HITACHI
Inspire the Next

Cosminexus V11 BPM/ESB 基盤 サービスプラットフォーム 開発ガイド 基本開発編

手引・操作書

3021-3-J43-80

前書き

■ 対象製品

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の前書きの対象製品の説明を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

Apache(R)，Apache Kafka(R)，および Kafka(R)は，Apache Software Foundation の米国およびその他の国における登録商標または商標です。

Linux は，Linus Torvalds 氏の米国およびその他の国における登録商標です。

Microsoft，Excel，Microsoft Edge，Visio，Windows，Windows Server は，マイクロソフト 企業グループの商標です。

Oracle(R)，Java，MySQL 及び NetSuite は，Oracle，その子会社及び関連会社の米国及びその他の国における登録商標です。

UNIX は，The Open Group の登録商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

Eclipse は，開発ツールプロバイダのオープンコミュニティである Eclipse Foundation, Inc.により構築された開発ツール統合のためのオープンプラットフォームです。

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

本ソフトウェアに含まれる第三者ソフトウェアに関する情報を次に示します。

第三者ソフトウェア：Trang

著作権者：Thai Open Source Software Center Ltd

当該ソフトウェアのライセンス条文：以下の通り。

Copyright (c) 2002, 2003, 2008 Thai Open Source Software Center Ltd

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Thai Open Source Software Center Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

■ マイクロソフト製品のスクリーンショットの使用について

マイクロソフトの許可を得て使用しています。

■ ランゲージパックの使用について

このマニュアルでは，Eclipse Babel Project が提供する BABEL 日本語ランゲージパックを適用した Eclipse メニュー表記を使用しています。使用するランゲージパックのバージョンによっては，マニュアル内の表記と異なる場合があります。

■ 発行

2026 年 1 月 3021-3-J43-80

■ 著作権

All Rights Reserved. Copyright (C) 2020, 2026, Hitachi, Ltd.

変更内容

変更内容(3021-3-J43-80) uCosminexus Service Architect 11-70，uCosminexus Service Platform 11-70

追加・変更内容	変更個所
Service Platform および Service Architect から次の適用 OS を削除した。 • Windows 10 x64 • Red Hat Enterprise Linux 8（AMD/Intel 64）	－
Service Architect で，JDK の種類に応じたプラグイン jar が自動的にインストールされるようにした。 これに伴い，開発環境を使用するためのセットアップ手順を変更した。	2.1
Service Architect のインストール時に，JDK の種類に応じた eclipse.ini の編集が自動で実行されるようにした。 これに伴い，開発環境を使用するためのセットアップ手順を変更した。	2.1
JDK 21 はサービスプラットフォームで使用できない旨を記載した。	2.1.2
繰り返しアクティビティのリスト指定方式で，永続化するビジネスプロセスを指定できるようにした。 これに伴い，永続化するビジネスプロセスについての注意事項を削除した。	5.2.1(3)，5.6.16(4)
繰り返しアクティビティの［繰り返し条件設定］ダイアログの XPath 式を［条件式の作成］ダイアログで指定できるようにした。	5.5.1(6)，5.6.16(2)，5.6.21，5.6.21(1)，5.6.21(2)，5.11.1(3)
XML スキーマを利用した JSON-XML 変換（拡張モード）ができるようにした。 これに伴い，次の点を修正した。 • 拡張モードに対し，既存の JSON-XML 変換を標準モードとして区別した。 • cscjson2xsdadv コマンドと cscjsonxmlconvadv コマンドの説明を追加した。	11.1，11.1.2，12 章，13.2，14.1，14.1.1，14.1.2
JSON-XML 変換の小数点を含む数値の出力結果を指数形式から小数点形式へ変更した。 これに伴い，次の点を修正した。 • setDecimalXmloutputControl メソッドの説明を追加した。 • json-xml-convert.decimal-xmloutput-control プロパティの説明を追加した。	11.3，11.3.6，13.2.1，13.2.1(6)，13.2.3(2)，14.2.1，14.2.3

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルをお読みになる際の前提情報については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」のはじめにの説明を参照してください。

目次

前書き　2
変更内容　4
はじめに　5

1　SOA に基づいたシステム開発の概要　21
1.1　SOA に基づいたシステム開発の手順　22
1.1.1　開発から実運用までの流れ　22
1.2　システム全体と開発環境の関係　29
1.2.1　運用上の位置づけ　29
1.2.2　ネットワーク上の位置づけ　30

2　システムを開発する前に　31
2.1　開発環境を使用するためのセットアップ　32
2.1.1　前提条件　33
2.1.2　インストール　34
2.1.3　環境設定　34
2.1.4　組み込みデータベースの構築　35
2.1.5　アンインストール　38
2.2　構成の形態，J2EE サーバの互換モード，および SOAP モードの選択　40
2.2.1　データベースと Reliable Messaging の使用有無　40
2.2.2　使用する SOAP モード　42
2.2.3　J2EE サーバの互換モードの選択　44
2.2.4　SOAP モードの選択　45
2.3　サービスプラットフォームを利用したシステムの開発　46
2.3.1　開発手順　46
2.4　テスト環境の簡易セットアップ　51
2.4.1　HCSC 簡易セットアップ機能で構築できる環境　51
2.4.2　HCSC 簡易セットアップ機能の実行　53
2.4.3　HCSC 簡易セットアップ機能でセットアップしたテスト環境の運用　63
2.4.4　テスト環境のカスタマイズ　69
2.4.5　SOAP モードの確認　75
2.5　HCSC 簡易セットアップ機能でセットアップした組み込みデータベースの運用　77
2.5.1　RD エリアの未使用領域を確認する　77
2.5.2　プロセスインスタンスの実行履歴を削除する　78
2.5.3　空きページおよび空きセグメントを解放する　79

2.6 利用できるサービス部品の種類と適用範囲 81
2.6.1 Web サービスを利用したサービス部品の適用範囲 81
2.6.2 SessionBean を利用したサービス部品の適用範囲 89
2.6.3 Reliable Messaging のローカルキューを利用したサービス部品の適用範囲 93
2.6.4 DB キューを利用したサービス部品の適用範囲 94
2.6.5 XML スキーマの適用範囲 94
2.7 コード変換機能の拡張（Code Converter 使用時） 97
2.7.1 Code Converter による拡張 97
2.7.2 文字コード変換 UOC による拡張 99

3 プロジェクトの管理とリポジトリの管理 101
3.1 プロジェクトの管理 102
3.1.1 プロジェクトを作成する 102
3.1.2 プロパティを設定する 106
3.1.3 プロジェクトをエクスポート/インポートする 108
3.1.4 プロジェクトを削除する 108
3.1.5 SOAP モードを変更する 109
3.1.6 開発時の注意事項 109
3.2 リポジトリの管理 110
3.2.1 リポジトリを設定する 110
3.2.2 リポジトリをエクスポートする 112
3.2.3 リポジトリをインポートする 116
3.3 設計情報の出力 128
3.3.1 設計情報として出力できる情報 128
3.3.2 出力前の確認事項 138
3.3.3 設計情報の出力方法 139
3.3.4 設計情報の出力に関する注意事項 143
3.4 差分確認用ドキュメントの出力 144
3.4.1 差分確認用ドキュメント出力機能の概要 144
3.4.2 差分確認用ドキュメント出力機能の実行手順 146
3.4.3 差分確認用ドキュメントが出力される出力ディレクトリの構成 146
3.4.4 差分確認用ドキュメント出力機能の実行例 147
3.4.5 差分確認用ドキュメント出力機能の注意事項 151
3.5 Eclipse に関する注意事項 152

4 電文フォーマットの作成 153
4.1 電文フォーマットとデータ変換 154
4.1.1 電文の種類 154
4.1.2 電文フォーマット 154

4.1.3 データ変換 154
4.2 電文フォーマットの種類 156
4.2.1 XML 形式のデータを扱う場合 156
4.2.2 バイナリ形式のデータを扱う場合 156
4.3 電文フォーマット（XML フォーマット定義ファイル）の作成方法 157
4.3.1 標準電文を作成する 157
4.3.2 サービス部品電文を作成する（Web サービスの場合） 157
4.3.3 サービス部品電文を作成する（SessionBean の場合） 158
4.3.4 サービス部品電文を作成する（MDB（WS-R または DB キュー）の場合） 158
4.4 電文フォーマット（バイナリフォーマット定義ファイル）の作成方法 159
4.4.1 バイナリフォーマット定義ファイルのデータ型と文字コードの種類 160
4.4.2 文字コード変換の種類 170
4.4.3 バイナリフォーマット定義ファイルを新規作成する 170
4.4.4 要素を定義する（CSV 以外の形式の場合） 171
4.4.5 要素を定義する（CSV 形式の場合） 178
4.4.6 バイナリフォーマット定義ファイルを編集する 180
4.4.7 バイナリフォーマット定義ファイルを検証する 180
4.4.8 バイナリフォーマット定義の注意事項 198
4.5 COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成 199
4.5.1 COBOL 登録集原文ファイルの規則 199
4.5.2 データ項目の設定（COBOL85 言語の形式を使用する場合） 204
4.5.3 データ項目の設定（サービスプラットフォームで独自に拡張した形式を使用する場合） 208
4.5.4 バイナリフォーマット定義ファイルの生成方法 229
4.6 バイナリフォーマット定義ファイルから XML スキーマファイルの生成 232
4.7 電文フォーマットの変更 233

5 ビジネスプロセスの定義 234
5.1 ビジネスプロセスの定義の手順 235
5.1.1 定義の流れ 235
5.2 ビジネスプロセスの追加 238
5.2.1 ビジネスプロセスを新規に追加する 238
5.2.2 定義済みビジネスプロセスを利用してビジネスプロセスを追加する 241
5.2.3 ビジネスプロセス追加後に設定する項目 241
5.3 ビジネスプロセスの内容の定義 243
5.3.1 アクティビティ 243
5.3.2 変数・相関セット 244
5.4 アクティビティの配置と連結 245
5.4.1 アクティビティを配置する 245
5.4.2 アクティビティを連結する 246

5.4.3 アクティビティのオートコネクト 250
5.4.4 フォルト処理を定義する 252
5.4.5 補償処理を定義する 258
5.5 変数・相関セットの定義 264
5.5.1 変数の定義 264
5.5.2 相関セットの定義 274
5.6 アクティビティの定義 284
5.6.1 開始アクティビティ 285
5.6.2 受付アクティビティ 285
5.6.3 応答アクティビティ 287
5.6.4 サービス呼出アクティビティ 289
5.6.5 Java 呼出アクティビティ 291
5.6.6 データ変換アクティビティ 306
5.6.7 代入アクティビティ 308
5.6.8 無操作アクティビティ 312
5.6.9 フォルト送出アクティビティ 313
5.6.10 補償アクティビティ 316
5.6.11 待機アクティビティ 322
5.6.12 検証アクティビティ 327
5.6.13 ログ出力アクティビティ 331
5.6.14 途中終了アクティビティ 331
5.6.15 スコープアクティビティ 332
5.6.16 繰り返しアクティビティ 335
5.6.17 分岐アクティビティ 341
5.6.18 並列処理アクティビティ 344
5.6.19 終了アクティビティ 350
5.6.20 順次処理アクティビティ 350
5.6.21 アクティビティ定義ダイアログの XPath の指定 350
5.7 コメントの配置 364
5.8 ビジネスプロセスの保存 365
5.9 ビジネスプロセスの編集 366
5.9.1 ビジネスプロセスおよびアクティビティの定義情報を変更する 366
5.9.2 アクティビティ名を変更する 367
5.9.3 運用中のビジネスプロセス定義を変更する 368
5.9.4 ビジネスプロセスのバージョンアップ 371
5.10 ビジネスプロセスのテスト 383
5.10.1 ビジネスプロセス実行コマンド使用時の前提条件 383
5.10.2 リクエストの送信 383
5.10.3 クライアント相関 ID の設定 384

5.11 ビジネスプロセスの検証 385
5.11.1 検証内容 385
5.11.2 検証方法 400
5.11.3 検証内容の表示 401
5.12 ビジネスプロセスの削除 402

6 サービス定義のインポート・エクスポート 403
6.1 サービス定義のインポート・エクスポートの概要 404
6.2 サービス定義のインポート・エクスポートの使用の目的 406
6.3 サービス定義 XML ファイルの内容 408
6.3.1 サービス定義 XML ファイルでのビジネスプロセス定義の考え方 408
6.3.2 サービス定義 XML ファイルの作成と指定内容 411
6.3.3 サービス定義 XML ファイルのルートを指定する要素 413
6.3.4 ビジネスプロセスを指定する要素 413
6.3.5 ユーザ定義受付（呼出先固定）の一覧を指定する要素 438
6.3.6 サービスアダプタを指定する要素 446
6.3.7 ユーザ定義受付（呼出先選択）を指定する要素 458
6.3.8 ビジネスプロセスの描画情報を指定する要素 459
6.4 サービス定義のインポート 464
6.4.1 サービス定義 XML ファイルのインポート手順 464
6.4.2 ビジネスプロセスの描画情報の指定 467
6.4.3 データ変換アクティビティに指定するマッピング定義の形式 467
6.4.4 マッピング定義ファイルを利用する際の注意事項 468
6.5 サービス定義のエクスポート 470
6.5.1 サービス定義 XML ファイルのエクスポート手順 470
6.5.2 マッピング定義のエクスポート形式 473
6.5.3 ビジネスプロセスの描画情報ファイルの出力有無 475
6.5.4 サービス定義がエクスポートされる出力ディレクトリの構成 475
6.5.5 エクスポート実行時に削除されるディレクトリ 486
6.5.6 注意事項（サービス定義のエクスポート） 487

7 データ変換の定義 491
7.1 データ変換に必要なファイルおよび定義 492
7.2 電文フォーマット定義ファイルの作成 493
7.2.1 電文フォーマット定義ファイルの作成 493
7.2.2 電文フォーマット定義ファイルの設定 493
7.2.3 電文フォーマット定義ファイル作成時の注意事項 493
7.2.4 セパレータが設定されているバイナリ電文フォーマットに関する注意事項 494
7.2.5 任意形式（any 形式）の電文フォーマットに関する注意事項 494

7.3 データ変換定義 496
7.3.1 データ変換の定義手順 496
7.3.2 電文フォーマットを変更した場合の定義手順 502
7.3.3 データ変換定義の注意事項 503
7.4 マッピングする 505
7.4.1 変換元ノードの値を変換先ノードに直接マッピングする 506
7.4.2 変換元ノードの値を加工して変換先ノードにマッピングする 507
7.4.3 変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする 512
7.4.4 変換先ノードの要素から候補を選択して自動でマッピングする 513
7.4.5 マッピングを取り消す 514
7.4.6 マッピング元の表示形式 515
7.4.7 マッピング線とファンクションを見やすくする 516
7.4.8 マッピング対象の範囲を限定する 517
7.4.9 自動マッピング時の類似度の判定 518
7.4.10 マッピングの注意事項 519
7.5 ファンクションを使用した値またはノードの加工 521
7.5.1 複数の文字列を連結する 522
7.5.2 1 つの文字列から一部分の文字列を取り出す 523
7.5.3 文字列の文字数を代入する 524
7.5.4 指定文字列の有無，または指定文字列の開始を調査する 525
7.5.5 文字列の空白を除去する 527
7.5.6 数値をフォーマット変換する 529
7.5.7 数値を演算する 530
7.5.8 小数点以下の端数を処理する 531
7.5.9 複数のノードセットのノードの数値を合計する 533
7.5.10 否定演算を使用する 534
7.5.11 論理演算を使用する 535
7.5.12 シフト演算を使用する 537
7.5.13 ノード数を代入する 540
7.5.14 ノード名を代入する 541
7.5.15 ノードの有無を調査する 542
7.5.16 くり返しを対応づける 543
7.5.17 条件によって異なる値を出力する 546
7.5.18 指定した値を代入する 549
7.5.19 変換テーブルを利用して値を変換する 550
7.5.20 基数変換する 554
7.5.21 変換元ノードの値にある値を付加する 557
7.5.22 変換元ノードの値を倍にする 558
7.5.23 ユーザが作成した任意の Java プログラムを呼び出す 559

7.5.24 変換先ノードやファンクションの入力値に XPath 式を指定する 561
7.6 くり返しの設定 564
7.6.1 ［くり返し設定］ダイアログを使用してマッピングする 564
7.6.2 くり返しを合成する 565
7.6.3 くり返し依存対象の対応づけ 566
7.6.4 くり返し依存対象を確認する 570
7.6.5 くり返し依存対象を変更する 571
7.6.6 くり返し依存対象が設定された変換元ノードのパスの表示 571
7.6.7 連携パスを設定して各要素のくり返し処理を連動させる 575
7.7 ノード条件を設定する 581
7.8 マッピング定義をコピーする 583
7.8.1 マッピング定義をコピーする流れ 584
7.8.2 マッピング定義の保存 584
7.8.3 マッピング定義の登録 584
7.8.4 マッピング定義のコピー 585
7.8.5 類似度の判定方法 586
7.8.6 マッピング定義をコピーする場合の注意事項 588
7.9 ファンクションをコピーして貼り付ける 592
7.9.1 ファンクションのコピー 592
7.9.2 ファンクションの貼り付け 594
7.10 カスタムファンクションで使用する Java プログラムの作成 596
7.10.1 変換ファンクション定義ファイルの作成 597
7.10.2 Java 雛形ファイルの生成 601
7.10.3 変換ファンクションからの外部 jar の参照 603
7.10.4 Java プログラムのコーディング，ビルド，およびデバッグ 603
7.10.5 Java プログラムのパッケージング 604
7.11 マッピングの条件 605
7.11.1 マッピング対象とマッピング対象外 605
7.11.2 マッピングできるノードおよびファンクションの対応 606
7.11.3 接続できるマッピング線の数 615
7.12 ファンクション名の直接編集 617
7.12.1 ファンクション名の編集方法 617
7.12.2 編集後のファンクション名の表示 618
7.13 Excel を利用したマッピング定義のインポート 620
7.13.1 表形式 XML ファイルの作成 621
7.13.2 マッピング定義の設定 623
7.13.3 マッピング定義のインポート 638
7.14 Excel を利用したマッピング定義のエクスポート 639
7.14.1 マッピング定義のエクスポート 640

7.14.2 表形式 XML ファイルのインポート 641
7.15 表形式 XML スキーマ定義ファイルの定義詳細 642
7.15.1 Namespaces（名前空間情報） 643
7.15.2 CopyObjects（変換先ノードへのマッピング） 645
7.15.3 ConcatenateObjects（文字列連結ファンクション） 647
7.15.4 SubstringObjects（部分文字列取得ファンクション） 648
7.15.5 LengthObjects（文字数取得ファンクション） 651
7.15.6 ContainObjects（文字列調査ファンクション） 652
7.15.7 TrimObjects（空白除去ファンクション） 653
7.15.8 FormatObjects（数値フォーマット変換ファンクション） 654
7.15.9 CalculateObjects（数値演算ファンクション） 656
7.15.10 RoundObjects（端数処理ファンクション） 657
7.15.11 SumObjects（総和ファンクション） 659
7.15.12 NotObjects（否定演算ファンクション） 660
7.15.13 BitOpObjects（論理演算ファンクション） 660
7.15.14 ShiftObjects（シフト演算ファンクション） 662
7.15.15 NameObjects（ノード名取得ファンクション） 663
7.15.16 CountObjects（ノード数取得ファンクション） 664
7.15.17 ExistObjects（ノード調査ファンクション） 665
7.15.18 LoopObjects（くり返しファンクション） 666
7.15.19 ChooseObjects（選択ファンクション） 669
7.15.20 ReplaceObjects（値置換えファンクション） 671
7.15.21 RadixObjects（基数変換ファンクション） 672
7.15.22 CustomObjects（カスタムファンクション） 673
7.15.23 ConstantObjects（定数ファンクション） 676
7.15.24 XpathObjects（XPath ファンクション） 677
7.15.25 同一の Name 要素を複数行に定義できるオブジェクト 678
7.16 Excel を利用したマッピング定義時の注意事項 680
7.17 名前空間接頭辞オプション 682
7.17.1 名前空間接頭辞オプションのデフォルト値設定 682
7.18 提供しているカスタムファンクション 683
7.18.1 getWorkFolderPath（作業フォルダパス取得関数） 683
7.18.2 currentDateTime（システム日付取得関数） 684
7.18.3 lrtrim（前後空白除去関数） 685
7.18.4 getSystemEnv（環境変数取得関数） 686
7.18.5 getPropertiesByGroup（外部プロパティ取込関数） 687
7.18.6 使用手順 689
7.19 XPath ファンクションの使用方法 694
7.19.1 変換先ノードへの代入 694

7.19.2 ファンクションへの代入 698
7.19.3 くり返し出現するノードの選択 698
7.19.4 くり返しファンクションとの連携 699

8 HCSC コンポーネントのパッケージングと配備定義 707
8.1 パッケージングと配備定義の概要 708
8.2 パッケージング 710
8.3 HCSC コンポーネントの配備定義 711
8.3.1 配備できるクラスタ（または単一の HCSC サーバ） 713
8.3.2 クラスタに HCSC コンポーネントを追加する 714
8.3.3 クラスタから HCSC コンポーネントを削除する 715
8.4 HCSC コンポーネントの情報の参照 716
8.4.1 参照できる HCSC コンポーネントの情報 716
8.4.2 HCSC コンポーネントの情報を表示する 717
8.4.3 HCSC コンポーネントの一覧を更新する 717
8.5 HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行 719
8.5.1 配備から開始までの処理の流れ 719
8.5.2 HCSC コンポーネントを HCSC サーバに配備して開始する方法 721
8.6 HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行 727
8.6.1 停止から削除までの処理の流れ 727
8.6.2 HCSC コンポーネントを停止して HCSC サーバから削除する方法 729

9 サービスリクエスタの作成 735
9.1 サービスリクエスタ作成の概要 736
9.2 標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（SOAP 通信基盤） 738
9.2.1 サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（SOAP 通信基盤） 738
9.2.2 WSDL を取得する 740
9.2.3 スタブを生成する 741
9.2.4 オブジェクトを生成する 742
9.2.5 パラメタを設定する 745
9.2.6 要求電文を作成する 746
9.2.7 応答電文を取得する 748
9.2.8 エラー情報を取得する 750
9.2.9 ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤） 754
9.2.10 サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤） 757
9.3 標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（JAX-WS エンジン） 766
9.3.1 サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（JAX-WS エンジン） 766
9.3.2 WSDL を取得する 768

9.3.3 サービスクラスを生成する 769
9.3.4 オブジェクトを生成する 771
9.3.5 パラメタを設定する 772
9.3.6 要求電文を作成する 772
9.3.7 応答電文を取得する 772
9.3.8 エラー情報を取得する 772
9.3.9 ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン） 780
9.3.10 サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン） 782
9.4 標準の同期受付（SessionBean）に要求を出すサービスリクエスタ 784
9.4.1 サービスリクエスタ（標準の同期受付（SessionBean））の作成手順 784
9.4.2 スタブを取得する 786
9.4.3 インスタンスを生成する 786
9.4.4 パラメタを設定する 788
9.4.5 要求電文を作成する 789
9.4.6 応答電文を取得する 790
9.4.7 エラー情報を取得する 790
9.4.8 ビジネスプロセスの再実行を要求するサービスリクエスタの作成（SessionBean） 797
9.5 標準の非同期受付（MDB（WS-R））に要求を出すサービスリクエスタ 800
9.5.1 サービスリクエスタ（標準の非同期受付（MDB（WS-R））の作成手順 800
9.5.2 転送キューを作成する 802
9.5.3 JMS メッセージを生成する 803
9.5.4 パラメタを設定する 804
9.5.5 要求電文を作成する 808
9.5.6 JMS メッセージを送信する 809
9.5.7 応答用のキューを設定する 810
9.5.8 応答を取り出す 811
9.5.9 応答電文を取得する 812
9.6 標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ 816
9.6.1 サービスリクエスタ（標準の非同期受付（MDB（DB キュー））の作成手順 817
9.6.2 送信用共用キューを作成する（JMS） 821
9.6.3 JMS メッセージを生成する（JMS） 822
9.6.4 バイナリデータを作成する（TP1/EE，JMS） 822
9.6.5 パラメタを設定する（TP1/EE，JMS） 826
9.6.6 要求電文を作成する（TP1/EE，JMS） 826
9.6.7 バイナリデータを受信用共用キューに設定する（TP1/EE） 828
9.6.8 JMS メッセージを送信する（JMS） 828
9.6.9 応答用のキューを設定する（TP1/EE，JMS） 828
9.6.10 応答を取り出す（TP1/EE，JMS） 830

9.6.11 応答電文を取得する（TP1/EE，JMS） 833
9.7 SOAP 受付に要求を出すサービスリクエスタ 834
9.7.1 サービスリクエスタ（SOAP 受付）の作成手順（SOAP 通信基盤） 834
9.7.2 WSDL を編集する 836
9.7.3 スタブを生成する 836
9.7.4 オブジェクトを生成する 837
9.7.5 応答電文を取得する 837
9.7.6 エラー情報を取得する 838
9.7.7 サービスリクエスタ（SOAP 受付）の作成手順（JAX-WS エンジン） 841

10 ビジネスプロセスのデバッグ 845
10.1 ビジネスプロセスのデバッグの手順 846
10.1.1 デバッグの流れ 846
10.2 ビジネスプロセスのデバッグの準備 849
10.2.1 ブレークポイントの設定 849
10.2.2 サービスのエミュレーションの設定 851
10.3 ビジネスプロセスのデバッグの開始 853
10.4 リクエストの送信 857
10.5 ビジネスプロセスのデバッグで実施できる操作 858
10.5.1 ステップ実行および再開 858
10.5.2 変数・相関セットの確認 871
10.5.3 変数の更新 871
10.5.4 XPath の評価 873
10.5.5 サービスの自動エミュレーション 874
10.5.6 サービスの手動エミュレーション 876
10.5.7 cscgenbinary コマンドの使用方法 877
10.6 ビジネスプロセスのデバッグの終了 885

11 JSON データの変換 886
11.1 JSON-XML 変換の概要 887
11.1.1 JSON-XML 変換で扱う JSON と標準仕様の JSON との差異 887
11.1.2 標準モードと拡張モードとの差異 888
11.2 JSON-XML 変換の規則 892
11.2.1 XML 宣言 892
11.2.2 XML 形式データのルート要素 892
11.2.3 オブジェクト型の変換 893
11.2.4 配列の変換 895
11.2.5 null 値の変換 896
11.2.6 文字列型の変換 897

11.2.7 数値型の変換 899
11.2.8 boolean 型の変換 900
11.2.9 XML 形式データの属性 901
11.2.10 JSON 形式から XML 形式に変換する際のオブジェクト名の順序付け 902
11.3 JSON-XML 変換定義ファイルの設定 904
11.3.1 csc-array 要素の要素名の変更 904
11.3.2 csc-element 要素の要素名の変更 904
11.3.3 csc-simple 要素の要素名の変更 905
11.3.4 csc-object 要素の要素名の変更 905
11.3.5 JSON のデータ種別を XML の属性に含めるかどうかの設定 905
11.3.6 小数点付き数値項目の出力形式の設定 906
11.4 JSON-XML 変換に関する注意事項 907
11.4.1 XML 形式のデータから JSON 形式のデータに変換する場合の注意事項 907
11.4.2 JSON 形式のデータと XML 形式のデータを相互変換する場合の注意事項 907

12 拡張モードでの JSON データの変換 908
12.1 拡張モードの JSON-XML 変換の概要 909
12.2 拡張モードの JSON-XML 変換の設定 910
12.2.1 OSS のインストール 910
12.2.2 クラスパスの設定 910
12.2.3 拡張モードの JSON-XML 変換の制限事項 911
12.3 拡張モードの JSON-XML 変換の規則 912
12.3.1 JSON→XSD 変換の規則 912
12.3.2 JSON→XML 変換の規則 921
12.3.3 XML→JSON 変換の規則 934
12.4 拡張モードの JSON-XML 変換 API の使用例 946
12.4.1 JSON→XSD 変換 946
12.4.2 JSON→XML 変換 948
12.4.3 XML→JSON 変換 949

13 API を使用したデータ変換 952
13.1 データ変換 API を使用したデータ変換 953
13.1.1 データ変換 API を使用したデータ変換の概要 953
13.1.2 データ変換 API の機能 953
13.1.3 データ変換 API の詳細 961
13.1.4 データ変換 API の使用方法 981
13.1.5 データ変換 API の使用例 986
13.1.6 データ変換時のボトルネックの特定 992
13.1.7 データ変換 API の注意事項 992

13.2 JSON-XML 変換 API を使用したデータの形式変換 993
13.2.1 JSON-XML 変換 API 変換定義の設定 994
13.2.2 JSON-XML 変換 API の障害対策 997
13.2.3 JSON-XML 変換 API の詳細 998
13.2.4 JSON-XML 変換 API の使用方法 1008
13.2.5 JSON-XML 変換 API の使用例 1009

14 JSON 変換ツールを使用したデータ変換 1012
14.1 JSON 変換ツールの概要 1013
14.1.1 JSON 形式データに対応する XML スキーマの生成 1015
14.1.2 JSON-XML 変換結果の確認 1015
14.2 変換動作の変更オプション 1016
14.2.1 設定できるプロパティ 1016
14.2.2 プロパティファイルの指定方法 1016
14.2.3 プロパティの指定に関する注意事項 1017
14.3 効率良く XML スキーマを生成する方法 1018
14.3.1 出現回数の制御 1018
14.3.2 データ型の制御 1019
14.4 JSON 変換ツールの実行例 1022
14.4.1 実行例 1 1022
14.4.2 実行例 2 1024
14.5 JSON 変換ツールの注意事項 1027

付録 1031
付録 A 旧バージョンからの移行 1032
付録 A.1 移行の対象となるバージョン 1032
付録 A.2 旧バージョンからの移行手順 1033
付録 A.3 旧バージョンで開発環境と運用環境のリポジトリを共有していた場合の移行手順 1037
付録 B 評価版から製品版への移行 1041
付録 B.1 評価版からの移行 1041
付録 C 上流設計ツールを利用したシステムの開発 1044
付録 C.1 上流設計ツールを利用したシステム開発の概要 1044
付録 C.2 上流設計ツールを利用したシステム開発の流れ 1046
付録 C.3 上流設計ツール利用時の前提事項 1048
付録 C.4 上流設計ツール利用時のトラブルシューティング 1051
付録 D 上流設計ツールを利用したシステムの開発例 1071
付録 D.1 ビジネスプロセスの概要を設計する 1073
付録 D.2 サービスの概要インターフェースを検討する 1075
付録 D.3 ビジネスプロセスを詳細化する 1078

付録 D.4　ビジネスプロセスを出力する　1084
付録 D.5　サービスの詳細インターフェースを検討する　1085
付録 D.6　サービスアダプタを作成する　1087
付録 D.7　ビジネスプロセスをインポートする　1090
付録 D.8　ユーザ定義受付を追加する　1091
付録 D.9　メッセージのスキーマを登録する　1093
付録 D.10　メッセージ変換とシステム例外処理を追加する　1098
付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲　1099
付録 E.1　BPEL1.1 のビジネスプロセス定義をインポートする場合　1100
付録 E.2　BPEL2.0 のビジネスプロセス定義をインポートする場合　1118
付録 F　サービスアダプタを利用した HTTP ヘッダおよび Cookie 情報の引き継ぎ　1141
付録 F.1　アクティビティの定義　1142
付録 F.2　ヘッダ変数のスキーマ作成　1142
付録 F.3　ヘッダ変数の作成　1148
付録 F.4　サービス呼出アクティビティへのヘッダ割当変数の設定　1149
付録 F.5　変換元変数と変換先変数のマッピング　1149
付録 G　コンポーネント共通 UOC　1151
付録 G.1　コンポーネント共通 UOC クラスのプロパティファイル　1151
付録 G.2　コンポーネント共通 UOC クラスの指定方法　1151
付録 G.3　UOC クラス用 API　1153
付録 G.4　電文取得用 API　1156
付録 G.5　UOC クラスの jar ファイル指定方法　1158
付録 G.6　注意事項　1158
付録 H　文字コード変換 UOC を利用した文字コード変換　1160
付録 H.1　文字コード変換 UOC の jar ファイルの開発　1160
付録 H.2　文字コード変換 UOC を使用するための設定　1160
付録 H.3　CSCOwnCodeConverter インターフェース　1162
付録 H.4　CSCOwnCodeReader インターフェース　1168
付録 H.5　CSCOwnCodeReaderContext インターフェース　1181
付録 H.6　注意事項　1188
付録 I　DB アダプタで取得したデータのフォーマットの変換例　1189
付録 I.1　例題　1189
付録 I.2　フォーマットの変換方法　1190
付録 J　DB アダプタで取得したデータの自動マッピング　1196
付録 J.1　カラム定義ファイルの作成　1196
付録 J.2　データ変換定義画面でのマッピング　1199
付録 K　外部バインディングファイルを利用した WSDL のカスタマイズ　1202
付録 L　IBM 漢字コードの文字セットの変更　1203
付録 L.1　IBM 漢字コードの文字セットを変更する手順　1203

付録 M　サイズノードの機能拡張　1204
付録 M.1　サイズノードにサイズノード自身のサイズを含めた値を指定する　1204
付録 M.2　サイズノードが表すデータサイズの単位をバイト数から桁数へ変更する　1207
付録 M.3　サイズノードにビット列型のノードを指定する　1209
付録 N　選択要素なし（入出力データが 0 バイトとなるケース）の選択構成の定義　1212
付録 N.1　前提条件　1212
付録 N.2　設定方法　1213
付録 N.3　注意事項　1214
付録 O　バイナリデータの未定義コードを置換する文字の定義　1217
付録 O.1　前提条件　1217
付録 O.2　検知した未定義コードを指定した文字に置換する処理の流れ　1217
付録 O.3　置換文字コードを指定する J2EE サーバ用ユーザプロパティ　1218
付録 O.4　未定義コードの任意文字置換時の注意事項　1219
付録 O.5　置換文字コードの設定例　1219
付録 P　バイナリデータ内に文字コードが混在する場合の対応　1224
付録 P.1　個別文字コードが指定できるデータ種別および指定可能な文字コード　1224
付録 P.2　個別文字コードの指定方法　1224
付録 P.3　個別文字コード指定とセパレータを併用する場合の注意事項　1225
付録 Q　CSV 形式の単純内容要素のデータなしの区別　1227
付録 Q.1　空データと Null データを区別するための処理の概要　1227
付録 Q.2　空データと Null データを区別するための設定　1229
付録 Q.3　CSV 形式のデータおよび DOM の扱われ方　1231
付録 Q.4　CSV 形式のデータと DOM の相互変換　1233
付録 R　LOW-VALUE（0x00 で埋められたデータ）と空要素を相互に変換する場合の対応　1238
付録 R.1　前提条件　1238
付録 R.2　設定方法　1238
付録 S　用語解説　1240

索引　1241

1 SOA に基づいたシステム開発の概要

この章では，SOA に基づいたシステム開発の概要について説明します。

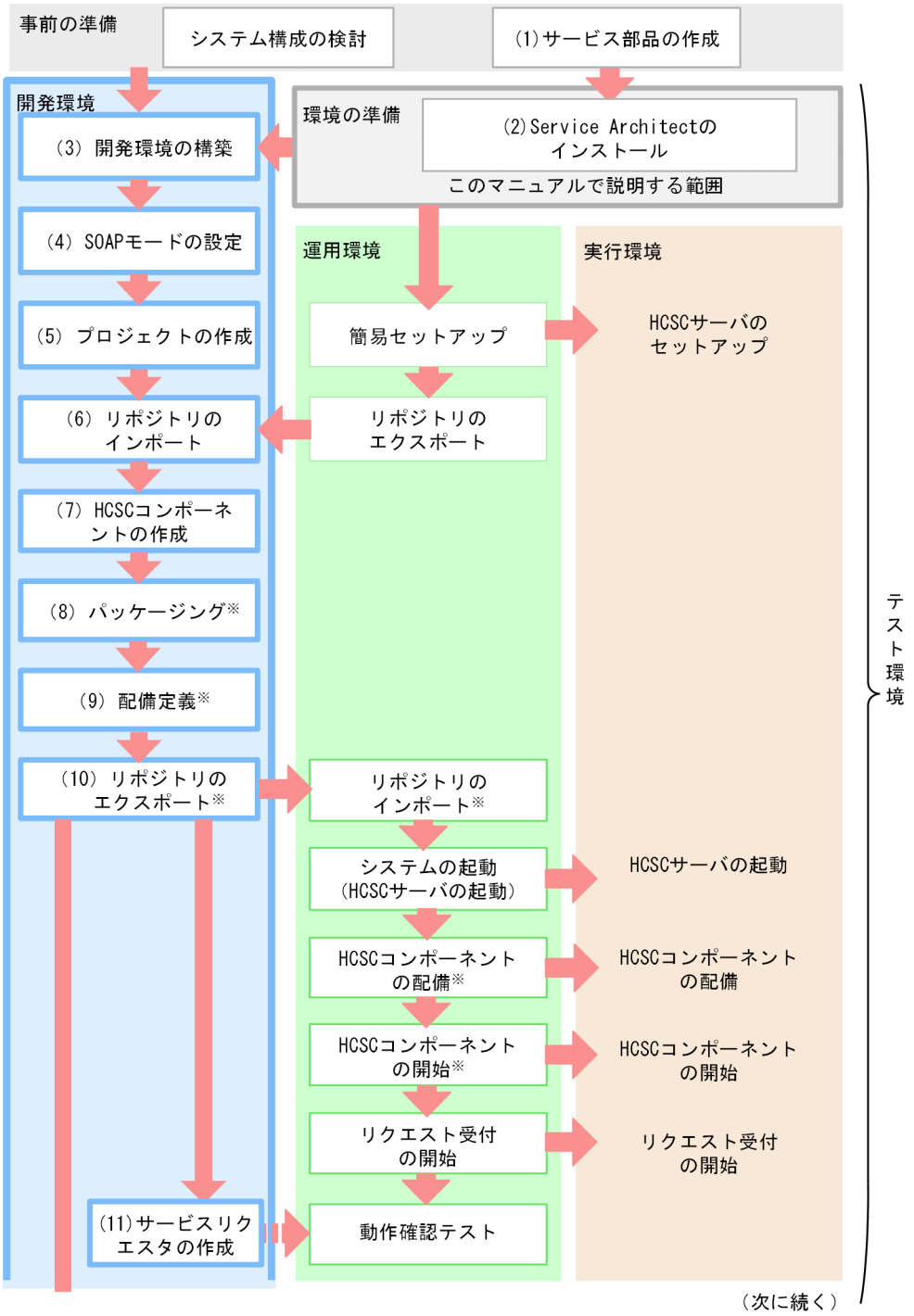
1.1　SOA に基づいたシステム開発の手順

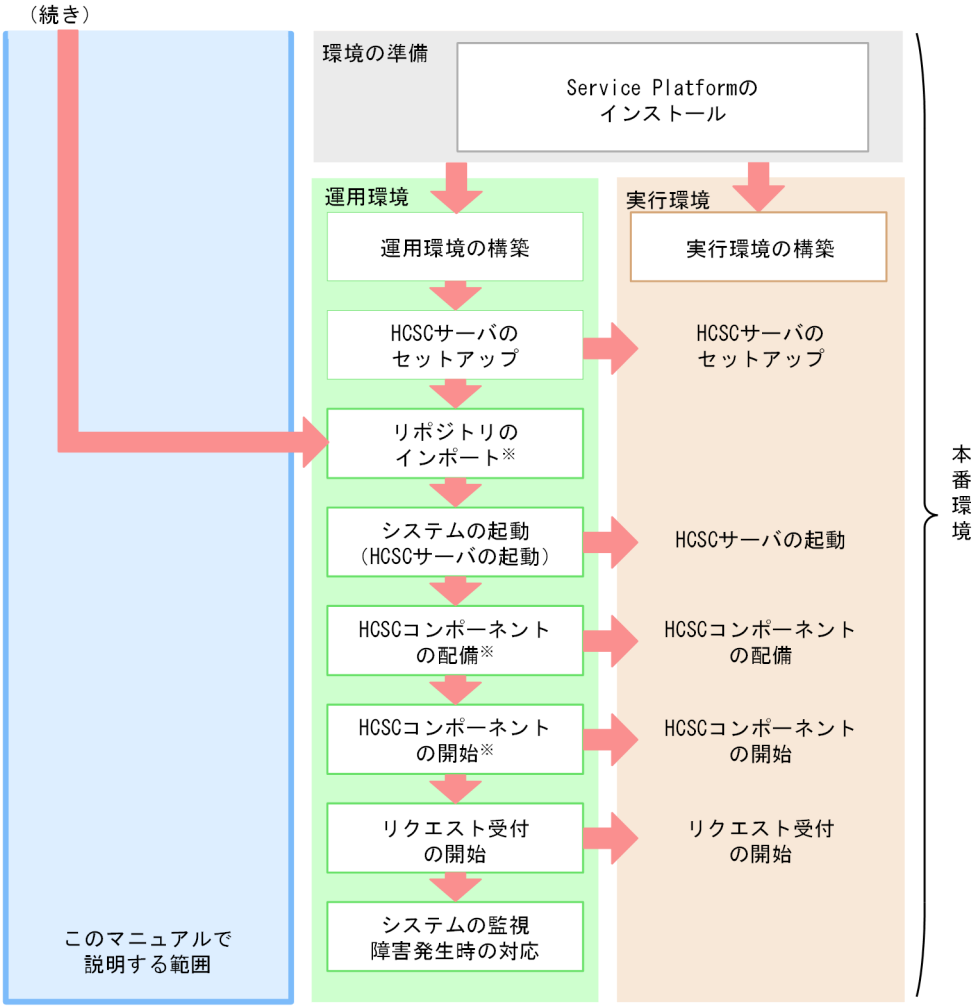
サービスプラットフォームを利用したシステムの開発から実運用までの手順について説明します。

1.1.1　開発から実運用までの流れ

サービスプラットフォームを利用したシステムの開発から実運用までの流れを次の図に示します。

図 1-1　開発から実運用までの流れ





注※

これらの作業を開発環境で一括して実行することもできます。ただし，一括実行は，システム開発時，または単体テストから結合テスト時を対象としています。詳細については，「8.5　HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行」を参照してください。

このマニュアルでは，図中の「開発環境」で実施する作業の手順について説明しています。運用環境および実行環境で実施する作業については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」を参照してください。

図中の各作業の概要を次に説明します。

(1)　サービス部品の作成

サービスプラットフォームで利用するサービス部品を作成します。サービス部品にはインターフェース情報が定義されている必要があります。利用できるサービス部品の種類などについては，「2.6　利用できるサービス部品の種類と適用範囲」を参照してください。

(2) Service Architect のインストール

開発環境として利用するマシンに Service Architect をインストールします。

インストールの詳細については，「2.1.2　インストール」を参照してください。

(3) 開発環境の構築

開発環境の構築では，Eclipse のアーカイブファイルを入手し，セットアップします。

詳細については，「2.1　開発環境を使用するためのセットアップ」の手順 2 および手順 3 を参照してください。

(4) SOAP モードの設定

システムで使用する SOAP モードを設定します。SOAP モードには，SOAP1.1 モードおよび SOAP1.1/1.2 併用モードがあります。なお，SOAP1.1 モードは J2EE サーバの互換モードが推奨モードの場合は使用できません。

SOAP モードの概要および設定方法については，「2.2.2　使用する SOAP モード」を参照してください。

(5) プロジェクトの作成

HCSCTE を組み込んだ Eclipse で，HCSCTE のプロジェクトを作成します。HCSC コンポーネントを開発する前に，プロジェクトを作成して，プロパティを設定します。

プロジェクトの作成の詳細については，「3.1.1　プロジェクトを作成する」を，プロパティの設定の詳細については，「3.1.2　プロパティを設定する」を参照してください。

(6) リポジトリのインポート

運用環境で簡易セットアップをして設定したリポジトリを，開発環境にインポートします。

リポジトリのインポート方法の詳細については，「3.2.3　リポジトリをインポートする」を参照してください。

(7) HCSC コンポーネントの作成

事前に作成したサービス部品を呼び出すサービスアダプタ，複数のサービス部品を呼び出すビジネスプロセス，サービスリクエスタからの実行要求を受け付けるユーザ定義受付などの HCSC コンポーネントを作成します。

なお，それぞれの HCSC コンポーネントを定義する前に，サービス部品を呼び出す電文の形式である電文フォーマットを作成します。

(a)　電文フォーマットの作成

サービス部品を呼び出す電文の形式（電文フォーマット）を作成します。電文フォーマットは，呼び出すサービス部品の種類および利用する電文の種類で作成方法が異なります。

電文フォーマットの作成については，「4.　電文フォーマットの作成」を参照してください。電文フォーマット（バイナリフォーマット定義ファイル）を作成するときに使用する画面については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.1　バイナリフォーマット定義画面」を参照してください。電文フォーマット（バイナリフォーマット定義ファイル）を作成するときに使用するダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.3　バイナリフォーマット定義に関連するダイアログ」を参照してください。

なお，DB アダプタ（データベースを操作するためのサービスアダプタ）を利用する場合の電文フォーマットの作成には，コマンドを利用します。コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csamkxmls（DB アダプタ用の XML フォーマット定義ファイルの生成)」を参照してください。

(b)　サービスアダプタの定義

サービス部品（データベースの操作を含む）を呼び出すためのサービスアダプタを作成し，詳細を定義します。

サービスアダプタの定義の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.　サービスアダプタの定義」を参照してください。サービスアダプタを定義するときに使用する画面，ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.2　サービスアダプタ定義画面」を参照してください。

(c)　ビジネスプロセスの定義

ビジネスプロセスを作成し，詳細を定義します。

ビジネスプロセスの定義の詳細については，「5.　ビジネスプロセスの定義」を参照してください。

また，ビジネスプロセスを定義するときに使用する画面，ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.3　ビジネスプロセス定義画面」を参照してください。

(d)　データ変換定義の作成

サービス部品を呼び出す要求電文の電文フォーマットと，呼び出すサービス部品側の電文フォーマットが異なる場合には，データ変換定義を作成します。

データ変換定義の作成の詳細については，「7.　データ変換の定義」を参照してください。

また，データ変換定義を作成するときに使用する画面，ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

(e)　ユーザ定義受付の定義

サービス部品実行要求を受け付け，応答を返すインターフェースとして任意の形式を定義する場合，ユーザ定義受付を定義します。また，ユーザ定義受付でリクエストを受け付けることができる HCSC コンポーネントは，ビジネスプロセスだけです。

ユーザ定義受付には次の 2 種類があります。使い分ける必要がない場合はユーザ定義受付と表記します。

ユーザ定義受付（呼出先固定）
　特定のビジネスプロセスの配下に追加するユーザ定義受付です。

ユーザ定義受付（呼出先選択）
　任意のビジネスプロセスを呼び出せるユーザ定義受付です。ユーザ定義受付（呼出先選択）は，HTTP 受付で使用できます。
　ユーザ定義受付（呼出先選択）の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「付録 L　ユーザ定義受付（呼出先選択）の定義」を参照してください。

ユーザ定義受付の定義の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.　ユーザ定義受付の定義」を参照してください。

また，ユーザ定義受付を定義するときに使用する画面，ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.5　ユーザ定義受付定義画面」を参照してください。

(8)　パッケージング

作成した HCSC コンポーネントは，EAR ファイルにパッケージングして，運用環境から実行環境に配備できる状態にします。

HCSC コンポーネントのパッケージングについては，「8.　HCSC コンポーネントのパッケージングと配備定義」を参照してください。

(9)　配備定義

パッケージングした HCSC コンポーネントを，運用環境から実行環境にどのように配備するかを定義（配備定義）し，システム構成定義を更新します。

配備定義する方法については，「8.　HCSC コンポーネントのパッケージングと配備定義」を参照してください。

(10)　リポジトリのエクスポート

HCSC コンポーネントの作成，パッケージング，および配備定義をしたリポジトリを，運用環境にエクスポートします。

リポジトリのエクスポート方法の詳細については，「3.2.2　リポジトリをエクスポートする」を参照してください。

(11) サービスリクエスタの作成

実行環境で業務担当者からのサービス部品の実行要求を受け付け，HCSC コンポーネントに実行要求を送信するためのサービスリクエスタを作成します。

サービスリクエスタの作成の詳細については，「9. サービスリクエスタの作成」を参照してください。

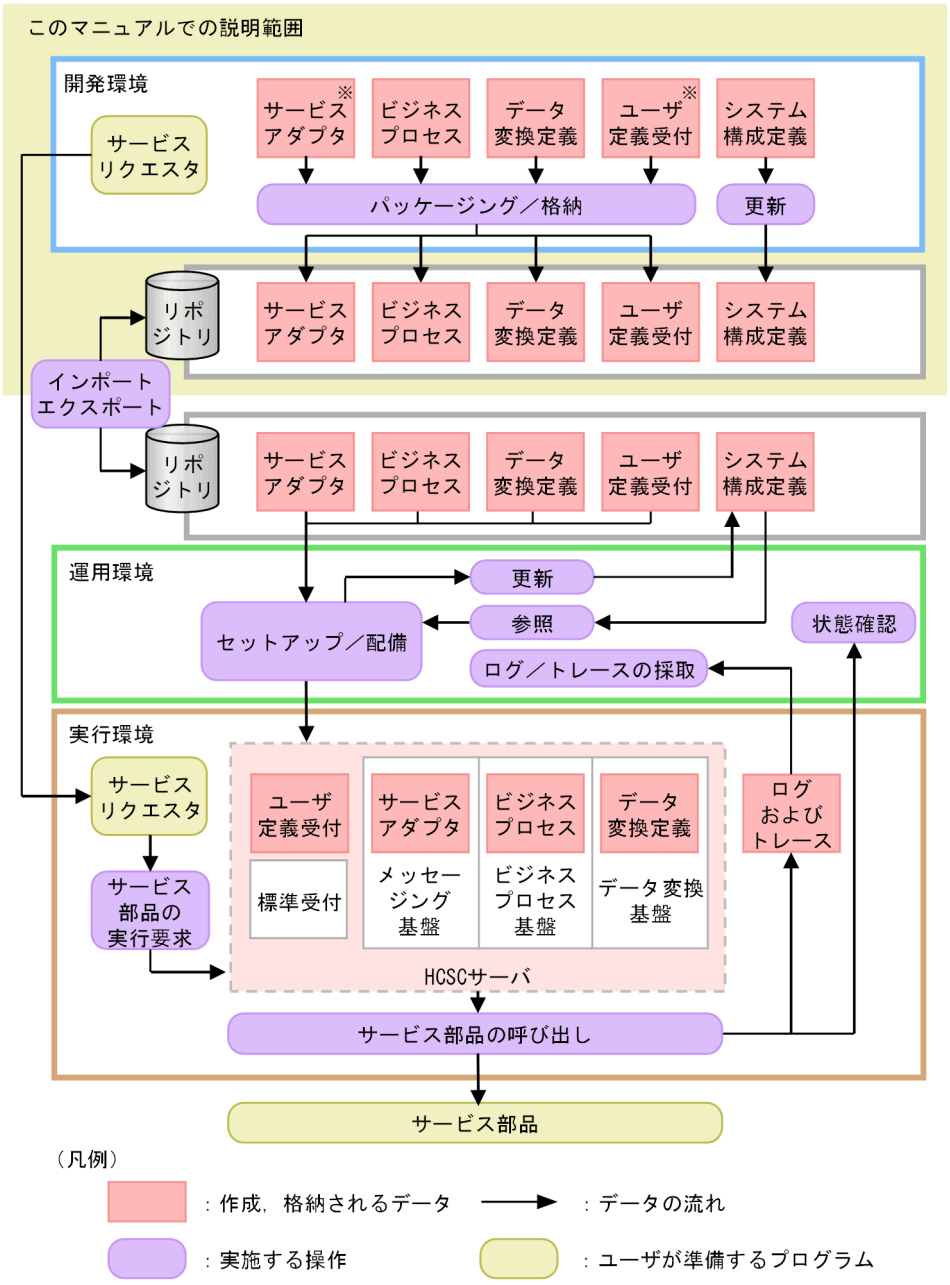
1.2　システム全体と開発環境の関係

サービスプラットフォームでは，開発環境，運用環境および実行環境が相互に関連してシステム全体を構成します。その中での開発環境の位置づけについて説明します。

1.2.1　運用上の位置づけ

システム全体の運用の中で，開発環境がどのように位置づけられるかを，次の図に示します。

図 1-2　運用上の位置づけ



注※

サービスアダプタの定義およびユーザ定義受付の定義については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2. ユーザ定義受付の定義」および「3. サービスアダプタの定義」を参照してください。

開発環境では，作成した HCSC コンポーネント（サービスアダプタ，ビジネスプロセス，およびユーザ定義受付）や配備定義をリポジトリに格納して，運用環境にデータを受け渡します。開発環境で作成し，リポジトリに格納されたデータは，運用環境から実行環境に配備されます。

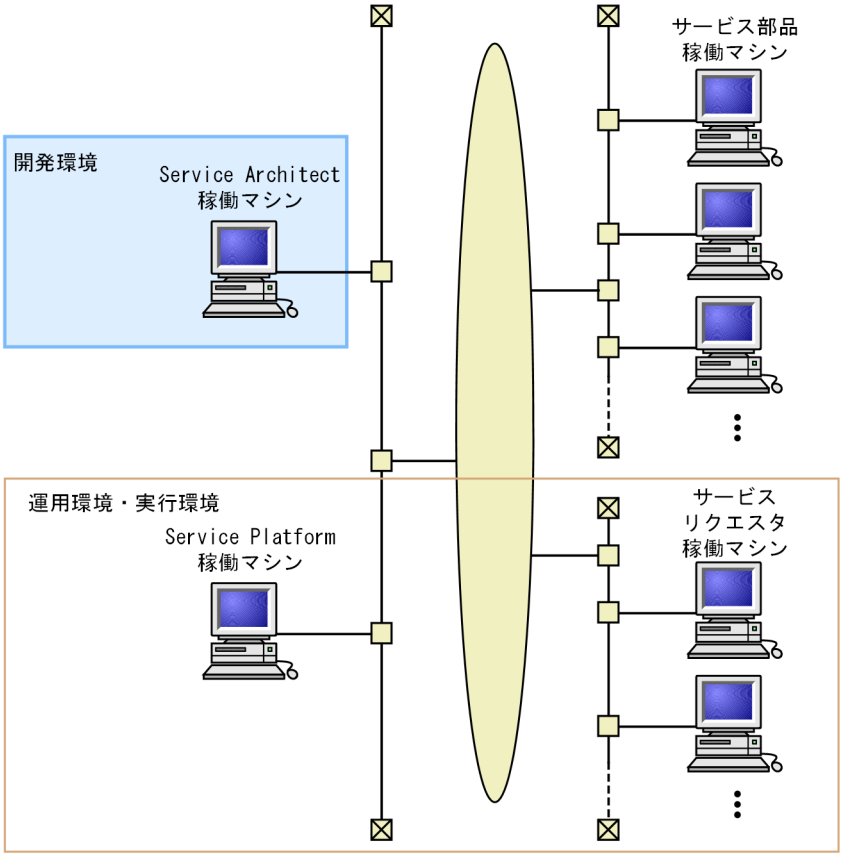
なお，配備定義の作成には，あらかじめ運用環境でセットアップした HCSC サーバの情報などが必要になります。この情報はリポジトリを利用して運用環境から取得する必要があります。リポジトリを利用した運用環境との情報の受け渡しについては，「3.2 リポジトリの管理」を参照してください。

1.2.2 ネットワーク上の位置づけ

開発環境，運用環境および実行環境は，インターネットやイントラネットなどのネットワークを使って連結してシステムを構成します。

システム全体での開発環境のネットワーク上の位置づけについて次の図に示します。

図 1-3 ネットワーク上の位置づけ



注 開発環境は別のネットワーク上に構築することもできます。

2 システムを開発する前に

この章では，サービスプラットフォームのシステムを開発する前に実施する作業，および知っておく必要のある内容について説明します。

2.1　開発環境を使用するためのセットアップ

サービスプラットフォームの開発環境を使用するためのセットアップ手順を次に示します。

1. Service Architect をインストールします。

   詳細は，「2.1.2　インストール」を参照してください。

2. Eclipse のアーカイブファイルを入手し，セットアップします。

   詳細は，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.4　Eclipse セットアップ機能を使用したセットアップ」およびマニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.5　Eclipse の設定」を参照してください。

   なお，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.4　Eclipse セットアップ機能を使用したセットアップ」に示す Eclipse のセットアップを実施後，Windows の［スタート］メニューから［Cosminexus］－［Service_Architect 用 Eclipse セットアップ］を選択して，Service Architect 用 Eclipse セットアップを実行してください。これによって，JDK バージョンに応じた Service Architect 用の-vmargs オプションが，eclipse.ini に追記されます。

   Service Architect 用 Eclipse セットアップの詳細については，マニュアル「サービスプラットフォーム ファーストステップガイド」の「3.3.3　Service Architect 用 Eclipse セットアップ機能を使用した開発環境の構築」を参照してください。

   追記される内容を次に示します。

   ■JDK 11 をインストールした環境の場合

   ```
   -Xbootclasspath/a:<Service Architectのインストールディレクトリ>¥jaxp¥lib¥csmjaxp.jar;<Service Architectのインストールディレクトリ>¥jaxp¥lib¥csmstax.jar
   ```

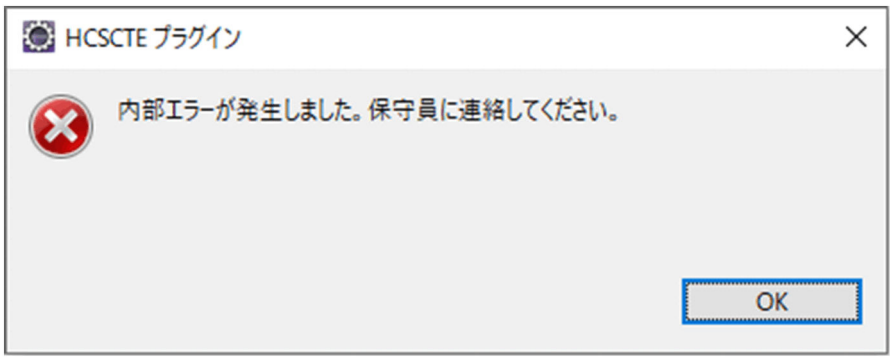
   ■JDK 17 をインストールした環境の場合

   ```
   "--add-opens java.base/java.lang=ALL-UNNAMED"
   ```

   参考

   Eclipse の起動時に，次のエラーダイアログが表示される場合の対処について説明します。

   図 2-1　Eclipse 起動時のエラーダイアログ

Developer's Kit for Java だけをアンインストールしたあとで Service Architect を更新インストールしていた場合，Service Architect をアンインストールして再インストールしてください。

3. 必要に応じてテスト環境を構築します。

HCSC 簡易セットアップ機能を利用する場合

詳細は，「2.4　テスト環境の簡易セットアップ」を参照してください。

HCSC 簡易セットアップ機能を利用しない場合

マニュアル「サービスプラットフォーム システム構築・運用ガイド」を参照して，運用・実行環境を構築してください。

また，テスト環境で組み込みデータベースを利用する場合，「2.1.4　組み込みデータベースの構築」を参照してください。

4. 環境設定をします。

詳細は，「2.1.3　環境設定」を参照してください。

注意事項

- テスト環境では，HCSC サーバ（J2EE サーバ）に，テストで使用するアプリケーション（サービスリクエスタおよびサービス部品）をデプロイできます。

  ただし，次に示す名称は予約されているため，デプロイするアプリケーションの名称として利用できません。

  ・「CSC」で始まる名称

  ・ビジネスプロセス，サービスアダプタのサービス ID

  ・ユーザ定義受付の受付 ID
- テスト環境でビジネスプロセスをデバッグする場合は，必ず HCSC 簡易セットアップ機能を利用してください。
- アプリケーションサーバおよびサービスプラットフォームが提供する Eclipse のプラグイン機能以外の，Eclipse に関するサポートはしていません。Eclipse の使用方法や Eclipse が表示するエラーの対処方法については，ユーザで調査，対処してください。

2.1.1　前提条件

サービスプラットフォームの開発環境を使用するための前提条件を次に示します。

開発環境を利用するユーザ

開発環境を利用するユーザは，「Administrators」または「Power Users」グループに属している必要があります。

インストールおよびセットアップが必要なソフトウェア

開発環境を使用する前に，次のソフトウェアがインストールおよびセットアップされている必要があります。

- Eclipse
- WSDL4J 1.5.1※

  注※　SOAP1.1 モードを使用する場合に必要になります。SOAP1.1 モードについては，「2.2.2 使用する SOAP モード」を参照してください。

2.1.2　インストール

Service Architect は，インストーラからインストールします。インストーラの指示に従ってインストールしてください。なお，インストール作業は Administrator 権限が設定されたユーザが実施してください。詳細は，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「2.2.2　Application Server を新規インストールする（Windows の場合)」を参照してください。なお，インストール手順の製品名「Application Server」は，「Service Platform」と読み替えてください。

インストール先を指定する場合，半角英数字で 50 文字までのパス名を指定してください。

インストール種別を選択する画面では「標準」を選択してください。

Service Architect のインストール終了後，再起動してください。

注意事項

Service Architect では，インストール時に JDK 21 を選択できません。JDK 11（形名：P-CC293Z-2194）または JDK 17（形名：P-CC9W3Z-21B1）をインストールしてください。

ポイント

以前のバージョンからバージョンアップする場合，バージョンアップ前の環境で使用していたリポジトリ情報を，バージョンアップ後も使用できる形式に移行できます。

移行の方法については「付録 A　旧バージョンからの移行」を参照してください。

2.1.3　環境設定

Eclipse を組み込んだ HCSCTE の環境を設定します。環境設定では，次の作業を実施します。

- SOAP モードの設定
- HCSCTE プロジェクトの作成

- プロパティの設定（リポジトリの設定および検証設定）

環境設定の詳細については，「2.2.4　SOAP モードの選択」および「3.　プロジェクトの管理とリポジトリの管理」を参照してください。

参考

電文フォーマットを表示する場合で，電文フォーマット定義ファイルにマルチバイト文字が含まれているときは，次の操作を行って，エンコードを「UTF-8」に変更してください。

1. ［ウィンドウ］－［設定］－［一般］－［ワークスペース］－［テキスト・ファイル・エンコード］を選択します。
2. 「その他（UTF-8)」を設定します。

2.1.4　組み込みデータベースの構築

Service Architect には，開発した HCSC コンポーネントをテストするためのテスト環境として利用できる，組み込みデータベースとして HiRDB/Single Server が用意されています。

開発環境でのテスト用に HCSC サーバを構築する場合，組み込みデータベースを利用すると，別途 DB サーバを用意・構築する必要がありません。

注意事項

- HCSC 簡易セットアップ機能を利用してテスト環境をセットアップする場合，以降で説明する組み込みデータベースの設定を行う必要はありません。

  HCSC 簡易セットアップ機能の詳細については，「2.4　テスト環境の簡易セットアップ」を参照してください。
- テスト用の HCSC サーバと組み込みデータベースは，開発環境の同じマシン上に構築してください。ほかのマシンに DB サーバを構築して組み込みデータベースを利用することはできません。

  また，組み込みデータベースは HiRDB を基にしています。ほかのデータベース（Oracle）を使用する場合，ほかのマシンに DB サーバを構築してください。
- HCSC 簡易セットアップ機能で組み込みデータベースをセットアップした場合，外部マシンから組み込みデータベースにリモート接続できません。HiRDB SQL Executer を使用して組み込みデータベースを操作する場合は，テスト環境マシンで HiRDB SQL Executer を起動し，［CONNECT］ダイアログのホスト名に「localhost」を指定してください。

組み込みデータベースを利用する場合に必要な設定について，以降で説明します。

(1) DB サーバ側の設定

組み込みデータベースを利用するには，次の作業が必要です。

- HiRDB/Single Server のインストール
- HiRDB SQL Executer のインストールおよび環境変数の設定
- DB サーバの構築および環境設定
- ユーザの設定およびスキーマの定義
- RD エリアの準備
- 組み込みデータベースの再起動

各作業について，次に説明します。

(a) HiRDB/Single Server のインストール

Service Architect に含まれる構成ソフトウェアの HiRDB/Single Server をインストールします。

Service Architect のインストールの詳細については，「2.1.2 インストール」を参照してください。

(b) HiRDB SQL Executer のインストールおよび環境変数の設定

HiRDB SQL Executer をインストールします。

組み込みデータベースを構築する場合の HiRDB SQL Executer のインストール方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「3.2 HiRDB SQL Executer のインストール」を参照してください。

インストールが完了したら，次に示す環境変数 PATH を追加・設定します。

```
<サービスプラットフォームのインストールディレクトリ>¥DB¥BIN;
<サービスプラットフォームのインストールディレクトリ>¥DB¥CLIENT¥UTL;
```

(c) DB サーバの構築および環境設定

組み込みデータベースの DB サーバの構築・環境設定には，Developer が提供するコマンドと HiRDB SQL Executer を利用します。

DB サーバの構築および環境設定の方法については，HiRDB のマニュアルを参照してください。

(d) ユーザの設定およびスキーマの定義

データベースを利用するユーザとスキーマを定義します。

組み込みデータベースのユーザの設定およびスキーマの定義の方法については，HiRDB のマニュアルを参照してください。

(e) RD エリアの準備

HCSC Messaging の管理情報テーブルを格納するために，必要に応じて RD エリアを作成，追加または拡張します。また，RD エリアを作成，追加または拡張した場合に，正しく処理されたかどうか確認します。

RD エリアを作成，追加または拡張する手順について，次に示します。

1. OS のシステム環境変数（PDHOST，PDNAMEPORT，PDUSER）に，組み込みデータベースとは異なる HiRDB サーバの接続情報を設定している場合，set コマンドで削除します。
2. Windows の［スタート］メニューの［プログラム］から，［HiRDBSingleServer_CS0］ − ［HiRDB コマンドプロンプト］を選択します。
   組み込みデータベースを操作する環境変数が設定されます。
3. 作業に応じて，次のコマンドを実行します。
   - pdfmkfs コマンド（RD エリアを作成する場合）
   - pdmod コマンド（RD エリアを追加，拡張する場合）
   - pddbls コマンド（RD エリアの状態を表示する場合）

   上記のコマンドについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。
4. RD エリアを追加・拡張した場合，必要に応じて組み込みデータベースの DB 定義ファイル（HiRDB システム定義ファイル）の pdbuffer オペランドを追加して，組み込みデータベースのグローバルバッファを拡張します。
   組み込みデータベースの DB 定義ファイルは，DB サーバの構築時に指定した DB 定義ファイルディレクトリの配下にあります。

(f) 組み込みデータベースの再起動

組み込みデータベースを停止して，再度起動します。

組み込みデータベースの停止および起動の方法については，HiRDB のマニュアルを参照してください。

(2) DB クライアント（HCSC サーバ）側の設定

組み込みデータベースを利用する場合，DB クライアント（HCSC サーバ）の設定として，次の作業が必要です。

- DB クライアントの環境変数の設定
- 環境変数グループの設定
- 組み込みデータベースの同時接続数の設定
- Reliable Messaging 用のユーザ権限の設定および RD エリアの準備

各作業について，次に説明します。

(a)　DB クライアントの環境変数の設定

環境変数 PDXAMODE および PDTXACANUM を設定します。

(b)　環境変数グループの設定

次に示すパスの「クライアント環境変数登録ツール」で環境変数のグループを登録します。

```
<サービスプラットフォームのインストールディレクトリ>¥DB¥CLIENT¥UTL
```

クライアント環境変数登録ツールで環境変数グループを登録する方法の詳細については，マニュアル「HiRDB UAP 開発ガイド」を参照してください。

なお，登録した環境変数グループ名は，DB Connector のプロパティの XA_OPEN 文字列で指定します。DB Connector のプロパティの XA_OPEN 文字列（XAOpenString）については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」の「4.2.2　DB Connector のプロパティ定義」を参照してください。

(c)　組み込みデータベースの同時接続数の設定

必要に応じて，組み込みデータベースの同時接続数を増やします。組み込みデータベースの DB 定義ファイルのオペランドを変更します。DB 定義ファイルは，「(1)(c)　DB サーバの構築および環境設定」の作業を行うときに，DB サーバ構築の設定画面で設定するディレクトリにあります。

DB 定義ファイルで変更するオペランドを次に示します。

- pd_max_users（最大同時接続数）
- pd_max_server_process（最大同時起動サーバプロセス数）
- pd_process_count（常駐プロセス数）

あわせて，環境変数 PDTXACANUM の値は，pd_max_users の指定値以上に設定します。

(d)　Reliable Messaging 用のユーザ権限の設定および RD エリアの準備

必要に応じて，Reliable Messaging 用のユーザ権限の設定および RD エリアの準備を行います。

ユーザ権限の設定方法は，「(1)(d)　ユーザの設定およびスキーマの定義」に示す方法と同じです。

RD エリアの準備の方法は「(1)(e)　RD エリアの準備」に示す方法と同じです。

2.1.5　アンインストール

Service Architect をアンインストールする方法について説明します。詳細は，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「3.3　システム環境のアンセットアップと Application

Server のアンインストール」を参照してください。なお，アンインストール手順の製品名「Application Server」は，「Service Platform」と読み替えてください。

アンインストール作業は Administrator 権限が設定されたユーザが実施してください。

アンインストール前に Eclipse 環境のアンセットアップを実行してください。手順は，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.8.2　Eclipse 環境のアンセットアップ」を参照してください。

また，J2EE サーバ，Management Server，PRF など実行環境の構成要素を停止してからアンインストールしてください。組み込みデータベースを使用している場合は，組み込みデータベースを停止してからアンインストールしてください。

アンインストールの手順について，次に示します。

1. 次のどちらかの操作を実施します。

   - Windows の［スタート］メニューから，［Cosminexus］－［uCosminexus Service Architect アンインストール］を選択する
   - ［コントロールパネル］の［プログラムの追加と削除］から，［uCosminexus Service Architect］を選択する

   Service Architect のアンインストールを確認するダイアログが表示されます。

2. ［はい］ボタン，または［いいえ］ボタンをクリックします。

   ［はい］ボタンをクリックした場合

   アンインストールが開始され，Service Architect の構成ソフトウェアがすべて削除されます。

   ［いいえ］ボタンをクリックした場合

   アンインストールする構成ソフトウェアを選択するダイアログが表示されます。アンインストールする構成ソフトウェアを選択して，［次へ］ボタンをクリックすると，アンインストールが開始され，選択した構成ソフトウェアが削除されます。

2.2　構成の形態，J2EE サーバの互換モード，および SOAP モードの選択

サービスプラットフォームのシステムを開発する前に，システムで利用する機能に応じて，J2EE サーバの互換モード，SOAP モードなどを明確にする必要があります。

この節では次の機能について説明します。各項で詳細を確認し，システムで利用する機能を選択してください。

- データベース／Reliable Messaging の使用有無
- 使用する SOAP モード
- J2EE サーバの互換モード

2.2.1　データベースと Reliable Messaging の使用有無

サービスプラットフォームのどの機能を使用し，どのような運用をしたいかによって，データベースと Reliable Messaging の使用有無が異なります。使用する機能ごとに，データベースと Reliable Messaging の使用有無を次の表に示します。

表 2-1　使用したい機能ごとのデータベースと Reliable Messaging の使用有無

使用したい機能	データベース	Reliable Messaging
次の同期受付を使用したい • 標準受付（Web サービス） • 標準受付（SessionBean） • SOAP 受付 • TP1/RPC 受付 • FTP 受付 • HTTP 受付 • Message Queue 受付 • ファイルイベント受付 • Kafka 受付 • スケジュール駆動受付 • gRPC 受付	×	×
次の非同期受付を使用したい • 標準受付（MDB（WS-R）） • 標準受付（MDB（DB キュー）） • Message Queue 受付	×	×
次の同期のサービスアダプタを使用したい • SOAP アダプタ • SessionBean アダプタ • TP1 アダプタ	×	×

使用したい機能		データベース	Reliable Messaging
• ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ • gRPC アダプタ		×	×
次の非同期のサービスアダプタを使用したい • MDB（WS-R）アダプタ • MDB（DB キュー）アダプタ		○	○
DB アダプタを使用したい		×※1	×
ビジネスプロセスを使用したい	永続化するビジネスプロセスを使用したい	○	×
	永続化しないビジネスプロセスを使用したい※2	×	×
プロセスインスタンスの実行履歴を管理したい		○	×

（凡例）

○：必須です。

×：必須ではありません。

注※1

DB アダプタを使用する場合，HCSC サーバが稼働しているマシン上にデータベースは不要です。しかし，DB アダプタから接続するサービス部品稼働マシンでは，データベースが必要になります。

注※2

永続化しないビジネスプロセスを使用する場合，開発環境で定義できるプロセスに制限がありますので注意してください。詳細は「5.2.1(3)　ステータスの永続化の設定」を参照してください。

この表から，データベースと Reliable Messaging の使用有無による構成形態は，次の 3 つのパターンがあることがわかります。

- データベースと Reliable Messaging の両方を使用する
- データベースと Reliable Messaging の両方を使用しない
- データベースを使用して Reliable Messaging を使用しない

本番環境をどのパターンで構築するかによって，テスト環境の設定内容が異なります。テスト環境と本番環境で設定する，データベースと Reliable Messaging の使用有無については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「1.3　テスト環境と本番環境との関係」を参照してください。また，テスト環境での設定内容については，「2.4　テスト環境の簡易セットアップ」を参照してください。

2.2.2　使用する SOAP モード

サービスプラットフォームでシステムを開発する前に，使用する SOAP モードを選択します。

ここでは，SOAP モードの種類と各モードのサポート範囲について説明します。

注意事項

Service Architect 09-71 から SOAP モードのデフォルトが SOAP1.1/1.2 併用モードに変更になりました。

(1)　SOAP モードの種類

SOAP モードの種類と，対応する Web サービス標準仕様および実行環境を示します。

- SOAP1.1 モード

  SOAP1.1 に対応したシステムを開発する場合に選択します。SOAP1.1 モードは WS-I Basic Profile1.0a に対応しています。

  SOAP1.1 モードでは，SOAP メッセージの送受信に SOAP 通信基盤を使用します。なお，SOAP1.1 モードは，J2EE サーバの互換モードが推奨モードの場合は使用できません。

  また，SOAP1.1 モードは，JDK 11 をインストールした環境の場合だけ選択できます。

- SOAP1.1/1.2 併用モード

  SOAP1.1 または SOAP1.2 に対応したシステムを開発する場合に選択します。SOAP1.1/1.2 併用モードは WS-I Basic Profile1.1 に対応しています。

  SOAP1.1/1.2 併用モードでは WS-I Attachments Profile -Version 1.0 で規定された wsi:swaRef 形式の添付ファイルに対応しています。

  SOAP1.1/1.2 併用モードでは，SOAP メッセージの送受信に JAX-WS エンジンを使用します。

  また，SOAP1.1/1.2 併用モードは，JDK 11 をインストールした環境および JDK 17 をインストールした環境の場合に選択できます。

(2)　SOAP モードのサポート範囲

受付，サービスアダプタ，およびビジネスプロセスの機能と SOAP モードの対応を次の表に示します。

表 2-2　SOAP モードのサポート範囲

分類	機能名	SOAP モード	
		1.1	1.1/1.2 併用
受付	標準受付（Web サービス（SOAP1.1））	○	○
	標準受付（Web サービス（SOAP1.2））	×	○
	標準受付（SessionBean）	○	○

分類	機能名	SOAP モード	
		1.1	1.1/1.2 併用
受付	標準受付（MDB（WS-R)）	○	×
	標準受付（MDB（DB キュー)）	○	○
	ユーザ定義受付	○	○
サービスアダプタ	SOAP アダプタ	○	○
	SessionBean アダプタ	○	×
	MDB（WS-R）アダプタ	○	×
	MDB（DB キュー）アダプタ	○	○
	DB アダプタ	○	○
	TP1 アダプタ	○	○
	ファイルアダプタ	○	○
	Object Access アダプタ	○	○
	Message Queue アダプタ	○	○
	FTP アダプタ	○	○
	ファイル操作アダプタ	○	○
	メールアダプタ	○	○
	HTTP アダプタ	×	○
	コマンドアダプタ	○	○
	SFTP アダプタ	○	○
	Kafka アダプタ	○	○
	gRPC アダプタ	○	○
	汎用カスタムアダプタ	○	○
ビジネスプロセス	ビジネスプロセス	○	○

（凡例）

○：サポートしています。

×：サポートしていません。

WSDL 定義のスタイルと SOAP モードの対応を次の表に示します。

表 2-3　SOAP モードのサポート範囲（WSDL 定義のスタイル）

SOAP のバージョン	WSDL 定義のスタイル	SOAP モード	
		1.1	1.1/1.2 併用
SOAP1.1	rpc/literal	○	×
	document/literal	○	○
SOAP1.2	rpc/literal	×	×
	document/literal	×	○

（凡例）

○：サポートしています。

×：サポートしていません。

なお，SOAP モードと Web サービスを利用したサービス部品との関係については，「2.6.1　Web サービスを利用したサービス部品の適用範囲」を参照してください。

2.2.3　J2EE サーバの互換モードの選択

J2EE サーバの互換モードには次の 2 種類があります。サービスプラットフォームでシステムを開発する前に，どちらで運用するか決定してください。

(1)　J2EE サーバの互換モードの種類

(a)　V9 互換モード

Java EE 7 以降の新機能に対応しない J2EE サーバの構成です。V9 以前からの互換性を重視する場合は，こちらの V9 互換モードを選択してください。

なお，V9 互換モードを選択した場合は，HTTP アダプタのリダイレクト機能は使用できません。

また，V9 互換モードは JDK 11 をインストールした環境の場合だけ選択できます。

(b)　推奨モード

Java EE 7 以降の新機能に対応する J2EE サーバの構成です。新規にシステムを構築する場合は，こちらの推奨モードを選択してください。

なお，推奨モードを選択した場合は，SOAP モードのうち SOAP1.1 モードは使用できません。

また，推奨モードは JDK 11 をインストールした環境および JDK 17 をインストールした環境の場合に選択できます。

(2)　サポート機能の差異

サービスプラットフォームで使用できる機能は，J2EE サーバの互換モードによって一部異なります。使用可否が異なる機能を次の表に示します。

表 2-4　J2EE サーバの互換モードごとの機能の使用可否

分類	機能名	J2EE サーバの互換モード	
		V9 互換モード	推奨モード
サービスアダプタ	DB アダプタのサポート DBMS • Amazon Aurora MySQL • Amazon Aurora PostgreSQL	×	○
	HTTP アダプタのリダイレクト機能	×	○
	HTTP アダプタの通信ログ	○	×

（凡例）

○：サポートしています。

×：サポートしていません。

2.2.4　SOAP モードの選択

SOAP モードは，HCSC 簡易セットアップ画面または cscsoapmode コマンドで設定できます。

HCSC 簡易セットアップ画面では，開発環境と実行環境の SOAP モードを設定できます。HCSC 簡易セットアップ画面での設定については，「2.4.2(1)(c)　HCSC 簡易セットアップ画面の入力項目」を参照してください。

cscsoapmode コマンドでは，開発環境の SOAP モードを設定できます。実行環境の SOAP モードを設定する方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.2.3 SOAP モードの選択」を参照してください。cscsoapmode コマンドについては，マニュアル「サービスプラットフォーム リファレンス」の「cscsoapmode（SOAP モードの設定)」を参照してください。

設定した SOAP モードを確認する方法については，「2.4.5　SOAP モードの確認」を参照してください。

設定した SOAP モードを変更する方法については，「3.1.5　SOAP モードを変更する」を参照してください。

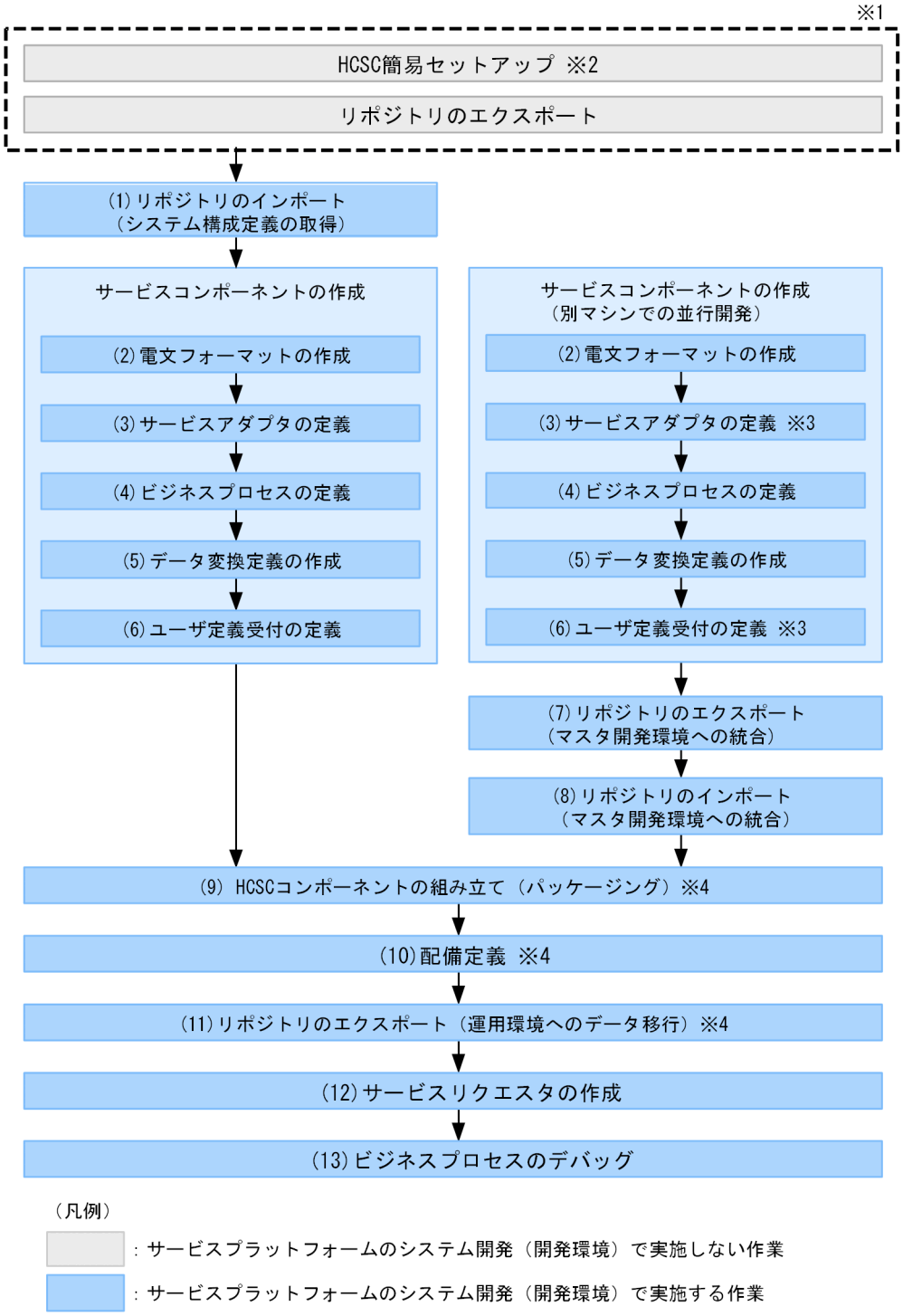
2.3　サービスプラットフォームを利用したシステムの開発

ここでは，サービスプラットフォームを利用してシステムを開発する手順について説明します。

2.3.1　開発手順

サービスプラットフォームを利用してシステムを開発する手順を次に示します。

図 2-2　システムの開発手順



注※1

HCSC 簡易セットアップについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「1.3　テスト環境と本番環境との関係」を参照してください。リポジトリのエクスポートについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「4.2　リポジトリのエクスポート」を参照してください。

注※2

HCSC 簡易セットアップ機能を使用してテスト環境を構築します。HCSC 簡易セットアップ機能については，「2.4　テスト環境の簡易セットアップ」を参照してください。HCSC 簡易セットアップ機能を使用しない場合は，マニュアル「サービスプラットフォーム システム構築・運用ガイド」を参照して，運用・実行環境を構築してください。

注※3

サービスアダプタの定義およびユーザ定義受付の定義については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.　ユーザ定義受付の定義」および「3.　サービスアダプタの定義」を参照してください。

注※4

これらの作業を一括して実行することもできます。ただし，一括実行は，システム開発時，または単体テストから結合テスト時を対象としています。なお，運用環境で実施する HCSC コンポーネントの配備と開始も一緒に実行されます。詳細については，「8.5　HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行」を参照してください。

各工程の作業を次に示します。

(1)　リポジトリのインポート（システム構成定義の取得）

運用環境で作成した HCSC サーバの情報をマスタ開発環境に統合するため，システム構成定義だけをインポートします。リポジトリのインポートについては，「3.2.3　リポジトリをインポートする」を参照してください。

(2)　電文フォーマットの作成

サービスリクエスタ，サービスアダプタ，およびサービス部品の間でやり取りされる電文の形式（電文フォーマット）を定義します。作成する電文フォーマットは，サービス部品の実行に利用する電文で XML 形式のデータを扱うか，バイナリ形式（XML 形式以外）のデータを扱うかによって異なります。電文フォーマットの作成の詳細については，「4.　電文フォーマットの作成」を参照してください。

(3)　サービスアダプタの定義

サービスアダプタを使用してシステムを開発する場合，サービスアダプタ追加用のウィザードおよび定義済みのサービスアダプタを利用して，サービスアダプタを追加します。追加したサービスアダプタは，サービスアダプタ定義画面で定義します。サービスアダプタの定義の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.　サービスアダプタの定義」を参照してください。

(4)　ビジネスプロセスの定義

ビジネスプロセスを使用してシステムを開発する場合，ビジネスプロセス追加用のウィザードおよび定義済みのビジネスプロセスを利用して，ビジネスプロセスを追加します。追加したビジネスプロセスは，ビ

ジネスプロセス定義画面で定義します。ビジネスプロセスの定義の詳細については，「5.　ビジネスプロセスの定義」を参照してください。

また，ビジネスプロセスの実行要求を標準受付以外で受け付ける場合，ユーザ定義受付を定義する必要があります。ユーザ定義受付の定義の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.　ユーザ定義受付の定義」を参照してください。

(5)　データ変換定義の作成

サービス部品との電文のやり取りでデータ変換が必要な場合は，データ変換を定義します。データ変換の定義の詳細については，「7.　データ変換の定義」を参照してください。

(6)　ユーザ定義受付の定義

サービス部品実行要求を受け付け，応答を返すインターフェースとして任意の形式を定義する場合，ユーザ定義受付を定義します。ユーザ定義受付の定義の詳細については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.　ユーザ定義受付の定義」を参照してください。

(7)　リポジトリのエクスポート（マスタ開発環境への統合）

複数の開発環境で HCSC コンポーネントを開発（分散開発）する場合，マスタ開発環境以外の環境で開発した HCSC コンポーネントが含まれるリポジトリをエクスポートします。

リポジトリのエクスポートについては，「3.2.2　リポジトリをエクスポートする」を参照してください。

(8)　リポジトリのインポート（マスタ開発環境への統合）

マスタ開発環境以外のマシンで分散開発した HCSC コンポーネントの情報をマスタ開発環境に統合するため，「(7)　リポジトリのエクスポート（マスタ開発環境への統合）」でエクスポートしたリポジトリの一部をマスタ開発環境にインポートします。リポジトリのインポートについては，「3.2.3　リポジトリをインポートする」を参照してください。

(9)　HCSC コンポーネントの組み立て（パッケージング）

「(3)　サービスアダプタの定義」～「(6)　ユーザ定義受付の定義」で定義した HCSC コンポーネントの定義に関連する定義ファイル，および実行環境が提供するファイルをまとめて EAR ファイルにパッケージングします。パッケージングの方法については，「8.　HCSC コンポーネントのパッケージングと配備定義」を参照してください。

(10)　配備定義

HCSC コンポーネントを配備するクラスタ（または単一の HCSC サーバ）の情報を定義します。配備定義の詳細については，「8.　HCSC コンポーネントのパッケージングと配備定義」を参照してください。

参考

配備する前に，配備するクラスタ（または単一の HCSC サーバ）に設定されている，データベースと Reliable Messaging の使用有無を確認する場合の手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［システム構成定義一覧］を選択します。
   ツリービューのシステム構成定義一覧に，配備するクラスタ（または単一の HCSC サーバ）の一覧が表示されます。
2. 配備するクラスタ（または単一の HCSC サーバ）の一覧から，確認したいクラスタ（または単一の HCSC サーバ）を選択し，ダブルクリックします。
   配備するクラスタ（または単一の HCSC サーバ）に設定されている，データベースと Reliable Messaging の使用有無の情報が表示されます。

(11)　リポジトリのエクスポート（運用環境へのデータ移行）

マスタ開発環境で，HCSC コンポーネントの配備に必要な定義ファイルを格納したリポジトリをエクスポートします。マスタ開発環境からエクスポートした情報は，運用環境にインポートして利用されます。リポジトリのエクスポートについては，「3.2.2　リポジトリをエクスポートする」を参照してください。

(12)　サービスリクエスタの作成

HCSC コンポーネントにサービス部品の実行を要求する電文を送信するためのサービスリクエスタを作成します。なお，サービスリクエスタの作成は，サービスアダプタの定義やビジネスプロセスの定義と同時に行うこともできます。同時に行う場合，サービスアダプタの定義やビジネスプロセスの定義で設定するインターフェース情報などをあらかじめ確認して，サービスリクエスタでも同じ設定にしてください。HCSC コンポーネントの作成方法については，「9.　サービスリクエスタの作成」を参照してください。

(13)　ビジネスプロセスのデバッグ

定義したビジネスプロセスを運用環境で利用する前に，開発環境でテストおよびデバッグします。ビジネスプロセスをデバッグする方法については，「10.　ビジネスプロセスのデバッグ」を参照してください。

2.4　テスト環境の簡易セットアップ

実行環境に配備して実運用を始める前に，Service Architect で開発したサービスアダプタやビジネスプロセスは，テストやデバッグを実施することを推奨します。そのためには，テストやデバッグを実施するための運用・実行環境（テスト環境）を構築する必要があります。

Service Architect には，テスト環境の構築を支援する **HCSC 簡易セットアップ機能**が備わっています。テストやデバッグを実施するための運用・実行環境には，さまざまな情報を設定する必要があるため，構築には相応の時間が掛かります。HCSC 簡易セットアップ機能を利用すると，テスト環境に必要なさまざまな情報を自動的に設定できます。また，HCSC 簡易セットアップ機能には，HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境を自動的にアンセットアップする機能もあります。

注意事項

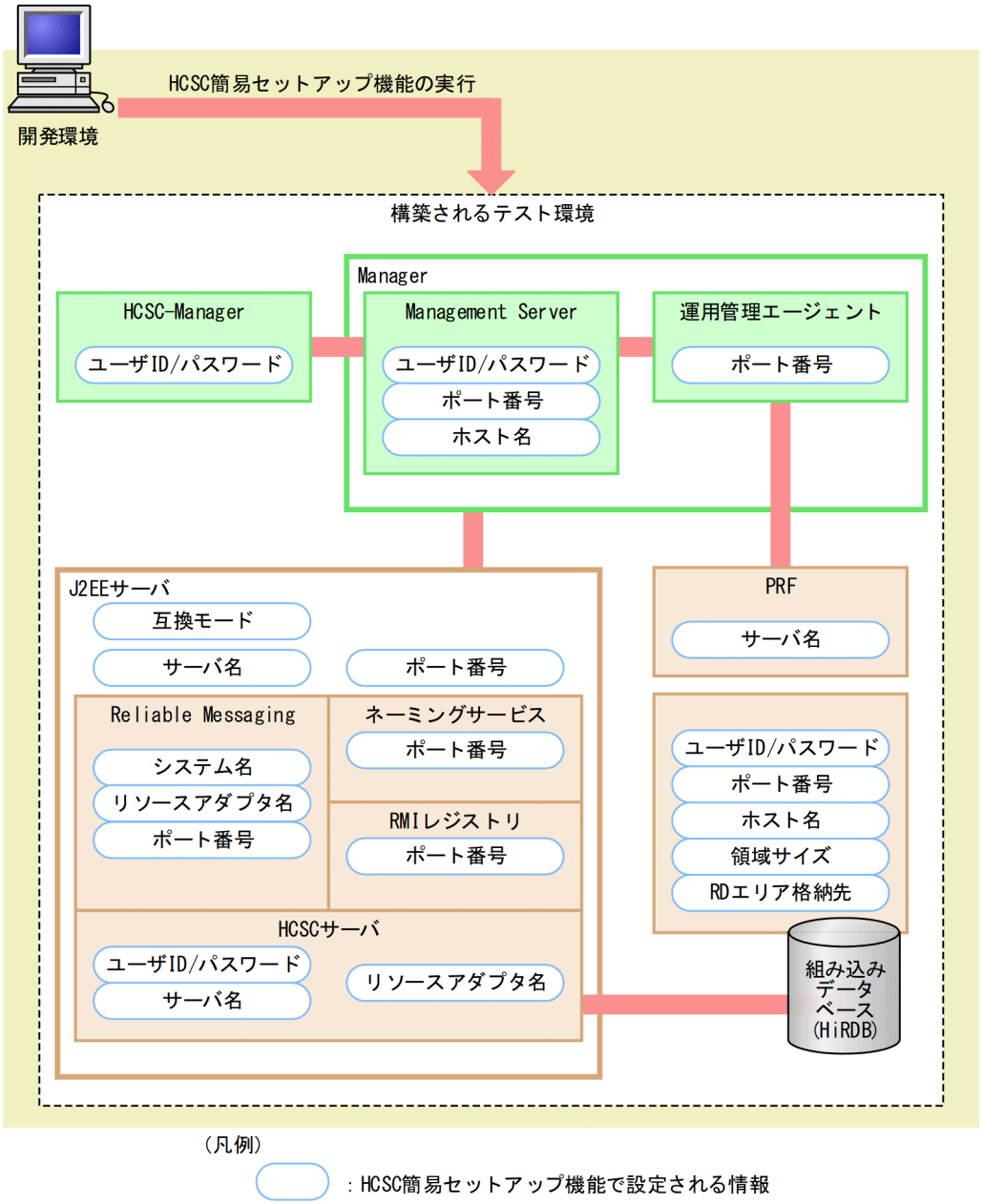
1 台のマシンに複数の運用・実行環境（テスト環境）を構築（CSCMNG_HOME 環境変数を設定）する場合，HCSC 簡易セットアップ機能を使用できません。

この節では，HCSC 簡易セットアップ機能の概要と利用方法について説明します。

2.4.1　HCSC 簡易セットアップ機能で構築できる環境

HCSC 簡易セットアップ機能を利用して構築されるテスト環境は，次の図のような構成になります。また，J2EE サーバの互換モードは，推奨モードまたは V9 互換モードで構築する必要があります。次の図に示すプロセス以外のものをテスト環境として利用する場合，別途設定が必要になります。

図 2-3　HCSC 簡易セットアップ機能を利用して構築されるテスト環境の構成



HCSC 簡易セットアップ機能を利用すると，テスト環境の各プロセスに，ユーザ ID，パスワード，ポート番号，ホスト名などの情報を自動的に設定できます。HCSC 簡易セットアップ機能を利用した場合にテスト環境に設定される情報の詳細については，「2.4.3(2)　テスト環境の運用に必要な情報」を参照してください。

HCSC 簡易セットアップ機能で構築した環境の Web コンテナ単位の同時実行数は 10 です。そのため，11 リクエスト以上を同時に処理する場合，Web コンテナのスレッドが枯渇するおそれがあります。Web コンテナ単位の同時実行数および同時接続数を変更する場合のプロパティを次に示します。

- J2EE サーバの互換モードが推奨モードの場合

simple_model_cjw_v11.xml ファイルの
webserver.connector.nio_http.max_servlet_execute_threads プロパティの定義を変更してください。

webserver.connector.nio_http.max_servlet_execute_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.14.2　実行環境での設定(J2EE サーバの設定)」を参照してください。

リクエスト処理で使用するスレッド数，コネクション数を合わせて制御する場合は，
simple_model_cjw_v11.xml ファイルの次の定義を変更してください。

- webserver.connector.nio_http.max_threads
- webserver.connector.nio_http.min_threads
- webserver.connector.nio_http.max_connections

webserver.connector.nio_http.max_threads プロパティ，および
webserver.connector.nio_http.min_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「7.4.2　実行環境での設定（J2EE サーバの設定)」を参照してください。webserver.connector.nio_http.max_connections プロパティについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「7.3.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

- J2EE サーバの互換モードが V9 互換モードの場合

  simple_model.xml ファイルまたは simple_model_cjl.xml ファイルの
  webserver.connector.inprocess_http.max_execute_threads プロパティの定義を変更してください。

  webserver.connector.inprocess_http.max_execute_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 互換編」の「5.11.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

  リクエスト処理で使用するコネクション数を合わせて制御する場合は，simple_model.xml ファイルまたは simple_model_cjl.xml ファイルの webserver.connector.inprocess_http.max_connections プロパティの定義を変更してください。

  webserver.connector.inprocess_http.max_connections プロパティについては，マニュアル「アプリケーションサーバ 機能解説 互換編」の「6.3.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

2.4.2　HCSC 簡易セットアップ機能の実行

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップ，またはアンセットアップするための前提条件と，テスト環境のセットアップおよびアンセットアップの方法について説明します。

(1)　テスト環境のセットアップ

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップする場合の前提条件，およびセットアップ方法について説明します。

(a) 前提条件

テスト環境をセットアップするための前提条件を次に示します。

テスト環境の前提ソフトウェア

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップするには，次のソフトウェアがインストールされている必要があります。

- Component Container※1
- XML Processor※1
- TPBroker※1
- Performance Tracer※1
- Reliable Messaging※1
- Service Coordinator※1
- WSDL4J 1.5.1※2

注※1

これらのソフトウェアは，Service Architect の構成ソフトウェアです。Service Architect をインストールしている環境であれば，改めて上記のソフトウェアをインストールする必要はありません。

上記のソフトウェアを改めてインストールする場合，インストーラからインストールします。インストーラの指示に従ってインストールしてください。なお，インストール作業は Administrator 権限が設定されたユーザが実施してください。

注※2

WSDL4J 1.5.1 は，これ以外の前提ソフトウェアをインストールしたあとにインストールしてください。WSDL4J 1.5.1 は，SOAP1.1 モードを使用する場合に必要になります。

HCSC 簡易セットアップ機能を利用するときのシステムの状態

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップする場合，システムが次の状態である必要があります。

- サービスプラットフォームを新規にインストールした直後であること。
- サービスプラットフォームに関する設定をしていないこと。

ただし，システム環境変数で，変数名「TZ」の値が有効になっている必要があります。環境変数「TZ」にはタイムゾーンを設定します。タイムゾーンには「JST-9」と指定してください。

なお，セットアップ後にテスト環境をカスタマイズした場合，カスタマイズした情報はアンセットアップ時に削除されます。カスタマイズした情報を削除したくない場合は，カスタマイズ時に追加・変更した定義ファイルのパラメタ情報を記録しておいてください。

データベースと Reliable Messaging の両方を使用する場合の前提条件

データベースと Reliable Messaging の両方を使用する場合（HCSC 簡易セットアップ画面で［DB/RM ありモデル］を選択する場合），次の条件を満たす必要があります。

- HiRDB に関する知識を持っていること。
- 高負荷テストや性能測定などを実施しないこと。

注

テスト環境をアンセットアップしたあとには，HCSC 簡易セットアップ機能を利用してテスト環境を再度セットアップできます。テスト環境のアンセットアップについては「2.4.2(2)　テスト環境のアンセットアップ」を参照してください。

(b)　セットアップ方法

HCSC 簡易セットアップ機能を利用したテスト環境のセットアップには，HCSC 簡易セットアップ画面を利用します。テスト環境のセットアップ方法を次に示します。

1. Windows の［スタート］メニューから，［Cosminexus］－［テスト環境セットアップ］を選択します。
   HCSC 簡易セットアップ画面が表示されます。

2. HCSC 簡易セットアップ画面で必要な情報を入力します。
   HCSC 簡易セットアップ画面で入力する項目については，「2.4.2(1)(c)　HCSC 簡易セットアップ画面の入力項目」を参照してください。

3. ［セットアップ］ボタンをクリックします。
   テスト環境のセットアップが開始されます。セットアップの状況は，HCSC 簡易セットアップ画面のコンソールに表示されます。コンソールに「HCSC 簡易セットアップ機能のセットアップを終了します」と表示されたら，テスト環境は正常にセットアップされています。

注意事項

コンソールにエラーが表示され，テスト環境のセットアップが異常終了した場合，再セットアップする必要があります。異常終了したときに，HCSC 簡易セットアップ画面の［セットアップ］ボタンが活性か，非活性かによって，再セットアップの手順が異なります。

［セットアップ］ボタンが活性の場合

［セットアップ］ボタンをクリックして，再セットアップできます。

［セットアップ］ボタンが非活性の場合

［アンセットアップ］ボタンをクリックして，一度アンセットアップします。そのあとで，再度セットアップしてください。

(c)　HCSC 簡易セットアップ画面の入力項目

HCSC 簡易セットアップ画面は，［メイン］タブと［サーバ名称］タブで分けられます。［サーバ名称］タブでは，サーバ名やクラスタ名などの情報をセットアップ時に変更できます。

HCSC 簡易セットアップ画面の入力項目について，HCSC 簡易セットアップ機能で構築される環境内の各プロセスとの関連を用いて説明します。

HCSC 簡易セットアップ画面を次の図に示します。

図 2-4　HCSC 簡易セットアップ画面（[メイン] タブ）

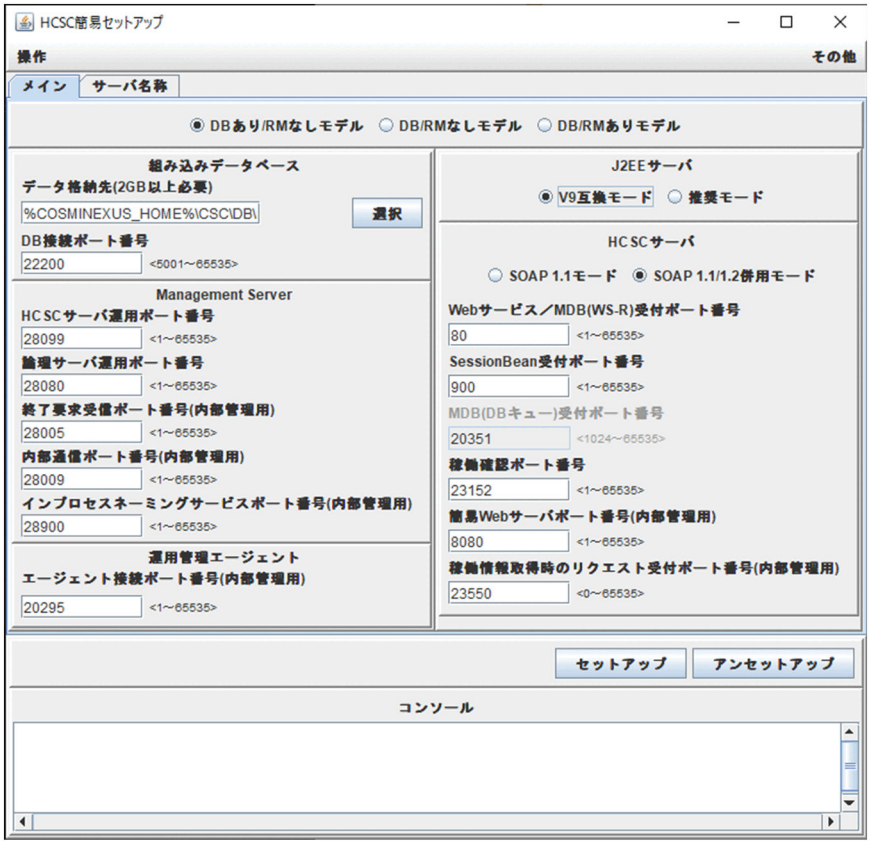
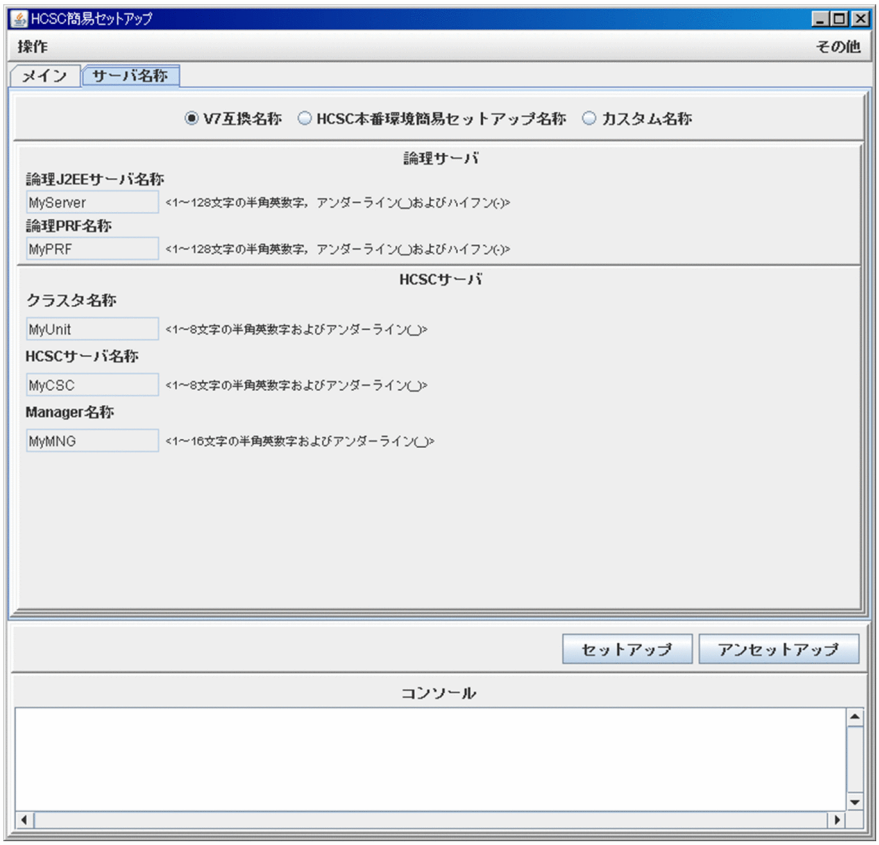
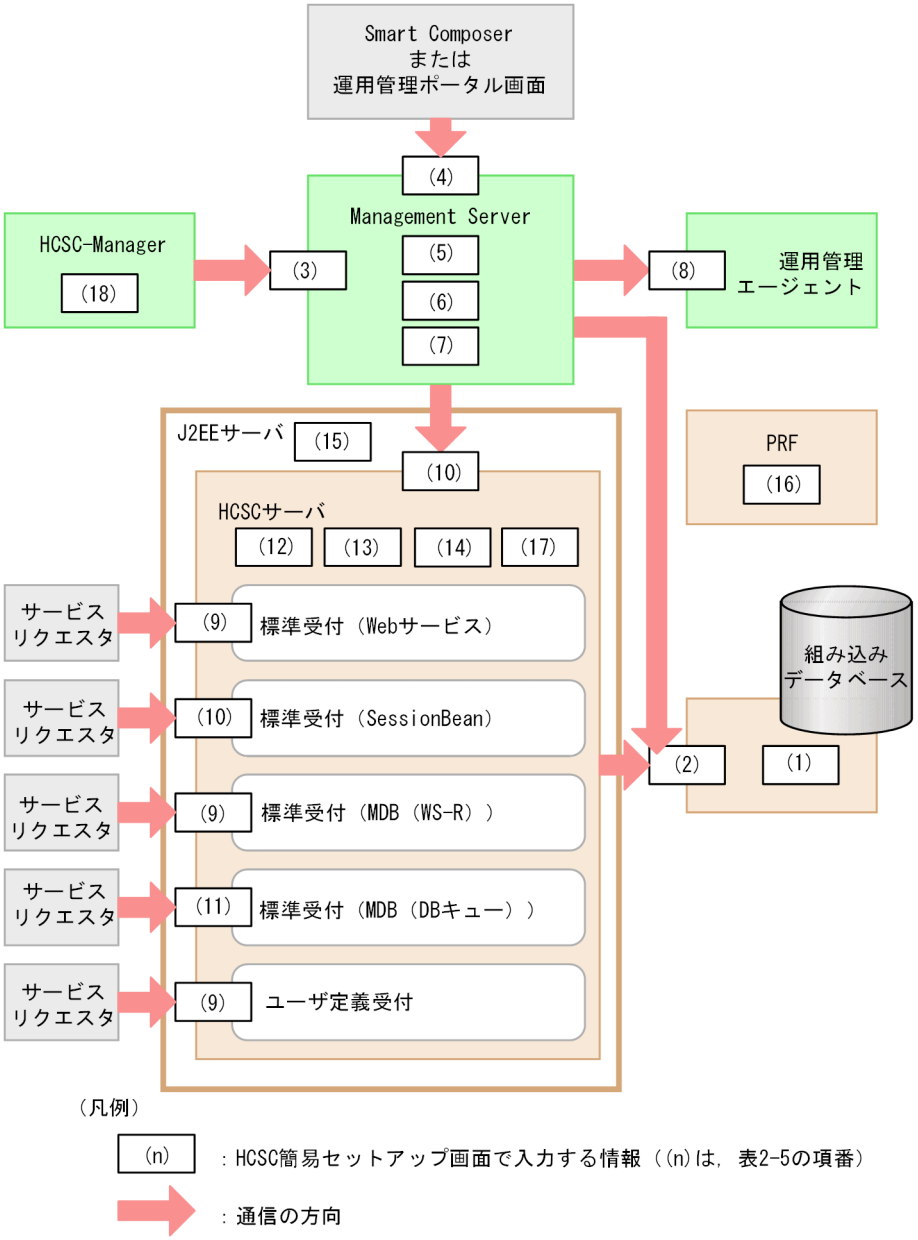


図 2-5　HCSC 簡易セットアップ画面（[サーバ名称] タブ）



HCSC 簡易セットアップ画面で入力する項目は，次の図のように，構築される環境の各プロセスに設定されます。設定される値の関連を次の図に示します。

図 2-6　HCSC 簡易セットアップ画面の入力項目の関連



[DB あり/RM なしモデル]，[DB/RM なしモデル]，または［DB/RM ありモデル］のどれを選択したかによって，設定する項目が異なります。図 2-6 の（n）で示されている設定値（HCSC 簡易セットアップ画面での入力項目）の詳細を次の表に示します。

表 2-5　HCSC 簡易セットアップ画面で入力する項目

入力項目				説明	初期値※2
タブ	分類	項番	HCSC 簡易セットアップ画面の項目名※1		
メイン	モデル	－	DB あり/RM なしモデル	データベースを使用し，Reliable Messaging を使用しない場合に選択します。	選択

入力項目				説明	初期値※2
タブ	分類	項番	HCSC 簡易セットアップ画面の項目名※1		
メイン	モデル	－	DB/RM なしモデル	データベースと Reliable Messaging の両方を使用しない場合に選択します。	－
		－	DB/RM ありモデル	データベースと Reliable Messaging の両方を使用する場合に選択します。	－
	組み込みデータベース	(1)	データ格納先	RD エリアを作成するディレクトリおよびシステムファイルを作成するディレクトリを指定します。2GB 以上使用できるディレクトリを指定します。 存在しないディレクトリを指定した場合，指定したディレクトリが新規作成されます。 ［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ指定します。	<サービスプラットフォームのインストールディレクトリ>¥CSC¥DB¥area
		(2)	DB 接続ポート番号	Management Server または HCSC サーバから組み込みデータベースにアクセスする場合に使用するポート番号を指定します。5001～65535 の整数で指定します。 ［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ指定します。	22200
	Management Server	(3)	HCSC サーバ運用ポート番号	HCSC-Manager から Management Server にアクセスする場合に使用するポート番号を指定します。1～65535 の整数で指定します。	28099
		(4)	論理サーバ運用ポート番号	Smart Composer または運用管理ポータル画面から Management Server にアクセスする場合に使用するポート番号を指定します。1～65535 の整数で指定します。	28080
		(5)	終了要求受信ポート番号（内部管理用）	Management Server が内部の管理用に利用するポート番号を指定します。1～65535 の整数で指定します。	28005
		(6)	内部通信ポート番号（内部管理用）	Management Server が内部の管理用に利用するポート番号（mserver.properties ファイルの webserver.connector.ajp13.port	28009

入力項目				説明	初期値※2
タブ	分類	項番	HCSC 簡易セットアップ画面の項目名※1		
メイン	Management Server	(6)	内部通信ポート番号（内部管理用）	キー）を指定します。1～65535 の整数で指定します。	28009
		(7)	インプロセスネーミングサービスポート番号（内部管理用）	Management Server が内部の管理用に利用するポート番号（mserver.properties ファイルの ejbserver.naming.port キー）を指定します。1～65535 の整数で指定します。	28900
	運用管理エージェント	(8)	エージェント接続ポート番号（内部管理用）	Management Server から運用管理エージェントにアクセスする場合に使用する内部管理用のポート番号を指定します。1～65535 の整数で指定します。	20295
	J2EE サーバ	－	V9 互換モード	V9 互換モードを使用する構成の場合に選択します。 また，JDK 11 をインストールした環境の場合に選択できます。JDK 17 をインストールした環境の場合は選択できません。	－
		－	推奨モード	V9 互換モードを使用しない構成の場合に選択します。 このモードを選択すると，HCSC サーバの［SOAP1.1/1.2 併用モード］が選択された状態になります。	選択
	HCSC サーバ	－	SOAP1.1 モード	SOAP1.1 モードを使用する場合に選択します。 このモードを選択すると，J2EE サーバの［V9 互換モード］が選択された状態になります。J2EE サーバで［推奨モード］を選択しているときは使用できません。 また，JDK 11 をインストールした環境の場合に選択できます。JDK 17 をインストールした環境の場合は選択できません。	－
		－	SOAP1.1/1.2 併用モード	SOAP1.1/1.2 併用モードを使用する場合に選択します。 J2EE サーバで［推奨モード］を選択しているときは，このモードが選択された状態になります。	選択

入力項目				説明	初期値※2
タブ	分類	項番	HCSC 簡易セットアップ画面の項目名※1		
メイン	HCSC サーバ	(9)	Web サービス/MDB（WS-R）受付ポート番号	サービスリクエスタから標準受付（Web サービスまたは MDB（WS-R)），またはユーザ定義受付にアクセスする場合に使用するポート番号を指定します。1～65535 の整数で指定します。	80
		(10)	SessionBean 受付ポート番号	Management Server から HCSC サーバに，またはサービスリクエスタから標準受付（SessionBean）にアクセスする場合に使用するポート番号を指定します。1～65535 の整数で指定します。	900
		(11)	MDB（DB キュー）受付ポート番号	サービスリクエスタから標準受付（MDB（DB キュー)）にアクセスする場合に使用するポート番号を指定します。1024～65535 の整数で指定します。 [DB/RM ありモデル］を選択した場合だけ指定します。	20351
		(12)	稼働確認ポート番号	HCSC サーバが内部の管理用に利用するポート番号を指定します。1～65535 の整数で指定します。	23152
		(13)	簡易 Web サーバポート番号（内部管理用）	HCSC サーバが内部の管理用に利用するポート番号を指定します。1～65535 の整数で指定します。	8080
		(14)	稼働情報取得時のリクエスト受付ポート番号（内部管理用）	JMX クライアントによる稼働情報取得時のリクエスト受付のポート番号を 0～65535 の整数で指定します。	23550
サーバ名称	サーバ名称の選択	－	V7 互換名称	HCSC 簡易セットアップで構築するテスト環境の各種サーバ名称について，バージョン 7 と同じ名称を使用する場合に選択します。	－
		－	HCSC 本番環境簡易セットアップ名称	HCSC 簡易セットアップで構築するテスト環境の各種サーバ名称について，HCSC 本番環境簡易セットアップでセットアップする名称と同じ名称を使用する場合に選択します。 HCSC 本番環境簡易セットアップについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.5　本番環境の簡易セットアップ」を参照してください。	選択

入力項目				説明	初期値※2
タブ	分類	項番	HCSC 簡易セットアップ画面の項目名※1		
サーバ名称	サーバ名称の選択	－	カスタム名称	HCSC 簡易セットアップで構築するテスト環境の各種サーバ名称について，任意の名称を指定する場合に選択します。	－
	サーバ名称	(15)	論理 J2EE サーバ名称	接続先の J2EE サーバ名称を指定します。1～128 文字以内の半角英数字，アンダーバーおよびハイフンを指定します。	J2EEServer
		(16)	論理 PRF 名称	HCSC サーバと連動して動作する PRF のサーバ名称を指定します。1～128 文字以内の半角英数字，アンダーバーおよびハイフンを指定します。	PRF
		－	クラスタ名称	セットアップする HCSC サーバが属するクラスタの名称を指定します。クラスタ間で一意になるクラスタ名を指定します。1～8 文字以内の半角英数字およびアンダーバーを指定します。	Cluster
		(17)	HCSC サーバ名称	セットアップする HCSC サーバの名称を指定します。1～8 文字以内の半角英数字およびアンダーバーを指定します。	HCSC
		(18)	Manager 名称	HCSC-Manager が Manager を認識するための HCSC-Manager 独自の識別名を指定します。1 つのリポジトリで複数の HCSC サーバを管理している場合，Manager（ホスト）単位に一意になる識別名を指定します。1～16 文字以内の半角英数字およびアンダーバーで指定します。	Manager

（凡例）

－：該当しません。

注※1

HCSC 簡易セットアップ画面の対応する項目の上でマウスカーソルを一時停止すると，その項目の説明文が一時的に表示されます。

注※2

初期値は，初めて HCSC 簡易セットアップ画面を表示したときに初期表示される値です。初期値を変更してセットアップした場合，次回の HCSC 簡易セットアップ画面には前回の設定値が初期値として表示されます。

(2) テスト環境のアンセットアップ

HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境をアンセットアップする場合の前提条件，およびアンセットアップ方法について説明します。

注意事項

HCSC 簡易セットアップ機能を利用してアンセットアップすると，テスト環境はセットアップ前の状態に戻ります。テスト環境で利用していたサービスアダプタやビジネスプロセスなどは自動的に削除されるので，必要なデータは，アンセットアップ前に必ずバックアップを取得してください。

(a) 前提条件

テスト環境をアンセットアップするための前提条件を次に示します。

- アンセットアップの対象が，HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境であること。
- Management Server，および運用管理エージェントが起動していること。
- HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択してテスト環境をセットアップした場合，組み込みデータベースのサービス部品が起動していること。

(b) アンセットアップ方法

テスト環境のアンセットアップには，HCSC 簡易セットアップ画面を利用します。テスト環境のアンセットアップ方法を次に示します。

1. Windows の［スタート］メニューから，［Cosminexus］－［テスト環境セットアップ］を選択します。
   HCSC 簡易セットアップ画面が表示されます。
2. HCSC 簡易セットアップ画面の［アンセットアップ］ボタンをクリックします。
   テスト環境のアンセットアップが開始されます。アンセットアップの状況は，HCSC 簡易セットアップ画面のコンソールに表示されます。コンソールに「HCSC 簡易セットアップ機能のアンセットアップを終了します」と表示されたら，テスト環境は正常にアンセットアップされています。

2.4.3 HCSC 簡易セットアップ機能でセットアップしたテスト環境の運用

HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境の運用方法，および運用に必要な情報について説明します。

(1) テスト環境の起動と停止

テスト環境の起動および停止方法について，次の表に示します。

表 2-6　テスト環境の起動および停止方法

起動・停止の対象	起動・停止方法
組み込みデータベース※	起動する場合： 1. Windows の［スタート］メニューの［プログラム］から，［HiRDBSingleServer_CS0］－［HiRDB コマンドプロンプト］を選択します。 2. pdstart コマンドを実行します。 停止する場合： 1. Windows の［スタート］メニューの［プログラム］から，［HiRDBSingleServer_CS0］－［HiRDB コマンドプロンプト］を選択します。 2. pdstop コマンドを実行します。
組み込みデータベースのサービス※	テスト環境のマシンを起動または停止すると，自動的に起動または停止されます。
• Management Server • 運用管理エージェント	テスト環境のマシンを起動すると，自動的に起動されます。また，テスト環境のマシンを停止すると，自動的に停止されます。
• Performance Tracer • J2EE サーバ • HCSC サーバ • 標準受付	起動する場合： Windows の［スタート］メニューから，［Cosminexus］－［テストサーバ起動］を選択します。 停止する場合： Windows の［スタート］メニューから，［Cosminexus］－［テストサーバ停止］を実行します。

注※
HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ実施します。

(2)　テスト環境の運用に必要な情報

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップした場合，必要な情報は自動的に設定されます。テスト環境を使ってテストおよびデバッグを行うには，HCSC 簡易セットアップ機能でテスト環境をセットアップする際に設定される情報を知っておく必要があります。

HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境に設定される情報を，次に説明します。

ポイント

HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境に設定される情報は，テスト環境のセットアップ後にカスタマイズできます。カスタマイズの方法については，「2.4.4　テスト環境のカスタマイズ」を参照してください。

(a) ユーザ ID およびパスワードの情報

HCSC 簡易セットアップ機能でセットアップしたテスト環境に設定されるユーザ ID およびパスワードの情報を次の表に示します。

表 2-7 テスト環境に設定されるユーザ ID およびパスワード

設定先	設定されるユーザ ID またはパスワード	初期値	説明
組み込みデータベース※1	ユーザ ID	admin※2	テーブル所有者の認可識別子（環境変数の USRID）。
	パスワード	admin※2	テーブル所有者のパスワード（環境変数の PSWD）。
Management Server	管理ユーザ ID	admin	Management Server にログインするための管理ユーザ ID（cmx_build_system -change コマンド※3 などで使用）。
	パスワード	admin	Management Server にログインするためのパスワード（cmx_build_system -change コマンド※3 などで使用）。
HCSC-Manager	HCSC-Manager ログインユーザ ID	admin	HCSC-Manager にログインするためのユーザ ID（cscsvstart コマンド※4 などで使用）。
	HCSC-Manager ログインパスワード	admin	HCSC-Manager にログインするためのパスワード（cscsvstart コマンド※4 などで使用）。
HCSC サーバ	HCSC サーバが使用するデータベースのユーザ ID	admin	HCSC サーバが使用するデータベースのユーザ ID（csccompodeploy コマンド※4 などで使用）。
	HCSC サーバが使用するデータベースのパスワード	admin	HCSC サーバが使用するデータベースのパスワード（csccompodeploy コマンド※4 などで使用）。

注※1

HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ設定されます。

注※2

組み込みデータベース内では，大文字「ADMIN」で扱われます。

注※3

このコマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cmx_build_system（Web システムの構築）」を参照してください。

注※4

cscsvstart コマンドについては，マニュアル「サービスプラットフォーム リファレンス」の「cscsvstart（HCSC サーバの起動）」を参照してください。csccompodeploy コマンドについては，マニュアル「サービスプラットフォーム リファレンス」の「csccompodeploy（HCSC コンポーネントの配備）」を参照してください。

(b)　ポート番号の情報

HCSC 簡易セットアップ機能でセットアップしたテスト環境には，次に示すポート番号が設定されます。ポート番号は，HCSC 簡易セットアップ画面で任意の値を指定できます。HCSC 簡易セットアップ画面での指定値の詳細については，「表 2-5　HCSC 簡易セットアップ画面で入力する項目」を参照してください。

組み込みデータベース（HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ）

- DB 接続ポート番号

Management Server

- HCSC サーバ運用ポート番号
- 論理サーバ運用ポート番号
- 終了要求受信ポート番号
- 内部通信ポート番号
- インプロセスネーミングサービスポート番号

運用管理エージェント

- エージェント接続ポート番号

J2EE サーバ

- Web サービス/MDB（WS-R）受付ポート番号
- SessionBean 受付ポート番号
- MDB（DB キュー）受付ポート番号
- 稼働確認ポート番号
- 簡易 Web サーバポート番号

(c)　名称の情報

HCSC 簡易セットアップ機能でセットアップしたテスト環境に設定されるホスト名やサーバ名などの情報を次の表に示します。なお，HCSC 簡易セットアップ画面の［サーバ名称］タブでサーバ名称を変更した場合は，変更後の名称が設定されます。

表 2-8　テスト環境に設定される名称

設定先	設定される名称	初期値	説明
組み込みデータベース※1	ホスト名	localhost	組み込みデータベースのホスト名（環境変数の DB_HOST）。
Management Server	ホスト名	localhost	Management Server のホスト名（cmx_build_system -change コマンド※2 などで使用）。

設定先	設定される名称	初期値	説明
HCSC-Manager	Manager 名	Manager	HCSC-Manager が Manager を認識するための HCSC-Manager 独自の識別名（cscsvstart コマンド※3 などで使用）。
PRF	サーバ名	PRF	PRF のサーバ名称。
J2EE サーバ	サーバ名	J2EEServer	接続先 J2EE サーバの名称（cjstartsv コマンド※2 などで使用）。
	ホスト名	localhost	接続先 J2EE サーバのホスト名。
	Reliable Messaging が連携するシステム全体で一意なシステム名	RM	Reliable Messaging が連携するシステム全体で一意なシステム名（環境変数の HRM_SYSTEM_NAME）。
	リソースアダプタ名	Reliable Messaging	対象となる RAR ファイルの表示名（cjdeployrar コマンド※2 などで使用）。
		DB_Connector_for_HiRDB_Type4	
		DB_Connector_for_HiRDB_Type4_Cosminexus_RM	
クラスタ	クラスタ名	Cluster	クラスタの名称。
HCSC サーバ	HCSC サーバ名	HCSC	HCSC サーバの名称（cscsvstart コマンド※3 などで使用）。

注※1

HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ設定されます。

注※2

cmx_build_system -change コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cmx_build_system（Web システムの構築）」を参照してください。cjstartsv コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartsv（J2EE サーバの開始）」を参照してください。cjdeployrar コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeployrar（リソースアダプタのデプロイ）」を参照してください。

注※3

このコマンドについては，マニュアル「サービスプラットフォーム リファレンス」の「cscsvstart（HCSC サーバの起動）」を参照してください。

(d) そのほかの情報

組み込みデータベース（HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合だけ）

HCSC 簡易セットアップ画面で［DB あり/RM なしモデル］または［DB/RM ありモデル］を選択した場合，HCSC 簡易セットアップ機能でセットアップしたテスト環境の組み込みデータベースには，次の情報が設定されます。

データ（RD エリア）格納先

RD エリア格納先の情報は，HCSC 簡易セットアップ画面で任意の値を指定できます。HCSC 簡易セットアップ画面での指定値の詳細については，「表 2-5　HCSC 簡易セットアップ画面で入力する項目」を参照してください。

構築するデータベースの領域サイズ

構築するデータベースの領域サイズ（環境変数の DB_SIZE）が設定されます。

初期値は「2GB」です。

J2EE サーバの互換モード

J2EE サーバで V9 互換モードを使用するかどうか，HCSC 簡易セットアップ画面で選択できます。HCSC 簡易セットアップ画面の詳細については，「表 2-5　HCSC 簡易セットアップ画面で入力する項目」を参照してください。初期値は「推奨モード」（V9 互換モードを使用しない）です。

(3)　組み込みデータベースの運用

HCSC 簡易セットアップ機能を利用して設定した組み込みデータベースは，定期的にデータベースの未使用領域を確認する必要があります。未使用領域が少ない場合は，空きページおよび空きセグメントを解放したり，不要な実行履歴を削除したりします。詳細については，「2.5　HCSC 簡易セットアップ機能でセットアップした組み込みデータベースの運用」を参照してください。

(4)　障害対策

HCSC 簡易セットアップ機能を利用して，HCSC サーバのセットアップやアンセットアップ，HCSC コンポーネントの配備や削除，およびリクエストを送信し続けると，組み込みデータベース内で未使用のセグメントが不足するおそれがあります。

未使用のセグメントが不足した場合，次の対策を実施してください。

方法 1：組み込みデータベース内の空きセグメントを解放する

1. 組み込みデータベース内の表を HiRDB SQL Executer で検索します。

   表の検索方法については，マニュアル「HiRDB SQL Executer オンラインヘルプ」の pdsql コマンドを参照してください。

2. 手順 1.で検索した結果から，「CSC」および「RM」で始まるすべての表に対して，pdreclaim コマンドを発行し，空きページを解放して再利用します。

   pdreclaim コマンドの詳細については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

方法 2：組み込みデータベース内の表を再編成する

1. 組み込みデータベース内の表を HiRDB SQL Executer で検索します。

   表の検索方法については，マニュアル「HiRDB SQL Executer オンラインヘルプ」の pdsql コマンドを参照してください。

2. 手順 1.で検索した結果から，「CSC」および「RM」で始まるすべての表に対して pdrorg コマンドを発行し，表を再編成します。

   表を再編成する方法については，マニュアル「HiRDB システム運用ガイド」の表の再編成に関する内容を参照してください。

   pdrorg コマンドの詳細については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

方法 3：HCSC 簡易セットアップ機能で構築したテスト環境を再セットアップする

HCSC 簡易セットアップ機能で構築したテスト環境をいったんアンセットアップし，再度セットアップします。この作業を行うと，組み込みデータベースが再生成されます。

ただし，再セットアップすると，作成したサービスアダプタやビジネスプロセスなども削除されます。そのため，あらかじめリポジトリをエクスポートしてから，再セットアップし，再セットアップ後にリポジトリをインポートしてください。

(5)　テスト環境マシンの IP アドレスまたはホスト名を変更する場合の運用

HCSC 簡易セットアップ機能を利用してテスト環境をセットアップした場合，テスト環境マシンの IP アドレスまたはホスト名の変更後にテスト環境を再度セットアップする必要はありません。

ただし，テスト環境を起動している場合は再起動してください。

2.4.4　テスト環境のカスタマイズ

HCSC 簡易セットアップ機能を利用してセットアップしたテスト環境には，「2.4.3(2)　テスト環境の運用に必要な情報」に示す情報が設定されています。これらの情報は，テスト環境をセットアップしたあとにカスタマイズできます。

テスト環境をカスタマイズするには，HCSC 簡易セットアップ機能でのテスト環境のセットアップ時に定義ファイルに設定された値を編集します。

編集できる定義ファイルを次の表に示します。各定義ファイルの詳細については，表の「参照先マニュアル」に示すマニュアルを参照してください。

表 2-9　編集できる定義ファイル

ファイル名	説明	格納先	参照先マニュアル	参照個所
adminagent.properties（運用管理エージェントプロパティファイル）	運用管理エージェントとの通信で使用するポート番号が設定されているプロパティファイルです。	A	アプリケーションサーバ リファレンス 定義編（サーバ定義）	8.2.1
cdsetupconfig.bat（組み込みデータベース設定バッチファイル）	組み込みデータベースの環境変数を設定するバッチファイルです。	B	HiRDB のマニュアル	−

ファイル名	説明	格納先	参照先マニュアル	参照個所
cmdconf.bat （HCSC-Messaging コマンド共通定義ファイル）	Service Platform の csmXXX コマンドで組み込みデータベースにアクセスするために，HiRDB type4 JDBC Driver のクラスパスを設定するバッチファイルです。	C	サービスプラットフォーム リファレンス	6.5.4 HCSC-Messaging コマンド共通定義ファイル
Cosminexus_Reliable_Messaging.xml （Connector 属性ファイル）	リソースアダプタの属性（コンフィグレーションプロパティの値やプロパティ値）が設定されている XML ファイルです。	D	アプリケーションサーバ リファレンス 定義編（アプリケーション／リソース定義）	4.1
csccmd.properties （HCSC-Manager コマンド共通定義ファイル）	運用環境で使用するコマンドの省略値（ログインユーザ ID やログインパスワード）が設定されているプロパティファイルです。	E	サービスプラットフォーム リファレンス	6.5.2 HCSC-Manager コマンド共通定義ファイル
cscmng.properties （HCSC-Manager 定義ファイル）	HCSC-Manager の動作に必要な情報が設定されているプロパティファイルです。	E	サービスプラットフォーム リファレンス	6.5.3 HCSC-Manager 定義ファイル
cscsvconfig.properties （HCSC サーバランタイム定義ファイル）	HCSC サーバの起動時に必要なランタイム情報（実行履歴管理情報，データベース情報）が設定されているプロパティファイルです。	D	サービスプラットフォーム リファレンス	6.5.6 HCSC サーバランタイム定義ファイル
cscsvsetup.properties （HCSC サーバセットアップ定義ファイル（DB あり/RM なしモデル））※1	DB あり/RM なしモデルで使用する，HCSC サーバのセットアップに必要な情報（J2EE サーバ，Reliable Messaging，データベースに関連する情報）が設定されているプロパティファイルです。	D	サービスプラットフォーム リファレンス	6.4.3 HCSC サーバセットアップ定義ファイル
cscsvsetup.properties.esb （HCSC サーバセットアップ定義ファイル（DB/RM なしモデル））※1	DB/RM なしモデルで使用する，HCSC サーバのセットアップに必要な情報（J2EE サーバ，Reliable Messaging，データベースに関連する情報）が設定されているプロパティファイルです。	D	サービスプラットフォーム リファレンス	6.4.3 HCSC サーバセットアップ定義ファイル
cscsvsetup.properties.rm （HCSC サーバセットアップ定義ファイル（DB/RM ありモデル））※1	DB/RM ありモデルで使用する，HCSC サーバのセットアップに必要な情報（J2EE サーバ，Reliable Messaging，データベースに関連する情報）が設定されているプロパティファイルです。	D	サービスプラットフォーム リファレンス	6.4.3 HCSC サーバセットアップ定義ファイル
cscsvsetup.xml （HCSC サーバ構成定義ファイル（DB あり/RM なしモデル））	DB あり/RM なしモデルで使用する，HCSC サーバの構成情報（cosminexus-manager，jms-physical-reception，ejb-reception	D	サービスプラットフォーム リファレンス	6.4.2 HCSC サーバ構成定義ファイル

ファイル名	説明	格納先	参照先マニュアル	参照個所
cscsvsetup.xml (HCSC サーバ構成定義ファイル（DB あり/RM なしモデル))	の定義）が設定されている XML ファイルです。	D	サービスプラットフォーム リファレンス	6.4.2 HCSC サーバ構成定義ファイル
cscsvsetup.xml.esb (HCSC サーバ構成定義ファイル（DB/RM なしモデル))	DB/RM なしモデルで使用する，HCSC サーバの構成情報（cosminexus-manager，jms-physical-reception，ejb-reception の定義）が設定されている XML ファイルです。	D	サービスプラットフォーム リファレンス	6.4.2 HCSC サーバ構成定義ファイル
cscsvsetup.xml.rm (HCSC サーバ構成定義ファイル（DB/RM ありモデル))	DB/RM ありモデルで使用する，HCSC サーバの構成情報（cosminexus-manager，jms-physical-reception，ejb-reception の定義）が設定されている XML ファイルです。	D	サービスプラットフォーム リファレンス	6.4.2 HCSC サーバ構成定義ファイル
DB_Connector_for_HiRDB_Type4.xml (Connector 属性ファイル)	リソースアダプタの属性（コンフィグレーションプロパティの値，プロパティ値）が設定されている XML ファイルです。	D	アプリケーションサーバ リファレンス 定義編（アプリケーション／リソース定義）	4.1
DB_Connector_for_HiRDB_Type4_LT.xml (Connector 属性ファイル)	リソースアダプタの属性（コンフィグレーションプロパティの値，プロパティ値）が設定されている XML ファイルです。 RM なし環境で使用するローカルトランザクション用の定義です。	D	アプリケーションサーバ リファレンス 定義編（アプリケーション／リソース定義）	4.1
DB_Connector_for_HiRDB_Type4_Cosminexus_RM.xml (Connector 属性ファイル)	リソースアダプタの属性（コンフィグレーションプロパティの値，プロパティ値）が設定されている XML ファイルです。	D	アプリケーションサーバ リファレンス 定義編（アプリケーション／リソース定義）	4.1
grantuser (ユーザ定義ファイル)	HiRDB のユーザの作成，およびスキーマの作成を行うファイルです。	B	HiRDB のマニュアル	−
hrmqueue-transmit.xml (アプリケーション属性ファイル)	アプリケーションの属性が設定されている XML ファイルです。	D	アプリケーションサーバ リファレンス 定義編（アプリケーション／リソース定義）	3.2
input.properties (システム共通情報記載プロパティファイル)	HCSC 簡易セットアップ画面で入力した情報，およびシステム共通の情報が設定されているプロパティファイルです。	D	−	−
inserttableshirdb.sql (テーブルインサート SQL ファイル)	システム管理情報テーブルにレコードをインサートする SQL ファイルです。	D	−	−

ファイル名	説明	格納先	参照先マニュアル	参照個所
mserver.cfg (Management Server 用オプション定義ファイル)	Management Server を実行する JavaVM の起動オプションが設定されているファイルです。	A	アプリケーションサーバ リファレンス 定義編 (サーバ定義)	8.2.7
mserver.properties (Management Server 環境設定ファイル)	Management Server が使用するポート番号や，障害検知時コマンドの動作が設定されているプロパティファイルです。	A	アプリケーションサーバ リファレンス 定義編 (サーバ定義)	8.2.6
qconf.txt (キュー定義ファイル)	キュー情報（DisplayName, QueueName）が設定されているテキストファイルです。	E	Reliable Messaging	3.4.4
setupscript (セットアップスクリプトファイル（DB あり/RM なしモデル・SOAP1.1 モード用))	DB あり/RM なしモデルで使用する，セットアップ処理のタスクの実行順が設定されているスクリプトファイルです。SOAP1.1 モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.cjw (セットアップスクリプトファイル（DB あり/RM なしモデル・SOAP1.1/1.2 併用モード用))	DB あり/RM なしモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.cjw.v11 (セットアップスクリプトファイル（DB あり/RM なしモデル・SOAP1.1/1.2 併用モード・推奨モード用))	DB あり/RM なしモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが推奨モードの環境をセットアップします。	D	−	−
setupscript.esb (セットアップスクリプトファイル（DB/RM なしモデル・SOAP1.1 モード用))	DB/RM なしモデルで使用する，セットアップ処理のタスクの実行順が設定されているスクリプトファイルです。SOAP1.1 モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.esb.cjw (セットアップスクリプトファイル（DB/RM なしモデル・SOAP1.1/1.2 併用モード用))	DB/RM なしモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.esb.cjw.v11 (セットアップスクリプトファイル（DB/RM なしモ	DB/RM なしモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。SOAP1.1/1.2 併用モード，かつ	D	−	−

ファイル名	説明	格納先	参照先マニュアル	参照個所
デル・SOAP1.1/1.2 併用モード・推奨モード用))	J2EE サーバの互換モードが推奨モードの環境をセットアップします。	D	−	−
setupscript.rm (セットアップスクリプトファイル（DB/RM ありモデル・SOAP1.1 モード用))	DB/RM ありモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。 SOAP1.1 モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.rm.cjw (セットアップスクリプトファイル（DB/RM ありモデル・SOAP1.1/1.2 併用モード用))	DB/RM ありモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。 SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップします。	D	−	−
setupscript.rm.cjw.v11 (セットアップスクリプトファイル（DB/RM ありモデル・SOAP1.1/1.2 併用モード・推奨モード用))	DB/RM ありモデルで使用する，セットアップ処理のタスクが実行順に記述されたスクリプトファイルです。 SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが推奨モードの環境をセットアップします。	D	−	−
simple_model.xml (HCSC 簡易セットアップ定義ファイル・SOAP1.1 モード用）※2	Smart Composer 機能のコマンドで構築する Web システムが定義されている XML ファイルです。SOAP1.1 モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップするときに使用します。	D	−	−
simple_model_cjl.xml (HCSC 簡易セットアップ定義ファイル・SOAP1.1/1.2 併用モード用）※2	Smart Composer 機能のコマンドで構築する Web システムが定義されている XML ファイルです。 SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが V9 互換モードの環境をセットアップするときに使用します。	D	−	−
simple_model_cjw_v11.xml (HCSC 簡易セットアップ定義ファイル・SOAP1.1/1.2 併用モード・推奨モード用）※3	Smart Composer 機能のコマンドで構築する Web システムが定義されている XML ファイルです。 SOAP1.1/1.2 併用モード，かつ J2EE サーバの互換モードが推奨モードの環境をセットアップするときに使用します。	D	−	−
tablecreate (テーブル定義ファイル)	HiRDB のスキーマ，およびテーブルの作成を行うファイルです。	B	HiRDB のマニュアル	−
unsetupscript	DB あり/RM なしモデルで使用する，アンセットアップ処理のタスクの実行	D	−	−

ファイル名	説明	格納先	参照先マニュアル	参照個所
(アンセットアップスクリプトファイル（DB あり/RM なしモデル))	順が設定されているスクリプトファイルです。	D	－	－
unsetupscript.esb (アンセットアップスクリプトファイル（DB/RM なしモデル))	DB/RM なしモデルで使用する，アンセットアップ処理のタスクの実行順が設定されているスクリプトファイルです。	D	－	－
unsetupscript.rm (アンセットアップスクリプトファイル（DB/RM ありモデル))	DB/RM ありモデルで使用する，アンセットアップ処理のタスクの実行順が設定されているスクリプトファイルです。	D	－	－

(凡例)

A：<サービスプラットフォームのインストールディレクトリ>¥manager¥config

B：<サービスプラットフォームのインストールディレクトリ>¥CSC¥DB¥bats

C：<サービスプラットフォームのインストールディレクトリ>¥CSC¥config¥msg

D：<サービスプラットフォームのインストールディレクトリ>¥CSC¥system¥manager¥setup

E：<サービスプラットフォームのインストールディレクトリ>¥CSC¥config¥manager

－：参照するマニュアルはありません。

注※1

セットアップ構成関連のプロパティ（db-use プロパティ，rm-use プロパティ，および hcscserver-data-filepath プロパティ）は変更できません。

注※2

J2EE サーバの互換モードが V9 互換モードの場合，運用に合わせて次のようにプロパティを変更してください。

・HCSC 簡易セットアップ機能で構築する環境の Web コンテナ単位の同時実行数を変更する場合は，
webserver.connector.inprocess_http.max_execute_threads プロパティを変更してください。
webserver.connector.inprocess_http.max_execute_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 互換編」の「5.11.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

・リクエスト処理で使用するコネクション数を合わせて制御する場合は，
webserver.connector.inprocess_http.max_connections プロパティを変更してください。
webserver.connector.inprocess_http.max_connections プロパティについては，マニュアル「アプリケーションサーバ 機能解説 互換編」の「6.3.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

注※3

J2EE サーバの互換モードが推奨モードの場合，運用に合わせて次のようにプロパティを変更してください。

・HCSC 簡易セットアップ機能で構築する環境の Web コンテナ単位の同時実行数を変更する場合は，
webserver.connector.nio_http.max_servlet_execute_threads プロパティを変更してください。
webserver.connector.nio_http.max_servlet_execute_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「2.14.2　実行環境での設定（J2EE サーバの設定))」を参照してください。

・リクエスト処理で使用するスレッド数，コネクション数を合わせて制御する場合は，次のプロパティを変更してください。

・webserver.connector.nio_http.max_threads

・webserver.connector.nio_http.min_threads

・webserver.connector.nio_http.max_connections

webserver.connector.nio_http.max_threads プロパティ，および webserver.connector.nio_http.min_threads プロパティについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「7.4.2　実行環境での設定(J2EE サーバの設定)」を参照してください。webserver.connector.nio_http.max_connections プロパティについては，マ

ニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」の「7.3.2　実行環境での設定（J2EE サーバの設定)」を参照してください。

参考

テスト環境をカスタマイズする場合，次のディレクトリ内のファイルをカスタマイズの参考として利用できます。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥log¥manager¥setup
```

1 台のマシンに複数のテスト環境を構築（環境変数 CSCMNG_HOME を設定）している場合は，次のディレクトリ内のファイルをカスタマイズの参考として利用できます。

```
%CSCMNG_HOME%¥log¥manager¥setup
```

これらのディレクトリに格納されているファイルを次に示します。

表 2-9 に示す定義ファイルと同じ名称のファイル

HCSC 簡易セットアップ機能を実行したときに設定された定義ファイルと同じ内容のファイル（HCSC 簡易セットアップ機能のログファイル）です。

adminagent.properties.bak（adminagent.properties のバックアップファイル）

現在の adminagent.properties の 1 つ前の adminagent.properties の設定が保存されています。

cmdconf.bat.bak（cmdconf.bat のバックアップファイル）

現在の cmdconf.bat の 1 つ前の cmdconf.bat の設定が保存されています。

csccmd.properties.bak（csccmd.properties のバックアップファイル）

現在の csccmd.properties の 1 つ前の csccmd.properties の設定が保存されています。

cscmng.properties.bak（cscmng.properties のバックアップファイル）

現在の cscmng.properties の 1 つ前の cscmng.properties の設定が保存されています。

mserver.cfg.bak（mserver.cfg のバックアップファイル）

現在の mserver.cfg の 1 つ前の mserver.cfg の設定が保存されています。

mserver.properties.bak（mserver.properties のバックアップファイル）

現在の mserver.properties の 1 つ前の mserver.properties の設定が保存されています。

2.4.5　SOAP モードの確認

現在使用している SOAP モードの確認方法について説明します。

(1) 開発環境の SOAP モードを確認する場合

開発環境の SOAP モードは Eclipse 上で確認できます。

1. Eclipse のメニューから，［ウィンドウ］－［設定］を選択します。
   ［設定］ダイアログが表示されます。

2. ダイアログ左側のツリービューで［HCSC-Definer］を選択します。
   ダイアログ右側の［現在の SOAP モード］に使用している SOAP モードが表示されます。

(2) 実行環境の SOAP モードを確認する場合

実行環境で現在使用している SOAP モードを確認する場合，cscrepls コマンドを使用します。コマンドを実行すると表示項目の SOAP-mode に次のどちらかが表示されます。

- 1.1：SOAP1.1 モードを示します。
- 1.1/1.2：SOAP1.1/1.2 併用モードを示します。

なお，cscrepls コマンドは，HCSC サーバのセットアップが完了している場合に実行できます。

cscrepls コマンドについては，マニュアル「サービスプラットフォーム リファレンス」の「cscrepls（リポジトリ情報の表示)」を参照してください。

2.5 HCSC 簡易セットアップ機能でセットアップした組み込みデータベースの運用

HCSC 簡易セットアップ機能でセットアップした組み込みデータベースの運用について説明します。

2.5.1 RD エリアの未使用領域を確認する

RD エリアの未使用領域を確認し，データベースの使用状況を調査します。RD エリアの未使用領域の確認は，定期的に実施する必要があります。RD エリアの未使用領域を確認する前提条件と確認方法について次に説明します。

(1) 前提条件

RD エリアの未使用領域を確認するには，データベースが起動状態である必要があります。

OS のシステム環境変数（PDHOST，PDNAMEPORT，PDUSER）に，組み込みデータベースとは異なる HiRDB サーバの接続情報を設定していない状態である必要があります。設定している場合，HiRDB コマンドプロンプト起動後に set コマンドでシステム環境変数を削除してください。

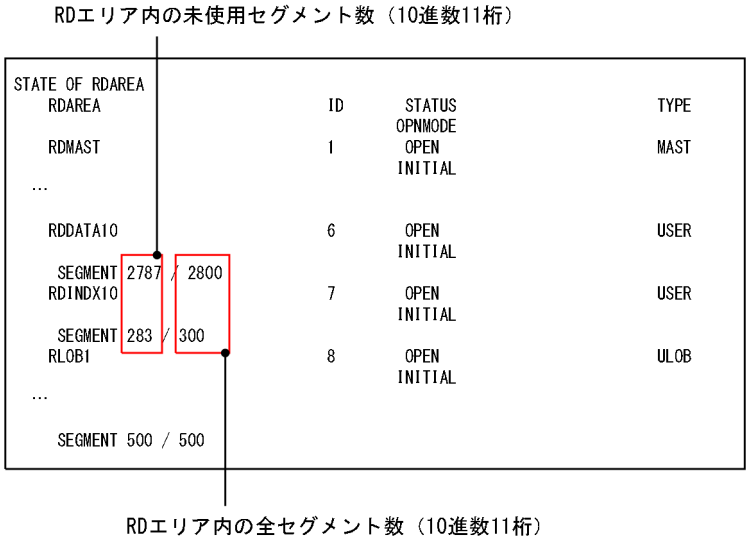
(2) 確認方法

RD エリアの未使用領域を確認する方法を次に示します。

1. Windows の［スタート］メニューの［プログラム］から，［HiRDBSingleServer_CS0］ − ［HiRDB コマンドプロンプト］を選択します。
   組み込みデータベースを操作する環境変数が設定されます。
2. pddbls コマンドを実行し，RD エリアの状態を表示します。
   pddbls コマンドの詳細については，マニュアル「HiRDB コマンドリファレンス」を参照してください。
   pddbls コマンドの実行形式を次に示します。

```
pddbls -r ALL -a
```

3. コマンドの実行結果から RD エリアの未使用領域を確認します。
   実行結果の例を次に示します。



4. RD エリアの未使用領域が少ない場合は，次の方法で未使用領域の不足を解消します。

- プロセスインスタンスの実行履歴を削除する
  「2.5.2　プロセスインスタンスの実行履歴を削除する」を参照してください。
- 空きページおよび空きセグメントを解放する
  「2.5.3　空きページおよび空きセグメントを解放する」を参照してください。

2.5.2　プロセスインスタンスの実行履歴を削除する

ビジネスプロセスのプロセスインスタンスの実行状況を履歴としてデータベースに記録している場合，実行履歴は日々追加されていくため，データベースの容量が圧迫されます。そのため，定期的にプロセスインスタンスの実行履歴を削除する必要があります。プロセスインスタンスの実行履歴を削除する前提条件と確認方法について次に説明します。

(1)　前提条件

プロセスインスタンスの実行履歴を削除するには，データベースが起動状態である必要があります。

(2)　削除方法

1. アクセスする HCSC サーバを決定します。

   HCSC 簡易セットアップ機能でセットアップした情報をカスタマイズしていない場合，HCSC サーバ名には「HCSC」が設定されています。

2. テスト環境（運用環境）で cscpidelete コマンドを実行します。

   プロセスインスタンスの実行履歴を削除する方法については，マニュアル「サービスプラットフォームシステム構築・運用ガイド」の「6.1.4　プロセスインスタンスの実行履歴を削除する」を参照してください。

3. 空きページおよび空きセグメントを解放します。

空きページおよび空きセグメントを解放する方法については，「2.5.3　空きページおよび空きセグメントを解放する」を参照してください。

2.5.3　空きページおよび空きセグメントを解放する

データの削除によって生じた空きページおよび空きセグメントを解放して，未使用状態に戻します。空きページおよび空きセグメントを解放する前提条件と解放方法について次に説明します。

注意事項

空きページだけでなく空きセグメントも必ず解放してください。

(1)　前提条件

空きページおよび空きセグメントを解放するには，次の状態を満たす必要があります。

- HCSC サーバが停止状態であること。
- データベースが起動状態であること。

OS のシステム環境変数（PDHOST，PDNAMEPORT，PDUSER）に，組み込みデータベースとは異なる HiRDB サーバの接続情報を設定していない状態である必要があります。設定している場合，HiRDB コマンドプロンプト起動後に set コマンドでシステム環境変数を削除してください。

(2)　解放方法

空きページおよび空きセグメントを解放する方法を次に示します。なお，パスワードの入力を求められた場合は，「admin」を入力してください。

1. Windows の［スタート］メニューの［プログラム］から，［HiRDBSingleServer_CS0］ − ［HiRDB コマンドプロンプト］を選択します。

組み込みデータベースを操作する環境変数が設定されます。

2. pdreclaim コマンドを実行し，使用中空きページを解放します。

使用中空きページ（データの削除によってデータが格納されていないページ）を解放して，未使用ページにします。

pdreclaim コマンドの詳細については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

pdreclaim コマンドの実行形式を次に示します。

表の使用中空きページを解放する場合

```
pdreclaim -u admin -k table -t all -o
```

インデクスの使用中空きページを解放する場合

```
pdreclaim -u admin -k index -t all
```

3. pdreclaim コマンドを実行し，使用中空きセグメントを解放します。

使用中空きセグメント（セグメント内の全ページが空きページになっているセグメント）を解放して，未使用セグメントにします。

pdreclaim コマンドの詳細については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

pdreclaim コマンドの実行形式を次に示します。

表の使用中空きセグメントを解放する場合

```
pdreclaim  -u admin -k table -t all -o -j
```

インデクスの使用中空きセグメントを解放する場合

```
pdreclaim  -u admin -k table -t all -j
```

4. 未使用領域の不足が解消されたかを確認します。

確認方法については，「2.5.1　RD エリアの未使用領域を確認する」を参照してください。

2.6　利用できるサービス部品の種類と適用範囲

サービスプラットフォームで提供する標準プロトコルで利用できるサービス部品の種類と適用範囲について説明します。また，各サービス部品で利用する XML スキーマの適用範囲についても説明します。

標準プロトコルで利用できるサービス部品にはインターフェース情報が定義されている必要があります。サービスプラットフォームで利用できる標準プロトコルのサービス部品の種類を次の表に示します。

表 2-10　利用できるサービス部品の種類

サービス部品の種類	通信形態	プロトコル
Web サービス	同期	SOAP（HTTP）
SessionBean		RMI-IIOP
MDB	非同期	WS-R
DB キュー		JMS

注

表に示すサービス部品以外に，標準プロトコル以外のシステムと接続してサービス部品を利用することもできます。標準プロトコル以外のシステムと接続した場合のサービス部品の呼び出しの概要については，マニュアル「サービスプラットフォーム解説」の「2.　さまざまなシステムと接続するための機能」を参照してください。

以降の項では，標準プロトコルで利用できるサービス部品ごとの適用範囲と，各サービス部品で利用する XML スキーマの適用範囲を説明します。

2.6.1　Web サービスを利用したサービス部品の適用範囲

Web サービスを利用したサービス部品の適用範囲を次に示します。

ポイント

このマニュアルで説明する WSDL および XML スキーマでは，特定のプリフィックスと名前空間 URI を対応づけて記述しています。プリフィックスと名前空間 URI の対応を次の表に示します。

表 2-11　プリフィックスと名前空間 URI の対応

プリフィックス	名前空間 URI
wsdl	http://schemas.xmlsoap.org/wsdl/
xsd	http://www.w3.org/2001/XMLSchema
soap	http://schemas.xmlsoap.org/wsdl/soap/
soap12	http://schemas.xmlsoap.org/wsdl/soap12/
wsi	http://ws-i.org/profiles/basic/1.1/xsd

(1) 前提となる仕様

Web サービスを利用したサービス部品の前提仕様について，SOAP1.1 モードの場合および SOAP1.1/1.2 併用モードの場合に分けて説明します。

- **SOAP1.1 モードの場合**

  SOAP1.1 モードの場合，Web サービスを利用したサービス部品は，次のバージョンの仕様に従って作成されている必要があります。

  - SOAP1.1
  - WSDL1.1
  - SAAJ1.2

  また，サービスプラットフォームでは，WS-I Basic Profile 1.0a に従ってサービス部品を作成することを推奨します。

- **SOAP1.1/1.2 併用モードの場合**

  SOAP1.1/1.2 併用モードの場合，Web サービスを利用したサービス部品は，次のバージョンの仕様に従って作成されている必要があります。

  - SOAP1.1 または SOAP1.2
  - WSDL1.1
  - SAAJ1.3

  また，サービスプラットフォームでは，WS-I Basic Profile 1.1 に従ってサービス部品を作成することを推奨します。

(2) SOAP メッセージの形式

SOAP メッセージは，SOAP ヘッダと SOAP ボディで構成されます。SOAP ヘッダにはメッセージの付加的な情報（識別情報など）が含まれ，SOAP ボディにはメッセージ本体が含まれています。

Web サービスを利用したサービス部品では，SOAP メッセージの形式として次の条件を満たしている必要があります。

- 要求電文および応答電文が SOAP メッセージの本文に格納されていること。

なお，08-53 よりも前のバージョンのサービスプラットフォームでは，soap:header 要素を定義した WSDL を基に SOAP アダプタを作成できますが，soap:header 要素は無視されます。

(3) WSDL の定義時の注意（SOAP1.1 モード）

SOAP1.1 モードを使用する場合の WSDL の記述形式，および WSDL 定義時の注意事項について説明します。WSDL1.1 仕様のサポート範囲については，「2.6.1(5) WSDL1.1 仕様のサポート範囲」を参照してください。

(a) wsdl:types 要素

wsdl:types 要素は，SOAP メッセージで使用する型に関する情報を定義する要素です。wsdl:types 要素は，次に示す規則に従って記述してください。

- wsdl:definitions 要素の子要素として記述してください。
- 記述できる個数は 0 個または 1 個です。2 個以上は記述できません。
- wsdl:documentation 要素および wsdl:import 要素よりもあとに記述してください。
- wsdl:documentation 要素および wsdl:import 要素を除く，ほかのすべての要素よりも前に記述してください。

(b) xsd:schema 要素

xsd:schema 要素は，XML スキーマを記述する要素です。xsd:schema 要素は，wsdl:types 要素の子要素として記述してください。XML スキーマを記述するときの規則については，「2.6.5 XML スキーマの適用範囲」を参照してください。

(c) wsdl:import 要素

wsdl:import 要素は，WSDL をインポートする場合に定義する要素です。wsdl:definitions 要素の子要素として記述してください。wsdl:import 要素は，次に示す規則に従って記述してください。

- wsdl:documentation 要素よりもあとに記述してください。
- wsdl:documentation 要素を除く，ほかのすべての要素よりも前に記述してください。
- namespace 属性は必ず記述してください。
- 値にパーセントエンコーディングした値を設定する場合，16 進数値を大文字で設定してください。例を次に示します。

  正：location="%E3%81%82.wsdl"

  誤：location="%e3%81%82.wsdl"
- wsdl:import 要素の location 属性を指定する場合の注意を次に示します。
  - RFC2396 で規定される文字および xsd:anyURI を満たす文字を使用してください。ただし，RFC2732（IPv6）は使用できません。
  - wsdl:import 要素の location 属性を絶対 URI で指定する場合は，http，https，および file のプロトコルのどれかを使用してください。

  location 属性の指定例を次に示します。

  （例 1）ローカルにある WSDL を相対パスで指定

  ./example/sample.wsdl

  （例 2）ローカルにある WSDL を URL 形式の絶対パスで指定

  file:///C:/example/sample.wsdl

（例 3）リモートにある WSDL を URL で指定

http://example.com/sample.wsdl

(d) soap:binding 要素

soap:binding 要素は，SOAP バインディングを定義する要素です。soap:binding 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。
- transport 属性には HTTP バインディングを示す"http://schemas.xmlsoap.org/soap/http"を指定してください。
- バインディング宣言は SOAP バインディングを宣言してください。SOAP バインディング以外は宣言できません。

(e) wsdl:operation 要素

wsdl:operation 要素は，次に示す規則に従って記述してください。

- wsdl:portType 要素の子要素である wsdl:operation 要素と，wsdl:binding 要素の子要素である wsdl:operation 要素は，1 対 1 になるように記述してください。
- name 属性は 255 バイト以下で記述してください。

(f) soap:operation 要素

soap:operation 要素は，SOAP バインディングでのオペレーションの情報を定義する要素です。
soap:operation 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素である wsdl:operation 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。
- style 属性が「rpc」の場合，soap:body 要素の namespace 属性に記述された名前空間に「オペレーション名」，または「オペレーション名」＋「Response」を name 属性として持つ element 要素を指定しないでください。

(g) soap:body 要素

soap:body 要素は，SOAP メッセージの soap:body 要素以下のメッセージを定義する要素です。
soap:body 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の孫要素である wsdl:input 要素または wsdl:output 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。
- 子要素は記述できません。

なお，style 属性が「document」の場合，parts 属性を記述することはできますが，無視されます。

(h) wsdl:fault 要素

wsdl:fault 要素はフォルトを定義する要素です。wsdl:fault 要素は，次に示す規則に従って記述してください。

- wsdl:portType 要素および wsdl:binding 要素の子要素として記述してください。
- 同じ name 属性の値を持つ wsdl:fault 要素を複数記述しないでください。
- wsdl:portType 要素の子要素である wsdl:operation 要素で定義した wsdl:fault 要素と対応するよう，wsdl:binding 要素の子要素である wsdl:operation 要素に wsdl:fault 要素を定義している必要があります。

(i) soap:fault 要素

soap:fault 要素は，SOAP メッセージの soap:fault 要素に含まれる detail 子要素以下のメッセージを定義する要素です。soap:fault 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素である wsdl:fault 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。

フォルトを使用する場合の条件

サービス部品から返された SOAP Fault では，次の条件をすべて満たす場合にフォルト名が faultCode に設定され，フォルトとして扱われます。

- サービス部品側に SOAP 通信基盤（Web Services）を使用していること。
- WSDL フォルト名（wsdl:fault 要素の name 属性値）と，schema（wsdl:types 内）に定義された例外型 complexType を指す element 名※が，名前空間を含めて一致していること。

これらの条件を満たしていない WSDL を利用する場合で，サービス部品から返された SOAP Fault をフォルトとして扱いたいときは，SOAP Fault 動作定義ファイルを定義してユーザ定義例外となるように設定します。

注※

element 名とは，「wsdl:fault が参照する wsdl:part 要素が，element 属性を用いて指定する要素」のことです。WSDL を次の順にたどって特定できる要素を指します。

1.wsdl:fault

2.wsdl:message（この要素をたどらない場合もあります）

3.wsdl:part

4.xsd:element

なお，サービスプラットフォームの開発支援コマンドである Java2WSDL コマンドを使用して，オペレーションの style 属性が「rpc」の WSDL ファイルを生成した場合，element 属性ではなく，type 属性で参照する WSDL ファイルが生成されます。

そのため，生成された WSDL ファイルをそのまま使用しないで，修正および再生成してから使用します。WSDL ファイルの修正および再生成の方法については，「4.3.2　サービス部品電文を作成する（Web サービスの場合）」の注意事項を参照してください。

サービスプラットフォームでフォルトを使用する場合，document スタイルの WSDL を使用することを推奨します。

(j)　soap:header 要素

soap:header 要素は，SOAP メッセージの soap:header 要素以下のメッセージを定義する要素です。soap:header 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の孫要素となる wsdl:input 要素および wsdl:output 要素の子要素として記述してください。
- 子要素は記述できません。
- message 属性には，wsdl:input 要素または wsdl:output 要素から参照される wsdl:message 要素とは異なる wsdl:message 要素を指定してください。
- part 属性には，message 属性に指定した wsdl:message 要素の子要素である wsdl:part 要素を指定してください。

(k)　wsdl:service 要素

wsdl:service 要素は，SOAP サービスを定義する要素です。1 個の wsdl:service 要素に wsdl:port 要素は必ず 1 個記述してください。2 個以上は記述できません。

(l)　添付ファイルを使用するための WSDL 定義

添付ファイルを使用するための WSDL の定義は記述できません。

(4)　WSDL の定義時と使用時の注意（SOAP1.1/1.2 併用モード）

SOAP1.1/1.2 併用モードを使用する場合，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「4.3.1　WSDL ファイルを作成する」を参考にして WSDL を記述してください。

ただし，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「4.3.1　WSDL ファイルを作成する」に記載されている内容と，サービスプラットフォームの動作が異なる点があります。SOAP1.1/1.2 併用モードを使用する場合の WSDL 定義時の注意事項を示します。

(a)　使用できる WSDL 定義のスタイル

WSDL 定義のスタイルは，document/literal で定義してください。

(b)　wsdl:port 要素

wsdl:port 要素の name 属性が重複している場合エラーが発生します。すべての WSDL で name 属性を一意にしてください。

(c) wsdl:types 要素

wsdl:types 要素の子要素としてスキーマを定義する場合，コメントの記述は xsd:annotation を利用して記述してください。

また，サービスアダプタで WSDL から作成される電文フォーマットの名前空間宣言に，JAXB の名前空間「http://java.sun.com/xml/ns/jaxb」が付加されます。

(d) wsdl:operation 要素

wsdl:operation 要素の name 属性は 255 バイト以下で記述してください。

(e) サービス部品側に SOAP 通信基盤を使用している場合の定義

サービス部品側に SOAP 通信基盤を利用している場合は，wsdl:fault 要素の name 属性と wsdl:part 要素の参照する xsd:element 要素の name 属性と一致させてください。

(f) 添付ファイルを使用するための WSDL 定義

WS-I Attachments Profile - Version 1.0 で規定された wsi:swaRef 形式で記述できます。マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「28.3.1　添付ファイル使用時の WSDL の記述（wsi:swaRef 形式）」に記載されている内容に従ってください。ただし，xsd:import 要素の schemaLocation 属性にファイルを指定する場合，「http://ws-i.org/profiles/basic/1.1/swaref.xsd」で公開されているスキーマファイルと同じ内容のファイルを指定してください。

(g) WSDL の検証

- Basic Profile に基づいた検証によって発生するエラーは，サービスプラットフォームでは警告になります。Basic Profile に基づいた検証によって発生する警告メッセージ ID と，対応するエラー ID および発生条件を次の表に示します。

表 2-12　Basic Profile に基づいた検証によって発生する警告（エラー）

項番	警告メッセージ ID	対応するエラー ID	発生条件
1	KDJW51209-W	KDJW51208-E	soap:body 要素に namespace 属性を記述したときに発生します。
2	KDJW51211-W	KDJW51210-E	soap:fault 要素に namespace 属性を記述したときに発生します。
3	KDJW51213-W	KDJW51212-E	拡張要素に wsdl:required 属性を記述したときに発生します。

- 次の条件に基づく検証は，サービスプラットフォームで実行されません。
  - WSDL に定義する要素の個数に関する検証
  - Java の予約語を記述している場合の検証

(5) WSDL1.1 仕様のサポート範囲

SOAP1.1 モードを使用する場合の WSDL1.1 仕様のサポート範囲を次の表に示します。

SOAP1.1/1.2 併用モードを使用する場合の WSDL1.1 仕様のサポート範囲については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

表 2-13　WSDL1.1 仕様のサポート範囲

分類		サポート	備考
大分類	小分類		
サービス定義：WSDL ドキュメント構造	Document Naming and Linking	○	名前空間のスコープについて規定しています。
	Authoring Style（部品の import 要素による取り込み）	○	ほかのファイルの import 要素による取り込みについて規定しています。
	Language Extensibility and Binding	×	wsdl:required 属性のデプロイについて規定しています。
	Documentation	○	要素内のコメントについて規定しています。
サービス定義：wsdl:types		○	扱うデータ型について規定しています。
サービス定義：wsdl:message		○	メッセージの論理定義を規定しています。
サービス定義：ポートタイプ	One-way Operation	×	メッセージの一方向操作について規定しています。
	Request-response Operation	○	メッセージの要求/応答操作について規定しています。
	Solicit-response Operation	×	メッセージの送信要求/応答操作について規定しています。
	Notification Operation	×	メッセージの通知操作について規定しています。
	Names of Elements within an Operation	○	input および output 要素の name 属性を示します。オーバーロードはできません。WSDL 内では名称を一意にする必要があります。
	Parameter Order within an Operation	○	操作内のパラメタの順序について規定しています。parameterOrder 属性によって，パラメタのリストを指定できます。
サービス定義：wsdl:binding		○	メッセージ形式とプロトコルの詳細の定義を規定しています。
サービス定義：wsdl:port		○	サービスの物理的定義について規定しています。
サービス定義：wsdl:service		○	サービスの位置について規定しています。1 つの WSDL に複数の SOAP サービスを対応させることはできません。

分類		サポート	備考
大分類	小分類		
SOAP バインディング	soap:binding	○	SOAP 形式のバインディングを規定しています。
	soap:operation	○	SOAP メッセージ内の SOAP オペレーションへの情報について規定しています。
	soap:body	○	SOAP メッセージ内の SOAP ボディのメッセージ部分の表示方法について規定しています。parts 属性を記述できますが，無視されます。
	soap:fault	○	SOAP メッセージ内の SOAP フォルトの内容について規定しています。
	soap:header	○	SOAP メッセージ内の SOAP ヘッダ要素内の内容について規定しています。
	soap:headerfault	×	
	soap:address	○	port 要素のアドレスについて規定しています。
MIME バインディング	mime:content	×	MIME タイプを規定しています。
	mime:multipartRelated	×	MIME パートの任意のセットを集約しています。
	mime:part	×	個々の MIME パートを規定しています。
	mime:mimeXml	×	特定のスキーマを持っている XML ペイロードを規定しています。SOAP には準拠していません。

（凡例）

○：サポートしています。

×：サポートしていません。

(6) ライフサイクル

サービスプラットフォームの実行環境とサービス部品の間では，セッションを維持するように設定（DeployScope に「Session」,「Application」を設定）しても，ステートフルな呼び出しはできません。サービスアダプタでは，常に「Request」での呼び出しです。

2.6.2 SessionBean を利用したサービス部品の適用範囲

SessionBean を利用したサービス部品の適用範囲を次に示します。

(1) 前提となる仕様

SessionBean を利用したサービス部品は，EJB 2.0 の仕様に従って作成されている必要があります。

(2) 利用できるサービス部品の配備先

SessionBean を利用したサービス部品として利用できるのは，サービスプラットフォーム上に配備されているサービス部品です。

(3) SessionBean の作成条件

SessionBean を利用したサービス部品は，次の条件に従って作成されている必要があります。

引数・戻り値に指定できるクラス

SessionBean を利用したサービス部品の引数・戻り値に指定するクラスは，次の条件を満たしている必要があります。

- ユーザ定義クラスが基本型クラスで構成されていること。
- 引数・戻り値に指定されるユーザ定義型のクラスが，private なフィールドとそのフィールドに外部からアクセスするための public なアクセスメソッド（setXX，getXX）を持っていること（JavaBeans 形式）。
- 公開するメソッドの引数・戻り値に interface のユーザ定義型を指定していないこと。
- 公開するメソッドの引数・戻り値に abstract クラスのユーザ定義型クラスを指定していないこと。
- 公開するメソッドの引数・戻り値に java.lang.Object 以外のクラスを継承するユーザ定義型クラスを指定していないこと。
- 公開するメソッドの引数・戻り値に，異なるパッケージのユーザ定義型クラスを指定していないこと。
- 次の表に示す Java データ型のうち，利用できる Java データ型だけを利用していること。

表 2-14　Java データ型の利用可否

Java でのデータ型	メソッドの引数・戻り値としての利用の可否		
	直接使用する場合	配列として使用する場合	ユーザ定義データ型クラスのメンバ変数として使用する場合
boolean	○	○	○
javax.xml.rpc.holders.BooleanHolder	○	×	×
byte	○	○	○
javax.xml.rpc.holders.ByteHolder	○	×	×
byte[]	○	○	○
javax.xml.rpc.holders.ByteArrayHolder	○	×	×
double	○	○	○
javax.xml.rpc.holders.DoubleHolder	○	×	×
float	○	○	○

Java でのデータ型	メソッドの引数・戻り値としての利用の可否		
	直接使用する場合	配列として使用する場合	ユーザ定義データ型クラスのメンバ変数として使用する場合
javax.xml.rpc.holders.FloatHolder	○	×	×
int	○	○	○
javax.xml.rpc.holders.IntHolder	○	×	×
long	○	○	○
javax.xml.rpc.holders.LongHolder	○	×	×
short	○	○	○
javax.xml.rpc.holders.ShortHolder	○	×	×
java.lang.Byte	○	○	○
javax.xml.rpc.holders.ByteWrapperHolder	○	×	×
java.lang.Byte[]	○	○	○
java.lang.Double	○	○	○
javax.xml.rpc.holders.DoubleWrapperHolder	○	×	×
java.lang.Float	○	○	○
javax.xml.rpc.holders.FloatWrapperHolder	○	×	×
java.lang.Integer	○	○	○
javax.xml.rpc.holders.IntegerWrapperHolder	○	×	×
java.lang.Long	○	○	○
javax.xml.rpc.holders.LongWrapperHolder	○	×	×
java.lang.Object	×	×	×
javax.xml.rpc.holders.ObjectHolder	×	×	×
java.lang.Object[]	×	×	×
java.lang.Short	○	○	○
javax.xml.rpc.holders.ShortWrapperHolder	○	×	×
java.lang.String	○	○	○
javax.xml.rpc.holders.StringHolder	○	×	×
java.math.BigDecimal	○	○	○
javax.xml.rpc.holders.BigDecimalHolder	○	×	×
java.math.BigInteger	○	○	○

Java でのデータ型	メソッドの引数・戻り値としての利用の可否		
	直接使用する場合	配列として使用する場合	ユーザ定義データ型クラスのメンバ変数として使用する場合
javax.xml.rpc.holders.BigIntegerHolder	○	×	×
java.util.Date	○	○	○
javax.xml.namespace.QName	○	○	○
javax.xml.rpc.holders.QNameHolder	○	×	×
java.lang.Boolean	○	○	○
javax.xml.rpc.holders.BooleanWrapperHolder	○	×	×
java.util.Calendar	○	○	○
javax.xml.rpc.holders.CalendarHolder	○	×	×

（凡例）

○：利用できます。

×：利用できません。

メソッドのオーバーロード

SessionBean を利用したサービス部品では，同じ名称のメソッドは指定できません。

トランザクション

SessionBean のトランザクション管理種別（transaction-type）やメソッドに割り当てるトランザクション属性（trans-attribute）は，HCSC サーバからのトランザクションを引き継がないよう設定します。SessionBean を利用したサービス部品には，トランザクションを引き継げません。

サービス部品でのトランザクション属性の指定の可否を次の表に示します。

表 2-15　サービス部品でのトランザクション属性の指定可否

サービス部品のトランザクションモード	トランザクション属性	指定の可否
BMT	−	○
CMT	Required	×
	RequiresNew	○
	Supports	×
	NotSupported	○
	Mandatory	×
	Never	×

（凡例）

○：利用できます（EJB の仕様どおりに動作します）。

×：利用できません。

－：該当しません。

メソッドの例外

メソッドの throws 節に記載できる例外は，java.rmi.RemoteException，および java.lang.RuntimeException の継承クラスだけです。

SessionBean の種別

SessionBean のサービス部品は，Stateless Session Bean として作成されている必要があります。

サービスプラットフォームの実行環境では，EJB サービスに対する実行要求ごとに接続(create)と切断(remove)を発行します。そのため，Stateful Session Bean を使用して内部的な状態を保持するようなサービス部品は利用できません。

(4) スキーマの形式

SessionBean を利用したサービス部品で利用するスキーマは，一定の条件を満たしている必要があります。スキーマの条件の詳細については，「2.6.5 XML スキーマの適用範囲」を参照してください。

2.6.3 Reliable Messaging のローカルキューを利用したサービス部品の適用範囲

Reliable Messaging のローカルキューを利用したサービス部品の適用範囲を次に示します。

(1) 前提となる仕様

Reliable Messaging のローカルキューを利用したサービス部品は，EJB 2.0 の仕様に従って作成されている必要があります。

(2) メッセージの種別

サービス部品が Reliable Messaging のローカルキューを利用している場合，サービスリクエスタから送信される電文をサービス部品に送信するためのメッセージ種別として，次のメッセージ種別を利用できます。

- BytesMessage
- ObjectMessage
- TextMessage

上記以外のメッセージ種別は利用できません。

(3) スキーマの形式

Reliable Messaging のローカルキューを利用したサービス部品で利用するスキーマは，一定の条件を満たしている必要があります。スキーマの条件の詳細については，「2.6.5 XML スキーマの適用範囲」を参照してください。

2.6.4　DB キューを利用したサービス部品の適用範囲

DB キューを利用したサービス部品の適用範囲を次に示します。

(1)　前提となる仕様

DB キューを利用したサービス部品は，次のバージョンの仕様に従って作成されている必要があります。

- TP1/Server Base Enterprise Option 02-00 以降
- EJB 2.0（サービス部品を MDB で実装した場合）

(2)　メッセージの種別

サービス部品が DB キューの場合，サービスリクエスタから送信される電文をサービス部品に送信するためのメッセージ種別として，BytesMessage を利用できます。

BytesMessage 以外のメッセージ種別は利用できません。

(3)　利用できるデータベース

DB キューを作成できるデータベースは HiRDB だけです。

(4)　スキーマの形式

DB キューを利用したサービス部品で利用するスキーマは，一定の条件を満たしている必要があります。スキーマの条件の詳細については，「2.6.5　XML スキーマの適用範囲」を参照してください。

2.6.5　XML スキーマの適用範囲

各サービス部品で利用する XML スキーマの適用範囲について説明します。

(1)　利用する XML スキーマの形式

- XML スキーマは次の条件を満たしている必要があります。
  - スキーマの XSD 名前空間が「http://www.w3.org/2001/XMLSchema」であること。
  - 選択した XML スキーマに，必ずルート要素が記述されていること。
  - XML スキーマをファイル指定する場合，ファイル名の長さが 128 バイト以下であること。
  - XML スキーマの属性（attribute）の先頭と末尾に White space 文字（半角スペース（#x20），タブ（#x9），および改行文字（#xA または#xD））を使用していないこと。
- maxOccurs 属性に大きな値を指定するとメモリを大量に消費します。maxOccurs 属性には，できる限り unbounded を指定することをお勧めします。

- xsd:element 要素の型の定義には，type 属性，ref 属性または内部定義（complexType または simpleType）が使用できます。
- ビジネスプロセスの受付として標準受付を使用する場合，xsd:schema 要素の elementFormDefault 属性には，qualified を指定することをお勧めします。

注意事項

- SOAP ヘッダや SOAP ボディの子要素に SOAP 固有の属性（mustUnderstand 属性，role 属性，encodingStyle 属性，actor 属性，および relay 属性）が出現した場合，XML スキーマにこの属性を定義しないでデータ変換アクティビティで変換すると，変換後の変数に SOAP 固有の属性が反映されません。また，検証処理実行時に検証エラーが発生します。このため，SOAP 固有の属性が出現する場合は，XML スキーマに xsd:anyAttribute を記述してください。
- XML スキーマファイル中の値にパーセントエンコーディングした値を設定する場合，16 進数値を大文字で設定してください。例を次に示します。

  正：schemaLocation="%E3%81%82.xsd"

  誤：schemaLocation="%e3%81%82.xsd"

(2) 外部の XML スキーマを参照する場合の注意事項

XML スキーマから外部の XML スキーマを参照して利用する場合の注意事項を，次に示します。

- WSDL の wsdl:types 要素以下に定義した XML スキーマの xsd:import 要素の schemaLocation 属性が指定されていない場合，wsdl:types 要素以下の XML スキーマを使用して名前空間を解決している必要があります。
- XML スキーマの xsd:import 要素の schemaLocation 属性は必ず指定してください。
- xsd:import 要素には，必ず namespace 属性を指定してください。
- xsd:import 要素および xsd:include 要素では，参照の呼び出し階層を 20 階層以内にしてください。
- schemaLocation 属性に redefine 要素があるスキーマ（2 段階以上の redefine）は指定しないでください。
- xsd:schema 要素の targetNamespace 属性指定されているスキーマからは，targetNamespace 属性が指定されていないスキーマを include しないでください。
- SOAP1.1 モードを使用する場合，xsd:import 要素の schemaLocation 属性，および xsd:include 要素の schemaLocation 属性には，UTF-8 でパーセントエンコーディングした値でも，パーセントエンコーディングしていない値でも設定できます。ただし，パーセントエンコーディングし，かつファイルパスにディレクトリを含む場合，ディレクトリデリミタ（/）をエンコードしないでください。
- SOAP1.1/1.2 併用モードを使用する場合，xsd:import 要素の schemaLocation 属性，および xsd:include 要素の schemaLocation 属性には，UTF-8 でパーセントエンコーディングした値を指定してください。ただし，パーセントエンコーディングし，かつファイルパスにディレクトリを含む場合，ディレクトリデリミタ（/）をエンコードしないでください。

- schemaLocation 属性のディレクトリデリミタには「/」を使用してください（「¥」は使用できません）。
- XML スキーマのファイルパスには，次の文字を使用しないでください。

  「;」，「?」，「:」，「@」，「&」，「=」，「+」，「$」，「,」，「<」，「>」，「#」，「%」，「"」，「{」，「}」，「|」，「^」，「[」，「]」，「`」
- schemaLocation 属性を指定する場合の注意を次に示します。
  - RFC2396 で規定されている文字および xsd:anyURI を満たす文字を使用してください。ただし，RFC2732（IPv6）は使用できません。
  - schemaLocation 属性を絶対 URI で指定する場合は，http，https，および file のプロトコルのどれかを使用してください。

  schemaLocation 属性の指定例を次に示します。

  （例 1）ローカルにある XML スキーマを相対パスで指定

  ./example/sample.xsd

  （例 2）ローカルにある XML スキーマを URL 形式の絶対パスで指定

  file:///C:/example/sample.xsd

  （例 3）リモートにある XML スキーマを URL で指定

  http://example.com/sample.xsd

2.7　コード変換機能の拡張（Code Converter 使用時）

サービスプラットフォームでは，次の方法でコード変換機能を拡張できます。

- Code Converter の機能を使用する
- 文字コード変換 UOC を使用する

2.7.1　Code Converter による拡張

Code Converter の機能を使用してコード変換機能を拡張する場合の手順を説明します。

(1)　ユーザマッピングテーブルを使用する

ユーザ固有の外字マッピングなどのユーザマッピングテーブルを，Code Converter を使用して作成し，サービスプラットフォームに組み込めます。

ユーザマッピングテーブルを作成する場合，次のファイルに記載された Code Converter 製品と同じバージョンの Development Kit が必要になります。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥codeconv_ver.txt
```

ファイルの記載例を次に示します。

- P-265Z-1344 04-02
  この場合，次の Code Converter 製品が必要です。
  Code Converter - Development Kit for Java 04-02

上記ファイルが存在しない場合，次の Code Converter 製品が必要です。

- P-265Z-2344
  Code Converter - Development Kit for Java 04-02

ユーザマッピングテーブル作成の手順は，マニュアル「Code Converter for Java」を参照してください。

実行環境に必要なユーザマッピングテーブルを組み込むための設定について説明します。

(a)　1 つの文字コードに対して，ユーザマッピングテーブルが 1 種類の場合

ユーザマッピングテーブルを組み込む手順を次に示します。

1. ＜サービスプラットフォームのインストールディレクトリ＞¥CSC¥lib¥external¥table ディレクトリをバックアップします。

2. ＜サービスプラットフォームのインストールディレクトリ＞¥CSC¥lib¥external¥table ディレクトリに，作成したユーザマッピングテーブルを上書きします。

(b)　1 つの文字コードに対して，ユーザマッピングテーブルが複数必要な場合

接続するシステムごとで外字マッピングが異なるなど，1 つの文字コードに対してユーザマッピングテーブルが複数必要な場合は，コード変換テーブルの格納パスをデフォルトから切り替えることで，バイナリフォーマット定義ファイルごとにコード変換テーブルの格納パスを設定できます。

コード変換テーブルの格納パスは，コード変換テーブル格納パスのルートパスとコード変換テーブル格納パスの格納ディレクトリ名を組み合わせて生成されます。

デフォルトから切り替えたコード変換テーブルの格納パスのディレクトリに，作成したユーザマッピングテーブルを格納します。

コード変換テーブルを切り替える手順を次に示します。

1. コード変換テーブルを格納するディレクトリのルートパスを指定します。

   HCSC サーバランタイム定義ファイルの code-table-root-path プロパティにコード変換テーブルの格納先のルートパスを指定します。code-table-root-path プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

2. コード変換テーブルの格納ディレクトリ名を設定します。

   バイナリフォーマット定義ファイルのフォーマット情報に，コード変換テーブルの格納ディレクトリ名を設定します。COBOL 登録集原文ファイルにコード変換テーブルの格納ディレクトリ名を設定し，バイナリフォーマット定義ファイルを新規作成することもできます。

   バイナリフォーマット定義ファイルの作成方法については，「4.4　電文フォーマット（バイナリフォーマット定義ファイル）の作成方法」を参照してください。

   COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの作成方法については，「4.5 COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

   注意事項

   コード変換テーブルの格納ディレクトリ名を設定しなかった場合は，csc.dt.tablepath に指定したコード変換テーブルが使用されます。

(2)　単体版の Code Converter に入れ替える

サービスプラットフォームにインストール済みの Code Converter を，単体版の Code Converter に入れ替えることで文字コード変換機能を拡張できます。

以降で，単体版の Code Converter へ入れ替える手順を説明します。

(a)　JAR の組み込み

JAR の組み込み手順を，次に示します。

1. ＜サービスプラットフォームのインストールディレクトリ＞¥CSC¥lib¥external¥codeconv.jar をバックアップします。

2. インストールした単体版の Code Converter の codeconv.jar をコピーします。
   コピー先を次に示します。

```
＜サービスプラットフォームのインストールディレクトリ＞¥CSC¥lib¥external¥codeconv.jar
```

(b)　コード変換テーブルの格納パスの設定

コード変換テーブルの格納パスを設定するには，＜サービスプラットフォームのインストールディレクトリ＞¥CC¥server¥usrconf¥ejb¥＜ J2EE サーバ名＞¥usrconf.properties に，次に示すパスを追加してください。

```
csc.dt.codetablepath=＜Code Converterのインストールディレクトリ＞/table
```

参考

この設定は Smart Composer 機能または運用管理ポータルで実施することを推奨します。Smart Composer 機能および運用管理ポータルについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「1.1　運用管理機能のツールとは」を参照してください。

単体版の Code Converter インストール直後の table ディレクトリには，サービスプラットフォームが実行するコード変換に必要な変換テーブルファイル（guniuni.tbl）が含まれていません。そのため，コード変換テーブルのパスを設定する場合は，table ディレクトリに次に示すファイルを必ずコピーしてください。

```
＜サービスプラットフォームのインストールディレクトリ＞¥CSC¥lib¥external¥table¥guniuni.tbl
```

「(1)(a)　1 つの文字コードに対して，ユーザマッピングテーブルが 1 種類の場合」で示したユーザマッピングテーブルを利用する場合は，table ディレクトリを＜ Code Converter のインストールディレクトリ＞/table に読み替えてください。

Code Converter のコードマッピングユティリティ，および変換テーブルファイル（guniuni.tbl）の詳細は，マニュアル「Code Converter for Java」を参照してください。

2.7.2　文字コード変換 UOC による拡張

サービスプラットフォームで対応していない文字コードを利用する場合，その文字コードを処理する文字コード変換 UOC を作成することで，対応していない文字コードを扱うことができます。文字コード変換 UOC の詳細は，「付録 H　文字コード変換 UOC を利用した文字コード変換」を参照してください。

文字コード変換 UOC 内で Code Converter が提供する API を使用する場合は，次に示す製品のライセンスが必要となります。

- Code Converter - Development Kit
  文字コード変換 UOC を開発する環境を利用する場合
- Code Converter - Server Runtime（Windows の場合）
  文字コード変換 UOC を実行する環境を利用する場合
- Code Converter - Runtime（UNIX の場合）
  文字コード変換 UOC を実行する環境を利用する場合

Code Converter の入れ替えが必要な場合は，「2.7.1(2)　単体版の Code Converter に入れ替える」の手順を参照してください。

3 プロジェクトの管理とリポジトリの管理

この章では，HCSCTE を組み込んだ Eclipse での，HCSCTE のプロジェクトの作成・削除など，プロジェクトの管理について説明します。また，プロジェクトのプロパティの設定で指定したリポジトリの管理についても説明します。

3.1　プロジェクトの管理

HCSC コンポーネントを開発する前に，HCSCTE プロジェクトを作成して，プロパティを設定します。また，作成したプロジェクトは，必要に応じて，エクスポートおよびインポートします。

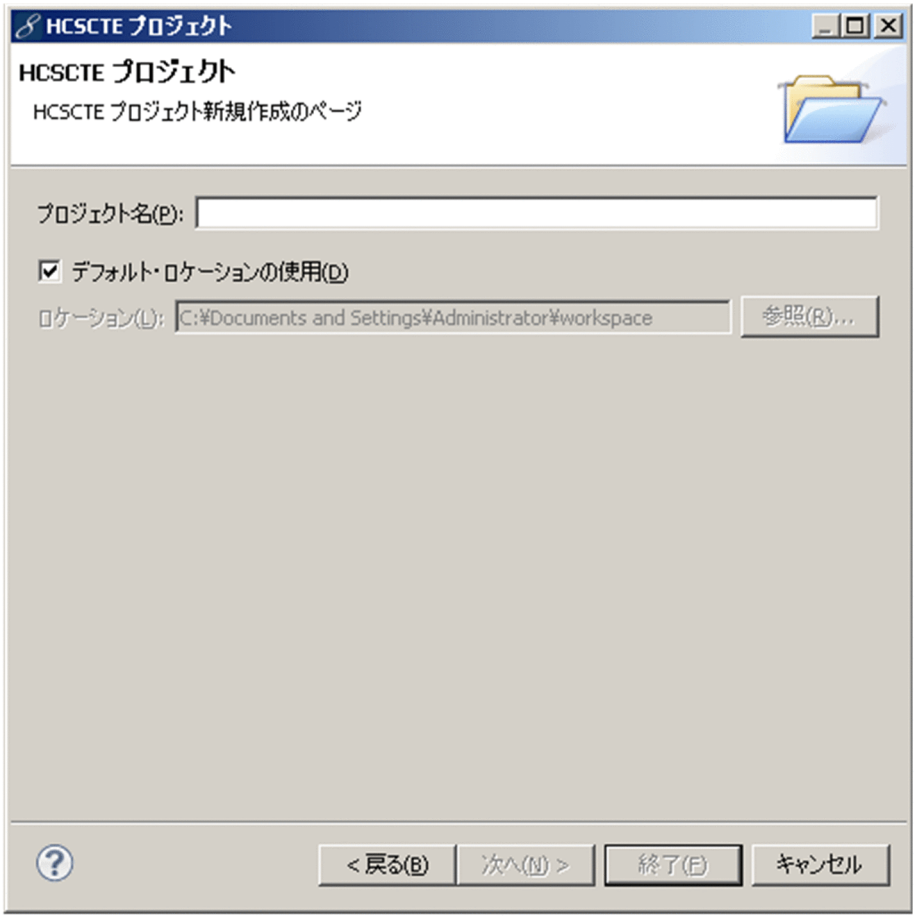
3.1.1　プロジェクトを作成する

HCSCTE プロジェクトを作成する前に，Eclipse のメニューから［ウィンドウ］－［設定］－［Java］－［ビルド・パス］を選択し，「ソースおよび出力フォルダー」に「プロジェクト」を設定してください。

HCSCTE プロジェクトの作成時に，サービスやユーザ定義受付で使用する ID（サービス ID および受付 ID）に任意のプリフィックスを設定できます。なお，プリフィックスは，HCSCTE プロジェクト作成後に設定することもできます。

HCSCTE プロジェクトの作成手順を次に示します。

1. Eclipse を起動します。
2. メニューから［ファイル］－［新規］－［プロジェクト］を選択します。
   ［新規プロジェクト］ダイアログが表示されます。
3. [HCSCTE プロジェクト］を選択して，［次へ］ボタンをクリックします。
   ［HCSCTE プロジェクト］ダイアログ（HCSCTE プロジェクト新規作成のページ）が表示されます。



4. 次の項目を設定して，［次へ］ボタンをクリックします。

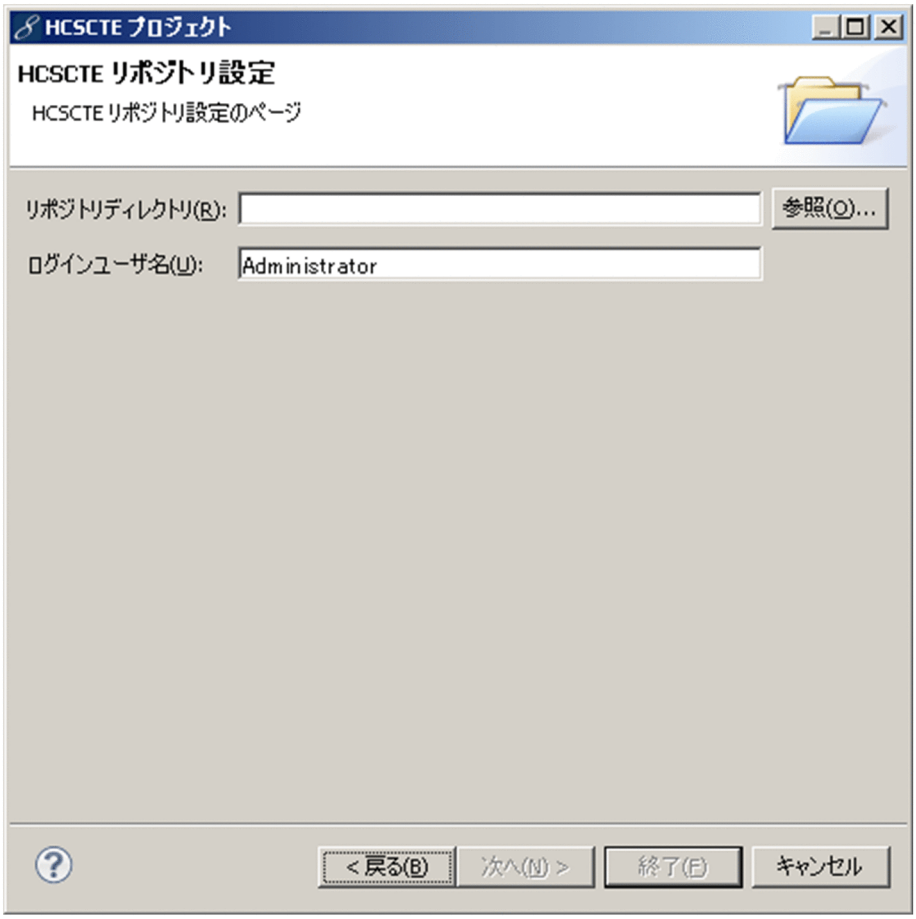
プロジェクト名

任意の名称を指定します。

デフォルト・ロケーションの使用

［デフォルト・ロケーションの使用］チェックボックスにチェックします。

［HCSCTE プロジェクト］ダイアログ（HCSCTE リポジトリ設定のページ）が表示されます。



5. 次の項目を設定します。

リポジトリディレクトリ

リポジトリ情報を格納する任意のディレクトリを指定します。リポジトリディレクトリを指定する場合は，次の点に注意してください。

・リポジトリディレクトリのパスとプロジェクトのパスには，同じパスを指定しないでください。

・パスは，絶対パスで指定してください。

・パスの長さは，正規化された絶対パスでチェックされます。

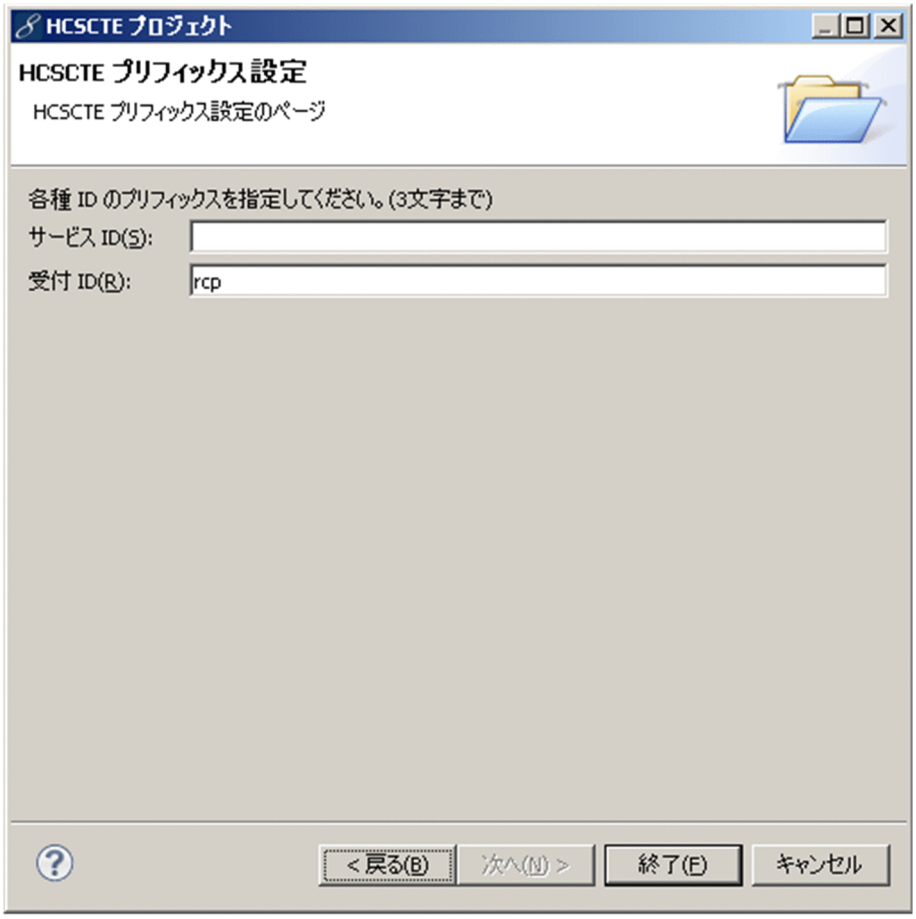
・HCSCTE プロジェクトの作成前に設定した SOAP モードと異なる SOAP モードの HCSCTE プロジェクトで設定したリポジトリディレクトリは，使用できません。

ログインユーザ名

リポジトリログインに使用するユーザ名を指定します。ユーザ名に使用できる文字は半角英数字だけで，長さは 1～16 文字になります。

6. サービス ID および受付 ID にプリフィックスを設定する場合は，［次へ］ボタンをクリックします。プリフィックスを設定しない場合は，手順 8.に進んでください。

［HCSCTE プロジェクト］ダイアログ（HCSCTE プリフィックス設定のページ）が表示されます。

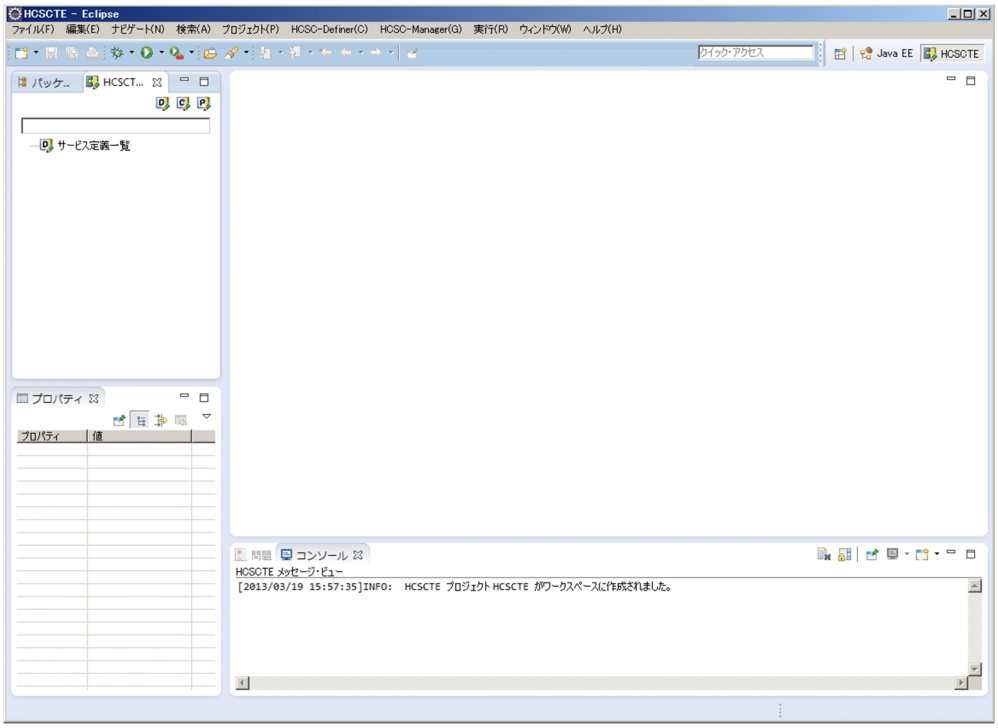


7. サービス ID および受付 ID のプリフィックスを指定します。

   サービス ID および受付 ID のプリフィックスは，3 文字までの半角英数字およびアンダーバー（_）で指定してください。

8. [終了] ボタンをクリックします。

   HCSCTE のプロジェクトが作成され，作成した HCSCTE のプロジェクトのパースペクティブが起動します。



注意事項

HCSCTE プロジェクトは，ワークスペースに 1 つしか作成できません。

3.1.2　プロパティを設定する

ここでは，HCSCTE のプロジェクトを作成したあとに実施するプロパティの設定について説明します。

(1)　リポジトリの設定

HCSCTE プロジェクト作成時と異なるリポジトリを使用する場合，必要に応じてリポジトリとして利用するディレクトリを指定します。

リポジトリの設定については，「3.2.1　リポジトリを設定する」を参照してください。

(2)　検証設定

検証設定では，データベースで使用する文字コードを指定します。

注意事項

HCSCTE プロジェクトで設定するデータベースの文字コードの変更は，HCSC コンポーネント※を作成する前に実施してください。HCSC コンポーネント※の作成後には変更しないでください。

注※

サービスアダプタ，ビジネスプロセス，およびユーザ定義受付を指します。

検証設定で文字コードを指定する手順を次に示します。

1. Eclipse のメニューから，［プロジェクト］－［プロパティ］を選択します。

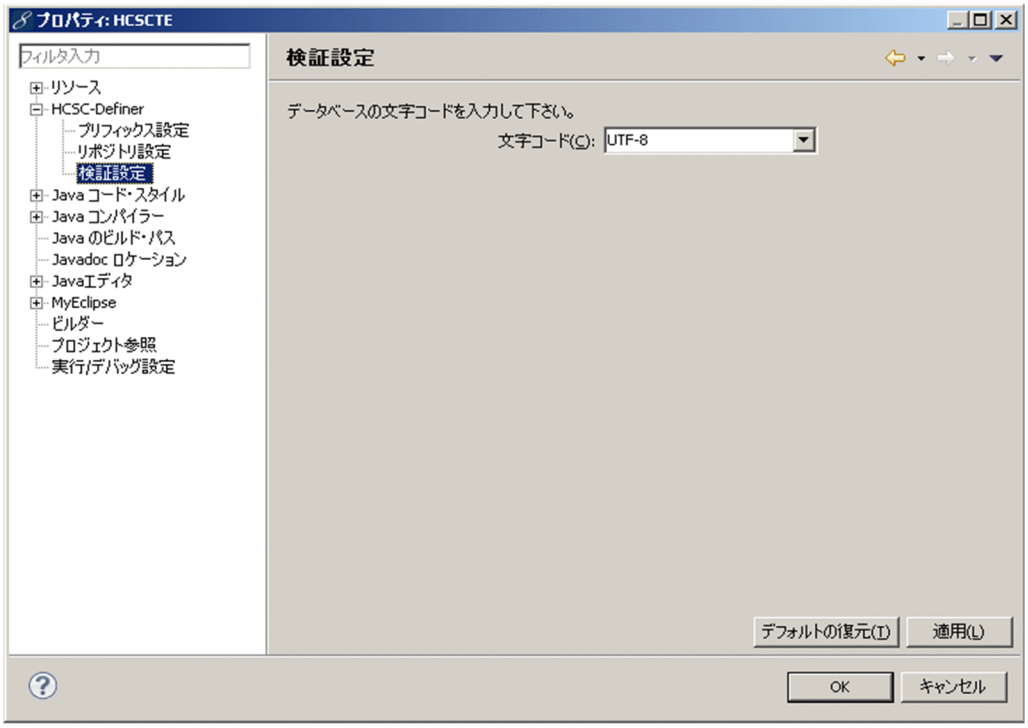
［プロパティ：HCSCTE※］ダイアログが表示されます。

注※

「HCSCTE」の部分には，プロジェクト名が表示されます。

2. ダイアログ左側のツリービューから，［HCSC-Definer］－［検証設定］を選択します。

ダイアログ右側に，検証設定に関する設定項目が表示されます。



3. ［文字コード］に，データベースで使用する文字コードを指定します。

ここで指定した値を使用して，サービス名やビジネスプロセスのアクティビティ名などの文字列長の検証が行われます。

4. ［適用］ボタン，［OK］ボタンの順にクリックします。

(3)　サービス ID および受付 ID に付加するプリフィックスの設定

サービス ID および受付 ID に，任意のプリフィックスを指定する場合の手順を次に示します。

1. Eclipse のメニューから，［プロジェクト］－［プロパティ］を選択します。

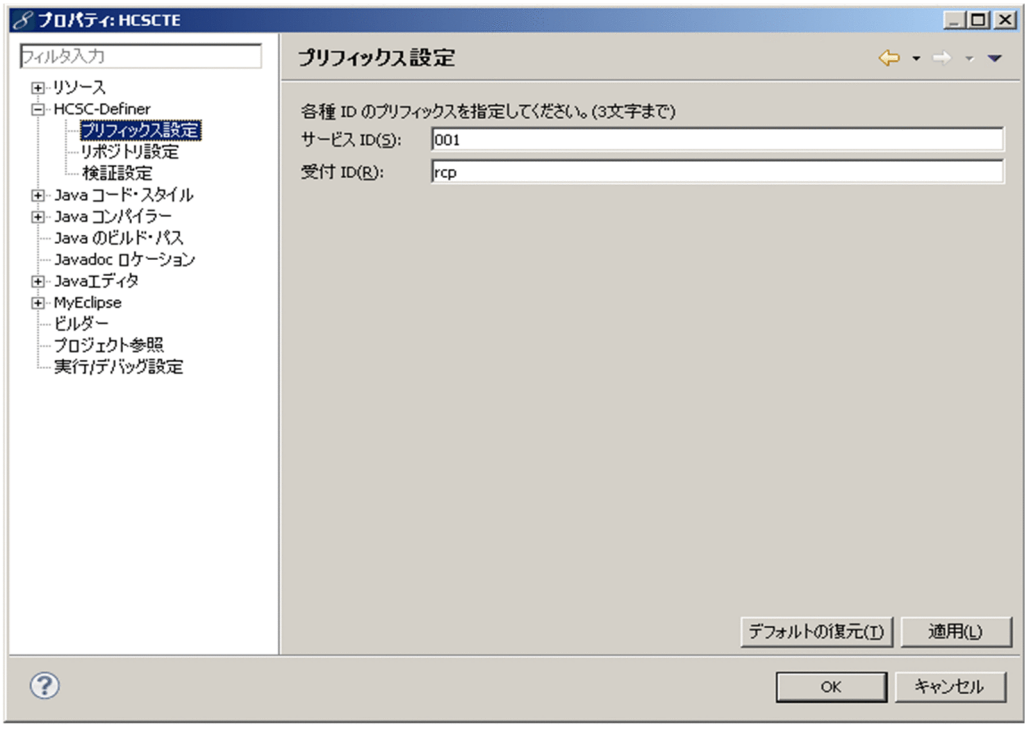
［プロパティ：HCSCTE※］ダイアログが表示されます。

注※

「HCSCTE」の部分には，プロジェクト名が表示されます。

2. ダイアログ左側のツリービューから，［HCSC-Definer］－［プリフィックス設定］を選択します。

ダイアログ右側に，プリフィックス設定に関する設定項目が表示されます。



3. ［サービス ID］および［受付 ID］に，付加する ID を指定します。

サービス ID および受付 ID のプリフィックスは，3 文字までの半角英数字およびアンダーバー（_）で指定してください。

4. ［適用］ボタン，［OK］ボタンの順にクリックします。

3.1.3　プロジェクトをエクスポート/インポートする

Eclipse の標準機能を使用して，HCSCTE プロジェクトをエクスポートできます。また，エクスポートした HCSCTE プロジェクトを，インポートすることもできます。ただし，1 つのワークスペースに，同時に 2 つ以上の HCSCTE プロジェクトを作成することはできません。

エクスポート方法およびインポート方法の詳細については，Eclipse のドキュメントを参照してください。

3.1.4　プロジェクトを削除する

HCSCTE プロジェクトを削除する手順を次に示します。

1. パッケージ・エクスプローラーから削除するプロジェクトを選択します。

2. メニューから［編集］－［削除］を選択します。
   選択したプロジェクトが削除されます。

3.1.5　SOAP モードを変更する

HCSCTE プロジェクトを作成したあとに SOAP モードを変更するには，HCSCTE プロジェクトを作成し直す必要があります。

SOAP モードを変更する手順を次に示します。

1. 作成した HCSCTE プロジェクトを削除します。
   HCSCTE プロジェクトの削除については，「3.1.4　プロジェクトを削除する」を参照してください。

2. Eclipse を終了します。

3. SOAP モードを設定します。
   SOAP モードの設定方法については，「2.2.4　SOAP モードの選択」を参照してください。

4. Eclipse を起動します。

5. HCSCTE プロジェクトを作成します。
   HCSCTE プロジェクトの作成については，「3.1.1　プロジェクトを作成する」を参照してください。

3.1.6　開発時の注意事項

HCSCTE プロジェクトの lib ディレクトリまたは src ディレクトリ下に，ファイルシステム内のファイルのリンクを登録しないでください。リンクで登録されたファイルは，サービスのパッケージング時に EAR ファイルに格納されません。

3.2　リポジトリの管理

リポジトリとは，開発環境と運用環境で共有する情報を格納するディレクトリのことをいいます。開発環境では，HCSCTE プロジェクトの作成時にリポジトリを設定します。また，設定したリポジトリは，必要に応じて，エクスポートおよびインポートします。

注意事項

バージョン 07-60 以降では，開発環境と運用環境で同じリポジトリを共有できません。旧バージョンでリポジトリを共有していた場合，開発環境と運用環境で別々のリポジトリを用意して，移行してください。移行手順については，「付録 A.3　旧バージョンで開発環境と運用環境のリポジトリを共有していた場合の移行手順」を参照してください。

3.2.1　リポジトリを設定する

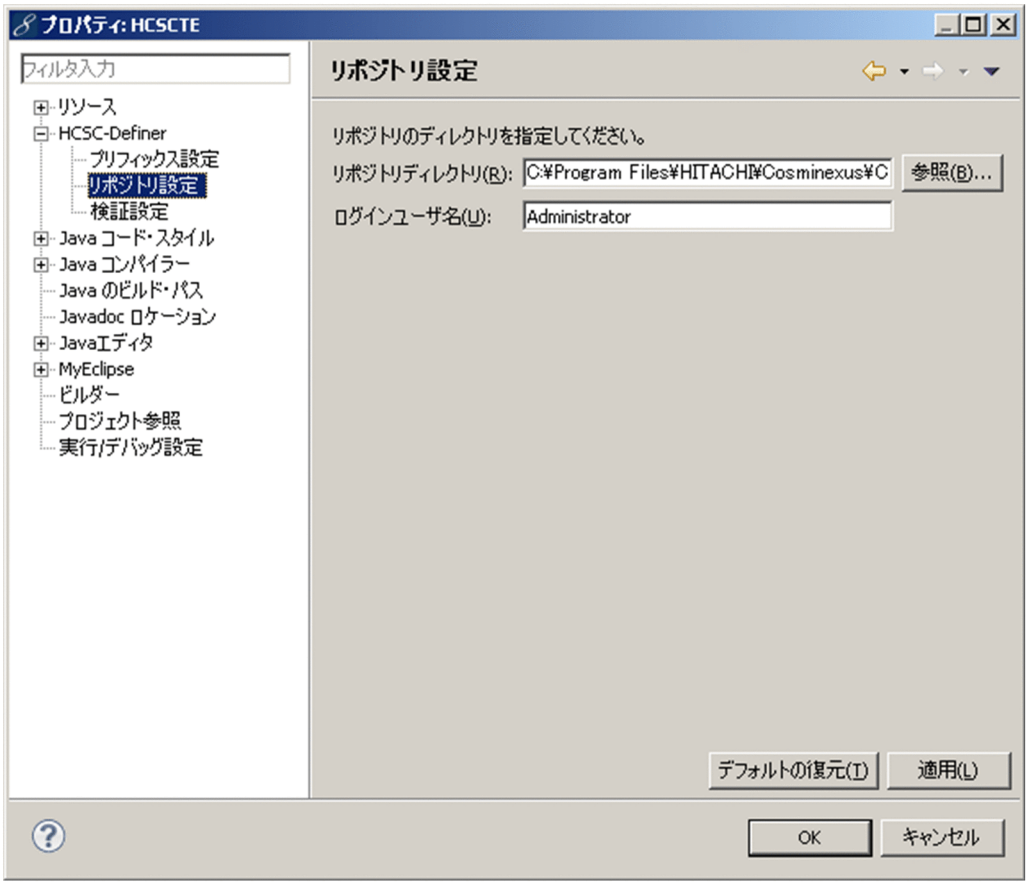
ここでは，HCSCTE プロジェクトの作成時に設定したリポジトリを変更する方法について説明します。

(1)　リポジトリの指定

リポジトリとするディレクトリを指定します。

リポジトリを指定する手順を次に示します。

1. Eclipse のメニューから，［プロジェクト］－［プロパティ］を選択します。
   ［HCSCTE※のプロパティ］ダイアログが表示されます。
   注※
   「HCSCTE」の部分には，プロジェクト名が表示されます。
2. ダイアログ左側のツリービューから，［HCSC-Definer］－［リポジトリ設定］を選択します。
   ダイアログ右側に，リポジトリの設定項目が表示されます。



3. [リポジトリディレクトリ] に，リポジトリとする任意のディレクトリを指定します。

   [リポジトリディレクトリ] を指定する場合は，次の点に注意してください。

   - すでに存在するディレクトリを指定してください。
   - リポジトリディレクトリのパスとプロジェクトのパスには，同じパスを指定しないでください。
   - パスは，絶対パスで指定してください。
   - パスの長さは，正規化された絶対パスでチェックされます。
   - HCSCTE プロジェクトの作成前に設定した SOAP モードと異なる SOAP モードの HCSCTE プロジェクトで設定したリポジトリディレクトリは，使用できません。

4. [ログインユーザ名] に，ログインに使用する任意のユーザ名を入力します。

   ユーザ名に使用できる文字は半角英数字だけで，長さは 1～16 文字になります。

5. [適用] ボタン，[OK] ボタンの順にクリックします。

(2) リポジトリの初期化

リポジトリの初期化を行うと，リポジトリの内容をすべて消去し，リポジトリを初期状態に戻すことができます。

HCSCTE プロジェクトの lib ディレクトリや src ディレクトリにアクセスしないようにしてください。エラーが発生した場合，次の点を確認してください。

- HCSCTE プロジェクトの src ディレクトリ以下のファイルを開いたエディタ（外部エディタを含む）を閉じているか
- HCSCTE プロジェクトの lib ディレクトリ以下のライブラリを利用したプログラムを実行していないか

リポジトリを初期化する手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［定義情報管理］－［定義情報初期化］を選択します。
   メッセージダイアログが表示されます。

2. ［はい］ボタンをクリックします。
   リポジトリが初期化されます。

(3) リポジトリの変更

リポジトリとするディレクトリを変更する場合の手順は，「(1) リポジトリの指定」と同じ手順になります。リポジトリを変更する場合，サービスアダプタ定義やビジネスプロセス定義など，リポジトリの変更以外の作業は終了させてください。

［デフォルトの復元］ボタンをクリックした場合，前回指定したリポジトリが設定されます。

(4) リポジトリの注意事項

リポジトリのファイルおよびディレクトリは，直接操作しないでください。ファイルおよびディレクトリの名前の変更，削除，内容の変更などを行った場合，動作の保証はできません。

3.2.2 リポジトリをエクスポートする

リポジトリのエクスポートでは，リポジトリを 1 つの ZIP ファイルにして，指定したディレクトリに保存することができます。

リポジトリのエクスポート方法には，次の 2 つがあります。

- リポジトリ中のすべての情報を ZIP ファイルにして保存する
- 必要なサービスの情報だけを選択し，ZIP ファイルにして保存する

(1) リポジトリをエクスポートする

リポジトリをエクスポートする手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［定義情報管理］－［全定義情報エクスポート］を選択します。
   ［リポジトリエクスポート］ダイアログが表示されます。

2. 保存するディレクトリを指定し，保存する ZIP ファイルのファイル名を入力します。

3. ［保存］ボタンをクリックします。

指定したディレクトリに ZIP ファイルが保存されます。

保存先のディレクトリに同じファイル名称のファイルがある場合，上書きを確認するダイアログが表示されます。上書きしてよいときは，［はい］ボタンをクリックして保存します。

参考

必要に応じてリポジトリをエクスポートし，リポジトリのデータをバックアップすることをお勧めします。

(2) サービス定義の一部だけをエクスポートする

必要なサービスを選択してエクスポートする場合，次に示す情報を指定したディレクトリに保存することができます。

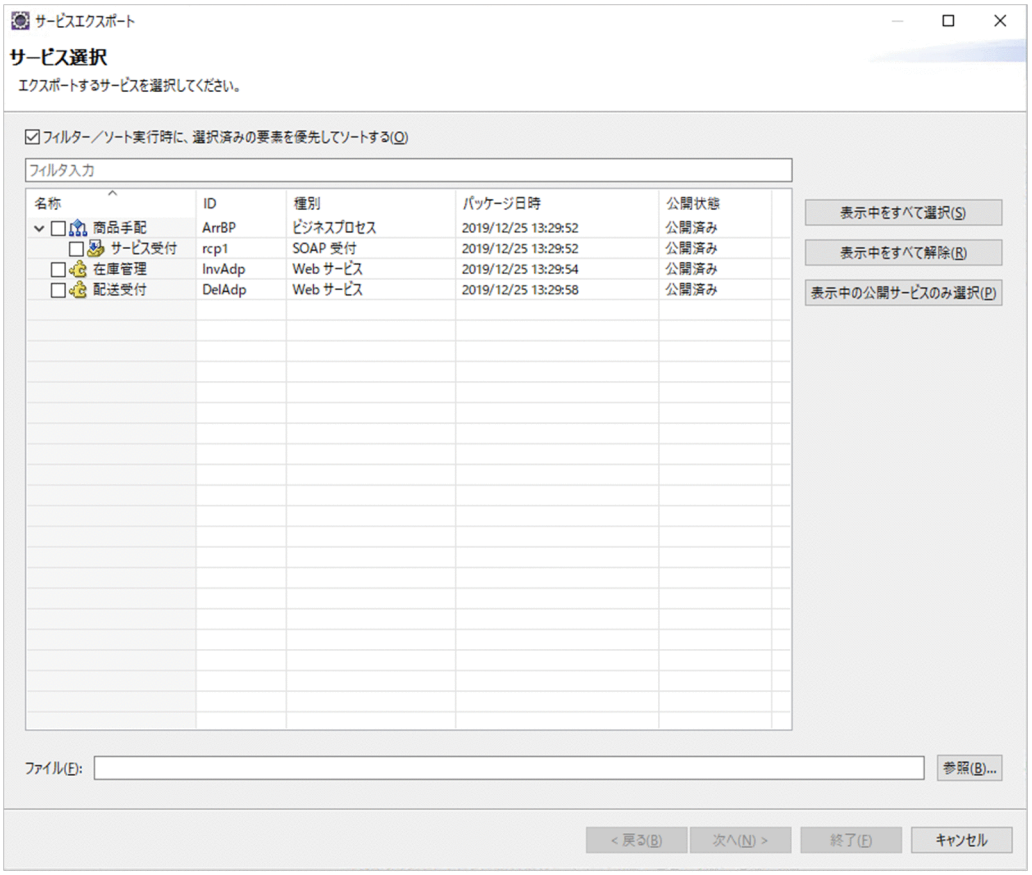
- HCSC コンポーネント（サービスアダプタ，ビジネスプロセス）に関する定義
- ユーザ定義 Java クラスのソースファイル
- ユーザ定義 Java クラスの前提ライブラリファイル

エクスポートの対象は複数選択できます。なお，エクスポート対象のビジネスプロセスに関連するユーザ定義受付も同時にエクスポートされます。

サービス定義の一部だけをエクスポートする手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［定義情報管理］－［サービスエクスポート］を選択します。

サービスエクスポートウィザードのサービス選択ページが表示されます。



2. エクスポートするサービスのチェックボックスにチェックします。

   公開済みのサービスだけをチェックする場合は［表示中の公開サービスのみ選択］をクリックします。

3. ［参照］ボタンをクリックして保存するディレクトリを指定し，保存する ZIP ファイルのファイル名を入力します。

   ユーザ定義 Java クラスの有無がチェックされ，［次へ］ボタンまたは［終了］ボタンのどちらかが活性化します。

4. 次のどれかの操作をします。

   ユーザ定義 Java クラスのソースファイルと前提ライブラリがともにない場合

   ［終了］ボタンをクリックします。

   選択したサービス定義が ZIP ファイルに保存されます。

   ユーザ定義 Java クラスの前提ライブラリがあって，ソースファイルがない場合

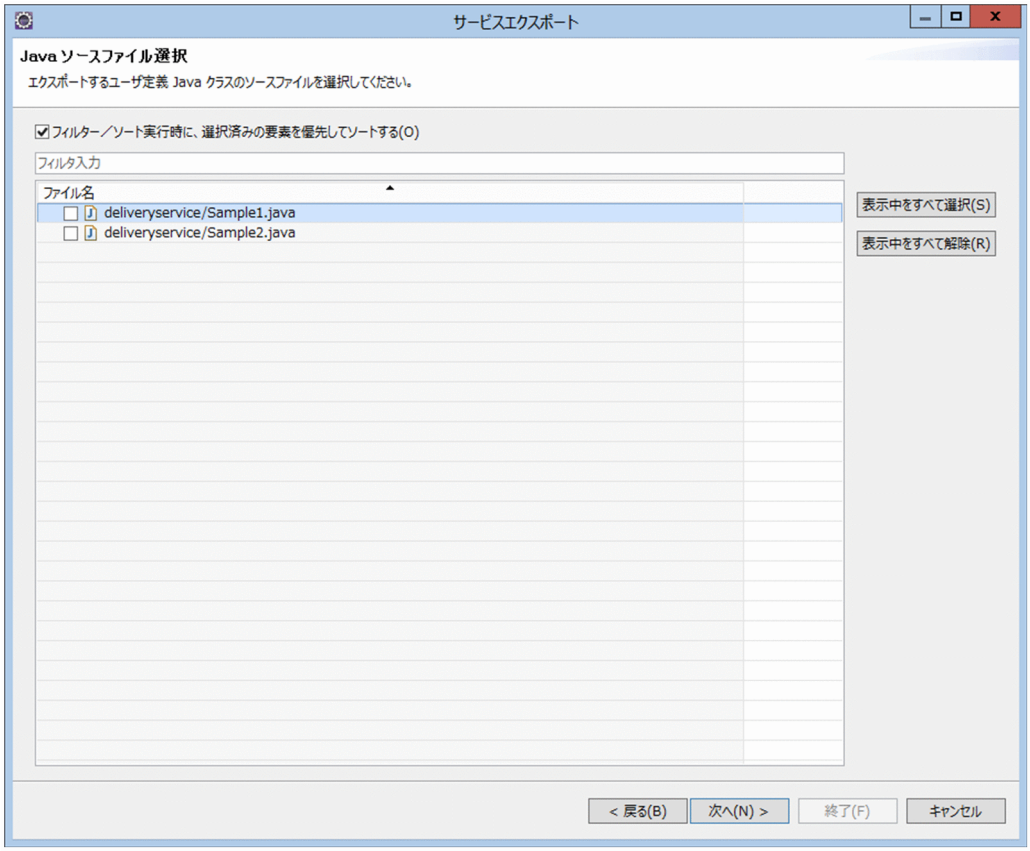
   ［次へ］ボタンをクリックします。

   サービスエクスポートウィザードの Java ライブラリファイル選択ページが表示されます。手順 7. に進んでください。

   ユーザ定義 Java クラスのソースファイルがある場合

   ［次へ］ボタンをクリックします。

   サービスエクスポートウィザードの Java ソースファイル選択ページが表示されます。次の手順に進んでください。



5. エクスポートするユーザ定義 Java クラスのチェックボックスにチェックします。

6. 次のどちらかの操作をします。

ユーザ定義 Java クラスの前提ライブラリがない場合

［終了］ボタンをクリックします。

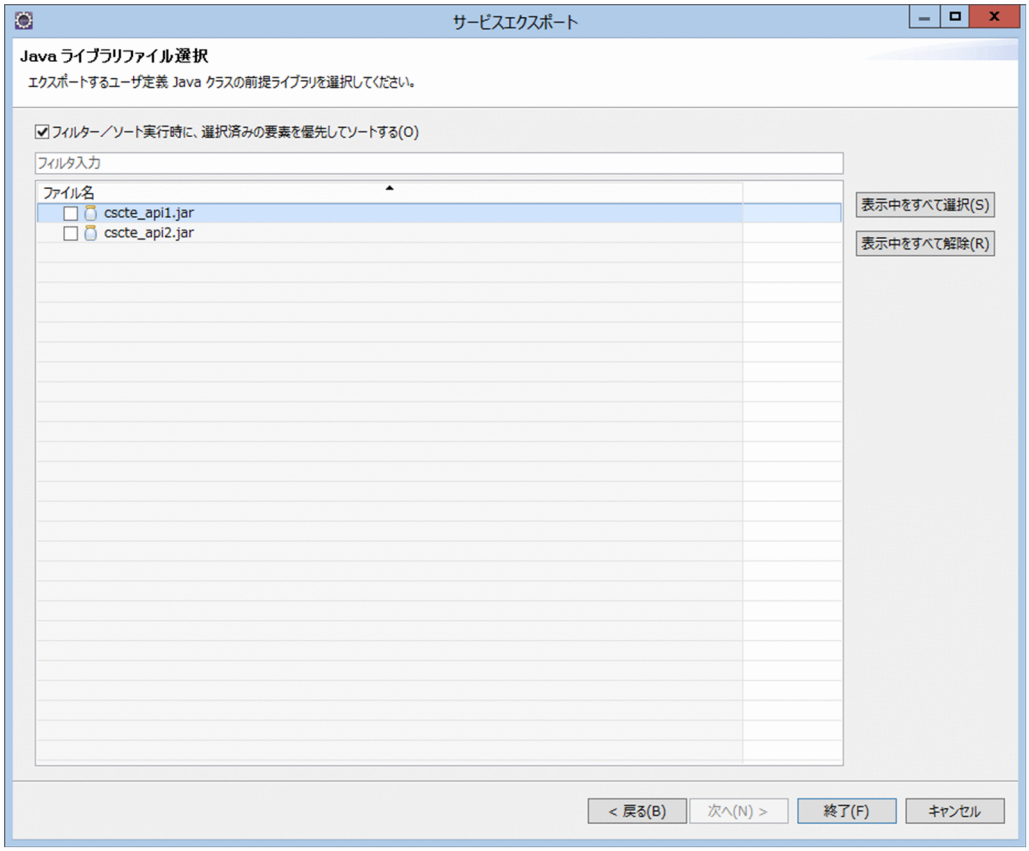
前の画面で選択した内容と，選択したユーザ定義 Java クラスが ZIP ファイルに保存されます。

ユーザ定義 Java クラスの前提ライブラリがある場合

［次へ］ボタンをクリックします。

サービスエクスポートウィザードの Java ライブラリファイル選択ページが表示されます。次の手順に進んでください。

7. エクスポートするユーザ定義 Java クラスのチェックボックスにチェックします。



8. [終了] ボタンをクリックします。

   前の画面までに選択した内容と，選択したユーザ定義 Java クラスの前提ライブラリが ZIP ファイルに保存されます。

3.2.3 リポジトリをインポートする

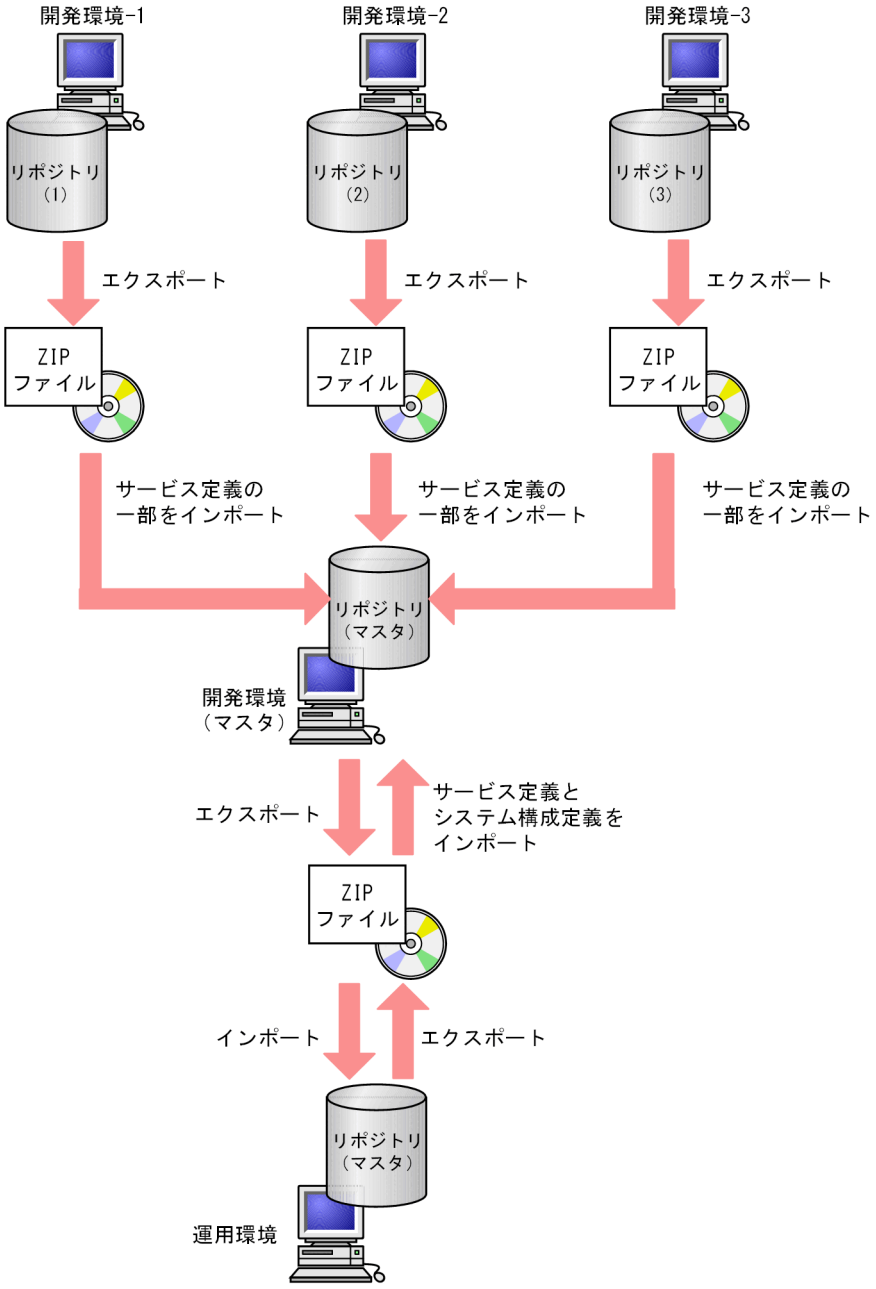
リポジトリのインポートでは，エクスポートで ZIP ファイルにしたリポジトリを読み込むことができます。インポートできる定義情報は，リポジトリに格納されたサービス定義とシステム構成定義です。エクスポートされたリポジトリと，インポート先のリポジトリの SOAP モードが一致している場合に，インポートできます。

リポジトリのインポート方法には，次の 2 つがあります。

- ZIP ファイル内のサービス定義とシステム構成定義，またはどちらか一方をインポートする
- ZIP ファイル内のサービス定義の一部だけをインポートする

2 つのインポート方法の使い分けについて次の図に例示します。

図 3-1　2 つのインポート方法の使い分け（例）



この図の例に示すように，複数の開発環境で同時に HCSC コンポーネントを開発する場合，それぞれの開発環境のリポジトリをエクスポートして，マスタのリポジトリにインポートします。このとき，サービス定義の中で必要な情報だけをインポートします。また，運用環境でエクスポートされたリポジトリの情報を開発環境のリポジトリ（マスタ）にインポートする場合，サービス定義とシステム構成定義をインポートします。

なお，エクスポートした情報は，媒体を利用しないでネットワーク経由で受け渡すこともできます。

注意事項

HCSCTE プロジェクトの lib ディレクトリや src ディレクトリにアクセスしないようにしてください。エラーが発生した場合，次の点を確認してください。

- HCSCTE プロジェクトの lib ディレクトリ以下のライブラリを利用したプログラムを実行していないか

2 つのインポート方法の詳細を次に説明します。

(1)　サービス定義とシステム構成定義をインポートする

(a)　インポート手順

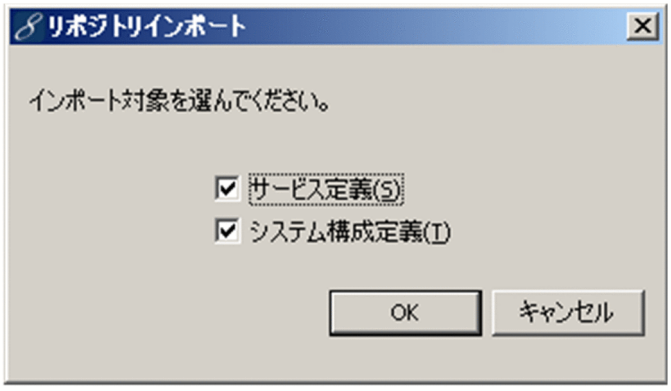
リポジトリ中のサービス定義とシステム構成定義をインポートする手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［定義情報管理］－［全定義情報インポート］を選択します。
   インポートを確認するダイアログが表示されます。

2. ［はい］ボタンをクリックします。
   リポジトリ情報の ZIP ファイルを選択する［リポジトリインポート］ダイアログが表示されます。

3. 読み込む ZIP ファイルを指定します。

4. ［開く］ボタンをクリックします。
   インポート対象の定義情報を選択する［リポジトリインポート］ダイアログが表示されます。



5. インポートしたい定義のチェックボックスにチェックします。
   初めてインポートする場合は，［サービス定義］と［システム構成定義］の両方にチェックしてください。

6. ［OK］ボタンをクリックします。
   「3.2.1(1)　リポジトリの指定」で指定したディレクトリにリポジトリが読み込まれます。

(b) 注意事項

- 旧バージョンのリポジトリをインポートすると，リポジトリ情報が今バージョンに引き継がれます。
- サービス定義をインポートした場合，ユーザ定義 Java クラスのソースファイルおよび前提ライブラリも上書きインポートします。
- Java 呼出アクティビティで利用するユーザ定義 Java クラス，またはユーザ定義 Java クラスの前提ライブラリをインポートした場合，HCSCTE プロジェクトをビルドしてください。
- サービス定義，システム構成定義のどちらか一方だけをインポートする場合，ZIP ファイル中のインポート対象の定義情報と現在のリポジトリ内の情報（サービス定義またはシステム構成定義）の整合性が保たれている必要があります。不整合がある場合はエラーとなり，サービス定義やシステム構成定義をリポジトリにインポートできません。

  リポジトリのインポートで整合性をチェックしている内容を次に示します。

  サービス定義だけをインポートする場合

  リポジトリ内の公開済みサービス定義と，対応する ZIP ファイル中のサービス定義について，整合性をチェックした場合に，エラーになる条件を次に示します。

  表 3-1　リポジトリのインポートの整合性チェックでエラーになる条件（サービス定義だけをインポートする場合 1）

項番	エラーになる条件
1	サービス ID が一致しない
2	サービス名が一致しない※1
3	種別（サービスアダプタまたはビジネスプロセス）が一致しない※1
4	ビジネスプロセスのバージョンが一致しない※1※2
5	ビジネスプロセスに定義されているユーザ定義受付の数が一致しない※1
6	ビジネスプロセスに定義されているユーザ定義受付の受付 ID が一致しない※1
7	ビジネスプロセスに定義されているユーザ定義受付の受付名が一致しない※1
8	ZIP ファイル中のサービス定義がパッケージングされていない※1※2

注※1

サービス ID が一致するサービス定義同士をチェックします。

注※2

旧バージョンで作成したリポジトリの場合はチェックしません。

また，リポジトリ内の公開済みサービス定義と，サービス ID が一致する ZIP ファイル中のサービス定義について，整合性をチェックした場合に，エラーになる条件を次に示します。なお，リポジトリ内のシステム構成定義で定義されているサーバ構成によって，エラーになる条件が異なります。

表 3-2　リポジトリのインポートの整合性チェックでエラーになる条件（サービス定義だけをインポートする場合 2）

項番	リポジトリ内の情報			エラーになる条件（サービス ID が一致する ZIP ファイル中のサービス定義）
	システム構成定義で定義されているサーバ構成		公開済みのサービス定義（サービスの種別）	
	データベースの使用有無	Reliable Messaging の使用有無		
1	使用する	使用する	すべて	−
2	使用しない	使用しない	ビジネスプロセス	ステータスの永続化に yes が設定されている
3			• SOAP アダプタ • SessionBean アダプタ • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ • gRPC アダプタ • 汎用カスタムアダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている
4	使用する	使用しない	• SOAP アダプタ • SessionBean アダプタ • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ • gRPC アダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている

項番	リポジトリ内の情報			エラーになる条件 （サービス ID が一致する ZIP ファイル中のサービス定義）
	システム構成定義で定義されているサーバ構成		公開済みのサービス定義（サービスの種別）	
	データベースの使用有無	Reliable Messaging の使用有無		
4	使用する	使用しない	• 汎用カスタムアダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている

（凡例）

－：エラーになる条件はありません。

エラーが発生した場合の対処方法

現状のリポジトリ内のサービスをすべて非公開にして，再度，サービス定義だけをインポートしてください。

システム構成定義だけをインポートする場合

ZIP ファイル中の公開済みサービス定義と，リポジトリ内のサービス定義について，整合性をチェックした場合に，エラーになる条件を次に示します。

表 3-3　リポジトリのインポートの整合性チェックでエラーになる条件（システム構成定義だけをインポートする場合 1）

項番	エラーになる条件
1	サービス ID が一致しない
2	サービス名が一致しない※1
3	種別（サービスアダプタまたはビジネスプロセス）が一致しない※1
4	ビジネスプロセスのバージョンが一致しない※1※2
5	ビジネスプロセスに定義されているユーザ定義受付の数が一致しない※1
6	ビジネスプロセスに定義されているユーザ定義受付の受付 ID が一致しない※1
7	ビジネスプロセスに定義されているユーザ定義受付の受付名が一致しない※1
8	ZIP ファイル中のサービス定義がパッケージングされていない※1※2

注※1

サービス ID が一致するサービス定義同士をチェックします。

注※2

旧バージョンで作成したリポジトリの場合はチェックしません。

また，ZIP ファイル中の公開済みサービス定義と，サービス ID が一致するリポジトリ内のサービス定義について，整合性をチェックした場合に，エラーになる条件を次に示します。なお，ZIP ファイル中のシステム構成定義で定義されているサーバ構成によって，エラーになる条件が異なります。

表 3-4　リポジトリのインポートの整合性チェックでエラーになる条件（システム構成定義だけをインポートする場合 2）

項番	ZIP ファイル中の情報			エラーになる条件 （サービス ID が一致する ZIP ファイル中のサービス定義）
	システム構成定義で定義されているサーバ構成		公開済みのサービス定義（サービスの種別）	
	データベースの使用有無	Reliable Messaging の使用有無		
1	使用する	使用する	すべて	−
2	使用しない	使用しない	ビジネスプロセス	ステータスの永続化に yes が設定されている
3			• SOAP アダプタ • SessionBean アダプタ • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ • gRPC アダプタ • 汎用カスタムアダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている
4	使用する	使用しない	• SOAP アダプタ • SessionBean アダプタ • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている

項番	ZIP ファイル中の情報			エラーになる条件 （サービス ID が一致する ZIP ファイル中のサービス定義）
	システム構成定義で定義されているサーバ構成		公開済みのサービス定義（サービスの種別）	
	データベースの使用有無	Reliable Messaging の使用有無		
4	使用する	使用しない	• HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ • gRPC アダプタ • 汎用カスタムアダプタ	サービスの種別に MDB_WSR または MDB_DBQ が設定されている

（凡例）

－：エラーになる条件はありません。

エラーが発生した場合の対処方法

次の手順を実行して，対処してください。

1. 現状のリポジトリをエクスポート（rep.zip）して退避します。
2. インポートしようとしたアーカイブファイルから，サービス定義とシステム構成定義の両方をインポートします。
3. 手順 2 でインポートしたサービス定義をすべて非公開にします。
4. 手順 1 で退避しておいた rep.zip から，サービス定義だけをインポートします。

(2) サービス定義の一部だけをインポートする

サービス定義の一部だけをインポートする場合，インポートの対象として選択できるのは次の情報です。

- HCSC コンポーネント（サービスアダプタ，ビジネスプロセス）に関する定義
- ユーザ定義 Java クラスのソースファイル
- ユーザ定義 Java クラスの前提ライブラリファイル

インポートの対象は複数選択できます。なお，インポート対象のビジネスプロセスに関連するユーザ定義受付も同時にインポートされます。

インポート時に，任意のサービス ID または受付 ID を指定して，サービスやユーザ定義受付をインポートできます。

(a) インポート手順

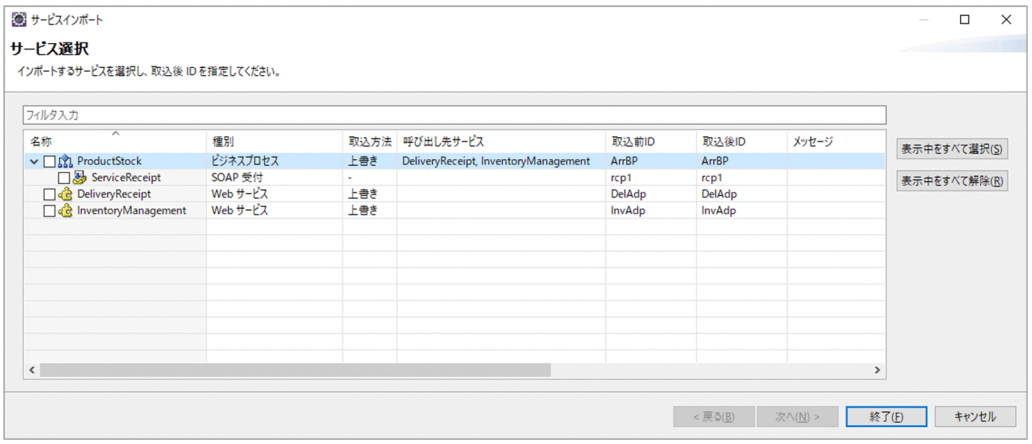
サービス定義の一部だけをインポートする手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［定義情報管理］－［サービスインポート］を選択します。

   アーカイブファイル（ZIP ファイル）を選択するダイアログが表示されます。

2. インポートする情報を含むアーカイブファイルを選択して，［開く］ボタンをクリックします。

   サービスインポートウィザードのサービス選択ページが表示されます。



3. インポートするサービスのチェックボックスにチェックします。

4. 必要に応じて，インポートするサービスの［取込後 ID］を変更します。

   次の点に注意して ID を指定してください。

   - ID は 8 バイト以内の英数字およびアンダーバー（_）で指定してください。
   - すでにインポート先で使用されている ID は指定できません。

     ただし，インポート先で使用されている ID であっても，同時にインポートするサービスによってその ID が使用されなくなる場合は，ID として指定できます。
   - ほかのサービスまたはユーザ定義受付と同じ ID は指定できません。
   - 複数のリポジトリで分散開発されたサービスを統合する場合，すべてのサービスがインポートされるまでにインターフェースの不整合が発生し，備考欄に警告メッセージが表示されることがあります。サービスを統合するまでに発生する不整合が意図したものであれば，メッセージ欄に警告メッセージが表示されても問題ありません。

     ただし，インターフェースに不整合がある場合，設計情報が正しく出力されないため，設計情報を出力しないでください。

   ユーザ定義 Java クラスの有無がチェックされ，［次へ］ボタンまたは［終了］ボタンのどちらかが活性化します。

5. 次のどれかの操作をします。

   ユーザ定義 Java クラスのソースファイルと前提ライブラリがともにない場合

   ［終了］ボタンをクリックします。

   「3.2.1(1)　リポジトリの指定」で指定したディレクトリに，選択したリポジトリの内容がインポートされます。

ユーザ定義 Java クラスの前提ライブラリがあって，ソースファイルがない場合

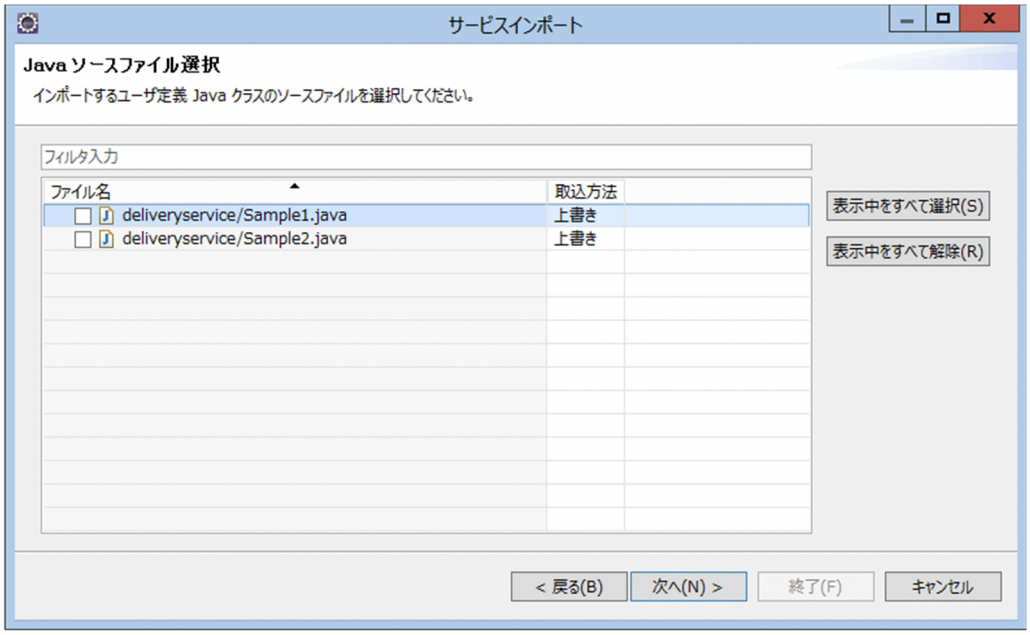
［次へ］ボタンをクリックします。

サービスインポートウィザードの Java ライブラリファイル選択ページが表示されます。手順 8.に進んでください。

ユーザ定義 Java クラスのソースファイルがある場合

［次へ］ボタンをクリックします。

サービスインポートウィザードの Java ソースファイル選択ページが表示されます。次の手順に進んでください。



6. インポートするユーザ定義 Java クラスのチェックボックスにチェックします。

7. 次のどちらかの操作をします。

ユーザ定義 Java クラスの前提ライブラリがない場合

［終了］ボタンをクリックします。

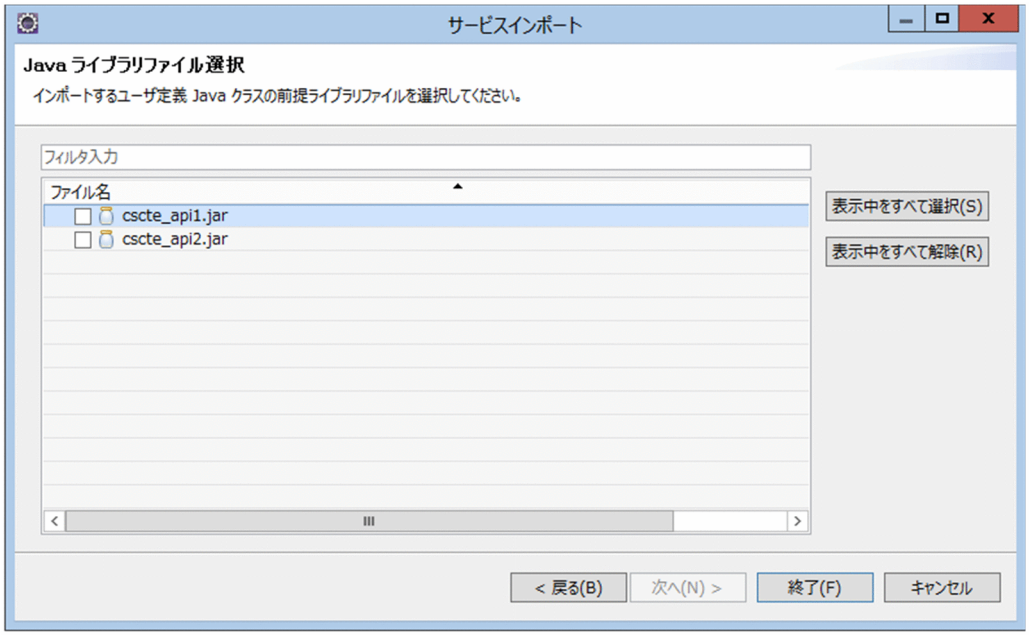
「3.2.1(1)　リポジトリの指定」で指定したディレクトリに，選択したリポジトリの内容が読み込まれます。

ユーザ定義 Java クラスの前提ライブラリがある場合

［次へ］ボタンをクリックします。

サービスインポートウィザードの Java ライブラリファイル選択ページが表示されます。次の手順に進んでください。

8. インポートするユーザ定義 Java クラスの前提ライブラリファイルのチェックボックスにチェックします。



9. [終了] ボタンをクリックします。

「3.2.1(1)　リポジトリの指定」で指定したディレクトリに，選択したリポジトリの内容が読み込まれます。

ポイント

- 同じ名称の HCSC コンポーネントがリポジトリで公開されている場合など，インポートの対象として選択できないとき，サービスインポートウィザードのサービス選択ページ内でのサービスの表示は非活性になります。メッセージ欄の内容を参考に，原因を取り除いてから再度インポートしてください。
- インポートした HCSC コンポーネント（サービスアダプタ，ビジネスプロセス）は，必ずパッケージングしてください。パッケージングの方法については，「8.2　パッケージング」を参照してください。
- Java 呼出アクティビティで利用するユーザ定義 Java クラス，またはユーザ定義 Java クラスの前提ライブラリをインポートした場合，HCSCTE プロジェクトをビルドしてください。

(b)　注意事項

- リポジトリ中で同じ名称の HCSC コンポーネントが公開済みの場合，インポートできません。インポートする前に非公開にしてください。
- リポジトリのサービス定義の一部だけをインポートする場合，システム構成定義はインポートされません。すでに公開している HCSC コンポーネントをインポートすると，インポートした HCSC コンポーネントは非公開になります。
- インポート先のリポジトリに，インポート対象の HCSC コンポーネントと同じ名称の HCSC コンポーネントがある場合，内容がすべて上書きされます。フォーマット ID はすべて更新されます。

- 古いバージョンで作成したリポジトリから，サービス定義の一部だけをインポートすることはできません。次の手順でリポジトリを移行し，サービス定義の一部をインポートしてください。

  1. 現在のリポジトリをアーカイブファイルにエクスポートします。
  2. 旧バージョンのアーカイブファイルから，すべての定義情報をインポートします。
  3. 手順 2 でインポートした定義情報を別のアーカイブファイルにエクスポートします。
  4. 手順 1 でエクスポートしたアーカイブファイルから，すべての定義情報をインポートします。
  5. 手順 3 でエクスポートしたアーカイブファイルから，サービス定義の一部をインポートします。

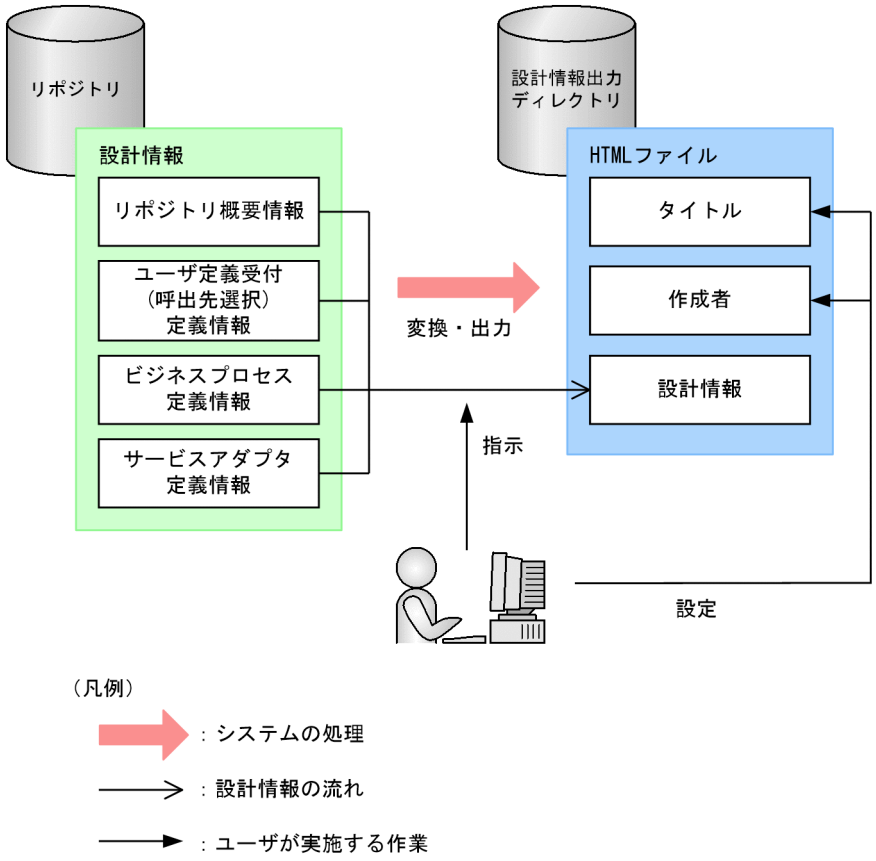
- 運用環境で HCSC サーバのセットアップを実施した直後にエクスポートされたリポジトリを，開発環境にインポートする場合，サービス定義の一部だけをインポートする操作（サービスインポート）ではなく，Eclipse のメニューから［HCSC-Definer］－［定義情報管理］－［全定義情報インポート］を選択して，定義情報をすべてインポートしてください。

3.3　設計情報の出力

開発環境では，リポジトリの概要情報，ユーザ定義受付（呼出先選択）の定義情報，ビジネスプロセスの定義情報，およびサービスアダプタの定義情報を設計情報として HTML ファイルに出力できます。設計情報を出力すると，開発環境の画面を利用しないで，定義した情報を確認できるため，作業の効率化が図れます。また，設計情報は統一されたフォーマットで HTML ファイルに出力されます。

設計情報の出力の概要を次の図に示します。

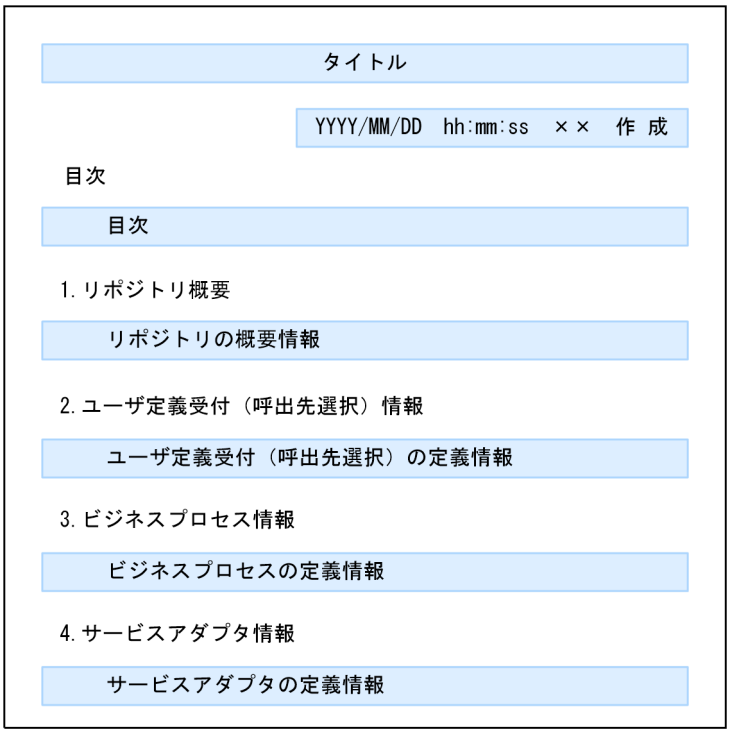
図 3-2　設計情報の出力の概要



3.3.1　設計情報として出力できる情報

設計情報として出力できる情報は，設計情報の出力ウィザードで選択できます。設計情報として出力できる情報を次の図に示します。

図 3-3　設計情報として出力できる情報



タイトル

設計情報の出力ウィザードの［タイトル］に指定した文字列が出力されます。

作成日時

設計情報の出力を開始したときの日時が出力されます。「YYYY/MM/DD　hh:mm:ss」の形式で表示されます。

作成者

設計情報の出力ウィザードの［作成者］に作成者を指定した場合，指定した作成者が出力されます。「×× 作 成」の形式で表示されます。「××」は作成者を示します。

作成者を指定しなかった場合，作成者は出力されず，「作 成」だけが表示されます。

目次

出力情報の目次が出力されます。

リポジトリの概要情報

製品バージョン，およびリポジトリに存在するすべてのサービス部品の概要情報と各サービスの依存関係が出力されます。

ユーザ定義受付（呼出先選択）の定義情報

ユーザ定義受付定義画面でユーザが定義した情報がユーザ定義受付（呼出先選択）単位で出力されます。

ビジネスプロセスの定義情報

ビジネスプロセス定義画面でユーザが定義した情報がサービス部品単位で出力されます。ビジネスプロセスの定義情報には，ビジネスプロセス定義画面で定義したアクティビティの配置図も含まれます。

サービスアダプタの定義情報

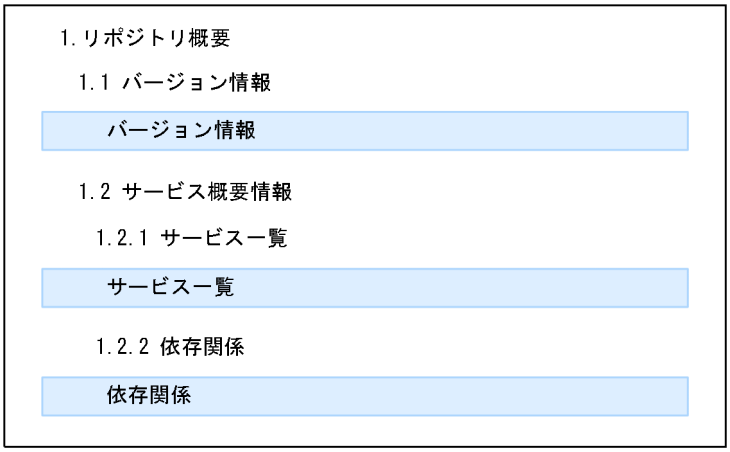
サービスアダプタ定義画面でユーザが定義した情報がサービス部品単位で出力されます。

リポジトリ概要，ユーザ定義受付（呼出先選択）情報，ビジネスプロセス情報，サービスアダプタ情報に出力できる情報を次に説明します。また，ビジネスプロセス情報内に含まれるデータ変換の定義情報についても説明します。

(1) リポジトリ概要

リポジトリ概要を次の図に示します。

図 3-4 リポジトリ概要



バージョン情報

Service Architect の製品バージョンが出力されます。

サービス一覧

設計情報の出力ウィザードで選択したサービス部品およびユーザ定義受付（呼出先選択）の概要情報が出力されます。

依存関係

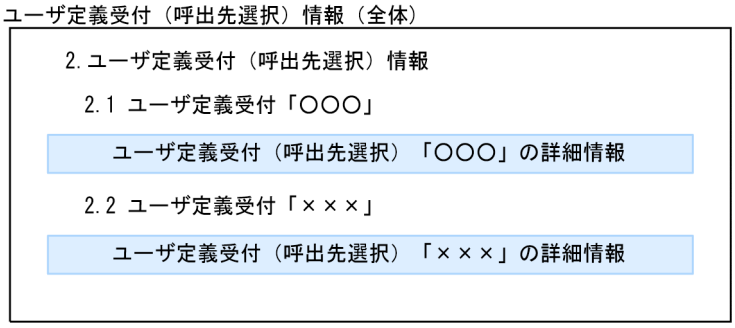
設計情報の出力ウィザードで選択した各サービス部品の依存関係が出力されます。ただし，ユーザ定義受付（呼出先選択）の依存関係はリポジトリで管理していないため出力されません。

(2) ユーザ定義受付（呼出先選択）情報

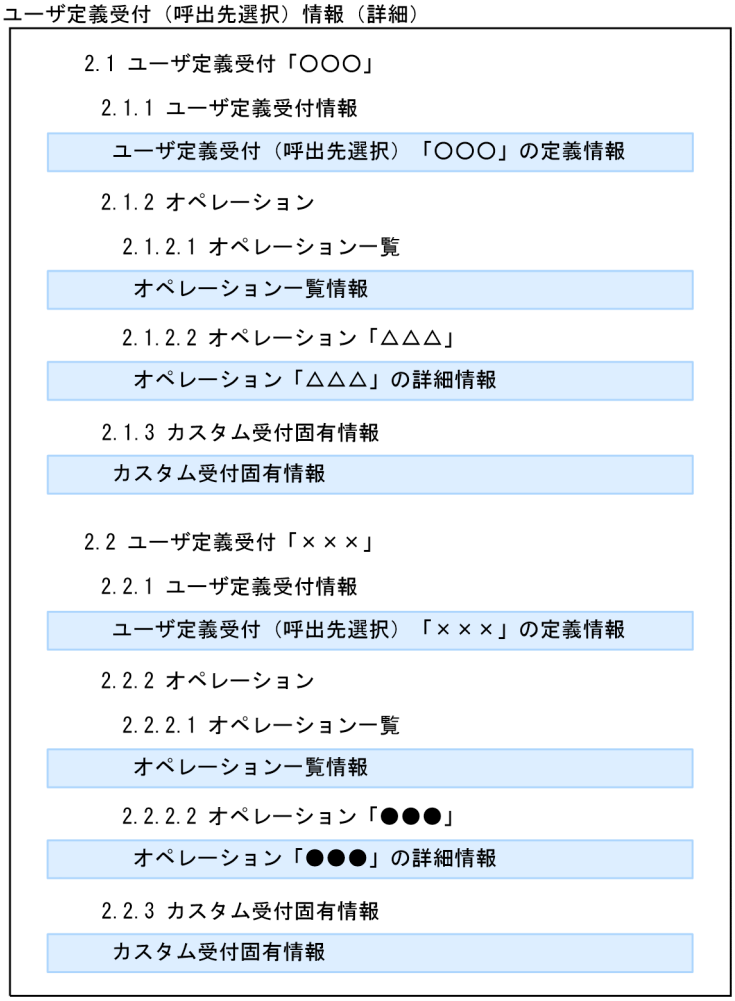
ユーザ定義受付（呼出先選択）情報の全体および詳細を次の図に示します。

図 3-5　ユーザ定義受付（呼出先選択）情報

ユーザ定義受付（呼出先選択）情報（全体）



ユーザ定義受付（呼出先選択）情報（詳細）



ユーザ定義受付（呼出先選択）情報

設計情報の出力ウィザードで選択したユーザ定義受付（呼出先選択）の詳細情報が，ユーザ定義受付（呼出先選択）ごとに出力されます。図内のユーザ定義受付「○○○」の設計情報として出力できる情報について，次に示します。

ユーザ定義受付（呼出先選択）「○○○」の定義情報

受付名，受付 ID，受付種別，およびデフォルトオペレーション名が出力されます。「○○○」にはユーザ定義受付名が表示されます。

オペレーション一覧情報

ユーザ定義受付「○○○」に定義されているオペレーションのオペレーション名，および通信モデルが一覧で出力されます。

オペレーション「△△△」の詳細情報

オペレーション「△△△」の，電文の定義情報が詳細情報として出力されます。詳細情報は，オペレーション一覧の順にオペレーションごとに出力されます。「△△△」にはオペレーション名が表示されます。

カスタム受付固有情報

独自定義ファイルの有無，および定義の状態によって出力される内容が異なります。

- 独自定義ファイルが存在しない場合

  独自定義ファイルの固有情報だけ出力されます。

- 独自定義ファイルが存在する場合

  独自定義ファイルまたは EAR ファイルの一覧と，それぞれのファイルの内容が出力されます。

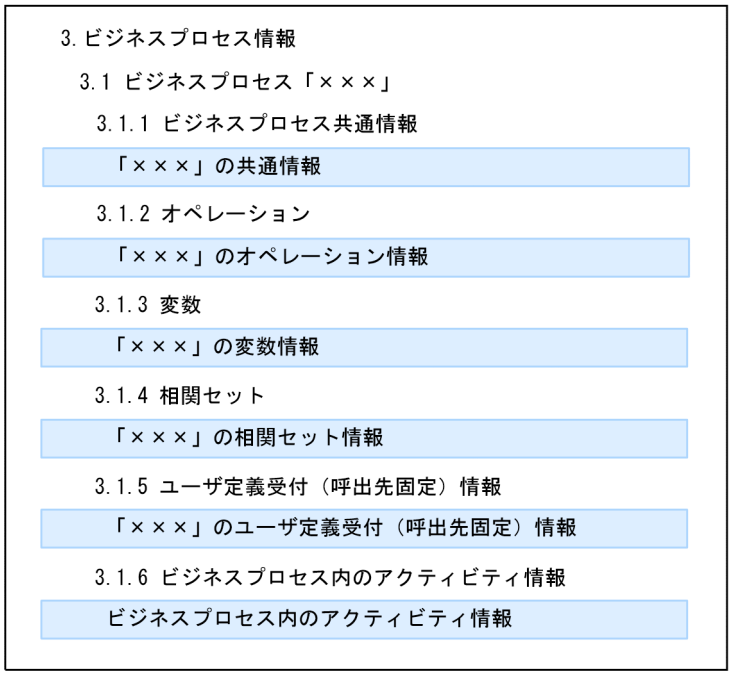
(3)　ビジネスプロセス情報

ビジネスプロセス情報は，スコープアクティビティおよび繰り返しアクティビティが定義されているかどうかによって，出力される情報が異なります。

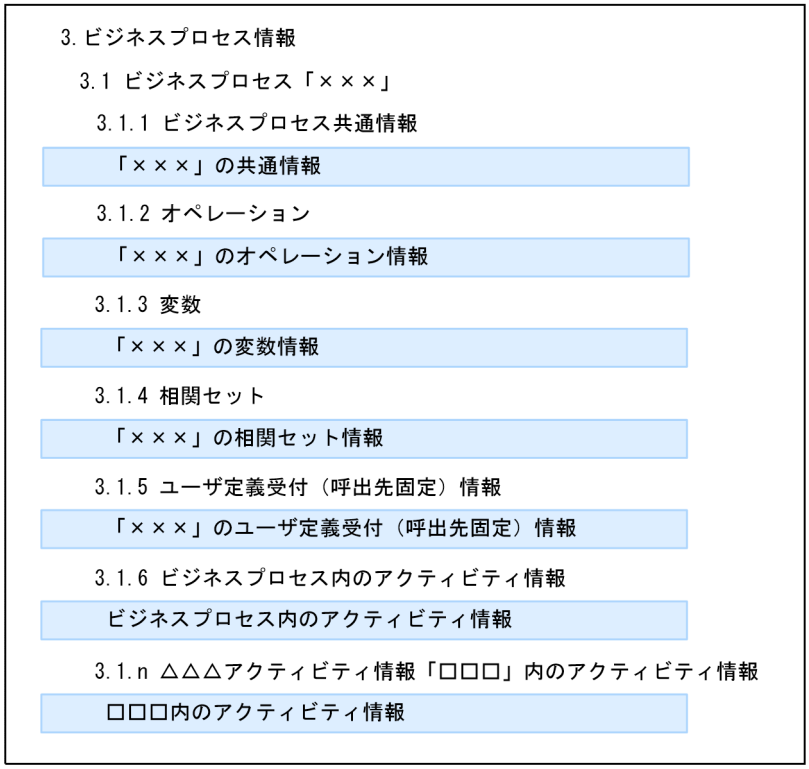
ビジネスプロセス情報を次の図に示します。

図 3-6　ビジネスプロセス情報

●スコープアクティビティおよび繰り返しアクティビティが定義されていない場合



●スコープアクティビティおよび繰り返しアクティビティが定義されている場合



「×××」の共通情報

ビジネスプロセスを作成するときに設定する情報，サービス部品の依存関係，およびアクティビティの一覧が出力されます。「×××」にはビジネスプロセス名が表示されます。

「×××」のオペレーション情報

ビジネスプロセスのオペレーション一覧，およびオペレーションごとの通信モデルや電文の定義情報がオペレーションの詳細情報として出力されます。

「×××」の変数情報

ビジネスプロセスで定義されているすべてのグローバル変数の情報が出力されます。

「×××」の相関セット情報

ビジネスプロセスで定義されているすべての相関セットの情報が出力されます。

「×××」のユーザ定義受付（呼出先固定）情報

ビジネスプロセスで定義されているユーザ定義受付（呼出先固定）の一覧，および各ユーザ定義受付（呼出先固定）の詳細情報が出力されます。

ビジネスプロセス内のアクティビティ情報

キャンバス上に配置した各アクティビティの情報が出力されます。

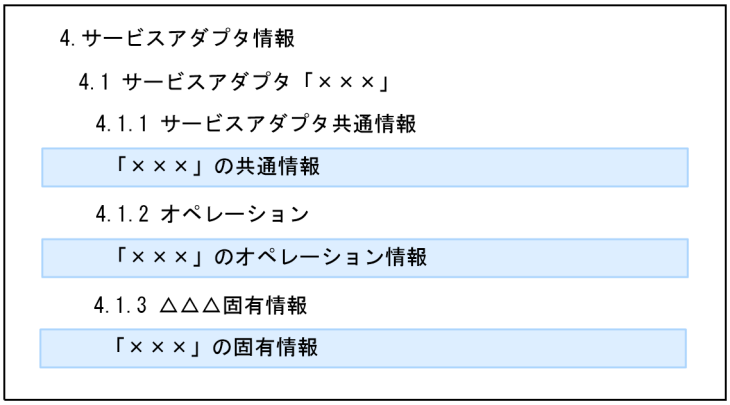
□□□内のアクティビティ情報

スコープアクティビティまたは繰り返しアクティビティの情報が繰り返し出力されます。「△△△」には「スコープ」または「繰り返し」が，「□□□」にはスコープアクティビティ名または繰り返しアクティビティ名が表示されます。

(4)　サービスアダプタ情報

サービスアダプタ情報を次の図に示します。

図 3-7　サービスアダプタ情報



「×××」の共通情報

サービス部品制御情報，および呼出し元サービスが出力されます。「×××」にはサービス名が表示されます。

「×××」のオペレーション情報

サービスアダプタのオペレーション一覧，およびオペレーションごとの通信モデルや電文の定義情報がオペレーションの詳細情報として出力されます。

「×××」の固有情報

開発環境の画面でサービスアダプタを定義したときに設定した情報が，サービス種別ごとに出力されます。「△△△」にはサービス種別が表示されます。固有情報を参照するときに注意する点を，サービス種別ごとに次に示します。

Web サービスの場合

- ベーシック認証を使用する場合，どのようなパスワードが設定されていても「****」が表示されます。
- クライアント定義ファイルに XML で使用できない文字が存在する場合，クライアント定義ファイルは正しく出力されません。

SessionBean の場合

クライアント定義ファイルに XML で使用できない文字が存在する場合，クライアント定義ファイルは正しく出力されません。

MDB（WS-R）の場合

ベーシック認証を使用する場合，どのようなパスワードが設定されていても「****」が表示されます。

MDB（DB キュー）の場合

固有情報は出力されません。

汎用カスタムアダプタの場合

独自定義ファイルが XML 形式でない場合，独自定義ファイルは正しく出力されません。

そのほかのサービスアダプタの場合

汎用カスタムアダプタの場合と同じです。DB アダプタ，TP1 アダプタ，ファイルアダプタ，Object Access アダプタ，Message Queue アダプタ，FTP アダプタ，ファイル操作アダプタ，メールアダプタ，HTTP アダプタ，コマンドアダプタ，SFTP アダプタ，Kafka アダプタ，または gRPC アダプタが該当します。

(5) データ変換の定義情報

ビジネスプロセスまたはサービスアダプタでデータ変換を定義した場合，データ変換の対応表が出力されます。データ変換の対応表を次の図に示します。

図 3-8 データ変換の対応表

#	変換先	代入値	関数詳細	依存対象
1	VariableX			
2	intf:reserve	loop1(/intf:reserve)	・ loop1 ファンクションタイプ=くり返し（Loop） ◦ ソート条件 無	
3	in0	/intf:reserve/in0		loop1
4	in1	contain1(/intf:reserve/in1)	・ contain1 ファンクションタイプ=文字列調査（contain），調査種別=指定文字列を含む，調査文字列='aaa'	loop1

変換先

変換先スキーマに含まれる各ノードが出力されます。

代入値

変換先ノードに代入する値が出力されます。1 つまたは複数の関数（ファンクション）や変換元ノードが出力されます。

関数詳細

「代入値」に出力された関数（ファンクション）に詳細情報がある場合，その詳細情報が出力されます。「代入値」に出力された関数（ファンクション）に複数の詳細情報がある場合は，複数の詳細情報が出力されます。

依存対象

「代入値」が依存するくり返しファンクションが出力されます。

「代入値」および「関数詳細」に出力される情報をファンクションごとに次の表に示します。

表 3-5 「代入値」および「関数詳細」に出力される情報

項番	ファンクション名	代入値の出力形式	関数詳細の出力形式
1	文字列連結	＜ファンクション名＞(＜連結する対象＞,…)	・＜ファンクション名＞

項番	ファンクション名	代入値の出力形式	関数詳細の出力形式
1	文字列連結	＜ファンクション名＞(＜連結する対象＞, …)	ファンクションタイプ=文字列連結 (concat)
2	部分文字列取得	＜ファンクション名＞(＜分割する対象＞)	「・＜ファンクション名＞」のあとに次のどれかが出力されます。 • ファンクションタイプ=部分文字列取得 (substr), 取得方法=先頭からの範囲指定, 開始位置=＜開始位置＞, ＜文字数=文字数 ｜ 開始位置から末尾までを取得＞ • ファンクションタイプ=部分文字列取得 (substr), 取得方法=末尾からの範囲指定, 開始位置=＜開始位置＞,＜文字数=文字数 ｜ 開始位置から先頭までを取得＞ • ファンクションタイプ=部分文字列取得 (substr), 取得方法=分割文字列指定, 分割文字列='＜分割に利用する文字列＞', 取得部位=＜前 ｜ 後＞
3	文字数取得	＜ファンクション名＞(＜文字数を取得する対象＞)	・＜ファンクション名＞ ファンクションタイプ=文字数取得 (length)
4	文字列調査	＜ファンクション名＞(＜特定の文字列が含まれているかどうかを調査する対象＞)	・＜ファンクション名＞ ファンクションタイプ=文字列調査 (contain), 調査種別=＜指定文字列を含む ｜ 指定文字列から開始＞, 調査文字列='＜調査文字列＞'
5	空白除去	＜ファンクション名＞(＜空白を除去する対象＞)	・＜ファンクション名＞ ファンクションタイプ=空白除去 (trim)
6	数値フォーマット変換	＜ファンクション名＞(＜フォーマットしたい数値＞)	・＜ファンクション名＞ ファンクションタイプ=数値フォーマット変換 (format), パターン='＜フォーマットパターン＞', 記号の変更=＜なし ｜ (小数点文字='＜小数点文字＞', 桁区切り文字='＜桁区切り文字＞')＞
7	数値演算	＜ファンクション名＞(＜１つ目の演算の対象＞, '＜+ ｜ - ｜ * ｜ / ｜ %＞', ＜２つ目の演算の対象＞)	・＜ファンクション名＞ ファンクションタイプ=数値演算 (calc)
8	端数処理	＜ファンクション名＞(＜端数処理を行う対象＞)	・＜ファンクション名＞ ファンクションタイプ=端数処理 (round), 端数処理種別=＜四捨五入 ｜ 切り捨て ｜ 切り上げ＞
9	総和	＜ファンクション名＞(＜総和を取る対象＞, …)	・＜ファンクション名＞ ファンクションタイプ=総和 (sum)
10	否定演算	＜ファンクション名＞(＜否定演算の対象＞)	・＜ファンクション名＞ ファンクションタイプ=否定演算 (not)
11	論理演算	＜ファンクション名＞(＜１つ目の演算の対象＞, ＜２つ目の演算の対象＞)	・＜ファンクション名＞

項番	ファンクション名	代入値の出力形式	関数詳細の出力形式
11	論理演算	<ファンクション名>(<１つ目の演算の対象>, <２つ目の演算の対象>)	ファンクションタイプ=論理演算 (bitop), 演算=<論理積 (AND)|論理和 (OR)|排他的論理和 (XOR)>
12	シフト演算	<ファンクション名>(<シフト演算の対象>)	シフト方向が「左」の場合 ・<ファンクション名> ファンクションタイプ=シフト演算 (shift), シフト量=<シフト量> ビット, 出力サイズ=<出力サイズ> ビット, シフト方向=左 シフト方向が「右」の場合 ・<ファンクション名> ファンクションタイプ=シフト演算 (shift), シフト量=<シフト量> ビット, 出力サイズ=<出力サイズ> ビット, シフト方向=右, シフト種別=<算術シフト|論理シフト>
13	ノード数取得	<ファンクション名>(<ノード数を取得したいノード>)	・<ファンクション名> ファンクションタイプ=ノード数取得 (count)
14	ノード名取得	<ファンクション名>(<ノード名を取得したいノード>)	・<ファンクション名> ファンクションタイプ=ノード名取得 (name)
15	ノード調査	<ファンクション名>(<存在を確認したいノード>)	・<ファンクション名> ファンクションタイプ=ノード調査 (exist)
16	くり返し	<ファンクション名>(<ループの基準となるパス>)	・<ファンクション名> ファンクションタイプ=くり返し (loop) ◦ ソート条件 <無 | キー=<キーとなるノード>, 順序=<昇順 | 降順>, 言語=<自動 | 日本語 | 英語>, タイプ=<テキスト | 数値>, 優先順位=<大文字 | 小文字>> ◦ 連携パス <無 | <連携パスのパス>>
17	選択	<ファンクション名>([<条件値>]=>[<出力値>], …, [どの条件にも合致しない場合]=>[<出力値>]) [<出力値>]には次のどれかが出力されます。 • ノードを出力しない • 空ノード • <出力される値> • 値を出力する	・<ファンクション名> ファンクションタイプ=選択 (choose)
18	定数	<ファンクション名>(<文字列:'<値>' | 数値: '<値>' | 論理値:<真 | 偽> | 特殊ノード:<ノード出力なし | 空ノード>>)	・<ファンクション名> ファンクションタイプ=定数 (const)

項番	ファンクション名	代入値の出力形式	関数詳細の出力形式
19	値置換え	＜ファンクション名＞(＜置換対象のノード＞)	・＜ファンクション名＞ ファンクションタイプ=値置換え (replace), 変換テーブル ID='＜変換テーブル ID ＞', パスプロパティ='csc.dt.valueTable.＜変換テーブル ID ＞', コードプロパティ='csc.dt.encodeType.＜変換テーブル ID ＞', 検索キー列指定=＜左列｜右列＞, 検索失敗時の動作=＜デフォルト値を代入, 値= '＜値＞'｜変換エラー＞
20	基数変換	＜ファンクション名＞(＜基数変換の対象＞)	・＜ファンクション名＞ ファンクションタイプ=基数変換 (radix), 入力基数=＜ 16 進数|10 進数|2 進数＞, 出力基数=＜ 16 進数|10 進数|2 進数＞
21	カスタム	＜ファンクション名＞(＜引数の入力値＞, …) 注 引数がない場合は，丸括弧の中には何も表示されません。	引数がない場合 ・＜ファンクション名＞ ファンクションタイプ=カスタム (custom), jar ファイル='＜ jar ファイル名＞', クラス='＜クラス名＞', メソッド='＜メソッド名＞', 戻り値の型='＜戻り値の型名＞', メソッドの説明='＜メソッドの説明＞' 引数がある場合 ・＜ファンクション名＞ ファンクションタイプ=カスタム (custom), jar ファイル='＜ jar ファイル名＞', クラス='＜クラス名＞', メソッド='＜メソッド名＞', 戻り値の型='＜戻り値の型名＞', 引数の型='＜引数の型名＞', メソッドの説明='＜メソッドの説明＞' 注 引数がある場合は，上記形式で表示される情報の直下に引数名と引数の説明が一覧表で表示されます。
22	XPath	＜ファンクション名＞(＜ XPath 式＞)	・＜ファンクション名＞ ファンクションタイプ=XPath (xpath)

3.3.2　出力前の確認事項

設計情報を出力する前の確認事項を次に示します。

(1)　利用できる Web ブラウザ

出力した設計情報（HTML）を参照するためには，次の Web ブラウザを利用してください。

- Microsoft Edge

(2)　リポジトリの状態

設計情報を出力する場合，HCSC コンポーネントのパッケージングでエラーが発生しない状態のリポジトリを使用することをお勧めします。データ変換定義の検証エラーによってパッケージング時にエラーが発生した場合，検証エラーとなったデータ変換定義の対応表は出力されません。

なお，データ変換定義の検証エラーが発生した場合，コンソールビューにエラーが発生したことを通知するメッセージが表示されます。また，設計情報の出力終了時に，エラーが発生したことを通知するダイアログが表示されます。

(3)　設計情報の出力先

設計情報の出力先は，ディレクトリまたはファイルが存在しない場所を指定してください。ディレクトリまたはファイルが存在する場合，ファイルが上書きされたり，不要なファイルが残ったりすることがあります。なお，指定した設計情報の出力先に，ディレクトリまたはファイルが存在する場合は，処理を続行するかを問い合わせるダイアログが表示されます。

注意事項

ユーザアクセスコントロール機能を有する OS で使用する場合

設計情報の出力先ディレクトリに管理者特権が必要なディレクトリを指定する場合は，Eclipse を管理者特権で実行してください。管理者特権が付与されていない場合，設計情報の出力中にエラーが発生します。

(4)　設計情報の出力時間

出力対象のデータが大きい場合や使用するマシンの性能によっては，設計情報の出力に時間が掛かることがあります。設計情報の出力中は，Eclipse 上でほかの作業はできませんので注意してください。

(5)　設計情報の印刷

ブラウザから印刷する場合，印刷用紙のサイズが小さいと出力内容の右端が切れて印刷されることがあります。そのため，印刷前に印刷プレビュー機能で印刷内容を確認して，適切な用紙サイズおよび余白を設定することをお勧めします。

3.3.3　設計情報の出力方法

設計情報の出力方法を次に示します。

1. メニューから［HCSC-Definer］－［設計情報の出力］を選択します。

サービスまたはユーザ定義受付を編集中の場合

編集中のサービスまたはユーザ定義受付の保管（非公開化）を確認するダイアログが表示されます。複数のサービスまたはユーザ定義受付が編集中の場合は，複数回ダイアログが表示されます。保管する場合は［はい］ボタンをクリックします。これ以降，確認ダイアログを表示しないで，すべて保管する場合は［すべてはい］ボタンをクリックします。保管しない場合は［キャンセル］ボタンをクリックして，処理を終了します。

手順 2.に進みます。

サービスまたはユーザ定義受付を編集中でない場合

設計情報の出力ウィザードが表示されます。手順 3.に進みます。

2. ［はい］ボタンまたは［すべてはい］ボタンをクリックします。

設計情報の出力ウィザード（基本情報の入力ページ）が表示されます。なお，編集中のサービスまたはユーザ定義受付が配備定義（公開）されているか配備定義されていないか（非公開）によって，次のように処理が異なります。

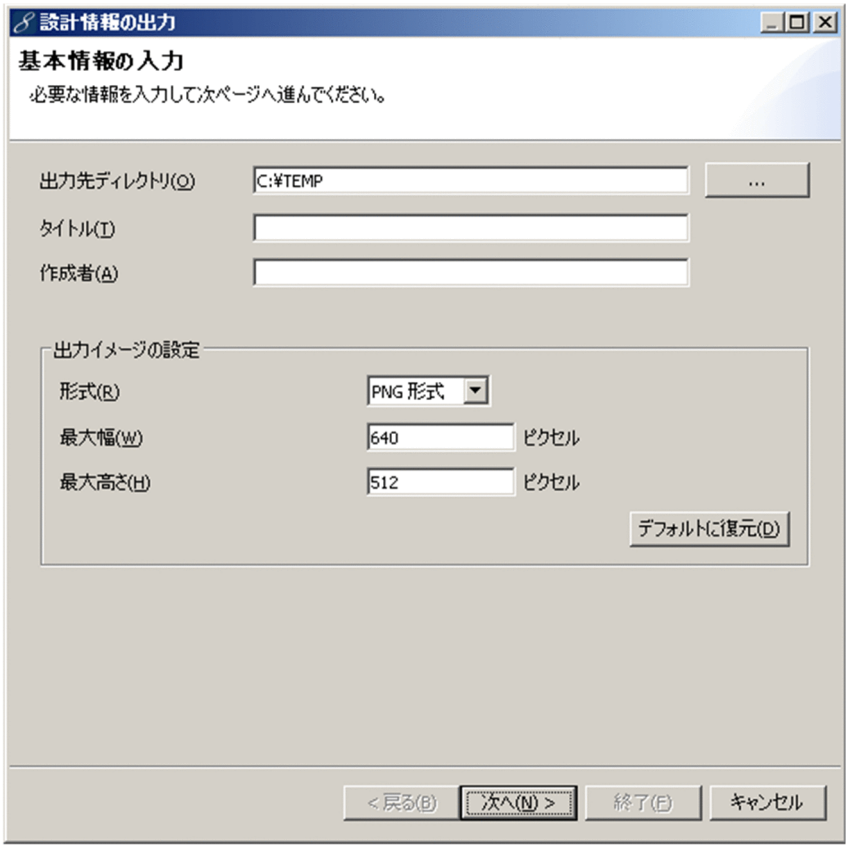
サービスまたはユーザ定義受付が配備定義されている場合

配備定義されているサービスまたはユーザ定義受付を配備定義されていない状態にします。そのあと，編集中のサービスまたはユーザ定義受付を保存して処理を続行します。

サービスまたはユーザ定義受付が配備定義されていない場合

編集中のサービスまたはユーザ定義受付を保存して処理を続行します。

3. 設計情報の出力ウィザード（基本情報の入力ページ）で，出力先ディレクトリなど必要な情報を設定します。



出力先ディレクトリ

設計情報を出力するディレクトリを指定します。設計情報の出力先を指定する場合は，次の点に注意してください。

・出力先ディレクトリには，ディレクトリまたはファイルが存在しない場所を指定してください。指定したディレクトリに，同一名のディレクトリまたはファイルが存在する場合は，上書きされます。また，ビジネスプロセス数が異なる場合は，不要な画像ファイルが残ることがあります。

・パスは絶対パスで指定してください。

・文字コード「MS932」で 239 バイト以下のパスを指定してください。

・設計情報の出力を開始したあとに，エラーが発生して出力処理を中止した場合，中止する前までのディレクトリまたはファイルが残ることがあります。

タイトル

設計情報のタイトルとして出力する文字列を指定します。

なお，タイトルに指定する文字列として，次の文字コードが使用できます。

U+0009，U+0020-U+D7FF，U+E000-U+FFFD，U+10000-U+10FFFF

また，タブ（U+0009），半角空白（U+0020），および全角空白（U+3000）は文字の間でだけ使用できます。

作成者

設計情報の作成者として出力する文字列を指定します。

なお，作成者に指定する文字列として，次の文字コードが使用できます。

U+0009，U+0020-U+D7FF，U+E000-U+FFFD，U+10000-U+10FFFF

また，タブ（U+0009），半角空白（U+0020），および全角空白（U+3000）は文字の間でだけ使用できます。

形式

アクティビティ配置図の出力イメージの形式を指定します。PNG 形式，BMP 形式，JPEG 形式から選択します。

最大幅

出力イメージの最大幅を指定します。指定できる範囲は 250～2500 です。

出力イメージの幅サイズが指定した最大幅を超える場合は，最大幅に収まるように，幅と高さの比率を保ったまま出力イメージを縮小して表示します。

最大高さ

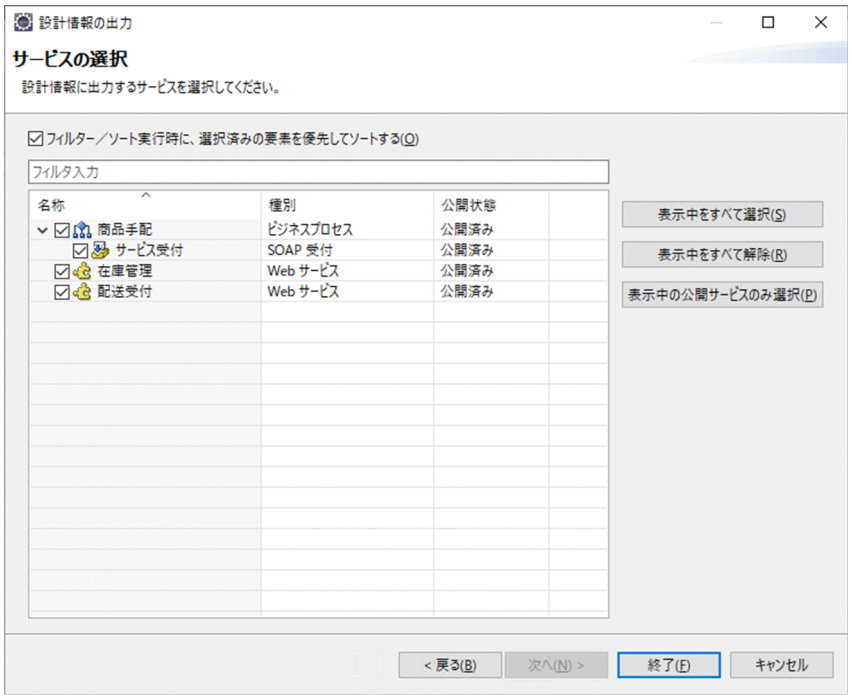
出力イメージの最大高さを指定します。指定できる範囲は 200～2000 です。

出力イメージの高さサイズが指定した最大高さを超える場合は，最大高さに収まるように，幅と高さの比率を保ったまま出力イメージを縮小して表示します。

4. [次へ] ボタンをクリックします。

設計情報の出力ウィザード（サービスの選択ページ）が表示されます。

5. 設計情報の出力ウィザード（サービスの選択ページ）で，設計情報として出力するサービスのチェックボックスにチェックします。



6. [終了] ボタンをクリックします。

設計情報の出力処理状況がダイアログに表示されます。

出力処理が完了すると処理結果がダイアログに表示され，出力先ディレクトリに設計情報が出力されます。

3.3.4　設計情報の出力に関する注意事項

設計情報の出力に関する注意事項を次に示します。

- 変数名を直接指定して情報を取得する方法で XPath を記載した場合，XPath 中の変数への参照先リンクは出力されません。ブラウザの検索機能などを使用して，変数情報を参照してください。
- 独自定義ファイルが XML 形式の場合，次の文字エンコーディングが指定されていると，設計情報出力でファイルの内容を出力できないことがあります。
  - ISO-10646-UCS-2
  - UTF-32
  - UTF-32BE
  - UTF-32LE
  - ISO-10646-UCS-4

3.4　差分確認用ドキュメントの出力

データ変換定義のモジュールの変更前と変更後の差分を確認するための HTML ファイルを出力できます。

この機能を差分確認用ドキュメント出力機能と呼びます。この機能で生成した HTML ファイルを，Word などを用いて比較することによって，次のことをビジュアライズに確認できます。

- 変更前と変更後のファイルを比較して，変更個所の変更内容が正しいこと
- 変更個所のないファイルを比較して，その差分がないこと

注意事項

出力された HTML ファイルを用いて差分を確認する際，使用する差分確認ツールによっては，表の体裁が崩れたり，書式の変更が差分として出力されたりすることがあります。

また，Word などを用いて HTML ファイルを解析して表示した結果から差分を取得するのではなく，テキストとして差分を取得した場合，HTML の表構造に依存した内容が差分として表示されることがあります。

3.4.1　差分確認用ドキュメント出力機能の概要

差分確認用ドキュメント出力機能は，データ変換定義で必要な次のモジュールに対して，差分を確認するための HTML ファイルを自動で出力できます。この HTML ファイルを，差分確認用ドキュメントと呼びます。

- XML スキーマファイル（拡張子：.xsd）
- バイナリフォーマット定義ファイル（拡張子：.fdx）
- マッピング定義ファイル（拡張子：.mdo）

差分確認用ドキュメント出力機能は，差分取得対象の入力データ，および出力ディレクトリを指定して実行します。

入力データの指定方法を次に示します。

- ファイルを単体で指定する方法
  ファイルを単体で指定した場合，単体指定したファイルだけが入力データとして入力されます。
- ディレクトリを指定する方法
  ディレクトリを指定した場合，指定したディレクトリ以下のサブディレクトリを含む全ファイルが入力データとして入力されます。

参考

差分取得対象の入力データとなるのは，データ変換定義で必要なモジュールとして示した 3 種類のファイルだけです。それ以外は無効なファイルとして扱われ，差分確認用ドキュメントは出力されません。

入力されたファイルは HTML ファイルに変換され，指定された出力ディレクトリに出力されます。

差分確認用ドキュメント出力機能の概要について次の図に示します。

図 3-9　入力データのファイルを単体で指定した場合

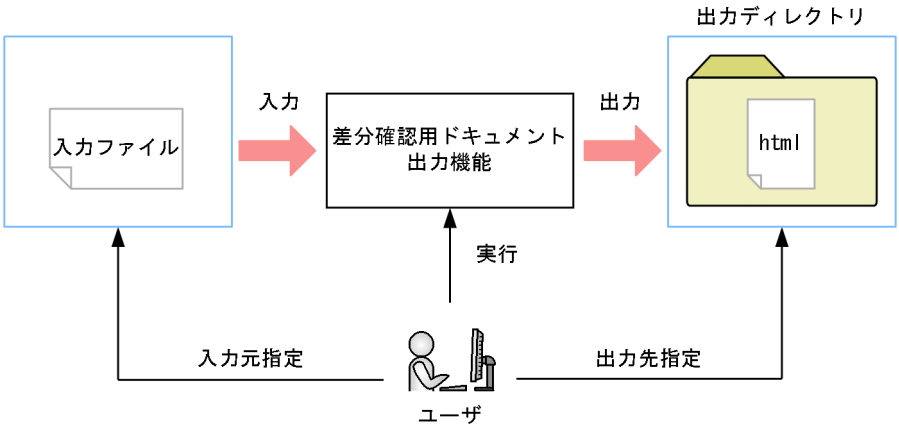
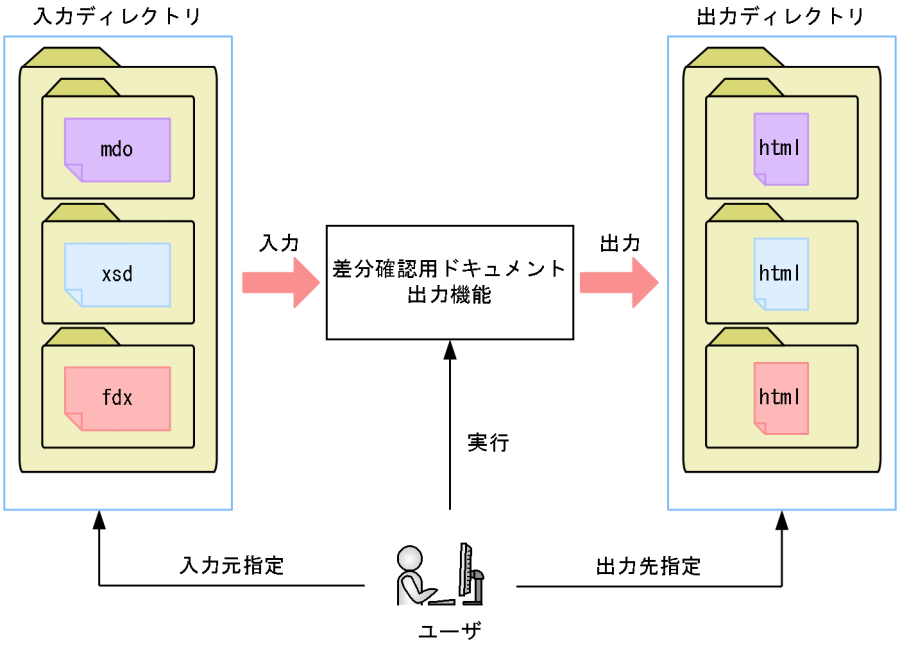


図 3-10　入力データをディレクトリで指定した場合



3.4.2　差分確認用ドキュメント出力機能の実行手順

差分確認用ドキュメント出力機能の実行手順を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［差分確認用ドキュメントの出力］を選択します。
   差分確認用ドキュメントの出力ウィザードが表示されます。
2. 差分確認用ドキュメントの出力ウィザードで，入力データ，および出力ディレクトリを指定します。
3. ［Finish］ボタンをクリックします。
   差分確認用ドキュメント出力機能が実行されます。
   - 処理が正常に終了した場合は，処理完了のメッセージが表示されます。
   - 処理が 1 つでも異常終了した場合は，エラーメッセージが表示されます。メッセージの内容に従い，エラー原因を取り除いてください。

3.4.3　差分確認用ドキュメントが出力される出力ディレクトリの構成

差分確認用ドキュメントは，実行日時に依存したディレクトリを起点として，入力ディレクトリと同じサブディレクトリの位置に出力されます。

実行日時に依存したディレクトリは，指定された出力ディレクトリの直下に作成されます。

差分確認用ドキュメントを出力するディレクトリの構成を次の表に示します。

表 3-6　差分確認用ドキュメントを出力するディレクトリの構成

ディレクトリ構成	種別	説明
<ユーザが指定した出力ディレクトリ>	ディレクトリ	差分確認用ドキュメント出力機能の結果を格納するディレクトリ。 実行前に準備する必要があります。
<実行日時>※	ディレクトリ	HTML ファイルを格納するディレクトリ。 実行するごとに自動的に生成されます。
<入力ディレクトリと同じディレクトリ構造>	ディレクトリ	入力ディレクトリ以下のサブディレクトリ。 入力データをディレクトリで指定した場合だけ有効になります。
<入力時のファイル名（拡張子除く）>.html	ファイル	差分確認用ドキュメントとして出力される HTML ファイル。

注※

実行日時は，yyyyMMddHHmmss の形式で出力されます。

・yyyy：西暦年

・MM：月（01≦MM≦12）

・dd：日（01≦dd≦31）

・HH：時（00≦HH≦23）
・mm：分（00≦mm≦59）
・ss：秒（00≦ss≦59）

3.4.4　差分確認用ドキュメント出力機能の実行例

差分確認用ドキュメント出力機能の実行例を示します。

(1)　差分確認用ドキュメントの出力例

入力データを単体で指定する場合と，入力データをディレクトリで指定する場合の入力値と出力例を次に示します。

(a)　入力データを単体で指定する場合

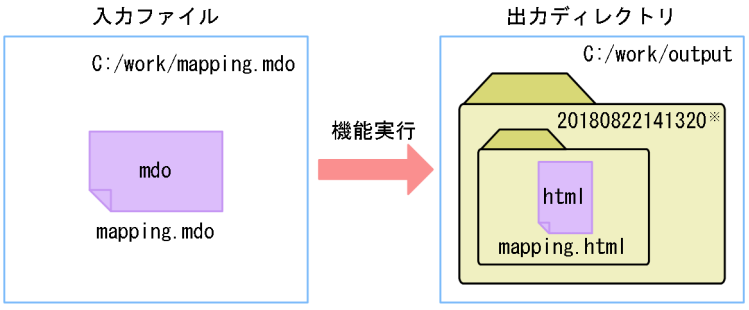
●入力値

入力データを単体で指定する場合の指定値の例を次の表に示します。

項目	指定値
入力ファイルパス	C:/work/mapping.mdo
出力ディレクトリパス	C:/work/output

●出力例

入力データを単体で指定した場合の出力例を次の図に示します。



注※
実行時の時刻によって異なります。

上記の出力例の場合，差分確認用ドキュメントの出力先は次のとおりです。

```
C:/work/output/20180822141320/mapping.html
```

(b)　入力データをディレクトリで指定する場合

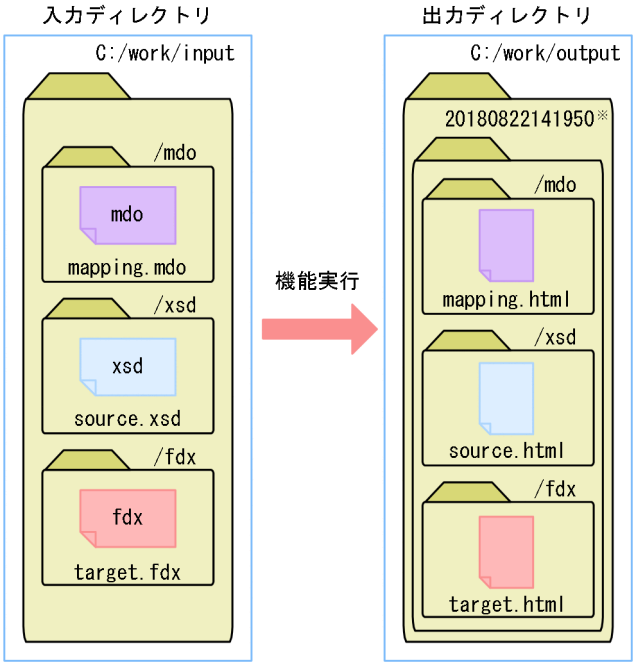
●入力値

入力データをディレクトリで指定する場合の指定値の例を次の表に示します。

項目	指定値
入力ファイルパス	C:/work/input
出力ディレクトリパス	C:/work/output

●出力例

入力データをディレクトリで指定した場合の出力例を次の図に示します。



注※
実行時の時刻によって異なります。

上記の出力例の場合，各データの差分確認用ドキュメントの出力先は次のとおりです。

・入力データが「mapping.mdo」の場合

```
C:/work/output/20180822141950/mdo/mapping.html
```

・入力データが「source.xsd」の場合

```
C:/work/output/20180822141950/xsd/source.html
```

・入力データが「target.fdx」の場合

```
C:/work/output/20180822141950/fdx/target.html
```

(2)　出力ディレクトリの構成に関する注意事項

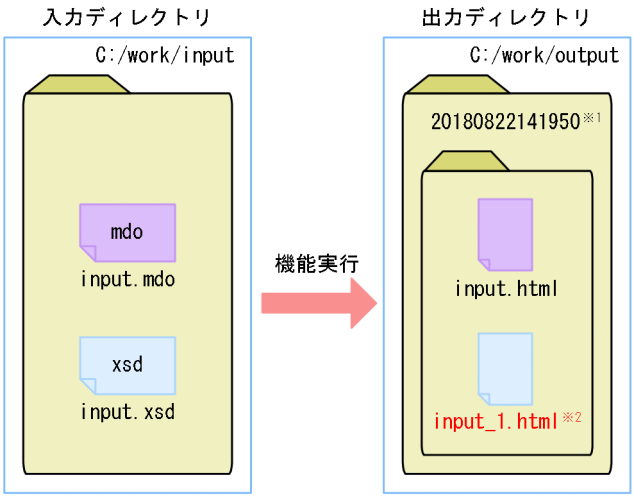
出力ディレクトリの構成に関する注意事項を次に示します。

(a)　拡張子が異なる同名ファイルのファイル名の変更について

入力ディレクトリ内の同じディレクトリに，同じファイル名で拡張子が異なるファイルが存在する場合は，2つ目以降のファイル名に連番を付与して出力します。この連番は，ファイル名が重複しないように，生

成した順に 1 から付けられる自然数の連番です。この際，入力ファイルパスと出力ファイルパスの対応関係は HCSCTE メッセージ・ビューに出力されます。

出力例を次に示します。



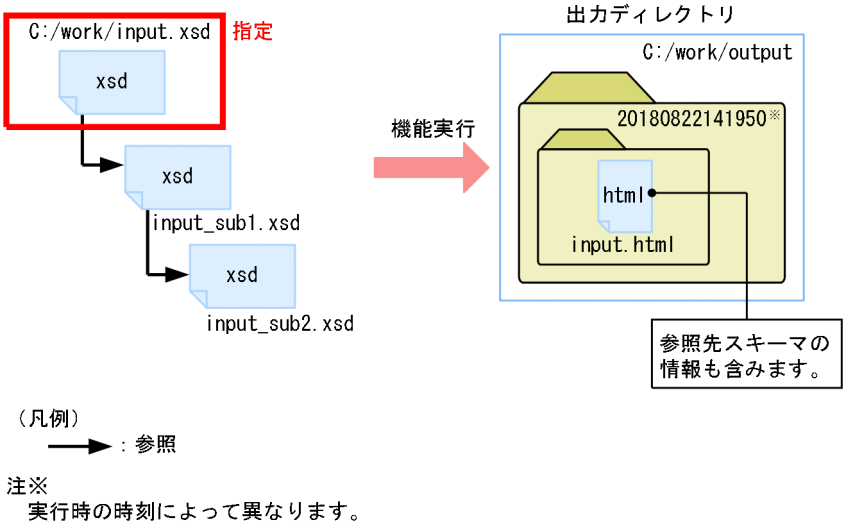
注※1
実行時の時刻によって異なります。
注※2
どのファイルに連番が付与されるかは，処理の実行順に依存します。

(b) 外部の XML スキーマを参照する XML スキーマの出力について

- 外部の XML スキーマを参照する XML スキーマを指定して実行した場合

  指定した XML スキーマに対して処理を実行し，参照先の XML スキーマの情報を含んだ HTML ファイルを出力します。

  外部の XML スキーマを参照する XML スキーマを指定して実行した例を次の図に示します。



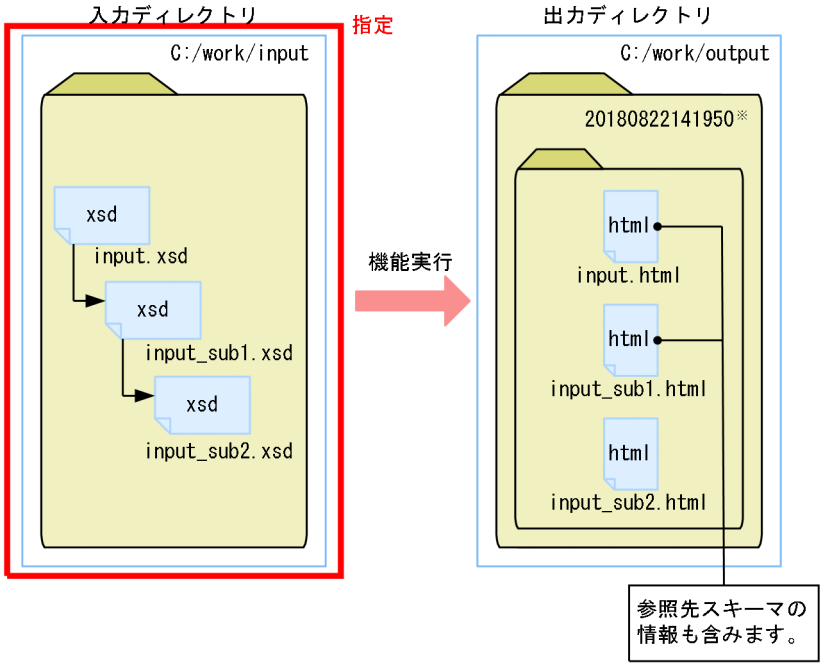
（凡例）
：参照

注※
実行時の時刻によって異なります。

- 外部の XML スキーマを参照する XML スキーマを含むディレクトリを指定して実行した場合

  被参照スキーマについても処理を実施し，出力ディレクトリに出力します。

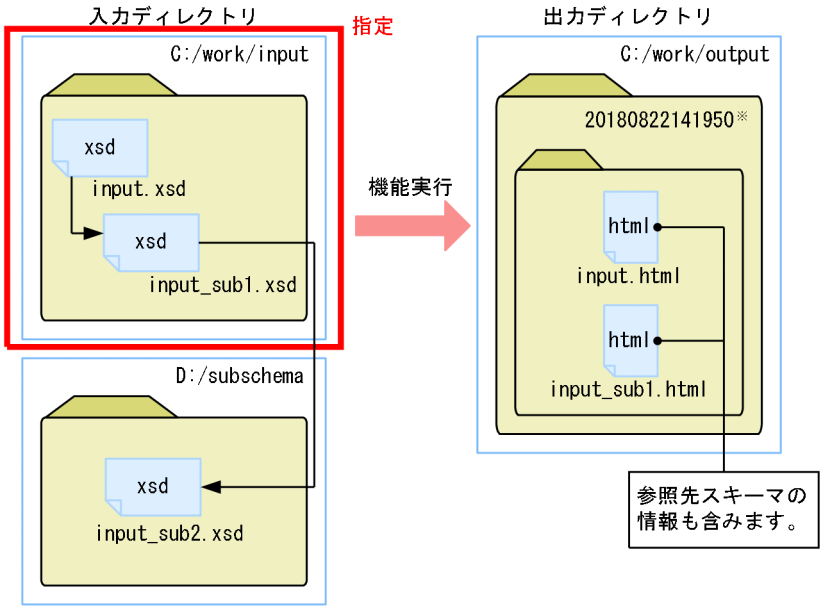
  外部の XML スキーマを参照する XML スキーマを含むディレクトリを指定して実行（全 XML スキーマが入力ディレクトリ内に存在する場合）した例を次の図に示します。



（凡例）
→ ：参照

注※
実行時の時刻によって異なります。

外部の XML スキーマを参照する XML スキーマを含むディレクトリを指定して実行（被参照 XML スキーマが入力ディレクトリ外に存在する場合）した例を次の図に示します。



（凡例）
→ ：参照

注※
実行時の時刻によって異なります。

3.4.5　差分確認用ドキュメント出力機能の注意事項

差分確認用ドキュメント出力機能の注意事項を次に示します。

- 入力するファイルについて

  入力データとして入力するファイルは，機能を実行する前に必ず検証して，検証済みのファイルを入力してください。検証でエラーが発生した場合，定義が不正である旨は通知されますが詳細情報は出力されません。検証でエラーが発生した際は，対象のファイルをエディタで開いて検証を実行し，詳細情報を確認してください。

- 出力される HTML ファイルについて

  出力される HTML ファイルは，Word などの差分確認のツールを使用して，差分を確認することを目的としています。そのため，出力された HTML ファイルをブラウザを使用して確認した場合は，リンクの設定や体裁を整えるための配置制御などは表示されません。

- HTML ファイルに出力される差分情報について

  差分確認用ドキュメント出力機能を使用して確認する差分は，実行環境で動作させた際に影響が出る項目を対象としています。そのため，例えばマッピング定義ファイルにおいて，何もマッピングされていないファンクション情報などは，出力された HTML ファイルには表示されません。

- 差分確認時に使用する HTML ファイルについて

  差分を確認する際に使用する HTML ファイルは，必ず同じバージョンの Service Architect で作成された HTML ファイルを使用してください。バージョンが異なる Service Architect で作成された HTML ファイルを用いて差分を確認した場合，意図しない差分が表示されることがあります。

3.5 Eclipse に関する注意事項

HCSCTE プロジェクト，HCSCTE リポジトリが正しく設定されている状態でも，Eclipse を起動すると，HCSCTE ビューにツリービューが表示されないで，かつエラーダイアログも表示されないことがあります。対処方法を次に示します。

1. いったん，HCSCTE ビューを閉じ，次の方法で HCSCTE ビューを再度開きます。
   - ［ウィンドウ］－［ビューの表示］－［その他］を選択する。
   - 開いた［ビューの表示］ダイアログから，［HCSC-Definer］－［HCSCTE ビュー］を選択し，［OK］ボタンをクリックする。
2. 手順 1 で解決できない場合は，Eclipse を終了し，次の手順を実施します。
   - <Eclipse インストールディレクトリ>¥eclipse.ini に，次の項目を追加する。

     ```
     -Dequinox.statechange.timeout=<timeout>
     ```

     <timeout>の部分に 20000 以上（単位：ミリ秒）のタイムアウト値を設定する。
   - Eclipse を再起動する。

4 電文フォーマットの作成

この章では，電文フォーマットの作成について説明します。

4.1　電文フォーマットとデータ変換

サービスリクエスタ，サービスアダプタ，およびサービス部品の間でやり取りされる電文とその形式（電文フォーマット），およびデータ変換の概要について説明します。

4.1.1　電文の種類

サービスリクエスタからサービスアダプタを経由して，サービス部品の実行を要求するための電文を**要求電文**といいます。

逆に，サービスアダプタを経由して，サービス部品の実行結果をサービスリクエスタに応答するための電文を**応答電文**といいます。

4.1.2　電文フォーマット

サービスリクエスタ，サービスアダプタ，およびサービス部品の間でやり取りされる電文の形式を**電文フォーマット**といいます。

サービスアダプタには，リクエストを受け付けるための標準的な電文フォーマットがあります。これを**標準電文**といいます。また，サービス部品独自の電文のフォーマットを**サービス部品電文**といいます。

電文フォーマットを定義したファイルを，**電文フォーマット定義ファイル**といいます。

電文フォーマット定義ファイルには，扱うデータによって XML フォーマット定義ファイルとバイナリフォーマット定義ファイルの 2 種類があります。電文フォーマット定義ファイルの種類については，「4.2 電文フォーマットの種類」を参照してください。

4.1.3　データ変換

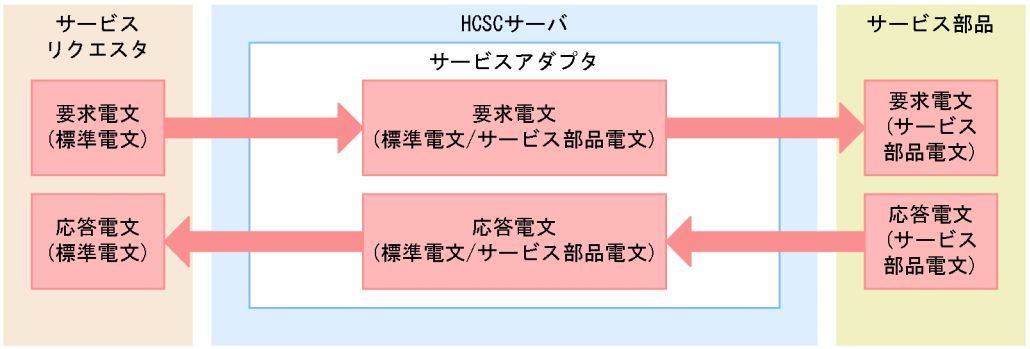
通常は，サービスリクエスタからサービス部品の呼び出し要求があった場合は，標準電文でサービス部品を呼び出せます。しかし，サービスリクエスタから受け付けた要求電文（電文フォーマットは標準電文）でサービス部品を呼び出せない場合，要求電文の電文フォーマットをサービス部品の電文フォーマット（サービス部品電文）に変換する必要があります。この変換を**データ変換**といいます。同様に，サービス部品からの応答電文をサービスリクエスタに返せない場合も，電文フォーマットを変換して標準電文の電文フォーマットに変換する必要があります。

データ変換の定義方法については，「7.　データ変換の定義」を参照してください。

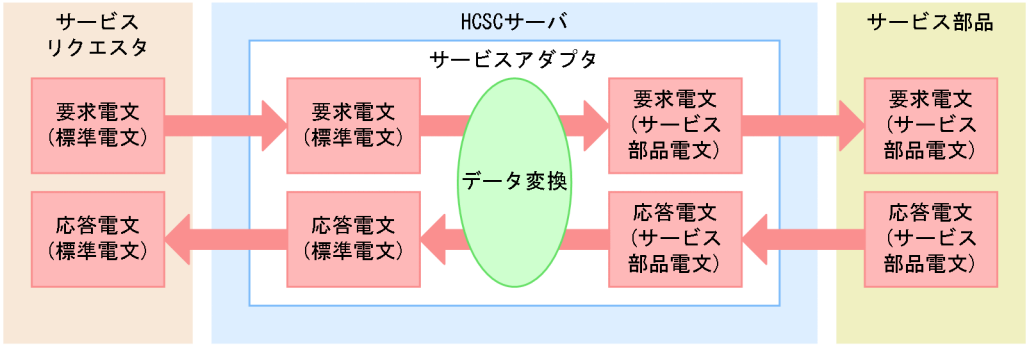
サービスアダプタを利用する場合のサービスリクエスタとサービス部品間の電文の流れを次の図に示します。

図 4-1　サービスリクエスタとサービス部品間の電文の流れ

●標準電文とサービス部品電文の形式が同一の場合（データ変換は不要）



●標準電文とサービス部品電文の形式が異なる場合（データ変換が必要）



（凡例）

：電文（および電文フォーマット）

：電文の流れ

4.2　電文フォーマットの種類

サービス部品の実行に利用する電文で XML 形式のデータを扱うか，バイナリ形式（XML 形式以外）のデータを扱うかによって，作成する電文フォーマットは異なります。

4.2.1　XML 形式のデータを扱う場合

XML 形式のデータを扱う場合，**XML フォーマット定義ファイル**を作成します。

XML フォーマット定義ファイルは，XML スキーマファイル（拡張子：.xsd）として作成します。作成する XML フォーマット定義ファイルの内容や作成方法は，実行するサービス部品の種類によって異なります。サービス部品ごとの XML フォーマット定義ファイルの作成方法については，「4.3　電文フォーマット（XML フォーマット定義ファイル）の作成方法」を参照してください。

4.2.2　バイナリ形式のデータを扱う場合

バイナリ形式（XML 形式以外）のデータを扱う場合，**バイナリフォーマット定義ファイル**を作成します。

バイナリフォーマット定義ファイルでは，テキスト形式，バイナリ形式，および CSV 形式のデータの構造（固定長，セパレータなど）と，データの文字コード（シフト JIS，JIS 漢字，Unicode，KEIS，IBM 漢字，JEF など）を定義できます。

バイナリフォーマット定義ファイルの作成方法については，「4.4　電文フォーマット（バイナリフォーマット定義ファイル）の作成方法」を参照してください。

4.3　電文フォーマット（XML フォーマット定義ファイル）の作成方法

サービス部品の実行に利用する電文で XML 形式のデータを利用する場合の，電文フォーマット（XML フォーマット定義ファイル）の作成方法について，以降の項で説明します。

4.3.1　標準電文を作成する

標準電文の XML フォーマット定義ファイルは，実行するサービス部品の種類に関係なく，Eclipse が提供する WST（Web Standard Tools）などを利用して作成します。

作成する XML フォーマット定義ファイルは，「2.6.5　XML スキーマの適用範囲」に示す条件を満たしている必要があります。条件の詳細については，「2.6.5　XML スキーマの適用範囲」を参照してください。

サービスアダプタおよびビジネスプロセスの定義で，利用する電文フォーマットファイルを選択したときにエラー（メッセージ ID が KECX で始まるエラーメッセージ）が発生した場合，XML スキーマファイルにエラーがあります。この場合，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「11.　KECX（XML Processor が出力するメッセージ)」を参照して対処してください。

4.3.2　サービス部品電文を作成する（Web サービスの場合）

利用するサービス部品が Web サービスの場合，サービス部品電文の XML フォーマット定義ファイルを作成する必要はありません。XML フォーマット定義ファイルは，サービスアダプタの作成時に自動的に作成されます。

ただし，サービスアダプタを作成する前に，「2.6.1　Web サービスを利用したサービス部品の適用範囲」に示す条件を満たしている WSDL を準備する必要があります。

注意事項：SOAP 通信基盤でユーザ定義の例外を throw する Web サービスの場合

SOAP アプリケーション開発支援機能の WSDL 生成機能で生成した WSDL の通信スタイルが rpc 形式の場合，WSDL を修正し，対応するサービスアダプタを作成する必要があります。

生成した WSDL の通信スタイルが document 形式の場合，WSDL を修正する必要はありません。サービスプラットフォームで fault を使用する場合，document 形式の WSDL を使用することを推奨します。

WSDL の修正手順を次に示します。

1. xsd:element を定義します。

   xsd:element 要素は，wsdl:definitions 要素の targetNamespace 属性で示されたネームスペースと同じ targetNamespace 属性を持つ wsdl:types 要素中の xsd:schema 要素内に定義してください。

2. wsdl:portType 要素内の wsdl:operation 要素中の wsdl:fault 要素の message 属性の値から対応する wsdl:message 要素を取得し，内部に含まれる wsdl:part 要素を確認します。

3. 手順 1.で追加した xsd:element 要素に name 属性を追加します。

   値は手順 2.で取得した wsdl:part 要素の type 属性のローカル名と同じ値にします。

4. 手順 3.と同様に，type 属性を追加します。
   値は手順 2.で取得した wsdl:part 要素の type 属性と同じ値にします。
5. 手順 2.で確認した wsdl:part 要素から type 属性を削除し，新たに element 属性を追加します。
   ネームスペースプリフィックスは，手順 1.で参照した wsdl:types 要素内の xsd:schema 要素を示すプリフィックスにします。また，ローカル名は手順 3.で追加した name 属性と同じ値にします。

WSDL を修正したあとには，Web サービスそのものを再作成する必要があります。
次に示すコマンドを実行して，修正した WSDL から Web サービス用のスケルトンを再生成してください。

```
WSDL2Java -s <WSDLファイル>
```

WSDL2Java コマンドの詳細については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」の「4.1.4　スケルトンおよびサービスデプロイ定義を生成する」を参照してください。

4.3.3　サービス部品電文を作成する（SessionBean の場合）

利用するサービス部品が SessionBean の場合，サービス部品電文の XML フォーマット定義ファイルを作成する必要はありません。XML フォーマット定義ファイルは，サービスアダプタの作成時に自動的に作成されます。

4.3.4　サービス部品電文を作成する（MDB（WS-R または DB キュー）の場合）

サービス部品電文の XML フォーマット定義ファイルは，Eclipse が提供する WST（Web Standard Tools）などを利用して作成します。

作成する XML フォーマット定義ファイルは，「2.6.5　XML スキーマの適用範囲」に示す条件を満たしている必要があります。

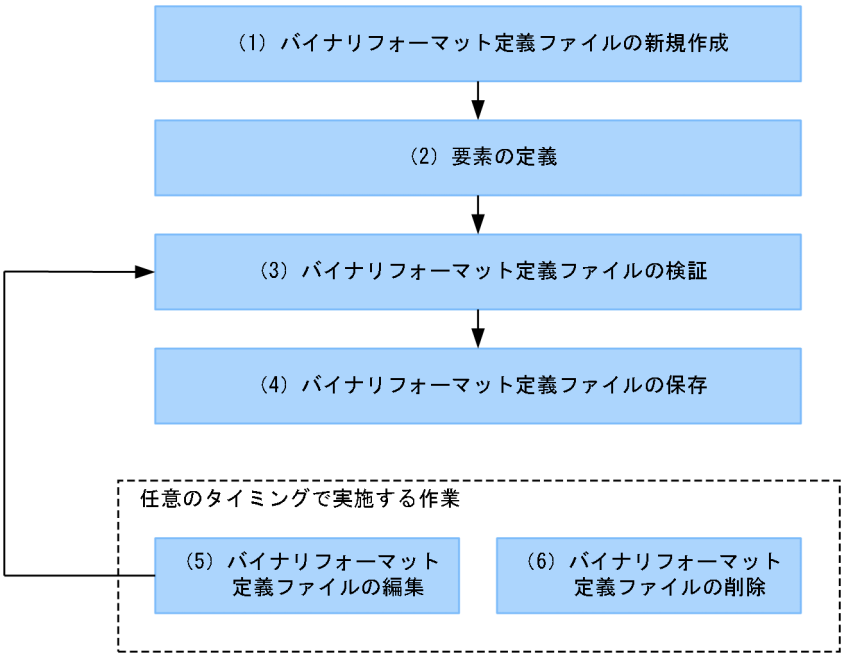
4.4　電文フォーマット（バイナリフォーマット定義ファイル）の作成方法

サービス部品の実行に利用する電文でバイナリ形式（XML 形式以外）のデータを利用する場合，バイナリフォーマット定義ファイルを作成します。

バイナリフォーマット定義ファイルで使用するデータ型と文字コードの種類については，「4.4.1　バイナリフォーマット定義ファイルのデータ型と文字コードの種類」を参照してください。

バイナリフォーマット定義ファイルの作成の流れを次の図に示します。

図 4-2　バイナリフォーマット定義ファイルの作成の流れ



(1) バイナリフォーマット定義ファイルの新規作成

ウィザードを利用してバイナリフォーマット定義ファイルを新規作成します。

バイナリフォーマット定義ファイルの新規作成の方法については，「4.4.3　バイナリフォーマット定義ファイルを新規作成する」を参照してください。

(2) 要素の定義

新規作成したバイナリフォーマット定義ファイルに要素を定義します。要素の定義の方法は，利用するバイナリデータが CSV 以外の形式の場合と，CSV 形式の場合とで異なります。

要素の定義方法については，「4.4.4　要素を定義する（CSV 以外の形式の場合）」または「4.4.5　要素を定義する（CSV 形式の場合）」を参照してください。

(3) バイナリフォーマット定義ファイルの検証

作成したバイナリフォーマット定義ファイルの整合性を検証します。バイナリフォーマット定義ファイルの要素の定義中や定義終了後など，任意のタイミングで検証できます。

バイナリフォーマット定義ファイルの検証方法については，「4.4.7　バイナリフォーマット定義ファイルを検証する」を参照してください。

(4) バイナリフォーマット定義ファイルの保存

要素を定義したバイナリフォーマット定義ファイルを保存します。バイナリフォーマット定義ファイルは，サービスアダプタやビジネスプロセスと異なり，リポジトリに保存されません。定義したバイナリフォーマット定義ファイルの保存には，Eclipse のパッケージ・エクスプローラーを利用してください。

バイナリフォーマット定義ファイルを保存するときには，自動的に検証が行われます（この場合，検証結果はコンソールビューに表示されません）。

(5) バイナリフォーマット定義ファイルの編集

すでに作成したバイナリフォーマット定義ファイルは，必要に応じて，定義内容を変更できます。バイナリフォーマット定義ファイルの編集方法については，「4.4.6　バイナリフォーマット定義ファイルを編集する」を参照してください。

(6) バイナリフォーマット定義ファイルの削除

不要になったバイナリフォーマット定義ファイルは，必要に応じて削除できます。バイナリフォーマット定義ファイルの削除には，Eclipse のパッケージ・エクスプローラーを利用してください。

4.4.1　バイナリフォーマット定義ファイルのデータ型と文字コードの種類

バイナリフォーマット定義エディタでは，次に示すデータ型を使用してフォーマットを定義できます。

- 数値
- 文字列
- バイト列
- ビット列
- 日付時刻

各データ型は，データ変換機能によって，バイナリ型から XML 型（DOM），または XML 型（DOM）からバイナリ型へ変換されます。このバイナリ型と中間形式との型変換処理をバイナリパース処理と呼びます。データ変換機能の変換処理については，マニュアル「サービスプラットフォーム 解説」の「付録 B データ変換機能の処理概要」を参照してください。

各データで扱う文字コードの種類については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.1　フォーマットダイアログ」を参照してください。

バイナリデータ内で文字コードが混在する場合の対応については，「付録 P　バイナリデータ内に文字コードが混在する場合の対応」を参照してください。

次に各データ型の内容について説明します。

(1)　数値

数値型には，数値属性によって文字列型数値とバイト列型数値に分けられます。それぞれの数値型に属する数値属性を次に示します。

●文字列型数値

- 整数
- 実数
- 固定小数部数値

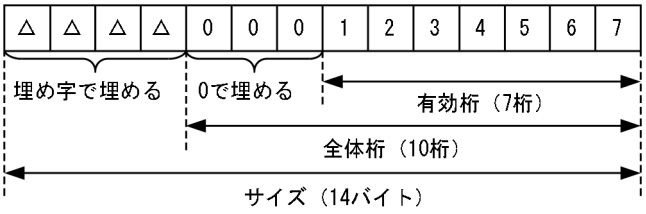
バイナリ型から XML 型（DOM）への変換仕様を説明します。

- 文字列型数値は，先頭埋め字（右寄せの場合），末尾埋め字（左寄せの場合），先頭 0，小数点以下の末尾 0 を除く有効桁と符号を，XML スキーマ decimal 型に変換する。
- 実数の場合，整数部が存在しない数値（先頭が小数点）は整数部に 0 を設定して変換する。また，小数点が存在するが小数部が存在しない場合（小数部がすべて 0 の場合も含む）は，小数点を切り捨てる。
- サイズノード指定によって 0 バイトとなるデータは変換できない。
- 埋め字（スペース）だけで埋められたデータは変換できない。

XML 型（DOM）からバイナリ型への変換仕様を説明します。

- 数値前後の空白（半角スペース，タブ，および改行）を除いたデータを変換対象とする。
- 数値の前の 0 は常に削除する。
- 不要文字は数値の前の 0 を表す。
- 実数は，小数部の桁数が小数部桁数の設定値より小さい場合，小数部の下位桁を 0 で埋める。
- 固定小数部数値は，有効桁が小数部桁数の設定値より小さい場合，有効桁の右側を 0 で埋める。
- 有効桁が全体桁数の設定値より小さい場合，有効桁の左側を 0 で埋める。
- 全体桁数の指定が有効になるのは，不要文字を削除しない設定の場合だけである。
- サイズが 0 のデータ（空ノードや空白だけのデータ）は，0 として扱う。
- サイズが固定長かつ変換後のデータ長が指定されたサイズに満たない場合，左右寄せの指定によって末尾または先頭を埋め字で埋める。

図 4-3　全体桁，サイズ，埋め字の関係（右寄せの例）



サイズ　：固定長（14バイト）
全体桁数：10桁
左右寄せ：右
不要文字：削除しない
埋め字　：半角スペース（△は半角スペースを表す）

図 4-4　全体桁，サイズ，埋め字の関係（左寄せの例）



サイズ　：固定長（14バイト）
全体桁数：10桁
左右寄せ：左
不要文字：削除しない
埋め字　：半角スペース（△は半角スペースを表す）

文字列型数値の符号は省略できます。符号の出現位置は数字列の前です。

●バイト列型数値

- ゾーン形式数値
- パック形式数値
- 符号付 2 進整数
- 符号無 2 進整数

バイナリ型から XML 型（DOM）への変換仕様を説明します。

- ゾーン形式またはパック形式数値は，先頭 0，小数点以下の末尾 0 を除く有効桁と符号を XML スキーマ decimal 形式に変換する。
- ゾーン形式またはパック形式数値に小数桁が指定され，整数部がない場合は”0.”を補う。
- サイズノード指定により 0 バイトとなるデータは変換できない。
- 符号付 2 進数または符号無 2 進整数は，XML スキーマ integer 形式に変換する。

XML 型（DOM）からバイナリ型への変換仕様を説明します。

数値前後の空白（半角スペース，タブ，改行）を除いたデータを変換対象とする。

- サイズが 0 のデータ（空ノードや空白だけのデータ）は，0 として扱う。
- ゾーン形式またはパック形式数値は，先頭 0 を削除した有効桁を右寄せで埋める。
- ゾーン形式またはパック形式数値に小数部桁数が指定されている場合，整数部の”0.”は変換しない。また，小数部が指定された小数部桁数に満たない場合は右側を 0 で埋める。
- ゾーン形式またはパック形式数値のサイズが固定長かつ，変換後のデータ長が指定されたサイズに満たない場合（左側が空いた場合）は 0 で埋める。

どの数値属性もデータ変換では 10 進数として扱います。扱える 10 進数としての最大桁（整数部と小数部の桁数の合計）は，整数部先頭の 0，小数部末尾の 0，符号，小数点を削除し，有効桁だけの状態で 34 桁です。

それぞれの数値属性について説明します。

(a) 整数

「数字」および「符号」で構成される数値属性です。この数値属性は，内部的に BigInteger クラスで処理されます。

整数は最大 34 桁で小数部を持ちません。

整数の例を次に示します。

(例 1)

1234567890

(例 2)

-1234567890

(b) 実数

「数字」,「符号」および「小数点」から構成される一般的な形式の数値属性です。この数値属性は内部的に BigDecimal クラスで処理されます。

実数は最大 34 桁で小数部を持ちます。ただし，入力データ中に数字がなく，小数点だけの場合はエラーになります。

実数の例を次に示します。

(例 1)

1234567890

(例 2)

-123.456

(c) 固定小数部数値

「符号」および「数字」から構成され，小数部の桁を指定すると小数点が暗黙的に設定される数値属性です。この数値属性は内部的に BigDecimal クラスで処理されます。

固定小数部数値は最大 34 桁で，小数部は最大 33 桁です。

小数部を 4 桁に指定した場合の固定小数部数値の例を示します。

(例 1)「1234567890」の場合

123456.789

(1234567890 → 123456.7890 → 123456.789)

(例 2)「+123」の場合

0.0123

（+123 → +0.0123 → 0.0123）

(例 3)「-1230000」の場合

-123

（-1230000 → -123.0000 → -123）

(d) ゾーン形式数値

ゾーン形式の数値属性です。小数部の桁を指定すると小数点が暗黙的に設定されます。この数値属性は内部的に BigDecimal クラスで処理されます。

ゾーン形式数値は最大 34 桁で，符号は項目の最終バイトのゾーン部で表します。

文字コードが JIS8 の場合，正符号は 0x3，負符号は 0x7 で表します。

文字コードが EBCDIK の場合，正符号は 0xC または 0xF，負符号は 0xD で表します。

符号の正負は変更できます。

最終バイト以外のゾーン部は各コードの正符号で表し，数値は各バイトの下位 4 ビットで表します（0x0～0x9）。

文字コードが JIS8 の場合のゾーン形式数値の例を示します。

(例 1)「0x31323334」（小数部の指定なし）の場合

1234

(例 2)「0x31323334」（小数部を 2 桁に指定）の場合

12.34

(例 3)「0x31323374」（小数部の指定なし）の場合

-1234

(e) パック形式数値

パック形式の数値属性です。小数部の桁を指定すると小数点が暗黙的に設定されます。この数値属性は内部的に BigDecimal クラスで処理されます。

パック形式数値は最大 34 桁で，桁数が偶数の場合，最上位バイトの上 4 ビットは 0 にしなければなりません。

符号は項目データの最終バイトの下位 4 ビットで表します。正符号は 0xC または 0xF，負符号は 0xD で表し，数値は 4 ビットで表します（0x0～0x9）。符号の正負は変更できます。なお，データ変換は正符号として 0xC を出力します。

文字コードが JIS8 の場合のパック形式数値の例を示します。

(例 1)「0x01234F」（小数部の指定なし）の場合

1234

(例 2)「0x01234F」（小数部を 2 桁に指定）の場合

12.34

(例 3)「0x01234D」（小数部の指定なし）の場合

-1234

(f)　符号付 2 進整数

1～8 バイトまでの符号付きの 2 進数で構成される数値属性です。この数値属性は内部的に BigInteger クラスで処理されます。

エンディアンはフォーマットの情報に従います。

ビッグエンディアンの場合の符号付 2 進整数の例を示します。

(例 1)「256」の場合

0x0100

(例 2)「-257」の場合

0xFEFF

(g)　符号無 2 進整数

1～8 バイトまでの符号なしの 2 進数で構成される数値属性です。この数値属性は内部的に BigInteger クラスで処理されます。

エンディアンはフォーマットの情報に従います。

ビッグエンディアンの場合の符号無 2 進整数の例を示します。

(例 1)「256」の場合

0x0100

(例 2)「65279」の場合

0xFEFF

(2)　文字列

バイナリ型から XML 型（DOM）への変換仕様を説明します。

- 指定されたサイズの文字列を，XML スキーマ string 型として文字列に変換する。
- 埋め字にスペースが指定されていて，かつ不要文字を削除しない設定の場合，スペースの付いた入力文字列が抽出される。

- 埋め字にスペースが指定されていて，かつ不要文字を削除する設定の場合，左右寄せの反対側のスペースを削除する。
- 埋め字に空文字（0x00）が指定されている場合，次の仕様で変換する。
  - 空文字（0x00）までをデータとする設定の場合，文字列の先頭から空文字の直前の文字までをユーザデータとして，XML の文字列に変換する。
  - 空文字（0x00）までをデータとしない設定で，かつ不要文字を削除する設定の場合，左右寄せの反対側の空文字を削除する。
  - 空文字（0x00）までをデータとしない設定で，かつ不要文字を削除しない設定の場合，空文字の付いた入力文字列が抽出される。
- XML で使用できない文字（使用不可文字）を代替文字で置き換えるように設定した場合，XML 上の使用不可文字を通常の文字として扱う。
- サイズノード指定によって 0 バイトとなるデータは変換できない。

XML 型（DOM）からバイナリ型への変換仕様を説明します。

- XML データを指定された文字コードで変換する。
- サイズが 0 のデータ（空ノード）は，空文字列として扱う。
- 固定長の場合，終端の連続する半角スペースを除去して変換する。変換後のデータ長が指定されたサイズに満たない場合は，埋め字で埋める。
- 埋め字に空文字（0x00）が指定されて，かつ空文字（0x00）までをデータとする設定の場合，入力文字列の末尾を空文字で埋める。
- 不要文字を削除する設定の場合，入力文字列の末尾の半角スペースを削除する。
- 不要文字を削除しない設定の場合，入力文字列から半角スペースや空文字を削除しない。

(3) バイト列

バイト列型は，プラットフォーム間でビットパターンが保存されるバイト列そのものです。バイト列をバイナリパース処理すると，XML スキーマの hexBinary 型または base64Binary 型に変換されます。

hexBinary 型では，8 ビット（1 バイト）のバイナリデータの場合，16 進数 2 桁の文字に変換します。例えば，300 バイトの元データを hexBinary 型で表現するには，300 バイト×2 文字（600 文字）が必要になります。

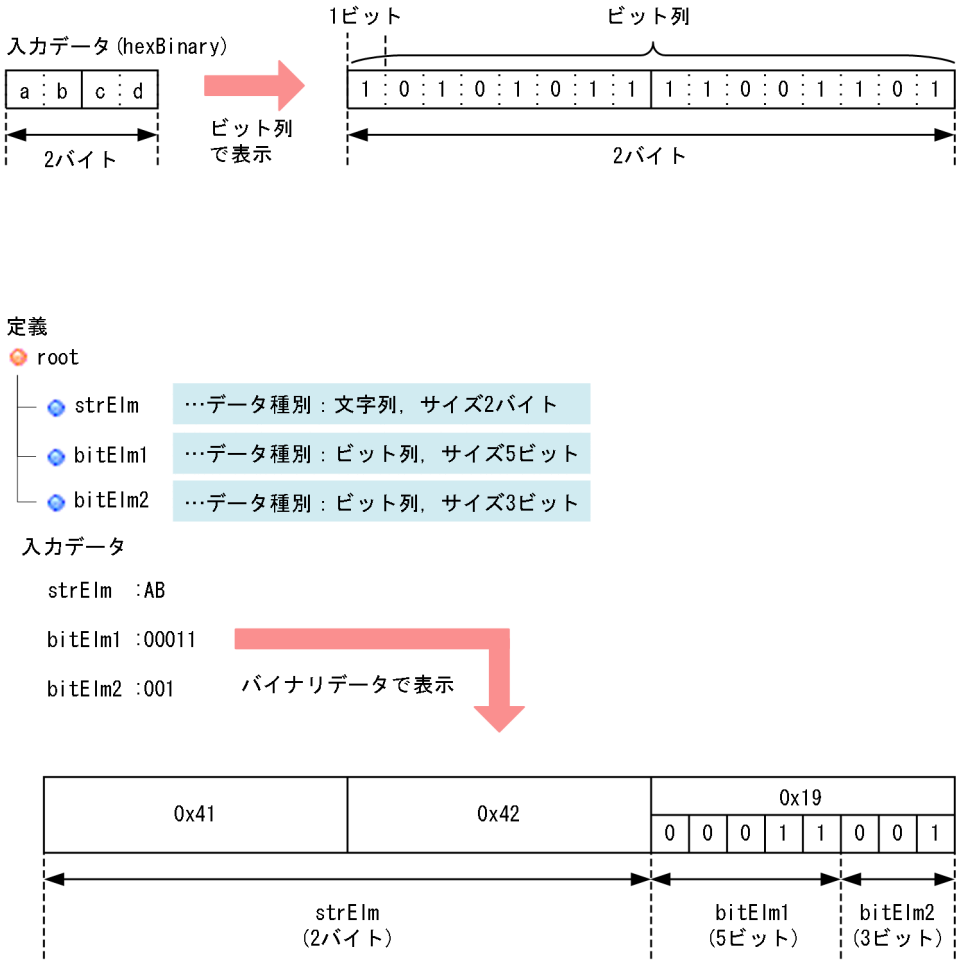
base64Binary 型では，6 ビット（6/8 バイト）のバイナリデータの場合，Base64 エンコードされた 1 桁の文字に変換します。例えば，300 バイトの元データを base64Binary 型で表現するには，300×8/6 文字（400 文字）が必要になります。

バイト列をバイナリパース処理する場合，サイズノード指定によって 0 バイトとなるデータは変換できません。また，XML 型（DOM）をバイナリパース処理する場合，前後の空白（半角スペース，タブ，および改行）を除いたデータが対象となります。サイズが 0 のデータ（空ノードや空白だけのデータ）は，0x00 として扱います。

(4) ビット列

ビット列とは，2 進数で表されるビットが並んだものです。ビット列型とは，2 進数のデータを 1 桁単位で扱うことのできるデータ型です。ビット列型はビッグエンディアンで処理されます。

図 4-5　ビット列およびビット列型のイメージ



バイナリ型から XML 型（DOM）には，次のとおり変換します。

- 指定されたビットサイズ分，入力バイナリの先頭から読み，8 ビット単位で XML スキーマの hexBinary 形式に変換します。
- すべての要素の読み込みが完了したあとに 1 バイト未満のデータが残っていた場合，残ったデータを桁合わせビットと見なし，切り捨てます。1 バイト以上のデータが残っていた場合はエラーになります。
- サイズノード指定によって 0 バイトとなるデータは変換できません。

XML 型（DOM）からバイナリ型には，次のとおり変換します。

- 前後の空白（半角スペース，タブ，および改行）を除いたデータが変換対象となります。
- hexBinary 形式のデータを下位ビットから指定ビットサイズ分をバイナリデータに変換します。

- XML データが変換先ビット数より大きい場合，変換先ビット数から上位ビットを切り捨て，エラーにしません。
- XML データが変換先ビット数より小さい場合，変換先ビット数までのビットを 0 で補完します。
- 指定されたビットサイズ分の XML データが 16 進文字列でない場合はエラーになります。
- サイズが 0 のデータ（空ノードや空白だけのデータ）は，0x00 として扱います。

(5) 日付時刻

日付時刻型は，年，月，日，時，分，秒のパートの組み合わせで構成する型です。個々のパートは数値を指定でき，秒を表すパートは小数部を持つことができます。

日付時刻型を XML 型（DOM）からバイナリ型の固定長要素に変換する場合，入力 XML の変換後のサイズが固定長要素のサイズより小さいときに，要素データを左右どちら側に寄せて表現するか（左右寄せ）を指定できます。

日付時刻型を XML 型（DOM）からバイナリ型の固定長要素に変換する場合，前後の空白（半角スペース，タブ，および改行）を除いたデータが変換対象となります。入力 XML の変換後のサイズが固定長要素のサイズより小さいときに，要素データを左右どちら側に寄せて表現するか（左右寄せ）を指定できます。

サイズが 0 のデータ（空ノードや空白だけのデータ）の場合，形式ごとにデータの扱いが異なります。

サイズが 0 のデータ（空ノードや空白だけのデータ）の場合の日付時刻の例を示します。

(例 1)「YYMMDD」形式の場合
000101

(例 2)「CCYYMMDD」形式の場合
20000101

(例 3)「hhmmss」形式の場合
000000

(例 4)「YYMMDDhhmmss」形式の場合
000101000000

(例 5)「CCYYMMDDhhmmss」形式の場合
20000101000000

日付時刻型を指定する場合，次の制限事項があります。

- 2 バイト文字コードは使用できません。
- 小数点として認識する文字は「.」だけです。

日付時刻型のパートを次の表に示します。

表 4-1　日付時刻型のパート

パート	意味	指定する桁数
CCYY	西暦年（0～9999）	1～4 桁
YY	年（西暦年下 2 桁）	最大 2 桁
MM	月（1～12）	最大 2 桁
DD	日（1～31）	最大 2 桁
hh	時（0～23）	最大 2 桁
mm	分（0～59）	最大 2 桁
ss	秒（0～59，小数部可）	整数部最大 2 桁，小数部最大 3 桁

パートの組み合わせによって，日付時刻型は 3 つのデータ型（日付型，時刻型，および日付時刻型）を表現することができます。

データ型とパートの組み合わせを次の表に示します。

表 4-2　データ型とパートの組み合わせ

データ型	組み合わせ	意味
日付型	CCYYMMDD	西暦年/月/日
	YYMMDD※	西暦年下 2 桁/月/日
時刻型	hhmmss	時/分/秒
日付時刻型	CCYYMMDDhhmmss	西暦年/月/日/時/分/秒
	YYMMDDhhmmss	西暦年下 2 桁/月/日/時/分/秒

注※

日付型の「YYMMDD」形式の場合，指定できる範囲は西暦年換算で 1951 年～2050 年になります。

日付型，時刻型，および日付時刻型を XML 変換すると，それぞれ次のように変換されます。

- 日付型は，XML スキーマの date 型に変換されます。
- 時刻型は，XML スキーマの time 型に変換されます。
- 日付時刻型は，XML スキーマの dateTime 型に変換されます。

日付時刻型の例を次に示します。

(例 1)「2001 年 1 月 24 日」の場合

20010124

(例 2)「1999 年 12 月 31 日 1 時 30 分 59 秒」の場合

991231013059

4.4.2　文字コード変換の種類

サービスプラットフォームの文字コード変換は，Code Converter を利用しています。この Code Converter の機能を利用すると，未定義コード検出時の動作を選択できます。

また，単体版の Code Converter を導入することで，ユーザマッピングテーブルを作成し，コード変換テーブルの変更ができます。これによって，KEIS や JEF といったホスト系文字コードを扱うこともできます。

コード変換処理で，Code Converter の機能を利用しない場合は，Java 提供のエンコーダー・デコーダーを使用することもできます。これによって，コード変換処理の性能が向上することがあります。

4.4.3　バイナリフォーマット定義ファイルを新規作成する

バイナリフォーマット定義ファイルを新規作成する方法を次に示します。

なお，バイナリフォーマット定義ファイルを作成する前には，プロジェクトを作成しておく必要があります。

1. Eclipse のメニューから，［ファイル］ − ［新規］ − ［その他］を選択します。
   ウィザードを選択するダイアログが表示されます。

2. ［HCSCTE フォーマット定義］の［バイナリフォーマット定義ファイル］を選択して，［次へ］ボタンをクリックします。
   ［新規バイナリフォーマット定義ファイル］ウィザードが表示されます。

3. ［親フォルダーを入力または選択］で，バイナリフォーマット定義ファイルを保存するフォルダを入力または選択します。

4. 保存するバイナリフォーマット定義ファイルのフォーマット名を［フォーマット名］に指定します。
   フォーマット名を指定すると，指定したフォーマット名に拡張子「.fdx」を付けた名称が［ファイル名］に自動設定されます。

5. 利用するバイナリデータが CSV 形式の場合，［CSV 形式］チェックボックスをチェックします。
   なお，CSV 形式のデータとして扱えるのは，次の区切り文字を使用しているデータです。
   - 中間区切り文字：コンマ（,）
   - 終了文字：改行

   要素の値に中間区切り文字と同じ文字を含む場合は，その要素全体をダブルクォーテーション（"）で囲みます。なお，要素内でダブルクォーテーションを値として使用する場合，ダブルクォーテーションを 2 つ続けてエスケープしてください。
   なお，開始文字を利用している場合は，CSV 形式のデータとして扱えないので注意してください。

6. ［終了］ボタンをクリックします。

新しいバイナリフォーマット定義ファイルが作成され，バイナリフォーマット定義画面が表示されます。バイナリフォーマット定義画面で，続けて要素を定義できます。

注意事項

バイナリフォーマット定義ファイルは，サービスアダプタやビジネスプロセスと異なり，リポジトリに保存されません。定義したバイナリフォーマット定義ファイルの保存や削除などのファイル管理は，Eclipse のパッケージ・エクスプローラーを利用してください。

4.4.4　要素を定義する（CSV 以外の形式の場合）

バイナリフォーマット定義ファイルの内容はバイナリフォーマット定義画面で定義します。バイナリフォーマット定義画面は，Eclipse のパッケージ・エクスプローラーに表示されるバイナリフォーマット定義ファイルをダブルクリックすると表示されます。

バイナリフォーマット定義画面については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.1　バイナリフォーマット定義画面」を参照してください。

参考

バイナリフォーマット定義画面では，複数のバイナリフォーマット定義ファイルを開いて編集できます。ただし，複数のバイナリフォーマット定義ファイルの間で，定義情報をコピーして貼り付けることはできません。

バイナリフォーマット定義画面では，要素の定義として次の作業を行います。

- フォーマット情報の設定
- グローバル定義の単純内容要素の設定
- 複合内容要素の設定
- 構成要素の設定
- 構成要素の選択条件の設定
- ルート要素の設定
- セパレータの設定

定義の方法を次に示します。

(1)　フォーマット情報を設定する

バイナリフォーマット定義ファイルのフォーマット情報として，バイナリデータで使用されている文字コードや，バイナリデータに含まれる 2 進整数のエンディアンを設定します。

フォーマット情報の設定方法を次に示します。

1. バイナリフォーマット定義エディタに表示されているフォーマット名を選択して右クリックし，［設定］を選択します。
   ［フォーマット］ダイアログが表示されます。

2. ［フォーマット］ダイアログに必要な情報を入力します。
   ［フォーマット］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.1　フォーマットダイアログ」を参照してください。

3. ［OK］ボタンをクリックします。
   ［フォーマット］ダイアログに入力した情報が設定されます。

(2)　グローバル定義の単純内容要素を設定する

単純内容要素とは，使用するバイナリデータの形式を XML スキーマで表現したときに，単純型の要素（xsd:simpleType 要素）であり，かつグローバル定義の要素として表現できるものです。バイナリフォーマット定義ファイルをデータ変換定義画面で表示した場合，単純内容要素は，単純内容のノードとして表示されます。

単純内容要素の設定では，要素のデータ種別やサイズを設定します。

単純内容要素の設定方法を次に示します。

1. バイナリフォーマット定義エディタに表示されているフォーマット名を選択して右クリックし，［単純内容要素の追加］を選択します。
   ［単純内容要素］ダイアログが表示されます。

2. ［単純内容要素］ダイアログに必要な情報を入力します。
   ［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

3. ［OK］ボタンをクリックします。
   ［単純内容要素］ダイアログに入力した情報で単純内容要素が設定されます。また，設定された単純内容要素がバイナリフォーマット定義エディタに表示されます。

(3)　複合内容要素を設定する

複合内容要素とは，使用するバイナリデータの形式を XML スキーマで表現したときに，複合型の要素（xsd:complexType 要素）であり，かつグローバル定義の要素として表現できるものです。バイナリフォーマット定義ファイルをデータ変換定義画面で表示した場合，複合内容のノードとして表示されます。

複合内容要素の設定では，複合内容要素を構成する要素の構成区分などを設定します。構成区分には，複合内容要素を構成する要素が逐次構成（xsd:sequence 要素）か，選択構成（xsd:choice 要素）かを設定します。

複合内容要素の設定方法を次に示します。

1. バイナリフォーマット定義エディタに表示されているフォーマット名を選択して右クリックし，［複合内容要素の追加］を選択します。

   ［複合内容要素］ダイアログが表示されます。

2. ［複合内容要素］ダイアログに必要な情報を入力します。

   ［複合内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.3　複合内容要素ダイアログ」を参照してください。

   構成区分として［選択構成］を選択した場合

   構成区分として［選択構成］を選択した場合，次のどちらかの方法で構成要素に選択条件を設定する必要があります。

   ・［選択条件ノード］を指定し，選択条件の判定値を含むノードを選択する

   ・［構成要素の開始セパレータ］を指定する

   選択条件を設定する方法については，「(5)　構成要素の選択条件を設定する」を参照してください。

   なお，構成要素に選択条件を設定する前に，複合内容要素を構成する要素を設定する必要があります。必ず次の手順 3.を行ってから，「(4)　構成要素を設定する」に示す手順に従って構成要素を設定し，そのあとで選択条件を設定してください。

   複合内容要素の構成要素である単純内容要素がセパレータデータの場合

   複合内容要素の構成要素である単純内容要素が，要素ごとに区切り文字（セパレータ）で区切られたデータ（セパレータデータ）の場合，複合内容要素にセパレータを設定します。

   セパレータの設定方法については，「(7)　セパレータを設定する」を参照してください。

3. ［OK］ボタンをクリックします。

   ［複合内容要素］ダイアログに入力した情報で複合内容要素が設定されます。また，設定された複合内容要素がバイナリフォーマット定義エディタに表示されます。

> **参考**
>
> バイナリフォーマット定義ファイルの要素の定義時に，最初に設定した複合内容要素は，ルート要素になります。ルート要素を変更する場合には，「(6)　ルート要素を設定する」を参照してください。

(4)　構成要素を設定する

構成要素とは，複合内容要素を構成する要素です。構成要素を設定する方法には次の 2 つの方法があります。

- グローバル定義の要素を構成要素に設定する
- ローカル定義の単純内容要素を設定する

それぞれの方法での構成要素の設定について次に示します。

(a) グローバル定義の要素を構成要素に設定する

すでに設定されているグローバルな要素（単純内容要素および複合内容要素）を，複合内容要素の構成要素に設定します。

設定方法を次に示します。

1. バイナリフォーマット定義エディタに表示されている複合内容要素を選択して右クリックし，［構成要素の追加］を選択します。

   ［要素選択］ダイアログが表示されます。［要素選択］ダイアログには，複合内容要素の構成要素として選択できるグローバル定義の要素がすべて表示されます。

   ［要素選択］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.7　要素選択ダイアログ」を参照してください。

2. ［要素選択］ダイアログの要素一覧で構成要素として設定する要素を選択します。

3. ［OK］ボタンをクリックします。

   複合要素内に構成要素が追加されて［要素選択］ダイアログが閉じます。

4. 追加された構成要素を右クリックして，［設定］を選択します。

   ［構成要素］ダイアログが表示されます。

5. ［構成要素］ダイアログに必要な情報を入力します。

   ［構成要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.4　構成要素ダイアログ」を参照してください。

6. ［OK］ボタンをクリックします。

   ［構成要素］ダイアログに入力した情報で，グローバル定義の要素が複合内容要素の構成要素として設定されます。設定した構成要素は，バイナリフォーマット定義エディタ上の複合内容要素の下位要素として表示されます。

(b) ローカル定義の単純内容要素を構成要素として設定する

ローカル定義の単純内容要素を複合内容要素の構成要素として設定します。

設定方法を次に示します。

1. バイナリフォーマット定義エディタに表示されている複合内容要素を選択して右クリックし，［ローカル単純内容要素の追加］を選択します。

   ［単純内容要素］ダイアログが表示されます。

2. ［単純内容要素］タブを選択し，必要な情報を入力します。

   ［単純内容要素］ダイアログの［単純内容要素］タブの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

3. ［構成要素］タブを選択し，必要な情報を入力します。

［構成要素］タブでは，構成要素の出現回数およびサイズを設定します。

［単純内容要素］ダイアログの［単純内容要素］タブの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

なお，出現回数およびサイズは，ほかの要素に依存させることができます。この場合，［ノード選択］ボタンをクリックして表示される［ノード選択］ダイアログで，依存させるノードを選択できます。

［ノード選択］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.6　ノード選択ダイアログ（バイナリフォーマット定義）」を参照してください。

［ノード選択］ダイアログで［OK］ボタンをクリックすると，設定した値が［単純内容要素］ダイアログの［構成要素］タブに反映されます。

4. ［OK］ボタンをクリックします。

［単純内容要素］ダイアログに入力した情報で，ローカル定義の単純内容要素が複合内容要素の構成要素として設定されます。設定した構成要素は，バイナリフォーマット定義エディタ上の複合内容要素の下位要素として表示されます。

(5)　構成要素の選択条件を設定する

複合内容要素を設定する際，構成区分として［選択構成］を選択した場合，構成要素に選択条件を設定する必要があります。選択条件を設定すると，複合内容要素に含まれる複数の構成要素のうち，どの要素を利用するかを別の要素の値から判定できます。

選択条件を設定する方法には，次の 2 つの方法があります。

- ［選択条件ノード］を選択して，選択条件を設定する
  選択条件ノードで選択したノードに設定した条件値から，利用する構成要素を決定します。
- ［構成要素の開始セパレータ］を選択して，選択条件を設定する
  構成要素に設定した開始セパレータから，利用する構成要素を決定します。

それぞれの方法での選択条件の設定について次に示します。

(a)　選択条件ノードを選択して選択条件を設定する

選択条件ノードを選択して選択条件を設定する方法を次に示します。

1. 選択条件を設定する構成要素を含む複合内容要素をバイナリフォーマット定義エディタ上で選択して右クリックし，［設定］を選択します。

［複合内容要素］ダイアログが表示されます。

2. ［構成区分］で［選択構成］を選択します。

3. ［選択種別］で［選択条件ノード］を選択し，［ノード選択］ボタンをクリックします。

［ノード選択］ダイアログが表示されます。［ノード選択］ダイアログでは，選択条件ノードを選択します。

［ノード選択］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.6　ノード選択ダイアログ（バイナリフォーマット定義）」を参照してください。

［ノード選択］ダイアログで［OK］ボタンをクリックすると，設定した値が［複合内容要素］ダイアログの［選択条件ノード］に反映されます。

4. ［構成要素一覧］から条件を設定する要素を選択し，［条件値設定］ボタンをクリックします。

［条件値設定］ダイアログが表示されます。1 つの要素に対して 1 つの選択条件値を指定する場合は［単一条件指定］を，1 つの要素に対して複数の選択条件値を指定する場合は［複数条件指定］を選択して条件値を入力します。

［条件値設定］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.5　条件値設定ダイアログ」を参照してください。

［条件値設定］ダイアログで［OK］ボタンをクリックすると，設定した値が［複合内容要素］ダイアログの［選択条件値］に反映されます。選択条件値が重複している場合は，先に定義されている要素が優先されます。ほかの構成要素にも選択条件を設定する場合，手順 3.と手順 4.を繰り返します。

なお，選択条件が設定されていない構成要素は，ほかの構成要素に設定されているどの選択条件にも当てはまらない場合に出現する構成要素になります。このような構成要素は，構成要素の中で 1 つだけ設定できます。

5. ［OK］ボタンをクリックします。

［複合内容要素］ダイアログに入力した情報で，選択条件が設定されます。

(b)　構成要素の開始セパレータを選択して選択条件を設定する

構成要素の開始セパレータを選択して選択条件を設定する方法を次に示します。

1. 選択条件を設定する構成要素を含む複合内容要素をバイナリフォーマット定義エディタ上で選択して右クリックし，［設定］を選択します。

［複合内容要素］ダイアログが表示されます。

2. ［構成区分］で［選択構成］を選択します。

3. ［選択種別］で［構成要素の開始セパレータ］を選択します。

［構成要素一覧］に，構成要素および構成要素に設定した開始セパレータが表示されます。

4. ［OK］ボタンをクリックします。

［複合内容要素］ダイアログに入力した情報で，選択条件が設定されます。なお，構成要素（複合内容要素）に開始セパレータを設定していない場合は，バイナリフォーマット定義ファイル検証時にエラーになります。

(6)　ルート要素を設定する

使用するバイナリデータを XML スキーマで表現したときに，ルート要素として扱う複合内容要素を設定します。

バイナリフォーマット定義ファイルの要素の定義時には，最初に設定した複合内容要素がルート要素になっています。ほかの複合内容要素をルート要素にするには，バイナリフォーマット定義エディタに表示されている複合内容要素を選択して右クリックし，［ルート要素に設定］を選択します。

［ルート要素に設定］を選択すると，バイナリフォーマット定義エディタ上で選択した複合内容要素がルート要素になります。バイナリフォーマット定義エディタ上では，ルート要素として扱う複合内容要素にルート要素のアイコンが表示されます。

なお，ある複合内容要素がすでにルート要素として設定されている場合，ほかの複合内容要素をルート要素として設定すると，ルート要素が変更されます。アウトラインビューの表示も，新しいルート要素からの表示に切り替わります。

(7)　セパレータを設定する

複合内容要素の構成要素として設定されている単純内容要素が，要素ごとに区切り文字（セパレータ）で区切られたデータ（セパレータデータ）の場合，複合内容要素にセパレータを設定します。

セパレータを設定する方法を次に示します。

1. 複合内容要素を選択して右クリックし，［設定］を選択します。
   ［複合内容要素］ダイアログが表示されます。

2. ［セパレータ］タブを選択します。
   ［複合内容要素］ダイアログの［セパレータ］タブの内容が表示されます。

3. 使用するセパレータ形式をドロップダウンリストから選択します。
   なお，セパレータ形式に CSV を選択した場合，開始セパレータおよび中間区切りセパレータは自動で設定されます。

4. 使用するセパレータの［使用する］チェックボックスをチェックし，［選択］ボタンをクリックします。
   ［セパレータ選択］ダイアログが表示されます。［セパレータ選択］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.8　セパレータ選択ダイアログ」を参照してください。

5. 使用するセパレータを一覧から選択し，［OK］ボタンをクリックします。
   ［複合内容要素］ダイアログのセパレータタブに選択したセパレータの情報が表示されます。
   なお，［セパレータ選択］ダイアログに，使用するセパレータがない場合，［セパレータ選択］ダイアログの［追加］ボタンをクリックします。表示される［セパレータ追加/変更］ダイアログでセパレータを追加/変更できます。
   ［セパレータ追加/変更］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.9　セパレータ追加/変更ダイアログ」を参照してください。

6. 必要に応じて，手順 3.と手順 4.を繰り返します。

7. ［OK］ボタンをクリックします。

［複合内容要素］ダイアログの［セパレータ］タブに入力した情報で，セパレータが設定されます。

4.4.5　要素を定義する（CSV 形式の場合）

CSV 形式の場合，バイナリフォーマット定義ファイルの内容は，バイナリフォーマット定義画面で定義します。バイナリフォーマット定義画面は，Eclipse のパッケージ・エクスプローラーに表示されるバイナリフォーマット定義ファイルをダブルクリックすると表示されます。

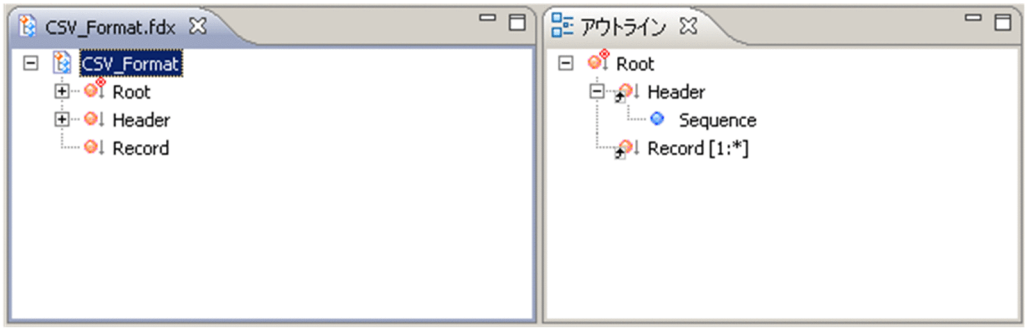
バイナリフォーマット定義画面については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.1　バイナリフォーマット定義画面」を参照してください。

参考

バイナリフォーマット定義画面では，複数のバイナリフォーマット定義ファイルを開いて編集できます。ただし，複数のバイナリフォーマット定義ファイルの間で，定義情報をコピーして貼り付けることはできません。

CSV 形式の場合，バイナリフォーマット定義ファイルを新規作成して，バイナリフォーマット定義画面を表示すると，ルート要素，ヘッダ要素，およびレコード要素が配置された状態で表示されます。

図 4-6　CSV 形式のデータを利用する場合のバイナリフォーマット定義画面の初期表示



ルート要素，ヘッダ要素，およびレコード要素はすべて複合内容要素です。ヘッダ要素およびレコード要素はルート要素の構成要素として設定されています。ヘッダ要素およびレコード要素には，ローカル定義の単純内容要素を構成要素として設定します。

バイナリフォーマット定義画面では次の作業を行います。

- フォーマット情報の設定
- ヘッダ要素およびレコード要素の出現回数の設定
- ヘッダ要素およびレコード要素の構成要素の設定
- ルート要素の設定

ポイント

CSV 形式のデータを利用する場合，バイナリフォーマット定義画面に初期表示されるルート要素，ヘッダ要素，およびレコード要素の構成は，変更できません。

定義の方法を次に示します。

(1)　フォーマット情報を設定する

バイナリフォーマット定義ファイルのフォーマット情報として，バイナリデータで使用されている文字コードや改行を指定します。

設定の方法は，CSV 以外の形式のバイナリデータを利用する場合と同じです。フォーマット情報の設定方法については，「4.4.4(1)　フォーマット情報を設定する」を参照してください。

(2)　ヘッダ要素およびレコード要素の出現回数を設定する

ヘッダ要素およびレコード要素の出現回数の設定方法を次に示します。

1. **ルート要素の構成要素として表示されているヘッダ要素またはレコード要素を選択して右クリックし，[設定] を選択します。**

   [構成要素] ダイアログが表示されます。

2. **[構成要素] ダイアログで出現回数を入力します。**

   なお，ヘッダ要素の出現回数は，CSV データにヘッダがない場合は「0」を，ヘッダがある場合は「1」を設定します。

   [構成要素] ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.4　構成要素ダイアログ」を参照してください。

3. **[OK] ボタンをクリックします。**

   [構成要素] ダイアログに入力した出現回数が設定されます。

(3)　ヘッダ要素およびレコード要素の構成要素を設定する

バイナリフォーマット定義エディタに，ルート要素の構成要素として表示されているヘッダ要素およびレコード要素の構成要素の設定方法を次に示します。

1. **ルート要素の構成要素として表示されているヘッダ要素またはレコード要素を選択して右クリックし，[ローカル単純内容要素の追加] を選択します。**

   [単純内容要素] ダイアログが表示されます。

2. **[単純内容要素] ダイアログに必要な情報を入力します。**

   [単純内容要素] ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

3. [OK] ボタンをクリックします。
［単純内容要素］ダイアログに入力した情報が設定されます。

(4) ルート要素を設定する

CSV データを利用する場合，バイナリフォーマット定義画面に初期表示されたルート要素以外の要素を，ルート要素に設定することはできません。ただし，ルート要素の名称だけ変更できます。

ルート要素の名称の変更方法を次に示します。

1. ルート要素を選択して右クリックし，［設定］を選択します。
［複合内容要素］ダイアログが表示されます。

2. [名称] を入力します。

3. [OK] ボタンをクリックします。
入力した名称が設定されます。

4.4.6 バイナリフォーマット定義ファイルを編集する

すでに作成，保存されているバイナリフォーマット定義ファイルの内容を変更できます。

バイナリフォーマット定義ファイルの内容を変更するには，Eclipse のパッケージ・エクスプローラーで編集するバイナリフォーマット定義ファイルをダブルクリックします。表示されるバイナリフォーマット定義画面で内容を変更します。

バイナリフォーマット定義画面で設定できる内容については，「4.4.4 要素を定義する（CSV 以外の形式の場合）」，または「4.4.5 要素を定義する（CSV 形式の場合）」を参照してください。

なお，作成済みのバイナリフォーマット定義ファイルをコピーして内容を変更し，別のバイナリフォーマット定義ファイルを作成することもできます。ただし，作成済みのバイナリフォーマット定義ファイルの定義内容の一部をコピーして流用することはできません。

4.4.7 バイナリフォーマット定義ファイルを検証する

必要な情報が設定されていなかったり，設定の内容や関係が正しくなかったりする場合，バイナリフォーマット定義ファイルを利用したデータ変換定義の作成や，データ変換が正しく行われません。

そのため，作成したバイナリフォーマット定義ファイルは，設定の内容が妥当かどうか検証します。検証は，任意のタイミングで実施できます。また，各要素を設定するダイアログでの入力時には，入力された値が自動的に検証されます。

(1) 検証内容

バイナリフォーマット定義ファイルの各要素の設定内容が検証できます。

検証内容とエラーまたは警告発生時の対処，およびどの検証内容がどの検証方法の場合に実施されるかについて，次の表に示します。

表 4-3 バイナリフォーマット定義ファイルの検証内容とエラーまたは警告発生時の対処

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
全体	名称が NCName 型か。	フォーマット名称および要素の名称を NCName 型で指定します。	○	−
フォーマット	逐次構成の複合内容要素が設定されているか。	逐次構成の複合内容要素を設定します。	−	○
	ルート要素が設定されているか。	バイナリフォーマット定義内の複合内容要素をルート要素として設定します。	−	○
	コード変換ライブラリが使用できるか（クラスをロードできるか）。	コード変換機能の jar ファイルを正しく設定したあと，Eclipse を再起動します。	−	○
	文字コードに KEIS，IBM_CODE，JEF を選択している場合に，セパレータを定義している複合内容要素がないか。	フォーマットの文字コードを次の文字コード以外にするか，または複合内容のセパレータの指定を解除します。 • KEIS • IBM_CODE • JEF	○	−
	文字コードに KEIS，IBM_CODE，JEF を選択している場合に，データ種別に文字列以外を設定している単純内容要素がないか。	フォーマットの文字コードを次の文字コード以外にするか，または単純内容のデータ種別に文字列を指定します。 • KEIS • IBM_CODE • JEF	○	−
	パック形式数値およびゾーン形式数値のビット値（16 進数）の値が 16 進数で設定されているか，または未設定でないか。	該当する符号ビットまたは文字を 16 進数で設定します。	○	−
	ゾーン形式数値の符号文字（16 進数）の値が 16 進数で設定されているか，または未設定でないか。	該当する符号ビットまたは文字を 16 進数で設定します。	○	−
	エスケープ文字がラテン文字または半角スペースの 1 文字か。	ラテン文字の範囲（0x21～0x7E）でエスケープ文字を設定します。	○	−
	エスケープ文字とセパレータ値に同一の文字が指定されていないか。※1	エスケープ文字とセパレータ値に重複しない文字を指定します。	○	−

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
フォーマット	エスケープ文字を指定している場合，セパレータを設定した複合内容要素内に個別文字コードを指定した単純内容要素が存在していないか。	次のどれかの対策を実施してください。 • エスケープ文字の指定を解除 • 複合内容要素のセパレータの指定を解除 • 単純内容要素の個別文字コードの指定を解除	○	○
	コード変換テーブルの格納ディレクトリ名が64文字以下の英数字（0～9，A～Z，a～z），およびアンダーバー（_）か。	コード変換テーブルの格納ディレクトリ名に64文字以下の英数字（0～9，A～Z，a～z），アンダーバー（_）を指定します。	○	○
	バイナリデータのエンコード，およびデコードでJava提供のエンコーダーとデコーダーを使用する場合，Javaでサポートされている文字コードが設定されているか。	文字コードにjava.nio API用の正準名または別名を指定してください。	○	－
単純内容要素	同じ名称の要素が同じ階層にないか。	同じ階層でユニークな名称を指定します。	○	－
	グローバル定義の単純内容要素は，複合内容要素の構成要素に設定されているか。	該当する単純内容要素を複合内容要素の構成要素に設定します。	○	－
	サイズが固定の場合，指定したサイズの値は次の式を満たすか。※2 0＜サイズ≦2,147,483,647	サイズを1～2,147,483,647の範囲で設定します。	○	－
	次の条件をすべて満たす場合，指定したサイズの値が偶数か。 • データ種別が文字列，整数，実数，固定小数部数値，または日付時刻 • サイズが固定 • 文字コードがUTF-16（BE/LE）	文字コードにUTF-16（BE/LE）以外を指定するか，該当する単純内容要素のサイズに偶数を指定します。	○	○
	次の条件をすべて満たす場合，バイトサイズが2以上か。 • データ種別が文字列 • サイズが固定 • 文字コードがKEIS，IBM_CODE，JEF	フォーマットの文字コード，または個別文字コードを次の文字コード以外にするか，該当する単純内容のサイズに2以上の値を指定します。 • KEIS • IBM_CODE • JEF	○	○
	次の条件をすべて満たす場合，0020～D7DE，E000～FFDC（大文字・小文字は区別しない）の値が設定されているか。 • データ種別が文字列 •「代替文字に置換」を指定している	0020～D7DE，E000～FFDCの16進数を指定します。	○	－
	次の条件をすべて満たす場合，埋め字が「0」以外か。 • データ種別が整数，実数，または固定小数部数値	左右寄せを「左」以外にするか，埋め字に「0」以外を指定します。	○	－

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
単純内容要素	• 左右寄せで「左」を指定している	左右寄せを「左」以外にするか，埋め字に「0」以外を指定します。	○	−
	サイズが固定の場合，指定した整数は次の式を満たすか。 全体桁数＞符号の桁数	全体桁数に符号の桁数よりも大きな値を指定します。	○	−
	サイズが固定で全体桁数の指定がない場合，指定した整数は次の式を満たすか。 サイズ＞符号サイズ	サイズに符号のバイトサイズよりも大きな値を指定します。	○	○
	サイズが固定で全体桁数の指定がある場合，指定した整数は次の式を満たすか。 サイズ≧符号サイズ＋整数部サイズ	サイズに全体桁数のバイトサイズ以上の値を指定します。	○	○
	次の条件をすべて満たす場合，文字コードがKEIS，IBM_CODE，JEF 以外か。 • データ種別が整数，実数，固定小数部数値，ゾーン形式数値，パック形式数値，符号付 2 進整数，符号無 2 進整数，バイト列，ビット列，または日付時刻	フォーマットの文字コードを次の文字コード以外にするか，データ種別を文字列に指定します。 • KEIS • IBM_CODE • JEF	○	−
	指定した整数，実数，または固定小数部数値は次の式を満たすか。 1≦全体桁数≦34	該当する桁数を，設定可能範囲内の数値で設定します。	○	−
	サイズが固定の場合，指定した実数は次の式を満たすか。 全体桁数≧符号の桁数＋小数点の桁数＋小数部桁数 かつ 全体桁数＞符号の桁数	全体桁数に符号と小数点と小数部を合わせた桁数以上の値を指定します。	○	−
	サイズが可変の場合，指定した実数または固定小数部数値は次の式を満たすか。 全体桁数≧小数部桁数	全体桁数に小数部桁数以上の値を指定します。	○	−
	サイズが固定で全体桁数と小数部桁数の指定がない場合，指定した実数は次の式を満たすか。 サイズ＞符号サイズ	サイズに符号のバイトサイズよりも大きな値を指定します。	○	○
	サイズが固定で，全体桁数の指定がなく小数部桁数の指定がある場合，指定した実数は次の式を満たすか。 サイズ≧符号サイズ＋小数点サイズ＋小数部サイズ かつ サイズ＞符号サイズ	サイズに符号と小数点と小数部を合わせたバイトサイズ以上の値を指定します。	○	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
単純内容要素	サイズが固定で全体桁数の指定がある場合，指定した実数は次の式を満たすか。 サイズ≧符号サイズ＋整数部サイズ＋小数点サイズ＋小数部サイズ	サイズに全体桁数のバイトサイズ以上の値を指定します。	○	○
	サイズが固定の場合，指定した固定小数部数値は次の式を満たすか。 全体桁数≧符号の桁数＋小数部桁数 かつ 全体桁数＞符号の桁数	全体桁数に符号と小数部を合わせた桁数以上の値を指定します。	○	－
	サイズが固定で全体桁数の指定がない場合，指定した固定小数部数値は次の式を満たすか。 サイズ≧符号サイズ＋小数部サイズ かつ サイズ＞符号サイズ	サイズに符号と小数部桁数を合わせたバイトサイズ以上の値を指定します。	○	○
	サイズが固定で全体桁数の指定がある場合，指定した固定小数部数値は次の式を満たすか。 サイズ≧符号サイズ＋整数部サイズ＋小数部サイズ	サイズに全体桁数のバイトサイズ以上の値を指定します。	○	○
	データ種別が実数，固定小数部数値，ゾーン形式数値，またはパック形式数値の場合，指定した小数部桁数は次の式を満たすか。 0≦小数部桁数≦33	小数部桁数を 0～33 で設定します。	○	－
	次の単純内容要素の祖先にセパレータを設定している場合，「セパレータ形式」が「ユーザ指定」になっているか。 • データ種別がゾーン形式数値，パック形式数値，符号付 2 進整数，符号無 2 進整数，またはバイト列	ゾーン形式数値，パック形式数値，符号付 2 進整数，符号無 2 進整数，またはバイト列以外のデータ種別を指定します。または，単純内容に対するセパレータ形式の指定をユーザ指定にします。	○	○
	次の条件をすべて満たす場合，「小数部桁数≦指定のサイズから求めた最大小数部桁数」になっているか。 • データ種別がパック形式数値 • サイズが固定	サイズに小数部桁数のバイト数以上の値を指定します。	○	○
	データ種別が符号付 2 進整数または符号無 2 進整数の場合，指定したサイズは次の式を満たすか。 1≦サイズ≦8	該当する桁数を，設定できる範囲内の数値で設定します。	○	－
	データ種別がビット列の場合，指定したサイズは次の式を満たすか。 1≦サイズ≦64	サイズを 1～64 の範囲で設定します。	○	－

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
単純内容要素	データ種別がビット列の場合，セパレータを定義している複合内容要素がバイナリフォーマット定義に存在しないか。	データ種別にビット列以外を指定するか，セパレータの指定を解除します。	○	−
	データ種別が日付時刻で，秒小数部桁数が指定されている場合，指定した秒小数部桁数は次の式を満たすか。 0≦秒小数部桁数≦3	秒小数部桁数を 0～3 で設定します。	○	−
	データ種別が日付時刻で，サイズが固定の場合，指定したサイズは次の式を満たすか。 サイズ≧形式サイズ＋小数点サイズ＋秒小数部桁数	サイズに日付時刻形式のバイト数以上の値を設定します。	○	○
	次の条件をすべて満たす場合，「小数部桁数≦指定のサイズから求めた最大小数部桁数」になっているか。 • データ種別がゾーン形式数値 • サイズが固定 • 符号の設定がカスタム • 符号の種類が符号ビット または，次の条件をすべて満たす場合，「小数部桁数≦指定のサイズから求めた最大小数部桁数」になっているか。 • データ種別がゾーン形式数値 • サイズが固定 • 符号の設定が自動	サイズに小数部桁数のバイト数以上の値を指定します。	○	○
	次の条件をすべて満たす場合，「指定したサイズの最大小数部桁数≧小数部桁数＋符号文字」になっているか。かつ，指定したサイズが 2 バイト以上か。 • データ種別がゾーン形式数値 • サイズが固定 • 符号の設定が「カスタム」 • 符号の種類が「符号文字」	サイズには，小数部桁数のバイト数＋符号文字のバイト数以上，かつ 2 バイト以上の値を指定します。	○	○
	次の条件をすべて満たす場合，祖先に CSV 形式のセパレータが設定されていないか。 • データ種別が文字列，整数，または実数 • 個別文字コードが指定されている	単純内容要素に対する CSV 形式のセパレータの指定を解除するか，個別文字コードの指定を解除してください。	○	○
	次の条件をすべて満たす場合，個別文字コードが指定されていないか。 • データ種別が文字列，整数，または実数 • 祖先にセパレータが設定されている • エスケープ文字が設定されている	次のどれかの対策を実施してください。 • 単純内容要素の個別文字コードの指定を解除 • 複合内容要素のセパレータの指定を解除 • エスケープ文字の指定を解除	○	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
単純内容要素	次の条件をすべて満たす場合，祖先にセパレータが設定されていないか。 • データ種別が文字列 • 文字コードが KEIS，IBM_CODE，JEF	単純内容要素に対するセパレータの指定を解除するか，単純内容要素の個別文字コードを次の文字コード以外にしてください。 • KEIS • IBM_CODE • JEF	○	○
	データ種別が整数，または実数の場合，個別文字コードが KEIS，IBM_CODE，JEF 以外か。	個別文字コードを次の文字コード以外にするか，データ種別を文字列に指定します。 • KEIS • IBM_CODE • JEF	○	○
	サイズが 0 バイトの場合，親要素が選択構成になっているか。	単純内容のサイズを 1 バイト以上に変更するか，単純内容が定義されている複合内容を選択構成に変更してください。	−	○
複合内容要素	同じ名称の要素が同じ階層にないか。	同じ階層でユニークな名称を指定します。	○	−
	ルート要素に設定されている複合内容要素が，ほかの複合内容要素の構成要素として設定されていないか。	複合内容要素の構成要素から，ルート要素に設定されている複合内容要素を削除します。	−	○
	複合内容要素はほかの複合内容要素の構成要素に設定されているか（ただしルート要素の場合は除く）。	該当する複合内容要素を，ほかの複合内容要素の構成要素に設定します。	−	○
	構成要素は設定されているか。	該当する複合内容要素に構成要素を設定します。	−	○
	セパレータ形式に「CSV」を指定している場合，次の単純内容要素が存在しないか。 • データ種別がゾーン形式数値，パック形式数値，符号付 2 進数整数，符号無 2 進数整数，またはバイト列	セパレータ形式をユーザ指定にするか，単純内容にゾーン形式数値，パック形式数値，符号付 2 進数整数，符号無 2 進数整数，またはバイト列以外のデータ種別を指定します。	○	○
	開始セパレータの指定が正しいか。	開始セパレータに正しいセパレータを指定します。	−	○
	中間区切りセパレータの指定が正しいか。	中間区切りセパレータに正しいセパレータを指定します。	−	○
	終了セパレータの指定が正しいか。	終了セパレータに正しいセパレータを指定します。	−	○
	セパレータを指定している場合，フォーマットの文字コードが KEIS，IBM_CODE，JEF でないか。	複合内容のセパレータの指定を解除するか，フォーマットの文字コードを次の文字コード以外にします。 • KEIS • IBM_CODE	○	−

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
複合内容要素	セパレータを指定している場合，フォーマットの文字コードが KEIS，IBM_CODE，JEF でないか。	• JEF	○	−
	セパレータ名称が重複していないか。	1 つの複合内容の中で，ほかのセパレータで指定されていないセパレータを指定します。	○	−
	セパレータ値が重複していないか。	1 つの複合内容の中で，ほかのセパレータ値と重複しないセパレータ値を指定します。	○	−
	セパレータを指定している場合，文字コード変換中にエラーが発生しないか。	エラーの原因となった例外情報を参照し，その対処に従います。	−	○
	セパレータを指定している場合，文字コード変換に失敗しないか。	エラーコードを参照し，その対処に従います。	−	○
	セパレータを指定している場合，文字コードがコード変換ライブラリでサポートされているか。	コード変換ライブラリでサポートしている文字コードを選択します。 または，サポートしているコード変換ライブラリを使用します。	−	○
	セパレータを指定している場合，バイナリフォーマット定義にビット列型の単純内容要素が存在しないか。	セパレータの指定を解除するか，単純内容要素のデータ種別にビット列以外を指定します。	○	−
	セパレータを指定している場合，バイナリフォーマット定義にサイズノードを指定した複合内容要素が存在しないか。	セパレータの指定を解除するか，サイズノードを除去します。	○	−
	逐次構成で，最後の子要素がビット列型の単純内容要素の場合，次の条件に該当しないか（ただし，ルート要素は除く）。 • 連続するビット列型のサイズを合計し，サイズがバイト単位でない	ビット列型の子要素のサイズを確認します。 ビット列型の要素のサイズは，次のように算出します。 出現回数が固定の場合，「サイズ×最大出現回数」とします。 出現回数が固定以外の場合，次のように算出します。 • 実際の出現回数が不明でサイズがバイト単位でない場合，警告とする。 • サイズがバイト単位の場合は，出現回数が不明でも，「サイズ×出現回数」の値がバイト単位に常になるため，出現回数は 1 として算出し，「サイズ×1」とする。	−	○
	選択構成で，子要素がビット列型の単純内容要素の場合，次の条件に該当しないか（ただし，ルート要素は除く）。 • サイズがバイト単位でない	ビット列型の子要素のサイズを確認します。 ビット列型の要素のサイズは，次のように算出します。 出現回数が固定の場合，「サイズ×最大出現回数」とします。	−	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
複合内容要素	選択構成で，子要素がビット列型の単純内容要素の場合，次の条件に該当しないか（ただし，ルート要素は除く）。 • サイズがバイト単位でない	出現回数が固定以外の場合，次のように算出します。 • 実際の出現回数が不明でサイズがバイト単位でない場合，警告とする。 • サイズがバイト単位の場合は，出現回数が不明でも，「サイズ×出現回数」の値がバイト単位に常になるため，出現回数は 1 として算出し，「サイズ×1」とする。	−	○
	選択構成の場合，選択条件ノードが設定されているか。	該当する複合内容要素に選択条件ノードを設定します。	−	○
	選択構成の場合，選択条件ノードに設定されている要素は単純内容要素か。	選択条件ノードとして単純内容要素のノードを設定します。	○	○
	選択構成の場合，各構成要素に選択条件値が設定されているか（ただし，1 つだけは選択条件値を設定しなくてもよい）。	各構成要素に選択条件値を設定し，選択条件値が設定されていない構成要素を 1 つ以下にします。	−	○
	選択構成の場合，選択条件ノードに設定されている単純内容要素のデータ種別と，各構成要素の選択条件値に指定した値が一致するか。※3	選択条件ノードに設定されている単純内容要素のデータ種別によって検証の内容と対処が異なります。詳細については「表 4-4　選択条件値に指定した値に関する検証内容と対処」を参照してください。	○	○
	選択構成の場合，各構成要素の選択条件値に重複はないか。※3	選択条件値が重複した構成要素に，ほかの構成要素で設定されていない選択条件値を設定します。	−	○
	選択構成の場合，選択条件ノード，およびそのすべての祖先ノードの出現回数が固定で，かつ 1 回か。	選択条件ノードとその祖先ノードの出現回数を 1 回固定になるように設定します。	−	○
	選択構成の場合，選択条件ノードのサイズが固定か。	選択条件ノードのサイズを固定に設定します。	−	○
	選択構成の場合，選択条件ノードのパスが正しいか。	選択条件ノードを示すパスが不正であるおそれがあります。対処方法については「表 4-5　パスが不正である場合の原因と対処」を参照してください。	○	○
	選択構成の場合，選択条件ノードのパスに選択構成の複合内容要素が含まれていないか（ただし，パスが絶対パスの場合，設定元のノードの祖先ノードは除く）。	選択条件ノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータで選択条件ノードに該当するデータが出現するか確認します。	−	○
	選択構成の場合，選択条件ノードに指定したノードが，サイズノード指定した複合内容要素を含まないか（パスが絶対パスの場合は，設定	選択条件ノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータで選択条件ノードに該当するデータが出現するか確認します。	−	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
複合内容要素	元のノードの祖先であるノードは除く。パスが基準パスの場合は，基準ノードを除く)。	選択条件ノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータで選択条件ノードに該当するデータが出現するか確認します。	−	○
	選択構成の場合，文字コード変換中にエラーが発生しないか。	エラーの原因となった例外情報を参照し，その対処に従います。	○	○
	選択構成の場合，文字コード変換に失敗しないか。	エラーコードを参照し，その対処に従います。	○	○
	選択構成の場合，文字コードがコード変換ライブラリでサポートされているか。	コード変換ライブラリでサポートしている文字コードを選択します。 または，サポートしているコード変換ライブラリを使用します。	○	○
	選択構成の場合で構成要素の開始セパレータが設定されているとき，構成要素に設定されている単純内容要素の数は 1 つ以下か。	構成要素に設定する単純内容要素を 1 つ以下にします。	−	○
	選択構成の場合，各構成要素（複合内容要素）に開始セパレータが設定されているか（ただし，構成要素に単純内容要素が設定されていない場合，1 つだけは開始セパレータを設定しなくてもよい)。	構成要素に単純内容要素が設定されている場合，各構成要素に開始セパレータを設定します。構成要素に単純内容要素が設定されていない場合，開始セパレータが設定されていない構成要素を 1 つ以下にします。	−	○
	選択構成の場合，各構成要素に設定されている開始セパレータの値に重複はないか。	開始セパレータが重複した構成要素に，ほかの構成要素で設定されていない開始セパレータの値を設定します。	−	○
	セパレータ形式に「CSV」を指定している場合，個別文字コードを設定した単純内容要素が存在しないか。	個別文字コードの指定を解除するか，複合内容要素のセパレータ形式を変更してください。	○	○
	セパレータを指定している場合，個別文字コードに KEIS，IBM_CODE，JEF を指定した単純内容要素が存在しないか。	複合内容要素のセパレータの指定を解除するか，単純内容の個別文字コードを次の文字コード以外にしてください。 • KEIS • IBM_CODE • JEF	○	○
	次の条件をすべて満たす場合，セパレータが指定されていないか。 • 個別文字コードが設定されている • エスケープ文字が設定されている	次のどれかの対策を実施してください。 • 複合内容要素のセパレータの指定を解除 • 単純内容要素の個別文字コードの指定を解除 • エスケープ文字の指定を解除	○	○
	選択条件ノード，サイズノード，または出現回数ノードに LOW-VALUE の変換が有効な単純内容要素が指定されていないか。	選択条件ノード，サイズノード，または出現回数ノードに「LOW-VALUE の変換」が無効である単純内容を指定してください。	○	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
複合内容要素	文字列型の選択条件ノードのサイズが 1 バイト以上か。	文字列型の選択条件ノードにはサイズが 1 バイト以上の単純内容を指定してください。	○	○
	複数条件指定の場合，条件値が区切り文字だけで指定されていないか。または，区切り文字の後ろに条件値が指定されているか。	正しい区切り文字で条件値を指定してください。	○	○
	複数条件指定の場合，区切り文字が指定されているか。	区切り文字を指定してください。	○	○
構成要素	出現回数が固定の場合，指定した出現回数は次の式を満たすか。 0 ＜出現回数≦2,147,483,647	出現回数を 1～2,147,483,647 の範囲で設定します。	○	－
	出現回数が「出現回数ノード」の場合，出現回数ノードが設定されているか。	該当する構成要素に出現回数ノードを設定します。	－	○
	出現回数が「出現回数ノード」の場合，出現回数ノードのデータ種別が数値型か（数値型とは，整数，実数，固定小数部数値，ゾーン形式数値，パック形式数値，符号付 2 進整数，または符号無 2 進整数）。	出現回数ノードには，データ種別が数値型の単純内容要素を設定します。	○	○
	出現回数が「出現回数ノード」の場合，出現回数ノード，およびそのすべての祖先ノードの出現回数が固定，かつ 1 回か。	出現回数ノードとその祖先ノードの出現回数を 1 回固定になるように設定します。	－	○
	出現回数が「出現回数ノード」の場合，出現回数ノードのパスが正しいか。	出現回数ノードを示すパスが不正であるおそれがあります。対処方法については「表 4-5　パスが不正である場合の原因と対処」を参照してください。	○	○
	出現回数が「出現回数ノード」の場合，出現回数ノードのパスに選択構成の複合内容要素が含まれていないか（ただし，パスが絶対パスの場合，設定元のノードの祖先ノードは除く）。	出現回数ノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータで出現回数ノードに該当するデータが出現するか確認します。	－	○
	出現回数ノードに指定したノードが，サイズノード指定した複合内容要素を含まないか（パスが絶対パスの場合は，設定元のノードの祖先であるノードは除く。パスが基準パスの場合は，基準ノードを除く）。	選択条件ノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータで選択条件ノードに該当するデータが出現するか確認します。	－	○
	再帰構造になっていないか。	再帰的に設定されている構成要素を削除します。	○	－
	親要素が逐次構成の場合，次のすべての条件に該当しないか。 • ビット列型以外の単純内容要素，または複合内容要素 • 1 つ前に存在する兄弟要素がビット列型の単純内容要素	ビット列型の兄要素のサイズを確認します。 ビット列型の要素のサイズは，次のように算出します。 出現回数が固定の場合，「サイズ×最大出現回数」とします。	－	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
構成要素	• 連続するビット列型のサイズを合計し，サイズがバイト単位でない	出現回数が固定以外の場合，次のように算出します。 • 実際の出現回数が不明でサイズがバイト単位でない場合，警告とする。 • サイズがバイト単位の場合は，出現回数が不明でも，「サイズ×出現回数」の値がバイト単位に常になるため，出現回数は 1 として算出し，「サイズ×1」とする。	−	○
	構成要素が単純内容要素の場合で，データ種別が「ビット列」のとき，次のすべての条件に該当しないか。 • 指定されたサイズが 8 ビット未満 • 出現回数が「データの終端まで」または「範囲指定」 • バイナリフォーマット定義の最後の要素 • 祖先ノードにサイズノードの指定がない	単純内容要素の出現回数を「固定」または「出現回数ノード」に変更するか，サイズに 8 ビット以上の値を指定します。	−	○
	構成要素が単純内容要素の場合で，サイズが「サイズノード」のとき，サイズノードが設定されているか。	該当する構成要素にサイズノードを設定します。	−	○
	構成要素が単純内容要素の場合で，サイズが「サイズノード」のとき，サイズノードのデータ種別が数値型か（数値型とは，整数，実数，固定小数部数値，ゾーン形式数値，パック形式数値，符号付 2 進整数，または符号無 2 進整数）。	サイズノードには，データ種別が数値型の単純内容要素を設定します。	○	○
	構成要素が単純内容要素の場合で，サイズが「サイズノード」のとき，サイズノード，およびそのすべて祖先ノードの出現回数が固定，かつ 1 回か。	サイズノードとその祖先ノードの出現回数を 1 回固定になるように設定します。	○	○
	構成要素が単純内容要素の場合で，サイズが「サイズノード」のとき，サイズノードのパスが正しいか。	サイズノードを示すパスが不正であるおそれがあります。対処方法については「表 4-5　パスが不正である場合の原因と対処」を参照してください。	○	○
	構成要素が単純内容要素の場合で，サイズが「サイズノード」のとき，サイズノードのパスに選択構成の複合内容要素が含まれていないか（ただし，パスが絶対パスの場合，設定元のノードの祖先ノードは除く）。	サイズノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータでサイズノードに該当するデータが出現するか確認します。	−	○
	構成要素が単純内容要素の場合で，サイズノードに指定したノードが，サイズノード指定した複合内容要素を含まないか（パスが絶対パスの場合は，設定元のノードの祖先であるノードは	サイズノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータでサイズノードに該当するデータが出現するか確認します。	−	○

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
構成要素	除く。パスが基準パスの場合は，基準ノードを除く)。	サイズノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータでサイズノードに該当するデータが出現するか確認します。	－	○
	構成要素が単純内容要素の場合で，出現回数が「データの終端まで」のとき，出現の終わりを示すセパレータまたはサイズノードが設定されているか。	出現回数が可変の要素で，出現の終わりを示すセパレータまたは複合内容要素のサイズノードを指定します。	○	○
	構成要素が単純内容要素の場合で，サイズが「データの終端まで」のとき，データの終了を示すセパレータまたはサイズノードが設定されているか。	サイズが可変長の要素で，データの終了を示すセパレータまたは複合内容要素のサイズノードを指定します。	○	○
	構成要素が複合内容要素の場合で，出現回数が「データの終端まで」のとき，出現の終わりを示すセパレータまたはサイズノードが設定されているか。	出現回数が可変の要素で，出現の終わりを示すセパレータまたは複合内容要素のサイズノードを指定します。	○	○
	構成要素が複合内容要素の場合で，出現回数が「データの終端まで」または「範囲指定」の場合，次のすべての条件に該当しないか。 1. バイナリフォーマット定義の最後の要素のデータ種別が「ビット列」 2. 1.の要素の出現回数が「固定」または「出現回数ノード」 3. 1.の要素から祖先ノードにサイズノードの指定がない 4. 1.の要素までの合計サイズが８ビット未満	複合内容要素の出現回数を「固定」または「出現回数ノード」に変更するか，複合内容要素のサイズが８ビット以上になるように定義します。 各要素のサイズは，次のように算出します。 • データ種別がビット列，かつ出現回数が固定の場合は，「サイズに指定された値×最大出現回数」とする。 • データ種別がビット列，かつ出現回数が固定以外の場合は，サイズに指定された値とする。ただし，最後の要素でなく，最小出現回数が０の場合，サイズを０とする。 • データ種別がビット列以外の場合，サイズを８とする（合計サイズが８ビットを超えた時点で検証が終了するため)。ただし，最小出現回数が０の要素，またはサイズノードが指定された複合内容要素の場合，サイズを０とする。	－	○
	構成要素が複合内容要素の場合で，サイズノードが設定され，子孫ノードにビット列が存在するとき，ビット列の合計サイズがバイト単位であるか。	ビット列型の単純内容要素のサイズを確認します。	－	○
	構成要素が複合内容要素でサイズノードを設定する場合，セパレータを定義している複合内容要素がバイナリフォーマット定義に存在しないか。	サイズノードの指定を除去するか，セパレータの指定を解除します。	○	－

検証の対象	検証内容	エラーまたは警告発生時の対処	検証方法	
			自動	任意
構成要素	構成要素が複合内容要素の場合で，サイズノードのデータ種別が数値型であるか（数値型とは，整数，実数，固定小数部数値，ゾーン形式数値，パック形式数値，符号付 2 進整数，または符号無 2 進整数）。	サイズノードに数値型の単純内容を指定します。	○	○
	構成要素が複合内容要素の場合で，サイズノードおよびそのすべての出現回数が固定かつ 1 回であるか。	サイズノードとその祖先は出現回数が 1 回固定のノードになるように定義します。	○	○
	構成要素が複合内容要素の場合で，サイズノードのパスが正しいか。	サイズノードに指定したパスが壊れているおそれがあります。 詳細情報を参考にサイズノードを指定し直します。	○	○
	構成要素が複合内容要素の場合で，サイズノードのパスに選択構成の複合内容要素を含まないか。	サイズノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータでサイズノードに該当するデータが出現するか確認します。	－	○
	構成要素が複合内容要素の場合で，サイズノードに指定したノードが，サイズノード指定した複合内容要素を含まないか（パスが絶対パスの場合は，設定元のノードの祖先であるノードは除く。パスが基準パスの場合は，基準ノードを除く）。	サイズノードが実際のバイナリデータで出現しないおそれがあります。実際のバイナリデータでサイズノードに該当するデータが出現するか確認します。	－	○
	構成要素が単純内容，または複合内容の場合，サイズノードのデータ種別が数値またはビット列型か。	サイズノードに数値型，またはビット列型の単純内容を指定してください。	○	○

（凡例）

○：検証されます。

－：検証されません。

注※1

セパレータ値の種別がバイト列のセパレータは検証されません。

注※2

文字列型のローカル単純内容要素だけ，サイズに 0 バイトを指定することができます。

注※3

複数の条件値を指定した場合，指定された条件値を先頭から順に検証します。すべての条件値が検証の対象になります。

表 4-4　選択条件値に指定した値に関する検証内容と対処

データの種別	検証内容	対処
文字列	「代替文字に置換」が指定されていないとき，次の式を満たすか。※1 文字コード変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。

データの種別	検証内容	対処
整数	指定した文字列の形式は n か。※2※3	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。※1 文字コード変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。
	符号の有無が「なし」の場合，0≦値か。	選択条件値に正の値を指定してください。
	「全体桁数」の指定がある場合，条件値の桁数が「全体桁数」の指定桁数以下か。	選択条件値の入力値の桁数を全体桁数以下で指定してください。
	「全体桁数」の指定がない場合，条件値の桁数が 34 桁以下か。	選択条件値に有効桁数が 34 桁以下の数値を指定してください。
実数	指定した文字列の形式は n または n.m か。※2※3	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。※1※4 文字コード変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。
	符号の有無が「なし」の場合，0≦値か。	選択条件値に正の値を指定してください。
	「小数部桁数」の指定がある場合，条件値の小数部桁が「小数部桁数」の指定桁数以下か。	選択条件値の小数部の桁数を小数部桁数以下で指定してください。
	「全体桁数」の指定がある場合，条件値の桁数が「全体桁数」の指定桁数以下か。※4	選択条件値の入力値の桁数を全体桁数以下で指定してください。
	「全体桁数」の指定がない場合，条件値の桁数が 34 桁以下か。	選択条件値に有効桁数が 34 桁以下の数値を指定してください。
固定小数部数値	指定した文字列の形式は n または n.m か。※2※3	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。※1 暗黙的小数部付き数値に変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。
	符号の有無が「なし」の場合，0≦値か。	選択条件値に正の値を指定してください。
	条件値の小数部桁が「小数部桁数」の指定桁以下か。	選択条件値の小数部の桁数を小数部桁数以下で指定してください。
	「全体桁数」の指定がある場合，暗黙的小数部付き数値に変換後の条件値の桁数が「全体桁数」の指定桁数以下か。	選択条件値の入力値の桁数を全体桁数以下で指定してください。
	「全体桁数」の指定がない場合，条件値の桁数が 34 桁以下か。	選択条件値に有効桁数が 34 桁以下の数値を指定してください。
ゾーン形式数値	指定した文字列の形式は n または n.m か。※2	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。 ゾーン形式に変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。

データの種別	検証内容	対処
ゾーン形式数値	整数部桁＋小数部桁が 34 桁以下の実数か。	選択条件値に有効桁数が 34 桁以下の数値を指定してください。
	小数部桁は「小数部桁数」に指定した範囲内か。	選択条件値の小数部の桁数を小数部桁数以下で指定してください。
	単純内容要素の符号の設定が「カスタム」，かつ符号の有無が「なし」の場合に，条件値に負の値を設定していないか。	選択条件値に正の値を指定してください。
パック形式数値	指定した文字列の形式は n または n.m か。※2	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。 パック形式に変換後のサイズ≦指定のサイズ	選択条件値のバイトサイズが選択条件ノードのサイズ以下になるように指定してください。
	整数部桁＋小数部桁が 34 桁以下の実数か。	選択条件値に有効桁数が 34 桁以下の数値を指定してください。
	小数部桁は「小数部桁数」に指定した範囲内か。	選択条件値の小数部の桁数を小数部桁数以下で指定してください。
	単純内容要素の符号の設定が「カスタム」，かつ符号の有無が「なし」の場合に，条件値に負の値を設定していないか。	選択条件値に正の値を指定してください。
符号付 2 進整数	指定した文字列の形式は n または-n か。※2	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	サイズが n バイト（n=1～8）の場合，次の式を満たす整数か。 $-1\times2^{(8\times n)-1}\leqq$値$\leqq2^{(8\times n)-1}-1$	選択条件値に有効な値を指定してください。
符号無 2 進整数	指定した文字列の形式は n か。※2	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	サイズが n バイト（n=1～8）の場合，次の式を満たす整数か。 $0\leqq$値$\leqq2^{8\times n}-1$	選択条件値に有効な値を指定してください。
バイト列	エンコード形式が hexBinary の場合，指定した文字列の形式は 0～9，a～f，A～F か。 エンコード形式が base64Binary の場合，指定した文字列の形式は 0～9，a～z，A～Z，+，/，=，空白文字か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。 バイト変換後のサイズ=指定のサイズ	条件値のバイトサイズが選択条件ノードのバイトサイズと同じになるように定義してください。
ビット列	指定した文字列の形式は 0～9，a～f，A～F か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	次の式を満たすか。 A=（<指定のサイズ>+7）/8（小数点以下切り捨て）	必要桁数を指定してください。

データの種別	検証内容	対処
ビット列	A×2=入力文字数	必要桁数を指定してください。
日付時刻	形式が CCYYMMDD の場合，指定した文字列形式は CCYYMMDD か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	形式が YYMMDD の場合，指定した文字列形式は YYMMDD か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	形式が hhmmss の場合，指定した文字列形式は hhmmss か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	形式が CCYYMMDDhhmmss の場合，指定した文字列形式は CCYYMMDDhhmmss か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	形式が YYMMDDhhmmss の場合，指定した文字列形式は YYMMDDhhmmss か。	選択条件値に選択条件ノードのデータ種別と一致した文字列を指定してください。
	秒小数点の指定は，秒小数点の桁数と一致するか。	選択条件値の秒小数点桁数と選択条件ノードに指定した値を一致させてください。
	指定した日付時刻は正しい日付時刻か。	選択条件値に正しい日付時刻を指定してください。

注

複数の条件値を指定した場合，指定された条件値を先頭から順に検証します。すべての条件値が検証の対象になります。

注※1

文字コードに CUSTOM 以外が設定されている場合。

注※2

n，m は 0～9 の数値を表します。ただし，n の先頭，および m の末尾は 1～9 の数値です。

注※3

条件値に付ける符号は，選択条件ノードの「符号の有無」の設定によって異なります。「常にあり」の場合は必ず符号を付けます。「なし」の場合，符号は指定できません。「負だけあり」の場合は「－」だけ指定できます。

注※4

条件値が小数部を持ち，整数部が 0 だけの場合，0 を削除してからカウントします。

表 4-5　パスが不正である場合の原因と対処

パス不正の原因	対処
選択条件，出現回数，またはサイズノードに指定したパスのノードが定義上から削除されている。	選択条件，出現回数，またはサイズノードを指定し直してください。
選択条件，出現回数，またはサイズノードを指定したあとに，階層構造が変更された。	選択条件，出現回数，またはサイズノードを指定し直してください。
選択条件，出現回数，またはサイズノードに指定したパスが，設定先の要素よりもあとに出現する。	選択条件，出現回数，またはサイズノードに，設定先より前に出現する要素を指定してください。
ノードパス設定先のノードを参照する構成要素が複数定義されている。	ノードパス設定先を参照する構成要素を 1 つにしてください。

パス不正の原因	対処
絶対パスの最上位要素がルート要素ではない。	ルート要素を絶対パスの最上位要素に設定するか，パスを設定し直してください。
設定元のノードの祖先に基準ノードが存在しない。	ノードパスを設定し直してください。
基準パスの基準ノードが逐次構成の要素ではない。	基準ノードが逐次構成になるパスを設定してください。
パスと設定元のノードの共通の親が，選択構成の複合内容である。	パスと設定元のノードの共通の親を逐次構成の複合内容に設定してください。

(2)　検証方法

検証方法には，次の 2 つがあります。検証方法によって検証される内容は異なります。詳細については，「(1)　検証内容」を参照してください。

ダイアログの入力時の検証

バイナリフォーマット定義ファイルの各要素を設定するときに利用するダイアログでは，入力された値が逐次自動的に検証されます。

任意のタイミングでの検証

バイナリフォーマット定義ファイルの作成後，または作成途中の任意のタイミングで検証を実施できます。

検証を実施するには，バイナリフォーマット定義エディタ上の要素を選択して右クリックし，［検証］を選択します。バイナリフォーマット定義エディタに表示されているバイナリフォーマット定義ファイルが検証されます。

なお，検証するときにバイナリフォーマット定義エディタ上で選択する要素は任意です。どの要素を選択しても，バイナリフォーマット定義ファイル全体が検証されます。

検証結果の表示については，「(3)　検証結果の表示」を参照してください。

(3)　検証結果の表示

検証を実施した結果は，画面およびダイアログ上に表示されます。

(a)　ダイアログの入力時の自動検証の結果の表示

ダイアログの入力時の自動検証の結果は，バイナリフォーマット定義に関連するダイアログの［インフォメーション］に表示されます。問題がない場合は，何も表示されません。

バイナリフォーマット定義に関連するダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.2.1　バイナリフォーマット定義画面」を参照してください。

(b)　任意のタイミングで検証した結果の表示

検証の結果が画面に表示されます。

画面の表示内容については，次の個所を参照してください。

- マニュアル「サービスプラットフォーム リファレンス」の「1.1.1　バイナリフォーマット定義ファイルを定義する画面の構成」
- マニュアル「サービスプラットフォーム リファレンス」の「1.2.1　バイナリフォーマット定義画面」

検証が成功した場合

コンソールビューに検証が成功したことを示すメッセージが表示されます。

検証が失敗した場合

コンソールビューにエラー，警告，および情報通知の内容を示すメッセージが表示されます。また，画面上の各部分にエラー，警告，および情報通知の内容が表示されます。

バイナリフォーマット定義エディタ，アウトラインビュー

問題のあったノードにエラーまたは警告を示すアイコンが表示されます。エラーと警告の両方がある場合，エラーのアイコンが優先して表示されます。

パッケージ・エクスプローラー

問題のあったファイルとその祖先のフォルダにエラーまたは警告を示すアイコンが表示されます。エラーと警告の両方がある場合，エラーのアイコンが優先して表示されます。

問題ビュー

問題点を示すメッセージ，リソース名（ファイル名）およびフォルダ名が一覧表示されます。

4.4.8　バイナリフォーマット定義の注意事項

- XML 型からバイナリ型への変換では，可変長の単純内容要素をサイズノードおよび出現回数ノードに指定できません。
- 選択条件値のバイトサイズの指定について

  データ種別が文字列，整数，実数，固定小数部数値のどれかの選択条件ノードでは，選択条件値のバイトサイズの検証処理が実行されます。しかし，バイナリフォーマット定義ファイルのフォーマット情報にコード変換テーブルの格納ディレクトリ名が指定されている場合は，選択条件値のバイトサイズの検証処理が実行されません。選択条件値のバイトサイズについては，選択条件ノードのサイズ以下になるように指定してください。
- コード変換テーブルの格納ディレクトリ名の英大文字と英小文字の混在

  バイナリフォーマット定義ファイルで指定するコード変換テーブルの格納ディレクトリ名には，大文字と小文字の英字が使用できます。しかし，ディレクトリ名の大文字英字と小文字英字の指定が異なっていると，OS によっては動作不良を引き起こす場合があります。コード変換テーブルの格納ディレクトリ名に大文字英字と小文字英字を混在させるときは，正しいディレクトリ名を指定してください。

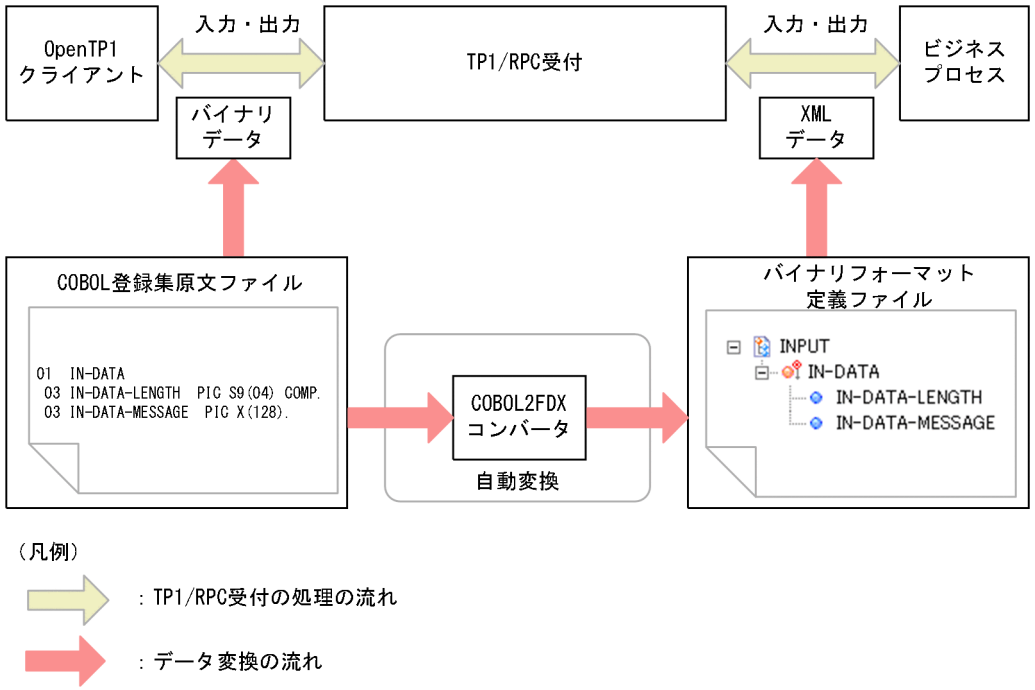
4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成

COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成とは，指定した COBOL 登録集原文ファイルをバイナリフォーマット定義ファイルに変換する機能です。

この機能を利用することで，COBOL 登録集原文ファイルからバイナリフォーマット定義ファイルの生成を自動化できます。そのため，データ変換作業の効率化を図れます。

COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成機能を利用して，TP1/RPC 受付を使用したビジネスプロセスのデータ変換のイメージを次の図に示します。

図 4-7　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成機能を利用したデータ変換（TP1/RPC 受付を使用した場合の例）



この機能では，COBOL2FDX コンバータを使用します。COBOL2FDX コンバータは，指定した COBOL 登録集原文ファイルのレコード定義を解析したあと，バイナリフォーマット定義ファイルとして出力します。

4.5.1　COBOL 登録集原文ファイルの規則

COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成機能で使用できる COBOL 登録集原文ファイルの記述形式について説明します。

この機能では，COBOL85 言語の利用を前提としています。しかし，ここで説明する規則および形式に従って記述された COBOL 登録集原文ファイルであれば使用できます。COBOL85 言語の機能および文法については，マニュアル「COBOL85 言語」を参照してください。また，COBOL85 言語の規則に従っ

ている形式のほかに，サービスプラットフォームで独自に拡張した形式（構文拡張した COBOL 登録集原文）もあわせて使用することができます。

(1) 指定できるファイル形式

この機能で使用できる COBOL 登録集原文ファイルは，固定形式（拡張子：.cbf 以外）だけです。フリー形式（拡張子：.cbf）のファイルはサポートしていません。

(2) COBOL 登録集原文ファイルの記述規則

この機能で使用できる COBOL 登録集原文ファイルは，次の記述規則に従います。

- COBOL 登録集原文ファイルの文字コードは MS932 である必要があります。
- 構文として正しいものを使用している必要があります。コンパイラが出力するメッセージに「重大エラー」または「警告エラー」が含まれていないことが前提です。
- 1～6 カラム目は，行番号としては使用されないで，6 桁の半角空白（0x20）として扱われます。エラー情報に出力される行番号としても使用されません。
- コメント行（7 カラム目が「*」，「/」）は無視されます。
- 73 カラム目以降のコメントは無視されます。
- デバッグ行（7 カラム目が「D」および「d」の行）は無視されます。
- 通常の COBOL と同様に，7 カラム目が「-」の行は継続行として扱われます。
- タブ文字は 1 個の空白文字として扱われます。
- 全角空白文字は分離符として扱われますが，「;」および「,」は分離符として扱われません。
- 拡張コード文字と標準コード文字は非等価として扱われます。
- 次の語は予約語として扱われます。それ以外の語は利用者語として扱われます。
  - COMP
  - COMP-3
  - COMPUTATIONAL
  - COMPUTATIONAL-3
  - DISPLAY
  - FILLER
  - IS
  - NATIONAL
  - OCCURS
  - PIC
  - PICTURE

- TIMES
- USAGE

(3) 構文拡張した COBOL 登録集原文を使用する場合の記述規則の変更点

サービスプラットフォームで独自に構文拡張した COBOL 登録集原文を使用する場合，COBOL 登録集原文（COBOL85 言語の形式）の記述規則とは異なります。

以降で，COBOL 登録集原文（COBOL85 言語の形式）の記述規則との変更点を説明します。

(a) プログラム識別番号領域の撤廃

プログラム識別番号領域の撤廃によって，73 カラム目以降もソースコードを記述できます。

COBOL 登録集原文では，73 カラム目以降はプログラム識別番号領域と呼ばれ，この領域に記述したコメントは無視されます。しかし，構文拡張した COBOL 登録集原文を使用する場合，次のことを考慮し，プログラム識別番号領域を撤廃しています。

- 指定できるオプションの増加に伴い，コード量が増加する。
- COBOL 登録集原文ファイルを Excel ファイルから生成する場合，1 行が短いと生成処理の難易度が高くなる。

(b) 予約語の追加

COBOL 登録集原文の構文拡張に伴い，予約語を追加しています。

追加した予約語を次に示します。

- BINARY
- BIT
- BY
- COMMA
- DATA
- DELETE
- DEPENDING
- JUST
- JUSTIFIED
- LEFT
- NULL
- OF
- ON

- RIGHT
- SIGN
- SIZE
- SPACE
- STRING
- TO
- UNTIL
- ZERO

(c) 構文拡張した COBOL 登録集原文共通の記述規則

各構文で共通の記述規則を次に示します。

●文字列定数の記述規則

文字列定数は，次のように文字列を「'」で囲みます。

```
’文字列’
```

なお，「'」自身を指定する場合は，「''」と記述してください。

●16 進定数の記述規則

16 進定数は，次のように 16 進数を「X'」と「'」で囲みます。

```
X’16進数’
```

●ノードとして指定するデータ項目の記述規則

ノードとして指定するデータ項目は，次のように記述します。

```
データ名A（OF データ名B)*
```

この記述は，データ A がデータ B 内に存在することを意味しています。データ B はデータ A の親だけではなく，先祖のデータ項目を指定できます。ただし，データ A を一意に特定できない場合はエラーになります。

これを基に，データ A，データ B およびデータ C が次に示す関係の場合の OF を用いた定義例を次に示します。

パターン 1

```
C
 └B
   └A
```

定義例	定義の可否
A	定義できる
A OF B	定義できる

定義例	定義の可否
A OF C	定義できる
A OF B OF C	定義できる
A OF C OF B	定義できない（C は B の子孫でないためエラーになる）

パターン 2

```
C
 └A
 └B
   └A
```

定義例	定義の可否
A	定義できない（A を一意に特定できないためエラーになる）
A OF B	定義できる
A OF C	定義できない（A を一意に特定できないためエラーになる）
A OF B OF C	定義できる
A OF C OF B	定義できない（C は B の子孫でないためエラーになる）

(d) データ名の最大長の拡張

データ名の最大長は COBOL の仕様に合わせて 30 文字としていましたが，バイナリフォーマット定義のノード名と同様に 256 文字に拡張しました。

ノード名の最大長の拡張に伴い，構文チェックも次のように変更になります。

- COBOL 登録集原文のデータ名が 256 文字を超えている場合は構文エラーになります。
- COBOL 登録集原文を解析した結果，データ名が重複している場合，データ項目設定ページでデータ名が重複しないように別名が付加されます。この別名はデータ名に連番を付与して作成されます。別名の付加によって 256 文字を超えてしまう場合，データ名の種別によって別名の付加方法が次のとおり異なります。

データ名の種別	別名の付加方法
NCName 型	別名を付加しません。※1
NCName 型以外	データ名のうち，NCName 型に適応しない文字を'_'に変更し，別名を付加します。※2 ただし，連番は付与されません。

注※1

データ名重複エラーのままとなります。

注※2

データ名が 256 文字を超えた場合は構文エラーとなります。

(e)　データ項目名の設定

数字だけの文字列は指定できません。指定した場合は，構文エラーとなります。

4.5.2　データ項目の設定（COBOL85 言語の形式を使用する場合）

COBOL85 言語の形式を使用する場合のデータ項目の設定について説明します。

(1)　データ項目の記述形式

英字項目や数字項目などのデータ項目の記述は，この項で説明する規則に従っている必要があります。規則に従っていない場合は，構文エラーとなります。

COBOL 登録集原文からバイナリフォーマット定義ファイルを生成する際に，個別文字コードを指定する場合のデータ項目の記述形式については，「付録 P　バイナリデータ内に文字コードが混在する場合の対応」を参照してください。

(a)　データ項目の書き方

COBOL のデータ項目の書き方を次に示します。

```
レベル番号 {データ名 | FILLER}
 [ {PICTURE | PIC} [IS] 文字列]
 [ [USAGE [IS]]  {COMPUTATIONAL-3 | COMP-3 | COMPUTATIONAL | COMP | DISPLAY | NATIONAL} ]
 [OCCURS 整数 [TIMES]] .
```

(b)　構文規則

1. レベル番号は，1～2 桁の符号なし整数で 1～49 でなければなりません。
2. データ名の文字数は 1～30 でなければなりません。
3. データ項目および FILLER 項目の数は 32,760 個を超えてはなりません。
4. OCCURS 句に指定する整数の値は 1～2,147,483,647 でなければなりません。
5. PICTURE 句に指定できる文字は「表 4-6　PICTURE 句に指定できる文字」の内容だけです。
6. レベル番号，データ名，PICTURE 句，USAGE 句，および OCCURS 句の構文規則で，1.～5.の構文規則で規定されている規則以外は，COBOL85 言語の構文規則に従います。
7. USAGE 句の構文規則を次に示します。
   - 集団項目では USAGE は指定できません。
   - 数値項目で DISPLAY，NATIONAL を指定した場合は無視されます。
   - 数値項目以外で USAGE を指定した場合は無視されます。
8. データ項目は，1 つのレコード（レベル番号 01 の集団項目），およびその従属データ項目だけでなければなりません。

9. COPY 文は指定できません。

表 4-6　PICTURE 句に指定できる文字

項番	データ型		PICTURE 句に指定できる文字		小数部	使用できる桁数の範囲（小数部含む）
			文字	桁数		
1	英字項目		「A」	1	－	1～2,147,483,647
2	英数字項目		「A」，「X」，「9」	1		
3	数字項目	外部 10 進形式	「9」	1	「V」以降の文字	1～18
			「S」，「V」	0		
4		内部 10 進形式	外部 10 進形式と同様			
5		2 進形式	「9」	1	×	
			「S」	0		
6	英数字編集項目		「A」，「X」，「9」，「B」，「0」，「/」	1	－	1～2,147,483,647
7	数字編集項目		「B」，「/」，「Z」，「0」，「9」，「,」，「.」，「*」，「+」，「-」，通貨編集文字	1	×	1～249
			「CR」，「DB」	2		
			「V」	0		
8	日本語項目		「N」	2	－	2～32,766
9	日本語編集項目		「N」，「B」	2		

（凡例）

－：該当する項目はありません。

×：この機能ではサポートしていません。

注意

通常の COBOL と同様に，各文字の直後に「(nn)」（nn は 1～2,147,483,647 の数値）が指定された場合は，文字の反復回数となります。

(c)　注意事項

PICTURE 句で指定する通貨編集文字や「Z」，「0」などの編集文字は桁数を算出する目的でだけ使用されます。データ項目に通貨編集文字などを埋め込む編集機能はサポートしていません。

(2)　データ型の対応

COBOL のデータ型と変換後のバイナリフォーマット定義ファイルのデータ型の対応を次の表に示します。

表 4-7　COBOL のデータ型とバイナリフォーマット定義ファイルのデータ型の対応

項番	COBOL のデータ型			バイナリフォーマット定義ファイルのデータ型			TP1/COBOL adapter でのデータ型
				データ種別	埋め字	XML スキーマ型	
1	英字項目			文字列	スペース	string	文字列データ（string）
2	英数字項目			文字列	スペース	string	文字列データ（string）
3	数字項目	外部 10 進形式		ゾーン形式数値	－	decimal	10 進データ（BigDecimal）
4		内部 10 進形式		パック形式数値	－	decimal	×
5		2 進形式	小数点なし	符号付（無）2 進数	－	integer	Short データ（Short）～Long データ（Long）
6			小数点あり	×			10 進データ（BigDecimal）
7		外部浮動小数点項目		×			×
8		内部浮動小数点項目	4 バイト				単精度データ（Float）
9			8 バイト				倍精度データ（Double）
10	英数字編集項目			文字列	スペース	string	文字列データ（string）
11	数字編集項目			文字列	スペース	string	文字列データ（string）
12	指標データ項目			×			×
13	日本語項目			文字列	スペース	string	文字列データ（string）
14	日本語編集項目			文字列	スペース	string	文字列データ（string）
15	外部ブール項目			×			×
16	内部ブール項目			×			×

（凡例）

－：該当する項目はありません。

×：この機能ではサポートしていません。

(3)　要素名の設定

COBOL のデータ項目に別名を設定している場合，設定した別名がバイナリフォーマット定義ファイルの要素名となります。別名を設定していない場合，変換前の COBOL のデータ名が，そのままバイナリフォーマット定義ファイルの要素名となります。

要素名に別名を設定する方法については，「4.5.4(3)　データ項目の設定」を参照してください。

(4) 符号の設定

バイナリフォーマット定義ファイルのデータ型に変換するときに，ゾーン形式数値およびパック形式数値を生成する場合は，符号の設定が「カスタム」の状態で生成し，単純内容要素ダイアログの項目に設定します。単純内容要素ダイアログの項目に設定される内容を次の表に示します。

表 4-8 属性の設定内容

項番	属性	設定内容
1	符号の設定	カスタム
2	符号の有無	• PICTURE 文字列で「S」が指定されている場合：あり • それ以外：なし
3	符号の種別	符号ビット
4	符号の位置	後

(5) COBOL のデータ型のサイズの算出方法

データ型のサイズ（バイト）は，次の表に示す方法で算出されます。

表 4-9 データ型のサイズの算出方法

COBOL のデータ型	サイズ（バイト）
英字項目	桁数と同じです。
英数字項目	
外部 10 進形式	
内部 10 進形式	((桁数) / 2) + 1
2 進形式	桁数から算出します。 • 1 桁～4 桁の場合：2 • 5 桁～9 桁の場合：4 • 10 桁～18 桁の場合：8
英数字編集項目	桁数と同じです。
数字編集項目	
日本語項目	
日本語編集項目	

4.5.3 データ項目の設定（サービスプラットフォームで独自に拡張した形式を使用する場合）

サービスプラットフォームで独自に拡張した形式を使用する場合のデータ項目の設定について説明します。

(1) 集団項目の記述形式

集団項目の記述形式を次に示します。

```
レベル番号 {データ名 | FILLER}
[{CHARSET {'MS932' | 'UTF8' | 'UTF16_BIG' | 'UTF16_LITTLE' | 'JIS' | 'EUC' | 'EUC_HJ' | 'KEIS_EBCDI
K' | 'KEIS_EBCDIC' | 'KEIS' | 'IBM_EBCDIC_LATIN' | 'IBM_EBCDIC_KANA' | 'IBM_CODE' | 'JEF_EBCDIK' |
'JEF_EBCDIC' | 'JEF' | 'CUSTOM'} | JAVA-ENCODING-CHARSET 文字コード}]
[ESCAPE-CHARACTER エスケープ文字]
[LINEFEED {LF | CRLF}]
[DELETE EXTRA-SEPARATORS]
[(SEPARATOR セパレータ名 セパレータ値)*]
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES] | DEPENDING ON ノード [ABSOLUTE]}]
[SIZE DEPENDING ON ノード [ABSOLUTE] [CALC {SYSTEM | DATA | NODE}]]
[CHOOSE {DEPENDING ON ノード [ABSOLUTE] | BY START-SEPARATOR}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[START-SEPARATOR 開始セパレータ名]
[MIDDLE-SEPARATOR 中間区切りセパレータ名]
[TERMINAL-SEPARATOR 終了セパレータ名]
[{LITTLE-ENDIAN | BIG-ENDIAN}]
[CODE-TABLE-DIRECTORY-NAME コード変換テーブルの格納ディレクトリ名]
.
```

次のオプションはレコードで記述できます。

- JAVA-ENCODING-CHARSET 句
- ESCAPE-CHARACTER 句
- LINEFEED 句
- DELETE EXTRA-SEPARATORS 句
- SEPARATOR 句
- LITTLE-ENDIAN 句
- BIG-ENDIAN 句
- CODE-TABLE-DIRECTORY-NAME 句

また，次のオプションはレコードで記述できません。

- OCCURS 句
- SIZE 句
- CALC 句

(2)　基本項目の記述形式

基本項目の記述方法はデータ型によって異なります。

基本項目の記述形式を次に示します。

(a)　PICTURE 句によるデータ型指定

PICTURE 句で指定できるのは，固定長のデータ型だけです。

PICTURE 句に指定できる文字については，「4.5.2(1)　データ項目の記述形式」を参照してください。

```
レベル番号 {データ名 | FILLER}
{PICTURE | PIC} [IS] 文字列
[[USAGE [IS]] {COMPUTATIONAL-3 | COMP-3 | COMPUTATIONAL | COMP | DISPLAY | NATIONAL}]
[{JUSTIFIED | JUST} {LEFT | RIGHT}]
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES] | DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[CHARSET {'MS932' | 'UTF8' | 'UTF16_BIG' | 'UTF16_LITTLE' | 'JIS' | 'EUC' | 'EUC_HJ' | 'KEIS_EBCDIK
' | 'KEIS_EBCDIC' | 'KEIS' | 'IBM_EBCDIC_LATIN' | 'IBM_EBCDIC_KANA' | 'IBM_CODE' | 'JEF_EBCDIK' | '
JEF_EBCDIC' | 'JEF' | 'CUSTOM'}]
[EMBEDDING-CHARACTER {SPACE | NULL [TERMINATION]}]
[DELETE UNNECESSARY-CHARACTER]
.
```

(b)　STRING 句によるデータ型の指定

文字列型の単純内容要素に変換する場合，STRING 句で指定します。

```
レベル番号 {データ名 | FILLER}
STRING SIZE {整数 | DEPENDING ON ノード [ABSOLUTE]
[CALC{SYSTEM | DATA | NODE}] | UNTIL EOD}
[{JUSTIFIED | JUST} {LEFT | RIGHT}]
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES] | DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[CHARSET {'MS932' | 'UTF8' | 'UTF16_BIG' | 'UTF16_LITTLE' | 'JIS' | 'EUC' | 'EUC_HJ' | 'KEIS_EBCDIK
' | 'KEIS_EBCDIC' | 'KEIS' | 'IBM_EBCDIC_LATIN' | 'IBM_EBCDIC_KANA' | 'IBM_CODE' | 'JEF_EBCDIK' | '
JEF_EBCDIC' | 'JEF' | 'CUSTOM'}]
[EMBEDDING-CHARACTER {SPACE | NULL [TERMINATION]}]
[DELETE UNNECESSARY-CHARACTER]
[CONVERT-LOW-VALUE]
.
```

(c)　INTEGER 句による整数型指定

整数型の単純内容要素に変換する場合，INTEGER 句で指定します。

```
レベル番号 {データ名 | FILLER}
INTEGER SIZE {整数 | DEPENDING ON ノード [ABSOLUTE]
[CALC{SYSTEM | DATA | NODE}] | UNTIL EOD}
[{JUSTIFIED | JUST} {LEFT | RIGHT}]
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES] | DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
```

```
[CHARSET {'MS932'|'UTF8'|'UTF16_BIG'|'UTF16_LITTLE'|'JIS'|'EUC'|'EUC_HJ'|'KEIS_EBCDIK
'|'KEIS_EBCDIC'|'KEIS'|'IBM_EBCDIC_LATIN'|'IBM_EBCDIC_KANA'|'IBM_CODE'|'JEF_EBCDIK'|'
JEF_EBCDIC'|'JEF'|'CUSTOM'}]
[EMBEDDING-CHARACTER {ZERO|SPACE}]
[DELETE UNNECESSARY-CHARACTER]
[SIGN {NONE|MINUS-ONLY|ALWAYS}]
[CONVERT-LOW-VALUE]
.
```

(d) ZONE 句および PACK 句によるゾーン形式数値およびパック形式数値の指定

ゾーン形式数値またはパック形式数値の単純内容要素に変換する場合，ZONE 句または PACK 句で指定します。

```
レベル番号 {データ名|FILLER}
{ZONE|PACK} SIZE {整数|DEPENDING ON ノード [ABSOLUTE]
[VALUE-UNIT※ {BYTES|DIGITS}] [CALC{SYSTEM|DATA|NODE}]|UNTIL EOD}
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES]|DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[DECIMAL-DIGITS 小数桁]
[SIGN {NONE|ALWAYS}]
.
```

注※

VALUE-UNIT はパック形式数値の場合だけ指定できます。

(e) BINARY 句による符号付 2 進数整数型または符号無 2 進数整数型の指定

符号付 2 進数整数型または符号無 2 進数整数型の単純内容要素に変換する場合，BINARY 句で指定します。

```
レベル番号 {データ名|FILLER}
BINARY SIZE 整数
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES]|DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[SIGN {NONE|ALWAYS}]
.
```

(f) BYTE 句によるバイト列指定

バイト列型の単純内容要素に変換する場合，BYTE 句で指定します。

```
レベル番号 {データ名|FILLER}
BYTE SIZE {整数|DEPENDING ON ノード [ABSOLUTE]
[CALC{SYSTEM|DATA|NODE}]|UNTIL EOD}
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES]|DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
[BASE64]
.
```

(g) BIT 句によるビット列指定

ビット列型の単純内容要素に変換する場合，BIT 句で指定します。

```
レベル番号 {データ名 | FILLER}
BIT SIZE {整数}
[OCCURS {最小出現回数 [TO UNBOUNDED] [TIMES] | DEPENDING ON ノード [ABSOLUTE]}]
[CASE 選択条件値 [CASE-DELIMITER 区切り文字]]
.
```

(3) フォーマットの設定方法

レコードの集団項目にオプションの句を定義することによって，バイナリフォーマット定義のフォーマットの各項目を設定できます。設定項目，句および省略時のデフォルト値を次に示します。

表 4-10 フォーマットの設定項目

設定項目		対応する COBOL の句	デフォルト値
文字コード	Code Converter が提供するエンコーダーとデコーダーで使用する文字コード	CHARSET	(ウィザードで設定した文字コード)
	Java が提供するエンコーダーとデコーダーで使用する文字コード	JAVA-ENCODING-CHARSET	windows-31j
セパレータ	エスケープ文字	ESCAPE-CHARACTER	(設定しない)
	改行文字	LINEFEED	CR+LF
	不要なセパレータの削除	DELETE EXTRA-SEPARATORS	無効
	セパレータの定義	SEPARATOR	－
2 進整数のエンディアン	リトルエンディアン	LITTLE-ENDIAN	(ウィザードで設定したエンディアン)
	ビッグエンディアン	BIG-ENDIAN	
コード変換テーブル	格納ディレクトリ名	CODE-TABLE-DIRECTORY-NAME	(設定しない)

(凡例)

－：該当する値はありません。

(a) 文字コード

バイナリフォーマット定義の文字コードを指定できます。

●Code Converter が提供するエンコーダーとデコーダーを使用する場合

文字コードを CHARASET 句で指定します。

文字コードはウィザード上でも指定できますが，COBOL 登録集原文で設定した内容が優先されます。

●Java が提供するエンコーダーとデコーダーを使用する場合

文字コードを JAVA-ENCODING-CHARSET 句で指定します。

文字コードは java.nio API 用の正準名または別名で設定します。ただし，次に示す文字コード以外の文字コードが設定された場合の動作は保障されません。

- windows-31j
- UTF-8
- UTF-16BE
- UTF-16LE
- ISO-2022-JP
- EUC-JP
- x-IBM939
- x-IBM930

次に文字コードオプションの書式を示します。

```
[{CHARSET {'MS932'|'UTF8'|'UTF16_BIG'|'UTF16_LITTLE'|'JIS'|'EUC'|'EUC_HJ'|'KEIS_EBCDIK'|'KEIS_EBCDIC'|'KEIS'|'IBM_EBCDIC_LATIN'|'IBM_EBCDIC_KANA'|'IBM_CODE'|'JEF_EBCDIK'|'JEF_EBCDIC'|'JEF'|'CUSTOM'}|JAVA-ENCODING-CHARSET 文字コード}]
```

(b) セパレータの指定

バイナリフォーマット定義のセパレータを指定できます。

●エスケープ文字を指定する場合

エスケープ文字を，ESCAPE-CHARACTER 句で指定します。

ESCAPE-CHARACTER 句の書式を次に示します。

```
ESCAPE-CHARACTER エスケープ文字
```

●改行文字を指定する場合

セパレータで使用する改行コードを，LINEFEED 句で指定します。

LINEFEED 句の書式を次に示します。

```
LINEFEED {LF|CRLF}
```

改行コードには，次のどちらかを指定できます。

- LF：LF（0x0A）
- CRLF：CR + LF（0x0D0A）

●不要なセパレータの削除を指定する場合

不要なセパレータの削除を，DELETE EXTRA-SEPARATORS 句で指定します。

DELETE EXTRA-SEPARATORS 句の書式を次に示します。

```
DELETE EXTRA-SEPARATORS
```

●セパレータを定義する場合

セパレータを定義する場合，SEPARATOR 句で指定します。

SEPARATOR 句の書式を次に示します。

```
SEPARATOR セパレータ名 セパレータ値
```

セパレータ名は文字列，セパレータ値は 16 進定数で指定します。

(c) 2 進整数のエンディアン

バイナリフォーマット定義の 2 進整数のエンディアンを指定できます。

2 進整数のエンディアンはウィザード上で指定できますが，COBOL 登録集原文で設定した内容が優先されます。

●リトルエンディアンを指定する場合

2 進整数のエンディアンオプションでリトルエンディアンを指定するときの書式を次に示します。

```
LITTLE-ENDIAN
```

●ビッグエンディアンを指定する場合

2 進整数のエンディアンオプションでビッグエンディアンを指定するときの書式を次に示します。

```
BIG-ENDIAN
```

(d) コード変換テーブル

バイナリフォーマット定義のコード変換テーブルの格納ディレクトリ名を指定できます。

コード変換テーブルの格納ディレクトリ名は，CODE-TABLE-DIRECTORY-NAME 句で指定します。

CODE-TABLE-DIRECTORY-NAME 句の書式を次に示します。

```
CODE-TABLE-DIRECTORY-NAME コード変換テーブルの格納ディレクトリ名
```

(4) 複合内容要素の設定方法

レコード以外の集団項目にオプションの句を定義することによって，バイナリフォーマット定義ファイルの複合内容要素の各項目を設定できます。設定項目，句および省略時のデフォルト値を次に示します。

表 4-11 複合内容要素の設定項目

設定項目	対応する COBOL の句	デフォルト値
出現回数	OCCURS	1
サイズノード	SIZE	(設定しない)

設定項目	対応する COBOL の句	デフォルト値
選択構成	CHOOSE	逐次構成
選択条件値	CASE	（設定しない）
開始セパレータ	START-SEPARATOR	（設定しない）
中間区切りセパレータ	MIDDLE-SEPARATOR	（設定しない）
終了セパレータ	TERMINAL-SEPARATOR	（設定しない）

(a) 出現回数

出現回数を設定する場合，OCCURS 句で指定します。

OCCURS 句の書式を次に示します。

●出現回数を固定とする場合

```
OCCURS 出現回数［TIMES］
```

出現回数には，1～2147483647 の整数を指定します。

●出現回数をデータの終端までとする場合

```
OCCURS 最小出現回数 TO UNBOUNDED［TIMES］
```

最小出現回数には，0 または 1 を指定できます。

1 を指定した場合，［構成要素タブ］-［出現回数］-［データの省略不可］オプションが有効になります。

0 を指定した場合，［構成要素タブ］-［出現回数］-［データの省略不可］オプションが無効になります。

●出現回数を出現回数ノード指定とした場合

出現回数ノード指定の場合，出現回数ノードに指定する単純内容要素のデータ項目名に対応するデータ項目名を設定します。

```
OCCURS DEPENDING ON 出現回数指定データ項目名［ABSOLUTE］
```

ABSOLUTE 句を指定した場合は絶対パスとして扱われ，指定しない場合は基準パスとして扱われます。

(b) サイズ指定

サイズノードを設定する場合，SIZE 句で指定します。

SIZE 句の書式を次に示します。

●サイズノードで「サイズ算出方法」を使用しない場合

```
SIZE DEPENDING ON サイズノード指定データ項目名［ABSOLUTE］
```

●サイズノードで「サイズ算出方法」を使用する場合

```
SIZE DEPENDING ON サイズノード指定データ項目名［ABSOLUTE］
  CALC {SYSTEM | DATA | NODE}
```

ABSOLUTE 句を指定した場合は絶対パスとして扱われ，指定しない場合は基準パスとして扱われます。

なお，CALC 句を記述しなかった場合の動作は次のとおりです。

- サイズノードが可変長の数値型の場合
  サイズ算出方法の指定なしとして扱われます。
- サイズノードが固定長の数値型の場合
  CALC SYSTEM を指定したとして扱われます。
- サイズノードがビット列型の場合
  CALC DATA を指定したとして扱われます。
  ビット列型のサイズノードに対して CALC 句を指定した場合，構文エラーになります。

(c) 選択構成

構成区分に選択構成を設定する場合，CHOOSE 句で指定します。CHOOSE 句を記述しない場合，構成区分は逐次構成となります。

CHOOSE 句の書式を次に示します。

●選択種別に「選択条件ノード」を使用する場合

```
CHOOSE DEPENDING ON 選択条件ノード［ABSOLUTE］
```

●選択種別に「構成要素の開始セパレータ」を使用する場合

```
CHOOSE BY START-SEPARATOR
```

(d) 選択条件値

選択条件値を設定する場合，CASE 句で指定します。また，複数の選択条件値を設定するときは，CASE-DELIMITER 句で区切り文字を指定します。

CASE 句の書式を次に示します。

●選択条件値を設定する場合

```
CASE 選択条件値
```

●区切り文字を指定して複数の選択条件値を設定する場合

```
CASE 選択条件値［CASE-DELIMITER 区切り文字］
```

CASE-DELIMITER 句に指定した区切り文字を使用して，複数の選択条件値を設定できます。

COBOL 登録集原文の定義例については「(7) 複数の選択条件値を設定する場合の COBOL 登録集原文の定義例」を参照してください。

なお，選択条件値と区切り文字は文字列定数であるため「'」で囲んで記述してください。バイト列，およびビット列の選択条件値を指定する場合も，16 進数の値を「'」で囲む必要があります（16 進定数ではあ

りません)。また，区切り文字を使用して複数の選択条件値を指定する場合は，選択条件値全体を「'」で囲んで記述してください。

(e) 開始セパレータ

開始セパレータを設定する場合，START-SEPARATOR 句で指定します。

START-SEPARATOR 句の書式を次に示します。

```
START-SEPARATOR 開始セパレータ名
```

(f) 中間区切りセパレータ

中間区切りセパレータを設定する場合，MIDDLE-SEPARATOR 句で指定します。

MIDDLE-SEPARATOR 句の書式を次に示します。

```
MIDDLE-SEPARATOR 中間区切りセパレータ名
```

(g) 終了セパレータ

終了セパレータを設定する場合，TERMINAL-SEPARATOR 句で指定します。

TERMINAL-SEPARATOR 句の書式を次に示します。

```
TERMINAL-SEPARATOR 終了セパレータ名
```

(5) 単純内容要素の設定方法

基本項目にオプションの句を定義することによって，バイナリフォーマット定義ファイルの単純内容要素の各項目を設定できます。

なお，PICTURE 句で型指定した場合と，PICTURE 句以外で型指定した場合とで設定項目，句および省略時のデフォルト値が異なります。

表 4-12 単純内容要素の設定項目（PICTURE 句で型指定した場合）

設定項目	対応する COBOL の句	デフォルト値
出現回数	OCCURS	1
選択条件値	CASE	（設定しない）
個別文字コード	CHARSET	（設定しない）
左右寄せ	JUSTIFIED	左
埋め字	EMBEDDING-CHARACTER	スペース（空白文字）
不要文字の削除	DELETE UNNECESSARY-CHARACTER	不要文字を削除しない

表 4-13　単純内容要素の設定項目（PICTURE 句以外で型指定した場合）

設定項目	対応する COBOL の句	デフォルト値
出現回数	OCCURS	1
選択条件値	CASE	（設定しない）
個別文字コード	CHARSET	（設定しない）
左右寄せ	JUSTIFIED	右（整数型の場合） 左（文字列型の場合）
埋め字	EMBEDDING-CHARACTER	スペース（空白文字）
不要文字の削除	DELETE UNNECESSARY-CHARACTER	不要文字を削除しない
符号の有無	SIGN	なし
小数部桁数	DECIMAL-DIGITS	0
エンコード形式	BASE64	hexBinary
LOW-VALUE の変換	CONVERT-LOW-VALUE	変換しない

(a)　出現回数

［単純内容要素］の［出現回数］を設定します。可変長（[データの終端まで]，［出現回数ノード］）も含みます。

出現回数を設定する場合，OCCURS 句で指定します。

OCCURS 句の書式を次に示します。

●出現回数を固定とする場合

```
OCCURS 出現回数［TIMES］
```

出現回数には，1～2147483647 の整数を指定します。

●出現回数をデータの終端までとする場合

```
OCCURS 最小出現回数 TO UNBOUNDED［TIMES］
```

最小出現回数には，0 または 1 を指定できます。

1 を指定した場合，［構成要素タブ］-［出現回数］-［データの省略不可］オプションが有効になります。

0 を指定した場合，［構成要素タブ］-［出現回数］-［データの省略不可］オプションが無効になります。

●出現回数を出現回数ノード指定とした場合

出現回数ノード指定の場合，出現回数ノードに指定する単純内容要素のデータ項目名に対応するデータ項目名を設定します。

```
OCCURS DEPENDING ON 出現回数指定データ項目名［ABSOLUTE］
```

ABSOLUTE 句を指定した場合は絶対パスとして扱われ，指定しない場合は基準パスとして扱われます。

(b)　選択条件値

選択条件値を設定する場合，CASE 句で指定します。

CASE 句の書式を次に示します。

```
CASE 選択条件値
```

なお，バイト列，およびビット列の選択条件値を指定する場合，16 進数の定数を「'」で囲む必要があります。

(c)　データ種別

データ種別の判定は，PICTURE 句で型指定された基本項目の場合と，PICTURE 句以外で型指定された基本項目の場合で，次のように異なります。

●PICTURE 句で型指定された基本項目の場合

PICTURE 句の文字列によって判定します。

●PICTURE 句以外で型指定された基本項目の場合

SIZE 句の前にある句によって判定します。

SIZE 句の前の句と単純内容要素のデータ種別の関係を次に示します。

SIZE 句の前の句	単純内容要素のデータ種別
INTEGER	整数
STRING	文字列
ZONE	ゾーン形式数値
PACK	パック形式数値
BYTE	バイト列
BINARY	符号付（無）2 進整数

(d)　サイズ指定

[単純内容要素] の［サイズ］を指定します。可変長（[データの終端まで]，[サイズノード]）も含みます。

●PICTURE 句での指定

PICTURE 句での指定は，PICTURE 句の文字列によって異なります。

PICTURE 句に指定できる文字については，「4.5.2(1)　データ項目の記述形式」を参照してください。

●PICTURE 句以外での指定

PICTURE 句以外の句で型指定された基本項目から変換する単純内容要素のサイズは，固定値のほか可変値（サイズノード指定，データの終端まで）にも対応します。

固定値の場合はバイト単位で記述します。ただし，ビット列型の場合はビット単位で記述します。

次に固定値の場合の書式を示します。

```
SIZE 整数
```

整数に指定できる値を次に示します。

型指定句	SIZE 句で指定できる整数値
INTEGER，ZONE，BYTE	1～2147483647
STRING	0※～2147483647
PACK	1～1073741823
BINARY	1～8
BIT	1～64

注※

文字列型の SIZE 句で 0 を指定した場合，サイズ以外で指定することができる項目は選択条件値（CASE 句）だけです。サイズ，選択条件値以外の項目が指定されていた場合は構文エラー（KECT26020-E）になります。

●サイズノードの指定

サイズノードを設定する場合，SIZE 句で指定します。サイズノードの指定で，ABSOLUTE 句を指定した場合は絶対パスとして扱われ，指定しない場合は基準パスとして扱われます。

SIZE 句の書式を次に示します。

＜サイズノードで「サイズ算出方法」を使用しない場合＞

```
SIZE DEPENDING ON サイズノード指定データ項目名［ABSOLUTE］
```

＜サイズノードで「サイズ算出方法」を使用する場合＞

```
SIZE DEPENDING ON サイズノード指定データ項目名［ABSOLUTE］
   CALC {SYSTEM｜DATA｜NODE}
```

なお，CALC 句を記述しなかった場合の動作は次のとおりです。

- サイズノードが可変長の数値型の場合
  サイズ算出方法の指定なしとして扱われます。
- サイズノードが固定長の数値型の場合
  CALC SYSTEM を指定したとして扱われます。
- サイズノードがビット列型の場合
  CALC DATA を指定したとして扱われます。
  ビット列型のサイズノードに対して CALC 句を指定した場合，構文エラーになります。
- サイズノード値の単位が桁数の場合
  CALC DATA を指定したとして扱われます。

＜サイズノードで「サイズノード値の単位」を設定する場合＞

```
PACK SIZE DEPENDING ON サイズノード指定データ項目名［ABSOLUTE］VALUE-UNIT {BYTES｜DIGITS}
```

なお，VALUE-UNIT 句を記述しなかった場合の動作は次のとおりです。

- サイズノードでパック形式数値型を設定した場合
  VALUE-UNIT BYTES を指定したとして扱われます。
- サイズノードでパック形式数値型以外を設定した場合
  サイズノード値の単位の指定なしとして扱われます。

●その他の指定

サイズをデータの終端までとする場合の書式を次に示します。

```
SIZE UNTIL EOD
```

(e) 詳細設定

次の項目について説明します。

- 個別文字コード
- 左右寄せ
- 埋め字
- 不要文字
- 小数部桁数
- 符号の有無
- エンコード形式
- LOW-VALUE の変換

●個別文字コード

個別文字コードを設定する場合，CHARSET 句で指定します。

また，複合内容要素の直下にある単純内容要素の個別文字コードを一括で設定する場合，レコード以外の集団項目に CHARSET 句を指定します。

CHARSET 句の書式を示します。

```
[CHARSET {'MS932'｜'UTF8'｜'UTF16_BIG'｜'UTF16_LITTLE'｜'JIS'｜'EUC'｜'EUC_HJ'｜'KEIS_EBCDIK'｜'KEIS_EBCDIC'｜'KEIS'｜'IBM_EBCDIC_LATIN'｜'IBM_EBCDIC_KANA'｜'IBM_CODE'｜'JEF_EBCDIK'｜'JEF_EBCDIC'｜'JEF'｜'CUSTOM'}]
```

●左右寄せ

左右寄せを設定する場合，JUSTIFIED 句で指定します。

JUSTIFIED 句の書式を示します。

```
{JUSTIFIED｜JUST} {LEFT｜RIGHT}
```

なお，EMBEDDING-CHARACTER 句で ZERO が指定されている場合，LEFT は指定できません。

●埋め字

埋め字を設定する場合，EMBEDDING-CHARACTER 句で指定します。

EMBEDDING-CHARACTER 句の書式を示します。

＜文字列型の場合＞

```
EMBEDDING-CHARACTER {SPACE | NULL [TERMINATION]}
```

＜整数型の場合＞

```
EMBEDDING-CHARACTER {ZERO | SPACE}
```

埋め字には，次のどれかを指定できます。

- ZERO：0
- SPACE：空白文字
- NULL：0x00

NULL を指定した場合に，TERMINATION 句を記述すると「空文字（0x00）までをデータとする」オプションが有効となります。

なお，JUSTIFIED 句で LEFT が指定されている場合，ZERO は指定できません。

●不要文字

不要文字の設定項目に「不要文字の削除」を設定する場合，DELETE 句および UNNECESSARY-CHARACTER 句で指定します。

「不要文字の削除」を設定する場合の書式を次に示します。

```
DELETE UNNECESSARY-CHARACTER
```

DELETE 句および UNNECESSARY-CHARACTER 句を記述しない場合，不要文字の設定項目に「不要文字を削除しない」が設定されます。

●小数部桁数

小数部桁数を設定する場合，DECIMAL-DIGITS 句で指定します。

DECIMAL-DIGITS 句の書式を示します。

```
DECIMAL-DIGITS 小数部桁数
```

●符号の有無

符号の有無を設定する場合，SIGN 句で指定します。

SIGN 句の書式を示します。

＜整数型の場合＞

```
SIGN {NONE | MINUS-ONLY | ALWAYS}
```

＜パック形式数値，ゾーン形式数値，または符号付(無)2 進整数の場合＞

```
SIGN {NONE | ALWAYS}
```

符号の有無には，次のどれかを指定できます。

- NONE：符号なし
- MINUS-ONLY：負の符号だけあり
- ALWAYS：常に符号あり

●エンコード形式

バイト列のエンコード形式を設定する場合，BASE64 句で指定します。

BASE64 句の書式を次に示します。

```
BASE64
```

BASE64 句を記述しない場合，「hexBinary」が設定されます。

●LOW-VALUE の変換

LOW-VALUE の変換を設定する場合，CONVERT-LOW-VALUE 句で指定します。

CONVERT-LOW-VALUE 句の書式を次に示します。

```
CONVERT-LOW-VALUE
```

CONVERT-LOW-VALUE 句の指定がない場合，LOW-VALUE の変換をしない設定となります。

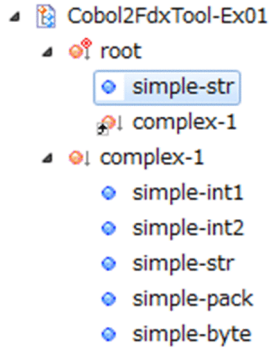
CONVERT-LOW-VALUE 句の定義がある場合でも，次の条件に一致しなければ構文エラーとなります。

- 文字列型の場合
  SIZE 句が整数（固定長かつ 0 でない），かつ EMBEDDING-CHARACTER 句が SPACE
- 整数型の場合
  SIZE 句が整数（固定長）

(6) COBOL 登録集原文からバイナリフォーマット定義への変換例

構文拡張した COBOL 登録集原文からバイナリフォーマット定義への変換例を示します。

(a)　変換例 1



●バイナリフォーマット定義の内容

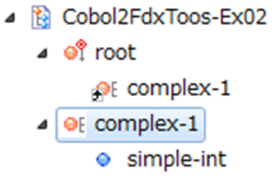
要素名	区分	設定内容
Cobol2FdxTool-Ex01	フォーマット	文字コード：UTF8 セパレータの設定： 　エスケープ文字：@ 　改行指定：LF 　不要なセパレータの削除：有効
root	複合内容要素	構成区分：逐次構成
simple-str	単純内容要素	データ種別：文字列 サイズ：1 バイト 個別文字コード：JIS 埋め字：空文字(0x00) 左右寄せ：右
complex-1	複合内容要素	構成区分：逐次構成 セパレータの定義： 　Sepa-1：0xAA 中間区切りセパレータ名：Sepa-1
simple-int1	単純内容要素	データ種別：整数 サイズ：可変（データの終端まで） 符号の有無：負だけあり 埋め字：0
simple-int2	単純内容要素	データ種別：整数 サイズ：可変（データの終端まで） 符号の有無：負だけあり 左右寄せ：左
simple-str	単純内容要素	データ種別：文字列 サイズ：可変（データの終端まで） 左右寄せ：左

要素名	区分	設定内容
simple-pack	単純内容要素	データ種別：パック形式数値 サイズ：可変（データの終端まで）
simple-byte	単純内容要素	データ種別：バイト列 サイズ：1 バイト

●COBOL 登録集原文の内容

```
 01 root CHARSET 'UTF8' ESCAPE-CHARACTER '@'
        LINEFEED LF DELETE EXTRA-SEPARATORS SEPARATOR 'Sepa-1' X'AA'.
   03 simple-str STRING SIZE 1 JUSTIFIED RIGHT CHARSET 'JIS' EMBEDDING-CHARACTER NULL.
   03 complex-1 MIDDLE-SEPARATOR 'Sepa-1'.
     05 simple-int1 INTEGER SIZE UNTIL EOD EMBEDDING-CHARACTER ZERO SIGN MINUS-ONLY.
     05 simple-int2 INTEGER SIZE UNTIL EOD JUSTIFIED LEFT SIGN MINUS-ONLY.
     05 simple-str STRING SIZE UNTIL EOD.
     05 simple-pack PACK SIZE UNTIL EOD.
     05 simple-byte BYTE SIZE 1.
```

(b)　変換例 2



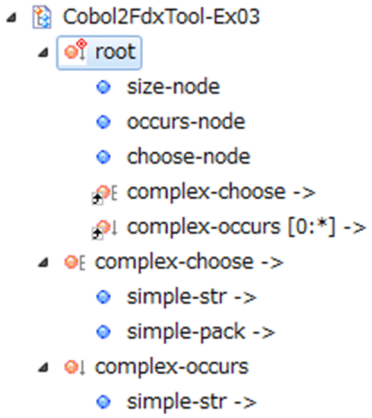
●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	構成区分：逐次構成
complex-1	複合内容要素	構成区分：選択構成 選択種別：構成要素の開始セパレータ セパレータの定義： Sepa-1：0xAA Sepa-2：0xBB Sepa-3：0xCC 開始セパレータ：Sepa-1 中間区切りセパレータ：Sepa-2 終了セパレータ：Sepa-3
simple-int	単純内容要素	データ種別：整数 サイズ：可変（データの終端まで）

●COBOL 登録集原文の内容

```
01 root SEPARATOR 'Sepa-1' X'AA' SEPARATOR 'Sepa-2' X'BB' SEPARATOR 'Sepa-3' X'CC'.
  03 complex-1 CHOOSE BY START-SEPARATOR
     START-SEPARATOR 'Sepa-1' MIDDLE-SEPARATOR 'Sepa-2'
     TERMINAL-SEPARATOR 'Sepa-3'.
     05 simple-int INTEGER SIZE UNTIL EOD.
```

(c) 変換例 3



●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	−
size-node	単純内容要素	データ種別：整数 サイズ：4 バイト
occurs-node	単純内容要素	データ種別：整数 サイズ：4 バイト
choose-node	単純内容要素	データ種別：整数 サイズ：1 バイト
complex-choose	複合内容要素	構成区分：選択構成 選択種別：選択条件ノード 選択条件ノード：root/choose-node (絶対パス)
simple-str	単純内容要素	データ種別：文字列 サイズ：可変（サイズノード） サイズノード：root/size-node (絶対パス) 選択条件値：1
simple-pack	単純内容要素	データ種別：パック形式数値 サイズ：可変（サイズノード） サイズノード：root/size-node (絶対パス)

要素名	区分	設定内容
complex-occurs	複合内容要素	構成区分逐次構成 出現回数ノード：root/occurs-node (絶対パス)
simple-str	単純内容要素	データ種別：文字列 サイズ：可変（サイズノード） サイズノード：root/size-node (絶対パス)

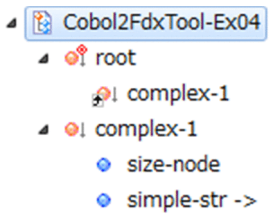
(凡例)

－：該当しません。

●COBOL 登録集原文の内容

```
 01 root.
   03 size-node INTEGER SIZE 4.
   03 occurs-node INTEGER SIZE 4.
   03 choose-node INTEGER SIZE 1.
   03 complex-choose CHOOSE DEPENDING ON choose-node ABSOLUTE.
     05 simple-str STRING SIZE DEPENDING ON size-node ABSOLUTE CASE '1'.
     05 simple-pack PACK SIZE DEPENDING ON size-node ABSOLUTE.
   03 complex-occurs OCCURS DEPENDING ON occurs-node ABSOLUTE.
     05 simple-str STRING SIZE DEPENDING ON size-node ABSOLUTE.
```

(d)　変換例 4



●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	－
complex-1	複合内容要素	－
size-node	単純内容要素	データ種別：整数 サイズ：4 バイト
simple-str	単純内容要素	データ種別：文字列 サイズ：可変（サイズノード） サイズノード：complex-1/size-node (基準パス)

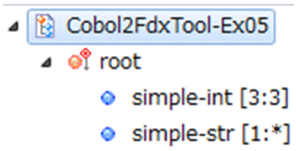
（凡例）

－：該当しません。

●COBOL 登録集原文の内容

```
 01 root.
   03 complex-1.
     05 size-node INTEGER SIZE 4.
     05 simple-str STRING SIZE DEPENDING ON size-node.
```

(e)　変換例 5



●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	－
simple-int	単純内容要素	データ種別：整数 サイズ：8 バイト 出現回数：固定（3 回）
simple-str	単純内容要素	データ種別：文字列 サイズ：8 バイト 出現回数：データの終端まで （データの省略不可）

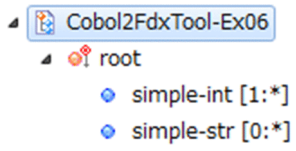
（凡例）

－：該当しません。

●COBOL 登録集原文の内容

```
01 root.
    05 simple-int INTEGER SIZE 8 OCCURS 3.
    05 simple-str STRING SIZE 8 OCCURS 1 TO UNBOUNDED.
```

(f)　変換例 6



●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	－
simple-int	単純内容要素	データ種別：整数 サイズ：8 バイト 出現回数：データの終端まで （データの省略不可）
simple-str	単純内容要素	データ種別：文字列 サイズ：8 バイト 出現回数：データの終端まで （データの省略可）

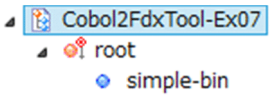
（凡例）

－：該当しません。

●COBOL 登録集原文の内容

```
01 root.
    05 simple-int INTEGER SIZE 8 OCCURS 1 TO UNBOUNDED.
    05 simple-str STRING SIZE 8 OCCURS 0 TO UNBOUNDED.
```

(g)　変換例 7



●バイナリフォーマット定義の内容

要素名	区分	設定内容
root	複合内容要素	2 進整数のエンディアン：ビッグエンディアン
simple-bin	単純内容要素	データ種別：符号無 2 進整数 サイズ：4 バイト

●COBOL 登録集原文の内容

```
01 root BIG-ENDIAN.
    05 simple-bin PIC 9(8) COMP.
```

(7)　複数の選択条件値を設定する場合の COBOL 登録集原文の定義例

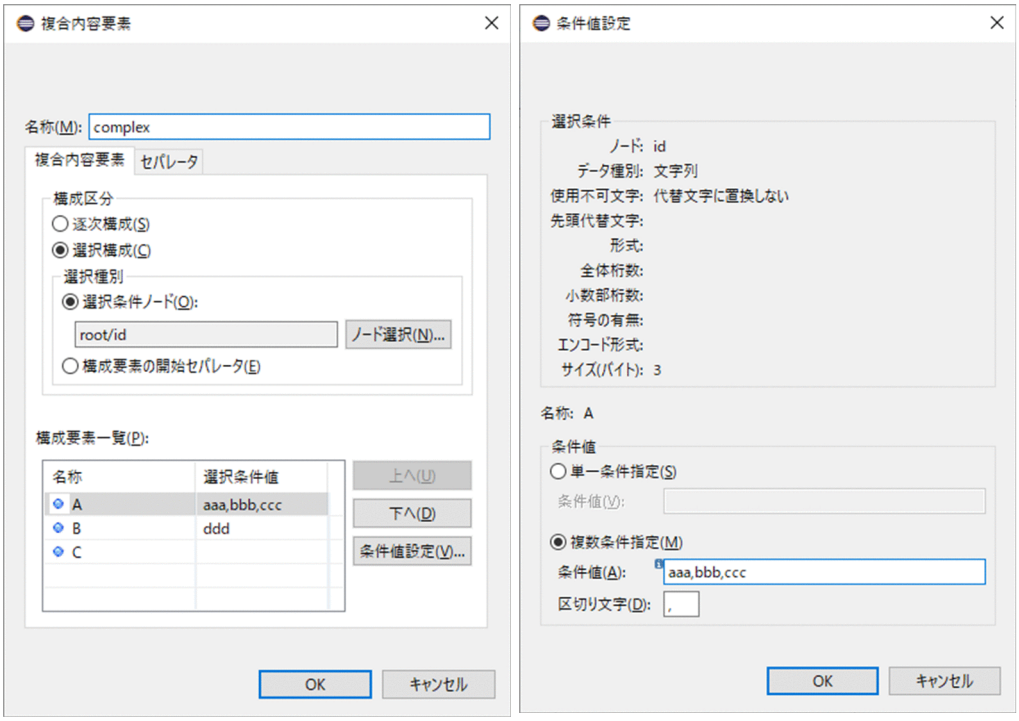
COBOL 登録集原文では，集団項目に CHOOSE 句（選択構成）を記述し，DEPENDING ON 句で選択条件ノードを指定してください。次に集団項目の配下の項目で CASE 句，および CASE-DELIMITER 句を使用して選択条件値を指定してください。定義例を次に示します。

●COBOL 登録集原文の内容

```
01 root.
  03 id STRING SIZE 3.
  03 complex CHOOSE DEPENDING ON id.
    05 A STRING SIZE 1 CASE 'aaa,bbb,ccc' CASE-DELIMITER ','.
    05 B INTEGER SIZE 1 CASE 'ddd'.
    05 C BYTE SIZE 1.
```

上記の COBOL 登録集原文でバイナリフォーマット定義ファイルを生成した場合，バイナリフォーマット定義エディタでは次の設定となります。

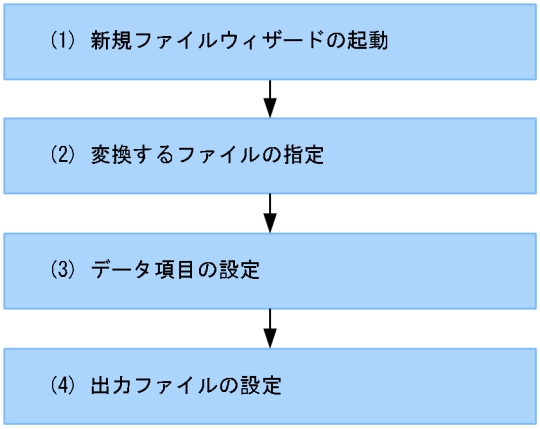
図 4-8　COBOL 登録集原文から生成したバイナリフォーマット定義



4.5.4　バイナリフォーマット定義ファイルの生成方法

COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成機能を利用したデータ変換の操作の流れを次に示します。

図 4-9　データ変換の操作の流れ



データ変換に関する操作について次に説明します。

(1)　新規ファイルウィザードの起動

新規ファイルウィザードの起動方法について説明します。

1. 次のどちらかの方法で新規ファイルウィザードを開きます。
   - Eclipse のメニューから［ファイル］－［新規］－［その他］を選択します。
   - パッケージ・エクスプローラーで任意の場所を右クリックし，［新規］－［その他］を選択します。
2. 新規ファイルウィザードで，［HCSCTE フォーマット定義］－［バイナリフォーマット定義ファイル(COBOL 登録集原文からの変換)］を選択します。
3. ［次へ］ボタンをクリックします。
   ［COBOL 登録集原文設定ページ］に進みます。

(2)　変換するファイルの指定

変換する COBOL 登録集原文ファイルの指定方法について説明します。

1. ［COBOL 登録集原文設定ページ］で［参照］ボタンをクリックし，変換する COBOL 登録集原文ファイルを指定します。
2. ［PICTURE 句の通貨編集文字］に，使用する PICTURE 句の通貨編集文字を指定します。
   PICTURE 句の通貨編集文字に指定できる文字については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.10　［新規バイナリフォーマット定義ファイル］ウィザードのページ」を参照してください。
3. ［基本項目の文字コード］で，COBOL 登録集原文ファイルの英数字項目などで使用している文字コードを選択します。
4. ［2 進形式のエンディアン］で，ビッグエンディアンまたはリトルエンディアンのどちらかを選択します。

5. [次へ] ボタンをクリックします。

COBOL 登録集原文ファイルが変換され，[データ項目設定ページ] に進みます。変換中にエラーが発生した場合は，エラーメッセージが表示されます。

(3) データ項目の設定

データ項目の設定方法について説明します。

1. 変換後のバイナリフォーマット定義ファイルの要素名を変更する場合は，[データ項目設定ページ] で [別名] のセルを選択し，任意の名称を入力します。

指定した名称に誤りがある場合は，ページ上部にメッセージが表示されます。[別名] に指定できる文字については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.10 [新規バイナリフォーマット定義ファイル] ウィザードのページ」を参照してください。

2. [次へ] ボタンをクリックします。

[出力ファイル設定ページ] に進みます。

[戻る] ボタンをクリックすると，設定した内容を破棄するかを確認するメッセージが表示されます。[COBOL 登録集原文設定ページ] に戻って COBOL 登録集原文ファイルを指定し直す場合は，[OK] ボタンをクリックします。

(4) 出力ファイルの設定

出力ファイルの設定方法について説明します。

1. [出力ファイル設定ページ] のツリービューから出力先のフォルダを指定します。

[出力ファイル設定ページ] の詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.3.10 [新規バイナリフォーマット定義ファイル] ウィザードのページ」を参照してください。

2. [フォーマット名] に，バイナリフォーマット定義ファイルのフォーマット名を入力します。

3. [終了] ボタンをクリックします。

変換後のバイナリフォーマット定義ファイルが出力され，バイナリフォーマット定義エディタが起動されます。ファイル出力中は，画面下部にプログレスバーが表示されます。

参考

変換後のバイナリフォーマット定義ファイルは，バイナリフォーマット定義エディタで直接作成されたファイルと同様に，バイナリフォーマット定義エディタで編集することができます。バイナリフォーマット定義エディタの詳細は，「4.4.1 バイナリフォーマット定義ファイルのデータ型と文字コードの種類」を参照してください。

4.6　バイナリフォーマット定義ファイルから XML スキーマファイルの生成

cscfdx2xsd コマンドを使用すると，バイナリフォーマット定義ファイルから XML スキーマファイルを生成できます。バイナリデータを同じ構造の XML データへ変換定義する場合に，このコマンドを使用すると変換先の XML スキーマファイルを容易に作成できます。

cscfdx2xsd コマンドの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「cscfdx2xsd (バイナリフォーマット定義ファイルから XML フォーマット定義ファイルの生成)」を参照してください。

次に cscfdx2xsd コマンドを活用する例として，バイナリデータを読み込むファイルアダプタの定義の流れを示します。

1. バイナリフォーマット定義ファイルの作成

   ウィザードを利用してバイナリフォーマット定義ファイルを作成します。

   バイナリフォーマット定義ファイルの作成方法については，「4.4　電文フォーマット（バイナリフォーマット定義ファイル）の作成方法」を参照してください。

2. XML フォーマット定義ファイルの作成

   作成したバイナリフォーマット定義ファイルを基に，cscfdx2xsd コマンドを実行し，XML スキーマファイルを生成します。コマンドでバイナリデータを同じ構造の XML データへ変換すると，XML スキーマ定義をエディタで作成する必要がなくなります。

3. ビジネスプロセス・サービスアダプタの定義

   ウィザードを利用して，作成したバイナリフォーマット定義の設定，コマンドで生成した XML スキーマ定義の設定，およびデータ変換の定義を作成します。

   ビジネスプロセスの定義については，「5.　ビジネスプロセスの定義」を参照してください。

   サービスアダプタの定義については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.　サービスアダプタの定義」を参照してください。データ変換定義については，「7.　データ変換の定義」を参照してください。

4.7　電文フォーマットの変更

電文フォーマットを変更した場合，データ変換を定義し直す必要があります。電文フォーマットを変更したあとに，データ変換を定義し直さないと，検証（サービスアダプタ，ビジネスプロセス）またはパッケージングでエラーが発生します。

データ変換を定義し直すときは，以前に作成したデータ変換定義を利用できます。新規に定義する必要はありません。電文フォーマットを変更した場合の定義手順および注意事項については，「7.3.2　電文フォーマットを変更した場合の定義手順」を参照してください。

5 ビジネスプロセスの定義

この章では，ビジネスプロセスの定義について説明します。

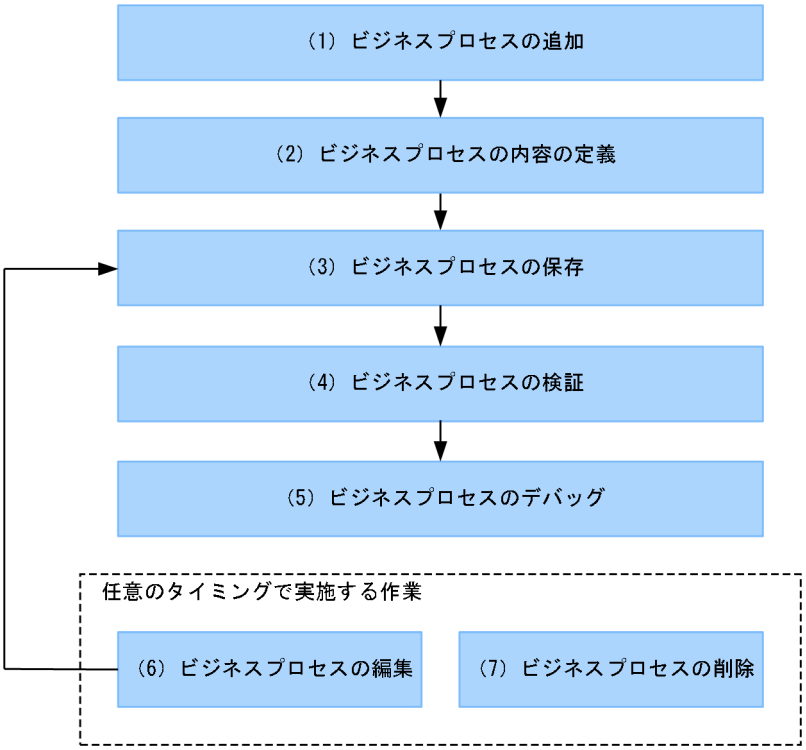
5.1　ビジネスプロセスの定義の手順

ここでは，ビジネスプロセスの定義を設定する手順について説明します。

5.1.1　定義の流れ

ビジネスプロセスの定義の流れを次の図に示します。

図 5-1　ビジネスプロセスの定義の流れ



ビジネスプロセスの定義に関する作業について次に説明します。

(1)　ビジネスプロセスの追加

ビジネスプロセスを追加するには，次のどちらかの方法を利用します。

新規に追加する場合

ウィザードを利用してビジネスプロセスを新規に追加します。

ウィザードを利用してビジネスプロセスを新規に追加するには，未定義のビジネスプロセスを追加する方法と，あらかじめ BPMN を用いた上流設計ツールで作成した BPEL ファイルをインポートして，ビジネスプロセスの定義に変換して追加する方法があります。

ビジネスプロセスを新規に追加する方法の詳細については，「5.2.1　ビジネスプロセスを新規に追加する」を参照してください。

定義済みのビジネスプロセスを利用する場合

定義済みのビジネスプロセスを複製してビジネスプロセスを追加できます。定義済みのビジネスプロセスを複製すると，同じ定義内容のビジネスプロセスが追加されます。複製したビジネスプロセスの定義内容を編集することもできます。

定義済みのビジネスプロセスを利用してビジネスプロセスを追加する方法の詳細は，「5.2.2　定義済みビジネスプロセスを利用してビジネスプロセスを追加する」を参照してください。

(2)　ビジネスプロセスの内容の定義

ビジネスプロセス定義画面でアクティビティ，および変数・相関セットを定義します。

定義の方法および定義の内容の詳細については，「5.3　ビジネスプロセスの内容の定義」を参照してください。ビジネスプロセス定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.3　ビジネスプロセス定義画面」を参照してください。

(3)　ビジネスプロセスの保存

編集したビジネスプロセスの定義情報は，必要に応じてリポジトリに保存する必要があります。

ビジネスプロセスの定義情報の保存方法と保存されるタイミングについては，「5.8　ビジネスプロセスの保存」を参照してください。

(4)　ビジネスプロセスの検証

定義したビジネスプロセスの整合性を検証します。ビジネスプロセスの定義中や定義が終わったあとなど，任意のタイミングで検証できます。

検証方法の詳細については，「5.11　ビジネスプロセスの検証」を参照してください。

(5)　ビジネスプロセスのデバッグ

定義したビジネスプロセスを開発環境でデバッグします。ビジネスプロセスの定義が終わったあと，任意のタイミングでデバッグできます。

ビジネスプロセスをデバッグする方法については，「10.　ビジネスプロセスのデバッグ」を参照してください。

(6)　ビジネスプロセスの編集

定義済みのビジネスプロセスは，必要に応じて定義内容を編集して変更できます。

ビジネスプロセスの編集方法の詳細については，「5.9　ビジネスプロセスの編集」を参照してください。

(7)　ビジネスプロセスの削除

不要になったビジネスプロセスは，必要に応じて削除できます。

ビジネスプロセスの削除方法については，「5.12　ビジネスプロセスの削除」を参照してください。

5.2　ビジネスプロセスの追加

ビジネスプロセスは次の方法で追加できます。

- ビジネスプロセス追加ウィザードを利用する。
  ビジネスプロセスを新規に追加する場合，ビジネスプロセス追加ウィザードを利用します。また，すでに作成されているビジネスプロセスを複製して，新しいビジネスプロセスを追加することもできます。
- XML ファイルを使用してビジネスプロセスをインポートする。
  ビジネスプロセスなどの HCSC コンポーネントの内容を定義した XML ファイルを，HCSCTE プロジェクトにインポートして追加します。この方法では，一度に複数のビジネスプロセスを追加できます。詳細については「6.　サービス定義のインポート・エクスポート」を参照してください。

ここでは，ビジネスプロセス追加ウィザードを利用する方法について説明します。また，ビジネスプロセス追加後に設定できる項目についても併せて説明します。

5.2.1　ビジネスプロセスを新規に追加する

ビジネスプロセス追加ウィザードを利用してビジネスプロセスを新規に追加する方法には，次の 2 つがあります。

- 未定義のビジネスプロセスの新規追加
  未定義のビジネスプロセスを新規に追加する方法です。この方法で新規に追加されるビジネスプロセスは，何も定義されていない状態になります。
- BPEL ファイルをインポートして新規追加
  あらかじめ BPMN を用いた上流設計ツールで BPEL ファイルを作成します。そのあと，BPEL ファイルをインポートしてビジネスプロセスの定義に変換して新規に追加する方法です。この方法で新規に追加されるビジネスプロセスには，基となった BPEL ファイルの定義情報が引き継がれます。

それぞれの方法について，次に説明します。

(1)　未定義のビジネスプロセスを新規に追加する

何も定義されていないビジネスプロセスを新規に追加する方法を次に示します。

1. ツリービューのサービス定義一覧で，右クリックして［ビジネスプロセス追加］を選択します。
   ビジネスプロセス定義を追加するためのダイアログが表示されます。
2. ビジネスプロセス名を入力し，ステータスを永続化するかどうか（ビジネスプロセスの実行状態をデータベースに保存するかどうか）を選択します。
   ビジネスプロセス名（ビジネスプロセスのサービス名）は，64 バイト以内で入力します。
3. ［終了］ボタンをクリックします。

必要なファイルが作成され，リポジトリに保存されます。

ビジネスプロセスが正常に保存されると，ビジネスプロセス定義画面が表示されます。

(2)　BPEL ファイルをインポートしてビジネスプロセスを新規に追加する

あらかじめ上流設計ツールで BPEL ファイルを作成し，BPEL ファイルをインポートしてビジネスプロセスを新規に追加する方法について説明します。

(a)　BPEL ファイルの作成

上流設計ツールで BPEL1.1 または BPEL2.0 に準拠した BPEL ファイルを作成します。BPEL ファイルの作成方法については，利用する上流設計ツールのドキュメントを参照してください。

BPEL ファイルに定義した要素および属性の内容と，ビジネスプロセス定義の内容との関連については，「付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲」を参照してください。

(b)　ビジネスプロセスの新規追加

BPEL ファイルをインポートして，ビジネスプロセスを新規に追加する方法を次に示します。

1. ツリービューのサービス定義一覧で，右クリックして［ビジネスプロセス追加］を選択します。

   ビジネスプロセス定義を追加するためのダイアログが表示されます。

2. ビジネスプロセス名を入力し，ステータスを永続化するかどうか（ビジネスプロセスの実行状態をデータベースに保存するかどうか）を選択します。

   ビジネスプロセス名（ビジネスプロセスのサービス名）は，64 バイト以内で入力します。

3. ［BPEL ファイル］の［インポートする］のチェックボックスをチェックします。

4. ［ファイル名］にインポートする BPEL ファイルを指定します。

   ［...］ボタンをクリックして，インポートする BPEL ファイルを選択することもできます。

5. アクティビティの配置方法を選択します。

6. ［終了］ボタンをクリックします。

   BPEL ファイルの定義が，ビジネスプロセスの定義に変換されます。変換が終わり，必要なファイルが作成されると，リポジトリに保存されます。

   ビジネスプロセスが正常に保存されると，ビジネスプロセス定義画面が表示されます。表示されるビジネスプロセス定義画面のキャンバスには，手順 5.で選択した方法でアクティビティが配置されます。

注意事項

BPEL ファイルをインポートする場合，次の点に注意が必要です。

変換されない BPEL の定義

サービスプラットフォームの開発環境でサポートしていない BPEL の定義は，無視されるか，無操作アクティビティに変換されます。

BPEL ファイルに定義した要素および属性の内容と，ビジネスプロセス定義の内容との関連については，「付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲」を参照してください。

BPEL ファイルでメッセージ型の変数が定義されていた場合

上流設計ツールでメッセージの名称を定義し，BPEL ファイルにメッセージ型の変数が定義されていた場合，BPEL ファイルをインポートして作成したビジネスプロセスでは，変数の種別が文字列型（string）に変換されます。

この場合，インポート後に変数の種別をメッセージ型（XML）に変更してください。変数の定義については，「5.5.1(6)　変数の定義方法」を参照してください。

サービス呼出アクティビティが定義されていた場合

BPEL ファイルをインポートしてビジネスプロセスを作成するとき，BPEL ファイルの invoke 要素はサービス呼出アクティビティに変換されます。invoke 要素の portType 属性のローカル名が呼び出すサービス名に，operation 属性がオペレーション名に変換されます。

ただし，リポジトリに該当するサービスやオペレーションがない場合には，サービス名やオペレーション名は変換されません。

サービス名やオペレーション名が変換されなかった場合，BPEL ファイルをインポートしたあと，呼び出す対象のサービスアダプタやビジネスプロセスを作成して，サービス名およびオペレーション名を割り当てる必要があります。

BPEL ファイルに長い名称のアクティビティが多数定義されていた場合

配置方法で横方向が指定されていて，BPEL ファイルに長い名称のアクティビティが多数定義されている場合，インポート後にキャンバスの端でアクティビティが重なってしまうことがあります。その場合は，アクティビティを配置し直してください。

(3)　ステータスの永続化の設定

ビジネスプロセスには，データベースにプロセスの実行状態および実行履歴を永続化するビジネスプロセスと，永続化しないビジネスプロセスの 2 種類があります。永続化とは，プロセスの実行状態および実行履歴をデータベースに記録することです。

永続化するビジネスプロセスは，プロセスの実行状態と実行履歴の記録があるため，障害が発生した場合に，プロセスの実行の進捗度がわかり，プロセスを再実行できます。永続化しないビジネスプロセスは，プロセスの実行状態と実行履歴の記録を採らないため，高性能を実現する必要があるビジネスプロセスの場合に設定します。

永続化するビジネスプロセスではできることが，永続化しないビジネスプロセスではできないといった制約事項があります。永続化しないビジネスプロセスは，次の条件を満たす場合に定義できます。

1. 非同期のサービスアダプタに対して，サービスを呼び出すサービス呼出アクティビティを含まない。
2. 待機アクティビティを含まない。
3. 受付アクティビティを複数含まない。
4. プロセスインスタンスの再実行を使用しない（障害発生時は，サービスリクエスタからサービス呼び出し要求（ビジネスプロセスの呼び出し）を再度実行する）。

5.2.2　定義済みビジネスプロセスを利用してビジネスプロセスを追加する

すでに定義されているビジネスプロセスを複製して，ビジネスプロセスを追加できます。

1. **ツリービューのサービス定義一覧で，複製するビジネスプロセスを選択して右クリックします。**
   サービス一覧のポップアップメニューが表示されます。

2. **ポップアップメニューから［複製］を選択します。**
   選択したビジネスプロセスの複製が作成されます。複製されたビジネスプロセス名（ビジネスプロセスのサービス名）やサービス ID※などは，システム内で競合しないよう，自動的に別の情報が割り当てられます。

注※
サービスアダプタおよびビジネスプロセスを追加するときに，自動的に割り当てられるシステム内で一意の番号です。サービス ID は変更できますが，すでに使用されているサービス ID には変更できません。

注意事項

ビジネスプロセスを複製すると，複製したビジネスプロセスに含まれるユーザ定義受付（呼出先固定）もあわせて複製されます。ただし，ユーザ定義受付（呼出先固定）だけを複製することはできません。なお，ユーザ定義受付（呼出先選択）だけを複製することはできます。

5.2.3　ビジネスプロセス追加後に設定する項目

ここでは，ビジネスプロセスを追加したあとに設定する項目について説明します。

(1)　ステータスの永続化の設定

ステータスの永続化は，ビジネスプロセス追加ウィザードで設定できますが，ビジネスプロセス追加後にも設定できます。手順の詳細については，「5.9.1(1)　ビジネスプロセスの定義情報を変更する」を参照してください。

(2)　スコープアクティビティのデフォルトハンドラの設定

ビジネスプロセスに補償処理を定義する場合，スコープアクティビティのデフォルトハンドラの設定を ON にすると，補償処理が暗黙的に定義されるようになります。補償処理の定義，およびスコープアクティビティのデフォルトハンドラの詳細については，「5.4.5　補償処理を定義する」を参照してください。

5.3　ビジネスプロセスの内容の定義

ビジネスプロセスの内容はビジネスプロセス定義画面で定義します。

ビジネスプロセス定義画面は，ツリービューのサービス定義一覧で表示されたサービス内のビジネスプロセスをダブルクリックすると表示されます。

ビジネスプロセス定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.3　ビジネスプロセス定義画面」を参照してください。

ビジネスプロセス定義画面では，アクティビティと変数・相関セットを定義します。

5.3.1　アクティビティ

ビジネスプロセスは，複数のアクティビティを連結して定義します。

アクティビティとは，ビジネスプロセスの構成要素となる部品であり，処理の構造も表現したものです。複数のアクティビティを連結して，ビジネスプロセスの処理フローを定義します。ビジネスプロセスは，アクティビティを画面上で配置・連結して定義します。

アクティビティの配置，連結方法の詳細については，「5.4　アクティビティの配置と連結」を参照してください。

注意事項

1 つのビジネスプロセスを実行する場合に，そのプロセスインスタンス内に生成できるアクティビティインスタンスの数は最大で 2,147,483,648 個です。ただし，次の場合は生成できる数が最大で 32,768 個になります。

- activitynumber-maximum-compatible プロパティに「ON」を設定した場合
- 旧バージョンからの移行で，データベースを使用している環境を上書きインストールした場合

最大数を変更するには，HCSC サーバの再セットアップが必要です。再セットアップすると，ビジネスプロセスの実行履歴が削除されるので注意してください。

最大数より多いアクティビティインスタンスを生成しようとしたときは，ビジネスプロセスの実行中にエラーになります。

なお，アクティビティインスタンスには，開始アクティビティ，終了アクティビティ，順次処理アクティビティなどのすべてのアクティビティが含まれます。

また，繰り返しアクティビティで処理を繰り返したときは，実行するアクティビティの数だけアクティビティインスタンスが生成されます。

5.3.2　変数・相関セット

ビジネスプロセスには，変数と相関セットを定義できます。

(1)　変数

ビジネスプロセスの処理の中で，条件式の項の値などに変数値を利用する場合，アクティビティの定義で変数を宣言します。

ビジネスプロセス内で利用できる変数の種類，アクティビティとの関係および定義方法については，「5.5.1 変数の定義」を参照してください。

(2)　相関セット

相関セットとは，サービスリクエスタから HCSC サーバを介してサービス部品に送信する要求電文を一意に識別するために利用する文字列です。

相関セットとアクティビティとの関係や定義方法については，「5.5.2　相関セットの定義」を参照してください。

注意事項

ビジネスプロセスの定義では，定義中に入力値が正しいかどうかのチェックはされません。入力値が正しいかどうかは，ビジネスプロセスの検証でチェックされます。入力値が正しくない場合，検証でエラーになります。

5.4　アクティビティの配置と連結

ビジネスプロセス定義画面では，アクティビティをキャンバスに配置し，連結してビジネスプロセスの処理フローを定義します。また，フォルト処理を設定する場合には，アクティビティ間の連結にフォルト処理を定義します。補償処理を設定する場合には，フォルトハンドラ，補償ハンドラ，補償アクティビティを組み合わせて補償処理を定義します。

以降の項では，アクティビティの配置方法，アクティビティの連結方法，フォルト処理の定義方法および補償処理の定義方法について説明します。

5.4.1　アクティビティを配置する

ビジネスプロセス定義画面でのアクティビティの配置方法を次に示します。

1. **パレットの基本アクティビティまたは構造アクティビティの中から，キャンバスに配置するアクティビティをクリックします。**

   クリックしたアクティビティが選択された状態になります。

2. **キャンバス上の適当な場所をクリックします。**

   選択したアクティビティがキャンバス上に配置されます。配置されたアクティビティは，ドラッグ&ドロップで場所を移動できます。

> **参考**
>
> 次の手順でもアクティビティを配置できます。
>
> 1. パレットの基本アクティビティまたは構造アクティビティの中から，キャンバスに配置するアクティビティをドラッグします。
> 2. キャンバス上の適当な場所にドロップします。

なお，各アクティビティで定義できる内容の詳細については，「5.6　アクティビティの定義」を参照してください。

> **注意事項**
>
> デフォルトでは，オートコネクト機能が有効になっています。キャンバスに配置済みのアクティビティの近くに，別のアクティビティを配置した場合，自動的にコネクションが連結されることがあります。コネクションの連結をしないでアクティビティを配置したいときは，配置済みのアクティビティから離れた位置に配置してください。オートコネクト機能については，「5.4.3　アクティビティのオートコネクト」を参照してください。

ポイント

上流設計ツールで作成した BPEL ファイルをインポートしてビジネスプロセスを追加した場合，ビジネスプロセスを追加するときに選択した配置方法によって，キャンバスに初期表示されるアクティビティの配置が異なります。

「縦方向」を選択した場合

上から下に処理が流れる形でアクティビティが初期表示されます。

「横方向」を選択した場合

左から右に処理が流れる形でアクティビティが初期表示されます。

必要に応じてドラッグ&ドロップしたり，アクティビティを整列したりしてください。ビジネスプロセス定義画面（キャンバス）での操作については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.3　ビジネスプロセス定義画面」を参照してください。

5.4.2　アクティビティを連結する

キャンバスに配置された 2 つのアクティビティを連結すると，ビジネスプロセスの実行順序が定義されます。

(1)　連結の種類

アクティビティの連結には，次の種類があります。

- コネクションでの連結

  通常のアクティビティの連結に利用します。
- フォルトハンドラとフォルトコネクションでの連結

  アクティビティでフォルトが発生したときのためのフォルト処理を定義する場合に利用します。フォルトハンドラでは，アクティビティでフォルトがスローされた場合に，フォルトをどのように処理するかを定義します。フォルトコネクションでは，フォルトハンドラとフォルト処理が実行される一連のアクティビティ群を対応づけます。

  フォルト処理の定義の詳細については，「5.4.4　フォルト処理を定義する」を参照してください。
- リンクコネクションでの連結

  並列処理アクティビティを利用した並列処理で，リンクを定義する場合に利用します。並列処理とリンクについては，「5.6.18　並列処理アクティビティ」を参照してください。
- 補償ハンドラと補償コネクションでの連結

  アクティビティでフォルトが発生したときに，完了済みのアクティビティを補償するための補償処理を定義する場合に利用します。補償ハンドラでは，補償処理として実行する一連のアクティビティ群を定義します。補償コネクションでは，補償ハンドラと補償処理が実行される一連のアクティビティ群を対応づけます。

補償処理の定義の詳細については，「5.4.5　補償処理を定義する」を参照してください。

連結方法によって，キャンバス上の連結線の表示は異なります。連結方法と連結線の関係を次の表に示します。

表 5-1　連結の画面表示（コネクション）

コネクション	表示される連結線	線の色	線の種類	終端
コネクション		黒	実線	三角の矢印
フォルトコネクション		赤	破線	三角の矢印
リンクコネクション		青	点線	線の矢印
補償コネクション		緑	破線	三角の矢印

表 5-2　連結の画面表示（ハンドラ）

ハンドラの種類	アイコン
フォルトハンドラ	
補償ハンドラ	

(2)　連結の設定方法

アクティビティを連結する方法を次に示します。

1. パレットの［コネクション］，［リンク］，［フォルト］，または［補償］をクリックします。

   コネクション，フォルトコネクション，リンクコネクション，または補償コネクションが選択された状態になります。

   この状態で，アクティビティにマウスポインタを合わせると，そのアクティビティを連結元に設定できる場合，アクティビティの背景色が変わります。

2. キャンバスに配置したアクティビティのうち，連結元となるアクティビティをクリックします。

   連結元となるアクティビティが設定されます。

   この状態で，ほかのアクティビティにマウスポインタを合わせると，そのアクティビティを連結先に設定できる場合，アクティビティの背景色が変わります。

3. キャンバスに配置したアクティビティのうち，連結先となるアクティビティをクリックします。

   連結元となるアクティビティと連結先となるアクティビティが連結されます。

(3)　連結の変更方法

すでに設定されている連結を変更するには，連結線の始点または終点をドラッグし，ほかのアクティビティにドロップします。

(4) 連結線の折り曲げ

連結線は，折り曲げて表示できます。アクティビティの連結が複雑になり，キャンバス上の連結線を整理する場合などに利用できます。連結線を折り曲げる方法について次に示します。

1. キャンバス上で連結線を選択します。
   連結線の始点と終点の中間に折り曲げられる個所（ベンドポイント）が表示されます。

2. ベンドポイントに選択ツールのポインタを合わせます。
   十字の矢印（ ✣ ）が表示されます。

3. 矢印を任意の位置にドラッグ＆ドロップします。
   連結線がベンドポイントから折れ曲がります。
   ベンドポイントで折り曲げた連結線では，次の個所もベンドポイントとして利用できます。
   - 連結線の始点とベンドポイントの中間
   - ベンドポイントと連結線の終点の中間
   - ベンドポイントとベンドポイントの中間

(5) アクティビティの連結条件

アクティビティと利用する連結の種類によって，アクティビティを連結元または連結先として利用できるかどうかが異なります。各アクティビティの連結元または連結先としての利用可否について，次の表に示します。

表 5-3 連結元または連結先として利用できるアクティビティ

アクティビティ	コネクション		フォルトコネクション		リンクコネクション		補償コネクション	
	連結元	連結先	連結元	連結先	連結元	連結先	連結元	連結先
開始	○	－	－	－	－	－	－	－
受付	○	○	－	○	○	○	－	○
応答	○	○	－	○	○	○	－	○
サービス呼出	○	○	○	○	○	○	○	○
Java 呼出	○	○	○	○	○	○	－	○
データ変換	○	○	－	○	○	○	－	○
代入	○	○	－	○	○	○	－	○
無操作	○	○	－	○	○	○	－	○
フォルト送出	－	○	－	○	○	○	－	○
補償	○	○	－	○	○	○	－	○

アクティビティ	コネクション		フォルトコネクション		リンクコネクション		補償コネクション	
	連結元	連結先	連結元	連結先	連結元	連結先	連結元	連結先
待機	○	○	−	○	○	○	−	○
検証	○	○	−	○	○	○	−	○
ログ出力	○	○	−	○	○	○	−	○
途中終了	−	○	−	○	○	○	−	○
スコープ	○	○	○	○	○	○	○	○
繰り返し	○	○	−	○	○	○	−	○
分岐（開始）	○	○	−	○	○	○	−	○
分岐（終了）	○	○	−	−	−	−	−	−
並列処理（開始）	○	○	−	○	○	○	−	○
並列処理（終了）	○	○	−	−	−	−	−	−
終了	−	○	−	−	−	−	−	−

（凡例）

○：連結元または連結先としてアクティビティを利用できます。

−：連結元または連結先としてアクティビティを利用できません。

また，アクティビティをコネクションで連結するとき，あるアクティビティへの遷移元となるアクティビティを遷移元アクティビティ，あるアクティビティからの遷移先となるアクティビティを遷移先アクティビティといいます。

コネクション（フォルトコネクションおよびリンクコネクションは除く）で連結できる遷移元アクティビティおよび遷移先アクティビティの数は，アクティビティごとに異なります。

アクティビティごとの連結できる遷移元・遷移先アクティビティの数を次の表に示します。

表 5-4　連結できる遷移元・遷移先アクティビティの数

アクティビティ	連結できる遷移元アクティビティの数	連結できる遷移先アクティビティの数
開始	0	1
受付	1	1
応答	1	1
サービス呼出	1	1
Java 呼出	1	1
データ変換	1	1
代入	1	1
無操作	1	1

アクティビティ	連結できる遷移元アクティビティの数	連結できる遷移先アクティビティの数
フォルト送出	1	0
補償	1	1
待機	1	1
検証	1	1
ログ出力	1	1
途中終了	1	0
スコープ	1	1
繰り返し	1	1
分岐（開始）	1	1 以上
分岐（終了）	1 以上	1
並列処理（開始）	1	1 以上
並列処理（終了）	1 以上	1
終了	1	0

注意事項

ビジネスプロセスで最初に実行できる基本アクティビティは，インスタンスを生成する受付アクティビティだけです。

(6)　コネクション使用時の注意事項

- リンクコネクションでリンク先の遷移条件が「あり」に設定した場合，遷移条件を設定してください。
- リンクコネクションは，循環しないように設定してください。
- リンク名はビジネスプロセス内で一意になるよう設定してください。
- リンクコネクションは，フォルト処理外からフォルト処理内へ張らないでください。
- コネクションで連結したアクティビティ同士を近づけると，連結線の矢印の向きが設定と逆に見える場合があります。連結したアクティビティ同士は，ある程度の距離をとって配置してください。
- リンクコネクションは，補償処理内と補償処理外をまたいで張らないでください。

5.4.3　アクティビティのオートコネクト

オートコネクトでは，次の 2 つの処理を同時に実行できます。

- アクティビティの配置

- アクティビティのコネクションでの連結

また，次の 2 つの処理を同時に実行できます。

- 配置済みアクティビティの移動
- アクティビティのコネクションでの連結

(1) アクティビティの配置と連結

パレットからアクティビティを配置すると，同時にコネクションで連結できる配置済みアクティビティが検索されます。連結する候補となったアクティビティのうち，次の条件に沿ったアクティビティと連結されます。

- 配置位置の最も近くにあるアクティビティ
- 連結できるエリア内にあるアクティビティ

連結できるエリアはマウスのドラッグ中に円状で表示されます。

(2) アクティビティの移動と連結

まだコネクションで連結されていない，キャンバスに配置済みのアクティビティを移動すると，同時にコネクションで連結できる別の配置済みアクティビティが検索されます。連結する候補となったアクティビティのうち，次の条件に沿ったアクティビティと連結されます。

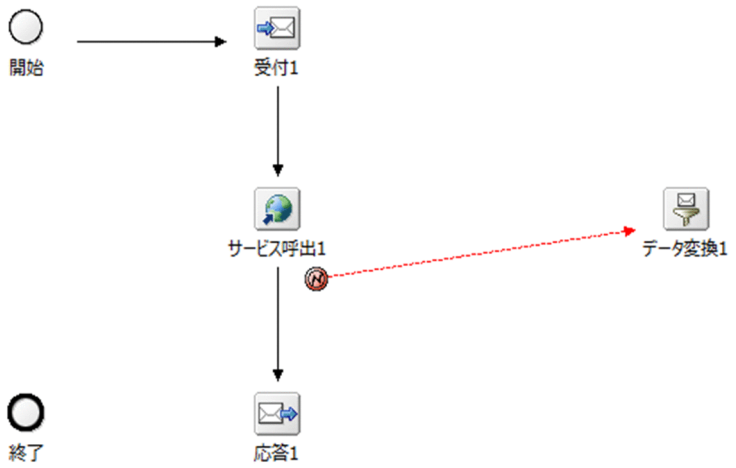
- 配置位置の最も近くにあるアクティビティ
- 連結できるエリア内にあるアクティビティ

連結できるエリアはマウスのドラッグ中に円状で表示されます。

(3) コネクションで連結できる配置済みのアクティビティ

キャンバスに配置済みのアクティビティのうち，コネクションで連結されていないアクティビティがオートコネクトの対象となります。

次に示す例の場合，［応答 1］，［終了］と［データ変換 1］が対象の候補となります。



また，キャンバスに配置済みのアクティビティのうち，コネクションで連結されている 2 つのアクティビティの組はオートコネクトの対象となります。上記の例では，［開始］と［受付 1］との組，［受付 1］と［サービス呼出 1］との組，［サービス呼出 1］と［応答 1］との組が対象の候補となります。この場合，配置，または移動するアクティビティは，2 つのアクティビティの間に挿入されます。

(4)　オートコネクトの設定

オートコネクトの有効化，または無効化の設定については，マニュアル「サービスプラットフォーム リファレンス」の「1.12.16　アクティビティのオートコネクト機能の有効・無効を設定する」を参照してください。

5.4.4　フォルト処理を定義する

アクティビティでフォルトが発生したときに何らかの処理を実行する場合，フォルト処理用のアクティビティを配置してフォルト処理を実行します。

フォルト処理は，フォルトが発生するアクティビティと，フォルト処理を行うアクティビティを連結して定義します。連結にはフォルトコネクションを利用します。

(1)　フォルト処理の定義手順

フォルト処理の定義手順を次に示します。

1. **フォルトが発生するアクティビティとフォルト処理を行うアクティビティをキャンバスに配置し，詳細を定義します。**

   フォルト処理を行うアクティビティは複数定義できます。

   アクティビティの配置方法については，「5.4.1　アクティビティを配置する」を参照してください。各アクティビティの定義方法の詳細については，「5.6　アクティビティの定義」を参照してください。

2. フォルトコネクションを利用してフォルトが発生するアクティビティとフォルト処理を行うアクティビティを連結します。

アクティビティの連結方法については，「5.4.2(2)　連結の設定方法」を参照してください。また，フォルトコネクションを利用できるアクティビティについては，「5.4.2(5)　アクティビティの連結条件」を参照してください。

3. 連結線をダブルクリックします。

［フォルト処理の割当］ダイアログが表示されます。

［フォルト処理の割当］ダイアログの入力・表示内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.6　フォルト処理の割当ダイアログ」を参照してください。なお，フォルト処理に割り当てられる変数はメッセージ型（XML）だけです。メッセージ型（non-XML および any）は割り当てられません。

4. フォルト処理の条件を定義します。

フォルトの対象となる変数と，フォルトが発生した場合に処理する連結先のアクティビティを，それぞれ［割当変数］および［遷移先］ドロップダウンリストで指定します。

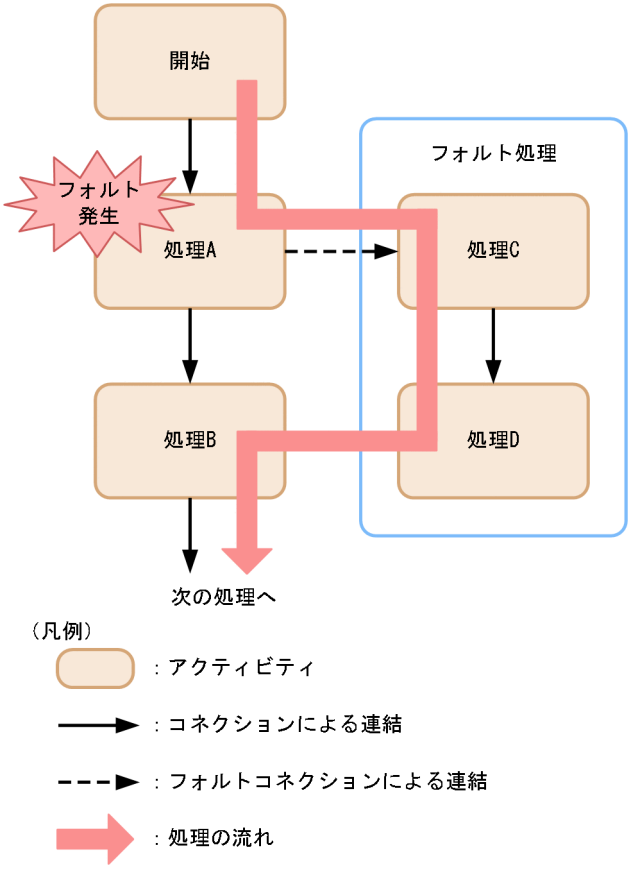
定義していないフォルトが発生した場合のフォルト処理を設定する場合，［割当変数］ドロップダウンリストで「catch-all」を選択します。

サービスアダプタからサービス部品を呼び出したときに発生したエラーの例外を，サービスアダプタでフォルト電文に変換する場合，変数の定義で設定した定義ファイルを［割当変数］ドロップダウンリストで選択します。定義ファイルの設定方法については，「5.5.1(6)(a)　新規に変数を定義する場合」を参照してください。

5. ［OK］ボタンをクリックします。

次の図に示すように，フォルトが発生した場合，フォルト処理を実行したあとに，フォルトが発生したアクティビティの次のアクティビティに処理が移ります。

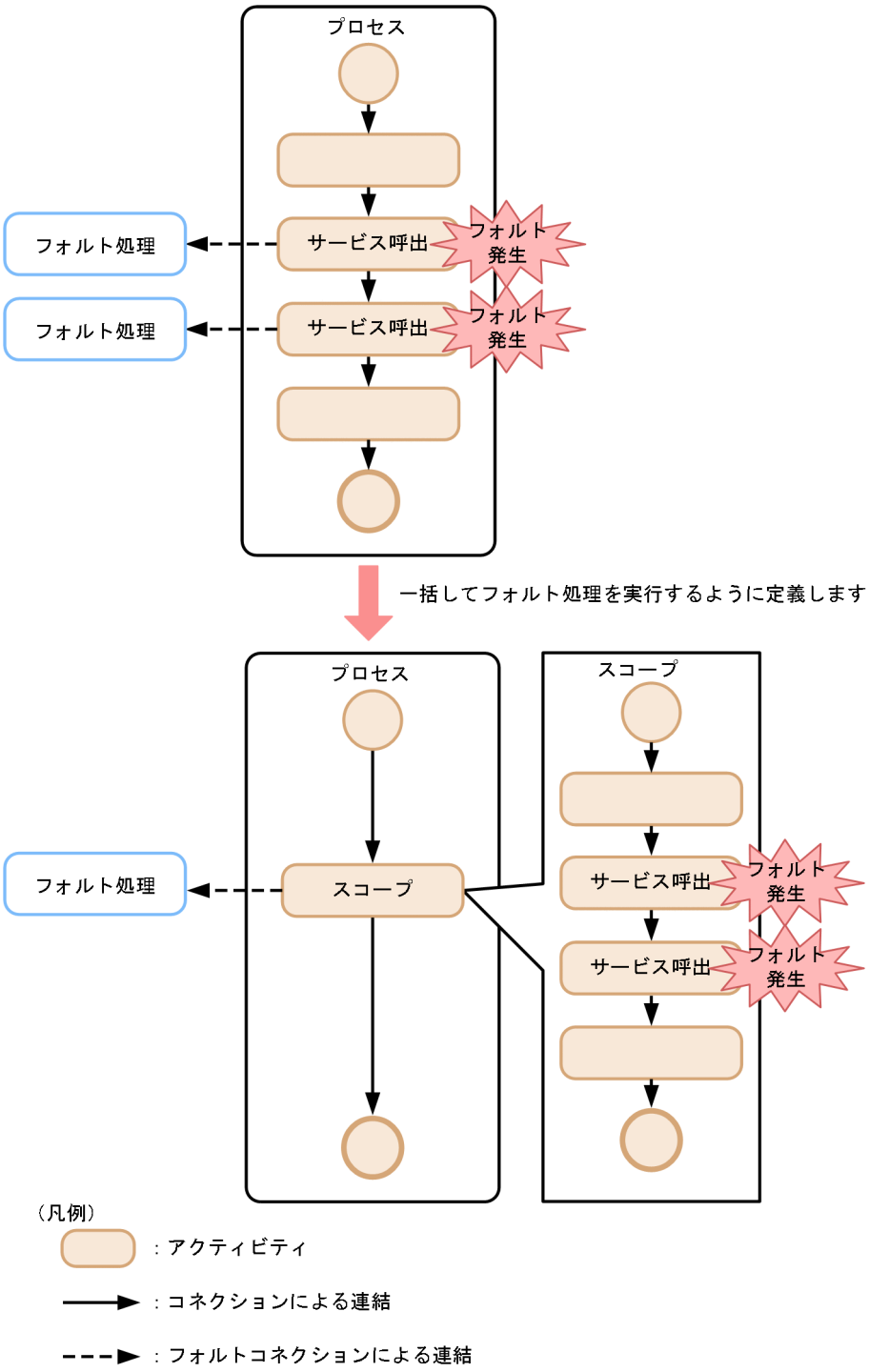
図 5-2　フォルト処理を実行したあとの処理の流れ



(2)　一括してプロセスのフォルト処理を実行する場合の定義

プロセス内にフォルトが返るおそれのあるアクティビティが複数あった場合，アクティビティごとにフォルト処理を定義するのではなく，一括してプロセスのフォルト処理を実行できます。その場合，次の図に示すようにスコープアクティビティを使って，プロセス内の処理を 1 つのまとまりにして，フォルト処理を定義します。

図 5-3　スコープアクティビティを使って定義したフォルト処理



(3)　ビジネスプロセスのフォルト処理の定義で使用する電文フォーマット

ビジネスプロセスのフォルト処理の定義で使用する電文フォーマットについて説明します。

(a)　複数のフォルト名に対応する電文フォーマット

複数のユーザ定義例外を throw する Java プログラムから，SOAP アプリケーション開発支援機能（Java2WSDL コマンド）を使用して Document スタイルの WSDL を生成し，これを入力としてサービ

スアダプタを定義すると，サービスアダプタ定義画面の［フォルト電文］に表示される複数のフォルト名に対応する電文フォーマットは，同一のものとなります。この場合，電文フォーマットは，次の例に示すような複数のルート要素を含む形式になっています。

複数のフォルト名に対応する電文フォーマット（ルートスキーマ）の例

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://service"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://service"
  xmlns:tns2="http://data.service"
  xmlns:tns3="http://fault.service">
  <xsd:import namespace="http://data.service"
              schemaLocation="cscformat2.xsd"/>
  <xsd:import namespace="http://fault.service"
              schemaLocation="cscformat3.xsd"/>
  <xsd:element name="order">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="in0" type="tns2:OrderData"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="orderResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="orderReturn" type="tns2:OrderResult"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="StockShortageFault"
               type="tns3:StockShortageFault"/>
  <xsd:element name="InvalidCustomerFault"
               type="tns:InvalidCustomerFault"/>
  <xsd:complexType name="InvalidCustomerFault">
    <xsd:sequence>
      <xsd:element name="CustomerName" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

上記の例では，「StockShortageFault」と「InvalidCustomerFault」という 2 つのフォルトに対応したフォルト電文が，「cscformat1.xsd」という 1 つの電文フォーマットで扱われています。

SOAP アプリケーション開発支援機能（Java2WSDL コマンド）を使用して rpc スタイルの WSDL を生成し，フォルトを扱えるように，「4.3.2　サービス部品電文を作成する（Web サービスの場合）」の注意事項の手順で修正した WSDL を基にサービスアダプタを定義した場合も，同様の形式となります。

発生するフォルトに応じて処理を切り分けるために，この電文フォーマットをフォルトごとに別々の変数として定義し，［フォルト処理の割当］ダイアログで 2 つの catch の割当変数として定義した場合，2 つの catch に割り当てられている変数のフォーマットは同一になってしまいます。そのため，どちらのフォルトが発生しても先に定義されている catch に適合してしまい，要因に応じたフォルト処理が呼び出されません。

フォルト電文がこのようなフォーマットになっている場合は，次に示す方法でビジネスプロセスを定義して対処してください。

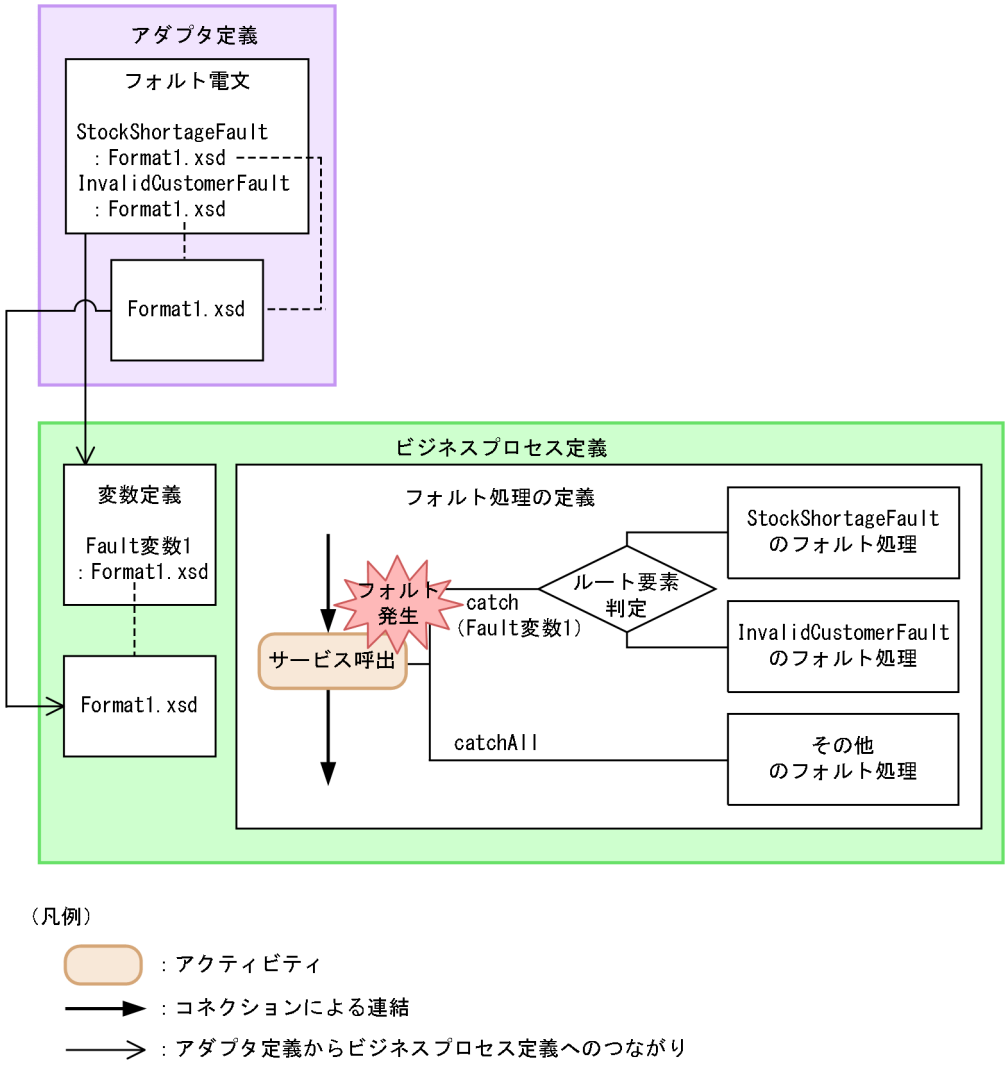
(b)　定義方法

フォルトごとに変数を定義しないで，電文フォーマットが同一のフォルトに対して共通となる変数を 1 つ定義します。フォルト処理の遷移先も 1 つにし，［フォルト処理の割当］ダイアログで，共通となる変数を割り当てます。ただし，想定外のフォルト処理のために，別途 catchAll の遷移先を設定することもできます。

フォルト処理の遷移先には分岐アクティビティを配置し，ルートの要素名などで発生したフォルトを切り分ける分岐条件を記述します。

共通となる変数を割り当てた場合の概念図を次に示します。

図 5-4　共通となる変数を割り当てた場合の概念図



ルート要素名を取得して一致判定（図中の「ルート要素判定」）を実施する分岐条件の記述例を次に示します。

```
csc:getVariableData('Fault変数1','local-name(/*)')="StockShortageFault"
```

「Fault 変数 1」の部分にフォルト共通変数の変数名，「StockShortageFault」の部分にルート要素名を指定します。

5.4.5　補償処理を定義する

補償処理とは，ビジネスプロセス実行中にフォルトが発生した場合に，実行完了していた処理を打ち消す処理のことです。

補償処理の定義には，次の 2 つの方法があります。運用に合わせてどちらかの方法を選択してください。

- 任意のアクティビティに，明示的に補償処理を定義する

  あるアクティビティでフォルトが発生した場合に，任意のアクティビティでだけ補償処理を実行したいときには，補償処理の対象となるアクティビティと，補償処理を実行するアクティビティを連結して，補償処理を定義します。連結には補償コネクションを利用します。
- スコープアクティビティのデフォルトハンドラの設定を ON にし，すべてのスコープアクティビティに対して暗黙的に補償処理を定義する

  スコープアクティビティのデフォルトハンドラとは，補償ハンドラが定義されているビジネスプロセス内のアクティビティでフォルトが発生した場合，それまでに正常終了したアクティビティを，すべて自動的に補償する設定のことです。明示的に補償処理を定義しなくても，この設定の対象となるアクティビティに対し，補償処理が自動的に実行されます。

  ただし，スコープアクティビティのデフォルトハンドラの設定を ON にすると，システムの処理時間が増加します。異常発生（あるアクティビティでフォルトが発生）時に，補償処理の対象となるアクティビティがどれか，ビジネスプロセス全体を探索するためです。異常発生時でのシステムのターンアラウンドタイムが許容できる場合にだけ，この設定を ON にしてください。

それぞれの詳細を次に示します。

(1)　任意のアクティビティに，明示的に補償処理を定義する

補償処理の定義手順と，補償処理の定義例について次に示します。

(a)　補償処理の定義手順

補償処理の定義手順を次に示します。

1. スコープアクティビティを配置します。

   操作の詳細は次の個所を参照してください。

   - アクティビティの配置方法

     「5.4.1　アクティビティを配置する」
   - 各アクティビティの定義方法の詳細

「5.6　アクティビティの定義」

- スコープアクティビティの詳細
  「5.6.15　スコープアクティビティ」

2. スコープアクティビティの中に，補償処理の対象となるアクティビティと，補償処理を実行するアクティビティを配置します。

3. 手順 2.のアクティビティを補償コネクションで連結します。
   アクティビティの連結方法については，「5.4.2(2)　連結の設定方法」を参照してください。また，補償コネクションを利用できるアクティビティについては，「5.4.2(5)　アクティビティの連結条件」を参照してください。

4. スコープアクティビティと同一の階層に戻り，補償アクティビティを配置します。

5. スコープアクティビティと補償アクティビティをフォルトコネクションで連結します。

6. 補償アクティビティをダブルクリックします。
   ［補償アクティビティ］ダイアログが表示されます。
   ［補償アクティビティ］ダイアログの入力・表示内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.18　補償アクティビティダイアログ」を参照してください。

7. 補償するアクティビティを設定します。
   ［補償するアクティビティ］ラジオボタンで，次のどちらかを選択します。
   - 「すべて」：補償範囲内に存在する，補償ハンドラが定義されたすべてのアクティビティに対し，補償処理を実行します。なお，スコープアクティビティのデフォルトハンドラの設定が ON の場合，補償範囲内のスコープアクティビティに暗黙的に定義される補償ハンドラも対象に含みます。補償範囲の詳細については「5.6.10(3)　補償範囲の定義」を参照してください。
   - 「1 つのみ」：コンボボックスで選択したアクティビティに対し，補償処理を実行します。なお，スコープアクティビティのデフォルトハンドラの設定が ON の場合，選択したスコープアクティビティの補償範囲内で暗黙的に定義される補償ハンドラも対象に含みます。

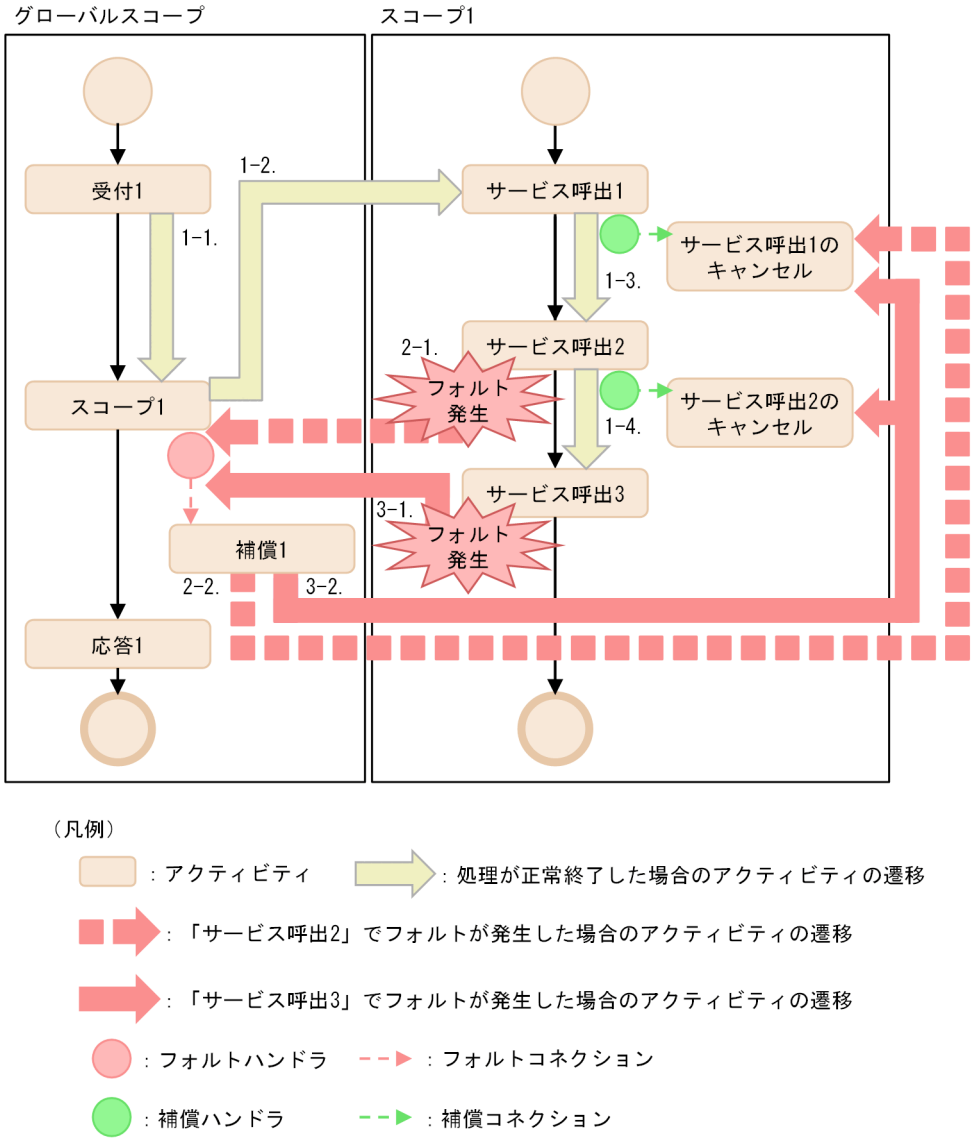
8. ［OK］ボタンをクリックします。

(b)　補償処理の定義例

補償処理の定義例について，図を使って説明します。

次の図に示すとおり，「サービス呼出 1」には補償処理として「サービス呼出 1 のキャンセル」，「サービス呼出 2」には補償処理として「サービス呼出 2 のキャンセル」のように，各アクティビティに補償処理をそれぞれ定義すると，これらの補償処理をまとめて「補償 1」の補償範囲に含められます。1 つの補償アクティビティの実行を契機に，複数の補償処理がまとめて実行されるので，補償アクティビティでの定義が簡単にできます。

図 5-5　1 つの補償アクティビティで複数の補償処理をまとめて実行する場合の定義例と，アクティビティの遷移



図で示したアクティビティの遷移の詳細を，処理が正常終了した場合と，処理の実行中にフォルトが発生した場合に分けて説明します。なお，図中の矢印や部品に付記した番号は，説明文の番号に対応しています。

処理が正常終了した場合のアクティビティの遷移

1-1. 「受付 1」を実行し，「スコープ 1」へ遷移する。

1-2. 「スコープ 1」を実行し，「サービス呼出 1」へ遷移する。

1-3. 「サービス呼出 1」を実行し，「サービス呼出 2」へ遷移する。

1-4. 「サービス呼出 2」を実行し，「サービス呼出 3」へ遷移する。

「サービス呼出 2」でフォルトが発生した場合のアクティビティの遷移

2-1. 「サービス呼出 2」でフォルトが発生し，「スコープ 1」のフォルトハンドラでキャッチする。

2-2. 「補償 1」で「サービス呼出 1 のキャンセル」のアクティビティを呼び出し，実行する。

「サービス呼出 3」でフォルトが発生した場合のアクティビティの遷移

3-1. 「サービス呼出 3」でフォルトが発生し，「スコープ 1」のフォルトハンドラでキャッチする。

3-2. 「補償 1」で「サービス呼出 1 のキャンセル」と「サービス呼出 2 のキャンセル」のアクティビティを呼び出し，実行する。

(2) スコープアクティビティのデフォルトハンドラの設定を ON にし，すべてのスコープアクティビティに対して暗黙的に補償処理を定義する

ここでは次の内容を説明します。

- スコープアクティビティのデフォルトハンドラの設定を ON にする手順
- 暗黙的な補償処理の動作例
- スコープアクティビティのデフォルトハンドラに関する注意事項

(a) スコープアクティビティのデフォルトハンドラの設定を ON にする手順

スコープアクティビティのデフォルトハンドラの設定を ON にする手順を次に示します。

1. ツリービューのサービス定義一覧で表示されたサービスから，該当するビジネスプロセスを選択し，ダブルクリックします。

   選択したビジネスプロセスのビジネスプロセス定義画面が表示されます。

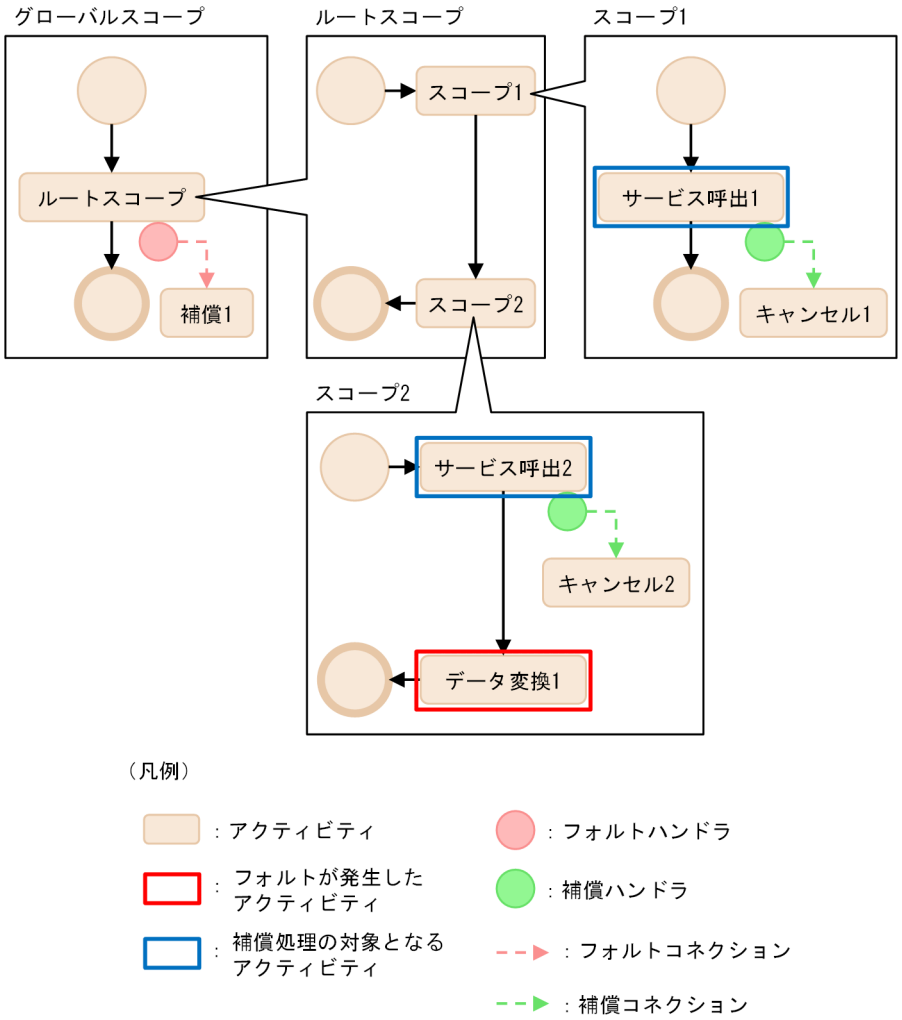
2. プロパティービューの［スコープアクティビティのデフォルトハンドラ］を ON に変更します。

(b) 暗黙的な補償処理の動作例

スコープアクティビティのデフォルトハンドラの設定を ON にすると，補償処理が暗黙的に定義されます。暗黙的に定義された補償処理がどのように動作するか，例を使って説明します。

例として使用するフォルト発生時の状態を次の図に示します。

図 5-6　暗黙的な補償処理の動作例



図で示したとおり，「データ変換 1」でフォルトが発生すると，「サービス呼出 2」「サービス呼出 1」で次のように補償処理が実行されます。

1.「データ変換 1」でフォルトが発生する。
2.「スコープ 2」に暗黙的に定義されているフォルトハンドラが起動する。
3. 2.のフォルトハンドラの補償アクティビティが，「スコープ 2」内の補償ハンドラの起動を実行する。
4.「サービス呼出 2」の補償ハンドラが起動し，「キャンセル 2」を実行する。
5. 2.のフォルトハンドラのフォルト送出アクティビティが，上位のスコープアクティビティにフォルトを送出する。
6.「ルートスコープ」のフォルトハンドラでフォルトをキャッチする。
7.「補償 1」が実行され，「ルートスコープ」内の補償ハンドラの起動を実行する。
8. 7.によって「スコープ 2」に暗黙的に定義されている補償ハンドラを起動しようとするが，「スコープ 2」がフォルト発生状態のため，補償ハンドラが起動しない。

9. 同じく 7.によって「スコープ 1」に暗黙的に定義されている補償ハンドラの起動を実行する。「スコープ 1」は正常終了の状態で，かつ補償処理が実行されていないため，補償ハンドラが起動する。
10. 9.の補償ハンドラの補償アクティビティが，「サービス呼出 1」内の補償ハンドラの起動を実行する。
11. 「サービス呼出 1」の補償ハンドラが起動し，「キャンセル 1」を実行する。

(c) スコープアクティビティのデフォルトハンドラに関する注意事項

- ビジネスプロセスの最上位に，複数のスコープアクティビティが存在する場合，最上位で発生したフォルトは，スコープアクティビティのデフォルトハンドラでキャッチできません。ビジネスプロセスの最上位に，複数のスコープアクティビティが存在する場合で，次のように運用したいときの対処を示します。

  想定する運用

  次の条件に当てはまるビジネスプロセス内のすべてのアクティビティを補償したい。

  ・正常終了している

  ・補償ハンドラが定義されている

  対処

  ビジネスプロセスの最上位に 1 つだけスコープアクティビティを配置し，その配下にアクティビティを定義する。

- スコープアクティビティのデフォルトハンドラの設定を ON にした状態で，ビジネスプロセス内のスコープアクティビティに，catch-all を指定したフォルトハンドラを定義した場合，スコープアクティビティのデフォルトハンドラによる暗黙的なフォルトハンドラは定義されません。また，同様に補償ハンドラを定義した場合も，スコープアクティビティのデフォルトハンドラによる暗黙的な補償ハンドラは定義されません。

  これらの仕様のため，次の条件が重なると，スコープアクティビティのデフォルトハンドラが動作しなくなり，ユーザの意図どおりに補償処理が実行されないおそれがあります。

  - スコープアクティビティのデフォルトハンドラの設定を ON にした状態である
  - ビジネスプロセス内のスコープアクティビティに，フォルトハンドラまたは補償ハンドラを明示的に定義する

5.5　変数・相関セットの定義

5.5.1　変数の定義

ビジネスプロセスを定義する場合，変数には，文字，メッセージに含まれる数値や条件式の項となる数値，またはメッセージを入れることができます。変数の値は，データベースに記録できるため，実行履歴として管理できます。

(1)　変数の種類

変数にはグローバル変数とローカル変数の 2 種類があります。

- **グローバル変数**

  グローバル変数は，1 つのプロセスインスタンス内であればどこからでも参照できます。また，プロセスが完了しても変数の値はデータベースに記録されているため，ビジネスプロセス全体の進行状況を分析できます。変数名は，プロセス内で一意にします。

- **ローカル変数**

  ローカル変数は，宣言したスコープ内で参照できます。宣言しているスコープが終了すると，変数の値もデータベースから削除されるため，プロセス実行中の条件判定など一時的に値を参照するために利用できます。変数名は，スコープ内で一意にします。スコープが異なれば同じ変数名を宣言できます。ただし，変数を参照する場合に，参照個所で最も内側に近いスコープで宣言された変数を参照します。

(2)　変数の型

変数には次に示す型の値を定義できます。変数は，定義した型の値だけを保持できます。

- boolean 型：真偽（true または false）
- numeric 型：数値（64 ビット浮動小数）
- string 型：文字列
- メッセージ型：変数に代入する値は，変数に関連づけられたフォーマット定義に従います。変数の種別によって，次のフォーマットを定義します。

  XML：XML スキーマでフォーマットを定義します。

  non-XML：バイナリフォーマット定義ファイルでフォーマットを定義します。

  any：フォーマットを定義しません。

(3)　変数を割り当てられるアクティビティ

次の表に示すアクティビティの項目について変数を割り当てられます。

表 5-5　変数を割り当てられるアクティビティと項目

アクティビティ	項目	割り当てられる内容	備考
受付アクティビティ	要求電文	サービスリクエスタから受け取る要求電文を参照するメッセージ型の変数	相関セットの指定にも利用されます。
応答アクティビティ	応答電文	サービスリクエスタへの応答電文またはフォルト電文を参照するメッセージ型の変数	
サービス呼出アクティビティ（同期）	要求電文	同期サービスを呼び出す際の要求電文を参照するメッセージ型の変数	
	応答電文	呼び出した同期サービスから受け取る応答電文を参照するメッセージ型の変数	
	フォルト電文	呼び出した同期サービスでフォルトが発生した場合に受け取る応答電文を参照するメッセージ型の変数	－
サービス呼出アクティビティ（非同期）	要求電文	非同期サービスを呼び出す際の要求電文を参照するメッセージ型の変数	相関セットの指定にも利用されます。
Java 呼出アクティビティ	引数	ユーザの作成した Java クラスを呼び出す際の引数となる値を参照する変数	－
	戻り値	呼び出した Java クラスからの戻り値を参照する変数	－
データ変換アクティビティ	変換元データ	データ変換アクティビティでデータ変換を行う際の変換元データを参照するメッセージ型の変数で，複数指定が可能	－
	変換先データ	データ変換アクティビティでデータ変換を行う際の変換先データを参照するメッセージ型の変数	－
代入アクティビティ	コピー元	代入アクティビティでデータのコピーを行う際の，コピー元となるデータを参照する変数	－
	コピー先	代入アクティビティでデータのコピーを行う際の，コピー先となるデータを参照する変数	－
フォルト送出アクティビティ	フォルト電文	フォルト送出アクティビティでフォルトを送出する際に，フォルト電文を参照するメッセージ型の変数	－
待機アクティビティ	待機時間	待機時間の値を参照する変数	－
ログ出力アクティビティ	埋め込み要素	ログメッセージに値を埋め込むために参照する変数	－
検証アクティビティ	検証する変数	検証する変数	－
繰り返しアクティビティ	繰り返し条件	繰り返しのループ変数や終端条件式の項となる値を参照する変数	－
	繰り返しリスト	繰り返しリストを取得するための式で参照する変数	－
	繰り返し要素変数	繰り返し要素を格納する変数	－
分岐アクティビティ	分岐条件	分岐条件の式の項となる値を参照する変数	－

（凡例）

－：該当しません。

(4)　変数の表示とパスの取得

変数・相関セット一覧ダイアログの［表示］ボタンをクリックすると，変数の表示ダイアログが表示されます。変数と一致した XML スキーマをツリービューとして表示できます。ツリービューで表示された変数を選択すると，［選択パス］に選択した変数のパスがルート要素からの絶対パスで表示されます。

また，変数を選択して，右クリックし，［パスの取得］を選択すると，ルート要素からの絶対パスを取得できます。取得したパスは，任意の個所に貼り付けることができます。

注意事項

変数の表示ダイアログを操作する場合は，次の点に注意してください。

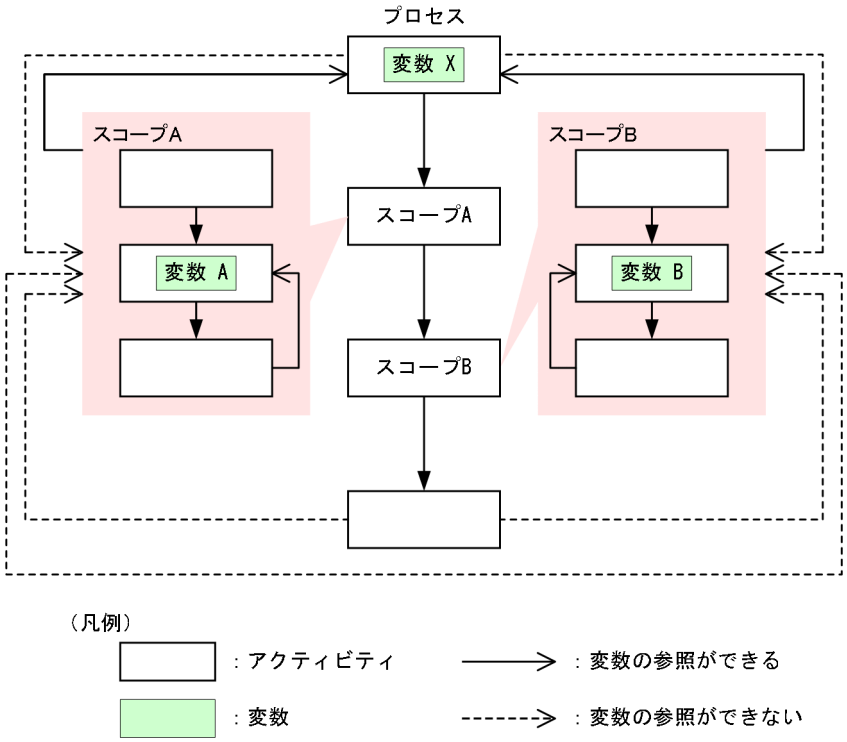
- 変数の表示ダイアログで表示される XML スキーマは編集できません。
- ツリービューで選択できる変数は 1 つだけです。
- 変数の表示ダイアログで表示しようとしている変数の XML スキーマで，要素が再帰になっている場合，2 階層目以降の要素は表示されません。
- XML スキーマの substitutionGroup 属性は無視されます。

(5)　変数を使用する場合の注意事項

(a)　スコープアクティビティを使用している場合の変数の使用

ビジネスプロセス内またはスコープ内のアクティビティで使用する変数を追加したり，編集したりできます。このとき，外側のスコープで宣言した変数は，内側のスコープから参照できますが，内側のスコープで宣言した変数はスコープ外からは参照できません。また，スコープが異なる場合は，同じ変数名を宣言できます。次に示す図の場合，スコープ A やスコープ B はプロセス（外側）の変数を参照できますが，プロセス（外側）からはスコープ A やスコープ B の変数は参照できません。

図 5-7　スコープと変数の関係



(b)　フォルト処理を定義する場合の変数

- スコープアクティビティにフォルト処理を定義する場合に，次の変数を定義するときに注意する点を説明します。
  - ［フォルト処理の割当］ダイアログの割り当て変数
  - フォルトコネクションの接続先のアクティビティで設定する変数

  これらの変数には，フォルト処理を定義したスコープの外側のスコープで定義された変数を定義してください。スコープ内で定義された変数は定義できません。

  これらの変数と，フォルト処理を定義したスコープで定義した変数が，同じ名前でないことを確認してください。同じ名前の変数が定義されていると，アクティビティ実行時にフォルト処理を定義したスコープの変数が使用され，意図しない動作をするおそれがあるためです。
- 次のアクティビティで同じ変数を使用したい場合は，フォルト処理を定義したスコープの外側のスコープで定義された変数を使用するよう定義してください。
  - スコープ内に定義したアクティビティ
  - フォルトコネクションの接続先のアクティビティ

(c)　補償処理を定義する場合の変数

- スコープアクティビティに補償処理を定義する場合に，補償コネクションの接続先のアクティビティで設定する変数を定義するときに注意する点を説明します。
  - この変数には，補償処理を定義したスコープの外側のスコープで定義された変数を定義してください。スコープ内で定義された変数は定義できません。

- この変数と，補償処理を定義したスコープで定義した変数が，同じ名前でないことを確認してください。同じ名前の変数が定義されていると，アクティビティ実行時に補償処理を定義したスコープの変数が使用され，意図しない動作をするおそれがあるためです。

- スコープの実行が完了すると，スコープに定義されたローカル変数は削除されます。そのため，次のアクティビティで同じ変数を使用したい場合は，補償処理を定義したスコープの外側のスコープで定義された変数を使用するよう定義してください。
  - スコープ内に定義したアクティビティ
  - 補償コネクションの接続先のアクティビティ

(d) 変数の定義で設定する電文フォーマット

電文フォーマットのファイル名

変数の定義時には，同じファイル名で異なる XML スキーマの電文フォーマットである定義ファイルを複数の変数で設定できます。しかし，実行時には，どれか 1 つの XML スキーマが使用されるため，意図しない動作をする場合があります。したがって，変数の定義で設定する電文フォーマットのファイル名と XML スキーマは，必ず 1 対 1 になるように設定してください。

外部の XML スキーマの参照を含む電文フォーマットの設定

外部の XML スキーマを参照している電文フォーマットを設定する場合，必ずルートスキーマに当たるファイルを設定してください。ルートスキーマから参照される外部の XML スキーマファイルは，自動的に取り込まれます。

電文フォーマットの形式

変数に使用する電文フォーマットは，一定の条件を満たしている必要があります。条件の詳細については，「2.6.5　XML スキーマの適用範囲」を参照してください。

(e) 変数の定義情報の変更

変数および相関セットの定義情報（変数名，型，電文フォーマット，部分）は，必要に応じて変更または削除できます。

注意事項

変数の定義情報を変更する場合は，次の点に注意してください。

- データ変換アクティビティで使用している変数の電文フォーマットを変更した場合は，データ変換を定義し直してください。電文フォーマットを変更した場合の手順については，「7.3.2 電文フォーマットを変更した場合の定義手順」を参照してください。
- データ変換定義ファイルが定義済みのデータ変換アクティビティで使用している変数の変数名を変更した場合，警告メッセージが表示されます。この場合，変更した変数に関わるマッピングは破棄されます。
- 変数の定義情報を変更すると，その変数を割り当てた個所に，再度割り当てが必要になる場合があります。

例えば，あるアクティビティに変数 X を割り当て，その後，その変数 X の名前を変数 Y に変更した場合，次のどちらかを行ってください。

・アクティビティへ再度変数を割り当てる。

・再度変数 X を定義する。

- ビジネスプロセスをバージョンアップしたあとに変数を変更する場合は注意が必要です。詳細は，「5.9.4(3)　ビジネスプロセスのバージョンアップ時の注意事項」を参照してください。

(f)　変数の定義情報の初期化

ビジネスプロセスを実行するときに，変数には，文字，メッセージに含まれる数値や条件式の項となる数値，またはメッセージを設定する必要があります。

サービスリクエスタからの要求を受け付けたときや，サービス呼出アクティビティでサービスからの応答を受け取ったときは，変数に値が入っていますが，それ以外はデータ変換アクティビティや代入アクティビティなどを使用して変数に値を設定（初期化）する必要があります。

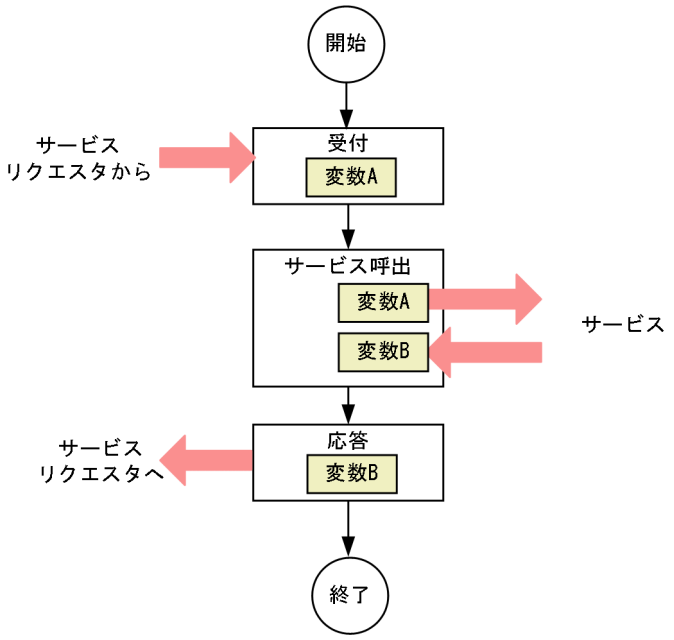
変数の初期化が不要な場合について説明します。

次の個所に指定している変数は，変数（電文）を参照する前に値が設定されているため，変数の初期化は不要です。

- サービスリクエスタから要求を受け付けた変数
- サービス呼出アクティビティの応答電文のボディ割当変数/ヘッダ割当変数
- Java 呼出アクティビティの戻り値用割当変数
- データ変換アクティビティの変換先変数
- 代入アクティビティのコピー先の変数
- フォルトコネクションに指定した割当変数

変数の初期化が不要な場合の例を次の図に示します。

図 5-8　変数の初期化が不要な場合の例



変数 A：

サービスリクエスタからの要求電文が変数の値として設定されます。また，サービス呼出は値が設定済みの変数 A を要求電文としているため，初期化は不要です。

変数 B：

サービスからの応答電文で変数が設定されるため，初期化は不要です。

変数の初期化が必要な場合について説明します。

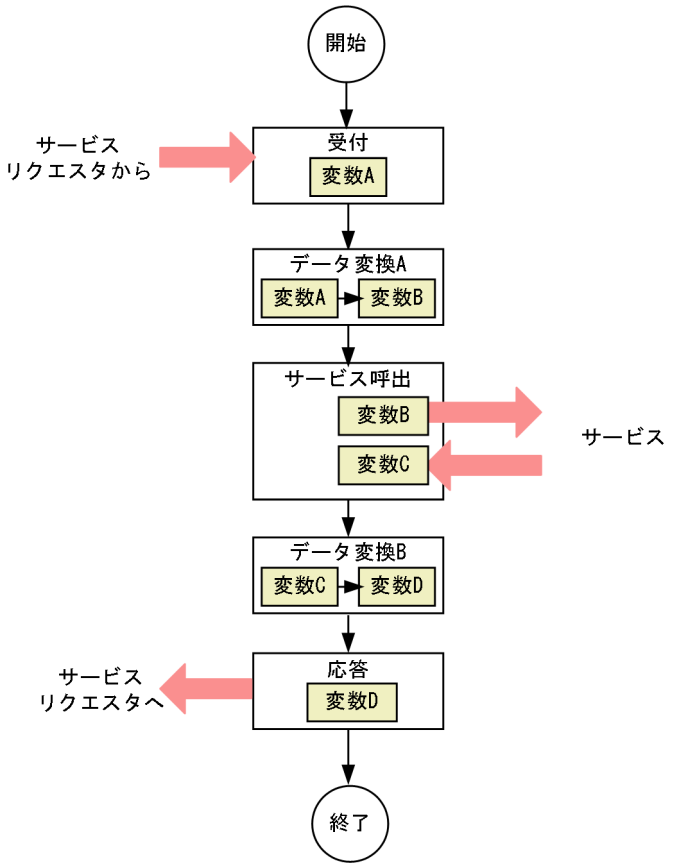
変数の初期化が不要な場合に示した個所に指定していない変数は値が設定されていないため，ビジネスプロセス上で参照するとエラーになります。事前に値を設定し，変数を初期化しておく必要があります。

値が設定されていない変数を指定する可能性のある個所を次に示します。

- サービス呼出アクティビティの要求電文のボディ割当変数/ヘッダ割当変数
- Java 呼出アクティビティの引数用割当変数
- データ変換アクティビティの変換元変数
- 代入アクティビティのコピー元の変数
- 待機アクティビティ，検証アクティビティ，繰り返しアクティビティ，分岐アクティビティ，リンクコネクションの XPath 式内で変数名を記述している個所
- 応答アクティビティのボディ割当変数/ヘッダ割当変数

変数の初期化が必要な場合の例を次の図に示します。

図 5-9　変数の初期化が必要な場合の例



変数 A：

サービスリクエスタからの要求電文が変数に設定されているため，初期化は不要です。

変数 B：

変数の初期化が必要になります。データ変換 A で変数 A の情報を基に，変数 B に値を設定します。

変数 C：

サービスからの応答電文が変数に設定されるため，初期化は不要です。

変数 D：

変数の初期化が必要になります。データ変換 B で変数 C の情報を基に，変数 D に値を設定します。

(6)　変数の定義方法

(a)　新規に変数を定義する場合

新規に変数を定義する手順を次に示します。なお，変数は，繰り返しアクティビティ内のキャンバスでは定義できません。

1. 次のどれかの方法で，変数・相関セット一覧ダイアログを表示します。
   - ビジネスプロセス定義画面のキャンバス上の［変数・相関セット］アイコンをダブルクリックします。
   - 次に示すアクティビティダイアログの［編集］ボタンをクリックします。

・受付アクティビティ
・応答アクティビティ
・サービス呼出アクティビティ
・Java 呼出アクティビティ
・データ変換アクティビティ
・代入アクティビティ
・フォルト送出アクティビティ
・検証アクティビティ

- 次に示すダイアログのボタンをクリックします。

  ・[条件式の作成]ダイアログ（繰り返しアクティビティまたは分岐アクティビティから表示）の[編集]ボタン

  ・繰り返しアクティビティの[繰り返しリスト設定]ダイアログの[編集]ボタン

2. 一覧から「変数一覧」を選択します。

3. 変数名を入力します。

4. 変数の種別をドロップダウンリストから選択します。

5. 変数の種別がメッセージ型の場合，必要に応じて，次のどちらかの操作をします。

呼び出すサービス部品またはユーザ定義受付の要求電文，応答電文，またはフォルト電文に設定されている電文フォーマットを，変数に設定する場合

[取込]ボタンをクリックし，[電文フォーマットの取込]ダイアログを表示します。[電文フォーマットの取込]ダイアログで，使用する電文フォーマットを指定します。

[電文フォーマットの取込]ダイアログの入力・表示項目の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5　電文フォーマットの取込ダイアログ」を参照してください。

[電文フォーマットの取込]ダイアログでの操作の終了後，手順 6.に進みます。

上記以外の電文フォーマットを変数に設定する場合

[参照]ボタンをクリックして，変数を使用する電文フォーマットの定義ファイルを設定します。

サービスアダプタからサービス部品を呼び出したときに発生したエラーの例外を，サービスアダプタでフォルト電文に変換する場合は，[参照]ボタンをクリックして，<サービスプラットフォームのインストールディレクトリ>¥CSC¥system¥msg¥cscfault.xsd を選択してください。

6. 変数の種別がメッセージ型の場合，[部分指定]チェックボックスにチェックします。

7. [行追加]ボタンをクリックし，部分名，指定式，および型を指定します。

8. [追加]ボタンをクリックします。

変数一覧に，追加した変数が表示されます。

9. [OK]ボタンをクリックします。

注意事項

変数・相関セット一覧ダイアログから電文フォーマットの定義ファイルを指定する場合，ファイル名は NCName 型で指定します。

NCName 型ではない場合，指定時にエラーとなります。

(b)　電文フォーマットの定義ファイルを変更した場合

変数の定義で設定した電文フォーマットの定義ファイルを変更した場合，変更後の定義ファイルの内容は，次の手順で反映できます。

1. 変数・相関セット一覧ダイアログを表示します。

   変数・相関セット一覧ダイアログを表示する方法については，「5.5.1(6)(a)　新規に変数を定義する場合」の手順 1.を参照してください。

2. 次のどちらかの操作をします。

   変数の種別がメッセージ型の場合で，呼び出すサービス部品またはユーザ定義受付の要求電文，応答電文，またはフォルト電文に設定されている電文フォーマットを，変数に設定する場合

   ［取込］ボタンをクリックし，［電文フォーマットの取込］ダイアログを表示します。［電文フォーマットの取込］ダイアログで，使用する電文フォーマットを指定します。

   ［電文フォーマットの取込］ダイアログの入力・表示項目の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5　電文フォーマットの取込ダイアログ」を参照してください。

   ［電文フォーマットの取込］ダイアログでの操作の終了後，手順 3.に進みます。

   変数の種別がメッセージ型の場合で，上記以外の場合

   ［参照］ボタンをクリックして，変数を使用する電文フォーマットの定義ファイルを設定します。

3. ［更新］ボタンをクリックします。

(7)　電文フォーマットの定義ファイルの出力

電文フォーマットの定義ファイルの出力方法について説明します。

コピー元のアクティビティに設定した変数の電文フォーマットをコピー先のアクティビティの変数に設定したい場合などに，変数・相関セット一覧ダイアログで電文フォーマットの定義ファイルを出力できます。

出力できる電文フォーマットの定義ファイルの形式は次のどちらかです。

- 電文フォーマットの定義ファイルの拡張子：xsd（変数の型：XML）
- 電文フォーマットの定義ファイルの拡張子：fdx（変数の型：non-XML）

電文フォーマットの定義ファイルの出力手順を次に示します。

1. 変数・相関セット一覧ダイアログの［出力］ボタンをクリックします。
   出力先を指定するためのダイアログが表示されます。

2. 出力先を指定して［OK］ボタンをクリックします。
   指定したフォルダに電文フォーマットの定義ファイルが出力されます。出力対象が複数の定義ファイルから成るメッセージ型（XML）変数の場合は，出力ファイル名が自動的に与えられます。

(8) ユーザ定義受付（呼出先選択）の電文フォーマットの取込

ビジネスプロセスにユーザ定義受付を定義していない場合，ユーザ定義受付（呼出先選択）を定義した電文フォーマットをビジネスプロセスに取り込めます。

電文フォーマットの作成については「4. 電文フォーマットの作成」を，［電文フォーマットの取込］ダイアログの入力・表示項目の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5 電文フォーマットの取込ダイアログ」を参照してください。

5.5.2 相関セットの定義

相関セットは，サービスリクエスタから送信された要求電文を一意に識別するために利用する文字列です。

(1) 相関セットによるプロセスインスタンスの識別

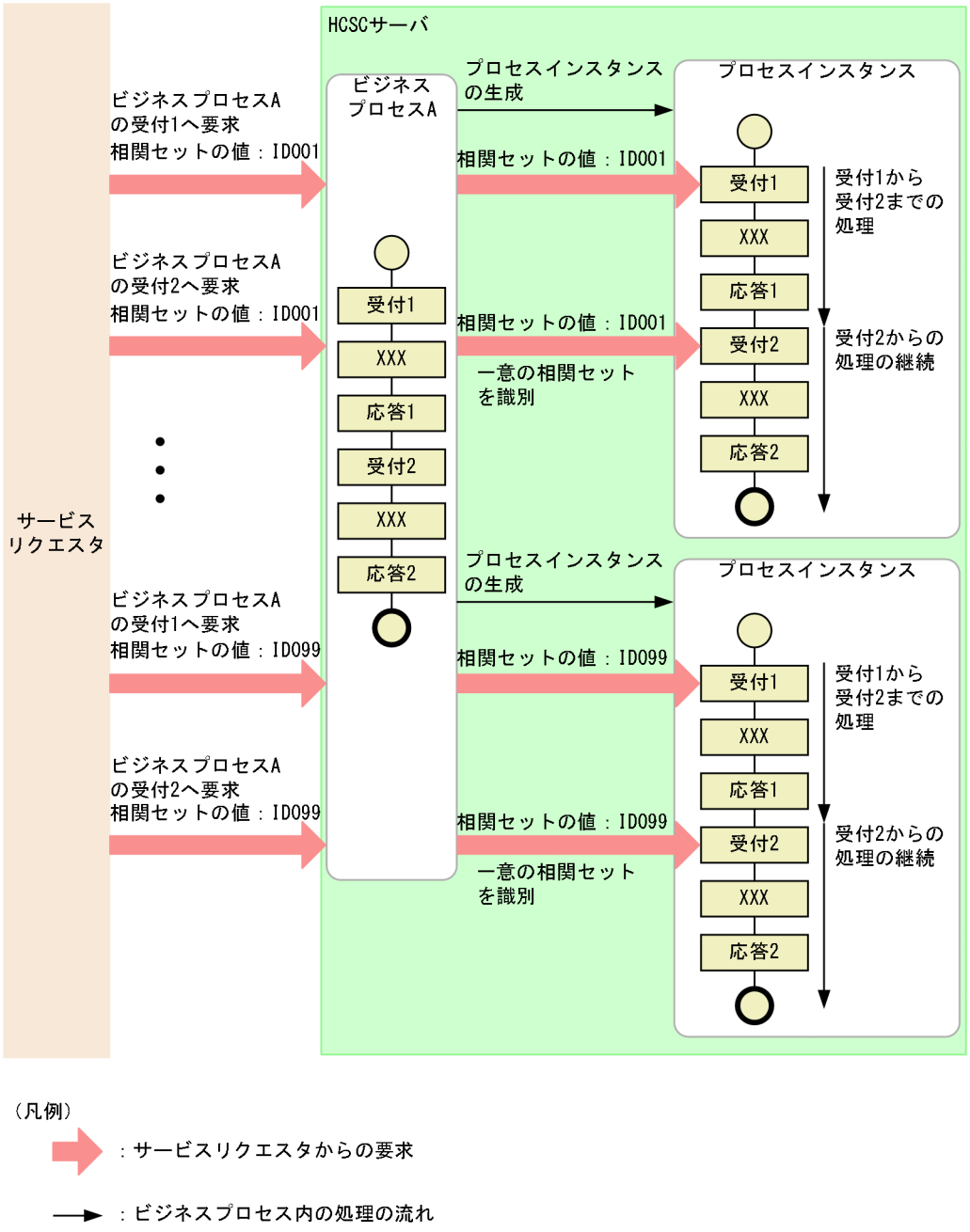
ビジネスプロセスは，受付アクティビティダイアログの［インスタンス生成］に［yes］を指定した受付アクティビティへの要求ごとに，プロセスインスタンスを生成します。

サービスリクエスタから要求電文を受け取れるプロセスインスタンスが HCSC サーバ上に複数ある場合，HCSC サーバは要求電文に含まれる相関セットの値を利用してプロセスインスタンスを特定します。

例えば，あるビジネスプロセスに，サービスリクエスタからの要求を受け付ける受付アクティビティが 2 つ定義されているとします。1 つ目の受付アクティビティへ要求を行い，2 つ目の受付アクティビティでビジネスプロセスが待ち受け状態になっている場合，1 つ目の受付アクティビティへの要求電文に含まれる相関セットの値と同じ値を 2 つ目の受付アクティビティへの要求電文に含めます。相関セットに同じ値を指定することで，2 つの要求が同じプロセスインスタンスへのリクエストであると特定できます。これによって，2 つ目の受付アクティビティ以降の処理を継続できます。

相関セットによるプロセスインスタンスの識別の例を次の図に示します。

図 5-10　相関セットによるプロセスインスタンスの識別の例



（凡例）

：サービスリクエスタからの要求

：ビジネスプロセス内の処理の流れ

複数の受付アクティビティがあるビジネスプロセスを実現するには，要求電文に含まれる値のうち，どの値を相関セットに使用するかをビジネスプロセスに定義する必要があります。ビジネスプロセスおよびサービスリクエスタを設計するときに，プロセスインスタンスを一意に特定するキーとなる値をサービスリクエスタからの要求電文に設定してください。

相関セットは，要求電文に含まれる 1 つの部分または複数の部分の連結によって構成することができます。

ビジネスプロセスで相関セットを使用すると，相関セットの値をキーとして，プロセスインスタンスの実行履歴を検索できます。実行履歴を検索することで，特定のリクエストの実行状態を確認できます。プロセスインスタンスの実行履歴の詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「6.1　プロセスインスタンスの実行履歴の管理」を参照してください。

(2) 相関セットを割り当てられるアクティビティ

ビジネスプロセスでは，受付アクティビティ，応答アクティビティおよびサービス呼出アクティビティに相関セットを設定できます。また，1 つの電文には複数の相関セットを設定できます。

アクティビティごとに，相関セットの設定の要否，および相関セットの役割が異なります。

(a) 受付アクティビティ，応答アクティビティの場合

受付アクティビティおよび応答アクティビティでは，インスタンスの生成の設定，割り当てた相関セットを初期化するかどうかの設定によって，相関セットの指定の要否および相関セットの内容が異なります。

インスタンスの生成の設定は，受付アクティビティダイアログで設定できます。また，相関セットを初期化するかどうかの設定は，割当相関セット群ダイアログで設定できます。

受付アクティビティダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.4.7　受付アクティビティダイアログ」を参照してください。割当相関セット群ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.4.3　割当相関セット群ダイアログ」を参照してください。

表 5-6　相関セットの設定の要否と内容（受付・応答）

アクティビティ	インスタンスの生成	初期化の設定	設定の要否	最大設定数	設定した相関セットの内容
受付	yes	yes	任意	1	プロセスインスタンスを生成するとき，プロセスインスタンスを識別するための相関セットが，変数として定義されている受信メッセージから生成されます。
		no	設定不可	－	－
	no	yes	任意	1	生成済みのプロセスインスタンスを検索したあと，検索されたプロセスインスタンスに対する新たな相関セットが，変数として定義されている受信メッセージから生成されます。
		no	必須	1	指定した相関セットは，生成済みのプロセスインスタンスを検索するために，変数として定義されている受信メッセージ内の相関セットとして利用されます。
応答	－	yes	任意	1	サービスリクエスタに応答を返すとき，変数として定義されている応答メッセージから新たな相関セットが生成されます。
	－	no	任意	1	サービスリクエスタに応答を返すとき，変数として定義されている応答メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。

（凡例）

－：該当しません。

(b)　サービス呼出アクティビティの場合

サービス呼出アクティビティでは，割り当てた相関セットのパターンの設定，および相関セットを初期化するかどうかの設定によって，相関セットの指定の要否および相関セットの内容が異なります。

割り当てた相関セットのパターンの設定，および相関セットを初期化するかどうかの設定は，割当相関セット群ダイアログで設定できます。

割当相関セット群ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.3　割当相関セット群ダイアログ」を参照してください。

表 5-7　相関セットを割り当てられるアクティビティ（サービス呼出）

アクティビティ	パターンの設定	初期化の設定	設定の要否	最大設定数	設定した相関セットの内容
サービス呼出	out	yes	任意	1	サービス部品にメッセージを送信するとき，変数として定義されている送信メッセージから新たな相関セットが生成されます。
		no	任意	1	サービス部品にメッセージを送信するとき，変数として定義されている送信メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。
	in	yes	任意	1	サービス部品から応答を受けたとき，変数として定義されている応答メッセージから新たな相関セットが生成されます。
		no	任意	1	サービス部品から応答を受けたとき，変数として定義されている応答メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。
	out-in	yes	任意	1	送信時 サービス部品にメッセージを送信するとき，変数として定義されている送信メッセージから新たな相関セットが生成されます。 受信時 サービス部品から応答を受けたとき，変数として定義されている応答メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。
		no	任意	1	送信時 サービス部品にメッセージを送信するとき，変数として定義されている送信メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。 受信時 サービス部品から応答を受けたとき，変数として定義されている応答メッセージ内の相関セットの値と，プロセスインスタンスの相関セットの値とが一致しているかどうかがチェックされます。

(3)　相関セットを使用する場合の注意事項

(a)　スコープアクティビティを使用している場合の相関セットの使用

ビジネスプロセス内またはスコープ内のアクティビティで使用する相関セットを，変数と同様に追加したり，編集したりできます。詳細は，「5.5.1(5)(a)　スコープアクティビティを使用している場合の変数の使用」を参照してください。

(b)　相関セットの定義情報の変更

相関セットの定義情報は必要に応じて変更または削除できます。

相関セットの定義で使用されている変数の定義情報を変更する場合の注意事項については，「5.5.1(5)(e) 変数の定義情報の変更」を参照してください。

ビジネスプロセスをバージョンアップしたあとに相関セットを変更する場合は注意が必要です。詳細は，「5.9.4(3)　ビジネスプロセスのバージョンアップ時の注意事項」を参照してください。

注意事項

相関セットの定義情報を変更すると，その相関セットを割り当てた個所に，再度割り当てが必要になる場合があります。

例えば，あるアクティビティに相関セット A を割り当て，その後，その相関セット A の名前を相関セット B に変更した場合，次のどちらかを行ってください。

- アクティビティへ再度相関セットを割り当てる。
- 再度相関セット A を定義する。

(c)　相関セットの有効範囲

ビジネスプロセスのグローバル変数で相関セットが宣言されている場合，該当するプロセスインスタンスの処理が完了するまでの間が有効範囲となります。

また，スコープ内のアクティビティで使用する相関セットを宣言している場合，該当するスコープ内の処理が完了するまでの間が有効範囲となります。

処理が完了した時点で相関セットは無効となります。

(d)　フォルト処理を定義する場合の相関セット

- スコープアクティビティにフォルト処理を定義する場合に，フォルトコネクションの接続先のアクティビティに相関セットを定義するときに注意する点を説明します。
  - フォルトコネクションの接続先のアクティビティには，フォルト処理を定義したスコープの外側のスコープで定義された相関セットを定義してください。スコープ内で定義された相関セットは定義できません。

- フォルトコネクションの接続先のアクティビティに定義した相関セットと，フォルト処理を定義したスコープで定義した相関セットが，同じ名前でないことを確認してください。同じ名前の相関セットが定義されていると，アクティビティ実行時にフォルト処理を定義したスコープの相関セットが使用され，意図しない動作をするおそれがあるためです。

- 次のアクティビティで同じ相関セットを使用したい場合は，フォルト処理を定義したスコープの外側のスコープで定義された相関セットを使用するよう定義してください。
  - スコープ内に定義したアクティビティ
  - フォルトコネクションの接続先のアクティビティ

(e) 補償処理を定義する場合の相関セット

- スコープアクティビティに補償処理を定義する場合に，補償コネクションの接続先のアクティビティに相関セットを定義するときに注意する点を説明します。
  - 補償コネクションの接続先のアクティビティには，補償処理を定義したスコープの外側のスコープで定義された相関セットを定義してください。スコープ内で定義された相関セットは定義できません。
  - 補償コネクションの接続先のアクティビティに定義した相関セットと，補償処理を定義したスコープで定義した相関セットが，同じ名前でないことを確認してください。同じ名前の相関セットが定義されていると，アクティビティ実行時に補償処理を定義したスコープの相関セットが使用され，意図しない動作をするおそれがあるためです。
- スコープの実行が完了すると，スコープに定義された相関セットは無効化されます。そのため，次のアクティビティで同じ相関セットを使用したい場合は，補償処理を定義したスコープの外側のスコープで定義された相関セットを使用するよう定義してください。
  - スコープ内に定義したアクティビティ
  - 補償コネクションの接続先のアクティビティ

(4) 相関セットの定義方法

(a) 相関セットの新規定義

相関セットを定義するには，あらかじめ変数を定義し，定義した変数の中で相関セットとして利用する部分の部分指定を定義しておきます。新規に相関セットを定義する手順を次に示します。なお，相関セットは，繰り返しアクティビティ内のキャンバスでは定義できません。

1. 次のどちらかの方法で，変数・相関セット一覧ダイアログを表示します。
   - ビジネスプロセス定義画面のキャンバス上の［変数・相関セット］アイコンをクリックします。
   - 次に示すアクティビティダイアログの［編集］ボタンをクリックします。
     ・受付アクティビティ
     ・応答アクティビティ
     ・サービス呼出アクティビティ

2. 一覧から「相関セット一覧」を選択します。

3. 相関セット名を入力します。

4. [行追加] ボタンをクリックし，[取得部] に利用変数および部分名を追加し，相関セットとして使用する部分を指定します。

   相関セットに指定できるのは，メッセージ型の変数（XML 型）で，かつメッセージ型以外の型を指定した部分名です。

   利用するメッセージ型の変数が複数ある場合は，利用変数および部分名をドロップダウンリストから選択します。

   なお，複数の部分を連結して連結後の文字列を相関セット値として設定する場合は，さらに［行追加］ボタンをクリックして利用変数および部分名を追加します。

5. [追加] ボタンをクリックします。

   相関セット一覧に，追加した相関セットが表示されます。

6. [OK] ボタンをクリックします。

(b) アクティビティからの相関セットの設定

「表 5-7 相関セットを割り当てられるアクティビティ（サービス呼出）」に示すアクティビティから，相関セットを設定する手順を次に示します。

1. 各アクティビティのダイアログの［設定］ボタンをクリックします。

   割当相関セット群ダイアログが表示されます。

2. アクティビティに割り当てる相関セットを［相関セット群］から選択します。

3. [相関セット群] にない場合は，[編集] ボタンをクリックします。

4. 変数・相関セット一覧ダイアログが表示されるので，相関セットを追加します。

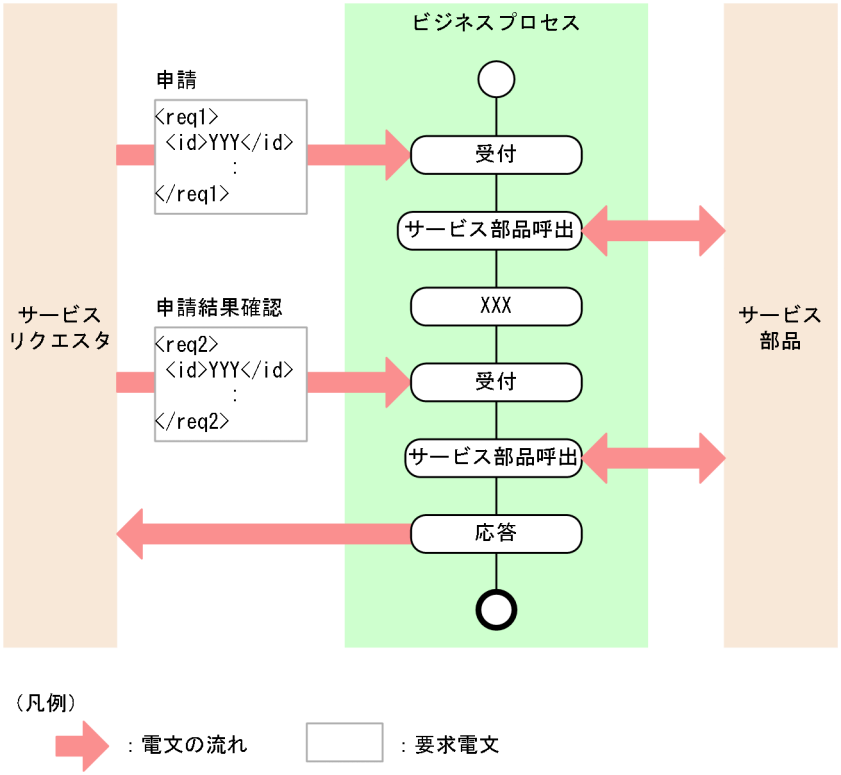
5. 割当相関セット群ダイアログで初期化するかどうかを指定します。

(5) 相関セットを利用した定義の例

相関セットを利用するビジネスプロセスの定義について，例を用いて説明します。

次の図に示すように，申請とその結果を非同期で確認する申請結果確認のリクエストを受け付けるビジネスプロセスを例として利用します。

図 5-11　定義するビジネスプロセスの例



上記の図に示したビジネスプロセスを定義する場合の変数，相関セット，および受付アクティビティの定義内容を次に示します。

変数および相関セットの定義

次の表に示すように，利用する変数と相関セットを定義します。変数および相関セットは，［変数・相関セット一覧］ダイアログで定義します。［変数・相関セット一覧］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

表 5-8　申請の要求電文に対応する変数の定義内容（例）

［変数・相関セット一覧］ダイアログ（変数情報の表示）での設定項目		設定する値
変数名		申請メッセージ
種別		XML
電文フォーマット		申請メッセージ.xsd
部分指定	部分名	申請 ID
	指定式	/req1/id
	型	string

表 5–9　申請結果確認の要求電文に対応する変数の定義内容（例）

[変数・相関セット一覧] ダイアログ（変数情報の表示）での設定項目		設定する値
変数名		申請確認メッセージ
種別		XML
電文フォーマット		申請確認メッセージ.xsd
部分指定	部分名	申請 ID
	指定式	/req2/id
	型	string

表 5–10　相関セットの定義内容（例）

[変数・相関セット一覧] ダイアログ（相関セット情報の表示）での設定項目		設定する値
相関セット名		申請相関 ID
取得部	利用変数	申請メッセージ※
	部分名	申請 ID

注※

利用変数には「申請メッセージ」と「申請確認メッセージ」のどちらでも指定できます。

受付アクティビティの定義

次の表に示すように，受付アクティビティを定義します。受付アクティビティは受付アクティビティダイアログで定義します。受付アクティビティダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.4.7　受付アクティビティダイアログ」を参照してください。

表 5–11　申請のリクエストを受け付ける受付アクティビティの定義内容（例）

[受付アクティビティ] ダイアログでの設定項目		設定する値
アクティビティ名		申請受付
オペレーション名		request
割当変数		申請メッセージ
割当相関セット群	相関セット名	申請相関 ID
	初期化	yes
通信モデル		非同期
インスタンス生成		yes

表 5-12　申請結果確認のリクエストを受け付ける受付アクティビティの定義内容（例）

［受付アクティビティ］ダイアログでの設定項目		設定する値
アクティビティ名		確認受付
オペレーション名		confirm
割当変数		申請確認メッセージ
割当相関セット群	相関セット名	申請相関 ID
	初期化	no
通信モデル		同期
インスタンス生成		no

5.6　アクティビティの定義

アクティビティは，キャンバス上に配置・連結するとともに，詳細内容の定義をする必要があります。

定義できるアクティビティと定義内容の概要を次の表に示します。

表 5-13　定義できるアクティビティ

アクティビティ	定義内容
開始	ビジネスプロセスの開始を表します。
受付※	サービスリクエスタからの要求電文を受け付けるためのインターフェースを定義します。
応答※	サービスリクエスタから同期受信した要求電文に対する応答またはフォルトを返すためのインターフェースを定義します。
サービス呼出※	定義済みの HCSC コンポーネントへの要求電文の送信を定義します。
Java 呼出	専用インターフェースを実装した Java クラスの呼び出しを定義します。
データ変換	データ変換処理を定義します。
代入	次の処理を実行するための定義をします。 • ある変数（基本型，メッセージ型）をほかの変数に代入する • ある変数の一部をほかの変数に代入する • 値（数値，文字列，真偽値）を作成し，変数に代入する
無操作	実行しても何の動作も行わないことを定義します。
フォルト送出	上位のスコープアクティビティへのフォルトの通知を定義します。
補償	どのアクティビティで補償処理を実行するかについて定義します。
待機※	処理フローを一定の間隔または期限まで待機させる処理を定義します。
検証	ビジネスプロセス内で送受信される電文の妥当性を検証します。
ログ出力	ログファイルに出力するメッセージの内容を定義します。
途中終了	ビジネスプロセスを途中終了する処理を定義します。
スコープ	1 つまたは複数のアクティビティから構成される処理フローを，ひとつの単位（スコープ）として定義します。
繰り返し	1 つまたは複数のアクティビティを，指定した条件で繰り返し処理するための定義をします。
分岐（開始）	条件式の結果によってビジネスプロセスの処理の分岐を定義します。分岐の開始個所と終了個所で定義が必要です。
分岐（終了）	
並列処理（開始）	処理フローを複数のシーケンスに分けて同時に実行する（処理フローの並列化）処理を定義します。 並列処理の開始個所と終了個所で定義が必要です。
並列処理（終了）	
終了	ビジネスプロセスの終了を表します。

注※

ビジネスプロセスにステータスの永続化が設定されている場合，受付，応答，サービス呼出，待機アクティビティの処理が，アクティビティの状態を確定するタイミング（トランザクションの開始とコミットのタイミング）になります。

トランザクションの開始とコミットのタイミングについては，マニュアル「サービスプラットフォーム 解説」の「3.4　ビジネスプロセスのトランザクション」を参照してください。

各アクティビティの定義の詳細について，以降の項で説明します。また，アクティビティを定義するダイアログには，変数の情報を取得するために XPath を指定する場合もあります。XPath の指定方法については，「5.6.21　アクティビティ定義ダイアログの XPath の指定」を参照してください。

参考

開始アクティビティおよび終了アクティビティ以外のアクティビティは，定義内容を保持したままビジネスプロセス画面上で任意の場所にコピーすることができます。

ただし，アクティビティのコピー時に，参照する変数がコピー先に存在しないなど，アクティビティ間で変数の不整合が発生する場合があります。この場合，コピー元の変数の電文フォーマットを出力して，コピー先の変数に設定してください。変数に設定した電文フォーマットの出力方法については，「5.5.1(7)　電文フォーマットの定義ファイルの出力」を参照してください。

5.6.1　開始アクティビティ

ビジネスプロセスの開始を表すアクティビティです。キャンバス上に 1 つだけ配置されます。このアクティビティは削除できません。

なお，開始アクティビティには，定義する内容はありません。

5.6.2　受付アクティビティ

ビジネスプロセスがサービスリクエスタからの要求電文を受け付けるためのインターフェースを定義するアクティビティです。1 つのビジネスプロセス内にサービスリクエスタに公開するオペレーションが複数ある場合，そのオペレーションの数だけ定義が必要です。

受付アクティビティは，［受付アクティビティ］ダイアログで詳細を定義します。

(1)　定義手順

定義の手順を次に示します。

1. 受付アクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 次のどちらかの方法で［受付アクティビティ］ダイアログを表示させます。

- キャンバスの受付アクティビティをダブルクリックする
- キャンバスの受付アクティビティを選択して右クリックし，［設定］を選択する

［受付アクティビティ］ダイアログが表示されます。

3. ［受付アクティビティ］ダイアログに必要な情報を入力します。

［受付アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.7　受付アクティビティダイアログ」を参照してください。

- ［オペレーション名］に設定するオペレーション名を取り込む場合

  ［取込］ボタンをクリックします。表示される［オペレーション名の取込］ダイアログで，取り込むオペレーション名を指定します。［オペレーション名の取込］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.8　オペレーション名の取込ダイアログ(受付アクティビティ)」を参照してください。

- ［ボディ割当変数］に設定する変数の内容を編集する場合

  ［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。この項目には，メッセージ型（XML，non-XML または any）の変数を設定できます。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

  なお，該当するオペレーションを持つユーザ定義受付を定義している場合，ユーザ定義受付の電文フォーマットを変数に設定できます。設定は，［電文フォーマットの取込］ダイアログで行います。［電文フォーマットの取込］ダイアログは，［変数・相関セット一覧］ダイアログの［取込］ボタンをクリックすると表示されます。

  ［電文フォーマットの取込］ダイアログを表示する際，［受付アクティビティ］ダイアログでオペレーション名を設定しておくと，ユーザ定義受付内にある該当するオペレーションの電文フォーマットが選択された状態になります。

  ［電文フォーマットの取込］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5　電文フォーマットの取込ダイアログ」を参照してください。

- ［ヘッダ割当変数］を設定する場合

  ［設定］ボタンをクリックします。表示される［ヘッダ割当変数］ダイアログでヘッダに割り当てる変数を設定します。この項目には，メッセージ型（XML）の変数を設定できます。メッセージ型（non-XML または any）の変数は設定できません。［ヘッダ割当変数］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.2　ヘッダ割当変数ダイアログ」を参照してください。

- ［割当相関セット群］を設定する場合

  ［設定］ボタンをクリックします。表示される［割当相関セット群］ダイアログで割当相関セットを設定します。［割当相関セット群］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.3　割当相関セット群ダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- オペレーション名は，ビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- 同期の受付アクティビティの場合，そのオペレーションに対応する応答アクティビティを設定してください。
- 同期の受付アクティビティの場合，そのオペレーションに対応する応答アクティビティの間には，別の受付アクティビティを設定しないでください。
- インスタンスを生成するすべての受付アクティビティは，使用する相関セットを同じになるように設定してください。
- インスタンスを生成する受付アクティビティを 1 つ以上作成してください。
- ビジネスプロセスをバージョンアップしたあとに受付アクティビティの定義情報を変更する場合は注意が必要です。詳細は，「5.9.4(3) ビジネスプロセスのバージョンアップ時の注意事項」を参照してください。

5.6.3 応答アクティビティ

ビジネスプロセスがサービスリクエスタから同期受信した要求電文に対して，応答またはフォルトを返すためのインターフェースを定義するアクティビティです。対応する受付アクティビティの通信モデルに同期受付を定義した場合，必ず定義します。

応答アクティビティは，［応答アクティビティ］ダイアログで詳細を定義します。

(1) 定義手順

定義の手順を次に示します。

1. 応答アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1 アクティビティを配置する」を参照してください。
2. 次のどちらかの方法で［応答アクティビティ］ダイアログを表示させます。
   - キャンバスの応答アクティビティをダブルクリックする
   - キャンバスの応答アクティビティを選択して右クリックし，［設定］を選択する

   ［応答アクティビティ］ダイアログが表示されます。
3. ［応答アクティビティ］ダイアログに必要な情報を入力します。

［応答アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.9　応答アクティビティダイアログ」を参照してください。

- ［オペレーション名］に設定するオペレーション名を取り込む場合

  ［取込］ボタンをクリックします。表示される［オペレーション名の取込］ダイアログで，取り込むオペレーション名を指定します。［オペレーション名の取込］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.10　オペレーション名の取込ダイアログ(応答アクティビティ)」を参照してください。

- ［ボディ割当変数］に設定する変数の内容を編集する場合

  ［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。この項目には，メッセージ型（XML，non-XML または any）の変数を設定できます。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

  なお，該当するオペレーションを持つユーザ定義受付を定義している場合，ユーザ定義受付の電文フォーマットを変数に設定できます。設定は，［電文フォーマットの取込］ダイアログで行います。［電文フォーマットの取込］ダイアログは，［変数・相関セット一覧］ダイアログの［取込］ボタンをクリックすると表示されます。

  ［電文フォーマットの取込］ダイアログを表示する際，［応答アクティビティ］ダイアログでオペレーション名を設定しておくと，ユーザ定義受付内にある該当するオペレーションの電文フォーマットが選択された状態になります。

  ［電文フォーマットの取込］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5　電文フォーマットの取込ダイアログ」を参照してください。

- ［ヘッダ割当変数］を設定する場合

  ［設定］ボタンをクリックします。表示される［ヘッダ割当変数］ダイアログでヘッダに割り当てる変数を設定します。この項目には，メッセージ型（XML）の変数を設定できます。メッセージ型（non-XML または any）の変数は設定できません。［ヘッダ割当変数］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.2　ヘッダ割当変数ダイアログ」を参照してください。

- ［割当相関セット群］を設定する場合

  ［設定］ボタンをクリックします。表示される［割当相関セット群］ダイアログで割当相関セットを設定します。［割当相関セット群］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.3　割当相関セット群ダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

- 必ずオペレーション名を設定してください。
- フォルト名が設定されると，フォルトを返します。
- 必ず割当変数を指定してください。
- 同一のオペレーション名を持つ応答アクティビティで，同じ割当変数名を設定してください。
- 同一オペレーションの応答アクティビティのフォルト名に対応する割当変数を 1 つに設定してください。
- 同期の受付アクティビティを設定した場合，その受付アクティビティのオペレーションに対応する応答アクティビティを設定してください。
- 受付アクティビティとそのオペレーションに対応する応答アクティビティの間には，別の応答アクティビティを設定しないでください。
- フォルト処理の割り当てに設定する割当変数には，メッセージ型（XML）の変数を設定してください。メッセージ型（non-XML または any）の変数は設定できません。
- ビジネスプロセスをバージョンアップしたあとに応答アクティビティの定義情報を変更する場合は注意が必要です。詳細は，「5.9.4(3)　ビジネスプロセスのバージョンアップ時の注意事項」を参照してください。

5.6.4　サービス呼出アクティビティ

定義済みの HCSC コンポーネントへの要求電文の送信について定義するアクティビティです。

サービス呼出アクティビティは，［サービス呼出アクティビティ］ダイアログで詳細を定義します。

(1)　定義手順

定義の手順を次に示します。

1. **サービス呼出アクティビティをキャンバスに配置します。**
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. **次のどちらかの方法で［サービス呼出アクティビティ］ダイアログを表示させます。**
   - キャンバスのサービス呼出アクティビティをダブルクリックする
   - キャンバスのサービス呼出アクティビティを選択して右クリックし，［設定］を選択する

   ［サービス呼出アクティビティ］ダイアログが表示されます。

3. **［サービス呼出アクティビティ］ダイアログに必要な情報を入力します。**
   ［サービス呼出アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.11　サービス呼出アクティビティダイアログ」を参照してください。
   - ［要求電文用ボディ割当変数］および［応答電文用ボディ割当変数］に設定する変数の内容を編集する場合

［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。この項目には，メッセージ型（XML，non-XML または any）の変数を設定できます。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

なお，呼び出すサービス部品で定義されている電文フォーマットを変数に設定する場合，［電文フォーマットの取込］ダイアログで設定できます。［電文フォーマットの取込］ダイアログを利用して呼び出すサービス部品で定義されている電文フォーマットを変数に設定する手順については，この手順の次に記載されている「ポイント」を参照してください。

［電文フォーマットの取込］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.5　電文フォーマットの取込ダイアログ」を参照してください。

- ［要求電文用ヘッダ割当変数］および［応答電文用ヘッダ割当変数］に変数を設定する場合

  ［設定］ボタンをクリックします。表示される［ヘッダ割当変数］ダイアログで割当変数を設定します。この項目には，メッセージ型（XML）の変数を設定できます。メッセージ型（non-XML または any）の変数は設定できません。［ヘッダ割当変数］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.2　ヘッダ割当変数ダイアログ」を参照してください。

- ［割当相関セット群］を設定する場合

  ［設定］ボタンをクリックします。表示される［割当相関セット群］ダイアログで割当相関セットを設定します。［割当相関セット群］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.3　割当相関セット群ダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

ポイント

呼び出すサービス部品で定義されている電文フォーマットを変数に設定する方法を次に示します。なお，要求電文の場合で説明していますが，応答電文についても同様の手順です。

1. ［サービス呼出アクティビティ］ダイアログで呼び出すサービス名およびオペレーション名を設定します。
2. ［要求電文用割当変数］の［編集］ボタンをクリックします。

   ［変数・相関セット一覧］ダイアログが表示されます。
3. 変数名を設定します。
4. ［取込］ボタンをクリックします。

   ［電文フォーマットの取込］ダイアログが表示されます。

   ［サービス名］および［オペレーション名］には，手順 2.で編集した値が設定されています。また，［電文種別］に「要求電文」が設定されています。
5. ［電文フォーマット］に電文フォーマット名を設定します。
6. ［OK］ボタンをクリックします。

［電文フォーマットの取込］ダイアログが閉じられ，［変数・相関セット一覧］ダイアログに戻ります。

7. ［追加］ボタンをクリックします。
   定義した変数が［変数一覧］に追加されます。
8. 変数一覧から追加した変数を選択します。
9. ［OK］ボタンをクリックします。
   ［変数・相関セット一覧］ダイアログが閉じられ，［サービス呼出アクティビティ］ダイアログに戻ります。

(2) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- サービス名およびオペレーション名には，定義済みの HCSC コンポーネントとそのオペレーションを設定してください。また，設定したオペレーションの通信モデルと定義されているオペレーションの通信モデルを一致させてください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 呼び出すサービス名を設定してください。
- 同期呼び出しの場合，必ず応答電文用の割当変数を設定してください。
- 永続化しないビジネスプロセスの場合，非同期のサービス呼出アクティビティを設定しないでください。
- ボディ割当変数に指定したメッセージ型変数（XML，non-XML，any）と，呼び出す HCSC コンポーネントのメッセージ型を一致させてください。不一致の場合，ビジネスプロセス定義からビジネスプロセスを呼び出すとき以外は，呼び出す HCSC コンポーネントでエラーになります。

5.6.5 Java 呼出アクティビティ

専用インターフェース（jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM，または jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterface）を実装した Java クラスの呼び出しを定義するアクティビティです。

Java 呼出アクティビティは，［Java 呼出アクティビティ］ダイアログで詳細を定義します。

(1) Java 呼出アクティビティの定義前の準備

Java 呼出アクティビティを定義する前には，次の準備をする必要があります。

HCSCTE プロジェクトの作成

［Java 呼出アクティビティ］ダイアログで指定する Java クラスは，HCSCTE プロジェクトの src ディレクトリに格納されます。そのため，必ず HCSCTE プロジェクトを作成しておく必要があります。

HCSCTE プロジェクトの作成方法については，「3.1　プロジェクトの管理」を参照してください。

ユーザ定義 Java クラスの作成

Java 呼出アクティビティで利用する Java クラスを作成します。利用する Java クラスのインターフェースについては「5.6.5(2)　利用する Java クラスのインターフェース」を参照してください。

注意事項

- Java クラスは，リポジトリ内のすべての HCSC コンポーネントで共有されます。そのため，既存の Java クラスを変更した場合，変更前の内容のクラスと変更後の内容のクラスの HCSC コンポーネントが混在してしまうおそれがあります。Java クラスを変更した場合，変更した Java クラスを呼び出す Java 呼出アクティビティが定義されているビジネスプロセスを，再度パッケージングしてください。

  また，Java 呼出アクティビティを削除しても，利用している Java クラスは削除されません。使用していない不要な Java クラスは，別途削除してください。
- Java クラスの呼び出し時に確保したリソースは，処理が終了しても保持し続けるため，システム全体に高い負荷がかかった場合に，Java ヒープ不足や Metaspace 領域不足などが原因で OutOfMemoryError が発生することがあります。そのため，OutOfMemoryError が発生したときに適切にリソースを解放する処理や，エラー処理（ロールバックなど）を実装しておく必要があります。
- 専用インターフェース（jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM，または jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterface）を実装した Java クラスをコンテナ拡張ライブラリに含めないでください。Java クラスはビジネスプロセスをパッケージングすると，自動的に EAR ファイルに追加されます。

ライブラリの追加

ビジネスプロセスをパッケージングして作成される EAR ファイルにライブラリを追加したい場合，HCSCTE プロジェクトの lib ディレクトリにライブラリをコピーしておいてください。

lib ディレクトリにコピーしたライブラリは，ビジネスプロセスをパッケージングすると，自動的に EAR ファイルに追加されます。

なお，lib ディレクトリには，ディレクトリや次の名前のライブラリを登録しないでください。

- csbdef.jar
- cscbp_ejb.jar
- csbjava.jar

注意事項

- HCSCTE プロジェクト内の src ディレクトリおよび lib ディレクトリへのリンクは削除しないでください。
- ビジネスプロセスをパッケージングする前に，Eclipse 上で Java クラスをビルドしてください。ビルドされていない Java クラスは，パッケージング時にビジネスプロセスに取り込まれません。

(2) 利用する Java クラスのインターフェース

Java 呼出アクティビティで呼び出す Java クラスは，CustomClassInterfaceM インターフェース，または CustomClassInterface インターフェース（CustomClassInterfaceM の互換用インターフェース）を実装している必要があります。

(a) CustomClassInterfaceM インターフェース

```
package jp.co.Hitachi.soft.csc.bp.receiver.ejb;

public interface CustomClassInterfaceM {
    public Object invoke(
        String processName,
        int version,
        String activityName,
        Object... inputData
    ) throws CSBUserException, CSBSystemException;
}
```

Java クラスの内容について次に説明します。

引数

仮引数名	説明
processName	ビジネスプロセス名
version	ビジネスプロセスのバージョン
activityName	アクティビティ名
inputData	[Java 呼出アクティビティ] ダイアログの［引数用割当変数］で指定した変数※

注※

可変長引数のため，複数の値を指定することができます。

戻り値

［Java 呼出アクティビティ］ダイアログの［戻り値用割当変数］に指定した変数に代入されます。

例外

throw される例外が CSBUserException か，CSBSystemException かによって処理が異なります。また，この 2 つの例外以外の例外が発生した場合，処理が中断されます。この場合，プロセスの実行状態を永続化しているときには，ロールバックが実行されます。

CSBUserException が throw された場合

Java 呼出アクティビティにフォルト処理が設定されている場合，CSBUserException が throw されると，フォルトコネクションでフォルト処理として定義されているアクティビティが実行されます。

Java 呼出アクティビティにフォルト処理が設定されていない場合，invokeJavaFault が発生したものとして処理が進められます。

CSBUserException のインターフェースを次に示します。

```
package jp.co.Hitachi.soft.csc.bp;

public class CSBUserException extends Exception {
   private String errorCode;
   private String detail;
   public CSBUserException() { super(); }
   public CSBUserException(String aMessage) { super(aMessage); }
   public CSBUserException(String aErrorCode, String aDetail) {
       super();
       this.errorCode = aErrorCode;
       this.detail = aDetail; }
   public CSBUserException(String aMessage, String aErrorCode, String aDetail) {
       super(aMessage);
       this.errorCode = aErrorCode;
       this.detail = aDetail; }
   public String getErrorCode(){ return this.errorCode; }
   public String getDetail(){ return this.detail; }
}
```

CSBSystemException が throw された場合

CSBSystemException が throw されると，システム例外が発生したものとして処理が中断されます。このとき，プロセスの実行状態を永続化している場合には，ロールバックが実行されます。

CSBSystemException のインターフェースを次に示します。

```
package jp.co.Hitachi.soft.csc.bp;

public class CSBSystemException extends Exception {
    public CSBSystemException() { super(); }
    public CSBSystemException(String message) { super(message); }
    public CSBSystemException(String message, Throwable cause) {
        super(message, cause);
    }
    public CSBSystemException(Throwable cause) { super(cause); }
}
```

(b) CustomClassInterface インターフェース

```
package jp.co.Hitachi.soft.csc.bp.receiver.ejb;
```

```
public interface CustomClassInterface {
    public Object invoke(
        String processName,
        int version,
        String activityName,
        Object inputData
    ) throws CSBUserException, CSBSystemException;
}
```

ポイント

CustomClassInterface は，CustomClassInterfaceM の互換用インターフェースです。
CustomClassInterfaceM を使用することを推奨します。

Java クラスの内容について次に説明します。

引数

仮引数名	説明
processName	ビジネスプロセス名
version	ビジネスプロセスのバージョン
activityName	アクティビティ名
inputData	[Java 呼出アクティビティ] ダイアログの［引数用割当変数］で指定した変数※

注※

変数が複数指定されている場合は，リクエスト処理時にエラーメッセージ（KDEC20093-E）が出力されます。

戻り値

[Java 呼出アクティビティ] ダイアログの［戻り値用割当変数］に指定した変数に代入されます。

例外

throw される例外が CSBUserException か，CSBSystemException かによって処理が異なります。またこの 2 つの例外以外の例外が発生した場合，処理が中断されます。このとき，プロセスの実行状態を永続化している場合には，ロールバックが実行されます。

CSBUserException，または CSBSystemException が throw された場合の処理については，「(a) CustomClassInterfaceM インターフェース」の戻り値の説明を参照してください。

CSBUserException および CSBSystemException 以外の例外が発生した場合，処理が中断されます。このとき，プロセスの実行状態を永続化している場合には，ロールバックが実行されます。

注意事項

- [Java 呼出アクティビティ] ダイアログの［引数用割当変数］および［戻り値用割当変数］に指定する変数の型と，このインターフェースの引数および戻り値の型の対応を次に示します。

［引数用割当変数］および［戻り値用割当変数］の変数型	引数および戻り値の型
boolean	java.lang.Boolean

［引数用割当変数］および［戻り値用割当変数］の変数型	引数および戻り値の型
numeric	java.lang.Double
string	java.lang.String
XML	byte[]
non-XML	byte[]
any	byte[]

上記の型以外を指定した場合，エラー（KDEC20030-E）になります。

- 利用する Java クラスのインターフェースは cscbp_ejb.jar に含まれます。そのため，コンパイルするときには，cscbp_ejb.jar をクラスパスに追加する必要があります。Eclipse のビルドパスに cscbp_ejb.jar が存在しないときは，<インストールディレクトリ>¥CSC¥lib¥cscbp_ejb.jar をビルドパスに追加してください。
- Java プログラムでは，呼び出しごとにデフォルトコンストラクタでインスタンスが生成されます。そのため，インスタンスにデータを保持することはできません。

(3)　定義手順

Java 呼出アクティビティの定義の手順を次に示します。

1. Java 呼出アクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 次のどちらかの方法で［Java 呼出アクティビティ］ダイアログを表示させます。

   - キャンバスの Java 呼出アクティビティをダブルクリックする（［Java エディタ起動］メニューが非活性状態の場合だけ）
   - キャンバスの Java 呼出アクティビティを選択して右クリックし，［設定］を選択する

   ［Java 呼出アクティビティ］ダイアログが表示されます。

3. ［Java 呼出アクティビティ］ダイアログに必要な情報を入力します。

   ［Java 呼出アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.12　Java 呼出アクティビティダイアログ」を参照してください。

   ［引数用割当変数］および［戻り値用割当変数］に設定する変数の内容を編集する場合，［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

4. ［Java 呼出アクティビティ］ダイアログの［OK］ボタンをクリックします。

   ［Java 呼出アクティビティ］ダイアログが閉じられます。

5. キャンバスの Java 呼出アクティビティをダブルクリックするか，または右クリックして［Java エディタ起動］を選択します。

   Eclipse の Java エディタが起動されます。

   Java エディタには，［Java 呼出アクティビティ］ダイアログで指定したクラスのソースコードが表示されます。［Java 呼出アクティビティ］ダイアログで指定したクラスが初めて編集するクラスの場合，Java エディタを起動すると jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM を implements したソースコードのテンプレートが表示されます。

6. Java 呼出アクティビティから呼び出される Java クラスのソースコードを，Java エディタで編集します。

7. 編集した Java クラスのソースコードをコンパイルします。

8. 編集したソースコードおよびコンパイルしたクラスファイルを保存して，Java エディタを終了します。

(4)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 必ず Java クラス名を設定してください。

(5)　Java 呼び出しでエラーが発生した場合の処理

アクティビティでフォルトが発生した場合に何らかの処理を実行するとき，フォルト処理用のアクティビティを配置してフォルト処理を実行します。

フォルト処理は，フォルトが発生するアクティビティと，フォルト処理を実行するアクティビティを連結して定義します。連結にはフォルトコネクションを利用します。

(a)　フォルト処理を実行する場合の定義

フォルト処理の定義手順を次に示します。

1. フォルトが発生するアクティビティとフォルト処理を実行するアクティビティをキャンバスに配置し，詳細を定義します。

   フォルト処理を実行するアクティビティは複数定義できます。

   アクティビティの配置方法については，「5.4.1　アクティビティを配置する」を参照してください。各アクティビティの定義方法の詳細については，「5.6　アクティビティの定義」を参照してください。

2. フォルトコネクションを利用してフォルトが発生するアクティビティとフォルト処理を実行するアクティビティを連結します。

アクティビティの連結方法については，「5.4.2(2)　連結の設定方法」を参照してください。また，フォルトコネクションを利用できるアクティビティについては，「5.4.2(5)　アクティビティの連結条件」を参照してください。

3. 連結線をダブルクリックします。

［フォルト処理の割当］ダイアログが表示されます。

［フォルト処理の割当］ダイアログの入力・表示内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.6　フォルト処理の割当ダイアログ」を参照してください。なお，フォルト処理に割り当てられる変数はメッセージ型（XML）だけです。メッセージ型（non-XML および any）は割り当てられません。

4. フォルト処理の条件を定義します。

フォルトの対象となる変数と，フォルトが発生した場合に処理する連絡先のアクティビティを，それぞれ［割当変数］および［遷移先］ドロップダウンリストで指定します。

定義していないフォルトが発生した場合のフォルト処理を設定する場合，［割当変数］ドロップダウンリストで「catch-all」を選択します。

5. ［OK］ボタンをクリックします。

(b)　フォルト処理の定義で使用するフォルト電文を定義するスキーマファイル

フォルト電文を定義するスキーマファイルの格納先を次に示します。

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥fault¥cscinvokejavafault.xsd

フォルト電文を定義するスキーマファイルを次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified"
            targetNamespace="http://www.msg.csc.soft.Hitachi.co.jp/cscBpJava"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="fault">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="exception-name"       type="xsd:string"/>
        <xsd:element name="exception-message"    type="xsd:string"/>
        <xsd:element name="exception-error-code" type="xsd:string"/>
        <xsd:element name="exception-detail"     type="xsd:string"/>
        <xsd:element name="scope-name"           type="xsd:string"/>
        <xsd:element name="activity-name"        type="xsd:string"/>
        <xsd:element name="activity-type"        type="xsd:string"/>
        <xsd:element name="process-instance-id"  type="xsd:string"/>
        <xsd:element name="root-ap-info"         type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

設定できる要素を次に示します。

exception-name

例外名（完全修飾クラス名）を設定します。

exception-message※

例外情報（CSBUserException.getMessage()で取得した情報）を設定します。

exception-error-code※

CSBUserException の errorCode に設定した情報（ユーザが任意に設定できる情報）を設定します。

exception-detail※

CSBUserException の detail に設定した情報（ユーザが任意に設定できる情報）を設定します。

scope-name

Java 呼出アクティビティが所属するスコープ名を設定します。

activity-name

フォルトが発生したアクティビティ名を設定します。

activity-type

フォルトが発生したアクティビティの型名を設定します。

アクティビティの型名には，java を設定してください。

process-instance-id

プロセスインスタンス識別子を設定します。

root-ap-info

ルートアプリケーション情報を設定します。

注※

CSBUserException に要素と対応する情報が設定されていない場合，空要素（空タグ）が設定されます。

(6) 処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

(7) Java 呼出アクティビティから利用できる API

外部プロパティ取込機能では，Java クラスから利用できる API を提供します。外部プロパティ取込機能を使用することで，外部プロパティファイルから API またはカスタムファンクションを利用して，指定する条件に合致するプロパティを取得できます。

(a) getProperties

filePath で指定した，外部プロパティファイルを必要に応じて読み込み，key で指定する文字列にマッピングされている GroupID に分類されたすべてのプロパティを保持する Map を返します。

Map 型返り値の各エントリの key はプロパティパス，value はプロパティ値を保持しています。

一度読み込んだ外部プロパティファイルは，処理の高速化のため次回以降の再読み込みを抑制するためキャッシングします。

API 仕様

API の仕様を次の表に示します。

表 5-14 getProperties の API 仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.api.propertiesutil
2	クラス	CSCCustomGroupedProperties
3	メソッド	getProperties
4	引数 1	filePath 外部プロパティファイルのパス
5	引数 1 の型	java.lang.String
6	引数 2	keyword 取得対象キーワード
7	引数 2 の型	java.lang.String
8	戻り値	keyword にマッピングされたグループに属するプロパティの Map
9	戻り値の型	java.util.Map<java.lang.String, java.lang.String>

記述例を次に示します。

```
public static Map<String,String> getProperties(String filePath, String keyword)
    throws FileNotFoundException,
           IOException,
           SecurityException,
           CSCExtAPIInvalidFileContentsException,
           CSCExtAPIInvalidGroupIDException,
           CSCExtAPIInvalidEscapeSequenceException,
           CSCExtAPIDuplicateMappingStringException,
           CSCExtAPIUnclosedQuotedStringException,
           CSCExtAPIUnknownGroupIDException,
```

```
        CSCExtAPIInvalidPropertyPathException,
        CSCExtAPIIllegalArgumentException,
        CSCExtAPIKeyNotFoundException,
        CSCExtAPIOrphanedStringException;
```

障害対策

getProperties メソッドでは，次の表に示すエラーが発生するおそれがあります。エラーを検出した場合は，次の表に示す例外をスローして処理を中断します。

表 5-15　getProperties の障害および例外クラス

障害内容	例外クラス
指定された外部プロパティファイルが存在しません。	FileNotFoundException
指定された外部プロパティファイルの読み込み中に入出力エラーが発生しました。	IOException
指定された外部プロパティファイルの読み込み中にセキュリティエラーが発生しました。	SecurityException
指定された外部プロパティファイル内に不正な Unicode エスケープシーケンスが存在します。	CSCExtAPIInvalidFileContentsException
GroupID が不正です。	CSCExtAPIInvalidGroupIDException
MappingString または PropertyValue に不正なエスケープシーケンスが存在します。	CSCExtAPIInvalidEscapeSequenceException
グループマッピング部に重複した MappingString が存在します。	CSCExtAPIDuplicateMappingStringException
MappingString または PropertyValue の引用符が閉じられていません。	CSCExtAPIUnclosedQuotedStringException
MappingString または PropertyValue の引用符が閉じたあとに不適切な文字が存在しています。	CSCExtAPIOrphanedStringException
プロパティ部に未定義の GroupID が指定されています。	CSCExtAPIUnknownGroupIDException
PropertyPath 内のパス要素が XML 要素名として不正です。	CSCExtAPIInvalidPropertyPathException
次のどちらかのエラーが発生しています。 • filePath または keyword が null の場合 • filePath が空文字列の場合	CSCExtAPIIllegalArgumentException
keyword に合致する MappingString が存在しません。	CSCExtAPIKeyNotFoundException

注

CSCExtAPI で始まる例外クラスはすべて jp.co.Hitachi.soft.csc.dt.api.propertiesutil.exception パッケージに属し，jp.co.Hitachi.soft.csc.dt.api.propertiesutil.exception.CSCExtAPIException 抽象クラスを継承します。

キャッシュ

getProperties メソッド呼び出し時に実行する，外部プロパティファイルの読込処理はキャッシングします。そのため，同じ外部プロパティファイルを使用する 2 回目以降の getProperties メソッドの処理性能は向上します。

キャッシュはJ2EEサーバの起動中，保持されます。そのため，J2EEサーバ起動中にすでに読み込まれた外部プロパティファイルに変更を加えても，内容は反映されません。変更内容を反映するには，J2EEサーバを再起動する必要があります。

キャッシュはAPIおよびカスタムファンクションで共通です。APIを利用して読み込んだ外部プロパティファイルのキャッシュは，カスタムファンクションでも有効です。また，カスタムファンクションで読み込んだ外部プロパティファイルのキャッシュは，APIでも有効です。

配置

Javaクラスから利用できるAPIは，既存のcscdt_api.jarにパッケージングして提供します。

(b) 外部プロパティファイルの設定

APIおよびカスタムファンクションを利用するには，外部プロパティファイルが必要です。

外部プロパティファイルは，ISO8859-1エンコーディングされたjava.util.Properties形式で作成します。定義ファイルの入力形式の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.2　定義ファイルの入力形式」を参照してください。

外部プロパティファイルは，次の部から構成されます。

- グループマッピング部
- プロパティ部

各部の出現順序，出現回数は問いません。グループマッピング部の規則に従った行はグループマッピング部，プロパティ部の規則に従った行はプロパティ部として，APIおよびカスタムファンクションに正しく認識されます。

グループマッピング部

プロパティを分類するグループの定義と，グループにマッピングする文字列の対応を定義します。

グループマッピング部は，次の形式で定義します。

```
GroupID = [MappingString[,][,MappingString…]]
```

GroupID

GroupIDは，英数字およびアンダースコアで構成された任意の文字列です。ただし，先頭文字に数字を指定することはできません。

重複したGroupIDは，java.util.Propertiesの仕様によって除去されるため検出できません。重複したGroupIDを指定した場合の動作はサポートの対象外となります。

MappingString

MappingStringは，コンマ（,）で区切った文字列をGroupIDにマッピングすることを指示します。MappingStringが省略された場合は，空文字列をGroupIDにマッピングすることを指示します。

MappingStringは次に示す規則に従って記述します。

- MappingString は，外部プロパティファイル内で一意になるように指定してください。重複した MappingString を検出すると，API またはカスタムファンクションはエラー終了します。
- MappingString の次の文字は取り除かれます。

  ・先頭のスペース

  ・末尾のスペース

  ・ダブルクォーテーション（"）

  そのため，「abc」，「△abc」，「abc△」，「"abc"」は，すべて「abc」として定義されます。正確に「abc△」を指定したい場合は，「"abc△"」とダブルクォーテーション（"）で囲む必要があります。
- MappingString 内にコンマ（,），円マーク（¥），ダブルクォーテーション（"）を含める場合は，例外を除き，それぞれ「¥¥,」，「¥¥¥¥」，「¥¥"」とエスケープする必要があります。プロパティファイルのエスケープ仕様によって，「¥」は「¥¥」と記述する必要があるため，すべての 1 文字の「¥」を「¥¥」と 2 文字で記述します。

  例えば「"ABC"」を指定する場合は「¥¥"ABC¥¥"」，「Co.,LTD.」を指定する場合は，「Co.¥¥,LTD.」と記述します。
- MappingString 全体を「"」で囲んだ場合はコンマ（,）をエスケープする必要はありません。例えば，「"Co., LTD."」「"Co.¥¥, LTD."」「Co.¥¥, LTD.」はすべて「Co.,LTD.」と解釈されます。
- MappingString 全体を「"」で囲まない場合（先頭文字が「"」でない場合）は，「"」をエスケープする必要はありません。例えば，「A"B」および「A¥¥"B」は「A"B」と解釈されます。また，「AB"」および「AB¥¥"」は「AB"」と解釈されます。

グループマッピング部の記述例を次に示します。

●有効なグループマッピングの記述例

```
Group1 =
Group2 = aaa, bbb
```

Group1 には空文字列を，Group2 には「aaa」と「bbb」をマッピングをすることを指示します。

```
Group1 = aaa,
Group2 = bbb, ccc
```

Group1 には「aaa」と空文字列を，Group2 には「bbb」と「ccc」をマッピングすることを指示します。

●不正なグループマッピングの記述例

```
Group1 = aaa,
Group2 = bbb,
```

Group1 には「aaa」と空文字列を，Group2 には「bbb」と空文字列をマッピングすることを指示していますが，空文字列が重複しているのでエラーとなります。

プロパティ部

プロパティパスとプロパティ値の対を定義します。

プロパティパスの接頭子（ピリオド（.）で区切られた最初の文字列要素）は，グループの分類に使用します。

プロパティパスの接頭子を除いた部分は，プロパティのキーとして利用します。データ変換アクティビティから利用できるカスタムファンクションでは，ピリオド（.）で区切られた各文字列を XML ノードとした階層構造に展開します。

プロパティ部は，次の形式で定義します。

```
GroupID.PropertyPath = PropertyValue
```

GroupID

GroupID は，グループマッピング部で定義した GroupID を指定します。

PropertyPath

PropertyPath は，プロパティのパスを定義します。ピリオド（.）は，セパレータとして使用します。各パス要素は，XML 要素名の規則に従って指定します。ただし，パス要素にピリオド（.）を含める場合は「¥¥.」とエスケープする必要があります。

また，java.util.Properties は，等号（=），コロン（:），および空白文字を，キーと値のセパレータと認識するため，パス要素にコロン（:）を含める場合は，「¥:」とエスケープする必要があります。

PropertyValue

PropertyValue は，GroupID，PropertyPath に対応するプロパティ値を定義します。

PropertyValue は，次の規則に従って記述します。コンマ（,）のエスケープが不要なこと以外は MappingString の記述規則と同じです。

- PropertyValue の次の文字は取り除かれます。

  ・先頭のスペース

  ・末尾のスペース

  ・ダブルクォーテーション（"）

  そのため，「abc」，「△abc」，「abc△」，「"abc"」は，すべて「abc」として定義されます。正確に「abc△」を指定したい場合は，「"abc△"」とダブルクォーテーション（"）で囲む必要があります。

- PropertyValue 内に円マーク（¥），ダブルクォーテーション（"）を含める場合は，それぞれ「¥¥¥」「¥¥"」とエスケープする必要があります。例えば，「"ABC"」を指定する場合は「¥¥"ABC¥¥"」，「C:¥Program Files」を指定する場合は「C:¥¥¥Program Files」と記述する必要があります。

外部プロパティファイルの定義例

外部プロパティファイルの定義例を次に示します。

```
# GROUP MAPPING PART
production = PROD, Production Environment, 本番環境
development = DEV, Development Environment, 開発環境
# PROPERTY PART
production.static_host = https://tp1server.hoge.com
production.static_port = 10100
development.static_host = https://testserver.fuga.com
```

```
development.static_port = 10200
development.elmA.elmB = SomeValue
```

この定義例では，production グループのキーとして「PROD」，「Production Environment」または「本番環境」を定義し，development グループのキーとして「DEV」，「Development Environment」または「開発環境」を定義しています。

また，プロパティ部の GroupID とプロパティの関係は次のようになっています。

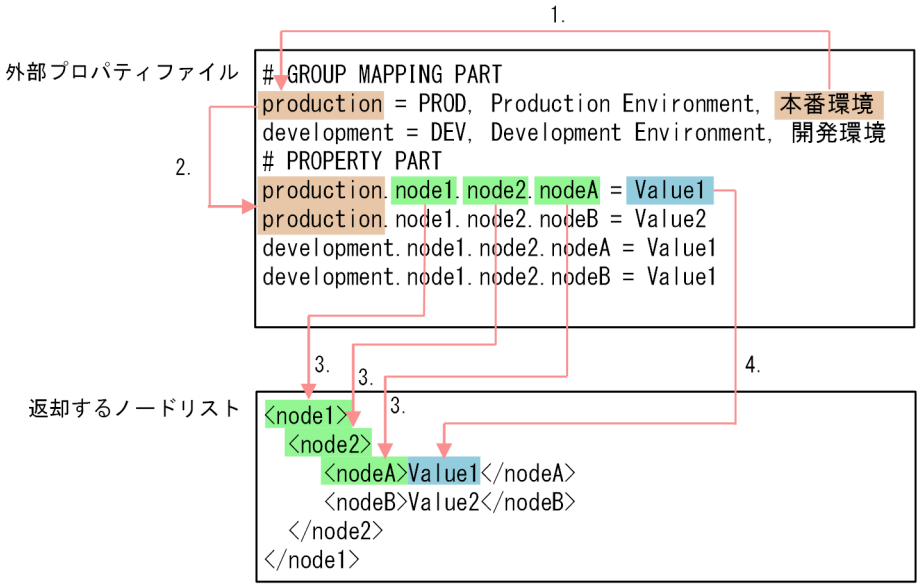
表 5-16　定義例のグループとプロパティの関係

GroupID	PropertyPath	PropertyValue
production	static_host	https://tp1server.hoge.com
	static_port	10100
development	static_host	https://testserver.fuga.com
	static_port	10200
	elmA.elmB	SomeValue

(c)　外部プロパティファイルの動作

データ変換アクティビティから利用できるカスタムファンクションの利用例を基に，外部プロパティファイルがどのように動作するかを次に示します。

図 5-12　外部プロパティファイルの動作



1. API の引数として渡された文字列に合致する MappingString を検索し，対応する GroupID を取得します。
2. 1.で取得した GroupID で始まるプロパティを取得対象として選択します。
3. 2.で選択したプロパティのそれぞれについて，PropertyPath をパス要素に分解し，各要素をノードとした XML 階層を作成します。

4. 3.で作成したノードの値として，PropertyValue を設定します。

5.6.6　データ変換アクティビティ

データ変換処理を定義するアクティビティです。

データ変換アクティビティは，［データ変換アクティビティ］ダイアログで詳細を定義します。

また，データ変換定義画面でデータの変換方法の定義情報（データ変換定義ファイル）を作成します。

(1)　定義手順

定義の手順を次に示します。

1. **データ変換アクティビティをキャンバスに配置します。**
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. **次のどちらかの方法で［データ変換アクティビティ］ダイアログを表示させます。**
   - キャンバスのデータ変換アクティビティをダブルクリックする（［マッピング定義起動］メニューが非活性状態の場合だけ）
   - キャンバスのデータ変換アクティビティを選択して右クリックし，［設定］を選択する

   ［データ変換アクティビティ］ダイアログが表示されます。

3. **［データ変換アクティビティ］ダイアログに必要な情報を入力します。**
   ［データ変換アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.13　データ変換アクティビティダイアログ」を参照してください。
   ［変換元変数］および［変換先変数］に設定する変数の内容を編集する場合，［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。［変換元変数］および［変換先変数］には，メッセージ型（any）の変数は設定できません。メッセージ型の変数を設定する場合は，メッセージ型（XML または non-XML）を設定してください。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

4. **［データ変換アクティビティ］ダイアログの［OK］ボタンをクリックします。**
   ［データ変換アクティビティ］ダイアログが閉じます。

5. **キャンバスのデータ変換アクティビティをダブルクリックするか，または右クリックして［マッピング定義起動］を選択します。**
   データ変換定義画面が表示されます。

なお，電文フォーマットを変更した場合は，電文フォーマットの変更を反映するかを確認するダイアログが表示されます。詳細は，「7.3.2　電文フォーマットを変更した場合の定義手順」を参照してください。

6. データ変換定義画面でデータ変換定義ファイルを作成します。

データ変換定義ファイルの作成方法の詳細については，「7.3　データ変換定義」を参照してください。

データ変換定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

なお，作成されたデータ変換定義ファイルは，ビジネスプロセスを保存するまで一時ディレクトリに保存されます。ビジネスプロセスを保存したときにリポジトリに格納されます。

注意事項

［データ変換アクティビティ］ダイアログの［データ変換定義］に指定したデータ変換定義ファイルがリポジトリに格納されたあとは，［変換元変数］および［変換先変数］に設定されている変数の情報のうち，変数の部分指定に関する情報および電文フォーマットだけ変更できます。変数の部分指定に関する情報および電文フォーマット以外を変更したい場合，［ファイルを削除］ボタンをクリックしてください。

なお，この場合に［ファイルを削除］をクリックしても，データ変換定義ファイルはリポジトリから削除されません。データ変換定義ファイルは，ビジネスプロセスを保存したときにリポジトリから削除されます。

(2)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 必ず変換先変数を設定してください。
- 必ず変換元変数を設定してください。
- 必ずデータ変換定義を設定してください。

(3)　アクティビティ内でシステム例外が発生した場合の処理

アクティビティ内で発生したシステム例外を，汎用フォルトに変換して送出できます。詳細は，マニュアル「サービスプラットフォーム 解説」の「4.7　システム例外をフォルトに変換する汎用フォルト」を参照してください。

(4) データ変換の処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert プロパティおよび syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

5.6.7　代入アクティビティ

次に示す処理を実行するための定義をするアクティビティです。

- 変数（基本型・メッセージ型）をほかの変数に代入する
- 変数の一部をほかの変数に代入する
- 一般式を作成し，変数に代入する
  一般式の評価結果は，文字列型として取得（java.lang.String へマッピング）されます。
- 固定値をほかの変数に代入する

代入アクティビティは，［代入アクティビティ］ダイアログで詳細を定義します。

なお，変数に値を代入する際に，変数の型が異なっていても，自動的に型を変換して代入できる場合があります。ただし，固定値を代入するときは，自動的な型の変換は行われません。変換の可否と変換規則を次の表に示します。

表 5-17　代入値の変換規則

代入する値のデータ型	変換後のデータ型					
	boolean	numeric	string	メッセージ		
				XML	non-XML	any
boolean	−	○※1	○※2	×	×	×

代入する値のデータ型		変換後のデータ型					
		boolean	numeric	string	メッセージ		
					XML	non-XML	any
numeric		○※3	－	○※4	×	×	×
string		○※5	△※6	－	×	×	×
メッセージ	XML	×	×	×	－	×	○
	non-XML	×	×	×	×	－	○
	any	×	×	×	○	○	－

（凡例）

○：変換して代入できます。

△：場合によって変換して代入できます。

×：変換して代入できません。

－：変換の必要はありません。

注※1

true の場合 1，false の場合 0 が代入されます。

注※2

true または false が代入されます。

注※3

0 の場合は false，それ以外の場合は true が代入されます。

注※4

java.lang.Double クラスの toString(double)メソッドで得られる値が代入されます。

注※5

java.lang.Boolean クラスの parseBoolean(String)メソッドで得られる値が代入されます。

注※6

java.lang.Double クラスの parseDouble(String)メソッドで得られる値が代入されます。

代入アクティビティでは，コピー元とコピー先に指定できない組み合わせがあります。コピー元とコピー先の組み合わせを次の表に示します。

表 5-18　コピー元とコピー先の組み合わせ

コピー元		コピー先							
		変数（メッセージ型）			変数（基本型）	変数の部分（メッセージ型）※		変数の部分（基本型）※	
		XML	non-XML	any		XML	non-XML	XML	non-XML
変数（メッセージ型）	XML	○	×	○	×	○	×	×	×
	non-XML	×	○	○	×	×	×	×	×

コピー元		コピー先							
		変数（メッセージ型）			変数（基本型）	変数の部分（メッセージ型）※		変数の部分（基本型）※	
		XML	non-XML	any		XML	non-XML	XML	non-XML
変数（メッセージ型）	any	○	○	○	×	×	×	×	×
変数（基本型）		×	×	×	○	×	×	○	×
変数の部分（メッセージ型）	XML	○	×	×	×	○	×	×	×
	non-XML	×	×	×	×	×	×	×	×
変数の部分（基本型）	XML	×	×	×	○	×	×	○	×
	non-XML	×	×	×	○	×	×	○	×
式		×	×	×	○	×	×	○	×
固定値（テキスト）		○	○	○	○	○	×	○	×
固定値（非テキスト）		×	○	○	×	×	×	×	×

（凡例）

○：指定できます。

×：指定できません。

注※

コピー先の変数が初期化されていない場合は，コピーできません。

(1)　定義手順

定義の手順を次に示します。

1. 代入アクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 次のどちらかの方法で［代入アクティビティ］ダイアログを表示させます。

   - キャンバスの代入アクティビティをダブルクリックする
   - キャンバスの代入アクティビティを選択して右クリックし，［設定］を選択する

   ［代入アクティビティ］ダイアログが表示されます。

3. ［代入アクティビティ］ダイアログに必要な情報を入力します。

   ［代入アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.14　代入アクティビティダイアログ」を参照してください。

［コピー元］および［コピー先］のデータを追加または編集する場合は，［追加］ボタンまたは［編集］ボタンをクリックしてください。表示されるダイアログ（［代入アクティビティ］サブダイアログ）で，［コピー元］および［コピー先］に設定するデータを追加，編集できます。また，設定する値が変数で，変数の内容を編集する場合は，［代入アクティビティ］サブダイアログの［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。

［代入アクティビティ］サブダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.4.15　代入アクティビティサブダイアログ」を参照してください。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1 変数・相関セット一覧ダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 1 つ以上の代入操作を設定してください。
- 必ずコピー元を設定してください。
- 必ずコピー先を設定してください。
- コピー元のタイプとして，変数と式のどちらかを設定してください。
- コピー元の変数が選択されている場合，必ず変数名を設定してください。
- コピー元の式が選択されている場合，必ず式を設定してください。
- コピー先とコピー元の変数の型が異なる場合，暗黙的に変換されたり，実行時にエラーが発生したりする場合があります。
- 変数（メッセージ型）から変数（メッセージ型）への代入を定義する場合，それぞれの変数の定義で設定する電文フォーマットには，同じ XML スキーマまたは同じバイナリフォーマットで，かつ同じファイル名の定義ファイルを指定してください。

(3)　アクティビティ内でシステム例外が発生した場合の処理

アクティビティ内で発生したシステム例外を，汎用フォルトに変換して送出できます。詳細は，マニュアル「サービスプラットフォーム 解説」の「4.7　システム例外をフォルトに変換する汎用フォルト」を参照してください。

(4)　代入処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert または syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

5.6.8　無操作アクティビティ

実行しても何の動作も行わないことを定義するアクティビティです。無操作アクティビティは，分岐アクティビティを使用して分岐した処理や，フォルトコネクションを使用したフォルト処理などの中で，何も処理しないことを示すために使用します。

無操作アクティビティは，［無操作アクティビティ］ダイアログで詳細を定義します。

(1)　定義手順

定義の手順を次に示します。

1. 無操作アクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 次のどちらかの方法で［無操作アクティビティ］ダイアログを表示させます。

   - キャンバスの無操作アクティビティをダブルクリックする
   - キャンバスの無操作アクティビティを選択して右クリックし，［設定］を選択する

   ［無操作アクティビティ］ダイアログが表示されます。

3. ［無操作アクティビティ］ダイアログに必要な情報を入力します。

   ［無操作アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.16　無操作アクティビティダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

5.6.9 フォルト送出アクティビティ

ビジネスプロセス内でフォルトを定義，送出して，上位のスコープアクティビティにフォルトを通知するためのアクティビティです。

フォルト送出アクティビティは，［フォルト送出アクティビティ］ダイアログで詳細を定義します。

(1) 定義手順

定義の手順を次に示します。

1. フォルト送出アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1 アクティビティを配置する」を参照してください。

2. 次のどちらかの方法で［フォルト送出アクティビティ］ダイアログを表示させます。
   - キャンバスのフォルト送出アクティビティをダブルクリックする
   - キャンバスのフォルト送出アクティビティを選択して右クリックし，［設定］を選択する

   ［フォルト送出アクティビティ］ダイアログが表示されます。

3. ［フォルト送出アクティビティ］ダイアログに必要な情報を入力します。
   ［フォルト送出アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.17 フォルト送出アクティビティダイアログ」を参照してください。
   ［割当変数］は，次の 2 つの設定方法があります。運用に合わせて，どちらかの方法で割当変数を設定してください。
   - フォルト送出アクティビティに割り当てられた変数を，上位のスコープまたはビジネスプロセスにフォルトとして送出させたい場合
     ［割当変数］に，フォルトを送出する際に割り当てる変数を設定します。［割当変数］には，メッセージ型（XML）の変数を設定できます。メッセージ型（non-XML または any）は設定できません。
     ［割当変数］に設定する変数の内容を編集する場合，［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1 変数・相関セット一覧ダイアログ」を参照してください。

なお，この設定の場合，ビジネスプロセス内の任意の位置にフォルト送出アクティビティを定義できます。

- すでに発生しているフォルトを，そのまま上位のスコープまたはビジネスプロセスに送出させたい場合［割当変数］に変数を設定しないでください。この設定の場合，フォルト送出アクティビティは，どれかのアクティビティのフォルトハンドラ内に定義する必要があります。

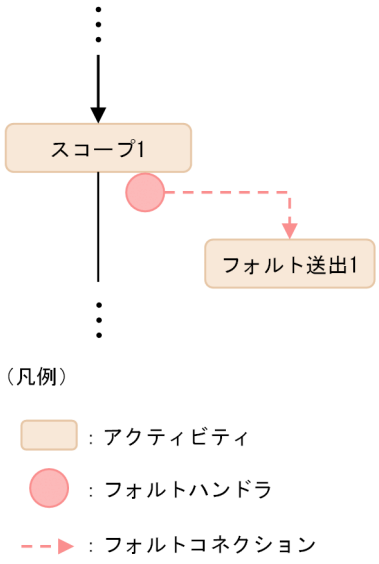
4. ［OK］ボタンをクリックします。

(2) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- フォルト送出アクティビティを含む上位のスコープに，対応する割当変数または catchAll を利用したフォルト処理を定義してください。
- フォルト処理の割り当てに設定する割当変数には，変数を設定しないか，メッセージ型（XML）の変数を設定してください。メッセージ型（non-XML または any）の変数は設定できません。
- フォルト処理の割当変数に変数を設定しない場合，フォルト送出アクティビティは，次のようにどれかのアクティビティのフォルトハンドラ内に定義してください。フォルト送出アクティビティがどれかのアクティビティのフォルトハンドラ内に定義されていない場合，動作を保証しません。

  フォルト送出アクティビティの正しい定義位置と，誤った定義位置の例を次に示します。

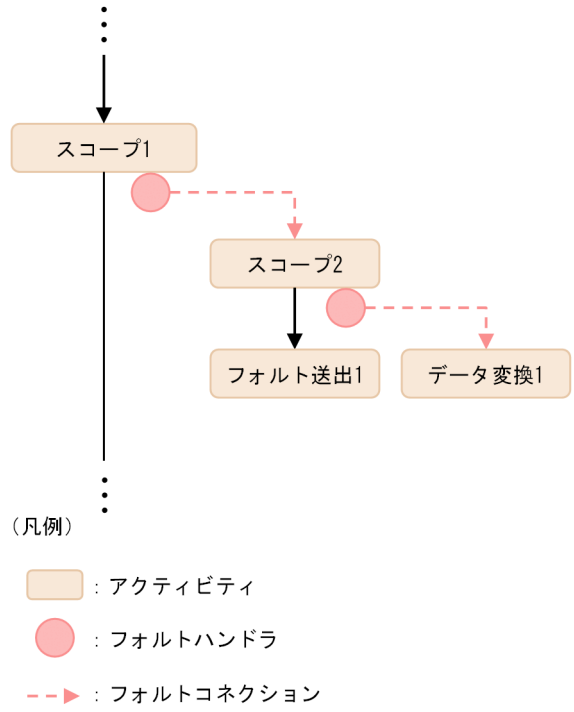
図 5-13 フォルト処理の割当変数に変数を設定しない場合の，フォルト送出アクティビティの正しい定義位置の例



フォルト送出アクティビティ（フォルト送出 1）は，この図のように，スコープ 1 のフォルトハンドラ配下に定義します。

次のようにフォルトハンドラが多段に定義されている場合，上位のフォルトハンドラ配下の位置には，フォルト送出アクティビティを定義できません。

図 5-14　フォルト処理の割当変数に変数を設定しない場合の，フォルト送出アクティビティの誤った定義位置の例



この図のような位置にフォルト送出アクティビティを定義した場合，動作を保証しません。

- フォルト送出アクティビティの割当変数の値が，フォルト処理で指定した割当変数に適合しない場合，フォルト処理の割当変数で指定したアクティビティに遷移しません。この場合，catchAll で指定したアクティビティに遷移します。

(3)　処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

5.6.10　補償アクティビティ

補償処理を実行するアクティビティを定義するアクティビティです。補償アクティビティは，ビジネスプロセス実行中にフォルトが発生した場合に，実行完了していた処理を取り消すときに使用します。

補償アクティビティは，［補償アクティビティ］ダイアログで詳細を定義します。

(1)　定義手順

定義の手順を次に示します。

1. 補償アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。
2. 次のどちらかの方法で［補償アクティビティ］ダイアログを表示させます。
   - キャンバスの補償アクティビティをダブルクリックする
   - キャンバスの補償アクティビティを選択して右クリックし，［設定］を選択する

   ［補償アクティビティ］ダイアログが表示されます。
3. ［補償アクティビティ］ダイアログで補償するアクティビティを設定します。
   ［補償アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.18　補償アクティビティダイアログ」を参照してください。
4. ［OK］ボタンをクリックします。

(2)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 補償アクティビティは，次のハンドラ内で定義します。
  - 補償ハンドラを定義したアクティビティが所属するスコープのフォルトハンドラ
  - 補償ハンドラを定義したアクティビティが所属するスコープの補償ハンドラ
- 次のアクティビティには補償ハンドラを定義できません。
  - フォルトハンドラまたは補償ハンドラ内で最上位のスコープアクティビティ，およびサービス呼出アクティビティ
  - ビジネスプロセスの最上位のスコープアクティビティ，およびサービス呼出アクティビティ
- 補償アクティビティ実行時に，補償ハンドラを起動させるためには，次の条件をすべて満たす必要があります。
  - 補償ハンドラが定義されたアクティビティが正常終了状態である

- 補償ハンドラが一度も起動されていない

- ある補償アクティビティが複数のアクティビティに対して補償処理を実行する場合，補償処理の実行順序は，ビジネスプロセスを永続化するかどうかによって，次のように異なります。
  - ビジネスプロセスを永続化しない場合，補償処理はアクティビティが正常終了した順の逆順に実行されます。
  - ビジネスプロセスを永続化する場合，補償処理の実行順序は不定です。
- ビジネスプロセスが構造化されているかの検証でエラーが発生する状態の場合，補償アクティビティダイアログにある［補償するアクティビティ］のコンボボックスの表示が不正となることがあります。
- BPEL 標準規格では，フォルトハンドラおよび補償ハンドラが定義されていない場合，暗黙的にフォルトハンドラおよび補償ハンドラを作成し，ビジネスプロセス内に定義したすべての補償処理を実行するよう規定されています。スコープアクティビティのデフォルトハンドラの設定を ON にすると，これらの動作を実行できます。スコープアクティビティのデフォルトハンドラの詳細については，「5.4.5 補償処理を定義する」を参照してください。

(3) 補償範囲の定義

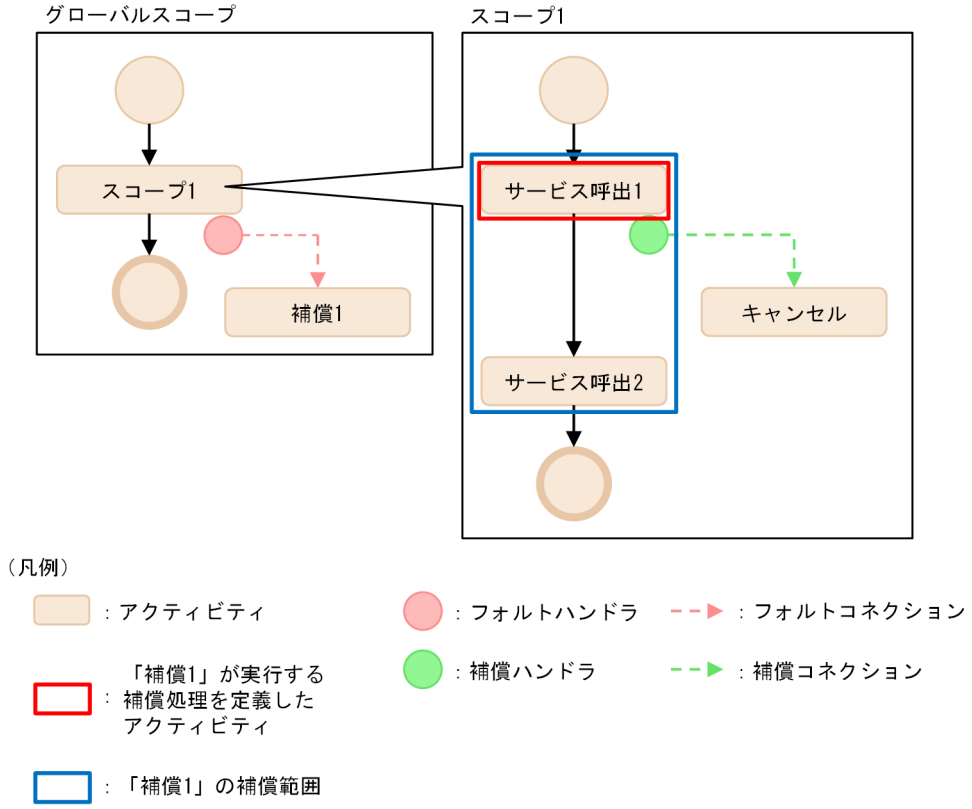
補償範囲とは，補償アクティビティと連結したフォルトハンドラまたは補償ハンドラを設定している，スコープ直下の領域のことです。ただし，フォルトハンドラ内，および補償ハンドラ内は，補償範囲から除外されます。

補償範囲にどこまでのアクティビティが含まれるのか，6 つの例を使って説明します。

(a) 基本的なパターン

次の図の場合，「サービス呼出 1」および「サービス呼出 2」が「補償 1」の補償範囲に含まれます。

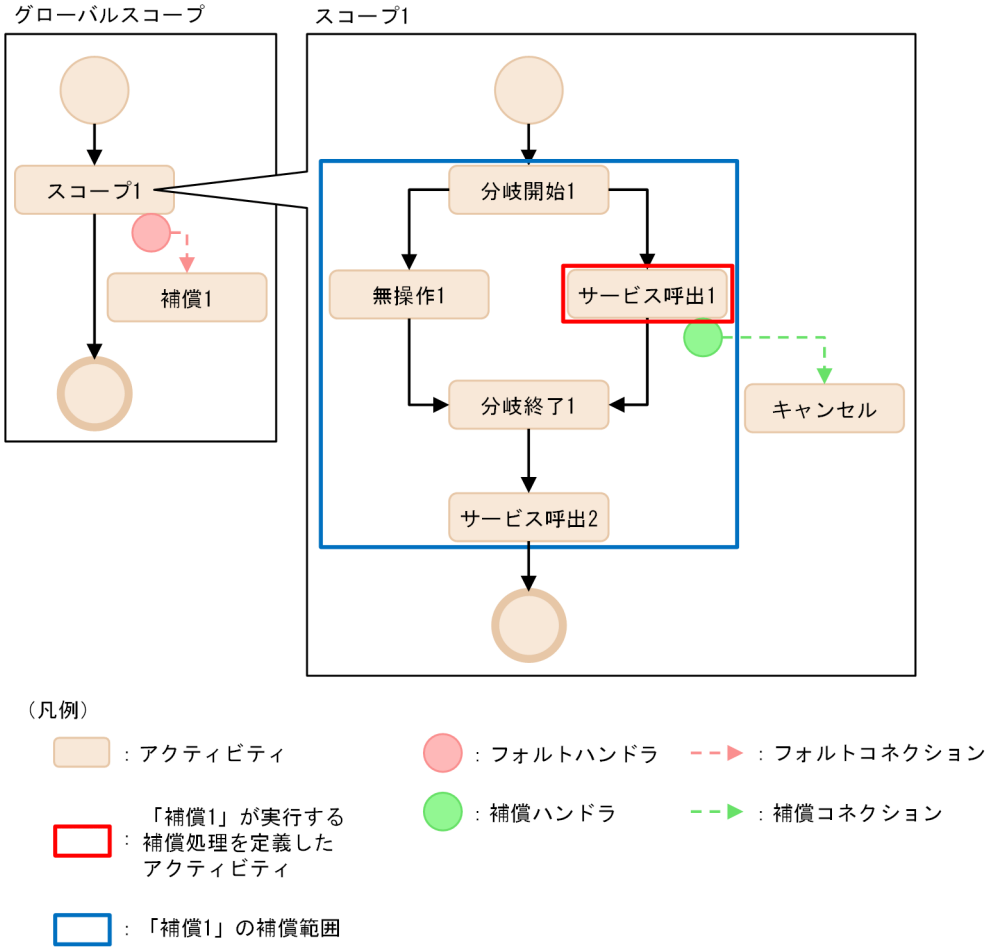
図 5-15　補償範囲：基本的なパターン



(b)　補償ハンドラの定義されたアクティビティが，分岐の中にあるパターン

次の図の場合，「分岐開始 1」，「無操作 1」，「サービス呼出 1」，「分岐終了 1」および「サービス呼出 2」が「補償 1」の補償範囲に含まれます。

図 5-16　補償範囲：補償ハンドラの定義されたアクティビティが，分岐の中にあるパターン

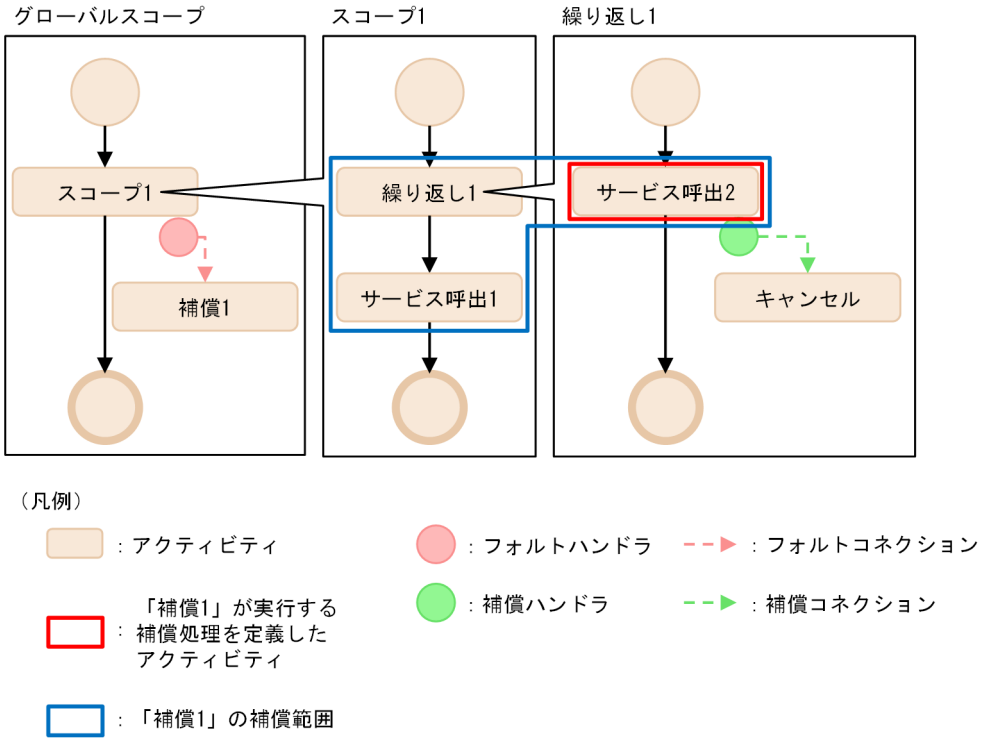


また，この図のうち，「分岐開始 1」「分岐終了 1」が並列アクティビティの場合でも，同様に「分岐開始 1」，「無操作 1」，「サービス呼出 1」，「分岐終了 1」および「サービス呼出 2」が「補償 1」の補償範囲に含まれます。

(c)　補償ハンドラの定義されたアクティビティが，繰り返しの中にあるパターン

次の図の場合，「繰り返し 1」，「サービス呼出 1」および「サービス呼出 2」が「補償 1」の補償範囲に含まれます。

図 5-17　補償範囲：補償ハンドラの定義されたアクティビティが，繰り返しの中にあるパターン



（凡例）

：アクティビティ　：フォルトハンドラ　：フォルトコネクション

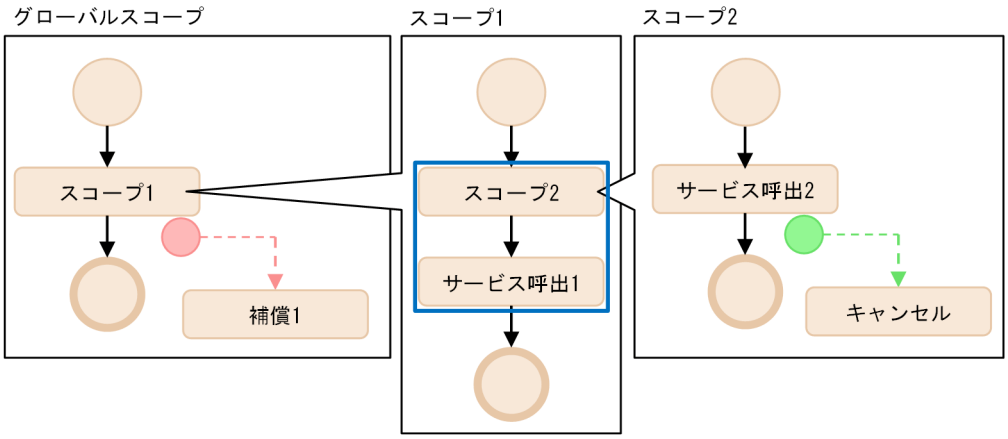
「補償1」が実行する補償処理を定義したアクティビティ　：補償ハンドラ　：補償コネクション

：「補償1」の補償範囲

(d)　補償ハンドラの定義されたアクティビティが，スコープの中にあるパターン

次の図の場合，「補償 1」の補償範囲に含まれるのは「スコープ 2」と「サービス呼出 1」であり，「サービス呼出 2」は「補償 1」の補償範囲に含まれません。「補償 1」の補償範囲は，「スコープ 1」直下の領域であるためです。もし「補償 1」を起点として「サービス呼出 2」の補償ハンドラを起動させたい場合は，「スコープ 2」に補償コネクションと補償アクティビティを定義する必要があります。

図 5-18　補償範囲：補償ハンドラの定義されたアクティビティが，スコープの中にあるパターン



（凡例）

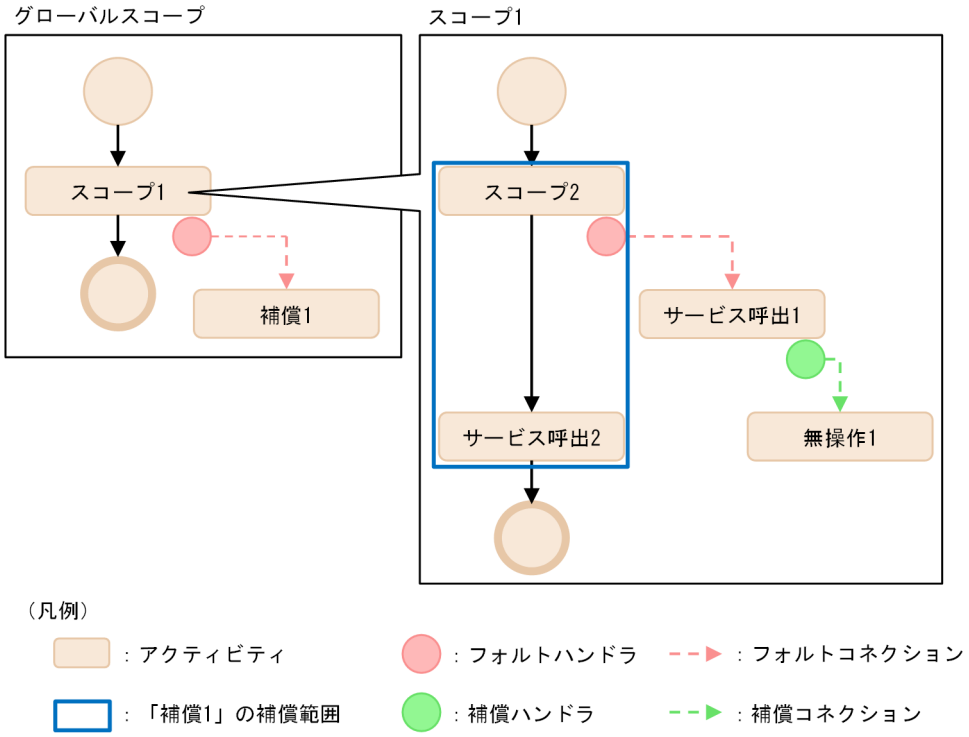
：アクティビティ　：フォルトハンドラ　：フォルトコネクション

：「補償1」の補償範囲　：補償ハンドラ　：補償コネクション

(e) 補償ハンドラの定義されたアクティビティが，フォルトハンドラ内または補償ハンドラ内にあるパターン

次の図の場合，「補償 1」の補償範囲に含まれるのは「スコープ 2」と「サービス呼出 2」であり，「サービス呼出 1」は「補償 1」の補償範囲に含まれません。「サービス呼出 1」は，「スコープ 2」のフォルトハンドラ内に配置されているので，「補償 1」の補償範囲から除外されるためです。

図 5-19 補償範囲：補償ハンドラの定義されたアクティビティが，フォルトハンドラ内または補償ハンドラ内にあるパターン

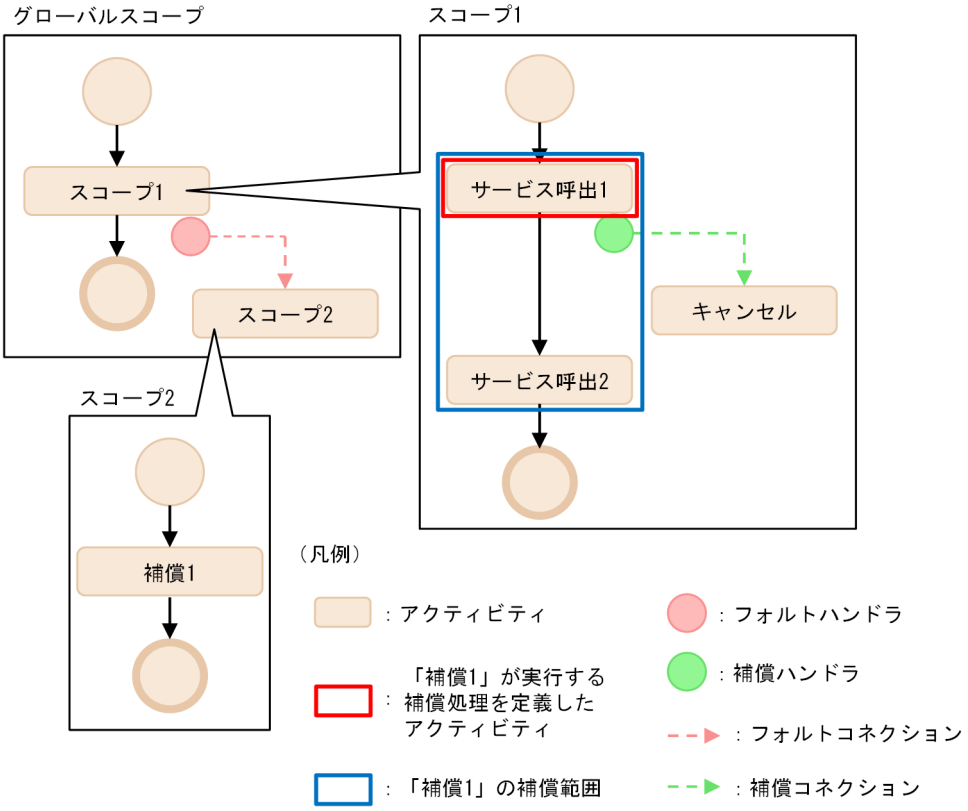


また，この図のうち，「スコープ 2」と「サービス呼出 1」が補償コネクションで連結されている場合でも，同様に「サービス呼出 1」は補償範囲に含まれません。

(f) 補償アクティビティがスコープに囲われているパターン

次の図の場合，「サービス呼出 1」および「サービス呼出 2」が「補償 1」の補償範囲に含まれます。

図 5-20　補償範囲：補償アクティビティがスコープに囲われているパターン



「補償 1」がスコープに囲われている場合でも，「サービス呼出 1」および「サービス呼出 2」は「スコープ 1」の直下に配置されているため，補償範囲に含まれます。なお，「補償 1」が，分岐アクティビティや並列アクティビティ，繰り返しアクティビティなどに囲われている場合でも，同様に「サービス呼出 1」および「サービス呼出 2」は「補償 1」の補償範囲に含まれます。

5.6.11　待機アクティビティ

待機アクティビティは，ビジネスプロセスの処理フローを一定の間隔または期限まで待機する処理を定義するアクティビティです。このアクティビティを使用すると，指定した期間の間，または指定した時刻まで，ビジネスプロセスの処理を待機することができます。

待機アクティビティは，［待機アクティビティ］ダイアログで詳細を定義します。待機する時間の間隔または期限は XPath 式で指定します。

(1)　定義手順

1. 待機アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。
2. 次のどちらかの方法で［待機アクティビティ］ダイアログを表示させます。
   - キャンバスの待機アクティビティをダブルクリックする

- キャンバスの待機アクティビティを選択して右クリックし，［設定］を選択する

［待機アクティビティ］ダイアログが表示されます。

3. ［待機アクティビティ］ダイアログに必要な情報を入力します。

待機時間には「間隔」または「期限」を選択し，待機時間の間隔または期限を XPath 式で指定します。

［待機アクティビティ］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.19　待機アクティビティダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2)　待機時間として指定できる文字列

待機時間に指定する式は XPath 式になります。ビジネスプロセスの処理の中で XPath 式を評価し，その結果で得た文字列を待機時間として使用します。XPath 式を評価して得ることができる文字列は，BPEL 標準規格により一部制限があります。

XPath 式を評価して得ることができる文字列を次に説明します。なお，XPath の指定については，「5.6.21 アクティビティ定義ダイアログの XPath の指定」を参照してください。

(a)　待機時間に「間隔」を指定する場合

待機時間に「間隔」を指定する場合，XPath 式を評価して得ることができる文字列は，XML Schema 型 duration（xsd:duration）に従っている必要があります。XML Schema 型 duration とは，PnYnMnDTnHnMnS 形式の経過時間を表す文字列を示す変数型です。

待機時間に「間隔」を指定する場合に，XPath 式を評価して得ることができる文字列の例を次に示します。

（例 1） 1 年 2 か月 3 日 4 時間 5 分 6 秒間隔を待機時間に指定する場合

XPath 式を評価して得ることができる文字列は，「P1Y2M3DT4H5M6S」になります。

（例 2） 10 秒間隔を待機時間に指定する場合

XPath 式を評価して得ることができる文字列は，「PT10S」になります。

なお，［待機アクティビティ］ダイアログの［式］には，XPath 式を指定する代わりに，「'」または「"」で囲むことで，直接待機する間隔を指定することもできます。「'」または「"」で直接待機する間隔を指定する場合の例を次に示します。

（例 3） 1 年 2 か月 3 日 4 時間 5 分 6 秒間隔を待機時間に指定する場合

［待機アクティビティ］ダイアログの［式］に，「'P1Y2M3DT4H5M6S'」を直接指定します。

（例 4） 10 秒間隔を待機時間に指定する場合

［待機アクティビティ］ダイアログの［式］に，「'PT10S'」を直接指定します。

(b) 待機時間に「期限」を指定する場合

待機時間に「期限」を指定する場合，XPath 式を評価して得ることができる文字列は，XML Schema 型 dateTime（xsd:dateTime）に従っている必要があります。XML Schema 型 dateTime とは，CCYY-MM-DDThh:mm:ss.sssTZ 形式（.sss と TZ は省略できる）の日時を表す文字列を示す変数型です。

待機時間に「期限」を指定する場合に，XPath 式を評価して得ることができる文字列の例を次に示します。

（例 1）2010 年 1 月 1 日 0 時 0 分 0 秒を待機時間に指定する場合

XPath 式を評価して得ることができる文字列は，「2010-01-01T00:00:00」または「2010-01-01T00:00:00.000TZ」になります。

なお，［待機アクティビティ］ダイアログの［式］には，XPath 式を指定する代わりに，「'」または「"」で囲むことで，直接待機する期限を指定することもできます。「'」または「"」で直接待機する期限を指定する場合の例を次に示します。

（例 2）日本時間の 2010 年 1 月 1 日 0 時 0 分 0 秒を待機時間に指定する場合

［待機アクティビティ］ダイアログの［式］に，「'2010-01-01T00:00:00+09:00'」を直接指定します。

（例 3）グリニッジ標準時間（GMT）の 2010 年 1 月 1 日 0 時 0 分 0 秒を待機時間に指定する場合

［待機アクティビティ］ダイアログの［式］に，「'2010-01-01T00:00:00+00:00'」または「'2010-01-01T00:00:00Z'」を直接指定します。

(3) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 必ず待機時間を指定してください。
- 永続化しないビジネスプロセスの場合，待機アクティビティを設定しないでください。
- 待機アクティビティは，応答アクティビティからサービスリクエスタに応答を返したあとに動作させなければならないため，応答アクティビティよりあとに設定してください。応答アクティビティより前に設定した場合，パッケージング時にはエラーになりませんが，ビジネスプロセス実行時にエラーになります。なお，応答アクティビティと，そのあとの受付アクティビティの間に待機アクティビティを設定している場合はエラーになりません。※1※2
- 待機アクティビティの後ろに，受付アクティビティと組になっていない応答アクティビティは設定しないでください。受付アクティビティと組になっていない応答アクティビティを設定した場合，パッケージング時にはエラーになりませんが，ビジネスプロセス実行時にエラーになります。なお，応答アクティビティ以外のアクティビティは設定してもエラーになりません。※1※3

注※1

エラーにならないケースを次に示します。

(例 1)

応答アクティビティよりもあとに，待機アクティビティを設定しています。待機処理以降は，非同期のビジネスプロセスとして実行されるため，エラーになりません。

待機アクティビティで処理を待機し，待機期限の時刻になったら，サービス呼出アクティビティが実行されます。



(例 2)

応答アクティビティよりもあとに，待機アクティビティを設定しています。待機処理以降は，非同期のビジネスプロセスとして実行されるため，エラーになりません。

サービスリクエスタからの要求に応じて，並列処理として片方の無操作アクティビティの処理が実行されます。サービスリクエスタからもう一方の要求を受けて，一度サービスリクエスタに応答を返したあと，待機処理が実行されます。



注※2

エラーになるケースを次に示します。

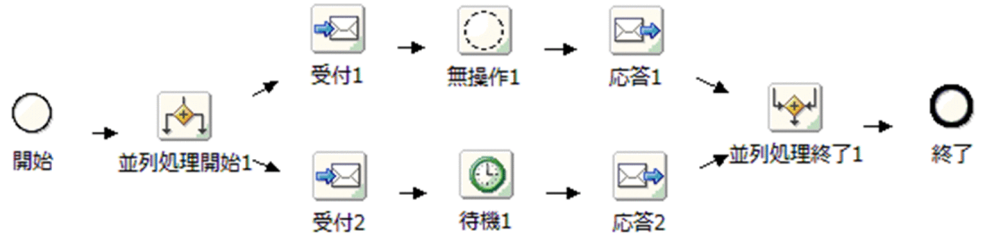
(例 1)

応答アクティビティよりも前に，待機アクティビティを設定しています。同期のビジネスプロセス内で待機アクティビティが実行されるため，ビジネスプロセスを実行するとエラーになります。

●並列処理アクティビティを使用しない場合



●並列処理アクティビティを使用した場合

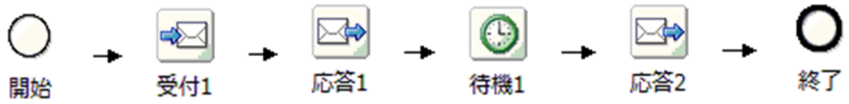


注※3

エラーになるケースを次に示します。

(例 1)

待機アクティビティよりもあとに，受付アクティビティと組になっていない応答アクティビティを設定しています。そのため，ビジネスプロセスを実行するとエラーになります。



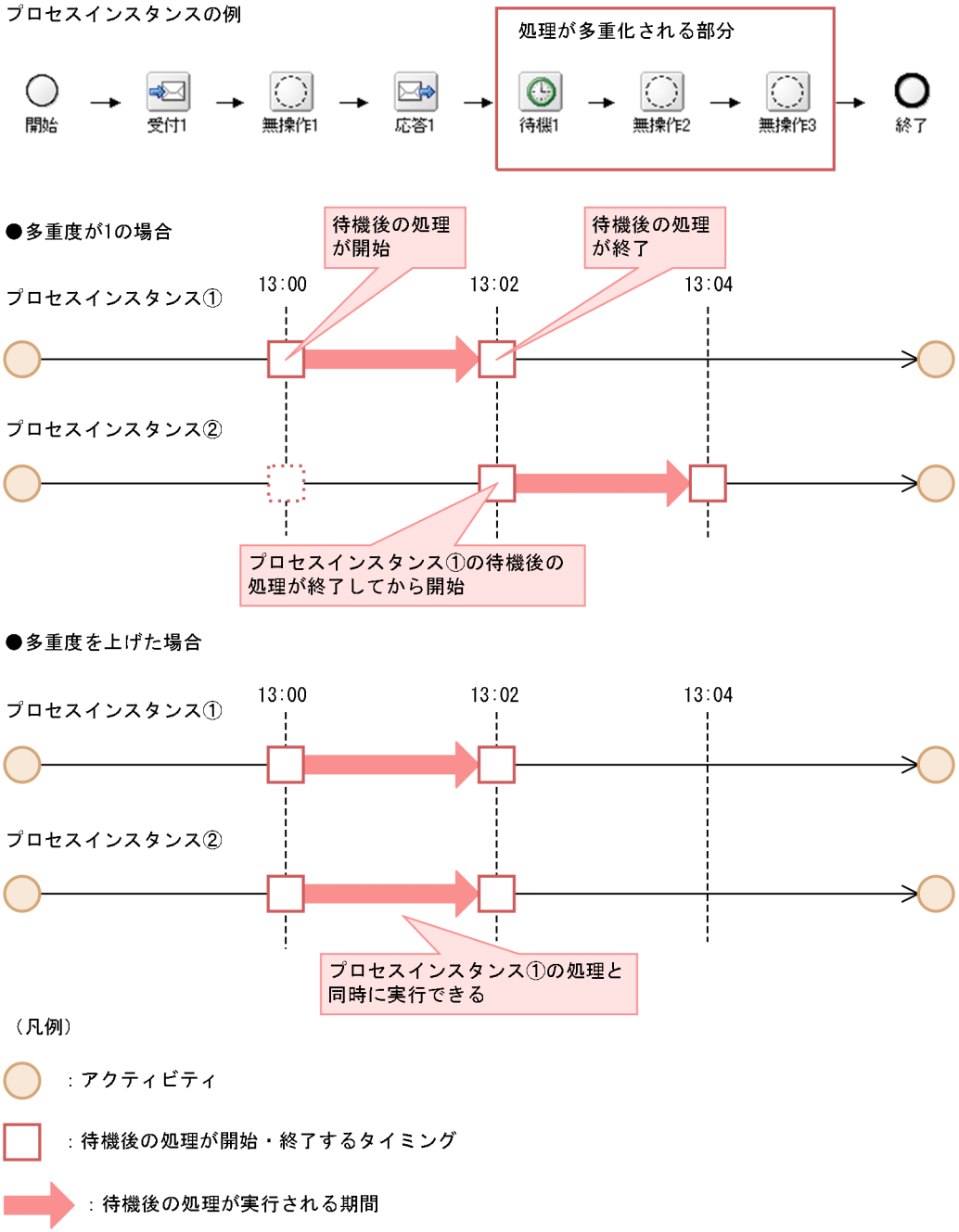
(4) HCSC 簡易セットアップ機能で構築した環境で多重実行する場合の注意事項

HCSC 簡易セットアップ機能で構築した環境では，待機アクティビティ後の処理の多重度がデフォルトで 1 に設定されているため，複数のプロセスインスタンスの待機アクティビティ後の処理は同時に実行されません。複数のプロセスインスタンスから同時に待機アクティビティ後の処理を行うと，あとから処理を開始したプロセスインスタンスの処理は，1 つ前の処理が終了してから実行されます。

複数のプロセスインスタンスの待機アクティビティ後の処理を同時に実行したい場合は，usrconf.properties の ejbserver.ejb.timerservice.maxCallbackThreads プロパティを変更して，処理の多重度を調節してください。

複数のプロセスインスタンスを実行している場合で，待機アクティビティ後の処理の多重度が 1 のときと多重度を上げたときの処理の例を次の図に示します。なお，この例では，複数のプロセスインスタンスの処理が待機アクティビティ（待機 1）で 13：00 まで待機したあと，待機後の処理（無操作 2 および無操作 3）に 2 分掛かるものとします。

図 5-21　待機後の処理の多重度が 1 のときと待機後の処理の多重度を上げたときの処理の例



usrconf.properties の ejbserver.ejb.timerservice.maxCallbackThreads プロパティの設定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3 usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。

5.6.12　検証アクティビティ

ビジネスプロセス内で送受信される電文の妥当性を検証するためのアクティビティです。受付アクティビティやサービス呼出アクティビティによって外部と送受信する電文が，対応する割当変数のスキーマと一致するかどうかを検証するときに使われます。

検証アクティビティは，［検証アクティビティ］ダイアログで詳細を定義します。

(1) 定義手順

定義の手順を次に示します。

1. 検証アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. どちらかの方法で［検証アクティビティ］ダイアログを表示させます。
   - キャンバスの検証アクティビティをダブルクリックする
   - キャンバスの検証アクティビティを選択して右クリックし，［設定］を選択する

   ［検証アクティビティ］ダイアログが表示されます。

3. ［検証アクティビティ］ダイアログに必要な情報を入力します。
   ［検証アクティビティ］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.20　検証アクティビティダイアログ」を参照してください。
   ［変数］に設定する変数の内容を編集する場合，［編集］ボタンをクリックします。表示される［変数・相関セット一覧］ダイアログで，変数の内容を編集できます。［変数］には，メッセージ型（XML）の変数を設定できます。メッセージ型（non-XML または any）の変数は設定できません。［変数・相関セット一覧］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。

4. ［OK］ボタンをクリックします。

(2) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

(3) 処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

(4)　検証でエラーが発生した場合の処理

検証結果が不正だった場合，次の種類のフォルトを送出できます。

- 検証アクティビティ固有のフォルト
- 汎用フォルト

送出するフォルトの選択方法については，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

参考

検証アクティビティに変数が複数設定されていた場合，最初に検証でエラーになった変数の情報でフォルト電文を作成します。

フォルト処理に必要な設定を次に説明します。

(a)　フォルト処理を実行する場合の定義

検証アクティビティでフォルトが発生した場合，検証アクティビティが所属するスコープアクティビティに設定されているフォルト処理の割り当てに従って処理が分岐されます。フォルト処理の割り当てが設定されていない場合は，フォルト例外が発生します。

フォルト処理の割り当て

検証アクティビティから直接フォルト処理を割り当てることはできません。検証アクティビティが所属するスコープアクティビティを使用して処理の割り当てをしてください。

フォルト処理の分岐

検証結果に対応した処理の分岐をさせたい場合は，分岐アクティビティを使用します。

フォルト電文に割り当てる変数から，変数の部分指定でフォルト情報を取得し，取得した情報によって処理が分岐するように定義してください。

任意のフォルトの発生

任意のフォルトを発生させたい場合は，フォルト送出アクティビティを使用します。

フォルトが発生したときに，フォルト送出アクティビティで任意の変数値のフォルトを発生させるように定義してください。

(b)　フォルト処理の定義で使用するフォルト電文を定義するスキーマファイル

検証アクティビティ固有のフォルト電文を定義するスキーマファイルの格納先を次に示します。

<サービスプラットフォームのインストールディレクトリ>¥CSC¥system¥msg¥cscvalidatefault.xsd

検証アクティビティが所属するスコープアクティビティのフォルト処理の割り当てに使用する，検証アクティビティ固有のフォルト電文を定義するスキーマファイルを次に示します。汎用フォルトのスキーマファイルについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.12.1　汎用フォルト電文を定義するスキーマファイル」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 DO NOT EDIT THIS FILE.
 -->
<xs:schema
    elementFormDefault="qualified"
    targetNamespace="http://www.msg.csc.soft.Hitachi.co.jp/cscBpValidate"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="fault">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="exception-name"    type="xs:string"/>
        <xs:element name="exception-message" type="xs:string"/>
        <xs:element name="scope-name"        type="xs:string"/>
        <xs:element name="activity-name"     type="xs:string"/>
        <xs:element name="variable-name"     type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

設定できる要素を次に示します。

exception-name

例外名を設定します。例外名には，Class.getName()の値を設定してください。

exception-message

例外情報を設定します。例外情報には，Exception.toString()の値を設定してください。

scope-name

検証アクティビティが所属するスコープ名を設定します。

activity-name

検証アクティビティの名称を設定します。

variable-name

検証をする変数名を設定します。

5.6.13　ログ出力アクティビティ

ログを出力するアクティビティです。ログの形式はユーザが任意に設定できます。ログ出力アクティビティは，［ログ出力アクティビティ］ダイアログで詳細を定義します。

ログ出力アクティビティは，次のようなビジネスプロセスに関する情報を活用したい場合に使用します。

- ビジネスプロセスでエラーが発生した場合の障害解析
- ビジネスプロセスで処理する電文の情報を統計情報として収集する

そのほか，ログ出力アクティビティに関する設定や，ログ出力アクティビティのログファイルの詳細については，マニュアル「サービスプラットフォーム 解説」の「3.9　ログ出力アクティビティでのログ出力」を参照してください。

(1)　定義手順

定義の手順を次に示します。

1. ログ出力アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。
2. 次のどちらかの方法で［ログ出力アクティビティ］ダイアログを表示させます。
   - キャンバスのログ出力アクティビティをダブルクリックする
   - キャンバスのログ出力アクティビティを選択して右クリックし，［設定］を選択する

   ［ログ出力アクティビティ］ダイアログが表示されます。
3. ［ログ出力アクティビティ］ダイアログに必要な情報を入力します。
   ［ログ出力アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.21　ログ出力アクティビティダイアログ」を参照してください。
4. ［OK］ボタンをクリックします。

(2)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

5.6.14　途中終了アクティビティ

ビジネスプロセスを途中終了するアクティビティです。途中終了アクティビティは，［途中終了アクティビティ］ダイアログで詳細を定義します。

途中終了アクティビティは，次のようにビジネスプロセスを途中終了する場合に使用します。

- フォルトが発生して強制終了させる
- 変数の値に応じてフローを分岐させて，それぞれのフローで終了させる

そのほか，途中終了アクティビティに関する設定や詳細については，マニュアル「サービスプラットフォーム 解説」の「3.4.8　途中終了アクティビティのトランザクション」を参照してください。

(1)　定義手順

定義の手順を次に示します。

1. 途中終了アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。
2. 次のどちらかの方法で［途中終了アクティビティ］ダイアログを表示させます。
   - キャンバスの途中終了アクティビティをダブルクリックする
   - キャンバスの途中終了アクティビティを選択して右クリックし，［設定］を選択する

   ［途中終了アクティビティ］ダイアログが表示されます。
3. ［途中終了アクティビティ］ダイアログに必要な情報を入力します。
   ［途中終了アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.23　途中終了アクティビティダイアログ」を参照してください。
4. ［OK］ボタンをクリックします。

(2)　定義時の注意事項

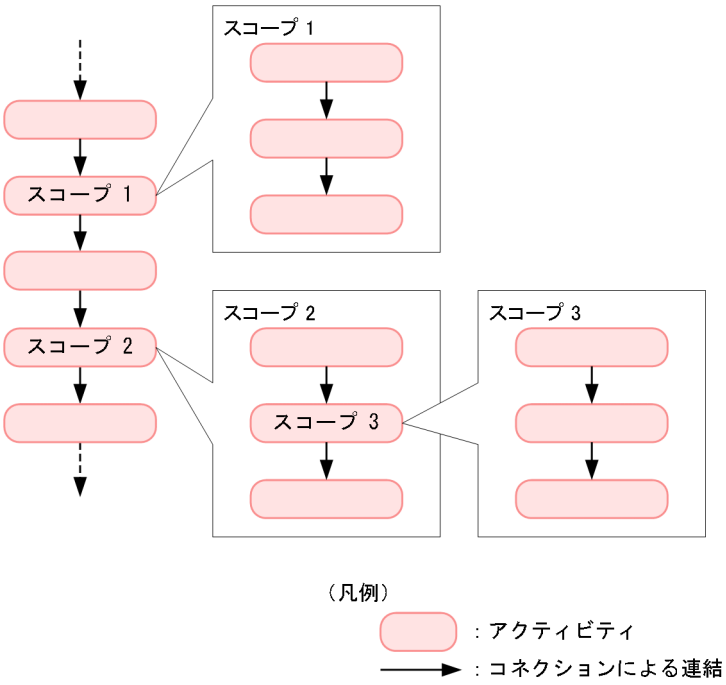
- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

5.6.15　スコープアクティビティ

1 つまたは複数のアクティビティを，ひとまとまりの処理単位として定義するためのアクティビティです。

スコープアクティビティを利用した処理単位について，次の図に示します。

図 5-22　スコープアクティビティを利用した処理単位の設定



スコープアクティビティを定義すると，スコープ内の処理フローを 1 つのまとまりとして扱うことができます。また，上図のスコープ 3 のように，スコープ内の処理フローにさらにスコープを定義することもできます。

なお，スコープ内で定義した変数は，そのスコープ内でだけ有効になります。スコープと変数の有効範囲についての詳細は，「5.5.1(5)(a)　スコープアクティビティを使用している場合の変数の使用」を参照してください。

(1)　定義手順

定義の手順を次に示します。

1. スコープアクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。
2. キャンバスのスコープアクティビティを選択して右クリックし，［設定］を選択します。
   ［スコープアクティビティ］ダイアログが表示されます。
3. アクティビティ名を入力します。
4. ［OK］ボタンをクリックします。
5. 次のどちらかの方法でスコープ内の処理を設定するキャンバスを表示させます。
   - キャンバスのスコープアクティビティをダブルクリックする
   - キャンバスのスコープアクティビティを選択して右クリックし，［開く］を選択する

キャンバスの下部にスコープアクティビティ名のタブが表示され，スコープ内の処理を設定するキャンバスが表示されます。

6. 任意のアクティビティを配置・連結・定義し，スコープ内の処理を設定します。

(2) 高度な設定

スコープアクティビティは，内部に定義されたアクティビティでトランザクションをコミットしないように設定できます。

[スコープアクティビティ］ダイアログの［高度な設定］で，［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択した場合，スコープアクティビティの開始時および終了時にコミットされます。スコープアクティビティ内に定義されたアクティビティはコミットされません。

[アクティビティ単位でコミットする］ラジオボタンを選択した場合，スコープアクティビティの開始時および終了時にコミットされません。スコープアクティビティ内に定義されたコミットを実行するアクティビティはコミットされます。

デフォルトでは，［アクティビティ単位でコミットする］ラジオボタンが選択されています。

[スコープアクティビティ］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.24　スコープアクティビティダイアログ」を参照してください。

[このスコープの開始時と終了時のみコミットする］ラジオボタンを選択した場合のトランザクションの詳細については，マニュアル「サービスプラットフォーム 解説」の「3.4.5　スコープの開始時および終了時にコミットする設定を選択した場合のトランザクション」を参照してください。

(3) 定義時の注意事項

スコープアクティビティ定義時の注意事項を次に示します。

- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

[このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ定義時の注意事項を次に示します。

- スコープアクティビティ内に，次のアクティビティは定義できません。
  - ［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ
- スコープアクティビティ内に，次のアクティビティを定義すると，検証時にエラーが発生します。
  - 受付アクティビティ
  - 応答アクティビティ
  - 待機アクティビティ

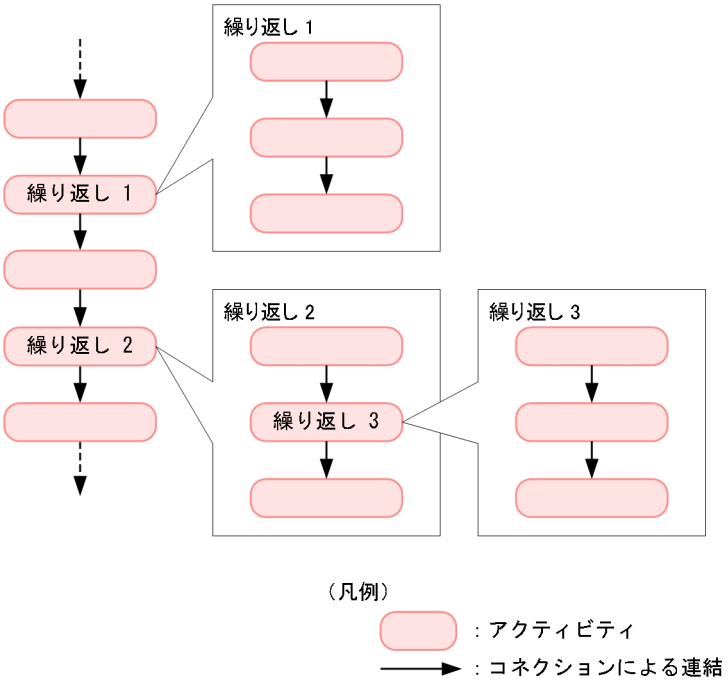
- ［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ内に定義されたサービス呼出アクティビティに指定できるのは，DB アダプタだけです。DB アダプタ以外のサービスアダプタ，およびビジネスプロセスを指定した場合は，動作を保証できません。
- ［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ内で途中終了アクティビティを実行した場合，その時点でコミットされ，ビジネスプロセスを終了します。
- ビジネスプロセスとサービス呼び出し先のトランザクションが別の場合，トランザクション制御の対象はビジネスプロセスのトランザクションとなります。サービス呼び出し先のトランザクションは，トランザクション制御の影響を受けないで動作します（ビジネスプロセスを呼ぶサービス呼出アクティビティ，RequiresNew の機能を呼ぶサービス呼出アクティビティを定義した場合など）。
- スコープアクティビティ内に DB アダプタを呼ぶサービス呼出アクティビティを定義した場合，HCSC サーバランタイム定義ファイルの「dba-separate-transaction」プロパティの値が「false」または，SQL オペレーション定義ファイルの「dba_separate_transaction」属性の値が「N」で動作することが前提となります。
- システム例外が発生し，ロールバックされたビジネスプロセスが再実行された場合，スコープアクティビティ内に定義されたアクティビティ，サービス呼び出し先のサービスが再度実行されることを想定する必要があります。

5.6.16　繰り返しアクティビティ

1 つまたは複数のアクティビティを，指定した条件に従って繰り返すための定義をするアクティビティです。

繰り返しアクティビティを利用した繰り返し処理について，次の図に示します。

図 5-23　繰り返しアクティビティを利用した繰り返し処理



繰り返しアクティビティを定義すると，1 つまたは複数の処理を指定した条件で繰り返すことができます。また，上図の繰り返し 3 のように，繰り返す処理の中に繰り返し処理を定義することもできます。

最初に実行されるアクティビティが繰り返しアクティビティで，繰り返しアクティビティ内に非同期の受付を定義し，そのあとにプロセスが中断されないよう定義した場合の動作は保障しません。

(1) 繰り返し方式

繰り返しアクティビティでは，次のどちらかの繰り返し方式を選択できます。

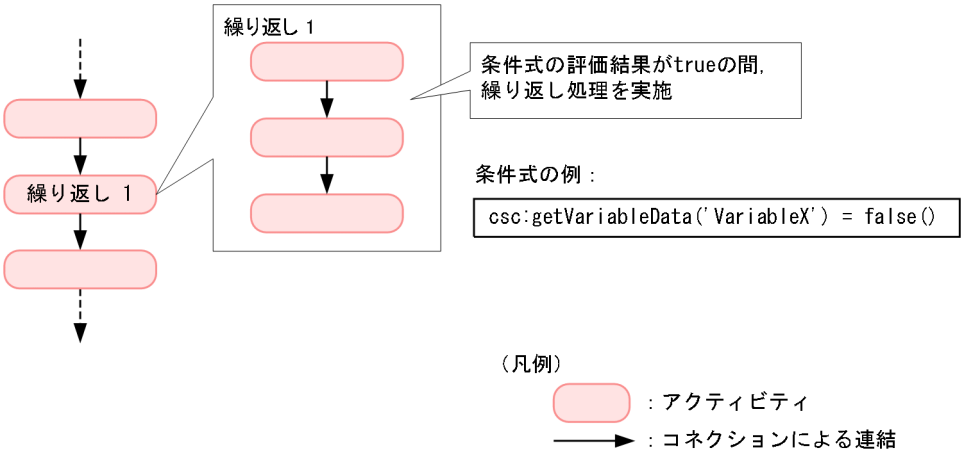
- 一定の条件に従って繰り返す条件指定方式
- 繰り返しリストに含まれる要素の数の分だけ繰り返すリスト指定方式

条件指定方式およびリスト指定方式を利用した繰り返し処理について説明します。

(a) 条件指定方式

条件指定方式を利用した繰り返し処理について，次の図に示します。

図 5-24 条件指定方式を利用した繰り返し処理

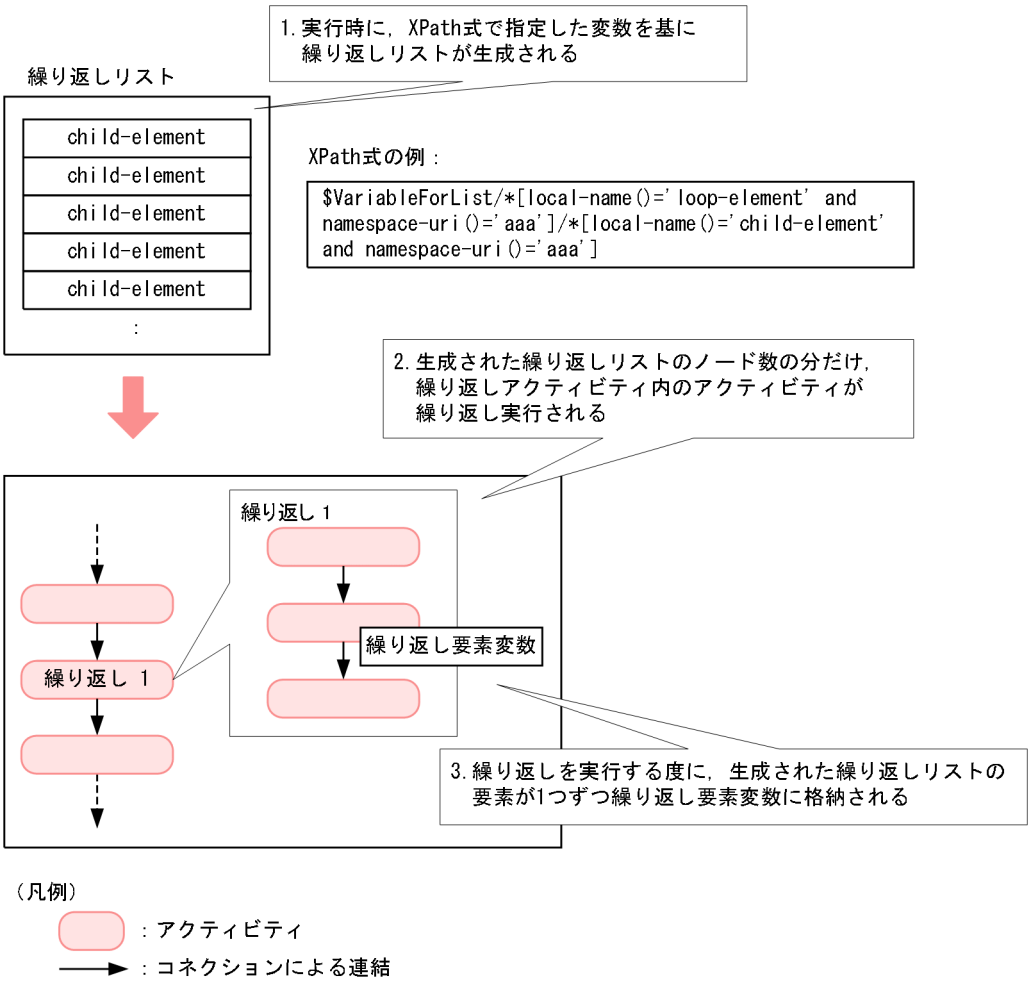


条件指定方式では，XPath 式で繰り返しの条件式を指定します。条件式の評価結果が true の間，繰り返しアクティビティ内のアクティビティが繰り返し実行されます。

(b) リスト指定方式

リスト指定方式を利用した繰り返し処理について，次の図に示します。

図 5-25　リスト指定方式を利用した繰り返し処理



リスト指定方式では，次の流れで繰り返し処理が実施されます。

1. XPath で指定した変数を基に繰り返しリストが生成されます。
2. 生成された繰り返しリストに含まれるノードの数の分だけ，繰り返しアクティビティ内のアクティビティが繰り返し実行されます。
3. 繰り返しを実行するたびに，繰り返しリストの要素が 1 つずつ繰り返し要素変数に格納されます。

繰り返しリストを生成するための変数および繰り返し要素変数は，開発環境で作成して［繰り返しリスト設定］ダイアログで指定します。

(2)　定義手順

定義の手順を次に示します。

(a)　条件指定方式

1. 繰り返しアクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. キャンバスの繰り返しアクティビティを選択して右クリックし，［設定］を選択します。

［繰り返しアクティビティ］ダイアログが表示されます。

3. ［繰り返しアクティビティ］ダイアログで任意のアクティビティ名を入力します。

［繰り返しアクティビティ］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.25　繰り返しアクティビティダイアログ」を参照してください。

4. ［条件指定方式］ラジオボタンを選択して［繰り返し条件設定］ボタンをクリックします。

繰り返しの条件を設定するための［繰り返し条件設定］ダイアログが表示されます。繰り返し条件の条件式は，このダイアログに XPath 式で入力するか，［条件式の作成］ボタンをクリックすると表示される［条件式の作成］ダイアログで設定します。

［繰り返し条件設定］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.27　繰り返し条件設定ダイアログ」を参照してください。

5. 必要な情報を設定して［OK］ボタンをクリックします。

6. 次のどちらかの方法で繰り返す処理を設定するキャンバスを表示させます。

- キャンバスの繰り返しアクティビティをダブルクリックする
- キャンバスの繰り返しアクティビティを選択して右クリックし，［開く］を選択する

キャンバスの下部に繰り返しアクティビティ名のタブが表示され，繰り返す処理を設定するキャンバスが表示されます。

7. 任意のアクティビティを配置・連結・定義し，繰り返す処理を設定します。

(b)　リスト指定方式

1. 繰り返しリストを生成するためのメッセージ型（XML）変数を定義します。

繰り返し要素を子要素に持つスキーマで変数を作成します。子要素はすべて同じ形式にしてください。

変数の定義方法については，「5.5.1(6)(a)　新規に変数を定義する場合」を参照してください。

作成例（子要素の name 属性を同一にし，maxOccurs 属性で要素数を指定する）

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    elementFormDefault="qualified"
    targetNamespace="http://example.com/sample"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="loop-element">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="5" minOccurs="0" name="child-element"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2. 繰り返し要素のメッセージ型（XML）変数を定義します。

手順 1.で作成した変数が生成する繰り返しリストの要素を格納するための変数を作成します。

作成例

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    elementFormDefault="qualified"
    targetNamespace="http://example.com/sample"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="child-element"/>
</xs:schema>
```

3. 繰り返しアクティビティをキャンバスに配置します。

   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

4. キャンバスの繰り返しアクティビティを選択して右クリックし，［設定］を選択します。

   ［繰り返しアクティビティ］ダイアログが表示されます。

5. ［繰り返しアクティビティ］ダイアログで任意のアクティビティ名を入力します。

   ［繰り返しアクティビティ］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.25　繰り返しアクティビティダイアログ」を参照してください。

6. ［リスト指定方式］ラジオボタンを選択して［繰り返しリスト設定］ボタンをクリックします。

7. 繰り返しリストを設定するための［繰り返しリスト設定］ダイアログが表示されます。

   ［繰り返しリスト設定］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.26　繰り返しリスト設定ダイアログ」を参照してください。

8. ［繰り返しリスト設定］ダイアログの［繰り返しリスト］の［式］に，手順 1.で定義した変数の要素のパスを XPath 式で指定します。

   XPath 式の指定方法については，「5.6.21(2)(d)　繰り返しリスト設定ダイアログでの指定」を参照してください。

9. ［繰り返し変数］の［変数名］に，手順 2.で定義した変数を指定します。

10. 必要に応じて［最大繰り返し回数］を設定します。

11. ［OK］ボタンをクリックします。

12. 次のどちらかの方法で繰り返す処理を設定するキャンバスを表示させます。

    - キャンバスの繰り返しアクティビティをダブルクリックする
    - キャンバスの繰り返しアクティビティを選択して右クリックし，［開く］を選択する

    キャンバスの下部に繰り返しアクティビティ名のタブが表示され，繰り返す処理を設定するキャンバスが表示されます。

13. 任意のアクティビティを配置・連結・定義し，繰り返す処理を設定します。

(3) 定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 必ず繰り返し条件を設定してください。

(4) 定義時の注意事項（リスト指定方式の場合だけ）

リスト指定方式を指定する場合の定義時の注意事項を次に示します。

- 繰り返しリストを生成するための変数と繰り返し要素変数は，どちらもメッセージ型（XML）である必要があります。メッセージ型（XML）以外の変数は指定できません。
- 繰り返しリストを生成するための変数は繰り返し処理内で変更できません。変更した場合は動作を保証しません。繰り返し要素変数は，繰り返し処理内で参照および変更できます。
- ［繰り返しリスト設定］ダイアログの［式］に指定した XPath 式に，存在しない要素のパスを指定した場合，繰り返し処理は実行されません。
- 次のような場合，扱う情報の形式が繰り返し要素変数の XML スキーマと合わなくなるおそれがあります。
  - 繰り返し要素変数の XML スキーマに合わない形式の情報を繰り返し要素に設定した場合
  - ［繰り返しリスト設定］ダイアログの［式］に指定した XPath 式に，要素ノード以外の種類のノードを指定した場合

  繰り返し処理中で，上記の情報の形式が合わない繰り返し要素変数を参照しようとした場合，参照できないときはシステム例外が発生し処理が停止します。
- ［繰り返しリスト設定］ダイアログの［式］に指定した XPath 式の評価結果が NodeList 以外の場合，システム例外が発生し処理が停止します。

(5) 処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

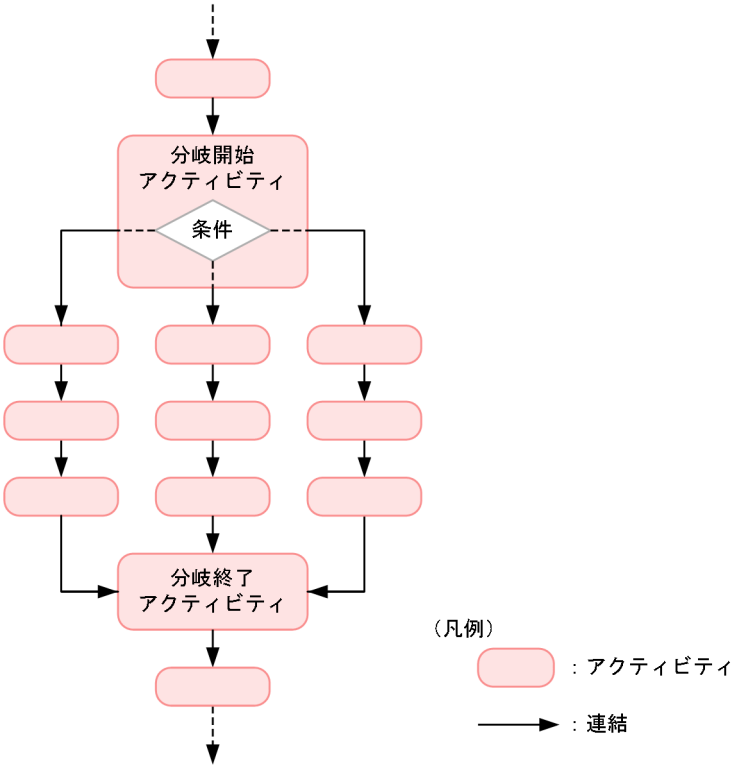
syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

5.6.17　分岐アクティビティ

条件の判定結果によるビジネスプロセスの処理の分岐を定義するアクティビティで，分岐開始アクティビティと分岐終了アクティビティの 2 つがあります。分岐の開始個所には分岐開始アクティビティを，終了個所には分岐終了アクティビティを配置します。

分岐アクティビティを利用した処理の分岐について，次の図に示します。

図 5–26　分岐アクティビティを利用した処理の分岐



分岐開始アクティビティの下には幾つかの処理を用意して，分岐開始アクティビティに設定した条件と条件の判定結果によって分岐させます。分岐の終端には，分岐終了アクティビティを配置します。

分岐開始アクティビティから分岐した処理のうち，フォルト送出アクティビティまたは途中終了アクティビティで終了する処理は，分岐終了アクティビティに連結する必要はありません。また，分岐開始アクティビティから分岐したすべての処理がフォルト送出アクティビティまたは途中終了アクティビティで終了する場合，分岐終了アクティビティを配置する必要はありません。

(1)　分岐開始アクティビティの定義

分岐開始アクティビティの定義の手順を次に示します。

1. 分岐アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 分岐アクティビティの連結先となるアクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

3. 分岐アクティビティと連結先のアクティビティを連結します。
   アクティビティを連結する方法については，「5.4.2(2)　連結の設定方法」を参照してください。

4. 次のどちらかの方法で［分岐アクティビティ］ダイアログを表示させます。
   - キャンバスの分岐アクティビティをダブルクリックする
   - キャンバスの分岐アクティビティを選択して右クリックし，［設定］を選択する

   ［分岐アクティビティ］ダイアログが表示されます。

5. ［分岐アクティビティ］ダイアログに必要な情報を入力します。
   ［分岐アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.28　分岐アクティビティダイアログ」を参照してください。
   分岐条件を設定する場合，［条件設定］ボタンをクリックして表示される［条件設定］ダイアログで，連結先ごとに分岐条件を設定できます。分岐条件は XPath 式で入力，または［条件式の作成］ダイアログで設定します。
   ［条件設定］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.29　条件設定ダイアログ（分岐アクティビティ）」を参照してください。［条件式の作成］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.30　条件式の作成ダイアログ」を参照してください。

6. ［OK］ボタンをクリックします。

(2)　分岐終了アクティビティの定義

分岐終了アクティビティの定義の手順を次に示します。

1. 分岐アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 分岐アクティビティと連結元のアクティビティを連結します。
   アクティビティを連結する方法については，「5.4.2(2)　連結の設定方法」を参照してください。

(3)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。

- 分岐条件を 1 つ以上設定してください。
- 遷移先を条件分岐またはデフォルトに割り当ててください。
- 必ず分岐の条件名を設定してください。
- 必ず分岐の条件を設定してください。
- 並列処理終了アクティビティに接続しないでください。
- 次に示すコネクションは接続しないでください。
  - 開始アクティビティを元とするコネクション※1
  - あるアクティビティのあるフォルト処理の開始点を元とするコネクション※1
  - あるアクティビティのある補償処理の開始点を元とするコネクション※1
- 分岐開始アクティビティは，対応する 1 つの分岐終了アクティビティに接続してください。※2
- 対応する分岐処理を分岐終了アクティビティに接続してください。※2
- 分岐処理は必ず分岐終了アクティビティに接続してください。※2
- 分岐アクティビティと，対応する分岐終了アクティビティの間には，1 つ以上のアクティビティを設定してください。

注※1

アクティビティを元とするコネクション，あるアクティビティのあるフォルト処理の開始点を元とするコネクション，およびあるアクティビティのある補償処理の開始点を元とするコネクションについては，「図 5-37　コネクションの例」を参照してください。

注※2

すべての分岐処理の終端がフォルト送出アクティビティまたは途中終了アクティビティの場合を除きます。

(4)　アクティビティ内でシステム例外が発生した場合の処理

アクティビティ内で発生したシステム例外を，汎用フォルトに変換して送出できます。詳細は，マニュアル「サービスプラットフォーム 解説」の「4.7　システム例外をフォルトに変換する汎用フォルト」を参照してください。

(5)　分岐処理中に例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert プロパティまたは syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert プロパティおよび syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

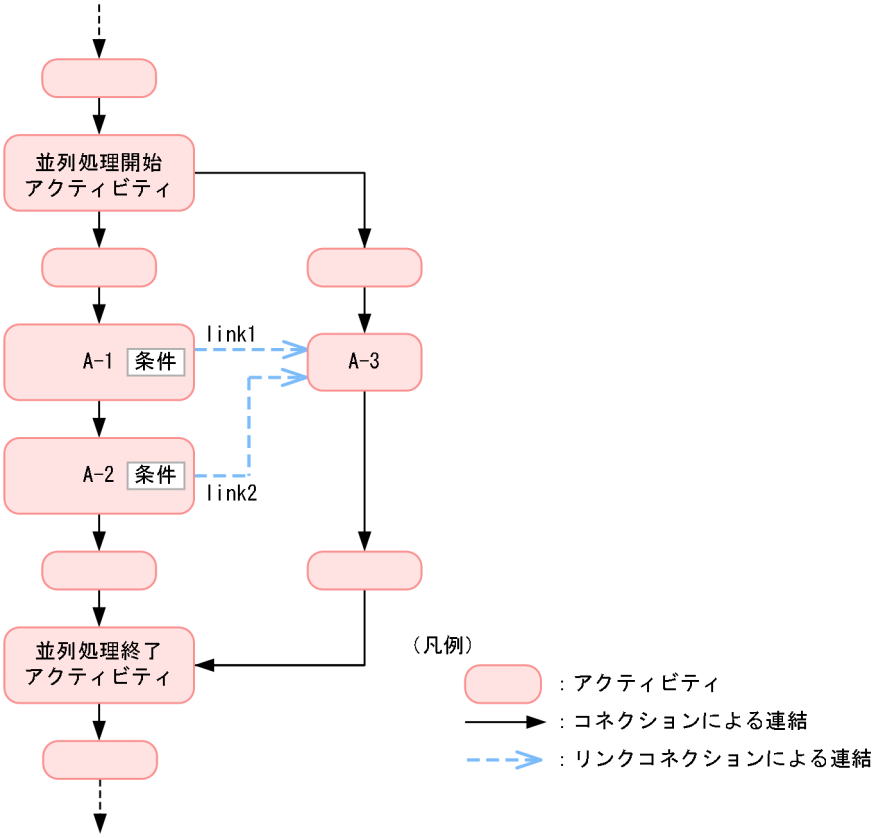
5.6.18　並列処理アクティビティ

ビジネスプロセスの処理フローを 2 つ以上のシーケンスに枝分かれさせて並列実行するための定義をするアクティビティで，並列処理開始アクティビティと並列処理終了アクティビティの 2 つがあります。並列処理の開始個所には並列処理開始アクティビティを，複数のシーケンスの合流個所には並列処理終了アクティビティを配置します。

並列処理開始アクティビティから枝分かれさせて並列実行するシーケンスのうち，フォルト送出アクティビティまたは途中終了アクティビティで終了するシーケンスは，並列処理終了アクティビティに連結する必要はありません。また，並列処理開始アクティビティから枝分かれさせて並列実行するすべてのシーケンスがフォルト送出アクティビティまたは途中終了アクティビティで終了する場合，並列処理終了アクティビティを配置する必要はありません。

並列処理アクティビティを利用した処理の並列実行について，次の図に示します。

図 5-27　並列処理アクティビティを利用した処理の並列実行



並列処理開始アクティビティの下には複数のシーケンスを用意して，並列して処理を実行できます。なお，複数のシーケンス間での処理の順序を設定することもできます。

例えば，A-1 および A-2 が完了してから A-3 を実行する場合，A-1 および A-2 を連結元，A-3 を連結先として，リンクコネクション（link1 および link2）で連結します。さらに，A-1 および A-2 の処理の終了に加えて，リンクの条件を設定して A-1 および A-2 と A-3 のリンクを制御することもできます。

この場合，連結先の A-3 の処理は，次のような過程で開始されます。

1. A-1 および A-2 の処理が完了します
2. A-1 および A-2 に設定されているリンクの条件が判定されます。

   A-1 および A-2 のリンクの条件が判定されると，それぞれリンクコネクション link1 および link2 が活性化します。リンクコネクションは，条件の判定結果によって真または偽に活性化します。なお，条件が設定されていない場合，リンクコネクションは，リンク元のアクティビティの処理が完了すると真に活性化します。
3. A-3 に連結されているすべてのリンクコネクションが活性化すると，A-3 の処理が開始されます。

   ただし，すべてのリンクコネクションが偽に活性化している場合，A-3 の処理は開始されないで省略されます。A-3 の処理が開始されるのは，リンクコネクションのうち，少なくとも 1 つのリンクコネクションが真に活性化している場合です。

並列処理終了アクティビティ以降のアクティビティの処理は，並列処理された複数のシーケンスがすべて完了したあとに実行されます。

(1)　並列処理アクティビティの定義

並列処理アクティビティの定義の手順を次に示します。

1. 並列処理開始アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

2. 並列処理開始アクティビティから枝分かれするシーケンスの各アクティビティを配置・定義します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を，アクティビティを定義する方法は，「5.6　アクティビティの定義」の各アクティビティの説明を参照してください。

3. 並列処理終了アクティビティをキャンバスに配置します。
   アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

4. 並列処理開始アクティビティから並列処理終了アクティビティまでの各アクティビティを連結します。
   アクティビティを連結する方法については，「5.4.2(2)　連結の設定方法」を参照してください。

5. 次のどちらかの方法で［並列処理アクティビティ］ダイアログを表示させます。
   - キャンバスの並列処理開始アクティビティをダブルクリックする
   - キャンバスの並列処理開始アクティビティを選択して右クリックし，［設定］を選択する

   ［並列処理アクティビティ］ダイアログが表示されます。

6. ［並列処理アクティビティ］ダイアログに必要な情報を入力します。
   ［並列処理アクティビティ］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.31　並列処理アクティビティダイアログ」を参照してください。複数シーケンス間のリンクの設定方法の詳細については，「5.6.18(2)　シーケンス間リンクの定義」を参照してください。

7. ［OK］ボタンをクリックします。

(2)　シーケンス間リンクの定義

並列に処理する複数のシーケンス間でのリンクの定義手順について次に示します。

1. 次のどちらかの方法で［並列処理アクティビティ］ダイアログを表示させます。
   - キャンバスの並列処理開始アクティビティをダブルクリックする
   - キャンバスの並列処理開始アクティビティを選択して右クリックし，［設定］を選択する

   ［並列処理アクティビティ］ダイアログが表示されます。

2. 並列処理内で設定するリンクの数だけ，リンク名を入力します。

3. ［OK］ボタンをクリックし，［並列処理アクティビティ］ダイアログを閉じます。

4. リンクさせるアクティビティを連結します。

アクティビティを配置する方法については，「5.4.1　アクティビティを配置する」を参照してください。

5. 次のどちらかの方法で［リンク］ダイアログを表示させます。

- 手順 4.で設定した連結線をダブルクリックする
- 手順 4.で設定した連結線を選択して右クリックし，［設定］を選択する

［リンク］ダイアログが表示されます。

6. ［リンク］ダイアログに必要な情報を入力します。

［リンク］ダイアログの表示・入力内容の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.32　リンクダイアログ」を参照してください。

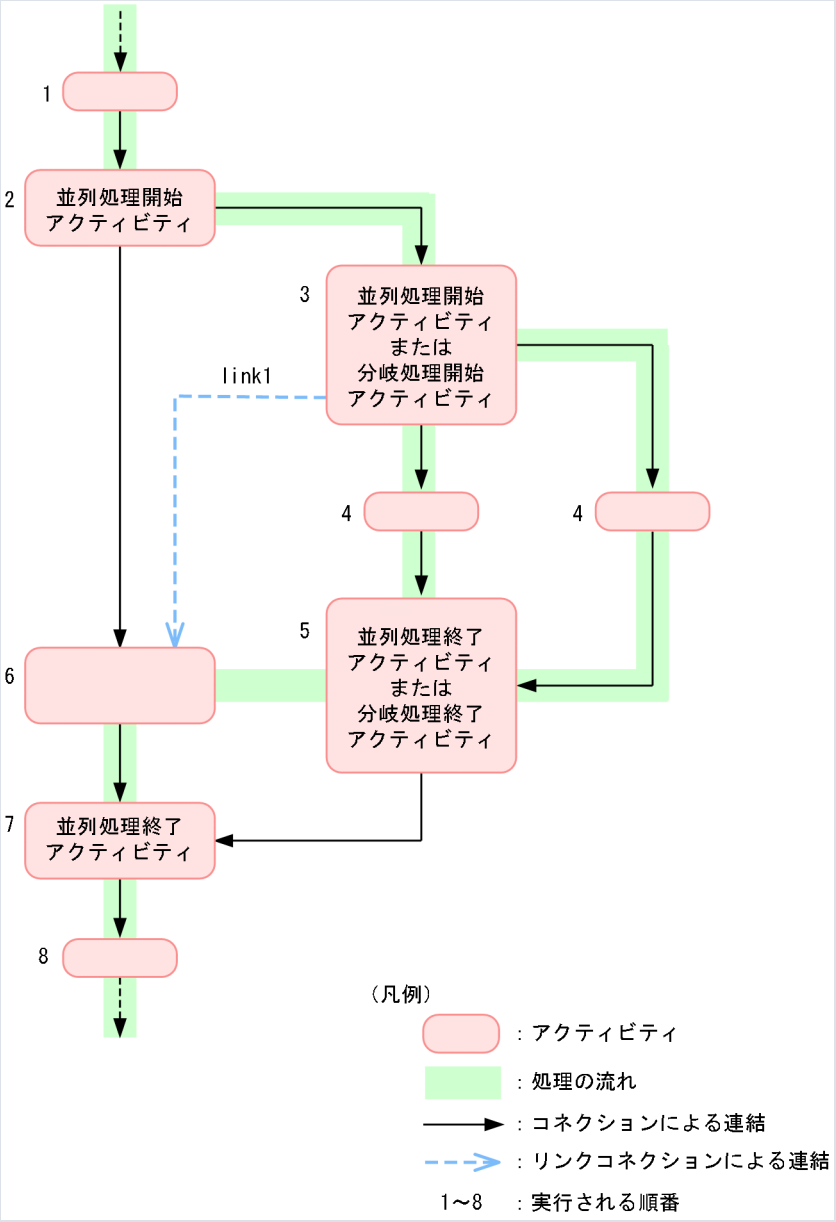
リンクの条件を設定する場合，［遷移条件設定］ボタンをクリックします。表示される［条件設定］ダイアログで，リンクの条件を設定できます。リンクの条件は XPath 式で入力します。［条件設定］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.4.33　条件設定ダイアログ（並列処理アクティビティ）」を参照してください。

7. ［OK］ボタンをクリックします。

注意事項

並列処理開始アクティビティまたは分岐処理開始アクティビティからリンクが定義された場合，処理の流れは次の図のようになります。

図 5-28　並列処理開始アクティビティまたは分岐処理開始アクティビティからリンクが定義された場合の処理の流れ



(3)　定義時の注意事項

- アクティビティ名はビジネスプロセス（スコープ内も含む）内で一意になるように指定してください。
- アクティビティ名は 64 バイト以内にしてください。
- ダイアログの入力フィールドに制御文字は指定しないでください。
- 設定するリンク名の長さは 64 バイト以内にしてください。
- 分岐終了アクティビティに接続しないでください。
- 次に示すコネクションは接続しないでください。

- 開始アクティビティを元とするコネクション※1
- あるアクティビティのあるフォルト処理の開始点を元とするコネクション※1
- あるアクティビティのある補償処理の開始点を元とするコネクション※1

- 並列処理開始アクティビティは，対応する 1 つの並列処理終了アクティビティに接続してください。※2
- 対応する並列処理を並列処理終了アクティビティに接続してください。※2
- 並列処理は必ず並列処理終了アクティビティに接続してください。※2
- 並列処理開始アクティビティと，対応する並列処理終了アクティビティの間には，1 つ以上のアクティビティを設定してください。
- リンクコネクションに有効なリンク名を設定してください。

注※1

アクティビティを元とするコネクション，あるアクティビティのあるフォルト処理の開始点を元とするコネクション，およびあるアクティビティのある補償処理の開始点を元とするコネクションについては，「図 5-37　コネクションの例」を参照してください。

注※2

並列処理するシーケンスの終端がフォルト送出アクティビティまたは途中終了アクティビティの場合を除きます。

(4)　リンクの条件の判定で例外が発生した場合の処理

例外が発生した場合の処理は，HCSC サーバランタイム定義ファイルの syserr-to-fault-convert-all-activity プロパティの設定によって次のように異なります。

syserr-to-fault-convert-all-activity プロパティの指定が ON のとき

メッセージを出力したあと，フォルト電文に設定する項目を含むメッセージを出力し，フォルトを送出します。

送出するフォルトについては，マニュアル「サービスプラットフォーム 解説」の「4.7.1　システム例外をフォルトに変換する汎用フォルトの概要」を参照してください。

syserr-to-fault-convert-all-activity プロパティの指定が OFF のとき

システム例外を throw します。

syserr-to-fault-convert-all-activity プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

5.6.19　終了アクティビティ

ビジネスプロセスの終了を表します。キャンバス上に 1 つだけ配置されます。このアクティビティは削除できません。

なお，終了アクティビティには，定義する内容はありません。

5.6.20　順次処理アクティビティ

順次処理される一連のアクティビティの情報を保持するためのアクティビティです。順序処理アクティビティは，必要に応じて自動で生成されます。順序処理アクティビティは，次に示すアクティビティの情報を保持します。

- 開始アクティビティと終了アクティビティの間に定義された一連のアクティビティ（スコープおよび繰り返しのキャンバスに定義されたアクティビティも同様）
- 分岐開始アクティビティと分岐終了アクティビティの間に定義された各分岐での一連のアクティビティ
- 並列処理開始アクティビティと並列処理終了アクティビティの間に定義された各分岐での一連のアクティビティ
- フォルトコネクションからつながるフォルト処理を行う一連のアクティビティ
- 補償コネクションからつながる補償処理を行う一連のアクティビティ

5.6.21　アクティビティ定義ダイアログの XPath の指定

一部のアクティビティのダイアログでは，入力欄に XPath を指定して変数の情報を取得できます。アクティビティの定義と XPath を指定するダイアログの入力欄を次の表に示します。

表 5–19　アクティビティの定義と XPath を指定するダイアログの入力欄

項番	アクティビティ名	ダイアログの入力欄
1	代入アクティビティ	［代入アクティビティ］サブダイアログの［コピー元］の［値］
2	待機アクティビティ	［待機アクティビティ］ダイアログの［待機時間］の［式］
3	ログ出力アクティビティ	［ログ出力アクティビティ］サブダイアログの［式］の［値］
4	繰り返しアクティビティ	• ［繰り返し条件設定］ダイアログの［条件式］ • ［繰り返しリスト設定］ダイアログの［繰り返しリスト］ • ［条件式の作成］ダイアログの［条件式］
5	分岐アクティビティ	• ［条件設定］ダイアログ（分岐アクティビティ）の［条件式］ • ［条件式の作成］ダイアログの［条件式］
6	並列処理アクティビティ（リンクコネクション）	リンクコネクションの［条件設定］ダイアログの［条件式］

XPath の中では，拡張関数を使用して変数内の情報を取得する方法と，変数名を直接指定して情報を取得する方法の 2 つがあります。拡張関数については，「5.6.21(3)　拡張関数」を参照してください。

(1)　拡張関数を使用して変数内の情報を取得する方法

拡張関数を使用して変数内の情報を取得する方法を次に示します。

(a)　代入アクティビティサブダイアログでの指定

コピー元の値を取得するために，［値］には拡張関数で XPath を指定します。

属性 propertyName の値を取得する場合

拡張関数 bpws:getVariableProperty を使用して，属性 propertyName の値を取得します。変数の型がメッセージ型（XML または non-XML）の場合に使用できます。

指定方法

```
bpws:getVariableProperty('variableName','propertyName')
```

- variableName：変数名を指定します。
- propertyName：variableName で指定した変数内で定義されている，xml 型以外の部分名を指定します。

指定例

```
bpws:getVariableProperty('VariableX','PropertyY')
```

変数の値を直接指定する場合

拡張関数 csc:getVariableData を使用して変数の値を取得します。変数の型がメッセージ型（XML または non-XML），string 型，numeric 型，または boolean 型の場合に使用できます。

指定方法（メッセージ型（XML または non-XML）のとき）

```
csc:getVariableData('variableName','locationPath')
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
csc:getVariableData('VariableX','input/forInvoke/depositData/id')
```

指定方法（string 型，numeric 型，または boolean 型のとき）

```
csc:getVariableData('variableName')
```

- variableName：変数名を指定します。

指定例

```
csc:getVariableData('VariableX')
```

ヘッダ割当変数の初期化状態を取得する場合

拡張関数 csc:getMessageInitialize を使用して，ヘッダ割当変数の初期化状態を取得します。変数の型がメッセージ型（XML，non-XML または any）の場合に使用できます。

指定方法

```
csc:getMessageInitialize('variableName')
```

- variableName：ヘッダ割当変数名を指定します。

指定例

```
csc:getMessageInitialize('VariableX')
```

メッセージ型変数（non-XML）の 16 進数形式の文字列を取得する場合

拡張関数 csc:getHexVariableData を使用して，メッセージ型変数（non-XML または any）の 16 進数形式の文字列（半角英数大文字）を取得します。

変数の型がメッセージ型（non-XML または any）の場合に使用できます。

指定方法

```
csc:getHexVariableData('variableName','beginIndex','compNumber')
```

- variableName：取得元のメッセージ型（non-XML または any）の変数名を指定します。
- beginIndex：取得開始位置のバイト数を指定します。
- compNumber：取得するバイト数を指定します。

指定例

```
csc:getHexVariableData('VariableX','2','1')
```

文字列を 16 進数形式で取得する場合

拡張関数 csc:getHexString を使用して，16 進数形式の文字列（半角英数大文字）を取得します。

指定方法

```
csc:getHexString('convertString','characterCode')
```

- convertString：16 進数形式に変換する文字列を指定します。
- characterCode：文字列からバイトデータにエンコードする場合に使用する文字コードを指定します。指定できる文字コードは，この関数を使用する環境の JavaVM に依存します。

指定例

```
csc:getHexString('PNG','UTF-8')
```

(b) 条件設定ダイアログ，繰り返し条件設定ダイアログ，条件式の作成ダイアログでの指定

変数の値を判定する条件式を設定するために，［条件式］に条件となる式を XPath で指定します。

参考

XPath を直接指定する代わりに，［条件式の作成］ダイアログで項目を指定して XPath を作成することもできます。

属性 propertyName の値を条件式にする場合

拡張関数 bpws:getVariableProperty を使用して，属性 propertyName の値により条件文を指定します。変数の型がメッセージ型（XML または non-XML）の場合に使用できます。

指定方法

次のどちらかの方法で指定します。

方法 1

```
bpws:getVariableProperty('variableName','propertyName') = "比較対象の値"
```

方法 2

```
bpws:getVariableProperty('variableName','propertyName') = bpws:getVariableProperty('variableName','propertyName')
```

- variableName：変数名を指定します。
- propertyName：variableName で指定した変数内で定義されている，xml 型以外の部分名を指定します。

指定例

```
bpws:getVariableProperty('VariableX','PropertyY') = "HITACHI"
```

変数の値を条件式にする場合

拡張関数 csc:getVariableData を使用して，属性 propertyName の値により条件文を指定します。変数の型がメッセージ型（XML または non-XML），string 型，numeric 型，または boolean 型の場合に使用できます。

指定方法（メッセージ型（XML または non-XML）のとき）

```
csc:getVariableData('variableName','locationPath') = "比較対象の値"
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
csc:getVariableData('VariableX','input/forInvoke/depositData/userName/firstName') = "Taro"
```

指定方法（string 型，numeric 型，または boolean 型のとき）

```
csc:getVariableData('variableName') = "比較対象の値"
```

- variableName：変数名を指定します。

指定例

```
csc:getVariableData('VariableX') = false()
```

注　VariableX が boolean 型の場合です。

ヘッダ割当変数の初期化状態を条件式にする場合

拡張関数 csc:getMessageInitialize を使用して，ヘッダ割当変数の初期化状態を条件式に指定します。変数の型がメッセージ型（XML，non-XML または any）の場合に使用できます。

指定方法

```
csc:getMessageInitialize('variableName') = "比較対象の値"
```

- variableName：ヘッダ割当変数名を指定します。

指定例

```
csc:getMessageInitialize('VariableX') = false()
```

メッセージ型変数（non-XML）の 16 進数形式の文字列を条件式にする場合

拡張関数 csc:getHexVariableData を使用して，メッセージ型変数（non-XML または any）の 16 進数形式の文字列（半角英数大文字）を取得します。

変数の型がメッセージ型（non-XML または any）の場合に使用できます。

指定方法

```
csc:getHexVariableData('variableName','beginIndex','compNumber') =
"比較対象の値の16進表記"
```

- variableName：取得元のメッセージ型（non-XML または any）の変数名を指定します。
- beginIndex：取得開始位置のバイト数を指定します。
- compNumber：取得するバイト数を指定します。

指定例 1（先頭から 3 バイト目の値を取得して判定する場合）

```
csc: getHexVariableData ('VariableX','2','1') = "4e"
```

指定例 2（先頭から 3 バイトの値を取得し，その文字列を拡張関数 csc:getHexString を使用して判定する場合）

```
csc: getHexVariableData ('VariableX','0','3') =
csc:getHexString('PNG','UTF-8')
```

16 進数形式で取得した文字列を条件式にする場合

拡張関数 csc:getHexString を使用して，16 進数形式の文字列（半角英数大文字）を条件式に指定します。

指定方法

```
csc:getHexString('convertString','characterCode') = "比較対象の値の16進表記"
```

- convertString：16 進数形式に変換する文字列を指定します。
- characterCode：文字列からバイトデータにエンコードする場合に使用する文字コードを指定します。指定できる文字コードは，この関数を使用する環境の JavaVM に依存します。

指定例

```
csc:getHexString('PNG','UTF-8') = "504E47"
```

(c) 待機アクティビティダイアログでの指定

［待機時間］の［式］には拡張関数で XPath を指定します。

属性 propertyName の値を取得する場合

拡張関数 bpws:getVariableProperty を使用して，属性 propertyName の値を取得します。変数の型がメッセージ型（XML または non-XML）の場合に使用できます。

指定方法

```
bpws:getVariableProperty('variableName','propertyName')
```

- variableName：変数名を指定します。
- propertyName：variableName で指定した変数内で定義されている，xml 型以外の部分名を指定します。

指定例

```
bpws:getVariableProperty('VariableX','PropertyY')
```

変数の値を直接指定する場合

拡張関数 csc:getVariableData を使用して，属性 propertyName の値を取得します。変数の型がメッセージ型（XML または non-XML）または string 型の場合に使用できます。

指定方法（メッセージ型（XML または non-XML）のとき）

```
csc:getVariableData('variableName','locationPath')
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（[変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
csc:getVariableData('VariableX','input/forInvoke/depositData/id')
```

指定方法（string 型のとき）

```
csc:getVariableData('variableName')
```

- variableName：変数名を指定します。

指定例

```
csc:getVariableData('VariableX')
```

ヘッダ割当変数の初期化状態を取得する場合

拡張関数 csc:getMessageInitialize を使用して，ヘッダ割当変数の初期化状態を取得します。変数の型がメッセージ型（XML，non-XML または any）の場合に使用できます。

指定方法

```
csc:getMessageInitialize('variableName')
```

- variableName：ヘッダ割当変数名を指定します。

指定例

```
csc:getMessageInitialize('VariableX')
```

メッセージ型変数（non-XML）の 16 進数形式の文字列を取得する場合

拡張関数 csc:getHexVariableData を使用して，メッセージ型変数（non-XML または any）の 16 進数形式の文字列（半角英数大文字）を取得します。

変数の型がメッセージ型（non-XML または any）の場合に使用できます。

指定方法

```
csc:getHexVariableData('variableName','beginIndex','compNumber')
```

- variableName：取得元のメッセージ型（non-XML または any）の変数名を指定します。
- beginIndex：取得開始位置のバイト数を指定します。
- compNumber：取得するバイト数を指定します。

指定例

```
csc:getHexVariableData('VariableX','2','1')
```

文字列を 16 進数形式で取得する場合

拡張関数 csc:getHexString を使用して，16 進数形式の文字列（半角英数大文字）を取得します。

指定方法

```
csc:getHexString('convertString','characterCode')
```

- convertString：16 進数形式に変換する文字列を指定します。
- characterCode：文字列からバイトデータにエンコードする場合に使用する文字コードを指定します。指定できる文字コードは，この関数を使用する環境の JavaVM に依存します。

指定例

```
csc:getHexString('PNG','UTF-8')
```

注意事項

待機時間を指定する場合は，次の点に注意してください。

- 待機時間に「間隔」を指定する場合
  XPath 式を評価して得ることができる文字列は，XML Schema 型 duration（xsd:duration）に従っている必要があります。
- 待機時間に「期限」を指定する場合
  XPath 式を評価して得ることができる文字列は，XML Schema 型 dateTime（xsd:dateTime）に従っている必要があります。

これらの仕様は BPEL 標準規格により定められています。そのため，ビジネスプロセスの numeric 型および boolean 型の変数は指定できません。

なお，XPath 式を指定する代わりに，「"」または「'」で囲むことで，直接間隔または期限を指定することもできます。

詳細については，「5.6.11(2)　待機時間として指定できる文字列」を参照してください。

(2)　変数名を直接指定して情報を取得する方法

変数名を直接指定して情報を取得する方法を次に示します。

(a)　代入アクティビティサブダイアログでの指定

［コピー元］の［値］を取得するための XPath を直接指定します。

属性 propertyName の値を取得する場合

拡張関数を使用して変数内の情報を取得する方法を実施してください。

変数の値を直接指定する場合

「$variableName」で変数名を直接指定します。変数の型がメッセージ型，string 型，numeric 型，または boolean 型の場合に使用できます。

指定方法（メッセージ型のとき）

```
$variableName/locationPath
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/input/forInvoke/depositData/id
```

指定方法（string 型，numeric 型，または boolean 型のとき）

```
$variableName
```

- variableName：変数名を指定します。

指定例

```
$VariableX
```

(b) 条件設定ダイアログ，繰り返し条件設定ダイアログ，条件式の作成ダイアログでの指定

［条件式］の XPath を直接指定します。

> **参考**
>
> XPath を直接指定する代わりに，［条件式の作成］ダイアログで項目を指定して XPath を作成することもできます。

属性 propertyName の値を条件式にする場合

拡張関数を使用して条件式を指定する方法を実施してください。

変数の値を条件式にする場合

「$variableName」で変数名を直接指定します。変数の型がメッセージ型，string 型，numeric 型，または boolean 型の場合に使用できます。

指定方法（メッセージ型のとき）

```
$variableName/locationPath
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/input/forInvoke/depositData/userName/firstName = ”Taro”
```

指定方法（string 型，numeric 型，または boolean 型のとき）

```
$variableName
```

- variableName：変数名を指定します。

指定例

```
$VariableX = false()
```

注　VariableX が boolean 型の場合です。

(c) 待機アクティビティダイアログでの指定

［待機時間］の［式］には拡張関数で XPath を指定します。

属性 propertyName の値を取得する場合

拡張関数を使用して変数内の情報を取得する方法を実施してください。

変数の値を直接指定する場合

「$variableName」で変数名を直接指定します。変数の型がメッセージ型または string 型の場合に使用できます。

指定方法（メッセージ型のとき）

```
$variableName/locationPath
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（[変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/input/forInvoke/depositData/id
```

指定方法（string 型のとき）

```
$variableName
```

- variableName：変数名を指定します。

指定例

```
$VariableX
```

注意事項

待機時間を指定する場合は，次の点に注意してください。

- 待機時間に「間隔」を指定する場合
  XPath 式を評価して得ることができる文字列は，XML Schema 型 duration（xsd:duration）に従っている必要があります。
- 待機時間に「期限」を指定する場合
  XPath 式を評価して得ることができる文字列は，XML Schema 型 dateTime（xsd:dateTime）に従っている必要があります。

これらの仕様は BPEL 標準規格により定められています。そのため，ビジネスプロセスの numeric 型および boolean 型の変数は指定できません。

なお，XPath 式を指定する代わりに，「"」または「'」で囲むことで，直接間隔または期限を指定することもできます。

詳細については，「5.6.11(2)　待機時間として指定できる文字列」を参照してください。

(d)　繰り返しリスト設定ダイアログでの指定

［繰り返しリスト］の［式］に XPath で直接指定します。なお，リスト指定方式の場合，拡張関数は使用できません。

繰り返し要素のノードを指定する場合

指定方法

```
$variableName/locationPath
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示す繰り返し要素のパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/*[local-name()='loop-element' and namespace-uri()='http://example.com/sampl
e']/*[local-name()='child-element' and namespace-uri()='http://example.com/sample']
```

繰り返し要素のノードを複数指定する場合

指定方法

```
($variableName/locationPath | $variableName/locationPath)
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示す繰り返し要素のパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
($VariableX/*[local-name()='loop-element' and namespace-uri()='http://example.com/samp
le']/*[local-name()='child-element' and namespace-uri()='http://example.com/sample'][1
] | $VariableX/*[local-name()='loop-element' and namespace-uri()='http://example.com/s
ample']/*[local-name()='child-element' and namespace-uri()='http://example.com/sample'
][3])
```

特定の範囲の繰り返し要素のノードを指定する場合

指定方法（範囲指定）

```
$variableName/locationPath[position()<10]
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示す繰り返し対象のパス（［変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/*[local-name()='loop-element' and namespace-uri()='http://example.com/sample']/*[local-name()='child-element' and namespace-uri()='http://example.com/sample'][position()<10]
```

(e) ログ出力アクティビティサブダイアログでの指定

［式］の［値］に XPath を直接指定します。変数の型が message 型，string 型，numeric 型，または boolean 型の場合に，変数名を「$variableName」で指定します。なお，message 型変数のうち，XML 変数以外の変数から取得したデータは，埋め込み文字として使用できません。

message 型変数の特定の要素を指定する場合

指定方法

```
$variableName/locationPath
```

- variableName：変数名を指定します。
- locationPath：XML スキーマのノードを示すパス（[変数の表示］ダイアログの［選択パス］で取得したパス）を指定します。

指定例

```
$VariableX/input/forInvoke/depositData/id
```

特定の変数を指定する場合

指定方法

```
$variableName
```

- variableName：変数名を指定します。

指定例

```
$VariableX
```

(3) 拡張関数

拡張関数には，次に示す(i)～(v)の 5 つがあります。拡張関数の形式，引数，および拡張関数を使用する場合の注意事項を説明します。

(i) bpws:getVariableProperty

形式

bpws:getVariableProperty('variableName','propertyName')

引数

variableName

変数名を指定します。

propertyName

variableName で指定した変数内で定義されている，xml 型以外の部分名を指定します。

(ii) csc:getVariableData

形式

csc:getVariableData('variableName','locationPath')

引数

variableName

変数名を指定します。

locationPath

XML スキーマのノードを示すパスを指定します。

注意事項

- 変数がメッセージ型以外の場合，引数には variableName だけを指定してください。
- 変数がメッセージ型以外の場合，関数は変数の型に応じたデータ型を返します。
- variableName で指定した変数のデータに対して，locationPath で指定したパスを評価した結果は，文字列型（java.lang.String へマッピング）で返します。
- locationPath に「'」を含む場合，locationPath を「"」で囲んでください。

(iii) getMessageInitialize

形式

csc:getMessageInitialize('variableName')

引数

variableName

変数名を指定します。

注意事項

- variableName に指定する変数の variable 要素は，xml 型で定義してください。
- variableName に指定した変数の初期化状態は，boolean 型で返します。
  初期化済みの場合：true
  未初期化の場合：false

(iv) getHexVariableData

形式

csc:getHexVariableData('variableName','beginIndex','compNumber')

引数

variableName

変数名を指定します。

beginIndex

取得開始位置のバイト数を指定します。

compNumber

取得するバイト数を指定します。

注意事項

- beginIndex には 0 以上，compNumber には 1 以上の値を指定できます。ただし，beginIndex の値と compNumber の値を足した数が，variableName に指定した変数のバイト数より小さくなるよう指定してください。beginIndex の値と compNumber の値を足した数が，variableName に指定した変数のバイト数以上の場合は，XPathFunctionException が発生します。
- beginIndex および compNumber に数値以外の値を指定した場合，XPathFunctionException が発生します。
- variableName に変数として存在しない値を指定した場合，XPathFunctionException が発生します。

(v) getHexString

形式

csc:getHexString('convertString','characterCode')

引数

convertString

16 進数形式に変換する文字列を指定します。

characterCode

文字列からバイトデータを取得する際に使用する文字コードを指定します。

注意事項

- convertString に空文字（''）を指定した場合は，処理結果として空文字を返します。
- 変数を使用する環境の JavaVM が対応する文字コード以外を characterCode に設定した場合，XPathFunctionException が発生します。

5.7　コメントの配置

コメントによって，ビジネスプロセスエディタ上にビジネスプロセスやアクティビティなどの説明を記載できます。

コメントは，ビジネスプロセスエディタのパレットから，［コメント］を選択して，配置します。

編集の開始方法

- コメントをダブルクリックする。
- コメントが選択状態のとき，コメントをシングルクリックする。
- コメントが選択状態のとき，［F2］キーを押す。
- コメントのコンテキストメニューで［設定］を選択する。

編集の終了方法

- 編集中のコメントの外側をクリックする。
- ［Ctrl］キー＋［Enter］キーを押す。

背景色の変更方法

- コメントのコンテキストメニューで［背景色］を選択する。

5.8　ビジネスプロセスの保存

ビジネスプロセス定義画面で編集中のビジネスプロセスの内容を，編集途中や編集終了時に新規保存したり，上書き保存したりできます。保存した内容はリポジトリに保存されます。

変数定義で指定した電文フォーマット定義ファイル，およびデータ変換定義で生成したデータ変換定義ファイルは，ビジネスプロセス定義画面と同時に保存されます。

ビジネスプロセスを保存する手順を次に示します。

方法 1

Eclipse のメニューから，［ファイル］－［保存］を選択します。

方法 2

Eclipse のメニューから，［ファイル］－［すべて保管］を選択します。

方法 3

ビジネスプロセス定義画面で，［Ctrl］キー＋［S］キーを押します。

なお，ビジネスプロセス定義画面を閉じてビジネスプロセスの定義を終了する場合に，ビジネスプロセスが保存されていないときは，リソース保管ダイアログが表示されます。リソース保管ダイアログでは，保存されていないビジネスプロセスの定義を保存できます。

注意事項

不正なデータが入力されている場合，ビジネスプロセスを保存できないことがあります。その場合，表示されているメッセージに従って対処してください。

5.9　ビジネスプロセスの編集

保存したビジネスプロセスの内容を変更できます。ビジネスプロセスの内容を変更するには，ビジネスプロセス定義画面を表示して，内容を編集します。ビジネスプロセス定義画面は，ツリービューのサービス定義一覧から，該当するビジネスプロセスを選択し，ダブルクリックすると表示されます。

ビジネスプロセスの内容の定義方法については，「5.3　ビジネスプロセスの内容の定義」を参照してください。

5.9.1　ビジネスプロセスおよびアクティビティの定義情報を変更する

ビジネスプロセス，およびアクティビティの定義情報を変更できます。

注意事項

定義情報をプロパティービューで変更している場合は，変更を反映してからほかの操作を実行してください。

(1)　ビジネスプロセスの定義情報を変更する

ビジネスプロセスの定義情報を変更するには，次の 2 つの方法があります。

方法 1

1. ツリービューのサービス定義一覧で表示されたサービスから，該当するビジネスプロセスを選択し，ダブルクリックします。
   選択したビジネスプロセスのビジネスプロセス定義画面が表示されます。
2. 内容を変更します。

方法 2

プロパティから，次に示すビジネスプロセスの定義情報を変更できます。次の情報以外を変更する場合は，方法 1 を実施します。

- ビジネスプロセス名
- サービス ID※
- ステータスの永続化
- スコープアクティビティのデフォルトハンドラ

注※

サービス ID を変更する場合，8 バイト以内の英数字およびアンダーバー（_）で指定してください。

1. ツリービューのサービス定義一覧で表示されたサービスから，該当するビジネスプロセスを選択します。

選択したビジネスプロセスに関するプロパティがプロパティービューに表示されます。

2. 変更する項目を選択して変更します。
   入力項目の前後に半角スペースや全角スペースを指定できません。

(2) アクティビティの定義情報を変更する

アクティビティの定義情報を変更するには，次の 2 つの方法があります。

方法 1

1. ツリービューのサービス定義一覧で表示されたサービスから，該当するビジネスプロセスを選択し，ダブルクリックします。
   ビジネスプロセス定義画面が表示されます。
2. キャンバス上の該当するアクティビティを選択します。
3. ダブルクリック，または右クリックで「設定」を選択します。
   アクティビティのダイアログが表示されます。
4. 内容を変更します。

方法 2

開始アクティビティおよび終了アクティビティ以外のアクティビティの場合，プロパティからサービス名を変更できます。サービス名以外の情報を変更する場合は，方法 1 を実施します。

アクティビティの定義情報をプロパティから変更する手順を次に示します。

1. ビジネスプロセス定義画面のキャンバス上のアクティビティを 1 つ選択します。
   選択したアクティビティに関するプロパティがプロパティービューに表示されます。
2. 変更する項目をダブルクリックして変更します。
   入力項目の前後に半角スペースや全角スペースを指定できません。

5.9.2 アクティビティ名を変更する

開始アクティビティおよび終了アクティビティを除く，すべてのアクティビティ名を変更できます。

アクティビティ名は，ビジネスプロセス内で一意である必要があります。また，入力項目の前後に半角スペースや全角スペースを指定できません。

注意事項

アクティビティ名などをプロパティービューで変更している場合は，変更を反映してからほかの操作を実行してください。

アクティビティ名を変更するには，次の 2 つの方法があります。

方法 1

1. キャンバス上でアクティビティを選択します。
2. アクティビティを選択したまま，左クリックまたは［F2］キーを押します。
3. アクティビティ名を編集します。
4. ［Enter］キーを押すか，カーソルを外します。
   アクティビティ名が変更されます。

変更後のアクティビティ名が，ビジネスプロセス内ですでに使用されている場合や空白にした場合は，エラーになり，変更前のアクティビティに戻ります。

方法 2

1. キャンバス上でアクティビティを選択します。
   プロパティービューに選択したアクティビティのプロパティが表示されます。
2. プロパティービューに表示されるアクティビティ名を選択して，アクティビティ名を編集します。
3. ［Enter］キーを押すか，カーソルを外します。

5.9.3 運用中のビジネスプロセス定義を変更する

業務の変更や改善に応じて運用中のビジネスプロセスの定義を変更する場合，変更後の運用に応じて次の中から変更方法を選択します。

(i) 変更前のビジネスプロセスを新規のプロセスとして呼び出す場合

変更後のビジネスプロセスとして，新規にビジネスプロセスを作成して運用します。変更前後のビジネスプロセスで，サービス名とビジネスプロセス定義名を区別する必要があります。

(ii) 変更前のビジネスプロセスと相関セットの一意性を保つことができない場合

変更後のビジネスプロセスとして，新規にビジネスプロセスを作成して運用します。変更前後のビジネスプロセスで，サービス名とビジネスプロセス定義名を区別する必要があります。

(iii) 変更前のビジネスプロセス定義に属するプロセスがすべて完了している場合

変更前のビジネスプロセスに属するプロセスをすべて削除し，変更後のビジネスプロセス定義に入れ替えます。

(iv) (i)～(iii)以外の場合でサービスリクエスタを変更できるとき

変更後のビジネスプロセスとして，新規にビジネスプロセスを作成して運用します。変更前後のビジネスプロセスで，サービス名とビジネスプロセス定義名を区別する必要があります。

(v) (i)～(iii)以外の場合でサービスリクエスタを変更できないとき

バージョンアップしたビジネスプロセスを作成して並行運用します。バージョンアップ後の変更内容によってサービスリクエスタの変更も必要です。

それぞれの方法を次に示します。

(1)　新規のビジネスプロセスを作成して運用する方法

(i)，(ii)，(iv)の場合，新規にビジネスプロセスを作成し，変更後のサービス名およびビジネスプロセス定義名でビジネスプロセスを運用します。手順を次に示します。

1. 運用環境からリポジトリ情報をエクスポートし，開発環境にインポートします。

2. 既存のビジネスプロセス定義を基に新規のビジネスプロセスを作成します。

   注　新規および既存のビジネスプロセスで，サービス名とビジネスプロセス定義名を区別する必要があります。

3. ビジネスプロセスをパッケージングします。

   パッケージングについては，「8.2　パッケージング」を参照してください。

4. ビジネスプロセスの配備定義をします。

   配備定義については，「8.3　HCSC コンポーネントの配備定義」を参照してください。

5. 手順 2.で作成したビジネスプロセスを含むリポジトリ情報をエクスポートし，運用環境にインポートします。

6. ビジネスプロセスを運用環境から実行環境に配備します。

   運用環境での操作については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.1.13　ビジネスプロセスを配備する」を参照してください。

(2)　既存のビジネスプロセスを削除してから最新のビジネスプロセスに入れ替える方法

(iii)の場合，既存のビジネスプロセスを実行環境から削除したあと最新のビジネスプロセスに入れ替えます。手順を次に示します。

1. 運用環境からリポジトリ情報をエクスポートし，開発環境にインポートします。

2. 既存のビジネスプロセスの配備定義を削除します。

   ビジネスプロセスの配備定義の削除については，「8.3　HCSC コンポーネントの配備定義」を参照してください。

3. 既存のビジネスプロセス定義をビジネスプロセス定義画面で変更します。

4. ビジネスプロセスをパッケージングします。

   パッケージングについては，「8.2　パッケージング」を参照してください。

5. ビジネスプロセスの配備定義をします。

   配備定義については，「8.3　HCSC コンポーネントの配備定義」を参照してください。

6. 運用環境で，既存のビジネスプロセス定義に属するプロセスがすべて完了したのを確認してから，実行環境に配備された HCSC コンポーネントを削除します。

7. 手順 2.および手順 3.で変更したビジネスプロセスを含むリポジトリ情報をエクスポートし，運用環境にインポートします。

8. ビジネスプロセスを運用環境から実行環境に配備します。
   運用環境での操作については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.1.13　ビジネスプロセスを配備する」を参照してください。

(3)　並行運用する方法

(v)の場合，既存のビジネスプロセスとバージョンアップしたビジネスプロセスを並行運用します。なお，ビジネスプロセスをバージョンアップする前に，「5.9.4　ビジネスプロセスのバージョンアップ」を確認してください。

1. 運用環境からリポジトリ情報をエクスポートし，開発環境にインポートします。

2. ビジネスプロセスの配備定義を削除します。
   ビジネスプロセスの配備定義の削除については，「8.3　HCSC コンポーネントの配備定義」を参照してください。

3. ツリービューのサービス定義一覧のビジネスプロセスを右クリックし，［バージョンアップ］を選択します。

4. 手順 3.で選択したビジネスプロセスをビジネスプロセス定義画面で編集します。

5. ビジネスプロセスをパッケージングします。
   パッケージングについては，「8.2　パッケージング」を参照してください。

6. ビジネスプロセスの配備定義をします。
   配備定義については，「8.3　HCSC コンポーネントの配備定義」を参照してください。

7. 運用環境で既存のビジネスプロセスを停止および削除します。

8. 手順 2.～4.で変更したビジネスプロセスを含むリポジトリ情報をエクスポートし，運用環境にインポートします。

9. ビジネスプロセスを運用環境から実行環境に配備します。
   運用環境での操作については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.1.13　ビジネスプロセスを配備する」を参照してください。

5.9.4　ビジネスプロセスのバージョンアップ

ここでは，ビジネスプロセスのバージョンアップ後に実施できる操作，実施できない操作，およびバージョンアップ時の注意事項について説明します。

(1)　バージョンアップ後に実施できる操作

ビジネスプロセスのバージョンアップ後に実施できる操作を表に示します。

表 5-20　バージョンアップ後に実施できる操作

#	対象のアクティビティ	操作	備考
1	受付アクティビティ※1	アクティビティ名の変更	－
2		オペレーション名の変更	バージョンアップ後の変更内容によって，オペレーション名の変更が必要になる場合があります。 オペレーション名の変更が必要になる操作は，#4，#7，#15，#18 を参照してください。
3			ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※3※6
4		ボディ割当変数の変更	相関セットの参照先に指定しているボディ割当変数を変更した場合，オペレーション名の変更が必要になります。
5			ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※6
6		ヘッダ割当変数の追加，変更，削除	SOAP 受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※4※6
7		割当相関セット群の追加，変更，削除	割当相関セット群を追加，変更，削除した場合，オペレーション名の変更が必要になります。これに伴い，ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※6
8			相関セット（名前と値のペア）は，すべてのバージョン間で一意である必要があります。
9		通信モデルの変更	ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※6
10		インスタンス生成の変更	－
11		アクティビティの追加，削除	－
12	応答アクティビティ※1	アクティビティ名の変更	－

#	対象のアクティビティ	操作	備考
13	応答アクティビティ※1	オペレーション名の変更	バージョンアップ後の変更内容によって，オペレーション名の変更が必要になる場合があります。 オペレーション名の変更が必要になる操作は，#4，#7，#15，#18 を参照してください。
14			ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※3※6
15		ボディ割当変数の変更	相関セット参照先に指定しているボディ割当変数を変更した場合，オペレーション名の変更が必要になります。
16			ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※6
17		ヘッダ割当変数の追加，変更，削除	SOAP 受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※4※6
18		割当相関セット群の追加，変更，削除	割当相関セット群を追加，変更，削除した場合，オペレーション名の変更が必要になります。これに伴い，ユーザ定義受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※6
19			相関セット（名前と値のペア）は，すべてのバージョン間で一意である必要があります。
20		フォルト名の変更，削除	SOAP 受付を使用している場合，ユーザ定義受付を新規追加する必要があります。※5※6
21		アクティビティの追加，削除	－
22	その他のアクティビティ	アクティビティの追加，変更，削除	サービス呼出アクティビティで割当相関セット群を追加，変更，削除した場合，相関セット（名前と値のペア）は，すべてのバージョン間で一意である必要があります。
23	ユーザ定義受付※2	ユーザ定義受付の追加※7	最新バージョンで使用するユーザ定義受付を追加できます。 また，最新および最新以外のバージョンで共用するユーザ定義受付も追加できます。
24		ユーザ定義受付の変更※7	バージョンに関係なくユーザ定義受付を変更できます。
25		ユーザ定義受付の削除	最新バージョンだけで使用しているユーザ定義受付は削除できます。

（凡例）

－：該当する説明はありません。

注※1

操作は最新のバージョンが対象です。最新以外のバージョンでは参照および変更できません。

注※2

ユーザ定義受付は，バージョンに関係なく参照できます。

注※3

ユーザ定義受付で定義されているほかのオペレーション名に変更する場合，またはユーザ定義受付を使用していて，ユーザ定義受付にオペレーション名を追加する場合は不要です。

注※4

データ検証機能が無効（HCSC サーバランタイム定義ファイルの telegram-notfound-soapheader プロパティに「IGNORE」を指定した場合）で，変更による動作上の問題がない場合は不要です。

注※5

ユーザ定義受付で定義されているほかのフォルト名に変更する場合は不要です。

注※6

ユーザ定義受付を新規追加する場合は「(3)(b)　ユーザ定義受付を使用しているときの注意」を参照してください。

注※7

対応するビジネスプロセスの定義内容を考慮してユーザ定義受付を定義する必要があります。なお，最新以外のバージョンとの整合性はユーザ定義受付の検証でチェックされます。ユーザ定義受付の検証については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.14　ユーザ定義受付の検証」を参照してください。

(2)　バージョンアップ後に実施できない操作

ビジネスプロセスのバージョンアップ後に実施できない操作を表に示します。

表 5-21　バージョンアップ後に実施できない操作

#	対象のアクティビティ	操作	備考
1	ビジネスプロセス※1	サービス ID の変更	－
2		ステータスの永続化の変更	－
3		ビジネスプロセス名の変更	－
4		スコープアクティビティのデフォルトハンドラの変更	－
5	ユーザ定義受付※2	ユーザ定義受付の追加	最新以外のバージョンで使用するユーザ定義受付は追加できません。
6		ユーザ定義受付の削除	バージョンアップを実行したときに定義されていたユーザ定義受付は削除できません。

（凡例）

－：該当する説明はありません。

注※1

操作は最新のバージョンが対象です。最新以外のバージョンでは参照および変更できません。

注※2

ユーザ定義受付は，バージョンに関係なく参照できます。

なお，ビジネスプロセスをバージョンアップした場合，最新以外のバージョンのビジネスプロセスでは次の機能を使用できません。

- ビジネスプロセスのデバッグ
- 新規プロセスインスタンスの呼び出し

(3) ビジネスプロセスのバージョンアップ時の注意事項

ビジネスプロセスをバージョンアップする場合，バージョンアップ後の変更内容によっては最新以外のバージョンのビジネスプロセスを実行できなくなる可能性があります。

次の内容を確認して，変更後の影響を検討してから定義情報を変更してください。また，バージョンアップ前に定義情報のバックアップを取得することを推奨します。

(a) 最新のバージョンでビジネスプロセス定義を変更するときの注意

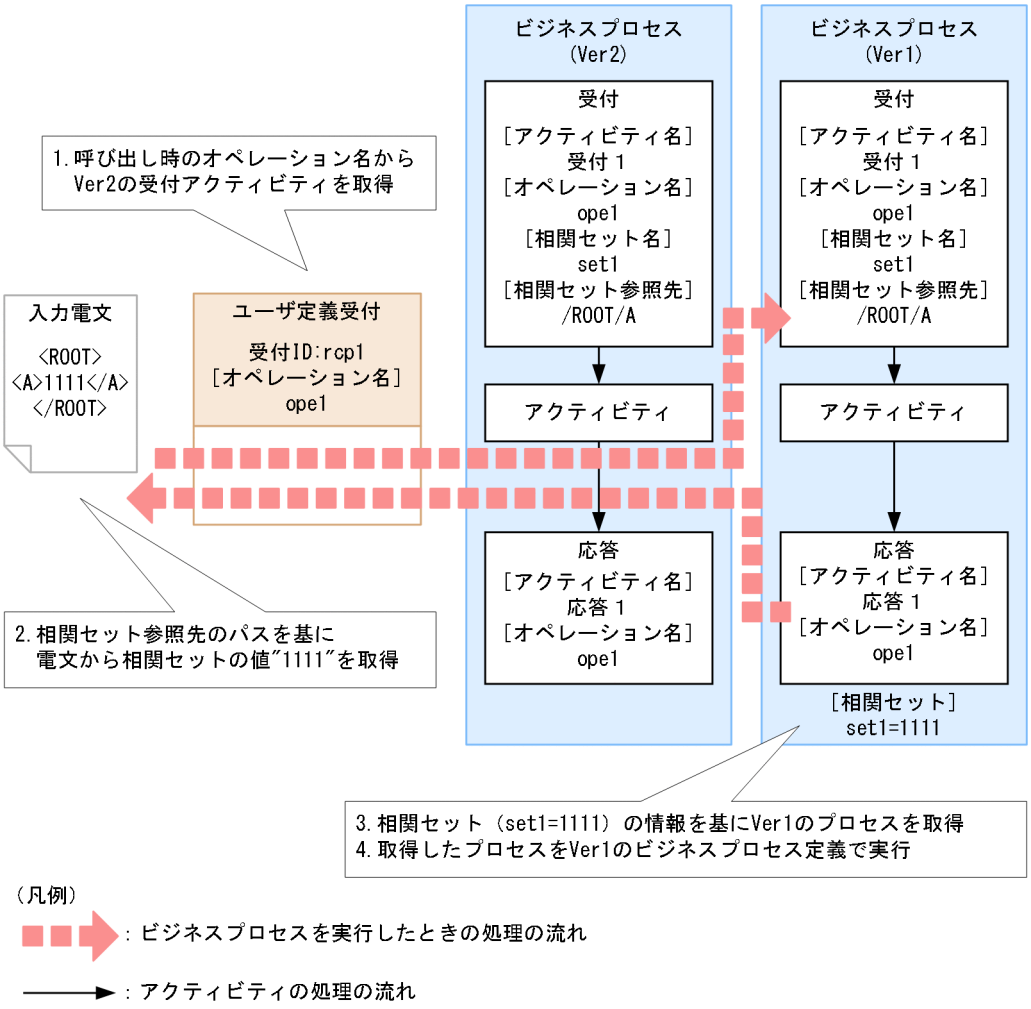
最新以外のバージョンのビジネスプロセスは次の流れで実行されます。

1. 呼び出し時に指定されたオペレーション名を基に，最新バージョンから最新以外のバージョンのビジネスプロセス定義を参照して，該当するオペレーション名が定義されている受付アクティビティを取得します。
2. 受付アクティビティの相関セット取得位置を基に，電文から相関セットの値を取得します。
3. 相関セットの値を基に，実行中のプロセスを取得します。
4. 取得したプロセスのバージョンを最新バージョンから順に比較して，一致した場合そのバージョンのビジネスプロセス定義で処理を続行します。

最新以外のバージョンの完了していないプロセスインスタンスが存在する環境でビジネスプロセスをバージョンアップしたあと，最新のバージョンのビジネスプロセス定義を変更する場合，上記の処理および変更の影響を考慮してビジネスプロセス定義を変更してください。

バージョンアップ後に最新以外のバージョンのビジネスプロセスを実行するときの処理の流れを次の図に示します。

図 5-29　バージョンアップ後に最新以外のバージョンのビジネスプロセスを実行するときの処理の流れ



(b)　ユーザ定義受付を使用しているときの注意

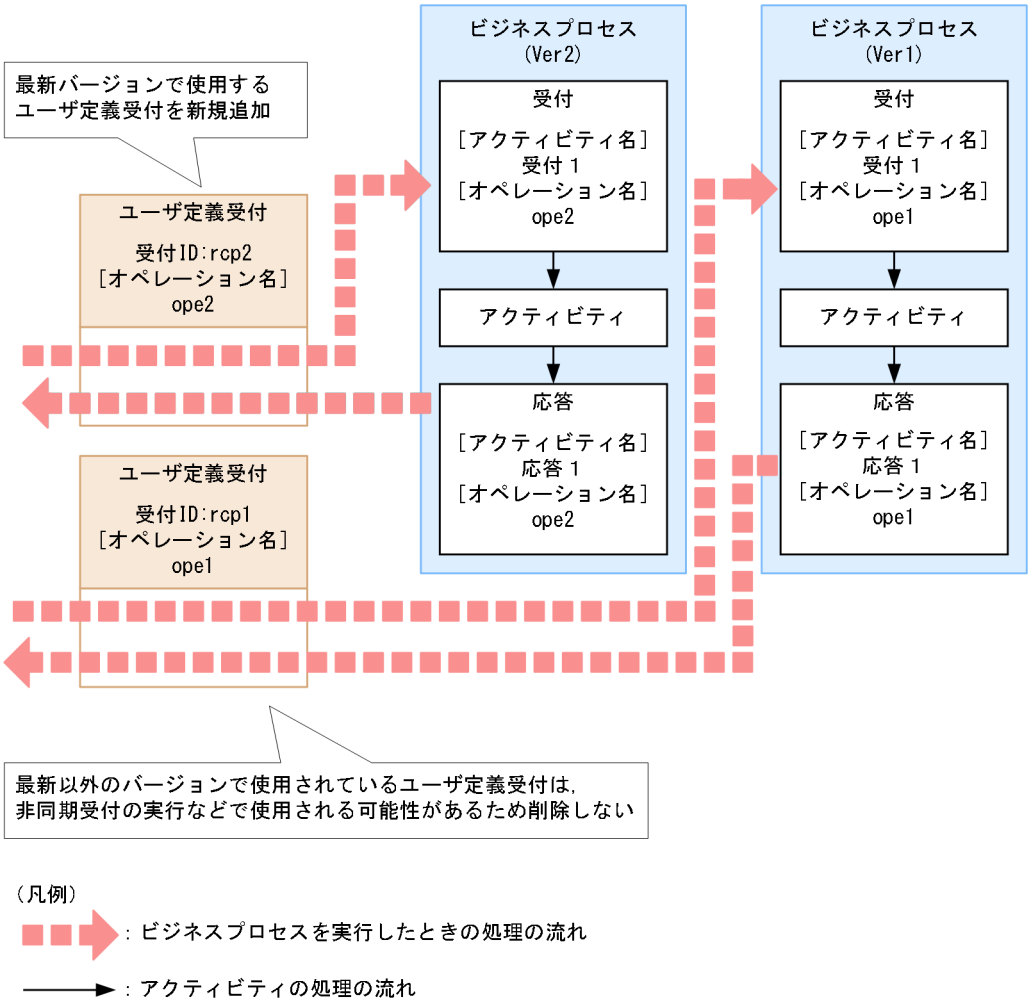
ユーザ定義受付を使用しているビジネスプロセスのバージョンアップ後に次の操作を実施した場合，最新以外のバージョンで使用しているユーザ定義受付を残したまま，最新バージョンで使用するユーザ定義受付を新規追加する必要があります。

- オペレーション名の変更
- ボディ割当変数の変更
- ヘッダ割当変数の追加，変更，削除
- 割当相関セット群の追加，変更，削除
- フォルト名の変更，削除

ただし，上記の場合でもユーザ定義受付を新規追加しなくてもよい場合があります。詳細は「表 5-20 バージョンアップ後に実施できる操作」の備考および注釈を参照してください。

バージョンアップ後にユーザ定義受付を新規追加する必要がある場合の，ビジネスプロセスの変更例を次の図に示します。

図 5-30　ユーザ定義受付を新規追加する必要がある場合の変更例

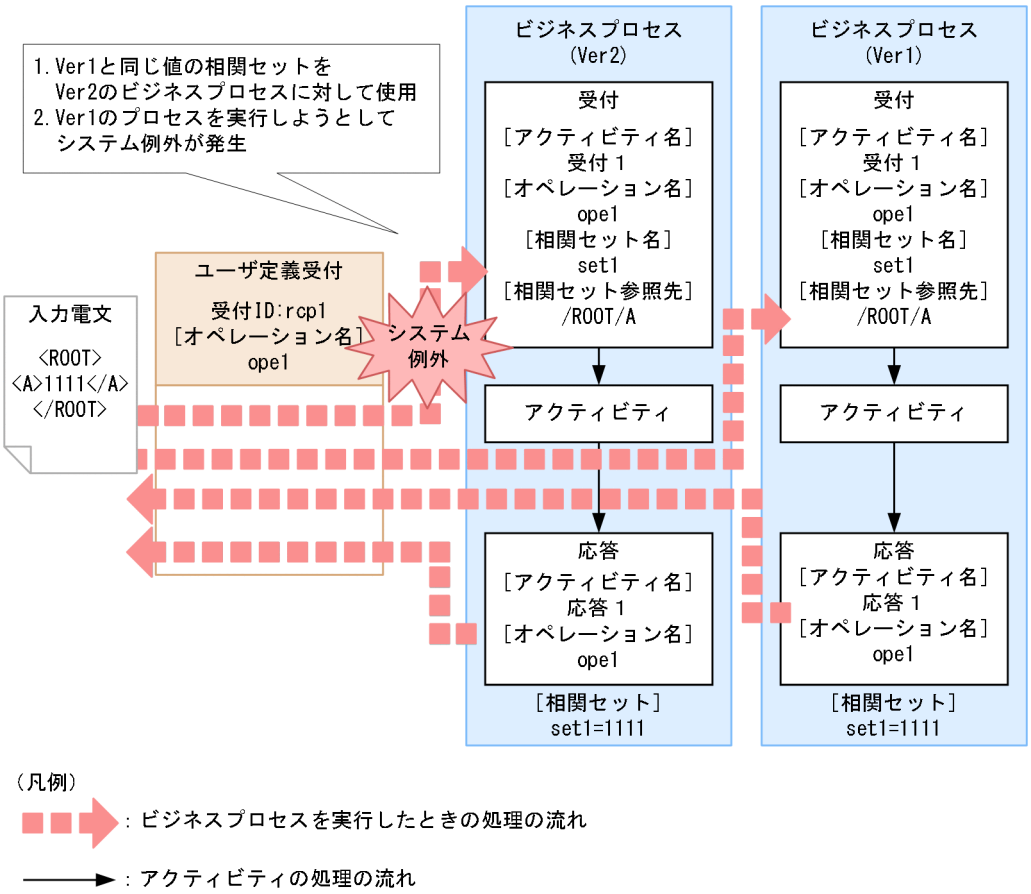


(c)　相関セットの一意性に関する注意

バージョンアップ後に相関セット（名前と値のペア）の一意性が保たれなかった場合，最新バージョンと最新以外のバージョンで予期しないプロセスの呼び出しが発生する可能性があります。

バージョンアップ後にバージョンアップ前と同じ値の相関セットを指定した場合の動作を次の図に示します。

図 5-31　相関セットの一意性が保たれなかった場合（受付で相関セットを設定）



(d)　相関セットに関わる変更を行うときの注意

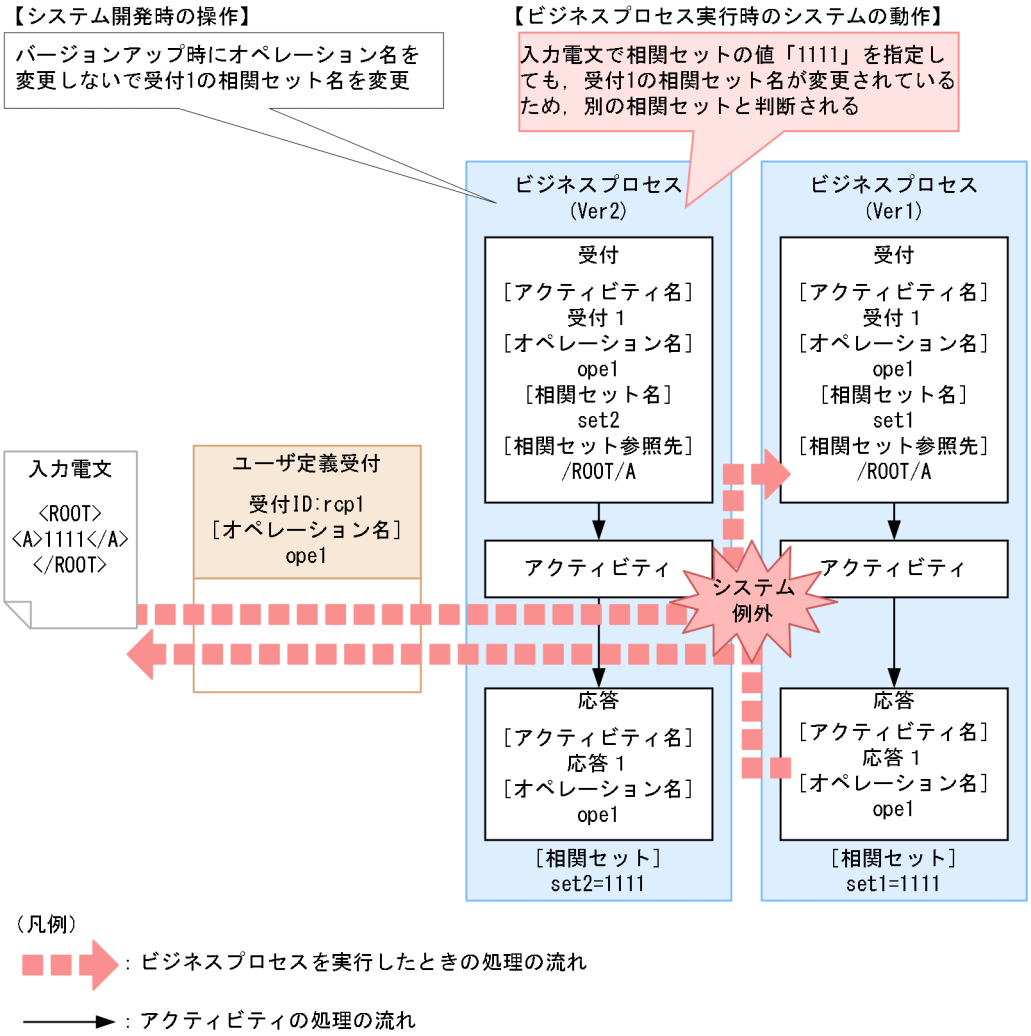
バージョンアップ後に次の操作を実施した場合，相関セットの値が最新バージョンと最新以外のバージョンで一致しなくなる可能性があります。

- 相関セット参照先に指定しているボディ割当変数（部分指定）の変更
- 割当相関セット群の追加，変更，削除

この場合，最新以外のバージョンのプロセスを実行できなくなることがあります。そのため，相関セットに関する変更を行った場合は，オペレーション名の変更が必要になります。

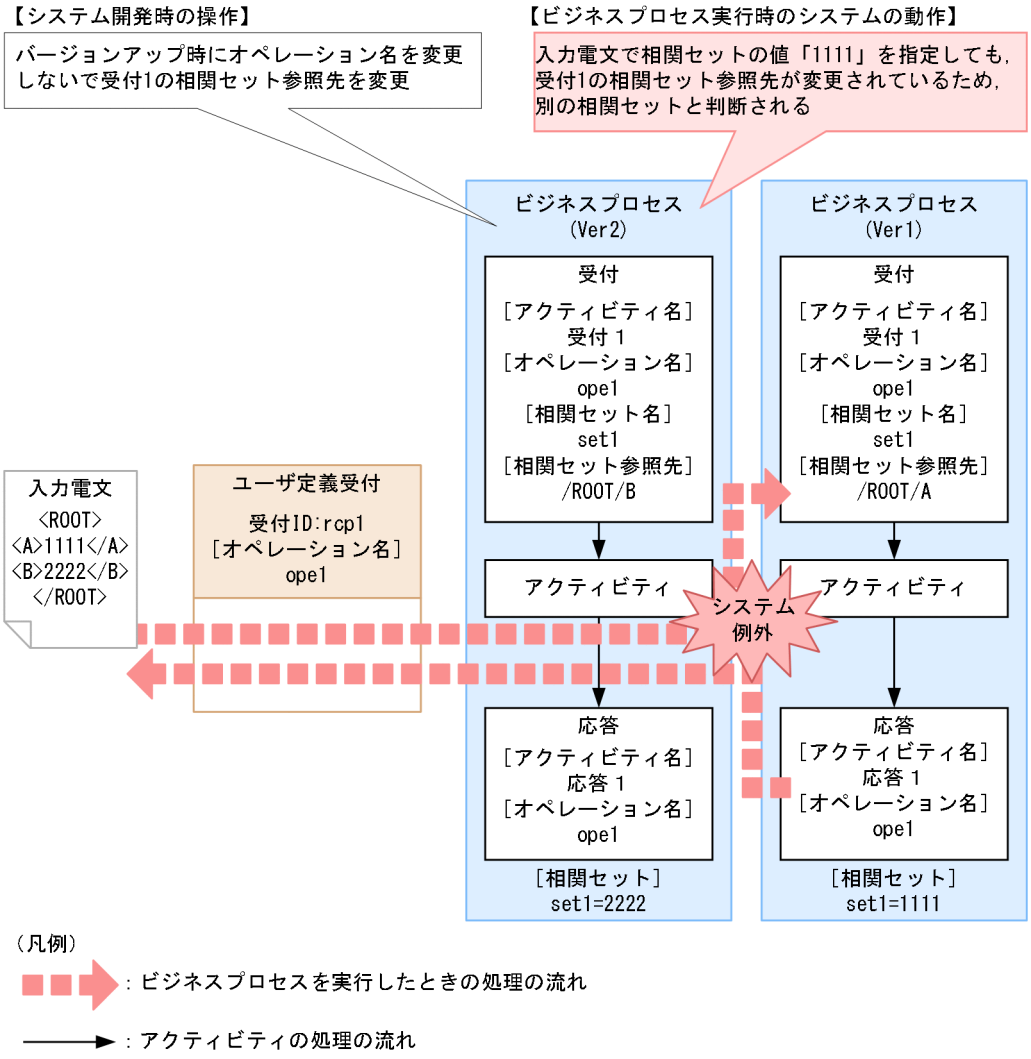
バージョンアップ後に相関セット名を変更した場合の動作を次の図に示します。

図 5-32　相関セットに関わる変更を行う場合（相関セット名を変更）



バージョンアップ後に相関セット参照先を変更した場合の動作を次の図に示します。

図 5-33　相関セットに関わる変更を行う場合（相関セット参照先を変更）



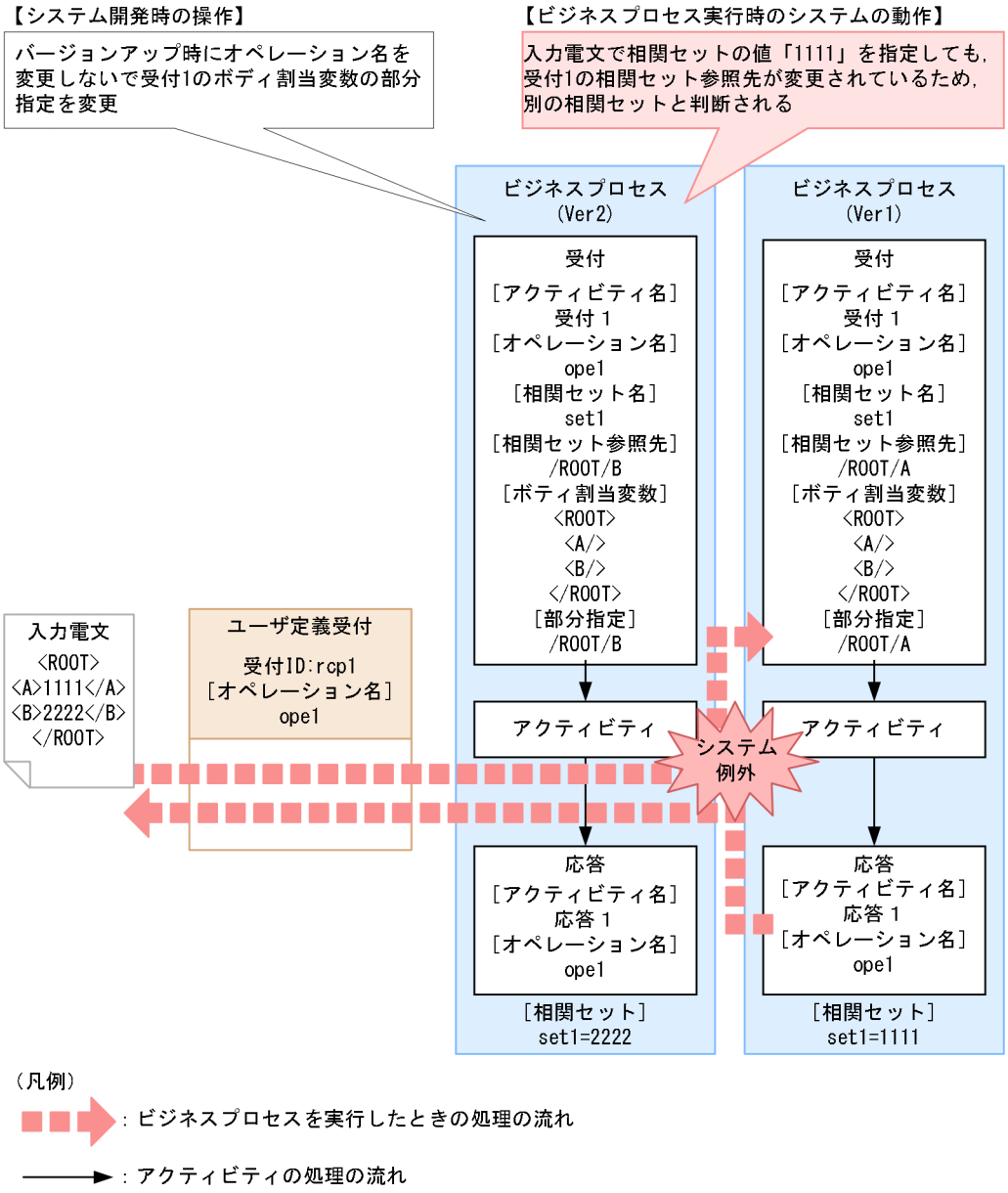
(e)　ボディ割当変数に関わる変更を行うときの注意

バージョンアップ後に，相関セット参照先に指定しているボディ割当変数（部分指定）の変更を行った場合，相関セットの値が最新バージョンと最新以外のバージョンで一致しなくなる可能性があります。この場合，最新以外のバージョンのプロセスを実行できなくなることがあります。

そのため，相関セット参照先に指定しているボディ割当変数（部分指定）を変更した場合は，オペレーション名の変更が必要になります。

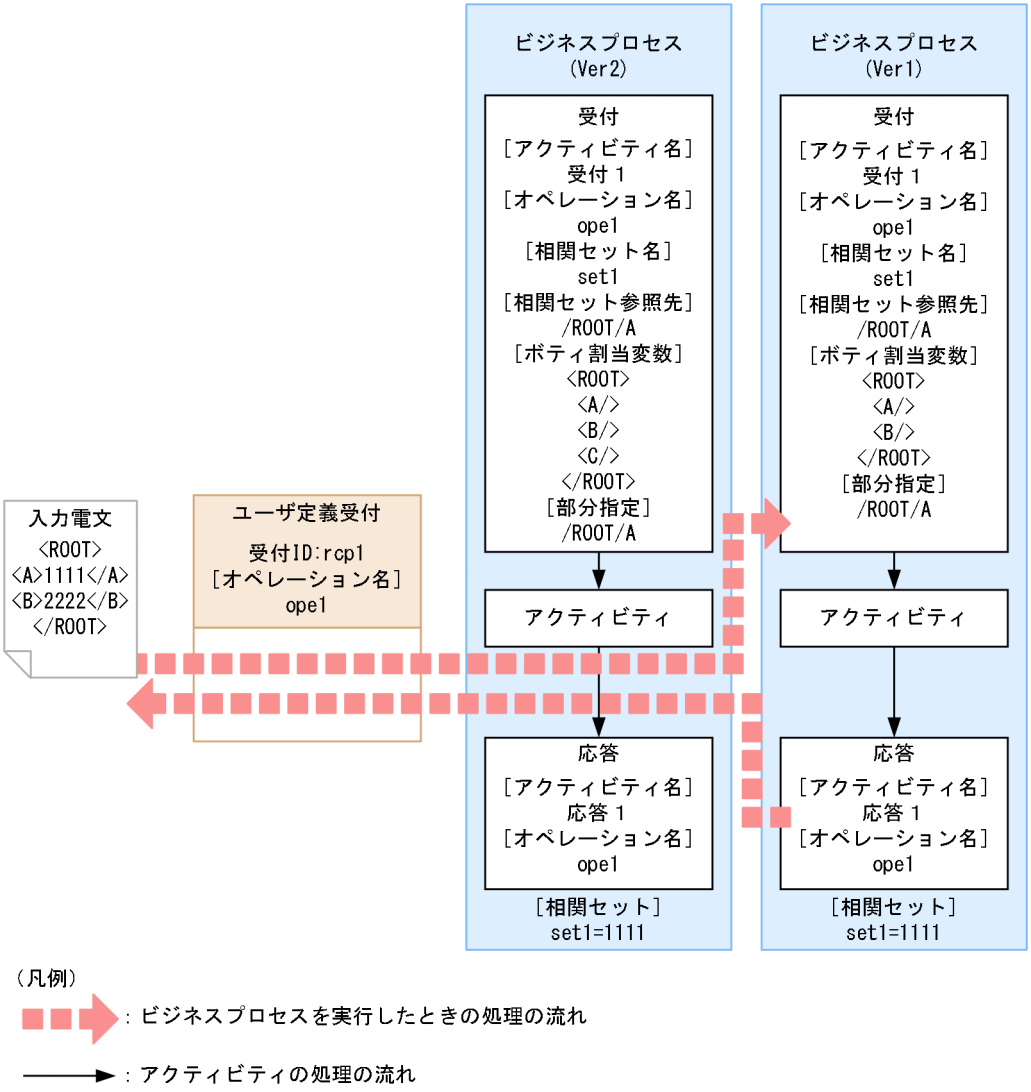
バージョンアップ後にボディ割当変数（部分指定）を変更した場合の動作を次の図に示します。

図 5-34　ボディ割当変数に関わる変更を行う場合（部分指定を変更）



次の図に示すように，バージョンアップ後にボディ割当変数の XML スキーマを変更した場合，相関セットの値が取得できて，最新バージョンと最新以外のバージョンで同一の値が取得できれば最新以外のビジネスプロセスを実行できます。

図 5-35　ボディ割当変数に関わる変更を行う場合（変数の XML スキーマを変更）

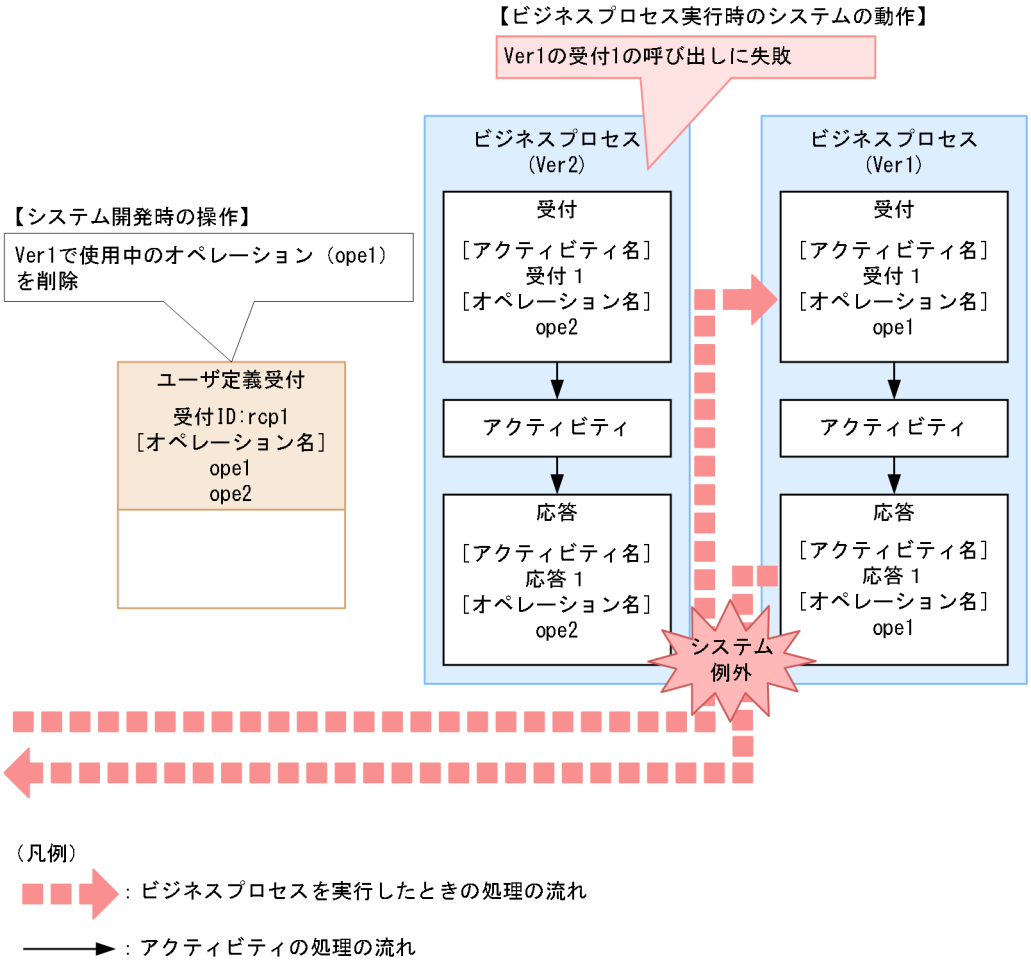


(f)　ユーザ定義受付のオペレーションを削除するときの注意事項

ユーザ定義受付を使用するビジネスプロセスで，バージョンアップ後に最新以外のバージョンのオペレーションを削除する場合，最新以外のバージョンのプロセスを実行できなくなる可能性があるため注意が必要です。

バージョンアップ後にユーザ定義受付のオペレーションを削除した場合の動作を次の図に示します。

図 5-36　ユーザ定義受付のオペレーションを削除した場合



（凡例）

：ビジネスプロセスを実行したときの処理の流れ

：アクティビティの処理の流れ

5.10　ビジネスプロセスのテスト

ビジネスプロセス実行コマンド（csmexecutebp コマンド）を使用することで，任意のユーザ定義受付を呼び出すリクエスタを用意しなくても，ビジネスプロセスにリクエストを送信できます。

ビジネスプロセスの開発時にリクエスト処理を開始できるため，ビジネスプロセスのテストが開発環境で実施できます。

5.10.1　ビジネスプロセス実行コマンド使用時の前提条件

ビジネスプロセス実行コマンドでは，標準の同期受付（Web サービス）を使用してビジネスプロセスを呼び出すため，あらかじめ次の状態にしておく必要があります。

- HCSC サーバの構築時に標準の同期受付（Web サービス）を使用するように定義する。
- Web サーバ，標準の同期受付（Web サービス），および呼び出すビジネスプロセスが起動状態となっている。

5.10.2　リクエストの送信

ビジネスプロセス実行コマンドの実行時に指定された接続先の標準の同期受付（Web サービス）にリクエストを送信します。

ビジネスプロセス実行コマンドの実行時に指定されたビジネスプロセス名にリクエストが送信されます。リクエストの送信の結果として，ビジネスプロセスから返却された応答を受信しますが，ビジネスプロセスから返却されたヘッダ電文を受信することはできません。

なお，ビジネスプロセスの呼び出し時に作業フォルダは生成されません。そのため，ビジネスプロセス内で作業フォルダは使用できません。

ビジネスプロセス実行コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csmexecutebp（ビジネスプロセスの実行）」を参照してください。

(1)　要求電文（ヘッダ）の送信

ビジネスプロセスの呼び出し時にビジネスプロセスに要求電文（ヘッダ）は送信できません。

そのため，ビジネスプロセスの受付アクティビティでヘッダ割当変数を指定している場合，ヘッダ割当変数に指定した変数は初期化されません。

(2) 要求電文（ボディ）の送信

ビジネスプロセス実行コマンドの実行時に要求電文が指定されている場合，要求電文が定義されたファイルを読み込みます。読み込んだファイルの内容は，要求電文（ボディ）としてビジネスプロセスに送信されます。

要求電文を定義したファイルの拡張子が「.xml」の場合は，XML 形式の電文が指定されたと見なされます。拡張子が「.xml」ではない場合，バイナリ形式の電文が指定されたと見なされます。

ビジネスプロセス実行コマンドの実行時に要求電文が指定されていない場合は，ビジネスプロセスの呼び出し時に要求電文は送信されません。

(3) 応答電文の出力

ビジネスプロセスから正常応答を受信した場合，メッセージ「KDEC11411-I」が出力されます。異常応答，またはフォルト応答の場合は，メッセージは出力されません。

メッセージに出力する内容は，ビジネスプロセスから返却された応答電文の形式によって異なります。なお，ビジネスプロセスが応答電文を返却しない場合は，空文字が出力されます。

ビジネスプロセスから返却された応答電文の形式と，メッセージの出力内容を次に示します。

表 5-22　返却された応答電文の形式とメッセージの出力内容

ビジネスプロセスから返却された応答電文の形式	メッセージの出力内容
XML 形式	ビジネスプロセスから返却された応答電文（XML 形式）を文字列に変換したものを出力します。
XML 形式以外	ビジネスプロセスから返却された応答電文（XML 形式以外）を 16 進数表記の文字列に変換したものを出力します。

5.10.3　クライアント相関 ID の設定

ビジネスプロセス実行コマンドの実行時に指定されたクライアント相関 ID（サービスリクエスタからの要求電文を一意に識別するための相関識別子）を，ビジネスプロセスの呼び出し時にクライアント相関 ID として設定します。

ビジネスプロセス実行コマンドの実行時にクライアント相関 ID が設定されていない場合は，リクエストにクライアント相関 ID は設定されません。

クライアント相関 ID は，ビジネスプロセス実行コマンド定義ファイルで設定します。クライアント相関 ID の設定については，マニュアル「サービスプラットフォーム リファレンス」の「3.21.1　ビジネスプロセス実行コマンド定義ファイル」を参照してください。

5.11　ビジネスプロセスの検証

作成したビジネスプロセスの内容が妥当かどうか検証できます。

必要なビジネスプロセス定義がない場合，または定義の関係が正しくない場合は正常に動作しません。そのため，実行環境で実行する前に，すべてのビジネスプロセス定義を検証する必要があります。

作成したビジネスプロセスに必要な項目がそろっているか，それらの関係が正しいかどうかを検証します。必要に応じて任意のタイミングで検証できます。

ビジネスプロセスの検証として，次の検証を行います。

- ビジネスプロセス定義の検証
  ビジネスプロセスの定義内容を検証します。設定項目，ビジネスプロセスが構造化されているか，およびそのほかの検証を行います。
- 呼出サービス部品の検証
  サービス呼出アクティビティで呼び出しているサービス部品との整合性を検証します。
- Java の検証
  Java 呼出アクティビティで使用する Java クラス，および Java クラスから使用するライブラリについて検証します。
- データ変換定義検証
  データ変換アクティビティのデータ変換定義の内容を検証します。
- ユーザ定義受付検証
  検証対象のビジネスプロセスに含まれるユーザ定義受付の定義内容を検証します。

ビジネスプロセス定義，呼出サービス部品，および Java の検証内容については，「5.11.1　検証内容」を参照してください。

ユーザ定義受付の検証の内容については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.14　ユーザ定義受付の検証」を参照してください。

5.11.1　検証内容

ビジネスプロセスの検証では，次の内容を検証します。

- 設定項目の検証
- 定義されたビジネスプロセスが構造化されているかの検証
- そのほかの検証

検証中にエラーが 100 件を超えた場合，検証途中でも検証が中止されます。この場合は，作成したビジネスプロセスを見直してください。

ビジネスプロセスの検証ではエラーにならなくても，ビジネスプロセスを実行したときに初めてエラーとなる場合があります。このような場合，ログを採取し，ログに出力されたメッセージに従って対処します。

なお，ビジネスプロセスの検証は，基本的に定義情報だけの検証であり，ビジネスプロセスの実行時にエラーが発生しないことを保証するものではありません。よって，運用する前に十分なテストを行ってください。

(1) 設定項目の検証

ビジネスプロセス定義画面に記述したビジネスプロセスの各アクティビティに，設定項目が正しく定義されているかどうかが検証されます。

表 5-23 設定項目の検証内容と対処

項番	検証内容	エラーになった場合の対処
1	ビジネスプロセス名の長さが，64 バイト以内か。	次のどちらかの方法で対処します。 • ビジネスプロセス名を 64 バイト以下に変更。 • 使用する文字コードを変更。
2	変数・相関セット一覧ダイアログで定義した変数名の長さが，64 バイト以内か。	次のどちらかの方法で対処します。 • 変数名を 64 バイト以下に変更。 • 使用する文字コードを変更。
3	変数・相関セット一覧ダイアログで定義した変数の型がメッセージ型の場合，変数に対応する電文フォーマットが設定されているか。	変数・相関セット一覧ダイアログで，メッセージ型の変数に対応する電文フォーマットを設定します。
4	変数・相関セット一覧ダイアログで定義した変数の部分名が設定されているか。	変数・相関セット一覧ダイアログで，変数の部分名を設定します。
5	変数・相関セット一覧ダイアログで定義した変数の部分名に，指定式が設定されているか。	変数・相関セット一覧ダイアログで，変数の部分名に指定式を設定します。
6	変数・相関セット一覧ダイアログで定義した変数の部分名に，型が設定されているか。	変数・相関セット一覧ダイアログで，変数の部分名に型を設定します。
7	変数・相関セット一覧ダイアログで定義した相関セット名の長さの長さが 64 バイト以内か。	次のどちらかの方法で対処します。 • 相関セット名を 64 バイト以下に変更。 • 使用する文字コードを変更。
8	変数・相関セット一覧ダイアログで定義した相関セットに，部分名が 1 つ以上設定されているか。	変数・相関セット一覧ダイアログで，相関セットに，部分名を 1 つ以上設定します。
9	変数・相関セット一覧ダイアログで定義した相関セットに，利用変数名が設定されているか。	変数・相関セット一覧ダイアログで，相関セットに利用変数を設定します。
10	変数・相関セット一覧ダイアログで定義した相関セットに，有効な変数名が設定されているか。	変数・相関セット一覧ダイアログで，有効な変数名を設定します。
11	変数・相関セット一覧ダイアログで定義した相関セットで指定されている変数に，基本型の部分があるか。	変数・相関セット一覧ダイアログで，基本型の部分を持つ変数を設定します。

項番	検証内容	エラーになった場合の対処
12	変数・相関セット一覧ダイアログで定義した相関セットに，部分名が設定されているか。	変数・相関セット一覧ダイアログで，相関セットに部分名を設定します。
13	変数・相関セット一覧ダイアログで定義した相関セットに，有効な部分名が設定されているか。	変数・相関セット一覧ダイアログで，有効な部分名を設定します。
14	アクティビティのフォルト処理で，有効な変数が設定されているか。	フォルト処理の割当ダイアログで，有効な変数を設定します。
15	アクティビティのフォルト処理で，遷移先が設定されているか。	フォルト処理の割当ダイアログで，遷移先を設定します。
16	アクティビティのフォルト処理で，割当変数が設定されているか。	フォルト処理の割当ダイアログで，割当変数を設定します。
17	アクティビティのフォルト処理で，遷移先が設定されているか。	フォルト処理の割当ダイアログで，遷移先を設定します。
18	受付アクティビティで，オペレーション名が設定されているか。	受付アクティビティで，オペレーション名を設定します。
19	アクティビティ名の長さが 64 バイト以内か。	次のどちらかの方法で対処します。 • アクティビティ名を 64 バイト以下に変更。 • 使用する文字コードを変更。
20	受付アクティビティで，有効なヘッダ割当変数が設定されているか。	受付アクティビティで，有効なヘッダ割当変数を設定します。
21	受付アクティビティで，有効なルート要素および名前空間が設定されているか。	受付アクティビティで，有効なルート要素および名前空間を設定します。
22	受付アクティビティで，有効なボディ割当変数が設定されているか。	受付アクティビティで，有効なボディ割当変数を設定します。
23	受付アクティビティで，有効な割当相関セットが設定されているか。	受付アクティビティで，有効な相関セットを設定します。
24	リンクコネクションで，リンク名が指定されているか。	リンクコネクションに，リンク名を設定します。
25	リンクコネクションで，リンク先の遷移条件が「あり」に設定されている場合に，XPath が設定されているか。	リンクコネクションについて，次のどちらかの方法で対処します。 • 遷移条件を設定。 • 遷移条件に「なし」を選択。
26	応答アクティビティで，オペレーション名が設定されているか。	応答アクティビティで，オペレーション名を設定します。
27	応答アクティビティで，有効なヘッダ割当変数が設定されているか。	応答アクティビティで，有効なヘッダ割当変数を設定します。
28	応答アクティビティで，有効なルート要素および名前空間が設定されているか。	応答アクティビティで，有効なルート要素および名前空間を設定します。

項番	検証内容	エラーになった場合の対処
29	応答アクティビティで，有効なボディ割当変数が設定されているか。	応答アクティビティで，有効なボディ割当変数を設定します。
30	応答アクティビティで，有効な割当相関セットが設定されているか。	応答アクティビティで，有効な相関セットを設定します。
31	応答アクティビティで，ボディ割当変数が指定されているか。	応答アクティビティで，ボディ割当変数を設定します。
32	サービス呼出アクティビティで，呼び出すサービス名が設定されているか。	サービス呼出アクティビティで，呼び出すサービス名を設定します。
33	サービス呼出アクティビティで，有効な要求電文用のヘッダ割当変数が設定されているか。	サービス呼出アクティビティで，有効な要求電文用のヘッダ割当変数を設定します。
34	サービス呼出アクティビティで，有効な要求電文用のルート要素および名前空間が設定されているか。	サービス呼出アクティビティで，有効な要求電文用のルート要素および名前空間を設定します。
35	サービス呼出アクティビティで，有効な応答電文用のヘッダ割当変数が設定されているか。	サービス呼出アクティビティで，有効な応答電文用のヘッダ割当変数を設定します。
36	サービス呼出アクティビティで，有効な応答電文用のルート要素および名前空間が設定されているか。	サービス呼出アクティビティで，有効な応答電文用のルート要素および名前空間を設定します。
37	サービス呼出アクティビティで，有効な要求電文用のボディ割当変数が設定されているか。	サービス呼出アクティビティで，有効な要求電文用のボディ割当変数を設定します。
38	サービス呼出アクティビティで，有効な応答電文用のボディ割当変数が設定されているか。	サービス呼出アクティビティで，有効な応答電文用のボディ割当変数を設定します。
39	サービス呼出アクティビティで，同期呼び出しの場合に，応答電文用のボディ割当変数が設定されているか。	サービス呼出アクティビティで，同期呼び出しの場合，応答電文用のボディ割当変数を設定します。
40	サービス呼出アクティビティで，有効な相関セットが設定されているか。	サービス呼出アクティビティで，有効な相関セットを設定します。
41	代入アクティビティで，1 つ以上の代入操作が設定されているか。	代入アクティビティで，1 つ以上の代入操作を設定します。
42	代入アクティビティで，コピー元が設定されているか。	代入アクティビティで，コピー元を設定します。
43	代入アクティビティで，コピー先が設定されているか。	代入アクティビティで，コピー先を設定します。
44	代入アクティビティで，有効なコピー元変数が設定されているか。	代入アクティビティで，有効なコピー元変数を設定します。
45	代入アクティビティで，コピー元に有効な部分名が設定されているか。	代入アクティビティで，コピー元に有効な部分名を設定します。
46	データ変換アクティビティで，変換元変数が設定されているか。	データ変換アクティビティで，変換元変数を設定します。
47	データ変換アクティビティで，有効な変換元変数が設定されているか。	データ変換アクティビティで，有効な変換元変数を設定します。
48	データ変換アクティビティで，データ変換定義が設定されているか。	データ変換アクティビティで，データ変換定義を設定します。

項番	検証内容	エラーになった場合の対処
49	データ変換アクティビティのデータ変換定義が正しいか。	検証で出力されたメッセージを参考にして，正しい定義に修正します。
50	代入アクティビティで，コピー元のタイプとして，変数と式のどちらかが選択されているか。	代入アクティビティで，コピー元のタイプとして，変数と式のどちらかを設定します。
51	代入アクティビティで，コピー元の変数が選択されている場合，変数名が設定されているか。	代入アクティビティで，コピー元の変数を設定します。
52	代入アクティビティで，コピー元の式が選択されている場合，式が設定されているか。	代入アクティビティで，コピー元の式を設定します。
53	代入アクティビティで，コピー先の変数名が設定されている場合，有効な変数が設定されているか。	代入アクティビティで，有効なコピー先変数を設定します。
54	データ変換アクティビティで，有効な変換先変数が設定されているか。	データ変換アクティビティで，有効な変換先変数を設定します。
55	代入アクティビティで，コピー先に有効な部分名が設定されているか。	代入アクティビティで，コピー先に有効な部分名を設定します。
56	代入アクティビティのコピー先，またはデータ変換アクティビティの変換先のどちらかで，変数が設定されているか。	次のどちらかの方法で対処します。 • 代入アクティビティで，コピー先の変数を設定します。 • データ変換アクティビティで，変換先の変数を設定します。
57	並列処理開始アクティビティで，設定するリンク名の長さが 64 バイト以内か。	並列処理開始アクティビティで，定義されている，リンク名を次のどちらかの方法で対処します。 • リンク名を 64 バイト以下に変更。 • 使用する文字コードを変更。
58	分岐アクティビティで，分岐条件を 1 つ以上設定しているか。	分岐アクティビティで，1 つ以上の分岐条件を設定します。
59	分岐アクティビティで，条件分岐またはデフォルトに割り当てられていない遷移先があるか。	分岐アクティビティで，遷移先を条件分岐またはデフォルトに割り当てます。
60	分岐アクティビティで，条件名が設定されているか。	分岐アクティビティで，分岐の条件名を設定します。
61	分岐アクティビティで，条件が設定されているか。	分岐アクティビティで，分岐の条件を設定します。
62	繰り返しアクティビティで，繰り返し条件が設定されているか（条件指定方式の場合だけ）。	繰り返しアクティビティで，繰り返し条件を設定します。
63	繰り返しアクティビティで，繰り返しリストが指定されているか（リスト指定方式の場合だけ）。	繰り返しアクティビティで，繰り返しリストを指定します。
64	繰り返しアクティビティで，繰り返し要素変数が設定されているか（リスト指定方式の場合だけ）。	繰り返しアクティビティで，繰り返し要素変数を設定します。
65	繰り返しアクティビティで，繰り返し要素変数に有効な変数名が設定されているか（リスト指定方式の場合だけ）。	繰り返しアクティビティで，有効な繰り返し要素変数を設定します。

項番	検証内容	エラーになった場合の対処
66	Java 呼出アクティビティで，Java クラス名が設定されているか。	Java 呼出アクティビティで，Java クラス名を設定します。
67	Java 呼出アクティビティで，有効な引数用割当変数が設定されているか。	Java 呼出アクティビティで，有効な引数用割当変数を設定します。
68	Java 呼出アクティビティで，有効な戻り値用割当変数が設定されているか。	Java 呼出アクティビティで，有効な戻り値用割当変数を設定します。
69	フォルト送出アクティビティで，有効な割当変数が設定されているか。	フォルト送出アクティビティで，有効な割当変数を設定します。
70	待機アクティビティで，待機時間が設定されているか。	待機アクティビティで，待機時間を設定します。
71	HCSCTE プロジェクトの lib ディレクトリにディレクトリがないか。	HCSCTE プロジェクトの lib ディレクトリからディレクトリを取り除きます。
72	HCSCTE プロジェクトの lib ディレクトリに次の名前のファイルがないか。 • csbdef.jar • cscbp_ejb.jar • csbjava.jar	ファイルを削除するか，ファイル名を変更し，HCSCTE プロジェクトの lib ディレクトリに左記のファイルが含まれないようにします。
73	検証アクティビティで，検証する変数が設定されているか。※	検証アクティビティで，検証する変数を設定します。
74	検証アクティビティで，検証する変数に有効な変数名が設定されているか。	検証アクティビティで，検証する変数に有効な変数名を設定します。
75	補償アクティビティダイアログで「1 つのみ」を選択した場合に，コンボボックスで補償するアクティビティを選択しているか。	補償アクティビティダイアログで，補償するアクティビティをコンボボックスから 1 つ設定します。
76	補償アクティビティダイアログで「1 つのみ」を選択した場合に，コンボボックスで設定した補償するアクティビティが存在するか。	補償アクティビティダイアログで，存在するアクティビティをコンボボックスから選択します。
77	ビジネスプロセスのプロパティの「スコープアクティビティのデフォルトハンドラ」が OFF の場合に，コンボボックスで設定した補償するアクティビティに補償ハンドラが定義されているか。	補償アクティビティダイアログのコンボボックスで設定した補償するアクティビティに，補償ハンドラを定義します。
78	ビジネスプロセスのプロパティの「スコープアクティビティのデフォルトハンドラ」が ON の場合に，コンボボックスで設定した補償するアクティビティは次のどちらかか。 • 補償するアクティビティに補償ハンドラが定義されている • 補償するアクティビティはスコープアクティビティである	補償アクティビティダイアログのコンボボックスで設定した補償するアクティビティは，次のどちらかとなるようにします。 • 補償するアクティビティに補償ハンドラを定義する • 補償するアクティビティには，スコープアクティビティを選択する
79	ログ出力アクティビティダイアログで，ログ出力識別子およびリソース Key の値が設定されているか。	ログ出力アクティビティダイアログで，ログ出力識別子およびリソース Key の値を設定します。
80	ログ出力アクティビティダイアログで，埋め込み要素に変数を設定している場合，有効な変数名か。	ログ出力アクティビティダイアログの埋め込み要素には，有効な変数名を指定します。

項番	検証内容	エラーになった場合の対処
81	ログ出力アクティビティダイアログで，埋め込み要素に有効な変数を設定しており，かつ部分名も設定している場合，その部分名は有効か。	ログ出力アクティビティダイアログの埋め込み要素には，有効な部分名を指定します。

注※

検証・パッケージングの動作設定ダイアログで，検証の実施有無，または検証メッセージの種類を変更できます。

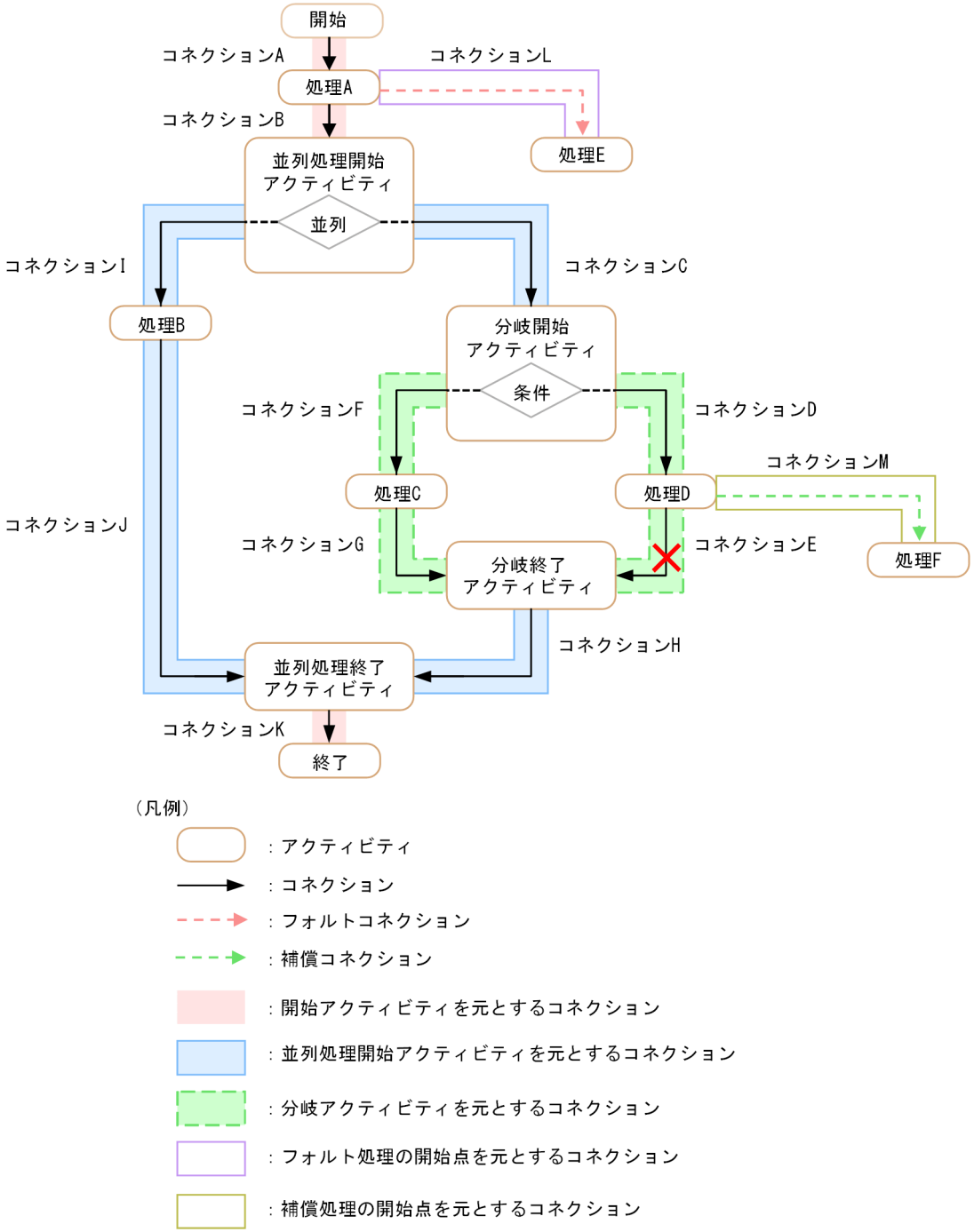
(2) 定義されたビジネスプロセスが構造化されているかの検証

ビジネスプロセスが，コネクションを使用して構造化されているかどうかが検証されます。

検証でエラーになった場合，エラーとなったコネクションの元のコネクションもエラーと見なされます。

例えば，次の図のように，コネクション E がなかった場合は，分岐アクティビティや並列アクティビティなどに正常なコネクションが含まれていても，分岐アクティビティを元とするコネクション，並列アクティビティを元とするコネクション，および開始アクティビティを元とするコネクションすべてがエラーと見なされます。ただし，処理 D がフォルト送出アクティビティまたは途中終了アクティビティの場合，コネクション E は設定できないためエラーにはなりません。

図 5-37　コネクションの例



定義されたビジネスプロセスの構造化の検証内容と対処を次の表に示します。

表 5-24　定義されたビジネスプロセスの構造化の検証内容と対処

項番	分類	検証内容	エラーになった場合の対処
1	次の接続先	遷移元が開始アクティビティの場合，次の接続先があるか。	開始アクティビティを先頭にして，接続移先のアクティビティを設定します。

項番	分類	検証内容	エラーになった場合の対処
2	次の接続先	遷移元が並列処理開始アクティビティの場合，次の接続先があるか。	並列処理開始アクティビティ，およびリンクコネクションで，接続するアクティビティを設定します。
3		遷移元が分岐開始アクティビティの場合，次の接続先があるか。	分岐開始アクティビティで，接続するアクティビティを設定します。
4		遷移元が上記以外のアクティビティの場合，次の接続先があるか。	遷移元のアクティビティに接続するアクティビティを設定します。
5	並列処理	並列処理が分岐終了アクティビティに接続していないか。	並列処理，対応する並列処理終了アクティビティに接続します。
6		次に示すコネクションが，並列処理終了アクティビティに接続していないか。 • 開始アクティビティを元とするコネクション※ • あるアクティビティのあるフォルト処理の開始点を元とするコネクション※ • あるアクティビティのある補償処理の開始点を元とするコネクション※	次のどちらかの方法で対処します。 • 並列処理終了アクティビティを削除。 • 並列処理終了アクティビティを，別のアクティビティに変更。
7		並列処理開始アクティビティからつながる並列処理終了アクティビティが複数ないか。	並列処理開始アクティビティを，対応する 1 つの並列処理終了アクティビティに接続します。
8		並列処理終了アクティビティが，対応する並列処理以外からのコネクションも受け付けていないか。	対応する並列処理を並列処理終了アクティビティに接続します。
9		並列処理が終了アクティビティに接続していないか。	並列処理を並列処理終了アクティビティに接続します。
10		並列処理開始アクティビティから直接，並列処理終了アクティビティに接続していないか。	並列処理開始アクティビティと，対応する並列処理終了アクティビティの間には，1 つ以上のアクティビティを設定します。
11	分岐処理	分岐処理が並列処理終了アクティビティに接続していないか。	分岐処理を対応する分岐終了アクティビティに接続します。
12		次に示すコネクションが，分岐処理終了アクティビティに接続していないか。 • 開始アクティビティを元とするコネクション※ • あるアクティビティのあるフォルト処理の開始点を元とするコネクション※ • あるアクティビティのある補償処理の開始点を元とするコネクション※	次のどちらかの方法で対処します。 • 分岐終了アクティビティを削除。 • 分岐終了アクティビティを，別のアクティビティに変更。
13		分岐開始アクティビティからつながる分岐終了アクティビティが複数ないか。	分岐開始アクティビティを，対応する 1 つの分岐終了アクティビティに接続します。

項番	分類	検証内容	エラーになった場合の対処
14	分岐処理	分岐終了アクティビティが，対応する分岐処理以外からのコネクションも受け付けていないか。	対応する分岐処理を分岐終了アクティビティに接続します。
15		分岐処理が終了アクティビティに接続していないか。	分岐処理を分岐終了アクティビティに接続します。
16		分岐開始アクティビティから直接，分岐終了アクティビティに接続していないか。	分岐アクティビティと，対応する分岐終了アクティビティの間には，1 つ以上のアクティビティを設定します。
17	フォルト処理	あるアクティビティのあるフォルト処理の開始点を元とするコネクションを処理中に，終了となっていないか。	フォルト処理の接続先から，終了アクティビティを削除します。
18	補償処理	あるアクティビティのある補償処理の開始点を元とするコネクションを処理中に，終了となっていないか。	補償処理の接続先から，終了アクティビティを削除します。

注※

アクティビティを元とするコネクション，あるアクティビティのあるフォルト処理の開始点を元とするコネクション，およびあるアクティビティのある補償処理の開始点を元とするコネクションについては，「図 5-37 コネクションの例」を参照してください。

(3)　そのほかの検証

そのほかに次の表に示す内容が検証されます。

表 5-25　そのほかの検証内容と対処

項番	分類	検証内容	エラーになった場合の対処
1	ビジネスプロセス共通	ビジネスプロセスの最初に実行される基本アクティビティが，インスタンスを生成する受付アクティビティであるか。	開始アクティビティの次に，インスタンスを生成する受付アクティビティを接続します。
2		リンクコネクションに有効なリンク名が設定されているか。	リンク元の並列処理開始アクティビティで，リンク名を設定します。
3		代入アクティビティのコピー元とコピー先の変数の型が等しいか。	代入アクティビティで，コピー先とコピー元の変数の型を同じに設定します。
4		複数の受付アクティビティで同一のオペレーション名が使用されていないか。	各受付アクティビティでオペレーション名が一意になるように設定します。
5		同一のオペレーション名を持つ応答アクティビティの変数の型が同一か。	同一のオペレーション名を持つ応答アクティビティで，同じ割当変数名を設定します。
6		同一オペレーションの応答アクティビティのフォルト名に対応付けられている変数が 1 つか。	フォルト処理の割当ダイアログで，同一オペレーションの応答アクティビティのフォルト名に対応する割当変数を 1 つに設定します。

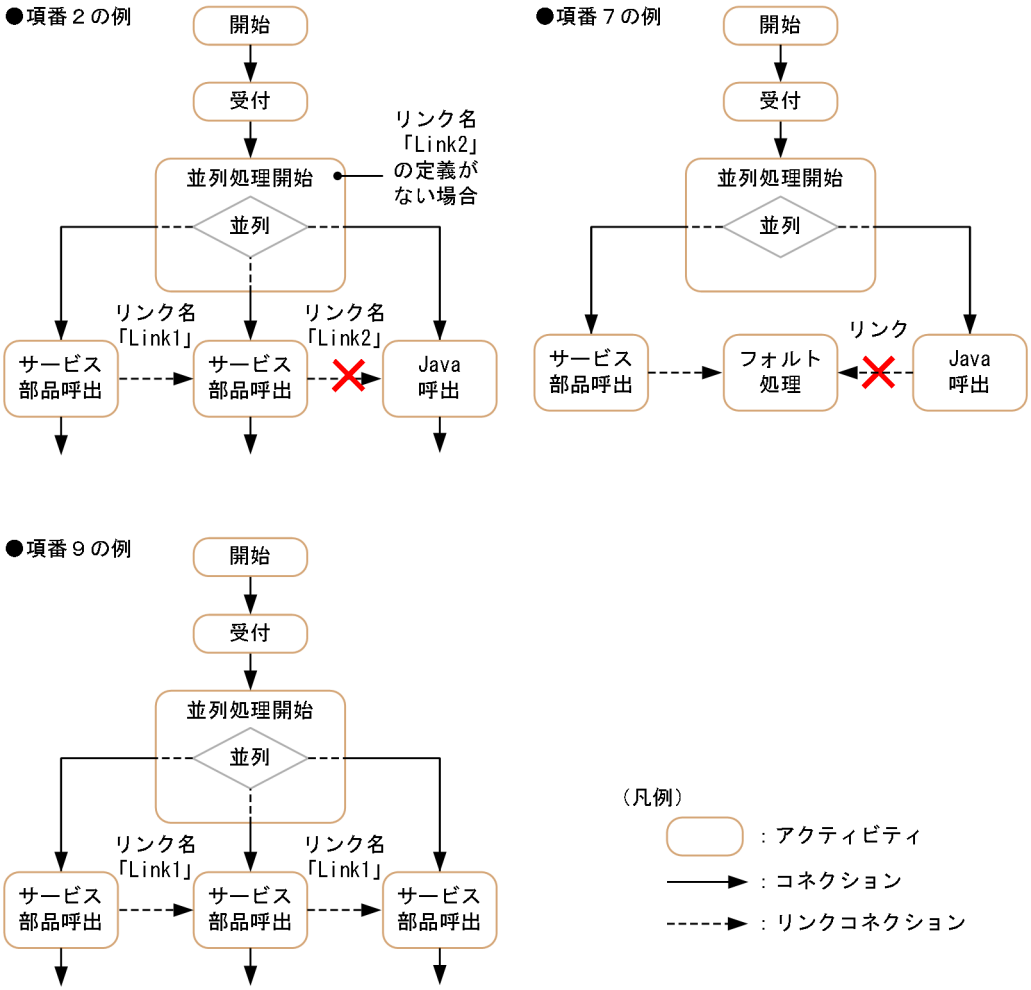
項番	分類	検証内容	エラーになった場合の対処
6	ビジネスプロセス共通	(正しくない例) • フォルト名 faultA に変数 variableX と variableY の 2 つが対応している場合 (正しい例) • フォルト名 faultA に変数 variableX，フォルト名 faultB に変数 variableY が対応している場合 • フォルト名 faultA に変数 variableX，フォルト名 faultB に変数 variableX が対応している場合	フォルト処理の割当ダイアログで，同一オペレーションの応答アクティビティのフォルト名に対応する割当変数を 1 つに設定します。
7		循環するリンクコネクションが設定されていないか。	リンクコネクションが循環しないように設定します。
8		フォルト処理外からフォルト処理内へのリンクコネクションが張られていないか。	フォルト処理からフォルト処理内へのリンクコネクションを削除するよう設定します。
9		リンク名が複数回利用されていないか。	リンクコネクションで，リンク名がビジネスプロセス内で一意になるよう変更します。
10		同期の受付アクティビティのオペレーションに対応する応答アクティビティがあるか。	同期の受付アクティビティのオペレーションに対応する応答アクティビティを作成します。
11		応答アクティビティのオペレーションに対応する，同期の受付アクティビティがあるか。	次のどちらかの方法で対処します。 • 応答アクティビティのオペレーションに対応する同期の受付アクティビティを作成します。 • 応答アクティビティのオペレーション名を変更します。
12		インスタンスを生成するすべての受付アクティビティで，使用されている相関セットが同じか。	インスタンスを生成するすべての受付アクティビティで，使用する相関セットが同じになるように設定します。
13		インスタンスを生成する受付アクティビティがあるか。	次のどちらかの方法で対処します。 • 受付アクティビティで，インスタンス生成を「yes」に変更します。 • インスタンスを生成する受付アクティビティを作成します。
14		ビジネスプロセスの開始点からコネクションを追っても到達できないアクティビティがあるか。	次のどちらかの方法で対処します。 • 到達できないアクティビティを削除します。 • 到達できるようにコネクションを設定します。
15		フォルト送出アクティビティから送出されたフォルト変数が，上位のスコープのフォルト処理でキャッチされるか。	次のどちらかの方法で対処します。 • フォルト処理に適切な割当変数を設定します。 • フォルト処理に割当変数として「catch-all」を設定します。

項番	分類	検証内容	エラーになった場合の対処
16	ビジネスプロセス共通	複数の並列処理開始アクティビティで同一のリンク名が定義されていないか。	ビジネスプロセス内で一意になるようにリンク名を設定します。
17		変数の部分が次の条件を満たしているか。 • 部分の型が XML 型の場合，部分名はビジネスプロセス内で一意であること • 部分の型が XML 型以外（numeric，string，または boolean）の場合，電文フォーマットが同じ変数の間で部分名が一意であること • 部分名が同じ場合は部分の型も同じであること	検証内容に示す条件をすべて満たすように変数の部分を設定します。
18		［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ内に，不正なアクティビティを定義していないか。	［このスコープの開始時と終了時のみコミットする］ラジオボタンを選択したスコープアクティビティ内に定義された次のアクティビティを削除します。 • 受付アクティビティ • 応答アクティビティ • 待機アクティビティ
19		補償アクティビティが次の条件を満たしているか。 • 補償アクティビティの定義位置が，あるスコープの補償ハンドラまたはフォルトハンドラ内にあること • ビジネスプロセス定義画面のプロパティービューで［スコープアクティビティのデフォルトハンドラ］が OFF の場合，かつ補償アクティビティダイアログで［すべて］ラジオボタンを選択しているとき，補償ハンドラの定義されたアクティビティが補償範囲内に存在すること • ビジネスプロセス定義画面のプロパティービューで［スコープアクティビティのデフォルトハンドラ］が ON の場合，かつ補償アクティビティダイアログで［すべて］ラジオボタンを選択しているとき，補償範囲内に補償ハンドラが定義されたアクティビティ，またはスコープアクティビティが存在すること • 補償アクティビティダイアログで［1つのみ］ラジオボタンを選択している場合，補償するアクティビティが補償範囲内に存在すること	検証内容に示す条件をすべて満たすように補償アクティビティを設定します。
20		ビジネスプロセス定義画面のプロパティービューで［スコープアクティビティのデ	次のどれかの方法で対処します。 • 補償ハンドラを削除します。

項番	分類	検証内容	エラーになった場合の対処
20	ビジネスプロセス共通	フォルトハンドラ］が OFF の場合，あるアクティビティに定義されている補償ハンドラが，補償アクティビティのどれかによって起動できるか。	• 補償ハンドラが起動できるよう補償アクティビティを設定します。 • ビジネスプロセス定義画面のプロパティービューで［スコープアクティビティのデフォルトハンドラ］を ON にします。
21		ビジネスプロセス定義画面のプロパティービューで［スコープアクティビティのデフォルトハンドラ］が ON の場合，あるアクティビティに定義されている補償ハンドラが，補償アクティビティのどれかによって起動できるか。 注 起動させる補償アクティビティには，スコープアクティビティのデフォルトハンドラによって暗黙的に設定される補償アクティビティも含む。	次のどちらかの方法で対処します。 • 補償ハンドラを削除します。 • 補償ハンドラが起動できるよう補償アクティビティを設定します。
22		リンクコネクションが次のどちらかのパターンで設定されていないか。 • リンクコネクションが，補償処理内から補償処理外へ設定されている。 • リンクコネクションが，補償処理外から補償処理内へ設定されている。	リンクコネクションは，補償処理の内外をまたがないよう設定してください。
23	永続化しないビジネスプロセスの場合	永続化しないビジネスプロセスで，受付アクティビティが 1 個だけか。	ビジネスプロセス全体の受付アクティビティを調べ，2 個以上ある場合は削除します。
24		永続化しないビジネスプロセスに，非同期のサービス呼出アクティビティがないか。	次のどちらかの方法で対処します。 • 非同期のサービス呼出アクティビティを同期に変更します。 • 非同期のサービス呼出アクティビティを削除します。
25		永続化しないビジネスプロセスに，待機アクティビティがないか。	待機アクティビティを削除します。

表に示した検証の対象となるリンクの状態を次の図に示します。

図 5-38　検証の対象となるリンクの状態



(4)　呼出サービス部品の検証

サービス呼出アクティビティに割り当てられているサービス部品のサービス名およびオペレーション名について検証されます。

呼出サービス部品の検証内容と対処を次の表に示します。

表 5-26　呼出サービス部品の検証内容と対処

項番	検証内容	エラーになった場合の対処
1	サービス呼出アクティビティに割り当てられているサービス名がリポジトリ内に実在するか。	サービス呼出アクティビティに，リポジトリ内に実在するサービス名を割り当ててください。
2	サービス呼出アクティビティに割り当てられているサービス名がリポジトリ内に実在する場合，アクティビティに割り当てられているオペレーション名が呼出先サービス部品に存在するか。	サービス呼出アクティビティに，リポジトリ内に実在するオペレーション名を割り当ててください。
3	サービス呼出アクティビティに割り当てられているサービス名およびオペレーション名がリポジトリ内に実在する場合，アクティビティに設定されている通信モデルが呼出先のサービス部品およびオペレーションの通信モデルと一致するかどうか。	サービス呼出アクティビティで，適切なサービス部品およびオペレーションを選択し直してください。

(5) Java の検証

Java 呼出アクティビティで設定されている Java クラスについて検証されます。

Java の検証内容と対処を次の表に示します。

表 5-27 Java の検証内容と対処

項番	検証内容	エラーになった場合の対処
1	Java 呼出アクティビティの「Java クラス名」に指定されているクラスが，次の個所に存在するか。※1 • HCSCTE プロジェクトの classes ディレクトリ • HCSCTE プロジェクトの lib 内の JAR ファイル内	次のどちらかの対処を実施してください。 • 「Java エディタ起動」から Java クラスを作成します。 • HCSCTE プロジェクトの lib ディレクトリに，Java クラスを含む JAR ファイルを指定します。 • Java 呼出アクティビティに HCSCTE プロジェクトの classes ディレクトリ内に存在するクラスを指定します。
2	Java 呼出アクティビティの「Java クラス名」に指定されているクラスが，次の個所に 1 つだけ存在するか。※1 • HCSCTE プロジェクトの classes ディレクトリ • HCSCTE プロジェクトの lib 内の JAR ファイル内	同一の Java クラスを 1 つだけにしてください。
3	Java 呼出アクティビティの「Java クラス名」に指定されているクラスが，専用インターフェース（jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM，または jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterface）を実装しているか。※1	Java 呼出アクティビティに設定する Java クラスは，専用インターフェースを実装してください。
4	項番 3 の検証に必要なクラス，またはインターフェースが存在するか。※1	次のどちらかの対処を実施してください。 • エラーの原因となっている Java クラスまたはインターフェースを作成します。 • HCSCTE プロジェクトの lib ディレクトリに，エラーの原因となっている Java クラスまたはインターフェースを含む JAR ファイルを指定します。
5	Java 呼出アクティビティの［引数用割当変数］に 2 個以上の変数が設定されていて，かつ Java 呼出アクティビティの［Java クラス名］に指定されているクラスが jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM を実装しているか。	Java 呼出アクティビティに設定する Java クラスは，jp.co.Hitachi.soft.csc.bp.receiver.ejb.CustomClassInterfaceM を実装してください。
6	HCSCTE プロジェクトの src ディレクトリに存在する Java クラス※2 で，コンパイルエラーが検出されていないか。	Java クラスのコンパイルエラーを解消してください。 ビルドパスに cscbp_ejb.jar がない場合は<インストールディレクトリ>¥CSC¥lib¥cscbp_ejb.jar をビルドパスに追加してください。

注※1

検証・パッケージングの動作設定ダイアログで，検証の実施有無，または検証メッセージの種類を変更できます。

注※2

検証・パッケージングの動作設定ダイアログで，検証する Java クラスの指定方法を 09-70 以前のバージョンの動作に変更できます。

(6) 検証できない内容

ビジネスプロセスの検証ではエラーにならなくても，ビジネスプロセスを実行したときに初めてエラーとなる場合があります。このような場合，ログを採取し，ログに出力されたメッセージに従って対処します。ログの出力先については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.3　アプリケーションサーバのログ（J2EE アプリケーションを実行するシステム）」を参照してください。

ビジネスプロセスの検証でエラーにならないが，ビジネスプロセスを実行したときにエラーになる例を次に示します。

- アクティビティが生成したインスタンスの数が 2,147,483,648 を超えた。
- サービスリクエスタで指定したオペレーション名の受付アクティビティに到達できない。
- サービスリクエスタで指定したオペレーション名の受付アクティビティが待ち受け状態となっている。
- サービスリクエスタで指定したオペレーション名の受付アクティビティに対応する応答アクティビティに到達できない。
- プロセスインスタンスを生成する受付アクティビティの前に，受付アクティビティ以外の基本アクティビティを実行した。
- サービスリクエスタ側で応答を待っていない状態で，応答アクティビティが実行された。
- サービスリクエスタで指定した受付アクティビティのオペレーション名と，サービスリクエスタに応答として返す応答アクティビティのオペレーション名が一致しない。
- 定められた回数を超えて繰り返しを実行した。

5.11.2　検証方法

検証する手順を次に示します。

1. ツリービューのサービス定義一覧で，右クリックします。
   サービス一覧のポップアップメニューが表示されます。
2. ポップアップメニューから「検証」を選択します。
   検証結果がコンソールビューに表示されます。

また，パッケージングを実施した場合も，自動的に検証されます。

なお，検証する前にビジネスプロセスおよびユーザ定義受付が保存されていないときはリソース保管ダイアログが表示され，定義を保存できます。複数のビジネスプロセスおよびユーザ定義受付が編集中の場合

は，複数回ダイアログが表示されます。保存する場合は［はい］ボタンをクリックします。これ以降，確認ダイアログを表示しないで，すべて保存する場合は［すべてはい］ボタンをクリックします。

5.11.3　検証内容の表示

検証結果として，メッセージがコンソールビューに表示されます。メッセージに従って，修正します。

表示されるメッセージの種類を次の表に示します。

表 5-28　メッセージの種類

種類	説明
エラー	次のどちらかの場合に表示されます。 • 定義の内容に誤りがある。 • 正しい構文で定義しているが，実行できない定義をしている。
警告	実行時にエラーとなる可能性がある定義をしている場合に表示されます。
情報	追加情報が表示されます。

5.12　ビジネスプロセスの削除

ビジネスプロセスをバージョンアップする場合などに，ビジネスプロセスを削除できます。

定義したビジネスプロセスは削除できます。すでに配備定義されている場合，ビジネスプロセスは削除できません。

削除する手順を次に示します。次に示す 2 つの方法があります。

方法 1

1. ツリービューのサービス定義一覧で，ビジネスプロセスを選択します。
2. [Delete] キーを押します。
   削除を確認するダイアログボックスが表示されます。
3. [はい] ボタンをクリックします。
   指定したビジネスプロセスが削除されます。

方法 2

1. ツリービューのサービス定義一覧で，ビジネスプロセスを選択して右クリックします。
   サービス一覧のポップアップメニューが表示されます。
2. ポップアップメニューから「削除」を選択します。
   削除を確認するダイアログボックスが表示されます。
3. [はい] ボタンをクリックします。
   指定したビジネスプロセスが削除されます。

注意事項

バージョンアップ時，最新のビジネスプロセスだけを削除することはできません。すべてのバージョンのビジネスプロセスが削除されます。

6 サービス定義のインポート・エクスポート

サービス定義のインポートおよびエクスポートについて説明します。

6.1　サービス定義のインポート・エクスポートの概要

サービス定義のインポートおよびエクスポートの概要について説明します。

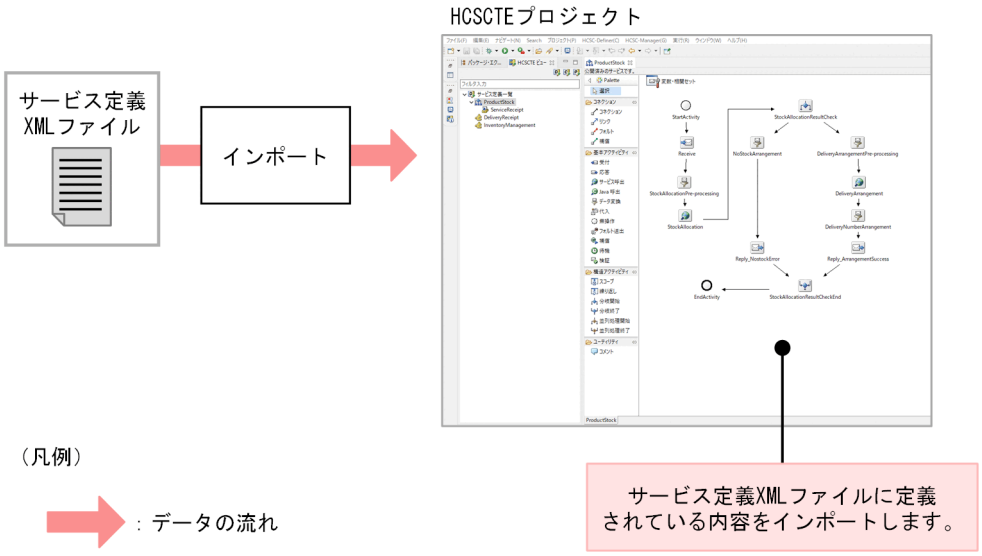
- サービス定義一覧の右クリックメニューの［サービス定義のインポート］を実行すると，HCSC コンポーネントの内容を定義した XML ファイルを，HCSCTE プロジェクトにインポートできます。この機能を，単にサービス定義のインポート機能と呼ぶ場合があります。
- サービス定義一覧の右クリックメニューの［サービス定義のエクスポート］を実行すると，HCSCTE プロジェクトで作成した HCSC コンポーネントを，XML ファイルにエクスポートできます。この機能を，単にサービス定義のエクスポート機能と呼ぶ場合があります。

また，この章では，HCSC コンポーネントの内容を定義した XML ファイルをサービス定義 XML ファイルと呼びます。

> **参考**
>
> ZIP ファイルにしたリポジトリのインポート・エクスポートについては，「3.2　リポジトリの管理」を参照してください。

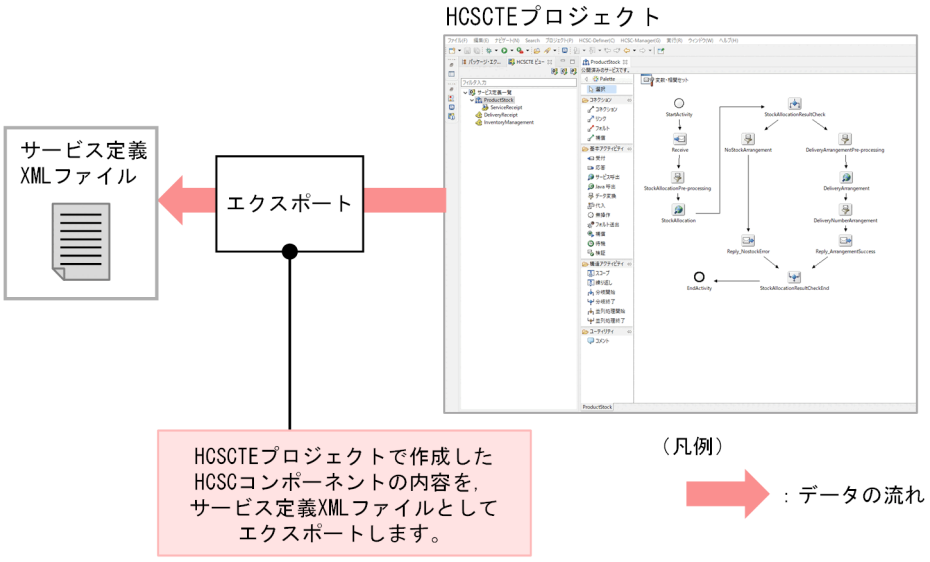
サービス定義 XML ファイルを HCSCTE プロジェクトにインポートする流れを次に示します。

図 6-1　サービス定義 XML ファイルを HCSCTE プロジェクトにインポートする流れ



HCSCTE プロジェクトの HCSC コンポーネントを XML ファイルにエクスポートする流れを次に示します。

図 6-2　HCSCTE プロジェクトの HCSC コンポーネントを XML ファイルにエクスポートする流れ



6.2　サービス定義のインポート・エクスポートの使用の目的

サービス定義のインポートおよびエクスポートは，次の目的で使用します。

- 大量の HCSC コンポーネントの開発

  ビジネスプロセス，サービスアダプタ，ユーザ定義受付（呼出先選択）などの HCSC コンポーネントの内容を定義したサービス定義 XML ファイルをインポートすることで，大量の HCSC コンポーネントを一括で開発できます。

  また，次の流れで HCSC コンポーネントを効率良く作成できます。

  1. HCSCTE プロジェクトに HCSC コンポーネントのサンプルを作成します。
  2. 手順 1 で作成したサンプルをエクスポートし，サービス定義 XML ファイルとして任意のディレクトリに出力します。
  3. サンプルを基に項目をカスタマイズした，サービス定義 XML ファイルを作成します。

     手順 2 のサービス定義 XML ファイルには，手順 1 の HCSC コンポーネントのサンプルの内容が反映されています。手順 2 のサービス定義 XML ファイルを参考にサービス定義 XML ファイルを作成し，サンプルと変えたい部分をカスタマイズします。
  4. サービス定義のインポートで，手順 3 で作成したサービス定義 XML ファイルをインポートします。

     HCSCTE プロジェクトに HCSC コンポーネントが作成されます。

- HCSC コンポーネントのバージョン管理

  サービス定義のインポートに利用したサービス定義 XML ファイルや，サービス定義のエクスポートで取得したサービス定義 XML ファイルを，バージョン管理システムに登録することで，HCSC コンポーネントの内容をバージョン管理できます。これによって，バージョン管理システム上で HCSC コンポーネントの内容を確認したり，差分を確認したりできます。

サービス定義 XML ファイルを使用したサービス定義のインポートでは，次のことができます。

- HCSC コンポーネントの新規作成
- HCSC コンポーネントの編集

サービス定義 XML ファイルで設定できる項目については，「6.3.2　サービス定義 XML ファイルの作成と指定内容」を参照してください。

注意事項

- 一部の HCSC コンポーネントやアクティビティなどの項目は，サービス定義 XML ファイルでは設定できません。また，サポートしていない項目を含むサービス定義をエクスポートした場合，実行時にエラーとなります。設定できない項目を，階層ごとに次の表に示します。なお，設定の可否欄が「－」となっている項目は階層を表します。

表 6-1　サービス定義 XML ファイルで設定できない項目

項目	設定の可否
ビジネスプロセス	−
アクティビティ	−
並列処理開始	設定できません。
並列処理終了	設定できません。
ユーザ定義受付（呼出先固定）	−
要求電文・応答電文	−
データ変換	設定できません。
ユーザ定義受付（呼出先選択）	−
要求電文・応答電文	−
データ変換	設定できません。
サービスアダプタ	−
要求電文・応答電文	−
データ変換	設定できません。

6.3　サービス定義 XML ファイルの内容

サービス定義 XML ファイルによるビジネスプロセス定義の考え方や指定内容について説明します。

6.3.1　サービス定義 XML ファイルでのビジネスプロセス定義の考え方

ここでは，サービス定義 XML ファイルでビジネスプロセスを定義する際の考え方について説明します。

(1)　ビジネスプロセスの各構成要素の指定方法

ビジネスプロセスを次の構成要素に分け，それぞれについて説明します。

- 概要情報
- 変数群
- 相関セット群
- アクティビティ群とコネクション群
- ユーザ定義受付（呼出先固定）
- ビジネスプロセスの描画情報

各構成要素が，サービスプラットフォームの開発環境ではどの画面に対応するか，およびサービス定義 XML ファイルのどの要素に対応するかを次の表に示します。

表 6-2　構成要素ごとの開発環境の画面とサービス定義 XML ファイルの要素の対応

項番	構成要素	サービスプラットフォームの開発環境の画面	サービス定義 XML ファイルの要素
1	概要情報	ビジネスプロセス定義画面表示時のプロパティービュー	<service-definition>－<business-process>－<overview>要素
2	変数群	［変数・相関セット一覧］ダイアログの［変数一覧］	<service-definition>－<business-process>－<variables>要素 • ビジネスプロセス全体で参照される変数は，<variables>－<global>要素以下で定義します。 • スコープアクティビティ内だけで参照される変数は，<variables>－<scope>要素以下で定義します。スコープアクティビティ名は，<scope>要素の name-ref 属性値で指定します。
3	相関セット群	［変数・相関セット一覧］ダイアログの［相関セット一覧］	<service-definition>－<business-process>－<correlation-sets>要素

項番	構成要素	サービスプラットフォームの開発環境の画面	サービス定義 XML ファイルの要素
3	相関セット群	［変数・相関セット一覧］ダイアログの［相関セット一覧］	• ビジネスプロセス全体で参照される相関セットは，<correlation-sets>－<global>要素以下で定義します。 • スコープアクティビティ内だけで参照される相関セットは，<correlation-sets>－<scope>要素以下で定義します。スコープアクティビティ名は，<scope>要素の name-ref 属性値で指定します。
4	アクティビティ群とコネクション群	ビジネスプロセス定義画面のキャンバス	<service-definition>－<business-process>－<activities>要素 • グローバルスコープで定義するアクティビティは，<activities>－<global>要素で定義します。 • スコープアクティビティ内で定義するアクティビティは，<activities>－<scope>要素で定義します。スコープアクティビティ名は，<scope>要素の name-ref 属性値で指定します。 • 繰り返しアクティビティ内で定義するアクティビティは，<activities>－<while>要素で定義します。繰り返しアクティビティ名は，<while>要素の name-ref 属性値で指定します。 • サービス定義 XML ファイルでは，各アクティビティ間のコネクションに直接対応する要素は存在しません。詳細については「(2)　シーケンスの分類」を参照してください。
5	ユーザ定義受付（呼出先固定）	ユーザ定義受付定義画面	<service-definition>－<business-process>－<receptions>要素
6	ビジネスプロセスの描画情報	［パレット設定］ダイアログの［アイコンサイズ］ ビジネスプロセス定義画面のキャンバス	• アイコンサイズは，<draw-info>－<icon-size-type>要素で指定します。 • コメントは，<comment>要素で指定します。 • アクティビティの座標は，<activity>要素で指定します。 • コネクションの折れ曲がり点座標は，<connection>要素で指定します。

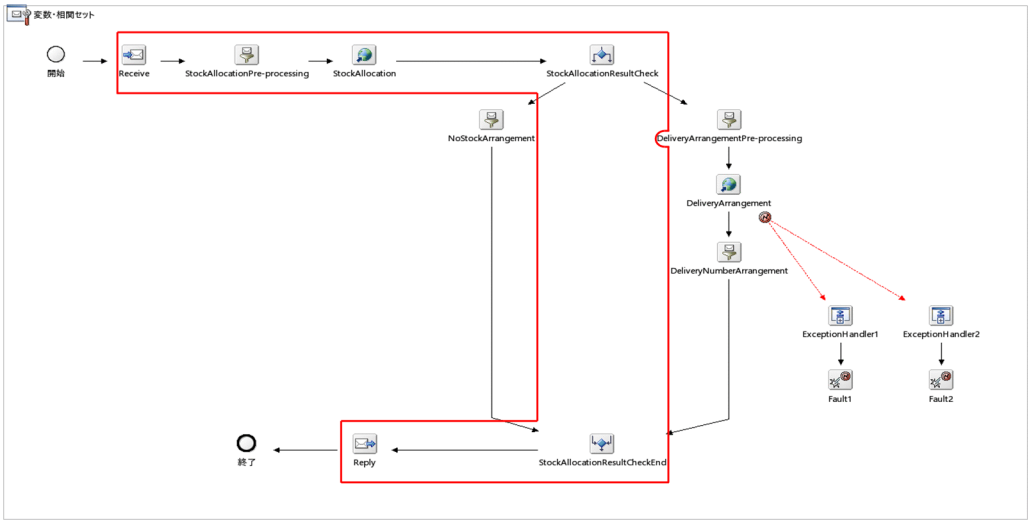
(2)　シーケンスの分類

サービス定義 XML ファイルでは，各アクティビティ間のコネクションに直接対応する要素は存在しません。サービス定義 XML ファイルのアクティビティ定義が出現する順にコネクションを接続します。

コネクションの接続状況に応じて，アクティビティ群を次の 2 つのパターンに分類します。なお，開始アクティビティと終了アクティビティはどちらにも分類されません。

- メインシーケンス

  開始アクティビティから終了アクティビティに到達するようにコネクションで接続されたアクティビティ群です。ただし，分岐開始アクティビティと対応する分岐終了アクティビティは，直接接続されているとみなします。そのため，分岐開始アクティビティと対応する分岐終了アクティビティとの間のアクティビティは，メインシーケンスに含めません。



（凡例）　：メインシーケンス

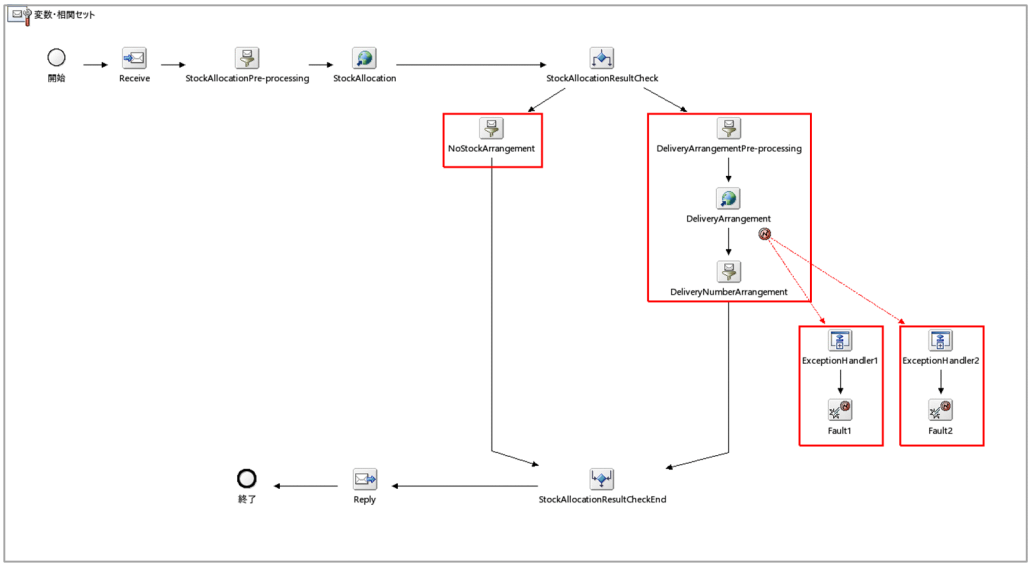
  この定義から，グローバルスコープ，スコープアクティビティ，または繰り返しアクティビティには，それぞれ最大 1 つのメインシーケンスが存在することになります。

  メインシーケンスは，サービス定義 XML ファイルの main-sequence 要素に対応します。

  main-sequence 要素については「6.3.4(6)　main-sequence」を参照してください。

- サブシーケンス

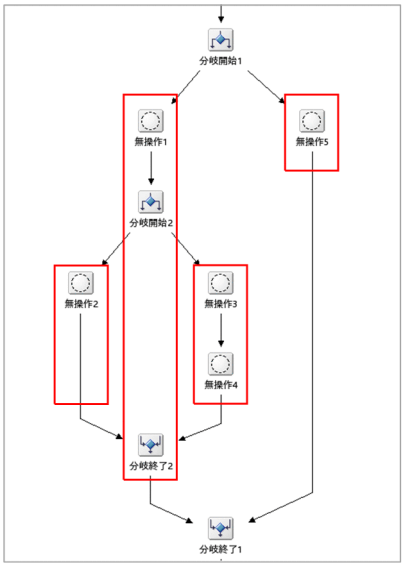
  メインシーケンスに含まれないアクティビティ群です。分岐がない先頭のアクティビティと対応する末端のアクティビティまでを，1 つのサブシーケンスとします。



（凡例） ：サブシーケンス

メインシーケンスに含まれないアクティビティは，どれか 1 つのサブシーケンスにだけ含まれます。また，メインシーケンスと同様に，分岐開始アクティビティと対応する分岐終了アクティビティは，直接接続されているとみなします。

例えば，分岐したコネクションの先に別の分岐がある場合，サブシーケンスは次の図のようになります。



（凡例） ：サブシーケンス

サブシーケンスは，サービス定義 XML ファイルの sub-sequence 要素に対応します。

sub-sequence 要素については「6.3.4(7)　sub-sequence」を参照してください。

6.3.2　サービス定義 XML ファイルの作成と指定内容

HCSCTE プロジェクトにインポートするサービス定義 XML ファイルを作成します。

サービス定義 XML ファイルには，ビジネスプロセス，サービスアダプタ，またはユーザ定義受付（呼出先選択）の内，どれか 1 つの内容を記載します。ビジネスプロセスの描画情報も記載できます。サービス定義 XML ファイルは，「6.3.3　サービス定義 XML ファイルのルートを指定する要素」～「6.3.8　ビジネスプロセスの描画情報を指定する要素」，または XML スキーマの内容に沿って作成してください。なお，サービス定義 XML ファイルは，サービス定義をエクスポートしてひな形を取得しておくと便利です。

XML スキーマの格納先を次に示します。

- サービス定義 XML ファイルのルート

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥service_text_import.xsd
```

- ビジネスプロセス

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥business_process.xsd
```

- ユーザ定義受付（呼出先固定）の一覧

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥fixed_reception.xsd
```

- サービスアダプタ

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥service_adapter.xsd
```

- ユーザ定義受付（呼出先選択）

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥selectable_reception.xsd
```

- ビジネスプロセスの描画情報

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥textimport¥draw_info.xsd
```

サービス定義 XML ファイルの指定内容については，次に示す個所を参照してください。

- 「6.3.3　サービス定義 XML ファイルのルートを指定する要素」
- 「6.3.4　ビジネスプロセスを指定する要素」
- 「6.3.5　ユーザ定義受付（呼出先固定）の一覧を指定する要素」
- 「6.3.6　サービスアダプタを指定する要素」
- 「6.3.7　ユーザ定義受付（呼出先選択）を指定する要素」
- 「6.3.8　ビジネスプロセスの描画情報を指定する要素」

これらの表中で使用する項目と説明を次に示します。

表 6-3　サービス定義 XML ファイルの表中で用いる項目と説明

項目名	説明
要素名	設定する要素を表します。@で始まる値は属性の名称を表します。 また，要素名にリンクがある場合，その要素に子要素があることを示します。リンクをクリックして子要素の表を参照してください。
出現パターン	要素が出現する回数を示します。
内容	要素に定義する内容を示します。
指定可能値	要素に指定できる値を示します。指定できない場合にはハイフン「-」を記述します。

6.3.3　サービス定義 XML ファイルのルートを指定する要素

サービス定義 XML ファイルのルートを指定する要素を次に示します。

(1)　service-definition

要素名	出現パターン	内容	指定可能値
service-definition	1 回	サービス定義 XML ファイルのルート要素です。	-
business-process※	1 回	ビジネスプロセスを定義する場合に指定する要素です。	-
service-adapter※	1 回	サービスアダプタを定義する場合に指定する要素です。	-
selectable-reception※	1 回	ユーザ定義受付（呼出先選択）を定義する場合に指定する要素です。	-

注※

business-process 要素，service-adapter 要素，または selectable-reception 要素の内，どれか 1 つを指定します。

6.3.4　ビジネスプロセスを指定する要素

ビジネスプロセスを指定する要素を次に示します。

(1)　business-process

要素名	出現パターン	内容	指定可能値
business-process	1 回	ビジネスプロセスを定義する要素です。	-
overview	1 回	ビジネスプロセスの概要を定義する要素です。	-
variables	1 回	変数情報の一覧を定義する要素です。	-

要素名	出現パターン	内容	指定可能値
correlation-sets	1 回	相関セットの一覧を定義する要素です。	-
activities	1 回	アクティビティの内容を定義する要素です。	-
receptions	1 回	ユーザ定義受付（呼出先固定）の一覧を定義する要素です。	-

(2) overview

要素名	出現パターン	内容	指定可能値
overview	1 回	ビジネスプロセスの概要を定義する要素です。	-
name	1 回	ビジネスプロセスの名称を指定します。	NCName 型
service-id	1 回	ビジネスプロセスのサービス ID を指定します。	1～8 文字の半角英数字，およびアンダーバー（_）
persistent	0 または 1 回	ステータスの永続化を指定します。 デフォルト値は「yes」です。	yes no
scope-default-handler-use	0 または 1 回	スコープアクティビティのデフォルトハンドラを有効にするかどうかを指定します。デフォルト値は「OFF」です。	ON OFF
draw-info-file-path	0 または 1 回	ビジネスプロセスの描画情報ファイルのパスです。 ファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(3) variables

要素名	出現パターン	内容	指定可能値
variables	1 回	変数情報の一覧を定義する要素です。	-
global	1 回	ビジネスプロセス全体で参照される変数を定義する要素です。	-
variable	0 回以上	変数情報の一覧を定義する要素です。	-
name	1 回	変数の名称を指定します。	NCName 型かつ global 要素の variable 要素間で一意な値
type	1 回	次に示す変数の種別を指定します。	XML

要素名								出現パターン	内容	指定可能値
					type			1 回	• XML 型（XML） • バイナリ型（non-XML） • any 型（any） • 数値（numeric） • 文字列（string） • 真偽値（boolean）	non-XML any numeric string boolean
					file-path			0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。 この要素は type 要素が XML，または non-XML の場合だけ指定します。 type 要素が XML，または non-XML 以外のとき，この指定値は無視されます。	1 文字以上の string 型
					parts			1 回	変数の部分指定の一覧を定義する要素です。type 要素が XML，または non-XML であり，かつ部分指定を定義する場合だけ指定します。 type 要素が XML，または non-XML 以外のとき，この要素以下の指定値は無視されます。	-
						part		0 回以上	変数の部分指定を定義する要素です。	-
							name	1 回	部分名を指定します。	NCName 型かつ part 要素間で一意な値
							expression	0 または 1 回	指定式を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型
							type	1 回	次に示す部分名に対応する型を選択します。 • 変数の種別がメッセージ型（XML）の場合 メッセージ型（XML） 数値（numeric） 文字列（string） 真偽値（boolean） • 変数の種別がメッセージ型（non-XML）の場合 数値（numeric） 文字列（string） 真偽値（boolean）	XML numeric string boolean

要素名								出現パターン	内容	指定可能値
			scope					0 回以上	スコープ内で参照される変数を定義する要素です。	-
				@name-ref				1 回	スコープの名称を指定します。	NCName 型かつ scope 要素間で一意な値
				variable				0 回以上	変数情報の一覧を定義する要素です。	-
					name			1 回	変数の名称を指定します。	NCName 型かつ scope 要素の variable 要素間で一意な値
					type			1 回	次に示す変数の種別を指定します。 • XML 型（XML） • バイナリ型（non-XML） • any 型（any） • 数値（numeric） • 文字列（string） • 真偽値（boolean）	XML non-XML any numeric string boolean
					file-path			0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。 この要素は type 要素が XML，または non-XML の場合だけ指定します。 type 要素が XML，または non-XML 以外のとき，この指定値は無視されます。	1 文字以上の string 型
					parts			1 回	変数の部分指定の一覧を定義する要素です。 type 要素が XML，または non-XML であり，かつ部分指定を定義する場合だけ指定します。 type 要素が XML，または non-XML 以外のとき，この要素以下の指定値は無視されます。	-
						part		0 回以上	変数の部分指定を定義する要素です。	-
							name	1 回	部分名を指定します。	NCName 型かつ part 要素間で一意な値
							expression	0 または 1 回	指定式を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型

要素名	出現パターン	内容	指定可能値
type	1 回	次に示す部分名に対応する型を選択します。 • 変数の種別がメッセージ型（XML）の場合 メッセージ型（XML） 数値（numeric） 文字列（string） 真偽値（boolean） • 変数の種別がメッセージ型（non-XML）の場合 数値（numeric） 文字列（string） 真偽値（boolean）	XML numeric string boolean

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(4) correlation-sets

要素名	出現パターン	内容	指定可能値
correlation-sets	1 回	相関セットの一覧を定義する要素です。	-
global	1 回	ビジネスプロセス全体で参照される相関セットの一覧を定義する要素です。	-
correlation-set	0 回以上	相関セットを定義する要素です。	-
name	1 回	相関セット名を指定します。	NCName 型かつ global 要素の correlation-set 要素間で一意な値
properties	1 回	相関セットの情報一覧を定義する要素です。	-
property	1 回以上	相関セットの情報を定義する要素です。	-
variable-name-ref	1 回	利用変数を指定します。	NCName 型
part-name-ref	1 回	部分名を指定します。	NCName 型かつ property 要素間で一意な値

要素名	出現パターン	内容	指定可能値
scope	0 回以上	スコープ内で参照される相関セットの一覧を定義する要素です。	-
@name-ref	1 回	スコープの名称を指定します。	NCName 型かつ scope 要素間で一意な値
correlation-set	0 回以上	相関セットを定義する要素です。	-
name	1 回	相関セット名を指定します。	NCName 型かつ scope 要素の correlation-set 要素間で一意な値
properties	1 回	相関セットの情報一覧を定義する要素です。	-
property	1 回以上	相関セットの情報を定義する要素です。	-
variable-name-ref	1 回	利用変数を指定します。	NCName 型
part-name-ref	1 回	部分名を指定します。	NCName 型かつ property 要素間で一意な値

(5) activities

要素名	出現パターン	内容	指定可能値
activities	1 回	アクティビティの内容を定義する要素です。	-
global	1 回	グローバルスコープ内のアクティビティを定義する要素です。	-
main-sequence	1 回	グローバルスコープ内のメインシーケンスを定義する要素です。	-
sub-sequence	0 回以上	グローバルスコープ内のサブシーケンスを定義する要素です。	-
scope	0 回以上	スコープ内のアクティビティを定義する要素です。	-
@name-ref	1 回	スコープの名称を指定します。	NCName 型かつ scope 要素, while 要素の間で一意な値

要素名	出現パターン	内容	指定可能値
main-sequence	1 回	スコープ内のメインシーケンスを定義する要素です。	-
sub-sequence	0 回以上	スコープ内のサブシーケンスを定義する要素です。	-
while	0 回以上	繰り返し内のアクティビティを定義する要素です。	-
@name-ref	1 回	繰り返しアクティビティの名称を指定します。	NCName 型かつ scope 要素，while 要素の間で一意な値
main-sequence	1 回	繰り返しアクティビティ内のメインシーケンスを定義する要素です。	-
sub-sequence	0 回以上	繰り返しアクティビティ内のサブシーケンスを定義する要素です。	-

(6) main-sequence

要素名	出現パターン	内容	指定可能値
main-sequence	1 回	メインシーケンスを定義する要素です。	-
<ActivityGroup に属する要素>	0 回以上	<ActivityGroup に所属する要素>を指定します。	-

(7) sub-sequence

要素名	出現パターン	内容	指定可能値
sub-sequence	0 回以上	サブシーケンスを定義する要素です。	-
@name	1 回	サブシーケンスの名称を指定します。 この属性には，シーケンスの先頭となる<ActivityGroup に属する要素>の activity-name 値を指定します。	NCName 型かつすべての sub-sequence 要素間で一意な値
<ActivityGroup に属する要素>	1 回以上	<ActivityGroup に所属する要素>を指定します。	-

(8) ActivityGroup

ActivityGroup に属する要素	説明	設定可否
receive-activity	受付アクティビティを定義する要素です。	○
reply-activity	応答アクティビティを定義する要素です。	○
invoke-activity	サービス呼出アクティビティを定義する要素です。	○
java-activity	Java 呼出アクティビティを定義する要素です。	○
datatrans-activity	データ変換アクティビティを定義する要素です。	○
assign-activity	代入アクティビティを定義する要素です。	○
empty-activity	無操作アクティビティを定義する要素です。	○
throw-activity	フォルト送出アクティビティを定義する要素です。	○
compensate-activity	補償アクティビティを定義する要素です。	○
wait-activity	待機アクティビティを定義する要素です。	○
validate-activity	検証アクティビティを定義する要素です。	○
log-activity	ログ出力アクティビティを定義する要素です。	○
exit-activity	途中終了アクティビティを定義する要素です。	○
scope-activity	スコープアクティビティを定義する要素です。	○
while-activity	繰り返しアクティビティを定義する要素です。	○
switch-start-activity	分岐開始アクティビティを定義する要素です。	○
switch-end-activity	分岐終了アクティビティを定義する要素です。	○
flow-start-activity	並列処理開始アクティビティを定義する要素です。	×
flow-end-activity	並列処理終了アクティビティを定義する要素です。	×

（凡例）

○：設定できます

×：設定できません。

(9) receive-activity

要素名	出現パターン	内容	指定可能値
receive-activity	-	受付アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビ

要素名								出現パターン	内容	指定可能値
						activity-name		1 回	アクティビティの名称を指定します。	ティ間で一意な値
						operation-name		0 または 1 回	オペレーション名を指定します。	NCName 型
						body-variable-name-ref		0 または 1 回	ボディ割当変数名を指定します。	NCName 型
						headers		1 回	ヘッダ割当変数の一覧を定義する要素です。	-
							header	0 回以上	ヘッダ割当変数を定義する要素です。	-
								variable-name-ref	1 回	割当変数名を指定します。	NCName 型かつ header 要素間で一意な値
								root-element-name	1 回	ルート要素を指定します。	NCName 型
						correlation-sets		1 回	割当相関セット群を定義する要素です。	-
							correlation-set	0 回以上	割当相関セットを定義する要素です。	-
								name-ref	1 回	相関セット名を指定します。	NCName 型かつ correlation-set 要素間で一意な値
								initialize	0 または 1 回	初期化するかどうかを指定します。 デフォルト値は「no」です。	yes no
						communication-type		0 または 1 回	次に示すどちらかの通信モデルを指定します。 • 同期（SYNC） • 非同期（ASYNC） デフォルト値は「SYNC」です。	SYNC ASYNC
						instance		0 または 1 回	インスタンスを生成するかどうかを指定します。 • 生成する（yes） • 生成しない（no） デフォルト値は「yes」です。	yes no

(10) reply-activity

要素名								出現パターン	内容	指定可能値
reply-activity								-	応答アクティビティを定義する要素です。	-
	activity-name							1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
	operation-name							0 または 1 回	オペレーション名を指定します。	NCName 型
	body-variable-name-ref							0 または 1 回	ボディ割当変数名を指定します。	NCName 型
	headers							1 回	ヘッダ割当変数の一覧を定義する要素です。	-
		header						0 回以上	ヘッダ割当変数を定義する要素です。	-
			variable-name-ref					1 回	割当変数名を指定します。	NCName 型かつ header 要素間で一意な値
			root-element-name					1 回	ルート要素を指定します。	NCName 型
	correlation-sets							1 回	割当相関セット群を定義する要素です。	-
		correlation-set						0 回以上	割当相関セットを定義する要素です。	-
			name-ref					1 回	相関セット名を指定します。	NCName 型かつ correlation-set 要素間で一意な値
			initialize					0 または 1 回	初期化するかどうかを指定します。 • 初期化する（yes） • 初期化しない（no） デフォルト値は「no」です。	yes no
	fault-name							0 または 1 回	フォルト名を指定します。	NCName 型

(11) invoke-activity

要素名								出現パターン	内容	指定可能値
					invoke-activity			-	サービス呼出アクティビティを定義する要素です。	-
						activity-name		1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						service-name		0 または 1 回	サービス名を指定します。	NCName 型
						operation-name		0 または 1 回	オペレーション名を指定します。	NCName 型
						communication-type		0 または 1 回	次に示すどちらかの通信モデルを指定します。 • 同期（SYNC） • 非同期（ASYNC） デフォルト値は「SYNC」です。	SYNC ASYNC
						input-variable-name-ref		0 または 1 回	要求電文のボディ割当変数名を指定します。	NCName 型
						input-headers		1 回	要求電文のヘッダ割当変数の一覧を定義する要素です。	-
							header	0 回以上	ヘッダ割当変数を定義する要素です。	-
								variable-name-ref	1 回	割当変数名を指定します。	NCName 型かつ header 要素間で一意な値
								root-element-name	1 回	ルート要素を指定します。	NCName 型
						output-variable-name-ref		0 または 1 回	応答電文のボディ割当変数名を指定します。 communication-type 要素の指定値が ASYNC の場合，この要素の指定値は無視されます。	NCName 型
						output-headers		1 回	応答電文のヘッダ割当変数の一覧を定義する要素です。 communication-type 要素の指定値が ASYNC の場合，この要素以下の指定値は無視されます。	-

要素名									出現パターン	内容	指定可能値
							header		0 回以上	ヘッダ割当変数を定義する要素です。	-
								variable-name-ref	1 回	割当変数名を指定します。	NCName 型かつ header 要素間で一意な値
								root-element-name	1 回	ルート要素を指定します。	NCName 型
							correlation-sets		1 回	割当相関セット群を定義する要素です。	-
								correlation-set	0 回以上	割当相関セットを定義する要素です。	-
									name-ref	1 回	相関セット名を指定します。	NCName 型かつ correlation-set 要素間で一意な値
									initialize	0 または 1 回	初期化するかどうかを指定します。 デフォルト値は「no」です。	yes no
									pattern	0 または 1 回	パターンを指定します。 デフォルト値は「in」です。	out in out-in
							fault-items		1 回	フォルトコネクションの遷移先一覧を定義する要素です。	-
							compensation-item		0 または 1 回	補償コネクションの遷移先を定義する要素です。	-

(12) java-activity

要素名							出現パターン	内容	指定可能値
					java-activity		-	Java 呼出アクティビティを定義する要素です。	-
						activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						java-class-name	0 または 1 回	Java プログラムのクラス名を指定します。	NCName 型

要素名							出現パターン	内容	指定可能値
						input-variable-names	1 回	引数用割当変数の一覧を定義する要素です。	-
						variable-name-ref	0 回以上	引数用割当変数名を指定します。	NCName 型
						output-variable-name-ref	0 または 1 回	戻り値用割当変数名を指定します。	NCName 型
						fault-items	1 回	フォルトコネクションの遷移先一覧を定義する要素です。	-

(13) datatrans-activity

要素名								出現パターン	内容	指定可能値
					datatrans-activity			-	データ変換アクティビティを定義する要素です。	-
						activity-name		1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						source-variable-names		1 回	変換元変数の一覧を定義する要素です。	-
							variable-name-ref	0 回以上	変換元変数名を指定します。 tabular-xml-import 要素で表形式 XML ファイルを定義する場合は，この要素を必ず指定してください。	NCName 型かつこの要素間で一意な値
						target-variable-name-ref		0 または 1 回	変換先変数名を指定します。 tabular-xml-import 要素で表形式 XML ファイルを定義する場合は，この要素を必ず指定してください。	NCName 型
						transform-definition-file-name		0 または 1 回	旧バージョンで使用していた互換性維持用の要素です。 11-50 以前では，データ変換定義のファイル名を指定します。前後の空白文字列はトリムされます。 11-60 以降では，指定なし，または空要素<transform-	空要素または XML で使用できる 1 文字以上の文字列

要素名								出現パターン	内容	指定可能値
						transform-definition-file-name		0 または 1 回	definition-file-name></transform-definition-file-name>を指定します。エクスポート時は，空要素<transform-definition-file-name></transform-definition-file-name>が出力されます。	空要素または XML で使用できる 1 文字以上の文字列
						mapping-definition-file-path		0 または 1 回	マッピング定義ファイルのパス※1 を絶対パスまたは相対パス※2 で指定します。 mapping-definition-file-path 要素と tabular-xml-import 要素はどちらか一方しか指定できません。	1 文字以上の string 型
						tabular-xml-import		0 または 1 回	表形式 XML ファイルを定義する要素です。 tabular-xml-import 要素と mapping-definition-file-path 要素はどちらか一方しか指定できません。	-
							file-path	1 回	表形式 XML ファイルのパスを絶対パスまたは相対パス※2 で指定します。	1 文字以上の string 型
							source-root-element-infos	1 回	変換元変数のルート要素の一覧を定義する要素です。	-
							root-element-info	1 回以上	変換元変数のルート要素の情報を定義する要素です。 variable-name-ref 要素で指定した変換元変数名の順番に合わせて指定してください。	-
							element-name	1 回	変換元変数のルート要素名を指定します。	NCName 型
							namespace	0 または 1 回	変換元変数のルート要素の，XML の名前空間を指定します。 変数のスキーマに同じ名称のルート要素が 1 つだけ定義されている場合は，この要素を省略できます。	anyURI 型
							target-root-element-info	1 回	変換先変数名のルート要素の情報を定義する要素です。	-

要素名	出現パターン	内容	指定可能値
element-name	1 回	変換先変数のルート要素名を指定します。	NCName 型
namespace	0 または 1 回	変換先変数のルート要素の XML の名前空間を指定します。 変数のスキーマに同じ名称のルート要素が 1 つだけ定義されている場合は，この要素を省略できます。	anyURI 型

注※1

ビジネスプロセス定義画面のデータ変換アクティビティから作成したマッピング定義ファイルのパスだけ指定できます。

注※2

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(14) assign-activity

要素名	出現パターン	内容	指定可能値
assign-activity	-	代入アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
copy-items	1 回	代入操作の一覧を定義する要素です。	-
copy-item	0 回以上	代入操作を定義する要素です。	-
from-item	1 回	コピー元の情報を定義する要素です。	-
from-variable	1 回	コピー元の情報を変数で定義する要素です。 from-variable 要素，from-expression 要素，from-text 要素，from-non-text 要素は，これらの要素からどれか 1 つを選択して指定します。	-

要素名	出現パターン	内容	指定可能値
variable-name-ref	1 回	コピー元の変数名を指定します。	NCName 型
part-name-ref	0 または 1 回	コピー元変数の部分名を指定します。	NCName 型
from-expression	1 回	コピー元の情報を XPath 式で定義する要素です。 from-expression 要素，from-variable 要素，from-text 要素，from-non-text 要素は，これらの要素からどれか 1 つを選択して指定します。	-
expression	1 回	コピー元の情報を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型
from-text	1 回	コピー元の固定値（テキスト）です。 from-text 要素，from-variable 要素，from-expression 要素，from-non-text 要素は，これらの要素からどれか 1 つを選択して指定します。	-
data※1	1 回	テキストデータです。	string 型（改行文字指定可能）
from-non-text	1 回	コピー元の固定値（非テキスト）です。 from-non-text 要素，from-variable 要素，from-expression 要素，from-text 要素は，これらの要素からどれか 1 つを選択して指定します。	-
file-path	1 回	非テキストデータのファイルパスです。 ファイルパスを絶対パスまたは相対パス※2 で指定します。	1 文字以上の string 型
to-item	1 回	コピー先の情報を定義する要素です。	-

要素名	出現パターン	内容	指定可能値
variable-name-ref	1 回	コピー先の変数名を指定します。	NCName 型
part-name-ref	0 または 1 回	コピー先変数の部分名を指定します。	NCName 型

注※1

要素値に含まれる改行（LF）は，インポート時に改行（CR+LF）に変換されます。

注※2

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(15) empty-activity

要素名	出現パターン	内容	指定可能値
empty-activity	-	無操作アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値

(16) throw-activity

要素名	出現パターン	内容	指定可能値
throw-activity	-	フォルト送出アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
variable-name-ref	0 または 1 回	割当変数名を指定します。	NCName 型

(17) compensate-activity

要素名	出現パターン	内容	指定可能値
compensate-activity	-	補償アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアク

要素名	出現パターン	内容	指定可能値
activity-name	1 回	アクティビティの名称を指定します。	ティビティ間で一意な値
activities-for-compensation	1 回	補償するアクティビティを定義する要素です。	-
all	1 回	補償範囲内に存在し，補償ハンドラが定義されているすべてのアクティビティ※を補償する場合に指定します。 all 要素と only-one 要素はどちらか一方しか指定できません。	-
only-one	1 回	補償範囲内に存在し，補償ハンドラが定義されているアクティビティ※の内，1 つのみを補償する場合に指定する要素です。 only-one 要素と all 要素はどちらか一方しか指定できません。	-
activity-name-ref	0 または 1 回	アクティビティ名を指定します。	NCName 型

注※

overview の scope-default-handler-use 要素が ON の場合，補償範囲内に存在する，補償ハンドラが定義されていないスコープアクティビティも含みます。

(18) wait-activity

要素名	出現パターン	内容	指定可能値
wait-activity	-	待機アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
standby-type	0 または 1 回	待機時間を指定する方法として次のどちらかを指定します。 • 間隔（for） • 期限（until） デフォルト値は「for」です。	for until
expression	0 または 1 回	待機時間の間隔または期限として指定する式を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型

(19) validate-activity

要素名	出現パターン	内容	指定可能値
validate-activity	-	検証アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
variable-names	1 回	検証する変数一覧を定義する要素です。	-
variable-name-ref	0 回以上	検証する変数の名称を指定します。	NCName 型かつこの要素間で一意な値

(20) log-activity

要素名	出現パターン	内容	指定可能値
log-activity	-	ログ出力アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
log-definition	0 または 1 回	ログ出力アクティビティの詳細情報を定義する要素です。	-
log-identifier	1 回	ログ出力識別子を指定します。前後の空白文字列はトリムされます。	1 文字以上の string 型
log-contents	1 回	ログ出力内容を定義する要素です。	-
resource-key	1 回	リソース Key を指定します。 前後の空白文字列はトリムされます。 resource-key 要素，any-string 要素は，これらの要素からどちらか 1 つを選択して指定します。	1 文字以上の string 型
any-string	1 回	任意の文字列を指定します。 resource-key 要素，any-string 要素は，これらの要素からどちらか 1 つを選択して指定します。	string 型

要素名											出現パターン	内容	指定可能値
							embedded-elements				1 回	埋め込み要素の一覧を定義する要素です。	-
								embedded-element			0 回以上	埋め込み要素を定義する要素です。	-
									variable		1 回	埋め込み要素を変数で定義する要素です。 variable 要素，expression 要素は，これらの要素からどちらか 1 つを選択して指定します。	-
										variable-name-ref	1 回	変数名を指定します。	NCName 型
										part-name-ref	0 または 1 回	部分名を指定します。	NCName 型
									expression		1 回	埋め込み要素を式で定義する要素です。前後の空白文字列はトリムされます。 variable 要素，expression 要素は，これらの要素からどちらか 1 つを選択して指定します。	1 文字以上の string 型

(21) exit-activity

要素名						出現パターン	内容	指定可能値
					exit-activity	-	途中終了アクティビティを定義する要素です。	-
					activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値

(22) scope-activity

要素名						出現パターン	内容	指定可能値
					scope-activity	-	スコープアクティビティを定義する要素です。 スコープ内のシーケンスは，<service-definition>－<business-	-

要素名							出現パターン	内容	指定可能値
					scope-activity		-	process>－<activities>－<scope>要素以下で定義します。	-
						activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						fault-items	1 回	フォルトコネクションの遷移先一覧を定義する要素です。	-
						compensation-item	0 または 1 回	補償コネクションの遷移先を定義する要素です。	-
						transaction-commit	0 または 1 回	次のどちらかのトランザクション制御を指定します。 • アクティビティ単位でコミットする（byActivity） • このスコープの開始時と終了時だけコミットする（onlyScope） デフォルト値は「byActivity」です。	byActivity onlyScope

(23) while-activity

要素名								出現パターン	内容	指定可能値
					while-activity			-	繰り返しアクティビティを定義する要素です。 繰り返しアクティビティ内のシーケンスは，<service-definition>－<business-process>－<activities>－<while>要素以下で定義します。	-
						activity-name		1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						condition-specified-mode		1 回	繰り返しモードが条件指定方式の場合に指定する要素です。 condition-specified-mode 要素と list-specified-mode 要素はどちらか一方しか指定できません。	-
							expression※	0 または 1 回	繰り返し条件の条件式を XPath 式で指定します。	1 文字以上の string 型（改

要素名								出現パターン	内容	指定可能値
							expression※	0 または 1 回	前後の空白文字列はトリムされます。	行文字指定可能）
							max-loop-count	0 または 1 回	繰り返しの回数の最大値を指定します。 この要素が省略された場合，最大で 2,147,483,647 回まで繰り返されます。	1～2,147,483,647 までの整数
						list-specified-mode		1 回	繰り返しモードがリスト指定方式の場合に指定する要素です。 list-specified-mode 要素と condition-specified-mode 要素はどちらか一方しか指定できません。	-
							expression※	0 または 1 回	評価結果が繰り返し要素を定義した変数の要素のパスとなる式を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型（改行文字指定可能）
							variable-name-ref	0 または 1 回	繰り返し要素を格納する変数の名称を指定します。	NCName 型
							max-loop-count	0 または 1 回	繰り返しの回数の最大値を半角の数字で指定します。 この要素を省略した場合，最大で 2,147,483,647 回まで繰り返されます。	1～2,147,483,647 までの整数

注※

要素値に含まれる改行（LF）は，インポート時に改行（CR+LF）に変換されます。

(24) switch-start-activity

要素名							出現パターン	内容	指定可能値
					switch-start-activity		-	分岐開始アクティビティを定義する要素です。	-
						activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
						case	0 回以上	分岐条件を定義する要素です。優先順位は，この要素の定義順に設定されます。 default 要素でデフォルト遷移先を定義する場合，この要素で必ず 1 つ以上の分岐条件を定義します。	-

要素名	出現パターン	内容	指定可能値
condition	0 または 1 回	条件名，条件式を定義する要素です。	-
name	1 回	条件名を指定します。 前後の空白文字列はトリムされます。	XML で使用できる 1 文字以上の文字列かつ case 要素間で一意な値
expression※	0 または 1 回	条件式を XPath 式で指定します。 前後の空白文字列はトリムされます。	1 文字以上の string 型（改行文字指定可能）
sub-sequence-ref	1 回	分岐先のサブシーケンスの参照先を定義する要素です。 サブシーケンスを次の位置で定義する場合にこの要素を指定します。 <service-definition>－<business-process>－<activities>－<global>，<scope>，または<while>－<sub-sequence>要素以下 sub-sequence-ref 要素と sub-sequence 要素は，どちらか一方しか指定できません。	-
sub-sequence	1 回	分岐先のサブシーケンスを定義する要素です。 サブシーケンスを直接定義する場合にこの要素を指定します。 sub-sequence 要素と sub-sequence-ref 要素は，どちらか一方しか指定できません。	-
default	0 または 1 回	デフォルト遷移先を定義する要素です。 この要素を定義する場合は，case 要素で必ず 1 つ以上の分岐条件を定義します。	-
sub-sequence-ref	1 回	分岐先のサブシーケンスの参照先を定義する要素です。 サブシーケンスを次の位置で定義する場合にこの要素を指定します。 <service-definition>－<business-process>－<activities>－<global>－，<scope>，または<while>－<sub-sequence>要素以下 sub-sequence-ref 要素と sub-sequence 要素は，どちらか一方しか指定できません。	-
sub-sequence	1 回	分岐先のサブシーケンスを定義する要素です。 サブシーケンスを直接定義する場合にこの要素を指定します。	-

要素名	出現パターン	内容	指定可能値
sub-sequence	1 回	sub-sequence 要素と sub-sequence-ref 要素は，どちらか一方しか指定できません。	-

注※

要素値に含まれる改行（LF）は，インポート時に改行（CR+LF）に変換されます。

(25) switch-end-activity

要素名	出現パターン	内容	指定可能値
switch-end-activity	-	分岐終了アクティビティを定義する要素です。	-
activity-name	1 回	アクティビティの名称を指定します。	NCName 型かつすべてのアクティビティ間で一意な値
source-activity-names	1 回	分岐終了アクティビティの遷移元アクティビティ一覧を定義する要素です。	-
activity-name-ref	0 回以上	分岐終了アクティビティの遷移元アクティビティ名を指定します。	NCName 型かつこの要素間で一意な値

(26) fault-items

要素名	出現パターン	内容	指定可能値
fault-items	1 回	フォルトコネクションの遷移先一覧を定義する要素です。	-
fault-item	0 回以上	フォルトコネクションの遷移先を定義する要素です。フォルト処理の割り当て順序は，この要素の定義順に設定されます。	-
variable-name-ref	0 または 1 回	フォルト処理の割当変数名を指定します。	NCName 型かつ fault-item 要素間で一意な値
sub-sequence-ref	1 回	フォルト遷移先のサブシーケンスの参照先を定義する要素です。 サブシーケンスを次の位置で定義する場合にこの要素を指定します。 <service-definition>－<business-process>－<activities>－<global>,	-

要素名	出現パターン	内容	指定可能値
sub-sequence-ref	1 回	<scope>，または<while>－<sub-sequence>要素以下 sub-sequence-ref 要素と sub-sequence 要素は，どちらか一方しか指定できません。	-
sub-sequence	1 回	フォルト遷移先のサブシーケンスを定義する要素です。 サブシーケンスを直接定義する場合にこの要素を指定します。 sub-sequence 要素と sub-sequence-ref 要素は，どちらか一方しか指定できません。	-
catch-all	0 または 1 回	割当変数が<catch-all>の場合の遷移先を定義する要素です。	-
sub-sequence-ref	1 回	フォルト遷移先のサブシーケンスの参照先を定義する要素です。 サブシーケンスを次の位置で定義する場合にこの要素を指定します。 <service-definition>－<business-process>－<activities>－<global>，<scope>，または<while>－<sub-sequence>要素以下 sub-sequence-ref 要素と sub-sequence 要素は，どちらか一方しか指定できません。	-
sub-sequence	1 回	フォルト遷移先のサブシーケンスを定義する要素です。 サブシーケンスを直接定義する場合にこの要素を指定します。 sub-sequence 要素と sub-sequence-ref 要素は，どちらか一方しか指定できません。	-

(27) compensation-item

要素名	出現パターン	内容	指定可能値
compensation-item	0 または 1 回	補償コネクションの遷移先を定義する要素です。	-
sub-sequence-ref	1 回	補償コネクションの遷移先のサブシーケンスの参照先を定義する要素です。	-

要素名	出現パターン	内容	指定可能値
sub-sequence-ref	1 回	サブシーケンスを次の位置で定義する場合にこの要素を指定します。 <service-definition>－<business-process>－<activities>－<global>，<scope>，または<while>－<sub-sequence>要素以下 sub-sequence-ref 要素と sub-sequence 要素は，どちらか一方しか指定できません。	-
sub-sequence	1 回	補償コネクション遷移先のサブシーケンスを定義する要素です。 サブシーケンスを直接定義する場合にこの要素を指定します。 sub-sequence 要素と sub-sequence-ref 要素は，どちらか一方しか指定できません。	-

(28) sub-sequence-ref

要素名	出現パターン	内容	指定可能値
sub-sequence-ref	1 回	次の位置で定義した分岐先のサブシーケンス，またはフォルト遷移先のサブシーケンスの参照先を定義します。 <service-definition>－<business-process>－<activities>－<global>，<scope>，または<while>－<sub-sequence>要素以下	-
@name-ref	1 回	参照先のサブシーケンスの名称を指定します。 サブシーケンスの名称は，次の値で定義されています。 <sub-sequence>の@name	NCName 型かつすべての sub-sequence-ref 要素間で一意な値

6.3.5 ユーザ定義受付（呼出先固定）の一覧を指定する要素

ユーザ定義受付（呼出先固定）の一覧を指定する要素を次に示します。

(1) receptions

要素名	出現パターン	内容	指定可能値
receptions	1 回	受付の一覧を定義する要素です。	-
reception	0 回以上	受付を定義する要素です。	-
name	1 回	受付名を指定します。	NCName 型かつ reception 要素間で一意な値
reception-id	1 回	受付 ID を指定します。	1～8 文字の半角英数字およびアンダーバー（_）
protocol-type	1 回	次に示す受付種別を指定します。 • SOAP 受付（SOAP） • TP1/RPC 受付（TP1/RPC） • FTP 受付（FTP） • HTTP 受付（HTTP） • Message Queue 受付（MessageQueue） • ファイルイベント受付（FileEvent） • Kafka 受付（Kafka） • スケジュール駆動受付（ScheduleTriggered） • gRPC 受付（gRPC） • カスタム受付（Custom）	SOAP TP1/RPC FTP HTTP MessageQueue FileEvent Kafka ScheduleTriggered gRPC Custom
soap-definition	1 回	SOAP 受付を定義する要素です。 この要素は protocol-type 要素に「SOAP」を指定している場合に指定します。※	-
grpc-definition	1 回	gRPC 受付を定義する要素です。 この要素は protocol-type 要素に「gRPC」を指定している場合に指定します。※	-
custom-definition	1 回	カスタム受付を定義する要素です。 この要素は，protocol-type 要素に「Custom」を指定している場合に指定します。※	-
protocol-definition	1 回	SOAP 受付，gRPC 受付，またはカスタム受付以外の受付を定義する要素です。 この要素は protocol-type 要素に「SOAP」「gRPC」または	-

要素名	出現パターン	内容	指定可能値
protocol-definition	1 回	「Custom」以外を指定している場合に指定します。※	-

注※

soap-definition 要素，grpc-definition 要素，custom-definition 要素，protocol-definition 要素のうち，どれか 1 つの要素を選択して指定します。

(2) soap-definition

要素名	出現パターン	内容	指定可能値
soap-definition	1 回	SOAP 受付を定義する要素です。	-
wsdl-file-path	1 回	WSDL ファイルを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
port-name-ref	1 回	WSDL に定義されているポート名を指定します。	NCName 型
context-root	0 または 1 回	コンテキストルートを指定します。 この要素を省略した場合，reception-id 要素に指定した値が設定されます。 コンテキストルートは半角英数字，および次に示す文字から構成される 1 文字以上の文字列で指定します。 • スラッシュ（/） • 単価記号（@） • 正符号（+） • ハイフン（-） • ドル記号（$） • アンダーバー（_） • ピリオド（.） • 感嘆符（!） • 始め小括弧（(） • 終わり小括弧（)） • パーセント（%） • 波ダッシュ（~）	半角英数字および次に示す半角記号から構成される 1 文字以上の文字列 / @ + - $ _ . ! ( ) % ~
operations	1 回	オペレーションの一覧を定義する要素です。	-
operation	0 回以上	オペレーションを定義する要素です。	-
name-ref	1 回	WSDL ファイルに定義されているオペレーション名を指定します。	NCName 型

要素名	出現パターン	内容	指定可能値
input-headers	0 または 1 回	要求電文のヘッダ一覧を定義する要素です。 この要素を指定するかどうかで，インポートの対象となるヘッダ電文が次のように異なります。 • 省略した場合 WSDL ファイルに定義されたヘッダ電文をインポートします。 • 指定した場合 この要素の子要素で定義されたヘッダ電文をインポートします。	-
header	0 回以上	要求電文のヘッダを定義する要素です	-
file-path	1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
root-element-name	1 回	ルート要素名を指定します。	NCName 型
output-headers	0 または 1 回	応答電文のヘッダ一覧を定義する要素です。 この要素を指定するかどうかで，インポートの対象となるヘッダ電文が次のように異なります。 • 省略した場合 WSDL ファイルに定義されたヘッダ電文をインポートします。 • 指定した場合 この要素の子要素で定義されたヘッダ電文をインポートします。	-
header	0 回以上	応答電文のヘッダ一覧を定義する要素です。	-
file-path	1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
root-element-name	1 回	ルート要素名を指定します。	NCName 型
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。 SOAP1.1 モードの場合，この要素以下の指定は無視されます。	-

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(3) grpc-definition

要素名	出現パターン	内容	指定可能値
grpc-definition	1 回	gRPC 受付を定義する要素です。	-
proto-paths	1 回	proto-path の一覧を定義する要素です。	-
proto-path	1 回以上	proto-path を絶対パスまたは相対パス※1 で指定します。	1 文字以上の string 型
proto-file-path	1 回	proto ファイルのパスを絶対パスまたは相対パス※1 で指定します。※2	1 文字以上の string 型
grpc-service-name-ref	1 回	proto ファイルに定義されている gRPC Service 名を指定します。※3	1 文字以上の string 型
operations	1 回	オペレーション（RPC メソッド）の一覧を定義する要素です。	-
operation	1 回	オペレーション（RPC メソッド）を定義する要素です。	-
name-ref	1 回	オペレーション名（gRPC Service に定義されている RPC メソッド名）を指定します。	1 文字以上の string 型
fault-messages	1 回	フォルト電文の一覧を定義する要素です。	-
fault-message	0 回または 1 回	フォルト電文を定義する要素です。 この要素は，修正したフォルト電文フォーマットファイルを使用する場合に使用します。 この要素を省略した場合は，デフォルトのフォルト電文フォーマットファイルが指定したと見なされます。	-
format-file-path	1 回	フォルト電文フォーマットのファイルパスを絶対パスまたは相対パス※1 で指定します。	1 文字以上の string 型
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

注※1

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

注※2

proto-file-path 要素のファイルパスは，proto-path 要素に設定したディレクトリパスのうちの，どれか 1 つの配下となるように設定します。この際，proto-file-path 要素と proto-path 要素のパスの表現は同じである必要があります。

例えば，次の設定をした場合は，proto-file-path 要素のファイルパスは proto-path 要素のディレクトリパスの配下に存在しないと見なされ，インポート実行時にエラーとなります。

```
<proto-paths>
    <proto-path>./sample</proto-path>
</proto-paths>
<proto-file-path>sample/helloworld.proto</proto-file-path>
```

上記を正しく設定した例は次のようになります。

```
<proto-paths>
    <proto-path>./sample</proto-path>
</proto-paths>
<proto-file-path>./sample/helloworld.proto</proto-file-path>
```

または

```
<proto-paths>
    <proto-path>sample</proto-path>
</proto-paths>
<proto-file-path>sample/helloworld.proto</proto-file-path>
```

なお，パスセパレータ「/」および「¥」は区別されないため，どちらを用いても問題ありません。

注※3

proto ファイルにパッケージ名が定義されている場合，grpc-service-name-ref 要素にはパッケージ名を含んだ完全修飾名の gRPC Service 名を指定します。指定例を次に示します。

＜指定例 1 ＞

- proto ファイルのパッケージ名：com.hitachi.sample
- gRPC Service 名：Service1
- grpc-service-name-ref 要素に指定する値：com.hitachi.sample.Service1

＜指定例 2 ＞

- proto ファイルのパッケージ名：定義なし
- gRPC Service 名：Service2
- grpc-service-name-ref 要素に指定する値：Service2

(4) custom-definition

要素名	出現パターン	内容	指定可能値
custom-definition	1 回	カスタム受付を定義する要素です。	-
ear-file-path	1 回	EAR ファイルのパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
operations	1 回	オペレーションの一覧を定義する要素です。	-
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(5) protocol-definition

要素名	出現パターン	内容	指定可能値
protocol-definition	1 回	SOAP 受付，gPRC 受付，またはカスタム受付以外の受付を定義する要素です。	-
operations	1 回	オペレーションの一覧を定義する要素です。	-
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

(6) operations

要素名	出現パターン	内容	指定可能値
operations	1 回	オペレーションの一覧を定義する要素です。	-
operation	0 回以上	オペレーションを定義する要素です。 1 番目の operation 要素がデフォルトオペレーションに設定されます。	-
name	1 回	オペレーション名を指定します。	NCName 型かつ operation 要素間で一意な値

要素名								出現パターン	内容	指定可能値
						communication-type		0 または 1 回	次に示すどちらかの通信モデルを指定します。 • 同期（SYNC） • 非同期（ASYNC） デフォルト値は「SYNC」です。	SYNC ASYNC
						input-body		1 回	要求電文を定義する要素です。	-
							any	1 回	要求電文に any 型を使う場合に指定します。 この要素には何も値を指定しません。 any 要素と format 要素はどちらか一方しか指定できません。	-
							format	1 回	要求電文に電文フォーマットを指定する場合に指定します。 format 要素と any 要素はどちらか一方しか指定できません。	-
								file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
						output-body		0 または 1 回	応答電文を定義する要素です。 この要素は，communication-type 要素の値が「SYNC」の場合に指定します。communication-type 要素の値が「ASYNC」のとき，この要素の指定は無視されます。	-
							any	1 回	応答電文に any 型を使う場合に指定します。 この要素には何も値を指定しません。 any 要素と format 要素はどちらか一方しか指定できません。	-
							format	1 回	応答電文に電文フォーマットを指定する場合に指定します。 format 要素と any 要素はどちらか一方しか指定できません。	-
								file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(7) custom-definition-files

要素名						出現パターン	内容	指定可能値
				custom-definition-files		1 回	独自定義ファイルの一覧を定義する要素です。	-
					file-path	0 回以上	独自定義ファイルのパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

6.3.6 サービスアダプタを指定する要素

サービスアダプタを指定する要素を次に示します。

(1) service-adapter

要素名			出現パターン	内容	指定可能値
service-adapter			1 回	サービスアダプタを定義する要素です。	-
	name		1 回	サービスアダプタ名を指定します。	NCName 型
	service-id		1 回	サービスアダプタのサービス ID を指定します。	1～8 文字の半角英数字およびアンダーバー（_）
	max-instance		0 または 1 回	最大インスタンス数を指定します。	0～65535 の整数
	protocol-type		1 回	次のどれかのサービス種別を指定します。 • SOAP アダプタ（SOAP） • SessionBean アダプタ（SessionBean） • MDB（WS-R）アダプタ（MDB/WS-R） • MDB（DB キュー）アダプタ（MDB/DBQueue） • DB アダプタ（DB） • TP1 アダプタ（TP1）	SOAP SessionBean MDB/WS-R MDB/DBQueue DB TP1 File ObjectAccess

要素名	出現パターン	内容	指定可能値
protocol-type	1 回	• ファイルアダプタ（File） • Object Access アダプタ（ObjectAccess） • Message Queue アダプタ（MessageQueue） • FTP アダプタ（FTP） • ファイル操作アダプタ（FileOperation） • メールアダプタ（Mail） • HTTP アダプタ（HTTP） • コマンドアダプタ（Command） • SFTP アダプタ（SFTP） • Kafka アダプタ（Kafka） • gRPC アダプタ（gRPC） • 汎用カスタムアダプタ（Custom）	MessageQueue FTP FileOperation Mail HTTP Command SFTP Kafka gRPC Custom
convert-system-exception-to-fault	0 または 1 回	システム例外をフォルトに変換するかどうかを指定します。 • 変換する（true） • 変換しない（false） デフォルト値は「false」です。	true false
soap-definition※	1 回	SOAP アダプタを定義する要素です。protocol-type 要素が SOAP の場合だけ指定します。	-
grpc-definition※	1 回	gRPC アダプタを定義する要素です。protocol-type 要素が gRPC の場合だけ指定します。	-
session-bean-definition※	1 回	SessionBean アダプタを定義する要素です。protocol-type 要素が SessionBean の場合だけ指定します。	-
mdb-wsr-definition※	1 回	MDB（WS-R）アダプタを定義する要素です。protocol-type 要素が MDB/WS-R の場合だけ指定します。	-
mdb-dbqueue-definition※	1 回	MDB（DB キュー）アダプタを定義する要素です。protocol-type 要素が MDB/DBQueue の場合だけ指定します。	-
custom-definition※	1 回	汎用カスタムアダプタを定義する要素です。protocol-type 要素が Custom の場合だけ指定します。	-

要素名	出現パターン	内容	指定可能値
protocol-definition※	1 回	次に示すアダプタ以外のアダプタを定義する要素です。 • SOAP アダプタ • gPRC アダプタ • SessionBean アダプタ • MDB（WS-R）アダプタ • MDB（DB キュー）アダプタ • 汎用カスタムアダプタ protocol-type 要素が次のどれかのアダプタの場合に指定します。 • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ • コマンドアダプタ • SFTP アダプタ • Kafka アダプタ	-

注※

soap-definition 要素，grpc-definition 要素，session-bean-definition 要素，mdb-wsr-definition 要素，mdb-dbqueue-definition 要素，custom-definition 要素，protocol-definition 要素の内，どれか 1 つの要素を選択して指定します。

(2) soap-definition

要素名	出現パターン	内容	指定可能値
soap-definition	1 回	SOAP アダプタを定義する要素です。	-
wsdl-file-path	1 回	WSDL ファイルのパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
port-name-ref	1 回	WSDL に定義されているポート名を指定します。	NCName 型
address	0 または 1 回	アドレスを指定します。 この要素を省略した場合，WSDL ファイルで定義されたアドレスを使用します。	string 型

要素名	出現パターン	内容	指定可能値
operations	1 回	オペレーションの一覧を定義する要素です。	-
operation	0 回以上	オペレーションを定義する要素です。	-
name-ref	1 回	WSDL ファイルに定義されているオペレーション名を指定します。	NCName 型
input-headers	0 または 1 回	要求電文のヘッダ一覧を定義する要素です。 この要素を指定するかどうかで，インポートの対象となるヘッダ電文が次のように異なります。 • 省略した場合 WSDL ファイルに定義されたヘッダ電文をインポートします。 • 指定した場合 この要素の子要素で定義されたヘッダ電文をインポートします。	-
header	0 回以上	要求電文のヘッダを定義する要素です。	-
file-path	1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
root-element-name	1 回	ルート要素名を指定します。	NCName 型
output-headers	0 または 1 回	応答電文のヘッダ一覧を定義する要素です。 この要素を指定するかどうかで，インポートの対象となるヘッダ電文が次のように異なります。 • 省略した場合 WSDL ファイルに定義されたヘッダ電文をインポートします。 • 指定した場合 この要素の子要素で定義されたヘッダ電文をインポートします。	-
header	0 回以上	応答電文のヘッダを定義する要素です。	-
file-path	1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
root-element-name	1 回	ルート要素名を指定します。	NCName 型

要素名					出現パターン	内容	指定可能値
			client-definition-file-path		1 回	クライアント定義ファイルのパスを，絶対パスまたは相対パス※で指定します。 ファイル名の大文字と小文字は区別されます。 使用している SOAP モードによって，次のどちらかのファイル名を指定します。 • SOAP1.1 モードの場合 c4webcl.properties • SOAP1.1/1.2 併用モードの場合 cscwsconf.properties	1 文字以上の string 型
			basic-authentication		0 または 1 回	ベーシック認証を定義する要素です。	-
				user-name	0 または 1 回	ユーザ名を指定します。	1 文字以上の string 型
				password	0 または 1 回	パスワードを指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(3) grpc-definition

要素名				出現パターン	内容	指定可能値
		grpc-definition		1 回	gRPC アダプタを定義する要素です。	-
			proto-paths	1 回	proto-path の一覧を定義する要素です。	-
				proto-path	1 回以上	proto-path を絶対パスまたは相対パス※1 で指定します。	1 文字以上の string 型
			proto-file-path	1 回	proto ファイルのパスを絶対パスまたは相対パス※1 で指定します。※2	1 文字以上の string 型
			grpc-service-name-ref	1 回	proto ファイルに定義されている gRPC Service 名を指定します。※3	1 文字以上の string 型
			operations	1 回	オペレーション（RPC メソッド）の一覧を定義する要素です。	-
				operation	0 回以上	オペレーション（RPC メソッド）を定義する要素です。	-

要素名	出現パターン	内容	指定可能値
name-ref	1 回	オペレーション名（gRPC Service に定義されている RPC メソッド名）を指定します。	1 文字以上の string 型
fault-messages	1 回	フォルト電文の一覧を定義する要素です。	-
fault-message	0 回または 1 回	フォルト電文を定義する要素です。 この要素は，修正したフォルト電文フォーマットファイルを使用する場合に使用します。 この要素を省略した場合は，デフォルトのフォルト電文フォーマットファイルが指定したと見なされます。	-
format-file-path	1 回	フォルト電文フォーマットのファイルパスを絶対パスまたは相対パス※1で指定します。	1 文字以上の string 型
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

注※1

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

注※2

proto-file-path 要素のファイルパスは，proto-path 要素に設定したディレクトリパスのうちの，どれか 1 つの配下となるように設定します。この際，proto-file-path 要素と proto-path 要素のパスの表現は同じである必要があります。

例えば，次の設定をした場合は，proto-file-path 要素のファイルパスは proto-path 要素のディレクトリパスの配下に存在しないと見なされ，インポート実行時にエラーとなります。

```
<proto-paths>
    <proto-path>./sample</proto-path>
</proto-paths>
<proto-file-path>sample/helloworld.proto</proto-file-path>
```

上記を正しく設定した例は次のようになります。

```
<proto-paths>
    <proto-path>./sample</proto-path>
</proto-paths>
<proto-file-path>./sample/helloworld.proto</proto-file-path>
```

または

```
<proto-paths>
    <proto-path>sample</proto-path>
```

```
</proto-paths>
<proto-file-path>sample/helloworld.proto</proto-file-path>
```

なお，パスセパレータ「/」および「¥」は区別されないため，どちらを用いても問題ありません。

注※3

proto ファイルにパッケージ名が定義されている場合，grpc-service-name-ref 要素にはパッケージ名を含んだ完全修飾名の gRPC Service 名を指定します。指定例を次に示します。

＜指定例 1 ＞

- proto ファイルのパッケージ名：com.hitachi.sample
- gRPC Service 名：Service1
- grpc-service-name-ref 要素に指定する値：com.hitachi.sample.Service1

＜指定例 2 ＞

- proto ファイルのパッケージ名：定義なし
- gRPC Service 名：Service2
- grpc-service-name-ref 要素に指定する値：Service2

(4) session-bean-definition

要素名	出現パターン	内容	指定可能値
session-bean-definition	1 回	SessionBean アダプタを定義する要素です。	-
ear-file-path	1 回	EAR ファイルのパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
session-bean-info	1 回	SessionBean 情報を定義する要素です。	-
ejb-jar-file-name-ref	1 回	EAR ファイルに格納されている EJB-JAR ファイル名を指定します。 ファイル名の大文字と小文字は区別されます。	1 文字以上の string 型
remote-interface-name-ref	1 回	デプロイメント記述子（ejb-jar.xml）に定義されたリモートインタフェース名を指定します。	1 文字以上の string 型
address	0 または 1 回	アドレスを指定します。	1 文字以上の string 型

要素名	出現パターン	内容	指定可能値
client-definition-file-path	1 回	クライアント定義ファイルのパスを絶対パスまたは相対パス※で指定します。 ファイル名は c4webcl.properties で指定します。 ファイル名の大文字と小文字は区別されます。	1 文字以上の string 型
user-definition-class	1 回	ユーザ定義クラス（JAR ファイル）を定義する要素です。	-
jar-file-path	0 回以上	JAR ファイルのパスを絶対パスまたは相対パス※で指定します。 重複したファイル名を指定した場合，あとから指定したファイル名が有効になります。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(5) mdb-wsr-definition

要素名	出現パターン	内容	指定可能値
mdb-wsr-definition	1 回	MDB（WS-R）アダプタを定義する要素です。	-
address	0 または 1 回	アドレスを指定します。	1 文字以上の string 型
input-body	1 回	要求電文を定義する要素です。	-
format	1 回	要求電文に電文フォーマットを指定する場合に指定します。	-
file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パスで指定します。	1 文字以上の string 型
mdb-control-info	1 回	MDB 制御情報を定義する要素です。	-
jms-message-type	0 または 1 回	次に示すどれかの JMS メッセージの種別を指定します。 • テキスト（Text） • バイト（Byte） • オブジェクト（Object） デフォルト値は「Text」です。	Text Byte Object

要素名	出現パターン	内容	指定可能値
invoke-setting	1 回	利用するサービス部品の呼び出し方法を定義する要素です。	-
remote-invoke	1 回	呼び出し方法にリモート呼出しを指定する場合に指定します。 remote-invoke 要素と local-invoke 要素はどちらか一方しか指定できません。	-
max-message	0 または 1 回	最大メッセージ数を指定します。	0～65535 の整数
destination-url	0 または 1 回	宛先 URL を指定します。	1 文字以上の string 型
basic-authentication	0 または 1 回	ベーシック認証を定義する要素です。	-
user-name	0 または 1 回	ユーザ名を指定します。	1 文字以上の string 型
password	0 または 1 回	パスワードを指定します。	1 文字以上の string 型
RD-area	0 または 1 回	転送キュー作成先 RD エリアを指定します。	1 文字以上の string 型
sequence-guarantee	0 または 1 回	順序保証するかどうかを指定します。 • 順序保証する（true） • 順序保証しない（false） デフォルト値は「false」です。	true false
local-invoke	1 回	呼び出し方法にローカル呼出しを指定する場合に指定します。 この要素には何も値を指定しません。 local-invoke 要素と remote-invoke 要素はどちらか一方しか指定できません。	-

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(6) mdb-dbqueue-definition

要素名	出現パターン	内容	指定可能値
mdb-dbqueue-definition	1 回	MDB（DB キュー）アダプタを定義する要素です。	-

要素名						出現パターン	内容	指定可能値
			address			0 または 1 回	アドレスを指定します。	1 文字以上の string 型
			input-body			1 回	要求電文を定義する要素です。	-
				format		1 回	要求電文に電文フォーマットを指定する場合に指定します。	-
					file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(7) custom-definition

要素名				出現パターン	内容	指定可能値
		custom-definition		1 回	汎用カスタムアダプタを定義する要素です。	-
			operations	1 回	オペレーションの一覧を定義する要素です。	-
			utility-class	1 回	ユーティリティクラス（JAR ファイル）を定義する要素です。	-
			custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

(8) protocol-definition

要素名			出現パターン	内容	指定可能値
		protocol-definition	1 回	次に示すアダプタを定義する要素です。※ • DB アダプタ • TP1 アダプタ • ファイルアダプタ • Object Access アダプタ • Message Queue アダプタ • FTP アダプタ • ファイル操作アダプタ • メールアダプタ • HTTP アダプタ	-

要素名	出現パターン	内容	指定可能値
protocol-definition	1 回	• コマンドアダプタ • SFTP アダプタ • Kafka アダプタ	-
operations	1 回	オペレーションの一覧を定義する要素です。	-
utility-class	1 回	ユーティリティクラス（JAR ファイル）を定義する要素です。	-
custom-definition-files	1 回	独自定義ファイルの一覧を定義する要素です。	-

注※

gRPC アダプタについては，サービス定義のインポート・エクスポートに対応していません。

(9) operations

要素名	出現パターン	内容	指定可能値
operations	1 回	オペレーションの一覧を定義する要素です。	-
operation	0 回以上	オペレーションを定義する要素です。	-
name	1 回	オペレーション名を指定します。	NCName 型かつこの要素間で一意な値
communication-type	0 または 1 回	次に示すどちらかの通信モデルを指定します。 • 同期（SYNC） • 非同期（ASYNC） デフォルト値は「SYNC」です。	SYNC ASYNC
input-body	1 回	要求電文を定義する要素です。	-
any	1 回	要求電文に any 型を使う場合に指定します。 この要素には何も値を指定しません。 any 要素と format 要素はどちらか一方しか指定できません。	-
format	1 回	要求電文に電文フォーマットを指定する場合に指定します。 format 要素と any 要素はどちらか一方しか指定できません。	-

要素名	出現パターン	内容	指定可能値
file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
output-body	0 または 1 回	応答電文を定義する要素です。 この要素は，communication-type 要素の値が「SYNC」の場合に指定します。communication-type 要素の値が「ASYNC」のとき，この要素の指定は無視されます。	-
any	1 回	応答電文に any 型を使う場合に指定します。 この要素には何も値を指定しません。 any 要素と format 要素はどちらか一方しか指定できません。	-
format	1 回	応答電文に電文フォーマットを指定する場合に指定します。 format 要素と any 要素はどちらか一方しか指定できません。	-
file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型
fault-messages	0 または 1 回	フォルト電文の一覧を定義する要素です。 この要素は，communication-type 要素の値が「SYNC」の場合に指定します。communication-type 要素の値が「ASYNC」のとき，この要素の指定は無視されます。	-
fault-message	0 回以上	フォルト電文を定義する要素です。	-
name	1 回	フォルト名を指定します。	1 文字以上の string 型かつ fault-message 要素間で一意な値
format-file-path	0 または 1 回	電文フォーマットのファイルパスを絶対パスまたは相対パス※で指定します。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(10) utility-class

要素名					出現パターン	内容	指定可能値
			utility-class		1 回	ユーティリティクラス（JAR ファイル）を定義する要素です。	-
				jar-file-path	0 回以上	JAR ファイルのパスを絶対パスまたは相対パス※で指定します。 この要素の値には，ユーザが作成した JAR ファイルパスだけを指定します。サービスプラットフォームが提供する JAR ファイルのパスの指定は不要です。 重複したファイル名を指定した場合，あとから指定したファイル名が有効になります。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

(11) custom-definition-files

要素名					出現パターン	内容	指定可能値
			custom-definition-files		1 回	独自定義ファイルの一覧を定義する要素です。	-
				file-path	0 回以上	独自定義ファイルのパスを絶対パスまたは相対パス※で指定します。 重複したファイル名を指定した場合，あとから指定したファイル名が有効になります。	1 文字以上の string 型

注※

相対パスで指定した場合，サービス定義 XML ファイルが存在するディレクトリを起点としたパスに変換されます。

6.3.7 ユーザ定義受付（呼出先選択）を指定する要素

ユーザ定義受付（呼出先選択）を指定する要素を次に示します。

(1) selectable-reception

要素名		出現パターン	内容	指定可能値
selectable-reception		1 回	ユーザ定義受付（呼出先選択）を定義する要素です。	-
	name	1 回	ユーザ定義受付（呼出先選択）の名称を指定します。	NCName 型
	reception-id	1 回	受付 ID を指定します。	1～8 文字の半角英数字，およびアンダーバー（_）
	protocol-type	1 回	受付種別として HTTP 受付（HTTP）を指定します。	HTTP
	protocol-definition	1 回	HTTP 受付を定義する要素です。	-

6.3.8 ビジネスプロセスの描画情報を指定する要素

ビジネスプロセスの描画情報を指定する要素を次に示します。

(1) draw-info

要素名	出現パターン	内容	指定可能値
draw-info	1 回	ビジネスプロセスの描画情報のルート要素です。	-
icon-size-type	0 または 1 回	ビジネスプロセスエディタ上のアイコン描画方法を指定します。 この要素を省略した場合，fixed_middle_1line が設定されます。 • 可変幅（expandable） • 固定幅（小）－ラベル 1 行（fixed_small_1line） • 固定幅（小）－ラベル 2 行（fixed_small_2lines） • 固定幅（小）－ラベル 3 行（fixed_small_3lines） • 固定幅（中）－ラベル 1 行（fixed_middle_1line） • 固定幅（中）－ラベル 2 行（fixed_middle_2lines） • 固定幅（中）－ラベル 3 行（fixed_middle_3lines） • 固定幅（大）－ラベル 1 行（fixed_large_1line）	expandable fixed_small_1line fixed_small_2lines fixed_small_3lines fixed_middle_1line fixed_middle_2lines fixed_middle_3lines fixed_large_1line fixed_large_2lines fixed_large_3lines

要素名	出現パターン	内容	指定可能値
icon-size-type	0 または 1 回	• 固定幅（大）－ラベル 2 行（fixed_large_2lines） • 固定幅（大）－ラベル 3 行（fixed_large_3lines）	expandable fixed_small_1line fixed_small_2lines fixed_small_3lines fixed_middle_1line fixed_middle_2lines fixed_middle_3lines fixed_large_1line fixed_large_2lines fixed_large_3lines
global	0 または 1 回	グローバルスコープ内の描画情報を定義する要素です。	-
comment	0 回以上	グローバルスコープ内にあるコメントを定義する要素です。	-
activity	0 回以上	グローバルスコープ内にあるアクティビティの座標を定義する要素です。	-
connection	0 回以上	グローバルスコープ内にあるコネクションの折れ曲がり点座標を定義する要素です。	-
scope	0 回以上	スコープアクティビティ内の描画情報を定義する要素です。	-
@name-ref	1 回	スコープアクティビティの名称です。	NCName 型かつ scope 要素，while 要素の間で一意な値
comment	0 回以上	スコープアクティビティ内にあるコメントを定義する要素です。	-
activity	0 回以上	スコープアクティビティ内にあるアクティビティの座標を定義する要素です。	-
connection	0 回以上	スコープアクティビティ内にあるコネクションの折れ曲がり点座標を定義する要素です。	-
while	0 回以上	繰り返しアクティビティ内にある描画情報を定義する要素です。	-
@name-ref	1 回	繰り返しアクティビティの名称です。	NCName 型かつ scope 要素，while 要素の間で一意な値
comment	0 回以上	繰り返しアクティビティ内にあるコメントを定義する要素です。	-

要素名			出現パターン	内容	指定可能値
		activity	0 回以上	繰り返しアクティビティ内にあるアクティビティの座標を定義する要素です。	-
		connection	0 回以上	繰り返しアクティビティ内にあるコネクションの折れ曲がり点座標を定義する要素です。	-

(2) comment

要素名			出現パターン	内容	指定可能値
comment			0 回以上	コメントを定義する要素です。	-
	text※1		1 回	コメントの本文です。	string 型（改行文字指定可能）
	background		0 または 1 回	コメントの背景色です。 この要素を省略した場合，コメントの背景色は yellow になります。 • 黄色（yellow） • 赤（red） • 青（blue） • 緑（green） • 灰色（gray） • カスタム背景色 1（custom1）※2 • カスタム背景色 2（custom2）※2 • カスタム背景色 3（custom3）※2	yellow red blue green gray custom1 custom2 custom3
	is-minimized		0 または 1 回	コメントの最小化状態です。 この要素を省略した場合，false となり最小化されません。	true false
	size		0 または 1 回	コメントのサイズを定義する要素です。 この要素を省略した場合，コメントのサイズは次の値になります。 width=150 height=100	-
		@height	1 回	コメント表示の高さです。単位はピクセルです。	16 以上の整数

要素名	出現パターン	内容	指定可能値
@width	1 回	コメント表示の幅です。単位はピクセルです。	16 以上の整数
point	0 または 1 回	コメントの座標を定義する要素です。 この要素を省略した場合，自動的に座標を決定します。	-
@x	1 回	コメントの x 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数
@y	1 回	コメントの y 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数

注※1

要素値に含まれる改行（LF）は，インポート時に改行（CR+LF）に変換されます。

注※2

Eclipse で設定した色を使用します。初期値は紺色です。

(3) activity

要素名	出現パターン	内容	指定可能値
activity	0 回以上	アクティビティの座標を定義する要素です。	-
@name-ref	1 回	アクティビティ名です。	NCName 型かつ兄弟 activity 要素間で一意の値
point	1 回	座標を定義する要素です。	-
@x	1 回	アクティビティの x 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数
@y	1 回	アクティビティの y 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数

(4) connection

要素名	出現パターン	内容	指定可能値
connection	0 回以上	コネクションの折れ曲がり点座標を定義する要素です。	-
@source-activity-name-ref	1 回	コネクションの遷移元アクティビティ名です。	NCName 型
@target-activity-name-ref	1 回	コネクションの遷移先アクティビティ名です。	NCName 型
bendpoint	1 回以上	折れ曲がり点座標を定義する要素です。 遷移元から近い順に定義します。	-
@x	1 回	折れ曲がり点の x 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数
@y	1 回	折れ曲がり点の y 座標です。エディタの左上端を原点とする座標系の値です。単位はピクセルです。	0 以上の整数

6.4　サービス定義のインポート

サービス定義のインポートについて説明します。

6.4.1　サービス定義 XML ファイルのインポート手順

サービス定義 XML ファイルを HCSCTE プロジェクトにインポートします。インポートするサービス定義 XML ファイルは，サービス定義 XML ファイルを格納したディレクトリを選択することで指定します。

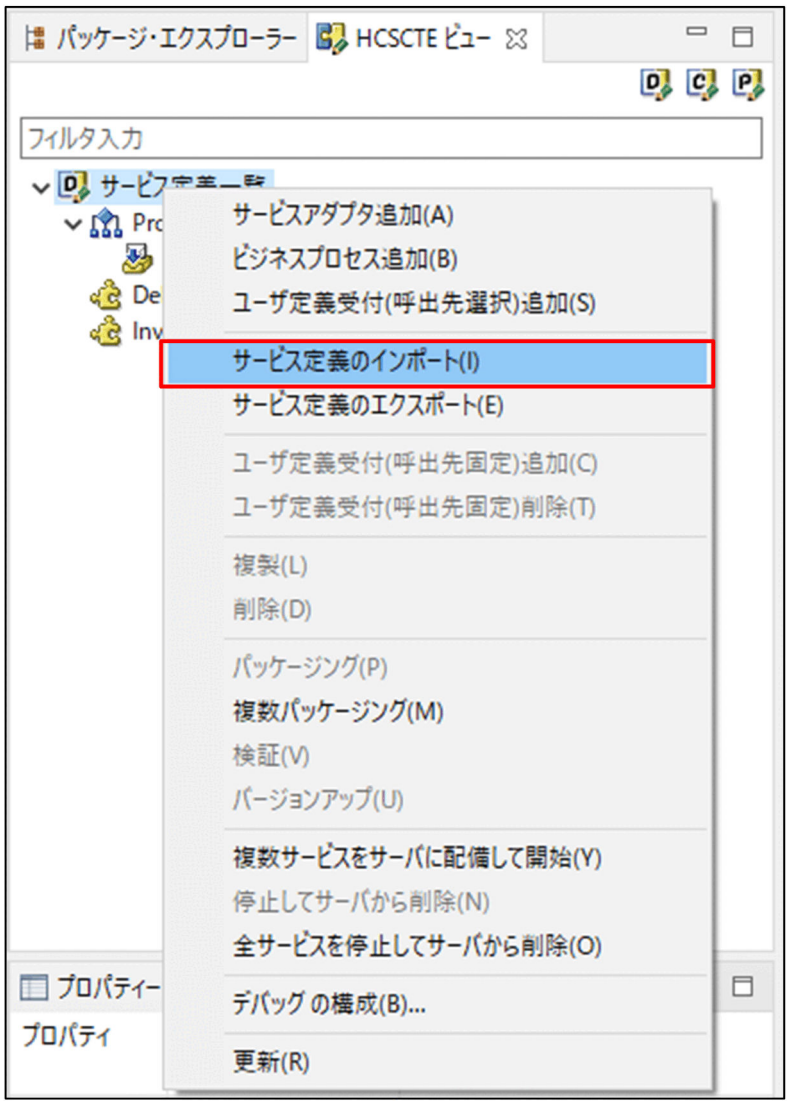
選択するディレクトリ（サブディレクトリも含みます）に配置されている XML ファイルは，次のどちらかである必要があります。

- サービス定義 XML ファイル（ルート要素が service-definition である XML ファイル）
- サービス定義 XML ファイルから参照される XML ファイル

> **注意事項**
>
> gRPC 受付，または gRPC アダプタに対してサービス定義のインポートを実行する場合は，インポートの前に，gRPC ライブラリ（OSS）のインストールと protoc コマンドの取得を実施してください。gRPC ライブラリ（OSS）のインストールと protoc コマンドの取得についての詳細は，マニュアル「サービスプラットフォーム 解説」の「2.13.1　gRPC プロトコルを利用したシステム」を参照してください。

インポートの手順を次に示します。

1. ツリービューのサービス定義一覧を右クリックして［サービス定義のインポート］を選択します。



ディレクトリを選択する画面が表示されます。

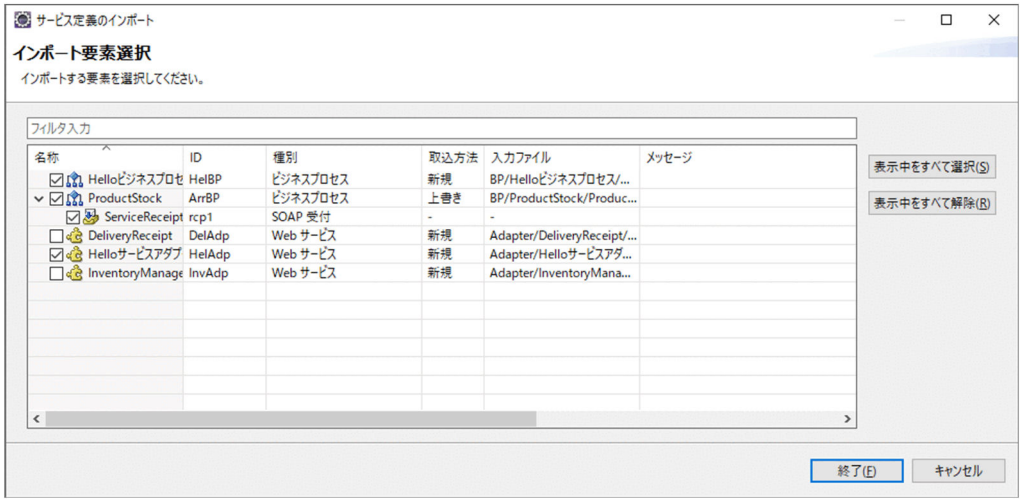
2. インポート対象のサービス定義 XML ファイルが格納されているディレクトリを選択します。

   サービス定義のインポートウィザードが表示されます。

   なお，インポート対象のサービス定義 XML ファイルが規定のフォーマットに合致していない場合は，エラーメッセージが出力されます。

   XML ファイルかどうかは拡張子で判断されます。

3. インポートする HCSC コンポーネントのチェックボックスをチェックします。



「フィルタ入力」にキーワードを入力すると，表示されるサービスをフィルタリングできます。

また，表示されたサービスのチェックボックスをすべてチェックするには［表示中をすべて選択］ボタンをクリックします。逆に，すべてのチェックを解除するには［表示中をすべて解除］ボタンをクリックします。

4. ［終了］ボタンをクリックします。

HCSC コンポーネントのインポートが，サービス定義 XML ファイル単位で実行されます。

インポートが正常終了した場合は，処理完了を伝えるダイアログが表示されます。

異常終了を示すダイアログが表示された場合は，コンソールビューを確認してください。コンソールビューにエラーメッセージが表示されている HCSC コンポーネントについては，インポートは実行されていません。コンソールビューのメッセージの内容に従いエラー原因を取り除いてから，再度インポートを実行してください。

インポート処理が正常に完了した HCSC コンポーネントについては，メッセージは何も表示されません。

参考

マッピング定義ファイルについて

サービス定義 XML ファイルに利用できるマッピング定義ファイルは，次に示す 2 つの形式があります。

・マッピング定義ファイル（.mdo）

・表形式 XML ファイル（.xml）

それぞれの使用ケースやメリットについては，「6.4.3　データ変換アクティビティに指定するマッピング定義の形式」を参照してください。

マッピング定義ファイル（.mdo）を利用する場合のインポート手順，およびマッピング定義ファイルの作成方法については，「6.4.4　マッピング定義ファイルを利用する際の注意事項」を参照してください。

ビジネスプロセスの描画情報ファイルについて

キャンバス上でのアクティビティの配置やコメントに関わる設定をインポートできます。詳細については「6.4.2　ビジネスプロセスの描画情報の指定」を参照してください。

6.4.2　ビジネスプロセスの描画情報の指定

キャンバス上でのアクティビティの配置やコメントに関わる設定は，ビジネスプロセスの描画情報ファイルに定義できます。ビジネスプロセスの描画情報ファイルで定義できる項目を次に示します。

- アクティビティのアイコンサイズ
- コメント
- アクティビティの座標
- コネクションの折れ曲がり点座標

ビジネスプロセスの描画情報ファイルの指定，およびビジネスプロセスの描画情報ファイル内の座標指定は任意です。座標が指定されていない場合，アクティビティやコメントは自動的に整列されます。

ビジネスプロセスの描画情報ファイルで定義する要素については，「6.3.8　ビジネスプロセスの描画情報を指定する要素」を参照してください。

6.4.3　データ変換アクティビティに指定するマッピング定義の形式

サービス定義 XML ファイルをインポートする際に，データ変換アクティビティに指定できるマッピング定義の形式には次の 2 つがあります。

- マッピング定義ファイル（.mdo）
- 表形式 XML ファイル（.xml）

次の表に示す利用ケースとメリットに応じて，使用する形式を選択してください。

表 6-4　マッピング定義の形式ごとの利用ケースとメリット

形式	利用ケース	メリット
マッピング定義ファイル（.mdo）※1	サービス定義のエクスポートで出力したファイルを利用する場合	インポートの実行速度が速くなります。
表形式 XML ファイル（.xml）	マッピング定義関連の情報を表形式 XML ファイルで保存して，差分を管理する場合	テキストベースのため，差分を管理しやすくなります。
	Eclipse のメニューで，［HCSCTE マッピング定義］－［マッピング定義ファイル］※2 からマッピング定義ファイルを利用する場合	データ変換定義画面のマッピングビューア上から［マッピング定義のエクスポート］※3 を実

形式	利用ケース	メリット
表形式 XML ファイル（.xml）	Eclipse のメニューで，［HCSCTE マッピング定義］－［マッピング定義ファイル］※2 からマッピング定義ファイルを利用する場合	行することで，表形式 XML ファイルが容易に作成できます。 この方法では，ビジネスプロセスを作成しないでマッピング定義を作成できます。

注※1

マッピング定義ファイル（.mdo）はビジネスプロセス定義画面で作成します。詳細については，「6.4.4　マッピング定義ファイルを利用する際の注意事項」を参照してください。

注※2

［HCSCTE マッピング定義］－［マッピング定義ファイル］の詳細については，「7.3.1　データ変換の定義手順」を参照してください。

注※3

マッピング定義のエクスポートの手順については，「7.14.1　マッピング定義のエクスポート」を参照してください。

6.4.4　マッピング定義ファイルを利用する際の注意事項

マッピング定義ファイルを利用する際の注意事項について次に示します。

サービス定義 XML ファイルのマッピング定義ファイルには，ビジネスプロセス定義画面のデータ変換アクティビティから作成したマッピング定義ファイルのパスしか指定できません。このため，マッピング定義ファイルを利用する場合は，サービス定義 XML ファイルにマッピング定義ファイルのパスを指定しない状態でいったんインポートすることを推奨します。

マッピング定義ファイルを利用する場合のサービス定義 XML ファイルのインポート，およびマッピング定義ファイルの作成の手順を次に示します。

(1)　マッピング定義ファイルのパスを指定していないサービス定義 XML ファイルのインポート

マッピング定義ファイルのパスはサービス定義 XML ファイルの<datatrans-activity>－<mapping-definition-file-path>要素で指定します。

まずは，<mapping-definition-file-path>要素を除いたサービス定義 XML ファイルを作成してインポートします。

<datatrans-activity>－<mapping-definition-file-path>要素の詳細については「6.3.4(13) datatrans-activity」を参照してください。

(2)　マッピング定義ファイルの作成

ビジネスプロセス定義画面でマッピング定義ファイルを作成します。手順を次に示します。

1. ツリービューのサービス定義一覧で，(1)でインポートしたビジネスプロセスを選択しダブルクリックします。
   ビジネスプロセス定義画面が表示されます。

2. 次のどちらかの方法でデータ変換定義画面を開きます。
   - データ変換アクティビティをダブルクリックする。
   - データ変換アクティビティを右クリックして［マッピング定義起動］を選択する。

3. データ変換を定義します。
   データ変換の詳細については，「7.3　データ変換定義」を参照してください。

4. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［マッピング定義保存］を選択し，任意の場所にマッピング定義ファイルを保存します。

(3)　マッピング定義ファイルのパスの指定

(1)で作成したサービス定義 XML ファイルに<mapping-definition-file-path>要素を追加して，(2)で作成したマッピング定義ファイルのパスを指定します。

(4)　サービス定義 XML ファイルの再インポート

(3)で作成したサービス定義 XML ファイルを再度インポートします。

6.5　サービス定義のエクスポート

サービス定義のエクスポートについて説明します。

6.5.1　サービス定義 XML ファイルのエクスポート手順

エクスポートの手順を次に示します。エクスポート先のディレクトリは，事前にユーザが用意する必要があります。ディレクトリ名は任意です。

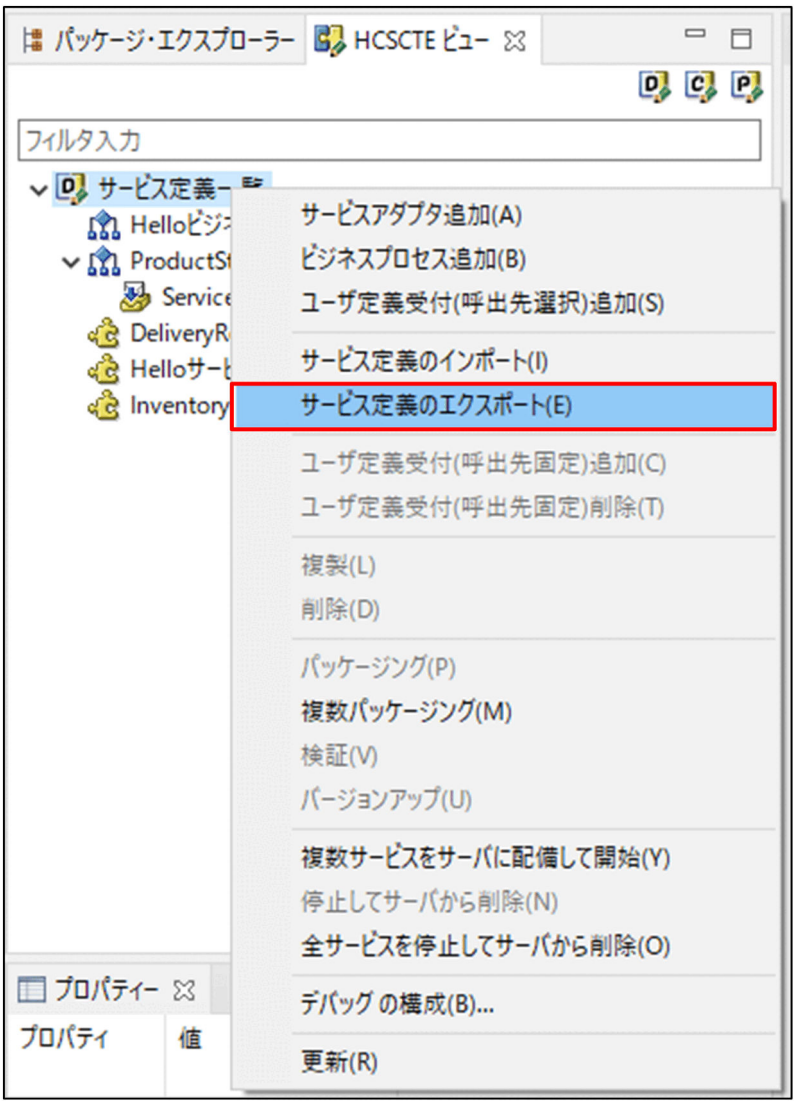
エクスポート先のディレクトリの詳細については，「6.5.4　サービス定義がエクスポートされる出力ディレクトリの構成」を参照してください。

注意事項

gRPC 受付，または gRPC アダプタに対してサービス定義のエクスポートを実行する場合は，エクスポートの前に，gRPC ライブラリ（OSS）のインストールと protoc コマンドの取得を実施してください。gRPC ライブラリ（OSS）のインストールと protoc コマンドの取得についての詳細は，マニュアル「サービスプラットフォーム 解説」の「2.13.1　gRPC プロトコルを利用したシステム」を参照してください。

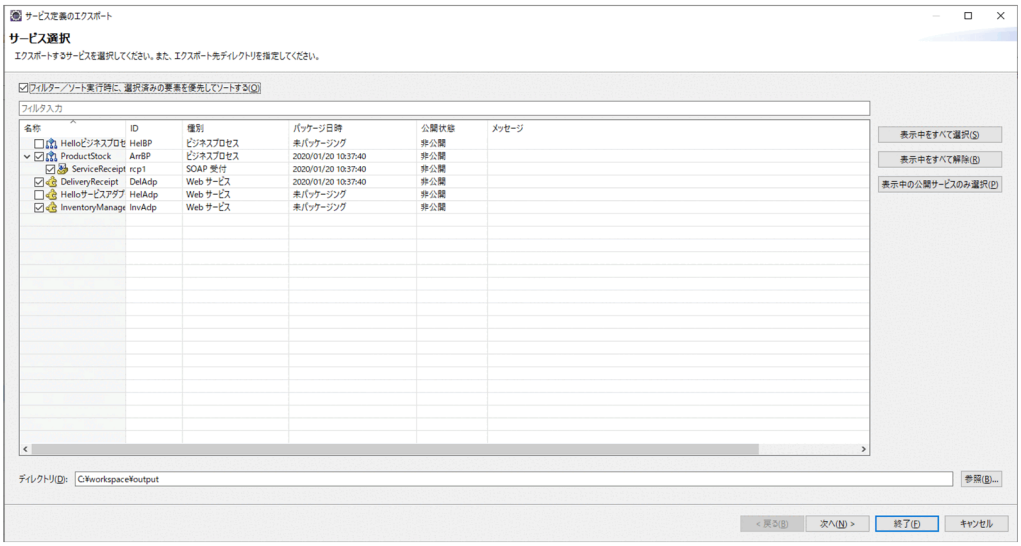
エクスポートの手順を次に示します。

1. ツリービューのサービス定義一覧を右クリックして［サービス定義のエクスポート］を選択します。



サービス定義のエクスポートウィザード（サービス選択ページ）が表示されます。

2. エクスポートする HCSC コンポーネントのチェックボックスをチェックし，エクスポート先のディレクトリを入力または選択します。



「フィルタ入力」にキーワードを入力すると，表示されるサービスをフィルタリングできます。

また，表示されたサービスのチェックボックスをすべてチェックするには［表示中をすべて選択］ボタンをクリックします。逆に，すべてのチェックを解除するには［表示中をすべて解除］ボタンをクリックします。

表示されたサービスの中から公開済みのサービスだけ選択したい場合は［表示中の公開サービスのみ選択］ボタンをクリックしてください。

3. 次のどちらかの操作をします。

動作オプションを変更しないでエクスポートする場合

［終了］ボタンをクリックします。

選択したすべての HCSC コンポーネントのエクスポートが開始されます。動作オプションはデフォルト値，または前回選択した値が適用されます。

動作オプションを変更したい場合

［次へ］ボタンをクリックします。サービス定義のエクスポートウィザード（オプションページ）が表示されます。次の手順に進んでください。



4. 必要に応じて動作オプションを変更します。

動作オプションの項目と，変更できる内容を次に示します。なお，オプションは前回選択した値が保存されています。

それぞれのオプションの動作や，どの場合にどのオプションを利用するかについては，「6.5.2　マッピング定義のエクスポート形式」および「6.5.3　ビジネスプロセスの描画情報ファイルの出力有無」を参照してください。

マッピング定義のエクスポート形式

データ変換ファイルの出力形式を選択します。

・マッピング定義ファイル

マッピング定義ファイル（.mdo）だけを出力します。

・表形式 XML ファイル

表形式 XML ファイル（.xml）だけを出力します。

・マッピング定義ファイルと表形式 XML ファイル

マッピング定義ファイル（.mdo）と表形式 XML ファイル（.xml）の両方を出力します。

ビジネスプロセスの描画情報ファイル

ビジネスプロセスの描画情報ファイルをエクスポートするかどうか選択します。

・エクスポートする

・エクスポートしない

5. **[終了] ボタンをクリックします。**

選択したすべての HCSC コンポーネントのエクスポートが実行されます。

エクスポートが正常終了した場合は，処理完了を伝えるダイアログが表示されます。

異常終了を示すダイアログが表示された場合は，コンソールビューを確認してください。コンソールビューにエラーメッセージが表示されている HCSC コンポーネントについては，エクスポートは実行されていません。コンソールビューのメッセージの内容に従いエラー原因を取り除いてから，再度エクスポートを実行してください。

エクスポート処理が正常に完了した HCSC コンポーネントについては，メッセージは何も表示されません。

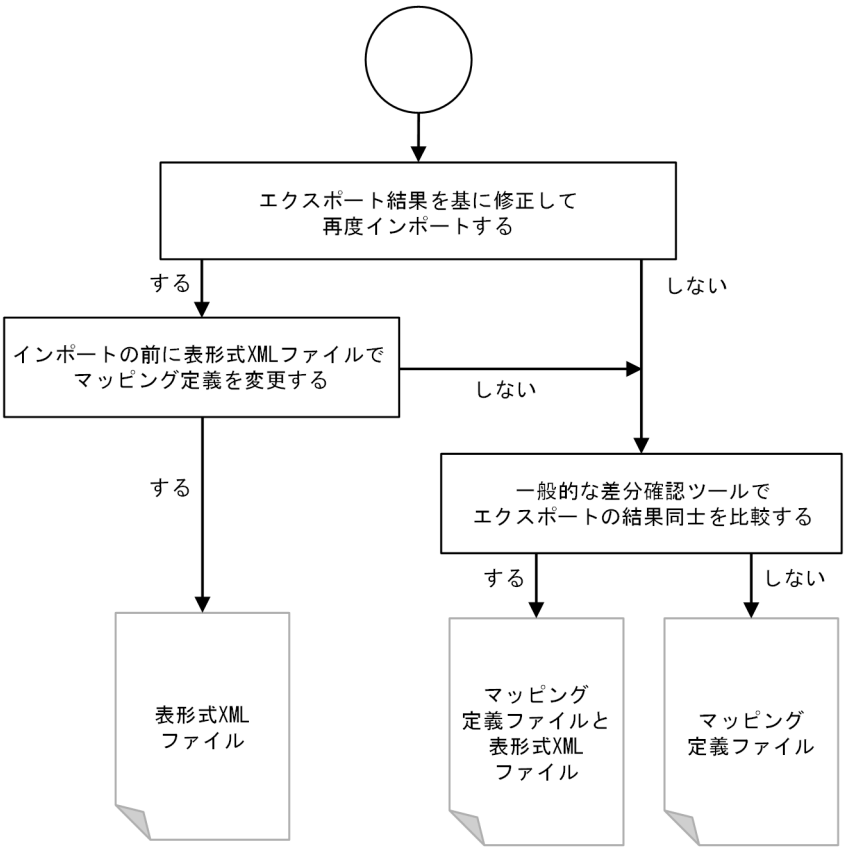
6.5.2 マッピング定義のエクスポート形式

マッピング定義のエクスポート形式を，サービス定義のエクスポートウィザード（オプションページ）で選択できます。

マッピング定義のエクスポート形式のオプションは，次の 3 つの利用ケースの有無によって選択することを推奨します。

- エクスポート結果を基に修正して，再度インポートすることがある
- インポートの前に，表形式 XML ファイルでマッピング定義を変更することがある
- 一般的な差分確認ツールで，エクスポートの結果同士を比較することがある

オプション決定の推奨フローを次に示します。なお，このフローに従わなくても構いません。



それぞれの形式を選択した場合の動作と利用ケースについて，次の表に示します。

表 6-5　マッピング定義のオプション項目ごとの動作と利用ケース（データ変換ファイルの出力形式）

オプションの項目名	動作	利用ケース
マッピング定義ファイル	• マッピング定義ファイル（.mdo）を出力します。 • サービス定義 XML ファイルに対応する mapping-definition-file-path 要素を生成します。	• マッピング定義エディタだけでマッピング定義を修正する場合 • マッピング定義の変更差分の確認が不要な場合
表形式 XML ファイル	• 表形式 XML ファイル（.xml）を出力します。 • サービス定義 XML ファイルに対応する tabular-xml-import 要素を生成します。	表形式 XML ファイルを使用してマッピング定義を修正し，再度インポートする場合
マッピング定義ファイルと表形式 XML ファイル	• マッピング定義ファイル（.mdo）と，表形式 XML ファイル（.xml）を両方出力します。 • サービス定義 XML ファイルに対応する mapping-definition-file-path 要素を生成します。	• マッピング定義エディタだけでマッピング定義を修正する場合 • 一般的な差分確認ツールでマッピング定義の変更を確認したい場合

6.5.3　ビジネスプロセスの描画情報ファイルの出力有無

ビジネスプロセスの描画情報ファイルを出力するかどうかを，サービス定義のエクスポートウィザード（オプションページ）で選択できます。

次の表に示す動作と利用ケースに応じて，ビジネスプロセスの描画情報ファイルの出力有無を選択してください。

表 6-6　マッピング定義のオプション項目ごとの動作と利用ケース（描画情報）

オプションの項目名	動作	利用ケース
エクスポートする	• ビジネスプロセスの描画情報ファイルを出力します • サービス定義 XML ファイルに draw-info-file-path 要素を生成します	• コメントをエクスポートする場合 • コメントを保持したまま，再度インポートする場合 • エディタで変更した座標を保持したまま，再度インポートする場合
エクスポートしない	ビジネスプロセスの描画情報ファイルを出力しません。	• ビジネスプロセスの描画情報をインポートに再利用しない場合 • アクティビティの座標やコメントを無視する場合

6.5.4　サービス定義がエクスポートされる出力ディレクトリの構成

サービス定義のエクスポート実行時に，エクスポートされるディレクトリの構成を次に示します。

参考

エクスポート先ディレクトリ内のサブディレクトリの存在状況に応じて，サブディレクトリが作成されることがあります。その場合，この項で示すディレクトリ構造に沿ったサブディレクトリが作成されます。ただし，作成されるサブディレクトリが空になる場合は，サブディレクトリは作成されません。

表 6-7　サービス定義がエクスポートされるディレクトリの構成

ディレクトリ・ファイル		種別	説明
<ユーザが指定した出力ディレクトリ>		ディレクトリ	サービス定義のエクスポート結果を格納するディレクトリです。 事前にユーザが準備する必要があります。
	BP	ディレクトリ	すべてのビジネスプロセスのエクスポート結果を格納するディレクトリです。 サブディレクトリ以下の構成は「(1)　ディレクトリ「BP」の構成」を参照してください。
	Adapter	ディレクトリ	エクスポートした全サービスアダプタのエクスポート結果を格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
Adapter	ディレクトリ	サブディレクトリ以下の構成は「(2)　ディレクトリ「Adapter」の構成」を参照してください。
Reception	ディレクトリ	エクスポートした全ユーザ定義受付（呼出先選択）のエクスポート結果を格納するディレクトリです。 サブディレクトリ以下の構成は「(3)　ディレクトリ「Reception」の構成」を参照してください。

(1)　ディレクトリ「BP」の構成

BP ディレクトリ以下の構成を次に示します。

表 6-8　BP ディレクトリ以下の構成

ディレクトリ・ファイル	種別	説明
<ビジネスプロセス名>	ディレクトリ	ビジネスプロセスごとのエクスポート結果を格納するディレクトリです。
<ビジネスプロセス名> .xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。
<ビジネスプロセス名>_draw_info.xml	ファイル	ビジネスプロセスの描画情報ファイルです。
Variable	ディレクトリ	ビジネスプロセス内で定義した変数の外部参照ファイルを格納するディレクトリです。
@Global	ディレクトリ	グローバルスコープ内で定義した変数のディレクトリ群を格納するディレクトリです。
<変数名>	ディレクトリ	変数の外部参照ファイルを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	変数の電文フォーマットとして出力される XSD または FDX ファイルです。
<スコープ名>	ディレクトリ	各スコープアクティビティ内で定義した変数のディレクトリ群を格納するディレクトリです。
<変数名>	ディレクトリ	変数の外部参照ファイルを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	変数の電文フォーマットとして出力される XSD または FDX ファイルです。
Activity	ディレクトリ	外部参照ファイルを持つアクティビティ種別の外部参照ファイルを格納するディレクトリです。
Datatrans	ディレクトリ	データ変換アクティビティのディレクトリ群を格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
<アクティビティ名>.mdo	ファイル	マッピング定義ファイルとして出力されるmdoファイルです。
<アクティビティ名>.xml	ファイル	表形式XMLファイルです。
Assign	ディレクトリ	代入アクティビティのディレクトリ群を格納するディレクトリです。
<アクティビティ名>	ディレクトリ	代入アクティビティのデータ群を格納するディレクトリです。
<コピー先変数名>	ディレクトリ	データファイルを格納するディレクトリです。
<ファイル名>	ファイル	固定値（非テキスト）のデータファイル名です。
<部分名>@<コピー先変数名>	ディレクトリ	データファイルを格納するディレクトリです。
<ファイル名>	ファイル	固定値（非テキスト）のデータファイル名です。
Reception	ディレクトリ	ビジネスプロセス内の全ユーザ定義受付のディレクトリ群を格納するディレクトリです。
<ユーザ定義受付名>	ディレクトリ	ユーザ定義受付内の外部参照ファイルを格納するディレクトリです。 受付種別によってサブディレクトリの構造が変わります。それぞれの参照先を次に示します。 • 「(a)　受付種別が「SOAP受付」の場合」 • 「(b)　受付種別が「gPRC受付」の場合」 • 「(c)　受付種別が「カスタム受付」の場合」 • 「(d)　受付種別が「SOAP受付，gPRC受付，またはカスタム受付以外の受付」の場合」

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

受付種別によって，サブディレクトリの構造が変わります。各種別でのディレクトリ構造を次に示します。

(a)　受付種別が「SOAP受付」の場合

SOAP受付以下の構成を次に示します。

表 6-9　SOAP 受付以下の構成

ディレクトリ・ファイル	種別	説明
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
InputHeader	ディレクトリ	要求電文ヘッダの電文フォーマットを格納するディレクトリです。
<ルート要素名>※1	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※2	ファイル	要求電文フォーマットとして出力される XSD ファイルです。
OutputHeader	ディレクトリ	応答電文ヘッダの電文フォーマットを格納するディレクトリです。
<ルート要素名 >※1	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※2	ファイル	応答電文フォーマットとして出力される XSD ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※1

次のどちらかの条件を満たすヘッダ電文が存在した場合，エクスポート時に 2 つ目以降のディレクトリ名に「_連番」が付与されます。この連番は，ディレクトリ名が重複しないように，生成した順に 1 から付けられる自然数の通番です。

・ルート要素名が「同名」のヘッダ電文

「同名」の定義については「6.5.6(1)　エクスポート時の同名ファイルの定義と検証」を参照してください。

・ルート要素名が同じであり，名前空間が異なるヘッダ電文

ルート要素名は，大文字は小文字が区別されます。

注※2

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(b)　受付種別が「gRPC 受付」の場合

gRPC 受付以下の構成を次に示します。

表 6-10　gRPC 受付以下の構成

ディレクトリ・ファイル	種別	説明
ProtoPath	ディレクトリ	proto_path ディレクトリ群を含むディレクトリです。

ディレクトリ・ファイル	種別	説明
ProtoPath	ディレクトリ	proto_path ディレクトリとは，gRPC 受付作成時のウィザード画面の「proto_path」項目に設定したディレクトリを指します。
proto_pathN	ディレクトリ	proto_path ディレクトリです。N は自然数を示します。
<相対ディレクトリ名>	ディレクトリ	proto_path ディレクトリ内の相対ディレクトリです。
<proto ファイル名>	ファイル	gRPC 受付作成時に使用した proto ファイルです。
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
Fault	ディレクトリ	フォルト電文フォーマットを格納するディレクトリです。
<フォルト名>	ディレクトリ	フォルト名に応じたフォルト電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※	ファイル	フォルト電文フォーマットとして出力される XSD ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(c) 受付種別が「カスタム受付」の場合

カスタム受付以下の構成を次に示します。

表 6–11 カスタム受付以下の構成

ディレクトリ・ファイル	種別	説明
<Ear ファイル名>.ear	ファイル	カスタム受付作成時に指定する ear ファイルです。
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
Input	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	要求電文フォーマットとして出力される XSD または FDX ファイルです。
Output	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	応答電文フォーマットとして出力される XSD または FDX ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(d) 受付種別が「SOAP 受付，gPRC 受付，またはカスタム受付以外の受付」の場合

SOAP 受付，gPRC 受付，またはカスタム受付以外の受付以下の構成を次に示します。

表 6–12 SOAP 受付，gPRC 受付，またはカスタム受付以外の受付以下の構成

ディレクトリ・ファイル	種別	説明
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
Input	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	要求電文フォーマットとして出力される XSD または FDX ファイルです。
Output	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	応答電文フォーマットとして出力される XSD または FDX ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(2) ディレクトリ「Adapter」の構成

アダプタ種別によってサブディレクトリの構造が変わります。各種別での Adapter ディレクトリ以下の構造を示します。

(a) アダプタ種別が「SOAP アダプタ」の場合

SOAP アダプタ以下の構成を次に示します。

表 6-13 SOAP アダプタ以下の構成

ディレクトリ・ファイル	種別	説明
<サービスアダプタ名>	ディレクトリ	サービスアダプタごとのエクスポート結果を格納するディレクトリです。
<サービスアダプタ名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
InputHeader	ディレクトリ	要求電文ヘッダの電文フォーマットを格納するディレクトリです。
<ルート要素名>※1	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※2	ファイル	要求電文フォーマットとして出力される XSD ファイルです。
OutputHeader	ディレクトリ	応答電文ヘッダの電文フォーマットを格納するディレクトリです。
<ルート要素名>※1	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※2	ファイル	応答電文フォーマットとして出力される XSD ファイルです。
ClientDefinitionFile	ディレクトリ	クライアント定義ファイルを格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
<クライアント定義ファイル名>	ファイル	クライアント定義ファイルです。

注※1

次のどちらかの条件を満たすヘッダ電文が存在した場合，エクスポート時に 2 つ目以降のディレクトリ名に「_連番」が付与されます。この連番は，ディレクトリ名が重複しないように，生成した順に 1 から付けられる自然数の通番です。

・ルート要素名が「同名」のヘッダ電文

「同名」の定義については「6.5.6(1)　エクスポート時の同名ファイルの定義と検証」を参照してください。

・ルート要素名が同じであり，名前空間が異なるヘッダ電文

ルート要素名は，大文字は小文字が区別されます。

注※2

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(b)　アダプタ種別が「SessionBean アダプタ」の場合

SessionBean アダプタ以下の構成を次に示します。

表 6–14　SessionBean アダプタ以下の構成

ディレクトリ・ファイル	種別	説明
<サービスアダプタ名>	ディレクトリ	サービスアダプタごとのエクスポート結果を格納するディレクトリです。
<サービスアダプタ名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。
ClientDefinitionFile	ディレクトリ	クライアント定義ファイルを格納するディレクトリです。
c4webcl.properties	ファイル	クライアント定義ファイルです。
UserDefinitionClass	ディレクトリ	ユーザ定義クラスを格納するディレクトリです。
<ユーザ定義クラスの JAR ファイル名>.jar	ファイル	ユーザ定義クラスとして出力される JAR ファイルです。

(c)　アダプタ種別が「MDB（WS-R）または MDB（DB キュー）アダプタ」の場合

MDB（WS-R）または MDB（DB キュー）アダプタ以下の構成を次に示します。

表 6–15　MDB（WS-R）または MDB（DB キュー）アダプタ以下の構成

ディレクトリ・ファイル	種別	説明
<サービスアダプタ名>	ディレクトリ	サービスアダプタごとのエクスポート結果を格納するディレクトリです。
<サービスアダプタ名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。

ディレクトリ・ファイル	種別	説明
Input	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	要求電文フォーマットとして出力される XSD または FDX ファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(d) アダプタ種別が「gRPC アダプタ」の場合

gRPC アダプタ以下の構成を次に示します。

表 6-16 gRPC アダプタ以下の構成

ディレクトリ・ファイル	種別	説明
<サービスアダプタ名>	ディレクトリ	サービスアダプタごとのエクスポート結果を格納するディレクトリです。
<サービスアダプタ名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。
ProtoPath	ディレクトリ	proto_path ディレクトリ群を含むディレクトリです。 proto_path ディレクトリとは，gRPC アダプタ作成時のウィザード画面の「proto_path」項目に設定したディレクトリを指します。
proto_pathN	ディレクトリ	proto_path ディレクトリです。N は自然数を示します。
<相対ディレクトリ名>	ディレクトリ	proto_path ディレクトリ内の相対ディレクトリです。
<proto ファイル名>	ファイル	gRPC アダプタ作成時に使用した proto ファイルです。
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>※1	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
Fault	ディレクトリ	フォルト電文フォーマットを格納するディレクトリです。
<フォルト名>	ディレクトリ	フォルト名に応じたフォルト電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd※2	ファイル	フォルト電文フォーマットとして出力される XSD ファイルです。

ディレクトリ・ファイル					種別	説明
			CustomDefinitionFile		ディレクトリ	独自定義ファイルを格納するディレクトリです。
				<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※1

<オペレーション名>が「同名」と判定された場合は，エクスポート時に 2 つ目以降のディレクトリ名に「_連番」が付与されます。この連番は，ディレクトリ名が重複しないように，生成した順に 1 から付けられる自然数の通番です。

「同名」の定義については「6.5.6(1)　エクスポート時の同名ファイルの定義と検証」を参照してください。

注※2

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(e)　アダプタ種別が「汎用カスタムアダプタ」，またはその他のアダプタの場合

ここで説明しているその他のアダプタとは，次のアダプタのことです。

- DB アダプタ
- TP1 アダプタ
- ファイルアダプタ
- Object Access アダプタ
- Message Queue アダプタ
- FTP アダプタ
- ファイル操作アダプタ
- メールアダプタ
- HTTP アダプタ
- コマンドアダプタ
- SFTP アダプタ
- Kafka アダプタ

汎用カスタムアダプタ，またはその他のアダプタ以下の構成を次に示します。

表 6-17　汎用カスタムアダプタ，またはその他のアダプタ以下の構成

ディレクトリ・ファイル				種別	説明
		<サービスアダプタ名>		ディレクトリ	サービスアダプタごとのエクスポート結果を格納するディレクトリです。
			<サービスアダプタ名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。
			Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。

ディレクトリ・ファイル	種別	説明
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
Input	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	要求電文フォーマットとして出力される XSD または FDX ファイルです。
Output	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	応答電文フォーマットとして出力される XSD/FDX ファイルです。
Fault	ディレクトリ	フォルト電文フォーマットを格納するディレクトリです。
<フォルト名>	ディレクトリ	フォルト名に応じたフォルト電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	フォルト電文フォーマットとして出力される XSD または FDX ファイルです。
UtilityClass	ディレクトリ	ユーティリティクラスを格納するディレクトリです。
<ユーティリティクラスの JAR ファイル名>.jar	ファイル	ユーティリティクラスとして出力される JAR ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

(3) ディレクトリ「Reception」の構成

Reception ディレクトリ以下の構成を示します。

表 6-18 Reception ディレクトリ以下の構成

ディレクトリ・ファイル	種別	説明
<ユーザ定義受付名>	ディレクトリ	各ユーザ定義受付（呼出先選択）のエクスポート結果を格納するディレクトリです。
<ユーザ定義受付名>.xml	ファイル	サービス定義 XML ファイルとして出力される XML ファイルです。

ディレクトリ・ファイル	種別	説明
Operation	ディレクトリ	オペレーション一覧の外部参照ファイルを格納するディレクトリです。
<オペレーション名>	ディレクトリ	オペレーション内の外部参照ファイルを格納するディレクトリです。
Input	ディレクトリ	要求電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	要求電文フォーマットとして出力される XSD または FDX ファイルです。
Output	ディレクトリ	応答電文フォーマットを格納するディレクトリです。
<電文フォーマットファイル名>.xsd または.fdx※	ファイル	応答電文フォーマットとして出力される XSD または FDX ファイルです。
CustomDefinitionFile	ディレクトリ	独自定義ファイルを格納するディレクトリです。
<独自定義ファイル名>	ファイル	独自定義ファイルとして出力されるファイルです。

注※

ファイル名は，電文フォーマットを取り込んだ際にユーザが設定したファイル名となります。電文フォーマットファイル内で外部ファイルを参照している場合，参照先ファイル名は参照元ファイル内で指定していたファイル名となります。

6.5.5　エクスポート実行時に削除されるディレクトリ

エクスポートを実行する際に，指定したエクスポート先ディレクトリ内のサブディレクトリの存在状況に応じて，サブディレクトリが削除される場合があります。

サブディレクトリが削除されるケースを次に示します。

表中の「同名」の定義については「6.5.6(1)　エクスポート時の同名ファイルの定義と検証」を参照してください。

表 6-19　サブディレクトリの削除有無

ディレクトリ名	サブディレクトリが存在する場合の削除有無
<ユーザが指定した出力ディレクトリ>	-
BP	削除されません。
<ビジネスプロセス名>	同名のビジネスプロセスをエクスポートする場合は削除されます。
Adapter	削除されません。
<サービスアダプタ名>	同名のサービスアダプタをエクスポートする場合は削除されます。
Reception	削除されません。

ディレクトリ名			サブディレクトリが存在する場合の削除有無
		<ユーザ定義受付名>	同名のユーザ定義受付（呼出先選択）をエクスポートする場合は削除されます。
	上記以外のディレクトリ		削除されません。

（凡例）

-：該当しません。

6.5.6　注意事項（サービス定義のエクスポート）

サービス定義のエクスポートの注意事項を次に示します。

(1)　エクスポート時の同名ファイルの定義と検証

サービス定義 XML をエクスポートする際は，サービス名やアクティビティ名などを利用して，エクスポート先のディレクトリやファイル名を作成します。

各サービスの編集エディタやサービス定義 XML 上では，アルファベットの大文字と小文字を区別しますが，Windows のファイルシステムでは，大文字と小文字を区別しません。

例えば，各サービスの編集エディタやサービス定義 XML 上では「ProcessA」と「processa」という 2 つのビジネスプロセスを作成できますが，Windows のファイルシステムでは「ProcessA」と「processa」を同一階層に作成できません。

このため，サービス定義のエクスポートでは大文字と小文字を区別しないで，「ProcessA」と「processa」を同名として扱います。エクスポート時に，ディレクトリ名およびファイル名として利用する文字列に対して同名が存在しないかどうかを検証し，同名が存在する場合は検証でエラーとします。

検証範囲，および範囲内で検証する項目を次に示します。

表 6–20　同名の検証範囲および範囲内で検証する項目

検証範囲			検証項目
全ビジネスプロセス			ビジネスプロセス名
	各ビジネスプロセス		ユーザ定義受付（呼出先固定）名
			データ変換アクティビティ名※1
			スコープアクティビティ名
			代入アクティビティ名
		各スコープ	変数名※2
		各代入アクティビティ	コピー元が固定値（非テキスト）の場合，次の項目を検証します。 • コピー先変数名

検証範囲	検証項目
各代入アクティビティ	• コピー先変数名と部分名の組
各ユーザ定義受付（呼出先固定）	オペレーション名
全サービスアダプタ	サービスアダプタ名
各サービスアダプタ	オペレーション名
各オペレーション	フォルト名
全ユーザ定義受付（呼出先選択）	ユーザ定義受付（呼出先選択）名
各ユーザ定義受付（呼出先選択）	オペレーション名
各変数／要求電文／応答電文／フォルト電文	電文フォーマットファイルのファイル名※3

注※1

マッピング定義が存在するデータ変換アクティビティが対象です。

注※2

XML 型変数および non-XML 型変数が対象です。

注※3

電文フォーマットファイル内で外部ファイルを参照している場合，参照ファイルの名前は電文フォーマットファイル名として扱います。また，各ファイルのファイルパスは考慮しないで，ファイル名だけを検証の対象とします。例えば，電文フォーマットファイルとして「¥C¥¥Sample1¥FormatA.xsd」を指定し，そのファイル内で「¥E¥Sample2¥formata.xsd」を参照している場合，電文フォーマットファイル名は「FormatA」と「formata」となるため，2 つのファイルは同名としてエクスポート時に検証エラーとなります。

(2) SOAP 受付または SOAP アダプタの WSDL ファイル，および SessionBean アダプタの EAR ファイルのエクスポートについて

SOAP 受付を含むビジネスプロセス，または SOAP アダプタをエクスポートする場合，WSDL ファイルは出力されません。同様に，SessionBean アダプタをエクスポートする場合，EAR ファイルは出力されません。そのため，エクスポートで取得したサービス定義 XML ファイルを使用して再度インポートする場合は，前回インポート時に指定した WSDL ファイルのパスを wsdl-file-path 要素に，ear ファイルのパスを ear-file-path 要素に指定してください。

(3) ディレクトリ/ファイルのパスの長さに関する制限について

Windows では，パスが長いディレクトリやファイルを扱えないアプリケーションが存在します。そのため，エクスポート先ディレクトリとして指定するディレクトリのパスが長い場合，サービス定義のエクスポートで作成したディレクトリやファイルを開けないおそれがあります。

(4) サービス定義 XML ファイルで指定できる値の範囲かどうかの検証

HCSC コンポーネントの設定値が，サービス定義 XML ファイルで指定できる値の範囲かどうかをエクスポート実行時に検証します。検証する項目および検証エラーとなる設定値を次に示します。

表 6-21　エクスポート時に検証エラーとなる設定値（ビジネスプロセス）

アクティビティ種別	項目	検証エラーとなる値
繰り返しアクティビティ	最大繰り返し回数	1 から 2,147,483,647 までの整数以外の値

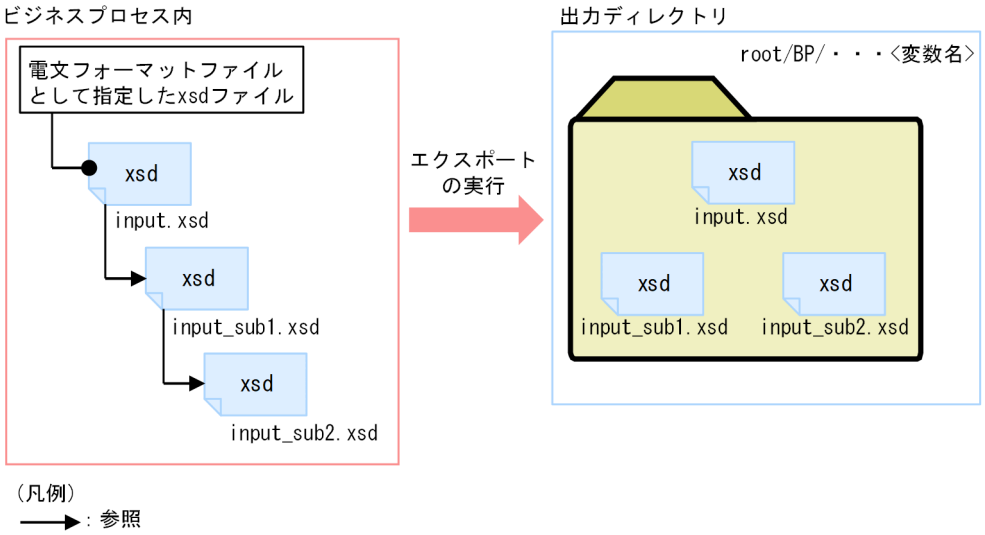
表 6-22　エクスポート時に検証エラーとなる設定値（サービスアダプタ）

アダプタ種別	項目	検証エラーとなる値
すべての種別	最大インスタンス数	0 から 65,535 までの整数以外の値
MDB（WS-R）アダプタ	最大メッセージ数	0 から 65,535 までの整数以外の値

ユーザ定義受付には，検証エラーとなる設定値がありません。

(5)　電文フォーマットファイルの参照ファイルのエクスポート

HCSC コンポーネント内で指定した電文フォーマットファイル内で外部ファイルを参照している場合，参照先のファイルもエクスポートされます。



(6)　エクスポート先のディレクトリ名およびファイル名のリネームについて

サービス定義のエクスポートでは，エクスポート先のディレクトリ名およびファイル名に，サービス名やアクティビティ名などを使用します。サービス名やアクティビティ名などが，Windows のファイルシステムで使用できない文字列の場合，エクスポート先のディレクトリ名およびファイル名の一部がリネームされることがあります。

(7)　サブシーケンスの定義方法

サブシーケンスは sub-sequence 要素，または sub-sequence-ref 要素のどちらかを利用して定義します。サービス定義のエクスポートによって作成するサービス定義 XML ファイルでは，サブシーケンスは sub-sequence-ref 要素を利用します。

シーケンスおよびサブシーケンスの考え方については，「6.3.1　サービス定義 XML ファイルでのビジネスプロセス定義の考え方」を，sub-sequence 要素については「6.3.4(7)　sub-sequence」を参照してください。

(8)　SOAP 受付または SOAP アダプタのヘッダ電文の定義方法

SOAP 受付または SOAP アダプタで定義された要求または応答ヘッダ電文は，WSDL ファイルで定義されたヘッダ電文であっても，すべて input-headers 要素または output-headers 要素の子要素に出力されます。

(9)　gRPC 受付または gRPC アダプタのオペレーション・フォルトの定義方法

サービス定義のエクスポート実行時，gRPC 受付または gRPC アダプタで定義されたオペレーションは，すべてサービス定義 XML ファイルの operation 要素に出力されます。また，gRPC 受付または gRPC アダプタで定義されたフォルトは，サービス定義 XML ファイルの fault-message 要素に出力されます。

7 データ変換の定義

この章では，データ変換に必要な電文フォーマット定義ファイルおよびデータ変換定義について説明します。

7.1　データ変換に必要なファイルおよび定義

データ変換に必要なファイルの作成および定義の手順を次に示します。

1. 電文フォーマット定義ファイルを作成します。
   詳細については，「7.2　電文フォーマット定義ファイルの作成」を参照してください。
2. データ変換を定義します。
   詳細については，「7.3　データ変換定義」を参照してください。

7.2　電文フォーマット定義ファイルの作成

データ変換定義の基となる電文フォーマット定義ファイルを作成する方法，および設定する方法について説明します。また，電文フォーマット定義ファイルおよびバイナリ電文フォーマットに関する注意事項についても説明します。

7.2.1　電文フォーマット定義ファイルの作成

電文フォーマット定義ファイルには，変換元の電文フォーマット定義ファイルと変換先の電文フォーマット定義ファイルがあります。電文フォーマット定義ファイルの種類および作成方法については，「4.　電文フォーマットの作成」を参照してください。

データ変換の定義に利用する電文フォーマット定義ファイルは，「2.6.5　XML スキーマの適用範囲」に示す条件に従って作成してください。

7.2.2　電文フォーマット定義ファイルの設定

作成した電文フォーマット定義ファイルは，サービスアダプタ定義画面またはビジネスプロセス定義画面で設定します。

サービスアダプタ定義画面での設定方法については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.3　サービスアダプタの内容の定義」を参照してください。ビジネスプロセス定義画面での設定方法については，「5.3　ビジネスプロセスの内容の定義」を参照してください。

7.2.3　電文フォーマット定義ファイル作成時の注意事項

画面に表示されない要素

annotation，appinfo，documentation，および notation は，変換元スキーマツリービューアおよび変換先スキーマツリービューアにツリー表示されません。

sequence 要素または choice 要素の画面表示

sequence 要素または choice 要素は，定義する XML スキーマの内容によって，変換元スキーマツリービューアおよび変換先スキーマツリービューアに次のように表示されます。

- #(sequence)

  sequence 要素または choice 要素下の，出現回数が 1 回に固定されている sequence 要素（コンポジタ）を示します。

- #(choice)

  sequence 要素または choice 要素下の，出現回数が 1 回に固定されている choice 要素（コンポジタ）を示します。

- #anonymous
  出現回数が 1 回に固定されていない sequence 要素，choice 要素，または all 要素（コンポジタ）を示します。

7.2.4 セパレータが設定されているバイナリ電文フォーマットに関する注意事項

セパレータが設定されているバイナリ電文フォーマットを利用する場合，次の点に注意が必要です。

- 出現回数に［データの終端まで］を定義した場合で，次の 2 つの条件を満たす場合，繰り返しによる空要素の生成は 1 つだけになります。
  - 弟要素が存在しない。
  - 中間区切り文字の直後に繰り返しの終了文字が出現する。
- 入力するバイナリデータにセパレータ文字と同じ値のバイトデータが含まれている場合（マルチバイト文字の構成バイトの一部分である場合も含む），バイナリデータの解析に失敗します。
- エスケープ文字を付加しなくてもデータと認識される場合があるので注意してください。
  例：開始文字と同じ文字列が，開始文字の直後に設定されている場合
- セパレータが設定されているバイナリ電文フォーマット定義内では，ビット列型の単純内容要素とサイズノード指定された複合内容要素を使用できません。
- 次の文字コードを指定しないでください。
  - KEIS+EBCDIK
  - KEIS+EBCDIC
  - IBM_CODE+EBCDIC(LATIN)
  - IBM_CODE+EBCDIC(KANA)
- セパレータの文字コードは，個別文字コードの設定に関係なく，フォーマット定義の文字コードとなります。
- 文字コードに JEF+EBCDIC，または JEF+EBCDIK を指定する場合，各要素のデータは半角シフト状態で終わるようにしてください。全角シフト状態で終わる場合，セパレータの解析ができません。

7.2.5 任意形式（any 形式）の電文フォーマットに関する注意事項

任意形式（any 形式）を電文フォーマット定義ファイルに設定した場合に，設定の一部が無視されます。

無視される設定を次の表に示します。

表 7-1　任意形式（any 形式）で無視される設定

対象となるコンポーネント	無視される設定
TP1/RPC 受付	独自定義ファイル(cscurecptp1rpc.properties) • urecp-tp1rpc.dt-skip 独自定義ファイル(csc_owncodeconvert.properties) • すべての設定
TP1 アダプタ HTTP アダプタ 汎用カスタムアダプタ	HCSC サーバランタイム定義ファイル(cscsvconfig.properties) • xmltelegram-maxcache-num • telegram-validation • xmltelegram-namespace-complement • telegram-undefined-character-code 独自定義ファイル(csccustomadapter.properties) • custom-adapter.dt-skip 独自定義ファイル(csc_owncodeconvert.properties) • すべての設定

7.3　データ変換定義

データ変換定義画面で，変換元の電文フォーマット定義ファイルおよび変換先の電文フォーマット定義ファイルを設定して，データ変換を定義します。データ変換定義画面で作成できる定義を次に示します。

データ変換定義

サービスアダプタやビジネスプロセスの定義で，各種フォーマット間のデータ変換パターンを定義することをデータ変換定義といいます。データ変換は，N 対 1，または 1 対 1 で定義できます。

データ変換定義は，XML スキーマファイルまたはバイナリフォーマット定義ファイルとデータ変換定義ファイルから構成されています。

マッピング定義

データ変換定義のデータ変換パターンの情報をマッピング定義といいます。

この情報をエクスポートしたものをマッピング定義ファイル（拡張子：.mdo）と呼びます。マッピング定義ファイルには，データ変換定義画面の編集結果が保存されます。

データ変換を定義する手順について説明します。

7.3.1　データ変換の定義手順

データ変換定義画面でデータ変換を定義する手順を次に示します。

(1)　新規にデータ変換を定義する場合

新規にデータ変換を定義する手順を次に示します。なお，データ変換定義画面を表示する方法によって，データ変換を定義する手順が異なります。また，データ変換を 1 対 1 で定義する場合と，N 対 1 で定義する場合とで，表示される画面および手順が異なります。

(a)　サービスアダプタ定義画面，またはビジネスプロセス定義画面のデータ変換アクティビティから表示する方法

1. **サービスアダプタ定義画面，またはビジネスプロセス定義画面のデータ変換アクティビティから，データ変換定義画面を表示します。**

   データ変換定義画面が表示されたあと，［ルート要素選択］ダイアログが表示されます。

   データ変換定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。［ルート要素選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.3　ルート要素選択ダイアログ」を参照してください。

2. **すべてのスキーマ論理名のルート要素を選択します。**

   ［OK］ボタンがクリックできる状態になります。

3. [OK] ボタンをクリックします。

変換元スキーマツリービューアおよび変換先スキーマツリービューアに，選択したルート要素のスキーマがツリー表示されます。

4. マッピングします。

マッピングについては，「7.4　マッピングする」を参照してください。

5. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，[検証] を選択します。

検証が実行されます。

注意事項

手順 3.で［キャンセル］ボタンをクリックした場合，エラーメッセージが表示されます。エラーメッセージが表示されたダイアログで［OK］ボタンをクリックすると，再度［ルート要素選択］ダイアログが表示されます。

データ変換アクティビティのマッピング定義からデータ変換定義を出力したい場合は，上記の手順に加えて，次に示す手順を実行することで実現できます。

1. データ変換定義画面の適当な場所で右クリックして，[データ変換定義生成] を選択します。

データ変換定義を保存するディレクトリおよびファイル名を指定するダイアログが表示されます。

2. データ変換定義を保存するディレクトリおよびファイル名を指定して，[保存] ボタンをクリックします。

検証が実行されます。

検証に成功した場合は「データ変換定義を生成しました。」のメッセージダイアログが表示され，データ変換定義が保存されます。検証に失敗した場合はエラーメッセージのダイアログが表示されます。

(b)　Eclipse のメニューから表示する場合

データ変換は，1 対 1，または N 対 1 で定義できます。それぞれの手順を次に示します。

1 対 1 のデータ変換の場合

1. Eclipse のメニューから，[ファイル] － [新規] － [その他] を選択します。

ウィザードを選択するダイアログが表示されます。

2. [HCSCTE マッピング定義] － [マッピング定義ファイル] を選択し，[次へ] ボタンをクリックします。

[新規マッピング定義ファイル] ページが表示されます。

3. マッピング定義ファイルを保存するディレクトリおよびマッピング定義ファイル名を入力し，[次へ] ボタンをクリックします。

[変換元・変換先スキーマファイル設定] ページが表示されます。

4. [1 対 1 変換] ラジオボタンを選択し，変換元スキーマおよび変換先スキーマを選択して，[終了] ボタンをクリックします。

   データ変換定義画面にマッピング定義エディタが表示されたあと，[ルート要素選択] ダイアログが表示されます。データ変換定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。[ルート要素選択] ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.3　ルート要素選択ダイアログ」を参照してください。

5. すべてのスキーマ論理名のルート要素を選択します。

   [OK] ボタンがクリックできる状態になります。

6. [OK] ボタンをクリックします。

   変換元スキーマツリービューアおよび変換先スキーマツリービューアに，選択したルート要素のスキーマがツリー表示されます。

7. マッピングします。

   マッピングについては，「7.4　マッピングする」を参照してください。

8. データ変換定義画面の適当な場所で右クリックして，[データ変換定義生成] を選択します。

   データ変換定義を保存するディレクトリおよびファイル名を指定するダイアログが表示されます。

9. データ変換定義を保存するディレクトリおよびファイル名を指定して，[保存] ボタンをクリックします。

   検証が実行され，データ変換定義ファイルが保存されます。

注意事項

手順 6.で［キャンセル］ボタンをクリックした場合，エラーメッセージが表示されます。エラーメッセージが表示されたダイアログで［OK］ボタンをクリックすると，再度［ルート要素選択］ダイアログが表示されます。

N 対 1 のデータ変換の場合

1. Eclipse のメニューから，[ファイル] － [新規] － [その他] を選択します。

   ウィザードを選択するダイアログが表示されます。

2. [HCSCTE マッピング定義] － [マッピング定義ファイル] を選択し，[次へ] ボタンをクリックします。

   [新規マッピング定義ファイル] ページが表示されます。

3. マッピング定義ファイルを保存するディレクトリおよびマッピング定義ファイル名を入力し，[次へ] ボタンをクリックします。

   [変換元・変換先スキーマファイル設定] ページが表示されます。

4. [N 対 1 変換] ラジオボタンを選択し，[追加] ボタンをクリックします。

   [変換元スキーマファイル設定] ダイアログが表示されます。

5. **変換元スキーマおよびスキーマ論理名を指定して，［OK］ボタンをクリックします。**

   ［変換元・変換先スキーマファイル設定］ページの［変換元スキーマ］欄にファイルパスとスキーマ論理名が表示されます。

6. **手順 4～手順 5 を繰り返して変換元スキーマを追加します。このとき，［N 対 1 変換］ラジオボタンを再度選択する必要はありません。**

7. **変換元スキーマの追加が終わったら，変換先スキーマを選択して［終了］ボタンをクリックします。**

   データ変換定義画面にマッピング定義エディタが表示されたあと，［ルート要素選択］ダイアログが表示されます。データ変換定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。［ルート要素選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.3　ルート要素選択ダイアログ」を参照してください。

8. **すべてのスキーマ論理名のルート要素を選択します。**

   ［OK］ボタンがクリックできる状態になります。

9. **［OK］ボタンをクリックします。**

   変換元スキーマツリービューアおよび変換先スキーマツリービューアに，選択したルート要素のスキーマがツリー表示されます。

10. **マッピングします。**

    マッピングについては，「7.4　マッピングする」を参照してください。

11. **データ変換定義画面の適当な場所で右クリックして，［データ変換定義生成］を選択します。**

    データ変換定義を保存するディレクトリおよびファイル名を指定するダイアログが表示されます。

12. **データ変換定義を保存するディレクトリおよびファイル名を指定して，［保存］ボタンをクリックします。**

    検証が実行され，データ変換定義ファイルが保存されます。

注意事項

手順 8.で［キャンセル］ボタンをクリックした場合，エラーメッセージが表示されます。エラーメッセージが表示されたダイアログで［OK］ボタンをクリックすると，再度［ルート要素選択］ダイアログが表示されます。

(2)　定義済みのデータ変換を編集する場合

定義済みのデータ変換を編集する手順を次に示します。

1. **サービスアダプタ定義画面，またはビジネスプロセス定義画面のデータ変換アクティビティから，データ変換定義画面を表示します。**

   定義済みの状態のデータ変換定義画面が表示されます。

   データ変換定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

2. 必要に応じてルート要素を変更します。
   ルート要素を変更する方法については，「(3)　ルート要素を変更する場合」を参照してください。

3. マッピングを編集します。
   マッピングについては，「7.4　マッピングする」を参照してください。

4. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［検証］を選択します。
   検証が実行されます。

(3)　ルート要素を変更する場合

ルート要素は，ルート要素を選択してスキーマをツリー表示させたあとに，変更することができます。

ルート要素を変更する方法を次に示します。なお，ルート要素を変更する方法は，データ変換定義画面の表示方法によって異なります。また，マッピングを 1 対 1 で定義している場合と，N 対 1 で定義している場合とで，表示される画面および手順が異なります。

注意事項

DB アダプタの定義の場合，変換元（DB アダプタ側）のルート要素は変更しないでください。ルート要素を変更した場合，エラーになります。

(a)　サービスアダプタ定義画面，またはビジネスプロセス定義画面のデータ変換アクティビティからデータ変換定義画面を表示した場合

1. 変換元スキーマツリービューアまたは変換先スキーマツリービューアのスキーマ論理名を右クリックして，［ルート要素選択］を選択します。
   ［ルート要素選択］ダイアログが表示されます。

2. スキーマ論理名のルート要素を変更します。

3. ［OK］ボタンをクリックします。
   変換元スキーマツリービューアおよび変換先スキーマツリービューアに，変更したルート要素のスキーマがツリー表示されます。
   変更前の状態でマッピングしていた場合，変更前の編集内容が破棄されることを知らせるダイアログが表示されます。変更前の編集内容を破棄して，新しく選択したルート要素のスキーマをツリー表示する場合は，［OK］ボタンをクリックします。

(b)　Eclipse のメニューからデータ変換定義画面を表示した場合

1 対 1 でマッピングを定義している場合と，N 対 1 でマッピングを定義している場合のそれぞれの手順を次に示します。

1 対 1 でマッピングを定義している場合

1. 変換元スキーマツリービューアまたは変換先スキーマツリービューアのスキーマ論理名を右クリックして，［ルート要素選択］を選択します。

   ［変換元・変換先スキーマファイル設定］ダイアログが表示されます。

2. 変換元スキーマおよび変換先スキーマを選択します。

   ［変換元・変換先スキーマファイル設定］ダイアログには，前回指定したファイルが設定されています。前回指定したファイルから変更しない場合，ファイルを設定し直す必要はありません。

3. ［OK］ボタンをクリックします。

   ［ルート要素選択］ダイアログが表示されます。

4. スキーマ論理名のルート要素を変更します。

5. ［OK］ボタンをクリックします。

   変換元スキーマツリービューアおよび変換先スキーマツリービューアに，変更したルート要素のスキーマがツリー表示されます。

   変更前の状態でマッピングしていた場合，変更前の編集内容が破棄されることを知らせるダイアログが表示されます。変更前の編集内容を破棄して，新しく選択したルート要素のスキーマをツリー表示する場合は，［OK］ボタンをクリックします。

N 対 1 でマッピングを定義している場合

1. 変換元スキーマツリービューアまたは変換先スキーマツリービューアのスキーマ論理名を右クリックして，［ルート要素選択］を選択します。

   N 対 1 用の［変換元・変換先スキーマファイル設定］ダイアログが表示されます。

   ［変換元・変換先スキーマファイル設定］ダイアログには，前回指定したファイルが設定されています。前回指定したファイルから変更しない場合，ファイルを設定し直す必要はありません。

2. 編集したい変換元スキーマを選択し［編集］ボタンをクリックします。

   ［変換元スキーマファイル設定］ダイアログが表示されます。

3. 変換元スキーマを選択して［OK］ボタンをクリックします。

   ［変換元・変換先スキーマファイル設定］ダイアログの［変換元スキーマ欄］に変更後のファイルパスとスキーマ論理名が表示されます。

4. 変換元スキーマの変更が終わったら，変換先スキーマを選択します。

5. ［OK］ボタンをクリックします。

   ［ルート要素選択］ダイアログが表示されます。

6. スキーマ論理名のルート要素を変更します。

7. ［OK］ボタンをクリックします。

   変換元スキーマツリービューアおよび変換先スキーマツリービューアに，変更したルート要素のスキーマがツリー表示されます。

変更前の状態でマッピングしていた場合，変更前の編集内容が破棄されることを知らせるダイアログが表示されます。変更前の編集内容を破棄して，新しく選択したルート要素のスキーマをツリー表示する場合は，［OK］ボタンをクリックします。

7.3.2　電文フォーマットを変更した場合の定義手順

電文フォーマットを変更した場合，データ変換を定義し直す必要があります。データ変換を定義し直すときは，以前に作成したデータ変換定義を利用できます。新規に定義する必要はありません。

注意事項

データ変換定義が編集中の電文フォーマットを変更した場合は，いったんデータ変換定義画面を終了してから，再度データ変換定義画面を表示してください。編集中のデータ変換定義画面を終了しないまま，新しくデータ変換定義画面を表示した場合，電文フォーマットの変更が一部反映されないことがあります。

電文フォーマットを変更してデータ変換を定義し直す手順を次に示します。

1. **サービスアダプタ定義画面，またはビジネスプロセス定義画面のデータ変換アクティビティから，データ変換定義画面を表示します。**

   データ変換定義画面が表示される前に，電文フォーマットの変更を反映するかを確認するダイアログが表示されます。［OK］ボタンをクリックすると，次の画面が表示されます。

   - ルート要素を変更または削除した場合

     データ変換定義画面が表示される前に，［ルート要素選択］ダイアログが表示されます。［ルート要素選択］ダイアログでルート要素を選択し，［OK］ボタンをクリックすると，データ変換定義画面が表示されます。［ルート要素選択］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.3　ルート要素選択ダイアログ」を参照してください。

   - ルート要素を変更または削除していない場合

     データ変換定義画面が表示されます。

   表示されたデータ変換定義画面には，電文フォーマットの変更が反映されています※。

2. **必要に応じてマッピングします。**

   マッピングについては，「7.4　マッピングする」を参照してください。

3. **データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［検証］を選択します。**

   検証が実行されます。

注※

電文フォーマットを変更する前にマッピングした内容がコピーされたデータ変換定義画面が表示されます。ただし，コピーされるのは，変更前と変更後の電文フォーマットを比較し，類似度の優先度が 1 で

あるものだけです。マッピング定義のコピーおよび類似度の判定については，「7.8　マッピング定義をコピーする」を参照してください。

7.3.3　データ変換定義の注意事項

(1)　開発環境における注意事項

要素数が 2,000 以上あるなど，大きな電文フォーマットのスキーマファイルを使用してデータ変換を定義する場合，データ変換定義の検証およびパッケージングを実行すると，メモリの不足によってエラーが発生することがあります。

エラーが発生した場合は，<Eclipse のインストールディレクトリ>¥eclipse¥eclipse.ini ファイル内の次のオプションを変更し，Eclipse を再起動してから実行し直してください。

- -XmxNm（N は 1 以上の整数（MB 単位））

また，次のオプションを追加してください。

- -XssNm（N は 1 以上の整数（MB 単位））

なお，次のオプションは指定しないでください。

- -Xverify:none

(2)　実行環境における注意事項

- データ変換定義の構造に依存して，大規模の入力電文を変換する際に，メモリを大量に確保する場合があります。条件として，次に示すどちらかのマッピング定義を利用する場合が該当します。
  - マッピングされている変換先ノード自身が省略できる，または，その先祖ノードが省略できる個所を含むマッピング定義。
  - 電文フォーマットに要素が多数定義されており，マッピング線が引かれている。

  このため，実行時にメモリ不足となる場合は，入力電文のデータ量を減らすか，ヒープサイズを増やしてください。

  なお，上記のマッピング定義によるデータ変換処理には Java の制限により変換できる限界が存在します。限界以上のデータを変換すると，変換処理が途中でエラー終了します。この場合には，エラーメッセージの内容に従い，入力電文のデータ量を減らすことで対処してください。
- HCSC サーバランタイム定義ファイルで，transformdef-maxcache-num を指定し，データ変換定義をキャッシュしている場合，Metaspace 領域を消費します。このため，実行時に Metaspace 領域不足となる場合は，Metaspace 領域サイズを増やしてください。
- データ変換定義を用いたデータ変換処理で扱える XML 形式のデータは，XML 標準仕様のバージョン「XML 1.0 Third Edition」に準拠していることが，前提条件となります。データ変換処理の概要につ

いては，マニュアル「サービスプラットフォーム 解説」の「付録 B　データ変換機能の処理概要」を参照してください。

XML 標準仕様に関する内容の一部を要約したものを次に示します。詳細は，XML の仕様書を参照してください。

- 要素値に含まれる改行コード（CR（#xD）または CR + LF（#xD #xA））は，DOM へのパース処理によって，#xA に正規化されます。

  #xA は，データ変換処理によって，各 OS に対応した改行コードとなります。
- 属性値に含まれる改行コード（LF（#xA），CR（#xD）または CR + LF（#xD #xA）），およびタブ（#x9）は，DOM へのパース処理によって，半角空白（#x20）に正規化されます。文字参照で指定した場合は，正規化の対象にはなりません。
- XML で使用できる文字は，次に示す Unicode の範囲となります。

  `0x09 | 0x0A | 0x0D | [0x20-0xD7FF] | [0xE000-0xFFFD] | [0x10000-0x10FFFF]`

- サービスプラットフォームで扱える XML 形式データの制限を次に示します。
  - 扱える文字コードは UTF-8 だけです。UTF-8 以外の文字コードを指定した場合，動作は保障されません。
  - ルートノードが存在しないデータは扱えません。
  - csc で始まる名前空間接頭辞は利用できません。
  - BOM（0xEF 0xBB 0xBF）を付加した場合，動作は保障されません。
- データ変換処理の結果は，BMP の範囲（UCS-2）の文字はそのまま文字で出力され，Unicode 補助文字については，10 進数の数値文字参照で出力されます。

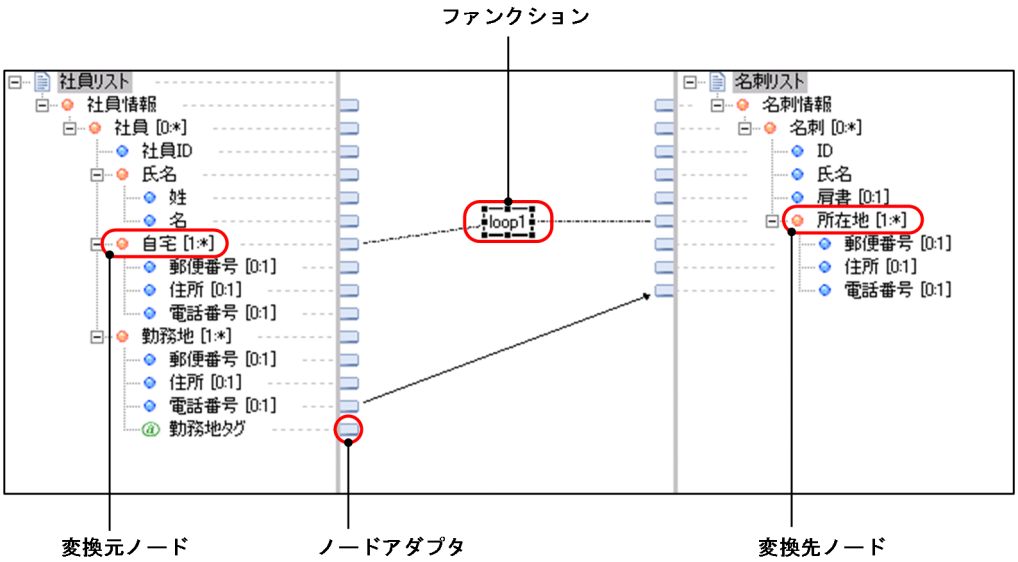
  ただし，特殊文字「<」，「>」，「&」は，それぞれ文字実体参照「<」「>」「&」として処理されます。属性の場合は，さらに「"」を文字実体参照「"」として処理されます。
- 妥当性検証は，明示的に検証機能を利用しない場合，実施されません。

  このため，データ変換処理の結果は，出力先電文フォーマットに妥当でないデータとなることがあります。
- データ変換のマッピング元にノードが存在しない場合，マッピング先もノードなしとなります（エラーにはなりません）。
- サービスアダプタの配備，アンデプロイ，およびキャッシュの登録を繰り返すと，次に示す仕様によって，サービスアダプタの不要なキャッシュが増え続けます。不要なキャッシュを削除するには，cscsvstop コマンドを実行して HCSC サーバを停止後，再配備してください。
  - サービスアダプタで電文のデータ変換を行う場合，フォーマット定義とデータ変換定義のオブジェクトが生成されます。しかし，サービスアダプタを再配備した場合，オブジェクト名の一部に配備の時刻が設定されているため，前回生成されたオブジェクトは参照されません。再配備のたびに新しいオブジェクトが生成されます。
  - csccompostop コマンドでは，サービスアダプタのキャッシュは削除されません。

7.4　マッピングする

マッピングとは，変換元（マッピング元）の値を変換先（マッピング先）の値に対応づけることです。マッピングの画面を次に示します。

図 7-1　マッピングの画面



マッピングは，大きく分けて次の 4 種類になります。

(1) 変換元ノードの値を変換先ノードに直接マッピングする

操作は，パレットのツールを使用する場合とダイアログを使用する場合があります。マッピング方法の詳細については，「7.4.1　変換元ノードの値を変換先ノードに直接マッピングする」を参照してください。

(2) 変換元ノードの値を加工して変換先ノードにマッピングする

変換元ノードの値の加工には，ファンクションを使用します。

操作は，パレットのツールを使用する場合とダイアログを使用する場合があります。マッピング方法の詳細については，「7.4.2　変換元ノードの値を加工して変換先ノードにマッピングする」を参照してください。

(3) 変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする

操作は，ダイアログを使用します。マッピング方法の詳細については，「7.4.3　変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする」を参照してください。

(4) 変換先ノードの要素から候補を選択して自動でマッピングする

操作は，ダイアログを使用します。マッピング方法の詳細については，「7.4.4　変換先ノードの要素から候補を選択して自動でマッピングする」を参照してください。

マッピングを取り消す方法については，「7.4.5　マッピングを取り消す」を参照してください。

また，マッピングするときの注意事項については，「7.4.10　マッピングの注意事項」を参照してください。

7.4.1　変換元ノードの値を変換先ノードに直接マッピングする

変換元ノードの値を変換先ノードに直接マッピングするには，パレットのツールを使用する方法とダイアログを使用する方法があります。それぞれの場合の手順を次に説明します。

(1)　パレットのツールを使用する場合

パレットのツールを使用したマッピングの手順を次に示します。

1. パレットから［マッピング］を選択します。

2. マッピング元となる変換元ノードのノードアダプタをクリックします。

   マッピング元には，マッピング対象※の変換元ノードを指定してください。マッピング元にできないノードを指定しようとした場合，マウスポインタに ⊘ が付き，指定できない状態になります。

3. マッピング先となる変換先ノードのノードアダプタをクリックします。

   マッピング線 ⟶（代入線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

   また，マッピング先には，マッピング対象※かつ「7.11.2　マッピングできるノードおよびファンクションの対応」で指定された条件を満たす変換先ノードを指定してください。マッピング先にできないノードを指定しようとした場合，マウスポインタに ⊘ が付き，指定できない状態になります。

注※

マッピング対象については，「7.11.1　マッピング対象とマッピング対象外」を参照してください。

(2)　ダイアログを使用する場合

［マッピング元設定］ダイアログを使用してマッピングします。［マッピング元設定］ダイアログを使用したマッピングの手順を次に示します。

1. 変換先スキーマツリービューアのマッピング先となる変換先ノードを右クリックして，［マッピング元］を選択します。

   ［マッピング元設定］ダイアログが表示されます。

2. ［ノード追加］ボタンをクリックします。

   ［ノード選択］ダイアログが表示されます。

3. マッピング元となる変換元ノードを指定して，［OK］ボタンをクリックします。

   ［パス/ファンクション名］に指定したノードのパス名が表示されます。パス名の表示形式の詳細については，「7.4.6　マッピング元の表示形式」を参照してください。

   マッピング元には，マッピング対象※の変換元ノードを指定してください。

4. ［OK］ボタンをクリックします。

マッピング線 ⟶ （代入線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

注※

マッピング対象については，「7.11.1　マッピング対象とマッピング対象外」を参照してください。

7.4.2　変換元ノードの値を加工して変換先ノードにマッピングする

変換元ノードの値を加工して変換先ノードにマッピングする場合，ファンクションを使用します。ファンクションの詳細については，「7.5　ファンクションを使用した値またはノードの加工」を参照してください。

(1)　パレットのツールを使用する場合

パレットのツールを使用したマッピングの手順を次に示します。

1. パレットから使用するファンクションを選択します。

2. マッピングビューアで適当な場所をクリックします。

   ファンクションがマッピングビューアに配置されます。配置されたファンクションは，ドラッグ＆ドロップで移動できます。

3. パレットから［マッピング］を選択します。

4. マッピング元となる変換元ノードのノードアダプタまたはファンクションをクリックします。

   マッピング元には，次の変換元ノードまたはファンクションを指定してください。

   - 変換元ノード

     マッピング対象※1 の変換元ノード

   - ファンクション

     出力側にマッピング線が設定されていないファンクション

   マッピング元にできないノードまたはファンクションを指定しようとした場合，マウスポインタに ⊘ が付き，指定できない状態になります。

5. マッピング先となる変換先ノードのノードアダプタまたはファンクションをクリックします。

   マッピング線 ⟶ （代入線）が設定されます。くり返しファンクションを使用した場合は，マッピング線 ┈┈⟶ （くり返し対応線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

   また，マッピング先には，次の変換先ノードまたはファンクションを指定してください。

   - 変換先ノード

マッピング対象※1 かつ「7.11.2　マッピングできるノードおよびファンクションの対応」で指定された条件を満たす変換先ノード

- ファンクション
  「7.11.2　マッピングできるノードおよびファンクションの対応」および「7.11.3　接続できるマッピング線の数」で指定された条件を満たし，かつファンクション同士の接続で閉路※2 とならないファンクション

マッピング先にできないノードまたはファンクションを指定しようとした場合，マウスポインタに ⊘ が付き，指定できない状態になります。

注※1

マッピング対象については，「7.11.1　マッピング対象とマッピング対象外」を参照してください。

注※2

ファンクション A からファンクション B に，ファンクション B からファンクション C に，ファンクション C からファンクション A にのように，ファンクション同士の接続が閉路を作成するマッピングです。

注意事項

ファンクションを配置する前にマッピング線を設定してしまった場合，マッピング線を削除して，もう 1 度やり直してください。

参考

ファンクションをマッピングビューアに配置すると，ファンクション名に「ファンクションタイプ（例：concat）＋連番」が設定されます。この連番は，ファンクション名が重複しないように，マッピングした順に 1 から付けられる自然数の通番です。

また，配置したファンクションの名称は，ファンクションを直接編集するか，またはファンクションをダブルクリックすると表示されるダイアログで変更できます。ファンクション名の変更方法の詳細は，「7.12　ファンクション名の直接編集」を参照してください。

なお，ファンクションを削除した場合，ファンクションに設定された名称はほかのファンクションで使用することができます。

(2)　ダイアログを使用する場合

ダイアログを使用したマッピングは，マッピング元とマッピング先の種類によって使用するダイアログが次のように異なります。

- マッピング元が変換元ノードでマッピング先がファンクションのとき
  マッピング先のファンクションを設定するダイアログを使用します。

マッピング先が選択ファンクションの場合は，「7.5.17　条件によって異なる値を出力する」を参照してください。

- マッピング元およびマッピング先がファンクションのとき

  マッピング先のファンクションを設定するダイアログを使用します。

  マッピング先が選択ファンクションの場合は，「7.5.17　条件によって異なる値を出力する」を参照してください。

- マッピング元がファンクションでマッピング先が変換先ノードのとき

  ［マッピング元設定］ダイアログを使用します。

  マッピング元がくり返しファンクションまたは選択ファンクション※の場合は，［くり返し設定］ダイアログも使用できます。［くり返し設定］ダイアログによるマッピングの詳細については，「7.6.1　［くり返し設定］ダイアログを使用してマッピングする」を参照してください。

  注※

  くり返しファンクションに接続されている，または何も接続されていない選択ファンクションになります。くり返しファンクション以外のファンクションに接続されている場合，［くり返し設定］ダイアログは使用できません。

(a)　マッピング元が変換元ノードでマッピング先がファンクションのとき

マッピング元が変換元ノードでマッピング先がファンクションのときのマッピングの手順を次に示します。

1. パレットから使用するファンクションを選択します。

2. マッピングビューアで適当な場所をクリックします。

   ファンクションがマッピングビューアに配置されます。配置されたファンクションは，ドラッグ＆ドロップで移動できます。

3. 次のどちらかの方法でマッピング先のファンクションを設定するダイアログを表示させます。

   - マッピング先のファンクションを右クリックして，［設定］を選択します。
   - マッピング先のファンクションをダブルクリックします。

   マッピング先のファンクションを設定するダイアログが表示されます。

4. ［ノード追加］ボタンまたは［ノード選択］ボタンをクリックします。

   ［ノード選択］ダイアログが表示されます。

5. マッピング元となる変換元ノードを指定して，［OK］ボタンをクリックします。

   マッピング先のファンクションを設定するダイアログの［入力］（くり返しファンクションの場合は［基準パス］）に，マッピング元の変換元ノードが設定されます。変換元ノードはパス名で表示されます。パス名の表示形式の詳細については，「7.4.6　マッピング元の表示形式」を参照してください。

   マッピング元には，マッピング対象※かつ「7.11.2　マッピングできるノードおよびファンクションの対応」で指定された条件を満たす変換元ノードを指定してください。

6. [OK] ボタンをクリックします。

マッピング線 ⟶ （代入線）が設定されます。くり返しファンクションを使用した場合は，マッピング線 ┈┈┈ （くり返し対応線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

注※

マッピング対象については，「7.11.1　マッピング対象とマッピング対象外」を参照してください。

(b)　マッピング元およびマッピング先がファンクションのとき

マッピング元およびマッピング先がファンクションのときのマッピングの手順を次に示します。

1. パレットから使用するファンクションを選択します。

2. マッピングビューアで適当な場所をクリックします。

ファンクションがマッピングビューアに配置されます。配置されたファンクションは，ドラッグ＆ドロップで移動できます。

3. 次のどちらかの方法でマッピング先のファンクションを設定するダイアログを表示させます。

- マッピング先のファンクションを右クリックして，［設定］を選択します。
- マッピング先のファンクションをダブルクリックします。

マッピング先のファンクションを設定するダイアログが表示されます。

4. ［ファンクション追加］ボタンまたは［ファンクション選択］ボタンをクリックします。

［ファンクション選択］ダイアログが表示されます。

5. マッピング元となるファンクションを指定して，［OK］ボタンをクリックします。

マッピング先のファンクションを設定するダイアログの［入力］に，マッピング元のファンクションが設定されます。定数ファンクションの場合はアポストロフィ（'）で囲んだ定数値，定数ファンクション以外のファンクションの場合はファンクション名が表示されます。

マッピング元には，「7.11.2　マッピングできるノードおよびファンクションの対応」で指定された条件を満たし，かつ出力側にマッピング線が設定されていないファンクションを指定してください。

6. [OK] ボタンをクリックします。

マッピング線 ⟶ （代入線）が設定されます。くり返しファンクションを使用した場合は，マッピング線 ┈┈┈ （くり返し対応線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

(c)　マッピング元がファンクションでマッピング先が変換先ノードのとき

1. パレットから使用するファンクションを選択します。

2. マッピングビューアで適当な場所をクリックします。

ファンクションがマッピングビューアに配置されます。配置されたファンクションは，ドラッグ＆ドロップで移動できます。

3. 変換先スキーマツリービューアのマッピング先となる変換先ノードを右クリックして，［マッピング元］を選択します。

［マッピング元設定］ダイアログが表示されます。

4. ［ファンクション追加］ボタンをクリックします。

［ファンクション選択］ダイアログが表示されます。

5. マッピング元となるファンクションを指定して，［OK］ボタンをクリックします。

［パス/ファンクション名］に指定したファンクション名が設定されます。

マッピング元には，「7.11.2　マッピングできるノードおよびファンクションの対応」で指定された条件を満たし，かつ出力側にマッピング線が設定されていないファンクションを指定してください。ただし，選択ファンクションで条件として指定したため，出力側にマッピング線 ┈┈→ （条件線）が設定されているファンクションは指定できます。

6. ［OK］ボタンをクリックします。

マッピング線 ⟶ （代入線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

注意事項

ファンクションを配置する前にマッピング線を設定してしまった場合，マッピング線を削除して，もう 1 度やり直してください。

参考

ファンクションをマッピングビューアに配置すると，ファンクション名に「ファンクションタイプ（例：concat）＋連番」が設定されます。この連番は，ファンクション名が重複しないように，マッピングした順に 1 から付けられる自然数の通番です。

また，配置したファンクションの名称は，ファンクションを直接編集するか，またはファンクションをダブルクリックすると表示されるダイアログで変更できます。ファンクション名の変更方法の詳細は，「7.12　ファンクション名の直接編集」を参照してください。

なお，ファンクションを削除した場合，ファンクションに設定された名称はほかのファンクションで使用することができます。

7.4.3　変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする

変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする場合，［自動マッピング元設定］ダイアログを使用します。［自動マッピング元設定］ダイアログでは，変換元ノードの範囲と変換先ノードの範囲を指定すると，変換元ノードと変換先ノードで類似度が高い要素同士を自動でマッピングします。範囲内にマッピング候補が存在しない場合やすでに変換先ノードアダプタにマッピング線が接続されている場合は，マッピングされません。マッピング時の類似度の判定については，「7.4.9　自動マッピング時の類似度の判定」を参照してください。

［自動マッピング元設定］ダイアログを使用したマッピングの手順を次に示します。

1. **変換先スキーマツリービューアのマッピング先となる変換先ノードの要素を右クリックして，［自動マッピング］を選択します。**

   変換先ノードの要素には，マッピングさせる範囲の最上位の要素を選択してください。

   ［自動マッピング元設定］ダイアログが表示されます。

2. **［ノード追加］ボタンをクリックします。**

   ［ノード選択］ダイアログが表示されます。

3. **マッピング元となる変換元ノードの要素を指定して，［OK］ボタンをクリックします。**

   変換元ノードの要素には，マッピングさせる範囲の最上位の要素を選択してください。

   選択した要素のパス名が［パス］に表示されます。パス名の表示形式の詳細については，「7.4.6　マッピング元の表示形式」を参照してください。

   要素を複数選択したい場合は，手順 3.を繰り返してください。

   なお，次の場合は，エラーメッセージが出力されます。

   - すでに選択している要素を選択した場合
   - 選択した要素の祖先または子孫がすでに選択されている場合

   エラーメッセージが出力された場合，［OK］ボタンをクリックしてメッセージダイアログを閉じたあと，要素を再選択してください。

4. **［OK］ボタンをクリックします。**

   指定した範囲内にマッピング候補が存在する場合,マッピング線 ⟶ （代入線）が自動で設定されます。範囲内にマッピング候補が存在しない場合やすでに変換先ノードアダプタにマッピング線が接続されている場合は，マッピングされません。

注意事項

マッピング候補が複数存在する場合は，最も類似度が高い候補が選択されるため，1 つの変換元ノードに複数のマッピング線が生成される場合があります。マッピング済みの変換元ノードを対象外にしたい場合など，マッピング候補を絞り込みたいときは，［自動マッピング候補選択］ダイア

ログを使用するか，自動でマッピングさせたい範囲を狭めるようにしてください。[自動マッピング候補選択] ダイアログを使用して自動でマッピングする方法については，「7.4.4　変換先ノードの要素から候補を選択して自動でマッピングする」を参照してください。

参考

設定されたマッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

7.4.4　変換先ノードの要素から候補を選択して自動でマッピングする

変換先ノードの要素から候補を選択して自動でマッピングする場合，[自動マッピング候補選択] ダイアログを使用します。[自動マッピング候補選択] ダイアログでは，変換元ノードの要素の候補を指定すると，選択した候補の要素がマッピングされ，マッピング線（代入線）が自動で設定されます。また，選択した要素のノードアダプタにマッピング線が接続済みの場合でも，同一の要素で依存関係が異なるマッピング線を接続できるときは，指定した候補とのマッピングを実行します。マッピング時の類似度の判定については，「7.4.9　自動マッピング時の類似度の判定」を参照してください。

[自動マッピング候補選択] ダイアログを使用したマッピングの手順を次に示します。

1. **変換先スキーマツリービューアのマッピング先となる変換先ノードからマッピングする要素を右クリックして，[自動マッピング候補選択] を選択します。**

   [自動マッピング候補選択] を選択できない場合は，その要素はマッピングの対象にはならないため，ほかの候補となる要素を選択してください。

   [自動マッピング候補選択] ダイアログが表示されます。

2. **[自動マッピング候補一覧] からマッピング候補を選択します。**

   [自動マッピング候補一覧] には優先度が高い候補ほど上位に表示されます。

   マッピングの候補が存在しない場合，[自動マッピング候補一覧] には何も表示されません。

3. **[OK] ボタンをクリックします。**

   選択した候補の要素がマッピングされ，マッピング線 ⟶ （代入線）が自動で設定されます。

   選択した要素のノードアダプタにマッピング線が接続済みの場合でも，同一の要素で依存関係が異なるマッピング線を接続できるときは，指定した候補とのマッピングが実行されます。

参考

設定されたマッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

また，スキーマツリービューアでは要素のパスをコピーできます。要素のパスをコピーする方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.6　スキーマツリービューアから要素のパスをコピーする」を参照してください。

7.4.5　マッピングを取り消す

マッピング線やファンクションを削除して，マッピングを取り消すことができます。

(1)　マッピング線を削除する

次のどちらかの方法で，マッピング線を削除します。

方法 1

削除したいマッピング線を右クリックして，［削除］を選択します。

方法 2

削除したいマッピング線を選択して，［Delete］キーを押します。

注意事項

条件線はユーザが削除できません。条件線は，条件の設定を取り消した場合に自動的に削除されます。

(2)　ファンクションを削除する

次のどちらかの方法で，ファンクションを削除します。

方法 1

削除したいファンクションを右クリックして，［削除］を選択します。

方法 2

削除したいファンクションを選択して，［Delete］キーを押します。

注意事項

条件としてファンクションを指定（[条件設定] ダイアログの［条件］でファンクションを指定）した場合，そのファンクションは削除できません。

7.4.6　マッピング元の表示形式

[マッピング元設定] ダイアログや各ファンクションを設定するダイアログで，マッピング元となる変換元ノードまたはファンクションを指定した場合，次のように表示されます。

(1)　変換元ノードの場合

スキーマツリーのノードパスが表示されます。ノードパスは次のように表示されます。

(a)　変換元ノードに条件が設定されているとき

条件が設定されたノード名の後ろに，角括弧（[　]）で囲まれた条件が表示されます。

（例）　/aa/bb[position()=’1’]/cc

条件としてファンクションを指定（[条件設定] ダイアログの［条件］でファンクションを指定）した場合，ファンクション名が波括弧（{　}）で囲まれて表示されます。

（例）　/aa/bb[{length1}='5']

条件式右辺の値にアンパサンド（&）およびアポストロフィ（'）が含まれる場合，アンパサンドおよびアポストロフィは実体参照で表示されます。

- アポストロフィ（'）：'
- アンパサンド（&）：&

（例）

条件式左辺が「/root/input」，条件式右辺が「a&'b'」の場合

root/input ='a&'b''

(b)　スキーマ論理名が複数あるとき

ノードパスの先頭に，「$<スキーマ論理名>」が表示されます。

（例）　$source1/aa/bb/cc

(c)　属性のとき

ノード名の前に「@」が表示されます。

（例） /aa/bb/@cc

(2) ファンクションの場合

ファンクション名が波括弧（{ }）で囲まれて表示されます。

（例） {ファンクション名}

7.4.7 マッピング線とファンクションを見やすくする

選択したノード，またはファンクションに関連するファンクションやマッピング線を強調して表示できます。

選択したノード，またはファンクションに関連するマッピング線やファンクションを強調するための手順を次に示します。

1. **データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックします。**
   ポップアップメニューが表示されます。

2. **ポップアップメニューから［強調表示］を選択します。**
   強調表示のメニューが表示されます。

3. **用途に合わせて，次のどれかを選択します。**
   選択したメニューに合わせて，関連するマッピング線とファンクションだけが強調されます。
   - 変換元ノードの関連を強調表示する
   - 変換先ノードの関連を強調表示する
   - ファンクションの関連を強調表示する

   強調したくない場合は「強調表示しない」を選択してください。

強調するマッピング線とファンクションは，選択しているノードとの関連の強さによって，色の濃淡が異なります。強調しているマッピング線とファンクションは濃い色で表示され，強調していないマッピング線とファンクションは薄い色で表示されます。

なお，強調表示をしている状態で，マッピング線やファンクションを追加した場合は，ファンクション，またはノードを選択し直したときに表示が更新されます。

参考

選択したいノードまたはファンクションが見つからない場合

変換元ノード，ファンクション，または変換先ノードを検索できます。

検索方法を次に示します。

1. ツリービューアまたはマッピングビューアの適当な場所で右クリックをして，［名称検索］を選択します。
   ［名称検索］ダイアログが表示されます。
2. ［検索対象］で次のどれかのチェックボックスをチェックします。複数チェックしても構いません。
   ・変換元ノード
   ・ファンクション
   ・変換先ノード
3. ［検索］に検索したい文字列を入力し［検索］ボタンをクリックします。
   ［検索結果一覧］に入力した文字列を含むノード，またはファンクションの一覧が表示されます。
4. ［検索結果一覧］から，検索していたノード名，またはファンクション名を選択します。
   選択したノード，またはファンクションがツリービューア上でも選択された状態になります。

注意

［名称検索］ダイアログを開いた状態で，ファンクション名の変更やファンクションを削除しないでください。これらの作業を実行した場合の動作は保障しません。

7.4.8　マッピング対象の範囲を限定する

マッピング定義エディタでデータ変換定義ファイルを作成する際，変換先スキーマに XML スキーマ（拡張子：xsd）を指定している場合は，マッピング対象のノードを限定することができます。この機能は，ファイル操作アダプタの変換オペレーションの出力ファイルが XML 形式の場合にだけ使用します。

操作手順を次に示します。

1. **マッピング定義エディタを起動します。**
   マッピング定義エディタの起動方法については，「7.3.1(1)(b) Eclipse のメニューから表示する場合」を参照してください。

2. **変換先スキーマツリービューアでレコードとして扱う変換先ノードの要素（単純内容要素または複合内容要素）を右クリックします。**
   ポップアップメニューが表示されます。

3. **ポップアップメニューから［子孫ノードを含めてマッピング対象にする］を選択します。**
   設定した要素とその子孫要素以外の変換先ノードが非活性になります。
   設定した要素とその子孫要素以外の変換先ノードにマッピング線が接続されている場合は確認ダイアログが表示されます。［はい］ボタンをクリックすると，変換先ノードに接続されているマッピング線が削除されたあと，設定した要素とその子孫要素以外の変換先ノードが非活性になります。
   非活性となった変換先ノードは，マッピング線が接続されていなくても検証時にエラーとなりません。

注意事項

- レコードとして扱う要素のパスはファイル操作アダプタ定義ファイルで指定してください。ファイル操作アダプタ定義ファイルの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「3.5.2　ファイル操作アダプタ定義ファイル」を参照してください。
- ルート要素を再選択した場合，［子孫ノードを含めてマッピング対象にする］の設定はリセットされます。
- マッピング定義をコピーした場合，［子孫ノードを含めてマッピング対象にする］を設定した状態はコピー先のマッピング定義に引き継がれます。

7.4.9　自動マッピング時の類似度の判定

自動マッピングでは，変換先ノードのスキーマ要素に対して，変換元ノードのスキーマ要素の類似度を判定し，マッピングの候補とします。類似した要素の候補が複数あるときは，優先度の高い順にマッピングされます。

変換元ノードのスキーマ要素の類似度を判定するときの優先度を次の表に示します。なお，優先度はレベル 1 が最も高く，レベルの値が大きいほど低くなります。

表 7-2　自動マッピングで判定される類似度の優先度

優先度	パス全体		末端の要素	
	名前空間	要素名	名前空間	要素名
1	○	○	−	−
2	×	○	−	−
3	−	△※	○	○
4	−	△※	×	○
5	−	△※	×	△

（凡例）

○：一致します。

△：一部一致します。

×：一致しません。

−：該当しません。

注※

判定によって一致しないこともあります。

7.4.10　マッピングの注意事項

マッピングしたあとに，データ変換定義の内容が妥当かどうか検証をします。検証でエラーが発生しないように，次の点を考慮してマッピングしてください。

- 「7.11　マッピングの条件」に従ってマッピングしてください。
- マッピング元の変換元ノードとマッピング先の変換先ノードの出現回数は一致させてください。変換元ノードが存在しない場合，変換先ノードが必須ノードであっても，ノードは生成されません。
- 最小出現回数が 2 回以上に設定されている複合内容の要素は，必ずマッピングしてください。ただし，祖先に最小出現回数が 0 回のノードが出現する場合は除きます。
- 最小出現回数が 1 回以上に設定されている単純内容の要素（any 要素を含む）は，必ずマッピングしてください。
- 複合内容要素の子要素に any 要素が複数存在しないようにしてください。
- choice 要素の子ノードまたは子孫ノードに対してマッピングする場合，複数の子要素が同時に出現するか，またはどの子要素も出現しないようなマッピングを定義しないでください。
- choice 要素の複数の子要素または子孫ノードに対してマッピングする場合は，複数の子要素が同時に出現しないようにしてください。
- choice 要素のすべての子要素の最小出現回数が 1 回以上に設定されていて，1 つの子要素の子孫要素がマッピングされている場合，その子要素に対して選択ファンクションを使用するときは，複数の子要素が同時に出現しないようにしてください。
- マッピング先の変換先ノードには，targetNamespace に xsl と xsi の URI が指定されている名前空間を定義しないでください。
- インスタンス属性（xsi:type，xsi:schemaLocation，xsi:noNamespaceSchemaLocation）は，マッピング元の変換元ノードで定義されていても，マッピング先の変換先ノードには出現しません。
- インスタンス属性 xsi:nil が定義された変換元ノードを nillable 属性が true と定義された変換先ノードにマッピングした場合，変換先ノードには，変換元ノードの xsi:nil で定義された内容が反映されます。同様に，インスタンス属性 xsi:nil が定義された変換元ノードを選択ファンクション（choose）を経由するようにマッピングした場合も，変換先ノードには，変換元ノードの xsi:nil で定義された内容が反映されます。
- nillable 属性が定義されている変換元ノードから，nillable 属性が定義されていない変換先ノードへマッピングしないでください。
- マッピング定義編集中に次のような電文フォーマットの変更をした場合，表示される電文フォーマットは変更前のものとなります。変更後の電文フォーマットを反映させるには，対象のサービスを閉じてから再度マッピング定義を起動してください。
  - グローバル要素の追加または削除
  - ルート要素以外のグローバル要素への変更

- import 先，または include 先のスキーマ間で 1 つの名前空間 URI に対して複数の名前空間接頭辞が定義されている場合，マッピング定義の検証（パッケージングを含む）でエラーとなります。1 つの名前空間 URI に対して使用する名前空間接頭辞は 1 種類にしてください。
- 電文フォーマットのスキーマファイルで，異なる名前空間接頭辞で同じ名前空間を指定している場合，マッピング定義起動時に毎回，ルート要素選択ダイアログが表示されることがあります。1 つの名前空間に対して，2 つ以上の名前空間接頭辞は定義しないでください。
- データ変換定義で，次のような#any 要素はマッピングしないでください。
  - 兄弟要素に#anonymous，#(sequence)，または#(choice)がある#any 要素
  - #(sequence)または#(choice)下に存在し，#(sequence)または#(choice)の兄弟要素にほかの#(sequence)または#(choice)が存在する#any 要素
- ノード一覧型カスタムファンクションの出力先を複合内容要素に指定した場合，同じ依存関係の子孫ノードにマッピングできません。
- 選択ファンクションの出力値にノード一覧型カスタムファンクションが指定されている場合の注意事項を次に示します。
  - 選択ファンクションのマッピング先が複合内容要素で，選択ファンクションの別の出力値が「値を出力する」の場合，ノード一覧型カスタムファンクション以外のものを設定できません。
  - 選択ファンクションのマッピング先が属性の場合，選択ファンクションの出力値にノード一覧型カスタムファンクションを設定できません。
- データ変換上で扱う数値は double 型であり，double 型では 2 進数表現で数値を扱っているため，計算で 2 のべき乗では表せない数値を扱った場合に丸め誤差が発生します。また，double 型の有効桁数は約 16 桁となるので，約 16 桁以上の数値を扱う場合も丸め誤差が発生します。そのため，数値演算ファンクションまたは総和ファンクションの演算結果に誤差が生じる可能性があります。必要に応じて，数値フォーマット変換ファンクションまたは端数処理ファンクションで調整してください。正確な演算が必要な場合はカスタムファンクションを利用して演算処理を実装してください。
- マッピング先の変換先ノードに出現回数が設定されていて，かつノード条件やくり返しファンクションを使用しない場合は，必ず先頭（position()='1'）のデータがマッピングされます。

7.5　ファンクションを使用した値またはノードの加工

変換元ノードの値を加工して変換先ノードにマッピングする場合，ファンクションを使用します。ノード一覧型カスタムファンクションを使用すると，値だけでなくノードも加工できます。ここでは，どのような場合にどのファンクションを使用するかを説明します。ファンクションを使用したマッピング方法については，「7.4.2　変換元ノードの値を加工して変換先ノードにマッピングする」を参照してください。

ファンクションの一覧を次の表に示します。なお，文字列はすべて UTF-16 に符号化されます。そのため，Unicode 補助文字（0x10000～0x10FFFF）は，サロゲートペアに符号化されます。

表 7-3　ファンクションの一覧

分類	ファンクション名	説明
文字列系	文字列連結	複数の文字列を連結します。
	部分文字列取得	1 つの文字列から一部分の文字列を取り出します。
	文字数取得	文字列の文字数をマッピング先に代入します。
	文字列調査	文字列に指定した文字列が含まれているか，または指定した文字列で開始されているかを調査して，マッピング先に論理値を代入します。
	空白除去	文字列の先頭と末尾の空白を除去します。また，文字列間の連続した空白を 1 つの半角スペースに置換します。
数値系	数値フォーマット変換	数値をフォーマット変換します。
	数値演算	数値を演算（+，-，*，div，mod）します。
	端数処理	小数点以下の端数を処理（四捨五入，切り捨て，および切り上げ）します。
	総和	複数のノードセットのノードの数値を合計します。
ビット演算系	否定演算	否定演算を実行します。
	論理演算	論理演算（AND，OR，XOR）を実行します。
	シフト演算	シフト演算を実行します。
ノード系	ノード数取得	ノード数をマッピング先に代入します。
	ノード名取得	ノード名をマッピング先に代入します。
	ノード調査	指定した変換元ノードが存在するかを調査し，マッピング先にその論理値を代入します。
制御系	くり返し	くり返しを対応づけます。
	選択	条件によって異なる値を出力します。
その他	定数	指定した値をマッピング先に代入します。
	値置換え	変換テーブルの指定に基づいて，マッピング元の値を変換してマッピング先に代入します。
	基数変換	基数変換を実行します。

分類	ファンクション名	説明
その他	カスタム	ユーザが作成した任意の Java プログラムを呼び出します。
	XPath	XPath 式を指定します。

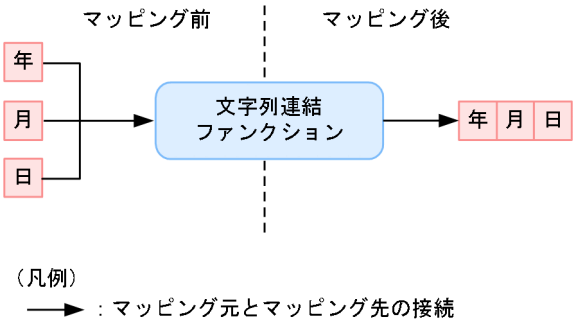
7.5.1　複数の文字列を連結する

複数のマッピング元の文字列を連結して，1 つの文字列にします。

(1)　使用するファンクション

複数のマッピング元の文字列を連結するには，文字列連結ファンクションを使用します。文字列連結ファンクションを使用した例を次に示します。

図 7-2　文字列連結ファンクションの使用例



(2)　設定手順

複数のマッピング元の文字列を連結して，1 つの文字列にする手順を次に示します。

1. パレットから文字列連結ファンクション（concat）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［文字列連結］ダイアログを表示させます。
   - 文字列連結ファンクションを右クリックして，［設定］を選択します。
   - 文字列連結ファンクションをダブルクリックします。

   ［文字列連結］ダイアログが表示されます。
   ［文字列連結］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.19　文字列連結ダイアログ」を参照してください。
4. ［入力］に連結させたい文字列のマッピング元を追加する場合，次のように設定します。
   - マッピング元が変換元ノードの場合

［ノード追加］ボタンをクリックすると，［ノード選択］ダイアログが表示されます。［ノード選択］ダイアログで，マッピング元となる変換元ノードを選択します。

- マッピング元がファンクションの場合

  ［ファンクション追加］ボタンをクリックすると，［ファンクション選択］ダイアログが表示されます。［ファンクション選択］ダイアログで，マッピング元となるファンクションを選択します。

［入力］に複数のマッピング元を指定した場合，［入力］に表示された連結対象は上から順に連結されます。いちばん上に表示された連結対象は，連結した文字列の左側に設定されます。

5. **［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。**

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

6. **［OK］ボタンをクリックします。**

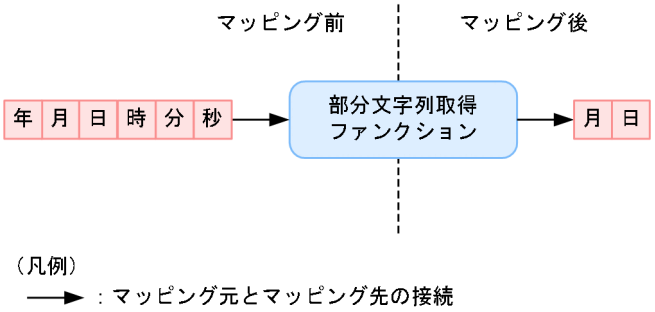
7.5.2　1 つの文字列から一部分の文字列を取り出す

1 つのマッピング元の文字列から，一部分の文字列を取り出します。

(1)　使用するファンクション

1 つのマッピング元の文字列から，一部分の文字列を取り出すには，部分文字列取得ファンクションを使用します。部分文字列取得ファンクションを使用した例を次に示します。

図 7-3　部分文字列取得ファンクションの使用例



(2)　設定手順

1 つのマッピング元の文字列から，一部分の文字列を取り出す手順を次に示します。

1. **パレットから部分文字列取得ファンクション（substr）を選択して，マッピングビューアに配置します。**

2. **マッピング線を設定します。**

3. 次のどちらかの方法で［部分文字列取得］ダイアログを表示させます。

- 部分文字列取得ファンクションを右クリックして［設定］を選択します。
- 部分文字列取得ファンクションをダブルクリックします。

［部分文字列取得］ダイアログが表示されます。

［部分文字列取得］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.20　部分文字列取得ダイアログ」を参照してください。

4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

［ノード条件設定］ダイアログが表示されます。

ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

5. 取り出す文字列を指定します。

- 取り出す範囲を文字列の先頭から指定する場合

  ［先頭からの範囲指定］ラジオボタンを選択し，［開始位置］に取り出す文字列の開始文字の位置を指定します。取り出す文字列の文字数を指定する場合，［文字数］に取り出す文字数を指定します。［開始位置］で指定した開始文字の位置から末尾までの文字列を取り出す場合，［開始位置から末尾までを取得］チェックボックスをチェックします。

- 取り出す範囲を文字列の末尾から指定する場合

  ［末尾からの範囲指定］ラジオボタンを選択し，［開始位置］に取り出す文字列の開始文字の位置を指定します。取り出す文字列の文字数を指定する場合，［文字数］に取り出す文字数を指定します。［開始位置］で指定した開始文字の位置から先頭までの文字列を取り出す場合，［開始位置から先頭までを取得］チェックボックスをチェックします。

- 文字列を 2 つに分割して，分割した前の部分かあとの部分かのどちらかを指定する場合

  ［分割文字列指定］ラジオボタンを選択し，［分割文字列］にマッピング元の文字列を分割する文字（文字列）を指定します。先頭の文字から［分割文字列］で指定した文字（文字列）より前の文字列を取り出したい場合，［取得部位］に「前」を指定します。［分割文字列］で指定した文字（文字列）のあとから末尾までの文字列を取り出したい場合，［取得部位］に「後」を指定します。

6. ［OK］ボタンをクリックします。

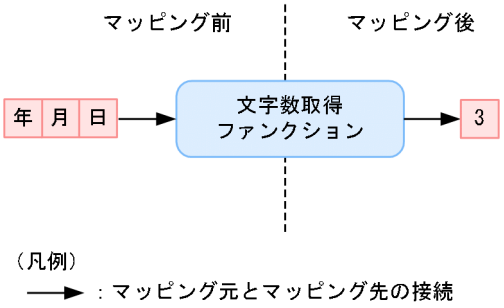
7.5.3　文字列の文字数を代入する

マッピング元の文字列の文字数をマッピング先に代入します。

(1)　使用するファンクション

マッピング元の文字列の文字数をマッピング先に代入するには，文字数取得ファンクションを使用します。文字数取得ファンクションを使用した例を次に示します。

図 7-4　文字数取得ファンクションの使用例



(2)　設定手順

マッピング元の文字列の文字数をマッピング先に代入する手順を次に示します。

1. パレットから文字数取得ファンクション（length）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［文字数取得］ダイアログを表示させます。
   - 文字数取得ファンクションを右クリックして，［設定］を選択します。
   - 文字数取得ファンクションをダブルクリックします。

   ［文字数取得］ダイアログが表示されます。

   ［文字数取得］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.21　文字数取得ダイアログ」を参照してください。
4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［OK］ボタンをクリックします。

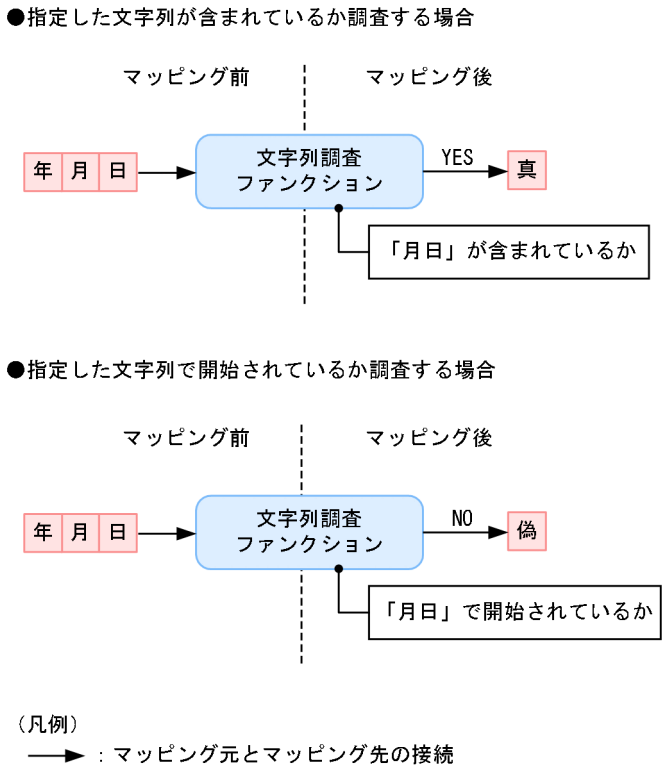
7.5.4　指定文字列の有無，または指定文字列の開始を調査する

マッピング元の文字列に指定した文字列が含まれているか，または指定した文字列で開始されているかを調査して，マッピング先にその論理値を代入します。

(1)　使用するファンクション

マッピング元の文字列に指定した文字列が含まれているか，または指定した文字列で開始されているかを調査して，マッピング先にその論理値を代入するには，文字列調査ファンクションを使用します。文字列調査ファンクションを使用した例を次に示します。

図 7-5　文字列調査ファンクションの使用例



(2)　設定手順

マッピング元の文字列を調査し，マッピング先に論理値を代入する手順を次に示します。

1. パレットから文字列調査ファンクション（contain）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［文字列調査］ダイアログを表示させます。
   - 文字列調査ファンクションを右クリックして，［設定］を選択します。
   - 文字列調査ファンクションをダブルクリックします。

   ［文字列調査］ダイアログが表示されます。

   ［文字列調査］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.22　文字列調査ダイアログ」を参照してください。
4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

［ノード条件設定］ダイアログが表示されます。

ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

5. **［調査種別］に次のどちらかを選択します。**

   - ［調査文字列］で指定した文字列が含まれているか調査する場合
     「指定文字列を含む」を選択します。
   - ［調査文字列］で指定した文字列で開始されているか調査する場合
     「指定文字列から開始」を選択します。

6. **［調査文字列］に調査対象となる文字列を指定します。**

7. **［OK］ボタンをクリックします。**

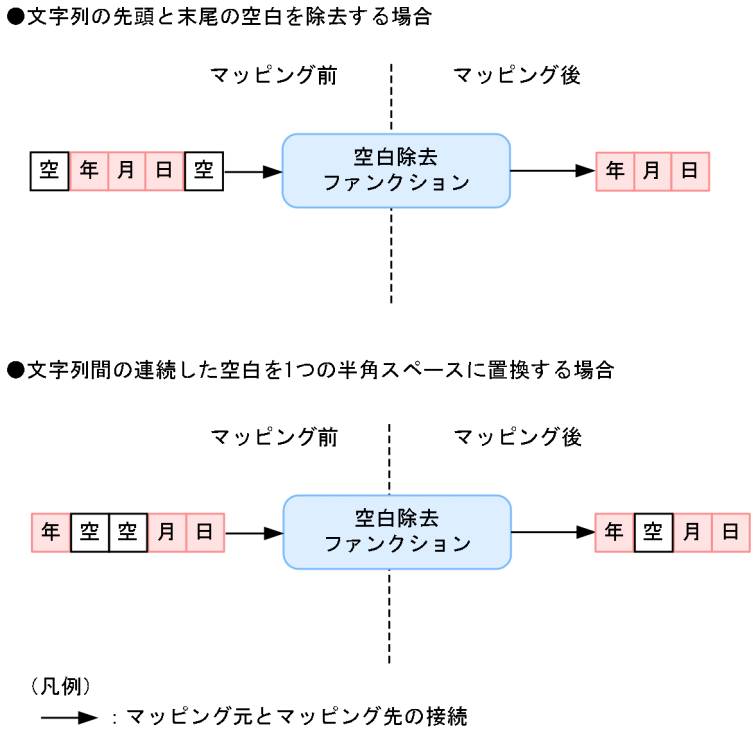
7.5.5　文字列の空白を除去する

マッピング元の文字列の先頭と末尾の空白（半角スペース，タブ，復帰（Carriage Return），および改行（Line Feed））を除去します。また，文字列間の連続した空白を 1 つの半角スペースに置換します。

(1)　使用するファンクション

文字列の空白を除去するには，空白除去ファンクションを使用します。空白除去ファンクションを使用した例を次に示します。

図 7-6　空白除去ファンクションの使用例



(2)　設定手順

文字列の空白を除去する手順を次に示します。

1. パレットから空白除去ファンクション（trim）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［空白除去］ダイアログを表示させます。
   - 空白除去ファンクションを右クリックして，［設定］を選択します。
   - 空白除去ファンクションをダブルクリックします。

   ［空白除去］ダイアログが表示されます。

   ［空白除去］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.23　空白除去ダイアログ」を参照してください。
4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［OK］ボタンをクリックします。

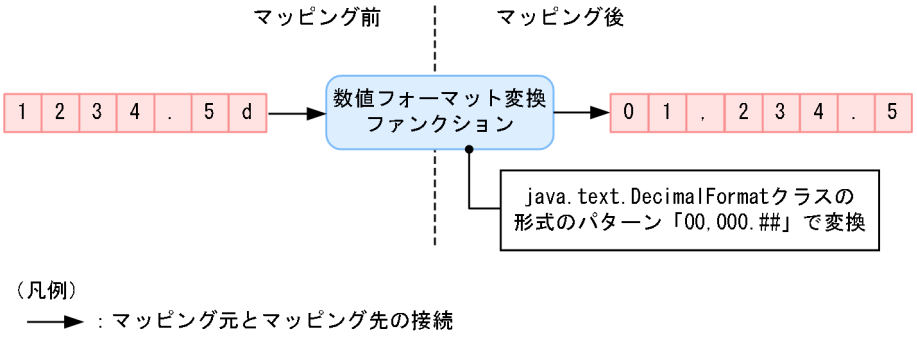
7.5.6　数値をフォーマット変換する

マッピング元の数値を java.text.DecimalFormat クラスの形式のパターンでフォーマット変換します。また，パターンに指定する小数点文字または桁区切り文字をデフォルトから変更できます。
java.text.DecimalFormat クラスの形式のパターンの詳細については，Java の API に関するドキュメントを参照してください。

(1)　使用するファンクション

マッピング元の数値を java.text.DecimalFormat クラスの形式のパターンでフォーマット変換するには，数値フォーマット変換ファンクションを使用します。数値フォーマット変換ファンクションを使用した例を次に示します。

図 7-7　数値フォーマット変換ファンクションの使用例



(2)　設定手順

マッピング元の数値を java.text.DecimalFormat クラスの形式のパターンでフォーマット変換する手順を次に示します。

1. パレットから数値フォーマット変換ファンクション（format）を選択して，マッピングビューアに配置します。

2. マッピング線を設定します。

3. 次のどちらかの方法で［数値フォーマット変換］ダイアログを表示させます。
   - 数値フォーマット変換ファンクションを右クリックして，［設定］を選択します。
   - 数値フォーマット変換ファンクションをダブルクリックします。

   ［数値フォーマット変換］ダイアログが表示されます。
   ［数値フォーマット変換］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.24　数値フォーマット変換ダイアログ」を参照してください。

4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。
   ［ノード条件設定］ダイアログが表示されます。

ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

5. ［パターン］に変換後の java.text.DecimalFormat クラスの形式のパターン文字列を指定します。

6. 小数点文字または桁区切り文字をデフォルトから変更する場合，［記号を変更する］チェックボックスをチェックし，次のように指定します。

   - 小数点文字を変更する場合

     ［小数点文字］に変更後の小数点文字の記号を指定します。省略した場合，デフォルトの記号となります。

   - 桁区切り文字を変更する場合

     ［桁区切り文字］に変更後の桁区切り文字の記号を指定します。省略した場合，デフォルトの記号となります。

7. ［OK］ボタンをクリックします。

7.5.7　数値を演算する

2 つのマッピング元の数値に対して，指定した演算をします。指定できる演算の種類を次に示します。

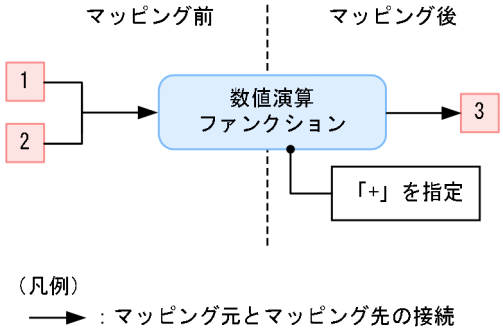
- ＋（加算）
- -（減算）
- *（乗算）
- /（除算）
- %（剰余）

入力値が数値以外の場合は，エラーの値（NaN）を返します。

(1)　使用するファンクション

2 つのマッピング元の数値に対して，指定した演算をするには，数値演算ファンクションを使用します。数値演算ファンクションを使用した例を次に示します。

図 7-8　数値演算ファンクションの使用例



(2)　設定手順

2 つのマッピング元の数値に対して，指定した演算をする手順を次に示します。

1. パレットから数値演算ファンクション（calc）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［数値演算］ダイアログを表示させます。
   - 数値演算ファンクションを右クリックして，［設定］を選択します。
   - 数値演算ファンクションをダブルクリックします。

   ［数値演算］ダイアログが表示されます。

   ［数値演算］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.25　数値演算ダイアログ」を参照してください。
4. ［演算］に実行したい演算を指定します。
5. ［入力 1］または［入力 2］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
6. ［OK］ボタンをクリックします。

7.5.8　小数点以下の端数を処理する

マッピング元の数値が小数の場合，小数点以下の端数を処理します。指定できる端数の処理を次に示します。

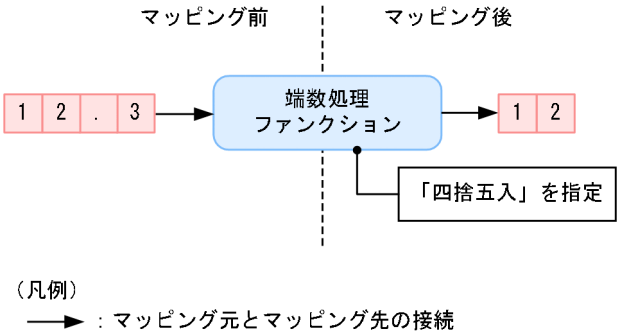
- 四捨五入
- 切り捨て

- 切り上げ

(1)　使用するファンクション

小数点以下の端数を処理するには，端数処理ファンクションを使用します。端数処理ファンクションを使用した例を次に示します。

図 7-9　端数処理ファンクションの使用例



(2)　設定手順

小数点以下の端数を処理する手順を次に示します。

1. パレットから端数処理ファンクション（round）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［端数処理］ダイアログを表示させます。
   - 端数処理ファンクションを右クリックして，［設定］を選択します。
   - 端数処理ファンクションをダブルクリックします。

   ［端数処理］ダイアログが表示されます。

   ［端数処理］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.26　端数処理ダイアログ」を参照してください。
4. ［入力］に指定されている変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［種別］に実行したい端数の処理を指定します。
6. ［OK］ボタンをクリックします。

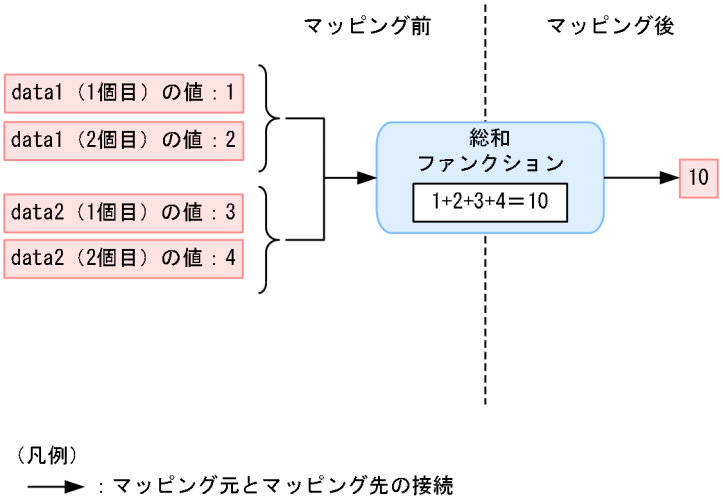
7.5.9　複数のノードセットのノードの数値を合計する

複数のマッピング元のノードセットを指定し，そのノードセットに含まれているノードの数値を合計します。

(1)　使用するファンクション

複数のノードセットのノードの数値を合計するには，総和ファンクションを使用します。総和ファンクションを使用した例を次に示します。

図 7-10　総和ファンクションの使用例



(2)　設定手順

複数のノードセットのノードの数値を合計する手順を次に示します。

1. パレットから総和ファンクション（sum）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［総和］ダイアログを表示させます。
   - 総和ファンクションを右クリックして，［設定］を選択します。
   - 総和ファンクションをダブルクリックします。

   ［総和］ダイアログが表示されます。

   ［総和］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.27　総和ダイアログ」を参照してください。
4. ［入力］に合計させたい数値のマッピング元を追加する場合，［ノード追加］ボタンをクリックします。

   ［ノード選択］ダイアログが表示されます。
5. マッピング元となるノードセットを指定します。

   ［入力］に合計させたい数値のマッピング元が設定されます。

6. [入力] に指定されている変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたい場合，[ノード条件設定] ボタンをクリックします。

   [ノード条件設定] ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，[ノード条件設定] ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

7. [OK] ボタンをクリックします。

7.5.10　否定演算を使用する

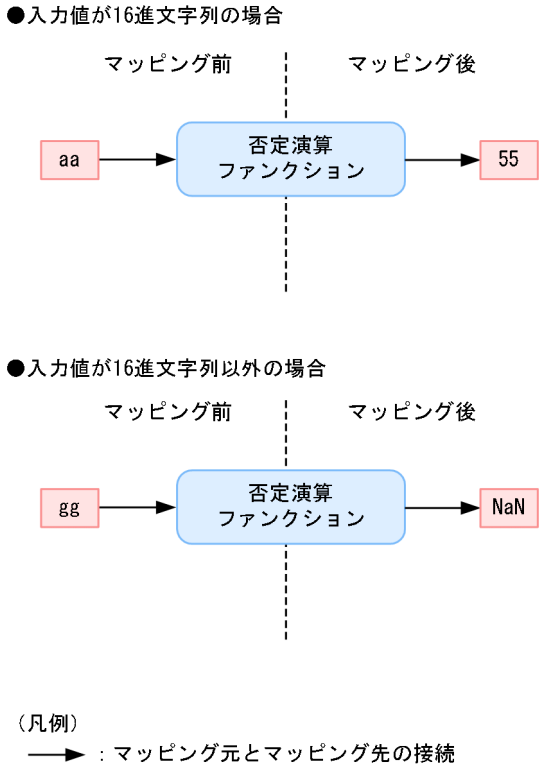
入力値を 16 進文字列であるものとし，否定演算を実行します。入力値が 16 進文字列以外の場合は，エラーの値（NaN）を返します。入力値が 64 ビットを超える場合，64 ビットより上位ビットを切り捨てます。

変換元ノード，ファンクションは変換実行時に文字列化されます。

(1)　使用するファンクション

否定演算を実行するには，否定演算ファンクションを使用します。使用例を次に示します。

図 7-11　否定演算ファンクションの使用例



(2) 設定手順

否定演算を設定する手順を次に示します。

1. パレットから否定演算ファンクション（not）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［否定演算］ダイアログを表示させます。
   - 否定演算ファンクションを右クリックして，［設定］を選択します。
   - 否定演算ファンクションをダブルクリックします。

   ［否定演算］ダイアログが表示されます。

   ［否定演算］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.28　否定演算ダイアログ」を参照してください。
4. ［入力］に否定演算の入力値を指定する場合，［ノード選択］ボタンをクリックします。

   ［ノード選択］ダイアログが表示されます。
5. 入力値となるノードセットを指定します。

   ［入力］に否定演算の入力値が設定されます。
6. ［入力］に指定されている入力値に条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
7. ［入力］に否定演算の対象とするファンクションを指定したい場合，［ファンクション選択］ボタンをクリックします。

   ［ファンクション選択］ダイアログが表示されます。

   ［ファンクション選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.9　ファンクション選択ダイアログ」を参照してください。
8. ［OK］ボタンをクリックします。

7.5.11　論理演算を使用する

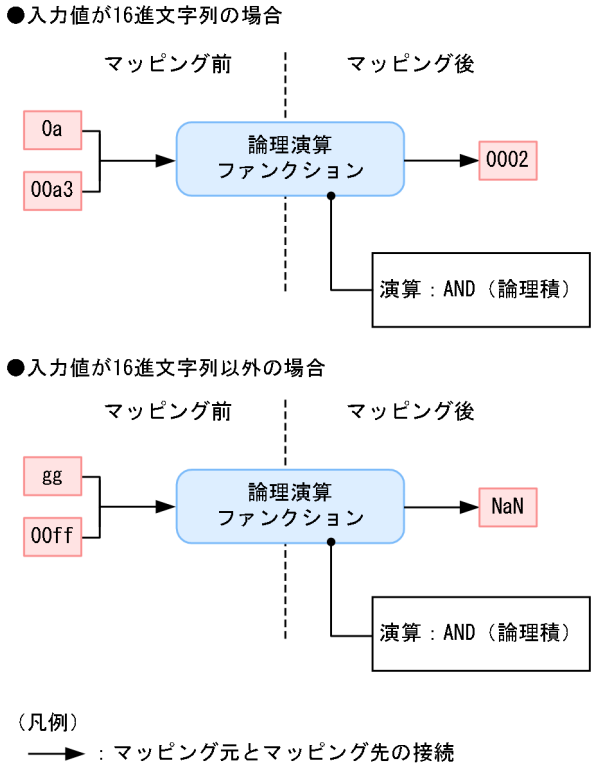
入力値を 16 進文字列であるものとし，論理演算を実行します。演算の種類として，「AND」（論理積），「OR」（論理和），「XOR」（排他的論理和）が指定できます。入力値が 16 進文字列以外の場合は，エラーの値（NaN）を返します。入力値が 64 ビットを超える場合，64 ビットより上位ビットを切り捨てます。

なお，論理演算ファンクションは，ビッグエンディアンで処理します。2 つの入力値の桁数が異なる場合は，大きい桁数に合わせて演算します。変換元ノード，ファンクションは変換実行時に文字列化されます。

(1)　使用するファンクション

論理演算を実行するには，論理演算ファンクションを使用します。使用例を次に示します。

図 7-12　論理演算ファンクションの使用例



(2)　設定手順

論理演算を設定する手順を次に示します。

1. パレットから論理演算ファンクション（bitop）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［論理演算］ダイアログを表示させます。
   - 論理演算ファンクションを右クリックして，［設定］を選択します。
   - 論理演算ファンクションをダブルクリックします。

   ［論理演算］ダイアログが表示されます。
   ［論理演算］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.29　論理演算ダイアログ」を参照してください。
4. ［入力 1］に論理演算の入力値を指定する場合，［入力 1］の［ノード選択］ボタンをクリックします。

［ノード選択］ダイアログが表示されます。

5. 入力値となるノードセットを指定します。
［入力 1］に論理演算の入力値が設定されます。

6. ［入力 2］に論理演算の入力値を指定する場合，［入力 2］の［ノード選択］ボタンをクリックします。
［ノード選択］ダイアログが表示されます。

7. 入力値となるノードセットを指定します。
［入力 2］に論理演算の入力値が設定されます。

8. ［入力 1］または［入力 2］に指定されている入力値に条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。
［ノード条件設定］ダイアログが表示されます。
ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

9. ［入力 1］または［入力 2］に論理演算の対象とするファンクションを指定したい場合，［ファンクション選択］ボタンをクリックします。
［ファンクション選択］ダイアログが表示されます。
［ファンクション選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.9　ファンクション選択ダイアログ」を参照してください。

10. ［演算］から演算方法を指定します。

11. ［OK］ボタンをクリックします。

7.5.12　シフト演算を使用する

入力値を 16 進文字列であるものとし，シフト演算を実行します。シフト演算では，シフト量，シフト方向，シフト種別で指定した方法でシフト演算した結果を出力サイズで指定したサイズ分，下位ビットから取得し，返します。入力値が 16 進文字列以外の場合は，エラーの値（NaN）を返します。入力値が 64 ビットを超える場合，64 ビットより上位ビットを切り捨てます。

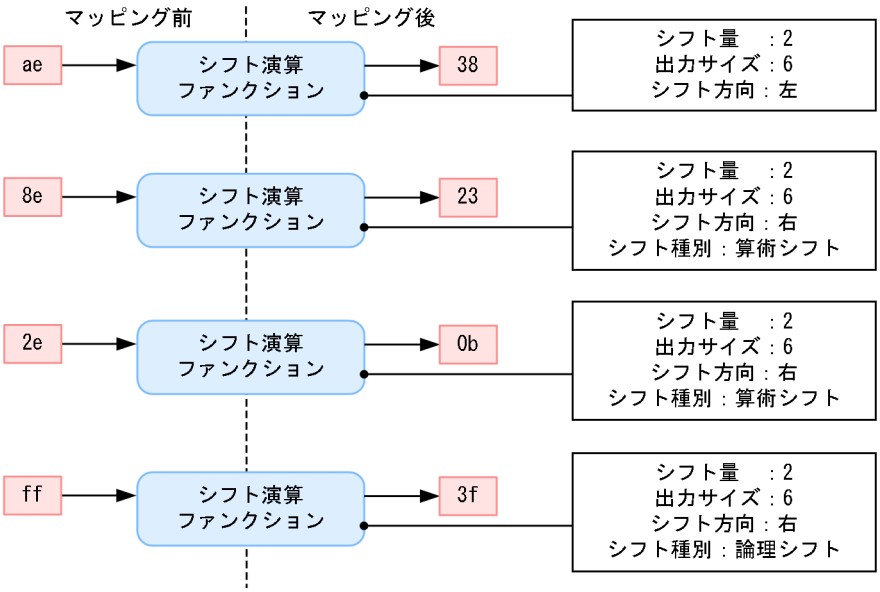
なお，シフト演算ファンクションは，ビッグエンディアンで処理します。

(1)　使用するファンクション

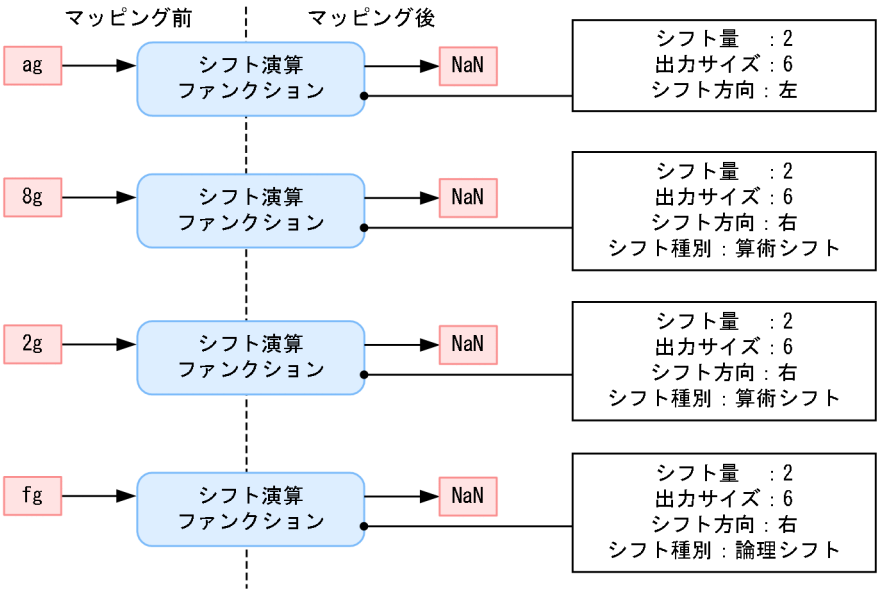
シフト演算を実行するには，シフト演算ファンクションを使用します。使用例を次に示します。

図 7-13　シフト演算ファンクションの使用例

●入力値が16進文字列の場合



●入力値が16進文字列以外の場合



（凡例）

⟶ ：マッピング元とマッピング先の接続

(2)　設定手順

シフト演算を設定する手順を次に示します。

1. パレットからシフト演算ファンクション（shift）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［シフト演算］ダイアログを表示させます。

- シフト演算ファンクションを右クリックして，［設定］を選択します。
- シフト演算ファンクションをダブルクリックします。

［シフト演算］ダイアログが表示されます。

［シフト演算］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.30　シフト演算ダイアログ」を参照してください。

4. ［入力］にシフト演算の入力値を指定する場合，［ノード選択］ボタンをクリックします。

［ノード選択］ダイアログが表示されます。

5. 入力値となるノードセットを指定します。

［入力］にシフト演算の入力値が設定されます。

6. ［入力］に指定されている入力値に条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

［ノード条件設定］ダイアログが表示されます。

ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

7. ［入力］にシフト演算の対象とするファンクションを指定したい場合，［ファンクション選択］ボタンをクリックします。

［ファンクション選択］ダイアログが表示されます。

［ファンクション選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.9　ファンクション選択ダイアログ」を参照してください。

8. シフト量を指定します。

9. 出力サイズを指定します。

10. シフト方向を指定します。

11. シフト種別を指定します。

12. ［OK］ボタンをクリックします。

(3)　実行例

実行例を次に示します。

入力※1	シフト量	出力サイズ	シフト方向	シフト種別	出力結果※1※2※3
76(0111 0110)	3	8	左	算術	b0(1011 0000)
			右	算術	0e(0000 1110)

入力※1	シフト量	出力サイズ	シフト方向	シフト種別	出力結果※1※2※3
76(0111 0110)	3	8	右	論理	0e(**000**0 1110)
89(1000 1001)			左	算術	48(0100 1**000**)
			右	算術	f1(**111**1 0001)
			右	論理	11(**000**1 0001)
76(0111 0110)	2	6	左	算術	18(0001 1000) [d8(1101 10**00**)]
			右	算術	1d(0001 1101) [1d(**00**01 1101)]
			右	論理	1d(0001 1101) [1d(**00**01 1101)]
89(1000 1001)			左	算術	24(0010 0100) [24(0010 01**00**)]
			右	算術	22(0010 0010) [e2(**11**10 0010)]
			右	論理	22(0010 0010) [22(**00**10 0010)]

注※1

( )内は，2 進整数文字列表記です。

注※2

[ ]内は，出力ビット数で切り捨てられる前の値です。

注※3

太字は，補完ビットです。

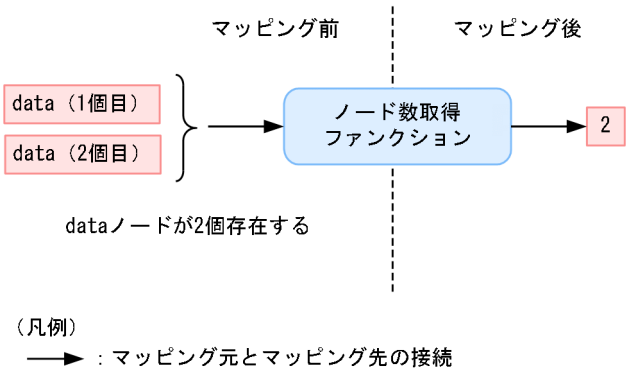
7.5.13　ノード数を代入する

マッピング元のノードセットのノード数をマッピング先に代入します。

(1)　使用するファンクション

マッピング元のノードセットのノード数をマッピング先に代入するには，ノード数取得ファンクションを使用します。ノード数取得ファンクションを使用した例を次に示します。

図 7-14　ノード数取得ファンクションの使用例



(2) 設定手順

マッピング元のノードセットのノード数をマッピング先に代入する手順を次に示します。

1. パレットからノード数取得ファンクション（count）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［ノード数取得］ダイアログを表示させます。
   - ノード数取得ファンクションを右クリックして，［設定］を選択します。
   - ノード数取得ファンクションをダブルクリックします。

   ［ノード数取得］ダイアログが表示されます。

   ［ノード数取得］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.32　ノード数取得ダイアログ」を参照してください。
4. ［入力］に指定されている変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［OK］ボタンをクリックします。

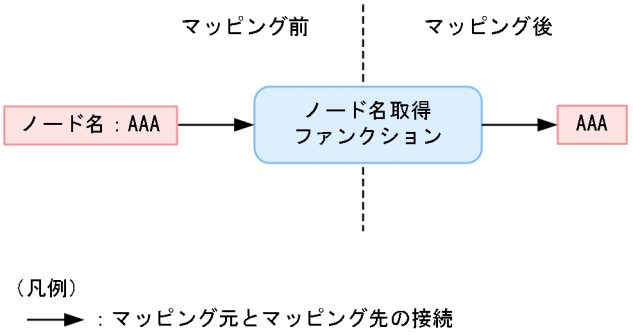
7.5.14　ノード名を代入する

マッピング元のノード名をマッピング先に代入します。マッピング元（変換元ノード）のインスタンスが存在しない場合，空文字列を代入します。

(1) 使用するファンクション

マッピング元のノード名をマッピング先に代入するには，ノード名取得ファンクションを使用します。ノード名取得ファンクションを使用した例を次に示します。

図 7-15　ノード名取得ファンクションの使用例



(2)　設定手順

マッピング元のノード名をマッピング先に代入する手順を次に示します。

1. パレットからノード名取得ファンクション（name）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［ノード名取得］ダイアログを表示させます。
   - ノード名取得ファンクションを右クリックして，［設定］を選択します。
   - ノード名取得ファンクションをダブルクリックします。

   ［ノード名取得］ダイアログが表示されます。

   ［ノード名取得］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.31　ノード名取得ダイアログ」を参照してください。
4. ［入力］に指定されている変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［OK］ボタンをクリックします。

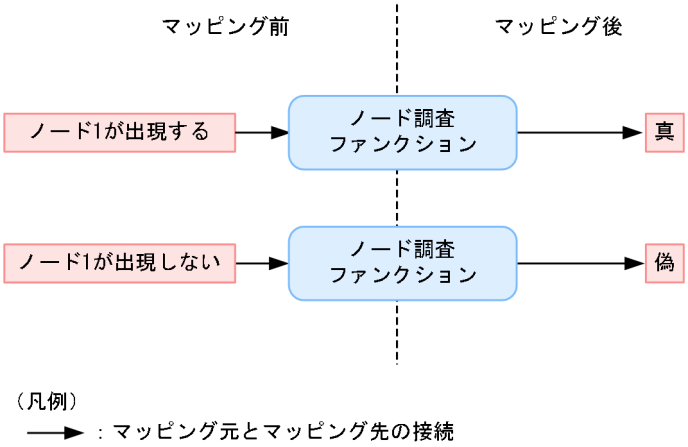
7.5.15　ノードの有無を調査する

指定したマッピング元の変換元ノードが存在するかを調査して，マッピング先にその論理値を代入します。

(1)　使用するファンクション

指定したマッピング元の変換元ノードが存在するかを調査して，マッピング先にその論理値を代入するには，ノード調査ファンクションを使用します。ノード調査ファンクションを使用した例を次に示します。

図 7-16　ノード調査ファンクションの使用例



(2)　設定手順

指定したマッピング元の変換元ノードが存在するか調査する手順を次に示します。

1. パレットからノード調査ファンクション（exist）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［ノード調査］ダイアログを表示させます。
   - ノード調査ファンクションを右クリックして，［設定］を選択します。
   - ノード調査ファンクションをダブルクリックします。

   ［ノード調査］ダイアログが表示されます。

   ［ノード調査］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.33　ノード調査ダイアログ」を参照してください。
4. ［入力］に指定されている変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。
5. ［OK］ボタンをクリックします。

7.5.16　くり返しを対応づける

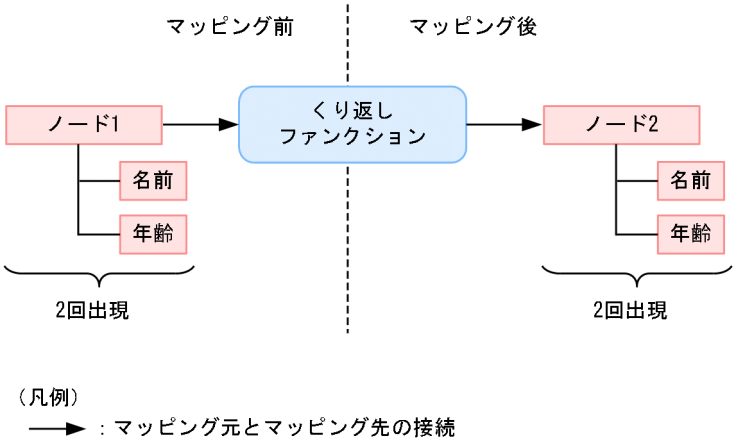
複数回出現する変換先ノードに対して，複数回出現する変換元ノードをマッピング元として，くり返しを対応づけます。また，複数回出現する変換元ノードのインスタンスをソートすることもできます。

ここでは，くり返しを対応づける基本的な方法を説明します。くり返しの合成，くり返し依存対象などの内容については，「7.6　くり返しの設定」を参照してください。

(1)　使用するファンクション

複数回出現する変換先ノードに対して，複数回出現する変換元ノードをマッピング元として，くり返しを対応づけるには，くり返しファンクションを使用します。くり返しファンクションを使用した例を次に示します。

図 7-17　くり返しファンクションの使用例



(2)　設定手順

複数回出現する変換先ノードに対して，複数回出現する変換元ノードをマッピング元として，くり返しを対応づける手順を次に示します。

1. パレットからくり返しファンクション（loop）を選択して，マッピングビューアに配置します。

2. マッピング線を設定します。

3. 次のどちらかの方法で［くり返し］ダイアログを表示させます。

   - くり返しファンクションを右クリックして，［設定］を選択します。
   - くり返しファンクションをダブルクリックします。

   ［くり返し］ダイアログが表示されます。

   ［くり返し］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.7　くり返し設定ダイアログ」を参照してください。

4. 変換元ノードのインスタンスをソートする場合，［追加］ボタンをクリックします

   ［ソート条件追加/編集］ダイアログが表示されます。

   ［ソート条件追加/編集］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.35　ソート条件追加/編集ダイアログ」を参照してください。

5. ［ノード選択］ボタンをクリックします。

［ノード選択］ダイアログが表示されます。［ノード選択］ダイアログで，ソートさせたいインスタンスの変換元ノードを指定します。

6. 次の項目を設定します。

- ［順序］
  ［昇順］または［降順］を選択します。
- ［言語］
  ［日本語］，［英語］，または［自動］を選択します。
- ［データタイプ］
  ［テキスト］または［数値］を選択します。
- ［優先順位］
  ［大文字］または［小文字］を選択します。

7. ［OK］ボタンをクリックします。
［くり返し］ダイアログの［ソート条件］に，［ソート条件追加/編集］ダイアログで指定した内容が設定されます。

8. ［ソート条件］にキーを追加する場合，手順 4～手順 7.をくり返します。
キーは 8 つまで設定できます。

9. ［ソート条件］の順番を変える場合，［上へ］ボタンまたは［下へ］ボタンをクリックします。
［ソート条件］でいちばん上に設定した条件から順にソートされます。
例えば，いちばん上に日付，2 番目に名前を条件として設定した場合，まず全体を日付でソートします。次に，ソートした中に同じ日付があった場合，その日付の中でさらに名前でソートします。

10. ［OK］ボタンをクリックします。

注意事項

- 同じ変換先ノードに対して，くり返しをネストして定義しないでください。つまり，同じ変換先ノードに対して，くり返しファンクションを直列に接続しないでください。
- 同じ変換先ノードに対して，同じくり返し依存対象の代入線を複数設定しないでください。くり返し依存対象については，「7.6.3　くり返し依存対象の対応づけ」を参照してください。
- 同じ変換先ノードに対して，同じくり返し依存対象のくり返し対応線および代入線を同時に設定しないでください。くり返し依存対象については，「7.6.3　くり返し依存対象の対応づけ」を参照してください。
- 変換先ノードにくり返しを定義した場合，変換先ノードの子孫ノードにはくり返し依存対象を設定してください。くり返し依存対象については，「7.6.3　くり返し依存対象の対応づけ」を参照してください。

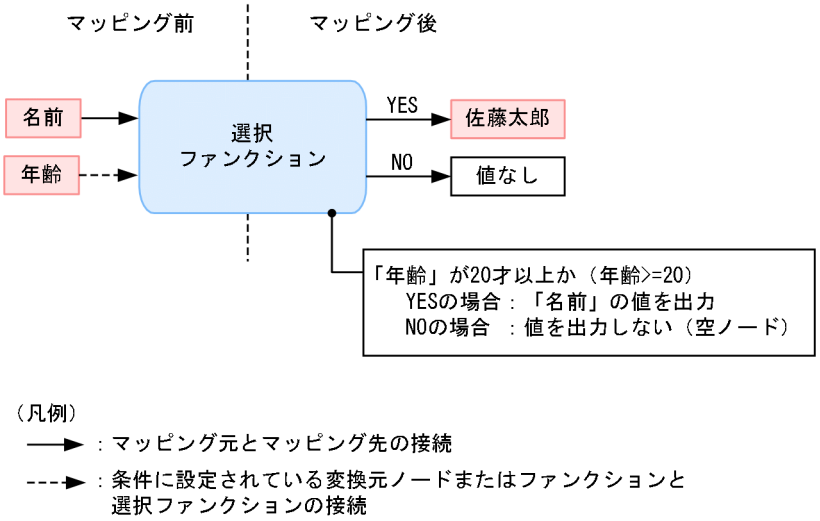
7.5.17　条件によって異なる値を出力する

ある条件を設定して，その条件に合致する場合または合致しない場合で，異なる値を出力します。

(1)　使用するファンクション

ある条件を設定して，その条件に合致する場合または合致しない場合で，異なる値を出力させるには，選択ファンクションを使用します。選択ファンクションを使用した例を次に示します。

図 7-18　選択ファンクションの使用例



(2)　設定手順

ある条件を設定して，その条件に合致する場合または合致しない場合で，異なる値を出力させる手順を次に示します。

1. パレットから選択ファンクション（choose）を選択して，マッピングビューアに配置します。

2. マッピング線を設定します。

   パレットのツールではなく，［選択］ダイアログでマッピング元を設定してマッピング線を設定する場合は，手順.7～手順.11 を実施します。

3. 次のどちらかの方法で［選択］ダイアログを表示させます。

   - 選択ファンクションを右クリックして，［設定］を選択します。
   - 選択ファンクションをダブルクリックします。

   ［選択］ダイアログが表示されます。

   手順 2.でマッピング線を設定した場合は，［条件と出力値］の［出力値］にマッピング元が設定されています。

   ［選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.36　選択ダイアログ」を参照してください。

4. 次のどちらかの方法で［条件追加/編集］ダイアログを表示させます。

- ［条件と出力値］の［出力値］にマッピング元が設定されている場合

  ［条件と出力値］の［出力値］に設定されているマッピング元を選択して，［編集］ボタンをクリックします。

- ［条件と出力値］の［出力値］にマッピング元が設定されていない場合

  ［追加］ボタンをクリックします。

［条件追加/編集］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.37　条件追加/編集ダイアログ」を参照してください。

5. ［条件設定］ボタンをクリックします。

［条件設定］ダイアログが表示されます。

6. 条件を設定します。

［条件設定］ダイアログでの条件設定の方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.12　条件設定ダイアログ（ノード条件の設定)」を参照してください。

7. ［条件が true の場合］欄で次のように指定します。

- 変換先ノードを出力させる場合

  ［ノードを出力する］チェックボックスにチェックします。

- 変換先ノードを出力させない場合

  ［ノードを出力しない］チェックボックスにチェックします。

なお，［条件と出力値］の［出力値］ですでにマッピング元が設定されている場合，［値］にマッピング元が表示されます。マッピング元を変更しないときは，手順.10 に進んでください。

8. 手順 7.で［ノードを出力する］チェックボックスにチェックした場合，［出力値］欄で次のように指定します。

- 変換先ノードに値を出力させる場合

  ［値を出力する］チェックボックスにチェックします。

- 変換先ノードに値を出力させない場合

  ［空ノード］チェックボックスにチェックします。

9. 手順 8.で［値を出力する］チェックボックスにチェックした場合，［値］を次のように指定します。

- 変換元ノードの値を出力させる場合

  ［ノード選択］ボタンをクリックすると，［ノード選択］ダイアログが表示されます。［ノード選択］ダイアログで，値を出力させたい変換元ノードを選択します。

- ファンクションの値を出力させる場合

  ［ファンクション選択］ボタンをクリックすると，［ファンクション選択］ダイアログが表示されます。［ファンクション選択］ダイアログで，値を出力させたいファンクションを選択します。

10. **[値] に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，[ノード条件設定] ボタンをクリックします。**

    [ノード条件設定] ダイアログが表示されます。

    ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，[ノード条件設定] ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

11. **[OK] ボタンをクリックします。**

    [条件追加/編集] ダイアログが閉じます。

12. **条件を追加または編集する場合，手順 4.～手順.11 をくり返します。**

    [条件と出力値] に複数の条件およびマッピング元を指定した場合，いちばん上に表示された条件から順に，条件を判定します。

13. **どの条件にも合致しない場合のマッピング元を，[選択] ダイアログの [どの条件にも合致しない場合] 欄で設定します。**

    設定方法は，[条件追加/編集] ダイアログの [条件が true の場合] 欄と同じです。

14. **[OK] ボタンをクリックします。**

    [選択] ダイアログで設定した条件中に，マッピング元以外の変換元ノードまたはファンクションを指定した場合，その条件中の変換元ノードまたはファンクションと選択ファンクションの間にはマッピング線 ┈┈┈┈→ （条件線）が設定されます。

注意事項

- 複合内容の変換先ノードにマッピングする場合，[値を出力する] を選択して [値] を指定しないでください。また，複合内容以外の変換先ノードにマッピングする場合，[値を出力する] を選択したときは，必ず [値] を指定してください。
- すべての子要素の最小出現回数が 1 回以上に設定されている choice 要素にマッピングする場合，[空ノード] を指定しないでください。
- 最小出現回数が 1 回以上に設定されている子要素を持つ要素（choice 要素以外）にマッピングする場合，[空ノード] を指定しないでください。
- 最小出現回数が 1 回以上に設定されているノードにマッピングする場合，[ノードを出力しない] を指定しないでください。
- 出力値にくり返しファンクションを一つでも設定した場合，ほかの出力値もすべてくり返しファンクションを設定してください。また，出力値にくり返しファンクション以外を設定した場合，ほかの出力値もすべてくり返しファンクション以外を設定してください。

7.5.18　指定した値を代入する

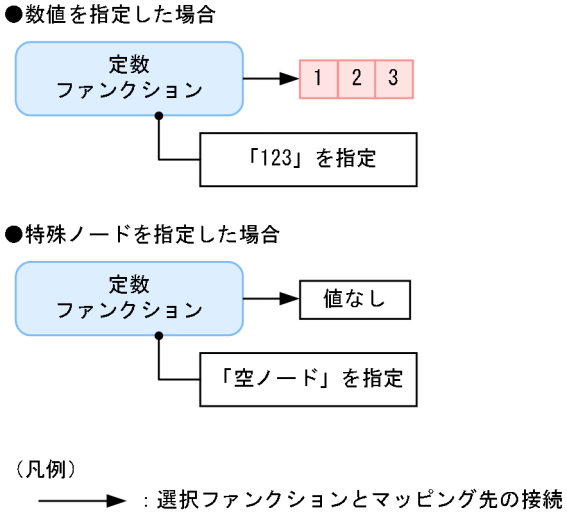
マッピング先に指定した値を代入します。指定できる値を次に示します。

- 文字列
- 数値
- 論理値（「真」または「偽」）
- 特殊ノード（「ノード出力なし」または「空ノード」）

(1)　使用するファンクション

マッピング先に，指定した値を代入するには，定数ファンクションを使用します。定数ファンクションに指定した値は，複数のマッピング先に代入できます。定数ファンクションにはマッピング元がありません。したがって，定数ファンクションの値がそのままマッピング先に代入されます。定数ファンクションの使用例を次の図に示します。

図 7-19　定数ファンクションの使用例



(2)　設定手順

マッピング先に，指定した値を代入する手順を次に示します。

1. パレットから定数ファンクション（const）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［定数］ダイアログを表示させます。
   - 定数ファンクションを右クリックして，［設定］を選択します。
   - 定数ファンクションをダブルクリックします。

   ［定数］ダイアログが表示されます。

［定数］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.41　定数ダイアログ」を参照してください。

4. マッピング先に代入する値を指定します。

- 文字列を指定する場合
  ［文字列］ラジオボタンを選択し，［値］に代入する値を指定します。
- 数値を指定する場合
  ［数値］ラジオボタンを選択し，［値］に代入する数値を指定します。
- 論理値を指定する場合
  ［論理値］ラジオボタンを選択し，［論理値］欄で［真］または［偽］を選択します。
- 特殊ノードを指定する場合
  ［特殊ノード］ラジオボタンを選択し，［値］欄で［ノード出力なし］または［空ノード］を選択します。

> **参考**
>
> 空文字列を指定したい場合
>
> 手順４で［文字列］ラジオボタンを選択し，［値］に何も入力しなければ空文字列が設定できます。

5. ［OK］ボタンをクリックします。

> **注意事項**
>
> - 特殊ノードを指定する場合，マッピング先には変換先ノードを指定してください。
> - 特殊ノード以外を指定する場合，マッピング先には複合型ノード以外のノードを指定してください。
> - 特殊ノードでノード出力なしを指定した場合，マッピング先には最小出現回数が 0 回の変換先ノードを指定してください。
> - すべての子要素の最小出現回数が 1 回以上に設定されている choice 要素にマッピングする場合，特殊ノードの空ノード以外を指定してください。
> - 最小出現回数が 1 回以上に設定されている子要素を持つ要素（choice 要素以外）にマッピングする場合，特殊ノードの空ノード以外を指定してください。

7.5.19　変換テーブルを利用して値を変換する

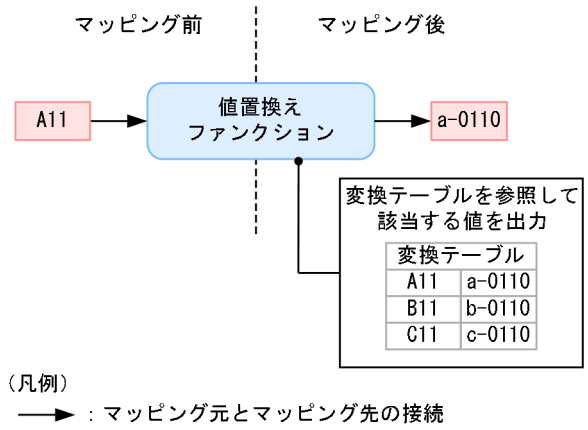
変換テーブルを利用して，マッピング元の値を変換してマッピング先に代入します。

(1) 使用するファンクション

変換テーブルを利用して，マッピング元の値を変換してマッピング先に代入するには，値置換えファンクションを使用します。

値置換えファンクションを使用した例を次に示します。

図 7-20 値置換えファンクションの使用例



(2) 設定手順

変換テーブルを利用して，マッピング元の値を変換してマッピング先に代入するには，まず，変換テーブルを作成・保存します。作成した変換テーブルの情報を J2EE サーバ用ユーザプロパティファイル（usrconf.properties）に記述すると，変換テーブルが利用できる状態になります。変換テーブルが利用できる状態になったあと，値置換えファンクションを定義します。

(a) 変換テーブルの作成と保存

変換テーブルの作成と保存について説明します。

ポイント

作成した変換テーブルはパッケージングの対象にはなりません。そのため，実行環境での実運用開始後にも，任意のタイミングで編集できます。編集された内容は，次回のシステム起動時に反映されます。

変換テーブルの作成

変換テーブルは，csv ファイルで作成します。csv ファイルの形式を次に示します。

```
”値A01”,”値B01”
”値A02”,”値B02”
”値A03”,”値B03”
        :
```

同じ行に記述した「値 Ann」と「値 Bnn」を一対の組み合わせとして値を置き換えます。置き換え対象の文字列はダブルクォーテーション（"）で囲んで記述します。置き換え対象の文字列にダブルクォー

テーションが含まれる場合，ダブルクォーテーションをエスケープ文字として使用できます。エスケープ対象外の文字をエスケープした場合，エスケープ文字は無視されて，以降の文字が処理されます。

なお，変換テーブルに使用できる文字コードは次のとおりです。

- MS932
- UTF8
- UTF16（ビッグエンディアン，またはリトルエンディアン）

変換テーブルの記述に使用する文字について次の表に示します。

表 7-4　変換テーブルに使用する文字

対象	使用する文字
開始文字	開始文字は不要です。
終了文字	次のどちらかの改行コードです。 • LF（0x0A） • CR + LF（0x0D0A） 実行する OS が UNIX の場合は LF，Windows の場合は CR+LF を使用します。 注 LF：ラインフィード CR：キャリッジリターン
中間区切り文字	コンマ（,）を利用します。

注意事項

- 変換テーブルに記述した変換対象は文字列として扱います。そのため，値置換えファンクションへの入力値が数値の場合，文字列に変換して処理されます。
- 変換対象および入力値に，空要素は指定できません。
- 属性値を入力する場合，入力値は正規化（タブをスペースに置き換えたり，連続したスペースを 1 つのスペースに置き換えたりするなど）した値で扱います。そのため，変換テーブルに記述した検索キー値も，正規化後の値を考慮する必要があります。

ポイント

「値 Ann」から「値 Bnn」の変換にするか，「値 Bnn」から「値 Ann」の変換にするかは，値置換えファンクションを定義するときに設定できます。

変換テーブルの保存

作成した変換テーブルは，任意の場所に保存します。

(b) J2EE サーバ用ユーザプロパティファイル（usrconf.properties）への登録

変換テーブルを作成したあとには，usrconf.properties に作成した変換テーブルを登録します。変換テーブルの登録方法の詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.3.9　J2EE サーバ用ユーザプロパティファイル（usrconf.properties）を設定する」の値置換えファンクションで使用する変換テーブルの登録に関する説明を参照してください。

(c) 値置換えファンクションの定義

値置換えファンクションを利用して，マッピング元の値を変換してマッピング先に代入するための手順を次に示します。

1. パレットから値置換えファンクション（replace）を選択して，マッピングビューアに配置します。

2. マッピング線を設定します。

3. 次のどちらかの方法で［値置換え］ダイアログを表示させます。

   - 値置換えファンクションを右クリックして，［設定］を選択します。
   - 値置換えファンクションをダブルクリックします。

   ［値置換え］ダイアログが表示されます。

   ［値置換え］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.38　値置換えダイアログ」を参照してください。

4. ［入力］に変換元ノードが指定されている場合，その変換元ノードに条件を設定して，その条件を満たす場合だけマッピングさせたいとき，［ノード条件設定］ボタンをクリックします。

   ［ノード条件設定］ダイアログが表示されます。

   ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

5. ［変換テーブル ID］に J2EE サーバ用ユーザプロパティファイル（usrconf.properties）に記述した変換テーブルの名称を指定します。

6. ［検索キー列指定］で，変換テーブルの左列と右列のどちらを値の置き換え元とするか選択します。

7. ［検索失敗時の動作］で，マッピング元からの入力値が変換テーブルに存在しなかった場合のシステムの動作を選択します。

   デフォルト値を出力してマッピング先に代入する場合

   ［デフォルト値を代入］をラジオボタンで選択し，［値］にデフォルト値として利用する文字列を入力します。

   変換エラーにする場合

   ［変換エラー］をラジオボタンで選択します。

8. ［OK］ボタンをクリックします。

7.5.20　基数変換する

入力値を入力基数で指定した基数であるものとし，入力値を出力基数に指定した基数に変換します。入力値が入力基数で指定した基数以外の場合は，エラーの値（NaN）を返します。入力値が 64 ビットを超える場合，64 ビットより上位ビットを切り捨てます。

なお，基数変換ファンクションは，ビッグエンディアンで処理します。

基数変換の仕様を次に示します。

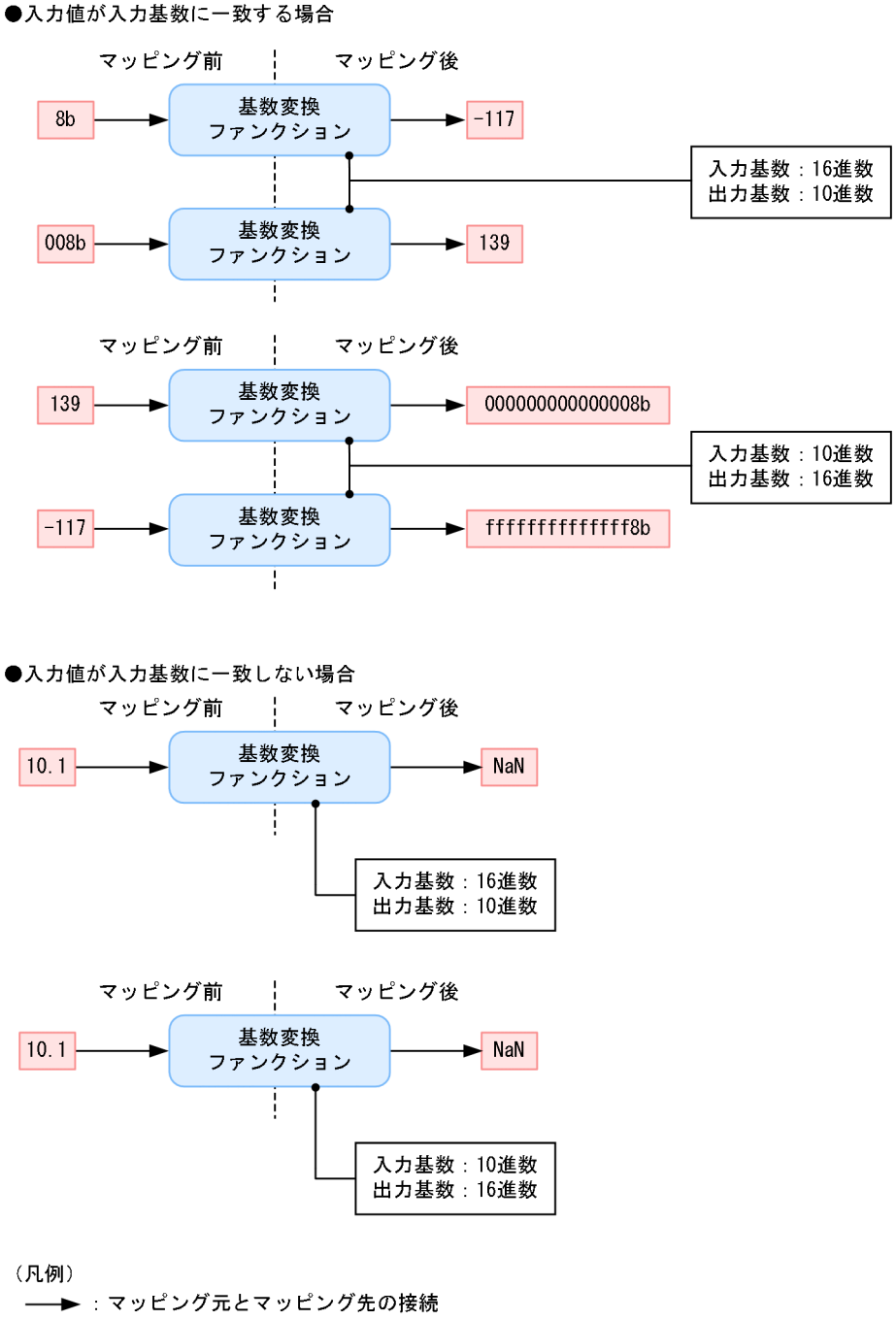
- 入力基数が 10 進数の場合，64 ビットとして基数変換する。
- 出力基数が 10 進数の場合，4 ビット単位に拡張した入力データの最上位ビットを符号ビットとする。
- 入力基数と出力基数が 10 進数でない場合，変換元のビット長に合わせる。入力基数が 2 進数の場合，4 ビットに満たないデータは 4 ビット単位になるよう拡張する。
- 入力基数の入力値の形式を次に示す。

入力基数	形式
2 進数	[01]+
10 進数	[+-]?[0-9]+
16 進数	[0-9a-fA-F]+

(1)　使用するファンクション

入力値を出力基数に指定した基数に変換するには，基数変換ファンクションを使用します。使用例を次に示します。

図 7-21　基数変換ファンクションの使用例



(2)　設定手順

入力値を出力基数に指定した基数に変換する手順を次に示します。

1. パレットから基数変換ファンクション（radix）を選択して，マッピングビューアに配置します。
2. マッピング線を設定します。
3. 次のどちらかの方法で［基数変換］ダイアログを表示させます。
   - 基数変換ファンクションを右クリックして，［設定］を選択します。

- 基数変換ファンクションをダブルクリックします。

［基数変換］ダイアログが表示されます。

［基数変換］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.39　基数変換ダイアログ」を参照してください。

4. ［入力］に基数変換の入力値を指定する場合，［ノード選択］ボタンをクリックします。

［ノード選択］ダイアログが表示されます。

5. 入力値となるノードセットを指定します。

［入力］に基数変換の入力値が設定されます。

6. ［入力］に指定されている入力値に条件を設定して，その条件を満たす場合だけマッピングさせたい場合，［ノード条件設定］ボタンをクリックします。

［ノード条件設定］ダイアログが表示されます。

ノード条件の設定方法については，「7.7　ノード条件を設定する」を参照してください。また，［ノード条件設定］ダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6.11　ノード条件設定ダイアログ」を参照してください。

7. ［入力］に基数変換の対象とするファンクションを指定したい場合，［ファンクション選択］ボタンをクリックします。

［ファンクション選択］ダイアログが表示されます。

［ファンクション選択］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.9　ファンクション選択ダイアログ」を参照してください。

8. 入力基数を指定します。

9. 出力基数を指定します。

10. ［OK］ボタンをクリックします。

(3)　実行例

実行例を次に示します。

入力	基数変換	結果
1101	2 進数→10 進数	-3
0100	2 進数→10 進数	4
1101	2 進数→16 進数	d
11	10 進数→2 進数	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1011※1
11	10 進数→16 進数	00 00 00 00 00 00 00 0b※1

入力	基数変換	結果
11	16 進数→2 進数	0001 0001
11	16 進数→10 進数	17
8b	16 進数→10 進数	-117
1102	2 進数→10 進数	NaN※2
110f	10 進数→16 進数	NaN※2

注※1

太字は，補完ビットです。

注※2

入力値を入力基数で指定した基数に変換できないため，結果が「NaN」となります。

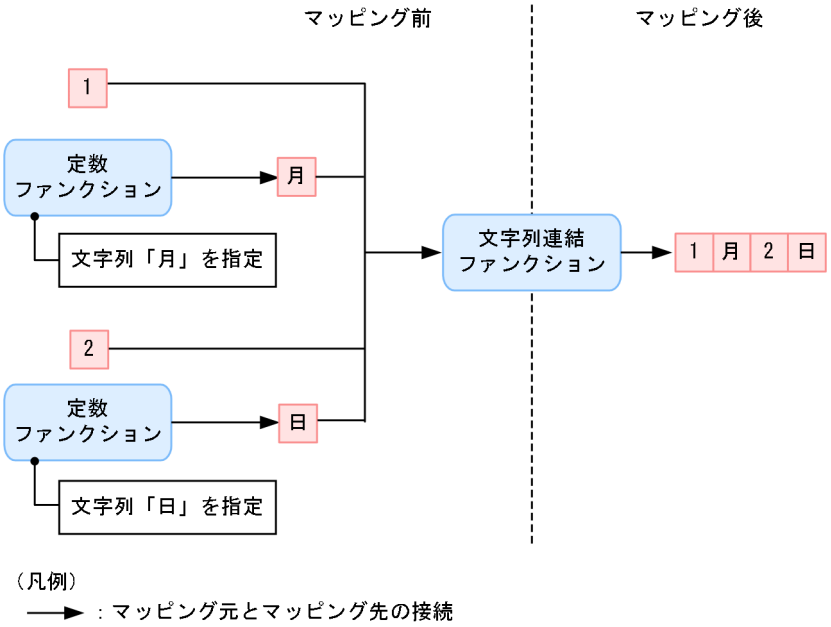
7.5.21　変換元ノードの値にある値を付加する

変換元ノードの値にある値を付加して，マッピング先に代入します。

(1)　使用するファンクション

変換元ノードの値にある値を付加して，マッピング先に代入するには，文字列連結ファンクションおよび定数ファンクションを組み合わせて使用します。文字列連結ファンクションおよび定数ファンクションを組み合わせて使用した例を次に示します。

図 7-22　文字列連結ファンクションおよび定数ファンクションを組み合わせた使用例



(2)　設定手順

変換元ノードの値にある値を付加して，マッピング先に代入する手順を次に示します。

1. パレットから定数ファンクション（const）を選択して，マッピングビューアに配置します。
2. 定数ファンクションで値を指定します。
   値を指定する方法については，「7.5.18　指定した値を代入する」を参照してください。
3. パレットから文字列連結ファンクション（concat）を選択して，マッピングビューアに配置します。
4. 定数ファンクションで指定した値と変換元ノードの値を，文字列連結ファンクションで連結します。
   連結方法については，「7.5.1　複数の文字列を連結する」を参照してください。

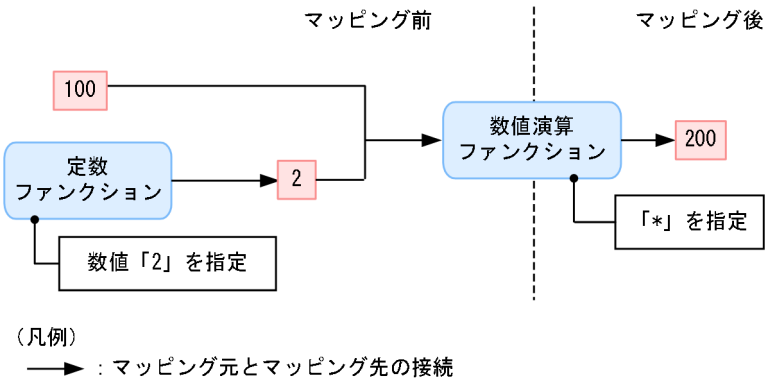
7.5.22　変換元ノードの値を倍にする

変換元ノードの値を倍にして，マッピング先に代入します。

(1)　使用するファンクション

変換元ノードの値を倍にして，マッピング先に代入するには，数値演算ファンクションおよび定数ファンクションを組み合わせて使用します。数値演算ファンクションおよび定数ファンクションを組み合わせて使用した例を次に示します。

図 7-23　演算ファンクションおよび定数ファンクションを組み合わせた使用例



(2)　設定手順

変換元ノードの値を倍にして，マッピング先に代入する手順を次に示します。

1. パレットから定数ファンクション（const）を選択して，マッピングビューアに配置します。
2. 定数ファンクションで値を指定します。
   値を指定する方法については，「7.5.18　指定した値を代入する」を参照してください。

3. パレットから数値演算ファンクション（calc）を選択して，マッピングビューアに配置します。

4. 定数ファンクションで指定した値と変換元ノードの値を，数値演算ファンクションで演算します。
   演算方法については，「7.5.7　数値を演算する」を参照してください。

7.5.23　ユーザが作成した任意の Java プログラムを呼び出す

ユーザが作成した任意の Java プログラムを呼び出して，ファンクションとして使用します。Service Architect が提供するほかのファンクションではできない加工をしたい場合に使用します。

Service Architect では，標準のファンクションのほかに，値の加工，取得に便利なカスタムファンクションを提供しています。提供しているカスタムファンクションについては，「7.18　提供しているカスタムファンクション」を参照してください。

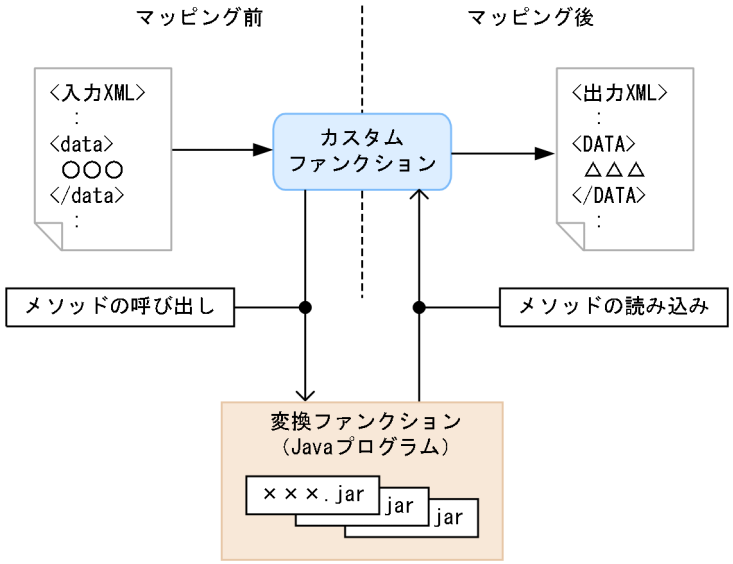
なお，呼び出す Java プログラムの作成方法については，「7.10　カスタムファンクションで使用する Java プログラムの作成」を参照してください。

(1)　使用するファンクション

ユーザが作成した任意の Java プログラムを呼び出すには，カスタムファンクションを使用します。

カスタムファンクションから Java プログラムを呼び出す方法を次の図に示します。なお，カスタムファンクションでは，呼び出す Java プログラムのことを**変換ファンクション**と呼びます。

図 7-24　カスタムファンクションから Java プログラム（変換ファンクション）を呼び出す方法



(2)　設定手順

ユーザが作成した任意の Java プログラムを呼び出す手順を次に示します。

1. Eclipse のメニューから［ウィンドウ］－［設定］を選択します。
   ［設定］ダイアログが表示されます。

2. ダイアログ左側で［HCSC-Definer］－［データ変換］を選択します。
   データ変換定義画面の設定項目がダイアログ右側に表示されます。

3. ［カスタムファンクション］タブをクリックします。
   ［設定］ダイアログの［カスタムファンクション］タブが表示されます。

4. 変換ファンクション定義ファイルを指定し，［Apply and Close］ボタンをクリックします。

5. パレットからカスタムファンクション（custom）を選択して，マッピングビューアに配置します。

6. マッピング線を設定します。
   設定するマッピング線の数は，呼び出す Java プログラムの引数と同数です。

7. 次のどちらかの方法で［カスタム］ダイアログを表示させます。
   - カスタムファンクションを右クリックして，［設定］を選択します。
   - カスタムファンクションをダブルクリックします。

8. ［変換ファンクション選択］ボタンをクリックします。
   ［変換ファンクション選択］ダイアログが表示されます。

9. カスタムファンクションから呼び出すメソッドを選択し，［OK］ボタンをクリックします。
   ［変換ファンクション選択］ダイアログが閉じ，［カスタム］ダイアログが表示されます。

10. ［引数名］に対応する入力値を［入力値］に指定します。
    指定した入力値が Java プログラムの引数に入力されます。

11. ［OK］ボタンをクリックします。

注意事項

- ［カスタム］ダイアログで変換ファンクションの設定をしたあとに，マッピングビューアでマッピング線を設定した場合，引数の入力値は自動で設定されません。再度［カスタム］ダイアログで［引数名］に対応する入力値を［入力値］に指定してください。
- カスタムファンクションは，［変換ファンクション選択］ダイアログで選択したときのメソッドの情報で呼び出します。［変換ファンクション選択］ダイアログでメソッドを選択したあとに，変換ファンクション定義ファイルの内容を変更しても，カスタムファンクションから呼び出すメソッドの情報には反映されません。変換ファンクション定義ファイルの変更をカスタムファンクションに反映させる場合は，変換ファンクションを選択し直してください。
- 次に示す設定内容が不一致の場合，ビジネスプロセスの検証時にエラーは発生しませんが，カスタムファンクションの実行時にエラーが発生します。そのため，設定内容を必ず一致させてください。
  - 変換ファンクション定義ファイルの定義内容

- カスタムファンクションの定義内容
- パッケージングした jar ファイルの内容

- 複数の変換ファンクション定義ファイルを使用したい場合は，［設定］ダイアログで変換ファンクション定義ファイルを切り替えてからカスタムファンクションを設定してください。変換ファンクション定義ファイルを切り替えても，設定済みのカスタムファンクションには影響しません。

7.5.24　変換先ノードやファンクションの入力値に XPath 式を指定する

変換先ノードやファンクションの入力値に XPath 式を指定します。

(1)　使用するファンクション

変換先ノードやファンクションの入力値に XPath 式を指定するには，XPath ファンクションを使用します。

(2)　設定手順

変換先ノードやファンクションの入力値に XPath 式を指定する手順を次に示します。

1. パレットから XPath ファンクション（xpath）を選択して，マッピングビューアに配置します。
2. 次のどちらかの方法で［XPath］ダイアログを表示させます。
   - XPath ファンクションを右クリックして，［設定］を選択します。
   - XPath ファンクションをダブルクリックします。

   ［XPath］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.42　XPath ダイアログ」を参照してください。
3. ［XPath］ダイアログで値を入力します。
   XPath ファンクションのファンクション名と XPath 式を入力します。
4. ［OK］ボタンをクリックします。

XPath ファンクションの使用方法については，「7.19　XPath ファンクションの使用方法」を参照してください。

(3)　指定できる XPath

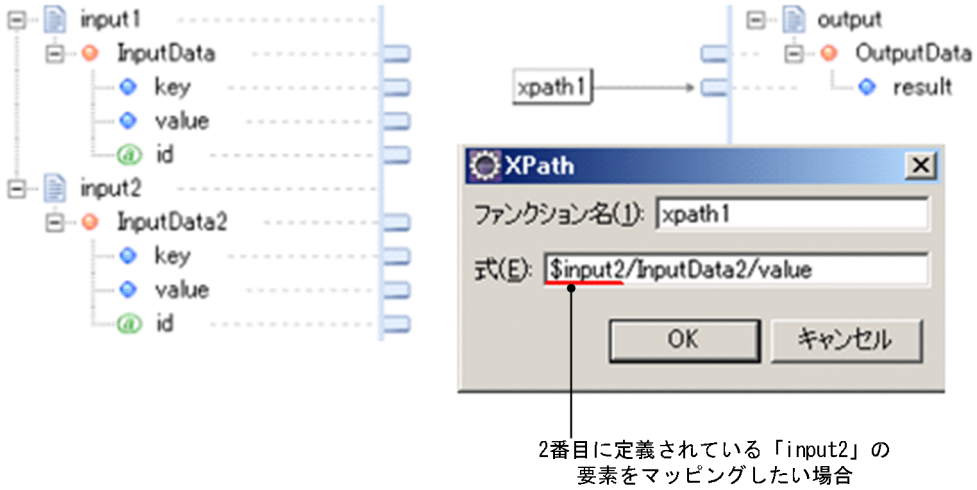
XPath の仕様の詳細については，マニュアル「XML Processor ユーザーズガイド」を参照してください。ここでは，XPath ファンクションで指定する文字列などについて説明します。

- エスケープが必要な文字は，データ変換定義の生成時に次に示す実体参照，または文字参照に置き換えます。
  - 左アングルブラケット（<）：<

- 右アングルブラケット（>）：>
- アンパサンド（&）：&
- ダブルクォーテーション（"）："
- アポストロフィ（'）：'

- 実体参照，および文字参照を含む XPath 式は使用できません。
- XPath の id 関数は使用できません。
- XSLT で定義されている関数や拡張関数は使用できません。
- N 対 1 変換（入力スキーマを複数指定した場合の変換）で，2 番目以降に定義されているスキーマ内のパスを XPath ファンクションで指定する場合，XPath 式は「$<スキーマの論理名(＝ BP の変数名)>/<要素パス>」としてください。なお，「$」を使用した変数の指定は，この指定以外はできません。
  指定例を次の図に示します。

図 7-25　N 対 1 変換でのパスの指定例

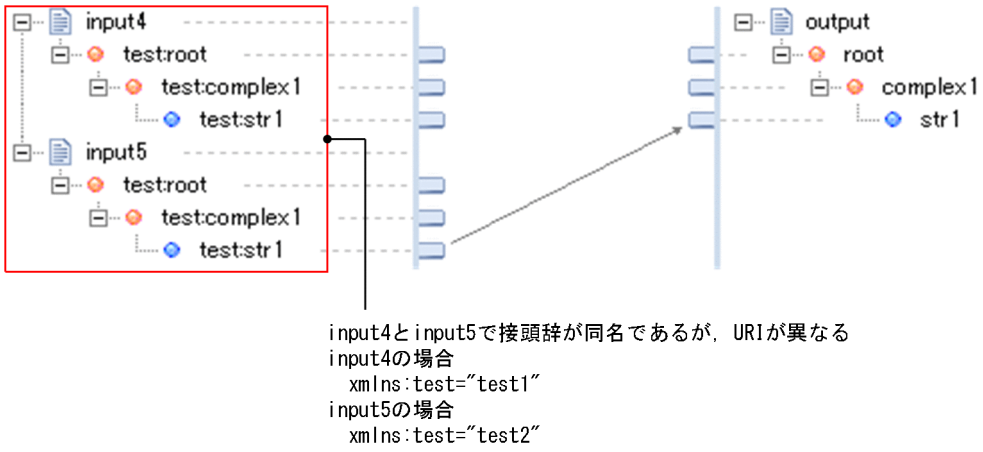


- N 対 1 変換時で，次のすべての条件に当てはまる場合は，XPath 関数の namespace-uri を使用して XPath 式を指定してください。

  1. 指定するパスに名前空間接頭辞が含まれる
  2. 1 の名前空間接頭辞がほかのスキーマですでに定義されていて，URI が異なる

  指定できないパスの例を次の図に示します。

図 7-26　N 対 1 変換時にパスが指定できないケース



上記の図の例では，XPath 関数の namespace-uri を使用することで，名前空間接頭辞を含まない XPath 式を指定できます。変数 input5 の str1 要素を指定する場合は，次に示す XPath 式となります。

```
$input5/*[local-name()='root' and namespace-uri()='test2']/…/*[local-name()='str1' and n
amespace-uri()='test2']
```

- デフォルト名前空間を含むパスを指定する場合は，XPath 関数の namespace-uri を使用して URI を指定する必要があります。
- 次に示す名前空間接頭辞は使用できません。
  - csc から始まる名前空間接頭辞
  - xsl
  - xsi

(4)　検証処理

XPath ファンクションで実施する検証について次に示します。

(a)　ダイアログおよびマッピング定義での検証

XPath ファンクションでは，ダイアログで［OK］ボタンをクリックしたとき，またはマッピング定義の検証を実行したときに，構文をチェックします。このとき，XPath 式に不正がある場合はエラーを表示します。

(b)　注意事項

XPath ファンクションの検証処理は，指定した XPath 式が XPath および XSLT の構文に沿っているかどうかを確認するものであるため，構文として正しい場合でも，パスの指定ミスや関数の使用誤りなどによって，意図した変換にならないケースがあります。そのため，XPath ファンクションを使用したマッピングが意図した変換になっていることを，csctransform コマンドでデータ変換を実行して確認してください。

csctransform コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csctransform（データ変換のテスト）」を参照してください。

7.6　くり返しの設定

くり返しファンクションを使用したマッピングでは，くり返しを合成したり，くり返し依存対象を変更したりできます。

7.6.1　［くり返し設定］ダイアログを使用してマッピングする

くり返しファンクションまたは選択ファンクション※から変換先ノードへのマッピングは，［くり返し設定］ダイアログを使用することもできます。

［くり返し設定］ダイアログを使用したマッピングの手順を次に示します。

1. パレットからくり返しファンクションまたは選択ファンクションを選択します。

2. マッピングビューアで適当な場所をクリックします。

   ファンクションがマッピングビューアに配置されます。配置されたファンクションは，ドラッグ＆ドロップで移動できます。

3. 変換先スキーマツリービューア上のマッピング先となる変換先ノードを右クリックして，［くり返し］を選択します。

   ［くり返し設定］ダイアログが表示されます。

4. ［ファンクション追加］ボタンをクリックします。

   ［ファンクション選択］ダイアログが表示されます。

5. マッピング元となるくり返しファンクションまたは選択ファンクションを指定して，［OK］ボタンをクリックします。

   ［ファンクション名］に指定したファンクション名が設定されます。

6. ［OK］ボタンをクリックします。

   マッピング線 ┈┈┈ （くり返し対応線）が設定されます。なお，マッピング線の色は変更できます。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

注※

くり返しファンクションに接続されている，または何も接続されていない選択ファンクションになります。くり返しファンクション以外のファンクションに接続されている場合，［くり返し設定］ダイアログは使用できません。

注意事項

変換元ノードにくり返しを設定した場合，その変換元ノードの子孫のノードもくり返しの対象となり，変換先ノードに出現します。

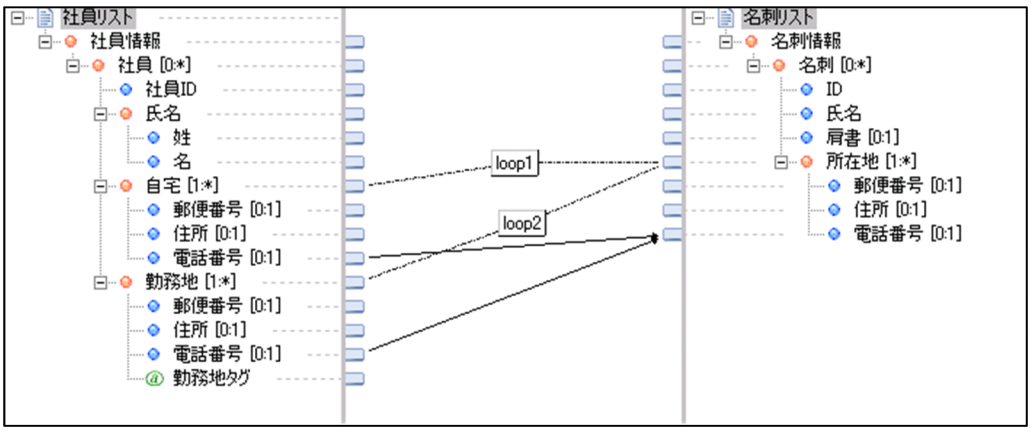
ただし，［マッピング元設定］ダイアログでマッピング元に絶対パスを指定した場合は，常に絶対パスで指定したノードの値が代入されます。［マッピング元設定］ダイアログでマッピング元に相対パスが表示されている場合には，くり返しを設定した変換元ノードの子孫のノードがくり返し対象となり，くり返しに対応したノードの値が代入されます。

変換元ノードのパスの表示については，「7.6.6　くり返し依存対象が設定された変換元ノードのパスの表示」を参照してください。

7.6.2　くり返しを合成する

複数の変換元ノードのくり返しを合成して，1 つの変換先ノードに対応づけることができます。くり返しを合成した場合の画面を次に示します。

図 7-27　くり返しを合成した場合の画面



くり返しを合成した場合，変換先ノードに出現させる変換元ノードの順序を［くり返し設定］ダイアログで設定できます。この場合，loop1 に対応した変換元ノード（自宅）と loop2 に対応した変換元ノード（勤務地）のどちらを先に変換先ノードに出現させるか決めることができます。なお，くり返しに対応した変換元ノードの子孫のノードもくり返しの対象となり，変換先ノードに出現します。

変換先ノードに代入する変換元ノードの順序を指定する手順を次に示します。

1. 変換先スキーマツリービューア上のマッピング先となる変換先ノードを右クリックして，［くり返し］を選択します。
   ［くり返し設定］ダイアログが表示されます。

2. ［くり返し］に表示されたファンクション名を，変換先ノードに出現させる順番に並べ替えます。

いちばん上に表示されたファンクション名に対応する変換元ノードから順に，変換先ノードに出現します。

- 順番を上にしたい場合
  上にしたいファンクション名を選択して［上へ］ボタンをクリックします。
- 順番を下にしたい場合
  下にしたいファンクション名を選択して［下へ］ボタンをクリックします。

3. ［OK］ボタンをクリックします。

7.6.3　くり返し依存対象の対応づけ

くり返しファンクションを使用して変換元ノードから変換先ノードへマッピングした場合，そのくり返しファンクションは，自動的に子孫ノードのマッピング元のくり返し対象として対応づけられます。また，くり返しファンクションやマッピング線を削除して，あるノードのくり返しを取り消したとします。その場合，その子孫ノードには，くり返しを取り消したノードの祖先ノードに設定されているくり返しファンクションが，自動的にマッピング元のくり返し対象として対応づけられます。これをくり返しの依存対象といいます。

くり返しの依存対象は次の場合に設定されます。

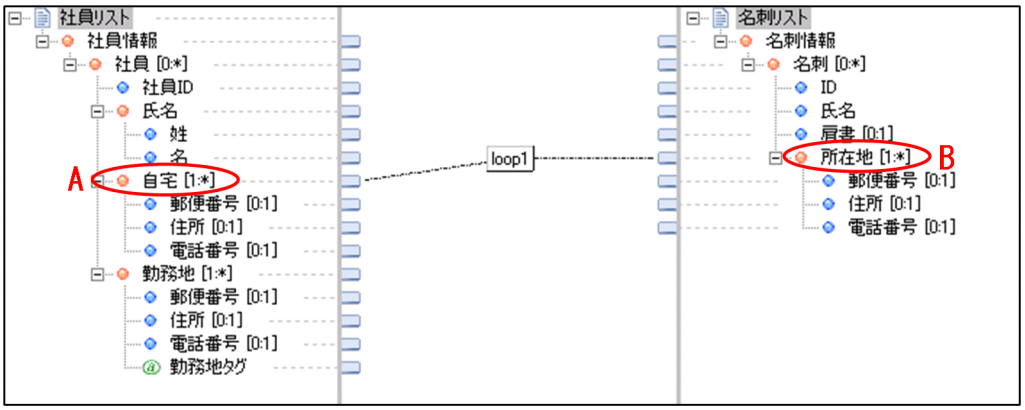
- あるノードにくり返しファンクションを使用してくり返しを設定したあと，その子孫ノードをマッピングした場合
  あるノードで使用したくり返しファンクションは，自動的にその子孫ノードのマッピング元のくり返し依存対象となります。
- あるノードをマッピングしたあと，その祖先ノードにくり返しファンクションを使用してくり返しを設定した場合
  祖先ノードで使用したくり返しファンクションは，その子孫ノードのマッピング元のくり返し依存対象となります。
- 祖先ノードにくり返しが設定されているノードのくり返しを取り消した場合
  祖先ノードに設定されているくり返しファンクションは，くり返しを取り消したノードの子孫ノードで，マッピング元のくり返し依存対象となります。

それぞれの場合のくり返し依存対象が対応づけられるタイミングについて説明します。

(1)　あるノードにくり返しを設定したあと，その子孫ノードをマッピングした場合

あるノードにくり返しを設定したあと，その子孫ノードをマッピングした場合の，くり返し依存対象が対応づけられるタイミングを次に示します。

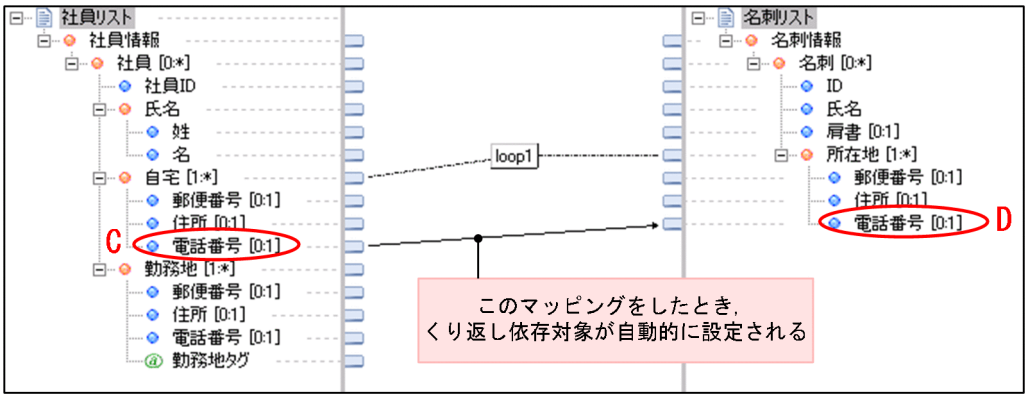
図 7-28　くり返しファンクションを使用したマッピング



まず，くり返しファンクションを使用して，変換元ノード A から変換先ノード B へマッピングします。

次に，変換元ノード A の子孫ノードであるノード C から，変換先ノード B の子孫ノードであるノード D へマッピングします。

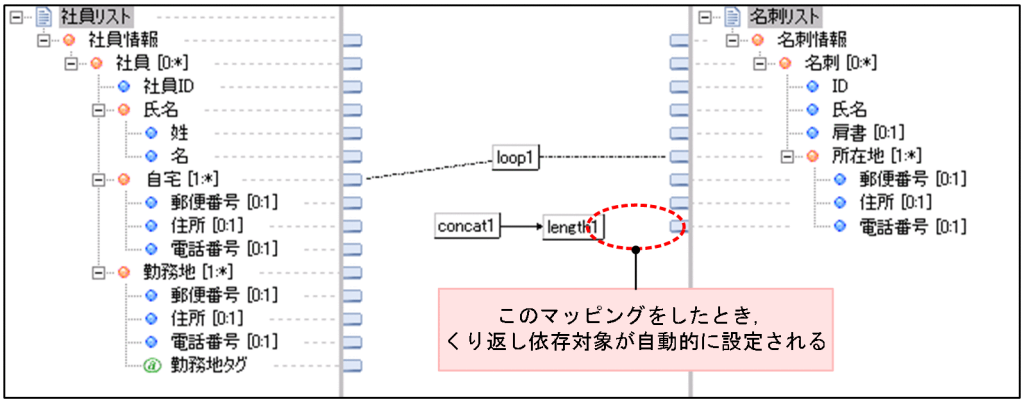
図 7-29　くり返し依存対象が対応づけられるタイミング（子孫ノードにファンクションがない場合）



ノード D に接続されたとき，子孫ノードに対してくり返し依存対象が自動的に設定されます。「図 7-29」の場合，子孫ノード D のマッピング元のくり返し依存対象に「loop1」が設定されます。

子孫の変換元ノードから変換先ノードへのマッピングで複数のファンクションを使用した場合は，変換先ノードに接続された時点でくり返し依存対象が自動的に設定されます。

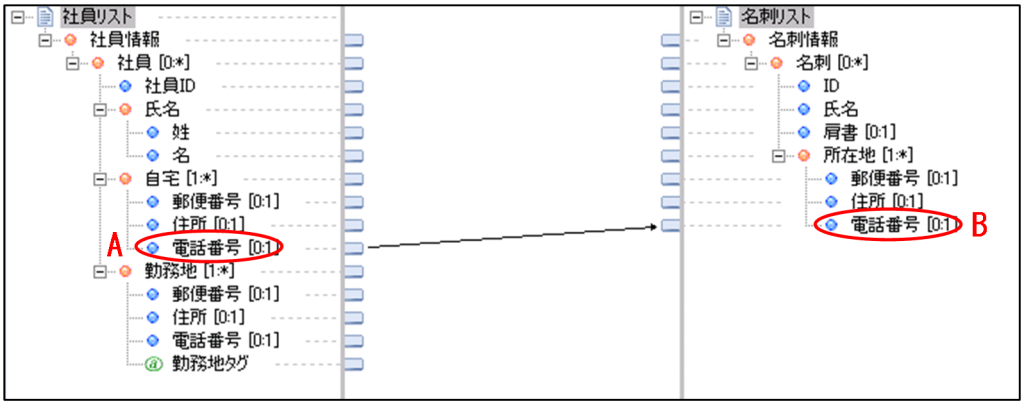
図 7-30　くり返し依存対象が対応づけられるタイミング（子孫のノードに複数のファンクションがある場合）



(2)　あるノードをマッピングさせたあと，その祖先ノードにくり返しを設定した場合

あるノードをマッピングさせたあと，その祖先ノードにくり返しを設定した場合の，くり返し依存対象が対応づけられるタイミングを次に示します。

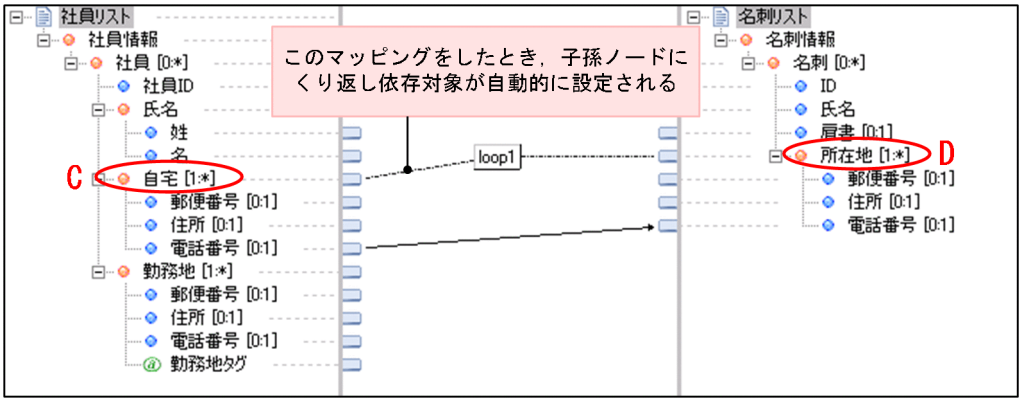
図 7-31　くり返しファンクションを使用しないマッピング



まず，ある変換元ノード A から変換先ノード B へマッピングします。

次に，変換元ノード A の祖先ノードであるノード C から，変換先ノード B の祖先ノードであるノード D へマッピングします。

図 7-32　くり返し依存対象が対応づけられるタイミング（祖先ノードのマッピングをあとにした場合）

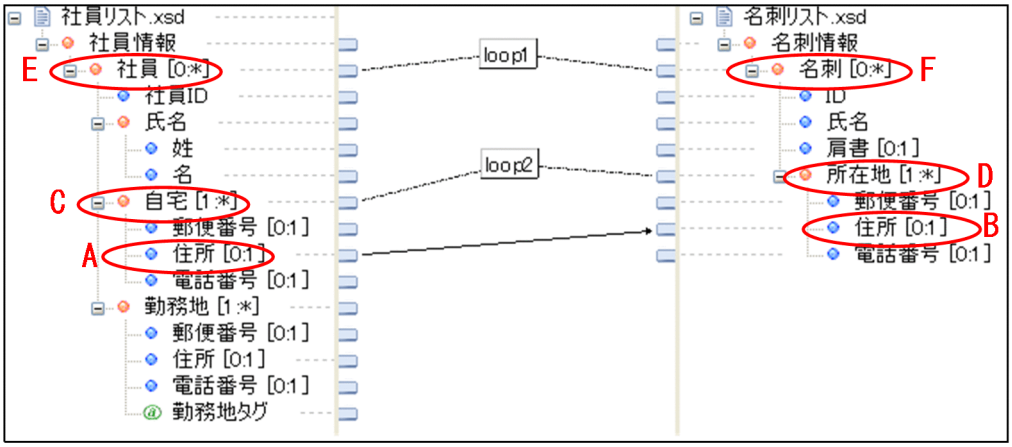


ノードＣからノードＤにマッピングしたとき，子孫ノードに対してくり返し依存対象が自動的に設定されます。「図 7-31」の場合，子孫ノードＢに対してくり返し依存対象「loop1」が設定されます。

(3)　祖先ノードにくり返しが設定されているノードのくり返しを取り消した場合

祖先ノードにくり返しが設定されているノードのくり返しを取り消した場合の，くり返し依存対象が対応づけられるタイミングを次に示します。

図 7-33　くり返しファンクションを使用したマッピング，およびくり返しファンクションを使用しないマッピング

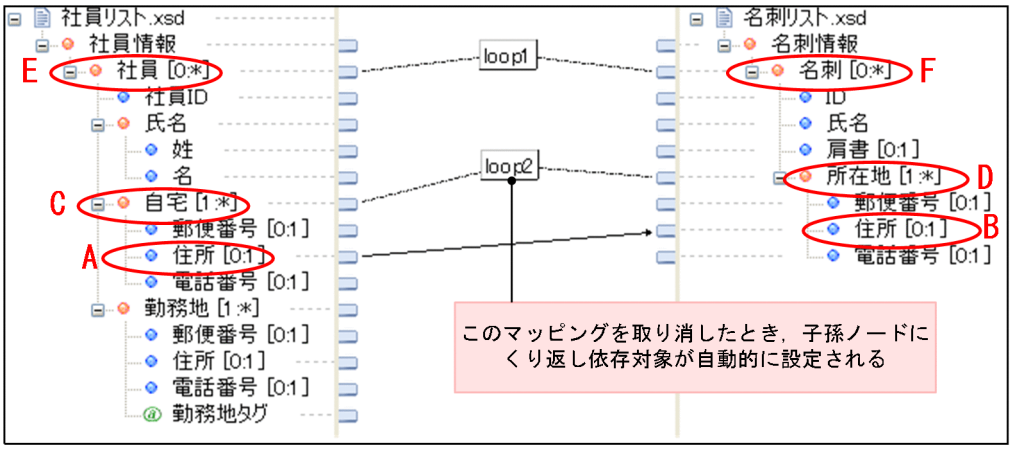


まず，ある変換元ノードＡから変換先ノードＢへマッピングします。

次に，くり返しファンクションを使用して，変換元ノードＡの祖先ノードであるノードＣから，変換先ノードＢの祖先ノードであるノードＤへマッピングします。

そのあと，くり返しファンクションを使用して，変換元ノードＡおよび変換元ノードＣの祖先ノードである変換元ノードＥから，変換先ノードＢおよび変換先ノードＤの祖先ノードである変換先ノードＦへマッピングします。

図 7-34　くり返し依存対象が対応づけられるタイミング（マッピングを取り消した場合）



ノードＣからノードＤへのマッピングを取り消したとき，マッピングを取り消したノードの子孫ノードに対して，くり返し依存対象が自動的に設定されます。取り消した場合，子孫ノードＢに対してくり返し依存対象「loop1」が設定されます。

7.6.4　くり返し依存対象を確認する

自動的に設定されたくり返し依存対象を確認する手順を次に示します。

1. くり返しを設定した変換先ノードの子孫ノードをスキーマツリービューア上で右クリックして，［マッピング元］を選択します。

   ［マッピング元設定］ダイアログが表示されます。



   ［パス/ファンクション名］には，くり返しファンクションが対応づけられた変換元ノードからの相対パスが表示されます。パス名の表示については，「7.6.6　くり返し依存対象が設定された変換元ノードのパスの表示」を参照してください。［依存対象］にはくり返し依存対象となるファンクション名が表示されます。

7.6.5　くり返し依存対象を変更する

複数の変換元ノードから 1 つの変換先ノードにマッピングしている場合，変換元ノードに対して自動的に設定されたくり返し依存対象を変更することができます。

くり返し依存対象を変更する手順を次に示します。

1. くり返しを設定した変換先ノードの子孫ノードをスキーマツリービューア上で右クリックして，［マッピング元］を選択します。
   ［マッピング元設定］ダイアログが表示されます。
2. ［マッピング元］の［依存対象］を変更します。
   変更した依存対象のパス名が絶対パスになります。パス名の表示については，「7.6.6　くり返し依存対象が設定された変換元ノードのパスの表示」を参照してください。
3. ［OK］ボタンをクリックします。

7.6.6　くり返し依存対象が設定された変換元ノードのパスの表示

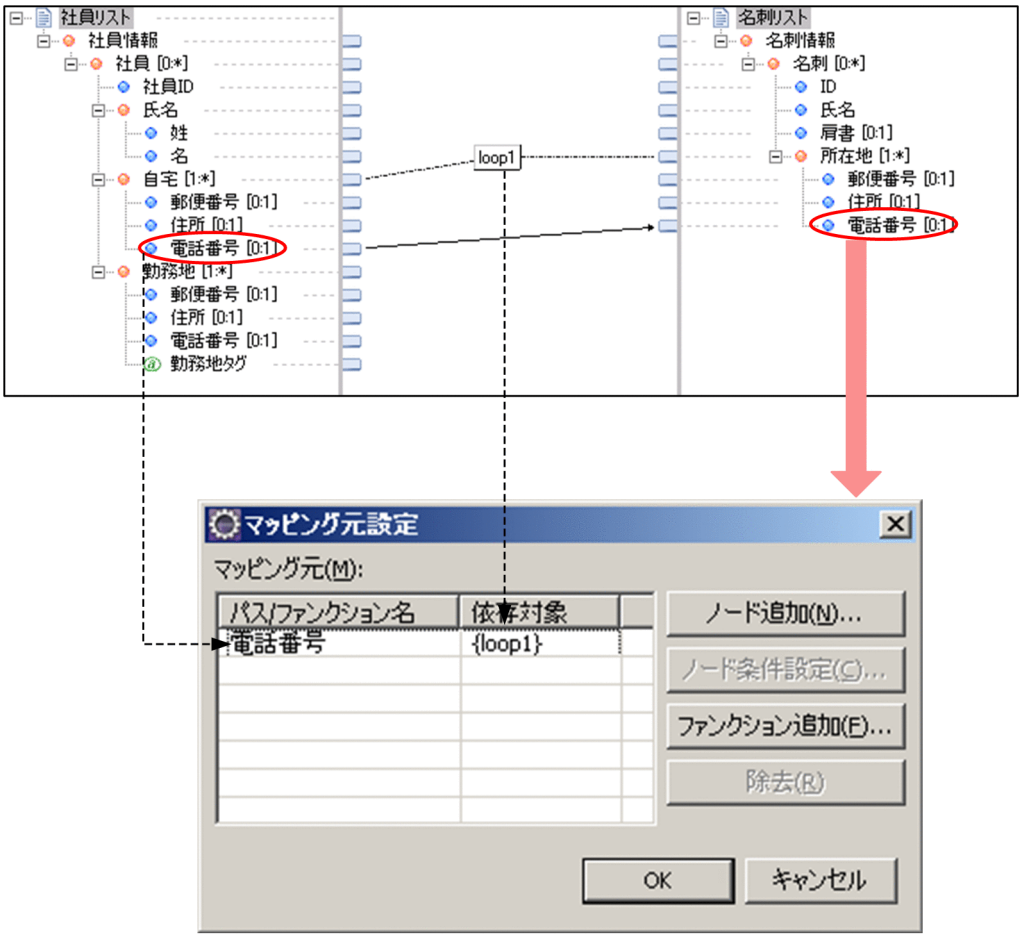
［マッピング元設定］ダイアログの［パス/ファンクション名］には，相対パスまたは絶対パスが表示されます。どのような場合に相対パスまたは絶対パスが表示されるかについて説明します。

(1)　相対パス

くり返しファンクションが対応づけられた変換元ノードの子孫の変換元ノードに対してくり返し依存対象が設定された場合，くり返しファンクションが対応づけられた変換元ノードからの相対パスが表示されます。

具体的な例を次に示します。

図 7-35　相対パスの例



自宅ノードにくり返しファンクション「loop1」を対応づけると，自宅ノードの子孫の電話番号ノードにくり返し依存対象「loop1」が設定されます。

このとき，マッピング後の電話番号ノードから［マッピング元設定］ダイアログを表示させると，［パス/ファンクション名］には，自宅ノードからの相対パスが表示されます。

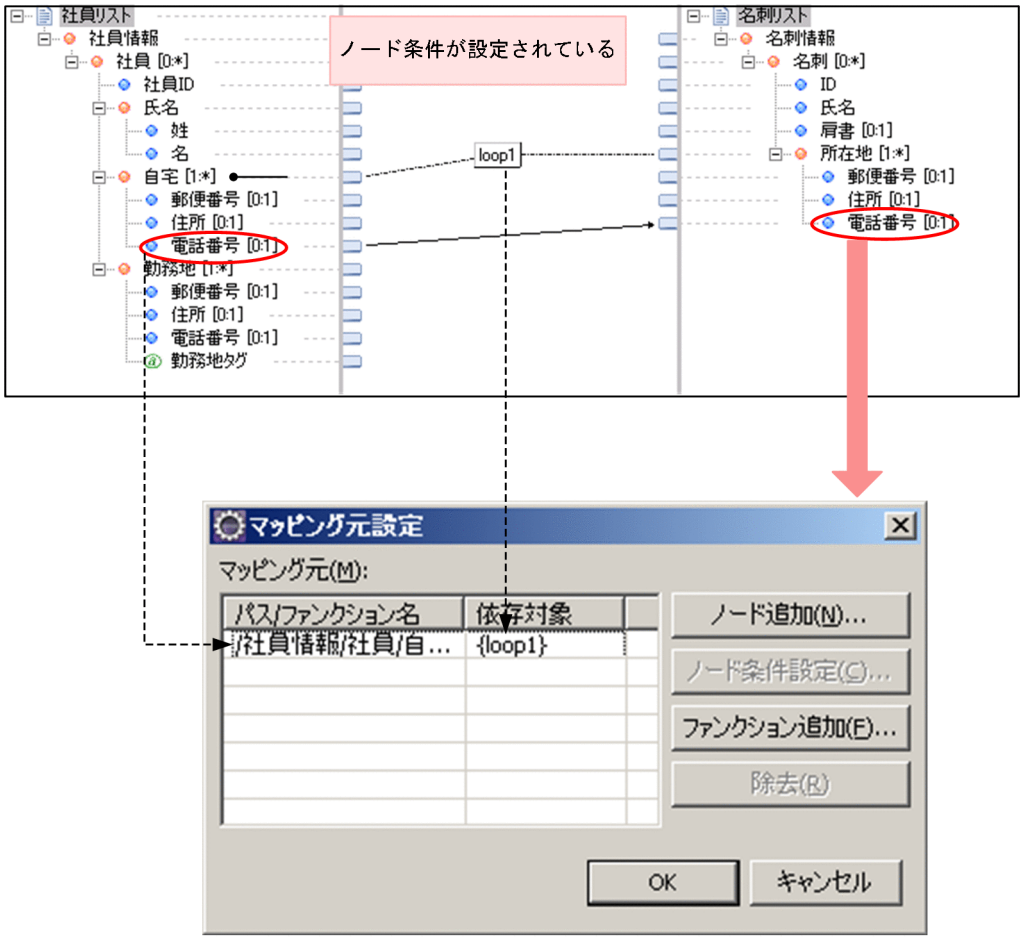
(2)　絶対パス

次の場合，絶対パスが表示されます。

- くり返し依存対象のくり返しファンクションが対応づけられた変換元ノードまたはその変換元ノードの祖先の変換元ノードに，ノード条件が設定されている場合

  具体的な例を次に示します。

図 7-36　絶対パスの例 1



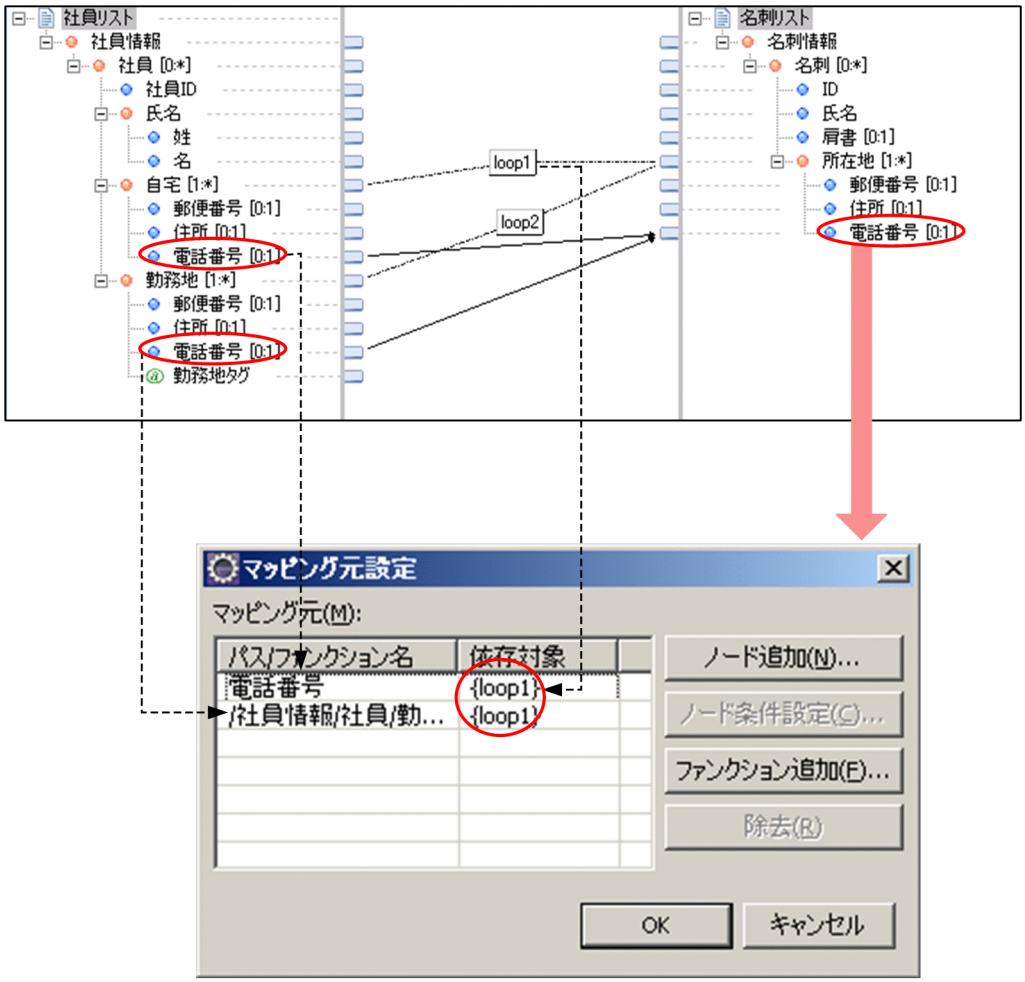
自宅ノードにくり返しファンクション「loop1」を対応づけると，自宅ノードの子孫の電話番号ノードにくり返し依存対象「loop1」が設定されます。なお，自宅ノードにはノード条件が設定されています。

このとき，マッピング後の電話番号ノードから［マッピング元設定］ダイアログを表示させると，［パス/ファンクション名］には，絶対パスが表示されます。

- くり返し依存対象が設定された変換元ノードが，くり返し依存対象のくり返しファンクションが対応づけられた変換元ノードの子孫のノードでない場合

  具体的な例を次に示します。

図 7-37　絶対パスの例 2（依存対象の変更前）

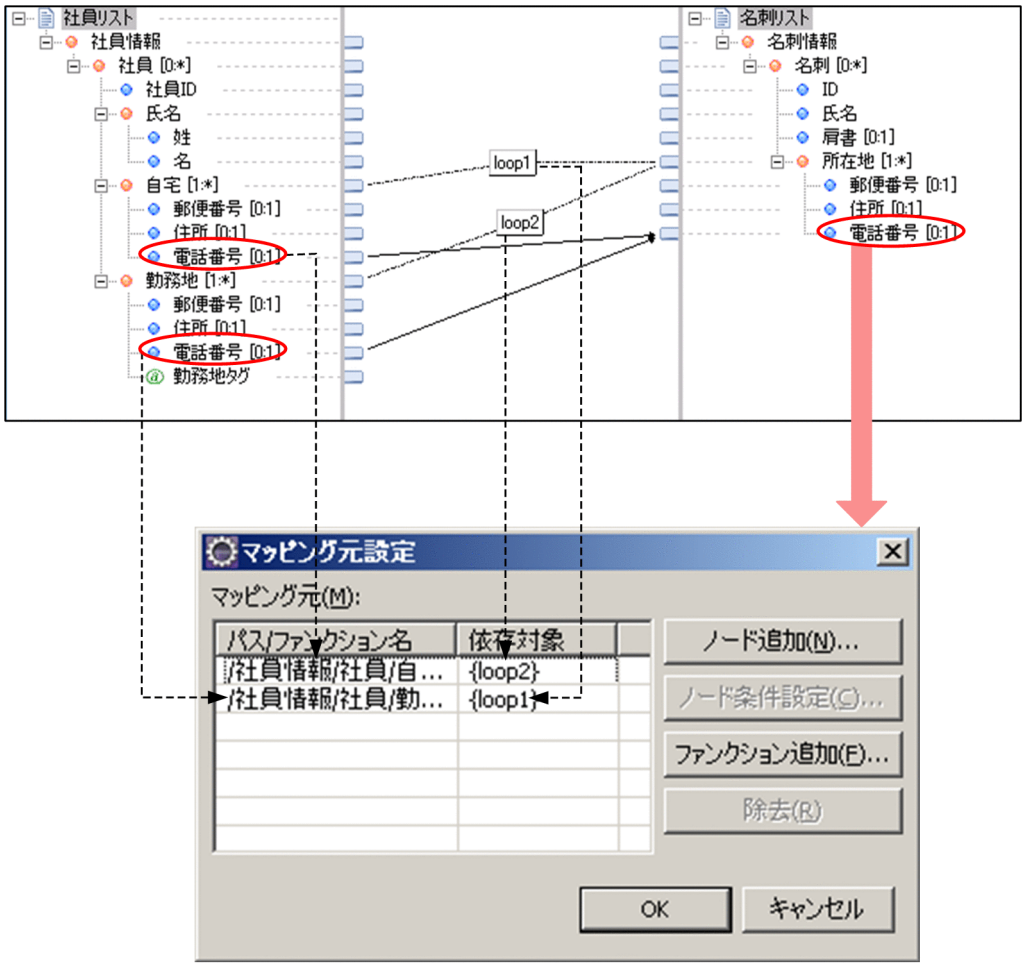


まず自宅ノードにくり返しファンクション「loop1」を対応づけ，次に勤務地ノードにくり返しファンクション「loop2」を対応づけます。これによって，自宅ノードの子孫の電話番号ノードおよび勤務地ノードの子孫の電話番号ノードには，最初に対応づけたくり返しファンクション「loop1」がくり返し依存対象に設定されます。

このとき，マッピング後の電話番号ノードから［マッピング元設定］ダイアログを表示させると，［パス/ファンクション名］には，次のように表示されます。

- 自宅ノード（「loop1」が対応づけられている）の子孫の電話番号ノード
  自宅ノードからの相対パスが表示されます。
- 勤務地ノード（「loop2」が対応づけられている）の子孫の電話番号ノード
  絶対パスが表示されます。

図 7-38　絶対パスの例 2（依存対象の変更後）



自宅ノードの子孫の電話番号ノードの依存対象を，勤務地ノードのくり返しファンクションである「loop2」に設定します。

このとき，マッピング後の電話番号ノードから［マッピング元設定］ダイアログを表示させると，［パス/ファンクション名］の表示が，次のように変更されます。

- 自宅ノード（「loop1」が対応づけられている）の子孫の電話番号ノード
  自宅ノードからの相対パスが絶対パスになります。

7.6.7　連携パスを設定して各要素のくり返し処理を連動させる

くり返しに依存しない要素をくり返しに依存する要素に連動させて変換する場合，くり返しファンクションに連携パスを設定する必要があります。

ここでは，くり返しファンクションに連携パスを設定する方法，連携パスの表示，連携パスを使用した変換例を説明します。

(1)　連携パスの設定

くり返しファンクションに連携パスを設定することで，変換元の各要素のくり返し処理を連動できます。

なお，くり返し連携の対象となるのは，次の条件に当てはまる場合です。

- マッピング線の変換先ノードが，該当するくり返しファンクションに依存している。
- マッピング線の変換元ノードのパスに，くり返しファンクションに指定した連携パスが含まれている。

注意事項

次の場合に当てはまるパスは，連携パスに追加できません。指定するとエラーメッセージが表示されます。

- 同じくり返しファンクションの基準パスの祖先ノードまたは子孫ノードであるパスを指定した場合
- 同じくり返しファンクションの連携パスの祖先ノードまたは子孫ノードを複数指定した場合

また，次の 1～3 の条件がすべて重なる連携パスは追加できません。指定すると検証時にエラーとなります。

1. 複数の繰り返しファンクションを使用し，多重ループとなっている。
2. 1.の繰り返しファンクションが兄弟の要素を基準としている。
3. 内側の繰り返しファンクションから外側の繰り返しファンクションの基準パスを連携パスに指定している。

連携パスは，次の手順で設定します。

1. 次のどちらかの方法で［くり返し］ダイアログを表示させます。
   - くり返しファンクションを右クリックして，［設定］を選択します。
   - くり返しファンクションをダブルクリックします。

   ［くり返し］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.6.7　くり返し設定ダイアログ」を参照してください。
2. ［連携パス］の［ノード追加］ボタンをクリックします。
   ［ノード選択ダイアログ］が表示されます。
3. 連携する要素を選択し，［OK］ボタンをクリックします。
   選択した要素のパスが［連携パス］に表示されます。複数の連携パスを指定する場合は，手順 2.～3.を繰り返してください。
4. ［くり返し］ダイアログの［OK］ボタンをクリックします。
   連携パスが設定されます。

(2)　連携パスの表示

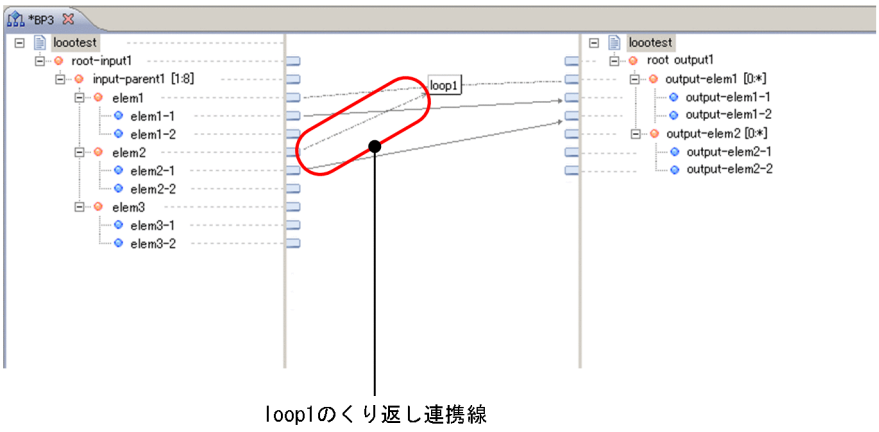
設定した連携パスは，マッピングビューアおよびプロパティービューに表示されます。

マッピングビューアでは，連携パスからくり返しファンクションに対して，マッピング線 ⇢ （くり返し連携線）が表示されます。マッピング線の色は，くり返し対応線と同じです。マッピング線の色を変更する方法については，マニュアル「サービスプラットフォーム リファレンス」の「1.13.5　マッピング線の色を変更する」を参照してください。

また，くり返し連携線をマッピングビューア上で削除し，マッピング（連携）を取り消すことができます。マッピングを取り消す方法については，「7.4.5　マッピングを取り消す」を参照してください。

くり返し連携線の表示例を次の図に示します。

図 7-39　くり返し連携線の表示例

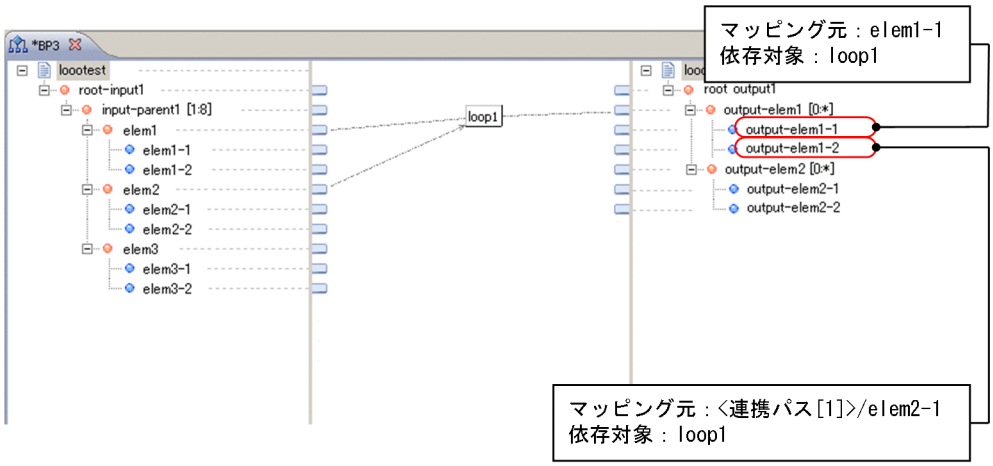


(3)　連携パスを使用した変換例

連携パスを使用したくり返しファンクションの変換例を次に示します。

- くり返しファンクションと連携している値をマッピングする場合

図 7-40　連携パスを使用した変換例 1



●入力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-input1>
    <input-parent1>
        <elem1>
            <elem1-1>あいうえお</elem1-1>
            <elem1-2>かきくけこ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>さしすせそ</elem2-1>
            <elem2-2>たちつてと</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>なにぬねの</elem3-1>
            <elem3-2>はひふへほ</elem3-2>
        </elem3>
    </input-parent1>
    <input-parent1>
        <elem1>
            <elem1-1>アイウエオ</elem1-1>
            <elem1-2>カキクケコ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>サシスセソ</elem2-1>
            <elem2-2>タチツテト</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>ナニヌネノ</elem3-1>
            <elem3-2>ハヒフヘホ</elem3-2>
        </elem3>
    </input-parent1>
</root-input1>
```

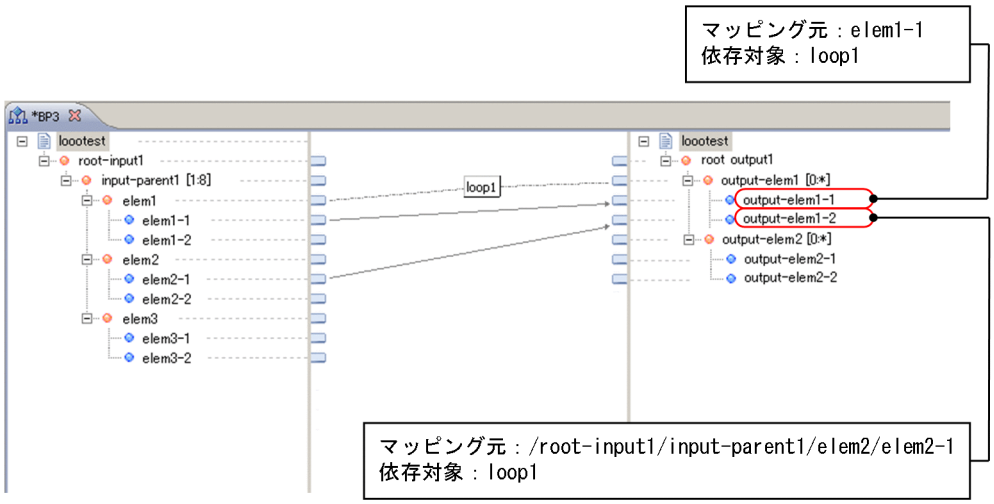
●出力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-output1>
    <output-elem1>
        <output-elem1-1>あいうえお</output-
elem1-1>
        <output-elem1-2>さしすせそ</output-
elem1-2>
    </output-elem1>
    <output-elem1>
        <output-elem1-1>アイウエオ</output-
elem1-1>
        <output-elem1-2>サシスセソ</output-
elem1-2>
    </output-elem1>
</root-output1>
```

loop1と連動したpositionの
elem2-1を取得する

- くり返しファンクションと連携していない値をマッピングする場合

図 7-41　連携パスを使用した変換例 2



●入力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-input1>
    <input-parent1>
        <elem1>
            <elem1-1>あいうえお</elem1-1>
            <elem1-2>かきくけこ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>さしすせそ</elem2-1>
            <elem2-2>たちつてと</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>なにぬねの</elem3-1>
            <elem3-2>はひふへほ</elem3-2>
        </elem3>
    </input-parent1>
    <input-parent1>
        <elem1>
            <elem1-1>アイウエオ</elem1-1>
            <elem1-2>カキクケコ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>サシスセソ</elem2-1>
            <elem2-2>タチツテト</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>ナニヌネノ</elem3-1>
            <elem3-2>ハヒフヘホ</elem3-2>
        </elem3>
    </input-parent1>
</root-input1>
```

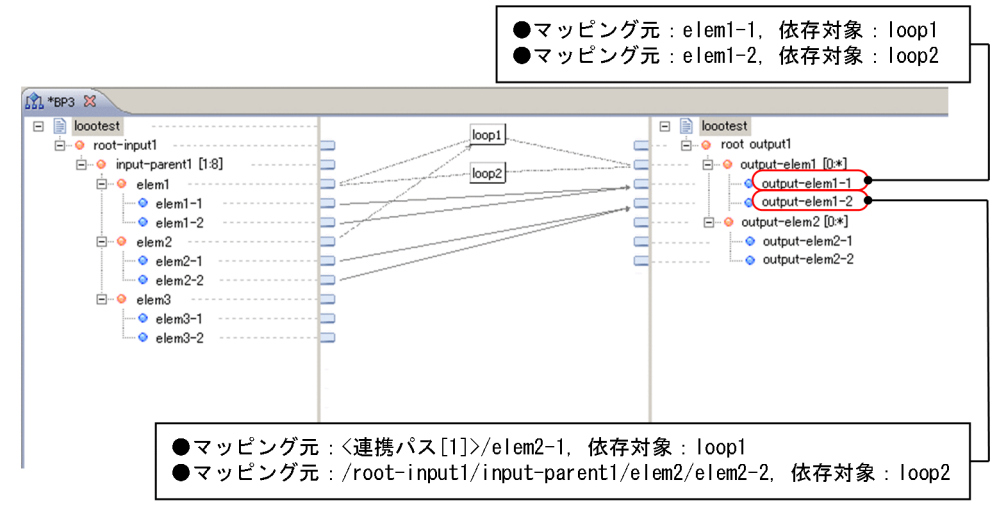
●出力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-output1>
    <output-elem1>
        <output-elem1-1>あいうえお</output-
elem1-1>
        <output-elem1-2>さしすせそ</output-
elem1-2>
    </output-elem1>
    <output-elem1>
        <output-elem1-1>アイウエオ</output-
elem1-1>
        <output-elem1-2>さしすせそ</output-
elem1-2>
    </output-elem1>
</root-output1>
```

連携しないため，常に1つ目のelem2-1を取得

- 複数のくり返しファンクションと連携している値を参照する場合

図 7-42　連携パスを使用した変換例 3



●入力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-input1>
    <input-parent1>
        <elem1>
            <elem1-1>あいうえお</elem1-1>
            <elem1-2>かきくけこ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>さしすせそ</elem2-1>
            <elem2-2>たちつてと</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>なにぬねの</elem3-1>
            <elem3-2>はひふへほ</elem3-2>
        </elem3>
    </input-parent1>
    <input-parent1>
        <elem1>
            <elem1-1>アイウエオ</elem1-1>
            <elem1-2>カキクケコ</elem1-2>
        </elem1>
        <elem2>
            <elem2-1>サシスセソ</elem2-1>
            <elem2-2>タチツテト</elem2-2>
        </elem2>
        <elem3>
            <elem3-1>ナニヌネノ</elem3-1>
            <elem3-2>ハヒフヘホ</elem3-2>
        </elem3>
    </input-parent1>
</root-input1>
```

●出力データ

```
<?xml version="1.0" encoding="UTF-8"?>
<root-output1>
    <output-elem1>
        <output-elem1-1>あいうえお</output-
elem1-1>
        <output-elem1-2>さしすせそ</output-
elem1-2>
    </output-elem1>
    <output-elem1>
        <output-elem1-1>アイウエオ</output-
elem1-1>
        <output-elem1-2>サシスセソ</output-
elem1-2>
    </output-elem1>
    <output-elem1>
        <output-elem1-1>かきくけこ</output-
elem1-1>
        <output-elem1-2>たちつてと</output-
elem1-2>
    </output-elem1>
    <output-elem1>
        <output-elem1-1>カキクケコ</output-
elem1-1>
        <output-elem1-2>たちつてと</output-
elem1-2>
    </output-elem1>
</root-output1>
```

loop1と連携したpositionの
elem2-1を取得

loop2と連携しないため，
常に1つ目のelem2-2を取得

7.7 ノード条件を設定する

マッピング元の変換元ノードに対して条件を設定して，その条件を満たす場合だけマッピングします。変換元ノードに条件を設定することを**ノード条件の設定**といいます。ノード条件の設定は，マッピング先によって設定を開始する方法が次のように異なります。

- マッピング先が変換先ノードの場合

  ［マッピング元設定］ダイアログからノード条件の設定を開始します。［マッピング元設定］ダイアログは，変換先ノードを右クリックして，［マッピング元］を選択すると表示されます。

- マッピング先がファンクションの場合

  各ファンクションを設定するダイアログからノード条件の設定を開始します。各ファンクションを設定するダイアログは，次のどちらかの方法で表示されます。

  - 各ファンクションを右クリックして，［設定］を選択します。
  - 各ファンクションをダブルクリックします。

ノード条件を設定する手順を次に示します。

1. **［マッピング元設定］ダイアログまたは各ファンクションを設定するダイアログで，ノード条件を設定する変換元ノードを選択します。**

2. **［ノード条件設定］ボタンをクリックします。**

   ［ノード条件設定］ダイアログが表示されます。

   ［ノード条件設定］ダイアログには，［マッピング元設定］ダイアログで選択した変換元ノードとその祖先のノードが表示されます。ノード条件は，［ノード条件設定］ダイアログで表示されたすべてのノードに設定できます。

3. **ノード条件を設定するノードを選択して，［条件設定］ボタンをクリックします。**

   ［条件設定］ダイアログが表示されます。

4. **条件を設定します。**

   ［条件］で設定する条件を次のラジオボタンから選択し，条件の詳細を設定します。

   - ［boolean 値を返すノードまたはファンクション］ラジオボタン
   - ［条件式］ラジオボタン
   - ［二つの条件の論理積・論理和］ラジオボタン

   設定した条件は，［生成する条件］に表示されます。

   なお，［条件］で変換元ノードまたはファンクションを指定する場合，次の変換元ノードまたはファンクションを指定してください。

   - 変換元ノード

     マッピング対象※であり，かつ単純内容または属性のノード

   - ファンクション

出力側にマッピング線が設定されていない制御系以外のファンクション

注※

マッピング対象については，「7.11.1　マッピング対象とマッピング対象外」を参照してください。

5. [生成する条件] に表示された条件と反対の内容をノード条件としたい場合，[条件の否定をとる] チェックボックスにチェックします。

6. [OK] ボタンをクリックします。

[ノード条件設定] ダイアログの [条件] に，[条件設定] ダイアログで設定した条件が表示されます。

7. ほかのノードのノード条件を設定する場合，手順 3.～手順 6.を繰り返します。

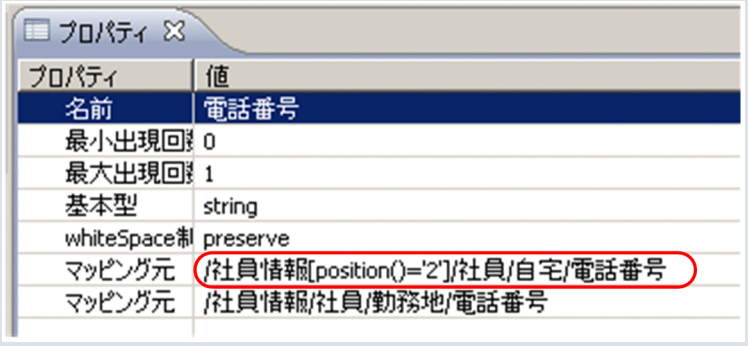
8. [ノード条件設定] ダイアログで，[OK] ボタンをクリックします。

ノード条件が設定されます。

[条件設定] ダイアログで設定した条件中に，マッピング元以外の変換元ノードまたはファンクションを指定した場合，その条件中の変換元ノードまたはファンクションとマッピング先の間にマッピング線 ┈┈→ （条件線）が設定されます。

ポイント

条件を設定した変換先ノードを選択すると，プロパティ領域の「マッピング元」に，設定したノード条件が表示されます。ノード条件は，条件を設定したノード名の後ろに角括弧（[　]）で囲まれて表示されます。なお，条件として指定したノード（[条件設定] ダイアログの [条件] で指定）が，条件を設定したノード（[ノード条件設定] ダイアログで指定）の子孫ノードの場合，条件を設定したノードからの相対パスが表示されます。それ以外は絶対パスが表示されます。



7.8　マッピング定義をコピーする

データ変換定義画面でマッピングした内容は，マッピング定義として保存できます。保存したマッピング定義は，別のデータ変換定義を作成する際にコピーして利用できます。

マッピング定義のコピーを次の図に示します。

図 7-43　マッピング定義のコピー



マッピング定義は，**マッピング情報**ごとにコピーされます。マッピング情報とは，1 つの変換先ノードと，そのノードからマッピング線をたどって取得できる変換元ノード，ファンクション，およびマッピング線のことです。

コピー元のマッピング定義ファイルを作成するときに使用したスキーマの種別（XSD ファイルまたは FDX ファイル）と，コピー先のスキーマの種別が異なる場合でも，マッピング定義をコピーできます。

マッピング定義をコピーする方法には，構造一致複製マッピングと要素名一致複製マッピングの 2 種類があります。構造一致複製マッピングは，コピー元とコピー先の要素名およびスキーマの構造が一致している場合に使用できる方法です。このため，マッピング定義のコピー先となるノードに，すでにマッピング線が接続されている場合，マッピング定義はコピーできません。要素名一致複製マッピングは，コピー元とコピー先の要素名が類似している場合に使用できる方法です。このため，構造一致複製マッピングとは異なり，スキーマの構造が異なっていてもコピーできます。それぞれのコピー方法については，「7.8.4 マッピング定義のコピー」で説明します。

7.8.1　マッピング定義をコピーする流れ

マッピング定義をコピーする流れを次に示します。

1. データ変換定義画面でマッピングした内容をマッピング定義として保存します。
   マッピング定義を保存する方法については，「7.8.2　マッピング定義の保存」を参照してください。

2. 保存したマッピング定義をコピー元として登録します。
   マッピング定義を登録する方法については，「7.8.3　マッピング定義の登録」を参照してください。

3. 変換元スキーマおよび変換先スキーマを指定して，コピー先となるデータ変換定義画面を表示します。
   データ変換定義画面を表示する方法については，「7.3.1　データ変換の定義手順」を参照してください。なお，データ変換定義画面を表示したら，マッピングはしないでください。

4. マッピング定義をコピーします。
   コピー元のマッピング定義のスキーマ情報とコピー先のスキーマ情報を比較し，類似した要素があった場合，マッピング定義が自動でコピーされます。マッピング定義がコピーされる条件については，「7.8.5　類似度の判定方法」を参照してください。
   マッピング定義をコピーする方法については，「7.8.4　マッピング定義のコピー」を参照してください。

7.8.2　マッピング定義の保存

マッピング定義ファイルを保存する方法を次に示します。

1. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［マッピング定義保存］を選択します。
   ［マッピング定義保存］ダイアログが表示されます。
   なお，［マッピング定義保存］ダイアログに最初に表示されるディレクトリは，［設定］ダイアログで任意の場所に設定できます。詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.6.18 設定ダイアログ（データ変換の設定）」を参照してください。

2. マッピング定義を保存するファイル（拡張子：.mdo）を指定します。

3. ［OK］ボタンをクリックします。
   マッピング定義が保存されます。

7.8.3　マッピング定義の登録

マッピング定義を登録する方法を次に示します。登録できるマッピング定義は，1 つの Eclipse に対して 1 つだけです。登録したマッピング定義は，すべてのデータ変換定義画面で共有できます。なお，Eclipse を終了すると，登録したマッピング定義は破棄されます。

1. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［マッピング定義登録］を選択します。

   ［マッピング定義登録］ダイアログが表示されます。

2. ［選択］ボタンをクリックし，コピー元として登録するマッピング定義ファイル（拡張子：.mdo）を指定します。

   なお，［選択］ボタンをクリックしたときに表示されるディレクトリは，［設定］ダイアログで任意の場所に設定できます。詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.6.18　設定ダイアログ（データ変換の設定)」を参照してください。

3. ［OK］ボタンをクリックします。

   マッピング定義が登録されます。

7.8.4　マッピング定義のコピー

マッピング定義をコピーする方法を次に示します。

> **参考**
>
> マッピング定義をコピーした場合，コピー先のファンクションには，コピー元のファンクションと同じ名称が設定されます。同名のファンクションがコピー先のマッピング定義エディタにすでに存在する場合は，ファンクション名の末尾に連番を追加した名称が設定されます。

(1)　構造一致複製マッピング

構造一致複製マッピングは，コピー先とコピー元の要素名だけでなく，スキーマの構造が一致している場合に使用できるコピー方法です。構造一致複製マッピングでマッピング定義をコピーする方法を次に示します。

1. データ変換定義画面の変換先スキーマツリービューアで，複合内容要素を右クリックして，［複製マッピング］－［構造一致］を選択します。

   ［複製マッピング元設定］ダイアログが表示されます。

2. ［ノード追加］ボタンをクリックします。

   ［ノード選択ダイアログ］が表示されます。

3. コピーする要素を選択し，［OK］ボタンをクリックします。

   選択した要素のパスが［パス］に追加されます。なお，変換元に複数のスキーマが存在する場合は，コピーする要素を複数選択してください。

4. ［OK］ボタンをクリックします。

   マッピング定義がコピーされます。

(2)　要素名一致複製マッピング

要素名一致複製マッピングは，コピー先とコピー元の要素名が一致している場合に使用できるコピー方法です。要素名一致複製マッピングでマッピング定義をコピーする方法を次に示します。

1. **データ変換定義画面の変換先スキーマツリービューアで，複合内容要素を右クリックして，［複製マッピング］－［要素名一致］を選択します。**
   ［複製マッピング範囲設定］ダイアログが表示されます。

2. **コピー元およびコピー先情報を設定します。**
   コピー元の情報は［複製マッピング元］に，コピー先の情報は［複製マッピング先］に設定します。
   各項目に設定するノードは，項目の横にある［ノード選択］ボタンをクリックして，選択してください。選択したノードが各項目に表示されます。

3. **［OK］ボタンをクリックします。**
   マッピング定義がコピーされます。

注意事項

次の場合は，エラーメッセージが表示され，マッピング定義はコピーされません。

- コピー元のマッピング定義ファイル（拡張子：.mdo）が登録されていない場合
- コピー元のマッピング定義ファイル（拡張子：.mdo）の読み込みに失敗した場合
- 登録してあるコピー元のマッピング定義ファイル（拡張子：.mdo）の内容が空の場合

7.8.5　類似度の判定方法

マッピング定義をコピーすると，コピー元となるマッピング定義のスキーマの要素と，コピー先のスキーマの要素が比較され，類似した要素があった場合にコピーされます。類似した要素が複数あるときは，優先度の高い順にコピーされます。マッピングがコピーされる優先度を次に説明します。

注意事項

次の条件に当てはまるファンクションについては，優先度に関係なく，コピーされません。

- ソート条件が設定されているくり返しファンクションのうち，ソート条件に対応する変換元ノードがないもの
- 連携パスが設定されているくり返しファンクションのうち，連携パスに対応する変換元ノードがないもの
- 条件が設定されている選択ファンクションのうち，条件に対応する変換元ノードがないもの

(1) 構造一致複製マッピング

構造一致複製マッピングでマッピング定義をコピーした場合に，マッピングがコピーされる優先度を次の表に示します。

表 7-5 マッピング定義がコピーされる優先度（構造一致複製マッピング）

優先度	スキーマ論理名	親パス	要素名
1	○	○	○
2	×	○	○

（凡例）

○：一致します。

×：一致しません。

(2) 要素名一致複製マッピング

要素名一致複製マッピングでマッピング定義をコピーした場合に，マッピングがコピーされる優先度を次の表に示します。

表 7-6 マッピング定義がコピーされる優先度（要素名一致複製マッピング）

優先度	要素		パス	
	名前空間	要素名	名前空間	要素名
1	×	×	○	○
2	×	×	×	○
3	○	○	×	△
4	×	○	×	△
5	×	△	×	△

（凡例）

○：一致します。

△：一部一致します。

×：一致しません。

なお，要素名一致複製マッピングでは，コピーした要素の依存関係に矛盾が生じないようにする必要があります。依存関係が設定されている変換先ノードをコピーする場合は，次の条件に合わせてコピーします。

- 依存対象となる変換先ノードがコピーされている場合は，依存対象の変換先ノードの子孫に含まれるようにコピーします。
- 依存対象となる変換先ノードをコピーできなかった場合は，依存対象となる変換先ノードと類似度が高い変換先ノードを検索して，そのノードの子孫に含まれるようにコピーします。類似度の高い変換先ノードが見つからなかった場合は，依存関係なしとしてコピーします。

また，次に示す値にノード条件が設定されている場合は，条件が設定されている祖先ノードが，変換先ノードおよびファンクションの接続元となる変換元ノードの祖先ノードになるようにコピーします。

- 変換先ノードの入力値
- ファンクションの入力値
- くり返しファンクションの連携パス

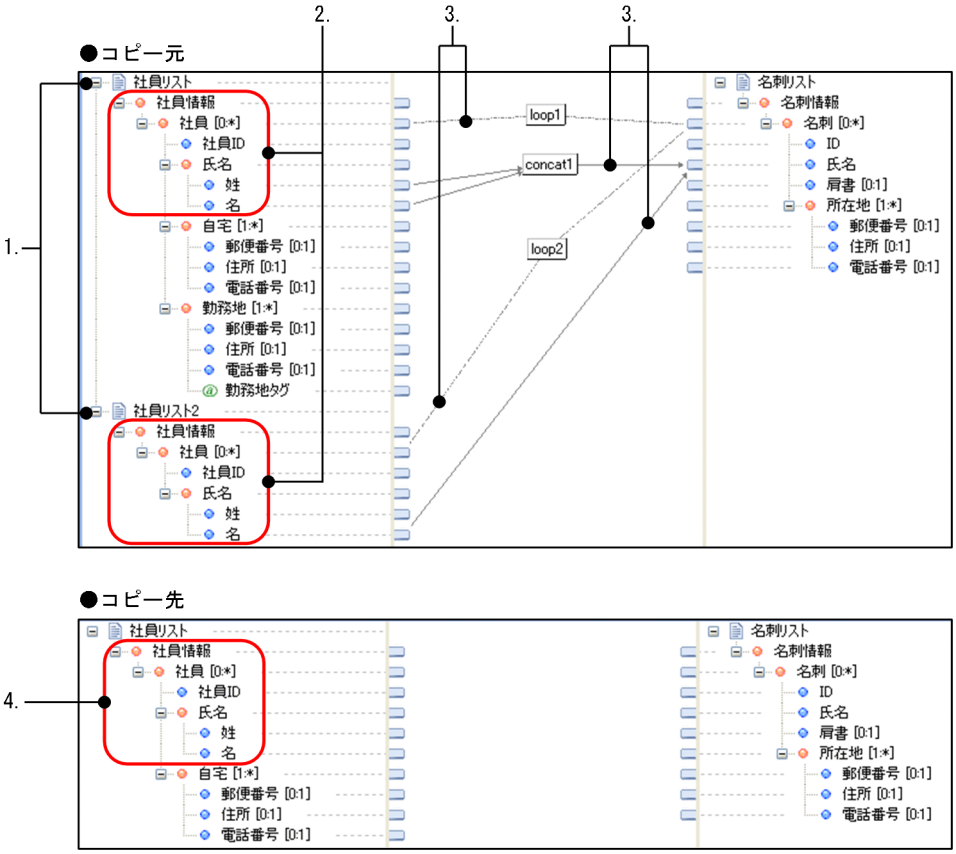
7.8.6　マッピング定義をコピーする場合の注意事項

コピー元のスキーマの内容およびコピー先のスキーマの内容によって，意図したコピーが実行されない場合があります。マッピング定義をコピーする場合の注意事項について説明します。

(1)　コピーする候補が複数ある場合

次に示す 1.～4.の条件がすべて該当する場合，コピーする候補が複数あるとみなされます。

図 7-44　コピーする候補が複数ある場合のコピー元およびコピー先の例



条件

1. コピー元のマッピング定義の変換元スキーマに，複数のスキーマが登録されている
2. コピー元のマッピング定義の変換元スキーマに，パスが同じノードが存在する

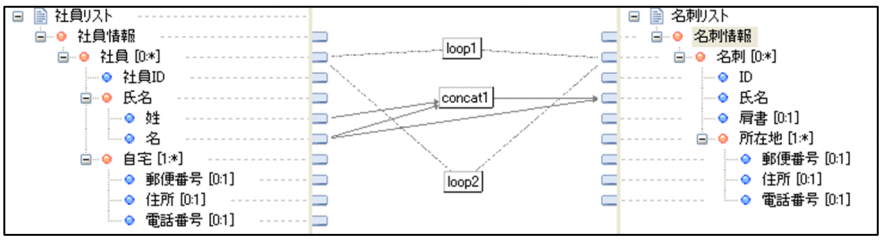
3. コピー元のマッピング定義に，2.のノードを含むマッピング情報がある

4. コピー先の変換元スキーマの一つに，2.と同じパスのノードが存在する

「図 7-44」の場合のコピー結果を次の図に示します。

図 7-45　コピーする候補が複数ある場合のコピー結果

●コピー先



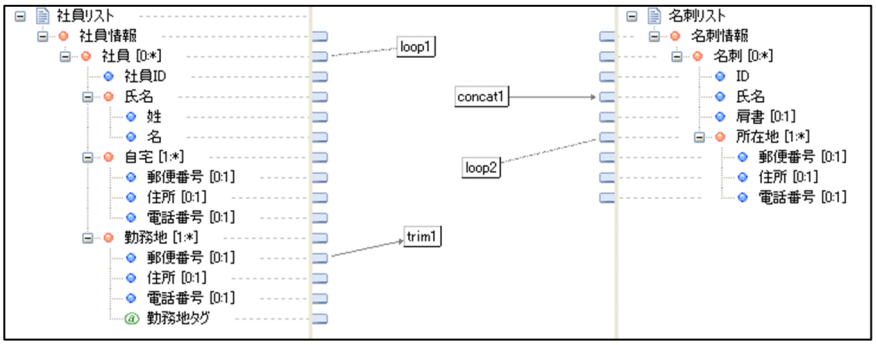
(2)　コピー元のマッピング定義が未完成の場合

変換元ノードと変換先ノード同士を接続していない，未完成のマッピング定義をコピー元のマッピング定義とした場合，変換先ノードに対するマッピングだけがコピーされ，変換元ノードに対するマッピングはコピーされません。

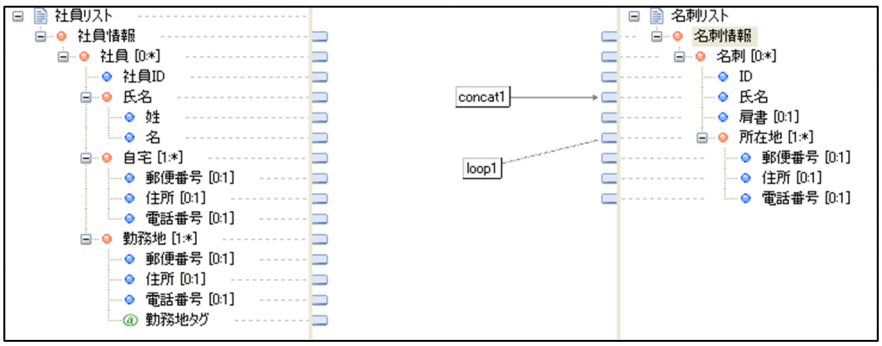
コピー元のマッピング定義が未完成の場合に，マッピング定義をコピーしたときの例を次の図に示します。

図 7-46　コピー元マッピング定義が未完成の場合のマッピング定義のコピー

●コピー元



●コピー先

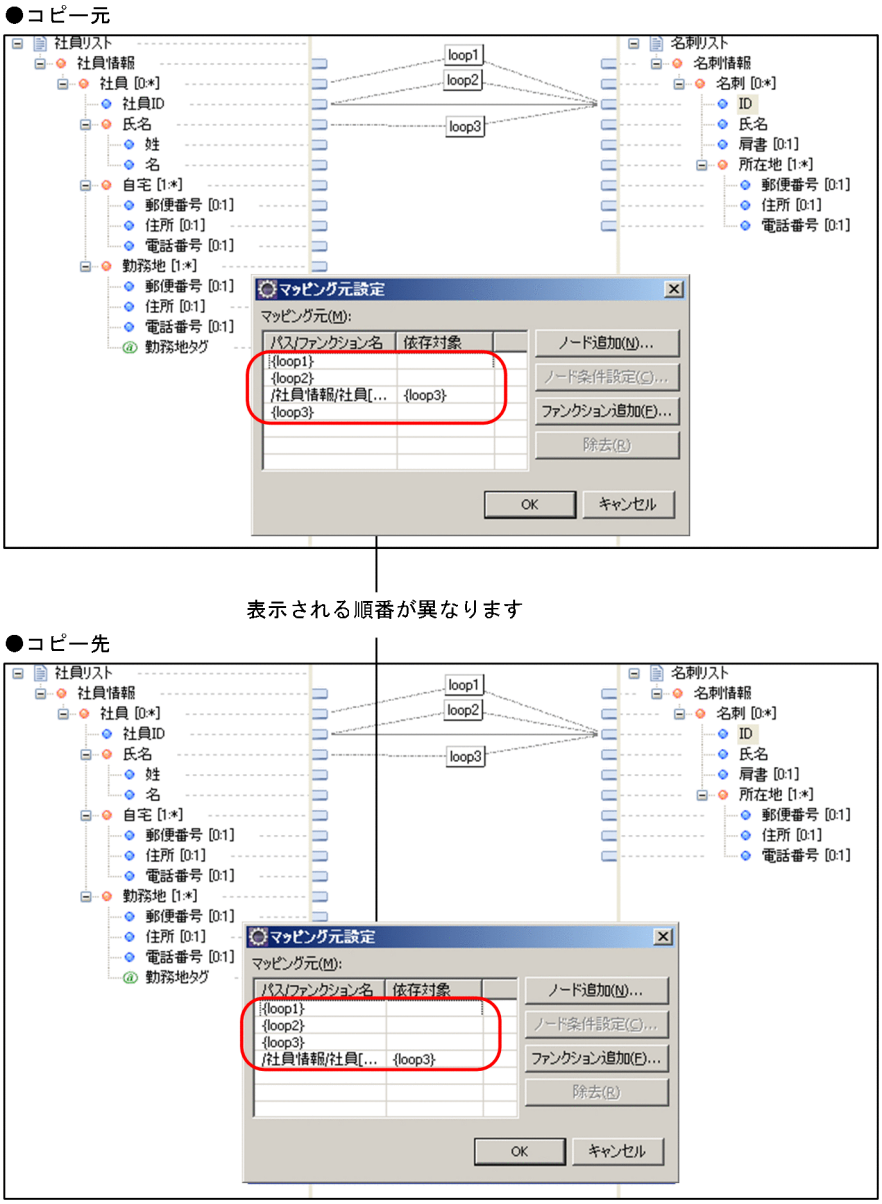


(3)　1 つの要素に複数のくり返しファンクションが接続されている場合

1 つの単純内容要素または複合内容要素に，複数のくり返しファンクションが接続されたマッピング定義をコピー元とした場合，マッピング定義をコピーすると，最初にくり返しファンクションがコピーされます。このとき，［マッピング元設定］ダイアログを表示すると，コピー先とコピー元で［パス/ファンクション名］の表示順が異なることがあります。ただし，マッピング定義のコピー結果に影響はありません。

1 つの要素に複数のくり返しファンクションが接続されている場合に，マッピング定義をコピーしたときの例を次の図に示します。

図 7-47　1 つの要素に複数のくり返しファンクションが接続されている場合のマッピング定義のコピー



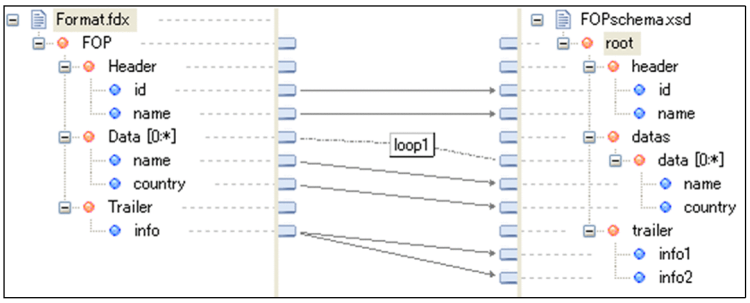
(4)　マッピング対象が制限されている場合

コピー元のマッピング定義ファイルでマッピングされている要素が，コピー先のマッピング定義ファイルでマッピングできない場合，その要素のマッピングはコピーされません。

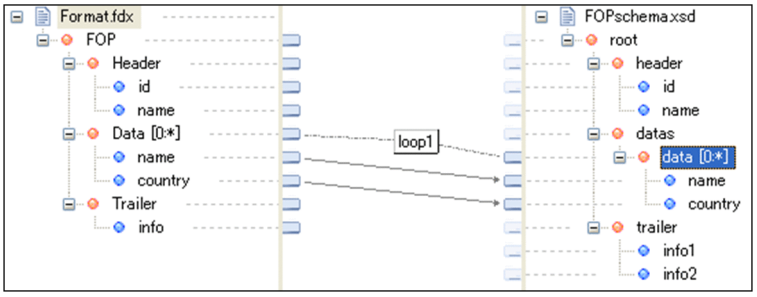
マッピング対象の制限については，「7.4.8　マッピング対象の範囲を限定する」を参照してください。

図 7-48　マッピング対象が制限されている場合のマッピングのコピー

●コピー元



●コピー先

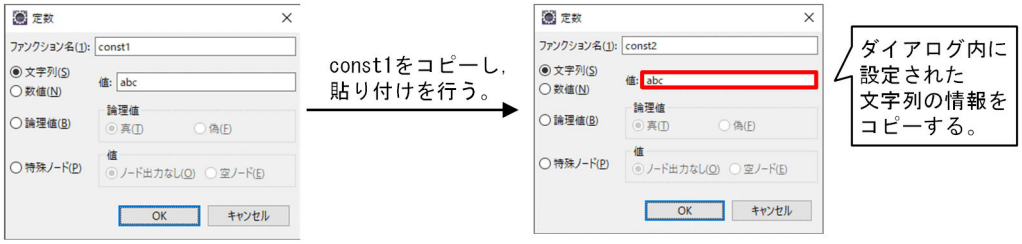


7.9　ファンクションをコピーして貼り付ける

作成したファンクションとファンクションの設定値は，別のファンクションを作成する際に，コピーおよび貼り付けで複製して利用できます。

定数ファンクションと設定値を複製する例を次の図に示します。

図 7-49　定数ファンクションと設定値の複製



ファンクションの複製は，マッピングビューア上で選択したファンクションごとに実行します。マッピング情報ごとコピーしたい場合は，「7.8　マッピング定義をコピーする」を参照してください。

7.9.1　ファンクションのコピー

ファンクションのコピーは，1 つまたは複数のファンクションを選択した状態で右クリックして［コピー］を選択するか，［Ctrl］キー＋［C］キーを押して実行します。

コピーできるファンクション，マッピング線，およびファンクションの設定値について説明します。

(1)　コピーできるファンクション

すべてのファンクションがコピーできます。

(2)　コピーできるマッピング線

マッピング線（代入線，条件線，くり返し対応線，およびくり返し連携線）が，次の条件に一致した場合にコピーできます。

- マッピング線の両端が選択された状態のファンクションである。
- 変換元ノードアダプタと，選択された状態のファンクションにマッピング線が引かれている。

(3)　コピー元と異なる設定値がコピー先に設定される場合

設定値の種類によって，コピー元の設定値と異なった設定値がコピー先のファンクションに設定される場合があります。設定値の種類ごとのコピー先の設定値を次に示します。

(a)　ノードに関する設定値

コピー元ファンクションの設定値がそのまま設定されます。

(b)　ファンクションに関する設定値

ファンクションに関する設定値は，マッピング線がコピーできるかどうかによって次のように変わります。

- コピーできるマッピング線の場合，コピー先のファンクションの設定値に置き換わります。
- コピーできないマッピング線の場合，空（空欄）となります。

コピーできるマッピング線については，「(2)　コピーできるマッピング線」を参照してください。

また，コピー先のファンクション名は，貼り付け時に設定します。

(c)　ノードおよびファンクションと関連がない設定値

コピー元ファンクションの設定値がそのまま設定されます。

(4)　条件内でファンクションを参照している場合の複製例

選択ファンクションの条件内で定数ファンクションを参照している場合の複製例を次に示します。

図 7-50　条件内で参照しているファンクションがコピー対象の場合

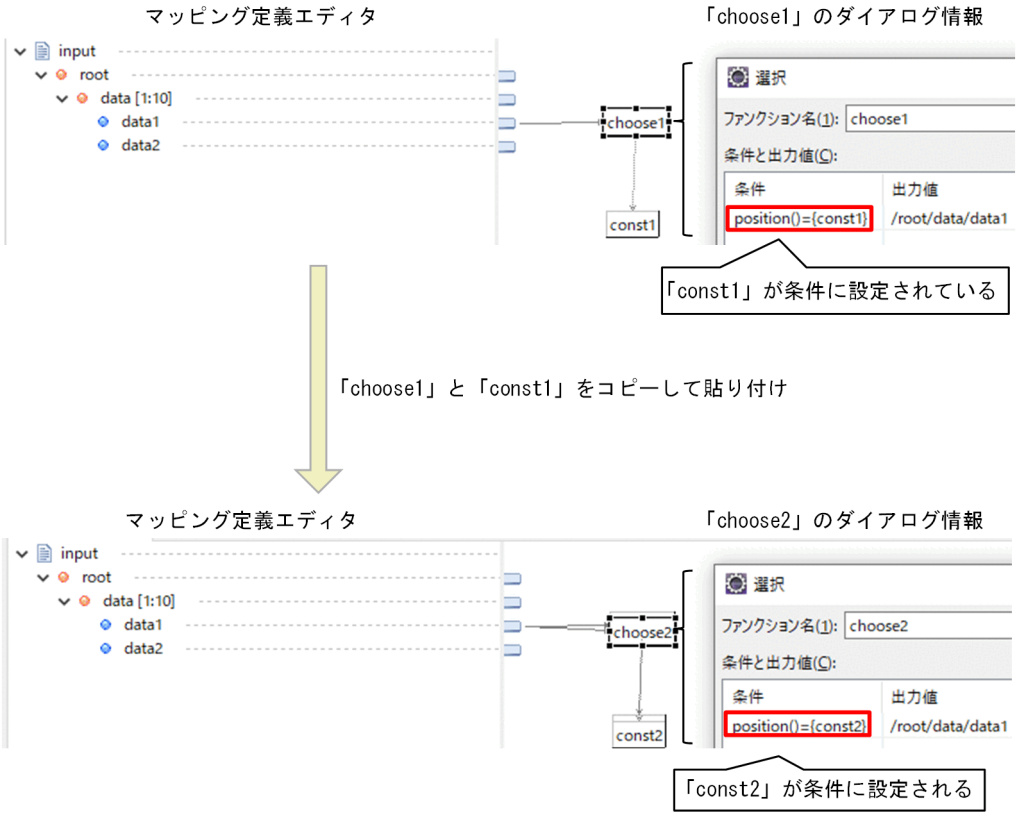
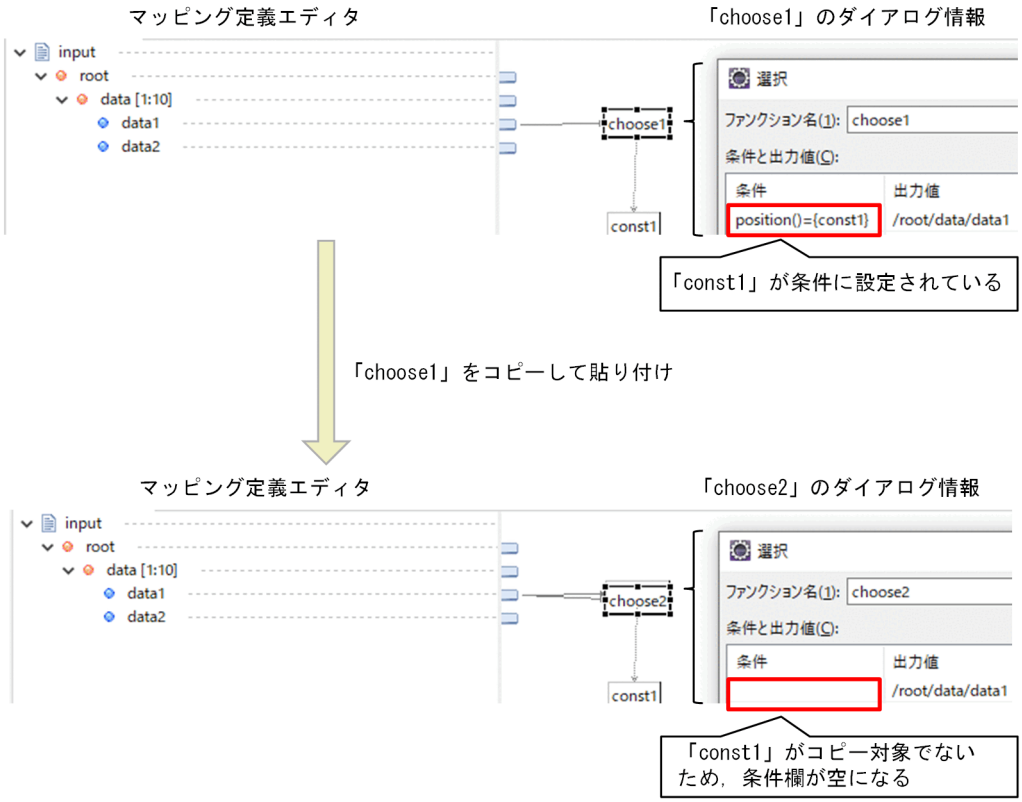


図 7-51　条件内で参照しているファンクションがコピー対象ではない場合



注意事項

コピーを実行した時点のファンクションの設定値がコピーされます。コピー後にコピー元ファンクションの設定値を変更しても，コピー先の設定値には反映されません。

7.9.2　ファンクションの貼り付け

ファンクションの貼り付けは，1 つまたは複数のファンクションをコピーしている状態で，マッピングビューア上で右クリックして［貼り付け］を選択するか，［Ctrl］キー＋［V］キーを押して実行します。

ファンクションの貼り付け後の動作，およびほかのエディタ上での動作を次に示します。

- ファンクションの貼り付け後は，貼り付けたファンクションが選択された状態になります。
- ファンクションの貼り付け後に，操作を元に戻したりやり直したりできます。
  マッピングビューア上で右クリックして［元に戻す］または［やり直し］を選択するか，［Ctrl］キー＋［Z］キーまたは［Ctrl］キー＋［Y］キーを押してください。
- ほかのエディタにファンクションの貼り付けはできません。

貼り付けたファンクションの名称，および貼り付け位置について以降で説明します。

(1)　貼り付け後のファンクション名

貼り付け後のファンクション名には「ファンクションタイプ（例：concat）＋連番」が設定されます。この連番は，ファンクション名が重複しないように，マッピングした順に 1 から付けられる自然数の通番です。

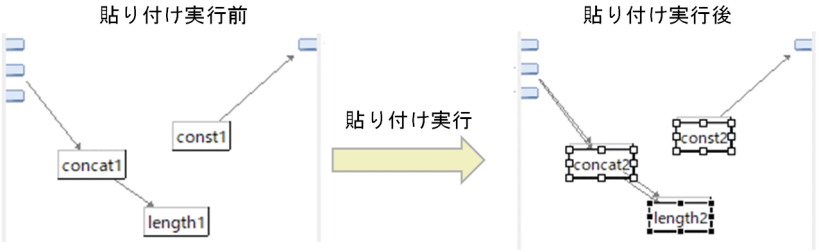
ファンクションの名称は，ファンクションを直接編集するか，またはファンクションをダブルクリックすると表示されるダイアログで変更できます。ファンクション名の変更方法の詳細は，「7.12　ファンクション名の直接編集」を参照してください。

(2)　ファンクションの貼り付け位置

コピー先ファンクションは，コピー実行時のコピー元ファンクションの下側に貼り付けられます。

ファンクションの貼り付け例を次の図に示します。

図 7-52　ファンクションの貼り付け位置



(3)　マッピング線の作成

コピー実行時にコピー対象となったマッピング線を作成して貼り付けます。

コピーできるマッピング線については，「7.9.1(2)　コピーできるマッピング線」を参照してください。

7.10　カスタムファンクションで使用する Java プログラムの作成

カスタムファンクションを使用すると，任意の Java プログラムをファンクションとして使用できます。Service Architect が提供するほかのファンクションではできない加工をしたい場合にカスタムファンクションを使用します。

カスタムファンクションには次の種類があります。

- 文字列型カスタムファンクション
  戻り値が文字列型のカスタムファンクションです。

戻り値の型	java.lang.String

- ノード一覧型カスタムファンクション
  戻り値がノード一覧型のカスタムファンクションです。戻り値として，ノードを直接返すことができます。

戻り値の型	org.w3c.dom.NodeList

引数が Object 型のカスタムファンクションには，単純内容要素，複合内容要素，any 要素，または anyAttribute 属性などノード自身を渡すことができます。

実際に引数に渡されるデータ型は，次のように引数の内容によって異なります。

表 7-7　引数の定義内容

引数の内容	引数の実際の型
単純内容要素・複合内容要素	org.w3c.dom.NodeList
属性	org.w3c.dom.NodeList
any 要素	org.w3c.dom.NodeList
anyAttribute 属性	org.w3c.dom.NodeList
文字列型カスタムファンクション	java.lang.String
ノード一覧型カスタムファンクション	org.w3c.dom.NodeList
そのほかのファンクション※	java.lang.String

注※
くり返しファンクション，選択ファンクションは除きます。また，引数（マッピング元）が XPath ファンクションの場合，カスタムファンクションに渡される引数の型は，XPath 式の評価結果によって java.lang.String または org.w3c.dom.NodeList になります。

ここでは，カスタムファンクションから呼び出す Java プログラムの作成方法について説明します。なお，カスタムファンクションでは，呼び出す Java プログラムのことを変換ファンクションと呼びます。Java プログラムを呼び出す方法については，「7.5.23　ユーザが作成した任意の Java プログラムを呼び出す」を参照してください。

変換ファンクションは，次の手順で作成します。

1. 変換ファンクション定義ファイルの作成

   変換ファンクションの構成を定義した変換ファンクション定義ファイルを作成します。変換ファンクション定義ファイルの作成方法については，「7.10.1　変換ファンクション定義ファイルの作成」を参照してください。

2. Java 雛形ファイルの生成

   作成した変換ファンクション定義ファイルを基に，Java プログラムのコーディング時に使用する Java 雛形ファイルを生成します。Java 雛形ファイルの生成方法については，「7.10.2　Java 雛形ファイルの生成」を参照してください。

3. Java プログラムのコーディング，ビルド，およびデバッグ

   Java 開発ツールを使用して，Java プログラムのコーディング，ビルド，およびデバッグを実施します。Java プログラムのコーディング，ビルド，およびデバッグ方法については，「7.10.4　Java プログラムのコーディング，ビルド，およびデバッグ」を参照してください。

4. Java プログラムのパッケージング

   ビルドツールを使用して，作成した Java プログラムを jar ファイルにパッケージングします。Java プログラムのパッケージング方法については，「7.10.5　Java プログラムのパッケージング」を参照してください。

注意事項

- 呼び出す Java プログラムの引数の数に応じてメモリ所要量（スタックメモリ）が増加します。そのため，引数が多いとスタックオーバーフローのエラーが発生するおそれがあります。引数は，多くても 10 個程度を目安としてください。
- Java プログラムの呼び出し時に確保したリソースは，処理が終了しても保持し続けるため，システム全体に高い負荷がかかった場合に，Java ヒープ不足や Metaspace 領域不足などが原因で OutOfMemoryError が発生することがあります。そのため，OutOfMemoryError が発生したときに適切にリソースを解放する処理や，ロールバックなどのエラー処理を実装しておく必要があります。

7.10.1　変換ファンクション定義ファイルの作成

変換ファンクション定義ファイルとは，カスタムファンクションから呼び出す変換ファンクションの構成を定義したファイルです。変換ファンクション定義ファイルは次のことに使用します。

- Java 雛形ファイルの生成
- カスタムファンクションの定義

変換ファンクション定義ファイルの記述例を次に示します。斜体部分がユーザの指定するデータです。

```
<?xml version="1.0" encoding="UTF-8"?>
<func:customFunc xmlns:func="http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/functions">
 <func:jar name="sample.jar">
  <func:package name="jp.co.Hitachi.soft.sample">
   <func:class name="SampleClass">
    <func:method name="sampleFunction">
     <func:comment>sampleFunction の説明</func:comment>
     <func:arguments>
      <func:argument name="arg1">
       <func:comment>arg1の説明</func:comment>
      </func:argument>
      <func:argument name="arg2">
       <func:comment>arg2の説明</func:comment>
      </func:argument>
      <func:argument name="arg3">
       <func:comment>arg3の説明</func:comment>
      </func:argument>
     </func:arguments>
    </func:method>
    <func:method name="sampleStringFunction">
     <func:comment>sampleStringFunctionの説明</func:comment>
     <func:arguments>
      <func:argument name="arg1" type="String">
       <func:comment>arg1の説明</func:comment>
      </func:argument>
     </func:arguments>
     <func:return type="String"/>
    </func:method>
    <func:method name="sampleNodeListFunction">
     <func:comment>sampleNodeListFunctionの説明</func:comment>
     <func:arguments>
      <func:argument name="arg1" type="Object">
       <func:comment>arg1の説明</func:comment>
      </func:argument>
     </func:arguments>
     <func:return type="NodeList"/>
    </func:method>
   </func:class>
  </func:package>
 </func:jar>
</func:customFunc>
```

変換ファンクション定義ファイルは，ユーザが XML エディタを使って作成します。ここでは，Eclipse の XML エディタを使って変換ファンクション定義ファイルを作成する方法を説明します。

1. Eclipse のメニューから，［ファイル］ − ［新規］ − ［その他］を選択します。
   ［新規］ダイアログが表示されます。

2. ［XML］ − ［XML File］を選択し，［次へ］ボタンをクリックします。
   ［XML ファイル］ページが表示されます。

3. 変換ファンクション定義ファイルを保存するディレクトリ，および変換ファンクション定義ファイル名を指定して，［次へ］ボタンをクリックします。

［次から XML ファイルを作成］ページが表示されます。

4. ［XML スキーマ・ファイルから XML ファイルを作成］ラジオボタンを選択し，［次へ］ボタンをクリックします。
   ［XML スキーマ・ファイルの選択］ページが表示されます。

5. ［ファイルのインポート］ボタンをクリックします。
   ［インポート］ダイアログが表示されます。

6. ［次のディレクトリーから］に次のディレクトリを指定します。

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥customfunc
```

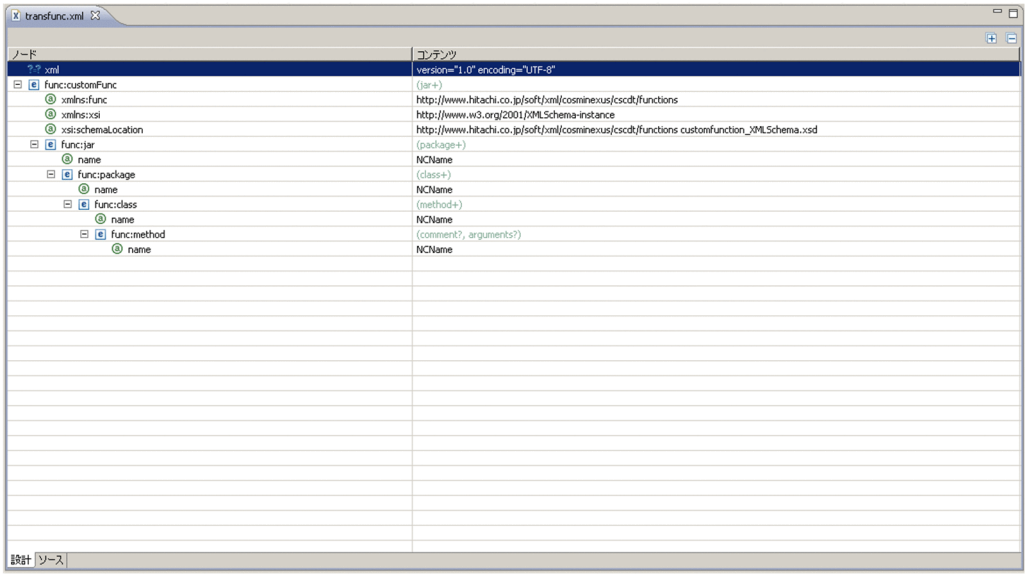
7. ［customfunc］チェックボックスおよび［customfunction_XMLSchema.xsd］チェックボックスにチェックします。

8. ［インポート先フォルダ］にスキーマファイルをインポートするフォルダを指定します。

9. ［終了］ボタンをクリックします。
   再度［XML スキーマ・ファイルの選択］ページが表示されます。

10. ［次へ］ボタンをクリックします。
    ［ルート要素の選択］ページが表示されます。

11. ［ルート要素］に［customFunc］を指定し，［終了］ボタンをクリックします。
    変換ファンクション定義ファイルの雛形が生成されます。



12. 変換ファンクション定義ファイルの雛形に必要な要素を追加し，変換ファンクション定義ファイルを作成します。
    変換ファンクション定義ファイルで使用するタグを次に示します。

表 7-8　変換ファンクション定義ファイルで使用するタグ一覧

項番	要素・属性名	内容	種別	個数		文字数	
				最小	最大	最小	最大
1	customFunc	変換ファンクション定義	要素	1	1	−	−
2	jar	jar ファイル情報	要素	1	255	−	−
3	name	jar ファイル名	属性	1	1	1	100
4	package	パッケージ情報	要素	1	255	−	−
5	name	パッケージ名	属性	1	1	1	255
6	class	クラス情報	要素	1	255	−	−
7	name	クラス名	属性	1	1	1	100
8	method	メソッド情報	要素	1	255	−	−
9	name	メソッド名	属性	1	1	1	100
10	comment	メソッドコメント	要素	0	1	0	1,024
11	arguments	引数情報	要素	0	1	−	−
12	argument	引数	要素	1	255	−	−
13	name	引数名	属性	1	1	1	100
14	type	引数の型（String または Object）	属性	0	1	−	−
15	comment	引数コメント	要素	0	1	0	1,024
16	return	戻り値	要素	0	1	−	−
17	type	戻り値の型（String または NodeList）	属性	1	1	−	−

（凡例）

−：文字数に制限はありません。

また，変換ファンクション定義ファイルを作成する際は，次のことに注意してください。

- クラス名とメソッド名を同じ名称にしないでください。
- クラス名に String を指定しないでください。
- コメントは指定された文字列をそのまま 1 行に埋め込むため，内容によっては，Java の構文エラーになるなど，意図しない結果になることがあります。意図しない結果になった場合は，生成された Java ファイルを直接修正してください。

7.10.2　Java 雛形ファイルの生成

Java 雛形ファイルとは，変換ファンクションを開発するために，パッケージ，クラス，およびメソッド（引数を含む）の構成を記述した Java プログラムの雛形のことです。Java 雛形ファイルは，変換ファンクション定義ファイルを基に，Service Architect によって自動で生成されます。

Java 雛形ファイルの例について説明します。斜体部分が変換ファンクション定義ファイルから入力されたデータです。

Method 定義内に NodeList 型がない例を次に示します。

```
/**
 * Java Template
 */

package jp.co.Hitachi.soft.sample;

public class SampleClass {

    /**
     * カスタムファンクションに関する説明・・・
     *
     * @param arg1
     *  選択している引数に関する説明・・・
     * @param arg2
     *  選択している引数に関する説明・・・
     * @param arg3
     *  選択している引数に関する説明・・・
     *
     * @return
     */
    public static String SampleFunction(String arg1, String arg2, String arg3) {

        // TODO Add Java Code Here
        return "";
    }
}
```

Method 定義内に NodeList 型がある例を次に示します。

```
/**
 * Java Template
 */

package jp.co.Hitachi.soft.cscdt.sample;

import org.w3c.dom.NodeList;

public class SampleFunctionClass {

    /**
     * カスタムファンクションに関する説明・・・
     *
     * @param arg1
```

```
     * 選択している引数に関する説明・・・
     *
     * @return
     */
    public static String SampleStringFunction(String arg1) {

        // TODO Add Java Code Here
        return "";
    }

    /**
     * カスタムファンクションに関する説明・・・
     *
     * @param arg1
     *  選択している引数に関する説明・・・
     *
     * @return
     */
    public static NodeList SampleNodeListFunction(Object arg1) {

        // TODO Add Java Code Here
        return null;
    }
}
```

「import org.w3c.dom.NodeList;」は，Class 定義内に NodeList 型の戻り値を持つ Method 定義が 1 つ以上存在する場合に追加します。ただし，Method 定義内の戻り値の型がすべて String 型の場合は追加しません。

「// TODO Add Java Code Here」に処理を実装して，Java プログラムを作成します。

Java 雛形ファイルを生成する手順を次に示します。

1. Eclipse のメニューから［ファイル］－［新規］－［Java プロジェクト］を選択します。
   ［新規 Java プロジェクト］ダイアログが表示されます。

2. プロジェクト名を指定し，［終了］ボタンをクリックします。
   Eclipse のワークスペースに Java プロジェクトが作成されます。

3. 「7.10.1　変換ファンクション定義ファイルの作成」で作成した変換ファンクション定義ファイルを次の場所に移動します。

   ```
   ＜作成したJavaプロジェクトのディレクトリ＞¥src
   ```

4. Eclipse のメニューから［ファイル］－［更新］を選択します。
   Eclipse の画面が更新されます。

5. ［パッケージ・エクスプローラー］ビューで，変換ファンクション定義ファイルを右クリックし，［HCSC-Definer］－［Java 雛形ファイルの生成］を選択します。
   Java 雛形ファイル出力ウィザードが表示されます。

6. Java 雛形ファイルの出力場所，および出力ファイルの文字コードセットを指定して，［次へ］を指定します。

   出力文字コードは［MS932］，［UTF-8］，［UTF-16］から選択できます。なお，［UTF-16］には BOM 制御コード（0xFEFF）が付きます。

7. Java 雛形ファイルに出力するクラスのチェックボックスにチェックし，［終了］をクリックします。

   Java 雛形ファイル出力ウィザードが閉じ，指定した場所に Java 雛形ファイルが生成されます。

7.10.3　変換ファンクションからの外部 jar の参照

カスタムファンクションでは，変換ファンクション jar ファイルとは別に，複数のユーザ定義 jar ファイル（外部 jar）を扱うことができます。

変換ファンクション jar ファイルから外部 jar を参照したい場合，usrconf.cfg（java アプリケーション用オプション定義ファイル）に外部 jar を追加します。設定内容を説明します。

usrconf.cfg の格納先

```
<サービスプラットフォームのインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥サーバ名¥usrconf.cfg
```

usrconf.cfg の記述形式

```
add.class.path=<外部jar名称>※
```

注※　外部 jar の名称はフルパスで設定する。

指定例

```
add.class.path=C:¥usrFunc¥usrFunc.jar
```

上記の指定例は外部 jar の格納先を「C:¥usrFunc¥usrFunc.jar」に指定しています。変換ファンクション jar ファイル実行時に参照されます。

usrconf.cfg の詳細は，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル)」を参照してください。

7.10.4　Java プログラムのコーディング，ビルド，およびデバッグ

作成された Java 雛形ファイルを基に，Java プログラムをコーディング，ビルド，およびデバッグします。Java プログラムのコーディング，ビルド，およびデバッグには，Eclipse などの Java 開発ツールを使用します。

Java プログラムのインターフェース仕様を次に示します。

表 7-9　Java プログラムのインターフェース仕様

項番	インターフェース	内容
1	メソッド	アクセス修飾子は「public」で，静的メソッド（static 宣言）とします。 オーバーロードはできません。
2	引数	0 個以上の引数を持つことができます。 String 型または Object 型の入力パラメタとなります。
3	関数値	String 型または NodeList 型の関数値を設定してください。null を返すことはできません。
4	例外	例外をスローした場合は，データ変換処理が失敗します。

Java プログラムを作成する際には，次のことに注意してください。

- 外部参照したい jar ファイルがある場合は，usrconf.cfg（J2EE サーバ用オプション定義ファイル）にクラスパスを追加してください。
- カスタムファンクションはマルチスレッドで動作します。そのため，Java プログラムはスレッドセーフとなるように実装してください。
- 外部リソースにアクセスする Java プログラムを実装しないでください。

7.10.5　Java プログラムのパッケージング

コーディング，ビルド，およびデバッグした Java プログラムを jar ファイルにパッケージングします。Java プログラムのパッケージングには，Ant などのビルドツールを使用します。

パッケージングした jar ファイルは次の場所に格納します。HCSC サーバをクラスタ構成で運用している場合は，それぞれの環境に格納してください。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥userlib¥customfunc
```

上記のディレクトリに格納された jar ファイルは，同一マシン上のすべての HCSC サーバで共有されます。

usrconf.properties（J2EE サーバ用ユーザプロパティファイル）にファイルパスを指定した場合，HCSC サーバ単位に jar ファイルの格納先を変更できます。この場合，usrconf.properties に指定されたファイルパスが上記ディレクトリより優先されます。

なお，HCSC サーバに jar ファイルを格納する場合は，J2EE サーバを再起動してください。

HCSC サーバに jar ファイルを格納する方法の詳細については，マニュアル「サービスプラットフォームシステム構築・運用ガイド」の「2.3.8　カスタムファンクションで使用する jar ファイルを格納する」を参照してください。

7.11　マッピングの条件

マッピング元（変換元ノードまたはファンクション），マッピング先（変換先ノードまたはファンクション），およびマッピング線の種類や組み合わせによってはマッピングできない場合があります。ここでは，マッピングできる条件について説明します。

7.11.1　マッピング対象とマッピング対象外

マッピングでは，マッピング対象の要素（ノード）とマッピング対象外の要素（ノード）があります。変換元ノードおよび変換先ノードに，マッピング対象外の要素は指定できません。マッピング対象外の要素を次の表に示します。

表 7-10　マッピング対象外の要素

項目	内容
出現回数が 1 回に固定されていない，かつ名前が指定されていない要素（コンポジタ）	名前が指定されていない complexType 要素の次の子要素（コンポジタ）のうち，出現回数が 1 回に固定されていない要素の子孫要素は，マッピング対象外とします。 • sequence※1 • choice • all これらの要素（コンポジタ）は，#anonymous※2 と表示されます。
再帰構造	再帰構造が定義されていた場合，再帰の開始要素だけを表示し，開始要素はマッピング対象外とします。
混在内容	混在内容である mixed 属性に true が指定されていた場合，要素間テキストはマッピング対象外とします。なお，要素間テキストは表示されません。
代替グループ	abstract 属性に true を指定した要素はマッピング対象外とします。また，SubstitutionGroup 属性の指定は無視されるため，SubstitutionGroup 属性を指定した要素はスキーマツリービューアに表示されません。
2 段階以上ネストした union 要素または list 要素	2 段階以上ネストした union 要素または list 要素（union 要素または list 要素の属性がさらに定義されている）はマッピング対象外とします。
fixed が指定されている単純内容および属性の変換先ノード	fixed が指定されている単純内容および属性の変換先ノードはマッピング対象外とします。

注※1

要素（コンポジタ）下に単純内容要素が 1 つしか存在しない場合は除きます。

注※2

#anonymous については，「7.2.3　電文フォーマット定義ファイル作成時の注意事項」を参照してください。

7.11.2　マッピングできるノードおよびファンクションの対応

マッピング時は，マッピングできる変換元ノード，変換先ノード，およびファンクションの対応を考える必要があります。マッピングできる変換元ノード，変換先ノード，およびファンクションの対応について説明します。

変換元ノードから変換先ノードへのマッピングの対応を次に示します。

表 7-11　変換元ノードから変換先ノードへのマッピング

マッピング元	マッピング先		
変換元ノード	変換先ノード		
	単純内容※1※2	複合内容※1	属性※1※3
単純内容※1※2	○	×	○
複合内容※1	×	×	×
属性※1※3	○	×	○

（凡例）

○：マッピングできます。

×：マッピングできません。または，マッピング線を引けますが，検証でエラーが発生します。

注※1

単純内容，複合内容，および属性については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

注※2

any 要素は any 要素とだけマッピングできます。

注※3

anyAttribute 属性は anyAttribute 属性とだけマッピングできます。

変換元ノードからファンクションへのマッピングの対応を次に示します。

表 7-12　変換元ノードからファンクションへのマッピング

マッピング元	マッピング先																					
変換元ノード	ファンクション																					
	文字列系					数値系				ビット系			ノード系			制御系		その他				
	文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算	ノード数取得	ノード名取得	ノード調査	くり返し	選択	定数	カスタム(文字列型)	カスタム(ノード一覧型)	値置換え	基数変換※1	XPath
単純内容※2※3	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○	×
複合内容※2	×	×	×	×	×	×	×	×	×	×	×	×	○	○	○	○	×	×	○※4	○	×	×	×
属性※2※5	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	×	○	○	○	○	×

(凡例)

○：マッピングできます。

×：マッピングできません。または，マッピング線を引けますが，検証でエラーが発生します。

注※1

入力基数は 16 進数，10 進数，または 2 進数です。

注※2

単純内容，複合内容，および属性については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

注※3

any 要素は，any 要素，文字列型カスタムファンクション，またはノード一覧型カスタムファンクションとマッピングできます。any 要素と any 要素の間にノード一覧型カスタムファンクションを入れたマッピングができます。any 要素とノード一覧型カスタムファンクションの間にファンクションを入れたマッピングはできません。また，文字列型カスタムファンクションの引数が String 型の場合，マッピング線を引いても検証でエラーが発生します。

注※4

文字列型カスタムファンクションの引数が String 型の場合，マッピング線を引いても検証でエラーが発生します。

注※5

anyAttribute 属性は，anyAttribute 属性，文字列型カスタムファンクション，またはノード一覧型カスタムファンクションとマッピングできます。anyAttribute 属性と anyAttribute 属性の間にファンクションを入れたマッピングはできません。anyAttribute 属性とノード一覧型カスタムファンクションの間にファンクションを入れたマッピングはできません。また，文字列型カスタムファンクションの引数が String 型の場合，マッピング線を引いても検証でエラーが発生します。

ファンクションから変換先ノードへのマッピングの対応を次に示します。

表 7-13　ファンクションから変換先ノードへのマッピング

マッピング元		マッピング先		
ファンクション		変換先ノード		
		単純内容※1※2	複合内容※1	属性※1※3
文字列系	文字列連結	○	×	○
	部分文字列取得	○	×	○
	文字数取得	○	×	○
	文字列調査	○	×	○
	空白除去	○	×	○
数値系	数値フォーマット変換	○	×	○
	数値演算	○	×	○
	端数処理	○	×	○
	総和	○	×	○
ビット系	否定演算	○	×	○
	論理演算	○	×	○
	シフト演算	○	×	○
ノード系	ノード数取得	○	×	○
	ノード名取得	○	×	○
	ノード調査	○	×	○
制御系	くり返し	○	○	×

マッピング元			マッピング先		
ファンクション			変換先ノード		
			単純内容※1※2	複合内容※1	属性※1※3
制御系	選択		○	○	○
その他	定数	文字列	○	×	○
		数値	○	×	○
		論理値	○	×	○
		特殊ノード	○	○	○
	カスタム(文字列型)		○	×	○
	カスタム(ノード一覧型)		○	○	×
	値置換え		○	×	○
	基数変換※4		○	×	○
	XPath		○	×	○

(凡例)

○：マッピングできます。

×：マッピングできません。または，マッピング線を引けますが，検証でエラーが発生します。

注※1

単純内容，複合内容，および属性については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.4　データ変換定義画面」を参照してください。

注※2

any 要素は any 要素またはノード一覧型カスタムファンクションとマッピングできます。any 要素と any 要素の間にノード一覧型カスタムファンクションを入れたマッピングができます。ノード一覧型カスタムファンクションと any 要素の間にファンクションを入れたマッピングはできません。

注※3

anyAttribute 属性は anyAttribute 属性とだけマッピングできます。anyAttribute 属性と anyAttribute 属性の間にファンクションを入れたマッピングはできません。

注※4

出力基数は 16 進数，10 進数，または 2 進数です。

ファンクションからファンクションへのマッピング（マッピング先が文字列系，数値系，ビット系）の対応を次に示します。

表 7-14　ファンクションからファンクションへのマッピング（マッピング先が文字列系，数値系，ビット系）

マッピング元※1		マッピング先											
ファンクション		ファンクション											
		文字列系					数値系				ビット系		
		文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算
文字列系	文字列連結	○	○	○	○	○	○	○	○	×	○	○	○
	部分文字列取得	○	○	○	○	○	○	○	○	×	○	○	○
	文字数取得	○	○	○	○	○	○	○	○	×	×	×	×
	文字列調査	○	○	○	○	○	○	○	○	×	×	×	×
	空白除去	○	○	○	○	○	○	○	○	×	○	○	○
数値系	数値フォーマット変換	○	○	○	○	○	○	○	○	×	×	×	×
	数値演算	○	○	○	○	○	○	○	○	×	×	×	×
	端数処理	○	○	○	○	○	○	○	○	×	×	×	×
	総和	○	○	○	○	○	○	○	○	×	×	×	×
ビット系	否定演算	○	○	○	○	○	×	×	×	×	○	○	○
	論理演算	○	○	○	○	○	×	×	×	×	○	○	○
	シフト演算	○	○	○	○	○	×	×	×	×	○	○	○
ノード系	ノード数取得	○	○	○	○	○	○	○	○	×	×	×	×
	ノード名取得	○	○	○	○	○	○	○	○	×	×	×	×
	ノード調査	○	○	○	○	○	○	○	○	×	×	×	×
制御系	くり返し	×	×	×	×	×	×	×	×	×	×	×	×
	選択	×	×	×	×	×	×	×	×	×	×	×	×
その他	値置換え	○	○	○	○	○	○	○	○	×	○	○	○
	基数変換(出力基数：2 進数)	○	○	○	○	○	×	×	×	×	×	×	×

マッピング元※1		マッピング先											
ファンクション		ファンクション											
		文字列系					数値系				ビット系		
		文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算
その他	基数変換(出力基数：10 進数)	○	○	○	○	○	○	○	○	×	×	×	×
	基数変換(出力基数：16 進数)	○	○	○	○	○	×	×	×	×	○	○	○
	カスタム(文字列型)	○	○	○	○	○	○	○	○	×	○	○	○
	カスタム(ノード一覧型)	×	×	×	×	×	×	×	×	×	×	×	×
	定数 文字列	○	○	○	○	○	○	○	○	×	○	○	○
	定数 数値	○	○	○	○	○	○	○	○	×	×	×	×
	定数 論理値	○	○	○	○	○	○	○	○	×	×	×	×
	定数 特殊ノード	×	×	×	×	×	×	×	×	×	×	×	×
	XPath	○	○	○	○	○	○	○	○	○※2	○	○	○

(凡例)

○：マッピングできます。

×：マッピングできません。または，マッピング線を引けますが，検証でエラーが発生します。

注※1

マッピング元のファンクションにくり返しファンクションを 1 つでも指定した場合，ほかのマッピング元もすべてくり返しファンクションを指定してください。

また，マッピング元のファンクションにくり返しファンクション以外を指定した場合，ほかのマッピング元もすべてくり返しファンクション以外を指定してください。

注※2

XPath ファンクションからは，総和ファンクションに対して代入線を引けますが，総和ファンクションのダイアログからは，接続元として XPath ファンクションを選択できません。

また，XPath ファンクションに指定された XPath 式の型が，文字列など node-set 以外の場合は，マッピング定義の検証でエラーになります。

ファンクションからファンクションへのマッピング（マッピング先がノード系，制御系，その他）の対応を次に示します。

表 7-15　ファンクションからファンクションへのマッピング（マッピング先がノード系，制御系，その他）

マッピング元※1		マッピング先												
ファンクション		ファンクション												
		ノード			制御		その他							
		ノード数取得	ノード名取得	ノード調査	くり返し※2	選択※3	値置換え	基数変換(入力基数:2進数)	基数変換(入力基数:10進数)	基数変換(入力基数:16進数)	カスタム(文字列型)	カスタム(ノード一覧型)	定数※4	XPath※5
文字列系	文字列連結	×	×	×	×	○	○	○	○	○	○	○	×	×
	部分文字列取得	×	×	×	×	○	○	○	○	○	○	○	×	×
	文字数取得	×	×	×	×	○	○	×	○	×	○	○	×	×
	文字列調査	×	×	×	×	○	○	×	×	×	○	○	×	×
	空白除去	×	×	×	×	○	○	○	○	○	○	○	×	×
数値系	数値フォーマット変換	×	×	×	×	○	○	×	×	×	○	○	×	×
	数値演算	×	×	×	×	○	○	×	○	×	○	○	×	×
	端数処理	×	×	×	×	○	○	×	○	×	○	○	×	×
	総和	×	×	×	×	○	○	×	○	×	○	○	×	×
ビット系	否定演算	×	×	×	×	○	○	×	×	○	○	○	×	×
	論理演算	×	×	×	×	○	○	×	×	○	○	○	×	×
	シフト演算	×	×	×	×	○	○	×	×	○	○	○	×	×

マッピング元※1		マッピング先												
ファンクション		ファンクション												
		ノード			制御		その他							
		ノード数取得	ノード名取得	ノード調査	くり返し※2	選択※3	値置換え	基数変換(入力基数:2進数)	基数変換(入力基数:10進数)	基数変換(入力基数:16進数)	カスタム(文字列型)	カスタム(ノード一覧型)	定数※4	XPath※5
ノード系	ノード数取得	×	×	×	×	○	○	×	○	×	○	○	×	×
	ノード名取得	×	×	×	×	○	○	×	×	×	○	○	×	×
	ノード調査	×	×	×	×	○	○	×	×	×	○	○	×	×
制御系	くり返し	×	×	×	×	○	×	×	×	×	×	×	×	×
	選択	×	×	×	×	×	×	×	×	×	×	×	×	×
その他	値置換え	×	×	×	×	○	○	○	○	○	○	○	×	×
	基数変換(出力基数：2進数)	×	×	×	×	○	○	○	×	×	○	○	×	×
	基数変換(出力基数：10進数)	×	×	×	×	○	○	×	○	×	○	○	×	×
	基数変換(出力基数：16進数)	×	×	×	×	○	○	×	×	○	○	○	×	×
	カスタム(文字列型)	×	×	×	×	○	○	○	○	○	○	○	×	×
	カスタム(ノード一覧型)	×	×	×	×	○※6	×	×	×	×	○※7	○	×	×
	定数 文字列	×	×	×	×	○	○	○	○	○	○	○	×	×
	定数 数値	×	×	×	×	○	○	×	○	×	○	○	×	×

マッピング元※1			マッピング先												
ファンクション			ファンクション												
			ノード			制御		その他							
			ノード数取得	ノード名取得	ノード調査	くり返し※2	選択※3	値置換え	基数変換(入力基数:2進数)	基数変換(入力基数:10進数)	基数変換(入力基数:16進数)	カスタム(文字列型)	カスタム(ノード一覧型)	定数※4	XPath※5
その他	定数	論理値	×	×	×	×	○	○	×	×	×	○	○	×	×
		特殊ノード	×	×	×	×	×	×	×	×	×	×	×	×	×
	XPath		○※8※9	○※8※9	○※8※9	×	○	○	○	○	○	○※9※10	○※9	×	×

（凡例）

○：マッピングできます。

×：マッピングできません。または，マッピング線を引けますが，検証でエラーが発生します。

注※1

マッピング元のファンクションにくり返しファンクションを 1 つでも指定した場合，ほかのマッピング元もすべてくり返しファンクションを指定してください。

また，マッピング元のファンクションにくり返しファンクション以外を指定した場合，ほかのマッピング元もすべてくり返しファンクション以外を指定してください。

注※2

出力側のくり返しファンクションには，次の制限があります。

くり返しファンクションの入力が，そのくり返しファンクション内にある連携パス自身または，祖先ノード，子孫ノードの場合は接続できません。

注※3

出力側の選択ファンクションには，次の制限があります。

・選択ファンクションの入力にくり返しがすでに接続されている場合，入力にくり返し以外は接続できません。

・選択ファンクションの入力にくり返し以外がすでに接続されている場合，入力にくり返しは接続できません。

注※4

定数ファンクションには，次の制限があります。

変換先ノード，文字列連結ファンクション，数値演算ファンクション，論理演算ファンクション，選択ファンクション，カスタムファンクションの入力に定数ファンクションがすでに接続されている場合，入力に同一の定数ファンクションは接続できません。

注※5

XPath ファンクションには，次の制限があります。

変換先ノード，文字列連結ファンクション，数値演算ファンクション，総和ファンクション，選択ファンクション，カスタムファンクションの入力に，XPath ファンクションがすでに接続されている場合，入力に同一の XPath ファンクションは接続できません。

注※6

選択ファンクションの出力先が属性の場合，検証でエラーが発生します。

注※7

文字列型カスタムファンクションの引数が String 型の場合，マッピング線を引いても検証でエラーが発生します。

注※8

XPath ファンクションからは，ノード系ファンクションに対して代入線を引けますが，ノード系ファンクションのダイアログからは，接続元として XPath ファンクションを選択できません。

注※9

XPath ファンクションに指定された XPath 式の型が，文字列など node-set 以外の場合は，マッピング定義の検証でエラーになります。

注※10

文字列型カスタムファンクションの引数が Object 型の場合，マッピング定義の検証でエラーになります。

7.11.3　接続できるマッピング線の数

変換元ノード，変換先ノード，およびファンクションを接続するマッピング線は，接続できる数が決められています。

出力として接続できるマッピング線の数を次に示します。

表 7-16　出力として接続できるマッピング線の数

ノードまたはファンクション	接続できるマッピング線の数
変換元ノード	複数※1
すべてのファンクション	1※2

注※1

上限値はありません。

注※2

定数ファンクションおよび XPath ファンクションは，代入線だけ複数接続できます。

入力として接続できるマッピング線の数を次に示します。

表 7-17　入力として接続できるマッピング線の数

ノードまたはファンクション		接続できるマッピング線の数
変換先ノード		複数※1
文字列系	文字列連結	複数※1
	部分文字列取得	1

ノードまたはファンクション		接続できるマッピング線の数
文字列系	文字数取得	1
	文字列調査	1
	空白除去	1
数値系	数値フォーマット変換	1
	数値演算	2
	端数処理	1
	総和	複数※1
ビット系	否定演算	1
	論理演算	2
	シフト演算	1
ノード系	ノード数取得	1
	ノード名取得	1
	ノード調査	1
制御系	くり返し	1
	選択	複数※1
その他	定数	0
	カスタム	複数※2
	値置換え	1
	基数変換	1
	XPath	0

注※1

上限値はありません。

注※2

対応する変換ファンクションの引数と同じ数が入力できます。0 の場合もあります。

7.12　ファンクション名の直接編集

マッピング定義で定義したファンクション名を，直接編集して任意の名称に変えることができます。

ファンクション名は「ファンクションタイプ＋通番」で自動的に付与され，同じタイプのファンクションが複数存在する場合の区別は番号だけとなります。このファンクション名を任意の名称に変更することで，同じタイプのファンクションが大量にある場合でも，名称で判別できるようになります。

7.12.1　ファンクション名の編集方法

ファンクション名を変更するには，マッピングビューア上に配置されたファンクションを直接編集する方法とダイアログを使用する方法があります。それぞれの場合の手順を次に説明します。

(1)　マッピングビューア上に配置されたファンクションを直接編集する場合

マッピングビューア上のファンクションを直接選択してファンクション名を編集する手順を次に示します。

1. マッピングビューア上でファンクション名を編集するファンクションを選択します。
2. ファンクションを選択した状態で，左クリックまたは［F2］キーを押します。
   ファンクション名を編集できる状態になります。
3. ファンクション名を入力します。
   ファンクション名を NCName で指定します。指定できる文字数は 1～64 文字です。

(2)　ダイアログを使用する場合

各ファンクションの設定ダイアログを使用してファンクション名を編集する手順を次に示します。

1. 次のどちらかの方法でファンクションを設定するダイアログを表示させます。
   - ファンクションを右クリックして，［設定］を選択します。
   - ファンクションをダブルクリックします。

   ファンクションを設定するダイアログが表示されます。
   各ファンクションのダイアログについては，マニュアル「サービスプラットフォーム リファレンス」の「1.6　データ変換に関連するダイアログ」を参照してください。
2. 任意のファンクション名を入力します。
   ファンクション名を NCName で指定します。指定できる文字数は 1～64 文字です。
3. ［OK］ボタンをクリックします。

7.12.2　編集後のファンクション名の表示

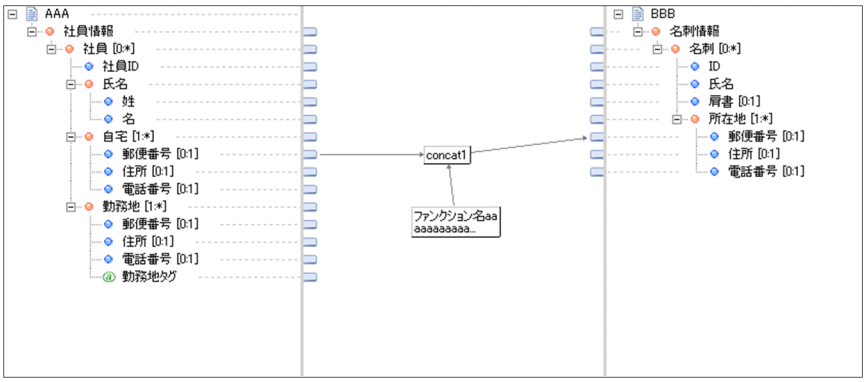
(1)　データ変換定義画面のマッピングビューアに表示されるファンクション名

ファンクション名を編集した場合，次のように表示されます。

- ファンクション名が 1～10 文字の場合
  ファンクション名全体を表示します。
- ファンクション名が 11～20 文字の場合
  10 文字で改行し，ファンクション名全体を表示します。
- ファンクション名が 21～64 文字の場合
  10 文字で改行し，先頭から 19 文字まではファンクション名を表示します。20 文字以降は「...」と省略されます。

編集後のファンクション名の表示例を次に示します。

図 7-53　編集後のファンクション名の表示例（データ変換定義画面の場合）



(2)　ダイアログに表示されるファンクション名

ファンクション名を編集した場合，［条件設定］ダイアログのノード条件などには，波括弧（{　}）で囲んだ形式で表示されます。

編集後のファンクション名の表示例を次に示します。

図 7-54　編集後のファンクション名の表示例（[条件設定] ダイアログの場合）

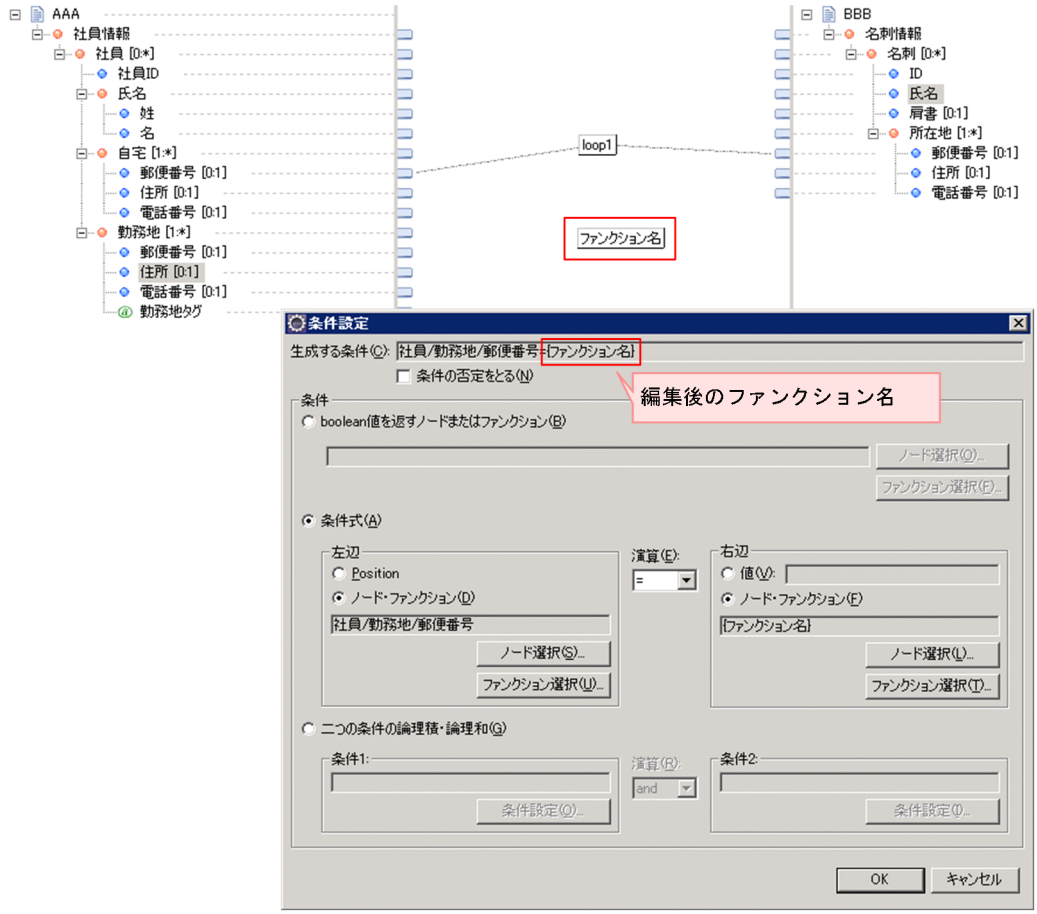
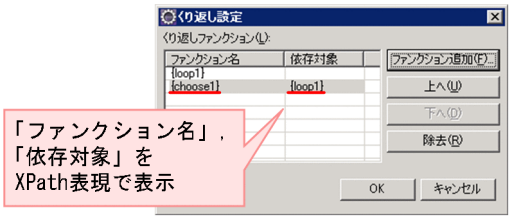


図 7-55　編集後のファンクション名の表示例（[くり返し設定] ダイアログの場合）



7.13　Excel を利用したマッピング定義のインポート

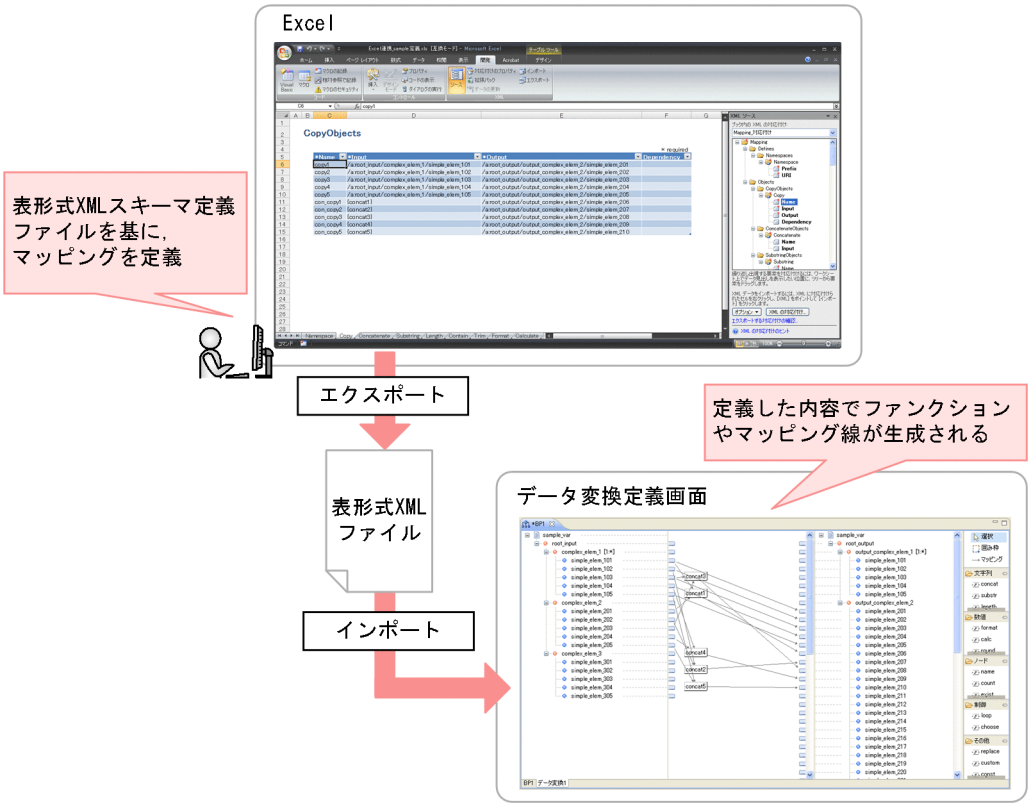
サービスプラットフォームでは，Excel で変換元と変換先の組み合わせ（マッピング）を定義したファイルを開発環境にインポートすることができます。

複数のデータ変換定義で同じようなマッピングをする場合，Excel で作成したマッピング定義を利用することで，1 つのマッピング定義を有効に活用できます。また，マッピングの定義内容を Excel 上で確認できるため，変換元と変換先の接続数が多い場合などに定義時のミスが低減し，開発効率を向上できます。

マッピング定義をインポートするには，表形式 XML スキーマ定義ファイルを基に Excel で定義したマッピングの内容を，表形式 XML ファイルとしてエクスポートします。そのあと，開発環境のデータ変換定義画面にインポートします。データ変換定義画面では，インポートした表形式 XML ファイルに従ってファンクションやマッピング線が生成されます。

マッピング定義のインポートの流れを次に示します。

図 7-56　マッピング定義のインポートの流れ



なお，表形式 XML スキーマ定義ファイルに従った XML ファイルを直接作成して，データ変換定義画面にインポートすることもできます。

Excel 以外のツールを使用して表形式 XML ファイルを作成する場合でも，表形式 XML ファイルが表形式 XML スキーマ定義ファイルに対して妥当であればインポートすることができます。

表形式 XML ファイルの作成手順およびマッピング定義のインポート手順を以降で説明します。

7.13.1　表形式 XML ファイルの作成

表形式 XML ファイルを作成する方法には，あらかじめ表形式 XML スキーマ定義ファイルの各要素が Excel に対応づけられたテンプレートファイルを利用する場合と，表形式 XML スキーマ定義ファイルを基に独自に要素を対応づける場合の 2 種類があります。

> **参考**
>
> テンプレートファイルには，あらかじめ次の設定がされています。
>
> - オブジェクトごとにシートを分けて要素が対応づけられています。
> - 必須項目には項目名の先頭にアスタリスク（*）が付けられています。必須項目は，定義するすべての行で，値を設定する必要があります。空の場合は，エラーとなります。
> - 項目の値が固定値で制約されている場合（"y"と"n"のどちらかしか入力できない項目など）は，データの入力規則が設定され，リストから値を選択できます。

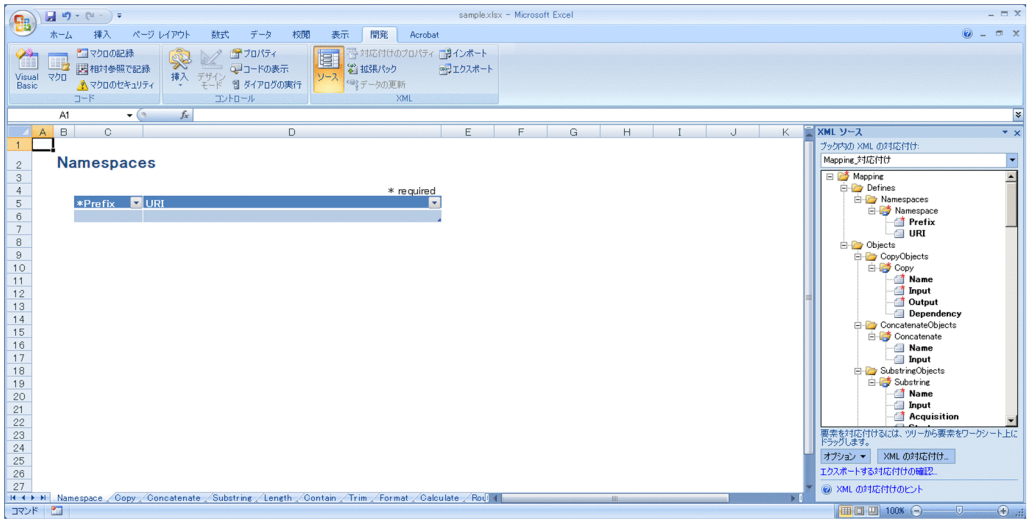
それぞれの場合の作成手順を次に示します。

(1)　テンプレートファイルを利用する場合

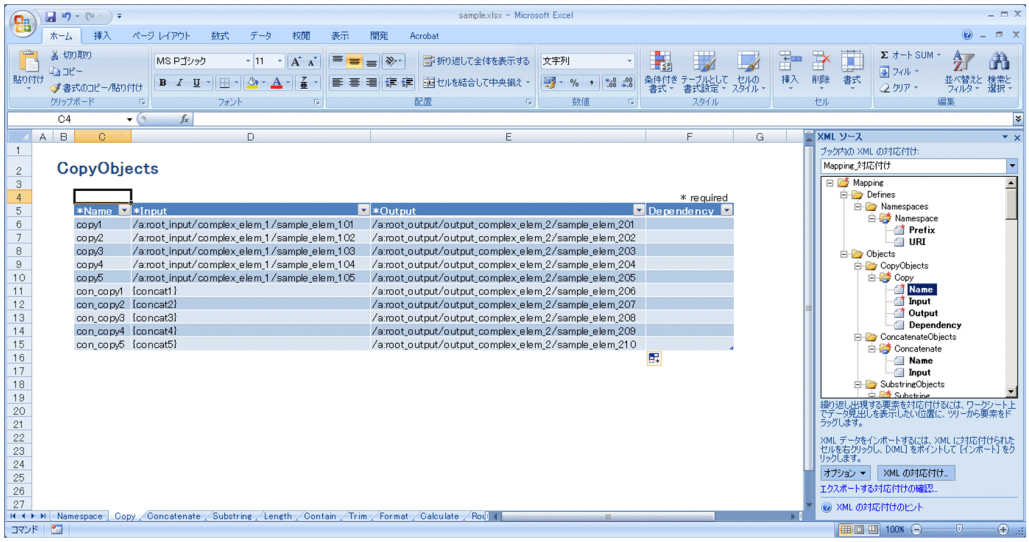
1. テンプレートファイル（dt_import_template.xlsx）をコピーして，任意の名称で保存します。

   テンプレートファイル（dt_import_template.xlsx）の格納場所は「<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥dt_import」です。

2. Microsoft Office Excel を起動して，コピーしたファイルを開きます。



3. XML 要素の対応づけをしたセルに，ファンクションやマッピング線を定義します。



定義方法の詳細は，「7.13.2　マッピング定義の設定」を参照してください。

4. [開発] タブの［エクスポート］ボタンをクリックし，XML 形式でファイルを保存します。

定義内容が表形式 XML ファイルとしてエクスポートされます。

(2)　表形式 XML スキーマ定義ファイルを基に要素を対応づける場合

1. Microsoft Office Excel を起動します。

2. [開発] タブの［ソース］ボタンを選択します。

［XML ソース］作業ウィンドウが表示されます。

3. [XML ソース] 作業ウィンドウの［XML の対応付け...］ボタンをクリックします。

［XML の対応付け］ダイアログが表示されます。

4. [追加] ボタンをクリックし，表形式 XML スキーマ定義ファイル（dt_import_mapping.xsd）を追加します。

表形式 XML スキーマ定義ファイル（dt_import_mapping.xsd）の格納先は「<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥dt_import」です。

［XML ソース］作業ウィンドウにスキーマがツリー表示されます。

作業時にエラーメッセージが表示された場合は，「7.16　Excel を利用したマッピング定義時の注意事項」を参照して対処してください。

5.「***Objects」または「Namespaces」の要素部分をセルにドラッグ＆ドロップします。※

ドラッグ＆ドロップした要素部分のデータを編集できるようになります。

注※

要素を対応づけるときは，オブジェクト（Copy や Concatenate など）ごとにシートを分けてください。同じシートに複数の要素を設定すると，マッピング定義を正しくインポートできません。

6. XML 要素の対応づけをしたセルに，ファンクションやマッピング線を定義します。

定義方法の詳細は，「7.13.2　マッピング定義の設定」を参照してください。

7. [開発] タブの［エクスポート］ボタンをクリックし，XML 形式でファイルを保存します。
定義内容が表形式 XML ファイルとしてエクスポートされます。

7.13.2　マッピング定義の設定

Excel でマッピングを定義するには，表形式 XML スキーマ定義ファイルを基に，各オブジェクトに変換元ノードや変換先ノードなどの値を設定します。

ここでは，マッピング定義の入出力値に定義できる項目，および各項目の設定方法について説明します。

(1)　入出力値に定義できる項目

CopyObjects（変換先ノードへのマッピング）や ConcatenateObjects（文字列連結ファンクション）などオブジェクトの Input には，データ変換定義画面上の変換元ノードのパスまたはファンクション名を設定します。CopyObjects（変換先ノードへのマッピング）の Output には，データ変換定義画面の変換先ノードのパスを設定します。また，入力値に変換元ノードを指定する場合，ノード条件を設定できます。

マッピング定義の各項目で定義できるノードおよびファンクション，ならびに変換元ノードへの条件設定の可否を表に示します。

表 7-18　入出力値の定義項目（ノード）

入出力値の定義項目		設定できるノード										変換元ノードへの条件設定可否
オブジェクト	項目	変換元ノード					変換先ノード					
		単純内容	複合内容	属性	any 要素	anyAttribute 属性	単純内容	複合内容	属性	any 要素	anyAttribute 属性	
CopyObjects（変換先ノードへのマッピング）	Input	○	○	○	○	○	−	−	−	−	−	○
	Output	−	−	−	−	−	○	○	○	○	○	−
	Dependency	×	×	×	×	×	−	−	−	−	−	−
ConcatenateObjects（文字列連結ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
SubstringObjects（部分文字列取得ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
LengthObjects（文字数取得ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○

入出力値の定義項目		設定できるノード										変換元ノードへの条件設定可否
オブジェクト	項目	変換元ノード					変換先ノード					
		単純内容	複合内容	属性	any要素	anyAttribute属性	単純内容	複合内容	属性	any要素	anyAttribute属性	
ContainObjects（文字列調査ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
TrimObjects（空白除去ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
FormatObjects（数値フォーマット変換ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
CalculateObjects（数値演算ファンクション）	Input1	○	×	○	×	×	−	−	−	−	−	○
	Input2	○	×	○	×	×	−	−	−	−	−	○
RoundObjects（端数処理ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
SumObjects（総和ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
NotObjects（否定演算ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
BitOpObjects（論理演算ファンクション）	Input1	○	×	○	×	×	−	−	−	−	−	○
	Input2	○	×	○	×	×	−	−	−	−	−	○
ShiftObjects（シフト演算ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
NameObjects（ノード名取得ファンクション）	Input	○	○	○	×	×	−	−	−	−	−	○
CountObjects（ノード数取得ファンクション）	Input	○	○	○	×	×	−	−	−	−	−	○
ExistObjects（ノード調査ファンクション）	Input	○	○	○	×	×	−	−	−	−	−	○
LoopObjects（くり返しファンクション）	Input	○	○	×	×	×	−	−	−	−	−	○
	RelationalPath	○	○	×	×	×	−	−	−	−	−	○
	SortKey	○	○	○	×	×	−	−	−	−	−	×
ChooseObjects（選択ファンクション）	Condition	○	×	○	×	×	−	−	−	−	−	−

入出力値の定義項目		設定できるノード										変換元ノードへの条件設定可否
オブジェクト	項目	変換元ノード					変換先ノード					
		単純内容	複合内容	属性	any要素	anyAttribute属性	単純内容	複合内容	属性	any要素	anyAttribute属性	
ChooseObjects（選択ファンクション）	Output Value	○	×	○	×	×	−	−	−	−	−	○
ReplaceObjects（値置換えファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
RadixObjects（基数変換ファンクション）	Input	○	×	○	×	×	−	−	−	−	−	○
CustomObjects（カスタムファンクション）	Input	○	○	○	○	○	−	−	−	−	−	○
変換元ノードへの条件設定		○	×	○	×	×	−	−	−	−	−	−

（凡例）

○：定義できます。

×：定義できません。

−：該当しません。

表 7-19　入出力値の定義項目（ファンクション）

入出力値の定義項目		設定できるファンクション																				
オブジェクト	項目	文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算	ノード名取得	ノード数取得	ノード調査	くり返し	選択	値置換え	基数変換	カスタム	定数	XPath
CopyObjects（変換先ノードへのマッピング）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	Output	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
	Dependency	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○	×	×	×	×	×	×
ConcatenateObjects（文字列連結ファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
SubstringObjects（部分文字列取得ファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○

入出力値の定義項目		設定できるファンクション																					
オブジェクト	項目	文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算	ノード名取得	ノード数取得	ノード調査	くり返し	選択	値置換え	基数変換	カスタム	定数	XPath
LengthObjects（文字数取得ファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
ContainObjects（文字列調査ファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
TrimObjects（空白除去ファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
FormatObjects（数値フォーマット変換ファンクション）	Input	○	○	○	○	○	○	○	○	○	×	×	×	○	○	○	×	×	○	○	○	○	○
CalculateObjects（数値演算ファンクション）	Input1	○	○	○	○	○	○	○	○	○	×	×	×	○	○	○	×	×	○	○	○	○	○
	Input2	○	○	○	○	○	○	○	○	○	×	×	×	○	○	○	×	×	○	○	○	○	○
RoundObjects（端数処理ファンクション）	Input	○	○	○	○	○	○	○	○	○	×	×	×	○	○	○	×	×	○	○	○	○	○
SumObjects（総和ファンクション）	Input	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○
NotObjects（否定演算ファンクション）	Input	○	○	×	×	○	×	×	×	×	○	○	○	×	×	×	×	×	○	○	○	○	○
BitOpObjects（論理演算ファンクション）	Input1	○	○	×	×	○	×	×	×	×	○	○	○	×	×	×	×	×	○	○	○	○	○
	Input2	○	○	×	×	○	×	×	×	×	○	○	○	×	×	×	×	×	○	○	○	○	○
ShiftObjects（シフト演算ファンクション）	Input	○	○	×	×	○	×	×	×	×	○	○	○	×	×	×	×	×	○	○	○	○	○
NameObjects（ノード名取得ファンクション）	Input	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○

入出力値の定義項目		設定できるファンクション																					
オブジェクト	項目	文字列連結	部分文字列取得	文字数取得	文字列調査	空白除去	数値フォーマット変換	数値演算	端数処理	総和	否定演算	論理演算	シフト演算	ノード名取得	ノード数取得	ノード調査	くり返し	選択	値置換え	基数変換	カスタム	定数	XPath
CountObjects（ノード数取得ファンクション）	Input	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○
ExistObjects（ノード調査ファンクション）	Input	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	○
LoopObjects（くり返しファンクション）	Input	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	Relational Path	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	SortKey	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
ChooseObjects（選択ファンクション）	Condition	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
	OutputValue	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	○	○	○	○	○
ReplaceObjects（値置換えファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
RadixObjects（基数変換ファンクション）	Input	○	○	○	×	○	×	○	○	○	○	○	○	×	○	×	×	×	○	○	○	○	○
CustomObjects（カスタムファンクション）	Input	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○
変換元ノードへの条件設定		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	×	×	○	○	○	○	○

（凡例）

○：定義できます。

×：定義できません。

－：該当しません。

各オブジェクトの定義項目および定義例の詳細は，「7.15　表形式 XML スキーマ定義ファイルの定義詳細」を参照してください。

(2)　構文規則

各オブジェクトの入出力値に対応する BNF を次の表に示します。

表 7-20　各オブジェクトの入出力値に対応する BNF

オブジェクト/項目	対応する BNF
Copy/Input	条件付き入力パスまたはファンクション
Concatenate/Input	
Substring/Input	
Length/Input	
Contain/Input	
Trim/Input	
Format/Input	
Calculate/Input1	
Calculate/Input2	
Round/Input	
Not/Input	
BitOp/Input1	
BitOp/Input2	
Shift/Input	
Choose/OutputValue	
Radix/Input	
Custom/Input	
Sum/Input	
Name/Input	
Count/Input	
Exist/Input	
Loop/Input	条件付き入力パス
Loop/RelationalPath	
Copy/Output	出力パス
Copy/Dependency	ファンクション
Loop/SortKey	入力パス
Choose/Condition	条件※

注※

入力相対パスは使用できません。

これらの BNF の構文規則を次の表に示します。

表 7-21　BNF の構文規則

BNF	BNF の構文規則
入力パス	::= [変数] ('/' ノード)+
条件付き入力パス	::= [変数] ('/' ステップ)+
ファンクション	::= '{' NCName '}'
出力パス	::= [変数] ('/' ノード)+
条件	::= 基本式 | ('not(' 条件 ')') | '(' 条件 ')'
変数	::= '$' NCName
ステップ	::= ノード ['[' 条件 ']']
ノード	::= (['@'] [名前空間接頭辞 ':'] NCName) | '#any' | '#anyAttribute'
名前空間接頭辞	::= NCName
基本式	::= 入力パス | 入力相対パス | ファンクション | 条件式 | 複合条件式
条件式	::= ('position()' | 入力パス | 入力相対パス | ファンクション) オペレータ (値 | ファンクション | 入力パス | 入力相対パス)
入力相対パス	::= '.' | ノード ('/' ノード)*
複合条件式	::= or 式
or 式	::= and 式 ['or' and 式]
and 式	::= 条件 ['and' 条件]
オペレータ	::= '=' | '!=' | '>' | '<' | '>=' | '<='
値※	::= "'" [^']+ "'"

注※

値は 1 文字以上，かつ 1,024 文字以内で指定できます。ただし，実体参照表現は 1 文字として扱われます。

(3)　変換元ノードの設定

オブジェクトに変換元ノードを設定する場合，Input に変換元ノードのパスを絶対パスで指定します。相対パスでの指定はサポートされません。ノードパスの設定方法については，「(7)　ノードパスの設定」を参照してください。

また，定義項目によっては変換元ノードにノード条件を設定できます。ノード条件の設定方法については，「(6)　ノード条件設定」を参照してください。

(4)　変換先ノードの設定

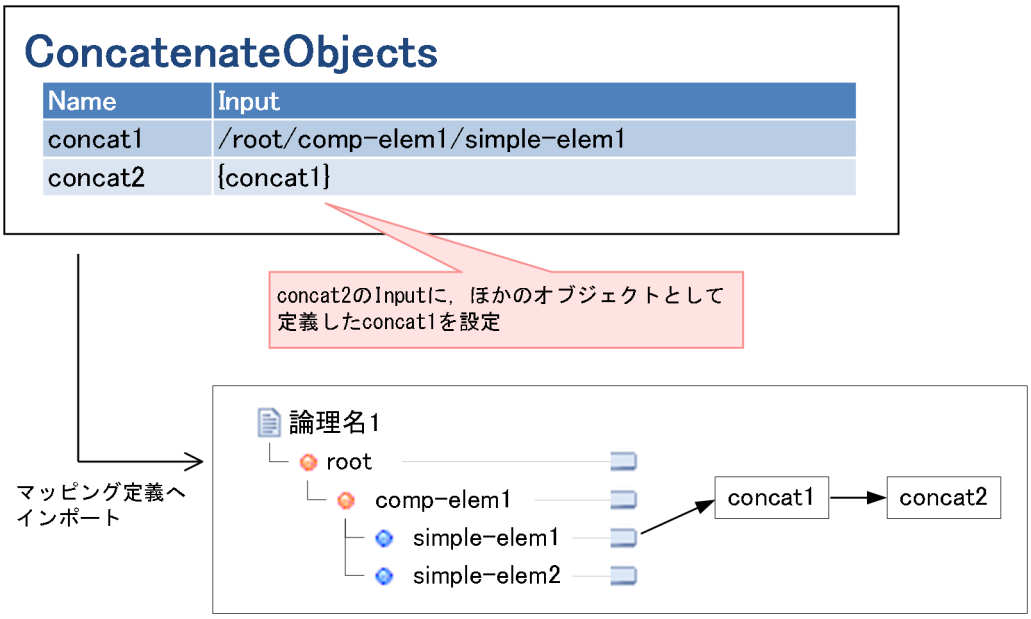
変換先ノードを設定するには，CopyObjects（変換先ノードへのマッピング）の Output に変換先ノードのパスを指定します。ノードパスの設定については「(7)　ノードパスの設定」を参照してください。

(5)　ファンクションの設定

ファンクションには，ほかのオブジェクトで定義したファンクション名を指定します。ファンクション名は，波括弧（{　}）で囲まれた部分に指定します。

定義例を次の図に示します。

図 7-57　定義例（ファンクションの設定）



(6)　ノード条件設定

ノード条件を設定できる項目で変換元ノードを設定した場合，変換元ノードの各要素にノード条件を設定できます。

ノード条件を指定できる変換元ノードまたはファンクションの種類については，表 7-18 および表 7-19 を参照してください。

ノード条件は各要素の後ろの角括弧（[　]）で囲まれた部分に指定します。

ノード条件は，次のどれかの方式で設定します。

表 7-22　ノード条件の方式

#	ノード条件の方式	概要
1	boolean 値を返すノードまたはファンクション	変換元ノードまたはファンクションを指定すると，その存在有無によって boolean 値を返します。
2	条件式	「<左辺><演算><右辺>」の形式で指定した条件に対して boolean 値を返します。左辺および右辺には，変換元ノードやファンクションなどを指定します。
3	二つの条件の論理積・論理和	「<条件 1><演算><条件 2>」上記#1,#2 の条件を 2 つ，and または or で組み合わせて指定した条件に対して boolean 値を返します。

ノード条件設定の変換元ノードパスは，絶対パス，または条件設定対象ノードからの相対パスのどちらかで指定します。ノードパスの指定については，「(7)　ノードパスの設定」を参照してください。

ただし，条件設定の対象となるノードの子孫ノードを条件で使用する場合，絶対パスの指定・相対パスの指定に関係なく，データ変換定義画面にインポートしたあと相対パスとして参照されます。

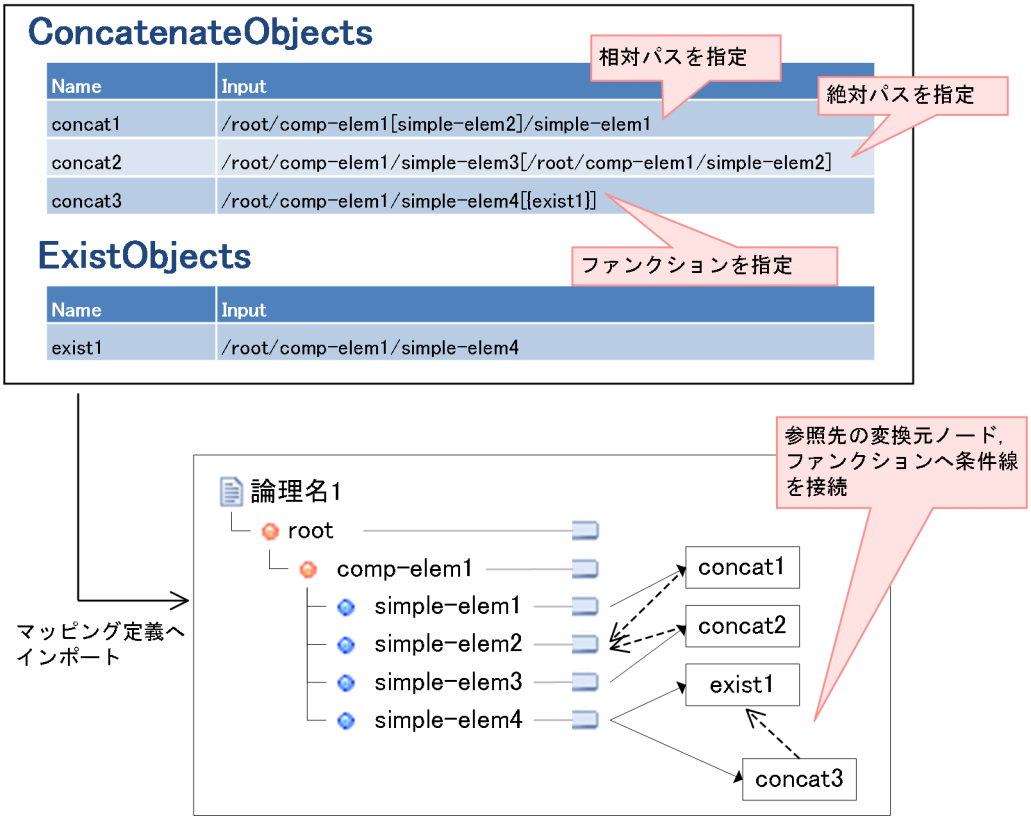
ファンクションの指定方法については，「(5)　ファンクションの設定」を参照してください。

(a)　boolean 値を返すノードまたはファンクション

変換元ノードまたはファンクションを指定すると，その存在有無によって boolean 値を返します。

ノードまたはファンクションの有無を条件に設定した場合の定義例を次の図に示します。

図 7-58　定義例（ノード条件設定 1）



(b)　条件式

「<左辺><演算><右辺>」の形式で指定した条件式に対して boolean 値を返します。左辺，演算，右辺に指定できる値を次の表に示します。

表 7-23　条件式の各項目に指定できる値

条件式の項目	指定できる値
左辺	次のどれかを指定できます。 • "position()"で指定するポジション関数

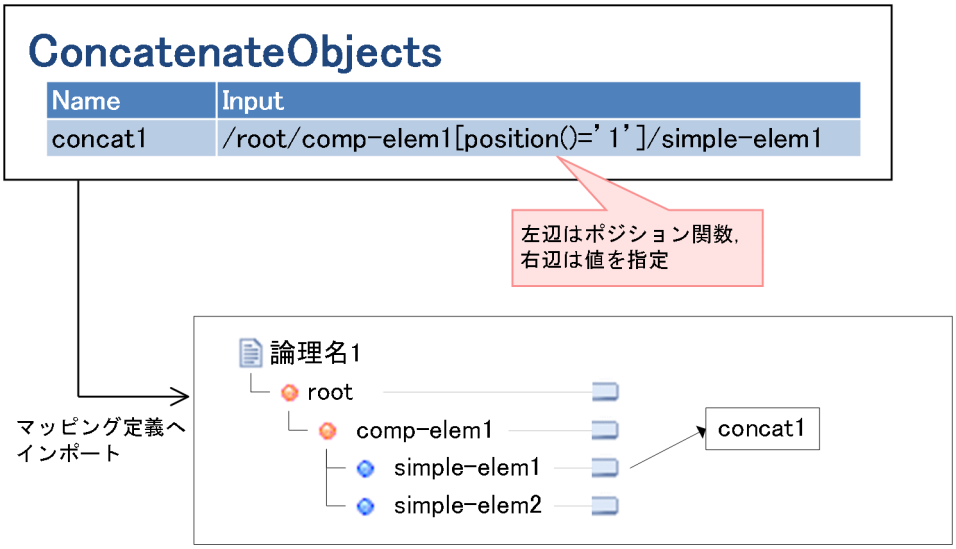
条件式の項目	指定できる値
左辺	• 条件設定で指定できる変換元ノード 変換元ノードは相対パスでも指定できます。※ • 条件設定で指定できるほかのファンクション
演算	「=」，「!=」，「>」，「<」，「>=」，「<=」のどれかを指定できます。
右辺	次のどれかを指定できます。 • アポストロフィ（'）で囲んだ値（空文字は指定できません）。 値にアポストロフィ（'）またはアンパサンド（&）を含める場合，次の例のように，アポストロフィは「'」，アンパサンドは「&」の実体参照の表現にしてください。 （例）「a & 'b'」の場合，「a & 'b'」と定義します。 • 条件設定で指定できる変換元ノード 変換元ノードは相対パスでも指定できます。※ • 条件設定で指定できるほかのファンクション

注※

相対パスの指定方法については，「(7)(c)　相対パス指定」を参照してください。

条件式を設定した場合の定義例を次の図に示します。

図 7-59　定義例（ノード条件設定 2）



(c)　二つの条件の論理積・論理和

「<条件 1><演算><条件 2>」の形式で，2 つの条件を and または or 演算で組み合わせた論理積・論理和で boolean 値を返します。<条件>には次のどれかを指定します。

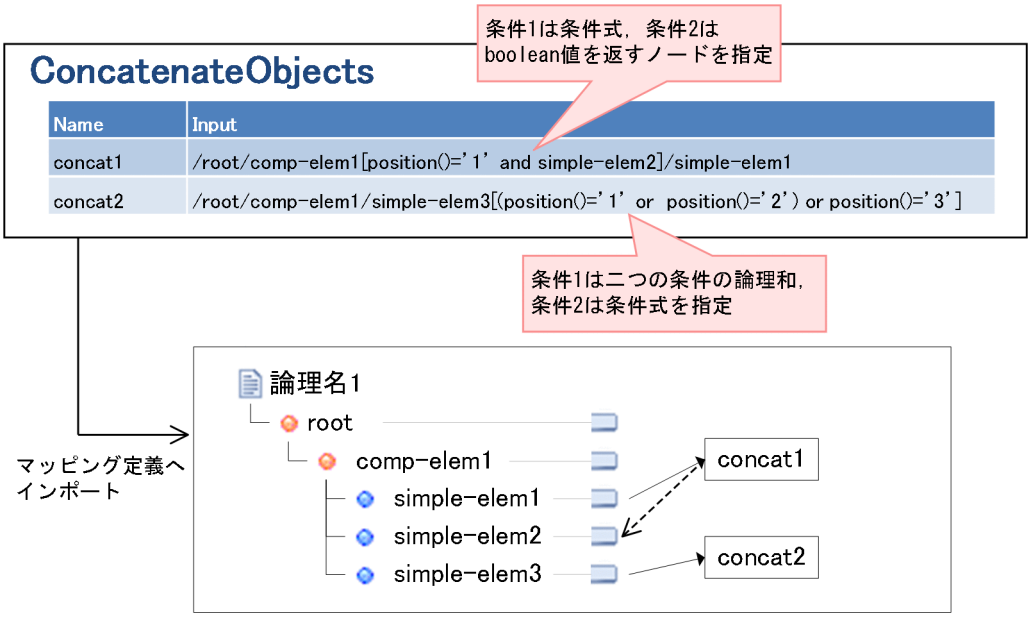
- boolean 値を返すノードまたはファンクション
- 条件式
- 二つの条件の論理積・論理和

条件に複数の「二つの条件の論理積・論理和」が存在する場合は，論理積が優先されます。ただし，「二つの条件の論理積・論理和」の前後を丸括弧（(　)）で囲んだ場合，その論理式が優先されます。

また，これらの条件の否定をとることもできます。詳細は「(d)　条件の否定をとる」を参照してください。

二つの条件の論理積・論理和を条件に設定した場合の定義例を次の図に示します。

図 7-60　定義例（ノード条件設定 3）

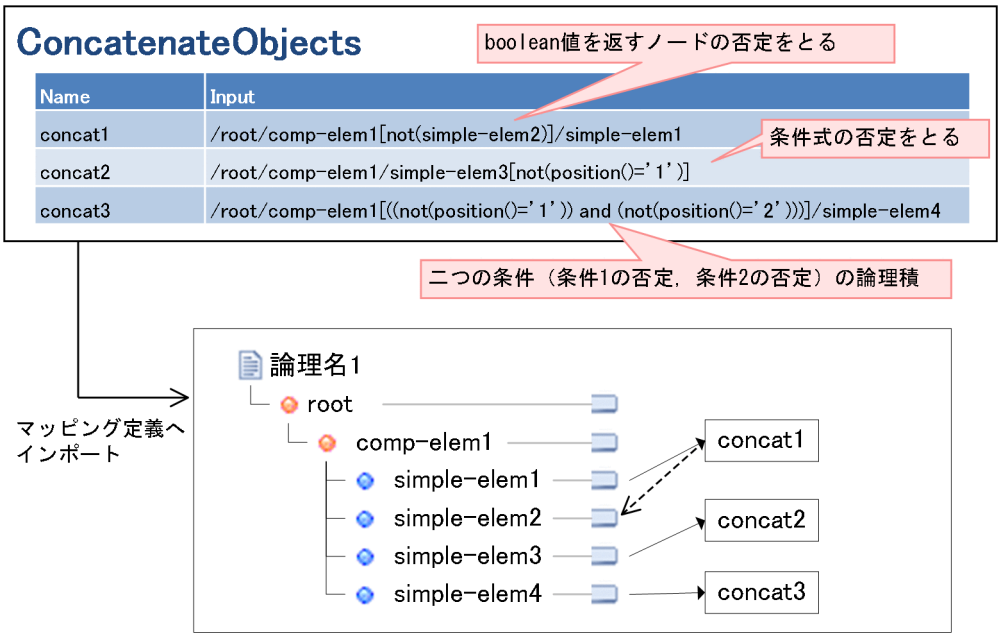


(d)　条件の否定をとる

「not(<条件>)」の形式で，条件の否定をとることができます。

条件の否定をとる場合の定義例を次の図に示します。

図 7-61　定義例（ノード条件設定 4）



(7)　ノードパスの設定

変換元ノードまたは変換先ノードのパスを指定する場合の，論理名および名前空間の指定方法について説明します。

また，ノード条件設定で変換元ノードの相対パスを指定する方法についても説明します。

(a)　論理名の指定

変換元ノードおよび変換先ノードの論理名は，「"$"＋論理名」の形式でルート要素の前に指定します。

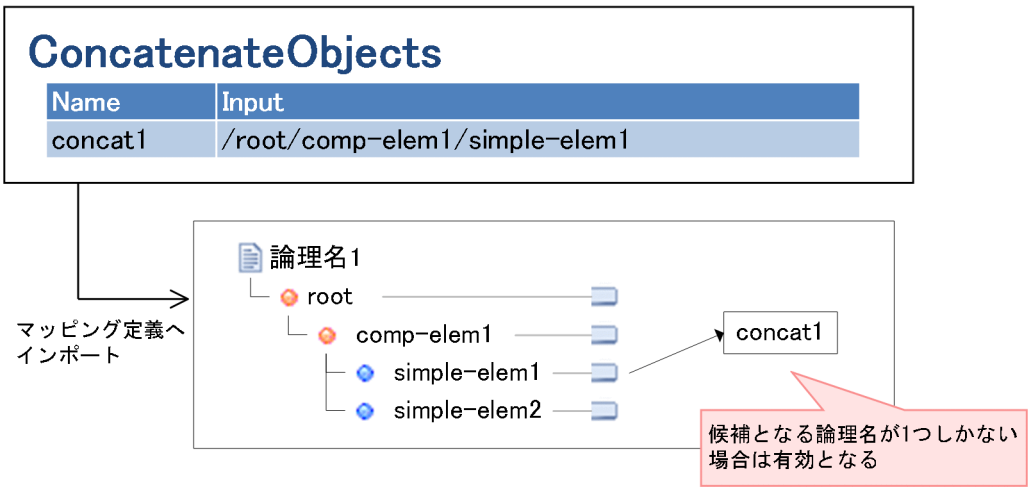
データ変換定義画面の変換元ノードが複数ある場合，変換元ノードの論理名指定は必須です。変換元ノードが 1 つしかない場合，論理名の指定は任意です。

なお，変換先ノードでは論理名の指定は任意です。

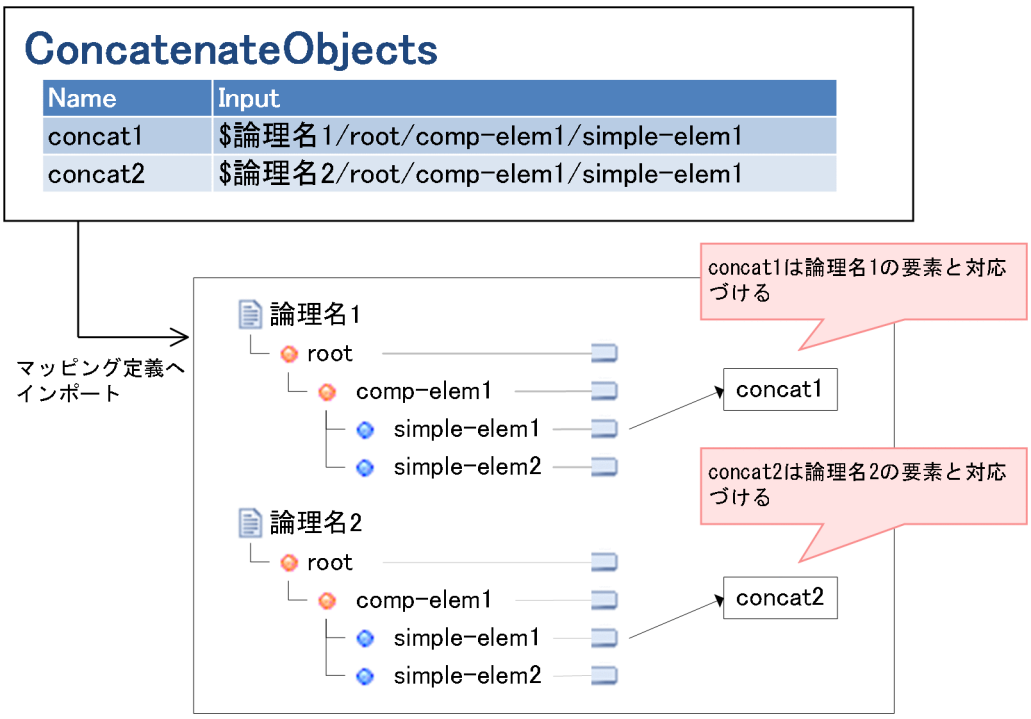
定義例を次の図に示します。

図 7-62　定義例（ノードパスの設定 1）

≪論理名の指定がない場合≫



≪論理名の指定がある場合≫



(b)　名前空間の指定

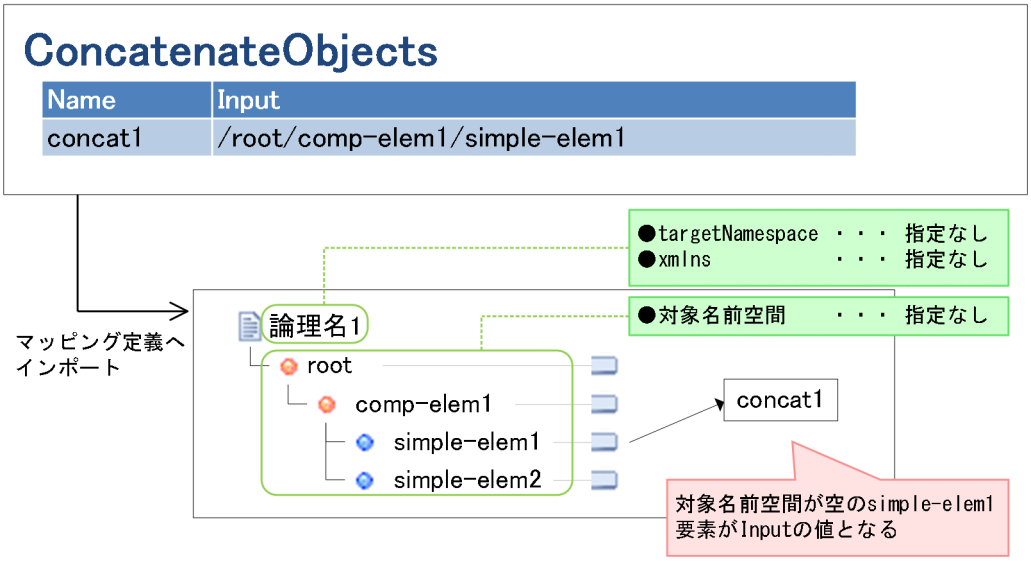
変換元ノードおよび変換先ノードのパス中の各要素名には，Namespaces に定義した名前空間接頭辞を使用できます。名前空間接頭辞の定義方法は「7.15.1　Namespaces（名前空間情報）」を参照してください。

名前空間接頭辞は，「名前空間接頭辞＋":"＋要素名」の形式で指定します。

定義例を次の図に示します。

図 7-63　定義例（ノードパスの設定 2）

≪名前空間の指定がない場合≫



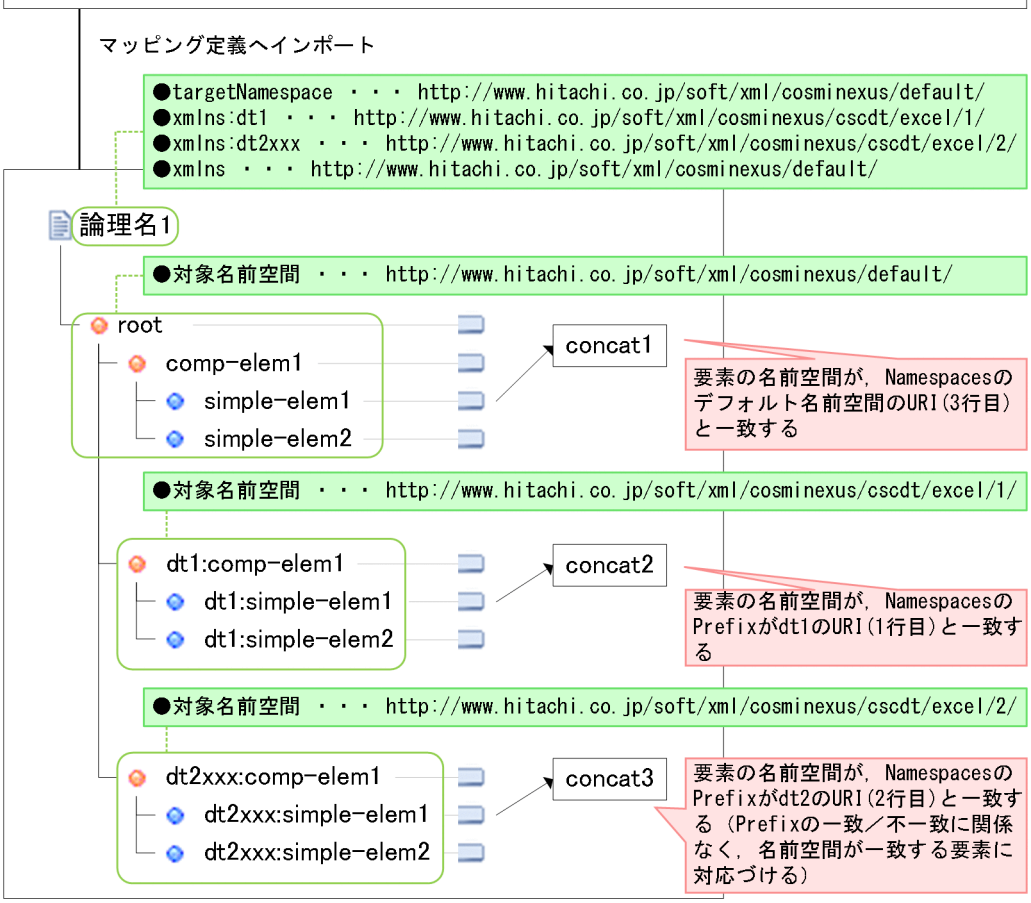
≪名前空間の指定がある場合≫

Namespaces

Prefix	URI
dt1	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/1/
dt2	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/2/
#default	http://www.hitachi.co.jp/soft/xml/cosminexus/default/

ConcatenateObjects

Name	Input
concat1	/root/comp-elem1/simple-elem1
concat2	/root/dt1:comp-elem1/dt1:simple-elem1
concat3	/root/dt1:comp-elem1/dt2:simple-elem1



(c) 相対パス指定

ノード条件を設定する場合，変換元ノードのパスは，条件設定対象のノードからの相対パスで指定できます。相対パスは，次の表に示すノードについて指定できます。それ以外の記述方式（「..」や「//」など）は使用できません。

表 7-24　相対パスの指定方法

相対パスで指定するノード	指定方法
条件設定の対象となるノード	ピリオド（.）を指定します。 指定例 変換元ノードのパスが"/root/comp-elem1/simple-elem1"で，条件設定の対象が"simple-elem1"の場合，"simple-elem1"の有無を条件にするときは"/root/comp-elem1/simple-elem1[.]"と指定します。
条件設定の対象となるノードの子孫ノード	条件設定の対象となるノードの，子ノードからのパスを指定します。 指定例 変換元ノードのパスが"/root/comp-elem1/simple-elem1"の場合，条件設定の対象が"comp-elem1"で，"simple-elem1"の有無を条件にするときは"/root/comp-elem1[simple-elem1]/simple-elem1"と指定します。 条件設定の対象が"root"で，"simple-elem1"の有無を条件にするときは"/root[comp-elem1/simple-elem1]/comp-elem1/simple-elem1"と指定します。

7.13.3　マッピング定義のインポート

表形式 XML ファイルをデータ変換定義画面にインポートする手順を次に示します。

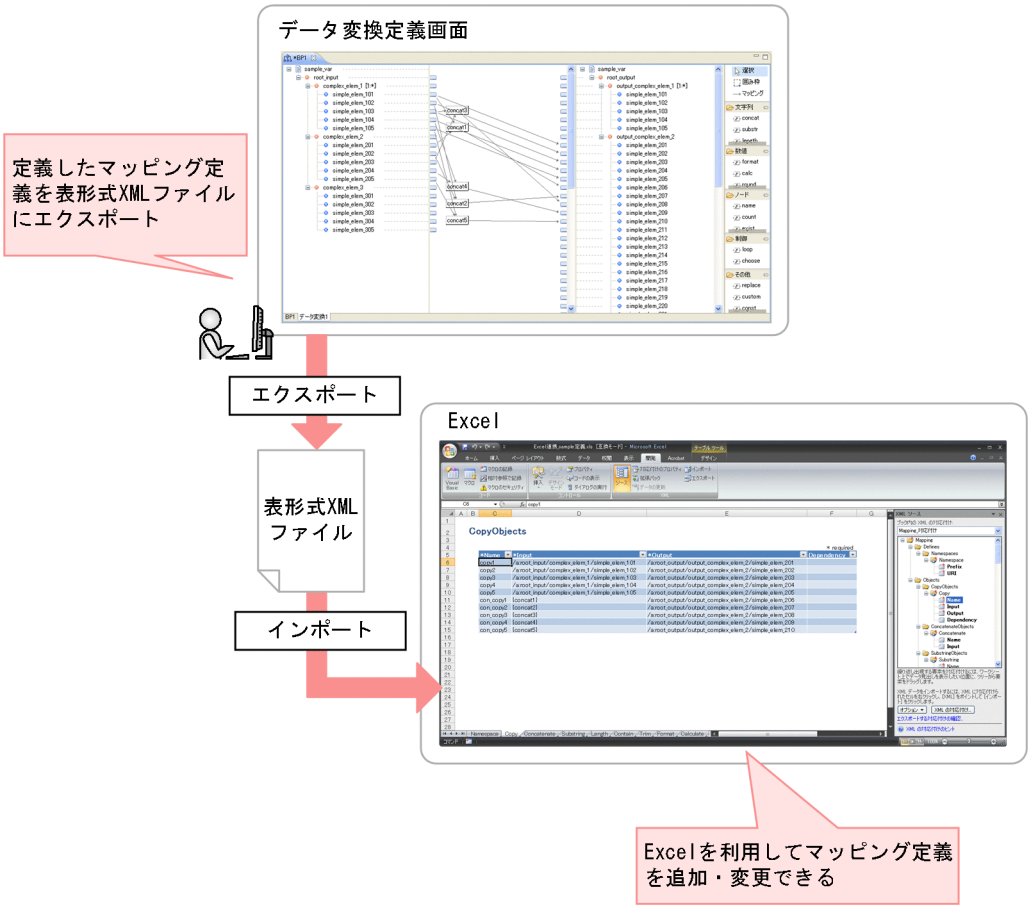
1. 変換元スキーマおよび変換先スキーマを選択して，データ変換定義画面を起動します。
2. データ変換定義画面のマッピングビューア上で右クリックして，［マッピング定義のインポート］を選択します。
   ［マッピング定義のインポート］画面が表示されます。
3. ［参照］ボタンをクリックします。
   ［表形式 XML ファイルの指定］ダイアログが表示されます。
4. インポートする XML ファイルを選択して，［開く］ボタンをクリックします。
   選択した XML ファイルが設定され，［表形式 XML ファイルの指定］ダイアログが閉じます。
5. ［終了］ボタンをクリックします。
   データ変換定義画面にすでにファンクションやマッピング線が定義されている場合は，確認メッセージが表示されます。問題がなければ［はい］ボタンをクリックします。
   Excel で定義したファンクションとマッピング線が，データ変換定義画面に反映されます。

7.14　Excel を利用したマッピング定義のエクスポート

サービスプラットフォームでは，開発環境のデータ変換定義画面で定義したマッピングを表形式 XML ファイルにエクスポートできます。エクスポートした表形式 XML ファイルを Excel にインポートすることで，効率良くマッピング定義の追加や変更ができます。

マッピング定義のエクスポートの流れを次に示します。

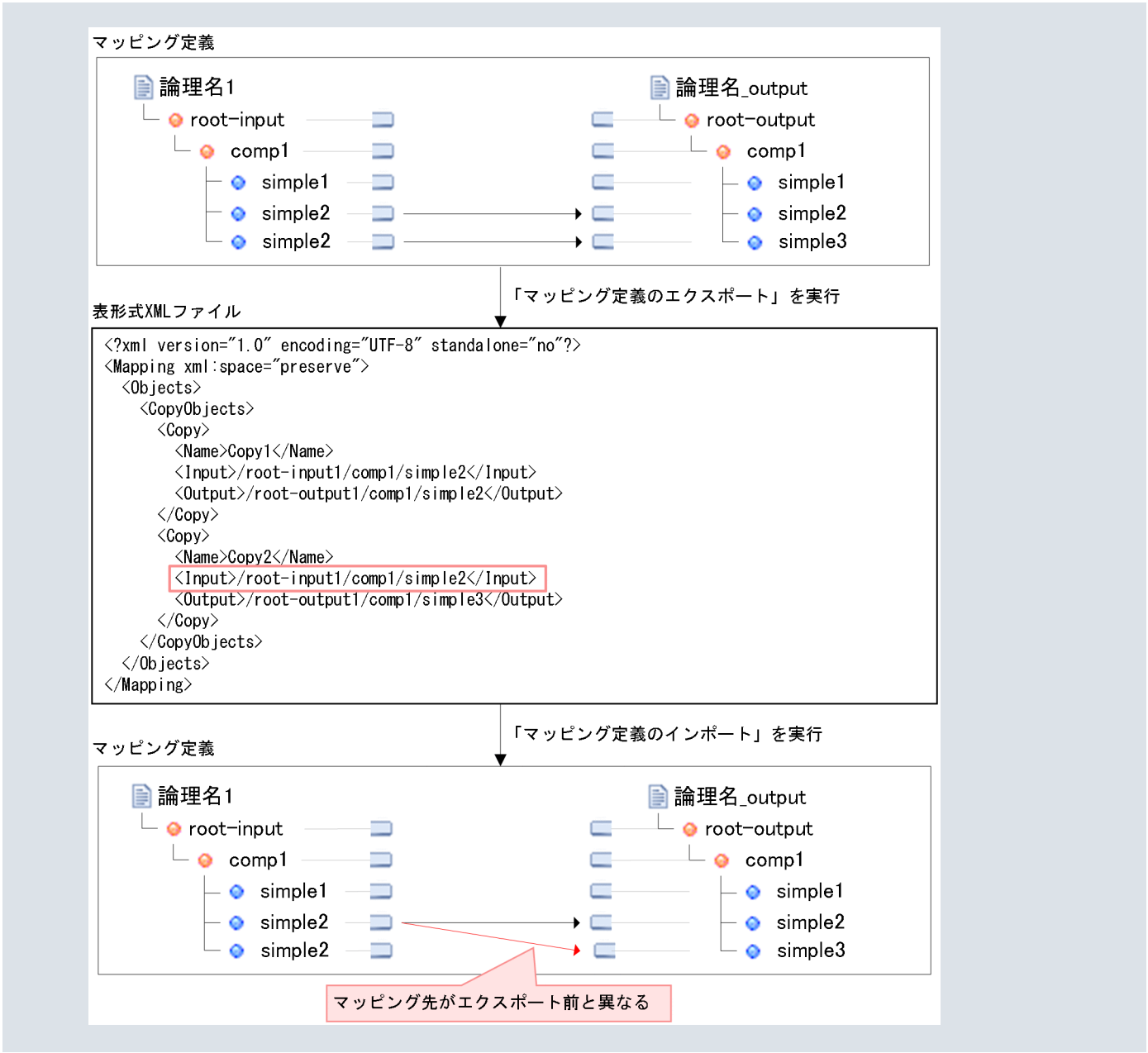
図 7-64　マッピング定義のエクスポートの流れ



注意事項

エクスポートしたマッピング定義には次の制限があります。

- データ変換定義画面のファンクションの位置情報は保持されません。
- 変換ファンクション定義ファイルで設定したカスタムファンクションのコメントは保持されません。
- 1 つの階層に同じ要素が複数ある場合，かつ 2 つ目以降の要素にマッピング線が引かれているとき，エクスポートしたマッピング定義をデータ変換定義画面に再度インポートすると，マッピング先が異なる場合があります。この例を次の図に示します。



7.14.1　マッピング定義のエクスポート

データ変換定義画面からマッピング定義をエクスポートする手順を次に示します。

1. 変換元スキーマおよび変換先スキーマを選択して，データ変換定義画面を起動します。
2. データ変換定義画面のマッピングビューア上で右クリックして，［マッピング定義のエクスポート］を選択します。
   ［マッピング定義のエクスポート］画面が表示されます。
3. エクスポート先および XML ファイル名を指定して，［保存］ボタンをクリックします。
   指定した名称で XML ファイルが保存され，完了メッセージが表示されます。

4. [OK] ボタンをクリックします。
   マッピング定義が表形式 XML ファイルとしてエクスポートされます。

7.14.2 表形式 XML ファイルのインポート

表形式 XML ファイルを Excel にインポートする手順について説明します。

1. Microsoft Office Excel を起動します。

2. [開発] タブの [ソース] ボタンを選択します。
   [XML ソース] 作業ウィンドウが表示されます。

3. [XML ソース] 作業ウィンドウの [XML の対応付け...] ボタンをクリックします。
   [XML の対応付け] ダイアログが表示されます。

4. [追加] ボタンをクリックし，表形式 XML スキーマ定義ファイル（dt_import_mapping.xsd）を追加します。
   表形式 XML スキーマ定義ファイル（dt_import_mapping.xsd）の格納先は「<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥dt_import」です。
   [XML ソース] 作業ウィンドウにスキーマがツリー表示されます。

5. 「***Objects」または「Namespaces」の要素部分をセルにドラッグ＆ドロップします。※
   ドラッグ＆ドロップした要素部分のデータを編集できるようになります。

   注※
   要素を対応づけるときは，オブジェクト（Copy や Concatenate など）ごとにシートを分けてください。同じシートに複数の要素を設定すると，マッピング定義を正しくインポートできません。

6. [開発] タブの [インポート] ボタンをクリックし，表形式 XML ファイルを XML 要素の対応付けをしたセルにインポートします。
   表形式 XML ファイルが Excel にインポートされます。インポートした表形式 XML ファイルを基に，マッピング定義を追加および変更できます。
   マッピング定義の設定方法については，「7.13.2 マッピング定義の設定」を参照してください。

注意事項

Excel のインポート機能では，Excel からエクスポートした表形式 XML ファイルをインポートした場合，表形式 XML ファイルの各項目値の前後のホワイトスペース（半角空白，タブ，ラインフィード，キャリッジリターン）が削除されます。必要に応じて，インポート後に Excel のセルを修正してください。

7.15　表形式 XML スキーマ定義ファイルの定義詳細

Excel のマッピング定義で利用する表形式 XML スキーマ定義ファイルの内容について説明します。

表形式 XML スキーマ定義ファイルの定義一覧を次の表に示します。

表 7-25　表形式 XML スキーマ定義ファイルの定義一覧

大分類	中分類	説明
Defines	－	オブジェクトを定義するための前提となる情報を定義します。
	Namespaces	データ変換定義画面上の変換元ノードまたは変換先ノードを示すパスの定義で使用する，名前空間情報を定義します。
Objects※	－	ファンクションやマッピング線などを定義します。
	CopyObjects	変換先ノードへのマッピング線を定義します。
	ConcatenateObjects	文字列連結ファンクションを定義します。
	SubstringObjects	部分文字列取得ファンクションを定義します。
	LengthObjects	文字数取得ファンクションを定義します。
	ContainObjects	文字列調査ファンクションを定義します。
	TrimObjects	空白除去ファンクションを定義します。
	FormatObjects	数値フォーマット変換ファンクションを定義します。
	CalculateObjects	数値演算ファンクションを定義します。
	RoundObjects	端数処理ファンクションを定義します。
	SumObjects	総和ファンクションを定義します。
	NotObjects	否定演算ファンクションを定義します。
	BitOpObjects	論理演算ファンクションを定義します。
	ShiftObjects	シフト演算ファンクションを定義します。
	NameObjects	ノード名取得ファンクションを定義します。
	CountObjects	ノード数取得ファンクションを定義します。
	ExistObjects	ノード調査ファンクションを定義します。
	LoopObjects	くり返しファンクションを定義します。
	ChooseObjects	選択ファンクションを定義します。
	ReplaceObjects	値置換えファンクションを定義します。
	RadixObjects	基数変換ファンクションを定義します。
	CustomObjects	カスタムファンクションを定義します。
	ConstantObjects	定数ファンクションを定義します。

大分類	中分類	説明
Objects※	XpathObjects	XPath ファンクションを定義します。

（凡例）

－：該当しません。

注※

CopyObjects と CopyObjects 以外の定義範囲は次のように異なります。

CopyObjects：

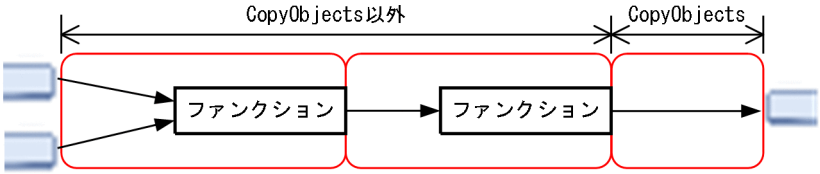
変換元ノードまたはファンクションから変換先ノードへのマッピング線が定義範囲です。

CopyObjects 以外：

ファンクション単体，または変換元ノード／入力側ファンクションと連結するマッピング線が定義範囲です。

CopyObjects と CopyObjects 以外の定義範囲を次の図に示します。

図 7-65　CopyObjects と CopyObjects 以外の定義範囲



以降で各定義の詳細を説明します。

7.15.1　Namespaces（名前空間情報）

オブジェクトの Input 項目など，マッピング定義エディタ上の変換元ノードまたは変換先ノードを示すパスの定義で使用する，名前空間情報を定義します。Namespaces に定義した Prefix 項目が，オブジェクトの Input 項目などに記述された要素の名前空間接頭辞として使用された場合，その名前空間接頭辞がどの名前空間に属するかを，Namespaces に定義された情報で判断します。

Prefix 項目が"#default"の定義は，オブジェクトの Input 項目などに記述された要素のデフォルト名前空間（名前空間接頭辞がない場合に属する名前空間）を示します。Prefix 項目が"#default"の定義をしない場合，デフォルト名前空間は空とします。

URI 項目が空の場合，該当する名前空間接頭辞が付いた要素の名前空間は，空とします。

(1)　定義項目

定義項目を次の表に示します。

表 7-26　Namespaces 定義

項目名	定義内容	定義できる値	補足説明
Prefix	名前空間接頭辞	NCName 型の値または"#default"	必須項目です。重複して同じ Prefix を定義できません。 "#default"の場合 デフォルト名前空間を設定します。
URI	名前空間	anyURI 型の値	－

（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-66　Namespaces 定義例 1

≪Prefixに"#default"の指定がある場合≫

Namespaces

Prefix	URI
dt	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/
#default	http://www.hitachi.co.jp/soft/xml/cosminexus/default/

●オブジェクトのInput項目などのパス定義に"/root/dt:sample1"と定義された場合

root要素・・・名前空間が"http://www.hitachi.co.jp/soft/xml/cosminexus/default/"の要素を示す。
sample1要素・・・名前空間が"http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/"の要素を示す。

図 7-67　Namespaces 定義例 2

≪Prefixに"#default"の指定がない場合≫

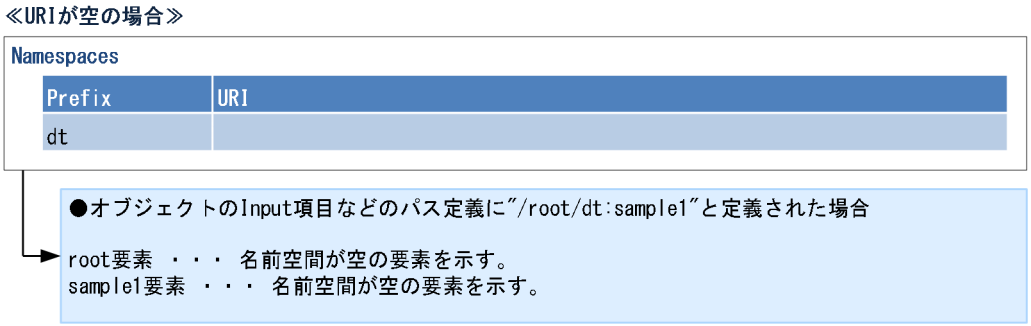
Namespaces

Prefix	URI
dt	http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/

●オブジェクトのInput項目などのパス定義に"/root/dt:sample1"と定義された場合

root要素　・・・　名前空間が空の要素を示す。
sample1要素　・・・　名前空間が"http://www.hitachi.co.jp/soft/xml/cosminexus/cscdt/excel/"の要素を示す。

図 7-68　Namespaces 定義例 3



7.15.2　CopyObjects（変換先ノードへのマッピング）

ほかのオブジェクトで定義したファンクションや変換元ノードから，変換先ノードへのマッピング線を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-27　CopyObjects 定義

項目名	定義内容	定義できる値	補足説明
Name	マッピング名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。 マッピング定義では使用されない項目です。
Input	入力	「7.13.2　マッピング定義の設定」を参照	必須項目です。
Output	出力	「7.13.2　マッピング定義の設定」を参照	必須項目です。
Dependency	依存対象	「7.13.2　マッピング定義の設定」を参照	依存対象にできる LoopObjects を複数定義している場合は，必須です。依存対象にできる LoopObjects が 1 つの場合は，その LoopObjects を依存対象として設定します。LoopObjects の定義については，「7.15.18 LoopObjects（くり返しファンクション）」を参照してください。

(2)　定義例

定義例を次の図に示します。

図 7-69　CopyObjects 定義例 1

≪依存対象にできるLoopObjectsがない場合≫

CopyObjects

Inputに変換元ノードを指定

Name	Input	Output	Dependency
copy1	/root-input/comp-elem1/simple-elem1	/root-output/comp-elem1/simple-elem1	
copy2	{concat1}	/root-output/comp-elem1/simple-elem2	

Inputにファンクションを指定

ConcatenateObjects

Name	Input
concat1	/root-input/comp-elem1/simple-elem2

マッピング定義へインポート

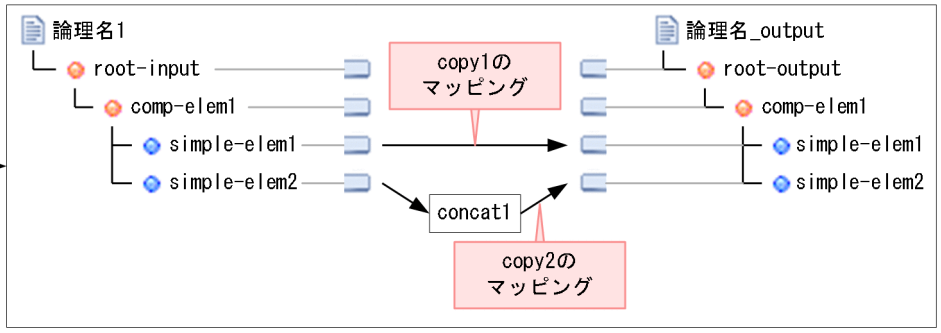


図 7-70　CopyObjects 定義例 2

≪依存対象にできるLoopObjectsが1つだけある場合≫

CopyObjects

Name	Input	Output	Dependency
copy1	/root-input/comp-elem1/simple-elem1	/root-output/comp-elem1/simple-elem1	{loop1}
copy2	/root-input/comp-elem1/simple-elem2	/root-output/comp-elem1/simple-elem2	
copy3	{loop1}	/root-output/comp-elem1	

LoopObjects

Name	Input	RelationalPath	SortKey	SortOrder	SortLanguage	SortDataType	SortCase
loop1	/root-input/comp-elem1						

マッピング定義へインポート

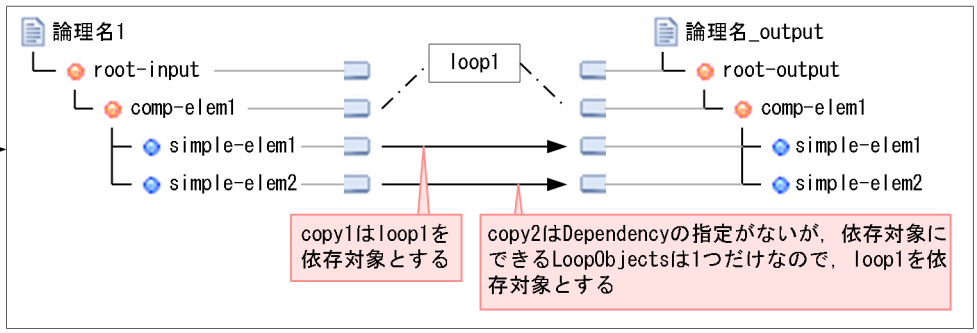
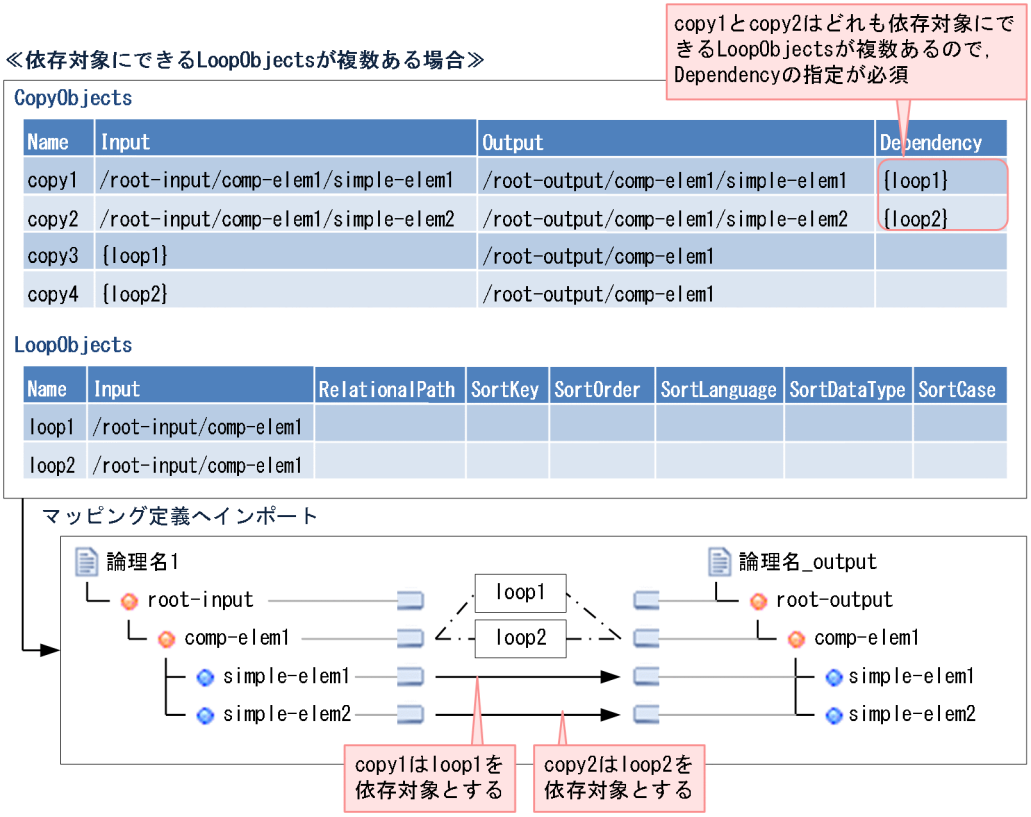


図 7-71　CopyObjects 定義例 3



7.15.3　ConcatenateObjects（文字列連結ファンクション）

文字列連結ファンクション（concat）を定義します。

(1)　定義項目

定義項目を次の表に示します。

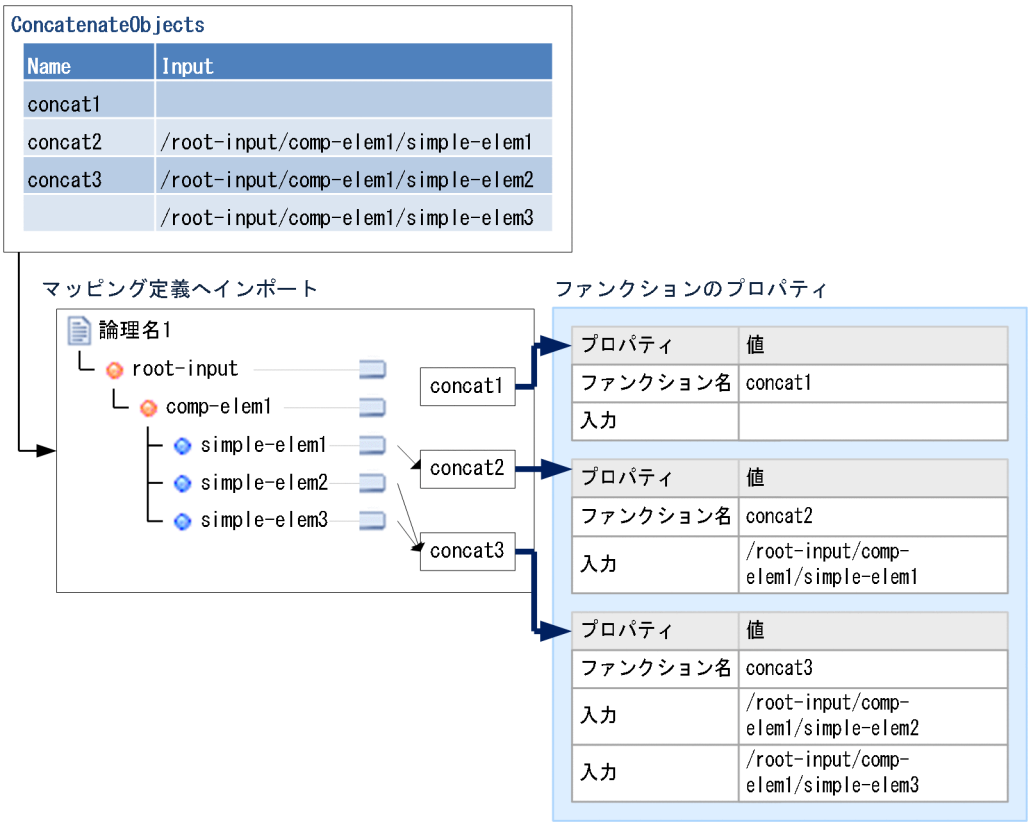
表 7-28　ConcatenateObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	定義できる値については，「7.15.25　同一の Name 要素を複数行に定義できるオブジェクト」を参照してください。
Input	入力	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。 値は，定義された順にリストに設定します。

(2)　定義例

定義例を次の図に示します。

図 7-72　ConcatenateObjects 定義例



7.15.4　SubstringObjects（部分文字列取得ファンクション）

部分文字列取得ファンクション（substr）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-29　SubstringObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－
Acquisition	指定方法	"left"，"right"，"division" のどれか	必須項目です。 "left"の場合 指定方法は「先頭からの範囲指定」を設定します。

項目名	定義内容	定義できる値	補足説明
Acquisition	指定方法	"left"，"right"，"division"のどれか	"right"の場合 　指定方法は「末尾からの範囲指定」を設定します。 "division"の場合 　指定方法は「分割文字列指定」を設定します。
Start	開始位置	1～2,147,483,647 の整数	Acquisition が"left"または"right"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。
Count	文字数	0～2,147,483,647 の整数または空	Acquisition が"left"または"right"で Count が空の場合，「開始位置から以降全て」に設定します。 Acquisition が"left"および"right"以外の場合，設定値を取り込みません。
String	分割文字列	1,024 文字以内の文字列（改行は定義できない）	Acquisition が"division"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。
Part	取得部位	"pre"または"post"	Acquisition が"division"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。 "pre"の場合 　取得部位は「前」を設定します。 "post"の場合 　取得部位は「後」を設定します。

（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-73　SubstringObjects 定義例 1

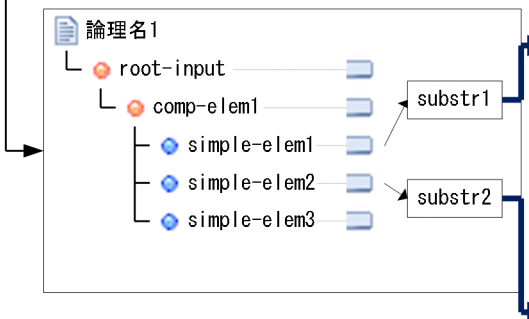
≪Acquisitionが"left"の場合≫

SubstringObjects

Name	Input	Acquisition	Start	Count	String	Part
substr1	/root-input/comp-elem1/simple-elem1	left	1			
substr2	/root-input/comp-elem1/simple-elem2	left	10	8	$$$	pre

"division"以外の場合は無視

マッピング定義へインポート



ファンクションのプロパティ

プロパティ	値
ファンクション名	substr1
入力	/root-input/comp-elem1/simple-elem1
指定方法	先頭からの範囲指定
開始位置	1
文字数	開始位置以降全て

プロパティ	値
ファンクション名	substr2
入力	/root-input/comp-elem1/simple-elem2
指定方法	先頭からの範囲指定
開始位置	10
文字数	8

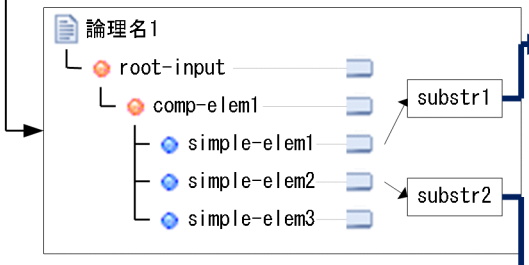
図 7-74　SubstringObjects 定義例 2

≪Acquisitionが"right"の場合≫

SubstringObjects

Name	Input	Acquisition	Start	Count	String	Part
substr1	/root-input/comp-elem1/simple-elem1	right	1			
substr2	/root-input/comp-elem1/simple-elem2	right	15	20		

マッピング定義へインポート

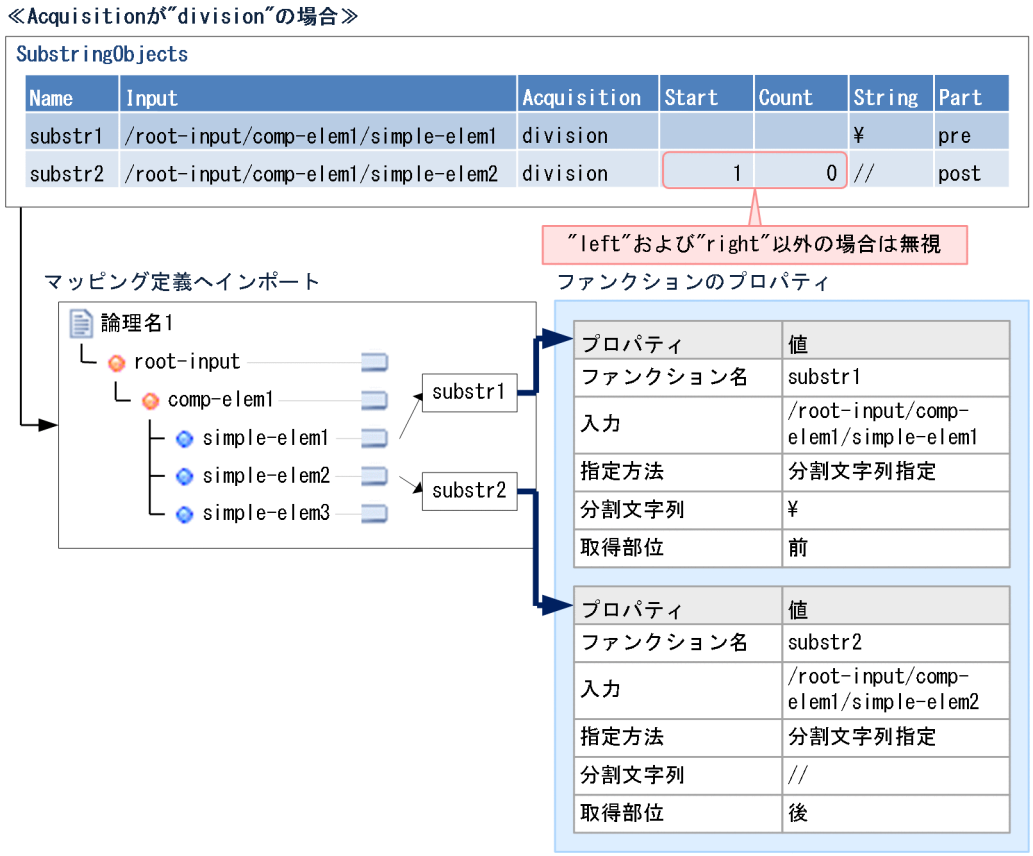


ファンクションのプロパティ

プロパティ	値
ファンクション名	substr1
入力	/root-input/comp-elem1/simple-elem1
指定方法	末尾からの範囲指定
開始位置	1
文字数	開始位置以降全て

プロパティ	値
ファンクション名	substr2
入力	/root-input/comp-elem1/simple-elem2
指定方法	末尾からの範囲指定
開始位置	15
文字数	20

図 7-75　SubstringObjects 定義例 3



7.15.5　LengthObjects（文字数取得ファンクション）

文字数取得ファンクション（length）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-30　LengthObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

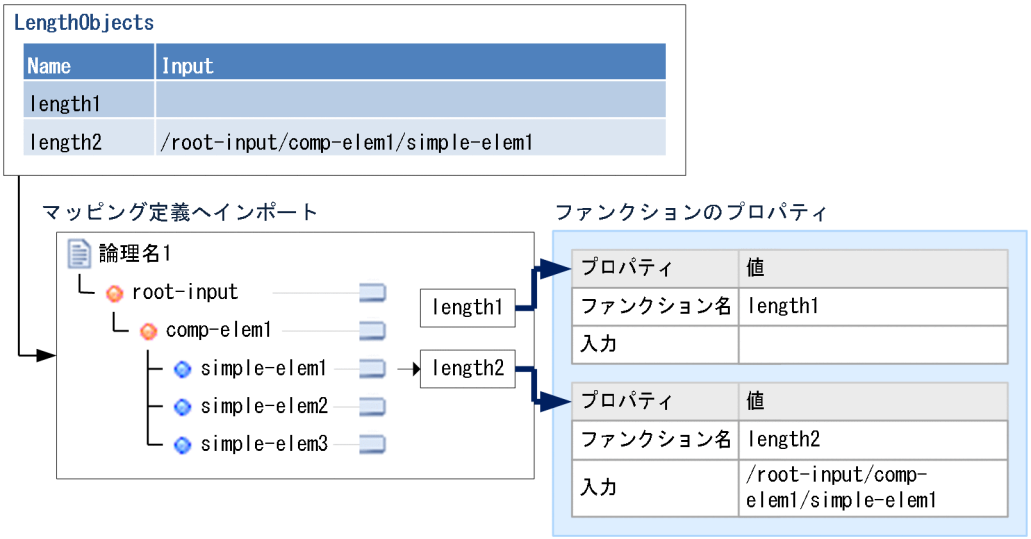
（凡例）

－：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-76 LengthObjects 定義例



7.15.6 ContainObjects（文字列調査ファンクション）

文字列調査ファンクション（contain）を定義します。

(1) 定義項目

定義項目を次の表に示します。

表 7-31 ContainObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2 マッピング定義の設定」を参照	－
Check	調査種別	"include"または"start"	必須項目です。 "include"の場合 調査種別は「指定文字列を含む」を設定します。 "start"の場合 調査種別は「指定文字列から開始」を設定します。

項目名	定義内容	定義できる値	補足説明
String	調査対象文字列	1,024 文字以内の文字列（改行は定義できない）	－

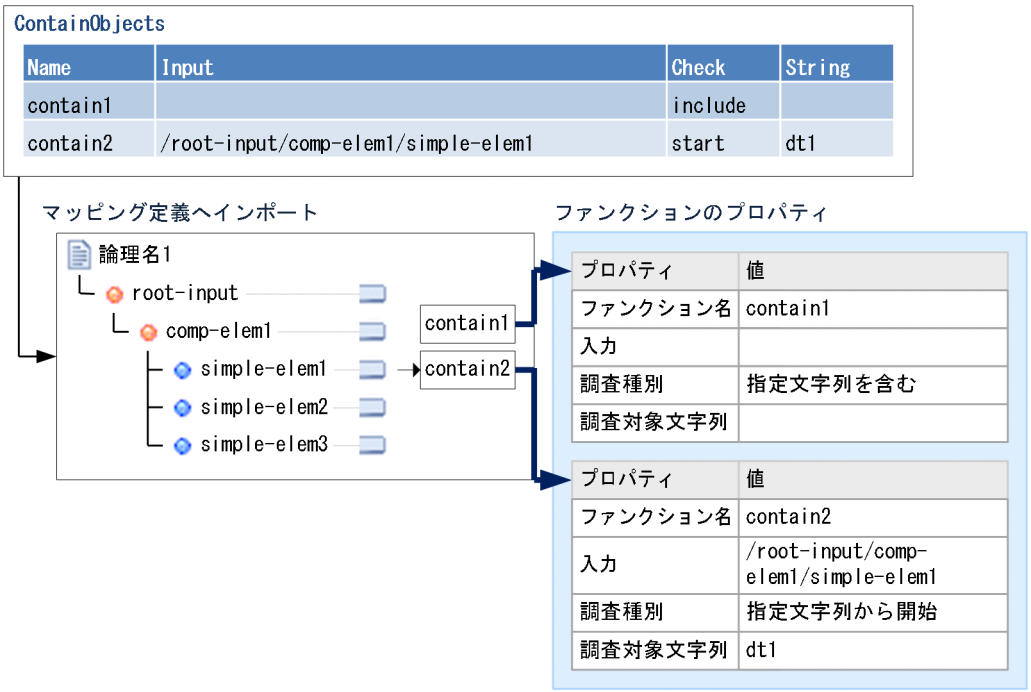
（凡例）

－：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-77 ContainObjects 定義例



7.15.7 TrimObjects（空白除去ファンクション）

空白除去ファンクション（trim）を定義します。

(1) 定義項目

定義項目を次の表に示します。

表 7-32 TrimObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。

項目名	定義内容	定義できる値	補足説明
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

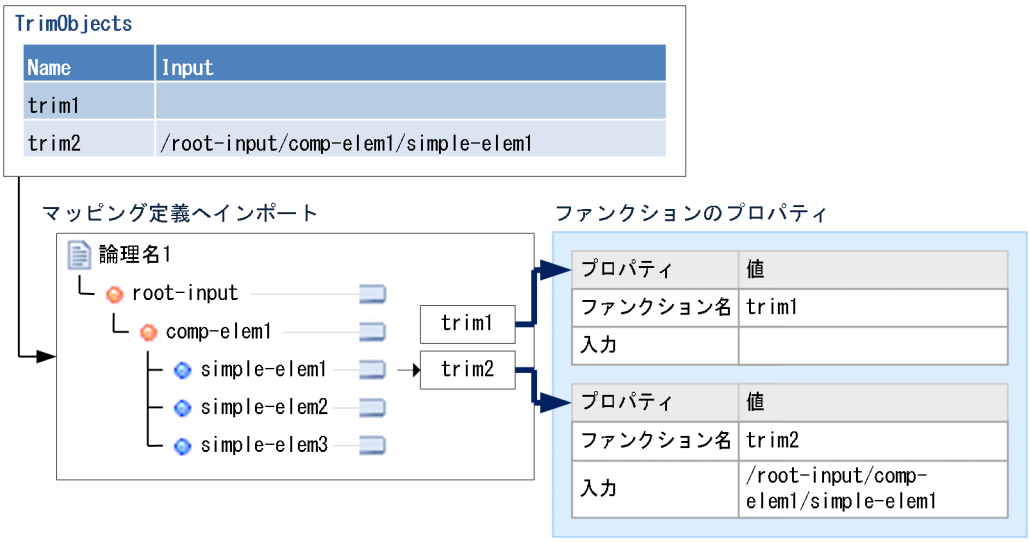
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-78　TrimObjects 定義例



7.15.8　FormatObjects（数値フォーマット変換ファンクション）

数値フォーマット変換ファンクション（format）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-33　FormatObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

項目名	定義内容	定義できる値	補足説明
Pattern	パターン	• 変換後の java.text.DecimalFormat クラスの形式のパターン文字列 • 1,024 文字以内（改行は定義できない）	－
ChangeSymbol	記号の変更	"y"または"n"	必須項目です。 "y"の場合 記号の変更は「有」を設定します。 "n"の場合 記号の変更は「無」を設定します。
DecimalSeparator	小数点文字	1 桁の文字	ChangeSymbol が"y"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。
GroupingSeparator	桁区切り文字	1 桁の文字	ChangeSymbol が"y"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。

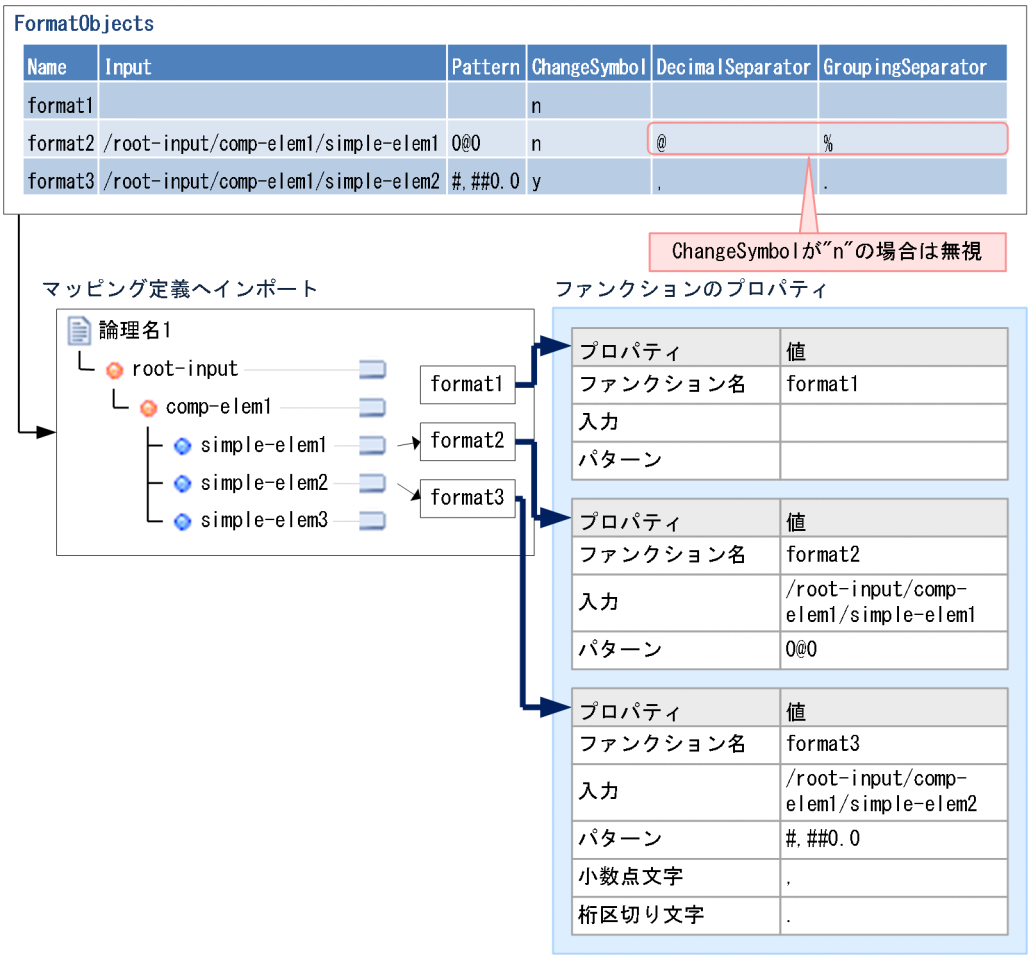
（凡例）

－：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-79　FormatObjects 定義例



7.15.9　CalculateObjects（数値演算ファンクション）

数値演算ファンクション（calc）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-34　CalculateObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input1	入力 1	「7.13.2　マッピング定義の設定」を参照	－
Operation	演算	"+"，"-"，"*"，"/"，"%"のどれか	必須項目です。

項目名	定義内容	定義できる値	補足説明
Input2	入力 2	「7.13.2　マッピング定義の設定」を参照	－

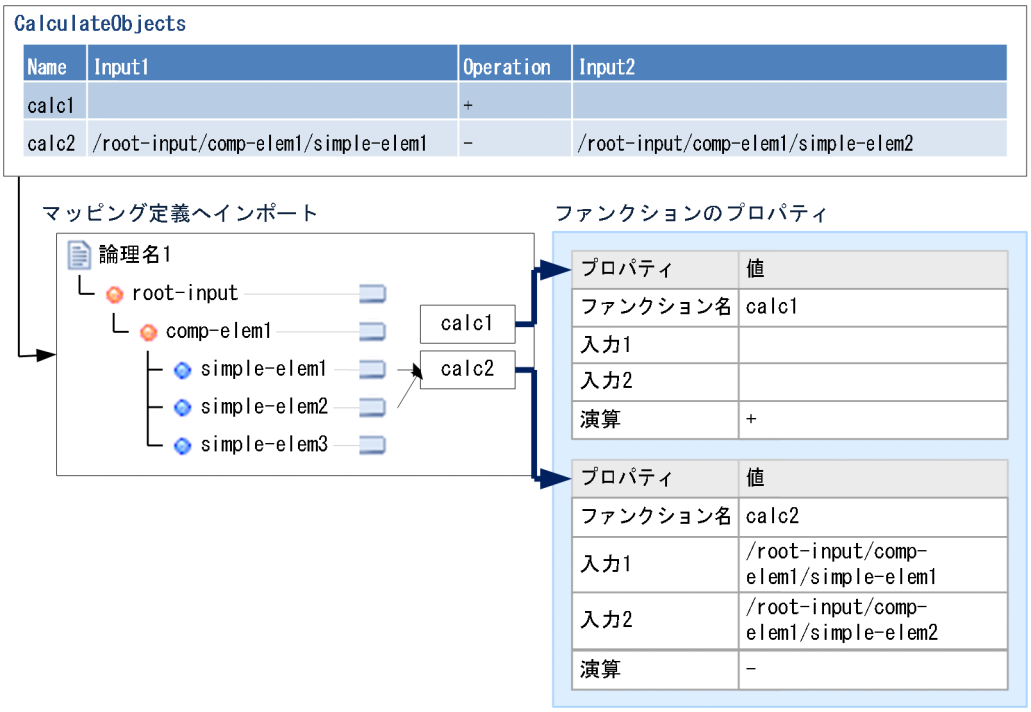
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-80　CalculateObjects 定義例



7.15.10　RoundObjects（端数処理ファンクション）

端数処理ファンクション（round）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-35　RoundObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。

項目名	定義内容	定義できる値	補足説明
Input	入力	「7.13.2　マッピング定義の設定」を参照	－
Type	端数処理種別	"off"，"down"，"up"のどれか	必須項目です。 "off"の場合 端数処理種別は「四捨五入」を設定します。 "down"の場合 端数処理種別は「切り捨て」を設定します。 "up"の場合 端数処理種別は「切り上げ」を設定します。

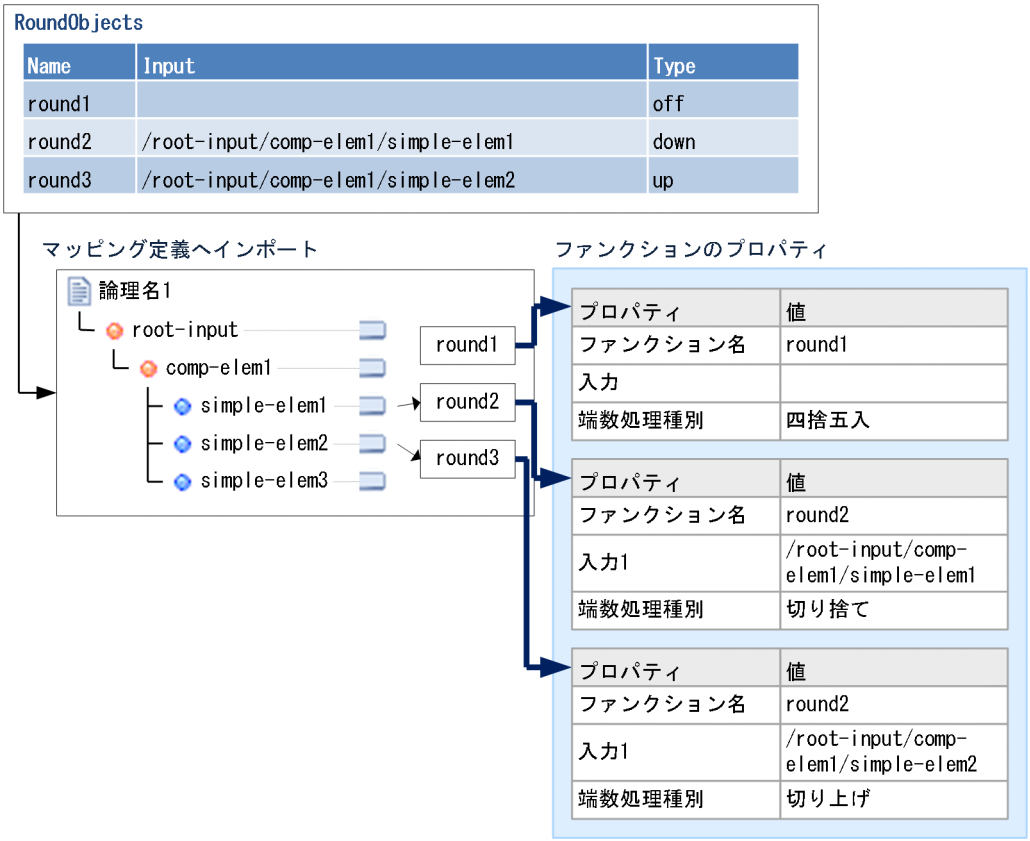
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-81　RoundObjects 定義例



7.15.11　SumObjects（総和ファンクション）

総和ファンクション（sum）を定義します。

(1)　定義項目

定義項目を次の表に示します。

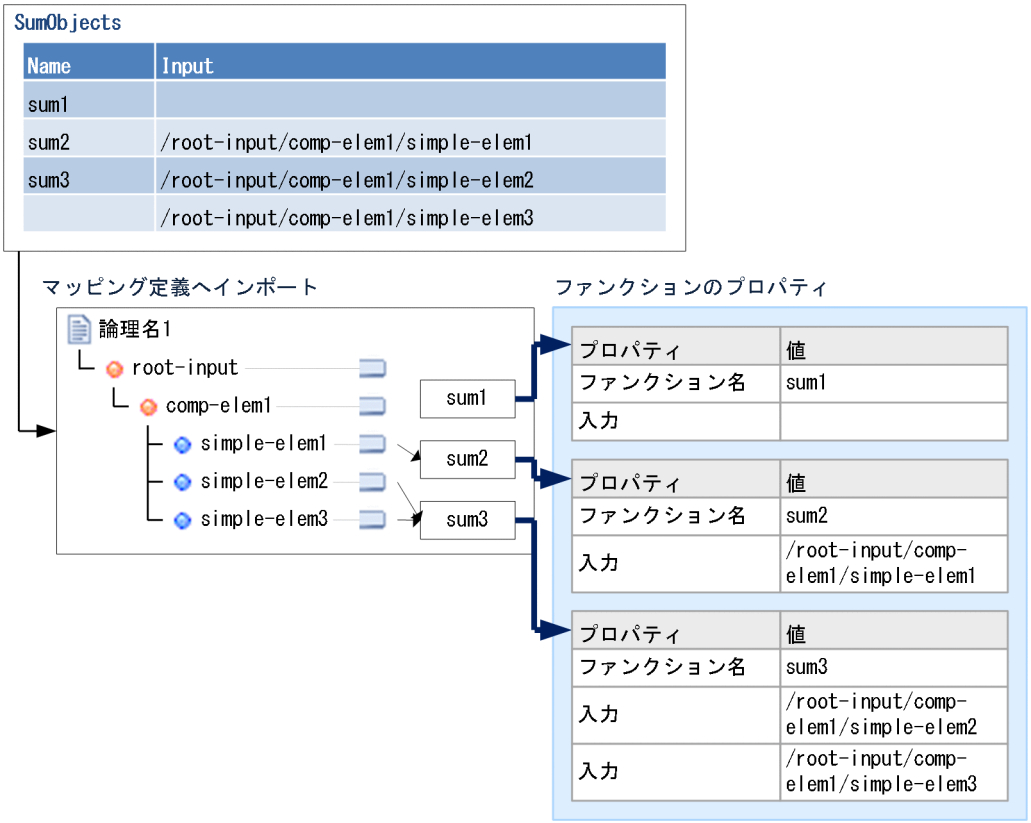
表 7-36　SumObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	定義できる値については，「7.15.25　同一の Name 要素を複数行に定義できるオブジェクト」を参照してください。
Input	入力	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。 ただし，同じ Name 内で同じ Input を複数指定できません。 値は，定義された順にリストに設定します。

(2)　定義例

定義例を次の図に示します。

図 7-82　SumObjects 定義例



7.15.12　NotObjects（否定演算ファンクション）

否定演算ファンクション（not）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-37　NotObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

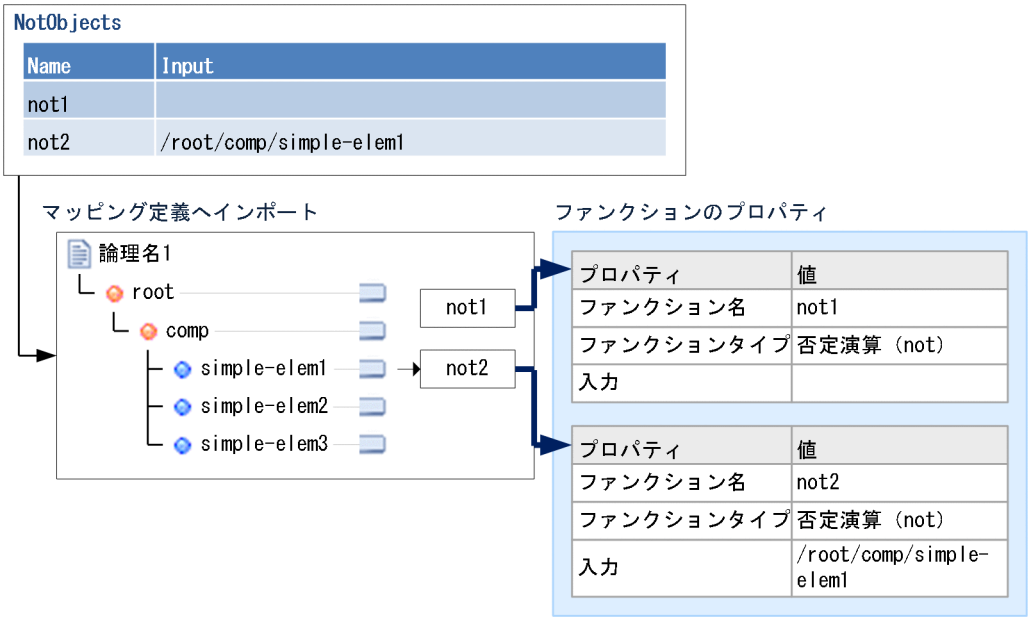
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-83　NotObjects 定義例



7.15.13　BitOpObjects（論理演算ファンクション）

論理演算ファンクション（bitop）を定義します。

(1) 定義項目

定義項目を次の表に示します。

表 7-38 BitOpObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Operation	演算	"and"，"or"，"xor"のどれか	必須項目です。
Input1	入力 1	「7.13.2 マッピング定義の設定」を参照	−
Input2	入力 2	「7.13.2 マッピング定義の設定」を参照	−

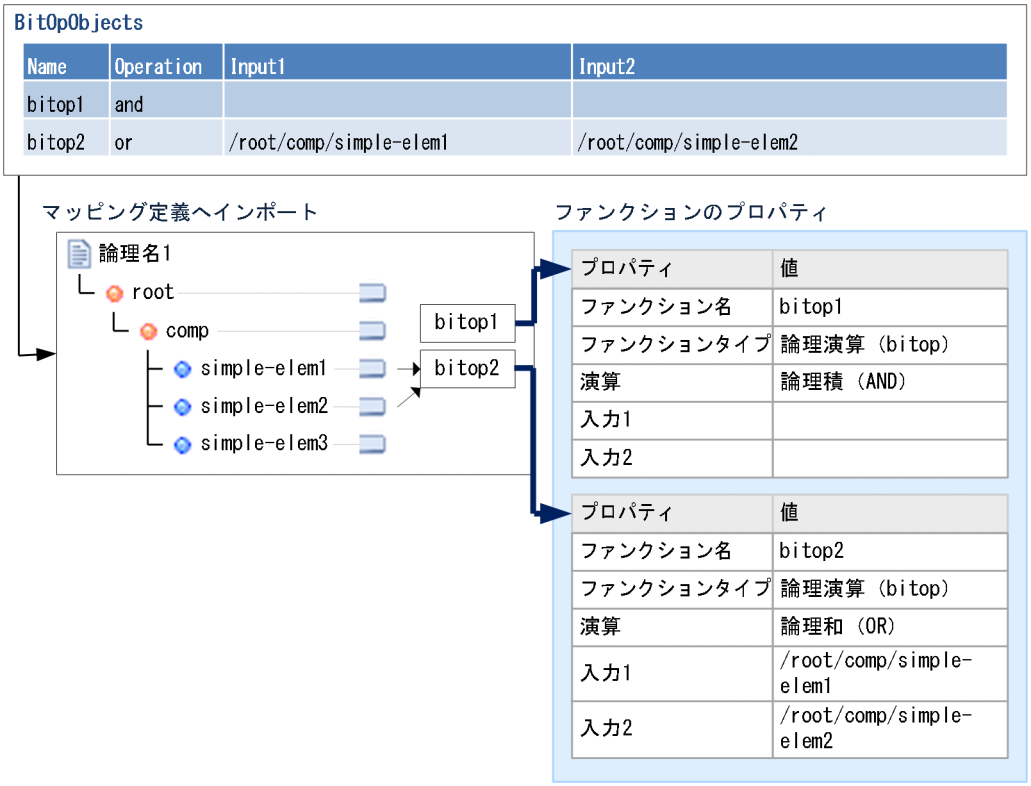
(凡例)

−：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-84 BitOpObjects 定義例



7.15.14　ShiftObjects（シフト演算ファンクション）

シフト演算ファンクション（shift）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-39　ShiftObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－
ShiftAmount	シフト量（ビット）	0～64 の数値	必須項目です。
OutputSize	出力サイズ（ビット）	1～64 の数値	必須項目です。
ShiftDirection	シフト方向	"left"または"right"	必須項目です。
ShiftType	シフト種別	"arithmetic"または"logical"	ShiftDirection が"right"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。 "arithmetic"は，算術シフトです。 "logical"は，論理シフトです。

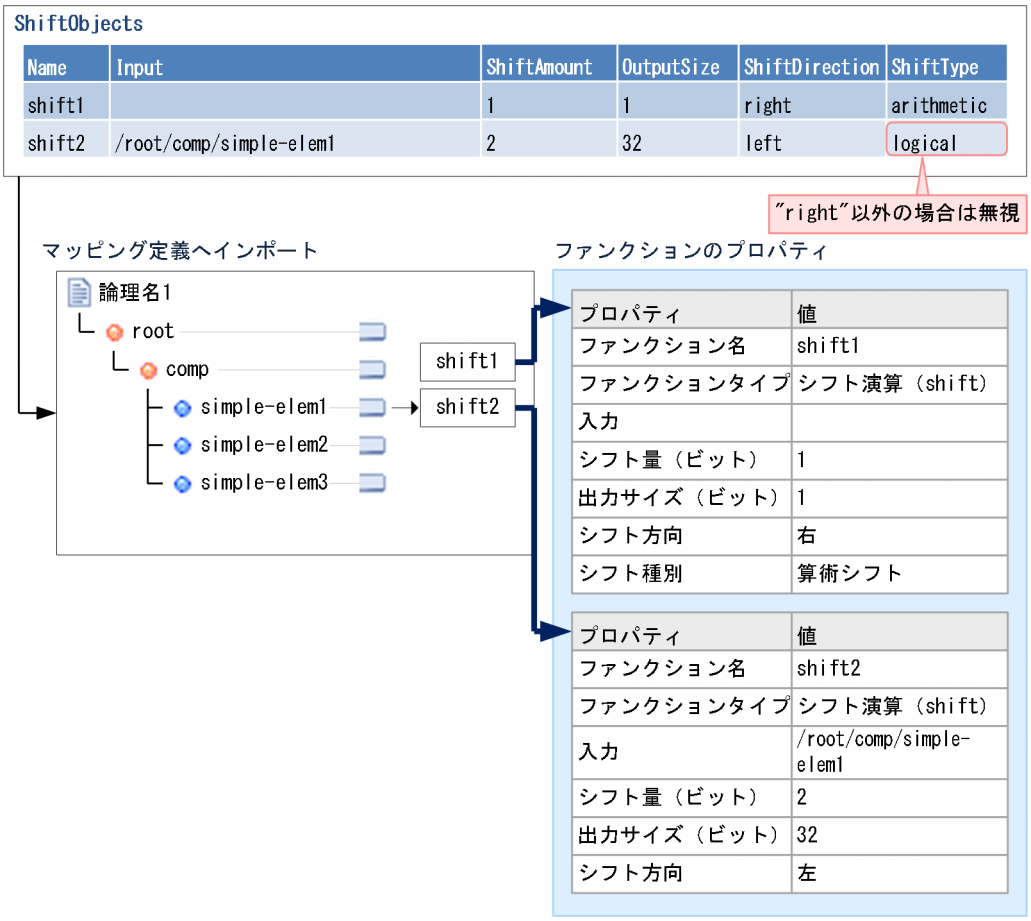
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-85　ShiftObjects 定義例



7.15.15　NameObjects（ノード名取得ファンクション）

ノード名取得ファンクション（name）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-40　NameObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

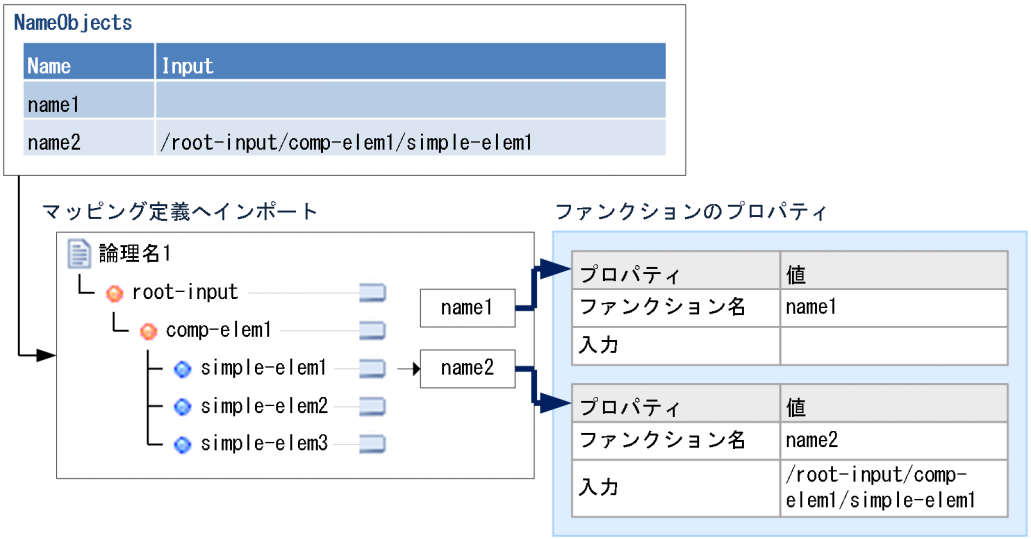
（凡例）

－：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-86 NameObjects 定義例



7.15.16 CountObjects（ノード数取得ファンクション）

ノード数取得ファンクション（count）を定義します。

(1) 定義項目

定義項目を次の表に示します。

表 7-41 CountObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2 マッピング定義の設定」を参照	－

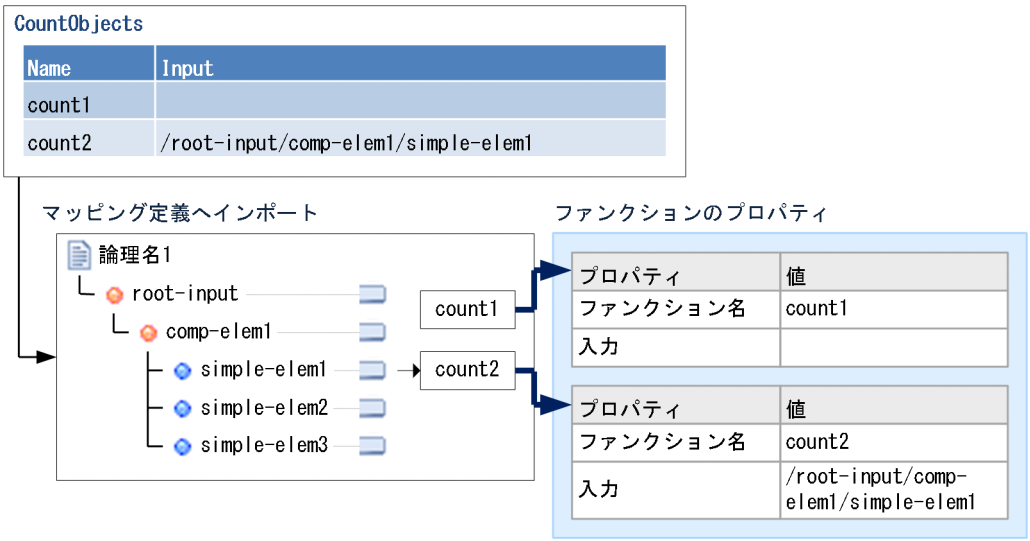
（凡例）

－：該当する項目はありません。

(2) 定義例

定義例を次の図に示します。

図 7-87　CountObjects 定義例



7.15.17　ExistObjects（ノード調査ファンクション）

ノード調査ファンクション（exist）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-42　ExistObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－

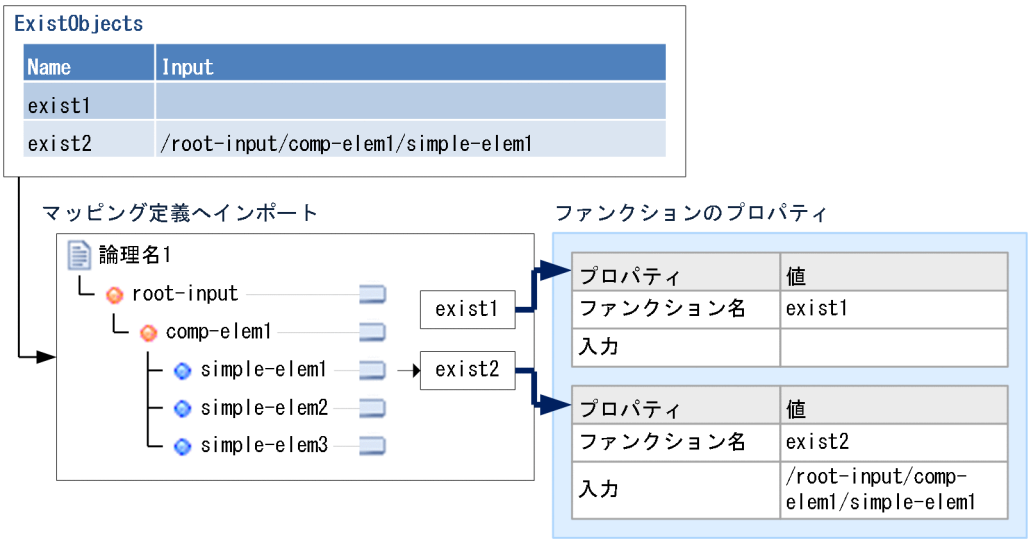
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-88　ExistObjects 定義例



7.15.18　LoopObjects（くり返しファンクション）

くり返しファンクション（loop）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-43　LoopObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	定義できる値については，「7.15.25　同一の Name 要素を複数行に定義できるオブジェクト」を参照してください。
Input	入力	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で 1 つだけ定義できます。※1
RelationalPath	連携パス	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。※1 値は，定義された順にリストに設定します。
SortKey	ソート条件/キー	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。※2※3
SortOrder	ソート条件/順序	"ascending"または"descending"	同じ Name 内で複数定義できます。※2※3 "ascending"の場合 　ソート条件/順序は「昇順」を設定します。 "descending"の場合 　ソート条件/順序は「降順」を設定します。

項目名	定義内容	定義できる値	補足説明
SortLanguage	ソート条件/言語	"auto"，"ja"，"en"のどれか	同じ Name 内で複数定義できます。※2※3 "auto"の場合 ソート条件/言語は「自動」を設定します。 "ja"の場合 ソート条件/言語は「日本語」を設定します。 "en"の場合 ソート条件/言語は「英語」を設定します。
SortDataType	ソート条件/データタイプ	"text"または"numeric"	同じ Name 内で複数定義できます。※2※3 "text"の場合 ソート条件/データタイプは「テキスト」を設定します。 "numeric"の場合 ソート条件/データタイプは「数値」を設定します。
SortCase	ソート条件/優先順位	"upper"または"lower"	SortDataType が"text"の場合，必須項目です。 上記以外の場合，設定値を取り込みません。 同じ Name 内で複数定義できます。※2※3 "upper"の場合 ソート条件/優先順位は「大文字」を設定します。 "lower"の場合 ソート条件/優先順位は「小文字」を設定します。

注※1

同じ Name 内で定義されている Input と RelationalPath は，お互いが同じノード，または祖先・子孫の関係になるノードは設定できません。

注※2

同じ行の中で，ソート条件項目（SortKey，SortOrder，SortLanguage，SortDataType，SortCase）のどれか 1 つでも設定した場合は，必須項目です。

注※3

ソート条件は，定義された順にリストに設定します。

(2) 定義例

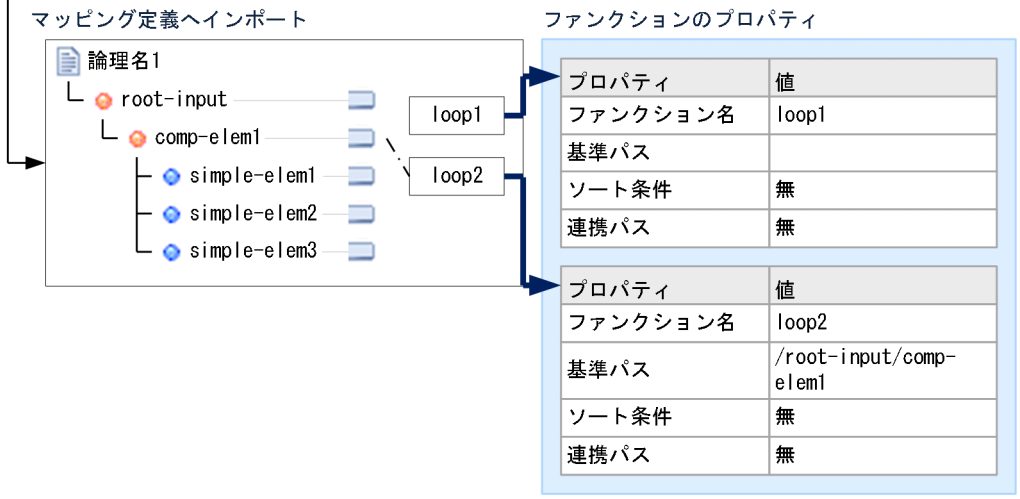
定義例を次の図に示します。

図 7-89　LoopObjects 定義例 1

≪基準パスを設定する場合≫

LoopObjects

Name	Input	RelationalPath	SortKey	SortOrder	SortLanguage	SortDataType	SortCase
loop1							
loop2	/root-input/comp-elem1						



プロパティ	値
ファンクション名	loop1
基準パス	
ソート条件	無
連携パス	無

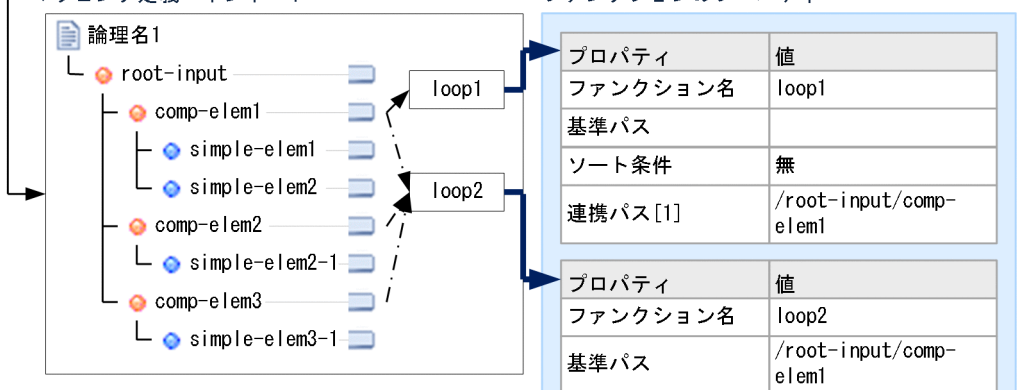
プロパティ	値
ファンクション名	loop2
基準パス	/root-input/comp-elem1
ソート条件	無
連携パス	無

図 7-90　LoopObjects 定義例 2

≪連携パスを設定する場合≫

LoopObjects

Name	Input	RelationalPath	SortKey	SortOrder	SortLanguage	SortDataType	SortCase
loop1		/root-input/comp-elem1					
loop2	/root-input/comp-elem1	/root-input/comp-elem2					
		/root-input/comp-elem3					



プロパティ	値
ファンクション名	loop1
基準パス	
ソート条件	無
連携パス[1]	/root-input/comp-elem1

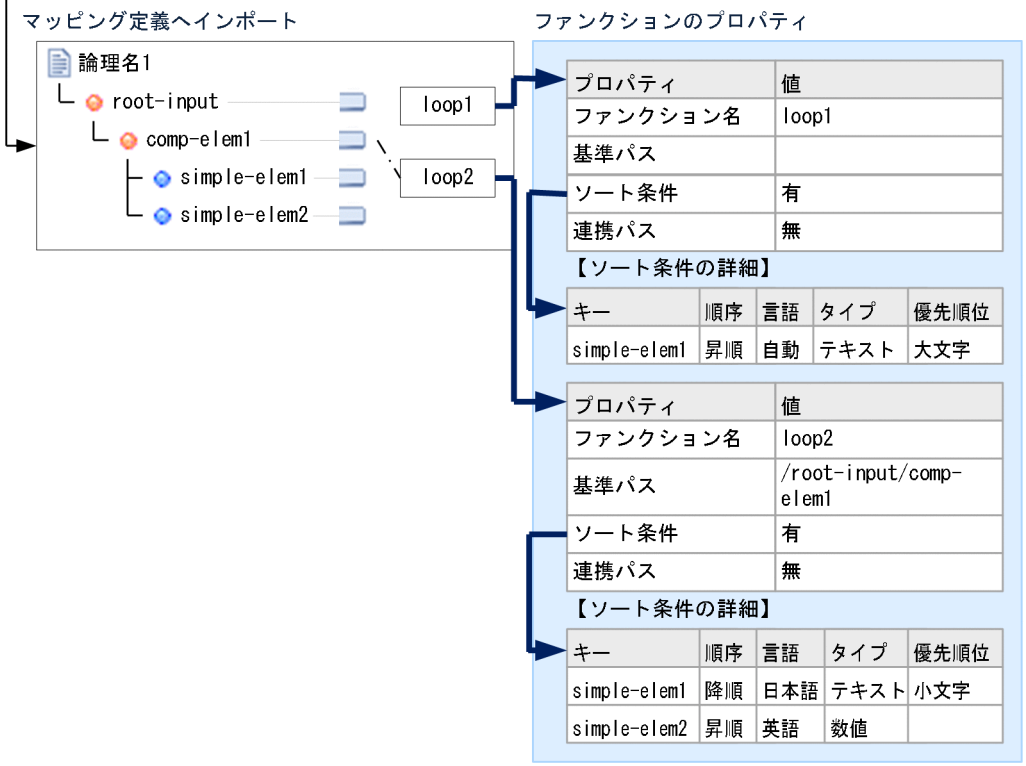
プロパティ	値
ファンクション名	loop2
基準パス	/root-input/comp-elem1
ソート条件	無
連携パス[1]	/root-input/comp-elem2
連携パス[2]	/root-input/comp-elem3

図 7-91　LoopObjects 定義例 3

≪ソート条件を設定する場合≫

LoopObjects

Name	Input	Relational Path	SortKey	SortOrder	Sort Language	Sort DataType	Sort Case
loop1			/root-input/comp-elem1/simple-elem1	ascending	auto	text	upper
loop2	/root-input/comp-elem1		/root-input/comp-elem1/simple-elem1	descending	ja	text	lower
			/root-input/comp-elem1/simple-elem2	ascending	en	numeric	lower



7.15.19　ChooseObjects（選択ファンクション）

選択ファンクション（choose）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-44　ChooseObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	定義できる値については，「7.15.25　同一の Name 要素を複数行に定義できるオブジェクト」を参照してください。
Condition	条件	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。※

項目名	定義内容	定義できる値	補足説明
Condition	条件	「7.13.2　マッピング定義の設定」を参照	どの条件にも合致しない場合の定義を行うときは，"#other"を指定します。同じ Name 内で"#other"を指定できるのは 1 行だけです。また，同じ Name 内で"#other"を 1 つも指定しなかった場合，「どの条件にも合致しない場合」の出力値は，「出力なし」を仮定します。
OutputValue	出力値	「7.13.2　マッピング定義の設定」を参照	同じ Name 内で複数定義できます。※ 「空ノード」の場合は"#empty"，「出力なし」の場合は"#notoutput"を指定します。

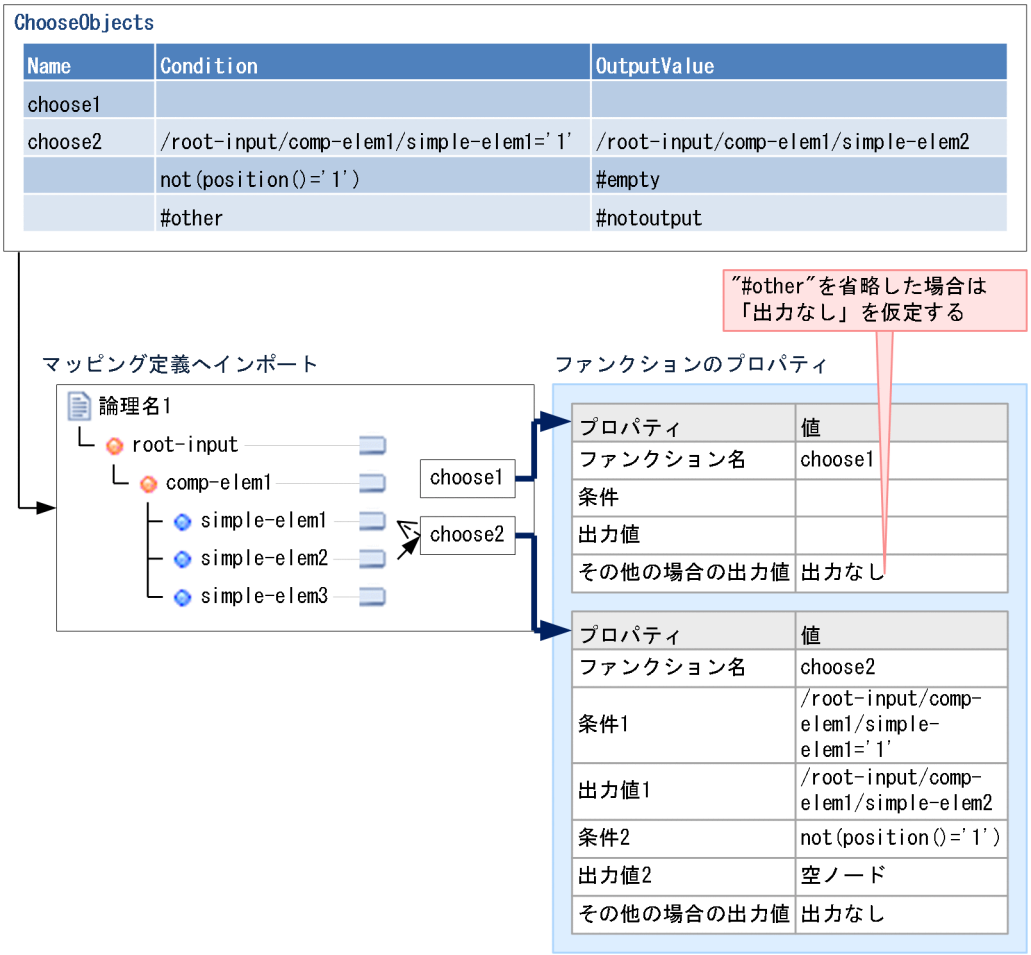
注※

条件と出力値は，定義された順にリストに設定します。

(2)　定義例

定義例を次の図に示します。

図 7-92　ChooseObjects 定義例



7.15.20　ReplaceObjects（値置換えファンクション）

値置換えファンクション（replace）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-45　ReplaceObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－
TableID	変換テーブル ID	1,024 文字以内の文字列（改行は定義できない）	－
KeyColumn	検索キー列指定	"left"または"right"	必須項目です。 "left"の場合 検索キー列指定は「左列」を設定します。 "right"の場合 検索キー列指定は「右列」を設定します。
FaultError	検索失敗時の動作	"y"または"n"	必須項目です。 "y"の場合 検索失敗時の動作は「変換エラー」を設定します。 "n"の場合 検索失敗時の動作は「デフォルト値を代入」を設定します。
DefaultValue	値	1,024 文字以内の文字列（改行は定義できない）	SortDataType が"y"の場合，設定値を取り込みません。

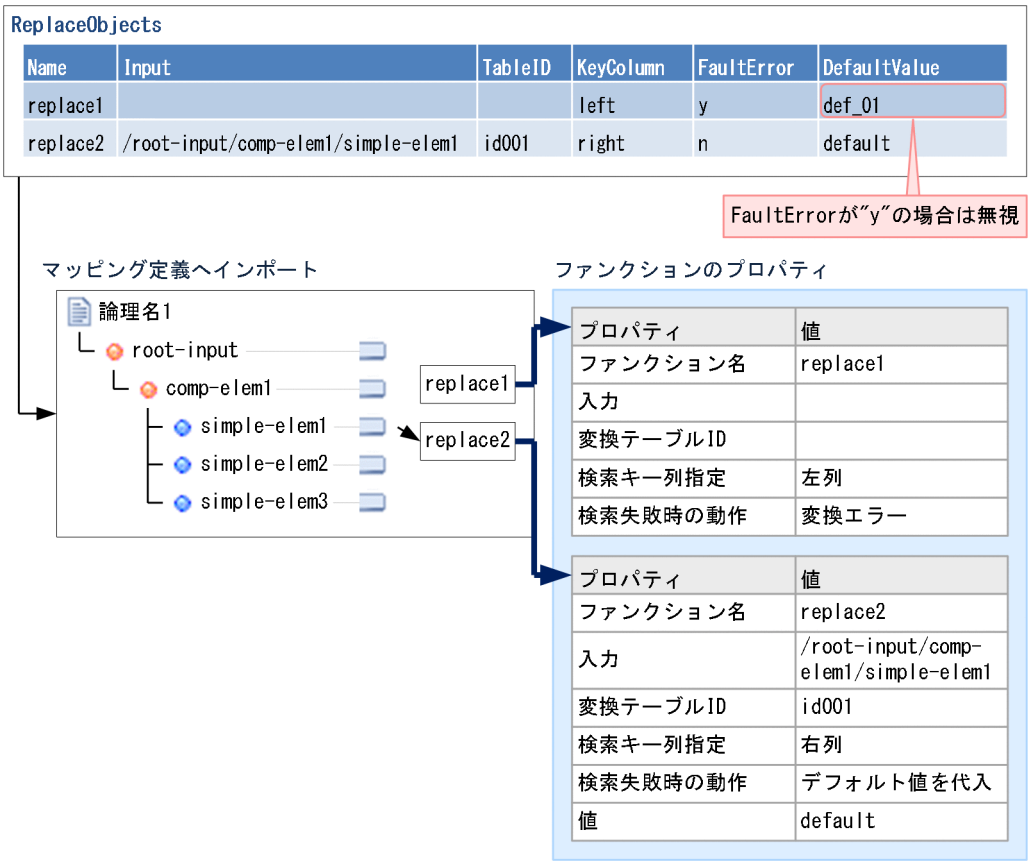
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-93　ReplaceObjects 定義例



7.15.21　RadixObjects（基数変換ファンクション）

基数変換ファンクション（radix）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-46　RadixObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Input	入力	「7.13.2　マッピング定義の設定」を参照	－
InputRadix	入力基数	"hexadecimal"，"decimal"，"binary"のどれか	必須項目です。 "hexadecimal"は，16 進数。 "decimal"は，10 進数。 "binary"は，2 進数。

項目名	定義内容	定義できる値	補足説明
InputRadix	入力基数	"hexadecimal"，"decimal"，"binary"のどれか	入力基数と出力基数に同じ基数を定義できません。
OutputRadix	出力基数	"hexadecimal"，"decimal"，"binary"のどれか	必須項目です。 "hexadecimal"は，16 進数。 "decimal"は，10 進数。 "binary"は，2 進数。 入力基数と出力基数に同じ基数を定義できません。

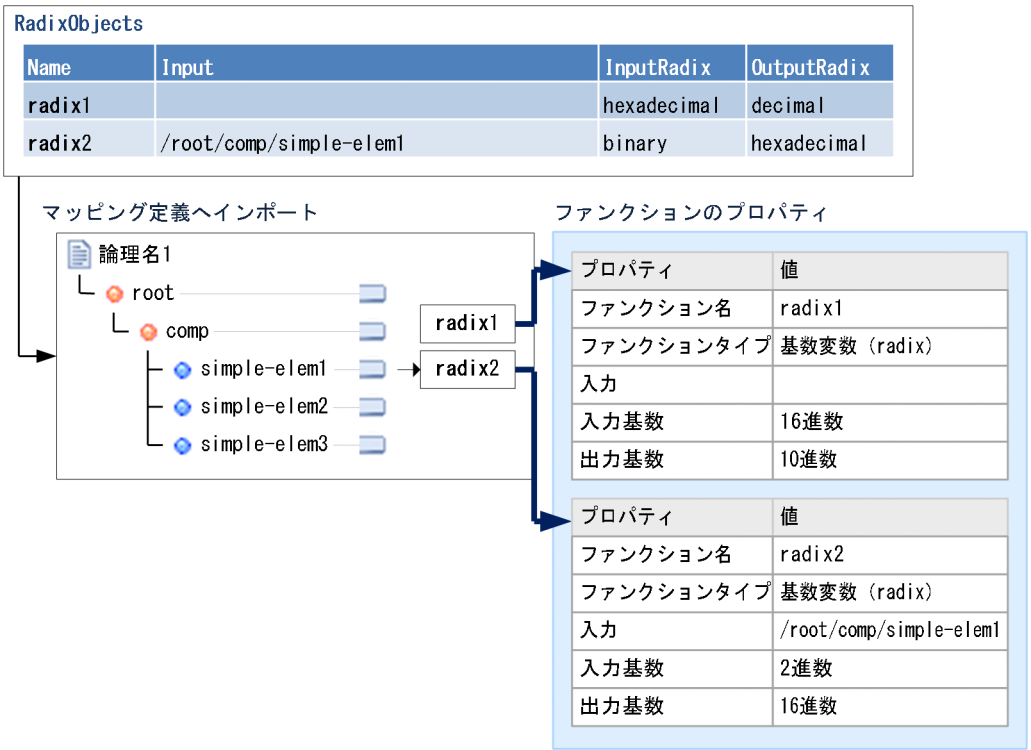
（凡例）

－：該当する項目はありません。

(2)　定義例

定義例を次の図に示します。

図 7-94　RadixObjects 定義例



7.15.22　CustomObjects（カスタムファンクション）

カスタムファンクション（custom）を定義します。

カスタムファンクションで指定するクラスと引数にはコメントが存在します。ただし，Excel を利用したマッピングではこれらの項目のインポートはサポートしません。

パッケージとクラスは別々に定義し，マッピング定義エディタの画面上では「クラス」の項目に Excel で定義した「パッケージ＋クラス」の値を設定します。

(1) 定義項目

定義項目を次の表に示します。

表 7-47 CustomObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	定義できる値については，「7.15.25 同一の Name 要素を複数行に定義できるオブジェクト」を参照してください。
Jar	JAR ファイル	100 文字以内の NCName 型の値	同じ Name 内で 1 つだけ定義できます。※1
Package	パッケージ	255 文字以内の NCName 型の値	同じ Name 内で 1 つだけ定義できます。※1
Class	クラス	100 文字以内の NCName 型の値	同じ Name 内で 1 つだけ定義できます。※1
Method	メソッド	100 文字以内の NCName 型の値	同じ Name 内で 1 つだけ定義できます。※1 メソッド名だけを設定し，後ろの"()"は付加しません。
ReturnType	戻り値の型	"String"または"NodeList"だけ	同じ Name 内で 1 つだけ定義できます。※1※2
Argument	引数	100 文字以内の NCName 型の値	同じ Name 内で複数定義できます。※3※4
ArgumentType	引数の型	"String"または"Object"だけ	同じ Name 内で複数定義できます。※2※3※4
Input	入力値	「7.13.2 マッピング定義の設定」を参照	同じ Name 内で複数定義できます。 値は，定義された順にリストに設定します。

注※1

変換ファンクション定義項目（Jar，Package，Class，Method，ReturnType）を指定する場合は，同じ Name 内の先頭行で必ず指定します。同じ Name 内の先頭行以外は，先頭行と同じ値または空を指定します。

注※2

ArgumentType が"String"の場合，ReturnType は"String"だけ定義できます。

注※3

同じ行の中で，引数項目（Argument，ArgumentType）のどれか 1 つでも設定した場合は必須項目です。

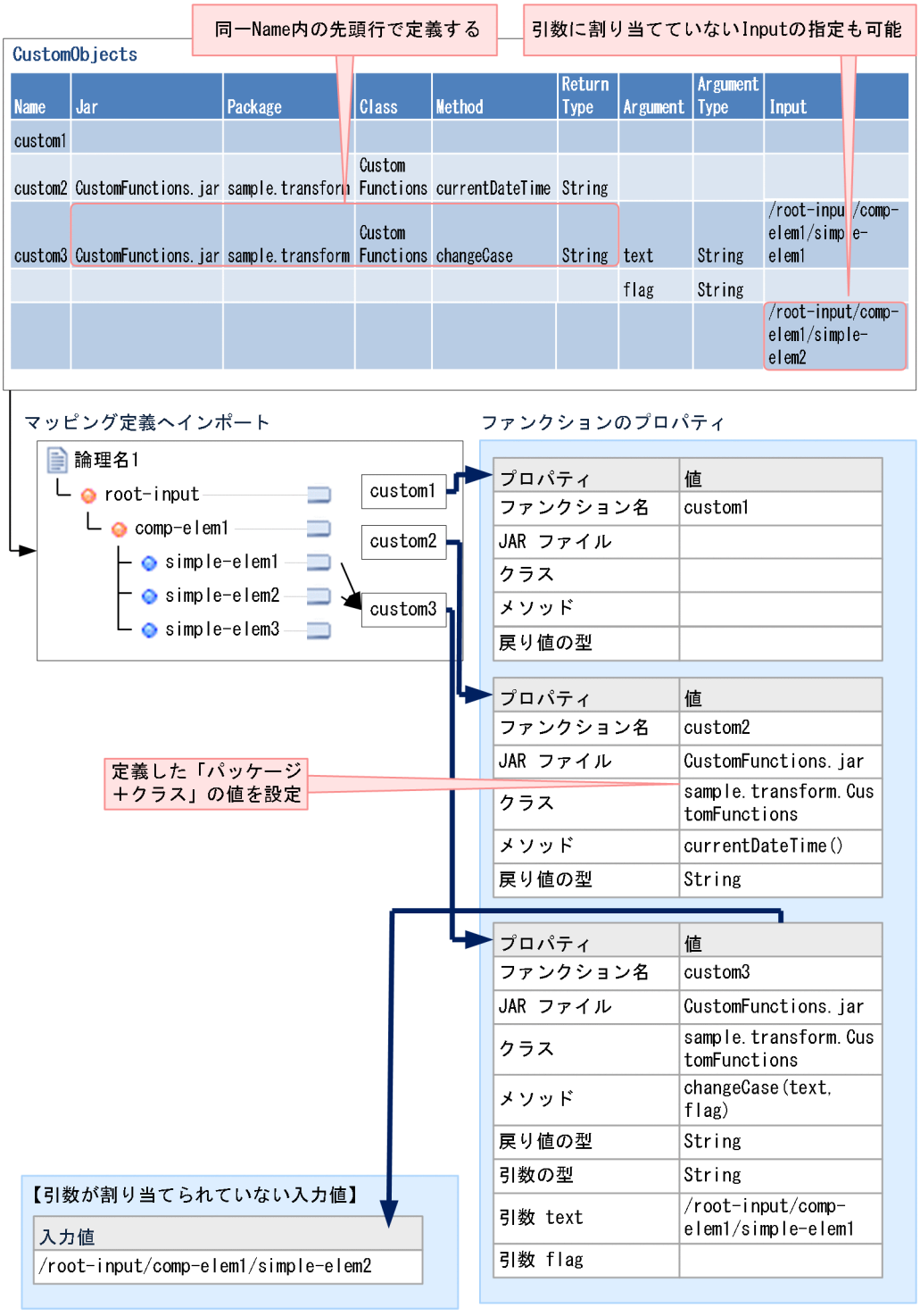
注※4

引数は，定義された順にリストに設定します。

(2) 定義例

定義例を次の図に示します。

図 7-95 CustomObjects 定義例



7.15.23　ConstantObjects（定数ファンクション）

定数ファンクション（const）を定義します。

(1)　定義項目

定義項目を次の表に示します。

表 7-48　ConstantObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Type	型	"string"，"number"，"boolean"，"notoutput"，"empty"のどれか	必須項目です。 "string"の場合 　型は「文字列」を設定します。 "number"の場合 　型は「数値」を設定します。 "boolean"の場合 　型は「論理値」を設定します。 "notoutput"または"empty"の場合 　型は「特殊ノード」を設定します。
Value	値	• Type が"string"の場合，1,024 文字以内の文字列（改行は定義できない） • Type が"number"の場合，1,024 桁以内の数値 • Type が"boolean"の場合，"true"または"false"※	Type が"boolean"または"number"の場合，必須項目です。 Type が"notoutput"または"empty"の場合，設定値を取り込みません。 Type が"notoutput"の場合 　値は「ノード出力なし」を設定します。 Type が"empty"の場合 　値は「空ノード」を設定します。

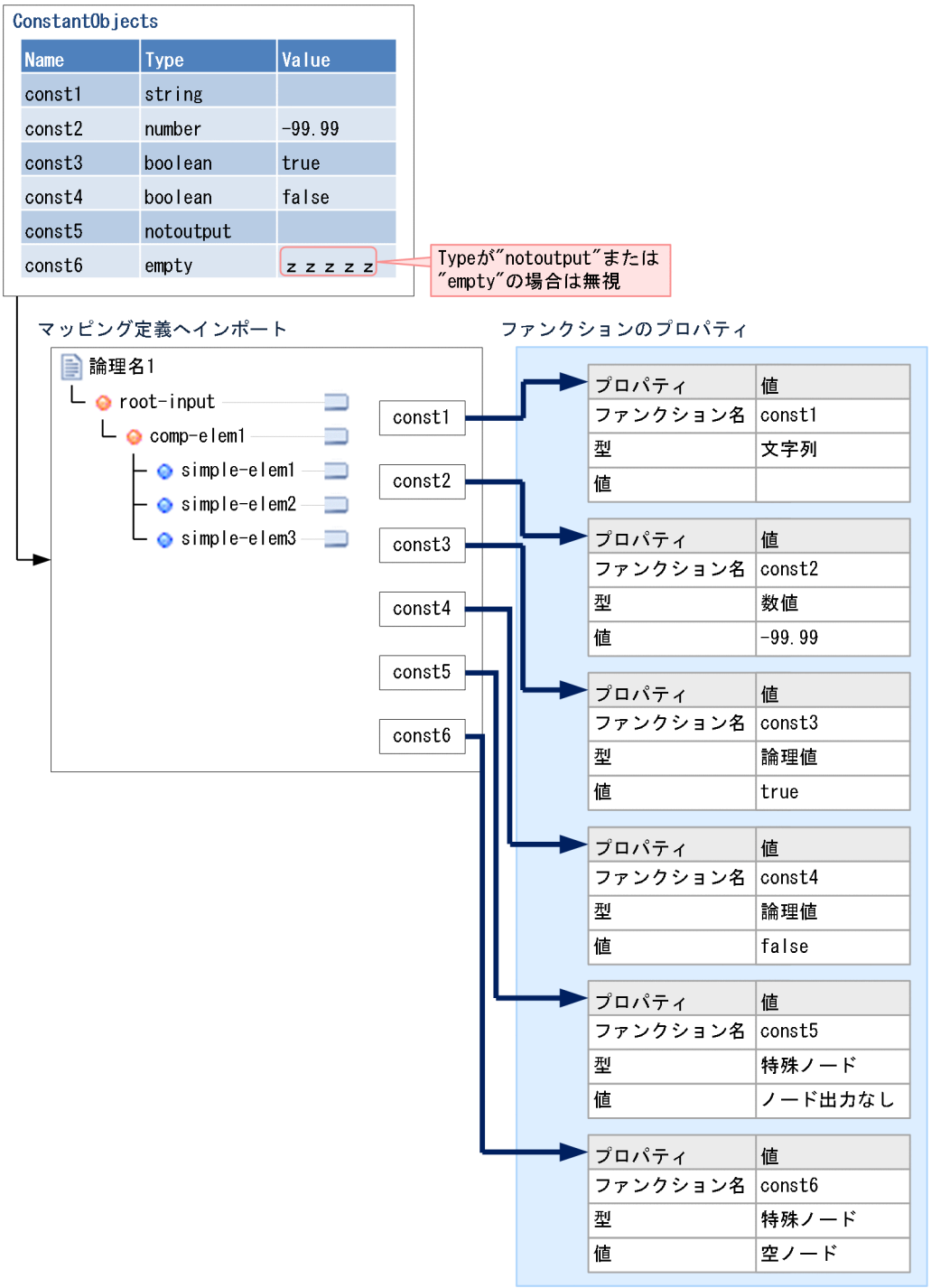
注※

大文字および小文字は区別されません。

(2)　定義例

定義例を次の図に示します。

図 7-96　ConstantObjects 定義例



7.15.24　XpathObjects（XPath ファンクション）

XPath ファンクション（xpath）を定義します。

(1)　定義項目

定義項目を次に示します。

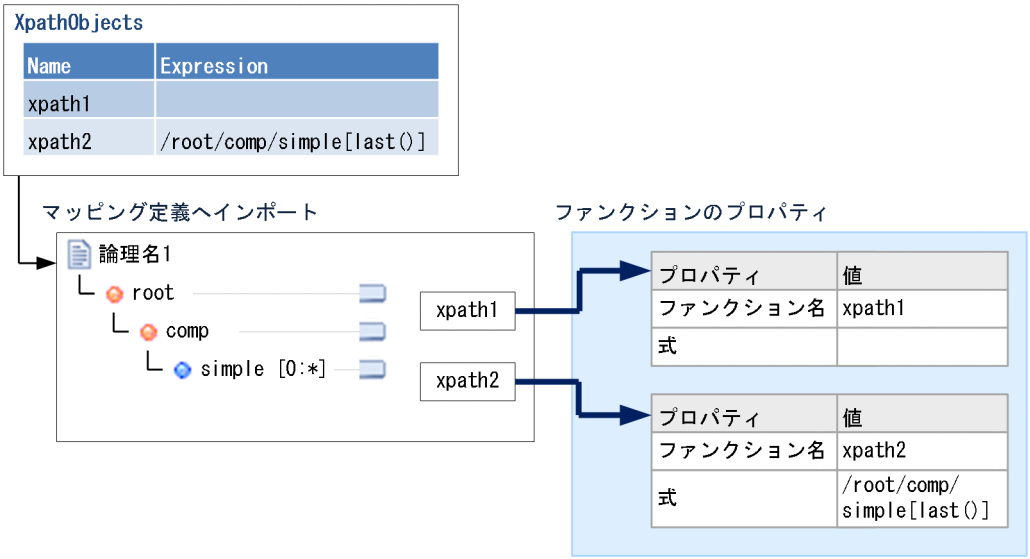
表 7-49　XpathObjects 定義項目

項目名	定義内容	定義できる値	補足説明
Name	ファンクション名	NCName 型の値	必須項目です。Objects 内で，重複して同じ Name を定義できません。
Expression	XPath 式	文字列（改行は定義できない）	空白の場合は，空文字列として扱います。 入力できる XPath の仕様については，「7.5.24　変換先ノードやファンクションの入力値に XPath 式を指定する」を参照してください。

(2)　定義例

定義例を次の図に示します。

図 7-97　XpathObjects 定義例



7.15.25　同一の Name 要素を複数行に定義できるオブジェクト

一部の Objects では，同一の Name 要素を複数行に定義することができます。同一の Name 要素の行は同じ 1 つのオブジェクトに対する定義と見なされ，複数の値の定義で使用できます。この場合，値は定義された順にリストに設定されます。

同一の Name 要素を複数行に定義できるオブジェクト，およびそのオブジェクトで複数の値を定義できる項目を次の表に示します。

表 7-50　同一の Name 要素を複数行に定義できるオブジェクト

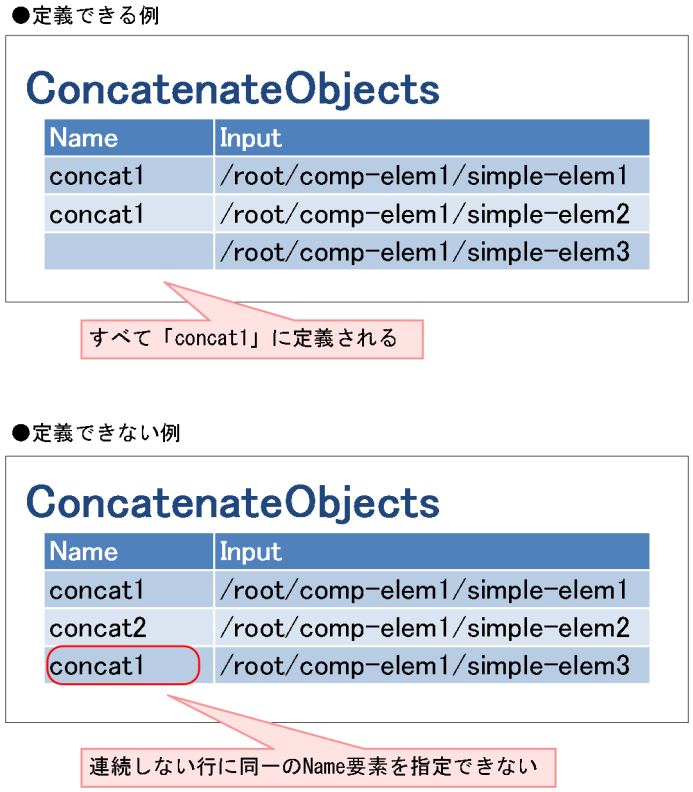
#	オブジェクト	複数の値を定義できる項目
1	ConcatenateObjects	Input

#	オブジェクト	複数の値を定義できる項目
2	SumObjects	Input
3	LoopObjects	RelationalPath, SortKey, SortOrder, SortLanguage, SortDataType, SortCase
4	ChooseObjects	Condition, OutputValue
5	CustomObjects	Argument, ArgumentType, Input

同一の Name 要素は連続する行でだけ定義できます。連続しない行に同一の Name 要素を定義した場合は検証時にエラーになります。また，2 行目以降で Name 要素が省略された場合は，上の行の値が設定されます。

同一の Name 要素を複数行に定義した場合の例を次の図に示します。

図 7-98　同一の Name 要素を設定した場合の例



7.16　Excel を利用したマッピング定義時の注意事項

Excel でマッピングを定義するときの注意事項を次に示します。

- Excel のオートフォーマット機能によって，入力値が別の値に置き換わるおそれがあります。そのため，［セルの書式設定］の表示形式を［文字列］にすることを推奨します。
- Excel に入力した値は，大文字・小文字が区別されます。
- データ変換定義画面上で設定できない定義がされた場合，インポート時にエラーになります。設定できない定義とは，同じファンクションを複数の入力値から参照する定義などのことです。
- データ変換定義画面上で非活性となる項目には，データ変換定義画面上でファンクションを配置したときのデフォルトの値が設定されます。
- XML 要素の対応づけをしたセルは，Excel のテーブルとなります。テーブルについては，Excel のヘルプを参照してください。
- XML 要素の対応づけをしたテーブルで不要な行を削除する場合，セルを空にするのではなく，行を削除してください。行を削除していないと，すべて空のセルであっても空行としてエクスポートされるため，必須項目がある場合，インポート時にエラーとなります。
- 旧バージョンのテンプレートファイルを使用して，かつ新しい定義を使用する場合，次のどちらかの作業を実施する必要があります。
  - 新しいテンプレートファイルを取得し直す。
    手順については，「7.13.1(1)　テンプレートファイルを利用する場合」を参照してください。
  - 旧バージョンのテンプレートファイルをそのまま使用し，表形式 XML スキーマ定義ファイルを再度対応づける。
    手順については，「7.13.1(2)　表形式 XML スキーマ定義ファイルを基に要素を対応づける場合」を参照してください。
- 不正な操作によって，手動で意図的な操作をしていなくても Excel 上に空行ができてしまうことがあります。不正な操作とは，テンプレートファイルを複数の定義で再利用する，過去に作成した不正な表形式 XML ファイルをテンプレートファイルにインポートするなどを指します。空行ができてしまった場合，その空行の行を削除してください。
- Excel のセルにホワイトスペース（半角空白，タブ，ラインフィード，キャリッジリターン）だけの定義が存在する状態でエクスポートし，出力された表形式 XML ファイルを再度 Excel にインポートするとホワイトスペースだけの定義が削除されます。そのため，エクスポート時に出力される表形式 XML ファイルのルート要素（Mapping）に次の属性を追加してください。
  - xml:space="preserve"
- 「7.13.1(2)　表形式 XML スキーマ定義ファイルを基に要素を対応づける場合」の手順を実施した際に，「名前空間の接頭文字には，予約語'xml'を使用できません。」というエラーメッセージが表示されることがあります。エラーが発生した場合，次の作業を実施して再度表形式 XML スキーマ定義ファイルを追加してください。

- 表形式 XML スキーマ定義ファイル（dt_import_mapping.xsd）を開いて xsd:schema 要素に定義されている名前空間接頭辞 xmlns:xml（xmlns:xml=http://www.w3.org/XML/1998/namespace）の定義を削除してください。

7.17　名前空間接頭辞オプション

名前空間接頭辞オプションによって，名前空間接頭辞を付けるかを設定できます。マッピング定義エディタの，変換元ビューア，変換先ビューア，またはマッピングビューアから設定します。コンテキストメニューを表示して，「名前空間接頭辞を XML に出力する」オプションを有効または無効にします。

次に該当する場合，名前空間接頭辞を XML に出力しない設定でも XSLT の仕様によって名前空間接頭辞が付加されます。

- any 要素や anyAttribute 属性のマッピングで入力 XML に接頭辞があった場合
- カスタムファンクションで接頭辞を明示的または暗黙的に付けた場合
- 属性に名前空間がある場合

旧バージョンのマッピング定義ファイルを開いた場合，「名前空間接頭辞を XML に出力する」は，有効な状態に設定されています。

7.17.1　名前空間接頭辞オプションのデフォルト値設定

eclipse.ini でプロパティを設定することで，名前空間接頭辞オプションのデフォルト値を設定できます。このデフォルト値によって，マッピング定義を新規に定義したときの「名前空間接頭辞を XML に出力する」オプションの有効／無効が決定されます。eclipse.ini は次のディレクトリに格納されています。

```
<Eclipseのインストールディレクトリ>¥eclipse
```

プロパティの設定内容を次に示します。

表 7-51　名前空間接頭辞オプションのデフォルト値の設定

プロパティ名	プロパティ値※	説明
cscte.dt.default.output.namespace.prefix	true	「名前空間接頭辞を XML に出力する」の設定が有効な状態をデフォルト値とする。
	false	「名前空間接頭辞を XML に出力する」の設定が無効な状態をデフォルト値とする。

注※

プロパティ値の指定は，大文字小文字の区別をしません。

プロパティ値の指定がない場合，および有効でない値が設定された場合は，true で処理します。

eclipse.ini のプロパティ値の設定例を次に示します。

```
-Dcscte.dt.default.output.namespace.prefix=false
```

7.18　提供しているカスタムファンクション

サービスプラットフォームでは，次に示す 4 つのカスタムファンクションを提供しています。

なお，カスタムファンクションの詳細については，「7.5.23　ユーザが作成した任意の Java プログラムを呼び出す」を参照してください。

表 7-52　関数の一覧

#	メソッド名	機能概要
1	getWorkFolderPath	作業フォルダのパスを取得する関数
2	currentDateTime	システム日付を取得する関数
3	lrtrim	文字列の前後の空白を削除する関数
4	getSystemEnv	システム環境変数を取得する関数
5	getPropertiesByGroup	外部プロパティファイルからプロパティを取得する関数

以降で各カスタムファンクションの詳細を説明します。

7.18.1　getWorkFolderPath（作業フォルダパス取得関数）

引数に指定したリクエスト ID に該当する作業フォルダのパスを取得する関数です。

また，作業フォルダのパスの末尾にセパレータを付与するかどうかを指定できます。

作業フォルダのルート以降のセパレータは，OS に依存します。

(1)　関数仕様

関数仕様を次の表に示します。

表 7-53　getWorkFolderPath の関数仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.customfunc
2	クラス	CSCCustomfunc
3	メソッド	getWorkFolderPath
4	引数 1	requestId リクエスト ID
5	引数 1 の型	java.lang.String
6	引数 2	endSeparatorFlag

#	項目	値
6	引数 2	末尾のセパレータの付与有無 0 を指定した場合，作業フォルダのパスの末尾にセパレータを付与しない 1 を指定した場合，作業フォルダのパスの末尾にセパレータを付与する※
7	引数 2 の型	java.lang.String
8	戻り値	リクエスト ID に該当する作業フォルダのパス
9	戻り値の型	java.lang.String

注※

0，または 1 以外を指定した場合，0 を指定したものとみなします。

7.18.2 currentDateTime（システム日付取得関数）

引数に指定したフォーマットでカスタムファンクション実行時のシステム日時を取得する関数です。取得時刻はローカルタイムのみとします。

(1) 関数仕様

関数仕様を次の表に示します。

表 7-54 currentDateTime の関数仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.customfunc
2	クラス	CSCCustomfunc
3	メソッド	currentDateTime
4	引数	format 日付時刻フォーマット（java.text.SimpleDateFormat に準拠）
5	引数の型	java.lang.String
6	戻り値	システム日付時刻
7	戻り値の型	java.lang.String

(2) 変換例

システム日付が「2015 年 06 月 03 日 16 時 29 分 03 秒」の場合の変換例を次の表に示します。

表 7-55 変換例

#	引数に指定した値	戻り値
1	""（空文字）	""（空文字）

#	引数に指定した値	戻り値
2	x	エラーが発生します
3	yyyy-MM-dd'T'HH:mm:ss.SSSXXX	2015-06-03T16:29:03.590+09:00

7.18.3 lrtrim（前後空白除去関数）

引数に指定した文字列の前後の空白を除去する関数です。この関数は，java.lang.String#trim を実行しています。

(1) 関数仕様

関数仕様を次の表に示します。

表 7-56 lrtrim の関数仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.customfunc
2	クラス	CSCCustomfunc
3	メソッド	lrtrim
4	引数	value 空白除去対象となる文字列
5	引数の型	java.lang.String
6	戻り値	空白除去後の文字列
7	戻り値の型	java.lang.String

(2) 変換例

変換例を次の表に示します。

表 7-57 変換例

#	引数に指定した値	戻り値
1	""（空文字）	""（空文字）
2	a△△a	a△△a
3	△△a△△a△△	a△△a
4	△□aa□	□aa□

（凡例）

△：半角スペース

□：全角スペース

7.18.4　getSystemEnv（環境変数取得関数）

引数に指定した環境変数名に該当するシステム環境変数を取得する関数です。この関数は，java.lang.System#getenv を実行しています。

なお，存在しない環境変数を指定した場合，空文字を返します。

(1)　関数仕様

関数仕様を次の表に示します。

表 7-58　getSystemEnv の関数仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.customfunc
2	クラス	CSCCustomfunc
3	メソッド	getSystemEnv
4	引数	name 値を取得したい環境変数名
5	引数の型	java.lang.String
6	戻り値	環境変数の値
7	戻り値の型	java.lang.String

(2)　変換例

サービスプラットフォームが「C:¥Program Files¥Hitachi¥Cosminexus」ディレクトリにインストールされた場合の変換例を次の表に示します。

表 7-59　変換例

#	引数に指定した値	戻り値
1	""（空文字）※	""（空文字）
2	aa※	""（空文字）
3	COSMINEXUS_HOME	C:¥Program Files¥Hitachi¥Cosminexus

注※

存在しない環境変数名を指定した場合

7.18.5　getPropertiesByGroup（外部プロパティ取込関数）

引数に指定した外部プロパティファイルから，引数に指定したキーワードがマッピングされているグループに属するプロパティを取得する関数です。

外部プロパティファイルの定義規則は，「5.6.5(7)　Java 呼出アクティビティから利用できる API」を参照してください。

外部プロパティ取込関数は，cscdt_api.jar にパッケージされたクラスを利用します。このため usrconf.cfg（J2EE サーバ用オプション定義ファイル）に cscdt_api.jar ファイルのクラスパスを指定してください。

(1)　関数仕様

関数仕様を次の表に示します。

表 7-60　getPropertiesByGroup の関数仕様

#	項目	値
1	パッケージ	jp.co.Hitachi.soft.csc.dt.customfunc
2	クラス	CSCCustomGroupedProperties
3	メソッド	getPropertiesByGroup 引数に指定した外部プロパティファイルから，引数に指定したキーワードに対応したグループに属するプロパティを取得する関数
4	引数 1	java.lang.Object filePath 外部プロパティファイルの絶対パス 相対パスを指定した場合は，サポートしません。
5	引数 2	java.lang.Object keyword 取得対象キーワード
6	引数 3	java.lang.Object namespace 戻り値の名前空間を使用する要素に付与する名前空間 URI 出力結果のプロパティに prefix 付きのパス要素があり，かつ，この引数に空文字列が指定された場合は，外部プロパティ取込関数は異常終了（例外をスロー）します。
7	戻り値	org.w3c.dom.NodeLis 外部プロパティファイル内で keyword にマッピングされたグループに属するプロパティのリスト

(2)　注意事項

外部プロパティファイルは，各プロパティのキー値を次の形式と見なして XML ノードに展開します。

```
GroupID.NodeName[.NodeName]*
```

先頭の GroupID は，取得対象のグループを決定するために使用し，戻り値の XML ノードリストから取り除きます。続く NodeName[.NodeName]をピリオド（.）で区切られた XML 要素名として XML 階層構造に展開します。

NodeName にピリオド（.）を含める場合は，「¥.」（円記号またはバックスラッシュに続くピリオド）にエスケープしてください。

filePath パラメタに指定したパスに，次に示す外部プロパティファイルに起因するエラーが発生した場合，外部プロパティ取込関数は例外を発生してエラー終了します。

- ファイルの読み込みに失敗した場合
- 内容が不正の場合

(3) 変換例

filePath で指定する外部プロパティファイルが次の定義例の場合の，変換例を「表 7-61 変換例」に示します。

```
group1 = Group1Key1, Group1Key2, Group1Key3
group2 = Group2Key1, Group2Key2, Group2Key3
group1.nodeA.node1 = Property Value 1
group1.nodeA.node2 = Property Value 2
group1.nodeA.node¥¥.X = Property Value 3
group2.nodeA = Property Value 4
group2.nodeB = Property Value 5
```

表 7-61 変換例

入力データ（keyword 引数）	出力結果
Group1Key1, Group1Key2, Group1Key3 のどれか	[<nodeA> <node1>Property Value 1</node1> <node2>Property Value 2</node2> <node.X>Property Value 3</node.X> </nodeA>]
Group2Key1, Group2Key2, Group2Key3 のどれか	[<nodeA>Property Value 4</nodeA> <nodeB>Property Value 5</nodeB>]
上記以外	[]（空のノードリスト）

外部プロパティ取込関数を TP1 アダプタ接続先動的変更のヘッダ割当変数に設定する場合の，外部プロパティファイルの定義例と変換例を次に示します。

```
# Sample external property for TP1 Dynamic Connection function
g0 = g0
g0.tpdh¥:TP1DynamicConnection.tpdh¥:static_host = localhost
g0.tpdh¥:TP1DynamicConnection.tpdh¥:static_port = 10001

g1 = g1
g1.tpdh¥:TP1DynamicConnection.tpdh¥:static_host = tp1.unknown.nothing
g1.tpdh¥:TP1DynamicConnection.tpdh¥:static_port = 10002
```

namespace 引数に「http://www.hitachi.co.jp/soft/xml/cosminexus/csc/tp1dynamics/header」を指定したときの変換例を次の表に示します。

表 7-62　変換例（ヘッダ割当変数に設定した場合）

入力データ（keyword 引数）	出力結果
g0	[<tpdh:TP1DynamicConnection xmlns:tpdh=”http://www.hitachi.co.jp/soft/xml/cosminexus/csc/tp1dynamics/header”> <tpdh:static_host>localhost</tpdh:static_host> <tpdh:static_port>10001</tpdh:static_port> </tpdh:TP1DynamicConnection>]
g1	[<tpdh:TP1DynamicConnection xmlns:tpdh=”http://www.hitachi.co.jp/soft/xml/cosminexus/csc/tp1dynamics/header”> <tpdh:static_host>tp1.unknown.nothing</tpdh:static_host> <tpdh:static_port>10002</tpdh:static_port> </tpdh:TP1DynamicConnection>]
上記以外	[]（空のノードリスト）

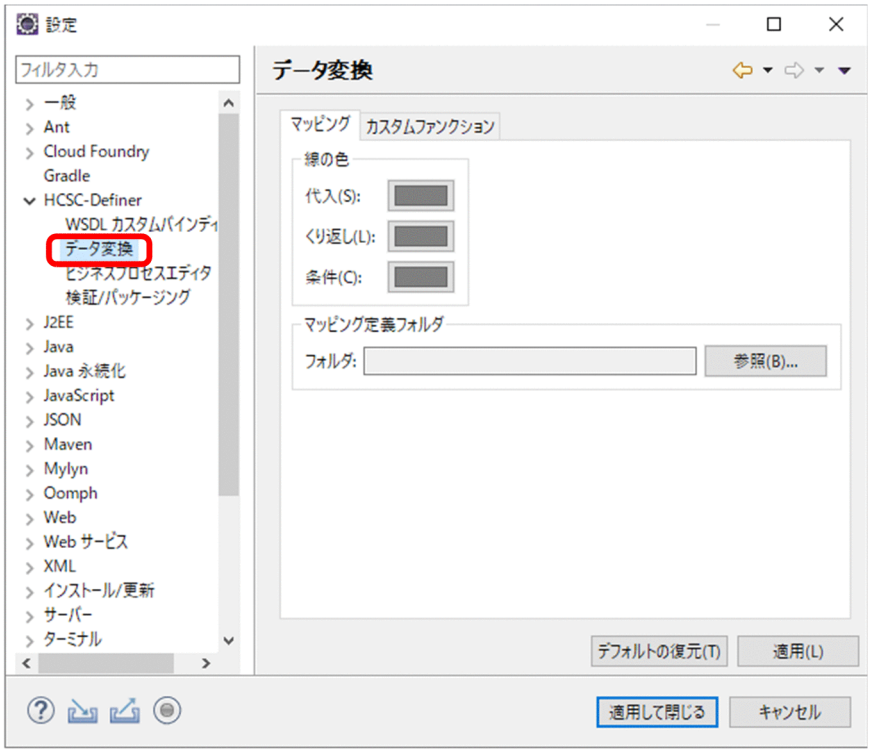
7.18.6　使用手順

サービスプラットフォームで提供しているカスタムファンクションの使用手順を次に示します。

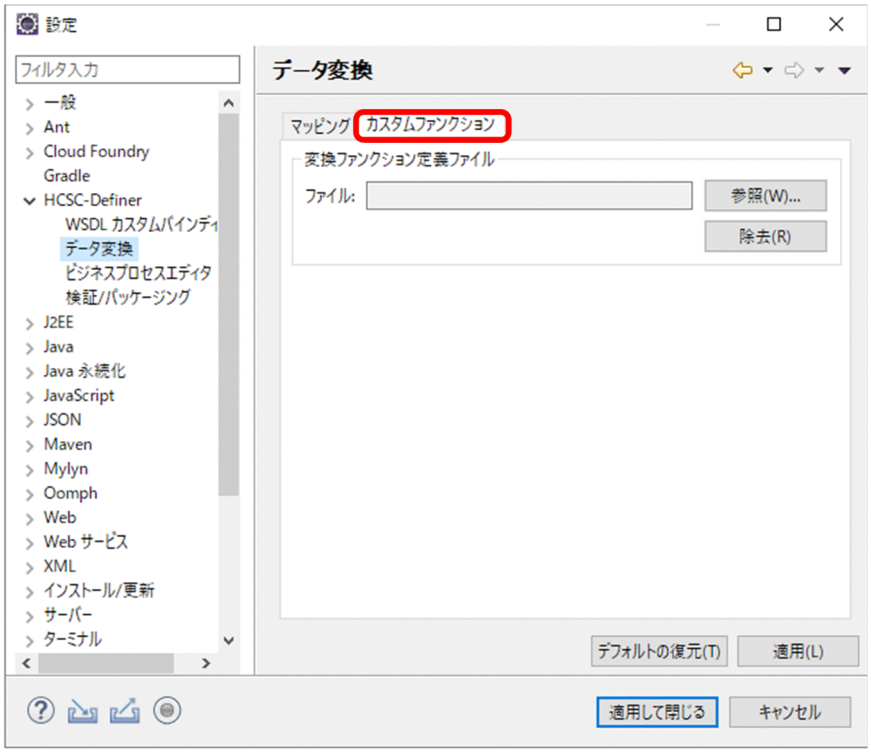
(1)　開発環境の設定

1. Eclipse のメニューから［ウィンドウ］－［設定］を選択します。
   ［設定］ダイアログが表示されます。
2. 左ペインで［HCSC-Definer］－［データ変換］を選択します。

データ変換定義画面の設定項目が表示されます。



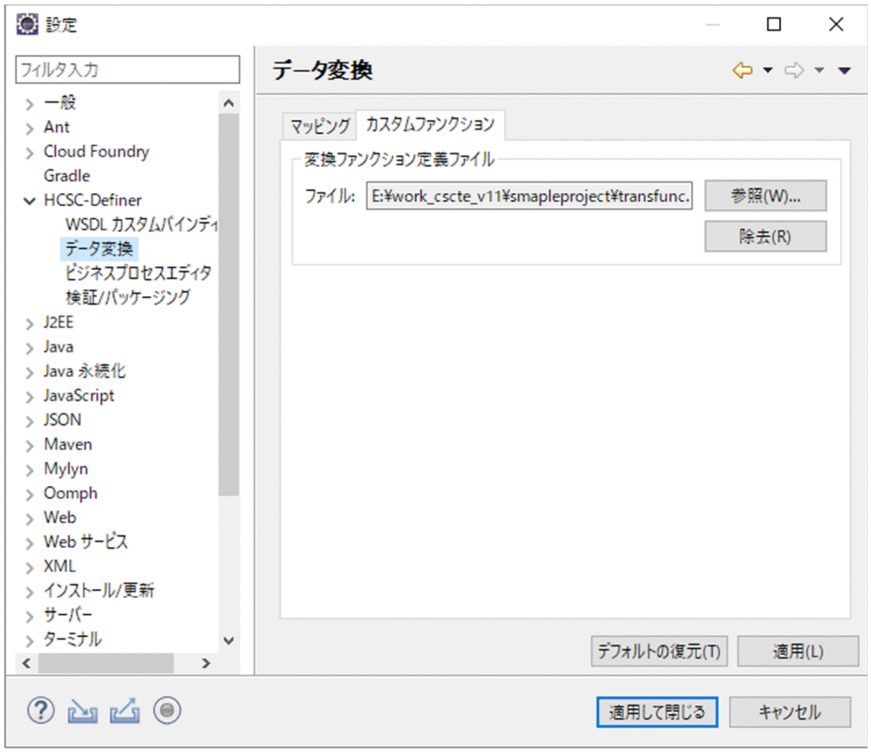
3. [カスタムファンクション] タブをクリックします。
   [設定] ダイアログの [カスタムファンクション] タブが表示されます。



4. [参照] ボタンをクリックします。
   ファイルの選択画面が表示されます。

5. <サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥customfunc ディレクトリに格納されている変換ファンクション定義ファイル（CSCCustomFunc.xml）を選択し，［OK］ボタンをクリックします。

   変換ファンクション定義ファイルが選択されます。



6. ［適用して閉じる］ボタンをクリックします。

   変換ファンクション定義ファイルが登録されます。

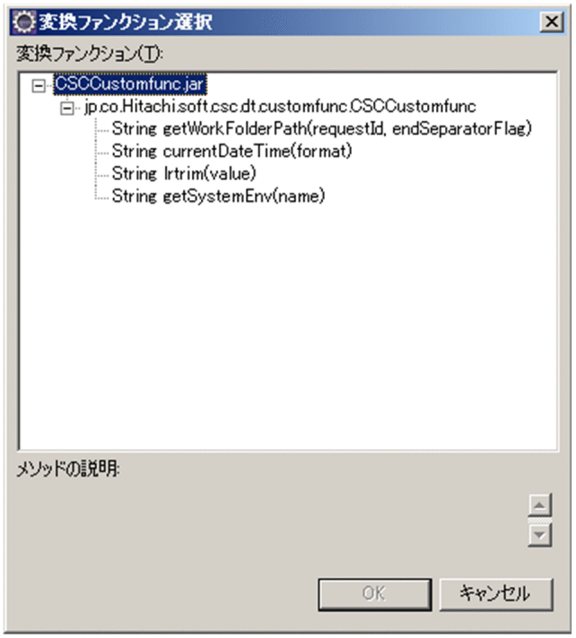
7. マッピング定義のパレットからカスタムファンクション（custom）を選択し，マッピングビューアに配置します。

8. 次のどちらかの方法で［カスタム］ダイアログを表示させます。

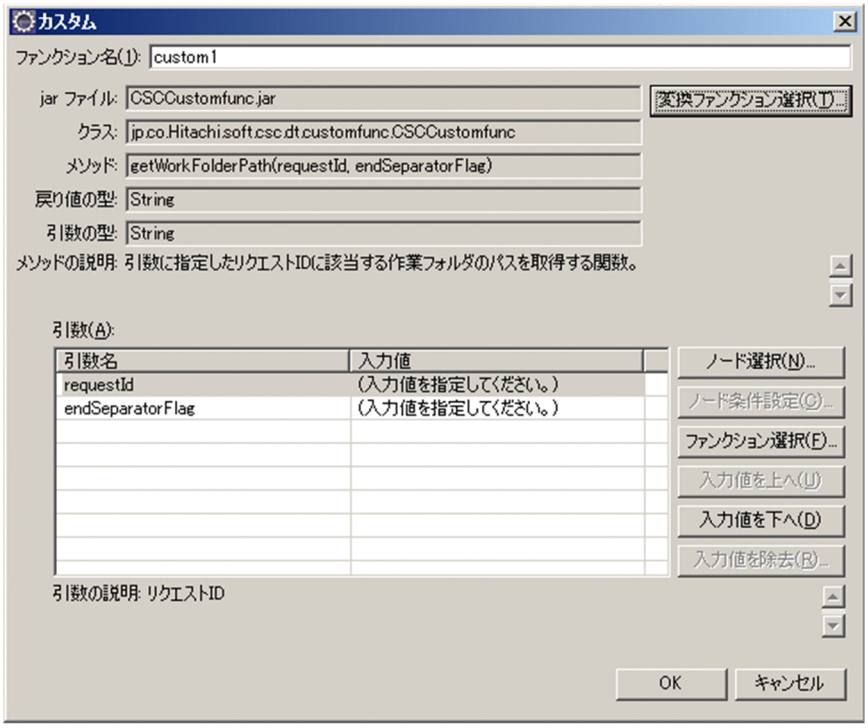
   - カスタムファンクションを右クリックして，［設定］を選択します。
   - カスタムファンクションをダブルクリックします。

9. ［変換ファンクション選択］ボタンをクリックします。

   ［変換ファンクション選択］ダイアログが表示されます。



10. カスタムファンクションから呼び出すメソッドを選択し，［OK］ボタンをクリックします。
    ［変換ファンクション選択］ダイアログが閉じ，［カスタム］ダイアログが表示されます。



11. [引数名］に対応する入力値を［入力値］に指定します。
    指定した入力値が引数に設定されます。
12. [OK］ボタンをクリックします。
    カスタムファンクションが設定されます。

(2) 実行環境の設定

1. <サービスプラットフォームのインストールディレクトリ>¥CSCTE¥resources¥customfunc¥userlib ディレクトリに格納されている jar ファイル（CSCCustomfunc.jar）を HCSC サーバに格納します。
   HCSC サーバに jar ファイルを格納する方法の詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.3.8　カスタムファンクションで使用する jar ファイルを格納する」を参照してください。
2. (1）で作成したマッピング定義を含むサービスを配備定義し，開始します。
   配備定義方法の詳細については，「8.5.1　配備から開始までの処理の流れ」を参照してください。
3. サービスを実行すると，カスタムファンクションが実行されます。

7.19　XPath ファンクションの使用方法

ここでは，データ変換定義で利用可能な XPath ファンクションの使用方法について説明します。XPath ファンクションの詳細は，「7.5.24　変換先ノードやファンクションの入力値に XPath 式を指定する」を参照してください。

7.19.1　変換先ノードへの代入

XPath ファンクションを使用して変換先ノードに値を代入する方法を説明します。

(1)　単純型ノード値の代入

単純型ノードの値を変換先ノードへ代入する場合は，単純型ノードのロケーションパスを XPath ファンクションに指定します。

参考

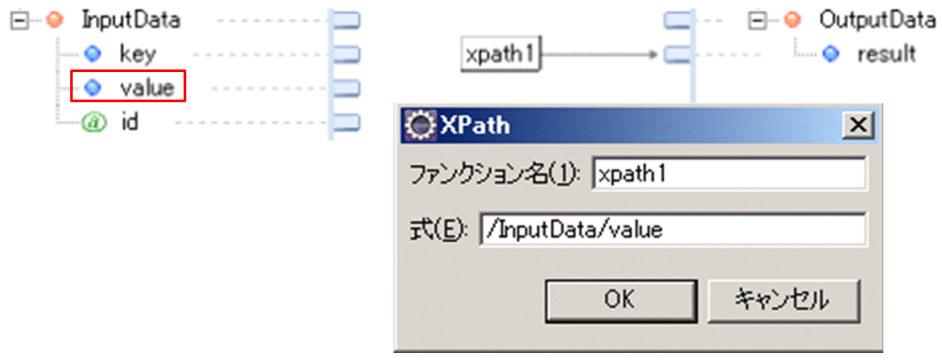
ロケーションパスは，XML 文書の階層構造の中から特定のノードを指定するための記述方法です。

child（軸） :: node（ノードテスト） [position() = 2]（述語）

ロケーションパスの詳細な仕様については，「XML Path Language (XPath) Version 1.0 - 2 Location Paths」を参照してください。

次に代入例を示します。

図 7-99　単純型ノード値を代入する例



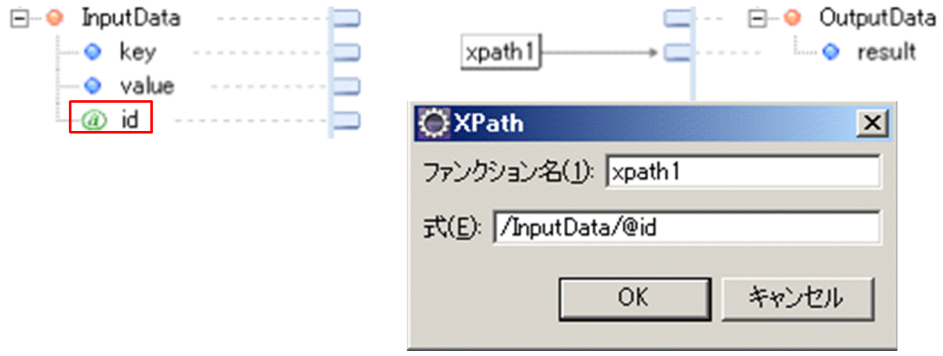
上記の例では，変換元の「/InputData/value」の値が変換先の「/OutputData/result」に代入されます。

(2)　属性ノード値の代入

属性ノードの値を変換先ノードへ代入する場合は，属性ノードのロケーションパスを XPath ファンクションに指定します。属性ノードは「@属性名」と表記します。

次に代入例を示します。

図 7-100　属性ノード値を代入する例



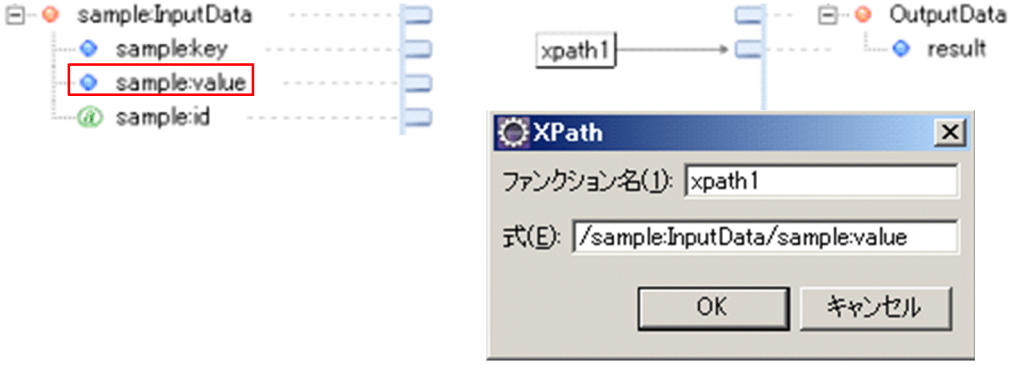
上記の例では，変換元の「/InputData/@id」の値が変換先の「/OutputData/result」に代入されます。

(3)　名前空間に属するノード値の代入

名前空間に属するノードの値を変換先ノードへ代入する場合は，名前空間接頭辞を含んだロケーションパスを XPath ファンクションに指定します。

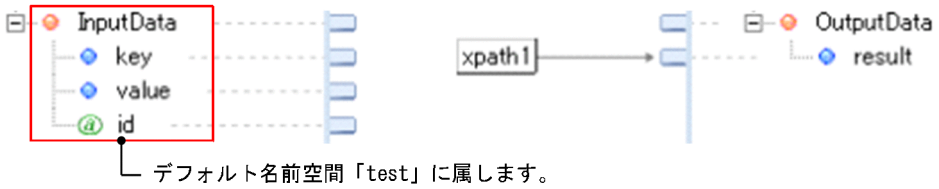
次に代入例を示します。

図 7-101　名前空間に属するノード値を代入する例

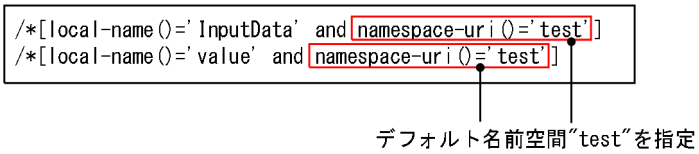


上記の例では，変換元の「/sample:InputData/sample:value」の値が変換先の「/OutputData/result」に代入されます。

デフォルト名前空間に属するノードの値を変換先ノードへ代入する場合は，名前空間 URI を明示的に指定する必要があります。



上記の例では，変換元の「/InputData/value」の値を変換先の「/OutputData/result」に代入する場合，次の XPath 式となります。

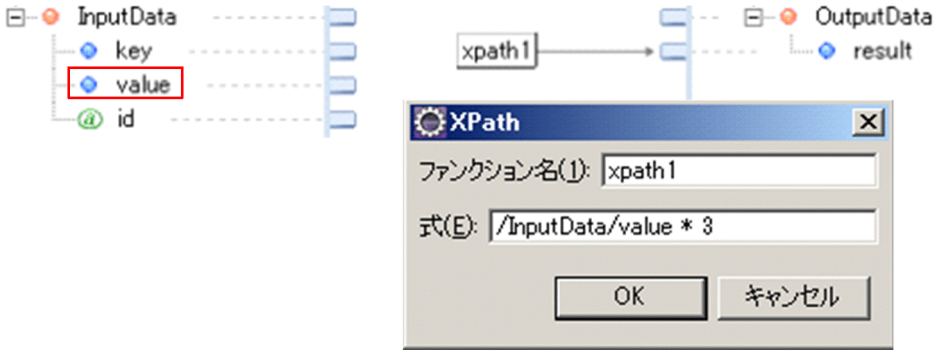


(4) 演算結果の代入

演算子を使用することで変換先ノードへ演算の結果を代入することができます。

次に代入例を示します。

図 7-102 演算結果を代入する例



上記の例では，変換元の「/InputData/value」の 3 倍の値が変換先の「/OutputData/result」に代入されます。

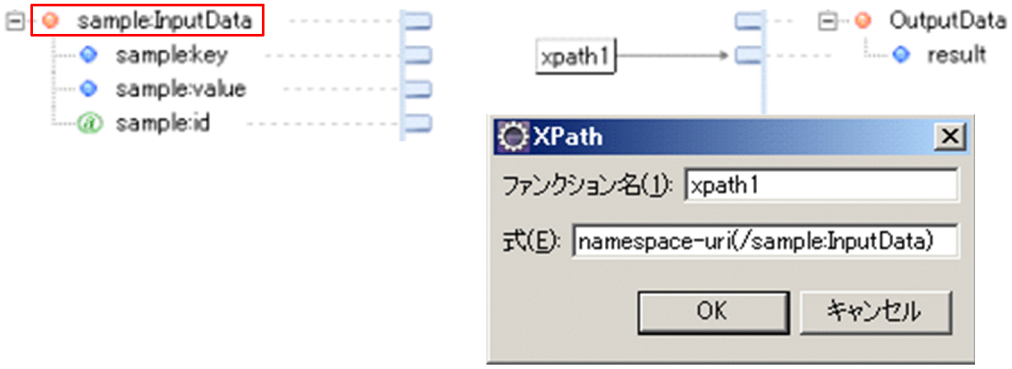
演算子の仕様については，「XML Path Language (XPath) Version 1.0 - 3 Expressions」を参照してください。

(5) XPath 関数の結果の代入

XPath 関数の結果を変換先ノードへ代入することができます。

次に代入例を示します。

図 7-103　XPath 関数の結果を変換先ノードへ代入する例



上記の例では，namespace-uri 関数の結果(変換元 sample:InputData の名前空間 URI)が変換先の「/OutputData/result」に代入されます。

XPath 関数の仕様については，「XML Path Language (XPath) Version 1.0 - 4 Core Function Library」を参照してください。

(6)　複数変数指定時のノード値の代入

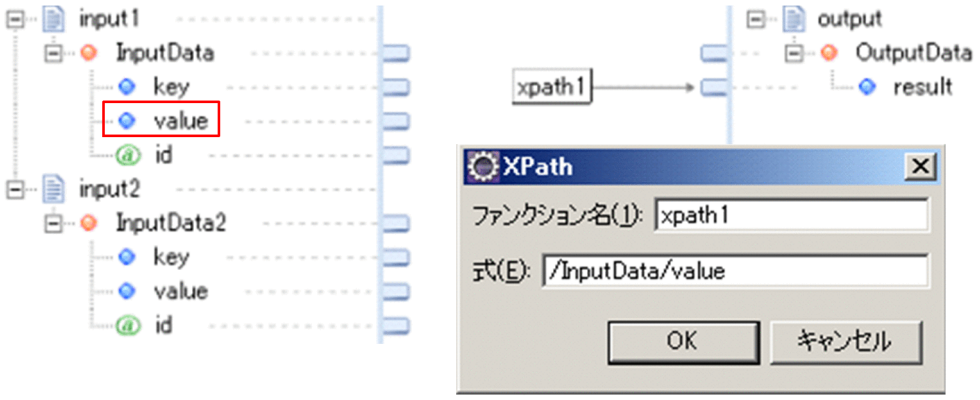
変換元に複数の変数を指定している場合のノード値の代入方法を次に示します。

(a)　1 番目に定義されている変数のノード値

1 番目に定義されている変数のノード値の代入は，(1)～(3)で示した方法と同じです。

次に代入例を示します。

図 7-104　1 番目に定義されている変数のノード値を代入する例

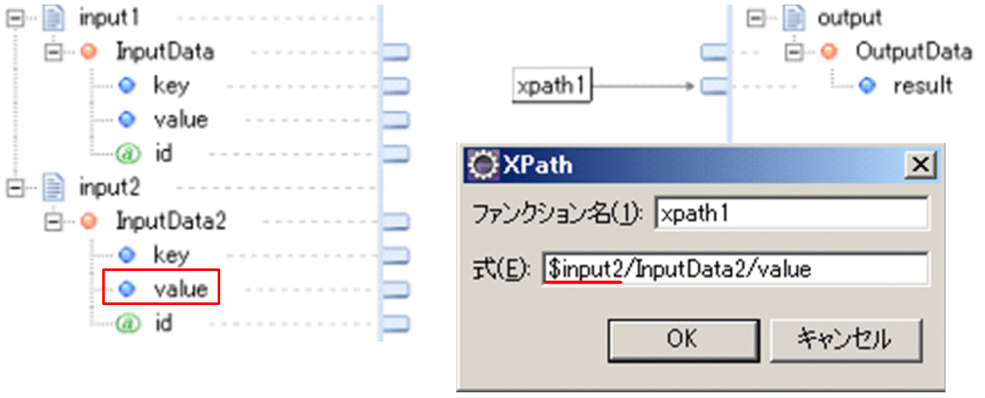


(b)　2 番目以降に定義されている変数のノード値

2 番目以降に定義されている変数のノード値を代入する場合は，(1)～(3)で示したロケーションパスの先頭に”$変数名”を付与します。

次に代入例を示します。

図 7-105　2 番目以降に定義されている変数のノード値を代入する例

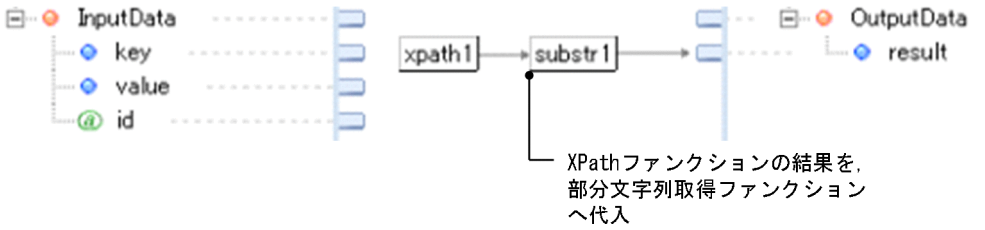


7.19.2　ファンクションへの代入

XPath ファンクションからほかのファンクションへマッピング線を引くことで XPath 式の評価結果をほかのファンクションに代入することができます。

次に代入例を示します。

図 7-106　XPath 式の評価結果をほかのファンクションに代入する例



代入可能なファンクションについては，「7.11.2　マッピングできるノードおよびファンクションの対応」を参照してください。

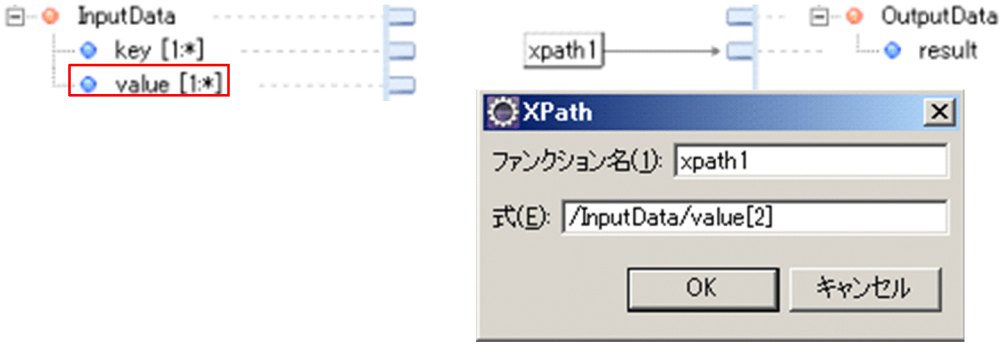
7.19.3　くり返し出現するノードの選択

くり返し出現するノードの中から特定のノードを選択する方法を説明します。

くり返し出現するノードの中から特定のノードを選択する場合は，ロケーションパスの述語に出現位置を指定します。

次に選択例を示します。

図 7-107　くり返し出現するノードの中から特定のノードを選択する例



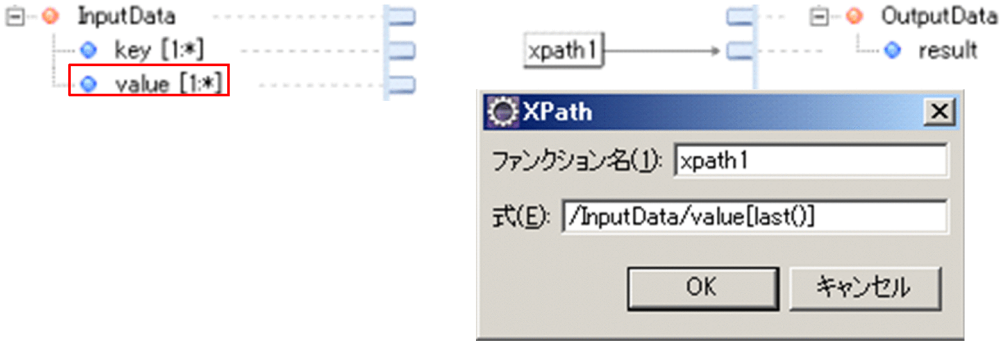
上記の例では，「/InputData/value」の述語に”[2]”と指定しているため，くり返し出現する value の 2 番目のノードが選択されます。述語の”[2]”は，”[position()=2]”と同義です。

出現位置は 1 から始まるため，1 番目のノードを選択したい場合，述語は”[1]”となります。述語を省略した場合は 1 番目のノードが選択されます。

最後のノードを選択したい場合は，XPath の last 関数を使用することで選択できます。

次に使用例を示します。

図 7-108　XPath に last 関数を使用する例



7.19.4　くり返しファンクションとの連携

XPath ファンクションとくり返しファンクションを組み合わせて使用する方法を説明します。

くり返しファンクションによって処理されるノードを XPath ファンクションで指定する場合は，相対ロケーションパスで指定します。

参考

ロケーションパスには，相対ロケーションパスと絶対ロケーションパスがあります。

- 相対ロケーションパス

あるノード（カレントノードなど）を起点として対象のノードまでの経路を指定する方法です。

- 絶対ロケーションパス
  ルートノードから対象のノードまでの経路を指定する方法です。

ロケーションパスの詳細な仕様については，「XML Path Language (XPath) Version 1.0 - 2 Location Paths」を参照してください。

(1)　くり返しの基準パスが単純型ノードの場合

くり返しファンクションの基準パスが単純型ノードである場合，その単純型ノードの相対ロケーションパスは”.”になります。”.”は自分自身のノードを表します。”self::node()”と同義です。

次に指定例を示します。

図 7-109　くり返しの基準パスが単純型ノードの場合の例



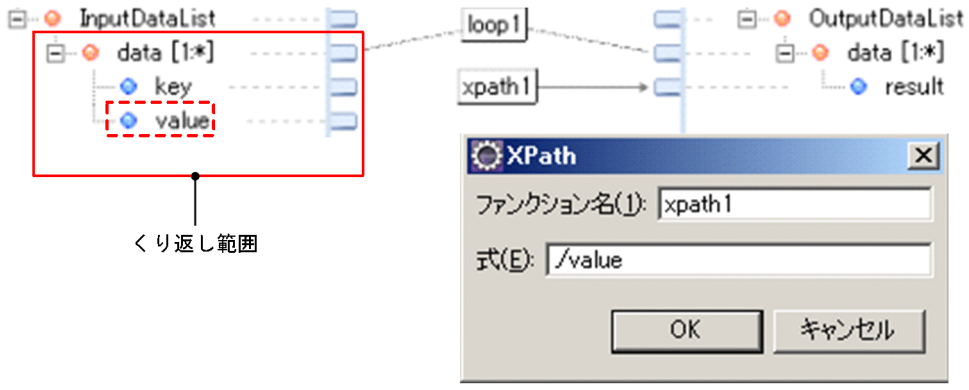
上記の例では,”.”がくり返しファンクションで処理される value1 つを表現しています。この場合,「/InputData/value」の値が 1 番目から順番に「/OutputData/result」へ代入されます。

(2)　くり返しの基準パスが複合型ノードの場合

くり返しファンクションの基準パスが複合型ノードである場合，その複合型ノードの相対ロケーションパスは”.”になります。複合型ノードの子ノードは，複合型ノードからの相対ロケーションパスで指定します。

次に指定例を示します。

図 7-110　くり返しの基準パスが複合型ノードの場合の例

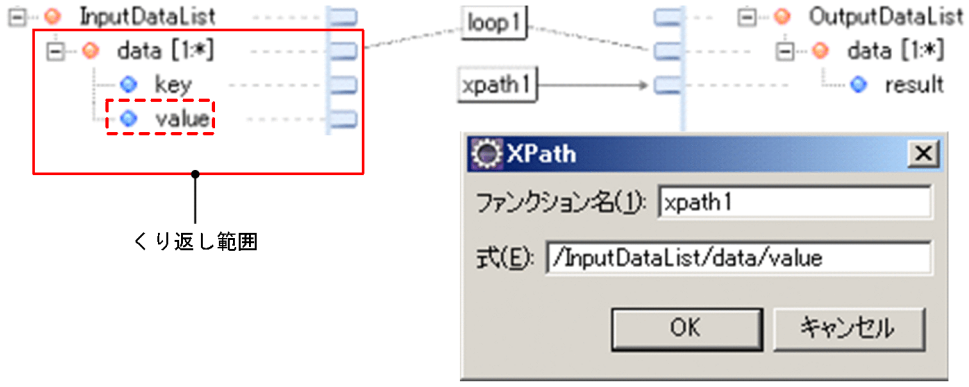


上記の例では，くり返しファンクションの範囲内にある data の子ノード value を相対ロケーションパス”./value”で指定しています。この場合，「/InputDataList/data/value」の値が 1 番目から順番に「/OutputDataList/data/result」へ代入されます。

絶対ロケーションパスで指定した場合は，くり返しファンクションと連動した値の代入が行われません。

次に指定例を示します。

図 7-111　絶対ロケーションパスで指定した場合の例



上記の例では，くり返しファンクションの範囲内にある data の子ノード value を絶対ロケーションパスで指定しているため，「/InputDataList/data/value」の 1 番目の値がくり返し代入されます。

(3)　くり返しファンクションとの連携例

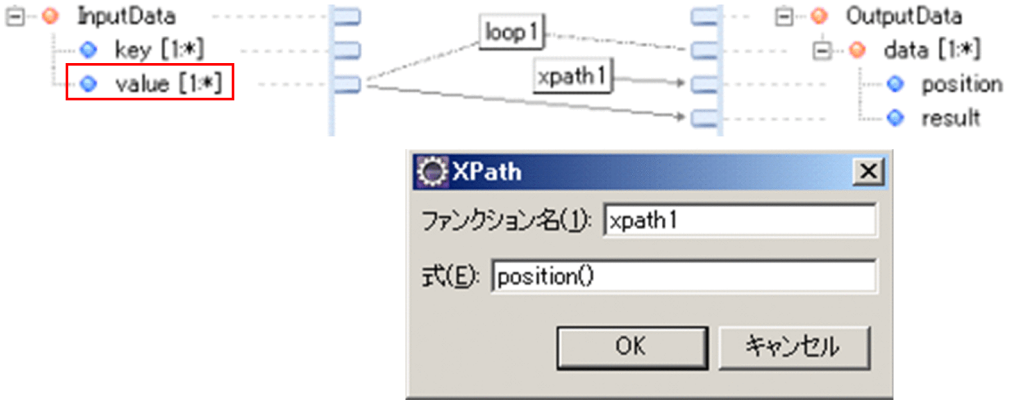
XPath ファンクションとくり返しファンクションの連携例を次に示します。

(a)　出現位置を代入する

XPath の position 関数を使用することでくり返しファンクションで処理されるノードの出現位置を代入することができます。

次に指定例を示します。

図 7-112　XPath の position 関数を指定する例



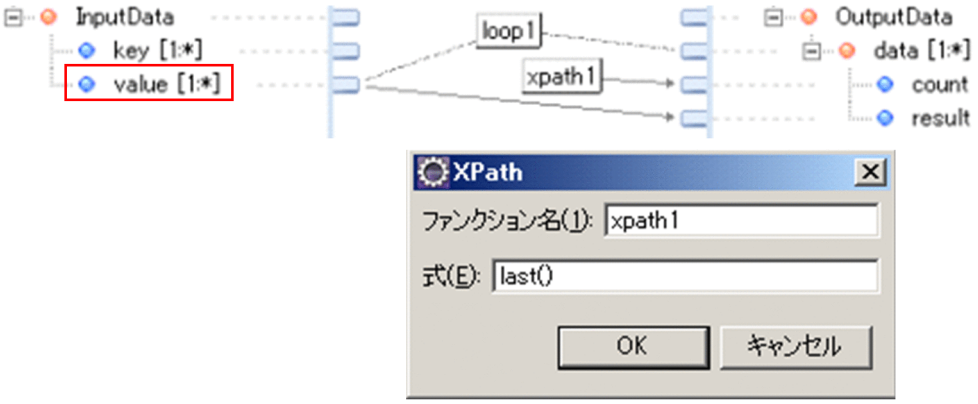
上記の例では，くり返しファンクションで処理される「/InputData/value」の出現位置(”1”,”2”，…)が「/OutputData/data/position」へ代入されます。

(b)　ノード数（くり返し回数）を代入する

XPath の last 関数を使用することでノード数(くり返し回数)を代入することができます。

次に指定例を示します。

図 7-113　XPath の last 関数を指定する例



上記の例では，くり返しファンクションで処理される「/InputData/value」のノード数が「/OutputData/data/count」へ代入されます。下記の入力データでは，”3”が代入されます。

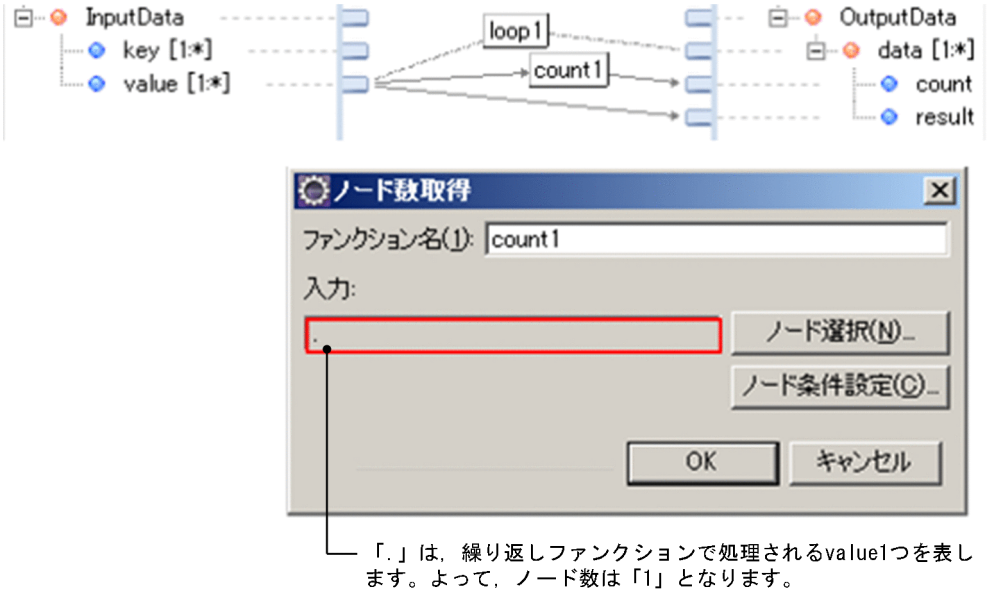
```
<?xml version="1.0" encoding="UTF-8"?>
<InputData>
 <key>key1</key>
 ...
 <value>value1</value>
 <value>value2</value>
 <value>value3</value>
</InputData>
```

ノード数：3

上記の例において，last 関数と同様の機能であるノード数取得ファンクションを使用した場合は，くり返しファンクションで処理される「/InputData/value」の一つ一つに対してノード数取得処理が実行されます。そのため，常に”1”が代入されます。

次に使用例を示します。

図 7-114　ノード数取得ファンクションを使用する例

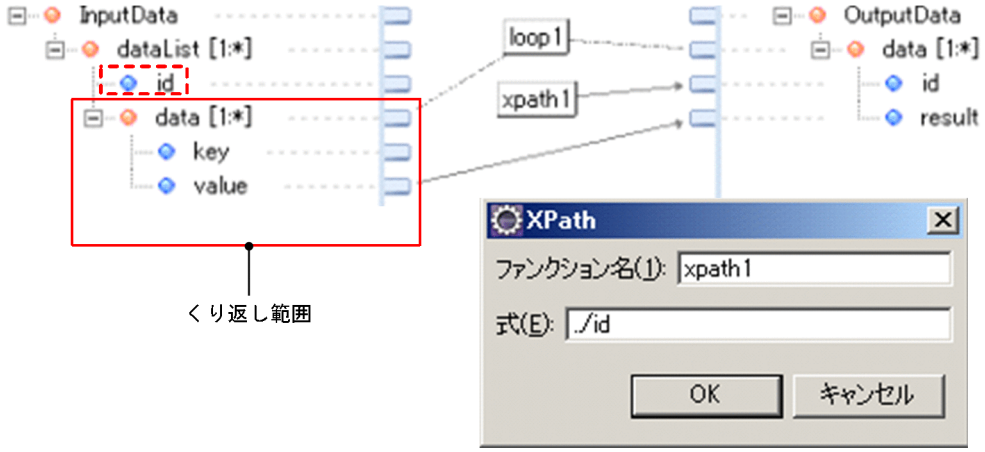


(c)　くり返し範囲外のノード値を代入する

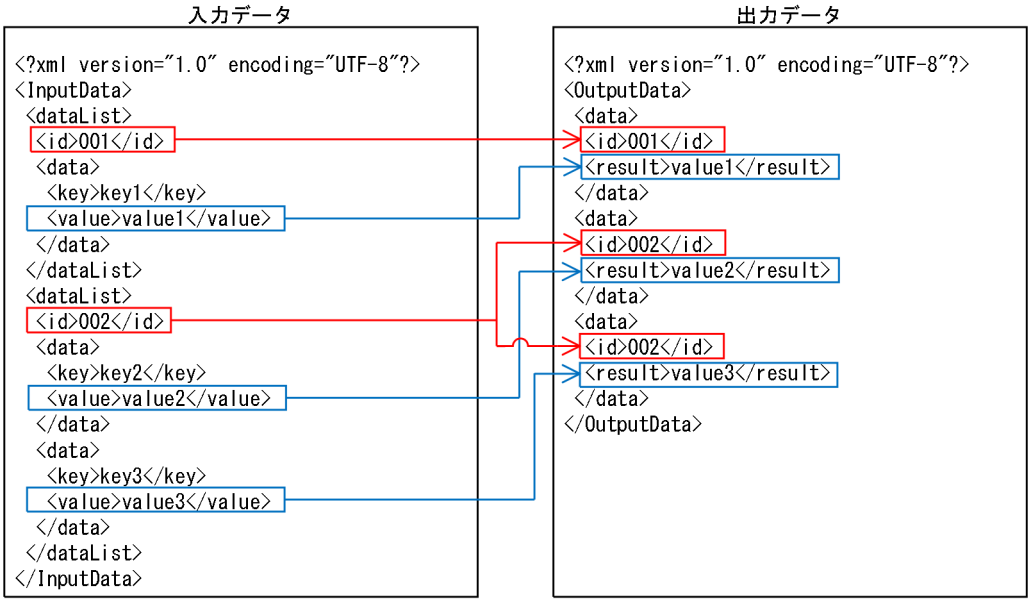
くり返しファンクションの範囲外にあるノードについて，相対ロケーションパスを使用することでくり返し処理と連動したノード値を代入することができます。

次に指定例を示します。

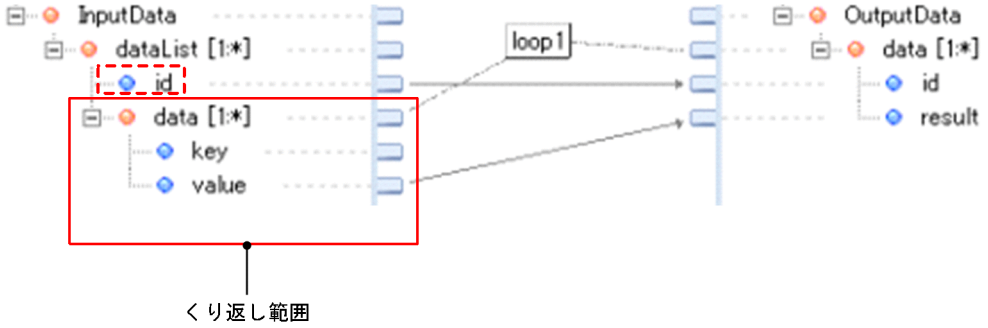
くり返し範囲外のノード値を代入する例

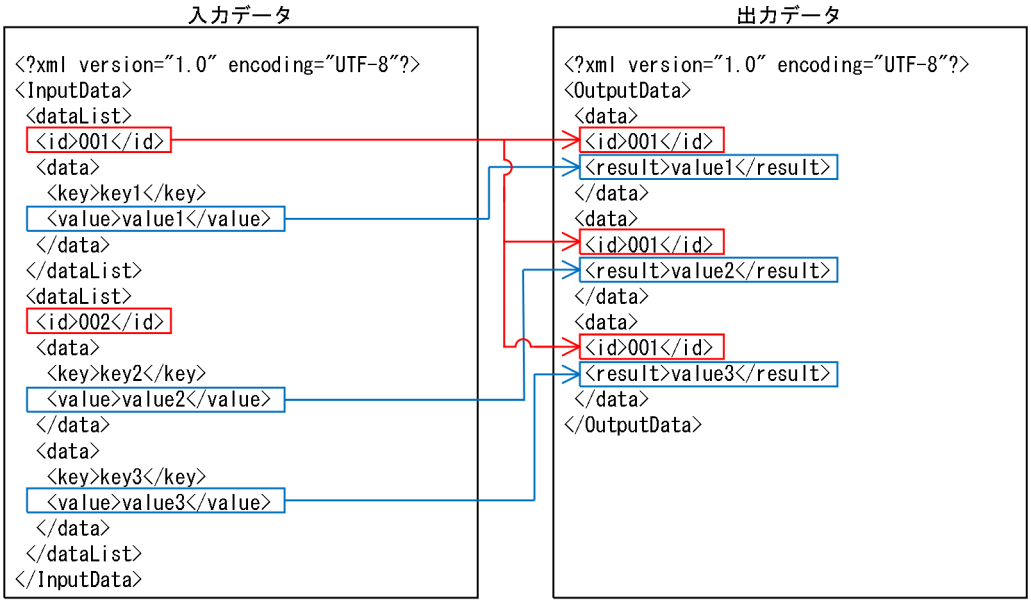


上記の例では，くり返しファンクションの範囲外にある id を基準パスである data から相対ロケーションパス(”../id”)で指定しています。この場合，「/InputData/dataList/data」に対応した「/InputData/dataList/id」の値が変換先へ代入されます。



絶対ロケーションパスで指定した場合やマッピング線で代入した場合は，くり返し処理と連動しないため，「/InputData/dataList/id」の 1 番目の値が変換先へ代入されます。



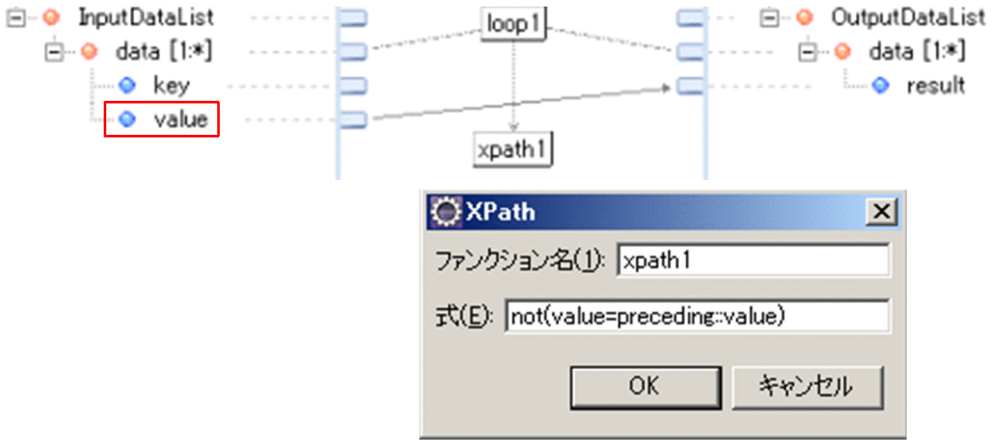


(d)　くり返し条件を XPath 式で指定する

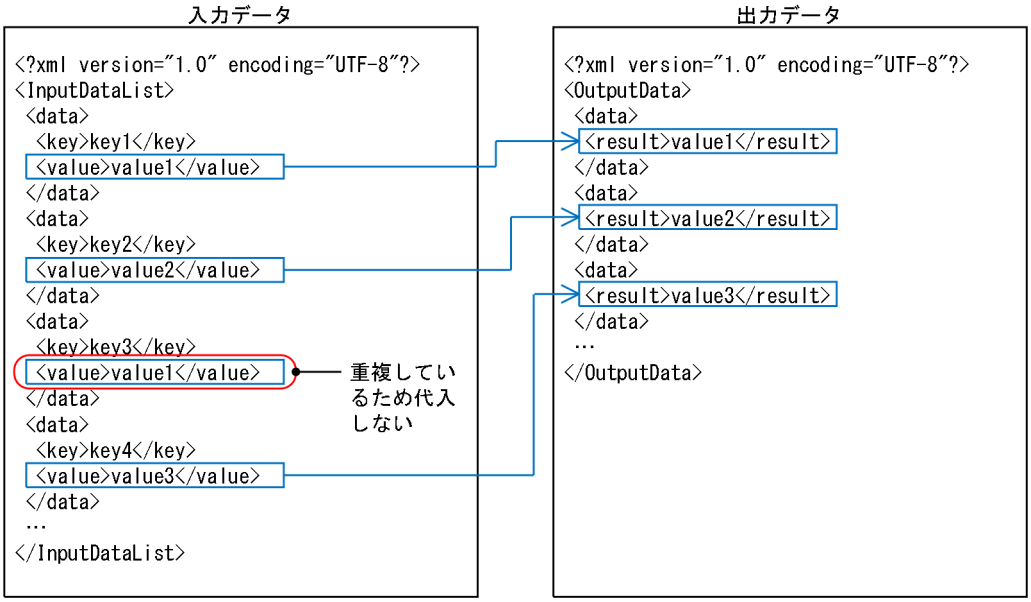
くり返しファンクションのノード条件に XPath ファンクションを指定することで，任意の XPath 式でくり返し処理を制御することができます。

次に指定例を示します。

くり返し条件を XPath 式で指定する例



上記の例では，くり返しファンクションのノード条件に not 関数，および preceding※を使用することで，「/InputDataList/data/value」の中で重複している値は代入しないように制御しています。



注※

preceding は，コンテキストノードより文書順で前にあるノードをすべて選択する軸です。preceding の詳細な仕様については，「XML Path Language (XPath) Version 1.0 - 2.2 Axes」を参照してください。

8 HCSC コンポーネントのパッケージングと配備定義

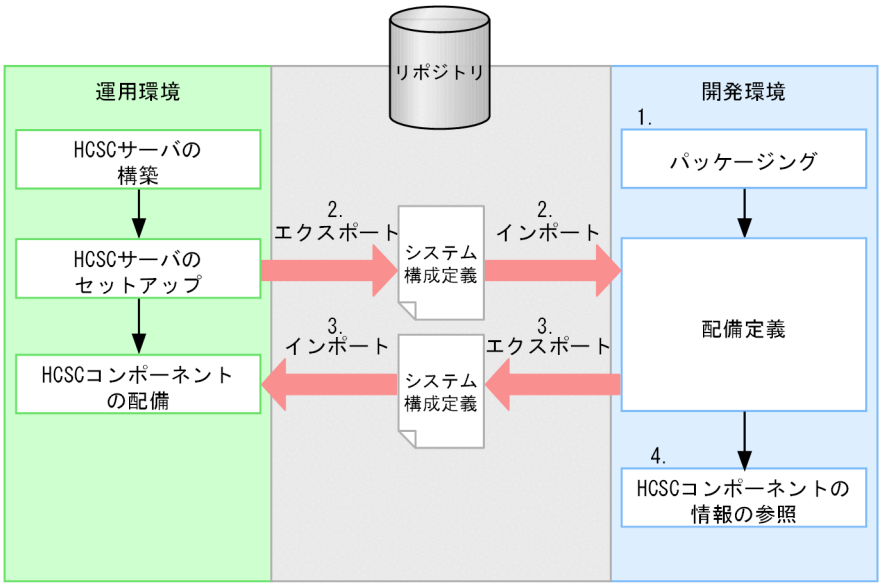
この章では，HCSC コンポーネントのパッケージングと配備定義について説明します。

8.1　パッケージングと配備定義の概要

定義した HCSC コンポーネントは，パッケージングしたり，どのクラスタ（または単一の HCSC サーバ）に配備するか決めたり（配備定義）します。

パッケージングから配備定義までの流れを次の図に示します。

図 8-1　パッケージングから配備定義までの流れ

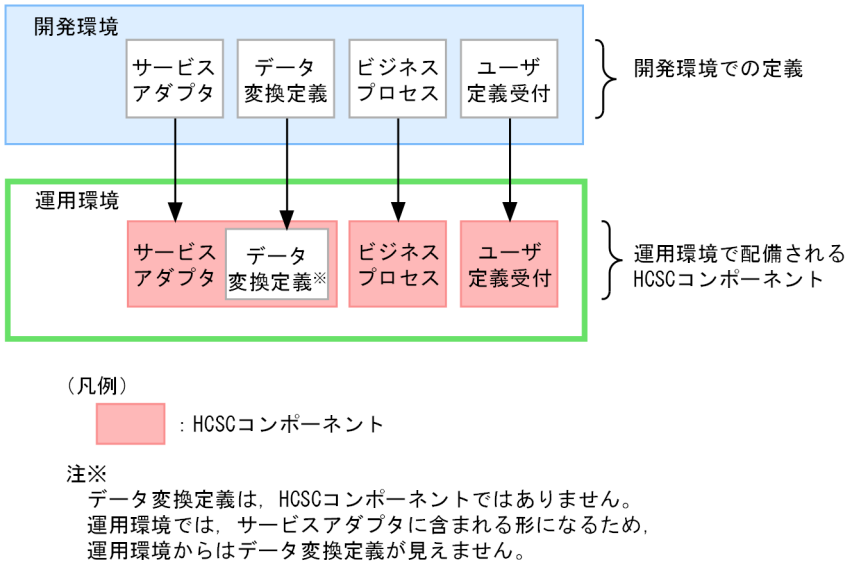


1. 開発環境で HCSC コンポーネントをパッケージングします。
2. 配備定義の前に，開発環境へリポジトリをインポートします。
3. 配備定義の実施後，リポジトリをエクスポートして，配備定義したシステム構成定義を運用環境に提供します。

   運用環境はシステム構成定義の内容に従って，実行環境のクラスタ（または単一の HCSC サーバ）に HCSC コンポーネントを配備します。
4. 開発環境では，配備定義をしたあとに，配備定義をした HCSC コンポーネントの情報を参照できます。

開発環境で定義した内容は，運用環境で HCSC コンポーネントとして配備されます。開発環境での定義と運用環境で配備される HCSC コンポーネントの関係を次の図に示します。

図 8-2　開発環境での定義と運用環境で配備される HCSC コンポーネントの関係



なお，次の一連の処理は，開発環境で一括して実行することもできます。開発環境または運用環境で一つ一つ実施していた作業を開発環境で一括実行するため，ユーザの操作負担が軽減できます。

1. 開発環境での HCSC コンポーネントのパッケージングと配備定義
2. 運用環境での HCSC コンポーネントの配備と開始

ただし，一括実行は，システム開発時，または単体テストから結合テスト時を対象としています。詳細については，「8.5　HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行」および「8.6　HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行」を参照してください。

注意事項

- リポジトリの一部の情報だけをインポートした場合には，インポートしたリポジトリ情報に含まれる HCSC コンポーネントをパッケージングする必要があります。

  リポジトリの一部の情報だけのインポートについては，「3.2.3　リポジトリをインポートする」を参照してください。
- ワークスペースのビルドやクリーンをしている間は，HCSCTE 上でパッケージングなどの操作を実施しないでください。
- サービスのオペレーション名を変更した場合は，そのサービスを呼び出しているビジネスプロセスを再パッケージングしてください。

8.2 パッケージング

実行環境に配備するために必要な HCSC コンポーネントに関するファイルを組み立てて，EAR ファイルを作成します。これを**パッケージング**と呼びます。ここでは，サービスアダプタまたはビジネスプロセスをまとめて HCSC コンポーネントと呼びます。

HCSC コンポーネントのパッケージングに伴い，パッケージング対象の HCSC コンポーネントと関連するデータ変換定義およびユーザ定義受付もパッケージングされます。

パッケージングを実行すると，HCSC コンポーネントが検証されます。よって，HCSC コンポーネントが正しく定義されている場合はパッケージングできますが，正しく定義されていない場合はパッケージング時の検証でエラーが発生します。パッケージングの前に，あらかじめ，HCSC コンポーネントを検証しておくと，円滑にパッケージングできます。

パッケージングの方法を次に示します。

1. ツリービューのサービス定義一覧で，パッケージングする HCSC コンポーネントを右クリックして，「パッケージング」または「複数パッケージング」を選択します。
   - 「パッケージング」を選択した場合
     メッセージダイアログが表示されます。
   - 「複数パッケージング」を選択した場合
     [複数パッケージング] ダイアログが表示されます。チェックボックスからパッケージングするサービスをチェックして，[OK] ボタンをクリックします。
2. 次のどちらかの操作をします。
   - パッケージングが成功した場合
     [OK] ボタンをクリックします。
     すでに EAR ファイルがある場合，上書きを確認するダイアログが表示されます。上書きしてよいときは，[はい] ボタンをクリックします。
   - パッケージングが失敗した場合
     ダイアログのメッセージに従って対処し，パッケージングを再実行します。

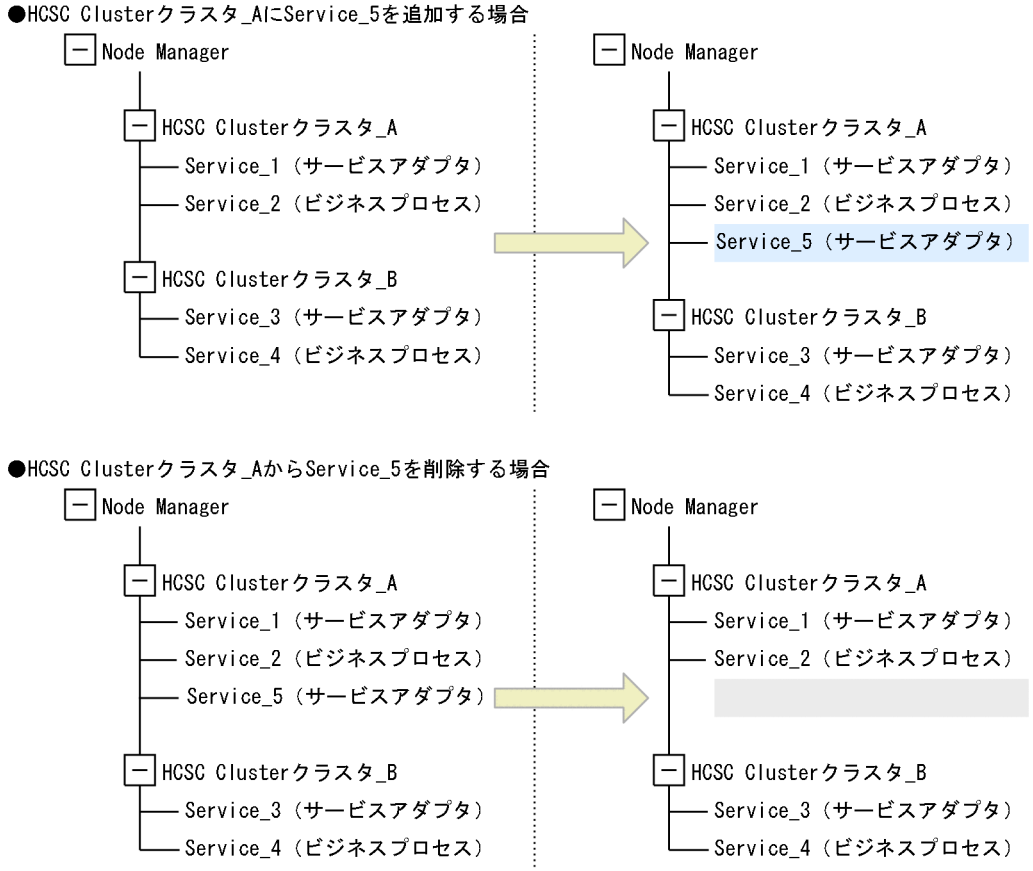
なお，パッケージングする前にサービスアダプタ，ビジネスプロセスおよびユーザ定義受付が保存されていないときは，リソース保管ダイアログが表示され，定義を保存できます。複数のサービスアダプタ，ビジネスプロセスおよびユーザ定義受付が編集中の場合は，複数回ダイアログが表示されます。保存する場合は［はい］ボタンをクリックします。これ以降，確認ダイアログを表示しないで，すべて保存する場合は［すべてはい］ボタンをクリックします。

8.3　HCSC コンポーネントの配備定義

配備定義とは，定義した HCSC コンポーネントをどのクラスタ（または単一の HCSC サーバ）に配備するか決めることです。

ツリー形式で表示されるツリービューのシステム構成定義一覧で，クラスタ（または単一の HCSC サーバ）の下位に表示される HCSC コンポーネントは，そのクラスタ（または単一の HCSC サーバ）に配備するよう定義されていることを表します。ツリービューのシステム構成定義一覧のクラスタ（または単一の HCSC サーバ）の下位に，配備する HCSC コンポーネントを追加・削除して配備定義を行います。HCSC コンポーネントの追加と削除の例を次の図に示します。

図 8-3　HCSC コンポーネントの追加と削除



開発環境で配備定義し，更新されたシステム構成定義の内容に従って，運用環境でクラスタ（または単一の HCSC サーバ）へ HCSC コンポーネントが配備されます。クラスタ（または単一の HCSC サーバ）への配備方法の詳細については，次の個所を参照してください。

- マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.1.8　サービスアダプタを配備する」
- マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「3.1.13　ビジネスプロセスを配備する」

ここではクラスタ（または単一の HCSC サーバ）に HCSC コンポーネントを追加・削除して配備定義する方法を説明します。

なお，システム構成定義で定義されているサーバ構成とサービスの組み合わせによっては，配備定義できない場合があります。システム構成定義で定義されているサーバ構成とサービスの組み合わせによる配備定義の可否を次の表に示します。

表 8-1　配備定義の可否

項番	システム構成定義で定義されているサーバ構成		サービスの種別	配備定義の可否
	データベースの使用有無	Reliable Messaging の使用有無		
1	使用する	使用する	すべて	○
2	使用しない	使用しない	ビジネスプロセス（永続化する）	×
3			ビジネスプロセス（永続化しない）	○
4			SOAP アダプタ	○
5			SessionBean アダプタ	○
6			MDB（WS-R）アダプタ	×
7			MDB（DB キュー）アダプタ	×
8			DB アダプタ	○
9			TP1 アダプタ	○
10			ファイルアダプタ	○
11			Object Access アダプタ	○
12			Message Queue アダプタ	○
13			FTP アダプタ	○
14			ファイル操作アダプタ	○
15			メールアダプタ	○
16			HTTP アダプタ	○
17			コマンドアダプタ	○
18			SFTP アダプタ	○
19			Kafka アダプタ	○
20			gRPC アダプタ	○
21			汎用カスタムアダプタ	○
22	使用する	使用しない	ビジネスプロセス（永続化する）	○
23			ビジネスプロセス（永続化しない）	○

項番	システム構成定義で定義されているサーバ構成		サービスの種別	配備定義の可否
	データベースの使用有無	Reliable Messaging の使用有無		
24	使用する	使用しない	SOAP アダプタ	○
25			SessionBean アダプタ	○
26			MDB（WS-R）アダプタ	×
27			MDB（DB キュー）アダプタ	×
28			DB アダプタ	○
29			TP1 アダプタ	○
30			ファイルアダプタ	○
31			Object Access アダプタ	○
32			Message Queue アダプタ	○
33			FTP アダプタ	○
34			ファイル操作アダプタ	○
35			メールアダプタ	○
36			HTTP アダプタ	○
37			コマンドアダプタ	○
38			SFTP アダプタ	○
39			Kafka アダプタ	○
40			gRPC アダプタ	○
41			汎用カスタムアダプタ	○

（凡例）

○：配備定義できます。

×：配備定義できません。

8.3.1　配備できるクラスタ（または単一の HCSC サーバ）

配備できるクラスタ（または単一の HCSC サーバ）は，ツリービューのシステム構成定義一覧に表示されます。表示される情報は，運用環境でセットアップしたクラスタ（または単一の HCSC サーバ）の構成情報です。この情報は，運用環境でエクスポートしたリポジトリ情報を，開発環境でインポートして取得します。

リポジトリのインポートについては，「3.2.3　リポジトリをインポートする」を参照してください。

8.3.2　クラスタに HCSC コンポーネントを追加する

クラスタ（または単一の HCSC サーバ）に HCSC コンポーネントを追加する方法を次に示します。なお，HCSC コンポーネントをクラスタ（または単一の HCSC サーバ）に追加すると，追加した HCSC コンポーネントと関連するデータ変換定義およびユーザ定義受付もあわせて追加されます。

1. Eclipse のメニューから，［HCSC-Definer］－［システム構成定義一覧］を選択します。

   ツリービューのシステム構成定義一覧に，クラスタ（または単一の HCSC サーバ）の一覧が表示されます。

2. クラスタ（または単一の HCSC サーバ）の一覧から HCSC コンポーネントを追加するクラスタ（または単一の HCSC サーバ）を選択し，ダブルクリックします。

   クラスタの情報が表示されます。



3. 手順 2.で選択したクラスタ（または単一の HCSC サーバ）を右クリックして，［サービス追加］または［複数サービス追加］を選択します。

   - ［サービス追加］を選択した場合

     ［サービス追加］ダイアログが表示されます。

     定義済みの HCSC コンポーネントを，パッケージングしたあとに編集した場合，その HCSC コンポーネントを追加対象にしてよいか確認するダイアログが表示されます。追加対象にしてよいときは，［はい］ボタンをクリックします。

   - ［複数サービス追加］を選択した場合

     ［複数サービス追加］ダイアログが表示されます。

4. 追加するサービスを選択します。

   - ［サービス追加］を選択した場合

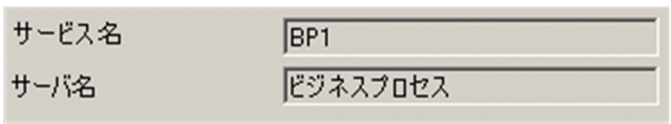
     追加する HCSC コンポーネントを［追加するサービス］のドロップダウンリストから選択します。

   - ［複数サービス追加］を選択した場合

     チェックボックスから追加するサービスをチェックします。

5. ［OK］ボタンをクリックします。

   追加した HCSC コンポーネント情報が表示されます。



8.3.3　クラスタから HCSC コンポーネントを削除する

クラスタ（または単一の HCSC サーバ）から HCSC コンポーネントを削除する方法を次に示します。なお，HCSC コンポーネントをクラスタ（または単一の HCSC サーバ）から削除すると，追加した HCSC コンポーネントと関連するデータ変換定義およびユーザ定義受付もあわせて削除されます。

1. ツリービューのシステム構成定義一覧から削除する HCSC コンポーネントを選択します。

2. 次のどちらかの方法で削除します。

   - HCSC コンポーネントを右クリックして，［サービス削除］または［複数サービス削除］を選択します。
   - ［Delete］キーを押します。

   ［複数サービス削除］を選択した場合は，［複数サービス削除］ダイアログが表示されます。チェックボックスから削除するサービスをチェックして，［OK］ボタンをクリックします。

   削除を確認するダイアログが表示されます。削除してよいときは，［はい］ボタンをクリックします。

8.4　HCSC コンポーネントの情報の参照

配備定義したあとに，配備定義をした HCSC コンポーネントの情報を参照できます。HCSC コンポーネントの情報は，HCSC コンポーネント情報表示画面に表示されます。HCSC コンポーネント情報表示画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.6　HCSC コンポーネント情報表示画面」を参照してください。

また，HCSC コンポーネントに含まれるユーザ定義受付の情報も参照できます。ユーザ定義受付の情報は，ユーザ定義受付情報表示画面に表示されます。ユーザ定義受付情報表示画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.7　ユーザ定義受付情報表示画面」を参照してください。

8.4.1　参照できる HCSC コンポーネントの情報

参照できる HCSC コンポーネントの情報を次に示します。

(1)　サービス部品の情報

サービス部品の情報として，次の情報を参照できます。

- インターフェース情報

  サービス名（HCSC コンポーネント名）とオペレーションの組み合わせに対応した，オペレーション情報，要求電文，応答電文，およびフォルト電文の情報を参照できます。
- サーバ情報

  サービス部品を呼び出すことができるクラスタ（または単一の HCSC サーバ）のサービス種別ごとの宛先を参照できます。
- オペレーション情報

  インターフェース情報で指定したオペレーションの名称，および通信モデルを参照できます。
- 要求電文

  標準電文フォーマット，および電文フォーマット ID を参照できます。
- 応答電文

  標準電文フォーマット，および電文フォーマット ID を参照できます。
- フォルト電文

  フォルト名，および電文フォーマットを参照できます。

(2)　ユーザ定義受付の情報

ユーザ定義受付の情報として，次の情報を参照できます。

- ユーザ定義受付情報

ユーザ定義受付の受付名，ポート名，および URL 情報を参照できます。

8.4.2 HCSC コンポーネントの情報を表示する

HCSC コンポーネントの情報を表示させる方法を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［公開サービス一覧］を選択します。
   ツリービューの公開サービス一覧に，HCSC コンポーネントの一覧が表示されます。

2. HCSC コンポーネントの一覧から，情報を表示する HCSC コンポーネントを選択し，ダブルクリックします。
   HCSC コンポーネント情報表示画面に，HCSC コンポーネントの情報が表示されます。

3. 次に示す操作をします。

   ［オペレーション情報］，［要求電文］，［応答電文］，および［フォルト電文］の情報を参照する場合

   ［インタフェース情報］の［オペレーション］のドロップダウンリストから，情報を参照するオペレーションを選択します。

   電文の内容を参照するときは，参照する電文（要求電文，応答電文，またはフォルト電文）の［表示］ボタンをクリックします。

   クラスタ（または単一の HCSC サーバ）のサービス種別ごとの宛先を参照する場合

   ［サーバ情報］の［サーバ名］に宛先を参照するクラスタ名が表示されます。

   Web サービスで使用する WSDL ファイルを取得するときは，［WSDL 取得］ボタンをクリックします。

   SessionBean で使用する RMI-IIOP スタブファイルを取得するときは，［スタブ取得］ボタンをクリックします。

   ユーザ定義受付の情報を参照する場合

   HCSC コンポーネント情報表示画面の下部の［ユーザ定義受付情報］タブをクリックして，ユーザ定義受付情報表示画面を表示させます。

8.4.3 HCSC コンポーネントの一覧を更新する

HCSC コンポーネントが変更された場合，リポジトリ内の情報とツリービューのシステム構成定義一覧に表示されている情報が不一致になる場合があります。このような場合，ツリービューのシステム構成定義一覧の表示内容を最新の情報に更新できます。

更新方法を次に示します。

1. ツリービューのシステム構成定義一覧で，表示されている HCSC コンポーネントを選択しないで右クリックして，［更新］を選択します。
   ツリービューのシステム構成定義一覧に表示されている HCSC コンポーネントの一覧が更新されます。

注意事項

ツリービューのシステム構成定義一覧に表示されるクラスタ（または単一の HCSC サーバ）の構成情報は，運用環境でセットアップしたクラスタ（または単一の HCSC サーバ）の構成情報です。

ツリービューのシステム構成定義一覧に最新のサーバ構成一覧を表示するには，必要に応じて，運用環境でエクスポートしたリポジトリ情報を開発環境でインポートします。インポートすると，自動的に表示が更新されます。

リポジトリのインポートについては，「3.2.3　リポジトリをインポートする」を参照してください。

8.5 HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行

HCSC コンポーネントを HCSC サーバに配備して開始する一連の処理を開発環境で一括実行できます。開発環境または運用環境で一つ一つ実施していた作業を開発環境で一括実行するため，ユーザの操作負担を軽減できます。なお，一括実行は，システム開発時，または単体テストから結合テスト時を対象としています。

ここでは，配備から開始までの処理の流れと操作方法について説明します。

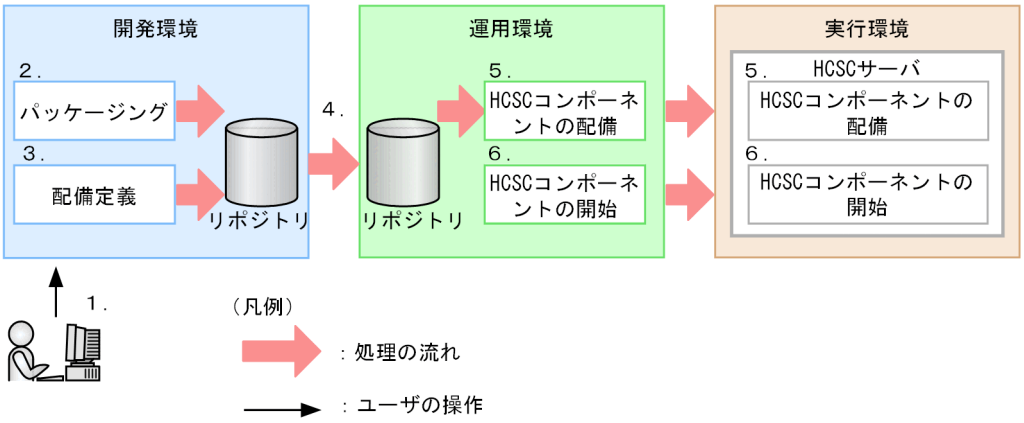
8.5.1 配備から開始までの処理の流れ

HCSC コンポーネントを HCSC サーバに配備して開始する一連の処理の流れ，操作対象となる HCSC コンポーネントの範囲，および HCSC コンポーネントの種類による処理の違いを次に説明します。

(1) 処理の流れ

HCSC コンポーネントを HCSC サーバに配備して開始する一連の処理の流れを次の図に示します。なお，図の 2.～6.は，ユーザが開発環境で操作すると，自動的に実行されます。

図 8-4 配備から開始までの処理の流れ



1. ユーザは開発環境で一括実行を実施します。
2. 開発環境で HCSC コンポーネントがパッケージングされ，リポジトリに格納されます。
3. 配備定義され，リポジトリ内のシステム構成定義が更新されます。
4. 開発環境のリポジトリが運用環境に転送されます。
5. 開発環境で更新されたシステム構成定義を基に，運用環境から実行環境の HCSC サーバに HCSC コンポーネントが配備されます。
6. 実行環境の HCSC サーバに配備された HCSC コンポーネントが開始されます。

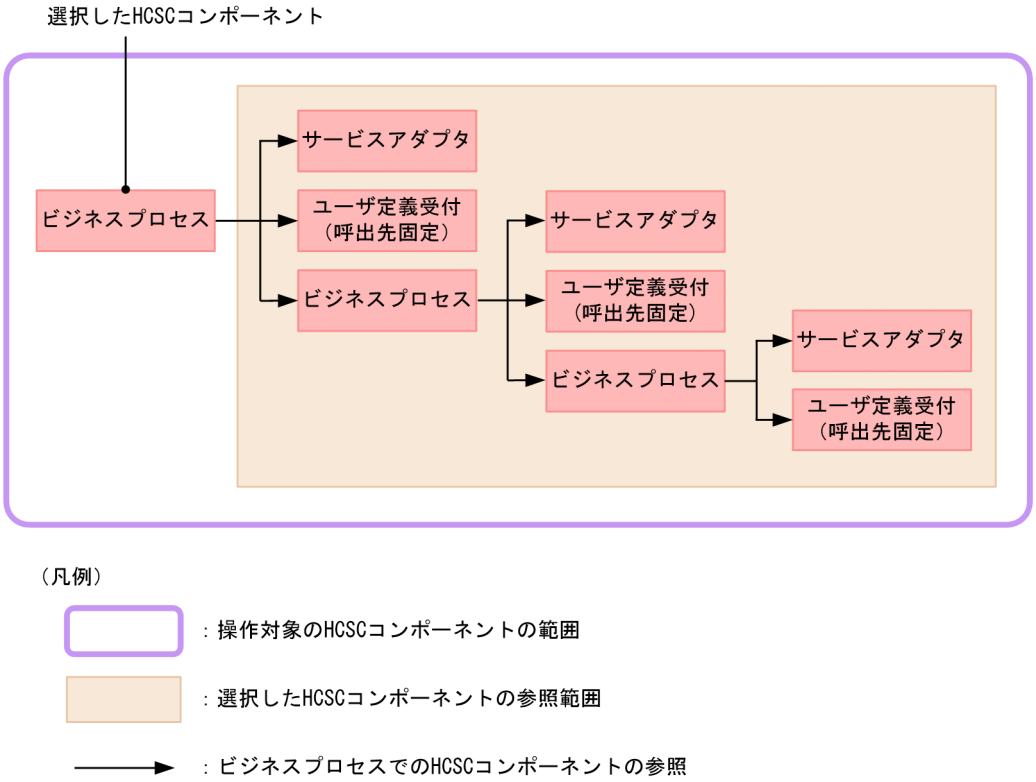
(2)　操作対象となる HCSC コンポーネント

「複数サービスをサーバに配備して開始」ウィザードのサービス選択ページで選択した HCSC コンポーネントが操作対象になります。なお，ここでいう操作対象とは，配備から開始までの一連の処理が実行される HCSC コンポーネントのことです。

選択した HCSC コンポーネントがビジネスプロセスの場合，そのビジネスプロセスに定義されている HCSC コンポーネント（ビジネスプロセス，サービスアダプタ，およびユーザ定義受付（呼出先固定））も操作対象に含まれます。

ビジネスプロセスを選択した場合に操作対象となる HCSC コンポーネントの範囲を次の図に示します。

図 8-5　操作対象となる HCSC コンポーネントの範囲（ビジネスプロセスを選択した場合）



なお，ここでの「HCSC コンポーネントの参照」とは，ユーザ定義受付（呼出先固定）の追加とサービス呼出アクティビティでの指定によって，ほかの HCSC コンポーネントを使用することです。

注意事項

ユーザ定義受付（呼出先固定）の場合は，開発環境でビジネスプロセスと同時に処理されるため，ユーザ定義受付（呼出先固定）だけを操作対象にすることはできません。ただし，ユーザ定義受付（呼出先選択）は個別の操作対象にできます。

(3) 再パッケージングと配備定義

操作対象の HCSC コンポーネントがすでにパッケージされている場合や，配備定義（公開）されている場合は，再度のパッケージングや配備定義をしません。

ただし，操作対象がビジネスプロセスの場合は，「複数サービスをサーバに配備して開始」ウィザードのオプションページで，再度パッケージングするように変更できます。パッケージングによって Java クラスの変更が反映されるため，必要に応じてオプションを変更してください。

8.5.2 HCSC コンポーネントを HCSC サーバに配備して開始する方法

HCSC コンポーネントを HCSC サーバに配備して開始する方法について説明します。

(1) 前提条件

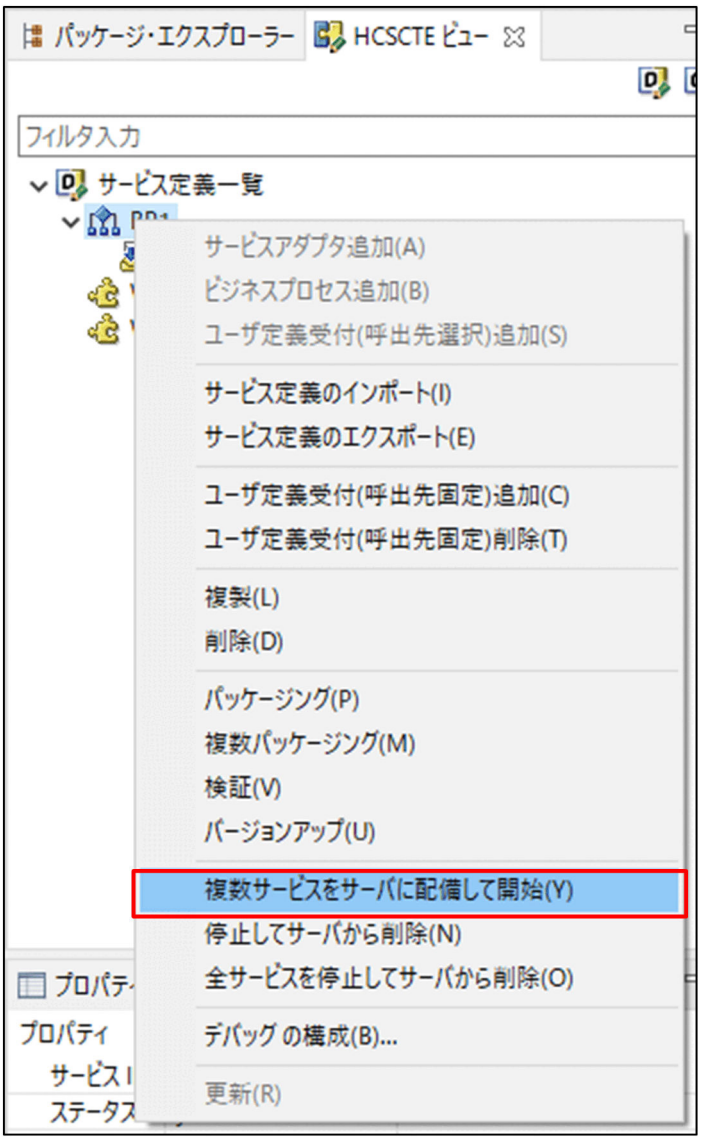
HCSC コンポーネントを HCSC サーバに配備して開始するためには，次に示す条件を満たしている必要があります。

- システム開発時，または単体テストから結合テスト時を対象に実施してください。
- 開発環境，運用環境，および実行環境は，同一のマシンに構築してください。複数マシンの構成にした場合，動作は保証できません。
- HCSC サーバはセットアップ済みの状態である必要があります。なお，セットアップは，単一の HCSC サーバ構成で実施してください。HCSC 簡易セットアップを利用する場合も，単一の HCSC サーバ構成でセットアップしてください。
- HCSC サーバおよびデータベースを起動しておいてください。

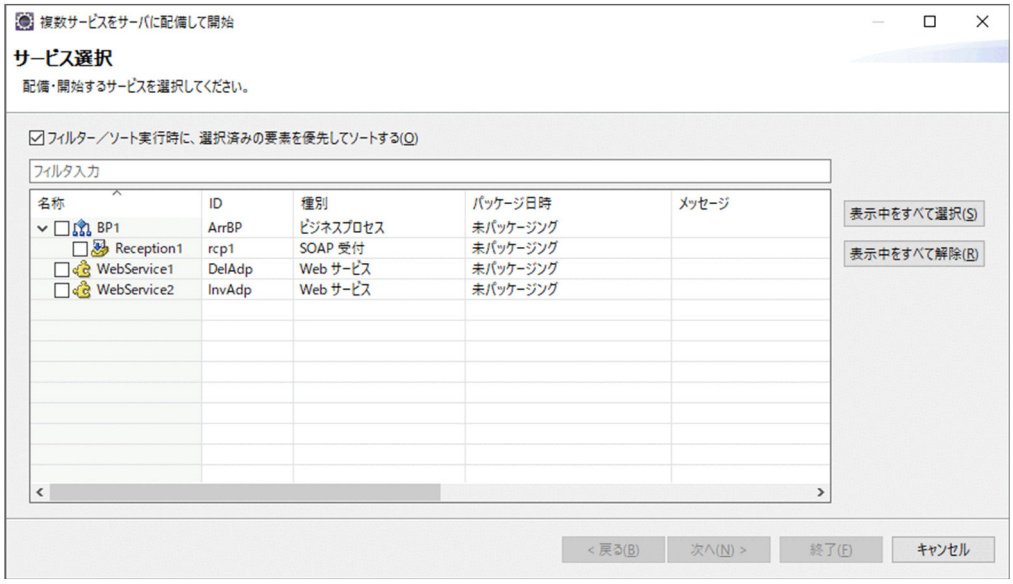
(2) 操作手順

操作手順を次に示します。

1. ツリービューのサービス定義一覧で，サービス定義一覧または HCSC コンポーネントを選択します。
2. 選択したサービス定義一覧または HCSC コンポーネントを右クリックして，［複数サービスをサーバに配備して開始］を選択します。



「複数サービスをサーバに配備して開始」ウィザードのサービス選択ページが表示されます。



3. サービス一覧の中から，配備・開始するサービスのチェックボックスにチェックします。

「フィルタ入力」にキーワードを入力すると，表示されるサービスをフィルタリングできます。

また，表示されたサービスのチェックボックスをすべてチェックするには［表示中をすべて選択］ボタンをクリックします。逆に，すべてのチェックを解除するには［表示中をすべて解除］ボタンをクリックします。

4. 次のどちらかの操作をします。

動作オプションを変更しないで配備・開始する場合

手順 6.に進んでください。動作オプションには，前回選択した値が適用されます。

動作オプションを変更したい場合

［次へ］ボタンをクリックします。

「複数サービスをサーバに配備して開始」ウィザードのオプションページが表示されます。次の手順に進んでください。

5. 必要に応じて動作オプションを変更します。



動作オプションの項目と，変更できる内容を次に示します。

- パッケージング済みビジネスプロセス

  パッケージング済みのビジネスプロセスを再パッケージングするかどうかを設定します。

  デフォルトでは，再パッケージングしない設定になっています。

- サービス呼出アクティビティに設定されているサービス

  サービス呼出アクティビティに設定されているサービスを自動的に配備・開始するかどうかを設定します。

  デフォルトでは，自動的に配備・開始しない設定になっています。

- エラー発生時の処理

  各サービス処理でエラーが発生したとき，スキップして次の処理を継続するかどうかを設定します。

  デフォルトでは，エラーが発生した要素をスキップして処理を継続する設定になっています。

6. [終了] ボタンをクリックします。

サービスが配備・開始されます。処理の経過はコンソールビューに表示されます。

続いて表示されるダイアログは，Eclipse 起動後の一括実行が初めてかどうかによって次のように異なります。

- Eclipse 起動後に初めて一括実行を実施する場合
  [アカウント認証] ダイアログが表示されます。
- Eclipse 起動後，2 回目以降に一括実行を実施する場合
  アカウント認証が省略されます。手順 8.に進んでください。

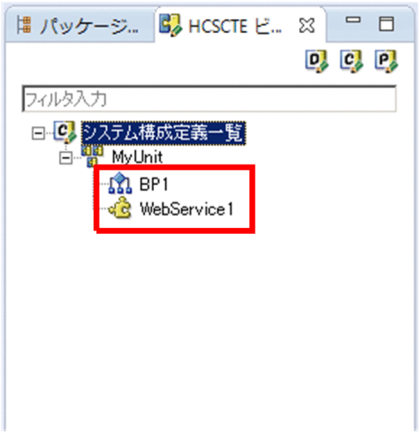
7. [アカウント認証] ダイアログにユーザ ID とパスワードを入力して，[OK] ボタンをクリックします。

8. 処理完了を伝えるダイアログが表示されたら，[OK] ボタンをクリックします。

処理が正常終了した場合，必要に応じて一括実行の処理結果を確認してください。なお，一括実行の処理結果は，「複数サービスをサーバに配備して開始」ウィザードのオプションページの設定によって異なります。ここでは，オプションページのオプションをすべてデフォルトに設定している場合を例に，確認事項を説明します。

配備定義されているかを確認する

ツリービューのシステム構成定義一覧に，選択した HCSC コンポーネントが追加されていることを確認します。

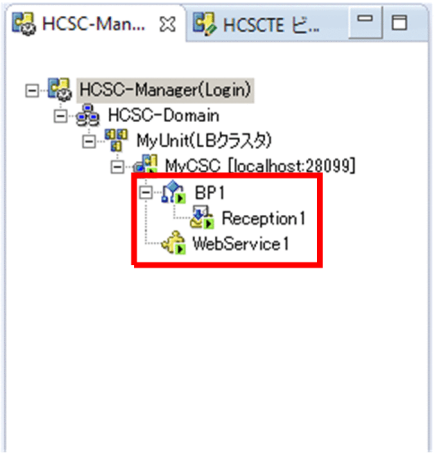


上の例では，選択したビジネスプロセス（BP1），および選択したビジネスプロセスに定義されているサービスアダプタ（WebService1）が追加されていることを確認できます。

HCSC サーバに配備して開始されているかを確認する

HCSC-Manager にログインして，HCSC-Manager ビューを選択します。HCSC-Manager のログイン方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「5.1 HCSC サーバのオペレーションをするための準備」を参照してください。

選択した HCSC コンポーネントが，HCSC サーバに配備して開始されていることを確認します。



上の例では，選択したビジネスプロセス（BP1），および選択したビジネスプロセスに定義されているサービスアダプタ（WebService1）ならびにユーザ定義受付（Reception1）が，配備して開始されていることを確認できます。

(3) 注意事項

HCSC コンポーネントを HCSC サーバに配備して開始する場合の注意事項を次に示します。

- 選択した HCSC コンポーネントを HCSC サーバに配備して開始する場合，次のときにエラーになることがあります。

  ・開発環境と運用環境でリポジトリの構成形態（データベースと Reliable Messaging の組み合わせ）が不一致である。

  ・運用環境のリポジトリで配備済みのサービスが，開発環境で更新または削除されている。

  ・開発環境と運用環境のリポジトリ内で SOAP モードが不一致である。

  エラーになる場合は，定義したすべての HCSC コンポーネントを HCSC サーバに配備して開始する方法を実施してください。

- オプションページで，［エラー発生時の処理］で［処理を打ち切る］を選択している場合，処理の途中でエラーが発生すると，処理を中断して一括実行を終了します。なお，エラー発生前に一括実行が終了している処理は，実行前の状態に戻りません。

  エラーが発生した場合，エラーの発生原因を取り除いて，再度実行してください。

- HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行に失敗すると，サービスが起動できない状態となり，HCSC コンポーネントを停止するまで再度実行できなくなることがあります。

  処理の失敗後，HCSC サーバに配備して開始を再度実行しても失敗する場合は，HCSC コンポーネントを停止し，アンデプロイしてください。停止・アンデプロイの手順については，「8.6 HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行」を参照してください。

  なお，参照先の方法で停止・アンデプロイに失敗する場合は，コマンドを次の順に実行してください。

  1. csccompostop -all コマンド

     csccompostop コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompostop（HCSC コンポーネントの停止）」を参照してください。

2. csccompoundeploy -all コマンド

csccompoundeploy コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompoundeploy（配備済みの HCSC コンポーネントの削除)」を参照してください。

8.6 HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行

HCSC コンポーネントを停止して HCSC サーバから削除する一連の処理を開発環境で一括実行できます。開発環境または運用環境で一つ一つ実施していた作業を開発環境で一括実行するため，ユーザの操作負担を軽減できます。なお，一括実行は，システム開発時，または単体テストから結合テスト時を対象としています。

HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行には，次の 2 つがあります。

- 選択した HCSC コンポーネントを停止して HCSC サーバから削除する。開発環境では，HCSC コンポーネントの配備定義を削除する。
- HCSC サーバに配備したすべての HCSC コンポーネントを停止して，HCSC サーバから削除する。開発環境では，すべての HCSC コンポーネントの配備定義を削除する。

ここでは，停止から削除までの処理の流れと操作方法について説明します。

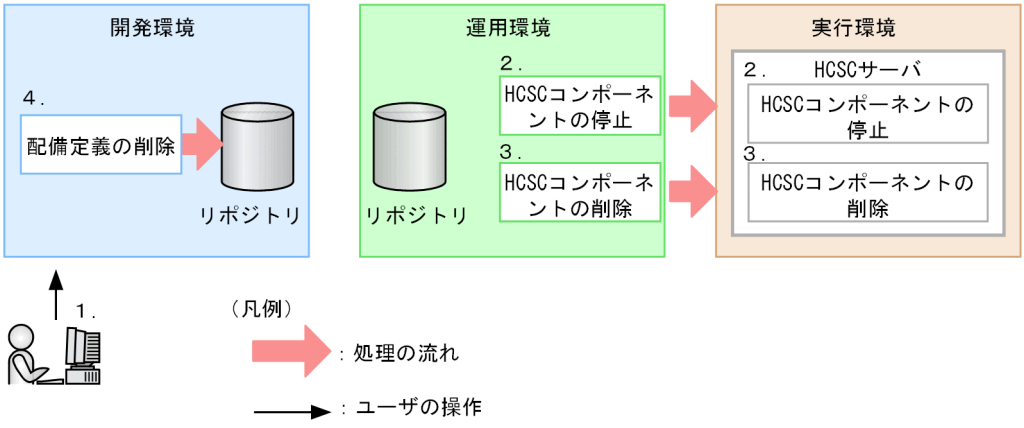
8.6.1 停止から削除までの処理の流れ

HCSC コンポーネントを停止して HCSC サーバから削除する一連の処理の流れ，および操作対象となる HCSC コンポーネントの範囲を次に説明します。

(1) 処理の流れ

HCSC コンポーネントを停止して HCSC サーバから削除する一連の処理の流れを次の図に示します。なお，図の 2.～4.は，ユーザが開発環境で操作すると，自動的に実行されます。

図 8-6 停止から削除までの処理の流れ



1. ユーザは開発環境で，一括実行を実施します。
2. 実行環境の HCSC サーバで稼働している HCSC コンポーネントが停止されます。

3. 実行環境の HCSC サーバから HCSC コンポーネントが削除されます。
4. 開発環境で HCSC コンポーネントの配備定義が削除され，リポジトリ内のシステム構成定義が更新されます。

(2) 操作対象となる HCSC コンポーネント

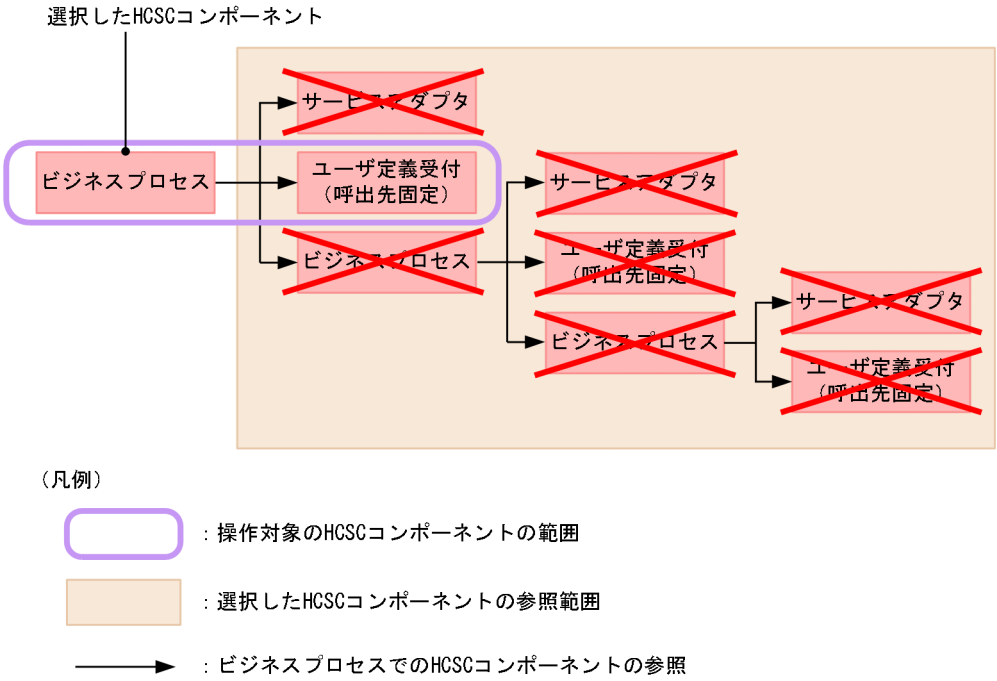
選択した HCSC コンポーネントを停止して HCSC サーバから削除する場合と，すべての HCSC コンポーネントを停止して HCSC サーバから削除する場合とで，操作対象となる HCSC コンポーネントが異なります。なお，ここでいう操作対象とは，停止から削除までの一連の処理が実行される HCSC コンポーネントのことです。

選択した HCSC コンポーネントを停止して HCSC サーバから削除する場合

開発環境で選択した HCSC コンポーネントが操作対象となります。選択した HCSC コンポーネントがビジネスプロセスの場合，そのビジネスプロセスに定義されているユーザ定義受付（呼出先固定）も操作対象に含まれます。選択した HCSC コンポーネントに定義されているビジネスプロセスおよびサービスアダプタは，操作対象に含まれません。

ビジネスプロセスを選択した場合に操作対象となる HCSC コンポーネントの範囲を次の図に示します。

図 8-7 操作対象となる HCSC コンポーネントの範囲（ビジネスプロセスを選択した場合）



なお，ここでの「HCSC コンポーネントの参照」とは，ユーザ定義受付（呼出先固定）の追加とサービス呼出アクティビティでの指定によって，ほかの HCSC コンポーネントを使用することです。

注意事項

ユーザ定義受付（呼出先固定）の場合は，開発環境でビジネスプロセスと同時に処理されるため，ユーザ定義受付（呼出先固定）だけを操作対象にすることはできません。ただし，ユーザ定義受付（呼出先選択）は個別の操作対象にできます。

すべての HCSC コンポーネントを停止して HCSC サーバから削除する場合

HCSC サーバに配備したすべての HCSC コンポーネント，および開発環境で配備定義したすべての HCSC コンポーネントが操作対象になります。

8.6.2 HCSC コンポーネントを停止して HCSC サーバから削除する方法

HCSC コンポーネントを停止して HCSC サーバから削除する方法について説明します。

(1) 前提条件

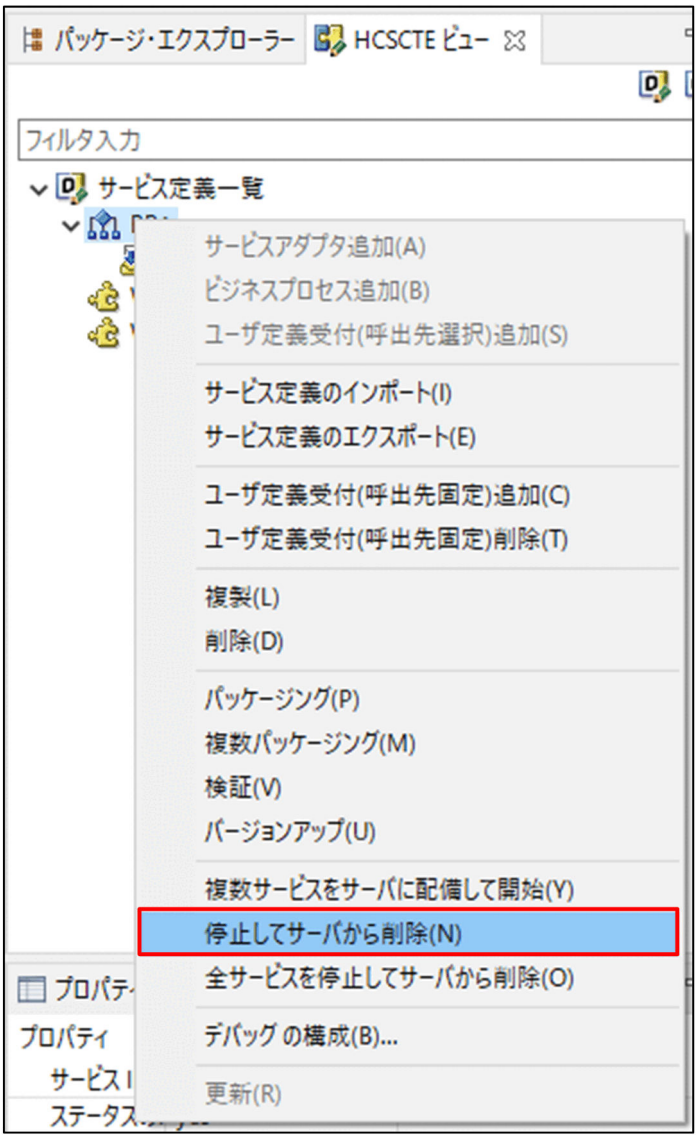
HCSC コンポーネントを停止して HCSC サーバから削除するためには，次に示す条件を満たしている必要があります。

- システム開発時，または単体テストから結合テスト時を対象に実施してください。
- 開発環境，運用環境，および実行環境は，同一のマシンに構築してください。複数マシンの構成にした場合，動作は保証できません。
- HCSC サーバはセットアップ済みの状態である必要があります。なお，セットアップは，単一の HCSC サーバ構成で実施してください。HCSC 簡易セットアップを利用する場合も，単一の HCSC サーバ構成でセットアップしてください。
- HCSC サーバおよびデータベースを起動しておいてください。

(2) 操作手順

(a) 選択した HCSC コンポーネントを停止して HCSC サーバから削除する場合

1. ツリービューのサービス定義一覧で，HCSC コンポーネントを停止して HCSC サーバから削除する HCSC コンポーネントを選択します。
2. 選択した HCSC コンポーネントを右クリックして，［停止してサーバから削除］を選択します。



Eclipse 起動後，最初に一括実行を実施するときは，［アカウント認証］ダイアログが表示されます。ユーザ ID とパスワードを入力して［OK］ボタンをクリックすると処理が開始されます。Eclipse 起動後，2 回目以降に一括実行を実施するときは，すぐに処理が開始されます。

3. 処理完了を伝えるダイアログが表示されたら，［OK］ボタンをクリックします。

4. 必要に応じて，次の内容を確認します。

配備定義が削除されているかを確認する

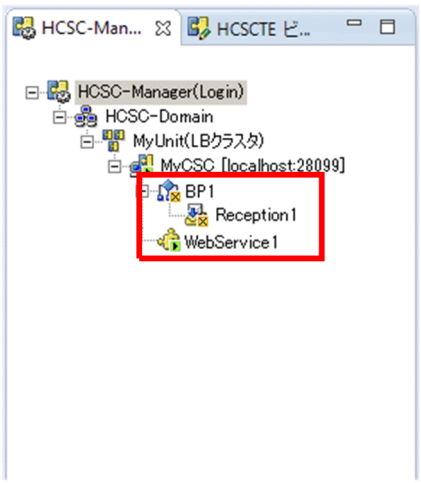
ツリービューのシステム構成定義一覧から，選択した HCSC コンポーネントが削除されていることを確認します。



上の例では，選択したビジネスプロセス（BP1）が削除されていることを確認できます。

HCSC コンポーネントが停止して HCSC サーバから削除されているかを確認する

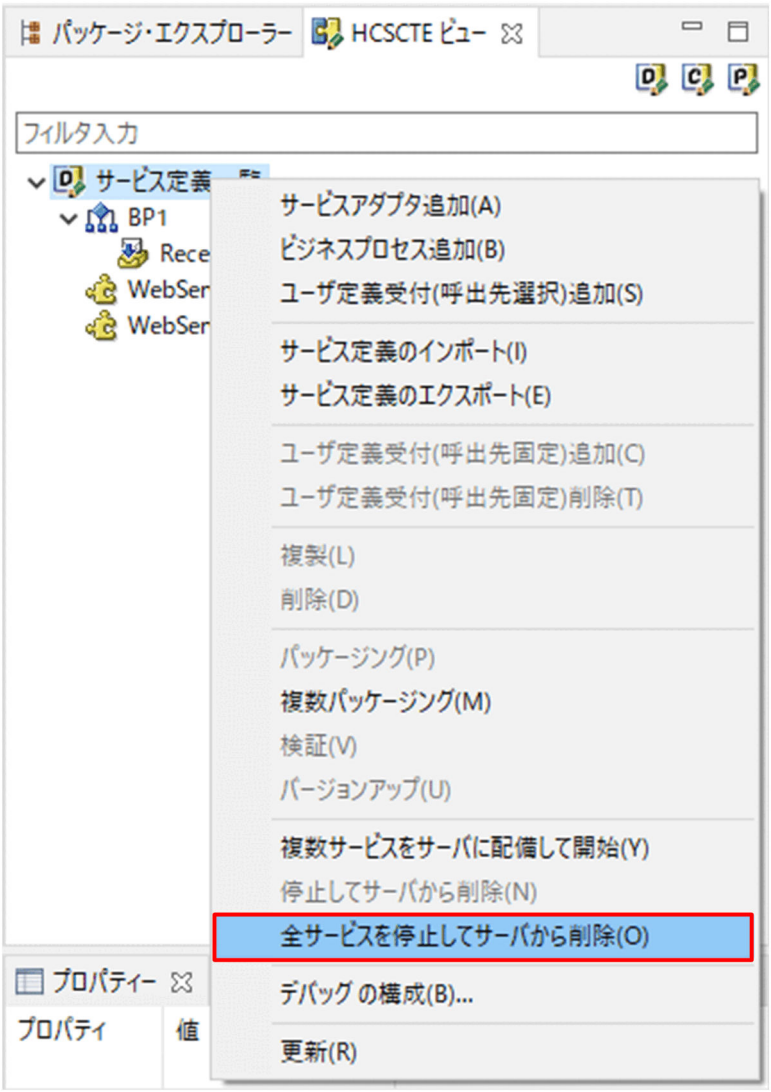
HCSC-Manager にログインして，HCSC-Manager ビューを選択します。選択した HCSC コンポーネントが停止して，HCSC サーバから削除されていることを確認します。HCSC-Manager のログイン方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「5.1　HCSC サーバのオペレーションをするための準備」を参照してください。



上の例では，選択したビジネスプロセス（BP1），および選択したビジネスプロセスに定義されているユーザ定義受付（Reception1）が，停止して削除されていることを確認できます。

(b)　すべての HCSC コンポーネントを停止して HCSC サーバから削除する場合

1. ツリービューのサービス定義一覧で右クリックして，［全サービスを停止してサーバから削除］を選択します。



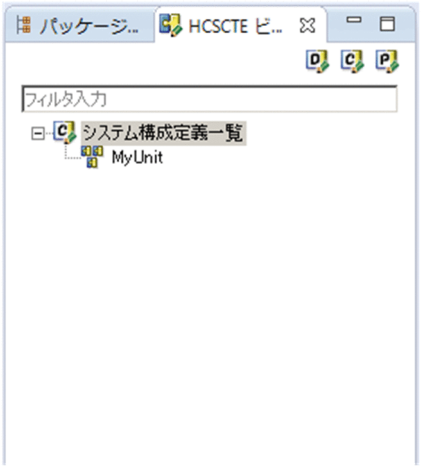
Eclipse 起動後，最初に一括実行を実施するときは，［アカウント認証］ダイアログが表示されます。ユーザ ID とパスワードを入力して［OK］ボタンをクリックすると処理が開始されます。Eclipse 起動後，2 回目以降に一括実行を実施するときは，すぐに処理が開始されます。

2. 処理完了を伝えるダイアログが表示されたら，［OK］ボタンをクリックします。

3. 必要に応じて，次の内容を確認します。

配備定義が削除されているかを確認する

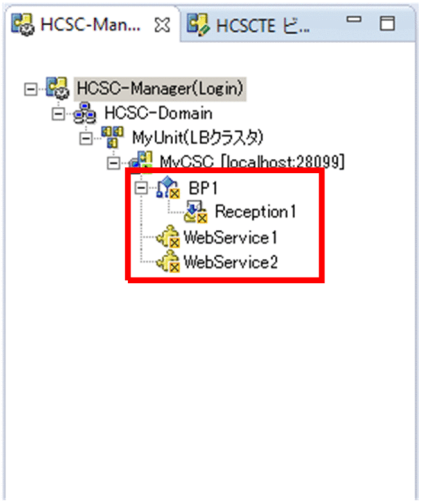
ツリービューのシステム構成定義一覧から，すべての HCSC コンポーネントが削除されていることを確認します。



上の例では，すべての HCSC コンポーネント（BP1，WebService1，WebService2）が削除されていることを確認できます。

HCSC コンポーネントが停止して HCSC サーバから削除されているかを確認する

HCSC-Manager にログインして，HCSC-Manager ビューを選択します。すべての HCSC コンポーネントが停止して，HCSC サーバから削除されていることを確認します。HCSC-Manager のログイン方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「5.1　HCSC サーバのオペレーションをするための準備」を参照してください。



上の例では，すべての HCSC コンポーネント（BP1，WebService1，WebService2，Reception1）が，停止して削除されていることを確認できます。

(3)　注意事項

HCSC コンポーネントを停止して HCSC サーバから削除する場合の注意事項を次に示します。

- 選択した HCSC コンポーネントを停止して HCSC サーバから削除する場合，次のときにエラーになることがあります。

  ・開発環境と運用環境でリポジトリの構成形態（データベースと Reliable Messaging の組み合わせ）が不一致である。

・運用環境のリポジトリで配備済みのサービスが，開発環境で更新または削除されている。

・開発環境と運用環境のリポジトリ内で SOAP モードが不一致である。

エラーになる場合は，すべての HCSC コンポーネントを停止して HCSC サーバから削除する方法を実施してください。

- 処理の途中でエラーが発生した場合，処理を中断して一括実行を終了します。ただし，エラー発生前に一括実行が終了している処理は，実行前の状態に戻りません。その場合，エラーの発生原因を取り除いて，再度実行してください。
- HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行に失敗すると，サービスが起動できない状態となり，HCSC コンポーネントを停止するまで再度実行できなくなることがあります。処理に失敗した場合は，csccompostop -all コマンド，csccompoundeploy -all コマンドの順に実行して HCSC コンポーネントを停止し，アンデプロイしてください。そのあと，HCSC コンポーネントを HCSC サーバから削除してください。

  csccompostop コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompostop（HCSC コンポーネントの停止)」を参照してください。csccompoundeploy コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompoundeploy（配備済みの HCSC コンポーネントの削除)」を参照してください。

9 サービスリクエスタの作成

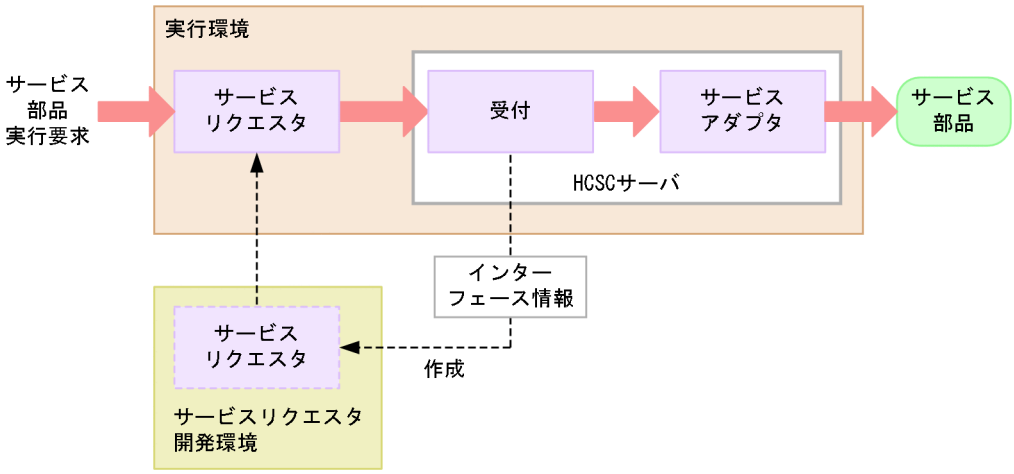
この章では，HCSC サーバの標準受付およびユーザ定義受付にサービス部品を要求するサービスリクエスタの作成について説明します。

9.1　サービスリクエスタ作成の概要

サービスリクエスタとは，業務担当者からの要求を受け付けて，サービスアダプタやビジネスプロセスなどの各 HCSC コンポーネントにサービス部品の実行要求を送信するアプリケーションです。サービスリクエスタは，実行環境が提供するインターフェース情報を基に，開発環境で作成し，実行環境に配備して利用します。サービスリクエスタの開発は，Java のプログラムを開発できる環境で行ってください。

サービスリクエスタの概要を次の図に示します。

図 9-1　サービスリクエスタの概要



サービスリクエスタは，HCSC サーバの標準受付およびユーザ定義受付にサービス部品の実行を要求します。なお，サービスリクエスタが使用するプロトコルによって，要求先は異なります。サービスリクエスタが使用するプロトコルと標準受付およびユーザ定義受付の種類を次の表に示します。

表 9-1　サービスリクエスタが使用するプロトコルと標準受付およびユーザ定義受付の種類

受付		プロトコル
標準受付	同期受付（Web サービス）	SOAP（HTTP）
	同期受付（SessionBean）	RMI-IIOP
	非同期受付（MDB（WS-R））	WS-R
	非同期受付（MDB（DB キュー））	• Reliable Messaging 独自のプロトコル • JMS
ユーザ定義受付	同期受付（Web サービス）	SOAP（HTTP）

注意事項

運用時の処理の流れを考慮して，サービスリクエスタを開発する必要があります。また，障害発生時にすぐに対策できるように，よく理解した上でエラー処理を作成する必要があります。

詳細については，次の個所を参照してください。

- マニュアル「サービスプラットフォーム 解説」の「2.　さまざまなシステムと接続するための機能」
- マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.　障害対策」

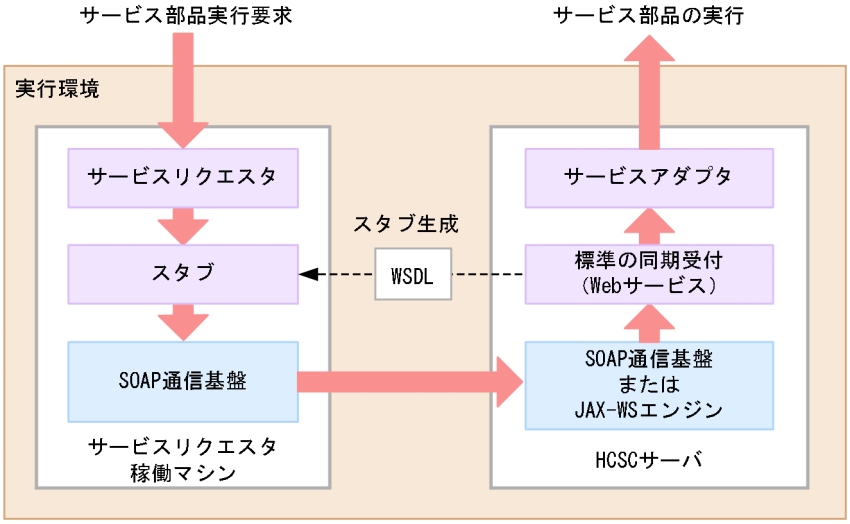
9.2　標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（SOAP 通信基盤）

標準の同期受付（Web サービス）へ要求電文を送信するサービスリクエスタは，SOAP を利用して通信します。サービスリクエスタから標準受付（Web サービス）へサービス部品実行の要求電文が送信され，HCSC サーバがサービス部品を実行します。

SOAP を利用する場合，同期受付（Web サービス）のインターフェース情報を WSDL から取得します。取得した WSDL からスタブを生成し，そのスタブを利用して，同期受付（Web サービス）に要求を出します。したがって，サービスリクエスタは，生成したスタブを利用できるように実装する必要があります。

標準の同期受付（Web サービス）に要求を出すサービスリクエスタと HCSC サーバの関係を次の図に示します。

図 9-2　標準の同期受付（Web サービス）に要求を出すサービスリクエスタと HCSC サーバの関係（SOAP 通信基盤）

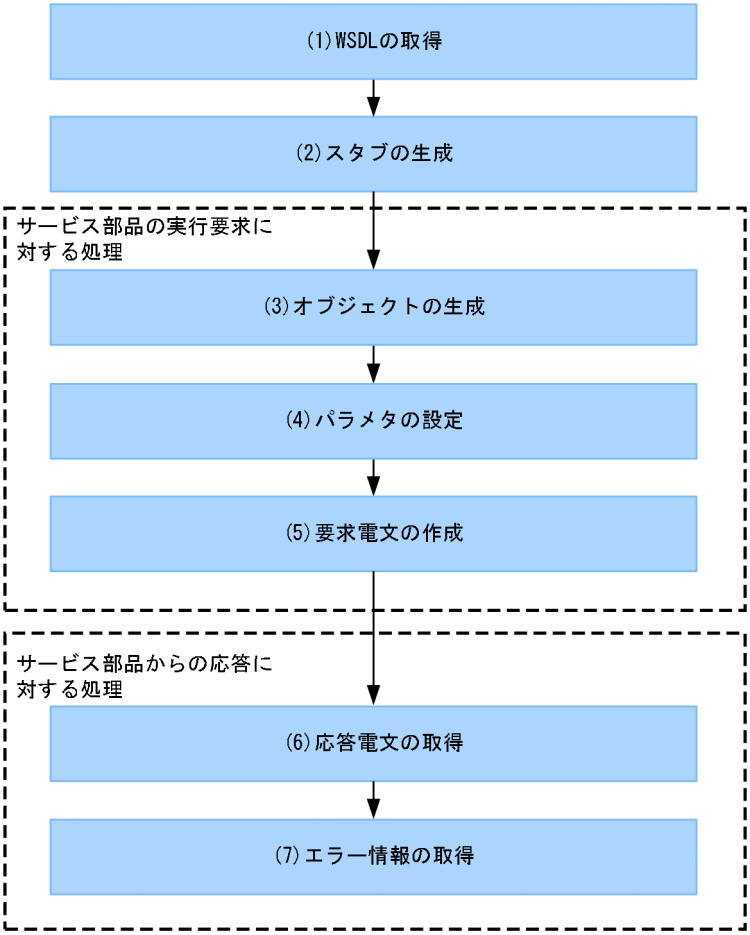


なお，JAX-WS エンジンを使用する場合，スタブではなくサービスクラスを生成します。詳細については，「9.3　標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（JAX-WS エンジン）」を参照してください。

9.2.1　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（SOAP 通信基盤）

標準の同期受付（Web サービス）にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-3　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（SOAP 通信基盤）



各工程の作業を次に示します。

(1)　WSDL の取得

サービス部品を呼び出す HCSC サーバの同期受付（Web サービス）のインターフェース情報を WSDL から取得します。WSDL の取得については，「9.2.2　WSDL を取得する」を参照してください。

(2)　スタブの生成

「(1)　WSDL の取得」で取得した WSDL からスタブを生成します。スタブの生成については，「9.2.3 スタブを生成する」を参照してください。

(3)　オブジェクトの生成

同期受付（Web サービス）のメソッドを呼び出すために，「(2)　スタブの生成」で生成したスタブからオブジェクトを生成します。オブジェクトの生成については，「9.2.4　オブジェクトを生成する」を参照してください。

(4)　パラメタの設定

同期受付（Web サービス）のメソッドの引数となるパラメタを設定します。パラメタの設定については，「9.2.5　パラメタを設定する」を参照してください。

(5)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.2.6　要求電文を作成する」を参照してください。

(6)　応答電文の取得

同期受付（Web サービス）から，サービス部品の実行要求に対する応答電文を取得します。応答電文の取得については，「9.2.7　応答電文を取得する」を参照してください。

(7)　エラー情報の取得

要求先のサービス部品，HCSC サーバ，および SOAP エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。エラー情報の取得については，「9.2.8　エラー情報を取得する」を参照してください。

9.2.2　WSDL を取得する

サービス部品を呼び出す HCSC サーバの同期受付（Web サービス）のインターフェース情報を WSDL から取得します。WSDL を取得するには，次の 2 つの方法があります。

方法 1

1. Eclipse のメニューから，［HCSC-Definer］－［公開サービス一覧］を選択します。
   ツリービューの公開サービス一覧に，HCSC コンポーネントの一覧が表示されます。

2. HCSC コンポーネントの一覧から，呼び出すサービス部品（HCSC コンポーネント）を選択し，ダブルクリックします。
   HCSC コンポーネント情報表示画面に，HCSC コンポーネントの情報が表示されます。HCSC コンポーネント情報表示画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.6　HCSC コンポーネント情報表示画面」を参照してください。
   ［サーバ情報］には，サービス部品の要求先となる HCSC サーバの情報が表示されます。

3. 必要に応じて，［バインディングスタイル］ラジオボタンおよび［SOAP バージョン］ラジオボタンで，取得する WSDL の種類を選択します。
   ［バインディングスタイル］ラジオボタンでは，取得する WSDL 定義のスタイルを選択します。［SOAP バージョン］ラジオボタンでは，対応する SOAP 仕様のバージョンを選択します。

4. [WSDL 取得] ボタンをクリックします。

［WSDL ファイルの保存］ダイアログが表示されます。

保存先を指定して，WSDL を取得します。

方法 2

1. HCSC-Messaging をインストールしたディレクトリから，サンプルとして提供されている WSDL を取得します。

用途に合わせて次に示すどちらかの WSDL を取得してください。

- rpc-literal 型

  cscmsg_ws.wsdl ファイル

- document-literal 型

  cscmsg_ws_doc.wsdl ファイル

HCSC-Messaging をインストールしたディレクトリについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.1.2　Service Platform のインストール」を参照してください。

2. WSDL の<soap:address>の location の内容を次のように変更します。

```
http://<ホスト名>:<ポート番号>※1/<コンテキストルート>※2
          /services/CSCMsgSyncServiceDeliveryWSImpl
```

注※1

HCSC サーバの URL です。

注※2

開発環境および運用環境で HCSC サーバ構成配備時に決定される HCSC サーバのクラスタ名です。

9.2.3　スタブを生成する

取得した WSDL からスタブを生成します。スタブは，サービスプラットフォームが開発支援コマンドとして提供する WSDL2Java コマンドで生成します。

コマンドの入力例（WSDL のスタイルが rpc-literal の場合）を次に示します。

```
WSDL2Java cscmsg_ws.wsdl
```

WSDL のスタイルが document-literal の場合，上記の例の「cscmsg_ws.wsdl」の部分には「cscmsg_ws_doc.wsdl」と設定します。

WSDL2Java コマンドのオプションなどの詳細については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」の「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

このコマンドを実行すると，次のディレクトリおよびファイルが作成されます。

```
/カレントディレクトリ
  └jp
    └co
      └Hitachi
        └soft
          └csc
            └msg
              └message
                └reception
                  └ejb
                    ├CSCMsgSyncServiceDeliveryWSImpl.java※1
                    ├CSCMsgSyncServiceDeliveryWSImplService.java※2
                    ├CSCMsgSyncServiceDeliveryWSImplServiceLocator.java※3
                    ├CSCMsgSyncServiceDeliveryWSImplSoapBindingStub.java※4
                    └CSCMsgServerFaultException.java※4
```

注※1

ユーザ定義のデータクラス（サービスリクエスタのインターフェースクラス）です。

注※2

サービス部品のインターフェースクラスです。

注※3

サービスクラス（サービス部品のインターフェースクラス）です。サービス部品への接続情報を保持します。

注※4

スタブクラスです。

9.2.4　オブジェクトを生成する

同期受付（Web サービス）のメソッドを呼び出すために，生成したスタブを使用して，オブジェクトを生成します。

(1)　使用するスタブ

次の 2 つのスタブを使用して，オブジェクトを生成します。

CSCMsgSyncServiceDeliveryWSImplServiceLocator.java クラス

このクラスは，サービス部品への接続先（endpoint）情報の参照および設定をします。このクラスでは次のメソッドを提供します。

表 9-2　CSCMsgSyncServiceDeliveryWSImplServiceLocator.java クラスのメソッド

メソッド名	機能説明
getCSCMsgSyncServiceDeliveryWSImplAddress()	サービス部品への接続先情報を返します。 戻り値： java.lang.String
getCSCMsgSyncServiceDeliveryWSImpl()	サービスクラスへのインターフェースクラスのオブジェクトポインタを返します。 戻り値： インターフェースクラスオブジェクト（CSCMsgSyncServiceDeliveryWSImpl オブジェクト）
getCSCMsgSyncServiceDeliveryWSImpl(java.net.URL portAddress)	設定されたサービス部品への接続先情報を使用して，サービスクラスへのオブジェクトポインタを返します。 戻り値： インターフェースクラスオブジェクト（CSCMsgSyncServiceDeliveryWSImpl オブジェクト）

CSCMsgSyncServiceDeliveryWSImpl.java クラス

サービスクラスとして利用できるメソッドの一覧が記述されています。このクラスを使用して SOAP サービスを利用します。

(2)　オブジェクトの生成手順

同期受付（Web サービス）のメソッドを呼び出すためのオブジェクトを生成する手順を次に示します。

1. サービス部品のインターフェースクラスである CSCMsgSyncServiceDeliveryWSImplServiceLocator クラスのオブジェクトを生成します。

(例)

```
CSCMsgSyncServiceDeliveryWSImplServiceLocator locator
            = new CSCMsgSyncServiceDeliveryWSImplServiceLocator();
```

2. サービス部品のインターフェースクラスのオブジェクトを使用して，サービスリクエスタのインターフェースクラスである CSCMsgSyncServiceDeliveryWSImpl.java クラスのオブジェクトを生成します。

なお，生成・取得したサービスリクエスタのインターフェースクラスのインスタンスは，複数のスレッドで共有できません。

(例)

```
CSCMsgSyncServiceDeliveryWSImpl ws = null;
try {
      ws = locator.getCSCMsgSyncServiceDeliveryWSImpl();
}catch (ServiceException e) {
             e.printStackTrace();
```

```
            return;
}
```

サービス部品への接続先は，WSDL 定義内の Service 要素の address 子要素である location 属性となります。サービスリクエスタのプログラム内で，サービス部品への接続先情報を取得する場合は，次のように記述します。

(例)

```
String url = locator.getCSCMsgSyncServiceDeliveryWSImplAddress();
```

また，サービスリクエスタのプログラム内で，サービス部品への接続先を変更する場合は，次のように記述します。

(例)

```
java.net.URL endpoint
 = new java.net.URL("http://hostname:80/<コンテキストルート>
                  /services/CSCMsgSyncServiceDeliveryWSImpl");
CSCMsgSyncServiceDeliveryWSImpl locator
 = locator.getCSCMsgSyncServiceDeliveryWSImpl(endpoint);
```

3. 生成した CSCMsgSyncServiceDeliveryWSImpl.java クラスのオブジェクトを使用して，同期受付（Web サービス）のメソッドを呼び出します。

(例) 要求電文が XML の場合

```
String result = ws.invokeXML(               // メソッドの呼び出し
                        serviceName,        // サービス名
                        clientID,           // クライアント相関ID
                        requestFormatID,    // リクエストフォーマットID
                        responseFormatID,   // レスポンスフォーマットID
                        operationName,      // オペレーション名
                        userData);          // ユーザ電文
```

(例) 要求電文がバイナリの場合

```
byte[] resultBinary = ws.invokeBinary(      // メソッドの呼び出し
                          serviceName,      // サービス名
                          clientID,         // クライアント相関ID
                          requestFormatID,  // リクエストフォーマットID
                          responseFormatID, // レスポンスフォーマットID
                          operationName,    // オペレーション名
                          userDataBinary.length,  // ユーザ電文長
                          userDataBinary);  // ユーザ電文
```

注意事項

バイナリ電文の要求電文を送信できるのは，サービス部品側で使用している電文形式がバイナリ形式のときだけです。

9.2.5　パラメタを設定する

同期受付（Web サービス）のメソッドを呼び出すため，メソッドの引数となるパラメタを設定します。パラメタの詳細を次の表に示します。

表 9-3　パラメタの詳細（標準の同期受付（Web サービス））

パラメタ名	データ型	パラメタ		説明
		invokeXML	invokeBinary	
サービス名（serviceName）	java.lang.String	in0（type="xsd:string"）		要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，開発環境で定義したサービスアダプタまたはビジネスプロセスを指定してください。
クライアント相関 ID（clientID）	java.lang.String	in1（type="xsd:string"）		サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 サービスリクエスタからの要求電文と，HCSC サーバで管理している実行履歴やログおよびトレースを対応づけるために使用します。したがって，HCSC サーバに送信した要求電文ごとに異なる ID を指定してください。 クライアント相関 ID を省略したい場合，NULL を指定してください。
リクエストフォーマット ID（requestFormatID）	java.lang.String	in2（type="xsd:string"）		サービスリクエスタからの要求電文フォーマットを一意に識別するための ID です。 このパラメタには NULL を指定してください。
レスポンスフォーマット ID（responseFormatID）	java.lang.String	in3（type="xsd:string"）		HCSC サーバからの応答電文フォーマットを一意に識別するための ID です。 このパラメタには NULL を指定してください。
オペレーション名（operationName）	java.lang.String	in4（type="xsd:string"）		要求先のサービス名に対するオペレーション名です。※ オペレーション名は，開発環境で定義したサービス部品のオペレーション名を指定します。255 バイト以内の XMLSchema の NCName 定義文字で指定してください。 要求先のサービス部品が同期のサービス部品（Web サービスまたは SessionBean）またはビジネスプロセスの場合，必ず指定してください。 要求先のサービス部品が非同期のサービス部品の場合，オペレーション名を省略できます。省略する場合，NULL を指定してください。

パラメタ名	データ型	パラメタ		説明
		invokeXML	invokeBinary	
ユーザ電文（userData）	java.lang.String	in5（type="xsd:string"）	×	サービスリクエスタからの要求電文です。※ 要求電文が XML の場合に指定します。要求電文がない場合，NULL または空文字（""）を指定してください。要求電文の詳細については，「9.2.6　要求電文を作成する」を参照してください。
ユーザ電文長（userDataBinary.length）	int	×	in5（type="xsd:int"）	要求電文の長さです。 要求電文がバイナリの場合に指定します。要求電文がバイナリの場合は，必ず指定してください。 要求電文がない場合，0 を指定してください。
ユーザ電文（userDataBinary）	byte[]	×	in6（type="base64Binary"）	サービスリクエスタからの要求電文です。※ 要求電文がバイナリの場合に指定します。要求電文の詳細については，「9.2.6　要求電文を作成する」を参照してください。 要求電文がない場合，NULL または 0 バイトのバイト配列を指定してください。

（凡例）

inX（X=1～6）：スタブで示しているメソッドごとのパラメタです。

×：指定できません。

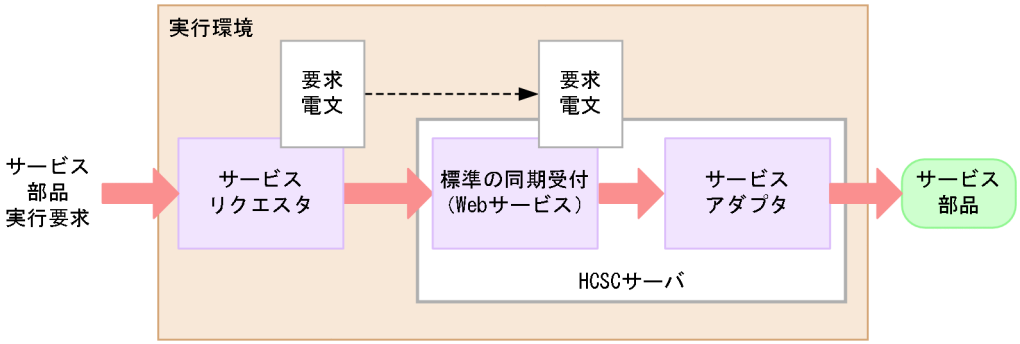
注※

要求先のサービス部品のプロトコルが SOAP の場合，ユーザ電文のルート要素の名称（データ変換を行う場合，データ変換後の電文のルート要素の名称）から呼び出すオペレーションを決定します。そのため，ユーザ電文のルート要素に誤った名称を指定した場合，意図しないオペレーションを呼び出してしまうので注意してください。

9.2.6　要求電文を作成する

サービスリクエスタから HCSC サーバの同期受付（Web サービス）に，サービス部品を要求する要求電文を作成します。サービスリクエスタから送信する要求電文の内容は，サービス部品側と同じ電文フォーマットで作成してください。要求電文の送信を次の図に示します。

図 9-4　要求電文の送信（標準の同期受付（Web サービス））



正常な要求電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.2　Web サービス（SOAP 通信）を使用したサービスとの接続」を参照してください。

また，サービスリクエスタから同期受付（Web サービス）に送信する要求電文は，電文種別に XML 電文とバイナリ電文の 2 種類があります。どちらの電文種別を使用するかは，サービス部品側で使用しているプロトコルによって異なります。電文種別が XML 電文の要求電文とバイナリ電文の要求電文を次に説明します。

(1)　XML 電文の要求電文

サービス部品側のプロトコルが次に示すものの場合，およびビジネスプロセスに対して要求を行う場合，XML 電文の要求電文を送信できます。

- SOAP（SOAP アダプタの場合）※1※2
- RMI-IIOP（SessionBean アダプタの場合）※1※2
- WS-R（MDB（WS-R）アダプタの場合）※2
- DB キュー（MDB（DB キュー）アダプタの場合）※2

注※1

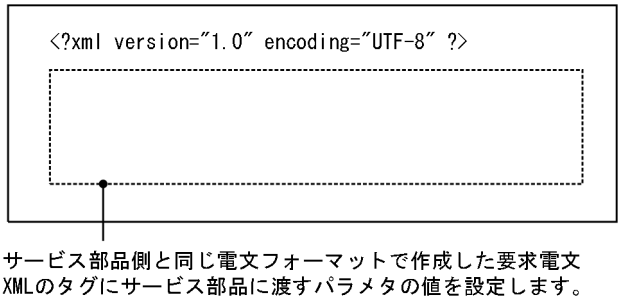
必ず要求電文を指定してください。

注※2

要求電文を指定しなかった場合，サービスアダプタでデータ変換処理が行われません。

XML 電文の要求電文を次の図に示します。

図 9-5　XML 電文の要求電文



注意事項

指定する XML 文書（要求電文）の先頭には，必ず XML 宣言を記述してください。XML 宣言がない場合，または XML 文書中のユーザ電文に XML 宣言がある場合，動作の保証はできません。

また，XML 宣言の文字コードの指定（encoding）には，必ず「UTF-8」を指定してください。

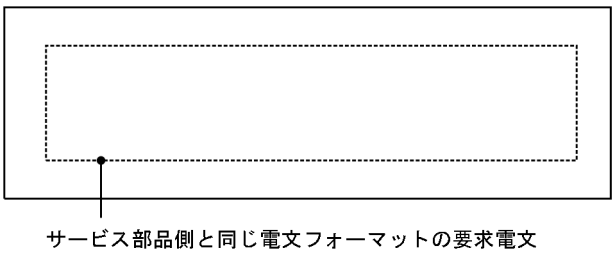
(2)　バイナリ電文の要求電文

バイナリ電文の要求電文を送信できるのは，サービス部品側で使用しているプロトコルが DB キューのときだけです。

なお，要求電文を指定しなかった場合，サービスアダプタでデータ変換処理が行われません。

バイナリ電文の要求電文を次の図に示します。

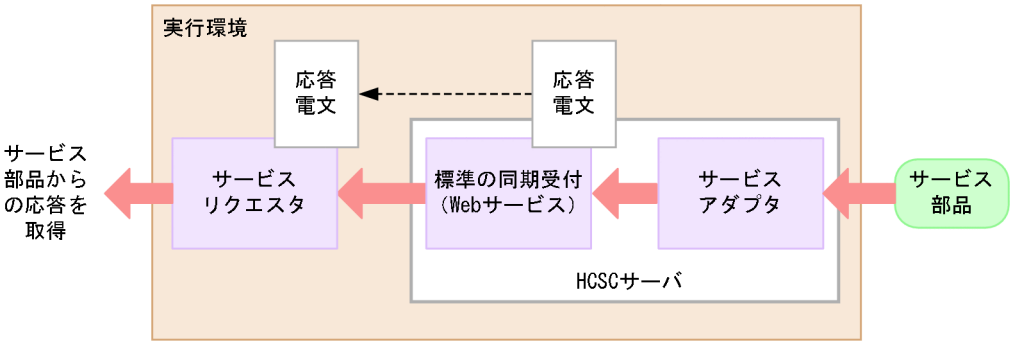
図 9-6　バイナリ電文の要求電文



9.2.7　応答電文を取得する

HCSC サーバの同期受付（Web サービス）から，サービス部品の実行要求に対する応答電文を取得します。サービスリクエスタは，サービス部品側と同じ電文フォーマットの応答電文を取得します。応答電文の取得を次の図に示します。

図 9-7　応答電文の取得（標準の同期受付（Web サービス））

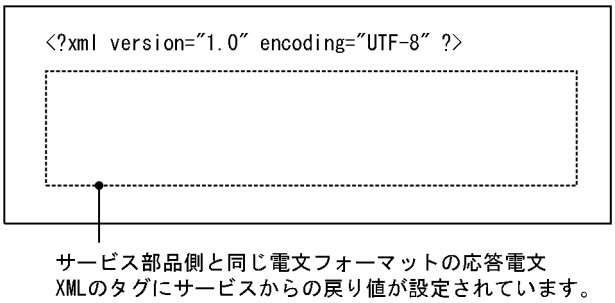


また，サービスリクエスタは，電文種別が XML 電文の応答電文を取得します。サービス部品からの応答電文がない場合は，NULL または 0 バイトのバイト配列を受信します。応答電文がある場合とない場合に分けて次に説明します。

(1)　応答電文がある場合

サービスリクエスタが取得する，XML 電文の応答電文を次に示します。

図 9-8　XML 電文の応答電文



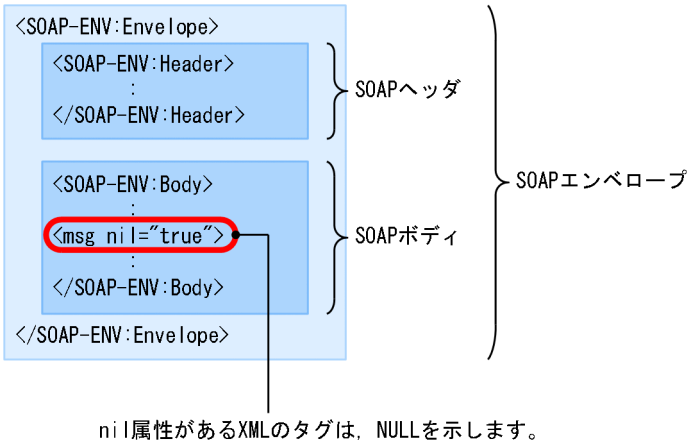
(2)　応答電文がない場合

応答電文がない場合，応答が String 型か byte[]型かによって，サービスリクエスタが受信する形式が異なります。

- 応答が String 型の場合

  サービスリクエスタは，次の図に示すような形式の NULL を受信します。サービスリクエスタで実装している SOAP エンジンによっては，形式が異なる場合があります。

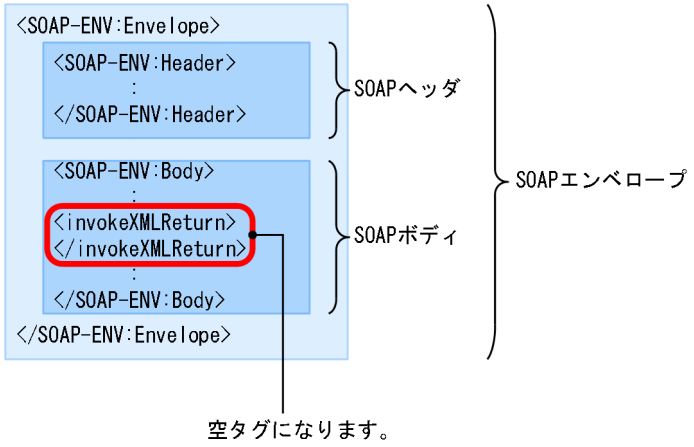
  図 9-9　受信する NULL の形式



- 応答が byte[]型の場合

  サービスリクエスタは，次の図に示すような形式の 0 バイトのバイト配列を受信します。SOAP エンベロープは次に示すように空タグとなります。サービスリクエスタで実装している SOAP エンジンによっては，形式が異なる場合があります。

図 9-10　応答が byte[]型の形式



9.2.8　エラー情報を取得する

要求先のサービス部品，HCSC サーバ，および SOAP エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。エラーの伝わり方については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.7.1　Web サービス（SOAP 通信）実行時の障害対策」を参照してください。

(1)　サービスリクエスタ側の実装例

エラー情報の取得方法は，SOAP 通信基盤の種類によって異なります。

(a)　サービスプラットフォームが提供している SOAP 通信基盤を使用している場合

サービスプラットフォームが提供している SOAP 通信基盤を使用している場合，CSCMsgServerFaultException オブジェクトをキャッチして，SOAP Fault のエラー情報を取得します。

エラー情報を取得するには，サービスリクエスタ側で次のように実装する必要があります。

```
        :
} catch (CSCMsgServerFaultException e) {
    System.err.println("Exception ErrorMessage = "
                + e.getErrorMessage() );
    System.err.println("Exception ErrorCode = "
                + e.getErrorCode() );
    System.err.println("Exception ProcessInstanceID = "
                + e.getProcessInstanceID() );
    System.err.println("Exception FaultCode = "
                + e.getCscmsgFaultCode() );
    System.err.println("Exception FaultString = "
                + e.getCscmsgFaultString() );
    System.err.println("Exception FaultActor = "
                + e.getCscmsgFaultActor() );
    System.err.println("Exception FaultDetails = "
                + new String(e.getCscmsgFaultDetail(), "UTF-8"));
```

```
    System.err.println("Exception FaultName = "
                + e.getFaultName() );
}
        :
```

各メソッドについて，次に説明します。

●getErrorMessage

説明

エラーメッセージを取得します。

次の例外の内容を取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品やビジネスプロセスからのフォルト

形式

public java.lang.String getErrorMessage()

●getErrorCode

説明

エラーコードを取得します。

次の例外の内容に対するエラーコードを取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品やビジネスプロセスからのフォルト

形式

public java.lang.String getErrorCode()

●getProcessInstanceID

説明

ビジネスプロセスインスタンス ID を取得します。

形式

public String getProcessInstanceID()

●getCscmsgFaultCode

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報を取得します。

形式

public java.lang.String getCscmsgFaultCode()

●getCscmsgFaultString

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報を取得します。

形式

public java.lang.String getCscmsgFaultString()

●getCscmsgFaultActor

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報を取得します。

形式

public java.lang.String getCscmsgFaultActor()

●getCscmsgFaultDetail

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報を取得します。

サービス部品が設定した Detail 情報をバイト配列でサービスリクエスタに引き渡します。したがって，取得したバイト配列を文字列に変換する必要があります。

なお，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報がない場合は，0 バイトのバイト配列（SOAP メッセージの空タグ）で応答します。

形式

public byte[] getCscmsgFaultDetail()

●getFaultName

説明

サービス部品（Web サービス）またはビジネスプロセスからの例外名を取得します。

形式

public String getFaultName()

(b) サービスプラットフォームが提供している SOAP 通信基盤を使用していない場合

取得するエラー情報は，サービスリクエスタ側で実装している SOAP エンジンに依存します。

(2) エラー情報（SOAP Fault）の形式

エラー情報（SOAP Fault）の形式を次の表に示します。

表 9-4　エラー情報（SOAP Fault）の形式

要素	名称	内容
faultcode	Fault コード	SOAP エンジンに依存する値です。 message part のメッセージ型属性が参照する QCName です。
faultstring	Fault 文字列	SOAP エンジンに依存する値です。 KDCCP0015-E を出力します。
faultactor	Fault 生成者	SOAP エンジンに依存する値です。値はありません。
detail※	Fault 詳細	wsdl:fault に対応する内容です。

注※

要素「detail」はエラー情報の詳細です。次の表に示す構造体で格納されます。

表 9-5　エラー情報（SOAP Fault）の詳細

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
errorMessage	次に示すエラーの内容です。 • HCSC サーバ内で検知したエラー • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
errorCode	次に示す例外の内容に対応するエラーコードです。 • HCSC サーバ内で検知したエラー • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
processInstanceID	ビジネスプロセスのインスタンス ID の情報です。 ビジネスプロセスでエラーが発生した場合に値が設定されます。	
cscmsgFaultCode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報です。	−
cscmsgFaultString	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報です。	−
cscmsgFaultActor	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報です。	−
cscmsgFaultDetail	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。	−
faultName	サービス部品（Web サービスまたは SessionBean）またはビジネスプロセスからのフォルト名（例外名）情報です。 次の場合に値が設定されます。 • サービス部品（Web サービスまたは SessionBean）からのユーザ定義例外の SOAP Fault の場合 • ビジネスプロセスからのフォルトの場合	−

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
faultName	SOAP Fault 動作定義ファイルに targetNamespace の URI を定義した Web サービスからの SOAP Fault の場合，値は設定されません。SOAP Fault 動作定義ファイルについては，マニュアル「サービスプラットフォーム リファレンス」の「6.6.1　SOAP Fault 動作定義ファイル」を参照してください。	－

（凡例）

－：該当しません。

9.2.9　ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤）

標準の同期受付（Web サービス）に要求を出すサービスリクエスタとして，ビジネスプロセスの再実行を要求するサービスリクエスタを作成できます。

標準の同期受付（Web サービス）にビジネスプロセスの再実行を要求するサービスリクエスタを作成する流れは，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。作成の流れについては，「9.2.1　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（SOAP 通信基盤）」を参照してください。

各工程で実施する内容を次に示します。

(1)　WSDL の取得

WSDL の取得方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

WSDL の取得方法については，「9.2.2　WSDL を取得する」を参照してください。

(2)　スタブの生成

スタブの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同様に，サービスプラットフォームが開発支援コマンドとして提供する WSDL2Java コマンドで生成します。

スタブの生成については，「9.2.3　スタブを生成する」を参照してください。

(3) オブジェクトの生成

オブジェクトの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

オブジェクトの生成については，「9.2.4　オブジェクトを生成する」を参照してください。

なお，ビジネスプロセスの再実行を要求するメソッドは，invokeBPXML()を使用して呼び出します。

ビジネスプロセスの再実行を要求するメソッドの例を次に示します。

（例）ビジネスプロセスの再実行を要求する場合

```
String result = ws.invokeBPXML(              // メソッドの呼び出し
                        serviceName,         // サービス名
                        bpRequestType,       // ビジネスプロセス用要求タイプ
                        bpProcessId,         // ビジネスプロセス用プロセスID
                        clientID,            // クライアント相関ID
                        requestFormatID,     // リクエストフォーマットID
                        responseFormatID,    // レスポンスフォーマットID
                        operationName,       // オペレーション名
                        userData);           // ユーザ電文
```

(4) パラメタの設定

メソッドの引数となるパラメタは，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と異なります。同期受付（Web サービス）にビジネスプロセスの再実行を要求するサービスリクエスタの場合に設定するパラメタの詳細を次に示します。

表 9-6　パラメタの詳細（標準の同期受付（Web サービス）／ビジネスプロセス再実行要求）

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
サービス名 (serviceName)	java.lang.String	in0（type="xsd:string"）	要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，開発環境で定義したビジネスプロセスを指定してください。
ビジネスプロセス用要求タイプ (bpRequestType)	java.lang.String	in1（type="xsd:string"）	要求電文の種類を示します。 ビジネスプロセスの再実行を要求する場合，文字列「RECOVER」を指定してください。※1
ビジネスプロセス用プロセス ID (bpProcessId)	java.lang.String	in2（type="xsd:string"）	ビジネスプロセスインスタンス ID です。 エラー情報から取得した値，または，メッセージログに出力された値を指定してください。
クライアント相関 ID (clientID)	java.lang.String	in3（type="xsd:string"）	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 このパラメタには NULL を指定してください。※1

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
リクエストフォーマット ID（requestFormatID）	java.lang.String	in4（type="xsd:string"）	サービスリクエスタからの要求電文フォーマットを一意に識別するための ID です。 このパラメタには NULL または空文字（""）を指定してください。※1
レスポンスフォーマット ID（responseFormatID）	java.lang.String	in5（type="xsd:string"）	HCSC サーバからの応答電文フォーマットを一意に識別するための ID です。 このパラメタには NULL または空文字（""）を指定してください。※1
オペレーション名（operationName）	java.lang.String	in6（type="xsd:string"）	要求先のサービス名に対するオペレーション名です。※2 このパラメタには NULL または空文字（""）を指定してください。※1
ユーザ電文（userData）	java.lang.String	in7（type="xsd:string"）	サービスリクエスタからの要求電文です。※2 このパラメタには NULL または空文字（""）を指定してください。※1

（凡例）

inX（X=0～7）：スタブで示しているメソッドごとのパラメタです。

注※1

これらのパラメタは，指定する値として固定値（「RECOVER」，NULL または空文字（""））が決められています。決められた値以外の値を指定すると，正常に動作しません。

注※2

要求先のサービス部品のプロトコルが SOAP の場合，ユーザ電文のルート要素の名称（データ変換を行う場合，データ変換後の電文のルート要素の名称）から呼び出すオペレーションを決定します。そのため，ユーザ電文のルート要素に誤った名称を指定した場合，意図しないオペレーションを呼び出してしまうので注意してください。

(5)　応答電文の取得

応答電文の取得方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

応答電文の取得については，「9.2.7　応答電文を取得する」を参照してください。

(6)　エラー情報の取得

エラー情報を取得するためのサービスリクエスタの実装は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

エラー情報の取得については，「9.2.8　エラー情報を取得する」を参照してください。

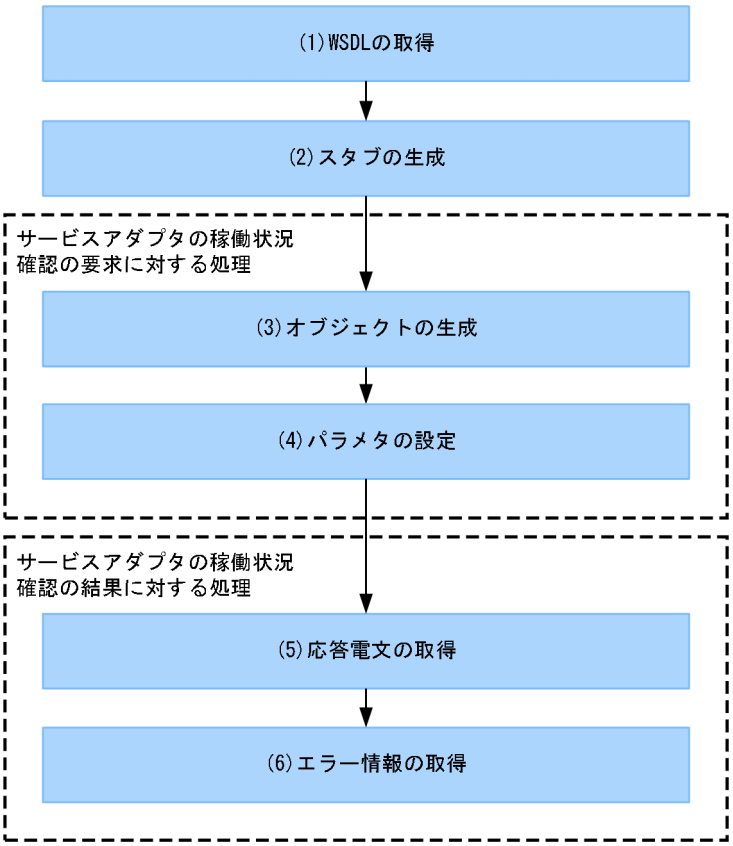
9.2.10　サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤）

標準の同期受付（Web サービス）に要求を出すサービスリクエスタとして，アプリケーションからサービスアダプタの稼働状況確認を要求するサービスリクエスタを作成できます。

サービスアダプタの稼働状況を確認するには，標準の同期受付（Web サービス）に要求を出すサービス部品の呼び出しとは別に，稼働状態確認用のメソッドを呼び出す処理を追加する必要があります。

標準の同期受付（Web サービス）にアプリケーションからサービスアダプタの稼働状況確認を要求するサービスリクエスタの作成手順を次に示します。

図 9-11　サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成手順（標準の同期受付（Web サービス・SOAP 通信基盤））



各工程の作業を次に示します。

(1)　WSDL の取得

WSDL の取得方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

なお，サービス部品の呼び出し用の処理と，稼働状態確認用の処理で，WSDL を使い分ける必要はありません。

WSDL の取得方法については，「9.2.2　WSDL を取得する」を参照してください。

(2)　スタブの生成

スタブの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同様に，サービスプラットフォームが開発支援コマンドとして提供する WSDL2Java コマンドで生成します。

スタブの生成については，「9.2.3　スタブを生成する」を参照してください。

(3)　オブジェクトの生成

サービスアダプタの稼働状況確認を要求するメソッドを呼び出すためのオブジェクトを生成する手順を次に示します。

1. サービス部品のインターフェースクラスである CSCMsgSyncServiceDeliveryWSImplServiceLocator クラスのオブジェクトを生成します。

   (例)

   ```
   CSCMsgSyncServiceDeliveryWSImplServiceLocator locator
               = new CSCMsgSyncServiceDeliveryWSImplServiceLocator();
   ```

2. サービス部品のインターフェースクラスのオブジェクトを使用して，サービスリクエスタのインターフェースクラスである CSCMsgSyncServiceDeliveryWSImpl.java クラスのオブジェクトを生成します。

   なお，生成・取得したサービスリクエスタのインターフェースクラスのインスタンスは，複数のスレッドで共有できません。

   (例)

   ```
   CSCMsgSyncServiceDeliveryWSImpl ws = null;
   try {
         ws = locator.getCSCMsgSyncServiceDeliveryWSImpl();
   }catch (ServiceException e) {
                e.printStackTrace();
                return;
   }
   ```

   サービス部品への接続先は，WSDL 定義内の Service 要素の address 子要素である location 属性となります。サービスリクエスタのプログラム内で，サービス部品への接続先情報を取得する場合は，次のように記述します。

   (例)

   ```
   String url = locator.getCSCMsgSyncServiceDeliveryWSImplAddress();
   ```

3. 生成した CSCMsgSyncServiceDeliveryWSImpl.java クラスのオブジェクトを使用して，同期受付（Web サービス）のサービスアダプタの稼働状況確認を要求するメソッドを呼び出します。

（例）option に"type=status"を指定した場合

```
String result = ws.getServiceInfo(          // メソッドの呼び出し
                        serviceName,        // サービス名
                        clientID,           // クライアント相関ID
                        "type=status");     // オプション
```

（例）option に"type=status,returnType=XML"を指定した場合

```
String result = ws.getServiceInfo(          // メソッドの呼び出し
                        serviceName,        // サービス名
                        clientID,           // クライアント相関ID
                        "type=status,returnType=XML");
                                            // オプション
```

(4) パラメタの設定

サービスアダプタの稼働状況確認を要求するためには，getServiceInfo メソッドを使用します。メソッドの引数となるパラメタは，標準の同期受付（Web サービス）に要求を出す通常のサービスリクエスタを作成する場合と異なります。

同期受付（Web サービス）にサービスアダプタの稼働状況確認を要求するサービスリクエスタの場合に設定するパラメタの詳細を次に示します。

表 9-7 パラメタの詳細（標準の同期受付（Web サービス）／サービスアダプタの稼働状況確認要求）

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
サービス名（serviceName）	java.lang.String	in0（type="xsd:string"）	サービスアダプタの稼働状況を確認するときの要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，HCSC サーバに配備されているサービスアダプタを指定してください。
クライアント相関 ID（clientID）	java.lang.String	in1（type="xsd:string"）	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 サービスリクエスタからの要求電文と，HCSC サーバで管理している実行履歴やログおよびトレースを対応づけるために使用します。したがって，HCSC サーバに送信した要求電文ごとに異なる ID を指定してください。 クライアント相関 ID を省略したい場合，NULL を指定してください。
オプション（option）	java.lang.String	in2（type="xsd:string"）	稼働状況の確認結果を取得するときに選択するオプションです。

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
オプション （option）	java.lang.String	in2（type="xsd:string"）	オプションの入力形式は，「キー名称=値」です。オプションの前後はダブルクォーテーション（"）で囲んでください。また，複数のオプションを指定する場合は，コンマ（,）で区切って指定してください。※1 （例） ”キー名称=値, キー名称=値” 指定できるオプションを次に示します。 type={all/status} 取得する情報のパターンを指定するオプションです。 • all（初期値） serviceName に指定したサービス名に該当するすべての情報を応答します。 • status serviceName に指定したサービス名に該当する状態，HCSC サーバ名，およびクラスタ名を応答します。 returnType={Properties/String/XML} 応答の形式を指定するオプションです。 • Properties（初期値） java.util.Properties クラスに含まれるすべてのプロパティを XML ドキュメント形式※2 で応答します。なお，値が存在しない場合は，プロパティのキーは存在しません。※3 • String java.util.Properties クラスを toString()変換した形式で応答します。なお，値が存在しない場合は，プロパティのキーは存在しません。 • XML HCSC サーバが定義する XML スキーマの XML ドキュメント形式で応答します。なお，値が存在しない場合は，空タグで応答します。

（凡例）

inX（X=0～2）：スタブで示しているメソッドごとのパラメタです。

注※1

コンマ（,）の前または後，イコール（=）の前または後に空白がある場合はエラーになります。また，文字列の最後がコンマ（,）にならないように指定してください。

注※2

Properties クラスの storeToXML()メソッドを使用して，UTF-8 エンコードした XML 形式です。この形式は，Properties クラスの loadFromXML()メソッドを使用して Properties クラスに復元できます。

注※3

Properties クラスで取得すると null を応答します。

(5)　応答電文の取得

パラメタのオプションで指定した形式でサービスアダプタの稼働状況確認の応答電文が返ります。

次に応答電文の形式と例を説明します。

(a)　応答電文の形式

応答電文で取得できる情報については，type オプションに指定する内容によって異なります。応答電文で取得できる情報を次に示します。

表 9-8　応答電文で取得できる情報

キーまたはタグ	情報の内容	type オプション指定による応答情報	
		all	status
HCSCServerName	HCSC サーバ名	○	○
ClusterName	クラスタ名	○	○
ServiceName	サービス名	○	×
ServiceStatus	サービスアダプタの状態	○	○
ServiceKind	サービスの種別	○	×
ServiceProtocolKind	サービスアダプタのプロトコル種別	○	×
AdapterName	サービスアダプタ名	○	×
AdapterLocalCall	ローカルキュー使用状況	○※	×
EntryTime	サービスアダプタ定義追加時刻	○	×
ModifiedTime	サービスアダプタ定義更新時刻	○	×

（凡例）

○：取得できます。

×：取得できません。

注※

ServiceProtocolKind が MDB_WSR の場合だけ応答します。

取得情報の詳細を次に説明します。

●HCSCServerName

説明

情報を取得した HCSC サーバ名を取得します。

●ClusterName

説明

サービスアダプタを所有しているクラスタ名を取得します。

●ServiceName

説明

オプションに"type=all"を指定した場合に，配備されているサービス名を取得します。オプションに"type=status"を指定した場合は，この情報は取得しません。

●ServiceStatus

説明

サービスアダプタの状態を取得します。

- active：起動状態
- inactive：停止状態
- starting：起動処理中
- startfailed：起動失敗状態
- stopping：停止処理中
- stopfailed：停止失敗状態
- deleting：削除処理中

●ServiceKind

説明

オプションに"type=all"を指定した場合に，配備されているサービスの種別を取得します。オプションに"type=status"を指定した場合は，この情報は取得しません。

- ServiceAdapter：サービスアダプタ

●ServiceProtocolKind

説明

オプションに"type=all"を指定した場合に，サービスアダプタのプロトコルの種別を取得します。オプションに"type=status"を指定した場合は，この情報は取得しません。

- WebService：SOAP（HTTP）（同期受付（Web サービス））
- SessionBean：RMI-IIOP（同期受付（SessionBean））
- MDB_WSR：WS-R（非同期受付（MDB（WS-R））
- MDB_DBQ：Reliable Messaging 独自のプロトコルまたは JMS（非同期受付（MDB（DB キュー）））
- Custom：サービスアダプタ

●AdapterName

説明

オプションに"type=all"を指定した場合に，配備されているサービスアダプタ名を取得します。オプションに"type=status"を指定した場合は，この情報は取得しません。

●AdapterLocalCall

説明

オプションに"type=all"を指定した場合に，サービスアダプタのローカルキューの使用状況を取得します。オプションに"type=status"を指定した場合は，この情報は取得しません。

この項目は ServiceProtocolKind が MDB_WSR の場合だけ応答します。

- true：ローカルキューを使用している
- false：ローカルキューを使用していない

●EntryTime

説明

オプションに"type=all"を指定した場合に，サービスアダプタを配備した時刻（定義登録時刻）を応答します。オプションに"type=status"を指定した場合は，この情報は応答しません。

形式

```
YYYY/MM/DD hh:mm:ss.SSS
```

- YYYY：西暦年
- MM：月
- DD：日
- hh：時
- mm：分
- ss：秒
- SSS：ミリ秒

●ModifiedTime

説明

オプションに"type=all"を指定した場合に，サービスアダプタを開発環境で定義した時刻（更新時刻）を応答します。オプションに"type=status"を指定した場合は，この情報は応答しません。

形式

```
YYYY/MM/DD hh:mm:ss
```

- YYYY：西暦
- MM：月
- DD：日
- hh：時
- mm：分
- ss：秒

(b) 応答電文の例

応答電文の例を次に示します。

(例 1) option に"type=all,returnType=Properties"を指定した場合

```
<?xml version="1.0" encoding="UTF-8"?>
<getServiceInfoResponse>
<HCSCServerName>HCSC</HCSCServerName>
<ClusterName>Cluster</ClusterName>
<ServiceName>Service1</ServiceName>
<ServiceStatus>Active</ServiceStatus>
<ServiceKind>ServiceAdapter</ServiceKind>
<ServiceProtocolKind>WebService</ServiceProtocolKind>
<AdapterName>ad1</AdapterName><AdapterLocalCall/>
<EntryTime>YYYY/MM/DD hh:mm:ss.SSS</EntryTime>
<ModifiedTime>YYYY/MM/DD hh:mm:ss</ModifiedTime>
</getServiceInfoResponse>
```

注 1

値の順番は変わることがあります（順不同です）。

注 2

option に"type=status,returnType=Properties"を指定した場合は，応答しない情報は応答電文に存在しません。

(例 2) option に"type=all,returnType=String"を指定した場合

```
{HCSCServerName=HCSC,ClusterName=Cluster,ServiceName=Service1,ServiceStatus=Active,ServiceKind=ServiceAdapter,ServiceProtocolKind=WebService,AdapterName=ad1,EntryTime=YYYY/MM/DD hh:mm:ss.SSS,ModifiedTime=YYYY/MM/DD hh:mm:ss}
```

注 1

値の順番は変わることがあります（順不同です）。

注 2

option に"type=status,returnType=String"を指定した場合は，応答しない情報は応答電文に存在しません。

(例 3) option に"type=all,returnType=XML"を指定した場合

```
<?xml version="1.0" encoding="UTF-8"?>
<getServiceInfoResponse>
<HCSCServerName>HCSC</HCSCServerName>
<ClusterName>Cluster</ClusterName>
<ServiceName>Service1</ServiceName>
<ServiceStatus>Active</ServiceStatus>
<ServiceKind>ServiceAdapter</ServiceKind>
<ServiceProtocolKind>WebService</ServiceProtocolKind>
<AdapterName>ad1</AdapterName>
<AdapterLocalCall/>
```

```
<EntryTime>YYYY/MM/DD hh:mm:ss.SSS</EntryTime>
<ModifiedTime>YYYY/MM/DD hh:mm:ss</ModifiedTime>
</getServiceInfoResponse>
```

注 1

値が存在しない場合は，空タグで応答します。

（例）

<AdapterLocalCall/>

注 2

オプションに"type=status,returnType=XML"を指定した場合は，応答しない情報の<タグ>は応答電文に存在しません。

(6) エラー情報の取得

サービスアダプタの稼働状況確認でエラーが発生した場合，CSCMsgServerFaultException オブジェクトをキャッチして，SOAP Fault のエラー情報を取得します。

エラー情報の取得については，「9.2.8 エラー情報を取得する」を参照してください。

エラーになる主な要因として，次の場合が挙げられます。

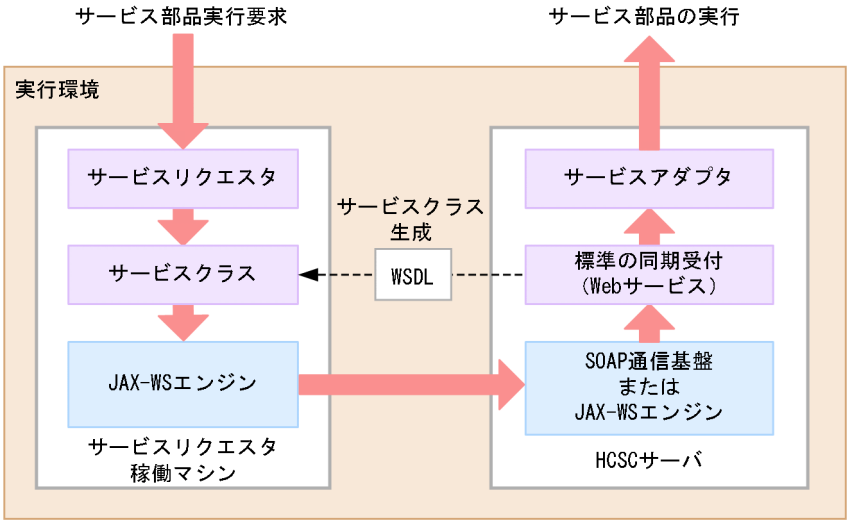
- 必須の引数が指定されていない。
- 引数の入力値に誤りがある（形式不正）。
- HCSC サーバに該当のサービス名のサービスアダプタが配備されていない。
- リクエスト受付が起動状態ではない（停止状態，起動処理中，起動失敗状態，停止処理中，停止失敗状態，削除処理中）。

9.3　標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（JAX-WS エンジン）

標準の同期受付（Web サービス）に要求を出すサービスリクエスタとして，JAX-WS エンジンを使用して通信するサービスリクエスタを作成できます。JAX-WS エンジンを使用して通信するサービスリクエスタは，SOAP（document-literal 型）を利用して，標準の同期受付（Web サービス）に要求電文を送信します。

JAX-WS エンジンの場合のサービスリクエスタと HCSC サーバの関係を次の図に示します。

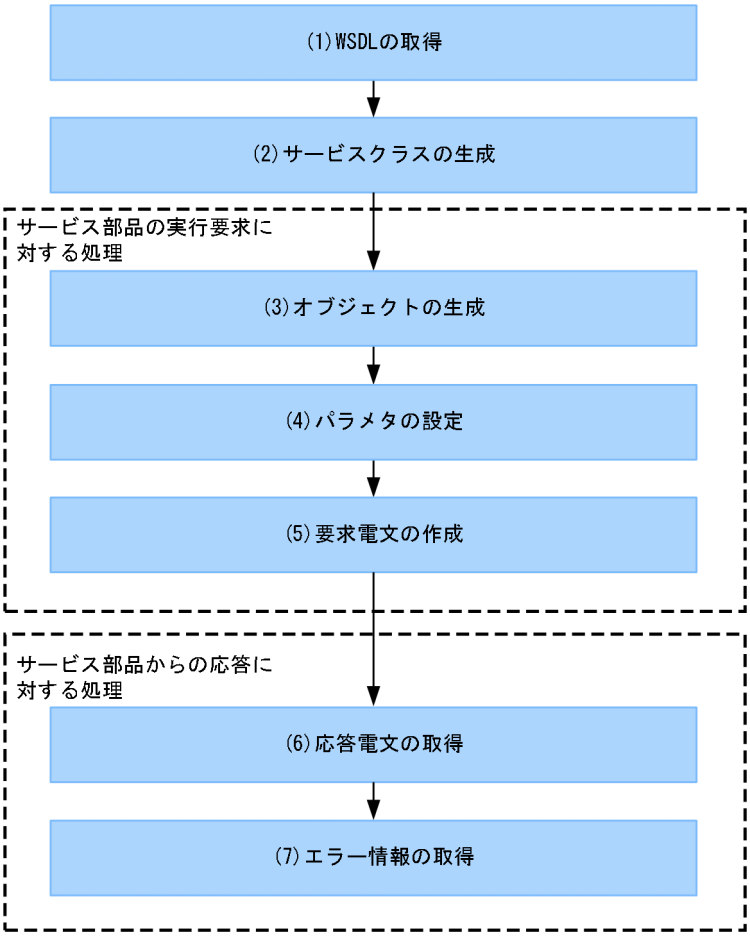
図 9-12　標準の同期受付（Web サービス）に要求を出すサービスリクエスタと HCSC サーバの関係（JAX-WS エンジン）



9.3.1　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（JAX-WS エンジン）

標準の同期受付（Web サービス）にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-13　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（JAX-WS エンジン）



各工程の作業を次に示します。

(1)　WSDL の取得

サービス部品を呼び出す HCSC サーバの同期受付（Web サービス）のインターフェース情報を WSDL から取得します。WSDL の取得については，「9.3.2　WSDL を取得する」を参照してください。

(2)　サービスクラスの生成

「9.3.2　WSDL を取得する」で取得した WSDL からサービスクラスを生成します。サービスクラスの生成については，「9.3.3　サービスクラスを生成する」を参照してください。

(3)　オブジェクトの生成

同期受付（Web サービス）のメソッドを呼び出すために，「9.3.3　サービスクラスを生成する」で生成したサービスクラスからプロキシクラスのオブジェクトを生成します。プロキシクラスのオブジェクトの生成については，「9.3.4　オブジェクトを生成する」を参照してください。

(4)　パラメタの設定

同期受付（Web サービス）のメソッドの引数となるパラメタを設定します。パラメタの設定については，「9.3.5　パラメタを設定する」を参照してください。

(5)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.3.6　要求電文を作成する」を参照してください。

(6)　応答電文の取得

同期受付（Web サービス）から，サービス部品の実行要求に対する応答電文を取得します。応答電文の取得については，「9.3.7　応答電文を取得する」を参照してください。

(7)　エラー情報の取得

要求先のサービス部品，HCSC サーバ，および JAX-WS エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。エラー情報の取得については，「9.3.8　エラー情報を取得する」を参照してください。

9.3.2　WSDL を取得する

サービス部品を呼び出す HCSC サーバの同期受付（Web サービス）のインターフェース情報を WSDL から取得します。WSDL を取得するには，次の 2 つの方法があります。

方法 1

1. **Eclipse のメニューから，［HCSC-Definer］－［公開サービス一覧］を選択します。**
   ツリービューの公開サービス一覧に，HCSC コンポーネントの一覧が表示されます。

2. **HCSC コンポーネントの一覧から，呼び出すサービス部品（HCSC コンポーネント）を選択し，ダブルクリックします。**
   HCSC コンポーネント情報表示画面に，HCSC コンポーネントの情報が表示されます。HCSC コンポーネント情報表示画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.6　HCSC コンポーネント情報表示画面」を参照してください。
   ［サーバ情報］には，サービス部品の要求先となる HCSC サーバの情報が表示されます。

3. **必要に応じて，［バインディングスタイル］ラジオボタンおよび［SOAP バージョン］ラジオボタンで，取得する WSDL の種類を選択します。**
   ［バインディングスタイル］ラジオボタンでは，取得する WSDL 定義のスタイルを選択します。［SOAP バージョン］ラジオボタンでは，対応する SOAP 仕様のバージョンを選択します。

4. [WSDL 取得] ボタンをクリックします。

［WSDL ファイルの保存］ダイアログが表示されます。

保存先を指定して，WSDL を取得します。

方法 2

1. HCSC-Messaging をインストールしたディレクトリから，WSDL のサンプルを取得します。

用途に合わせて次に示す WSDL のどちらかを取得してください。

- SOAP1.1 用／document-literal 型

  cscmsg_ws_doc.wsdl ファイル
- SOAP1.2 用／document-literal 型

  cscmsg_ws_doc_1_2.wsdl ファイル

HCSC-Messaging をインストールしたディレクトリについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.1.2　Service Platform のインストール」を参照してください。

2. WSDL の<soap:address>または<soap12:address>の location の内容を次のように変更します。

```
http://<ホスト名>:<ポート番号>※1/<コンテキストルート>※2/services/CSCMsgSyncServiceDeliveryWSImpl
```

注※1

HCSC サーバの URL です。

注※2

開発環境および運用環境で HCSC サーバ構成配備時に決定される HCSC サーバのクラスタ名です。SOAP1.2 の場合，クラスタ名に接尾辞"12"が付与されます。

9.3.3　サービスクラスを生成する

取得した WSDL からサービスクラスを生成します。サービスクラスは cjwsimport コマンドで生成します。

コマンドの入力例を次に示します。

```
cjwsimport -s <ソースファイルの出力先ディレクトリ> -d <コンパイル済みクラスファイルの出力先ディレクトリ> cscmsg_ws_doc.wsdl
```

JAX-WS エンジンの場合，WSDL のスタイルは document-literal になるため，「cscmsg_ws_doc.wsdl」と設定します。

なお，このコマンドの入力例は，「9.3.2　WSDL を取得する」で取得した WSDL（cscmsg_ws_doc.wsdl）が cjwsimport コマンドを実行するカレントディレクトリに格納されている場合の例になります。

cjwsimport コマンドのオプションなどの詳細については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「14.1.1　cjwsimport コマンド」を参照してください。

このコマンドを実行すると，指定したソースファイルの出力先ディレクトリに，次のディレクトリおよびファイルが生成されます。

```
/ソースファイルの出力先ディレクトリ
  ├jp/co/hitachi/soft/csc/msg/message/reception/ejb/
  │ ├CSCMsgServerFaultException.java
  │ │    //HCSCサーバ独自のフォルト情報が格納されるクラス
  │ ├CSCMsgServerFaultException_Exception.java
  │ │    //CSCMsgServerFaultExceptionをラッパする例外クラス
  │ ├CSCMsgSyncServiceDeliveryWSImpl.java
  │ │    //WSDLのportType要素に対応するサービスエンドポイントインターフェース
  │ ├CSCMsgSyncServiceDeliveryWSImplService.java
  │ │    //サービスクラス
  │ ├GetServiceInfo.java
  │ │    //getServiceInfoオペレーションの要求電文に対するJavaBeanクラス※1
  │ ├GetServiceInfoResponse.java
  │ │    //getServiceInfoオペレーションの応答電文に対するJavaBeanクラス※1
  │ ├InvokeBinary.java
  │ │    //invokeBinaryオペレーションの要求電文に対するJavaBeanクラス※1
  │ ├InvokeBinaryResponse.java
  │ │    //invokeBinaryオペレーションの応答電文に対するJavaBeanクラス※1
  │ ├InvokeBPXML.java
  │ │    //invokeBPXMLオペレーションの要求電文に対するJavaBeanクラス※1
  │ ├InvokeBPXMLResponse.java
  │ │    //invokeBPXMLオペレーションの応答電文に対するJavaBeanクラス※1
  │ ├InvokeXML.java
  │ │    //invokeXMLオペレーションの要求電文に対するJavaBeanクラス※1
  │ ├InvokeXMLResponse.java
  │ │    //invokeXMLオペレーションの応答電文に対するJavaBeanクラス※1
  │ ├ObjectFactory.java
  │ │    //JAXB2.1仕様のObjectFactoryクラス
  │ └package-info.java
  │      //JAXB2.1で使用するファイル
  └cscmsg_ws_doc.wsdl※2
```

注※1

JAXB2.1 で使用します。

注※2

この出力例は，次の状態で cjwsimport コマンドを実行した場合の例になります。

- cjwsimport コマンドの-s オプションに指定したディレクトリに「9.3.2　WSDL を取得する」で取得した WSDL（cscmsg_ws_doc.wsdl）が格納されている。cjwsimport コマンドに指定した WSDL も，-s オプションに指定したディレクトリに存在する WSDL を指定している。
- cjwsimport コマンドで-s オプションを省略して実行し，cjwsimport コマンドを実行するカレントディレクトリに「9.3.2　WSDL を取得する」で取得した WSDL（cscmsg_ws_doc.wsdl）が格納されている。cjwsimport コマンドに指定した WSDL もディレクトリを指定しないで，カレントディレクトリに存在する WSDL を指定している。

なお，cjwsimport コマンドに指定した WSDL（cscmsg_ws_doc.wsdl）は，コマンドを実行したあと，-s オプションに指定したソースファイルの出力先ディレクトリには出力されません。

注意事項

JAX-WS エンジンに対応したサービスリクエスタの場合，プログラム実行時に WSDL を読み込みます。サービスクラスのデフォルトコンストラクタを使用する場合は，cjwsimport コマンドで指定した WSDL のパス（「9.3.2　WSDL を取得する」で取得した WSDL ファイルをサービスクラス生成のために格納したディレクトリ）の WSDL を読み込みます。そのため，cjwsimport コマンドを実行したあと，サービスクラスを基点とした WSDL ファイルの構成位置は，相対関係を壊すような構成にならないよう，WSDL ファイルを移動しないでください。サービスリクエスタ実行時に参照する WSDL の構成位置を，cjwsimport コマンドで指定した WSDL のパスから変更する場合は，URL を指定できるコンストラクタを使用してください。

9.3.4　オブジェクトを生成する

同期受付（Web サービス）のメソッドを呼び出すために，生成したサービスクラスを使用して，プロキシクラスのオブジェクトを生成します。プロキシクラスのオブジェクトを生成する手順を次に示します。

1. CSCMsgSyncServiceDeliveryWSImplService クラスのオブジェクトを生成します。

（例）

```
CSCMsgSyncServiceDeliveryWSImplService service
            = new CSCMsgSyncServiceDeliveryWSImplService();
```

2. wsdl:portType に対応するプロキシクラスである CSCMsgSyncServiceDeliveryWSImpl.java クラスを生成します。

（例）

```
CSCMsgSyncServiceDeliveryWSImpl ws
            = service.getCSCMsgSyncServiceDeliveryWSImpl();
```

3. 生成した CSCMsgSyncServiceDeliveryWSImpl.java クラスのオブジェクトを使用して，同期受付（Web サービス）のメソッドを呼び出します。

（例）要求電文が XML の場合

```
String result = ws.invokeXML(                    // メソッドの呼び出し
                        serviceName,             // サービス名
                        clientID,                // クライアント相関ID
                        requestFormatID,         // リクエストフォーマットID
                        responseFormatID,        // レスポンスフォーマットID
                        operationName,           // オペレーション名
                        userData);               // ユーザ電文
```

（例）要求電文がバイナリの場合

```
byte[] resultBinary = ws.invokeBinary(           // メソッドの呼び出し
                            serviceName,         // サービス名
                            clientID,            // クライアント相関ID
```

```
                                requestFormatID,          // リクエストフォーマットID
                                responseFormatID,         // レスポンスフォーマットID
                                operationName,            // オペレーション名
                                userDataBinary.length,    // ユーザ電文長
                                userDataBinary);          // ユーザ電文
```

注意事項

バイナリ電文の要求電文を送信できるのは，サービス部品側で使用している電文形式がバイナリ形式のときだけです。

9.3.5　パラメタを設定する

メソッドの引数となるパラメタは，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤のパラメタの設定については，「9.2.5　パラメタを設定する」を参照してください。

9.3.6　要求電文を作成する

要求電文の作成方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤の要求電文の作成については，「9.2.6　要求電文を作成する」を参照してください。

9.3.7　応答電文を取得する

応答電文の取得方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤の応答電文の取得については，「9.2.7　応答電文を取得する」を参照してください。

9.3.8　エラー情報を取得する

要求先のサービス部品，HCSC サーバ，および JAX-WS エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。

JAX-WS エンジンを使用して通信するサービスリクエスタの場合，情報がラップされた例外（SOAP1.1 の場合は CSCMsgServerFaultException_Exception，SOAP1.2 の場合は CSCMsgServerFault12Exception_Exception）※がキャッチされます。そのため，getFaultInfo()メソッドを使用してフォルト情報の格納クラスを取得する必要があります。

注※

cjwsimport コマンド実行時に，"_"（アンダーバー）は除去され，フォルト情報の格納クラス（FaultBean）との名前重複を回避するために接尾辞"_Exception"が付与された例外名となります。

(1) サービスリクエスタ側の実装例

getFaultInfo()メソッドを使用して，フォルト情報の格納クラスを取得する場合の，サービスリクエスタ側の実装の例を次に示します。

- SOAP1.1 の場合

```
/**
 * Sample Program
 */
{
    try {
         :
      // Webサービス呼び出し
         :
    } catch (CSCMsgServerFaultException_Exception e) {
      // CSC内部で例外またはユーザ定義例外が発生

      // CSCで定義している例外オブジェクトを取得
      CSCMsgServerFaultException faultInfo = e.getFaultInfo();

      // 例外情報を出力
      System.err.println("errorCode="
                         + faultInfo.getErrorCode());
      System.err.println("errorMessage="
                         + faultInfo.getErrorMessage());
      System.err.println("processInstanceID="
                         + faultInfo.getProcessInstanceID());
      System.err.println("faultCode="
                         + faultInfo.getCscmsgFaultCode());
      System.err.println("faultActor="
                         + faultInfo.getCscmsgFaultActor());
      System.err.println("faultString="
                         + faultInfo.getCscmsgFaultString());
      System.err.println("faultName="
                         + faultInfo.getFaultName());
      byte[] faultDetail = faultInfo.getCscmsgFaultDetail();
      try {
        if(faultDetail != null) {
          System.err.println("faultDetail="
                         + new String(faultDetail, "UTF-8"));
        }
      } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
      }
    } catch (SOAPFaultException e) {
         :
    }
}
```

- SOAP1.2 の場合

```
/**
 * Sample Program
 */
{
    try {
        :
      // Webサービス呼び出し
        :
    } catch (CSCMsgServerFault12Exception_Exception e) {
      // CSC内部で例外またはユーザ定義例外が発生

      // CSCで定義しているフォルト情報のオブジェクトを取得
      CSCMsgServerFault12Exception faultInfo = e.getFaultInfo();

      // 例外情報を出力
      System.err.println("errorCode="
                         + faultInfo.getErrorCode());
      System.err.println("errorMessage="
                         + faultInfo.getErrorMessage());
      System.err.println("processInstanceID="
                         + faultInfo.getProcessInstanceID());
      CscmsgFaultCode code = faultInfo.getCscmsgFaultCode();
      if (code == null) {
        System.err.println("Code=" + code);
      } else {
        List<String> values = code.getValues();
        for(String value : values) {
          System.err.println("Code Value=" + value);
        }
      }
      CscmsgFaultReason reason = faultInfo.getCscmsgFaultReason();
      if (reason == null) {
        System.err.println("Reason=" + reason);
      } else {
        List<CscmsgFaultReasonText> texts =
          reason.getCscmsgFaultReasonText();
        Locale locale = Locale.getDefault();
        for(CscmsgFaultReasonText text : texts) {
          if(locale.equals(text.getLocale())) {
            System.err.println("Reason=" + text.getText());
          }
        }
      }
      System.err.println("Role="
                         + faultInfo.getCscmsgFaultRole());
      System.err.println("Node="
                         + faultInfo.getCscmsgFaultNode());
      System.err.println("faultName="
                         + faultInfo.getFaultName());
      byte[] faultDetail = faultInfo.getCscmsgFaultDetail();
      try {
        if(faultDetail != null) {
          System.err.println("faultDetail="
                         + new String(faultDetail, "UTF-8"));
        }
      } catch (UnsupportedEncodingException e1) {
        e1.printStackTrace();
```

```
      }
    } catch (SOAPFaultException e) {
        :
    }
}
```

SOAP エンベロープの detail 情報には，Web サービスからの例外（SOAP Fault）の場合には，サービス（SOAP Fault 発行元）が返してきた SOAP Fault の情報と HCSC-Messaging が設定したエラーメッセージ，およびエラーコードが含まれます。

HCSC-Messaging で検出した例外（Exception）の場合や，Web サービス以外のサービスからの例外（Exception）の場合には，HCSC-Messaging で検出したエラーのエラーコードとエラー情報が含まれます。

各メソッドについて，次に説明します。

(a) 例外情報保持クラスが持つメソッド

●getErrorMessage（SOAP1.1／SOAP1.2 共通）

説明

エラーメッセージを取得します。

次の例外の内容を取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品やビジネスプロセスからのフォルト

形式

public java.lang.String getErrorMessage()

●getErrorCode（SOAP1.1／SOAP1.2 共通）

説明

エラーコードを取得します。

次の例外の内容に対するエラーコードを取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品やビジネスプロセスからのフォルト

形式

public java.lang.String getErrorCode()

●getProcessInstanceID（SOAP1.1／SOAP1.2 共通）

説明

ビジネスプロセスインスタンス ID を取得します。

形式

public String getProcessInstanceID()

●getCscmsgFaultCode（SOAP1.1 用）

説明

SOAP1.1 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報を取得します。

形式

public java.lang.String getCscmsgFaultCode()

●getCscmsgFaultString（SOAP1.1 用）

説明

SOAP1.1 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報を取得します。

形式

public java.lang.String getCscmsgFaultString()

●getCscmsgFaultActor（SOAP1.1 用）

説明

SOAP1.1 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報を取得します。

形式

public java.lang.String getCscmsgFaultActor()

●getCscmsgFaultCode（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode または Code 情報を保持する CscmsgFaultCode クラスを取得します。

形式

public jp.co.hitachi.soft.csc.msg.message.reception.ejb.CscmsgFaultCode
getCscmsgFaultCode()

●getCscmsgFaultReason（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Reason 情報を保持する CscmsgFaultReason クラスを取得します。

形式

public jp.co.hitachi.soft.csc.msg.message.reception.ejb.CscmsgFaultReason
getCscmsgFaultReason()

●getCscmsgFaultRole（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Role 情報を取得します。

形式

public java.lang.String getCscmsgFaultRole()

●getCscmsgFaultNode（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Node 情報を取得します。

形式

public java.lang.String getCscmsgFaultNode()

●getCscmsgFaultDetail（SOAP1.1／SOAP1.2 共通）

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報を取得します。

サービス部品が設定した Detail 情報をバイト配列でサービスリクエスタに引き渡します。したがって，取得したバイト配列を文字列に変換する必要があります。

なお，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報がない場合は，0 バイトのバイト配列（SOAP メッセージの空タグ）で応答します。

形式

public byte[] getCscmsgFaultDetail()

●getFaultName（SOAP1.1／SOAP1.2 共通）

説明

サービス部品（Web サービス）またはビジネスプロセスからの例外名を取得します。

形式

public String getFaultName()

(b) CscmsgFaultCode クラスが持つメソッド

●getValues（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode または Code 情報が持つ Value 情報を取得します。

形式

public java.util.List<java.lang.String> getValues()

(c) CscmsgFaultReason クラスが持つメソッド

●getTexts（SOAP1.2 用）

説明

SOAP1.2 による通信の場合に，サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Reason 情報が持つ Text 情報を取得します。

形式

public java.util.List<java.lang.String> getTexts()

(2) エラー情報（SOAP Fault）の形式

エラー情報（SOAP Fault）の形式を次の表に示します。

表 9-9 エラー情報（SOAP Fault）の形式

要素		名称	内容
SOAP1.1	SOAP1.2		
faultcode	Code	Fault コード	JAX-WS エンジンに依存する値です。 違反コードを出力します。
faultstring	－	Fault 文字列	JAX-WS エンジンに依存する値です。 エラーメッセージを出力します。
－	Reason	Fault 理由	JAX-WS エンジンに依存する値です。 エラーメッセージを出力します。
faultactor	Role	Fault 生成者	JAX-WS エンジンに依存する値です。値はありません。
－	Node	Fault ノード	JAX-WS エンジンに依存する値です。値はありません。
detail※1	Detail※2	Fault 詳細	wsdl:fault に対応する内容です。

（凡例）

－：該当しません。

注※1

要素「detail」はエラー情報の詳細です。「表 9-10」に示す構造体で格納されます。

注※2

要素「Detail」はエラー情報の詳細です。「表 9-11」に示す構造体で格納されます。

表 9-10 エラー情報（SOAP Fault）の詳細（SOAP1.1）

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
errorMessage	次に示すエラーの内容です。 • HCSC サーバ内で検知したエラー	

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
errorMessage	• サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
errorCode	次に示す例外の内容に対応するエラーコードです。 • HCSC サーバ内で検知したエラー • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
processInstanceID	ビジネスプロセスのインスタンス ID の情報です。 ビジネスプロセスでエラーが発生した場合に値が設定されます。	
cscmsgFaultCode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報です。	−
cscmsgFaultString	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報です。	−
cscmsgFaultActor	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報です。	−
cscmsgFaultDetail	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。	−
faultName	サービス部品（Web サービスまたは SessionBean）またはビジネスプロセスからのフォルト名（例外名）情報です。 次の場合に値が設定されます。 • サービス部品（Web サービスまたは SessionBean）からのユーザ定義例外の SOAP Fault の場合 • ビジネスプロセスからのフォルトの場合 SOAP Fault 動作定義ファイルに targetNamespace の URI を定義した Web サービスからの SOAP Fault の場合，値は設定されません。SOAP Fault 動作定義ファイルについては，マニュアル「サービスプラットフォーム リファレンス」の「6.6.1　SOAP Fault 動作定義ファイル」を参照してください。	−

（凡例）

−：該当しません。

表 9-11　エラー情報（SOAP Fault）の詳細（SOAP1.2）

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
errorMessage	次に示すエラーの内容です。 • HCSC サーバ内で検知したエラー • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
errorCode	次に示す例外の内容に対応するエラーコードです。	

フィールド名	説明	
	サービス部品，ビジネスプロセス，サービスアダプタ，または連携する HCSC サーバからのエラー（フォルト）	HCSC サーバで検知したエラー（例外）
errorCode	• HCSC サーバ内で検知したエラー • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー	
processInstanceID	ビジネスプロセスのインスタンス ID の情報です。 ビジネスプロセスでエラーが発生した場合に値が設定されます。	
cscmsgFaultCode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報です。	−
cscmsgFaultReason	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultReason 情報です。	−
cscmsgFaultRole	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Role 情報です。	−
cscmsgFaultDetail	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。	−
faultName	サービス部品（Web サービスまたは SessionBean）またはビジネスプロセスからのフォルト名（例外名）情報です。 次の場合に値が設定されます。 • サービス部品（Web サービスまたは SessionBean）からのユーザ定義例外の SOAP Fault の場合 • ビジネスプロセスからのフォルトの場合 SOAP Fault 動作定義ファイルに targetNamespace の URI を定義した Web サービスからの SOAP Fault の場合，値は設定されません。SOAP Fault 動作定義ファイルについては，マニュアル「サービスプラットフォーム リファレンス」の「6.6.1　SOAP Fault 動作定義ファイル」を参照してください。	−

（凡例）

−：該当しません。

9.3.9　ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン）

標準の同期受付（Web サービス）に要求を出すサービスリクエスタとして，ビジネスプロセスの再実行を要求するサービスリクエスタを作成できます。

サービスリクエスタと標準の同期受付（Web サービス）の間は，document/literal スタイルの SOAP メッセージで送受信されます。

標準の同期受付（Web サービス）にビジネスプロセスの再実行を要求するサービスリクエスタを作成する流れは，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同

じです。作成の流れについては，「9.3.1　サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（JAX-WS エンジン）」を参照してください。

各工程で実施する内容を次に示します。

(1)　WSDL の取得

WSDL の取得方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

WSDL の取得方法については，「9.3.2　WSDL を取得する」を参照してください。

(2)　サービスクラスの生成

サービスクラスの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

サービスクラスの生成方法については，「9.3.3　サービスクラスを生成する」を参照してください。

(3)　オブジェクトの生成

オブジェクトの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

オブジェクトの生成については，「9.3.4　オブジェクトを生成する」を参照してください。

なお，ビジネスプロセスの再実行を要求するメソッドは，invokeBPXML()を使用して呼び出します。

(4)　パラメタの設定

メソッドの引数となるパラメタは，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤のパラメタの設定については，「9.2.5　パラメタを設定する」を参照してください。

(5)　要求電文の作成

要求電文の作成方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤の要求電文の作成については，「9.2.6　要求電文を作成する」を参照してください。

(6)　応答電文の取得

応答電文の取得方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤の応答電文の取得については，「9.2.7　応答電文を取得する」を参照してください。

(7)　エラー情報の取得

エラー情報を取得するためのサービスリクエスタの実装は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

エラー情報の取得については，「9.3.8　エラー情報を取得する」を参照してください。

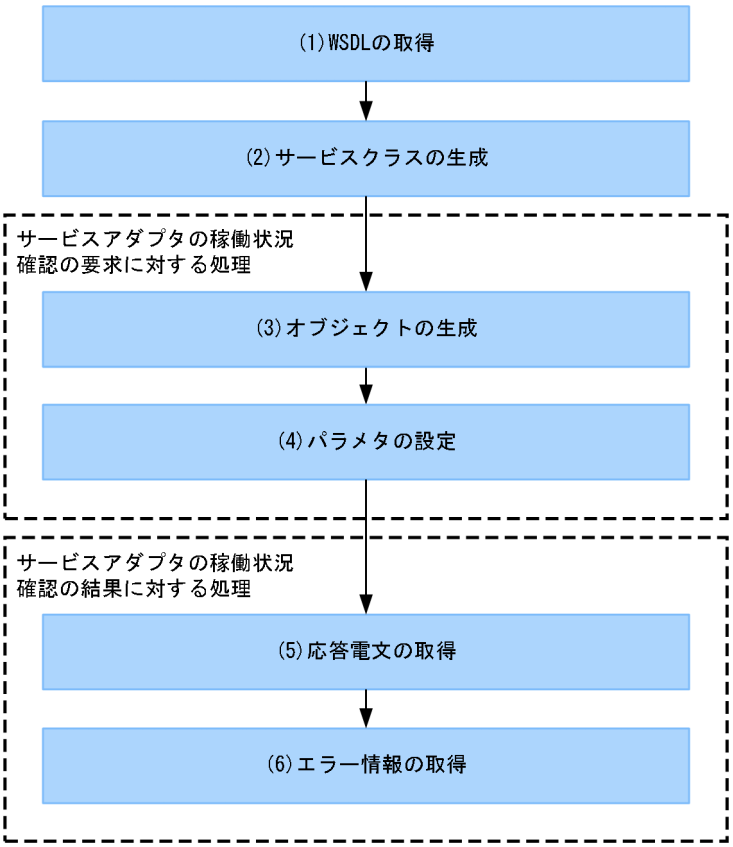
9.3.10　サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン）

標準の同期受付（Web サービス）に要求を出すサービスリクエスタとして，アプリケーションからサービスアダプタの稼働状況確認を要求するサービスリクエスタを作成できます。

サービスアダプタの稼働状況を確認するには，標準の同期受付（Web サービス）に要求を出すサービス部品の呼び出しとは別に，稼働状態確認用のメソッドを呼び出す処理を追加する必要があります。

標準の同期受付（Web サービス）にアプリケーションからサービスアダプタの稼働状況確認を要求するサービスリクエスタの作成手順を次に示します。

図 9-14　サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成手順（標準の同期受付（Web サービス・JAX-WS エンジン））



各工程の作業を次に示します。

(1)　WSDL の取得

WSDL の取得方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

WSDL の取得方法については，「9.3.2　WSDL を取得する」を参照してください。

(2)　サービスクラスの生成

サービスクラスの生成方法は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

サービスクラスの生成方法については，「9.3.3　サービスクラスを生成する」を参照してください。

(3)　オブジェクトの生成

オブジェクトの生成方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

オブジェクトの生成については，「9.2.10(3)　オブジェクトの生成」を参照してください。

なお，サービスのメソッドを呼び出すには，クラスのオブジェクトを使用して，次のように実行します。

```
String result = ws.getServiceInfo(          // メソッドの呼び出し
                      serviceName,          // サービス名
                      clientID,             // クライアント相関ID
                      "type=all,returnType=Properties");      // オプション
```

(4)　パラメタの設定

メソッドの引数となるパラメタは，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤のパラメタの設定については，「9.2.10(4)　パラメタの設定」を参照してください。

(5)　応答電文の取得

応答電文の取得方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

SOAP 通信基盤の応答電文の取得については，「9.2.10(5)　応答電文の取得」を参照してください。

(6)　エラー情報の取得

エラー情報を取得するためのサービスリクエスタの実装は，標準の同期受付（Web サービス）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

エラー情報の取得については，「9.3.8　エラー情報を取得する」を参照してください。

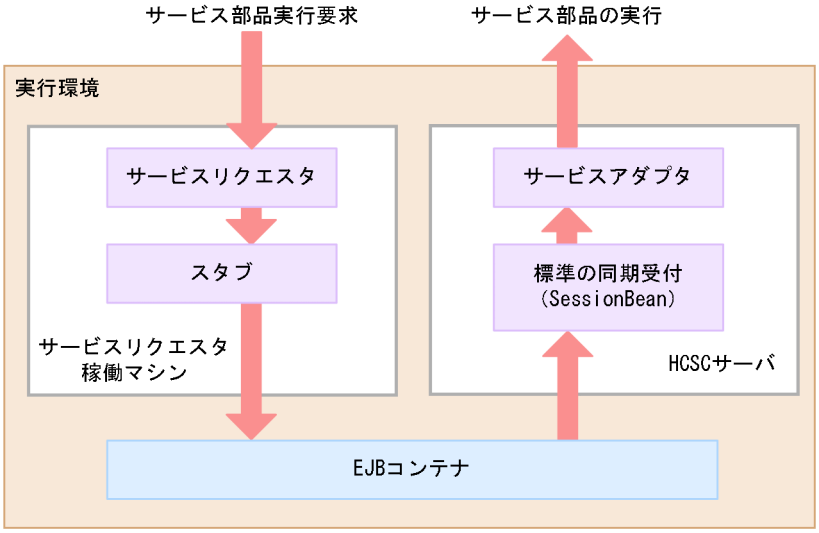
9.4　標準の同期受付（SessionBean）に要求を出すサービスリクエスタ

標準の同期受付（SessionBean）へ要求電文を送信するサービスリクエスタは，RMI-IIOP を利用して通信します。サービスリクエスタから標準受付（SessionBean）へサービス部品実行の要求電文が送信され，HCSC サーバがサービス部品の実行を行います。

ユーザは開発環境からスタブを取得し，そのスタブを利用して，同期受付（SessionBean）に要求を出します。したがって，サービスリクエスタは，取得したスタブを利用できるように実装する必要があります。

標準の同期受付（SessionBean）に要求を出すサービスリクエスタと HCSC サーバの関係を次の図に示します。

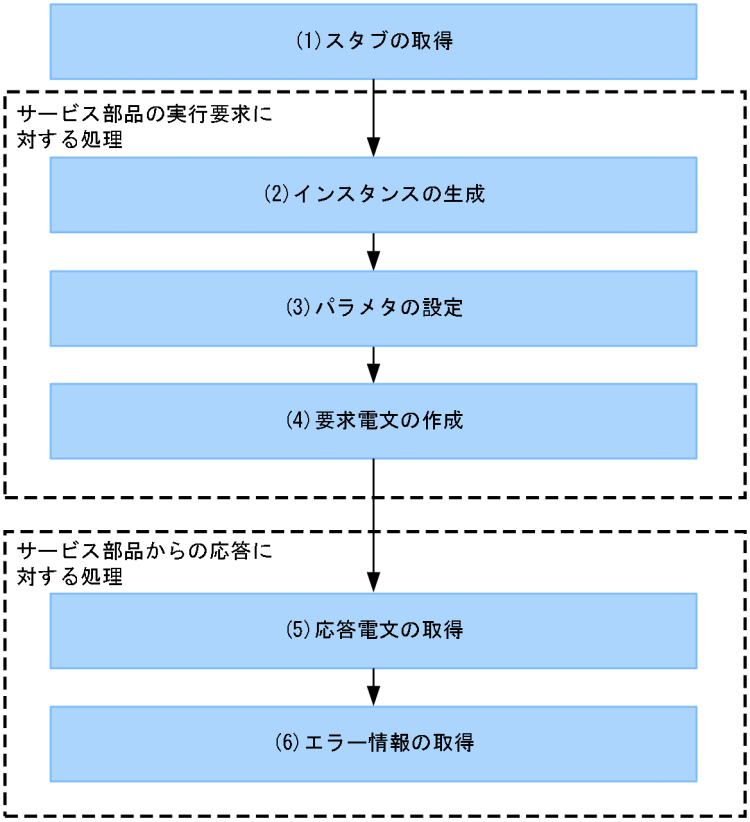
図 9-15　標準の同期受付（SessionBean）に要求を出すサービスリクエスタと HCSC サーバの関係



9.4.1　サービスリクエスタ（標準の同期受付（SessionBean））の作成手順

標準の同期受付（SessionBean）にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-16　サービスリクエスタの作成手順（標準の同期受付（SessionBean））



各工程の作業を次に示します。

(1)　スタブの取得

開発環境の HCSC コンポーネント情報表示画面から，HCSC サーバが配備されている J2EE サーバ（JNDI 名）に対応したスタブを取得します。スタブの取得については，「9.4.2　スタブを取得する」を参照してください。

(2)　インスタンスの生成

同期受付（SessionBean）のメソッドを呼び出すために，「(1)　スタブの取得」で取得したスタブから Enterprise Bean のインスタンスを生成します。インスタンスの生成については，「9.4.3　インスタンスを生成する」を参照してください。

(3)　パラメタの設定

同期受付（SessionBean）のメソッドの引数となるパラメタを設定します。パラメタの設定については，「9.4.4　パラメタを設定する」を参照してください。

(4)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.4.5　要求電文を作成する」を参照してください。

(5)　応答電文の取得

同期受付（SessionBean）から，サービス部品の実行要求に対する応答電文を取得します。応答電文の取得については，「9.4.6　応答電文を取得する」を参照してください。

(6)　エラー情報の取得

要求先のサービス部品，HCSC サーバ，および EJB コンテナでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。エラー情報の取得については，「9.4.7　エラー情報を取得する」を参照してください。

9.4.2　スタブを取得する

HCSC サーバが配備されている J2EE サーバ（JNDI 名）に対応したスタブを取得します。スタブを取得する方法を次に示します。

1. Eclipse のメニューから，［HCSC-Definer］－［公開サービス一覧］を選択します。
   ツリービューの公開サービス一覧に，HCSC コンポーネントの一覧が表示されます。

2. HCSC コンポーネントの一覧から，呼び出すサービス部品（HCSC コンポーネント）を選択し，ダブルクリックします。
   HCSC コンポーネント情報表示画面に，HCSC コンポーネントの情報が表示されます。HCSC コンポーネント情報表示画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.6　HCSC コンポーネント情報表示画面」を参照してください。
   ［サーバ情報］には，サービス部品の要求先となる HCSC サーバの情報が表示されます。

3. ［スタブ取得］ボタンをクリックします。
   ［スタブファイルの保存］ダイアログが表示されます。
   保存先を指定して，スタブを取得します。

取得したスタブの構成を次に示します。

```
/カレントディレクトリ
  └cscmsg_ejb_client.jar
```

9.4.3　インスタンスを生成する

同期受付（SessionBean）のメソッドを呼び出すために，取得したスタブを使用して，Enterprise Bean のインスタンスを生成します。Enterprise Bean のインスタンスを生成する手順を次に示します。

1. EJB ホームオブジェクトのリファレンスの検索に利用する JNDI ネーミングコンテキストを生成します。

(例)

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

2. 生成した JNDI ネーミングコンテキストを使用して，EJB ホームオブジェクトのリファレンスを取得します。

EJB ホームオブジェクトのリファレンスは，ユーザ指定名前空間または EJB コンテナ名前空間を利用して検索し，取得します。EJB ホームオブジェクトのリファレンスの検索・取得の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「3.4.2　Enterprise Bean を呼び出すための実装」を参照してください。

(例)

```
Object objref
     = initial.lookup("HITACHI_EJB/SERVERS/" + "<J2EEサーバ名称>"
       + "/EJB/CSCMsgSyncServiceDelivery/CSCMsgSyncServiceDeliveryEJB");

CSCMsgSyncServiceDeliveryHome home
     = (CSCMsgSyncServiceDeliveryHome)PortableRemoteObject
        .narrow(objref, CSCMsgSyncServiceDeliveryHome.class);
```

3. EJB ホームオブジェクトの create メソッドによって，Enterprise Bean のインスタンスを生成します。生成した Enterprise Bean のインスタンスを使用して，同期受付（SessionBean）のメソッド(Enterprise bean のメソッド）を呼び出します。

(例) 要求電文が XML の場合

```
CSCMsgSyncServiceDelivery reception = home.create();
      // Enterprise Beanのインスタンスを生成
String result = reception .invokeXML(    // メソッドの呼び出し
               serviceName,              // サービス名
               clientID,                 // クライアント相関ID
               requestFormatID,          // リクエストフォーマットID
               responseFormatID,         // レスポンスフォーマットID
               operationName,            // オペレーション名
               userData);                // ユーザ電文
```

(例) 要求電文がバイナリの場合

```
CSCMsgSyncServiceDelivery reception = home.create();
      // Enterprise Beanのインスタンスを生成
byte[] resultBinary = reception .invokeBinary(  // メソッドの呼び出し
               serviceName,                 // サービス名
               clientID,                    // クライアント相関ID
               requestFormatID,             // リクエストフォーマットID
               responseFormatID,            // レスポンスフォーマットID
               operationName,               // オペレーション名
               userDataBinary.length,       // メッセージ長
               userDataBinary);             // ユーザ電文
```

注意事項

バイナリ電文の要求電文を送信できるのは，サービス部品側で使用している電文形式がバイナリ形式のときだけです。

9.4.4　パラメタを設定する

同期受付（SessionBean）のメソッドを呼び出すため，メソッドの引数となるパラメタを設定します。パラメタの詳細を次の表に示します。

表 9–12　パラメタの詳細（標準の同期受付（SessionBean））

パラメタ名	データ型	パラメタ		説明
		invokeXML	invokeBinary	
サービス名	java.lang.String	serviceName		要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，開発環境で定義したサービスアダプタまたはビジネスプロセスを指定してください。
クライアント相関 ID	java.lang.String	clientID		サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 サービスリクエスタからの要求電文と，HCSC サーバで管理している実行履歴やログおよびトレースを対応づけるために使用します。したがって，HCSC サーバに送信した要求電文ごとに異なる ID を指定してください。 クライアント相関 ID を省略したい場合，NULL を指定してください。
リクエストフォーマット ID	java.lang.String	cscRequestFormatID		サービスリクエスタからの要求電文フォーマットを一意に識別するための ID です。 このパラメタには NULL を指定してください。
レスポンスフォーマット ID	java.lang.String	cscResponseFormatID		HCSC サーバからの応答電文フォーマットを一意に識別するための ID です。 このパラメタには NULL を指定してください。
オペレーション名	java.lang.String	serviceOperationName		要求先のサービス名に対するオペレーション名です。※ オペレーション名は，開発環境で定義したサービス部品のオペレーション名を指定します。255 バイト以内の XMLSchema の NCName 定義文字で指定してください。

パラメタ名	データ型	パラメタ		説明
		invokeXML	invokeBinary	
オペレーション名	java.lang.String	serviceOperationName		要求先のサービス部品が同期のサービス部品（Web サービスまたは SessionBean）またはビジネスプロセスの場合，必ず指定してください。 要求先のサービス部品が非同期のサービス部品の場合，オペレーション名を省略できます。省略する場合，NULL を指定してください。
ユーザ電文	java.lang.String	msg	×	サービスリクエスタからの要求電文です。※ 要求電文が XML の場合に指定します。要求電文がない場合，NULL または空文字（""）を指定してください。要求電文の詳細については，「9.4.5　要求電文を作成する」を参照してください。
ユーザ電文長	int	×	requestMessageLength	要求電文の長さです。 要求電文がバイナリの場合に指定します。要求電文がバイナリの場合は，必ず指定してください。 要求電文がない場合，0 を指定してください。
ユーザ電文	byte[]	×	msg	サービスリクエスタからの要求電文です。※ 要求電文がバイナリの場合に指定します。要求電文の詳細については，「9.4.5　要求電文を作成する」を参照してください。 要求電文がない場合，NULL または 0 バイトのバイト配列を指定してください。

（凡例）

×：指定できません。

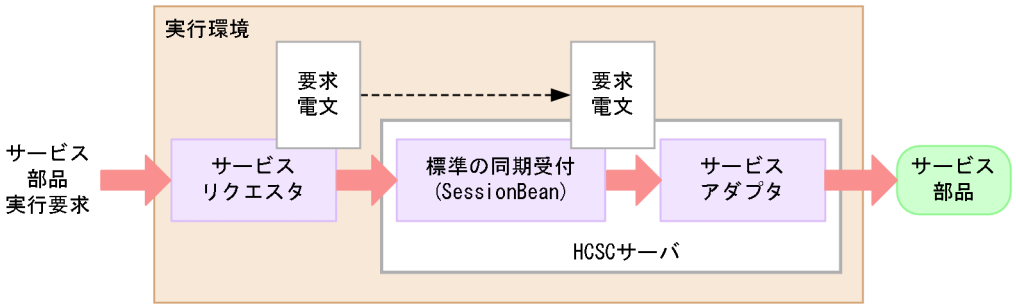
注※

要求先のサービス部品のプロトコルが SOAP の場合，ユーザ電文のルート要素の名称（データ変換を行う場合，データ変換後の電文のルート要素の名称）から呼び出すオペレーションを決定します。そのため，ユーザ電文のルート要素に誤った名称を指定した場合，意図しないオペレーションを呼び出してしまうので注意してください。

9.4.5　要求電文を作成する

サービスリクエスタから HCSC サーバの同期受付（SessionBean）に，サービス部品を要求する要求電文を作成します。サービスリクエスタから送信する要求電文の内容は，サービス部品側と同じ電文フォーマットで作成してください。要求電文の送信を次の図に示します。

図 9-17　要求電文の送信（標準の同期受付（SessionBean））



また，サービスリクエスタから同期受付（SessionBean）に送信する要求電文は，電文種別に XML 電文とバイナリ電文の 2 種類があります。どちらの電文種別を使用するかは，サービス部品側で使用しているプロトコルによって異なります。

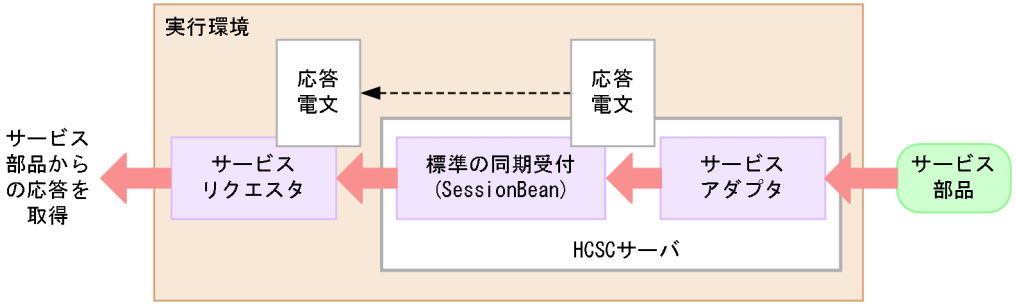
XML 電文およびバイナリ電文の要求電文については，「9.2.6　要求電文を作成する」を参照してください。

正常な要求電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.3 SessionBean を使用したサービスとの接続」を参照してください。

9.4.6　応答電文を取得する

HCSC サーバの同期受付（SessionBean）から，サービス部品の実行要求に対する応答電文を取得します。サービスリクエスタは，サービス部品側と同じ電文フォーマットの応答電文を取得します。応答電文の取得を次の図に示します。

図 9-18　応答電文の取得（標準の同期受付（SessionBean））



また，サービスリクエスタは，電文種別が XML 電文の応答電文を取得します。サービス部品からの応答電文がない場合は，NULL を受信します。

XML 電文の応答電文および NULL の受信については，「9.2.7　応答電文を取得する」を参照してください。

9.4.7　エラー情報を取得する

要求先のサービス部品，HCSC サーバ，および EJB コンテナでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。

エラー情報は，サービスリクエスタ側で CSCMsgServerException クラスを取得することによって取得できます。CSCMsgServerException クラスについては「(2)　CSCMsgServerException クラス」を参照してください。エラーの伝わり方については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.7.2　SessionBean 実行時の障害対策」を参照してください。

エラーの詳細については，getErrorMessage メソッドおよび getErrorCode メソッドを使用して取得します。また，Web サービスからのエラー情報（SOAP Fault）を取得する場合は，checkSoapFault メソッドを使用します。

(1)　サービスリクエスタ側の実装例

CSCMsgServerException クラスを取得する場合のサービスリクエスタ側の実装例を次に示します。

```
          :
} catch (CSCMsgServerException e) {
    System.err.println("Exception ErrorMessage = "
                + e.getErrorMessage() );
    System.err.println("Exception ErrorCode = "
                + e.getErrorCode() );
    if (e.checkFaultMessage() == true) {
        System.err.println("Exception ProcessInstanceID = "
                + e.getProcessInstanceID() );
        System.err.println("Exception FaultCode = "
                + e.getFaultCode() );
        System.err.println("Exception FaultString = "
                + e.getFaultString() );
        System.err.println("Exception FaultActor = "
                + e.getFaultActor() );
        System.err.println("Exception FaultDetails = "
                + new String(e.getFaultDetail(), "UTF-8"));
    System.err.println("Exception FaultName = "
                + e.getFaultName() );
    }
}
          :
```

(2)　CSCMsgServerException クラス

サービスリクエスタが取得する例外クラスです。

(a)　クラス定義

パッケージ

```
jp.co.Hitachi.soft.csc.msg.message.reception
```

クラス

```
public class CSCMsgServerException
extends java.lang.Exception
```

(b) フィールドの一覧

フィールド名	データ型	説明
errorMessage	java.lang.String	次に示す例外の内容です。 • HCSC-Messaging 内で検知した例外 • サービス部品またはビジネスプロセスからのフォルト
errorCode	java.lang.String	次に示す例外の内容に対応するエラーコードです。 • HCSC-Messaging 内で検知した例外 • サービス部品またはビジネスプロセスからのフォルト
processInstanceID	java.lang.String	ビジネスプロセスのインスタンス ID の情報です。 ビジネスプロセスでエラーが発生した場合に値が設定されます。
cscmsgFaultCode	java.lang.String	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報です。
cscmsgFaultString	java.lang.String	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報です。
cscmsgFaultActor	java.lang.String	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報です。
cscmsgFaultDetail	byte[]	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。
faultName	java.lang.String	サービス部品（Web サービスまたは SessionBean），またはビジネスプロセスからのフォルト名（例外名）情報です。 次の場合に値が設定されます。 • サービス部品（Web サービスまたは SessionBean）からのユーザ定義例外の SOAP Fault の場合 • ビジネスプロセスからのフォルトの場合 SOAP Fault 動作定義ファイルに targetNamespace の URI を定義した Web サービスからの SOAP Fault の場合，値は設定されません。SOAP Fault 動作定義ファイルについては，マニュアル「サービスプラットフォーム リファレンス」の「6.6.1　SOAP Fault 動作定義ファイル」を参照してください。

(c) メソッドの一覧

メソッド名	データ型	説明
getErrorMessage	java.lang.String	エラーメッセージを取得します。
getErrorCode	java.lang.String	エラーコードを取得します。
checkFaultMessage	boolean	サービス部品，ビジネスプロセス，またはサービスアダプタからのフォルト情報があるかどうかをチェックします。
getFaultCode	java.lang.String	FaultCode 情報を取得します。
getFaultString	java.lang.String	FaultString 情報を取得します。

メソッド名	データ型	説明
getFaultActor	java.lang.String	FaultActor 情報を取得します。
getFaultDetail	byte[]	Detail 情報を取得します。
getProcessInstanceID	java.lang.String	ビジネスプロセスのインスタンス ID を取得します。
getFaultName	java.lang.String	フォルト名（例外名）を取得します。

(d) メソッドの詳細

●getErrorMessage

説明

エラーメッセージを取得します。

次の例外の内容を取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品またはビジネスプロセスからのフォルト

形式

public java.lang.String getErrorMessage()

パラメタ

なし

戻り値

エラーメッセージ

例外

なし

●getErrorCode

説明

エラーコードを取得します。

次の例外の内容に対するエラーコードを取得する場合に使用します。

- HCSC-Messaging 内で検知した例外
- サービス部品またはビジネスプロセスからのフォルト

形式

public java.lang.String getErrorCode()

パラメタ

なし

戻り値

エラーコード

例外

なし

●checkFaultMessage

説明

サービス部品，ビジネスプロセス，またはサービスアダプタからのフォルト情報があるかどうかをチェックします。

サービス部品またはビジネスプロセスが Web サービスの場合，SOAP Fault として返されたエラーの内容が含まれているかどうかチェックします。

また，サービスアダプタからの個別のエラーが含まれているかどうかをチェックします。

形式

public boolean checkFaultMessage()

パラメタ

なし

戻り値

true：Fault メッセージ情報が存在します。

false：Fault メッセージ情報が存在しません。

例外

なし

●getFaultCode

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報を取得します。

形式

public java.lang.String getFaultCode()

パラメタ

なし

戻り値

faultcode

例外

なし

●getFaultString

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報を取得します。

形式

public java.lang.String getFaultString()

パラメタ

なし

戻り値

faultstring

例外

なし

●getFaultActor

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報を取得します。

形式

public java.lang.String getFaultActor()

パラメタ

なし

戻り値

faultactor

例外

なし

●getFaultDetail

説明

サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報を取得します。

サービス部品が設定した Detail 情報をバイト配列でサービスリクエスタに引き渡します。したがって，取得したバイト配列を文字列に変換する必要があります。

形式

public byte[] getFaultDetail()

パラメタ

なし

戻り値

detail

例外

なし

●getProcessInstanceID

説明

ビジネスプロセスのインスタンス ID を取得します。

形式

public String getProcessInstanceID()

パラメタ

なし

戻り値

String

例外

なし

●getFaultName

説明

フォルト名（例外名）を取得します。

形式

public String getFaultName()

パラメタ

なし

戻り値

String

例外

なし

(e) サブクラスの一覧

クラス名	サブクラス名と説明	
CSCMsgServerException	CSCMsgServiceException サービス部品例外	CSCMsgServiceUserException サービス部品からのフォルト情報または Exception
		CSCMsgBusinessProcessUserException ビジネスプロセスからのフォルト情報または Exception
		CSCMsgServiceExecuteException サービス部品（Web サービス）からユーザ定義例外以外の例外が返ってきた場合の例外

クラス名	サブクラス名と説明	
CSCMsgServerException	CSCMsgServiceDeliveryException サービス部品要求振り分けエラー	CSCMsgLocationSearchException ロケーション検索失敗
		CSCMsgRoutingExecException ルーティング失敗
		CSCMsgDelivererExecException メッセージ配送失敗
		CSCMsgDBQServiceDeliveryException 標準の非同期受付（MDB（DB キュー））から DB キューのサービス部品を呼んだ場合の例外
	CSCMsgServerParameterException 入力パラメタエラー	－
	CSCMsgServerInternalException 上記以外のエラー	－

（凡例）

－：サブクラスはありません。

9.4.8　ビジネスプロセスの再実行を要求するサービスリクエスタの作成（SessionBean）

標準の同期受付（SessionBean）に要求を出すサービスリクエスタとして，ビジネスプロセスの再実行を要求するサービスリクエスタを作成できます。

標準の同期受付（SessionBean）にビジネスプロセスの再実行を要求するサービスリクエスタの作成手順は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。作成手順については，「9.4.1　サービスリクエスタ（標準の同期受付（SessionBean））の作成手順」を参照してください。

手順ごとに実施する内容を次に示します。

(1)　スタブの取得

スタブの取得方法は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

スタブの取得については，「9.4.2　スタブを取得する」を参照してください。

(2) インスタンスの生成

インスタンスの生成方法は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

インスタンスの生成については，「9.4.3　インスタンスを生成する」を参照してください。

なお，ビジネスプロセスの再実行を要求するメソッドは，invokeBPXML()を使用して呼び出します。

ビジネスプロセスの再実行を要求するメソッドの例を次に示します。

（例）ビジネスプロセスの再実行を要求する場合

```
String result = ws.invokeBPXML(              // メソッドの呼び出し
                        serviceName,         // サービス名
                        bpRequestType,       // ビジネスプロセス用要求タイプ
                        bpProcessId,         // ビジネスプロセス用プロセスID
                        clientID,            // クライアント相関ID
                        requestFormatID,     // リクエストフォーマットID
                        responseFormatID,    // レスポンスフォーマットID
                        operationName,       // オペレーション名
                        userData);           // ユーザ電文
```

(3) パラメタの設定

メソッドの引数となるパラメタは，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と異なります。同期受付（SessionBean）にビジネスプロセスの再実行を要求するサービスリクエスタの場合に設定するパラメタの詳細を次に示します。

表 9-13　パラメタの詳細（標準の同期受付（SessionBean）／ビジネスプロセス再実行要求）

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
サービス名	java.lang.String	serviceName	要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，開発環境で定義したビジネスプロセスを指定してください。
ビジネスプロセス用要求タイプ	java.lang.String	cscBpRequestType	要求電文の種類を示します。 ビジネスプロセスの再実行を要求する場合，文字列「RECOVER」を指定してください。※
ビジネスプロセス用プロセス ID	java.lang.String	cscBpProcessID	ビジネスプロセスインスタンス ID です。 エラー情報から取得した値，または，メッセージログに出力された値を指定してください。
クライアント相関 ID	java.lang.String	clientID	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 このパラメタには NULL を指定してください。※

パラメタ名	データ型	パラメタ（invokeBPXML）	説明
リクエストフォーマット ID	java.lang.String	cscRequestFormatID	サービスリクエスタからの要求電文フォーマットを一意に識別するための ID です。 このパラメタには NULL または空文字（""）を指定してください。※
レスポンスフォーマット ID	java.lang.String	cscResponseFormatID	HCSC サーバからの応答電文フォーマットを一意に識別するための ID です。 このパラメタには NULL または空文字（""）を指定してください。※
オペレーション名	java.lang.String	serviceOperationName	要求先のサービス名に対するオペレーション名です。 このパラメタには NULL または空文字（""）を指定してください。※
ユーザ電文	java.lang.String	msg	サービスリクエスタからの要求電文です。 このパラメタには NULL または空文字（""）を指定してください。※

注※

これらのパラメタは，指定する値として固定値（「RECOVER」，NULL または空文字（""））が決められています。決められた値以外の値を指定すると，正常に動作しません。

(4)　要求電文の作成

要求電文の作成方法は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

要求電文の作成については，「9.4.5　要求電文を作成する」を参照してください。

(5)　応答電文の取得

応答電文の作成方法は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

応答電文の取得については，「9.4.6　応答電文を取得する」を参照してください。

(6)　エラー情報の取得

エラー情報を取得するためのサービスリクエスタの実装は，標準の同期受付（SessionBean）に要求を出す，通常のサービスリクエスタを作成する場合と同じです。

エラー情報の取得については，「9.4.7　エラー情報を取得する」を参照してください。

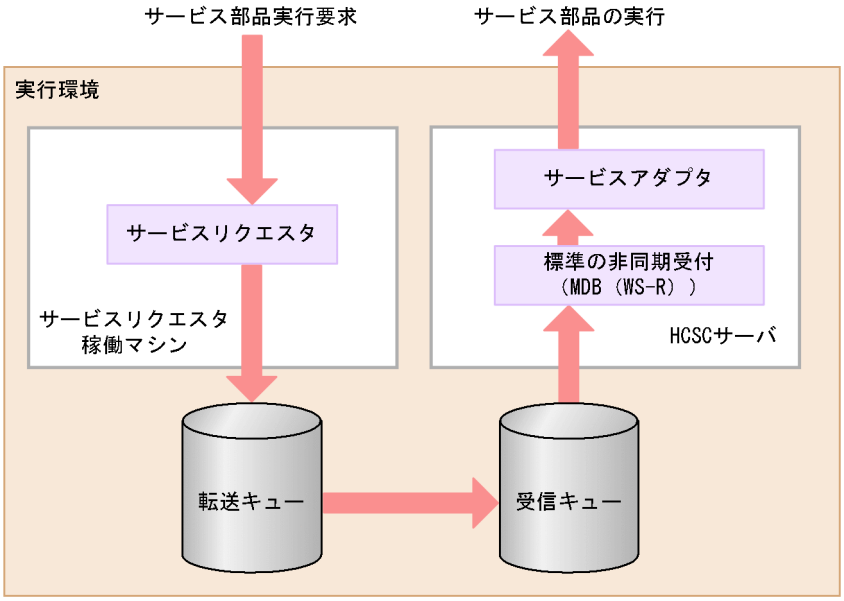
9.5　標準の非同期受付（MDB（WS-R））に要求を出すサービスリクエスタ

標準の非同期受付（MDB（WS-R））へ要求電文を送信するサービスリクエスタは，転送キューおよび受信キューを通して通信を行います。

サービスリクエスタから転送キューに送信された要求電文は，受信キューを通して非同期受付（MDB（WS-R））に送信されます。

標準の非同期受付（MDB（WS-R））に要求を出すサービスリクエスタと HCSC サーバの関係を次の図に示します。

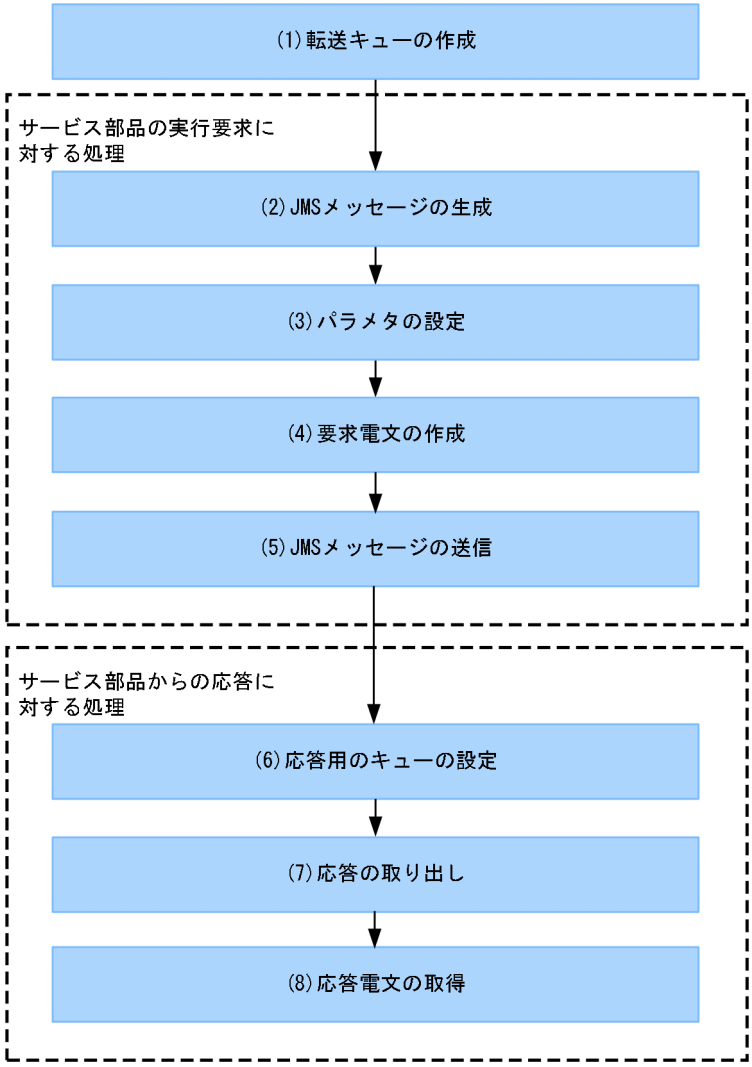
図 9-19　標準の非同期受付（MDB（WS-R））に要求を出すサービスリクエスタと HCSC サーバの関係



9.5.1　サービスリクエスタ（標準の非同期受付（MDB（WS-R））の作成手順

標準の非同期受付（MDB（WS-R））にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-20　サービスリクエスタの作成手順（標準の非同期受付（MDB（WS-R））



各工程の作業を次に示します。

(1)　転送キューの作成

サービスリクエスタから要求電文を送信するために必要な転送キューを作成します。転送キューの作成については，「9.5.2　転送キューを作成する」を参照してください。

(2)　JMS メッセージの生成

非同期受付（MDB（WS-R））にサービス部品の実行を要求するために，QueueSender のオブジェクトおよび JMS メッセージを生成します。QueueSender のオブジェクトおよび JMS メッセージの生成については，「9.5.3　JMS メッセージを生成する」を参照してください。

(3)　パラメタの設定

「(2)　JMS メッセージの生成」で生成した JMS メッセージにパラメタを設定します。パラメタの設定については，「9.5.4　パラメタを設定する」を参照してください。

(4)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.5.5　要求電文を作成する」を参照してください。

(5)　JMS メッセージの送信

JMS メッセージを転送キューに送信します。JMS メッセージの送信については，「9.5.6　JMS メッセージを送信する」を参照してください。

(6)　応答用のキューの設定

同期のサービス部品から応答を受け取るために，応答用のキュー（転送キュー）を設定します。応答用のキューの設定については，「9.5.7　応答用のキューを設定する」を参照してください。

(7)　応答の取り出し

応答用のキュー（転送キュー）から受信キューに送信された応答を取り出します。応答の取り出しについては，「9.5.8　応答を取り出す」を参照してください。

(8)　応答電文の取得

取り出した応答から応答電文を取得します。応答電文の取得については，「9.5.9　応答電文を取得する」を参照してください。

9.5.2　転送キューを作成する

サービスリクエスタから要求電文を送信するために必要な転送キューを作成します。転送キューは，サービスリクエスタからの要求電文を，HCSC サーバが提供する受信キューに送信します。

転送キューの作成には，Reliable Messaging の hrmmkaddr コマンドおよび hrmmkque コマンドを使用します。

ここでは，hrmmkaddr コマンドおよび hrmmkque コマンドのオプションの指定値について説明します。ここで説明していない hrmmkaddr コマンドのオプションについては，マニュアル「Reliable Messaging」の「8.3.16　hrmmkaddr（あて先登録）」を参照してください。また，hrmmkque コマンドのオプションについては，マニュアル「Reliable Messaging」の「8.3.17　hrmmkque（ローカルキューの作成）」を参照してください。

(1)　hrmmkaddr コマンドのオプションの指定値

hrmmkaddr コマンドのオプションの指定値を次に示します。

-u 宛先アドレス

HCSC サーバの宛先アドレス（転送先システムのキュー間転送用 Web アプリケーションの URL）を指定します。

```
http://<ホスト名>:<ポート番号>※1/<コンテキストルート>※2/services/HRMReceiver/
https://<ホスト名>:<ポート番号>※1/<コンテキストルート>※2/services/HRMReceiver/
```

注※1

HCSC サーバの URL です。

注※2

受信側 Reliable Messaging のキュー間転送用 Web アプリケーションのコンテキストルートです。デフォルトは「uCosminexusRM」です。

(2) hrmmkque コマンドのオプションの指定値

-t キューの種類

transmit（転送キュー）を指定します。

-m キューの永続性

persistent（永続キュー属性）を指定します。

-v 転送先キュー名

次のどちらかを指定します。

CSC<HCSC サーバ名>ACPT_RCVQ

HCSC サーバがロードバランスクラスタ構成の場合に指定します。

CSC<HCSC クラスタ名>ACPT_RCVQ

HCSC サーバが HA クラスタ構成の場合に指定します。

HCSC クラスタ名は，開発環境および運用環境で HCSC サーバ構成配備時に決定される HCSC サーバのクラスタ名です。

-i キューの転送モード

compatible（互換モード）を指定します。

9.5.3 JMS メッセージを生成する

標準の非同期受付（MDB（WS-R））にサービス部品の実行を要求するためには，QueueSender のオブジェクトおよび JMS メッセージを生成し，生成した JMS メッセージを転送キューに送信します。JMS メッセージを送信するアプリケーションの開発については，マニュアル「Reliable Messaging」を参照してください。

JMS メッセージを生成する手順を次に示します。

1. QueueConnectionFactory と Queue の開始処理（lookup）を定義します。

(例)

```
Context ic = new InitialContext();
          :
QueueConnectionFactory qcFactory
      = (QueueConnectionFactory) ic.lookup(”java:comp/env/jms/qcf”);
          :
Queue queue = (Queue) ic.lookup(”java:comp/env/jms/queue”);
          :
```

2. QueueSession を生成します。

(例)

```
QueueSession qSession
    = qConnection.createQueueSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. QueueSession から QueueSender を作成します。

(例)

```
qSender = qSession.createSender(queue);
```

4. QueueSession から JMS メッセージを生成します。

(例) 要求電文が XML の場合

```
TextMessage textMessage = qSession.createTextMessage();
```

(例) 要求電文がバイナリの場合

```
BytesMessage bytesMessage = qSession.createBytesMessage();
```

9.5.4 パラメタを設定する

非同期受付（MDB（WS-R））のメソッドを呼び出すため，JMS メッセージにパラメタを設定します。

(1) プロパティの設定

JMS メッセージのプロパティを設定します。設定するプロパティの詳細については，「(4) パラメタの詳細」を参照してください。

(例) 要求電文が XML の場合

```
textMessage = qSession.createTextMessage();
textMessage.setStringProperty(”CSCServiceName”, serviceName);
     // サービス部品名
textMessage.setStringProperty(”CSCCorrelationID”, clientID);
     // クライアント相関ID
textMessage.setStringProperty(”CSCRequestFormatID”, requestFormatID);
     // リクエストフォーマットID
```

```
textMessage.setStringProperty("CSCResponseFormatID", responseFormatID);
     // レスポンスフォーマットID
textMessage.setStringProperty("CSCServiceOperationName", operationName);
     // オペレーション名
textMessage.setStringProperty("CSCReplyToQueueName", replyToQueueName);
     // 応答用のキュー名
textMessage.setStringProperty("CSCMessageType", "XML");
     // 電文種別
```

注意事項

バイナリ電文の要求電文を送信できるのは，サービス部品側で使用している電文形式がバイナリ形式のときだけです。DB キューのサービス部品に送信する要求電文はバイナリ変換します。バイナリ変換する場合は，サービスリクエスタ側とサービス部品側で同じエンコードの方式を使用してください。

(2) 引き継がれるヘッダおよびプロパティ

次のヘッダおよびプロパティは，要求時の内容がサービス部品側のキューまで引き継がれます。したがって，必要に応じて次のヘッダおよびプロパティを設定してください。

- JMSReplyTo
- JMSCorrelationID※1
- JMSType
- JMSXGroupID
- JMSXGroupSeq
- ユーザ固有のプロパティ※2

注※1

指定できる範囲は，255 バイトまでです。

注※2

ただし，次のプロパティはサービス部品側のキューまで引き継がれません。

- JMSX で始まる JMS 定義によるプロパティ名
- JMS_で始まるプロバイダ指定のプロパティ名
- CSC で始まるプロパティ名
- HCSC で始まるプロパティ名

(3) ペイロードの設定

JMS メッセージのペイロードに要求電文（ユーザ電文）を設定します。ペイロードの詳細については，「(4) パラメタの詳細」を参照してください。

（例）要求電文が XML の場合

```
textMessage.setText( userData );
```

(4) パラメタの詳細

パラメタの詳細を次の表に示します。

表 9-14 パラメタの詳細（標準の非同期受付（MDB（WS-R））

パラメタ名	データ型	パラメタ		プロパティ/ペイロード	説明
		TextMessage	BytesMessage		
サービス名	String	StringProperty ("CSCServiceName")		プロパティ	要求先のサービス名です。 このパラメタは必ず指定してください。 要求先のサービス名は，開発環境で定義したサービスアダプタまたはビジネスプロセスを指定してください。
クライアント相関 ID	String	StringProperty ("CSCCorrelationID")		プロパティ	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 サービスリクエスタからの要求電文と，HCSC サーバで管理している実行履歴やログおよびトレースを対応づけるために使用します。したがって，HCSC サーバに送信した要求電文ごとに異なる ID を指定してください。 なお，このパラメタを指定しないことで，クライアント相関 ID を省略できます。
リクエストフォーマット ID	String	StringProperty ("CSCRequestFormatID")		プロパティ	サービスリクエスタからの要求電文フォーマットを一意に識別するための ID です。 1,024 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 このパラメタには NULL を指定してください。なお，このパラメタを指定しないことで，NULL が設定されます。
レスポンスフォーマット ID	String	StringProperty ("CSCResponseFormatID")		プロパティ	HCSC サーバからの応答電文フォーマットを一意に識別するための ID です。 1,024 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。

パラメタ名	データ型	パラメタ		プロパティ/ペイロード	説明
		TextMessage	BytesMessage		
レスポンスフォーマットID	String	StringProperty ("CSCResponseFormatID")		プロパティ	このパラメタにはNULLを指定してください。なお，このパラメタを指定しないことで，NULLが設定されます。
オペレーション名	String	StringProperty ("CSCServiceOperationName")		プロパティ	要求先のサービス名に対するオペレーション名です。※ オペレーション名は，開発環境で定義したサービス部品のオペレーション名を指定します。255バイト以内のXMLSchemaのNCName定義文字で指定してください。 要求先のサービス部品が同期のサービス部品（Webサービスまたは SessionBean）またはビジネスプロセスの場合，必ず指定してください。 要求先のサービス部品が非同期のサービス部品の場合，オペレーション名を省略できます。このパラメタを指定しないことで，オペレーション名を省略できます。
応答用のキュー名	String	StringProperty ("CSCReplyToQueueName")		プロパティ	要求先のサービス部品が同期（Webサービスまたは SessionBean）またはビジネスプロセスの場合に，サービス部品またはビジネスプロセスからの応答を受け取るキュー名です。 要求先のサービス部品が同期か非同期かわからない場合にこの指定をすると，応答を受け取ることができます。 応答用のキューの詳細については，「9.5.7　応答用のキューを設定する」を参照してください。 なお，このパラメタを指定しないことで，応答用のキュー名を省略できます。省略した場合，NULLが設定され，応答を受け取ることができません。
電文種別	String	StringProperty ("CSCMessageType")		プロパティ	次に示す要求電文の種別を指定してください。 • XML電文（ペイロードがTextMessage）の場合：XML • バイナリ電文（ペイロードがBytesMessage）の場合：Binary 要求電文がある場合，このパラメタは必ず指定してください。 要求電文がない場合，このパラメタを指定しないことで，電文種別を省略することができます。

パラメタ名	データ型	パラメタ		プロパティ/ペイロード	説明
		TextMessage	BytesMessage		
ユーザ電文	−	TextMessage	×	ペイロード	サービスリクエスタからの要求電文です。※ 要求電文が XML の場合に指定します。要求電文がない場合，指定する必要はありません。要求電文の詳細については，「9.5.5　要求電文を作成する」を参照してください。
ユーザ電文長	long	×	LongProperty ("CSCMessageLength")	プロパティ	要求電文の長さです。 要求電文がバイナリの場合に指定します。
ユーザ電文	−	×	BytesMessage	ペイロード	サービスリクエスタからの要求電文です。※ 要求電文がバイナリの場合に指定します。要求電文がない場合，指定する必要はありません。要求電文の詳細については，「9.5.5　要求電文を作成する」を参照してください。

（凡例）

−：該当しません。

×：指定できません。

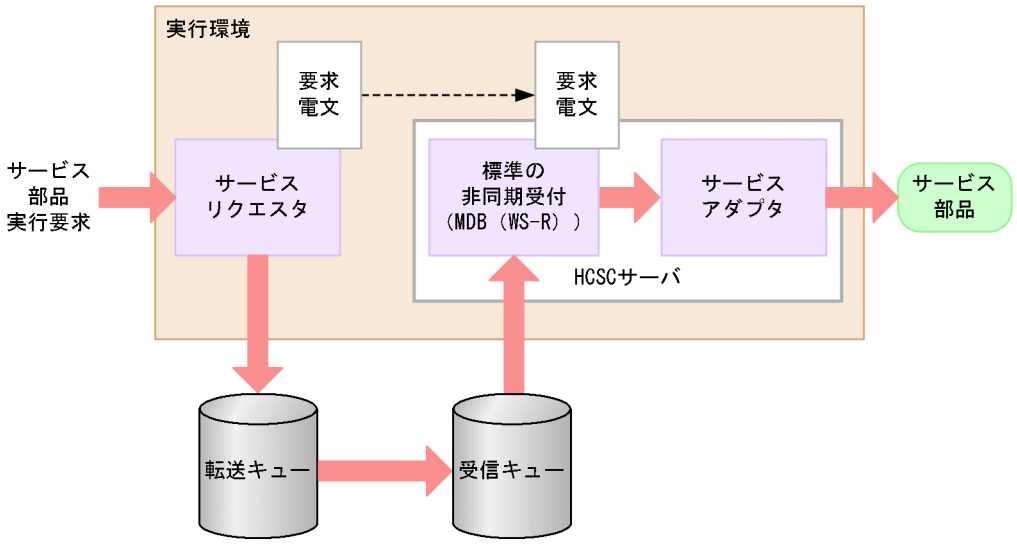
注※

要求先のサービス部品のプロトコルが SOAP の場合，ユーザ電文のルート要素の名称（データ変換を行う場合，データ変換後の電文のルート要素の名称）から呼び出すオペレーションを決定します。そのため，ユーザ電文のルート要素に誤った名称を指定した場合，意図しないオペレーションを呼び出してしまうので注意してください。

9.5.5　要求電文を作成する

サービスリクエスタから HCSC サーバの非同期受付（MDB（WS-R））に，サービス部品を要求する要求電文を作成します。サービスリクエスタから送信する要求電文の内容は，サービス部品側と同じ電文フォーマットで作成してください。要求電文の送信を次の図に示します。

図 9-21　要求電文の送信（標準の非同期受付（MDB（WS-R））



また，サービスリクエスタから非同期受付（MDB（WS-R））に送信する要求電文は，電文種別に XML 電文とバイナリ電文の 2 種類があります。どちらの電文種別を使用するかは，サービス部品側で使用しているプロトコルによって異なります。

XML 電文およびバイナリ電文の要求電文については，「9.2.6　要求電文を作成する」を参照してください。

正常な要求電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.4　WS-R（WS-Reliability）を使用したサービスとの接続」を参照してください。

9.5.6　JMS メッセージを送信する

パラメタを設定した JMS メッセージを転送キューに送信します。

JMS メッセージを送信する手順を次に示します。

1. ローカルトランザクションの開始および JMS メッセージの登録を行います。

(例)

```
qSender.send(message);
```

2. 終了処理を定義します。

(例)

```
qSession.commit();        // ローカルトランザクションのコミット
qSender.close();          // QueueSenderの解放
qSession.close();         // QueueSessionの解放
qConnection.close();      // QueueConnectionの解放
```

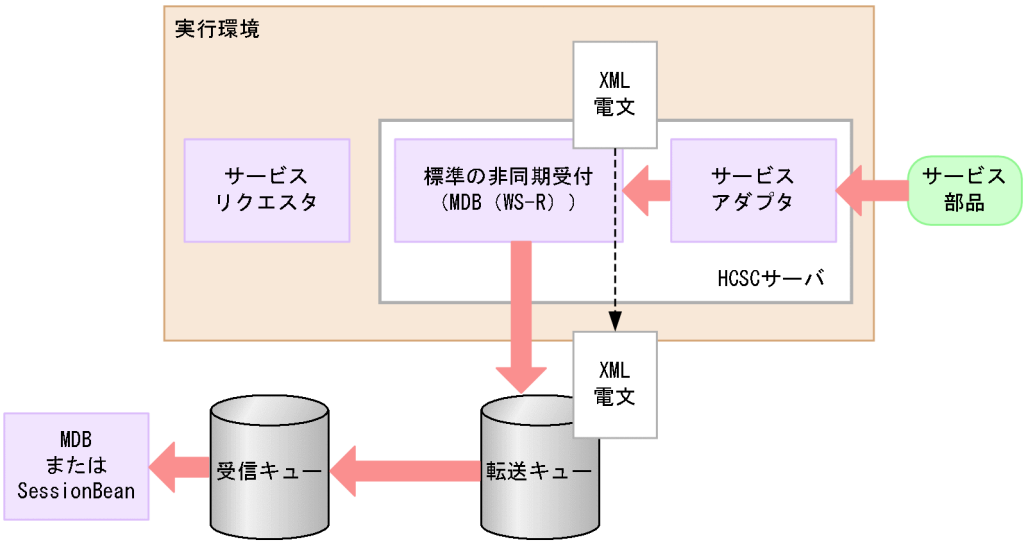
9.5.7　応答用のキューを設定する

標準の非同期受付（MDB（WS-R））は，非同期の要求電文の標準受付であり，基本的にはサービス部品からの応答がない場合に使用します。しかし，非同期受付（MDB（WS-R））から同期のサービス部品（Web サービスまたは SessionBean）を要求した場合，同期のサービス部品から応答が返るときがあります。このような場合，要求時に応答用のキュー（転送キュー）を指定することで，サービスリクエスタは同期のサービス部品から応答を受け取ることができます。要求時の応答用のキューの指定については，「9.5.4　パラメタを設定する」を参照してください。

標準の非同期受付（MDB（WS-R））は，同期サービス部品からの応答である XML 電文（ペイロードが TextMessage）を転送キューに送信します。転送キューで受け取った応答は，受信キューに送信します。受信キューに送信した応答を取り出すことで，サービス部品からの応答電文を取得できます。応答の取り出しについては，「9.5.8　応答を取り出す」を参照してください。

標準の非同期受付（MDB（WS-R））、転送キュー，および受信キューの関係を次に示します。

図 9-22　標準の非同期受付（MDB（WS-R））、転送キュー，および受信キューの関係



転送キュー（応答用のキュー）

応答用のキューとなる転送キューを，HCSC サーバが稼働している J2EE サーバ（Reliable Messaging）内に作成します。

転送キューの転送先（宛先アドレス）には，受信キューを指定します。転送先以外の転送キューの設定は，ユーザの運用に合わせ，必要に応じて行ってください。

転送キューの作成方法については，「9.5.2　転送キューを作成する」，およびマニュアル「Reliable Messaging」の「8.3.20　hrmmkque（転送キューの作成）」を参照してください。

受信キュー（ローカルキュー）

サービスリクエスタが稼働している J2EE サーバ（Reliable Messaging）内に，受信キュー（ローカルキュー）を作成します。

受信キューの設定は，ユーザの運用に合わせ，必要に応じて行ってください。

受信キューの作成方法については，マニュアル「Reliable Messaging」の「8.3.17　hrmmkque（ローカルキューの作成)」を参照してください。

9.5.8　応答を取り出す

受信キューから応答を取り出すには，MDB で実装する方法と SessionBean で実装する方法があります。正常な応答電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.4.3　同期のサービス部品を呼び出す場合の応答電文の扱い」を参照してください。

(1)　応答の内容

サービス部品からは次の内容が応答として送信されます。

- 電文種別（StringProperty プロパティ（"CSCMessageType"））
  応答電文の電文種別は XML 電文のため，StringProperty プロパティには「XML」が設定されています。
- サービス部品からの応答電文（ペイロード（TextMessage））
  XML 電文（TextMessage）の応答電文が設定されています。
  同期のサービス部品を実行したがその応答電文がない場合は，ペイロードなしの TextMessage が送信されます。

(2)　引き継がれるヘッダおよびプロパティ

次のヘッダおよびプロパティは，サービスリクエスタからサービス部品を要求したときに設定した内容が，応答用のキューを通し，受信キューまで引き継がれます。

(a)　HCSC サーバ関連のプロパティ

- StringProperty（"CSCServiceName"）
  サービスリクエスタが指定した，要求先のサービス名です。
- StringProperty（"CSCCorrelationID"）
  サービスリクエスタが指定した，サービスリクエスタからの要求電文を一意に識別するための相関識別子です。
- StringProperty（"CSCResponseFormatID"）
  NULL が設定されています。
- StringProperty（"CSCServiceOperationName"）
  サービスリクエスタが指定した，要求先のサービス名に対するオペレーション名です。

(b)　Reliable Messaging 関連のヘッダ，およびプロパティ

- JMSReplyTo
- JMSCorrelationID

- JMSType
- JMSXGroupID
- JMSXGroupSeq
- ユーザ固有のプロパティ名称

9.5.9　応答電文を取得する

取り出した応答から応答電文を取得します。取得する応答電文は，次のどちらかになります。

- サービス部品からの応答電文
- サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー情報（フォルト情報）

(1)　サービス部品からの応答電文

取得する応答電文は，サービス部品側と同じ電文フォーマットの XML 電文になります。XML 電文の応答電文については，「9.2.7　応答電文を取得する」を参照してください。

同期のサービス部品（Web サービスまたは SessionBean）からの応答電文がない場合は，ペイロードなしの TextMessage を受信します。

非同期のサービス部品の場合は，応答がありません。

(2)　サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー情報（フォルト情報）

取得する電文（エラー情報）は，非同期受付（MDB（WS-R））が応答用のキューに送信した XML 電文になります。

SOAP1.2 の場合と SOAP1.2 以外の場合とに分けて，エラー情報の出力形式およびスキーマを示します。

(a)　SOAP1.2 以外の場合

SOAP1.2 以外を利用している場合の取得する XML 電文（エラー情報）の形式を次に示します。

表 9-15　取得する XML 電文（エラー情報）の形式（SOAP1.2 以外の場合）

タグ	説明
errorcode	サービス部品，ビジネスプロセス，およびサービスアダプタからのエラーであることを示すエラーコードです。
errorstring	サービス部品，ビジネスプロセス，およびサービスアダプタからのエラーであることを示すエラーメッセージです。
processinstanceid	ビジネスプロセスのインスタンス ID の情報です。

タグ	説明
cscmsgcode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultCode 情報です。
cscmsgstring	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultString 情報です。
cscmsgactor	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの FaultActor 情報です。
cscmsgdetail	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。

SOAP1.2 以外を利用している場合の取得する XML 電文（エラー情報）のスキーマを次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cscmsgerror">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="errorcode" type="xsd:string"/>
        <xsd:element name="errorstring" type="xsd:string"/>
        <xsd:element name="processinstanceid" type="xsd:string"/>
        <xsd:element ref="errordetail"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="errordetail">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="cscmsgcode" type="xsd:string"/>
        <xsd:element name="cscmsgstring" type="xsd:string"/>
        <xsd:element name="cscmsgactor" type="xsd:string"/>
        <xsd:element name="cscmsgdetail" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

(b) SOAP1.2 の場合

SOAP1.2 を利用している場合の取得する XML 電文（エラー情報）の形式を次に示します。

表 9-16 取得する XML 電文（エラー情報）の形式（SOAP1.2 の場合）

タグ	説明
errorcode	サービス部品，ビジネスプロセス，およびサービスアダプタからのエラーであることを示すエラーコードです。
errorstring	サービス部品，ビジネスプロセス，およびサービスアダプタからのエラーであることを示すエラーメッセージです。

タグ	説明
processinstanceid	ビジネスプロセスのインスタンス ID の情報です。
cscmsgcode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Code 情報です。
cscmsgvalue	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Code が持つ Value 情報です。
cscmsgreason	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Reason 情報です。
cscmsgtext	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Reason が持つ Text 情報です。
cscmsgrole	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Role 情報です。
cscmsgnode	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Node 情報です。
cscmsgdetail	サービス部品（Web サービス），ビジネスプロセス，またはサービスアダプタからの Detail 情報です。

SOAP1.2 を利用している場合の取得する XML 電文（エラー情報）のスキーマを次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cscmsgerror">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="errorcode" type="xsd:string"/>
        <xsd:element name="errorstring" type="xsd:string"/>
        <xsd:element name="processinstanceid" type="xsd:string"/>
        <xsd:element ref="errordetail"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="errordetail">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="cscmsgcode"/>
        <xsd:element ref="cscmsgreason"/>
        <xsd:element name="cscmsgrole" type="xsd:string"/>
        <xsd:element name="cscmsgnode" type="xsd:string"/>
        <xsd:element name="cscmsgdetail" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="cscmsgcode" nillable="true">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="cscmsgvalue" type="xsd:string" maxOccurs="unbounded" nillable="tr
ue"/>
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="cscmsgreason" nillable="true">
    <xsd:complexType>
      <xsd:sequence>
      <xsd:element name="cscmsgtext" type="tns:cscmsgFaultReasonText" maxOccurs="unbounded"
nillable="true" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xs:complexType name="cscmsgFaultReasonText">
    <xs:sequence>
      <xs:element name="locale" type="xs:string" nillable="true" />
      <xs:element name="text" type="xs:string" nillable="true" />
    </xs:sequence>
  </xs:complexType>
</xsd:schema>
```

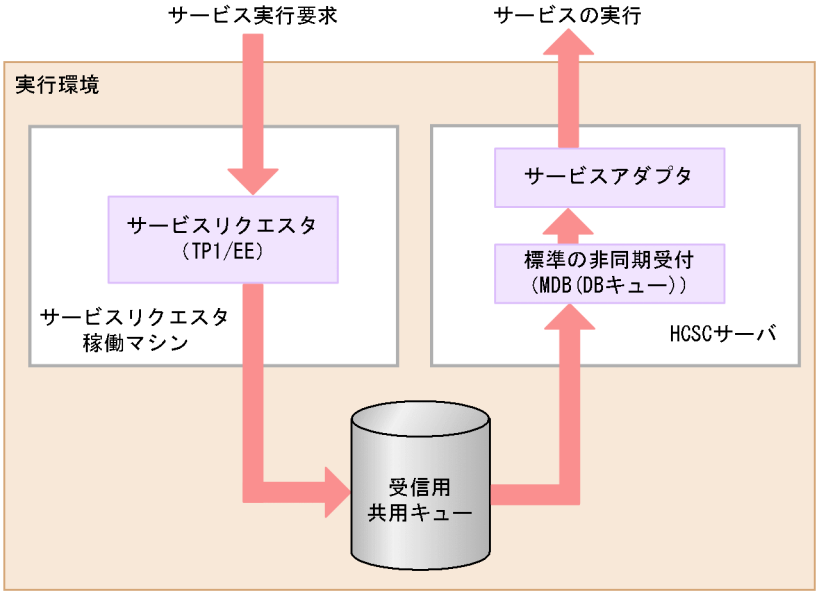
9.6 標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ

サービス部品を呼び出す HCSC サーバの標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタは，TP1/Server Base Enterprise Option（TP1/EE）と JMS の 2 つがあります。

TP1/Server Base Enterprise Option（TP1/EE）のサービスリクエスタ

サービスリクエスタ（TP1/EE）とサービス部品を呼び出す HCSC サーバの標準の非同期受付（MDB（DB キュー））の間は，受信用共用キューを通して通信を行います。標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ（TP1/EE）と HCSC サーバの関係を次の図に示します。

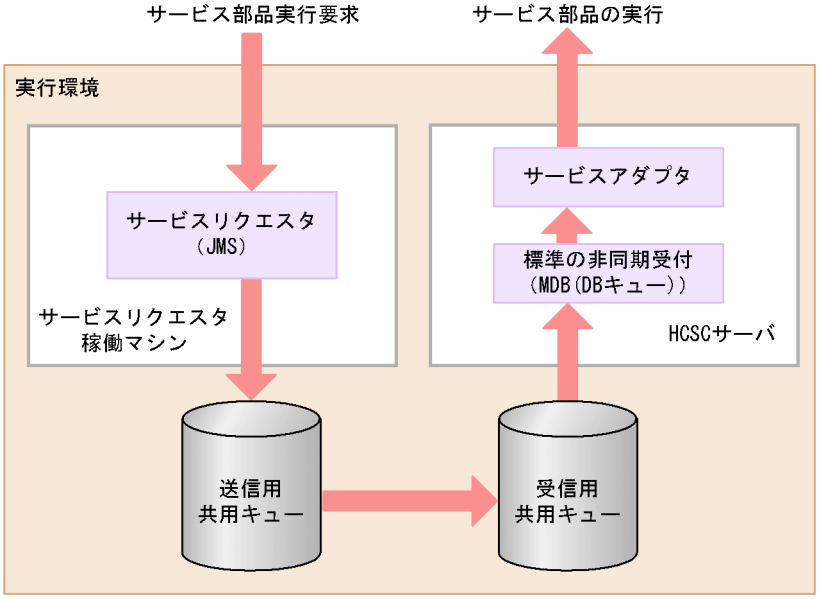
図 9-23 標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ（TP1/EE）と HCSC サーバの関係



JMS のサービスリクエスタ

サービスリクエスタ（JMS）とサービス部品を呼び出す HCSC サーバの標準の非同期受付（MDB（DB キュー））の間は，送信用共用キューおよび受信用共用キューを通して通信を行います。標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ（JMS）と HCSC サーバの関係を次の図に示します。

図 9-24　非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ（JMS）と HCSC サーバの関係



9.6.1　サービスリクエスタ（標準の非同期受付（MDB（DB キュー））の作成手順

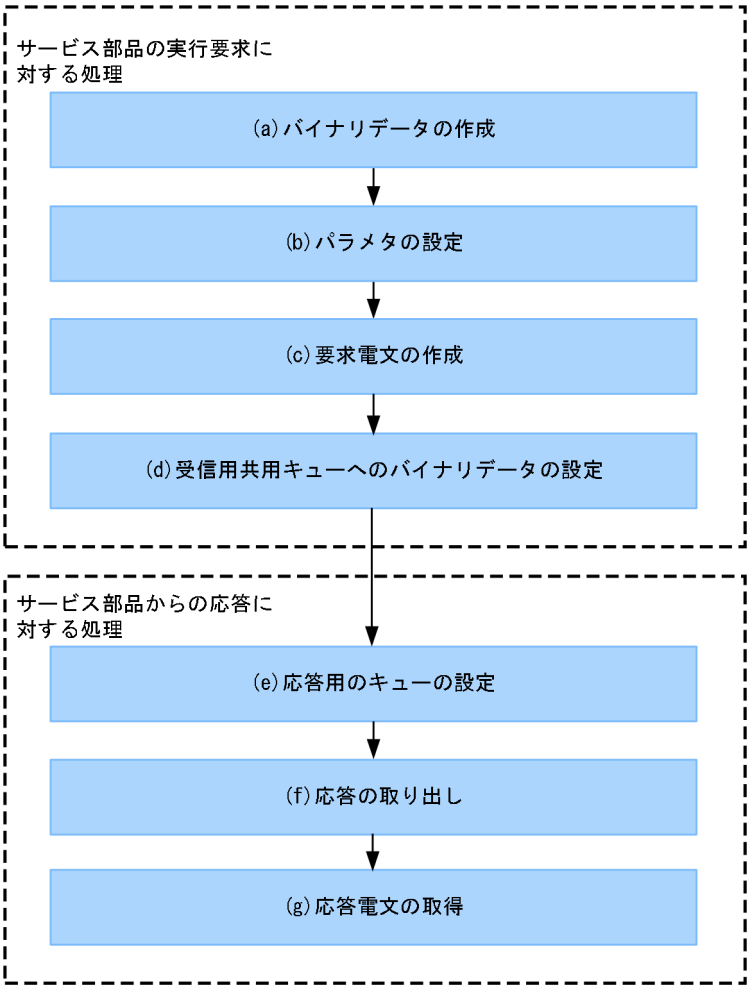
標準の非同期受付（MDB（DB キュー））にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

作成手順は，サービスリクエスタが TP1/EE の場合と JMS の場合で異なります。

(1)　TP1/EE の場合

サービスリクエスタが TP1/EE の場合に，標準の非同期受付（MDB（DB キュー））にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-25　サービスリクエスタ（TP1/EE）の作成手順（標準の非同期受付（MDB（DB キュー））



(a)　バイナリデータの作成

標準の非同期受付（MDB（DB キュー））に要求電文を送信する場合，バイナリデータを作成する必要があります。バイナリデータの作成については，「9.6.4　バイナリデータを作成する（TP1/EE，JMS）」を参照してください。

(b)　パラメタの設定

「(1)(a)　バイナリデータの作成」で作成したバイナリデータの各タグに対応するパラメタを設定して，バイナリデータを組み立てます。パラメタの設定については，「9.6.5　パラメタを設定する（TP1/EE，JMS）」を参照してください。

(c)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.6.6　要求電文を作成する（TP1/EE，JMS）」を参照してください。

(d)　受信用共用キューへのバイナリデータの設定

パラメタを設定したバイナリデータを TP1/EE の API（DB キュー制御）を使用して，受信用共用キューがあるデータベースに設定します。受信用共用キューへのバイナリデータの設定については，「9.6.7　バイナリデータを受信用共用キューに設定する（TP1/EE）」を参照してください。

(e)　応答用のキューの設定

同期のサービス部品（Web サービスまたは SessionBean）から応答を受け取るために，応答用のキュー（受信用共用キュー）を設定します。応答用のキューの設定については，「9.6.9　応答用のキューを設定する（TP1/EE，JMS）」を参照してください。

(f)　応答の取り出し

TP1/EE の API（DB キュー制御）を使用して，応答用のキュー（受信用共用キュー）があるデータベースを参照して，応答のバイナリデータを取り出します。応答の取り出しについては，「9.6.10　応答を取り出す（TP1/EE，JMS）」を参照してください。

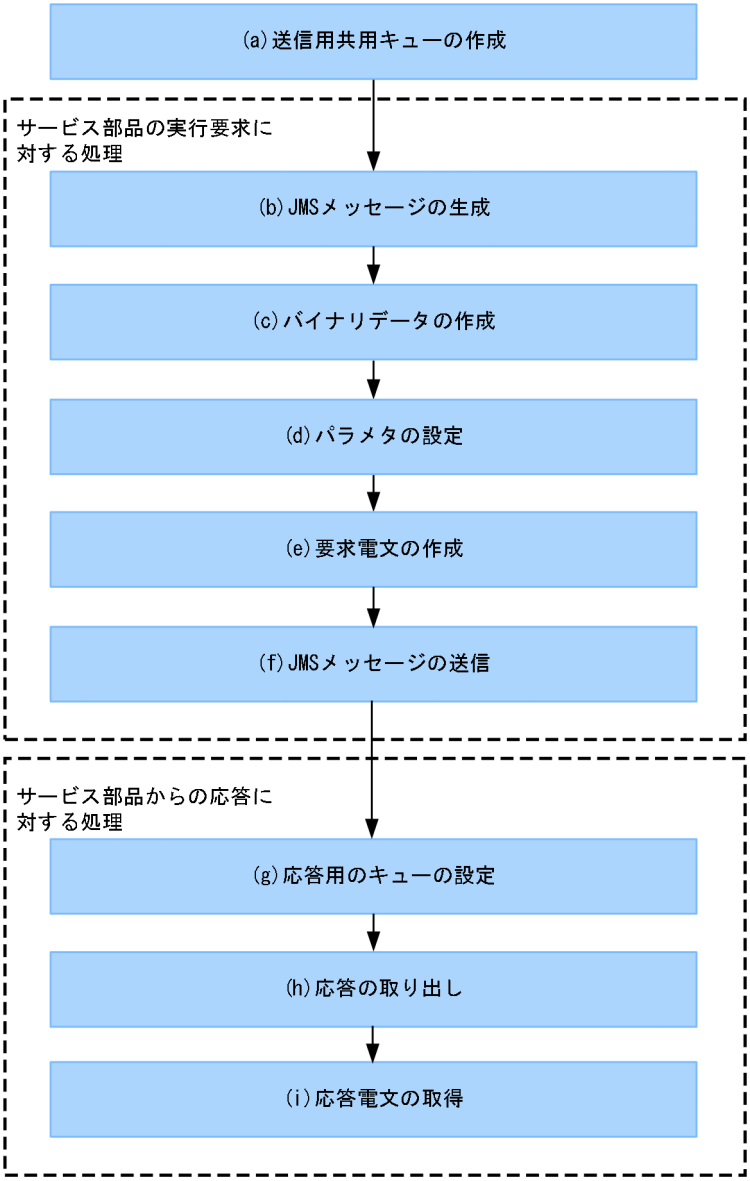
(g)　応答電文の取得

取り出した応答から応答電文を取得します。応答電文の取得については，「9.6.11　応答電文を取得する（TP1/EE，JMS）」を参照してください。

(2)　JMS の場合

サービスリクエスタが JMS の場合に，標準の非同期受付（MDB（DB キュー））にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-26　サービスリクエスタ（JMS）の作成手順（標準の非同期受付（MDB（DB キュー））



(a)　送信用共用キューの作成

サービスリクエスタ（JMS）から要求電文を送信するために必要な送信用共用キューを作成します。送信用共用キューの作成については，「9.6.2　送信用共用キューを作成する（JMS）」を参照してください。

(b)　JMS メッセージの生成

非同期受付（MDB（DB キュー））にサービス部品の実行を要求するために，QueueSender のオブジェクトおよび JMS メッセージを生成します。QueueSender のオブジェクトおよび JMS メッセージの生成については，「9.6.3　JMS メッセージを生成する（JMS）」を参照してください。

(c)　バイナリデータの作成

非同期受付（MDB（DB キュー））に要求電文を送信する場合，バイナリデータを作成する必要があります。バイナリデータの作成については，「9.6.4　バイナリデータを作成する（TP1/EE，JMS）」を参照してください。

(d)　パラメタの設定

「(2)(c)　バイナリデータの作成」で作成したバイナリデータの各タグに対応するパラメタを設定して，バイナリデータを組み立てます。パラメタの設定については，「9.6.5　パラメタを設定する（TP1/EE，JMS）」を参照してください。

(e)　要求電文の作成

サービス部品の実行を要求する要求電文を作成します。要求電文の作成については，「9.6.6　要求電文を作成する（TP1/EE，JMS）」を参照してください。

(f)　JMS メッセージの送信

ペイロードにバイナリデータを設定した JMS メッセージを送信用共用キューに送信します。JMS メッセージの送信については，「9.6.8　JMS メッセージを送信する（JMS）」を参照してください。

(g)　応答用のキューの設定

同期のサービス部品（Web サービスまたは SessionBean）から応答を受け取るために，応答用のキュー（送信用共用キュー）を設定します。応答用のキューの設定については，「9.6.9　応答用のキューを設定する（TP1/EE，JMS）」を参照してください。

(h)　応答の取り出し

応答用のキュー（送信用共用キュー）から受信用共用キューに送信された応答を取り出します。応答の取り出しについては，「9.6.10　応答を取り出す（TP1/EE，JMS）」を参照してください。

(i)　応答電文の取得

取り出した応答から応答電文を取得します。応答電文の取得については，「9.6.11　応答電文を取得する（TP1/EE，JMS）」を参照してください。

9.6.2　送信用共用キューを作成する（JMS）

サービスリクエスタから要求電文を送信するために必要な送信用共用キューを作成します。送信用共用キューは，サービスリクエスタからの要求電文を，HCSC サーバが提供する受信用共用キューに送信します。

送信用共用キューの作成には，Reliable Messaging の hrmmkque コマンドを使用します。ここでは，hrmmkque コマンドのオプションの指定値について説明します。ここで説明していないオプションについ

ては，マニュアル「Reliable Messaging」の「8.3.17　hrmmkque（ローカルキューの作成)」を参照してください。

(1)　hrmmkque コマンドのオプションの指定値

-t キューの種類

shr_send（送信用共用キュー）を指定します。

-b 共用キューを使用する場合の登録先キュー名

次のどちらかを指定します。

＜Reliable Messaging システム名＞_SHR_CSC<HCSC サーバ名>ACPT_DBQ

HCSC サーバがロードバランスクラスタ構成の場合に指定します。

＜Reliable Messaging システム名＞_SHR_CSC<HCSC クラスタ名>ACPT_DBQ

HCSC サーバが HA クラスタ構成の場合に指定します。

HCSC クラスタ名は，開発環境および運用環境で HCSC サーバ構成配備時に決定される HCSC サーバのクラスタ名です。

9.6.3　JMS メッセージを生成する（JMS）

標準の非同期受付（MDB（DB キュー））にサービス部品の実行を要求するためには，QueueSender のオブジェクトおよび JMS メッセージを生成し，送信用共用キューに送信します。JMS メッセージを送信するアプリケーションの開発については，マニュアル「Reliable Messaging」を参照してください。

生成した JMS メッセージのペイロードには，パラメタを設定したバイナリデータを指定します。なお，JMS メッセージは BytesMessage で生成し，パラメタを設定したバイナリデータを BytesMessage のペイロードに指定してください。要求電文がない場合でも，バイナリデータを BytesMessage のペイロードに指定してください。

また，JMS メッセージのプロパティを設定しても，プロパティの値は無視されます。

JMS メッセージを生成する手順については，「9.5.3　JMS メッセージを生成する」を参照してください。

9.6.4　バイナリデータを作成する（TP1/EE，JMS）

標準の非同期受付（MDB（DB キュー））に要求電文を送信する場合，次の表に示す形式のバイナリデータを作成する必要があります。

表 9-17　バイナリデータの形式（要求電文）

項番	項目	長さ（バイト）	データ型※1	説明	必須
1	ヘッダタグ	4	char	サービス部品要求のメッセージ振り分け用の先頭タグです（アイキャッチャ）。 「DBQH」を指定します。 ASCII コードを使用してください。	○
2	バイトオーダーフラグ	1	byte	ヘッダ内の数値データのバイトオーダー（エンディアン）識別フラグです。 ヘッダ内の数値データをバイナリにエンコードする方式を次のどちらかから指定します。 • 0b00000001：ビッグエンディアン • 0b00000010：リトルエンディアン	○
3	予備	3	byte	予備領域です。	○
4	長さ	4	int	サービス部品要求のメッセージングのヘッダの長さ（項番 5 から項番 28 までの長さ）です。 バイト単位で指定してください。	○
5	タグ	8	char	パラメタ識別用のタグです（アイキャッチャ）。 「ServiceN」を指定します。 ASCII コードを使用してください。	△
6	長さ	4	int	次のエリア（項番 7）の長さです。 バイト単位で指定してください。	
7	サービス名	任意	String	要求先のサービス名です。 要求先のサービス名は，開発環境で定義したサービスアダプタまたはビジネスプロセスを指定してください。 UTF-8 コードを使用してください。	
8	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ClientID」を指定します。 ASCII コードを使用してください。	△
9	長さ	4	int	次のエリア（項番 10）の長さです。 バイト単位で指定してください。	
10	クライアント相関 ID	任意	String	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で指定してください。 サービスリクエスタからの要求電文と，HCSC サーバで管理している実行履歴やログおよびトレースを対応づけるために使用します。したがって，HCSC サーバに送信した要求電文ごとに異なる ID を指定してください。 UTF-8 コードを使用してください。	

項番	項目	長さ（バイト）	データ型※1	説明	必須
10	クライアント相関 ID	任意	String	省略する場合，このパラメタ，長さ，およびタグの，バイナリデータを組み立てる必要はありません。	△
11	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ReqFmtID」を指定します。 ASCII コードを使用してください。	×
12	長さ	4	int	次のエリア（項番 13）の長さです。 バイト単位で指定してください。	
13	HCSC リクエストフォーマット ID	任意	String	要求電文のフォーマット定義を一意に識別するための ID です。 このパラメタ，長さ，およびタグのバイナリデータは組み立てないでください。	
14	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ResFmtID」を指定します。 ASCII コードを使用してください。	×
15	長さ	4	int	次のエリア（項番 16）の長さです。 バイト単位で指定してください。	
16	HCSC レスポンスフォーマット ID	任意	String	応答電文のフォーマット定義を一意に識別するための ID です。 このパラメタ，長さ，およびタグのバイナリデータは組み立てないでください。	
17	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ReplyToQ」を指定します。 ASCII コードを使用してください。	△
18	長さ	4	int	次のエリア（項番 19）の長さです。 バイト単位で指定してください。	
19	応答用のキュー名	任意	String	要求先のサービス部品が同期（Web サービスまたは SessionBean）またはビジネスプロセスの場合に，サービス部品またはビジネスプロセスからの応答を受け取るキュー名です。 要求先のサービス部品が同期か非同期かわからない場合にこの指定をすると，応答を受け取ることができます。 詳細については，「9.6.9　応答用のキューを設定する（TP1/EE, JMS）」を参照してください。 UTF-8 コードを使用してください。 省略すると，NULL が設定され，応答を受け取ることができません。省略する場合，このパラメタ，長さ，およびタグの，バイナリデータを組み立てる必要はありません。	
20	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「OperatiN」を指定します。 ASCII コードを使用してください。	△
21	長さ	4	int	次のエリア（項番 22）の長さです。	

項番	項目	長さ（バイト）	データ型※1	説明	必須
21	長さ	4	int	バイト単位で指定してください。	△
22	サービスオペレーション名	任意	String	要求先のサービス名に対するオペレーション名です。※2 オペレーション名は，開発環境で定義したサービス部品のオペレーション名を指定します。255 バイト以内の XMLSchema の NCName 定義文字で指定してください。 要求先のサービス部品が同期のサービス部品（Web サービスまたは SessionBean）またはビジネスプロセスの場合，必ず指定してください。 UTF-8 コードを使用してください。 要求先のサービス部品が非同期のサービス部品の場合，応答用のキュー名を省略できます。省略する場合，このパラメタ，長さ，およびタグの，バイナリデータを組み立てる必要はありません。	
23	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「MessageT」を指定します。 ASCII コードを使用してください。	△
24	長さ	4	int	次のエリア（項番 25）の長さです。 バイト単位で指定してください。	
25	電文種別	任意	String	ユーザ電文（要求電文）の種別です。 XML 電文（ペイロードが TextMessage）の場合は「XML」を，バイナリ電文（ペイロードが BytesMessage）の場合は「Binary」を指定します。 メッセージ（要求電文）がある場合，必ず指定してください。 UTF-8 コードを使用してください。 要求電文がない場合，省略することができます。省略する場合，このパラメタ，長さ，およびタグの，バイナリデータを組み立てる必要はありません。	
26	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「MgLength」を指定します。 ASCII コードを使用してください。	○
27	長さ	4	int	次のエリア（項番 28）の長さです。 「4」を指定してください。	
28	メッセージ長	4	int	バイナリ変換したユーザ電文（要求電文）のメッセージの長さを指定します。 ユーザ電文（要求電文）がない場合，0 を指定してください。	
29	メッセージ	任意	byte[]	ユーザ電文（要求電文）です。※2 DB キューのヘッダ（項番 1～28）の後ろに，バイナリ変換したユーザ電文（要求電文）を指定します。 なお，XML 電文をバイナリに変換したメッセージの場合は，UTF-8 コードでエンコードしてください。	△

項番	項目	長さ（バイト）	データ型※1	説明	必須
29	メッセージ	任意	byte[]	ユーザ電文（要求電文）がない場合，省略できます。	△

（凡例）

○：必ず指定してください。

△：必ず指定しなければいけない場合があります。説明の内容を確認してください。

×：指定しないでください。

注※1

Java のデータ型です。

注※2

要求先のサービス部品のプロトコルが SOAP の場合，ユーザ電文のルート要素の名称（データ変換を行う場合，データ変換後の電文のルート要素の名称）から呼び出すオペレーションを決定します。そのため，ユーザ電文のルート要素に誤った名称を指定した場合，意図しないオペレーションを呼び出してしまうので注意してください。

9.6.5 パラメタを設定する（TP1/EE，JMS）

作成したバイナリデータの各タグ（表 9-17 の項番 5，8，11，17，20，23，および 26）に対応するパラメタを設定して，バイナリデータを組み立てます。

バイナリデータのタグの形式を次の図に示します。

図 9-27 バイナリデータのタグの形式



バイナリデータ中の各タグの出現順序は任意です。ただし，タグ，長さ，値の順序は，この図に示すように固定されています。

また，DB キューのヘッダ（項番 1～28）の後ろに，バイナリ変換したメッセージ（要求電文）を指定します。要求電文がない場合，項番 26 のタグに MgLength を指定し，その長さを 0 にすることもできます。

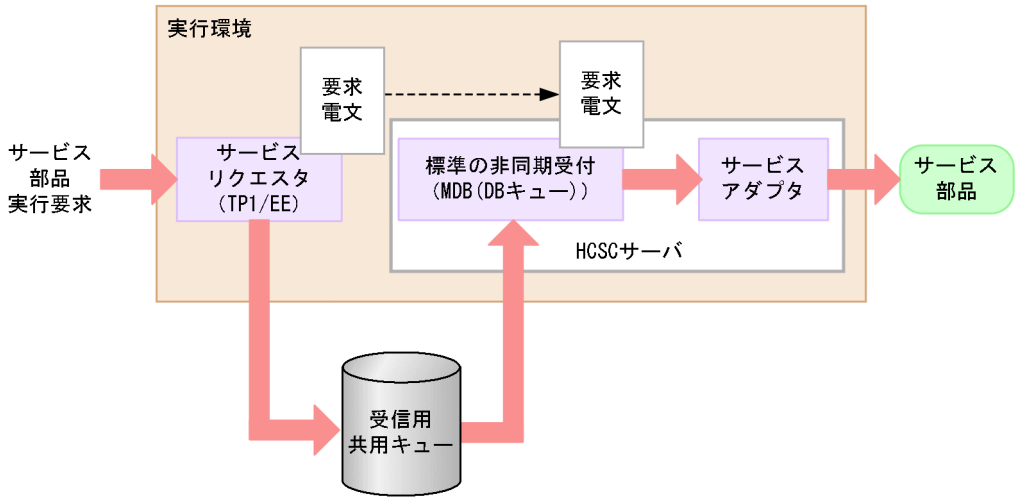
9.6.6 要求電文を作成する（TP1/EE，JMS）

サービスリクエスタから HCSC サーバの非同期受付（MDB（DB キュー））に，サービス部品を要求する要求電文を作成します。サービスリクエスタから送信する要求電文の内容は，サービス部品側と同じ電文フォーマットで作成してください。

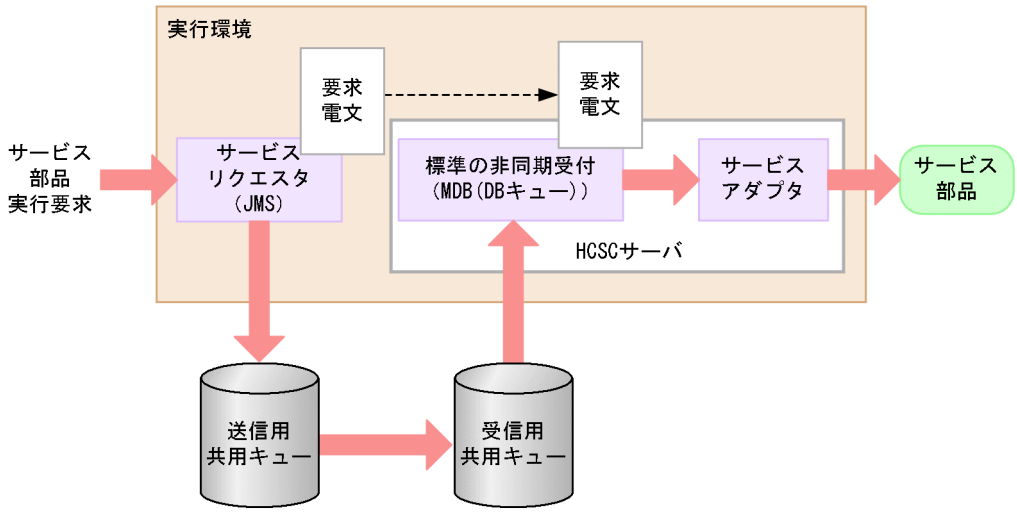
サービスリクエスタが TP1/EE の場合と JMS の場合の，要求電文の送信を次の図に示します。

図 9-28　要求電文の送信（標準の非同期受付（MDB（DB キュー））

●TP1/EEの場合



●JMSの場合



正常な要求電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.5　DB キューを使用したサービスとの接続」を参照してください。

また，サービスリクエスタでは，電文種別が XML 電文の要求電文を作成します。作成した要求電文はバイナリ変換します。バイナリ変換する場合は，サービスリクエスタ側とサービス部品側で同じエンコードの方式を使用してください。XML 電文の要求電文については，「9.2.6　要求電文を作成する」を参照してください。

なお，プロトコルの種類が DB キューのサービス部品の場合，非同期受付（MDB（DB キュー））から要求電文を送信できません。非同期受付（MDB（DB キュー））から要求電文を送信できるのは，サービス部品側で次のプロトコルを使用しているときになります。

- SOAP
- RMI-IIOP
- WS-R

9.6.7　バイナリデータを受信用共用キューに設定する（TP1/EE）

パラメタを設定したバイナリデータを TP1/EE の API（DB キュー制御）を使用して，受信用共用キューがあるデータベースに設定します。

詳細は，マニュアル「TP1/Server Base Enterprise Option　プログラム作成の手引」を参照してください。

9.6.8　JMS メッセージを送信する（JMS）

ペイロードにバイナリデータを設定した JMS メッセージを，送信用共用キューに送信します。

JMS メッセージを送信する手順については，「9.5.6　JMS メッセージを送信する」を参照してください。

9.6.9　応答用のキューを設定する（TP1/EE，JMS）

標準の非同期受付（MDB（DB キュー））は，非同期の要求電文の標準受付であり，基本的にはサービス部品からの応答がない場合に使用します。しかし，非同期受付（MDB（DB キュー））から同期のサービス部品（Web サービスまたは SessionBean）を要求した場合，同期のサービス部品から応答が返るときがあります。このような場合，要求時に応答用のキューを指定することで，サービスリクエスタは同期のサービス部品から応答を受け取ることができます。要求時の応答用のキューの指定については，「9.5.6 JMS メッセージを送信する」を参照してください。

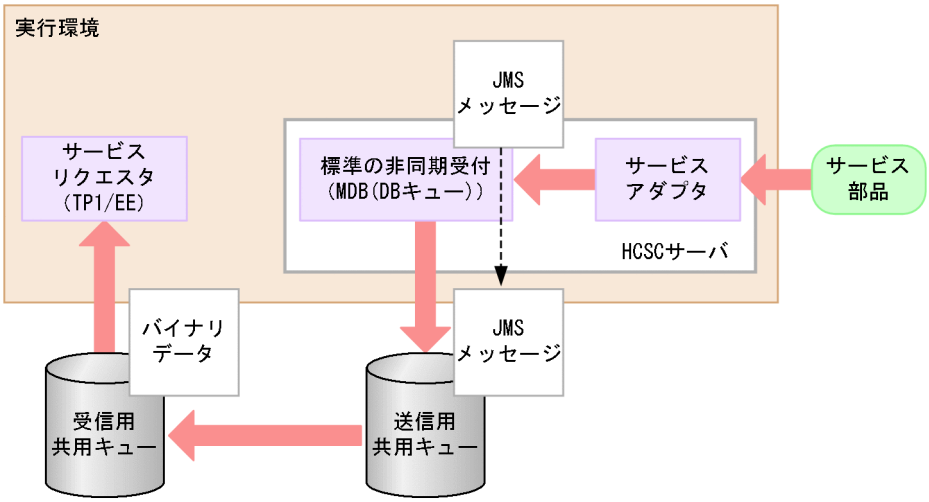
(1)　TP1/EE の場合

非同期受付（MDB（DB キュー））は，応答電文を含むバイナリデータがペイロードに設定された JMS メッセージを応答用のキュー（送信用共用キュー）に送信します。

送信用共用キューは，受け取った応答からバイナリデータを取り出し，受信用共用キューに送信します。受信用共用キューに送信した応答を取り出して，サービス部品からの応答電文を取得できます。応答の取り出しについては，「9.6.10　応答を取り出す（TP1/EE，JMS）」を参照してください。

標準の非同期受付（MDB（DB キュー））と送信用共用キュー，および受信用共用キューの関係を次に示します。

図 9-29　標準の非同期受付（MDB（DB キュー））と送信用共用キュー，および受信用共用キューの関係（TP1/EE の場合）



送信用共用キュー（応答用のキュー）

応答用のキューとなる送信用共用キューを，HCSC サーバが稼働している J2EE サーバ（Reliable Messaging）内に作成します。送信用共用キューの登録先キューには，受信用共用キューを指定します。

送信用共用キューの作成方法については，マニュアル「Reliable Messaging」の「2.2.4　送信用共用キュー」を参照してください。

受信用共用キュー

TP1/EE の DB キューを，送信用共用キューがある（Reliable Messaging が使用している）データベースの同一のスキーマ定義内に作成します。詳細は，マニュアル「Reliable Messaging」の「2.2.3 受信用共用キュー」を参照してください。

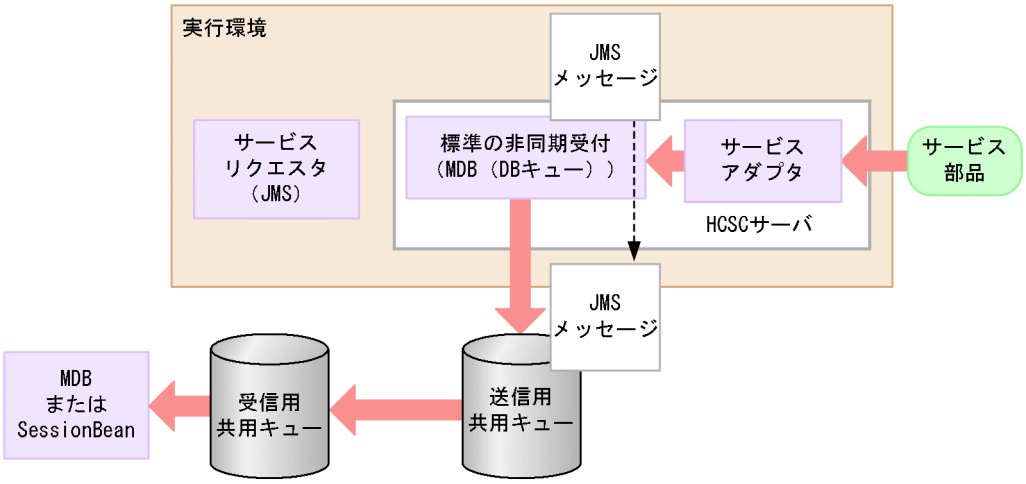
また，DB キューの作成の詳細については，マニュアル「TP1/Server Base Enterprise Option　使用の手引」の DB キューに関する内容を参照してください。

(2)　JMS の場合

標準の非同期受付（MDB（DB キュー））は応答として，応答電文を含むバイナリデータがペイロードに設定された JMS メッセージを応答用のキュー（送信用共用キュー）に送信します。送信用共用キューで受け取った応答は，受信用共用キューに送信します。受信用共用キューに送信した応答を取り出すことで，サービス部品からの応答電文を取得できます。応答の取り出しについては，「9.6.10　応答を取り出す（TP1/EE，JMS）」を参照してください。

標準の非同期受付（MDB（DB キュー））と送信用共用キュー，および受信用共用キューの関係を次に示します。

図 9-30　標準の非同期受付（MDB（DB キュー））と送信用共用キュー，および受信用共用キューの関係（JMS の場合）



送信用共用キュー（応答用のキュー）

応答用のキューとなる送信用共用キューを，HCSC サーバが稼働している J2EE サーバ（Reliable Messaging）内に作成します。送信用共用キューの登録先キューには，受信用共用キューを指定します。

送信用共用キューの作成方法については，マニュアル「Reliable Messaging」の「2.2.4　送信用共用キュー」を参照してください。

受信用共用キュー

サービスリクエスタが稼働している J2EE サーバ（Reliable Messaging）内に，受信用共用キューを作成します。

受信用共用キューの設定は，ユーザの運用に合わせ，必要に応じて行ってください。

受信用共用キューの作成方法については，マニュアル「Reliable Messaging」の「2.2.3　受信用共用キュー」を参照してください。

9.6.10　応答を取り出す（TP1/EE，JMS）

(1)　TP1/EE の場合

TP1/EE の API（DB キュー制御）を使用して，応答用のキュー（受信用共用キュー）があるデータベースを参照して，応答のバイナリデータを取り出します。正常な応答電文の伝わり方については，マニュアル「サービスプラットフォーム 解説」の「2.5.3　同期のサービス部品を呼び出す場合の応答電文の扱い」を参照してください。

詳細については，マニュアル「TP1/Server Base Enterprise Option　プログラム作成の手引」を参照してください。

応答のバイナリデータの形式を次の表に示します。

表 9-18　バイナリデータの形式（応答電文）

項番	項目	長さ（バイト）	データ型※	説明
1	ヘッダタグ	4	char	HCSC メッセージ振り分け用の先頭タグです（アイキャッチャ）。 「DBQH」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
2	バイトオーダーフラグ	1	byte	要求時に指定したバイトオーダーフラグが設定されます。
3	予備	3	byte	予備領域です。
4	長さ	4	int	HCSC メッセージング用のヘッダの長さ（項番 5 から項番 19 までの長さ）が設定されます。
5	タグ	8	char	パラメタ識別用のタグです（アイキャッチャ）。 「ServiceN」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
6	長さ	4	int	次のエリア（項番 7）の長さです。 バイト単位で設定されます。
7	サービス名	任意	String	要求時に指定したサービスアダプタまたはビジネスプロセスが設定されます。 UTF-8 コードでエンコードされているバイナリデータが設定されます。
8	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ClientID」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
9	長さ	4	int	次のエリア（項番 10）の長さです。 バイト単位で設定されます。
10	クライアント相関 ID	任意	String	サービスリクエスタからの要求電文を一意に識別するための相関識別子です。 255 文字以内の英数字，アンダーバー（_），ピリオド（.）およびハイフン（-）で設定されます。 UTF-8 コードでエンコードされているバイナリデータが設定されます。
11	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「ResFmtID」を指定します。 ASCII コードを使用してください。
12	長さ	4	int	次のエリア（項番 13）の長さです。 バイト単位で指定してください。
13	HCSC レスポンスフォーマット ID	任意	String	応答電文のフォーマット定義を一意に識別するための ID です。 このパラメタ，長さ，およびタグは設定されません。
14	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「OperatiN」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
15	長さ	4	int	次のエリア（項番 16）の長さです。

項番	項目	長さ（バイト）	データ型※	説明
15	長さ	4	int	バイト単位で設定されます。
16	サービスオペレーション名	任意	String	要求先のサービス名に対するオペレーション名です。 255 バイトの XMLSchema の NCName 定義文字で設定されます。 UTF-8 コードでエンコードされているバイナリデータが設定されます。
17	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「MessageT」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
18	長さ	4	int	次のエリア（項番 19）の長さです。 バイト単位で設定されます。
19	応答電文種別	任意	String	メッセージ（応答電文）の種別です。 電文種別が XML 電文の場合，「XML」が設定されます。メッセージ（応答電文）がない場合，このパラメタは省略されます。 UTF-8 コードでエンコードされているバイナリデータが設定されます。
20	タグ	8	byte	パラメタ識別用のタグです（アイキャッチャ）。 「MgLength」が設定されます。 ASCII コードでエンコードされているバイナリデータが設定されます。
21	長さ	4	int	次のエリア（項番 22）の長さです。 「4」が設定されます。
22	メッセージ長	4	int	メッセージ（応答電文）の長さが設定されます。 メッセージ（応答電文）がない場合，0 バイトが設定されます。
23	メッセージ	任意	byte[]	次のどちらかの情報が設定されます。 • サービス部品からの応答電文 • サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー情報（フォルト情報） UTF-8 コードでエンコードされているバイナリデータが設定されます。

注※

Java のデータ型です。

(2) JMS の場合

受信用共用キューから応答（JMS メッセージ）を取り出すには，MDB で実装する方法と SessionBean で実装する方法があります。MDB および SessionBean の実装については，マニュアル「Reliable Messaging」を参照してください。

なお，取り出した応答（JMS メッセージ）のペイロードに設定されたバイナリデータの形式については，表 9-18 を参照してください。

9.6.11　応答電文を取得する（TP1/EE，JMS）

取り出した応答から応答電文を取得します。取得する応答電文は，次のどちらかになります。

- サービス部品からの応答電文
- サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー情報（フォルト情報）

(1)　サービス部品からの応答電文

取得する応答電文は，サービス部品側と同じ電文フォーマットの XML 電文になります。XML 電文の応答電文については，「9.2.7　応答電文を取得する」を参照してください。

(2)　サービス部品，ビジネスプロセス，およびサービスアダプタからのエラー情報（フォルト情報）

取得する電文（エラー情報）は，非同期受付（MDB（DB キュー））が応答用のキューに送信した XML 電文になります。取得する XML 電文（エラー情報）の形式については，「9.5.9　応答電文を取得する」を参照してください。

9.7　SOAP 受付に要求を出すサービスリクエスタ

SOAP 受付へ要求電文を送信するサービスリクエスタは，SOAP を利用して通信します。サービスリクエスタから SOAP 受付へサービス部品実行の要求電文が送信され，HCSC サーバがサービス部品の実行を行います。

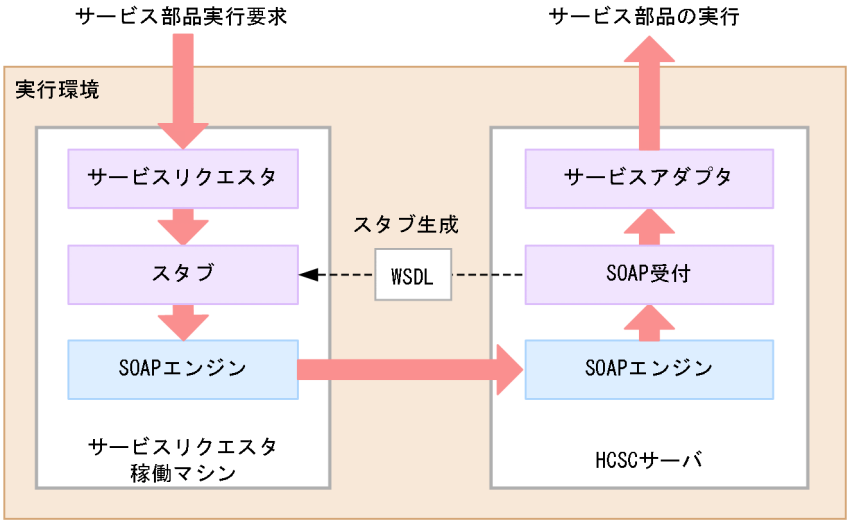
SOAP 受付のインターフェース情報は，開発環境でユーザ定義受付を定義した際に指定した WSDL になります。WSDL からスタブを生成し，そのスタブを利用して，SOAP 受付に要求を出します。

ポイント

SOAP 受付に要求を出すサービスリクエスタを作成する場合，標準受付とは異なり，要求電文を作成する必要はありません。サービスリクエスタのスタブの中で WSDL に合わせた形式の要求電文（SOAP メッセージ）が自動的に生成され，サービス部品の実行要求が出されます。

SOAP 受付に要求を出すサービスリクエスタと HCSC サーバの関係を次の図に示します。

図 9-31　SOAP 受付に要求を出すサービスリクエスタと HCSC サーバの関係

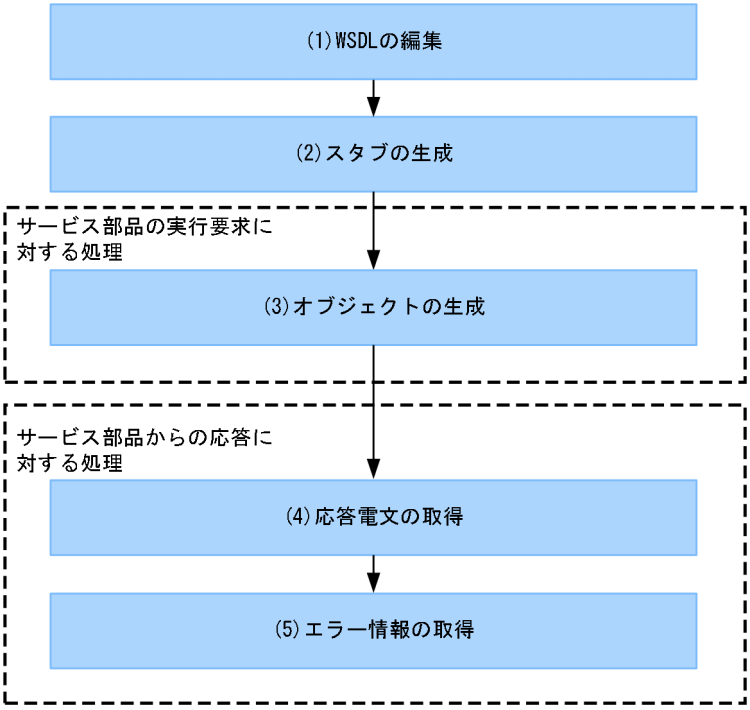


なお，JAX-WS エンジンを使用して通信するサービスリクエスタの場合，スタブではなくサービスクラスを生成します。JAX-WS エンジンの場合のサービスリクエスタの作成については，「9.7.7　サービスリクエスタ（SOAP 受付）の作成手順（JAX-WS エンジン）」を参照してください。

9.7.1　サービスリクエスタ（SOAP 受付）の作成手順（SOAP 通信基盤）

SOAP 受付にサービス部品の実行を要求し，サービス部品を呼び出すためのサービスリクエスタの作成手順を次に示します。

図 9-32　サービスリクエスタ（SOAP 受付）の作成手順（SOAP 通信基盤）



各工程の作業を次に示します。

(1)　WSDL の編集

ユーザ定義受付の定義内容を確認して，スタブの生成に必要な WSDL の内容を編集します。WSDL の編集については，「9.7.2　WSDL を編集する」を参照してください。

(2)　スタブの生成

「(1)　WSDL の編集」で編集した WSDL からスタブを生成します。スタブの生成については，「9.7.3 スタブを生成する」を参照してください。

(3)　オブジェクトの生成

SOAP 受付のメソッドを呼び出すために，「(2)　スタブの生成」で生成したスタブからオブジェクトを生成します。オブジェクトの生成については，「9.7.4　オブジェクトを生成する」を参照してください。

(4)　応答電文の取得

SOAP 受付から，サービス部品の実行要求に対する応答電文を取得します。応答電文の取得については，「9.7.5　応答電文を取得する」を参照してください。

(5)　エラー情報の取得

要求先のサービス部品，HCSC サーバ，および SOAP エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。エラー情報の取得については，「9.7.6　エラー情報を取得する」を参照してください。

9.7.2　WSDL を編集する

SOAP 受付に実行要求を出すサービスリクエスタを作成するには，開発環境でユーザ定義受付を定義した際に指定した WSDL を利用します。WSDL の作成については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.11　WSDL の作成」を参照してください。

開発環境でユーザ定義受付を定義した際に指定した WSDL は，サービスロケーションの値（wsdl:port 要素内の soap-address 要素の location 属性に設定している値）に，仮の値を設定しています。そのため，WSDL のサービスロケーションの値を，使用するユーザ定義受付の URL 情報に編集する必要があります。

HCSC サーバをセットアップする際に，HCSC サーバセットアップ定義ファイルに SOAP 受付を使用するように指定（request-userdef-soap=ON を指定）している場合は，開発環境のユーザ定義受付情報表示画面の［Web サービス］に URL が表示されます。表示される URL がユーザ定義受付の URL 情報です。この URL を WSDL のサービスロケーションの値に設定します。

HCSC サーバをセットアップしていない場合，または HCSC サーバをセットアップしたが HCSC サーバセットアップ定義ファイルで SOAP 受付を使用しないように指定している場合は，次の規則に従ってサービスロケーションの値を設定してください。

```
http://<ホスト名>:<ポート番号>※1/<コンテキストルート>※2/services/CSCMsgUserDefinedReception
```

注※1

HCSC サーバの URL（ホスト名およびポート番号）です。

注※2

コンテキストルート名は，開発環境でユーザ定義受付を定義したときに割り当てられた受付 ID と同じ値がデフォルトで設定されます。ユーザ定義受付定義画面の［コンテキストルート］に表示されます。

ユーザ定義受付情報表示画面の表示方法は，「8.4.2　HCSC コンポーネントの情報を表示する」を参照してください。

9.7.3　スタブを生成する

編集した WSDL からスタブを生成します。スタブは，サービスプラットフォームが開発支援コマンドとして提供する WSDL2Java コマンドで生成できます。

コマンドの入力例を次に示します。

```
WSDL2Java xxxxx.wsdl
```

このコマンドを実行すると，指定した WSDL に記述されている内容に基づいてディレクトリおよびファイルが作成されます。

WSDL2Java コマンドのオプションなどの詳細については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」の「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。生成されるスタブの内容については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」の「3.7.1　スタブの使用」を参照してください。

9.7.4　オブジェクトを生成する

SOAP 受付のメソッドを呼び出すために，生成したスタブを使用して，オブジェクトを生成します。

SOAP 受付のメソッドを呼び出すためのオブジェクトを生成する手順を次に示します。

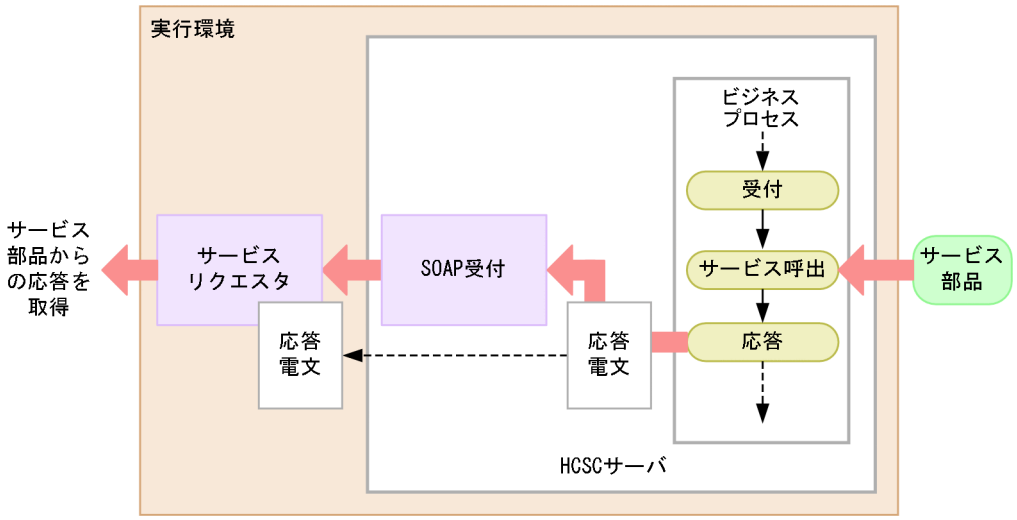
1. インターフェースクラスである Locator クラスのオブジェクトを生成します。
2. Locator クラスを使って，ユーザ定義受付のインターフェースクラスのオブジェクトを生成します。
   なお，生成・取得したサービスリクエスタのインターフェースクラスのインスタンスは，複数のスレッドで共有できません。

生成したオブジェクトのメソッドを呼び出すことで，ユーザ定義受付にサービス部品の実行が要求されます。スタブのオブジェクト内で，WSDL で定義された形式の要求電文（SOAP メッセージ）を自動生成します。

9.7.5　応答電文を取得する

SOAP 受付から，サービス部品の実行要求に対する応答電文を取得します。スタブのオブジェクトは，SOAP 受付からの応答電文として WSDL で定義された形式の応答電文（SOAP メッセージ）を受け取り，サービスリクエスタに応答を返します。応答電文の取得を次の図に示します。

図 9-33　応答電文の取得（SOAP 受付）



サービス部品からの応答電文がない場合は，NULL を受信します。応答電文がある場合とない場合に分けて次に説明します。

(1)　応答電文がある場合

サービスリクエスタは，生成したオブジェクトのメソッドを呼び出した戻り値としてビジネスプロセスから応答電文を取得します。

(2)　応答電文がない場合

サービスリクエスタは，生成したオブジェクトのメソッドを呼び出した戻り値として NULL を取得します。

9.7.6　エラー情報を取得する

要求先のサービス部品，HCSC サーバ，および SOAP エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。

エラーの伝わり方については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.7.1　Web サービス（SOAP 通信）実行時の障害対策」を参照してください。

(1)　サービスリクエスタ側の実装例

エラー情報の取得方法は，SOAP 通信基盤の種類によって異なります。

(a) サービスプラットフォームが提供している SOAP 通信基盤を使用している場合

サービス部品からの SOAP Fault の取得方法

WSDL に定義されているエラー情報用の例外オブジェクトをキャッチして SOAP Fault のエラー情報を取得します。SOAP Fault の形式は，WSDL に定義されている Fault 形式（サービス部品側の Fault 形式）です。

サービス部品からの SOAP Fault を取得するには，サービスリクエスタ側で次のように実装する必要があります。

```
{
    try {
        :
      // Webサービス呼び出し
        :
    } catch (xxxxxxxxxxException e) {
        :
    } catch (C4Fault e) {
        :
    }
}
```

HCSC サーバ内で発生した例外の取得方法

サービスプラットフォームの SOAP 通信基盤で提供する C4Fault オブジェクトをキャッチしてエラー情報を取得します。

サービスリクエスタ側の処理の流れを次に示します。

1. C4Fault または RuntimeException でキャッチします。
2. C4Fault から getFaultDetails()メソッドを使用して Element[]を取り出し，各配列に対して getChildNodes()メソッドで NodeList を取得します。
3. NodeList から Node を取り出し，個々の Node に対して getTextContent()メソッドでエラー情報を取り出します。

HCSC サーバ内で発生した例外を取得する場合の，サービスリクエスタ側の実装の例を次に示します。

```
{
    try {
        :
    } catch (xxxxxxxxxxException e) {
        :
    } catch (C4Fault e) {
      // C4FaultからFault情報を取得
      System.out.println("C4Fault Message          = " + e.getMessage());
      System.out.println("C4Fault FaultCode        = " + e.getFaultCode());
      System.out.println("C4Fault FaultActor       = " + e.getFaultActor());
      System.out.println("C4Fault FaultString      = " + e.getFaultString());
      // C4FaultからDetailを取得
      Element ele[] = e.getFaultDetails();
      printElement(ele);
    }
}
```

```
/**
 * Print Element
 */
  private static void printElement(Element[] ele) {
    int eleNumber = ele.length;
    System.out.println("Element count = " + eleNumber);
    for (int i=0; i<eleNumber; i++) {
      // Element配列からNodeListを取得
      if (ele[i] != null) {
        printNodeList(ele[i].getChildNodes());
      } else {
        System.out.println("Element[" + i + "] = null");
      }
    }
  }

/**
 * Print NodeList
 */
  private static void printNodeList(NodeList nodelist) {
    int nodelistcount = nodelist.getLength();
    for (int j=0; j<nodelistcount; j++) {
      Node node1 = nodelist.item(j);
      NodeList nodelist1 = node1.getChildNodes();
      int nodelist1len = nodelist1.getLength();

      if (nodelist1len == 0) {
        System.out.println("empty ChildNode");
      } else if(nodelist1len == 1) {
        Node node2 = node1.getFirstChild();
        if(node2 != null) {
          System.out.println("ChildNode[" + j + "] getTextContent = "
                             + node2.getTextContent());
        } else {
          System.out.println("ChildNode[" + j + "] = null");
        }
      } else {
        printNodeList(nodelist1);
      }
    }
  }
```

getFaultDetails()メソッドを使用して取得した Element[]の構造を次に示します。

表 9-19　getFaultDetails()メソッドを使用して取得した Element[]の構造

名称	説明
errorMessage	HCSC サーバ内で発生した例外に設定されているエラーメッセージです。
errorCode	HCSC サーバ内で発生した例外に対応するエラーコードです。
processInstanceID	ビジネスプロセスのインスタンス ID の情報です。 HCSC サーバ内で発生した例外の場合は設定されていないため，null（nil 属性）になります。

(b)　サービスプラットフォームが提供している SOAP 通信基盤を使用していない場合

サービス部品からの SOAP Fault の取得方法

WSDL に定義されているエラー情報用の例外オブジェクトをキャッチして SOAP Fault のエラー情報を取得します。SOAP Fault の形式は，WSDL に定義されている Fault 形式（サービス部品側の Fault 形式）です。

HCSC サーバ内で発生した例外の取得方法

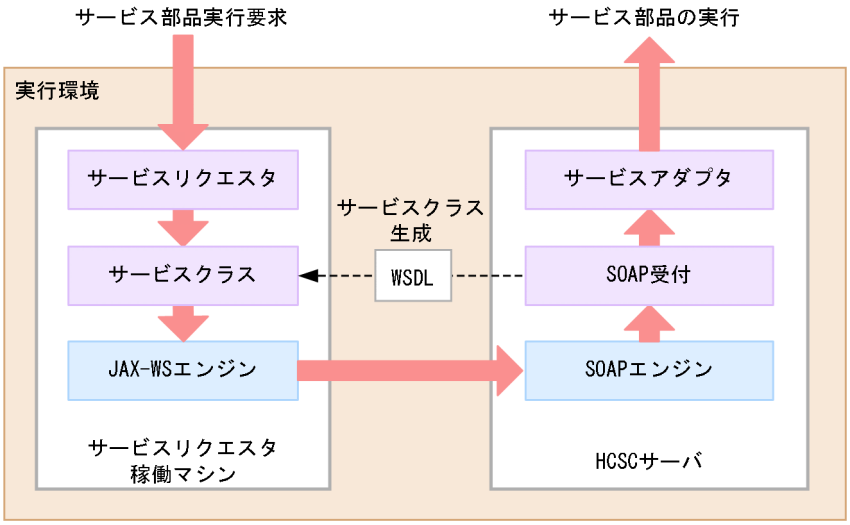
取得する方法は，サービスリクエスタ側で実装している SOAP エンジンに依存します。

9.7.7　サービスリクエスタ（SOAP 受付）の作成手順（JAX-WS エンジン）

SOAP 受付に要求を出すサービスリクエスタとして，JAX-WS エンジンを使用して通信するサービスリクエスタを作成できます。JAX-WS エンジンを使用して通信するサービスリクエスタは，SOAP（document-literal 型）を利用して，SOAP 受付に要求電文を送信します。

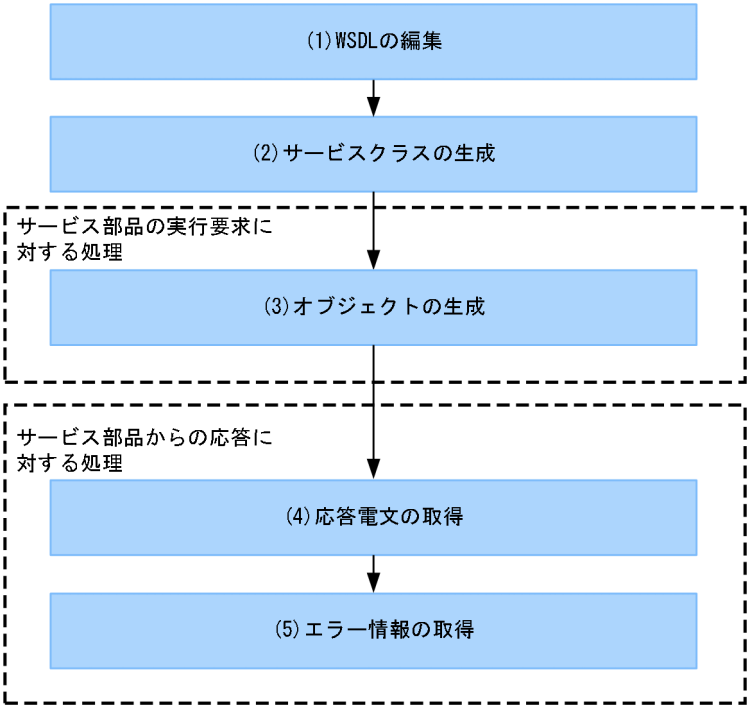
JAX-WS エンジンの場合のサービスリクエスタと HCSC サーバの関係を次の図に示します。

図 9-34　JAX-WS エンジンを使用して通信するサービスリクエスタと HCSC サーバの関係（SOAP 受付）



JAX-WS エンジンを使用して通信するサービスリクエスタの作成手順を次に示します。

図 9-35　JAX-WS エンジンを使用して通信するサービスリクエスタの作成手順（SOAP 受付）



各工程の作業を次に示します。

(1)　WSDL の編集

WSDL の編集方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

WSDL の編集方法については，「9.7.2　WSDL を編集する」を参照してください。

(2)　サービスクラスの生成

編集した WSDL からサービスクラスを生成します。サービスクラスは，サービスプラットフォームが開発支援コマンドとして提供する cjwsimport コマンドで生成します。

コマンドの入力例を次に示します。

```
cjwsimport -s <ソースファイルの出力先ディレクトリ> -d <コンパイル済みクラスファイルの出力先ディレクトリ> <WSDLファイル>
```

cjwsimport コマンドのオプションなどの詳細については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「14.1.1　cjwsimport コマンド」を参照してください。

なお，「(1)　WSDL の編集」で編集した WSDL ファイルが，cjwsimport コマンドを実行するカレントディレクトリと異なる場所に格納されている場合は，ディレクトリも指定してください。

このコマンドを実行すると，WSDL に記述されている内容に基づいて，指定したソースファイルの出力先ディレクトリに，ディレクトリおよびファイルが生成されます。

注意事項

JAX-WS エンジンを使用して通信するサービスリクエスタの場合，プログラム実行時に WSDL を読み込みます。サービスクラスのデフォルトコンストラクタを使用する場合は，cjwsimport コマンドで指定した WSDL のパス（「(1)　WSDL の編集」で取得した WSDL ファイルをサービスクラス生成のために格納したディレクトリ）の WSDL を読み込みます。そのため，cjwsimport コマンドを実行したあと，サービスクラスを基点とした WSDL ファイルの構成位置は，相対関係を壊すような構成にならないよう，WSDL ファイルを移動しないでください。サービスリクエスタ実行時に参照する WSDL の構成位置を，cjwsimport コマンドで指定した WSDL のパスから変更する場合は，URL を指定できるコンストラクタを使用してください。

(3)　オブジェクトの生成

SOAP 受付のメソッドを呼び出すために，生成したサービスクラスを使用して，プロキシクラスのオブジェクトを生成します。

生成したサービスクラスを基に，SOAP 受付を呼ぶためのプロキシクラスのオブジェクトを生成する手順を示します。

1. サービスクラスを生成します。
2. wsdl:portType に対応するプロキシクラスを生成します。
3. 生成したプロキシクラスのオブジェクトを使用して，サービスのメソッドを呼び出します。

生成したオブジェクトのメソッドを呼び出すことで，SOAP 受付にサービス部品の実行が要求されます。サービスクラスのオブジェクト内で，WSDL で定義された形式の要求電文（SOAP メッセージ）を自動生成します。

(4)　応答電文の取得

応答電文の取得方法は，SOAP 通信基盤でサービスリクエスタを作成する場合と同じです。

応答電文の取得については，「9.7.5　応答電文を取得する」を参照してください。

(5)　エラー情報の取得

要求先のサービス部品，HCSC サーバ，および SOAP エンジンでエラーが発生した場合，エラー情報を取得し，その情報に従って対処します。

サービス部品からの SOAP Fault の取得方法

JAX-WS エンジンを使用して通信するサービスリクエスタの場合，情報がラップされた例外がキャッチされます。そのため，ユーザ定義例外を取得するには，getFaultInfo()メソッドを使用してフォルト情報の格納クラスを取得する必要があります。

HCSC サーバ内で発生した例外の取得方法

javax.xml.ws.soap.SOAPFaultException オブジェクトをキャッチしてエラー情報を取得します。

サービスリクエスタ側の処理の流れを次に示します。

1. SOAPFaultException でキャッチします。
2. SOAPFaultException から getFault()メソッドを使用して SOAP Fault を取り出します。
3. SOAP Fault から SOAP フォルトの情報を取り出します。

HCSC サーバ内で発生した例外を取得する場合の，サービスリクエスタ側の実装の例を次に示します。

```
/**
 * Sample Program
 */
{
    try {
          :
    } catch (xxxxxxxxxxException_Exception e) {
       // WSDLに定義しているSOAPフォルトが返ってきた場合
      xxxxxxxxxxException faultInfo = e.getFaultInfo();
          :
    } catch (SOAPFaultException e) {
       // HCSCサーバ内で何らかのエラーが発生した場合
      SOAPFault soapFault = e.getFault();
      if(soapFault != null) {
         // SOAPフォルトの情報を出力
         System.err.println("faultCode=" + soapFault.getFaultCode());
         System.err.println("faultActor=" + soapFault.getFaultActor());
         System.err.println("faultString=" + soapFault.getFaultString());
         Detail detail = soapFault.getDetail();
         if(detail != null) {
           for(Iterator ite = detail.getDetailEntries(); ite.hasNext(); ) {
             printDetail((Element)ite.next());
           }
         }
      }
    }
}
```

なお，getDetail()メソッドを使用して取得した Element[]の構造は，次の表のようになります。

表 9-20　getDetail()メソッドを使用して取得した Element[]の構造

名称	内容
errorMessage	例外に設定されているエラーメッセージです。
errorCode	例外に設定されているエラーメッセージに対応するエラーコードです。
processInstanceID	ビジネスプロセスのプロセスインスタンス ID です。メッセージング基盤内で検出したエラーの場合は設定されていないため null（nil 属性）となります。ビジネスプロセスで発生したエラーの場合に該当します。

10 ビジネスプロセスのデバッグ

この章では，ビジネスプロセスのデバッグの流れ，および操作手順について説明します。

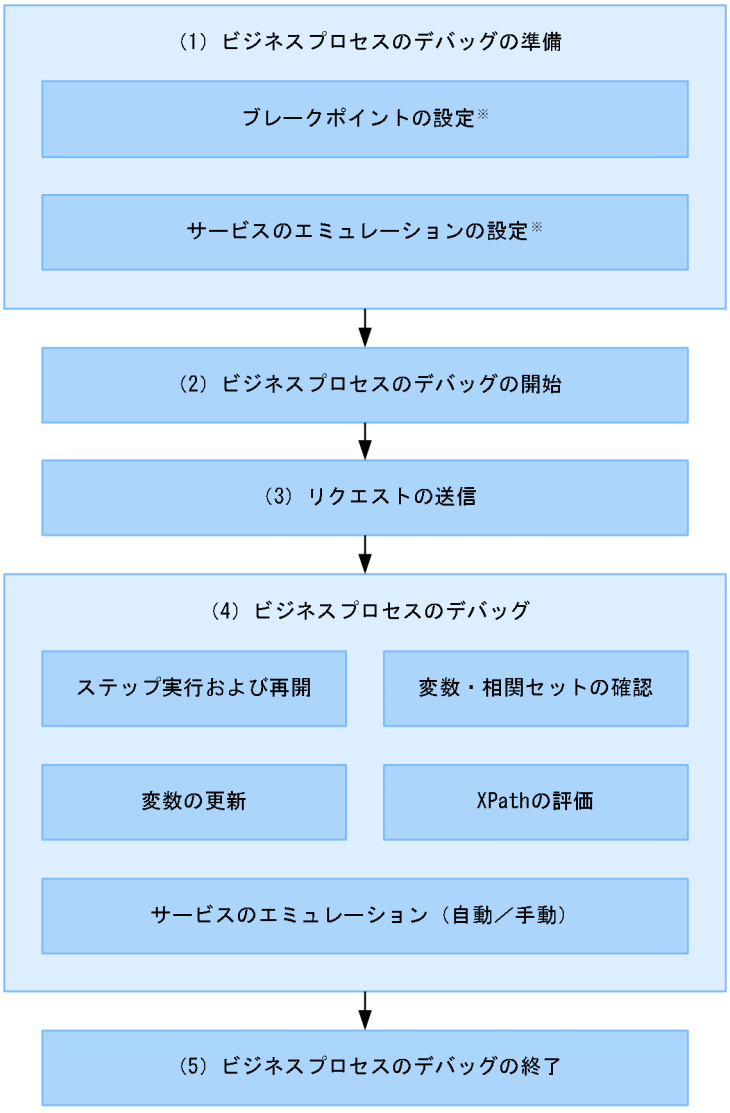
10.1　ビジネスプロセスのデバッグの手順

ここでは，ビジネスプロセスのデバッグを実行する手順について説明します。

10.1.1　デバッグの流れ

ビジネスプロセスのデバッグの流れを次の図に示します。

図 10-1　ビジネスプロセスのデバッグの流れ



注※
ビジネスプロセスのデバッグ中にも設定できます。

ビジネスプロセスのデバッグに関する作業について次に説明します。

(1)　ビジネスプロセスのデバッグの準備

ビジネスプロセスのデバッグの準備では，次の項目を実施します。

- ブレークポイントの設定

  任意のアクティビティでプロセスインスタンスの処理を中断したい場合，中断するアクティビティにブレークポイントを設定します。

  ブレークポイントの設定については，「10.2.1　ブレークポイントの設定」を参照してください。

- サービスのエミュレーションの設定

  作成済みの電文を実際にサービスを呼び出したときの応答電文の代わりに利用したい場合，サービスのエミュレーションを設定します。

  サービスのエミュレーションの設定については，「10.2.2　サービスのエミュレーションの設定」を参照してください。

(2)　ビジネスプロセスのデバッグの開始

ビジネスプロセスのデバッグを開始します。

ビジネスプロセスのデバッグの開始方法については，「10.3　ビジネスプロセスのデバッグの開始」を参照してください。

(3)　リクエストの送信

サービスリクエスタやサービスリクエスタのエミュレーションを使って，受付アクティビティにリクエストを送信します。リクエストの送信後，ブレークポイントが設定されているアクティビティまでプロセスインスタンスの処理が進むと，プロセスインスタンスの処理が自動で中断されます。

リクエストの送信については，「10.4　リクエストの送信」を参照してください。

(4)　ビジネスプロセスのデバッグ

ビジネスプロセスのデバッグでは，次の項目を実施できます。

- ステップ実行および再開

  プロセスインスタンスの処理をアクティビティ単位で中断しながら，逐次処理を進めることができます。プロセスインスタンスの処理が中断しているときに実施できます。

  ステップ実行および再開の方法については，「10.5.1　ステップ実行および再開」を参照してください。

- 変数・相関セットの確認

  ビジネスプロセスで使用している変数，および相関セットを確認します。プロセスインスタンスの処理が中断しているときに実施できます。

  変数・相関セットの確認方法については，「10.5.2　変数・相関セットの確認」を参照してください。

- 変数の更新

ビジネスプロセスで使用している変数の値を変更し，ビジネスプロセスに反映します。プロセスインスタンスの処理が中断しているときに実施できます。

変数の更新方法については，「10.5.3　変数の更新」を参照してください。

- XPath の評価

  分岐アクティビティや代入アクティビティに指定した条件式の妥当性を評価したり，変数の値の一部を参照したりできます。プロセスインスタンスの処理が中断しているときに実施できます。

  XPath の評価方法については，「10.5.4　XPath の評価」を参照してください。

- サービスのエミュレーション（自動／手動）

  設定した応答電文をサービスの応答電文の代わりに利用します。

  サービスの自動エミュレーションの実行方法については，「10.5.5　サービスの自動エミュレーション」を参照してください。サービスの手動エミュレーションの実行方法については，「10.5.6　サービスの手動エミュレーション」を参照してください。

(5)　ビジネスプロセスのデバッグの終了

ビジネスプロセスを定義し直す場合や，ビジネスプロセスの動作確認が終わった場合，ビジネスプロセスのデバッグを終了します。

ビジネスプロセスのデバッグの終了方法については，「10.6　ビジネスプロセスのデバッグの終了」を参照してください。

10.2　ビジネスプロセスのデバッグの準備

ビジネスプロセスのデバッグの準備では，ブレークポイントおよびサービスのエミュレーションを設定します。

なお，ブレークポイントおよびサービスのエミュレーションは，ビジネスプロセスのデバッグの開始後に設定することもできます。ビジネスプロセスのデバッグの開始後に設定する場合は，プロセスインスタンスの処理が中断しているときに実施してください。プロセスインスタンスの処理状態は，デバッグビューで確認できます。デバッグビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3　ビジネスプロセスのデバッグ画面の構成」を参照してください。

10.2.1　ブレークポイントの設定

任意のアクティビティでプロセスインスタンスの処理を中断したい場合，中断するアクティビティにブレークポイントを追加します。追加したブレークポイントは，必要に応じて条件設定を変更したり，エクスポートおよびインポートしたりします。

ブレークポイントを設定できるアクティビティの種類については，「10.5.1(1)　プロセスインスタンスの処理を中断できるアクティビティ」を参照してください。

(1)　ブレークポイントを追加する

ブレークポイントには，通常のブレークポイントと条件付きブレークポイントの 2 つがあります。通常のブレークポイントを追加した場合，アクティビティに処理が進んだときにプロセスインスタンスの処理が必ず中断されます。条件付きブレークポイントを追加した場合，アクティビティに処理が進んだときに，入力した条件式を満たしている（評価結果が真になる）と，プロセスインスタンスの処理が中断されます。

アクティビティに通常のブレークポイント，および条件付きブレークポイントを追加する手順を次に示します。

(a)　通常のブレークポイントの追加

1. ビジネスプロセス定義画面でブレークポイントを追加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。

   アクティビティにブレークポイントが追加され，アクティビティの横にブレークポイントを表すチェックが付きます。ビジネスプロセス定義画面の詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.2.3　ビジネスプロセス定義画面」を参照してください。

(b)　条件付きブレークポイントの追加

1. ビジネスプロセス定義画面でブレークポイントを追加するアクティビティを右クリックし，［条件付きブレークポイントの追加］を選択します。

   ［条件設定］ダイアログが表示されます。

2. 条件式を XPath で入力します。※

3. [OK] ボタンをクリックします。

アクティビティにブレークポイントが追加され，アクティビティの横にブレークポイントを表すチェックが付きます。

ブレークポイントを追加したアクティビティに処理が進んだときに，入力した XPath の条件式を満たしている（評価結果が真になる）と，プロセスインスタンスの処理が中断されます。

注※

条件式を入力しなかった場合は，通常のブレークポイントとして扱われます。

(2) ブレークポイントの条件設定を変更する

追加した通常のブレークポイントおよび条件付きブレークポイントは，必要に応じて中断の条件を変更できます。

ブレークポイントの条件設定を変更する手順を次に示します。

1. ビジネスプロセス定義画面でブレークポイントを追加したアクティビティを右クリックし，[条件の変更] を選択します。

[条件設定] ダイアログが表示されます。

2. 条件式を XPath で入力します。※

3. [OK] ボタンをクリックします。

ブレークポイントの条件設定が変更されます。

ブレークポイントの条件設定を変更したアクティビティに処理が進んだときに，入力した XPath の条件式を満たしている（評価結果が真になる）と，プロセスインスタンスの処理が中断されます。

注※

条件式を入力しなかった場合は，通常のブレークポイントとして扱われます。

(3) ブレークポイントを削除する

ビジネスプロセス定義画面でブレークポイントを削除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削除] を選択すると，対象のアクティビティのブレークポイントが削除されます。

(4) ブレークポイントを無効にする

Eclipse の標準機能を使用して，ブレークポイントビューから，アクティビティに追加したブレークポイントを一時的に無効にできます。

ブレークポイントビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3 ビジネスプロセスのデバッグ画面の構成」を参照してください。

(5)　ブレークポイントをエクスポート／インポートする

Eclipse の標準機能を使用して，アクティビティに追加したブレークポイントをエクスポートできます。また，エクスポートしたブレークポイントをインポートすることもできます。

エクスポート方法およびインポート方法の詳細については，Eclipse のドキュメントを参照してください。

注意事項

ブレークポイントをインポートする前に，すべてのアクティビティに設定されているブレークポイントを必ず削除してください。アクティビティにブレークポイントの設定が残っている場合，正常にインポートできないおそれがあります。

10.2.2　サービスのエミュレーションの設定

作成済みの応答電文を実際のサービスを呼び出す代わりに利用したい場合，サービスのエミュレーションを設定します。

サービスのエミュレーションを利用することで，ビジネスプロセスから呼び出すサービスがなくても，プロセスインスタンスの処理を進めることができます。

サービスのエミュレーションには，自動エミュレーションと手動エミュレーションの 2 つの機能があります。

サービスの自動エミュレーションとは，サービス呼出アクティビティからサービスを呼び出す前に，エミュレーションするサービスのオペレーションや，オペレーションの応答電文を設定しておくことで，プロセスインスタンスの処理がサービス呼出アクティビティに進んだときに，自動的にサービスのエミュレーションが実行される機能です。

サービスの手動エミュレーションとは，プロセスインスタンスの処理がサービス呼出アクティビティで中断しているときに，エミュレーションするサービスの応答電文を設定し，手動でサービスのエミュレーションを実行する機能です。

(1)　サービスの自動エミュレーションの設定

サービスの自動エミュレーションでは，エミュレーションするサービスのオペレーション，応答種別，応答電文などを設定します。

サービスの自動エミュレーションの設定手順については，「10.5.5　サービスの自動エミュレーション」を参照してください。

(2)　サービスの手動エミュレーションの設定

サービスの手動エミュレーションでは，プロセスインスタンスの処理がサービス呼出アクティビティで中断しているときに，応答種別および応答電文を設定します。

サービスの手動エミュレーションの設定手順については，「10.5.6　サービスの手動エミュレーション」を参照してください。

10.3　ビジネスプロセスのデバッグの開始

ビジネスプロセスのデバッグを開始する手順について説明します。

1. テスト環境を起動します。
   テスト環境を起動する方法については，「2.4.3(1)　テスト環境の起動と停止」を参照してください。

2. 標準受付を開始します。
   ビジネスプロセスに標準受付を使用していない場合，この手順は不要です。
   標準受付を開始する方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「5.3.10　標準受付を開始する」を参照してください。

3. 次のどれかの方法で，［デバッグ構成］ダイアログを表示します。
   - Eclipse のツールバーから［デバッグ（ )］－［デバッグの構成］を選択します。
   - ツリービューのサービス定義一覧からデバッグするビジネスプロセスを右クリックし，［デバッグの構成］を選択します。
   - パッケージ・エクスプローラーから対象のプロジェクトを右クリックし，［デバッグ］－［デバッグの構成］を選択します。
   - Eclipse のメニューから［実行］－［デバッグの構成］を選択します。

4. ［デバッグ構成］ダイアログのメニューから［HCSC-BP］を右クリックし，［新規］を選択します。

5. ［名前］にデバッグの名称を入力します。

6. ［ビジネスプロセス設定］タブの［デバッグ対象］のドロップダウンリストから，デバッグするビジネスプロセスを選択します。

7. ［デバッグ設定］タブを選択し，必要な情報を指定します。
   ［デバッグ設定］タブの表示・指定する内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.5.1　デバッグ構成ダイアログ」を参照してください。

8. ［デバッグ］ボタンをクリックします。
   デバッグの開始処理の経過がコンソールビューに表示されます。ビジネスプロセス，ユーザ定義受付，およびサービスアダプタが編集中の場合は，編集中の定義を保管するかを確認するダイアログが表示されます。
   ビジネスプロセスエディタが表示されます。ビジネスプロセスエディタの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3　ビジネスプロセスのデバッグ画面の構成」を参照してください。
   Eclipse 起動後，最初にビジネスプロセスのデバッグを開始するときは，［アカウント認証］ダイアログが表示されます。ユーザ ID とパスワードを入力して［OK］ボタンをクリックすると，ビジネスプロセスのデバッグの開始処理が開始されます。

9. [OK] ボタンをクリックします。

ビジネスプロセスのデバッグが開始されます。

一度ビジネスプロセスのデバッグを開始すると，Eclipse の［デバッグ］メニューにデバッグ構成のショートカットが作成されます。以降は，このショートカットを選択することで，手順 3～手順 8 の操作を省略してデバッグを開始できます。

注意事項

- 開発環境，運用環境，および実行環境は，同一のマシンに構築してください。複数マシンの構成にした場合，動作は保証できません。
- HCSC サーバはセットアップ済みの状態である必要があります。なお，セットアップは，単一の HCSC サーバ構成で実施してください。HCSC 簡易セットアップを利用する場合も，単一の HCSC サーバ構成でセットアップしてください。
- HCSC サーバおよびデータベースを起動しておいてください。
- ビジネスプロセスのデバッグを開始する場合，次のときにエラーになることがあります。

  ・開発環境と運用環境でリポジトリの構成形態（データベースと Reliable Messaging の組み合わせ）が不一致である。

  ・運用環境のリポジトリで配備済みのサービスが，開発環境で更新または削除されている。

  ・開発環境と運用環境のリポジトリ内で SOAP モードが不一致である。

  エラーになる場合は，定義したすべての HCSC コンポーネントを停止して HCSC サーバから削除したあと，再度ビジネスプロセスのデバッグを開始してください。
- ビジネスプロセスのデバッグの開始処理の途中でエラーが発生した場合，処理を中断して終了します。ただし，エラー発生前に終了している処理は，開始前の状態に戻りません。その場合，エラーの発生原因を取り除いて，再度開始してください。
- ビジネスプロセスのデバッグを開始したあとでビジネスプロセスやサービスアダプタを変更すると，デバッグの各機能が正常に動作しないおそれがあります。デバッグ中にビジネスプロセスやサービスアダプタを変更した場合は，ビジネスプロセスのデバッグをいったん終了してから再度デバッグを開始してください。
- ビジネスプロセスのデバッグは，単一のマシン内で開始してください。複数のマシン間でデバッグを開始した場合，動作を保証しません。
- ビジネスプロセスのデバッグの開始および終了操作は一度に 1 つしか実行できません。デバッグビューで複数のデバッグ構成を選択した状態で，デバッグの開始や終了操作をしないでください。
- ［デバッグ構成］ダイアログの［デバッグ対象］で選択したビジネスプロセスだけ，デバッグが開始されます。ビジネスプロセスから呼び出すほかのビジネスプロセスなど，複数のビジネスプロセスを同時にデバッグする場合は，それぞれビジネスプロセスのデバッグを開始してください。

- デバッグの開始に失敗すると，サービスが起動できない状態となり，HCSC コンポーネントを停止するまでビジネスプロセスのデバッグを開始できなくなることがあります。

  デバッグの開始に失敗したあと，デバッグの開始を再度実行しても失敗する場合は，HCSC コンポーネントを停止し，アンデプロイしてください。停止・アンデプロイの手順については，「8.6　HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行」を参照してください。

  なお，参照先の方法で停止・アンデプロイに失敗する場合は，コマンドを次の順に実行して HCSC コンポーネントを停止し，アンデプロイしてください。

  1. csccompostop -all コマンド

  csccompostop コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompostop（HCSC コンポーネントの停止)」を参照してください。

  2. csccompoundeploy -all コマンド

  csccompoundeploy コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「csccompoundeploy（配備済みの HCSC コンポーネントの削除)」を参照してください。
- 同一のビジネスプロセスを重複して開始することはできません。
- 最新のバージョンのビジネスプロセスでだけデバッグを開始できます。それ以外の場合，ビジネスプロセスのデバッグを実行できません。
- ビジネスプロセスのデバッグを開始したあとは，次の操作をしないでください。

  ・HCSCTE プロジェクトのクローズ

  ・HCSCTE プロジェクトの削除

  ・リポジトリディレクトリの変更

  ・リポジトリの初期化

  ・リポジトリのインポート

  ・サービスのインポート

  ・サービスのエクスポート

参考

ビジネスプロセスのデバッグを開始するときは，HCSC コンポーネントを HCSC サーバに配備して開始する処理が一括で実行されます。このとき，ビジネスプロセス，サービスアダプタ，およびユーザ定義の配備定義の処理は，［デバッグ構成］ダイアログの次に示す設定によって異なります。

- ［パッケージング済みビジネスプロセス］で，パッケージング済みのビジネスプロセスを再パッケージングする設定にしているかどうか
- ［サービス呼出アクティビティに設定されているサービス］で，サービス呼出アクティビティに設定されているサービスを自動的に配備・開始する設定にしているかどうか

詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.5.1　デバッグ構成ダイアログ」のデバッグ設定タブの説明を参照してください。

10.4　リクエストの送信

サービスリクエスタやサービスリクエスタのエミュレーションを使って，受付アクティビティにリクエストを送信します。

リクエストを送信すると，パースペクティブを切り替えるかどうかを確認するダイアログが表示されます。[はい] ボタンをクリックすると，[HCSCTE] パースペクティブから［デバッグ］パースペクティブに切り替わり，デバッグビュー，変数ビューなどのビューが表示されます。

リクエストの送信後，プロセスインスタンスの処理が開始され，ブレークポイントを設定しているアクティビティでプロセスインスタンスの処理が中断されます。

10.5　ビジネスプロセスのデバッグで実施できる操作

ビジネスプロセスのデバッグで実施できる操作について説明します。

ビジネスプロセスのデバッグの各操作は，プロセスインスタンスの処理が中断しているときに行ってください。プロセスインスタンスの処理状態は，デバッグビューで確認できます。

10.5.1　ステップ実行および再開

プロセスインスタンスの処理を中断できるアクティビティの種類と，ステップ実行および再開の操作について説明します。

(1)　プロセスインスタンスの処理を中断できるアクティビティ

プロセスインスタンスの処理を中断できるアクティビティの種類を次の表に示します。

表 10–1　プロセスインスタンスの処理を中断できるアクティビティの種類

アクティビティの分類	アクティビティ	中断の可否
基本アクティビティ	開始アクティビティ	×
	受付アクティビティ	○
	応答アクティビティ	○
	サービス呼出アクティビティ	○
	Java 呼出アクティビティ	○
	データ変換アクティビティ	○
	代入アクティビティ	○
	無操作アクティビティ	○
	フォルト送出アクティビティ	○
	補償アクティビティ	○
	待機アクティビティ	○
	検証アクティビティ	○
	ログ出力アクティビティ	○
	途中終了アクティビティ	○
	終了アクティビティ	×
構造アクティビティ	スコープアクティビティ	○
	繰り返しアクティビティ	○

アクティビティの分類	アクティビティ	中断の可否
構造アクティビティ	分岐開始アクティビティ	○
	分岐終了アクティビティ	×
	並列処理開始アクティビティ	○
	並列処理終了アクティビティ	×

（凡例）

○：中断できます。

×：中断できません。

(2)　ステップ実行および再開の操作

プロセスインスタンスをステップ実行，または再開するには，デバッグビューで中断しているアクティビティを選択し，実行したい操作のアイコンをクリックします。

ステップ実行および再開の操作と，デバッグの動作を次の表に示します。

表 10–2　ステップ実行および再開の操作と，デバッグの動作

ステップ実行の操作	アイコン	デバッグの動作
ステップイン		ステップインが実行されます。アクティビティごとにステップインを実行したときに中断するアクティビティについては，「(3)　ステップインを実行したときに中断するアクティビティ」を参照してください。
ステップオーバー		ステップオーバーが実行されます。アクティビティごとにステップオーバーを実行したときに中断するアクティビティについては，「(4)　ステップオーバーを実行したときに中断するアクティビティ」を参照してください。
ステップリターン		ステップリターンが実行されます。アクティビティごとにステップリターンを実行したときに中断するアクティビティについては，「(5)　ステップリターンを実行したときに中断するアクティビティ」を参照してください。
再開		次にブレークポイントを設定しているアクティビティまで，プロセスインスタンスの処理が再開されます。

なお，サービス呼出アクティビティからフォルト処理に移る場合，フォルト処理でプロセスインスタンスの処理を中断させるには，フォルト処理を行うアクティビティにブレークポイントを設定するか，ステップインを実行する必要があります。

(3)　ステップインを実行したときに中断するアクティビティ

アクティビティごとにステップインを実行したときに中断するアクティビティを次の表および図に示します。

表 10-3　アクティビティごとにステップインを実行したときに中断するアクティビティ

アクティビティの分類	アクティビティ	説明
基本アクティビティ（途中終了アクティビティ，補償アクティビティ以外）		次のアクティビティに進みます。
基本アクティビティ（途中終了アクティビティ）		ビジネスプロセスを終了します。
基本アクティビティ（補償アクティビティ）		補償アクティビティが実行する補償処理の最初のアクティビティに進みます。補償処理を行うアクティビティが存在しない場合は，補償アクティビティの次のアクティビティに進みます。
構造アクティビティ	スコープアクティビティ	スコープアクティビティ内で最初に処理されるアクティビティに進みます。
	繰り返しアクティビティ	繰り返しアクティビティ内で最初に処理されるアクティビティに進みます。
	分岐開始アクティビティ	分岐開始アクティビティから分岐終了アクティビティまでの間で，最初に処理されるアクティビティに進みます。
	並列処理開始アクティビティ	並列処理開始アクティビティから並列処理終了アクティビティまでの間で，最初に処理されるアクティビティに進みます。

図 10-2　基本アクティビティ（補償アクティビティ以外）でステップインを実行したときに中断するアクティビティ

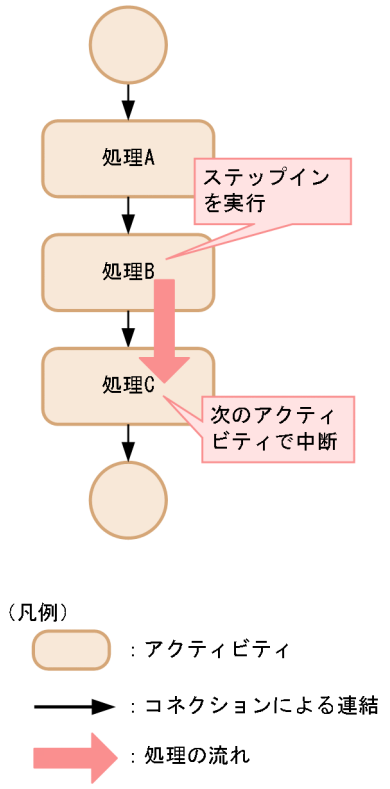


図 10-3　基本アクティビティ（補償アクティビティ）でステップインを実行したときに中断するアクティビティ

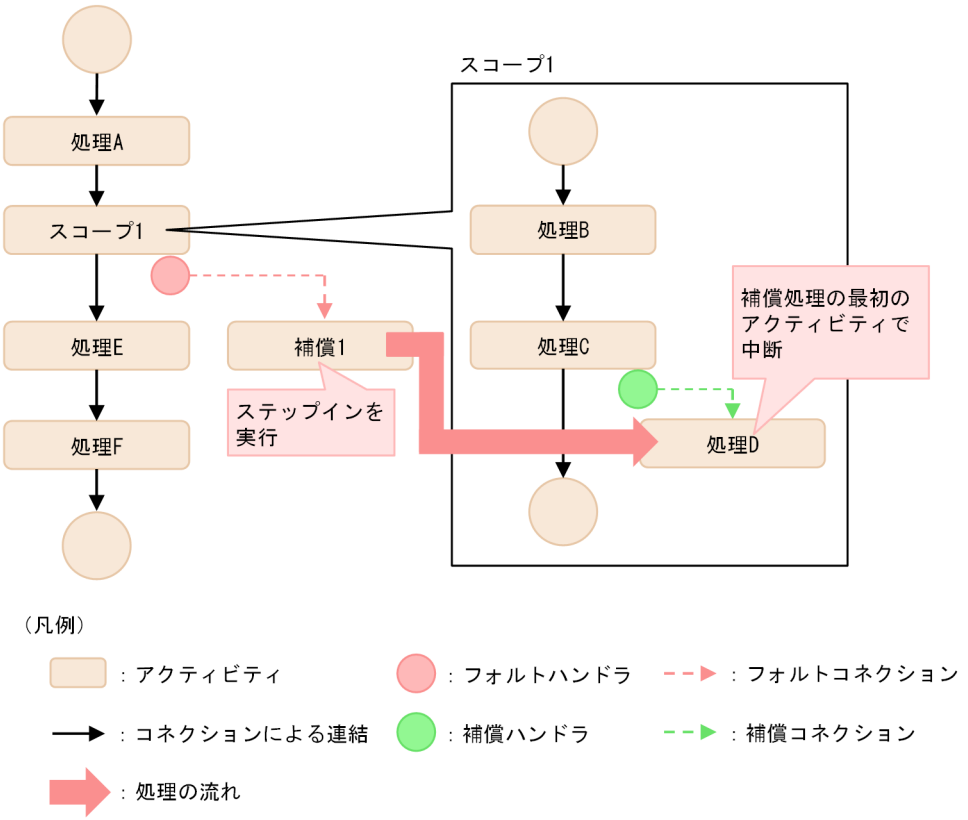


図 10-4　構造アクティビティ（スコープ，繰り返し）でステップインを実行したときに中断するアクティビティ

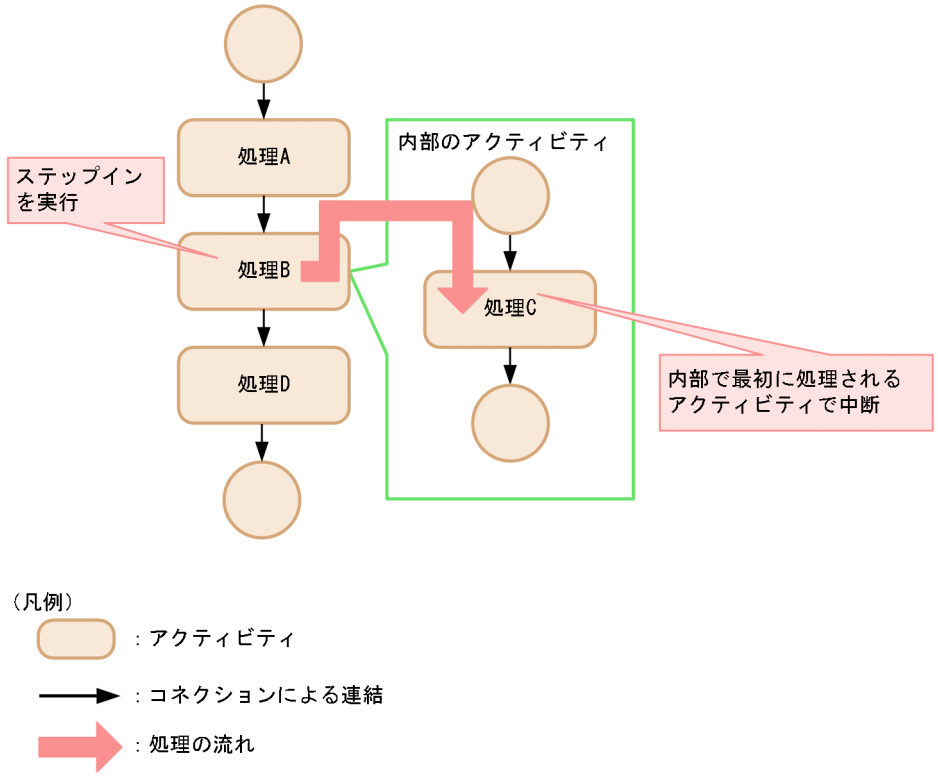
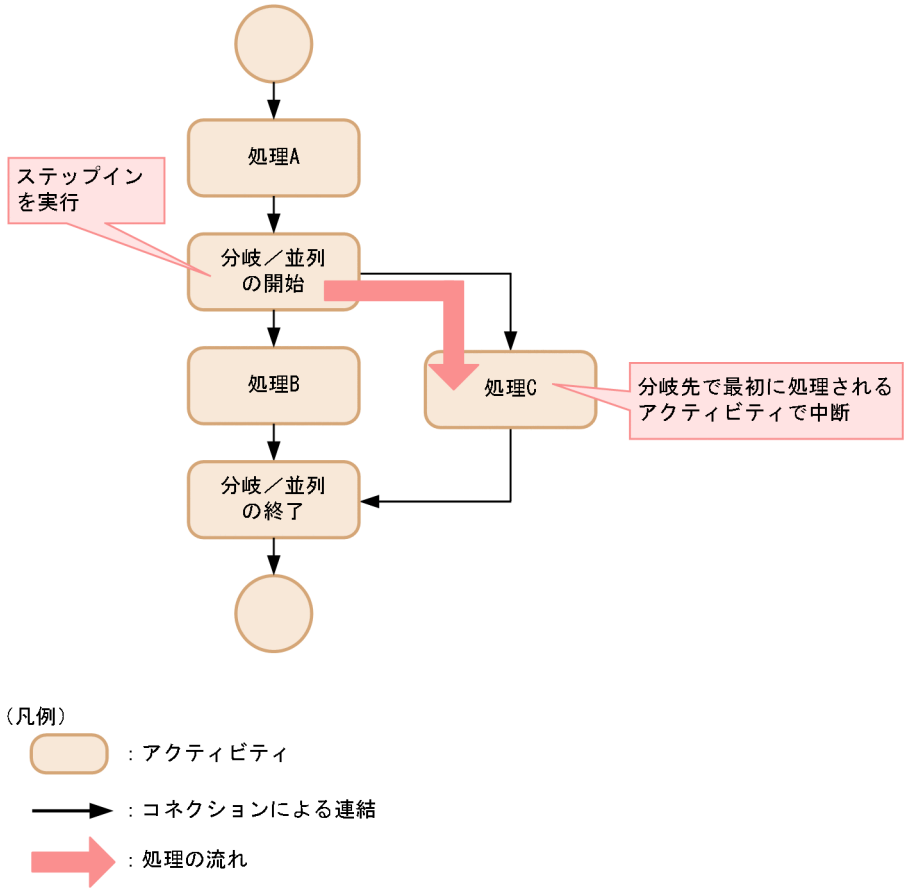


図 10-5　構造アクティビティ（分岐，並列）でステップインを実行したときに中断するアクティビティ



(4)　ステップオーバーを実行したときに中断するアクティビティ

アクティビティごとにステップオーバーを実行したときに中断するアクティビティを次の表および図に示します。

表 10-4　アクティビティごとにステップオーバーを実行したときに中断するアクティビティ

アクティビティの分類	アクティビティ	説明
基本アクティビティ（途中終了アクティビティ，補償アクティビティ以外）		次のアクティビティに進みます。
基本アクティビティ（途中終了アクティビティ）		ビジネスプロセスを終了します。
基本アクティビティ（補償アクティビティ）		補償処理として定義したアクティビティを処理して次のアクティビティに進みます。補償アクティビティが実行する補償処理内にブレークポイントが設定されている場合は，そこで中断します。
構造アクティビティ	スコープアクティビティ	スコープアクティビティの内部のアクティビティを処理して次のアクティビティに進みます。途中でブレークポイントが設定されている場合は，そこで中断します。

アクティビティの分類	アクティビティ	説明
構造アクティビティ	繰り返しアクティビティ	繰り返しアクティビティの内部のアクティビティを処理して次のアクティビティに進みます。途中でブレークポイントが設定されている場合は，そこで中断します。
	分岐開始アクティビティ	分岐終了アクティビティの次のアクティビティまで処理します。途中でブレークポイントが設定されている場合は，そこで中断します。
	並列処理開始アクティビティ	並列処理終了アクティビティの次のアクティビティまで処理します。途中でブレークポイントが設定されている場合は，そこで中断します。

図 10-6　基本アクティビティ（補償アクティビティ以外）でステップオーバーを実行したときに中断するアクティビティ

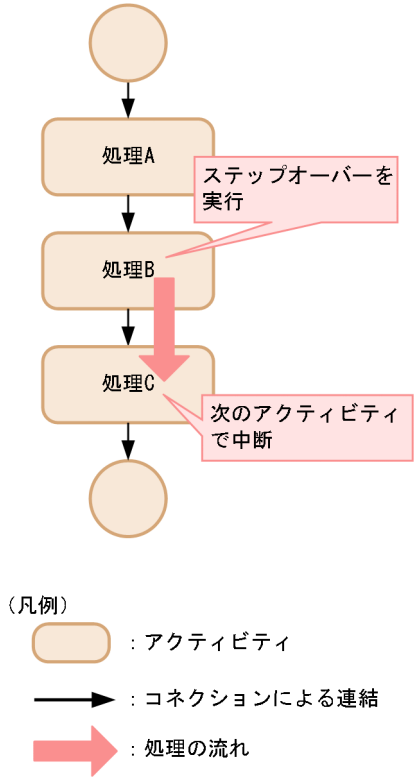


図 10-7　基本アクティビティ（補償アクティビティ）でステップオーバーを実行したときに中断するアクティビティ

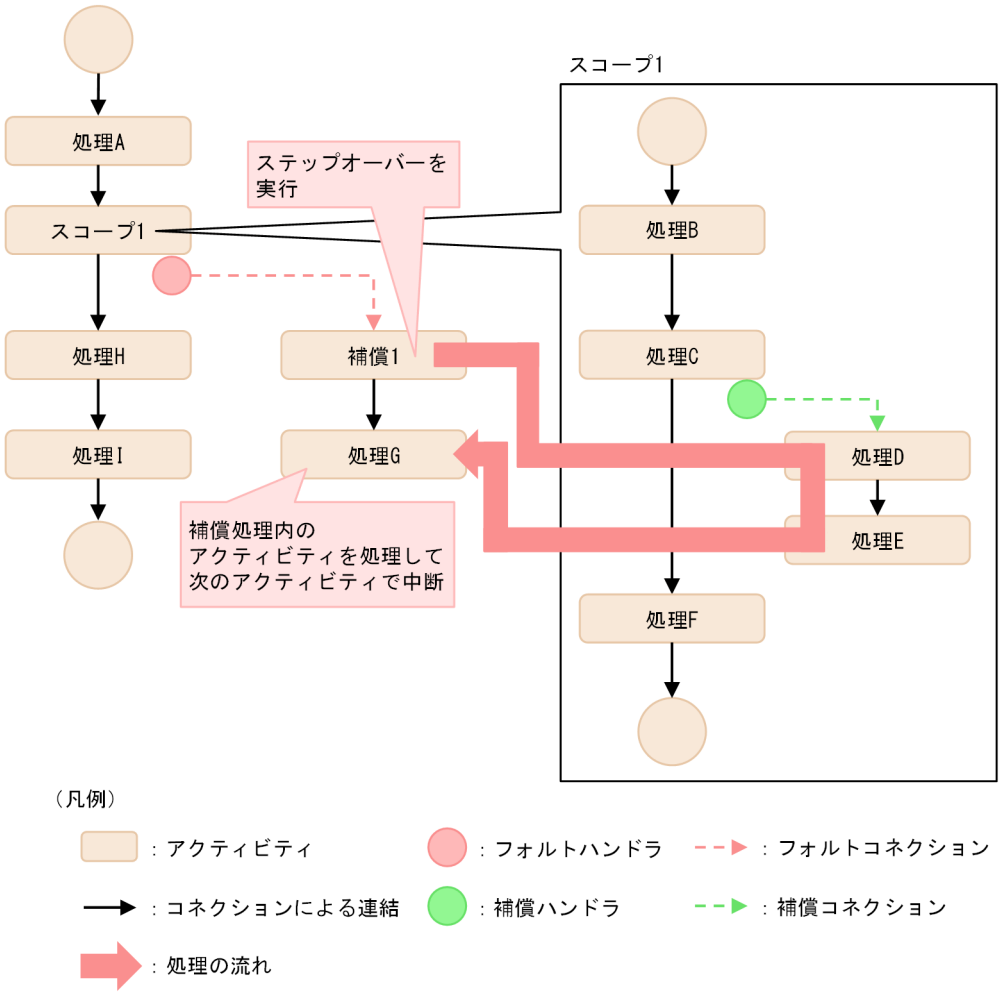


図 10-8　構造アクティビティ（スコープ，繰り返し）でステップオーバーを実行したときに中断するアクティビティ

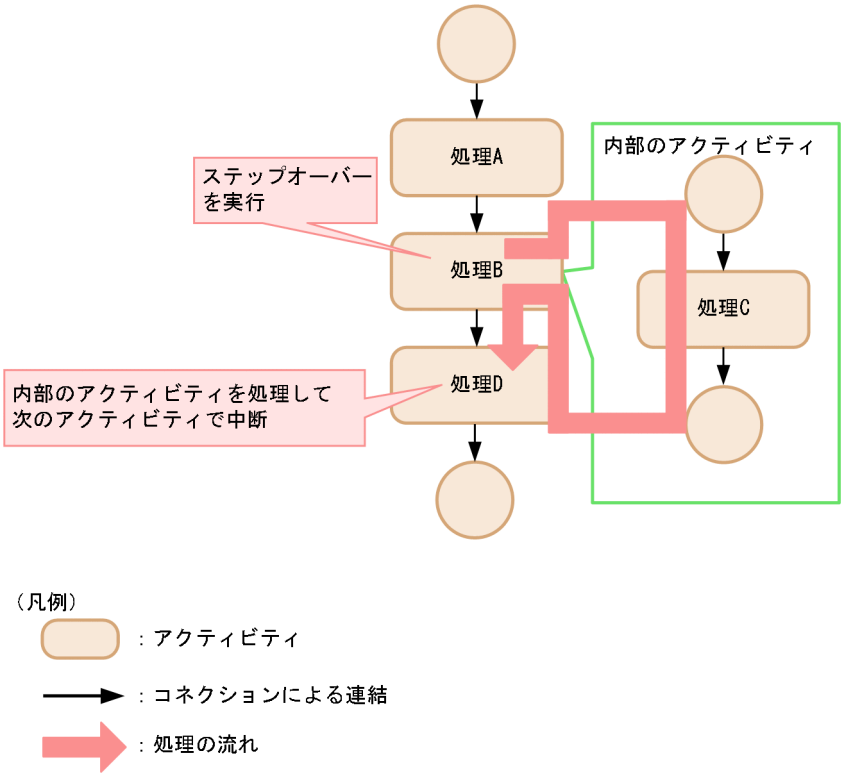
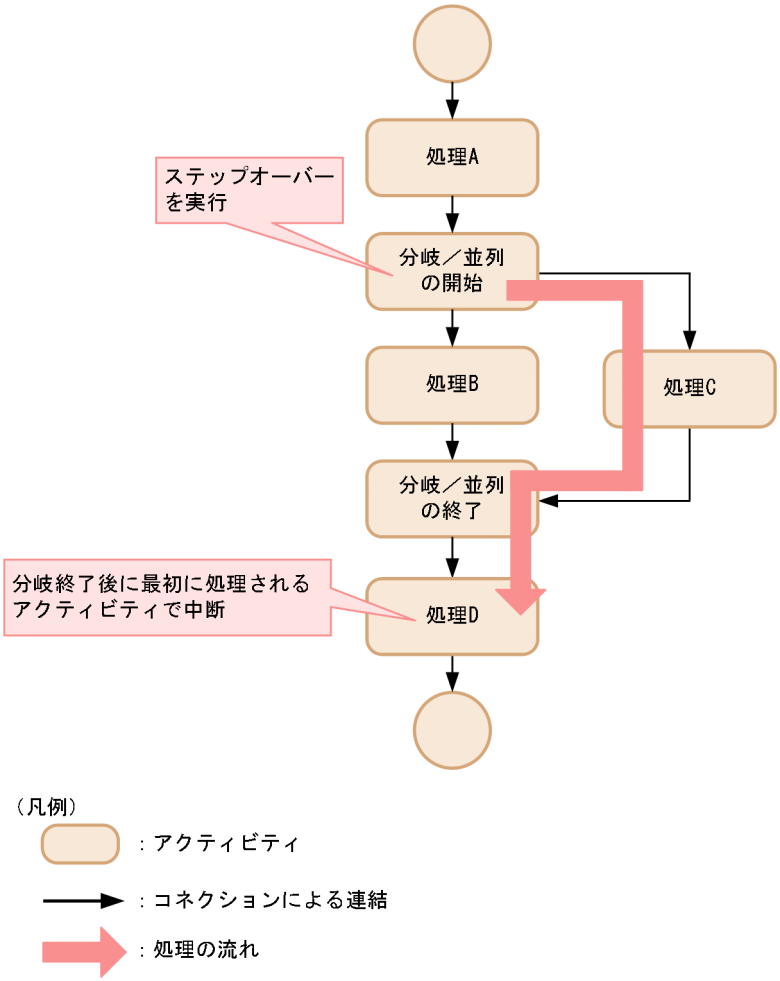


図 10-9　構造アクティビティ（分岐，並列）でステップオーバーを実行したときに中断するアクティビティ



(5)　ステップリターンを実行したときに中断するアクティビティ

アクティビティごとにステップリターンを実行したときに中断するアクティビティについて説明します。

- 構造アクティビティ内の基本アクティビティで実行した場合は，所属する構造アクティビティの次のアクティビティまで処理します。
- 所属する構造アクティビティが，スコープアクティビティまたは繰り返しアクティビティの場合は，内部のアクティビティをすべて処理して，次のアクティビティに進みます。
- 所属する構造アクティビティが分岐開始アクティビティまたは並列処理開始アクティビティの場合は，対となる分岐終了アクティビティ，または並列処理終了アクティビティの次の位置まで処理します。
- フォルト処理内のアクティビティで実行した場合は，フォルトコネクションの接続元アクティビティの次のアクティビティまで処理します。
- 補償処理内のアクティビティで実行した場合は，補償処理を実行している補償アクティビティの次のアクティビティまで処理します。
- 中断しているアクティビティが次の条件にすべて当てはまる場合は，再開と同じ動作をします。

- 構造アクティビティ内にない
- フォルト処理内にない
- 補償処理内にない

- 途中でブレークポイントが設定されている場合は，そこで中断します。
- 途中終了アクティビティを実行した時点で，ビジネスプロセスを終了します。

構造アクティビティ内の各アクティビティでステップリターンを実行したときに中断するアクティビティを次の図に示します。

図 10–10　構造アクティビティ内の基本アクティビティでステップリターンを実行したときに中断するアクティビティ

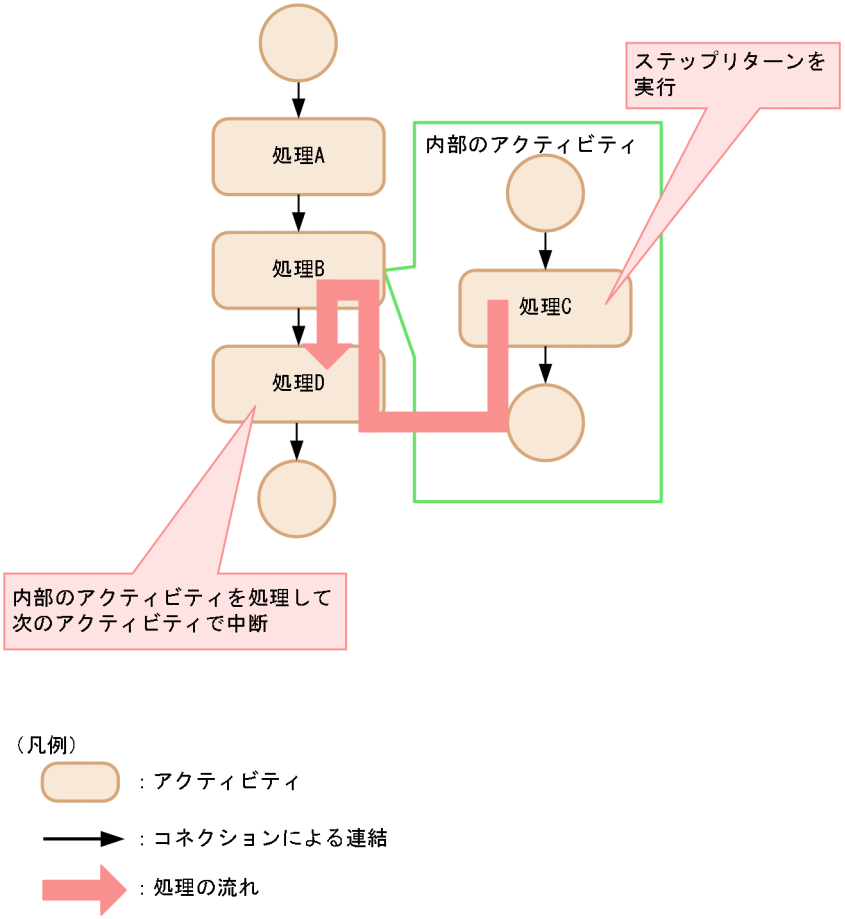


図 10-11　構造アクティビティ内の構造アクティビティ（スコープ，繰り返し）でステップリターンを実行したときに中断するアクティビティ

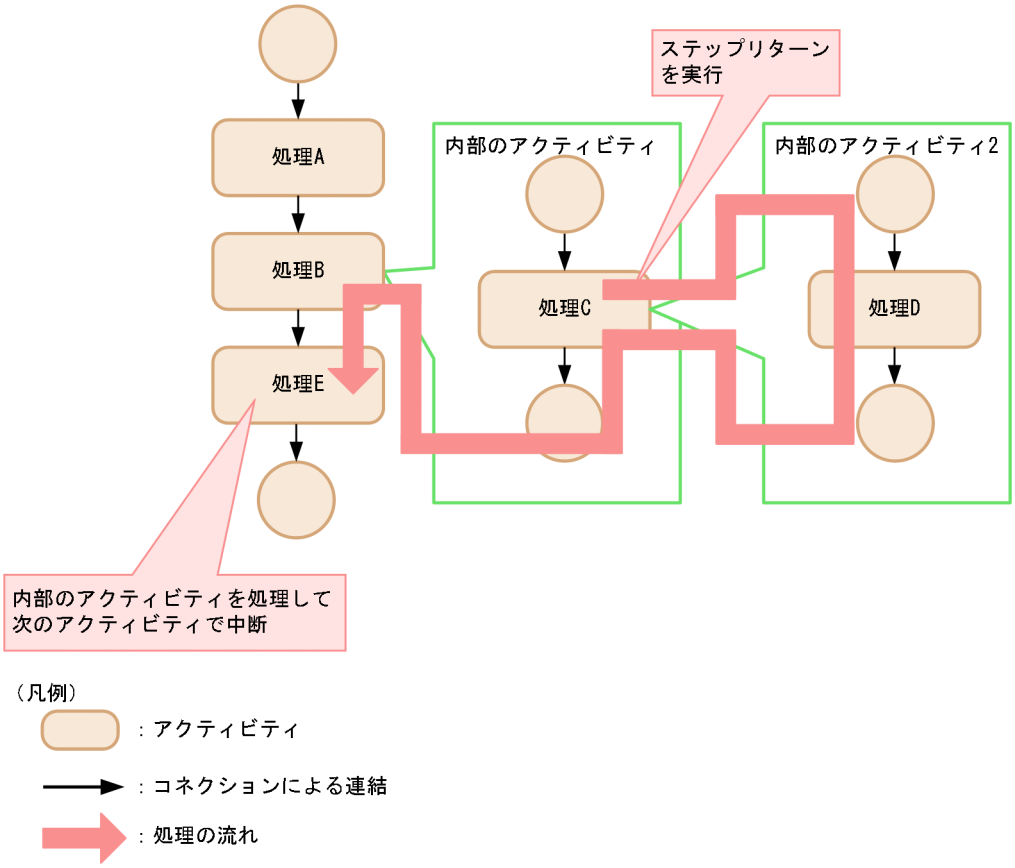


図 10-12　構造アクティビティ内の構造アクティビティ（分岐，並列）でステップリターンを実行したときに中断するアクティビティ

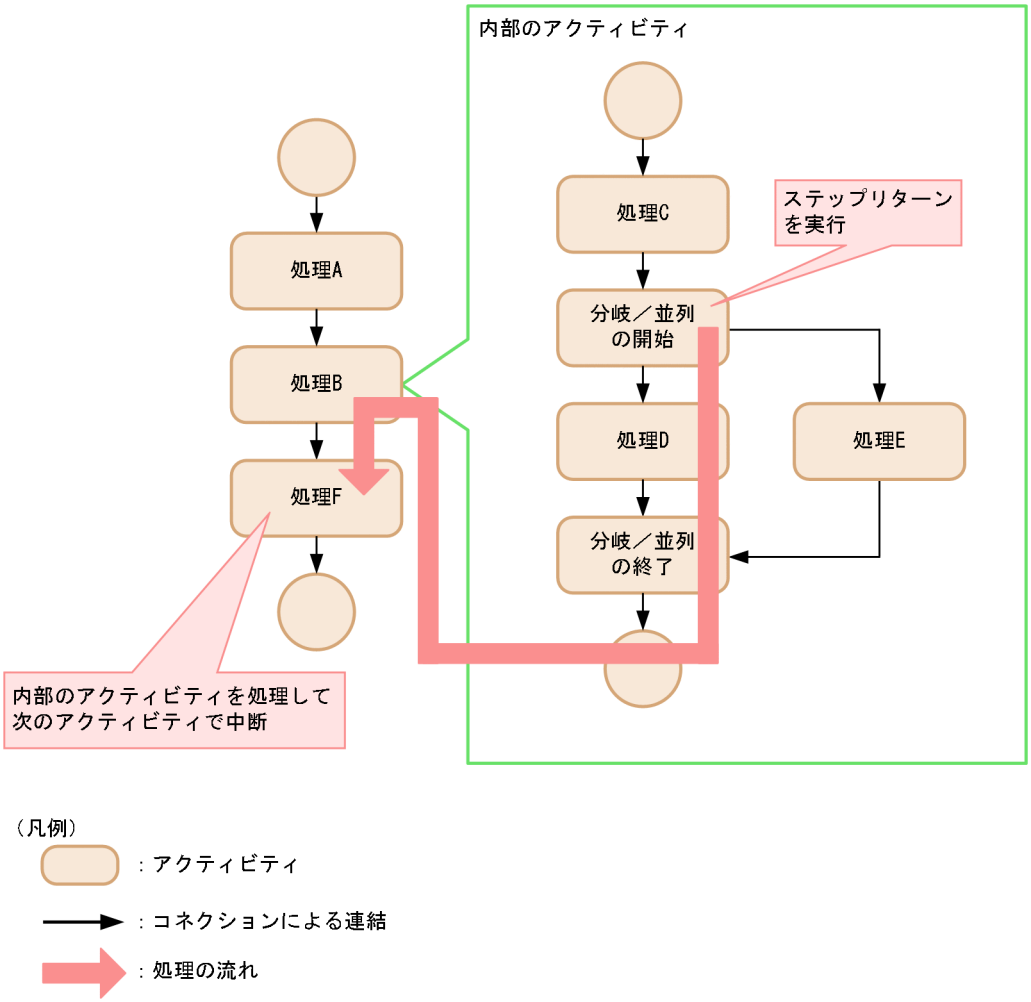


図 10-13　フォルト処理内のアクティビティでステップリターンを実行したときに中断するアクティビティ

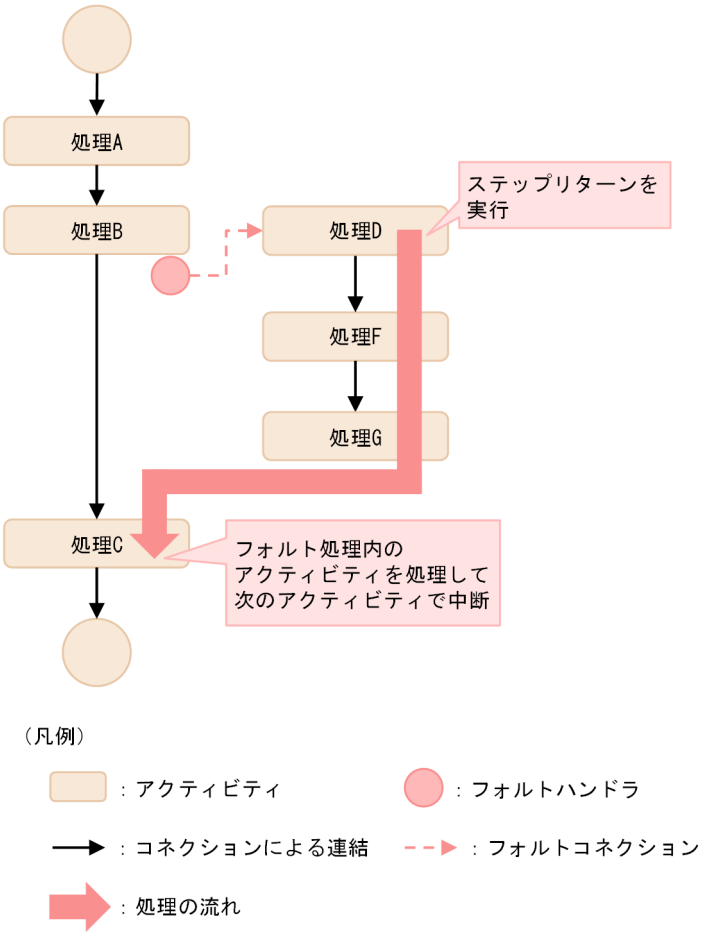
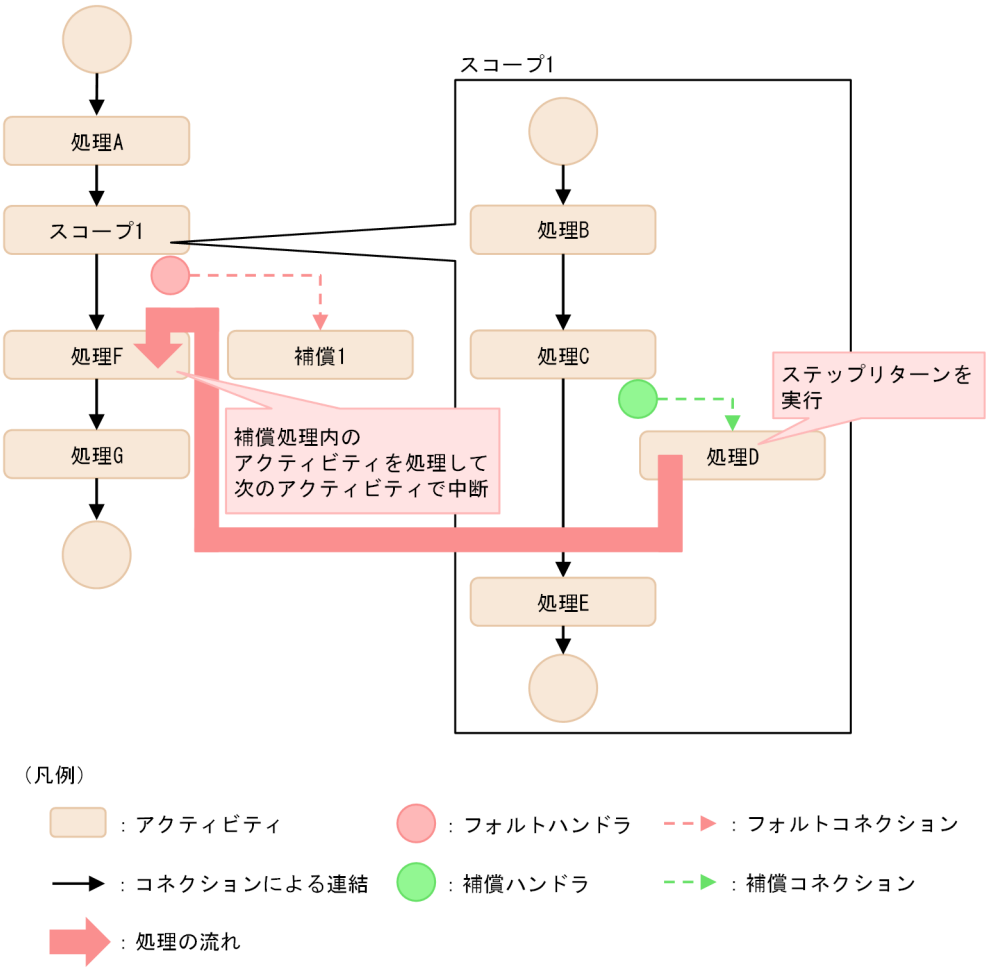


図 10-14　補償処理内のアクティビティでステップリターンを実行したときに中断するアクティビティ



10.5.2　変数・相関セットの確認

プロセスインスタンスの処理が中断しているときに，デバッグビューで中断しているアクティビティを選択すると，ビジネスプロセスで現在使用している変数名，変数の値，相関セット名，および相関 ID が変数ビューに表示されます。

変数ビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3　ビジネスプロセスのデバッグ画面の構成」を参照してください。

10.5.3　変数の更新

変数ビューに表示された変数の値を変更し，ビジネスプロセスに反映します。

変数の型ごとに入力できる値の範囲を次の表に示します。

表 10–5　変数の型ごとに入力できる値の範囲

変数の型	値の範囲
numeric 型	java.lang.Double の valueOf（java.lang.String）メソッドで解釈できる値（例：+1，3.14，1e-2d）
boolean 型	true または false
string 型	任意の文字列（例：15 インチ LCD ディスプレイ）
メッセージ型※1	• XML 型の場合 任意の XML（例：<?xml version="1.0" encoding="UTF-8"?><message>Hello</message>） • non-XML 型の場合※2 任意のバイナリデータ（例：Hello） • any 型の場合※2 任意の形式

注※1

メッセージ型の変数を更新すると，XML の文字コードは必ず UTF-8 になります（UTF-8 以外の文字コードを指定しても UTF-8 に変換されます）。

注※2

変数の型が non-XML 型および any 型の場合，値は 16 進数表記で入力してください。

変数の更新には，［値の設定］ダイアログで更新する方法，変数ビューの入力フィールドから更新する方法，および変数ビューの［値］セルから更新する方法があります。

変数を更新する手順を次に示します。

(1)　［値の設定］ダイアログで更新する場合

1. 変数ビューから更新する変数を右クリックし，［値の変更］を選択します。

   ［値の設定］ダイアログが表示されます。［値の設定］ダイアログに表示される内容は，メッセージ型の場合とメッセージ型以外の場合とで異なります。それぞれの場合の［値の設定］ダイアログの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.5.3　値の設定ダイアログ」を参照してください。

2. ［値の設定］ダイアログの入力フィールドに値を入力します。

   変数の型がメッセージ型の場合，［保存］ボタンをクリックすることで，入力中の変数の値をファイルに保存できます。また，［読み込み］ボタンをクリックすることで，ファイルに保存した変数の値を入力フィールドに読み込めます。

3. ［OK］ボタンをクリックします。

   入力した変数の値がビジネスプロセスに反映されます。更新した変数のセルが黄色で表示されます。

(2)　変数ビューの入力フィールドから更新する場合

1. 変数ビューから更新する変数を選択します。

変数ビューの下部の入力フィールドに，選択した変数の値が表示されます。

2. 入力フィールドに値を入力します。

3. 次のどちらかの方法で変数を更新します。

   - ［Ctrl］キー＋［S］キーを押す
   - 右クリックし，［値の代入］を選択する

   入力した変数の値がビジネスプロセスに反映されます。更新した変数のセルが黄色で表示されます。

(3) 変数ビューの［値］セルから更新する場合

1. 変数ビューから，更新する変数の［値］セルを選択します。

2. ［値］セルに値を入力します。

3. ［Enter］キーを押します。
   入力した変数の値がビジネスプロセスに反映されます。更新した変数のセルが黄色で表示されます。

10.5.4 XPath の評価

分岐アクティビティや代入アクティビティに指定した条件式の妥当性を評価したり，変数の値の一部を参照したりできます。プロセスインスタンスの処理が中断しているときに実行できます。

注意事項

ビジネスプロセスのデバッグで XPath を評価する場合，改行を含むデータの評価はしないでください。改行を含むデータの評価が必要な場合は，XPath 関数の normalize-space()を使用して，改行を単一の空白に置き換えてから評価してください。

XPath の評価の手順を次に示します。

1. Eclipse のメニューから，［ウィンドウ］－［ビューの表示］－［その他］を選択します。
   ［ビューの表示］ダイアログが表示されます。

2. ［デバッグ］－［HCSC XPath 評価］を選択し，［開く］ボタンをクリックします。
   HCSC XPath 評価ビューが表示されます。HCSC XPath 評価ビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3 ビジネスプロセスのデバッグ画面の構成」を参照してください。

3. デバッグビューで，条件式を評価するビジネスプロセスのアクティビティを選択します。

4. HCSC XPath 評価ビューの入力フィールドに XPath 式を入力し，［評価］ボタンをクリックします。

XPath 式に対する評価結果が入力フィールドの下に表示されます。

10.5.5　サービスの自動エミュレーション

サービスの自動エミュレーションの設定，および実行手順について説明します。

参考

エミュレーションに必要な応答電文がない場合は，cscgenbinary コマンドで作成できます。cscgenbinary コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscgenbinary（バイナリフォーマット定義ファイルからバイナリ電文の生成)」を参照してください。

(1)　サービスの自動エミュレーションの設定

1. Eclipse のメニューから，［ウィンドウ］－［ビューの表示］－［その他］を選択します。
   ［ビューの表示］ダイアログが表示されます。

2. ［デバッグ］－［HCSC 自動エミュレート］を選択し，［開く］ボタンをクリックします。
   HCSC 自動エミュレートビューが表示されます。
   HCSC 自動エミュレートビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3　ビジネスプロセスのデバッグ画面の構成」を参照してください。

3. HCSC 自動エミュレートビューの［追加］ボタンをクリックします。
   HCSC 自動エミュレートビューのテーブルに行が追加されます。

4. ［サービス名］のセルをクリックし，ドロップダウンリストからエミュレーションしたいサービスを選択します。

5. ［オペレーション名］のセルをクリックし，ドロップダウンリストからエミュレーションしたいサービスのオペレーションを選択します。

6. ［応答種別］のセルを選択し，ドロップダウンリストから通常応答またはフォルト応答を選択します。

7. ［応答電文］のセルを選択し，［...］ボタンをクリックします。
   ［ファイル選択］ダイアログが表示されます。

8. 応答電文のファイルを選択し，［OK］ボタンをクリックします。※
   ファイルの内容が応答電文としてそのまま使用されます。
   ただし，拡張子が.xsl のファイルを指定した場合，指定したスタイルシートファイルで要求電文を変換した結果が応答電文として使用されます。拡張子が.xsl のファイルは，要求電文の変数種別と応答電文の変数種別がともに XML のときだけ使用できます。

注※

応答電文に XML 形式のデータを指定する場合は，XML の文字コードが UTF-8 のものしか指定できません。

9. 必要に応じて，［条件］のセルを選択し，XPath 式で条件を入力します。

サービスの要求電文に対して条件式を評価した結果が真の場合に，サービスの自動エミュレーションが実行されます。条件を入力しなかった場合は，常に真として評価されます。

条件は，要求電文の変数種別が次の場合だけ設定できます。

- XML
- non-XML

条件式を入力するときは，次の点に注意してください。

- サービスの要求電文に対して条件を評価するため，変数名の指定はできません。
- 空文字（0x00）および csc:getVariableData などの拡張関数は使用できません。
- 条件式を指定しなかった場合は，常に真として評価されます。

10. ［有効］のセルのチェックボックスをチェックします。

サービスの自動エミュレーションが設定されます。

HCSC 自動エミュレートビューの設定内容をエミュレート設定情報定義ファイルに保存したい場合は，HCSC 自動エミュレートビューで［保存］ボタンをクリックします。

また，［読み込み］ボタンをクリックすることで，エミュレート設定情報定義ファイルに保存した設定内容を取得できます。すでに HCSC 自動エミュレートビューに設定内容がある場合は，HCSC 自動エミュレートビューの設定内容を上書きしてもよいか確認するダイアログが表示されます。

HCSC 自動エミュレートビューに同一のサービスとオペレーションの組み合わせを複数設定した場合，優先順位が最も高いサービスのエミュレーションが実行されます。サービスの優先順位を変更するには，変更したい行を選択し，［上へ］ボタンまたは［下へ］ボタンをクリックします。

応答電文のファイル名を右クリックし，［ファイルの表示］を選択すると，対象のファイルを開くことができます。

複数のビジネスプロセスをデバッグする場合，自動エミュレーションの設定（サービス名，オペレーション名）は，すべてのビジネスプロセスで有効になります。

なお，自動エミュレーションの対象となるサービス呼出アクティビティに対して手動エミュレーションを実行した場合，手動エミュレーションの実行が優先されます。

(2) サービスの自動エミュレーションの実行

HCSC 自動エミュレートビューに設定したサービス呼出アクティビティにプロセスインスタンスの処理が進むと，設定した応答電文がサービスの応答電文の代わりに自動的に利用されます。

XML 形式，またはバイナリ形式のファイルを応答電文に指定した場合，ファイルの内容がそのままサービスの応答電文としてエミュレーションされます。XSL 形式のファイルを応答電文に指定した場合，サービスの要求電文をスタイルシートファイルで変換した結果がサービスの応答電文としてエミュレーションされます。

なお，サービスの自動エミュレーションを設定していない場合は，通常のサービス呼び出しが実行されます。

注意事項

HCSC 自動エミュレートビューを閉じている場合，サービスの自動エミュレーションは実行されません。

10.5.6　サービスの手動エミュレーション

サービスの手動エミュレーションの設定，および実行手順について説明します。

参考

エミュレーションに必要な応答電文がない場合は，cscgenbinary コマンドで作成できます。cscgenbinary コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscgenbinary（バイナリフォーマット定義ファイルからバイナリ電文の生成)」を参照してください。

(1)　サービスの手動エミュレーションの設定

1. Eclipse のメニューから，［ウィンドウ］－［ビューの表示］－［その他］を選択します。
   ［ビューの表示］ダイアログが表示されます。

2. ［デバッグ］－［HCSC エミュレート］を選択し，［開く］ボタンをクリックします。
   HCSC エミュレートビューが表示されます。
   HCSC エミュレートビューの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「1.1.3　ビジネスプロセスのデバッグ画面の構成」を参照してください。

3. デバッグビューで手動エミュレーションするアクティビティを選択します。

4. HCSC エミュレートビューの［追加］ボタンをクリックします。
   HCSC エミュレートビューのテーブルに行が追加されます。

5. ［応答種別］のセルを選択し，ドロップダウンリストから通常応答またはフォルト応答を選択します。

6. ［応答電文］のセルを選択し，［...］ボタンをクリックします。
   ［ファイル選択］ダイアログが表示されます。

7. 応答電文の XML ファイル（XML 形式の場合）※，または任意形式のファイル（non-XML 形式および any 形式の場合）を選択し，［OK］ボタンをクリックします。

   サービスのエミュレーションが設定されます。

   注※

   XML の文字コードが UTF-8 のものしか指定できません。

HCSC エミュレートビューの設定内容をエミュレート設定情報定義ファイルに保存したい場合は，HCSC エミュレートビューで［保存］ボタンをクリックします。

また，［読み込み］ボタンをクリックすることで，エミュレート設定情報定義ファイルに保存した設定内容を取得できます。すでに HCSC エミュレートビューに設定内容がある場合は，HCSC エミュレートビューの設定内容を上書きしてもよいか確認するダイアログが表示されます。

応答電文のファイル名を右クリックし，［ファイルの表示］を選択すると，対象のファイルを開くことができます。

(2) サービスの手動エミュレーションの実行

プロセスインスタンスの処理がサービス呼出アクティビティで中断しているときに，HCSC エミュレートビューの［再開］ボタンまたは［ステップ実行］ボタンをクリックすると，設定した応答電文がサービスの応答電文の代わりに利用されます。

なお，フォルト応答をエミュレーションしてサービス呼出アクティビティからフォルト処理に移る場合，フォルト処理でプロセスインスタンスの処理を中断させるには，フォルト処理を行うアクティビティにブレークポイントを設定する必要があります。

10.5.7 cscgenbinary コマンドの使用方法

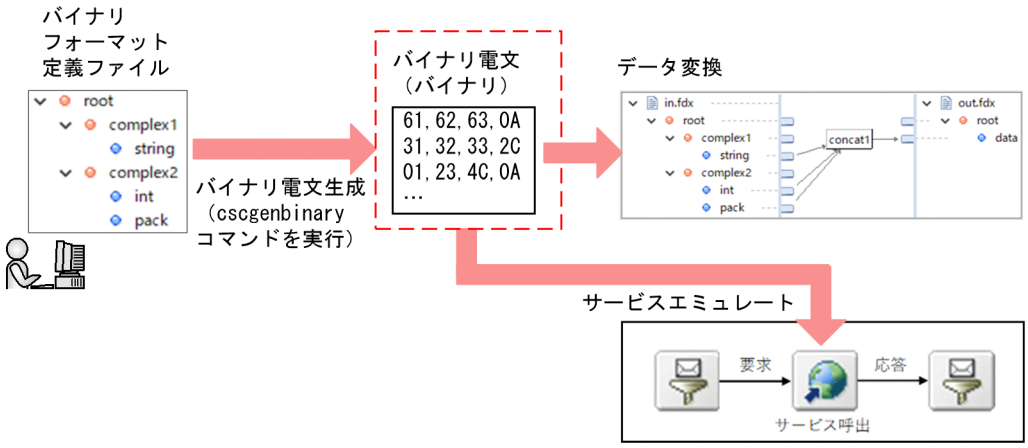
cscgenbinary コマンドでは，ビジネスプロセス，サービスアダプタやデータ変換機能のエミュレーションに使用する，テスト用の電文を作成できます。

例えば，次のような場合に利用できます。

- ビジネスプロセスを動かしたいが，要求電文がない場合
- バイナリフォーマット定義が作成済みで，サービスのエミュレーション機能を使用したいが，応答電文がない場合
- データ変換機能のエミュレーションを使用したいが，入力電文がない場合

cscgenbinary コマンドの実行の流れを次に示します。

図 10–15　cscgenbinary コマンドの実行の流れ

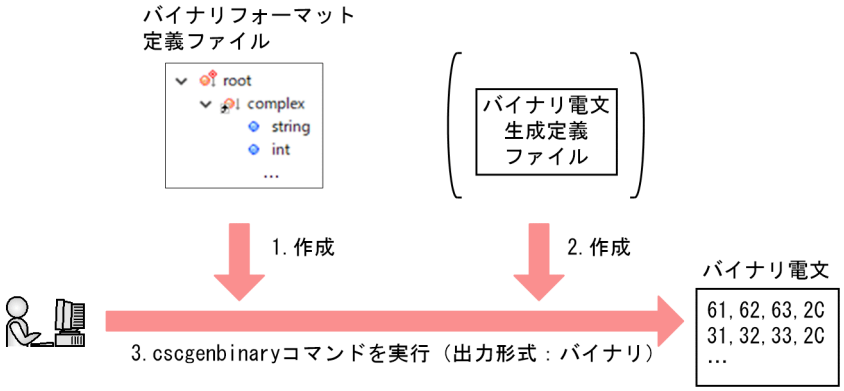


(1)　cscgenbinary コマンドの実行手順

cscgenbinary コマンドの実行手順を次に示します。

(a)　実行手順 1

図 10–16　cscgenbinary コマンドの実行手順 1



1. バイナリフォーマット定義ファイル（.fdx）を用意します。

2. 必要に応じて，バイナリ電文生成定義ファイルに要素値を定義します。

   バイナリ電文生成定義ファイルには，設定を変更したい要素と値を設定します。デフォルトの要素値を使用する場合はこのファイルは作成不要です。

   例：/root/complex/string=abc

   バイナリ電文生成定義ファイルの作成方法については，マニュアル「サービスプラットフォーム リファレンス」の「3.19.4　バイナリ電文生成定義ファイル」を参照してください。

3. cscgenbinary コマンドを実行します。

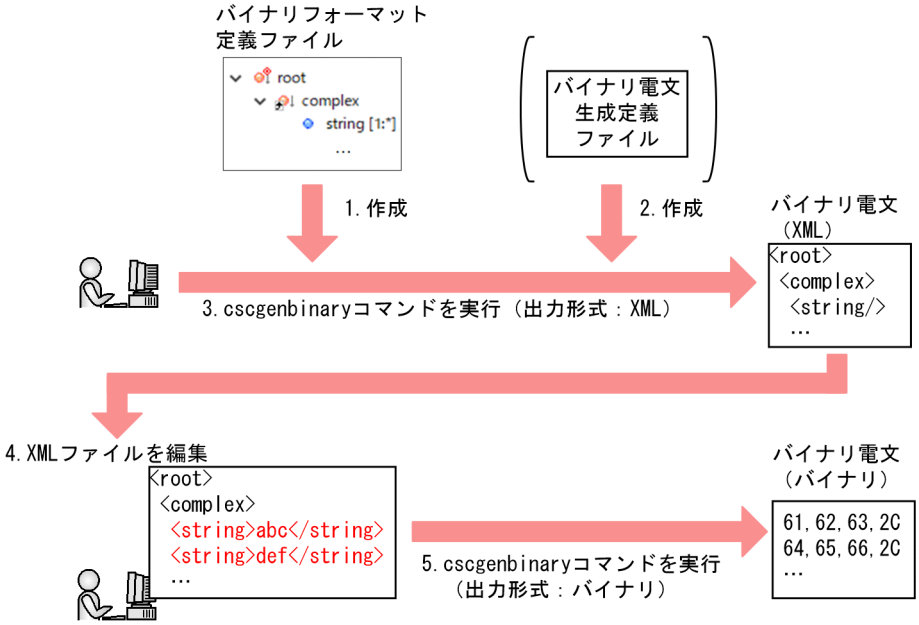
   コマンド引数には，手順 1 と手順 2 で作成したファイルのほか，ファイルの出力先パスと出力形式を指定します。出力形式としてバイナリファイルを指定すれば，バイナリファイルが出力されます。

これによって，バイナリ電文がファイルに出力されます。

(b)　実行手順 2：要素の構成を変更する場合

要素値のほかに要素の構成（繰り返し回数など）を変更したい場合は，バイナリ電文をいったん XML 形式で出力し，内容を編集してバイナリファイルに出力する方法があります。

図 10–17　cscgenbinary コマンドの実行手順 2



1. バイナリフォーマット定義ファイル（.fdx）を用意します。

2. 必要に応じて，バイナリ電文生成定義ファイルに要素値を定義します。

3. cscgenbinary コマンドを実行します。

   ここでは，コマンド引数に出力形式として XML ファイルを指定します。これによって，バイナリフォーマット定義のツリー情報が XML ファイルで出力されます。

4. 出力した XML ファイルを編集します。

   例えば，繰り返し要素を追加する場合，次の下線部分を修正します。

   XML ファイルの内容（修正前）

```
<root>
  <complex>
    <string/>
    …
```

   XML ファイルの内容（修正後）

```
<root>
  <complex>
    <string>abc</string>
    <string>def</string>
```

…

5. cscgenbinary コマンドを再実行します。

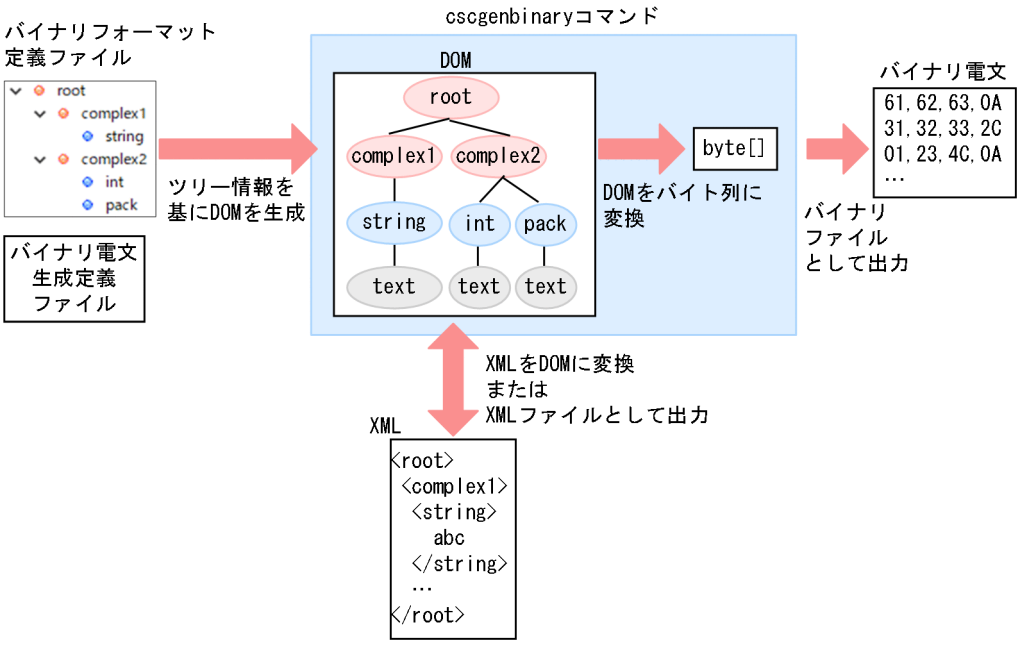
コマンド引数には出力形式としてバイナリファイルを指定します。

これによって，バイナリ電文がファイルに出力されます。

(2) DOM の生成規則

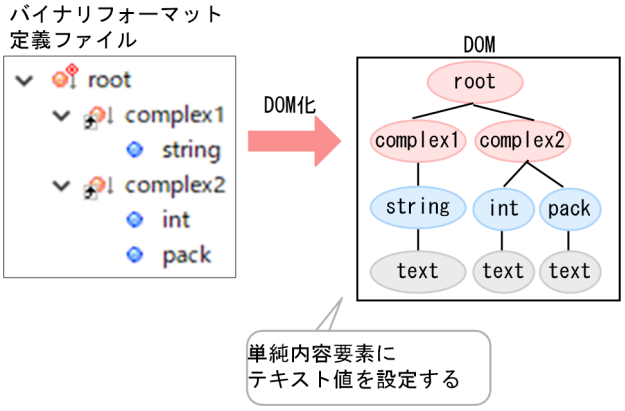
バイナリ電文と XML 電文の生成処理は，次の図のように DOM を介して実行されます。

図 10-18 バイナリ電文と XML 電文の生成処理



DOM はバイナリフォーマット定義ファイルのツリー情報を基に生成されます。単純内容要素には子要素としてテキストノードが追加され，テキスト値が設定されます。単純内容要素の DOM の生成例を次の図に示します。

図 10-19 DOM の生成例（単純内容要素の場合）



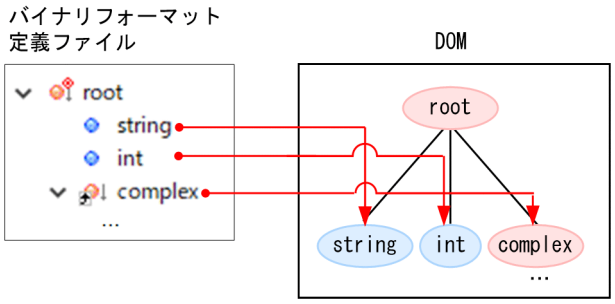
DOM からバイト列への処理では，バイナリフォーマット定義ファイルの定義内容に基づいて DOM の各要素値をバイト列へ変換します。変換の概要については，マニュアル「サービスプラットフォーム 解説」の「付録 B.2　データ変換処理の概要」を参照してください。

(a)　複合内容要素

複合内容要素の DOM の生成規則について説明します。

逐次構成

逐次構成の場合は，バイナリフォーマット定義ファイルのツリー構造に従って要素を生成します。



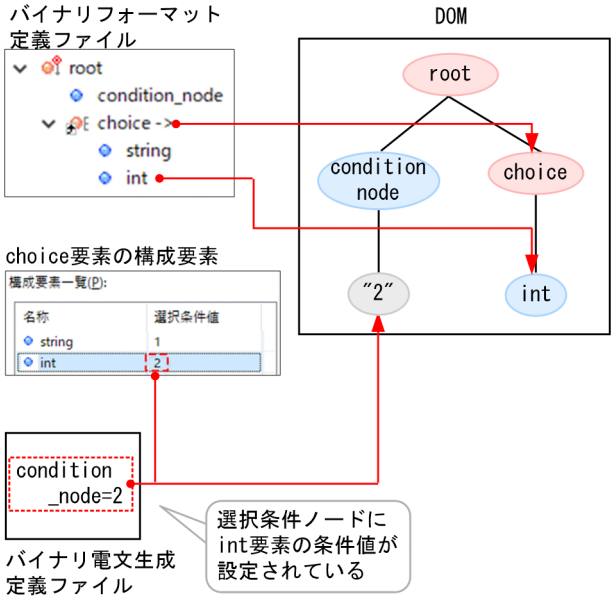
選択構成

選択構成の場合は，［選択種別］で選択したラジオボタンによって次のように生成規則が異なります。

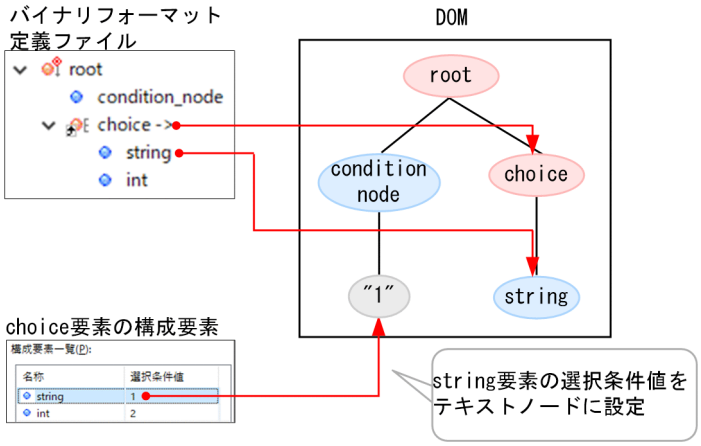
［選択条件ノード］を選択した場合

選択条件ノードで設定した条件値に合致する要素が存在する場合は，その要素を生成します。

例えば，次の例ではバイナリ電文生成定義ファイルで condition_node 要素（選択条件ノード）に「2」を設定しています。この場合，choice 要素の構成要素である int 要素の条件値と一致するため，int 要素が生成されます。

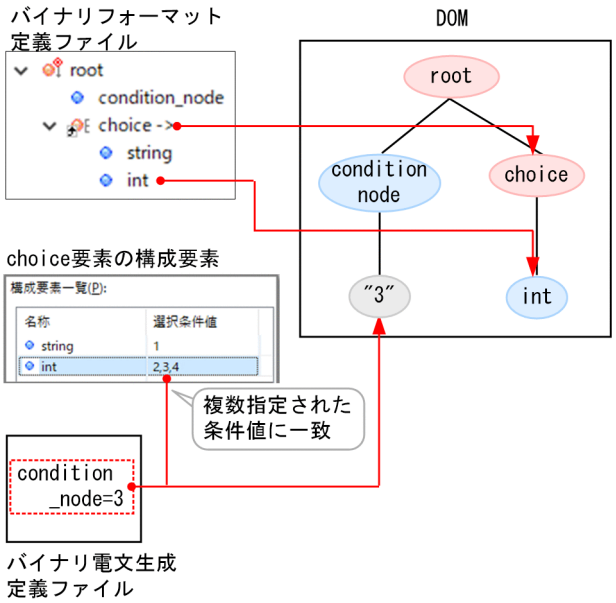


条件値に一致する要素がない場合は，先頭に定義されている要素を生成します。そのとき選択条件ノードのテキストノードには，先頭に定義されている要素の選択条件値を設定します。



選択条件値が複数指定されている場合は，生成する要素を次のように判定します。複数条件指定については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.5　条件値設定ダイアログ」にある，［複数条件指定］の説明を参照してください。

1. 先頭の構成要素に設定されている 1 番目の条件値から順に選択条件ノード値（バイナリ電文生成定義ファイルで設定した値）と比較します。
2. 条件値に一致する要素がある場合は，その要素を生成します。

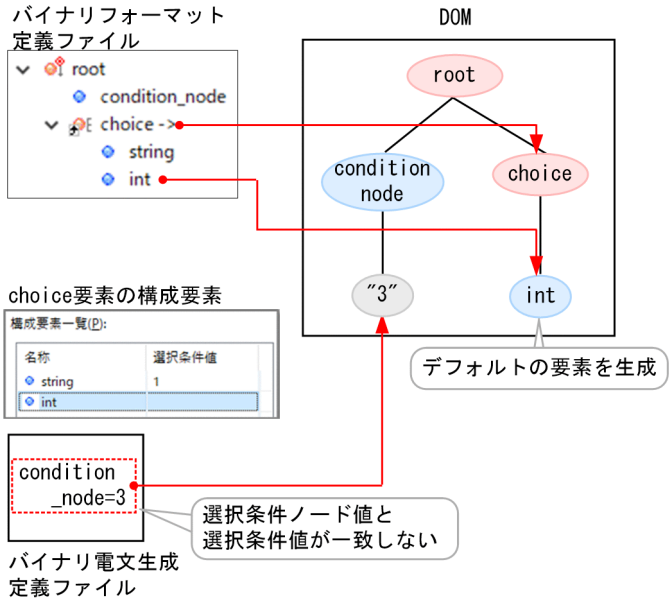


3. 条件値に一致する要素がない場合は，先頭に定義されている要素を生成します。

   先頭要素の条件値が複数指定されていた場合は，複数指定された条件値の中で先頭にある値を選択条件ノードのテキストノードに設定します。

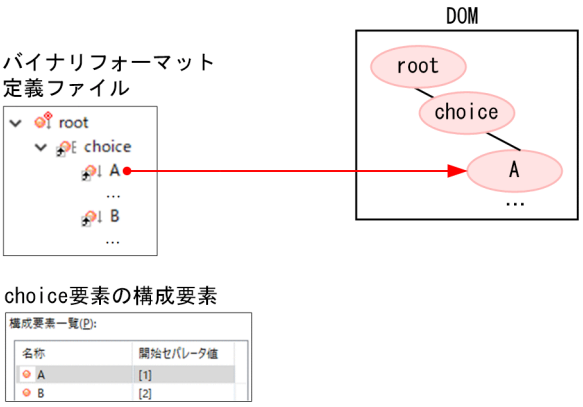
なお，選択構成では，選択条件値が空の要素（デフォルトの要素）を 1 つだけ定義できます。この要素は，どの条件値にも一致しなかった場合に生成されます。

次に示す例では，バイナリ電文生成定義ファイルで設定された選択条件ノード値「3」に一致する選択条件値を持つ要素が存在しないため，デフォルトの要素である int 要素が生成されます。



［構成要素の開始セパレータ］を選択した場合

構成要素の先頭に定義されている要素を生成します。



(b)　単純内容要素

単純内容要素の場合，子要素にテキストノードを追加します。生成内容は「(2) DOM の生成規則」の「図 10-19　DOM の生成例（単純内容要素の場合）」を参照してください。

テキストノードの設定値はバイナリ電文生成定義ファイルから読み込みます。バイナリ電文生成定義ファイルに定義がない要素は，デフォルト値が使用されます。各要素のデフォルト値については，マニュアル「サービスプラットフォーム リファレンス」の「3.19.4　バイナリ電文生成定義ファイル」の表を参照してください。

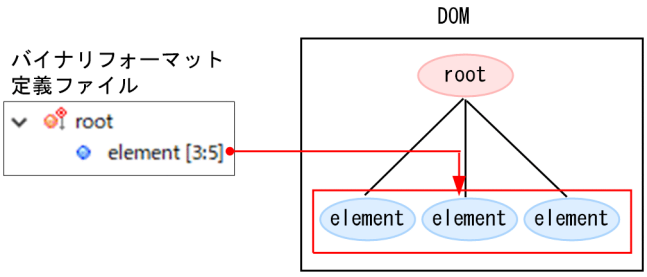
(c)　出現回数に関する生成規則

［構成要素］ダイアログの［出現回数］の選択に応じた生成規則を次に示します。

［出現回数］の選択	生成規則
［固定］ラジオボタン	指定された出現回数の分だけ要素を生成します。

［出現回数］の選択	生成規則
［範囲指定］ラジオボタン	指定された最小出現回数の分だけ要素を生成します。
［データの終端まで］ラジオボタン	要素を 1 つだけ生成します。データ省略可の場合も同様です。
［出現回数ノード］ラジオボタン	要素を 1 つだけ生成します。

［範囲指定］を選択した場合の生成例を次の図に示します。



(d)　ノード指定に関する生成規則

ノード指定が定義されている場合の生成規則を次に示します。

- サイズノード・出現回数ノード

  サイズノードおよび出現回数ノードは，空要素として生成され，バイナリデータ生成時（DOM→バイナリ）にノード指定元要素のデータサイズや要素数を基に要素値が自動で設定されます。

  そのため，バイナリ電文生成定義ファイルや XML ファイルで要素値を設定しないでください。

- 選択条件ノード

  「(2)(a)　複合内容要素」の「選択構成」の「［選択条件ノード］を選択した場合」と同じです。

注意事項

サイズノード，出現回数ノード，選択条件ノードが正しく生成されるよう定義してください。選択構成の構成要素内に上記のノードを定義すると，正しく生成されないでバイナリ電文の生成時にエラーとなる場合があります。

(e)　空データと Null データを区別するための設定

バイナリフォーマット定義ファイルで CSV 形式の空データと Null データを区別する設定を有効にしている場合，csctransform コマンドなどのデータ変換処理では nulldata 属性が付与されますが，cscgenbinary コマンドの実行では付与されません。そのため，「(1)(b)　実行手順 2：要素の構成を変更する場合」の手順 4.に従って，nulldata 属性を付与してください。

CSV 形式の空データと Null データを区別するための設定については，「付録 Q.2　空データと Null データを区別するための設定」を参照してください。

10.6　ビジネスプロセスのデバッグの終了

デバッグを終了する手順，およびデバッグを終了したときのビジネスプロセスとアクティビティの状態について説明します。

なお，ビジネスプロセスのデバッグの終了操作は，プロセスインスタンスの処理が中断しているときに行ってください。プロセスインスタンスの処理状態は，デバッグビューで確認できます。

デバッグを終了する手順を次に示します。

1. デバッグビューでデバッグ中のビジネスプロセスを選択します。

2. [終了] アイコン（  ）をクリックします。

   ビジネスプロセスのデバッグが終了されます。

プロセスインスタンスの処理中にデバッグを終了した場合，ビジネスプロセスとアクティビティのステータスは，デバッグを終了する直前に実行されたトランザクションのコミット時のステータスと同じ状態となります。

トランザクションのコミットのタイミングについては，「5.6　アクティビティの定義」を参照してください。

注意事項

ビジネスプロセスのデバッグで使用したビジネスプロセスやサービスアダプタは，デバッグの終了を行ったあとも HCSC サーバに配備，開始した状態になります。必要に応じて，「8.6　HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行」の方法で HCSC コンポーネントを停止して HCSC サーバから削除してください。

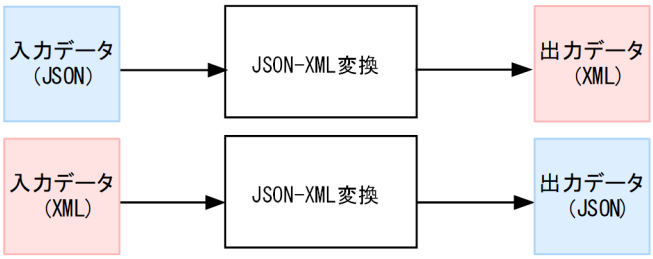
11 JSON データの変換

ここでは，JSON-XML 変換の概要，および JSON-XML 変換の規則や設定について説明します。JSON-XML 変換 API を使用したデータの形式変換については，「13.2　JSON-XML 変換 API を使用したデータの形式変換」を参照してください。

11.1　JSON-XML 変換の概要

JSON-XML 変換を利用して，JSON 形式のデータと XML 形式のデータを相互に変換できます。JSON-XML 変換では，JSON 形式のデータと XML 形式のデータを一定の基準に従って変換します。

図 11-1　JSON-XML 変換の概要



JSON-XML 変換で扱う JSON と標準仕様の JSON には差異があります。JSON-XML 変換で扱う JSON と標準仕様の JSON との差異については，「11.1.1　JSON-XML 変換で扱う JSON と標準仕様の JSON との差異」を参照してください。

また，JSON-XML 変換には，標準モードと拡張モードを利用できます。拡張モードと標準モードの差異については，「11.1.2　標準モードと拡張モードとの差異」を参照してください。

11.1.1　JSON-XML 変換で扱う JSON と標準仕様の JSON との差異

JSON-XML 変換で扱う JSON の仕様は，RFC8259 に準拠します。ただし，一部の仕様で RFC8259 と異なる点があります。詳細を次に示します。

- 単純型（オブジェクト型，配列以外）だけで構成されるデータは，JSON データとして扱わない。

  RFC8259 の仕様では，単純型（オブジェクト型，配列以外）だけの JSON データが許容されていますが，JSON-XML 変換機能では対応していません。単純型のデータが入力された場合エラーとなります。

  単純型として扱われる記述例を次に示します。

```
string (”Hello world!”)
number (42)
boolean (true)
```

- ws（空白）とみなす文字が異なる。
- 使用できるエスケープシーケンスが異なる。

  RFC8259 で定義されているエスケープシーケンスのうち，バックスペース（¥b），および改ページ（¥f）は使用できません。

  使用できるエスケープシーケンスを次に示します。

エスケープシーケンス	意味
¥”	ダブルクォーテーション（"）

エスケープシーケンス	意味
¥¥	バックスラッシュ
¥/	スラッシュ
¥n	ラインフィード（改行）
¥r	キャリッジリターン（行頭復帰）
¥t	タブ
¥uXXXX（4 桁の 16 進数）	Unicode 文字

11.1.2 標準モードと拡張モードとの差異

標準モードと拡張モードとの差異を次に示します。

表 11–1 標準モードと拡張モードとの差異

差異項目	標準モード	拡張モード
JSON 形式データの制限	あり	なし
変換結果※1	XML データの可読性が低い	XML データの可読性が高い
データタイプ	維持されない	維持される
XML スキーマ※2	変換時に不要	変換時に必要
変換結果※3	JSON 変換ツールで確認が必要	明確
可逆変換	できない	できる※4

注※1

標準モードでは，JSON 配列内の要素（無名要素）のタグ名に JSON-XML 変換定義ファイルで指定した固定名称が使用されますが，拡張モードでは XML スキーマで定義した要素タグが使用され，変換した XML データに任意の要素名を付与できます。

注※2

標準モードでは，cscjson2xsd コマンドで生成する XSD ファイルは標準モードの変換結果に合わせた XSD ファイルとなりますが，拡張モードでは，JSON データから XSD ファイルを参照して，XML スキーマに従った変換が行われます。

拡張モードでは，XSD ファイルに定義されていない要素が出現した場合，該当する要素の変換をスキップして，変換処理を続行することができます。

注※3

標準モードの JSON-XML 変換は，OSS に依存していますが，拡張モードの JSON-XML 変換は，マニュアルの記載に従って変換されます。

注※4

Fixed 要素や default 属性など，一部の設定は可逆変換ができません。

標準モードと拡張モードとの変換結果の差異を次に示します。

(1) JSON→XML 変換の変換結果

標準モードと拡張モードの JSON→XML 変換の変換結果の例を次に示します。ただし，拡張モードの変換結果は，適切な XSD ファイルを設定した場合とします。

表 11-2 JSON→XML 変換の変換結果の例

入力 JSON データ	標準モードの変換結果	拡張モードの変換結果
false	変換できない	<root>false</root>
null	変換できない	<root xsi:nill="true"/>
true	変換できない	<root>true</root>
123	変換できない	<root>123</root>
"abc"	変換できない	<root>abc</root>
["abc", "def", true]	<csc-array> <csc-element_csc-array> <csc-simple>abc</csc-simple> </csc-element_csc-array> <csc-element_csc-array> <csc-simple>def</csc-simple> </csc-element_csc-array> <csc-element_csc-array> <csc-simple>true</csc-simple> </csc-element_csc-array> </csc-array>	<root> <unnamed_1>abc</unnamed_1> <unnamed_1>def</unnamed_1> <unnamed_2>def</unnamed_2> </root> 配列要素の型ごとに異なる要素タグに変換します。

{
"field1":"abc",
"field2":"def"
}
	<csc-object>
	<field1>abc</field1>
	<field2>def</field2>
	</csc-object>
		<root>
		<field1>abc</field1>
		<field2>def</field2>
		</root>

{
"<fld>":"abc"
}
	変換できない	<root>
		<_fld_>abc</_fld_>
		</root>

[
true,
"def",
{
"field":"abc"
}
]
	<csc-array>
	<csc-element_csc-array>
	<csc-simple>true</csc-simple>
	</csc-element_csc-array>
	<csc-element_csc-array>
	<csc-simple>def</csc-simple>
	</csc-element_csc-array>
	<csc-element_csc-array>
	<field>abc</field>
	</csc-element_csc-array>
		<root>
		<unnamed_1>true</unnamed_1>
		<unnamed_2>def</unnamed_2>
		<unnamed_3>
		<field>abc</field>
		</unnamed_3>
		</root>

入力 JSON データ	標準モードの変換結果	拡張モードの変換結果
	</csc-array>	

注　拡張モードの変換結果の<unnamed_?>などの要素タグは，XML スキーマを編集し，適切な要素名に変更できます。

(2)　XML→JSON 変換の変換結果

標準モードと拡張モードの XML→JSON 変換の変換結果の例を次に示します。ただし，拡張モードの変換結果は，適切な XSD ファイルを設定した場合とします。

表 11-3　XML→JSON 変換の変換結果の例

入力 XML データ	標準モードの変換結果	拡張モードの変換結果
<root>false</root>	{"#text":"false"}	false
<root> <field>false</field> </root>	{"field":"false"}	{"field":false}
<root>123</root>	{"#text":"123"}	123
<root> <field>123</field> </root>	{"field":"123"}	{"field":123}
<root>abc</root>	{"#text":"abc"}	"abc"
<csc-object>abc</csc-object>	["abc"]	"abc"
<root> <field>abc</field> </root>	{"field":"abc"}	{"field":"abc"}
<root> <d1>value1</d1> <d1>value2</d1> </root>	{"d1": ["value1","value2"] }	アンラップ変換の場合{"d1": ["value1","value2"]} ラップ変換の場合 ["value1","value2"]
<csc-array> <csc-element_csc-array> <csc-simple>value1</csc-simple> </csc-element_csc-array> <csc-element_csc-array> <csc-simple>value2</csc-simple> </csc-element_csc-array> </csc-array>	["value1","value2"] csc-array に namespace を指定した場合は { "csc-element_csc-array":[{"csc-simple":"value1"},	["csc-element_csc-array":[{"csc-simple":"value1"}, {"csc-simple":"value2"}]] または [{"csc-simple":"value1"}, {"csc-simple":"value2"}

入力 XML データ	標準モードの変換結果	拡張モードの変換結果
	{"csc-simple":"value2"}] }	] または [["value1"], ["value2"]]

11.2　JSON-XML 変換の規則

JSON 形式から XML 形式へデータを変換，および XML 形式から JSON 形式へデータを変換する際の規則について説明します。これらの変換規則は，JSON 変換ツールの使用によって動作を確認できます。

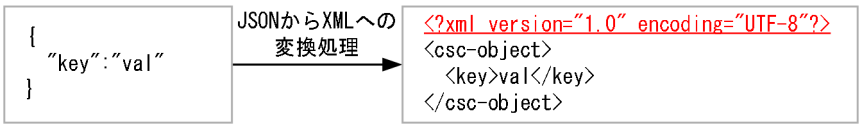
注意事項

説明に使用している変換例はインデントや改行などを加えていますが，実際の変換結果は整形されていません。

11.2.1　XML 宣言

JSON 形式から XML 形式への変換では，変換後の XML 形式データに XML 宣言が付与されます。また，XML 宣言の末尾には改行が付与されます。

図 11-2　XML 宣言の付与例



11.2.2　XML 形式データのルート要素

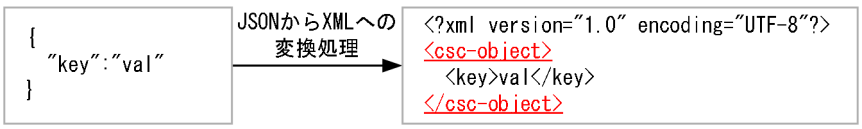
(1)　JSON 形式から XML 形式への変換

JSON 形式から XML 形式への変換では，変換後の XML 形式データに次に示すルート要素が設定されます。

- JSON 形式データがオブジェクト型から始まる場合

  変換後の XML 形式データのルート要素に csc-object 要素が設定されます。

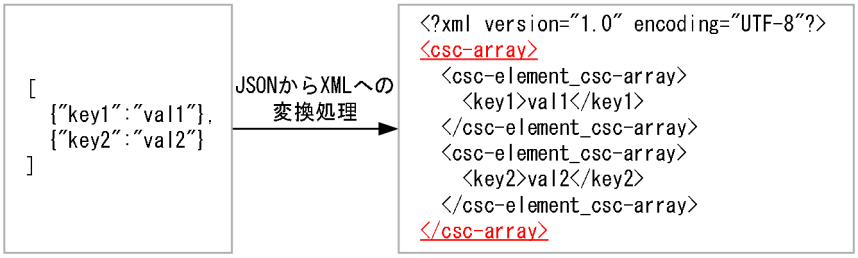
  図 11-3　ルート要素の設定例（オブジェクト型の変換）

- JSON 形式データが配列から始まる場合

  変換後の XML 形式データのルート要素に csc-array 要素が設定されます。

図 11-4　ルート要素の設定例（配列の変換）

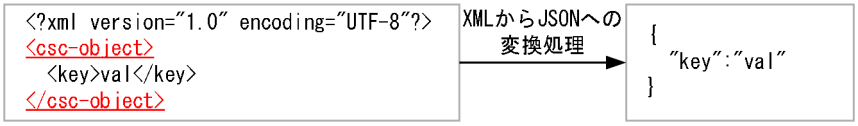


(2)　XML 形式から JSON 形式への変換

XML 形式から JSON 形式の変換では，XML 形式データのルート要素が削除されます。

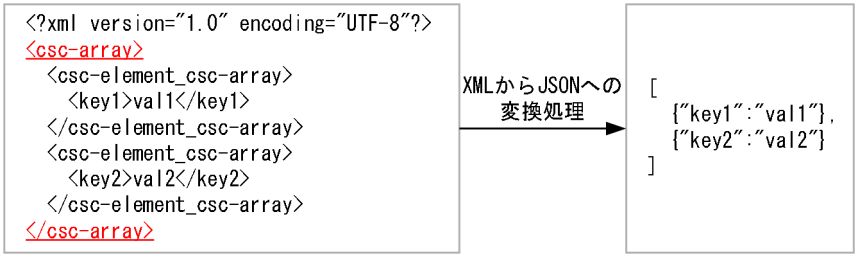
- オブジェクト型の変換の場合

図 11-5　ルート要素の削除例（オブジェクト型の変換）



- 配列の変換の場合

図 11-6　ルート要素の削除例（配列の変換）



11.2.3　オブジェクト型の変換

(1)　JSON 形式から XML 形式への変換

JSON のオブジェクト型は次の規則に従って XML の要素に変換されます。

- JSON のオブジェクト名：XML の要素名に変換
- JSON のオブジェクト値：XML の要素値に変換

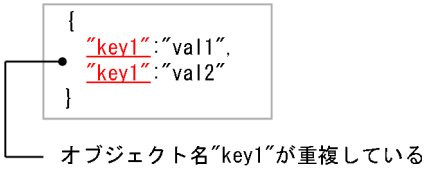
JSON 形式のオブジェクト名は，次の規則に従って指定してください。

- NCName で指定してください。
- JSON-XML 変換定義で使用している要素名と同じ名称を指定しないでください。

JSON-XML 変換定義の詳細については，「13.2.1　JSON-XML 変換 API 変換定義の設定」，および，マニュアル「サービスプラットフォーム リファレンス」の「3.18.1　JSON-XML 変換定義ファイル」を参照してください。

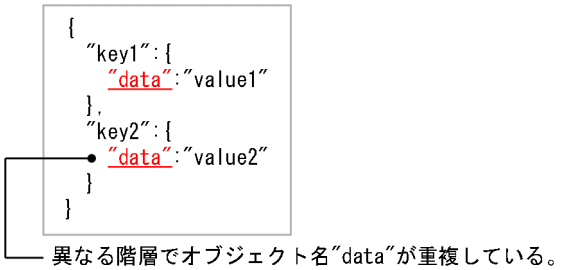
- 1 つのオブジェクト内で重複したオブジェクト名を指定しないでください。

図 11-7　オブジェクト名が重複しているデータ例



階層が異なる場合は，重複するオブジェクト名を指定できます。

図 11-8　異なる階層でオブジェクト名が重複しているデータ例



オブジェクト型の変換例を次の図に示します。

図 11-9　オブジェクト型の場合の変換例



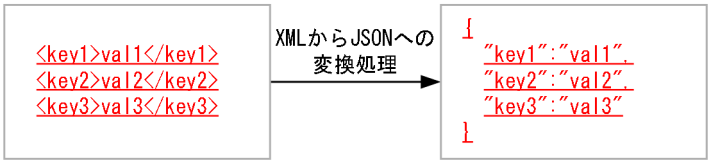
(2)　XML 形式から JSON 形式への変換

XML 形式から JSON 形式への変換では，XML の要素は次の規則に従って JSON のオブジェクト型に変換されます。

- XML の要素名：JSON のオブジェクト名に変換
- XML の要素値：JSON のオブジェクト値に変換

オブジェクト型への変換例を次の図に示します。

図 11-10　オブジェクト型への変換例



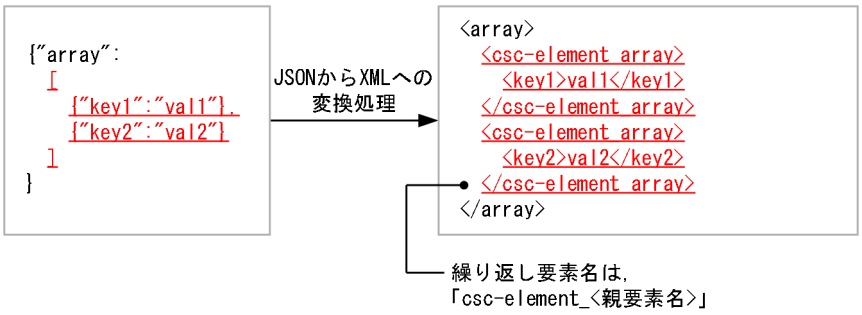
11.2.4　配列の変換

(1)　JSON 形式から XML 形式への変換

JSON の配列は，次の規則に従って XML の繰り返し要素（csc-element 要素）に変換されます。

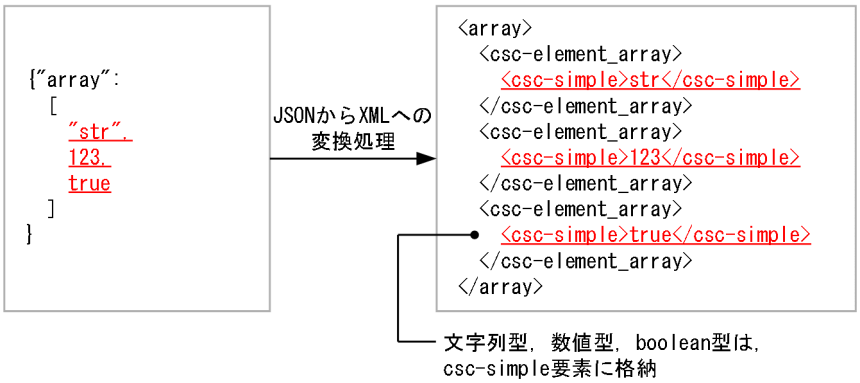
- JSON の配列値：XML の繰り返し要素の要素値に変換
  このとき，XML の繰り返し要素の名称は「csc-element ＋アンダーバー（_）＋親要素名」となります。

  図 11-11　配列の変換例



- JSON の配列値が文字列型，数値型，または boolean 型の場合：XML の繰り返し要素に子要素（csc-simple 要素）が追加され，値が格納されます。
  このとき，XML の子要素の名称は「csc-simple」となります。

  図 11-12　文字列型，数値型，boolean 型を含む配列の変換例



(2)　XML 形式から JSON 形式への変換

XML の繰り返し要素は，次の規則に従って JSON の配列に変換されます。

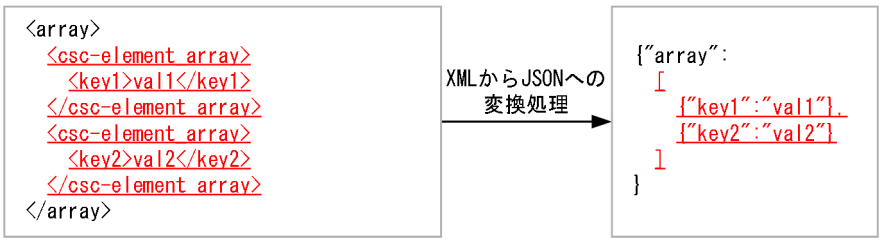
- XML の繰り返し要素の要素値：JSON の配列値に変換

  なお，要素数が 1 つの場合は，オブジェクト型に変換されます。要素数が 1 つの配列に変換したい場合は，class 属性（class="array"）を設定してください。

  class 属性については，「11.2.9(3)　class 属性」，および「13.2.1(5)　JSON のデータ種別を XML の属性に含めるかどうかの設定」を参照してください。

  繰り返し要素の変換例を次に示します。

  図 11-13　繰り返し要素の変換例



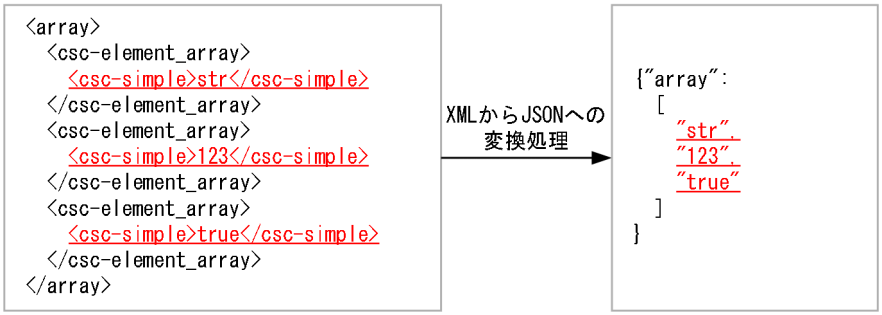
- XML の繰り返し要素内の子要素（csc-simple 要素）：JSON の文字列型の配列値に変換

  なお，繰り返し要素に type 属性が設定されている場合は，type 属性に対応する JSON 形式データに変換されます。

  type 属性については，この節の各データ型の変換規則，および「13.2.1(5)　JSON のデータ種別を XML の属性に含めるかどうかの設定」を参照してください。

  csc-simple 要素を含む繰り返し要素の変換例を次に示します。

  図 11-14　csc-simple 要素を含む繰り返し要素の変換例



11.2.5　null 値の変換

(1)　JSON 形式から XML 形式への変換

JSON の null 値は，XML の空要素に変換され，null 属性（null="true"）が設定されます。null 値の変換例を次の図に示します。

図 11-15　null 値の変換例（オブジェクト型）

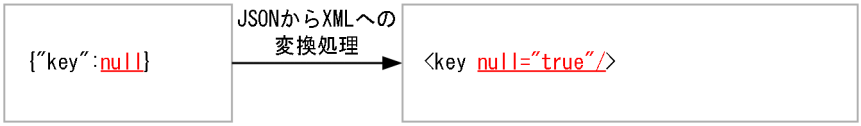
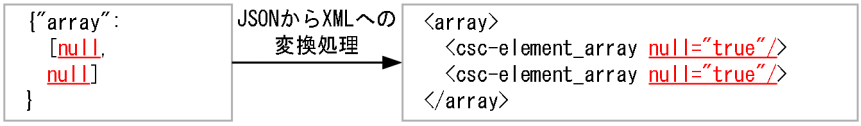


図 11-16　null 値の変換例（配列）



(2)　XML 形式から JSON 形式への変換

XML 形式の要素が次の条件を満たす場合は，JSON の null 値に変換されます。

- 要素が空要素である
- 要素の属性に null 属性（null="true"）が設定されている

null 値への変換例を次の図に示します。

図 11-17　null 値への変換例（オブジェクト型）

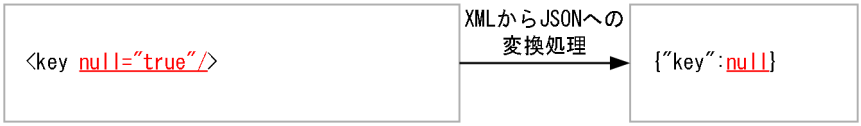
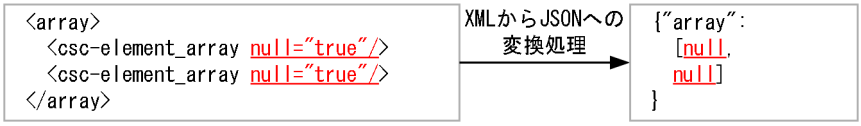


図 11-18　null 値への変換例（配列）

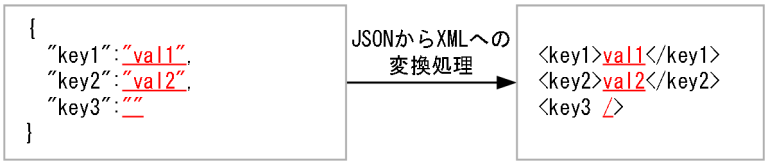


11.2.6　文字列型の変換

(1)　JSON 形式から XML 形式への変換

JSON の文字列型は，XML の要素値に変換されます。文字列型の変換例を次の図に示します。

図 11-19　JSON 文字列型の変換例



エスケープシーケンスの変換

JSON の文字列型にエスケープシーケンスを指定した場合は，XML の要素値に変換されます。エスケープシーケンスの変換規則を次の表に示します。

JSON の文字列型	XML の要素値
¥”（ダブルクォーテーション）	”
¥¥（バックスラッシュ）	¥
¥/（スラッシュ）	/
¥n（ラインフィード）	（ラインフィード）
¥r（キャリッジリターン）
¥t（タブ）	（タブ）
¥uXXXX（4 桁の 16 進数）	XXXX が示す文字（例：¥u3042 を"あ"に変換）

（凡例）

¥：バックスラッシュを示します。

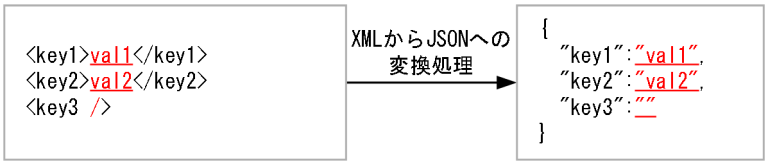
注意事項

JSON の文字列型に単一のバックスラッシュ（¥）を指定した場合，変換処理でエラーになります。

(2) XML 形式から JSON 形式への変換

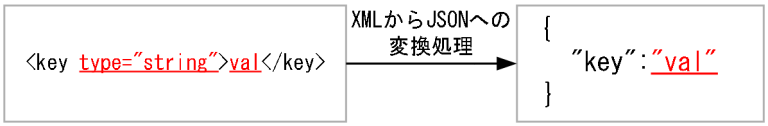
XML の要素値（文字列）は，JSON の文字列型に変換されます。文字列型への変換例を次の図に示します。

図 11-20 文字列型への変換例



また，XML の要素に type 属性（type="string"）を設定した場合，要素値が文字列型に変換されます。

図 11-21 文字列型への変換例（type 属性あり）



エスケープシーケンスへの変換

JSON の文字列型にエスケープシーケンスを設定する場合の，XML の要素値の変換規則を次の表に示します。

XML の要素値	JSON の文字列型
”	¥”（ダブルクォーテーション）
¥	¥¥（バックスラッシュ）

XML の要素値	JSON の文字列型
/	/（スラッシュ）
（ラインフィード）	¥n（ラインフィード）
¥r（キャリッジリターン）
（タブ）	¥t（タブ）

注意事項

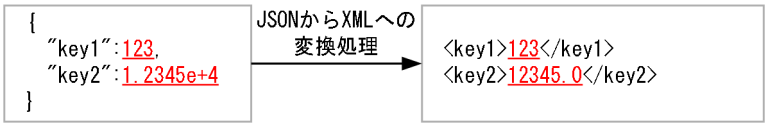
¥uXXXX（4 桁の 16 進数）への変換はできません。例えば，XML の要素値に「あ」を指定しても JSON の文字列型は¥u3042 ではなく「あ」となります。

11.2.7　数値型の変換

(1)　JSON 形式から XML 形式への変換

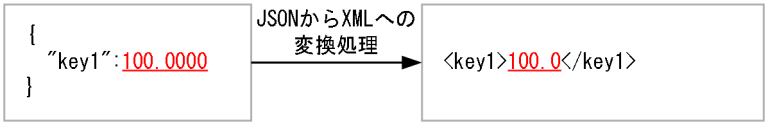
JSON の数値型は，XML の要素値に変換されます。数値は java.lang.Integer または java.lang.Double の範囲で設定してください。数値型の変換例を次に示します。

図 11-22　数値型の変換例



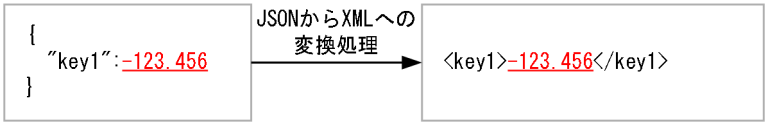
有効数字の情報は保持されないため，小数点第二位以下の 0 は削除されます。

図 11-23　数値型の変換例（余分な 0 の削除）



整数部の先頭に「-」マイナスを指定した場合の変換例を次の図に示します。

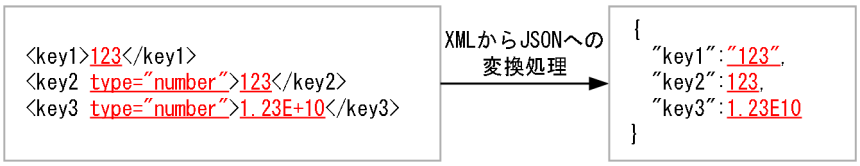
図 11-24　数値型の変換例（先頭にマイナスを指定）



(2)　XML 形式から JSON 形式への変換

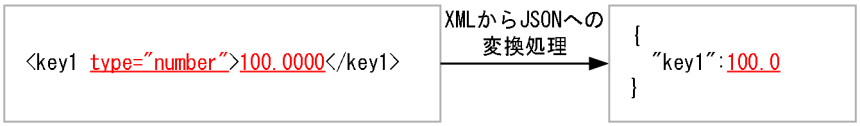
XML の要素値（数値）は，JSON の文字列型に変換されます。JSON の数値型として変換する場合は，該当する要素に type 属性（type="number"）を設定してください。数値は java.lang.Integer または java.lang.Double の範囲で設定してください。数値型への変換例を次の図に示します。

図 11-25　数値型への変換例



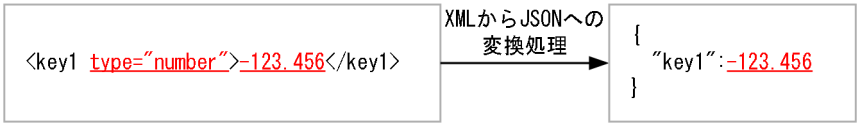
有効数字の情報は保持されないため，小数点第二位以下の 0 は削除されます。

図 11-26　数値型への変換例（余分な 0 の削除）



整数部の先頭に「-」マイナスを指定した場合の変換例を次の図に示します。

図 11-27　数値型への変換例（先頭にマイナスを指定）



11.2.8　boolean 型の変換

(1)　JSON 形式から XML 形式への変換

JSON の boolean 型（true，false）は，XML の要素値に変換されます。boolean 型の変換例を次の図に示します。

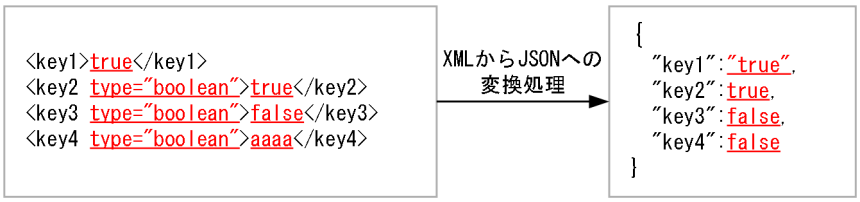
図 11-28　boolean 型の変換例



(2)　XML 形式から JSON 形式への変換

XML の要素値の「true」および「false」は，JSON の文字列型に変換されます。JSON の boolean 型として変換する場合は，該当する要素に type 属性（type="boolean"）を設定してください。また，「true」，「false」以外の値が設定されている場合は，「false」に変換されます。boolean 型への変換例を次の図に示します。

図 11-29　boolean 型の変換例



11.2.9　XML 形式データの属性

JSON-XML 変換では，XML 形式データに次の属性を設定できます。

- type
- null
- class

そのほかの属性は定義できません。属性が変換規則に従って定義されていない場合は，変換処理が正常に動作しません。

各属性の詳細を説明します。

(1)　type 属性

XML の要素に，JSON の文字列型，数値型，boolean 型の情報を持たせるために使用する属性です。XML 形式から JSON 形式への変換では，文字列型，数値型，boolean 型に対応する要素に type 属性を設定することで，JSON の各データ型に変換できます。

type 属性については，この節の各データ型の変換規則，および「13.2.1(5)　JSON のデータ種別を XML の属性に含めるかどうかの設定」を参照してください。

(2)　null 属性

XML の要素に，JSON の null 値の情報を持たせるために使用する属性です。XML 形式から JSON 形式への変換では，対象の要素に null 属性を設定することで，JSON の null 値に変換できます。

詳細については，「11.2.5　null 値の変換」を参照してください。

(3)　class 属性

XML の要素に，子要素の構成情報を持たせるために使用する属性です。

class 属性を指定しても子要素が空文字の場合の動作は不定です。

次の値が設定できます。

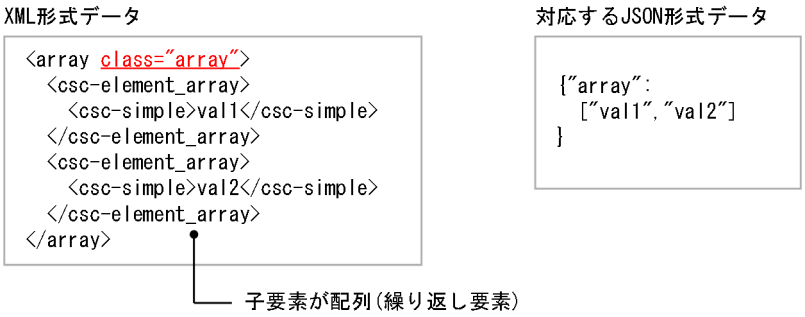
- 子要素が JSON のオブジェクト型で構成されている場合：「class="object"」

図 11-30　class 属性の設定例（object）



- 子要素が JSON の配列で構成されている場合：「class="array"」

図 11-31　class 属性の設定例（array）



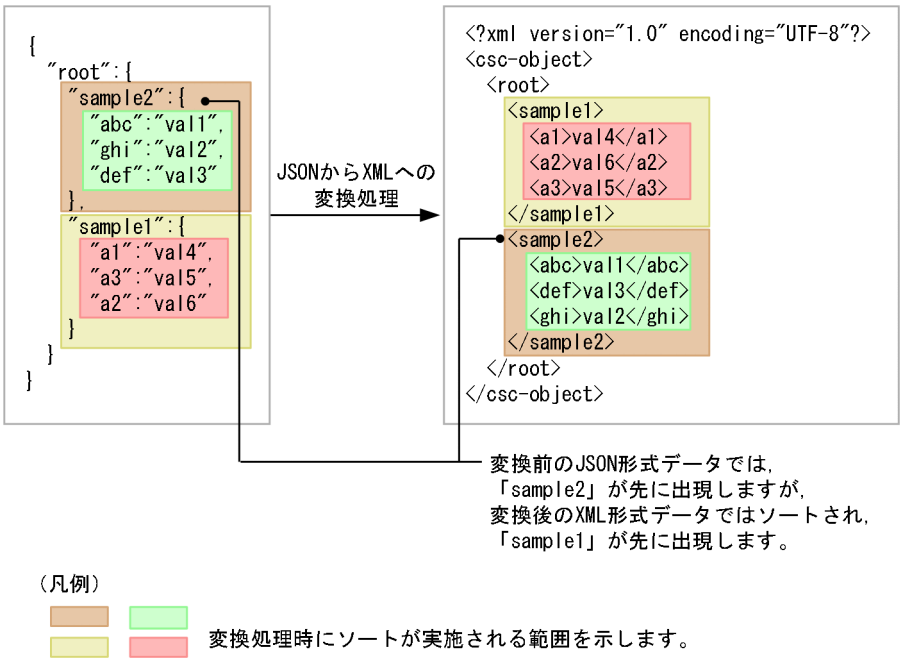
XML 形式から JSON 形式の変換で，上記の class 属性を設定しない場合は，JSON-XML 変換機能が子要素の構成を自動で判別し変換されます。class 属性を設定した場合は，設定値に応じて変換します。

11.2.10　JSON 形式から XML 形式に変換する際のオブジェクト名の順序付け

JSON 形式から XML 形式への変換では，JSON 形式データをオブジェクト名の昇順にソートしてから XML 形式データに変換します。ソートは同列のオブジェクト単位で実施されます。そのため，変換前の JSON 形式データと変換後の XML 形式データでは，データの出現順序が異なることがあります。

JSON 形式から XML 形式に変換する場合のオブジェクト名の順序付けの例を次の図に示します。

図 11-32　JSON 形式から XML 形式に変換する際のオブジェクト名の順序付け



なお，XML 形式から JSON 形式の変換では，これらのソートは実施されません。

11.3　JSON-XML 変換定義ファイルの設定

JSON-XML 変換定義ファイルで，JSON-XML 変換時の動作を設定できます。

JSON-XML 変換定義ファイルの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「3.18.1　JSON-XML 変換定義ファイル」を参照してください。

JSON-XML 変換定義ファイルでは，JSON 形式から XML 形式に変換する際の変換動作を変更できます。変換動作を変更できる JSON-XML 変換定義ファイルのプロパティを次の表に示します。

表 11–4　JSON 形式から XML 形式に変換する際に変換動作を変更できるプロパティ

項番	JSON-XML 変換定義ファイルのプロパティ名	説明
1	json-xml-convert.arrays.parent-element-name	csc-array 要素の要素名を変更できます。
2	json-xml-convert.arrays.child-element-name	csc-element 要素の要素名を変更できます。
3	json-xml-convert.arrays.child-simple-element-name	csc-simple 要素の要素名を変更できます。
4	json-xml-convert.root-element-name	csc-object 要素の要素名を変更できます。
5	json-xml-convert.json-type.include	JSON 形式データの種別を，XML 形式データの属性に含めるかどうかを設定できます。
6	json-xml-convert.decimal-xmloutput-control	JSON 形式データから XML 形式データへの変換を行う際に，小数点を含む数値項目の出力形式を制御するかどうかを設定できます。

各プロパティの詳細を次に示します。表の項番 1～6 は以降の 11.3.1～11.3.6 項と対応しています。

11.3.1　csc-array 要素の要素名の変更

配列から始まる JSON 形式データを XML 形式データに変換する際に使用される csc-array 要素の要素名として，任意の名称を設定できます。

11.3.2　csc-element 要素の要素名の変更

JSON の配列を XML に変換する際に使用される csc-element 要素の要素名として，任意の名称を設定できます。変換後の要素名は，「設定値＋アンダーバー（_）＋親要素名」となります。

11.3.3　csc-simple 要素の要素名の変更

JSON の配列を XML に変換する際に使用される csc-simple 要素の要素名として，任意の名称を設定できます。

11.3.4　csc-object 要素の要素名の変更

オブジェクトから始まる JSON 形式データを XML 形式データに変換する際に使用される csc-object 要素の要素名として，任意の名称を設定できます。

11.3.5　JSON のデータ種別を XML の属性に含めるかどうかの設定

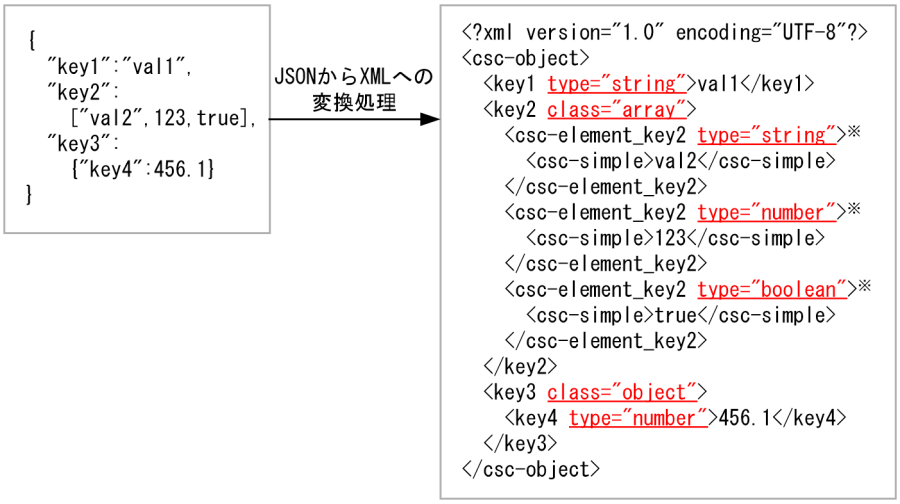
JSON の「文字列型」などのデータ種別を，変換後の XML の属性に含めるかどうかの設定を変更できます。デフォルトでは，JSON のデータ種別を XML の属性に含めません。JSON のデータ種別，および変換後の XML に設定される属性を次の表に示します。

表 11-5　JSON のデータ種別と XML の属性の対応表

JSON のデータ種別	XML の属性
オブジェクト	class="object"
配列	class="array"
文字列	type="string"
数値	type="number"
boolean	type="boolean"

JSON のデータ種別を XML の属性に含める場合の変換例を，次の図に示します。

図 11-33　JSON のデータ種別を XML の属性に含める場合の変換例



注

下線が付いている属性が JSON のデータ種別です。

注※

配列値が文字列型，数値型，boolean 型の場合，属性は csc-element 要素に設定されます。csc-simple 要素には設定されません。

注意事項

JSON-XML 変換に失敗した場合，メッセージログにエラーメッセージ（KDEC11202-E）が出力されます。また，メッセージ引数の「message」には，JSON-XML 変換のエラーメッセージが格納されます。

11.3.6　小数点付き数値項目の出力形式の設定

JSON 形式データから XML 形式データへの変換を行う際に，小数点を含む数値項目の出力形式を制御するかどうかを設定できます。デフォルトでは，小数点を含む数値項目の次のどれかのメソッド仕様に従い数値を出力します。

- java.lang.Float.toString()
- java.lang.Double.toString()
- java.math.BigDecimal.toString()

デフォルトで設定した場合の変換例を，次に示します。

入力データ（JSON）

```
{”key1”:10000000.0}
```

変換結果（XML）

```
<?xml version=”1.0” encoding=”UTF-8”?>
<csc-object><key1>1.0E7</key1></csc-object>
```

11.4 JSON-XML 変換に関する注意事項

JSON-XML 変換に関する注意事項を次に示します。

11.4.1 XML 形式のデータから JSON 形式のデータに変換する場合の注意事項

XML 形式から JSON 形式の変換で使用する XML 形式データの注意事項を次に示します。

- XML 形式データの名前空間と名前空間接頭辞は，JSON 形式データに変換されません。
- XML 形式データのコメントは，JSON 形式データに変換されません。

11.4.2 JSON 形式のデータと XML 形式のデータを相互変換する場合の注意事項

JSON-XML 変換の注意事項を次に示します。

- JSON-XML 変換を実行する場合は，正しい形式のデータを使用して，変換処理をしてください。不正なデータを使用すると意図しない変換結果となる場合があります。
- JSON-XML 変換の変換結果を再度変換した場合に，変換結果が変換前と同じデータに戻らない場合があります。

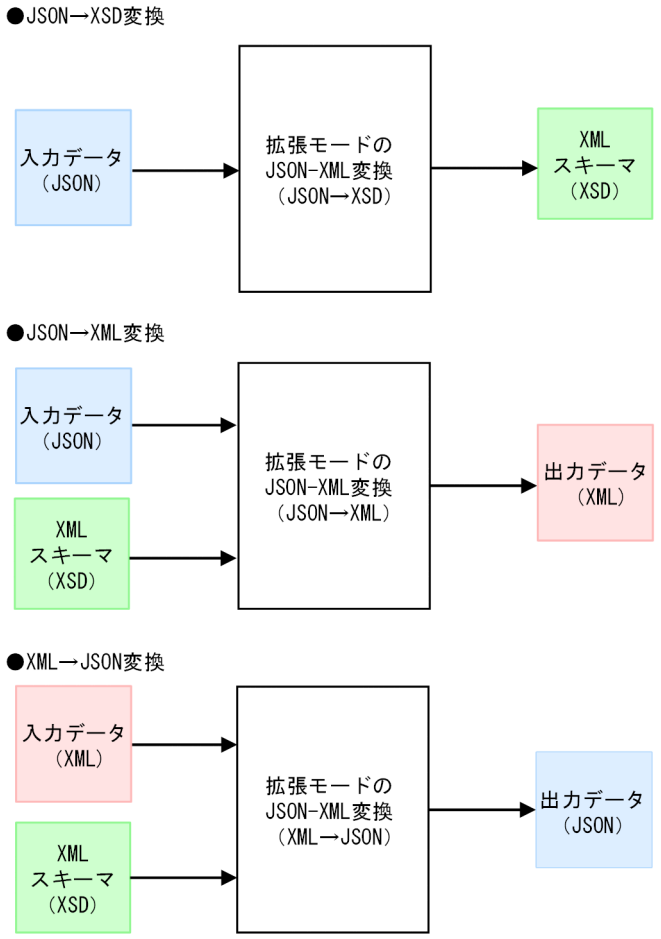
12 拡張モードでの JSON データの変換

この章では，拡張モードでの JSON データの変換について説明します。

12.1　拡張モードの JSON-XML 変換の概要

拡張モードの JSON-XML 変換では，XML スキーマ（XSD ファイル）を参照して変換することで，XML スキーマに従って JSON 形式データと XML 形式データを相互に変換できます。参照する XML スキーマは，拡張モードの JSON→XSD 変換で，JSON 形式データから作成できます。

図 12–1　拡張モードの JSON-XML 変換の概要



JSON-XML 変換には，XML スキーマを参照しないで変換する標準モードもあります。標準モードと拡張モードの差異については，「11.1.2　標準モードと拡張モードとの差異」を参照してください。

12.2　拡張モードの JSON-XML 変換の設定

拡張モードの JSON-XML 変換を利用するには，次の設定を準備します。

12.2.1　OSS のインストール

拡張モードの JSON-XML 変換を利用する場合，次の OSS が必要です。

表 12-1　導入 OSS 一覧

項番	OSS	説明
1	xmlschema-core-2.3.1.jar	Apache WS XmlSchema Core
2	jackson-annotations-2.17.1.jar	Jackson Annotations
3	jackson-core-2.17.1.jar	Jackson Core
4	jackson-databind-2.17.1.jar	Jackson Databind

これらの OSS を次のディレクトリにインストールしてください。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external
```

12.2.2　クラスパスの設定

usrconf.cfg（J2EE サーバ用オプション定義ファイル）の add.class.path に，次のクラスパスを追加してください。

```
add.class.path=<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥csc_advjsonxml.j
ar
add.class.path=<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥xmlsche
ma-core-2.3.1.jar
add.class.path=<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥jackson
-annotations-2.17.1.jar
add.class.path=<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥jackson
-core-2.17.1.jar
add.class.path=<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥jackson
-databind-2.17.1.jar
```

usrconf.cfg の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.2　usrconf.cfg（J2EE サーバ用オプション定義ファイル)」を参照してください。

12.2.3　拡張モードの JSON-XML 変換の制限事項

拡張モードで利用する OSS には，次の制限があります。

これらの制限値を超えた JSON 形式データを扱う場合，JSONtoXMLConverter#toXML(JsonNode jsonNode)メソッドを使用する必要があります。

(1)　JSON 形式データの読み込み制限

JSON 形式データの読み込みには，次の制限があります。

- 数値の最大桁数：1,000 桁
- 文字列の最大長：20,000,000 文字
- ノードの深さ：1,000 レベル
- プロパティ名（フィールド名）の最大長：50,000 バイト

ドキュメント長は無制限です。

(2)　JSON 形式データの出力制限

ノードの深さは，1,000 レベルまでです。

12.3　拡張モードの JSON-XML 変換の規則

拡張モードで JSON-XML 変換をする際の規則について説明します。

12.3.1　JSON→XSD 変換の規則

JSON→XSD 変換の規則について説明します。

表 12–2　JSON→XSD 変換の規則

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
false	<xs:element name="root" type="xs:boolean"/>
null	<xs:element name="root" type="xs:string" nillable="true"/>※
true	<xs:element name="root" type="xs:boolean"/>
123	<xs:element name="root" type="xs:decimal"/>
"abc"	<xs:element name="root" type="xs:string"/>
["abc", "def"]	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="unnamed_1" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element>

{
"field1":"abc",
"field2":"def"
}
	<xs:element name="root" csc:wrapped="true">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>

注※　type 属性は要素によって異なります。

XML 宣言および schema 要素については，「12.3.1(1)　XML 宣言および schema 要素の変換」を参照してください。

(1)　XML 宣言および schema 要素の変換

JSON→XSD 変換で生成する XSD の XML 宣言および schema 要素は，次のように出力されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema  xmlns:csc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/jsonxml">
</xs:schema>
```

(2) minOccurs 属性の変換

ルート要素および配列要素以外の element 要素には，minOccurs 属性に 0（省略可）を設定します。ルート要素および配列要素の element 要素には，minOccurs 属性はデフォルトの 1 が適用されます。

表 12-3 minOccurs 属性の設定例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
"abc"	<xs:element name="root" type="xs:string"/>
["abc"]	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="unnamed_1" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element>

{
"field1":"v1",
"field2":123
}
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string" />
	<xs:element name="field2" minOccurs="0" type="xs:decimal"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>

(3) 要素名の自動生成

JSON 形式データでは，JSON Object のフィールド以外にフィールド名（要素名）は存在しませんが，XML では，要素名（要素タグ）が必要です。そのため，JSON→XSD 変換では，要素名を次のように自動生成します。

- ルート要素の要素名：root
- ルート要素以外の無名要素名：unnamed_<連番>

JSON→XSD 変換の結果を編集し，読みやすい要素名に変更することをお勧めします。

自動生成する要素名は，JSONtoXSDConverterFactory.getConverter(Map<Object,Object> options) の options パラメタで変更できます。

表 12–4　自動生成する要素名の変更パラメタ

対象	options パラメタ key	デフォルト値
ルート要素の要素名	"rootTag"	"root"
ルート要素以外の無名要素名	"unnamedTagPrefix"	"unnamed_"

(4)　1 次元配列の JSON→XSD 変換

JSON 形式データでは，配列に複数のデータ型の要素を含めることができます。JSON→XSD 変換では，配列内にあるデータ型をすべて choice 要素内の element 要素に変換します。

ただし，JSON オブジェクトの 1 次元配列フィールドで，各要素のデータ型が単一の場合は，maxOccurs="unbounded"を指定した<element>に変換されます。

表 12–5　1 次元配列の JSON→XSD 変換の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
[　true, 　null, 　false, 　123, 　"abc", 　{ 　　"field1":"v1", 　　"field2":"v2" 　}, 　[　　"def", 　　"hij" 　]]	<xs:element name="root" csc:wrapped="true"> 　<xs:complexType> 　　<xs:choice minOccurs="0" maxOccurs="unbounded"> 　　　<xs:element name="unnamed_1" nillable="true" type="xs:boolean"/> 　　　<xs:element name="unnamed_2" nillable="true" type="xs:string"/> 　　　<xs:element name="unnamed_3" nillable="true" type="xs:decimal"/> 　　　<xs:element name="unnamed_4" nillable="true"> 　　　　<xs:complexType> 　　　　　<xs:sequence> 　　　　　　<xs:element name="field1" minOccurs="0" type="xs:string"/> 　　　　　　<xs:element name="field2" minOccurs="0" type="xs:string"/> 　　　　　</xs:sequence> 　　　　</xs:complexType> 　　　</xs:element> 　　　<xs:element name="unnamed_6" nillable="true" csc:wrapped="true"> 　　　　<xs:complexType> 　　　　　<xs:choice minOccurs="0" maxOccurs="unbounded"> 　　　　　　<xs:element name="unnamed_5" type="xs:string"/> 　　　　　</xs:choice> 　　　　</xs:complexType> 　　　</xs:element> 　　</xs:choice> 　</xs:complexType> </xs:element>

注　XML 宣言と schema 要素は省略しています。

多次元配列に対応する JSON→XSD 変換については，「12.3.1(5)　多次元配列の JSON→XSD 変換」を，Object 型の配列要素については，「12.3.1(6)　配列要素が JSON Object の場合の JSON→XSD 変換」を参照してください。

(5)　多次元配列の JSON→XSD 変換

多次元配列の各次元のデータ型は 1 つの choice 要素内に含まれます。

表 12–6　多次元配列の JSON→XSD 変換の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

[
[true,false],
[123, "abc"],
[
{
"field1":"v1",
"field2":"v2"
},
"def"
],
[
{
"field1":"v3",
"field2":"v4"
}
]
]
	<xs:element name="root" csc:wrapped="true">
	<xs:complexType>
	<xs:choice minOccurs="0" maxOccurs="unbounded">
	<xs:element name="unnamed_2" csc:wrapped="true">
	<xs:complexType>
	<xs:choice maxOccurs="unbounded">
	<xs:element name="unnamed_1" type="xs:boolean"/> <-- true, false -->
	<xs:element name="unnamed_3" type="xs:decimal"/> <-- 123 -->
	<xs:element name="unnamed_4" type="xs:string"/> <-- "abc", "def" -->
	<xs:element name="unnamed_5"> <-- {...} -->
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string" />
	<xs:element name="field2" minOccurs="0" type="xs:string" />
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>

注　XML 宣言と schema 要素は省略しています。

(6)　配列要素が JSON Object の場合の JSON→XSD 変換

配列要素に次の条件を満たす複数の JSON Object が存在する場合，各 JSON Object のフィールドをマージし，1 つの sequence 要素にまとめて定義されます。

- 条件 1：同名かつ同データ型のフィールドを持つ JSON Object
- 条件 2：同名のフィールドを持たない JSON Object

次の例では，field1 は条件 1 に合致し，field2，field3，field4 は条件 2 に合致するため，3 つの Object をマージして 1 つの sequence 要素にまとめて定義されます。

表 12–7　マージする配列要素 Object の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

[
{
"field1":"v1",
"field2":"v2"
},
{
"field1":"v3",
"field3":123
},
{
"field1":"v4",
"field4":true
}
]
	<xs:element name="root" csc:wrapped="true">
	<xs:complexType>
	<xs:choice minOccurs="0" maxOccurs="unbounded">
	<xs:element name="unnamed_1">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0" type="xs:string"/>
	<xs:element name="field3" minOccurs="0" type="xs:decimal"/>
	<xs:element name="field4" minOccurs="0" type="xs:boolean"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>

注　XML 宣言と schema 要素は省略しています。

複数の配列要素 Object が次の条件に合致する場合は，別の異なる sequence 要素を持つ，別の element 要素として定義されます。

- 条件 3：異なるデータタイプの同名フィールドが存在する

次の例では，field1 は条件 1 に合致し，field2 は条件 3 に合致するため，それぞれの Object は異なる sequence 要素を持つ element 要素として定義されます。

表 12–8　マージしない配列要素 Object の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
[{ "field1":"v1", "field2":"v2" }, { "field1":"v3", "field2":123 },	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="unnamed_1"> <xs:complexType> <xs:sequence> <xs:element name="field1" minOccurs="0" type="xs:string"/> <xs:element name="field2" minOccurs="0" type="xs:string"/> </xs:sequence>

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

{
"field1":"v4",
"field2":true
}
]
	</xs:complexType>
	</xs:element>
	<xs:element name="unnamed_2">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0" type="xs:decimal"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	<xs:element name="unnamed_3">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0" type="xs:boolean"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>

注　XML 宣言と schema 要素は省略しています。

JSON→XSD 変換では，これらの判定は再帰的に行われます。

次の例では，JSON Object は field2 の sub2 のデータ型が異なるため，異なる element 要素として定義されます。

表 12–9　再帰的判定の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

[
{
"field1": "v1",
"field2":{
"sub1":"s1",
"sub2":"s2"
}
},
{
	<xs:element name="root" csc:wrapped="true">
	<xs:complexType>
	<xs:choice minOccurs="0" maxOccurs="unbounded">
	<xs:element name="unnamed_1">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0">
	<xs:complexType>

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

"field1":"v2",
"field2":{
"sub1":"s3",
"sub2":123
}
}
]
	<xs:sequence>
	<xs:element name="sub1" minOccurs="0" type="xs:string"/>
	<xs:element name="sub2" minOccurs="0" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	<xs:element name="unnamed_2" minOccurs="0">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string"/>
	<xs:element name="field2" minOccurs="0">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="sub1" minOccurs="0" type="xs:string"/>
	<xs:element name="sub2" minOccurs="0" type="xs:decimal"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>

注　XML 宣言と schema 要素は省略しています。

JSON Object をマージするかどうかの判定は，JSON Object の出現順に実行され，最初にマージ可能と判定された JSON Object 同士をマージします。

例えば，次の例では，Object 配列要素の 1 番目の Object と 2 番目の Object，2 番目の Object と 3 番目の Object は，それぞれマージできますが，出現順序が早い 1 番目の Object と 2 番目の Object をマージし，3 番目の Object は独立した element 要素として定義されます。

表 12-10　Object 配列要素のマージ順序の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
[{	<xs:element name="root" csc:wrapped="true"> <xs:complexType>

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

"field1":"v1",
"field2":"v2"
},
{
"field1":"v3",
"field3":true
},
{
"field2":123,
"field3":false
}
]
	<xs:choice minOccurs="0" maxOccurs="unbounded">
	<xs:element name="unnamed_1">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" minOccurs="0" type="xs:string" />
	<xs:element name="field2" minOccurs="0" type="xs:string" />
	<xs:element name="field3" minOccurs="0" type="xs:boolean" />
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	<xs:element name="unnamed_2">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field2" minOccurs="0" type="xs:decimal" />
	<xs:element name="field3" minOccurs="0" type="xs:boolean" />
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:choice>
	</xs:complexType>
	</xs:element>

注　XML 宣言と schema 要素は省略しています。

(7)　null の変換

null は対象要素の nillable 属性に true が設定されます。配列要素の場合，配列要素内の element 要素のすべての nillable 属性に true が設定されます。対象要素が特定できない場合は，string 型の要素を生成し，nillable 属性に true が設定されます。

表 12–11　null の変換の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）
null	<xs:element name="root" type="xs:string" nillable="true"/>
[null]	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="unnamed_1" type="xs:string" nillable="true"/> </xs:choice> </xs:complexType> </xs:element>

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

[
  null,
  true,
  "abc"
]
	<xs:element name="root" csc:wrapped="true">
	  <xs:complexType>
	  <xs:choice minOccurs="0" maxOccurs="unbounded">
	  <xs:element name="unnamed_1" type="xs:boolean" nillable="true"/>
	  <xs:element name="unnamed_2" type="xs:string" nillable="true"/>
	  </xs:choice>
	  </xs:complexType>
	</xs:element>

[
  {
  "field1":"v1",
  "field2":"v2"
  },
  {
  "field1":null,
  "field2":"v2"
  }
]
	<xs:element name="root" csc:wrapped="true">
	  <xs:complexType>
	  <xs:choice minOccurs="0" maxOccurs="unbounded">
	  <xs:element name="unnamed_1">
	  <xs:complexType>
	  <xs:sequence>
	  <xs:element name="field1" minOccurs="0" type="xs:string" nillable="true"/>
	  <xs:element name="field1" minOccurs="0" type="xs:string" />
	  </xs:sequence>
	  </xs:complexType>
	  </xs:element>
	  </xs:choice>
	  </xs:complexType>
	</xs:element>

(8)　mixed contents の変換

フィールド名#text を持つ JSON Object は，mixed content の complexType 要素として生成されます。mixed content の complexType 要素の sequence 要素は，#text 以外のフィールドを子要素として持ちます。

表 12–12　mixed content の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

{
  "field1":"v1",
  "field2":"v2",
  "#text":"v3"
}
	<xs:element name="root" >
	  <xs:complexType mixed="true">
	  <xs:sequence>
	  <xs:element name="field1" minOccurs="0" type="xs:string"/>
	  <xs:element name="field2" minOccurs="0" type="xs:string"/>
	  </xs:sequence>
	  </xs:complexType>
	</xs:element>

mixed contents 内の要素データ（#text で指定されるデータ）は，データ変換のマッピング対象外です。

(9) 別名のフィールドの変換

JSON Object のフィールド名が，XML の要素タグとして使用できない場合，フィールド名の使用できない文字を「_（アンダースコア）」に置き換え，csd:jsonfield 属性に元のフィールド名が設定されます。置き換えた要素名が重複する場合，末尾に連番が付与されます。

表 12-13　フィールド別名の変換の例

入力する JSON ファイルの定義	生成する XSD（XML スキーマ）

{
"<field1>":"v1",
"2field":"v2",
"3field":"v3"
}
	<xs:element name="root" >
	<xs:complexType mixed="true">
	<xs:sequence>
	<xs:element name="_field1_" minOccurs="0" type="xs:string" csc:jsonfield="<field1> "/>
	<xs:element name="_field" minOccurs="0" type="xs:string" csc:jsonfield="2field"/>
	<xs:element name="_field1" minOccurs="0" type="xs:string" csc:jsonfield="3field"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>

12.3.2　JSON→XML 変換の規則

JSON→XML 変換の規則について説明します。

表 12-14　JSON→XML 変換の規則

入力する JSON ファイルの定義	対応する XSD（XML スキーマ）	XML 変換結果
false	<xs:element name="root" type="xs:boolean"/>	<root>false</root>
null	<xs:element name="root" type="xs:string" nillable="true"/>※1	<root xsi:nil="true"/>
true	<xs:element name="root" type="xs:boolean"/>	<root>true</root>
123	<xs:element name="root" type="xs:decimal"/>	<root>123</root>
"abc"	<xs:element name="root" type="xs:string"/>	<root>abc</root>

{
"field":"value"
}
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
		<root>
		<field>value</field>
		</root>

入力する JSON ファイルの定義	対応する XSD（XML スキーマ）	XML 変換結果
	<xs:element name="field" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element>	
["value1", "value2"]	<xs:element name="field" type="xs:string" maxOccurs="unbounded"/>※2	<field>value1</field> <field>value2</field>※3
	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※1 <xs:element name="unnamed_" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element>	<root> <unnamed_1> value1 </unnamed_1> <unnamed_1> value2 </unnamed_1> </root>
[["value1","value2"], ["value3","value4"]]	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※1 <xs:element name="unnamed_1" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※2 <xs:element name="unnamed_2" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:choice> </xs:complexType> </xs:element>	<root> <unnamed_1> <unnamed_2> value1 </unnamed_2> <unnamed_2> value2 </unnamed_2> </unnamed_1> <unnamed_1> <unnamed_2> value3 </unnamed_2> <unnamed_2> value4 </unnamed_2> </unnamed_1> </root>

注　多次元配列および配列で始まる JSON 形式データは，アンラップ変換はできません。

注※1　type は任意タイプを指定できます。

注※2　maxOccurs の指定値は 2 以上の最大配列要素数です。unbounded は無制限の配列要素数を表します。

注※3　ルート要素または多次元配列の場合，変換できません。

(1) ルート要素および choice 要素の選択

JSON→XML 変換では，複数のルート要素の中から最適なルート要素を選択し変換します。また，choice 要素内の複数の要素についても，同様に最適な要素を選択し変換します。

表 12–15 ルート要素の選択の例

入力する JSON ファイルの定義	対応する XSD（XML スキーマ）	XML 変換結果
123	<xs:schema …> <xs:element name="strData" type="xs:string"/> <xs:element name="intData" type="xs:integer"/> </xs:schema>	<intData>123</intData>
"abcd"		<strData>abcd</strData>

要素の選択は，ルート要素または choice 要素内の要素ごとに，JSON 形式データの各要素とのマッチングを行い，マッチ度を計算して，最大のマッチ度を持つ XML スキーマを選択します。最大のマッチ度を持つ要素が複数存在する場合，次の基準で選択します。

- ルート要素の場合：ルート要素名が文字コード順で最も早い要素を選択します。
- choice 要素内の要素の場合：先に定義された要素を選択します。

最大のマッチ度は，次の表のマッチ度を合計した値となります。

表 12–16 マッチ度の条件

条件	マッチ度
対応する attribute のデータ型と JSON value 型および突き合わせ条件に合致する attribute の属性値に変換する。	4
対応する element のデータ型と JSON value 型および突き合わせ条件に合致する element の要素値に変換する。	4
対応する attribute のデータ型と JSON value 型に合致する attribute の属性値に変換する。	3
対応する element のデータ型と JSON value 型に合致する element の要素値に変換する。	3
対応する attribute のデータ型の JSON value 型が string の場合に string 以外の JSON value を attribute の属性値に変換する。	2
対応する element のデータ型の JSON value 型が string の場合に string 以外の JSON value を element の要素値に変換する。	2
対応する element が存在しない。	0

XML スキーマのデータ型および突き合わせ条件は，「12.3.2(2) データ型の突き合わせ条件」を参照してください。

(2)　データ型の突き合わせ条件

JSON→XML 変換では，JSON 形式データを XML スキーマの仕様で定義された組み込みデータ型と突き合わせを行い，条件に合うデータ型を選択して変換します。組み込みデータ型以外の派生したデータタイプは，派生元の組み込みデータ型を摘出して突き合わせます。組み込みデータ型を派生して加えた pattern や enumeration などの制約条件は考慮されないため，不正な JSON 形式データが入力された場合，変換結果の XML データのバリデーションで制約違反が検出されるおそれがあります。

なお，JSON→XSD 変換で生成した XSD ファイルは，組み込みデータ型だけを使用するため，このような制約違反は発生しません。

XML スキーマの組み込みデータ型と JSON 形式データの突き合わせ条件を次に示します。

表 12–17　データ型の突き合わせ条件

組み込みデータ型	JSON のデータ型	突き合わせ条件
anyURI	string	0 文字以上の任意の文字列
base64Binary	string	0 文字以上の base64 文字列
boolean	true, false	－
date	string	dateTime から時分秒を除く 例：2025-01-01, 2025-01-01Z, 2025-01-01+09:00
dateTime	string	{年}-{月}-{日}T (({時}:{分}:{秒}) | {24 時}){タイムゾーン}? 例：2025-01-01T09:00:00Z, 2025-01-01:24:00:00
decimal	number	[+-]?(({数字}+)|({数字}+.{数字}+)|(.{数字}+)) 例：123, 123.456, .456
integer	number	小数点以下を含まない digit
long	number	-9223372036854775808～9223372036854775807
int	number	-2147483648～2147483647
short	number	-32768～32767
byte	number	-128～127
nonNegativeInteger	number	0 以上の integer
positiveInteger	number	1 以上の integer
unsignedLong	number	0～18446744073709551615
unsignedInt	number	0～4294967295
unsignedShort	number	0～65535
unsignedByte	number	0～255
nonPositiveInteger	number	0 以下の integer

組み込みデータ型	JSON のデータ型	突き合わせ条件
negativeInteger	number	-1 以下の integer
double	number	Double.valueOf で例外が発生しない。
duration	string	Duration.parse()で例外が発生しない。 例：P1Y, P1M1D, P1DT5M
float	number	Float.valeOf で例外が発生しない。
gDay	string	---{日}{タイムゾーン}?
gMonth	string	--{月}{タイムゾーン}?
gMonthDay	string	--{月}-{日}{タイムゾーン}?
gYear	string	{年}{タイムゾーン}?
gYearMonth	string	{年}-{月}{タイムゾーン}?
hexBinary	string	0 文字以上の 16 進文字列
NOTATION	string	XML QName
QName	string	XML QName
string	string	0 文字以上の任意の文字列
normalizedString	string	CR，LF，TAB を含まない任意の文字列
token	string	normalizedString でかつ先頭および末尾にスペースがなく，2 文字以上の連続したスペースがない文字列
language	string	[a-zA-Z]{1,8}(-[a-zA-Z0-9]{1,8})*
Name	string	XML Name
NCName	string	XML NCName
ENTITY	string	XML NCName
ID	string	XML NCName
IDREF	string	XML NCName（対象の ID の存在有無は突き合わせない）
NMTOKEN	string	XML Nmtoken
time	string	(({時}:{分}:{秒}) | {24 時}){タイムゾーン}?
ENTITIES	string	スペースで区切られた複数の ENTITY
IDREFS	string	スペースで区切られた複数の IDREF
NMTOKENS	string	スペースで区切られた複数の NMTOKEN

（凡例）－：該当しない。

注

- JSON が null の場合，nillable に true が指定されている要素をデータ型に関係なくマッチします。

- ENTITIE，IDREFS，NMTOKENS などの list タイプのデータは，データ変換ではマッピングできません。

(3) simpleType と list の突き合わせ

XML スキーマの simpleType 要素内の list 要素の構文では，指定したデータ型を持つ値のリスト（スペース区切り）を要素値として定義できます。変換元の JSON 形式データが，文字列の場合，変換元文字列をスペース区切りで部分文字列に分割し，分割したすべての部分文字列が itemType 属性で指定されたデータ型に合うことを確認します。

ただし，simpleType 要素内の list 要素のデータは，データ変換でマッピングできません。

表 12-18 simpleType 要素内の list 要素の変換例

入力する JSON ファイルの定義	対応する XSD（XML スキーマ）	XML 変換結果
"1 2 3 4 5"	<xs:schema …> <xs:element name="root"> <xs:simpleType> <xs:list itemType="xs:integer"/> </xs:simpleType> </xs:element> </xs:schema>	<root>1 2 3 4 5</root>
"1 2 3 4 abc"		変換されません。 エラーリストに登録されます。

(4) サポートする XML スキーマ要素

JSON→XML 変換がサポートする XML スキーマ要素の組み合わせを次に示します。

表 12-19 サポートする XML スキーマ要素

XML スキーマ要素	親の XML スキーマ要素	子の XML スキーマ要素
<xs:all>	<xs:group>, <xs:complexType>, <xs:restriction>, <xs:extension>	<xs:element>
<xs:attribute>	<xs:attributeGroup>, <xs:schema>, <xs:complexType>, <xs:restriction>, <xs:extension>	<xs:simpleType>
<xs:attributeGroup>	<xs:attributeGroup>, <xs:complexType>, <xs:schema>, <xs:restriction>, <xs:extension>	<xs:attributeGroup>, <xs:anyAttribute>, <xs:attribute>
<xs:choice>	<xs:group>, <xs:choice>, <xs:sequence>, <xs:complexType>, <xs:restriction>, <xs:extension>	<xs:choice>, <xs:sequence>, <xs:element>, <xs:group>
<xs:complexContent>	<xs:complexType>	<xs:restriction>, <xs:extension>
<xs:complexType>	<xs:element>, <xs:schema>, <xs:redefine>	<xs:all>, <xs:choice>, <xs:sequence>, <xs:simpleContent>, <xs:complexContent>,

XML スキーマ要素	親の XML スキーマ要素	子の XML スキーマ要素
		<xs:group>, <xs:attribute>, <xs:attributeGroup>, <xs:anyAttribute>
<xs:element>	<xs:schema>, <xs:choice>, <xs:all>, <xs:sequence>, <xs:group>	<xs:complexType>, <xs:simpleType>
<xs:extension>	<xs:simpleContent>, <xs:complexContent>	<xs:group>, <xs:sequence>, <xs:choice>, <xs:attribute>, <xs:attributeGroup>
<xs:group>	<xs:schema>, <xs:choice>, <xs:sequence>, <xs:complexType>, <xs:restriction>, <xs:extension>	<xs:all>, <xs:choice>, <xs:sequence>
<xs:include>	<xs:schema>	なし
<xs:import>	<xs:schema>	なし
<xs:list>	<xs:simpleType>	<xs:simpleType>
<xs:redefine>	<xs:schema>	<xs:simpleType>, <xs:complexType>, <xs:group>, <xs:attributeGroup>
<xs:restriction>※	<xs:simpleType>, <xs:simpleContent>, <xs:complexContent>	<xs:simpleType>, <xs:all>, <xs:choice>, <xs:sequence>, <xs:attribute>, <xs:group>, <xs:attributeGroup>, <xs:anyAttribute>
<xs:schema>	なし	<xs:include>, <xs:import>, <xs:redefine>, <xs:simpleType>, <xs:complexType>,<xs:group>, <xs:attributeGroup>, <xs:element>, <xs:attribute>
<xs:sequence>	<xs:group>, <xs:choice>, <xs:sequence>, <xs:complexType>, <xs:restriction>, <xs:extension>	<xs:element>, <xs:group>, <xs:choice>, <xs:sequence>
<xs:simpleContent>	<xs:complexType>	<xs:restriction>, <xs:extension>
<xs:simpleType>	<xs:attribute>, <xs:element>, <xs:list>, <xs:restriction>, <xs:schema>, <xs:union>	<xs:list>, <xs:restriction>, <xs:union>
<xs:union>	<xs:simpleType>	<xs:simpleType>

注　この表に記載がない XML スキーマ要素は無視されます。

注※　restriction 自体は base 属性で指定する元のデータタイプを取得するためにサポートされますが，子要素による制約の追加はサポートされません。

(5)　属性データへの変換

JSON→XML 変換では，JSON Object のフィールド名に一致する要素がなく，属性名に一致する場合は属性データに変換されます。

JSON Object のフィールド名が<要素名>@<属性名>形式で，要素名に対応する要素に属性名に一致する属性が定義されている場合は，該当する属性の属性値として変換されます。対象要素が配列の場合は，<要素名>@<属性名>配列の対応する配列インデックス要素の属性値として変換されます。JSON Object のフィールド名が<要素名>@<属性名>でも，対応する要素および属性が XSD ファイルに定義されていない場合は変換されません。

JSON→XSD 変換で生成した XSD ファイルでは属性データは定義されません。属性データへの変換を行う場合は，XSD ファイルを編集し，属性データを定義する必要があります。

表 12-20　属性データへの変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML]変換結果

{
"data":"value1",
"attr1":"value2",
"attr2":"value3"
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="data" type="xs:string"/>
	</xs:sequence>
	<xs:attribute name="attr1" type="xs:string"/>
	<xs:attribute name="attr2" type="xs:string"/>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root attr1="value2" attr2="value3">
		<data>value1</data>
		</root>

{
"field1":"value1",
"field1@attr":"a1",
"field2":"value2"
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1">
	<xs:complexType>
	<xs:simpleContent>
	<xs:extension base="xs:string">
	<xs:attribute name="attr" type="xs:string"/>
	</xs:extension>
	</xs:simpleContent>
	</xs:complexType>
	</xs:element>
	<xs:element name="field2" type="xs:string" />
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<field1 attr="a1">
		value1
		</field1>
		<field2>value2</field2>
		</root>

{	<xs:schema …>	<root>

入力する JSON ファイルの定義	対応する XML スキーマ	XML]変換結果
"field1":["value1", "value2", "value3"], "field1@attr":["a1", null, "a3"], "field2":"value2" }	<xs:element name="root"> <xs:complexType> <xs:sequence> <xs:element name="field1" maxOccurs="unbounded"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="attr1" type="xs:string"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element> <xs:element name="field2" type="xs:string" /> </xs:sequence> </xs:complexType> </xs:element> </xs:schema>	<field1 attr="a1"> value1 </field1> <field1> value1 </field1> <field1 attr="a3"> value1 </field1> <field2>value2</field2> </root>

(6) 配列データの変換（アンラップ変換およびラップ変換）

配列データをアンラップ変換およびラップ変換で変換できます。

アンラップ変換は，配列要素のデータ型が単一の場合にだけ実行できます。

表 12–21 アンラップ変換の例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"field":[
"data1",
"data2"
]
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" type="xs:string" maxOccurs="unbounded"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<field>data1</field>
		<field>data2</field>
		</root>

ラップ変換では，csc:wrapped 属性に true が設定された要素の子要素（choice 要素内の element 要素）から，最適な element 要素を選択し変換します。

表 12–22　ラップ変換の例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"field":[
"data1",
"data2"
]
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" csc:wrapped="true">
	<xs:complexType>
	<xs:choice maxOccurs="unbounded">
	<xs:element name="item" type="xs:string"/>
	</xs:choice>
	</xs:complexType>
	</xs:element>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<field>
		<item>data1</item>
		<item>data2</item>
		</field>
		</root>

(a)　ルート要素が配列の場合

JSON 形式データのルート要素が配列の場合，アンラップ変換では変換できません。csc:wrapped 属性に true を設定し，ラップ変換を行う必要があります。配列のルート要素をアンラップ変換した場合は，エラーとなります。

表 12–23　ルート配列の変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果
["data1", "data2"]	<xs:schema …> <xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element name="item" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	<root> <item>data1</item> <item>data2</item> </root>

(b)　多次元配列の場合

多次元配列の場合，アンラップ変換では変換できません。csc:wrapped 属性に true を設定しラップ変換を行う必要があります。多次元配列をアンラップ変換した場合は，エラーとなります。

表 12–24　多次元配列の変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果
[["data1","data2"], ["data3","data4"]]	<xs:schema …> <xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element name="mid" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element name="item" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	<root> <mid> <item>data1</item> <item>data2</item> </mid> <mid> <item>data3</item> <item>data4</item> </mid> </root>

(c)　混合タイプの配列の場合

配列要素に異なるデータタイプを持つ配列は，アンラップ変換できません。csc:wrapped 属性に true を設定し，ラップ変換を行う必要があります。

表 12–25　混合タイプ配列の変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

[
"data1",
true,
{
"field1":"data2",
"field2":"data3"
}
]
	<xs:schema …>
	<xs:element name="root" csc:wrapped="true">
	<xs:complexType>
	<xs:choice maxOccurs="unbounded">
	<xs:element name="strItem" type="xs:string"/>
	<xs:element name="boolItem" type="xs:boolean"/>
	<xs:element name="objItem">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" type="xs:string"/>
	<xs:element name="field2" type="xs:string"/>
	</xs:sequence>
		<root>
		<strItem>data1</strItem>
		<boolItem>true</boolItem>
		<objItem>
		<field1>data2</field1>
		<field2>data3</field2>
		</objItem>
		</root>

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果
	</xs:complexType> </xs:element> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	

(7) 要素の出現順序の維持

要素の出現順序は，JSON Object のフィールドの出現順に関係なく，XML スキーマに定義された順序に従って XML データに変換されます。

表 12–26 要素順序を維持した変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"givenname":"Taro",
"surname":"Yamada",
"id":1111
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="surname" type="xs:string"/>
	<xs:element name="givenname" type="xs:string"/>
	<xs:element name="id" type="xs:integer"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<surname>Yamada</surname>
		<givenname>Taro</givenname>
		<id>1111</id>
		</root>

(8) 要素名の別名を使用した変換

XML スキーマの element 要素の csc:jsonfield 属性に JSON フィールド名を指定すると，JSON Object の JSON フィールド名を，XML スキーマの element 要素の name 属性で指定された要素タグを持つ XML 要素に変換できます。

表 12–27 別名を使用した変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果
{	<xs:schema …>	<root>

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果
"<field>":"data" }	<xs:element name="root"> <xs:complexType> <xs:sequence> <xs:element csc:jsonfield="<field>" name="field" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element> </xs:schema>	<field>data</field> </root>

(9) default または fixed の指定

JSON 形式データに対応するフィールドがない element 要素が必須フィールドの場合（minOccurs 属性が 1 以上，または minOccurs 属性の指定がない），default または fixed の指定があるときはそこで指定された値に変換されます。

default または fixed を指定した要素の変換は，JSON→XML 変換をしたあと XML→JSON 変換を行った場合，異なる JSON 形式に変換されます。

表 12–28　default または fixed の変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"name":"dogName"
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="name" type="xs:string"/>
	<xs:element minOccurs="1" fixed="animal" name="type" type="xs:string"/>
	<xs:element minOccurs="1" default="dog" name="kind" type="xs:string"/>
	<xs:element minOccurs="0" default="French bull" name="subkind" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<name>dogName</name>
		<type>animal</type>
		<kind>dog</kind>
		</root>

注　subkind は，必須要素ではないため変換されません。

(10) mixed content の変換

XML スキーマで mixed content と指定された要素に対応するフィールドに，「#text」サブフィールドが存在する場合，mixed content の値となります。

mixed content のデータは，データ変換でマッピングできません。

表 12–29 mixed content の変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"field":"data1",
"#text":"data2"
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType mixed="true">
	<xs:sequence>
	<xs:element name="field" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<field>data1</field>
		data2
		</root>

(11) 重複フィールドの変換

JSON Object に重複したフィールド名が存在する場合，最後に出現したフィールド値だけが有効となります。

表 12–30 重複フィールドの変換例

入力する JSON ファイルの定義	対応する XML スキーマ	XML 変換結果

{
"field":"data1",
"field":"data2"
}
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		<root>
		<field>data2</field>
		</root>

12.3.3 XML→JSON 変換の規則

XML→JSON 変換の規則について説明します。

表 12-31　XML→JSON 変換の規則

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
<root>false</root>	<xs:element name="root" type="xs:boolean"/>	false
<root>true</root>		true
<root>123</root>	<xs:element name="root" type="xs:decimal"/>	123
<root>123</root>	<xs:element name="root" type="xs:string"/>	"123"
<root xsi:nil="true"/>	<xs:element name="root" type="xs:string" nillable="true"/>※1	null
<root>true</root>	<xs:element name="root" type="xs:boolean"/>	true

<root>
<field>value</field>
</root>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
		{
		"field":"value"
		}

<root>
<field>value1</field>
<field>value2</field>
</root>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" maxOccurs="unbounded" type="xs:string"/>※2
	</xs:sequence>
	</xs:complexType>
	</xs:element>
		{
		"field":[
		"value1",
		"value2"
		]
		}

<root> <unnamed_1> value1 </unnamed_1> <unnamed_1> value2 </unnamed_1> </root>	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※2 <xs:element name="unnamed_1" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element>	["value1", "value2"]
<root> <d1> <d2>value1</d2> <d2>value2</d2> </d1> <d1> <d2>value1</d2> <d2>value2</d2>	<xs:element name="root"> <xs:complexType> <xs:sequence> <xs:element name="d1" maxOccurs="unbounded">※2 <xs:complexType> <xs:sequence>	{ "d1":[{"d2":["value1", "value2"]}, { "d2":[

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
</d1> </root>	<xs:element name="d2" maxOccurs="unbounded" type="xs:string"/>※2 </xs:sequence> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>	"value3", "value4"]}] }
	<xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※2 <xs:element name="d1" csc:wrapped="true"> <xs:complexType> <xs:choice maxOccurs="unbounded">※2 <xs:element name="d2" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:choice> </xs:complexType> </xs:element>	[["value1","value2"], ["value3","value4"]]

注※1　type は任意タイプを指定できます。

注※2　maxOccurs の指定値は 2 以上の最大配列要素数です。unbounded は無制限の配列要素数を表します。

(1)　XML 要素の変換

XML 要素ごとに対応する XML スキーマ要素を突き合わせて変換します。XML 要素に対応する XML スキーマ要素がない場合，該当する XML 要素をスキップし，エラーリストに登録します。

ルート要素にマッチする XML スキーマのルート要素がない場合の API メソッドの返り値は null となります。

(2)　データ型の対応

対応する XML スキーマ要素のデータタイプによって，変換される JSON のデータ型が異なります。

変換元 XML タイプと変換先の JSON のデータ型の対応を次に示します。

表 12–32　変換元 XML タイプと変換先の JSON のデータ型の対応

変換元 XML タイプ	変換先 JSON データ型
anyURI	string

変換元 XML タイプ	変換先 JSON データ型
base64Binary	string
boolean	true，false
date	string
dateTime	string
decimal	number
integer	number
long	number
int	number
short	number
byte	number
nonNegativeInteger	number
positiveInteger	number
unsignedLong	number
unsignedInt	number
unsignedShort	number
unsignedByte	number
nonPositiveInteger	number
negativeInteger	number
double	number
duration	string
float	number
gDay	string
gMonth	string
gMonthDay	string
gYear	string
gYearMonth	string
hexBinary	string
NOTATION	string
QName	string
string	string
normalizedString	string

変換元 XML タイプ	変換先 JSON データ型
token	string
language	string
Name	string
NCName	string
ENTITY	string
ID	string
IDREF	string
NMTOKEN	string
time	string
ENTITIES	string
IDREFS	string
NMTOKENS	string

XML スキーマ仕様で定義されていないデータタイプについては，派生元の組み込みデータ型を摘出して変換します。

complexType は，次のデータ型に変換されます。

表 12–33　complexType の変換規則

変換元 XML タイプ	変換先 JSON データ型
<complexType><sequence>	object
<complexType><all>	object
<complexType><choice>	• maxOccurs>1 の場合は，JSON Array • maxOccurs≦1 の場合は，子要素のデータ型 sequence の子要素として定義された場合以外は子要素の要素名は無視されます。

complexType の choice 要素の変換例を次に示します。

表 12–34　complexType の choice 要素の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
<root> <f1>0</f1> </root>	<xs:schema …> <xs:element name="root"> <xs:complexType> <xs:choice>	0

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	<xs:element name="f1" type="xs:byte"/> <xs:element name="f2" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	
<root> <f2>1</f2> </root>	<xs:schema …> <xs:element name="root"> <xs:complexType> <xs:choice> <xs:choice> <xs:element name="f1" type="xs:byte"/> </xs:choice> <xs:choice> <xs:element name="f2" type="xs:string"/> </xs:choice> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	"1"
<root> <f2>1</f2> </root>	<xs:schema …> <xs:element name="root"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element name="f1" type="xs:byte"/> <xs:element name="f2" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	["1"]

<root>
<f1>0</f1>
<f2>2</f2>
<f1>1</f1>
<f2>3</f2>
</root>
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:choice maxOccurs="unbounded">
	<xs:element name="f1" type="xs:byte"/>
	<xs:element name="f2" type="xs:string"/>
	</xs:choice>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
		{
		"f1":[0,1],
		"f2":["2","3"]
		}

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	</xs:schema>	

(3) simpleType の list 要素の変換

simpleType の list 要素では，指定したデータ型を持つ値のリスト（スペース区切り）を要素値として定義できます。変換元の XML のデータ型が list 要素に指定されている場合，XML データ型に関係なく JSON の string に変換されます。

表 12–35 simpleType の list 要素の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
<root>1 2 3 4 5</root>	<xs:schema …> <xs:element name="root"> <xs:simpleType> <xs:list itemType="xs:integer"/> </xs:simpleType> </xs:element> </xs:schema>	"1 2 3 4 5"

(4) simpleType の union 要素の変換

simpleType の union 要素では，複数のデータ型を持つ要素を定義できます。XML→JSON 変換では，複数のデータ型を定義順に要素値が変換できるか確認し，最初に変換できると判定したデータ型で JSON データを変換します。simpleType の union 要素を使用した XML スキーマを利用し，JSON→XML 変換後に XML→JSON 変換を行った場合，変換結果は元の JSON 形式データと異なる場合があります。

表 12–36 simpleType の union 要素の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
<root>1</root>	<xs:schema …> <xs:element name="root"> <xs:simpleType> <xs:union memberTypes="*xs:byte xs:string*"/> </xs:simpleType> </xs:element> </xs:schema>	1
<root>abc</root>		"abc"
<root>1</root>	<xs:schema …> <xs:element name="root"> <xs:simpleType>	"1"

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	<xs:union memberTypes="*xs:string xs:byte*"/> </xs:simpleType> </xs:element> </xs:schema>	

注　*斜体*は，複数のデータ型を持つ要素を表します。

(5)　サポートする XML スキーマ要素

XML→JSON 変換がサポートする XML スキーマ要素の組み合わせは，JSON→XML 変換がサポートする XML スキーマ要素と同じです。

JSON→XML 変換がサポートする XML スキーマ要素については，「12.3.2(4) サポートする XML スキーマ要素」を参照してください。

(6)　属性データへの変換

complexType の sequence 要素に，attribute 属性が定義されている場合，属性名をフィールド名，属性値をフィールド値とした JSON Object のフィールドに変換されます。

complexType の simpleContent 要素に，attribute 属性が定義されている場合，「要素名@属性名」をフィールド名，属性値をフィールド値とした JSON Object のフィールドに変換されます。該当する要素が複数あり，配列にまとめる場合，属性値も配列にまとめられます。だたし，simpleContent 要素がルート要素の場合，属性は変換されません。attribute 属性が定義されていない場合は変換されません。エラーリストにも登録されません。

属性名と同名のフィールドがある場合，変換は行われないで，該当する属性がエラーリストに登録されます。

表 12–37　属性データへの変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果

<root attr1="123"
attr2="value3">
<data>value1</data>
</root>
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="data" type="xs:string"/>
	</xs:sequence>
	<xs:attribute name="attr1" type="xs:decimal"/>
	<xs:attribute name="attr2" type="xs:string"/>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		{
		"data":"value1",
		"attr1":123,
		"attr2":"value3"
		}

	<xs:schema …>	{

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	<xs:element name="root"> <xs:complexType> <xs:sequence> <xs:element name="data" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element> </xs:schema>	"data":"value1" }

<root>
<field1 attr="a1">
value1
</field1>
<field2>value2</field2>
</root>
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1">
	<xs:complexType>
	<xs:simpleContent>
	<xs:extension base="xs:string">
	<xs:attribute name="attr" type="xs:string"/>
	</xs:extension>
	</xs:simpleContent>
	</xs:complexType>
	</xs:element>
	<xs:element name="field2" type="xs:string" />
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		{
		"field1":"value1",
		"field1@attr":"a1",
		"field2":"value2"
		}

<root>
<field1 attr="a1">
value1
</field1>
<field1>
value1
</field1>
<field1 attr="a3">
value1
</field1>
<field2>value2</field2>
</root>
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field1" maxOccurs="unbounded">
	<xs:complexType>
	<xs:simpleContent>
	<xs:extension base="xs:string">
	<xs:attribute name="attr1" type="xs:string"/>
	</xs:extension>
	</xs:simpleContent>
	</xs:complexType>
	</xs:element>
		{
		"field1":[
		"value1",
		"value2",
		"value3"
		],
		"field1@attr":[
		"a1",
		null,
		"a3"
		],
		"field2":"value2"
		}

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	<xs:element name="field2" type="xs:string" /> </xs:sequence> </xs:complexType> </xs:element> </xs:schema>	

(7) 配列データの変換（アンラップ変換およびラップ変換）

配列データをアンラップ変換およびラップ変換で変換できます。

XML 要素に対応する element 要素の maxOccurs 属性に 2 以上が設定されている場合，JSON array に変換できます（アンラップ変換）。

配列の変換は対象の element 要素のタイプが，complexType の choice 要素の場合，子要素の値を親要素の JSON array に変換できます（ラップ変換）。

表 12–38 配列の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果

<root>
<field>data1</field>
<field>data2</field>
</root>
	<!--アンラップ変換 -->
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType>
	<xs:sequence>
	<xs:element name="field" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		{
		"field":[
		"data1",
		"data2"
		]
		}

<root>
<field>data1</field>
</root>
		{
		"field":[
		"data1"
		]
		}

<root> <field>data1</field> <field>data2</field> </root>	<!--ラップ変換 --> <xs:schema …> <xs:element name="root" csc:wrapped="true"> <xs:complexType> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="field" type="xs:string"/> </xs:choice> </xs:complexType> </xs:element> </xs:schema>	["data1", "data2"]
<root> <field>data1</field> </root>		["data1"]

(8) 要素名の別名を使用した変換

element 要素の csc:jsonfield 属性に「JSON フィールド名」を設定すると，対応する要素の JSON フィールド名を「JSON フィールド名」に変更できます。

表 12-39 別名を使用した変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果

<root>
  <_field_>data</_field_>
</root>
	<xs:schema …>
	  <xs:element name="root">
	  <xs:complexType>
	  <xs:sequence>
	  <xs:element csc:jsonfield="<field>" name="_field_" type="xs:string"/>
	  </xs:sequence>
	  </xs:complexType>
	  </xs:element>
	</xs:schema>
		{
		  "<field>":"data"
		}

(9) default または fixed の変換

element 要素に default または fixed 属性が指定され，次の条件を満たす場合，JSON フィールドが生成されます。

- 必須フィールド（minOccurs 属性が 1 以上，または minOccurs 属性の指定がない）
- 対応する XML 要素が存在しない

default または fixed 属性を指定した XML スキーマを利用し，JSON→XML 変換後に XML→JSON 変換を行った場合，変換結果は元の JSON 形式データと異なる場合があります。

表 12-40 default または fixed の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果

<root>
  <name>dogName</name>
</root>
	<xs:schema …>
	  <xs:element name="root">
	  <xs:complexType>
	  <xs:sequence>
	  <xs:element name="name" type="xs:string"/>
	  <xs:element fixed="animal" name="type" type="xs:string"/>
	  <xs:element default="dog" name="kind" type="xs:string"/>
	  <xs:element minOccurs="0" default="French bull" name="subkind" type="xs:string"/>
		{
		  "name":"dogName",
		  "type":"animal",
		  "kind":"dog"
		}

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果
	</xs:sequence> </xs:complexType> </xs:element> </xs:schema>	

(10) mixed content の変換

mixed 属性に true が指定された ComplexType 要素に対応する XML 要素に，テキストデータが設定されている場合，JSON Object の#text フィールドに該当するテキストデータが変換されます。

表 12–41 mixed content の変換例

入力する XML ファイルの定義	対応する XML スキーマ	JSON 変換結果

<root>
<field>data1</field>
data2
</root>
	<xs:schema …>
	<xs:element name="root">
	<xs:complexType mixed="true">
	<xs:sequence>
	<xs:element name="field" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>
	</xs:element>
	</xs:schema>
		{
		"field":"data1",
		"#text":"data2"
		}

12.4　拡張モードの JSON-XML 変換 API の使用例

拡張モードの JSON-XML 変換 API の使用例を次に示します。

12.4.1　JSON→XSD 変換

JSON-XML 変換 API を使用して，JSON 形式のデータを XSD 形式のデータに変換する場合の実装例を次に示します。

```
package jp.co.Hitachi.soft.csc.common.converter.jsonxml.sample;

import java.io.IOException;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Properties;

import org.apache.ws.commons.schema.XmlSchema;

import jp.co.Hitachi.soft.csc.common.converter.jsonxml.JSONtoXSDConverter;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.JSONtoXSDConverterFactory;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.XmlSchemaUtils;

public class GenXSD {
    /**
     * JSONファイルからXML Schemaファイルを作成する。
     * @param jsonPath 入力JSONファイルパス名
     * @param xsdPath 出力XSDファイルパス名
     */
    public void toXSDFile(String jsonPath, String xsdPath) {
        try {
            // JSONtoXSDConverterFactoryインスタンスを取得する
            JSONtoXSDConverterFactory factory = JSONtoXSDConverterFactory.getConverterFactor
y();

            // JSONtoXSDConverterの実装インスタンスを取得する
            JSONtoXSDConverter converter = factory.getConverter(null);

            // 入力JSONファイルからXmlSchemaインスタンスを生成する。
            XmlSchema schema = converter.toSchema(Path.of(jsonPath));

            // OuputStreamを取得し
            try(OutputStream os = Files.newOutputStream(Path.of(xsdPath))) {
                // XSDファイルを書込む
                XmlSchemaUtils.printXmlSchema(schema, os);
            }
        } catch (IOException e) {
            // JSONファイルが読込めない、JSONファイルの形式が不正、
            // XSDファイルに書込めない等
            e.printStackTrace();
        }
    }
```

```
    /**
     * JSONファイルからXML Schemaファイルを作成する。
     * ルートタグ、無名タグをカスタマイズする
     * @param jsonPath 入力JSONファイルパス名
     * @param xsdPath 出力XSDファイルパス名
     */
    public void toXSDFile(String jsonPath, String xsdPath, String rootTag, String unnamedTag
) {
        try {
            // JSONtoXSDConverterFactoryインスタンスを取得する
            JSONtoXSDConverterFactory factory = JSONtoXSDConverterFactory.getConverterFactor
y();

            // オプションデータ格納用mapの宣言
            Properties options = null; // <- Map<Object,Object>でもOK
            if((rootTag != null && rootTag.length() > 0) ||
               (unnamedTag != null && unnamedTag.length() > 0)) {
                // rootTag, unnamedTagのいずれかが設定されている。
                options = new Properties();
                if(rootTag != null && rootTag.length() > 0) {
                    // rootTagをカスタマイズする場合
                    options.put(
                            JSONtoXSDConverterFactory.ROOT_TAG_OPTION_KEY,
                            rootTag);
                }
                if(unnamedTag != null && unnamedTag.length() > 0) {
                    // 無名タグをカスタマイズする場合
                    options.put(
                            JSONtoXSDConverterFactory.UNNAMED_TAG_PREFIX_OPTION_KEY,
                            unnamedTag);
                }
            }

            // JSONtoXSDConverterの実装インスタンスを取得する
            JSONtoXSDConverter converter = factory.getConverter(options);

            // 入力JSONファイルからXmlSchemaインスタンスを生成する。
            XmlSchema schema = converter.toSchema(Path.of(jsonPath));

            // OuputStreamを取得し
            try(OutputStream os = Files.newOutputStream(Path.of(xsdPath))) {
                // XSDファイルを書込む
                XmlSchemaUtils.printXmlSchema(schema, os);
            }
        } catch (ReflectiveOperationException e) {
            // JSONtoXSDConverterFactoryが取得できなかった。
            // デフォルトの状態では発生しない。
            // jsonxml.JSONtoXSDConverterFactoryClass システムプロパティに
            // 設定したクラスが存在しない場合に発生する。
            e.printStackTrace();
        } catch (IOException e) {
            // JSONファイルが読込めない、JSONファイルの形式が不正、
            // XSDファイルに書込めない等
            e.printStackTrace();
        }
    }
}
```

12.4.2 JSON→XML 変換

JSON-XML 変換 API を使用して，JSON 形式のデータを XML 形式のデータに変換する場合の実装例を次に示します。

```
package jp.co.Hitachi.soft.csc.common.converter.jsonxml.sample;

import java.io.IOException;
import java.nio.file.Path;
import java.util.List;

import javax.xml.parsers.ParserConfigurationException;

import org.apache.ws.commons.schema.XmlSchema;
import org.w3c.dom.Document;

import jp.co.Hitachi.soft.csc.common.converter.jsonxml.ConversionErrorInfo;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.JSONtoXMLConverter;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.JSONtoXMLConverterFactory;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.XSDLoader;

public class CnvXML {
    /**
     * JSONファイルをXML Schemaを使用してXMLに変換する。
     * @param jsonPath JSONファイルのパス
     * @param xsdPath  XML Schemaファイルのパス
     * @return XML Document
     */
    Document toXML(String jsonPath, String xsdPath) {
        try {
            // XML Schemaを読込む
            XmlSchema schema = XSDLoader.getXmlSchemaLoader()
                    .load(Path.of(xsdPath));

            // JSON->XML変換コンテキストのファクトリクラスの取得
            JSONtoXMLConverterFactory factory =
                    JSONtoXMLConverterFactory.getConverterFactory();

            // JSON->XML変換コンテキストの取得
            JSONtoXMLConverter converter = factory.getConverter(schema,null);

            // JSON->XML変換の実行
            Document doc = converter.toDocument(Path.of(jsonPath));

            // 変換エラーリストを取得
            List<ConversionErrorInfo> errors = converter.getLastErrors();
            if(errors == null || errors.size() == 0) {
                // 変換エラーはない
            } else {
                // 各々の変換エラーを出力
                // ConversionErrorInfo.toString()は以下の文字列
                // エラーメッセージ（field=フィールド名, value=値）
                // ConversionErrorInfoのgetterを利用して、
                // カスタムメッセージを出力する事も可能
                for(ConversionErrorInfo error : errors) {
                    System.err.println(error);
```

```
                }
            }
            return doc;
        } catch(IOException e) {
            // XML Schema, JSONの読込み失敗
        } catch (ParserConfigurationException e) {
            // XML Documentが生成できない。
             // DocumentBuilderFactory#newDocumentBuilderがスローした
        }
        return null;
    }
}
```

12.4.3 XML→JSON 変換

JSON-XML 変換 API を使用して，XML 形式のデータを JSON 形式のデータに変換する場合の実装例を次に示します。

```
package jp.co.Hitachi.soft.csc.common.converter.jsonxml.sample;

import java.io.File;
import java.io.IOException;
import java.nio.file.Path;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.ws.commons.schema.XmlSchema;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

import jp.co.Hitachi.soft.csc.common.converter.jsonxml.ConversionErrorInfo;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.XMLtoJSONConverter;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.XMLtoJSONConverterFactory;
import jp.co.Hitachi.soft.csc.common.converter.jsonxml.XSDLoader;

public class ConvJSON {
    /**
     * XMLファイルの内容をJSONに変換しファイルに書き出す。
     *
     * @param jsonPath 出力JSONファイルパス
     * @param xmlPath 変換対象XMLファイルのパス
     * @param xsdPath  変換対象XMLに合致したXSDファイルのパス
     */
    void xml2json(String jsonPath, String xmlPath, String xsdPath) {
        try {
            // XMLファイルからJsonNodeに変換する
            JsonNode json = xml2json(xmlPath,xsdPath);
            if(json == null) {
                // 変換結果が空(XMLとXSDが合致しないなど)
```

```
                // エラー内容は既にエラー出力に書込まれている。
            } else {
                // 書き出し用 ObjectMapperを取得
                ObjectMapper mapper = new ObjectMapper();
                // JsonNodeをファイルに書き出す
                mapper.writerWithDefaultPrettyPrinter().writeValue(new File(jsonPath), json)
;
            }
        } catch (ParserConfigurationException |
                SAXException |
                IOException e) {
            // 変換失敗
            e.printStackTrace();
        }
    }

    /**
     * XMLファイルの内容をJSONに変換する。
     * @param xmlPath 変換対象XMLファイルのパス
     * @param xsdPath  変換対象XMLに合致したXSDファイルのパス
     * @return JSONノード
     * @throws ParserConfigurationException DocumentBuilderが生成できない
     * @throws IOException XMLファイルまたはXSDファイルの読込み時にIO例外が発生した
     * @throws SAXException XMLファイルの解析中に例外が発生した。
     */
    JsonNode xml2json(String xmlPath, String xsdPath)
            throws ParserConfigurationException, SAXException, IOException {
        // XMLの読込み
        DocumentBuilderFactory builderfactory =
                DocumentBuilderFactory.newDefaultInstance();
        DocumentBuilder builder = builderfactory.newDocumentBuilder();
        Document doc = builder.parse(xmlPath);
        // XSDの読込み
        XmlSchema schema = XSDLoader.getXmlSchemaLoader()
                .load(Path.of(xsdPath));
        // 変換実行
        return xml2json(doc,schema);
    }

    /**
     * XML DOMをXML Schemaを利用してJsonNodeに変換する
     * @param doc 変換対象XML DOMインスタンス
     * @param schema 変換対象に合致したXML Schemaインスタンス
     * @return 変換結果
     */
    JsonNode xml2json(Document doc, XmlSchema schema) {
        // XML->JSON変換コンテキストのファクトリの取得
        XMLtoJSONConverterFactory factory =
                XMLtoJSONConverterFactory.getConverterFactory();
        // 変換コンテキストの取得
        // オプションを指定する場合は第２引数に指定する。
        XMLtoJSONConverter converter = factory.getConverter(schema, null);
        // 変換実行
        JsonNode node = converter.toJSON(doc);
        if(converter.getLastErrors() == null || converter.getLastErrors().size() == 0) {
            // エラーは無い
        } else {
            // 何らかのエラーがあるのでエラー出力に書き出す。
```

```
            for(ConversionErrorInfo err : converter.getLastErrors()) {
                System.err.println(err.toString());
            }
        }
        return node;
    }
}
```

13 API を使用したデータ変換

この章では，API を使用したデータ変換について説明します。

13.1　データ変換 API を使用したデータ変換

データ変換 API は，サービスプラットフォームが提供するデータ変換基盤を利用するためのインターフェースを定義した API です。データ変換 API を利用することで，HCSC サーバの構築やビジネスプロセスを定義しなくても，データ変換ができるようになります。

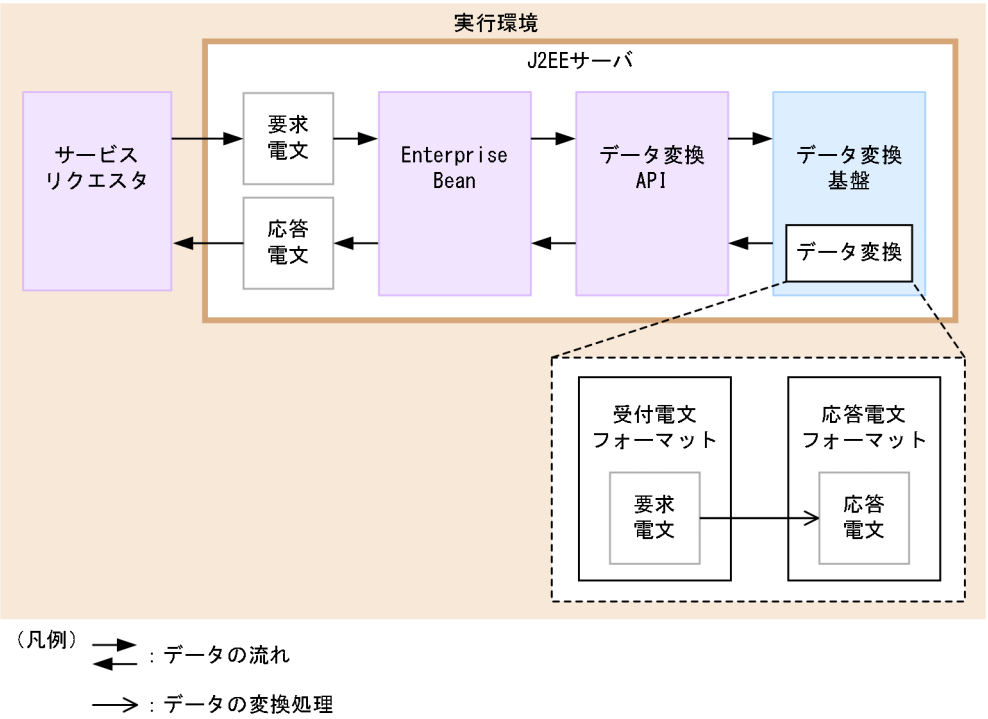
13.1.1　データ変換 API を使用したデータ変換の概要

データ変換 API を利用してデータ変換をする場合は，データ変換 API を呼び出す Enterprise Bean の実装が必要です。Enterprise Bean を J2EE サーバ上で稼働させ，サービスリクエスタから Enterprise Bean を経由してデータ変換 API を呼び出します。

なお，Enterprise Bean は，Stateless Session Bean または Message-driven Bean を使用してください。

データ変換 API を利用したデータ変換の実現方式を次の図に示します。

図 13–1　データ変換 API を利用したデータ変換の実現方式



データ変換 API の詳細については，「13.1.3　データ変換 API の詳細」を参照してください。

13.1.2　データ変換 API の機能

データ変換 API では次の機能が利用できます。

- データ変換

- キャッシュ操作
- グループ管理
- プロパティの変更
- 障害対策

以降で，それぞれの機能について説明します。

(1)　データ変換

サービスプラットフォームが提供するデータ変換アクティビティに相当する機能です。データ変換 API では，入力としてデータ変換前の電文を受け取り，出力としてデータ変換後の電文を返します。

(a)　データ形式

データ変換 API では，データ変換前，およびデータ変換後の電文のデータ形式に XML とバイナリを指定できます。

データ変換前，およびデータ変換後のデータ形式の組み合わせのパターンを次に示します。

- XML→XML
- XML→バイナリ
- バイナリ→XML
- バイナリ→バイナリ

(b)　データ変換前とデータ変換後の電文の数量関係

データ変換 API ではデータ変換前とデータ変換後の電文が 1 対 1 または N 対 1 となるデータ変換をサポートします。

(c)　キャッシュの利用

データ変換 API を使用したデータ変換でも，データ変換定義で使用するフォーマット定義，およびデータ変換定義のキャッシュ機能を利用できます。定義情報がキャッシュされている場合，キャッシュを利用してデータ変換をします。また，データ変換時にキャッシュされていない定義情報が指定された場合は，データ変換の際にキャッシュとして登録されます。

キャッシュ処理の詳細については，「13.1.2(2)　キャッシュ操作」を参照してください。

(d)　データ変換基盤がサポートする機能の利用

データ変換基盤で実装されている機能は，データ変換 API を使用したデータ変換でも使用することができます。ただし，次の機能については制限事項があります。

- キャッシュ処理

データ変換 API を使用したデータ変換では，バイナリフォーマット定義およびデータ変換定義だけがキャッシュの対象となります。XML 電文オブジェクト，および XML Schema はキャッシュの対象にできません。

キャッシュ処理の詳細については，「13.1.2(2)　キャッシュ操作」を参照してください。

- XML Schema の指定

  要求電文，および応答電文のデータ形式が XML の場合，データ変換では XML Schema を使用しません。データ変換のメソッドの引数に XML Schema の定義ファイルが指定されても参照しません。また，XML Schema のキャッシュも登録されません。ただし，データ変換のメソッドの引数に，定義ファイルの種別の指定は必要です。データ変換のメソッドの定義ファイルの種別の引数に XSD を，キャッシュ名の引数に null をそれぞれ指定した場合，キャッシュ名には製品の固定値（CSCDT_XMLFormatCache）が設定されます。

  キャッシュ名は，データ変換基盤のメッセージ出力時の詳細情報や性能解析トレース出力時の付加情報として利用されます。

(2)　キャッシュ操作

データ変換 API では，cscprecache コマンドに相当するフォーマット定義，およびデータ変換定義のキャッシュ処理を実現する API を提供します。データ変換で使用するフォーマット定義，およびデータ変換定義の定義情報を事前にキャッシュすることで，初回のリクエスト処理の性能向上を実現できます。

(a)　キャッシュ名

定義情報のキャッシュは，キャッシュの種別ごとに一意となるキャッシュ名で管理されます。定義情報を登録および使用する場合は，データ変換 API の呼び出し元で必ずキャッシュ名を指定します。

キャッシュ名で使用できる文字種に制限はありませんが，N 対 1 変換の場合，要求電文のフォーマット定義ファイルに使用するキャッシュ名は，マッピング定義で指定した変換元のスキーマ論理名と同じ名称にする必要があります。

変換元のスキーマ論理名の詳細については，「7.3.1(1)(b)　Eclipse のメニューから表示する場合」を参照してください。

また，InputDataDefinition クラスの配列はマッピング定義で指定した変換元スキーマと同じ定義順序にする必要があります。

(b)　キャッシュの登録対象

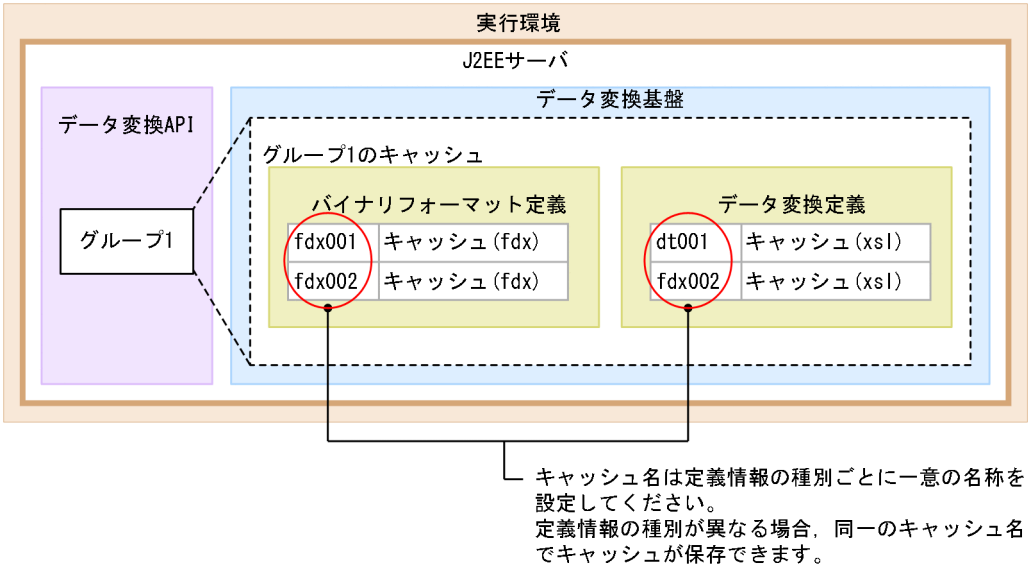
データ変換 API では，次の定義情報をキャッシュできます。

- バイナリフォーマット定義
- データ変換定義（XSLT スタイルシート）

(c)　キャッシュの保存方法

データ変換 API のキャッシュの保存方法を次の図に示します。

図 13–2　キャッシュの保存方法



キャッシュの上限数，および生存期間は次のとおりです。

●キャッシュの上限数

キャッシュの上限数はデータ変換 API のシステムプロパティで管理されます。

データ変換 API のシステムプロパティの詳細については，「13.1.2(4)　プロパティの変更」を参照してください。

●キャッシュの生存期間

登録されたキャッシュは，キャッシュの登録数が上限数に達するまで保持されます。キャッシュがあふれた場合，最後に参照されてから最も時間の経過しているキャッシュが削除されます。

(d)　キャッシュ操作の機能概要

データ変換 API のキャッシュ操作では，次の機能が利用できます。

- キャッシュの登録
- キャッシュの存在確認
- キャッシュの削除
- キャッシュ名の一覧取得
- キャッシュの一括削除

以降で，それぞれの機能について説明します。

●キャッシュの登録

フォーマット定義，およびデータ変換定義の定義情報をキャッシュします。

- 登録されていないキャッシュにこの機能を使用する場合

  キャッシュを新規に登録します。

- すでに登録されているキャッシュにこの機能を使用する場合
  キャッシュを更新します。

キャッシュの登録処理で XML Schema が指定された場合は，メソッド引数不正エラーが発生します。

●キャッシュの存在確認

フォーマット定義，およびデータ変換定義の定義情報がキャッシュされているかどうか確認します。キャッシュの存在確認の結果は真偽値で返します。キャッシュの存在確認処理で XML Schema が指定された場合はメソッド引数不正エラーが発生します。

●キャッシュの削除

キャッシュの登録機能によって登録された定義情報のキャッシュを削除します。登録されていないキャッシュに対してこの機能を使用すると，キャッシュの削除処理をしないで正常終了します。キャッシュの削除処理で XML Schema が指定された場合はメソッド引数不正エラーが発生します。

●キャッシュ名の一覧取得

フォーマット定義，およびデータ変換定義のキャッシュ名の一覧を取得します。キャッシュ名の一覧は，Set<String>型で返します。キャッシュ名の順序性は保証されません。キャッシュが登録されていない場合は，空の Set を返します。また当処理で XML Schema が指定された場合はメソッド引数不正エラーが発生します。

●キャッシュの一括削除

フォーマット定義，およびデータ変換定義の定義情報のキャッシュをすべて削除します。キャッシュが登録されていない場合は，キャッシュの削除処理をしないで正常終了します。

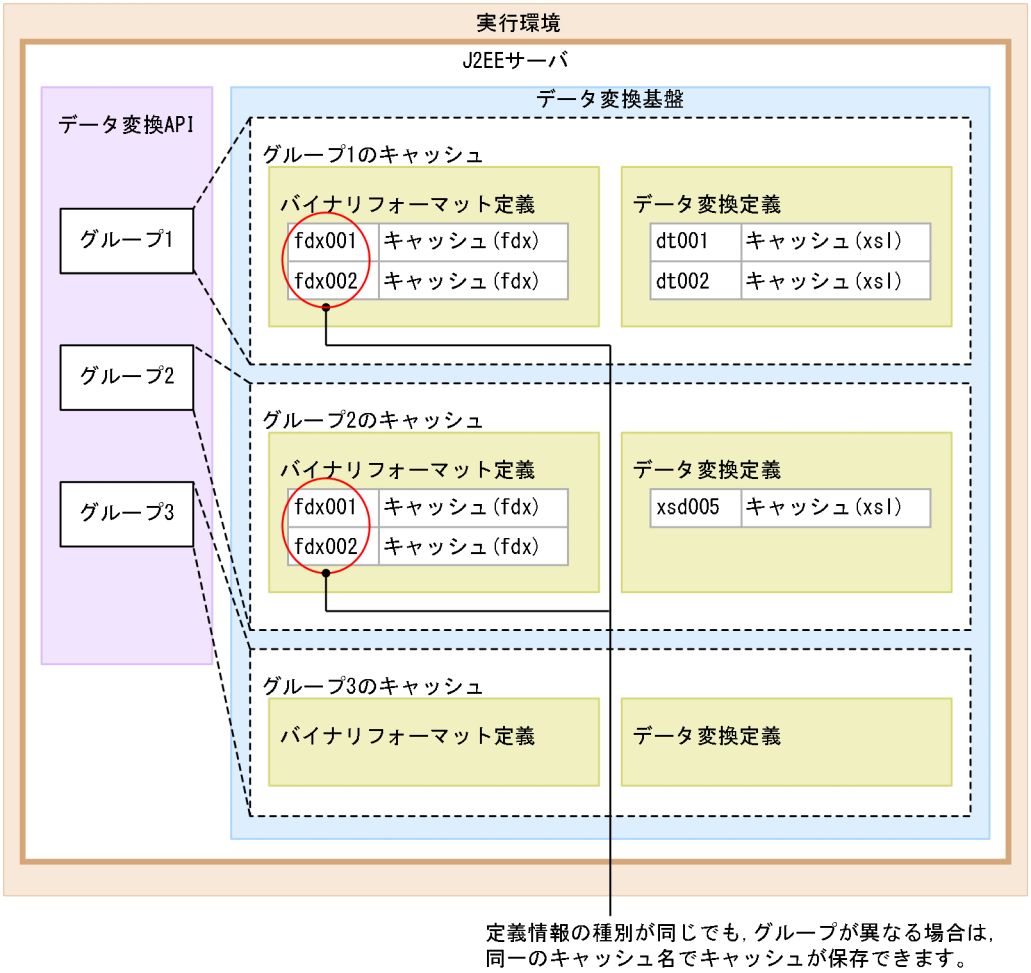
(3) グループ管理

データ変換 API ではデータ変換に必要な設定（キャッシュやプロパティ）を複数保持できます。この設定を保持する単位を変換グループと呼びます。変換グループを使用することで，データ変換の設定（キャッシュやプロパティ）を業務ごとに管理できます。変換グループに設定された情報は，J2EE サーバを停止するか，変換グループを破棄するまで保持されます。変換グループはグループ名で識別します。グループ名は J2EE サーバで一意とする必要があります。

データ変換 API を使用する場合は，最初のデータ変換 API の呼び出し元で任意のグループ名を指定し，データ変換やキャッシュ操作を行うオブジェクトを取得してから設定してください。

データ変換 API の変換グループごとの設定の保存イメージを次の図に示します。

図 13-3　グループごとのキャッシュの保存イメージ



(a)　グループ名

グループ名で使用できる文字種は，半角英数字，アンダーバー（_），アットマーク（@），ハイフン（-），およびドル記号（$）です。最大文字列長は 32 バイトです。

(b)　変換グループに関する機能

データ変換 API の変換グループで使用できる機能を次に示します。

- 変換グループの取得
- 変換グループの削除

それぞれの機能について説明します。

●変換グループの取得

指定された変換グループ名の変換グループを取得します。指定された変換グループが登録されていない場合は，新規に生成します。

●変換グループの削除

指定された変換グループ名の変換グループを削除します。変換グループが存在しない場合は，変換グループの削除処理をしないで，正常終了します。

(4) プロパティの変更

データ変換 API，およびデータ変換基盤の動作はシステムプロパティで変更できます。システムプロパティは usrconf.properties（J2EE サーバ用ユーザプロパティファイル）に指定します。

ここでは，データ変換 API のシステムプロパティについて説明します。

データ変換基盤で定義しているシステムプロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「3.19　データ変換単体テストコマンドで利用するファイル」を参照してください。

usrconf.properties の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」，および，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.3.9　J2EE サーバ用ユーザプロパティファイル（usrconf.properties）を設定する」を参照してください。

データ変換 API が定義するシステムプロパティを次に示します。

表 13–1　データ変換 API が定義するシステムプロパティ

キー名称	デフォルト値	説明
csc.dt.formatdef.maxcache.num.<グループ名>	2,147,483,647	バイナリフォーマット定義をキャッシュできる数を 1～2,147,483,647 の整数で指定します。このプロパティの指定値は，バイナリフォーマット定義のキャッシュの上限値となります。 設定値はデフォルト値を推奨します。
csc.dt.transformdef.maxcache.num.<グループ名>	2,147,483,647	データ変換定義をキャッシュできる数を 1～2,147,483,647 の整数で指定します。 設定値はデフォルト値を推奨します。
csc.dt.telegram.undefined.characterCode.<グループ名>	ERROR	バイナリデータの文字コード変換処理で未定義の文字コードが検出された際の動作を，ERROR または REPLACE のどちらかの値で指定します。 • ERROR を指定した場合，検出時にエラーとなり処理を終了します。 • REPLACE を指定した場合，未定義部分の文字コードを空白の文字コードに置換して処理します。
csc.dt.codeTable.rootPath.<グループ名>	<サービスプラットフォームのインストールディレクトリ>¥¥CSC¥¥userlib¥¥codetables	Code Converter で使用するコード変換テーブル格納パスのルートパスを絶対パスで指定します。ディレクトリ区切りは「/」または「¥¥」を使用します。存在しないディレクトリまたはファイルを設定した場合，グループの生成に失敗します。
csc.dt.debugtrace	OFF	デバッグ情報の出力有無を ON または OFF のどちらかの値を指定します。

キー名称	デフォルト値	説明
csc.dt.debugtrace	OFF	• ON を指定した場合，デバッグ情報を出力します。 • OFF を指定した場合，デバッグ情報を出力しません。 未定義，未設定，または不正な値を設定した場合，デフォルト値を使用します。
csc.dt.debugtrace.filepath	<サービスプラットフォームのログ出力ディレクトリ>¥¥csc	デバッグ情報の出力先パスを絶対パスで指定します。 ディレクトリ区切りは「/」または「¥¥」を使用します。 未定義，未設定，または存在しないディレクトリを指定した場合，デフォルト値を使用します。 デフォルトの出力先のディレクトリが存在しない場合に作成されます。 csc.dt.debugtrace で ON を指定した場合だけ有効となります。
csc.dt.debugtrace.filenum	4	デバッグ情報ファイルの最大面数を 1～16 の整数で指定します。 未定義，未設定，または不正な値を設定した場合，デフォルト値を使用します。 csc.dt.debugtrace で ON を指定した場合だけ有効となります。
csc.dt.debugtrace.filesize	20971524	デバッグ情報ファイルの 1 面当たりの最大サイズを 4,096～2,147,483,647（単位：バイト）の整数で指定します。 未定義，未設定，または不正な値を設定した場合，デフォルト値を使用します。 csc.dt.debugtrace で ON を指定した場合だけ有効となります。
csc.dt.pool.DocumentBuilder.initialSize	0	DOM パーサプールを作成するときの初期割り当て数を 0～2,147,483,647 の整数で指定します。 0 を指定すると，J2EE サーバの起動時に DocumentBuilder プールは生成されません。
csc.dt.pool.DocumentBuilder.maxSize	64	DOM パーサの最大プール数を 0～2,147,483,647 の整数で指定します。 HCSC サーバ内のすべてのリクエスト受付の最大実行数と，データ変換 API から実行されるリクエストの最大実行数の総和以上の値を指定します。

(5)　障害対策

データ変換 API では，データ変換やキャッシュ操作で障害が発生した場合，メッセージログにメッセージを出力し，データ変換 API 専用の例外クラスのオブジェクトをスローします。また，データ変換基盤でトレース情報を性能解析トレースへ出力します。

データ変換 API 専用の例外クラスの詳細については，「13.1.3(5)　例外クラスの詳細」を参照してください。

データ変換 API で出力するログおよびトレースの種類について説明します。

(a)　メッセージログ

データ変換 API を操作するときのインフォメーション情報や障害時のエラー情報をメッセージログへ出力します。

メッセージログの詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.4.1　メッセージログ」を参照してください。

(b)　性能解析トレース

データ変換基盤の性能解析トレースを出力します。データ変換の前後で出力される 0x9A43 および 0x9A44 のイベント ID の付加情報（OPT（ASCII））に，要求電文，応答電文，およびデータ変換定義のキャッシュ名を出力します。

データ変換基盤の性能解析トレースの取得ポイントの詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.4.3　性能解析トレース」を参照してください。データ変換基盤のイベント ID および付加情報を使用した性能解析の調査方法については，「13.1.6　データ変換時のボトルネックの特定」を参照してください。

13.1.3　データ変換 API の詳細

サービスプラットフォームが提供するデータ変換 API は，パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」に所属します。

(1)　パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」に所属するクラス

パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」に所属するインターフェースおよびクラスの一覧を次に示します。

●インターフェース

表 13–2　インターフェースの一覧（パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」）

インターフェース名	説明
DataTransformer	データ変換基盤のデータ変換処理を呼び出すインターフェースです。
FormatDataCacheController	データ変換基盤のキャッシュ操作を実行するインターフェースです。

●クラス

表 13–3　クラスの一覧（パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」）

クラス名	説明
DataTransformerCacheManager	データ変換 API のインスタンスを管理するクラスです。

クラス名	説明
FormatDefinition	フォーマット定義およびデータ変換定義を格納するクラスです。
InputDataDefinition	変換元データおよび要求電文の FormatDefinition オブジェクトを格納するクラスです。

●列挙型

表 13–4　列挙型の一覧（パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」）

列挙型名	説明
FormatDataKind	フォーマット定義およびデータ変換定義の種別を表す列挙型です。

●例外クラス

表 13–5　例外クラスの一覧（パッケージ「jp.co.Hitachi.soft.csc.dt.api.transformer」）

クラス名	説明
CacheNotFoundException	データ変換処理でフォーマット定義およびデータ変換定義のキャッシュが見つからなかったことを通知する例外クラスです。
DataTransformException	データ変換 API の例外を扱うクラスです。

(2)　インターフェースの詳細

(a)　DataTransformer インターフェース

データ変換基盤のデータ変換処理を呼び出すインターフェースです。

●クラス定義

```
public interface DataTransformer
```

説明

transform メソッドの引数に要求電文，フォーマット定義ファイルの情報，データ変換定義ファイルの情報を指定し，変換後の応答電文をバイト配列で返します。

なお，サポートするデータ変換の形式は次のとおりです。

- XML→XML
- XML→バイナリ
- バイナリ→XML
- バイナリ→バイナリ

このクラスは，バイナリフォーマット定義（FDX），およびデータ変換定義（XSLT）のキャッシュを保持します。それぞれのキャッシュは別々に管理されます。キャッシュを保持できる上限値は，システムプロパティで設定します。

データ変換処理では，バイナリフォーマット定義，およびデータ変換定義のキャッシュを参照し，データ変換をします。キャッシュを登録していない場合は，transform メソッドに対して定義ファイルの情報を指定してください。

未登録のキャッシュは，データ変換処理時にキャッシュが登録されます。

XML Schema はデータ変換で使用されません。そのため，transform メソッドに対して XML Schema の定義ファイルを指定する必要はありません。また，このクラスで XML Schema のキャッシュは保持しません。

インスタンスを破棄する際には，stop メソッドを一度だけ呼び出してください。

注意事項

要求電文と応答電文の N 対 1 のデータ変換には対応していません。

●メソッド

DataTransformer インターフェースのメソッドを次の表に示します。

表 13-6　DataTransformer インターフェースのメソッド

メソッド名	説明
transform	引数で指定した変換元のデータに対し，データ変換処理を実行し，変換後のデータを返します。 記述形式には次の 2 種類があります。 • N 対 1 変換（1 対 1 変換も含みます） • 1 対 1 変換限定（推奨しません）
stop	停止処理を実行します。

●transform（N 対 1 変換が可能）

要求電文と応答電文の N 対 1 変換（1 対 1 変換も含む）が可能です。

引数で指定した変換元のデータに対し，データ変換処理を実行し，変換後のデータを返します。

このメソッドはスレッドセーフです。

形式

```
byte[] transform(InputDataDefinition[] inDataDefinition,
                 FormatDefinition outFormatDefinition,
                 FormatDefinition transformFormatDefinition)
          throws CacheNotFoundException,
                 DataTransformException
```

引数

- inDataDefinition

  変換元データと要求電文のフォーマット定義ファイルを格納するオブジェクトを指定します。

  要求電文のフォーマット定義ファイルで指定するキャッシュ名には，ユニークな名称を設定してください。

  InputDataDefinition クラスの配列は，マッピング定義で指定した変換元スキーマと同じ定義順序にする必要があります。

- outFormatDefinition

  応答電文のフォーマット定義ファイルを格納するオブジェクトを指定します。

- transformFormatDefinition

  データ変換定義ファイルを格納するオブジェクトを指定します。

説明

要求電文と応答電文の N 対 1 変換のデータ変換に対応します。

変換元データおよび要求電文のフォーマット定義ファイルの情報は，配列で複数指定できます。

FormatDefinition クラスでは，定義ファイルの種別，キャッシュ名，定義ファイル（byte[]）を設定します。定義ファイル（byte[]）については，FormatDataCacheController クラスでキャッシュを登録している場合，設定が省略できます。

また，定義ファイルの種別が XML Schema の場合は設定が省略できます。

注意事項

- 種別，キャッシュ名，および定義ファイルに，null または無効な値は指定できません（定義ファイルの種別が XSD の場合のキャッシュ名を除きます）。
- 定義ファイルの種別がバイナリフォーマット定義またはデータ変換定義である場合，transform メソッド実行時にキャッシュがなく，定義ファイル（byte[]）の設定も省略されていると，例外（CacheNotFoundException）が発生します。
- N 対 1 変換の場合，要求電文のフォーマット定義ファイルに使用するキャッシュ名は，マッピング定義で指定した変換元のスキーマ論理名と同じ名称にする必要があります。

戻り値

変換後のデータを返します。

例外

- データ変換処理で定義ファイルのキャッシュが見つからない場合，例外（CacheNotFoundException）をスローし，エラーメッセージ（KDEC41002-E）をメッセージログに出力します。
- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41013-E）をメッセージログに出力します。不正となる引数を次に示します。

  ・各引数が null のとき（定義ファイルの種別が XSD の場合のキャッシュ名の引数を除く）

  ・空の配列が指定されたとき

  ・キャッシュ名が空文字列のとき

  ・定義ファイルの種別が不正なとき
- データ変換処理で例外が発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41005-E）をメッセージログに出力します。

- キャッシュの登録処理が失敗した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41006-E）をメッセージログに出力します。

●transform（1 対 1 変換限定）

要求電文と応答電文の 1 対 1 変換だけ可能です。推奨しません。

引数で指定した変換元のデータに対し，データ変換処理を実行し，変換後のデータを返します。

このメソッドはスレッドセーフです。

形式

```
byte[] transform(byte[] data,
                 FormatDefinition inFormatDefinition,
                 FormatDefinition outFormatDefinition,
                 FormatDefinition transformFormatDefinition)
          throws CacheNotFoundException,
                 DataTransformException
```

引数

- data

  変換元データを指定します。

- inFormatDefinition

  要求電文のフォーマット定義ファイルを格納するオブジェクトを指定します。

- outFormatDefinition

  応答電文のフォーマット定義ファイルを格納するオブジェクトを指定します。

- transformFormatDefinition

  データ変換定義ファイルを格納するオブジェクトを指定します。

説明

要求電文と応答電文の 1 対 1 のデータ変換に対応します。

データ変換をするには，要求電文と応答電文のフォーマット定義，およびデータ変換定義の情報が必要です。

定義ファイルの情報は FormatDefinition クラスで指定します。

FormatDefinition クラスでは，定義ファイルの種別，およびキャッシュ名の設定が必須です。定義ファイルの種別が XSD で，キャッシュ名が null の場合，キャッシュ名には製品の固定値（CSCDT_XMLFormatCache）が設定されます。

バイナリフォーマット定義，およびデータ変換定義のキャッシュを登録していない場合は，定義ファイルの情報も FormatDefinition クラスに指定してください。

注意事項

- 種別，キャッシュ名，および定義ファイルに，null または無効な値は指定できません（定義ファイルの種別が XSD の場合のキャッシュ名を除きます）。
- XML Schema の定義ファイルの情報は，このメソッドで使用されません。定義ファイルの種別とキャッシュ名を，FormatDefinition クラスで指定してください。

戻り値

変換後のデータを返します。

例外

- データ変換処理で定義ファイルのキャッシュが見つからない場合，例外（CacheNotFoundException）をスローし，エラーメッセージ（KDEC41002-E）をメッセージログに出力します。
- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。不正となる引数を次に示します。
  ・各引数が null のとき（定義ファイルの種別が XSD の場合のキャッシュ名の引数を除く）
  ・空の配列が指定されたとき
  ・キャッシュ名が空文字列のとき
  ・定義ファイルの種別が不正なとき
- データ変換処理で例外が発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41005-E）をメッセージログに出力します。
- キャッシュの登録処理が失敗した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41006-E）をメッセージログに出力します。

●stop

停止処理を実行します。

形式

```
void stop()
```

(b) FormatDataCacheController インターフェース

データ変換基盤のキャッシュ操作を実行するインターフェースです。

バイナリフォーマット定義（FDX），データ変換定義（XSLT）のキャッシュに関する操作を提供します。

●クラス定義

```
public interface FormatDataCacheController
```

説明

キャッシュの登録や削除の対象となるキャッシュの情報は，FormatDefinition クラスに指定します。FormatDefinition インスタンスの生成時に，定義ファイルの種別やキャッシュ名を設定し，このクラスの各メソッドに FormatDefinition インスタンスを設定してください。

注意事項

このクラスで，XML Schema のキャッシュ操作はできません。このクラスの各メソッドの引数に XML Schema を指定すると例外が発生します。

●メソッド

FormatDataCacheController インターフェースのメソッドを次の表に示します。

表 13-7　FormatDataCacheController インターフェースのメソッド

メソッド名	説明
clear	フォーマット定義およびデータ変換定義のキャッシュをすべて削除します。
contains	フォーマット定義およびデータ変換定義のキャッシュが含まれている場合に true を返します。
getCacheNameList	フォーマット定義およびデータ変換定義のキャッシュ名の一覧を返します。
register	フォーマット定義およびデータ変換定義のキャッシュを登録します。
remove	フォーマット定義およびデータ変換定義のキャッシュを削除します。

●clear

フォーマット定義およびデータ変換定義のキャッシュをすべて削除します。キャッシュが一つも登録されていない場合は，削除対象が存在しないので，処理をしないで終了します（正常終了となります)。

このメソッドはスレッドセーフです。

形式

```
void clear()
          throws DataTransformException
```

引数

ありません。

戻り値

ありません。

例外

データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。

●contains

フォーマット定義およびデータ変換定義のキャッシュが含まれている場合に true を返します。

このメソッドはスレッドセーフです。

形式

```
boolean contains(FormatDefinition formatDefinition)
          throws DataTransformException
```

引数

- formatDefinition

  定義ファイルの情報を格納するオブジェクトを指定します。

説明

FormatDefinition クラスに設定した種別，およびキャッシュ名と一致するキャッシュから検索します。

注意事項

種別，およびキャッシュ名に null または無効な値は指定できません。

戻り値

キャッシュが登録されている場合は true を返します。

例外

- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。不正となる引数を次に示します。

  ・各引数が null のとき

  ・キャッシュ名が空文字列のとき

  ・定義ファイルの種別が不正なとき

●getCacheNameList

フォーマット定義およびデータ変換定義のキャッシュ名の一覧を返します。

このメソッドはスレッドセーフです。

形式

```
java.util.Set<java.lang.String> getCacheNameList(FormatDataKind formatDataKind)
          throws DataTransformException
```

引数

- formatDataKind

  定義ファイルの種別を指定します。

注意事項

- XML Schema（XSD）を表す種別は指定できません。
- キャッシュがひとつも登録されていない場合は空の Set を返します。
- このメソッドでは一覧の順序性を保証しません。

戻り値

キャッシュ名の Set を返します。

例外

- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。

●register

フォーマット定義およびデータ変換定義のキャッシュを登録します。

このメソッドはスレッドセーフです。

形式

```
void register(FormatDefinition formatDefinition)
throws DataTransformException
```

引数

- formatDefinition

  定義ファイルの情報を格納するオブジェクトを指定します。

説明

FormatDefinition クラスに設定した種別，キャッシュ名，定義ファイルのバイト配列を参照し，キャッシュとして登録します。

指定されたキャッシュ名でキャッシュが登録されている場合はキャッシュを更新します。N 対 1 変換の場合，要求電文のフォーマット定義ファイルに使用するキャッシュ名は，マッピング定義で指定した変換元のスキーマ論理名と同じ名称にする必要があります。

注意事項

種別，キャッシュ名，および定義ファイルのバイト配列に，null または無効な値は指定できません。

戻り値

ありません。

例外

- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。不正となる引数を次に示します。

  ・各引数が null のとき

  ・空の配列が指定されたとき

  ・キャッシュ名が空文字列のとき

  ・定義ファイルの種別が不正なとき

- キャッシュの登録処理が失敗した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41006-E）をメッセージログに出力します。

●remove

フォーマット定義およびデータ変換定義のキャッシュを削除します。指定された種別，キャッシュ名でキャッシュが登録されていない場合は，削除対象が存在しないので，処理をしないで終了します（正常終了となります）。

このメソッドはスレッドセーフです。

形式

```
void remove(FormatDefinition formatDefinition)
throws DataTransformException
```

引数

- formatDefinition

  定義ファイルの情報を格納するオブジェクトを指定します。

説明

FormatDefinition クラスに設定した種別，キャッシュ名と一致するキャッシュを削除します。

注意事項

種別，およびキャッシュ名に null または無効な値は指定できません。

戻り値

ありません。

例外

- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。不正となる引数を次に示します。

  ・各引数が null のとき

  ・キャッシュ名が空文字列のとき

  ・定義ファイルの種別が不正なとき

(3) クラスの詳細

(a) DataTransformerCacheManager クラス

データ変換 API のインスタンスを管理するクラスです。

●クラス定義

```
public class DataTransformerCacheManager
extends java.lang.Object
```

説明

このクラスは，DataTransformer インスタンスを複数保持し，グループ名という一意の識別子で管理します。

データ変換処理を呼び出す前に，getDataTransformer メソッドにグループ名を指定し，DataTransformer インスタンスを取得してください。データ変換処理は DataTransformer インスタンスから呼び出します。

DataTransformer インスタンスは，getDataTransformer メソッドの呼び出し時にキャッシュに存在しない場合，生成されます。

DataTransformer インスタンスは，データ変換基盤のキャッシュとロガーを保持します。キャッシュ数の上限数などの値はシステムプロパティ値が適用されます。

値を変更する場合は，システムプロパティを変更し，DataTransformer インスタンスを再作成してください。

FormatDataCacheController インスタンスは，データ変換基盤のキャッシュに対する操作を提供します。グループ名に関連する DataTransformer インスタンスにアクセスし，キャッシュの登録や削除をします。

キャッシュ処理を呼び出す前に，createFormatDataCacheController メソッドを呼び出し，FormatDataCacheController インスタンスを生成してください。

●メソッド

DataTransformerCacheManager クラスのメソッドを次の表に示します。

表 13-8　DataTransformerCacheManager クラスのメソッド

メソッド名	説明
createFormatDataCacheController	FormatDataCacheController インスタンスを生成します。
getDataTransformer	指定されたグループ名に関連付けられている DataTransformer インスタンスを返します。
removeDataTransformer	指定されたグループ名に関連付けられている DataTransformer インスタンスを削除します。

●createFormatDataCacheController

FormatDataCacheController インスタンスを生成します。

形式

```
public static FormatDataCacheController createFormatDataCacheController(java.lang.String groupName)
throws DataTransformException
```

引数

- groupName
  グループ名を指定します。

戻り値

FormatDataCacheController インスタンスを返します。

例外

- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。
- プロパティの値が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41004-E）をメッセージログに出力します。

●getDataTransformer

指定されたグループ名に関連付けられている DataTransformer インスタンスを返します。

グループ名に関連付けられているインスタンスが存在しない場合は，新規に生成します。

形式

```
public static DataTransformer getDataTransformer(java.lang.String groupName)
throws DataTransformException
```

引数

- groupName

  グループ名を指定します。

戻り値

DataTransformer インスタンスを返します。

例外

- データ変換 API の処理中に内部エラーが発生した場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41000-E）をメッセージログに出力します。
- メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。
- プロパティの値が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41004-E）をメッセージログに出力します。

●removeDataTransformer

指定されたグループ名に関連付けられている DataTransformer インスタンスを削除します。インスタンスが存在しない場合は，削除処理をしません。

DataTransformer インスタンスを削除する前に DataTransformer インスタンスの停止処理が呼び出されます。

形式

```
public static boolean removeDataTransformer(java.lang.String groupName)
          throws DataTransformException
```

引数

- groupName

  グループ名を指定します。

戻り値

インスタンスが削除された場合は true を返します。

例外

メソッドの引数が不正の場合，例外（DataTransformException）をスローし，エラーメッセージ（KDEC41001-E）をメッセージログに出力します。

(b) FormatDefinition クラス

フォーマット定義およびデータ変換定義を格納するクラスです。

●クラス定義

```
public class FormatDefinition
extends java.lang.Object
```

説明

コンストラクタに次に示す情報を指定してください。

- 定義ファイルの種別を表す FormatDataKind 列挙型
  定義ファイルの種別は，データ変換処理，およびキャッシュのすべての処理で必要な情報です。
- キャッシュ名
  次の処理で必要な情報です。
  ・データ変換
  ・キャッシュの登録
  ・キャッシュの削除
  ・キャッシュの存在確認
- 定義ファイルのバイト配列
  データ変換処理とキャッシュの登録で使用します。

●コンストラクタ

FormatDefinition クラスのコンストラクタを次の表に示します。

表 13-9　FormatDefinition クラスのコンストラクタ

コンストラクタ名	説明
FormatDefinition	(形式 1) FormatDefinition インスタンスを生成し，定義情報の種別およびキャッシュ名を設定します。 (形式 2) FormatDefinition インスタンスを生成し，定義情報の種別，キャッシュ名，および定義ファイルのバイト配列を設定します。

●FormatDefinition（形式 1）

FormatDefinition インスタンスを生成し，定義情報の種別およびキャッシュ名を設定します。

形式

```
public FormatDefinition(FormatDataKind formatDataKind,
                        java.lang.String cacheName)
```

引数

- formatDataKind
  定義ファイルの種別を指定します。

- cacheName

  定義ファイルのキャッシュ名を指定します。定義ファイルの種別が XSD であり，cacheName に null を設定した場合，製品の固定値（CSCDT_XMLFormatCache）が設定されます。

●FormatDefinition（形式 2）

FormatDefinition インスタンスを生成し，定義情報の種別，キャッシュ名，および定義ファイルのバイト配列を設定します。

形式

```
public FormatDefinition(FormatDataKind formatDataKind,
                        java.lang.String cacheName,
                        byte[] definitionBytes)
```

引数

- formatDataKind

  定義ファイルの種別を指定します。

- cacheName

  定義ファイルのキャッシュ名を指定します。定義ファイルの種別が XSD であり，cacheName に null を設定した場合，製品の固定値（CSCDT_XMLFormatCache）が設定されます。

- definitionBytes

  定義ファイルのバイト配列を指定します。

●メソッド

FormatDefinition クラスのメソッドを次の表に示します。

表 13-10　FormatDefinition クラスのメソッド

メソッド名	説明
getCacheName	定義ファイルのキャッシュ名を返します。
getDefinitionBytes	定義ファイルのバイト配列を返します。
getFormatDataKind	定義ファイルの種別を FormatDataKind 列挙型で返します。

●getCacheName

定義ファイルのキャッシュ名を返します。

形式

```
public java.lang.String getCacheName()
```

戻り値

定義ファイルのキャッシュ名を返します。

●getDefinitionBytes

定義ファイルのバイト配列を返します。

形式

```
public byte[] getDefinitionBytes()
```

戻り値

定義ファイルのバイト配列を返します。

●getFormatDataKind

定義ファイルの種別を FormatDataKind 列挙型で返します。

形式

```
public FormatDataKind getFormatDataKind()
```

戻り値

定義ファイルの種別を返します。

(c) InputDataDefinition クラス

変換元データおよび要求電文の FormatDefinition オブジェクトを格納するクラスです。

●クラス定義

```
public class InputDataDefinition
extends java.lang.Object
```

説明

コンストラクタに次に示す情報を指定してください。

- 変換元データ
- 要求電文の FormatDefinition オブジェクト

●コンストラクタ

InputDataDefinition クラスのコンストラクタを次の表に示します。

表 13-11 InputDataDefinition クラスのコンストラクタ

コンストラクタ名	説明
InputDataDefinition	InputDataDefinition インスタンスを生成します。

形式

```
public InputDataDefinition(byte[] data,
                           FormatDefinition inFormatDefinition)
```

引数

- data

  変換元データを指定します。

- inFormatDefinition

  要求電文の FormatDefinition オブジェクトを指定します。

●メソッド

InputDataDefinition クラスのメソッドを次の表に示します。

表 13-12　InputDataDefinition クラスのメソッド

メソッド名	説明
getData	変換元データを返します。
getInFormatDefinition	要求電文の FormatDefinition オブジェクトを返します。

●getData

変換元データを返します。

形式

```
public byte[] getData()
```

戻り値

変換元データ

●getInFormatDefinition

要求電文の FormatDefinition オブジェクトを返します。

形式

```
public FormatDefinition getInFormatDefinition()
```

戻り値

要求電文の FormatDefinition オブジェクトを返します。

(4)　列挙型の詳細

(a)　FormatDataKind

フォーマット定義およびデータ変換定義の種別を表す列挙型です。

●定義

```
public enum FormatDataKind
extends java.lang.Enum<FormatDataKind>
```

●定数

FormatDataKind の列挙型定数を次の表に示します。

表 13-13　FormatDataKind の列挙型定数

定数名	説明
FDX	バイナリフォーマット定義であることを表します。
XSD	XML Schema であることを表します。

定数名	説明
XSLT	XSLT スタイルシート（データ変換定義）であることを表します。

●FDX

バイナリフォーマット定義であることを表します。

形式

```
public static final FormatDataKind FDX
```

●XSD

XML Schema であることを表します。

形式

```
public static final FormatDataKind XSD
```

●XSLT

XSLT スタイルシート（データ変換定義）であることを表します。

形式

```
public static final FormatDataKind XSLT
```

●メソッド

FormatDataKind のメソッドを次の表に示します。

表 13-14　FormatDataKind のメソッド

メソッド名	説明
valueOf	指定した名前を持つこの型の列挙型定数を返します。
values	この列挙型の定数を含む配列を宣言されている順序で返します。

●valueOf

指定した名前を持つこの型の列挙型定数を返します。

形式

```
public static FormatDataKind valueOf(java.lang.String name)
```

説明

文字列は，この型の列挙型定数を宣言するのに使用した識別子と正確に一致している必要があります（余分な空白文字を含めることはできません）。

引数

- name

  返される列挙型定数の名前を指定します。

戻り値

指定した名前の列挙型定数を返します。

例外

- この列挙型に指定した名前の定数がない場合，例外（java.lang.IllegalArgumentException）をスローします。
- 引数が null の場合，例外（java.lang.NullPointerException）をスローします。

●values

この列挙型の定数を含む配列を宣言されている順序で返します。

形式

```
public static FormatDataKind[] values()
```

説明

このメソッドは，次のようにして定数を反復するために使用できます。

```
for(FormatDataKind c: FormatDataKind.values())
  System.out.println(c);
```

戻り値

この列挙型の定数を含む，宣言されている順序での配列を返します。

例外

ありません。

(5) 例外クラスの詳細

(a) CacheNotFoundException クラス

データ変換処理で，フォーマット定義およびデータ変換定義のキャッシュが見つからなかったことを通知する例外クラスです。

●クラス定義

```
public class CacheNotFoundException
extends java.lang.Exception
```

●コンストラクタ

CacheNotFoundException クラスのコンストラクタを次の表に示します。

表 13-15 CacheNotFoundException クラスのコンストラクタ

コンストラクタ名	説明
CacheNotFoundException	（形式 1） 空の CacheNotFoundException インスタンスを生成します。 （形式 2） CacheNotFoundException インスタンスを生成し，詳細メッセージを設定します。

コンストラクタ名	説明
CacheNotFoundException	（形式 3） CacheNotFoundException インスタンスを生成し，詳細メッセージとキャッシュ名を設定します。

●CacheNotFoundException（形式 1）

空の CacheNotFoundException インスタンスを生成します。

形式

```
public CacheNotFoundException()
```

引数

ありません。

●CacheNotFoundException（形式 2）

CacheNotFoundException インスタンスを生成し，詳細メッセージを設定します。

形式

```
public CacheNotFoundException(java.lang.String s)
```

引数

- s

  詳細メッセージを指定します。

●CacheNotFoundException（形式 3）

CacheNotFoundException インスタンスを生成し，詳細メッセージとキャッシュ名を設定します。

形式

```
public CacheNotFoundException(java.lang.String s,
          java.lang.String cacheName)
```

引数

- s

  詳細メッセージを指定します。

- cacheName

  定義ファイルのキャッシュ名を指定します。

●メソッド

CacheNotFoundException クラスのメソッドを次の表に示します。

表 13-16　CacheNotFoundException クラスのメソッド

メソッド名	説明
getCacheName	定義ファイルのキャッシュ名を返します。

●getCacheName

定義ファイルのキャッシュ名を返します。

形式

```
public java.lang.String getCacheName()
```

戻り値

定義ファイルのキャッシュ名を返します。

(b) DataTransformException クラス

データ変換 API の例外を扱うクラスです。

●クラス定義

```
public class DataTransformException
extends java.lang.Exception
```

●コンストラクタ

DataTransformException クラスのコンストラクタを次の表に示します。

表 13-17 DataTransformException クラスのコンストラクタ

コンストラクタ名	説明
DataTransformException	(形式 1) 空の DataTransformException インスタンスを生成します。 (形式 2) DataTransformException インスタンスを生成し，詳細メッセージを設定します。 (形式 3) DataTransformException インスタンスを生成し，例外の原因となるクラスを設定します。 (形式 4) DataTransformException インスタンスを生成し，詳細メッセージと例外の原因となるクラスを設定します。

●DataTransformException（形式 1）

空の DataTransformException インスタンスを生成します。

形式

```
public DataTransformException()
```

引数

ありません。

●DataTransformException（形式 2）

DataTransformException インスタンスを生成し，詳細メッセージを設定します。

形式

```
public DataTransformException(java.lang.String s)
```

引数

- s
  詳細メッセージを指定します。

●DataTransformException（形式 3）

DataTransformException インスタンスを生成し，例外の原因となるクラスを設定します。

形式

```
public DataTransformException(java.lang.Throwable t)
```

引数

- t
  例外の原因となるクラスを指定します。

●DataTransformException（形式 4）

DataTransformException インスタンスを生成し，詳細メッセージと例外の原因となるクラスを設定します。

形式

```
public DataTransformException(java.lang.String s,
                              java.lang.Throwable t)
```

引数

- s
  詳細メッセージを指定します。
- t
  例外の原因となるクラスを指定します。

13.1.4　データ変換 API の使用方法

データ変換 API の使用方法について説明します。

(1)　クラスパスの設定

usrconf.cfg（J2EE サーバ用オプション定義ファイル）に cscdt_api.jar ファイルのクラスパスを指定してください。

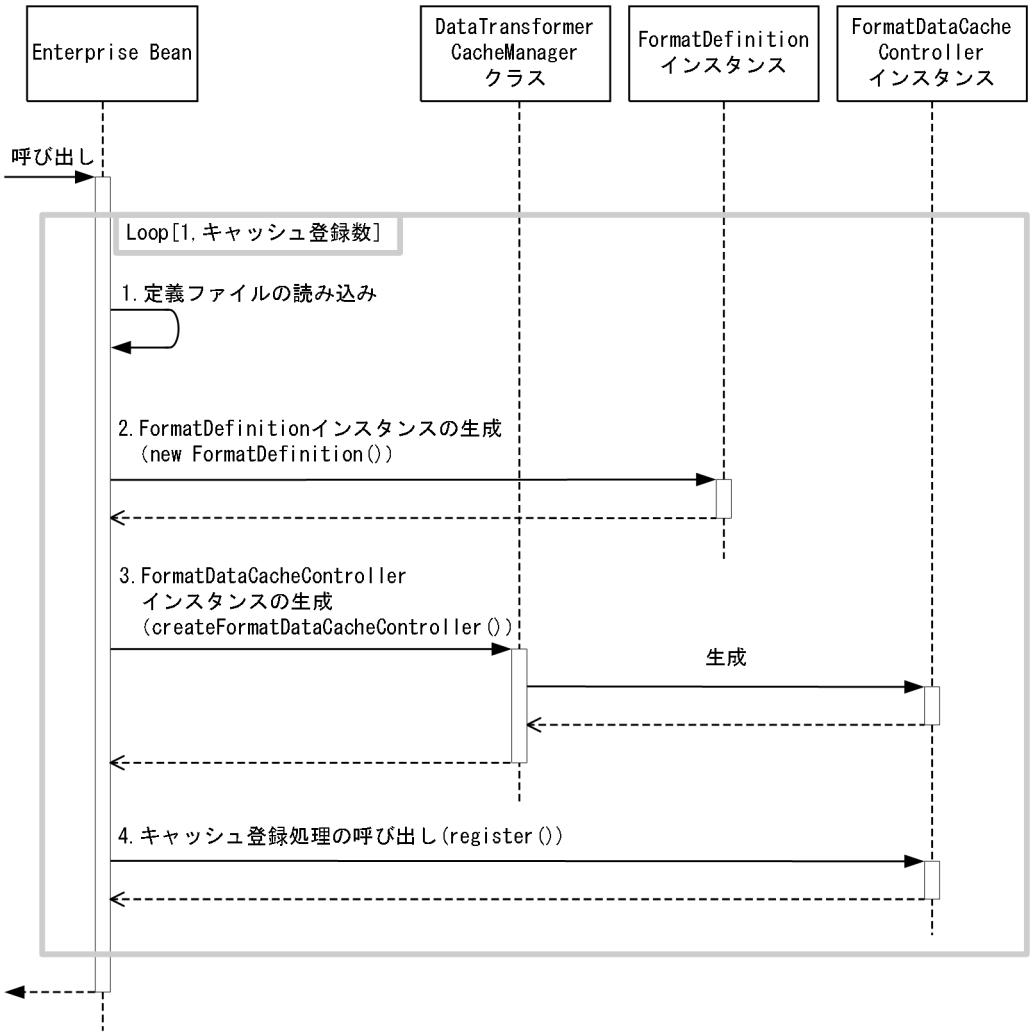
usrconf.cfg の詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.2　usrconf.cfg（J2EE サーバ用オプション定義ファイル)」を参照してください。

(2)　定義ファイルのキャッシュ登録

データ変換 API では定義ファイルのキャッシュの利用を推奨します。キャッシュ登録を実装する手順のシーケンスを次に示します。

全体を通した実装例は「13.1.5(1)　定義ファイルのキャッシュ登録」を参照してください。

図 13-4　キャッシュ登録を実装する手順のシーケンス



キャッシュ登録を実装する手順を次に示します。

1. 定義ファイルの読み込み

キャッシュを登録する定義ファイルをバイト配列で読み込みます。キャッシュできる定義ファイルの種類については，「13.1.2(2)　キャッシュ操作」を参照してください。

2. FormatDefinition インスタンスの生成

FormatDefinition クラスのコンストラクタに次の引数を指定して，FormatDefinition インスタンスを生成してください。

- 第一引数：定義ファイルの種別

- 第二引数：定義ファイルのキャッシュ名
- 第三引数：手順 1 で読み込んだ定義ファイルのバイト配列

FormatDefinition インスタンスの生成の記述例を次に示します。

```
FormatDefinition formatDefinition = new FormatDefinition(
FormatDataKind.FDX, cacheName, definitionBytes);
```

定義ファイルの種別は FormatDataKind 列挙型で指定します。定義ファイルの種別と FormatDataKind 列挙型の対応を次の表に示します。

表 13-18　定義ファイルの種別と FormatDataKind 列挙型の対応表

定義ファイルの種別	FormatDataKind 列挙型の指定値
バイナリフォーマット定義	FDX
データ変換定義（XSLT スタイルシート）	XSLT

なお，FormatDataKind 列挙型の指定値で XSD を指定すると，キャッシュ登録処理でエラーになります。

3. FormatDataCacheController インスタンスの生成

DataTransformerCacheManager クラスの createFormatDataCacheController メソッドを使用し，FormatDataCacheController インスタンスを生成します。createFormatDataCacheController メソッドの引数には，キャッシュの登録先である DataTransformer インスタンスのグループ名を指定してください。

FormatDataCacheController インスタンスの生成の記述例を次に示します。

```
FormatDataCacheController formatDataCacheController =
DataTransformerCacheManager.createFormatDataCacheController(groupName);
```

4. キャッシュ登録処理の呼び出し

手順 3 で生成した FormatDataCacheController インスタンスの register メソッドを使用し，キャッシュを登録します。register メソッドの引数には，手順 2 で生成した FormatDefinition インスタンスを指定してください。

キャッシュ登録処理の呼び出しの記述例を次に示します。

```
formatDataCacheController.register(formatDefinition);
```

5. ほかの定義ファイルのキャッシュ登録処理

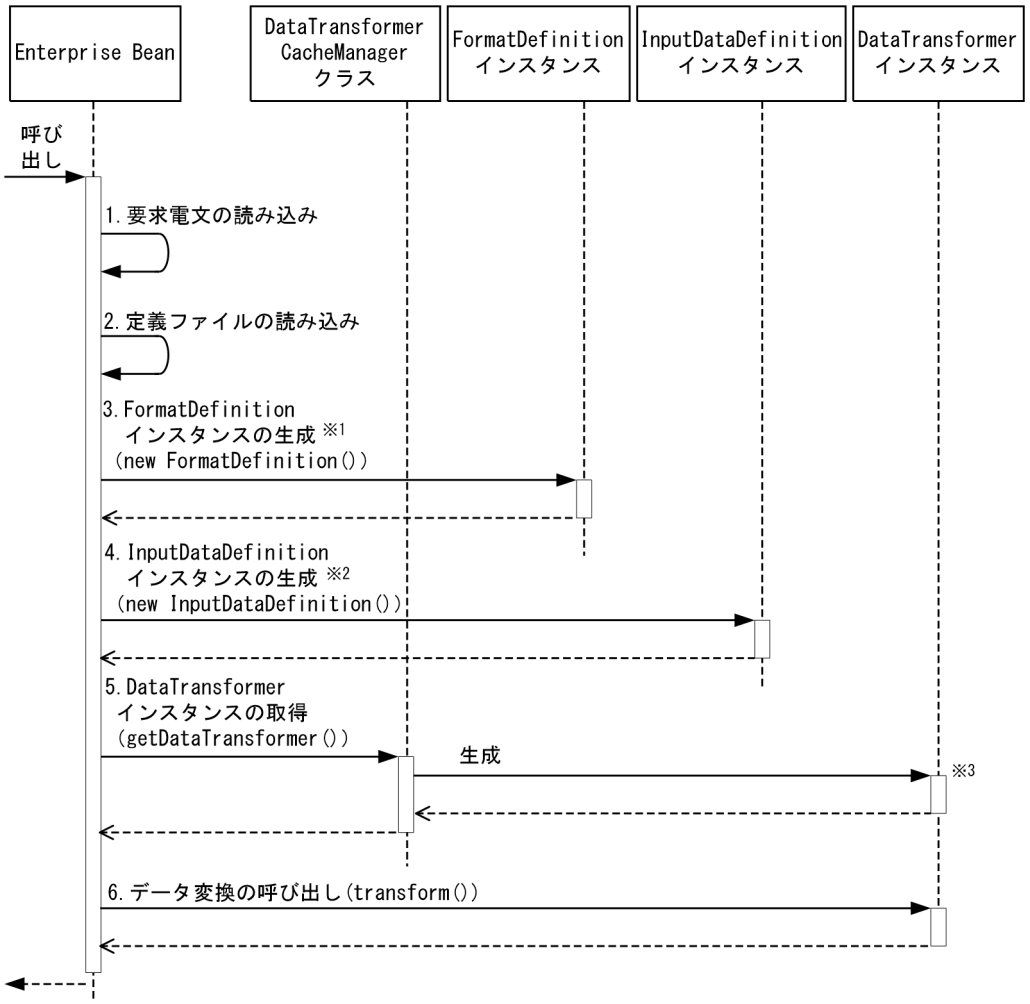
キャッシュに登録する定義ファイルがほかに存在する場合は，手順 1 へ戻り，キャッシュの登録処理を実行してください。

(3)　データ変換の呼び出し

データ変換を実装する手順のシーケンスを次に示します。

全体を通した実装例は「13.1.5(2)　データ変換の呼び出し」を参照してください。

図 13-5　データ変換を実装する手順のシーケンス



注※1

要求電文フォーマット定義，応答電文フォーマット定義，およびデータ変換定義の情報を設定したFormatDefinitionインスタンスがそれぞれ必要です。

注※2

要求電文の数だけInputDataDefinitionインスタンスの生成が必要です。

注※3

DataTransformerCacheManagerクラスが，DataTransformerインスタンスを保持していない場合だけ生成します。

データ変換を実装する手順を次に示します。

1.要求電文の読み込み

データ変換の変換元データである要求電文をバイト配列として読み込みます。

2.定義ファイルの読み込み

データ変換で使用する定義ファイルをバイト配列として読み込みます。必要な定義ファイルを次に示します。

- 要求電文フォーマット定義
- 応答電文フォーマット定義
- データ変換定義

次の場合，定義ファイルの読み込みは不要です。

- 該当する定義ファイルのキャッシュが DataTransformer インスタンスへ登録されている場合
- 要求電文，および応答電文のデータ形式が XML の場合

3. FormatDefinition インスタンスの生成

FormatDefinition クラスのコンストラクタに次の引数を指定して，FormatDefinition インスタンスを生成してください。

- 第一引数：定義ファイルの種別
- 第二引数：定義ファイルのキャッシュ名

FormatDefinition インスタンスは，要求電文フォーマット定義，応答電文フォーマット定義，およびデータ変換定義のそれぞれの情報を格納するインスタンスが必要です。

要求電文，または応答電文のデータ形式が XML の場合に第二引数に null を指定したときは，キャッシュ名に製品の固定値（CSCDT_XMLFormatCache）が設定されます。

FormatDefinition インスタンスの生成の記述例（1 対 1 変換の要求電文のデータ形式が XML の場合）を次に示します。

```
FormatDefinition inFormatDefinition = new FormatDefinition(
FormatDataKind.XSD, null);
```

FormatDefinition インスタンスの生成の記述例（定義ファイルがキャッシュされている場合）を次に示します。

```
FormatDefinition inFormatDefinition = new FormatDefinition(
FormatDataKind.FDX, cacheName);
```

第三引数の登録について次に示します。

- 定義ファイルのキャッシュが DataTransformer インスタンスへ登録されていない場合は第三引数の登録が必要です。

  FormatDefinition クラスのコンストラクタの第三引数に，手順 2 で読み込んだバイト配列を指定してください。定義ファイルのキャッシュが登録されている場合は，FormatDefinition クラスのコンストラクタに定義ファイルのバイト配列を指定しても，データ変換時に無視されます。
- 要求電文，および応答電文のデータ形式が XML の場合は第三引数の登録は不要です。

  第三引数に定義ファイルのバイト配列を指定しても，データ変換時に無視されます。

FormatDefinition インスタンスの生成の記述例（定義ファイルがキャッシュされていない場合）を次に示します。

```
FormatDefinition inFormatDefinition = new FormatDefinition(
FormatDataKind.FDX, cacheName, definitionBytes);
```

4. InputDataDefinition インスタンスの生成

InputDataDefinition クラスのコンストラクタに次の引数を指定して，InputDataDefinition インスタンスを生成してください。

- 第一引数：手順 1 で生成した要求電文
- 第二引数：手順 3 で生成した要求電文の FormatDefinition インスタンス

InputDataDefinition インスタンスの生成の記述例を次に示します。

```
InputDataDefinition[] inDataDefinition = new InputDataDefinition[3];
inDataDefinition[0] = new InputDataDefinition(inputData, inFormatDefinition);
…
```

5. DataTransformer インスタンスの取得

DataTransformerCacheManager クラスの getDataTransformer メソッドを使用し，DataTransformer インスタンスを取得します。getDataTransformer メソッドの引数には，DataTransformer インスタンスを特定するグループ名を指定してください。

DataTransformer インスタンスの取得の記述例を次に示します。

```
DataTransformer dataTransformer =
DataTransformerCacheManager.getDataTransformer(groupName);
```

6. データ変換の呼び出し

手順 5 で生成した DataTransformer インスタンスの transform メソッドを使用し，データ変換を実行します。transform メソッドに次の引数を指定してください。

- 第一引数：手順 4 で作成した InputDataDefinition インスタンス
- 第二引数：応答電文フォーマット定義の FormatDefinition インスタンス
- 第三引数：データ変換定義の FormatDefinition インスタンス

データ変換の呼び出しの記述例を次に示します。

```
byte[] outputData = dataTransformer.transform(
inDataDefinition, outFormatDefinition, transformFormatDefinition);
```

13.1.5　データ変換 API の使用例

キャッシュの登録とデータ変換の使用例を次に示します。

(1)　定義ファイルのキャッシュ登録

Stateless Session Bean を実装し，データ変換 API を使用して定義ファイルのキャッシュの登録処理を呼び出す実装例を次に示します。

●Stateless Session Bean の実装例（Remote インターフェース）

```
package test.implementation.ejb;

import java.rmi.RemoteException;
import java.util.Set;
```

```
import javax.ejb.EJBObject;

import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDataKind;

/**
 * 受付処理(Stateless Session Bean)の実装例(Remoteインターフェース)
 */
public interface TestEJBRemote extends EJBObject {
   public boolean registerCache(String groupName, String cacheName, String filePath) throws
RemoteException;
   public boolean removeCache(String groupName, String cacheName) throws RemoteException;
   public Set<String> getCacheNameList(String groupName, FormatDataKind formatDataKind) thro
ws RemoteException;
}
```

●Stateless Session Bean の実装例（Home インターフェース）

```
package test.implementation.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
* 受付処理(Stateless Session Bean)の実装例(Homeインターフェース)
*/
public interface TestEJBHome extends EJBHome {
   public TestEJBRemote create() throws RemoteException, CreateException;
}
```

●Stateless Session Bean の実装例（Bean クラス）

```
package test.implementation.ejb;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.rmi.RemoteException;
import java.util.Set;

import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import jp.co.Hitachi.soft.csc.dt.api.transformer.DataTransformerCacheManager;
import jp.co.Hitachi.soft.csc.dt.api.transformer.DataTransformException;
import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDataKind;
import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDataCacheController;
import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDefinition;

/**
 * Stateless Session Beanの実装例(Beanクラス)
 */
public class TestEJBBean implements SessionBean {
public TestEJBBean() {
```

```
}
public void ejbActivate() throws EJBException, RemoteException {
}

public void ejbCreate() throws CreateException {
}

public void ejbPassivate() throws EJBException, RemoteException {
}

public void ejbRemove() throws EJBException, RemoteException {
}

public boolean registerCache(
         String groupName, String cacheName, String filePath) {
         boolean ret = false;
         try {
             // オブジェクト取得
             FormatDataCacheController formatDataCacheController =

DataTransformerCacheManager.createFormatDataCacheController(groupName);

             byte[] definitionBytes = Files.readAllBytes(Paths.get(filePath));
             FormatDefinition outFormatDefinition = new FormatDefinition(
                    FormatDataKind.FDX, cacheName, definitionBytes);
             // キャッシュする
             formatDataCacheController.register(outFormatDefinition);

             ret = true;

        } catch (DataTransformException e) {
             // 例外処理を実装する
        } catch (IOException e) {
             // 例外処理を実装する
        } finally {
             // 終了処理を実装する
        }

        return ret;
   }
   public boolean removeCache(String groupName, String cacheName) {

        boolean ret = false;

        try {
            // オブジェクト取得
            FormatDataCacheController formatDataCacheController =

DataTransformerCacheManager.createFormatDataCacheController(groupName);

            // 要求電文フォーマット定義のキャッシュの削除
            FormatDefinition formatDefinition = new FormatDefinition(
                  FormatDataKind.FDX, cacheName);
            formatDataCacheController.remove(formatDefinition);

            ret = true;

        } catch (DataTransformException e) {
```

```
            // 例外処理を実装する
        } finally {
            // 終了処理を実装する
        }

        return ret;
    }

    public Set<String> getCacheNameList(String groupName, FormatDataKind formatDataKind) {

        Set<String> cacheNameSet = null;

        try {
            // オブジェクト取得
            FormatDataCacheController formatDataCacheController =

DataTransformerCacheManager.createFormatDataCacheController(groupName);

            // キャッシュ名の一覧取得
            cacheNameSet = formatDataCacheController.getCacheNameList(formatDataKind);

        } catch (DataTransformException e) {
            // 例外処理を実装する
        } finally {
            // 終了処理を実装する
        }

        return cacheNameSet;

    }

    public void setSessionContext(final SessionContext sc) throws EJBException {
    }
}
```

(2) データ変換の呼び出し

Message-Driven Bean を実装し，データ変換 API を使用してデータ変換処理を呼び出す実装例を次に示します。

●Message-Driven Bean の実装例（N 対 1 変換の場合）

```
package test.implementation.mdb;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;

import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;

import jp.co.Hitachi.soft.csc.dt.api.transformer.CacheNotFoundException;
import jp.co.Hitachi.soft.csc.dt.api.transformer.DataTransformerCacheManager;
```

```
import jp.co.Hitachi.soft.csc.dt.api.transformer.DataTransformException;
import jp.co.Hitachi.soft.csc.dt.api.transformer.InputDataDefinition;
import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDataKind;
import jp.co.Hitachi.soft.csc.dt.api.transformer.FormatDefinition;
import jp.co.Hitachi.soft.csc.dt.api.transformer.DataTransformer;

import com.hitachi.software.ejb.adapter.tp1.TP1InMessage;
import com.hitachi.software.ejb.adapter.tp1.TP1MessageListener;
import com.hitachi.software.ejb.adapter.tp1.TP1OutMessage;

/**
 * Message-Driven Beanの実装例。(3対1のデータ変換)
 */
public class TestMDB2 implements TP1MessageListener, MessageDrivenBean {

    public TestMDB2() {
    }

    public void setMessageDrivenContext(final MessageDrivenContext context) throws EJBExcept
ion {
    }

    public void ejbCreate() {
    }

    public void ejbRemove() throws EJBException {
    }

    public TP1OutMessage onMessage(TP1InMessage inMessage) {

        TP1OutMessage outMessage = null;

        try {
            // 入力データの取得
            byte[] inputData = inMessage.getInputData();

            // 入力データの1～8バイトをグループ名と見なす
            String groupName = new String(Arrays.copyOfRange(inputData, 0, 8)).trim();

            // 入力データの9～16バイト，17～24バイト，25～32バイトを要求電文のフォーマット定
義のキャッシュ名と見なす
            String inFormatCacheName1 = new String(Arrays.copyOfRange(inputData, 8, 16)).tri
m();
            String inFormatCacheName2 = new String(Arrays.copyOfRange(inputData, 16, 24)).tr
im();
            String inFormatCacheName3 = new String(Arrays.copyOfRange(inputData, 24, 32)).tr
im();

            // 入力データの33～40バイトを応答電文のフォーマット定義のキャッシュ名と見なす
            String outFormatCacheName = new String(Arrays.copyOfRange(inputData, 32, 40)).tr
im();

            // 入力データの41～48バイトをデータ変換定義のキャッシュ名と見なす
            String transformCacheName = new String(Arrays.copyOfRange(inputData, 40, 48)).tr
im();

            // 入力データの49～64バイト，65～80バイト，81バイト以降を業務データと見なす
            byte[] businessData1 = Arrays.copyOfRange(inputData, 48, 64);
```

```
            byte[] businessData2 = Arrays.copyOfRange(inputData, 64, 80);
            byte[] businessData3 = Arrays.copyOfRange(inputData, 80, inputData.length);

            // データ変換公開APIのインスタンスの取得
            DataTransformer dataTransformer = DataTransformerCacheManager.getDataTransforme
r(groupName);

            // 要求電文フォーマット定義のオブジェクトの生成
            byte[] inDefinitionBytes1 = Files.readAllBytes(Paths.get("ファイルパスを指定する
"));
            FormatDefinition inFormatDefinition1 = new FormatDefinition(
                    FormatDataKind.FDX, inFormatCacheName1, inDefinitionBytes1);
            byte[] inDefinitionBytes2 = Files.readAllBytes(Paths.get("ファイルパスを指定する
"));
            FormatDefinition inFormatDefinition2 = new FormatDefinition(
                    FormatDataKind.FDX, inFormatCacheName2, inDefinitionBytes2);
            byte[] inDefinitionBytes3 = Files.readAllBytes(Paths.get("ファイルパスを指定する
"));
            FormatDefinition inFormatDefinition3 = new FormatDefinition(
                    FormatDataKind.FDX, inFormatCacheName3, inDefinitionBytes3);

            // 業務データと要求電文フォーマット定義のオブジェクトをInputDataDefinitionに設定
            InputDataDefinition[] inDataDefinition = new InputDataDefinition[3];
            inDataDefinition[0] = new InputDataDefinition(businessData1, inFormatDefinition1
);
            inDataDefinition[1] = new InputDataDefinition(businessData2, inFormatDefinition2
);
            inDataDefinition[2] = new InputDataDefinition(businessData3, inFormatDefinition3
);

            // 応答電文フォーマット定義のオブジェクトの生成
            byte[] outDefinitionBytes = Files.readAllBytes(Paths.get("ファイルパスを指定する
"));
            FormatDefinition outFormatDefinition = new FormatDefinition(
                    FormatDataKind.FDX, outFormatCacheName, outDefinitionBytes);

            // データ変換定義のオブジェクトの生成
            byte[] transformDefinitionBytes = Files.readAllBytes(Paths.get("ファイルパスを指
定する"));
            FormatDefinition transformFormatDefinition = new FormatDefinition(
                    FormatDataKind.XSLT, transformCacheName, transformDefinitionBytes);

            byte[] outputData = null;

            // データ変換処理の呼び出し
            outputData = dataTransformer.transform(inDataDefinition,
                    outFormatDefinition, transformFormatDefinition);

            // 出力データの生成
            outMessage = inMessage.createOutMessage();
            int outoutLength = outputData.length;
            byte[] outBuf = outMessage.getOutputData(outoutLength);
            System.arraycopy(outputData, 0, outBuf, 0, outoutLength);

        } catch (CacheNotFoundException e) {
            // 例外処理を実装する
        } catch (DataTransformException e) {
            // 例外処理を実装する
```

```
        } catch (IOException e) {
            // 例外処理を実装する
        } finally {
            // 終了処理を実装する
        }

        return outMessage;
    }
}
```

13.1.6　データ変換時のボトルネックの特定

データ変換時のボトルネックは，データ変換基盤が出力する性能解析トレースから調査できます。データ変換の前後で出力される 0x9A43 および 0x9A44 のイベント ID の OPT（ASCII）には，付加情報として要求電文，応答電文，およびデータ変換定義のキャッシュ名が出力されるため，これらの情報からデータ変換で使用されたキャッシュを特定できます。

0x9A43 および 0x9A44 の OPT（ASCII）の出力例を次に示します。この例では，要求電文フォーマットのキャッシュ名を In_Cache（N 対 1 変換の場合は In_Cache1，In_Cache2），応答電文のキャッシュ名を Out_Cache，データ変換定義のキャッシュ名を Trans_Cache としています。

- 1 対 1 変換の場合

```
In_Cache -> Out_Cache Trans_Cache
```

- N 対 1 変換の場合

```
In_Cache1, In_Cache2 -> Out_Cache Trans_Cache
```

13.1.7　データ変換 API の注意事項

データ変換 API では，定義ファイルのキャッシュを管理しません。データ変換 API で登録した定義ファイルのキャッシュあふれの検知，およびキャッシュあふれの回避について説明します。

(1)　キャッシュあふれの検知

フォーマット定義，およびデータ変換定義のキャッシュを登録できる数に小さな値を設定した場合，キャッシュの登録時に登録済みのキャッシュがあふれることがあります。定義ファイルのキャッシュがあふれているかどうかを，キャッシュ名の一覧取得機能で検知できます。キャッシュ名の一覧を定期的に取得し，以前に取得した一覧と比較することでキャッシュのあふれを検知できます。

(2)　キャッシュあふれの回避

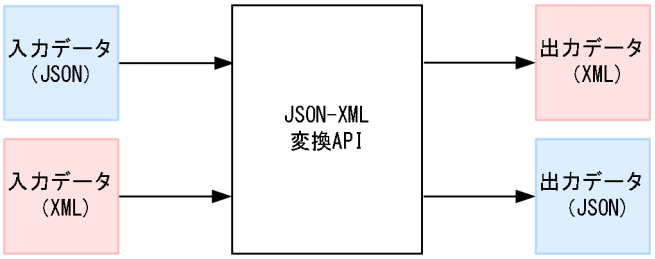
キャッシュのあふれを回避する場合は，データ変換で使用する定義ファイルの最大数よりも大きな値をキャッシュの上限数に設定してください。

13.2　JSON-XML 変換 API を使用したデータの形式変換

JSON-XML 変換 API は，JSON 形式のデータと XML 形式のデータを相互に変換するための API です。JSON-XML 変換 API を使用したライブラリでは，JSON 形式のデータと XML 形式のデータを，一定の基準に従って直接変換できます。

JSON-XML 変換 API を使用した変換例を次の図に示します。

図 13-6　JSON-XML 変換 API を使用した変換例



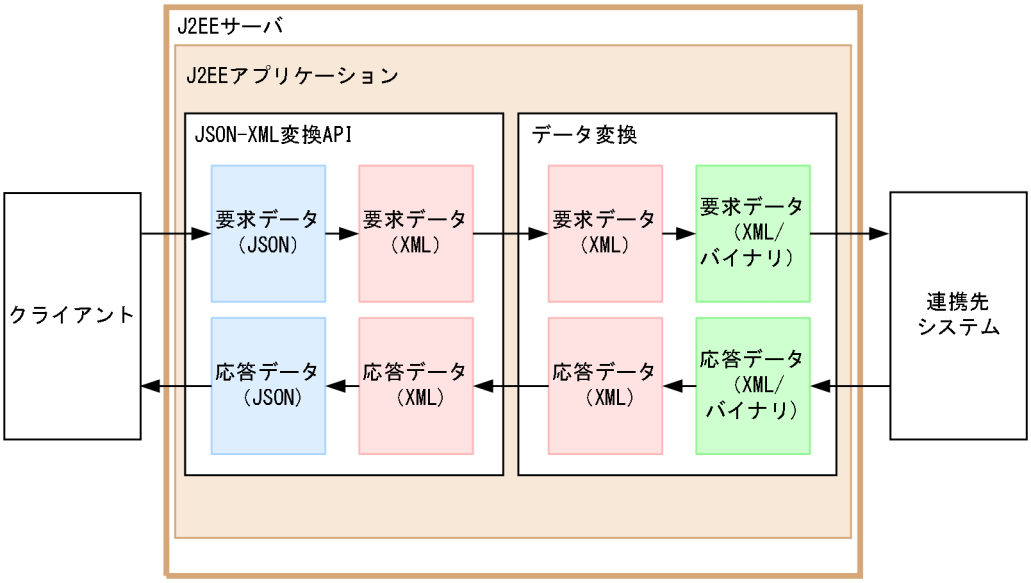
注意事項

J2EE サーバ上で稼働する J2EE アプリケーション，またはビジネスプロセス中の Java 呼出アクティビティから，JSON-XML 変換 API を使用してください。

JSON 形式のデータを XML 形式のデータに変換，または XML 形式のデータを JSON 形式のデータに変換することによって，サービスプラットフォームが提供するデータ変換機能と連携ができます。

データ変換機能と連携する変換例を次の図に示します。

図 13-7　データ変換機能と連携する変換例

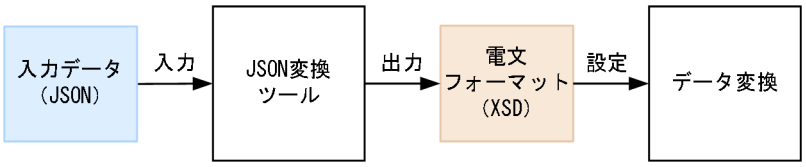


データ変換機能に設定するフォーマット（XML スキーマ）は，JSON 変換ツールで生成します。JSON 変換ツールは，JSON 形式のデータを受け取り，JSON-XML 変換後の XML 形式のデータに適合するフォーマットを生成します。

JSON 変換ツールの詳細については，「14.　JSON 変換ツールを使用したデータ変換」を参照してください。

JSON 形式のデータからフォーマット（XML スキーマ）を作成し，設定するまでの流れを次の図に示します。

図 13–8　JSON 形式のデータからフォーマット（XML スキーマ）を作成，設定するまでの流れ



また，JSON 変換ツールを使用して，JSON-XML 変換の動作を確認できます。JSON-XML 変換をする前に JSON 変換ツールを実行して，JSON-XML 変換処理が正しくできるかどうか，および変換結果が正しいかどうかを確認してください。

JSON-XML 変換で扱う JSON と標準 JSON との差異，および JSON-XML 変換の規則の詳細については，「11.　JSON データの変換」を参照してください。

JSON-XML 変換 API には，XML スキーマを使用しないで変換する標準モードと XML スキーマを使用して変換する拡張モードがあります。この節では，標準モードの JSON-XML 変換 API について説明します。拡張モードの JSON-XML 変換 API については，「12.　拡張モードでの JSON データの変換」を参照してください。

13.2.1　JSON-XML 変換 API 変換定義の設定

JSON-XML 変換 API 変換定義で，JSON-XML 変換 API の変換動作を設定することができます。変換動作を設定する場合は，次に示す JsonXmlConverterProperty クラスを使用します。

JSON-XML 変換 API の詳細については，「13.2.3　JSON-XML 変換 API の詳細」を参照してください。

パッケージ

　jp.co.Hitachi.soft.csc.common.converter.json

クラス

　JsonXmlConverterProperty

メソッド名とデフォルト値を次に示します。JSON 形式から XML 形式に変換する際の変換動作を変更できます。

表 13–19　JSON-XML 変換の変更可能な動作と対応するメソッド名およびデフォルト値

項番	メソッド名	デフォルト値	説明
1	setArrayName	csc-array	csc-array 要素の要素名を変更できます。
2	setElementName	csc-element	csc-element 要素の要素名を変更できます。
3	setSimpleElementName	csc-simple	csc-simple 要素の要素名を変更できます。
4	setObjectName	csc-object	csc-object 要素の要素名を変更できます。
5	setTypeIncludeEnabled	false	JSON 形式データの種別を，XML 形式データの属性に含めるかどうかを設定できます。
6	setDecimalXmloutputControl	true	JSON 形式データから XML 形式データへの変換を行う際に，小数点を含む数値項目の出力形式を制御するかどうかを設定できます。

各メソッドの詳細を次に示します。(1)～(6)と表の項番 1～6 は対応しています。

(1)　csc-array 要素の要素名の変更

NCName 書式の値で任意の名称を設定できます。配列から始まる JSON 形式データを XML 形式データに変換する際に使用される，csc-array 要素の要素名です。

(2)　csc-element 要素の要素名の変更

NCName 書式の値で任意の名称を設定できます。JSON の配列を XML に変換する際に使用される，csc-element 要素の要素名です。変換後の要素名は，「設定値＋アンダーバー（_）＋親要素名」となります。

(3)　csc-simple 要素の要素名の変更

NCName 書式の値で任意の名称を設定できます。JSON の配列を XML に変換する際に使用される，csc-simple 要素の要素名です。

(4)　csc-object 要素の要素名の変更

NCName 書式の値で任意の名称を設定できます。オブジェクトから始まる JSON 形式データを XML 形式データに変換する際に使用される，csc-object 要素の要素名です。

(5)　JSON のデータ種別を XML の属性に含めるかどうかの設定

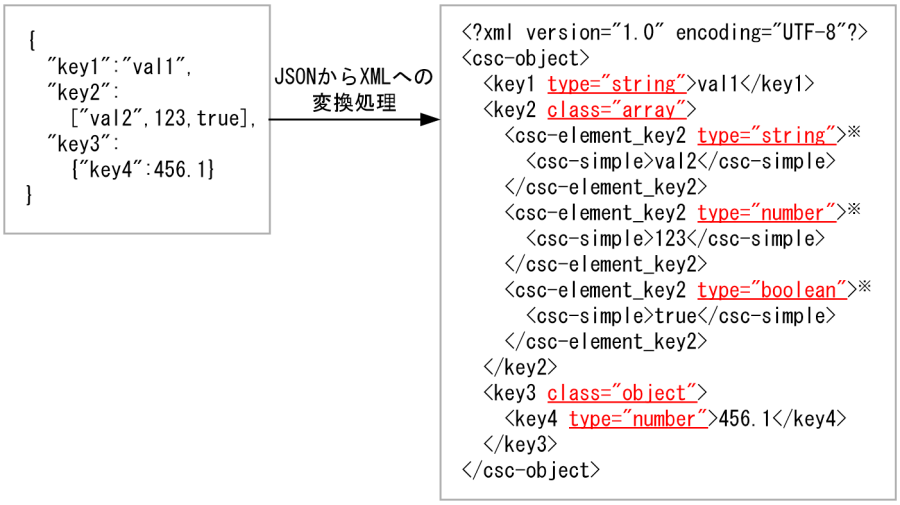
JSON の「文字列型」などのデータ種別を，変換後の XML の属性に含めるかどうかの設定を変更できます。デフォルトでは，JSON のデータ種別を XML の属性に含めません。JSON のデータ種別，および変換後の XML に設定される属性を次の表に示します。

表 13-20　JSON のデータ種別と XML の属性の対応表

JSON のデータ種別	XML の属性
オブジェクト	class="object"
配列	class="array"
文字列	type="string"
数値	type="number"
boolean	type="boolean"

JSON のデータ種別を XML の属性に含める場合の変換例を，次の図に示します。

図 13-9　JSON のデータ種別を XML の属性に含める場合の変換例



注

下線が付いている属性が JSON のデータ種別です。

注※

配列値が文字列型，数値型，boolean 型の場合，属性は csc-element 要素に設定されます。csc-simple 要素には設定されません。

(6)　小数点を含む数値項目の出力形式を制御するかどうかの設定

JSON 形式データから XML 形式データへの変換を行う際に，小数点を含む数値項目の出力形式を制御するかどうかを設定できます。デフォルトでは，小数点を含む数値項目の次のどれかのメソッド仕様に従い数値を出力します。

- java.lang.Float.toString()
- java.lang.Double.toString()
- java.math.BigDecimal.toString()

デフォルトで設定した場合の変換例を，次に示します。

入力データ（JSON）

```
{"key1":10000000.0}
```

変換結果（XML）

```
<?xml version="1.0" encoding="UTF-8"?>
<csc-object><key1>1.0E7</key1></csc-object>
```

13.2.2　JSON-XML 変換 API の障害対策

JSON-XML 変換 API で障害が発生した場合の通知方針および対策方法について説明します。

(1)　障害の通知

JSON-XML 変換 API で障害が発生した場合，メッセージログにエラーメッセージを出力します。また，DataConvertException クラスをスローして JSON-XML 変換 API の呼び出し元へ通知します。このオブジェクトには，障害発生原因となった例外から取得したメッセージが含まれます。

JSON-XML 変換 API で出力するログおよびトレースについて説明します。

- メッセージログ

  JSON-XML 変換 API の実行時に発生した障害のエラー情報をメッセージログへ出力します。

  メッセージログの詳細については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「7.4.1　メッセージログ」を参照してください。

- スタックトレース

  JSON-XML 変換 API はスタックトレースを出力しません。次に示す出力処理はユーザ作成の J2EE アプリケーションに実装してください。

  - スタックトレースの出力処理（例外発生時にスローされる DataConvertException クラスのオブジェクトを参照して実装）
  - 変換に使用したデータの出力処理

- 性能解析トレース

  JSON-XML 変換 API は，性能解析トレースを出力しません。

- 電文トレース

  JSON-XML 変換 API は，電文トレースを出力しません。

  JSON-XML 変換 API で処理するデータの出力が必要な場合は，必要に応じて JSON 形式のデータ，XML 形式のデータを出力する処理を実装してください。

(2) 障害発生時の対策

障害発生時の対策方法を次に示します。メッセージの詳細については，マニュアル「サービスプラットフォーム メッセージ」の「2.1 KDEC00001～KDEC19999 のメッセージ」を参照してください。

(a) JSON-XML 変換 API 変換定義の内容が原因のエラーが発生した場合

JSON-XML 変換 API 変換定義の内容が不正な場合は，JsonXmlConverterFactory クラスの newJsonXmlConverter メソッド発行時にメッセージログへエラーメッセージ（KDEC11201-E）が出力され，JsonXmlConverter クラスのインスタンスの生成が失敗します。

このエラーが発生した場合は，定義内容を修正して処理を再実行してください。

(b) データの変換に失敗した場合

変換時に指定したデータが不正な場合は，JsonXmlConverter クラスの変換メソッド発行時に，メッセージログへエラーメッセージ（KDEC11202-E）が出力され，変換処理が失敗します。

このエラーが発生した場合は，指定したデータを修正して処理を再実行してください。

13.2.3 JSON-XML 変換 API の詳細

サービスプラットフォームが提供する JSON-XML 変換 API は，パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」に所属します。

(1) パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」に所属するクラス

パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」に所属するインターフェースおよびクラスの一覧を次に示します。

●インターフェース

表 13–21 インターフェースの一覧（パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」）

インターフェース名	説明
JsonXmlConverter	JSON-XML 変換を実行する API を提供するクラスのインターフェースです。 このクラスの API を使用することで，JSON-XML 変換を実行できます。
JsonXmlConverterProperty	JSON-XML 変換ライブラリの設定を変更するインターフェースです。

●クラス

表 13–22　クラスの一覧（パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」）

クラス名	説明
JsonXmlConverterFactory	JSON-XML 変換クラスのファクトリクラスです。 JsonXmlConverterFactory.newInstance()の呼び出しインスタンスを生成してください。

●例外クラス

表 13–23　例外クラスの一覧（パッケージ「jp.co.Hitachi.soft.csc.common.converter.json」）

クラス名	説明
DataConvertException	• JSON-XML 変換クラスの例外を扱うクラスです。 • JSON-XML 変換処理中に発生する例外を包括するクラスです。

(2)　インターフェースの詳細

(a)　JsonXmlConverter インターフェース

JSON-XML 変換を実行する API を提供するクラスのインターフェースです。

このクラスの API を使用することで，JSON-XML 変換を実行できます。

JSON-XML 変換クラスのファクトリクラスの JsonXmlConverterFactory.newJsonXmlConverter()の呼び出しインスタンスを生成してください。

JSON-XML 変換クラスは，複数のインスタンスを生成できます。

生成したインスタンスはマルチスレッドで使用できます。

●クラス定義

```
public interface JsonXmlConverter
```

●メソッド

JsonXmlConverter インターフェースのメソッドを次の表に示します。

表 13–24　JsonXmlConverter インターフェースのメソッド

メソッド名	説明
json2Xml	JSON を XML に変換するメソッドです。 引数に指定した JSON 形式のデータ（String）を，XML 形式のデータ（String）に変換します。
xml2Json	XML を JSON に変換するメソッドです。 引数に指定した XML 形式のデータ（String）を，JSON 形式のデータ（String）に変換します。

json2Xml

引数に指定した JSON 形式のデータ（String）を，XML 形式のデータ（String）に変換します。

形式

```
java.lang.String json2Xml (java.lang.String json)
          throws DataConvertException
```

引数

- json

  変換元の JSON 形式のデータを指定します。

  なお，null，空文字，および null 文字列は指定できません。

戻り値

JSON 形式のデータを変換した XML 形式のデータを返します。

例外

DataConvertException：JSON-XML 変換処理でエラーが発生した場合にスローされる例外

次の場合に例外（DataConvertException）をスローし，エラーメッセージ（KDEC11202-E）をメッセージログに出力します。

- JSON データが null の場合
- JSON データが空の場合
- JSON データが null 文字列だけの場合
- 引数に指定した JSON 形式のデータが不正などで JSON-XML 変換処理の実行に失敗した場合

xml2Json

引数に指定した XML 形式のデータ（String）を，JSON 形式のデータ（String）に変換します。

形式

```
java.lang.String xml2Json (java.lang.String xml)
          throws DataConvertException
```

引数

- xml

  変換元の XML データを指定します。

  なお，null，および空文字は指定できません。

戻り値

XML 形式のデータを変換した JSON 形式のデータを返します。

例外

DataConvertException：JSON-XML 変換処理中にエラーが発生した場合にスローされる例外

次の場合に例外をスローし，エラーメッセージ（KDEC11202-E）をメッセージログに出力します。

- XML データが null の場合

- XML データが空の場合
- 引数に指定した XML 形式のデータが不正などで JSON-XML 変換処理の実行に失敗した場合

(b) JsonXmlConverterProperty インターフェース

JSON-XML 変換ライブラリの設定を変更するインターフェースです。

JsonXmlConverterFactory.createConverterProperties()でインスタンスを取得してください。

設定を変更後，JsonXmlConverterFactory.newJsonXmlConverter(JsonXmlConverterProperty)のパラメタに指定することで，変更した設定を反映することができます。

変更しなかった設定は，デフォルト値を使用します。

●クラス定義

```
public interface JsonXmlConverterProperty
```

●注意事項

設定を変更したインスタンスを再利用して，設定の変更や JSON-XML 変換を実行しないでください。
設定を変更したインスタンスを再利用した場合の動作は保証しません。

●メソッド

JsonXmlConverterProperty インターフェースのメソッドを次の表に示します。

表 13-25 JsonXmlConverterProperty インターフェースのメソッド

メソッド名	説明
setArrayName	JSON の配列を XML の繰り返し要素に変換する際の要素名を変更します。
setElementName	JSON の配列を XML の繰り返し要素に変換する際に，XML の繰り返し要素内の子要素名の一部に任意の名称を設定できます。
setObjectName	オブジェクトから始まる JSON 形式データを，XML 形式に変換する際の要素名に任意の名称を設定できます。
setSimpleElementName	XML の繰り返し要素内の単純型要素名を設定します。
setTypeIncludeEnabled	JSON データ型を type 属性に出力するかどうかを設定します。
setDecimalXmloutputControl	小数点を含む数値項目の出力形式を制御するかどうかを設定します。

setArrayName

JSON から XML に変換する際に設定します。

配列から始まる JSON 形式データを，XML 形式に変換する際の要素名に任意の名称を設定できます。

形式

```
void setArrayName (java.lang.String arrayName)
```

引数

- arrayName
  JSON の配列を XML の繰り返し要素に変換する際に変更する要素名を指定します。

注意事項

- NCName として使用できる文字列を指定してください。
- 次の場合は，デフォルトの要素名として csc-array を使用します。
  ・設定を変更しない場合
  ・要素名に null または空文字を設定した場合

setElementName

JSON から XML に変換する際に設定します。

JSON の配列を XML の繰り返し要素に変換する際に，XML の繰り返し要素内の子要素名の一部に任意の名称を設定できます。

変換後の要素名は，「設定値+アンダースコア(_) +親要素名」となります。

形式

```
void setElementName (java.lang.String elementName)
```

引数

- elementName
  XML の繰り返し要素内の子要素名を指定します。

注意事項

- NCName 書式に使用できる文字列を指定してください。
- 次の場合は，設定値に csc-element を使用します。
  ・設定を変更しない場合
  ・要素名に null または空文字を設定した場合

setObjectName

JSON から XML に変換する際に設定します。

オブジェクトから始まる JSON 形式データを，XML 形式に変換する際の要素名に任意の名称を設定できます。

形式

```
void setObjectName (java.lang.String objectName)
```

引数

- objectName
  オブジェクトから始まる JSON 形式データを，XML 形式に変換する際の要素名に任意の名称を設定できます。

注意事項

- NCName として使用できる文字列を指定してください。
- 次の場合は，デフォルトの要素名として csc-object を使用します。
  ・設定を変更しない場合
  ・null または空文字を設定した場合

setSimpleElementName

JSON から XML に変換する，または XML から JSON に変換する際の設定をします。

- JSON から XML に変換する場合
  JSON の配列の中の単純型要素名に任意の名称を設定することができます。
- XML から JSON に変換する場合
  XML 解析時に，配列内の単純型要素名を判断するために使用します。

形式

```
void setSimpleElementName (java.lang.String simpleElementName)
```

引数

- simpleElementName
  XML の繰り返し要素内の単純型要素名を指定します。
  なお，単純型要素名の単純型とは，次のことを指します。
  ・string
  ・number
  ・"true"
  ・"false"

注意事項

- NCName 書式に使用できる文字列を指定してください。
- 設定項目を変更して JSON データから XML データを作成し，その XML データから JSON データに変換する場合は，同じ名称を設定してください。
- 次の場合は，設定値に csc-simple を使用します。
  ・設定を変更しない場合
  ・要素名に null または空文字を設定した場合

setTypeIncludeEnabled

JSON データ型を type 属性に出力するかどうかを設定します。
JSON から XML に変換する際に設定します。
デフォルトは false です。

形式

```
void setTypeIncludeEnabled (boolean isTypeIncludeEnabled)
```

引数

- isTypeIncludeEnabled
  true を設定すると，type 属性に JSON データ型を出力します。
  false を設定すると，type 属性を出力しません。

setDecimalXmloutputControl

小数点付き数値項目の出力形式制御を設定します。

JSON から XML に変換する際に設定します。

デフォルトは true です。

形式

```
void setDecimalXmloutputControl(boolean isDecimalXmloutputControl)
```

引数

- isDecimalXmloutputControl
  true を設定すると，小数点を含む数値項目の次のどれかのメソッド仕様に従い数値を出力します。
  ・java.lang.Float.toString()
  ・java.lang.Double.toString()
  ・java.math.BigDecimal.toString()
  false を設定すると，入力値の文字列表現を保持して出力します。

(3) クラスの詳細

(a) JsonXmlConverterFactory クラス

JSON-XML 変換クラスのファクトリクラスです。

newInstance()の呼び出しインスタンスを生成してください。

変換定義を設定するクラス JsonXmlConverterProperty の取得，および JSON-XML 変換を実行する API を提供するクラス JsonXmlConverter のインスタンスの生成ができます。

●クラス定義

```
public abstract class JsonXmlConverterFactory
extends java.lang.Object
```

●コンストラクタ

JsonXmlConverterFactory クラスのコンストラクタを次に示します。

JsonXmlConverterFactory ()

JsonXmlConverterFactory

形式

```
public JsonXmlConverterFactory ()
```

●メソッド

JsonXmlConverterFactory クラスのメソッドを次の表に示します。

表 13–26　JsonXmlConverterFactory クラスのメソッド

メソッド名	説明
createConverterProperties	JSON-XML 変換クラスの変換定義を管理するクラスのインスタンスを取得します。
newInstance	JSON-XML 変換クラスのファクトリクラスのインスタンスを生成します。
newJsonXmlConverter	JSON-XML 変換クラスのインスタンスを生成します。

createConverterProperties

JSON-XML 変換クラスの変換定義を管理するクラスのインスタンスを取得します。

形式

```
public abstract JsonXmlConverterProperty createConverterProperties ()
```

戻り値

JSON-XML 変換クラスの変換定義を管理するクラスを返します。

例外

ありません。

newInstance

JSON-XML 変換クラスのファクトリクラスのインスタンスを生成します。

形式

```
public static JsonXmlConverterFactory newInstance ()
```

戻り値

JSON-XML 変換クラスのファクトリクラスのインスタンスを返します。

例外

ありません。

newJsonXmlConverter（形式 1）

JSON-XML 変換クラスのインスタンスを生成します。

JSON-XML 変換ライブラリ変換定義はデフォルト値を適用します。

JSON-XML 変換クラスは，複数のインスタンスを生成できます。

生成したインスタンスはマルチスレッドで使用できます。

```
public abstract JsonXmlConverter newJsonXmlConverter ()
          throws DataConvertException
```

戻り値

JSON-XML 変換クラスのインスタンスを返します。

例外

DataConvertException：JSON-XML 変換処理でエラーが発生した場合にスローされる例外

次の場合に例外をスローし，エラーメッセージ（KDEC11201-E）をメッセージログに出力します。

- 内部例外が発生した場合

newJsonXmlConverter（形式 2）

JSON-XML 変換クラスのインスタンスを生成します。

JSON-XML 変換クラスは，複数のインスタンスを生成できます。

生成したインスタンスはマルチスレッドで使用できます。

形式

```
public abstract JsonXmlConverter newJsonXmlConverter (JsonXmlConverterProperty propert
y)
          throws DataConvertException
```

引数

- property

  JSON-XML 変換ライブラリ変換定義

  なお，null を指定した場合はデフォルト値が適用されます。

戻り値

JSON-XML 変換クラスのインスタンスを返します。

例外

DataConvertException：JSON-XML 変換処理でエラーが発生した場合にスローされる例外

次の場合に例外をスローし，エラーメッセージ（KDEC11201-E）をメッセージログに出力します。

- 変換定義に設定した値が不正な場合

(4) 例外クラスの詳細

(a) DataConvertException クラス

JSON-XML 変換クラスの例外を扱うクラスです。

JSON-XML 変換処理中に発生する例外を包括します。

変換定義に設定した値が不正な場合もこの例外がスローされます。

このクラスは，JSON-XML 変換 API で発生した例外の情報，および例外の詳細メッセージを保持します。

●クラス定義

```
public class DataConvertException
extends java.lang.Exception
```

●コンストラクタ

DataConvertException クラスのコンストラクタを次の表に示します。

表 13-27　DataConvertException クラスのコンストラクタ

コンストラクタ名	説明
DataConvertException	(形式 1) DataConvertException インスタンスを生成し，例外の原因となるクラスを設定します。 (形式 2) DataConvertException インスタンスを生成し，詳細メッセージを設定します。 (形式 3) DataConvertException インスタンスを生成し，詳細メッセージと例外の原因となるクラスを設定します。

DataConvertException（形式 1）

DataConvertException インスタンスを生成し，例外の原因となるクラスを設定します。

形式

```
public DataConvertException (java.lang.Exception e)
```

引数

- e

  例外の原因となるクラスを指定します。

DataConvertException（形式 2）

DataConvertException インスタンスを生成し，詳細メッセージを設定します。

形式

```
public DataConvertException (java.lang.String errMsg)
```

引数

- errMsg

  詳細メッセージを指定します。

DataConvertException（形式 3）

DataConvertException インスタンスを生成し，詳細メッセージと例外の原因となるクラスを設定します。

形式

```
public DataConvertException (java.lang.String errMsg,
                             java.lang.Throwable e)
```

引数

- errMsg

詳細メッセージを指定します。

- e
  例外の原因となるクラスを指定します。

13.2.4　JSON-XML 変換 API の使用方法

(1)　準備

(a)　J2EE アプリケーション開発時にコンパイラに指定するクラスパスの設定

JSON-XML 変換 API を使用して J2EE アプリケーションを開発する場合，json_xml_api.jar ファイルをコンパイラのクラスパスに指定してください。

json_xml_api.jar ファイルの格納場所を次に示します。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥json_xml_api.jar
```

注意事項

json_xml_api.jar は開発用の jar のため，usrconf.cfg への設定は不要です。設定した場合の動作は保証しません。

(2)　JSON-XML 変換 API の呼び出し方法

JSON-XML 変換 API の呼び出し方法について説明します。

(a)　変換クラスのインスタンスの取得

変換クラスのインスタンスの取得は，JSON-XML 変換 API 変換定義の設定の有無によって異なります。生成したインスタンスはマルチスレッドで使用できます。

変換メソッドの呼び出しごとにインスタンスを作成する必要はありません。

注意事項

インスタンス取得時に設定した JSON-XML 変換 API 変換定義は変更できません。変更が必要な場合は，JsonXmlConverter クラスのインスタンスを再度取得してください。

●JSON-XML 変換 API 変換定義を設定しない場合

JsonXmlConverterFactory クラスの newJsonXmlConverter メソッドを使用して，JsonXmlConverter クラスのインスタンスを取得します。

インスタンスを取得する例を次に示します。

```
JsonXmlConverterFactory factory = JsonXmlConverterFactory.newInstance();
JsonXmlConverter converter = factory.newJsonXmlConverter();
```

●JSON-XML 変換 API 変換定義を設定する場合

JsonXmlConverterFactory クラスの newInstance メソッドを使用して，JsonXmlConverterFactory のインスタンスを取得します。

インスタンスを取得する例を次に示します。

```
JsonXmlConverterFactory factory = JsonXmlConverterFactory.newInstance();
```

JsonXmlConverterFactory クラスの createConverterProperties メソッドを使用して，JsonXmlConverterProperty クラスのインスタンスを取得します。JsonXmlConverterProperty クラスのメソッドを使用して変換定義を設定してください。配列名を設定する例を次に示します。

```
sonXmlConverterProperty property = factory.createConverterProperties();
property.setArrayName("json_arrayName");
```

JsonXmlConverterFactory クラスの newJsonXmlConverter メソッドの引数に，JSON-XML 変換 API 変換定義クラスのインスタンスを指定します。

インスタンスを指定する例を次に示します。

```
JsonXmlConverter converter = factory.newJsonXmlConverter(property);
```

(b)　JSON→XML 変換処理の呼び出し

JsonXmlConverter クラスの json2xml メソッドを使用して変換します。メソッドの引数には java.lang.String 型の JSON 形式のデータを指定してください。

```
String xml = converter.json2xml(json);
```

(c)　XML→JSON 変換処理の呼び出し

JsonXmlConverter クラスの xml2json メソッドを使用して変換します。メソッドの引数には java.lang.String 型の XML を指定してください。

```
String json = converter.xml2json(xml);
```

13.2.5　JSON-XML 変換 API の使用例

JSON-XML 変換 API の使用例を次に示します。

(1)　JSON→XML 変換

JSON-XML 変換 API を使用して，JSON 形式のデータを XML 形式のデータに変換する場合の実装例を次に示します。

```
import jp.co.Hitachi.soft.csc.common.converter.json.DataConvertException;
import jp.co.Hitachi.soft.csc.common.converter.json.JsonXmlConverterFactory;
import jp.co.Hitachi.soft.csc.common.converter.json.JsonXmlConverterProperty;
import jp.co.Hitachi.soft.csc.common.converter.json.JsonXmlConverter;

/**JSON-XML変換の実装例(JSON→XML)*/
public class TestClass{

 public void testMethod(String json_str)throws DataConvertException{  ※1

  try{
    //JSON-XML変換クラスのファクトリクラスを生成
    JsonXmlConverterFactory factory=JsonXmlConverterFactory.newInstance();
    //変換定義を設定するクラスを取得
    JsonXmlConverterProperty property=factory.createConverterProperties();
    //変換定義を設定（オブジェクト名[data-object]を設定）
    property.setObjectName("data-object");
    //JSON-XML変換を行うクラスを生成
    JsonXmlConverter converter=factory.newJsonXmlConverter(property);

    //JSONデータ(String)→XMLデータ(String)に変換します。
    String xml_str = converter.json2Xml(json_str);  ※2

  }catch(DataConvertException e){
    //例外処理
    //・スタックトレース出力処理（発生した例外をJ2EEサーバに伝播させ例外ログに出力,
        または標準エラーストリームに出力など）
    //・変換に使用したデータの出力処理など
    e.printStackTrace();
    throw e;
  }catch(Exception e){
    //省略
  }catch(Throwable t){
    //省略
  }
 }
}
```

注※1

変換対象のデータ（JSON データ（String））を示します。

```
{"key":"val"}
```

注※2

変換結果のデータ（XML データ（String））を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<data-object>
  <key>val</key>
</data-object>
```

(2) XML→JSON 変換

JSON-XML 変換 API を使用して，XML 形式のデータを JSON 形式のデータに変換を行う場合の実装例を次に示します。

```
import jp.co.Hitachi.soft.csc.common.converter.json.DataConvertException;
import jp.co.Hitachi.soft.csc.common.converter.json.JsonXmlConverterFactory;
import jp.co.Hitachi.soft.csc.common.converter.json.JsonXmlConverter;

/**JSON-XML変換の実装例(XML→JSON)*/
public class TestClass {

 public void testMethod(String xml_str)throws DataConvertException{  ※1

  try{
    //JSON-XML変換クラスのファクトリクラスを生成
    JsonXmlConverterFactory factory=JsonXmlConverterFactory.newInstance();
    //JSON-XML変換を行うクラスを生成
    JsonXmlConverter converter=factory.newJsonXmlConverter();

    //XMLデータ(String)をJSONデータ(String)に変換します。
    String json_str=converter.xml2Json(xml_str);  ※2

  }catch (DataConvertException e){
    //例外処理
    //・スタックトレース出力処理（発生した例外をJ2EEサーバに伝播させ例外ログに出力,
        または標準エラーストリームに出力など）
    //・変換に使用したデータの出力処理など
    e.printStackTrace();
    throw e;
  }catch(Exception e){
    //省略
  }catch(Throwable t){
    //省略
  }
 }
}
```

注※1

変換対象のデータ（XML データ（String））を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<data-object>
  <key>val</key>
</data-object>
```

注※2

変換結果のデータ（JSON データ（String））を示します。

```
{"key":"val"}
```

14 JSON 変換ツールを使用したデータ変換

この章では，JSON 変換ツールを使用したデータ変換について説明します。

14.1　JSON 変換ツールの概要

JSON 変換ツールは，サービスプラットフォームで JSON 形式のデータを利用するために必要な定義の開発およびデータ変換のテストを支援するためのツールです。

プロパティファイルは，cscjson2xsd コマンド，cscjsonxmlconv コマンド，cscjsonxmlconvadv コマンド，および cscjson2xsdadv コマンドで利用できます。プロパティの有効範囲は，コマンド実行単位です。

JSON 変換ツールは，次のコマンドで構成されています。

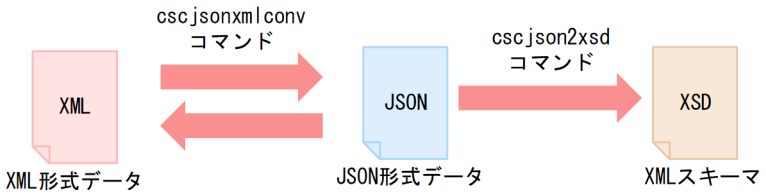
表 14–1　JSON 変換ツールで使用するコマンド

使用するモード	コマンド名	説明
標準モード	cscjson2xsd	JSON 形式データに対応する XML スキーマを生成します。生成した XML スキーマをマッピング定義エディタで使用し，マッピング定義を作成します。
	cscjsonxmlconv	JSON 形式データと XML 形式データとを相互に変換した結果をファイルとして生成します。サービスプラットフォームが自動的に変換する結果を，生成されたファイルと突き合わせて確認できます。
拡張モード	cscjsonxmlconvadv	JSON 形式データおよび XML スキーマを入力し，XML ファイルに変換します。また，その逆に XML ファイルと XML スキーマから JSON 形式データに変換することもできます。
	cscjson2xsdadv	JSON 形式データに対応する XML スキーマを生成します。

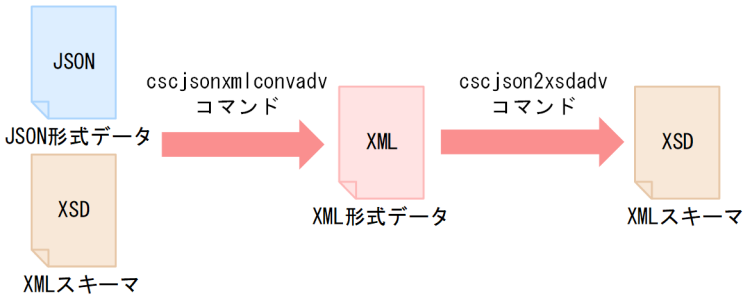
JSON 変換ツールのコマンドと変換対象のデータの関係を次に示します。

図 14-1　JSON 変換ツールのコマンドと変換対象のデータの関係

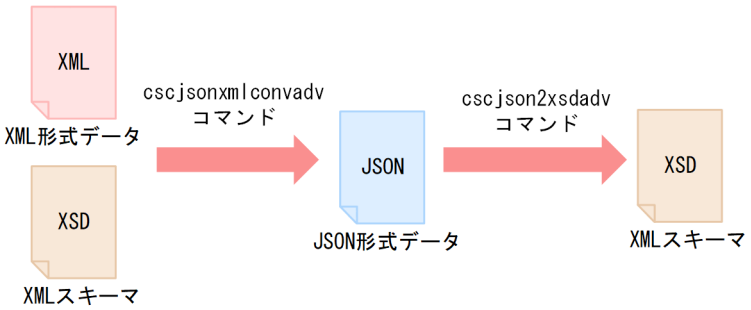
●標準モードの場合



●拡張モード（JSON→XML変換）の場合



●拡張モード（XML→JSON変換）の場合



JSON 変換ツールのコマンドは，次の場所に格納されています。必要に応じて，PATH 環境変数にパスを追加してください。

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥bin
```

JSON 変換ツールの実行には，次の条件が必要です。

- ユーザに管理者権限がある。
- システム環境変数 COSMINEXUS_HOME が正しく設定されている。
- JSON-XML 変換の拡張モードを使用する場合は，OSS のインストールが必要です。必要な OSS については，「11.1.2　標準モードと拡張モードとの差異」を参照してください。

14.1.1　JSON 形式データに対応する XML スキーマの生成

JSON 形式のデータを入力として，XML スキーマを生成するには，cscjson2xsd コマンド，または cscjson2xsdadv コマンドを使用します。生成する XML スキーマは，サービスプラットフォームが自動的に JSON 形式のデータから変換する XML 形式のデータに対応します。

コマンドの入力として，JSON 形式のデータを複数指定できます。複数指定した場合，すべての入力データに対応する XML スキーマが 1 つ生成されます。

cscjson2xsd コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscjson2xsd（JSON 形式データから XSD ファイルへの変換（標準モード））」を，cscjson2xsdadv コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscjson2xsdadv（JSON 形式データから XSD ファイルへの変換（拡張モード））」を参照してください。

生成された XML スキーマは，出現回数や取り得る値の制限など，必要に応じて手動で修正して構いません。修正後のスキーマが正しいかどうかは，cscjsonxmlconv コマンドで生成した XML 形式のデータと突き合わせて確認できます。

参考

cscjson2xsd コマンド，または cscjson2xsdadv コマンドを実行すると，内部処理の延長で Windows の Temp フォルダが利用されます。

14.1.2　JSON-XML 変換結果の確認

JSON 形式のデータと XML 形式のデータを相互に変換して，結果をファイルとして生成するには，cscjsonxmlconv コマンド，または cscjsonxmlconvadv コマンドを使用します。なお，変換結果の JSON 形式データおよび XML 形式データは整形されていません。

なお，cscjsonxmlconvadv コマンドを使用する場合は，変換時に XSD ファイルが必要です。

cscjsonxmlconv コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscjsonxmlconv（JSON 形式データと XML 形式データの相互変換（標準モード））」を，cscjsonxmlconvadv コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscjsonxmlconvadv（JSON 形式データと XML 形式データの相互変換（拡張モード））」を参照してください。

14.2　変換動作の変更オプション

JSON 形式データと XML 形式データとの変換動作を変更するオプションを，プロパティファイルで指定できます。

14.2.1　設定できるプロパティ

プロパティの形式を次に示します。

```
[json-xml-convert.arrays.parent-element-name=<XML形式データのルート要素の名称>]
[json-xml-convert.arrays.child-element-name=<XML形式データの繰り返し要素の名称>]
[json-xml-convert.arrays.child-simple-element-name=<XML形式データの繰り返し要素内に定義される子要素の名称>]
[json-xml-convert.root-element-name=<XML形式データのルート要素の名称>]
[json-xml-convert.json-type.include={true | false}]
[json-xml-convert.decimal-xmloutput-control={true | false}]
```

プロパティの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「3.18　JSON 形式データと XML 形式データの変換で利用するファイル」を参照してください。

14.2.2　プロパティファイルの指定方法

プロパティファイルは，テンプレートファイルをコピーして使います。テンプレートファイルの格納場所を次に示します。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥config¥msg¥templates¥csc_json_converter.properties
```

作成したプロパティファイルは，次に示す 2 つの方法で指定できます。両方の方法で指定された場合は，コマンドで入力したファイルが有効になります。

(1)　コマンドによる指定

コマンドオプションでプロパティファイルを指定します。指定するプロパティファイルのファイル名は任意です。

コマンドオプションの詳細については，「14.1　JSON 変換ツールの概要」を参照してください。

(2)　所定の場所に格納して指定

プロパティファイルを所定のディレクトリに格納することで，自動的にプロパティファイルが読み込まれます。ただし，JSON-XML 変換定義ファイルが存在しない場合は，プロパティが指定されていないとみな

されます。その場合，プロパティのデフォルト値で動作します。ファイル名は，csc_json_converter.properties 固定です。ファイルを格納する所定のディレクトリを次に示します。

```
<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥config¥tools¥json
```

JSON-XML 変換定義ファイルについては，マニュアル「サービスプラットフォーム リファレンス」の「3.18.1　JSON-XML 変換定義ファイル」を参照してください。

14.2.3　プロパティの指定に関する注意事項

実行環境上で JSON 形式データと XML 形式データとを変換する場合も，変換動作を制御する実行環境用のプロパティファイルが指定できます。JSON 変換ツールによる実行結果が同じになるように，開発環境と実行環境で同一のプロパティファイルを使用してください。

プロパティファイルの記述例を次に示します。

```
json-xml-convert.arrays.parent-element-name=csc-array
json-xml-convert.arrays.child-element-name=csc-element
json-xml-convert.arrays.child-simple-element-name=csc-simple
json-xml-convert.root-element-name=csc-object
json-xml-convert.json-type.include=false
json-xml-convert.decimal-xmloutput-control=true
```

14.3　効率良く XML スキーマを生成する方法

cscjson2xsd コマンドを利用して，効率良く XML スキーマを生成できます。

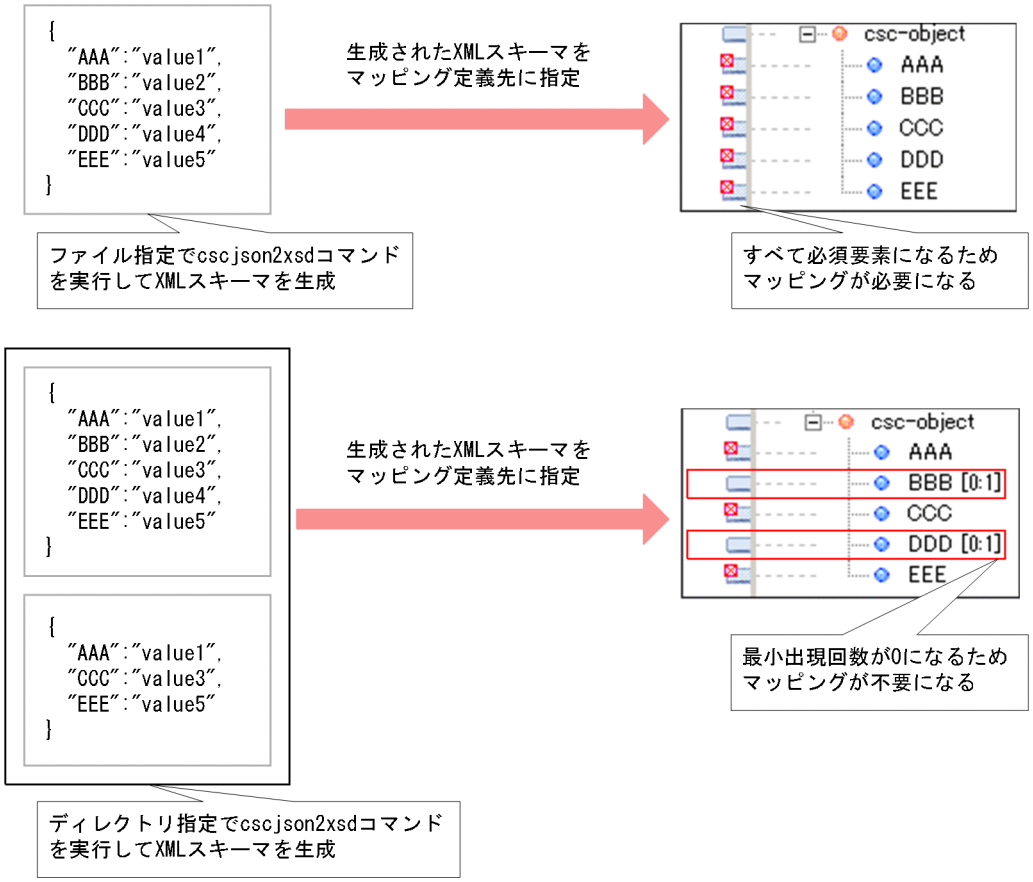
14.3.1　出現回数の制御

JSON 形式データの特定オブジェクトの出現が動的に変更される場合，その JSON 形式データに対応する XML スキーマでは，特定オブジェクトに該当する XML 要素の最小出現回数（minOccurs 属性）を 0 にする必要があります。最小出現回数を 0 にしないと，マッピング定義の変換先スキーマとして使用したときにマッピングが必要になり，オブジェクトがない状態を表現できません。

特定のオブジェクトの有無が動的に変更される場合，次の 2 つの JSON 形式データのファイルを利用して XML スキーマを生成します。これによって，特定のオブジェクトに対応する XML 要素の最小出現回数を 0 にできます。

- 出現が可変となるオブジェクトがすべて出現している JSON 形式データ
- 出現が可変となるオブジェクトがすべて出現していない JSON 形式データ

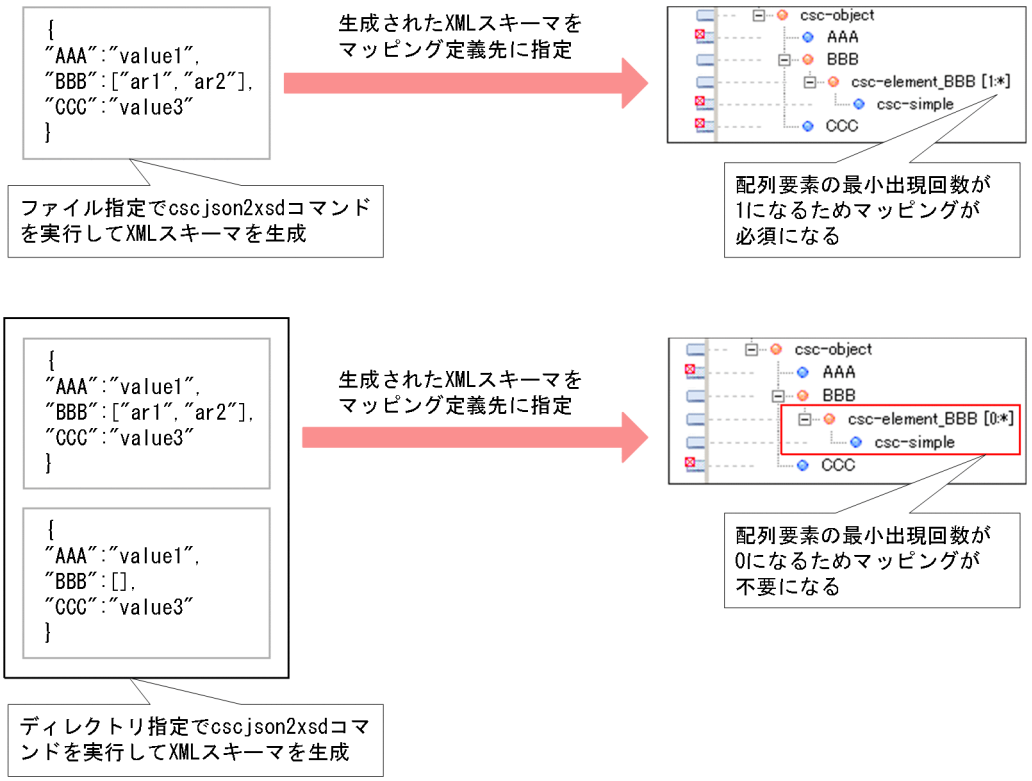
図 14-2　オブジェクトの出現回数を制御した XML スキーマの生成



同様に，配列内の値がまったく存在しない（空の配列）データの可能性がある場合，次に示す JSON 形式データのファイルを利用して XML スキーマを生成します。これによって，配列内の値に対応する XML 要素の最小出現回数を 0 にできます。

- 配列内の値が存在する JSON 形式データ
- 配列内の値がまったく存在しない JSON 形式データ

図 14-3　配列要素の出現回数を制御した XML スキーマの生成



14.3.2　データ型の制御

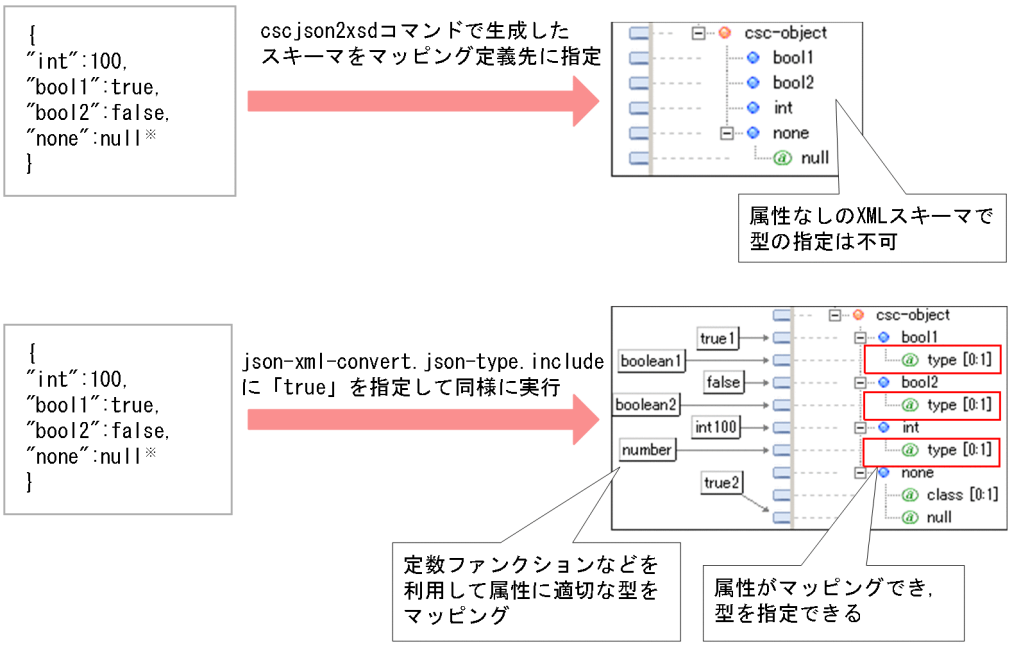
XML 形式のデータを JSON 形式のデータに変換する場合，オブジェクト値および配列値は文字列型となります。明示的に数値型，真偽型など，文字列型以外の型で JSON 形式の値に変換したいときは，XML 形式のデータ要素に属性を指定します。

属性を指定した XML 形式のデータをデータ変換で作成するには，属性が定義された XML スキーマの作成が必要です。属性が定義された XML スキーマは，json-xml-convert.json-type.include プロパティに「true」を指定することで生成できます。

(1)　データの値の型

XML 形式データを JSON 形式データに変換すると，オブジェクト値は文字列型として変換されます。文字列型以外にしたいときは，json-xml-convert.json-type.include に「true」を指定して cscjson2xsd コマンドを実行します。

図 14-4　出力 JSON 形式データの値の型の制御



注※
null属性はデータ値がnullの場合にjson-xml-convert.json-type.includeプロパティの指定に関係なく生成されます。

■データ変換結果のXML形式データをJSON形式データに変換

属性を指定しない場合

```
{
"int":"100",
"bool1":"true",
"bool2":"false",
"none":"null"
}
```

属性を指定した場合

```
{
"int":100,
"bool1":true,
"bool2":false,
"none":null
}
```

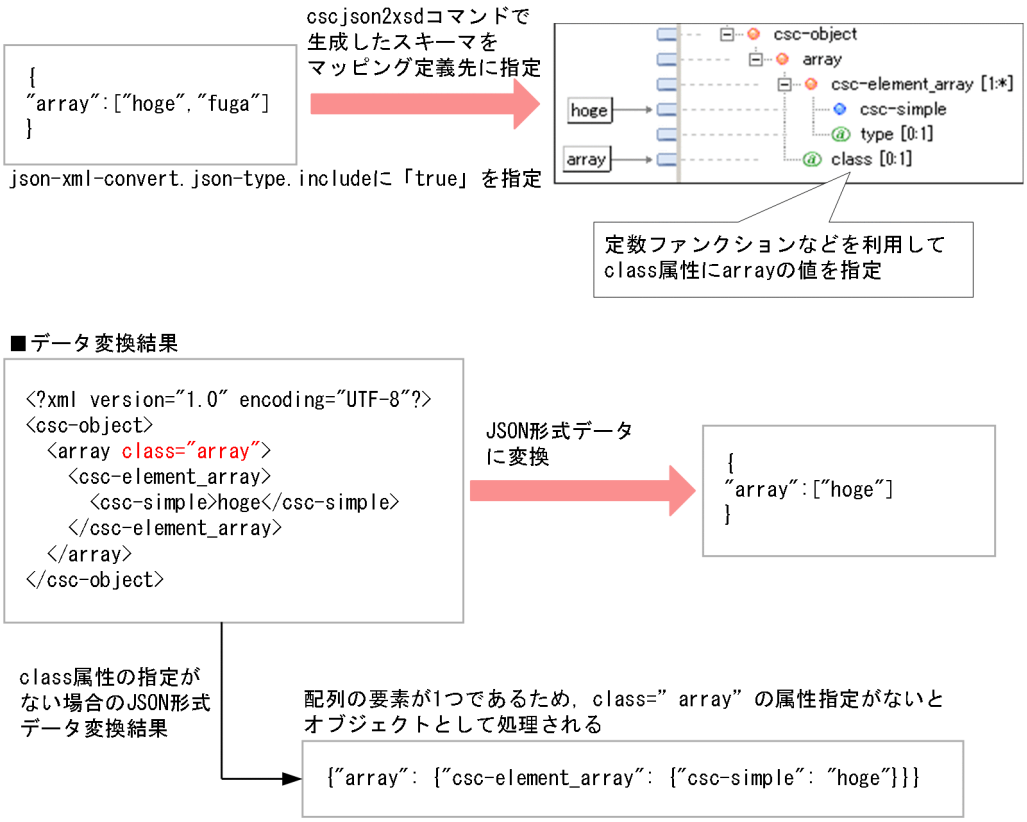
参考

json-xml-convert.json-type.include プロパティで「true」を指定すると，すべての要素に対して属性が付与されます。属性が不要な要素については，XML スキーマを修正して属性を削除して構いません。

(2)　配列の型

配列要素が 1 つである XML 形式データを JSON 形式データに変換すると，オブジェクトとして変換されます。class 属性を指定することで，配列として JSON 形式データに変換できます。

図 14-5　出力 JSON 形式データの配列の型の制御



14.4　JSON 変換ツールの実行例

JSON 形式データに対し，cscjson2xsd コマンド，cscjsonxmlconv コマンドをそれぞれ実行した結果の例を示します。コマンド実行時のプロパティの設定はすべてデフォルトとします。

注意事項

cscjsonxmlconv の実行結果は，見やすくするため整形して記載しています。実際の出力データは整形されていません。

14.4.1　実行例 1

(1)　入力

JSON 形式データ

```
{
  "Image":{
     "Width":800,
     "Height":600,
     "Title":"View from 15th Floor",
     "Thumbnail":{
        "Url":"http://www.example.com/image/481989943",
        "Height":125,
        "Width":100
     },
     "Animated":false,
     "IDs":[116,943,234,38793]
   }
}
```

(2)　コマンドの実行結果

cscjson2xsd コマンドの実行結果

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"elementFormDefault="qualified">
    <xs:element name="csc-object">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Image"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Image">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Animated"/>
```

```
          <xs:element ref="Height"/>
          <xs:element ref="IDs"/>
          <xs:element ref="Thumbnail"/>
          <xs:element ref="Title"/>
          <xs:element ref="Width"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Animated" type="xs:boolean"/>
    <xs:element name="IDs">
      <xs:complexType>
        <xs:sequence>
          <xs:element maxOccurs="unbounded" ref="csc-element_IDs"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="csc-element_IDs">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="csc-simple"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="csc-simple" type="xs:integer"/>
    <xs:element name="Thumbnail">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Height"/>
          <xs:element ref="Url"/>
          <xs:element ref="Width"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Url" type="xs:anyURI"/>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Height" type="xs:integer"/>
    <xs:element name="Width" type="xs:integer"/>
 </xs:schema>
```

cscjsonxmlconv コマンドの実行結果

```
<?xml version="1.0" encoding="UTF-8"?>
<csc-object>
  <Image>
   <Animated>false</Animated>
   <Height>600</Height>
   <IDs>
     <csc-element_IDs>
       <csc-simple>116</csc-simple>
     </csc-element_IDs>
     <csc-element_IDs>
       <csc-simple>943</csc-simple>
     </csc-element_IDs>
     <csc-element_IDs>
       <csc-simple>234</csc-simple>
     </csc-element_IDs>
     <csc-element_IDs>
```

```
      <csc-simple>38793</csc-simple>
    </csc-element_IDs>
  </IDs>
  <Thumbnail>
    <Height>125</Height>
    <Url>http://www.example.com/image/481989943</Url>
    <Width>100</Width>
  </Thumbnail>
  <Title>View from 15th Floor</Title>
  <Width>800</Width>
 </Image>
</csc-object>
```

14.4.2 実行例 2

(1) 入力

JSON 形式データ

```
{
"elem":-123,
"elem1":123.456,
"elem2":1.0123E-5,
"elem3":"ABC",
"elem4":"123ABC",
"elem5":"日立",
"elem6":"ヒタチ",
"elem7":"",
"elem8":" ",
"elem9":"true",
"elemA":false,
"elemB":"2018-04-01",
"elemC":"2018-04-01T09:00:00",
"elemD":"1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12",
"elemE":"44GC44GE44GG44GI44GK44GL44GN44GP44GR44GT44GV44GX44GZ44Gb44Go44Gf44Gh44Gk44Gm44Go44Gq44Gr44Gs44Gt44Gu44Gv44Gy44G144G444G744G+44G/44KA44KB44KC44KE44KG44KI44KJ44KK44KL44KM44KS44KS44KTDQo="
}
```

(2) コマンドの実行結果

cscjson2xsd コマンド実行結果

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"elementFormDefault="qualified">
  <xs:element name="csc-object">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="elem"/>
        <xs:element ref="elem1"/>
```

```
        <xs:element ref="elem2"/>
        <xs:element ref="elem3"/>
        <xs:element ref="elem4"/>
        <xs:element ref="elem5"/>
        <xs:element ref="elem6"/>
        <xs:element ref="elem7"/>
        <xs:element ref="elem8"/>
        <xs:element ref="elem9"/>
        <xs:element ref="elemA"/>
        <xs:element ref="elemB"/>
        <xs:element ref="elemC"/>
        <xs:element ref="elemD"/>
        <xs:element ref="elemE"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="elem" type="xs:integer"/>
  <xs:element name="elem1" type="xs:decimal"/>
  <xs:element name="elem2" type="xs:double"/>
  <xs:element name="elem3" type="xs:NCName"/>
  <xs:element name="elem4" type="xs:NMTOKEN"/>
  <xs:element name="elem5" type="xs:NCName"/>
  <xs:element name="elem6" type="xs:string"/>
  <xs:element name="elem7" type="xs:string"/>
  <xs:element name="elem8" type="xs:string"/>
  <xs:element name="elem9" type="xs:boolean"/>
  <xs:element name="elemA" type="xs:boolean"/>
  <xs:element name="elemB" type="xs:NMTOKEN"/>
  <xs:element name="elemC" type="xs:NMTOKEN"/>
  <xs:element name="elemD" type="xs:hexBinary"/>
  <xs:element name="elemE" type="xs:base64Binary"/>
</xs:schema>
```

cscjsonxmlconv コマンド実行結果

```
 <?xml version="1.0" encoding="UTF-8"?>
 <csc-object>
   <elem>-123</elem>
   <elem1>123.456</elem1>
   <elem2>1.0123E-5</elem2>
   <elem3>ABC</elem3>
   <elem4>123ABC</elem4>
   <elem5>日立</elem5>
   <elem6>ヒタチ</elem6>
   <elem7/>
   <elem8>      </elem8>
   <elem9>true</elem9>
  <elemA>false</elemA>
  <elemB>2018-04-01</elemB>
  <elemC>2018-04-01T09:00:00</elemC>

<elemD>1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345
67890ABCDEF1234567890ABCDEF1234567890ABCDEF12</elemD>
<elemE>44GC44GE44GG44GI44GK44GL44GN44GP44GR44GT44GV44GX44GZ44Gb44Go44Gf44Gh44Gk44Gm44Go44Gq4
4Gr44Gs44Gt44Gu44Gv44Gy44G144G444G744G+44G/44KA44KB44KC44KE44KG44KI44KJ44KK44KL44KM44KS44KS4
```

```
4KTDQo=</elemE>
 </csc-object>
```

14.5 JSON 変換ツールの注意事項

文字コードと BOM についての注意事項を次に示します。

- JSON 変換ツールの入力ファイルとして指定するファイルの文字コードは，UTF-8 である必要があります。ほかの文字コードの場合，結果が正しく出力されない可能性があります。JSON 変換ツールが生成するファイルの文字コードは UTF-8 です。
- JSON 変換ツールの入力ファイルに BOM が付与されている場合，BOM は取り除いて処理されます。JSON 変換ツールが生成するファイルに BOM は付与されません。

マッピング対象外の要素を含む XML スキーマについての注意事項を次に示します。

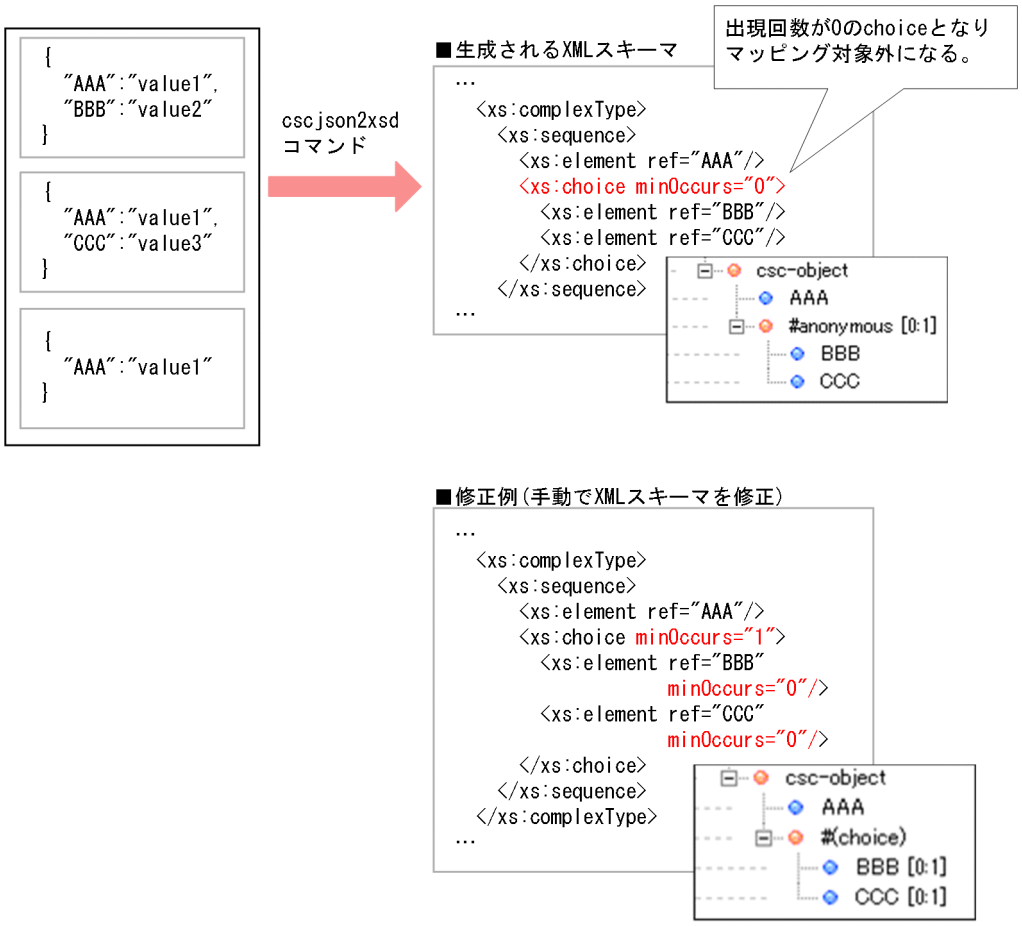
- 入力する JSON 形式のデータによっては，マッピング対象外の要素となる XML スキーマが生成される可能性があります。その場合，XML スキーマを手動で修正してください。
  - choice の最小出現回数が 0 となるケース

    次の図に示すように，JSON 形式データをそれぞれ別の入力ファイルとして準備します。cscjson2xsd コマンドをディレクトリ指定で実行すると，最小出現回数が 0 となる choice 構造の XML スキーマが生成されます。生成された XML スキーマをマッピング定義エディタで確認すると，配下の要素が anonymous となり，BBB 要素，および CCC 要素へのマッピングができません。この場合，次の修正によってマッピングができるようになります。

    ・choice の最小出現回数を 1 とする

    ・choice 配下の BBB 要素と CCC 要素の最小出現回数を 0 とする

図 14-6　choice 構造によりマッピング対象外となる XML スキーマ例



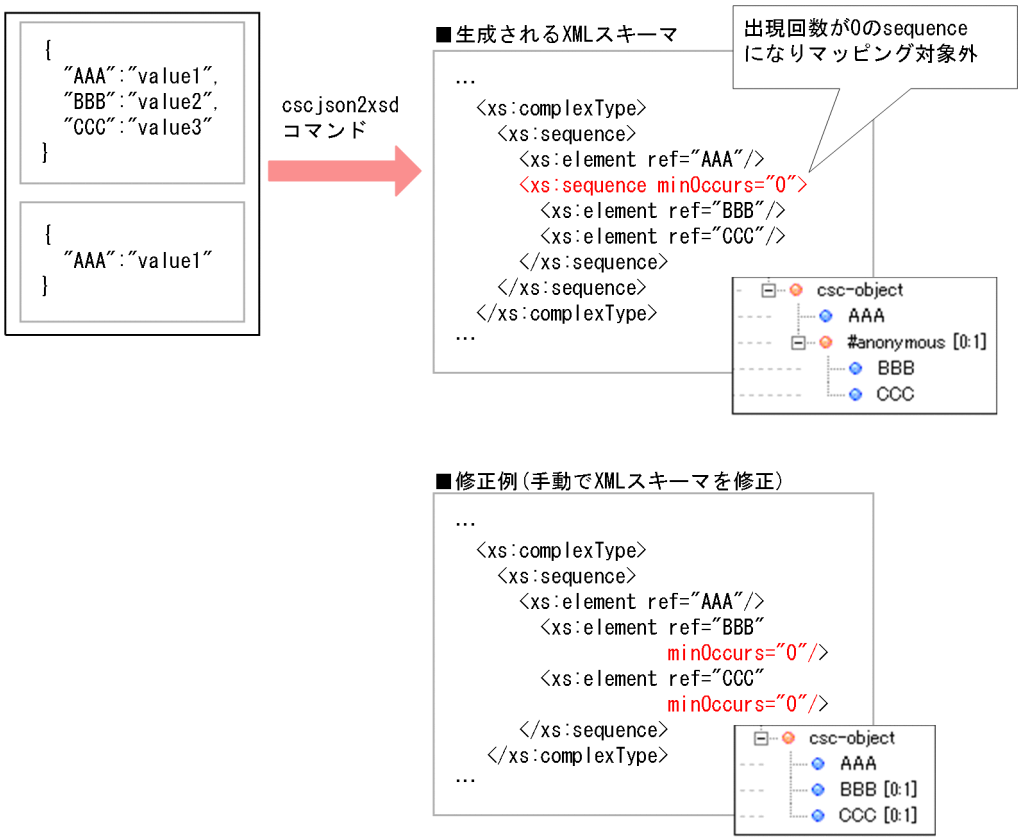
- sequence の最小出現回数が 0 になるケース

  次の図に示すように，JSON 形式データをそれぞれ別の入力ファイルとして準備します。cscjson2xsd コマンドをディレクトリ指定で実行すると，最小出現回数が 0 となる sequence 構造の XML スキーマが生成されます。生成された XML スキーマをマッピング定義エディタで確認すると，配下の要素が anonymous となり，BBB 要素，および CCC 要素へのマッピングができません。この場合，次の修正によりマッピングができるようになります。

  ・BBB 要素と CCC 要素を囲む sequence を削除

  ・BBB 要素と CCC 要素の要素の最小出現回数を 0 とする

図 14-7　sequence 構造によりマッピング対象外となる XML スキーマ例



- JSON 形式データのオブジェクト出現順序について

  XML 形式データを JSON 形式データに変換する場合，JSON 形式データの出現順は，変換前の XML 形式データの出現順になります。このため，データ変換後の XML 形式データから，JSON 形式データの出現順を制御したい場合は，XML スキーマを修正して出現順を変更してください。

図 14-8　生成する JSON 形式データの出現順の制御

■cscjson2xsdで生成したマッピング先のXMLスキーマ

```
…
  <xs:element name="csc-object">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="AAA"/>
        <xs:element ref="BBB"/>
        <xs:element ref="CCC"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="AAA" type="xs:string"/>
  <xs:element name="BBB" type="xs:string"/>
  <xs:element name="CCC" type="xs:string"/>
…
```



データ変換後のXML形式データ

```
<csc-object>
  <AAA>hoge</AAA>
  <BBB>fuga </BBB>
  <CCC>piyo</CCC>
</csc-object>
```

JSON形式データに変換

```
{
  "AAA": "value1",
  "BBB": "value2",
  "CCC": "value3"
}
```

■順番を変更したマッピング先のXMLスキーマ

```
…
  <xs:element name="csc-object">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CCC"/>
        <xs:element ref="AAA"/>
        <xs:element ref="BBB"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="AAA" type="xs:string"/>
  <xs:element name="BBB" type="xs:string"/>
  <xs:element name="CCC" type="xs:string"/>
…
```



データ変換後のXML形式データ

```
<csc-object>
  <CCC>value3</CCC>
  <AAA>value1</AAA>
  <BBB>value2</BBB>
</csc-object>
```

JSON形式データに変換

```
{
  "CCC": "value3",
  "AAA": "value1",
  "BBB": "value2"
}
```

オブジェクトの出現順を制御できる

付録

付録 A　旧バージョンからの移行

旧バージョンの開発環境で使用していたリポジトリ情報を，バージョンアップ後も使用する場合のバージョンアップ方法について説明します。

なお，バージョンアップ前に使用していたリポジトリ情報をそのまま使用して，バージョンアップ後も運用する場合のバージョンアップ方法については，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「付録 G　旧バージョンからの移行」を参照してください。

注意

- 08-10 以前の開発環境で使用していたリポジトリ情報は，SOAP 1.1 モードの場合だけ移行できます。08-50 以降は SOAP 1.1 モードおよび SOAP 1.1/1.2 併用モードの場合に移行できます。
- 組み込みデータベースについて

  バージョン 11 以降では，標準で提供していた開発環境で使用できる組み込みデータベースを HiRDB Embedded Server から HiRDB/Single Server へ変更しています。組み込みデータベースの用途は開発時の利用に限定されているため，本番環境で利用することはできません。

  バージョンアップする場合は，旧バージョンで構築した環境を削除してからバージョンアップしてください。詳細については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「付録 F　旧バージョンからの移行（WTP からの移行の場合)」を参照してください。

付録 A.1　移行の対象となるバージョン

リポジトリ情報の移行の対象となる旧バージョンを次に示します。これらのバージョンのリポジトリ情報を移行して，開発環境を 11-70 にバージョンアップできます。

- Service Architect　07-10
- Service Architect　07-20
- Service Architect　07-50
- Service Architect　07-60
- Service Architect　08-00
- Service Architect　08-10
- Service Architect　08-50
- Service Architect　08-51
- Service Architect　08-53
- Service Architect　08-70
- Service Architect　09-00
- Service Architect　09-50

- Service Architect　09-51
- Service Architect　09-60
- Service Architect　09-70
- Service Architect　09-71
- Service Architect　09-87
- Service Architect　11-00
- Service Architect　11-10
- Service Architect　11-20
- Service Architect　11-30
- Service Architect　11-40
- Service Architect　11-50
- Service Architect　11-60

付録 A.2　旧バージョンからの移行手順

次の図に示す(1)～(6)の手順に従って，サービスプラットフォーム（開発環境）を旧バージョンから 11-70 へ移行してください。

図 A-1　旧バージョンからの移行手順

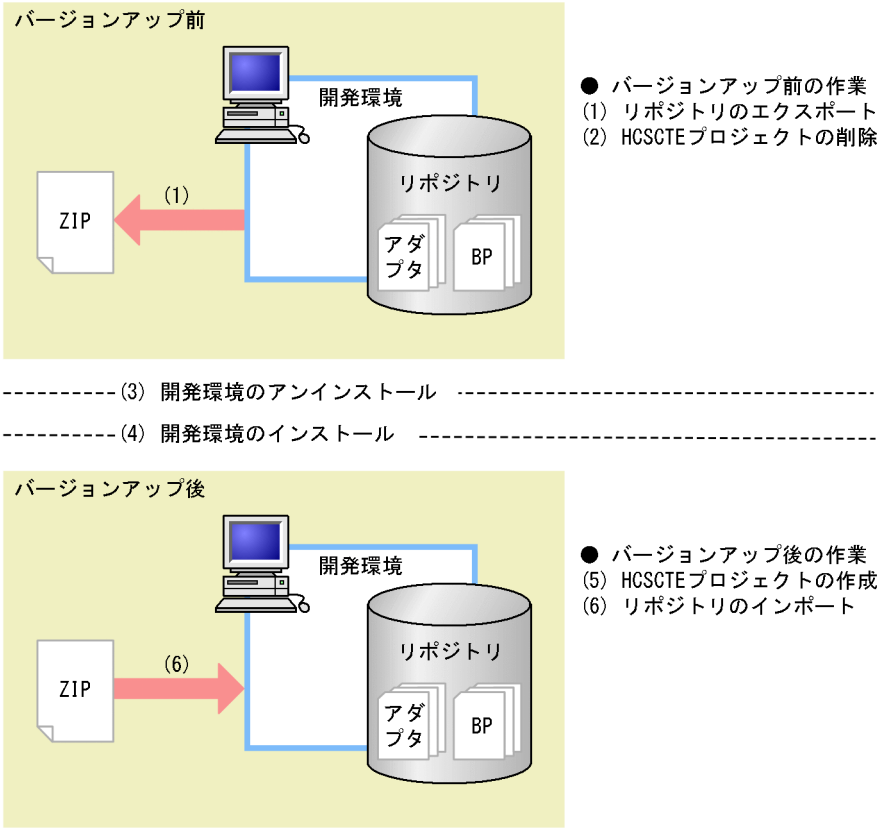


図 A-1 の(1)～(6)の手順の詳細を次に示します。

(1)　リポジトリのエクスポート

バージョンアップ前の開発環境のリポジトリをエクスポートして，いったん退避します。複数のリポジトリを使用している場合，必要なリポジトリをすべてエクスポートして，退避します。

リポジトリのエクスポート方法の詳細については，「3.2.2　リポジトリをエクスポートする」を参照してください。

(2)　HCSCTE プロジェクトの削除

HCSCTE プロジェクトをいったん削除します。

手順の詳細は，「3.1.4　プロジェクトを削除する」を参照してください。

(3)　開発環境のアンインストール

アンインストールの前に，Eclipse のアンセットアップを実施してください。

手順については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「2.8.2　Eclipse 環境のアンセットアップ」を参照してください。

手動で configuration フォルダも削除する必要があります。

Eclipse のアンセットアップ後に，旧バージョンのサービスプラットフォームをアンインストールします。サービスプラットフォームのインストールディレクトリ以下のディレクトリ情報を退避したあとに，旧バージョンのサービスプラットフォームをアンインストールすることをお勧めします。

注意

- J2EE サーバ，Management Server，PRF など実行環境の構成要素を停止してからアンインストールしてください。
- 組み込みデータベースを使用している場合，組み込みデータベースを停止してからアンインストールしてください。

(4)　開発環境のインストール

開発環境で 11-70 のサービスプラットフォームをインストールしてバージョンアップします。

また，バージョンに対応した Eclipse をセットアップします。

注意

09-71 より前のバージョンから移行する場合の注意

- 09-71 から SOAP モードのデフォルトが SOAP1.1/1.2 併用モードに変更されたため，インストール直後の SOAP モードは SOAP1.1/1.2 併用モードです。旧バージョンで指定した SOAP

モードと同じ SOAP モードを設定してください。SOAP モードの設定手順については「2.2.4 SOAP モードの選択」を参照してください。

- 次に示すディレクトリ配下の情報を必要に応じて退避してからインストールしてください。
  <サービスプラットフォームのインストールディレクトリ>¥CSCTE¥Samples

(5) HCSCTE プロジェクトの作成

バージョンアップした開発環境で，新たに HCSCTE プロジェクトを作成します。また，リポジトリとして利用するディレクトリを指定します。なお，リポジトリとして利用するディレクトリには，運用環境のリポジトリを指定しないでください。

手順の詳細は，「3.1.1　プロジェクトを作成する」を参照してください。

旧バージョンで外部バインディングファイルを利用していた場合は，Eclipse でファイルを指定し直す必要があります。手順の詳細は，「付録 K　外部バインディングファイルを利用した WSDL のカスタマイズ」を参照してください。

(6) リポジトリのインポート

「(1)　リポジトリのエクスポート」でエクスポートしたリポジトリ情報を開発環境にインポートします。

旧バージョンのリポジトリをインポートすると，パッケージングおよび配備定義が自動で実施され，リポジトリ情報が今バージョンに引き継がれます。なお，パッケージングおよび配備定義が自動で実施されるのは，旧バージョンでパッケージングおよび配備定義したサービスアダプタ，ビジネスプロセス，およびユーザ定義受付だけです。旧バージョンの DB アダプタは使用しないで，新規に追加してください。DB アダプタを新規に追加する方法については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.2.5　DB アダプタを新規に追加する」を参照してください。

リポジトリのインポート方法については，「3.2.3　リポジトリをインポートする」を参照してください。

注意

07-50 より前のバージョンから移行する場合の注意

- SessionBean アダプタのユーザ定義クラスに次のファイル名が設定されているときは，警告メッセージが表示されます。

  ・csmsvcadpdef.jar

  ・cscmsg_adpejb.jar

  警告メッセージが表示された場合，サービスアダプタ定義画面で，上記のユーザ定義クラスを削除し，ファイル名を変更してから，再度設定してください。
- 07-50 より前のバージョンで定義したビジネスプロセスの変数で，デフォルト名前空間が指定されていない電文フォーマットのスキーマファイルが設定されている場合，バージョンアップ後に電文フォーマットファイルを登録し直してください。

07-60 以降のバージョンから移行する場合の注意

07-60 以降のバージョンから移行する場合，実行環境に配備するサービスアダプタおよびユーザ定義受付は，開発環境にインポートして再度パッケージングしてください。サービスアダプタを再度パッケージングしないと，メッセージ KDEC03007-E が出力されないことがあります。また，ユーザ定義受付を再度パッケージングしないと，メッセージ KDEC00001-E で不正な埋字が出力されることがあります。パッケージングの方法については，「8.2　パッケージング」を参照してください。

08-10 より前のバージョンから移行する場合の注意

データ変換定義で，sequence 要素または choice 要素下に，出現回数が 1 回に固定されている次の要素（コンポジタ）を定義している場合，ノードの表示が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

- sequence
- choice

エラーメッセージが表示された場合は，再度マッピング定義を起動し，変更を反映させてください。

09-50 より前のバージョンから移行する場合の注意

データ変換定義で，次に該当するスキーマを定義している場合，名前空間が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

1. import 先で対象名前空間（targetNamespace）を定義しているスキーマを，変換元ノードに指定している。
2. 1.の名前空間が，import または include 先を含むすべての名前空間宣言（xmlns[:prefix]）で定義されていない。

エラーメッセージが表示された場合は，マッピング定義を再度起動し，変更を反映させてください。

09-70 より前のバージョンから移行する場合の注意

データ変換定義で，次に該当するスキーマを定義している場合，名前空間が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

1. 1 つの名前空間 URI に対して，複数の異なる名前空間接頭辞を指定している。

エラーメッセージが表示された場合は，マッピング定義を再度起動し，変更を反映させてください。

メモリについての注意

旧バージョンのリポジトリをインポートするときは，メモリを多く必要とします。そのため，次の手順でメモリサイズが十分かどうかを確認し，メモリが不足している場合は，メモリサイズを増やしてからリポジトリをインポートしてください。

1. Eclipse のメニューから，［ウィンドウ］－［設定］を選択します。
2. ダイアログ左側のツリービューから，［一般］を選択します。
3. ダイアログ右側の［ヒープ・ステータスを表示］チェックボックスにチェックします。
4. Eclipse のウィンドウの右下のゴミ箱アイコンを選択し，ガーベッジ・コレクターを実行します。

5. ゴミ箱アイコンの左に表示されているヒープ・サイズを確認します。
6. 未使用のヒープ・サイズが不足している場合，eclipse.ini を編集し，メモリサイズ（-Xmx の値）を大きくします。eclipse.ini の編集については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「付録 B.1　インストール」を参照してください。

開発環境にインポートしたリポジトリ情報を使用して実際に運用する場合，このリポジトリ情報を開発環境から運用環境にエクスポートし，運用環境から HCSC コンポーネントを配備します。

付録 A.3　旧バージョンで開発環境と運用環境のリポジトリを共有していた場合の移行手順

バージョン 07-60 以降では，開発環境と運用環境で同じリポジトリを共有できません。開発環境と運用環境で同じリポジトリを共有していた場合，次の図に示す(1)～(6)の手順に従って，サービスプラットフォームを旧バージョンから 11-70 へ移行してください。

図 A-2　旧バージョンからの移行手順（開発環境と運用環境のリポジトリを共有していた場合）

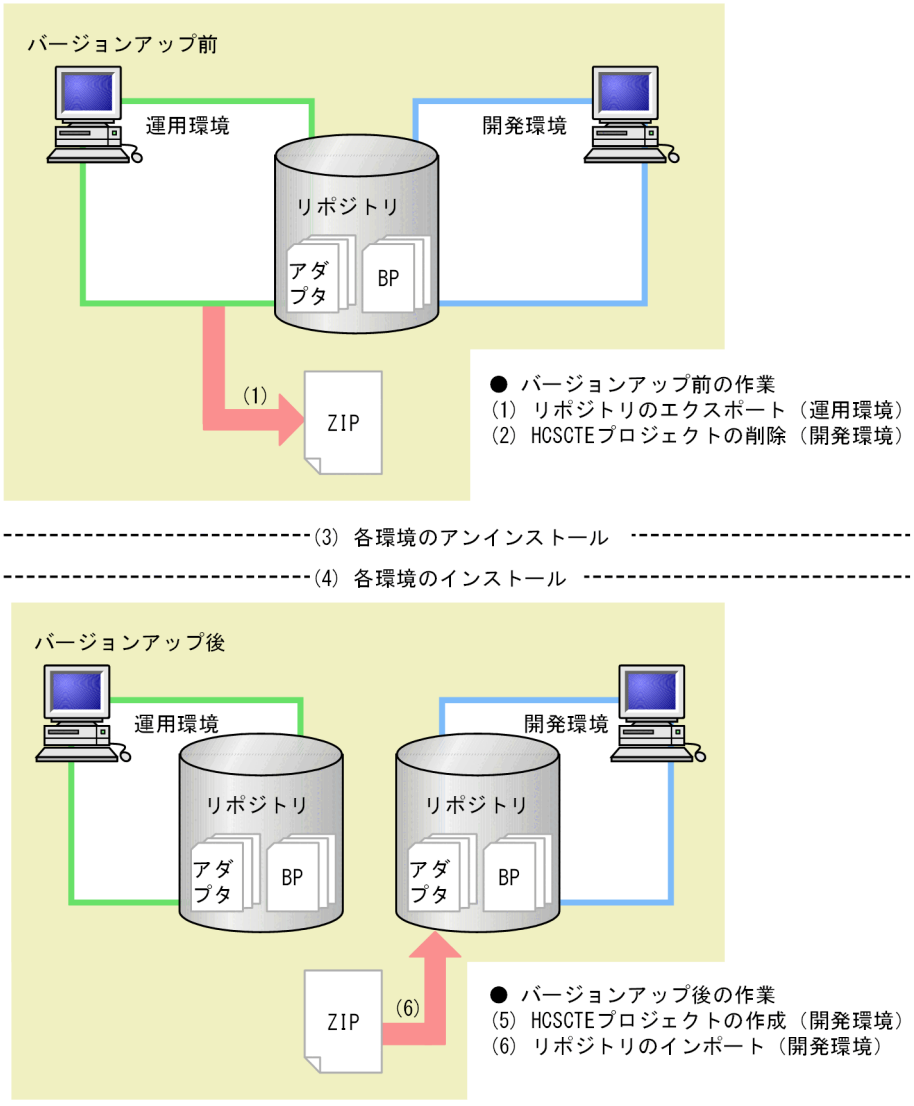


図 A-2 の(1)～(6)の手順の詳細を次に示します。

(1)　リポジトリのエクスポート（運用環境）

cscrepctl コマンド（-export オプション）を運用環境で実行して，バージョンアップ前のリポジトリをエクスポートします。複数のリポジトリを使用している場合，必要なリポジトリをすべてエクスポートして，退避します。

(2)　HCSCTE プロジェクトの削除（開発環境）

HCSCTE プロジェクトをいったん削除します。

手順の詳細は，「3.1.4　プロジェクトを削除する」を参照してください。

(3)　各環境のアンインストール

開発環境，運用環境で旧バージョンのサービスプラットフォームをアンインストールします。サービスプラットフォームのインストールディレクトリ以下のディレクトリ情報を退避したあとに，旧バージョンのサービスプラットフォームをアンインストールすることをお勧めします。

注意

- 開発環境は，Eclipse のアンセットアップをしてからアンインストールしてください。configuration フォルダも削除してください。
- J2EE サーバ，Management Server，PRF など実行環境の構成要素を停止してからアンインストールしてください。
- 組み込みデータベースを使用している場合，組み込みデータベースを停止してからアンインストールしてください。
- HCSC サーバをアンセットアップしたあとに，リポジトリルート配下を削除してください。

(4)　各環境のインストール

開発環境，運用環境で 11-70 のサービスプラットフォームをインストールしてバージョンアップします。

また，開発環境では，バージョンに対応した Eclipse をセットアップします。

(5)　HCSCTE プロジェクトの作成（開発環境）

バージョンアップした開発環境で，新たに HCSCTE プロジェクトを作成します。また，リポジトリとして利用するディレクトリを指定します。なお，リポジトリとして利用するディレクトリには，運用環境のリポジトリを指定しないでください。

手順の詳細は，「3.1.1　プロジェクトを作成する」を参照してください。

(6) リポジトリのインポート（開発環境）

「(1) リポジトリのエクスポート（運用環境）」でエクスポートしたリポジトリ情報を開発環境にインポートします。

旧バージョンのリポジトリをインポートすると，パッケージングおよび配備定義が自動で実施され，リポジトリ情報が今バージョンに引き継がれます。なお，パッケージングおよび配備定義が自動で実施されるのは，旧バージョンでパッケージングおよび配備定義した HCSC コンポーネントだけです。

リポジトリのインポート方法については，「3.2.3 リポジトリをインポートする」を参照してください。

注意

07-50 より前のバージョンから移行する場合の注意

SessionBean アダプタのユーザ定義クラスに次のファイル名が設定されているときは，警告メッセージが表示されます。

- csmsvcadpdef.jar
- cscmsg_adpejb.jar

警告メッセージが表示された場合，サービスアダプタ定義画面で，上記のユーザ定義クラスを削除し，ファイル名を変更してから，再度設定してください。

08-10 より前のバージョンから移行する場合の注意

データ変換定義で，sequence 要素または choice 要素下に，出現回数が 1 回に固定されている次の要素（コンポジタ）を定義している場合，ノードの表示が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

- sequence
- choice

エラーメッセージが表示された場合は，再度マッピング定義を起動し，変更を反映させてください。

09-50 より前のバージョンから移行する場合の注意

データ変換定義で，次に該当するスキーマを定義している場合，名前空間が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

1. import 先で対象名前空間（targetNamespace）を定義しているスキーマを，変換元ノードに指定している。
2. 1.の名前空間が，import または include 先を含むすべての名前空間宣言（xmlns[:prefix]）で定義されていない。

エラーメッセージが表示された場合は，マッピング定義を再度起動し，変更を反映させてください。

09-70 より前のバージョンから移行する場合の注意

データ変換定義で，次に該当するスキーマを定義している場合，名前空間が変更されるため，マッピング定義ファイルで使用している XML Schema が変更されている旨のエラーメッセージが表示されます。

1. 1 つの名前空間 URI に対して，複数の異なる名前空間接頭辞を指定している。

エラーメッセージが表示された場合は，マッピング定義を再度起動し，変更を反映させてください。

メモリについての注意

旧バージョンのリポジトリをインポートするときは，メモリを多く必要とします。そのため，次の手順でメモリサイズが十分かどうかを確認し，メモリが不足している場合は，メモリサイズを増やしてからリポジトリをインポートしてください。

1. Eclipse のメニューから，［ウィンドウ］－［設定］を選択します。
2. ダイアログ左側のツリービューから，［一般］を選択します。
3. ダイアログ右側の［ヒープ・ステータスを表示］チェックボックスにチェックします。
4. Eclipse のウィンドウの右下のゴミ箱アイコンを選択し，ガーベッジ・コレクターを実行します。
5. ゴミ箱アイコンの左に表示されているヒープ・サイズを確認します。
6. 未使用のヒープ・サイズが不足している場合，eclipse.ini を編集し，メモリサイズ（-Xmx の値）を大きくします。eclipse.ini の編集については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「付録 B.1　インストール」を参照してください。

開発環境にインポートしたリポジトリ情報を使用して実際に運用する場合，このリポジトリ情報を開発環境から運用環境にエクスポートし，運用環境から HCSC コンポーネントを配備します。

付録 B　評価版から製品版への移行

ここでは，評価版から製品版へ移行する手順について説明します。

付録 B.1　評価版からの移行

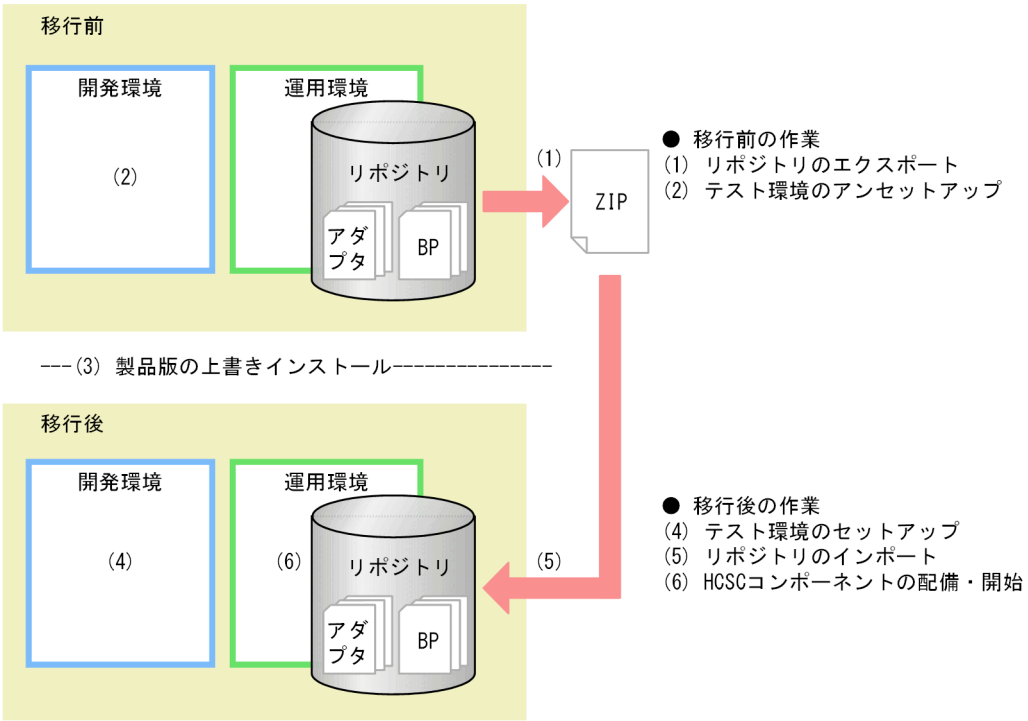
Service Architect の評価版を使用していた場合，評価版の運用環境のリポジトリ情報を製品版の運用環境に移行できます。

注意事項

プロセスインスタンスの実行履歴の情報は移行できません。

評価版から製品版への移行の流れを次の図に示します。

図 B-1　評価版からの移行の流れ



(1)　リポジトリのエクスポート

評価版の運用環境のリポジトリをエクスポートして，いったん退避します。複数のリポジトリを使用している場合，必要なリポジトリをすべてエクスポートして，退避します。

1. 次のコマンドを入力します。

```
cscrepctl -user admin -pass admin -export <エクスポートするリポジトリのファイル名（拡張子：.zip）>
```

cscrepctl コマンドの詳細については，マニュアル「サービスプラットフォーム リファレンス」の「cscrepctl（インポート/エクスポート）」を参照してください。

(2)　テスト環境のアンセットアップ

評価版の開発環境で構築したテスト環境をアンセットアップします。

テスト環境のアンセットアップ手順については，「2.4.2(2)　テスト環境のアンセットアップ」を参照してください。

(3)　製品版の上書きインストール

Service Architect の製品版を上書きインストールします。

注意

- J2EE サーバ，Management Server，PRF など実行環境の構成要素を停止してから上書きインストールしてください。
- 組み込みデータベースを使用している場合，組み込みデータベースを停止してから上書きインストールしてください。

(4)　テスト環境のセットアップ

開発環境でテスト環境を構築します。

手順の詳細は，「2.4.2(1)　テスト環境のセットアップ」を参照してください。

なお，HCSC 簡易セットアップ画面（[メイン] タブ）では，評価版のテスト環境のセットアップ時に指定した SOAP モード（SOAP 1.1 モードまたは SOAP 1.1/1.2 併用モード）と同じ SOAP モードを選択してください。異なる SOAP モードを選択した場合，正常に動作しないおそれがあります。

(5)　リポジトリのインポート

エクスポートしたリポジトリ情報を運用環境にインポートします。

1. 次のコマンドを入力します。

```
cscrepctl -user admin -pass admin -import <インポートするリポジトリのファイル名（拡張子：.zip）>
```

インポートしたリポジトリ情報を使用して開発する場合，このリポジトリ情報を開発環境にインポートします。

開発環境にリポジトリをインポートする手順については，「3.2.3　リポジトリをインポートする」を参照してください。

(6) HCSC コンポーネントの配備・開始

HCSC サーバを起動して，インポートした HCSC コンポーネントを配備・開始します。

1. 次のコマンドを入力して，HCSC サーバを起動します。

```
cscsvstart -user admin -pass admin -system
```

2. 次のコマンドを入力して，HCSC サーバに HCSC コンポーネントを配備します。

```
csccompodeploy -user admin -pass admin -csc MyCSC -all
```

3. 次のコマンドを入力して，配備した HCSC コンポーネントを起動します。

```
csccompostart -user admin -pass admin -csc MyCSC -all
```

4. 次のコマンドを入力して，標準受付を開始します。

```
cscrcptnstart -user admin -pass admin -csc MyCSC
```

付録 C　上流設計ツールを利用したシステムの開発

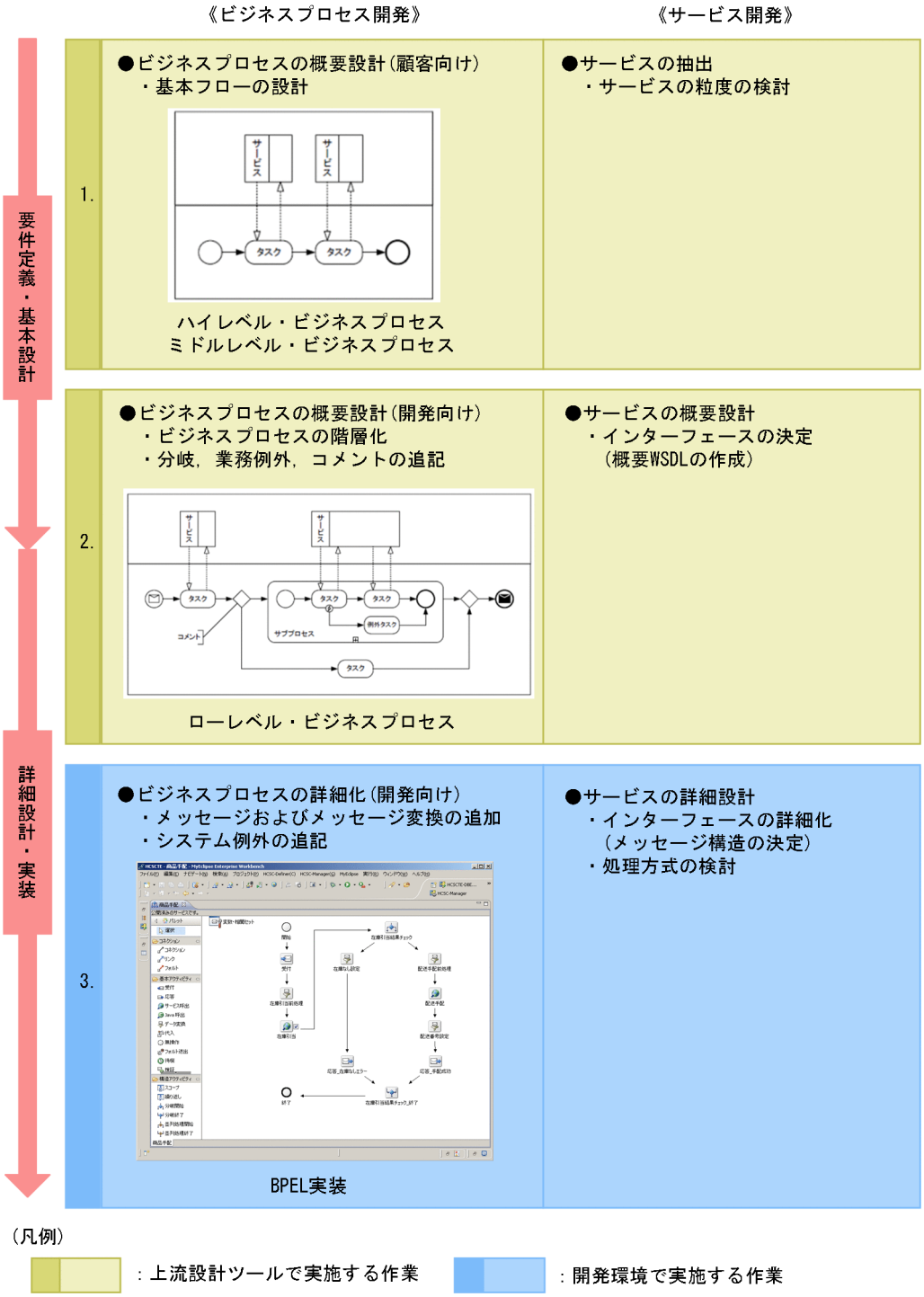
サービスプラットフォームの開発環境は，BPEL をサポートした上流設計ツールと連携してビジネスプロセスを開発できます。上流設計ツールの多くは，BPMN などのわかりやすいフロー表記法をサポートしているため，システム開発の上流工程で顧客を交えてビジネスプロセスを検討できるといった特長があります。

ここでは，上流設計ツールとサービスプラットフォームを組み合わせてシステムを開発する方法について説明します。

付録 C.1　上流設計ツールを利用したシステム開発の概要

上流設計ツールとサービスプラットフォームを組み合わせた場合のシステム開発の流れと，成果物のイメージを次の図に示します。

図 C-1　上流設計ツールとサービスプラットフォームを組み合わせたシステム開発



SOA に基づいたシステムの開発は，ビジネスプロセス開発とサービス開発の 2 つから成り，それぞれの成果物をドリルダウンしながら並行に作業を進めます。

各開発の作業手順とその内容を次の表に示します。

表 C-1　ビジネスプロセス開発とサービス開発の開発手順と作業内容

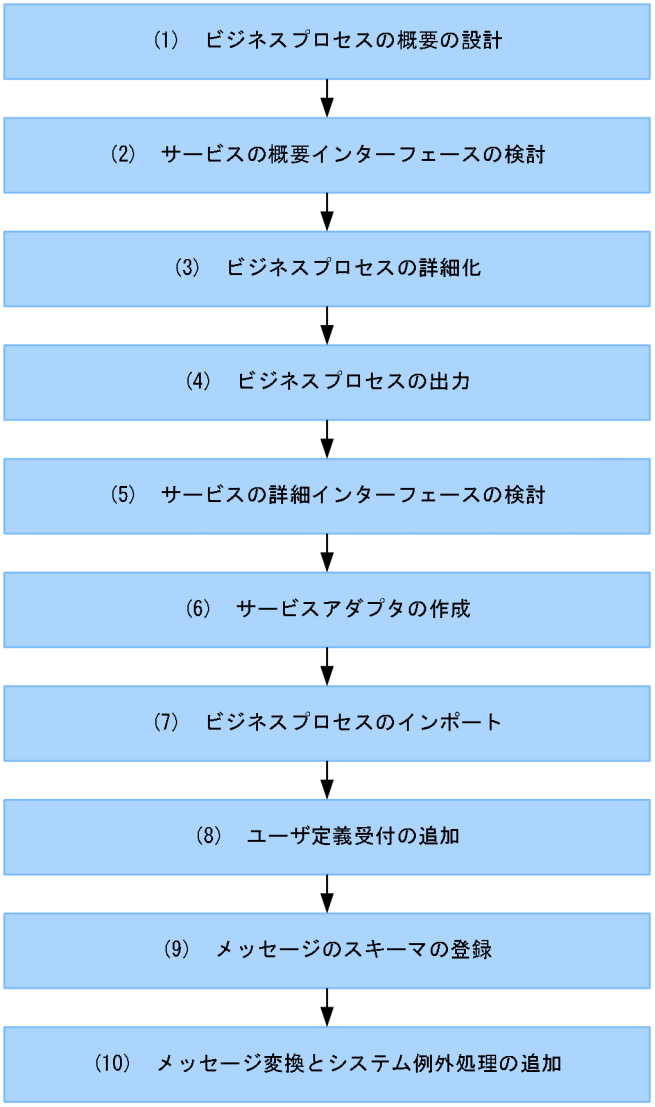
手順	ビジネスプロセス開発の作業	サービス開発の作業
1	●ビジネスプロセスの概要設計（顧客向け） システム化する業務の内容や範囲を顧客と検討するため，業務の基本フローをハイレベル・ビジネスプロセス，およびミドルレベル・ビジネスプロセスとして作成します。	●サービスの抽出 ビジネスプロセス設計と並行して，業務を処理するサービスの粒度を検討します。
2	●ビジネスプロセスの概要設計（開発向け） ビジネスプロセスを詳細化し，ローレベル・ビジネスプロセスにします。詳細化では，再利用性や可読性を考慮したフローの階層化や，業務例外への対応，コメントの追加などをします。	●サービスの概要設計 サービスのインターフェース（サービスが公開するオペレーションの一覧）を決定し，概要 WSDL を作成します。
3	●ビジネスプロセスの詳細化（開発向け） サービスの抽出および概要設計の結果を受けて，ビジネスプロセスにメッセージ定義やメッセージ変換の処理を追加します。また，実際に動かすことを想定したシステム例外への対応処理なども追加します。	●サービスの詳細設計 サービスのオペレーションが使用するメッセージの構造やサービスの処理方式を決定し，実装します。

上流設計ツールとサービスプラットフォームを組み合わせてシステムを開発する場合，手順 1 および手順 2 の範囲で上流設計ツールを使用し，手順 3 からサービスプラットフォームの開発環境を使用します。上流設計ツールから開発環境への設計の引き継ぎは，BPEL ファイルを通して行います。

付録 C.2　上流設計ツールを利用したシステム開発の流れ

上流設計ツールを利用したシステム開発の流れを次の図に示します。「付録 D　上流設計ツールを利用したシステムの開発例」では，この流れに沿ってサンプルプログラムを開発します。

図 C-2　上流設計ツールを利用したシステム開発の流れ



(1)　ビジネスプロセスの概要の設計

上流設計ツールでハイレベル・ビジネスプロセスを作成します。ハイレベル・ビジネスプロセスに対して顧客と検討を繰り返し，より具体化したミドルレベル・ビジネスプロセスにします。

(2)　サービスの概要インターフェースの検討

ミドルレベル・ビジネスプロセスの検討が終わったら，サービスの概要インターフェースを検討します。また，検討した結果を基に概要 WSDL を作成します。

(3)　ビジネスプロセスの詳細化

ビジネスプロセスを詳細化し，ローレベル・ビジネスプロセスにします。サービスプラットフォームに最適なビジネスプロセスとして設計するため，「付録 C.3　上流設計ツール利用時の前提事項」に従ってフローを設計してください。

(4)　ビジネスプロセスの出力

上流設計ツールで設計したビジネスプロセスを BPEL ファイル形式で出力します。このとき，ビジネスプロセスと連携するサービスの WSDL も一緒にまとめておきます。

(5)　サービスの詳細インターフェースの検討

これまでに検討した仕様を踏まえて，各サービスのオペレーションのメッセージ構造を検討します。また，検討した結果を基に，概要 WSDL にメッセージ構造の宣言を追加します。

(6)　サービスアダプタの作成

サービスプラットフォームの開発環境で，WSDL からサービスアダプタを作成します。サービスの名前は，WSDL に記載されたポートタイプ（portType）属性と同じにします。サービスアダプタを作成する方法については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.2 サービスアダプタの追加」を参照してください。

(7)　ビジネスプロセスのインポート

サービスプラットフォームの開発環境で，「(4)　ビジネスプロセスの出力」で出力した BPEL ファイルをビジネスプロセスとしてインポートします。インポートする方法については，「5.2.1　ビジネスプロセスを新規に追加する」を参照してください。

(8)　ユーザ定義受付の追加

ビジネスプロセスにユーザ定義受付を追加します。ユーザ定義受付を追加する方法については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「2.2.2　SOAP 受付の追加」を参照してください。

(9)　メッセージのスキーマの登録

ビジネスプロセスで使用するメッセージ（変数）のスキーマ（構造）を登録します。

(10)　メッセージ変換とシステム例外処理の追加

必要に応じて，ビジネスプロセスにメッセージ変換処理，システム例外の対応処理などを追加します。

付録 C.3　上流設計ツール利用時の前提事項

サービスプラットフォーム用のビジネスプロセスを設計するときの前提事項（上流設計ツールを使うときの注意点）について説明します。

(1)　ビジネスプロセス全体に関する前提事項

(a)　インスタンス補正の設定

サービスプラットフォームはインスタンス補正を行いません。上流設計ツールでビジネスプロセスのインスタンス補正を設定できる場合は，「補正無し」を指定してください。設定できない場合は，インスタンス補正されないことを前提にフローを設計してください。

(b)　抽象プロセスの指定

サービスプラットフォームは，実行可能な実装レベルのビジネスプロセスだけ扱うことができます。上流設計ツールでビジネスプロセスを抽象プロセスとして設定できる場合は，「抽象プロセス」として指定しないでください。

(c)　joinFailure フォルトの抑止

サービスプラットフォームは，joinFailure フォルトを常に抑止して動作します。上流設計ツールで joinFailure フォルトの抑止を設定できる場合は，「抑止する」を指定してください。設定できない場合は，joinFailure フォルトが抑止されることを前提にフローを設計してください。

(2)　アクティビティに関する前提事項

(a)　未サポート要素および属性の使用

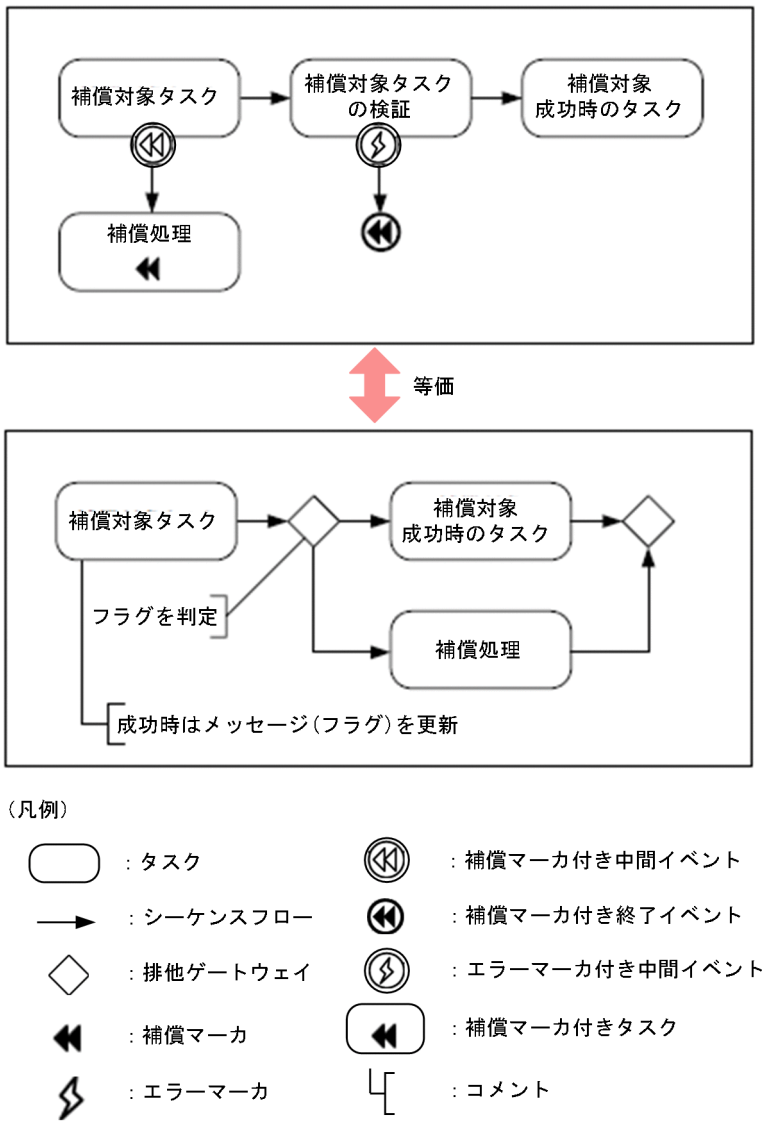
BPEL インポート機能がサポートしない BPEL の要素，および属性は使用しないでください。サポートしていない要素や属性は，取り込みのスキップ，またはほかの要素へ置き換えられるため，意図しないフローに変わることがあります。BPEL インポート機能のサポート状況については，「付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲」を参照してください。

また，上流設計ツールの各要素（BPMN 要素）や設定項目と，BPEL の各要素や属性との対応については，BPMN 仕様または上流設計ツールのマニュアルを参照してください。

(b)　補償ハンドラの使用

BPEL インポート機能は，compensationHandler 要素をサポートしていません。BPMN を利用した上流設計ツールで補償マーカを使用すると，compensationHandler 要素として BPEL に出力されることがあるため，補償マーカを使用しないで設計することを推奨します。補償マーカを使用しないで同等の設計をする例を示します。

図 C-3　補償マーカを使用しないで設計する場合の例



（凡例）

(c)　サービス名の文字種

サービスプラットフォームは，ビジネスプロセスと連携するサービスの名前に NCName だけ使用できます。上流設計ツールでサービスの名前（BPMN では Interface 属性）を設定する場合は，NCName で値を指定してください。NCName 以外の文字が使用されている場合，情報の取り込みが一部スキップされます。

(d)　メッセージの部分指定

上流設計ツールではメッセージの部分指定をしないでください。メッセージの部分指定は，メッセージの構造が確定する必要があるため，開発環境にビジネスプロセス定義を取り込んだあとで指定します。BPEL の assign アクティビティなどに部分指定（part）属性を指定した場合，BPEL インポート機能によって指定が無視されます。

(3) メッセージに関する前提事項

(a) メッセージタイプの指定

上流設計ツールではメッセージの構造を指定しないでください。メッセージの構造は，サービスのインターフェースが確定する必要があるため，開発環境にビジネスプロセスを取り込んだあとで指定します。BPELにメッセージタイプ（messageType）属性を指定した場合，string 型の変数に置き換えて取り込まれます。

(b) タイプの指定

メッセージが基本型（構造を持たない）の場合は，XML スキーマの名前空間（http://www.w3.org/2001/XMLSchema）に属する型を指定してください。BPEL のタイプ（type）属性にそのほかの型を指定した場合，互換性があると推測される型に置き換えて取り込まれます。

付録 C.4 上流設計ツール利用時のトラブルシューティング

上流設計ツールで作成した BPEL ファイルをインポートするときに表示されるメッセージについて，メッセージの詳細と解決方法を説明します。

ここで説明するメッセージの形式を次に示します。

```
メッセージテキスト
```

メッセージの説明

出力条件

メッセージの出力条件

対処方法

メッセージの対策および補足事項

メッセージテキスト

サービスプラットフォームが出力したメッセージテキストを示します。

[ ]で囲まれた部分は XPath で，{ }で囲まれた部分は値で表示されます。

メッセージの説明

メッセージテキストに対する補足説明を示します。

出力条件

メッセージが出力される条件を示します。

対処方法

メッセージが出力された場合の対処方法，または対処に関する補足事項を示します。

定義される BPEL の要素ごとに，出力されるメッセージを示します。

(1) プロセス

要素[process]の属性 targetNamespace はサポートしていないため反映されません。

サービスプラットフォームでは，常にシステム予約の固定値を targetNamespace 属性値として使用します。targetNamespace 属性値を指定しても，BPEL ファイルのインポート時にシステム予約の固定値に置き換えられます。

出力条件

targetNamespace 属性が定義されている場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

要素[process]の属性 enableInstanceCompensation はサポートしていないため反映されません。

サービスプラットフォームでは，enableInstanceCompensation 属性（インスタンス補正）はサポートしていません。そのため，常に enableInstanceCompensation 属性に「no」を指定した場合と同じ動作になります。

なお，BPEL1.1 の場合，enableInstanceCompensation 属性のデフォルト値は「no」です。BPEL2.0 の場合，enableInstanceCompensation 属性は指定できません。

出力条件

enableInstanceCompensation 属性が定義されている場合

対処方法

このメッセージが出力された場合，BPEL ファイルの enableInstanceCompensation 属性の指定を確認し，次の観点で対処してください。

- enableInstanceCompensation 属性に「no」を指定している場合
  サービスプラットフォームと同じ動作になります。特に対処の必要はありません。
- enableInstanceCompensation 属性に「yes」を指定している場合
  BPEL ファイルのインポート時に，enableInstanceCompensation 属性の指定は「no」に置き換えられます。そのため，インスタンス補正を前提にフローを設計している場合は，ビジネスプロセスの実行結果に差異が生じるおそれがあります。インポートしたビジネスプロセスをビジネスプロセスエディタで開いて，ビジネスプロセスが意図した内容に定義されているか見直してください。

インスタンス補正の内容はベンダーに依存します。インスタンス補正を前提にビジネスプロセスを作成した場合，実行時の互換性が保証できないことがあります。

そのため，上流設計の段階から enableInstanceCompensation 属性は指定しないか，または「no」を指定して設計してください。

要素[process]の属性 abstractProcess はサポートしていないため反映されません。

サービスプラットフォームでは，abstractProcess 属性はサポートしていません。サービスプラットフォームは，実行可能な実装レベルのビジネスプロセスを対象としています。そのため，常に abstractProcess 属性に「no」を指定した場合と同じ動作になります。

なお，BPEL1.1 の場合，abstractProcess 属性のデフォルト値は「no」です。BPEL2.0 の場合，abstractProcess 属性は指定できません。

出力条件

abstractProcess 属性が定義されている場合

対処方法

このメッセージが出力された場合，BPEL ファイルの abstractProcess 属性の指定を確認し，次の観点で対処してください。

- abstractProcess 属性に「no」を指定している場合
  サービスプラットフォームと同じ動作になります。特に対処の必要はありません。
- abstractProcess 属性に「yes」を指定している場合
  実行不可能な抽象レベルのビジネスプロセスであるおそれがあります。インポートしたビジネスプロセスをビジネスプロセスエディタで開いて，ビジネスプロセス定義を見直してください。

(2) パートナーリンク

要素[partnerLinks]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

サービスプラットフォームでは，partnerLinks 属性はサポートしていません。BPEL ファイルのインポートでは，ビジネスプロセスと外部サービスの関係を表す partnerLinks 要素は取り込みません。

出力条件

partnerLinks 要素が定義されている場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

サービスプラットフォームでは，BPEL ファイルのパートナーリンク情報はユーザ定義受付およびサービスアダプタに相当します。

必要に応じて，BPEL ファイルとセットになった WSDL ファイルからユーザ定義受付やサービスアダプタを作成してください。

次に示すメッセージの対処方法も合わせて参照してください。

- 参照先：(5)　アクティビティ（receive）
  要素[receive]の属性 partnerLink はサポートしていないため反映されません。
- 参照先：(6)　アクティビティ（reply）
  要素[reply]の属性 partnerLink はサポートしていないため反映されません。
- 参照先：(8)　アクティビティ（invoke）
  要素[invoke]の属性 partnerLink はサポートしていないため反映されません。

(3)　変数・相関セット（変数）

変数{変数名}は string 型に置き換えました。変数の種別を XML に変更し、電文フォーマットを登録してください。

BPEL ファイルのインポートでは，構造を持ったメッセージは変数として取り込みません。

上流設計ツールではメッセージ（変数）の名前だけを指定し，詳細設計に入ってからメッセージの構造を定義してください。

出力条件

variable 要素の messageType 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで変数の種別を XML に変更し，スキーマファイルを指定してください。

変数{変数名}の型{変更前の変数の型}は名前空間"http://www.w3.org/2001/XMLSchema"に属している必要があります。{変更後の変数の型}型として反映されますが、問題ないか確認してください。

XML スキーマの名前空間で定義されていない型が変数に指定されています。互換性があると推測される型に置き換えて，変数を取り込みました。

出力条件

XML スキーマの名前空間で定義されていない型が変数に指定されている場合

対処方法

置き換えられた変数の型がビジネスプロセスに影響ないか確認してください。必要に応じて，BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで変数の型を見直してください。

(4) 変数・相関セット（相関セット）

要素[correlationSets]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL ファイルのインポートでは，相関セットを定義した correlationSets 要素は取り込みません。

上流設計ツールでは相関セットを定義しないでください。

出力条件

correlationSets 要素が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで相関セットを定義し直してください。

(5) アクティビティ（receive）

要素[receive]の属性 partnerLink はサポートしていないため反映されません。

BPEL ファイルのインポートでは，partnerLink 属性は取り込みません。

出力条件

partnerLink 属性が定義されている場合

対処方法

標準受付を使う範囲では特に気にする必要はありません。

receive 要素の partnerLink 属性はサービスプラットフォームではユーザ定義受付に相当します。

必要に応じて，BPEL ファイルをインポートしたあとに，ビジネスプロセスの WSDL ファイルからユーザ定義受付を作成してください。

要素[receive]の属性 portType はサポートしていないため反映されません。

BPEL ファイルのインポートでは，portType 属性は取り込みません。

出力条件

portType 属性が定義されている場合

対処方法

portType 属性は BPEL ファイルと WSDL ファイルの可読性のために指定する属性です。ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

アクティビティ[receive]のオペレーション名は重複しているため{オペレーション名}に置き換えました。

異なる portType 属性値を持つ複数の receive 要素に，同じオペレーション名が指定されています。重複しないように，BPEL ファイルのインポート時にオペレーション名を置き換えて取り込みました。

出力条件

異なる portType 属性値を持つ複数の receive 要素に同じオペレーション名が指定されている場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。ただし，ビジネスプロセスの WSDL と一致しなくなるおそれがあるため，必要に応じてビジネスプロセスエディタでオペレーション名を変更してください。

receive アクティビティ[receive]は portType と operation が同一のものが既に存在するため、無操作アクティビティに置き換えました。

同じ portType 属性値を持つ複数の receive 要素に，同じオペレーション名が指定されています。BPEL ファイルのインポート時に，2 つ目以降の receive 要素を無操作アクティビティに置き換えて取り込みました。

出力条件

同じ portType 属性値を持つ複数の receive 要素に同じオペレーション名が指定されている場合

対処方法

BPEL ファイルをインポートしたあとに，無操作アクティビティを適切な受付アクティビティとして再定義してください。

要素[receive]の属性[createInstance]に指定されている値"no"を"yes"に置き換えました。

BPEL ファイルのインポートでは，receive 要素の createInstance 属性は常に「yes」として取り込みます。

出力条件

- createInstance 属性が指定されていない場合
- createInstance 属性に「no」が指定されている場合

対処方法

BPEL ファイルをインポートしたあとに，受付アクティビティのインスタンス生成を指定し直してください。

要素[correlations]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL ファイルのインポートでは，相関セットを指定した correlations 要素は取り込みません。上流設計ツールでは相関セットを指定しないでください。

出力条件

correlations 要素が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで相関セットを指定してください。

(6) アクティビティ（reply）

要素[reply]の属性 partnerLink はサポートしていないため反映されません。

BPEL ファイルのインポートでは，partnerLink 属性は取り込みません。

出力条件

partnerLink 属性が定義されている場合

対処方法

サービスプラットフォームでは，reply 要素の partnerLink 属性はユーザ定義受付に相当します。

必要であれば，BPEL ファイルをインポートしたあとに，ビジネスプロセスの WSDL ファイルからユーザ定義受付を作成してください。

要素[reply]の属性 portType はサポートしていないため反映されません。

BPEL ファイルのインポートでは，portType 属性は取り込みません。

出力条件

portType 属性が定義されている場合

対処方法

portType 属性は BPEL ファイルと WSDL ファイルの可読性のために指定する属性です。ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

要素[reply]の属性 faultName に指定されている値{名前空間}:{ローカル名}を{ローカル名}に置き換えました。

サービスプラットフォームでは，デフォルトではフォルトの名前空間をサポートしていません。

BPEL ファイルにフォルトの名前空間が指定されている場合は，ローカル名だけ取り込みます。

出力条件

faultName 属性値に名前空間指定がある場合

対処方法

標準受付を使用する場合，フォルト電文はローカル名だけ保持します。

ユーザ定義受付を使用する場合，フォルト電文は名前空間とローカル名の両方を保持します。

必要に応じて，フォルト電文がビジネスプロセスのインターフェースに従うように，ビジネスプロセスの WSDL ファイルからユーザ定義受付を作成してください。

アクティビティ[reply]のオペレーション名は重複しているため{オペレーション名}に置き換えました。

異なる portType 属性値を持つ複数の reply 要素に，同じオペレーション名が指定されています。重複しないように，BPEL ファイルのインポート時にオペレーション名を置き換えて取り込みました。

出力条件

異なる portType 属性値を持つ複数の reply 要素に同じオペレーション名が指定されている場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。ただし，ビジネスプロセスの WSDL と一致しなくなるおそれがあるため，必要に応じてビジネスプロセスエディタでオペレーション名を変更してください。

要素[correlations]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL ファイルのインポートでは，相関セットを指定した correlations 要素は取り込みません。上流設計ツールでは相関セットを指定しないでください。

出力条件

correlations 要素が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで相関セットを指定してください。

(7) アクティビティ（assign）

要素[assign/copy/from]の属性 part はサポートしていないため反映されません。

BPEL ファイルのインポートでは，part 属性は取り込みません。

上流設計ツールではメッセージ（変数）の名前だけを指定し，詳細設計に入ってからメッセージの構造を定義してください。

出力条件

part 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでメッセージの構造定義および部分指定をしてください。

要素[assign/copy/to]の属性 part はサポートしていないため反映されません。

BPEL ファイルのインポートでは，part 属性は取り込みません。

上流設計ツールではメッセージ（変数）の名前だけを指定し，詳細設計に入ってからメッセージの構造を定義してください。

出力条件

part 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでメッセージの構造定義および部分指定をしてください。

assign アクティビティ[assign]の copy 要素は、copy/from に定義されている属性がサポートしていない形式であるため反映されません。

assign アクティビティの copy 要素を取り込む場合，copy/from 要素の属性が取り込み可能な組み合わせに定義されている必要があります。

取り込み可能な組み合わせについては，「付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲」を参照してください。

出力条件

copy/from 要素の属性が取り込みできない組み合わせで定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，assign アクティビティの代入設定を定義し直してください。

assign アクティビティ[assign]の to 要素は、copy/to に定義されている属性がサポートしていない形式であるため反映されません。

assign アクティビティの to 要素を取り込む場合，copy/to 要素の属性が取り込み可能な組み合わせに定義されている必要があります。

取り込み可能な組み合わせについては，「付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲」を参照してください。

出力条件

copy/to 要素の属性が取り込みできない組み合わせで定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，assign アクティビティの代入設定を定義し直してください。

(8)　アクティビティ（invoke）

要素[invoke]の属性 partnerLink はサポートしていないため反映されません。

BPEL ファイルのインポートでは，partnerLink 属性は取り込みません。

出力条件

partnerLink 属性が定義されている場合

対処方法

サービスプラットフォームでは，invoke 要素の partnerLink 属性はサービスアダプタに相当します。すでに partnerLink 属性に対応するサービスアダプタを作成している場合は，特に対処の必要はありません。

リポジトリ内に該当するサービスまたはオペレーションが存在しないため、invoke アクティビティ[invoke]の portType に指定されているサービス名{サービス名}は反映されません。

portType 属性および operation 属性に対応するサービスアダプタがリポジトリにない場合，portType 属性は取り込みません。

出力条件

- portType 属性に対応するサービスアダプタがリポジトリに存在しない場合

- portType 属性に対応するサービスアダプタが operation 属性に対応するオペレーションを持たない場合

対処方法

対応するサービスアダプタを作成し，呼出アクティビティのサービス名を再設定してください。

リポジトリ内に該当するサービスまたはオペレーションが存在しないため、invoke アクティビティ[invoke]の operation に指定されているオペレーション名{オペレーション名}は反映されません。

portType 属性および operation 属性に対応するサービスアダプタがリポジトリにない場合，operation 属性は取り込みません。

出力条件

- portType 属性に対応するサービスアダプタがリポジトリに存在しない場合
- portType 属性に対応するサービスアダプタが operation 属性に対応するオペレーションを持たない場合

対処方法

対応するサービスアダプタを作成し，呼出アクティビティのオペレーション名を再設定してください。

invoke アクティビティ[invoke]に指定されている属性 inputVariable の値{変数名}はリポジトリ内に対応するオペレーション{オペレーション名}が存在しないため反映されません。

portType 属性および operation 属性に対応するサービスアダプタがリポジトリにない場合，inputVariable 属性は取り込みません。

出力条件

- portType 属性に対応するサービスアダプタがリポジトリに存在しない場合
- portType 属性に対応するサービスアダプタが operation 属性に対応するオペレーションを持たない場合

対処方法

対応するサービスアダプタを作成し，呼出アクティビティの要求電文用割当変数を再設定してください。

invoke アクティビティ[invoke]に指定されている属性 outputVariable の値{変数名}はリポジトリ内に対応するオペレーション{オペレーション名}が存在しないため反映されません。

portType 属性および operation 属性に対応するサービスアダプタがリポジトリにない場合，outputVariable 属性は取り込みません。

出力条件

- portType 属性に対応するサービスアダプタがリポジトリに存在しない場合
- portType 属性に対応するサービスアダプタが operation 属性に対応するオペレーションを持たない場合

対処方法

対応するサービスアダプタを作成し，呼出アクティビティの応答電文用割当変数を再設定してください。

invoke アクティビティ[invoke]に指定されている属性 outputVariable の値{変数名}はリポジトリ内のサービス{サービス名}のオペレーション{オペレーション名}が非同期のため反映されません。

呼び出し先サービスのオペレーションの通信方法が非同期の場合，応答電文は必要ないため，outputVariable 属性は取り込みません。

出力条件

呼び出し先サービスのオペレーションの通信方法が非同期で，outputVariable 属性が定義されている場合

対処方法

入力に使用した BPEL ファイルと WSDL ファイルの整合性が取れていないおそれがあります。オペレーションの通信方法が非同期で問題ないか確認してください。

要素[invoke/catch]の属性 faultName はサポートしていないため反映されません。

サービスプラットフォームでは，faultName 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultName 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

要素[invoke/catch]の属性 faultMessageType はサポートしていないため反映されません。

サービスプラットフォームでは，faultMessageType 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultMessageType 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

要素[invoke/catch]の属性 faultElement はサポートしていないため反映されません。

サービスプラットフォームでは，faultElement 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultElement 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

要素[correlations]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL ファイルのインポートでは，相関セットを指定した correlations 要素は取り込みません。

上流設計ツールでは相関セットを指定しないでください。

出力条件

correlations 要素が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで相関セットを指定してください。

(9) アクティビティ（flow）

要素[flow/links]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL ファイルのインポートでは，リンクを指定した links 要素は取り込みません。

上流設計ツールではリンクを指定しないでください。

出力条件

links 要素が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでリンクを指定してください。

(10) アクティビティ（throw）

要素[throw]の属性 faultName はサポートしていないため反映されません。

サービスプラットフォームでは，faultName 属性を使用したフォルトの送出をサポートしていません。

出力条件

faultName 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの送出を設定してください。

(11) アクティビティ（全般）

アクティビティ[someElement]の名称は重複しているため{新しい名称}に置き換えました。

サービスプラットフォームでは，アクティビティの名前（name 属性）を重複して指定できません。

BPEL ファイルのインポート時にアクティビティの名前が重複している場合，アクティビティの名前の末尾に数値を追加して取り込みます。数値は 1 から順番に指定されます。

出力条件

アクティビティの名前（name 属性）が重複している場合

対処方法

アクティビティの名前は運用環境の実行履歴や実行環境のログ出力に使用されます。必要に応じて，アクティビティの名前を変更してください。

要素[someElement]はサポートしていないため無操作アクティビティに置き換えました。また配下に属性や要素が存在する場合は、それらも反映されません。

サービスプラットフォームでは，[someElement]で表されるアクティビティはサポートしていません。

BPEL ファイルのインポート時に，[someElement]を無操作アクティビティに置き換えて取り込みました。

出力条件

サポートしていないアクティビティが定義されている場合

対処方法

サービスプラットフォームがサポートするアクティビティに，処理を置き換えてください。

(12) ハンドラ（補償ハンドラ）

要素[compensationHandler]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

BPEL インポート機能では，compensationHandler 要素はサポートしていません。

出力条件

compensationHandler 要素が定義されている場合

対処方法

サービスプラットフォームがサポートするアクティビティに，処理を置き換えてください。

(13) ハンドラ（イベントハンドラ）

要素[eventHandler]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

サービスプラットフォームでは，eventHandler 要素はサポートしていません。

出力条件

eventHandler 要素が定義されている場合

対処方法

サービスプラットフォームがサポートするアクティビティに，処理を置き換えてください。

(14) ハンドラ（終了ハンドラ）

要素[terminationHandler]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

サービスプラットフォームでは，terminationHandler 要素はサポートしていません。

出力条件

terminationHandler 要素が定義されている場合

対処方法

サービスプラットフォームがサポートするアクティビティに，処理を置き換えてください。

(15) ハンドラ（フォルトハンドラ）

faultHandlers[faultHandlers]は catch/catchAll が定義されていないため、反映されません。

catch または catchAll が 1 つ以上定義されていないフォルトハンドラは取り込みません。

出力条件

faultHandler 要素に catch 要素または catchAll 要素が 1 つ以上定義されていない場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

要素[faultHandlers/catch]の属性 faultName はサポートしていないため反映されません。

サービスプラットフォームでは，faultName 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultName 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

要素[faultHandlers/catch]の属性 faultMessageType はサポートしていないため反映されません。

サービスプラットフォームでは，faultMessageType 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultMessageType 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

要素[faultHandlers/catch]の属性 faultElement はサポートしていないため反映されません。

サービスプラットフォームでは，faultElement 属性を使用したフォルトの捕捉はサポートしていません。

出力条件

faultElement 属性が定義されている場合

対処方法

BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでフォルトの捕捉を再設定してください。

(16) その他

要素[someElement]の属性 suppressJoinFailure に指定されている値 no を yes に置き換えました。

サービスプラットフォームでは，suppressJoinFailure 属性はサポートしていません。そのため，常に bpws:joinFailure フォルトは抑止され，suppressJoinFailure 属性に「yes」を指定した場合と同じ動作になります。

なお，BPEL1.1 および BPEL2.0 の場合，suppressJoinFailure 属性のデフォルト値は「no」です。

出力条件

- suppressJoinFailure 属性が指定されていない場合
- suppressJoinFailure 属性に「no」が指定されている場合

対処方法

BPEL ファイルのインポート時に，suppressJoinFailure 属性の指定は「yes」に置き換えられます。そのため，ビジネスプロセスの実行結果に差異が生じるおそれがあります。BPEL ファイルをビジネスプロセスエディタで開いて，ビジネスプロセスが意図した内容に定義されているか見直してください。

注意事項

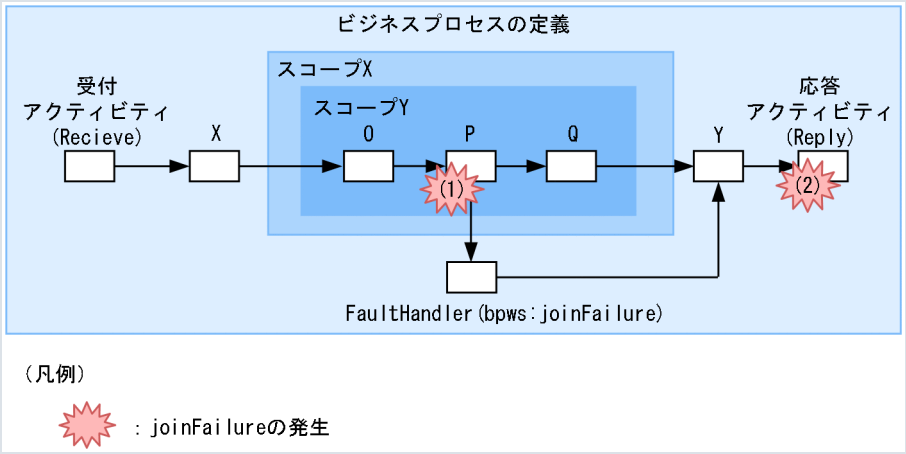
suppressJoinFailure 属性の値によって，ビジネスプロセスの動作が異なります。

サービスプラットフォームでは，常に suppressJoinFailure 属性が「yes」として動作するため，アクティビティの joinCondition が満たされない場合，ビジネスプロセスは直ちに再開できない中断状態になります。

suppressJoinFailure 属性が「no」として動作する環境では，joinCondition が満たされない場合，bpws:joinFailure フォルトがスローされても，ビジネスプロセス定義に従って処理を続行します。

suppressJoinFailure 属性の指定と bpws:joinFailure の関係を次の図に示します。

図 C-4　suppressJoinFailure 属性の指定と bpws:joinFailure の関係



suppressJoinFailure 属性が"no"の場合：

（1）の joinFailure が発生したときは，joinFailure が上位のスコープに伝播されます。この例では，スコープ Y→スコープ X→プロセスの順に伝播しますが，スコープ X の FaultHandler で joinFailure のエラー処理が行われ，処理が継続されます。

（2）の joinFailure が発生したときは，プロセスの上位に伝播しようとしますが，joinFailure を捕捉するものがないため，ビジネスプロセスが停止します。このとき，ビジネスプロセスは再開可能な中断状態になります。

suppressJoinFailure 属性が"yes"の場合：

（1）の joinFailure が発生したときは，joinFailure の伝播が抑止され，ビジネスプロセスは再開不能な中断状態になります。

（2）の joinFailure が発生したときも，joinFailure の伝播が抑止され，ビジネスプロセスは再開不能な中断状態になります。

要素[someElement]の属性 queryLanguage はサポートしていないため反映されません。

サービスプラットフォームでは，queryLanguage 属性はサポートしていません。クエリ言語として常に XPath1.0 を使用します。サービスプラットフォームの XPath1.0 は，一部独自の拡張があります。

なお，BPEL1.1 および BPEL2.0 の場合，クエリのデフォルト値は XPath1.0 です。

出力条件

queryLanguage 属性が定義されている場合

対処方法

上流設計ツールでクエリを使用していない場合は，特に対処の必要はありません。

XPath1.0 以外のクエリ言語を使用している場合は，BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタでクエリの指定を修正してください。

要素[someElement]の属性 expressionLanguage はサポートしていないため反映されません。

サービスプラットフォームでは，expressionLanguage 属性はサポートしていません。式言語として常に XPath1.0 を使用します。サービスプラットフォームの XPath1.0 は，一部独自の拡張があります。

なお，BPEL1.1 および BPEL2.0 の場合，式のデフォルト値は XPath1.0 です。

出力条件

expressionLanguage 属性が定義されている場合

対処方法

上流設計ツールで式を使用していない場合は，特に対処の必要はありません。

XPath1.0 以外の式を使用している場合は，BPEL ファイルをインポートしたあとに，ビジネスプロセスエディタで式の指定を修正してください。

要素[someElement]にアクティビティが定義されていないため、無操作アクティビティを追加しました。

子要素を持つことができる BPEL 要素（while アクティビティなど）に子要素が 1 つ以上定義されていない場合，BPEL ファイルのインポート時に，無操作アクティビティを子要素として追加します。

出力条件

[someElement]の activity 要素が子要素を持たない場合

対処方法

ビジネスプロセスの実行には影響しません。特に対処の必要はありません。

要素[someElement]はサポートしていないため反映されません。また配下に属性や要素が存在する場合は、それらも反映されません。

サービスプラットフォームでは，[someElement]で表される BPEL 要素はサポートしていません。

出力条件

サポートしていない要素が定義されている場合

対処方法

サービスプラットフォームがサポートする機能に，処理を置き換えてください。

要素[someElement]の属性[someAttribute]はサポートしていないため反映されません。

サービスプラットフォームでは，[someAttribute]で表される BPEL 属性はサポートしていません。

出力条件

サポートしていない属性が定義されている場合

対処方法

サービスプラットフォームがサポートする機能に，処理を置き換えてください。

付録 D　上流設計ツールを利用したシステムの開発例

上流設計ツールとサービスプラットフォームを連携したシステム開発の例を説明します。開発するシステムの概要，および使用する上流設計ツールを次に示します。

開発するシステムの概要

ビジネスプロセスからサービスアダプタを介して商品の在庫を引き当てたり，配送を手配したりするサービス部品を呼び出す「商品手配システム」を開発します。

サービスプラットフォームでは，商品手配システムのサンプルプログラムを提供しています。

商品手配システムのサンプルプログラムの格納先

商品手配システムのサンプルプログラムは，次の場所にあります。

<Service Architectのインストールディレクトリ>¥CSCTE¥Samples¥SOAP1.1_1.2mode¥ProductStock

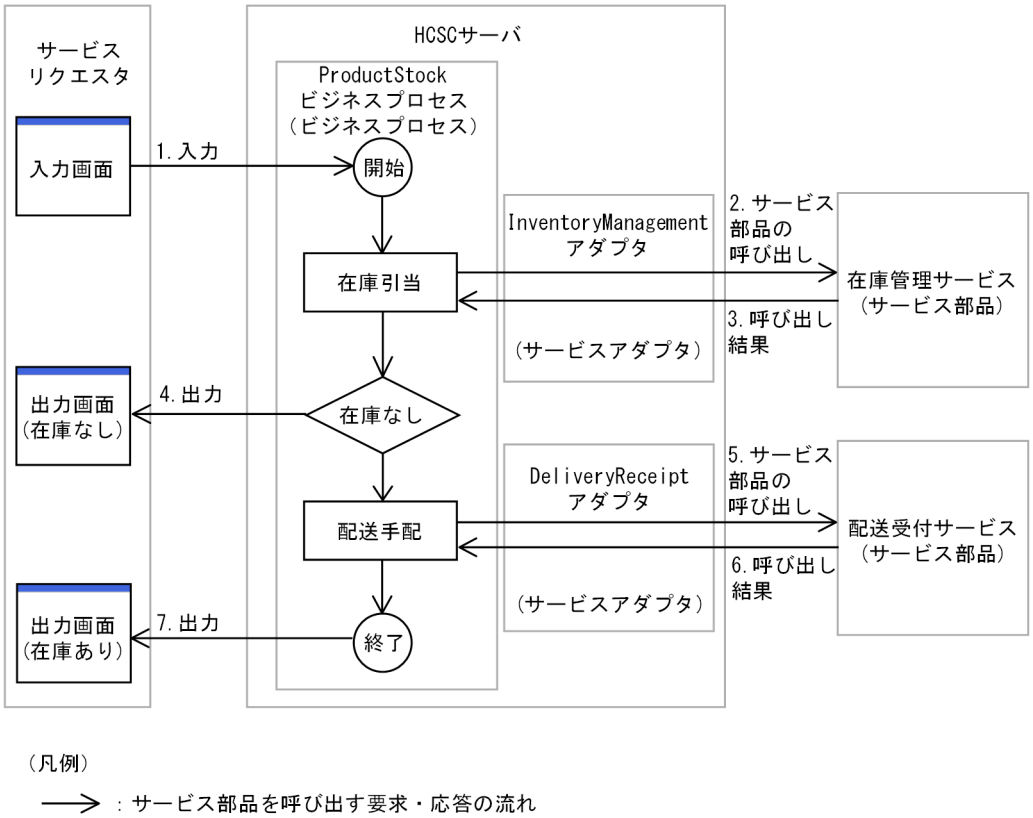
商品手配システムの処理内容

商品手配システムは，サービスリクエスタからの要求に基づいて，注文商品の在庫引当や配送手配を行います。

ここで説明する商品手配システムは，店舗の担当者が業務システムを使って商品の手配をする場合に，在庫引当や配送手配の処理をすることを想定しています。

商品手配システムの処理の詳細を次の図および表に示します。

図 D-1　商品手配システムの処理詳細



（凡例）

⟶ ：サービス部品を呼び出す要求・応答の流れ

表 D-1　商品手配システムの処理内容

コンポーネント	形態	内容
サービスリクエスタ	JSP サーブレット	• 入力画面を表示し，ProductStock ビジネスプロセスを呼び出します。 • 在庫管理サービスを呼び出した結果を戻り値として取得し，出力画面に表示します。 • 配送受付サービスを呼び出した結果を戻り値として取得し，出力画面に表示します。
ProductStock ビジネスプロセス	ビジネスプロセス	• サービスリクエスタからの要求を受け取り，InventoryManagement アダプタを介して在庫管理サービスを呼び出します。 • サービスリクエスタからの要求を受け取り，DeliveryReceipt アダプタを介して配送受付サービスを呼び出します。 • 在庫管理サービスを呼び出した結果を InventoryManagement アダプタを介して取得し，サービスリクエスタへ返します。 • 配送受付サービスを呼び出した結果を DeliveryReceipt アダプタを介して取得し，サービスリクエスタへ返します。
InventoryManagement アダプタ	サービスアダプタ	• ProductStock ビジネスプロセスからの要求を受け取り，在庫管理サービスを呼び出します。 • 在庫管理サービスの応答を ProductStock ビジネスプロセスへ返します。
DeliveryReceipt アダプタ	サービスアダプタ	• ProductStock ビジネスプロセスからの要求を受け取り，配送受付サービスを呼び出します。 • 配送受付サービスの応答を ProductStock ビジネスプロセスへ返します。
在庫管理サービス	サービス部品	• ProductStock ビジネスプロセスからの呼び出しに応答して商品の在庫を引き当て，ProductStock ビジネスプロセスへ引当番号を返します。
配送受付サービス	サービス部品	• ProductStock ビジネスプロセスからの呼び出しに応答し，配送手配をした結果の配送番号を返します。

なお，ここでの開発では，次に示す内容が日本語表記になります。

- サービス名
- アクティビティ名

使用する上流設計ツール

Process Modeler
（Process Modeler 5 for Microsoft Visio Professional Edition）

付録 D.1　ビジネスプロセスの概要を設計する

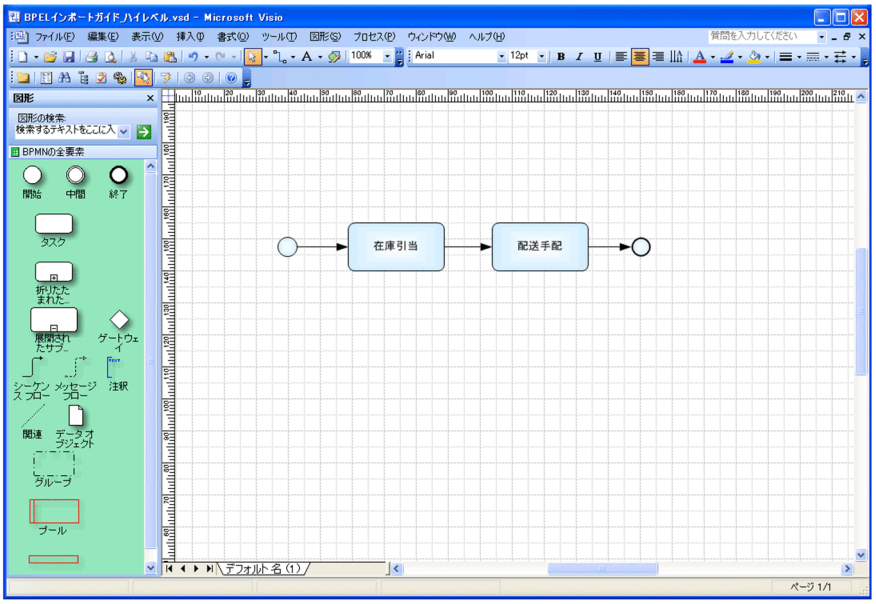
業務の基本フローをハイレベル・ビジネスプロセスとして作成し，作成したハイレベル・ビジネスプロセスを具体化します。

(1)　ハイレベル・ビジネスプロセスを作成する

上流設計ツールでハイレベル・ビジネスプロセスを作成します。商品手配システムのビジネスプロセスの場合，システム化する業務は在庫引当と配送手配の 2 つのタスクなので，これを基本フローとして記述します。

商品手配システムのハイレベル・ビジネスプロセスの記述例を次に示します。

図 D–2　ハイレベル・ビジネスプロセスの記述例

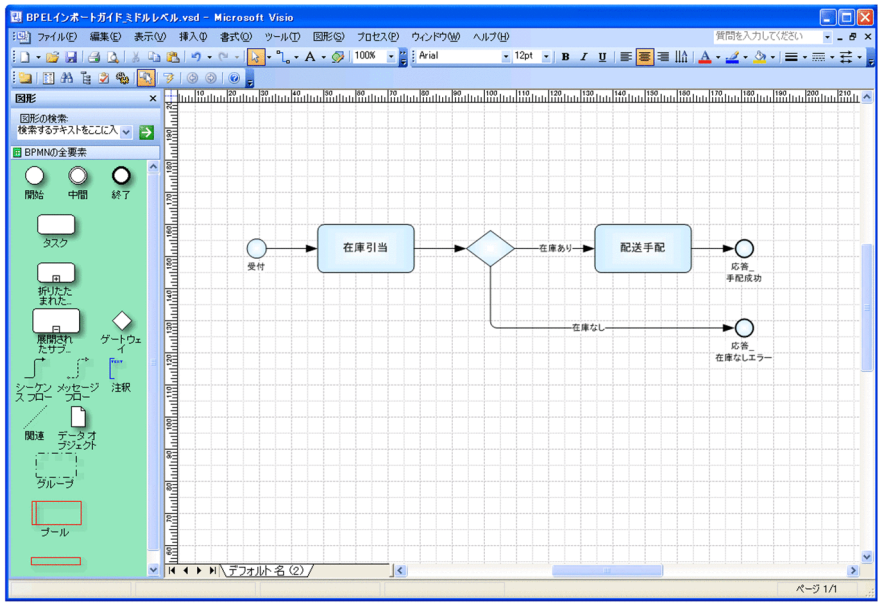


(2)　ミドルレベル・ビジネスプロセスを作成する

作成したハイレベル・ビジネスプロセスを具体化し，業務全体の流れが把握できるミドルレベル・ビジネスプロセスを作成します。ここでは，在庫引当タスクのあとに排他ゲートウェイを追加し，手配を受けた商品の在庫がない場合の業務エラーフローを追加します。作成したビジネスプロセスに対して顧客と検討を繰り返し，検討不足の業務フローや改善点があればブラッシュアップします。

商品手配システムのミドルレベル・ビジネスプロセスの記述例を次に示します。

図 D-3　ミドルレベル・ビジネスプロセスの記述例



商品手配システムでは，次の表に示す要素でビジネスプロセスを実現することにします。

表 D-2　商品手配ビジネスプロセスを構成する要素

項番	名前	要素の種類	備考
1	受付	開始ノード	
2	在庫引当	タスク	
3	－	排他ゲートウェイ	在庫引当の結果を判定
4	配送手配	タスク	
5	応答_手配成功	終了ノード	配送手配が成功したときの終了ノード
6	応答_在庫なしエラー	終了ノード	在庫がなかったときの終了ノード

（凡例）

－：名前はありません。

空欄：備考として説明する内容はありません。

また，この段階でサービスの粒度も検討します。ここでは，次の表に示すように在庫引当タスクと配送手配タスクをそれぞれ 1 つのサービスで実現することにします。

表 D-3　サービスの粒度の検討

項番	タスク	サービス（実装時のサービス名）
1	在庫引当	在庫引当サービス（InventoryManager）
2	配送手配	配送手配サービス（Delivery）

付録 D.2　サービスの概要インターフェースを検討する

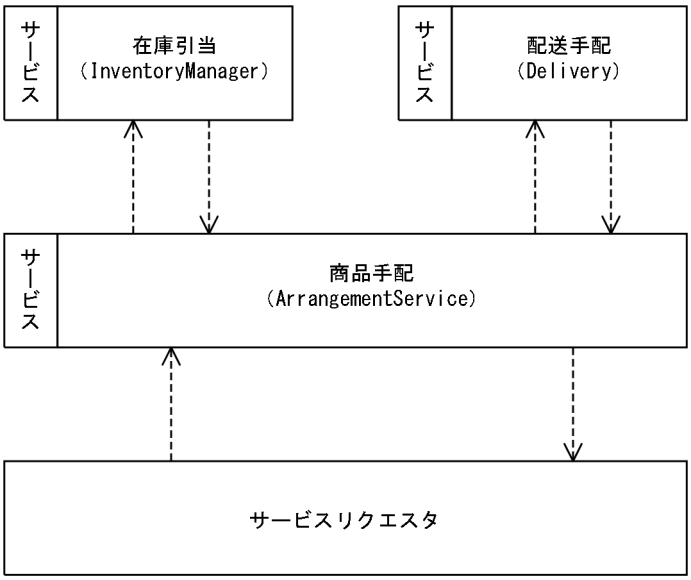
ミドルレベル・ビジネスプロセスの検討が終わったら，サービスの概要インターフェースを検討し，検討した結果を基に概要 WSDL を作成します。検討するサービスを次に示します。

- 在庫引当サービス（InventoryManager）
- 配送手配サービス（Delivery）
- 商品手配ビジネスプロセス （ArrangementService）

(1)　概要インターフェースを検討する

最初に，サービスの概要インターフェースを検討します。次の図のように，サービスリクエスタから見ると商品手配ビジネスプロセス自体もサービスになるため，概要インターフェースの検討対象になります。

図 D-4　サービスのコンポーネント構成



商品手配システムでは，次の図および次の表のように概要インターフェースを決定したものとします。

図 D-5　サービスの概要インターフェース

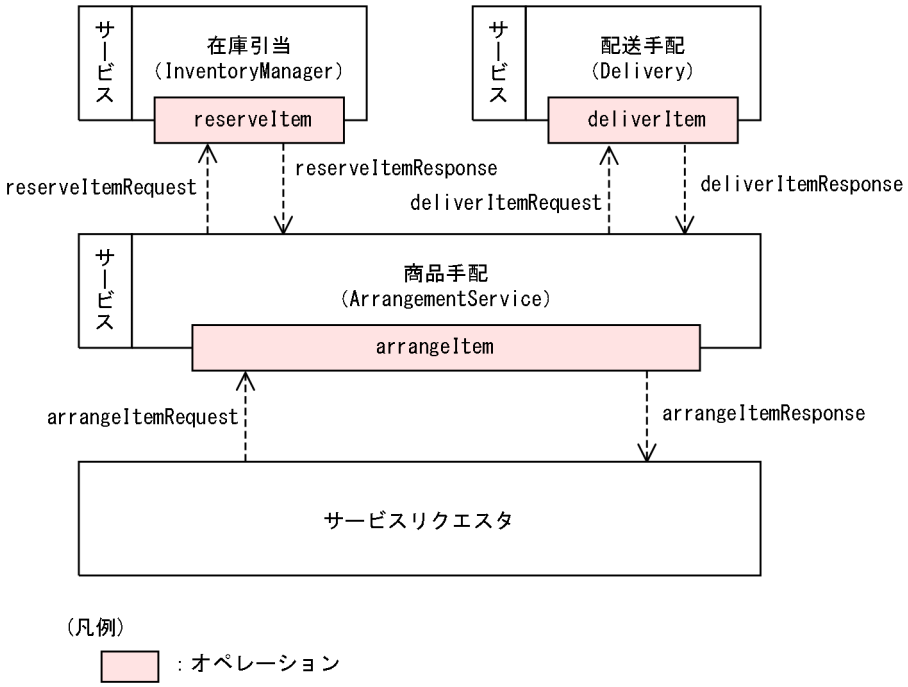


表 D-4　概要インターフェースの詳細

項番	サービス名	提供オペレーション名 (オペレーションの内容)	要求メッセージ名	応答メッセージ名
1	在庫引当サービス	reserveItem (商品を在庫引当する)	reserveItemRequest	reserveItemResponse
2	配送手配サービス	deliverItem (商品を配送する)	deliverItemRequest	deliverItemResponse
3	商品手配ビジネスプロセス	arrangeItem (商品を手配し配送する)	arrangeItemRequest	arrangeItemResponse

(2)　概要 WSDL を作成する

概要インターフェースが決定したら，概要 WSDL を作成します。商品手配システムでは，サービスごとに次の表に示す名前で WSDL ファイルを作成します。

表 D-5　サービスごとの WSDL ファイルの名前

項番	サービス名	WSDL ファイルの名前
1	在庫引当サービス	InventoryManagementService.wsdl
2	配送手配サービス	DeliveryService.wsdl
3	商品手配ビジネスプロセス	ArrangementService.wsdl

概要 WSDL を作成するときのポイントは次のとおりです。

概要 WSDL 作成のポイント

- サービス名を wsdl:port の name 属性に記述する。
- 提供オペレーション名を wsdl:operation の name 属性に記述する。
- 要求メッセージ名を wsdl:input の name 属性に記述する。
- 応答メッセージ名を wsdl:output の name 属性に記述する。
- メッセージの構造は記述しない
  （ルート要素だけを宣言し，型は xsd:string などの基本型にする）。

在庫引当サービスの WSDL（InventoryManagementService.wsdl）の例を次に示します。WSDL 中の太字は「概要 WSDL 作成のポイント」で示した部分の記述を表します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
      targetNamespace="http://sample/InventoryManagementService"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ims="http://sample/InventoryManagementService">

   <wsdl:types>
      <xsd:schema
            targetNamespace="http://sample/InventoryManagementService"
            elementFormDefault="qualified">
         <xsd:element name="reserveItem" type="xsd:string" />
         <xsd:element name="reserveItemResponse" type="xsd:string" />
      </xsd:schema>
   </wsdl:types>

   <wsdl:message name="reserveItemRequest">
      <wsdl:part name="parameters" element="ims:reserveItem" />
   </wsdl:message>

   <wsdl:message name="reserveItemResponse">
      <wsdl:part name="parameters" element="ims:reserveItemResponse" />
   </wsdl:message>

   <wsdl:portType name="InventoryManager">
      <wsdl:operation name="reserveItem">
         <wsdl:input message="ims:reserveItemRequest" name="reserveItemRequest" />
         <wsdl:output message="ims:reserveItemResponse" name="reserveItemResponse" />
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="InventoryManagerSoapBinding" type="ims:InventoryManager">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="reserveItem">
         <soap:operation soapAction="" />
         <wsdl:input name="reserveItemRequest">
            <soap:body use="literal" />
```

```
         </wsdl:input>
         <wsdl:output name="reserveItemResponse">
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="InventoryManagementService">
      <wsdl:port binding="ims:InventoryManagerSoapBinding" name="InventoryManager">
         <soap:address location="http://localhost/InventoryManagementServiceWeb/services/Inv
entoryManager" />
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

付録 D.3　ビジネスプロセスを詳細化する

「付録 D.1(2)　ミドルレベル・ビジネスプロセスを作成する」で作成したミドルレベル・ビジネスプロセスに設定を追加し，詳細化します。

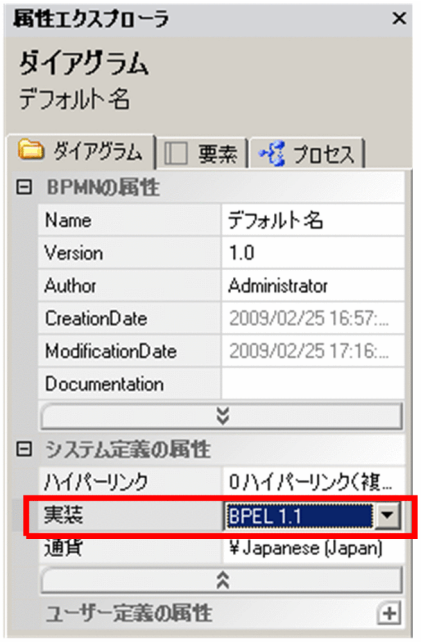
(1)　実装方式を決定する

最初に，ダイアグラムの実装方式を設定します。

1. Process Modeler のウィンドウメニューから［プロセス］－［表示］－［属性エクスプローラ］を選択します。
   ウィンドウに［属性エクスプローラ］が表示されます。

2. フローの何もないところをマウスでクリックし，ダイアグラムを選択します。
   ［属性エクスプローラ］にダイアグラムの属性が表示されます。

3. ［システム定義の属性］を展開し，［実装］のドロップダウンリストから「BPEL 1.1」を選択します。

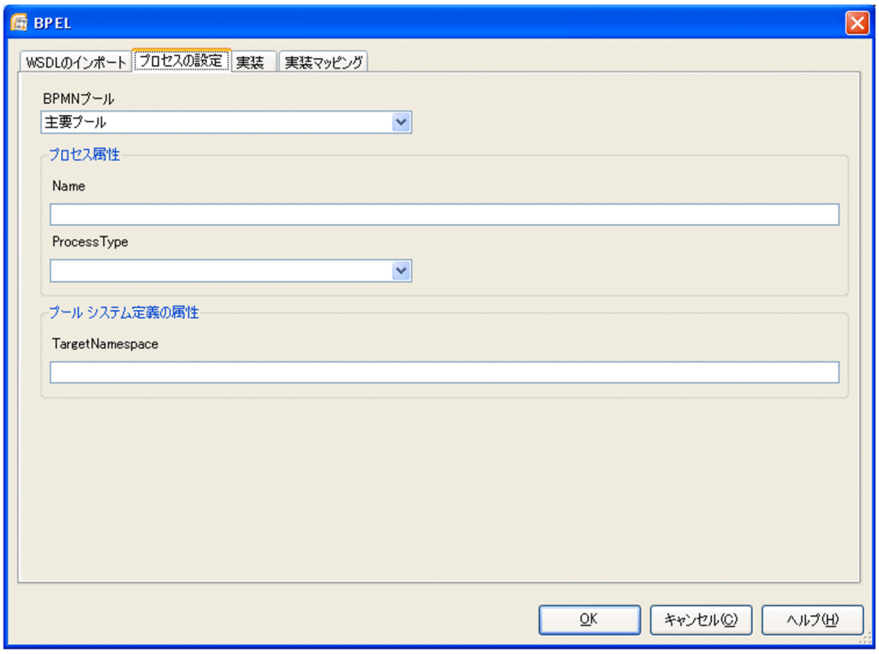


(2)　プロセスの属性を設定する

ビジネスプロセス全体の属性を設定します。

1. Process Modeler のウィンドウメニューから［プロセス］－［実装］－［BPEL］を選択します。
   ［BPEL］ダイアログが表示されます。

2. ［プロセスの設定］タブを選択します。



3. ［プロセス属性］と［プール システム定義の属性］を次のとおりに指定します。

   - Name：未指定または任意の値※

- ProcessType：未指定
- TargetNamespace：未指定または任意の値※

注※

任意の値を指定できますが，BPEL インポート機能ではこれらの値は使用しません。

(3) ノード・タスクを詳細化する

ビジネスプロセスの開始ノード，終了ノード，およびタスクの詳細な設定をします。

最初に，サービスリクエスタからの要求メッセージによってビジネスプロセスが開始されるように設定します。開始ノードに設定する手順を示します。

1. ビジネスプロセスの開始ノードを選択します。

2. 右クリックして表示されるメニューから［Trigger］－［Message］を選択します。

また，ビジネスプロセスの実行結果をサービスリクエスタに応答メッセージとして返すようにするため，終了ノードも同じように，右クリックして表示されるメニューから［Result］－［Message］を選択します。

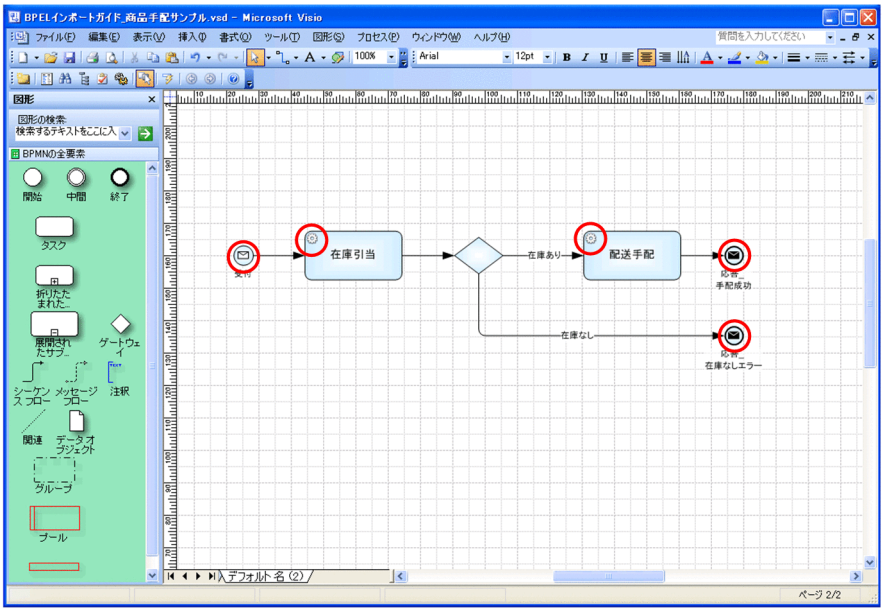
次に，在庫引当タスクと配送手配タスクをサービスとして実行することを表すために，在庫引当タスクと配送手配タスクをそれぞれ右クリックしてメニューを表示し，［TaskType］－［Service］を選択します。

開始ノード，終了ノード，およびタスクの設定内容は次のとおりです。

表 D-6　開始ノード／終了ノード／タスクの設定内容

項番	要素	設定項目	設定値
1	開始ノード	Trigger	Message
2	在庫引当タスク	TaskType	Service
3	配送手配タスク	TaskType	Service
4	終了ノード	Result	Message

ここまでの手順を終えると，開始ノード，終了ノード，およびタスクにメッセージを表す手紙のアイコンや，サービスを表す歯車のアイコンが追加されます。

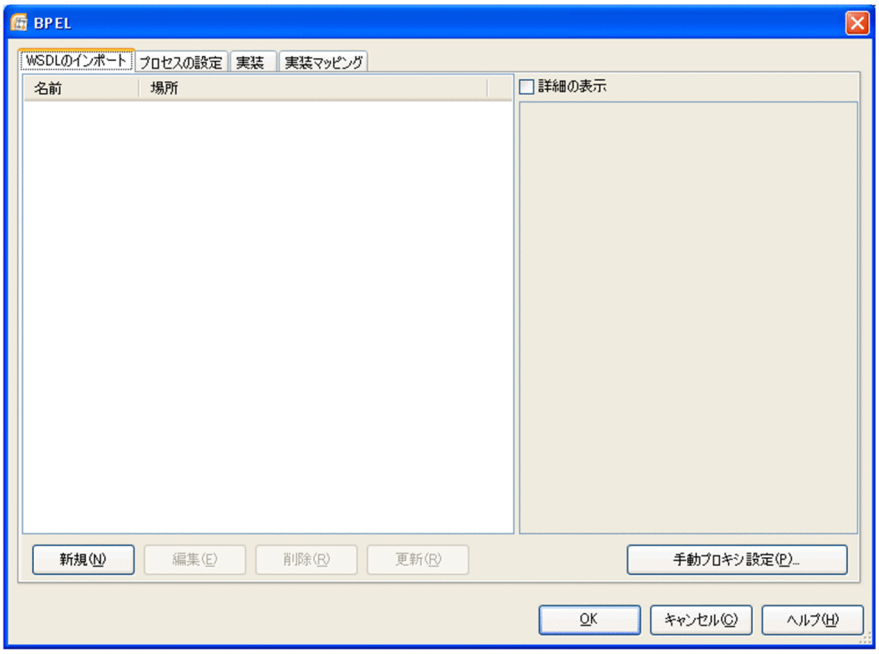


(4)　WSDL を登録する

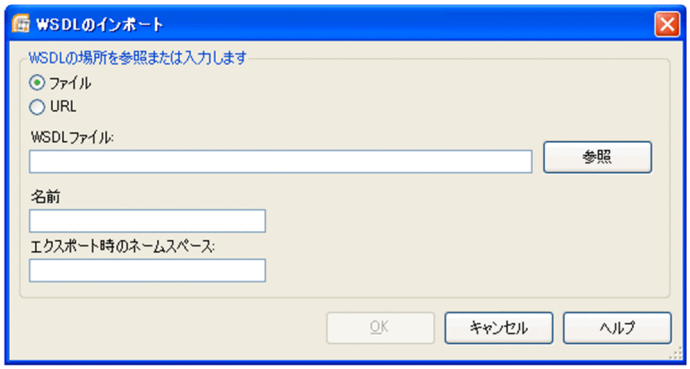
「付録 D.2(2)　概要 WSDL を作成する」で作成したサービスの概要 WSDL を Process Modeler に登録します。

1. Process Modeler のウィンドウメニューから［プロセス］－［実装］－［BPEL］を選択します。
   ［BPEL］ダイアログが表示されます。

2. ［WSDL のインポート］タブを選択します。



3. ［新規］ボタンをクリックします。
   ［WSDL のインポート］ダイアログが表示されます。



4. WSDL ごとに次の内容を入力します。

表 D-7　インポート対象の WSDL と設定値

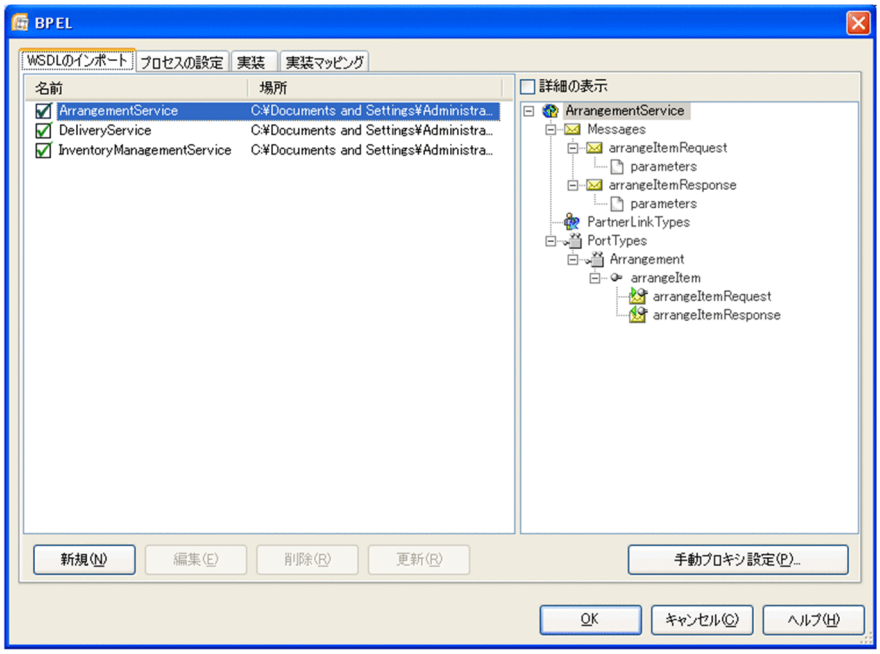
項番	インポート対象	項目	設定値
1	ArrangementService.wsdl （商品手配ビジネスプロセスの WSDL）	WSDL の場所	ファイル
		WSDL ファイル	［参照］ボタンから選択
		名前	ArrangementService
		エクスポート時のネームスペース	任意の値※
2	InventoryManagementService.wsdl （在庫引当サービスの WSDL）	WSDL の場所	ファイル
		WSDL ファイル	［参照］ボタンから選択
		名前	InventoryManagementService
		エクスポート時のネームスペース	任意の値※
3	DeliveryService.wsdl （配送手配サービスの WSDL）	WSDL の場所	ファイル
		WSDL ファイル	［参照］ボタンから選択
		名前	DeliveryService
		エクスポート時のネームスペース	任意の値※

注※

任意の値を指定できますが，BPEL インポート機能でこれらの値を使用しません。

5. ［OK］ボタンをクリックします。

WSDL が登録されます。表 D-7 で示した WSDL をすべてインポートすると，［WSDL のインポート］タブは次のようになります。



(5)　オペレーションを割り当てる

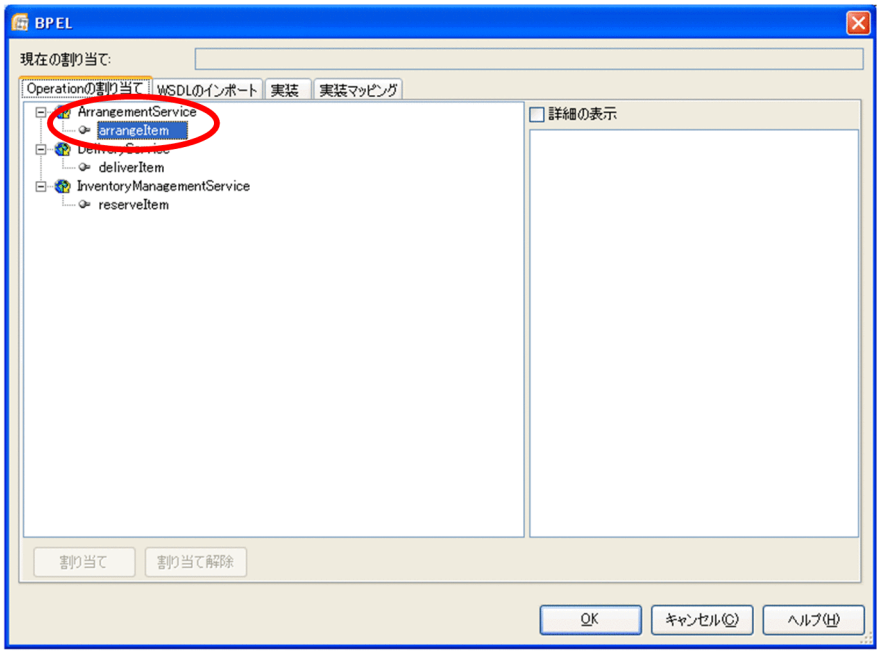
登録した WSDL を使用して，開始ノード，終了ノード，およびタスクがどのように外部のサービスリクエスタやサービスと連携するか，オペレーションとそのインターフェースを設定します。

最初に，開始ノードのインターフェースを設定します。

1. 開始ノードを右クリックして表示されるメニューから［実装］－［BPEL］を選択します。
   ［BPEL］ダイアログが表示されます。

2. ［Operation の割り当て］タブを選択します。

3. ［ArrangementService］－［arrangeItem］を選択して，［割り当て］ボタンをクリックします。
   開始ノードのオペレーションは，名前「ArrangementService」の WSDL に「arrangeItem」として定義されているため，「arrangeItem」を割り当てます。



残りのノード，タスクについても，次の表を参考にオペレーションを割り当てます。

表 D-8　［Operation の割り当て］タブの設定内容

項番	要素の名前	WSDL の名前	Operation の名前
1	受付	ArrangementService	arrangeItem
2	在庫引当	InventoryManagementService	reserveItem
3	配送手配	DeliveryService	deliverItem
4	応答_手配成功	ArrangementService	arrangeItem
5	応答_在庫なしエラー	ArrangementService	arrangeItem

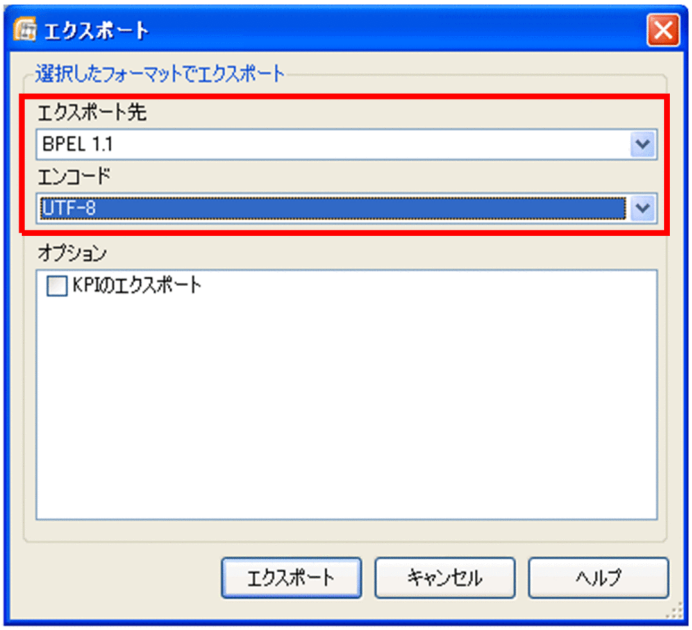
以上で，上流設計ツールを使用したビジネスプロセス設計は終了です。

付録 D.4　ビジネスプロセスを出力する

上流設計ツールで作成したビジネスプロセスをサービスプラットフォームの開発環境に引き継ぐために，BPEL ファイルとして出力します。

1. Process Modeler のウィンドウメニューから［プロセス］－［ダイアグラム］－［エクスポート］を選択します。
   ［エクスポート］ダイアログが表示されます。

2. ［エクスポート先］が「BPEL 1.1」になっていることを確認し，［エンコード］を「UTF-8」に変更します。



3. [エクスポート] ボタンをクリックします。
   [フォルダの参照] ダイアログが表示されます。

4. BPEL ファイルを出力する場所を選択し，[OK] ボタンをクリックします。
   選択したフォルダに次に示すファイルが生成されます。

   - 主要プール.bpel
     作成したビジネスプロセスの BPEL ファイルです。
   - 主要プール_types.wsdl
     このファイルは使用しません。

   なお，プールに名前を付けた場合は，<プールの名前>.bpel と<プールの名前>_types.wsdl が生成されます。

付録 D.5　サービスの詳細インターフェースを検討する

これまでに検討した仕様を踏まえて，各サービスのオペレーションのメッセージ構造を決定します。具体的には，「付録 D.2(2)　概要 WSDL を作成する」で作成した概要 WSDL にメッセージ構造の宣言を追加します。

各サービスの要求メッセージと応答メッセージには，次の表に示す情報が必要になるため，これを概要 WSDL のメッセージ宣言に反映（ルート要素の下に宣言を追加）します。

表 D-9　サービスの要求／応答メッセージに必要な情報

項番	サービス名	メッセージ	必要な情報		
			名前	型	説明
1	在庫引当サービス	reserveItemRequest	ItemName	xsd:string	在庫引当する商品の名前
			Quantity	xsd:int	在庫引当する個数
		reserveItemResponse	ReservationNumber	xsd:string	引当番号
2	配送手配サービス	deliverItemRequest	ReservationNumber	xsd:string	配送手配する引当番号
		deliverItemResponse	DeliveryNumber	xsd:string	配送番号
3	商品手配ビジネスプロセス	arrangeItemRequest	ItemName	xsd:string	手配する商品の名前
			Quantity	xsd:int	手配する商品の個数
		arrangeItemResponse	DeliveryNumber	xsd:string	配送番号

在庫引当サービスを例に WSDL（InventoryManagementService.wsdl）の定義例を次に示します。WSDL 中の太字は，追記した内容を表します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
      targetNamespace="http://sample/InventoryManagementService"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ims="http://sample/InventoryManagementService">

   <wsdl:types>
      <xsd:schema
            targetNamespace="http://sample/InventoryManagementService"
            elementFormDefault="qualified">
         <xsd:element name="reserveItem">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="ItemName" type="xsd:string" />
                  <xsd:element name="Quantity" type="xsd:int" />
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="reserveItemResponse">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="ReservationNumber" type="xsd:string" />
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>

   <wsdl:message name="reserveItemRequest">
      <wsdl:part name="parameters" element="ims:reserveItem" />
   </wsdl:message>
```

```
   <wsdl:message name="reserveItemResponse">
      <wsdl:part name="parameters" element="ims:reserveItemResponse" />
   </wsdl:message>

   <wsdl:portType name="InventoryManager">
      <wsdl:operation name="reserveItem">
         <wsdl:input message="ims:reserveItemRequest" name="reserveItemRequest" />
         <wsdl:output message="ims:reserveItemResponse" name="reserveItemResponse" />
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="InventoryManagerSoapBinding" type="ims:InventoryManager">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="reserveItem">
         <soap:operation soapAction="" />
         <wsdl:input name="reserveItemRequest">
            <soap:body use="literal" />
         </wsdl:input>
      <wsdl:output name="reserveItemResponse">
            <soap:body use="literal" />
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="InventoryManagementService">
      <wsdl:port binding="ims:InventoryManagerSoapBinding" name="InventoryManager">
         <soap:address location="http://localhost/InventoryManagementServiceWeb/services/Inv
entoryManager" />
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

詳細インターフェースを記載した WSDL は，サンプルプログラムに同梱されています。サンプルプログラムの格納場所については，マニュアル「サービスプラットフォーム ファーストステップガイド」の「付録 A　サービス部品のサンプルプログラムのファイル構成」を参照してください。

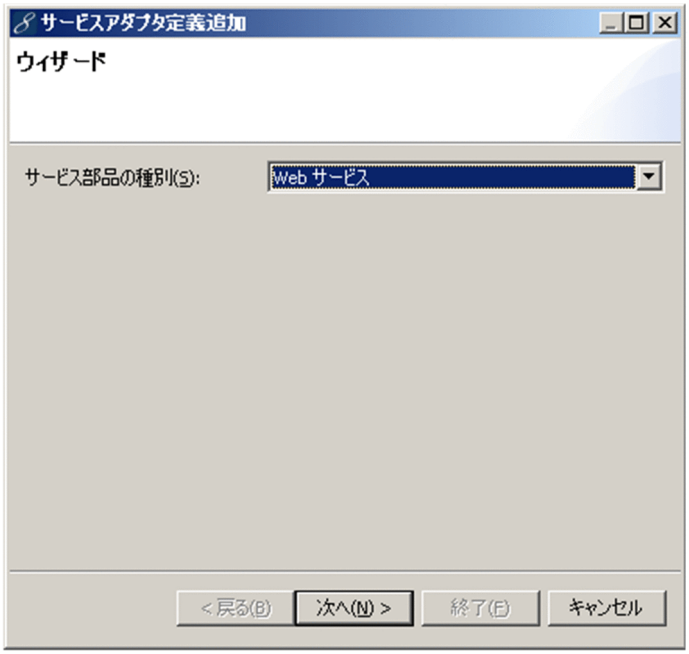
付録 D.6　サービスアダプタを作成する

「付録 D.5　サービスの詳細インターフェースを検討する」で詳細化した WSDL を使用して，ビジネスプロセスと，在庫引当サービスおよび配送手配サービスをつなげるためのサービスアダプタを追加します。

まず，在庫管理サービス（InventoryManager）のサービスアダプタを作成します。

1. ツリービューのサービス定義一覧で右クリックして「サービスアダプタ追加」を選択します。
   サービスアダプタ定義追加ウィザードが表示されます。

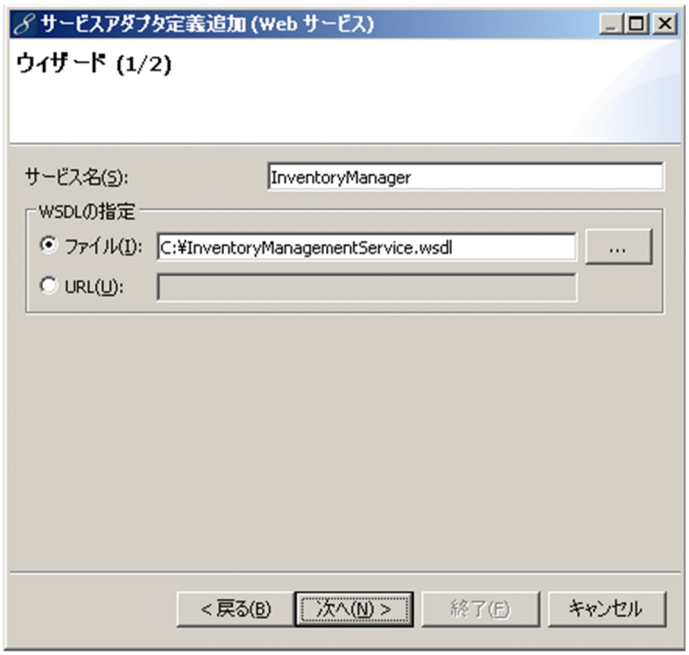
2. [サービス部品の種別] に「Web サービス」を指定します。



3. [次へ] ボタンをクリックします。

   サービス名とサービスの WSDL ファイルを指定するページが表示されます。

4. サービス名と作成した在庫管理サービスの WSDL を指定します。

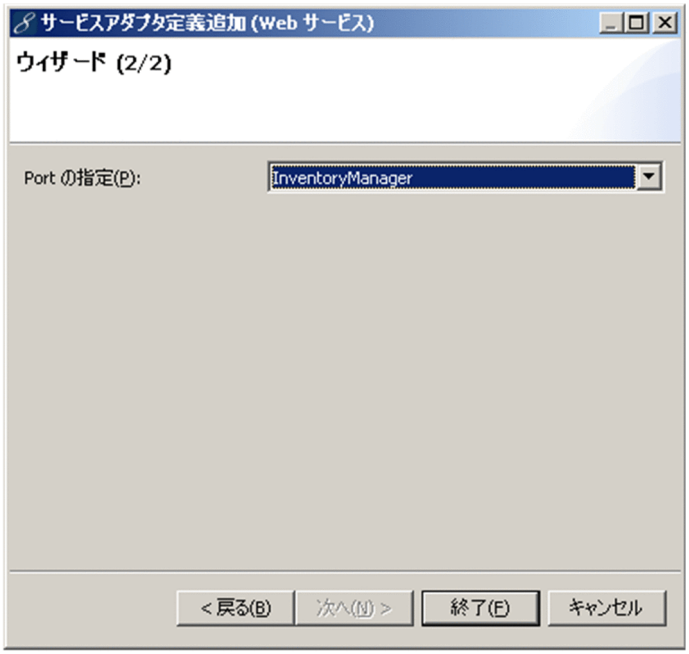


   注意

   サービス名は WSDL に書かれたポートタイプ要素（portType）の名前属性値（name）と同じ値にしてください。

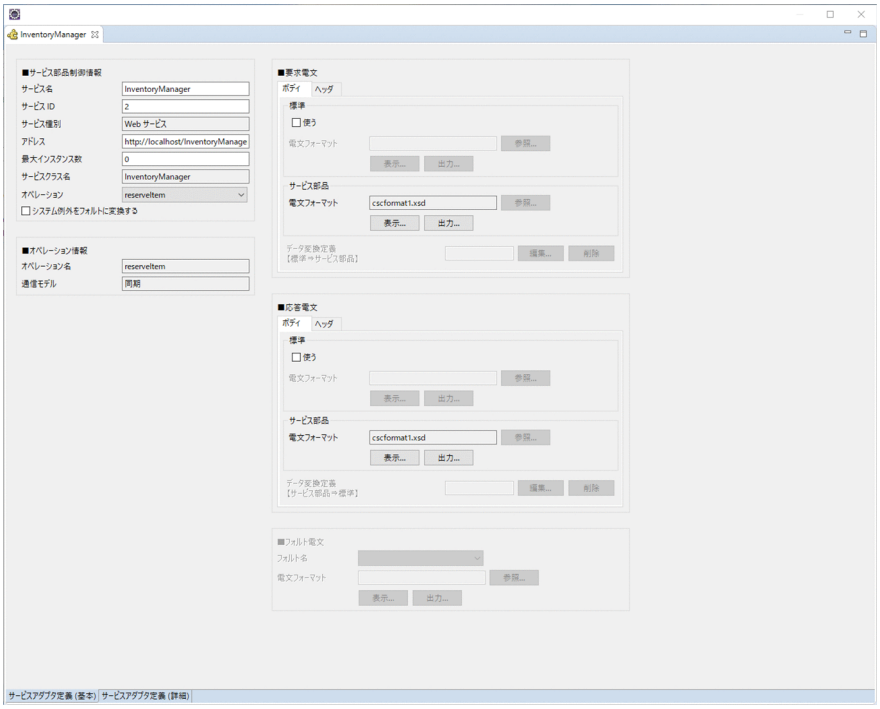
5. [次へ] ボタンをクリックします。

   WSDL のポートを指定するページが表示されます。



6. [Port の指定] が「InventoryManager」になっていることを確認して，[終了] ボタンをクリックします。

7. WSDL の解析が終わるとサービスアダプタが作成されます。



配送手配サービス（Delivery）についても，次に示す設定をして，サービスアダプタを作成します。

表 D-10　配送手配サービスのサービスアダプタの設定値

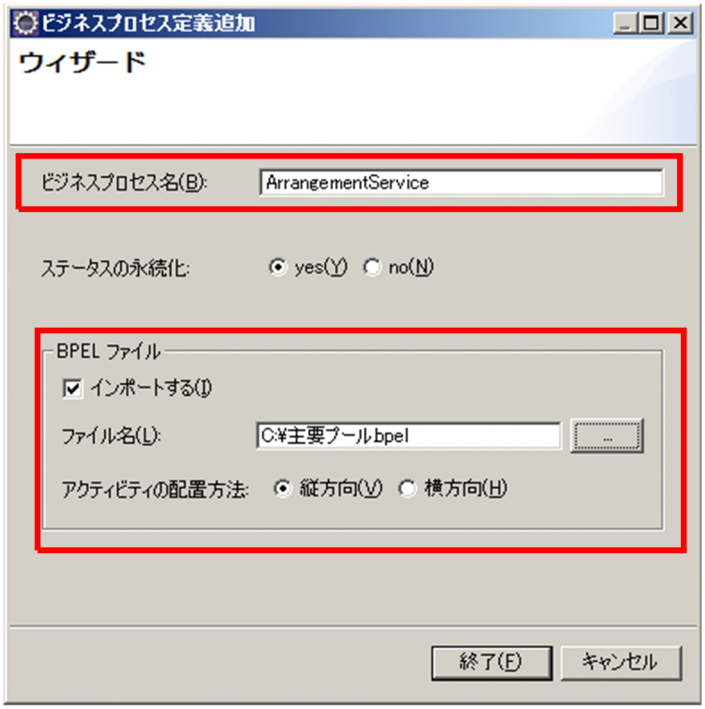
項番	項目	設定値
1	サービス部品の種別	Web サービス
2	サービス名	Delivery
3	WSDL ファイル	DeliveryService.WSDL
4	ポート	Delivery

付録 D.7　ビジネスプロセスをインポートする

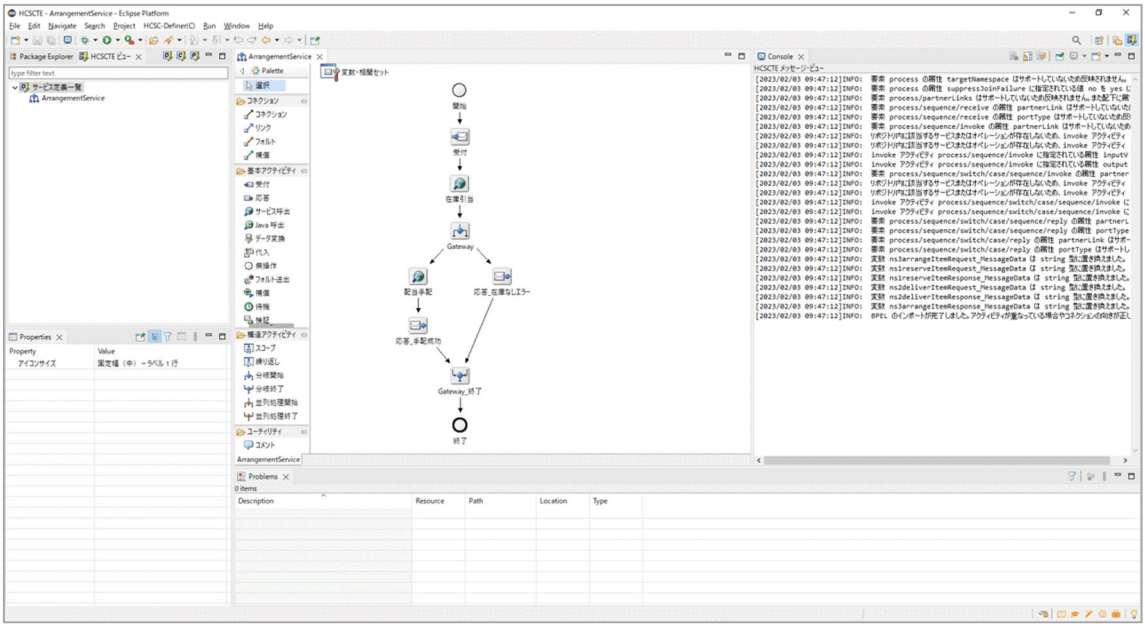
「付録 D.4　ビジネスプロセスを出力する」で出力した BPEL ファイルをサービスプラットフォームの開発環境に取り込みます。

1. ツリービューのサービス定義一覧で右クリックして「ビジネスプロセス追加」を選択します。
   ビジネスプロセス定義追加ウィザードが表示されます。

2. ビジネスプロセス名と出力した BPEL ファイルを指定します。



3. [終了] ボタンをクリックします。
   BPEL ファイルのインポートが終わると，上流設計ツールから引き継いだビジネスプロセスが表示されます。



ここでは，次のように BPMN 要素が変換されています。

表 D–11　BPMN 要素の変換結果

項番	BPMN 要素	ビジネスプロセスの要素
1	「受付」開始ノード	「受付」受付アクティビティ
2	「在庫引当」タスク	「在庫引当」サービス呼出アクティビティ
3	排他ゲートウェイ	「Gateway」分岐開始アクティビティ 「Gateway_終了」分岐終了アクティビティ
4	「配送手配」タスク	「配送手配」サービス呼出アクティビティ
5	「応答_手配成功」終了ノード	「応答_手配成功」応答アクティビティ
6	「応答_在庫なしエラー」終了ノード	「応答_在庫なしエラー」応答アクティビティ

4. コンソールビューにインポートのログが表示されているので，BPEL ファイルが正しく取り込まれているかを確認します。

   ログに出力された問題の解決方法については，「付録 C.4　上流設計ツール利用時のトラブルシューティング」を参照してください。

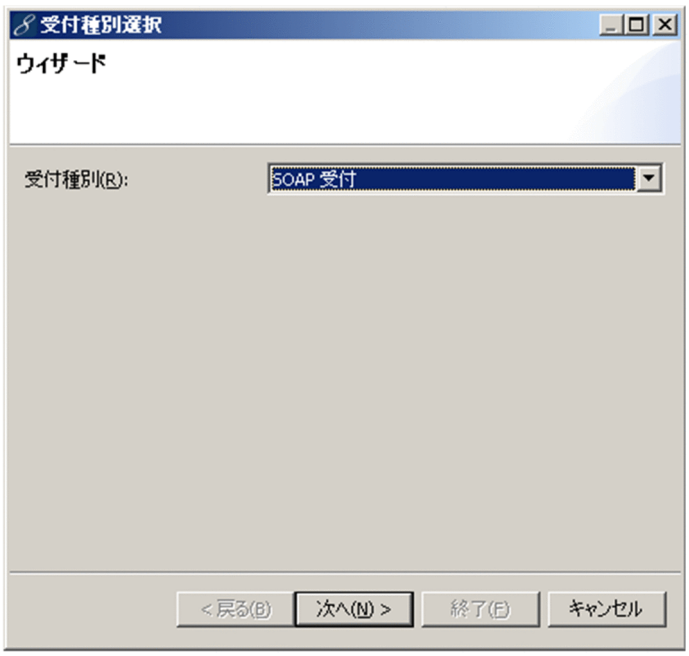
付録 D.8　ユーザ定義受付を追加する

「付録 D.5　サービスの詳細インターフェースを検討する」で詳細化した WSDL を使用して，ビジネスプロセスにユーザ定義受付を追加します。

1. ツリービューのサービス定義一覧で ArrangementService（商品手配サービス）を選択し，右クリックして［ユーザ定義受付（呼出先固定）追加］を選択します。

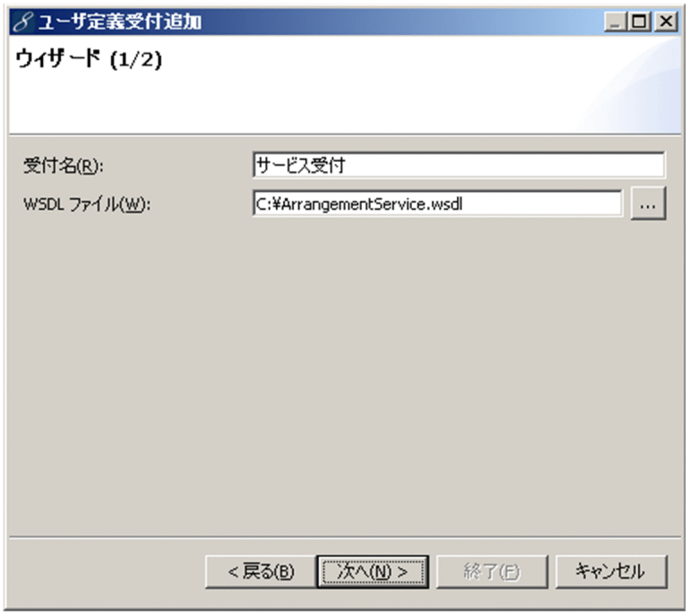
受付種別選択ウィザードが表示されます。

2. [受付種別] のドロップダウンリストから [SOAP 受付] を選択します。



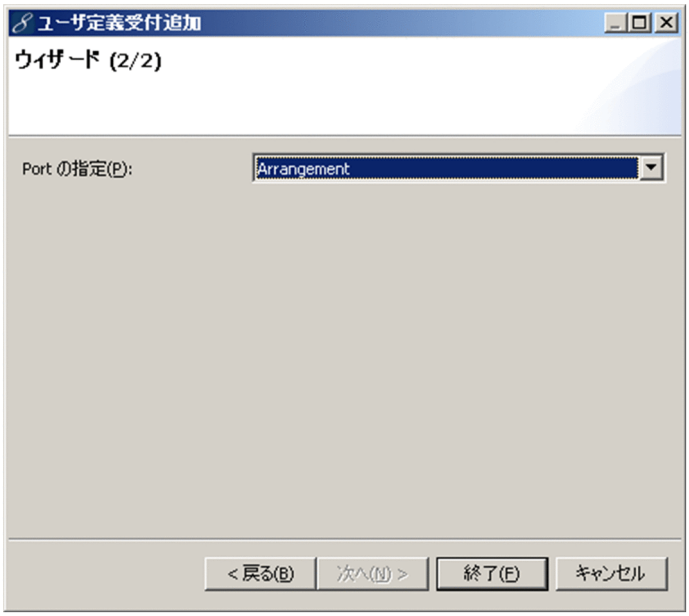
3. [次へ] ボタンをクリックします。
ユーザ定義受付追加ウィザードが表示されます。

4. 受付名と作成したビジネスプロセスの WSDL を指定します。

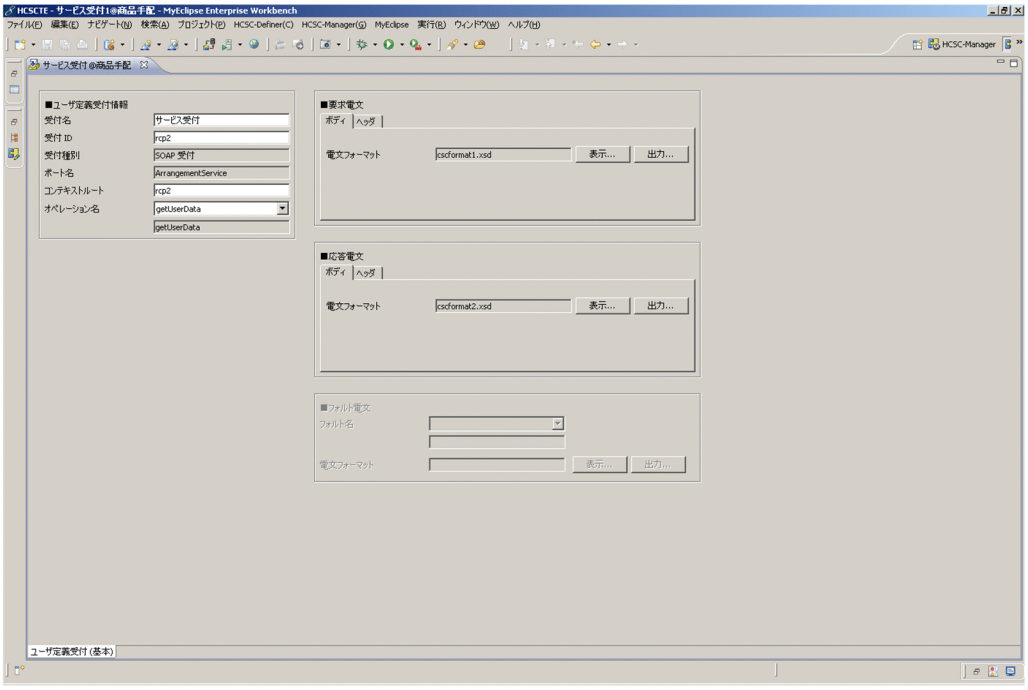


5. [次へ] ボタンをクリックします。
ポートを指定するページが表示されます。

6. [Port の指定] が「Arrangement」になっていることを確認して [終了] ボタンをクリックします。



WSDL の解析が終わると，次のようにユーザ定義受付が作成されます。



付録 D.9　メッセージのスキーマを登録する

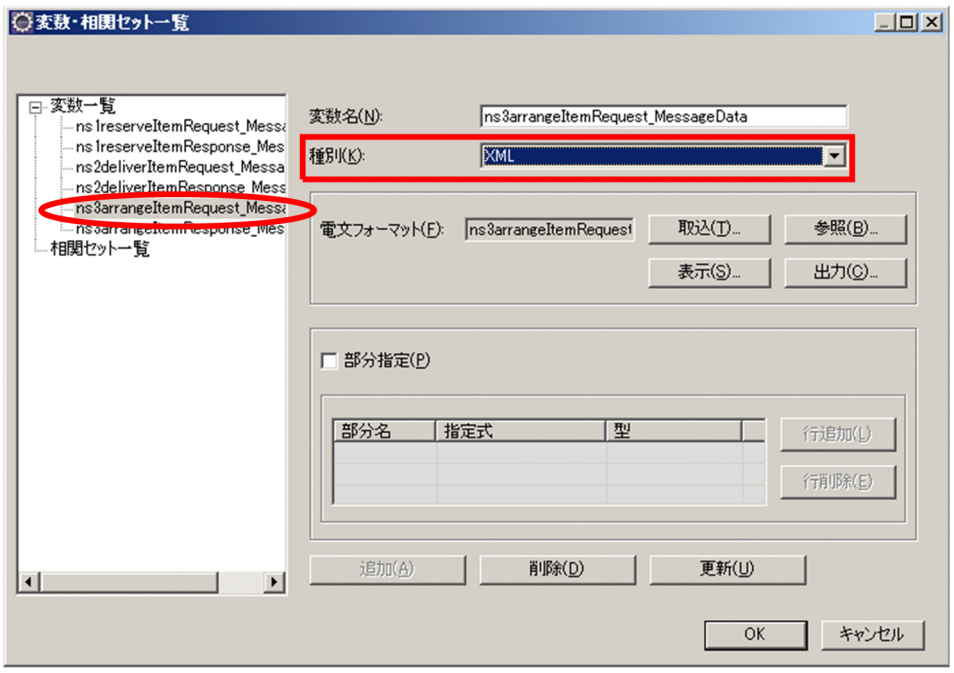
受付アクティビティ，応答アクティビティ，サービス呼出アクティビティで使用するメッセージ（変数）のスキーマ（構造）を登録します。

まず，受付アクティビティで使用するメッセージのスキーマを登録します。

1. ビジネスプロセス定義画面のキャンバス上の［変数・相関セット］アイコンをダブルクリックします。

［変数・相関セット一覧］ダイアログが表示されます。

2. 受付アクティビティが使用するメッセージ「ns3arrangeItemRequest_MessageData」※を選択し，［種別］を「XML」に変更します。



注※

Process Modeler は，WSDL で定義したメッセージ名の前に名前空間の修飾辞を付け，メッセージ名の後に「_MessageData」を付けて BPEL ファイルを生成します。例を次に示します。

（例）

arrangeItemRequest → ns3arrangeItemRequest_MessageData

arrangeItemResponse → ns3arrangeItemResponse_MessageData

この例にあるように，［変換・相関セット一覧］ダイアログでメッセージを選択するときは，メッセージ名が変更されていることに注意してください。なお，修飾辞の「ns3」の部分は「表 D-7」で設定したエクスポート時のネームスペースになるため，マニュアル上と実際に操作する画面で表示が異なる場合があります。

次に，受付アクティビティが使用するメッセージのスキーマ（電文フォーマット）をビジネスプロセスのユーザ定義受付から取り込みます。

3. ［電文フォーマット］横の［取込］ボタンをクリックします。

［電文フォーマットの取込］ダイアログが表示されます。

4. 次の画面および表に示すとおりに指定します。

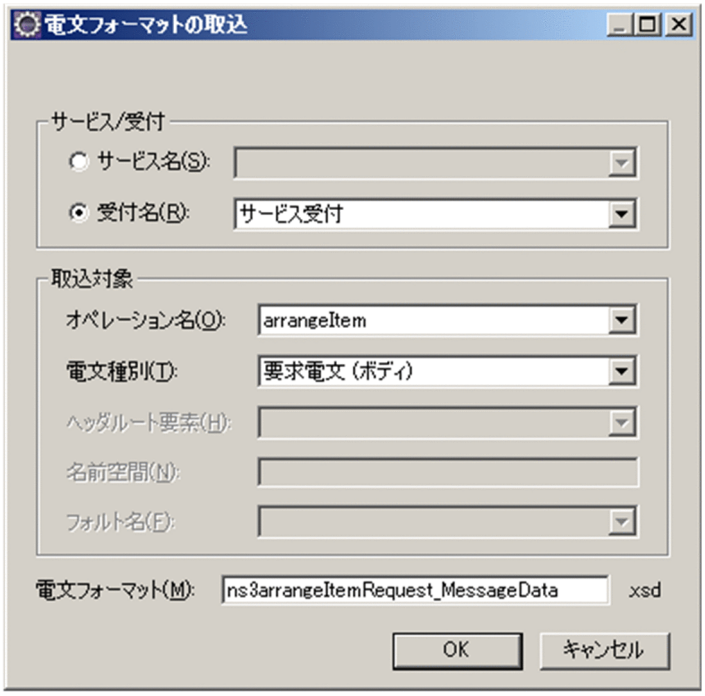
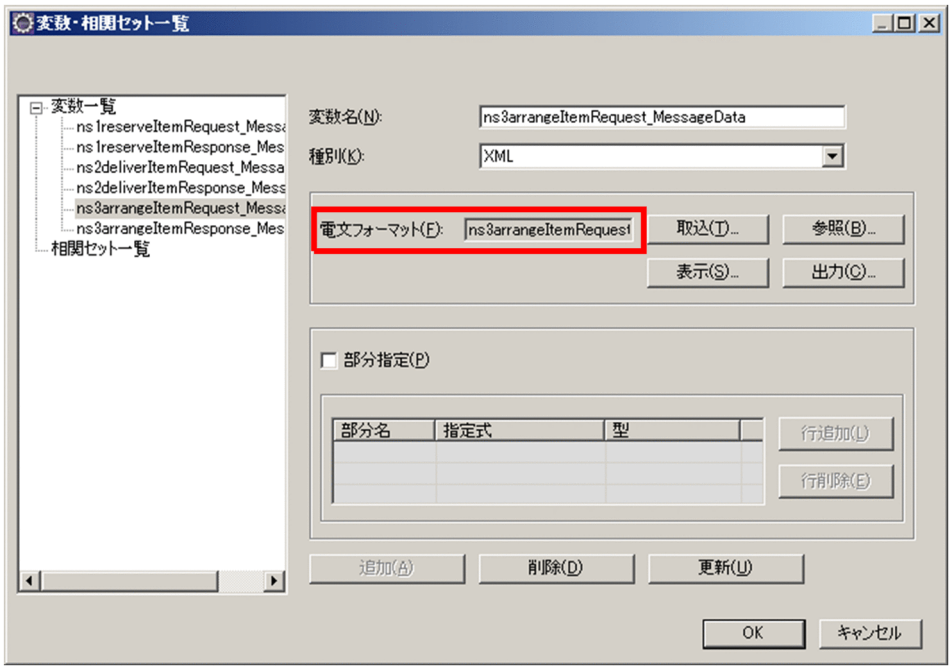


表 D–12　受付アクティビティの電文フォーマットの取り込み指定

項番	項目		指定値
1	サービス/受付		受付名：サービス受付
2	取込対象	オペレーション名	arrangeItem
3		電文種別	要求電文（ボディ）
4	電文フォーマット		ns3arrangeItemRequest_MessageData （自動設定されます）

5. [OK] ボタンをクリックします。

メッセージのスキーマの取り込みが終わると，［電文フォーマット］に電文フォーマット名が表示されます。



6. [更新] ボタンをクリックします。

これで，受付アクティビティで使用するメッセージのスキーマが登録されます。

次に示す電文についても，メッセージのスキーマを設定します。

- 「在庫引当」サービス呼出アクティビティの要求電文（ns1reserveItemRequest_MessageData）
- 「在庫引当」サービス呼出アクティビティの応答電文（ns1reserveItemResponse_MessageData）
- 「配送手配」サービス呼出アクティビティの要求電文（ns2deliverItemRequest_MessageData）
- 「配送手配」サービス呼出アクティビティの応答電文（ns2deliverItemResponse_MessageData）
- 応答アクティビティの応答電文（ns3arrangeItemResponse_MessageData）

各メッセージのスキーマの取り込み指定を次の各表に示します。すべてのスキーマの取り込みが終わったら，[変数・相関セット一覧] ダイアログで [OK] ボタンをクリックします。

表 D-13　ns1reserveItemRequest_MessageData の取り込みの指定

項番	項目		指定値
1	サービス/受付		受付名：InventoryManager
2	取込対象	オペレーション名	reserveItem
3		電文種別	要求電文
4	電文フォーマット		ns1reserveItemRequest_MessageData （自動設定されます）

表 D-14　ns1reserveItemResponse_MessageData の取り込みの指定

項番	項目		指定値
1	サービス/受付		サービス名：InventoryManager
2	取込対象	オペレーション名	reserveItem
3		電文種別	応答電文
4	電文フォーマット		ns1reserveItemResponse_MessageData （自動設定されます）

表 D-15　ns2deliverItemRequest_MessageData の取り込みの指定

項番	項目		指定値
1	サービス/受付		サービス名：Delivery
2	取込対象	オペレーション名	deliverItem
3		電文種別	要求電文
4	電文フォーマット		ns2deliverItemRequest_MessageData （自動設定されます）

表 D-16　ns2deliverItemResponse_MessageData の取り込みの指定

項番	項目		指定値
1	サービス/受付		サービス名：Delivery
2	取込対象	オペレーション名	deliverItem
3		電文種別	応答電文
4	電文フォーマット		ns2deliverItemResponse_MessageData （自動設定されます）

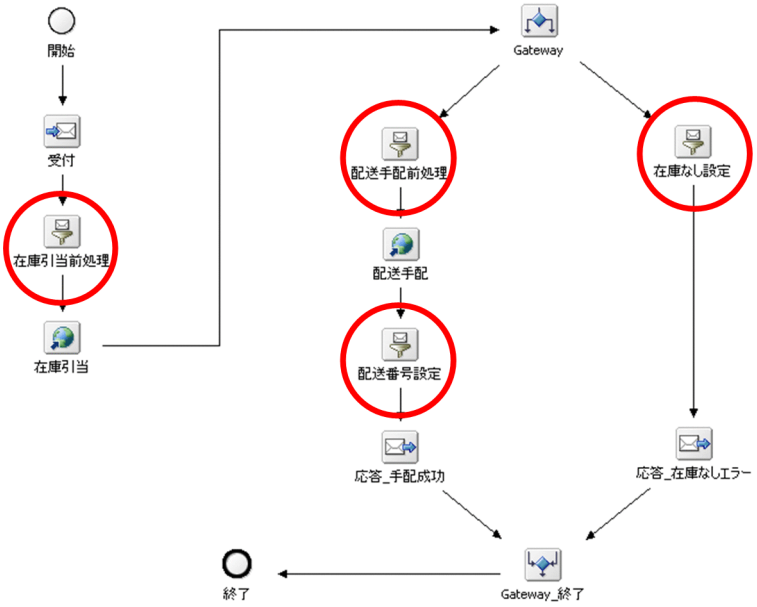
表 D-17　ns3arrangeItemResponse_MessageData の取り込みの指定

項番	項目		指定値
1	サービス/受付		受付名：サービス受付
2	取込対象	オペレーション名	arrangeItem
3		電文種別	応答電文
4	電文フォーマット		ns3arrangeItemResponse_MessageData （自動設定されます）

付録 D.10　メッセージ変換とシステム例外処理を追加する

必要に応じて，ビジネスプロセスにメッセージ変換（データ変換アクティビティ）や実行時のシステム例外に対処する処理を追加します。商品手配システムでは，次のようにメッセージ変換を追加しました。

図 D-6　メッセージ変換の追加



メッセージ変換の詳細な定義内容については，商品手配ビジネスプロセスのサンプルデータを参照してください。

以上で，上流設計ツールを利用したビジネスプロセスの開発は終了です。

付録 E　上流設計ツールとの連携で使用する BPEL のサポート範囲

サービスプラットフォームでは，BPEL1.1 と BPEL2.0 のインポートをサポートしています。BPEL1.1 と BPEL2.0 では，BPEL ファイルのサポート範囲およびビジネスプロセス定義への変換ルールが異なります。

ここでは，上流設計ツールで作成した BPEL ファイルをインポートする場合の，BPEL ファイルのサポート範囲およびビジネスプロセス定義への変換ルールを BPEL1.1 と BPEL2.0 に分けて説明します。

注意事項

BPEL ファイルの要素・属性のうち，以降の説明に登場しない要素・属性は，ビジネスプロセスの定義に変換されません。

BPEL ファイル内の要素が，ビジネスプロセスのどの定義内容に変換されるのかの関係を次の表に示します。

表 E-1　BPEL ファイルの要素とビジネスプロセス定義の関係

変換前の要素（BPEL ファイル）		変換後の定義内容（ビジネスプロセス）	変換方法の参照先	
			BPEL1.1	BPEL2.0
process 要素	variables 要素	変数	付録 E.1(1)(b)	付録 E.2(1)(b)
	correlationSets 要素	相関セット	付録 E.1(1)(c)	付録 E.2(1)(c)
	faultHandlers 要素	フォルト処理	付録 E.1(1)(d)	付録 E.2(1)(d)
	receive 要素	受付アクティビティ	付録 E.1(2)(a)	付録 E.2(2)(a)
	reply 要素	応答アクティビティ	付録 E.1(2)(b)	付録 E.2(2)(b)
	invoke 要素	サービス呼出アクティビティ	付録 E.1(2)(c)	付録 E.2(2)(c)
	assign 要素	代入アクティビティ	付録 E.1(2)(d)	付録 E.2(2)(d)
	empty 要素	無操作アクティビティ	付録 E.1(2)(e)	付録 E.2(2)(e)
	throw 要素	フォルト送出アクティビティ	付録 E.1(2)(f)	付録 E.2(2)(f)
	scope 要素	スコープアクティビティ	付録 E.1(3)(a)	付録 E.2(3)(a)
	while 要素	繰り返しアクティビティ	付録 E.1(3)(b)	付録 E.2(3)(b)
	switch 要素※1	分岐開始アクティビティ 分岐終了アクティビティ	付録 E.1(3)(c)	－
	if 要素※2		－	付録 E.2(3)(c)
	flow 要素	並列処理開始アクティビティ 並列処理終了アクティビティ	付録 E.1(3)(d)	付録 E.2(3)(d)
	sequence 要素	順次処理アクティビティ	付録 E.1(3)(e)	付録 E.2(3)(e)
	wait 要素	待機アクティビティ	付録 E.1(2)(g)	付録 E.2(2)(g)
	compensate 要素※3	無操作アクティビティ	－	－

変換前の要素（BPEL ファイル）		変換後の定義内容（ビジネスプロセス）	変換方法の参照先	
			BPEL1.1	BPEL2.0
process 要素	terminate 要素※1※3	無操作アクティビティ	－	－
	pick 要素※3			
	extensionActivity 要素※2※3			
	rethrow 要素※2※3			
	exit 要素※2※3			
	validate 要素※2※3			
	compensateScope 要素※2※3			
	forEach 要素※2※3			
	repeatUntil 要素※2※3			

（凡例）

－：参照先はありません。

注※1

BPEL1.1 の場合だけ定義できる要素です。

注※2

BPEL2.0 の場合だけ定義できる要素です。

注※3

これらの要素はサポート外のため，インポート時に無操作アクティビティに変換されます。

ポイント

- ビジネスプロセス内の各アクティビティを連結するコネクションの定義内容は，BPEL ファイルの構造，各要素の定義などから判断して設定されます。
- BPEL ファイルをインポートしてビジネスプロセスを作成した場合に，Java 呼出アクティビティおよびデータ変換アクティビティが定義されることはありません。

以降の項で，BPEL ファイルの各要素と，ビジネスプロセス定義の内容との関係の詳細を BPEL1.1 と BPEL2.0 に分けて説明します。

付録 E.1　BPEL1.1 のビジネスプロセス定義をインポートする場合

BPEL1.1 の場合の，BPEL ファイルのサポート範囲およびビジネスプロセス定義への変換ルールについて説明します。

サービスプラットフォームでは，BPEL1.1 の場合の BPEL ファイルに定義されている要素および属性で，一部サポートしていないものがあります。BPEL1.1 の場合の BPEL ファイルに定義されている要素および属性について，サービスプラットフォームでのサポート有無を次の表に示します。

表 E-2　BPEL ファイルに定義されている要素および属性のサポート有無（BPEL1.1 の場合）

分類	要素	下位要素または属性			サポート有無
ビジネスプロセス全体の定義に関する要素	process 要素	name 属性			×
		targetNamespace 属性			×
		queryLanguage 属性			×
		expressionLanguage 属性			×
		suppressJoinFailure 属性			×
		enableInstanceCompensation 属性			×
		abstractProcess 属性			×
		partnerLinks 要素			×
		partners 要素			×
		correlationSets 要素			×
		variables 要素	variable 要素	name 属性	○
				messageType 属性	○
				type 属性	○
				element 属性	×
		faultHandlers 要素	catch 要素	faultName 属性	×
				faultVariable 属性	○
				activity※1	○
			catchAll 要素	*activity*※1	○
		compensationHandler 要素			×
		eventHandlers 要素			×
		activity※1			○
基本アクティビティに関連する要素	receive 要素	name 属性			○
		joinCondition 属性			×
		suppressJoinFailure 属性			×
		partnerLink 属性			×

分類	要素	下位要素または属性		サポート有無
基本アクティビティに関連する要素	receive 要素	portType 属性		×
		operation 属性		○
		variable 属性		○
		createInstance 属性		△
		target 要素		×
		source 要素		×
		correlations 要素		×
	reply 要素	name 属性		○
		joinCondition 属性		×
		suppressJoinFailure 属性		×
		partnerLink 属性		×
		portType 属性		×
		operation 属性		○
		variable 属性		○
		faultName 属性		○
		target 要素		×
		source 要素		×
		correlations 要素		×
	invoke 要素	name 属性		○
		joinCondition 属性		×
		suppressJoinFailure 属性		×
		partnerLink 属性		×
		portType 属性		○
		operation 属性		○
		inputVariable 属性		○
		outputVariable 属性		○
		target 要素		×
		source 要素		×
		correlations 要素		×
		catch 要素	faultName 属性	×
			faultVariable 属性	○

分類	要素	下位要素または属性			サポート有無
基本アクティビティに関連する要素	invoke 要素	catch 要素	*activity*※1		○
		catchAll 要素	*activity*※1		○
		compensationHandler 要素			×
	assign 要素	name 属性			○
		joinCondition 属性			×
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
		copy 要素	from 要素	variable 属性	○
				expression 属性	○
				part 属性	×
				partnerLink 属性	×
				endpointReference 属性	×
				property 属性	×
				opaque 属性	×
				query 属性	×
				タグ値※2	○
			to 要素	variable 属性	○
				part 属性	×
				partnerLink 属性	×
				property 属性	×
				query 属性	×
	empty 要素	name 属性			○
		joinCondition 属性			×
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
	throw 要素	faultName 属性			×

分類	要素	下位要素または属性	サポート有無
基本アクティビティに関連する要素	throw 要素	faultVariable 属性	○
		name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
	wait 要素	name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		for 属性	○
		until 属性	○
	terminate 要素※3	name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
	compensate 要素※3	name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		scope 属性	×
		target 要素	×
		source 要素	×
構造アクティビティに関連する要素	scope 要素	name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		variableAccessSerializable 属性	×
		target 要素	×
		source 要素	×
		correlationSets 要素	×

分類	要素	下位要素または属性			サポート有無
構造アクティビティに関連する要素	scope 要素	variables 要素	variable 要素	name 属性	○
				messageType 属性	○
				type 属性	○
				element 属性	×
		faultHandlers 要素	catch 要素	faultName 属性	×
				faultVariable 属性	○
				activity※1	○
			catchAll 要素	*activity*※1	○
		compensationHandler 要素			×
		eventHandlers 要素			×
		activity※1			○
	while 要素	name 属性			○
		joinCondition 属性			×
		condition 属性			○
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
		activity※1			○
	switch 要素	name 属性			○
		joinCondition 属性			×
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
		case 要素	condition 属性		○
			activity※1		○
		otherwise 要素	*activity*※1		○
	flow 要素	name 属性			○
		joinCondition 属性			×

分類	要素	下位要素または属性	サポート有無
構造アクティビティに関連する要素	flow 要素	suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		links 要素	×
		activity※1	○
	sequence 要素	name 属性	×
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		activity※1	○
	pick 要素※3	name 属性	○
		joinCondition 属性	×
		suppressJoinFailure 属性	×
		createInstance 属性	×
		target 要素	×
		source 要素	×
		onAlarm 要素	×
		onMessage 要素	×

（凡例）

○：サポートしています。

△：制限付きでサポートしています。

×：サポートしていません。

注※1

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

注※2

from 要素のタグ値が変換元になります。

注※3

未サポートアクティビティのため，name 属性だけを反映した無操作アクティビティとして取り込まれます。

(1) ビジネスプロセス全体の定義に関連する要素の変換

BPEL ファイルに定義されている要素のうち，ビジネスプロセス全体に関連する内容（ビジネスプロセスそのものの設定，利用する変数など）の変換について次に説明します。

(a) process 要素の変換

process 要素およびその下位要素・属性は，ビジネスプロセス全体の定義に関連する内容に変換されます。

変換の詳細を次の表に示します。

表 E-3 process 要素の変換

BPEL ファイルの要素		ビジネスプロセスの定義内容	
		定義内容	説明
process 要素	variables 要素	変数	詳細は「(1)(b) variables 要素の変換」を参照してください。
	correlationSets 要素	相関セット	詳細は「(1)(c) correlationSets 要素の変換」を参照してください。
	faultHandlers 要素	フォルト処理	詳細は「(1)(d) faultHandlers 要素の変換」を参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
−		ビジネスプロセス名	ビジネスプロセス定義を追加するためのダイアログで指定したビジネスプロセス名が設定されます。
−		ビジネスプロセスのバージョン	「1」が設定されます。
−		永続化	ビジネスプロセス定義を追加するためのダイアログで指定した永続化の有無が設定されます。

（凡例）

−：該当する要素はありません。定義内容は，自動設定されるか，またはビジネスプロセス定義を追加するためのダイアログで指定した内容が設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

ビジネスプロセスの各アクティビティに変換される要素については，「(2) 基本アクティビティに関連する要素の変換」および「(3) 構造アクティビティに関連する要素の変換」を参照してください。

(b) variables 要素の変換

variables 要素およびその下位要素・属性は，ビジネスプロセス（またはスコープ内）に設定されている変数の定義に変換されます。インポートしたあとは，定義内容を変数・相関セット一覧ダイアログで変更できます。

注意事項

サービスプラットフォームでは，スコープ内のフォルト処理を構成するアクティビティの割当変数として定義できる変数は，フォルト処理のあるスコープの上位スコープで定義されている変数です。

そのため，BPEL ファイルの scope/faultHandlers 要素以下の要素で，変数として同じ scope 要素内の変数を利用している場合，インポートしたあとにビジネスプロセスで割当変数を再定義してください。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，変数・相関セット一覧ダイアログの項目名です。

表 E-4　variables 要素の変換

variables 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
variable 要素	name 属性	変数名	変数の名称が設定されます。
	messageType 属性	種別	この属性が定義されていた場合，type 属性は文字列型（string）に変換されます。 そのため，変換後に変数の種別をメッセージ型（XML）に変更し，電文フォーマットを登録する必要があります。
	type 属性	種別	次に示す変数の種別が設定されます。 • boolean の場合 「boolean」が設定されます。 • double で表現可能な型※の場合 「numeric」が設定されます。 • 上記以外または未定義の場合 「string」が設定されます。
	－	部分指定	変数の部分指定には，値は設定されません。部分指定を利用する場合，インポートしたあとに変数・相関セット一覧ダイアログで設定してください。

（凡例）

－：該当する要素はありません。

注※

次の型が該当します。

int，short，byte，unsignedInt，unsignedShort，unsignedByte，float，double

(c)　correlationSets 要素の変換

correlationSets 要素の定義内容は，ビジネスプロセスの定義では「相関セット」に当たる情報です。BPEL ファイルをインポートしてビジネスプロセスを作成しても値は設定されません。

ビジネスプロセスで相関セットを利用する場合，インポートしたあとに変数・相関セット一覧ダイアログで相関セットを定義し，相関セットを使用するアクティビティの割当相関セット群ダイアログで相関セットを割り当ててください。

(d)　faultHandlers 要素の変換

faultHandlers 要素およびその下位要素・属性は，ビジネスプロセス内のフォルト処理の定義内容に変換されます。変換後の定義内容は，フォルト処理の割当ダイアログで変更できます。

注意事項

- process 要素の直下に faultHandlers 要素が定義されている場合，最上位のビジネスプロセスにスコープが作成され，そのスコープのフォルト処理として定義されます。その場合，process 要素の直下のアクティビティは作成されたスコープ内に移動されます。
- scope 要素の配下に faultHandlers 要素が定義されている場合，上位のスコープ内にフォルト処理が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，フォルト処理の割当ダイアログの項目名です。

表 E-5　faultHandlers 要素の変換

faultHandlers 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
catch 要素	faultVariable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
catchAll 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	－	割当変数	常に「catch-all」が設定されます。
－		遷移先	catch 要素または catchAll 要素が定義されている場合，フォルト処理の遷移先が自動的に設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(2)　基本アクティビティに関連する要素の変換

BPEL ファイルに定義されている要素のうち，基本アクティビティに変換される要素の変換について次に説明します。

(a)　receive 要素の変換

receive 要素およびその下位要素・属性は，受付アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を受付アクティビティダイアログで変更できます。

注意事項

次の条件を満たす receive 要素が複数定義されている場合，2 つ目以降の receive 要素は無操作アクティビティに変換されます。

- portType 属性が同一である
- operation 属性が同一である

変換の詳細を次の表に示します。表内の「項目」に示す名称は，受付アクティビティダイアログの項目名です。

表 E-6　receive 要素の変換

receive 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
operation 属性	オペレーション名	portType 属性が異なり，かつ operation 属性が同一の receive 要素がすでに定義されている場合，オペレーション名の末尾に<n>（n は 1 以上の整数）が付与されます。
variable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
createInstance 属性	インスタンス生成	属性値に関係なく「yes」に変換されます。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
－	通信モデル	ビジネスプロセス定義の内容から判断して「同期」または「非同期」が設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

(b)　reply 要素の変換

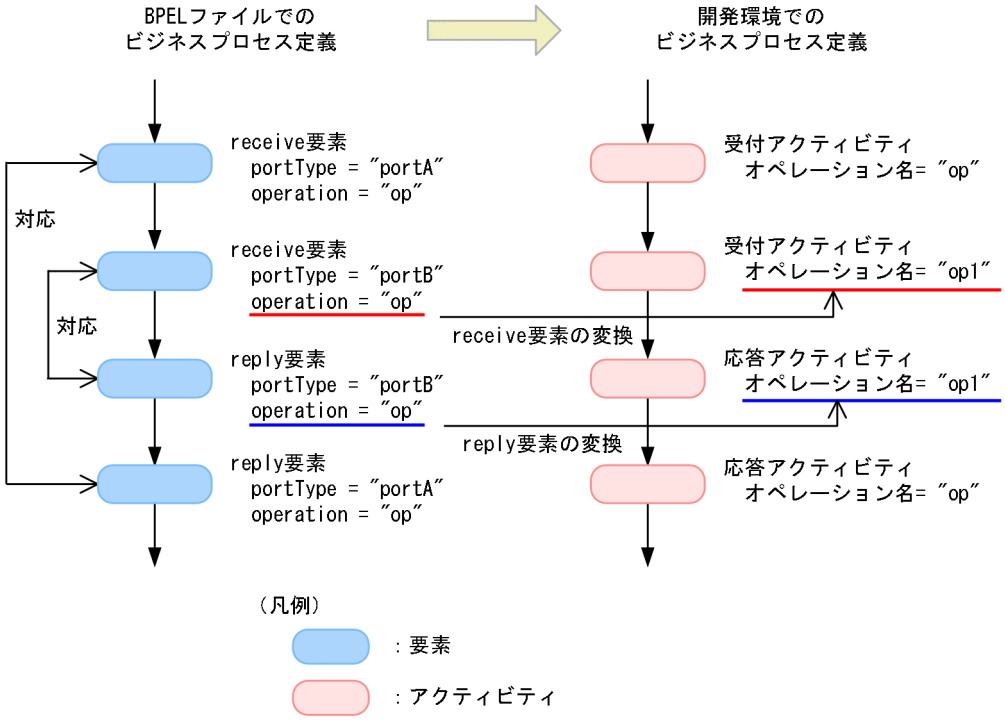
reply 要素およびその下位要素・属性は，応答アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を応答アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，応答アクティビティダイアログの項目名です。

表 E-7　reply 要素の変換

reply 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
operation 属性	オペレーション名	portType 属性と operation 属性が一致する receive 要素のオペレーション名の末尾に<n>（n は 1 以上の整数）が付与された場合（表 E-6 の operation 属性を参照），reply 要素のオペレーション名の末尾にも同じ<n>が付与されます。 portType 属性と operation 属性が一致する，receive 要素と reply 要素の変換について図 E-1 に示します。
variable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
faultName 属性	フォルト名	プリフィックスが付与されている場合は，ローカル名に変換されます。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

図 E-1　portType 属性と operation 属性が一致する receive 要素と reply 要素の変換



(c)　invoke 要素の変換

invoke 要素およびその下位要素・属性は，サービス呼出アクティビティの定義内容に変換されます。invoke 要素の下位要素として，catch 要素または catchAll 要素が定義されている場合，これらはサービス呼出アクティビティのフォルト処理（フォルトコネクションで連結したアクティビティ）に変換されます。イン

ポートしたあとは，サービス呼出アクティビティダイアログまたはフォルト処理の割当ダイアログから定義内容を変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，サービス呼出アクティビティダイアログまたはフォルト処理の割当ダイアログの項目名です。

表 E-8　invoke 要素の変換

invoke 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
portType 属性		サービス名	該当するサービス部品やオペレーションがリポジトリにある場合にだけ変換されます。ローカル名に変換されます。
operation 属性		オペレーション名	該当するサービス部品やオペレーションがリポジトリにある場合にだけ変換されます。
inputVariable 属性		要求電文用割当変数	サービス部品とオペレーションがリポジトリに存在する場合に設定されます。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
outputVariable 属性		応答電文用割当変数	サービス部品とオペレーションがリポジトリに存在し，かつ通信タイプが同期の場合に設定されます。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
name 属性		アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
catch 要素	faultVariable 属性	割当変数	フォルト処理の割当ダイアログの項目です。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
catchAll 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	－	割当変数	フォルト処理の割当ダイアログの項目です。 常に「catch-all」が設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(d)　assign 要素の変換

assign 要素およびその下位要素・属性は，代入アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を代入アクティビティダイアログまたは代入アクティビティサブダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，代入アクティビティダイアログまたは代入アクティビティサブダイアログの項目名です。

表 E-9　assign 要素の変換

assign 要素の下位要素または属性			ビジネスプロセスの定義内容	
			項目	説明
name 属性			アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
copy 要素	from 要素	variable 属性	名前（コピー元の変数）	代入アクティビティサブダイアログの項目です。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
		expression 属性	値（コピー元の式）	代入アクティビティサブダイアログの項目です。 variable 属性が定義されている場合は変換されません。
		タグ値※	値（コピー元の式）	代入アクティビティサブダイアログの項目です。 variable 属性および expression 属性が定義されていない場合，タグ値が式の値として設定されます。
	to 要素	variable 属性	変数名（コピー先）	代入アクティビティサブダイアログの項目です。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。

注※

from 要素のタグ値が変換元になります。

(e)　empty 要素の変換

empty 要素は，無操作アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を無操作アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，無操作アクティビティダイアログの項目名です。

表 E-10　empty 要素の変換

empty 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

(f)　throw 要素の変換

throw 要素およびその下位要素・属性は，フォルト送出アクティビティの定義内容に変換されます。インポートしたあとは，定義内容をフォルト送出アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，フォルト送出アクティビティダイアログの項目名です。

表 E-11　throw 要素の変換

throw 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
faultVariable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

(g)　wait 要素の変換

wait 要素は，待機アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を待機アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，待機アクティビティダイアログの項目名です。

表 E-12　wait 要素の変換

wait 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場

wait 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
for 属性※	式	待機する間隔の情報です。［間隔］が ON になります。
until 属性※	式	待機する期限の情報です。［期限］が ON になります。

注※

for 属性と until 属性の両方が定義されていた場合，for 属性が取り込まれ，until 属性が無視されます。また，for 属性と until 属性の両方が定義されていない場合，［間隔］が ON になります。

(3)　構造アクティビティに関連する要素の変換

BPEL ファイルに定義されている要素のうち，構造アクティビティに変換される要素の変換について次に説明します。

(a)　scope 要素の変換

scope 要素およびその下位要素・属性は，スコープアクティビティの定義内容に変換されます。インポートしたあとは，定義内容をスコープアクティビティダイアログで変更できます。

また，scope 要素内に定義されている変数，相関セット，フォルト処理，トランザクションの制御の情報は，変数・相関セット一覧ダイアログおよびフォルト処理の割当ダイアログで変更できます。

変換の詳細を次の表に示します。

表 E-13　scope 要素の変換

scope 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
−	トランザクションの制御	トランザクション制御を実行するかが設定されます。デフォルトでは，［アクティビティ単位でコミットする］が ON になります。
variables 要素	変数	スコープ内で利用する変数の情報です。 変数については「(1)(b)　variables 要素の変換」を参照してください。
correlationSets 要素	相関セット	スコープ内で利用する相関セットの情報です。 相関セットの情報については「(1)(c)　correlationSets 要素の変換」を参照してください。

scope 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
faultHandlers 要素	フォルト処理	スコープ内のフォルト処理の情報です。 フォルト処理の情報については「(1)(d)　faultHandlers 要素の変換」を参照してください。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(b)　while 要素の変換

while 要素およびその下位要素・属性は，繰り返しアクティビティの定義内容に変換されます。インポートしたあとは，定義内容を繰り返しアクティビティダイアログおよび条件設定ダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，繰り返しアクティビティダイアログまたは条件設定ダイアログの項目名です。

表 E-14　while 要素の変換

while 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
condition 属性	条件式	条件設定ダイアログの条件式に値が設定されます。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
－	最大繰り返し回数	「100」が自動的に設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(c)　switch 要素の変換

switch 要素およびその下位要素・属性は，分岐開始アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を分岐アクティビティダイアログおよび条件設定ダイアログで変更できます。

注意事項

分岐終了アクティビティは，switch 要素以下が分岐開始アクティビティに変換されたときに自動的に設定されます。その際，分岐終了アクティビティの名称として，分岐開始アクティビティの名称に「_終了」を付けた名称が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，分岐アクティビティダイアログまたは条件設定ダイアログの項目名です。

表 E-15　switch 要素の変換

switch 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
name 属性		アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
case 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	condition 属性	条件式	条件設定ダイアログの条件式に値が設定されます。
	－	条件名	分岐アクティビティダイアログの項目です。 「condition<n>」が自動的に設定されます。<n>は分岐内でユニークになる整数が設定されます。
	－	優先順位	分岐アクティビティダイアログの項目です。 switch 要素内の条件の出現順に設定されます。
otherwise 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	－	優先順位	分岐アクティビティダイアログの項目です。 default に設定されます。
－		遷移先	分岐アクティビティダイアログの項目です。 case 要素および otherwise 要素の定義から判断して設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(d)　flow 要素の変換

flow 要素およびその下位要素・属性は，並列処理開始アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を並列処理アクティビティダイアログで変更できます。

注意事項

- リンクの定義（BPEL ファイル内の links 要素以下の定義）は変換されません。
- 並列処理終了アクティビティは，flow 要素以下が並列処理開始アクティビティに変換されたときに自動的に設定されます。その際，並列処理終了アクティビティの名称として，並列処理開始アクティビティの名称に「_終了」を付けた名称が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，並列処理アクティビティダイアログの項目名です。

表 E-16　flow 要素の変換

flow 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(e)　sequence 要素の変換

sequence 要素は，順次処理アクティビティの定義内容に変換されます。定義内容は変更できません。

なお，sequence 要素には，変換される下位属性はありません。

サービスプラットフォームでは，「5.6.20　順次処理アクティビティ」に相当する位置に記述された sequence 要素だけが，ビジネスプロセスの定義に変換されます。

付録 E.2　BPEL2.0 のビジネスプロセス定義をインポートする場合

BPEL2.0 の場合の，BPEL ファイルのサポート範囲およびビジネスプロセス定義への変換ルールについて説明します。

サービスプラットフォームでは，BPEL2.0 の場合の BPEL ファイルに定義されている要素および属性で，一部サポートしていないものがあります。BPEL2.0 の場合の BPEL ファイルに定義されている要素および属性について，サービスプラットフォームでのサポート有無を次の表に示します。

表 E-17　BPEL ファイルに定義されている要素および属性のサポート有無（BPEL2.0 の場合）

分類	要素	下位要素または属性			サポート有無
ビジネスプロセス全体の定義に関する要素	process 要素	name 属性			×
		targetNamespace 属性			×
		queryLanguage 属性			×
		expressionLanguage 属性			×
		suppressJoinFailure 属性			×
		exitOnStandardFault 属性			×
		partnerLinks 要素			×
		correlationSets 要素			×
		variables 要素	documentation 要素		×
			variable 要素	name 属性	○
				messageType 属性	○
				type 属性	○
				element 属性	×
				documentation 要素	×
				from 要素	×
		faultHandlers 要素	documentation 要素		×
			catch 要素	faultName 属性	×
				faultVariable 属性	○
				faultMessageType 属性	×
				faultElement 属性	×
				documentation 要素	×
				activity※1	○
			catchAll 要素	documentation 要素	×
				activity※1	○
		eventHandlers 要素			×
		import 要素			×
		messageExchanges 要素			×
		documentation 要素			×
		activity※1			○

分類	要素	下位要素または属性	サポート有無
基本アクティビティに関連する要素	receive 要素	name 属性	○
		suppressJoinFailure 属性	×
		partnerLink 属性	×
		portType 属性	×
		operation 属性	○
		variable 属性	○
		createInstance 属性	△
		messageExchange 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		correlations 要素	×
		fromParts 要素	×
	reply 要素	name 属性	○
		suppressJoinFailure 属性	×
		partnerLink 属性	×
		portType 属性	×
		operation 属性	○
		variable 属性	○
		faultName 属性	○
		messageExchange 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		correlations 要素	×
		toParts 要素	×
	invoke 要素	name 属性	○
		suppressJoinFailure 属性	×
		partnerLink 属性	×
		portType 属性	○

分類	要素	下位要素または属性			サポート有無
基本アクティビティに関連する要素	invoke 要素	operation 属性			○
		inputVariable 属性			○
		outputVariable 属性			○
		target 要素			×
		source 要素			×
		documentation 要素			×
		correlations 要素			×
		toParts 要素			×
		fromParts 要素			×
		catch 要素	faultName 属性		×
			faultVariable 属性		○
			faultMessageType 属性		×
			faultElement 属性		×
			documentation 要素		×
			activity※1		○
		catchAll 要素	documentation 要素		×
			activity※1		○
		compensationHandler 要素			×
	assign 要素	name 属性			○
		suppressJoinFailure 属性			×
		validate 属性			×
		target 要素			×
		source 要素			×
		documentation 要素			×
		copy 要素	keepSrcElementName 属性		×
			ignoreMissingFromData 属性		×
			documentation 要素		×
			from 要素	variable 属性	○
				expressionLanguage 属性	×
				part 属性	×

分類	要素	下位要素または属性				サポート有無
基本アクティビティに関連する要素	assign 要素	copy 要素	from 要素	partnerLink 属性		×
				endpointReference 属性		×
				property 属性		×
				documentation 要素		×
				literal 要素		○
				query 要素	queryLanguage 属性	×
			to 要素	variable 属性		○
				part 属性		×
				partnerLink 属性		×
				expressionLanguage 属性		×
				property 属性		×
				documentation 要素		×
				query 要素	queryLanguage 属性	×
	empty 要素	name 属性				○
		suppressJoinFailure 属性				×
		target 要素				×
		source 要素				×
		documentation 要素				×
	throw 要素	name 属性				○
		suppressJoinFailure 属性				×
		faultName 属性				×
		faultVariable 属性				○
		target 要素				×
		source 要素				×
		documentation 要素				×
	wait 要素	name 属性				○
		suppressJoinFailure 属性				×
		target 要素				×
		source 要素				×

分類	要素	下位要素または属性		サポート有無
基本アクティビティに関連する要素	wait 要素	documentation 要素		×
		for 要素		○
		└	expressionLanguage 属性	×
		until 要素		○
		└	expressionLanguage 属性	×
	compensate 要素※2	name 属性		○
		suppressJoinFailure 属性		×
		target 要素		×
		source 要素		×
		documentation 要素		×
	extensionActivity 要素※2	*anyElementQName*※3	name 属性	○
			suppressJoinFailure 属性	×
			anyAttribute※3	×
			target 要素	×
			source 要素	×
			documentation 要素	×
			any※3	×
	rethrow 要素※2	name 属性		○
		suppressJoinFailure 属性		×
		target 要素		×
		source 要素		×
		documentation 要素		×
	exit 要素※2	name 属性		○
		suppressJoinFailure 属性		×
		target 要素		×
		source 要素		×
		documentation 要素		×
	validate 要素※2	name 属性		○
		suppressJoinFailure 属性		×
		variables 属性		×

分類	要素	下位要素または属性			サポート有無
基本アクティビティに関連する要素	validate 要素※2	target 要素			×
		source 要素			×
		documentation 要素			×
	compensateScope 要素※2	name 属性			○
		suppressJoinFailure 属性			×
		target 属性			×
		target 要素			×
		source 要素			×
		documentation 要素			×
構造アクティビティに関連する要素	scope 要素	name 属性			○
		suppressJoinFailure 属性			×
		exitOnStandardFault 属性			×
		isolated 属性			×
		target 要素			×
		source 要素			×
		documentation 要素			×
		correlationSets 要素			×
		variables 要素	documentation 要素		×
			variable 要素	name 属性	○
				messageType 属性	○
				type 属性	○
				element 属性	×
				documentation 要素	×
				from 要素	×
		faultHandlers 要素	documentation 要素		×
			catch 要素	faultName 属性	×
				faultVariable 属性	○
				faultMessageType 属性	×
				faultElement 属性	×
				documentation 要素	×

分類	要素	下位要素または属性			サポート有無
構造アクティビティに関連する要素	scope 要素	faultHandlers 要素	catch 要素	*activity*※1	○
			catchAll 要素	documentation 要素	×
				activity※1	○
		compensationHandler 要素			×
		eventHandlers 要素			×
		terminationHandler 要素			×
		partnerLinks 要素			×
		messageExchanges 要素			×
		activity※1			○
	while 要素	name 属性			○
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
		documentation 要素			×
		condition 要素			○
		└		expressionLanguage 属性	×
		activity※1			○
	if 要素	name 属性			○
		suppressJoinFailure 属性			×
		target 要素			×
		source 要素			×
		documentation 要素			×
		condition 要素			○
		└		expressionLanguage 属性	×
		elseif 要素		documentation 要素	×
				condition 要素	○
				activity※1	○
		else 要素		documentation 要素	×
				activity※1	○
				activity※1	○

分類	要素	下位要素または属性	サポート有無
構造アクティビティに関連する要素	flow 要素	name 属性	○
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		links 要素	×
		activity※1	○
	sequence 要素	name 属性	○
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		activity※1	○
	pick 要素※2	name 属性	○
		suppressJoinFailure 属性	×
		createInstance 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		onAlarm 要素	×
		onMessage 要素	×
	forEach 要素※2	name 属性	○
		suppressJoinFailure 属性	×
		counterName 属性	×
		parallel 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		startCounterValue 要素	×
		finalCounterValue 要素	×

分類	要素	下位要素または属性	サポート有無
構造アクティビティに関連する要素	forEach 要素※2	completionCondition 要素	×
		scope 要素	×
	repeatUntil 要素※2	name 属性	○
		suppressJoinFailure 属性	×
		target 要素	×
		source 要素	×
		documentation 要素	×
		condition 要素	×

（凡例）

○：サポートしています。

△：制限付きでサポートしています。

×：サポートしていません。

注※1

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

注※2

未サポートアクティビティのため，name 属性だけを反映した無操作アクティビティとして取り込まれます。

注※3

anyElementQName の部分には，実際には拡張アクティビティを表す要素が入ります。*anyAttribute* は拡張アクティビティ固有の属性，*any* は拡張アクティビティ固有の要素を示します。

(1)　ビジネスプロセス全体の定義に関連する要素の変換

BPEL ファイルに定義されている要素のうち，ビジネスプロセス全体に関連する内容（ビジネスプロセスそのものの設定，利用する変数など）の変換について次に説明します。

(a)　process 要素の変換

process 要素およびその下位要素・属性は，ビジネスプロセス全体の定義に関連する内容に変換されます。

変換の詳細を次の表に示します。

表 E-18　process 要素の変換

BPEL ファイルの要素		ビジネスプロセスの定義内容	
		定義内容	説明
process 要素	variables 要素	変数	詳細は「(1)(b)　variables 要素の変換」を参照してください。
	correlationSets 要素	相関セット	詳細は「(1)(c)　correlationSets 要素の変換」を参照してください。

BPEL ファイルの要素		ビジネスプロセスの定義内容	
		定義内容	説明
process 要素	faultHandlers 要素	フォルト処理	詳細は「(1)(d)　faultHandlers 要素の変換」を参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
－		ビジネスプロセス名	ビジネスプロセス定義を追加するためのダイアログで指定したビジネスプロセス名が設定されます。
－		ビジネスプロセスのバージョン	「1」が設定されます。
－		永続化	ビジネスプロセス定義を追加するためのダイアログで指定した永続化の有無が設定されます。

（凡例）

－：該当する要素はありません。定義内容は，自動設定されるか，またはビジネスプロセス定義を追加するためのダイアログで指定した内容が設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

ビジネスプロセスの各アクティビティに変換される要素については，「(2)　基本アクティビティに関連する要素の変換」および「(3)　構造アクティビティに関連する要素の変換」を参照してください。

(b)　variables 要素の変換

variables 要素およびその下位要素・属性は，ビジネスプロセス（またはスコープ内）に設定されている変数の定義に変換されます。インポートしたあとは，定義内容を変数・相関セット一覧ダイアログで変更できます。

注意事項

サービスプラットフォームでは，スコープ内のフォルト処理を構成するアクティビティの割当変数として定義できる変数は，フォルト処理のあるスコープの上位スコープで定義されている変数です。

そのため，BPEL ファイルの scope/faultHandlers 要素以下の要素で，変数として同じ scope 要素内の変数を利用している場合，インポートしたあとにビジネスプロセスで割当変数を再定義してください。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，変数・相関セット一覧ダイアログの項目名です。

表 E-19　variables 要素の変換

variables 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
variable 要素	name 属性	変数名	変数の名称が設定されます。
	messageType 属性	種別	この属性が定義されていた場合，type 属性は文字列型（string）に変換されます。 そのため，変換後に変数の種別をメッセージ型（XML）に変更し，電文フォーマットを登録する必要があります。
	type 属性	種別	次に示す変数の種別が設定されます。 • boolean の場合 「boolean」が設定されます。 • double で表現可能な型※の場合 「numeric」が設定されます。 • 上記以外または未定義の場合 「string」が設定されます。
	－	部分指定	変数の部分指定には，値は設定されません。部分指定を利用する場合，インポートしたあとに変数・相関セット一覧ダイアログで設定してください。

（凡例）

－：該当する要素はありません。

注※

次の型が該当します。

int，short，byte，unsignedInt，unsignedShort，unsignedByte，float，double

(c)　correlationSets 要素の変換

correlationSets 要素の定義内容は，ビジネスプロセスの定義では「相関セット」に当たる情報です。BPEL ファイルをインポートしてビジネスプロセスを作成しても値は設定されません。

ビジネスプロセスで相関セットを利用する場合，インポートしたあとに変数・相関セット一覧ダイアログで相関セットを定義し，相関セットを使用するアクティビティの割当相関セット群ダイアログで相関セットを割り当ててください。

(d)　faultHandlers 要素の変換

faultHandlers 要素およびその下位要素・属性は，ビジネスプロセス内のフォルト処理の定義内容に変換されます。変換後の定義内容は，フォルト処理の割当ダイアログで変更できます。

注意事項

- process 要素の直下に faultHandlers 要素が定義されている場合，最上位のビジネスプロセスにスコープが作成され，そのスコープのフォルト処理として定義されます。その場合，process 要素の直下のアクティビティは作成されたスコープ内に移動されます。
- scope 要素の配下に faultHandlers 要素が定義されている場合，上位のスコープ内にフォルト処理が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，フォルト処理の割当ダイアログの項目名です。

表 E-20　faultHandlers 要素の変換

faultHandlers 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
catch 要素	faultVariable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
catchAll 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	－	割当変数	常に「catch-all」が設定されます。
－		遷移先	catch 要素または catchAll 要素が定義されている場合，フォルト処理の遷移先が自動的に設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(2)　基本アクティビティに関連する要素の変換

BPEL ファイルに定義されている要素のうち，基本アクティビティに変換される要素の変換について次に説明します。

(a)　receive 要素の変換

receive 要素およびその下位要素・属性は，受付アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を受付アクティビティダイアログで変更できます。

注意事項

次の条件を満たす receive 要素が複数定義されている場合，2 つ目以降の receive 要素は無操作アクティビティに変換されます。

- portType 属性が同一である
- operation 属性が同一である

変換の詳細を次の表に示します。表内の「項目」に示す名称は，受付アクティビティダイアログの項目名です。

表 E-21　receive 要素の変換

receive 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
operation 属性	オペレーション名	portType 属性が異なり，かつ operation 属性が同一の receive 要素がすでに定義されている場合，オペレーション名の末尾に<n>（n は 1 以上の整数）が付与されます。
variable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
createInstance 属性	インスタンス生成	属性値に関係なく「yes」に変換されます。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
－	通信モデル	ビジネスプロセス定義の内容から判断して「同期」または「非同期」が設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

(b)　reply 要素の変換

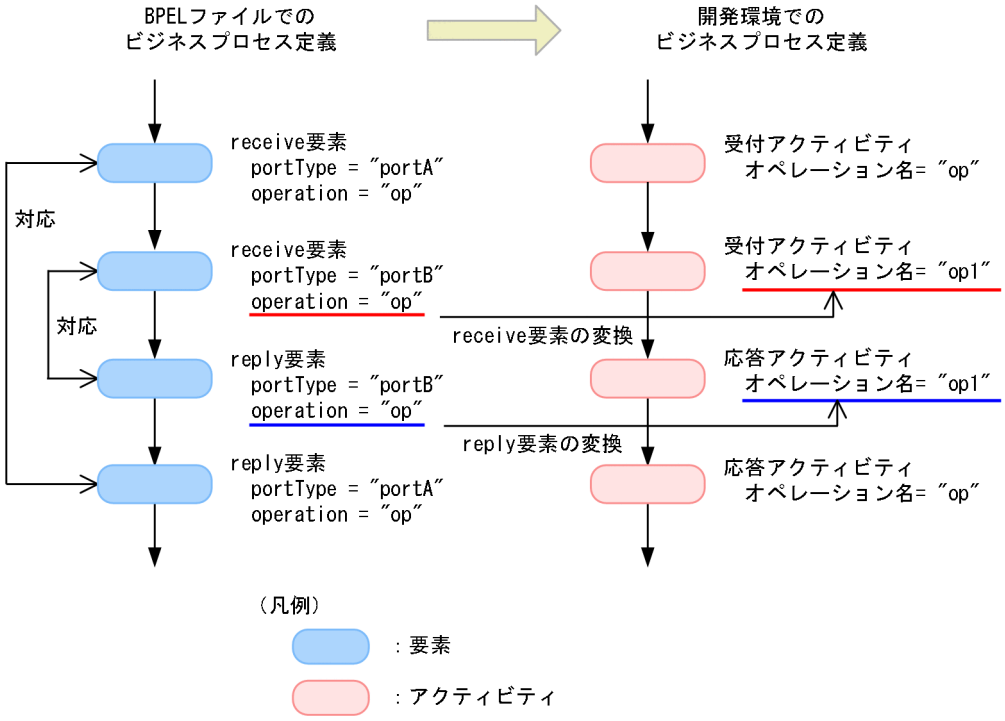
reply 要素およびその下位要素・属性は，応答アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を応答アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，応答アクティビティダイアログの項目名です。

表 E-22　reply 要素の変換

reply 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
operation 属性	オペレーション名	portType 属性と operation 属性が一致する receive 要素のオペレーション名の末尾に<n>（n は 1 以上の整数）が付与された場合（表 E-21 の operation 属性を参照），reply 要素のオペレーション名の末尾にも同じ<n>が付与されます。 portType 属性と operation 属性が一致する，receive 要素と reply 要素の変換について図 E-2 に示します。
variable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
faultName 属性	フォルト名	プリフィックスが付与されている場合は，ローカル名に変換されます。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

図 E-2　portType 属性と operation 属性が一致する receive 要素と reply 要素の変換



(c)　invoke 要素の変換

invoke 要素およびその下位要素・属性は，サービス呼出アクティビティの定義内容に変換されます。invoke 要素の下位要素として，catch 要素または catchAll 要素が定義されている場合，これらはサービス呼出アクティビティのフォルト処理（フォルトコネクションで連結したアクティビティ）に変換されます。イン

ポートしたあとは，サービス呼出アクティビティダイアログまたはフォルト処理の割当ダイアログから定義内容を変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，サービス呼出アクティビティダイアログまたはフォルト処理の割当ダイアログの項目名です。

表 E-23　invoke 要素の変換

invoke 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
portType 属性		サービス名	該当するサービス部品やオペレーションがリポジトリにある場合にだけ変換されます。ローカル名に変換されます。
operation 属性		オペレーション名	該当するサービス部品やオペレーションがリポジトリにある場合にだけ変換されます。
inputVariable 属性		要求電文用割当変数	サービス部品とオペレーションがリポジトリに存在する場合に設定されます。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
outputVariable 属性		応答電文用割当変数	サービス部品とオペレーションがリポジトリに存在し，かつ通信タイプが同期の場合に設定されます。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
name 属性		アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
catch 要素	faultVariable 属性	割当変数	フォルト処理の割当ダイアログの項目です。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
catchAll 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	－	割当変数	フォルト処理の割当ダイアログの項目です。 常に「catch-all」が設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(d)　assign 要素の変換

assign 要素およびその下位要素・属性は，代入アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を代入アクティビティダイアログまたは代入アクティビティサブダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，代入アクティビティダイアログまたは代入アクティビティサブダイアログの項目名です。

表 E-24　assign 要素の変換

assign 要素の下位要素または属性			ビジネスプロセスの定義内容	
			項目	説明
name 属性			アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
copy 要素	from 要素	variable 属性	名前（コピー元の変数）	代入アクティビティサブダイアログの項目です。 代入アクティビティサブダイアログに設定される値は，variable 属性，literal 要素，および query 要素の指定によって異なります。詳細は，表 E-25 を参照してください。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
		literal 要素	値（コピー元の式）	代入アクティビティサブダイアログの項目です。 代入アクティビティサブダイアログに設定される値は，variable 属性，literal 要素，および query 要素の指定によって異なります。詳細は，表 E-25 を参照してください。
		query 要素	値（コピー元の式）	代入アクティビティサブダイアログの項目です。 代入アクティビティサブダイアログに設定される値は，variable 属性，literal 要素，および query 要素の指定によって異なります。詳細は，表 E-25 を参照してください。
	to 要素	variable 属性	変数名（コピー先）	代入アクティビティサブダイアログの項目です。 変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。

variable 属性，literal 要素，および query 要素の指定によって，代入アクティビティサブダイアログに設定される値を次の表に示します。

表 E-25　代入アクティビティサブダイアログに設定される値

項番	assign 要素下の from 要素の属性			代入アクティビティサブダイアログに設定される値
	variable 属性	literal 要素	query 要素	
1	×	×	×	設定されません。
2	○	×	×	［コピー元］の［変数］の［名前］に，variable 属性で指定した値が設定されます。
3	×	○	×	［コピー元］の［式］の［値］に，literal 要素で指定した値が設定されます。
4	×	×	○	［コピー元］の［式］の［値］に，query 要素で指定した値が設定されます。
5	○	○	×	［コピー元］の［変数］の［名前］に，variable 属性で指定した値が設定されます。
6	○	×	○	［コピー元］の［式］の［値］に，query 要素で指定した値が設定されます。
7	×	○	○	assign 要素は変換されません（エラーが発生します）。
8	○	○	○	assign 要素は変換されません（エラーが発生します）。

（凡例）

○：値を指定していることを示します。

×：値を指定していないことを示します。

(e)　empty 要素の変換

empty 要素は，無操作アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を無操作アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，無操作アクティビティダイアログの項目名です。

表 E-26　empty 要素の変換

empty 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

(f)　throw 要素の変換

throw 要素およびその下位要素・属性は，フォルト送出アクティビティの定義内容に変換されます。インポートしたあとは，定義内容をフォルト送出アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，フォルト送出アクティビティダイアログの項目名です。

表 E-27　throw 要素の変換

throw 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
faultVariable 属性	割当変数	変数については「(1)(b)　variables 要素の変換」の注意事項も参照してください。
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。

(g)　wait 要素の変換

wait 要素は，待機アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を待機アクティビティダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，待機アクティビティダイアログの項目名です。

表 E-28　wait 要素の変換

wait 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
for 要素※	式	待機する間隔の情報です。［間隔］が ON になります。
until 要素※	式	待機する期限の情報です。［期限］が ON になります。

注※

for 要素と until 要素の両方が定義されていた場合，for 要素が取り込まれ，until 要素が無視されます。また，for 要素と until 要素の両方が定義されていない場合，［間隔］が ON になります。

(3)　構造アクティビティに関連する要素の変換

BPEL ファイルに定義されている要素のうち，構造アクティビティに変換される要素の変換について次に説明します。

(a) scope 要素の変換

scope 要素およびその下位要素・属性は，スコープアクティビティの定義内容に変換されます。インポートしたあとは，定義内容をスコープアクティビティダイアログで変更できます。

また，scope 要素内に定義されている変数，相関セット，フォルト処理，トランザクションの制御の情報は，変数・相関セット一覧ダイアログおよびフォルト処理の割当ダイアログで変更できます。

変換の詳細を次の表に示します。

表 E-29 scope 要素の変換

scope 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
－	トランザクションの制御	トランザクション制御を実行するかが設定されます。デフォルトでは，［アクティビティ単位でコミットする］が ON になります。
variables 要素	変数	スコープ内で利用する変数の情報です。 変数については「(1)(b) variables 要素の変換」を参照してください。
correlationSets 要素	相関セット	スコープ内で利用する相関セットの情報です。 相関セットの情報については「(1)(c) correlationSets 要素の変換」を参照してください。
faultHandlers 要素	フォルト処理	スコープ内のフォルト処理の情報です。 フォルト処理の情報については「(1)(d) faultHandlers 要素の変換」を参照してください。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(b) while 要素の変換

while 要素およびその下位要素・属性は，繰り返しアクティビティの定義内容に変換されます。インポートしたあとは，定義内容を繰り返しアクティビティダイアログおよび条件設定ダイアログで変更できます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，繰り返しアクティビティダイアログまたは条件設定ダイアログの項目名です。

表 E-30　while 要素の変換

while 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
condition 要素	条件式	条件設定ダイアログの条件式に値が設定されます。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
－	最大繰り返し回数	「100」が自動的に設定されます。

（凡例）

－：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(c)　if 要素の変換

if 要素およびその下位要素・属性は，分岐開始アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を分岐アクティビティダイアログおよび条件設定ダイアログで変更できます。

注意事項

分岐終了アクティビティは，if 要素以下が分岐開始アクティビティに変換されたときに自動的に設定されます。その際，分岐終了アクティビティの名称として，分岐開始アクティビティの名称に「_終了」を付けた名称が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，分岐アクティビティダイアログまたは条件設定ダイアログの項目名です。

表 E-31　if 要素の変換

if 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されている場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
condition 要素	条件式	条件設定ダイアログの条件式に値が設定されます。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
－	優先順位	分岐アクティビティダイアログの項目です。

if 要素の下位要素または属性		ビジネスプロセスの定義内容	
		項目	説明
−		優先順位	常に「1」が設定されます。
elseif 要素	condition 要素	条件式	条件設定ダイアログの条件式に値が設定されます。
	activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	−	条件名	分岐アクティビティダイアログの項目です。 「condition<n>」が自動的に設定されます。<n>は分岐内でユニークになる整数が設定されます。
	−	優先順位	分岐アクティビティダイアログの項目です。 出現順に「2」から順に設定されます。
else 要素	*activity*※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。
	−	優先順位	分岐アクティビティダイアログの項目です。 default に設定されます。

（凡例）

−：該当する要素はありません。定義内容は自動で設定されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(d)　flow 要素の変換

flow 要素およびその下位要素・属性は，並列処理開始アクティビティの定義内容に変換されます。インポートしたあとは，定義内容を並列処理アクティビティダイアログで変更できます。

注意事項

- リンクの定義（BPEL ファイル内の links 要素以下の定義）は変換されません。
- 並列処理終了アクティビティは，flow 要素以下が並列処理開始アクティビティに変換されたときに自動的に設定されます。その際，並列処理終了アクティビティの名称として，並列処理開始アクティビティの名称に「_終了」を付けた名称が設定されます。

変換の詳細を次の表に示します。表内の「項目」に示す名称は，並列処理アクティビティダイアログの項目名です。

表 E-32　flow 要素の変換

flow 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	アクティビティの名称が設定されます。ビジネスプロセス内に同一名称のアクティビティがすでに定義されてい

flow 要素の下位要素または属性	ビジネスプロセスの定義内容	
	項目	説明
name 属性	アクティビティ名	る場合，アクティビティ名の末尾に<n>（n は 1 以上の整数）が付与されます。
activity※	*アクティビティ*	ビジネスプロセスを構成するアクティビティに変換されます。

注※

activity の部分には，実際には receive 要素，reply 要素，invoke 要素などが入ります。

(e) sequence 要素の変換

sequence 要素は，順次処理アクティビティの定義内容に変換されます。定義内容は変更できません。

なお，sequence 要素には，変換される下位属性はありません。

サービスプラットフォームでは，「5.6.20 順次処理アクティビティ」に相当する位置に記述された sequence 要素だけが，ビジネスプロセスの定義に変換されます。

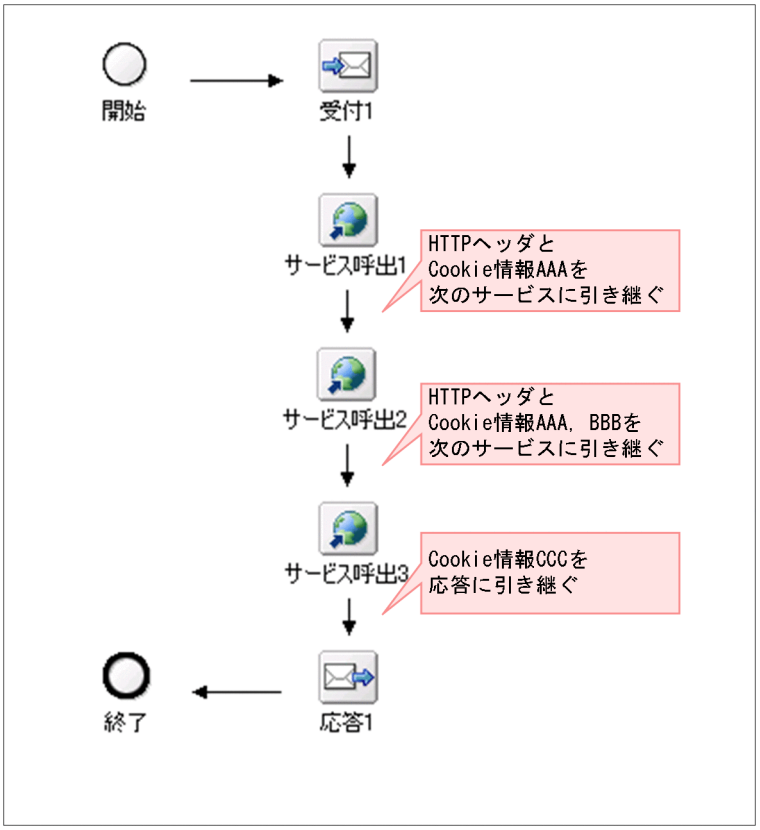
付録 F　サービスアダプタを利用した HTTP ヘッダおよび Cookie 情報の引き継ぎ

サービスアダプタの HTTP ヘッダ引き継ぎ機能を使用して，ビジネスプロセスで HTTP ヘッダおよび Cookie 情報を引き継ぐ方法について説明します。

なお，HTTP ヘッダ引き継ぎ機能の概要については，マニュアル「サービスプラットフォーム 解説」の「2.2.5　Web サービス（SOAP 通信）の場合の HTTP ヘッダ引き継ぎ」を参照してください。

ここでは，次のビジネスプロセスで HTTP ヘッダおよび Cookie 情報を引き継ぐ場合を例に，ビジネスプロセスの定義方法について説明します。

図 F-1　Cookie 情報の引き継ぎに使用するビジネスプロセス



ビジネスプロセスの定義の流れを次に示します。

1. アクティビティを定義する
2. ヘッダ変数のスキーマを作成する
3. ヘッダ変数を作成する
4. サービス呼出アクティビティにヘッダ割当変数を設定する
5. 変換元変数と変換先変数をマッピングする

付録 F.1　アクティビティの定義

ビジネスプロセス定義画面でアクティビティを配置します。アクティビティの配置および連結方法については，「5.4　アクティビティの配置と連結」を参照してください。

付録 F.2　ヘッダ変数のスキーマ作成

HTTP リクエスト用ヘッダ変数および HTTP レスポンス用ヘッダ変数のスキーマを作成します。作成するスキーマは，次に示す目的に応じて，サービスプラットフォームが提供するテンプレートファイルを使い分けます。

- Cookie 情報を編集しないですべて一括して引き継ぎたい。
- 編集できる状態で Cookie 名ごとに分けて引き継ぎたい。

また，SOAP1.1 モードを利用して「JSESSIONID」の Cookie を送受信する場合，クライアント定義ファイルの c4web.application.app_maintainsession を true に変更してください。クライアント定義ファイルの詳細については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」の「10.3　クライアント定義ファイルの設定」を参照してください。

(1)　HTTP リクエスト用ヘッダ変数のスキーマ

HTTP リクエスト用ヘッダ変数のスキーマを作成します。斜体部分が変更個所です。

- HTTP ヘッダリクエスト部分の定義（Cookie 情報を一括で扱う場合）

  テンプレートファイルの格納先：

  <サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_http_header_request1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="soap_cookie_request1.xsd※1"/>
  <xsd:element name="HTTPHeader_request">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Cookies" type="hrc:Cookie_types※2" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="HTTPHeader" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any namespace="##any" processContents="skip"/>
            </xsd:sequence>
          </xsd:complexType>
```

```
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

注※1

Cookie 要素を定義する include 対象のスキーマファイル名に変更します。

注※2

include 対象の name 属性に合わせて変更します。

- Cookie 情報部分を一括で扱う場合に使用する定義（Cookies 要素に include される部分）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_cookie_request1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Cookie_types※">
    <xsd:sequence>
      <xsd:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

注※

受信予定の Cookie 名の name 属性を記述します。

- HTTP ヘッダリクエスト部分の定義（Cookie 情報を個別に扱う場合）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_http_header_request2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="soap_cookie_request2.xsd※1"/>
  <xsd:element name="HTTPHeader_request">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <xsd:element name="Cookies" type="hrc:Cookie_types※2" minOccurs="0" maxOccurs
="1"/>
        <xsd:element name="HTTPHeader" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any namespace="##any" processContents="skip"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

注※1

Cookie 要素を定義する include 対象のスキーマファイル名に変更します。

注※2

include 対象の name 属性に合わせて変更します。

- Cookie 情報部分を個別に扱う場合に使用する定義（Cookies 要素に include される部分）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_cookie_request2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_r
equest"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_request
"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Cookie_types※1">
    <xsd:sequence>
      <xsd:element name="Cookie" type="hrc:Cookie_type" minOccurs="0" maxOccurs="unbou
nded"※2 />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Cookie_type">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="path" type="xsd:string" use="optional"/>
        <xsd:attribute name="host" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

注※1

受信予定の Cookie 名の name 属性を記述します。

注※2

定義できる要素数の上限と下限を maxOccurs 属性および minOccurs 属性で設定します。

HTTP リクエストヘッダ設定用のスキーマの内容を次に示します。

表 F-1　HTTP リクエストヘッダ設定用のスキーマの内容

#	タグ名	型	出現回数	説明
1	HTTPHeader_request	−	1 回	−
2	├<Cookies>
	−	0 または 1 回	HTTP リクエストヘッダ内に格納された Cookie 情報を取得して格納します。
3	└<HTTPHeader>	−	0 または 1 回	HTTP リクエストヘッダ内に格納された拡張ヘッダを取得して格納します。HTTP リクエストヘッダにある拡張ヘッダは無視します。

（凡例）

−：該当する説明はありません。

(2)　HTTP レスポンス用ヘッダ変数のスキーマ

HTTP レスポンス用ヘッダ変数のスキーマを作成します。斜体部分が変更個所です。

- HTTP ヘッダレスポンス部分の定義（Cookie 情報を一括で扱う場合）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_http_header_response1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="soap_cookie_response1.xsd※1"/>
  <xsd:element name="HTTPHeader_response">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Cookies" type="hrc:Cookie_types※2" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="HTTPHeader" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any namespace="##any" processContents="skip"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

注※1

Cookie 要素を定義する include 対象のスキーマファイル名に変更します。

注※2

include 対象の name 属性に合わせて変更します。

- Cookie 情報部分を一括で扱う場合に使用する定義（Cookies 要素に include される部分）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_cookie_response1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Cookie_types※">
    <xsd:sequence>
      <xsd:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

注※

受信予定の Cookie 名の name 属性を記述します。

- HTTP ヘッダレスポンス部分の定義（Cookie 情報を個別に扱う場合）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_http_header_response2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="soap_cookie_response2.xsd※1"/>
  <xsd:element name="HTTPHeader_response">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Cookies" type="hrc:Cookie_types※2" minOccurs="0" maxOccurs="1"/>
```

```
        <xsd:element name="HTTPHeader" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any namespace="##any" processContents="skip"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

注※1

Cookie 要素を定義する include 対象のスキーマファイル名に変更します。

注※2

include 対象の name 属性に合わせて変更します。

- Cookie 情報部分を個別に扱う場合に使用する定義（Cookies 要素に include される部分）

テンプレートファイルの格納先：

<サービスプラットフォームのインストールディレクトリ>¥CSC¥schema¥soap¥soap_cookie_response2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2012, Hitachi, Ltd. -->
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:hrc="http://www.hitachi.co.jp/soft/xml/cosminexus/csc/soap/http/header_response"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Cookie_types※1">
  <xsd:sequence>
    <xsd:element name="Cookie" type="hrc:Cookie_type" minOccurs="0" maxOccurs="unbounded"※2 />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Cookie_type">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="path" type="xsd:string" use="optional"/>
        <xsd:attribute name="host" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

注※1

受信予定の Cookie 名の name 属性を記述します。

注※2

定義できる要素数の上限と下限を maxOccurs 属性および minOccurs 属性で設定します。

HTTP レスポンスヘッダ設定用のスキーマの内容を次に示します。

表 F-2　HTTP レスポンスヘッダ設定用のスキーマの内容

#	タグ名	型	出現回数	説明
1	HTTPHeader_response	−	1 回	−
2	├<Cookies>
−	0 または 1 回	HTTP レスポンスヘッダ内に格納された Cookie 情報を取得して格納します。HTTP レスポンスヘッダにある Set-Cookie 属性は削除されます。
3	└<HTTPHeader>	−	0 または 1 回	HTTP レスポンスヘッダ内に格納された拡張ヘッダを取得して格納します。HTTP レスポンスヘッダにある拡張ヘッダは削除されます。

（凡例）

−：該当する説明はありません。

付録 F.3　ヘッダ変数の作成

HTTP リクエスト用ヘッダ変数および HTTP レスポンス用ヘッダ変数を作成します。作成手順を次に示します。

1. ビジネスプロセス定義画面のキャンバス上の［変数・相関セット］アイコンをダブルクリックします。
   変数・相関セット一覧ダイアログが表示されます。変数・相関セット一覧ダイアログの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.4.1　変数・相関セット一覧ダイアログ」を参照してください。
2. 一覧から［変数一覧］を選択します。
3. 変数名に任意の名称を入力します。
4. ［種別］ドロップダウンリストから［XML］を選択します。
5. ［...］ボタンをクリックして，「付録 F.2　ヘッダ変数のスキーマ作成」で作成した HTTP リクエスト用ヘッダ変数のスキーマを設定します。
6. ［部分指定］チェックボックスにチェックします。
7. ［行追加］ボタンをクリックし，部分名，指定式，および型を指定します。
8. ［追加］ボタンをクリックします。
   変数一覧に，追加した変数が表示されます。
9. 手順 3.～8.と同様の手順で HTTP レスポンス用ヘッダ変数を作成します。

10. [OK] ボタンをクリックします。

付録 F.4　サービス呼出アクティビティへのヘッダ割当変数の設定

サービス呼出アクティビティのヘッダ割当変数に，作成した要求電文用および応答電文用の変数を設定します。

1. ビジネスプロセス定義画面でサービス呼出アクティビティ「サービス呼出 1」をダブルクリックします。
   サービス呼出アクティビティダイアログが表示されます。サービス呼出アクティビティダイアログの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.4.11　サービス呼出アクティビティダイアログ」を参照してください。
2. [要求電文] の［ヘッダ割当変数］の［設定］ボタンをクリックします。
   要求電文用ヘッダ割当変数ダイアログが表示されます。
3. [追加] ボタンをクリックします。
   割当変数を設定するための行が追加されます。
4. [割当変数] 欄をクリックし，サービス呼出 1 に割り当てるリクエスト用の割当変数を選択します。
5. [ルート要素] 欄をクリックし，ルート要素を選択します。
6. [名前空間] 欄に表示される割当変数の名前空間を確認します。
7. [OK] ボタンをクリックします。
8. 手順 2.～7.と同様の手順で［応答電文］のヘッダ割当変数にリクエスト用の割当変数を設定します。
9. 手順 1.～8.と同様の手順でサービス呼出アクティビティ「サービス呼出 2」「サービス呼出 3」にもリクエスト用とレスポンス用の割当変数を設定します。

付録 F.5　変換元変数と変換先変数のマッピング

データ変換定義画面で変換元スキーマと変換先スキーマをマッピングします。

サービス呼出 1 とサービス呼出 2 の間で Cookie 情報を引き継ぐには，サービス呼出 1 の応答電文に設定した割当変数と，サービス呼出 2 の要求電文に設定した割当変数をマッピングします。

同様に，サービス呼出 2 とサービス呼出 3 の間で Cookie 情報を引き継ぐには，サービス呼出 2 の応答電文に設定した割当変数と，サービス呼出 3 の要求電文に設定した割当変数をマッピングします。

また，サービス呼出 3 と応答 1 の間で Cookie 情報を引き継ぐには，サービス呼出 3 の応答電文に設定した割当変数と，応答 1 に設定した割当変数をマッピングします。

手順を次に示します。

1. データ変換アクティビティ「データ変換 1」を右クリックし，［マッピング定義起動］を選択します。

2. 変換元変数のノードと変換先変数のノードをマッピングします。
   引き継ぐデータを編集する場合は，各ファンクションを配置してデータを編集できます。

3. 手順 1.～2.と同様の手順でサービス呼出 2 からサービス呼出 3，およびサービス呼出 3 から応答 1 をマッピングします。

4. データ変換定義画面の変換元スキーマツリービューア，マッピングビューア，または変換先スキーマツリービューアの適当な場所で右クリックして，［検証］を選択します。
   検証が実行されます。検証結果に問題がなければビジネスプロセスを実行できます。

付録 G　コンポーネント共通 UOC

コンポーネントとコンポーネント共通 UOC クラスの関連づけは，プロパティファイルに記述します。

付録 G.1　コンポーネント共通 UOC クラスのプロパティファイル

ファイル名

cscmsg_uoc.properties

記述形式

J2SE のプロパティファイル形式とする。

（UOC クラスの呼び出し元）＝（クラスパス）

記述例

```
ADP1=sample.AdpUocClass
ADP2=sample.AdpUocClass
RCP1=sample.package.RcpUocClass
adapter.standard.soap=sample.SoapUocClass
adapter.custom.ftp=sample.FtpAdpUocClass
reception.urecp.ftp=sample.FtpRcpUocClass
ADP3=
```

付録 G.2　コンポーネント共通 UOC クラスの指定方法

コンポーネント共通 UOC クラスの指定方法は次のとおりです。

- 特定のコンポーネントだけで呼び出すクラスを指定する場合

  コンポーネント個別で，UOC クラスを指定します。

  受付の受付 ID，またはサービスアダプタのサービス ID を（UOC クラスの呼び出し元）に設定します。この場合，受付種別，またはアダプタ種別で，UOC クラスを指定していたとしても，コンポーネント個別に指定した UOC クラスだけを呼び出します。

- 指定したコンポーネント種別で，同一の UOC クラスを呼び出したい場合

  受付種別，またはアダプタ種別で，UOC クラスを指定します。

  UOC クラスの呼び出し元に呼び出し元種別 ID を設定します。

  受付の呼び出し元種別 ID を次に示します。

表 G-1　呼び出し元種別 ID 一覧（受付）

受付		呼び出し元種別 ID
標準受付	標準受付（Web サービス）	reception.standard.soap
	標準受付（SessionBean）	reception.standard.ejb

受付		呼び出し元種別 ID
標準受付	標準受付（MDB（WS-R））	reception.standard.wsr
	標準受付（MDB（DB キュー））	reception.standard.dbq
ユーザ定義受付	SOAP 受付	reception.urecp.soap
	TP1/RPC 受付	reception.urecp.tp1
	FTP 受付	reception.urecp.ftp
	HTTP 受付	reception.urecp.http
	Message Queue 受付	reception.urecp.mq
	ファイルイベント受付	reception.urecp.fileevent
	Kafka 受付	reception.urecp.kafka
	スケジュール駆動受付	reception.urecp.scheduletriggered
	gRPC 受付	reception.urecp.grpc
	カスタム受付	reception.urecp.rcpfw

サービスアダプタの呼び出し元種別 ID を次に示します。

表 G–2　呼び出し元種別 ID 一覧（サービスアダプタ）

サービスアダプタ	呼び出し元種別 ID
SOAP アダプタ	adapter.soap
SessionBean アダプタ	adapter.ejb
MDB（WS-R）アダプタ	adapter.wsr
MDB（DB キュー）アダプタ	adapter.dbq
DB アダプタ	adapter.db
TP1 アダプタ	adapter.tp1
ファイルアダプタ	adapter.ff
Object Access アダプタ	adapter.oa
Message Queue アダプタ	adapter.mq
FTP アダプタ	adapter.ftp
ファイル操作アダプタ	adapter.fop
メールアダプタ	adapter.mail
HTTP アダプタ	adapter.http
コマンドアダプタ	adapter.cmd
SFTP アダプタ	adapter.sftp

サービスアダプタ	呼び出し元種別 ID
Kafka アダプタ	adapter.kafka
gRPC アダプタ	adapter.grpc
汎用カスタムアダプタ	adapter.custom

- 受付種別，またはアダプタ種別で UOC クラスを指定しているが，特定のサービスからは UOC クラスを呼び出さない場合

  特定のサービスのクラスパスを空にします。この場合，そのサービスからは UOC クラスを呼び出しません。

  例） ADP3=

コンポーネント共通 UOC 用プロパティファイルの記述に対する動作を次に示します。

- ISO-8859-1 文字エンコーディングを使用する必要がある。
- 改行までが値になる。
- #で始まる行はコメントと見なす。
- クラスパスの前後の空白は無視する。
- クラスパスの後ろには，コメントなどの文字列は追加できない。追加した場合，コメントを含めてクラスパスと解釈されるため，クラスが見つからない例外となる可能性がある。
- 1 つの呼び出し元に対して，複数のクラスパスを指定した場合，どれか 1 つが有効になる。
- 複数の呼び出し元に対して，1 つのクラスパスを設定した場合，1 つのクラスインスタンスをその呼び出し元で使用する。
- UOC のクラスパスが設定されていないコンポーネントは，UOC を呼び出さない。
- 設定されたクラスパスに対応するクラスが存在しない場合，UOC を呼び出さない。
- コンポーネントの存在有無はチェックしないため，存在しないサービスアダプタのサービス ID，または受付 ID を指定した場合も，UOC クラスのインスタンスを作成する。
- プロパティファイルの変更は，HCSC サーバを再度起動したあとに反映される。

付録 G.3　UOC クラス用 API

インターフェースは，UOC クラスを呼び出すタイミングごとに存在し，インターフェースが追加されるタイミングで UOC クラスを実行します。複数のインターフェースを追加することで，複数の呼び出しタイミングに対応できます。

UOC クラスに実行するインターフェースを次に示します。

(1) インターフェース

UOC クラス内で例外が発生した場合は，CSCUocSystemException にラップして throw します。

(a) UOC クラス共通インターフェース

パッケージ

jp.co.Hitachi.soft.csc.msg.uoc

インターフェース名

ComponentCommonUoc

説明

コンポーネント共通 UOC で使用する定数を定義する共通インターフェースです。

定数

定数を次に示します。

定数	説明
COMPO_TYPE	サービスアダプタか受付を判別するコードを取得する key
COMPO_KIND	コンポーネント種別を取得する key
COMPO_ID	コンポーネント ID を取得する key
TELEGRAM_DATA	電文取得用 API を取得する key
MONITOR_SEND_DATA	パラメタからモニタリングに送信するデータをセットする Map を取得する key
DESTINATION_DATA	呼び出すサービスの宛先情報※を取得する key

注※

ユーザ定義受付（SOAP 受付を除く）の場合だけ取得できます。

定数の使用方法については，マニュアル「サービスプラットフォーム 解説」の「4.9.1　コンポーネント共通 UOC の概要」を参照してください。

(b) リクエスト処理時用インターフェース

パッケージ

jp.co.Hitachi.soft.csc.msg.uoc

インターフェース名

ComponentRequestUoc

説明

リクエスト処理時に呼び出される UOC クラスのインターフェースです。

#request

#request の操作に関する説明を次に示します。

引数

Map<String,Object>

（UOC クラスに渡すデータをまとめたオブジェクト）

戻り値

なし

例外

CSCUocSystemException

(c) 応答処理時用インターフェース

パッケージ

jp.co.Hitachi.soft.csc.msg.uoc

インターフェース名

ComponentResponseUoc

説明

応答処理時に呼び出される UOC クラスのインターフェースです。

#response

#response の操作に関する説明を次に示します。

引数

Map<String,Object>

（UOC クラスに渡すデータをまとめたオブジェクト）

戻り値

なし

例外

CSCUocSystemException

(2) 例外クラス

UOC クラスから例外を throw する場合は，次のクラスにラップして throw します。

パッケージ

jp.co.Hitachi.soft.csc.msg.uoc

例外クラス名

CSCUocSystemException

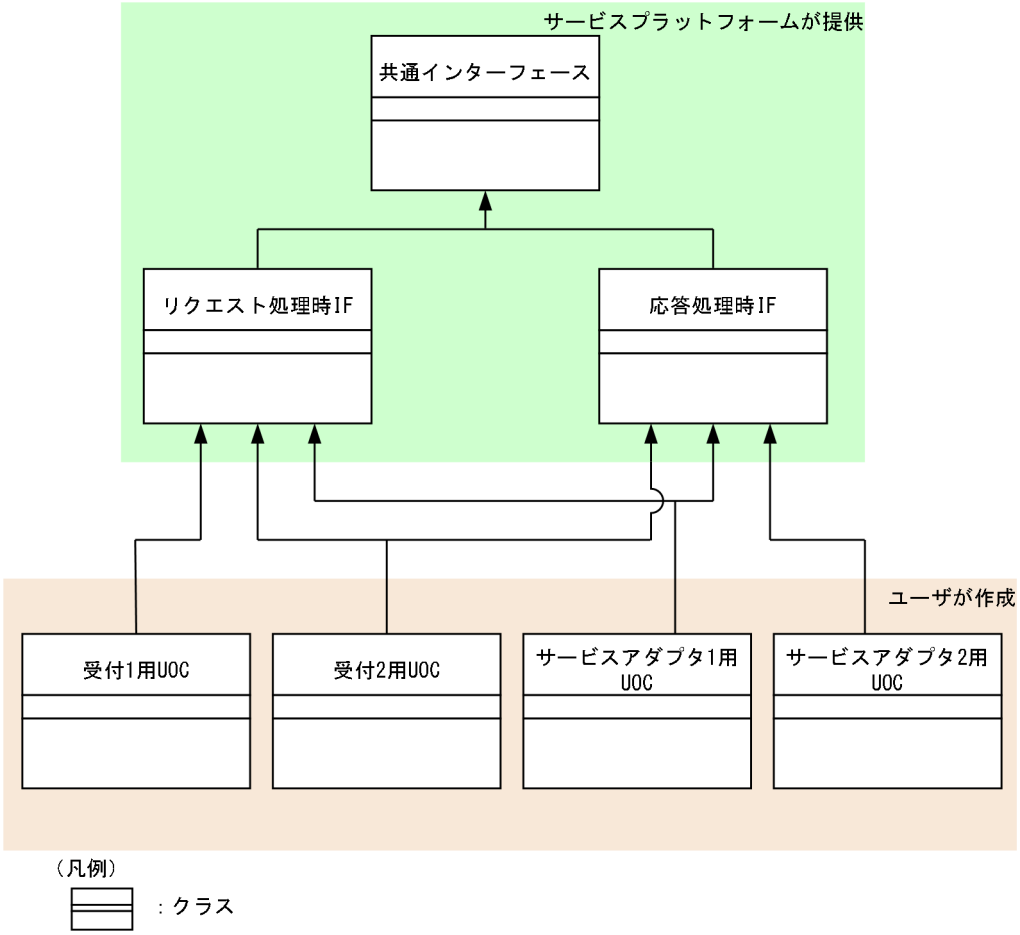
コンストラクタ

```
public CSCUocSystemException()
public CSCUocSystemException(String errorMessage )
```

(3)　クラス構成例

クラス構成例を次に示します。

図 G–1　クラス構成例



付録 G.4　電文取得用 API

UOC クラスにパラメタで設定される電文取得用 API の詳細を次に示します。

パッケージ

jp.co.Hitachi.soft.csc.msg.uoc

クラス名

CSCMsgTelegramManager

定数

定数を次に示します。

定数	説明
static final int	ボディ電文が存在しないことを示す。

定数	説明
NONE = 0	ボディ電文が存在しないことを示す。
static final int TYPE_XML = 1	ボディ電文が XML であることを示す。
static final int TYPE_BINARY = 2	ボディ電文がバイナリであることを示す。
static final int TYPE_ANY = 3	ボディ電文が任意形式（any 型）であることを示す。

#getBody

#getBody の操作に関する説明を次に示します。

引数

なし

戻り値

byte[]（ボディ電文）

説明

ボディ電文を取得します。

ボディ電文が存在しない場合，null が返却されます。

#getHeader

#getHeader の操作に関する説明を次に示します。

引数

String（名前空間），String（要素名）

戻り値

byte[](ヘッダ電文)

説明

ヘッダ電文を取得します。

ヘッダ電文が設定されていない場合，null が返却されます。

#setBody

#setBody の操作に関する説明を次に示します。

引数

byte[]（ボディ電文）

戻り値

なし

説明

ボディ電文を更新します。

#setHeader

#setHeader の操作に関する説明を次に示します。

引数

String（名前空間），String（要素名），byte[]（ヘッダ電文）

戻り値

なし

説明

ヘッダ電文を更新します。

#getTelegramType

#getTelegramType の操作に関する説明を次に示します。

引数

なし

戻り値

int (電文種別)

0:none(電文なし)

1:XML

2:binary

3:any

説明

ボディ電文の種類を取得します。

付録 G.5　UOC クラスの jar ファイル指定方法

UOC クラスをまとめた jar ファイルは，usrconf.cfg に，格納場所を追加します。

指定例

```
add.class.path=C:¥uocjar¥uocClass2.jar
```

付録 G.6　注意事項

UOC クラスを設計する際の注意事項を説明します。

- UOC クラスは J2EE サーバのコンテナ拡張ライブラリの延長として動作するため，利用できる機能に制限があります。

  詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「19.6 コンテナ拡張ライブラリおよびサーバ起動・停止フック機能利用時の制限事項」を参照してください。

UOC クラスは，システムクラスローダで読み込まれるため，上記の制限範囲以外であれば，フルパス指定で外部ファイルの読み書きができます。

- UOC クラス作成時，クラスのアクセス設定修飾子には public を指定します。

  public ではない（アクセス設定修飾子を書かない=パッケージプライベート）場合，IllegalAccessException が発生し，UOC データを送信できません。

- マルチスレッドアクセスを考慮した処理にしてください。

  UOC クラスは，複数のスレッドから呼ばれることがあるため，スレッドセーフな実装になっている必要があります。また，メモリ不足が発生した場合を考慮して，リソースを適切に解放している必要があります。

- 標準受付（SessionBean）の UOC クラスでは，リクエスト処理時の電文に null が入ることを考慮する必要があります。

  ビジネスプロセスを再実行すると，電文が null の状態で標準受付（SessionBean）が呼ばれます。

- コンパイルするときには，cscmsg.jar をクラスパスに追加する必要があります。利用するインターフェースは cscmsg.jar に含まれます。

付録 H　文字コード変換 UOC を利用した文字コード変換

サービスプラットフォームで対応していない文字コードを利用する場合，その文字コードを処理する文字コード変換 UOC を作成することで，対応していない文字コードを扱えるようにします。

注意事項

文字コード変換 UOC は，HCSC サーバに 1 つだけ定義できます。

ここでは，文字コード変換 UOC について説明します。

付録 H.1　文字コード変換 UOC の jar ファイルの開発

文字コード変換 UOC は，サービスプラットフォームが提供する jar ファイルを使用して開発します。

文字コード変換 UOC の開発に必要な jar ファイルに定義されているインターフェースおよび例外クラスは，次の場所に格納されています。

```
<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥cscdt_uoc.jar
```

注意事項

- 開発した文字コード変換 UOC の jar ファイルは，任意のディレクトリに格納してください。また，文字コード変換 UOC をコンパイルするときは，cscdt_uoc.jar をクラスパスに含めてください。
- 文字コード変換 UOC の実行時に確保したリソースは，処理が終了しても保持し続けるため，システム全体に高い負荷が掛かった場合に，Java ヒープ不足などが原因で OutOfMemoryError が発生することがあります。そのため，OutOfMemoryError が発生したときに適切にリソースを解放する処理や，ロールバックなどのエラー処理を実装しておく必要があります。

付録 H.2　文字コード変換 UOC を使用するための設定

文字コード変換 UOC を使用するために必要な設定について説明します。

(1)　プロパティ（独自定義ファイル）の定義

文字コード変換 UOC は，カスタム受付または独自定義ファイルを取得することで，文字コード変換処理を切り分けることができます。ただし，独自定義ファイルが指定できないデータ変換アクティビティや，データ変換 API で文字コード変換 UOC を利用する場合は切り分けることができません。独自定義ファイルのファイル名は，必ず「csc_owncodeconvert.properties」にしてください。

独自定義ファイルは，「<キー値>= <指定値>」の形式で指定します。

キー値および指定値には，任意の値を指定してください。

(2) 文字コード変換 UOC の jar ファイルを使用するための設定

開発環境で作成した文字コード変換 UOC を実行環境で使用するには，次の設定が必要です。

- J2EE サーバ用オプション定義ファイル（usrconf.cfg）にクラスパスを追加
- J2EE サーバ用ユーザプロパティファイル（usrconf.properties）にクラス名を登録

(a) クラスパスの追加

J2EE サーバ用オプション定義ファイル（usrconf.cfg）にクラスパスを追加する手順を次に示します。

1. 運用管理ポータルから，［論理サーバの環境設定］を選択します。
2. サーバビューから，［論理 J2EE サーバ］ − ［J2EE サーバ］ − ［<論理サーバ名>］を選択します。
3. ［コンテナ］タブの［拡張パラメタ］にクラスパスを指定します。
   クラスパスは次の形式で指定します。

```
add.class.path=<jarファイルのパス>
```

   <jar ファイルのパス>
   文字コード変換 UOC の jar ファイルを絶対パスで指定します。

(b) jar ファイルの登録

J2EE サーバ用ユーザプロパティファイル（usrconf.properties）にクラス名を登録する手順を次に示します。

1. 運用管理ポータルから，［論理サーバの環境設定］を選択します。
2. サーバビューから，［論理 J2EE サーバ］ − ［J2EE サーバ］ − ［<論理サーバ名>］を選択します。
3. ［JVM］タブの［プロパティ］にクラス名（パッケージ名を含む）を指定します。
   クラス名（パッケージ名を含む）は次の形式で指定します。

```
csc.dt.ownCodeConverter.className=<文字コード変換UOCのクラス名>
```

   <文字コード変換 UOC のクラス名>
   文字コード変換 UOC（選択文字コード：CUSTOM）のクラス名を完全修飾名で指定します。

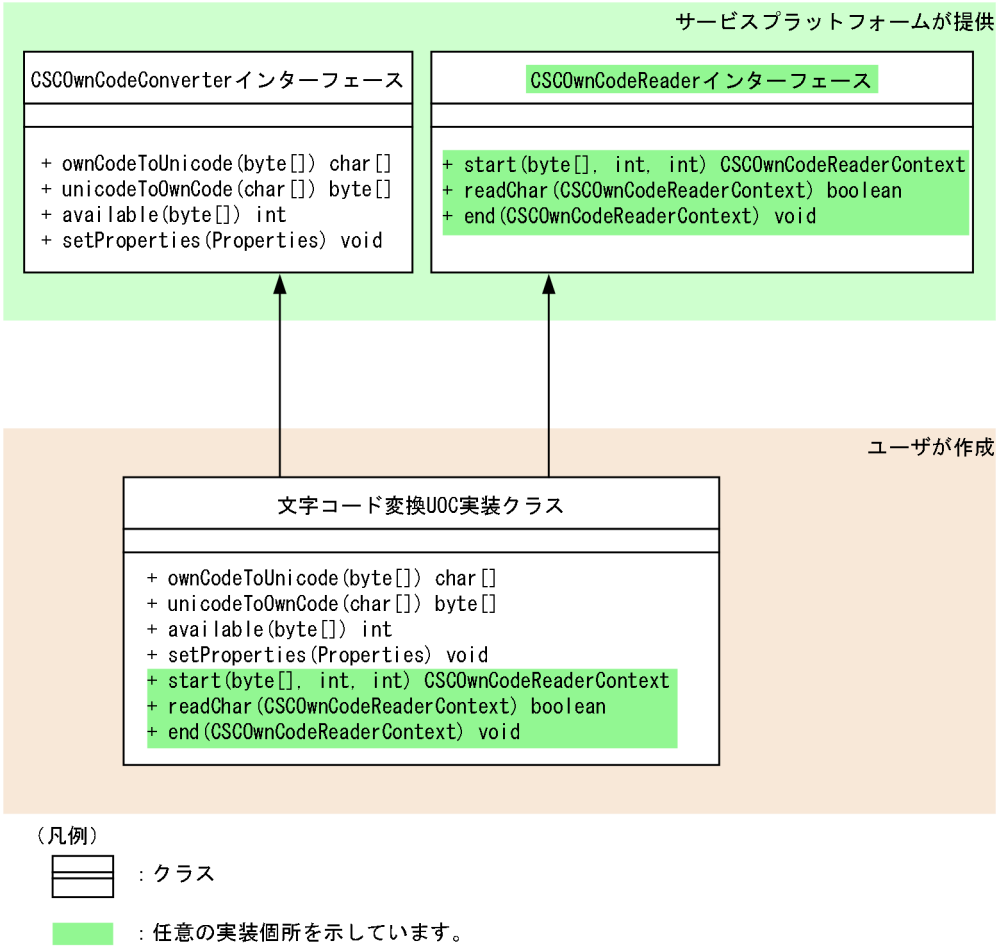
付録 H.3　CSCOwnCodeConverter インターフェース

(1)　インターフェース

CSCOwnCodeConverter インターフェースを次に示します。

文字コード変換 UOC を開発する場合，次のインターフェースの実装クラスを作成します。

図 H-1　文字コード変換 UOC クラス



インターフェース名

CSCOwnCodeConverter インターフェース

説明

文字コード変換 UOC を実装するためのインターフェースです。

CSCOwnCodeConverter のパッケージ名は，
jp.co.Hitachi.soft.csc.dt.uoc.CSCOwnCodeConverter です。

形式

```
public class OwnCodeConverter implements CSCOwnCodeConverter
{
  public void setProperties(final Properties properties)
           throws CSCOwnCodeConverterException;
```

```
  public char[] ownCodeToUnicode(final byte[] inBuffer)
           throws CSCOwnCodeConverterException;
  public byte[] unicodeToOwnCode(final char[] inBuffer)
           throws CSCOwnCodeConverterException;
  public int available(final byte[] inBuffer)
           throws CSCOwnCodeConverterException;
}
```

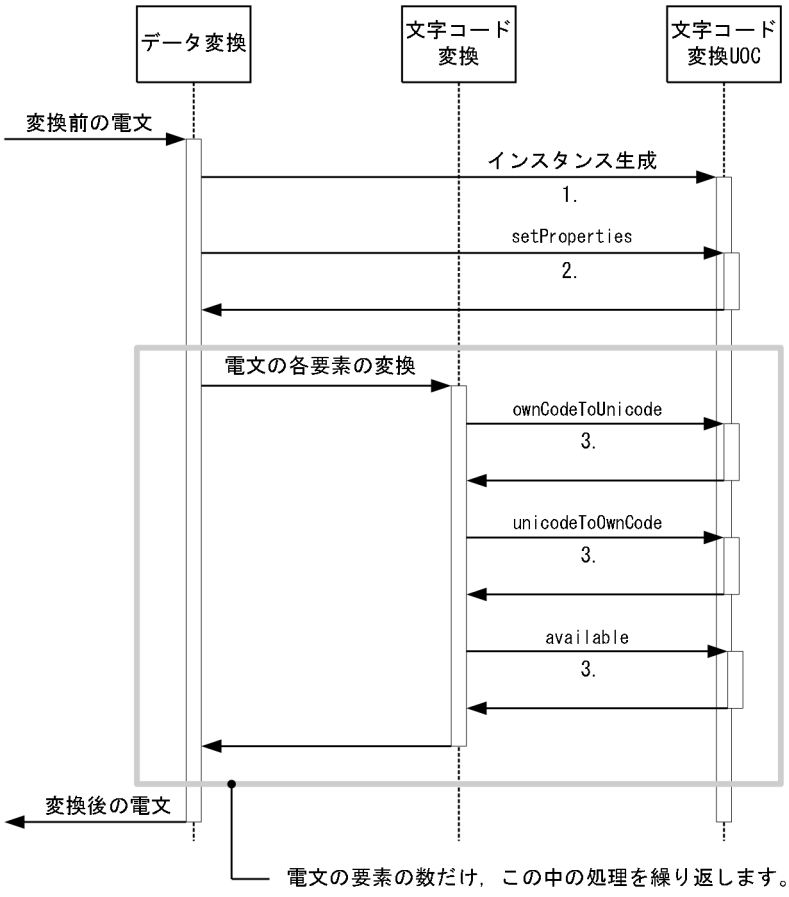
メソッド

CSCOwnCodeConverter インターフェースのメソッドを次の表に示します。

メソッド名	説明
setProperties メソッド	独自定義ファイルの定義内容を文字コード変換 UOC に渡すためのメソッドです。
ownCodeToUnicode メソッド	独自文字コードの文字列を Unicode に変換するメソッドです。
unicodeToOwnCode メソッド	Unicode（UTF-16）の文字列を独自文字コードに変換するメソッドです。
available メソッド	文字コード変換時に変換できる文字列のバイト数を返すメソッドです。

文字コード変換 UOC からの各メソッドの呼び出し順序を次の図に示します。

図 H-2　文字コード変換 UOC からの各メソッドの呼び出し順序



1. インスタンス生成

変換前の電文を受け取ったあと，データ変換によって文字コード変換 UOC のインスタンスを生成します。

2. setProperties メソッド

独自定義ファイルの定義内容を文字コード変換 UOC に渡します。このメソッドは，文字コード変換 UOC 開始時に一度だけ呼ばれます。

3. ownCodeToUnicode メソッド，unicodeToOwnCode メソッド，available メソッド

それぞれのメソッドは，文字コード変換 UOC 実行時に文字コード変換から呼ばれます。メソッドの実行順序は電文フォーマットに依存します。

- ownCodeToUnicode メソッド
  このメソッドで独自文字コードの文字列を Unicode に変換します。
- unicodeToOwnCode メソッド
  このメソッドで Unicode（UTF-16）の文字列を独自文字コードに変換します。
- available メソッド
  このメソッドで文字コード変換時に変換できる文字列のバイト数を返します。

(a) setProperties メソッド

説明

独自定義ファイルの定義内容を文字コード変換 UOC に渡します。

形式

```
public void setProperties(final Properties properties)
         throws CSCOwnCodeConverterException;
```

パラメタ

properties：

独自定義ファイルの定義内容が格納されます。

注意事項

次の場合，setProperties メソッドの引数は必ず null になります。setProperties メソッドの引数が null の場合でも，文字コード変換 UOC がエラーにならないように実装してください。

- 独自定義ファイルが存在しない（定義されていない）場合
- カスタム受付以外の受付（独自定義ファイルを定義できない受付）から文字コード変換 UOC を実行した場合
- 独自定義ファイルを定義できないアダプタから文字コード変換 UOC を実行した場合

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

なし。

(b) ownCodeToUnicode メソッド

説明

独自文字コードの文字列を Unicode に変換します。

形式

```
public char[] ownCodeToUnicode(final byte[] inBuffer)
         throws CSCOwnCodeConverterException;
```

パラメタ

inBuffer：

独自文字コードの文字列を格納したバイト配列のバッファが格納されます。このバッファは読み込み専用のため，編集はできません。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

Unicode に変換した文字列を格納した文字配列のバッファを返します。

(c) unicodeToOwnCode メソッド

説明

Unicode(UTF-16)の文字列を独自文字コードに変換します。

形式

```
public byte[] unicodeToOwnCode(final char[] inBuffer)
         throws CSCOwnCodeConverterException;
```

パラメタ

inBuffer：

Unicode の文字列を格納した文字配列のバッファが格納されます。

このバッファは読み込み専用のため，編集はできません。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

独自文字コードに変換した文字列を格納したバイト配列バッファを返します。

(d) available メソッド

説明

文字コード変換時に変換できる文字列のバイト数を返します。

形式

```
public int available(final byte[] inBuffer)
         throws CSCOwnCodeConverterException;
```

パラメタ

inBuffer：

独自文字コードの文字列を格納したバイト配列のバッファが格納されます。このバッファは読み込み専用のため，編集はできません。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

正常に変換できる文字列のバイト数を格納します。ただし，変換できない文字がある場合は，その文字の前の文字までのバイト数を格納します。

(2) 例外クラス

文字コード変換 UOC の開発時に発生する例外クラスを次に示します。

クラス名

CSCOwnCodeConverterException クラス

説明

文字コード変換処理中に異常が発生した場合に送出する例外です。

この例外が発生した場合は，データ変換処理全体を中断します。

(3) 実装例

CSCOwnCodeConverter インターフェースの実装例を次に示します。

```
public class OwnConvertUoc implements CSCOwnCodeConverter {

   // コード変換オプション
   HJCOption option = null;
   // コード変換結果
   HJCResult result = null;
   // エンコード
   String encode = null;

   // コンストラクタ
```

```
public void OwnConvertUoc() {
   result = new HJCResult();
   option = new HJCOption();
}

// 独自定義ファイルをJavaのプロパティ形式で受け取る
public void setProperties(final Properties properties)
               throws CSCOwnCodeConverterException {

   if ( properties == null ) {
      // 独自定義ファイルが登録されていない
      // 独自定義ファイルを定義できない受付・サービスアダプタから実行されている
      // 可能性があるのでエラーにしない
      encode = "";
      return;
   }

   encode = properties.getProperty("encode");

   if ( encode == null ) {
      // キーがない場合
      String message = "文字コードが指定されていません。";
      throw new CSCOwnCodeConverterException(message);
   }
}

// 独自コードをUnicodeに変換する
public char[] ownCodeToUnicode(final byte[] inBuffer)
                throws CSCOwnCodeConverterException {

   if ( encode.equals("SJIS") ) {
      // SJIS(MS932)からUnicodeに変換
      // コード変換を使用
      HJCString inStr = new HJCString( inBuffer );
      try {
         HJCConverters.cs_ms932tounicode(inStr, result, option);
      } catch ( Exception e ){
         throw new CSCOwnCodeConverterException(e);
      }
      return result.getStrResult().toString().toCharArray();
   }
   else if ( encode.equals("KEIS") ) {
      // KEISからUnicodeに変換
      // コード変換を使用
      HJCString inStr = new HJCString( inBuffer );
      try {
         HJCConverters.cs_keistounicode(inStr, result, option);
      } catch ( Exception e ){
         throw new CSCOwnCodeConverterException(e);
      }
      return result.getStrResult().toString().toCharArray();
   }
   else {
      // 対象外の文字コード
      String message = "対象外の文字コードです。";
      throw new CSCOwnCodeConverterException(message);
   }
}
```

```
    // Unicodeを独自コードに変換する
    public byte[] unicodeToOwnCode(final char[] inBuffer)
                    throws CSCOwnCodeConverterException {
        // (中略)
    }

    // 正常に変換できるバイト数を返す
    public int available(final byte[] inBuffer)
                    throws CSCOwnCodeConverterException {
        // (中略)
    }
}
```

注意事項

例と同じ実装にする場合は，別途 Code Converter を購入する必要があります。

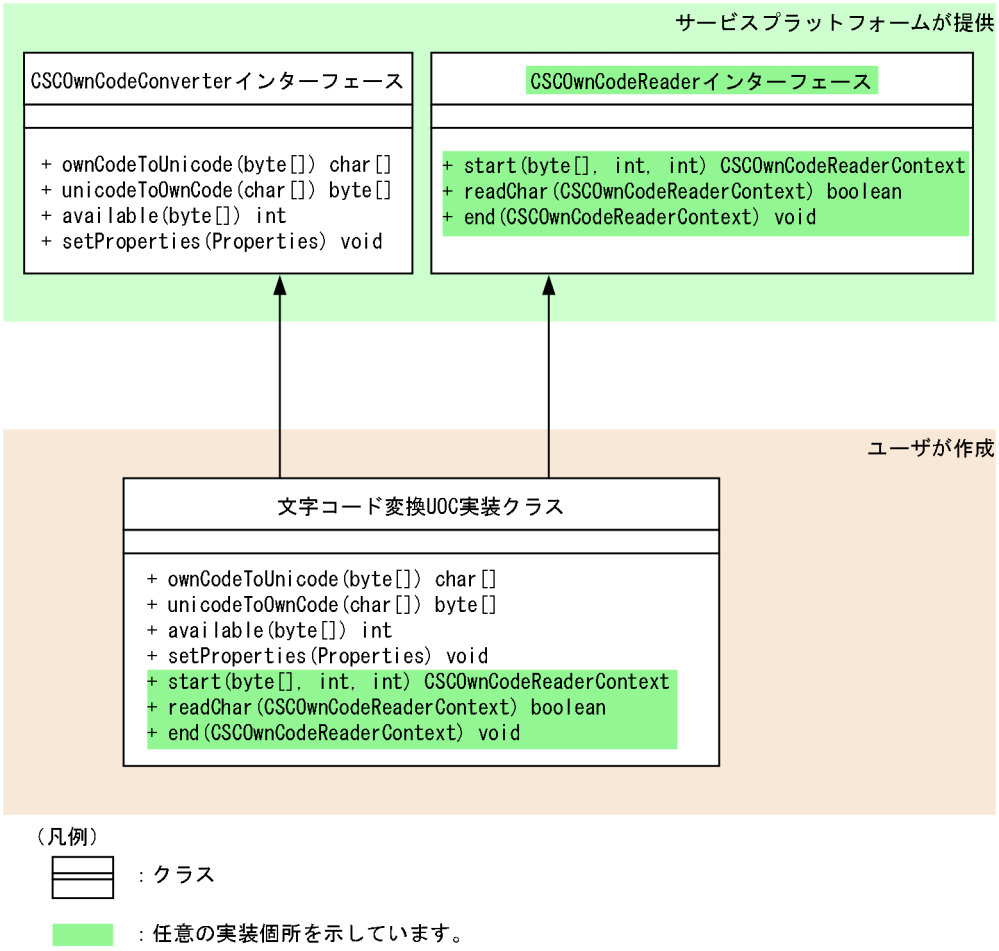
付録 H.4　CSCOwnCodeReader インターフェース

文字コード変換 UOC を開発する場合，CSCOwnCodeReader インターフェースの実装クラスを作成します。未定義文字をセパレータに使用する場合はこのインターフェースを実装してください。

(1)　インターフェース

CSCOwnCodeReader インターフェースを次に示します。

図 H-3　CSCOwnCodeReader インターフェース



インターフェース名

CSCOwnCodeReader インターフェース

説明

独自文字コードの文字列を読み込むためのインターフェースです。

CSCOwnCodeReader のパッケージ名は，jp.co.Hitachi.soft.csc.dt.uoc.CSCOwnCodeReader です。

バイナリ文字列の読み込み処理で，文字コードを変換しないでセパレータを解析することで，バイト数や文字数が変化する変換の制限を緩和したり，変換の性能を向上させたりします。

このインターフェースの実装は任意です。実装していない場合，文字コード変換を実行してセパレータを解析します。

このインターフェースのインスタンスは，複数のスレッドで共用されるため，実装はスレッドセーフにする必要があります。

形式

```
package jp.co.Hitachi.soft.csc.dt.uoc ;

import jp.co.Hitachi.soft.csc.dt.uoc.CSCOwnCodeConverterException ;

public interface CSCOwnCodeReader {
```

```
        CSCOwnCodeReaderContext start( byte[] data, int offset, int length )
                throws CSCOwnCodeConverterException ;

        boolean readChar( CSCOwnCodeReaderContext context )
                throws CSCOwnCodeConverterException ;

        void end( CSCOwnCodeReaderContext context ) ;
}
```

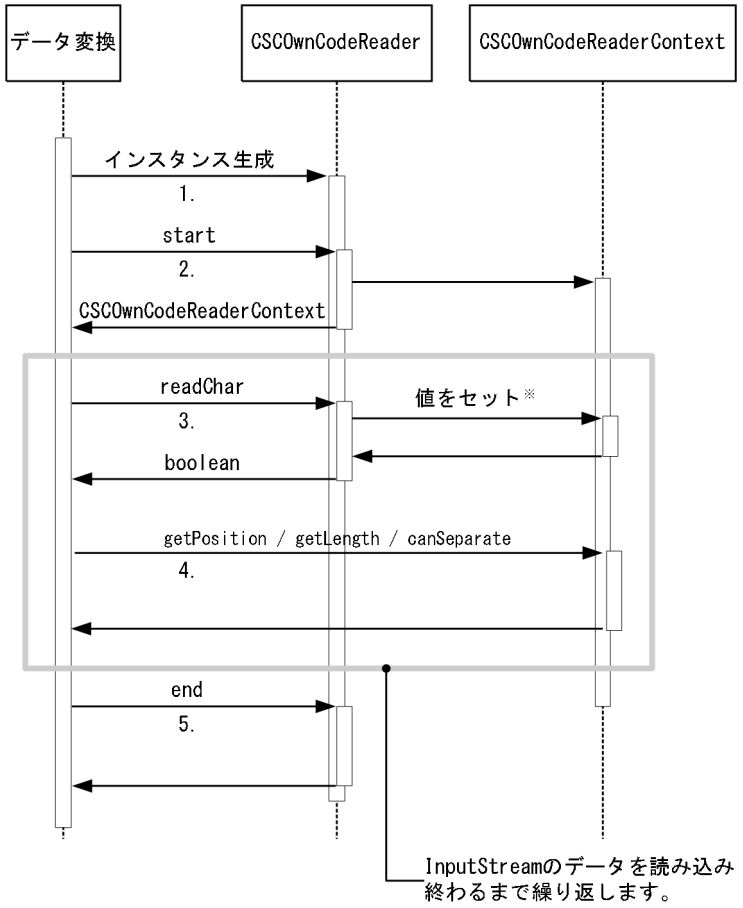
メソッド

CSCOwnCodeReader インターフェースのメソッドを次の表に示します。

メソッド名	説明
start メソッド	独自文字コード文字列の読み込みを開始するためのメソッドです。
readChar メソッド	独自文字コード文字を 1 文字読み込むメソッドです。
end メソッド	独自文字コード文字列の読み込みを終了するメソッドです。

データ変換の対象が可変長文字列で，かつバイナリフォーマット定義にセパレータが設定されている場合，データ変換はセパレータの解析処理を実行します。CSCOwnCodeReader および CSCOwnCodeReaderContext の各メソッドの呼び出し順序を次の図に示します。

図 H-4　CSCOwnCodeReader および CSCOwnCodeReaderContext の各メソッドの呼び出し順序



注※
ユーザが実装するクラス間での値の受け渡しであるため，
インターフェースは規定していない。

1. インスタンス生成

データ変換によって CSCOwnCodeReader のインスタンスを生成します。

2. CSCOwnCodeReader#start メソッド

自スレッド専用の CSCOwnCodeReaderContext のインスタンスを生成します。以降，CSCOwnCodeReader との値の受け渡しは，自スレッドが保持する CSCOwnCodeReaderContext のインスタンスで実行します。

3. CSCOwnCodeReader#readChar メソッド

readChar の実行時，データ変換の処理は引数に CSCOwnCodeReaderContext のインスタンスを渡します。readChar の処理では，解析結果を CSCOwnCodeReaderContext のインスタンスにセットします。セットされた解析結果は，データ変換の処理でセパレータの解析に使用します。

4. getPosition メソッド，getLength メソッド，canSeparate メソッド

それぞれのメソッドは，データ変換から呼ばれます。メソッドの実行順序は電文フォーマットに依存します。

- getPosition メソッド
  このメソッドで現在の文字位置を返します。
- getLength メソッド
  このメソッドで現在の文字長を返します。
- canSeparate メソッド
  このメソッドで現在の文字をセパレータの解析対象とするかどうかを返します。

5. CSCOwnCodeReader#end メソッド

CSCOwnCodeReaderContext の実装クラスの解放処理が必要な場合は，end メソッドを実行します。

(a) start メソッド

説明

独自文字コード文字列の読み込みを開始します。

形式

```
public CSCOwnCodeReaderContext start( byte[] data, int offset, int length )
```

パラメタ

data：

読み込み対象データです。

offset：

読み込み開始位置です。

length：

読み込み開始位置からの長さです。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

CSCOwnCodeReaderContext の実装クラスのインスタンスです。このインスタンスは，ほかのスレッドで使用できません。

(b) readChar メソッド

説明

独自文字コード文字を 1 文字読み込みます。

文字の位置，長さは，CSCOwnCodeReaderContext#getPosition，CSCOwnCodeReaderContext#getLength で取得できます。

形式

```
public boolean readChar( CSCOwnCodeReaderContext context )
```

パラメタ

context：

#start で返された CSCOwnCodeReaderContext の実装クラスのインスタンスです。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

読み込める上限を超えた場合，および文字を変換できない場合，false を返します。それ以外の場合は true を返します。

(c) end メソッド

説明

独自文字コード文字列の読み込みを終了します。

読み込み処理の成功または失敗（エラー）に関係なく必ず呼ばれます。

形式

```
public void end( CSCOwnCodeReaderContext context )
```

パラメタ

context：

#start で返された CSCOwnCodeReaderContext の実装クラスのインスタンスです。

例外

CSCOwnCodeConverterException：

文字コード変換処理中に異常が発生したため，データ変換処理全体を中断しました。

戻り値

なし。

(2) 例外クラス

文字コード変換 UOC の開発時に発生する例外クラスを次に示します。

クラス名

CSCOwnCodeConverterException クラス

説明

文字コード変換処理中に異常が発生した場合に送出する例外です。

この例外が発生した場合は，データ変換処理全体を中断します。

(3) 実装例（MS932）

CSCOwnCodeReader インターフェースの実装例（MS932）を次に示します。

```
public class CSCOwnCodeReaderImpl implements CSCOwnCodeConverter, CSCOwnCodeReader {

    private static final String UNICODE = "ISO-10646-UCS-2" ;

    private final HJCOption option ;

    private static byte[] charSizeTable = initCharSizeTable() ;

    private static byte[] initCharSizeTable() {

        final byte[] objTable = new byte[0x100] ;

        for ( int i = 0; i <= 0xff; i++ ) {
            if ( i <= 0x80
                || (i >= 0xA0 && i <= 0xDF)
                || (i >= 0xFD && i <= 0xFF) ) {
                objTable[i] = 1 ;
            }
            else {
                objTable[i] = 2 ;
            }
        }

        return objTable ;
    }


    public CSCOwnCodeReaderImpl() {

        option = new HJCOption() ;
        try {
            // Unicodeはビッグエンディアン
            option.enableOption( HJCOption.COP_BIGENDIAN ) ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
        }
    }


    @Override
    public void setProperties( Properties properties )
        throws CSCOwnCodeConverterException {

        String codetablepath = null ;
        if ( properties != null ) {
            codetablepath = properties.getProperty( "codetablepath" ) ;
        }

        try {
```

```
            if ( codetablepath == null ) {
                option
                    .setTablePath( "C:¥¥Program Files¥¥HITACHI¥¥Cosminexus¥¥CSC¥¥lib¥¥extern
al¥¥table" ) ;
            }
            else {
                option.setTablePath( codetablepath ) ;
            }
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
        }
    }


    @Override
    public int available( byte[] inBuffer ) throws CSCOwnCodeConverterException {

        if ( inBuffer == null ) {
            final String message = "空文字列は変換できません。" ;
            throw new CSCOwnCodeConverterException( message ) ;
        }

        int retInt = -1 ;
        final HJCResult result = new HJCResult() ;
        final HJCString inStr = new HJCString( inBuffer ) ;

        try {
            HJCConverters.cs_ms932tounicode( inStr, result, option ) ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        final byte[] resultData = result.getStrResult().getBytes() ;
        final int resultState = result.getConvertState() ;
        if ( resultState == HJCConvertState.CST_NORMAL ) {
            // 変換が正常終了
            if ( resultData != null ) {
                retInt = result.getResultLength() ;
            }
        }
        else {
            // 変換が異常終了
            final byte[] bytes = new byte[result.getResultLength() - 1] ;
            System.arraycopy(
                inBuffer,
                0,
                bytes,
                0,
                result.getResultLength() - 1 ) ;
            retInt = available( bytes ) ;
        }
        return retInt ;
    }
```

```
    @Override
    public char[] ownCodeToUnicode( byte[] inBuffer )
        throws CSCOwnCodeConverterException {

        char[] retChar = null ;
        final HJCResult result = new HJCResult() ;
        final HJCString inStr = new HJCString( inBuffer ) ;

        try {
            HJCConverters.cs_ms932tounicode( inStr, result, option ) ;
            final byte[] resultData = result.getStrResult().getBytes() ;
            final String retstr = new String( resultData, UNICODE ) ;
            retChar = retstr.toCharArray() ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        return retChar ;
    }


    @Override
    public byte[] unicodeToOwnCode( char[] inBuffer )
        throws CSCOwnCodeConverterException {

        byte[] retByte = null ;
        final String data = new String( inBuffer ) ;
        final HJCResult result = new HJCResult() ;

        try {
            final HJCString inStr = new HJCString( data.getBytes( UNICODE ) ) ;
            HJCConverters.cs_unicodetoms932( inStr, result, option ) ;
            retByte = result.getStrResult().getBytes() ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        return retByte ;
    }


    @Override
    public CSCOwnCodeReaderContext start( byte[] data, int offset, int length )
        throws CSCOwnCodeConverterException {

        return new CSCOwnCodeReaderContextImpl( Arrays.copyOfRange(
            data,
            offset,
            length ) ) ;
    }


    @Override
    public boolean readChar( CSCOwnCodeReaderContext context )
```

```
        throws CSCOwnCodeConverterException {

        final CSCOwnCodeReaderContextImpl contextImpl = (CSCOwnCodeReaderContextImpl)context
 ;

        final int offset = contextImpl.getNextPosition() ;
        contextImpl.setPosition( offset ) ;

        final byte[] data = contextImpl.getData() ;

        // 入力データの最大長を取得(文字のサイズではない)
        final int maxLength = data.length ;
        if ( offset >= maxLength ) {
            // 現在位置が入力データの範囲外である
            return false ;
        }

        final int len = charSizeTable[data[offset] & 0xff] ;

        contextImpl.setLength( len ) ;

        final int next = offset + len ;
        contextImpl.setNextPosition( next ) ;

        if ( next > maxLength ) {
            // 変換結果が異常
            // 入力データが不正な場合に発生
            // falseを返して解析を中止する
            return false ;
        }

        return true ;
    }


    @Override
    public void end( CSCOwnCodeReaderContext context )
        throws CSCOwnCodeConverterException {

        // 解放すべきリソースがないため，何もしない
    }

}
```

(4) 実装例（IBM 漢字コード）

CSCOwnCodeReader インターフェースの実装例（IBM 漢字コード）を次に示します。

```
public class CSCOwnCodeReaderImpl
    implements
        CSCOwnCodeConverter,
        CSCOwnCodeReader {

    private static final String UNICODE = "ISO-10646-UCS-2" ;

    private static final byte SHIFT_SINGLEBYTE = (byte)0x0f ;
```

```
    private static final byte SHIFT_MULTIBYTE = (byte)0x0e ;

    private final HJCOption option ;


    public CSCOwnCodeReaderImpl() {

        option = new HJCOption() ;
        try {
            // Unicodeはビッグエンディアン
            option.enableOption( HJCOption.COP_BIGENDIAN ) ;
            // EBCDIC
            option.enableOption( HJCOption.COP_EBCDIC ) ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
        }
    }


    @Override
    public void setProperties( Properties properties )
        throws CSCOwnCodeConverterException {

        String codetablepath = null ;
        if ( properties != null ) {
            codetablepath = properties.getProperty( "codetablepath" ) ;
        }

        try {
            if ( codetablepath == null ) {
                option
                    .setTablePath( "C:¥¥Program Files¥¥HITACHI¥¥Cosminexus¥¥CSC¥¥lib¥¥extern
al¥¥table" ) ;
            }
            else {
                option.setTablePath( codetablepath ) ;
            }
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
        }
    }


    @Override
    public int available( byte[] inBuffer ) throws CSCOwnCodeConverterException {

        if ( inBuffer == null ) {
            final String message = "空文字列は変換できません。" ;
            throw new CSCOwnCodeConverterException( message ) ;
        }

        int retInt = -1 ;
        final HJCResult result = new HJCResult() ;
        final HJCString inStr = new HJCString( inBuffer ) ;
```

```
        try {
            HJCConverters.cs_ibmtounicode( inStr, result, option ) ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        final byte[] resultData = result.getStrResult().getBytes() ;
        final int resultState = result.getConvertState() ;
        if ( resultState == HJCConvertState.CST_NORMAL ) {
            // 変換が正常終了
            if ( resultData != null ) {
                retInt = result.getResultLength() ;
            }
        }
        else {
            // 変換が異常終了
            final byte[] bytes = new byte[result.getResultLength() - 1] ;
            System.arraycopy(
                inBuffer,
                0,
                bytes,
                0,
                result.getResultLength() - 1 ) ;
            retInt = available( bytes ) ;
        }
        return retInt ;
    }


    @Override
    public char[] ownCodeToUnicode( byte[] inBuffer )
        throws CSCOwnCodeConverterException {

        char[] retChar = null ;
        final HJCResult result = new HJCResult() ;
        final HJCString inStr = new HJCString( inBuffer ) ;

        try {
            HJCConverters.cs_ibmtounicode( inStr, result, option ) ;
            final byte[] resultData = result.getStrResult().getBytes() ;
            final String retstr = new String( resultData, UNICODE ) ;
            retChar = retstr.toCharArray() ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        return retChar ;
    }


    @Override
    public byte[] unicodeToOwnCode( char[] inBuffer )
        throws CSCOwnCodeConverterException {
```

```
        byte[] retByte = null ;
        final String data = new String( inBuffer ) ;
        final HJCResult result = new HJCResult() ;

        try {
            final HJCString inStr = new HJCString( data.getBytes( UNICODE ) ) ;
            HJCConverters.cs_unicodetoibm( inStr, result, option ) ;
            retByte = result.getStrResult().getBytes() ;
        }
        catch ( Exception e ) {
            e.printStackTrace() ;
            throw new CSCOwnCodeConverterException( e ) ;
        }

        return retByte ;
    }


    @Override
    public CSCOwnCodeReaderContext start( byte[] data, int offset, int length )
        throws CSCOwnCodeConverterException {

        final byte[] tempData = Arrays.copyOfRange( data, offset, length ) ;

        // offsetの位置から解析を開始
        return new CSCOwnCodeReaderContextImpl( tempData ) ;
    }


    @Override
    public boolean readChar( CSCOwnCodeReaderContext context )
        throws CSCOwnCodeConverterException {

        final CSCOwnCodeReaderContextImpl contextImpl = (CSCOwnCodeReaderContextImpl)context
;

        // 現在位置
        final int position = contextImpl.getNextPosition() ;

        // 解析対象の文字列のバイト列
        final byte[] data = contextImpl.getData() ;

        // 解析の上限は入力データの最後まで
        final int maxLength = data.length ;

        // 現在位置から1バイトずつ解析する
        for ( int i = position; i < maxLength; i++ ) {

            // 次の文字を解析
            if ( data[i] == SHIFT_SINGLEBYTE ) {
                // シングルバイトモードに遷移
                contextImpl.setLength( 1 ) ;
                contextImpl.setCanSeparate( true ) ;
                continue ;
            }
            else if ( data[i] == SHIFT_MULTIBYTE ) {
                // マルチバイトモードに遷移
                contextImpl.setLength( 2 ) ;
```

```
                contextImpl.setCanSeparate( false ) ;
                continue ;
            }

            contextImpl.setPosition( i ) ;
            contextImpl.setNextPosition( i + contextImpl.getLength() ) ;

            if ( contextImpl.getNextPosition() > maxLength ) {
                // データが足りない
                return false ;
            }

            // 解析に成功した
            return true ;
        }

        contextImpl.setPosition( maxLength ) ;
        contextImpl.setNextPosition( maxLength ) ;

        return false ;
    }


    @Override
    public void end( CSCOwnCodeReaderContext context )
        throws CSCOwnCodeConverterException {

        // 解放すべきリソースがないため，何もしない
    }

}
```

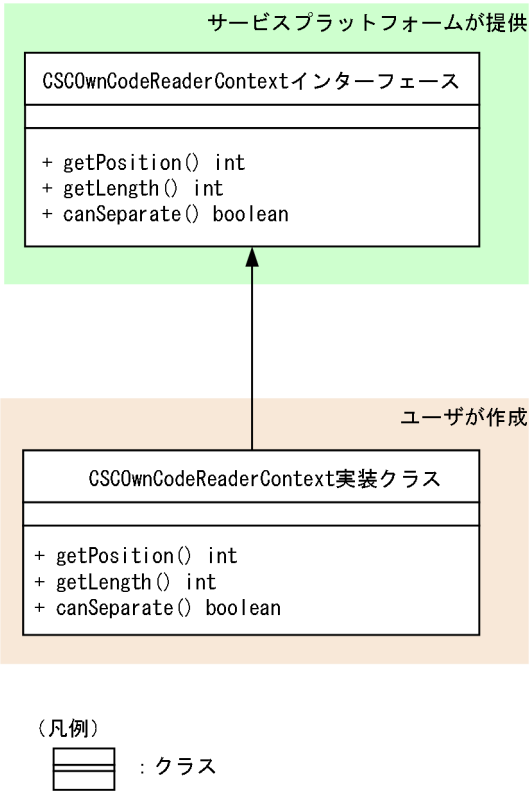
付録 H.5　CSCOwnCodeReaderContext インターフェース

(1)　インターフェース

読み込み情報インターフェースを次に示します。

文字コード変換 UOC を開発する場合，次のインターフェースの実装クラスを作成します。

図 H-5　CSCOwnCodeReaderContext 実装クラス



インターフェース名

CSCOwnCodeReaderContext インターフェース

説明

読み込み情報インターフェースです。

CSCOwnCodeReaderContext のパッケージ名は，
jp.co.Hitachi.soft.csc.dt.uoc.CSCOwnCodeReaderContext です。

CSCOwnCodeReader の解析結果を，データ変換の処理に渡すために使用します。

1 つのスレッドに 1 つのインスタンスを生成します。CSCOwnCodeReader#readChar では，文字列の解析結果を CSCOwnCodeReaderContext の実装クラスにセットする必要があります。セットされた情報は，データ変換の処理で参照されます。

形式

```
package jp.co.Hitachi.soft.csc.dt.uoc ;

public interface CSCOwnCodeReaderContext {

       int getPosition() ;

       int getLength() ;

       boolean canSeparate() ;
}
```

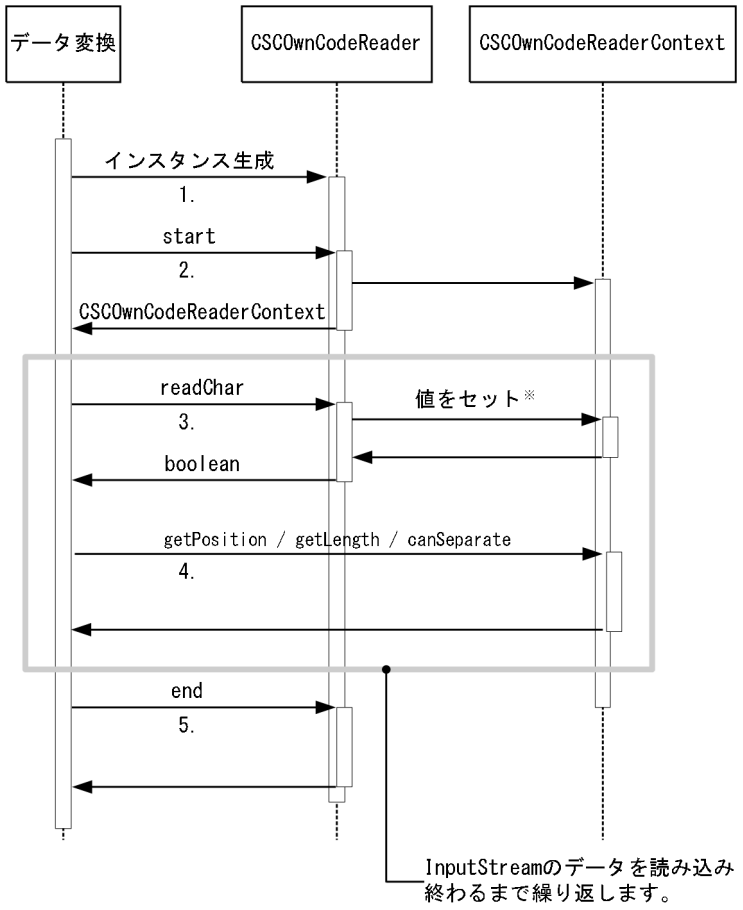
メソッド

CSCOwnCodeReaderContext インターフェースのメソッドを次の表に示します。

メソッド名	説明
getPosition メソッド	現在の文字位置または読み込めるデータのサイズを返すメソッドです。
getLength メソッド	現在の文字長を返すメソッドです。
canSeparate メソッド	現在の文字をセパレータの解析対象とするかどうかを返すメソッドです。

データ変換の対象が可変長文字列で，かつバイナリフォーマット定義にセパレータが設定されている場合，データ変換はセパレータの解析処理を実行します。CSCOwnCodeReader および CSCOwnCodeReaderContext の各メソッドの呼び出し順序を次の図に示します。

図 H-6　CSCOwnCodeReader および CSCOwnCodeReaderContext の各メソッドの呼び出し順序



注※
ユーザが実装するクラス間での値の受け渡しであるため，
インターフェースは規定していない。

1. インスタンス生成

データ変換によって CSCOwnCodeReader のインスタンスを生成します。

2. CSCOwnCodeReader#start メソッド

自スレッド専用の CSCOwnCodeReaderContext のインスタンスを生成します。以降，CSCOwnCodeReader との値の受け渡しは，自スレッドが保持する CSCOwnCodeReaderContext のインスタンスで実行します。

3. CSCOwnCodeReader#readChar メソッド

readChar の実行時，データ変換の処理は引数に CSCOwnCodeReaderContext のインスタンスを渡します。readChar の処理では，解析結果を CSCOwnCodeReaderContext のインスタンスにセットします。セットされた解析結果は，データ変換の処理でセパレータの解析に使用します。

4. getPosition メソッド，getLength メソッド，canSeparate メソッド

それぞれのメソッドは，データ変換から呼ばれます。メソッドの実行順序は電文フォーマットに依存します。

- getPosition メソッド
  このメソッドで現在の文字位置を返します。
- getLength メソッド
  このメソッドで現在の文字長を返します。
- canSeparate メソッド
  このメソッドで現在の文字をセパレータの解析対象とするかどうかを返します。

5. CSCOwnCodeReader#end メソッド

CSCOwnCodeReaderContext の実装クラスの解放処理が必要な場合は，end メソッドを実行します。

(a) getPosition メソッド

説明

readChar の結果が true の場合，現在の文字位置を返します。

readChar の結果が false の場合，読み込めるデータのサイズを返します。

形式

```
public int getPosition()
```

パラメタ

なし。

例外

なし。

戻り値

現在の文字位置，または読み込めるデータのサイズ（単位はバイト）です。

CSCOwnCodeReader#start で渡した offset の位置を 0 とします。

(b) getLength メソッド

説明

現在の文字長を返します。

形式

```
public int getLength()
```

パラメタ

なし。

例外

なし。

戻り値

現在の文字長を返します。

(c) canSeparate メソッド

説明

現在の文字をセパレータの解析対象とするかどうかを返します。false を返した場合，セパレータのバイト列と現在の文字が一致しても，セパレータと見なしません。

形式

```
public boolean canSeparate()
```

パラメタ

なし。

例外

なし。

戻り値

セパレータを解析する場合，true を返します。

セパレータを解析しない場合，false を返します。

(2) 例外クラス

文字コード変換 UOC の開発時に発生する例外クラスを次に示します。

クラス名

CSCOwnCodeConverterException クラス

説明

文字コード変換処理中に異常が発生した場合に送出する例外です。

この例外が発生した場合は，データ変換処理全体を中断します。

(3) 実装例（MS932）

CSCOwnCodeReaderContext インターフェースの実装例（MS932）を次に示します。

```
public class CSCOwnCodeReaderContextImpl implements CSCOwnCodeReaderContext {

    private final byte[] data ;

    private int position = 0 ;

    private int next = 0 ;

    private int length = 0 ;


    public CSCOwnCodeReaderContextImpl(
        final byte[] data ) {

        this.data = data ;
    }


    @Override
    public int getPosition() {

        return position ;
    }


    @Override
    public int getLength() {

        return length ;
    }


    @Override
    public boolean canSeparate() {

        // MS932はシフト（エスケープシーケンス）状態を持たず,
        // セパレータの出現に制限がないため, 常にtrueを返す
        return true ;
    }


    public byte[] getData() {

        return data ;
    }


    public void setPosition( int position ) {

        this.position = position ;
    }
```

```
    public void setLength( int length ) {

        this.length = length ;
    }


    public int getNextPosition() {

        return this.next ;
    }


    public void setNextPosition( int position ) {

        this.next = position ;
    }
}
```

(4) 実装例（IBM 漢字コード）

CSCOwnCodeReaderContext インターフェースの実装例（IBM 漢字コード）を次に示します。

```
public class CSCOwnCodeReaderContextImpl implements CSCOwnCodeReaderContext {

    private final byte[] data ;

    private int position = 0 ;

    private int length = 1 ;

    private int next = 0 ;

    private boolean canSeparate = true ;


    public CSCOwnCodeReaderContextImpl( final byte[] data ) {

        this.data = data ;
    }


    @Override
    public int getPosition() {

        return position ;
    }


    @Override
    public int getLength() {

        return length ;
    }


    @Override
```

```
    public boolean canSeparate() {

        return canSeparate ;
    }


    public byte[] getData() {

        return data ;
    }


    public void setPosition( int position ) {

        this.position = position ;
    }


    public void setLength( int length ) {

        this.length = length ;
    }


    public void setCanSeparate( boolean canSeparate ) {

        this.canSeparate = canSeparate ;
    }


    public int getNextPosition() {

        return this.next ;
    }


    public void setNextPosition( int position ) {

        this.next = position ;
    }
}
```

付録 H.6　注意事項

文字コード変換 UOC を利用した場合は，telegram-undefined-character-code の指定による未定義コード検出時の動作を変更する機能は有効になりません。未定義コード検出時の動作は，文字コード変換 UOC の実装に依存します。文字コード変換 UOC でも，未定義コード検出時の動作の指定が必要な場合は，文字コード変換 UOC 側で期待する動作となるように実装してください。

詳細については，マニュアル「Code Converter - Development Kit for Java」を参照してください。

付録 I　DB アダプタで取得したデータのフォーマットの変換例

DB アダプタで取得したデータは，列名をキーにして簡単に値を参照できません。そこで，列名が要素名となるフォーマットに変換することで，データを扱いやすくできます。

ここでは，DB アダプタで取得したデータを，列名が要素名となるフォーマットに変換する方法について例題を使って説明します。

付録 I.1　例題

例題で使用するデータベースの内容，実行する SQL の形式，SQL オペレーション定義ファイル，および変換後のフォーマットを次に示します。

データベースの内容

データベース参照名：DB_SERVER1

スキーマ名：DBA

テーブル名：受注表

テーブル構成：

受注番号（INTEGER）	得意先コード（CHAR）	商品コード（CHAR）	受注個数（INTEGER）
1	AA001	0001	5
2	AB002	0001	1
3	AA001	0102	3
4	XA005	0103	1
5	AA001	0105	1

実行する SQL の形式

```
OPERATION1：SELECT * FROM DBA.受注表 WHERE <val1> <val2> <val3>
```

SQL オペレーション定義ファイル

```
<?xml version="1.0" encoding="UTF-8" ?>
<DBadapter_SQL_OPERATION>
    <DATABASE_DATA>
        <DB_NAME>DB_SERVER1</DB_NAME>
        <DB_TYPE>HIRDB</DB_TYPE>
    </DATABASE_DATA>
    <SQL_DATA>
        <OPERATION1>
            SELECT * FROM DBA.受注表 WHERE <val1 dba_inf="column"/>
                <val2 dba_inf="preset"/>
                <val3 dba_inf="data" data_type="CHAR"/>
        </OPERATION1>
```

```
    </SQL_DATA>
</DBadapter_SQL_OPERATION>
```

SQL オペレーション定義ファイルの各項目については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.3.5　DB アダプタを定義する」の，SQL オペレーション定義ファイルの作成に関する説明を参照してください。また，DB アダプタで取得できるデータの形式については，マニュアル「サービスプラットフォーム 開発ガイド 受付・アダプタ定義編」の「3.3.5　DB アダプタを定義する」の，応答電文の形式に関する説明を参照してください。

変換後のフォーマットの XML スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.example.org/DBA_Sample1"
 xmlns:tns="http://www.example.org/DBA_Sample1"
 elementFormDefault="qualified">

  <complexType name="DBA_Sample1Type">
    <sequence>
      <element name="record" type="tns:RecordType"
               maxOccurs="unbounded" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="RecordType">
    <sequence>
      <element name="受注番号" type="int" maxOccurs="unbounded" minOccurs="0"/>
      <element name="得意先コード" type="string" maxOccurs="unbounded" minOccurs="0"/>
      <element name="商品コード" type="string" maxOccurs="unbounded" minOccurs="0"/>
      <element name="受注個数" type="int" maxOccurs="unbounded" minOccurs="0"/>
    </sequence>
  </complexType>

  <element name="受注表" type="tns:DBA_Sample1Type"/>
</schema>
```

付録 I.2　フォーマットの変換方法

DB アダプタで取得したデータのフォーマットの変換の流れを次に示します。

1. 変換元および変換先のデータの設定

   DB アダプタで取得したデータのフォーマットの変換には，データ変換定義を使用します。データのフォーマットの変換を始める前に，データ変換定義画面に変換元および変換先のデータを設定します。

2. 列名の cid 属性値の取得

   空白除去ファンクションを使用して，列名の ID（cid 属性値）を取得します。

3. くり返しファンクションの設定

   くり返しファンクションを設定して，DB アダプタで取得したすべての値を変換先に代入できるようにします。

4. 変換先の列名要素への値の代入

列名の ID と値の ID を比較して，ID が等しい場合に，変換先に値を代入するように設定します。

5. 定義した内容の検証

定義した内容を検証し，マッピングに矛盾がないことを確認します。

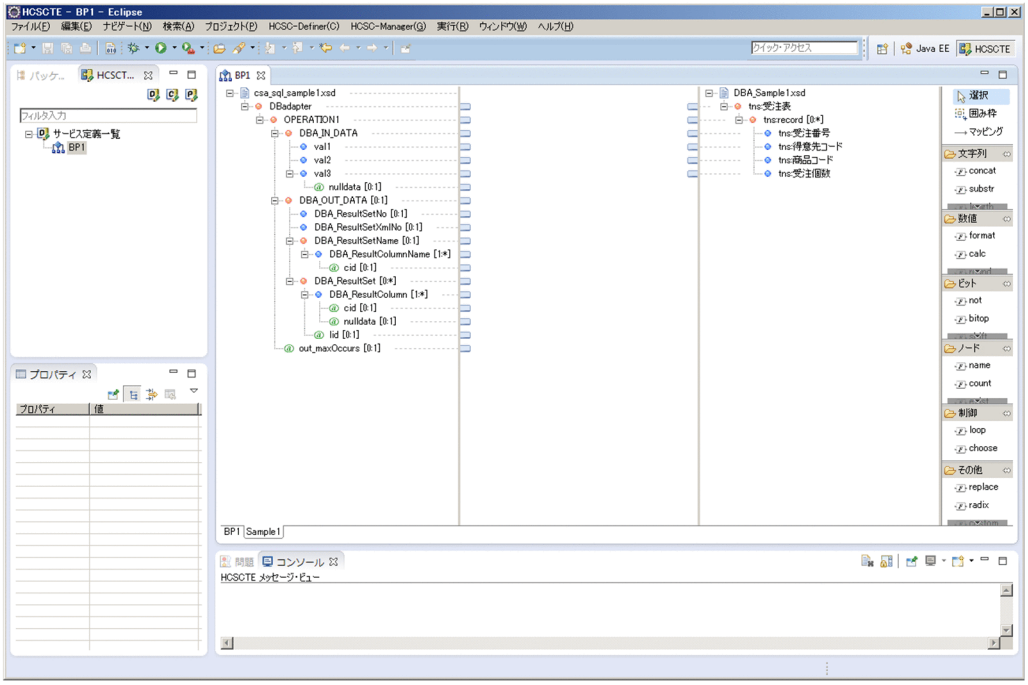
それぞれの手順について説明します。

なお，これらのフォーマット変換の操作は自動で行うこともできます。詳細は，「付録 J　DB アダプタで取得したデータの自動マッピング」を参照してください。

(1)　変換元および変換先のデータの設定

DB アダプタで取得したデータのフォーマットの変換には，データ変換定義を使用します。

データ変換定義画面の変換元スキーマツリービューアに DB アダプタで取得したデータ，変換先スキーマツリービューアに変換後のフォーマットの XML スキーマを設定してください。



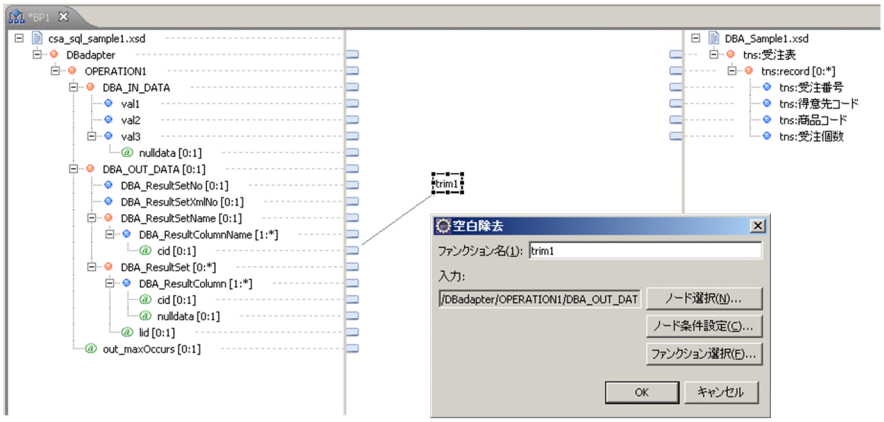
(2)　列名の cid 属性値の取得

応答電文の DBA_ResultSetName 要素には，列名とその ID（cid 属性）が対で格納されています。ここでは，空白除去ファンクションを使用して，ノード条件に指定した列名に対応する cid 属性を取得します。

列名の cid 属性値を取得する手順を次に示します。

1. DBA_ResultColumnName 要素の下の cid 属性に空白除去ファンクションを定義します。

空白除去ファンクションを定義する方法については，「7.5.5　文字列の空白を除去する」を参照してください。

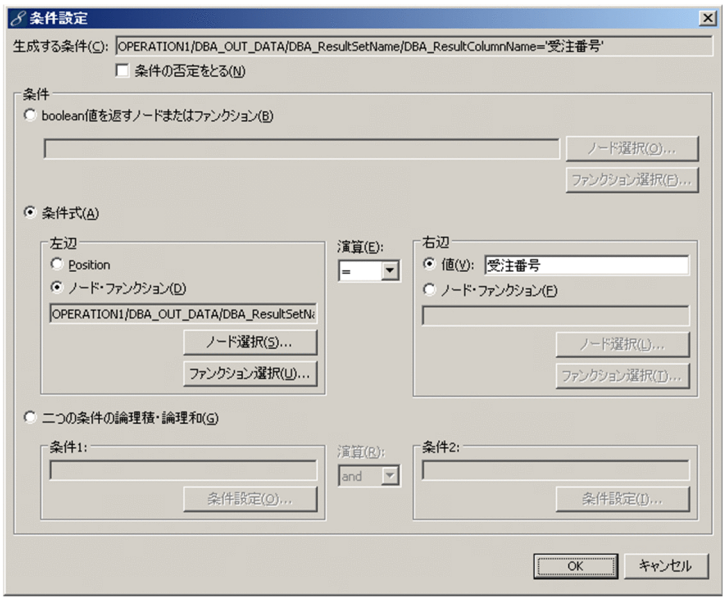


2. [空白除去] ダイアログで［ノード条件設定］ボタンをクリックします。
   ［ノード条件設定］ダイアログが表示されます。

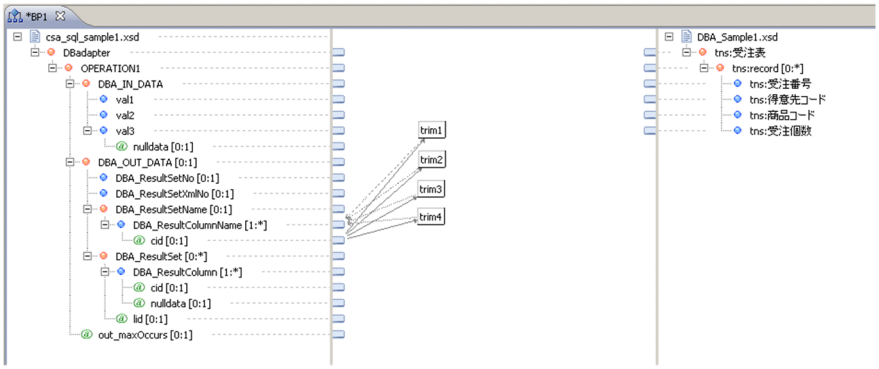
3. [ノード条件設定] ダイアログで［条件設定］ボタンをクリックします。
   ［条件設定］ダイアログが表示されます。

4. ノード条件を設定します。
   次のように設定してください。

   - 左辺：DBA_ResultColumnName 要素
   - 右辺：列名（ここでは［受注番号］）

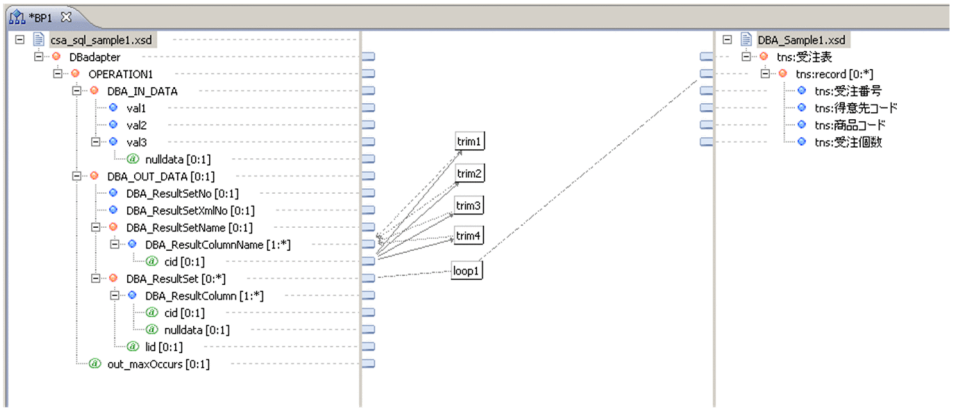


5. 「得意先コード」「商品コード」「受注個数」の各要素についても，同様に定義します。



(3)　くり返しファンクションの設定

変換元の DBA_ResultSet 要素から，変換先の record 要素にくり返しファンクションをマッピングします。くり返しファンクションを定義する方法については，「7.5.16　くり返しを対応づける」を参照してください。

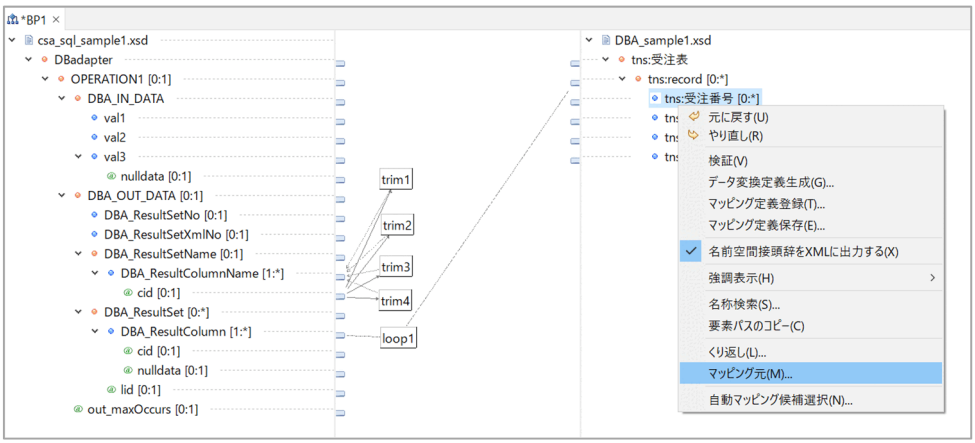


(4)　変換先の列名要素への値の代入

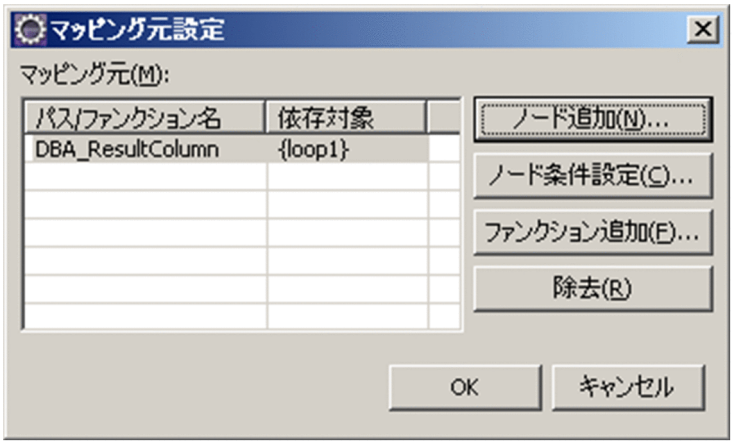
変換先の各行名要素に値を代入します。このとき，DBA_ResultColumn 要素の下の cid 属性と「(2)　列名の cid 属性値の取得」で定義した空白除去ファンクションが一致するように条件を設定します。

変換先の各行名要素に値を代入する手順を次に示します。

1. 変換先の要素（ここでは「受注番号」）を右クリックし，［マッピング元］を選択します。
   ［マッピング元設定］ダイアログが表示されます。



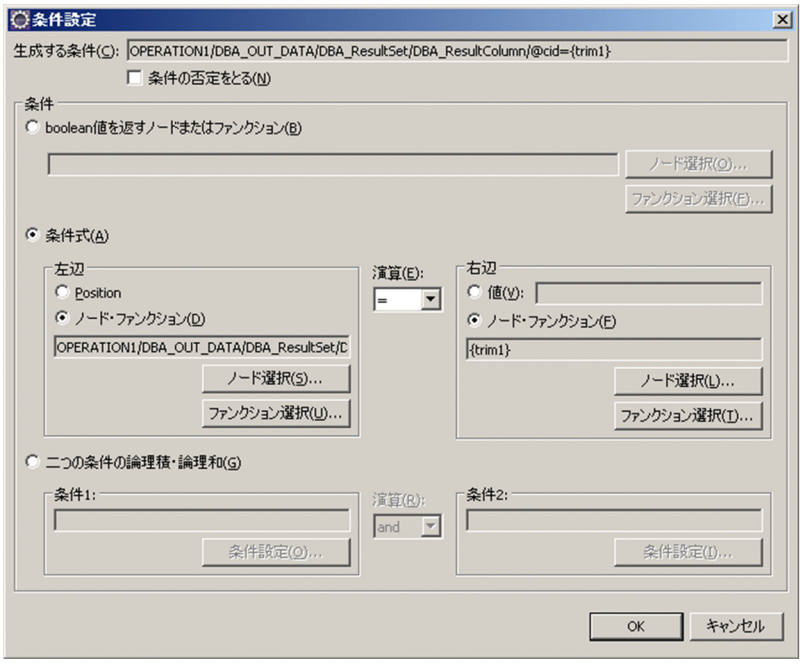
2. [マッピング元] に DBA_ResultColumn を選択します。



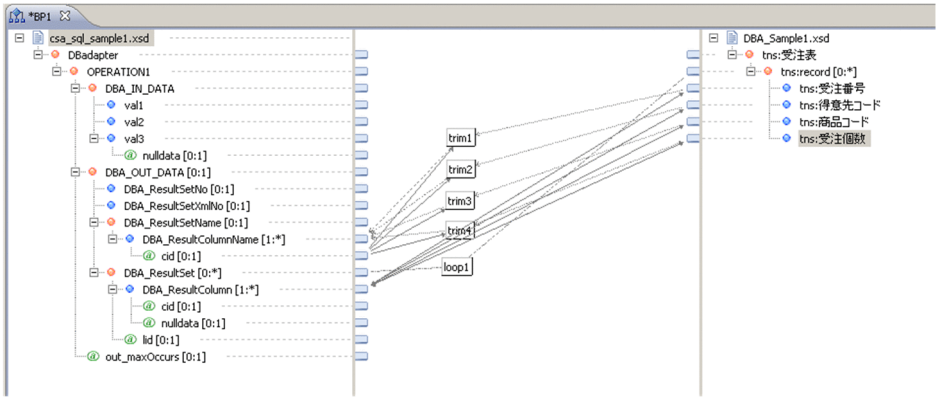
3. [マッピング元設定] ダイアログで［ノード条件設定］ボタンをクリックします。
   ［ノード条件設定］ダイアログが表示されます。

4. [ノード条件設定] ダイアログで［条件設定］ボタンをクリックします。
   ［条件設定］ダイアログが表示されます。

5. ノード条件を設定します。
   次のように設定してください。
   - 左辺：DBA_ResultColumn 要素の下の cid 属性
   - 右辺：変換先の列名に対応するように定義した空白除去ファンクション



6.「得意先コード」「商品コード」「受注個数」の各要素についても，同様に定義します。



(5)　定義した内容の検証

定義したマッピングに矛盾がないことを検証します。検証方法については，「5.11.2　検証方法」を参照してください。

付録 J　DB アダプタで取得したデータの自動マッピング

ここでは，DB アダプタで取得したデータを，列名が要素名となるフォーマットに自動でマッピングする方法について説明します。

この方法は「付録 I　DB アダプタで取得したデータのフォーマットの変換例」を自動で行うためのものです。ただし，定義時には次の点に注意してください。

- 「付録 I　DB アダプタで取得したデータのフォーマットの変換例」では cid 属性をキーにしてマッピングしていますが，この方法では，変換元列番号をキーにして自動マッピングしています。
- くり返しファンクションは自動で設定されないため，手動で設定する必要があります。

マッピングの流れを次に示します。

1. SQL オペレーション定義ファイル，要求電文，応答電文を基にカラム定義ファイルを作成します。
2. 作成したカラム定義ファイルを利用してデータ変換定義画面でマッピングします。

それぞれの作業について次に示します。

付録 J.1　カラム定義ファイルの作成

カラム定義ファイルとは，DB アダプタの検索結果の Position()[変換元列番号]と XML スキーマの変換先要素名を関連づける CSV 形式のファイルです。

(1)　形式

カラム定義ファイルの形式を次に示します。

```
変換元列番号，変換先要素名[改行]
      :
```

カラム定義ファイルの構文を次に示します。

表 J-1　カラム定義ファイルの構文

項目	カラム定義ファイルの構文
カラム定義ファイル	(行データ 改行コード)*
行データ	空白* [変換元列番号 空白*] [',' 空白* 変換先要素名 空白*]
改行コード	'¥r¥n' | '¥r' | '¥n'
空白	' ' | '¥t'
変換元列番号	('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9') +

項目	カラム定義ファイルの構文
変換先要素名	(MS932 文字 - (',' | ':' | 空白 | 改行コード))+

(2) 定義例

「付録 I.1 例題」のデータベースの内容を例に，次のテーブル構成のデータを DB アダプタで取得する場合の，カラム定義ファイルの定義例について説明します。

表 J-2 テーブル構成（先頭行は列名を表す）

受注番号（INTEGER）	得意先コード（CHAR）	商品コード（CHAR）	受注個数（INTEGER）
1	AA001	0001	5
2	AB002	0001	1
3	AA001	0102	3
4	XA005	0103	1
5	AA001	0105	1

このテーブル構成のデータは，DB アダプタから次のように取得されます。

表 J-3 DB アダプタから取得したデータ（先頭行はデータの取得順（列番号）を表す）

1	2	3	4
1	AA001	0001	5
2	AB002	0001	1
3	AA001	0102	3
4	XA005	0103	1
5	AA001	0105	1

このように，DB アダプタから取得したデータは，列名をキーにして値を参照できないため，列名が要素名となるフォーマットに変換する必要があります。

そのため，カラム定義ファイルを使用して，DB アダプタから取得されたデータのどの列番号をどの要素名にマッピングするのかを定義します。

カラム定義ファイルの定義例

```
1,受注番号
2,得意先コード
3,商品コード
4,受注個数
```

このカラム定義ファイルの 1 行目は，DB アダプタが 1 列目のデータを［受注番号］にマッピングすることを表します。

このカラム定義ファイルを使用して自動マッピングすることで，DB アダプタから取得されたデータは次のように変換され，要素名（列名）をキーにして値を参照できるようになります。

表 J-4　テーブル構成（先頭行は列名を表す）

受注番号	得意先コード	商品コード	受注個数
1	AA001	0001	5
2	AB002	0001	1
3	AA001	0102	3
4	XA005	0103	1
5	AA001	0105	1

(3)　注意事項

カラム定義ファイルを作成するときの留意点について説明します。

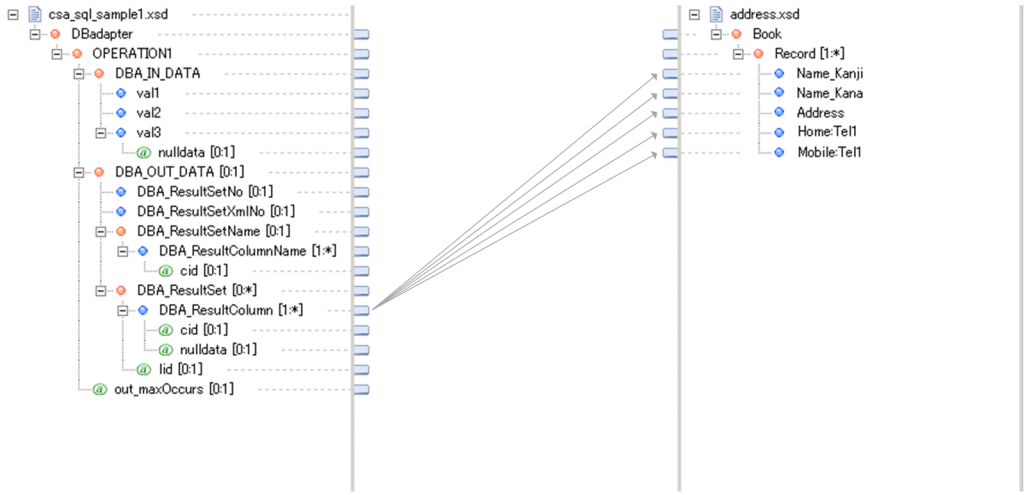
- ファイル名および拡張子は任意です。
- 使用できる文字コードは MS932 です。
- 変換先要素名は大文字・小文字が区別されます。
- 変換元列番号，変換先要素名，または両方を省略した場合，その行の定義は無視され，警告メッセージが出力されます。
- 変換元列番号の文字列の長さは最大 1,024 文字です。
- 同一の変換先要素名を重複して定義した場合，先に定義した要素は無視されます。このとき警告メッセージが出力されます。
- カラム定義ファイルを利用した自動マッピングでは，カラム定義ファイルの変換先要素名だけで評価され，名前空間は区別されません。

  例えば，カラム定義ファイルに次のように定義した場合，要素名が「Home:Tel1」「Mobile:Tel1」と重複しているスキーマにマッピングすると，図のように両方の要素にマッピングされます。

  カラム定義ファイルの内容

```
1,Name_Kanji
2,Name_Kana
3,Address
4,Tel1
```

図 J-1　名前空間が異なる同一要素名がある場合のマッピングの例



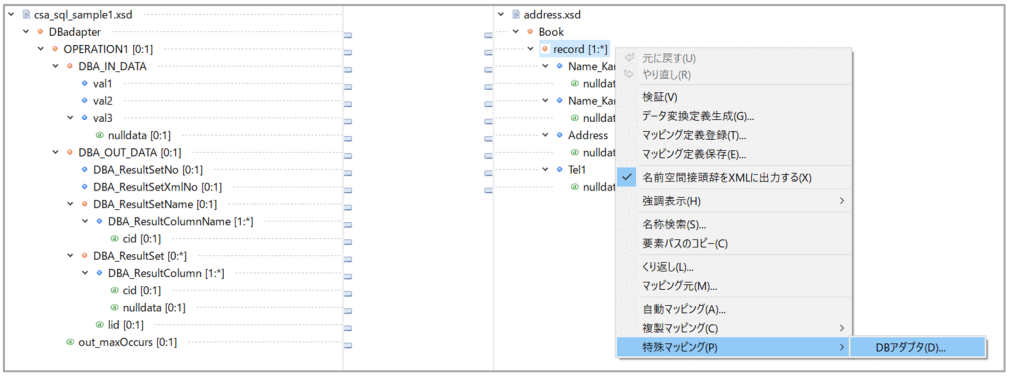
付録 J.2　データ変換定義画面でのマッピング

列名が要素名となるフォーマットに自動でマッピングするには，データ変換定義を使用します。

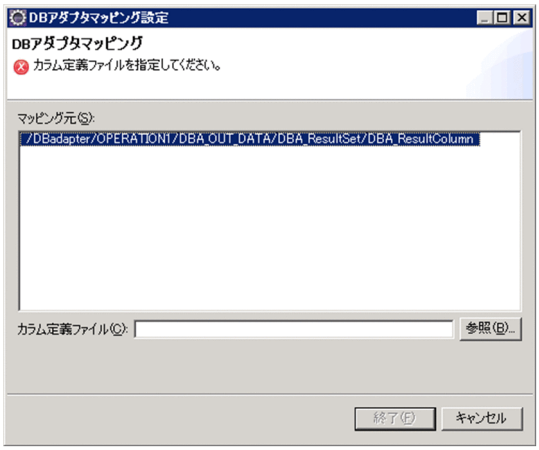
データ変換定義画面の変換元スキーマツリービューアに DB アダプタで取得したデータ，変換先スキーマツリービューアに変換後のフォーマットの XML スキーマを設定してください。

マッピングの方法を次に示します。

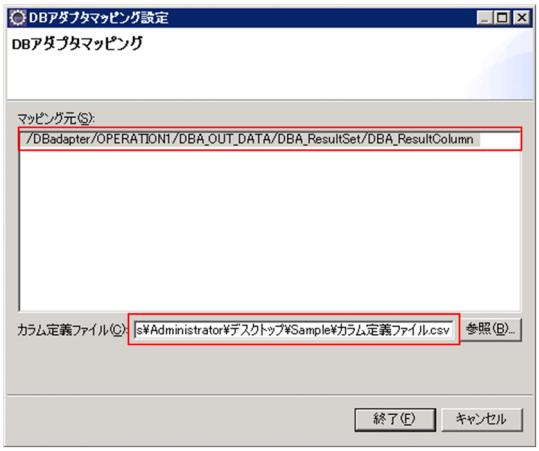
1. 変換先のマッピング対象とする要素の親要素（複合内容要素）を選択します。
2. 親要素を選択した状態で，右クリックします。
   ポップアップメニューが表示されます。



3. ポップアップメニューから［特殊マッピング］－［DB アダプタ］を選択します。
   DB アダプタマッピング設定ダイアログが表示されます。

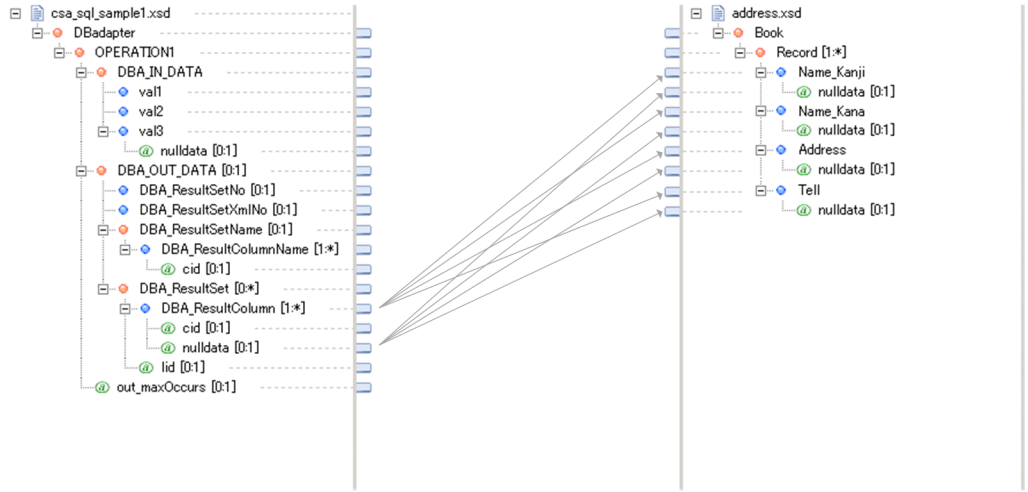


4. マッピング元を選択し，カラム定義ファイルを指定します。



5. [終了] ボタンをクリックします。

自動でマッピングされます。



ポップアップメニューで［DB アダプタ］を選択したノード配下の要素名とカラム定義ファイルの変換先要素名が一致しない，または一致する要素がマッピング対象でない場合，警告メッセージが出力されます。

6. 変換元要素から変換先要素にくり返しファンクションをマッピングします。

くり返しファンクションを定義する方法については，「7.5.16　くり返しを対応づける」を参照してください。

付録 K　外部バインディングファイルを利用した WSDL のカスタマイズ

外部バインディングファイルを利用して，WSDL ファイルの解析方法をカスタマイズできます。外部バインディングファイルを利用した WSDL のカスタマイズの流れを次に示します。

1. 外部バインディングファイルを作成します。

   ファイル名は，任意です。

   外部バインディングファイルの作成例については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」の「15.2　WSDL から Java へのマッピングのカスタマイズ」を参照してください。

2. 作成した外部バインディングファイルを WSDL カスタムバインディング設定ページで設定します。

   WSDL カスタムバインディング設定ページの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.9.1　設定ダイアログ（WSDL カスタムバインディングファイルの設定)」を参照してください。

   09-00 以前で外部バインディングファイルを使用しており，上書きインストールで 09-50 以降にバージョンアップした場合，WSDL カスタムバインディング設定ページで，使用していた外部バインディングファイル（<サービスプラットフォームのインストールディレクトリ>¥CSCTE¥config¥tools¥custom¥custombinding.xml）を指定してください。

格納した外部バインディングファイルを基に，WSDL ファイルを解析します。カスタマイズした WSDL ファイルは，サービスアダプタやユーザ定義受付を作成するときに使用できます。

付録 L　IBM 漢字コードの文字セットの変更

バイナリフォーマット定義ファイルで文字コードとして IBM 漢字コードを選択する場合，デフォルトでは次のとおり設定されます。

表 L-1　文字コードに IBM 漢字コードを選択した場合のデフォルト

文字コード種別	コード体系
IBM_CODE+EBCDIC(LATIN)	IBM 漢字コード＋標準英小文字セット
IBM_CODE+EBCDIC(KANA)	IBM 漢字コード＋標準カタカナ文字セット

バイナリフォーマット定義ファイルの文字コードを設定するときは，［フォーマット］ダイアログを使用します。［フォーマット］ダイアログの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.3.1　フォーマットダイアログ」を参照してください。

文字コードの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「付録 A　文字コード対応表」を参照してください。

付録 L.1　IBM 漢字コードの文字セットを変更する手順

HCSC サーバを停止させた状態で，次に示すコピー元のディレクトリに格納されたテーブルファイルをすべてコピー先のディレクトリに上書きしてください。

表 L-2　IBM 漢字コードの変更

変更内容	コピー元	上書き先
標準文字セットから拡張文字セットに変更する（Windows の場合）	<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥table¥ibmE	<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥table
拡張文字セットから標準文字セットに変更する（Windows の場合）	<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥table¥ibmB	<サービスプラットフォームのインストールディレクトリ>¥CSC¥lib¥external¥table
標準文字セットから拡張文字セットに変更する（UNIX の場合）	/opt/Cosminexus/CSC/lib/external/table/ibmE	/opt/Cosminexus/CSC/lib/external/table
拡張文字セットから標準文字セットに変更する（UNIX の場合）	/opt/Cosminexus/CSC/lib/external/table/ibmB	/opt/Cosminexus/CSC/lib/external/table

付録 M　サイズノードの機能拡張

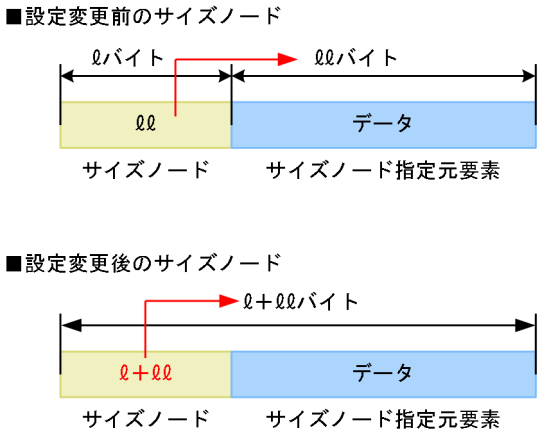
サイズノードは，単純内容要素および複合内容要素の可変長のデータサイズを指定する要素です。ここでは，サイズノードに指定する値を変更する方法について説明します。

付録 M.1　サイズノードにサイズノード自身のサイズを含めた値を指定する

サイズノードの値は，サイズノード指定元要素の長さを示す値で，サイズノード自身の長さを含みません。サイズノードにサイズノード自身の長さを含むようにするには，バイナリフォーマット定義ファイル，または J2EE サーバ用ユーザプロパティファイル（usrconf.properties）で設定を変更する必要があります。

設定変更前と設定変更後のサイズノードの値の取り扱いの差異を次に示します。

図 M–1　設定変更前後のサイズノードの値の違い



(1)　サイズノードの設定を変更できるバイナリフォーマット定義

次のように定義された要素に対して，サイズノードの値にサイズノード自身のサイズを含める設定ができます。

表 M–1　サイズノードの設定を変更できるバイナリフォーマット定義

対象	条件
サイズノード指定元要素	単純内容要素でかつデータ種別が次のどれかである。 • 文字列 • 整数 • 実数 • 固定小数部数値 • ゾーン形式数値 • パック形式数値 • バイト列 • 日付時刻

対象	条件
サイズノード指定元要素	複合内容要素である。
サイズノードに指定する要素	数値型の単純内容要素でサイズが固定長である。

(2) 設定方法

J2EE サーバ用ユーザプロパティファイル，バイナリフォーマット定義ファイルによる設定，および COBOL2FDX コンバータによる設定を次に示します。

(a) J2EE サーバ用ユーザプロパティファイルによる設定

サイズノードの値にサイズノード自身のサイズを含めるどうかを J2EE サーバ用ユーザプロパティファイル（usrconf.properties）で設定できます。

J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の設定は，バイナリフォーマット定義エディタの［構成要素］タブにある［サイズ算出方法］で［システム既定値］が指定された場合に有効になります。

●usrconf.properties の格納先

usrconf.properties の格納先を次に示します。

Windows の場合

```
<サービスプラットフォームのインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥<サーバ名称>¥usrconf.properties
```

UNIX の場合

```
/opt/Cosminexus/CC/server/usrconf/ejb/<サーバ名称>/usrconf.properties
```

usrconf.properties の詳細は，次のマニュアルを参照してください。

- J2EE サーバの互換モードが推奨モードの場合

  マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3 usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」
- J2EE サーバの互換モードが V9 互換モードの場合

  マニュアル「アプリケーションサーバ 機能解説 互換編」の「10.1.1　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」

●usrconf.properties に記述する内容

usrconf.properties には次の形式で記述します。

```
csc.dt.sizeNode.includeSelfSize=<値>
```

指定できる値は次のとおりです。大文字と小文字は区別されません。

true

　サイズノードの値にサイズノード自身のサイズを含みます。

false

サイズノードの値にサイズノード自身のサイズを含みません。

●usrconf.properties の記述に関する注意事項

次の場合は，システム既定値に従います。

- 値に true または false 以外の値を指定した場合
- プロパティ名が誤っている場合
- プロパティが記述されていない場合

(b)　バイナリフォーマット定義による設定

サイズノードの値にサイズノード自身のサイズを含めるどうかを，バイナリフォーマット定義ファイルの要素単位で指定できます。バイナリフォーマット定義による設定は，J2EE サーバ用ユーザプロパティファイルの設定よりも優先されます。

●サイズノードのサイズの算出方法（単純内容要素の場合）

バイナリフォーマット定義エディタの［単純内容要素］ダイアログで定義（［構成要素］タブの［サイズ算出方法］で選択）します。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

●サイズノードのサイズの算出方法（複合内容要素の場合）

バイナリフォーマット定義エディタの［複合内容要素］ダイアログで定義（［構成要素］タブの［サイズ算出方法］で選択）します。

［複合内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.3　複合内容要素ダイアログ」を参照してください。

注意事項

サイズ算出方法の設定を変更していないバイナリフォーマット定義を利用する場合は，サイズ算出方法はシステム既定値と仮定され，J2EE サーバ用ユーザプロパティファイルの設定に従います。

(c)　COBOL2FDX コンバータによる設定

COBOL2FDX コンバータでサイズ算出方法を指定する場合，SIZE 句に CALC 句を設定してください。

COBOL2FDX コンバータ，および COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

付録 M.2　サイズノードが表すデータサイズの単位をバイト数から桁数へ変更する

バイナリフォーマット定義では，単純内容要素のサイズの指定方法として，バイト単位のデータサイズを設定したサイズノードを使用することができます。サイズノード値の桁数指定オプションを使用することによって，サイズノードが表すデータサイズの単位を，データの桁数に変更することができます。なお，このオプションは，パック形式数値の単純内容要素にだけ有効です。

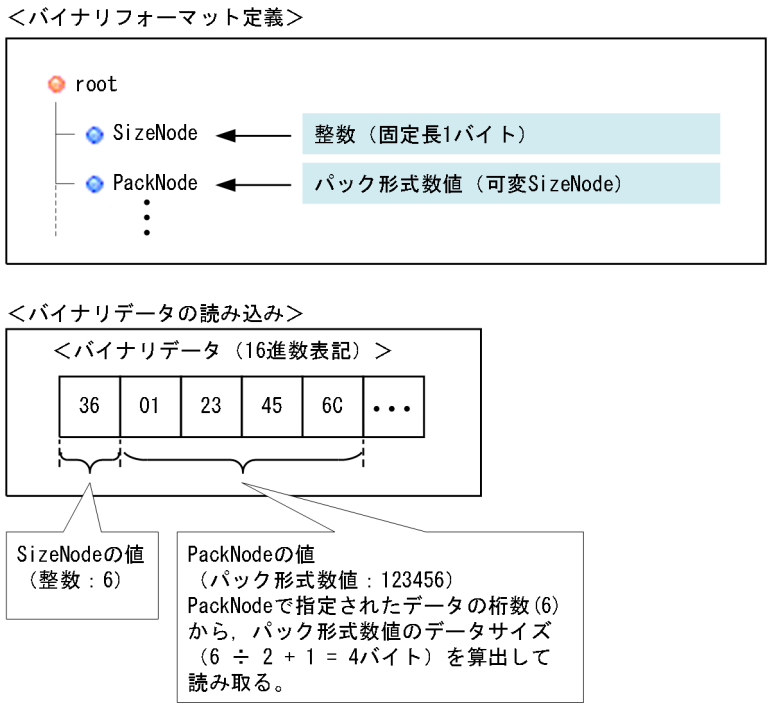
(1)　サイズノード値の桁数指定オプションの使用例

サイズノード値の桁数指定オプションを使用した場合，サイズノードで指定されたデータの桁数から，パック形式数値型要素の実際のバイトサイズを算出し，バイナリデータを読み取ります。バイトサイズは，次の計算式に従って算出されます。

```
（桁数÷2（商の小数点以下は切り捨て））＋1
```

サイズノード値の桁数指定オプションの使用例を次に示します。

図 M-2　サイズノード値の桁数指定オプションの使用例



参考

データに小数値が含まれる場合，次の規則に従ってデータの桁数を算出します。

- 整数部分と小数部分の和でデータの桁数を解釈します。
- 整数部分が 0 の場合，桁数に含めません。

- 小数部分が 0 の場合，バイナリフォーマット定義の単純内容要素の小数部の数だけ桁数に含めます。

入力値と出力結果のバイナリ（パック形式数値），およびデータの桁数の算出例を次の表に示します。

入力値	出力結果のバイナリ（パック形式数値）	データの桁数
"0.2"	0x2C（1）※	1
"0.02"	0x002C（2）※	2
"-0.02"	0x002D（2）※	2
"0.0"	0x0C（1）※	1
"0.00"	0x000C（2）※	2
"10.2"	0x102C（1）※	3
"10.02"	0x01002C（2）※	4

注※

丸括弧「()」内の数値は，バイナリフォーマット定義の小数部桁数を表します。

(2)　サイズノード値の桁数指定オプションの設定方法

サイズノード値の桁数指定オプションは，バイナリフォーマット定義にサイズノード値の単位を設定することによって使用できます。

バイナリフォーマット定義エディタによる設定，および COBOL2FDX コンバータによる設定を次に示します。

(a)　バイナリフォーマット定義エディタによる設定

サイズノード値の桁数指定オプションは，［単純内容要素］ダイアログの［構成要素］タブから［サイズノード値の単位］コンボボックスで「桁数」を選択してください。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

注意事項

［サイズノード値の単位］で［桁数］を指定している場合は，サイズノードの値にサイズノード自身のサイズを含めることができません。そのため，バイナリフォーマット定義エディタでは，［サイズノード値の単位］で［桁数］を指定している場合，［構成要素］タブの［サイズ算出方法］のコンボボックスが非活性となります。

(b)　COBOL2FDX コンバータによる設定

サイズノード値の桁数指定オプションは，VALUE-UNIT 句で指定します。パック形式数値型の基本項目に対し，VALUE-UNIT 句で DIGITS を指定してください。

サイズノード値の桁数指定オプションを使用する場合の COBOL 登録集原文の定義例を次に示します。

```
01      Root.
  03    SizeNode  INTEGER SIZE 1.
  03    PackNode  PACK SIZE DEPENDING ON SizeNode VALUE-UNIT DIGITS.
```

COBOL2FDX コンバータ，および COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

注意事項

サイズノード値の単位で桁数を指定している場合は，サイズノードの値にサイズノード自身のサイズを含めることができません。そのため，COBOL2FDX コンバータでは，サイズノード値の単位で桁数を指定している基本項目に対し CALC 句を指定すると，COBOL 登録集原文ファイルの変換でエラー（KECT26019-E）が発生します。

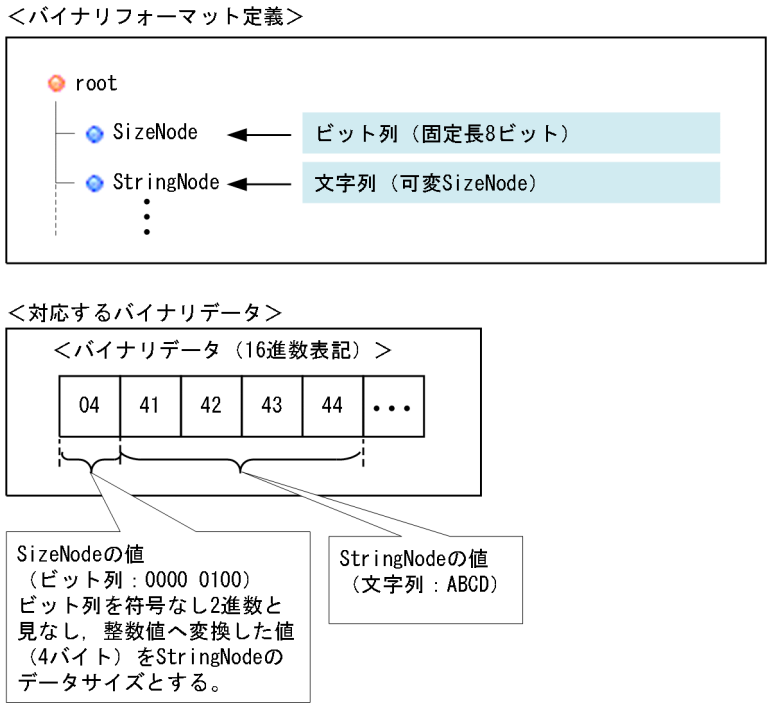
付録 M.3　サイズノードにビット列型のノードを指定する

サイズノードとしてビット列型の単純内容要素を指定することができます。ビット列型サイズノードでは，ビット列を符号なし 2 進数と見なし，整数値へ変換した値を参照元要素のデータサイズとして使用します。

(1)　ビット列型サイズノードの使用例

ビット列型サイズノードの使用例を次に示します。

図 M-3　ビット列型サイズノードの使用例



(2)　ビット列型サイズノードの指定方法

バイナリフォーマット定義エディタによる設定，および COBOL2FDX コンバータによる設定を次に示します。

(a)　バイナリフォーマット定義エディタによる設定

単純内容要素，および複合内容要素にビット列型サイズノードを指定する場合は，［ノード選択］ダイアログでビット列型の単純内容要素を選択してください。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

［複合内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.3　複合内容要素ダイアログ」を参照してください。

注意事項

ビット列型サイズノードの場合は，サイズノードの値にサイズノード自身のサイズを含めることができません。そのため，バイナリフォーマット定義エディタで，ビット列型サイズノードを指定している場合，［構成要素］タブの［サイズ算出方法］のコンボボックスが非活性となります。

(b)　COBOL2FDX コンバータによる設定

COBOL 登録集原文ファイルにおいて，ビット列型の基本項目が定義できます。ビット列型の基本項目は，BIT 句を使用して定義してください。

基本項目，および集団項目にビット列型のサイズノードを指定する場合は，SIZE 句，および DEPENDING ON 句を使用します。

ビット列型サイズノードを使用する場合の COBOL 登録集原文の定義例を次に示します。

```
01      Root.
  03    SizeNode BIT SIZE 8.
  03    StringNode STRING SIZE DEPENDING ON SizeNode.
```

選択条件ノードにビット列型の要素を定義する選択構成を使用した COBOL 登録集原文の定義例を示します。

```
01      Root.
  03    SelectNode BIT SIZE 8.
  03    ComplexNode CHOOSE DEPENDING ON SelectNode.
    05  StringNode STRING SIZE 16 CASE '00'.
    05  PackNode PACK SIZE 4 CASE '01'.
```

BIT 句，SIZE 句，および DEPENDING ON 句の使用方法，COBOL2FDX コンバータ，および COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

注意事項

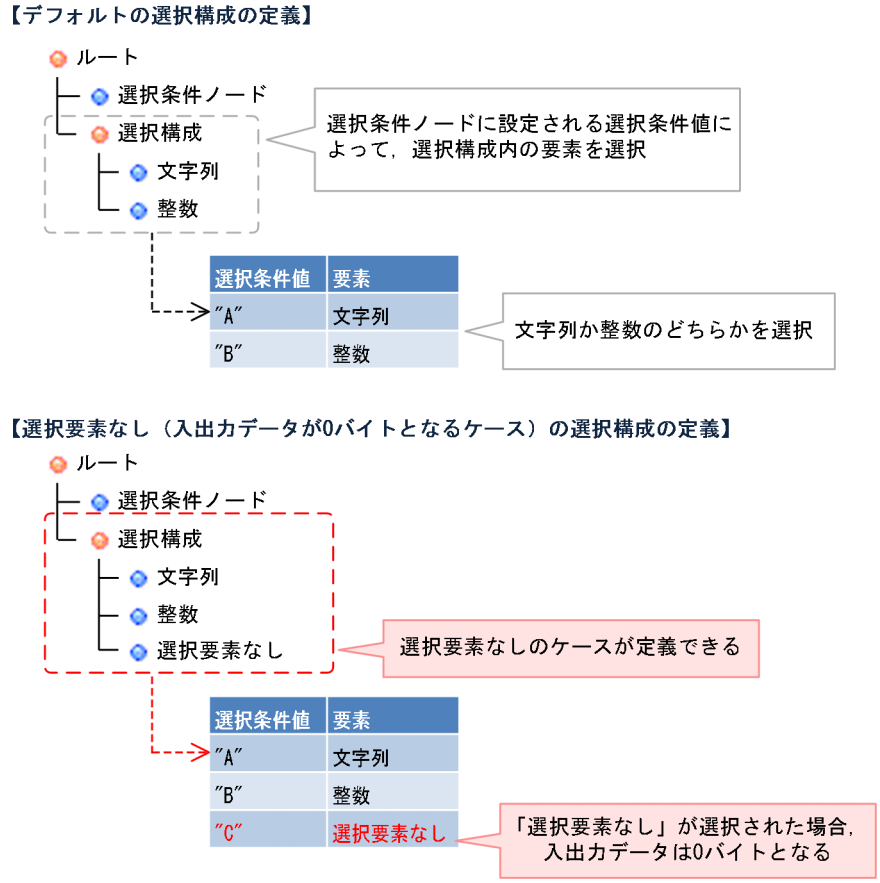
ビット列型サイズノードの場合は，サイズノードの値にサイズノード自身のサイズを含めることができません。そのため，COBOL2FDX コンバータでは，ビット列型サイズノードを指定する基本項目，および集団項目に対し CALC 句を指定すると，COBOL 登録集原文ファイルの変換でエラー（KECT26018-E）が発生します。

付録 N　選択要素なし（入出力データが 0 バイトとなるケース）の選択構成の定義

バイナリフォーマット定義では，選択構成である複合内容要素の構成要素に，文字列型でサイズの設定が 0 バイトである単純内容要素（サイズ 0 要素）を使用することで，選択要素なしのケースを定義できます。この場合，入出力データが 0 バイト（＝データが存在しない状態）となるため，選択要素が存在しない状態を表現できます。

選択要素なしのケース（入出力データが 0 バイトとなるケース）の定義例を，次に示します。

図 N-1　選択要素なしのケース（入出力データが 0 バイトとなるケース）の定義例



付録 N.1　前提条件

サイズ 0 要素は，次の条件を両方満たしている場合に定義できます。

- データ種別が文字列型でローカル単純内容要素である
- 親要素が選択構成である

付録 N.2　設定方法

バイナリフォーマット定義エディタによる設定，およびCOBOL2FDXコンバータによる設定を次に示します。

(1)　バイナリフォーマット定義エディタによる設定

選択構成である複合内容要素の構成要素にサイズ0要素を定義します。

サイズ0要素を定義する手順を次に示します。

1. [ローカル単純内容要素の追加]で［単純内容要素］ダイアログを表示します。
2. [データ種別]に「文字列型」を選択します。
3. [サイズ]のテキストボックスに「0」を指定します。

要素の定義方法については，「4.4.4　要素を定義する（CSV以外の形式の場合）」を参照してください。

注意事項

- サイズ0要素は，逐次構成では定義できません。

  逐次構成である親要素の構成要素として定義した場合，バイナリフォーマット定義の検証でエラー（KECT24040-E）となります。バイナリフォーマット定義の検証の詳細は，「4.4.7　バイナリフォーマット定義ファイルを検証する」を参照してください。
- サイズ0要素は，サイズ以外の項目（「個別文字コード」や「埋め字」など）を設定できません。

  ［単純内容要素］ダイアログで，サイズに0バイトが設定されている場合，［データ種別］および［サイズ］以外の設定項目が非活性となります。また，［構成要素］タブの設定項目が非活性となります。

  ［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。
- サイズ0要素を，選択条件ノードとして使用できません。

  次のダイアログで，選択条件ノードにサイズ0要素を選択した場合，ダイアログの上部にエラーメッセージが出力され，［OK］ボタンが非活性となります。

  ・［ノード選択］ダイアログ

  ・［複合内容要素］ダイアログ

  ・［構成要素］ダイアログ

  ・［条件値設定］ダイアログ

  また，選択条件ノードにサイズ0要素を選択した状態でバイナリフォーマット定義の検証を実行した場合，エラー（KECT24039-E）が発生します。

  各ダイアログの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「1.3　バイナリフォーマット定義に関連するダイアログ」を参照してください。

- サイズ 0 要素は，CSV 形式のバイナリフォーマット定義で使用することができません。

  バイナリフォーマット定義エディタで，バイナリフォーマット定義ファイルの新規作成時に「CSV 形式」を選択した場合，［単純内容要素］ダイアログで［サイズ］のテキストボックスに「0」を指定できません。

(2)　COBOL2FDX コンバータによる設定

CHOOSE 句（選択構成）が指定されている集団項目の構成要素にサイズ 0 要素を定義します。サイズ 0 要素の定義は，STRING 句を指定した基本項目に SIZE 句を使用して「0」を指定してください。

選択条件ノードにサイズ 0 要素を定義する選択構成を使用した COBOL 登録集原文の定義例を記載します。

```
01      ルート CHARSET 'MS932'.
  03    選択条件ノード STRING SIZE 1.
  03    選択構成 CHOOSE DEPENDING ON 選択条件ノード.
    05  文字列 STRING SIZE 1 CASE 'A'.
    05  整数 INTEGER SIZE 1 CASE 'B'.
    05  サイズ0 STRING SIZE 0 CASE 'C'.
```

STRING 句および SIZE 句の使用方法，COBOL2FDX コンバータおよび COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

注意事項

- サイズ 0 要素は，逐次構成では定義できません。

  逐次構成である親要素の構成要素として定義した場合，バイナリフォーマット定義の検証でエラー（KECT24040-E）となります。バイナリフォーマット定義の検証の詳細は，「4.4.7　バイナリフォーマット定義ファイルを検証する」を参照してください。
- サイズ 0 要素は，サイズ以外の項目（「個別文字コード」や「埋め字」など）を設定できません。

  SIZE 句に「0」が指定されている文字列型の基本項目について，選択条件値（CASE 句）以外の項目が設定されていた場合，COBOL 登録集原文ファイルの変換でエラー（KECT26020-E）が発生します。
- サイズ 0 要素を，選択条件ノードとして使用できません。

  COBOL 登録集原文ファイルで，選択条件ノードにサイズ 0 要素を選択した場合，バイナリフォーマット定義生成時の検証でエラー（KECT24039-E）が発生します。

付録 N.3　注意事項

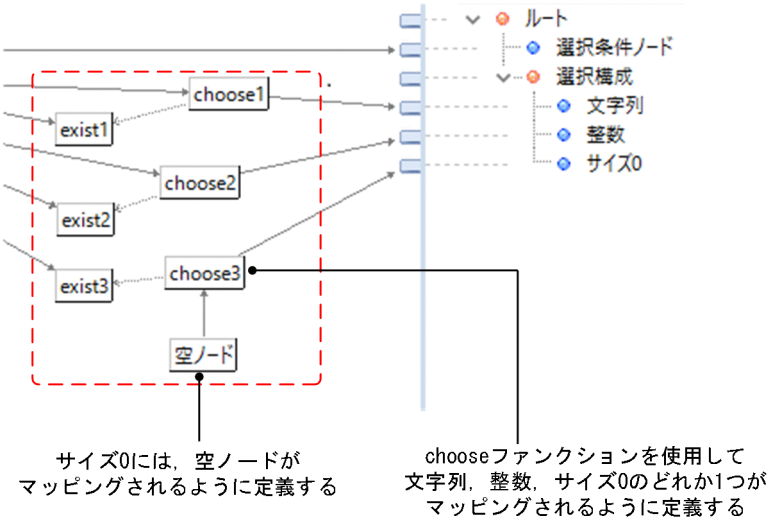
選択構成の機能拡張を使用する場合の注意事項について説明します。

(1)　マッピング先にサイズ 0 要素を使用する場合について

マッピング先にサイズ 0 要素を使用する場合，サイズ 0 要素には空ノードをマッピングしてください。データ変換時に空ノード以外の値がマッピングされた場合は，エラー（KDEC40477-E）が発生します。また，何もマッピングされなかった場合もエラー（KDEC40436-E）が発生します。

マッピング定義の例を次に示します。

図 N-2　マッピング先にサイズ 0 要素を使用した場合の定義例



(2)　CSV 形式のセパレータとの併用について

祖先ノードのセパレータ形式に［CSV］を選択した場合，サイズ 0 要素の選択構成は定義できません。サイズ 0 要素の選択構成を定義しようとした場合，動作は保障できません。

(3)　サイズ 0 要素が出現回数の設定によって繰り返し出現する場合について

次に示す条件がすべて当てはまる場合，データ変換時にエラーとなります。

1. サイズ 0 要素が入力データの末尾に存在する。
2. 1.の条件のサイズ 0 要素の祖先ノードで，出現回数の設定が次のどちらかである。
   - 2 回以上の範囲指定
   - データの終端まで（データの省略不可）
3. 選択構成で 1.の条件のサイズ 0 要素が選択される。

上記の条件でエラーとなった場合，次のエラーメッセージが出力されます。

- 出現回数の設定が 2 回以上の範囲指定：KDEC40702-E
- 出現回数の設定がデータの終端まで（データの省略不可）：KDEC40454-E

エラーになるバイナリフォーマット定義の例を次に示します。

図 N-3　エラーになるバイナリフォーマット定義の例（出現回数が範囲指定の場合）

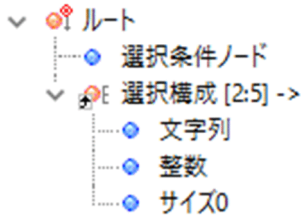
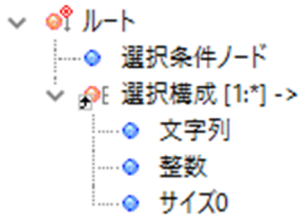


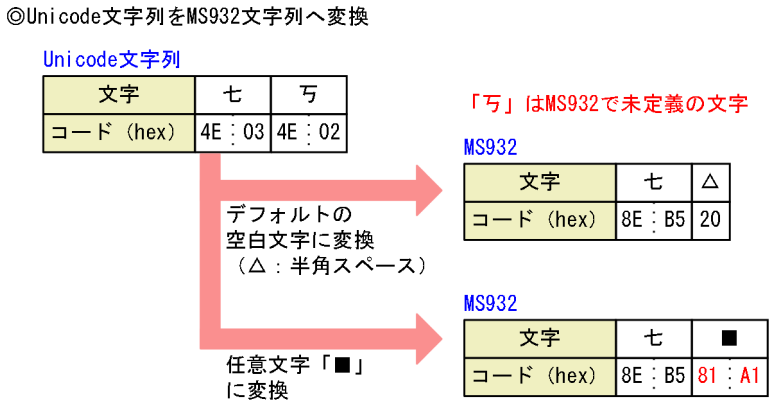
図 N-4　エラーになるバイナリフォーマット定義の例（出現回数がデータの終端までの場合）



付録 O　バイナリデータの未定義コードを置換する文字の定義

バイナリデータの文字コード変換処理中に未定義コードを検知した場合，空白文字に置換するのではなく，任意の文字に置換することができます。

図 O-1　未定義コードを任意の文字に置換する例



ここでは，未定義コードを検知した場合，任意の文字に置換する方法について説明します。なお，以降で説明する方法で設定をしなければ，未定義コードは空白文字に変換されます。

付録 O.1　前提条件

未定義コードを任意の文字に置換するには，HCSC サーバランタイム定義ファイルの telegram-undefined-character-code プロパティに REPLACE を指定する必要があります。

HCSC サーバランタイム定義ファイルの詳細は，マニュアル「サービスプラットフォーム リファレンス」の「6.5.6　HCSC サーバランタイム定義ファイル」を参照してください。

付録 O.2　検知した未定義コードを指定した文字に置換する処理の流れ

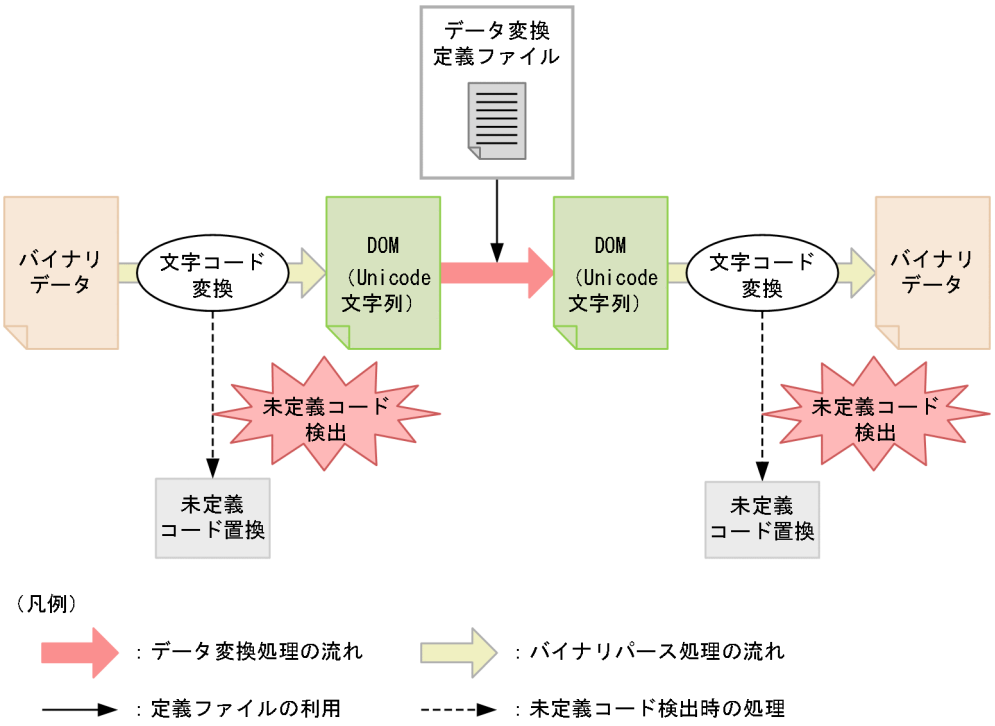
未定義コードを検知した場合に，指定した文字に置換するタイミングは次のとおりです。

- バイナリデータを Unicode 文字列に変換するとき
- Unicode 文字列をバイナリデータに変換するとき

Unicode 変換を経由してバイナリデータからバイナリデータへ変換する際に，検知した未定義コードを指定した文字に置換する処理の流れを次に示します。

バイナリデータの変換については，マニュアル「サービスプラットフォーム 解説」の「付録 B　データ変換機能の処理概要」を参照してください。

図 O-2　検知した未定義コードを指定した文字に置換する処理の流れ



未定義コードを任意の文字ではなく，スペースに置換する場合の処理の指定については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.1　フォーマットダイアログ」を参照してください。

付録 O.3　置換文字コードを指定する J2EE サーバ用ユーザプロパティ

未定義コードを任意の文字に変換する際に利用する置換文字コードを J2EE サーバ用ユーザプロパティファイル（usrconf.properties）に登録します。

usrconf.properties の格納先を次に示します。

Windows の場合

<サービスプラットフォームのインストールディレクトリ>¥CC¥server¥usrconf¥ejb¥<サーバ名称>¥usrconf.properties

UNIX の場合

/opt/Cosminexus/CC/server/usrconf/ejb/<サーバ名称>/usrconf.properties

usrconf.properties の詳細は，次のマニュアルを参照してください。

- J2EE サーバの互換モードが推奨モードの場合

  マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3　usrconf.properties (J2EE サーバ用ユーザプロパティファイル)」
- J2EE サーバの互換モードが V9 互換モードの場合

マニュアル「アプリケーションサーバ 機能解説 互換編」の「10.1.1　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」

usrconf.properties に指定できるプロパティキーについては，マニュアル「サービスプラットフォーム システム構築・運用ガイド」の「2.3.9　J2EE サーバ用ユーザプロパティファイル（usrconf.properties）を設定する」を参照してください。

付録 O.4　未定義コードの任意文字置換時の注意事項

(1)　置換時の設定について

不適切な設定によって，半角と全角のシフト状態が崩れたり，バイト数がずれたりして，置換後のデータが壊れることがないようにしてください。

指定した任意の文字へ置き換えるため，次のどれかを設定した場合は注意が必要です。

- 0 バイトの置換
- 元のサイズより大きいまたは小さいバイト数の指定
- 未定義文字の指定
- シフトコードの指定
- エスケープシーケンスの指定
- 特殊な文字（BOM など）の指定
- 半角と全角のシフト状態の変更

(2)　未定義コードの検知

未定義コードの検知は，変換対象のコードが，コード変換テーブルの対応として存在しない場合に行われます。ユーザマッピングテーブルを作成し，変換テーブルに対応を追加したことによって，未定義コードと検知されなくなった場合は，その対応どおりに変換します。意図した変換結果にならない場合，その原因の 1 つとして，ユーザマッピングテーブルによってテーブルを編集していないかどうかを確認してください。

付録 O.5　置換文字コードの設定例

置換文字コードの設定例を次に示します。

(1) バイナリデータから Unicode 文字列へ変換する例（置換種別が半角で半角文字に置換する場合）

変換元の文字コードを「UTF16」とし，入力文字を「𠮟」（Unicode 文字列：0xD840 0xDC0B（サロゲートペア））とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを Unicode の「a」（0x0061）に変換するプロパティの設定例と，置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.bin2xml=0061

●置換結果

「0xD840 0xDC0B」（未定義コード）→「0x0061」

(2) バイナリデータから Unicode 文字列へ変換する例（置換種別が半角で全角文字に置換する場合）

変換元の文字コードを「UTF16」とし，入力文字を「𠮟」（Unicode 文字列：0xD840 0xDC0B（サロゲートペア））とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを Unicode の「あ」（0x3042）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.bin2xml=3042

●置換結果

「0xD840 0xDC0B」（未定義コード）→「0x3042」

(3) バイナリデータから Unicode 文字列へ変換する例（置換種別が全角で全角文字に置換する場合）

変換元の文字コードを「UTF16」とし，入力文字を「𠮟」（Unicode 文字列：0xD840 0xDC0B（サロゲートペア））とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角スペースに置換する」を指定します。

未定義コードを Unicode の「あ」（0x3042）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.bin2xml.full=3042

●置換結果

「0xD840 0xDC0B」（未定義コード）→「0x3042」

(4) Unicode 文字列からバイナリデータへ置換する例（置換種別が半角の場合）

変換先の文字コードを「KEIS+EBCDIC」とし，入力文字を「ｶ」（0xFF76）とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを KEIS+EBCDIC の「a」（0x81）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.keisC=81

●置換結果

「0xFF76」（未定義コード）→「0x81」

(5) Unicode 文字列からバイナリデータへ置換する例（置換種別が全角の場合）

変換先の文字コードを「KEIS+EBCDIC」とし，入力文字を「ｾﾗ」（ﾗ：0x4E02）とします。「ﾗ」が未定義文字となり，置換種別を全角にするために入力文字列に「ｾ」を指定しています。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを KEIS+EBCDIC の「丁」（0xC3FA）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.keisC.full=C3FA

●置換結果

「0x4E02」（未定義コード）→「0xC3FA」

(6) Unicode 文字列からバイナリデータへ置換する例（置換種別が半角で全角文字に置換する場合）

変換先の文字コードを「KEIS+EBCDIC」とし，入力文字を「ｶ」（0xFF76）とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを KEIS+EBCDIC の「丁」（0xC3FA）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.keisC=0A42C3FA0A41※

注※

全角文字に置換するため，シフトコード（0A42 と 0A41）を定義します。

●置換結果

「0xFF76」（未定義コード）→「0x0A42 0xC3FA 0x0A41」

(7) Unicode 文字列からバイナリデータへ置換する例（置換種別が全角で半角文字に置換する場合）

変換先の文字コードを「KEIS+EBCDIC」とし，入力文字を「ｾﾗ」（ﾗ：0x4E02）とします。「ﾗ」が未定義文字となり，置換種別を全角にするために入力文字列に「ｾ」を指定しています。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを KEIS+EBCDIC の「a」（0x81）に変換するプロパティ設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.keisC.full=0A41810A42※

注※

半角文字に置換するため，シフトコード（0A42 と 0A41）を定義します。

●置換結果

「0x4E02」（未定義コード）→「0x0A41 0x81 0x0A42」

(8) Unicode 文字列からバイナリデータへ置換する例（未定義文字を削除する場合）

変換先の文字コードを「MS932」とし，入力文字を「ﾟ」（0x2232）とします。

バイナリフォーマット定義の［未定義コード置換時の処理］は，「全角・半角スペースに置換する」を指定します。

未定義コードを削除（0 バイト置換）するプロパティの設定例，および置換結果は次のとおりです。

●プロパティの設定例

csc.dt.undefined.replace.ms932=

●置換結果

「0x2232」（未定義コード）→「」（0 バイト置換）

付録 P　バイナリデータ内に文字コードが混在する場合の対応

バイナリデータ内で文字コードが混在する場合に，個別文字コードを使用することで単純内容要素ごとに文字コードが指定できます。個別文字コードは，バイナリフォーマット定義ファイル作成時に指定します。

付録 P.1　個別文字コードが指定できるデータ種別および指定可能な文字コード

個別文字コードが指定できるデータ種別は，次の要素です。

- 文字列型
- 整数型
- 実数型

要素のサイズが固定でも可変でも，個別文字コードを指定できます。

選択できる文字コード種別については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.1 フォーマットダイアログ」を参照してください。

付録 P.2　個別文字コードの指定方法

バイナリフォーマット定義エディタによる設定，および COBOL2FDX コンバータによる設定を次に示します。

(1)　バイナリフォーマット定義エディタによる設定

個別文字コードは，［単純内容要素］ダイアログで指定します。

［データ種別］で［文字列］，［整数］，または［実数］を選択し，［個別文字コード］の［指定する］チェックボックスにチェックを入れることで，個別文字コードを指定できます。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

(2)　COBOL2FDX コンバータによる設定

COBOL 登録集原文からバイナリフォーマット定義ファイルを生成する場合の個別文字コードの指定方法について説明します。

COBOL2FDX コンバータ，および COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

(a) 個別文字コード指定

文字列型要素の個別文字コード指定は，文字列型の基本項目（STRING 句）に CHARSET 句を指定してください。また，整数型要素の個別文字コード指定は，整数型の基本項目（INTEGER 句）に CHARSET 句を指定してください。

(b) 個別文字コードの一括指定

個別文字コードの一括指定は，集団項目（複合内容要素）に CHARSET 句を指定することで，集団項目の直下にある基本項目（単純内容要素）の文字コードを一括指定できます。

集団項目（複合内容要素）に CHARSET 句を指定した場合の動作を次に示します。

指定対象	動作
レコード	フォーマット定義の［文字コード］を設定する。※1
レコード以外の集団項目	直下の［単純内容要素］の［個別文字コード］を設定する。※2※3

注※1

レコードの CHARSET 句は，フォーマット定義の文字コード（バイナリフォーマット定義エディタの［フォーマット］ダイアログで設定した文字コード）として使用するため，レコードの直下にある単純内容要素は，基本項目の CHARSET 句で個別に設定する必要があります。

注※2

基本項目の CHARSET 句で，既に個別文字コードが設定されている単純内容要素については，単純内容要素の設定を優先します。

注※3

一括指定の対象となる単純内容要素は，CHARSET 句を指定した集団項目の「直下」にある単純内容要素だけです。配下すべての単純内容要素ではないため，孫の関係に当たる単純内容要素は含まれません。

付録 P.3 個別文字コード指定とセパレータを併用する場合の注意事項

個別文字コード指定とセパレータを併用する場合の注意事項について説明します。

- セパレータの文字コードは，個別文字コードの設定に関係なく，フォーマット定義の文字コードとなります。
- 個別文字コードには，次の文字コードを使用できません。
  - KEIS
  - IBM_CODE
  - JEF
  - KEIS+EBCDIK
  - KEIS+EBCDIC
  - IBM_CODE+EBCDIC(LATIN)

- IBM_CODE+EBCDIC(KANA)

- 個別文字コードに JEF+EBCDIC，または JEF+EBCDIK を指定する場合，各要素のデータは半角シフト状態で終わるようにしてください。全角シフト状態で終わる場合，セパレータの解析ができません。
- エスケープ文字は使用できません。［フォーマット］ダイアログ上でエスケープ文字を指定した場合，ダイアログの上部にエラーメッセージが出力され，［OK］ボタンが非活性となります。
- CSV 形式のデータは使用できません。バイナリフォーマット定義エディタ上で CSV 形式を選択した場合，次の動作となります。
  - バイナリフォーマット定義ファイルの新規作成時に「CSV 形式」を選択した場合

    ［単純内容要素］ダイアログの［個別文字コード］が非活性となり，個別文字コードが設定できなくなります。
  - 複合内容要素の［セパレータ］タブでセパレータ形式に「CSV」を選択した場合

    ダイアログの上部にエラーメッセージが出力され，［OK］ボタンが非活性となります。

付録 Q　CSV 形式の単純内容要素のデータなしの区別

サービスプラットフォームで扱う CSV 形式のデータは，単純内容要素が空データ（ダブルクォーテーションの囲みだけ）と，データなし（ダブルクォーテーションの囲みのない空データ）（以降，Null データと表記します）では，変換後の DOM 上で区別がありません。

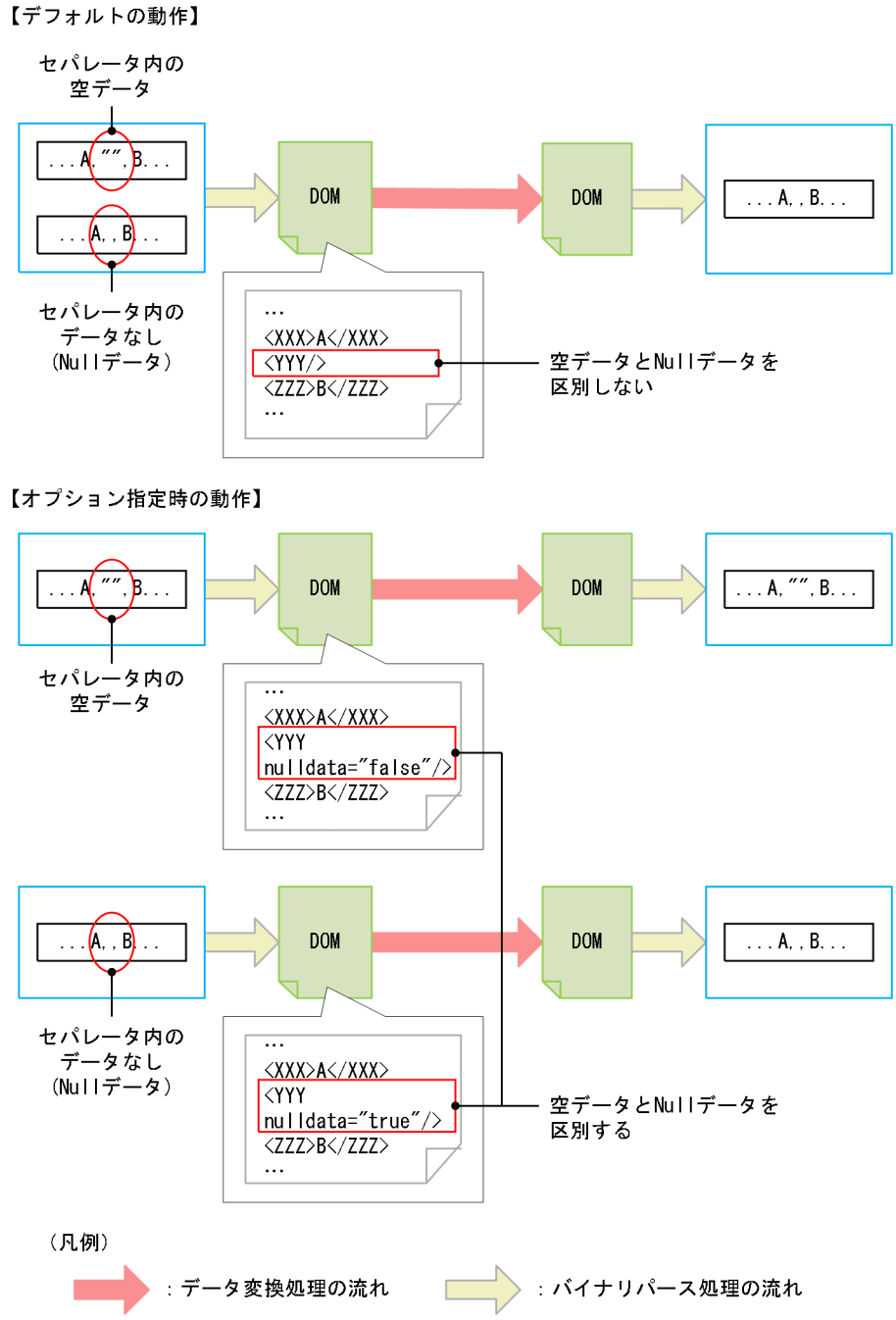
ここでは，CSV 形式のデータで，空データと Null データを明確に区別するための方法について説明します。

付録 Q.1　空データと Null データを区別するための処理の概要

サービスプラットフォームは，バイナリフォーマット定義ファイルで定義した non-XML 型変数を，バイナリパース処理で中間形式である DOM の形に内部的に変換します。CSV 形式の空データと Null データは，DOM 上では同様に扱いますが，空データと Null データを区別するオプションを指定することで，空データと Null データを区別して変換することができます。

CSV 形式のデータを別の CSV 形式のデータに変換する際の変換の流れを，次の図に示します。

図 Q-1　CSV 形式のデータを別の CSV 形式のデータに変換する際の変換の流れ



CSV 形式のデータと DOM で，空データと Null データを区別するための条件を次の表に示します。

表 Q-1　空データと Null データを区別する際のデータの条件

データ種別	データの条件	データの扱い
CSV 形式	""（ダブルクォーテーションだけ）	空データ
	（0 バイトデータ）	Null データ
DOM	nulldata 属性※なし， または nulldata 属性※の値が true でも 1 でもない	空データ

データ種別	データの条件	データの扱い
DOM	nulldata 属性※の値が true または 1	Null データ

注※

DOM の nulldata 属性については，「付録 Q.3　CSV 形式のデータおよび DOM の扱われ方」を参照してください。

付録 Q.2　空データと Null データを区別するための設定

CSV 形式データの空データと Null データを区別する設定について説明します。

(1)　前提条件

空データと Null データの区別は，入力元のバイナリデータが CSV 形式であることが前提となります。このため，区別可能な単純内容要素のデータ種別は，CSV 形式で使用できる種別だけになります。また，区別可能な要素は，バイナリフォーマット定義エディタ上で，ローカル単純内容要素，かつサイズが可変長であることが前提となります。

空データと Null データの区別対象となる単純内容要素の条件を，次の表に示します。

表 Q-2　空データと Null データの区別対象となる単純内容要素の条件

対象	条件
親複合内容要素のセパレータ形式	CSV 形式
単純内容要素のデータ種別	データ種別が次のどれかである。 • 文字列 • 整数 • 実数 • 固定小数部数値 • 日付時刻
サイズ	可変長（データの終端まで）

(2)　設定方法

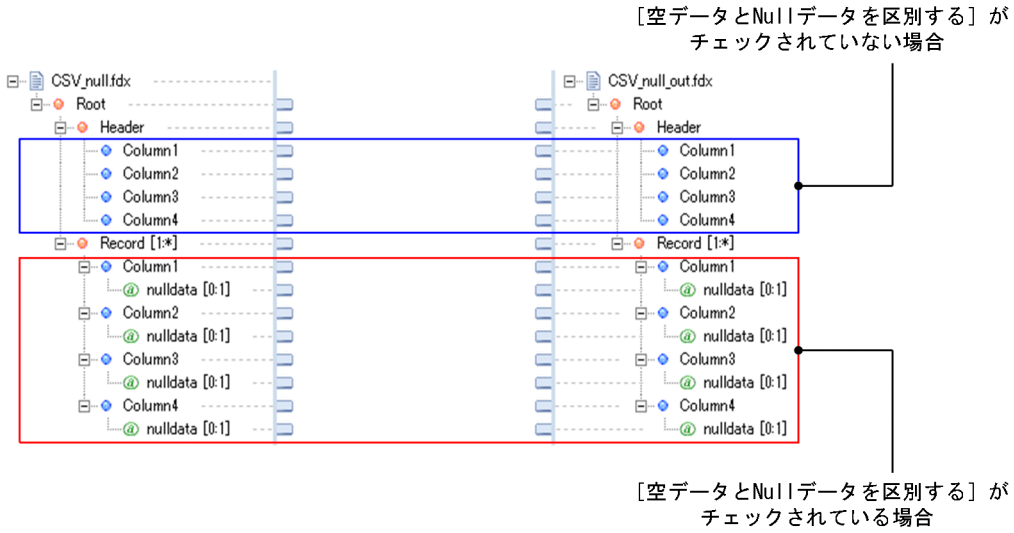
空データと Null データを区別する場合，バイナリフォーマット定義ファイルの［単純内容要素］ダイアログで，［CSV 形式］の［空データと Null データを区別する］チェックボックスをチェックします。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

［空データと Null データを区別する］チェックボックスをチェックすると，マッピング定義エディタ上で，対象の要素に nulldata 属性が表示されます。

［空データと Null データを区別する］チェックボックスをチェックした場合とチェックしなかった場合で，それぞれのマッピング画面のイメージを次の図に示します。

図 Q-2　［空データと Null データを区別する］チェックボックスをチェックした場合とチェックしなかった場合のマッピング画面のイメージ

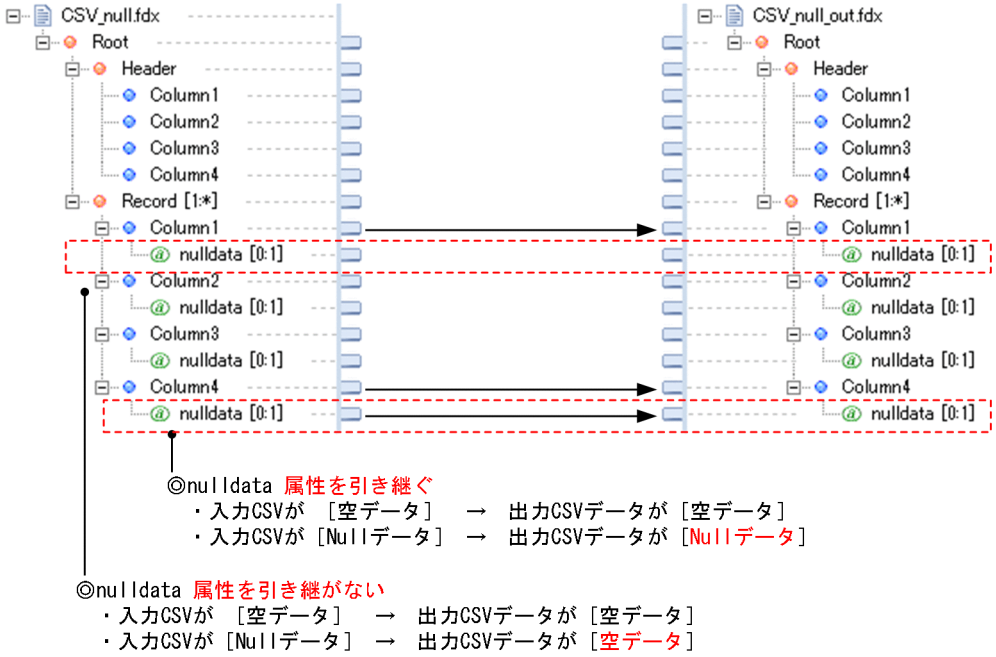


注意事項

CSV データの変換元で，空データと Null データを区別したまま，変換先に引き継ぎたい場合には，マッピング定義上で nulldata 属性同士をマッピングし，nulldata 属性の値も一緒に引き継ぐ必要があります。

複合内容要素 Record 配下の各要素に対して，空データと Null データを区別する設定が有効の場合を例に，nulldata 属性の引き継ぎの有無の違いによるデータの差異を，次の図に示します。

図 Q-3　nulldata 属性を引き継がない場合と引き継いだ場合の差異



付録 Q.3　CSV 形式のデータおよび DOM の扱われ方

バイナリフォーマット定義ファイルの［単純内容要素］ダイアログの［CSV 形式］の［空データと Null データを区別する］チェックボックスをチェックした場合（Null データを区別する設定が有効の場合）の，CSV 形式のデータと DOM の扱われ方について説明します。

(1)　CSV 形式のデータの扱われ方

CSV 形式のデータでは，空データと Null データを区別する設定が有効の場合，ダブルクォーテーション（"）の囲みの有無によって，データの扱われ方は次のようになります。

- ダブルクォーテーション（"）で囲まれている：空データとして扱われます。
- ダブルクォーテーション（"）で囲まれていない：Null データとして扱われます。

値がある（空白だけを含む）場合は，空データと Null データを区別をしないため，Null データとして扱われません。

データの内容ごとのデータの扱いを次の表に示します。

表 Q-3　データの扱い（CSV 形式のデータ）

データの内容		データの扱い	
「"」の囲み	内容	設定※が有効	設定※が無効
あり	空文字列	空	空
	空白だけ	空白だけ	空白だけ
	空白以外の文字列	空白以外の文字列	空白以外の文字列
なし	空文字列	Null	空
	空白だけ	空白だけ	空白だけ
	空白以外の文字列	空白以外の文字列	空白以外の文字列

注※

空データと Null データを区別する設定を表します。

空データと Null データを区別する設定が有効の場合のデータの扱われ方の例を次に示します。

CSV 形式のデータ※	データの扱われ方
,"",	空
,,	Null
,"△△△",	"△△△" （空白を埋め字とし，不要文字を削除する場合は空）
,△△△,	"△△△" （空白を埋め字とし，不要文字を削除する場合は空）

CSV 形式のデータ※	データの扱われ方
,a,	a

（凡例）

△：半角スペース

注※

コンマ（,）で囲まれた中央部分が対象のデータです。

(2) DOM の扱われ方

DOM では，空データと Null データを区別する設定が有効の場合，対象の単純内容要素に nulldata 属性が指定できるようになり，属性の値で「空」と「Null」が区別されます。指定できる nulldata 属性の属性値を次に示します。

- 属性の型は xsd:boolean 型で，"true"，"false"，"1"，"0"を設定できます。
- "true"または"1"の場合は「真」，"false"または"0"の場合は「偽」になります。
- 設定可能な値以外の値が指定された場合は，「偽」になります。
- 属性値の大文字と小文字を区別し，先頭および終端の空白は削除してから判定します。

nulldata 属性の属性値が「真」の場合「Null」として扱われ，要素に値が設定されていても無視します（「Null」と解釈します）。nulldata 属性がない場合，または属性値が「偽」の場合，要素の値で扱われます。

DOM を CSV 形式に戻す場合のデータの扱いを次の表に示します。

表 Q-4　データの扱い（DOM）

DOM		CSV 形式のデータ
nulldata 属性の属性値	要素の値	データの扱い
• "true"，"1"	空文字列	Null
	空白だけ	
	空白以外の文字列	
• "false"，"0" • "true"以外の値，"1"以外の値 • 属性の指定なし	空文字列	空
	空白だけ	空白だけ
	空白以外の文字列	空白以外の文字列

空データと Null データを区別する設定が有効の場合のデータの扱われ方の例を次に示します。DOM の elem 要素が，空データと Null データを区別する設定が有効となっている単純内容要素に対応しています。

DOM	データの扱われ方
<elem/>	空
<elem nulldata="true"/>	Null

DOM	データの扱われ方
<elem>△△△</elem>	"△△△" （空白を埋め字とし，不要文字を削除する場合は空）
<elem nulldata="true">△△△</elem>	Null
<elem>a</elem>	a
<elem nulldata="true">a</elem>	Null
<elem nulldata="1△">a</elem>	Null
<elem nulldata="TRUE">a</elem>	a

（凡例）

△：半角スペース

付録 Q.4　CSV 形式のデータと DOM の相互変換

空データと Null データを区別する設定が有効な場合の，CSV 形式データと DOM の相互変換について説明します。

(1)　CSV 形式から DOM への変換時の内容

DOM への変換では，空データと Null データを区別する設定が有効の場合，CSV 形式のデータを読み込む際に「空」と「Null」を区別します。入力データが「空」に該当する場合，DOM のうち対応する要素の nulldata 属性に「false」を出力します。入力データが「Null」に該当する場合，nulldata 属性に「true」を出力します。

空データと Null データを区別する設定が有効の場合の，CSV 形式のデータから DOM への変換時の内容を次の表に示します。

表 Q-5　CSV 形式のデータから DOM への変換時の内容

データの扱い	DOM の内容※
空	<elem nulldata="false"/>
Null	<elem nulldata="true"/>
空白だけ	<elem>［空白］</elem> ただし，空白を削除する設定の場合は，<elem/>
空白以外の文字列	<elem>［空白以外の文字列］</elem>

注※

対応する FDX の単純内容要素の要素名はすべて「elem」とします。

CSV 形式のデータから DOM への変換例を次の表に示します。

表 Q-6　CSV 形式のデータから DOM への変換例

変換元					変換先
CSV 形式のデータ※1	バイナリフォーマット定義エディタの設定※2				DOM※5
	データ種別	埋め字	不要文字	空データと Null データの区別	
,"",	文字列 文字列型数値※3 日付時刻※4	（任意）	（任意）	有効	<elem nulldata="false"/>
				無効	<elem/>
,,	文字列 文字列型数値※3 日付時刻※4	（任意）	（任意）	有効	<elem nulldata="true"/>
				無効	<elem/>
,"△△△", ,△△△,	文字列	スペース	削除する	（任意）	<elem/>
			削除しない		<elem>△△△</elem>
		空文字（0x00） 空文字までをデータとする	－		<elem>△△△</elem>
		空文字（0x00）	（任意）		<elem>△△△</elem>
	文字列型数値※3	スペース	削除する	（任意）	エラー （有効な数値でない）
			削除しない		エラー （有効な数値でない）
		0	（任意）		エラー（不正な文字）
	日付時刻※4	－	－	（任意）	エラー （有効な数値でない）
,"20190101", ,20190101,	文字列 文字列型数値※3	（任意）	（任意）	（任意）	<elem>20190101</elem>
	日付時刻※4	－	－		<elem>20190101</elem>

（凡例）

△：半角スペース

－：該当する設定がありません

注※1

コンマ（,）で囲まれた中央部分が対象のデータです。

注※2

サイズは，すべて「可変長（データの終端まで)」とします（「サイズノード指定」は含みません)。

注※3

文字列型数値は，整数，実数，固定小数部数値を指します。

注※4

日付時刻の形式は，"CCYYMMDD"を選択しているとします。

注※5

対応する FDX の単純内容要素の要素名は，すべて「elem」とします。

(2) DOM から CSV 形式への変換時の内容

DOM を読み込む際に，空データと Null データを区別する設定が有効の場合，「空」と「Null」を区別します。入力データが「空」に該当する場合，ダブルクォーテーション（"）の囲みを出力します。入力データが「Null」に該当する場合，何も出力しません。

入力データが「空」または「Null」に該当する場合，バイナリフォーマット定義の［出力時のダブルクォーテーション「"」の囲み］チェックボックスの指定は無視されます。入力データが「空」でも「Null」でもない場合は，［出力時のダブルクォーテーション「"」の囲み］チェックボックスの指定が有効となります。

また，入力データがバイナリフォーマット定義で指定された［埋め字］だけで，［不要文字］が削除する設定になっている場合は，不要文字が削除されて空となります。この場合，空データと Null データを区別する設定が有効であればダブルクォーテーション（"）で囲まれるので，「空」と区別できます。空データと Null データを区別する設定が無効の場合は，ダブルクォーテーション（"）で囲まないで出力されます。

空データと Null データを区別する設定が有効の場合の，DOM から CSV 形式への変換時の内容を次の表に示します。

表 Q-7 DOM から CSV 形式への変換時の内容

データの扱い	バイナリフォーマット定義の設定		CSV 形式のデータ※
	「"」の囲み	不要文字	
空	あり	削除する	,"",
		削除しない	
	なし	削除する	
		削除しない	
Null	あり	削除する	,,
		削除しない	
	なし	削除する	
		削除しない	
空白だけ	あり	削除する	,"",
		削除しない	,"［空白］",
	なし	削除する	,"",

データの扱い	バイナリフォーマット定義の設定		CSV 形式のデータ※
	「"」の囲み	不要文字	
空白だけ	なし	削除しない	,［空白］,
空白以外の文字列	あり	削除する	,"［空白以外の文字列］",
		削除しない	
	なし	削除する	,［空白以外の文字列］,
		削除しない	

注※

コンマ（,）で囲まれた中央部分が対象のデータです。

DOM から CSV 形式のデータへの変換例を次の表に示します。

表 Q-8　DOM から CSV 形式のデータへの変換例

変換元	変換先					
DOM※1	バイナリフォーマット定義エディタの設定※2					CSV 形式のデータ※5
	「"」の囲み	データ種別	埋め字	不要文字	空データと Null データの区別	
<elem/> <elem nulldata="false"/> <elem nulldata="0"/>	あり	文字列 文字列型数値※3 日付時刻※4	（任意）	（任意）	有効	,"",
					無効	,,
	なし		（任意）	（任意）	有効	,"",
					無効	,,
<elem nulldata="true"/> <elem nulldata="1"/> <elem nulldata="true">△△△</elem> <elem nulldata="true">a</elem>	（任意）	文字列 文字列型数値※3 日付時刻※4	（任意）	（任意）	有効	,,
<elem>△△△</elem>	あり	文字列	スペース	削除する	（任意）	,"",
				削除しない		,"△△△",
			空文字（0x00） 空文字までをデータとする	－		,"△△△",
			空文字（0x00）	削除する		,"",
				削除しない		,"△△△",
	なし		スペース	削除する	有効	,"",

変換元	変換先					
DOM※1	バイナリフォーマット定義エディタの設定※2					CSV 形式のデータ※5
	「"」の囲み	データ種別	埋め字	不要文字	空データと Null データの区別	
<elem>△△△</elem>	なし	文字列	スペース	削除する	無効	,,
				削除しない	（任意）	,△△△,
			空文字（0x00） 空文字までをデータとする	－	（任意）	,△△△,
			空文字（0x00）	削除する	有効	,"",
					無効	,,
				削除しない	（任意）	,△△△,
	あり	文字列型数値※3	（任意）	（任意）	（任意）	,"0",
	なし		（任意）	（任意）	（任意）	,0,
	あり	日付時刻※4	－	－	（任意）	,"20000101",
	なし		－	－	（任意）	,20000101,
<elem>20190101</elem>	あり	文字列 文字列型数値※3	（任意）	（任意）	（任意）	,"20190101",
	なし		（任意）	（任意）	（任意）	,20190101,
<elem>2019-01-01</elem>	あり	日付時刻※4	－	－	（任意）	,"20190101",
	なし		－	－	（任意）	,20190101,

（凡例）

△：半角スペース

－：該当する設定がありません

注※1

対応する FDX の単純内容要素の要素名は，すべて「elem」とします。

注※2

サイズは，すべて「可変長（データの終端まで)」とします（「サイズノード指定」は含みません)。

注※3

文字列型数値は，整数，実数，固定小数部数値を指します。

注※4

日付時刻の形式は，"CCYYMMDD"を選択しているとします。

注※5

コンマ（,）で囲まれた中央部分が対象のデータです。

付録 R　LOW-VALUE（0x00 で埋められたデータ）と空要素を相互に変換する場合の対応

バイナリデータ変換では，LOW-VALUE 変換オプションを使用すると，COBOL 特有のデータ形式である LOW-VALUE（0x00 で埋められたデータ）を，空要素に変換できます。また，空要素から LOW-VALUE（0x00 で埋められたデータ）への変換もできます。

ここでは，LOW-VALUE と空要素を相互に変換するための LOW-VALUE 変換オプションについて説明します。

付録 R.1　前提条件

LOW-VALUE 変換オプションに対応する要素およびサイズ種別を次の表に示します。

表 R-1　LOW-VALUE 変換オプションに対応する要素およびサイズ種別

対象	条件
LOW-VALUE 変換指定元要素	単純内容要素でかつデータ種別が次のどれかである。 • 文字列 • 整数 • 実数 • 固定小数部数値
サイズ種別	固定長である。

付録 R.2　設定方法

バイナリフォーマット定義エディタによる設定，および COBOL2FDX コンバータによる設定を次に示します。

(1)　バイナリフォーマット定義エディタによる設定

LOW-VALUE 変換オプションは，［単純内容要素］ダイアログで指定します。

［単純内容要素］ダイアログの表示・入力内容については，マニュアル「サービスプラットフォーム リファレンス」の「1.3.2　単純内容要素ダイアログ」を参照してください。

(a)　データ種別が文字列型の場合の設定方法

［単純内容要素］ダイアログの［構成要素］タブを選択し，次の情報を設定します。

[データ種別]

［文字列］を選択します。

[サイズ]

［固定］チェックボックスにチェックを入れます。

[埋め字]

［スペース］を選択します。

[COBOL データ形式]

［LOW-VALUE の変換］チェックボックスをチェックします。

(b)　データ種別が整数型，実数型，および固定小数部数値型の場合の設定方法

［単純内容要素］ダイアログの［構成要素］タブを選択し，次の情報を設定します。

[データ種別]

［整数］，［実数］または［固定小数部数値］を選択します。

[サイズ]

［固定］チェックボックスにチェックを入れます。

[COBOL データ形式]

［LOW-VALUE の変換］チェックボックスをチェックします。

(c)　注意事項

- LOW-VALUE 変換オプションが設定された要素を，ノード指定（選択条件ノード，サイズノード，および出現回数ノード）に使用できません。
- バイナリフォーマット定義が CSV 形式の場合は，LOW-VALUE 変換オプションが使用できません。

(2)　COBOL2FDX コンバータによる設定

LOW-VALUE 変換オプションを使用する場合は，文字列型の基本項目（STRING 句），または整数型の基本項目（INTEGER 句）に，CONVERT-LOW-VALUE 句を追加してください。

COBOL2FDX コンバータ，および COBOL 登録集原文ファイルによるバイナリフォーマット定義ファイルの生成方法については，「4.5　COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成」を参照してください。

注意事項

CONVERT-LOW-VALUE 句を定義した場合，次の条件が満たされていなければエラーになります。

- 文字列型の基本項目（STRING 句）
  SIZE 句が整数（固定長かつ 0 でない），かつ EMBEDDING-CHARACTER が SPACE。
- 整数型の基本項目（INTEGER 句）
  SIZE 句が整数（固定長）。

付録 S　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

数字

1 つの文字列から一部分の文字列を取り出す 523

A

API を使用したデータ変換 952
assign 要素の変換〔BPEL1.1〕 1113
assign 要素の変換〔BPEL2.0〕 1134

B

BitOpObjects（論理演算ファンクション） 660
BPEL1.1 のビジネスプロセス定義をインポートする場合 1100
BPEL2.0 のビジネスプロセス定義をインポートする場合 1118
BPEL のサポート範囲 1099
BPEL ファイルの作成 239

C

CalculateObjects（数値演算ファンクション） 656
ChooseObjects（選択ファンクション） 669
COBOL 登録集原文ファイルからのバイナリフォーマット定義ファイルの生成 199
COBOL 登録集原文ファイルの記述規則 200
COBOL 登録集原文ファイルの規則 199
Code Converter による拡張 97
ConcatenateObjects（文字列連結ファンクション） 647
ConstantObjects（定数ファンクション） 676
ContainObjects（文字列調査ファンクション） 652
CopyObjects（変換先ノードへのマッピング） 645
correlationSets 要素の変換〔BPEL1.1〕 1108
correlationSets 要素の変換〔BPEL2.0〕 1129
CountObjects（ノード数取得ファンクション） 664
cscgenbinary コマンドの使用方法 877
cscjson2xsdadv コマンド〔JSON 変換ツール〕 1015
cscjson2xsd コマンド 1018
cscjson2xsd コマンド〔JSON 変換ツール〕 1015
cscjsonxmlconvadv コマンド〔JSON 変換ツール〕 1015
cscjsonxmlconv コマンド〔JSON 変換ツール〕 1015
CSCMsgServerException クラス 791
CSCOwnCodeConverter インターフェース 1162
CSCOwnCodeReaderContext インターフェース 1181
CSCOwnCodeReader インターフェース 1168
CSV 形式の単純内容要素のデータなしの区別 1227
currentDateTime（システム日付取得関数） 684
CustomObjects（カスタムファンクション） 673

D

DB アダプタで取得したデータの自動マッピング 1196
DB アダプタで取得したデータのフォーマットの変換例 1189
DB キューを利用したサービス部品の適用範囲 94
DB クライアント（HCSC サーバ）側の設定 37
DB クライアントの環境変数の設定 38
DB サーバ側の設定 36
DB サーバの構築および環境設定 36

E

Eclipse に関する注意事項 152
element 名 85
empty 要素の変換〔BPEL1.1〕 1113
empty 要素の変換〔BPEL2.0〕 1135
Excel を利用したマッピング定義時の注意事項 680
Excel を利用したマッピング定義のインポート 620
Excel を利用したマッピング定義のエクスポート 639
ExistObjects（ノード調査ファンクション） 665

F

faultHandlers 要素の変換〔BPEL1.1〕 1109
faultHandlers 要素の変換〔BPEL2.0〕 1129
flow 要素の変換〔BPEL1.1〕 1117
flow 要素の変換〔BPEL2.0〕 1139

FormatObjects（数値フォーマット変換ファンクション） 654

G

getPropertiesByGroup（外部プロパティ取込関数） 687
getSystemEnv（環境変数取得関数） 686
getWorkFolderPath（作業フォルダパス取得関数） 683
gRPC 受付または gRPC アダプタのオペレーション・フォルトの定義方法 490

H

HCSC 簡易セットアップ画面（[サーバ名称] タブ） 57
HCSC 簡易セットアップ画面（[メイン] タブ） 56
HCSC 簡易セットアップ画面で入力する項目 58
HCSC 簡易セットアップ機能 51
HCSC 簡易セットアップ機能で構築できる環境 51
HCSC 簡易セットアップ機能でセットアップした組み込みデータベースの運用 77
HCSC 簡易セットアップ機能でセットアップしたテスト環境の運用 63
HCSC 簡易セットアップ機能の実行 53
HCSC コンポーネントの一覧を更新する 717
HCSC コンポーネントの情報の参照 716
HCSC コンポーネントの情報を表示する 717
HCSC コンポーネントのバージョン管理 406
HCSC コンポーネントのパッケージングと配備定義 707
HCSC コンポーネントを HCSC サーバに配備して開始する処理の一括実行 719
HCSC コンポーネントを HCSC サーバに配備して開始する方法 721
HCSC コンポーネントを停止して HCSC サーバから削除する処理の一括実行 727
HCSC コンポーネントを停止して HCSC サーバから削除する方法 729
HiRDB/Single Server のインストール 36
HiRDB SQL Executer のインストールおよび環境変数の設定 36

I

IBM 漢字コードの文字セットの変更 1203
if 要素の変換〔BPEL2.0〕 1138
invoke 要素の変換〔BPEL1.1〕 1111
invoke 要素の変換〔BPEL2.0〕 1132

J

J2EE サーバの互換モードの選択 44
Java 雛形ファイル 601
Java 雛形ファイルの生成 601
Java プログラムのコーディング，ビルド，およびデバッグ 603
Java プログラムのパッケージング 604
Java 呼出アクティビティ 291
JMS のサービスリクエスタ 816
JMS メッセージを生成する〔標準の非同期受付（MDB(DB キュー)）〕 822
JMS メッセージを生成する〔標準の非同期受付（MDB(WS-R)）〕 803
JMS メッセージを送信する〔標準の非同期受付（MDB(DB キュー)）〕 828
JMS メッセージを送信する〔標準の非同期受付（MDB(WS-R)）〕 809
JSON 形式から XML 形式に変換する際のオブジェクト名の順序付け 902
JSON 形式データに対応する XML スキーマの生成 1015
JSON データの変換 886
JSON 変換ツールの概要 1013
JSON 変換ツールの実行例 1022
JSON 変換ツールの注意事項 1027
JSON 変換ツールを使用したデータ変換 1012
JSON-XML 変換 API の障害対策 997
JSON-XML 変換 API の詳細 998
JSON-XML 変換 API の使用方法 1008
JSON-XML 変換 API の使用例 1009
JSON-XML 変換 API 変換定義の設定 994
JSON-XML 変換 API を使用したデータの形式変換 993
JSON-XML 変換結果の確認 1015

JSON-XML 変換で扱う JSON と標準仕様の JSON との差異 887
JSON-XML 変換定義ファイルの設定 904
JSON-XML 変換に関する注意事項 907
JSON-XML 変換の概要 887
JSON-XML 変換の規則 892

L

LengthObjects（文字数取得ファンクション） 651
LoopObjects（くり返しファンクション） 666
LOW-VALUE（0x00 で埋められたデータ）と空要素を相互に変換する場合の対応 1238
lrtrim（前後空白除去関数） 685

N

NameObjects（ノード名取得ファンクション） 663
Namespaces（名前空間情報） 643
NotObjects（否定演算ファンクション） 660

P

process 要素の変換〔BPEL1.1〕 1107
process 要素の変換〔BPEL2.0〕 1127

R

RadixObjects（基数変換ファンクション） 672
RD エリアの準備 37
RD エリアの未使用領域を確認する 77
receive 要素の変換〔BPEL1.1〕 1110
receive 要素の変換〔BPEL2.0〕 1130
Reliable Messaging のローカルキューを利用したサービス部品の適用範囲 93
Reliable Messaging 用のユーザ権限の設定および RD エリアの準備 38
ReplaceObjects（値置換えファンクション） 671
reply 要素の変換〔BPEL1.1〕 1110
reply 要素の変換〔BPEL2.0〕 1131
RoundObjects（端数処理ファンクション） 657

S

scope 要素の変換〔BPEL1.1〕 1115
scope 要素の変換〔BPEL2.0〕 1137
sequence 要素の変換〔BPEL1.1〕 1118
sequence 要素の変換〔BPEL2.0〕 1140
SessionBean の作成条件 90
SessionBean を利用したサービス部品の適用範囲 89
ShiftObjects（シフト演算ファンクション） 662
soap:binding 要素 84
soap:body 要素 84
soap:fault 要素 85
soap:header 要素 86
soap:operation 要素 84
SOAP1.1/1.2 併用モード 42
SOAP1.1 モード 42
SOAP 受付に要求を出すサービスリクエスタ 834
SOAP 受付または SOAP アダプタの WSDL ファイル，および SessionBean アダプタの EAR ファイルのエクスポートについて 488
SOAP 受付または SOAP アダプタのヘッダ電文の定義方法 490
SOAP モードの確認 75
SOAP モードのサポート範囲 42
SOAP モードの種類 42
SOAP モードの選択 45
SOAP モードを変更する 109
SOA に基づいたシステム開発の概要 21
SubstringObjects（部分文字列取得ファンクション） 648
SumObjects（総和ファンクション） 659
switch 要素の変換〔BPEL1.1〕 1116

T

throw 要素の変換〔BPEL1.1〕 1114
throw 要素の変換〔BPEL2.0〕 1135
TP1/Server Base Enterprise Option（TP1/EE）のサービスリクエスタ 816
TrimObjects（空白除去ファンクション） 653

V

V9 互換モード 44
variables 要素の変換〔BPEL1.1〕 1107

variables 要素の変換〔BPEL2.0〕 1128

W

wait 要素の変換〔BPEL1.1〕 1114
wait 要素の変換〔BPEL2.0〕 1136
Web サービスを利用したサービス部品の適用範囲 81
while 要素の変換〔BPEL1.1〕 1116
while 要素の変換〔BPEL2.0〕 1137
wsdl:fault 要素 85
wsdl:import 要素 83
wsdl:operation 要素〔SOAP1.1/1.2 併用モード〕 87
wsdl:operation 要素〔SOAP1.1 モード〕 84
wsdl:port 要素 86
wsdl:service 要素 86
wsdl:types 要素〔SOAP1.1/1.2 併用モード〕 87
wsdl:types 要素〔SOAP1.1 モード〕 83
WSDL1.1 仕様のサポート範囲 88
WSDL を取得する〔標準の同期受付（Web サービス/JAX-WS エンジン)〕 768
WSDL を取得する〔標準の同期受付（Web サービス/SOAP 通信基盤)〕 740
WSDL を登録する 1081
WSDL を編集する〔SOAP 受付（Web サービス/SOAP 通信基盤)〕 836

X

XML Schema 型 dateTime 324
XML Schema 型 duration 323
XML スキーマの適用範囲 94
XML フォーマット定義ファイル 156
XpathObjects（XPath ファンクション） 677
XPath の評価 873
xsd:schema 要素 83

あ

空きページおよび空きセグメントを解放する 79
アクティビティ 243
アクティビティからの相関セットの設定 280
アクティビティ定義ダイアログの XPath の指定 350
アクティビティの移動と連結 251
アクティビティのオートコネクト 250
アクティビティの定義 284
アクティビティの定義情報を変更する 367
アクティビティの配置と連結 245, 251
アクティビティの連結条件 248
アクティビティ名を変更する 367
アクティビティを配置する 245
アクティビティを連結する 246
値置換えファンクション 551
アンインストール 38

い

移行の対象となるバージョン 1032
インスタンスを生成する〔標準の同期受付(SessionBean)〕 786
インストール 34
インポート〔プロジェクト〕 108
インポート〔リポジトリ〕 116

う

受付アクティビティ 285
運用中のビジネスプロセス定義を変更する 368

え

永続化 240
エクスポート先のディレクトリ名およびファイル名のリネームについて 489
エクスポート実行時に削除されるディレクトリ 486
エクスポート時の同名ファイルの定義と検証 487
エクスポート〔プロジェクト〕 108
エクスポート〔リポジトリ〕 112
エラー情報を取得する〔SOAP 受付（Web サービス/SOAP 通信基盤)〕 838
エラー情報を取得する〔標準の同期受付(SessionBean)〕 790
エラー情報を取得する〔標準の同期受付（Web サービス/JAX-WS エンジン)〕 772
エラー情報を取得する〔標準の同期受付（Web サービス/SOAP 通信基盤)〕 750

お

応答アクティビティ　287
応答電文　154
応答電文を取得する〔SOAP 受付（Web サービス/SOAP 通信基盤）〕　837
応答電文を取得する〔標準の同期受付（SessionBean）〕　790
応答電文を取得する〔標準の同期受付（Web サービス/JAX-WS エンジン）〕　772
応答電文を取得する〔標準の同期受付（Web サービス/SOAP 通信基盤）〕　748
応答電文を取得する〔標準の非同期受付（MDB（DB キュー））〕　833
応答電文を取得する〔標準の非同期受付（MDB（WS-R））〕　812
応答用のキューを設定する〔標準の非同期受付（MDB（DB キュー））〕　828
応答用のキューを設定する〔標準の非同期受付（MDB（WS-R））〕　810
応答を取り出す〔標準の非同期受付（MDB（DB キュー））〕　830
応答を取り出す〔標準の非同期受付（MDB（WS-R））〕　811
オブジェクトを生成する〔SOAP 受付（Web サービス/SOAP 通信基盤）〕　837
オブジェクトを生成する〔標準の同期受付（Web サービス/JAX-WS エンジン）〕　771
オブジェクトを生成する〔標準の同期受付（Web サービス/SOAP 通信基盤）〕　742
オペレーションを割り当てる　1083

か

開始アクティビティ　285
開発から実運用までの流れ　22
開発環境を使用するためのセットアップ　32
開発時の注意事項　109
開発手順　46
外部バインディングファイルを利用した WSDL のカスタマイズ　1202
概要 WSDL を作成する　1076
概要インターフェースを検討する　1075
拡張関数　361
拡張モードでの JSON データの変換　908
カスタムファンクションで使用する Java プログラムの作成　596
空データと Null データを区別するための処理の概要　1227
空データと Null データを区別するための設定　1229
環境設定　34
環境変数グループの設定　38

き

基数変換する　554
基数変換ファンクション　554
基本アクティビティに関連する要素の変換〔BPEL1.1〕　1109
基本アクティビティに関連する要素の変換〔BPEL2.0〕　1130
旧バージョンからの移行　1032
旧バージョンからの移行手順　1033
旧バージョンで開発環境と運用環境のリポジトリを共有していた場合の移行手順　1037

く

空白除去ファンクション　527
組み込みデータベースの構築　35
組み込みデータベースの再起動　37
組み込みデータベースの同時接続数の設定　38
クライアント相関 ID の設定　384
クラスタから HCSC コンポーネントを削除する　715
クラスタに HCSC コンポーネントを追加する　714
繰り返しアクティビティ　335
くり返し依存対象が設定された変換元ノードのパスの表示　571
くり返し依存対象の対応づけ　566
くり返し依存対象を確認する　570
くり返し依存対象を変更する　571
くり返し設定ダイアログを使用してマッピングする　564
くり返しの設定　564
くり返しファンクション　544

くり返しを合成する　565
くり返しを対応づける　543
グローバル定義の要素を構成要素に設定する　174
グローバル変数　264

け

検証アクティビティ　327
検証結果の表示〔バイナリフォーマット定義ファイル〕　197
検証設定　106
検証内容の表示〔ビジネスプロセス〕　401
検証内容〔バイナリフォーマット定義ファイル〕　181
検証内容〔ビジネスプロセス〕　385
検証方法〔バイナリフォーマット定義ファイル〕　197
検証方法〔ビジネスプロセス〕　400

こ

構成の形態，J2EE サーバの互換モード，および SOAP モードの選択　40
構成要素　173
構成要素の開始セパレータを選択して選択条件を設定する　176
構成要素の選択条件を設定する　175
構成要素を設定する　173
構造アクティビティに関連する要素の変換〔BPEL1.1〕　1115
構造アクティビティに関連する要素の変換〔BPEL2.0〕　1136
構文拡張した COBOL 登録集原文を使用する場合の記述規則の変更点　201
効率良く XML スキーマを生成する方法〔cscjson2xsd コマンド〕　1018
コード変換機能の拡張（Code Converter 使用時）97
固定小数部数値　163
コネクションで連結できる配置済みのアクティビティ　251
個別文字コードが指定できるデータ種別および指定可能な文字コード　1224
個別文字コード指定とセパレータを併用する場合の注意事項　1225
個別文字コードの指定方法　1224
コメント　364
コンポーネント共通 UOC　1151

さ

サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン）　782
サービスアダプタの稼働状況確認を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤）　757
サービスアダプタを作成する　1087
サービスアダプタを指定する要素　446
サービスアダプタを利用した HTTP ヘッダおよび Cookie 情報の引き継ぎ　1141
サービスクラスを生成する〔標準の同期受付（Web サービス/JAX-WS エンジン）〕　769
サービス定義 XML ファイルでのビジネスプロセス定義の考え方　408
サービス定義 XML ファイルのインポート手順　464
サービス定義 XML ファイルのエクスポート手順　470
サービス定義 XML ファイルの作成と指定内容　411
サービス定義 XML ファイルの内容　408
サービス定義 XML ファイルのルートを指定する要素　413
サービス定義がエクスポートされる出力ディレクトリの構成　475
サービス定義のインポート　464
サービス定義のインポート機能　404
サービス定義のインポート・エクスポート　403
サービス定義のインポート・エクスポートの概要 404
サービス定義のインポート・エクスポートの使用の目的　406
サービス定義のエクスポート　470
サービス定義のエクスポート機能　404
サービスのエミュレーションの設定　851
サービスの概要インターフェースを検討する　1075
サービスの自動エミュレーション　874
サービスの手動エミュレーション　876
サービスの詳細インターフェースを検討する　1085
サービス部品電文　154

サービス部品電文を作成する（MDB（WS-R または DB キュー）の場合） 158
サービス部品電文を作成する（SessionBean の場合） 158
サービス部品電文を作成する（Web サービスの場合） 157
サービスプラットフォームを利用したシステムの開発 46
サービス呼出アクティビティ 289
サービスリクエスタ 736
サービスリクエスタ（SOAP 受付）の作成手順（JAX-WS エンジン） 841
サービスリクエスタ（SOAP 受付）の作成手順（SOAP 通信基盤） 834
サービスリクエスタ（標準の同期受付（SessionBean））の作成手順 784
サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（JAX-WS エンジン） 766
サービスリクエスタ（標準の同期受付（Web サービス））の作成手順（SOAP 通信基盤） 738
サービスリクエスタ（標準の非同期受付（MDB（DB キュー））の作成手順 817
サービスリクエスタ（標準の非同期受付（MDB（WS-R））の作成手順 800
サービスリクエスタが使用するプロトコルと標準受付およびユーザ定義受付の種類 736
サービスリクエスタ作成の概要 736
サービスリクエスタの作成 735
サイズノードが表すデータサイズの単位をバイト数から桁数へ変更する 1207
サイズノードにサイズノード自身のサイズを含めた値を指定する 1204
サイズノードにビット列型のノードを指定する 1209
サイズノードの機能拡張 1204
サブシーケンスの定義方法 490
差分確認用ドキュメントが出力される出力ディレクトリの構成 146
差分確認用ドキュメント出力機能 144
差分確認用ドキュメント出力機能の概要 144
差分確認用ドキュメント出力機能の実行手順 146
差分確認用ドキュメント出力機能の実行例 147
差分確認用ドキュメント出力機能の注意事項 151
差分確認用ドキュメントの出力 144
参照できる HCSC コンポーネントの情報 716

し

システム全体と開発環境の関係 29
システムを開発する前に 31
実数 163
実装方式を決定する 1078
指定した値を代入する 549
指定文字列の有無，または指定文字列の開始を調査する 525
自動マッピング時の類似度の判定 518
シフト演算ファンクション 537
シフト演算を使用する 537
終了アクティビティ 350
出現回数の制御（JSON 形式データから XML スキーマを生成する場合） 1018
出力前の確認事項 138
順次処理アクティビティ 350
条件によって異なる値を出力する 546
小数点以下の端数を処理する 531
使用する SOAP モード 42
使用手順 689
上流設計ツール利用時の前提事項 1048
上流設計ツール利用時のトラブルシューティング 1051
上流設計ツールを利用したシステム開発の概要 1044
上流設計ツールを利用したシステム開発の流れ 1046
上流設計ツールを利用したシステムの開発 1044
上流設計ツールを利用したシステムの開発例 1071

す

推奨モード 44
数値演算ファンクション 530
数値型 190
数値フォーマット変換ファンクション 529
数値を演算する 530
数値をフォーマット変換する 529
スコープアクティビティ 332

スコープアクティビティのデフォルトハンドラ 258
スタブを取得する〔標準の同期受付（SessionBean）〕 786
スタブを生成する〔SOAP 受付（Web サービス/SOAP 通信基盤）〕 836
スタブを生成する〔標準の同期受付（Web サービス/SOAP 通信基盤）〕 741
ステータスの永続化 240
ステップ実行および再開 858

せ

整数 163
設計情報として出力できる情報 128
設計情報の出力 128
設計情報の出力に関する注意事項 143
設計情報の出力方法 139
接続できるマッピング線の数 615
設定できるプロパティ〔JSON 変換ツール〕 1016
セパレータを設定する 177
選択条件ノードを選択して選択条件を設定する 175
選択ファンクション 546
選択要素なしの選択構成の定義 1212
前提条件 33

そ

相関セット 244
相関セットによるプロセスインスタンスの識別 274
相関セットの新規定義 279
相関セットの定義 274
相関セットの定義方法 279
相関セットの有効範囲 278
相関セットを利用した定義の例 280
相関セットを割り当てられるアクティビティ 276
送信用共用キューを作成する〔標準の非同期受付（MDB（DB キュー））〕 821
総和ファンクション 533
ゾーン形式数値 164

た

待機アクティビティ 322
待機時間として指定できる文字列 323
代入アクティビティ 308
単純内容要素 172
単純内容要素を設定する 172

ち

注意事項（サービス定義のエクスポート） 487

て

定義済みビジネスプロセスを利用してビジネスプロセスを追加する 241
定義の流れ〔ビジネスプロセス〕 235
提供しているカスタムファンクション 683
停止から削除までの処理の流れ 727
定数ファンクション 549
ディレクトリ/ファイルのパスの長さに関する制限について 488
データ型の制御（JSON 形式のデータに変換する場合） 1019
データ項目の設定（COBOL85 言語の形式を使用する場合） 204
データ項目の設定（サービスプラットフォームで独自に拡張した形式を使用する場合） 208
データベースと Reliable Messaging の使用有無 40
データベースと Reliable Messaging の使用有無を確認する 50
データ変換 154
データ変換 API の機能 953
データ変換 API の詳細 961
データ変換 API の使用方法 981
データ変換 API の使用例 986
データ変換 API の注意事項 992
データ変換 API を使用したデータ変換 953
データ変換 API を使用したデータ変換の概要 953
データ変換アクティビティ 306
データ変換アクティビティに指定するマッピング定義の形式 467
データ変換定義 496
データ変換定義の注意事項 503
データ変換に必要なファイルおよび定義 492

データ変換の定義　491
データ変換の定義手順　496
テスト環境のアンセットアップ　63
テスト環境の運用に必要な情報　64
テスト環境のカスタマイズ　69
テスト環境の簡易セットアップ　51
テスト環境の起動と停止　63
テスト環境のセットアップ　53
テスト環境の前提ソフトウェア　54
デバッグの流れ　846
転送キューを作成する〔標準の非同期受付（MDB（WS-R））〕　802
添付ファイルを使用するための WSDL 定義〔SOAP1.1/1.2 併用モード〕　87
添付ファイルを使用するための WSDL 定義〔soap1.1 モード〕　86
電文の種類　154
電文の流れ　154
電文フォーマット　154
電文フォーマット（XML フォーマット定義ファイル）の作成方法　157
電文フォーマット（バイナリフォーマット定義ファイル）の作成方法　159
電文フォーマット定義ファイル　154
電文フォーマット定義ファイルの作成　493
電文フォーマットとデータ変換　154
電文フォーマットの作成　153
電文フォーマットの種類　156
電文フォーマットの変更　233
電文フォーマットファイルの参照ファイルのエクスポート　489
電文フォーマットを変更した場合の定義手順　502

と

同一の Name 要素を複数行に定義できるオブジェクト　678
途中終了アクティビティ　331

な

名前空間接頭辞オプション　682

に

入出力データが 0 バイトとなるケースの選択構成の定義　1212

の

ノード条件を設定する　581
ノード数取得ファンクション　540
ノード数を代入する　540
ノード調査ファンクション　542
ノードの有無を調査する　542
ノード名取得ファンクション　541
ノード名を代入する　541
ノード・タスクを詳細化する　1080

は

バイト列　166
バイト列型　166
バイト列型数値　162
バイナリデータ内に文字コードが混在する場合の対応　1224
バイナリデータの未定義コードを置換する文字の定義　1217
バイナリデータを作成する〔標準の非同期受付（MDB（DB キュー））〕　822
バイナリデータを受信用共用キューに設定する〔標準の非同期受付（MDB（DB キュー））〕　828
バイナリフォーマット定義の注意事項　198
バイナリフォーマット定義ファイル　156
バイナリフォーマット定義ファイルから XML スキーマファイルの生成　232
バイナリフォーマット定義ファイルの作成の流れ　159
バイナリフォーマット定義ファイルの生成方法　229
バイナリフォーマット定義ファイルのデータ型　160
バイナリフォーマット定義ファイルの文字コードの種類　160
バイナリフォーマット定義ファイルを検証する　180
バイナリフォーマット定義ファイルを新規作成する　170
バイナリフォーマット定義ファイルを編集する　180
配備から開始までの処理の流れ　719

配備定義　711
配備できるクラスタ（または単一の HCSC サーバ）　713
ハイレベル・ビジネスプロセスを作成する　1073
端数処理ファンクション　532
パック形式数値　164
パッケージング　710
パッケージングから配備定義までの流れ　708
パッケージングと配備定義の概要　708
パラメタを設定する〔標準の同期受付（SessionBean）〕　788
パラメタを設定する〔標準の同期受付（Web サービス/JAX-WS エンジン）〕　772
パラメタを設定する〔標準の同期受付（Web サービス/SOAP 通信基盤）〕　745
パラメタを設定する〔標準の非同期受付（MDB（DB キュー））〕　826
パラメタを設定する〔標準の非同期受付（MDB（WS-R））〕　804

ひ

ビジネスプロセスおよびアクティビティの定義情報を変更する　366
ビジネスプロセス実行コマンド使用時の前提条件　383
ビジネスプロセス全体の定義に関連する要素の変換〔BPEL1.1〕　1106
ビジネスプロセス全体の定義に関連する要素の変換〔BPEL2.0〕　1127
ビジネスプロセス追加後に設定する項目　241
ビジネスプロセスの概要を設計する　1073
ビジネスプロセスの検証　385
ビジネスプロセスの再実行を要求するサービスリクエスタの作成（SessionBean）　797
ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・JAX-WS エンジン）　780
ビジネスプロセスの再実行を要求するサービスリクエスタの作成（Web サービス・SOAP 通信基盤）　754
ビジネスプロセスの削除　402
ビジネスプロセスの追加　238
ビジネスプロセスの定義　234
ビジネスプロセスの定義情報を変更する　366
ビジネスプロセスのテスト　383
ビジネスプロセスのデバッグ　845
ビジネスプロセスのデバッグで実施できる操作　858
ビジネスプロセスのデバッグの開始　853
ビジネスプロセスのデバッグの終了　885
ビジネスプロセスのデバッグの準備　849
ビジネスプロセスのデバッグの手順　846
ビジネスプロセスの内容の定義　243
ビジネスプロセスのバージョンアップ　371
ビジネスプロセスの描画情報の指定　467
ビジネスプロセスの描画情報ファイルの出力有無　475
ビジネスプロセスの描画情報を指定する要素　459
ビジネスプロセスの編集　366
ビジネスプロセスの保存　365
ビジネスプロセスをインポートする　1090
ビジネスプロセスを指定する要素　413
ビジネスプロセスを出力する　1084
ビジネスプロセスを詳細化する　1078
ビジネスプロセスを新規に追加する　238
ビジネスプロセスを新規に追加する〔BPEL ファイルのインポート〕　239
ビジネスプロセスを新規に追加する〔未定義のビジネスプロセス〕　238
日付時刻型　168
ビット列　167
ビット列型　167
否定演算ファンクション　534
否定演算を使用する　534
評価版からの移行　1041
表形式 XML スキーマ定義ファイルの定義詳細　642
表形式 XML ファイルのインポート　641
表形式 XML ファイルの作成　621
標準電文　154
標準電文を作成する　157
標準の同期受付（SessionBean）に要求を出すサービスリクエスタ　784
標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（JAX-WS エンジン）　766

標準の同期受付（Web サービス）に要求を出すサービスリクエスタ（SOAP 通信基盤） 738
標準の非同期受付（MDB（DB キュー））に要求を出すサービスリクエスタ 816
標準の非同期受付（MDB（WS-R））に要求を出すサービスリクエスタ 800

ふ

ファンクションのコピー 592
ファンクションの貼り付け 594
ファンクション名の直接編集 617
ファンクション名の編集方法 617
ファンクションをコピーして貼り付ける 592
ファンクションを削除する 514
ファンクションを使用した値またはノードの加工 521
フォーマット情報を設定する〔CSV 以外の形式の場合〕 171
フォーマット情報を設定する〔CSV 形式の場合〕 179
フォーマットの変換方法〔DB アダプタで取得したデータのフォーマット変換例〕 1190
フォルト処理を定義する 252
フォルト送出アクティビティ 313
複合内容要素 172
複合内容要素を設定する 172
複数のノードセットのノードの数値を合計する 533
複数の文字列を連結する 522
符号付 2 進整数 165
符号無 2 進整数 165
部分文字列取得ファンクション 523
ブレークポイントの条件設定を変更する 850
ブレークポイントの設定 849
ブレークポイントをエクスポート/インポートする 851
ブレークポイントを削除する 850
ブレークポイントを追加する 849
ブレークポイントを無効にする 850
プロジェクトの管理 102
プロジェクトの管理とリポジトリの管理 101
プロジェクトを削除する 108
プロジェクトを作成する 102
プロセスインスタンスの識別 274
プロセスインスタンスの実行履歴を削除する 78
プロセスの属性を設定する 1079
プロパティの指定に関する注意事項〔JSON 変換ツール〕 1017
プロパティファイルの指定方法〔JSON 変換ツール〕 1016
プロパティを設定する 106
分岐アクティビティ 341

へ

並列処理アクティビティ 344
ヘッダ要素およびレコード要素の構成要素を設定する 179
ヘッダ要素およびレコード要素の出現回数を設定する 179
変換先ノードの要素から候補を選択して自動でマッピングする 513
変換先ノードやファンクションの入力値に XPath 式を指定する 561
変換テーブルの作成 551
変換テーブルの保存 552
変換テーブルを利用して値を変換する 550
変換動作の変更オプション〔JSON 変換ツール〕 1016
変換ファンクション定義ファイルの作成 597
変換元ノードの値にある値を付加する 557
変換元ノードの値を加工して変換先ノードにマッピングする 507
変換元ノードの値を倍にする 558
変換元ノードの値を変換先ノードに直接マッピングする 506
変換元ノードの範囲と変換先ノードの範囲を指定して自動でマッピングする 512
編集後のファンクション名の表示 618
編集できる定義ファイル 69
変数 244
変数の型 264
変数の更新 871
変数の種類 264
変数の定義 264
変数の定義方法 271
変数の表示 266

変数を割り当てられるアクティビティ 264
変数・相関セットの確認 871
変数・相関セットの定義 264

ほ

補償アクティビティ 316
補償処理を定義する 258
補償範囲の定義 317
補償ハンドラの使用 1049

ま

マッピング 505
マッピング情報 583
マッピングする 505
マッピング線とファンクションを見やすくする 516
マッピング線を削除する 514
マッピング対象とマッピング対象外 605
マッピング対象の範囲を限定する 517
マッピング定義 496
マッピング定義のインポート 638
マッピング定義のエクスポート 640
マッピング定義のエクスポート形式 473
マッピング定義のコピー 585
マッピング定義の設定 623
マッピング定義の登録 584
マッピング定義の保存 584
マッピング定義ファイルの作成（サービス定義 XML ファイル） 468
マッピング定義ファイルを利用する際の注意事項 468
マッピング定義をコピーする 583
マッピング定義をコピーする流れ 584
マッピング定義をコピーする場合の注意事項 588
マッピングできるノードおよびファンクションの対応 606
マッピングの条件 605
マッピングの注意事項 519
マッピング元の表示形式 515
マッピングを取り消す 514

み

ミドルレベル・ビジネスプロセスを作成する 1073

む

無操作アクティビティ 312

め

メッセージのスキーマを登録する 1093
メッセージ変換とシステム例外処理を追加する 1098

も

文字コード変換 UOC 99
文字コード変換 UOC の jar ファイルの開発 1160
文字コード変換 UOC を使用するための設定 1160
文字コード変換 UOC を利用した文字コード変換1160
文字コード変換の種類 170
文字数取得ファンクション 525
文字列 165
文字列型数値 161
文字列調査ファンクション 526
文字列の空白を除去する 527
文字列の文字数を代入する 524
文字列連結ファンクション 522

ゆ

ユーザが作成した任意の Java プログラムを呼び出す 559
ユーザ定義受付（呼出先固定）の一覧を指定する要素 438
ユーザ定義受付（呼出先選択）を指定する要素 458
ユーザ定義受付を追加する 1091
ユーザの設定およびスキーマの定義 36
ユーザマッピングテーブル 97

よ

要求電文 154
要求電文を作成する〔標準の同期受付（SessionBean）〕 789
要求電文を作成する〔標準の同期受付（Web サービス/JAX-WS エンジン）〕 772

要求電文を作成する〔標準の同期受付（Web サービス/SOAP 通信基盤）〕 746
要求電文を作成する〔標準の非同期受付（MDB（DB キュー））〕 826
要求電文を作成する〔標準の非同期受付（MDB（WS-R））〕 808
用語解説 1240
要素を定義する（CSV 以外の形式の場合） 171
要素を定義する（CSV 形式の場合） 178

り

リクエストの送信 383, 857
リポジトリ 110
リポジトリの管理 110
リポジトリの指定 110
リポジトリの初期化 111
リポジトリの設定 106
リポジトリの変更 112
リポジトリを設定する 110
利用できるサービス部品の種類と適用範囲 81

る

類似度の判定方法 586
ルート要素を設定する〔CSV 以外の形式の場合〕 176
ルート要素を設定する〔CSV 形式の場合〕 180
ルート要素を変更する 500

れ

例題〔DB アダプタで取得したデータのフォーマット変換例〕 1189
連携パスを設定して各要素のくり返し処理を連動させる 575

ろ

ローカル定義の単純内容要素を構成要素として設定する 174
ローカル変数 264
ログ出力アクティビティ 331
論理演算ファンクション 536
論理演算を使用する 535