルには、プロセスセントラルのユーザーインターフェースの keyIdentifiers と値が含まれています。
4. プロセスコンソールの [リソース詳細] ページに、プロパティファイルのリソース定義が表示されます。
5. プロセスセントラルに独自のロゴを追加するには、avoscentral.properties から次の行をコピーします。
ae.avc.header.logo.image=../avc-res/images/ActiveVOScentralTitle.png
6. この行をプロパティファイル（avoscentral_en.properties）に貼り付けます。
7. 識別子の値を変更して、ロゴの場所と名前を指定します。以下に例を示します。
ae.avc.header.logo.image=project:/myProject/images/myLogo197x35.png
8. ロゴとプロパティファイルをデプロイするための新しい.bprd ファイルを作成します。

9. .bprd には、プロパティファイルのカatalogエントリが含まれています。以下に例を示します。

```
<otherentry typeURI="res:/text/plain" location="project:/myProject/deploy/avoscentral_en.properties"
classpath="resources/myProject/deploy/avoscentral_en.properties" />
```
10. プロパティファイルの場所のヒントを変更します。プロパティファイルは、次のように、avoscentral.properties の [リソース詳細] ページの場所のヒントで指定したCatalogの場所に保存する必要があります。

```
location="project:/com.activeee.avoscentral.services/bundle/avoscentral_en.properties"
```

ヒント:

- ファイルをデプロイするためのショートカットとして、次のトピックに記載されているように、プロジェクトを作成してファイルをエクスポートすることをお勧めします: [「Web ブラウザでの優先言語によるプロセスセントラルの表示」 \(ページ 533\)](#)。
- Web サイトの開発者向けの使い方のセクションで、「プロセスセントラルとフォーム」を参照してください。
- フォームの作成に関する詳細については、<http://www.activevos.com> の「製品ドキュメント」セクションから Informatica Business Process Manager SDK をダウンロードしてください。

第 34 章

プロセスサーバーおよびプロセスセントラルレポート

サーバーにデプロイするリソースを追加します。

プロセスコンソールの大部分のページには、デプロイされたアクティブなプロセスに関する情報およびユーザープロセス用の BPEL からのアクティブなタスクが表示されます。多くの場合、これらのページには、Eclipse のビジネスインテリジェンスおよびレポートツール (BIRT) を使用して Process Developer で設計されたレポートが含まれます。

レポートは基本的に、データベースにクエリを実行し、表やグラフなど、必要なレイアウトに結果を返すことで設計します。

Process Developer で独自のレポートを作成し、それらのレポートをサーバーやプロセスセントラルにデプロイできます。サーバー上でプロセスを実行すると、データベースが更新され、レポートも更新されます。

レポートを設計して保存した後に、BPR ファイルでプロセスサーバーにデプロイできます。レポートは、サーバーに保存されるリソースです。作成したレポートは、標準のレポートとともにプロセスコンソールの [レポート] ページに表示されます。

プロセスセントラルでもレポートを表示する場合は、そのレポート用のプロセスセントラル構成ファイルを作成する必要があります。詳細については、[「プロセスセントラル構成ファイルの作成」 \(ページ 520\)](#)を参照してください。

レポート定義はプロセスコンソールカタログに保存されるため、必要に応じて、各レポートの XML ソースコードを表示、更新、および削除します。

レポートを設計する場合は、Process Developer に含まれるサンプルレポートを確認することをお勧めします。

レポートに関する説明が記載されたトピックは次のとおりです。

- [「ユーザーレポート Orchestration プロジェクトの作成」 \(ページ 539\)](#)
- [「Process Developer レポートテンプレートの使用」 \(ページ 539\)](#)
- [「Process Developer データソースの使用」 \(ページ 541\)](#)
- [「Process Developer データソースからのデータセットの作成」 \(ページ 544\)](#)
- [「Process Developer データモデルの理解」 \(ページ 545\)](#)
- [「Process Developer レポートのデプロイ」 \(ページ 545\)](#)
- [「デプロイされたレポートの更新または削除」 \(ページ 546\)](#)
- [「Reporting Service」 \(ページ 546\)](#)

ユーザーレポート Orchestration プロジェクトの作成

サンプルのレポートを表示およびデプロイするには、テンプレートから Orchestration プロジェクトを作成します。

1. **【ファイル】 > 【新規】 > 【Orchestration プロジェクト】** を選択します。
2. プロジェクトにサンプルレポートまたはユーザーレポートなどの名前を付け、**【次へ】** をクリックします。
3. **【テンプレートからの新規 Orchestration プロジェクト】** ページで、**【ユーザーレポート】** を選択します。

Orchestration プロジェクトで、report というフォルダが表示されます。フォルダ内のレポートの 1 つをダブルクリックして、**【レポートデザイン】** パースペクティブを開きます。

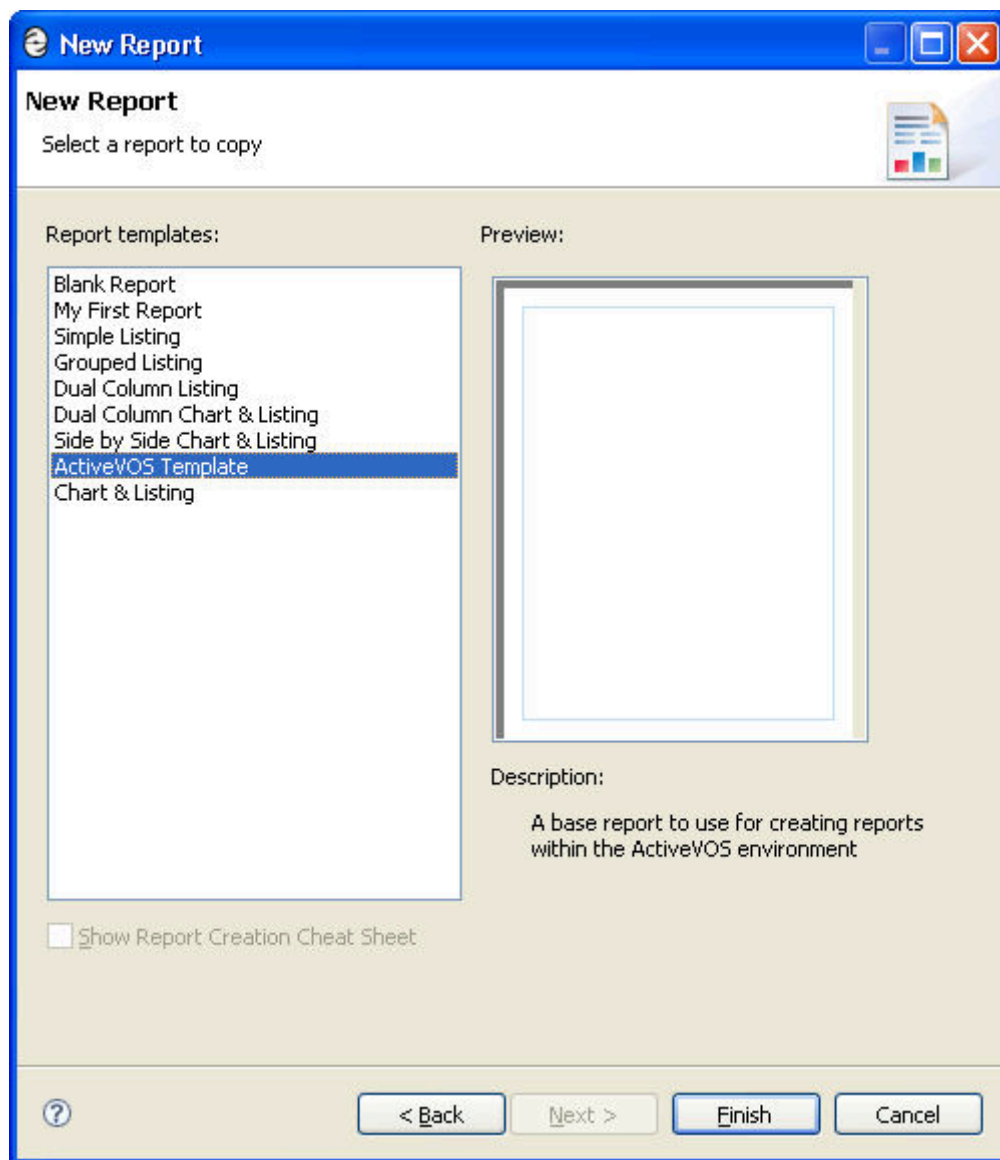
サンプルレポートは、Process Developer テンプレートに基づいています。詳細については、[「Process Developer レポートテンプレートの使用」 \(ページ 539\)](#)を参照してください。

サンプルレポートを Process Developer 組み込みサーバーにデプロイする場合は、[「Process Developer レポートのデプロイ」 \(ページ 545\)](#)を参照してください。

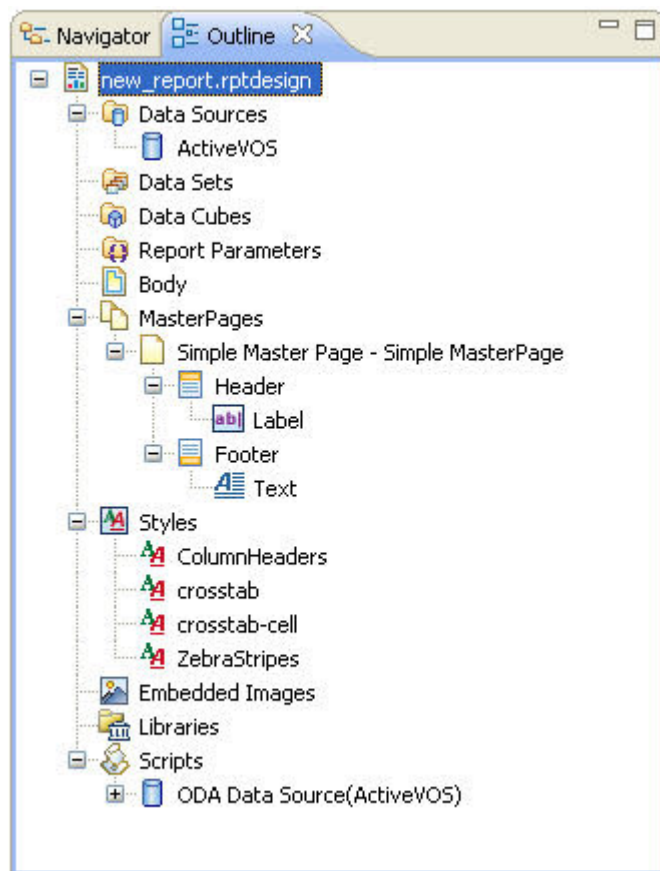
Process Developer レポートテンプレートの使用

Process Developer テンプレートを使用して、新しい Process Developer レポートを作成できます。

1. **【ファイル】 > 【新規】 > 【その他】 > 【ビジネスインテリジェンスとレポートツール】 > 【レポート】** を選択します。
2. 新しいレポートに *new_report.rptdesign* などの名前を付け、**【次へ】** をクリックします。
3. 図に示すように、**【新規レポート】** ページから **【Process Developer テンプレート】** を選択します。



4. **【完了】**を選択し、提案を受け入れて、**［レポートデザイン］** パースペクティブを開きます。
図に示すように、**［アウトライン］** ビューでテンプレートコンポーネントを表示できます。



テンプレートには次のものが含まれています。

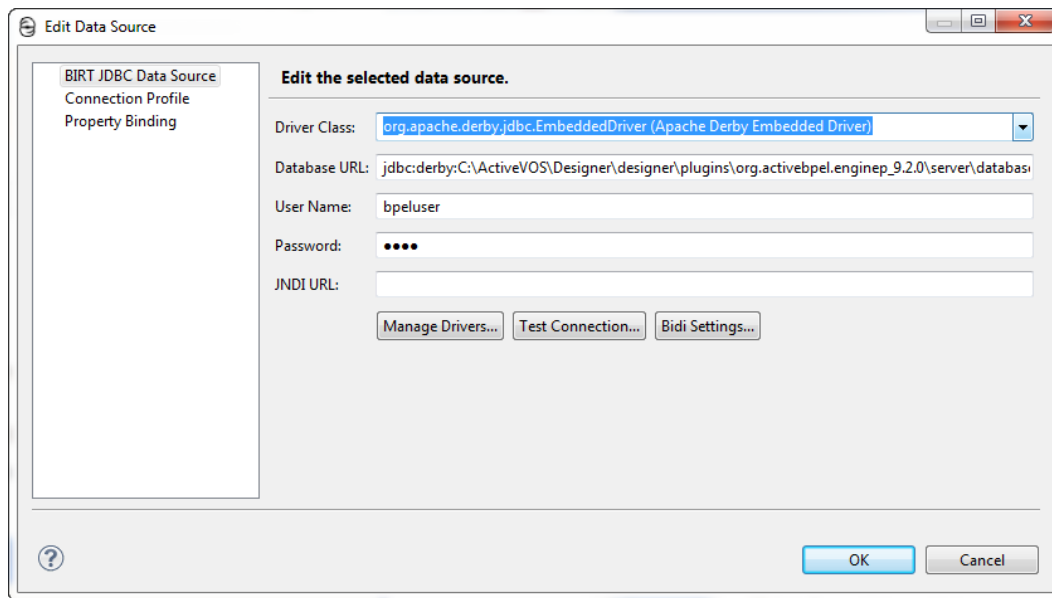
- **データソースの接続。**この接続は、プロセスサーバーに含まれている Derby データベースを指すように事前定義されています。Derby には組み込み JDBC ドライバが含まれています。データソースの詳細については、[「Process Developer データソースの使用」 \(ページ 541\)](#)を参照してください。
- **マスタページのヘッダーとフッター。**ヘッダーまたはフッターをダブルクリックして、プロパティを表示または変更します。
- **スタイル。**データカラムのレイアウトのレポートには、複数のテーブル行スタイルが含まれています。
- **スクリプト。**プロセスサーバーにレポートをデプロイして実行すると、レポート XML ファイルに埋め込まれているスクリプトは、提供された AeBirtContext クラスを使用してデータソースの接続情報を検出します。詳細については、[「Process Developer レポートのデプロイ」 \(ページ 545\)](#)を参照してください。

Process Developer データソースの使用

Process Developer テンプレートには、プロセスサーバーに含まれる Derby データベースを指すように事前定義されたデータソースへの接続が含まれています。Derby には組み込み JDBC ドライバが含まれています。

Derby は一度に 1 つの接続のみを受け入れるため、レポートの設計中に Process Developer 組み込みサーバーを実行することはできません。

接続設定を表示するには、[アウトライン] ビューを開き、[データソース] を展開します。図に示すように、Process Developer データソースをダブルクリックします。



別のプロセスサーバーの設定を変更するには、そのサーバーに構成されたものと同じドライバクラスおよびデータベース URL を追加する必要があります。

例えば、Apache Tomcat を使用してプロセスサーバーを購入し、Apache Tomcat サーバーで MySQL データベースを構成する場合は、次の設定を使用します。

ドライバクラス:

`com.mysql.jdbc.Driver`

データベース URL:

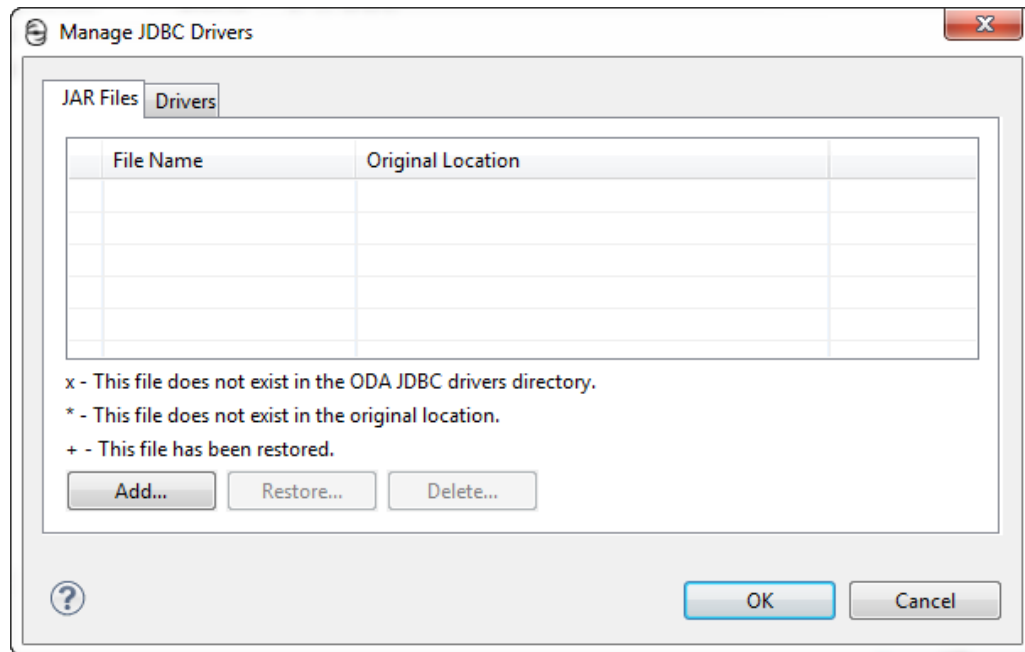
`jdbc:mysql://localhost:3306/ActiveVOS?useUnicode=true
&characterEncoding=UTF-8&characterSetResults=utf8`

使用するサーバーのデータベース接続が、DDL スクリプトへの変更、またはユーザー名やパスワードなどの設定の詳細で構成されている場合は、これらの設定も更新する必要があります。表示される [ユーザー名] と [パスワード] は、すべてのサーバーのデフォルトです。

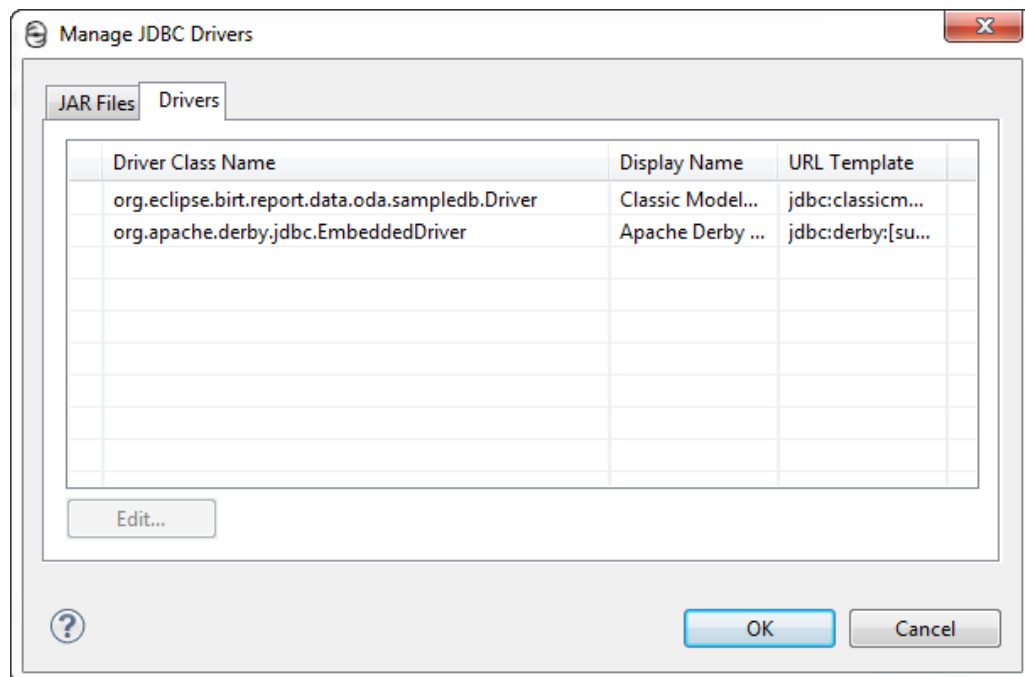
データソースの JNDIURL は実行時に検出されます。

次のような 3 つのボタンを使用します。

- **ドライバの管理:** このボタンを押すと、2 つのタブがあるダイアログが表示されます。1 つ目のタブでは、データソースで使用する JAR ファイルを管理できます。

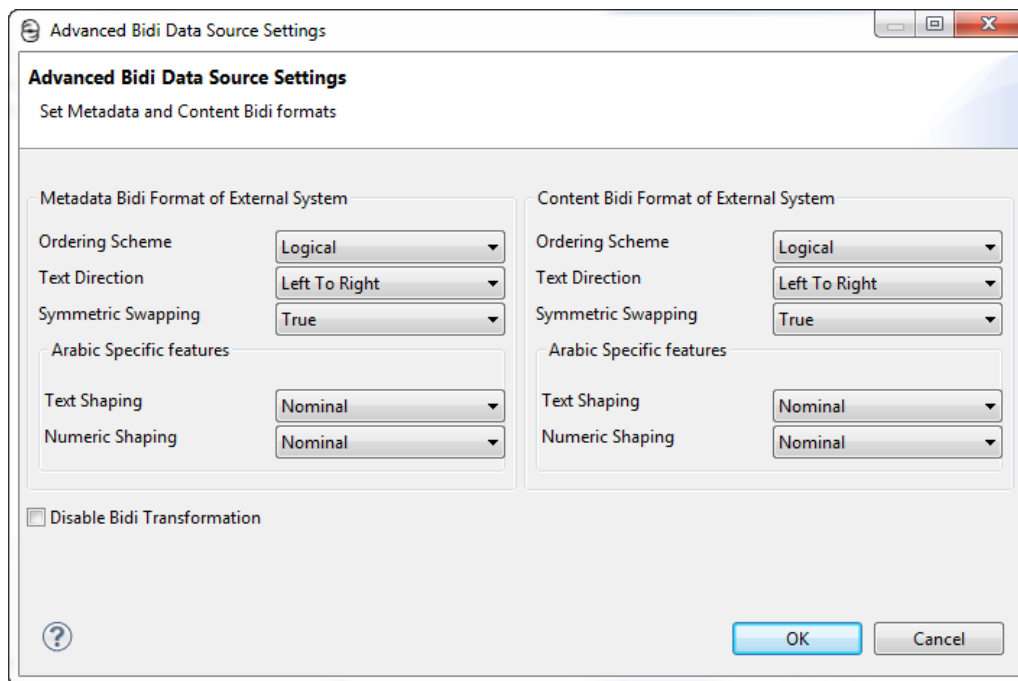


2 つ目のタブでは、ドライバを管理できます。



- **接続のテスト:** このボタンを押すと、データソースへの接続がテストされます。

- **Bidi 設定:** (bidi は「bi-directional」の略です) このボタンを押すと、[高度な Bidi データソースの設定] ダイアログボックスが表示されます。その中にあるコントロールを使用して、メタデータとコンテンツの Bidi 形式を設定します。



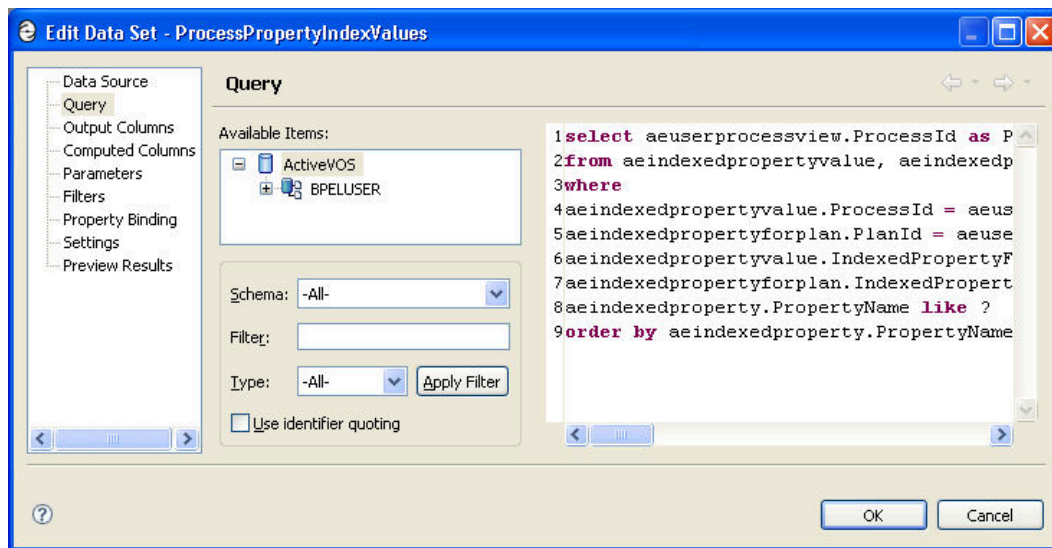
[「Process Developer データソースからのデータセットの作成」 \(ページ 544\)](#)および [「Process Developer データモデルの理解」 \(ページ 545\)](#)を参照してください。

Process Developer データソースからのデータセットの作成

Process Developer レポートを作成するには、データセットを作成します。データセットにより、データソースから取得するデータを指定します。

ユーザーレポート Orchestration プロジェクトのレポートからサンプルデータセットを表示できます。開始するには、[「ユーザーレポート Orchestration プロジェクトの作成」 \(ページ 539\)](#)を参照してください。

ユーザーレポート Orchestration プロジェクトのレポートフォルダからレポートを開きます。次に、以下の例に示すように、データセットをダブルクリックしてプロパティを表示します。



データの使用の詳細については、「*BIRT Report Developer Guide*」の「*How to create a data set*」ヘルプトピックを参照してください。

取得するデータを特定するには、[「Process Developer データモデルの理解」](#) (ページ 545)を参照してください。

Process Developer データモデルの理解

プロセスサーバー用に作成したレポートは、Process Developer レポートテンプレートに含まれる Process Developer データソースからデータを取得します。このテンプレートに基づいてレポートデザインを作成するには、[「Process Developer レポートテンプレートの使用」](#) (ページ 539)を参照してください。

データソースに基づいてデータセットを作成するには、プロセスデータモデルとヒューマンタスクデータモデルに関する詳細が必要です。

説明およびサンプルのエンティティ関係図を表示するには、このヘルプの他の場所にある Process Developer データソースモデルを参照してください。

使用するバージョンの Process Developer に関連付けられた完全なデータソースモデルを表示および使用するには、SDK ページ (<http://www.activevos.com>) にアクセスして [レポート] セクションまでスクロールします。

Process Developer レポートのデプロイ

レポートは、プロセスサーバーにデプロイし、必要に応じてプロセスセントラルに表示可能なリソースです。これは、コントリビューション - ビジネスプロセスアーカイブエクスポートを使用して、BPEL/PDD またはその他のリソースをデプロイする場合と同じ方法でデプロイします。

Derby データベースを含む Process Developer 組み込みサーバー以外にデプロイする場合は、デプロイする前に必ずデータソース接続の詳細を変更してください。詳細については、[「Process Developer データソースの使用」](#) (ページ 541)を参照してください。

また、レポートをプロセスセントラルで表示するには、[「プロセスセントラル構成ファイルの作成」](#) (ページ 520)に記載されている構成ファイルを作成する必要があります。

デプロイ中に、各レポートにグループ名と説明を追加して、レポートが [プロセスサーバーレポート] ページでグループ化されるようにすることができます。

デプロイメントの手順については、[「追加リソースのデプロイ」](#) (ページ 487)を参照してください。

レポートは、Report Process Developer の [XML ソース] タブで表示可能な XML ファイルとしてデプロイします。[「デプロイされたレポートの更新または削除」](#) (ページ 546)に記載されているように、サーバー上の XML ファイルを編集することができます。

デプロイされたレポートの更新または削除

レポートをプロセスサーバーにデプロイした後に、プロセスコンソールの [カタログ] ページからレポートの XML ソースを表示できます。

デプロイされたレポート定義を更新または削除する手順

1. プロセスサーバーを起動し、コンソールを開きます。
2. コンソールで、[カタログ] > [リソース] > [レポート定義] に移動します。
3. 必要に応じて、レポートを選択し、編集します。
4. 必要に応じて、[更新] を選択して変更を送信するか、[削除] を選択します。

Reporting Service

プロセスサーバーは、Reporting Service をサポートしています。これは、サーバーに要求を送信して、デプロイされたレポートのスナップショットを作成し、レポートを PDF、MS Word、またはその他の形式で保存して、プロセスに返す BPEL プロセスを作成できることを意味します。このような操作を行うためには、Invoke アクティビティを使用します。レポートを電子メールで送信するため、ユーザーアクティビティで使用するため、または、呼び出されたサービスに基づいてエンドポイントへ送信するためのプログラミングロジックを BPEL プロセス内で指定します。プロセスを毎日（または他の頻度で）実行するようにスケジュールし、日次レポートを電子メールで送信することができます。

レポートベースのアクティビティが実装された BPEL プロセスは、*Informatica Business Manager* で提供される WSDL をインポートします。WSDL の名前は `reporting.wsdl` で、そのポートタイプである `reportingPT` は、[システムサービス] の下の [新規パートナーサービスプロバイダ] ウィザードの [パーティシパント] ビューにあります。WSDL にはレポート操作が含まれています。

```
<invoke
  inputVariable="reportMessageInput"
  operation="report"
  outputVariable="reportMessageOutput"
  partnerLink="Provider" />
```

レポートを要求する BPEL プロセスを作成する手順

1. プロジェクトエクスプローラで、新しい Orchestration プロジェクトを作成します。
2. [ファイル] > [新規] > [BPEL プロセス] を選択します。
3. プロセスに名前を付けて、[完了] をクリックします。
4. [パーティシパント] ビューで、[新規パートナーサービスプロバイダ] を選択します。

5. ウィザードで、[システムサービス] を選択します。
6. [レポートの実行] を選択し、[完了] を選択します。
7. [パーティシパント] ビューから、[レポート] 操作をキャンバスにドラッグして、Invoke アクティビティを作成します。
8. 以下の説明に従って、[入力] 値を入力します。
9. アクティビティとプログラミングロジックをさらに追加して、レポートを電子メール、ユーザーアクティビティ、またはその他のサービスにコピーします。
ヒント: 添付ファイル関数 setAttachmentProperty を使用して、レポートが電子メールに添付された場合にわかりやすい名前を付けます。
10. 必要に応じて、WSDL フォールトメッセージのフォールト名 reportFault と変数を使用してフォールト処理を追加します。

プロセスコンソールで、[URN マッピング] ページに移動し、必要に応じて内部レポートの URL を更新します。デフォルトは localhost:8080 です。

レポートサービスに対する入力メッセージ

例に示すように、Invoke アクティビティの入力メッセージには次のパートがあります。

```
<ns2:reportMessageInput xmlns:ns2=
"http://schemas.active-endpoints.com/reporting/2009/05/reporting.xsd">
  <ns2:report>ServiceActivityDetail.rptdesign</ns2:report>
  <ns2:format>pdf</ns2:format>
  <ns2:parameters>
    <ns2:param name="Service_Name" value="OrderProcessSvc"/>
  </ns2:parameters>
  <ns2:otherOptions>
    <ns2:option name="BIRT Parameter" value="true"/>
  </ns2:otherOptions>
</ns2:reportMessageInput>
```

次のように、Invoke アクティビティの入力パラメータを指定します。

<report>	(必須)。レポートデザインテンプレートの名前。これは、プロセスサーバーにデPLOYされたレポートです。レポート名は、プロセスコンソールカタログに表示されます
<format>	(必須)。生成されたレポートの形式。有効な形式は次のとおりです。 - pdf (デフォルト) - doc (Word) - xls (Excel) - ppt (PowerPoint) - ps (PostScript)
<parameters>	それぞれのレポートには、レポートに入力される独自のパラメータのセットを含めることができます (例えば、デPLOYされたサービスの名前を指定して、実行中の特定のプロセスに関するレポートを取得するなど)。標準レポートのパラメータに関する説明を以下に示します。
<otherOptions>	レポートを作成するためのオプション。さまざまなオプションがあり、各オプションについては BIRT ドキュメントに記載されています。詳細については、このリンクを参照してください。 http://www.birt-exchange.org/devshare/deploying-birt-reports/492-birt-viewer-2-3-user-reference/#description

レポートとレポートのパラメータ

Reporting Service を使用して、独自のカスタムレポート、またはプロセスコンソールに表示される Informatica Business Process Manager の標準レポートにアクセスできます。

プロセスコンソールでは、Informatica Business Process Developer のレポートデザインのリストを表示できます。次に、reportMessageInput 変数の<report></report>要素で名前を付けるレポートデザインを選択します。

レポートデザイン名のリストを表示するには、次の手順を実行します。

1. サーバーが実行されていることを確認します。
 2. プロセスコンソールを開きます。
 3. **【カタログ】 > 【リソース】 > 【レポート定義】** を選択します。
 4. **【フィルタ】** テーブルで、**【システムの非表示】** のチェックボックスを無効にして、**【送信】** を選択します。
- 一部のレポートにはパラメータがあり、reportMessageInput 変数の<ael:param name="string" value="string"/>要素で使用できます。

次のレポートには、パラメータと有効な値が含まれています。

アクティブなタスク

- RP_State - (状態の完全なリストを表示するには、プロセスコンソールの **【アクティブなタスク】** ページに移動し、**【レポートの実行】** ボタンを選択します。RP_State 選択リストには、異なる 14 個の状態が表示されます。)
- RP_CreatedAfter - Date
- RP_CreatedBefore - Date
- RP_CompletedAfter - Date
- RP_CompletedBefore - Date
- RP_Name - String
- RP_Owner - String
- RP_SearchBy - String
- RP_OwnerType - integer
- RP_IdentityType - integer
- RP_NamesText - String

サーバーログ

- RP_LogLevel VERBOSE = 1, INFO = 5, WARNING = 6, ERROR = 7 CRITICAL = 8
- RP_LogSource USER = 8, MAINTENANCE = 9 and SYSTEM = 10
- RP_LogService ENGINE = 3, ALERT = 4, MESSAGING = 5, TASK = 6, VERBOSE = 7, EVENTING = 8, EMAIL = 9, IDENTITY = 10
- 1 以上の RP_LogEngineId 整数

[「電子メールサービス」 \(ページ 311\)](#)も参照してください。