WebSphere Business Integration Server
Express and Express Plus

マップ開発ガイド

4.3

お願い

本書および本書で紹介する製品をご使用になる前に、 551 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Business Integration Server Express バージョン 4.3、IBM WebSphere Business Integration Server Express Plus バージョン 4.3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： WebSphere Business Integration Server Express and Express Plus
Map Development Guide
4.3

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.8

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について xi
対象読者 xi
本書の使用法 xi
関連文書 xii
表記上の規則 xiii

本リリースの新機能 xv
リリース 4.3 の新機能. xv

第 1 部 マップ 1

第 1 章 マップ開発の概要 3
データ・マッピングについて 3
マップ: 詳細 5
マップ開発用のツール 8
マップ開発の概要 12

第 2 章 マップの作成 15
Map Designer Express の概要 15
マップの作成: 基本手順 33
マッピングの標準 58

第 3 章 マップの処理 61
マップのオープンとクローズ 61
マップ・プロパティー情報の指定 64
マップ文書の使用 67
マップの自動化の使用 72
マップの情報の検索 79
テキストの検索および置換 81
マップの印刷 82
オブジェクトの削除 82
実行順序の使用 85
ポリモアフィック・マップの作成 86
InterChange Server Express からのマップのインポートとエクスポート 87

第 4 章 マップのコンパイルとテスト 89
変換コードの検査 89
マップの検証 90
マップのコンパイル 91
マップ・セットのコンパイル 93
マップのテスト 94
拡張デバッグの実行 103
関係を含むマップのテスト 103
マップのデバッグ 109

第 5 章 マップのカスタマイズ 113
Activity Editor の概要 113
アクティビティー定義の処理 123
Java パッケージと他のカスタム・コードのインポート 174
Web サービスを Activity Editor にエクスポートする 180

変数の使用 184
その他の属性の変換方法. 190
マップ・インスタンスの再利用 202
例外処理 203
カスタム・データ検証レベルの作成 205
マップの実行コンテキストの理解. 207
子ビジネス・オブジェクトのマッピング 211
サブマップの使用の続き. 216
データベース照会の実行. 223

第 2 部 関係 243

第 6 章 関係の概要 245
関係とは 245
関係: 詳細 251
関係開発プロセスの概要. 257

第 7 章 関係定義の作成 259
Relationship Designer Express の概要. 259
関係定義の作成. 266
一致関係の定義. 267
参照関係の定義. 270
関係表スキーマの作成 271
関係定義と参加者定義のコピー 271
関係定義または参加者定義の名前変更 272
関係の拡張設定の指定 272
関係定義の削除. 277
関係の最適化 277

第 8 章 関係のインプリメント 281
関係のインプリメント 281
参照関係の使用. 282
単純一致関係の使用 287
複合一致関係の使用 300
子インスタンスの管理 309
動詞の設定 312
外部キー参照の実行 318
カスタム関係の維持 324
安全な関係コードの作成. 326
リレーションシップ・データベースの照会の実行 328
関係のロードとアンロード 339

第 3 部 マッピング API リファレンス 343

第 9 章 BaseDLM クラス 345
getDBConnection() 345
getName(). 347
getRelConnection() 348
implicitDBTransactionBracketing() 349
isTraceEnabled(). 350
logError()、logInfo()、logWarning() 350
raiseException() 352
releaseRelConnection() 354
trace() 355

第 10 章 BusObj クラス. 357
例外と例外タイプ 358
階層ビジネス・オブジェクトをトラバースするための構文 359
copy() 360
duplicate(). 360
equalKeys() 361
equals() 362
equalsShallow() 363
exists(). 363
getBoolean()、getDouble()、 getFloat()、getInt()、 getLong()、get()、 getBusObj()、 getBusObjArray()、
getLongText()、getString() 364
getLocale() 366
getType() 366
getVerb() 367
isBlank() 367
isKey(). 368
isNull() 368
isRequired() 370
keysToString() 370
set() 371
setContent() 372
setDefaultAttrValues() 373
setKeys() 373
setLocale() 374
setVerb() 374
setVerbWithCreate(). 375
setWithCreate() 375
toString() 376
validData() 377
使用すべきでないメソッド 378

第 11 章 BusObjArray クラス 379
addElement() 380
duplicate(). 381
elementAt() 381
equals() 382
getElements(). 382
getLastIndex() 383
max() 383
maxBusObjArray() 384
maxBusObjs() 385
min() 386
minBusObjArray() 387
minBusObjs(). 388
removeAllElements() 389
removeElement(). 389
removeElementAt() 390
setElementAt() 390
size() 391
sum() 391
swap() 392
toString() 392

第 12 章 CwDBConnection クラス 395
beginTransaction() 395
commit() 396

executePreparedSQL() 398
executeSQL() 399
executeStoredProcedure() 401
getUpdateCount() 402
hasMoreRows() 402
inTransaction() 403
isActive() 404
nextRow() 404
release() 405
rollBack() 406

第 13 章 CwDBStoredProcedureParam クラス 409
CwDBStoredProcedureParam() 409
getParamType() 410
getValue() 411

第 14 章 DtpConnection クラス 413
beginTran() 413
commit() 414
executeSQL() 415
execStoredProcedure() 416
getUpdateCount() 417
hasMoreRows() 418
inTransaction() 418
nextRow() 419
rollBack() 419

第 15 章 DtpDataConversion クラス 421
getType() 421
isOKToConvert() 423
toBoolean() 425
toDouble() 425
toFloat() 426
toInteger() 427
toPrimitiveBoolean() 427
toPrimitiveDouble() 428
toPrimitiveFloat() 429
toPrimitiveInt() 429
toString() 430

第 16 章 DtpDate クラス 433
DtpDate() 435
addDays() 437
addWeekdays() 438
addYears() 439
after() 439
before() 440
calcDays() 441
calcWeekdays() 441
get12MonthNames() 442
get12ShortMonthNames() 443
get7DayNames() 443
getCWDate() 443
getDayOfMonth() 444
getDayOfWeek() 444
getHours() 445

getIntDay() . . . 445
getIntDayOfWeek() . . . 445
getIntMilliSeconds() . . . 446
getIntMinutes() . . . 446
getIntMonth() . . . 446
getIntSeconds() . . . 447
getIntYear() . . . 447
getMSSince1970() . . . 448
getMaxDate() . . . 448
getMaxDateBO() . . . 449
getMinDate() . . . 451
getMinDateBO() . . . 452
getMinutes() . . . 453
getMonth() . . . 454
getNumericMonth() . . . 454
getSeconds() . . . 454
getShortMonth() . . . 455
getYear() . . . 455
set12MonthNames() . . . 456
set12MonthNamesToDefault() . . . 456
set12ShortMonthNames() . . . 457
set12ShortMonthNamesToDefault() . . . 457
set7DayNames() . . . 458
set7DayNamesToDefault() . . . 458
toString() . . . 458

第 17 章 DtpMapService クラス . . . 461
runMap() . . . 461

第 18 章 DtpSplitString クラス . . . 463
DtpSplitString() . . . 463
elementAt() . . . 464
firstElement() . . . 465
getElementCount() . . . 465
getEnumeration() . . . 466
lastElement() . . . 466
nextElement() . . . 467
prevElement() . . . 468
reset() . . . 468

第 19 章 DtpUtils クラス . . . 471
padLeft() . . . 471
padRight() . . . 471
stringReplace() . . . 472
truncate() . . . 473

第 20 章 IdentityRelationship クラス . . . 475
addMyChildren() . . . 475
deleteMyChildren() . . . 477
foreignKeyLookup() . . . 478
foreignKeyXref() . . . 480
maintainChildVerb() . . . 483
maintainCompositeRelationship() . . . 485
maintainSimpleIdentityRelationship() . . . 487
updateMyChildren() . . . 489

第 21 章 MapExeContext クラス 493
getConnName() 493
getInitiator() 494
getLocale() 495
getOriginalRequestBO() 496
setConnName() 496
setInitiator() 497
setLocale() 498
使用すべきでないメソッド 499

第 22 章 Participant クラス 501
Participant() 501
getBusObj()、getString()、getLong()、 getInt()、getDouble()、 getFloat()、 getBoolean() 503
getInstanceId() 504
getParticipantDefinition() 504
getRelationshipDefinition(). 505
set() 505
setInstanceId() 506
setParticipantDefinition() 506
setRelationshipDefinition() 507

第 23 章 Relationship クラス 509
addParticipant() 510
create(). 512
deactivateParticipant() 513
deactivateParticipantByInstance() 514
deleteParticipant() 516
deleteParticipantByInstance() 517
getNewID() 518
retrieveInstances() 518
retrieveParticipants() 520
updateParticipant() 521
updateParticipantByInstance() 522
使用すべきでないメソッド 523

第 24 章 UserStoredProcedureParam クラス 525
UserStoredProcedureParam() 526
getParamDataTypeJavaObj() 526
getParamDataTypeJDBC() 527
getParamIndex() 528
getParamIOType() 528
getParamName() 529
getParamValue() 530
setParamDataTypeJavaObj() 531
setParamDataTypeJDBC() 531
setParamIndex() 532
setParamIOType() 532
setParamName() 533
setParamValue() 534

第 4 部 付録 535

付録 A. メッセージ・ファイル 537
メッセージの位置 537
マップ・メッセージのフォーマット 541

ファイルの保守 542
メッセージ・ファイルを使用する操作 543

付録 B. 属性プロパティー 549

特記事項 551
プログラミング・インターフェース情報 552
商標 553

索引 555

本書について

IBM(R) WebSphere(R) Business Integration Server Express 製品および IBM(R) WebSphere(R) Business Integration Server Express Plus 製品に含まれるコンポーネントとしては、Interchange Server Express、関連の Toolset Express、CollaborationFoundation、およびソフトウェア統合アダプターのセットがあります。Toolset Express に含まれるツールは、ビジネス・プロセスの作成、変更、および管理に役立ちます。プリパッケージされている各種アダプターは、お客様の複数アプリケーションにまたがるビジネス・プロセスに応じて、いずれかを選べるようになっています。標準的な処理のテンプレートである CollaborationFoundation は、カスタマイズされたプロセスを簡単に作成できるようにするためのものです。

本書は、マップおよび関係の使用法の概要を示すとともに、Map Designer Express や Relationship Designer Express を使用してマップおよび関係を作成し変更する方法について説明します。

特に明記されていない限り、本書の情報は、いずれも、IBM WebSphere Business Integration Server Express と IBM WebSphereBusiness Integration Server Express Plus の両方に当てはまります。「WebSphere Business Integration Server Express」という用語およびそのバリアントは、この両方の製品を指します。

対象読者

本書は、ビジネス・オブジェクト定義またはマップを作成あるいは変更するコネクター開発者、コラボレーション開発者、および IBM WebSphere コンサルタントを対象としています。

本書の使用法

本書の構成は、以下のとおりです。

第 1 部: マップ

『第 1 章 マップ開発の概要』	マップおよび WebSphere Business Integration Express マッピング・ツールの概要を示します。
『第 2 章 マップの作成』	マップを作成および変更するための Map Designer Express の使用法の概要を示します。
『第 3 章 マップの処理』	マップの作成後に使用する Map Designer Express の拡張機能について説明します。
『第 4 章 マップのコンパイルとテスト』	マップを実行可能な形式にコンパイルして、マップの正確さを検証するためにテスト実行を行う方法について説明します。
『第 5 章 マップのカスタマイズ』	マップをインプリメントする方法について説明します。

第 2 部: 関係

『第 6 章 関係の概要』 WebSphere Business Integration Express がサポートする関係の種類や、このシステムが関係をインプリメントする方法などを含む、関係の概要を示します。
『第 7 章 関係定義の作成』 関係定義を作成および変更するための Relationship Designer Express の使用法の概要を示します。
『第 8 章 関係のインプリメント』 関係をインプリメントする方法について説明します。

第 3 部: マッピング API リファレンス

『第 9 章 BaseDLM クラス』 マッピング API のクラスのメソッドに関する参照ページがあります。
『第 10 章 BusObj クラス』
『第 11 章 BusObjArray クラス』
『第 12 章 CwDBConnection クラス』
『第 13 章 CwDBStoredProcedureParam クラス』
『第 14 章 DtpConnection クラス』
『第 15 章 DtpDataConversion クラス』
『第 17 章 DtpMapService クラス』
『第 18 章 DtpSplitString クラス』
『第 19 章 DtpUtils クラス』
『第 20 章 IdentityRelationship クラス』
『第 21 章 MapExeContext クラス』
『第 22 章 Participant クラス』
『第 23 章 Relationship クラス』
『第 24 章 UserStoredProcedureParam クラス』
『付録 A. メッセージ・ファイル』
『付録 B. 属性プロパティー』

関連文書

資料の完全セットは、すべての WebSphere Business Integration Server Express および WebSphere Business Integration Server Express Plus のインストールに共通の機能およびコンポーネントについて説明するとともに、特定のコンポーネントに関する参照資料も記載しています。

関連文書は、http://www.ibm.com/websphere/wbiserverexpress/infocenter でダウンロード、インストール、および表示することができます。

注: 本書の発行後に公開されたテクニカル・サポートの技術情報や速報に、本書の対象製品に関する重要な情報が記載されている場合があります。これらの技術情報や速報は、WebSphere Business Integration のサポート Web サイト (http://www.ibm.com/software/integration/websphere/support/) で参照できます。適切なコンポーネント領域を選択し、「Technotes (技術情報)」セクションと「Flashes (速報)」セクションを参照してください。

表記上の規則

本書では、以下の規則を使用します。

courier フォント	コマンド名、入力情報、システムが画面に出力した情報など、記述されたとおりの値を示します。
イタリック または**イタリック**	変数名、タイトル名、または初出語を示します。
青いアウトライン	オンラインで表示したときにのみ見られる青いアウトラインは、ハイパーリンクとなっている相互参照です。アウトライン内をクリックすることにより、参照先オブジェクトに飛ぶことができます。
ProductDir	製品がインストールされているディレクトリーを表します。

本リリースの新機能

このセクションでは、本書で取り上げる IBM WebSphere Business Integration Server Express、IBM WebSphere Business Integration Server Express Plus、およびマップや関係開発用の関連ツールにおける新機能と変更された機能について説明します。

リリース 4.3 の新機能

本書の最初のリリースです。

第 1 部 マップ

第 1 章 マップ開発の概要

この章では、データ・マッピングの概要を示し、マップのインプリメントに使用するツールを紹介して、マップ定義と関係定義について説明します。

この章の内容は次のとおりです。

- 『データ・マッピングについて』
- 5 ページの『マップ: 詳細』
- 8 ページの『マップ開発用のツール』
- 12 ページの『マップ開発の概要』

データ・マッピングについて

データ・マッピング は、データをあるアプリケーション固有の形式から別の形式に変換 (またはマッピング) するプロセスです。マッピングは、異なるアプリケーション間で情報を転送するプロセスの中心をなし、特定のアプリケーションに依存しないコラボレーション (ビジネス・プロセス) を提供します。アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクト間でデータをマッピングすることで、WebSphere は、「最善の組み合わせの」アプリケーションに使用できる環境を作成します。 WebSphere Business Integration システムは、マップを簡単に保守できるように、モジュラーおよび拡張可能なアーキテクチャーを提供します。

WebSphere マップ開発システムは、ビジネス・オブジェクト間のマッピングを、次の機能も含めて包括的にサポートします。

- データ値を、ソース・ビジネス・オブジェクトの 1 つ以上の属性から、宛先ビジネス・オブジェクトの 1 つ以上の属性に変換します。
- 同等であっても表現が異なるために直接変換できないデータ・エンティティー間の関係を設定および保守します。
- サード・パーティーのマッピング製品や照会を実行するためのデータベースなど、外部のマッピング・リソースにアクセスできるようにします。

異なるアプリケーション間にデータ・マッピングを設定すると、あるアプリケーションでのイベントの発生が、そのアプリケーションがマップされている他のアプリケーションでも実行されます。イベントが発生するのは、データが作成、検索、更新、または削除されたときです。

マッピングは、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクト間のデータの転送 (または変換) を定義するマップ を使用します。マップ開発環境では、データは、アプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクト、または汎用ビジネス・オブジェクトからアプリケーション固有のビジネス・オブジェクトにマップされます。表 1 に、必要なマッピングのタイプを示します。

表 1. マッピング要件

ビジネス・オブジェクトの方向	ソース・ビジネス・オブジェクト	宛先ビジネス・オブジェクト	マップのタイプ
コネクターからコラボレーション コラボレーションからコネクター	アプリケーション固有 汎用	汎用 アプリケーション固有	インバウンド・マップ アウトバウンド・マップ

例: 図 1 に、実行時にマッピングがどのように発生するかを示します。脚色された従業員管理コラボレーションを例として使用します。

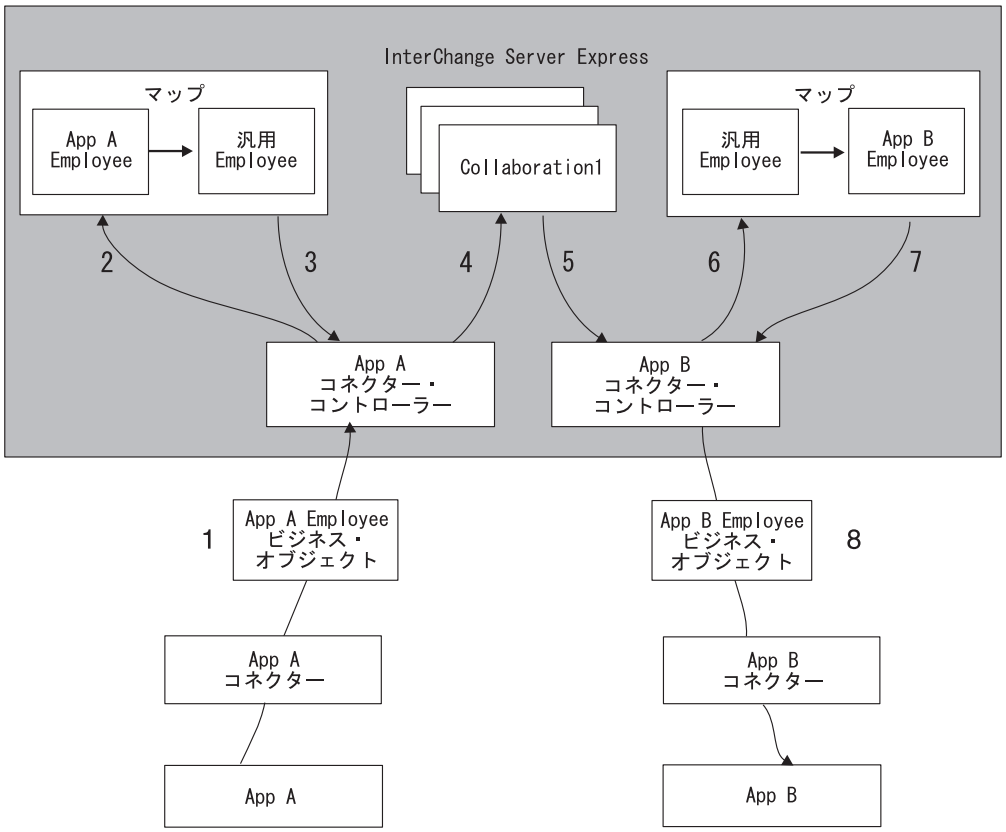

図 1. 実行時のデータ・マッピング

従業員管理コラボレーション (Collaboration1) は、ソース・コネクター (App A) から Employee ビジネス・オブジェクトを受け取り、Employee ビジネス・オブジェクトを宛先コネクター (App B) に送信します。図 1 に、発生するシーケンスを示します (以下の番号は、図中の番号に対応しています)。

1. App A でイベントが発生します。App A コネクターは、App A Employee ビジネス・オブジェクトを作成して、App A コネクター・コントローラーに送信します。
2. App A コネクター・コントローラーは、App A Employee ビジネス・オブジェクトを従業員管理コラボレーション (Collaboration1) に送信します。コラボレーションは、マッピングのために InterChange Server Express に常駐しています。要求には、コネクター構成で指定されたマップ名に基づいて、サーバーが使用する必要があるデータ・マップの名前が含まれます。

3. インバウンド・マップは、汎用 Employee ビジネス・オブジェクトを App A コネクター・コントローラーに戻します。
4. App A コネクター・コントローラーは、汎用 Employee ビジネス・オブジェクトへのサブスクリプションを持つコラボレーションを検査します。この場合、Collaboration1 にはサブスクリプションがあるため、コネクター・コントローラーは、ビジネス・オブジェクトを Collaboration1 に渡します。
5. コラボレーションは処理を実行してから、出力として別の汎用 Employee ビジネス・オブジェクトを作成します。このビジネス・オブジェクトは、App B コネクター・コントローラーに送信されます。
6. App B コネクター・コントローラーは、汎用ビジネス・オブジェクトを InterChange Server Express に送信して、App B Employee ビジネス・オブジェクトへのマッピングを要求します。
7. アウトバウンド・マップは、アプリケーション固有の Employee ビジネス・オブジェクトを App B コネクター・コントローラーに戻します。
8. App B コネクター・コントローラーは、App B Employee オブジェクトを App B コネクター・コントローラーに渡し、ビジネス・オブジェクトのデータが App B に渡されます。

この図は、次の 2 種類のマップが使用されていることを示します。

- コラボレーションで使用される、App A Employee ビジネス・オブジェクトから汎用 Employee ビジネス・オブジェクトへのインバウンド・マップ
- 汎用 Employee ビジネス・オブジェクトから App B Employee ビジネス・オブジェクトへのアウトバウンド・マップ

従業員データは、Application A から Application B の方向にのみ移動します。両方のアプリケーション間で両方向で従業員データを交換する場合は、2 つ以上のマップが必要です。

- Application B のアプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのインバウンド・マップ
- 汎用ビジネス・オブジェクトから Application A のアプリケーション固有のビジネス・オブジェクトへのアウトバウンド・マップ

マップ: 詳細

表 2 に示すように、マップは 2 つの部分からなるエンティティーで、マップ定義とランタイム・オブジェクトで構成されます。

マップ定義

マップ開発システムに対して、マップ定義 を使用してマップを定義します。マップ定義は、System Manager のプロジェクトに保管されます。Map Designer Express ツールは、マップ定義 (通常は単にマップと呼ばれます) の作成を支援するダイアログを提供します。完了したマップ定義を System Manager のプロジェクトに保管する処理も行います。

Map Designer Express を使用してマップ定義を作成する方法の詳細については、33 ページの『マップの作成: 基本手順』を参照してください。

マップ定義は、マップに関する次の情報を提供します。

- マップ名
- マップのソース・オブジェクトと宛先オブジェクト
- マップ変換

マップ定義名

マップ定義は、単なるテンプレート、またはマップの説明です。マップ定義は、あるビジネス・オブジェクトの属性を別のビジネス・オブジェクトに変換する方法についての情報を提供します。したがって、マップ定義の名前は、マップの方向と、マップが変換するビジネス・オブジェクトを識別する必要があります。

ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクト

マップは、1 つ以上のソース・ビジネス・オブジェクトと 1 つ以上の宛先ビジネス・オブジェクトで構成されます。ソース・ビジネス・オブジェクト は、変換されるビジネス・オブジェクトです。*宛先ビジネス・オブジェクト* は、ソース・ビジネス・オブジェクトのデータを使用して生成されたビジネス・オブジェクトです。

マップ変換

マップの残りの部分は、一連の変換ステップで構成されます。変換ステップ は、宛先属性の値を戻す Java コードのセグメントです。マップには、変換される宛先属性ごとに 1 つの変換ステップが含まれます。変換は Java コードとしてインプリメントされるため、Java ソース (.java) ファイルに保管されます。

表 2 に、宛先ビジネス・オブジェクトで実行できる変換の一部を示します。標準的な変換には、値の設定、移動、結合、分割、サブマップ、および相互参照があります。カスタム変換を作成するときは、グラフィカル関数ブロックを使用できるだけでなく、「関係」、「内容ベースのロジック」、「日付変換」、および「ストリング」変換の Java コードを使用することもできます。

表 2. マップの変換

変換	説明	詳細情報の参照先
標準的な変換	Map Designer Express でコードを自動生成できる変換	
値を設定	宛先属性の値を指定します	41 ページの『属性への値の指定』
移動 (コピー)	ソース属性を宛先属性にコピーします	43 ページの『ソース属性の宛先属性へのコピー』
結合	2 つ以上のソース属性を 1 つの宛先属性に結合します	44 ページの『属性の結合』
分割	ソース属性を 2 つ以上の宛先属性に分割します	46 ページの『属性の分割』
サブマップ	子ビジネス・オブジェクトのマップを呼び出します	48 ページの『サブマップを使用した変換』
相互参照	ビジネス・オブジェクトの一致関係を保守します	53 ページの『一致関係の相互参照』
カスタム変換	上記の標準的な変換以外の変換を作成します	54 ページの『カスタム変換の作成』

表 2. マップの変換 (続き)

変換	説明	詳細情報の参照先
関係	各アプリケーションが独自の形式でデータを保持しているため、直接マップできないビジネス・オブジェクトを関連付けます	281 ページの『第 8 章 関係のインプリメント』
内容ベースのロジック	ソース属性の内容に基づいて宛先属性を変換します	190 ページの『内容ベースのロジック』
日付変換	ソース属性の形式から宛先属性の形式に日付を変換します	195 ページの『日付の形式設定』
ストリング	大文字小文字変換やサブストリングの取得など、ストリングで基本的な変換を実行します	199 ページの『式ビルダーを使用したストリング変換』

ソース属性と宛先属性の間に明白な対応が存在する場合、変換ステップは、ソース値を宛先属性にそのままコピーします。その他の変換には、計算、ストリング操作、データ型変換、Java を使用してコーディングできるその他のロジックがあります。

図 2 に、一般的な属性変換の種類を示します。

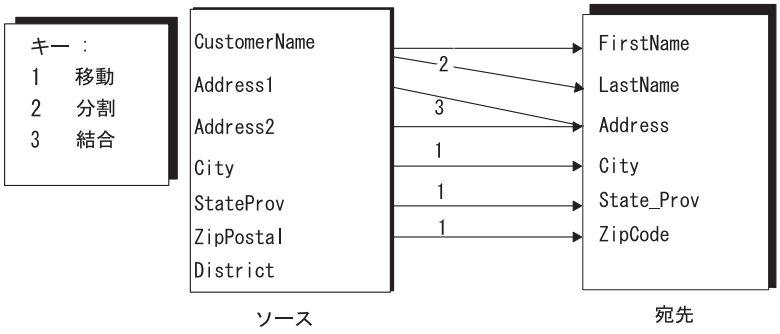

図 2. 一般的な属性変換

図 2 に示すように、ソース・ビジネス・オブジェクトの属性は、次のように処理されます。

- 宛先属性 (City、StateProv、ZipPostal) にコピーされます。
- 複数の宛先属性 (CustomerName) に分割されます。
- 1 つの宛先属性 (Address1、Address2) に結合されます。
- 宛先オブジェクトに同等の属性 (District) がない場合は無視されます。

値を属性にコピーする、値を 2 つ以上の属性に分割する、2 つ以上の値を 1 つの属性に結合するなど、単純な変換では、ステップを図で指定すると、Map Designer Express によって Java コードが生成されます。複雑な変換を行うには、グラフィカル・エディターを使用して変換をカスタマイズするか、独自の Java コードを作成します。

マップ・インスタンス

マップ定義は、マップのランタイム・インスタンス化 (マップ・インスタンス) を行うためのテンプレートです。マップの実行時に、マップ開発システムは、マップ定義と変換コードに基づいてマップのインスタンスを作成します。

各マップ・インスタンスは、以下の情報を提供します。

- BaseDLM クラスのメソッドによる、ロギング、トレース、接続、例外処理などの基本的な機能
- マップの実行コンテキスト

  詳細については、207 ページの『マップの実行コンテキストの理解』を参照してください。

マップ・インスタンスは、BaseDLM クラスのインスタンスによって、マッピング API で表されます。

マップ開発用のツール

表 3 に、マッピングに使用する 2 つのグラフィック設計ツールを示します。

表 3. データ・マッピング・システムの主要コンポーネント

設計ツール	マッピング・コンポーネント	説明
Map Designer Express	Map	Java コードを使用して、1 つ以上のソース・ビジネス・オブジェクトから 1 つ以上の宛先ビジネス・オブジェクトに属性を変換する方法を指定します。通常は、変換したいソース・ビジネス・オブジェクトごとに 1 つのマップを作成しますが、マップを複数のサブマップに分割することもできます。
Relationship Designer Express	Relationship	マップ開発システムの 2 つ以上のデータ・エンティティー間の関連を設定します。関係定義は通常、2 つ以上のビジネス・オブジェクトを関連付けます。ビジネス・オブジェクト間で同等でも、表示が異なるデータを変換するには、関係定義を使用します。例えば、ミシガン州の州コードは、あるアプリケーションでは MI、別のアプリケーションでは MICH として表示されることがあります。このデータは同等ですが、各アプリケーションで表示が異なります。ほとんどのマップでは、1 つまたは少数の関係定義を使用します。

これらのグラフィック・ツールは、Windows 2000 と Windows XP で動作します。したがって、これらのプラットフォームはマップ開発に使用されます。

表 4 に、マップ開発用にサポートされるその他のツールを示します。

表 4. マップ開発用のツール

ツール	説明
マッピング API	生成されたマッピング・コードをカスタマイズすることのできる一連の Java クラスです。

表 4. マップ開発用のツール (続き)

ツール	説明
System Manager	マップ・インスタンスと関係オブジェクトを構成するための、グラフィック・ウィンドウを提供します。

Map Designer Express

Map Designer Express は、マップを作成してコンパイルします。「ツール」メニューから「Map Designer Express」を選択すると、System Manager から Map Designer Express を起動できます。Map Designer Express を起動するその他の方法については、16 ページの『Map Designer Express の始動』を参照してください。 Map Designer Express は、マップ情報を表示するためのタブ・ウィンドウを提供します。このウィンドウには、「テーブル」タブ、「ダイアグラム」タブ、「メッセージ」タブ、または「テスト」タブのいずれかが表示されます。

図 3 に、Map Designer Express の「ダイアグラム」タブに表示されるマップを示します。

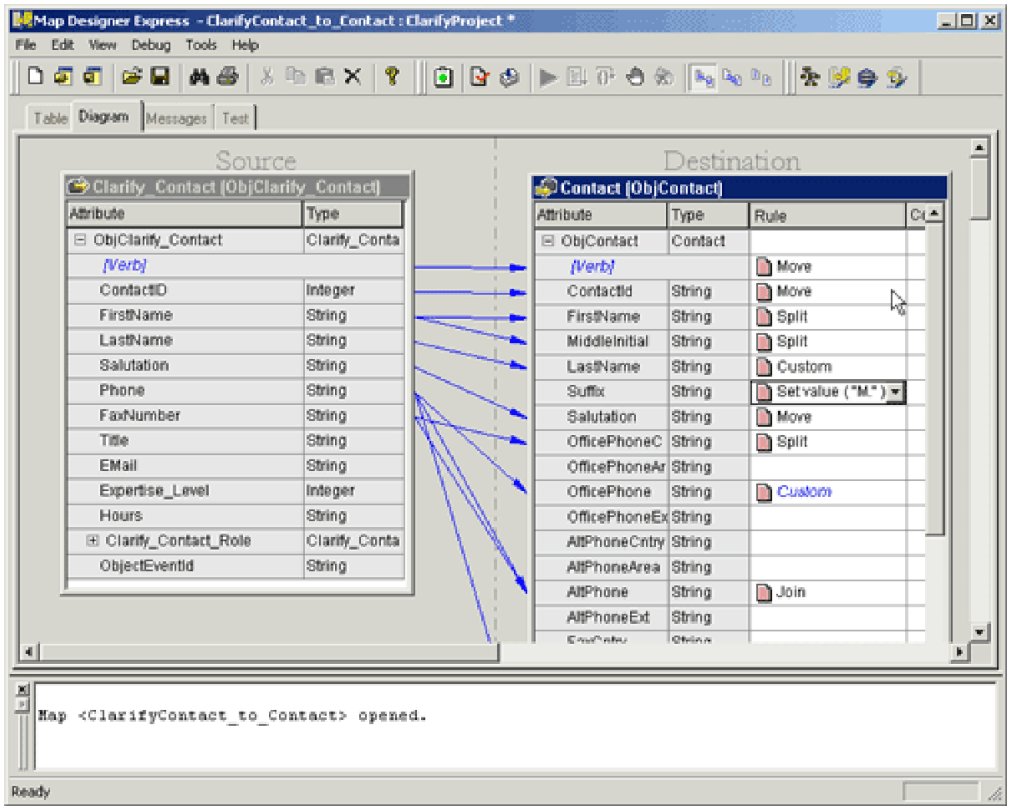

図 3. Map Designer Express

Map Designer Express を使用してマップを作成する方法の詳細については、 15 ページの『第 2 章 マップの作成』を参照してください。

Relationship Designer Express

Relationship Designer Express は、ランタイム関係インスタンス・データを保管する関係定義を作成します。「ツール」メニューから「Relationship Designer Express」を選択すると、System Manager から Relationship Designer Express を起動できます。図 4 に、Relationship Designer Express に表示される複数の関係を示します。

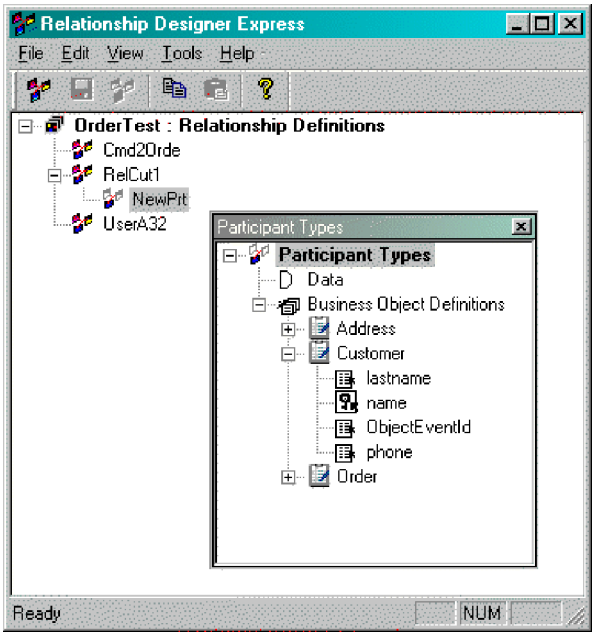

図 4. Relationship Designer Express

Relationship Designer Express の使用法の詳細については、 259 ページの『第 7 章 関係定義の作成』を参照してください。

マッピング API

多くの変換ステップは、標準の Java メソッドを使用してプログラムできます。変換ステップを簡単に作成するために、マップ開発システムでは、最も一般的なデータ変換状態の処理方法とともに、マッピング API (詳しくは343 ページの『第 3 部 マッピング API リファレンス』で説明します) を提供しています。マッピング API には、次のクラスがあります。

- DTP (Data Transformation Package) クラスは、ストリング操作、データ型変換、日付操作、サブマップ呼び出し、および SQL 照会実行のためのメソッドを提供します。クラスは以下のとおりです。
  - `DtpConnection` (使用すべきでない)
  - `DtpDataConversion`
  - `DtpDate`
  - `DtpMapService`
  - `DtpSplitString`
  - `DtpUtils`

- ビジネス・オブジェクト・クラスは、コラボレーション開発とマッピングの両方に使用されます。クラスは以下のとおりです。
  - BusObj
  - BusObjArray
- 関係管理クラスは、関係インスタンスを作成および管理するためのメソッドを提供します。クラスは以下のとおりです。
  - Participant
  - Relationship
  - IdentityRelationship
- データベース接続クラスは、SQL 照会実行のためのメソッドを提供します。クラスは以下のとおりです。
  - CwDBConnection
  - CwDBStoredProcedureParam
  - DtpConnection (使用すべきでない)
  - UserStoredProcedureParam (使用すべきでない)
- ユーティリティー・クラスは、エラー処理とデバッグ、およびマップの重要なランタイム値の設定を支援します。クラスは以下のとおりです。
  - BaseDLM
  - MapExeContext

System Manager

System Manager は、InterChange Server Express とリポジトリーへのインターフェースを提供するグラフィック・ツールです。System Manager は、マップを管理して、マップ定義を構成するための手段を提供します。以下のことができます。

- トレース・レベルやデータ妥当性検査レベルを含む、マップ定義の一般プロパティーを設定します。
- マップのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトを表示します。
- マップ定義をコンパイルします。

注: System Manager は、Map Designer Express を始動する方法を提供しています。詳細については、16 ページの『Map Designer Express の始動』を参照してください。

System Manager は、関係を管理する手段も提供します。以下のことができます。

- 関係表の位置を含む、関係の一般プロパティーを設定します。
- 関係の参加者を表示します。

注: System Manager は、Relationship Designer Express を始動する方法も提供しています。詳細については、259 ページの『Relationship Designer Express の始動』を参照してください。

マップ開発の概要

このセクションでは、マップ開発の概要を説明します。次の高レベルのタスクがあります。

1. マップ開発ソフトウェアをインストールして設定し、Java Development Kit をインストールします。
2. マップを設計してインプリメントします。

開発環境の設定

要件: 開発プロセスを開始する前に、次の事項を確認してください。

- マップ開発ソフトウェアが、アクセス可能なマシンにインストールされていること。

  マップ開発ソフトウェア・システムのインストールと始動の方法については、ご使用のシステムのインストール・ガイドを参照してください。

- IBM Java Development Kit (JDK) が製品 CD からインストールされていること。

  インストール済みの Java ディレクトリーが含まれるように、PATH 環境変数を更新します。パスを更新した後に InterChange Server Express を再始動します。

- System Manager が実行されていること。

  System Manager の始動については、ご使用のシステムのインストール・ガイドを参照してください。

- Map Designer Express が開いていて、System Manager に接続されていること。

  Map Designer Express を始動する方法の詳細については、15 ページの『Map Designer Express の概要』を参照してください。

マップの設計とインプリメント

マップを設計してインプリメントするには、以下の手順を実行する必要があります。

1. マップに含まれるすべてのビジネス・オブジェクトで使用されるデータ形式を調べます。
2. Map Designer Express 内でマップを作成します。
3. 必要な変換規則をカスタマイズします。
4. マップが必要とする Relationship Designer Express 内で関係を定義します。
5. 関係管理を実行するように、マッピング変換をカスタマイズします。
6. エラーおよびメッセージ処理をインプリメントします (該当する場合)。
7. .java ファイルとコンパイルされたコードを生成します。コンパイルされたコードが実行可能 Java クラスです。詳細については、13 ページの『マップ開発ファイル』を参照してください。
8. マップをテスト、デバッグし、必要であれば再コーディングを行います。

図 5 に、マップ開発の概要を示します。また、特定のトピックについての情報がどの章に記載されているかについても示します。

ヒント: マップの開発をチームで行う場合、マップ開発の主要なタスクを開発チームの各メンバーが並行して行うことができます。

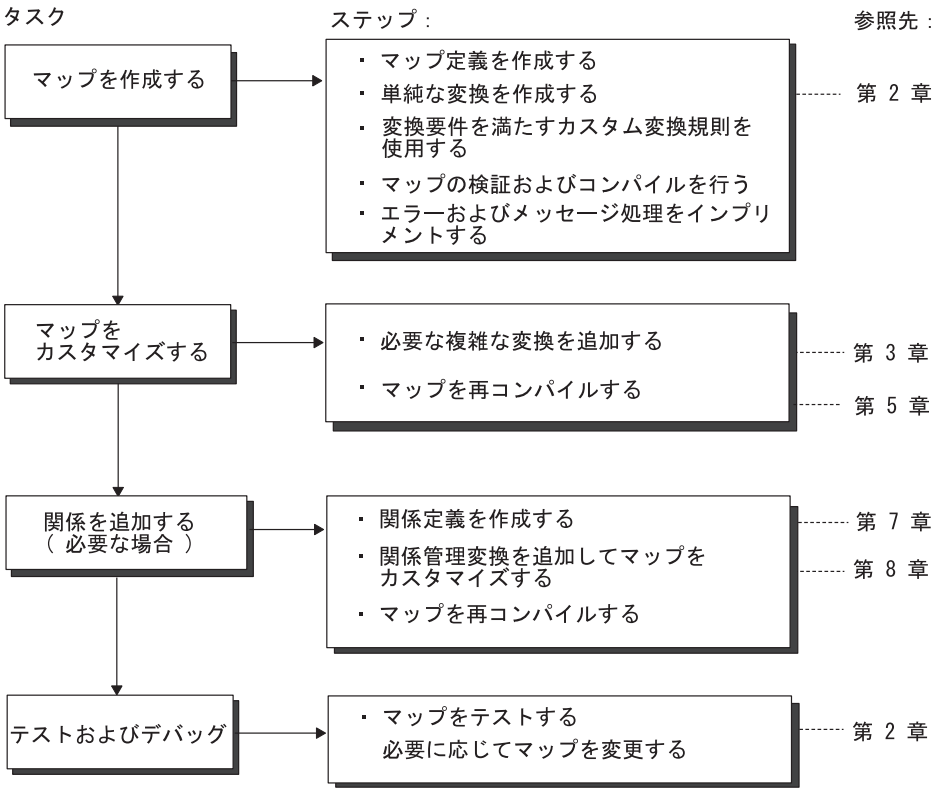

図 5. マップ開発タスクの概要

マップ開発ファイル

次の情報は、マップの基本を形成します。

- マップをコンパイルするときに、Map Designer Express は、2 つのタイプのファイル (.java、.class) を生成するか、マップにマップ指定のメッセージが定義されている場合は、オプションのメッセージ・ファイル (.txt) を生成します。これらのファイルは、System Manager のプロジェクトに保管されます。
- Map Designer Express は、マップを System Manager のプロジェクトに保管するときにマップ定義を生成します。このマップ定義には、マップに関する一般情報 (マップ・プロパティーなど) と、宛先属性がマップされる方法に関する情報が含まれます。

重要: *mapname*.java ファイルを変更しないでください。このファイルを変更しても、変更はマップ設計には反映されません。マップ設計は、System Manager のプロジェクトに保管されます。したがって、これらの変更は Map Designer Express では編集できません。Map Designer Express は、マップ定義のみを読み取ります。

Relationship Designer Express も、関係定義を XML 形式で System Manager に保管します。配置時に、関係のランタイム・インスタンス・データを含めるために、System Manager はリレーションシップ・データベースに表スキーマを作成します。関係ごとに、すべての関係表の位置を指定することができます。これらの表のデフォルトの位置は、IBM WebSphere Business Integration Server Express リポジトリーです。

表 5 に、Map Designer Express が生成できるファイルのタイプ (.java、.class、.cwm、 .bo、.txt) と、System Manager ワークプレースを基準とした相対的な位置を示します。

表 5. マップ・ファイルのタイプ

ファイル・タイプ	説明	System Manager ワークスペースを基準とした相対的な位置
.java	マップをコンパイルするときに、Map Designer Express によって作成された、生成済みの Java コード。	ProjectName¥Maps¥Src に保管されます。
.class	マップをコンパイルするときに、Map Designer Express で作成された、コンパイル済みの Java コード。	ProjectName¥Maps¥Classes に保管されます。
.cwm	マップ定義を保管するときに、Map Designer Express で生成されたマップ定義ファイル。	System Manager に「保管」するときに、ProjectName¥Maps に保管されます。
.bo	テスト実行データの保管とロード、およびテスト実行結果の保管に使用されるプレーン・テキスト・ファイル。	これらのファイルは、任意の位置に保管できます。
.txt	マップをコンパイルするときに、「Message」タブの情報から Map Designer Express によって作成されたメッセージ・ファイル。	ProjectName¥Maps¥Messages に保管されます。

第 2 章 マップの作成

この章では、Map Designer Express の概要と、Map Designer Express を使用してマップを作成する方法について説明します。

注: この章では、マップ とマップ定義 という用語を同義として頻繁に使用します。マップ という用語は、マップ定義 (Map Designer Express を介してアクセスされる) を指しています。

この章の内容は次のとおりです。

- 『Map Designer Express の概要』
- 33 ページの『マップの作成: 基本手順』
- 58 ページの『マッピングの標準』

WebSphere Business Integration システムがマップを使用する仕組みの背景情報については、3 ページの『第 1 章 マップ開発の概要』を参照してください。

Map Designer Express の概要

Map Designer Express は、マップを作成および変更するためのグラフィカルな開発ツールです。マップ とは、宛先ビジネス・オブジェクトの各属性の値を計算する方法を定義する、一連の変換ステップのことです。マップの作成とは、変換したい各宛先属性に変換ステップを指定するプロセスです。

Map Designer Express を使用すると、ソース属性を同一のデータ型の宛先属性へドラッグ・アンド・ドロップによって対話式にコピーする、などの簡単な変換ステップを指定できます。Map Designer Express は、変換の実行に必要な Java コードを自動的に生成します。

ソース属性を複数の宛先属性に分割したり、複数のソース属性を 1 つの宛先属性に結合したりというその他の一般的な変換を支援するために、Map Designer Express はどの属性を分割するか、結合するかを示す区切り文字などの情報を入力するよう要求し、その後必要な Java コードを生成します。さらに複雑な変換を指定するには、Activity Editor を使用してカスタム変換規則にアクティビティーをグラフィカルに定義するか、Activity Editor のウィンドウで直接 Java コードを変更するか、または独自の変換ステップを最初から作成します。

このセクションには、Map Designer Express の概要を説明する以下のトピックがあります。

- 16 ページの『Map Designer Express の始動』
- 16 ページの『プロジェクトでの作業』
- 17 ページの『Map Designer Express のレイアウト』
- 22 ページの『設定の割り当て』
- 25 ページの『メインウィンドウのカスタマイズ』
- 27 ページの『Map Designer Express の機能の使用法』

Map Designer Express の始動

Map Designer Express を起動するには、次のいずれかを行います。

- System Manager から、以下のアクションのいずれかを実行します。
  - 「ツール」メニューから「Map Designer Express」を選択します。
  - プロジェクトのマップ・フォルダーをクリックし、System Manager のツールバーで Map Designer Express アイコンを使用可能にします。その後、Map Designer Express アイコンをクリックします。
  - プロジェクト内のマップ・フォルダーを右マウス・ボタンでクリックし、コンテキスト・メニューから「新規マップの作成」を選択します。
  - 1 つのマップを右マウス・ボタンでダブルクリックすると、選択されたマップが開いた状態で Map Designer Express が始動します。
- Business Object Designer Express、Relationship Designer Express、Process Designer Express などの開発ツールから、次のいずれかのアクションを実行します。
  - 「ツール」メニューから「Map Designer Express」を選択します。
  - プログラム・ツールバーで、「Map Designer Express」ボタンをクリックします。

  制約事項: Process Designer Express は、WebSphere Business Integration Express Plus でのみ使用可能な開発ツールです。
- システム・ショートカットを使用して、以下のように順に選択していきます。

  ```
  Start > Programs > IBM WebSphere Business Integration
  Express > Toolset Express > Development > Map Designer Express
  ```

要確認: Map Designer Express が System Manager に保管されているマップにアクセスできるようにするには、Map Designer Express が System Manager のインスタンスに接続されている必要があります。前述の手順では、System Manager がすでに始動していることを想定しています。System Manager がすでに実行されている場合は、Map Designer Express は自動的に接続します。

Map Designer Express が固有のアプリケーション・ウィンドウ内に表示されます。複数のマップを編集したいときは、一度に複数の Map Designer Express のインスタンスを起動できます。

プロジェクトでの作業

Map Designer Express は、*プロジェクト・ベースで* System Manager に格納されたマップの表示、編集、および変更を行うことができます。*プロジェクト* は、管理や展開のためにエンティティーを論理的にグループ化したものにすぎません。System Manager では複数のプロジェクトを作成できます。

Map Designer Express は、System Manager への接続を確立すると、現在のプロジェクトで定義されているビジネス・オブジェクトのリストを取得します。Business Object Designer Express を使用してビジネス・オブジェクトの追加または削除を行うと、System Manager は Map Designer Express に通知し、ビジネス・オブジェクト定義のリストが動的に更新されます。

マップで作業するには、「プロジェクト」ダイアログの「Open a Map」でプロジェクトの名前を入力することにより、マップが存在するプロジェクトを選択する必要があります。 他のプロジェクトに切り替える前に、現在のプロジェクトで変更したマップを保管する必要があります。 プロジェクトからマップをオープンするには、62 ページの『System Manager のプロジェクトからのマップのオープン手順』を参照してください。プロジェクトにマップを保管するには、55 ページの『マップのプロジェクトへの保管』を参照してください。

Map Designer Express のレイアウト

最初にマップを指定せずに Map Designer Express を開いたときは、Map Designer Express のタブ・ウィンドウは空で出力ウィンドウは表示されません。既存のマップを開いたとき、Map Designer Express ウィンドウのタブ・ウィンドウにマップ・タブが表示されます。

表 6 に、Map Designer Express のメインウィンドウの各コンポーネントを説明します。

表 6. Map Designer Express ウィンドウのコンポーネント

ウィンドウの領域	説明	詳細情報の参照先
メニュー	Map Designer Express の機能にアクセスするためのオプションがあります。	27 ページの『Map Designer Express のプルダウン・メニュー』
ツールバー	実際には 3 つのツールバーに分かれており、それぞれに Map Designer Express の機能にアクセスするためのボタン群があります。	31 ページの『Map Designer Express のツールバー』
Map Designer Express タブ・ウィンドウ	4 つのマップ・タブのうちのいずれかに、開いているマップについてのマップ情報が表示されます。	18 ページの『「テーブル」タブ』 20 ページの『「ダイアグラム」タブ』 21 ページの『「メッセージ」タブ』 21 ページの『「テスト」タブ』
出力ウィンドウ	マップのコンパイルの結果とその他の状況メッセージが表示されます。 Map Designer Express が状況メッセージを生成したときに出力ウィンドウが表示されていない場合、このウィンドウが自動的に開きます。出力ウィンドウの内容は、「表示」メニューの「出力をクリア」オプションを使って消去できます。 **ヒント:** 「表示」メニューの「出力ウィンドウ」オプションを使用すると、出力ウィンドウ・ペインを Map Designer Express のメインウィンドウの一部として表示するかどうかを制御できます。	該当なし
ステータス・バー	Map Designer Express の状況メッセージが表示されます。 **ヒント:**「表示」メニューの「ステータス・バー」オプションを使用すると、ステータス・バーを Map Designer Express ウィンドウの一部として表示するかどうかを制御できます。	該当なし

以降の各セクションで、Map Designer Express のタブ・ウィンドウに表示される各タブの一般的なレイアウトについて説明します。

「テーブル」タブ

Map Designer Express の「テーブル」タブには、すべてのマッピング属性と変換を示したマッピング情報が表形式で表示されます。

「テーブル」タブには、以下の領域があります。

- 属性変換テーブル
- ビジネス・オブジェクト・ペイン

属性変換テーブル: 属性変換テーブルでは、マップに関連した変換すべてを表形式で示します。表 7 にこのテーブルの各列を示します。

表 7. 属性変換テーブルの各列

列名	説明
実行順序	宛先属性の実行順序。 このテーブルの最後に変換を追加すると、Map Designer Express が自動的にその変換の実行順序をそのテーブル内の最後と指定します。「実行順序」フィールドに希望の順序番号を入力すると、属性の実行順序を変更することができます。 **注:** **「マップを定義: 実行順序を自動的に調整」**オプションを使用すると、Map Designer Express による宛先属性の実行順序の処理方法を指定することができます。デフォルトでは、このオプションは使用不可です。オプションを使用可能にすると、ほかの属性の実行順序は Map Designer Express により自動的に調整されます。このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。
ソース属性	変換のソース属性の名前。 このフィールドにはコンボ・ボックスがあり、すべてのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトがリストされ、それぞれの属性がその下に表示されます。このリストから該当するソース属性をクリックします。コンボ・ボックス・リストの「複数の属性」のエントリーをクリックすると、複数のソース属性を選択できます。Map Designer Express が「複数の属性」ダイアログを表示し、そのダイアログから複数の属性を選択することができます。 **注:** **「マップを定義: 完全な属性パスを表示」**オプションを使用すると、Map Designer Express によるソース属性名の表示方法を指定することができます。デフォルトでは、このオプションは使用不可になっており、Map Designer Express はすべてのソース属性名を *...AttrName* と表示します。このオプションを使用可能にした場合、Map Designer Express は完全な属性パス Obj*SrcBusObj* を表示します。*AttrName*. このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。
ソース・タイプ	ソース属性のデータ型。 このフィールドは読み取り専用です。

表 7. 属性変換テーブルの各列 (続き)

列名	説明
宛先属性	変換の宛先属性の名前。 このフィールドにはコンボ・ボックスがあり、すべてのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトがリストされ、それぞれの属性がその下に表示されます。このリストから該当する宛先属性をクリックします。 **注:** 「マップを定義: 完全な属性パスを表示」オプションを使用すると、Map Designer Express による宛先属性名の表示方法を指定することができます。デフォルトでは、このオプションは使用不可になっており、Map Designer Express はすべての宛先属性名を *...AttrName* と表示します。このオプションを使用可能にした場合、Map Designer Express は完全な属性パス Obj*DestBusObj* を表示します。*AttrName*. このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。
宛先タイプ	宛先属性のデータ型。 このフィールドは読み取り専用です。
変換規則	この属性の変換ステップのための変換規則とコード。 このフィールドにはコンボ・ボックスがあり、以下の標準の変換がリストされます。 • なし (変換なし) • 結合 • 移動 • 分割 • 値を設定 • サブマップ • 相互参照 • カスタム このリストから該当する変換をクリックし、フィールドにその変換を入力します。詳細については、41 ページの『標準の属性変換の指定』を参照してください。
コメント	属性の変換についての情報を伝える説明。 59 ページの『属性のコメント・フィールドでのコメントの設定』を参照してください。

「テーブル」タブからのマップの定義手順: 「テーブル」タブからマップを定義するには、以下の一般的な手順に従ってください。

1. 「ソース属性」列の中の空のセルをクリックします。使用可能なコンボ・ボックスから、変換するソース属性をクリックします。
2. 「宛先属性」列の中のこれに対応するセルをクリックします。使用可能なコンボ・ボックスから、宛先属性をクリックします。
3. 「変換規則」列の中のこれに対応するセルをクリックします。この列には、コンボ・ボックスがあります。
   - 標準の変換 (結合、移動、分割、値を設定、サブマップ、相互参照) の場合は、リストから関連するオプションを選択します。Map Designer Express がこれら標準の変換用のコードを生成します。このコードは、必要に応じてカスタマイズすることが可能です。詳細については、41 ページの『標準の属性変換の指定』を参照してください。

- このコンボ・ボックスにはない 変換を実行する場合は、リストから「カスタム」を選択してから、Activity Editor でカスタム Java コードを追加してください。詳細については、54 ページの『カスタム変換の作成』を参照してください。

4. 「コメント」列の中のこれに対応するセルをクリックします。詳細については、59 ページの『属性のコメント・フィールドでのコメントの設定』を参照してください。

ビジネス・オブジェクト・ペイン: ビジネス・オブジェクト・ペインには、マップに関連したソース・ビジネス・オブジェクトおよび宛先ビジネス・オブジェクトすべてがリスト形式で表示されます。左側にソース・ビジネス・オブジェクトが、右側に宛先ビジネス・オブジェクトが表示されます。マップに一時ビジネス・オブジェクトが含まれている場合は、ビジネス・オブジェクト・ペインに、「ソース・ビジネス・オブジェクト」、「一時ビジネス・オブジェクト」、「宛先ビジネス・オブジェクト」の 3 つの領域があります。

ヒント: 「表示」メニューの「ビジネス・オブジェクト・ペイン」オプションを使用すると、ビジネス・オブジェクト・ペインを「テーブル」タブの一部として表示するかどうかを制御できます。

「ダイアグラム」タブ

Map Designer Express の「ダイアグラム」タブには、変換の定義と見直しのためのドラッグ・アンド・ドロップによるインターフェースがあります。ウィンドウの右側に表示されるマップ・ワークスペース内でマップを表示し、設計します。

「ダイアグラム」タブには、以下の領域があります。

- ビジネス・オブジェクト・ブラウザー。ウィンドウの左端にあるプロジェクト・ペイン内に表示されます。このブラウザーは、Map Designer Express が System Manager に接続されると、System Manager 内のプロジェクトの中のビジネス・オブジェクトを、階層形式でリストします。ビジネス・オブジェクト・ブラウザー内のビジネス・オブジェクトのリストを最新表示するには、ビジネス・オブジェクト・ブラウザーの中で右マウス・ボタンをクリックしてコンテキスト・メニューから「すべて最新表示」を選択します。 Map Designer Express は System Manager に照会して、ビジネス・オブジェクト・ブラウザーを最新のビジネス・オブジェクトで更新します。

  注: System Manager 内のプロジェクトでビジネス・オブジェクトの追加または削除を行うと、System Manager がビジネス・オブジェクト定義リストを動的に更新します。

  ヒント: 「表示」プルダウン・メニューの「Project Pane」オプションを使用すると、ビジネス・オブジェクト・ブラウザーをダイアグラム表示の一部として表示するかどうかを制御できます。
- マップ・ワークスペース。現行マップに関する情報を常に表示します。

  マップを開いたとき、マップ・ワークスペースには、そのマップに使われているソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトそれぞれのビジネス・オブジェクト・ウィンドウが表示されます。それぞれのビジネス・オブジェクト・ウィンドウには、そのビジネス・オブジェクトに定義されている属性が

すべてまたは一部 (現在選択されている表示モードによる) リストされます。宛先ビジネス・オブジェクトまたは一時ビジネス・オブジェクトの場合、ビジネス・オブジェクト・ウィンドウには、属性に関連した変換規則とコメントもリストされます。マップ・ワークスペースでは、マップに変換を追加したり、削除したり、マップの変換を変更したりすることができます。属性と属性をつなぐ線は、属性間の変換を表します。

ヒント:「表示」>「ダイアグラム」サブメニューの各オプションを使用すると、「ダイアグラム」タブ内にソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトのどちらの属性を表示するかを制御することができます。このサブメニューでは、すべての属性を表示するか、リンクされている (マップされている) 属性のみを表示するか、あるいはリンクされていない (マップされていない) 属性のみを表示するかを選択できます。

「メッセージ」タブ

「メッセージ」タブには、マップのメッセージが表示されます。メッセージは、メッセージ ID とその ID に関連付けられているメッセージ・テキストから構成されています。

「メッセージ」タブは、2 つのペインに分かれています。上部のペインはメッセージ・グリッドで、「メッセージ ID」列、「メッセージ」列、「説明」列 (メッセージ・ファイル全般に関するコメント) の 3 つの列から構成されています。下部のペインまたは「記述」ペインは、プレーン・テキストを入力するためのものです。「記述」ペインでテキストを入力すると、入力したテキストが生成されるメッセージ・ファイルの先頭にコメントとして追加されます。Map Designer Express は、マップのメッセージに加えられた変更内容はすべて System Manager のプロジェクト内に保管します。

メッセージおよびメッセージの使用法の詳細については、537 ページの『付録 A. メッセージ・ファイル』を参照してください。メッセージの形式については、541 ページの『マップ・メッセージのフォーマット』を参照してください。

ユーザーが新しいマップをコンパイルすると、Map Designer Express により「メッセージ」タブに入力された情報に基づいて、外部メッセージ・ファイルが生成されます。このメッセージ・ファイルは、メッセージ・ディレクトリーに保管されます。

重要: マップのメッセージに対する変更はすべて、Map Designer Express の「メッセージ」タブを通して行ってください。生成されたメッセージ・ファイルに変更を加える場合、外部テキスト・エディターを*使用しない*でください。外部エディターから加えた変更内容は、プロジェクトのマップ定義に保管*されない* ため、Map Designer Express からは*不可視* となります。また、このような変更は、次回そのマップをコンパイルする時に上書きされてしまいます。

「テスト」タブ

「テスト」タブには、マップをテストし、その結果を表示するためのインターフェースがあります。このタブでは、変換が正しく動作するかどうかを検証するためのテストを実行できます。

「テスト」タブには、以下の領域があります。

- テスト・パス・ダイアグラム

  ウィンドウ上部にあるテスト・パス・ダイアグラムは、マップ・テストを一連のアイコン群の形で示します。

  – 「ソース・テスト・データ」矢印は、マップの変換の向きを示し、当該マップ・テストに関係しているソース・ビジネス・オブジェクトのビジネス・オブジェクト・タイプのラベルが付いています。
  – 「マップ」アイコンは、テストの対象となっている現在開いているマップを表します。
  – 「宛先テスト・データ」矢印は、マップの変換の向きを示し、当該マップ・テストの結果として生成される宛先ビジネス・オブジェクトのビジネス・オブジェクト・タイプのラベルが付いています。

- 「ソース・テスト・データ」ペイン

  左下のウィンドウにあるソース・テスト・データ領域には、当該マップに関係するソース・ビジネス・オブジェクトの属性が階層形式でリストされます。ソース・ビジネス・オブジェクトのそばにある正記号 (+) をクリックすると、ソース・ビジネス・オブジェクトが展開します。この領域には、ソース・ビジネス・オブジェクト用のテスト・データを入力します。

- 「宛先テスト・データ」ペイン

  右下のウィンドウにある宛先テスト・データ領域には、マップの結果として生成される宛先ビジネス・オブジェクトの属性が階層形式でリストされます。ビジネス・オブジェクトのそばにある正記号 (+) をクリックすると、ソース・ビジネス・オブジェクトが展開します。この領域には、宛先ビジネス・オブジェクトのテスト結果のデータが表示されます。

注: Map Designer Express は、マップのテスト実行の結果を出力ウィンドウに表示します。

「テスト」タブの使用法の詳細については、94 ページの『マップのテスト』を参照してください。

設定の割り当て

「設定」ダイアログでは、Map Designer Express ツールの動作をカスタマイズできます。「設定」ダイアログを表示するには、以下のようにします。

- 「表示」メニューから「設定」を選択します。
- キーボード・ショートカット `Ctrl+U` を使用します。

図 6 は「設定」ダイアログを示しています。

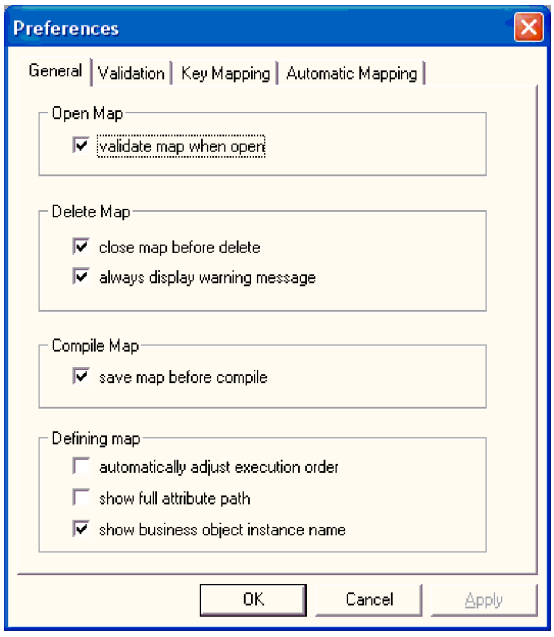

図 6. 「設定」ダイアログ

Map Designer Express は各設定値を Windows レジストリーに保管します。したがって、現行の Map Designer Express のセッションだけでなく、将来のセッションでも有効です。「設定」ダイアログには、以下のタブがあります。

- その他
- 「検証」
- 「キー・マップ」
- 自動マッピング

一般的な設定の指定

「設定」ダイアログの「一般」タブには、Map Designer Express によるマップの管理方法としてユーザーが指定できる一般的な設定が表示されます。

表 8. Map Designer Express の一般的な設定

一般的な設定	説明	詳細情報の参照先
マップを開く		
開くときにマップを検証	このオプションを使用可能にした場合、Map Designer Express はマップを開く時点でマップを検証します。 **推奨:** マップが使用しているビジネス・オブジェクトに 1,000 を超えるような多数の属性がある場合に、このオプションを使用可能にすると、マップを開くのに時間がかかることがあります。このようなケースで、この状況が望ましくない場合は、このオプションを使用不可にしてください。	61 ページの『マップのオープン』

表 8. Map Designer Express の一般的な設定 (続き)

一般的な設定	説明	詳細情報の参照先
マップを削除		
削除後にマップを閉じる	このオプションを使用可能にした場合、Map Designer Express は必ず現在開いているマップを閉じてから「マップを削除」ダイアログを表示します。	84 ページの『マップの削除手順』
常に警告メッセージを表示	このオプションを使用可能にした場合、Map Designer Express は必ず確認を表示してからマップを削除します。	84 ページの『マップの削除手順』
マップをコンパイル		
コンパイル前にマップを保管	このオプションを使用可能にした場合、Map Designer Express は必ず現行マップを System Manager のプロジェクトに保管してから、マップをコンパイルします。	91 ページの『マップのコンパイル』
マップを定義		
実行順序を自動的に調整	このオプションを使用可能にした場合、既存の属性の実行順序が変更されたとき、Map Designer Express が「テーブル」タブの宛先属性の実行順序の番号を自動的に変更します。	85 ページの『実行順序の使用』
完全な属性パスを表示	このオプションを使用可能にした場合、Map Designer Express は「テーブル」タブ内にソース属性と宛先属性の名前の完全な属性パスを表示します。	18 ページの『「テーブル」タブ』
ビジネス・オブジェクト・インスタンス名を表示	このオプションを使用可能にした場合、Map Designer Express はソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの名前に加えて それぞれの変数名を表示します。このオプションを使用不可にした場合、Map Designer Express は「テーブル」タブと「ダイアグラム」タブの両方のビジネス・オブジェクト変数の名前を省略します。	186 ページの『ビジネス・オブジェクト変数の変更手順』

検証の指定

「設定」ダイアログの「検証」タブには、Map Designer Express がマップの保管時にマップの検証を実行するように選択できるオプションがあります。オプションは以下のとおりです。

- 動詞がマップされていない場合に警告を表示する
- キー属性がマップされていない場合に警告を表示する
- 必須属性がマップされていない場合に警告を表示する
- 子ビジネス・オブジェクトがマップされていない場合に警告を表示する

Map Designer Express は、選択された検証を、当該マップのレベルで使用される他の変換規則に応じた深さまで実行します。

例: a.b.c というパスがマップされている場合、Map Designer Express はビジネス・オブジェクトのレベル a、a.b、および a.b.c についてこれらの検証を実行します。

詳細については、 90 ページの『マップの検証』を参照してください。

キー・マップの指定

「設定」ダイアログの「キー・マップ」タブには、「ダイアグラム」タブにあるいくつかの標準の変換のキー・マップが表示されます。

表 9. Map Designer Express のキー・マップの設定

キー・マップ	説明	詳細情報の参照先
移動/結合/サブマップ	移動、結合、またはサブマップの変換を作成するときに使用するキー・マップ。Map Designer Express は、変換と変換の区別をソース属性のタイプと数によって行います。	
	• 「移動」: 子ビジネス・オブジェクトではない 1 つのソース属性	43 ページの『ソース属性の宛先属性へのコピー』
	• 「結合」: 子ビジネス・オブジェクトではない 複数のソース属性	44 ページの『属性の結合』
	• 「サブマップ」: 子ビジネス・オブジェクトである 1 つ以上のソース属性	48 ページの『サブマップを使用した変換』
分割/相互参照	分割変換を作成するときに、あるいは一致関係の保守のために使用するキー・マップ。	46 ページの『属性の分割』, 53 ページの『一致関係の相互参照』
カスタム	カスタム変換を作成するときに使用するキー・マップ。	54 ページの『カスタム変換の作成』

「キー・マップ」タブには、以下の機能があります。

- キー・マップを変更するには、該当する変換フィールド内をクリックして、コンボ・ボックスからこの変換に必要なキー・マップを選択します。「OK」をクリックします。
- キー・マップをデフォルト値に戻す場合は、「デフォルトを使用」をクリックしてから「OK」をクリックします。

自動マッピングの指定

「設定」ダイアログの「自動マッピング」タブには、Map Designer Express がマップ自動化のためにビジネス・オブジェクト内で一致する属性名を検索するときに使用するように選択できるオプションがあります。オプションは以下のとおりです。

- 大/小文字を区別しない: 検索ストリングに関して大/小文字を区別しない名前の突き合わせを実行する。
- 非互換のデータ・タイプを無視する: 検索ストリングに関して非互換のデータ型との名前の突き合わせを実行する。

  注: このオプションを選択すると、データが失われる可能性があります。

詳細については、 72 ページの『マップの自動化の使用』を参照してください。

メインウィンドウのカスタマイズ

Map Designer Express では、次の方法でメイン・ウィンドウをカスタマイズできます。

- 26 ページの『ウィンドウの外観の選択』
- 27 ページの『連結可能なウィンドウの浮動化』

ウィンドウの外観の選択

最初にマップを指定せずに Map Designer Express を開いたときは、メインウィンドウは空で、ツールバーとステータス・バーが表示されています。マップを開くと、「ダイアグラム」タブが Map Designer Express のタブ・ウィンドウに表示され、さらに出力ウィンドウが開きます。デフォルトでは、以下のマップ・タブが表示されます。

- 「テーブル」タブ: 属性変換テーブルの下にビジネス・オブジェクト・ペインが表示されます。
- 「ダイアグラム」タブ: マップ・ワークスペース域が表示されますが、空です。
- 「メッセージ」タブと「テスト」タブ: それぞれ、21 ページの『「メッセージ」タブ』および 21 ページの『「テスト」タブ』を参照してください。

メインウィンドウの外観とマップ・タブは「表示」メニューのオプションでカスタマイズできます。表 10 に、「表示」プルダウン・メニューの各オプションと各オプションの Map Designer Express ウィンドウの外観に与える効果を示します。

表 10. Map Designer Express ウィンドウのカスタマイズ用の「表示」メニューの各オプション

「表示」メニュー・オプション	**表示される要素**
ツールバー	Map Designer Express の以下のツールバー用のオプションがあるサブメニューです。 • 標準ツールバー • Designer ツールバー • プログラム・ツールバー
ステータス・バー	1 行のペイン。Map Designer Express の状況情報が表示されます。
ビジネス・オブジェクト・ペイン	Map Designer Express の「テーブル」タブ内で、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトを表示するペイン。
プロジェクト・ペイン	Map Designer Express の「ダイアグラム」タブ内で、ビジネス・オブジェクト・ブラウザーを表示するペイン。
ダイアグラム	「ダイアグラム」タブのビジネス・オブジェクト・ウィンドウに、ソース・ビジネス・オブジェクトおよび宛先ビジネス・オブジェクトのどちらの属性を表示するかを選択する以下のオプションがあるサブメニューです。 • 「すべての属性」 • 「リンクされている属性のみ」 • 「リンクされていない属性のみ」 Designer ツールバーには、これらの属性を示すアイコンもあります。
出力ウィンドウ	Map Designer Express ウィンドウの下部の横に表示される小さなウィンドウ。「表示」メニューの「出力をクリア」オプションを指定すると、出力ウィンドウ内のテキストがすべて消去されます。

ヒント: 各メニュー・オプションの左横にチェック・マークが付いているとき、対応する要素が画面に表示されます。要素を非表示にするには、対応するメニュー・オプションを選択します。選択すると、チェック・マークが消え、その要素は表示されないことを示します。逆に、表示されていない要素を表示させるには、対応するメニュー・オプションを選択します。選択すると、表示させる要素の横にチェック・マークが現れます。

連結可能なウィンドウの浮動化

Map Designer Express は、連結可能なウィンドウとしての機能をサポートします。

- メインウィンドウのツールバー
  - 標準ツールバー
  - Designer ツールバー
  - プログラム・ツールバー

  これらのツールバーの機能の詳細については、31 ページの『Map Designer Express のツールバー』を参照してください。
- 「出力ウィンドウ」
- 「検索」コントロール・ペイン。詳細については、79 ページの『マップの情報の検索』を参照してください。

ヒント: 連結可能なウィンドウは、デフォルトでは通常メインウィンドウの端に沿って配置され、メインウィンドウの一部として移動します。連結可能なウィンドウを浮動させるときは、メインウィンドウから切り離します。これにより、独立したウィンドウとして機能させることができます。連結可能なウィンドウを浮動させるには、マウスの左ボタンを押したままウィンドウの境界をつかみ、メインウィンドウまたはデスクトップ上にドラッグします。

Map Designer Express の機能の使用法

Map Designer Express の機能には、以下のいずれの方法を使用してもアクセスできます。

- プルダウン・メニュー
- コンテキスト・メニュー
- ツールバーのボタン
- キーボードのショートカット

Map Designer Express のプルダウン・メニュー

Map Designer Express には、以下のプルダウン・メニューがあります。

- 「ファイル」メニュー
- 「編集」メニュー
- 「表示」メニュー
- 「デバッグ」メニュー
- 「ツール」メニュー
- 「ヘルプ」メニュー

以下のセクションでは、これらの各メニューのオプションについて説明します。

「ファイル」メニューの機能: Map Designer Express の「ファイル」プルダウン・メニューには、表 11 に示すオプションがあります。

表 11. Map Designer Express の「ファイル」メニュー・オプション

「ファイル」メニュー・オプション	説明	詳細情報の参照先
「新規」	マップ・ファイルを新規に作成し、既存のマップをマップ・ワークスペースから消去します。	33 ページの『マップの作成: 基本手順』
「開く」	既存のマップを「プロジェクトから」または「ファイルから」開きます。	61 ページの『マップのオープン』
「閉じる」	現行マップを閉じます。	64 ページの『マップを閉じる』
「保管」	現行マップを同じ名前で「プロジェクトに」または「ファイルに」保管します。	55 ページの『マップの保管』
「別名保管」	現行マップを、そのマップとは異なる名前で「プロジェクトに」または「ファイルに」保管します。	55 ページの『マップの保管』
「削除」	指定されたマップを削除します。	82 ページの『オブジェクトの削除』
「マップを検証」	現行マップを検証します。	90 ページの『マップの検証』
「コンパイル」	現行マップをコンパイルします。	91 ページの『マップのコンパイル』
「サブマップでコンパイル」	現行マップとそのサブマップをコンパイルします。	91 ページの『マップのコンパイル』
「すべてコンパイル」	定義されているマップのすべて、またはサブセットをコンパイルします。	93 ページの『マップ・セットのコンパイル』
「マップ文書を作成」	ビジネス・オブジェクト間のマップを記述した HTML ファイルを作成します。	70 ページの『マップ文書の作成手順』
「マップ文書を表示」	HTML のマップ文書ファイルを HTML ブラウザーで表示します。	71 ページの『マップ文書の表示』
「印刷設定」、「印刷プレビュー」、「印刷」	プレビュー、印刷、および印刷ジョブの構成に関するオプションがあります。	82 ページの『マップの印刷』
「終了」	Map Designer Express を終了します。	該当なし

「編集」メニューの機能: Map Designer Express の「編集」プルダウン・メニューには、以下のオプションがあります。

- Windows の標準編集オプション: 「切り取り」、「コピー」、および「貼り付け」
- 「現在の選択範囲を削除」: 現在選択されているオブジェクトを削除します。
- 「全選択」: 「ダイアグラム」タブで、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクト間の変換すべてを選択します。
- 「行を挿入」: 「テーブル」タブの属性変換テーブル内で現在行の前に 1 行を挿入します。
- 「ビジネス・オブジェクトを追加」: マップにビジネス・オブジェクト (ソース、宛先、および一時) を追加する場合に、「ビジネス・オブジェクトを追加」ダイアログを表示します。
- 「ビジネス・オブジェクトを削除」: ビジネス・オブジェクトを削除する場合に、「ビジネス・オブジェクトを削除」ダイアログを表示します。

- 「検索」: 属性名または変換コードを検索して、マップされていない属性のテキストまたは変換コードを探します。
- 「置換」: カスタム Java コードまたはコメント内で検索および置換を行います。
- 「マップ・プロパティー」:「マップ・プロパティー」ウィンドウを表示します。

「表示」メニューの機能: Map Designer Express の「表示」プルダウン・メニューには、以下の表示オプションがあります。

- 「ビジネス・オブジェクト・ペイン」: このオプションを使用可能にした場合、Map Designer Express ウィンドウの「テーブル」タブの下のペインにソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが表示されます。
- 「ダイアグラム」: 「ダイアグラム」タブのビジネス・オブジェクト・ウィンドウに属性を表示するオプションがあります。
- 「プロジェクト・ペイン」: 常に使用可能になっており、ビジネス・オブジェクト・ブラウザーが Map Designer Express ウィンドウの「ダイアグラム」タブの左のペインとして表示されます。
- 「出力をクリア」: 出力ウィンドウの内容を消去します。
- 「出力ウィンドウ」: このオプションを使用可能にした場合、マップのオープン、マップの検証、マップの保管、マップのコンパイル、マップのテスト実行に関するメッセージなどの状況メッセージが表示されます。
- 「ツールバー」: Map Designer Express ツールバーの表示オプションとして「標準」、「Designer」、および「プログラム」があります。
- 「ステータス・バー」: このオプションを使用可能にした場合、メインウィンドウの最下部に 1 行の状況メッセージが表示されます。
- 「設定」: Map Designer Express の設定を行う「設定」ダイアログを表示します。

表示を制御する「表示」メニュー・オプションの詳細については、26 ページの『ウィンドウの外観の選択』を参照してください。

「デバッグ」メニューの機能: 「デバッグ」プルダウン・メニューからは、Map Designer Express のデバッグ機能にアクセスすることができます。オプションは以下のとおりです。

- 「テスト実行」: サーバーに接続し、プロジェクトから開かれたマップのテスト実行を開始します。
- 「続行」: ブレークポイントで停止した後、実行を続行します。
- 「ステップオーバー」: ブレークポイントで停止した後実行を続行しますが、次の属性を実行する前に実行を停止します。
- 「テスト実行を停止」: マップのテスト実行を停止します。
- 「拡張」: サーバー内にあるマップのテストのためにサーバーに接続するとき (「接続」) のオプションと、サーバーとの接続を切断し、マップを閉じるとき (「終了」) のオプションを提供します。
- 「ブレークポイントの切り替え」: マップ内にブレークポイントを設定します。これにより、選択された属性の変換直前に実行を一時停止させます。
- 「ブレークポイント」: マップのすべてのブレークポイントを表示します。

- 「すべてのブレークポイントをクリア」: マップ内のすべてのブレークポイントを消去します。

Map Designer Express のテスト機能とデバッグ機能の使用法の詳細については、 94 ページの『マップのテスト』を参照してください。

「ツール」メニューの機能: Map Designer Express の「ツール」プルダウン・メニューには、各ツール (マップの自動化ツールを含む) を始動するオプションがあります。

- 自動マッピング
- マップの反転
- Process Designer Express

  制約事項: このツールは、WebSphere Business Integration Express Plus でのみ使用可能です。
- Map Designer Express
- Business Object Designer Express
- Relationship Designer Express

「ヘルプ」メニューの機能: 「ヘルプ」メニューには、標準 Windows ヘルプ・オプションがあります。

- 「ヘルプ・トピック」
- 「ドキュメンテーション」
- Map Designer Express に関する

コンテキスト・メニュー

コンテキスト・メニューは、変換規則列、「テーブル」の表示内の行ヘッダー、ソースのテスト・ペイン内の子ビジネス・オブジェクト、ダイアログ内の編集ボックスなど、多様な場所から右マウス・ボタンをクリックすることによって使用可能になるショートカット・メニューです。役立つコマンドが入ったメニューが開きます。このメニューはクリックする場所によって異なります。

例: 変換規則列の内部をマウス・ボタンでクリックすると、以下のオプションを示すコンテキスト・メニューが表示されます。

- 「開く」: 「結合」、「分割」、「サブマップ」などの変換規則の対応するダイアログ・ボックスを開きます。カスタム変換の場合は、Activity Editor を開きます。
- 「新規ウィンドウで開く」: カスタム変換で、Activity Editor の新しいインスタンスを開き、変換規則の詳細を表示します。
- 「ソースを表示」: Activity Editor で変換の対応する Java コードを表示します。変換の種類によっては、コードが読み取り専用になることがあります。

注: 変換のセルをダブルクリックしたときのデフォルトのアクションは「開く」です。「開く」が使用不可になっている変換では、そのアクションが有効でないことを示すメッセージがステータス・バーに表示されます。

Map Designer Express のツールバー

Map Designer Express には、ユーザーが実行する必要のある一般的なタスク用のツールバーが 3 つあります。

- 標準ツールバー
- Designer ツールバー
- プログラム・ツールバー

これらのツールバーは連結可能で、メインウィンドウのパレットから引き離してメインウィンドウやデスクトップの上に浮動させることができます。

ヒント: それぞれのツールバー・ボタンの目的を確認するには、マウス・カーソルで各ボタンをロールオーバーしてください。

標準ツールバー: 図 7 に標準ツールバーを示します。

図 7. 標準ツールバー

以下のリストに、標準ツールバーの各ボタンの機能を (左から右の順に) 示します。

1. 「新規マップ」
2. 「開く」
3. 「プロジェクトに保管」
4. 「ファイルから開く」
5. 「ファイルに保管」
6. 「マップで検索」
7. 「マップを印刷」
8. 「切り取り」
9. 「コピー」
10. 「貼り付け」
11. 「削除」
12. 「ヘルプ」

Designer ツールバー: 図 8 に Designer ツールバーを示します。

図 8. Designer ツールバー

以下のリストに、Designer ツールバーの各ボタンの機能を (左から右の順に) 示します。

1. 「ビジネス・オブジェクトを追加」
2. 「検証」
3. 「コンパイル」
4. 「テスト実行」

5. 「続行」
6. 「ステップオーバー」
7. 「ブレークポイントの切り替え」
8. 「すべてのブレークポイントをクリア」
9. 「すべての属性」
10. 「リンクされている属性のみ」
11. 「リンクされていない属性のみ」

プログラム: 図 9 にプログラム・ツールバーを示します。

図 9. プログラム・ツールバー

以下のリストに、プログラム・ツールバーの各ボタンの機能を (左から右の順に) 示します。

1. Process Designer Express

   制約事項: このツールバー・ボタンは、WebSphere Business Integration Express Plus でのみ使用可能です。
2. Map Designer Express
3. Business Object Designer Express
4. Relationship Designer Express

キーボードのショートカット

Map Designer Express には多数のメニュー・オプション用のキーボード・ショートカットがあります。これを表 12 に示します。

表 12. Map Designer Express のキーボード・ショートカット

キーボード・ショートカット	説明	詳細情報の参照先
Ctrl+E	現行マップ定義をマップ定義ファイルに保管します。	57 ページの『マップのファイルへの保管』
Ctrl+F	マップ内のテキストまたはリンクされていない属性を見つけるための「検索」コントロール・パネルを表示します (置換の場合は Ctrl+H を使用します)。	79 ページの『マップの情報の検索』
Ctrl+H	「置換」ダイアログを表示して、変換規則のカスタマイズ済み Java コードおよびコメント内のテキストを検索および置換します。	81 ページの『テキストの検索および置換』
Ctrl+I	現行マップ定義ファイルを開きます。	63 ページの『ファイルからのマップのオープン手順』
Ctrl+M	マップ文書を表示します。	71 ページの『マップ文書の表示』
Ctrl+N	「新規マップ」ウィザードを表示して、新しいマップを作成します。	33 ページの『マップの作成: 基本手順』
Ctrl+O	System Manager のプロジェクトからマップ定義を開きます。	62 ページの『System Manager のプロジェクトからのマップのオープン手順』
Ctrl+P	マップ定義を印刷します。	82 ページの『マップの印刷』

表 12. Map Designer Express のキーボード・ショートカット (続き)

キーボード・ショートカット	説明	詳細情報の参照先
Ctrl+S	Map Designer Express のメインウィンドウ内: 現行マップ定義を System Manager のプロジェクトに保管します。	55 ページの『マップのプロジェクトへの保管』
Ctrl+U	Map Designer Express の各種設定のための「設定」ダイアログを表示します。	22 ページの『設定の割り当て』
Ctrl+Alt+F	現行マップ定義を別の名前のマップ定義ファイルに保管します (別名保管)。	57 ページの『マップのファイルへの保管』
Ctrl+Alt+S	現行マップ定義を別の名前で System Manager のプロジェクトに保管します (別名保管)。	55 ページの『マップのプロジェクトへの保管』
Ctrl+Shift+P	マップ定義の印刷についての情報を指定する「印刷設定」ダイアログを表示します。	82 ページの『マップの印刷』
Ctrl+Enter	「マップ・プロパティー」ダイアログを表示します。このダイアログから、マップの一般的なプロパティーおよびビジネス・オブジェクトのプロパティーを設定できます。	64 ページの『マップ・プロパティー情報の指定』
F7	現行マップをコンパイルします。	91 ページの『マップのコンパイル』
Alt+F4	現行マップを閉じます。	64 ページの『マップを閉じる』
Del	現在選択されているエンティティーを削除します。	該当なし
F1	現在のダイアログまたはウィンドウについての状況に応じたヘルプを表示します。	該当なし
Ctrl+F7	System Manager に定義されているすべてのマップまたはマップのサブセットをコンパイルします。	93 ページの『マップ・セットのコンパイル』
F8	テスト実行中に、マップの終わりまたは別のアクティブなブレークポイントまで実行することによって、一時停止されているマップの実行を続行します。	101 ページの『ブレークポイント処理の手順』
F9	変換規則のブレークポイントの状態を切り替えます。	98 ページの『ブレークポイントの設定』
F10	テスト実行中に、次の単一ステップを実行することによって、一時停止されているマップの実行を続行します。	101 ページの『ブレークポイント処理の手順』

マップの作成: 基本手順

表 13 に、新規マップを作成するための基本手順の概要を示します。

表 13. 新規マップの作成手順

作成手順	詳細情報の参照先
1. 「新規マップ」ウィザードを使用して、新しいマップ・ファイルを作成します。新規マップ用のプロジェクト、ソース・ビジネス・オブジェクト、宛先ビジネス・オブジェクト、および新規マップの名前を指定します。	34 ページの『マップ定義の作成手順』

表 13. 新規マップの作成手順 (続き)

作成手順		詳細情報の参照先
2.	各宛先ビジネス・オブジェクトに動詞を設定します。ほとんどの場合、宛先ビジネス・オブジェクトの動詞は、ソース・ビジネス・オブジェクトの動詞と同じになります。動詞の値が常に特定の値になるように設定することもできます。	40 ページの『宛先ビジネス・オブジェクトの動詞の設定』
3.	マップの対象となる宛先属性ごとに変換ステップを指定します。変換ステップの指定方法は、必要とする変換の種類によって異なります。	41 ページの『標準の属性変換の指定』
4.	宛先属性についてのコメントを指定します。この情報はオプションですが、これを指定しておくと、Map Designer Express でのマップ情報が大幅に読みやすくなります。	59 ページの『属性のコメント・フィールドでのコメントの設定』
5.	マップを保管します。	55 ページの『マップの保管』
6.	完了の確認、マップの検証、およびマップのコンパイルを行います。	58 ページの『完全性のチェック』、 90 ページの『マップの検証』、および 91 ページの『マップのコンパイル』
7.	マップをテストし、デバッグします。	94 ページの『マップのテスト』

マップ定義の作成手順

Map Designer Express には、マップ定義の作成を支援する「新規マップ」ウィザードがあります。「新規マップ」ウィザードを使用してマップ定義を作成するには、次のステップを実行します。

1. 以下のいずれかの方法で、「新規マップ」ウィザードを始動します。
   - 「ファイル」メニューから「新規」を選択して、マップを新規作成します。
   - キーボード・ショートカット `Ctrl+N` を使用します。
   - 標準ツールバーにある「新規マップ」ボタンをクリックします。

   結果: Map Designer Express が「新規マップ」ウィザードの最初のウィンドウを表示します。

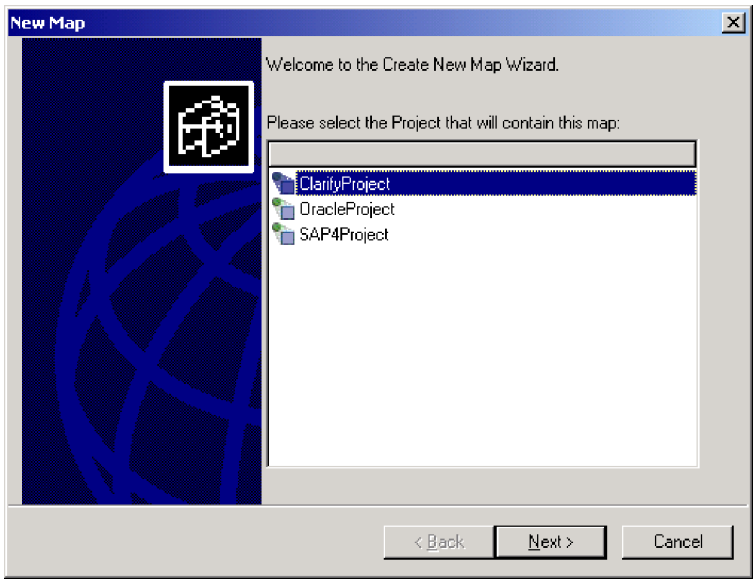

図 10. 「新規マップ」ウィザードの最初のウィンドウ

2. リスト・ボックスから、作成するマップ用のプロジェクトの名前を選択します。
3. そのマップのソース・ビジネス・オブジェクトとして使用するビジネス・オブジェクトを選択します。必要なビジネス・オブジェクトの「使用」列内をクリックすることによって、1 つ以上のソース・ビジネス・オブジェクトを選択することができます。さらに「次へ」をクリックして先へ進みます。

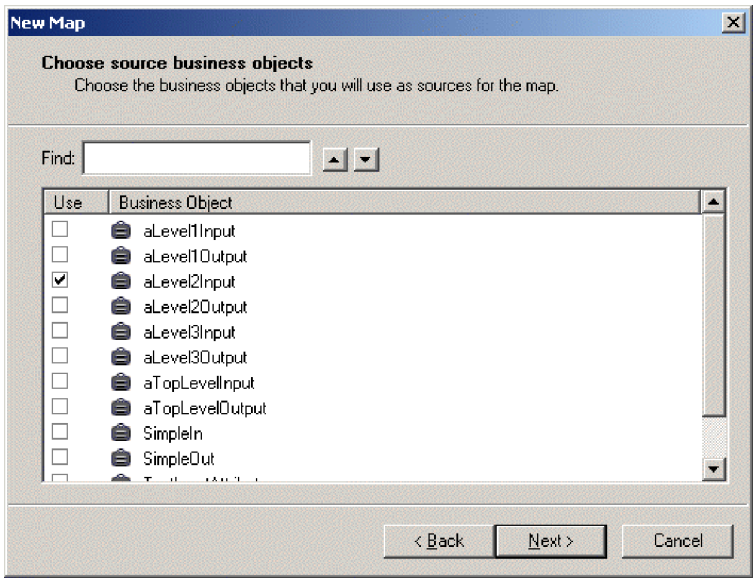

図 11. ソース・ビジネス・オブジェクトの選択

ヒント: 特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力してください。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。「次へ」をクリックして先へ進みます。

「新規マップ」ウィザードでは、ソース・ビジネス・オブジェクトの指定は必須ではありません。ソース・ビジネス・オブジェクトを選択せずに「次へ」をクリックし、このビジネス・オブジェクト定義の指定を後にすることができます。後で、「ダイアグラム」タブのマップ・ワークスペースで指定することができます。詳細については、37 ページの『ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの作成』を参照してください。

注: System Manager 内でビジネス・オブジェクトの追加または削除を行うと、ビジネス・オブジェクト定義リストが動的に更新されます。

4. そのマップの宛先ビジネス・オブジェクトとして使用するビジネス・オブジェクトのタイプを選択します。必要なビジネス・オブジェクトの「使用」列内をクリックすることによって、1 つ以上の宛先ビジネス・オブジェクトを選択することができます。さらに「次へ」をクリックして先へ進みます。

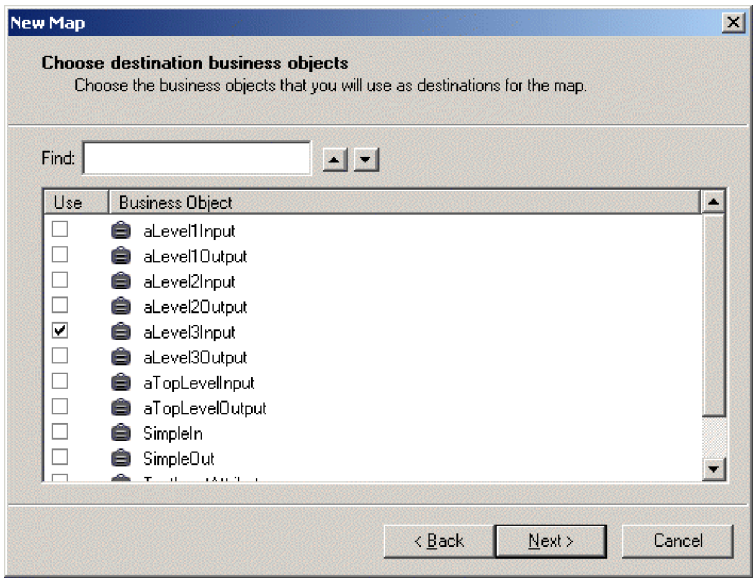

図 12. 宛先ビジネス・オブジェクトの選択

ヒント: 特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力してください。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。「次へ」をクリックして先へ進みます。

「新規マップ」ウィザードでは、宛先ビジネス・オブジェクトの指定は必須ではありません。宛先ビジネス・オブジェクトを選択せずに「次へ」をクリックし、このビジネス・オブジェクト定義の指定を後にすることができます。後で、「ダイアグラム」タブのマップ・ワークスペースで指定することができます。詳細については、37 ページの『ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの作成』を参照してください。

注: System Manager 内でビジネス・オブジェクトの追加または削除を行うと、ビジネス・オブジェクト定義リストが動的に更新されます。

5. マップに関連付ける名前を指定します。

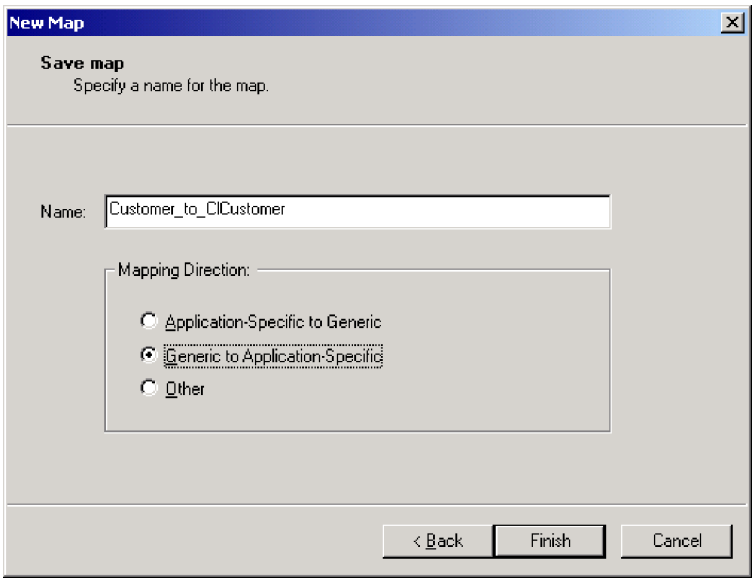

図 13. 新規マップの保管

規則: マップ名には、最大 80 文字までの英数字と下線 (_) を指定できます。Map Designer Express では、いくつかの命名上の制約が課されます。例えば、マップ名にはピリオド、左中括弧 ([)、右中括弧 (])、一重引用符、二重引用符、スペースなどのいくつかの句読点記号を指定できません。

「新規マップ」ウィザードでは、マップ名の指定は必須ではありません。マップ名を入力せずに「Finish」をクリックし、このマップ定義の命名を後で行うことが可能です。マップを保管するときに、Map Designer Express により「マップを別名保管」ダイアログが表示され、必要なマップ名を指定するように要求されます。詳細については、55 ページの『マップのプロジェクトへの保管』を参照してください。

マップがインバウンドであるかアウトバウンドであるかを指定します。このマップの役割は、自動的に生成される関係コードのために必要です。

6. 「Finish」をクリックして、指定したソース・ビジネス・オブジェクトおよび宛先ビジネス・オブジェクトが入った新しいマップ定義を保管します。

結果: Map Designer Express が「ダイアグラム」タブに新しいマップの情報を表示します。

ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの作成

マップのソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトを「新規マップ」ウィザードで指定しなかった場合は、「ビジネス・オブジェクトを追加」ダイアログか、ビジネス・オブジェクト・ブラウザーの「ダイアグラム」タブから指定できます。

「ビジネス・オブジェクトを追加」ダイアログからビジネス・オブジェクトを指定する手順

「ビジネス・オブジェクトを追加」ダイアログの「一般」タブから、マップにソース・ビジネス・オブジェクトまたは宛先ビジネス・オブジェクトを追加するには、以下の手順を行います。

1. 「ビジネス・オブジェクトを追加」ダイアログは、以下のいずれかの方法で表示します。
   - Map Designer Express の「編集」メニューから「ビジネス・オブジェクトを追加」を選択します。
   - Designer ツールバー内にある、「ビジネス・オブジェクトを追加」ボタンをクリックします。
   - 「テーブル」タブから、ビジネス・オブジェクト・ペインの空の領域の中で右マウス・ボタンをクリックし、コンテキスト・メニューから「ビジネス・オブジェクトを追加」を選択します。
   - 「ダイアグラム」タブから、マップ・ワークスペースの中で右マウス・ボタンをクリックし、コンテキスト・メニューから「ビジネス・オブジェクトを追加」を選択します。
2. ソース・ビジネス・オブジェクトを指定するには、以下のようにします。
   - ビジネス・オブジェクト・リストにあるビジネス・オブジェクトをクリックします。
   - 「ソースに追加」ボタンをクリックします。

   ヒント: 特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力してください。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。
3. 宛先ビジネス・オブジェクトを指定するには、以下のようにします。
   - ビジネス・オブジェクト・リストにあるビジネス・オブジェクトをクリックします。
   - 「宛先に追加」ボタンをクリックします。

   ヒント: 特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力してください。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。
4. ダイアログを閉じるには、「完了」をクリックします。

ビジネス・オブジェクト・ブラウザーの「ダイアグラム」タブからビジネス・オブジェクトを指定する手順

「ダイアグラム」タブから、ソース・ビジネス・オブジェクトまたは宛先ビジネス・オブジェクトをマップに追加することができます。これを行うには、以下の手順を実行します。

1. ソース・ビジネス・オブジェクトをビジネス・オブジェクト・ブラウザーからマップ・ワークスペースの左側にドラッグします。そのビジネス・オブジェクトが表示され、タイトルの先頭が Src となっています。

2. 宛先ビジネス・オブジェクトをビジネス・オブジェクト・ブラウザーからマップ・ワークスペースの右側にドラッグします。そのビジネス・オブジェクトが表示され、タイトルの先頭が Dest となっています。

注: 点線の境界線がワークスペースの左半分と右半分を分割し、マップ・ワークスペースのソース部分と宛先部分を区別しています。各オブジェクトを適切な場所にドロップするように注意してください。

図 14 に、マップのワークスペースに表示されたソース・ビジネス・オブジェクトと宛先・ビジネス・オブジェクトを示します。

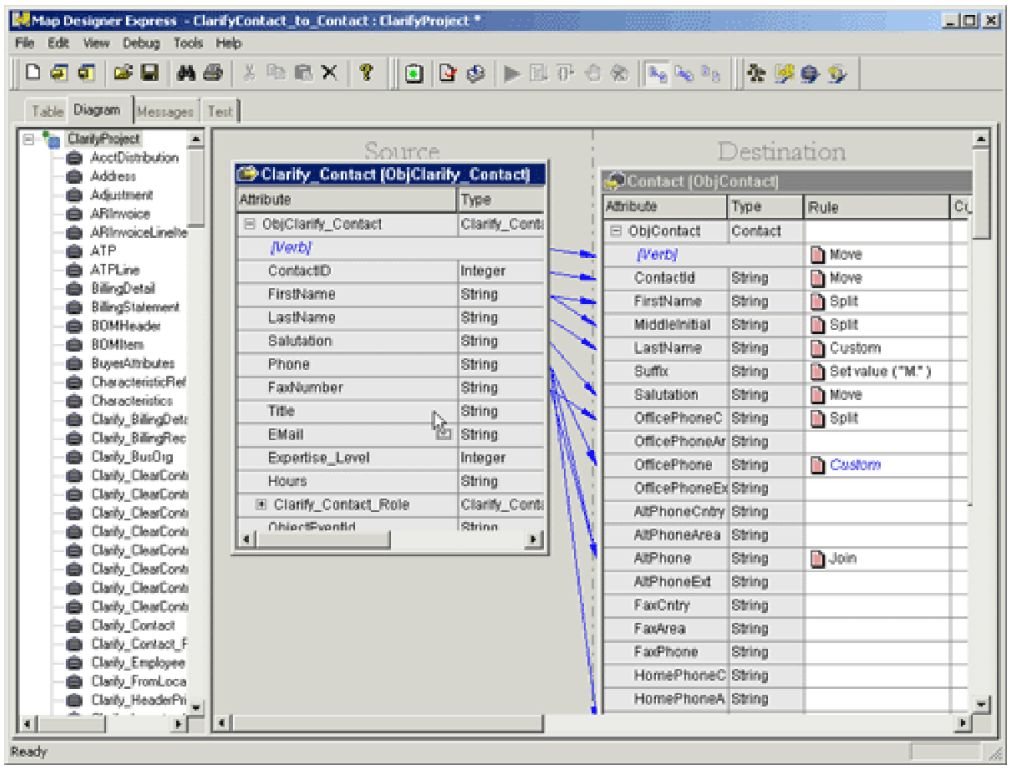

図 14. ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの定義

ヒント: 別の方法で、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトを作成することもできます。ビジネス・オブジェクト・ブラウザー内で、ビジネス・オブジェクトを右マウス・ボタンでクリックし、コンテキスト・メニューから「コピー」を選択します。次に、マップ・ワークスペースで右マウス・ボタンをクリックして、「入力オブジェクトとして貼り付け」または「出力オブジェクトとして貼り付け」を選択します。

Map Designer Express は、ソース・オブジェクトと宛先オブジェクト用にビジネス・オブジェクト・ウィンドウ と呼ばれるウィンドウを作成します。このウィンドウのタイトル・バーには、ビジネス・オブジェクトのインスタンス名が表示されます。ビジネス・オブジェクト・ウィンドウのタイトル・バーの説明については、184 ページの『生成されたビジネス・オブジェクト変数および属性の使用』を参照してください。ソース・ビジネス・オブジェクト用のビジネス・オブジェクト・ウィンドウには、各ソース属性の名前用の列とデータ型用の列があります。宛先ビジネ

ス・オブジェクト用のビジネス・オブジェクト・ウィンドウには、名前、データ型、変換規則 (変換ステップを示す)、およびオプションのコメント用の各列があります。

ガイドライン: 間違ったビジネス・オブジェクトをドラッグしたり、ビジネス・オブジェクトを入力オブジェクトではなく、出力オブジェクトにしてしまったなど、操作を間違えた場合は、そのオブジェクトをマップ・ワークスペースから削除してやり直すことが可能です。マップ・ワークスペースからビジネス・オブジェクトを削除するには、以下のいずれかの方法で行います。

- 削除するビジネス・オブジェクトを選択し、「編集」メニューの「現在の選択範囲を削除」オプションを使用します (あるいは、Del キーを押します)。
- ビジネス・オブジェクトのウィンドウのタイトル・バーを右マウス・ボタンでクリックして、コンテキスト・メニューから「削除」を選択します。

宛先ビジネス・オブジェクトの動詞の設定

動詞とは、システムがビジネス・オブジェクトのデータをどう処理すべきかを示します。マップの実行時に、システムが作成する各宛先ビジネス・オブジェクトにどの動詞を割り当てるかを認識している必要があります。

マップにソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが 1 つずつのみ存在する場合は、宛先ビジネス・オブジェクトの動詞は、通常ソース・ビジネス・オブジェクトの動詞と同じになります。

この場合、ソース・ビジネス・オブジェクトの動詞を宛先ビジネス・オブジェクトへコピーする必要があります (39 ページの図 14 を参照)。このためには、ソース属性をソース・ビジネス・オブジェクトの動詞として、宛先属性を宛先ビジネス・オブジェクトの動詞として、「*移動*」変換規則を定義します。詳細については、 43 ページの『ソース属性の宛先属性へのコピー』を参照してください。

ヒント: ソース・ビジネス・オブジェクトの動詞を宛先ビジネス・オブジェクトへドラッグ・アンド・ドロップして、動詞の値を定義することも可能です。

ソース・ビジネス・オブジェクトにはない動詞を持つ宛先ビジネス・オブジェクトがマップにある場合は、宛先属性を宛先ビジネス・オブジェクトの動詞として、「*値を設定*」変換規則を定義することにより、この動詞を定数値に設定する必要があります。「値を設定」ダイアログ・ボックスで、動詞の定数値を入力してください。詳細については、 41 ページの『属性への値の指定』を参照してください。

マップに複数のソース・ビジネス・オブジェクトまたは複数の宛先ビジネス・オブジェクトが含まれている場合もあります。このようなオブジェクトは、いくつかの子ビジネス・オブジェクトを持つことがあります。この場合、それぞれの宛先ビジネス・オブジェクトにどの動詞を割り当てるかを慎重に考慮する必要があります。宛先ビジネス・オブジェクトによっては、1 つ以上のソース・ビジネス・オブジェクトの動詞を基にして動詞を設定するカスタム・ロジックが必要になることがあります。

標準の属性変換の指定

Map Designer Express では、いくつかの標準の属性変換を対話式に指定することができます。その際、Java コードをほとんど、あるいはまったく作成する必要がありません。表 14 に、Map Designer Express で指定できる標準の変換を示します。

表 14. 一般的な属性変換

名前	変換ステップ	目的
値を設定	『属性への値の指定』	ソース・ビジネス・オブジェクトにはないが、宛先アプリケーションに必要な宛先ビジネス・オブジェクトの属性に指定します。
移動	43 ページの『ソース属性の宛先属性へのコピー』	ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの両方で同じ属性に指定します。
結合	44 ページの『属性の結合』	ソース・ビジネス・オブジェクトのいくつかの属性の組み合わせから成る、宛先ビジネス・オブジェクトの属性に指定します。
分割	46 ページの『属性の分割』	以下のいずれかが当てはまる、宛先ビジネス・オブジェクトの属性に指定します。 • ソース・ビジネス・オブジェクトの 1 つの属性の一部分である。 • いくつかのフィールドで構成されるが、ソース・ビジネス・オブジェクトとは異なる区切り文字を持つ。
サブマップ	48 ページの『サブマップを使用した変換』	子ビジネス・オブジェクトを持つソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの属性に指定します。
相互参照	53 ページの『一致関係の相互参照』	ビジネス・オブジェクトの一致関係を管理する目的で指定します。
カスタム	54 ページの『カスタム変換の作成』	自動生成の変換では提供されない変換が必要な属性に指定します。

実行可能なこの他の変換については、190 ページの『その他の属性の変換方法』を参照してください。

「ダイアグラム」タブで、「表示」>「ダイアグラム」メニューのオプションを使用して、ビジネス・オブジェクト・ウィンドウに表示する属性を選択することができます。全属性を表示するか、リンクされている (マップされている) 属性のみを表示するか、あるいはリンクされていない (マップされていない) 属性のみを表示するかを選択できます。

ヒント: 属性は、ビジネス・オブジェクト定義内と同じ順序で表示されます。長い属性リストの中で特定の属性を見つけるには、「編集」メニューから「検索」を選択します (あるいは、キーボード・ショートカット `Ctrl+F` を使用します)。詳細については、79 ページの『マップの情報の検索』を参照してください。

属性への値の指定

一部の宛先属性の値はソース属性に依存しないため、定数値を入力することができます。これは、宛先ビジネス・オブジェクトに、ソース・ビジネス・オブジェクト

にはないが、宛先アプリケーションに必要な属性が多数含まれる場合に特に当てはまります。属性のデフォルト値の例として、`CustomerStatus = "active"` や `AddressType = "business"` があります。

このタイプの変換を、*値を設定* 変換と呼びます。 図 15 に示す「値を設定」ダイアログを使用して、宛先属性の値を設定します。

値を設定変換の指定手順: 値を設定変換を指定するには、以下の手順を実行します。

1. 「値を設定」ダイアログを、以下のいずれかの方法で表示します。
   - 「テーブル」タブから次の手順を実行する。
     a. 値を設定する宛先属性を選択します。
     b. 「変換規則」列内のリストから「値を設定」をクリックします。
   - 「ダイアグラム」タブから次の手順を実行する。
     a. 値を設定する宛先属性を選択します。
     b. 宛先ビジネス・オブジェクトの「規則」列内のリストから、「値を設定」をクリックします。
   - 値を設定変換がすでに定義済みである場合は、「値を設定」ダイアログを表示して、変換の確認を行うことができます (変換コードの変更も可能です)。以下のいずれかの方法で行います。
     – 変換規則列内の対応するセルをダブルクリックします。
     – 変換規則列にある「値を設定」のビットマップ・アイコンをクリックします。

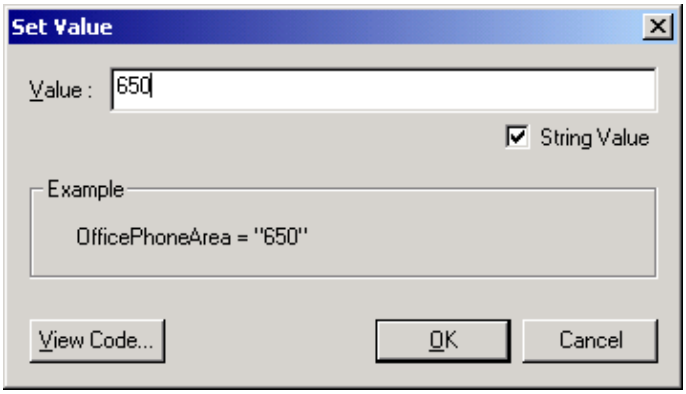

図 15. 「値を設定」ダイアログ

2. 「値を設定」ダイアログから、宛先属性に割り当てる定数値を設定します。「値を設定」ダイアログには、以下の機能があります。
   - 定数値を指定するには、「値」フィールドに値を入力します。数値の場合は、数を入力して、「ストリング値」チェック・ボックスが選択されていないことを確認するだけです。ストリング値の場合は、「値」フィールドにストリング値を入力し、「ストリング値」チェック・ボックスを選択します。

注: 「値を設定」ダイアログでは、「例」領域を使用して、結果として生成される宛先属性がどうなるかを示します。

- 入力済みの値を変更するには、「値」フィールド内をクリックして、必要に応じた編集を行います。
- 生成されたコードをカスタマイズするには、「コードを表示」プッシュボタンをクリックします。

  結果: Map Designer Express は Activity Editor を Java 表示で開き、宛先属性の変換コードのサンプルを読み取り専用モードで表示します。変換コードに変更を加えるには、Activity Editor 内の「コードを編集」をクリックします。詳細については、113 ページの『Activity Editor の概要』を参照してください。

  注: Activity Editor で変更を保管すると、変更内容が Map Designer Express に伝達されます。変更内容は、マップを保管するときにも保管されます。
- 変換の設定を確認するには、「OK」をクリックします。

ソース属性の宛先属性へのコピー

変換ステップのうち最も簡単なものは、1 つのソース属性を対応する宛先属性にコピーすることです。このタイプの変換を、*移動* 変換と呼びます。

移動変換の指定手順: 移動変換を指定するには、以下のいずれかのマップ・タブから手順を実行します。

- 「テーブル」タブから
  1. ソース属性を選択します。
  2. 宛先属性を選択します。
  3. 「変換規則」列内のリストから「移動」をクリックします。
- 「ダイアグラム」タブから
  1. ソース属性を選択します。
  2. `Ctrl+Drag` を使用して宛先属性に移動します。つまり、`Ctrl` キーを押しながら属性を宛先ビジネス・オブジェクト・ウィンドウ内の宛先属性へドラッグします。`Ctrl` キーは、マウス・ボタンを放すまで押したままにします。そのようにしないと、操作が正常に終了しません。

     結果: Map Designer Express により、ソース・オブジェクトから宛先オブジェクトまでの青の矢印が作成されます。変換に関係するソース属性が子ビジネス・オブジェクトではない 1 つのソース属性である場合、Map Designer Express はその変換が移動であると想定し、自動的に宛先属性の「規則」列に「移動」を割り当てます。

ヒント: 「設定」ダイアログの「キー・マップ」タブから、「ダイアグラム」タブの「移動」変換を開始するためのキー・シーケンスをカスタマイズすることができます。詳細については、25 ページの『キー・マップの指定』を参照してください。

結果: Map Designer Express がソース属性の値を宛先属性にコピーするコードを生成します。ソース属性と宛先属性のデータ型が異なる場合、Map Designer Express は型変換が可能かどうかを判断し、可能であれば、ソースの型を宛先の型に変換するコードを生成します。型変換が*可能でない* 場合、あるいは、結果としてデータが

失われる可能性がある場合、Map Designer Express は操作を確認またはキャンセルするためのダイアログ・ボックスを表示します。

移動変換で生成されるコードのサンプルを確認したい場合は、規則列のコンテキスト・メニューで、「ソースを表示」を選択してください。

属性の結合

複数のソース属性の値を連結、つまり結合して、1 つの宛先属性を作成することができます。このタイプの変換を、*結合* 変換と呼びます。例えば、ソース・ビジネス・オブジェクトが、市外局番、電話番号、内線を別々の属性に格納しているのに対して、宛先ビジネス・オブジェクトはこれらの値をまとめて 1 つの属性に格納することができます。

属性を結合できる他に、これらの属性の順序を変更したり、区切り文字、括弧、その他の文字を挿入することもできます。例えば、市外局番と電話番号の別々の属性を結合して 1 つの属性をつくる場合に、市外局番の前後に括弧を挿入することもできます。

ヒント: 結合する複数の属性が、親ビジネス・オブジェクト内とその子ビジネス・オブジェクトのうちの 1 つなど、複数のソース・ビジネス・オブジェクトに存在する場合もあります。属性を定義済みの変数と結合することも可能です。(変数の定義の詳細については、187 ページの『一時変数の作成』を参照してください。)

複数のソース属性を 1 つの宛先属性に結合するには、図 16 に示す「結合」ダイアログを使用します。

結合変換の指定手順: 結合変換を指定するには、以下の手順を実行します。

1. 「結合」ダイアログは、以下のいずれかの方法で表示します。
   - 「テーブル」タブから
     a. 結合するソース属性 (複数) を選択します。

        ヒント: コンボ・ボックスの中の「複数の属性」をクリックすると、「複数の属性」ダイアログを表示することができます。このダイアログで、複数のソース属性を選択することができます。特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力します。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。ソース属性を選択したら、「OK」をクリックしてダイアログを閉じます。

     b. 1 つの宛先属性を選択します。
     c. 「変換規則」列内のリストから「結合」をクリックします。
   - 「ダイアグラム」タブから
     a. 2 つ以上のソース属性を選択します。
     b. `Ctrl+Drag` を使用して宛先属性に移動します。つまり、`Ctrl` キーを押しながら、選択されたソース属性を宛先属性にドラッグします。`Ctrl` キーは、マウス・ボタンを放すまで押したままにします。そのようにしないと、操作が正常に終了しません。

結果: 変換に関係するソース属性が複数ある場合、Map Designer Express はその変換が「結合」であると想定します。宛先属性の「規則」列に自動的に「結合」が割り当てられ、「結合」ダイアログが表示されます。

ヒント: 「設定」ダイアログの「キー・マップ」タブから、「ダイアグラム」タブの「結合」変換を開始するためのキー・シーケンスをカスタマイズすることができます。詳細については、25 ページの『キー・マップの指定』を参照してください。

- 結合変換がすでに定義済みである場合は、「結合」ダイアログを使用して、変換の確認を行うことができます (変換コードの変更も可能です)。以下のいずれかの方法で行います。
  - 変換規則列内の対応するセルをダブルクリックします。
  - 変換規則列にある「結合」のビットマップ・アイコンをクリックします。

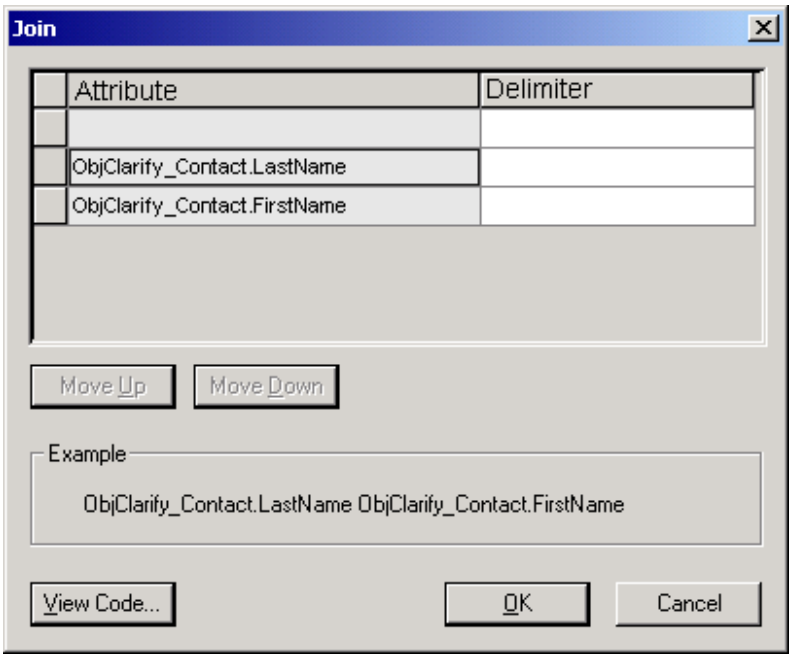

図 16. 「結合」ダイアログ

2. 「結合」ダイアログから、区切り文字の追加、括弧を使ったグループ化、属性の順序変更を必要に応じて行うことによって、ソース属性を連結する式を作成します。「結合」ダイアログには、以下の機能があります。
   - 区切り文字や括弧を挿入するには、その属性に関連付けられた「区切り文字」フィールドに区切り文字や括弧を入力します。区切り文字の前後に引用符を入れないで ください。入力した区切り文字は、関連付けられた属性に付加されます。先頭に区切り文字を入力する場合は、最初のブランク行の「区切り文字」フィールドに区切り文字を入力してください。

     注: 「結合」ダイアログでは、結合の後に「例」領域を使用して、結果として生成されるストリングがどうなるかを示します。
   - 入力した区切り文字や括弧を変更するには、「区切り文字」フィールドをクリックして、必要に応じた編集を行います。

- 区切り文字や属性の順序を変更するには、左端の列をクリックして行を選択した後、「上に移動」または「下に移動」をクリックして、行全体を上下に移動させます。
- 生成されたコードをカスタマイズするには、「コードを表示」プッシュボタンをクリックします。

  結果: Map Designer Express は Activity Editor を Java 表示で開き、宛先属性の変換コードのサンプルを読み取り専用モードで表示します。変換コードに変更を加えるには、Activity Editor 内の「コードを編集」をクリックします。詳細については、113 ページの『Activity Editor の概要』を参照してください。

  注: Activity Editor で変更を保管すると、変更内容が Map Designer Express に伝達されます。変更内容は、マップを保管するときにも保管されます。
- 変換の設定を確認するには、「OK」をクリックします。

結果: Map Designer Express がソース属性を結合するコードを生成します。データ型が宛先属性とは異なるソース属性があった場合、Map Designer Express は、型変換に必要な `DtpDataConversion` クラス内のメソッドを呼び出します。

属性の分割

ソース属性を 2 つ以上の宛先属性に分割する場合、各宛先属性それぞれに変換を指定します。このタイプの変換を、*分割* 変換と呼びます。例えば、`phone_number` などの 1 つのソース属性を `area_code`、`tel_number`、`extension` などの 3 つの宛先属性に分割する場合、`area_code`、`tel_number`、および `extension` のそれぞれに変換を指定します。

図 17 に示す「分割」ダイアログを使用して、1 つのソース属性を複数の宛先属性に分割します。

分割変換の指定手順: 分割変換を指定するには、以下の手順を実行します。

1. 「分割」ダイアログは、以下のいずれかの方法で表示します。
   - 「テーブル」タブから次の手順を実行する。
     a. 分割するソース属性を 1 つ選択します。
     b. 必要な宛先属性のうちの 1 つを選択します。
     c. 「変換規則」列内のリストから「分割」をクリックします。
     d. ソース属性のセグメントを受け取る各宛先属性ごとに、以上の手順を繰り返します。
   - 「ダイアグラム」タブから次の手順を実行する。
     a. 分割するソース属性を 1 つ選択します。
     b. `Alt+Drag` を使用して宛先属性のうちの 1 つに移動します。つまり、`Alt` キーを押しながら、ソース属性をいずれかの宛先属性にドラッグします。

        結果: 変換に関係する宛先属性が複数ある場合、Map Designer Express はその変換が「分割」であると想定します。宛先属性の「規則」列に自動的に「分割」が割り当てられ、「分割」ダイアログが表示されます。
     c. ソース属性のセグメントを受け取る各宛先属性ごとに、以上の手順を繰り返します。

ヒント: 「設定」ダイアログの「キー・マップ」タブから、「ダイアグラム」タブで分割変換を開始するためのキー・シーケンスをカスタマイズすることができます。詳細については、25 ページの『キー・マップの指定』を参照してください。

- 結合変換がすでに定義済みである場合は、「分割」ダイアログを使用して、変換の確認を行うことができます (変換コードの変更も可能です)。以下のいずれかの方法で行います。
  - 変換規則列内の対応するセルをダブルクリックします。
  - 変換規則列にある「分割」のビットマップ・アイコンをクリックします。

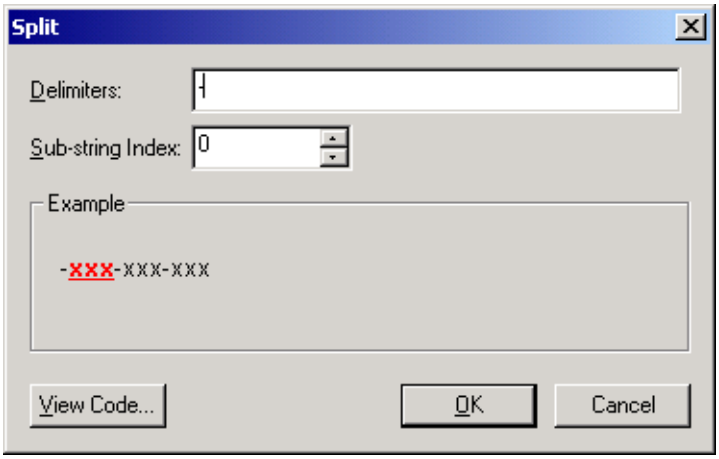

図 17. 「分割」ダイアログ

2. 「分割」ダイアログから、1 つの式を、区切り文字で分けられた複数のセグメントに分割します。各セグメントはインデックス番号で識別されます。最初のセグメントのインデックス番号はゼロです。「分割」ダイアログには、以下の機能があります。
   - ソース属性を解析する手段としての区切り文字を示すには、「区切り文字」フィールドにその区切り文字を入力します。区切り文字の前後に引用符を入れないで ください。このフィールドには、1 つ以上の区切り文字を指定できます。変換では、指定された各区切り文字を使用して、ストリングをセグメントに解析します。例えば、`LastName,FirstName` を分割する場合、区切り文字として「`,`」を指定し、セグメント 0 (最初のセグメント) として `LastName` を、セグメント 1 (2 番目のセグメント) として `FirstName` を指定します。

     注: 「分割」ダイアログでは、「例」領域を使用して、ソース属性のストリングがどうなるかを表示し、現在アクセス中のセグメントを示します。アクセスされたセグメントは、赤の太字で表示されます。
   - 入力した区切り文字や括弧を変更するには、「区切り文字」フィールドをクリックして、必要に応じた編集を行います。
   - 宛先属性にコピーされるソース属性のセグメントを識別するには、「サブストリングのインデックス」フィールドにインデックス番号を入力してください。
   - 生成されたコードをカスタマイズするには、「コードを表示」プッシュボタンをクリックします。

結果: Map Designer Express は Activity Editor を Java 表示で開き、宛先属性の変換コードのサンプルを読み取り専用モードで表示します。変換コードに変更を加えるには、Activity Editor 内の「コードを編集」をクリックします。詳細については、113 ページの『Activity Editor の概要』を参照してください。

注: Activity Editor で変更を保管すると、変更内容が Map Designer Express に伝達されます。変更内容は、マップを保管するときにも保管されます。

- 変換の設定を確認するには、「OK」をクリックします。

結果: Map Designer Express が宛先属性用の変換コードを生成します。生成されるコードは、`DtpSplitString()` クラスのメソッドを使用して、ソース属性を各セグメントに解析します。

サブマップを使用した変換

サブマップ とは、メイン・マップ と呼ばれる別のマップから呼び出されるマップです。このセクションでは、サブマップの以下の内容について説明します。

- 『サブマップの用途』
- 50 ページの『サブマップ変換の指定手順』

サブマップの用途: サブマップは、宛先属性の値を取得する目的で呼び出すことができますが、サブマップの最も一般的な用途は以下のとおりです。

- マップをモジュール化する
- 子ビジネス・オブジェクト間の変換を指定する

マップのモジュール性の向上: サブマップを利用すると、複数のマップで再利用可能な共通の変換を分離させることによって、マップのモジュール性を向上させることができます。例えば、`Customer` ビジネス・オブジェクトが `Address` という子ビジネス・オブジェクトを持ち、この子ビジネス・オブジェクトが `Order` ビジネス・オブジェクトの子でもあるとします。`Address` ビジネス・オブジェクト用にサブマップを作成しておくと、`Customer` と `Order` 両方のビジネス・オブジェクトのマップ内でこのサブマップを再利用できます。

図 18 に、サブマップ `MyAddrToGenAddr` を 2 つの異なるマップで再利用する仕組みを示します。

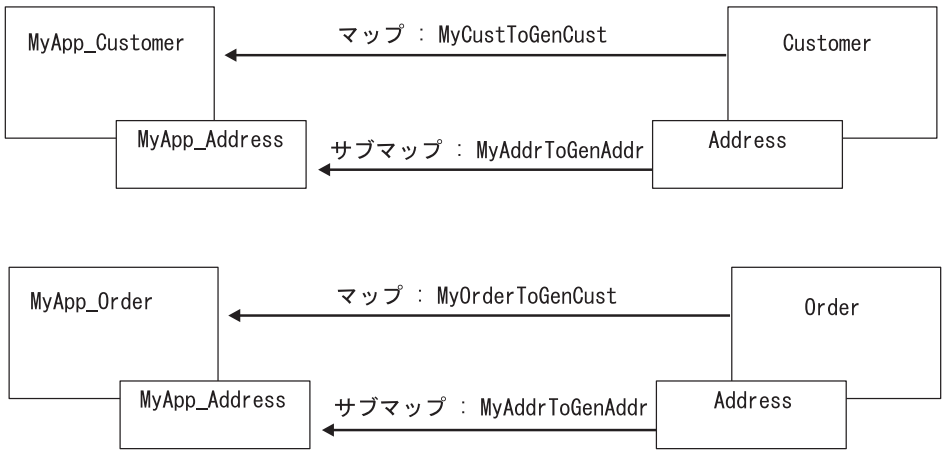

図 18. モジュール性を求めたサブマップの使用

子ビジネス・オブジェクトの変換: ソース属性および宛先属性に、複数カーディナリティー子ビジネス・オブジェクトが含まれている場合、サブマップを使用してこれらの属性の変換を指定すると便利です。複数カーディナリティー子ビジネス・オブジェクトの一般的な例として、1 人の顧客の複数の住所や 1 つの注文の中の複数の品目名があります。

最も単純なケースでは、各ソース子ビジネス・オブジェクトを 1 つの宛先子ビジネス・オブジェクトに、1 対 1 の関係で変換します。図 19 に、Employee ビジネス・オブジェクトと、EmployeeAddress の複数のインスタンスを含む、子ビジネス配列にサブマップを使用する例を示します。

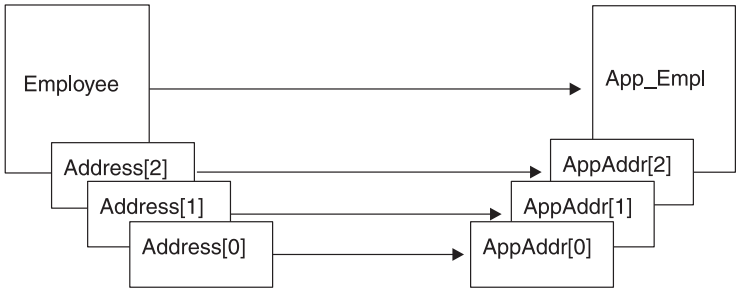

図 19. 子ビジネス・オブジェクト配列の 1 対 1 の変換

サブマップは、サブマップを実行するかどうかを制御する条件ステートメントと関連付けることができます。例えば、 50 ページの図 20 の場合を考えてみます。Order ビジネス・オブジェクトは、OrderLine という属性を持ち、この属性に複数カーディナリティー子ビジネス・オブジェクト OrderLine が含まれているとします。OrderLine ビジネス・オブジェクトは、DeliverySchedule という属性を持ち、この属性に複数カーディナリティー子ビジネス・オブジェクト DelSched が含まれています。

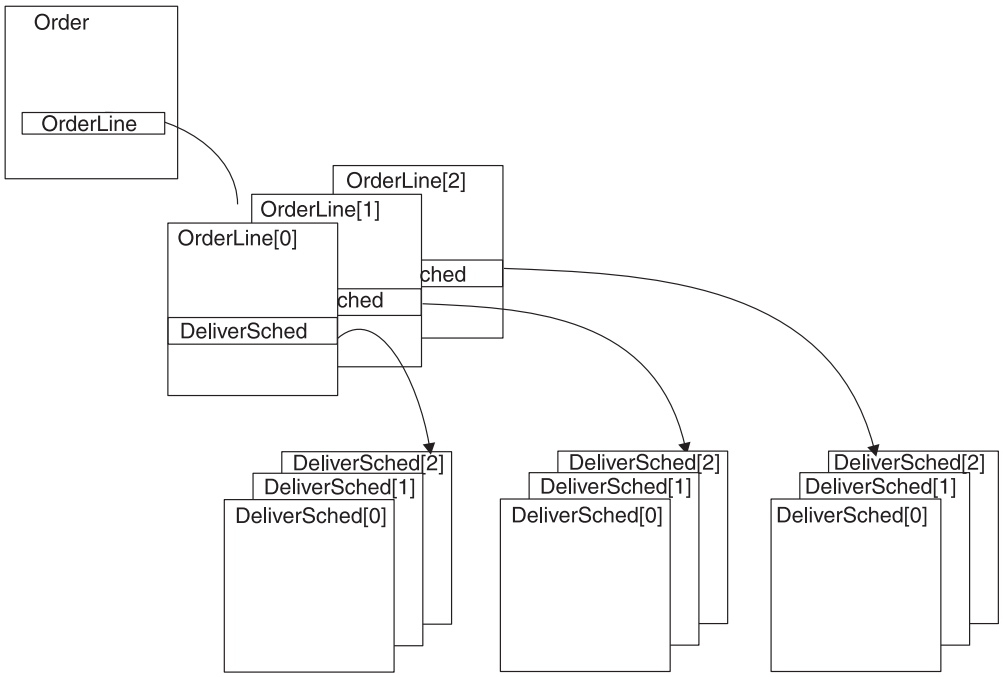

図 20. 複数カーディナリティー子ビジネス・オブジェクトを持つソース・ビジネス・オブジェクト

Order のマップに書き込み可能な条件として、以下のものがあります。

- Order 内の別の属性が特定の値を持つ場合にのみ、Order の OrderLine 属性を変換するサブマップを実行する。
- OrderLine 内の別の属性が特定の値を持つ場合にのみ、OrderLine の DeliverSched 属性を変換するサブマップを実行する。
- Order 内の 1 つの属性が特定の値を持つ場合にのみ、OrderLine の DeliverSched 属性を変換するサブマップを実行する。

サブマップ変換の指定手順: サブマップ変換を指定するには、以下の手順を実行します。

1. サブマップとして使用するマップを作成します。

   推奨: これは、他のマップを作成し、保管する場合と同じ方法で行います。IBM の命名規則では、サブマップ名をストリング「Sub_」で始めることを推奨しています。

2. サブマップを System Manager のプロジェクトに保管し、サブマップをコンパイルします。
3. このサブマップを呼び出す必要のある親ビジネス・オブジェクトの属性について、サブマップの変換を指定します。このソース属性には、子ビジネス・オブジェクトを持つ宛先属性にマップされる子ビジネス・オブジェクトが含まれています。

図 21 に示す「サブマップ」ダイアログを使用して、サブマップを呼び出す必要があることを指定します。「サブマップ」ダイアログは、以下のいずれかの方法で表示します。

- 「テーブル」タブから次の手順を実行する。
  a. 親マップ内で、(子ビジネス・オブジェクトである) ソース属性を選択します。
  b. (子ビジネス・オブジェクトである) 必要とする宛先属性を選択します。
  c. 「変換規則」列内のリストから、「サブマップ」をクリックします。
  d. サブマップのソース・ビジネス・オブジェクトであるソース属性ごと、このサブマップの宛先ビジネス・オブジェクトである宛先属性ごとに以上の手順を繰り返します。
- 「ダイアグラム」タブから次の手順を実行する。
  a. 親マップ内で、(子ビジネス・オブジェクトである) ソース属性を選択します。
  b. `Ctrl+Drag` を使用して宛先属性に移動します。つまり、`Ctrl` キーを押しながら、ソース属性を宛先属性にドラッグします。`Ctrl` キーは、マウス・ボタンを放すまで押したままにします。そのようにしないと、操作が正常に終了しません。

     変換に関係するソース属性が子ビジネス・オブジェクトである場合、Map Designer Express はその変換が「サブマップ」であると想定します。宛先属性の「規則」列に自動的に「サブマップ」が割り当てられ、「サブマップ」ダイアログが表示されます。

  ヒント: 「設定」ダイアログの「キー・マップ」タブから、「ダイアグラム」タブの「サブマップ」変換を開始するためのキー・シーケンスをカスタマイズすることができます。詳細については、25 ページの『キー・マップの指定』を参照してください。
- 「サブマップ」変換がすでに定義済みである場合は、「サブマップ」ダイアログを使用して、変換の確認を行うことができます (変換コードの変更も可能です)。以下のいずれかの方法で行います。
  – 変換規則列内の対応するセルをダブルクリックします。

– 変換規則列にある「サブマップ」のビットマップ・アイコンをクリックします。

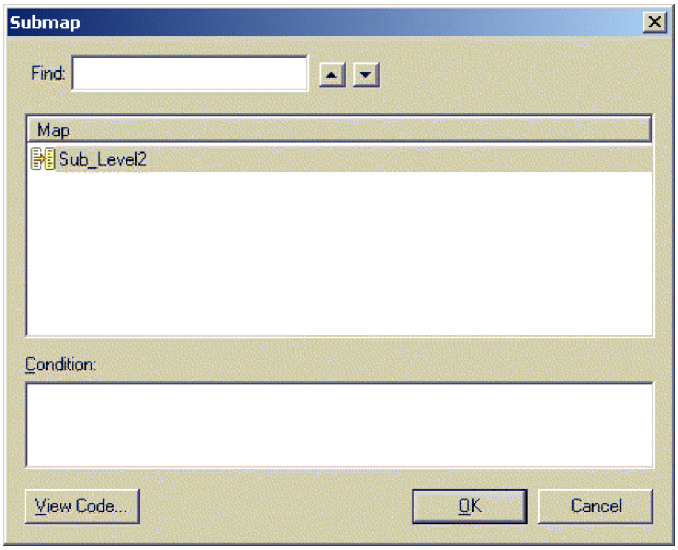

図 21. 「サブマップ」ダイアログ

4. 「サブマップ」ダイアログから、呼び出すサブマップの名前を指定します。「サブマップ」ダイアログには、以下の機能があります。
   - 呼び出すサブマップを示すには、「マップ」領域内のリストからサブマップの名前を選択します。このマップ・リストには、サブマップのソース・ビジネス・オブジェクトおよび宛先ビジネス・オブジェクトのビジネス・オブジェクト定義が、ユーザーが選択したソース属性および宛先属性と同じである、という条件を満たしているマップが表示されます。

     ヒント: 特定のサブマップを見つけるには、「検索」フィールドにサブマップの名前を入力してください。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。
   - サブマップに条件を指定するには、「サブマップ」ダイアログの「条件」領域に入力します。ここで条件を入力できます。あるいは、ダイアログを閉じて、宛先属性の生成コードに条件を入力することもできます。
   - 生成されたコードをカスタマイズするには、「コードを表示」プッシュボタンをクリックします。

     結果: Map Designer Express は Activity Editor を Java 表示で開き、宛先属性の変換コードのサンプルを読み取り専用モードで表示します。変換コードに変更を加えるには、Activity Editor 内の「コードを編集」をクリックします。詳細については、113 ページの『Activity Editor の概要』を参照してください。

     注: Activity Editor で変更を保管すると、変更内容が Map Designer Express に伝達されます。変更内容は、マップを保管するときにも保管されます。
   - 変換の設定を確認するには、「OK」をクリックします。

結果: Map Designer Express が指定されたサブマップを呼び出す Java コードを生成します。Map Designer は、自動的にこのサブマップを呼び出す `runMap()` メソッド呼び出しを作成します。

注: 属性のコードには、式ビルダーを使用して、マップの実行呼び出しを挿入することができます。詳細については、219 ページの『式ビルダーを使用したサブマップの呼び出し』を参照してください。

一致関係の相互参照

ソース属性によっては、関係表を参照して、宛先属性にどんな値を設定するかを調べる必要がある場合もあります。これは、*相互参照* 変換を使用して実行できます。

相互参照変換の指定手順: 相互参照変換を指定するには、以下の手順を実行します。

1. 他の変換についてすでに説明した方法でソース属性および宛先属性を選択します。どちらもビジネス・オブジェクトである必要があります。
2. 対応する変換セル内で「相互参照」を選択します。

   結果: 「相互参照」ダイアログが表示されます。

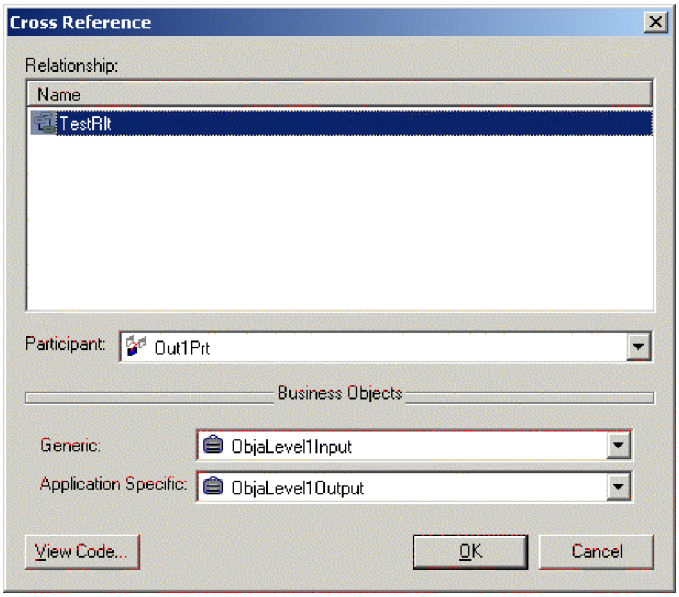

図 22. 「相互参照」ダイアログ

3. このダイアログで、リストから関係名を選択します。

   結果: 「参加者」コンボ・ボックスが、選択された関係の全参加者を示した状態で表示されます。デフォルトでは、「ビジネス・オブジェクト」コンボ・ボックスが、マップ・プロパティーに定義されたマップの役割に従って表示されます。コンボ・ボックスは変更可能です。

カスタム変換の作成

カスタム 変換では、Activity Editor を使用して変換のアクティビティーをグラフィカルにカスタマイズするか、あるいは、ソース属性を宛先属性に変換する Java コードを入力します。

カスタム変換の指定手順: カスタム変換を定義するには、以下のいずれかのマップ・タブから手順を実行します。

- 「テーブル」タブから
  1. ソース属性を選択します。
  2. 必要な宛先属性を選択します。
  3. 「変換規則」列内のリストから、「カスタム」をクリックします。
- 「ダイアグラム」タブから
  1. ソース属性を選択します。
  2. 必要な宛先属性を選択します。
  3. ソース属性を、宛先ビジネス・オブジェクト・ウィンドウ内の宛先属性にドラッグします。
- カスタム変換がすでに定義済みである場合は、以下のいずれかの方法で変換コードを変更することができます。
  - 変換規則列内の対応するセルをダブルクリックします。
  - 変換規則列にある「カスタム」のビットマップ・アイコンをクリックします。

ヒント: 「設定」ダイアログの「キー・マップ」タブから、「カスタム」変換を開始するためのキー・シーケンスをカスタマイズすることができます。詳細については、25 ページの『キー・マップの指定』を参照してください。

結果: Activity Editor がグラフィック表示で表示されます。Activity Editor の詳細については、113 ページの『Activity Editor の概要』を参照してください。

表 15 に、カスタム変換の定義に役立つ本ガイド内の情報をまとめます。

表 15. カスタム変換の定義

提供されている情報	詳細情報の参照先
Activity Editor を使用して変換コードをカスタマイズする方法	113 ページの『第 5 章 マップのカスタマイズ』
関係属性に対する関係を作成する方法	関係の一般的な概要については、245 ページの『第 6 章 関係の概要』を参照してください。
1. Map Designer Express を使用して、関係を含むビジネス・オブジェクトにマップを作成する。	15 ページの『第 2 章 マップの作成』
2. Relationship Designer Express を使用して関係を定義する。	259 ページの『第 7 章 関係定義の作成』
3. Map Designer Express に戻って属性間の関係をコーディングする。	281 ページの『第 8 章 関係のインプリメント』
実行可能なより複雑な変換	190 ページの『その他の属性の変換方法』
内容ベースのロジック	190 ページの『内容ベースのロジック』
日付の形式設定	195 ページの『日付の形式設定』
ストリングの処理	199 ページの『式ビルダーを使用したストリング変換』

注: Activity Editor から生成コードを変更することによって、既存の変換をカスタマイズすることもできます。自動更新モードでコードを変更する場合、Activity Editor が確認を要求します。ユーザーが変更を確認すると、Activity Editor がカスタマイズしたコードを保管します。「テーブル」タブまたは「ダイアグラム」タブの「変換規則」列の中の変換アイコンのラベルが、黒の通常テキストから青のイタリック・テキストの表示に変わります。この青のアイコン・ラベルによって、自動更新モードのコード (Map Designer Express に生成されたコード) とユーザーがカスタマイズしたコードの区別がしやすくなります。

「設定」ダイアログで設定を変更して、Activity Editor に確認しないように指示することもできます。

マップの保管

マップ定義を後で使用できるように保存するには、マップを保管する必要があります。Map Designer Express はマップを保管する前に、まずマップを検証します。詳細については、90 ページの『マップの検証』を参照してください。

Map Designer Express では、以下の 2 通りの方法で現行マップを保管することができます。

- 55 ページの『マップのプロジェクトへの保管』
- 57 ページの『マップのファイルへの保管』

要確認: Map Designer Express がマップを保管するためには、マップが現在開いている必要があります。

マップのプロジェクトへの保管

マップ定義には、System Manager のプロジェクト内のマップ情報が格納されます。このマップ定義には、マップの以下についての情報が入っています。

- マップのプロパティーを含む一般的なマップ情報
- 変換のマッピングを含むマップ・デザイン
- カスタム変換コード

マップを System Manager のプロジェクトに保管するには、表 16 に示すアクションのいずれかを実行することができます。

表 16. マップのプロジェクトへの保管

必要なアクション. .	実行する操作. .
マップ定義を、現在開いているマップの名前で保管する場合	次のいずれかを行います。 • 「ファイル」> 「保管」サブメニューから「プロジェクトに」を選択します。 • キーボード・ショートカット Ctrl+S を使用します。 • 標準ツールバーにある「マップをプロジェクトに保管」ボタンをクリックします。

表 16. マップのプロジェクトへの保管 (続き)

必要なアクション. .	実行する操作. .
マップ定義を、現在開いているマップとは異なる名前で保管する場合	次のいずれかを行います。 • 「ファイル」> 「別名保管」サブメニューから「プロジェクトに」を選択します。 • キーボード・ショートカット `Ctrl+Alt+S` を使用します。 **結果:** Map Designer Express が「マップを別名保管」ダイアログを表示し、ここでマップ名を指定することができます。

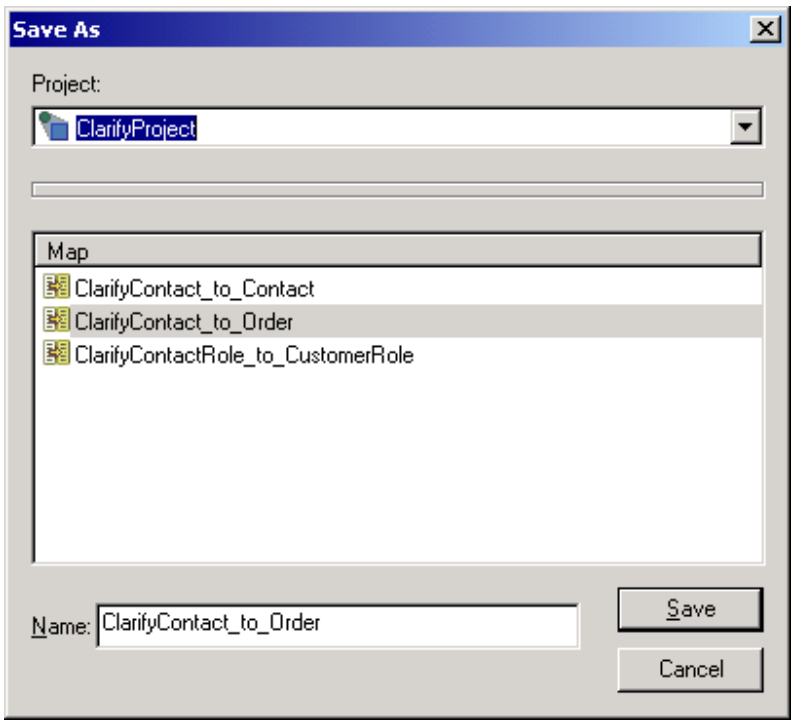

図 23. 「別名保管」ダイアログ

マップを保管すると、Map Designer Express がマップ定義とマップの内容を System Manager のプロジェクトに保管します。マップの内容は XML データとして保管されます。

注: 「マップをコンパイル: コンパイル前にマップを保管」オプションを使用すれば、コンパイル前に System Manager のプロジェクトにマップを自動的に保管するかどうかを指定できます。デフォルトでは、このオプションは使用可能です。このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。

ヒント: 既存のマップの名前を変更するには、「ファイル」> 「別名保管」サブメニューから「プロジェクトに」を選択します。

マップのファイルへの保管

マップ定義は、マップ定義ファイル と呼ばれるオペレーティング・システム・ファイルの中にテキストとして保管できます。マップ定義ファイルには、完全なマップ定義が入っています。つまり、このファイルは、マップ定義の以下の部分を XML (Extended Markup Language) 形式で表します。

- マップのプロパティーを含む一般的なマップ情報
- 非圧縮形式の変換マッピングを含むマップの内容

推奨: Map Designer Express はマップ定義ファイルを `.cwm` という拡張子で作成します。マップ定義ファイルを区別するため、ファイル拡張子 (`.cwm`) を使用することなど、マップ定義ファイルの命名規則に従う必要があります。

既存のマップ定義ファイルを開くことによって、マップ定義が Map Designer Express にインポートされます。詳細については、63 ページの『ファイルからのマップのオープン手順』を参照してください。

表 17 に示す方法のいずれかで、現在開いているマップをマップ定義ファイルに保管することができます。

表 17. マップのマップ定義ファイルへの保管

必要なアクション. . .	実行する操作. . .
マップ定義を、現在開いているマップの名前で以下の形式で保管したい場合 *MapName*.cwm (ここで、*MapName* は現在開いているマップの名前) **注:** 現在開いているマップをファイルから開かない場合、Map Designer Express は常に「File Save」ダイアログを開きます。	次のいずれかを行います。 • 「ファイル」> 「保管」サブメニューから「ファイルに」を選択します。 • キーボード・ショートカット `Ctrl+E` を使用します。 • 標準ツールバーにある「マップをファイルに保管」ボタンをクリックします (図 23 を参照)。
マップを、指定のマップ定義ファイルに保管します。Map Designer Express が、ファイル名を選択するためのダイアログ・ボックスを表示します。	次のいずれかを行います。 • 「ファイル」 > 「別名保管」サブメニューから「ファイルに」を選択します。 • キーボード・ショートカット `Ctrl+Alt+F` を使用します。

注: 「ファイル」>「保管」または「ファイル」>「別名保管」メニューから「ファイルに」オプションを選択すると、Map Designer Express はファイル名を選択するためのダイアログ・ボックスを表示します。このファイル名でファイルを識別します。ファイル名は必ずしもマップの名前でなくてもかまいません。

例: `MapA` というマップを `fileA.cwm` という名前のファイルに保管することができます。この `fileA` ファイルに、`MapA` というマップのマップ定義が格納されます。Map Designer Express が `fileA` マップ定義ファイルを開いたとき、`MapA` のマップ定義が表示されます。

ヒント: マップをエクスポートすると、マップのみがコピーされます。

完全性のチェック

2 つの大きなビジネス・オブジェクト同士をマップするときは、必須属性を見落としがちです。まだマップされていない属性を検索して、必要な変換すべての指定が完了したかどうかを確認することができます。このような属性を、*リンクされていない属性* と呼びます。

完了したかどうかをチェックするには、以下の手順を実行します。

- 「編集」メニューから「検索」を選択し、「検索」コントロール・ペインの中の「リンクされていない属性」オプションをクリックします。

  結果: Map Designer Express が変換コードが存在しない属性のリストを表示します。詳細については、79 ページの『マップの情報の検索』を参照してください。

注: コードが完成したら、コンパイルし、テストする必要があります。マップのコンパイルについては、91 ページの『マップのコンパイル』を参照してください。マップのテストについては、94 ページの『マップのテスト』を参照してください。

マッピングの標準

このセクションでは、以下に示すマップの手順上の基準について説明します。

- 『個々の属性のマップについてのヒント』
- 59 ページの『属性のコメント・フィールドでのコメントの設定』

個々の属性のマップについてのヒント

以下に、個々の属性をマップするときの一般的な方法を示します。

- 属性のマップに関係の管理が含まれない 場合は、まずソース属性を宛先属性にコピー (43 ページの『ソース属性の宛先属性へのコピー』を参照) し、その後生成されるコードを必要に応じて変更します。
- 属性のマップで、マッピング API でのメソッド呼び出しが必要な場合は、属性をコピーせずにコードを作成します。
- ソース属性が null のときに宛先属性にデフォルト値が必要な場合は、属性をコピーして、生成されるコードにソース属性のチェックのための `if` ステートメントが 2 つ組み込まれていることを確認します。以下の指定を行うことができます。
  - 両方の `if` ステートメントで、`else` ステートメントの中にデフォルトを指定します。
  - ソース属性に null があるかどうかチェックしてから、デフォルト値を追加するコードの先頭に、`if` ステートメントをさらに 1 つ追加します。残りのコードを `else` ステートメントの中に置きます。

要確認: `ObjectEventid` 属性はマップしないでください。 InterChange Server Express では、固有の処理目的で `ObjectEventId` を予約しています。`ObjectEventId` を宛先属性として含むカスタム・コードは、実行されません。

属性のコメント・フィールドでのコメントの設定

属性にコメントを付けておくと、マップが読みやすくなります。ただし、Map Designer Express では属性へのコメントは*自動生成されません*。表 18 に、属性のコメントについて推奨される基準を、宛先属性に関連付けられている変換のタイプごとに示します。

表 18. 属性のコメントの設定

状況	属性のコメントの設定
子ビジネス・オブジェクトがマップされていない場合	=No mapping
「値を設定」変換	=SET VALUE(*value*)
「移動」変換	=MOVE
「結合」変換	=JOIN(*srcAttr1*, *srcAttr2*, ...)
「分割」変換	=SPLIT(*srcAttr* [*index*])
子ビジネス・オブジェクトで、属性を確認するためにオブジェクトを展開する必要があることを示すサブマップを*呼び出さずに* マップが行われている場合	=Mapping here
サブマップを呼び出すコードが生成される場合	=SUBMAP(*mapName*)
属性のマップに以下のような関係をインプリメントするマッピング API 呼び出しが含まれている場合 • retrieveInstances() • retrieveParticipants() • maintainSimpleIdentityRelationship() • maintainCompositeRelationship() • IdentityRelationship クラス内の foreignKeyLookup() と foreignKeyXref() *以外の* 他のすべてのメソッド	=Relationship(*type*) ここで、*type* には以下を指定できます。 • identity • lookup • custom
属性のマップに foreignKeyLookup() が含まれている場合	=foreignKeyLookup()
属性のマップに ruiLnKeyLoXiLr が含まれている場合	=foreignKeyXref()
上記にリストされている変換*以外の* カスタム変換 (関係または外部キー)	=CUSTOM(*summary*)
属性のコードに、動詞の設定以外のものが含まれていない場合	=SET VERB

第 3 章 マップの処理

この章では、マップの作成後に使用できる Map Designer Express の拡張機能について説明します。

この章の内容は次のとおりです。

- 『マップのオープンとクローズ』
- 64 ページの『マップ・プロパティー情報の指定』
- 67 ページの『マップ文書の使用』
- 72 ページの『マップの自動化の使用』
- 79 ページの『マップの情報の検索』
- 81 ページの『テキストの検索および置換』
- 82 ページの『マップの印刷』
- 82 ページの『オブジェクトの削除』
- 85 ページの『実行順序の使用』
- 86 ページの『ポリモアフィック・マップの作成』
- 87 ページの『InterChange Server Express からのマップのインポートとエクスポート』

マップのオープンとクローズ

Map Designer Express はタブ・ウィンドウ内で一度に 1 つのマップを表示します。このマップは*現行マップ* と呼びます (「現在開いているマップ」と呼ぶこともあります)。次の Map Designer Express の手順を使用して、現行マップをどのマップにするかを制御できます。

- 『マップのオープン』
- 64 ページの『マップを閉じる』

マップのオープン

マップは Map Designer Express で開いた後、マップ・タブに情報を表示したり、その情報を変更する必要があります。 Map Designer Express でマップを開くとき、**「開くときにマップを検証」**設定が使用可能になっていると、はじめにマップに対して一連の検証が実行されます。

注: Map Designer Express でマップを開くときにマップを検証するかどうかを指定するには、**「マップを開く: 開く時にマップを検証」**オプションを使用します。デフォルトでは、このオプションは使用可能です。

大きなビジネス・オブジェクト (数千の属性を持つ) を使用するマップを開くときにこの設定が使用可能になっていると、Map Designer Express によるマップのオープンに時間がかかることがあります。 このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。

Map Designer Express はマップに次のような検証を行います。

- マップが使用する各ビジネス・オブジェクト定義が System Manager のプロジェクトに定義されていることを確認する。
- マップのすべての属性が、System Manager のプロジェクトに定義されているように、指定されたビジネス・オブジェクト定義に存在していることを確認する。
- マップの各属性のタイプが、System Manager のプロジェクトに定義されているように、指定されたビジネス・オブジェクト定義のタイプと一致していることを確認する。
- 変換を検証する。
  - 実行順序が正しいこと、つまり実行順序が一意であり、正であり、連続していることを確認する。
  - 属性が相互に循環依存関係になっていないかどうかを確認する。循環変換が見つかった場合、Map Designer Express は出力ウィンドウに循環規則を表示する。
  - 変換情報を検査する。

    「移動」変換: 1 つのソース属性のみ関係する。

    「結合」変換: 複数のソース属性が関係する。

    「分割」変換: 1 つのソース属性のみ関係する。分割インデックスは 0 以上になる。分割区切り文字は空ではない。

    「値を設定」変換: ソース属性は関係しない。値が指定されている。

    「サブマップ」変換: 少なくとも 1 つのソース属性が関係する。サブマップ名が指定されている。

    「相互参照」変換: 1 つのソース属性のみ関係する。

Map Designer Express では次の方法でマップを開きます。

- 62 ページの『System Manager のプロジェクトからのマップのオープン手順』
- 63 ページの『ファイルからのマップのオープン手順』

System Manager のプロジェクトからのマップのオープン手順

System Manager のプロジェクトからマップを開くには、次のステップを実行します。

1. 次のいずれかの方法で「Open a Map from a Project」ダイアログを開きます。
   - 「ファイル」>「開く」>「プロジェクトから」をクリックします。
   - Ctrl + O のキーボード・ショートカットを使用する。
   - 標準ツールバーで「マップをプロジェクトから開く」ボタンをクリックする。

結果: 「マップを開く」ダイアログが表示されます。

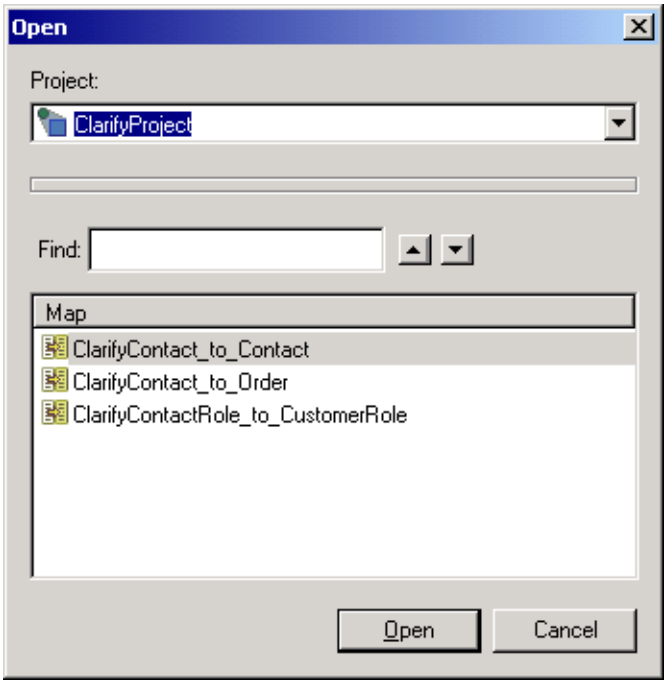

図 24. 「マップを開く」ダイアログ

2. プロジェクトを選択します。
3. System Manager のプロジェクトに現在定義されているマップのリストからマップの名前を選択します。

   ヒント: 特定のマップ名を探し出すには、「検索」フィールドにその名前を入力します。上下矢印を使用すると、マップ・リストの中をスクロールします。
4. 「開く」ボタンをクリックします。プロジェクトからマップが開きます。

ファイルからのマップのオープン手順

マップ定義はマップ定義ファイル と呼ばれるオペレーティング・システム・ファイルに XML 形式で保管できます。マップ定義ファイルを作成するには、Map Designer Express でマップ設計ファイル (.cwm) としてマップを保管します。詳細については、57 ページの『マップのファイルへの保管』を参照してください。

マップ定義ファイルを開くときは、Map Designer Express でマップを開きます。

マップ定義ファイルを開くには、次のステップを実行します。

1. 次のいずれかの方法で「Open a Map from a File」ダイアログを開きます。
   - 「ファイル」>「開く」>「ファイルから」をクリックします。
   - Ctrl + I のキーボード・ショートカットを使用する。
   - 標準ツールバーで「マップをファイルから開く」ボタンをクリックする。

   結果: 「マップを含むファイルを開く」ダイアログ・ボックスが表示されます。

2. 開きたいマップ定義ファイルを選択します。ファイルは Map Designer Express で作成された .cwm ファイルでなければなりません。

   結果: Map Designer Express でマップ定義ファイルが開きます。マップ・タブにマップ情報が表示されます。

   要確認: Map Designer Express でマップを開いても、マップは自動的にはプロジェクトに保管されません。このマップをプロジェクトに保管するには、ステップ 3 に進んでください。

3. System Manager でプロジェクトにマップを保管します。詳細については、55 ページの『マップのプロジェクトへの保管』を参照してください。

   規則: マップをコンパイルするには、System Manager でプロジェクトにマップを保管する必要があります。マップをコンパイルするには、「ファイル」メニューから「コンパイル」を選択します。詳細については、94 ページの『マップのテスト』を参照してください。

マップを閉じる

タブ・ウィンドウに表示されている現行マップを閉じるには、次のいずれか 1 つの処置を行います。

- 61 ページの『マップのオープン』で説明しているいずれかの方法で新しいマップを開く。

  結果: Map Designer Express は現行マップを閉じた後に新規マップを開きます。

- 「ファイル」メニューから「閉じる」を選択する。

  結果: Map Designer Express は現行マップを閉じ、タブ・ウィンドウを消去します。新しいマップを現行マップにするには、新規マップを作成するか、既存のマップを開きます。

- 次のいずれかの方法で Map Designer Express を終了する。
  - 「ファイル」メニューから「終了」を選択する。
  - Alt + F4 のキーボード・ショートカットを使用する。

  結果: Map Designer Express は現行マップを閉じた後に終了します。

注: 現行マップを最後に保管した後に変更を加えた場合、Map Designer Express は確認ボックスを表示してマップのクローズを確認します。

マップ・プロパティー情報の指定

「マップ・プロパティー」ダイアログ (図 25 を参照) を使用して、マップのプロパティー情報を表示および指定します。「マップ・プロパティー」ダイアログを表示するには、次のいずれかの処置を実行します。

- 「編集」メニューから「マップ・プロパティー」を選択する。
- Ctrl + Enter のキーボード・ショートカットを使用する。
- 「ダイアグラム」タブのマップ・ワークスペースで右マウス・ボタンをクリックし、コンテキスト・メニューから「マップ・プロパティー」を選択する。

「マップ・プロパティー」ダイアログには次のタブがあります。

- 「一般」タブ
- 「ビジネス・オブジェクト」タブ

図 25 は「マップ・プロパティー」ダイアログの「一般」タブを示しています。

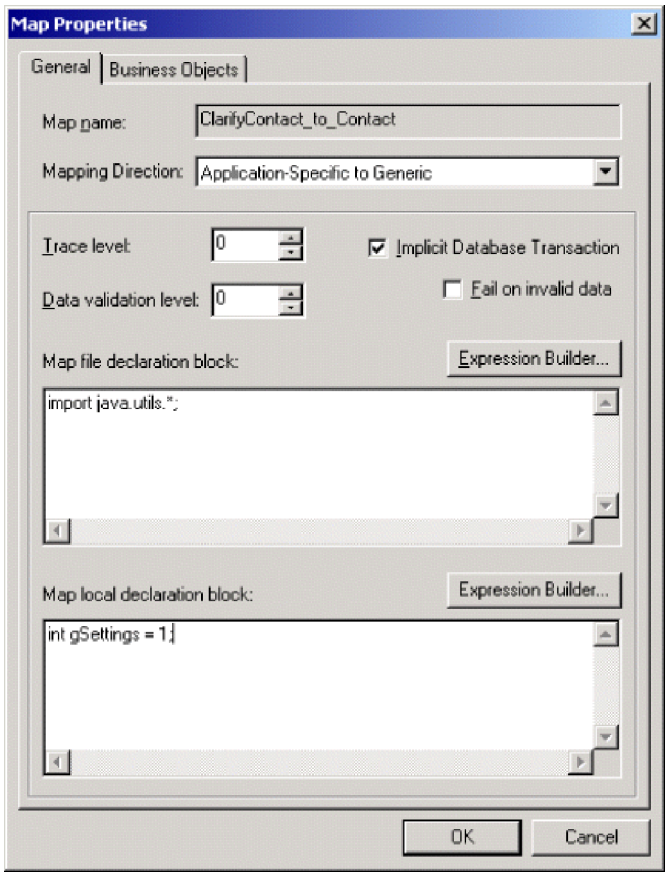

図 25. 「マップ・プロパティー」ダイアログの「一般」タブ

一般プロパティー情報の定義

「マップ・プロパティー」ダイアログの「一般」タブには、表 19 に示す一般プロパティー情報が表示されます。

表 19. 一般マップ・プロパティー情報

一般マップ・プロパティー	説明	詳細情報の参照先
マップ名	ダイアログに表示されているプロパティーを持つマップを識別します。このフィールドは新規マップの作成時に初期設定され、編集はできません。	該当なし

表 19. 一般マップ・プロパティー情報 (続き)

一般マップ・プロパティー	説明	詳細情報の参照先
マッピングの役割	マップの目的を識別します。マッピングの役割に指定できる値は、次のとおりです。 • アプリケーション固有から汎用 • 汎用からアプリケーション固有 • その他 (特定のマッピングの方向が関連付けられていないマップの場合) **注:** 以前定義され、このプロパティー情報がないマップの場合、コンボ・ボックスは空になります。関係変換規則を使用しない限りは問題ありません。最初に関係変換規則を作成した場合、この値が空であると、Map Designer Express はユーザーにこの値の入力を求めます。	
ランタイム・プロパティー	マップ・プロパティー (トレース・レベル、データ検証レベル、暗黙的なデータベース・トランザクション、および無効なデータの場合は失敗) を指定し、実行時にマップ・インスタンスに適用されます。これらのプロパティーは、Map Designer Express の「マップ・プロパティー」ダイアログの「一般」タブで指定でき、また System Manager の「マップ・プロパティー」ウィンドウでも指定できます。変更はローカル・ファイル・システムに対して行われます。マップをサーバーに配置しても、ランタイム・インスタンスは更新されません。 **注:** これらのマップ・プロパティーをサーバー・コンポーネント管理ビューから動的に更新するには、マップを右クリックして、コンテキスト・メニューからプロパティーを選択します。変更が自動的にサーバーに更新されます。	
トレース・レベル	マップのトレース・レベルを設定します。	545 ページの『トレース・メッセージの追加』
データ検証レベル	マップの各操作を検査し、受信ビジネス・オブジェクトのデータが変換できないときにエラーをログに記録できるようにします。	205 ページの『カスタム・データ検証レベルの作成』
暗黙的なデータベース・トランザクション	InterChange Server Express がその接続でトランザクションに暗黙的なトランザクション・ブラケット処理を使用するかどうかを決定します。	
無効なデータの場合は失敗	データが無効な場合にマップの実行を失敗させるかどうかを決定します。	205 ページの『カスタム・データ検証レベルの作成』
変数宣言	ユーザー独自の Java 変数を宣言して自分の変換コードで使用できます。詳細については、184 ページの『変数の使用』を参照してください。	
マップ・ファイル宣言ブロック	Java パッケージ (`MapUtils` など) をマップにインポートして変換コード内で使用できるようにします。	174 ページの『Java パッケージと他のカスタム・コードのインポート』
マップ・ローカル宣言ブロック	Map Designer Express の外側で開発されたカスタム Java コードをマップにインポートし、変換コード内で使用できるようにします。	174 ページの『Java パッケージと他のカスタム・コードのインポート』

ビジネス・オブジェクトの定義

「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブは、マップのビジネス・オブジェクトに関する情報を表示します。ソース・ビジネス・オブ

ジェクトと宛先ビジネス・オブジェクト、および定義されている一時ビジネス・オブジェクトがリストされます。詳細については、186 ページの『ビジネス・オブジェクト変数の変更手順』を参照してください。

マップ文書の使用

マップ文書 を作成して、1 つのマップ内または 2 つのマップ間でのすべての変換を表示できます。マップの検査中、各属性を別々に開いて表示するのではなく、1 つの操作ですべての変換を表示したいことがあります。その場合、すべての変換を含むマップ文書を作成できます。マップ文書を使用すると、ネイティブ・マップ変換を自動的に文書化できます。

このセクションの内容は次のとおりです。

- マップ文書を構成する 2 つの HTML ファイルの説明
- 新規マップ文書の作成方法
- マップ文書の表示方法
- マップ文書の印刷方法

マップ文書とは

マップ文書 は 1 つのマップ (またはマップの集合) のすべての変換を記述する 2 つの HTML ファイルから構成されます。

- 表形式でマップ変換を記述するマップ・テーブル・ファイル。

  マップ・テーブル・ファイルの名前は *mapDoc*.HTM になります。
- マップ変換のコードが入った Java コード・ファイル。

  Java コード・ファイルの名前は *mapDoc*JavaCode.HTM になります。

これら両方の HTML ファイルで、*mapDoc* はマップ文書のユーザー指定名を示します。

マップ文書には、すべての属性、マップ変換がある属性のみ、またはマップ変換がない属性 (リンクされていない属性) のみについての情報を格納できます。すべての属性を指定した場合、ソース・ビジネス・オブジェクトおよび宛先ビジネス・オブジェクトにあるリンクされていない属性のリストも含まれます。

以下のセクションでは、マップ文書の 2 つの HTML ファイルの形式について説明します。

マップ・テーブル・ファイルの形式

マップ・テーブル・ファイル *mapDoc*.HTM は表形式でマップ変換を記述します。

- マップ文書が *1* つのマップのみ を記述する場合、Map Designer Express は単一マップ・マップ・テーブルを作成する。
- マップ文書が *2* つのマップ を記述する場合、Map Designer Express は複数マップ・マップ・テーブルを作成する。

単一マップ・マップ・テーブル: *単一マップ・マップ・テーブル* は 1 つのマップのマッピング・フローを記述します。つまり、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクト間の変換を記述します。単一マップ・マップ・テーブルには次の列があります。

- 「ソース属性」はソース・ビジネス・オブジェクトの属性の名前を示す。
- 「変換規則」はソース・ビジネス・オブジェクトの属性 (左側の列) と宛先ビジネス・オブジェクトの属性 (右側の列) 間のマッピング変換の種類を記述する。この列に示された変換は、マップ用の *mapDoc*JavaCode.HTM Java コード・ファイルの属性がある位置へのハイパーテキスト・リンクです。
- 「宛先属性」は宛先ビジネス・オブジェクトの属性の名前を示す。

図 26 は単一マップ・マップ・テーブルを含んでいる HTML ファイルを示しています。

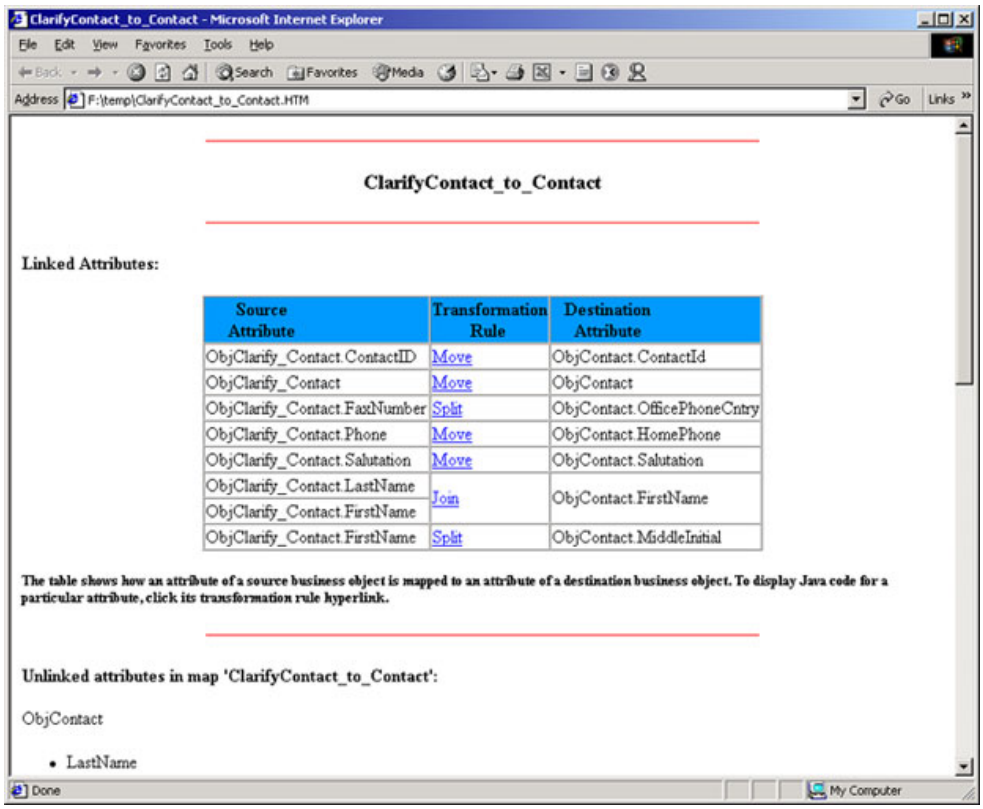

図 26. 単一マップ・マップ・テーブル

注: 「マップ文書を作成」ダイアログの「コメント」チェック・ボックスを使用可能にした場合、マップ・テーブルには「コメント」という 4 番目の列が含まれます。この列は、テーブルの各宛先属性に関するコメントを示します。

複数マップ・マップ・テーブル: *複数マップ・マップ・テーブル* は 2 つのマップ間のマッピング・フローを記述します。つまり、インバウンド・マップ (アプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクト) および

アウトバウンド・マップ (汎用ビジネス・オブジェクトからアプリケーション固有のビジネス・オブジェクト) における変換を記述します。複数マップ・マップ・テーブルには次の列があります。

- 「ソース属性」はアプリケーション固有のビジネス・オブジェクトの属性名を示す。
- 最初の「変換規則」は、アプリケーション固有のビジネス・オブジェクトの属性 (左側の列) と汎用ビジネス・オブジェクトの属性 (右側の列) 間のマッピング変換の種類を記述する。この列に示された変換は、インバウンド (アプリケーション固有から汎用) マップ用の *mapDoc*JavaCode.HTM Java コード・ファイルの属性がある位置へのハイパーテキスト・リンクです。
- 「Common Attribute」は汎用ビジネス・オブジェクトの属性の名前を示す。
- 2 番目の「変換規則」は、汎用ビジネス・オブジェクトの属性 (左側の列) とアプリケーション固有のビジネス・オブジェクトの属性 (右側の列) 間のマッピング変換の種類を記述する。この列に示された変換は、アウトバウンド (汎用からアプリケーション固有) マップ用の *mapDoc* JavaCode.HTM Java コード・ファイルの属性がある位置へのハイパーテキスト・リンクです。
- 「宛先属性」はアプリケーション固有のビジネス・オブジェクトの属性名を示す。

図 27 は複数マップ・マップ・テーブルを含んでいる HTML ファイルを示しています。

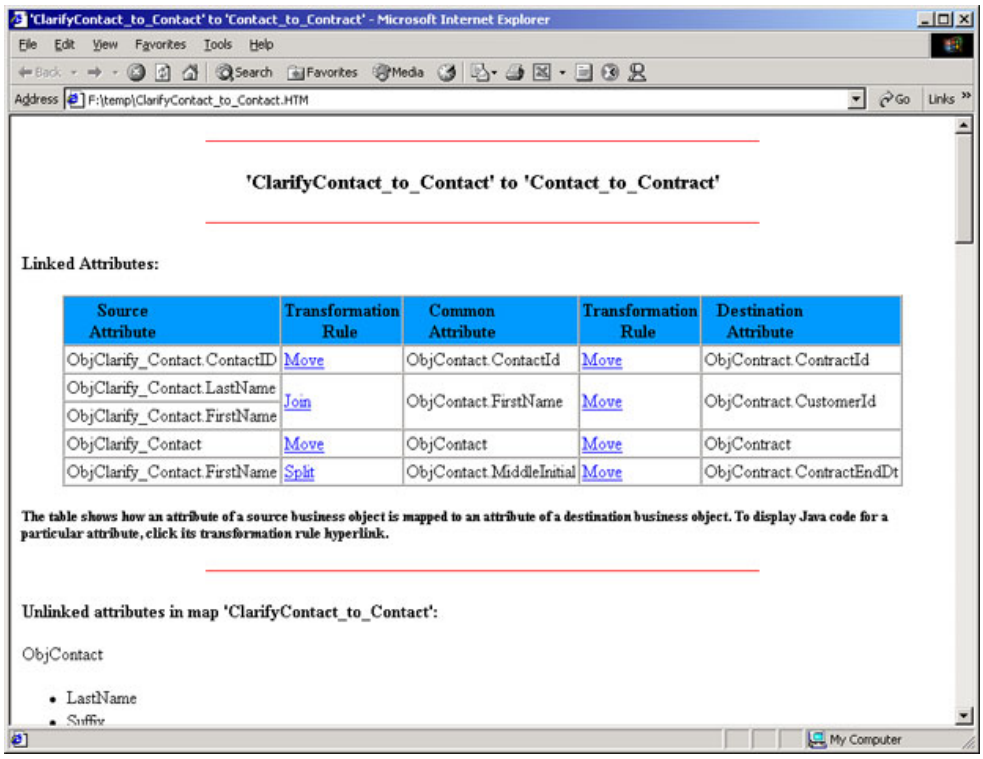

図 27. 複数マップ・マップ・テーブル

Java コード・ファイルの形式

Java コード・ファイル *mapDoc*JavaCode.HTM は、マップに関するさらに詳しい情報を提供します。ファイルには、変換を実行する Java コードが含まれています。このコードは標準的なプログラム形式です。マップの検査中、各属性を別々に開いて表示するのではなく、1 つの操作ですべての変換を表示する場合、Java コード・ファイルが役に立ちます。

マップ文書の作成手順

マップ文書を作成するには、次のステップを実行します。

1. 「ファイル」メニューから「マップ文書を作成」を選択する。

   結果: 「マップ文書を作成」ダイアログが表示されます (図 28 を参照)。

2. 「マップ文書を作成」ダイアログからマップ文書の構成オプションを選択します。
   - プロジェクトを指定する。
   - マップ文書に関係するマップを指定する。

     ガイドライン: 「2 つのマップでマップ・フローを表示」チェック・ボックスを選択していない 場合、ドロップダウン・リストから 1 つのマップしか選択できません。ドロップダウン・リストには、現在定義されているすべてのマップが含まれている。マップが現在開いている場合、デフォルトでその名前が表示されます。

     「2 つのマップでマップ・フローを表示」チェック・ボックスを選択した場合、2 番目のドロップダウン・リストが使用可能になる。この 2 番目のドロップダウン・リストは、最初のマップと同じ汎用ビジネス・オブジェクトを共用するマップのみを示します。このリストから、マップ文書に入れる 2 番目のマップの名前を選択できます。
   - マップ文書に入れる宛先ビジネス・オブジェクトの属性を指定する。

     すべての属性、マップされた属性のみ、マップされていない属性のみのいずれをマップ文書に入れるか、該当するラジオ・ボタンをクリックして示します。
   - 新規マップ文書の名前を指定する。

     ガイドライン: 「参照」ボタンをクリックし、マップ文書ファイルの場所を検索できます。入力したマップ文書名には自動的にサフィックス .HTM が付加されます。したがって、ファイル拡張子を指定する必要はありません。

3. マップ文書の作成を開始するには、次のどちらかのオプションを選択します。
   - 「保管」をクリックして選択したマップをマップ文書に保管する。
   - 「保管/表示」をクリックし、選択した文書をマップ文書に保管して、この新規マップ文書を HTML ブラウザーに表示する。

図 28 は「マップ文書を作成」ダイアログを示しています。

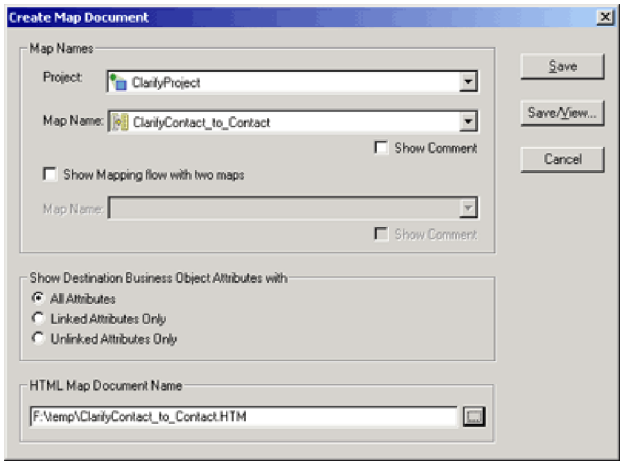

図 28. 「マップ文書を作成」ダイアログ

マップ文書を作成すると、Map Designer Express は HTML (ハイパーテキスト・マークアップ言語) ファイル (*mapDoc*.HTM) および関連する Java コード・ファイル (*mapDoc*JavaCode.HTM) としてマップ文書を作成します。*mapDoc* は「マップ文書構成」ダイアログに指定したマップ文書名を示します。

マップ文書の表示

次のいずれかの方法でマップ文書を表示できます。

- 次のどちらかの方法で既存のマップ文書を開く。
  - 「ファイル」メニューから「マップ文書を表示」を選択する。
  - キーボード・ショートカット Ctrl+M を使用します。

  結果: 「開く」ダイアログに使用可能なマップ文書ファイルが表示されます。読み取る HTML マップ文書を指定し、「開く」をクリックします。
- 「Map Document Configuration」ダイアログで「保管/表示」をクリックして、新規マップ文書を開く。
- マップ文書が入ったディレクトリーに移動し、必要なファイルをダブルクリックする。

結果: ブラウザーが起動し、選択した HTML マップ文書ファイルが表示されます。

ガイドライン: さらに、マップ・テーブルの「Mapping Action」列の項目をクリックして、特定の変換に関連した Java コードを表示できます。ブラウザーには、関連したソース属性と宛先属性間のマッピングをインプリメントする対応 Java コード・セグメントが表示されます。

マップ文書の印刷

マップ文書ファイルを印刷するには、次のステップを実行します。

1. 必要なファイルを HTML ブラウザーに表示します。

   詳細については、71 ページの『マップ文書の表示』を参照してください。

2. 以下のいずれかの方法で、表示している HTML ファイルをブラウザーから印刷します。
   - ブラウザーの「ファイル」メニューから「印刷」を選択する。
   - Ctrl + P のキーボード・ショートカットを使用する。
   - 標準ツールバーの「印刷」ボタンをクリックする。

マップの自動化の使用

マップの自動化によって、類似する属性を持つビジネス・オブジェクト間のマップを自動的に作成できます。また、任意のマップに対して反転マップを生成できます。

このセクションの内容は次のとおりです。

- 『マップの自動作成』
- 76 ページの『反転マップの自動作成』
- 78 ページの『自動化の同義語の使用』

マップの自動作成

Map Designer Express では、ソース属性と宛先属性の名前が同じであるビジネス・オブジェクト間のマップを自動生成できます。異なるビジネス・オブジェクトでも、特定のエレメントを共通に持っていることがあります。例えばカスタマー・ビジネス・オブジェクトには、カスタマー・データを保守するために、通常はファーストネーム、ラストネーム、住所、および郵便番号の属性があります。

ビジネス・オブジェクトを自動的にマップするためには、Map Designer Express は、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの間で名前が一致している属性を検索して、移動変換を使用します。マッピングは、対応するレベルでのみ発生します。つまり、ソース・ビジネス・オブジェクトのトップレベルの属性は、宛先ビジネス・オブジェクトのトップレベルの属性にマップされます (それ以外のレベルにはマップされません)。同様に、ソース側の子ビジネス・オブジェクトは、対応する子オブジェクトが宛先ビジネス・オブジェクトの同じレベルで検出された場合にのみ、マップの自動化が考慮されます。

マップの自動作成手順

始める前に: ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトが指定されたマップ定義ファイルを用意する必要があります。新規マップ・ウィザードで新規マップ定義ファイルを作成する方法については、34 ページの『マップ定義の作成手順』を参照してください。

マップを自動的に作成するには、次のステップを実行します。

1. 「ツール」メニューから「自動マッピング」を選択します。

   結果: 「自動マッピング」ダイアログが表示されます。ここでは、Map Designer Express が属性の検索に使用するプレフィックスまたはサフィックスを指定できます。

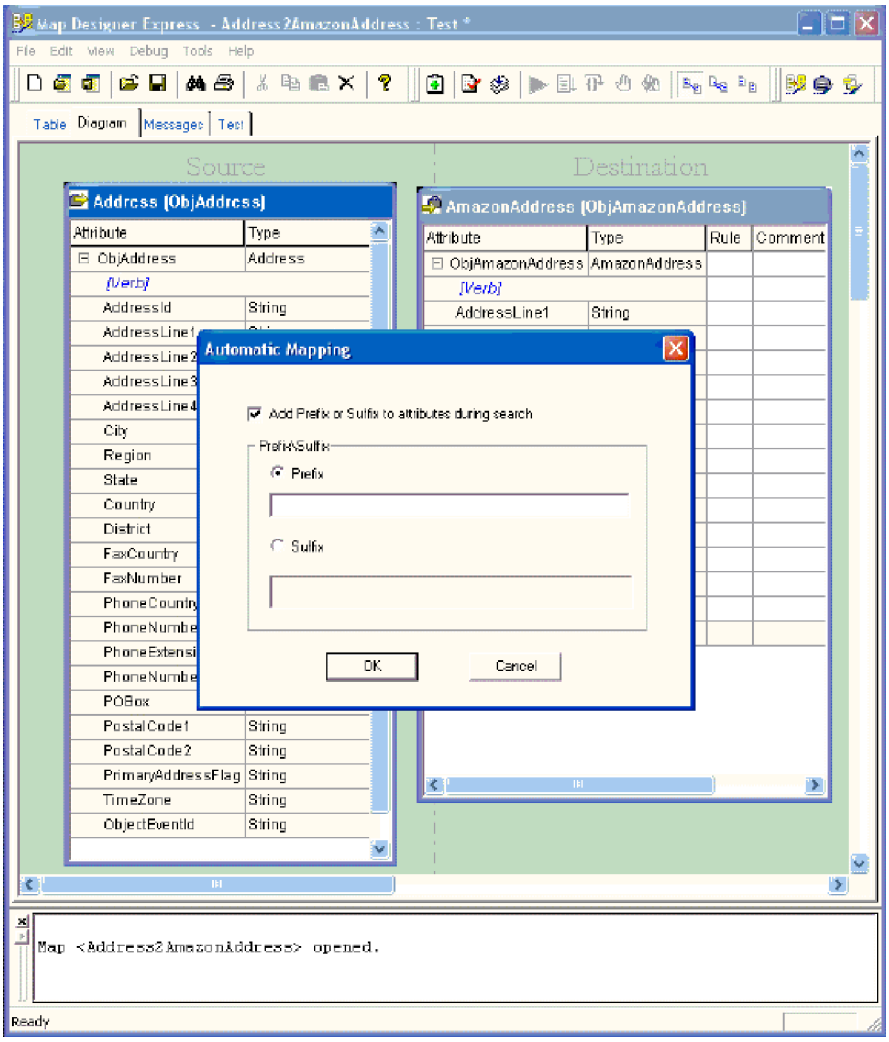

図 29. プレフィックスおよびサフィックスの設定ダイアログ

2. このオプションを使用するには「自動マッピング」ダイアログで以下の手順を実行します。
   a. チェック・ボックス「検索中にプレフィックスまたはサフィックスを属性に追加」を選択します。

      注: このオプションは、デフォルトでは使用不可です。
   b. 「プレフィックス」または「サフィックス」を選択します。そして、提供されたスペース内に、特定セッションの検索ストリングに追加するプレフィックスまたはサフィックスを入力します。

      制約事項: どのインスタンスの場合も、選択できるのはサフィックスまたはプレフィックスのみです。検索用として両方を同時に使用することはできません。
   c. 「OK」をクリックします。

注: Map Designer Express では、「設定」ダイアログの「自動マッピング」タブで指定した大/小文字およびデータ型に関する設定も使用されます。

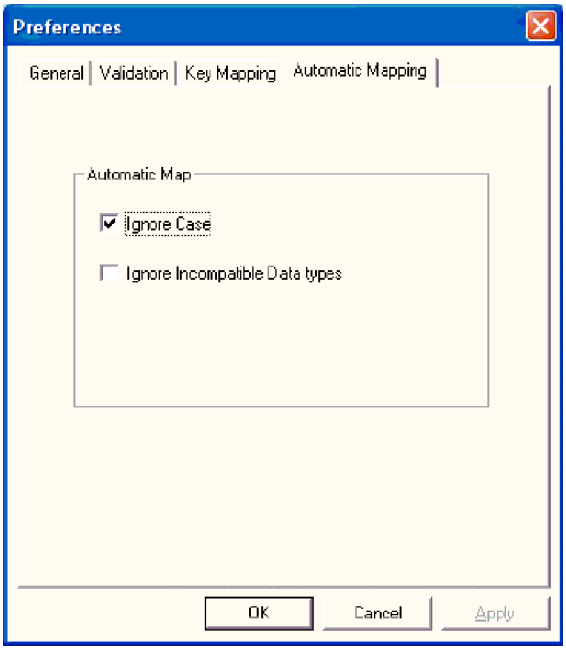

図 30. 「設定」ダイアログの「自動マッピング」タブ

これらの設定については、25 ページの『自動マッピングの指定』を参照してください。

結果: Map Designer Express は、ソース側のすべての属性に対し、宛先側の検索ストリングに追加されたプレフィックスまたはサフィックスを付けて、検索を実行します。一致する属性が宛先ビジネス・オブジェクトで検出されるたびに、ソース属性と、プレフィックスが付加された宛先属性の間で自動マッピングが行われます。

自動マッピングの例

以下に示す自動マッピングの例では、プレフィックスを追加します。

ソース・ビジネス・オブジェクトには以下の属性があるとします。

1. FirstName
2. LastName
3. Address
4. Zip

宛先ビジネス・オブジェクトには、以下の属性があります。

1. ORCL_FirstName
2. ORCL_LastName
3. ORCL_Address
4. Pin
5. State

6. Country

「自動マッピング」ダイアログで、チェック・ボックス「検索中にプレフィックスまたはサフィックスを属性に追加」を選択します。「プレフィックス」スペースに ORCL と入力して、「OK」をクリックします。

注: この例では、以前に「設定」ダイアログの「自動マッピング」タブの設定が「大/小文字を区別しない」(名前の大/小文字を区別せずに検索を実行する) にされたと仮定しています。

結果: Map Designer Express は、ソース側の属性 (FirstName、LastName、および Address) に対し、宛先側の検索ストリング (ORCL_FirstName、ORCL_LastName、ORCL_Address) に追加されたプレフィックス ORCL を付けて、大/小文字を区別しない検索を実行します。一致する属性が宛先ビジネス・オブジェクトで検出されるたびに、ソース属性と、プレフィックスが付加された宛先属性の間で、移動変換を使用した自動マッピングが行われます。この例では、FirstName と ORCL_FirstName、LastName と ORCL_LastName、Address と ORCL_Address の間でマッピングが発生します。他の属性は一致しないため、それらの間でマッピングは行われません。

図 31 に、この例を示します。

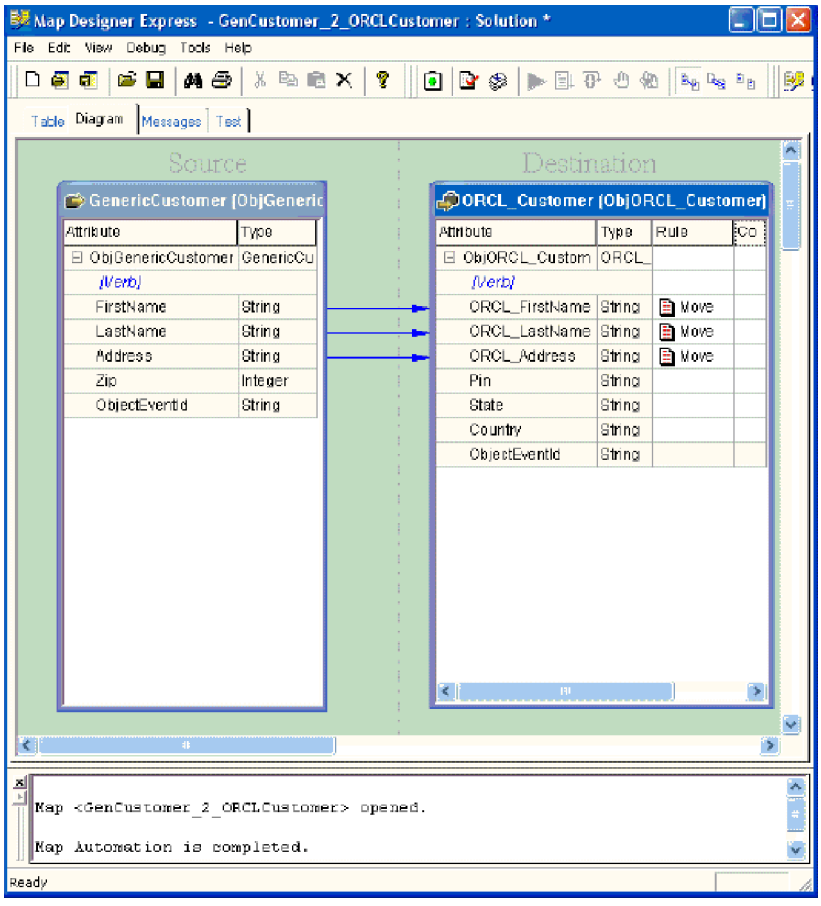

図 31. 自動マッピングでプレフィックスを追加した例

反転マップの自動作成

一般に、マップはペアで使用されます。マップが使用されるほとんどの場所では、反対方向にもマップが必要です。マップの反転を使用すれば、反転マップの作成に必要なステップが自動化されます。次の表は、Map Designer Express が現在サポートする標準変換規則 (現在のマップの列) と、マップの反転に現在含まれている変換規則 (反転マップの列) を示します。

表 20. 現在のマップから反転マップへの変換に使用される規則

現在のマップ	反転マップ
移動	移動
分割	結合
結合	分割
値を設定	マッピングなし
カスタム	マッピングなし
相互参照	マッピングなし
サブマップ	サブマップ (ある場合)

表 20 に示すように、反転マッピングには移動、分割、結合、およびサブマップ変換があります。値を設定、相互参照、およびカスタム変換規則は、反転マップの作成中はそのまま残ります。

制約事項: 結合から分割への反転マッピングを行う場合は、区切り文字を指定する必要があります。これに対し、分割から結合への反転マッピングでは、区切り文字はオプションです。

反転マップの自動生成手順

反転マップを自動的に作成するには、次のステップを実行します。

1. 反転マップを必要とするマップを開きます。
2. 「ツール」メニューから「マップの反転」を選択します。

   結果: 「別名保管」ダイアログが表示されます。
3. 反転マップの名前を入力して、「保管」をクリックします。

   結果: 現在開いているマップの反転マップが作成され、その反転マップが Map Designer Express の新規インスタンスとして開きます。

反転マッピングの例

次の例は、マップ反転シナリオ前とシナリオ後を示します。

図 32 は、反転マップを必要とするマップを示します。このマップでは、移動、カスタム、結合、分割、および値を設定の変換が使用されます。

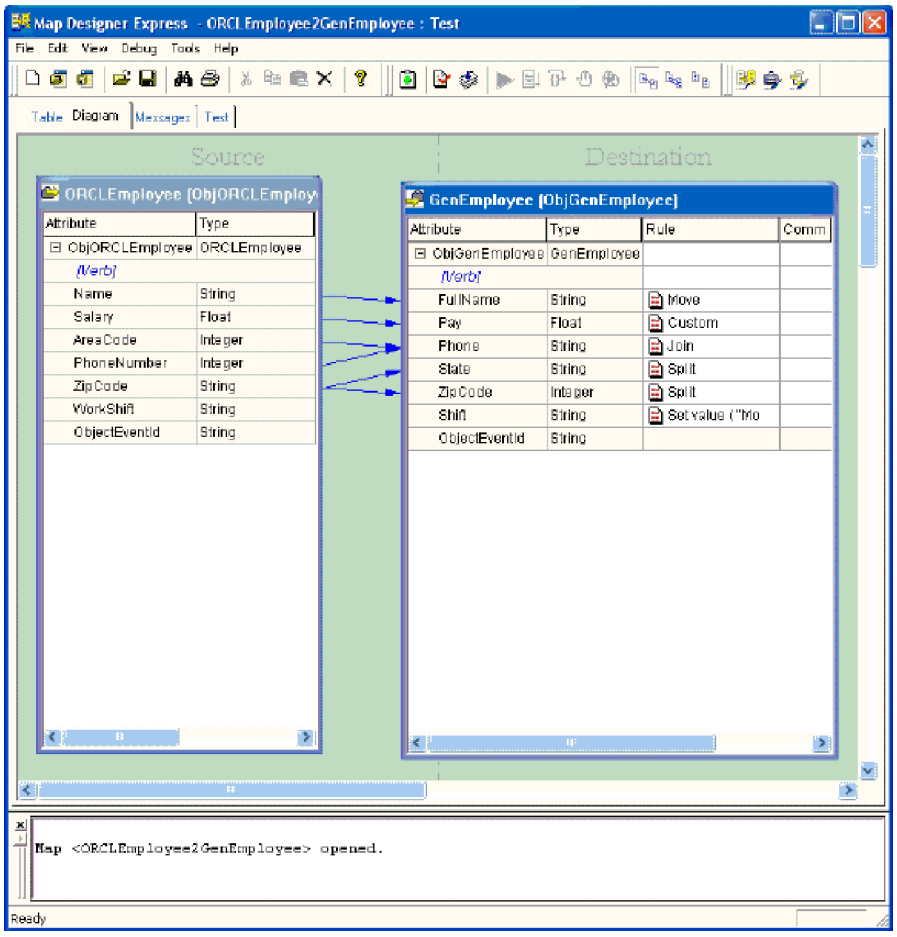

図 32. 反転マップを必要とするマップ

反転マップの自動作成のステップ (76 ページの『反転マップの自動生成手順』参照) を実行すると、次のマップが開きます。

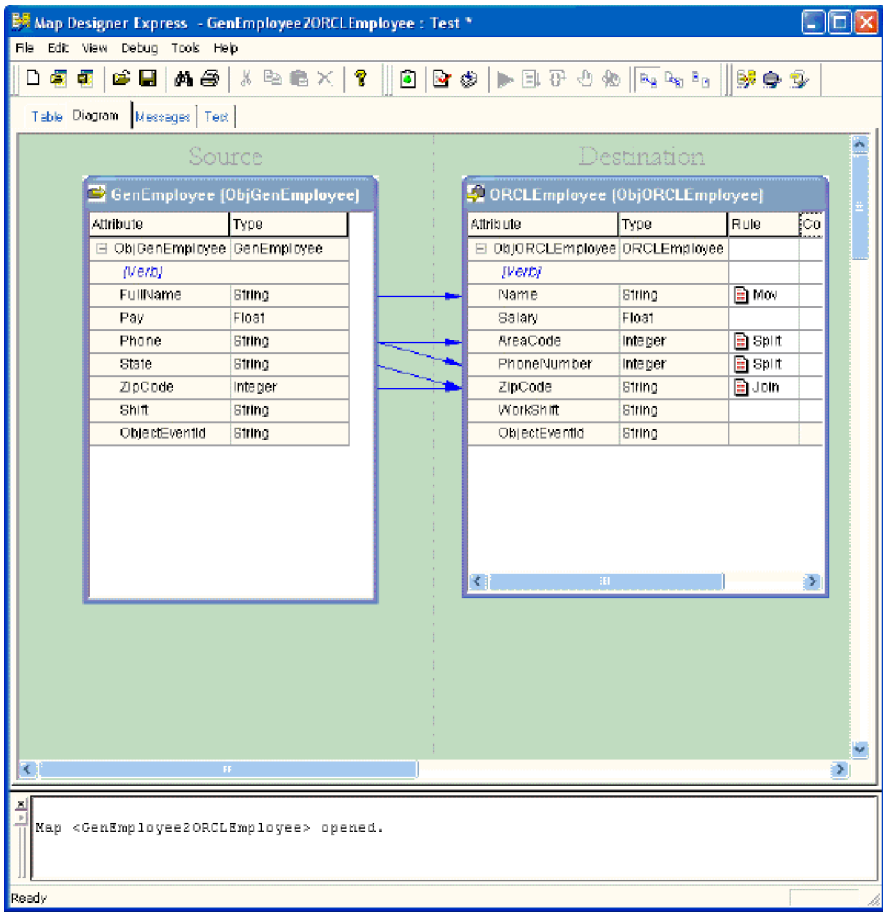

図 33. 反転の結果、自動作成されたマップ

図 33 にあるように、移動変換は、反転マップでも移動になります。分割および結合の変換は反転します。カスタム (Pay) および値を設定 (Shift) の変換はそのまま残ります。これらは、Activity Editor を使って手動で行う必要があります。反転方向で実行できない変換は、出力ウィンドウに警告としてリストされます。

Activity Editor の使用については、113 ページの『第 5 章 マップのカスタマイズ』を参照してください。

自動化の同義語の使用

基本のマッチング・プロセスを拡張する目的で、属性名に複数の同義語を作成できます。例えば、属性名を、一致する 1 つの名前と突き合わせるだけでなく、考えられるいくつかの同等な名前と突き合わせることができます。

例: ソース側に *CR* という属性名があるとします。この名前は、宛先側の以下の属性名と一致する可能性があります。

- Defect
- Change request

- Bug number
- Defect number
- CR

これらの同義語は、System Manager の「同義語」ウィンドウのプロジェクト・レベルで追加できます。ここでエントリーを編集し、さらにストリングをコンマで区切って追加すれば、マップの自動化に役立ちます。プロジェクト内のすべてのビジネス・オブジェクトに適用されるグローバル 同義語も作成できます。

System Manager でマップの自動化用に同義語を作成する手順については、「システム・インプリメンテーション・ガイド」を参照してください。

System Manager は、与えられた属性に関するすべての同義語を検索して、一致するものが検出されると自動マッピングを実行します。例えば、*Defect*、*Change request*、*Bug number*、および *CR* が「同義語」ウィンドウで同義語として追加済みであれば、ソース側の *CR* は、これらと一致します。これらのいずれかの語が検出されると、自動的にマッピングが実行されます。

マップの情報の検索

Map Designer Express の検索機能を使用し、次の検索を実行できます。

- 属性名または属性の変換コード内のテキストの検索。
- リンクされていない属性の検索。

マップの情報の検索手順

マップ内の情報を検索するには、次のステップを実行します。

1. 次のいずれかの方法で検索を開始します。
   - 「編集」メニューから「検索」を選択します。
   - キーボード・ショートカット `Ctrl+F` を使用します。
   - 標準ツールバーで「検索」ボタンをクリックする。

   結果: Map Designer Express で「検索」コントロール・ペインが表示されます (図 34 を参照)。

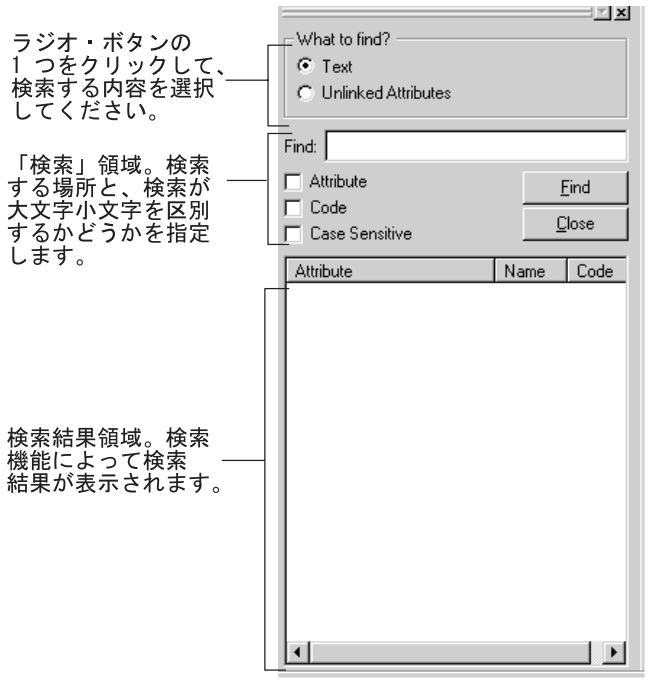

図 34. 「検索」コントロール・ペイン

2. 「検索」コントロール・ペインから「検索対象」領域のラジオ・ボタンの 1 つを選択し、実行する検索の種類を示します。
   - テキストを検索するには、次の操作を行う。
     a. 「テキスト」ラジオ・ボタンを選択する。
     b. 「検索」フィールドに検索するテキストを入力する。必要に応じて複数の単語やスペースを入力できます。
     c. 「検索」領域で 1 つ以上のオプションを選択し、テキストを検索する先を指定する。

        「属性」: 指定したテキストを属性名で検索する。

        「コード」: 属性の変換コードから指定したテキストを検索する。「属性」と「コード」はどちらか一方または両方を選択できます。

        「大文字と小文字を区別」: テキスト検索で大文字小文字を区別する。入力した大文字小文字と正確に一致するテキストのみを検索するには、「大文字と小文字を区別」を選択してください。

        制限: データ型およびコメントは検索できません。
     d. 「検索」をクリックして検索を開始する。
   - リンクされていない属性を検索するには、次の操作を行う。
     a. 「リンクされていない属性」ラジオ・ボタンを選択する。「検索」コントロール・ペインの「検索」領域は非アクティブになります。
     b. 「検索」をクリックして検索を開始する。

   結果: 検索結果が検索結果領域に表示されます。属性名をクリックすると、マップでその属性が自動的に選択されます。

3. 「検索」コントロール・ペインを閉じるには、「閉じる」をクリックします。

テキストの検索および置換

Map Designer Express の検索および置換機能を使用すると、変換規則のカスタマイズ済み Java コード内またはコメント内 (あるいはこれらの両方) で、指定したテキストを検索し、他の指定テキストと置換できます。

テキストの検索および置換手順

テキストを検索および置換するには、次の手順を実行します。

1. 次のいずれかの方法で検索および置換を開始します。
   - 「編集」メニューから「置換」を選択します。
   - キーボード・ショートカット `Ctrl+H` を使用します。

   結果: 「置換」ダイアログが表示されます。

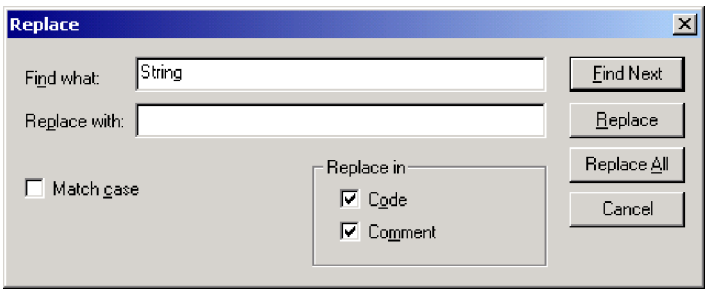

図 35. 「置換」ダイアログ

2. 「置換」ダイアログで、「検索対象」フィールドに検索対象のテキストを入力し、「置換後の文字列」フィールドには置換するテキストを入力します。必要に応じて、大文字小文字の区別を選択します。
3. コードかコメント、または両方を選択して、置換を実行する場所を指示します。
4. 「次を検索」をクリックして検索を開始します。

   結果: 以下のいずれかが表示されます。
   - コード内での置換を指定した場合、変換規則のカスタマイズ済み Java コードでテキストが見つかると場合、Activity Editor がそのカスタマイズ済み Java コードをクイック表示モードで表示します。
   - コメント内での置換を指定した場合、「テーブル」の表示がアクティブになり、「テーブル」の表示の「コメント」列にテキストが表示されます。
5. 「置換」をクリックして、一致したテキストを新規のテキストと置換します。

   ガイドライン: 「すべてを置換」をクリックすると、同様の一致テキストを、1 回のアクションで一括して置換できます。
6. 指定したテキストの検索および置換をインスタンスごとに続けるには、ステップ 4 および 5 を繰り返します。

マップの印刷

Map Designer Express ではマップを印刷できます。マップは、「テーブル」タブに表示されるマップと同様に、表形式で表されます。次のいずれかの方法でマップを印刷できます。

- 「ファイル」メニューから「印刷」を選択して、現在のマップを印刷する。
- Ctrl + P のキーボード・ショートカットを使用する。
- 標準ツールバーで「印刷」ボタンをクリックする。

また、Map Designer Express では、次の標準的な印刷タスクもサポートしています。

- 印刷プレビュー: 「ファイル」メニューから「印刷プレビュー」を選択すると、現行マップのページ・レイアウトをプレビューできる。
- 印刷設定
  - 「ファイル」メニューから「印刷設定」を選択すると、「印刷設定」ダイアログが表示される。「印刷設定」ダイアログでは、プリンター設定、用紙のサイズと方向などの情報を構成できる。
  - Ctrl + Shift + P のキーボード・ショートカットを使用する。

ガイドライン: Map Designer Express が印刷タスクまたは印刷プレビュー・タスクを実行するとき、「テーブル」タブの属性変換テーブルがコピーされます。印刷する前に、属性変換テーブルで個々の列の幅および個々の行の高さを調整し、マップ全体が 1 ページに収まるようにしたり、印刷結果をカスタマイズしたりできます。

オブジェクトの削除

このセクションでは、次のオブジェクトを削除する方法について説明します。

- 『マップ変換ステップの削除手順』
- 83 ページの『ビジネス・オブジェクトの削除手順』
- 84 ページの『マップの削除手順』

マップ変換ステップの削除手順

マップ変換ステップの削除は、次の 3 つの処理から構成されます。

- 変換コードの削除
- コメントの削除
- データ・フロー矢印の削除

このいずれかのマップ・タブから変換ステップを削除するには、次のステップを実行します。

- 「テーブル」タブから: 属性行を選択して削除します。左端の列 (「実行順序」の左側の列) をクリックして、次のいずれかのアクションを行ってください。
  - 右マウス・ボタンをクリックし、コンテキスト・メニューから「行を削除」を選択する。
  - 「編集」メニューから「現在の選択範囲を削除」を選択します。
  - Del のキーボード・ショートカットを使用する。

結果: マップを保管するときに、不完全な変換は自動的に削除されます。

- 「ダイアグラム」タブから: データ・フロー矢印を選択して、次のいずれかのアクションを行います。
  - 「編集」メニューから「現在の選択範囲を削除」を選択します。
  - Del のキーボード・ショートカットを使用する。
  - 右マウス・ボタンをクリックし、マップ・ワークスペースのコンテキスト・メニューから「削除」を選択する。

結果: 関連したデータ・フロー矢印を削除するかどうかを尋ねるダイアログが表示されます。「はい」をクリックすると、関連したコードを削除するかどうかを尋ねる 2 番目の確認メッセージが表示されます。「はい」をクリックします。3 つの項目がすべて削除されます。

ビジネス・オブジェクトの削除手順

マップからビジネス・オブジェクトを削除するには、次のステップを実行します。

1. 次のいずれかの方法で「ビジネス・オブジェクトを削除」ダイアログを表示します。
   - 「編集」メニューから「ビジネス・オブジェクトを削除」を選択します。
   - 「テーブル」タブから次のいずれかの処置を実行する。
     - ビジネス・オブジェクト・ペインの空の領域で右マウス・ボタンをクリックし、コンテキスト・メニューから「ビジネス・オブジェクトを削除」を選択する。
     - ビジネス・オブジェクト・ペインのビジネス・オブジェクトを右マウス・ボタンでクリック (セルの名前をクリック) し、「Delete <*BusObjName* >」を選択する (ここで、*BusObjName* は選択されたビジネス・オブジェクトの名前を示します)。

   結果: 「ビジネス・オブジェクトを削除」ダイアログが表示されます。

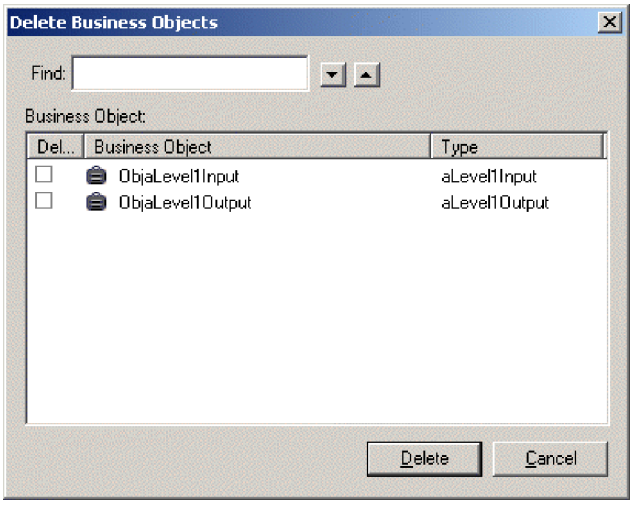

図 36. 「ビジネス・オブジェクトを削除」ダイアログ

2. 「ビジネス・オブジェクトを削除」ダイアログにより、マップから削除するビジネス・オブジェクトを指定します。「ビジネス・オブジェクトを削除」ダイアログには、次の機能があります。
   - ビジネス・オブジェクトを削除するには、次の操作を行う。
     - ビジネス・オブジェクト・リストでビジネス・オブジェクトを選択する。
     - 「削除」ボタンをクリックします。
   - 特定のビジネス・オブジェクトを見つけるには、「検索」フィールドにそのオブジェクトの名前を入力する。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。
   - ダイアログを閉じるには、「完了」をクリックする。

マップの削除手順

System Manager のプロジェクトからマップを削除するには、次のステップを実行します。

1. 「ファイル」メニューから「削除」を選択する。

   結果: 図 37 に示すように、「マップを削除」ダイアログが表示されます。

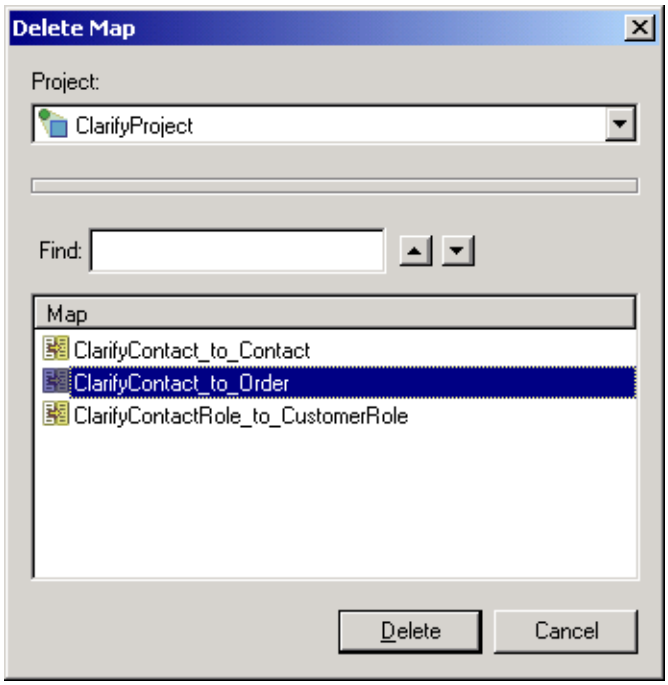

図 37. 「マップを削除」ダイアログ

注: マップが現在開いている場合は、このマップが自動的に閉じた後に「マップを削除」ダイアログが表示されます。「マップを削除: 削除前にマップを閉じる (Delete Map: close map before delete)」オプションを使用すれば、現在開いているマップを自動的に閉じるかどうかを指定できます。デフォルトでは、このオプションは使用可能です。このオプションを使用不可にした場合は、「マップを削除」ダイアログから現在開いているマップを選択すると、確認プロンプトが表示されます。このオプションの設定は、「設定」ダ

イアログの「一般」タブで変更できます。詳細については、 23 ページの『一般的な設定の指定』を参照してください。

2. プロジェクト名を入力します。
3. 削除するマップを選択します (複数も可)。

「マップを削除」ダイアログから、次の操作を実行できます。

- リストのマップ名をクリックして、単一のマップを選択する。
- Ctrl キーまたは Shift キーを押しながらマップ名をクリックして、複数のマップを選択する。
- 「検索」フィールドにビジネス・オブジェクトの名前を入力して、特定のビジネス・オブジェクトを探し出す。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。

4. 「削除」ボタンをクリックしてマップを削除します。

結果: 削除を確認するボックスが表示されます。

注: 「マップを削除: 常に警告メッセージを表示」オプションを使用すれば、マップの削除を確認するかどうかを指定できます。デフォルトでは、このオプションは使用可能です。このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、 23 ページの『一般的な設定の指定』を参照してください。

実行順序の使用

デフォルトでは、宛先属性が「テーブル」タブに表示される順序でマップは実行されます。変換がある宛先属性のみが実行されます。多くの場合、実行順序は、宛先属性が宛先ビジネス・オブジェクトで定義されている順序です。図 38 は A から B へのマップの実行順序を示しています。ここで、宛先属性は定義された順序で実行されます。

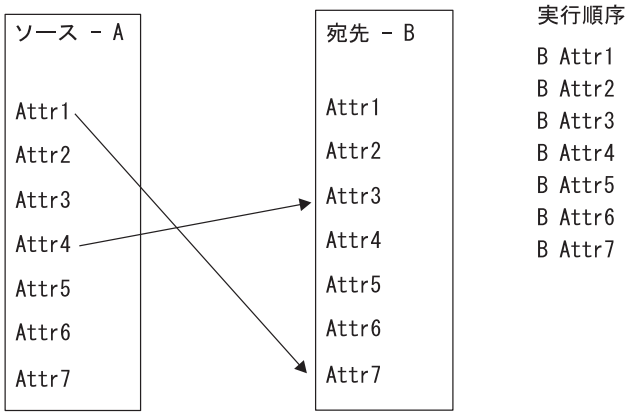

図 38. デフォルトの実行順序

注: 図 38 では、すべての宛先属性に変換コードがあることを想定しています。

しかし、属性には実行順序に依存するものもあります。特定の属性の変換コードをほかの属性の変換コードより前に確実に実行させるため、コードの実行順序を指定

できます。実行順序を変更してデータ・フローを指定できます。例えば、A から B へのマップで Attr7 は Attr3 の直後に実行する必要があるとします (言い換えれば、Attr7 は Attr4 の前に実行する必要があります)。図 39 は宛先ビジネス・オペレーションのシーケンス指定がシーケンスをどのように変えるかを示しています。

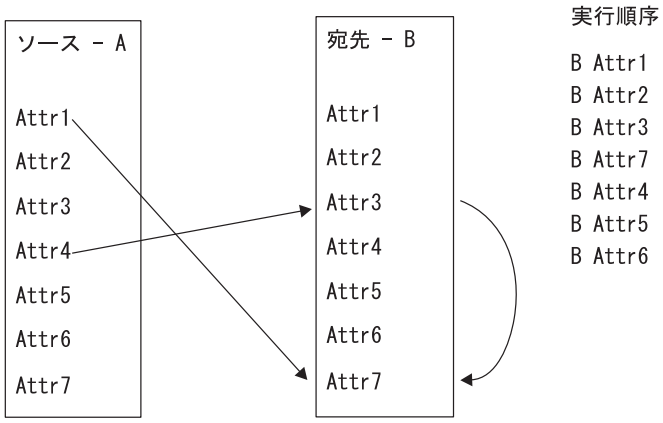

図 39. 実行順序の変更

Map Designer Express の「テーブル」タブから、デフォルト順序をオーバーライドする明示的な実行シーケンスを指定できます。「テーブル」タブで 2 つの宛先属性間の変換シーケンスを指定するには、実行順序を変更する宛先属性の「実行順序」フィールドをクリックし、必要な実行順序の値を入力します。

注: 「マップを定義: 実行順序を自動的に調整」オプションを使用すれば、この変更の影響を受ける属性の実行順序の再番号付けを自動的に行うかどうかを指定できます。デフォルトでは、このオプションは使用不可です。オプションを使用可能にすると、ほかの属性の実行順序は Map Designer Express により自動的に調整されます。このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。

デフォルトでは、変換が定義された順序で「テーブル」タブに属性が表示されます。これらのマップされた属性は、実行順序か属性名の順に表示できます。また、属性変換テーブルのほかの列の順に並べ替えることもできます。列の見出しをクリックするだけで、属性はその列の値順に配列されます。

要確認: 変換の行見出しをクリックし、新しい位置に変換をドラッグ・アンド・ドロップすると、変換規則の表示順序が変わります。しかし、この処置は実行順序には影響しません。

ポリモアフィック・マップの作成

ポリモアフィック・マッピングを使用すると、単一のソース・ビジネス・オブジェクトを多くの潜在的な宛先ビジネス・オブジェクトの 1 つにマップできます。この形式のマッピングを行うには、次の操作が必要です。

1. 発生しうる結果ごとに異なるマップ (1 つのソース・オブジェクトと 1 つの宛先オブジェクト) を作成します。

2. 単一のソース・ビジネス・オブジェクトと複数の宛先オブジェクトを持つメイン・ポリモアフィック・マップを作成します。
3. 各宛先ビジネス・オブジェクトの最初の属性内で、どの宛先ビジネス・オブジェクトを入力するかを指定する条件を検査します。条件が真である場合、望ましい結果を得るため、runMap() メソッドを使用して該当するマップを実行します。

例: 下に示すのは、メイン・ポリモアフィック・マップの宛先ビジネス・オブジェクトの 1 つにある最初の属性から取ったサンプル・コードです。この例で、ObjInput はソース・ビジネス・オブジェクトのインスタンス変数です。ObjOutput1 はこのコードが入った出力オブジェクトのインスタンス変数です。InputToOutput1 は ObjInput から ObjOutput1 へ実際のマッピングを行うサブマップです。この場合、マッピングの実行を指定する条件は、ソース・ビジネス・オブジェクト内の Attr1 属性の値に基づいています。当然ながら、ユーザーの実際の条件は異なります。

```
BusObj[] rSrcBO = new BusObj[1];
BusObj[] rDstBO = new BusObj[1];

rSrcBO[0] = ObjInput;
String Attr1Val = ObjInput.getString("Attr1");

if (Attr1Val.equals("Poly1"))
   {
   try
      {
      rDstBO = DtpMapService.runMap("InputToOutput1",
         DtpMapService.CWMAPTYPE,rSrcBO,cwExecCtx);

      ObjOutput1.setContent(rDstBO[0]);
      }

   catch (MapFailureException e)
      {
      e.toString();
      e.printStackTrace();
      raiseException(e);
      }

   catch (MapNotFoundException e)
      {
         raiseException("MapNotFoundException",
            "runMap did not find map");
      }

   catch (Exception e)
      {
      e.printStackTrace();
      }
   }
```

InterChange Server Express からのマップのインポートとエクスポート

repos_copy ユーティリティーで -e オプションを使用すると、リポジトリーで指定したマップ定義をロードしたりアンロードしたりできます。マップ・リポジトリー・ファイル は、repos_copy ユーティリティーがリポジトリーから .jar ファイルにマップ定義を抽出するときに作成するファイルです。このファイルには IBM WebSphere Business Integration Server Express 定義の .jar 形式でマップ定義が格納されます。

推奨: マップ・リポジトリー・ファイルには .jar ファイル拡張子を使用してください。

例: 次の repos_copy コマンドは、ClCwCustomer (ClarifyBusOrg から汎用 Customer への) マップ定義を、WebSphereICS という InterChange Server Express のリポジトリーからマップ・リポジトリー・ファイルにアンロード (エクスポート) します。

```
repos_copy -eMap:ClCwCustomer+BusObj:Customer+BusObj:Clarify_Customer
-oNM_ClCwCustomer.jar -sWebSphereICS -pnull -uadmin
```

次のものを含め、すべてのマップ定義ファイルが入った 1 つのリポジトリー・ファイルを作成できます。

- メイン・マップ定義
- サブマップ定義
- *両方の* 方向に対するファイル (適用できる場合)

例: ClarifyBusOrg/Customer マッピングのすべての関連マップ定義をマップ・リポジトリー・ファイルにコピーするには、次の repos_copy コマンドを使用します。

```
repos_copy -eMap:ClCwCustomer+Map:CwClCustomer
-oNM_ClCwCustomer_and_CwClCustomer.jar -sWebSphereICS -pnull -uadmin
```

複数のマップでサブマップを再使用する場合、メイン・テキスト・ファイルに入れずに、異なる repos_copy ファイルを作成してください。

また、repos_copy を使用して、マップ・リポジトリー・ファイルからリポジトリーにマップ定義をロード (インポート) することもできます。

例: 次の repos_copy コマンドは、ClCwCustomer マップ定義を WebSphereICS という InterChange Server Express のリポジトリーにロードします。

```
repos_copy -iNM_ClCwCustomer.jar -sWebSphereICS -uadmin -pnull
```

この repos_copy コマンドでは、マップ定義 ClCwCustomer と CwClCustomer が現在リポジトリーに*存在しない* ことを想定しています。存在する場合、このコマンドはこれらの新規マップ定義のロードに失敗します。repos_copy の -a オプションの 1 つを使用すると、重複するマップ定義を処理する方法を選択できます。

-ai	ロード中、重複したマップ定義はスキップします。
-ar	重複するマップ定義をマップ・リポジトリー・ファイルのマップ定義で上書きします。
-arp	重複するマップ定義をマップ・リポジトリー・ファイルのマップ定義で上書きするかどうか、ユーザーに対話的に照会します。

注: 実動モードでは、マップは自動的にコンパイルされます。

また、repos_copy を使用して、リポジトリーの関係定義のロードとアンロードを行うこともできます。詳細については、339 ページの『関係のロードとアンロード』を参照してください。

第 4 章 マップのコンパイルとテスト

この章では、Map Designer Express を使用してマップの検証、コンパイル、およびテストを行う方法について説明します。

この章の内容は次のとおりです。

- 『変換コードの検査』
- 90 ページの『マップの検証』
- 91 ページの『マップのコンパイル』
- 93 ページの『マップ・セットのコンパイル』
- 94 ページの『マップのテスト』
- 103 ページの『拡張デバッグの実行』
- 103 ページの『関係を含むマップのテスト』
- 109 ページの『マップのデバッグ』

変換コードの検査

宛先属性に関連した変換コードの作成が終了したら、コードに対して限定的な構文チェックを実行できます。検査しながら作業を進めることで、マップ開発過程の最後に必要なデバッグ時間を短縮できます。属性コードの検査には、対応しない区切り文字を検索する技法を使用できます。

注: コンパイル・エラーが発生し、エラー・メッセージからすぐに原因を判別できない場合にも、この技法が役立ちます。

対応しない区切り文字の検索

Map Designer Express は、対応しない区切り文字の検査機能を提供しています。この機能は、非常に検出が難しいプログラム・エラーの 1 つを解決するために役立ちます。この機能では、属性の変換コードの中で対応しない区切り文字を検査します。Map Designer Express では、( )、[ ]、{ }、“ ”、および ‘ ’ の対のトークンを検査します。

対応しない区切り文字の検索手順

属性の変換コードに対してこの構文チェックを実行するには、次の操作を行います。

1. Activity Editor を Java モードで起動します。

   Activity Editor の表示方法については、114 ページの『Activity Editor の始動』を参照してください。

2. Activity Editor で「対応しない区切り文字を検査」オプションを使用します。右マウス・ボタンをクリックし、コンテキスト・メニューから「対応しない区切り文字を検査」を選択します。

注: 区切り文字の 1 つが対にならずに片方しか存在しない場合、出力ウィンドウにメッセージが表示され、エラーが解決できなかった行番号が示されます。この行番号は、区切り文字が実際に欠落している行であるとは限りません。

3. 対応しない区切り文字がある位置に移動するには、ウィンドウ下部に表示された行番号に注意してください。

   ヒント: この行に移動するには、Activity Editor の「編集」メニューまたはコンテキスト・メニューから「行に移動」オプションを使用します。行番号を入力して、問題が発生した行に移動します。

注: ストリング・リテラルの一方の端に対応しない引用符があるために問題が発生した場合、ストリングは所定のピンク色で表示されません。欠落している引用符を追加すると、ストリング全体がピンク色になります。

マップの検証

Map Designer Express の検証プロセスでは、次の検査を行うことで、マップのデータ・フローが正確であるかどうかを検証します。

- マップには未完了の変換ステップがないことを確認する。
- ビジネス・オブジェクト配列のインデックスがゼロ (0) から始まり、正しく連続していることを確認する。
- 変換ステップが `ObjectEventId` 属性にマップする場合、警告を行う。
- 変換を検証する。
  - 実行順序が正しいこと、つまり実行順序が一意であり、正であり、連続していることを確認する。
  - 属性が相互に循環依存関係になっていないかどうかを確認する。循環変換が見つかった場合、Map Designer Express は出力ウィンドウに循環規則を表示する。
  - 変換情報を検査する。

    「移動」変換: 1 つのソース属性のみ関係する。

    「結合」変換: 複数のソース属性が関係する。

    「分割」変換: 1 つのソース属性のみ関係する。分割インデックスは 0 以上になる。分割区切り文字は空ではない。

    「値を設定」変換: ソース属性は関係しない。値が指定されている。

    「サブマップ」変換: 少なくとも 1 つのソース属性が関係する。サブマップ名が指定されている。

    「相互参照」変換: 1 つのソース属性のみ関係する。

Map Designer Express でマップを保管すると、マップは自動的に検証されます。また、次のいずれかのアクションを行うことで、マップの検証を選択することもできます。

- 「ファイル」メニューから「マップを検証」を選択する。

- Designer ツールバーで、「検証」ボタンをクリックする。

この時点で、「設定」ダイアログの「検証」タブにオプションを指定していた場合、特定の条件がマップされていないと、Map Designer Express は警告を出します。

属性間の依存関係の設定の詳細については、85 ページの『実行順序の使用』を参照してください。

マップのコンパイル

Map Designer Express はマップをコンパイルすると、マップの変換用 Java コードを保持する .java ファイルから .class ファイルを生成します。この .java ファイルはプロジェクトのマップ定義の一部として保管された変換コードから生成されます。

要確認: マップをコンパイルできるようにするには、Java コンパイラー (javac) がシステムに存在し、そのパスが PATH システム変数に設定されている必要があります。詳細については、12 ページの『開発環境の設定』を参照してください。

Map Designer Express からのマップのコンパイル手順

Map Designer Express 内から、次のいくつかの方法でマップのコンパイルを開始できます。

- *現在の* マップをコンパイルするには、次のいずれかを行います。
  - 「ファイル」メニューから「コンパイル」を選択する。
  - F7 のキーボード・ショートカットを使用する。
  - Designer ツールバーで、「コンパイル」ボタンをクリックする。
- このマップが使用している*現行*マップおよびすべてのサブマップ をコンパイルする。
  - 「ファイル」メニューから「サブマップでコンパイル」を選択する。
- System Manager に定義されているすべて のマップまたはマップのサブセットをコンパイルするには、以下のいずれかを行います。
  - 「ファイル」メニューから「すべてコンパイル」を選択する。
  - Ctrl + F7 のキーボード・ショートカットを使用する。

  詳細については、93 ページの『マップ・セットのコンパイル』を参照してください。

デフォルトでは、Map Designer Express はマップをプロジェクトに保管した後にマップをコンパイルし、.java ファイルと .class ファイルに Java コードを生成します。メッセージ・ファイルが必要な場合は、メッセージ・ファイルも生成されます。

注: マップのコンパイル前に自動的にマップをプロジェクトに保管するかどうかを指定するには、**「マップをコンパイル: コンパイル前にマップを保管」**オプションを使用します。デフォルトでは、このオプションは使用可能です。このオ

プションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。

コンパイルするには、Map Designer Express はマップの Java ソース・コード (.java ファイル) に対して Java コンパイラーを呼び出します。その後の動作は、コンパイルが成功するかどうかによって異なります。

System Manager からのマップのコンパイル手順

また、System Manager もマップをコンパイルするいくつかの方法を提供しています。

- 単一のマップをコンパイルするには、次のいずれかを行います。
  - 必要なマップを強調表示し、「コンポーネント」メニューから「コンパイル」を選択する。
  - 必要なマップを右マウス・ボタンでクリックし、コンテキスト・メニューから「コンパイル」を選択する。
- マップとそのサブマップをコンパイルするには、次のようにします。
  - 必要なマップを右マウス・ボタンでクリックし、コンテキスト・メニューから「サブマップでコンパイル」を選択する。
- プロジェクトに定義されているすべての マップをコンパイルするには、次のようにします。
  - マップ・フォルダーを強調表示にし、「コンポーネント」メニューから「すべてコンパイル」を選択する。

  注: すべてのマップをコンパイルするには、マップ・フォルダーをクリックし、コンテキスト・メニューから「すべてコンパイル」を選択してプロジェクトのマップ・フォルダーを選択する必要があります。

マップのコンパイルの成功

マップのコンパイルが成功するとき、Map Designer Express は次の手順に従います。

- Java コードを .java ファイルにコンパイルする。
- 各マップ・タブの下部にある出力ウィンドウに、コンパイル中にエラーがなかったことを示す次のメッセージを表示する。

```
Compilation is successful.
```

マップのコンパイルの失敗

コンパイル中にエラーが発生すると、Map Designer Express はエラー・メッセージを生成し、画面下部の出力ウィンドウに表示します。出力ウィンドウがまだ開いていない場合、Map Designer Express ではマップ・タブの下部に出力ウィンドウを開き、コンパイル・エラー・メッセージを表示します。

コンパイル・エラーが発生すると、出力ウィンドウにエラー・メッセージおよび問題となる属性名と行番号が青色で表示されます。ハイパーリンクをクリックすると、Activity Editor の Java 表示に問題箇所が表示されます。

ヒント:「表示」メニューから「出力をクリア」を選択すると、出力ウィンドウからメッセージをクリアできます。

エラーには、検出が簡単なものもあり、難しいものもあります。

マップ・セットのコンパイル

「ファイル」メニューの「すべてコンパイル」オプションを使用すると、System Manager のすべてのマップまたはマップのサブセットをコンパイルできます。

マップ・セットのコンパイル手順

マップ・セットをコンパイルするには、次の手順を実行します。

1. 「ファイル」メニューから「すべてコンパイル」を選択する。

   結果: 「すべてのマップをコンパイル」ウィンドウが表示されます。
2. マップ・コンパイル用のプロジェクトを選択する。
3. コンパイルするマップを選択します。

   ガイドライン: ルートにあるチェック・ボックスを選択すると、すべての子チェック・ボックスには自動的にチェックマークが付きます。したがって、プロジェクトを選択すると、そのプロジェクトのすべてのマップが選択されます。マップのサブセットのみ選択するには、該当するコンパイル・チェック・ボックスを選択解除します。

図 40 に、「すべてのマップをコンパイル」ウィンドウを示します。

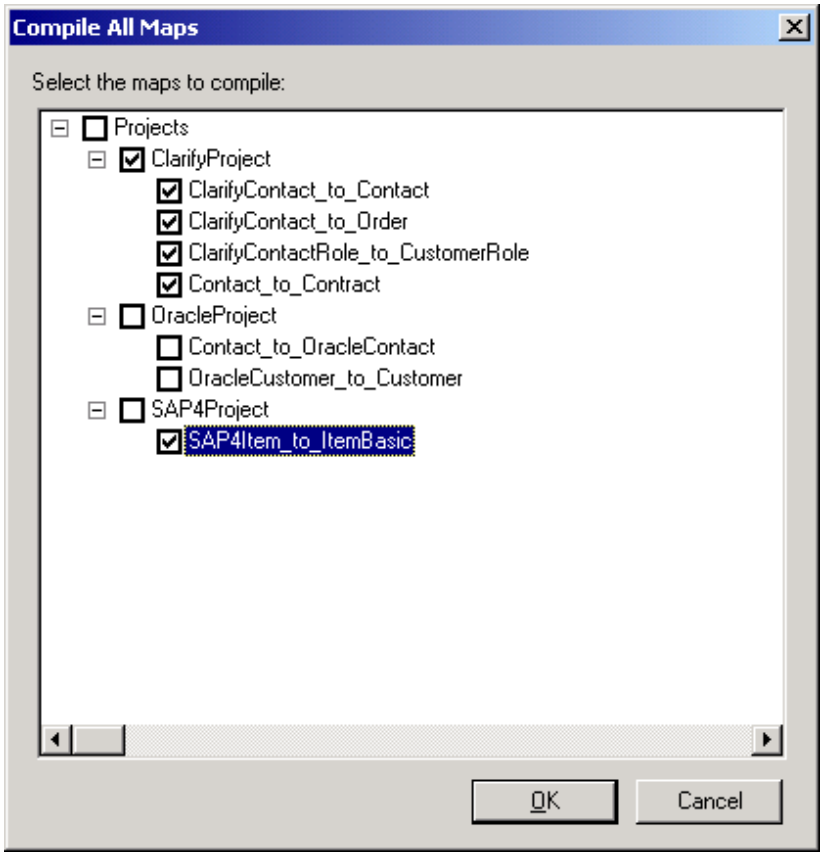

図 40. 「すべてのマップをコンパイル」ウィンドウ

結果: 出力ウィンドウに各マップのコンパイルの成功または失敗が表示されます。表示されるコンパイル状況メッセージを増やすため、コンパイル・プロセスが開始する前に出力ウィンドウのサイズを大きくできます。

マップのテスト

ソース・ビジネス・オブジェクトにサンプル・データを指定し、マップのテスト実行を行って、マップの変換ステップをテストできます。*テスト実行* はコネクターにより送信されたイベントまたはアクセス・クライアントにより送信された呼び出しに関係しないマップの実行です。マップは Map Designer Express 内で実行されます。マップをテストし、テスト結果を表示するため、Map Designer Express ウィンドウには「テスト」タブが異なるタブとして提供されています。

注: デバッグのためにマップがテスト環境から選択されると、テスト環境では Map Designer Express を起動し、テスト中のマップへの入力ビジネス・オブジェクトを Map Designer Express に与えます。

このセクションでは、以下の主要ステップを使用して、テスト実行をセットアップし、実行する方法について説明します。

- 95 ページの『テスト実行の準備手順』

- 『テスト・データの作成』
- 98 ページの『ブレークポイントの設定』
- 101 ページの『テスト・マップの実行』
- 102 ページの『テスト実行結果の表示』
- 102 ページの『マップの変更と再実行手順』

ヒント: 上記と異なるテスト方法は、ここで詳しくは説明しませんが、マップにブレークポイントを設定し、コネクターからトリガー・イベントを送って、マップを実行させる方法です。

テスト実行の準備手順

テストを実行する前に、次の手順を実行します。

1. マップを開いてプロジェクトからデバッグします。
2. マップを最後に変更した後にコンパイルしていない 場合、「ファイル」メニューから「コンパイル」を選択してマップをコンパイルします。詳細については、91 ページの『マップのコンパイル』を参照してください。
3. Map Designer Express の「テスト」タブが現在タブ・ウィンドウに表示されていない 場合、「テスト」タブを選択します。

テスト・データの作成

マップをテストするごとに、ソース・ビジネス・オブジェクトにデータをロードする必要があります。それには、「テスト」タブの「ソース・テスト・データ」ペインを使用します (図 41 を参照)。「ソース・テスト・データ」ペインでは、次のテスト情報を指定できます。

- 呼び出しのコンテキスト: マップを実行するマップの実行コンテキストを示す。
- 汎用ビジネス・オブジェクト: 一致関係の SERVICE_CALL_RESPONSE 呼び出しコンテキストをテストするときに汎用ビジネス・オブジェクトのテスト・データを提供する。
- テスト・データ: ソース・ビジネス・オブジェクトの属性のデータ。

要確認: 呼び出しのコンテキストと汎用ビジネス・オブジェクトは、マップ内の関係をテストする場合にのみ 必要です。詳細については、103 ページの『関係を含むマップのテスト』を参照してください。

最初のマップ・テスト

はじめてマップをテストするときは、「ソース・テスト・データ」ペインに属性の値を手入力する必要があります。

次のセクションでは、このデータを入力する方法について説明します。

- 『ソース・ビジネス・オブジェクトのテスト・データの作成に関するガイドライン』
- 97 ページの『子ビジネス・オブジェクトのテスト・データ作成手順』

ソース・ビジネス・オブジェクトのテスト・データの作成に関するガイドライン:
はじめてソース・ビジネス・オブジェクトのデータを作成するときは、次の規則に従ってください。

- 動詞を設定するには、動詞行の動詞コンボ・ボックスから動詞を選択する。
- ソース属性に値を割り当てるには、属性の「値」列に値を入力する。すべての属性に値を指定する必要はない。
- 関係属性に値を割り当てるには、「値」列に適切な値を指定する。また、正しい呼び出しのコンテキストも指定する。詳細については、103 ページの『関係を含むマップのテスト』を参照してください。
- 子ビジネス・オブジェクトに値を割り当てるには、子オブジェクトを右マウス・ボタンでクリックし、コンテキスト・メニューから「インスタンスを追加」を選択する。詳細については、97 ページの『子ビジネス・オブジェクトのテスト・データ作成手順』を参照してください。
- ソース・ビジネス・オブジェクトにデフォルト値を割り当てるには、ソース・ビジネス・オブジェクトを選択し、コンテキスト・メニューから「リセット」を選択する。
- 関係をテストする場合、関係に参加するソース親オブジェクトとすべての子オブジェクトの `ObjectEventId` を設定する。
- 将来のテスト実行で使用するために入力した値を保管するには、ビジネス・オブジェクト (`.bo`) ファイルを作成します。それには、ソース・ビジネス・オブジェクトを選択し、次のいずれかのアクションを行います。
  - 「ソース・テスト・データ」ペインで「保管」ボタンをクリックする。
  - コンテキスト・メニューから「保管」を選択する。入力を求められたら、値の保管先ファイル名を入力する。

    結果: 次回このマップをテストするときは、「ロード」ボタンをクリックすると、ビジネス・オブジェクト・ファイルから属性が自動的に入力されます。

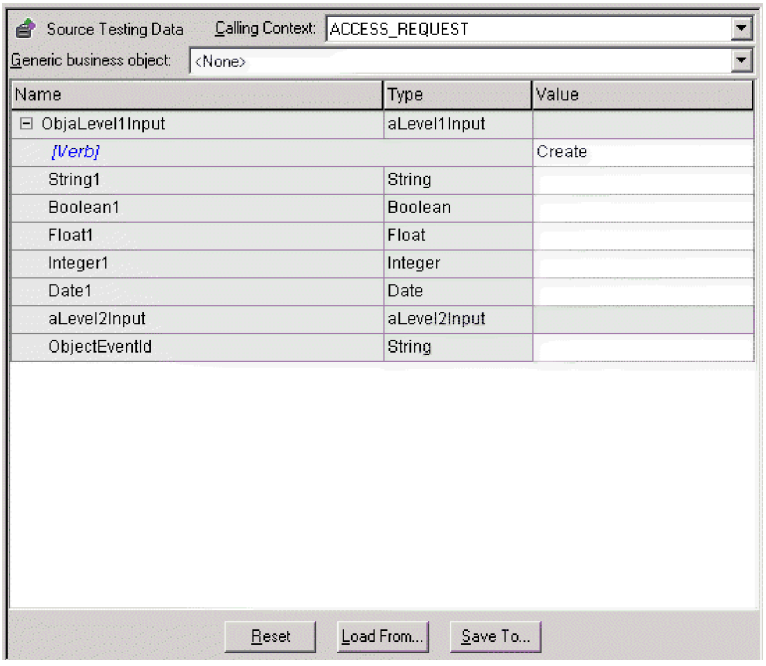

図 41. 「テスト」タブの「ソース・テスト・データ」ペイン

子ビジネス・オブジェクトのテスト・データ作成手順: ソース・ビジネス・オブジェクトに子ビジネス・オブジェクトがあり、子属性のテスト・データを指定する場合、必要な各子オブジェクトのインスタンスを最初に作成する必要があります。それには以下の手順を実行します。

1. 子ビジネス・オブジェクト名を右マウス・ボタンでクリックし、コンテキスト・メニューから「インスタンスを追加」を選択します。オブジェクトを展開すると、Map Designer Express が作成したインスタンスが表示されます。

   ガイドライン: 追加した最初のインスタンスのインデックス番号はゼロです。インスタンスには必要な数を設定できます (子属性に複数のカーディナリティーがある場合に限る)。
2. インスタンス・インデックス番号の横にある正記号 (+) をクリックし、子ビジネス・オブジェクトを展開します。

   結果: オブジェクトを展開すると、このインスタンスの子属性が表示されます。
3. 子ビジネス・オブジェクト・インスタンスのデータを作成するときは、次のガイドラインに従います。
   - 子ビジネス・オブジェクトの動詞を設定するには、動詞行の動詞コンボ・ボックスから動詞を選択する。
   - 子属性の値を指定するには、その属性を指定し、「値」列に値を入力する。
   - 属性名の後に (N) が付いている場合、その属性には複数カーディナリティーの子ビジネス・オブジェクトが含まれており、さらにインスタンスを追加できる。

     配列の最後に子ビジネス・オブジェクトを追加するには、最後のインデックスを右マウス・ボタンでクリックし、コンテキスト・メニューから「インスタンスを追加」を選択します。
   - 必要な数のインスタンスの値を変更する。次のようにインスタンスの追加と削除を行う。
     - インスタンスを追加するには、子インスタンス名を右マウス・ボタンでクリックし、「インスタンスを追加」を選択する。
     - インスタンスを削除するには、削除する子インスタンスのインスタンス名を右マウス・ボタンでクリックし、「インスタンスを削除」を選択する。
     - すべての インスタンスを削除するには、子インスタンス名を右マウス・ボタンでクリックし、「すべてのインスタンスを削除」を選択する。子ビジネス・オブジェクトに複数カーディナリティーがある場合にのみ、このオプションは使用可能になります。

2 回目以降のマップ・テスト

2 回目以降のテスト実行では、Map Designer Express は前に指定されたテスト・データを再利用します。このデータに対して、次のいずれかの処置を実行することができます。

- すべてのテスト・データをそのままにする。
- 「値」列の該当する項目を変更して、個々の属性の値を変更する。

ヒント: データを変更した場合、ビジネス・オブジェクト (.bo) ファイルを必ず再保管すること。

- ビジネス・オブジェクト (.bo) ファイルから値の集合をロードする。

  ビジネス・オブジェクト・ファイルから属性値をロードするには、ソース・ビジネス・オブジェクトを選択し、次のいずれかのアクションを行います。
  - 「ソース・テスト・データ」ペインで「ロード」ボタンをクリックする。
  - コンテキスト・メニューから「ロード」を選択する。

  入力を求められたら、ロードするビジネス・オブジェクト・ファイルの名前を入力します。
- ソース・ビジネス・オブジェクトを選択し、コンテキスト・メニューから「リセット」オプションを選択して、すべてのソース宛先値を定義済みデフォルト値に戻す。

ブレークポイントの設定

ブレークポイントを設定すると、マップの実行はブレークポイントが設定された宛先属性の変換の直前で一時停止します。ブレークポイントを使用して、マップの実行をステップスルーし、個々の操作のシーケンスと結果を検査できます。ブレークポイントは必要な数を設定できます。

ガイドライン: ブレークポイントはマップの定義の一部にはなっていません。Map Designer Express でマップを開いた後、および (「デバッグ」>「テスト実行」、または「デバッグ」>「拡張」>「起動」のどちらかを使用して) マップをデバッグするときに、マップ上にブレークポイントを設定してください。Map Designer Express からマップをデバッグしない場合、マップはブレークポイントによる影響を受けません。

注: 変換が定義されている宛先属性にのみブレークポイントを設定できます。

ブレークポイントの設定手順

ブレークポイントを設定するには、次のステップを実行します。

1. 次のメソッドのいずれかを行います。
   - 「宛先テスト・データ」ペインの宛先属性を右マウス・ボタンでクリックし、コンテキスト・メニューから「ブレークポイントを設定」を選択する。宛先ソース属性が展開していない場合、次のどちらかのコマンドを使用して展開できます。
     - 宛先ビジネス・オブジェクトの横にある正記号 (+) をクリックする。
     - 宛先ビジネス・オブジェクトを選択し、コンテキスト・メニューから「展開」を選択する。

       注: 宛先ビジネス・オブジェクトのコンテキスト・メニューには、「縮小」オプションもあります。
   - 「デバッグ」メニューから「ブレークポイントの切り替え」を選択する。
   - F9 のキーボード・ショートカットを使用する。
   - Designer ツールバーで、「ブレークポイントの切り替え」ボタンをクリックする。

注: 「ブレークポイントの切り替え」オプションは、ブレークポイント定義のオンとオフを切り替えます。ブレークポイントが現在設定されていない 場合、「ブレークポイントの切り替え」により設定されます。ブレークポイントが現在設定されている場合、「ブレークポイントの切り替え」により設定が解除されます。

結果: 図 42 に示すように、ブレークポイントが設定されている宛先属性の横には暗色の円が表示されます。

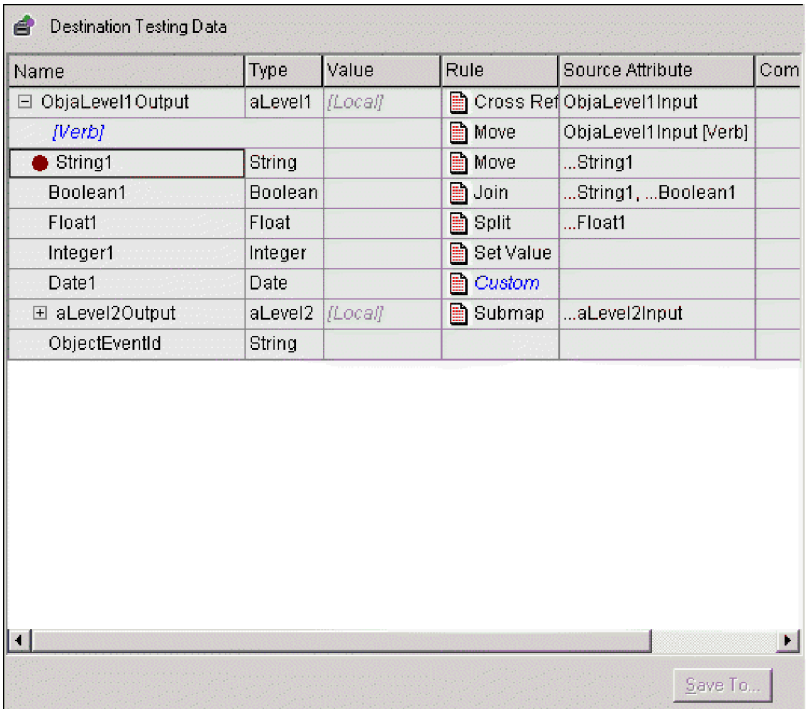

図 42. 設定されたブレークポイント

ブレークポイントを設定すると、マップ・インスタンスの実行はこのブレークポイントで一時停止し、マップの現在の状況を調べることができます。少なくとも 1 つのブレークポイントを指定しない限り、マップは最後まで実行され、次のメッセージを出して終了します。

```
Test run finished
```

規則: ブレークポイントを設定した宛先属性に関連したソース・データには常に値を指定する必要があります。値を指定しないと、変換規則は正常に実行され、ブレークポイントは正常に実行されますが、通常、宛先値は空になります (変換規則の定義に従います)。詳細については、95 ページの『テスト・データの作成』を参照してください。

マップのすべてのブレークポイントを表示するには、「デバッグ」メニューから「ブレークポイント」を選択します。

結果: 「ブレークポイント」ダイアログが表示されます (図 43 を参照)。

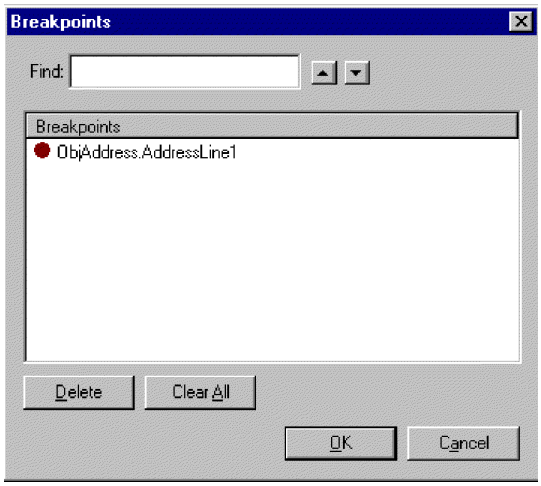

図 43. 「テスト」タブの「ブレークポイント」ダイアログ

2. 「ブレークポイント」ダイアログから、次のいずれかのアクションを実行できます。
   - ブレークポイントが設定されている宛先属性の検索: ブレークポイント名をダブルクリックする。

     ヒント: 特定のブレークポイントを探し出すには、「検索」フィールドにその名前を入力します。上下矢印を使用すると、ビジネス・オブジェクト・リストの中をスクロールします。「宛先テスト・データ」ペインで、宛先属性が強調表示されます。
   - ブレークポイントの削除: 「ブレークポイント」領域で削除するブレークポイントを選択し、「削除」ボタンをクリックする。

     また、次のいずれかのアクションを行ってブレークポイントを削除することもできます。
     - 「宛先テスト・データ」ペインの宛先属性を右マウス・ボタンでクリックし、コンテキスト・メニューから「ブレークポイントをクリア」を選択する。
     - 既存のブレークポイントに対して「ブレークポイントの切り替え」オプションのいずれかのコマンドを使用する。詳細については、98 ページの『ブレークポイントの設定』を参照してください。
   - 「ブレークポイント」領域に表示されたすべてのブレークポイントのクリア: 「すべてクリア」ボタンをクリックする。

     また、次のいずれかのアクションを行って、すべてのブレークポイントをクリアすることもできます。
     - 「デバッグ」メニューから「すべてのブレークポイントをクリア」を選択する。
     - Designer ツールバーで、「すべてのブレークポイントをクリア」ボタンをクリックする。

テスト・マップの実行

ソース・テスト・データを入力し、必要なブレークポイントを設定したら、マップ・テストの準備が整いました。マップ・テストの実行は、次の手順から構成されます。

1. 『テスト実行の開始手順』
2. 『ブレークポイント処理の手順』 (ブレークポイントが設定されている場合)

テスト実行の開始手順

テスト実行を開始するには、次の手順を実行します。

1. 次のいずれかのアクションを行います。
   - 「デバッグ」メニューから「テスト実行」を選択する。
   - Designer ツールバーで、「テスト実行」ボタンをクリックする。

   結果: 「Connect to IBM WebSphere Business Integration Server Express」ダイアログ・ボックスが表示され、ここでテスト用のサーバーに接続できます。
2. ダイアログで、サーバー名、ユーザー名、およびパスワードを入力します。
3. テスト実行のため、マップと従属ビジネス・オブジェクトを配置するかどうかを指定します。

   ガイドライン: テスト用に最小限のビジネス・オブジェクト・セットをサーバーに配置すると、デバッグの初期化時間を減らすことができます。

結果: マップの実行が開始します。Map Designer Express の出力ウィンドウに次のメッセージが表示されます。

```
Starting test run...
```

ブレークポイント処理の手順

ブレークポイントを設定した宛先属性に到達すると、マップの実行は一時停止します。ブレークポイントに到達すると、Map Designer Express は次のアクションを行います。

1. ブレークポイントが設定された宛先属性を強調表示にし、その横に暗色の円と黄色の矢印を表示します。
2. 出力ウィンドウに次のメッセージを表示します。

   ```
   Test Run stopped at attribute AttrName (next transformation > "Rule").
   ```

   ヒント: マップの実行の一時停止中、「宛先テスト・データ」ペインの「値」列を調べ、それまでに処理された宛先属性の値を調べることができます。
3. ユーザーが次のいずれかのアクションを行った場合、ブレークポイントを処理し、マップの実行を続行します。
   - 次のブレークポイントかマップの最後 (どちらか先に現れたもの) まで進む。

     マップの実行を続行するには、次のいずれかのアクションを行います。
     - 「デバッグ」メニューから「続行」を選択する。
     - F8 のキーボード・ショートカットを使用する。
     - Designer ツールバーで、「続行」ボタンをクリックする。
   - この宛先属性を実行し、次の属性を実行する前に停止する。

さらに 1 ステップだけマップの実行を続行するには、次のいずれかのアクションを行います。

– 「デバッグ」メニューから「ステップオーバー」を選択する。

ヒント: 属性ごとにコード実行を監視するときは、このオプションを選択すること。

– F10 のキーボード・ショートカットを使用する。

– Designer ツールバーで、「ステップオーバー」ボタンをクリックする。

結果: テスト実行がランタイム・エラーなしに終了すると、出力ウィンドウに次のメッセージが表示されます。

```
Test run finished.
```

テスト実行結果の表示

テスト実行の結果は、「宛先テスト・データ」ペインにある宛先ビジネス・オブジェクトに表示されます。マップ変換による値はこのテーブルの「値」列に表示されます。テスト実行結果は、次のいずれかの方法で表示できます。

- 『プロセスの監視』
- 『実行後の結果の表示』

プロセスの監視

データとブレークポイントが設定されているテスト実行の間、宛先ビジネス・オブジェクトに値が入力される過程を監視できます。処理が進むにつれて、値が「宛先テスト・データ」ペインの「値」列に表示されます。マップの実行がブレークポイントで一時停止すると、その属性より実行順序が*前である* すべての宛先属性の値が表示されています。

変換が実行される過程を表示するには、次の操作を行います。

- 2 番目の宛先属性にブレークポイントを設定し、「ステップオーバー」オプションを設定してマップの実行をステップスルーする。マップは読み取り専用になります。

実行後の結果の表示

マップの実行が終わってからテスト実行結果を表示するには、「宛先テスト・データ」ペインで宛先ビジネス・オブジェクトを調べてください。

テスト結果を保管するには、次の操作を行います。

- 宛先ビジネス・オブジェクトを強調表示にし、コンテキスト・メニューから「保管」を選択する。

結果: 宛先属性の値はビジネス・オブジェクト (.bo) ファイルに保管されます。

マップの変更と再実行手順

マップをテストしていると、マップの変更が必要になることがあります。マップを編集し、その後テストを続行するには、次の手順を実行します。

1. 「テーブル」タブまたは「ダイアグラム」タブに切り替え、マップ変換を表示します。

2. マップを編集し、エラーを訂正します。
3. マップを再コンパイルします。
4. 「テスト」タブに切り替えて、テスト・プロセスを続行します。
5. 新規テスト実行を開始します。

要確認:

1. 成功か失敗かにかかわらず、必ずテスト実行を完了してからマップを再コンパイルしてください。
2. マップを変更する場合は、変更後そのマップを配置して、変更内容がサーバーに反映されるようにしてください。

拡張デバッグの実行

ローカル・プロジェクトに保管されているマップをデバッグするだけでなく、サーバーにあるマップを直接デバッグすることもできます。これを行うには、以下の手順を実行します。

1. 「デバッグ」>「拡張」>「起動」を選択します。

   結果: 「Connect to WebSphere Business Integration Server Express」ダイアログが表示されます。
2. サーバー名、ユーザー名、およびパスワードを入力し、「接続」をクリックします。

   結果: そのサーバーの新規マップのリストが表示されます。
3. 起動するマップを選択します。

   結果: マップは読み取り専用モードで開きます。
4. 特定の変換規則でサーバーがマップの実行を一時停止するように、マップにブレークポイントを設定します。

   結果: サーバーでブレークポイントに到達すると、通常どおりマップの実行をステップオーバーするか続行することができます。実行結果のビジネス・オブジェクト値は「宛先テスト・データ」ペインに表示されます。
5. デバッグ・セッションは、「デバッグ」>「拡張」>「終了」を使用していつでも停止できます。

   結果: マップが閉じます。

関係を含むマップのテスト

関係変換を含むマップをテストするときは、テスト・データに加えて次の情報を指定する必要があります。

- 呼び出しコンテキスト

  マップの実行コンテキストには呼び出しのコンテキストも含まれる。マッピング API の多くの関係メソッドでは、この呼び出しのコンテキストを使用して、マッ

ピング中に行うアクションを決定する。そのため、マップの関係属性をテストする場合は、通常は変換に対して適切な呼び出しのコンテキストを指定する必要がある。

- 汎用ビジネス・オブジェクト定義

  一致関係の呼び出しコンテキスト SERVICE_CALL_RESPONSE をテストするとき、テスト実行で関係の汎用キー値を探し出せるようにするため、マップに汎用ビジネス・オブジェクトを指定する必要がある。

「テスト」タブの「ソース・テスト・データ」ペインでこの情報を指定します。

ヒント: 「ソース・テスト・データ」ペインの幅が狭く、「呼び出しのコンテキスト」コンボ・ボックスのメニュー・オプションの一部が見えない場合は、<-ll-> 記号が見えるまでカーソルを右側境界の上に移動し、境界を右側にドラッグすることで領域のサイズを拡張できます。

関係をテストする場合は、ビジネス・オブジェクトのリストから適切な汎用オブジェクトを選択し、「呼び出しのコンテキスト」を選択し、「テスト・データ」画面ですでに設定したオブジェクトと一致する親オブジェクトと子オブジェクトに ObjectEventId を設定します。指定が必要な呼び出しのコンテキスト、および汎用ビジネス・オブジェクトを指定する必要があるかどうかは、テストする関係のタイプによって異なります。このセクションの内容は次のとおりです。

- 『一致関係のテスト』
- 107 ページの『参照関係のテスト』

一致関係のテスト

一致関係の 2 地点間マッピング (アプリケーション 1 から アプリケーション 2 へ) をテストするには、次の 3 つのマップを使用します。

- Application 1 のアプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのインバウンド・マップ: App1_to_Generic
- 汎用ビジネス・オブジェクトから Application 2 のアプリケーション固有のビジネス・オブジェクトへのアウトバウンド・マップ: Generic_to_App2
- Application 2 のアプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのインバウンド・マップ: App2_to_Generic

例: 図 44 は Clarify アプリケーションと SAP アプリケーション間で行われる顧客データの 2 地点間通信の例を示しています。各アプリケーションが固有キー値を使用して顧客を識別する場合、これら 3 つのビジネス・オブジェクトは一致関係で関連付けられます。したがって、各マップには「相互参照」変換規則が組み込まれています。各マップが実行されると、これらの関係メソッドは実行するアクションを決定するため、呼び出しのコンテキストにアクセスします。

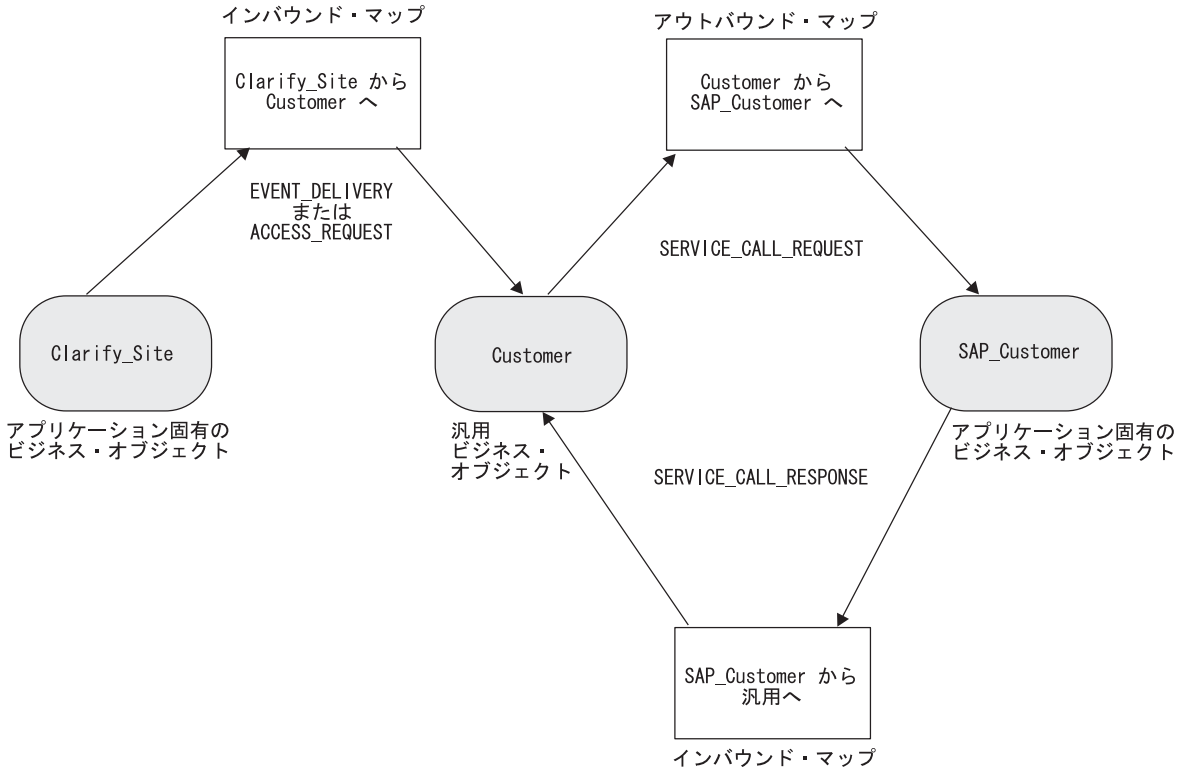

図 44. 一致関係の 2 地点間テストに関係するマップ

Create 動詞をテストするには、Application 1 (図 44 では Clarify アプリケーション) の新しいアプリケーション固有のキー値により、汎用ビジネス・オブジェクトに新しい汎用キー値が追加され、さらに Application 2 (図 44 では SAP アプリケーション) に新しいアプリケーション固有のキー値が追加されることを検証する必要があります。したがって、テストは次の 3 つの手順から構成されます。

1. インバウンド・マップ `App1_to_Generic` をテストし、Application 1 から新規キー値を送信し、汎用ビジネス・オブジェクトに新規キー値が生成されることを確認します。 表 21 の手順に従ってください。

表 21. 一致関係用の App1-to-Generic マップのテスト

テスト実行をセットアップするには	テスト実行を検査するには
1. 「呼び出しのコンテキスト」コンボ・ボックスから適切な呼び出しのコンテキストを選択して、呼び出しのコンテキストを `EVENT_DELIVERY` または `ACCESS_REQUEST` に設定します。 2. ソース・ビジネス・オブジェクトのキーにアプリケーション固有の値を入力します。この値は Application 1 のキー属性に対して固有の値です。 3. テストを実行します。	4. 宛先ビジネス・オブジェクトに生成された汎用キー値を読み取ります。この値は `App1/Generic` 一致関係の関係表に追加されています。 5. 宛先ビジネス・オブジェクトを選択し、コンテキスト・メニューから「保管」を選択して、`.bo` ファイル (例えば、`App1_to_Generic.bo`) に宛先ビジネス・オブジェクト・データを保管します。

2. アウトバウンド・マップ `Generic_to_App2` をテストし、新しい汎用キー値が Application 2 に送信されることを確認します。

アウトバウンド・マップ Generic_to_App2 の一致関係をテストするには、ソース・テスト・データで汎用キー値を指定する必要があります。次の処理が行われることがありますが、*どちらも間違っています。*

- 汎用ビジネス・オブジェクトの基本キー属性に任意の数を入力した後、マップを実行する。
- 関係表に直接レコードを作成する。

どちらの場合も、Map Designer Express は RelationshipRuntimeException または NullPointerException を生成します。 SERVICE_CALL_REQUEST が正しく動作するには、システムに汎用キー値が存在する必要があるため、エラーが発生しました。また、関係表は汎用キーが保管される唯一の場所ではありません。

正しい解決策は、同じ一致関係を使用するインバウンド・マップ EVENT_DELIVERY (または ACCESS_REQUEST) を最初に実行することです (1 を参照)。アウトバウンド・マップ Generic_to_App2 をテストするには、表 22 のステップに従ってください。

表 22. 一致関係用の generic-to-app2 マップのテスト

テスト実行をセットアップするには	テスト実行を検査するには
1. 「呼び出しのコンテキスト」コンボ・ボックスから SERVICE_CALL_REQUEST を選択して、呼び出しのコンテキストをこの値に設定します。 2. 前の手順のテスト結果を使用して汎用ビジネス・オブジェクトをロードします (例えば、App1_to_Generic.bo)。 3. テストを実行します。	4. 宛先ビジネス・オブジェクトに生成されたアプリケーション固有のキー値を読み取ります。Application 2 はまだキー値を生成していないため、この値は空です。 5. 宛先ビジネス・オブジェクトを選択し、コンテキスト・メニューから「保管」を選択して、.bo ファイル (例えば、Generic_to_App2.bo) に宛先ビジネス・オブジェクト・データを保管します。

3. インバウンド・マップ app2_to_generic をテストし、Application 2 の新規キー値が新規汎用キーと関連しているかどうかを検査します。

   呼び出しのコンテキストが SERVICE_CALL_RESPONSE である場合、一致関係はアプリケーション固有のビジネス・オブジェクトの ID を汎用ビジネス・オブジェクトの ID に相互参照させる必要があります。したがって、このテストでは、汎用ビジネス・オブジェクト定義を指定する必要があります。 表 23 の手順に従ってください。

表 23. 一致関係用の App2_to_Generic マップのテスト

テスト実行をセットアップするには	テスト実行を検査するには
1. 「呼び出しのコンテキスト」コンボ・ボックスから SERVICE_CALL_RESPONSE を選択して、呼び出しのコンテキストをこの値に設定します。 2. 「汎用ビジネス・オブジェクト」コンボ・ボックスから適切な汎用ビジネス・オブジェクトの名前を選択して、汎用ビジネス・オブジェクトを設定します。指定した汎用ビジネス・オブジェクトが「ソース・テスト・データ」ペインに追加されます。 3. 前の手順のテスト結果を使用してアプリケーション固有のビジネス・オブジェクトをロードします (例えば、Generic_to_App2.bo)。 4. アプリケーション固有のビジネス・オブジェクトで、ビジネス・オブジェクトのキーにアプリケーション固有の値を入力します。 5. 汎用ビジネス・オブジェクトに、Application 1 キーに関連した汎用キー値を入力します。この値は、EVENT_DELIVERY/ACCESS_REQUEST テスト (手順 1) で汎用ビジネス・オブジェクトに生成されたものと同じキー値になります。 6. テストを実行します。	7. 宛先ビジネス・オブジェクトに生成された汎用キー値を読み取ります。この値は、汎用ソース・ビジネス・オブジェクトに入力したものと同値になります。 8. Relationship Manager を使用し、正しいアプリケーション固有のキー値がこの一致関係用の汎用キー値に関連付けられていることを検査できます。

他の動詞のテストも同様のステップに従います。一致関係用の関係メソッドの動作の詳細については、281 ページの『第 8 章 関係のインプリメント』を参照してください。

参照関係のテスト

参照関係の 2 地点間マッピング (アプリケーション 1 から アプリケーション 2 へ) をテストするには、次の 2 つのマップを使用します。

- Application 1 のアプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのマップ: App1_to_Generic
- 汎用ビジネス・オブジェクトから Application 2 のアプリケーション固有のビジネス・オブジェクトへのマップ: Generic_to_App2

例: 図 45 は Clarify アプリケーションと SAP アプリケーション間で行われる顧客データの 2 地点間通信の例を示しています。各アプリケーションが特殊な静的コードを使用して都道府県を識別する場合、これら 3 つのビジネス・オブジェクトは参照関係で関連付けることができます。したがって、各マップには静的参照を実行するカスタム変換が組み込まれています。詳細については、171 ページの『例 3: 変換での Static Lookup の使用』に示す「静的参照」アクティビティーの例を参照して

ください。各マップが実行されると、これらの関係メソッドは実行するアクションを決定するため、呼び出しのコンテキストにアクセスします。

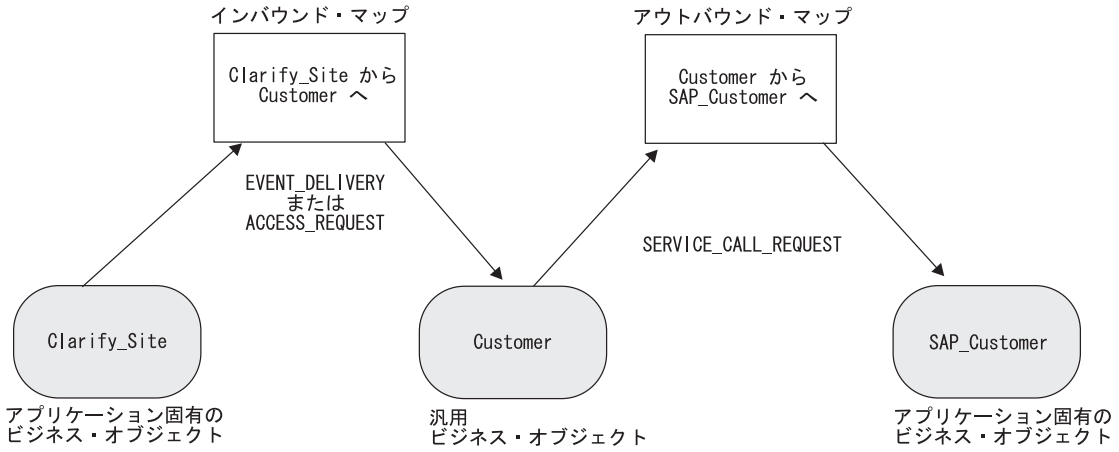

図 45. 参照関係の 2 地点間テストに関係するマップ

Create 動詞をテストするには、Application 1 (図 45 では Clarify アプリケーション) の既存のアプリケーション固有の参照値により、汎用ビジネス・オブジェクトに関連する汎用参照値が追加され、さらに Application 2 (図 45 では SAP アプリケーション) の関連するアプリケーション固有の参照値がそのビジネス・オブジェクトに追加されます。したがって、テストは次の 2 つの手順から構成されます。

1. インバウンド・マップ App1_to_Generic をテストし、Application1 から既存の参照値を送信し、汎用ビジネス・オブジェクト用に関連した汎用参照値が取得されることを確認します。 表 24 の手順に従ってください。

表 24. 参照関係用の App1-to-Generic マップのテスト

テスト実行をセットアップするには	テスト実行を検査するには
1. 「呼び出しのコンテキスト」コンボ・ボックスから適切な呼び出しのコンテキストを選択して、呼び出しのコンテキストを EVENT_DELIVERY または ACCESS_REQUEST に設定します。 2. ソース・ビジネス・オブジェクトの参照フィールドにアプリケーション固有の値を入力します。この値は、App1/Generic 関係表にすでにデータがロードされている既存の参照値です。 3. テストを実行します。	4. 宛先ビジネス・オブジェクトに生成された汎用参照値を読み取ります。この値は App1/Generic 参照関係の関係表に対して取得されています。 5. 宛先ビジネス・オブジェクトを選択し、コンテキスト・メニューから「保管」を選択して、.bo ファイル (例えば、App1_to_Generic.bo) にビジネス・オブジェクト・データを保管します。

2. アウトバウンド・マップ Generic_to_App2 をテストし、汎用参照値を送信して、Application 2 への関連する参照値が取得されていることを確認します。 表 25 の手順に従ってください。

表 25. 参照関係用の Generic-to-App2 マップのテスト

テスト実行をセットアップするには	テスト実行を検査するには
1. 「呼び出しのコンテキスト」コンボ・ボックスから SERVICE_CALL_REQUEST を選択して、呼び出しのコンテキストをこの値に設定します。 2. 前の手順のテスト結果を使用して汎用ビジネス・オブジェクトをロードします (例えば、App1_to_Generic.bo)。 3. テストを実行します。	4. 宛先ビジネス・オブジェクトに生成されたアプリケーション固有のキー値を読み取ります。この値は Application 2 参照値を含んでいます。 5. 宛先ビジネス・オブジェクトを強調表示にし、コンテキスト・メニューから「保管」を選択して、.bo ファイル (例えば、Generic_to_App2.bo) にビジネス・オブジェクト・データを保管します。

注: SERVICE_CALL_RESPONSE 呼び出しのコンテキストの参照関係をテストできます。しかし、通常、このテストが必要になるのは、マップが参照データを必要とする他の処理を行う場合のみです。マッピング API の参照関係用関係メソッドは、関係表にデータを書き込みません。

マップのデバッグ

このセクションでは、マップのデバッグに関する次の内容について説明します。

- 『ランタイム・エラーの解決』
- 110 ページの『デバッグのヒント』

関係のテスト方法については、103 ページの『関係を含むマップのテスト』を参照してください。

ランタイム・エラーの解決

マップのコンパイルに成功しても、デバッガーでのマップの実行中にランタイム・エラーが発生することがあります。

例 1: 一方には汎用ビジネス・オブジェクトを持つアウトバウンド・マップがあり、もう一方にはアプリケーション固有のビジネス・オブジェクトがあります。このマップの内部には一致関係があるとします。

1. 「テスト」タブに切り替えて、呼び出しのコンテキスト SERVICE_CALL_REQUEST を選択します。
2. 動詞「Update」を選択します。
3. テストを実行します。

   結果: 次のようなエラー・メッセージが表示されます。

```
Exception at step 17,
attribute <attribute name>,
java.lang.nullpointerexception
```

この例外は、元々作成されていないリポジトリー内のエントリーをマップが更新しようとしたために発生しました。理想的には、ステップの順序が正しいかを確認する必要があります。問題のマップに属する関係エントリーのデータベースを調べてください。そして、SERVICE_CALL_REQUEST が作動可能かどうかに基づいて、結論を導き出す必要があります。

例 2: Customer.CustomerId に対する次のマッピング・コード行があります。

```
_cw_CpBTBSourceValue = ObjSAP_CustomerMaster.get("CustomerIdd");
```

ここには、明らかに入力ミスがあります (属性名に余分な文字 *d* があります)。残念なことに、エラーはストリング定数であるため、コンパイラーはこのエラーを検出しません。コンパイラーでは、定数値が「正しい」かどうかを検査する方法がありません。しかし、マップを実行すると、次の ICS Express エラー・ダイアログが表示されます。

```
ICS Error: Exception at step 3, attribute CustomerId, Exception msg
number - 11030, Error11030 Attribute CustomerIdd doesn't exist in business
object SAP_CustomerMaster.
```

このランタイム・エラーを受け取った場合は、「テスト」タブを退出して、マップを修正する必要があります。

デバッグのヒント

このセクションでは、マップのデバッグに役立つ次のヒントについて説明します。

- 『ロギング・メッセージの使用』
- 111 ページの『安全なマッピング・コードの作成』

ロギング・メッセージの使用

マップの実行をトラッキングするには、logInfo() メソッドを使用します。このメソッドは引き数として String をとり、InterChange Server Express ログに送信されます。このメソッドは、実行をトラッキングする必要がある属性を対象として、Activity Editor で入力する必要があります。サブマップが実行されたことを確認するには、カスタム変換規則を作成し、「Log Information」関数ブロックを使用してアクティビティーをカスタマイズするか、コードを直接書き込みます。

例: コードは、次のような単純な形式になります。

```
logInfo("in submap");
```

サブマップで宛先オブジェクトの最初の属性の先頭コード行に指定します。

例: SAP.CustomerName など、特定の属性の値をトラッキングする必要がある場合、次のように使用します。

```
logInfo(ObjSAP_CustomerMaster.getString("CustomerName"));
```

このメッセージの表示が必要ないこともあります。必要がない場合、マップの DataValidationLevel プロパティーを変更します。

DataValidationLevel を設定するには、Map Designer Express の「編集」メニューから「マップ・プロパティー」オプションを選択し、0 を 1 以上の数に変更します。設定は次のとおりです。

0	データ検証なし
1	IBM データ検証レベル
2 以上	ユーザー定義のデータ検証

logInfo メッセージが表示されないようにするには、DataValidationLevel を 1 に設定します。ユーザーのコードで、logInfo() メソッドを呼び出す前に、データ検証レベルを検査してください。コードは次のとおりです。

```
if (dataValidationLevel > 1)
   logInfo(ObjSAP_CustomerMaster.getString("CustomerName"));
```

このコードを使用すると、データ検証レベルが 1 より大きい数値に設定されている場合にのみ logInfo は実行されるようになります。メッセージを表示する場合は、「マップ・プロパティー」でデータ検証レベル設定を 2 に変更します。

安全なマッピング・コードの作成

変換規則を Activity Editor でカスタマイズする場合や、独自のマッピング・コードを作成する場合、そのコードが実行時に正しく動作する*保証はありません*。エラーが発生してもマップの実行を継続し、エラーの通知を受け取るようにするには、Activity Editor で「Catch Error」関数ブロックを使用するか、Java の例外処理の方法に従ってください。

例: 次のように、try ブロック内にコードを入れます。

```
try
   {
   BusObj temp = new BusObj("SAP_Order");
// rest of your code
   }
```

次に、catch ブロックを使用し、コードの実行時に発生するすべての例外をキャッチします。

```
catch (Exception e)
   {
   logInfo(e.toString());
   }
```

logInfo() メソッドを使用し、システムが生成したエラー・メッセージを InterChange Server Express ログに送信できます。

第 5 章 マップのカスタマイズ

この章では、Java コードを生成する 2 つの方法について説明します。一つは Activity Editor を使用して変換規則をグラフィカルに定義する方法で、もう一つは Java コードを直接書き込む方法です。

この章の内容は次のとおりです。

- 『Activity Editor の概要』
- 123 ページの『アクティビティー定義の処理』
- 174 ページの『Java パッケージと他のカスタム・コードのインポート』
- 180 ページの『Web サービスを Activity Editor にエクスポートする』
- 184 ページの『変数の使用』
- 190 ページの『その他の属性の変換方法』
- 202 ページの『マップ・インスタンスの再利用』
- 203 ページの『例外処理』
- 205 ページの『カスタム・データ検証レベルの作成』
- 207 ページの『マップの実行コンテキストの理解』
- 211 ページの『子ビジネス・オブジェクトのマッピング』
- 216 ページの『サブマップの使用の続き』
- 223 ページの『データベース照会の実行』

Activity Editor の概要

Activity Editor を使用すると、プログラミングや Java コードに関する知識がなくても、特定の変換規則のアクティビティーの流れをグラフィカルに指定できます。Map Designer Express の各変換規則ごとに、1 つのアクティビティーとそのサブアクティビティーを表示できます。関連付けられた属性の変換コードをグラフィカルに表示させ、そのコードを変更し、ツールによって対応する Java コードを生成させることができます。

Activity Editor は、 Map Designer Express から直接起動します (114 ページの『Activity Editor の始動』を参照してください)。始動時には、Activity Editor は System Manager と通信し、許可されている一連のアクティビティーを検索します。特定の変換規則のアクティビティーを設計し、Activity Editor で変更を保管すると、変更内容が Map Designer Express に伝達されます。

このセクションには、Activity Editor の概要を説明する以下のトピックがあります。

- 114 ページの『Activity Editor の始動』
- 114 ページの『Activity Editor のレイアウト』
- 119 ページの『Activity Editor 機能の使用』

Activity Editor の始動

Activity Editor は、Map Designer Express の「テーブル」タブまたは「ダイアグラム」タブの変換規則列から起動します。これを行うには、以下の手順を実行します。

1. 処理したい属性を選択します。
2. 次のいずれかを行います。
   - 変換規則列の属性の対応するセルをダブルクリックします。
   - 変換規則列の対応するセル内のビットマップ・アイコンをクリックします。

結果: これらのアクションに対する Map Designer Express の応答は、以下の要因によって異なります。

- コードが自動更新モードであるか否か

  変換コードが Map Designer Express によって生成されており、何もカスタマイズしていない場合、その変換コードは自動更新モードになります。自動更新コードをカスタマイズすると、Activity Editor により確認プロンプトが表示され、自動更新モードから外れてコードが保管されることが通知されます。コードが自動更新モードでない場合、Map Designer Express は、変換規則の列に変換規則を青のイタリック・フォントで表示します。

  変換コードが自動更新モードでない 場合 (つまり、自動生成されたコードを変更した場合) は、属性の変換規則セルをダブルクリックするか、マッピング規則アイコンをクリックすると、Map Designer Express は Activity Editor を Java 表示で開きます。

- 定義された変換タイプ

  自動更新モードでの変換コードは、Map Designer Express が変換規則列のコンボ・ボックスに用意する標準変換の 1 つから生成されます。属性の変換規則セルをダブルクリックするか、マッピング規則アイコンをクリックすると、変換タイプによって Map Designer Express の表示内容が決定します。
  - カスタム変換の場合は、変換コードに関する Activity Editor が開きます。
  - その他のすべての標準変換 (値を設定、結合、分割、サブマップ、および相互参照) の場合は、その変換のダイアログが表示されます。このダイアログで「コードを表示」プッシュボタンをクリックすると、Activity Editor が新規ウィンドウで開き、そのタイトル・バーに属性名が表示されます。複数の Activity Editor のインスタンスを同時に開くことができます。

Activity Editor のレイアウト

Activity Editor には、グラフィック表示と Java 表示の 2 つのメイン表示があります。特定の時点のアクティビティーの性質に応じて、いずれか一方のみが可視となります。したがって、Map Designer Express がグラフィカル・アクティビティーを表示するために Activity Editor を起動する場合、Activity Editor はグラフィック表示で開始します。このグラフィカル・アクティビティーを Java コードに翻訳すると、グラフィック表示の代わりに Java 表示が表示されます。

制限: アクティビティーを Java コードに変更した後は、グラフィカルな性質に変換されません。

どちらの表示のデザイン・モードおよびクイック表示モードにも、表 26 に示す共通のウィンドウの要素があります。

表 26. 共通のウィンドウの要素

ウィンドウの要素	説明
タイトル・バー	アプリケーションの名前 (Activity Editor)、アプリケーション・アイコン、およびメイン・アクティビティーの名前を含みます。
メニュー	1 次メニューを含みます (デザイン・モードのみ)。
ツールバー	さまざまな機能およびツールへのショートカットを持つ連結可能なツールバーを含みます (デザイン・モードのみ)。
文書表示域	アクティビティー定義の表現を表示します。ワークブックのように編成されています。
ステータス・バー	状況情報および一部の便利なショートカットを表示します。

グラフィック表示での作業

Map Designer Express から Activity Editor を起動して、グラフィカルに表現できるアクティビティー定義を開いた場合、そのアクティビティー定義は、Activity Editor で使用可能な 2 つの表示モード (デザイン・モードまたはクイック表示モード) のいずれか一方のグラフィック表示に表示されます。

- **デザイン・モード:** デザイン・モードでは、Activity Editor の表示は通常のアプリケーションと類似しています。メイン編集ウィンドウのほかに、メニュー・バー、ツールバー、「ライブラリー」ウィンドウ、「コンテンツ」ウィンドウ、および「プロパティー」ウィンドウがあり、これらはアクティビティー定義の設計段階で必要となる編集をサポートします。

  116 ページの図 46 に、デザイン・モードのグラフィック表示を示します。

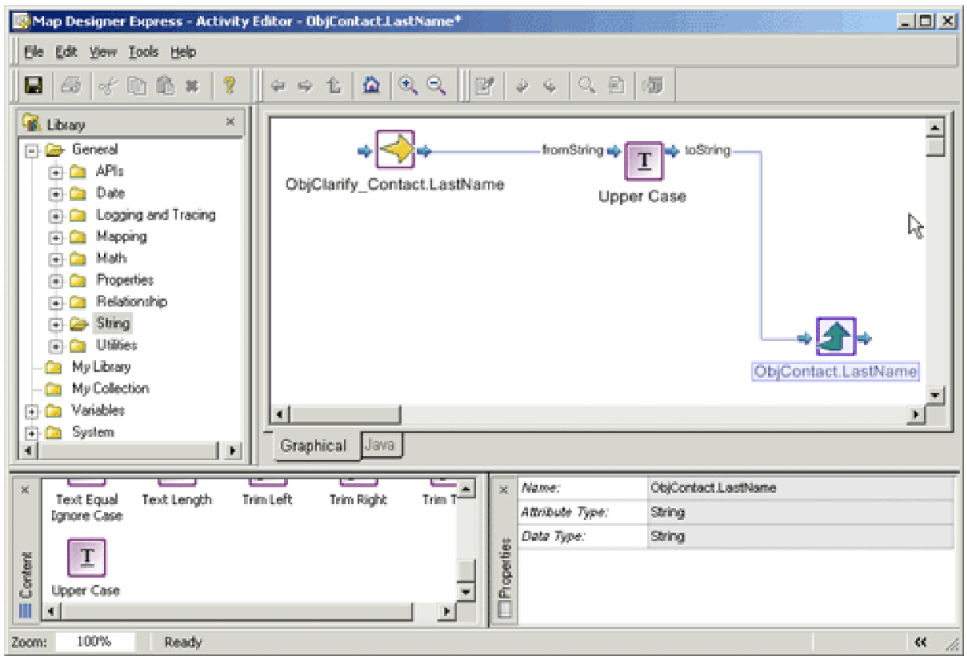

図 46. デザイン・モードのグラフィック表示

グラフィック表示には、メイン・ウィンドウとして、「アクティビティー・ワークブック」ウィンドウ、「ライブラリー」ウィンドウ、「コンテンツ」ウィンドウ、および「プロパティー」ウィンドウの 4 つがあります。

– 「アクティビティー・ワークブック」ウィンドウ: このウィンドウはメインのアクティビティー編集域であり、通常は編集キャンバスと呼ばれます。アクティビティー・キャンバスやグラフィカル・キャンバスとも呼ばれます。この領域に、関数ブロックをドラッグ・アンド・ドロップします。

– 「ライブラリー」ウィンドウ: 使用可能な関数ブロックおよびオプションで名前付きブロックのツリー表示が表示されます。関数ブロックは、目的に従ってフォルダーに並べられます (126 ページの『サポートされている関数ブロックの識別』 を参照)。フォルダーを展開すると実際の関数ブロックが表示されます。関数ブロックは、「コンテンツ」ウィンドウのアイコンとして表示させることもできます。

  さらに「ライブラリー」ウィンドウには、以下のフォルダーがあります。

  - システム: このフォルダーには、編集キャンバスに追加できるシステム・エレメントが含まれます。システム・エレメントとしては、コメント、説明、ラベル、予定タグ、および定数があります。
  - ライブラリー: このフォルダーを使用すると、「ライブラリー」ウィンドウをカスタマイズできます。ここには、System Manager の「Activity の設定」ビューで指定されたユーザー定義の関数ブロックが含まれます。
  - ユーザー・コレクション: このフォルダーを使用すると、頻繁に使用するコンポーネントのコレクションを作成できます。このフォルダーに通常の関数ブロックを置いたり、ユーザー独自の再使用可能コンポーネント・グループを作成したりできます。
  - 変数: このフォルダーには、現在のアクティビティーにアクセス可能なグローバル変数が含まれます。一般的には、ポートのビジネス・オブジェクト変

数、シナリオに定義されたその他のすべてのビジネス・オブジェクトと変数、およびグローバル変数 cwExecCtx が含まれます。

- 「コンテンツ」ウィンドウ: 「ライブラリー」ウィンドウで現在選択されているフォルダーで使用可能な関数ブロックの大きなアイコンのリストを含みます。関数ブロックを選択すると、「プロパティー」ウィンドウにその記述およびプロパティーが表示されます。また、関数ブロックを編集キャンバスにドラッグ・アンド・ドロップすると、アクティビティー・フローの一部が作成されます。
- 「プロパティー」ウィンドウ: 現在選択されている関数ブロックのプロパティーがグリッド状のレイアウトで表示されます。一部のプロパティーは編集可能ですが、その他は読み取り専用です。

- **クイック表示モード:** クイック表示モードでは、Activity Editor にはメイン編集キャンバスのみが表示されます。サポートされているその他のウィンドウ (ライブラリー、コンテンツ、およびプロパティー)、メニュー・バー、およびツールバーは表示されません。

図 47 に、クイック表示モードのグラフィック表示を示します。

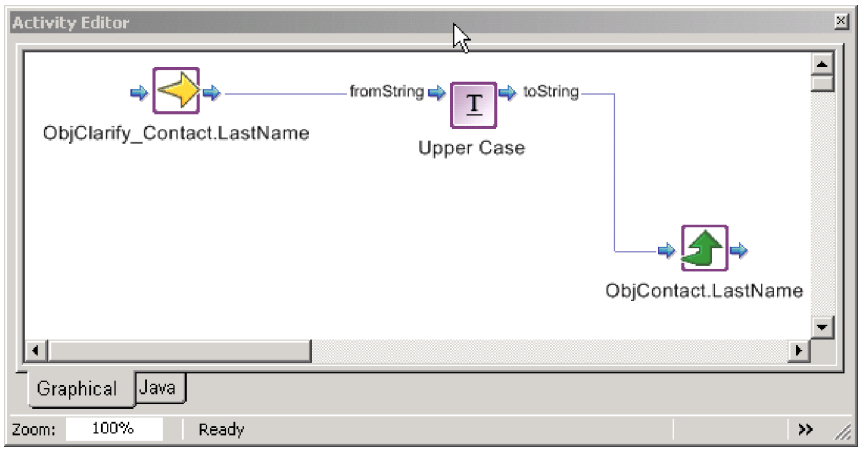

図 47. クイック表示モードのグラフィック表示

最初の状態では、グラフィックの性質を持つアクティビティー定義を開くと、Activity Editor は、タブ付きウィンドウに定義のトップレベル・ビューを表示します。タブ付きウィンドウには*編集キャンバス* があります。編集キャンバスでのアクティビティー定義の操作については、123 ページの『アクティビティー定義の処理』 を参照してください。

Java 表示での作業

Map Designer Express が Activity Editor を起動して、カスタム Java コードのみが含まれるアクティビティー定義を開く場合、そのアクティビティー定義は Java 表示で表示されます。グラフィック表示と同様に、Java 表示の Activity Editor でもデザイン・モードおよびクイック表示モードの 2 つの表示モードを使用できます。

- **「デザイン・モード」:** デザイン・モードでは、Activity Editor の Java 表示にメイン Java WordPad が表示されます。ここでは、カスタム Java コードを表示および編集してアクティビティーの定義を指定できます。 WordPad はタブ付きウ

ィンドウ領域に表示されます。WordPad の通常の編集オプション (切り取り、コピー、貼り付け、削除、全選択、元に戻す、やり直し) に加え、Java WordPad では Java プログラム言語の構文が強調表示されます。

デフォルトでは、コメントは緑色で、ストリング・リテラルはピンク色で、キーワードは青色で表示されます。

ヒント:「設定」ダイアログで構文を強調表示する方式をカスタマイズできます。

図 48 に、デザイン・モードの Java 表示を示します。

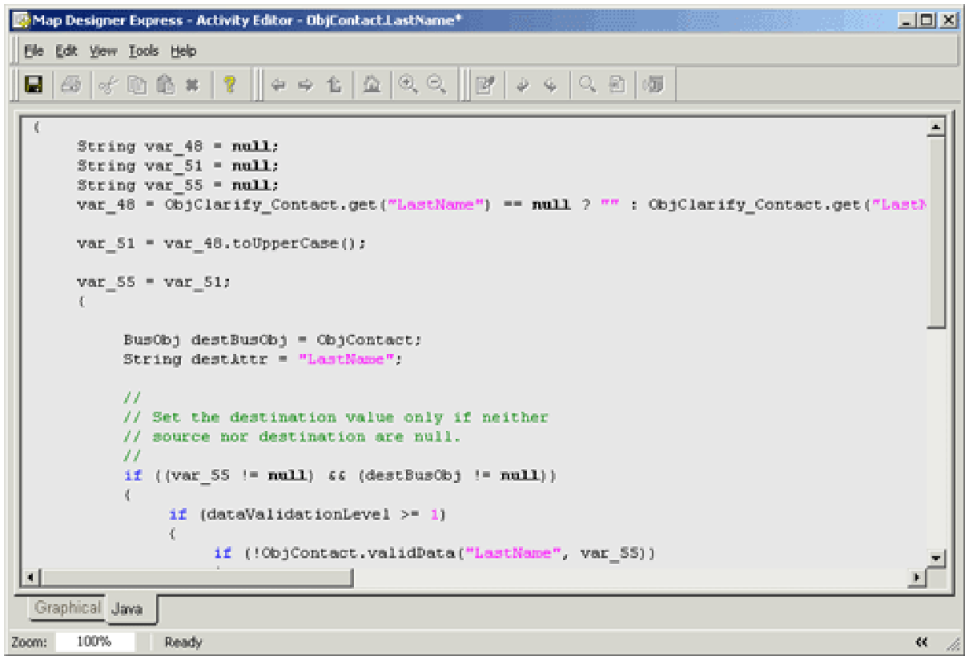

図 48. デザイン・モードの「Java」表示

- **「クイック表示モード」:** クイック表示モードでは、Java 表示には WordPad のみが表示されます。図 49 に、クイック表示モードの Java 表示を示します。

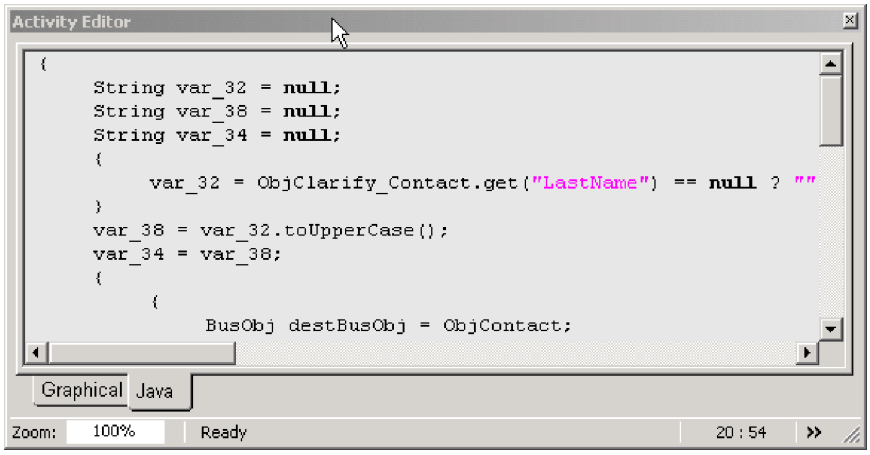

図 49. クイック表示モードの「Java」表示

ヒント: クイック表示モードからデザイン・モードに変更するには、ステータス・バーの「>>」ボタンをクリックします。「>>」ボタンが表示されていない場合は、ボタンが表示されるまで「Quick View」ウィンドウのサイズを変更します。

注: 最初の状態では、Java 表示は読み取り専用モードです。カスタマイズされた Java コードを入力するには、「コードを編集」ツールバー・ボタンをクリックするか、「ツール」メニューから「コードを編集」を選択します。

Activity Editor 機能の使用

以下のいずれを使用しても Activity Editor の機能にアクセスできます。

- プルダウン・メニュー
- コンテキスト・メニュー
- ツールバーのボタン
- キーボードのショートカット

Activity Editor のプルダウン・メニューおよびキーボードのショートカット

Activity Editor には、以下のプルダウン・メニューがあります。

- 「ファイル」メニュー
- 「編集」メニュー
- 「表示」メニュー
- 「ツール」メニュー
- 「ヘルプ」メニュー

以下のセクションでは、これらの各メニューのオプションおよび関連したキーボード・ショートカットについて説明します。

「ファイル」メニューの機能: Activity Editor の「ファイル」プルダウン・メニューには以下のオプションがあります。

- 「保管」[Ctrl+S]: アクティビティーを Map Designer Express に保管します。

- 「印刷設定」[Ctrl+Shift+P]: 「印刷設定」ダイアログ・ボックスが開きます。このダイアログ・ボックスでは印刷オプションを指定できます。
- 「印刷プレビュー」: Activity Editor を印刷プレビュー・モードに切り替えます。
- 「印刷」[Ctrl+P]:「印刷」ダイアログ・ボックスが開きます。このダイアログ・ボックスでは現在のアクティビティーを印刷できます。
- 「閉じる」: Activity Editor を閉じます。

「編集」メニューの機能: Activity Editor の「編集」プルダウン・メニューには以下のオプションがあります。

- 「元に戻す」[Ctrl+A]: 最後に行ったアクションを元に戻します。
- 「やり直し」[Ctrl+Y]: 最後に元に戻したアクションをやり直します。
- 「切り取り」[Ctrl+X]: 選択状態の項目を削除し、クリップボードにコピーします。
- 「コピー」[Ctrl+C]: 選択状態の項目をクリップボードにコピーします。
- 「貼り付け」[Ctrl+P]: クリップボードにあるオブジェクトをカーソル位置に貼り付けます (互換性がある場合) 。
- 「削除」[Del]: 選択状態の項目を削除します。
- 「全選択」[Ctrl+A]: すべての項目を選択します。
- 「検索」[Crtl+F]: 編集域で特定のテキストを検索します。
- 「置換」[Ctrl+H]: 編集域で特定のテキストを別のテキストに置換します。
- 「行に移動」[Ctrl+G]: 特定の行に移動します。

「表示」メニューの機能: Activity Editor の「表示」プルダウン・メニューには以下のオプションがあります。

- 「デザイン・モード」: デザイン・モードとクイック表示モードを切り替えます。(特定の時点で使用可能なのはいずれかのモードのみです。)
- 「クイック表示モード」: クイック表示モードとデザイン・モードを切り替えます。(特定の時点で使用可能なのはいずれかのモードのみです。)
- 「移動」: 以下のオプションを提供します。
  - 「戻る」[Alt+Left Arrow]: グラフィック表示のナビゲーション履歴で逆方向に移動します。
  - 「進む」[Alt+Right Arrow]: グラフィック表示のナビゲーション履歴で次に移動します。
  - 「1 レベル上へ」: 1 つ上のレベルからダイアグラムを表示します。
  - 「ホーム」[Alt+Home]: グラフィック表示でトップレベルのダイアグラムに移動します。
- 「ズームイン」[Ctrl++]: Activity Editor の内容を拡大します。
- 「ズームアウト」[Ctrl+-]: Activity Editor の内容を縮小します。
- 「倍率指定ズーム」[Crtl+M]: 「ズーム」ダイアログ・ボックスが開きます。このダイアログ・ボックスでは特定のズーム・レベルを指定できます。
- 「ライブラリー」ウィンドウ:「ライブラリー」ウィンドウのオンとオフを切り替えます。

- 「コンテンツ」ウィンドウ: 「コンテンツ」ウィンドウのオンとオフを切り替えます。
- 「プロパティー」ウィンドウ:「プロパティー」ウィンドウのオンとオフを切り替えます。
- 「ツールバー」: ツールバー (標準、グラフィックス、および Java) のオンとオフを切り替えるサブメニューが開きます。
- 「ステータス・バー」: ステータス・バーのオンとオフを切り替えます。
- 「設定...」[Ctrl+U]:「設定」ダイアログ・ボックスを開きます。このダイアログ・ボックスでは、Activity Editor のデフォルトの振る舞いを指定できます。

「ツール」メニューの機能: Activity Editor の「ツール」プルダウン・メニューには以下のオプションがあります。

- 「変換」[Ctrl+T]: 現在のアクティビティーを Java コードに変換し、Java 表示を開きます。
- 「コードを編集」: Java のコードを編集できます。
- 「対応しない区切り文字を検査」: Java コードに対応しない区切り文字があるかどうかを検査します。
- 「式ビルダー」: 式ビルダー・ユーティリティーを開きます。

「ヘルプ」メニューの機能: Activity Editor の「ヘルプ」プルダウン・メニューには以下のオプションがあります。

- 「ヘルプ・トピック」[F1]: 状況依存ヘルプ・トピックを開きます。
- 「ドキュメンテーション」: InterChange Server Express の資料を開きます。

コンテキスト・メニュー

アクティビティーには、編集キャンバス上で多くのタスクを実行するためのコンテキスト・メニューもあります。コンテキスト・メニューにアクセスするには、編集中のキャンバスを右マウス・ボタンでクリックします。コンテキスト・メニューには以下のオプションがあります。

- 「新規定数」: 編集中のキャンバスに新規定数コンテナーを作成します。
- 「ラベルを追加」: 編集中のキャンバスに新規ラベル・コンポーネントを作成します。
- 「記述を追加」: 編集中のキャンバスに新規記述コンポーネントを作成します。
- 「コメントを追加」: 編集中のキャンバスに新規コメント・コンポーネントを作成します。
- 「予定を追加」: アクティビティーに新規の覚え書コンポーネントを作成します。
- 「ユーザー・コレクションに追加」: 再利用できるように新規グループ・コンポーネントを「ライブラリー」ウィンドウに作成します。

Activity Editor のツールバー

Activity Editor には、ユーザーが実行する必要のある一般的なタスクのためのツールバーが 3 つあります。

- 標準ツールバー
- グラフィックス・ツールバー

- Java ツールバー

ツールバー・ボタンの機能は、対応するメニュー項目と同じです。

ヒント: それぞれのツールバー・ボタンの機能を確認するには、マウス・カーソルで各ボタンをロールオーバーしてください。

標準ツールバー: 図 50 に標準ツールバーを示します。

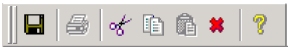

図 50. Activity Editor の標準ツールバー

表 27 に、標準ツールバーの各ボタンおよび対応するメニュー・コマンドを (左から右の順に) 示します。

表 27. 標準ツールバー・ボタンの機能

機能	対応するメニュー・コマンド
アクティビティーを保管	「ファイル」>「保管」
Print Activity	「ファイル」>「印刷」
切り取り	「編集」>「切り取り」
コピー	「編集」>「コピー」
貼り付け	「編集」>「貼り付け」
削除	「編集」>「削除」
ヘルプ	「ヘルプ」>「ヘルプ・トピック」

グラフィックス・ツールバー: 図 51 にグラフィックス・ツールバーを示します。

図 51. Activity Editor のグラフィックス・ツールバー

表 28 に、グラフィックス・ツールバーの各ボタンおよび対応するメニュー・コマンドを (左から右の順に) 示します。

表 28. グラフィックス・ツールバー・ボタンの機能

機能	対応するメニュー・コマンド
戻る	「表示」>「移動」>「戻る」
進む	「表示」>「移動」>「進む」
1 レベル上へ	「表示」>「移動」>「1 レベル上へ」
ホーム	「表示」>「移動」>「ホーム」
ズームイン	「表示」>「ズームイン」
ズームアウト	「表示」>「ズームアウト」

123 ページの図 52 は、Java ツールバーを示しています。

図 52. Activity Editor の Java ツールバー

表 29 に、Java ツールバーの各ボタンおよび対応するメニュー・コマンドを (左から右の順に) 示します。

表 29. Java ツールバー・ボタンの機能

機能	対応するメニュー・コマンド
コードを編集	「ツール」>「コードを編集」
元に戻す	「編集」>「元に戻す」
やり直し	「編集」>「やり直し」
テキストを検索	「編集」>「検索」
行に移動	「編集」>「行に移動」
式ビルダー	「ツール」>「式ビルダー」

ステータス・バーの要素: Activity Editor にはステータス・バーもあります。これを図 53 に示します。

図 53. Activity Editor のステータス・バー

表 30 に、ステータス・バーの各要素の機能を (左から右の順に) 示します。

表 30. ステータス・バーの要素の機能

要素	機能
ズーム: 100%	ズームのパーセンテージを指定するための編集ボックス
作動可能	状況メッセージ
10.9	Java エディターにおける *I* バー の現在位置を示すナビゲーション・ペイン
>> (クイック表示モードで表示します)	デザイン・モードとクイック表示モードを切り替えます
<< (デザイン・モードで表示します)	

アクティビティー定義の処理

Activity Editor は、変換規則のアクティビティー定義の定義および変更に使用されます。この作業は、編集キャンバス上でキャンバス・コンポーネントを使用して行います。

- *関数ブロック* : アクティビティー定義は*関数ブロック* を使って作成されます。関数ブロックは、アクティビティー定義の個別のパーツを表すもので、定数、変数、またはプログラミング・メソッドなどがあります。Activity Editor における関数ブロックの多くは、マッピング API の個々のメソッドに対応します。編集キャンバスに関数ブロックを置くには、「ライブラリー」ウィンドウまたは「コンテンツ」ウィンドウから関数ブロックをドラッグ・アンド・ドロップします。関

数ブロックを編集キャンバス上にドロップしたら、その関数ブロックは移動させることができます。それには、関数ブロックをクリックして選択し、希望の場所までドラッグします。

関数ブロックには入力、出力、あるいはその両方があります。各関数ブロックの入出力は事前定義されており、指定された値タイプのみを受け入れます。関数ブロックが編集キャンバス上にドロップされると、その入出力ポートが矢印で表されます。これらのポートは、関数ブロックと他のコンポーネントの間をリンクするための接続点として機能します。デフォルトでは、各入出力の名前がその接続ポートの横に表示されます (名前を隠すには、「表示」>「設定」オプションを使用します)。

Map Designer Express および Relationship Designer Express のコンテキストでサポートされている関数ブロックについては、126 ページの『サポートされている関数ブロックの識別』を参照してください。

注: Activity Editor では、標準の関数ブロックを使用できるだけでなく、ユーザー独自の Java ライブラリーもインポートして関数ブロックとして使用することができます。カスタム JAR ライブラリーをアクティビティー設定にインポートすると、その JAR ライブラリーに含まれる public メソッドを、Activity Editor で関数ブロックとして使用できます。詳細については、174 ページの『Java パッケージと他のカスタム・コードのインポート』 を参照してください。

- *接続リンク* : 関数ブロックは*接続リンク* によって接続されます。接続リンクは、アクティビティー定義におけるさまざまなコンポーネント間のアクティビティーのフローを定義します。接続リンクによって、ある関数ブロックの出力ポートが別の関数ブロックの入力ポートに接続されます。

  注: 出力ポートは複数の接続リンクに接続できますが、入力ポートは 1 つの接続リンクにしか接続できません。

  ヒント: ユーザーがドラッグ・アンド・ドロップ操作で関数ブロック間を接続すると、Activity Editor では、「設定」ダイアログの「検証」タブのオプション設定に基づいて、ドラッグ・アンド・ドロップ元とドラッグ・アンド・ドロップ先のパラメーターの型が一致しているかどうかを検証する必要の有無を判断します。

  – このオプションは、デフォルトでは「警告」に設定されています。つまり、型の異なる 2 つのパラメーターの間にリンクを作成しようとすると、コンパイル・エラーが発生するおそれがあることを通知するメッセージが Activity Editor によって表示されます。
  – このオプションを「無視」に設定すると、Activity Editor に対し、検証を行わないように指示したことになります。
  – このオプションを「エラー」に設定すると、Activity Editor に対し、型が一致しない場合はリンクの作成を許可しないように指示したことになります。

  例: 関数ブロック A の出力が関数ブロック B の入力になることを指定するには、以下の手順を実行します。

  1. 関数ブロック A の出力ポートの上で左マウス・ボタンをクリックし、押したままにします。

2. 左マウス・ボタンを押した状態で、カーソルを関数ブロック B の入力ポートまで移動させます。
3. 左マウス・ボタンを放します。

結果: 接続リンクは、関数ブロック A の出力ポートと関数ブロック B の入力ポートの間に配置されます。視覚的には、接続リンクはコンポーネント間の直角に曲がった線として表示されます。関数ブロック B の入力ポートがすでに他の接続リンクに接続されている場合は、新しい接続リンクによって既存の接続リンクが置き換えられます。

- *ラベル、説明、コメント、および予定 タグ:* 「システム」フォルダー (「ライブラリー」ウィンドウおよび「コンテンツ」ウィンドウにある) には、コメント、説明、ラベル、および予定タグをアクティビティー定義に追加する関数ブロックが含まれています。これらのタグは、各アクティビティーまたはサブアクティビティーを識別するのに役立ちます。また、行うべき作業の覚え書にもなります。これらの関数ブロックを、他の関数ブロックと同様に、編集キャンバス上にドラッグ・アンド・ドロップします。ただし、入出力ポートはありません。

  新規タグを編集するには、タグの中央をシングルクリックします。カーソルが I 形に変化し、テキストの入力が可能になります。タグのテキスト行が長くなると、そのテキスト行は自動的に折り返されます。改行する場合は、Enter キーを押してください。

  タグのサイズを変更するには、タグの右下隅を左マウス・ボタンでクリックし、左マウス・ボタンを押したまま、タグを希望のサイズになるまでドラッグします。

  図 54 は、ラベル・タグのサイズ変更および複数行テキストの入力を示します。

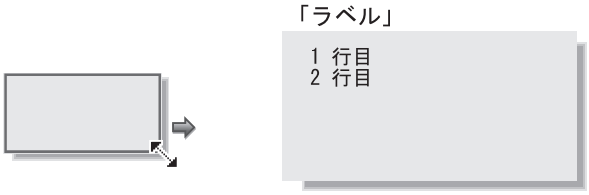

図 54. ラベルのサイズ変更および複数行テキストの入力

制約事項: これらの編集コンポーネントにはそれぞれ最小化サイズがあるため、コンポーネントを一定のサイズより小さくすることはできません。

キャンバスでタグを移動するには、コンポーネントの端をクリックしてドラッグ・アンド・ドロップします。

- *「新規定数」アイコン*: Activity Editor には、New Constant 関数ブロックがあります。これを編集キャンバス上にドラッグ・アンド・ドロップすれば、他の関数ブロックへの入力として設定し使用する定数値を定義できます。New Constant 関数ブロックは、「ライブラリー」ウィンドウおよび「コンテンツ」ウィンドウの「システム」フォルダーにあります。この関数ブロック・アイコンの上に、定数

値を入力するためのテキスト編集ボックスが表示されます。この値を変更するには、「定数」アイコンをダブルクリックして新しい値を入力します。「定数」には 1 つの出力ポートがあります。

注: 「定数」は、単一行の値のみを受け入れる唯一のアクティビティー定義コンポーネントです。定数が Java コード文字列に変換されることと、システムが複数行の定数値を変換できないことがその理由です。複数行の入力データが必要な場合は、「定数」内の行の間を「¥n」値で区切ってください。

例: 値が「line1¥nline2」の場合、テキストを 2 行で出力するよう、システムに指示が出されます。

コンポーネントのグループ化の手順

編集キャンバス上で関数ブロックのセットを使ってアクティビティー・フローを定義したら、このアクティビティー・フロー全体または一部を選択し、名前付きグループとして保管できます。この名前付きグループは、通常の関数ブロックのように後から別のアクティビティー定義で再利用できます。以下に手順を示します。

始める前に: 「設定」ダイアログで「ライブラリー・ウィンドウで子関数を表示」を使用可能にし、追加したグループを表示させる必要があります。

以下の手順を実行します。

1. 編集キャンバス上で一まとめにしたい関数ブロックを選択します。複数の関数ブロックを選択するには、`Ctrl` キーを押したまま各関数ブロックをクリックします。
2. キャンバスを右マウス・ボタンでクリックし、コンテキスト・メニューを開きます。次に「ユーザー・コレクションに追加」を選択します。

   結果: 「ユーザー・コレクションに追加」ダイアログ・ボックスが表示されます。
3. 「ユーザー・コレクションに追加」ダイアログで、作成するコンポーネント・グループの名前と説明 (オプション) を入力して、このグループを表すアイコンを選択します。次に「OK」をクリックします。

   結果: 追加したグループが「ユーザー・コレクション」フォルダーの「ライブラリー」ウィンドウおよび「コンテンツ」ウィンドウに表示されます。このアイコンは、編集キャンバス上にドラッグ・アンド・ドロップして、アクティビティー定義に使用することができます。

サポートされている関数ブロックの識別

Map Designer Express コンテキストでサポートされている関数ブロックは、次の表に示すカテゴリーに分類されます。これらのカテゴリーは、「ライブラリー」ウィンドウおよび「コンテンツ」ウィンドウのフォルダーに対応しています。

表 31. 関数ブロックの構成

関数ブロックのフォルダー	説明	詳細情報の参照先
General/APIs/Business Object	ビジネス・オブジェクトの処理に関する関数ブロック。	128 ページの表 32

表 31. 関数ブロックの構成 (続き)

関数ブロックのフォルダー	説明	詳細情報の参照先
General/APIs/Business Object/Array	BusObj クラスの Java 配列の処理に関する関数ブロック。	132 ページの表 33
General/APIs/Business Object/Constants	BusObj クラスの Java 定数の処理に関する関数ブロック。	132 ページの表 34
General/APIs/Business Object Array	ビジネス・オブジェクト配列の処理に関する関数ブロック。	133 ページの表 35
General/APIs/Database Connection	データベース接続の作成および保守に関する関数ブロック。	134 ページの表 36
General/APIs/Identity Relationship	一致関係の処理に関する関数ブロック。	136 ページの表 37
General/APIs/Maps	マップ実行に必要なランタイム値の照会および設定に関する関数ブロック。	138 ページの表 38
General/APIs¥Maps/ Constants	関数ブロック定数。	138 ページの表 39
General/APIs/Maps/ Exception	マップの新規例外オブジェクトの作成に関する関数ブロック。	139 ページの表 40
General/APIs/Participant	一致関係における参加者の値の設定および検索に関する関数ブロック。	140 ページの表 41
General/APIs/Participant/ Array	参加者配列の作成および処理に関する関数ブロック。	142 ページの表 42
General/APIs/Participant/ Constants	参加者に使用する関数ブロック定数。	142 ページの表 43
General/APIs/Relationship	関係のランタイム・インスタンスの操作に関する関数ブロック。	143 ページの表 44
General/Date	日付の処理に関する関数ブロック。	145 ページの表 45
General/Date/Formats	さまざまな日付形式の指定に関する関数ブロック。	147 ページの表 46
General/Logging and Tracing	ログ・メッセージおよびトレース・メッセージの処理に関する関数ブロック。	147 ページの表 47
General/Logging and Tracing/Log Error	エラー・メッセージのフォーマット設定に関する関数ブロック。	148 ページの表 48
General/Logging and Tracing/Log Information	情報メッセージのフォーマット設定に関する関数ブロック。	148 ページの表 49
General/Logging and Tracing/Log Warning	警告メッセージのフォーマット設定に関する関数ブロック。	149 ページの表 50
General/Logging and Tracing/Trace	トレース・メッセージのフォーマット設定に関する関数ブロック。	150 ページの表 51

表 31. 関数ブロックの構成 (続き)

関数ブロックのフォルダー	説明	詳細情報の参照先
General/Mapping	指定されたコンテキスト内でのマップ実行に関する関数ブロック。	150 ページの表 52
General/Math	基本的な計算タスクの関数ブロック。	151 ページの表 53
General/Properties	構成プロパティー値の検索に関する関数ブロック。	153 ページの表 54
General/Relationship	一致関係の保守および照会に関する関数ブロック。	153 ページの表 55
General/String	String オブジェクトの操作に関する関数ブロック。	153 ページの表 56
General/Utilities	例外のスローとキャッチ、およびループ、属性移動、条件設定に関する関係ブロック。	156 ページの表 57
General¥Utilities/Vector	Vector オブジェクトの処理に関する関数ブロック。	157 ページの表 58

以下の表に、各カテゴリーの関数ブロックと、それらの入出力として受け入れられる値を示します。

注: WebSphere Business Integration Express Plus では、Activity Editor は、Process Designer Express とともに使用することもできます。Process Designer Express でコラボレーション関数ブロックなどの関数ブロックを使用する方法については、「*コラボレーション開発ガイド*」を参照してください。

表 32. General/APIs/Business Object

名前	説明	入力、出力、および受け入れ可能な値
Copy	指定のコピー元ビジネス・オブジェクトから、すべての属性値をコピーします。 API: BusObj.copy()	入力: • copy to: BusObj • copy from: BusObj
Duplicate	元のビジネス・オブジェクトとまったく同じビジネス・オブジェクトを作成します。 API: BusObj.duplicate()	入力:original: BusObj 出力: duplicate: BusObj
Equal Keys	business object 1 の値と business object 2 の値を比較し、等しいかどうかを判別します。 API: BusObj.equalKeys()	入力: • business object 1: BusObj • business object 2: BusObj 出力: key values equal?: boolean
Equals	business object 1 の値と business object 2 の値を、子ビジネス・オブジェクトを含めて比較し、等しいかどうかを判別します。 API: BusObj.equals()	入力: • business object 1: BusObj • business object 2: BusObj 出力: equal?: boolean

表 32. General/APIs/Business Object (続き)

名前	説明	入力、出力、および受け入れ可能な値
Exists	指定の名前のビジネス・オブジェクト属性が存在するかどうかを調べます。 API: BusObj.exists()	入力: • business object: BusObj • attribute: String 出力: exists?: boolean
Get Boolean	ビジネス・オブジェクトから 1 つの属性の値を boolean 値として取得します。 API: BusObj.getBoolean()	入力: • business object: BusObj • attribute: String 出力: value: boolean
Get Business Object	ビジネス・オブジェクトから 1 つの属性の値をビジネス・オブジェクトとして取得します。 API: BusObj.getBusObj()	入力: • business object: BusObj • attribute: String 出力: value: BusObj
Get Business Object Array	ビジネス・オブジェクトから 1 つの属性の値をビジネス・オブジェクト配列として取得します。 API: BusObj.getBusObjArray()	入力: • business object: BusObj • attribute: String 出力: value: BusObjArray
Get Business Object Type	ビジネス・オブジェクトの基になったビジネス・オブジェクト定義の名前を取得します。 API: BusObj.getType()	入力: business object: BusObj 出力: type: String
Get Double	ビジネス・オブジェクトから 1 つの属性の値を double 値として取得します。 API: BusObj.getDouble()	入力: • business object: BusObj • attribute: String 出力: value: double
Get Float	ビジネス・オブジェクトから 1 つの属性の値を float 値として取得します。 API: BusObj.getFloat()	入力: • business object: BusObj • attribute: String 出力: value: float
Get Int	ビジネス・オブジェクトから 1 つの属性の値を int 値として取得します。 API: BusObj.getInt()	入力: • business object: BusObj • attribute: String 出力: value: int
Get Long	ビジネス・オブジェクトから 1 つの属性の値を long 値として取得します。 API: BusObj.getLong()	入力: • business object: BusObj • attribute: String 出力: value: long

表 32. General/APIs/Business Object (続き)

名前	説明	入力、出力、および受け入れ可能な値
Get Long Text	ビジネス・オブジェクトから 1 つの属性の値を longtext 値として取得します。 API: BusObj.getLongText()	入力: • business object: BusObj • attribute: String 出力: value: String
Get Object	ビジネス・オブジェクトから 1 つの属性の値をオブジェクトとして取得します。その属性は、属性名または属性位置として指定できます。 API: BusObj.get()	入力: • business object: BusObj • attribute: String、int 出力: value: Object
Get String	ビジネス・オブジェクトから 1 つの属性の値を string 値として取得します。 API: BusObj.getString()	入力: • business object: BusObj • attribute: String 出力: value: String
Get Verb	ビジネス・オブジェクトの動詞を取得します。 API: BusObj.getVerb()	入力: business object: BusObj 出力: verb: String
Is Blank	属性の値がゼロ長ストリングに設定されているかどうかを調べます。 API: BusObj.isBlank()	入力: • business object: BusObj • attribute: String 出力: blank?: boolean
Is Business Object	値がビジネス・オブジェクト (BusObj) であるかどうかを調べます。	入力: value: Object 出力: result: boolean
Is Key	ビジネス・オブジェクトの属性がキー属性として定義されているかどうかを調べます。 API: BusObj.isKey()	入力: • business object: BusObj • attribute: String 出力: key?: boolean
Is Null	ビジネス・オブジェクトの属性の値が null かどうかを調べます。 API: BusObj.isNull()	入力: • business object: BusObj • attribute: String 出力: null?: boolean
Is Required	ビジネス・オブジェクトの属性が必須属性として定義されているかどうかを調べます。 API: BusObj.isRequired()	入力: • business object: BusObj • attribute: String 出力: required?: boolean

表 32. General/APIs/Business Object (続き)

名前	説明	入力、出力、および受け入れ可能な値
Iterate Children	子ビジネス・オブジェクト配列を繰り返し処理します。	入力: • business object: BusObj • attribute: String • current index: int • current element: BusObj
Key to String	ビジネス・オブジェクトの基本キー属性の値をストリングとして取得します。 API: BusObj.keysToString()	入力: business object: BusObj 出力: key string: String
New Business Object	指定のタイプのビジネス・オブジェクトのインスタンス (BusObj) を新規作成します。 API: Collaboration.BusObj()	入力: type: String 出力: business object: BusObj
Set Content	ビジネス・オブジェクトに、別のビジネス・オブジェクトの内容を設定します。これらの 2 つのビジネス・オブジェクトは、内容を共有します。一方のビジネス・オブジェクトに加えられた変更は、もう一方のビジネス・オブジェクトにも反映されます。 API: BusObj.setContent()	入力: • business object: BusObj • content: BusObj
Set Default Attribute Values	すべての属性にそれぞれのデフォルト値を設定します。 API: BusObj.setDefaultAttrValues()	入力: business object: BusObj
Set Keys	「to business object」のビジネス・オブジェクトのキー属性に、「from business object」のビジネス・オブジェクトのキー属性の値を設定します。 API: BusObj.setKeys()	入力: • from business object: BusObj • to business object: BusObj
Set Value with Create	ビジネス・オブジェクトの属性に、特定のデータ型の値を設定します。値の設定先となるオブジェクトが存在しない場合は、そのオブジェクトを作成します。 API: BusObj.setWithCreate()	入力: • business object: BusObj • attribute: String • value: BusObj、BusObjArray、Object
Set Verb	ビジネス・オブジェクトの動詞を設定します。 API: BusObj.setVerb()	入力: • business object: BusObj • verb: String
Set Verb with Create	子ビジネス・オブジェクトの動詞を設定します。子ビジネス・オブジェクトが存在しない場合は、作成します。 API: BusObj.setVerbWithCreate()	入力: • business object: BusObj • attribute: String • verb: String

表 32. General/APIs/Business Object (続き)

名前	説明	入力、出力、および受け入れ可能な値
Set Value	ビジネス・オブジェクトの属性に、特定のデータ型の値を設定します。 API: BusObj.set()	入力: • business object: BusObj • attribute: String • value: boolean、double、float、int、long、Object、String、BusObj
Shallow Equals	business object 1 の値と business object 2 の値を、子ビジネス・オブジェクトを除いて比較し、等しいかどうかを判別します。 API: BusObj.equalsShallow()	入力: • business object 1: BusObj • business object 2: BusObj 出力: equal?: boolean
To String	ビジネス・オブジェクト内のすべての属性の値を 1 つのストリングとして取得します。 API: BusObj.toString()	入力: business object: BusObj 出力: string: String
Valid Data	指定した値が指定した属性に対して有効な型かどうかを検査します。 API: BusObj.validData()	入力: • business object: BusObj • attribute: String • value: Object、BusObj、BusObjArray、String、long、int、double、float、boolean 出力: valid?: boolean

表 33. General/APIs/Business Object/Array

名前	説明	入力、出力、および受け入れ可能な値
Get BusObj At	ビジネス・オブジェクト配列内の、指定のインデックスの位置にあるエレメントを取得します。	入力: • array: BusObj[] • index: int 出力: business object: BusObj
New Business Object Array	新しいビジネス・オブジェクト配列を作成します。	入力: size: int 出力: array: BusObj[]
Set BusObj At	ビジネス・オブジェクト配列内の、指定のインデックスの位置にあるエレメントを設定します。	入力: • array: BusObj[] • index: int • business object: BusObj
Size	ビジネス・オブジェクト配列のサイズを取得します。	入力: array: BusObj[] 出力: size: int

表 34. General/APIs/Business Object/Constants

名前	説明	入力、出力、および受け入れ可能な値
Verb: Create	ビジネス・オブジェクト動詞「Create」。	出力: Create: String
Verb: Delete	ビジネス・オブジェクト動詞「Delete」。	出力: Delete: String

表 34. General/APIs/Business Object/Constants (続き)

名前	説明	入力、出力、および受け入れ可能な値
Verb: Retrieve	ビジネス・オブジェクト動詞「Retrieve」。	出力: Retrieve: String
Verb: Update	ビジネス・オブジェクト動詞「Update」。	出力: Update: String

表 35. General/APIs/Business Object Array

名前	説明	入力、出力、および受け入れ可能な値
Add Element	ビジネス・オブジェクトにビジネス・オブジェクトを追加します。 API: BusObjArray.addElement()	入力: • business object array: BusObjArray • element: BusObj
Duplicate	元のビジネス・オブジェクト配列とまったく同じビジネス・オブジェクト配列を作成します。 API: BusObjArray.duplicate()	入力: original: BusObjArray 出力: duplicate: BusObjArray
Equals	ビジネス・オブジェクト配列 1 (array 1) の値とビジネス・オブジェクト配列 2 (array 2) の値を比較し、等しいかどうかを判別します。 API: BusObjArray.equals()	入力: • array 1: BusObjArray • array 2: BusObjArray 出力: equal?: boolean
Get Element At	いずれか 1 つのビジネス・オブジェクトのビジネス・オブジェクト配列内での位置が指定されると、そのビジネス・オブジェクトを取得します。 API: BusObjArray.elementAt()	入力: • business object array: BusObjArray • index: int 出力: element: BusObj
Get Elements	ビジネス・オブジェクト配列の内容を取得します。 API: BusObjArray.getElements()	入力: business object array: BusObjArray 出力: element: BusObj[]
Get Last Index	ビジネス・オブジェクト配列の最後の有効なインデックスを検索します。 API: BusObjArray.getLast Index()	入力: business object array: BusObjArray 出力: last index: int
Is Business Object Array	値がビジネス・オブジェクト配列 (BusObjArray) であるかどうかを調べます。	入力: value: Object 出力: result: boolean
Max attribute value	ビジネス・オブジェクト配列のすべてのエレメントを対象として、指定の属性の最大値を検索します。 API: BusObjArray.max()	入力: • business object array: BusObjArray • attribute: String 出力: max: String
Min attribute value	ビジネス・オブジェクト配列のすべてのエレメントを対象として、指定の属性の最小値を検索します。 API: BusObjArray.min()	入力: • business object array: BusObjArray • attribute: String 出力: min: String

表 35. General/APIs/Business Object Array (続き)

名前	説明	入力、出力、および受け入れ可能な値
Remove All Elements	ビジネス・オブジェクト配列からすべてのエレメントを除去します。 API: BusObjArray.removeAllElements()	入力: business object array: BusObjArray
Remove Element	ビジネス・オブジェクト配列から 1 つのビジネス・オブジェクト・エレメントを除去します。 API: BusObjArray.removeElement()	入力: • business object array: BusObjArray • element: BusObj
Remove Element At	ビジネス・オブジェクト配列内の特定の位置にあるエレメントを除去します。 API: BusObjArray.removeElementAt()	入力: • business object array: BusObjArray • index: int
Set Element At	ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を設定します。 API: BusObjArray.setElementAt()	入力: • business object array: BusObjArray • index: int • element: BusObj
Size	ビジネス・オブジェクト配列内のエレメントの数を取得します。 API: BusObjArray.size()	入力: business object array: BusObjArray 出力: size: int
Sum	ビジネス・オブジェクト配列の中のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。 API: BusObjArray.sum()	入力: • business object array: BusObjArray • attribute: String 出力: sum: double
Swap	ビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。 API: BusObjArray.swap()	入力: • business object array: BusObjArray • index 1: int • index 2: int
To String	ビジネス・オブジェクト配列内の値を 1 つのストリングとして取得します。 API: BusObjArray.to String()	入力: business object array: BusObjArray 出力: string: String

表 36. General/APIs/Database Connection

名前	説明	入力、出力、および受け入れ可能な値
Begin Transaction	現行接続に対して明示的なトランザクションを開始します。 API: CwDBConnection.beginTransaction()	入力: database connection: CwDBConnection
Commit	現行接続に関連付けられたアクティブなトランザクションをコミットします。 API: CwDBConnection.commit()	入力: database connection: CwDBConnection

表 36. General/APIs/Database Connection (続き)

名前	説明	入力、出力、および受け入れ可能な値
Execute Prepared SQL	指定の SQL 構文を使用して、準備済み SQL 照会を実行します。 API: CwDBConnection.executePreparedSQL()	入力: • database connection: CwDBConnection • query: String 出力: equal?: boolean
Execute Prepared SQL with Parameter	指定の SQL 構文とパラメーターを使用して、準備済み SQL 照会を実行します。 API:CwDBConnection.executePreparedSQL()	入力: • database connection: CwDBConnection • query: String • parameters: java.util.Vector
Execute SQL	指定の SQL 構文を使用して、静的 SQL 照会を実行します。 API: CwDBConnection.executeSQL()	入力: • database connection: CwDBConnection • query: String
Execute SQL with Parameter	指定の SQL 構文とパラメーターを使用して、静的 SQL 照会を実行します。 API: CwDBConnection.executeSQL()	入力: • database connection: CwDBConnection • query: String • parameters: java.util.Vector
Execute Stored Procedure	指定の名前とパラメーター配列を使用して、SQL ストアード・プロシージャーを実行します。 API: CwDBConnection.executeStoredProcedure()	入力: • database connection: CwDBConnection • query: String • parameters: java.util.Vector
Get Database Connection	データベースへの接続を確立して、CwDBConnection() オブジェクトを戻します。 API: BaseDLM.getDBConnection() または BaseCollaboration.getDBConnection()	入力: connection pool name: String 出力: database connection: CwDBConnection
Get Database Connection with Transaction	データベースへの接続を確立して、CwDBConnection() オブジェクトを戻します。 API: BaseDLM.getDBConnection() または BaseCollaboration.getDBConnection()	入力: • connection pool name: String • implicit transaction: boolean 出力: database connection: CwDBConnection
Get Next Row	照会結果から次の行を取得します。 API: CwDBConnection.nextRow()	入力: database connection: CwDBConnection 出力: row: java.util.Vector
Get Update Count	データベースへの最後の書き込み操作によって影響を受けた行の数を取得します。 API: CwDBConnection.getUpdateCount()	入力: database connection: CwDBConnection 出力: count: int
Has More Rows	照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。 API: CwDBConnection.hasMoreRows()	入力: database connection: CwDBConnection 出力: more rows?: boolean
In Transaction	現行接続でトランザクションが進行中かどうかを判別します。 API: CwDBConnection.inTransaction()	入力: database connection: CwDBConnection 出力: in transaction?: boolean

表 36. General/APIs/Database Connection (続き)

名前	説明	入力、出力、および受け入れ可能な値
Is Active	現行接続がアクティブかどうかを判別します。 API: CwDBConnection.isActive()	入力: database connection: CwDBConnection 出力: is active?: boolean
Release	現行接続の使用を解放して、接続プールに戻します。 API: CwDBConnection.release()	入力: database connection: CwDBConnection
Roll Back	現行接続に関連付けられたアクティブなトランザクションをロールバックします。 API: CwDBConnection.rollback()	入力: database connection: CwDBConnection

表 37. General/APIs/Identity Relationship

名前	説明	入力、出力、および受け入れ可能な値
Add My Children	指定した子インスタンスを、一致関係の親/子関係に追加します。 API: IdentityRelationship.addMyChildren()	入力: • map: BaseDLM • parentChildRelDefName: String • parentParticipantDefName: String • parentBusObj: BusObj • childParticipantDefName: String • childBusObjList: BusObj、BusObjArray
Delete All My Children	指定の親に属する一致関係の親/子関係から、すべての子インスタンスを除去します。 API: IdentityRelationship.deleteMyChildren()	入力: • map: BaseDLM • parentChildRelDefName: String • parentParticipantDefName: String • parentBusObj: BusObj • childParticipantDefName: String
Delete My Children	指定の親に属する一致関係の親/子関係から、指定の子インスタンスを除去します。 API: IdentityRelationship.deleteMyChildren()	入力: • map: BaseDLM • parentChildRelDefName: String • parentParticipantDefName: String • parentBusObj: BusObj • childParticipantDefName: String • childBusObjList: BusObj、BusObjArray

表 37. General/APIs/Identity Relationship (続き)

名前	説明	入力、出力、および受け入れ可能な値
Foreign Key Cross-Reference	ソース・ビジネス・オブジェクトの外部キーに基づいて、リレーションシップ・データベースの関係表でルックアップを実行します。外部キーが存在しない場合は、外部関係表で新しい関係インスタンスを追加します。 API: IdentityRelationship.foreignKeyXref()	入力: • map: BaseDLM • RelDefName: String • appParticipantDefName: String • genParticipantDefName: String • appSpecificBusObj: BusObj • appForeignAttr: String • genericBusObj: BusObj • genForeignAttr: String
Foreign Key Lookup	ソース・ビジネス・オブジェクトの外部キーに基づいて、外部関係表でルックアップを実行します。外部関係表に外部キーが存在しない場合、関係インスタンスの検索は失敗します。 API: IdentityRelationship.foreignKeyLookup()	入力: • map: BaseDLM • relDefName: String • appParticipantDefName: String • appSpecificBusObj: BusObj • appForeignAttr: String • genericBusObj: BusObj • genForeignAttr: String
Maintain Child Verb	マップの実行コンテキストと、親ビジネス・オブジェクトの動詞に基づいて、子ビジネス・オブジェクトの動詞を設定します。 API: IdentityRelationship.maintainChildVerb()	入力: • map: BaseDLM • relDefName: String • appSpecificParticipantName: String • genericParticipantName: String • appSpecificObj: BusObj • appSpecificChildObj: String • genericObj: BusObj • genericChildObj: String • to_Retrieve: boolean • Is_Composite: boolean
Maintain Composite Relationship	親マップ内から複合一致関係を保守します。 API: IdentityRelationship.maintain CompositeRelationship()	入力: • map: BaseDLM • relDefName: String • participantDefName: String • appSpecificBusObj: BusObj • genericBusObjList: BusObj、BusObjArray
Maintain Simple Identity Relationship	親マップまたは子マップ内から単純一致関係を保守します。 API: IdentityRelationship.maintain SimpleIdentityRelationship()	入力: • map: BaseDLM • relDefName: String • participantDefName: String • appSpecificBusObj: BusObj • genericBusObj: BusObj

表 37. General/APIs/Identity Relationship (続き)

名前	説明	入力、出力、および受け入れ可能な値
Update My Children	必要に応じて、一致関係の指定された親/子関係で子インスタンスを追加または削除します。 API: IdentityRelationship.updateMyChildren()	入力: • map: BaseDLM • parentChildRelDefName: String • parentParticipantDef: String • parentBusObj: BusObj • childParticipantDef: String • childAttrName: String • childIdentityRelDefName: String • childIdentityParticipantDefName: String

表 38. General/APIs/Maps

名前	説明	入力、出力、および受け入れ可能な値
Get Adapter Name	現行マップ・インスタンスに関連付けられているアダプター名を取得します。 API: MapExeContext.getConnName()	入力: map: BaseDLM 出力: adapter name: String
Get Calling Context	現行マップ・インスタンスに関連付けられた呼び出しコンテキストを検索します。 API: MapExeContext.getInitiator()	入力: map: BaseDLM 出力: calling context: String
Get Original Request Business Object	現行マップ・インスタンスに関連付けられたオリジナル要求ビジネス・オブジェクトを検索します。 API: MapExeContext.getOriginalRequestBO()	入力: map: BaseDLM 出力: original business object: BusObj

表 39. General/APIs/Maps/Constants

名前	説明	入力、出力、および受け入れ可能な値
Calling Context: ACCESS _REQUEST	アクセス・クライアントにより外部アプリケーションから InterChange Server Express にアクセス要求が送信されました。 API: MapExeContext.ACCESS_REQUEST	出力: ACCESS_REQUEST: String
Calling Context: ACCESS _RESPONSE	ソース・ビジネス・オブジェクトは、サブスクリプション送達要求に応答してソース・アクセス・クライアントに送り返されました。 API: MapExeContext.ACCESS_RESPONSE	出力: ACCESS_RESPONSE: String
Calling Context: EVENT _DELIVERY	コネクターによりアプリケーションから InterChange Server Express にイベントが送信されました (イベントにより起動されたフロー)。 API: MapExeContext.EVENT_DELIVERY	出力: EVENT_DELIVERY: String

表 39. General/APIs/Maps/Constants (続き)

名前	説明	入力、出力、および受け入れ可能な値
Calling Context: SERVICE_CALL _FAILURE	コラボレーションのサービス呼び出し要求は失敗しました。したがって、訂正アクションを実行する必要がある場合があります。 API: MapExeContext.SERVICE_CALL_FAILURE	出力: SERVICE_CALL_FAILURE: String
Calling Context: SERVICE_CALL _REQUEST	コラボレーションにより、ビジネス・オブジェクトがサービス呼び出し要求を経由してアプリケーションに送信中です。 API: MapExeContext.SERVICE_CALL_REQUEST	出力: SERVICE_CALL_REQUEST: String
Calling Context: SERVICE_CALL _RESPONSE	コラボレーションのサービス呼び出し要求の正常な応答結果として、アプリケーションからビジネス・オブジェクトが受信されました。 API: MapExeContext.SERVICE_CALL _RESPONSE	出力: SERVICE_CALL_RESPONSE: String

表 40. General/APIs/Maps/Exception

名前	説明	入力、出力、および受け入れ可能な値
Raise Map Exception	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String
Raise Map Exception 1	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String • parameter 1: String
Raise Map Exception 2	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String • parameter 1: String • parameter 2: String
Raise Map Exception 3	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String • parameter 1: String • parameter 2: String • parameter 3: String

表 40. General/APIs/Maps/Exception (続き)

名前	説明	入力、出力、および受け入れ可能な値
Raise Map Exception 4	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String • parameter 1: String • parameter 2: String • parameter 3: String • parameter 4: String
Raise Map Exception 5	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception type: String • message: String • parameter 1: String • parameter 2: String • parameter 3: String • parameter 4: String • parameter 5: String
Raise Map RunTimeEntity Exception	マップの実行時例外を発生させます。 API: raiseException()	入力: • map: BaseDLM • exception: RunTimeEntityException

表 41. General/APIs/Participant

名前	説明	入力、出力、および受け入れ可能な値
Get Boolean Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getBoolean()	入力: participant: Server.RelationshipServices.Participant 出力: data: boolean
Get Business Object Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getBusObj()	入力: participant: Server.RelationshipServices.Participant 出力: data: BusObj
Get Double Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getDouble()	入力: participant: Server.RelationshipServices.Participant 出力: data: double
Get Float Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getFloat()	入力: participant: Server.RelationshipServices.Participant 出力: data: float
Get Instance ID	参加者インスタンスが参加している関係の関係インスタンス ID を取得します。 API: Participant.getInstanceId()	入力: participant: Server.RelationshipServices.Participant 出力: instance ID: int

表 41. General/APIs/Participant (続き)

名前	説明	入力、出力、および受け入れ可能な値
Get Int Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getInt()	入力: participant: Server.RelationshipServices.Participant 出力: data: int
Get Long Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getLong()	入力: participant: Server.RelationshipServices.Participant 出力: data: long
Get Participant Name	参加者インスタンスの作成に使用された参加者定義の名前を取得します。 API: Participant.getParticipantDefinition()	入力: participant: Server.RelationshipServices.Participant 出力: name: String
Get Relationship Name	参加者インスタンスが参加している関係定義の名前を取得します。 API: Participant.getRelationshipDefinition()	入力: participant: Server.RelationshipServices.Participant 出力: name: String
Get String Data	参加者オブジェクトに関連付けられているデータを取得します。 API: Participant.getString()	入力: participant: Server.RelationshipServices.Participant 出力: data: String
New Participant	関係インスタンスのない新しい参加者インスタンスを作成します。 API: Participant()	入力: • relDefName: String • partDefName: String • partData: BusObj、String、long、int、double、float、boolean 出力: participant: Server.RelationshipServices.Participant
New Participant in Relationship	関係インスタンスの既存の参加者に加える新しい参加者インスタンスを作成します。 API: RelationshipServices.Participant()	入力: • relDefName: String • partDefName: String • instanceId: int • partData: BusObj、String、long、int、double、float、boolean 出力: participant: Server.RelationshipServices.Participant
Set Data	参加者インスタンスに関連付けられたデータを設定します。 API: Participant.set()	入力: • participant: Server.RelationshipServices.Participant • partData: BusObj、String、long、int、double、float、boolean
Set Instance ID	参加者インスタンスが参加している関係のインスタンス ID を設定します。 API: Participant.setInstanceId()	入力: • participant: Server.RelationshipServices.Participant • id: int

表 41. General/APIs/Participant（続き）

名前	説明	入力、出力、および受け入れ可能な値
Set Participant Definition	参加者インスタンスの作成に使用された参加者定義の名前を設定します。 API: Participant.setParticipantDefinition()	入力: • participant: Server.RelationshipServices.Participant • partDefName: String
Set Relationship Definition	参加者インスタンスが参加している関係定義を設定します。 API: Participant.setRelationshipDefinition()	入力: • participant: Server.RelationshipServices.Participant • relDefName: String

表 42. General/APIs/Participant/Array

名前	説明	入力、出力、および受け入れ可能な値
Get Participant At	参加者配列内の、指定のインデックスの位置にあるエレメントを取得します。	入力: • array: Server.RelationshipServices.Participant[] • index: int 出力: participant: Server.RelationshipServices.Participant
New Participant Array	指定のサイズの参加者配列を新規作成します。	入力: size: int 出力: array: Server.RelationshipServices.Participant[]
Set Participant At	参加者配列内の、指定のインデックスの位置にあるエレメントを設定します。	入力: • array: Server.RelationshipServices.Participant[] • index: int • participant: Server.RelationshipServices.Participant
Size	参加者配列のサイズを取得します。	入力: array: Server.RelationshipServices.Participant[] 出力: size: int

表 43. General/APIs/Participant/Constants

名前	説明	入力、出力、および受け入れ可能な値
Participant: INVALID _INSTANCE_ID	参加者 ID が無効であることを示す参加者定数。 API: Participant.INVALID_INSTANCE_ID	出力: INVALID_INSTANCE_ID: int

表 44. General/APIs/Relationship

名前	説明	入力、出力、および受け入れ可能な値
Add Participant	関係インスタンスに既存の参加者オブジェクトを追加します。 API: Relationship.addParticipant()	入力: participant: Server.RelationshipServices.Participant 出力: result instance ID: int
Add Participant Data	既存の関係インスタンスに新しい参加者を追加します。 API: Relationship.addParticipant()	入力: • relDefName: String • partDefName: String • instanceId: int • partData: BusObj、String、long、int、double、float、boolean 出力: result instance ID: int
Add Participant Data to New Relationship	新しい関係インスタンスに参加者を追加します。 API: Relationship.addParticipant()	入力: • relDefName: String • partDefName: String • partData: BusObj、String、long、int、double、float、boolean 出力: result instance ID: int
Create Relationship	新しい関係インスタンスを作成します。 API: Relationship.create()	入力: • relDefName: String • partDefName: String • partData: BusObj、String、long、int、double、float、boolean 出力: instance ID: int
Create Relationship with Participant	新しい関係インスタンスを作成します。 API: Relationship.create()	入力: participant: Server.RelationshipServices.Participant 出力: instance ID: int
Deactivate Participant	1 つ以上の関係インスタンスから参加者を非アクティブにします。 API: Relationship.deactivateParticipant()	入力: participant: Server.RelationshipServices.Participant
Deactivate Participant By Data	1 つ以上の関係インスタンスから参加者を非アクティブにします。 API: Relationship.deactivateParticipant()	入力: • relDefName: String • partDefName: String • partData: BusObj、String、long、int、double、float、boolean
Deactivate Participant By Instance	特定の関係インスタンスから参加者を非アクティブにします。 API: Relationship.deactivateParticipantByInstance()	入力: • relDefName: String • partDefName: String • instanceId: int

表 44. General/APIs/Relationship (続き)

名前	説明	入力、出力、および受け入れ可能な値
Deactivate Participant By Instance Data	参加者に関連付けられているデータを使用して、特定の関係インスタンスから参加者を非アクティブにします。 API: Relationship.deactivateParticipantByInstance()	入力: • relDefName: String • partDefName: String • instanceId: int • partData: BusObj、String、long、int、double、float、boolean
Delete Participant	1 つ以上の関係インスタンスから参加者インスタンスを除去します。 API: Relationship.deleteParticipant()	入力: participant: Server.RelationshipServices.Participant
Delete Participant By Instance	特定の関係インスタンスから参加者を除去します。 API: Relationship.deleteParticipanByInstancet()	入力: • relDefName: String • partDefName: String • instanceId: int
Delete Participant By Instance Data	参加者に関連付けられているデータを使用して、特定の関係インスタンスから参加者を除去します。 API: Relationship.deleteParticipanByInstancet()	入力: • relDefName: String • partDefName: String • instanceId: int • partData: BusObj、String、long、int、double、float、boolean
Delete Participant with Data	1 つ以上の関係インスタンスから参加者インスタンスを除去します。 API: Relationship.deleteParticipant()	入力: • relDefName: String • partDefName: String • partData: BusObj、String、long、int、double、float、boolean
Get Next Instance ID	関係定義名に基づいて、次に使用可能な関係の関係インスタンス ID を戻します。 API: Relationship.getNewID()	入力: relDefName: String 出力: ID: int
Retrieve Instances	特定の参加者 (複数指定も可) を含む関係インスタンスがあれば、その ID をすべて取得します。 API: Relationship.retrieveInstances()	入力: • relDefName: String • partDefName: String、String[] • partData: BusObj、String、long、int、double、float、boolean 出力: instance IDs: int
Retrieve Instances for Participant	特定の単一の参加者を含む関係インスタンスがあれば、その ID をすべて取得します。 API: Relationship.retrieveInstances()	入力: • relDefName: String • partData: BusObj、String、long、int、double、float、boolean 出力: instance IDs: int

表 44. General/APIs/Relationship (続き)

名前	説明	入力、出力、および受け入れ可能な値
Retrieve Participants	関係インスタンスからゼロまたは複数の参加者を検索します。 API: Relationship.retrieveParticipants()	入力: • relDefName: String • partDefName: String、String[] • instanceId: int 出力: participant instances: Server.RelationshipServices.Participant[]
Retrieve Participants with ID	関係インスタンスからゼロまたは複数の参加者を検索します。 API: Relationship.retrieveParticipants()	入力: • relDefName: String • instanceId: int 出力: participant instances: Server.RelationshipServices.Participant[]
Update Participant	1 つ以上の関係インスタンスの参加者を更新します。 API: Relationship.updateParticipant()	入力: • relDefName: String • partDefName: String • partData: BusObj
Update Participant By Instance	特定の関係インスタンスの参加者を更新します。 API: Relationship.updateParticipantByInstance()	入力: • relDefName: String • partDefName: String、String[] • instanceId: int
Update Participant By Instance Data	特定の関係インスタンスの参加者を、その参加者に関連付けるべきデータを使って更新します。 API: Relationship.updateParticipantByInstance()	入力: • relDefName: String • partDefName: String • instanceId: int • partData: BusObj、String

表 45. General/Date

名前	説明	入力、出力、および受け入れ可能な値
Add Day	from date の日付の日数を増加させます。	入力: • from date: String • date format: String • day to add: int 出力: to date: String
Add Month	from date の日付の月数を増加させます。	入力: • from date: String • date format: String • month to add: int 出力: to date: String

表 45. General/Date (続き)

名前	説明	入力、出力、および受け入れ可能な値
Add Year	from date の日付の年数を増加させます。	入力: • from date: String • date format: String • year to add: int 出力: to date: String
Date After	2 つの日付を比較し、Date 1 が Date 2 より後かどうかを判別します。	入力: • Date 1: String • Date 1 format: String • Date 2: String • Date 2 format: String 出力: Is Date 1 after Date 2?: boolean
Date Before	2 つの日付を比較し、Date 1 が Date 2 より前かどうかを判別します。	入力: • Date 1: String • Date 1 format: String • Date 2: String • Date 2 format: String 出力: Is Date 1 before Date 2?: boolean
Date Equals	2 つの日付を比較し、等しいかどうかを判別します。	入力: • Date 1: String • Date 1 format: String • Date 2: String • Date 2 format: String 出力: Are they equal?: boolean
Format Change	日付の形式を変更します。	入力: • date: String • input format: String • output format: String 出力: formatted date: String
Get Day	指定の日付が月の何番目の日であるかを、日付形式を基に特定し、数値で戻します。	入力: • Date: String • Format: String 出力: Day: int
Get Month	指定の日付が年の何番目の月にあるかを、日付形式を基に特定し、数値で戻します。	入力: • Date: String • Format: String 出力: Month: int

表 45. General/Date (続き)

名前	説明	入力、出力、および受け入れ可能な値
Get Year	指定の日付の年を、日付形式を基に特定し、数値で戻します。	入力: • Date: String • Format: String 出力: Year: int
Get Year Month Day	入力された日付から、年、月、日の部分をそれぞれ抽出します。	入力: • Date: String • Format: String 出力: • Year: int • Month: int • Day: int
Now	今日の日付を取得します。	入力: format: String 出力: now: String

表 46. General/Date/Formats

名前	説明	入力、出力、および受け入れ可能な値
yyyy-MM-dd	yyyy-MM-dd の日付形式 **例:** 2003-02-25	出力: format: String
yyyyMMdd	yyyyMMdd の日付形式 **例:** 20030225	出力: format: String
yyyyMMdd HH:mm:ss	yyyyMMdd HH:mm:ss の日付形式 **例:** 20030225 12:36:40	出力: format: String

表 47. General/Logging and Tracing

名前	説明	入力、出力、および受け入れ可能な値
Log error	指定のエラー・メッセージを ICS Express ログ・ファイルに送信します。	入力: message: String, byte, short, int, long, float, double
Log error ID	指定の ID に関連付けられているエラー・メッセージを ICS Express ログ・ファイルに送信します。	入力: ID: String, byte, short, int, long, float, double
Log information	指定の情報メッセージを ICS Express ログ・ファイルに送信します。	入力: message: String, byte, short, int, long, float, double
Log information ID	指定の ID に関連付けられている情報メッセージを ICS Express ログ・ファイルに送信します。	入力: ID: String, byte, short, int, long, float, double
Log warning	指定の警告メッセージを ICS Express ログ・ファイルに送信します。	入力: message: String, byte, short, int, long, float, double

表 47. General/Logging and Tracing (続き)

名前	説明	入力、出力、および受け入れ可能な値
Log warning ID	指定の ID に関連付けられている警告メッセージを ICS Express ログ・ファイルに送信します。	入力: ID: String, byte, short, int, long, float, double
Trace	指定のトレース・メッセージを ICS Express ログ・ファイルに送信します。	入力: message: String, byte, short, int, long, float, double

表 48. General/Logging and Tracing/Log Error

名前	説明	入力、出力、および受け入れ可能な値
Log error ID 1	指定の ID に関連付けられているエラー・メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter: String, byte, short, int, long, float, double
Log error ID 2	指定の ID に関連付けられているエラー・メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double
Log error ID 3	指定の ID に関連付けられているエラー・メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double • parameter 3: String, byte, short, int, long, float, double

表 49. General/Logging and Tracing/Log Information

名前	説明	入力、出力、および受け入れ可能な値
Log information ID 1	指定の ID に関連付けられている情報メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter: String, byte, short, int, long, float, double

表 49. General/Logging and Tracing/Log Information (続き)

名前	説明	入力、出力、および受け入れ可能な値
Log information ID 2	指定の ID に関連付けられている情報メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double
Log information ID 3	指定の ID に関連付けられている情報メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double • parameter 3: String, byte, short, int, long, float, double

表 50. General/Logging and Tracing/Log Warning

名前	説明	入力、出力、および受け入れ可能な値
Log warning ID 1	指定の ID に関連付けられている警告メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter: String, byte, short, int, long, float, double
Log warning ID 2	指定の ID に関連付けられている警告メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double
Log warning ID 3	指定の ID に関連付けられている警告メッセージに、パラメーターとして指定された値を組み込んでフォーマット設定し、ICS Express ログ・ファイルに送信します。	入力: • ID: String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double • parameter 3: String, byte, short, int, long, float, double

表 51. General/Logging and Tracing/Trace

名前	説明	入力、出力、および受け入れ可能な値
Trace ID 1	トレースが指定のレベル以上のレベルに設定されていれば、指定の ID に関連付けられているトレース・メッセージを、パラメーターとして指定された値を組み込んで表示します。	入力: • ID: String, byte, short, int, long, float, double • level:String, byte, short, int, long, float, double • parameter: String, byte, short, int, long, float, double
Trace ID 2	トレースが指定のレベル以上のレベルに設定されていれば、指定の ID に関連付けられているトレース・メッセージを、パラメーターとして指定された値を組み込んで表示します。	入力: • ID: String, byte, short, int, long, float, double • level:String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double
Trace ID 3	トレースが指定のレベル以上のレベルに設定されていれば、指定の ID に関連付けられているトレース・メッセージを、パラメーターとして指定された値を組み込んで表示します。	入力: • ID: String, byte, short, int, long, float, double • level:String, byte, short, int, long, float, double • parameter 1: String, byte, short, int, long, float, double • parameter 2: String, byte, short, int, long, float, double • parameter 3: String, byte, short, int, long, float, double
Trace on Level	トレースが指定のレベル以上のレベルに設定されていれば、トレース・メッセージを表示します。	入力: • message: String, byte, short, int, long, float, double • level:String, byte, short, int, long, float, double

表 52. General/Mapping

名前	説明	入力、出力、および受け入れ可能な値
Run Map	現在の呼び出しコンテキストで指定のマップを実行します。	入力: • Map Name: String • Source Business Objects: BusObj、BusObj[] 出力: Map Results: BusObj、BusObj[]

表 52. General/Mapping（続き）

名前	説明	入力、出力、および受け入れ可能な値
Run Map with Context	指定の呼び出しコンテキストで指定のマップを実行します。	入力: • Map Name: String • Source Business Objects: BusObj、BusObj[] • calling context: String 出力: Map Results: BusObj、BusObj[]

表 53. General/Math

名前	説明	入力、出力、および受け入れ可能な値
Absolute value	a=abs(b) API: Math.abs()	入力: b: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Ceiling	指定の数値表現と等しいかそれよりも大きい整数のうち、最小のものを戻します。	入力: number: String, float, double 出力: ceiling: int
Divide	a=b/c	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Equal	value 1 が value 2 と等しいかどうかを調べます。	入力: • value 1: String, byte, short, int, long, float, double • value 2: String, byte, short, int, long, float, double 出力: are they equal?: boolean
Floor	指定の数値表現と等しいかそれよりも小さい整数のうち、最大のものを戻します。	入力: number: String, float, double 出力: floor: int
Greater than	value 1 が value 2 よりも大きいかどうかを調べます。	入力: • value 1: byte, short, int, long, float, double • value 2: byte, short, int, long, float, double 出力: result: boolean
Greater than or Equal	value 1 が value 2 以上であるかどうかを調べます。	入力: • value 1: byte, short, int, long, float, double • value 2: byte, short, int, long, float, double 出力: result: boolean
Less than	result = value 1 が value 2 より小さいかどうか。	入力: • value 1: byte, short, int, long, float, double • value 2: byte, short, int, long, float, double 出力: result: boolean

表 53. General/Math (続き)

名前	説明	入力、出力、および受け入れ可能な値
Less than or equal	value 1 が value 2 以下であるかどうかを調べます。	入力: • value 1: byte, short, int, long, float, double • value 2: byte, short, int, long, float, double 出力: result: boolean
Maximum	a=max(b, c) API: Math.max()	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Minimum	a=min(b, c) API: Math.min()	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Minus	a=b-c	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Multiply	a=b*c	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double
Not Equal	result = value 1 が value 2 と等しくないかどうか。	入力: • value 1: String, byte, short, int, long, float, double • value 2: String, byte, short, int, long, float, double 出力: are they not equal?: boolean
Not a Number	入力データが数値ではない場合、true を戻します。	入力: input: String 出力: is not a number: boolean
Number to String	数値表現を文字表現に変換します。	入力: number: String、short、int、long、float、double 出力: string: String
Plus	a=b+c	入力: • b: byte, short, int, long, float, double • c: byte, short, int, long, float, double 出力: a: byte, short, int, long, float, double

表 53. *General/Math (続き)*

名前	説明	入力、出力、および受け入れ可能な値
Round	数値表現が 5 未満の場合は、小数部を切り捨ててその数値表現を整数にします。それ以外の場合は小数部を切り上げて整数にします。	入力: number: String, float, double 出力: rounded number: int
String to Number	文字表現を数値表現に変換します。 API: Math.type()	入力: string: String 出力: String、short、int、long、float、double

表 54. *General/Properties*

名前	説明	入力、出力、および受け入れ可能な値
Get Property	指定の構成プロパティーの値を取得します。	入力: property name: String 出力: property value: String

表 55. *General/Relationship*

名前	説明	入力、出力、および受け入れ可能な値
Maintain Identity Relationship	maintainSimpleIdentityRelationship() Relationship API を使用して、一致関係を保守します。	入力: • relationship name: String • participant name: String • Generic Business Object: String • Application-Specific Business Object: String • calling context: String
Static Lookup	関係内で静的な値を検索します。	入力: • relationship name: String • participant name: String • inbound?: boolean • source value: String 出力: lookup value: String

表 56. *General/String*

名前	説明	入力、出力、および受け入れ可能な値
Append Text	ストリング「in string 1」の末尾にストリング「in string2」を付加します。	入力: • in string 1: String • in string 2: String 出力: result: String
If	条件が true の場合は第 1 の値を戻し、false の場合は第 2 の値を戻します。	入力: • condition: boolean、Boolean • value 1: String • value 2: String 出力: result: String

表 56. General/String (続き)

名前	説明	入力、出力、および受け入れ可能な値
Is Empty	第 1 の値が空の場合、第 2 の値を戻します。	入力: • value 1: String • value 2: String 出力: result: String
Is NULL	第 1 の値が null の場合、第 2 の値を戻します。	入力: • value 1: String • value 2: String 出力: result: String
Left Fill	指定の長さのストリングを戻します。長さが足りない場合は、元のストリングの左側を指定の値で埋めます。	入力: • string: String • fill string: String • length: int 出力: filled string: String
Left String	ストリングの左端から順に、指定の桁数分の文字を取り出して戻します。	入力: • string: String • length: int 出力: left string: String
Lower Case	すべての文字を小文字に変更します。	入力: fromString: String 出力: toString: String
Object To String	オブジェクトをストリングで表現したものを取得します。	入力: object: Object 出力: string: String
Repeat	指定の文字表現が指定の回数だけ繰り返されている文字ストリングを戻します。	入力: • repeating string: String • repeat count: int 出力: result: String
Replace	ストリングの一部を、指定の値データで置換します。	入力: • input: String • old string: String • new string: String 出力: replaced string: String
Right Fill	指定の長さのストリングを戻します。長さが足りない場合は、元のストリングの右側を指定の値で埋めます。	入力: • string: String • fill string: String • length: int 出力: filled string: String

表 56. General/String (続き)

名前	説明	入力、出力、および受け入れ可能な値
Right String	ストリングの右端から順に、指定の桁数分の文字を取り出して戻します。	入力: • string: String • length: int 出力: right string: String
Substring by position	開始パラメーターと終了パラメーターの値に基づいて、ストリングの一部を戻します。	入力: • string: String • start position: int • end position: int 出力: substring: String
Substring by value	開始パラメーターと終了パラメーターの値に基づいて、ストリングの一部を戻します。戻されたサブストリングには、指定の開始値と終了値は含まれません。	入力: • string: String • start value: int • end value: int 出力: substring: String
Text Equal	ストリング「inString1」および「inString2」を比較し、同じかどうかを判別します。	入力: • inString1: String • inString2: String 出力: are they equal?: boolean
Text Equal Ignore Case	ストリング「inString1」および「inString2」の大文字と小文字の違いを無視して辞書順に比較します。	入力: • inString1: String • inString2: String 出力: are they equal?: boolean
Text Length	ストリングに含まれる文字の総数を調べます。	入力: str: String 出力: length: byte, short, int, long, float, double
Trim Left	ストリングの左端から順に、指定の数の文字を切り取ります。	入力: • input: String • trim length: int 出力: result: String
Trim Right	ストリングの右端から順に、指定の数の文字を切り取ります。	入力: • input: String • trim length: int 出力: result: String
Trim Text	ストリングの前後の空白文字を切り取ります。	入力: in string: String 出力: trimmed string: String

表 56. General/String (続き)

名前	説明	入力、出力、および受け入れ可能な値
Upper Case	すべての文字を大文字に変更します。	入力: fromString: String 出力: toString: String

表 57. General/Utilities

名前	説明	入力、出力、および受け入れ可能な値
Catch Error	現在のアクティビティーおよびそのサブアクティビティーでスローされたすべての例外をキャッチします。(サブアクティビティーを定義するには、キャンバスの関数ブロック・アイコンをダブルクリックします。)	入力: • Error Name: String • Error Message: String
Catch Error Type	現在のアクティビティーおよびそのサブアクティビティーでスローされた、指定した例外タイプをキャッチします。(サブアクティビティーを定義するには、キャンバスの関数ブロック・アイコンをダブルクリックします。)	入力: • error type: String • Error Message: String
Condition	「Condition」が true の場合は「True Action」に定義されたサブアクティビティーを実行します。それ以外の場合は「False Action」に定義されたサブアクティビティーを実行します。(サブアクティビティーを定義するには、キャンバスの関数ブロック・アイコンをダブルクリックします。)	入力: Condition: boolean
Loop	「条件」が false になるまでサブアクティビティーを繰り返します。(サブアクティビティーを定義するには、キャンバスの関数ブロック・アイコンをダブルクリックします。)	入力: Condition: boolean
Move Attribute in Child	「from attribute」から「to attribute」へ値を移動します。	入力: • source parent: BusObj • source child BO attribute: string • from attribute: String • destination parent: BusObj • destination child BO attribute: String • to attribute: String
Raise Error	指定のメッセージで新しい Java 例外をスローします。	入力: message: String
Raise Error Type	指定のメッセージで指定した Java 例外をスローします。	入力: • error type: String • message: String

表 58. *General/Utilities/Vector*

名前	説明	入力、出力、および受け入れ可能な値
Add Element	指定のエレメントを Vector の末尾に追加し、Vector のサイズを 1 増加させます。	入力: vector: java.util.Vector 出力: element: Object
Get Element	Vector オブジェクト内の、指定のインデックスの位置にあるエレメントを取得します。	入力: • vector: java.util.Vector • index: int 出力: element: Object
Iterate Vector	Vector オブジェクトを繰り返し処理します。	入力: • vector: java.util.Vector • current index: int • current element: Object
New Vector	新しい Vector オブジェクトを作成します。	出力: vector: java.util.Vector
Size	Vector 内のエレメントの数を取得します。	入力: vector: java.util.Vector 出力: size: int
To Array	Vector 内のすべてのエレメントを含む配列表現を取得します。	入力: vector: java.util.Vector 出力: array: Object[]

例 1: 値を大文字に変更する手順

以下の例に、Activity Editor を使用してソース属性の値をすべて大文字に変更し、変更内容を宛先属性に代入するためのステップを示します。

以下の手順を実行します。

1. 「ダイアグラム」タブからソース属性を宛先属性にドラッグし、カスタム変換規則を作成します。その後、カスタム変換規則のアイコンをクリックし、Activity Editor を開きます。

   結果: Activity Editor がオープンします。

   例: 図 55 に、この例で使用するカスタム変換を示します。ソース属性は ObjClarify_contact.LastName であり、宛先属性は ObjContact.LastName です。

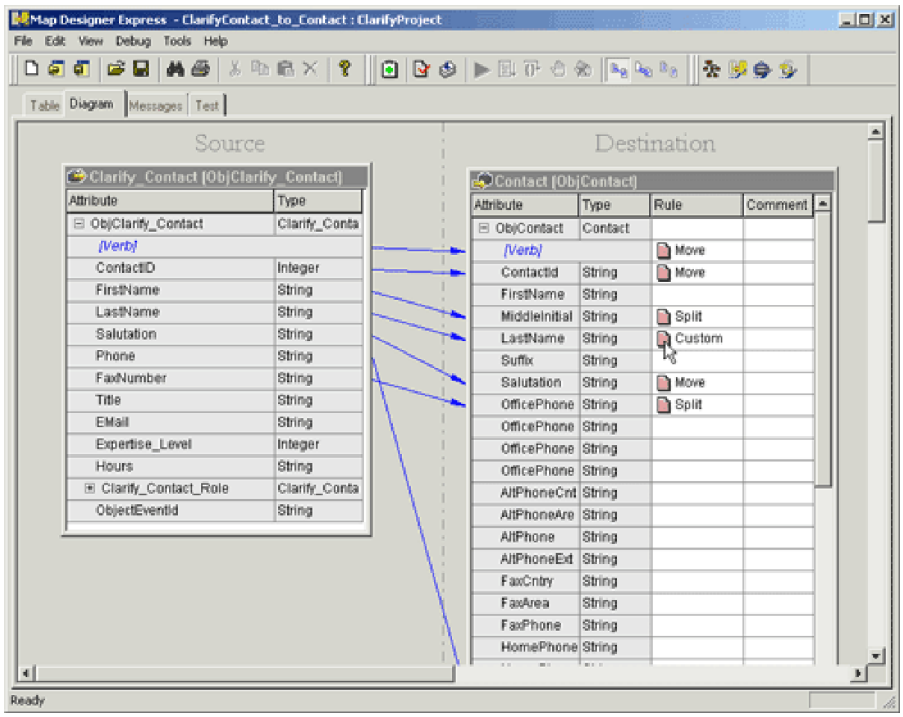

図 55. カスタム変換規則

カスタム変換などの変換を作成する方法については、15 ページの『第 2 章 マップの作成』を参照してください。

2. 「ライブラリー」ウィンドウ (左上) でカテゴリーを選択し、「コンテンツ」ウィンドウ (左下) でそのカテゴリーで使用可能な関数ブロックをアイコンとして表示させます。

   図 56 に、「ストリング」カテゴリーの場合に使用可能な関数ブロックを示します。この例のソース属性および宛先属性は、編集中のキャンバスにアイコンとして表示されます。

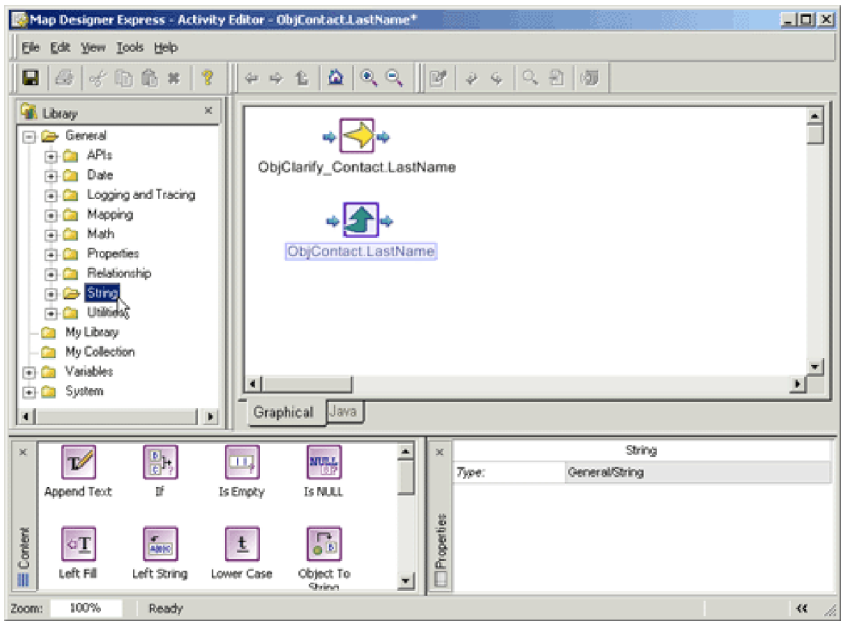

図 56. 「ストリング」カテゴリーの関数ブロックおよびソース属性および宛先属性のアイコン

3. アクティビティーの関数ブロックを使用するには、「ライブラリー」ウィンドウのツリーから関数ブロックをドラッグして編集中のキャンバスにドロップするか、「コンテンツ」ウィンドウからアイコンをドラッグして編集中のキャンバスにドロップします。

   例: この例では、ソース属性をすべての大文字に変更します。このため、「コンテンツ」ウィンドウから「ストリング」カテゴリーの Upper Case 関数ブロックを編集中のキャンバスにドラッグ・アンド・ドロップします。この操作を図 57 に示します。

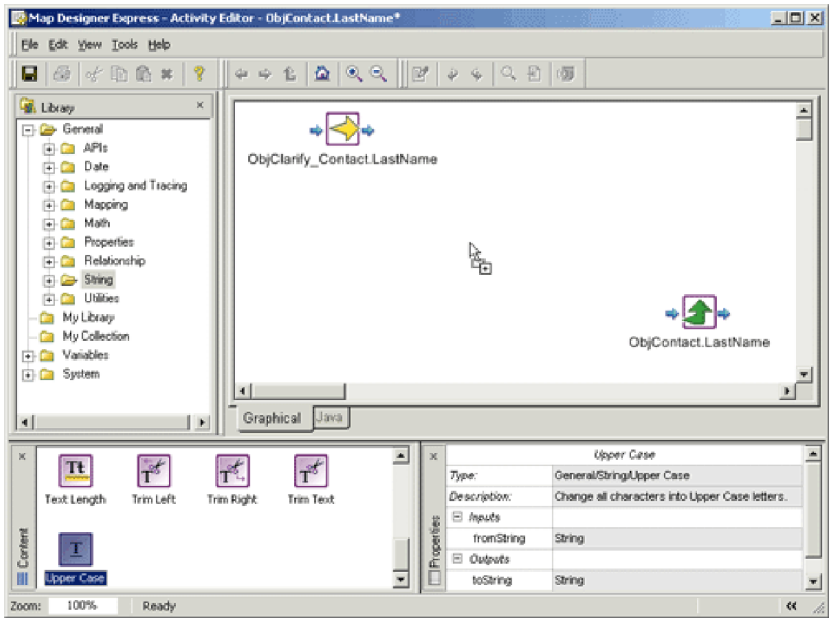

図 57. Upper Case 関数ブロックのドラッグ

4. 編集中のキャンバスに関数ブロックをドロップした後、関数ブロック・アイコンを選択して所定の位置にドラッグ・アンド・ドロップすることにより、キャンバスの関数ブロックを移動できます。関数ブロックを適切な位置に置いたら、関数ブロックの入力および出力を接続し、実行のフローを定義できます。

   例: この例では、ObjClarify_Contact.LastName の属性値をすべての大文字に変換します。これを行うために、ObjClarify_Contact.LastName のアイコンの出力を Upper Case 関数ブロックの入力に接続します。これを行うには、ポート ObjClarify_Contact.LastName のアイコンの出力にマウス・カーソルを移動します。

   結果: アイコンの形が矢印に変化し、その点でリンクを開始できることが示されます。これを図 58 に示します。

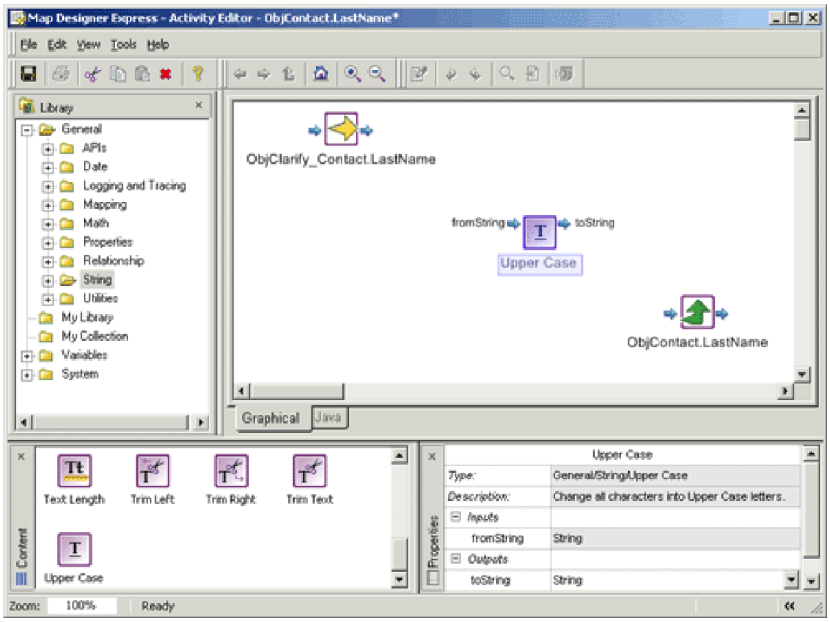

図 58. ObjClarify_Contact.LastName の出力ポートに置かれた矢印カーソル

5. マウス・アイコンが矢印に変化したら、マウス・ボタンを押したままマウスを Upper Case 関数ブロックの入力に移動し、マウス・ボタンを放します。接続リンクが描画され、入力および出力が接続されます。

   Upper Case 関数ブロックの結果を宛先属性 (この例では ObjContact.LastName) に代入することを示すために、同じステップを繰り返し、Upper Case 関数ブロックの出力から ObjContact.LastName ポート・アイコンの入力にドラッグ・アンド・ドロップします。図 59 に接続リンクを示します。

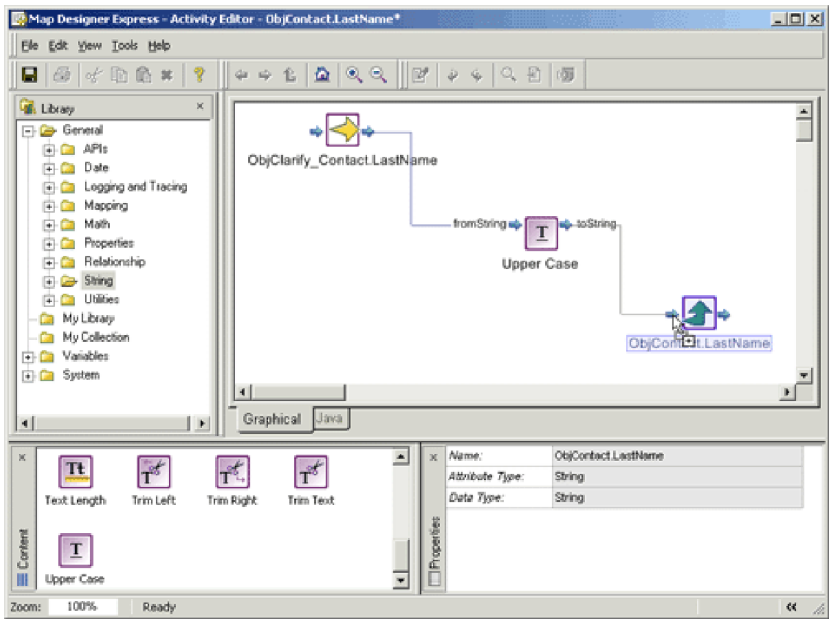

図 59. 接続リンクが付いた Upper Case 関数ブロック

結果: ソース属性の値を受け取って大文字に変換し、その大文字に変換した値を宛先属性に設定するアクティビティーが定義されました。

6. 「ファイル」>「保管」サブメニューから「プロジェクトに」または「ファイルに」を選択するか、標準ツールバーの「マップをプロジェクトに保管」ボタンまたは「マップをファイルに保管」を選択し、アクティビティーを保管します。
7. このアクティビティーによって生成される Java コードの例を参照するには、「Java」タブをクリックします。

結果:「Java」タブにより、サンプル Java コードがアクティブ化されます。これを図 60 に示します。

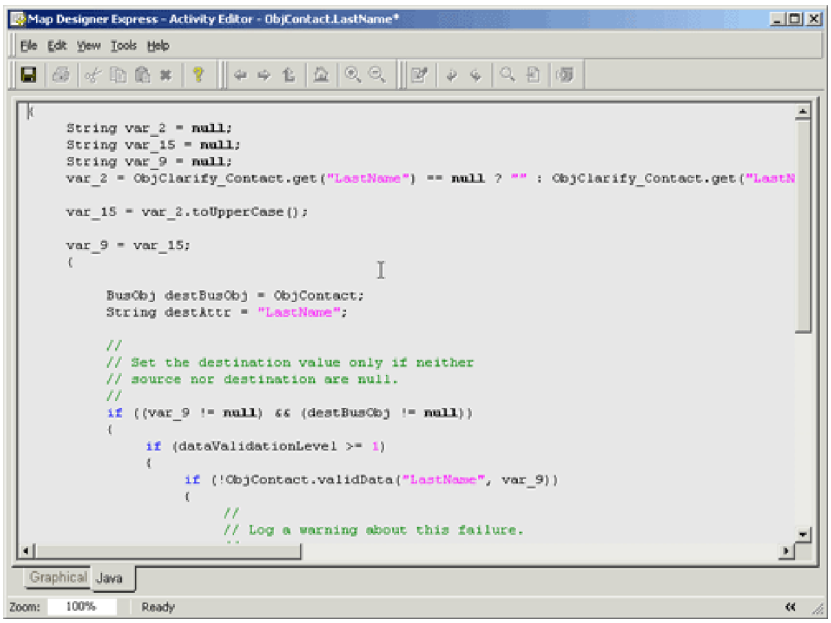

図 60. 「Java」タブでのコードの表示

例 2: 日付形式変更の手順

以下の例に、Activity Editor を使用してソースの値の日付形式を別の形式に変更し、宛先属性に代入するためのステップを示します。

以下の手順を実行します。

1. 「ダイアグラム」タブからソース属性を宛先属性にドラッグし、カスタム変換規則を作成します。その後、カスタム変換規則のアイコンをクリックし、Activity Editor を開きます。

   結果: Activity Editor がオープンします。

   例: 図 61 に、この例で使用するカスタム変換を示します。ソース属性は ObjClarify_QuoteSchedule.PriceProgExpireDate であり、宛先属性は ObjARInvoice.GLPostingDate です。

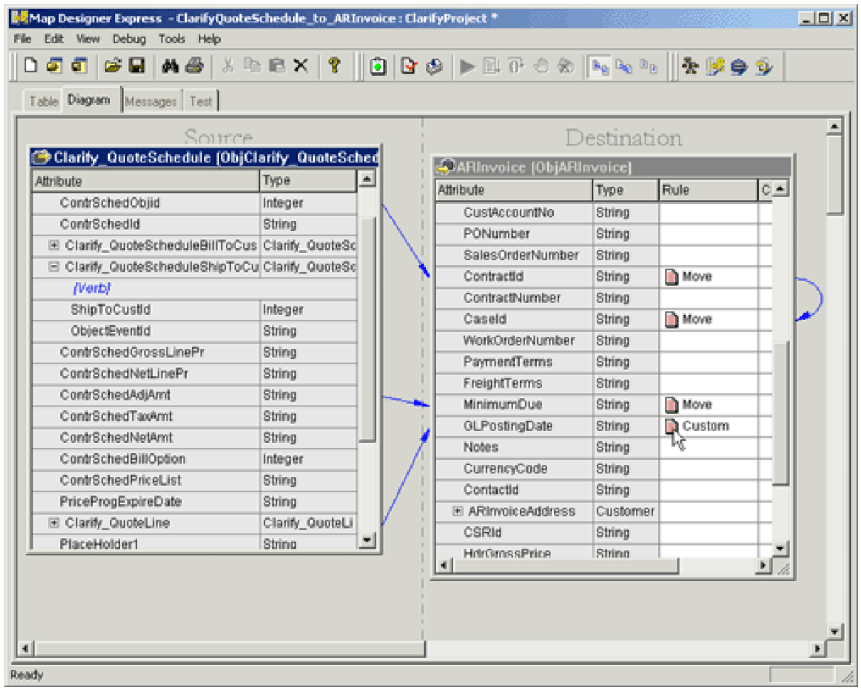

図 61. カスタム変換規則

カスタム変換などの変換を作成する方法については、15 ページの『第 2 章 マップの作成』を参照してください。

2. 「ライブラリー」ウィンドウ (左上) でカテゴリーを選択し、「コンテンツ」ウィンドウ (左下) でそのカテゴリーで使用可能な関数ブロックをアイコンとして表示させます。

   図 62 に、「日付」カテゴリーの場合に使用可能な関数ブロックを示します。この例のソース属性および宛先属性は、編集中のキャンバスにアイコンとして表示されます。

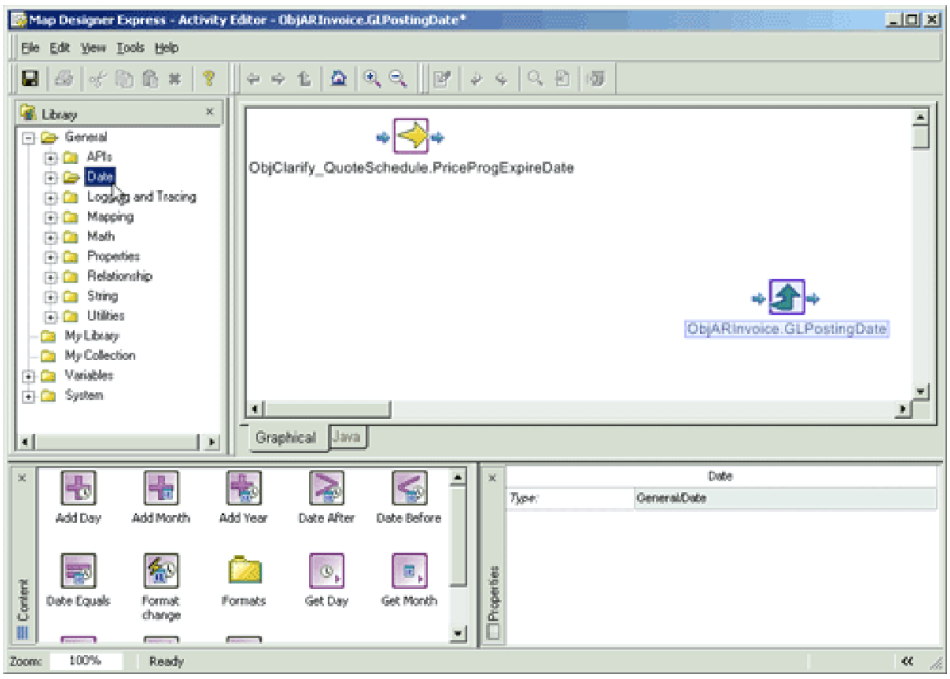

図 62. 「日付」カテゴリーの関数ブロックおよびソース属性および宛先属性のアイコン

3. アクティビティーの関数ブロックを使用するには、「ライブラリー」ウィンドウのツリーから関数ブロックをドラッグして編集キャンバスにドロップするか、「コンテンツ」ウィンドウからアイコンをドラッグしてアクティビティー・キャンバスにドロップします。

   例: ソース属性の日付形式を「yyyyMMdd」から「yyyy.MM.dd G 'at' HH:mm:ss z」に変更し、宛先属性に代入します。このため、「コンテンツ」ウィンドウから「日付」カテゴリーの Format Change 関数ブロックを編集中のキャンバスにドラッグ・アンド・ドロップします。これを図 63 に示します。

   注: 「yyyyMMdd」でフォーマット設定された日付は「20030227」などの日付です。「yyyy.MM.dd G 'at' HH:mm:ss z」でフォーマット設定された日付は「2003.02.27 AD at 00:00:00 PDT」などの日付です。

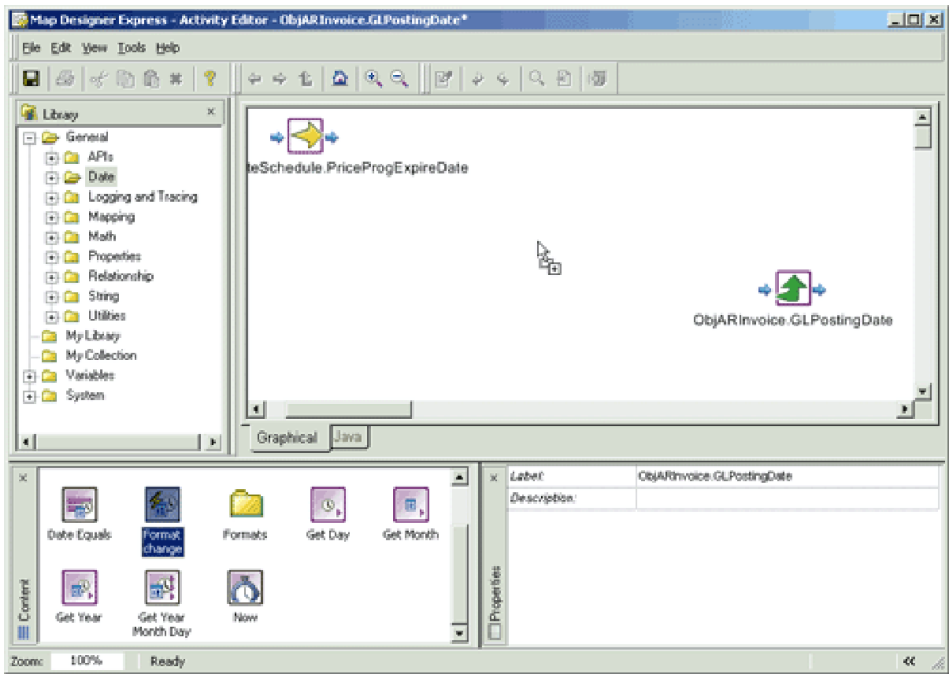

図 63. Date Format Change 関数ブロックのドラッグ

4. アクティビティー・キャンバスに関数ブロックをドロップした後は、関数ブロック・アイコンを選択して任意の場所にドラッグ・アンド・ドロップすることにより、関数ブロックをキャンバス上で移動できます。関数ブロックを適切な位置に置いたら、関数ブロックの入力および出力を接続し、実行のフローを定義できます。

   例: ソース属性 ObjClarify_QuoteSchedule.PriceProgExpireDate の日付形式を変更します。これを行うために、ObjClarify_QuoteSchedule.PriceProgExpireDate のポート・アイコンの出力を Format Change 関数ブロックの date 入力へ接続します。これを行うには、ポート ObjClarify_QuoteSchedule.PriceProgExpireDate のアイコンの出力にマウス・カーソルを移動します。

   結果: アイコンの形が矢印に変化し、その点でリンクを開始できることが示されます。これを図 64 に示します。

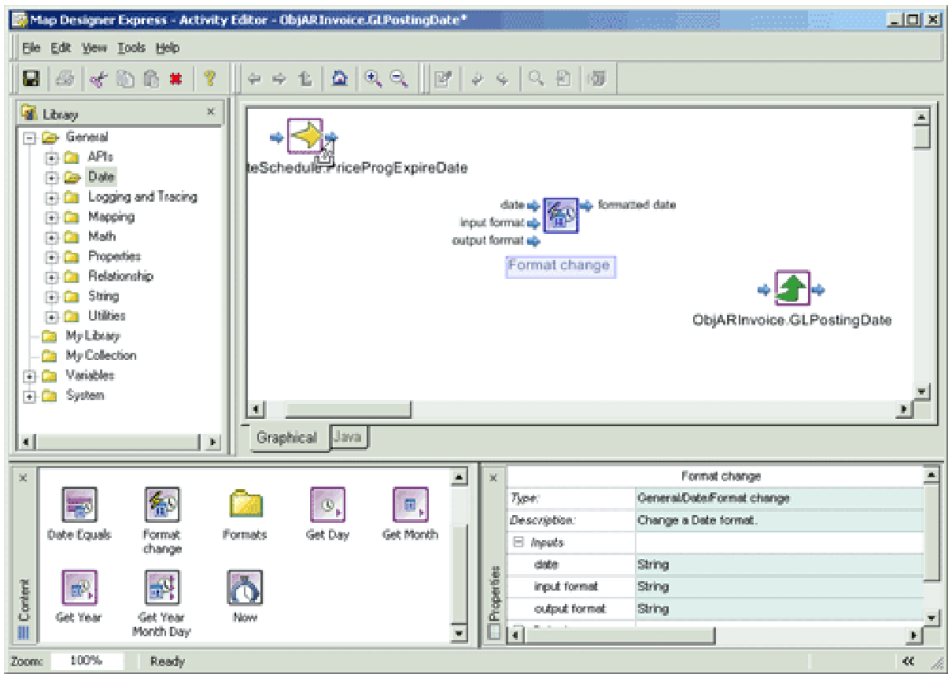

図 64. ObjClarify_QuoteSchedule.PriceProgExpireDate の出力ポートに置かれた矢印カーソル

5. マウス・アイコンが矢印に変化したら、マウス・ボタンを押したままマウスを Format Change 関数ブロックの date 入力へ移動し、マウス・ボタンを放します。接続リンクが描画され、入力および出力が接続されます。

   Format Change 関数ブロックの結果を宛先属性 ObjARInvoice.GLPostingDate に代入することを示すために、同じステップを繰り返し、Format Change 関数ブロックの出力から ObjARInvoice.GLPostingDate ポート・アイコンの入力にドラッグ・アンド・ドロップします。図 65 に接続リンクを示します。

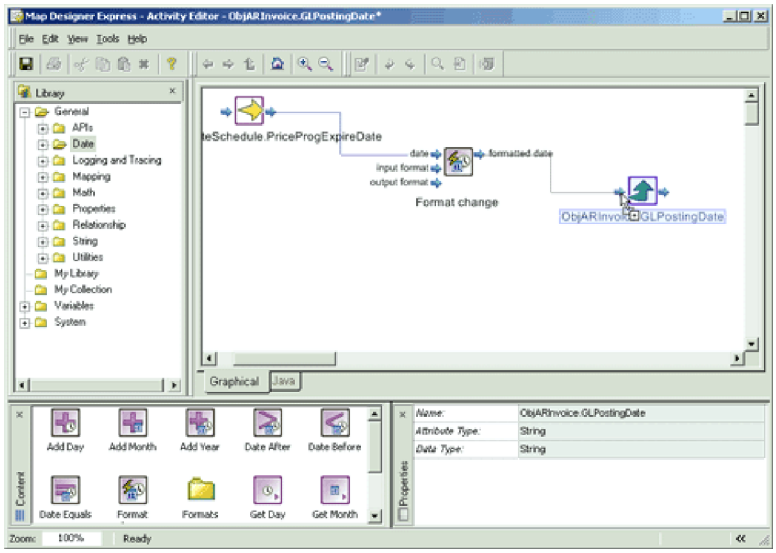

図 65. 接続リンクが付いた Date Format Change 関数ブロック

結果: 属性 ObjClarify_QuoteSchedule.PriceProgExpireDate から入力を受け取り、日付形式を変更し、結果を属性 ObjARInvoice.GLPostingDate に代入するための Format Change 関数ブロックについて説明しました。しかし、Format Change 関数ブロックに元の日付形式および結果の形式を指定する必要があります。

6. この例では、ソース属性 ObjClarify_QuoteSchedule.PriceProgExpireDate が yyyMMDD の日付形式 (すなわち 20030227) である場合は、定義済みの Date Format 関数ブロック yyyyMMdd を使用できます。yyyyMMdd 関数ブロックをアクティビティー・キャンバスにドラッグ・アンド・ドロップし、yyyyMMdd 関数ブロックの format 出力を Format Change 関数ブロックの input format に接続します。

   結果: これにより、日付の入力形式が yyyyMMdd 形式であることが指定されます。これを図 66 に示します。

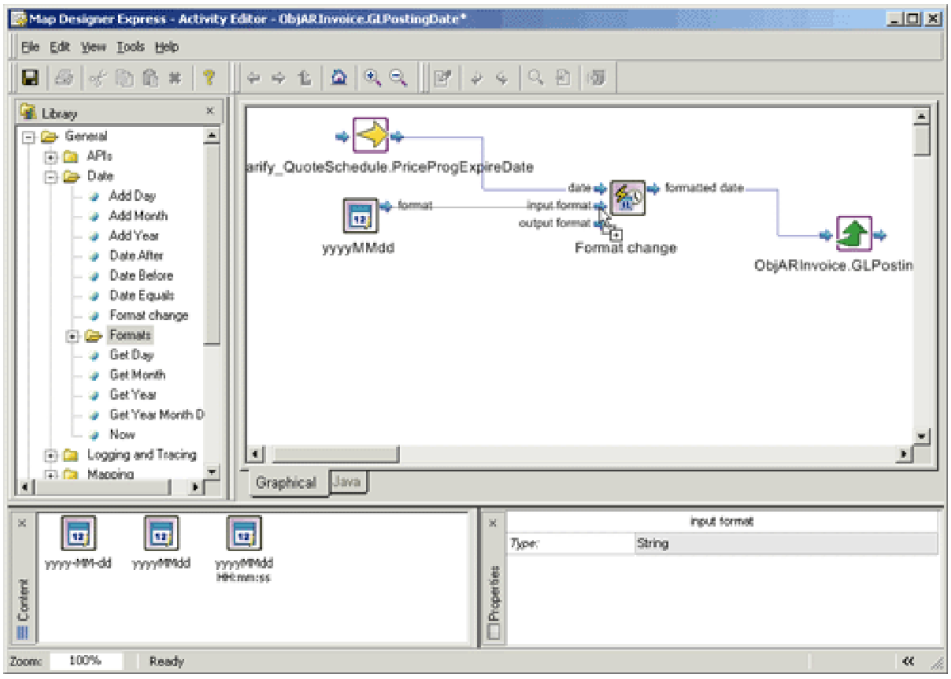

図 66. 入力日付形式

7. Activity Editor には、yyyyMMDD HH:mm:ss、yyyyMMDD、および yyyy-MM-dd の 3 つの定義済みの日付形式があります。必要な日付形式がこれら 3 つの定義済みの形式のいずれでもない場合は、定数 を使用して日付形式を指定できます。

   例: この例では、Format Change 関数ブロックで日付形式を yyyy.MM.dd G 'at' Hh″mm″ss z に変更します。これは定義済みの形式ではないため、「*新規定数*」アイコン (「システム」カテゴリーの下にある) を「コンテンツ」ウィンドウからアクティビティー・キャンバスにドラッグ・アンド・ドロップすることによってアクティビティー・キャンバスに新規定数コンポーネントを作成します。図 67 は、このアクションの結果を示しています。

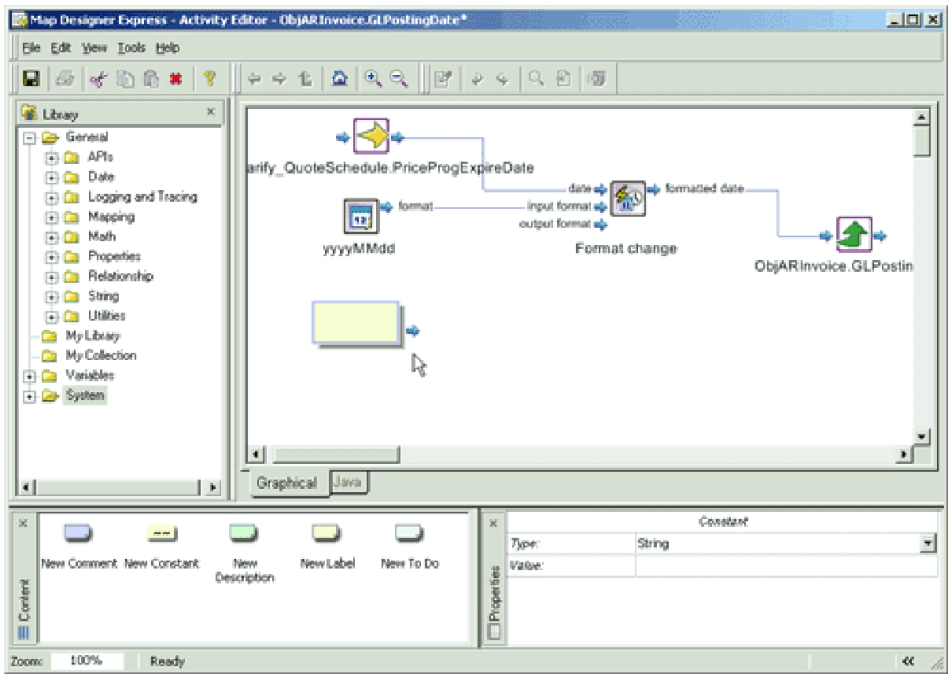

図 67. アクティビティー・キャンバスへの「新規定数」アイコンのドロップ

8. 値 yyyy.MM.dd G 'at' Hh″mm″ss z の定数を指定するには、アクティビティー・キャンバスで「定数」コンポーネントの編集可能域をクリックし、テキスト `yyyy.MM.dd G 'at' Hh"mm"ss z` を入力します。デフォルトでは、定数コンポーネントの型は String です (定数コンポーネントを選択した状態の「プロパティー」ウィンドウに表示されます)。ただし、定数を選択して「プロパティー」ウィンドウのコンボ・ボックスを使用することによって、定数の型を変更できます。図 68 に、テキスト値を入力した「新規定数」アイコンを示します。

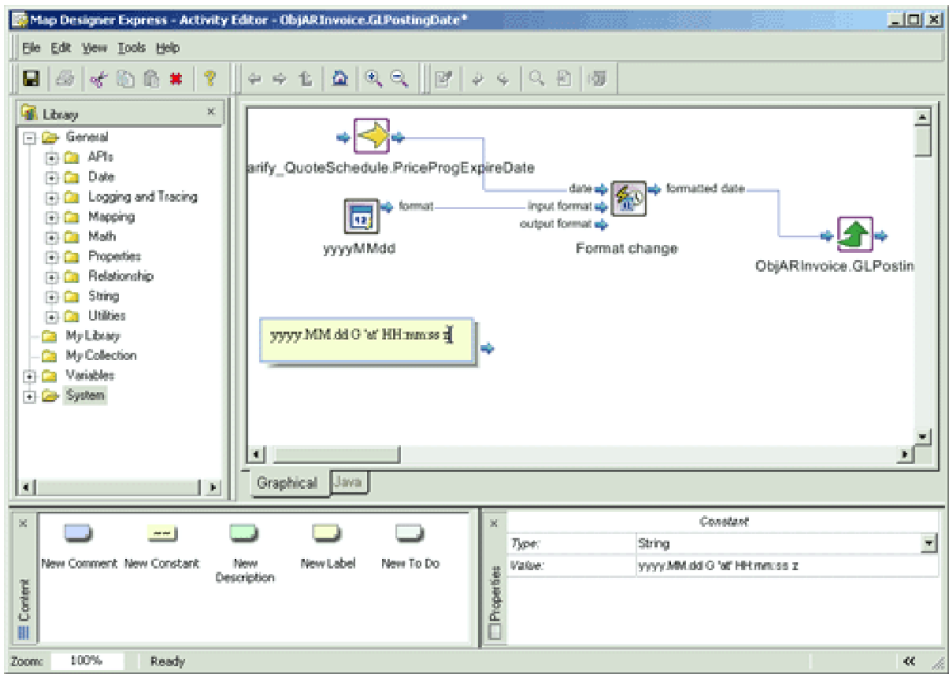

図 68. テキストが入力された「新規定数」

9. Format Change 関数ブロックの出力形式として yyyy.MM.dd G 'at' Hh″mm″ss z を指定するために、定数コンポーネントと Format Change 関数ブロックの output format の間に接続リンクを定義します。

   結果: ソース属性の日付形式を新しい日付形式に変更し、宛先属性に代入するアクティビティー定義が完了しました。

10. コメントまたは記述を追加してこのアクティビティーの処理内容が後からもわかりやすくするために、アクティビティーに*記述* コンポーネントを追加し、記述に入力できます。

    ヒント: 編集中のキャンバスのコンテキスト・メニューを使用して「記述を追加」を選択するか、「コンテンツ」ウィンドウのシステム・フォルダーにある「*新規記述*」アイコンをドラッグして編集中のキャンバスにドロップします。図 69 に、コンテキスト・メニューを使用して記述コンポーネントを追加する様子を示します。

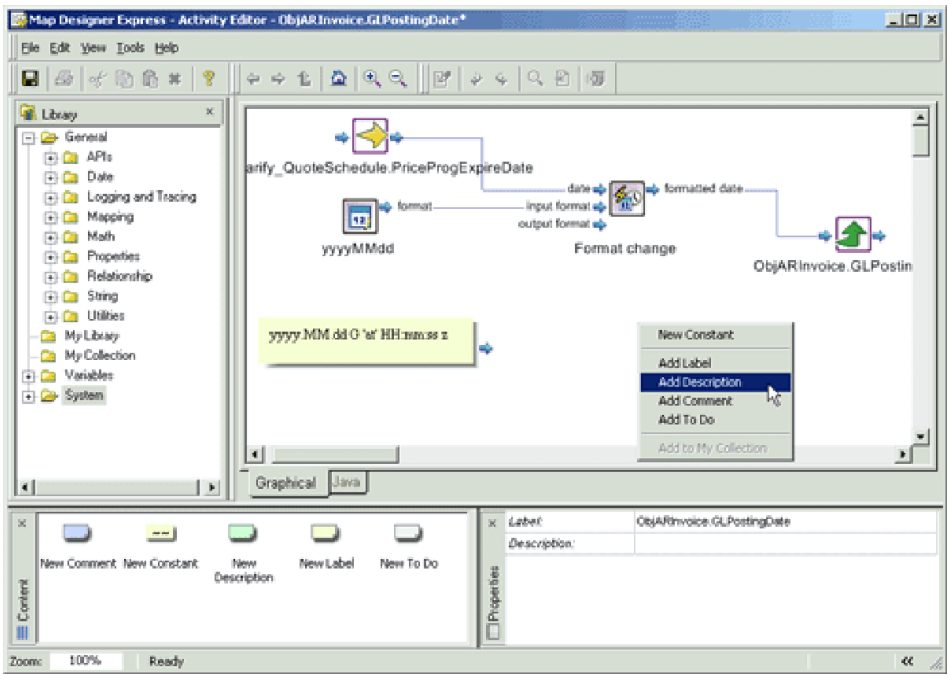

図 69. コンテキスト・メニューを使用した記述の追加

結果: 記述コンポーネントが編集中のキャンバスに作成されます。

11. コンポーネントの編集可能域をクリックしてコンポーネントに直接入力することにより、記述コンポーネントに記述を入力します。記述コンポーネントの右下隅をクリックして移動することによって、記述のサイズを変更できます。図 70 に記述の追加を示します。

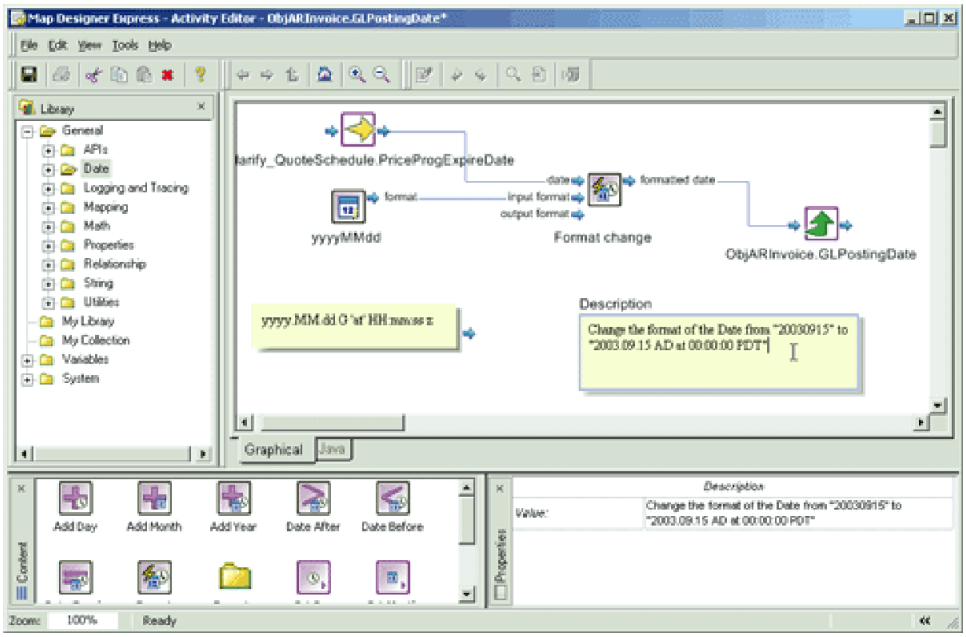

図 70. 記述の追加

12. 「ファイル」>「保管」サブメニューから「プロジェクトに」または「ファイルに」を選択するか、標準ツールバーの「マップをプロジェクトに保管」ボタンまたは「マップをファイルに保管」を選択し、アクティビティーを保管します。図 71 にアクティビティーの保管を示します。

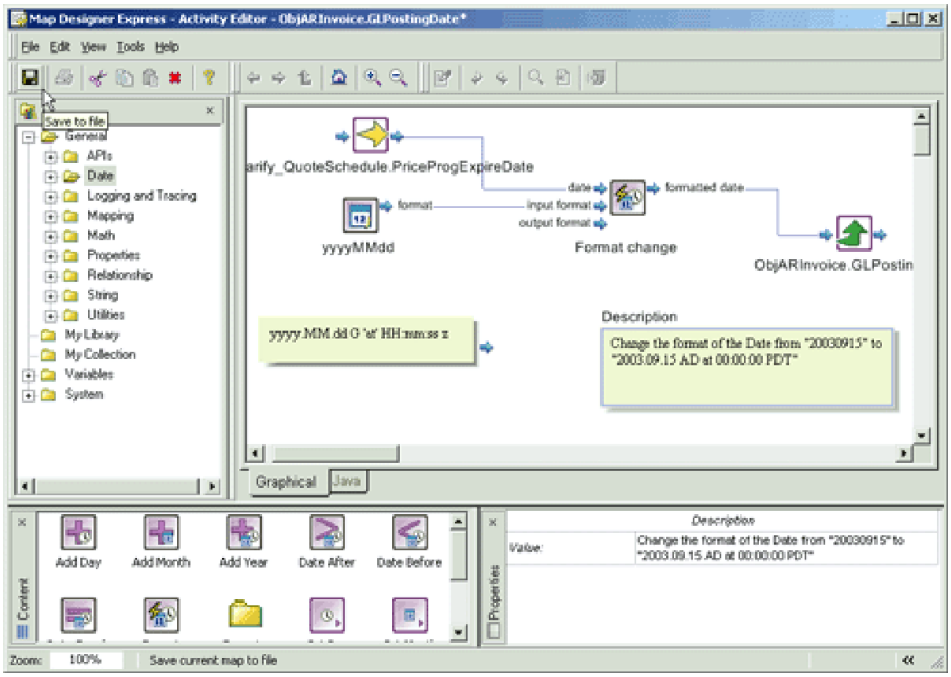

図 71. アクティビティーの保管

例 3: 変換での Static Lookup の使用

次の例に、Activity Editor での *Static Lookup* 関係関数ブロックの使用法を示します。

WebSphere Business Integration Server Express では、静的参照関係は通常は複数の関係表で構成されます。例えば、図 72 に示した 3 つのエンド・アプリケーションから構成されたシステムについて考えてみましょう。

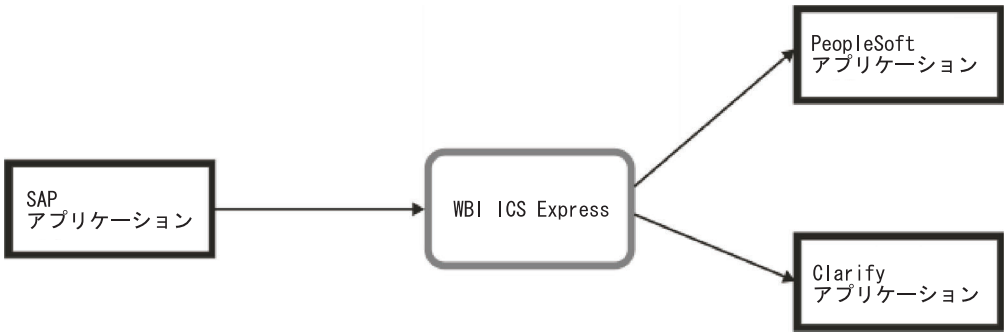

図 72. 3 つのエンド・アプリケーションを使用した静的参照関係

表 59 に示すように、これら 3 つのアプリケーションはそれぞれ、「州」の情報に関しては情報の表記が異なっています。

表 59. 州の情報に関するアプリケーション固有の表記

	SAP アプリケーション	PeopleSoft アプリケーション	Clarify アプリケーション
カリフォルニア	CA	01	State1
ワシントン	WA	02	State2
ハワイ	HI	03	State3
デラウェア	DE	04	State4

州の情報を SAP アプリケーションから WebSphere Business Integration システムに送信した場合、SAP で指定した州のコードが InterChange Server Express に送信されます。ただし、この情報を InterChange Server Express から他のアプリケーションに渡す必要がある場合は、州の情報をターゲット・アプリケーションに認識されるフォーマットに変換する必要があります。この変換を行うには、「州」の情報に関するシステム表記が汎用のものでなければなりません。システムは、汎用の表記を使用することで、ビジネス・ロジックを汎用的かつ統一された形で処理することができます。汎用表記は、必要な場合にのみ、アプリケーション固有のフォーマットに変換されます。

このため、前述のような例では、この「州」を変換する目的に静的参照関係を作成します。その際には、アプリケーション固有のデータを WebSphere Business Integration 管理下の参加データとして使用します。このセットアップを準備しておけば、WebSphere Business Integration システムで、汎用 ID を使用して州の情報を表すことができます。表 60 は、この表記を示しています。

表 60. 州の情報の汎用表記

	汎用 ID	SAP アプリケーション	PeopleSoft アプリケーション	Clarify アプリケーション
カリフォルニア	1	CA	01	State1
ワシントン	2	WA	02	State2
ハワイ	3	HI	03	State3
デラウェア	4	DE	04	State4

アプリケーション固有のデータは、InterChange Server Express システムに入力されると 汎用 ID に変換され、汎用 ID はアプリケーション固有のデータに変換されてからシステムから出力されます。このデータ変換を図 73 に示します。

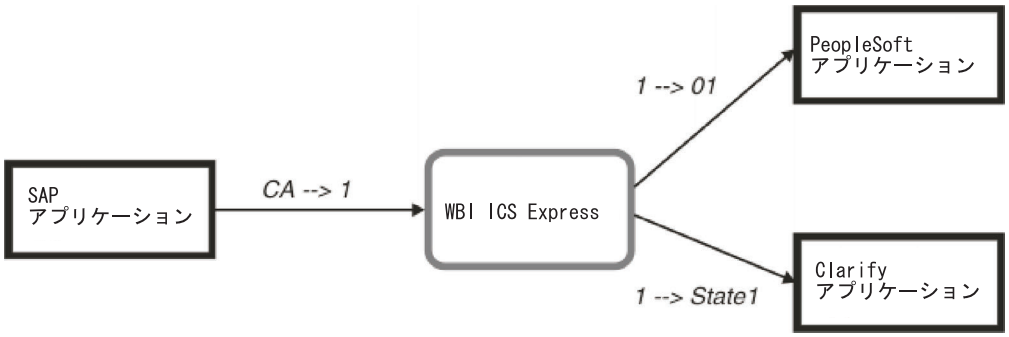

図 73. アプリケーション固有、汎用、アプリケーション固有へのデータ変換

ID 変換は通常、アプリケーション固有のビジネス・オブジェクトから汎用のビジネス・オブジェクトへの変換、またはその逆の変換を行うマップで実行されます。例えば、SAP-to-Generic マップで、データ「CA」に対して静的参照を実行して、「CA」を InterChange Server Express に認識される汎用表記「1」に変換してみましょう。また、Generic-to-Clarify マップでは、代わりに汎用データ「1」に対して静的参照を実行して、「1」を「State1」に変換します。どちらのマップの場合も、静的参照は 1 回だけで済みます。

図 74 は、Static Lookup 関数ブロックを使用して、SAP で指定した州のデータを InterChange Server Express で処理される InterChange Server Express 汎用の州データに変換する方法を示しています。

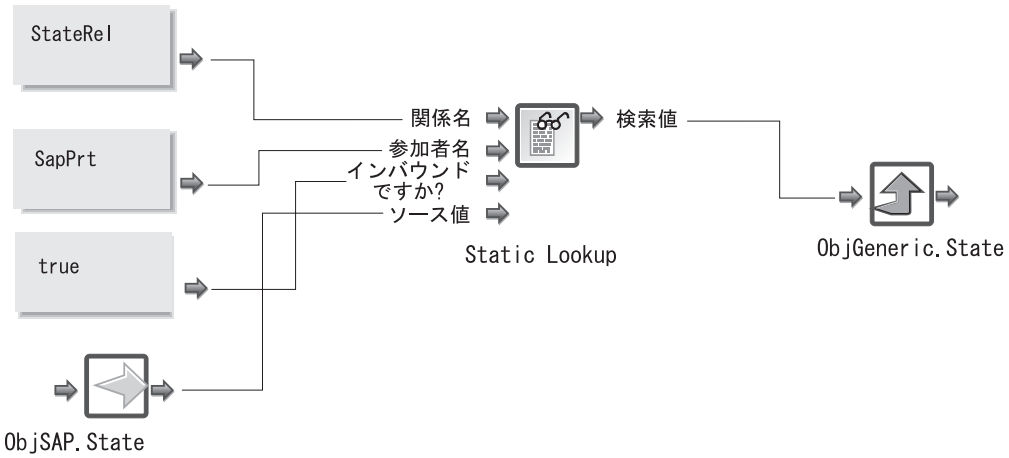

図 74. Static Lookup 関数ブロックを使用した、SAP 固有の州データから InterChange Server Express 汎用の州データへの変換

同様に、Static Lookup 関数ブロックを使用して、Generic-to-Clarify マップで InterChange Server Express 汎用の州データを Clarify 固有の州データに変換することもできます。この変換を 174 ページの図 75 に示します。

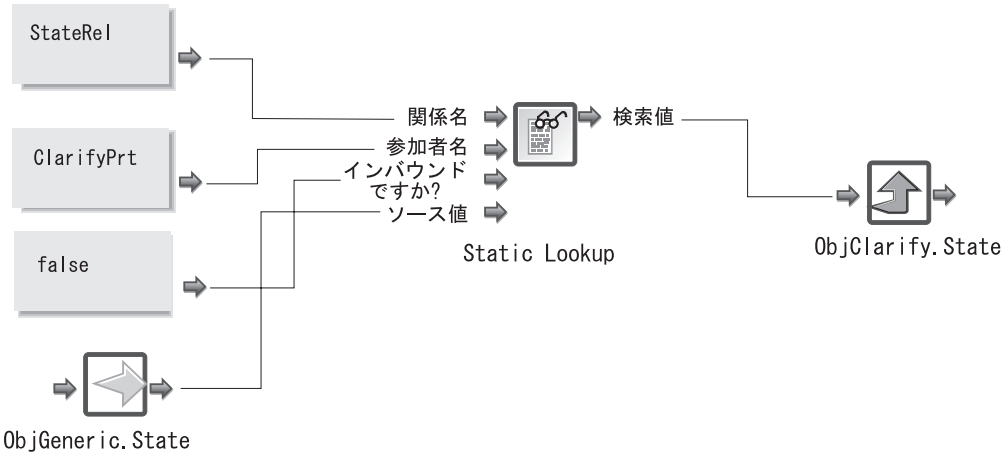

図 75. Static Lookup 関数ブロックを使用した、InterChange Server Express 汎用の州データから Clarify 固有の州データへの変換

通常、静的参照関係では、アプリケーション固有のデータを汎用データに変換するか、または汎用データをアプリケーション固有のデータに変換します。これらのシナリオでは、Static Lookup 関数ブロックを 1 つだけ使用します。ただし、名前と値のペアを直接に検索するといった特別な事例では、Static Lookup 関数ブロックが 2 つ必要です。

静的関係の定義および使用について詳しくは、259 ページの『第 7 章 関係定義の作成』を参照してください。

Java パッケージと他のカスタム・コードのインポート

Map Designer Express では、以下の 2 通りの方法で Java パッケージやその他のカスタム・コードをインポートすることができます。

- 『JAR ライブラリーのアクティビティー関数ブロックとしてのインポート』
- 177 ページの『「マップ・プロパティー」ダイアログを使用したインポート』

ここでは、これらの方法について詳しく説明します。

JAR ライブラリーのアクティビティー関数ブロックとしてのインポート

Map Designer Express では、Activity Editor の標準の関数ブロックを使用できるだけでなく、ユーザー独自の Java ライブラリーをインポートして、Activity Editor で関数ブロックとして使用することもできます。カスタム JAR ライブラリーをアクティビティー設定にインポートすると、その JAR ライブラリーに含まれる public メソッドを、Activity Editor で関数ブロックとして使用できます。

JAR ライブラリーをアクティビティー関数ブロックとしてインポートする手順

始める前に: 適切なユーザー独自の Java クラスを、.jar ファイルとしてエクスポートする必要があります。

JAR ライブラリーを Activity Editor にインポートするには、以下の手順を実行します。

1. System Manager で、「ウィンドウ」>「Show View」>「その他」をクリックし、カテゴリー「System Manager」の「Activity Settings」を選択して、「Activity Settings」ビューを開きます。
2. 「BuildBlock Libraries」を右マウス・ボタンでクリックし、「Add Library」を選択します。図 76 に、カスタム JAR ライブラリーを追加するときに使用する「Activity Settings」ビューを示します。

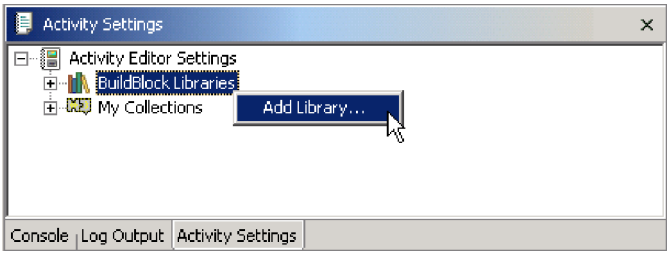

図 76. 「Activity Settings」ビュー

3. 「Open File」ダイアログ・ボックスで、使用するカスタム .jar ファイルを探して「開く」を選択します。

   カスタム .jar ファイルを関数ブロックとして使用できるように Activity Editor にインポートする処理が、System Manager によって試行されます。カスタム .jar ファイルが正常にインポートされると、「Activity Settings」ビューの「BuildBlock Libraries」の下にそのファイルの名前が表示されます。

   ヒント: 「Activity Settings」にカスタム .jar ファイルをインポートした後、System Manager でマップやコラボレーション・テンプレートをコンパイルすると、そのカスタム .jar ファイルが、コンパイル CLASSPATH に自動的に含められます。InterChange Server Express でコンパイルを実行できるようにするには、InterChange Server の CLASSPATH に custom.jar ファイルを含める必要があります。カスタム .jar ファイルをインポートできるように InterChange Server Express をセットアップする方法については、179 ページの『サード・パーティーのクラスを Interchange Server Express にインポートする』を参照してください。
4. Map Designer Express を再始動します。

   規則: 「Activity の設定」ビューの設定を変更したときは、変更内容を Activity Editor に反映させるため、Map Designer Express を再始動する必要があります。

結果: Activity Editor を開くと、「My Library」の下に、インポートしたカスタム JAR ライブラリーが表示されます。デフォルトでは、使用可能なカスタム関数ブロックがそれぞれのパッケージ構造に従って表示されます。これらのカスタム関数ブロックは、標準の関数ブロックを使用する場合と同様の方法で、アクティビティーに使用することができます。

カスタム JAR ライブラリーの表示設定のカスタマイズ

Activity Editor にインポートした関数ブロックの表示設定 (名前やアイコンの設定など) は、カスタム JAR ライブラリーのプロパティーを変更することによって、カスタマイズできます。これを行うには、以下の手順を実行します。

- System Manager の「Activity Settings」ビューの「BuildBlock Libraries」の下で、表示設定をカスタマイズするカスタム JARライブラリーを右マウス・ボタンでクリックし、そのカスタム JAR ライブラリーの「プロパティー」ウィンドウを表示します。

  結果: 選択したカスタム JAR ライブラリーの「プロパティー」ウィンドウが開き、そのカスタム・ライブラリーに含まれる使用可能な関数ブロックが、ダイアログの右側にツリー表示されます。これらの使用可能な関数ブロックは、それらが含まれる Java クラスや Java パッケージの下位ノードとして表示されます。

Java パッケージや Java クラスに関しては、各エントリーの表示名をカスタマイズできます。また、「Hide level in tree display」チェック・ボックスの設定を変更すれば、その Java パッケージまたは Java クラスを Activity Editor の「My Library」ツリー構造に表示するかどうかについてもカスタマイズすることができます。このオプションを選択したエントリーは、Activity Editor の「My Library」サブツリーに表示されません。通常、このオプションを使用すると便利なのは、カスタム JAR ライブラリー内の Java メソッドが、多数の階層から成るパッケージに含まれる Java クラスに属している場合です。このオプションを選択することで、Activity Editor の「My Library」サブツリーの編成を簡素化することができます。

図 77 に、JAR ライブラリーの表示設定をカスタマイズするときに使用するダイアログを示します。

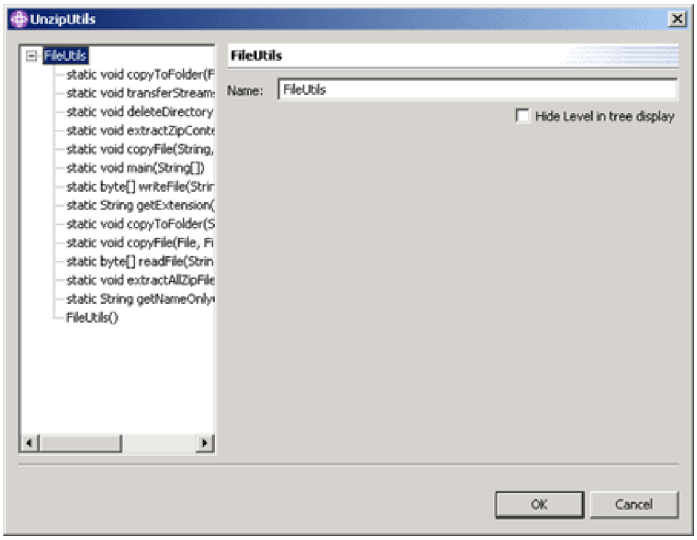

図 77. JAR ライブラリーの表示設定をカスタマイズするときに使用する「プロパティー」ダイアログ

Activity Editor で関数ブロックとして使用される Java メソッドに関しては、「プロパティー」ウィンドウを使用して、関数ブロックの表示名、説明、およびアイコンを指定したり、パラメーターの表示名を指定したりすることができます。関数ブロ

ックのアイコンをインポートすると、選択したアイコンは「Activity Settings」フォルダーにコピーされ、同じパッケージのほかの関数ブロックに対しても使用できるようになります。

推奨: 関数ブロックのアイコンをインポートする場合、インポートするアイコンは、サイズが 32 ピクセル x 32 ピクセルで形式が .bmp のものにしてください。アイコンの色の深さは、24 ビットまでです。

図 78 に、JAR ライブラリー内の関数ブロックの表示設定をカスタマイズするときに使用する「プロパティー」ダイアログを示します。

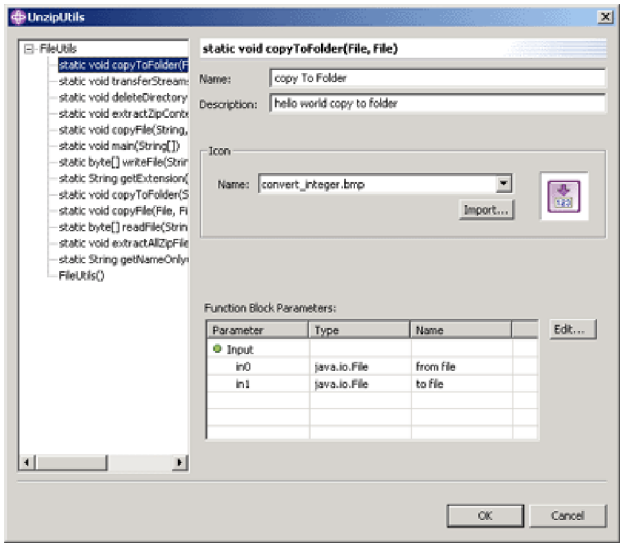

図 78. JAR ライブラリー内の関数ブロックの表示設定をカスタマイズするときに使用する「プロパティー」ダイアログ

規則: 「Activity の設定」ビューの設定を変更したときは、変更内容を Activity Editor に反映させるため、Map Designer Express を再始動する必要があります。

「マップ・プロパティー」ダイアログを使用したインポート

Map Designer Express は、マップの実行に必要な Java パッケージを自動的にインポートします。別の Java パッケージのメソッドを使用する変換コードを作成するか、またはマップ・ユーティリティー (MapUtils) パッケージを使用する場合は、パッケージをマップにインポートする必要があります。 Map Designer Express 外で独自のカスタム Java コードを作成して、そのコードを変換コード内で使用するためにマップにインポートすることもできます。

注: マップ開発には、IBM WebSphere Business Integration Server Express リリースに該当する Java Development Kit (JDK) のバージョンをインストールする必要があります。

重要: Map Designer Express では、マップにインポートされたコードのロジックをデバッグまたは検証することができません。したがって、インポートされたコードに起因するすべてのエラーや例外はマップ開発者の責任となります。

Java パッケージまたは他のカスタム・コードのインポート手順

Java パッケージや他のカスタム・コードをインポートするには、次の手順を行います。

1. 以下のようにして、「マップ・プロパティー」ダイアログの「一般」タブを表示します。

   「編集」メニューから「マップ・プロパティー」を選択します。「マップ・プロパティー」ダイアログを表示する他の方法については、64 ページの『マップ・プロパティー情報の指定』を参照してください。「マップ・プロパティー」ダイアログ・ボックスが表示されます (206 ページの図 87 を参照)。
2. マップ・ファイル宣言ブロックに `import` ステートメントを入力します。「**マップ・ローカル宣言ブロック**」に他の Java ステートメントを入力することもできます。

   注: マップをコンパイルする場合、Java コンパイラーは `CLASSPATH` 環境変数で定義されているディレクトリーからインポートされたパッケージを探します。パッケージをマップにインポートして、コンパイルする前に別の InterChange Server Express システムにマップを配置する場合は、マップと共にインポートされたパッケージも配置してください。

   System Manager でマップをコンパイルするための要件については、91 ページの『マップのコンパイル』を参照してください。

   マップの開始時にすべてのステートメントを実行した後に、変換ステップを実行します。
3. 「OK」をクリックして、「マップ・プロパティー」ダイアログ・ボックスを閉じます。

マップ・ユーティリティーのインポート手順

マップ・ユーティリティー・パッケージを使用するには、以下の手順を実行してマップにインポートする必要があります。

1. `CollabUtils.jar` ファイルが <ProductDir>¥lib ディレクトリーに存在することを確認します。
2. `start_server.bat` (または `CWSharedEnv.sh`) ファイルに `CollabUtils.jar` ファイルの参照が含まれていることを確認します。

   注: ステップ 1 および 2 はサーバー・コンパイルのために必要です。
3. コンパイラーの CLASSPATH 設定を変更して、System Manager からマップやコラボレーション・テンプレートをコンパイルできるようにします。
   a. System Manager で、「ウィンドウ」>「設定」を選択して「設定」ダイアログを開きます。
   b. 「System Manager Preferences」を展開し、「Compiler」を選択します。
   c. 「Compiler preference」ページで、「新規」をクリックし、マップやコラボレーション・テンプレートのコンパイル時に必要な `CLASSPATH` に含める `.jar` ファイルを選択します。
4. Map Designer Express で、マップ・ユーティリティー・パッケージを使用するのに必要なマップを開きます。

5. 「マップ・プロパティー」ダイアログを表示します。
6. 「マップ・プロパティー」ダイアログ・ボックスの、「マップ・ファイル宣言ブロック」に次の import ステートメントを入力します。

   ```
   import com.crossworlds.MapUtils.*;
   ```

7. 「OK」をクリックして、「マップ・プロパティー」ダイアログ・ボックスを閉じます。

サード・パーティーのクラスを Interchange Server Express にインポートする

インポートしたクラスが JDK ではなくサード・パーティーのものである場合、サーバー・コンパイルをセットアップするために、このクラスを、JCLASSES 変数のインポート済みクラスのパスに追加する必要があります。

推奨: なんらかのメカニズムを使用して JCLASSES 内の標準のクラスとカスタム・クラスを区別することをお勧めします。

例: 以下の手順を実行し、このようなカスタム・クラスのみを保持する新しい変数を作成し、この新しい変数を JCLASSES に追加します。

1. 例えば DEPENDENCIES と呼ばれる、新しいマップ・プロパティーを作成します。
2. その独自のディレクトリーに CwMacroUtils.jar を保管します。

   例: 製品ディレクトリーの下に dependencies ディレクトリーを作成して、そこに JAR ファイルを保管します。

3. dependencies ディレクトリーを InterChange Server Express の始動に使用されるファイル (デフォルトでは、start_server.bat または CWSharedEnv.sh) に追加します。このファイルは製品ディレクトリーの下の bin ディレクトリーに存在します。例えば、UNIX 用に以下のエントリーを追加します。

   ```
   set DEPENDENCIES=$ProductDir/
   dependencies/CwMacroUtils.jar
   ```

   さらに、Windows 用に以下のエントリーを追加します。

   ```
   set DEPENDENCIES="%ProductDirS%"¥dependencies¥
   CwMacroUtils.jar
   ```

4. DEPENDENCIES を JCLASSES エントリーに追加します。

   UNIX の場合、以下を追加します。

   ```
   set JCLASSES=$JCLASSES:ExistingJarFiles:
   $DEPENDENCIES
   ```

   Windows の場合、以下を追加します。

   ```
   set JCLASSES=ExistingJarFiles
   ;%DEPENDENCIES%
   ```

5. クラスを使用する各マップに、CwMacroUtils.jar ファイルで指定された *PackageName.ClassName* を組み込みます。
6. InterChange Server Express を再始動し、マップに対してメソッドを有効にします。

注: System Manager からマップやコラボレーション・テンプレートをコンパイルできるようにコンパイラーの CLASSPATH 設定が変更されていることを確認します。それには以下の手順を実行します。

1. System Manager で、「ウィンドウ」>「設定」を選択して「設定」ダイアログを開きます。
2. 「System Manager Preferences」を展開し、「Compiler」を選択します。
3. 「Compiler preference」ページで、「新規」をクリックし、マップやコラボレーション・テンプレートのコンパイル時に必要な CLASSPATH に含める .jar ファイルを選択します。

ガイドライン: カスタム・クラスをインポートするときに、ソフトウェアがこのカスタム・クラスを検出できなかったことを示すエラー・メッセージが表示されることがあります。その場合は、以下をチェックしてください。

- カスタム・クラスがパッケージの一部であるかどうかチェックします。プログラミングの慣例上、カスタム・クラスはパッケージに格納するのが理想的です。カスタム・クラス・コードに正しいパッケージ・ステートメントが含まれており、このステートメントがソース・ファイルの先頭 (任意のクラスまたはインターフェース宣言より前) にあることを確認します。
- マップ定義コードの import ステートメントが正しいことを検証します。import ステートメントは正しいパッケージ名を参照している必要があります。インポート・ステートメントは、さらにカスタム・クラスの名前を指定したり、パッケージ内のすべてのクラスを参照できます。

  例: パッケージ名が COM.acme.graphics で、カスタム・クラスが Rectangle の場合、パッケージ全体をインポートできます。

  ```
  import COM.acme.graphics.*;
  ```

  また、Rectangle カスタム・クラスのみをインポートすることもできます。

  ```
  import COM.acme.graphics.Rectangle;
  ```

- カスタム・クラスが格納されたパッケージ、またはカスタム・クラス自体 (パッケージに格納されていない場合) へのパスを含むよう CLASSPATH 環境変数が更新されていることを確認します。

例: カスタム・クラスをインポートする場合、%ProductDir%¥lib¥com¥<ProductDir>¥package (ここで、package はパッケージの名前) と呼ばれるフォルダーを作成することがあります。次に、作成したフォルダーにカスタム・クラス・ファイルを格納します。最後に、start_server.bat ファイル内の CLASSPATH 変数に、パス %ProductDir%¥lib を設定します。

Web サービスを Activity Editor にエクスポートする

Web サービスは、ビジネス・オブジェクトやマップと同様、System Manager の InterChange コンポーネント・ライブラリー・プロジェクトの一部です。Web サービスの登録、テスト、および検証が完了すれば、その Web サービスは関数ブロックとして Activity Editor にエクスポートできます。これにより、そのサービスおよびメソッドは、他の関数ブロックと同様、マップ内で簡単に使用できます。Websphere Business Integration Server を変更しなくても、マップの実行時に Web サービスが呼び出されます。

Web サービスの登録、テスト、検証、およびエクスポートについては、「システム・インプリメンテーション・ガイド」を参照してください。

Activity Editor での Web サービスの使用

System Manager から Web サービスをエクスポートした後は、Activity Editor を再始動する必要があります。Activity Editor が開くと、エクスポートされた Web サービスは、「ユーザー・ライブラリー」の下にカテゴリーとして追加されています。このカテゴリーには、「ユーザー・ライブラリー」内の他のカテゴリーと同じ機能があります。

図 79 に、System Manager からエクスポートした後の Activity Editor の「Web サービス」カテゴリーおよび関数ブロックを示します。

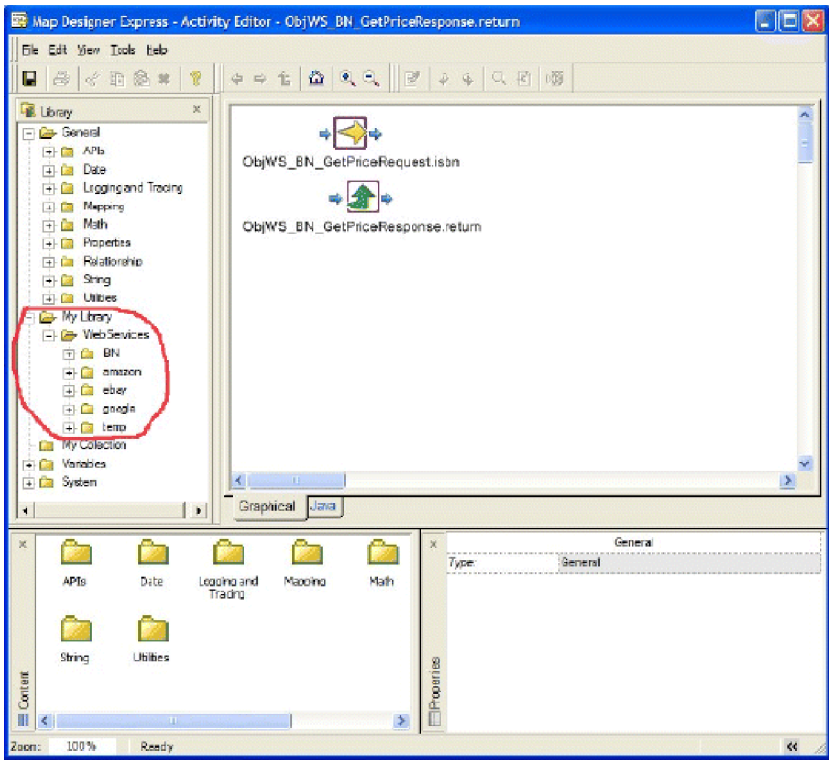

図 79. Activity Editor の「Web サービス」カテゴリー

Activity Editor で他の関数ブロックを使用する場合と同様、Web サービスの関数ブロックを使用するには、関数ブロックをドラッグ・アンド・ドロップして、入力と出力を接続します。 Activity Editor の使用については、123 ページの『アクティビティー定義の処理』を参照してください。

マップでの Web サービスの使用例

次の例は、Activity Editor を使用して Web サービスを呼び出し、ソース属性の郵便番号を市区町村の気温に変更し、変更内容を宛先属性に割り当てる方法を示します。

以下の手順を実行します。

1. Map Designer Express の「ダイアグラム」タブで、カスタム変換を作成します。まず、ソース・ビジネス・オブジェクト属性 ObjWS_Temp_GetTempRequest.zipcode を、宛先ビジネス・オブジェクト属性 ObjWS_Temp_GetTempResponse.return の上までドラッグします。次に、カスタム変換規則のアイコンをクリックして、Activity Editor を起動します。

   図 80 に、このカスタム変換を示します。

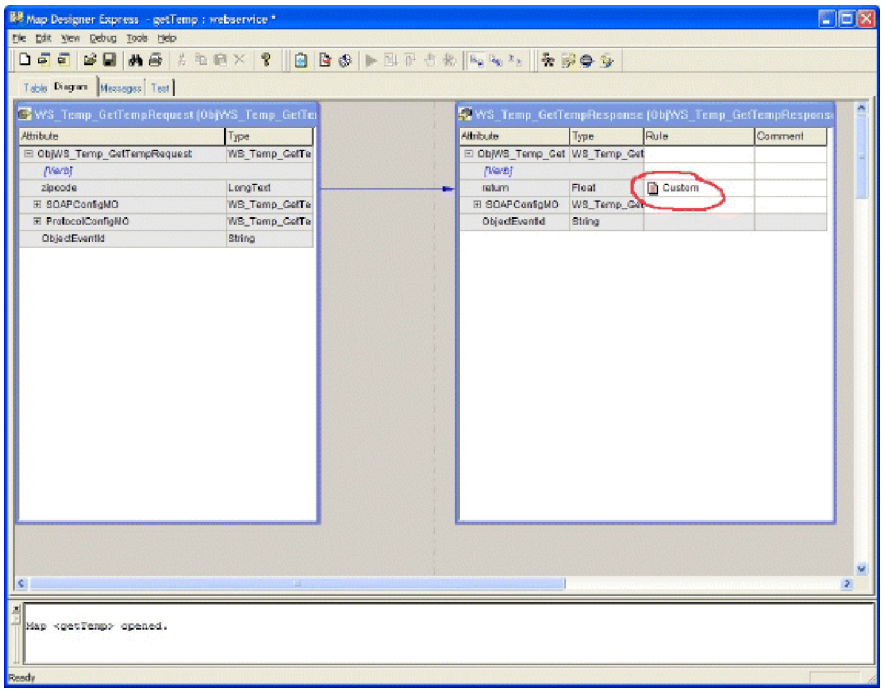

図 80. カスタム変換の作成

2. 「ライブラリー」ウィンドウで「Web サービス」カテゴリーを選択します。これにより、そのカテゴリーで使用可能な関数ブロックが「コンテンツ」ウィンドウにアイコンとして表示されます。
3. 「コンテンツ」ウィンドウ内の Web サービス getTemp 関数ブロックを編集キャンバスまでドラッグ・アンド・ドロップします。
4. ソース・ビジネス・オブジェクト属性 ObjWS_Temp_GetTempRequest.zipcode のアイコンの出力ポートを、getTemp 関数ブロックの入力ポート「zipcode」に接続します。そして、getTemp 関数ブロックの出力ポート「return」を、宛先ビジネス・オブジェクト属性 ObjWS_Temp_GetTempResponse.return のアイコンの入力ポートに接続します。

図 81 に、接続された getTemp 関数ブロックの入力と出力を示します。

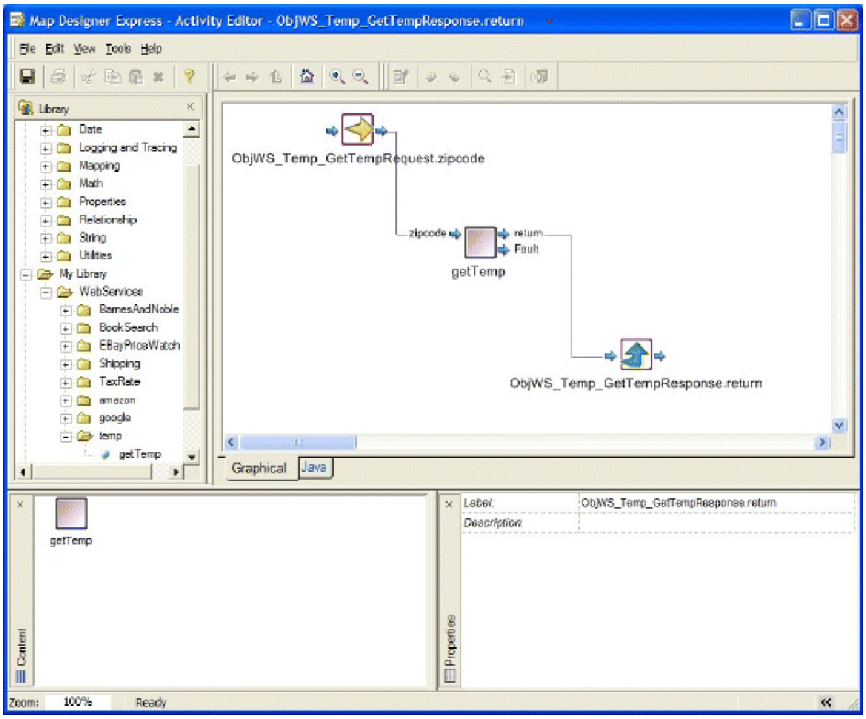

図 81. 入力と出力の接続

5. アクティビティー・テンプレートとマップを保管します。
6. Map Designer Express のテスト表示に切り替えます。ソースの「zipcode」フィールドに有効な郵便番号を入力します。「マップのデバッグ」をクリックします。マップおよびビジネス・オブジェクトをまだサーバーに配置していない場合は、配置するように選択できます。

   結果: テスト実行が完了すると、その郵便番号の地域の現在の気温が宛先ビジネス・オブジェクトに表示されます。

図 82 は、ソース・ビジネス・オブジェクト属性の郵便番号 94010 が、宛先ビジネス・オブジェクト属性では 59 度に変換されたことを示します。

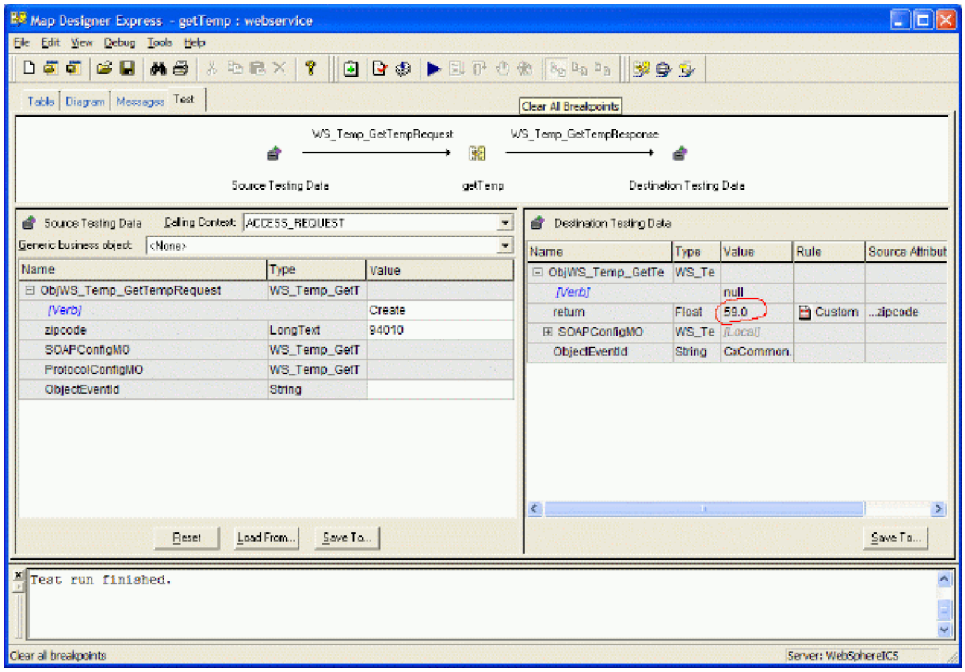

図 82. テスト表示の結果

変数の使用

変数は Java コード内の値のプレースホルダー (置き換え) です。このセクションでは、変換コードでの変数の使用について、以下の内容を説明します。

- 『生成されたビジネス・オブジェクト変数および属性の使用』
- 187 ページの『一時変数の作成』

生成されたビジネス・オブジェクト変数および属性の使用

このセクションでは、ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトのビジネス・オブジェクト変数の生成について説明します。マップにビジネス・オブジェクトを追加する場合、Map Designer Express により以下のものが自動的に生成されます。

- インスタンス名

  Map Designer Express が生成するインスタンス名はシステム宣言ローカル変数で、マッピング・コードでこのビジネス・オブジェクトを参照するために使用することができます。これには、文字 `Obj` が前に付加され、その直後にビジネス・オブジェクト定義の名前が続きます。

例: マップに Customer を追加すると、そのインスタンス名は ObjCustomer となります。Map Designer Express はソースと宛先ビジネス・オブジェクトの両方のインスタンス名を生成します。

Activity Editor でコードを作成する場合、インスタンス名を使用してビジネス・オブジェクトとその属性を参照します。

- ビジネス・オブジェクト配列内のビジネス・オブジェクトのインデックス (ビジネス・オブジェクトが複数カーディナリティーの場合)

  ビジネス・オブジェクト・インデックスは、このソースまたは宛先ビジネス・オブジェクトの順序を表します。マップのソースと宛先ビジネス・オブジェクトの最初のインデックス番号はゼロです。追加されたビジネス・オブジェクトは次の使用可能なインデックス番号 (1、2、3 など) となります。

  マップが実行されると、インデックス番号は、マップ (ソース・ビジネス・オブジェクト) に渡されるか、またはマップ (宛先ビジネス・オブジェクト) から戻される配列内のビジネス・オブジェクトの位置を表します。

Map Designer Express により、この情報が次の場所に表示されます。

- 「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブ

  ビジネス・オブジェクト・ウィンドウのタイトル・バーを右マウス・ボタンでクリックして、コンテキスト・メニューから「プロパティー」を選択します。「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブが表示され、選択したビジネス・オブジェクトがリスト内で強調表示されます。このタブには、ビジネス・オブジェクト配列内のインスタンス名とインデックスの両方が表示されます (ビジネス・オブジェクトが複数カーディナリティーの場合)。
- 「テーブル」タブ: ビジネス・オブジェクト・ペイン
- 「ダイアグラム」タブ: ビジネス・オブジェクト・ウィンドウのタイトル・バー。フォーマットは以下のとおりです。

  タイトル・バーにビジネス・オブジェクトのインスタンス名が表示されます。

注: オプション「**マップを定義: ビジネス・オブジェクト・インスタンス名を表示**」を使用して、Map Designer Express がソースおよび宛先ビジネス・オブジェクトの変数名を表示するかどうかを指定することができます。デフォルトでは、このオプションは使用可能で、Map Designer Express により「テーブル」および「ダイアグラム」の両方のタブにこれらの変数名 (Obj*BusObj*) が表示されます。このオプションが使用不可の場合は、ソースおよび宛先ビジネス・オブジェクトの名前のみが表示されます。このオプションの設定は、「設定」ダイアログの「一般」タブで変更できます。詳細については、23 ページの『一般的な設定の指定』を参照してください。

ビジネス・オブジェクト変数の変更手順

「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブで、これらのビジネス・オブジェクト変数を変更できます (図 83 を参照)。

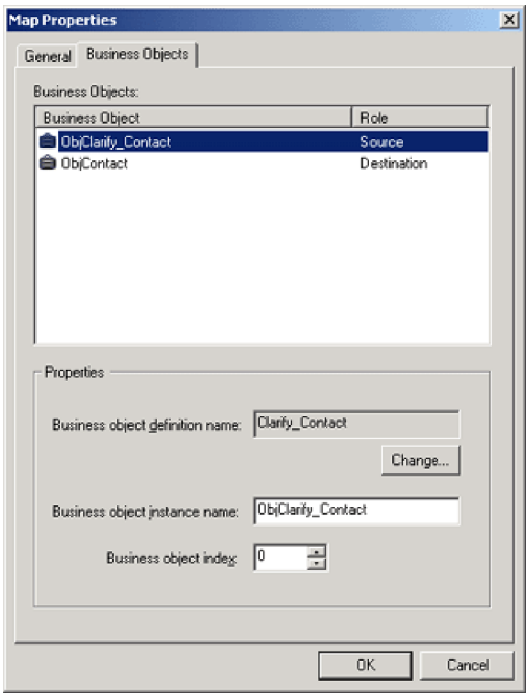

図 83. 「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブ

「マップ・プロパティー」ダイアログで、マップに含まれるソース・ビジネス・オブジェクトまたは宛先ビジネス・オブジェクトのビジネス・オブジェクト・タイプを変更するには、以下の手順を実行します。

1. マップを開きます。
2. 次のいずれかの方法で、「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブを表示します。
   - 「編集」メニューから「マップ・プロパティー」を選択します。
   - 「ダイアグラム」タブで、ビジネス・オブジェクト・ウィンドウを右マウス・ボタンでクリックして、コンテキスト・メニューから「プロパティー」を選択します。

     結果: 「マップ・プロパティー」ダイアログ・ボックスの「一般」タブが表示されます。「ビジネス・オブジェクト」タブをクリックします。

   「マップ・プロパティー」ダイアログを表示する他の方法については、64 ページの『マップ・プロパティー情報の指定』を参照してください。
3. 変更するビジネス・オブジェクト・タイプを選択します。
4. 「ビジネス・オブジェクト・タイプ」の下の「変更」プッシュボタンをクリックします。
5. 新しいタイプを選択します。
6. 「OK」をクリックして、「Select Business Object」ダイアログを閉じます。

7. 「OK」をクリックして、「マップ・プロパティー」ダイアログを閉じます。

注: 無効な変換規則は削除されます。

ビジネス・オブジェクト属性の参照

Map Designer Express が生成するビジネス・オブジェクト変数を使用して、以下のようにビジネス・オブジェクトとその属性を参照します。

- ソースまたは宛先ビジネス・オブジェクト内の属性を参照するには、ビジネス・オブジェクト名の後にピリオド (.) を分離文字として指定し、属性名を続けます。

  例:

  Obj*BusObjName.AttrName*

- 子ビジネス・オブジェクト内の属性を参照するには、子のビジネス・オブジェクト名と子の属性名を使用します。

  例: 次の例では、ObjPsft_Employee 内の OrigHireDate 属性の値を、ObjEmployee の子である EmployeeHR_Misc の HireDate 属性に設定します。

  ```
  ObjPsft_Employee.set("OrigHireDate",
     ObjEmployee.getString("EmployeeHR_Misc.HireDate"));
  ```

- 子ビジネス・オブジェクトが n カーディナリティーの場合 (複数の子インスタンスを親に関連付けできる)、子ビジネス・オブジェクトのインデックス番号を指定する必要があります。

  例: ObjCustomer の複数カーディナリティーの子である、Address の TimeZone 属性の値を設定します。

  ```
  ObjCustomer.set("Address[0].TimeZone",
  ObjSAP_Customer.getString("TimeZone"));
  ```

一時変数の作成

Map Designer Express により、マップ全体の変換ステップでアクセスできる一時変数が作成できるようになります。つまり、一時変数はマップに対してグローバルです。例えば、ある変換ステップで値を計算し、一時変数に格納して、別の変換ステップでその変数を参照することができます。これは、ある計算が繰り返し実行される場合に便利です。つまり、計算を 1 回実行して、結果を一時変数に格納し、必要に応じて (例えば、移動変換で) 値を検索できます。

一時ビジネス・オブジェクト変数の作成手順

一時変数は、一時ビジネス・オブジェクト内に定義されます。一時ビジネス・オブジェクト変数を作成するには、以下の手順を実行します。

1. 「編集」メニューから「ビジネス・オブジェクトを追加」を選択します。

   結果: 「ビジネス・オブジェクト・プロパティーを追加」ダイアログ・ボックスの「一般」タブが表示されます。

   「ビジネス・オブジェクトを追加」ダイアログを表示する他の方法については、38 ページの『「ビジネス・オブジェクトを追加」ダイアログからビジネス・オブジェクトを指定する手順』を参照してください。

2. 「一時」タブをクリックします。ここで一時変数を定義します。図 84 に、「ビジネス・オブジェクトを追加」ダイアログの「一時」タブを示します。「名前」フィールドには、Map Designer Express によって生成された一時ビジネス・オブジェクトの名前が表示されます。最初に生成される名前は、ObjTemporary です。このフィールドは読み取り専用です。

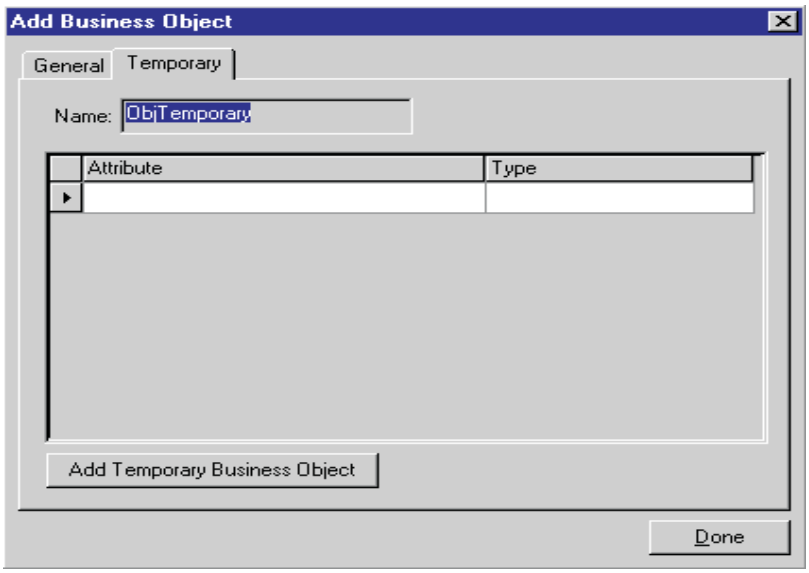

図 84. 「ビジネス・オブジェクトを追加」ダイアログの「一時」タブ

3. 「属性」フィールド内をクリックします。

   結果: 新しい行が変数テーブルに表示されます。一時変数の名前を入力します。

   注: 同じ名前で 2 つの一時変数を作成しないでください。
4. 「タイプ」フィールドをクリックして、プルダウン・リストから一時変数のデータ型を選択します。

   注: InterChange Server Express データ型スキームとの互換性を保つために、一時変数の型はすべて内部型 String です。「ビジネス・オブジェクトを追加」ダイアログで指定されたデータ型は、変数の初期化方法のみに影響します。カスタム Java コードを作成して値を一時変数に代入する場合は、最初にその値を String に変換する必要があります。
5. マップで必要な各一時変数について、ステップ 3 と 4 を繰り返します。
6. 「一時ビジネス・オブジェクトを追加」ボタンをクリックします。
7. 別の一時ビジネス・オブジェクトを定義するか、または「Done」をクリックして終了します。

変換ステップでの一時ビジネス・オブジェクト変数の使用手順

次の方法で、一時変数を変換ステップで使用します。

- 「ダイアグラム」タブから
  1. 一時属性の行見出し (左端) 列をクリックします。

2. Ctrl キーを押したまま変数を属性の上にドラッグして、変数値を属性にコピーします。

- Activity Editor の「Java」タブで、属性に変換ステップの変数名を使用します。

要確認: 一時変数はグローバル変数なので、「マップ・インスタンスを再利用」オプションを使用する場合は、一時変数を null に明示的に初期化する必要があります。明示的に初期化しないと、マップ・インスタンスの前の実行の一時変数の値が同じマップのその後の実行で、一時変数の値として誤って使用されることがあります。「マップ・インスタンスを再利用」オプションを使用しない 場合、マップの分離された呼び出し間に InterChange Server Express システムにより一時変数が自動的に初期化されます。

結果: Map Designer Express により一時ビジネス・オブジェクトが作成されると、マップの他のビジネス・オブジェクトと共にこのビジネス・オブジェクトが「テーブル」および「ダイアグラム」タブに表示されます。以下のようになります。

- 「テーブル」タブから
  - 「ビジネス・オブジェクト」ペインに、一時ビジネス・オブジェクト用の新しい領域が追加されます。一時ビジネス・オブジェクトの名前を右マウス・ボタンでクリックすると、このビジネス・オブジェクトを編集し、削除するためのオプションを指定するコンテキスト・メニューが開きます。
  - 一時ビジネス・オブジェクトとその属性が、属性変換テーブルの Source Attribute と Dest. Attribute 列のコンボ・ボックスに表示されます。
- 「ダイアグラム」タブから。マップ・ワークスペースに一時ビジネス・オブジェクト用の新しいビジネス・オブジェクト・ウィンドウが追加されます。

  この一時ビジネス・オブジェクト・ウィンドウには、ビジネス・オブジェクト・ウィンドウと同じ多くの特性があります。作成した変数は、ビジネス・オブジェクトの属性の場合のように、変数テーブルに表示されます。このビジネス・オブジェクト・ウィンドウには「規則」と「コメント」列があり、そこで一時変数の変換コードとコメントをそれぞれ追加できます。

  一時ビジネス・オブジェクト・ウィンドウのタイトル・バー内を右マウス・ボタンでクリックすると、このビジネス・オブジェクトとそのプロパティーを編集し、削除するためのオプションを指定するコンテキスト・メニューを表示できます。次のいずれかの方法で、変数の値を指定します。
  - 変数の値を戻すコードを入力するには、「規則」列をダブルクリックして、該当する変換規則を選択し、「コードを編集」をクリックして Activity Editor でコードを入力します。
  - ビジネス・オブジェクト属性から変数に値をコピーするには、Ctrl キーを押したまま属性を変数名の上にドラッグします。属性を分割して変数に結合することもできます。

注: 一時ビジネス・オブジェクトは、「マップ・プロパティー」ダイアログの「ビジネス・オブジェクト」タブにも表示されます。

変数の宣言

ヒント: 変数を宣言する場合は、以下のヒントに留意してください。

- 現行属性にローカルな変数 (他のすべての属性に不可視) を作成する場合、現行属性の変換ステップの先頭で宣言します (Activity Editor を使用)。
- 現行マップにグローバルな変数 (すべての属性に可視) を作成する場合は、「マップ・プロパティー」ダイアログの「一般」タブの「マップ・ローカル宣言ブロック」セクションで宣言します。これらの変数に値を割り当てるコードを作成する場合は、宛先オブジェクトの最初の属性の Activity Editor の先頭でこれを行います (実行順序の指定に従う)。

その他の属性の変換方法

属性の変換は、以下の方法で対話式に行うことができます。

- Map Designer Express のみ を使用: 標準変換の 1 つを作成

  41 ページの表 14 に、Map Designer Express がコードを生成できる標準変換をリストします。
- コードの変更および拡張のために Map Designer Express と Activity Editor を併用: カスタム変換を作成

  次のいずれでもカスタム変換を作成できます。
  - 標準変換を作成し、Activity Editor を開いて生成されたコードを変更する方法。標準変換をカスタマイズ後、Map Designer Express により変換規則列に青のイタリック・フォントで変換タイプが表示されます。
  - カスタム変換を作成し、Activity Editor を開いて変換を定義する方法。カスタム変換を作成後、Map Designer Express により変換規則列に黒のフォントでキーワード `Custom` が表示されます。詳細については、54 ページの『カスタム変換の作成』を参照してください。

このセクションでは、次のカスタム変換のインプリメント方法を説明します。

- 『内容ベースのロジック』
- 195 ページの『日付の形式設定』
- 199 ページの『式ビルダーを使用したストリング変換』

内容ベースのロジック

```
Customer.CustomerStatus = 'Inactive'
if SAP_CustomerMaster.DeleteInd = 'X'.
```

あるいは、`CustomerStatus = 'Active'` です。

カスタム変換を作成し、すべてのコードを作成して内容ベースのロジックを自分でインプリメントすることができます。しかし、より適切な方法は、`DeleteInd` (`SAP_CustomerMaster` オブジェクト内) と `CustomerStatus` 間に移動変換を作成することから開始する方法です (属性の移動手順については、43 ページの『ソース属性の宛先属性へのコピー』を参照)。

この結果、Map Designer Express により移動変換が生成されます。サンプル・コードを以下に示します。

```
{
   Object _cw_CpBTBSourceValue = null;

   //
   // RETRIEVE SOURCE
   // ---------------
   //
   // Retrieve the source value from the source business object and
   // place it in a local variable for code safety.
   //
   _cw_CpBTBSourceValue = ObjSAP_CustomerMaster.get("DeleteInd");

   //
   // SET DESTINATION
   // ---------------
   //
   // Put the source value into the destination business object
   // attribute.
   //
   {
      Object _cw_SetSrcVal = _cw_CpBTBSourceValue;
      BusObj _cw_SetDestBusObj = ObjCustomer;
      String _cw_SetDestAttr = "CustomerStatus";

      //
      // Set the destination value only if neither
      // source nor destination is null.
      //
      if ((_cw_SetSrcVal != null) && (_cw_SetDestBusObj != null))
      {
         if (dataValidationLevel >= 1)
         {
            if (!_cw_SetDestBusObj.validData(_cw_SetDestAttr, _cw_SetSrcVal))
            {
               String warningMessage =
"Invalid data encountered when attempting to set the value of the
¥'_cw_SetDestAttr¥' attribute of BusObj ¥'_cw_SetDestBusObj¥' while running
map ¥'" + getName() + "¥'.  The invalid value was ¥'" + _cw_SetSrcVal
+ "\'.";
               //
               // Log a warning about this failure.
               //
               logWarning(warningMessage);
               if (failOnInvalidData)
               {
                  //
                  // Fail the map execution with a warning message.
                  //
                  throw new MapFailureException(warningMessage);
               }
            }
         }
// SECTION THAT NEEDS TO BE UPDATED WITH THE LOGIC

      if (_cw_SetSrcVal != null)
         {
         if (_cw_SetSrcVal instanceof BusObj)
            {
            //
            // Since BusObjs are not immutable, we need to make
            // a copy of the source object before actually
            // putting it into the destination attribute.
            //
            _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
               ((BusObj)_cw_SetSrcVal).duplicate());
            }
         else if (_cw_SetSrcVal instanceof BusObjArray)
```

```
            {
            //
            // Since BusObjs are not immutable, we need to make
            // a copy of the source object before actually
            // putting it into the destination attribute.
            //
            _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
               ((BusObjArray)_cw_SetSrcVal).duplicate());
            }
         else
            {
            //
            // Since our version of simple data types are immutable in
            // Java (Strings included), we do not have to make a copy
            // of the source value here.
            //
            _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, _cw_SetSrcVal);
            }
         }
      }
   }
}

// HERE IS THE MODIFIED CODE

   if (_cw_SetSrcVal != null)
      {
      if (((String)_cw_SetSrcVal).equals("X"))
         _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, "Inactive");
      else
         _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, "Active");
      }

      }
   }
}
```

Activity Editor の「ファイル」メニューから、「保管」を選択して変更を保管します。

動詞ベース・ロジック

`Customer.CustomerStatus = 'Inactive' if Verb = 'Create'` あるいは、`CustomerStatus = 'Active'` です。

190 ページの『内容ベースのロジック』の手順に従います。ただし、以下の条件ステートメントを使用して、生成されたコードのセクションを置き換えます。

```
// HERE IS THE MODIFIED CODE
if (_cw_SetSrcVal != null)
   {
   if (ObjSAP_CustomerMaster.getVerb() .equalsIgnoreCase("Create"))
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, "Inactive");
   else
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,"Active");
```

ソース・データが不足している場合のデフォルト値の指定

`SAP_CustomerMaster.State` を `Customer.CustomerAddress.State` にマップします。SAP の状態が不足している場合は、デフォルトで `CA` に設定します。

宛先属性にソース値を設定される前に、ソース属性が null でないかどうかをチェックします。

例: この例では、ソース・データが不足していると、宛先属性にデフォルト値が設定されます。

SAP_CustomerMaster.State を Customer.CustomerAddress.State に移動することから開始します。条件ステートメントを以下のように変更します。

```
// HERE IS THE MODIFIED CODE
if (_cw_SetSrcVal != null)
   _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, _cw_SetSrcVal);
else
   _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, "CA");
_cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, "CA");
```

ヒント: ソース属性を宛先属性にコピーすることからコーディングを開始した場合は、デフォルト値を設定するコード行は 2 回入力する必要があります。これは、ソース属性が宛先属性に移動されるときに生成されるコードで、ソース属性が null でないことを 2 回チェックするからです。したがって、デフォルト値を入力する必要があります。

呼び出しコンテキストに基づくロジック

呼び出しコンテキストの値をチェックする必要がある場合は、String 型である組み込み変数 strInitiator を使用します。

例: 呼び出しコンテキストが EVENT_DELIVERY かどうかをチェックするには、以下のステートメントを使用します。

```
if (strInitiator.equals(MapExeContext.EVENT_DELIVERY))
//rest of the code
```

ソース・データが不足する場合は、マップを強制的に失敗させる

SAP_CustomerMaster.State を Customer.CustomerAddress.State にマップします。SAP の状態が不足している場合は、マップの実行を停止します。

SAP_CustomerMaster.State を Customer.CustomerAddress.State に移動することから開始します。その後、1 つの if() 文を追加して、SAP の State 属性が null かどうかをチェックします。生成されるコードは、以下のようになります。

```
{
   Object _cw_CpBTBSourceValue = null;

   //
   // RETRIEVE SOURCE
   // ---------------
   //
   // Retrieve the source value from the source business object and
   // place it in a local variable for code safety.
   //
   _cw_CpBTBSourceValue = ObjSAP_CustomerMaster.get("State");

   //
   // SET DESTINATION
   // ---------------
   //
   // Put the source value into the destination business object
   // attribute.
   //
   {
      Object _cw_SetSrcVal = _cw_CpBTBSourceValue;
      BusObj _cw_SetDestBusObj = ObjCustomer;
      String _cw_SetDestAttr = "CustomerAddress.State";
```

```
   // New code
   if (_cw_SetSrcVal == null)
      {
      String errorMessage = "Data in the state attribute is missing";
      logError(errorMessage);
      //
      // Fail the map execution with a warning message.
      //
      throw new MapFailureException(errorMessage);
      }
   // End of new code

   //
   // Set the destination value only if neither
   // source nor destination is null.
   //
   else if ((_cw_SetSrcVal != null) && (_cw_SetDestBusObj != null))
      {
      if (dataValidationLevel >= 1)
         {
         if (!ObjCustomer.validData("CustomerAddress.State", _cw_SetSrcVal))
            {
            String warningMessage =
"Invalid data encountered when attempting to set the value of
the ¥"CustomerAddress.State¥" attribute of BusObj ¥'ObjCustomer¥'
while running map ¥'" + getName() + "¥'.  The invalid value
was ¥'" + _cw_SetSrcVal + "¥'.";

            //
            // Log a warning about this failure.
            //
            logWarning(warningMessage);
            if (failOnInvalidData)
               {
               //
               // Fail the map execution with a warning message.
               //
               throw new MapFailureException(warningMessage);
               }
            }
         }

      if (_cw_SetSrcVal != null)
         _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr, _cw_SetSrcVal);

      }
   }
}
```

マップが実行されて、State 属性が設定されていないと、ダイアログ・ウィンドウとサーバー画面の両方にエラー・メッセージが表示されます。サーバー画面のメッセージは、次のようになります。

```
[1999/09/22 17:58:51.008] [Server]
Sub_SaCwCustomerMaster: Error Data
in the state attribute is missing
```

マップは実行を停止します。

ヒント: 以下のコードを使用して、マップが失敗したシステムで生成された実際のエラー・メッセージを表示します。

```
try
   {
   // your code
   }
```

```
catch (Exception e)
   {
  throw new MapFailureException(e.toString());
   }
```

メッセージ・ファイルからのメッセージのロギング

193 ページの『ソース・データが不足する場合は、マップを強制的に失敗させる』のサンプルで、エラー・メッセージを画面に表示するための logError() メソッドの使用方法に注意してください。汎用メッセージ・ファイル CWMapMessages txt (¥DLMs¥messages に保管されています) に用意されているメッセージを使用できます。メッセージ・ファイルの詳細については、537 ページの『付録 A. メッセージ・ファイル』を参照してください。

CWMapMessages.txt のフォーマットは、以下のとおりです。

```
# critical data is missing error
10
Data in the {1} attribute is missing. Map execution stopped.
# Another error
11
Another error message
```

{1} の代わりに単語 State が、表示される必要のあるエラーに直接対応して使用できることに注意してください。ただし、特定の単語を直接インプリメントすると、メッセージの汎用性は失われます。別の属性にも重大なデータが存在する場合は、その特定の属性に固有のメッセージ・ファイルに、別のメッセージが必要です。

注: CWMapMessages.txt ファイルは変更しないでください。独自のメッセージを作成する必要がある場合は、Map Designer Express の「メッセージ」タブに入力します。Map Designer により *mapName_locale*.txt という名前 (例: mapName_en_US.txt) のマップ固有のエラー・メッセージ・ファイルが作成されます (ここで、*mapName* はマップと同じ名前です)。サーバーにより、展開後にこのメッセージ・ファイルが ¥DLMs¥messages ディレクトリー内に保管されます。

メッセージ番号 10 を表示するには、以下のコードを使用します。

```
logWarning(10, "State");
```

単語 State により、メッセージ・テキスト内の {1} が置換されます。

次のメッセージが InterChange Server Express ログ・ファイルに表示されます。

```
[1999/09/23 10:17:43.648] [Server]
Sub_SaCwCustomerMaster:
Warning 10: Data in the State attribute is missing.
Map execution stopped.
```

日付の形式設定

マッピング API では、DtpDate クラスに日付の形式設定のメソッドを用意しています。表 61 に、日付の形式設定メソッドを要約します。

表 61. DtpDate クラスの日付の形式設定メソッド

日付の形式設定	DtpDate メソッド
日付から月名を取得	getMonth(), getShortMonth()

表 61. *DtpDate* クラスの日付の形式設定メソッド (続き)

日付の形式設定	DtpDate メソッド
日付から月の数値を取得	getIntMonth(), getNumericMonth()
日を取得	getDayOfMonth(), getIntDay()
曜日を取得	getDayOfWeek(), getIntDayOfWeek()
日付から年を取得	getYear(), getIntYear()
時間値の取得	getHours()
日付から分値を取得	getMinutes(), getIntMinutes()
日付から秒値を取得	getSeconds(), getIntSeconds()
日付からミリ秒値を取得	getMSSince1970()
リストから最も古い日付を取得	getMinDate(), getMinDateBO()
リストから最新の日付を取得	getMaxDate(), getMaxDateBO()
指定形式に従った日付を構文解析	DtpDate()
指定またはデフォルトの形式で日付を取得	toString()
IBM 汎用日付形式へ日付を再形式設定	getCWDate()
日付への日数の追加	addDays()
日付への平日の数の追加	addWeekdays()
日付への年数の追加	addYears()
日付間の日数の計算	calcDays()
日付間の平日の数の計算	calcWeekdays()
日付の比較	after(), before()
完全な月名を使用	get12MonthNames(), set12MonthNames(), set12MonthNamesToDefault()
短い月名を使用	get12ShortMonthNames(), set12ShortMonthNames(), set12ShortMonthNamesToDefault()
曜日名の使用	get7DayNames(), set7DayNames(), set7DayNamesToDefault()

ヒント: 常に DtpDateException をキャッチします。これにより、日付形式が無効な場合、メッセージが InterChange Server Express ログに送信されることが保証されます。

汎用日付形式の使用

IBM では、汎用ビジネス・オブジェクト内の次の日付形式を使用します。

YYYYMMDD HHMMSS

この形式は、*汎用日付形式* と呼ばれます。

汎用日付形式への変換手順: アプリケーション固有の日付をこの汎用形式に変換するには、DtpDate クラスの getCWDate() メソッドを使用します。

例: 例えば、SAP 日付属性を汎用日付にマップするには、ソース属性 (SAP 日付ストリング) を宛先属性 (汎用日付ストリング) にコピーします。

1. DtpDate オブジェクトを作成して汎用日付を保持します。

   DtpDate() コンストラクターで SAP 日付ストリング (*YYYYMMDD*) を構文解析して、ソース属性 (SAP 日付ストリング) を新しい DtpDate オブジェクト (汎用日付ストリング) にコピーします。
2. getCWDate() メソッドで、SAP 日付ストリングを汎用日付形式 (*YYYYMMDD HHMMSS*) に変換します。

3. setWithCreate() メソッドで、宛先属性を作成して、その値を汎用日付形式に初期化します。

このコードの最後のセクションは、次のようになります。

```
// HERE IS THE MODIFIED CODE
if (_cw_SetSrcVal != null)
   {
   try
      {
      DtpDate myDate = new DtpDate((String)_cw_SetSrcVal, "YMD");
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
      myDate.getCWDate());
      }
   catch (DtpDateException de)
      {
      logError(5501);
      logInfo(de.getMessage());
      }
   }
```

例: Clarify 日付属性を汎用日付にマップするには、ソース属性 (Clarify 日付ストリング) を宛先属性 (汎用日付ストリング) にコピーします。

1. DtpDate オブジェクトを作成して汎用日付を保持します。

   DtpDate() コンストラクターで Clarify 日付ストリング (*MM/DD/YYYY HH:MM:SS*) を構文解析して、ソース属性 (Clarify 日付ストリング) を新しい DtpDate オブジェクト (汎用日付ストリング) にコピーします。
2. getCWDate() メソッドで、Clarify 日付ストリングを汎用日付形式 (*YYYYMMDD HHMMSS*) に変換します。
3. setWithCreate() メソッドで、宛先属性を作成して、その値を汎用日付形式に初期化します。

次のコードは、Clarify 日付を汎用日付に変換します。

```
if (_cw_SetSrcVal != null)
   {
   try
      {
      DtpDate myDate = new DtpDate((String)_cw_SetSrcVal,
         "M/D/Y h:m:s");
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
         myDate.getCWDate());
      }
   catch (DtpDateException de)
      {
      logError(5501);
      logInfo(de.getMessage());
      }
   }
```

汎用日付を Clarify 形式に変換するには、次のコードを使用します。

```
if (_cw_SetSrcVal != null)
   {
   try
      {
      DtpDate myDate = new DtpDate((String)_cw_SetSrcVal, "YMD hms");
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
      myDate.toString("M/D/Y h:m:s", true));
      }
   catch (DtpDateException de)
```

```
      {
      logError(5501);
      logInfo(de.getMessage());
      }
}
```

汎用日付形式からの変換手順: 汎用日付形式から SAP 日付形式への変換は、次の手順で行います。

1. DtpDate オブジェクトを作成して SAP 日付を保持します。

   DtpDate() コンストラクターで汎用日付ストリング (*YYYYMMDD HHMMSS*) を構文解析して、ソース属性 (汎用日付ストリング) を新しい DtpDate オブジェクト (SAP 日付ストリング) にコピーします。
2. toString() メソッドで IBM 汎用日付形式を SAP 日付ストリング (*YYYYMMDD*) メソッドに変換します。
3. setWithCreate() メソッドで宛先属性を作成し、その値を SAP 日付形式に初期化します。

次のコード・フラグメントは、この汎用日付から SAP 日付への変換を示します。

```
// HERE IS THE MODIFIED CODE
if (_cw_SetSrcVal != null)
   {
   try
      {
      DtpDate myDate = new DtpDate((String)_cw_SetSrcVal, "YMD hms");
      _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
      myDate.toString("YMD"));
      }
   catch (DtpDateException de)
      {
      logError(5501);
      logInfo(de.getMessage());
      }
   }
```

現在日付の取得

現在のシステム日付で初期化される DtpDate オブジェクトを作成するには、パラメーターを指定しないで DtpDate() コンストラクターを使用します。

```
DtpDate()
```

例: 現在日付を汎用形式にマップするには、次のコードを使用します。

```
//get the current date
DtpDate myDate = new DtpDate();
// format the date to the destination object's format and map
_cw_SetDestBusObj.set(_cw_SetDestAttr, myDate.getCWDate());
```

現在日付をアプリケーション固有の形式に変換するには、次のコードを使用します。

```
//get the current date
DtpDate myDate = new DtpDate();
// format the date to the destination object's format and map
_cw_SetDestBusObj.set(_cw_SetDestAttr, myDate.toString(
   "application format");
```

式ビルダーを使用したストリング変換

変換コードを作成する場合、特定の属性の参照、ストリングの操作、または API メソッドの呼び出しのために、複雑な Java 式を作成する必要があることがあります。これらの式は、Activity Editor で手動で入力することも、式ビルダーを使用して式を対話式に作成することもできます。式ビルダーは、Activity Editor 内から使用できるユーティリティーです。

ヒント: 代わりに、Activity Editor のグラフィック表示を使用することもできます。

式ビルダーを表示するには、Activity Editor 内の式を挿入する位置にカーソルを合わせ、以下のいずれかを行います。

- 「ツール」メニューから「式ビルダー」を選択します。
- Java ツールバーで、「式ビルダー」ボタンをクリックします。
- Activity Editor ウィンドウの任意の場所を右マウス・ボタンでクリックして、コンテキスト・メニューから「式ビルダー」を選択します。

図 85 に、式ビルダーの主要コンポーネントを示します。

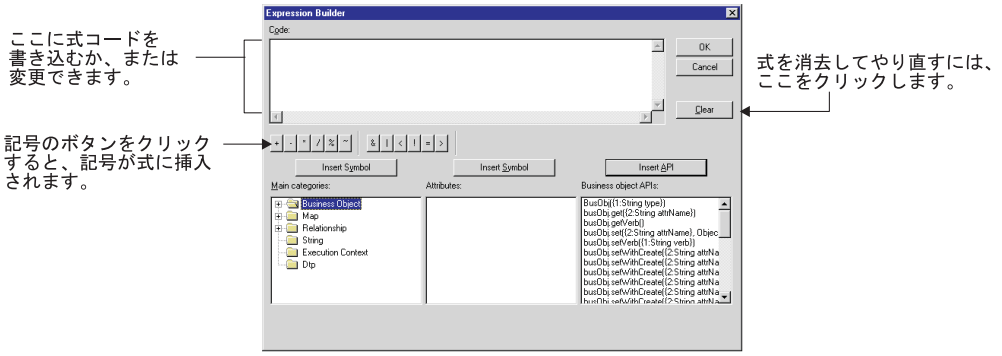

図 85. 「式ビルダー」ダイアログ

注: API リストでは、オブジェクトが存在する場合、番号 1 は左側のウィンドウから選択されたオブジェクトを参照し、番号 2 は中央のウィンドウから選択されたオブジェクトを参照します。API を挿入する場合は、括弧で変数を囲みます (<<変数>>)。括弧と変数を値で置き換える必要があります。

このセクションでは、式ビルダーを使用して次のストリング変換を実行する方法について説明します。

- 『大文字テキストへの変換手順』
- 200 ページの『ストリング処理メソッドの使用』

大文字テキストへの変換手順

`SAP_CustomerMaster.CustomerName` を `Customer.AccountOpenedBy` に移動して、すべての文字を大文字に変換します。

`toUpperCase()` ストリング処理関数を使用するには、次の手順を行います。

1. `SAP_CustomerMaster.CustomerName` から `Customer.AccountOpenedBy` への移動変換を実行します。

2. Customer.AccountOpenedBy に関連した変換規則列内をダブルクリックして、「Activity Editor」ウィンドウを表示します。Activity Editor で、宛先がソース・データに設定される位置までスクロールダウンします。それを変更して、次のコードのみを含むようにします。

```
if (_cw_SetSrcVal != null)
   {
   //
   // Since our version of simple data types are immutable in
   // Java (Strings included), we do not have to make a copy
   // of the source value here.
   //
   _cw_SetDestBusObj.setWithCreate(
   _cw_SetDestAttr, _cw_SetSrcVal);
   }
```

   _cw_SetSrcVal 変数には SAP.CustomerName が含まれ、_cw_SetDestAttr には AccountOpenedBy が含まれます。

3. CustomerName を大文字に変換してから、AccountOpenedB にコピーします。

   使用するメソッドがわかっている場合は、それを上記のコード行に追加します。そうでない場合は、式ビルダーを使用すると、正しいメソッドを見つけることができます。次の手順で行います。

   a. _cw_SetSrcVal をコピー (Ctrl+C) します。
   b. _cw_SetSrcVal を強調表示し (このコードは式ビルダーで生成されるコードで置き換えられます)、クリックしてツール・メニューから「式ビルダー」を選択します。
   c. 「メイン・カテゴリー」リストで「ストリング」を選択します。
   d. ストリング API のリストから toUpperCase() メソッドを選択します。
   e. 「Insert API」をクリックします。
   f. 上部ウィンドウにこのメソッドが表示されるときに、プレースホルダーである単語「string,」が、先程コピーした _cw_SetSrcVal で置き換えられます。
   g. 「OK」をクリックします。

      結果: メソッド呼び出しがコードに挿入されます。

   h. 「ファイル」メニューから「保管」オプションを選択して、コードを保管します。

ストリング処理メソッドの使用

次のストリング変換では、ストリング処理メソッド (length() や substring() など) を使用して、SAP_CustomerMaster.AddressLine1 を Customer.CustomerAddress.AddressLine1 と AddressLine2 に移動します。

- AddressLine1 の長さが 10 文字より少ない場合は、SAP AddressLine1 を CustomerAddress.AddressLine1 に移動し、AddressLine2 はマップしません。
- 長さが 10 文字以上の場合は、SAP AddressLine1 のコンマより後をすべて CustomerAddress.AddressLine2 にマップします。コンマが検出されない場合は、9 文字を AddressLine1 に、残りを AddressLine2 にマップします。

ストリング変換の実行手順: このストリング変換は、次の手順で行います。

1. AddressLine1 の CustomerAddress.AddressLine1 への移動変換を実行します。

2. AddressLine1 の Activity Editor ウィンドウを表示して、宛先がソース・データに設定される位置までスクロールダウンします。それを変更して、次のコードを含むようにします。

```
if (_cw_SetSrcVal != null)
   //
   // Since our version of simple data types are immutable in
   // Java (Strings included), we do not have to make a copy
   // of the source value here.
   //
   {
   _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
      _cw_SetSrcVal);
   }
```

   _cw_SetSrcVal 変数には SAP.AddressLine1 が含まれ、_cw_SetDestAttr には CustomerAddress.AddressLine1 が含まれます。

3. SAP.AddressLine1 の長さをチェックして、ストリング全体を CustomerAddress.AddressLine1 にマップするか、またはコンマに先行する SAP.AddressLine1 のサブストリングをマップします。

   コードは自分で変更することも式ビルダーを使用することもできます。どちらの場合も、String クラスの length()、substring(int, int)、substring(int)、および indexOf(int) メソッドを使用します。

4. コードを以下のように変更します。

```
if (_cw_SetSrcVal != null)
   {
   // first check the length of _cw_SetSrcVal
   if (((String)_cw_SetSrcVal).length() < 10)
      {
      // if it is less then 10, map it to AddressLine1
      _cw_SetDestBusObj.setWithCreate(
         _cw_SetDestAttr, _cw_SetSrcVal);
      }
   else
      {
      // if the length is not less than 10, search for comma
      int index = ((String)_cw_SetSrcVal).indexOf(",");

      // if comma is found, take a substring of _cw_SetSrcVal up to
      // the comma and map it to AddressLine1
      if (index != -1)
         _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
            ((String)_Cw_SetSrcVal).substring(0, index));
         // if comma is not found, take first 9 characters of
         // _cw_SetSrcVal and map them to AddressLine1
      else
         _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
            ((String)_Cw_SetSrcVal).substring(0, 9));
      }
   }
```

5. SAP.AddressLine1 を CustomerAddress.AddressLine2 に移動します。コードの下部を変更して、次のコードを組み込みます。

```
if (_cw_SetSrcVal != null)
   {
   // first check the length of _cw_SetSrcVal
   if (((String)_cw_SetSrcVal).length() >= 10)
      {
      // if the length is not less than 10, search for comma
      int index = ((String)_cw_SetSrcVal).indexOf(",");
```

```
        // if comma is found, take a substring of _cw_SetSrcVal after
        // the comma and map it to AddressLine2

        if (index != -1)
           _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
              ((String)_Cw_SetSrcVal).substring(index + 1));
           // if comma is not found, take all characters of
           // _cw_SetSrcVal starting at the 10th and map them to
           // AddressLine1

        else
           _cw_SetDestBusObj.setWithCreate(_cw_SetDestAttr,
                 ((String)_Cw_SetSrcVal).substring(9));
        }
     }
```

ソース・データの検証

自分でコードを作成する場合、ソース・データが null またはブランクでないようにします。ソース属性が null かどうかを検査するには、次のいずれかを使用します。

- `if (ObjSource.isNull("Attr"))`: 空の場合は `true` を戻します。
- `if (ObjSource.get("Attr") == null)`: *attr* が空の場合は `true` を戻します。

ソース属性が空のストリングかどうかを検査するには、以下を実行します。

- `if (ObjSource.isBlank("Attr"))`: ブランクの場合は true を戻します。
- `if (ObjSource.getString("Attr").length() == 0)`: String 型の *attr* がブランクの場合は true を戻します。

マップ・インスタンスの再利用

通常は、マップ開発システムではマップのインスタンスを作成して、ソースと宛先ビジネス・オブジェクト間の各データ変換を処理します。インスタンスが変換の処理を完了すると、そのリソースが解放されます。メモリー使用を少なくするために、IBM システムではマップ・インスタンスをキャッシングし、後で同じタイプのマップがインスタンス化される場合に再使用してインスタンスをリサイクルします。IBM システムで既存のマップ・インスタンスをリサイクルできると、マップ・インスタンス化のオーバーヘッドを回避できるので、システム全体のパフォーマンスとメモリー使用が向上します。

制約事項: マップ開発システムではマップ・インスタンスを自動的にキャッシュします。つまり、マップ・インスタンスはデフォルトで「マップ・インスタンスを再利用」オプションを使用します。しかし、「マップ・インスタンスを再利用」オプションでは、マップに次のプログラミング要件が必要になります。

- マップ・コードでグローバル変数を使用しないようにします。

  グローバル変数とは、「マップ・プロパティー」ダイアログの「一般」タブの「**マップ・ローカル宣言ブロック**」で宣言する変数です。
- マップでグローバル変数が必要な場合は、宣言時にこれらのグローバル変数を初期化しないようにします。代わりに、グローバル変数がマップ・ノード、できればマップ内の最初の変換 (属性) ノードで、常に初期化されるようにします。

重要: 最初の変換ノードで初期化されない グローバル変数を含むマップは、インスタンスが再使用されるときにキャッシュされたマップ・インスタンス内の変

数値が持続されるため、安全にリサイクルできません。キャッシュされたマップ・インスタンスが再使用されて実行が開始される際に、各グローバル変数にはマップ・インスタンスが前に使用されたときの最後の値が含まれています。

マップを前の制約事項に適合するように定義することができない場合は、このマップの「マップ・インスタンスを再利用」オプションを使用不可にする必要があります。このオプションを使用不可にするには、System Manager のマップの「マップ・プロパティー」ウィンドウに表示される「**マップ・インスタンスを再利用**」ボックスのチェック・マークを解除します。このウィンドウでマップ・インスタンス・プールのサイズを指定することもできます。

注: マップをサーバーに配置しても、ランタイム・インスタンスは更新されません。これらのマップ・プロパティーをサーバー・コンポーネント管理ビューから動的に更新するには、マップを右クリックして、コンテキスト・メニューからプロパティーを選択します。変更が自動的にサーバーに更新されます。

例外処理

例外 とは、マップ内で明示的に処理されない場合に、マップの実行を停止する事象の発生です。マップの実行中に、実行時例外が発生することがあります。カスタム変換規則を定義する場合は、「Catch Error」関数ブロックを使用すれば、実行時例外をトラップできます。特定の例外をキャッチすると、この例外の処理方法を判別できます。

関係例外

マップに関係を使用すると、いくつかの例外が発生することがあります。これらの例外はすべて `RelationshipRuntimeException` のサブクラスです。例外の種類は特に考慮せず、単にすべてをキャッチする場合は、`RelationshipRuntimeException` をキャッチします。あるいは、特定クラスの次のいずれの例外でもキャッチできます。

- `RelationshipRuntimeDataAccessException`: リレーションシップ・データベースにアクセス中に問題が発生した場合にスローされます。この例外は、`Relationship` または `Participant` クラスからのどのメソッド呼び出しでもキャッチできます。
- `RelationshipRuntimeDuplicateIdentityEntryException`: 既存の関係インスタンスと同じ関係インスタンス ID を持つ一致関係に参加者を追加しようとした場合にスローされます。この例外は、`addMyChildren()` および `create()` メソッド呼び出しでキャッチできます。
- `RelationshipRuntimeUserErrorException`: 抽象例外です。この例外は、`RelationshipRuntimeMetaDataErrorException` または `RelationshipRuntimeGeneralUserErrorException` が発生した場合にのみスローされます。この例外は、マップ開発中に `Relationship` または `Participant` クラスのどのメソッド呼び出しからでもキャッチできます。マップがデバッグされると、この例外のハンドラーを除去できます。
- `RelationshipRuntimeMetaDataErrorException`: 参加者インスタンスに関連付けられているメタデータ (関係名や参加定義名など) が処理されているときにエラーが

発生した場合にスローされます。この例外は、参加者インスタンスを追加、変更、または削除するどのメソッド呼び出しでもキャッチできます。

- `RelationshipRuntimeGeneralUserErrorException` : `Relationship` または `Participant` クラス・メソッドの呼び出しで指定されたランタイム・データにエラーが存在する場合にスローされます。

  例: 間違ったタイプのビジネス・オブジェクトを `create()` メソッドに渡すとスローされます。

図 86 に、関係の実行時例外の階層を示します。キャッチするどの例外も、下位の階層の例外を自動的にキャッチします。しかし、下位の階層の例外がスローされた場合、特別にそれをキャッチしないと、どの例外がスローされたかを正確に識別することはできません。

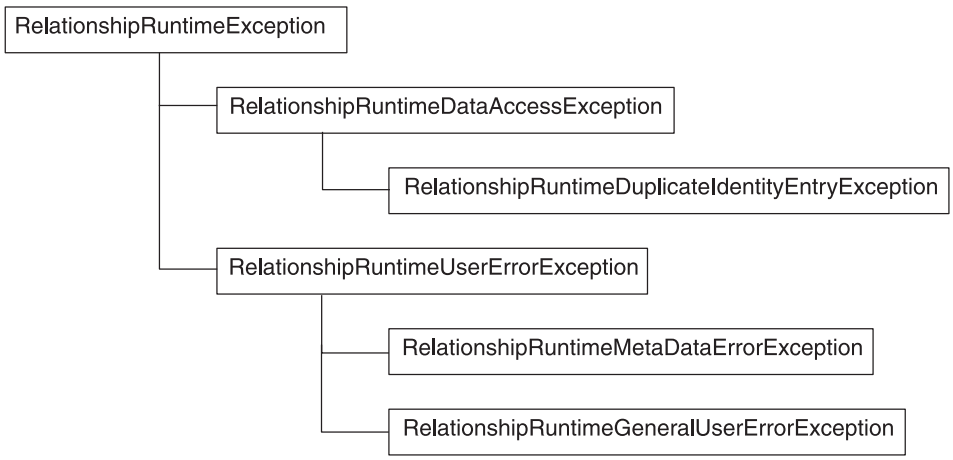

図 86. 関係の実行時例外

例: `RelationshipRuntimeUserErrorException` をキャッチすると、`RelationshipRuntimeMetaDataErrorException` と `RelationshipRuntimeGeneralUserErrorException` も自動的にキャッチします。しかし、例外を演算子 `instanceof` でテストしない限り、これらのうちのどれが実際にスローされたのかは簡単に認識できません。キャッチする例外は、例外をどのように処理するかにより異なります。

例: 重複する関係インスタンス ID の処理

`addMyChildren()` または `create()` メソッドを使用して参加者を一致関係に追加する場合、すでに存在する関係インスタンス ID を渡すと、`RelationshipRuntimeDuplicateIdentityEntryException` がスローされます。この例外では、重複を発生させた関係インスタンス ID を検索する `getInstanceId()` メソッドも使用します。

例: 次の例で、この例外をキャッチして ID を検索する方法を示します。

```
int instanceId;

try
   {
   instanceId = Relationship.create(myParticipantObj);
```

```
   }
catch (RelationshipRuntimeDuplicateIdentityEntryException rrdiee)
   {
   /*
   ** There already is a relationship instance with this
   ** entry, grab the instanceId from the exception
   */
   instanceId = rrdiee.getInstanceId();
   }
```

カスタム・データ検証レベルの作成

あるビジネス・オブジェクトから変換コードに基づく別のビジネス・オブジェクトに値がマップされると、誤ったデータとなることがあります。データ検証機能により、入力のビジネス・オブジェクトのデータが特定の規則に従って出力するビジネス・オブジェクトのデータに変換できない場合に、マップの各操作を検査して、エラーがログに記録されます。

例: マップによりソース・ビジネス・オブジェクトのストリング値が宛先ビジネス・オブジェクトの整数値に変換されるとします。この型変換は、入力ストリング値が整数値 (例えば、「1234」は整数 1234 を表します) を表す場合には、正しく動作します。しかし、ストリング値が整数値を表さない場合 (例えば、「ABCD」は無効なデータと示されます) は、正しく動作しません。

このセクションでは、マップのデータ検証レベルの使用について、以下の内容を説明します。

- 『データ検証レベルのコーディング』
- 206 ページの『データ検証レベルのテスト手順』

データ検証レベルのコーディング

マップ開発システムではデータ検証レベル 0 と 1 を定義します。レベル 2 以上はユーザーが定義に使用できます。表 62 に、データ検証レベルを要約します。

表 62. データ検証レベル

レベル	説明
0	デフォルト。データを検証しません。
1	IBM 定義のデータ型を検査します。
2 以上	ユーザー定義の検証を検査します。

カスタム検証レベルを作成するには、属性をダブルクリックして「Activity Editor」ウィンドウにそのコードを表示します。次の `if` 文は、無効なデータを検出した場合に例外をスローするコードを設定します。

```
if (dataValidationLevel >= 1)
```

データ検証レベルに関連付けるアクションを指定するには、独自の `if` 文を追加します。レベルと関連付けするデータ検証規則は、ビジネス・ロジックやアプリケーションに該当するもの、および必要とするもののうちいずれでも構いません。例えば、特定の属性のレベル 2 規則で、特定の値を検査し、存在しなければ属性をデフォルトに設定できます。

例: 次の例は、データ検証レベル 2 で、クレジット・カード番号が有効であることを検査します。

```
/*
** At data validation level 2, check credit card numbers resulting
** from data entry errors, etc.
*/
if (dataValidationLevel >= 2)
   {
   if (!CustUtility.validateCreditCard(ObjPurchaseReq.getString
         ("creditCardNumber")))
      {
      logWarning("Invalid credit card number sent through.");
      if (failOnInvalidData)
         {
         throw new MapFailureException(
            "Invalid credit card number.");
         }
      }
   }
```

データ検証レベルのテスト手順

マップのテストで、データ検証レベルを設定して適切に動作するかどうか確認できます。マップ・インスタンスのデータ検証レベルのテストは、次の手順で行います。

1. 以下のようにして、「マップ・プロパティー」ダイアログを表示します。

   「編集」メニューから「マップ・プロパティー」を選択します。「マップ・プロパティー」ダイアログを表示する他の方法については、64 ページの『マップ・プロパティー情報の指定』を参照してください。

   結果: 「マップ・プロパティー」ダイアログの「一般」タブが表示されます。

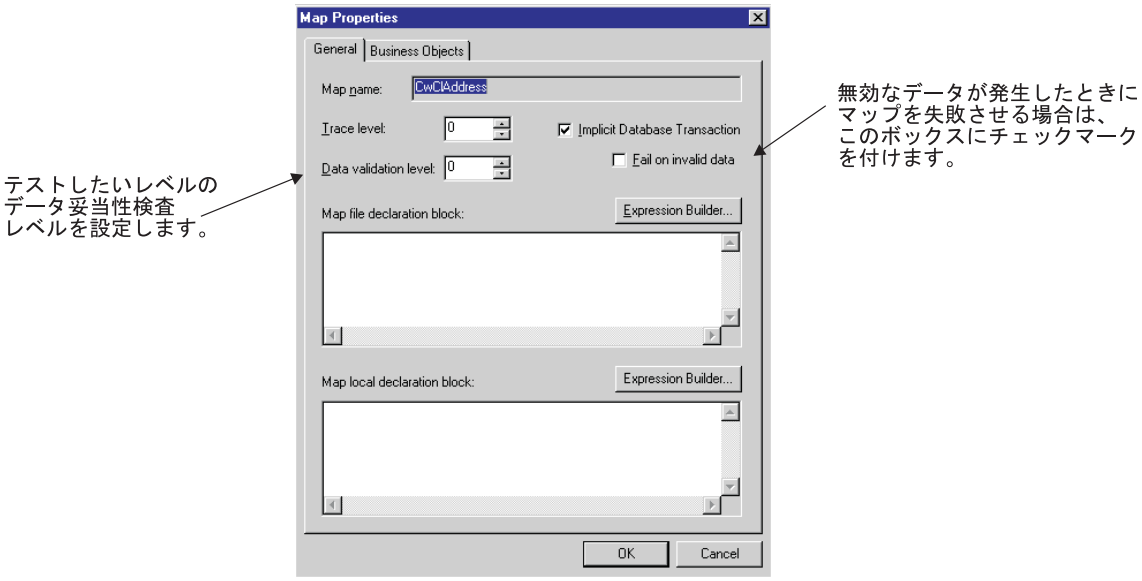

図 87. 「マップ・プロパティー」ダイアログの「一般」タブ

2. テストするデータ検証レベルを設定します。必要により、データが無効の場合はマップが失敗するように設定します。
3. 新しいデータ検証レベルを有効にするためにマップを停止し、再始動します。
   - マップのコードを変更した場合は、マップを再コンパイルします。再コンパイルすると、マップが自動的に再始動します。
   - マップ・コードを変更していない 場合は、再コンパイルする必要はありません。ただし、新しいデータ検証レベルを有効にするためには、マップを明示的に停止して再始動する必要があります。Service Manager の「コンポーネント」メニューを使用して、マップを停止して始動します。

注: データ検証レベルと、「マップ・プロパティー」ウィンドウおよびサーバー・コンポーネント管理ビューから無効なデータが提示された場合に失敗するかどうかも設定できます。

マップの実行コンテキストの理解

各マップ・インスタンスは、コネクター・コントローラーで設定される特定の実行コンテキスト 内で実行されます。マッピング API では、マップの実行コンテキストを `MapExeContext` クラスのインスタンスで表します。

Map Designer Express が生成するすべてのマップの場合、マップの実行コンテキストは `cwExecCtx` というシステム定義の変数を使用してアクセスできます。この変数は、Activity Editor の「変数」フォルダーで参照できます。あるいは実行コンテキストが必要なマッピング API メソッド (表 63 のメソッドなど) を呼び出すときに、この変数を参照できます。

表 63. マップの実行コンテキストが必要なマッピング API メソッド

目的	マッピング API メソッド	詳細情報の参照先
サブマップの呼び出し	runMap()	48 ページの『サブマップを使用した変換』
関係の維持	• addMyChildren() • deleteMyChildren() • updateMyChildren() • maintainChildVerb() • maintainSimpleIdentityRelationship() • maintainCompositeRelationship() • foreignKeyXref() • foreignKeyLookup()	281 ページの『第 8 章 関係のインプリメント』

表 64 に、マップの実行コンテキストで頻繁に必要になる 2 つの情報を示します。

表 64. コンテキスト情報の MapExeContext メソッド

実行コンテキスト情報	MapExeContext メソッド
呼び出しコンテキスト	`getInitiator()`, `setInitiator()`
オリジナル要求ビジネス・オブジェクト	`getOriginalRequestBO()`

このセクションでは、この両方のマップの実行コンテキスト情報について説明します。

注: さらに、MapExeContext の getConnName() と setConnName() メソッドを使用して、マップの実行コンテキストからコネクター名にアクセスできます。

呼び出しコンテキスト

呼び出しコンテキスト により、現行マップの実行の目的が示されます。関係属性を変換する場合、通常はマップの呼び出しコンテキストに基づいたアクションが必要です。表 65 に、呼び出しコンテキストの有効な定数をリストします。

表 65. 呼び出しコンテキスト

呼び出しコンテキスト定数	説明
EVENT_DELIVERY	マップされるソース・ビジネス・オブジェクトはアプリケーションからのイベントで、サブスクリプション要求に応答してコネクターから InterChange Server Express に送信されます (イベントにより起動されたフロー)。
ACCESS_REQUEST	マップされるソース・ビジネス・オブジェクトはアプリケーションから呼び出され、アクセス・クライアントから InterChange Server Express に送信されます (コール・トリガー・フロー)。
ACCESS_RESPONSE	マップされるソース・ビジネス・オブジェクトは、サブスクリプション送達要求に応答してアクセス・クライアントに送り返されます。
SERVICE_CALL_REQUEST	マップされるソース・ビジネス・オブジェクトは、InterChange Server Express からコネクターを経由して、アプリケーションに送信されます。
SERVICE_CALL_RESPONSE	マップされるソース・ビジネス・オブジェクトは、正常なサービス呼び出し要求の応答として、アプリケーションから InterChange Server Express に送り返されます。
SERVICE_CALL_FAILURE	マップされるソース・ビジネス・オブジェクトは、サービス呼び出し要求の失敗後、アプリケーションから InterChange Server Express に送り返されます。

これらの呼び出しコンテキストは、Map Designer Express で作成されるすべてのマップで使用可能な MapExeContext オブジェクト内の定数として参照できます。

例: SERVICE_CALL_REQUEST 呼び出しコンテキストを MapExeContext.SERVICE_CALL_REQUEST として参照します。

図 88 に、イベントにより起動されたフローでの各呼び出しコンテキストの発生条件を示します。イベントにより起動されたフローは、コネクターが InterChange Server Express 内のコラボレーションにイベントを送信したときに開始されます。

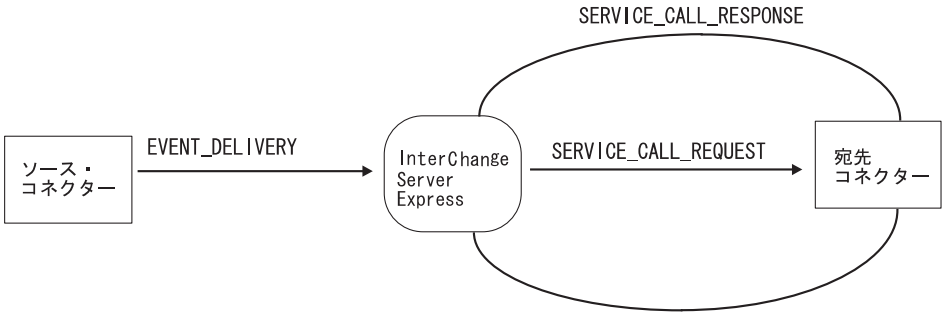

図 88. イベントにより起動されたフローの呼び出しコンテキスト

図 88 で示すように、コネクターから InterChange Server へのすべてのマッピング要求 (つまり、アプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのマップ) には、EVENT_DELIVERY の呼び出しコンテキストがあります。 InterChange Server からコネクターへのすべてのマッピング要求 (つまり、汎用ビジネス・オブジェクトからアプリケーション固有のビジネス・オブジェクトへのマップ) には、SERVICE_CALL_REQUEST の呼び出しコンテキストがあります。コラボレーションのサービス呼び出し要求に応答してコネクターにより送信されるマッピング要求には、SERVICE_CALL_RESPONSE または SERVICE_CALL_FAILURE のコンテキストがある場合があります。

図 89 に、コール・トリガー・フローでの各呼び出しコンテキストの発生条件を示します。コール・トリガー・フローは、アクセス・クライアントから InterChange Server Express のコラボレーションに直接サーバー・アクセス・インターフェース呼び出しが送信されると開始されます。

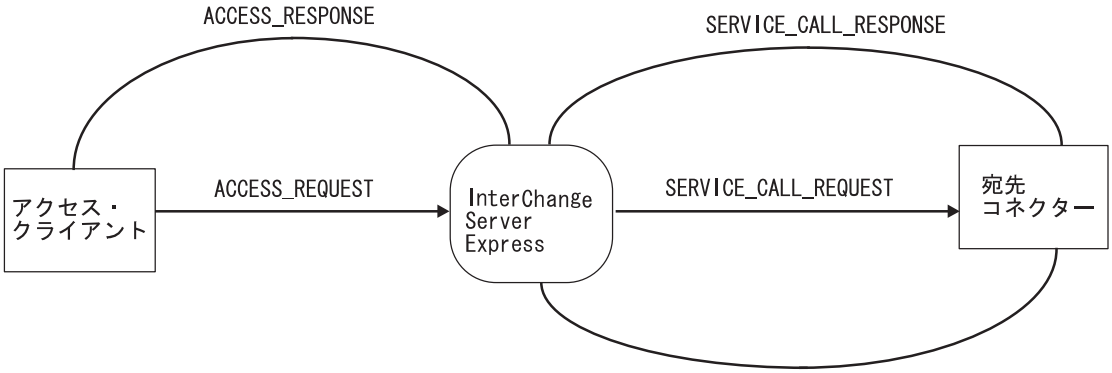

図 89. コール・トリガー・フローの呼び出しコンテキスト

図 89 で示すように、アクセス・クライアントから InterChange Server へのすべてのマッピング要求 (つまり、アプリケーション固有のビジネス・オブジェクトから汎用ビジネス・オブジェクトへのマップ) には、ACCESS_REQUEST の呼び出しコンテキストがあります。 InterChange Server Express からアクセス・クライアントへのす

べてのマッピング要求 (つまり、汎用ビジネス・オブジェクトからアプリケーション固有のビジネス・オブジェクトへのマップ) には、ACCESS_RESPONSE の呼び出しコンテキストがあります。

オリジナル要求ビジネス・オブジェクト

マップのコンテキストにおける異なる重要な部分は、*オリジナル要求ビジネス・オブジェクト* です。このビジネス・オブジェクトは、マップの実行を開始するビジネス・オブジェクトです。表 66 に、呼び出しコンテキストと関連するオリジナル要求ビジネス・オブジェクトを示します。

表 66. 呼び出しコンテキストとそれに関連するオリジナル要求ビジネス・オブジェクト

呼び出しコンテキスト	オリジナル要求ビジネス・オブジェクト	オリジナル要求ビジネス・オブジェクトの例
EVENT_DELIVERY, ACCESS_REQUEST	アプリケーションから発生したアプリケーション固有のビジネス・オブジェクト	AppA 固有
SERVICE_CALL_REQUEST, SERVICE_CALL_FAILURE	InterChange Server Express から送信された汎用ビジネス・オブジェクト	汎用
SERVICE_CALL_RESPONSE	SERVICE_CALL_REQUEST から送信された汎用ビジネス・オブジェクト	汎用
ACCESS_RESPONSE	最初にアクセス要求から発生したアプリケーション固有のビジネス・オブジェクト	AppA 固有

例えば、*汎用ビジネス・オブジェクト* は、SERVICE_CALL_RESPONSE、SERVICE_CALL_FAILURE、または SERVICE_CALL_REQUEST の呼び出しコンテキストで実行されるマップのオリジナル要求ビジネス・オブジェクトです。これらのマップは汎用ビジネス・オブジェクトを使用して、変換される関係属性の関係インスタンス ID を保管します。マップには、関係インスタンスを検索して新たに作成または更新されたオブジェクトの関連する参加者データを設定するために、関係インスタンス ID が必要です。

例: 次の例で、これが顧客同期シナリオでどのように機能するかを示します。システムを使用して、アプリケーション A とアプリケーション B の間で同期されたデータを保持すると仮定します。両アプリケーションとも顧客データを保管し、顧客 ID 属性は関係を使用して管理されます。この例では、コラボレーションとコネクターの詳細は省略します。

新規顧客がアプリケーション A に追加される条件

1. マップにより、AppA 固有のビジネス・オブジェクトが EVENT_DELIVERY の呼び出しコンテキストで汎用ビジネス・オブジェクトに変換されます。

   顧客 ID 属性を変換する場合、マップにより新規の関係インスタンスが顧客 ID 関係表に作成され、汎用ビジネス・オブジェクトの顧客 ID 属性に新規の関係インスタンス ID が挿入されます。
2. マップにより、汎用ビジネス・オブジェクトが SERVICE_CALL_REQUEST の呼び出しコンテキストで AppB 固有のビジネス・オブジェクトに変換されます。

   関係表は変更されません。アプリケーション B は、新規顧客をアプリケーションに正常に追加します。

3. マップにより、AppB 固有のビジネス・オブジェクトが SERVICE_CALL_RESPONSE の呼び出しコンテキストで汎用ビジネス・オブジェクトに変換されます。このマップの実行コンテキストには、ステップ 1 で生成された汎用ビジネス・オブジェクトが含まれます。

   これを実行する理由は、ステップ 1 で作成された関係インスタンスの新規参加者データを設定するためです。この場合、新規参加者データはアプリケーション B に追加された新規顧客の顧客 ID です。

図 90 に、新規顧客 ID をアプリケーション B に正常に追加する、コール・トリガー・フローに関して、各ステップのマップの実行が発生する条件を示します。

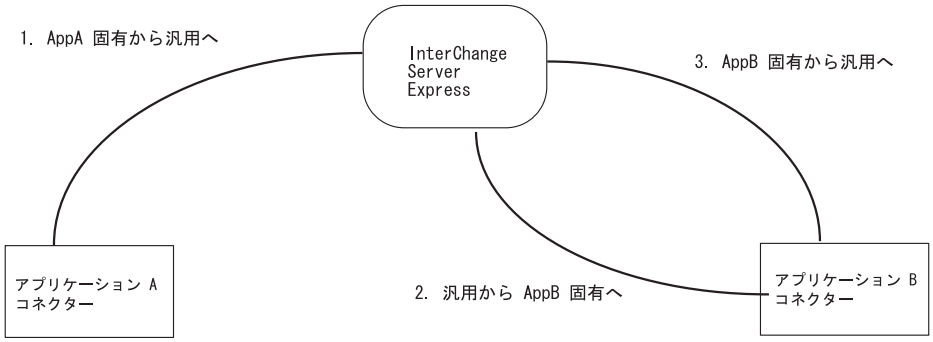

図 90. 呼び出しコンテキストの例

子ビジネス・オブジェクトのマッピング

ソース・ビジネス・オブジェクトが子ビジネス・オブジェクトを含む場合、子ビジネス・オブジェクトのマップ方法はソースおよび宛先ビジネス・オブジェクト内の子のカーディナリティーにより異なります。このセクションでは、次のケースでの子ビジネス・オブジェクトのマッピング方法について説明します。

- 『単一カーディナリティーのソースおよび宛先のマッピング』
- 212 ページの『単一カーディナリティーのソースから複数カーディナリティーの宛先へのマッピング』
- 212 ページの『複数カーディナリティーのソースおよび宛先のマッピング』

単一カーディナリティーのソースおよび宛先のマッピング

単一カーディナリティーのソース子ビジネス・オブジェクトを単一カーディナリティーの宛先ビジネス・オブジェクトにマップするには、次の手順を行います。

- ソース子オブジェクト名の左の正記号 (+) をクリックして展開し、単純な変換ステップ (41 ページの『標準の属性変換の指定』および 190 ページの『その他の属性の変換方法』を参照) に従ってソース属性を宛先属性にマップします。このサブマップを作成できますが、必要ありません。

- 宛先子オブジェクトの動詞を設定することをお勧めします。このためには、40 ページの『宛先ビジネス・オブジェクトの動詞の設定』の指示に従います。動詞をソース親オブジェクトの動詞に設定します (オブジェクトに特別な関係要件がない場合)。

例: このようなマッピングは次のマッピングの場合に必要になります。

```
SAP_CustomerMaster
```

から次へ

```
Customer.CustomerAddress
```

単一カーディナリティーのソースから複数カーディナリティーの宛先へのマッピング

ソース子ビジネス・オブジェクトがカーディナリティー 1 の場合、これには宛先子ビジネス・オブジェクトの単一インスタンスにのみマップされるデータが含まれます。単一カーディナリティーのソース子オブジェクトを複数カーディナリティーの宛先子オブジェクトにマップするには、211 ページの『単一カーディナリティーのソースおよび宛先のマッピング』で説明する手順に従います。サブマップを作成する必要はありません。

例: このようなマッピングは次のマッピングの場合に必要になります。

```
SAP_CustomerMaster.SAP_CustCreditCentralData[1]
```

から次へ

```
Customer.CustomerInformation.CustomerCreditData[n]
```

複数カーディナリティーのソースおよび宛先のマッピング

複数カーディナリティーのソース子ビジネス・オブジェクトを複数カーディナリティーの宛先ビジネス・オブジェクトにマップするには、複数カーディナリティー・サブマップを作成する必要があります。サブマップの概要については、48 ページの『サブマップを使用した変換』を参照してください。

例: このようなマッピングは次のマッピングの場合に必要になります。

```
SAP_CustBankData[n]
```

から次へ

```
Customer.CustomerInformation.CustomerBankData[n]
```

同じタイプの複数のソース・ビジネス・オブジェクトを変換する場合は、表 67 の手順に従い、実行します。

表 67. 複数カーディナリティー・サブマップの作成

作成手順	詳細情報の参照先
1. 複数カーディナリティー・サブマップの作成。これによりあるソース・オブジェクト内の属性からある宛先ビジネス・オブジェクトへの変換が実行されます。	213 ページの『複数カーディナリティー・サブマップの作成手順』

表 67. 複数カーディナリティー・サブマップの作成 (続き)

作成手順	詳細情報の参照先
2. 宛先ビジネス・オブジェクトの複数カーディナリティー属性内の、メイン・マップからこのサブマップの呼び出し。サブマップの呼び出しは、各ビジネス・オブジェクトがサブマップに渡されるように、複数カーディナリティー・オブジェクト内の各ビジネス・オブジェクトをループする for ループ内で行います。	『複数カーディナリティー・サブマップの呼び出し手順』

複数カーディナリティー・サブマップの作成手順

ソースから宛先オブジェクトに複数カーディナリティーの子オブジェクトをマップするサブマップを作成するには、単一ソース・ビジネス・オブジェクトと単一宛先ビジネス・オブジェクトでマップを作成します。このマップには、宛先オブジェクトの対応する属性へのソース・オブジェクト内の属性の変換を含みます。

複数カーディナリティー・サブマップを作成するには、Map Designer Express で次の手順を実行します。

1. メイン・マップを閉じて、新規マップを開始します。
2. 「ダイアグラム」タブで、ソース子ビジネス・オブジェクトをマップ・ワークスペース域の左半分にドラッグし、宛先子ビジネス・オブジェクトを右半分にドラッグします。

   SAP_CustBankData ビジネス・オブジェクトを Customer.CustomerInformation.CustomerBankData ビジネス・オブジェクトにマップするには、SAP_CustBankData をワークスペースの左半分にドラッグし、CustomerBankData を右半分にドラッグします。
3. サブマップを保管します。

   推奨: サブマップ名はプレフィックス「Sub_」で始めてください。
   例: Sub_SaCwCustBankData
4. 宛先オブジェクトの動詞を、 40 ページの『宛先ビジネス・オブジェクトの動詞の設定』の説明に従って設定します。
5. 個々の属性を、41 ページの『標準の属性変換の指定』と 190 ページの『その他の属性の変換方法』の説明に従って、マップします。
6. 「ファイル」メニューから「コンパイル」を選択して、サブマップをコンパイルします。

   結果: すべてが正しいと、次のメッセージが表示されます。

   ```
   マップ検証が正常に終了しました。
   マップのコンパイルが正常に終了しました。
   ```

ヒント: サブマップのコンパイルを忘れると、「サブマップ」ダイアログで表示できません。メッセージ「サブマップが使用不能です。」が表示されます。

複数カーディナリティー・サブマップの呼び出し手順

複数カーディナリティー・サブマップを呼び出すには、宛先ビジネス・オブジェクトの複数カーディナリティー属性内の、メイン・マップから runMap() メソッドで呼び出します。サブマップの呼び出しは、各ビジネス・オブジェクトがサブマップ

に渡されるように、複数カーディナリティー・オブジェクト内の各ビジネス・オブジェクトをループする for ループ内で行います。

複数カーディナリティー・サブマップを呼び出すには、Map Designer Express で次の手順を実行します。

1. サブマップを閉じて、メイン・マップを開きます。
2. ソース子オブジェクトと宛先子オブジェクトを選択して、変換規則のコンボ・ボックスから「サブマップ」を選択します。

   SAP_CustBankData から Customer.CustomerInformation.CustomerBankData への変換の場合、SAP_CustBankData と CustomerBankData を選択します。

   結果: 「サブマップ」ダイアログ・ボックスが表示されます。
3. サブマップの名前を選択して、「OK」をクリックします。

   例: Sub_SaCwCustBankData

   この例では、条件を指定する必要がないので「OK」をクリックします。
4. 宛先子オブジェクトの Activity Editor を開きます。

   SAP_CustBankData から Customer.CustomerInformation.CustomerBankData への変換の場合、Activity Editor に次のようなコードが表示されます。

```
{
BusObjArray srcCollection_For_ObjSAP_Order_SAP_OrderPartners =
   ObjSAP_Order.getBusObjArray("SAP_OrderPartners");

//
// LOOP ONLY ON NON-EMPTY ARRAYS
// -----------------------------
//
// Perform the loop only if the source array is non-empty.
//
if ((srcCollection_For_ObjSAP_Order_SAP_OrderPartners != null) &&
      (srcCollection_For_ObjSAP_Order_SAP_OrderPartners.size() > 0))
   {
   int currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners;
   int lastInputIndex_For_ObjSAP_Order_SAP_OrderPartners =
   srcCollection_For_ObjSAP_Order_SAP_OrderPartners.getLastIndex();
   for (currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners = 0;
      currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners <=
      lastInputIndex_For_ObjSAP_Order_SAP_OrderPartners;
      currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners++)
      {
      BusObj currentBusObj_For_ObjSAP_Order_SAP_OrderPartners =
(BusObj) (srcCollection_For_ObjSAP_Order_SAP_OrderPartners.elementAt(
      (currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners));

      //
      // INVOKE MAP ON VALID OBJECTS
      // ---------------------------
      //
      // Invoke the map only on those child objects that meet certain
      // criteria.
      //
      if (currentBusObj_For_ObjSAP_Order_SAP_OrderPartners != null)
         {
         BusObj[] _cw_inObjs =
            { currentBusObj_For_ObjSAP_Order_SAP_OrderPartners };
         BusObj[] _cw_outObjs =
```

```
            DtpMapService.runMap(
            "Sub_SaOrderPartners_to_CwCustomerRole", "CwMap",
            _cw_inObjs, cwExecCtx);
   ObjOrder.setWithCreate("AssociatedCustomers",
            _cw_outObjs[0]);
         }
      }
   }
}
```

runMap() は静的メソッドなので、呼び出しは次のようにします。

```
DtpMapService.runMap()
```

5. 以下のように、このコードを多少変更します。
   - ソース・ビジネス・オブジェクトでは、親オブジェクトのみが関連する動詞を持ちます。子オブジェクトをマップするサブマップを呼び出すコードの場合は、この子オブジェクトをサブマップに渡す前に、ソース子オブジェクトの動詞を親の動詞の 1 つに設定することをお勧めします。 BusObj クラスの setVerb() メソッドを使用します。

     注: サブマップで関係管理 (相互参照) を実行する必要がある場合は、動詞の引き渡しが必要です。詳細については、299 ページの『単純一致関係の変換規則の定義』を参照してください。
   - サブマップ呼び出しのコード行は、MapNotFoundException を確実にキャッチするために try/catch ブロックの内部に組み込む必要があります。

   以下は、変更されたコードです (変更部分は太字で強調表示されます)

```
   {
   BusObjArray srcCollection_For_ObjSAP_
Order_SAP_OrderPartners =      ObjSAP_Order.getBusObjArray("SAP_OrderPartners");

   //
   // LOOP ONLY ON NON-EMPTY ARRAYS
   // -----------------------------
   //
   // Perform the loop only if the source array is non-empty.
   //
   if ((srcCollection_For_ObjSAP_Order_SAP_OrderPartners
      != null) &&
         (srcCollection_For_ObjSAP_Order_SAP_OrderPartners.size()
    > 0))
      {
      int currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners;
      int lastInputIndex_For_ObjSAP_Order_SAP_OrderPartners =
         srcCollection_For_ObjSAP_Order_SAP_OrderPartners.getLastIndex();

      for (currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners
    = 0;
            currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners
    <=
            lastInputIndex_For_ObjSAP_Order_SAP_OrderPartners;
            currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners++)
         {
            BusObj currentBusObj_For_ObjSAP_Order_SAP_OrderPartners =
 (BusObj)
                (srcCollection_For_ObjSAP_Order_SAP_OrderPartners.elementAt(
            currentBusObjIndex_For_ObjSAP_Order_SAP_OrderPartners));

         //
         // INVOKE MAP ON VALID OBJECTS
         // ---------------------------
```

```
            //
            // Invoke the map only on those child objects that meet
               certain
            // criteria.
            //
            if (currentBusObj_For_ObjSAP_Order_SAP_OrderPartners != null)
               {
               currentBusObj_For_ObjSAP_Order_SAP_OrderPartners.setVerb(
                  ObjSAP_Order.getVerb());            BusObj[] _cw_inObjs

               =  { currentBusObj_For_ObjSAP_Order_SAP_OrderPartners };

               try
                  {
                  BusObj[] _cw_outObjs = DtpMapService.runMap(
                     "Sub_SaOrderPartners_to_CwCustomerRole", "CwMap",
                     _cw_inObjs, cwExecCtx);
                  ObjOrder.setWithCreate("AssociatedCustomers", _cw_outObjs[0]);
                  }
               catch (MapNotFoundException me)
                  {
                  logError(5502, "Sub_SaOrderPartners_to_CwCustomerRole");
                  throw new MapFailureException ("Submap not found");
                  }            }
            }
         }
      }
```

6. サブマップの呼び出しを追加したら、メイン・マップを再コンパイルします。

サブマップの使用の続き

このセクションでは、サブマップの使用についての次のヒントを説明します。

- 『サブマップ呼び出し時の条件の指定』
- 219 ページの『式ビルダーを使用したサブマップの呼び出し』
- 220 ページの『異なるタイプのビジネス・オブジェクトのサブマップへの引き渡し』

サブマップの概要については、48 ページの『サブマップを使用した変換』を参照してください。

サブマップ呼び出し時の条件の指定

マップに、サブマップを呼び出す条件を判別するプログラミング・ロジックが必要な場合がよくあります。時には、呼び出されるサブマップが true となる必要のある条件があります。このロジックは、サブマップの宛先ビジネス・オブジェクトを含む属性内で、runMap() メソッドを呼び出す前に、メイン・マップに渡されます。これらの条件を「サブマップ」ダイアログの「条件」領域に入力すると (52 ページの図 21 を参照)、Map Designer Express によりこれらの条件が runMap() 呼び出しを囲む if 文に追加されます。

ガイドライン: これらの条件を入力する場合は、以下の点に留意してください。

- 条件の入力は 1 つのみですが、AND (&&) 演算子で結合して、複数の文節を指定できます。
- 行はセミコロンで終了しないでください。これは、入力した条件が宛先属性の生成されたコードで if 文節に変換されるためです。

例: 50 ページの図 20 に示すビジネス・オブジェクトでは、OrderLine ビジネス・オブジェクトは子ビジネス・オブジェクトである DelSched という属性を持ちます。サブマップ条件で、その属性を次のようにして参照できます。

```
currentBusObj_For_ObjOrder_OrderLine_DelSched
```

「サブマップ」ダイアログの「条件」領域に次のサブマップ条件を入力して、同じビジネス・オブジェクトの TransportType 属性の値がストリング AIR と等しい場合にのみ DelSched ビジネス・オブジェクトでサブマップを実行します。

```
currentBusObj_For_ObjOrder_OrderLine_DelSched.getString(
   "TransportType").equals("AIR")
```

次の条件は、OrderLine LinePrice 属性値が $10,000.00 より大きい場合にのみ OrderLine DelSched 属性のサブマップを実行します。

```
ObjOrderLine.getFloat["LinePrice") > 10000.00
```

例: 次のマップの場合、マッピングは SAP_CustPartnerFunctions.PartnerId が SAP_CustomerMaster.CustomerId と等しくない 場合にのみ 行われるので、条件が必要です。

```
SAP_CustomerMaster.SAP_CustSalesAreaData.SAP_
CustPartnerFunctions[n]
```

から次へ

```
Customer.RelatedCustomerRef
```

次のセクションでは、マップ呼び出しを作成する手順について説明します。

- 『サブマップの作成手順』
- 『サブマップの呼び出し手順』

サブマップの作成手順

Sub_SaCwCustCreditAreaData サブマップを作成するには、Map Designer Express で次の手順を実行します。

1. メイン・マップを閉じて、新規サブマップを開始します。

   SAP_CustPartnerFunctions のソース・ビジネス・オブジェクトと RelatedCustomerRef の宛先ビジネス・オブジェクトを指定します。マップを Sub_SaCwCustPartners と命名します。
2. 40 ページの『宛先ビジネス・オブジェクトの動詞の設定』の説明に従って動詞を設定します。
3. 個々の属性を、41 ページの『標準の属性変換の指定』と 190 ページの『その他の属性の変換方法』の説明に従って、マップします。
4. 「ファイル」メニューから「コンパイル」を選択して、サブマップをコンパイルします。すべてが正しいと、次のメッセージが表示されます。

   ```
   マップ検証が正常に終了しました。
   Native Map: Code generation succeeded.
   ```

サブマップの呼び出し手順

Sub_SaCwCustCreditAreaData サブマップを呼び出すには、Map Designer Express で次の手順を実行します。

1. メイン・マップに戻り、「ダイアグラム」タブで Ctrl キーを押したまま SAP_CustPartnerFunctions を RelatedCustomerRef の上へドラッグします。宛先ビジネス・オブジェクトの「規則」列のリストから「サブマップ」をダブルクリックします。「サブマップ」ダイアログで SaCwCustPartners マップを選択して「OK」をクリックします。

   単純な条件の場合は、「サブマップ」ダイアログ・ウィンドウの「条件」領域に条件を入力できます。

   ```
   currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_
   SAP_CustPartnerFunctions.getString("PartnerId")).equals(
   parent_custid)
   ```

   Map Designer Express により、サブマップ呼び出しに先行する if 文に条件が追加されて、サブマップ呼び出しのコードが生成されます。

   「条件」フィールドをブランクのままで「OK」をクリックすると、Map Designer Express により条件なしでサブマップを呼び出すコードが生成されます。コードを明示的に追加する方法については、ステップ 2 に進みます。

2. 生成されたコードを変更して、サブマップ条件を追加します。

   Activity Editor にアクセスするには、「規則」列を再度ダブルクリックして、「サブマップ」ダイアログの「コードを表示」プッシュボタンをクリックします。

   RelatedCustomerRef 属性について、条件を含むコードは次のとおりです。

```
{
   BusObjArray
      srcCollection_For_ObjSAP_CustomerMaster_
SAP_CustSalesAreaData_
SAP_CustPartnerFunctions
      = ObjSAP_CustomerMaster.getBusObjArray(
      "SAP_CustSalesAreaData.SAP_CustPartnerFunctions");

   String parent_custid = ObjSAP_CustomerMaster.getString("CustomerId");
   //
   // LOOP ONLY ON NON-EMPTY ARRAYS
   // -----------------------------
   //
   // Perform the loop only if the source array is non-empty.
   //
   if (
      (srcCollection_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_
         SAP_CustPartnerFunctions != null) &&
         (srcCollection_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
         _CustPartnerFunctions.size() > 0))
      {
      int currentBusObjIndex_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions;
      int lastInputIndex_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions =
         srcCollection_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions.getLastIndex();

      for (currentBusObjIndex_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions = 0;
           currentBusObjIndex_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions <= lastInputIndex_For_ObjSAP_CustomerMaster_
SAP_CustSalesAreaData_SAP_CustPartnerFunctions;
           currentBusObjIndex_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
```

```
_CustPartnerFunctions++)
         {
         BusObj currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions = (BusObj) (srcCollection_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions.elementAt(currentBusObjIndex_For_ObjSAP_CustomerMaster_SAP
_CustSalesAreaData_SAP_CustPartnerFunctions));

         //
         // INVOKE MAP ON VALID OBJECTS
         // ---------------------------
         //
         // Invoke the map only on those child objects that meet certain
         // criteria.
         //
         if (currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions != null)
            {
            if (!
         (currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions.getString("PartnerId")).equals(parent_custid))
       {
       currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions.setVerb(ObjSAP_CustomerMaster.getVerb());

               BusObj[] _cw_inObjs =
            { currentBusObj_For_ObjSAP_CustomerMaster_SAP_CustSalesAreaData_SAP
_CustPartnerFunctions };
               try
                  {
                  BusObj[] _cw_outObjs =
                     DtpMapService.runMap("Sub_SaCwCustPartners", "CwMap",
                     _cw_inObjs, cwExecCtx);

                  ObjCustomer.setWithCreate("RelatedCustomerRef",
                     _cw_outObjs[0]);
                  }
               catch (MapNotFoundException me)
                  {
                  logError(5502, "Sub_SaCwCustPartners");
                  throw new MapFailureException ("Submap not found");
                  }
               }
            }
         }
}
```

3. サブマップの呼び出しを追加したら、メイン・マップを再コンパイルします。

式ビルダーを使用したサブマップの呼び出し

Map Designer Express を使用すると、「サブマップ」ダイアログでのサブマップ呼び出しを自動的に生成できます。しかし、式ビルダーを使用してサブマップ呼び出しを生成することもできます。マップ呼び出しをコーディングすると、Map Designer Express によるグラフィックでのサポートよりも変化のある操作を作成できます。例えば、子ビジネス・オブジェクトを含まない属性の値を指定するサブマップ、または複数の入力および出力のあるサブマップを使用できます。

式ビルダーを使用したサブマップの呼び出し手順

式ビルダーを使用してサブマップ呼び出しを生成するには、次の手順を行います。

1. 「テーブル」タブまたは「ダイアグラム」タブで変換規則をダブルクリックして、Java 表示の Activity Editor を表示させます。

2. Activity Editor の任意の場所を右マウス・ボタンでクリックして、コンテキスト・メニューから「式ビルダー」を選択します。
3. 式ビルダーで、次の操作をします。
   - 「メイン・カテゴリー」列で、マップ・フォルダーを展開します。
   - マップ下のマップ・タイプでネイティブ・マップ・フォルダーを展開して、システムで使用可能なすべてのコンパイル済みマップのリストを表示します。
   - 呼び出すマップを選択して、「マップ API」列の `DtpMapService.runMap()` メソッドを選択します。
   - 「Insert API」ボタンをクリックして、コード生成を開始します (図 91 を参照)。

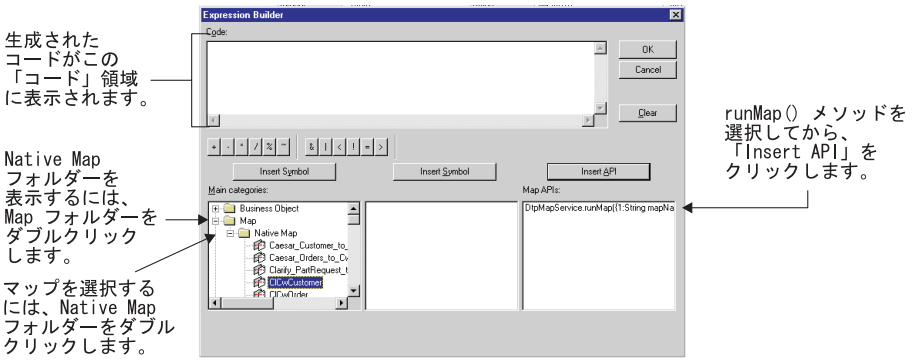

図 91. サブマップ呼び出しのコーディング

結果: Map Designer Express により、サブマップとして指定するマップを呼び出すために必要な Java コードが生成されます。これは、「式ビルダー」ウィンドウの上部の「コード」領域に表示されます。

4. 生成されたコードで、入力および出力プレースホルダーをサブマップの実際の現行ソースと宛先ビジネス・オブジェクト名で置き換えて、サブマップが戻す値を処理するコードを作成します。

   プレースホルダーの `InObj`、`InAttrName`、`OutObj`、および `OutAttrName` を次の正しい名前で置き換えます。
   - トップレベル・ソース・ビジネス・オブジェクト変数: `Obj`*`SrcBusObj`*
   - ソース属性: 子ビジネス・オブジェクトを含む属性の名前へのパス
   - 宛先ビジネス・オブジェクト変数: `Obj`*`DestBusObj`*
   - 宛先属性
5. 「OK」をクリックして、式ビルダーを閉じます。

   結果: Activity Editor の挿入ポイントにコードが挿入されます。

異なるタイプのビジネス・オブジェクトのサブマップへの引き渡し

異なるタイプのソース・ビジネス・オブジェクトを宛先ビジネス・オブジェクトにマップするには、多対 1 のサブマップを作成する必要があります。

例: このようなマッピングは次のマッピングの場合に必要になります。

SAP_CustCreditControlAreaData[n]

から次へ
Customer.CustomerInformation.CustomerCreditData[0].
CreditAreaCreditData[n]

属性の一部が SAP_CustomerMaster と SAP_CustCreditCentralData からマップされる必要があります。

タイプが異なる複数のソース・ビジネス・オブジェクトを変換するには、以下の主要な手順を実行します。

1. 『多対 1 のサブマップの作成手順』: 複数のソース・ビジネス・オブジェクト内の必要な属性を単一宛先ビジネス・オブジェクト内の属性に変換する、多対 1 サブマップを作成します。
2. 222 ページの『多対 1 のサブマップの呼び出し手順』: メイン・マップから、サブマップの宛先ビジネス・オブジェクトを保持する属性でこのサブマップを呼び出します。

多対 1 のサブマップの作成手順

異なるタイプのソース子ビジネス・オブジェクトを 1 つの宛先子オブジェクトにマップするサブマップを作成するには、複数のソース・ビジネス・オブジェクトと 1 つの宛先ビジネス・オブジェクトでマップを作成します。このマップには、宛先オブジェクトの対応する属性への異なるソース・オブジェクト内の属性の変換を含みます。

多対 1 のサブマップを作成するには、Map Designer Express で次の手順を実行します。

1. メイン・マップを閉じて、新規マップを開始します。新しいマップには、少なくとも 2 つのソース・オブジェクトと 1 つの宛先オブジェクトが必要です。

   SAP_CustCreditControlAreaData、SAP_CustCreditCentralData、および SAP_CustomerMaster ビジネス・オブジェクトをマッピング画面の左方にドラッグし、CreditAreaCreditData を右方にドラッグします。新しいマップには、3 つのソース・オブジェクトと 1 つの宛先オブジェクトがあります。
2. サブマップを保管します。

   推奨: サブマップ名はプレフィックス「Sub_」で始めてください。
   例: Sub_SaCwCustCreditAreaData
3. 宛先オブジェクトの動詞を、40 ページの『宛先ビジネス・オブジェクトの動詞の設定』の説明に従って設定します。

   このサブマップで関係管理が実行されない限り、動詞の設定にどのソース・オブジェクトを使用しても問題ありません。このサブマップを呼び出す場合は、特定の同じオブジェクトの動詞を設定してから、サブマップにそのオブジェクトを渡してください。ソース・オブジェクトの 1 つがソース親オブジェクトで、かつその動詞を使用する場合は、動詞を設定してからこのオブジェクトをサブマップに渡す必要はありません。すでに関連する動詞を持っています。
4. 個々の属性を、41 ページの『標準の属性変換の指定』と 190 ページの『その他の属性の変換方法』の説明に従って、マップします。

5. 「ファイル」メニューから「コンパイル」を選択して、サブマップをコンパイルします。

   結果: すべてが正しいと、次のメッセージが表示されます。

   マップ検証が正常に終了しました。
   Native Map: Code generation succeeded.

多対 1 のサブマップの呼び出し手順

多対 1 のサブマップを呼び出すには、メイン・マップから、サブマップの宛先ビジネス・オブジェクトを保持する属性で、runMap() メソッドを使用して呼び出します。 Sub_SaCwCustCreditAreaData サブマップを呼び出すには、Map Designer Express で次の手順を実行します。

1. メイン・マップを開きます。

   Sub_SaCwCustCreditAreaData サブマップの場合、メイン・マップは CreditAreaCreditData ビジネス・オブジェクトを属性として含むマップです。

2. サブマップの宛先オブジェクトの変換規則列をダブルクリックして、Activity Editor を開きます。

   CreditAreaCreditData 属性に関連した変換規則列をダブルクリックして、Activity Editor を開き、次のコードを入力します。

```
{
BusObj srcobj1 = ObjSAP_CustomerMaster;
BusObj srcobj2 = (BusObj)
   (ObjSAP_CustomerMaster.getBusObj("SAP_CustCreditCentralData"));
BusObjArray srcobj3 =
   (BusObjArray)(ObjSAP_CustomerMaster.get(
      "SAP_CustCreditControlAreaData"));

//
// INVOKE MAP ON VALID OBJECTS
// ---------------------------
//
// Invoke the map only on those child objects that meet certain
// criteria.
//
int i = 0;

// When checking all 3 source objects, != null might be not required
if (srcobj1 != null && srcobj2 != null & srcobj3 != null)
   {
   for (i = 0; i < srcobj3.size(); i++)
      {
      BusObj[] _cw_inObjs = new BusObj[3];

      // set verb for the one of the following objects
      _cw_inObjs[0] = srcobj1;
      _cw_inObjs[1]= srcobj2;
      _cw_inObjs[2] = srcobj3.elementAt(i);

      try
         {
         BusObj[] _cw_outObjs = DtpMapService.runMap(
            "Sub_SaCwCustCreditAreaData",
            "CwMap",
            _cw_inObjs,
            cwExecCtx);
         ObjCustomer.setWithCreate(
"CustomerInformation.CustomerCreditData[0].CreditAreaCreditData["
```

```
            + i + "]", _cw_outObjs[0]);
         }

      catch (MapNotFoundException me)
         {
         logError(5502, "Sub_SaCwCustCreditAreaData");
         throw new MapFailureException ("Submap not found");
         }
      }
   }
}
```

3. マップの実行を追跡するには、for ループ内で logInfo() メソッドを使用します。コードは次のとおりです。

   ```
   logInfo("in for loop");
   ```

   または

   ```
   trace("in for loop");
   ```

4. サブマップの呼び出しを追加したら、メイン・マップを再コンパイルします。

すべてが適切に動作し、どのソース・オブジェクトも null でない場合、ソース子オブジェクトのインスタンスが存在する回数だけ in for loop メッセージが InterChange Server Express ログ・ファイルに出力されます。

データベース照会の実行

マップの実行時に、データベース (リレーションシップ・データベースなど) から情報を取得する必要があることがあります。データベースの情報を取得または変更するには、その表を照会します。*照会* は、データベースに送信して実行させる、通常は SQL (Structured Query Language) ステートメント形式の要求です。表 68 に、データベースでの照会の実行に関する手順を示します。

注: IBM により JDBC でサポートされるすべての外部データベースに、Oracle シン・ドライバーや WebLogic ドライバーを使用してアクセスできます。データベースには、Oracle と Microsoft SQL Server を含みます。

表 68. 照会の実行手順

照会の実行作業	詳細情報の参照先
1. データベースへの接続 (CwDBConnection オブジェクト) を取得します。	223 ページの『接続の取得』
2. CwDBConnection オブジェクトを使用して、照会を送信し、データベース内のトランザクションを管理します。	224 ページの『照会の実行』 237 ページの『トランザクションの管理』
3. 接続を解放します。	241 ページの『接続の解放』

接続の取得

データベースの照会を行うには、最初に BaseDLM クラスの getDBConnection() メソッドを使用してデータベースへの接続を取得する必要があります。取得する接続を識別するには、この接続が含まれる接続プールの名前を指定します。特定の接続プールのすべての接続は、同じデータベースに対して行われます。接続プール内の接続の数は、接続プール構成の一部として決定されます。照会対象のデータベースの接続が含まれる接続プールの名前を決定する必要があります。

要確認: 接続は、InterChange Server Express がブートしたときに開かれるか、新しい接続プールが構成されたときに動的に開かれます。このため、目的のデータベースに対する接続が含まれる接続プールは、接続を要求するマップ・インスタンスが実行される前に構成する必要があります。IBM System Manager 内に接続プールを構成します。

図 92 では、getDBConnection() の呼び出しにより、CustDBConnPool 接続プール内の接続に関連するデータベースへの接続が取得されます。

```
CwDBConnection connection =
 getDBConnection("CustDBConnPool");
```

図 92. 接続プールからの接続の取得

getDBConnection() 呼び出しにより CwDBConnection オブジェクトが connection 変数に戻されます。その後これを使用して、接続に関連するデータベースにアクセスできます。

ヒント: getDBConnection() メソッドは、接続のトランザクション・プログラミング・モデルを指定できる追加形式を提供します。詳細については、237 ページの『トランザクションの管理』を参照してください。

照会の実行

表 69 に、CwDBConnection クラスのメソッドを使用して SQL 照会を実行する方法を示します。

表 69. CwDBConnection メソッドの使用による SQL 照会の実行

照会のタイプ	説明	CwDBConnection メソッド
静的照会	SQL ステートメントは、テキストとしてデータベースに送信されます。	executeSQL()
準備済み照会	初回の実行後、SQL ステートメントは、コンパイルされた実行可能形式で保管されます。これにより、後続の実行時に、このプリコンパイル形式が使用できます。	executePreparedSQL()
ストアード・プロシージャー	SQL ステートメントおよび条件ロジックが含まれるユーザー定義のプロシージャー。	executeSQL()executePreparedSQL() executeStoredProcedure()

静的照会の実行

executeSQL() メソッドにより、静的照会がデータベースに送信されて実行されます。*静的照会* は、ストリングとしてデータベースに送信される SQL ステートメントです。このデータベースがこのストリングを解析し、結果の SQL ステートメントを実行します。このセクションでは、executeSQL() を使用して以下の種類の SQL 照会をデータベースに送信する方法について説明します。

- データベースからデータを戻す照会 (SELECT)
- データベースのデータを変更する照会 (INSERT、UPDATE、DELETE)
- データベースに定義されているストアード・プロシージャーを実行する照会

データを戻す静的照会の実行 (SELECT): SQL ステートメント SELECT は、1 つ以上の表のデータを照会します。SELECT ステートメントをデータベースに送信して実行させるには、SELECT のストリング表記を引き数として executeSQL() メソッドに指定します。

例: executeSQL() への次の呼び出しにより、customer 表からの 1 つの列値を取得する SELECT が送信されます。

```
connection.executeSQL(
   "select cust_id from customer where active_status = 1");
```

注: 前述のコードでは、connection 変数は前の getDBConnection() メソッドの呼び出しで取得した CwDBConnection オブジェクトです (図 92 を参照)。

また、executeSQL() メソッドの 2 番目の形式を使用して、パラメーターを持つ SELECT ステートメントを送信することもできます。

例: executeSQL() に対する以下の呼び出しにより、前の例と同じ操作が行われます。ただし、この場合、アクティブ状況がパラメーターとして SELECT ステートメントに渡されます。

```
Vector argValues = new Vector();
String active_stat = "1";
argValues.add( active_stat );
connection.executeSQL(
   "select cust_id from customer where
active_status = ?", argValues);
```

SELECT ステートメントは、データベース表からデータを行として戻します。各行は、SELECT の WHERE 文節の条件と一致するデータの 1 行を示します。各行には、SELECT ステートメントが指定した列の値が含まれます。戻されたデータは、これらの行および列による 2 次元配列として表示できます。

ヒント: SELECT ステートメントの構文は、アクセスする特定のデータベースに対して有効である必要があります。SELECT ステートメントの正確な構文については、データベース文書を確認してください。

戻されたデータにアクセスするには、次の手順を実行します。

1. 1 行のデータを取得します。
2. 列の値を 1 つずつ取得します。

表 70 に、戻された照会データの行にアクセスするための CwDBConnection クラスのメソッドを示します。

表 70. 行にアクセスするための CwDBConnection メソッド

行にアクセスするための操作	CwDBConnection メソッド
行があるかどうかのチェック	hasMoreRows()
1 行のデータの取得	nextRow()

hasMoreRows() メソッドを使用して、戻された行のループを制御します。hasMoreRows() から false が戻されると、行ループが終了します。1 行のデータを取得するには、nextRow() メソッドを使用します。このメソッドは、選択された列の値を Java Vector オブジェクトの要素として戻します。これにより、Enumeration

クラスを使用して、列の値に個別にアクセスできます。 Vector と Enumeration の両クラスは java.util パッケージに存在します。

表 71 に、戻された照会行の列にアクセスするための Java メソッドを示します。

表 71. 列の値にアクセスするための Java メソッド

列にアクセスするための操作	Java メソッド
列の数の確認	Vector.size()
Enumeration に対する Vector のキャスト	Vector.elements()
列があるかどうかのチェック	Enumeration.hasMoreElements()
1 列のデータの取得	Enumeration.nextElement()

hasMoreElements() メソッドを使用して、列の値のループを制御します。 hasMoreElements() から false が戻されると、列ループが終了します。列の値を取得するには、nextElement() メソッドを使用します。

例: 以下のコード・サンプルは、顧客情報が格納されたデータベースへの接続である CwDBConnection クラスのインスタンスを取得します。次に、SELECT ステートメントを実行します。これにより、顧客 ID 20987 の会社名 CrossWorlds の単一列が含まれる 1 行が戻されます。

```
CwDBConnection connectn = null;
Vector theRow = null;
Enumeration theRowEnum = null;
String theColumn1 = null;

try
   {
   // Obtain a connection to the database
   connectn = getDBConnection("sampleConnectionPoolName");
   }

catch(CwDBConnectionFactoryException e)
   {
   System.out.println(e.getMessage());
   throw e;
   }

// Test for a resulting single-column, single-row, result set
try
   {
   // Send the SELECT statement to the database
   connectn.executeSQL(
      "select company_name from customer where cust_id = 20987");

   // Loop through each row
   while(connectn.hasMoreRows())
      {
      // Obtain one row
      theRow = connectn.nextRow();
      int length = 0;
      if ((length = theRow.size())!= 1)
         {
         return methodName + "Expected result set size = 1," +
            " Actual result state size = " + length;
         }

      // Get column values as an Enumeration object
      theRowEnum = theRow.elements();

      // Verify that column values exist
```

```
      if (theRowEnum.hasMoreElements())
         {
         // Get the column value
         theColumn1 = (String)theRowEnum.nextElement();
         if (theColumn1.equals("CrossWorlds")==false)
            {
            return "Expected result = CrossWorlds,"
               + " Resulting result = " + theColumn1;
            }
         }
      }
   }

// Handle any exceptions thrown by executeSQL()
catch(CwDBSQLException e)
   {
   System.out.println(e.getMessage());
   }
```

例: 以下の例は、複数の行を戻す SELECT ステートメントのコード・フラグメントを示します。各行には、顧客 ID および関連会社名の 2 列が含まれます。

```
CwDBConnection connectn = null;
Vector theRow = null;
Enumeration theRowEnum = null;
Integer theColumn1 = 0;
String theColumn2 = null;

try
   {
   // Obtain a connection to the database
   connectn = getDBConnection("sampleConnectionPoolName");
   }

catch(CwDBConnectionFactoryException e)
   {
   System.out.println(e.getMessage());
   throw e;
   }

// Code fragment for multiple-row, multiple-column result set.
// Get all rows with the specified columns, where the
// specified condition is satisfied
try
   {
   connectn.executeSQL(
"select cust_id, company_name from customer where active_status = 1");

   // Loop through each row
   while(connectn.hasMoreRows())
      {
      // Obtain one row
      theRow = connectn.nextRow();

      // Obtain column values as an Enumeration object
      theRowEnum = theRow.elements();
      int length = 0;
      if ((length = theRow.size()) != 2)
      {
         return "Expected result set size = 2," +
            " Actual result state size = " + length;
      }
      // Verify that column values exist
      if (theRowEnum.hasMoreElements())
         {
         // Get the column values
         theColumn1 =
```

```
            ((Integer)theRowEnum.nextElement()).intValue();
          theColumn2 = (String)theRowEnum.nextElement();
          }
       }
   }
catch(CwDBSQLException e)
   {
   System.out.println(e.getMessage());
   }
```

注: SELECT ステートメントでは、データベースの内容は変更されません。したがって、通常は SELECT ステートメントのトランザクション管理を行う必要はありません。

データを変更する静的照会の実行: データベース表のデータを変更する SQL ステートメントには、以下が含まれます。

- INSERT は、データベース表に新しい行を追加します。
- UPDATE は、データベース表の既存の行を変更します。
- DELETE は、データベース表から行を除去します。

これらのステートメントの 1 つを静的照会としてデータベースに送信して実行させるには、executeSQL() メソッドの引き数として、ステートメントをストリング表記で指定します。

例: executeSQL() に対する以下の呼び出しにより、現在の接続に関連付けられたデータベースの abc 表に対する 1 行の INSERT が送信されます。

```
connection.executeSQL("insert into abc values (1, 3, 6)");
```

注: 前述のコードでは、connection 変数は前の getDBConnection() メソッドの呼び出しで取得した CwDBConnection オブジェクトです。

UPDATE または INSERT ステートメントの場合、getUpdateCount() メソッドを使用して、変更または追加されたデータベース表の行数を確認できます。

要確認: INSERT、UPDATE、および DELETE ステートメントはデータベースの内容を変更するため、これらのステートメントに対してトランザクション管理の必要性を評価してください。詳細については、237 ページの『トランザクションの管理』を参照してください。

静的ストアード・プロシージャーの実行: 以下の*両方の* 条件が満たされていれば、executeSQL() メソッドを使用してストアード・プロシージャー呼び出しを実行できます。

- ストアード・プロシージャーは、OUT パラメーターを使用しません。

  ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用して実行する必要があります。
- ストアード・プロシージャーは 1 回のみ呼び出されます。

  executeSQL() メソッドは、ストアード・プロシージャー呼び出しに対する準備済みステートメントを保管しません。このため、例えば、ループ内で同じストアード・プロシージャーを複数回呼び出す場合、executeSQL() を使用すると、準備済

みステートメントを保管するメソッド executePreparedSQL() または executeStoredProcedure() を呼び出す場合より処理が遅くなることがあります。

詳細については、231 ページの『ストアード・プロシージャーの実行』を参照してください。

準備済み照会の実行

executePreparedSQL() メソッドにより、準備済み照会がデータベースに送信されて実行されます。*準備済み照会* は、データベースによる実行可能形式にすでにプリコンパイルされている SQL ステートメントです。executePreparedSQL() は、初めてデータベースに照会を送信するときに照会をストリングとして送信します。データベースは、この照会を受信し、ストリングを解析して実行可能形式にコンパイルし、その結果の SQL ステートメントを実行します (executeSQL() に対する操作と同様)。ただし、データベースはこの SQL ステートメントのコンパイル済み形式を executePreparedSQL() に戻し、このメソッドがこのステートメントをメモリーに格納します。このコンパイル済み SQL ステートメントは、*準備済み*ステートメントと呼ばれます。

これと同じ照会が後で実行されるときには、executePreparedSQL() は、この照会に対して準備済みステートメントがすでに存在するかどうかを最初にチェックします。準備済みステートメントが存在する場合、executePreparedSQL() は、照会ストリングの代わりにこれをデータベースに送信します。この後にこの照会が実行される場合、データベースはストリングを解析して準備済みステートメントを実行する必要がないため、処理がより効率的になります。

executePreparedSQL() を使用して、以下の種類の SQL 照会をデータベースに送信できます。

- データベースからデータを戻す照会 (SELECT)
- データベースのデータを変更する照会 (INSERT、UPDATE、DELETE)
- データベースに定義されているストアード・プロシージャーを実行する照会

データを戻す準備済み照会の実行 (SELECT): 同じ SELECT ステートメントを複数回実行する必要がある場合は、executePreparedSQL() を使用してステートメントのプリコンパイル・バージョンを作成します。SELECT ステートメントを準備する場合は、以下の点に注意する必要があります。

- この SELECT ステートメントのパラメーターを使用して、準備済みステートメントの実行のたびに特定の情報を渡すことができます。準備済みステートメントに対してパラメーターを使用する方法の例については、図 93 を参照してください。
- executePreparedSQL() で SELECT ステートメントを実行する場合、同じメソッドを使用して戻されたデータにアクセスします (表 70 と表 71)。詳細については、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

データを変更する準備済み照会の実行: 同じ INSERT、UPDATE、または DELETE ステートメントを複数回実行する必要がある場合は、executePreparedSQL() を使用してステートメントのプリコンパイル・バージョンを作成します。再実行する SQL ステートメントは、実行するたびに正確に同じでなくても、準備済みステートメン

トの利点を利用できます。SQL ステートメントのパラメーターを使用して、ステートメントの実行のたびに情報を動的に提供できます。

図 93 のコード・フラグメントでは、employee 表に 50 行挿入します。executePreparedSQL() は、初めて呼び出されるときに、INSERT ステートメントのストリング・バージョンをデータベースに送信します。これにより、このデータベースは、これを解析し、実行し、その実行可能形式である準備済みステートメントを戻します。INSERT ステートメントが実行される次の 49 回 (すべての INSERT の実行が成功であると仮定) では、executePreparedSQL() は、準備済みステートメントが存在することを認識し、この準備済みステートメントをデータベースに送信して実行させます。

```
CwDBConnection connection;
Vector argValues = new Vector();

argValues.setSize(2);

int emp_id = 1;
int emp_id = 2000;

for (int = 1; i < 50; i++)
   {
   argValues.set(0, new Integer(emp_id));
   argValues.set(1, new Integer(emp_num));

   try
      {
      // Send the INSERT statement to the database
      connection.executePreparedSQL(
"insert into employee (employee_id, employee_number) values (?, ?)",
          argValues);

      // Increment the argument values
      emp_id++;
      emp_num++
      }

   catch(CwDBSQLException e)
      {
      System.out.println(e.getMessage());
      }
   }
```

図 93. 準備済みステートメントへの引き数値の引き渡し

ヒント: 通常、INSERT ステートメントの準備済みバージョンの実行により、アプリケーションのパフォーマンスが向上しますが、アプリケーションのメモリー・フットプリントは増加します。

データベースを変更する SQL ステートメントを再実行する場合、トランザクション・プログラミング・モデルに応じてトランザクションを処理する必要があります。詳細については、237 ページの『トランザクションの管理』を参照してください。

注: コードをわかりやすくするために、図 93 にはトランザクション管理が含まれていません。

準備済みストアード・プロシージャーの実行: 以下の*両方の* 条件が満たされていれば、executePreparedSQL() メソッドを使用してストアード・プロシージャー呼び出しを実行できます。

- ストアード・プロシージャーには、OUT パラメーターを設定できません。

  ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用して実行する*必要*があります。
- ストアード・プロシージャーが複数回呼び出されます。

  executePreparedSQL() メソッドは、ストアード・プロシージャー呼び出しに対する準備済みステートメントをメモリーに保管します。このため、ストアード・プロシージャーを 1 回のみ呼び出す場合、executePreparedSQL() を使用すると、準備済みステートメントを保管しないメソッド executeSQL() を使用してストアード・プロシージャーを呼び出す場合より多くのメモリーが使用されることがあります。

詳細については、231 ページの『ストアード・プロシージャーの実行』を参照してください。

ストアード・プロシージャーの実行

ストアード・プロシージャー は、SQL ステートメントおよび条件ロジックが含まれるユーザー定義のプロシージャーです。ストアード・プロシージャーは、データとともにデータベースに格納されます。

注: 新しい関係を作成する場合、各関係表を保守するためにストアード・プロシージャーが作成されます。

表 72 に、ストアード・プロシージャーを呼び出す CwDBConnection クラスのメソッドを示します。

表 72. ストアード・プロシージャーを呼び出す CwDBConnection メソッド

ストアード・プロシージャーの呼び出し方法	CwDBConnection メソッド	使用
データベースに CALL ステートメントを送信し、ストアード・プロシージャーを実行します。	executeSQL()	OUT パラメーターを*使用しない*ストアード・プロシージャーを呼び出して、*1 回のみ* 実行するため。
	executePreparedSQL()	OUT パラメーターを*使用しない*ストアード・プロシージャーを呼び出して、*複数回* 実行するため。
ストアード・プロシージャーの名前およびパラメーターの配列を指定し、プロシージャー呼び出しを作成し、これをデータベースに送信して実行させます。	executeStoredProcedure()	OUT パラメーターがある任意のストアード・プロシージャーを呼び出すため。

注: JDBC メソッドを使用して、ストアード・プロシージャーを直接実行できます。しかし、CwDBConnection クラスで用意されているインターフェースは単純

で、データベース・リソースを再使用するので、実行効率を向上させることができます。ストアード・プロシージャーを実行するには、CwDBConnection クラスのメソッドを使用してください。

ストアード・プロシージャーは、1 行または複数行の形式でデータを戻すことができます。この場合は、SELECT ステートメントで戻されたデータに対してと同じ Java メソッド (hasMoreRows() や nextRow() など) を使用して、照会から戻された行にアクセスします。詳細については、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

表 72 に示す、どのメソッドを使用してストアード・プロシージャーを呼び出すかは、以下によって決まります。

- プロシージャーに OUT パラメーターがあるかどうか。

  OUT パラメーターは、ストアード・プロシージャーがこれを介して値を呼び出しコードに戻すパラメーターです。ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用してストアード・プロシージャーを呼び出す必要があります。
- ストアード・プロシージャーを呼び出す回数。

  executeStoredProcedure() メソッドで、ストアード・プロシージャーのコンパイル済みバージョンを保管します。このため、例えば、ループ内で同じストアード・プロシージャーを複数回呼び出す場合、executeStoredProcedure() を使用すると、データベースがプリコンパイル済みバージョンを使用できるため、executeSQL() より処理が速くなることがあります。

次のセクションでは、executeSQL() や executeStoredProcedure() メソッドを使用した、ストアード・プロシージャーの呼び出し方法について説明します。

OUT パラメーターがないストアード・プロシージャーの呼び出し: OUT パラメーターがない ストアード・プロシージャーを呼び出すには、CwDBConnection の以下のメソッドのいずれかを使用できます。

- executeSQL() メソッドにより、静的ストアード・プロシージャー呼び出しがデータベースに送信されます。

  このプロシージャーは、ストリングとしてデータベースに送信され、ここで、実行される前に準備済みステートメントにコンパイルされます。この準備済みステートメントは保管されません。このため、1 回のみ呼び出す必要があるストアード・プロシージャーの場合は executeSQL() が役に立ちます。
- executePreparedSQL() メソッドにより、準備済みストアード・プロシージャー呼び出しがデータベースに送信されます。

  このプロシージャー呼び出しは、初回の呼び出し時にデータベースに送信されます。これにより、準備済みステートメントが作成され実行されます。ただし、データベースはこの準備済みステートメントを executePreparedSQL() に送信し、このメソッドがこのステートメントをメモリーに格納します。このため、例えば、ループ内などで複数回呼び出す必要があるストアード・プロシージャーに executePreparedSQL() が役に立ちます。

これらいずれかのメソッドを使用してストアード・プロシージャーを呼び出すには、メソッドの引き数として、ストアード・プロシージャーおよび引き数を含む CALL ステートメントをストリング表記で指定します。 図 94 では、executeSQL() の呼び出しにより、setOrderCurrDate() ストアード・プロシージャーを実行する CALL ステートメントが送信されます。

```
connection.executeSQL("call setOrderCurrDate(345698)");
```

図 94. executeSQL() によるストアード・プロシージャーの呼び出し

図 94 では、connection 変数は、getDBConnection() メソッドの前の呼び出しで取得した CwDBConnection オブジェクトです。executeSQL() を使用して、setOrderCurrDate() ストアード・プロシージャーを実行できます。これは、その単一引き数が IN パラメーターである、つまり、その値がストアード・プロシージャーのみ に送信されるためです。このストアード・プロシージャーは、どの OUT パラメーターも使用しません。

パラメーター配列を受け入れる executeSQL() または executePreparedSQL() の形式を使用して、その引き数値をストアード・プロシージャーに渡すことができます。ただし、これらのメソッドを使用して、OUT パラメーターを使用するストアード・プロシージャーを呼び出すことはできません。このようなストアード・プロシージャーを実行するには、executeStoredProcedure() を使用する必要があります。詳細については、233 ページの『executeStoredProcedure() によるストアード・プロシージャーの呼び出し』を参照してください。

注: CwDBConnection .executeSQL() メソッドを使用して ODBC を介して Oracle ストアード PL/SQL オブジェクトを呼び出す場合は、匿名 PL/SQL ブロックを使用します。以下は、受け入れ可能なフォーマットを示します (ストアード・プロシージャー名は myproc です)。

```
connection.executeSQL("begin myproc(...);
 end;");
```

executeStoredProcedure() によるストアード・プロシージャーの呼び出し:
executeStoredProcedure() メソッドは、OUT パラメーターを使用するストアード・プロシージャーを含む、任意のストアード・プロシージャーを実行できます。このメソッドは、executePreparedSQL() の場合と同じように、ストアード・プロシージャー呼び出しに対する準備済みステートメントを保管します。このため、executeStoredProcedure() は、複数回実行されるストアード・プロシージャー呼び出しのパフォーマンスを向上できます。

executeStoredProcedure() メソッドを使用してストアード・プロシージャーを呼び出す手順: executeStoredProcedure() メソッドを使用してストアード・プロシージャーを呼び出すには、以下の手順を行います。

1. 実行するストアード・プロシージャーの名前を String で指定します。
2. CwDBStoredProcedureParam オブジェクトの Vector パラメーター配列を作成します。これにより、各ストアード・プロシージャーのイン/アウト・パラメーター・タイプやパラメーター値などのパラメーター情報が指定されます。

パラメーター は、ストアード・プロシージャーに対して、またはストアード・プロシージャーから送信できる値です。パラメーターのイン/アウト・タイプにより、ストアード・プロシージャーがパラメーター値を使用する方法が決まります。

- IN パラメーターは*入力専用* です。ストアード・プロシージャーはパラメーター値を入力として受け入れますが、呼び出し元コードに値を戻すためにこのパラメーターを使用しません。
- OUT パラメーターは*出力専用* です。ストアード・プロシージャーはパラメーター値を入力として解釈せずに、呼び出し元コードに値を戻すために使用します。
- INOUT パラメーターは*入力および出力用* です。ストアード・プロシージャーはその値を入力として受け入れ、値を戻すときにもそのパラメーターを使用します。

CwDBStoredProcedureParam オブジェクトで、ストアード・プロシージャーの単一パラメーターを記述します。表 73 に、CwDBStoredProcedureParam オブジェクトに含まれる情報と、このパラメーター情報を検索して設定するメソッドを示します。

表 73. CwDBStoredProcedureParam オブジェクトのパラメーター情報

パラメーター情報	CwDBStoredProcedureParam メソッド
パラメーター値	getValue()
パラメーターのイン/アウト・タイプ	getParamType()

executeStoredProcedure() メソッドを使用してストアード・プロシージャーにパラメーターを渡す手順: executeStoredProcedure() メソッドを使用してストアード・プロシージャーにパラメーターを渡す手順は、以下のとおりです。

1. CwDBStoredProcedureParam オブジェクトを作成して、パラメーター情報を保持します。

   CwDBStoredProcedureParam() コンストラクターを使用して、新しい CwDBStoredProcedureParam オブジェクトを作成します。このコンストラクターに以下のパラメーター情報を渡すことにより、オブジェクトを初期化します。

   - イン/アウト・パラメーター・タイプにより、パラメーターが IN、INOUT、OUT のいずれであるかを指定します。
   - パラメーター値は、パラメーターに割り当てる値が含まれる Java データ型です。CwDBStoredProcedureParam クラスには、パラメーター値に関連付けることができるさまざまなデータ型をサポートする多くのバージョンのコンストラクターがあります。OUT パラメーターの場合、このパラメーター値はダミー値にすることができますが、データ型は、ストアード・プロシージャー宣言の OUT パラメーターのデータ型と一致する必要があります。
2. 各ストアード・プロシージャーのパラメーターについて、ステップ 1 を繰り返します。
3. すべてのストアード・プロシージャーのパラメーターを保持するのに十分な要素を持つ Vector オブジェクトを作成します。
4. 初期化された CwDBStoredProcedureParam オブジェクトをパラメーター Vector オブジェクトに追加します。

Vector クラスの addElement() または add() メソッドを使用して、CwDBStoredProcedureParam オブジェクトを追加します。

5. すべての CwDBStoredProcedureParam オブジェクトを作成し、これらを Vector パラメーター配列に追加したら、このパラメーター配列を 2 番目の引き数として executeStoredProcedure() メソッドに渡します。

   executeStoredProcedure() メソッドは、ストアード・プロシージャーおよびそのパラメーターをデータベースに送信して実行させます。

例: 以下のように、データベースに get_empno() ストアード・プロシージャーが定義されているとします。

```
create or replace procedure get_empno(emp_id IN number,
      emp_number OUT number) as
   begin
      select emp_no into emp_number
      from emp
      where emp_id = 1;
   end;
```

この get_empno() ストアード・プロシージャーには、次の 2 つのパラメーターがあります。

- 最初のパラメーター emp_id は、IN パラメーターです。

  このため、PARAM_IN のイン/アウト・タイプ、およびストアード・プロシージャーに送信する適切な値を使用して、その関連 CwDBStoredProcedureParam オブジェクトを初期化する必要があります。 emp_id は SQL NUMBER 型 (整数値を保持する) として宣言されているので、このパラメーターの値は整数値 Integer を保持する Java Object です。

- 2 番目のパラメーター emp_number は、OUT パラメーターです。

  このパラメーターの場合、空の CwDBStoredProcedureParam オブジェクトを作成して、ストアード・プロシージャーに送信します。このオブジェクトは、PARAM_OUT のイン/アウト・タイプを使用して初期化します。ただし、このパラメーターにはダミーの Integer 値を入力します。ストアード・プロシージャーによる実行が終了したら、getValue() メソッドを使用してこの OUT パラメーターから戻された値を取得できます。

図 95 は、executeStoredProcedure() メソッドで get_empno() ストアード・プロシージャーを実行して、従業員 ID が 65 の従業員番号を取得します。

```
CwDBConnection connectn = null;

try
   {
   // Get database connection
   connectn = getDBConnection("CustomerDBPool");

   // Create parameter Vector
   Vector paramData = new Vector(2);

   // Create IN parameter for the employee id and add to parameter
   // vector
   paramData.add(
      new CwDBStoredProcedureParam(PARAM_IN, new Integer(65)));

   // Create dummy argument for OUT parameter and add to parameter
   // vector
   paramData.add(
      new CwDBStoredProcedureParam(PARAM_OUT, new Integer(0));

   // Call the get_empno() stored procedure
   connectn.executeStoredProcedure("get_empno", paramData);

   // Get the result from the OUT parameter
   CwDBStoredProcedureParam outParam =
      (CwDBStoredProcedureParam) paramData.get(1);
   int emp_number = ((Integer) outParam.getValue().intValue();
   }
```

図 95. get_empno() ストアード・プロシージャーの実行

ヒント: Java Vector オブジェクトは、ゼロから始まる配列です。前のコードでは、Vector 配列がゼロから始まるため、Vector パラメーター配列からこの OUT パラメーターの値にアクセスするために、get() 呼び出しは、インデックス値 1 を指定します。

ストアード・プロシージャーは、そのパラメーターを SQL データ型として処理します。SQL および Java データ型は同一ではない ため、executeStoredProcedure() メソッドは、これら 2 つのデータ型間でパラメーター値を変換する必要があります。IN パラメーターの場合、executeStoredProcedure() は、パラメーター値を Java データ型から SQL データ型に変換します。OUT パラメーターの場合、executeStoredProcedure() は、パラメーター値を SQL データ型から Java データ型に変換します。

executeStoredProcedure() メソッドは、JDBC データ型を内部で使用し、ストアード・プロシージャーに送受信されるパラメーター値を保持します。JDBC は、java.sql.Types クラスで汎用 SQL タイプの ID の集合を定義します。これらのタイプは、最も一般的に使用される SQL タイプを示します。また、JDBC には、JDBC タイプから Java データ型への標準マッピングも用意されています。

例: 通常、JDBC INTEGER は Java int タイプにマッピングされます。executeStoredProcedure() メソッドは、表 74 に示すマッピングを使用します。

表 74. Java と JDBC データ型とのマッピング

Java データ型	JDBC データ型
String	CHAR、VARCHAR、または LONGVARCHAR
Integer、int	INTEGER

表 74. *Java と JDBC データ型とのマッピング (続き)*

Java データ型	JDBC データ型
Long	BIGINT
Float、float	REAL
Double、double	DOUBLE
java.math.BigDecimal	NUMERIC
Boolean、boolean	BIT
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Clob	CLOB
java.sql.Blob	BLOB
byte[]	BINARY、VARBINARY、または LONGVARBINARY
Array	ARRAY
Struct	STRUCT

トランザクションの管理

トランザクション は、1 単位として実行される操作ステップの集合です。トランザクション内で実行されるすべての SQL ステートメントは、1 単位として成功または失敗します。このセクションでは、トランザクションの管理に関する以下の情報について説明します。

- 『トランザクション・プログラミング・モデルの判別』
- 238 ページの『トランザクション・スコープの指定』

トランザクション・プログラミング・モデルの判別

トランザクションに対するデータベース操作の実行ステップのグループ化は、トランザクション・ブラケット と呼ばれます。各接続には、以下のトランザクション・プログラミング・モデルの 1 つが関連付けられます。

- 暗黙的なトランザクション・ブラケット: データベース操作は、*暗黙的なトランザクション* の一部です。これは、接続が獲得されると同時に開始され、接続が解放されると終了します。トランザクション・ブラケットは、InterChange Server Express によって暗黙的に管理されます。
- 明示的なトランザクション・ブラケット: データベース操作は*明示的なトランザクション* の一部で、その開始と終了はプログラムで判別されます。

マップ・インスタンスは実行時に、獲得する各接続に対して使用するトランザクション・プログラミング・モデルを決定します。デフォルトでは、マップは獲得するすべての 接続により、そのトランザクション・プログラミング・モデルとして暗黙的なトランザクション・ブラケットが使用されることを前提としています。デフォルトのトランザクション・プログラミング・モデルは、表 75 の任意の方法を使用してオーバーライドできます。

表 75. 接続のトランザクション・プログラミング・モデルのオーバーライド

オーバーライドするトランザクション・プログラミング・モデル	実行するアクション
特定のマップ・インスタンスで取得されたすべての*接続に関する* トランザクション・プログラミング・モデルの指定	「マップ・プロパティー」ダイアログの「一般」タブの「暗黙的なデータベース・トランザクション」チェック・ボックスを選択または選択解除します。 Service Manager の「マップ・プロパティー」ウィンドウでこのプロパティーを設定することもできます。
特定の接続に関する トランザクション・プログラミング・モデルの指定	boolean 値を入力し、(この接続のみに対して) 目的のトランザクション・プログラミング・モデルを getDBConnection() メソッドのオプションの 2 番目の引き数として指定します。 以下の getDBConnection() 呼び出しは、ConnPool 接続プールから取得された接続に明示的なトランザクション・ブラケットを指定します。 `conn = getDBConnection("ConnPool", false);`

BaseDLM.implicitDBTransactionBracketing() メソッドを使用して、接続が使用する現在のトランザクション・プログラミング・モデルを決定できます。このメソッドは、トランザクション・プログラミング・モデルが暗黙的なトランザクション・ブラケットであるかどうかを示す boolean 値を戻します。

トランザクション・スコープの指定

接続のトランザクション・プログラミング・モデルにより、データベース・トランザクションのスコープの指定方法が決まります。このセクションの内容は次のとおりです。

- 『暗黙的なトランザクション・ブラケットでのトランザクション・スコープ』
- 239 ページの『明示的なトランザクション・ブラケットでのトランザクション・スコープ』

暗黙的なトランザクション・ブラケットでのトランザクション・スコープ:

InterChange Server Express は、単一の暗黙的なトランザクションでのマップのアクションを処理します。接続が暗黙的なトランザクション・ブラケットを使用する場合、InterChange Server は、接続プールの接続に関連付けられた、外部データベースで実行された操作のトランザクション管理も処理します。マップでデータベース操作を実行する場合、InterChange Server Express はこれらのデータベース操作を、メイン・トランザクション (マップ) のサブトランザクションである暗黙的な操作としても処理します。このデータベース・サブトランザクションは、マップが接続を取得すると直ちに開始されます。 InterChange Server Express は、マップの実行が完了すると、このサブトランザクションを暗黙的に終了します。

このデータベース・サブトランザクションの成功または失敗は、以下のように、マップの成功または失敗により異なります。

- マップの実行が成功した場合、InterChange Server Express はデータベース・サブトランザクションをコミットします。

- マップの実行が失敗した場合、InterChange Server Express はデータベース・サブトランザクションをロールバックします。このロールバックが失敗すると、InterChange Server Express は、CwDBTransactionException 例外をスローし、エラーをログに記録します。

明示的なトランザクション・ブラケットでのトランザクション・スコープ: 接続で明示的なトランザクション・ブラケットを使用すると、InterChange Server Express はマップ定義を予期して、各データベース・トランザクションのスコープを明示的に指定します。マップの成功または失敗とは関係ないデータベース操作を実行する必要がある場合、明示的なトランザクション・ブラケットが役に立ちます。

例: 特定の表がアクセスされたことを示す監査を実行する必要がある場合、この監査は、テーブルへのアクセスが成功と失敗のいずれであるかとは関係なく実行する必要があります。明示的なトランザクションにデータベース操作の監査が含まれる場合、これらの操作はマップの成功または失敗とは関係なく実行されます。

表 76 は、明示的なトランザクションのトランザクション境界を管理するための CwDBConnection クラスのメソッドを示します。

表 76. 明示的なトランザクションを管理するための CwDBConnection メソッド

トランザクション管理操作	CwDBConnection メソッド
新しいトランザクションを開始します。	beginTransaction()
トランザクションの実行時にデータベースに対して行われたすべての変更をコミット (保管) し、トランザクションを終了します。	commit()
トランザクションが現在アクティブかどうかを確認します。	inTransaction()
トランザクションの実行時に行われたすべての変更をロールバック (バックアウト) し、トランザクションを終了します。	rollBack()

明示的なトランザクションのトランザクション・スコープ指定手順: 明示的なトランザクションのトランザクション・スコープを指定するには、次の手順を実行します。

1. beginTransaction() メソッドを呼び出して、トランザクションを開始します。
2. beginTransaction() の呼び出しとトランザクションの終了との*間の* 単位として成功または失敗する必要があるすべての SQL を実行します。
3. 以下のいずれかの方法により、トランザクションを終了します。
   - commit() を呼び出して、トランザクションを正常終了します。SQL ステートメントで行われたすべての変更は、データベースに*保管* されます。
   - rollBack() を呼び出して、トランザクションを異常終了します。SQL ステートメントで行われたすべての変更は、データベースからバックアウト されます。

   トランザクションが失敗した原因となる条件を選択できます。失敗条件が満たされた場合は、条件をテストし、rollBack() を呼び出します。あるいは、commit() を呼び出して、トランザクションを正常終了します。

要確認: beginTransaction() を使用して明示的なトランザクションの開始を指定しない 場合、データベースは、各 SQL ステートメントを個別トランザクションとして実行します。beginTransaction() を組み込むが、接続が解放される前に commit() または rollback() を使用してデータベース・トランザクションの終了を指定しない 場合、マップが成功したかどうかに基づいて InterChange Server Express により暗黙的にトランザクションが終了されます。マップが成功の場合、InterChange Server Express は、このデータベース・トランザクションをコミットします。マップが成功しなかった 場合、InterChange Server Express はデータベース・トランザクションを暗黙的にロールバックします。マップが成功かどうかにかかわらず、InterChange Server Express は警告を記録します。

以下のコード・フラグメントは、CustDBConnPool の接続に関連付けられたデータベースの 3 つの表を更新します。これらの更新がすべて 成功すると、コード・フラグメントではこれらの変更を commit() メソッドでコミットします。なんらかのトランザクション・エラーが発生すると、CwDBTransactionException 例外となり、このコード・フラグメントは rollback() メソッドを呼び出します。

```
CwDBConnection connection =
getDBConnection("CustDBConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Update several tables
try
   {
   connection.executeSQL("update table1....");
   connection.executeSQL("update table2....");
   connection.executeSQL("update table3....");

   // Commit the transaction
   connection.commit();
   }

catch (CwDBSQLException e)
   {
   // Roll back the transaction if an executeSQL() call throws
   // an exception
   connection.rollback();
   }

// Release the database connection
connection.release();
```

トランザクションが現在アクティブであるかどうかを判別するには、inTransaction() メソッドを使用します。

重要: beginTransaction()、commit()、および rollback() メソッドは、接続で明示的なトランザクション・ブラケットを使用する場合にのみ 使用します。接続で暗黙的なトランザクション・ブラケットを使用する場合は、これらのメソッドを使用すると、CwDBTransactionException 例外が発生します。

接続の解放

接続は、解放されると、その接続プールに戻され、ここでその他のコンポーネントが使用できるようになります。データベースへの接続が解放される方法は、トランザクション・プログラミング・モデルによって決まります。このセクションの内容は次のとおりです。

- 『暗黙的なトランザクション・ブラケットでの接続の解放』
- 『明示的なトランザクション・ブラケットでの接続の解放』

暗黙的なトランザクション・ブラケットでの接続の解放

InterChange Server Express は、データベース・トランザクションを終了すると、暗黙的なトランザクション・ブラケットを使用する接続を自動的に解放します。InterChange Server Express は、マップが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。つまり、InterChange Server Express は、マップ・インスタンスが実行を終了するときにこれらの接続を解放します。マップが正常に実行されると、InterChange Server Express は、アクティブなデータベース・トランザクションを自動的にコミットします。マップの実行が失敗 (例えば、`catch` ステートメントによって処理されない例外がスローされる場合) すると、InterChange Server Express は、アクティブなトランザクションを自動的にロールバックします。

明示的なトランザクション・ブラケットでの接続の解放

明示的なトランザクション・ブラケットを使用する接続の場合、接続は、以下のいずれかの状況で終了します。

- InterChange Server Express は、明示的なトランザクション・ブラケットを使用する接続を自動的に解放します。
- `CwDBConnection` クラスの `release()` メソッドを使用して、接続を明示的に解放できます。

`CwDBConnection.isActive()` メソッドを使用して、接続が解放されたかどうかを判別できます。接続が解放されると、次のコード・フラグメントに示すように、`isActive()` により `false` が戻されます。

```
if (connection.isActive())
   connection.release();
```

重要: トランザクションが現在アクティブな場合は、`release()` メソッドを使用しないでください。暗黙的なトランザクション・ブラケットの場合、InterChange Server Express は、マップが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。このため、暗黙的なトランザクション・ブラケットを使用する接続にこのメソッドを使用すると、`CwDBTransactionException` 例外が発生します。この例外を明示的に処理しないと、アクティブなトランザクションも自動的にロールバックされます。`inTransaction()` メソッドを使用すると、トランザクションがアクティブであるかどうかを判別できます。InterChange Server Express は、使用しているトランザクション・プログラミング・モデルとは関係なく、接続を自動的に解放します。多くの場合、接続は明示的に解放する必要はありません。

第 2 部 関係

第 6 章 関係の概要

この章では、WebSphere Business Integration の関係および関係開発プロセスの概要について説明します。

この章の内容は次のとおりです。

- 245 ページの『関係とは』
- 251 ページの『関係: 詳細』
- 257 ページの『関係開発プロセスの概要』

関係とは

ソース・ビジネス・オブジェクトと宛先ビジネス・オブジェクトの属性に、表示が異なる同等のデータが含まれている場合、変換ステップは*関係* を使用します。関係は、2 つ以上のビジネス・オブジェクトのデータ間の関連を設定します。各ビジネス・オブジェクトは、関係の*参加者* と呼ばれます。

通常は、関係を使用して変換するデータは、次のとおりです。

- 顧客 ID や製品 ID などの ID 番号
- 国、通貨、既婚/独身など、コードとして表されるその他の値

例: アプリケーション A は顧客 ID に連続する整数を使用し、アプリケーション B は生成された顧客コードを使用するとします。`TashiCo` は、アプリケーション A では顧客 ID `806`、アプリケーション B では `A100` です。アプリケーション A と B の間で顧客 ID データを転送するには、顧客 ID 属性に基づいて、アプリケーション A の顧客ビジネス・オブジェクト、汎用顧客ビジネス・オブジェクト、およびアプリケーション B 顧客ビジネス・オブジェクト間に関係を作成します。

この関係は、顧客ビジネス・オブジェクトのキー属性に基づいて、アプリケーション A とアプリケーション B の顧客間に関連を設定します。図 96 の各ボックスは `CustIden` という関係の参加者を示しています。

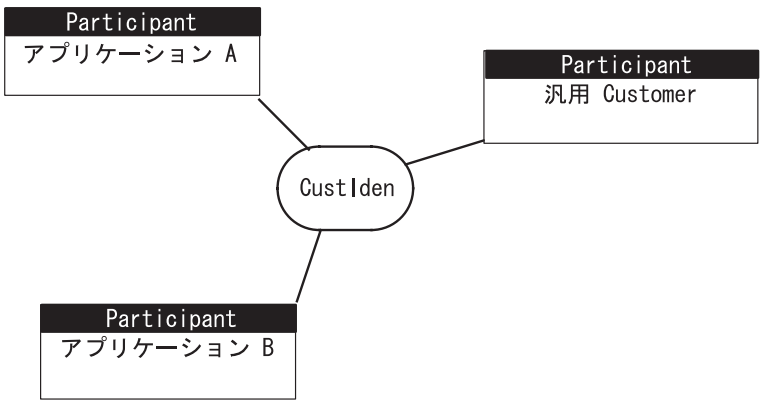

図 96. 3 つの参加者の関係

関係は、参加者のデータ型と、関連する各参加者のインスタンス数に基づいて、次のカテゴリーに分類されます。

- *参照関係* は、ビジネス・オブジェクトの属性など、データ間の関連を設定します。データは、*1 対 1*、*1 対多*、または*多対多* で関連させることができます。参照関係は通常、既婚/独身や通貨コードなど、値がコードで表される非キー属性を変換します。これらの属性値が静的な場合、つまり新しい値が追加されたり既存の値が除去されることが少ない場合に、参照関係を使用します。
- *一致関係* は、ビジネス・オブジェクトまたは他のデータ間に *1 対 1* の関連を設定します。関係インスタンスごとに、各参加者の 1 つのインスタンスのみ設定できます。一致関係は通常、ID 番号や製品コードなど、ビジネス・オブジェクトのキー属性を変換します。図 96 の関係は、一致関係の例を示します。キー値が動的な場合、つまりキー値が頻繁に追加されたり、既存の値が除去される場合に、一致関係を使用します。
- *非一致関係* は、ビジネス・オブジェクトまたは他のデータ間に *1 対多* または*多対多* の関連を設定します。関係インスタンスごとに、各参加者の複数のインスタンスを設定できます。非一致関係の例は、RMA-to-Order 変換です。この変換では、単一の RMA (Return Materials Authorization) ビジネス・オブジェクトが、1 つ以上の Order ビジネス・オブジェクトを作成できます。

参照関係

参照関係 は、2 つの非キー・データを関連付けます。例えば、`Clarify_Site` から `Customer` へのマップでは、参照関係を使用して、`SiteStatus` など、値がコードや省略形で表された属性を変換します。参照関係では、アプリケーション固有の各ビジネス・オブジェクトに 1 つの参加者が存在します。

図 97 の `CustLkUp` 関係では、Clarify および SAP アプリケーションの顧客状況コード間に、参照関係が設定されます。各ボックスは、`CustLkUp` 参照関係の参加者を表します。この関係には、アプリケーション固有の各ビジネス・オブジェクトに 1 つずつ、計 2 つの参加者があることに注意してください。

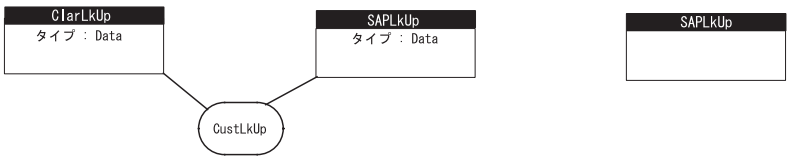

図 97. CustLkUp 参照関係定義

注: 参照関係は、関連する属性を表示しないため、参加者は `Data` と呼ばれる特別なタイプを使用します。詳細については、256 ページの『参加者タイプ』を参照してください。

例: Clarify アプリケーションは、サイト状況が `Inactive` の非アクティブな顧客を表し、SAP では対応する値は `05` です。これらの顧客状況コードは異なりますが、図 98 に示すように、同じ状況を表します。

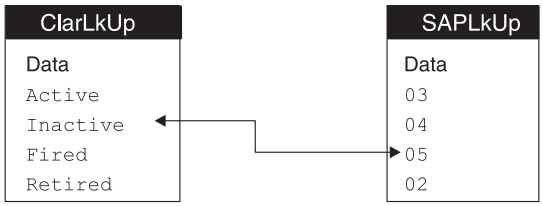

図 98. CustLkUp 参照関係の関係データ

表 77 に、参照関係の作成に必要な手順を示します。

表 77. 参照関係の作成手順

作成手順		詳細情報の参照先
1.	Relationship Designer Express で参照関係を定義します。	270 ページの『参照関係の定義』
2.	参照関係を保持するように、マッピング・コードをカスタマイズします。	282 ページの『参照関係の使用』
3.	参照関係をテストして、正しくインプリメントされているかを検証します。	107 ページの『参照関係のテスト』

一致関係

一致関係 は、ビジネス・オブジェクトまたは他のデータ間に *1 対 1* の関連を設定します。 1 対 1 の関係を保持するには、各ビジネス・オブジェクトにキーが必要です。つまり、オブジェクトには、値がオブジェクトを一意的に識別する、少なくとも 1 つの属性 (*キー属性*) が含まれます。両方のビジネス・オブジェクトにキーが含まれる場合は、両方が一致関係に参加できます。

WebSphere Business Integration システムは、次の種類の一致関係をサポートしています。

- 『単純一致関係』
- 249 ページの『複合一致関係』

両方の種類の一致関係に、関連するビジネス・オブジェクト属性が含まれています。したがって、一致関係の各参加者は、参加者タイプとしてビジネス・オブジェクトを持っています。参加者タイプの詳細については、256 ページの『参加者タイプ』を参照してください。

単純一致関係

単純一致関係 は、2 つのビジネス・オブジェクトを単一のキー属性によって関連付けます。つまり、各ビジネス・オブジェクトには、オブジェクトを一意的に識別する単一値が含まれます。

例: CustIden 関係 (図 96 を参照) が詳細化され、顧客ビジネス・オブジェクトのキー属性に基づいて、Clarify および SAP アプリケーションから顧客間の関連を設定するとします。 図 99 の各ボックスは、この顧客一致関係の参加者を表します。この関係には、アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトに 1 つずつ参加者があることに注意してください。

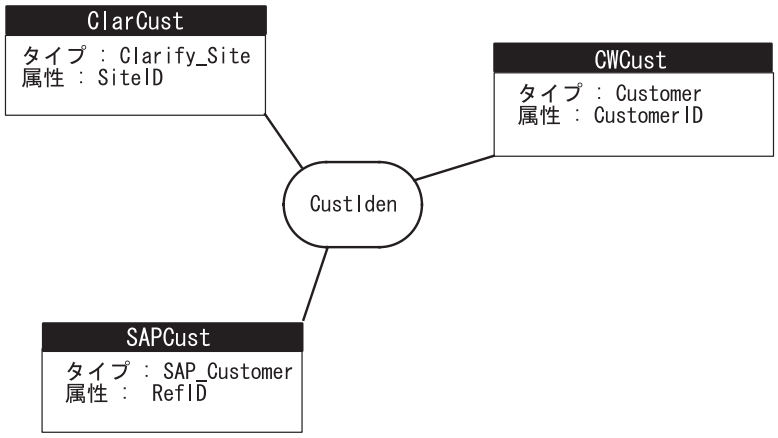

図 99. CustIden の単純一致関係定義

TashiCo 会社は、Clarify アプリケーションではキー値 A100 で識別され、SAP アプリケーションでは、同じ会社がキー値 806 で識別されます。これらのアプリケーション ID は異なりますが、図 100 に示すように、同じ顧客を表します。

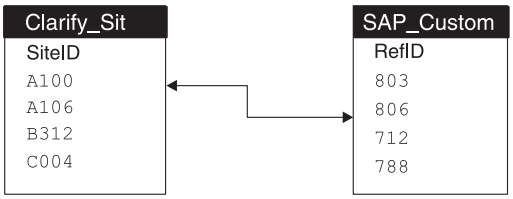

図 100. custIden 単純一致関係の関係データ

したがって、次のマップは、単純一致関係を使用して、キー属性間の変換を保持します。

- インバウンド・マップ (Clarify アプリケーション固有のビジネス・オブジェクトと汎用 Customer ビジネス・オブジェクト間) は、単純一致関係を使用して、Clarify_Site ビジネス・オブジェクトの SiteID 属性と、汎用 Customer ビジネス・オブジェクトの汎用 CustomerID 属性間の変換を保持します。
- アウトバウンド・マップ (汎用 Customer ビジネス・オブジェクトと SAP アプリケーション固有のビジネス・オブジェクト間) も、単純一致関係を使用して、SAP_Customer ビジネス・オブジェクトの RefID 属性と、汎用 Customer ビジネス・オブジェクトの汎用 CustomerID 属性間の変換を保持します。

表 78 に、単純一致関係の作成に必要な手順を示します。

表 78. 単純一致関係の作成手順

作成手順		詳細情報の参照先
1.	Relationship Designer Express で単純一致関係を定義します。	267 ページの『一致関係の定義』
2.	単純一致関係を保持するように、マッピング・コードをカスタマイズします。	287 ページの『単純一致関係の使用』
3.	単純一致関係をテストして、正しくインプリメントされているかを検証します。	104 ページの『一致関係のテスト』

複合一致関係

複合一致関係 は、2 つのビジネス・オブジェクト複合キーによって関連付けます。「複合」という用語が示すように、複合キーは、複数の属性で構成されるキーです。オブジェクトを一意的に識別するには、すべて の属性の値が必要です。複合キーは、親ビジネス・オブジェクトの固有キーと、子ビジネス・オブジェクトの非固有キーで構成されます。

例: Clarify アプリケーションで `TashiCo` からの特定の注文がキー値 8765 で識別されるとします。SAP アプリケーションでの同じ注文は、キー値 0003411 で識別されます。2 つの注文番号は一意的に同じ注文を識別するため、キー属性は単純一致関係に関連付けられます。ただし、注文には注文明細も含まれます。すべての参加アプリケーションが固有の値を使用してこれらの注文明細を識別する場合、単純一致関係はその注文明細の変換を保持できます。

ただし、アプリケーションが明細番号のみを使用して注文明細品目を識別する場合がよくあります。つまり、各注文には、1 で識別される明細品目と、2、3 などで識別される後続の品目が含まれます。これらの明細番号は、注文明細品目を一意的に識別しません。このような品目を一意的に識別するために、アプリケーションは複合キーを使用します。複合キーは、(親注文ビジネス・オブジェクトからの) 注文番号と、(子注文明細ビジネス・オブジェクトからの) 明細番号で構成されます。

図 101 では、`OrdrLine` 関係は、複合キー属性に基づいて、Clarify および SAP アプリケーションから注文明細間の関係を設定します。複合キー属性は、親注文ビジネス・オブジェクトの固有キー属性と、子注文明細ビジネス・オブジェクトの注文明細番号の組み合わせです。各ボックスは、`OrdrLine` 複合一致関係の参加者を表します。各参加者には 2 つの属性があることに注意してください。

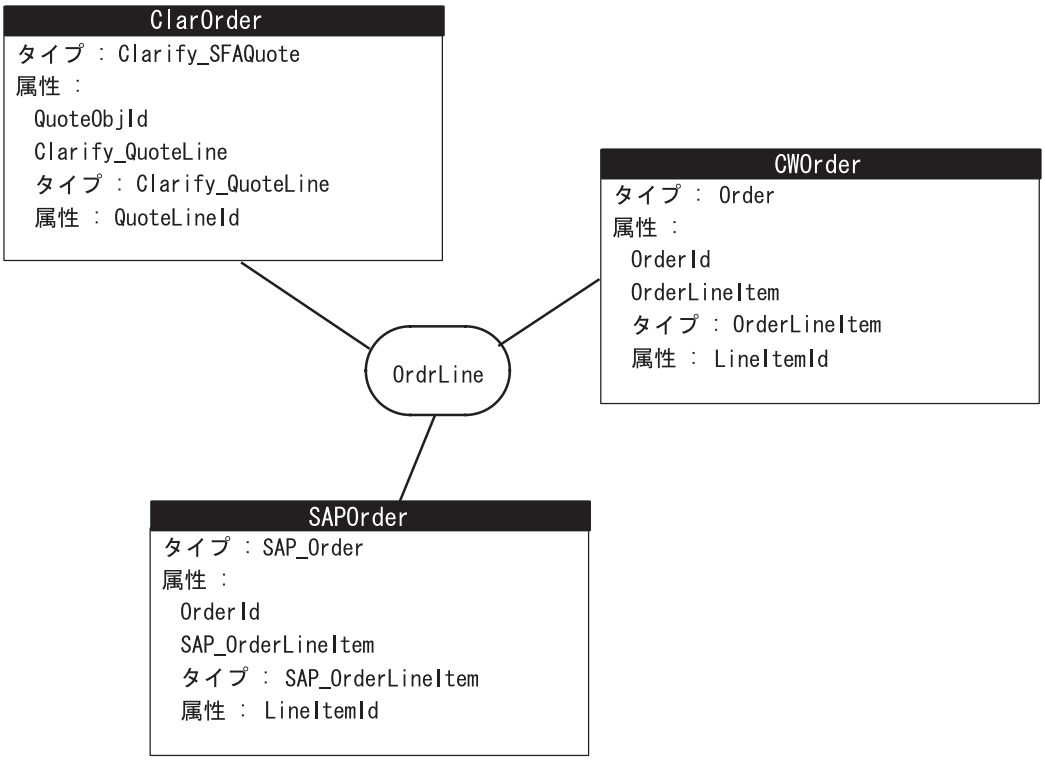

図 101. OrdrLine の複合一致関係定義

例: Clarify アプリケーション (図 101 の参加者 ClarOrder で表されます) は、連続した整数を使用して注文明細を識別し、SAP アプリケーションは、明細番号を使用してこれらの品目を識別するとします。Clarify アプリケーションは、各注文明細品目を一意的に識別します。したがって、Clarify アプリケーション固有のビジネス・オブジェクトと汎用 Order ビジネス・オブジェクト (参加者 CWOrder で表されます) 間のマップは、単純一致関係を使用して、注文明細品目の変換を保持できます。

ただし、SAP アプリケーション (参加者 SAPOrder で表されます) は、明細番号を使用して注文明細品目を識別します。この場合、品目は一意的に識別されません。すべての注文には、1 で識別される明細品目と、2、3 などの番号が付いた後続の品目が含まれます。注文 0003411 の 3 番目の注文明細品目を一意的に識別するには、複合キーを使用する必要があります。複合キーには、注文番号 (0003411) と明細番号 (3) の両方が含まれます。したがって、SAP アプリケーション固有のビジネス・オブジェクトと汎用 Order ビジネス・オブジェクト間のマップは、複合一致関係を使用して、注文明細品目の変換を保持する必要があります。

TashiCo 注文の 3 番目の明細品目 (8765) は、Clarify アプリケーションでは単一キー値 1171 で識別されます。ただし、この同じ明細品目は、SAP アプリケーションでは複合キー値 0003411 (注文番号) と 3 (明細番号) で識別されます。これらの注文明細は異なって識別されますが、図 102 に示すように、同じ注文明細品目を表しています。

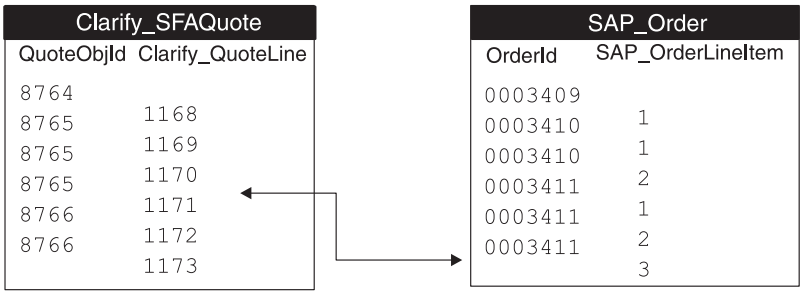

図 102. OrdrLine 複合一致関係の関係データ

表 79 に、複合一致関係の作成に必要な手順を示します。

表 79. 複合一致関係の作成手順

作成手順	詳細情報の参照先
1. Relationship Designer Express で複合一致関係を定義します。	267 ページの『一致関係の定義』
2. 複合一致関係を保持するように、マッピング・コードをカスタマイズします。	300 ページの『複合一致関係の使用』
3. 複合一致関係をテストして、正しくインプリメントされているかを検証します。	104 ページの『一致関係のテスト』

関係: 詳細

WebSphere Business Integration システムがサポートする関係のタイプを理解するには、IBM が次の概念をどのようにインプリメントしているかを理解する必要があります。

- 『関係』
- 255 ページの『参加者』

関係

表 80 に示すように、関係は 2 つの部分からなるエンティティーで、リポジトリー・エンティティーとランタイム・オブジェクトで構成されます。

表 80. 関係のパーツ

リポジトリー・エンティティー	ランタイム・オブジェクト
関係定義	関係インスタンス

関係定義

WebSphere Business Integration システムとの関係は、*関係定義* を使用して定義します。関係定義は、各参加者を識別して、参加者を関連付ける方法を指定します。図 96 では、`CustIden` は関係定義で、アプリケーション A、アプリケーション B、汎用顧客 という 3 つの参加者に関する情報を含んでいます。

システムは、関係定義をリポジトリーに保管します。 Relationship Designer Express ツールは、関係定義の作成に役立つダイアログを提供します。このツールを使用すると、完了した関係定義をリポジトリーに保管することもできます。

ヒント: Relationship Designer Express を使用して関係定義を作成する方法の詳細について、 263 ページの『メインウィンドウのカスタマイズ』を参照してください。

関係定義は、関係に関する次の情報を提供します。

- 関係名
- リレーションシップ・データベースの名前

関係定義名: 関係定義は、関係の単なるテンプレートまたは説明であり、実際のビジネス・オブジェクトではありません。したがって、関係定義の名前は、関連するビジネス・オブジェクトの名前にすることはできません。

リレーションシップ・データベース: *リレーションシップ・データベース* は、関係の関係表を保持します。関係は、関係表を使用して、関連するアプリケーション固有の値を追跡します。詳細については、253 ページの『関係表』を参照してください。

実行時にリレーションシップ・データベースにアクセスするには、次の情報が必要です。

- リレーションシップ・データベースを管理するデータベース管理システム (DBMS) のタイプ
- リレーションシップ・データベースにアクセスするユーザー・アカウントの名前とパスワード
- リレーションシップ・データベースの位置

デフォルトでは、リレーションシップ・データベースは WebSphere Business Integration システム・リポジトリーです。つまり、Relationship Designer Express は、リポジトリーのすべての関係表を作成します。 Relationship Designer Express を使用すると、次のいずれかの方法で関係表の位置を指定できます。

- すべての 関係のリレーションシップ・データベースのデフォルト位置を変更します。

  詳細については、275 ページの『グローバル・デフォルト設定』を参照してください。
- 関係定義を作成するプロセスの一部として、各関係の表の位置をカスタマイズします。

  詳細については、273 ページの『関係定義の拡張設定』を参照してください。

関係インスタンス

関係定義は、関係のランタイム・インスタンス化 (*関係インスタンス*) を行うためのテンプレートです。マップの実行時に、関係定義に基づいて、変換される実際のビジネス・オブジェクトの値を使用して、関係のインスタンスが作成されます。

例: `CustLkUp` 参照関係の関係データ (図 98 を参照してください) は、Clarify アプリケーションの顧客状況 `Inactive` が、SAP アプリケーションの顧客状況 `05` と同じであることを示しています。これらの状況コードは異なりますが、図 103 に示すように、同じ顧客状況を表しているため、同じ関係インスタンスにあります。

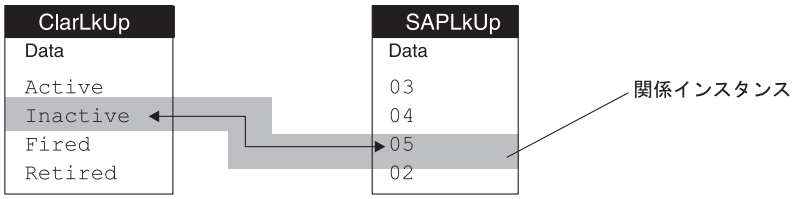

図 103. CustLkUp 関係のある関係インスタンス

関係インスタンスは、Relationship クラスまたは IdentityRelationship クラスのインスタンスによって、マッピング API で表されます。

関係インスタンスを検索するには、次の情報が必要です。

- 特定の参加者の関係インスタンスを含む表を識別するための*関係表*
- 関係表内の実際の関係インスタンスを識別するための*関係インスタンス ID*

関係表: *関係表* は、関係のある参加者に関する関係ランタイム・データを保持するデータベース表です。 InterChange Server Express は、関係インスタンスを関係表に保管します。1 つの関係表 (*参加者表* とも呼ばれる) には、その関係における 1 つの参加者に関する情報が保管されます。例えば、図 97 の CustLkUp 参照関係の場合、InterChange Server Express は図 103 に示すように、2 つの参加者表を必要とします。

関係定義を作成すると、Relationship Designer Express によって、関係が必要とする表スキーマが自動的に作成されます。つまり、各参加者に必要な列を含む関係表が作成されます。実行時に、これらの表は関係インスタンスのデータを保持します。

注: 一致関係の場合、InterChange Server Express は、関係表を自動的に取り込みます。参照関係の場合は、データとともに関係表を取り込む必要があります。詳細については、283 ページの『参照表にデータを設定』を参照してください。

実行時に関係表にアクセスするには、次の情報が必要です。

- 関係表の名前

  関係表は参加者に関連付けられているため、この表の名前は、参加者定義の一部として定義されます。デフォルトでは、関係表の名前の形式は次のとおりです。

  RelationshipDefName_ParticipantDefName

  Relationship Designer Express を使用すると、参加者定義を作成するプロセスの一部として、関係表の名前をカスタマイズできます。

  詳細については、274 ページの『参加者定義の拡張設定』を参照してください。
- 関係表を含むデータベースの名前

  リレーションシップ・データベースの名前は、関係定義の一部として設定されます。デフォルトでは、リレーションシップ・データベースはシステム・リポジトリーです。詳細については、273 ページの『関係定義の拡張設定』を参照してください。

マップ変換ステップでは、関係表は、Relationship クラス、IdentityRelationship クラス、および Participant クラスのメソッドを使用して管理されます。マッピン

グ API メソッドによっては、関係表を自動的に管理するものもあります。これらの関係表に明示的にアクセスして、この関係データを取得することもできます。

関係インスタンス ID: WebSphere Business Integration システムは、*関係インスタンス* と呼ばれる固有の整数値を割り当てることで、各関係インスタンスを一意的に識別します。このインスタンス ID を使用すると、参加者値を関連付けることができます。通常は、関係の任意の参加者の場合、関係インスタンス ID を指定することで、関係の他の参加者に関するデータを検索できます。

例: Clarify アプリケーションと SAP アプリケーションの顧客状況コード間の関係の場合、WebSphere Business Integration システムは、参照関係の各関係インスタンスに関係インスタンス ID を割り当てます。図 104 に、ClarLkUpと SAPLkUp の 2 つのアプリケーション固有の参加者がインスタンス ID 47 を介して関連付けられている状態を示します。 ID 47 が Inactive の Clarify 顧客状況と、05 の SAP 顧客状況値をどのように関連付けるかを示します。この関係は基本的に、関係インスタンス ID を追加することで、図 103 の関係と同じであることに注意してください。

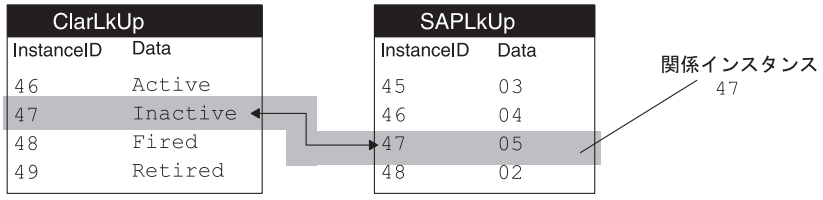

図 104. 関係インスタンス ID のある参照関係

WebSphere Business Integration システムは、一致関係の参加者間の関係にも関係インスタンス ID を使用します。 CustIden 関係 (図 99 を参照してください) では、このインスタンス ID は、Clarify_Site ビジネス・オブジェクトの SiteID 属性、汎用 Customer ビジネス・オブジェクトの CustomerID 属性、および SAP_Customer ビジネス・オブジェクトの RefID 属性に保管された顧客 ID を関連付けます。図 105 に、CustIden 関係の各参加者の関係インスタンス・データが、関係インスタンス ID を使用して関連付けられる方法を示します。

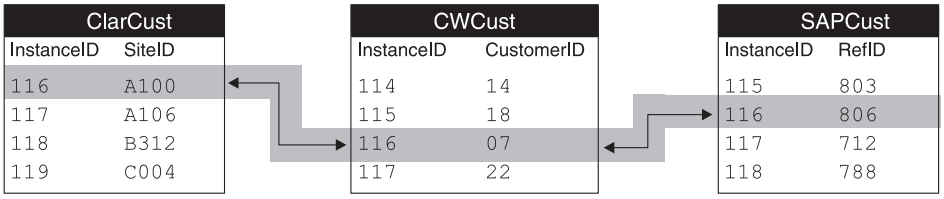

図 105. 関係インスタンス ID との顧客 ID 関係

図 105 では、わかりやすくするために CWCust 参加者の関係表が含まれていますが、表は厳密には必要ありません。実際には、関係の汎用ビジネス・オブジェクトを表す参加者の関係表は、汎用ビジネス・オブジェクトの関連する属性に汎用 ID を生成する場合のみ 必要です。図 105 の関係は、汎用 Customer ビジネス・オブジェクトの CustomerID 属性の汎用 ID (07) を生成します。

関係定義を簡略化して、汎用ビジネス・オブジェクトを表す参加者の関係表を除去することで、パフォーマンスを向上させることができます。これは、関係定義を作

成するときに、参加者の「managed」オプションを選択して実行します。この設定の詳細については、274 ページの『参加者定義の拡張設定』を参照してください。

CWCust 参加者に対して managed 設定が指定された場合、CustIden 関係で関係インスタンス・データがどのように関連付けられるかを、図 106 に示します。

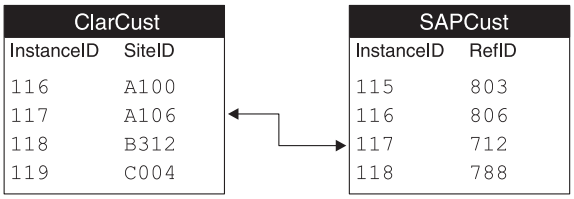

図 106. 汎用表のない一致関係インスタンス

WebSphere Business Integration システムは、関係インスタンス ID を各参加者の関係表に保管します。 図 104 から図 106 に示すように、関係の各関係表には、関係インスタンス ID の列があります。このインスタンス ID 列は、ICS Express で表スキーマが作成されるときに、自動的に作成されます。

参加者

関係には*参加者* が含まれます。参加者は、関係に参加するエンティティーを説明したものです。表 81 に示すように、参加者は 2 つの部分からなるエンティティーで、リポジトリー・エンティティーとランタイム・オブジェクトで構成されます。

表 81. 参加者のパーツ

リポジトリー・エンティティー	ランタイム・オブジェクト
参加者定義	参加者インスタンス

参加者定義

関係定義には、*参加者定義* のリストが含まれています。例えば、図 99 の CustIden 関係定義は、Clarify および SAP の顧客ビジネス・オブジェクトを関連付け、SAPCust、CWCust、および ClarCust という参加者定義を含んでいます。

WebSphere Business Integration システムは、参加者定義をリポジトリーに保管します。 Relationship Designer Express ツールは、参加者定義の作成に役立つダイアログを提供します。このツールを使用すると、完了した参加者定義をリポジトリーに保管することもできます。

参加者定義は、参加者に関する次の情報を提供します。

- 参加者名
- 参加者タイプ
- 参加者表とストアード・プロシージャーの名前

参加者定義名: 参加者定義は、参加者の単なるテンプレートまたは説明であり、実際のビジネス・オブジェクトではありません。したがって、参加者定義の名前は、関連するビジネス・オブジェクトの名前にすることはできません。

参加者タイプ: ビジネス・オブジェクト定義の属性と同様に、関係定義の参加者にも関連するタイプがあります。参加者タイプは、参加者のインスタンスに関連付けられたデータの種類を指定します。参加者タイプは、次のいずれかです。

- ビジネス・オブジェクト定義の名前

  このタイプの参加者を含む関係は、ビジネス・オブジェクト全体の関連を設定します。この場合、参加者を関係の他の参加者に関連付けるビジネス・オブジェクトの属性を指定します。選択した属性 (通常はビジネス・オブジェクトのキー属性) は、*参加者インスタンス ID* になります。

- Data (語)

  参加者定義では、Data は、String、long、int、double、float、または boolean など、サポートされている属性データ型を表します。ビジネス・オブジェクトの特定の属性間に関連を設定する関係の参加者のタイプとして、Data を指定します。参照関係の参加者には、Data の参加者タイプがあります。

参加者のタイプの定義方法の詳細については、266 ページの『関係定義の作成』を参照してください。

参加者表とストアード・プロシージャー: InterChange Server Express は、各参加者ごとに以下のデータベース・エンティティーを作成します。

- 関係インスタンス ID と関連する参加者のアプリケーション固有の値を保持する参加者表
- 参加者表で Retrieve (Select)、Insert、Delete、および Update 操作を実行するストアード・プロシージャー

デフォルトでは、Relationship Designer Express は、参加者の表とストアード・プロシージャーに *RelName_ParticipantName_X* という形式の名前を割り当てます。*RelName* は関係定義の名前、*ParticipantName* は参加者定義の名前、*X* は、参加者表の場合は T、ストアード・プロシージャーの場合は SP になります。デフォルトでは、Relationship Designer Express は、WebSphere Business Integration システム・リポジトリーに関係表を作成します。

Relationship Designer Express を使用すると、参加者表とストアード・プロシージャーの名前をカスタマイズできます。参加者表とストアード・プロシージャーの命名の詳細については、274 ページの『参加者定義の拡張設定』を参照してください。

参加者インスタンス

参加者定義は、参加者のランタイム・インスタンス化 (*参加者インスタンス*) を行うためのテンプレートです。マップの実行時に、WebSphere Business Integration システムは、参加者定義と、変換される実際のビジネス・オブジェクトの属性値に基づいて、参加者のインスタンスを作成します。

WebSphere Business Integration システムは、参加者の関係表の列として参加者インスタンスを保管します。

例: 図 99 の CustIden 関係の場合、ClarCust 参加者には、参加者インスタンスの値を保持するために、参加者表に SiteID という列があります。SAPCust 参加者には、参加者インスタンスの値を保持するために、参加者表に RefID という列があります。

各参加者インスタンスには、以下の情報が含まれています。

- 関係定義の名前
- 関係インスタンス ID
- 参加者定義の名前
- 参加者に関連付けられたデータ

参加者インスタンスは、`Participant` クラスのインスタンスによって、マッピング API で表されます。

関係開発プロセスの概要

WebSphere Business Integration システムの関係は、次の 2 つの部分からなるエンティティーです。

- 参加者を定義して、リポジトリーに保管される関係定義
- 関係表にアクセスすることで関係をインプリメントするためのマップ内のコード

WebSphere Business Integration システムで関係を定義するには、以下の基本ステップを行ってください。

1. 必要な関係のタイプを決定します。
2. Relationship Designer Express 内で、関係定義を定義して、複合参加者を定義します。
3. 必要に応じて、関係を保守できるように変換規則を Map Designer Express でカスタマイズします。
4. 影響を受けるマップを再コンパイルします。
5. 「スキーマを作成」オプションを選択してから InterChange Server Express に関係とマップを配置します。
6. リレーションシップ・データベースが存在し、関係定義内で正しく定義されていることを確認します。
7. 参照関係の関係表を取り込みます。オプションで、テスト・フェーズ用にテスト・データとともに他の関係表を取り込みます。
8. マップごとに、マップのすべての関係を開始します。
9. Test Connector を使用して関係をテストします。各テストの一部として、適切な呼び出しコンテキストを設定してください。

図 107 に、関係開発プロセスの概要を示します。また、特定のトピックについての情報がどの章に記載されているかについても示します。マップの開発をチームで行う場合、マップの開発の主要なタスクをチームのメンバーが並行して行うことができます。

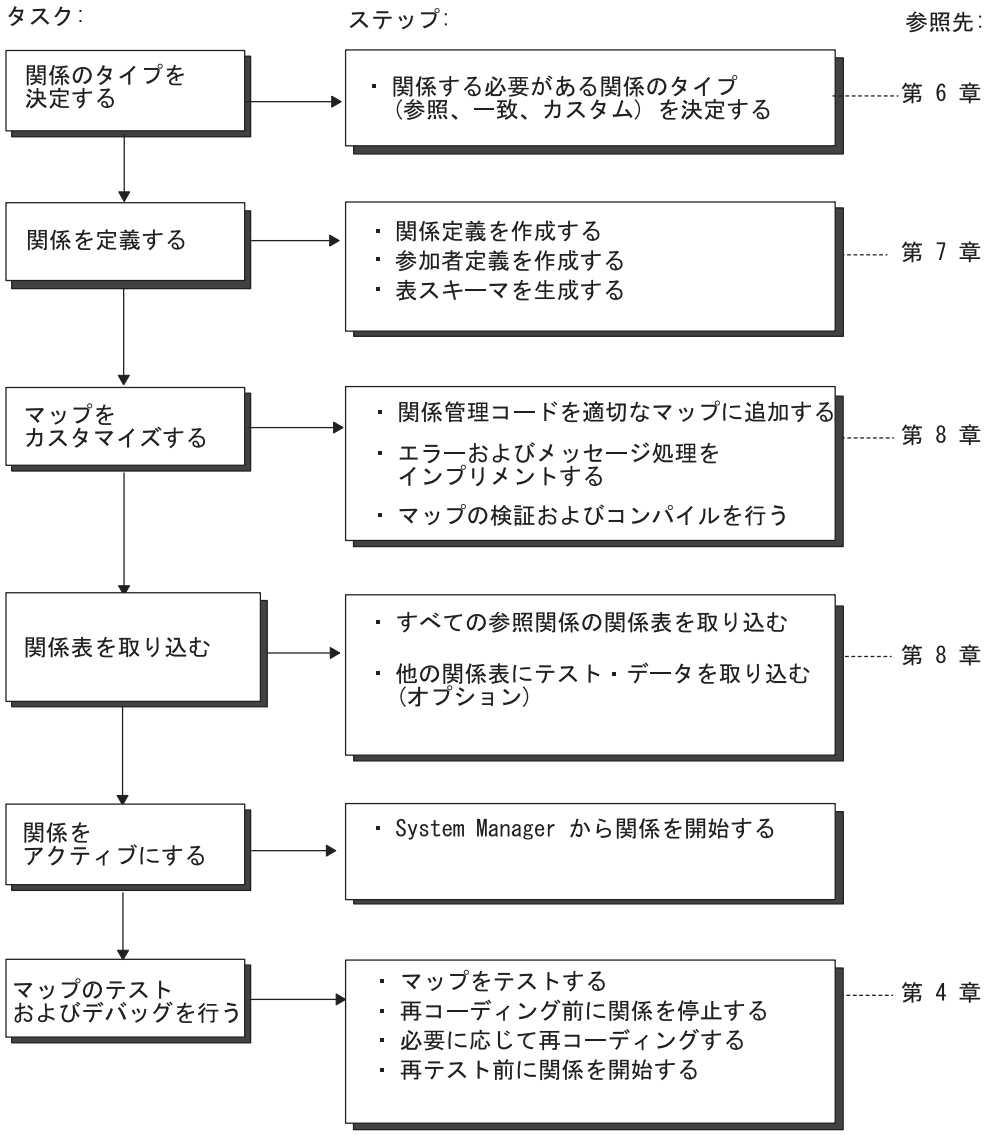

図 107. 関係開発タスクの概要

第 7 章 関係定義の作成

この章では、Relationship Designer Express を使用しての関係定義の作成方法と変更方法について説明します。WebSphere Business Integration システムでのマッピングの関係の使用方法の背景情報については、245 ページの『第 6 章 関係の概要』を参照してください。マップ内の関係のカスタマイズに役立つ情報については、113 ページの『第 5 章 マップのカスタマイズ』を参照してください。

この章の内容は次のとおりです。

- 『Relationship Designer Express の概要』
- 266 ページの『関係定義の作成』
- 267 ページの『一致関係の定義』
- 270 ページの『参照関係の定義』
- 271 ページの『関係表スキーマの作成』
- 271 ページの『関係定義と参加者定義のコピー』
- 272 ページの『関係定義または参加者定義の名前変更』
- 272 ページの『関係の拡張設定の指定』
- 277 ページの『関係定義の削除』
- 277 ページの『関係の最適化』

Relationship Designer Express の概要

Relationship Designer Express は関係定義の作成と変更を行うグラフィカル開発ツールです。*関係定義* は 2 つ以上の参加者間の関連を設定します。関係定義を作成するには、関係の参加者を指定し、各参加者に関連したデータ・ソースおよびその他のプロパティーを定義します。

このセクションには、Relationship Designer Express の概要を説明する以下のトピックがあります。

- 『Relationship Designer Express の始動』
- 260 ページの『プロジェクトの処理』
- 261 ページの『Relationship Designer Express のレイアウト』
- 263 ページの『メインウィンドウのカスタマイズ』
- 263 ページの『Relationship Designer Express の機能の使用法』

Relationship Designer Express の始動

Relationship Designer Express を起動するには、次のいずれかを行います。

- System Manager から、以下のアクションのいずれかを実行します。
  - 「ツール」メニューから「Relationship Designer Express」を選択します。
  - プロジェクトの関係フォルダーをクリックし、System Manager のツールバーで「Relationship Designer Express」アイコンを使用可能にします。「Relationship Designer Express」アイコンをクリックします。

– プロジェクト内の関係フォルダーを右マウス・ボタンでクリックし、コンテキスト・メニューから「Relationship Designer Express」を選択します。
– 動的フォルダーまたは静的フォルダーの関係を右マウス・ボタンでクリックし、コンテキスト・メニューから「定義を編集」を選択します。

結果: Relationship Designer Express が起動し、選択された関係が強調表示されます。

- Business Object Designer Express、Map Designer Express、Process Designer Express などの開発ツールから、次のいずれかのアクションを実行します。
  – 「ツール」メニューから「Relationship Designer Express」を選択します。
  – プログラム・ツールバーで、「Relationship Designer Express」ボタンをクリックします。

  制約事項: Process Designer Express は、WebSphere Business Integration Express Plus でのみ使用可能な開発ツールです。
- システム・ショートカットを使用して、以下のように順に選択していきます。

```
Start > Programs > IBM WebSphere Business Integration Express
  > Toolset Express > Development > Relationship
 Designer Express
```

要確認: Relationship Designer Express が System Manager に保管されている関係にアクセスできるようにするには、Relationship Designer Express は System Manager のインスタンスに接続されている必要があります。前述の手順では、System Manager がすでに始動していることを想定しています。System Manager がすでに実行されている場合は、Relationship Designer Express は自動的に接続します。

プロジェクトの処理

System Manager はサーバーと対話する唯一のツールです。これは、InterChange Server Express と System Manager プロジェクトの間でエンティティー (関係、マップ) のインポートおよびエクスポートを行います。Relationship Designer Express などのさまざまなツールは、System Manager に接続してプロジェクト・ベースでこれらのエンティティーの表示、編集、および変更を行います。

プロジェクト は、管理および展開のために単にエンティティーを論理的にグループ化したものです。エンティティーを InterChange Server Express に展開すると、元になったプロジェクトは意味を失います。

System Manager では複数のプロジェクトを作成できます。関係を処理するには、関係が存在するプロジェクトを選択する必要があります。

プロジェクトの選択手順

処理するプロジェクトを選択するには、以下の手順を実行します。

1. 「ファイル」メニューから「プロジェクトへの切り替え」を選択します。
2. 「プロジェクトへの切り替え」サブメニューでプロジェクトの名前を選択します。

結果: そのプロジェクトの関係を処理できるようになります。他のプロジェクトに切り替える前に、現在のプロジェクトで変更した関係を保管する必要があります。

図 108 に、「プロジェクトへの切り替え」オプションでプロジェクトをブラウズする様子を示します。

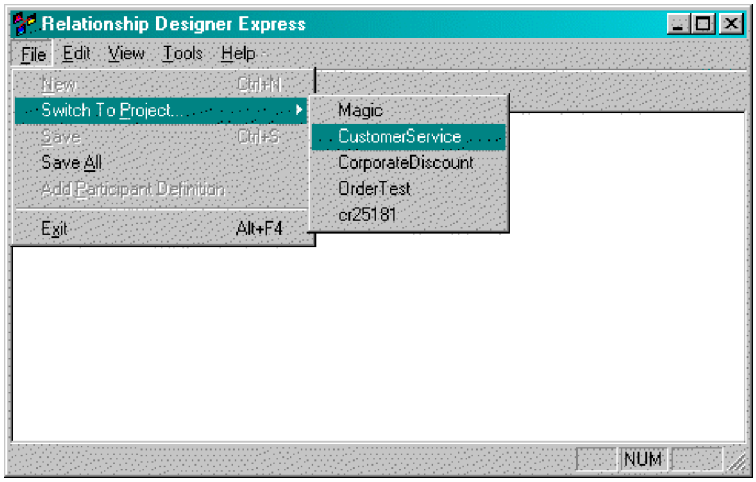

図 108. プロジェクトのブラウズ

Relationship Designer Express は、System Manager への接続を確立すると、現在のプロジェクトで定義されているビジネス・オブジェクトのリストを取得します。このリストでは参加者を定義できます。

Business Object Designer Express を使用してビジネス・オブジェクトを追加または削除すると、System Manager は Relationship Designer Express に通知し、ビジネス・オブジェクト定義のリストが動的に更新されます。

Relationship Designer Express のレイアウト

「Relationship Designer Express」ウィンドウで、現在のプロジェクトに保管されている関係定義のリストは左側に表示されます。この関係定義リストで、各関係定義の内容は Windows エクスプローラーに似た階層形式で表示されます。関係名の横にある正記号 (+) をクリックして関係名を展開すると、その参加者定義、参加者タイプ、および関連する属性のリストを表示できます。図 109 に、関係定義リストを示します。

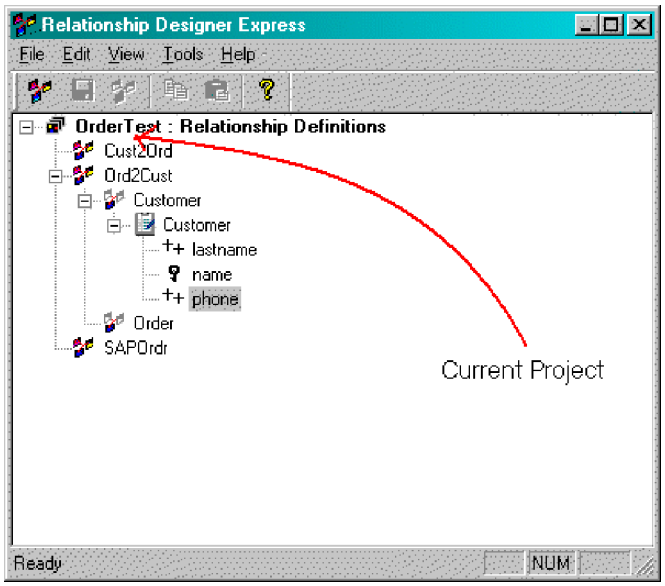

図 109. 関係定義リスト

「参加者タイプ」ウィンドウには、参加者に関連付けることができる、現在のプロジェクトで使用可能なデータ型のリストが表示されます。

図 110 に、「関係定義」リストおよび「参加者タイプ」ウィンドウの両方を表示した Relationship Designer Express のメインウィンドウを示します。

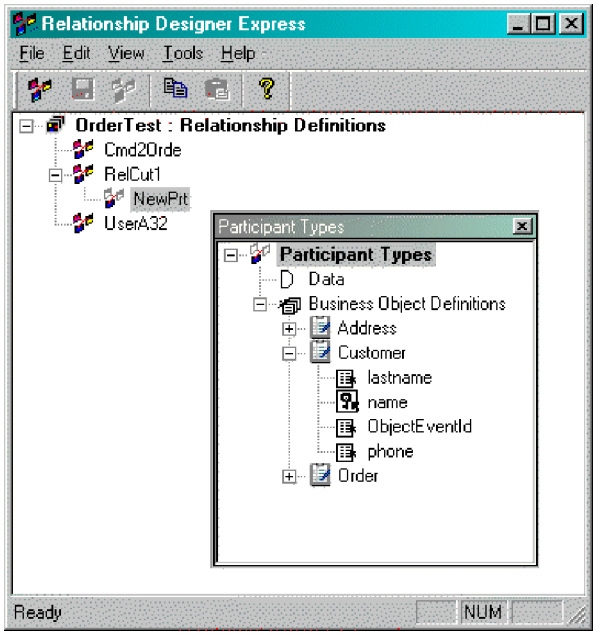

図 110. Relationship Designer Express のメインウィンドウ

メインウィンドウのカスタマイズ

Relationship Designer Express では、次の方法でメイン・ウィンドウをカスタマイズできます。

- 263 ページの『表示するウィンドウの選択』
- 『連結可能なウィンドウの浮動化』

表示するウィンドウの選択

はじめて Relationship Designer Express を開いたとき、メインウィンドウには関係定義リストのみが表示されます。「参加者タイプ」ウィンドウは表示されません。メインウィンドウの外観は、「表示」プルダウン・メニューからのオプションでカスタマイズできます。表 82 は「表示」メニューのオプション、およびそれらが Relationship Designer Express のメインウィンドウに与える効果について示しています。

表 82. メインウィンドウのカスタマイズ用の「表示」メニューの各オプション

「表示」 メニュー・オプション	表示される要素
参加者タイプ	「参加者タイプ」ウィンドウが表示されます。
ツールバー	標準ツールバーです。Relationship Designer Express の主な機能を提供します。
ステータス・バー	1 行のペイン。Relationship Designer Express の状況情報が表示されます。

ヒント: 各メニュー・オプションの左横にチェック・マークが付いているとき、対応する要素が画面に表示されます。要素を非表示にするには、対応するメニュー・オプションを選択します。選択すると、チェック・マークが消え、その要素は表示されないことを示します。逆に、表示されていない要素を表示させるには、対応するメニュー・オプションを選択します。選択すると、表示させる要素の横にチェック・マークが現れます。

連結可能なウィンドウの浮動化

Relationship Designer Express では、メインウィンドウ内の以下の機能を連結可能なウィンドウにすることができます。

- 標準ツールバー
- 「参加者タイプ」ウィンドウ

ヒント: 連結可能なウィンドウは、デフォルトでは通常メインウィンドウの端に沿って配置され、メインウィンドウの一部として移動します。連結可能なウィンドウを浮動させるときは、メインウィンドウから切り離します。これにより、独立したウィンドウとして機能させることができます。連結可能なウィンドウを浮動させるには、マウスの左ボタンを押したままウィンドウの境界をつかみ、メインウィンドウまたはデスクトップ上にドラッグします。

Relationship Designer Express の機能の使用法

Relationship Designer Express の機能には、以下のいずれの方法を使用してもアクセスできます。

- プルダウン・メニュー

- コンテキスト・メニュー
- ツールバーのボタン
- キーボードのショートカット

Relationship Designer Express のプルダウン・メニュー

Relationship Designer Express には、以下のプルダウン・メニューがあります。

- 「ファイル」メニュー
- 「編集」メニュー
- 「表示」メニュー
- 「ツール」メニュー
- 「ヘルプ」メニュー

以下のセクションでは、これらの各メニューのオプションについて説明します。これらのオプションについては、示されたキーボード・ショートカットが使用可能です。

「ファイル」メニューの機能: Relationship Designer Express の「ファイル」プルダウン・メニューには、表 83 に示すオプションがあります。「プロジェクトへの切り替え」オプションを除き、「ファイル」メニューのオプションは、すべて現在のプロジェクトのオブジェクトに影響します。

表 83. Relationship Designer Express の「ファイル」メニュー・オプション

「ファイル」メニュー・オプション	説明	詳細情報の参照先
「新規 (Ctrl+N)」	新規関係定義を作成します。	266 ページの『関係定義の作成』
「プロジェクトへの切り替え (Ctrl+S)」	その他のプロジェクトをリストします。	260 ページの『プロジェクトの処理』
「保管」	現在の関係定義をファイルに保管します。	266 ページの『関係定義の作成』
「すべて保管」	開いているすべての関係定義を保管します。	該当なし
「参加者定義を追加」	現在の関係定義に新規参加者定義を追加します。	266 ページの『関係定義の作成』

「編集」メニューの機能: Relationship Designer Express の「編集」プルダウン・メニューには、以下のオプションがあります。

- 「名前変更」: 関係定義を名前変更する。
- 「コピー」(Ctrl+C) : 現在の関係定義をコピーする。
- 「貼り付け」(Ctrl+V) : コピーされた関係定義を貼り付ける。
- 「切り取り」(Ctrl+X) : 現在の関係定義を削除する。
- 「拡張設定」:「拡張設定」ウィンドウを表示する。

「表示」メニューの機能: Relationship Designer Express の「表示」プルダウン・メニューには、次のオプションが表示されます。

- 「参加者タイプ」:「参加者タイプ」ウィンドウが表示される。
- 「ツリーを展開」: 関係定義リストの現在のレベルのメンバーを表示する (レベルの横にある正記号のクリックと同じ結果になる)。

- 「ツリーを縮小」: 関係定義リストの現在のレベルを圧縮し、メンバーが表示されないようにする (レベルの横にある負記号のクリックと同じ結果になる)。
- 「ツールバー」: このオプションを使用可能にした場合、標準ツールバーが表示される。
- 「ステータス・バー」: このオプションを使用可能にした場合、メインウィンドウの最下部に 1 行の状況メッセージが表示されます。

表示を制御する「表示」メニュー・オプションの詳細については、263 ページの『表示するウィンドウの選択』を参照してください。

「ツール」メニューの機能: Relationship Designer Express の「ツール」プルダウン・メニューには、それぞれの WebSphere Business Integration Tools を始動するオプションがあります。

- 「Relationship Manager」
- Process Designer Express

  制約事項: このツールは、WebSphere Business Integration Express Plus でのみ使用可能です。
- Map Designer Express
- Business Object Designer Express

「ヘルプ」メニューの機能: Relationship Designer Express には、次のオプションを持つ標準ヘルプ・メニューがあります。

- 「ヘルプ・トピック」(F1)
- 「ドキュメンテーション」
- Relationship Designer Express に関する

コンテキスト・メニュー: コンテキスト・メニューは、いろいろな場所から右マウス・ボタンをクリックすることによって使用できるショートカット・メニューです。役立つコマンドが入ったメニューが開きます。このメニューはクリックする場所によって異なります。

Relationship Designer Express の標準ツールバー

Relationship Designer Express には、通常のタスクを実行するために必要な標準ツールバーがあります。このツールバーは連結可能で、メインウィンドウのパレットから引き離してメインウィンドウやデスクトップ上に浮動させることができます。

ヒント: それぞれのツールバー・ボタンの目的を確認するには、マウス・カーソルで各ボタンをロールオーバーしてください。

図 111 に、Relationship Designer Express の標準ツールバーを示します。

図 111. Relationship Designer Express の標準ツールバー

以下のリストに、標準ツールバーの各ボタンの機能を (左から右の順に) 示します。

- 「新規関係」

- 「関係を保管」
- 「新規参加者」
- 「コピー」
- 「貼り付け」
- 「ヘルプ」

関係定義の作成

関係定義を作成するには、次の手順を実行します。

1. 次のいずれか 1 つの方法で関係名を作成します。
   - 「ファイル」メニューから「新規関係定義」を選択する。
   - キーボード・ショートカット Ctrl+N を使用します。
   - 標準ツールバーで「新規関係」ボタンをクリックする。
2. 関係定義のアイコンに名前を付けます。

   規則: 関係定義名は最大 8 文字であり、文字と数値のみ指定でき、先頭は文字にする必要があります。
3. 関連付ける各ビジネス・オブジェクトの参加者定義を作成します。

   作成するには、関係定義名を選択し、次のいずれかの操作を実行します。
   - 「ファイル」メニューから「参加者定義を追加」を選択します。
   - 標準ツールバーで「新規参加者」ボタンをクリックする。
4. 参加者定義ごとに、参加者定義のアイコン名を付けます。

   規則: 参加者定義名は最大 8 文字であり、文字と数値のみ指定でき、先頭は文字にする必要があります。
5. 「参加者タイプ」ウィンドウから参加者定義にタイプをドラッグし、各参加者にデータ型を関連付けます。

   ヒント: 「参加者タイプ」ウィンドウを表示するには、「表示」メニューから「参加者タイプ」を選択します。
   - ビジネス・オブジェクトのデータ型を関連付けるには、「参加者タイプ」ウィンドウからビジネス・オブジェクト定義をドラッグする。

     一致関係の参加者は参加者タイプとしてビジネス・オブジェクト定義を使用します。詳細については、267 ページの『一致関係の定義』を参照してください。
   - Java データ型を関連付けるには、「参加者タイプ」ウィンドウから Data 参加者タイプをドラッグする。

     関係定義で、Data 参加者タイプはビジネス・オブジェクト・タイプ以外のすべての データ型を表します。参照関係の参加者は、参加者タイプとして Data を使用します。詳細については、270 ページの『参照関係の定義』を参照してください。
6. ビジネス・オブジェクト定義である参加者タイプの場合、参加者と関連付ける属性を追加または変更します。

選択した属性に基づいてビジネス・オブジェクトは関連付けられます。

7. 次のいずれか 1 つの方法で関係定義を保管します。
   - 「ファイル」メニューから「関係定義の保管」を選択する。
   - キーボード・ショートカット Ctrl+S を使用します。
   - 標準ツールバーで「関係を保管」ボタンをクリックする。
8. 関係定義を使用するマップを実行する前に、次の手順を実行します。
   a. 関係をアクティブにする。新しい関係は、InterChange Server Express に配置した時点ではまだアクティブになっていません。しかし、マッピング API メソッドが関係表にアクセスできるようにするには、関係表をアクティブにする必要があります。関係をアクティブにするには、System Manager で関係名をクリックし、「コンポーネント」メニューから「開始」オプションを選択します。
   b. 作成した関係を使用するマップを、コンパイルして配置する。

      結果: InterChange Server Express でマップの配置とコンパイルが正常に完了すると、実行可能なマップ・コードが作成され、マップがアクティブ化されます。詳細については、91 ページの『マップのコンパイル』を参照してください。

制約事項:

1. IBM では、InterChange Server Express リポジトリーでサポートされるデータベースおよびプラットフォームでのみ、関係表の作成がサポートされます。
2. 関係定義を作成するか変更する場合、はじめに System Manager の「関係」メニューから関係を停止し、関係を変更し、関係を再開始する必要があります。

一致関係の定義

一致関係 は 2 つ以上のビジネス・オブジェクト間の 1 対 1 に基づいた関連を確立します。つまり、特定の関係インスタンスに、各参加者のインスタンスは 1 つしかありません。通常、顧客 ID、製品 ID など、ビジネス・オブジェクトのキー属性を変換するとき、一致関係を作成します。背景情報の詳細については、247 ページの『一致関係』を参照してください。

InterChange Server Express では、表 84 に示す一致関係の種類がサポートされます。

表 84. 一致関係の種類

一致関係タイプ	説明	詳細情報の参照先
単純一致関係	単一のキー属性で 2 つのビジネス・オブジェクトを関連付けます。	287 ページの『単純一致関係の使用』
複合一致関係	複合キー (複数の属性から構成) で 2 つのビジネス・オブジェクトを関連付けます。	300 ページの『複合一致関係の使用』

一致関係の定義手順

Relationship Designer Express を使用して一致関係を定義するには、次の手順を実行します。

1. 266 ページの『関係定義の作成』のステップ 1 から 4 に従って、関係定義と参加者定義を作成します。

   ガイドライン: 関連付ける各ビジネス・オブジェクトの参加者定義を作成します。一致関係では、アプリケーション固有のビジネス・オブジェクトおよび汎用ビジネス・オブジェクトの参加者が必要です。
2. 「参加者タイプ」ウィンドウから参加者定義にビジネス・オブジェクト定義をドラッグして、各参加者定義とビジネス・オブジェクトを関連付けます。Relationship Designer Express のメインウィンドウに正記号 (+) 記号が表示されたら、ドラッグ・ボタンを解放できます。「参加者タイプ」ウィンドウを開く方法については、266 ページの『関係定義の作成』のステップ 5 を参照してください。

   一致関係の場合、参加者タイプはビジネス・オブジェクトです。すべての一致関係は、汎用ビジネス・オブジェクトの参加者タイプを持つ 1 つの参加者、およびアプリケーション固有のビジネス・オブジェクトごとに 1 つの参加者から構成されます。
3. 特定の定義と関連付けるビジネス・オブジェクトごとに、そのビジネス・オブジェクトを他の参加者と関連付ける属性を追加します。

   追加するためには、「参加者タイプ」ウィンドウで関連したビジネス・オブジェクトを展開し、属性を選択し、Relationship Designer Express メインウィンドウのビジネス・オブジェクトにドラッグします。選択した属性が、ビジネス・オブジェクト間の関係の基礎となります。

   一致関係の場合、通常、属性は各ビジネス・オブジェクト定義のキー属性です。キーのタイプにより一致関係の種類が決まります。
   - 単一キーの場合、単純一致関係を使用する。各参加者は、ビジネス・オブジェクトの固有キーという 1 つの属性のみで構成されます。詳細については、299 ページの『子関係定義の作成手順』を参照してください。
   - 複合キーの場合、複合一致関係を使用する。複合キーを指定するには、複合キーに現れる順序で各キー属性を追加します。各参加者は複数の属性を含めることができます。通常、親ビジネス・オブジェクトの固有キー、および子ビジネス・オブジェクト (親ビジネス・オブジェクト内) の属性が最低 1 つ含まれます。サーバーに展開すると、参加者定義に表示される順序で属性を連結した名前で関係が表に保管されます。一部のデータベースにおけるインデックス・サイズの制限などの詳細については、300 ページの『複合一致関係定義の作成』を参照してください。
4. 関係定義名を強調表示にし、「編集」メニューから「拡張設定」を選択します。

   はじめに、「拡張設定」ウィンドウに関係定義設定が表示されます (273 ページの図 113 を参照)。

   a. 次のように関係定義を変更します。
      - 「**関係タイプ**」で「ID」ボックスを選択する。

        結果: この設定は、各参加者の関係インスタンス ID とキー属性に一意性制約を設定して、InterChange Server Express が関係を一致関係として処理

するように指定します。このアクションにより、各関係インスタンスのすべての参加者間で 1 対 1 の対応が保証されます。

- 関係表をデフォルト・データベース (デフォルトでは WebSphere Business Integration システム・リポジトリー) 以外のデータベースに配置する場合、ウィンドウの「DBMS 設定」領域に適切なデータベース情報を入力する。詳細については、273 ページの『関係定義の拡張設定』を参照してください。

b. 参加者定義の拡張設定を変更します。

- 「拡張設定」ウィンドウのオブジェクト・ブラウザーで関係定義を展開し、汎用ビジネス・オブジェクトを表す参加者定義を強調表示にし、参加者定義設定を表示する (274 ページの図 114 を参照)。「IBM WBI-managed」というラベルのボックスを選択します。

  結果: このアクションは、Relationship Designer Express が汎用ビジネス・オブジェクトの関係表を*作成しない* ように指定します。`maintainSimpleIdentityRelationship()` メソッドを使用して関係を保守するとき、WebSphere Business Integration システムはアプリケーション固有の関係表に保管された関係インスタンス ID を使用して、関係属性を変換します。

- この参加者の関係表またはストアード・プロシージャーの名前をカスタマイズする場合は、ウィンドウの該当するフィールドに名前を入力する。詳細については、274 ページの『参加者定義の拡張設定』を参照してください。

c. 「OK」をクリックして「拡張設定」ウィンドウを閉じます。

5. 266 ページの『関係定義の作成』のステップ 7 から 8 の記述に従って、関係定義を保管します。

子ビジネス・オブジェクトの関連付け

一致関係を作成するとき、多くの場合、関連付けるビジネス・オブジェクトは子ビジネス・オブジェクトを備えています。例えば、一部の顧客ビジネス・オブジェクトには、住所情報を保管する子ビジネス・オブジェクトがあります。子オブジェクトは、表 85 が示す種類の関係に参加できます。

表 85. 子ビジネス・オブジェクト用の関係

子ビジネス・オブジェクトの状態	関係の種類	詳細情報の参照先
子ビジネス・オブジェクトのキーは、その親のコンテキストを越えて子を*一意的に* 識別します。	単純一致関係	299 ページの『子レベルの単純一致関係のコーディング』
子ビジネス・オブジェクトのキーは、その親のコンテキストを越えて子を一意的に*識別しません*。	複合一致関係	
更新操作中、子ビジネス・オブジェクトを一致関係の一部として維持します。	親子関係	309 ページの『子インスタンスの管理』

子が複数カーディナリティーの子ビジネス・オブジェクトである場合、インデックスを変更し、参加者に特定の子を参照させることができます。それには、子のキー

属性を選択し、右マウス・ボタンをクリックし、コンテキスト・メニューから「インデックスを変更」を選択します。マップのソースと宛先の子が 1 対 1 に対応する場合、インデックスは重要ではなく、変更する必要はありません。しかし、マップがそれ以外の方法で子を変換する場合、特定のインデックス番号を入力できます。例えば、子ビジネス・オブジェクトが住所を表し、3 番目のソース住所が最初の宛先住所に対応する場合、インデックスをそれぞれ 2 と 0 に変更できます。

参照関係の定義

参照関係 はビジネス・オブジェクト間で等価だが異なる方法で表現されることがあるデータを関連付けます。この場合、1 つのビジネス・オブジェクトの値があると、関係は関係表で別のビジネス・オブジェクトの等価な値を参照できます。参照を必要とすることが多い属性の例として、コード (EmployeeType、PayLevel、OrderStatus) および省略形 (State、Country、Currency) があります。背景情報の詳細については、246 ページの『参照関係』を参照してください。

参照用の関係定義を作成する場合、関連付ける属性を含むビジネス・オブジェクトごとに 1 つの参加者定義を追加します。しかし、実際のビジネス・オブジェクト定義または属性名を参加者定義とは関連付けません。代わりに、各参加者定義の参加者タイプとして Data を指定します。

参照関係の定義手順

Relationship Designer Express を使用して参照関係を定義するには、次の手順に従います。

1. 266 ページの『関係定義の作成』のステップ 1 から 4 に従って、関係定義と参加者定義を作成します。

   ヒント: 関連付ける各ビジネス・オブジェクトの参加者定義を作成します。

2. 「参加者タイプ」ウィンドウから参加者定義に Data 参加者タイプをドラッグして、参加者定義ごとに参加者タイプとして Data を指定します。

   関係定義で、Data 参加者タイプはビジネス・オブジェクト・タイプ以外のすべての データ型を表します。マップを作成し、Relationship、IdentityRelationship、Participant の各クラスのメソッドを使用して関係のインスタンスを扱う場合、String、int、long、float、double、boolean など、サポートされている任意の Java データ型のデータを使用できます。

3. 各参加者定義の参照値を保管する表名をメモにとります。各参加者定義の参照値を表に入力するためには表名が必要です。または、参照値が入った表がすでにある場合、生成された表名を自分の表名に置き換えることができます。

   関係定義で各参加者定義用の表名を取得するか、独自の表名を指定するには、次の操作を行います。

   a. 参加者定義を強調表示にし、「編集」メニューから「拡張設定」を選択します。

      結果: 「拡張設定」ダイアログ・ボックスに、その参加者のストレージ設定が表示されます。これらの設定の詳細については、272 ページの『関係の拡張設定の指定』を参照してください。

b. 参加者のストレージ設定を書き留めるか、表への独自の情報で設定を上書きします。

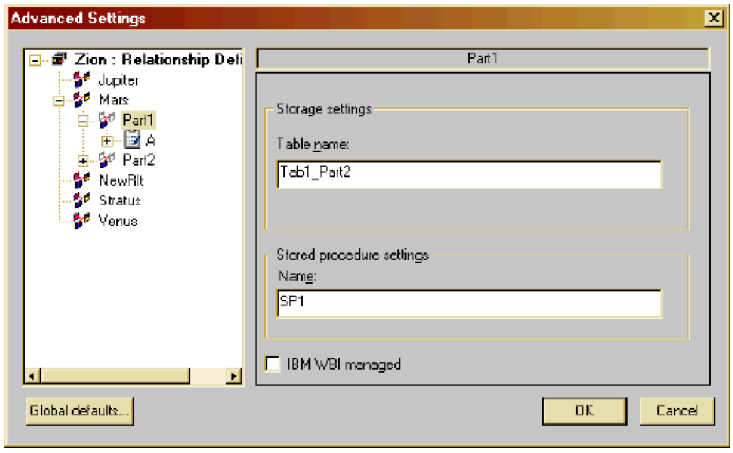

図 112. 「拡張設定」ダイアログ

c. それぞれの参加者定義ごとにステップ 3a とステップ 3b を繰り返します。

d. 「OK」をクリックして「拡張設定」ダイアログ・ボックスを閉じます。

4. 266 ページの『関係定義の作成』のステップ 7 から 8 の記述に従って、関係定義を保管します。

   ヒント: 関係表を作成するには、System Manager の「Deploy Project」ダイアログの「スキーマを作成」ボックスを選択します。ランタイム・スキーマの作成時期の詳細については、271 ページの『関係表スキーマの作成』を参照してください。

5. ステップ 3 で取得した情報を使用し、関係表に各参加者の参照値を入力するか、参照値の独自の表をデータベースに追加します。詳細については、283 ページの『参照表にデータを設定』を参照してください。

関係表スキーマの作成

作成した関係定義ごとに、InterChange Server Express は次のデータベース・オブジェクトを使用して、関係のインスタンスに対するランタイム・データを保守します。

- リレーションシップ・データベース表は関係インスタンスのデータを保持する。
- リレーションシップ・データベースのストアード・プロシージャーは関係表を保守する。

関係定義と参加者定義のコピー

既存の関係定義に似た新しい関係定義を作成するには、既存の定義をコピーし、必要に応じて変更します。また、関係定義から参加者定義をコピーし、同じ関係定義か別の関係定義に貼り付けることもできます。

現在のプロジェクトでの関係定義のコピー手順

関係定義をコピーするには、次の手順を実行します。

1. コピーする関係定義 (CustToClient など) を選択し、「ファイル」メニューから「関係定義の保管」を選択します。
2. コピーする関係定義を選択し、「編集」メニューから「コピー」を選択します。
3. プロジェクト名 (ルート・ツリー・ノード) を選択し、「編集」メニューから「貼り付け」を選択します。

   結果: Relationship Designer Express によって `Copy of CustToClient` という名前の新規関係定義が作成されます。定義名は編集モードで表示されます。
4. 関係定義の新規名を入力し、Enter を押します。
5. リポジトリーに新規定義を保管するには、「ファイル」メニューから「関係定義の保管」を選択します (または `Ctrl+S` のキーボード・ショートカットを使用します)。

ヒント: InterChange Server Express 間で関係定義をコピーするには、`repos_copy` コマンドを使用します。`repos_copy` コマンドでは InterChange Server Express リポジトリーにオブジェクトをコピーしたり、リポジトリーからオブジェクトをコピーしたりできます。

現在のプロジェクトでの参加者定義のコピー手順

参加者定義をコピーするには、次の手順を実行します。

1. コピーする参加者定義が属する関係定義を選択し、「ファイル」メニューから「関係定義の保管」を選択します。
2. コピーする参加者定義を選択し、「編集」メニューから「コピー」を選択します。
3. 参加者定義をコピーする関係定義を選択し、「編集」メニューから「貼り付け」を選択します。

   結果: `Copy` という名前の新規参加者定義が作成されます。定義名は編集モードで表示されます。
4. 参加者定義の新規名を入力し、Enter を押します。

関係定義または参加者定義の名前変更

関係定義または参加者定義は、リポジトリーに保管する前に名前を変更できます。定義の保管後に定義名を変更する場合は、定義を新規名にコピーし、古い名前を削除する必要があります。詳細については、271 ページの『関係定義と参加者定義のコピー』を参照してください。

関係の拡張設定の指定

Relationship Designer Express では、作成した関係定義ごとに、関係インスタンス・データのストレージと処理に影響を与える拡張設定を維持します。

注: 関係表スキーマの作成後にログイン・アカウント名、パスワード、表名など、データベース関連の設定を変更した場合、変更を反映させるには、System Manager を使用して関係表スキーマを再作成する必要があります。

設定の表示または変更を行うには、「編集」メニューから「拡張設定」を選択します。「拡張設定」ダイアログでは、左側で次のいずれを選択したかによって、右側に表示される設定は異なります。

- 関係定義
- 参加者定義
- 属性

関係定義の拡張設定

関係定義の設定を表示または変更するには、関係名を選択します。次の図に、このレベルにおける拡張設定の例を示します。

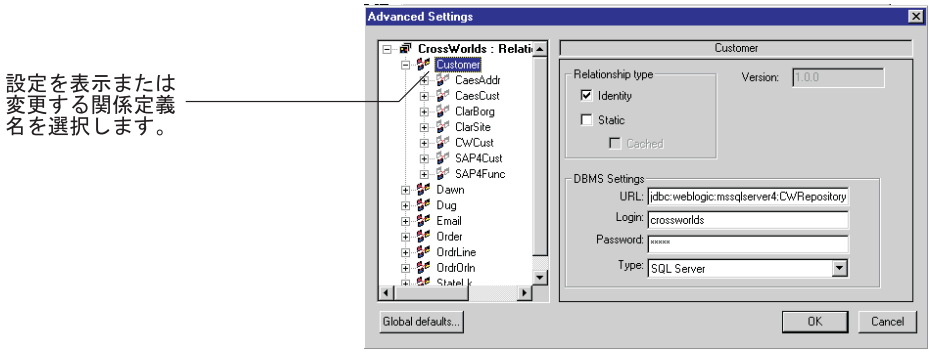

図 113. 関係定義の拡張設定

表 86 では関係定義に使用可能な設定をまとめています。DBMS 設定のデフォルト値は、275 ページの『グローバル・デフォルト設定』で説明している「グローバル・デフォルト設定」ダイアログ・ボックスから取得しています。

表 86. 関係定義の拡張設定の要約

設定	説明
関係タイプ	
ID	このオプションを使用可能にした場合、関係は一致関係です。詳細については、267 ページの『一致関係の定義』を参照してください。
静的	このオプションを使用可能にした場合、関係は静的関係です。詳細については、270 ページの『参照関係の定義』を参照してください。
キャッシュ	「静的」フィールドを使用可能にした場合、このフィールドは使用可能になります。関係表をメモリー内にキャッシュする場合は、このフィールドを選択します。詳細については、277 ページの『関係の最適化』を参照してください。
バージョン	このフィールドは読み取り専用です。関係定義のバージョンはこのリリースではサポートされていません。
DBMS 設定	
URL	この関係定義の関係表が置かれている場所への JDBC パス。すべての関係表のデフォルト位置は「グローバル・デフォルト設定」(275 ページを参照) に指定されています。
ログイン	リレーションシップ・データベースにログインするためのユーザー名。

表 86. 関係定義の拡張設定の要約 (続き)

設定	説明
パスワード	リレーションシップ・データベースにログインするためのパスワード。
タイプ	SQL Server、および DB2 などのリレーションシップ・データベース・タイプ。

注: InterChange Server Express のリポジトリー・データベースとは異なるデータベースを関係表に指定した場合、サーバーが作成できる接続プールの最大数を増加させることが必要になる場合があります。接続プール数を指定するサーバー構成パラメーターは MAX_CONNECTION_POOLS です。デフォルト値は 10 です。

参加者定義の拡張設定

参加者定義の設定を表示または変更するには、参加者定義名を選択します。次の図に、このレベルにおける拡張設定の例を示します。

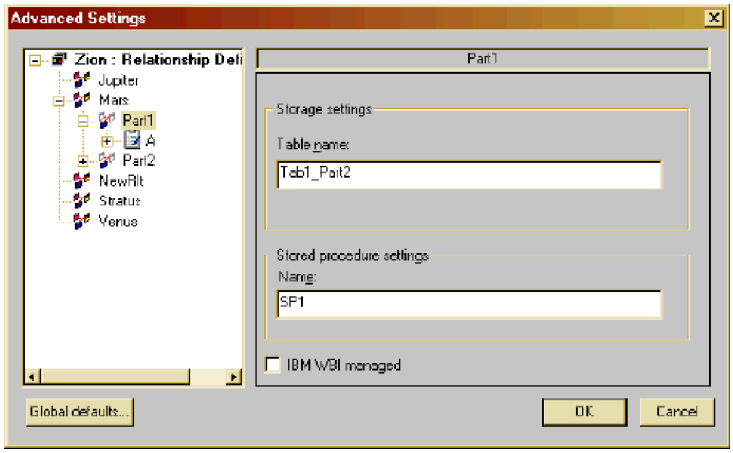

図 114. 参加者定義の拡張設定

表 87 は参加者定義に使用可能な設定をまとめています。

表 87. 参加者定義の拡張設定の要約

設定	説明
テーブル名	この参加者インスタンスの関係データが入っているリレーションシップ・データベースの関係表名。 **規則:** DB2 リレーションシップ・データベースでは、関係表名に使用できる文字数は最大で 18 文字です。 DB2 で表名に制限はありませんが、インデックス名に制限があります。Relationship Designer Express は表名に基づいて関係表のインデックス名を生成するため、DB2 データベースの関係表名は 18 文字以下にする必要があります。
ストアード・プロシージャー名	関係表を保守するストアード・プロシージャー名。

表 87. 参加者定義の拡張設定の要約 (続き)

設定	説明
IBM WBI で管理	選択した場合、この参加者には関係表が作成されません。次の場合にのみ、この設定値を選択してください。 • この参加者定義に関連付けられたビジネス・オブジェクトが汎用ビジネス・オブジェクトである。 • 参加者に 1 つの属性のみ関連付けられており、その属性がキー属性である。

属性の拡張設定

属性の設定を表示または変更するには、属性を選択します。次の図に、拡張設定の例を示します。

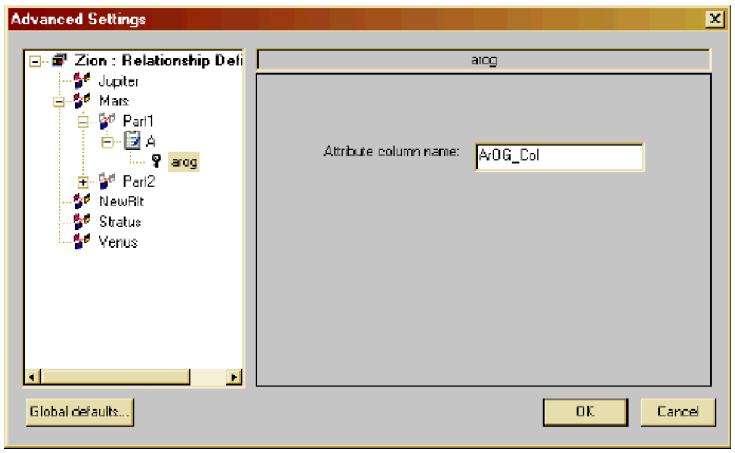

図 115. 属性の拡張設定

属性の場合、属性列名のみ設定できます。列名は、選択された属性の値が入った関係表の列の名前です。通常、属性名と同じになります。Relationship Designer Express が作成するデフォルトでの表の代わりにユーザーが作成した表を使用する場合は、列名の変更が必要になることがあります。

グローバル・デフォルト設定

新規関係定義を保管し、関係表スキーマを作成する場合、関係表のデータベースの位置、データベースのタイプ、および有効なユーザー名とパスワードでデータベースにアクセスする方法を Relationship Designer Express に指定する必要があります。 Relationship Designer Express はこれらの設定のデフォルト値を維持しており、作成するすべての新規関係定義に使用します。関係定義を作成すると、関係定義と共にこれらの設定が保管されます。各関係定義の設定は個々に変更できます。

デフォルトでは、データベース名とアクセス情報は、InterChange Server Express リポジトリーで使用されるものと同じになります。関係表を別の位置に保管する場合、グローバル設定を変更できます。

グローバル・デフォルト設定の表示または変更手順

グローバル・デフォルト設定を表示または変更するには、次の手順を実行します。

1. Relationship Designer Express で、「編集」メニューから「拡張設定」を選択します。

   結果: 「拡張設定」ダイアログ・ボックスが表示されます。

2. 「グローバル・デフォルト」ボタンをクリックします。

   結果: 「グローバル・デフォルト設定」ダイアログ・ボックスが表示されます。

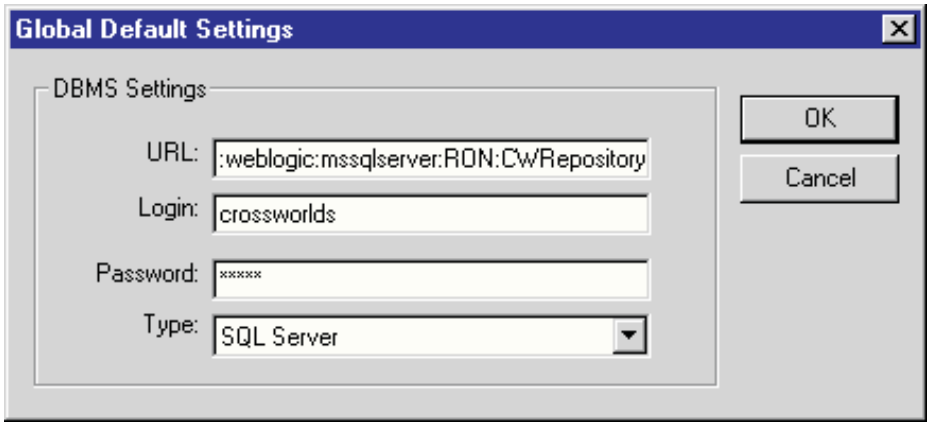

図 116. 「グローバル・デフォルト設定」ダイアログ

表 88 に、関係のグローバル・デフォルト設定を示します。

表 88. 関係のグローバル・デフォルト設定

設定	説明
URL	リレーションシップ・データベースが置かれている場所への JDBC パス。デフォルトは InterChange Server Express のリポジトリー・データベースです。
ログイン	リレーションシップ・データベースにログインするためのユーザー名。
パスワード	リレーションシップ・データベースにログインするためのパスワード。
タイプ	SQL Server、および DB2 などのリレーションシップ・データベース・タイプ。

注: InterChange Server Express のリポジトリー・データベースとは異なるデータベースを関係表に指定した場合、サーバーが作成できる接続プールの最大数を増加させることが必要になる場合があります。接続プール数を指定するサーバー構成パラメーターは `MAX_CONNECTION_POOLS` です。デフォルト値は 10 です。

3. 表示または変更が終了したら、「OK」をクリックします。保管せずに終了する場合は「キャンセル」をクリックします。

注: グローバル・デフォルト設定に行った変更は新規関係定義にのみ適用されます。既存の関係には影響を与えません。既存の関係の設定を変更する場合は、272 ページの『関係の拡張設定の指定』を参照してください。

関係定義の削除

Relationship Designer Express のメイン・ウィンドウにリストされた関係定義は、次のいずれかの方法で削除できます。

- 定義を強調表示して、「編集」メニューから「削除」を選択する。
- 定義を右マウス・ボタンでクリックして、「削除」を選択する。

関係の最適化

デフォルトでは、各関係の関係表はリレーションシップ・データベースに保管されます。関係は、ランタイム・データを検索するか変更するごとに、SQL ステートメントを使用してこのデータベースにアクセスします。関係表へのアクセスが頻繁に行われる場合、CPU 使用と InterChange Server Express リソースの点でパフォーマンスに大きな影響を与えることがあります。関係の設計の一部として、これらの関係表をメモリーにキャッシュするかどうかを決定できます。

この決定を行うには、関係のランタイム・データの変更頻度を調べる必要があります。WebSphere Business Integration システムでは、関係を次のいずれかに分類できます。

- 動的関係: ランタイム・データが頻繁に変わる関係。つまり、関係表には挿入、更新、または削除の操作が頻繁に行われる。デフォルトで、すべての関係は動的である。
- 静的関係: ランタイム・データはほとんど変わらない。つまり、関係表には挿入、更新、または削除の操作がほとんど行われない。例えば、参照表はコード、状況値などの情報を保管するため、多くの場合、データは静的になる。そのような表はメモリーへのキャッシュ対象として適している。

注: WebSphere Business Integration の System Manager は関係をこれら 2 つの同じカテゴリーに分類しています。System Manager で関係フォルダーを展開すると、動的と静的の 2 つのサブフォルダーが表示されます。

関係定義の「拡張設定」ダイアログから、関係が動的か静的かを定義します。以下のセクションでは、このダイアログから動的関係と静的関係を定義する方法について説明します。「拡張設定」ダイアログを表示する方法については、272 ページの『関係の拡張設定の指定』を参照してください。

動的関係の定義

動的関係の場合、InterChange Server Express はリレーションシップ・データベースの関係表からのランタイム・データにアクセスします。デフォルトでは、InterChange Server Express は関係が動的であることを想定しています。したがって、動的関係を定義するために特別な手順を実行する必要はありません。

- 一致関係の場合、267 ページの『一致関係の定義』で説明しているように、「拡張設定」ダイアログの「ID」をクリックする。
- 参照関係の場合、270 ページの『参照関係の定義』で説明しているように、「ID」が*選択されていない* ことを確認する。

注: 動的関係の場合、「拡張設定」ダイアログの「静的」フィールドまたは「キャッシュ」フィールドをクリックしないでください。

System Manager では、関係フォルダーの下にある動的フォルダーにすべての動的関係をリストします。

静的関係の定義

静的関係の場合、InterChange Server Express はキャッシュされた関係表からランタイム・データにアクセスできます。静的関係に対してキャッシングを使用可能にすると、InterChange Server Express は関係表のコピーをメモリーに格納します。関係表をキャッシュに入れるかどうかについては、次の条件を考慮に入れて判断してください。

- InterChange Server Express がメモリー内に関係表をキャッシュした場合、通常はパフォーマンスが向上します。

  この場合、サーバーはランタイム・データを取得するために SQL ステートメントを使用してリレーションシップ・データベースにアクセスする必要がありません。代わりに、メモリーにアクセスしてこのデータを取得できるため、処理は高速になります。静的関係のランタイム・データが現在メモリーに存在しない場合、InterChange Server Express はデータに最初にアクセスするときにデータベースからメモリーに適切な関係表を読み込みます。その後のアクセスでは、InterChange Server Express はキャッシュ内の表を使用してください。

  しかし、表がメモリーに読み込まれると、InterChange Server Express はデータベースの関係表とキャッシュ内の表との間の整合性を維持する必要があります。更新、挿入、削除の各操作で、InterChange Server Express はデータベース表およびキャッシュ内の表の*両方* を変更する必要があります。このような二重更新はパフォーマンスに影響を与えることがあります。関係の表をキャッシュに入れるかどうかを決定するときは、データの予想存続時間とリフレッシュ速度を考慮に入れてください。
- 関係表をメモリーにキャッシュすると、メモリー使用量は増加します。メモリー使用量は、メモリー内のすべての表の合計サイズにほぼ一致します。

  推奨: 1000 行を超える関係表はキャッシュに入れないでください。

  要確認: InterChange Server Express はメモリー使用量が多すぎるかどうかをチェック*しません*。メモリー使用量がシステムの制限内におさまっていることを確認してください。

静的関係を定義するには、関係定義の「拡張設定」ダイアログ (図 113 を参照) を表示し、次のようにこのダイアログから「静的」フィールドを設定します。

- 一致関係の場合、「ID」フィールドと「静的」フィールドの両方を使用可能にします。「ID」フィールドの使用の詳細については、267 ページの『一致関係の定義』を参照してください。
- 参照関係の場合、「静的」フィールドを使用可能にします (「ID」フィールドは使用可能にしないでください)。

「拡張設定」ダイアログで「静的」フィールドを使用可能にすると、「キャッシュ」フィールドも使用可能になります。「キャッシュ」フィールドを使用すると、InterChange Server Express が関係の表を実際にキャッシュに入れる時期を制御できます。

- 「キャッシュ」を使用可能にすると、InterChange Server Express は静的関係の関係表をキャッシュに入れることができます。関係に関連するすべての 関係表をキャッシュに入れます。
- 「キャッシュ」を使用不可にすると、InterChange Server Express はメモリーに関係表をキャッシュしません。代わりに、それ以降のアクセスではリレーションシップ・データベース表を使用します。

静的と定義された関係のみキャッシングを制御できます。

注: 「拡張設定」ダイアログから関係の静的状態またはキャッシュ状態を変更した後は、変更をプロジェクトに保管するため、関係定義を保管してください。

注: サーバー・コンポーネント管理ビューから、関係プロパティー「キャッシュ」および「再ロード」を変更できるようになりました。これを実行するには、静的関係を右クリックして、コンテキスト・メニューからプロパティーを選択します。

- 「キャッシュ」: 関係の表のキャッシングを制御する。
- 「再ロード」: 関係の表をメモリーに再読み込みするよう、InterChange Server Express に指定する。

第 8 章 関係のインプリメント

関係属性とは、関係を使用して変換する属性です。ソース属性から宛先属性にドラッグして関係属性を変換することはできません。その代わり、カスタム変換を作成して、Activity Editor で関数ブロックを使用して宛先関係属性の変換規則をカスタマイズするか、あるいは `Relationship`、`IdentityRelationship`、および `Participant` クラスのメソッドを使用して宛先関係属性のコードを作成します。

この章では、マップ内でコードを作成して別の種類の関係のインプリメント方法を説明します。タスクは、次のとおりです。

注: また、この章では、関係に関する関係定義をすでに作成済みであることを前提とします。詳細については、259 ページの『第 7 章 関係定義の作成』を参照してください。

- 『関係のインプリメント』
- 282 ページの『参照関係の使用』
- 287 ページの『単純一致関係の使用』
- 300 ページの『複合一致関係の使用』
- 309 ページの『子インスタンスの管理』
- 312 ページの『動詞の設定』
- 318 ページの『外部キー参照の実行』
- 324 ページの『カスタム関係の維持』
- 326 ページの『安全な関係コードの作成』
- 328 ページの『リレーションシップ・データベースの照会の実行』
- 339 ページの『関係のロードとアンロード』

関係のインプリメント

Relationship Designer Express で関係定義を作成したら、マップ内に関係をインプリメントできます。関係定義の作成については、259 ページの『第 7 章 関係定義の作成』を参照してください。

関係をインプリメントするには、マップの宛先オブジェクトに relationship 関数ブロックを使用するか、マップの宛先オブジェクト内の属性のコードにマッピング API メソッドを追加します。

表 89 に、使用する関数ブロックを示します。

表 89. Relationship の関数ブロック

関係の種類	関数ブロック	詳細情報の参照先
参照	General/APIs/Relationship/Retrieve Instances General/APIs/Relationship/Retrieve Participants	282 ページの『参照関係の使用』

表 89. Relationship の関数ブロック (続き)

関係の種類	関数ブロック	詳細情報の参照先
単純 ID	General/APIs/Identity Relationship/Maintain Simple Identity Relationship /General/APIs/Identity Relationship/Maintain Child Verb	287 ページの『単純一致関係の使用』
複合 ID	General/APIs/Identity Relationship/Maintain Composite Relationship General/APIs/Identity Relationship/Maintain Child Verb General/APIs/Identity Relationship/Update My Children (オプション)	300 ページの『複合一致関係の使用』
カスタム	General/APIs/Relationship/Create Relationship General/APIs/Identity Relationship/Add My Children General/APIs/Relationship/Add Participant	

表 89 に、別の種類の関係を維持するマッピング API メソッドを示します。

表 90. 関係のマッピング API メソッド

関係の種類	マッピング API メソッド	詳細情報の参照先
参照	`retrieveInstances()retrieveParticipants()`	『参照関係の使用』
単純 ID	`maintainSimpleIdentityRelationship()` `maintainChildVerb()`	287 ページの『単純一致関係の使用』
複合 ID	`maintainCompositeRelationship()maintainChildVerb()` `updateMyChildren()`(オプション)	300 ページの『複合一致関係の使用』
カスタム	`create()addMyChildren()addParticipant()`	324 ページの『カスタム関係の維持』

関係属性を変換する場合は、マップの呼び出しコンテキストをマップが認識する必要があります。呼び出しコンテキストを判別するには、マップの実行コンテキストの次の情報がマップに必要です。

- マップの実行コンテキストの一部である、マップの呼び出しコンテキスト。

  詳細については、208 ページの『呼び出しコンテキスト』を参照してください。
- ビジネス・オブジェクトの一部である、動詞。

これらの 2 つの要因により、関係表に行う必要のあるアクションをマップが認識します。

表 89 の関係の場合、関連するマッピング API メソッドにより関係表に該当する操作が実行されます。したがって、これらのメソッドでは、呼び出しコンテキストとビジネス・オブジェクト動詞が引き数として渡される必要があります。

参照関係の使用

参照関係 はビジネス・オブジェクト間で等価だが異なる方法で表現されることがあるデータを関連付けます。次のセクションでは、参照関係を使用する手順について説明します。

- 283 ページの『参照関係定義の作成』

- 『参照表にデータを設定』
- 285 ページの『参照関係のマップ変換のカスタマイズ』

注: 背景情報については、246 ページの『参照関係』を参照してください。

参照関係定義の作成

参照関係定義は、参加者タイプがビジネス・オブジェクトでなく、data タイプであることが一致関係定義と異なります (参加者タイプ・リストでの最初の選択)。参照関係の関係定義の作成方法については、270 ページの『参照関係の定義』を参照してください。

例: AddressType 値について StatAdtp と呼ばれる参照関係を作成するとします。図 117 では、各ボックスは StatAdtp 参照関係の参加者を表します。この関係の各参加者のタイプは、data タイプです。

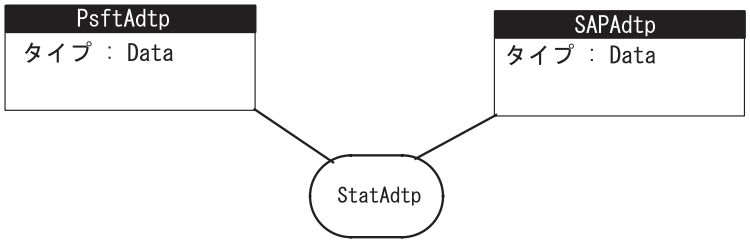

図 117. StatAdtp 参照関係定義

参照関係では関連する属性は示さないので、複数の属性の変換に 1 つの関係定義を使用することができます。実際に、変換するビジネス・オブジェクトに関係なく、参照が必要なすべての属性に関して 1 つの参照関係定義を使用できます。しかし、各関係定義には 1 つの表セットしか作成されないので、すべての参照関係に 1 つの関係定義を使用すると、表が大きくなり維持が難しくなります。

適切な方法は、国別コードや状況などの共通のデータ単位について 1 つの参照関係定義を作成することです。この方法では、関係表セットのそれぞれに意味が関係する情報が含まれます。このように定義された関係はまたモジュール方式でもあります。つまり、新規のコラボレーションやアプリケーションをサポートする場合に新規の参加者を追加し、同じ関係定義を再使用できるからです。例えば、国別コードに関して参照関係定義を作成し、Clarify_Site ビジネス・オブジェクトを SAP_Customer に変換するとします。後で、新規コラボレーションまたは新規アプリケーションを追加する場合、国別コードに関係する各変換の同じ関係定義を再使用することができます。

参照表にデータを設定

「スキーマを作成」オプションを使用可能にして参照関係定義を配置すると、Interchange Server Express によって関係表 (*参照表* とも呼ばれる) が参加者ごとに生成されます。各参照表には、次のような形式の名前が付いています。

RelationshipDefName _ParticipantDefName

「スキーマを作成」オプションを使用可能にして StatAdtp 関係定義を配置すると (図 117 を参照)、Interchange Server Express によって次の 2 つの参照表が生成されます。

- StatAdtp_PsftAdtp_T
- StatAdtp_SAPAdtp_T

参照表には、関係インスタンス ID の列 (INSTANCEID) と関連する参加者インスタンス・データ (data) が含まれます。図 118 に、StatAdtp 参照関係の PsftAdtp および SAPAdtp 参加者に関する参照表を示します。これらの 2 つの参照表は関係インスタンス ID を使用して参加者を相互に関連付けます。例えば、116 のインスタンス ID は、PsftAdtp の値 Fired と SAPAdtp の値 04 と相互に関係します。

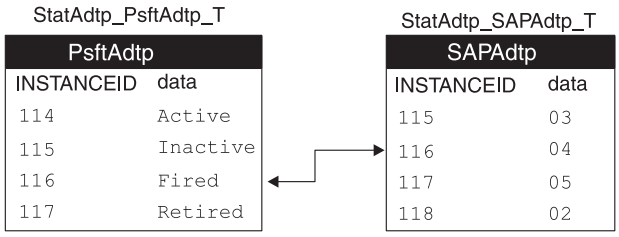

図 118. CustLkUp 参照関係の関係表

一致関係のデータを保持する関係表と異なり、参照表は自動的に設定されません。表の列にデータを挿入してこれらの表を設定する必要があります。次のいずれの方法でも、参照表にデータを設定できます。

- SQL INSERT ステートメントを含むスクリプトを作成して、参照表に必要なデータを設定します。
- Relationship Manager を使用して参照表に行を追加します。

SQL を使用した参加者インスタンスの挿入

SQL の INSERT ステートメントを使用して、参照表に参加者データを挿入することができます。この方法は、参照表に多くの行データを追加する必要がある場合に便利です。1 つの INSERT ステートメントについての構文を作成し、その後エディターを使用して、行を挿入する回数だけこの行をコピーおよび貼り付けします。各行では、挿入されるデータ (通常は、INSERT ステートメントの VALUES 文節) の編集のみが必要です。

INSERT ステートメントを使用するには、参照関係表とその列の名前を知る必要があります。表 91 に、参照表の列名を示します。

表 91. 参照表の列

参照表の列	説明
INSTANCEID	関係インスタンス ID。
data	参加者データ。
STATUS	参加者がアクティブの場合、ゼロ (0) に設定されます。
LOGICAL_STATE	参加者インスタンスが論理的に削除されたかどうかを示します (ゼロは「削除されていない」を示します)。
TSTAMP	参加者インスタンスの最終変更日。

重要: SQL ステートメントを使用して参加者データを参照表に挿入する場合は、STATUS、LOGICAL_STATE、および TSTAMP 列の値を必ず指定してください。IBM WebSphere Business Integration ツールが正しく動作するには、すべての値が必要です。特に、TSTAMP 値が不足していると Relationship Manager が参加者データを検索できなくなります。タイム・スタンプ値が存在しないと、Relationship Manager により例外が発行されます。

例: アドレス・タイプの情報を持つ関係表に参加者データを追加する必要があるとします (表 92 を参照)。

表 92. PsftAdtp 参加者のアドレス・タイプのサンプル値

INSTANCEID	STATUS	LOGICAL_STATE	TSTAMP	データ
1	0	0	現在日付	Home
2	0	0	現在日付	Mailing

次の INSERT ステートメントは、PstfAdtp 参照表に表 92 の参加者データを作成します。

```
INSERT INTO StatAdtp_PsftAdtp_T

   (INSTANCEID, STATUS, LOGICAL_STATE, TSTAMP, data)
   VALUES (1, 0, 0, getDate(), 'Home')

INSERT INTO StatAdtp_PsftAdtp_T
   (INSTANCEID, STATUS, LOGICAL_STATE, TSTAMP, data)
   VALUES (2, 0, 0, getDate(), 'Mailing')
```

注: 先行する INSERT 構文は MicroSoft SQL Server 7.0 と互換性があります。関係表に別のデータベース・サーバーを使用している場合は、そのサーバーと互換性のある INSERT 構文を使用してください。

Relationship Manager を使用した参加者インスタンスの挿入

Relationship Manager は IBM WebSphere Business Integration ツールで、関係表内のランタイム・データをグラフィックで表示します。Relationship Manager は、参照表に数行を追加するのみの場合などに便利です。

参照関係のマップ変換のカスタマイズ

参照関係に関連する関係定義と参加者定義の作成が完了すると、参照を実行できるようにマップの変換規則をカスタマイズすることができるようになります。Activity Editor での参照関係のカスタマイズについては、171 ページの『例 3: 変換での Static Lookup の使用』を参照してください。

表 93 に、参照関係のインプリメントに必要なマッピング API メソッドを示します。この表では、API 呼び出しが必要なマップもリストします。

表 93. 参照関係のマッピング API メソッド

参照関係でのステップ	マップ	マッピング API メソッド
ソース・ビジネス・オブジェクトからの参加者データの関係インスタンス ID の取得。インスタンス ID は汎用ビジネス・オブジェクトに保管されます。	インバウンド・マップ	retrieveInstances()
汎用ビジネス・オブジェクトからの関係インスタンス ID の参加者インスタンスの取得。参加者データはアプリケーション固有のビジネス・オブジェクトに保管されます。	アウトバウンド・マップ	retrieveParticipants()

ヒント: retrieveInstances() と retrieveParticipants() メソッドは、参照表にデータを設定しません。参加者データはすでに表に存在していると想定します。参照関係を含むマップを実行する前に、参照表にデータが設定されていることを確認してください。詳細については、283 ページの『参照表にデータを設定』を参照してください。

インバウンド・マップのコーディング

インバウンド・マップ内の retrieveInstances() メソッドを呼び出して、ソース・ビジネス・オブジェクト内の参加者データの関係インスタンス ID を検索できます。次のコード例は、インバウンド・マップ内の参照を実行します。

```
String addrtype = ObjSrcObj.getString("SrcAttr");
int[] generic_ids =
Relationship.retrieveInstances(
   "RelationshipDefName", "ParticipantDefName", dataFromSrcObj);
if (generic_ids != null && generic_ids.length > 0)
   {
   ObjDestObj.setWithCreate("DestAttr", generic_ids[0]);
   }

else
   {
   throw new MapFailureException(
   logError("No generic instance ID for lookup found");
      "No generic instance ID for lookup found");
   }
```

ヒント: retrieveInstances() メソッドをコーディングする場合は、以下のヒントに留意してください。

- retrieveInstances() メソッドは、特定の参加者データに一致するインスタンス ID を検出しなくても、例外を発生しません。一致するインスタンス ID が検出されない場合に例外を発生するには、それに先行するコードで、戻されたインスタンス ID (generic_ids) が null 値かどうかを検査した後で、宛先ビジネス・オブジェクトに setWithCreate() を設定します。
- retrieveInstances() メソッドは、関係インスタンス ID の*配列* を戻します。通常、参照関係は構造化されているので、指定された各参加者データは 1 つのインスタンス ID とのみ関連付けされます。しかし、retrieveInstances() は 1 対 1 の対応を前提としません。配列を戻すので、複数の関係インスタンス ID を戻すことができます。

アウトバウンド・マップのコーディング

アウトバウンド・マップ内の retrieveParticipants() メソッドを呼び出して、ソース・ビジネス・オブジェクト内の参加者データの関係インスタンス ID を検索できます。次のコード例は、アウトバウンド・マップ内の参照を実行します。

```
int addrtype = ObjSrcObj.getInt("SrcAttr");

if (addrtype != null)
   {
   Participant[] psft_part = Relationship.retrieveParticipants(
      "RelationshipDefName", "ParticipantDefName", addrtype);
   if (psft_part != null && psft_part.length > 0)
      ObjDestObj.setWithCreate("DestAttr", psft_part[0].getString());
   }
```

ヒント: retrieveParticipants() メソッドをコーディングする場合は、以下のヒントに留意してください。

- ユーザーのアウトバウンド・マップで汎用ビジネス・オブジェクトに関係インスタンス ID が含まれることを想定できない場合は、null 値のインスタンス ID を検査した後で retrieveParticipants() を呼び出す必要があることがあります。retrieveParticipants() メソッドは、null 値のインスタンス ID を受け取ると RelationshipRuntimeException 例外を発生します。
- retrieveParticipants() メソッドは、関係インスタンス ID の*配列* を戻します。通常、参照関係は構造化されているので、各関係インスタンス ID は 1 つの参加者データとのみ関連付けされます。しかし、retrieveParticipants() は 1 対 1 の対応を前提としません。配列を戻すので、複数の参加者インスタンスを戻すことができます。

単純一致関係の使用

一致関係は、ビジネス・オブジェクトまたは他のデータ間に *1 対 1* の関連を設定します。単純一致関係では、単一キー属性で 2 つのビジネス・オブジェクトを関連付けします。次のセクションで、単純一致関係を使用する手順を説明します。

- 『単純一致関係定義の作成』
- 288 ページの『一致関係表のアクセス』
- 299 ページの『単純一致関係の変換規則の定義』

単純一致関係定義の作成

一致関係定義は、参加者タイプがビジネス・オブジェクトであり data タイプでない ということが、参照関係定義と異なっています (参加者タイプ・リストでの最初の選択)。単純一致関係の場合、関係は汎用ビジネス・オブジェクトと少なくとも 1 つのアプリケーション固有のビジネス・オブジェクトで構成されます。単純一致関係の参加者タイプは、すべて の参加者に関して 1 つのビジネス・オブジェクトです。すべての参加者に関する参加者属性は、ビジネス・オブジェクトの単一キー属性です (単純一致関係の関係定義の詳細については、267 ページの『一致関係の定義』を参照)。

一致関係表のアクセス

単純一致関係を参照する場合は、アプリケーション固有のビジネス・オブジェクトと汎用ビジネス・オブジェクトの間に、「相互参照」変換規則を定義します。詳細については、53 ページの『一致関係の相互参照』を参照してください。

例: CustIden 関係 (図 99 を参照) により Clarify 顧客内の SiteID キー属性が SAP 顧客内の RefID キー属性に変換されます。これには次のオブジェクト間のマップが組み込まれます。

- インバウンド・マップ: Clarify_Site から Customer

  ClarCust 関係表から、SiteID キー値に関連する関係インスタンス ID を取得します。
- アウトバウンド・マップ: Customer から SAP_Customer

  SAPCust 関係表から、関係インスタンス ID と関連する RefID キー値を取得します。

図 119 に、CustIden 関係表を使用して SiteID 値の A100 を RefID 値の 806 に変換する方法を示します。

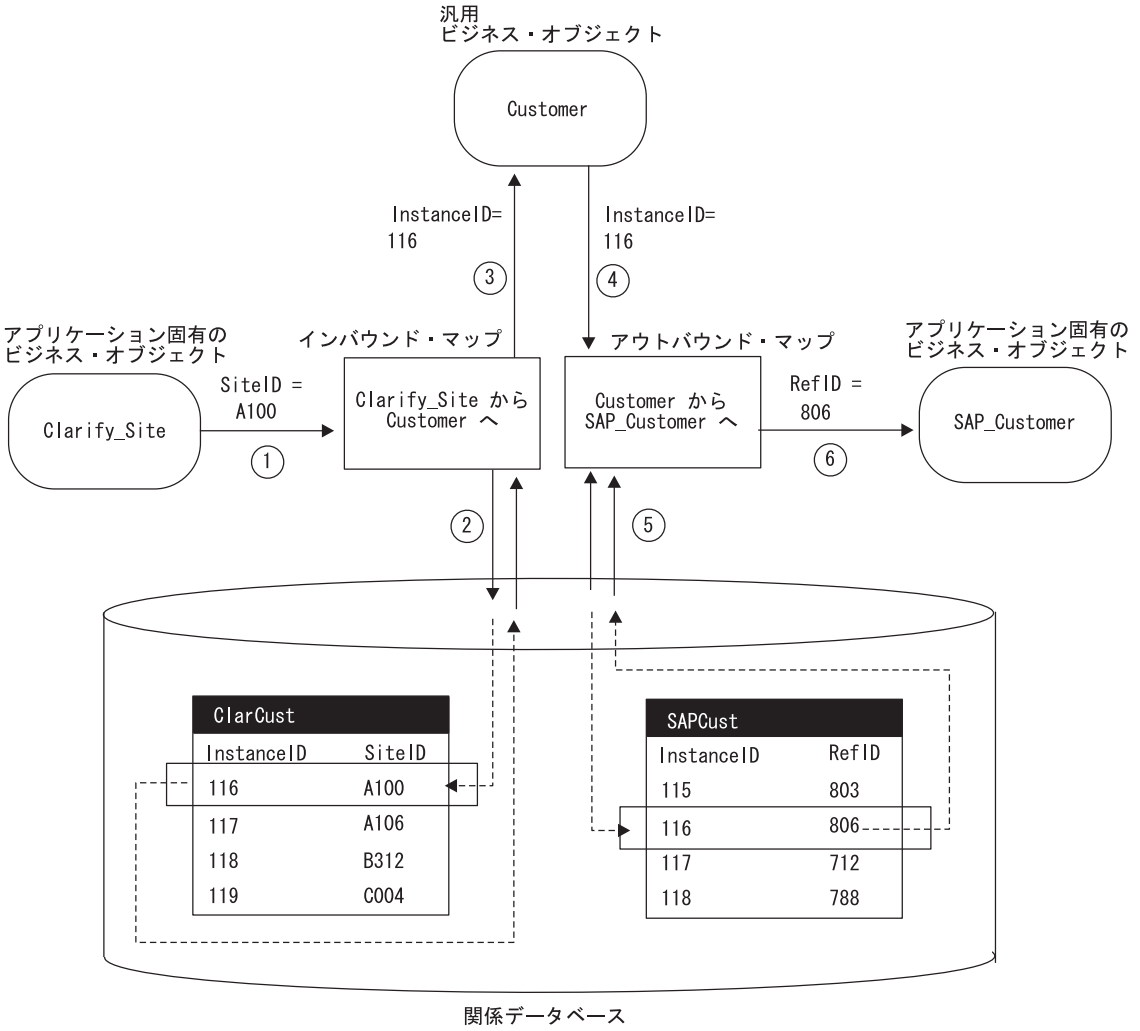

図 119. 関係表を使用した SiteID から RefID への変換

maintainSimpleIdentityRelationship() メソッドは関係表を管理して、関連するアプリケーション固有のキーが単一管理インスタンス ID と関連付けされるようにする必要があります。大まかに説明すると、「相互参照」変換規則を定義した場合には、以下の処理を行うコードが生成されます。

1. 渡される引き数の検証を実行します。引き数が無効の場合、メソッドから RelationshipRuntimeException 例外がスローされます。「相互参照」変換のために生成された Java コードによって実行される検証のリストについては、475 ページの『第 20 章 IdentityRelationship クラス』の maintainSimpleIdentityRelationship() API の説明を参照してください。
2. 次の要素を含む呼び出しコンテキストに基づいて、適切なアクションを行い、関係表を保守します。
   - ビジネス・オブジェクトの動詞

     「相互参照」変換では、ソース・ビジネス・オブジェクトからビジネス・オブジェクト動詞が取得されます。インバウンド・マップの場合、ソースはアプリケーション固有のビジネス・オブジェクトであり、アウトバウンド・マップの場合、ソースは汎用ビジネス・オブジェクトです。

- 呼び出しコンテキストの値

  「相互参照」変換規則を定義した場合、マップの実行コンテキストから呼び出しコンテキストが取得されます。

この変換は、表 94 に示す呼び出しコンテキストを処理します。

表 94. maintainSimpleIdentityRelationship() の呼び出しコンテキスト

呼び出しコンテキスト	説明
EVENT_DELIVERY	コネクターによりアプリケーションから InterChange Server Express にイベントが送信されました (イベントにより起動されたフロー)。
ACCESS_REQUEST	アクセス・クライアントにより外部アプリケーションから InterChange Server Express にアクセス要求が送信されました。
SERVICE_CALL_REQUEST	コラボレーションにより、ビジネス・オブジェクトがサービス呼び出し要求を経由してアプリケーションに送信中です。
SERVICE_CALL_RESPONSE	コラボレーションのサービス呼び出し要求の正常な応答結果として、アプリケーションからビジネス・オブジェクトが受信されました。
SERVICE_CALL_FAILURE	コラボレーションのサービス呼び出し要求は失敗しました。したがって、訂正アクションを実行する必要がある場合があります。
ACCESS_RESPONSE	ソース・ビジネス・オブジェクトは、サブスクリプション送達要求に応答してソース・アクセス・クライアントに送り返されました。

この表 94 に示した各呼び出しコンテキストでの「相互参照」変換の振る舞いについては、次以降のセクションで説明します。

EVENT_DELIVERY および ACCESS_REQUEST 呼び出しコンテキスト

呼び出しコンテキストが EVENT_DELIVERY または ACCESS_REQUEST の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換されます。コネクターにより EVENT_DELIVERY 呼び出しコンテキストが送信され、アクセス・クライアントにより ACCESS_REQUEST 呼び出しコンテキストが送信されます。いずれの場合でも、インバウンド・マップはアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。したがって、「相互参照」変換のタスクは、特定のアプリケーション固有のキー値に対応する関係インスタンス ID を、関係表から取得することです。

呼び出しコンテキストが EVENT_DELIVERY および ACCESS_REQUEST の場合、「相互参照」変換のために生成された Java コードは、次のアクションを実行します。

1. 指定されたアプリケーション固有のビジネス・オブジェクトのキー値と一致する関係表に、関係インスタンスを配置します。表 95 に、「相互参照」変換のために生成された Java コードがアプリケーション固有のビジネス・オブジェクトの動詞に基づいて関係表で実行するアクションを示します。
2. 検索した関係インスタンスからインスタンス ID を取得します。

3. インスタンス ID を汎用ビジネス・オブジェクトにコピーします。

表 95. EVENT_DELIVERY および ACCESS_REQUEST 呼び出しコンテキストのアクション

アプリケーション固有のビジネス・オブジェクトの動詞	maintainSimpleIdentityRelationship() が実行するアクション
Create	アプリケーション固有のビジネス・オブジェクトのキー値について新規エントリーを関係表に挿入します。 このキー値のエントリーがすでに存在する場合は既存のエントリーを検索します。表にさらにエントリーを追加しないでください。
Update	指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。 このキー値のエントリーが存在しない 場合は、表にエントリーを追加します。
Delete	1. 指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。 2. 関係エントリーを「非アクティブ」とマーク付けします。
Retrieve	指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。このキー値のエントリーが存在しない 場合は、RelationshipRuntimeException 例外がスローされます。

AppA アプリケーションに固有のビジネス・オブジェクトから AppB アプリケーションに固有のビジネス・オブジェクトへの変換をサポートする一致関係において、呼び出しコンテキストが EVENT_DELIVERY (または ACCESS_REQUEST) で、AppA アプリケーションに固有のビジネス・オブジェクトの動詞が Create または Update である場合に、「相互参照」変換のために生成された Java コードが参加者 AppA に関連付けられている関係表にどのようにアクセスするかを、図 120 に示します。

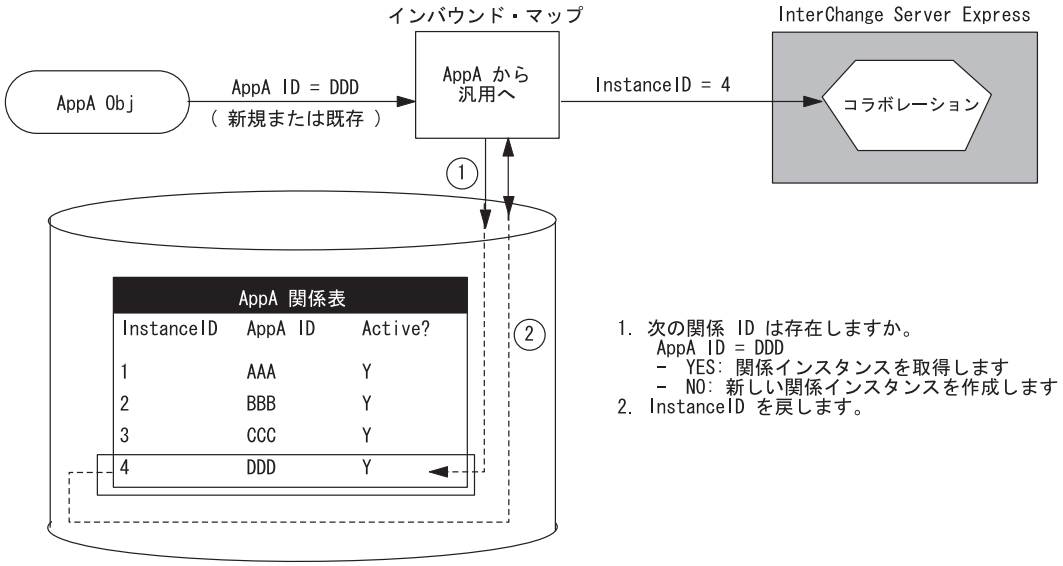

図 120. Create および Update 動詞を使用した EVENT_DELIVERY および ACCESS_REQUEST

呼び出しコンテキストが EVENT_DELIVERY (または ACCESS_REQUEST) で、アプリケーション固有の動詞が Create または Update である場合、AppA アプリケーションに固有のキー値に一致するエントリーが存在しないときには、「相互参照」変換のために生成された Java コードによって、図 121 に示すような書き込みが関係表に対して行われます。

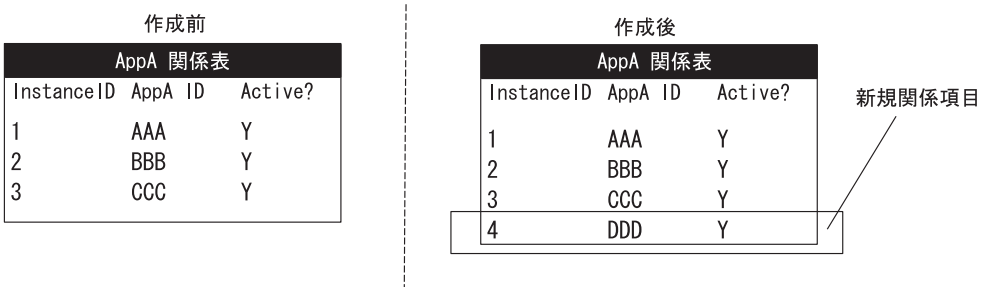

図 121. 新規の関係エントリーに関して関係表に行う書き込み

呼び出しコンテキストが EVENT_DELIVERY (または ACCESS_REQUEST) で、アプリケーション固有の動詞が Delete である場合には、「相互参照」変換のために生成された Java コードによって、図 122 に示すような書き込みが AppA の関係表に対して行われます。

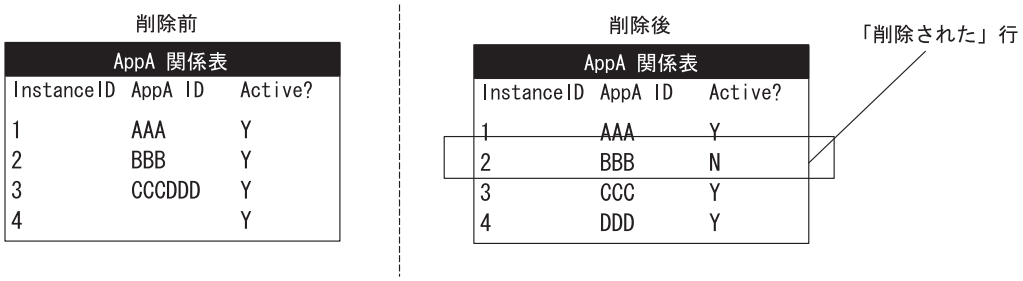

図 122. Delete 動詞に関する関係表への書き込み

SERVICE_CALL_REQUEST 呼び出しコンテキスト

呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、呼び出されるマップはアウトバウンド・マップです。つまり、汎用ビジネス・オブジェクトがアプリケーション固有のビジネス・オブジェクトに変換されます。アウトバウンド・マップは汎用ビジネス・オブジェクトを入力として受け取り、アプリケーション固有のビジネス・オブジェクトを出力として戻します。したがって、「相互参照」変換のタスクは、アプリケーション固有のビジネス・オブジェクトのキー値のうち、特定の関係インスタンス ID に対応するものを、関係表から取得することです (動詞が Update、Delete、Retrieve のいずれかである場合のみ)。「相互参照」変換では、動詞が Create の場合、アプリケーション固有のキー値は取得されません 。

「相互参照」変換に従って、汎用ビジネス・オブジェクトの動詞を基に関係表で実行されるアクションを、表 96 に示します。

表 96. SERVICE_CALL_REQUEST 呼び出しコンテキストのアクション

汎用ビジネス・オブジェクトの動詞	「相互参照」変換により実行されるアクション
Create	アクションは行いません。
Update、Delete、Retrieve	呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、メソッドにより新規エントリーが関係表に実際に書き込まれます。詳細については、294 ページの『SERVICE_CALL_RESPONSE 呼び出しコンテキスト』を参照してください。 1. マップの実行コンテキスト内のオリジナル要求ビジネス・オブジェクトから汎用ビジネス・オブジェクトのキー値 (関係インスタンス ID) を取得します。 2. 指定された汎用ビジネス・オブジェクトのキー値の関係表から、エントリーを検索します。このキー値のエントリーが存在しない 場合は、RelationshipRuntimeException 例外がスローされます。動詞が Retrieve で参加者が検出されない場合、CxMissingIDException 例外がスローされます。 3. 検索した関係エントリーからアプリケーション固有のキー値を取得します。 4. アプリケーション固有のキー値をアプリケーション固有のビジネス・オブジェクトにコピーします。

表 96 からわかるように、動詞が Create の場合、「相互参照」変換のために生成された Java コードは、関係表に新規エントリーを書き込みません。関係インスタンス ID に対応するアプリケーション固有のキー値をまだ持っていないので、この書き込み操作は実行されません。コネクターがアプリケーション固有のビジネス・オブジェクトを処理する場合、新規の行の挿入 (複数行の場合もある) が必要なことがアプリケーションに通知されます。この挿入に成功すると、アプリケーションからコネクターに通知され、コネクターにより Create 動詞とアプリケーションのキー値を使用して該当するアプリケーション固有のビジネス・オブジェクトが作成されます。

動詞がその他の動詞 (Update、Delete、および Retrieve) である場合、「相互参照」変換のために生成された Java コードは、関係表に対して読み取り操作を実行します。AppA アプリケーションに固有のビジネス・オブジェクトから AppB アプリケーション固有のビジネス・オブジェクトへの変換をサポートしている一致関係において、呼び出しコンテキストが SERVICE_CALL_REQUESTで、汎用ビジネス・オブジェクトの動詞が Update、Delete、または Retrieve である場合には、「相互参照」変換の際に、参加者 AppB に関連付けられている関係表へのアクセスが発生します (図 123 を参照)。

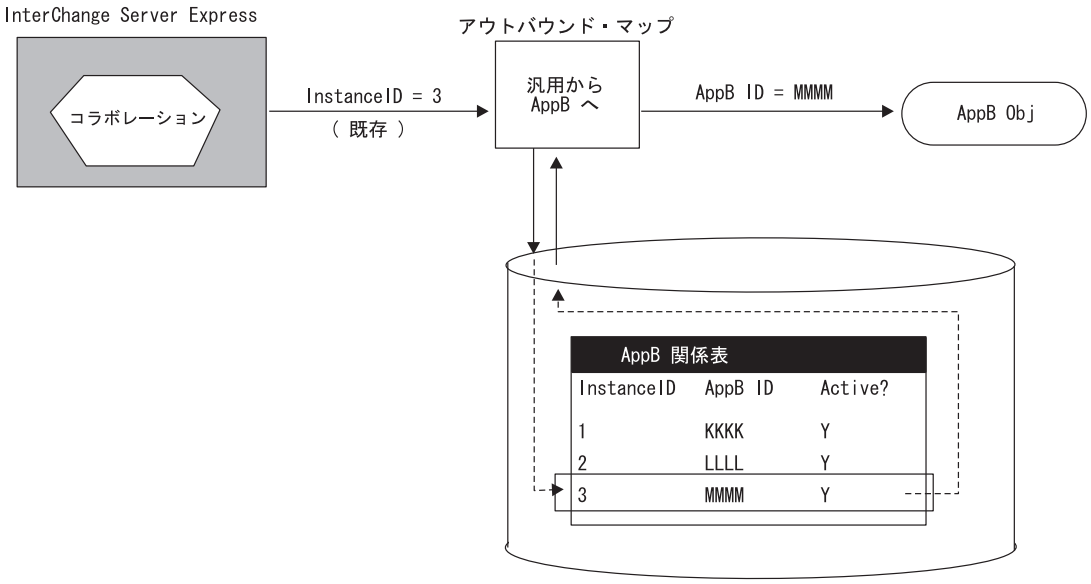

図 123. Update、Delete、または Retrieve 動詞を使用した SERVICE_CALL_REQUEST

SERVICE_CALL_RESPONSE 呼び出しコンテキスト

呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換されます。インバウンド・マップはアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。SERVICE_CALL_RESPONSE 呼び出しコンテキストは、宛先アプリケーションが新規エンティティーの固有値を作成でき、またコネクターがアプリケーション固有のビジネス・オブジェクトを戻すことを示すため、Create 動詞では重要です。

「相互参照」変換規則のタスクは、関係表内のアプリケーション固有のビジネス・オブジェクトのキー値のうち、いずれかの既存の関係インスタンス ID に対応するものを保守することです。呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、「相互参照」変換のために生成された Java コードは、次のアクションを実行します。

1. 汎用ビジネス・オブジェクトが null かどうかを判別します。
   - 動詞が Update、Delete、Retrieve のいずれかである場合、汎用ビジネス・オブジェクトが null であれば、変換時に RelationshipRuntimeException がスローされます。
   - Create 動詞の場合、null 値の汎用ビジネス・オブジェクトは有効です。
2. 指定されたアプリケーション固有のビジネス・オブジェクトのキー値と一致する関係表に、エントリーを配置します。表 97 に、「相互参照」変換のために生成された Java コードがアプリケーション固有の動詞に基づいて関係表で実行するアクションを示します。

表 97. SERVICE_CALL_RESPONSE 呼び出しコンテキストのアクション

アプリケーション固有のビジネス・オブジェクトの動詞	maintainSimpleIdentityRelationship() が実行するアクション
Create	指定されたアプリケーション固有のキーに関して、アプリケーション固有のビジネス・オブジェクトのキー値を含む新規関係エントリーとその関連する関係インスタンス ID が関係表に挿入されます。このメソッドは、マップの実行コンテキスト (cwExecCtx) 内のオリジナル要求ビジネス・オブジェクトから関係インスタンス ID を取得します。 このキー値のエントリーがすでに存在する場合は既存のエントリーを検索します。表にさらにエントリーを追加しないでください。
Delete	1. 指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。 2. 関係エントリーを「非アクティブ」とマーク付けします。
Update	指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。
Retrieve	指定されたアプリケーション固有のビジネス・オブジェクトのキー値の関係表から、関係エントリーを検索します。

AppA アプリケーションに固有のビジネス・オブジェクトから AppB アプリケーションに固有のビジネス・オブジェクトへの変換をサポートする一致関係において、呼び出しコンテキストが SERVICE_CALL_RESPONSE で、AppB アプリケーションに固有のビジネス・オブジェクトの動詞が Create である場合に、「相互参照」変換のために生成された Java コードが参加者 AppB に関連付けられている関連表にどのようにアクセスするかを、図 124 に示します。

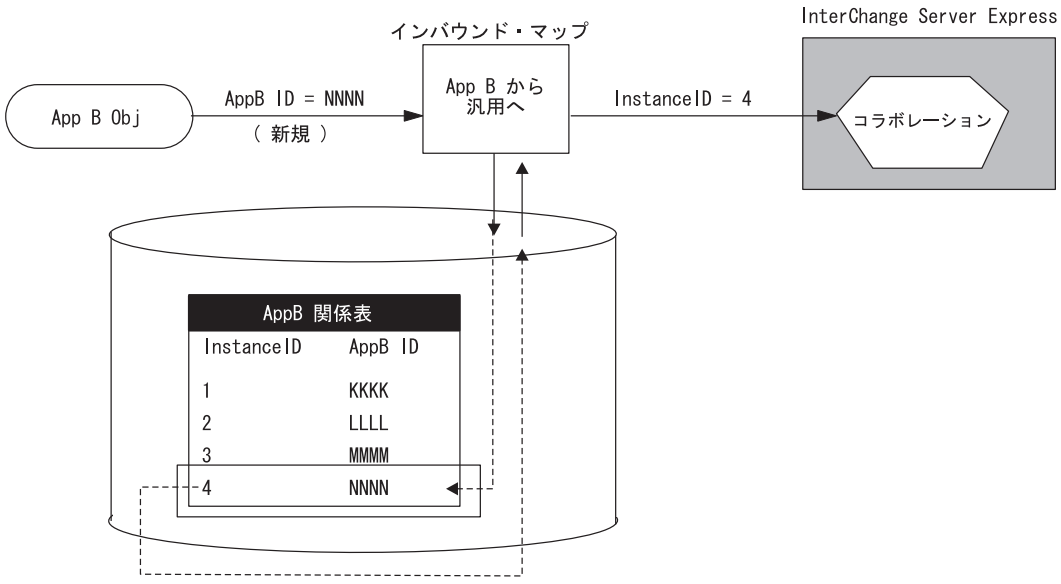

図 124. Create 動詞を使用した SERVICE_CALL_RESPONSE

呼び出しコンテキストが SERVICE_CALL_RESPONSE で動詞が Create の場合、次のアクションに応答してコネクター・コントローラーによりインバウンド・マップが呼び出されます。

- アプリケーションにより新規の行が挿入されたことがコネクターに通知された。

  コネクターは、Create 動詞を使用してアウトバウンド・マップからアプリケーション固有のビジネス・オブジェクトを受け取ると、この挿入要求をアプリケーションに送信します。このアウトバウンド・マップには、呼び出しコンテキスト SERVICE_CALL_REQUEST があります。呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、「相互参照」変換の際に新規の関係インスタンスを関係表に書き込むことができません。これは、新しいインスタンス ID に対応するアプリケーション固有のキー値がまだないためです。
- コネクターにより、新規のアプリケーション固有の行の値に基づき、かつ Create 動詞を使用して、新規のアプリケーション固有のビジネス・オブジェクトが生成された。

  このアプリケーション固有のビジネス・オブジェクトは、コネクターから InterChange Server Express に送信され、コネクター・コントローラーで受信されます。
- コネクター・コントローラーによりインバウンド・マップが呼び出されて、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換された。

  このインバウンド・マップには、新しいアプリケーション固有のキーに対応するエントリーを関係表に作成する「相互参照」変換が含まれています。

呼び出しコンテキストが SERVICE_CALL_RESPONSE で、アプリケーション固有の動詞が Create である場合には、「相互参照」変換のために生成された Java コードによって、図 125 に示すような書き込みが関係表に対して行われます。

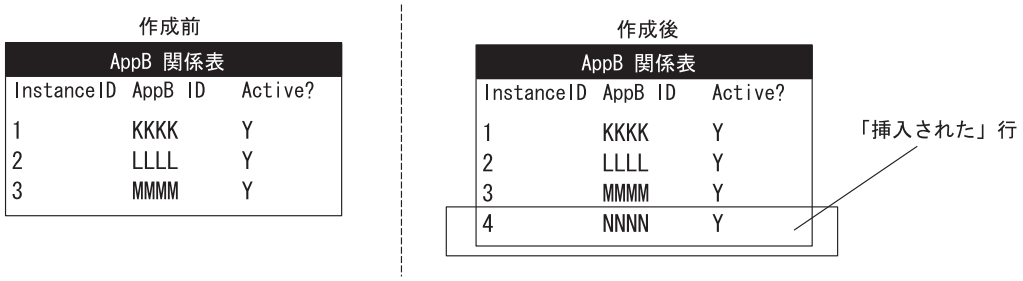

図 125. Create 動詞の場合の関係表への書き込み

AppB アプリケーションに固有の新しいキーは、「相互参照」変換によって AppA アプリケーション内の同等の値と関連付けられなければなりません。呼び出しコンテキストが EVENT_DELIVERY または ACCESS_REQUEST の場合、この関連付けに関して「相互参照」変換に求められるのは、新しい関係インスタンス ID の生成だけです。しかし、呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合には、「相互参照」変換では新しいインスタンス ID の生成だけでは不十分です。代わりに、すでに AppA キー値に割り当てたものと同じ関係インスタンス ID を AppB キー値に

割り当てる必要があります。このメソッドは、マップの実行コンテキストの一部である、オリジナル要求ビジネス・オブジェクトからの AppA キー値と関連付けされたインスタンス ID を取得します。

呼び出しコンテキストが SERVICE_CALL_RESPONSE で、動詞が Create である場合には、「相互参照」変換のために生成された Java コードによって、以下の手順が実行されます (図 125 を参照)。

- マップの実行コンテキスト内のオリジナル要求ビジネス・オブジェクトから 4 のインスタンス ID を取得します。
- このインスタンス ID (4) と新規のアプリケーション固有のキー (NNNN) に関して AppB 関係表に新規のエントリーを作成します。

EVENT_DELIVERY (または ACCESS_REQUEST) および SERVICE_CALL_RESPONSE 呼び出しコンテキストの両方 (および Create 動詞) を使用したマップの実行が完了すると、AppA と AppB の関係表は共通の関係インスタンス ID を使用してキーを関連付けます。これについて 図 126 に示します。

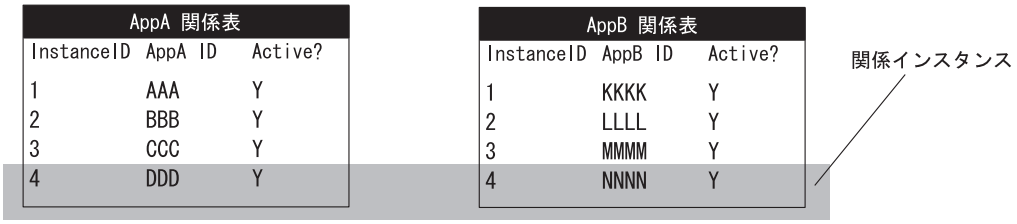

図 126. 関係インスタンスの作成

動詞が Update または Delete の場合 (インスタンス ID がすでに関係表に存在していれば、Retrieve の場合も)、「相互参照」変換の際に関係表に対して行われるのは、関係インスタンス ID の検索だけです。呼び出しコンテキストが SERVICE_CALL_RESPONSE で、アプリケーション固有の動詞が Delete である場合には、「相互参照」変換の際、関係インスタンスを非アクティブ化するため、追加のステップの実行が必要になります (図 127 を参照)。

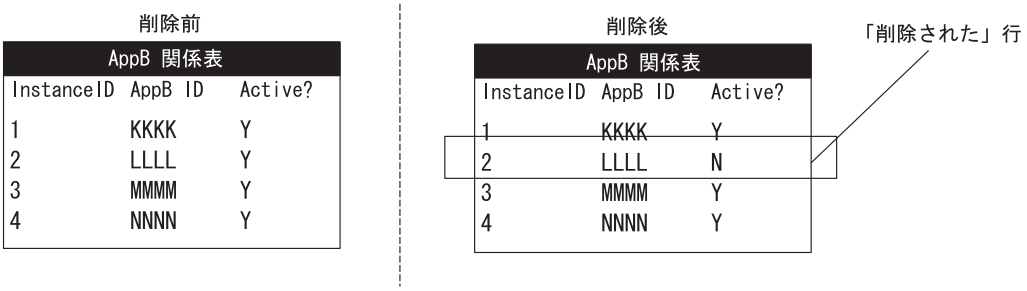

図 127. SERVICE_CALL_RESPONSE で Delete 動詞に関する関係表への書き込み

SERVICE_CALL_FAILURE 呼び出しコンテキスト

呼び出しコンテキストが SERVICE_CALL_FAILURE の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェク

トが汎用ビジネス・オブジェクトに変換されます。SERVICE_CALL_FAILURE の場合、インバウンド・マップはヌルのアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。SERVICE_CALL_FAILURE 呼び出しコンテキストは、宛先アプリケーションが新規のエンティティーの固有値を作成できず、したがってコネクターがアプリケーション固有のビジネス・オブジェクトを戻すことができないことを示すため、Create 動詞では重要です。「相互参照」変換のタスクは、表 98 に示すように、すべての動詞に対して同じです。

表 98. SERVICE_CALL_FAILURE 呼び出しコンテキストのアクション

アプリケーション固有のビジネス・オブジェクトの動詞	maintainSimpleIdentityRelationship() が実行するアクション
Create、Delete、Update、Retrieve	1. 汎用ビジネス・オブジェクトからキー値 (関係インスタンス ID) を取得します。この汎用ビジネス・オブジェクトはマップの実行コンテキスト内にあります。 2. 検索されたインスタンス ID を汎用ビジネス・オブジェクトにコピーします。

ACCESS_RESPONSE 呼び出しコンテキスト

呼び出しコンテキストが ACCESS_RESPONSE の場合、呼び出されるマップはコール・トリガー・フローの結果としてのアウトバウンド・マップです。これにより汎用ビジネス・オブジェクトがアプリケーション固有のビジネス・オブジェクトに変換されます。アウトバウンド・マップは汎用ビジネス・オブジェクトを入力として受け取り、アプリケーション固有のビジネス・オブジェクトを出力として戻します。したがって、「相互参照」変換のタスクは、表 99 に示すように、すべての動詞に対して同じです。

表 99. ACCESS_RESPONSE 呼び出しコンテキストのアクション

汎用ビジネス・オブジェクトの動詞	maintainSimpleIdentityRelationship() が実行するアクション
Create、Delete、Update、Retrieve	1. 汎用ビジネス・オブジェクトからキー値 (関係インスタンス ID) を取得します。この汎用ビジネス・オブジェクトはマップの実行コンテキスト内にあります。 2. 関係インスタンス ID を整数値に変換します。この変換が失敗した場合は、例外がスローされます。 3. オリジナル要求ビジネス・オブジェクトのキー値をアプリケーション固有のビジネス・オブジェクトにコピーします。

ACCESS_RESPONSE のオリジナル要求ビジネス・オブジェクトはアプリケーション固有のビジネス・オブジェクトなので、「相互参照」変換により、マップの実行コンテキスト (cwExecCtx) 内のオリジナル要求ビジネス・オブジェクトからキー値が自動的に取得されます。

「相互参照」変換では、オリジナル要求ビジネス・オブジェクトへのアクセスが可能である限り、表 99 のタスクを実行できます。しかし、ある場合にはこのビジネ

ス・オブジェクトにアクセスできないことがあります。例えば、「相互参照」変換では、基本要求に存在しない子オブジェクトを処理するとき、その子ビジネス・オブジェクトの関係インスタンス ID の検索を試みます。該当する関係インスタンスが見つからない場合は、その子オブジェクトのキーに CxIgnore 値を設定します。

単純一致関係の変換規則の定義

「相互参照」関係の定義については、53 ページの『一致関係の相互参照』を参照してください。

子レベルの単純一致関係のコーディング

子ビジネス・オブジェクトに固有なキー属性がある場合、単純一致関係でこれらの子ビジネス・オブジェクトを関連付けできます。

次のセクションで、この単純一致関係をコーディングする手順を説明します。

- 『子関係定義の作成手順』
- 『親マップのカスタマイズ手順』
- 300 ページの『サブマップのカスタマイズ手順』

子関係定義の作成手順: 子ビジネス・オブジェクト間の単純一致関係の関係定義を作成するには、次の手順を実行します。

1. 参加者タイプが子ビジネス・オブジェクトである参加者定義を作成します。
2. 子ビジネス・オブジェクトのキー属性に参加者属性を設定します。

   ヒント: 子ビジネス・オブジェクトを展開して、キー属性を選択します。
3. 各参加者について、ステップ 1 と 2 を繰り返します。すべての単純一致関係の場合のように、この関係には汎用ビジネス・オブジェクトに関して 1 参加者と、アプリケーション固有のビジネス・オブジェクトに関して少なくとも 1 参加者を含みます。各参加者には単一属性、つまりビジネス・オブジェクトのキーが含まれます。

親マップのカスタマイズ手順: 親ビジネス・オブジェクトのマップ (メイン・マップ) で、子ビジネス・オブジェクトを含む属性にマッピング・コードを追加します。この属性の Activity Editor で、次の手順を実行して単純一致関係をコーディングします。

1. 子オブジェクトのサブマップを作成した場合、メイン・マップの子属性からこのサブマップを呼び出します。通常、子オブジェクトのマッピング変換はサブマップ内で行われます。特に子オブジェクトが複数カーディナリティーの場合はサブマップ内で行われます。
2. General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックを使用して、ユーザーのソース子オブジェクトの動詞を設定します。

   General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックの最後のパラメーターは boolean 値フラグであり、子オブジェクトが複合関係に参加しているかどうかを示します。この子オブジェクトは複合一致関係でなく単純一致関係に参加しているため、General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックへの最後の引き数は false にする必要があります。子オブジェクトにサブマップが存在する場合は、General/APIs/Identity Relationship/Maintain Child

Verb 関数ブロックを*呼び出し*てから、サブマップを呼び出します。詳細については、314 ページの『ソース子動詞の設定』を参照してください。

注: 親ビジネス・オブジェクトのキー属性も単純一致関係に参加している場合は、メイン・マップに「相互参照」変換を定義します (53 ページの『一致関係の相互参照』を参照)。

サブマップのカスタマイズ手順: サブマップで、以下の手順を実行します。

1. 子ビジネス・オブジェクトに対して「移動」変換または「値を設定」変換を定義します。
2. 子ビジネス・オブジェクトに対して「相互参照」変換を定義し、関係名と参加者を指定します。詳細については、53 ページの『一致関係の相互参照』を参照してください。

複合一致関係の使用

一致関係は、ビジネス・オブジェクトまたは他のデータ間に *1 対 1* の関連を設定します。複合一致関係では、複合キー属性で 2 つのビジネス・オブジェクトを関連付けします。

次のセクションで、複合一致関係を使用する手順を説明します。

- 『複合一致関係定義の作成』
- 301 ページの『関係アクションの判別』
- 303 ページの『複合一致関係に関連するマップの規則のカスタマイズ』

複合一致関係定義の作成

一致関係定義は、参加者タイプがビジネス・オブジェクトであり `data` タイプでない ということが、参照関係定義と異なっています (参加者タイプ・リストでの最初の選択)。単純一致関係の場合と同様に、次の特徴があります。

- 複合一致関係は汎用ビジネス・オブジェクトと少なくとも 1 つのアプリケーション固有のビジネス・オブジェクトで構成されます。
- 参加者タイプは、すべて の参加者に関して 1 つのビジネス・オブジェクトです。

しかし、複合一致関係の場合、すべての参加者に関する参加者属性は複合キーです。この複合キーは、通常は親ビジネス・オブジェクトの固有キーと子ビジネス・オブジェクトの非固有キーで構成されます。

複合一致関係定義の作成手順

複合一致関係の関係定義を作成するには、次の手順を実行します。

1. 参加者タイプが親ビジネス・オブジェクトである参加者定義を作成します。
2. 最初の参加者属性を親ビジネス・オブジェクトのキーに設定します。

   ヒント: 親ビジネス・オブジェクトを展開して、キー属性を選択します。
3. 2 番目の参加者属性を子属性のキーに追加します。

ヒント: 親ビジネス・オブジェクトを展開して、親内の子属性を展開します。この子オブジェクトからキー属性を選択します。

4. 各参加者について、ステップ 1 から 3 を繰り返します。すべての複合一致関係の場合のように、この関係は汎用ビジネス・オブジェクトに関して 1 参加者と、アプリケーション固有のビジネス・オブジェクトに関して少なくとも 1 参加者を含みます。各参加者は、親ビジネス・オブジェクトのキーと (親ビジネス・オブジェクト内の属性の) 子ビジネス・オブジェクトのキーの、2 つの属性から構成されます。

制限: 複合関係を管理するために、サーバーは内部表を作成します。表は関係の役割ごとに作成されます。さらに、関係のすべてのキー属性 にわたってこれらの表に固有のインデックス が作成されます。(すなわち、関係のキー属性に対応する列はインデックスの参加者です。) 内部表の列サイズは、関係の属性と直接関連し、関係の MaxLength 属性の値によって決定されます。

通常、データベースでは作成可能なインデックスのサイズが制限されています。例えば、DB2 では、デフォルト・ページ・サイズの場合、インデックスは 1,024 バイトまでに制限されます。したがって、関係の MaxLength 属性および関係の属性数によっては、複合関係の作成中にインデックス・サイズの制限に達する場合があります。

要確認:

- 関係のすべてのキー属性のリポジトリー・ファイルで適切な MaxLength 値を設定し、総インデックスが下位 DBMS のインデックス・サイズの制限を超えないようにしてください。

  型 String の MaxLength 属性が指定されていない場合、SQLServer でのデフォルトは nvarchar(255) となります。したがって、関係に *N* 個のキーがあり、型がすべて String であり、デフォルトの MaxLength 属性が 255 バイトである場合は、インデックス・サイズは ((N*255)*2) + 16 バイトになります。このことから明らかなように、N >= 2 の値を取り、String のデフォルトの MaxLength 値が 255 バイトである場合は、SQLServer 7 の制限である 900 バイトを超えてしまいます。
- また、一部の DBMS は大規模なインデックスをサポートしていますが、パフォーマンスが低下してしまうため、必ずインデックス・サイズを最小限にしておくようにしてください。

複合一致関係の関係定義の作成方法の詳細については、267 ページの『一致関係の定義』を参照してください。

関係アクションの判別

表 100 が示しているのは、マッピング API に用意されている、親ソース・ビジネス・オブジェクトの子属性から複合一致関係を保守するためのアクティビティー関数ブロックです。これらのメソッドが行うアクションは、ソース・オブジェクトの動詞と呼び出しコンテキストにより異なります。

表 100. 子属性の複合一致関係の維持

関数ブロック	説明
General/APIs/Identity Relationship/ Maintain Child Verb	ソース子動詞を正しく設定します。
General/APIs/Identity Relationship/ Maintain Composite Relationship	関係表に適切なアクションを実行します。

General/APIs/Identity Relationship/Maintain Composite Relationship のアクション

Maintain Composite Relationship 関数ブロックは、マッピング APImaintainCompositeRelationship() を呼び出す Java コードを生成します。呼び出された API は、複合一致関係の関係表を管理します。このメソッドで、各関係インスタンス ID について関連するアプリケーション固有のキー値が関係インスタンスに確実に含まれるようになります。このメソッドは、複合一致関係の参加者および関係インスタンスの基本的な追加や削除のすべてを自動的に処理します。

maintainCompositeRelationship() が行うアクションは、ビジネス・オブジェクトの動詞の値と呼び出しコンテキストにより異なります。メソッドは指定された参加者の子オブジェクト間を繰り返し、各子オブジェクトで maintainSimpleIdentityRelationship() を呼び出して子のキー値を正しく設定します。maintainSimpleIdentityRelationship() の場合のように、maintainCompositeRelationship() が行うアクションは、次の情報により異なります。

- 呼び出しコンテキスト: EVENT_DELIVERY、ACCESS_REQUEST、SERVICE_CALL_REQUEST、SERVICE_CALL_RESPONSE、SERVICE_CALL_FAILURE、および ACCESS_RESPONSE。
- ソース・ビジネス・オブジェクトの動詞: Create、Update、Delete、または Retrieve。

maintainSimpleIdentityRelationship() が実行するアクションの詳細については、288 ページの『一致関係表のアクセス』を参照してください。

maintainCompositeRelationship() メソッドは、2 つのネスト・レベルにのみ展開される複合キーのみ を処理します。つまりメソッドは、子オブジェクトの複合キーが祖父母オブジェクトの値に基づく場合は処理できません。

例: A がトップレベル・ビジネス・オブジェクト、A の子が B、B の子が C の場合、2 つのメソッドは子オブジェクト C の参加者定義をサポートしません。これは次のようになります。

- 参加者タイプは A で、属性は次のとおりです。

```
key attribute of A: ID
key attribute of B: B[0].ID
key attribute of C: B[0].C[0].ID
```

- 参加者タイプは A で、属性は次のとおりです。

```
key attribute of A: ID
key attribute of C: B[0].C[0].ID
```

孫オブジェクトにアクセスするために、これらのメソッドは次のように参加者定義のみサポートします。

- 参加者タイプは B で、属性は次のとおりです。

```
key attribute of B: ID
key attribute of C: C[0].ID
```

- 参加者タイプは B で、属性は次のとおりです。

```
key attribute of B: ID
first key attribute of C: C[0].ID1
second key attribute of C: C[0].ID2
```

General/APIs/Identity Relationship/Maintain Child Verb のアクション

Maintain Child Verb 関数ブロックは、マッピング APImaintainChildVerb() を呼び出す Java コードを生成します。呼び出された API は、宛先ビジネス・オブジェクトの子オブジェクトの動詞を保守します。またキー属性が複合一致関係の一部である子オブジェクトを処理することができます。複合関係の一部として maintainChildVerb() を呼び出す場合は、その最後のパラメーターの値が true であることを確認してください。このメソッドにより、親ソース・オブジェクトと呼び出しコンテキストに適切に動詞が設定されます。maintainChildVerb() のアクションの詳細については、314 ページの『ソース子動詞の設定』を参照してください。

複合一致関係に関連するマップの規則のカスタマイズ

複合一致関係に関連する関係定義と参加者定義の作成が完了すると、複合一致関係を保守できるようにマップをカスタマイズすることができるようになります。複合一致関係では、複合キーが管理されます。したがって、このような関係の管理では、複合キーを構成する 2 つの部分の管理が必要になります。複合一致関係をコーディングするには、親ビジネス・オブジェクトと子ビジネス・オブジェクトの両方のマッピング変換規則をカスタマイズする必要があります (表 101 を参照)。

表 101. 複合一致関係関連のアクティビティー関数ブロック

関係するマップ	関係するビジネス・オブジェクト	属性	アクティビティー関数ブロック
メイン	親ビジネス・オブジェクト	トップレベル・ビジネス・オブジェクト	「相互参照」変換規則を使用する。
		子属性 (子ビジネス・オブジェクト)	General/APIs/Identity Relationship/Maintain Composite Relationship General/APIs/Identity Relationship/Maintain Child Verb General/APIs/Identity Relationship/Update My Children (オプション)
サブマップ	子ビジネス・オブジェクト	キー属性 (非固有キー)	動詞に対して「移動」変換または「値を設定」変換を定義する。

子ビジネス・オブジェクトに非固有キー属性がある場合、複合一致関係でこれらの子ビジネス・オブジェクトを関連付けできます。

次のセクションで、この複合一致関係をカスタマイズする手順を説明します。

- 『メイン・マップのカスタマイズ手順』
- 308 ページの『サブマップのカスタマイズ』
- 309 ページの『子インスタンスの管理』

メイン・マップのカスタマイズ手順

親ビジネス・オブジェクトのマップ (メイン・マップ) で、親属性にマッピング・コードを追加します。

1. 「移動」変換規則または「値を設定」変換規則を定義して、トップレベル・ビジネス・オブジェクトの動詞をマップする。
2. トップレベル・ビジネス・オブジェクトどうしの間に「相互参照」変換を定義する。
3. 子属性に対してカスタム変換を定義し、Activity Editor で General/APIs/Identity Relationship/Maintain Composite Relationship 関数ブロックを追加する。

子属性のコーディング手順

親オブジェクトの子属性には、子ビジネス・オブジェクトが含まれています。子オブジェクトは、通常は複数カーディナリティーのビジネス・オブジェクトです。またキー属性を含み、その値で子が識別されます。しかし、このキー値は固有である必要はありません。したがって、同じ親の子オブジェクト間で 1 つの子オブジェクトを一意的に識別せず、親オブジェクトのすべてのインスタンスの子オブジェクト間で子オブジェクトを識別するのも十分でありません。

このような子オブジェクトを一意的に識別するために、関係で複合キーが使用されます。複合キーでは、親キーにより親オブジェクトが一意的に識別されます。親キーと子キーの組み合わせにより、子オブジェクトが一意的に識別されます。親ビジネス・オブジェクトのマップ (メイン・マップ) で、子ビジネス・オブジェクトを含む属性にマッピング・コードを追加します。この属性の Activity Editor で、次の手順を実行して複合一致関係をコーディングします。

1. メイン・マップの子ビジネス・オブジェクト属性に対して「サブマップ」変換を定義します。通常、子オブジェクトのマッピング変換はサブマップ内で行われます。特に子オブジェクトが複数カーディナリティーの場合はサブマップ内で行われます。
2. メイン・マップで子動詞に対してカスタム変換規則を定義し、General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックを追加して、子ビジネス・オブジェクトの動詞を保守できるようにします。

   General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックの最後の入力パラメーターは boolean 値フラグであり、子オブジェクトが複合関係に参加しているかどうかを示します。この子オブジェクトは単純一致関係でなく複合一致関係に参加しているので、`maintainChildVerb()` への最後の引き数として `true` を確実に渡します。 `maintainChildVerb()` を呼び出した後で サブマップを呼び出します。詳細については、314 ページの『ソース子動詞の設定』を参照してください。
3. 親ソース・オブジェクトの複合キーを保守できるようにするため、マッピング規則をカスタマイズして General/APIs/Identity Relationship/Maintain Composite Relationship 関数ブロックを追加します。

4. 親オブジェクトの動詞が Update でその理由が子オブジェクトの削除である場合に関係表を保守できるようにするため、マッピング規則をカスタマイズして General/APIs/Identity Relationship/Update My Children 関数ブロックを追加します。

   ヒント: Update My Children 関数ブロックを含む変換規則は、Maintain Composite Relationship 関数ブロックを含む変換規則の後に実行されるようにしてください。

複合一致関係に関連するマップのカスタマイズの例

次の例では、複合一致関係用にマップをカスタマイズする方法について説明します。

1. メイン・マップで、子ビジネス・オブジェクトの動詞の間にカスタム変換規則を定義します。カスタマイズされたアクティビティーで General/APIs/Identity Relationship/Maintain Child Verb 関数ブロックを使用して、子ビジネス・オブジェクトの動詞を保守します。

   このカスタム・アクティビティーの目的は、maintainChildVerb() API を使用して、子ビジネス・オブジェクトの動詞を、マップ実行コンテキストおよび親ビジネス・オブジェクトの動詞に基づいて設定することです。図 128 は、このカスタム・アクティビティーを示します。

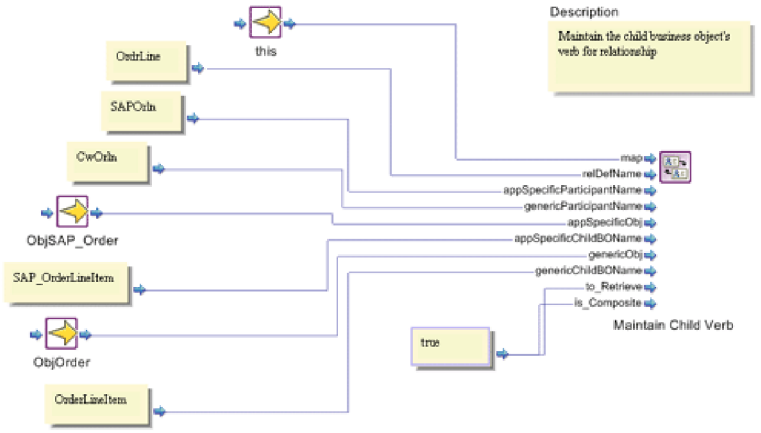

図 128. Maintain Child Verb の関数ブロックの使用

2. 必要な場合は、子レベルで必要なマッピングを実行するように子ビジネス・オブジェクト間のサブマップ変換規則を定義します。
3. トップレベル・ビジネス・オブジェクトどうしの間にカスタム変換規則を定義します。カスタマイズされたアクティビティーで General/APIs/Identity Relationship/Maintain Composite Relationship 関数ブロックを使用して、このマップの複合一致関係を保守します。

   このカスタム・アクティビティーの目的は、maintainComposite Relationship() API を使用して、マップ内の複合一致関係を保守することです。図 129 は、このカスタム・アクティビティーを示します。

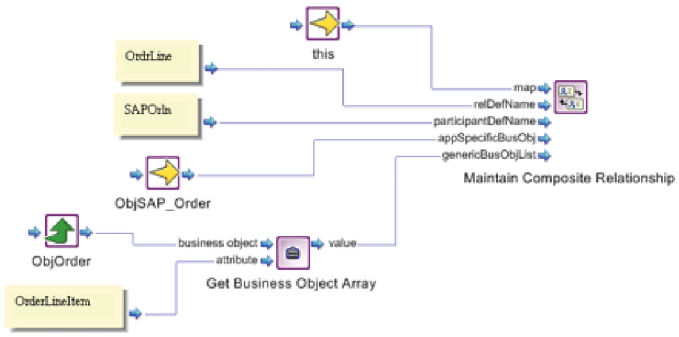

図 129. 複合関係の保守のための関数ブロックの使用

4. ソースのトップレベル・ビジネス・オブジェクトから宛先の子ビジネス・オブジェクト属性へのカスタム変換規則マッピングを定義します。カスタマイズされたアクティビティーで General/APIs/Identity Relationship/Update My Children 関数ブロックを使用して、関係の子インスタンスを保守します。

   このカスタム・アクティビティーの目的は、updateMyChildren() API を使用して、一致関係における指定の親/子関係の子インスタンスを追加または削除することです。図 130 は、このカスタム・アクティビティーを示します。

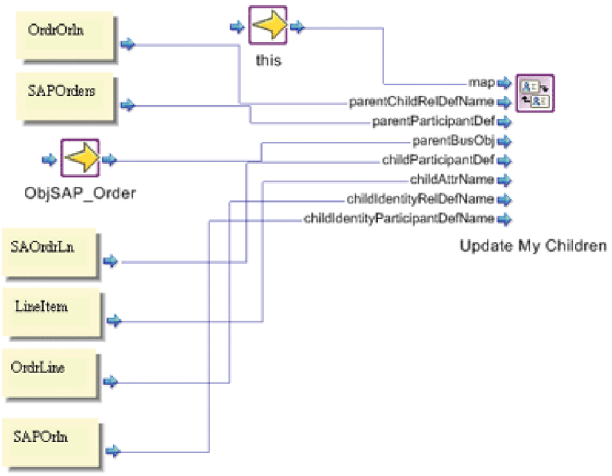

図 130. Update My Children の関数ブロックの使用

子属性のコーディングの例

以下は、親マップの子属性のコーディング例です。このコード・フラグメントは、SAP Order ビジネス・オブジェクトの Order Line Item 属性に存在します。ここでは `maintainChildVerb()` を使用して子オブジェクト動詞を設定し、その後 Order line Item 子オブジェクトのマッピングを処理するために `for` ループ内でサブマップ (`Sub_SaOrderLieItem_to_CwOrderLineItem`) を呼び出します。

```
{
BusObjArray srcCollection_For_ObjSAP_Order_SAP_OrderLineItem =
   ObjSAP_Order.getBusObjArray("SAP_OrderLineItem");

//
// LOOP ONLY ON NON-EMPTY ARRAYS
// -----------------------------
//
// Perform the loop only if the source array is non-empty.
//
if ((srcCollection_For_ObjSAP_Order_SAP_OrderLineItem != null) &&
      (srcCollection_For_ObjSAP_Order_SAP_OrderLineItem.size() > 0))
   {
   int currentBusObjIndex_For_ObjSAP_Order_SAP_OrderLineItem;
   int lastInputIndex_For_ObjSAP_Order_SAP_OrderLineItem =
   srcCollection_For_ObjSAP_Order_SAP_OrderLineItem.getLastIndex();

   // ----
   IdentityRelationship.maintainChildVerb(
      "OrdrLine",
      "SAPOrln",
      "CWOrln",
      ObjSAP_Order,
      "SAP_OrderLineItem",
      ObjOrder,
      "OrderLineItem",
      cwExecCtx,
      true,
      true);

   // ----
   for (currentBusObjIndex_For_ObjSAP_Order_SAP_
         OrderLineItem = 0;
         currentBusObjIndex_For_ObjSAP_Order_SAP_OrderLineItem <=
            lastInputIndex_For_ObjSAP_Order_SAP_OrderLineItem;
         currentBusObjIndex_For_ObjSAP_Order_SAP_OrderLineItem++)
      {
      BusObj currentBusObj_For_ObjSAP_Order_SAP_OrderLineItem =
(BusObj) (srcCollection_For_ObjSAP_Order_SAP_OrderLineItem.elementAt(
            currentBusObjIndex_For_ObjSAP_Order_SAP_OrderLineItem));

      //
      // INVOKE MAP ON VALID OBJECTS
      // ---------------------------
      //
      // Invoke the map only on those children objects that meet
      // certain criteria.
      //
      if (currentBusObj_For_ObjSAP_Order_SAP_OrderLineItem != null)
         {
         BusObj[] _cw_inObjs = new BusObj[2];
         _cw_inObjs[0] =
               currentBusObj_For_ObjSAP_Order_SAP_OrderLineItem;
         _cw_inObjs[1] = ObjSAP_Order;
         logInfo ("*** Inside SAPCW header, verb is: " +
            (_cw_inObjs[0].getVerb()));

         try
            {
            BusObj[] _cw_outObjs = DtpMapService.runMap(
               "Sub_SaOrderLineItem_to_CwOrderLineItem",
               "CwMap",
               _cw_inObjs,
               cwExecCtx);
            _cw_outObjs[0].setVerb(_cw_inObjs[0].getVerb());
            ObjOrder.setWithCreate("OrderLineItem", _cw_outObjs[0]);
            }
```

```
         catch (MapNotFoundException me)
            {
            logError(5502,
               " Sub_SaOrderLineItem_to_CwOrderLineItem ");
            throw new MapFailureException ("Submap not found");
            }
         }
      }

   // Start of the child relationship code
   BusObjArray temp = (BusObjArray)ObjOrder.get("OrderLineItem");
   try
      {
      IdentityRelationship.maintainCompositeRelationship(
         "OrdrLine",
         "SAPOrln",
         ObjSAP_Order,
         temp,
         cwExecCtx);
      }

   catch RelationshipRuntimeException re
      {
      logError(re.toString());
      }

   // This call to updateMyChildren() assumes the existence of the
   // OrdrOrln parent/child relationship between the SAP_Order
   // (parent) and SAP_OrderItem (child)
   IdentityRelationship.updateMyChildren(
      "OrdrOrln",
      "SAOrders",
      ObjSAP_Order,
      "SAOrdrLn",
      "LineItem",
      "OrdrLine",
      "SAPOrln",
      cwExecCtx);

   // End of the child relationship code
   }
}
```

サブマップのカスタマイズ

子ビジネス・オブジェクトのマップ (サブマップ) で、マッピング・コードを子オブジェクトのキー属性に追加します。追加する必要のある唯一のコードは、子オブジェクトの動詞を親オブジェクトの動詞に設定する `setVerb()` メソッドの呼び出しです。詳細については、40 ページの『宛先ビジネス・オブジェクトの動詞の設定』を参照してください。

注: 子オブジェクトの基本キーで `maintainCompositeRelationship()` メソッドが必要な場合は、親マップ内の、サブマップを呼び出す `for` ループの最後のすぐ後で呼び出します。サブマップでは、宛先オブジェクトの基本キーのコードに、次の行が組み込まれている必要があります。

```
// maintainCompositeRelationship()
is called in the parent map.
```

子インスタンスの管理

Activity Editor には、表 102 に示すような、一致関係に関わる親に属する子オブジェクト・インスタンスを管理するための関数ブロックが用意されています。

表 102. 子インスタンスを管理するための関数ブロック

関数ブロック	説明
General/APIs/Identity Relationship/Add My Children	子関係インスタンスを親/子関係表に追加します。
General/APIs/Identity Relationship/Delete My Children	親/子関係表から子関係インスタンスを削除します。
General/APIs/Identity Relationship/Update My Children	親/子関係表から子関係インスタンスを削除または追加します。

注: 表 102 に示した関数ブロックの最も一般的な用途は、複合一致関係を含むカスタム関係に関わる子ビジネス・オブジェクトの保守です。

表 102 の関数ブロックでは、渡される親ビジネス・オブジェクトは変更後イメージであることが前提になっています。つまり、動詞の操作が行われた*後* のビジネス・オブジェクトのイメージであることが前提になっています。例えば、ビジネス・オブジェクトに、新規の子オブジェクトの追加により発生した更新の Update 動詞がある場合、これらの新規子オブジェクトはビジネス・オブジェクトにすでに存在します。同様に、ビジネス・オブジェクトに、子オブジェクトの削除により発生した更新の Update 動詞がある場合、そのビジネス・オブジェクトでは、該当の子オブジェクトはすでに削除されています。

以下のセクションでは、子インスタンスの管理手順について説明します。

- 『親/子関係定義の作成』
- 310 ページの『親ビジネス・オブジェクトに対する更新処理』

親/子関係定義の作成

親/子関係 は、親 (1) と子 (多) の 1 対多の関係です。親/子関係には、次の参加者が含まれます。

- 親ビジネス・オブジェクトのキー属性を含む参加者。
- 子ビジネス・オブジェクトのキーを含む参加者。

表 102 の関数ブロックは、親/子関係の関係表を利用して、特定の親ビジネス・オブジェクトに関連付けられている子ビジネス・オブジェクトを追跡します。

親/子関係定義の作成手順

親/子関係の関係定義を作成するには、Relationship Designer Express で次の手順を実行します。

1. 参加者タイプが親ビジネス・オブジェクトである参加者定義を作成します。
2. 参加者属性を親ビジネス・オブジェクトのキーに設定します。

   親ビジネス・オブジェクトを展開して、キー属性を選択します。
3. 参加者タイプが子ビジネス・オブジェクトである参加者定義を作成します。
4. 参加者属性を子属性のキーに追加します。

子ビジネス・オブジェクト (親オブジェクトの子属性ではない) を展開して、この子オブジェクトからキー属性を選択します。

注: 親子関係は、子オブジェクトが固有キーを持たない 場合にのみ維持する必要があります。つまり、子オブジェクトはその親のコンテキスト内にのみ 存在します。

詳細については、267 ページの『一致関係の定義』を参照してください。

親ビジネス・オブジェクトに対する更新処理

このセクションには、複合一致関係に参加する子オブジェクトが更新時に正しく管理されるようにするための説明があります。

- 『変更前イメージと変更後イメージの比較』
- 311 ページの『Update My Children の使用についてのヒント』

変更前イメージと変更後イメージの比較

Update My Children 関数ブロックは、親/子関係の関係表を更新します。親/子関係は、子オブジェクトが親ビジネス・オブジェクトに追加されたのか、それとも親ビジネス・オブジェクトから削除されたのかの判別を容易にするために必要です。

指定された親ビジネス・オブジェクトについて、このメソッドでビジネス・オブジェクトの次のイメージが一致することが確認されます。

- 変更前イメージ情報が親/子関係の関係表に組み込まれる。
- 変更後イメージが親ビジネス・オブジェクトに組み込まれる。

マップで子ビジネス・オブジェクトが削除されたことを検出するには、親ビジネス・オブジェクトが更新前に持っていた (変更前イメージ) このタイプの子オブジェクトのインスタンス数を判別して、親オブジェクトが現在所持している (変更後イメージ) インスタンス数と比較する必要があります。マップでは、Update My Children 関数ブロックを使用してこの比較を行い、削除または追加されたものを検出することができます。

Update My Children は、変更前イメージと変更後イメージを比較するときに、親ビジネス・オブジェクトに存在しない 子オブジェクトの関係表から、関連する関係インスタンスを除去するかどうかを決定することができます。このメソッドは、次の関係表から関係インスタンスを除去します。

- 親/子関係内の子参加者の関係表。
- 親および子オブジェクトを含む複合一致関係内の参加者の関係表。

注: Update My Children は、親ビジネス・オブジェクトに*存在する* (しかし子関係表には存在しない) 子オブジェクトの関係表にインスタンスを追加することもできます。ただし、複合一致関係のコンテキストで呼び出された場合には、この処理は必要ありません。親オブジェクトの新規子オブジェクトは、いずれも Maintain Composite Relationship 関数ブロックによってすでに関係表に追加されています。詳細については、302 ページの『General/APIs/Identity Relationship/Maintain Composite Relationship のアクション』を参照してください。

Update My Children の使用についてのヒント

複合一致関係に関わる子オブジェクトの関係表を Update My Children 関数ブロックを使用して保守する場合は、以下のヒントに留意してください。

- Update My Children 関数ブロックは、Maintain Composite Relationship 関数ブロックの後 に使用してください。また、子ビジネス・オブジェクトに適切な動詞が設定されていることを確認してください。
- Update My Children 関数ブロックが必要なのは、複合関係に関わる子オブジェクトを追跡する場合だけです。

  単純一致関係に関わる子オブジェクトを追跡する場合には、Update My Children 関数ブロックを使用する必要はありません。詳細については、299 ページの『子レベルの単純一致関係のコーディング』を参照してください。
- Update My Children 関数ブロックは、Maintain Composite Relationship 関数ブロックと同様に、親とその直接の子の 2 つのネスト・レベルにのみまたがる複合キーに限り処理します。

  つまり、メソッドは孫オブジェクトの複合キーが祖父母オブジェクトの値に基づく場合は処理できません。例えば、A がトップレベル・ビジネス・オブジェクト、A の子が B、B の子が C の場合、2 つのメソッドは子オブジェクト C の参加者定義をサポートしません。これは次のようになります。
  - 参加者タイプは A で、属性は次のとおりです。

    ```
    key attribute of A: ID
    key attribute of B: B[0].ID
    key attribute of C: B[0].C[0].ID
    ```
  - 参加者タイプは A で、属性は次のとおりです。

    ```
    key attribute of A: ID
    key attribute of C: B[0].C[0].ID
    ```

  孫オブジェクトにアクセスするために、これらのメソッドは次のように参加者定義のみサポートします。
  - 参加者タイプは B で、属性は次のとおりです。

    ```
    key attribute of B: ID
    key attribute of C: C[0].ID
    ```
  - 参加者タイプは B で、属性は次のとおりです。

    ```
    key attribute of B: ID
    first key attribute of C: C[0].ID1
    second key attribute of C: C[0].ID2
    ```
- Update My Children 関数ブロックによって親/子関係表が管理されるのは、呼び出しコンテキストが EVENT_DELIVERY である場合と SERVICE_CALL_RESPONSE である場合に限られます。

  呼び出しコンテキストが SERVICE_CALL_REQUEST または ACCESS_RESPONSE である場合、Update My Children 関数ブロックを実行しても、この関係表は変更されません。
- Update My Children 関数ブロックは、子ビジネス・オブジェクトが固有 ID を持っている場合、つまり子オブジェクトが単純一致関係に参加する場合にも使用できます。この場合、親/子関係を定義する必要があります (309 ページの『親/子関係定義の作成』を参照)。

動詞の設定

このセクションでは、マップに参加しているビジネス・オブジェクトの動詞の設定方法について説明します。

- 『宛先動詞の条件付設定』
- 314 ページの『ソース子動詞の設定』

宛先ビジネス・オブジェクトの動詞の設定の詳細については、40 ページの『宛先ビジネス・オブジェクトの動詞の設定』を参照してください。

宛先動詞の条件付設定

通常、動詞に関しては、「移動」変換を定義して宛先動詞にソース動詞の値を設定するだけで十分です。(このアクションの詳細については、40 ページの『宛先ビジネス・オブジェクトの動詞の設定』を参照。)しかし、ソース・アプリケーションで通常と異なる方法でビジネス・オブジェクト動詞が設定されることがあります。例えば、イベントが新規でも動詞に Update が設定される場合です。別の例では、動詞に常に Retrieve が設定される場合があります。このような状況の場合、マップで宛先動詞を実際のイベントに対応する動詞にリセットする必要があります。

ソース・ビジネス・オブジェクトのキーが関係に参加する場合、マップにより関係表の*静的参照* を実行して、ソース・ビジネス・オブジェクトが存在するかどうかを判別できます。その後、対応するエントリーがテーブル内で検出されるかどうかに基づいて、マップで宛先動詞に Update または Create のいずれかを宛先動詞に設定できます。この静的参照は、参照関係にアクセスする方法とおおよそ同じ方法で行います。表 103 に、各種の静的参照に使用する関数ブロックを示します。

表 103. ソース・ビジネス・オブジェクトの存在の検査

ソース・ビジネス・オブジェクトのタイプ	マップ・タイプ	関数ブロック
アプリケーション固有	インバウンド	General/APIs/Relationship/Retrieve Instances
汎用	アウトバウンド	General/APIs/Relationship/Retrieve Participants

インバウンド・マップのカスタマイズ手順の例

次の例では、インバウンド・マップで、条件に従って参照の結果を基に宛先動詞を設定できるようにする方法を示します。

1. マップで、ソース・ビジネス・オブジェクトと宛先動詞の間にカスタム変換を定義します。
2. カスタム変換のアクティビティーで、以下の手順を実行します。このアクティビティーの目標は、関係の参加者が関わっているインスタンスの数を確認することです。関係の参加者のインスタンスが存在しない場合、宛先ビジネス・オブジェクトの動詞を Create にします。それ以外の場合は、Update にします。

a. 関係の参加者が関わっているインスタンスの数を確認できるように、アクティビティーを定義します (図 131 を参照)。

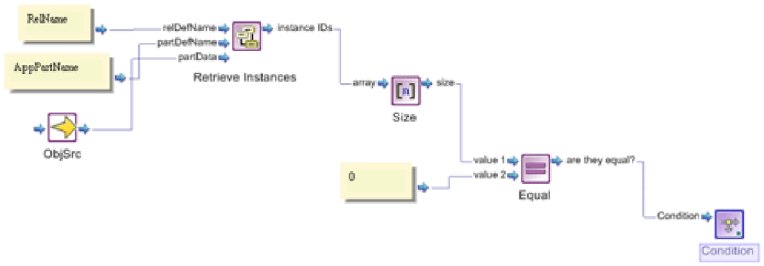

図 131. 関係の参加者が関わっているインスタンスの数の確認

b. キャンバスで、Condition 関数ブロックをダブルクリックして開きます。「True Action」を選択して、条件が true の場合に実行するアクションを定義します。「True Action」は、図 132 のとおりに定義してください。

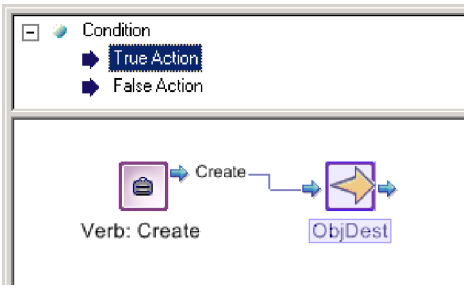

図 132. 「True Action」の定義

c. 「False Action」を選択して、参加者が関わっている関係インスタンスの数がゼロではない場合に実行するアクションを定義します。「False Action」は、図 133 のとおりに定義してください。

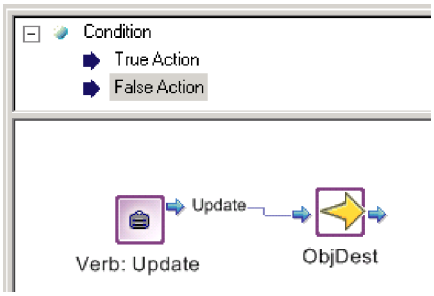

図 133. 「False Action」の定義

アウトバウンド・マップのカスタマイズ手順の例

アウトバウンド・マップでも、前述の手順と同様の手順により、汎用オブジェクトの基本キーを基に静的参照を実行することができます。ただし、そのためには、

General/APIs/Relationship/Retrieve Instances の関数ブロックではなく、General/APIs/Relationship/Retrieve Participants の関数ブロックを使用する必要があります。手順は以下のとおりです。

1. マップで、ソース・ビジネス・オブジェクトのキー属性と宛先動詞の間にカスタム変換を定義します。
2. カスタム変換のアクティビティーで、以下の手順を実行します。このアクティビティーの目標は、関係の参加者の数を確認することです。関係の参加者のインスタンスが存在しない場合、宛先ビジネス・オブジェクトの動詞を Create にします。それ以外の場合は、Update にします。
   a. 関係の参加者の数を確認できるように、アクティビティーを定義します (図 134 を参照)。

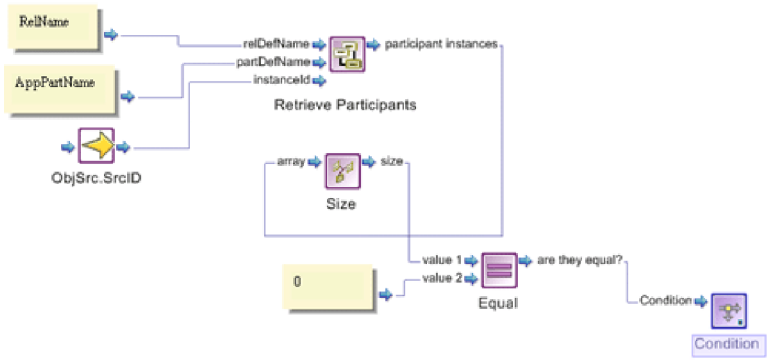

図 134. 関係の参加者の数の確認

   b. 312 ページの『インバウンド・マップのカスタマイズ手順の例』のステップ 2b と 2c を実行します。

ソース子動詞の設定

親ソース・ビジネス・オブジェクトが子ビジネス・オブジェクトを持つ場合、ソース子動詞の値は、通常、親動詞の値と同じです。したがって、ソース子オブジェクトの動詞は、親動詞から子動詞への「移動」変換を定義することによって設定します。ただし、親オブジェクトの動詞が Update の場合、更新は 表 104 に示すなんらかの変更の結果です。

表 104. 親ビジネス・オブジェクトの更新

更新タスク	子オブジェクトの動詞
親オブジェクト内の一部の子以外の属性の変更	Update
子オブジェクト内の一部の属性の変更	Update
子オブジェクトをさらに追加	Create
既存の子オブジェクトの削除	Delete

表 104 のすべての変更は、親オブジェクトでは Update 動詞で表されます。しかし、この変更のすべてが子オブジェクトに対する更新を表すわけではありません。ソース子動詞の値は、親動詞で行われたアクションにより異なります。子オブジェクトのキーが一致関係 (複合または単純) に参加する場合、ソース子動詞の値は親動

詞だけでなく呼び出しコンテキストにも依存します。このような場合は、Maintain Child Verb 関数ブロックを使用して、ソース子オブジェクトの動詞を設定します。

このセクションでは、Maintain Child Verb 関数ブロックを使用してソース子オブジェクトの動詞を維持する方法について説明します。

- 『子動詞設定の決定』
- 317 ページの『Maintain Child Verb 関数ブロックの使用についてのヒント』

子動詞設定の決定

Maintain Child Verb 関数ブロックは、親ソース・オブジェクトの動詞と呼び出しコンテキストが与えられると、それに応じてソース・ビジネス・オブジェクトの子オブジェクトの動詞を適切に設定します。このメソッドが行うアクションは、親ソース・オブジェクトの動詞と呼び出しコンテキストにより異なります。

EVENT_DELIVERY および ACCESS_REQUEST 呼び出しコンテキスト: 呼び出しコンテキストが EVENT_DELIVERY または ACCESS_REQUEST の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換されます。インバウンド・マップはアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。EVENT_DELIEVERY (または ACCESS_REQUEST) の場合、子動詞の設定時に処理する特別なケースはありません。したがって、maintainChildVerb() メソッドは、表 105 に示すように、すべての動詞の値に関して親動詞を子動詞にコピーするのみです。

表 105. EVENT_DELIVERY および ACCESS_REQUEST 呼び出しコンテキストのアクション

汎用ビジネス・オブジェクトの動詞	Maintain Child Verb 関数ブロックが実行するアクション
Create、Delete、Update、Retrieve	ソース・オブジェクト内のすべての子オブジェクトの動詞を親ソース・オブジェクト内の動詞に設定します。このアクションにより、子オブジェクト内のすべての既存の動詞が上書きされます。

SERVICE_CALL_REQUEST 呼び出しコンテキスト: 呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、呼び出されるマップはアウトバウンド・マップです。つまり、汎用ビジネス・オブジェクトがアプリケーション固有のビジネス・オブジェクトに変換されます。アウトバウンド・マップは汎用ビジネス・オブジェクトを入力として受け取り、アプリケーション固有のビジネス・オブジェクトを出力として戻します。呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、Maintain Child Verb 関数ブロックに対応する Java コードは、Update 動詞に関連する特別なケースを処理します。親オブジェクトの変更の内容が新しい子オブジェクトの作成である場合には、Maintain Child Verb 関数ブロックは、現在関係表に存在しない子オブジェクトのすべてで動詞を Create に変更します (表 106 を参照)。

表 106. SERVICE_CALL_REQUEST 呼び出しコンテキストのアクション

汎用ビジネス・オブジェクトの動詞	Maintain Child Verb 関数ブロックが実行するアクション
Create、Delete、Retrieve	ソース・オブジェクト内のすべての子オブジェクトの動詞を親ソース・オブジェクト内の動詞に設定します。このアクションにより、子オブジェクト内のすべての既存の動詞が上書きされます。

表 106. SERVICE_CALL_REQUEST 呼び出しコンテキストのアクション (続き)

汎用ビジネス・オブジェクトの動詞	Maintain Child Verb 関数ブロックが実行するアクション
Update	1. 指定された汎用ビジネス・オブジェクトのキー値の子関係表から、関係インスタンスを検索します。 2. 表参照の成功に基づいて子オブジェクトの動詞を設定します。 • 子オブジェクトの関係インスタンスが存在する場合、子オブジェクトの動詞を Update に設定します。 • この子オブジェクトの関係インスタンスが存在しない 場合、子オブジェクトの動詞を Create に設定します。

SERVICE_CALL_RESPONSE 呼び出しコンテキスト: 呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換されます。インバウンド・マップはアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。

Maintain Child Verb 関数ブロックの振る舞いは、対応するメソッドの、最後から 2 番目に指定されるパラメーターに応じて決まります。このパラメーターは *to_Retrieve* boolean 値フラグで、その値は、表 107 に示すように、コラボレーション要求を処理するときに子オブジェクトの動詞をアプリケーションでリセットするかまたは保持するかどうかを示します。

表 107. コネクターの振る舞い

to_Retrieve フラグの値	コネクターの振る舞い
true	コネクターにより、子オブジェクトの動詞がアプリケーションに入力された動詞とは別の値に設定されます。 例えば、ビジネス・オブジェクトはコネクターに親動詞の Update と子動詞の Create で入力された場合、アプリケーションが操作を完了後、コネクターによりすべての子オブジェクト動詞がその親の値にリセットされることがあります。この場合、子動詞は Update に変更されます。
false	コネクターにより子オブジェクト動詞は保持されます。 例えば、ビジネス・オブジェクトがコネクターに親動詞が Update、子動詞が Create で入力された場合、コネクターによりすべての子オブジェクト動詞が保持されます。この場合、子動詞は Create のままです。

注: Maintain Child Verb 関数ブロックに対応する Java コードは、SERVICE_CALL_RESPONSE 呼び出しコンテキストを処理する場合のみ、*to_Retrieve* パラメーターの値を使用します。

引き数 *to_Retrieve* が true の場合、Maintain Child Verb 関数ブロックは、表 108 に示すタスクを実行します。

表 108. SERVICE_CALL_RESPONSE 呼び出しコンテキストのアクション

汎用ビジネス・オブジェクトの動詞	Maintain Child Verb 関数ブロックが実行するアクション
Create、Delete、Retrieve、Update	ソース・オブジェクト内のすべての子オブジェクトの動詞を親ソース・オブジェクト内の動詞に設定します。このアクションにより、子オブジェクト内のすべての既存の動詞が上書きされます。 1. 子関係表内の各子オブジェクトを参照します。 2. 表参照の成功に基づいて子オブジェクトの動詞を設定します。 • 子オブジェクトの関係インスタンスが存在する場合、子オブジェクトの動詞を Update に設定します。 • この子オブジェクトの関係インスタンスが存在しない 場合、子オブジェクトの動詞を Create に設定します。

注: アプリケーションの振る舞いが不明の場合は、*to_Retrieve* 引き数を true に設定します。このようにフラグ値 true を設定すると、パフォーマンスに影響する可能性があります。これは、Maintain Child Verb 関数ブロックに対応する Java コードによって、不要な参照が実行される場合があるからです。しかし、子オブジェクトに誤った動詞が設定されるよりも不要な参照の方が、通常は安全です。

Maintain Child Verb 関数ブロックの使用についてのヒント

Maintain Child Verb 関数ブロックは、ソース・ビジネス・オブジェクトの子オブジェクトの動詞を維持します。また、単純または複合一致関係の一部である子オブジェクトを処理することができます。この関数ブロックは、親ソース・オブジェクトの動詞と呼び出しコンテキストが与えられると、それに応じて動詞を適切に設定します。

Maintain Child Verb 関数ブロックを使用する場合は、以下のヒントに留意してください。

- メソッドの 2 番目から最後のパラメーターは、*to_Retrieve* boolean 値フラグで、アプリケーションで子オブジェクトの動詞をリセットするか保持するかどうかを示します。

  to_Retrieve フラグの設定方法の詳細については、316 ページの『SERVICE_CALL_RESPONSE 呼び出しコンテキスト』を参照してください。
- このメソッドの最後のパラメーターは boolean 値フラグの *is_Composite* で、子オブジェクトが単純または複合一致関係の一部であるかどうかを示します。

  子ビジネス・オブジェクトのキー属性は、次の種類の一致関係のいずれかに参加できます。

  – 単純一致関係内の固有キーとして。

    is_Composite フラグの値を false に設定します。

  – 複合一致関係内の複合キーの非固有キーとして。この場合、複合キーの他の部分は親ビジネス・オブジェクト内の固有キーです。

    is_Composite フラグの値を true に設定します。

- Maintain Child Verb 関数ブロックは、サブマップを呼び出す*前*に、ソース親マップの子属性で使用するようにします。

  子オブジェクトが複数カーディナリティーの子オブジェクトである場合は、`for` ループの開始の*直前* に Maintain Child Verb 関数ブロックを使用します。対応するメソッドによって子オブジェクトが繰り返し処理され、子の動詞が正しく設定されます。

外部キー参照の実行

外部キー とは、別のビジネス・オブジェクトのキー値を含む 1 つのビジネス・オブジェクト内の属性です。このキー値は、一部の他のビジネス・オブジェクトを識別するので、ソース・ビジネス・オブジェクトには「外部」と見なされます。ソース・ビジネス・オブジェクト内で外部キーを変換するには、外部キーが参照するビジネス・オブジェクトと関連する関係表 (外部関係表) にアクセスする必要があります。この外部関係表から、宛先ビジネス・オブジェクトの外部キーに関連するキー値を取得できます。

マッピング API には、外部キー参照の実行のため表 109 に示すメソッドが用意されています。

表 109. 外部キー参照のための関数ブロック

関数ブロック	説明
General/APIs/Identity Relationship/Foreign Key Lookup	外部キー参照を実行し、外部キーが外部関係表に存在しない場合は、関係インスタンスの検出を失敗します。
General/APIs/Identity Relationship/Foreign Key Cross-Reference	外部キー参照を実行し、外部キーが存在しない場合は、外部関係表に新規の関係インスタンスを追加します。

Foreign Key Lookup 関数ブロックの使用

Foreign Key Lookup 関数ブロックに対応する Java コードは、外部関係表で、ソース・ビジネス・オブジェクトの外部キーを検索します。この関数ブロックは、以下のアクションを実行します。

1. アプリケーション固有の参加者に、複合キーではなく、単一キーが含まれていることを検証します。

   アプリケーション固有のビジネス・オブジェクトである、アプリケーション固有の参加者の参加者タイプを判別します。また、このビジネス・オブジェクトに存在するキー属性が 1 つだけであることを確認します。複数のキー属性が存在する場合、Foreign Key Lookup 関数ブロックは、それらのアプリケーション固有のキー属性のうちのどれに、汎用ビジネス・オブジェクトの外部キーに対応するアプリケーション固有のキーを取り込めばよいかを判断できません。したがって、`RelationshipRuntimeException` 例外がスローされます。
2. 汎用ビジネス・オブジェクト内の外部キーの値と一致する外部関係表内の関係インスタンスを検索します。
3. 検索した関係インスタンスからアプリケーション固有のキー値を取得します。

4. アプリケーション固有のキー値をアプリケーション固有のビジネス・オブジェクトの外部キーにコピーします。

Foreign Key Lookup 関数ブロックに対応する Java コードは、ソース・ビジネス・オブジェクトの動詞に関係なく、これらのアクションを外部関係表で実行します。

Foreign Key Cross-Reference 関数ブロックの使用

Foreign Key Lookup 関数ブロックと同様に、Foreign Key Cross-Reference 関数ブロックも、ソース・ビジネス・オブジェクトの外部キーに基づいて外部関係表で参照を実行します。ただし、Foreign Key Cross-Reference 関数ブロックには、参照が失敗した場合に関係表にエントリーを追加できるという追加の機能が備わっています。次以降のセクションでは、各呼び出しコンテキストでの Foreign Key Cross-Reference 関数ブロックの振る舞いについて説明します。

EVENT_DELIVERY、ACCESS_REQUEST、および SERVICE_CALL_RESPONSE 呼び出しコンテキスト

呼び出しコンテキストが EVENT_DELIVERY、ACCESS_REQUEST、または SERVICE_CALL_RESPONSE の場合、呼び出されるマップはインバウンド・マップです。つまり、アプリケーション固有のビジネス・オブジェクトが汎用ビジネス・オブジェクトに変換されます。インバウンド・マップはアプリケーション固有のビジネス・オブジェクトを入力として受け取り、汎用ビジネス・オブジェクトを出力として戻します。したがって、Foreign Key Cross-Reference 関数ブロックのタスクは、特定のアプリケーション固有のキー値に対応する汎用キーを、外部参照表から取得することです。

呼び出しコンテキストが EVENT_DELIVERY、ACCESS_REQUEST、または SERVICE_CALL_RESPONSE の場合、Foreign Key Cross-Reference 関数ブロックは以下のアクションを実行します。

1. 汎用参加者に、複合キーではなく、単一キーが含まれていることを検証します。

   汎用ビジネス・オブジェクトである、汎用参加者の参加者タイプを判別します。また、このビジネス・オブジェクトに存在するキー属性が 1 つだけであることを確認します。複数のキー属性が存在する場合、Foreign Key Cross-Reference 関数ブロックは、それらの汎用キー属性のうちのどれに、アプリケーション固有のビジネス・オブジェクトの外部キーに対応する汎用キーを取り込めばよいかを判断できません。したがって、RelationshipRuntimeException 例外がスローされます。
2. アプリケーション固有のビジネス・オブジェクトの外部キー値と一致する関係インスタンスを外部関係表で検索します。表 110 に、Foreign Key Cross-Reference 関数ブロックがアプリケーション固有のビジネス・オブジェクトの動詞に基づいて外部関係表で実行するアクションを示します。
3. 検索した関係インスタンスからインスタンス ID を取得します。

4. インスタンス ID を汎用ビジネス・オブジェクトの外部キーにコピーします。

表 110. EVENT_DELIVERY、ACCESS_REQUEST、および SERVICE_CALL_RESPONSE 呼び出しコンテキストのアクション

アプリケーション固有のビジネス・オブジェクトの動詞	Foreign Key Cross-Reference 関数ブロックが実行するアクション
Create	EVENT_DELIVERY と ACCESS_REQUEST 呼び出しコンテキストの場合、アプリケーション固有のビジネス・オブジェクトのキー値について外部関係表に新規の関係エントリーを挿入します。 SERVICE_CALL_RESPONSE 呼び出しコンテキストの場合、アプリケーション固有のビジネス・オブジェクトのキー値とその関連する関係インスタンスを含む新規の関係エントリーを関係表に挿入します。このメソッドは、マップの実行コンテキスト (cwExecCtx) 内のオリジナル要求ビジネス・オブジェクトから関係インスタンス ID を取得します。SERVICE_CALL_RESPONSE の振る舞いの詳細については、294 ページの『SERVICE_CALL_RESPONSE 呼び出しコンテキスト』を参照してください。 このキー値のエントリーがすでに存在する場合は既存のエントリーを検索します。表にさらにエントリーを追加しないでください。
Update	指定されたアプリケーション固有のビジネス・オブジェクトの外部キー値の外部関係表から、関係エントリーを検索します。 この外部キー値のエントリーが存在しない 場合は、アプリケーション固有のビジネス・オブジェクトの外部キー値の外部関係表に新規の関係エントリーを挿入します。
Retrieve	指定されたアプリケーション固有のビジネス・オブジェクトの外部キー値の外部関係表から、関係エントリーを検索します。

呼び出しコンテキストが EVENT_DELIVERY 、ACCESS_REQUEST、または SERVICE_CALL_RESPONSE で、アプリケーション固有のビジネス・オブジェクト (App Obj A) の動詞が Create または Update である場合に、Foreign Key Cross-Reference 関数ブロックが外部関係表 (App Obj C 用) に対してどのようなアクセスを行うかを、図 135 に示します。

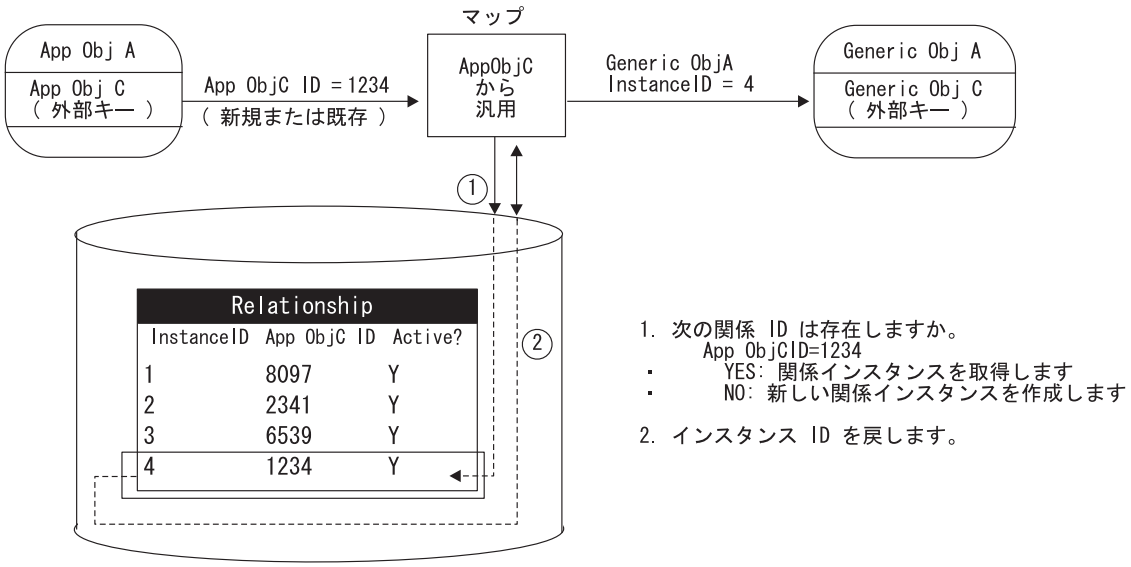

図 135. Create または Update 動詞についての外部キー参照

注: Foreign Key Cross-Reference 関数ブロックが関係インスタンスを追加するのは、インバウンド・マップの外部関係表だけです。

SERVICE_CALL_REQUEST 呼び出しコンテキストと外部キー

呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、呼び出されるマップはアウトバウンド・マップです。つまり、汎用ビジネス・オブジェクトがアプリケーション固有のビジネス・オブジェクトに変換されます。アウトバウンド・マップは汎用ビジネス・オブジェクトを入力として受け取り、アプリケーション固有のビジネス・オブジェクトを出力として戻します。呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、Foreign Key Cross-Reference 関数ブロックは、以下のアクションを実行します。

1. アプリケーション固有の参加者に、複合キーではなく、単一キーが含まれていることを検証します。

   アプリケーション固有のビジネス・オブジェクトである、アプリケーション固有の参加者の参加者タイプを判別します。また、このビジネス・オブジェクトに存在するキー属性が 1 つだけであることを確認します。複数のキー属性が存在する場合、Foreign Key Cross-Reference 関数ブロックは、それらのアプリケーション固有のキー属性のうちのどれに、汎用ビジネス・オブジェクトの外部キーに対応するアプリケーション固有のキーを取り込めばよいかを判断できません。したがって、RelationshipRuntimeException 例外がスローされます。
2. アプリケーション固有のビジネス・オブジェクトの動詞に基づいて、表 111 で説明するタスクが実行されます。

   Foreign Key Cross-Reference 関数ブロックは、動詞が Update、Delete、または Retrieve の場合に限り、特定の関係インスタンス ID に対応するアプリケーション固有のキー値を外部関係表から取得します。動詞が Create の場合、Foreign Key Cross-Reference 関数ブロックは、アプリケーション固有のキー値を取得しません。

表 111 に、Foreign Key Cross-Reference 関数ブロックが汎用ビジネス・オブジェクトの動詞に基づいて外部関係表で実行するアクションを示します。

表 111. SERVICE_CALL_REQUEST 呼び出しコンテキストと外部キーのアクション

汎用ビジネス・オブジェクトの動詞	Foreign Key Cross-Reference 関数ブロックが実行するアクション
Create	アクションは行いません。
Update、Delete、Retrieve	呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、このメソッドで新規の関係インスタンスが外部関係表に書き込まれます。詳細については、319 ページの『EVENT_DELIVERY、ACCESS_REQUEST、および SERVICE_CALL_RESPONSE 呼び出しコンテキスト』を参照してください。 1. マップの実行コンテキスト内のオリジナル要求ビジネス・オブジェクトから汎用ビジネス・オブジェクトのキー値 (関係インスタンス ID) を取得します。 2. 指定された汎用ビジネス・オブジェクトのキー値の外部関係表から、関係インスタンスを検索します。このキー値の関係インスタンスが存在しない 場合は、RelationshipRuntimeException 例外がスローされます。動詞が Retrieve で参加者が検出されない場合、CxMissingIDException 例外がスローされます。 3. 検索した関係インスタンスからアプリケーション固有のキー値を取得します。 4. アプリケーション固有のキー値をアプリケーション固有のビジネス・オブジェクトにコピーします。

表 111 からわかるように、動詞が Create の場合、Foreign Key Cross-Reference 関数ブロックは、関係表に新規の関係インスタンスを書き込みません。インスタンス ID に対応するアプリケーション固有の外部キー値をまだ持っていないので、この書き込み操作は実行されません。コネクターがアプリケーション固有のビジネス・オブジェクトを処理する場合、新規の行の挿入 (複数行の場合もある) が必要なことがアプリケーションに通知されます。この挿入に成功すると、アプリケーションからコネクターに通知され、コネクターにより Create 動詞とアプリケーションのキー値を使用して該当するアプリケーション固有のビジネス・オブジェクトが作成されます。

注: 呼び出しコンテキストが SERVICE_CALL_REQUEST の場合、Foreign Key Cross-Reference 関数ブロックは、Maintain Simple Identity Relationship 関数ブロックが関係表を管理するときと同様の方法で、外部関係表を管理します。

ACCESS_RESPONSE 呼び出しコンテキストと外部キー

呼び出しコンテキストが ACCESS_RESPONSE の場合、呼び出されるマップはアウトバウンド・マップです。つまり、汎用ビジネス・オブジェクトがアプリケーション固有のビジネス・オブジェクトに変換されます。アウトバウンド・マップは汎用ビジネス・オブジェクトを入力として受け取り、アプリケーション固有のビジネス・オブジェクトを出力として戻します。したがって、Foreign Key Cross-Reference 関数ブロックのタスクは、特定の汎用キー値に対応するアプリケーション固有のキーを、外部参照表から取得することです。

呼び出しコンテキストが ACCESS_RESPONSE の場合、Foreign Key Cross-Reference 関数ブロックは、以下のアクションを実行します。

1. アプリケーション固有の参加者に、複合キーではなく、単一キーが含まれていることを検証します。

アプリケーション固有のビジネス・オブジェクトである、アプリケーション固有の参加者の参加者タイプを判別します。また、このビジネス・オブジェクトに存在するキー属性が 1 つだけであることを確認します。複数のキー属性が存在する場合、Foreign Key Cross-Reference 関数ブロックは、それらのアプリケーション固有のキー属性のうちのどれに、汎用ビジネス・オブジェクトの外部キーに対応するアプリケーション固有のキーを取り込めばよいかを判断できません。したがって、`RelationshipRuntimeException` 例外がスローされます。

2. 汎用ビジネス・オブジェクト内の外部キーの値と一致する外部関係表内の関係インスタンスを検索します。
3. 検索した関係インスタンスからアプリケーション固有のキー値を取得します。
4. アプリケーション固有のキー値をアプリケーション固有のビジネス・オブジェクトの外部キーにコピーします。

Foreign Key Cross-Reference 関数ブロックは、汎用ビジネス・オブジェクトの動詞に関係なく、これらのアクションを外部関係表で実行します。

Foreign Key Cross-Reference 関数ブロックと Foreign Key Lookup 関数ブロックの使用についてのヒント

Foreign Key Cross-Reference 関数ブロックや Foreign Key Lookup 関数ブロックを使用する場合は、以下のヒントに留意してください。

- Foreign Key Lookup 関数ブロックまたは Foreign Key Cross-Reference 関数ブロックの呼び出しは、宛先ビジネス・オブジェクトの外部キー属性の変換ステップに含めます。
- Foreign Key Lookup 関数ブロックと Foreign Key Cross-Reference 関数ブロックは、複合キーを外部キーとしてサポートしません。
- Foreign Key Lookup 関数ブロックを使用した後は、宛先側の外部キー属性に null 値が含まれていない かどうかを検査します。外部キー値として取り込まれた null は、ソース・ビジネス・オブジェクトの外部キーに対応する外部キー値を、Foreign Key Lookup 関数ブロックが見つけられなかったことを示しています。この条件を示すために、メッセージ番号 5007 または 5008 (マップが強制的に失敗されたかどうかにより異なる) がログに記録され、必要により、`MapFailureException` 例外がスローされてマップが停止します。

  Foreign Key Cross-Reference 関数ブロックに関しては、使用後、この検査を行う必要はありません。この関数ブロックを使用すると、目的のアプリケーション固有のキー値が存在しない場合には、外部関係表にエントリーが自動的に追加されるからです。
- 子オブジェクト属性のいずれかで、(Maintain Simple Identity Relationship 関数ブロックや Maintain Composite Relationship 関数ブロックではなく) Foreign Key Cross-Reference 関数ブロックまたは Foreign Key Lookup 関数ブロックを使用する必要がある場合には、ソース親オブジェクトの動詞を子ビジネス・オブジェクトの動詞にする「移動」変換を定義することによって、ソース子オブジェクトの動詞を設定することができます。呼び出しは `for` *ループの内部* で、`runMap()` メソッドを呼び出す直前で行います。

カスタム関係の維持

マッピング API には、関係の参照や識別などの一部の基本的な関係について関係表の操作を処理するメソッドが用意されています。その他の関係については、参加者の追加および削除をユーザーがプログラミングしてカスタム関係を作成できます。

関係属性の変換コードのプログラミング方法は、マップの実行ごとに変動する幾つかの要因により異なります。通常は、発生する可能性のある各状況を処理する一連のケースとしてコードを構造化します。以下が、通常、考慮する必要のある要因です。

- ソース・ビジネス・オブジェクトに関連する動詞の Create、Retrieve、Update、または Delete。例えば、動詞が Create の場合、通常、マップでは新規の関係インスタンスが作成されるか、または既存の関係インスタンスに新規の関係インスタンスが追加されます。アプリケーション固有のビジネス・オブジェクトとそのコネクターでサポートされる各動詞をユーザーのコードで処理する必要があります。
- マップ・インスタンスに関連する呼び出しコンテキスト。呼び出しコンテキストは現行マップの実行の目的を示し、それぞれの動詞の処理方法に影響することがあります。例えば、動詞が Create で呼び出しコンテキストが EVENT_DELIVERY の場合、通常は新規の関係インスタンスを作成します。また呼び出しコンテキストが SERVICE_CALL_RESPONSE の場合、通常は参加者を関係インスタンスに追加します。呼び出しコンテキストの詳細については、207 ページの『マップの実行コンテキストの理解』を参照してください。
- マップに関連するコラボレーションに含まれるビジネス・ロジック。例えば、コラボレーションで 2 つのアプリケーション間のデータの同期を管理する場合、マップ開発者はコラボレーション開発者と連係してコラボレーションに含むビジネス・ロジックとマップで処理するビジネス・ロジックを決定する必要があります。

Relationship と IdentityRelationship クラスには、関係インスタンスを作成するメソッドと、参加者インスタンスを追加、削除、および更新するメソッドがあります。Participant クラスには、「set」および「get」メソッドが用意されていて、参加者インスタンスの各種プロパティーを指定および検索することができます。

次のセクションで、カスタム関係の管理方法について説明します。

- 『新規の関係インスタンスの作成』
- 325 ページの『参加者インスタンスの作成』
- 325 ページの『参加者インスタンスの削除』

新規の関係インスタンスの作成

新規の関係インスタンスを作成するには、create() または addMyChildren() メソッドを使用します。両メソッドとも 1 つの新規の参加者インスタンスで関係インスタンスを作成します。

新規の関係インスタンスを作成する最も一般的な例は、動詞が Create で呼び出しコンテキストが EVENT_DELIVERY (または ACCESS_REQUEST) の場合です。この場合、アプリケーションによりコラボレーションがサブスクライブする「作成」イベントが生成され、マップによりアプリケーション固有のビジネス・オブジェクトが汎用ビ

ジネス・オブジェクトに変換されます。新規の関係インスタンスを作成するには、関係定義名、追加する参加者の参加者定義名、および参加者に関連するデータを指定する必要があります。

次のコードは、顧客が Clarify アプリケーションに追加された後、CustIden と呼ばれる新規の関係インスタンスを作成します。Clarify を表す参加者定義は、ClarCust と呼ばれます。

```
instanceId = Relationship.create
("CustIden","ClarCust",appBusObj);
```

戻り値 instanceId は、新規の関係インスタンスのインスタンス ID です。

参加者インスタンスの作成

関係に関する新規の参加者インスタンスを作成するには、addParticipant() メソッドを使用します。新規の参加者インスタンスを作成するのは、通常、次の場合です。

- ソース・ビジネス・オブジェクトの動詞が Create で、呼び出しコンテキストが EVENT_DELIVERY。この場合、新規の関係インスタンスと新規の参加者インスタンスを同時に作成します。
- ソース・ビジネス・オブジェクトの動詞が Create で、呼び出しコンテキストが SERVICE_CALL_RESPONSE。この場合、既存の関係インスタンスに新規の参加者を追加します。

次のコードは、CustIden と呼ばれる関係に新規の参加者インスタンスを追加します。ここでは、動詞は Create、呼び出しコンテキストは SERVICE_CALL_RESPONSE と想定しています。relID 変数には、新規の参加者インスタンスを受け取るための関係インスタンスの ID が含まれます。

```
instanceId = Relationship.addParticipant("CustIden","SAPCust",
   relID, appBusObj);
```

参加者インスタンスの削除

関係インスタンスから参加者インスタンスを削除するには、表 112 にリストされている Relationship クラスのメソッドの 1 つを使用します。

表 112. 参加者インスタンスを削除するメソッド

Relationship メソッド	説明
deleteParticipant()	関係定義名、参加者定義名、および指定した参加者データと一致するすべての 参加者インスタンスを削除します。
deleteParticipantByInstance()	指定した特定の関係インスタンスから *1* つの 参加者を削除します。
deactivateParticipant()	deleteParticipant() と同じですが、参加者のレコードを関係表に残すことが異なります。
deactivateParticipantByInstance()	deleteParticipantByInstance() と同じですが、参加者のレコードを関係表に残すことが異なります。

ソース・ビジネス・オブジェクトの動詞が Delete で呼び出しコンテキストが EVENT_DELIVERY (または ACCESS_REQUEST) か SERVICE_CALL_RESPONSE の場合に、通常、参加者インスタンスを削除します。

例: 次のコードは、呼び出しコンテキストが SERVICE_CALL_RESPONSE で動詞が Delete であることを想定しています。このコードは、Clarify 顧客を表す参加者を CustIden 関係から削除します。

```
Relationship.deleteParticipant
("CustIden","ClarCust",appBusObj);
```

安全な関係コードの作成

推奨: 関係管理を必要とする属性に関して次の防御コーディング標準を使用してください。

- マッピング API のメソッドを呼び出す前に、ソース属性が null でないことを常に確認します。
- マッピング API のメソッドの呼び出しは常に try/catch ブロックの内部で行い、catch セクションの中で該当するエラー・メッセージを表示します。

null のソース属性の検査

表 113 内のいずれかのマッピング API を呼び出す*前*に、ソース属性が null でないことを確認してください。属性が null の場合、エラーがログに記録され、メソッドを呼び出しません。

表 113. null のソース属性の処理

マッピング API メソッド	ログに記録するエラー番号	マップの実行を停止しますか?
maintainSimpleIdentityRelationship()	5000	はい
foreignKeyXref()	5003	マッピング仕様で、マップの実行を停止するかどうかを指定します。
foreignKeyLookup()		

マップの実行を停止するには、MapFailureException 例外をスローします。

マッピング API メソッドからの例外の処理

表 114 のマッピング API メソッドから発生するすべての例外を確実にキャッチするには、マッピング API メソッドの呼び出しを try/catch ブロックの内部で行い、catch セクションの中で該当するエラー・メッセージをログに記録します。

表 114. マッピング API メソッドからの例外の処理

マッピング API メソッド	キャッチする例外	ログに記録するエラー番号
maintainSimpleIdentity Relationship()	RelationshipRuntimeException	5001
maintainComposite Relationship()	CxMissingIDException	5002

表 114. マッピング API メソッドからの例外の処理 (続き)

マッピング API メソッド	キャッチする例外	ログに記録するエラー番号
foreignKeyLookup()	RelationshipRuntimeException	5007 または 5008
foreignKeyXref()	RelationshipRuntimeException	5009

例: 次のコード・フラグメントには、RelationshipRuntimeException および CxMissingIDException 例外の両方をキャッチし、情報メッセージをログに記録してサーバーにより生成されたエラー・テキストを表示し、さらに MapFailureException をスローしてマップの実行を停止する maintainSimpleIdentityRelationship() メソッドの呼び出しが組み込まれています。

```
try
   {
   // API call
   IdentityRelationship.maintainSimpleIdentityRelationship(...);
   }

catch (RelationshipRuntimeException re)
   {
   logError(5001);
   logInfo(re.toString());
   throw new MapFailureException("RelationshipRuntimeException");
   }

catch (CxMissingIDException ce)
   {
   logError(5002);
   logInfo(ce.toString());
   throw new MapFailureException("RelationshipRuntimeException");
   }
```

例: 次のコード・フラグメントは、RelationshipRuntimeException 例外をキャッチし、情報メッセージをログに記録してサーバーにより生成されたエラー・テキストを表示して、その後、宛先属性が正常にマップされたかどうかを検査する foreignKeyLookup() メソッドの例外処理を示します。宛先属性が正常にマップされない場合、このコード・フラグメントでは、マップが実行を停止する必要がある場合はエラー 5007、マップが実行を継続できる場合は エラー 5008 を表示します。

```
try
   {
   // API call
   IdentityRelationship.foreignKeyLookup(...);
   }

catch (RelationshipRuntimeException re)
   {
   logInfo(re.toString());
   }

if (ObjDest.isNull("DestAttr")
   {
   logError(5007, "DestAttrName", "SrcAttrName", "RelationshipName",
      "ParticipantName", strInitiator);
   throw new MapFailureException("foreignKeyLookup() failed");
   }
If the map execution is to be continued, use the following if statement:
```

```
if (ObjDest.isNull("DestAttr")
   {
   logError(5008, "DestAttrName", "SrcAttrName", "RelationshipName",
      "ParticipantName", strInitiator);
   }
```

例: 次のコード・フラグメントは、RelationshipRuntimeException 例外をキャッチして、情報メッセージをログに記録してサーバーにより生成されたエラー・テキストを表示し、その後、宛先属性が正常にマップされたかどうかを検査する foreignKeyXref() メソッドの例外処理を示します。宛先属性が正常にマップされない場合、このコード・フラグメントでは、エラー・メッセージ 5009 を表示し、MapFailureException をスローしてマップの実行を停止します。

```
try
   {
   // API call
   IdentityRelationship.foreignKeyXref(...);
   }

catch (RelationshipRuntimeException re)
   {
   logInfo(re.toString());
   }

if (ObjDest.isNull("DestAttr")
   {
   logError(5009, "DestAttrName", "SrcAttrName", "RelationshipName",
      "ParticipantName", strInitiator);
   throw new MapFailureException("foreignKeyXref() failed");
   }
```

リレーションシップ・データベースの照会の実行

関係を使用する場合、関係定義についての情報を取得する必要がある場合があります。関係情報はリレーションシップ・データベースの特別な表に格納されます。関係についての情報を取得するには、その関係表を照会します。*照会* は、データベースに送信して実行させる、通常は SQL (Structured Query Language) ステートメント形式の要求です。

リレーションシップ・データベースに照会を実行するには、次の手順を行います。

1. リレーションシップ・データベースへの接続を開いて、DtpConnection オブジェクトを取得します。
2. DtpConnection オブジェクトを使用して、照会を実行し、リレーションシップ・データベース内のトランザクションを管理します。

マップの実行が完了すると、接続は自動的にクローズします。

要確認: DtpConnection クラスとそのメソッドを使用してリレーションシップ・データベースへの接続を確立することは、後方互換性のためのみ にサポートされています。これらの*使用すべきでない*メソッド によりエラーが生成されることはありませんが、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。新規マップの開発では、CwDBStoredProcedureParam クラスとそのメソッドを使用して、データ

ベース接続を取得し、SQL 照会を実行します。詳細については、223 ページの『データベース照会の実行』を参照してください。

接続のオープン

リレーションシップ・データベースの照会を行うには、最初に BaseDLM クラスの getRelConnection() メソッドを使用してこのデータベースへの接続をオープンする必要があります。オープンするリレーションシップ・データベースを識別するには、照会を行う関係定義の名前を指定します。リポジトリーで各関係定義の関係表のロケーションを追跡します。詳細については、273 ページの『関係定義の拡張設定』を参照してください。

例: 次の getRelConnection() 呼び出しにより SapCust 関係の関係表を含むリレーションシップ・データベースがオープンします。

```
DtpConnection connection = getRelConnection("SapCust");
```

この呼び出しにより connection 変数に DtpConnection オブジェクトが戻され、その後この変数を使用してリレーションシップ・データベースにアクセスできます。

照会の実行

executeSQL() メソッドにより、実際の照会がリレーションシップ・データベースに送信されて実行されます。このセクションでは、以下の種類の SQL 照会の実行について説明します。

- 関係表からのデータを戻す照会 (SELECT)
- 関係表を変更する照会 (INSERT、UPDATE、DELETE)
- ストアード・プロシージャーを実行する照会

データを戻す照会 (SELECT)

SQL ステートメント SELECT は、1 つ以上の表のデータを照会します。SELECT ステートメントをリレーションシップ・データベースに送信して実行させるには、SELECT のストリング表記を引き数として executeSQL() メソッドに指定します。

注: DtpConnection.executeSQL() メソッドは、後方互換性のためにのみサポートされています。新規コードの開発ではこのメソッドを使用しないでください。代わりに、CwDBConnection クラスの executeSQL() メソッドを使用してください。

例: executeSQL() への次の呼び出しにより、RelRT_T 表からの 1 つの列値を取得する SELECT が送信されます。

```
connection.executeSQL(
   "select data from RelRT_T where INSTANCEID = 2");
```

注: 前述のコードでは、connection 変数は前の getRelConnection() メソッドの呼び出しで取得した DtpConnection オブジェクトです。

また、executeSQL() メソッドの 2 番目の形式を使用して、パラメーターを持つ SELECT ステートメントを送信することもできます。

例: 次のexecuteSQL() への呼び出しにより、前述の例と同じ操作が行われます。ただし、この場合、インスタンス ID がパラメーターとして SELECT ステートメントに渡されます。

```
Vector argValues = new Vector();

String instance_id = "2";
argValues.addElement( instance_id );
connection.executeSQL(
   "select data from RelRT_T where INSTANCEID = ?", argValues);
```

SELECT ステートメントでは、関係表からデータが行として戻されます。各行は、SELECT の条件と一致する指定された関係表の 1 行です。各行には、SELECT ステートメントが指定した列の値が含まれます。戻されたデータは、これらの行および列による 2 次元配列として表示できます。

ヒント: SELECT ステートメントの構文は、アクセスする特定のリレーションシップ・データベースに対して有効である必要があります。SELECT ステートメントの正確な構文については、データベース文書を確認してください。

戻されたデータにアクセスするには、次の手順を実行します。

1. 1 行のデータを取得します。
2. 列の値を 1 つずつ取得します。

表 115 に、戻された照会データにアクセスするための DtpConnection クラスのメソッドを示します。

表 115. 行にアクセスするための DtpConnection メソッド

行にアクセスするための操作	DtpConnection メソッド
行があるかどうかのチェック	hasMoreRows()
1 行のデータの取得	nextRow()

hasMoreRows() メソッドを使用して、戻された行のループを制御します。hasMoreRows() から false が戻されると、行ループが終了します。1 行のデータを取得するには、nextRow() メソッドを使用します。このメソッドは、選択された列の値を Java Vector オブジェクトの要素として戻します。これにより、Enumeration クラスを使用して、列の値に個別にアクセスできます。 Vector と Enumeration の両クラスは java.util パッケージに存在します。戻された照会行の列にアクセスするための Java メソッドについては、226 ページの表 71 を参照してください。

注: 照会結果の行にアクセスするメカニズムは、使用すべきでない DtpConnection クラスと、その置き換えである CwDBConnection クラスでは同じです。詳細については、225 ページの『データを戻す静的照会の実行 (SELECT)』を参照してください。

例: 以下のコード例では、sampleRelationshipName 関係定義を保管するリレーションシップ・データベースへ接続する DtpConnection クラスのインスタンスを取得します。次に、SELECT ステートメントが実行され、「CrossWorlds」の単一ストリング値が含まれる 1 行のみが戻されます。

```
Vector theRow = null;
Enumeration theRowEnum = null;
String theColumn1 = null;
```

```
DtpConnection connectn = null;

try
   {
   connectn = getRelConnection("sampleRelationshipName");
   }

catch(DtpConnectionException e)
   {
   System.out.println(e.getMessage());
   }

// Test for a resulting single column, single row, result set
// specified condition
try
   {
   connectn.executeSQL(
      "select data from RelRT_T where INSTANCEID = 2");

   // Loop through each row
   while(connectn.hasMoreRows())
      {
      theRow = connectn.nextRow();
      int length = 0;
      if ((length = theRow.size())!= 1)
         {
         return "Expected result set size = 1," +
            " Actual result state size = " + length;
         }

      // Get each column
      theRowEnum = theRow.elements();
      if(theRowEnum.hasMoreElements())
         {
         // Get the value
         theColumn1 = (String)theRowEnum.nextElement();
         if(theColumn1.equals("CrossWorlds")==false)
            {
            return "Expected result = CrossWorlds,"
               + " Resulting result = " + theColumn1;
            }
         }
      }
   }

catch(DtpConnectionException e)
   {
   System.out.println(e.getMessage());
   }
```

注: SELECT ステートメントでは、リレーションシップ・データベースの内容は変更されません。したがって、通常は SELECT ステートメントのトランザクション管理を行う必要はありません。

関係表を変更する照会

関係表のデータを変更する SQL ステートメントには、以下が含まれます。

- INSERT は、関係表に新規の行を追加します。
- UPDATE は、関係表の既存の行を変更します。
- DELETE は、関係表から行を除去します。

これらのステートメントのリレーションシップ・データベースに送信して実行させるには、executeSQL() メソッドの引き数として、ステートメントをストリング表記で指定します。

注: DtpConnection.executeSQL() メソッドは、後方互換性のためにのみサポートされています。新規コードの開発ではこのメソッドを使用しないでください。代わりに、CwDBConnection クラスの executeSQL() メソッドを使用してください。

例: 次のexecuteSQL() への呼び出しにより、RelRT_T 表への 1 行の INSERT が送信されます。

```
connection.executeSQL("insert into RelRT_T values
(1, 3, 6)");
```

注: 前述のコードでは、connection 変数は前の getRelConnection() メソッドの呼び出しで取得した DtpConnection オブジェクトです。

UPDATE または INSERT ステートメントの場合、getUpdateCount() メソッドを使用して、変更または追加された関係表の行数を判別できます。

INSERT、UPDATE、および DELETE ステートメントはリレーションシップ・データベースの内容を変更するため、これらのステートメントについては、トランザクション管理を行う必要があります。トランザクション は、1 単位として実行される操作ステップの集合です。トランザクション内で実行されるすべての SQL ステートメントは、1 単位として成功または失敗します。表 116 に、SQL 照会のトランザクション・サポートを行う DtpConnection クラスのメソッドを示します。

表 116. トランザクション管理のための DtpConnection メソッド

トランザクション管理操作	DtpConnection メソッド
新しいトランザクションを開始します。	beginTran()
トランザクションの実行時にデータベースに対して行われたすべての変更をコミット (保管) し、トランザクションを終了します。	commit()
トランザクションが現在アクティブかどうかを確認します。	inTransaction()
トランザクションの実行時に行われたすべての変更をロールバック (バックアウト) し、トランザクションを終了します。	rollBack()

リレーションシップ・データベース内のトランザクションの開始をマーク付けするには、beginTran() メソッドを使用します。この beginTran() 呼び出しとトランザクションの終了の間で、単位として成功または失敗となるすべての SQL ステートメントを実行します。以下のいずれの方法でも、トランザクションを終了できます。

- commit() を呼び出して、トランザクションを正常終了します。SQL ステートメントで行われたすべての変更は、リレーションシップ・データベースに保管 されます。
- rollBack() を呼び出して、トランザクションを異常終了します。SQL ステートメントで行われたすべての変更は、リレーションシップ・データベースからバックアウト されます。

トランザクションが失敗した原因となる条件を選択できます。失敗条件が満たされた場合は、条件をテストし、rollBack() を呼び出します。あるいは、commit() を呼び出して、トランザクションを正常終了します。

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();

// release the database connection
releaseRelConnection(true);
```

トランザクションが現在アクティブであるかどうかを判別するには、inTransaction() メソッドを使用します。

ストアード・プロシージャーを呼び出す照会

ストアード・プロシージャーは、SQL ステートメントおよび条件ロジックが含まれるユーザー定義のプロシージャーです。ストアード・プロシージャーは、データベースに格納されます。新しい関係を作成する場合、各関係表を保守するためにストアード・プロシージャーが作成されます。

表 117 に、ストアード・プロシージャーを呼び出す DtpConnection クラスのメソッドを示します。これらのメソッドは、後方互換性のためにのみサポートされています。新規コードの開発ではこのメソッドを使用しないでください。代わりに、CwDBConnection クラスの executeSQL() と executeStoredProcedure() メソッドを使用してください。

表 117. ストアード・プロシージャーを呼び出す DtpConnection メソッド

ストアード・プロシージャーの呼び出し方法	DtpConnection メソッド	使用
ストアード・プロシージャーを実行する CALL ステートメントをリレーションシップ・データベースに送信します。	executeSQL()	OUT パラメーターを持たない ストアード・プロシージャーを呼び出します。
ストアード・プロシージャーの名前およびそのパラメーターの配列を指定してプロシージャー呼び出しを作成します。これがリレーションシップ・データベースに送信されて実行されます。	execStoredProcedure()	OUT パラメーターがある任意のストアード・プロシージャーを呼び出すため。

注: JDBC メソッドを使用して、ストアード・プロシージャーを直接実行できます。しかし、マッピング API に用意されている単純なインターフェースを使用して、データベース・リソースを再使用し、実行効率を向上することができます。ストアード・プロシージャーはマッピング API を使用して実行する必要があります。

表 117 に示す、どのメソッドを使用してストアード・プロシージャーを呼び出すかは、以下によって決まります。

- プロシージャーに OUT パラメーターがあるかどうか。

  OUT パラメーターは、ストアード・プロシージャーがこれを介して値を呼び出しコードに戻すパラメーターです。ストアード・プロシージャーが OUT パラメーターを使用する場合、execStoredProcedure() を使用してストアード・プロシージャーを呼び出す必要があります。
- ストアード・プロシージャーを呼び出す回数。

  execStoredProcedure() メソッドにより、ストアード・プロシージャーはプリコンパイルされます。したがって、同じストアード・プロシージャーを複数回呼び出す場合 (例えば、ループ内) は、execStoredProcedure() を使用すると、リレーションシップ・データベースがプリコンパイル済みバージョンを使用できるため、executeSQL() より処理が速くなります。

次のセクションでは、executeSQL() や execStoredProcedure() メソッドを使用した、ストアード・プロシージャーの呼び出し方法を説明します。

executeSQL() によるストアード・プロシージャーの呼び出し: executeSQL() メソッドを使用してストアード・プロシージャーを呼び出すには、executeSQL() メソッドの引き数として、ストアード・プロシージャーとすべて引き数を含ませた CALL ステートメントをストリング表記で指定します。

例: 次の executeSQL() 呼び出しにより、setOrderCurrDate() ストアード・プロシージャーを実行する CALL ステートメントが送信されます。

```
connection.executeSQL("call setOrderCurrDate(345698)");
```

注: 前述のコードでは、connection 変数は前の getRelConnection() メソッドの呼び出しで取得した DtpConnection オブジェクトです。

setOrderCurrDate() ストアード・プロシージャーを実行できます。その単一引き数が IN パラメーターである、つまり、その値がストアード・プロシージャーにのみに送信されるためです。このストアード・プロシージャーは、どの OUT パラメーターも使用しません。

注: パラメーター配列を受け入れる executeSQL() の形式を使用して、パラメーター値を渡すことができます。ただし、executeSQL() を使用して OUT パラメーターを使用するストアード・プロシージャーを呼び出すことはできません。このようなストアード・プロシージャーを実行するには、execStoredProcedure() を使用する必要があります。

DtpConnection executeSQL メソッドを使用して、ODBC を介して Oracle ストアード PL/SQL オブジェクトを呼び出す場合は、匿名 PL/SQL ブロックを使用します。以下は、受け入れ可能なフォーマットを示します (ストアード・プロシージャー名は myproc です)。

```
executeSQL("begin myproc(...); end;");
```

execStoredProcedure() によるストアード・プロシージャーの呼び出し:
execStoredProcedure() メソッドを使用してストアード・プロシージャーを呼び出す手順は、以下のとおりです。

1. 実行するストアード・プロシージャーの名前をストリングで指定します。

2. UserStoredProcedureParam オブジェクトの Vector パラメーター配列を作成します。これによりパラメーター情報 (各パラメーターの名前、タイプ、および値など) が指定されます。

パラメーターは、ストアード・プロシージャーに対して、またはストアード・プロシージャーから送信できる値です。パラメーターのイン/アウト・タイプにより、ストアード・プロシージャーがパラメーター値を使用する方法が決まります。

- IN パラメーターは*入力専用* です。ストアード・プロシージャーはパラメーター値を入力として受け入れますが、呼び出し元コードに値を戻すためにこのパラメーターを使用しません。
- OUT パラメーターは*出力専用* です。ストアード・プロシージャーはパラメーター値を入力として解釈せずに、呼び出し元コードに値を戻すために使用します。
- IN/OUT パラメーターは*入力および出力用* です。ストアード・プロシージャーはその値を入力として受け入れ、値を戻すときにもそのパラメーターを使用します。

UserStoredProcedureParam オブジェクトは、ストアード・プロシージャーの単一のパラメーターについて記述します。表 118 に、UserStoredProcedureParam オブジェクトに含まれる情報と、このパラメーター情報を検索して設定するメソッドを示します。

表 118. UserStoredProcedureParam オブジェクト内のパラメーター情報

パラメーター情報	UserStoredProcedureParam メソッド
パラメーター名	getParamName(), setParamName()
パラメーター値	getParamValue(), setParamValue()
パラメーターのデータ型	
• Java Object の場合	getParamDataTypeJavaObj(), setParamDataTypeJavaObj()
• JDBC データ型の場合	getParamDataTypeJDBC(), setParamDataTypeJDBC()
パラメーターのイン/アウト・タイプ	getParamIOType(), setParamIOType()
パラメーター・インデックスの位置	getParamIndex(), setParamIndex()

ストアード・プロシージャーにパラメーターを渡す手順は、以下のとおりです ストアード・プロシージャーにパラメーターを渡すには、以下の手順に従います。

1. UserStoredProcedureParam オブジェクトを作成して、パラメーター情報を保持します。

   UserStoredProcedureParam() コンストラクターを使用して、新しい UserStoredProcedureParam オブジェクトを作成します。このコンストラクターに以下のパラメーター情報を渡すことにより、オブジェクトを初期化します。
   - パラメーター・インデックスにより、このパラメーターに対するストアード・プロシージャーの宣言内の位置が示されます。
   - パラメーターのデータ型は、通常はパラメーター値を保持する Java Object の名前です。

- パラメーター値は、パラメーターに割り当てる値が含まれる Java Object です。OUT パラメーターの場合、この値はダミー値にすることができますが、Object 型は、ストアード・プロシージャー宣言の OUT パラメーターのデータ型と一致する必要があります。
- イン/アウト・パラメーター・タイプにより、パラメーターが IN、INOUT、OUT のいずれであるかを指定します。
- パラメーター名により、ストリング名とパラメーターが関連付けされます。

2. 各ストアード・プロシージャーのパラメーターについて、ステップ 1 を繰り返します。
3. すべてのストアード・プロシージャーのパラメーターを保持するのに十分な要素を持つ Vector オブジェクトを作成します。
4. 初期化された UserStoredProcedureParam オブジェクトをパラメーター Vector オブジェクトに追加します。

   Vector クラスの addElement() メソッドを使用して、UserStoredProcedureParam オブジェクトを追加します。
5. すべての UserStoredProcedureParam オブジェクトを作成し、これらを Vector パラメーター配列に追加したら、このパラメーター配列を 2 番目の引き数として execStoredProcedure() メソッドに渡します。

   execStoredProcedure() メソッドにより、ストアード・プロシージャーとそのパラメーターがリレーションシップ・データベースへ送信され、実行されます。

注: execStoredProcedure() の最初の引き数は、実行するストアード・プロシージャーの名前です。

例: 以下のように、get_empno() ストアード・プロシージャーが定義されているとします。

```
create or replace procedure get_empno(emp_id IN number,
      emp_number OUT number) as
   begin
      select emp_no into emp_number
      from emp
      where emp_id = 1;
   end;
```

このストアード・プロシージャーには、次の 2 つのパラメーターがあります。

- 最初のパラメーター emp_id は、IN パラメーターです。

  このため、「IN」のイン/アウト・タイプ、およびストアード・プロシージャーに送信する適切なデータ型、名前、および値を使用して、その関連する UserStoredProcedureParam オブジェクトを初期化する必要があります。emp_id は SQL NUMBER 型 (整数値を保持する) として宣言されているので、このパラメーターのデータ型と値は 整数値 Integer を保持する Java Object です。
- 2 番目のパラメーター emp_number は、OUT パラメーターです。

  このパラメーターでは、空の UserStoredProcedureParam オブジェクトを作成して、ストアード・プロシージャーに送信します。このオブジェクトは、「OUT」のイン/アウト・タイプと適切なデータ型および名前を使用して初期化します。ただし、このパラメーターにはダミーの値を入力します。ストアード・プロシージ

ャーによる実行が終了したら、getParamValue() メソッドを使用してこの OUT パラメーターから戻された値を取得できます。

例: 以下は、execStoredProcedure() メソッドを使用して get_empno() ストアード・プロシージャーを実行する例です。

```
DtpConnection connectn = null;
try
   {
   // Get database connection
   connectn = getRelConnection("Customer");

   // Create parameter Vector
   Vector paramData = new Vector(2);

   // Construct the procedure name
   String sProcName = "get_empno";

   // Create IN parameter
   UserStoredProcedureParam arg_in = new UserStoredProcedureParam(
      1, "Integer", new Integer(6), "IN", "arg_in");

   // Create dummy argument for OUT parameter
   UserStoredProcedureParam arg_out = new UserStoredProcedureParam(
      2, "Integer", new Integer(0), "OUT", "arg_out");

   // Add these two parameters to the parameter Vector
   paramData.addElement(arg_in);
   paramData.addElement(arg_out);

   // Run get_empno() stored procedure
   connectn.execStoredProcedure(sProcName, paramData);

   // Get the result from the OUT parameter
   arg_out = (UserStoredProcedureParam) paramData.elementAt(1);
   Integer emp_number = (Integer) arg_out.getParamValue();
   }
```

ヒント: Vector オブジェクトはゼロから始まる配列で、一方 UserStoredProcedureParam オブジェクトは 1 から始まる配列としてインデックスが付けられます。前述のコードでは、このパラメーター配列は 1 から始まるので、OUT パラメーターはインデックス値 2 で UserStoredProcedureParam() コンストラクターに作成されます。しかし、Vector パラメーター配列のこの OUT パラメーターの値にアクセスするには、この Vector 配列はゼロから始まるので、elementAt() 呼び出しでインデックス値 1 を指定します。

ストアード・プロシージャーは、そのパラメーターを SQL データ型として処理します。SQL と Java のデータ型は同一でない ため、マッピング API ではこれらの 2 つのデータ型の間でパラメーター値を変換する必要があります。IN パラメーターの場合、マッピング API ではパラメーター値を Java Object からその SQL データ型に変換します。OUT パラメーターの場合、マッピング API では SQL データ型から Java Object に変換します。マッピング API では、次の 2 つのデータ型マッピング層を使用して、これらの 2 つのデータ型の間でパラメーター値を変換します。

- Java タイプから JDBC タイプ
- JDBC タイプから SQL データ型

マッピング API では、JDBC データ型を内部で使用して、ストアード・プロシージャーとの間で送受信されるパラメーター値を保持します。JDBC は、java.sql.Types クラスで汎用 SQL タイプの ID の集合を定義します。これらのタイプは、最も一般的に使用される SQL タイプを示します。また、JDBC には、JDBC タイプから Java データ型への標準マッピングも用意されています。例えば、通常、JDBC INTEGER は Java int タイプにマッピングされます。

IN (または INOUT) パラメーターを Java オブジェクトからそれに対応する JDBC にマップするために、マッピング API では 表 119 のマッピングを使用します。

表 119. Java オブジェクトから対応する JDBC データ型へのマッピング

Java オブジェクトから	JDBC データ型へ
String	CHAR、VARCHAR、または LONGVARCHAR
java.math.BigDecimal	NUMERIC
Boolean	BIT
Integer	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
byte[]	BINARY、VARBINARY、または LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
Clob	CLOB
Blob	BLOB
Array	ARRAY
Struct	STRUCT
Ref	REF

OUT (または INOUT) パラメーターを JDBC データ型からそれに対応する Java オブジェクトにマップするために、マッピング API では表 120 のマッピングを使用します。

表 120. JDBC データ型から Java オブジェクトへのマッピング

JDBC データ型から	Java オブジェクトへ
CHAR、VARCHAR、LONGVARCHAR	String
NUMERIC、DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float
FLOAT、DOUBLE	Double
BINARY、VARBINARY、または LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
STRUCT	Struct
REF	Ref

したがって、すべての UserStoredProcedureParam オブジェクトには、表 121 に示すように、そのデータ型の 2 つの表現が含まれます。

表 121. パラメーターのデータ型

パラメーターのデータ型	説明	UserStoredProcedureParam メソッド
Java Object	マップ変換コードでパラメーター値を保持するために使用されるデータ型	getParamDataTypeJavaObj(), setParamDataTypeJavaObj()
JDBC データ型	マッピング API でパラメーター値を保持するために内部で使用されるデータ型	getParamDataTypeJDBC(), setParamDataTypeJDBC()

表 121 の UserStoredProcedureParam メソッドを使用して、いずれかの形式のパラメーター・データ型にアクセスできます。ただし、次の理由から、Java Object データ型 (Integer、String、または Float など) を使用する必要があります。

- IN (および INOUT) パラメーターの場合、パラメーター値を Java Object として指定する必要があります。したがって、パラメーターのデータ型を Java Object として指定する方が整合性がとれます。
- execStoredProcedure() メソッドは、Vector パラメーター配列のパラメーターを送信します。Vector オブジェクトに含むことができるのは、Java Object 要素のみです。

get_empno() の emp_id パラメーターは、整数値を含む、NUMBER の SQL データ型で宣言されます。したがって、337 で開始するコード例では、emp_id パラメーター (インデックス位置 1 のパラメーター) の UserStoredProcedureParam() 呼び出しにより、次の 3 番目の引き数でその値が 6 に設定されます。

```
new Integer(6)
```

この呼び出しでは、次の 2 番目の引き数でパラメーターのタイプが同じ Java Object タイプにも設定されます。

```
"Integer"
```

接続のクローズ

リレーションシップ・データベースへの接続は、マップの実行が完了すると解放されます。マップが正常に実行されると、すべてのトランザクションは、明示的にコミットされていなくても、自動的にコミットされます。マップの実行が失敗すると (例えば、catch ステートメントによって処理されない例外がスローされる場合)、すべてのトランザクションは、明示的にロールバックされなくても、ロールバックされます。

関係のロードとアンロード

repos_copy ユーティリティーを使用すると、リポジトリーへの指定された関係定義のロードとそこからのアンロードを行うことができます。

注: また、repos_copy を使用して、リポジトリーとのマップ定義のロードおよびアンロードを行うこともできます。詳細については、87 ページの『InterChange Server Express からのマップのインポートとエクスポート』を参照してください。

関係定義のアンロード

repos_copy ユーティリティーを使用すると、-e オプションを指定してリポジトリー内の指定された関係定義をアンロードできます。「*関係リポジトリー・ファイル*」は、この repos_copy ユーティリティーがリポジトリーからテキスト (.jar) ファイルへ関係定義ファイルを抽出するときに作成するファイルです。

例: 次の repos_copy コマンドは、StateLk 関係定義を、WebSphereICS という InterChange Server Express のリポジトリーから関係リポジトリー・ファイルにアンロードします。

```
repos_copy -eRelationship:StateLk -oRL_StateLookup.jar
-sWebSphereICS -uadmin -pnull
```

重要: 関係はファースト・クラス・エンティティーではありません。したがって、そのネーム・スペースはファースト・クラス・エンティティーで分離されます。ファースト・クラス・エンティティーは同じ名前にできませんが、関係はファースト・クラス・エンティティー (例えば、ビジネス・オブジェクトやコラボレーション) と同じ名前にできます。しかし、関係定義の名前が既存のいずれかの ファースト・クラス・エンティティーと一致する場合は、repos_copy の -e オプションを使用して、その関係定義をアンロードまたはロードすることはできません。関係定義を含む、リポジトリー全体をロードおよびアンロードすることはできます。

いくつかの関係定義を 1 つの関係リポジトリー・ファイルにコピーできます。

例: StateLk と CustLkUp の両関係定義をコピーするには、次の repos_copy コマンドを使用します。

```
repos_copy -eRelationship:StateLk+Relationship:CustLkUp
-oRL_Lookup_Relationships.jar -sWebSphereICS -uadmin -
pnull
```

関係定義のロード

repos_copy を使用して、関係リポジトリー・ファイルから関係定義をリポジトリーにロードすることもできます。

例: 次の repos_copy コマンドは、StateLk 関係定義を WebSphereICS という InterChange Server Express のリポジトリーにロードします。

```
repos_copy -iRL_StateLookup.jar
-sWebSphereICS -uadmin -pnull
```

repos_copy ユーティリティーは、関係定義をロードするときに、次の検証を行います。

- ロードする関係定義のデータベース URL を検証します。
- 関係定義に関するすべての従属オブジェクトがリポジトリー内にすでに存在することを検証します。

repos_copy でこれらの検証を両方とも実行できない場合は、関係定義をロードできません。ただし、repos_copy にはこれらの検証を抑止または限定する特別なコマンド行オプションが用意されています。これについては、次のセクションで説明します。

データベース URL の検証

repos_copy ユーティリティーには、関係定義をリポジトリーにロードしやすくする -r オプションが用意されています。-r オプションにより repos_copy に、実行時スキーマを作成しないで、関係定義をリポジトリーに追加することが指示されます。repos_copy でリポジトリー全体をバックアップする場合 (-o オプションを使用)、作成されたリポジトリー・テキスト・ファイル内の情報の一部に関係定義が記述されます。その後 repos_copy を使用して (-r オプションを使用しない) このリポジトリー・テキスト・ファイルの内容を持つ別のリポジトリーをロードすると、repos_copy は関係定義をロードしようとしたときに、次の形式のエラーを生成することがあります。

```
Server error: An error occurred during the validation
of the runtime database connection information for
relationship definition Customer. The database URL
used is: jdbc:weblogic:mssqlserver4:Cwrelns312@CWDEV:1433.
The database login name used is: crossworlds.
The database type used is: W55s/wPE/l4=1.
Reason: SqlServer.
```

このエラーの原因は、repos_copy がリレーションシップ・データベースに関する URL を検証しようとしたことです。関係定義の一部は、リレーションシップ・データベースのデータベース URL です。

repos_copy でリレーションシップ・データベースを検出できない場合は、エラーが生成されて、リポジトリーのロードがロールバックされます。InterChange Server Express をバックアップして同じサーバー (同じリレーションシップ・データベース) に復元するだけの場合、-r オプションを指定する必要はありません。リレーションシップ・データベース URL を見つけることができるので、リレーションシップ・データベース URL の検証は成功します。したがって、リポジトリーのロード (関係定義を含む) は成功します。

しかし、マイグレーションのインポート・プロセスで、あるマシンから別のマシンにリポジトリー・データを移動する場合には、-r オプションは役に立ちます。リポジトリー・データの既存のリレーションシップ・データベースを検出できない環境で repos_copy コマンドを実行すると、repos_copy によりエラーが生成されます。この検証を抑止するには、リポジトリーをロードするときに、repos_copy の -r オプションを指定します。この検証を抑止すると、repos_copy はリポジトリーに関係定義を正常に追加できます。このメソッドはリレーションシップ・データベースのロケーションとして現行のリポジトリー・データベースを使用します。その後、Relationship Designer Express を使用して、各リレーションシップ・データベースの該当するロケーションをポイントするようにデータベース URL を変更します。

例: 次の repos_copy コマンドは、StateLk 関係定義をリポジトリーにロードし、データベース URL の検証を抑止します。

```
repos_copy -rStateLk -iRL_StateLookup.txt -sWebSphereICS -uadmin
-pnull
```

従属オブジェクトの検証

デフォルトでは、repos_copy により関係定義をロードするときに、すべての従属オブジェクトが存在するかどうか検証されます。例えば、関係に関連するすべてのビジネス・オブジェクトがリポジトリーに存在することを検査します。すべての従属オブジェクトが存在しない 場合は、repos_copy によりエラーが生成されて、リポジトリーのロードがロールバックされます。repos_copy コマンド・ウィンドウに、次のメッセージが表示されます。

```
Some of the participants for relationships were missing.
For more info, refer to InterChange Server Express log file.
```

第 3 部 マッピング API リファレンス

第 9 章 BaseDLM クラス

この章では、マップ・インスタンスのメソッドについて説明します。メソッドは、IBM WebSphere Business Integration Server Express 定義のクラス BaseDLM に定義されています。BaseDLM クラスは、すべてのマップ・インスタンスの基本クラスです。作成されたマップはすべて BaseDLM のサブクラスで、これらのメソッドを継承します。BaseDLM クラスは、エラー処理とマップ内でのデバッグ、およびデータベースへの接続を確立するためのユーティリティー・メソッドを提供します。このクラスのメソッドはすべて、クラス名を参照せずに呼び出すことができます。

表 122 に、BaseDLM クラスのメソッドについて要約します。

表 122. BaseDLM メソッドの要約

メソッド	説明	ページ
getDBConnection()	データベースへの接続を確立して、CwDBConnection オブジェクトを戻します。	345
getName()	現行マップの名前を検索します。	347
getRelConnection()	リレーションシップ・データベースへの接続を確立して、DtpConnection オブジェクトを戻します。	348
implicitDBTransactionBracketing()	マップ・インスタンスが取得する任意の接続に対して、マップ・インスタンスが使用するトランザクション・プログラミング・モデルを検索します。	349
isTraceEnabled()	指定したトレース・レベルとマップの現行のトレース・レベルを比較します。	350
logError()、logInfo()、logWarning()	InterChange Server ログ・ファイルにエラー、情報、または警告のメッセージを送信します。	350
raiseException()	例外を発生させます。	352
releaseRelConnection()	リレーションシップ・データベースへの接続を解放します。	354
trace()	トレース・メッセージを生成します。	355

getDBConnection()

データベースへの接続を確立して、CwDBConnection オブジェクトを戻します。

構文

```
CwDBConnection getDBConnection(String connectionPoolName)
CwDBConnection getDBConnection(String connectionPoolName,
   boolean implicitTransaction)
```

パラメーター

connectionPoolName
有効な接続プールの名前。接続がこの接続プールに指定されているデータベースに、メソッドは接続されます。

implicitTransaction
トランザクション・プログラミング・モデルを、接続に関連付けられたデータベースに対して使用することを指示する boolean 値です。次の値が有効です。

true	データベースは、暗黙的なトランザクション・ブラケットを使用します
false	データベースは、明示的なトランザクション・ブラケットを使用します

戻り値

CwDBConnection オブジェクトを戻します。

例外

CwDBConnectionFactoryException - データベース接続の確立中にエラーが発生した場合

注記

getDBConnection() メソッドは、*connectionPoolName* で指定された接続プールから接続を取得します。この接続によって、接続に関連付けられたデータベースへの照会および更新の実行が可能になります。個々の接続プールでのすべての接続は、同じデータベースに対して行われます。メソッドは CwDBConnection オブジェクトを戻して、それによってデータベースでの照会の実行およびトランザクションの管理が可能になります。詳しくは、CwDBConnection クラスのメソッドを参照してください。

デフォルトでは、暗黙的なトランザクション・ブラケットをすべての接続でトランザクション・プログラミング・モデルとして使用します。*特定の接続に対して* トランザクション・プログラミング・モデルを指定するには、getDBConnection() メソッドのオプションの *implicitTransaction* 引き数として目的のトランザクション・プログラミング・モデルを示す boolean 値を指定します。以下の getDBConnection() 呼び出しは、ConnPool 接続プールから取得された接続に明示的なトランザクション・ブラケットを指定します。

```
conn = getDBConnection("ConnPool",false);
```

マップ・インスタンスの実行が完了すると、接続が解除されます。 release() メソッドを使用して、接続を明示的に閉じることができます。接続が解除されたかどうかを、isActive() メソッドを使用して判断することができます。

例

次の例では、CustConnPool 接続プールにおいて、接続に関連付けられたデータベースへの接続を確立します。その後で、接続は暗黙的なトランザクションを使用し、データベース表に行を挿入および更新します。

```
CwDBConnection connection = getDBConnection("CustConnPool");

// Insert a row
```

```
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");
```

getDBConnection() に先行する呼び出しに、オプションの第 2 引き数は含まれません。そのため、この接続は、トランザクション・プログラミング・モデルとして、暗黙的なトランザクション・ブラケットを使用します (トランザクション・プログラミング・モデルが「マップ・プロパティー」ダイアログでオーバーライドされる場合を除く)。したがって、この接続は、beginTransaction()、commit()、および rollback() で明示的なトランザクション境界を指定しません。実際に、これらのトランザクション・メソッドの 1 つを暗黙的なトランザクション・ブラケットで呼び出そうとすると、CwDBTransactionException 例外が生じます。

注: implicitDBTransactionBracketing() メソッドを使用して、現行のトランザクション・プログラミング・モデルをチェックすることができます。

次の例でも、CustConnPool 接続プールにおいて、接続に関連付けられたデータベースへの接続を確立します。しかし、この例では接続の明示的なトランザクション・ブラケットの使用が指定されます。その結果、次のような明示的なトランザクションを使用してデータベース表に行を挿入する、あるいはデータベース表の行を更新します。

```
CwDBConnection connection = getDBConnection("CustConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

getDBConnection() に先行する呼び出しには、オプションの *implicitTransaction* 引き数が含まれています。そのため、トランザクション・プログラミング・モデルが明示的なトランザクション・ブラケットに設定されます。したがって、この例では明示的なトランザクション呼び出しを使用して、トランザクションの境界を示します。これらのトランザクション・メソッドが省略されると、InterChange Server Express はそのトランザクションを暗黙的なものと見なして処理します。

参照項目

『第 12 章 CwDBConnection クラス』, implicitDBTransactionBracketing(), isActive(), release()

---

getName()

現行マップの名前を検索します。

構文

```
String getName()
```

パラメーター

なし。

戻り値

なし。

例外

なし。

例

次の例では、現行マップ・オブジェクト名を取得して、情報メッセージを記録します。

```
String mapName = getName();
logInfo(mapName + " is starting");
```

---

getRelConnection()

リレーションシップ・データベースへの接続を確立して、DtpConnection オブジェクトを戻します。

構文

```
DtpConnection getRelConnection(String relDefName)
```

パラメーター

relDefName　関係定義名。この関係定義の関係表を含むデータベースに、メソッドは接続されます。

戻り値

DtpConnection オブジェクトを戻します。

例外

DtpConnectionException - データベース接続の確立中にエラーが発生した場合

注記

このメソッドは、*relDefName* 関係で使用される関係表を含むデータベースへの接続を確立して、リレーションシップ・データベースへの照会と更新の実行を可能にします。メソッドは DtpConnection オブジェクトを戻して、それによって照会の実行およびトランザクションの管理が可能になります。詳しくは、DtpConnection クラスのメソッドを参照してください。

マップの実行が完了すると、接続が解除されます。 releaseRelConnection() メソッドを使用して、接続を明示的に閉じることができます。

例

次の例では、SapCust 関係の関係表を含むデータベースへの接続を確立します。接続が確立されると、トランザクションを使用して、SapCust 関係の中の表に行を挿入する照会を実行します。

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// update rows...
connection.executeSQL("update...");

// commit the transaction
connection.commit();
```

参照項目

getDBConnection(), 『第 14 章 DtpConnection クラス』, releaseRelConnection()

---

implicitDBTransactionBracketing()

マップ・インスタンスが取得する任意の接続に対して、マップ・インスタンスが使用するトランザクション・プログラミング・モデルを検索します。

構文

```
boolean implicitDBTransactionBracketing()
```

パラメーター

なし。

戻り値

すべてのデータベース接続で使用されるトランザクション・プログラミング・モデルを示す boolean 値。

注記

implicitDBTransactionBracketing() メソッドは以下のような boolean 値を戻します。この値は、マップ・インスタンスが前提とするトランザクション・プログラミング・モデルのうち、いずれのモデルを使用してすべての 接続を取得するかを指示します。

- true の値は、すべての接続が*暗黙的な* トランザクション・ブラケットを使用することを示します。
- false の値は、すべての接続が*明示的な* トランザクション・ブラケットを使用することを示します。

現行のトランザクション・プログラミング・モデルが適切かどうかを確認するには、接続を取得する前にこのメソッドを使用することです。

注: getDBConnection() メソッドを使用して、特定の接続のトランザクション・プログラミング・モデルをオーバーライドすることができます。

例

次の例では、マップ・インスタンスが、conn 接続と関連したデータベースに明示的なトランザクション・ブラケットを使用していることを示しています。

```
if (implicitDBTransactionBracketing())
   CwDBConnection conn = getDBConnection("ConnPool", false);
```

参照項目

getDBConnection()

---

isTraceEnabled()

指定したトレース・レベルとマップの現行のトレース・レベルを比較します。

構文

```
Boolean isTraceEnabled(int traceLevel)
```

パラメーター

traceLevel　　現行のトレース・レベルと比較されるトレース・レベル。

戻り値

指定されたトレース・レベルに現行システムのトレース・レベルが設定されると、true を戻します。2 つのトレース・レベルが異なる場合、false を戻します。

注記

isTraceEnabled() メソッドは、トレース・メッセージを記録するかを決定するときに有効です。トレースによってパフォーマンスが低下することがあるため、このメソッドはプロジェクトの開発段階において有効です。

例

```
if ( isTraceEnabled(3) )
   {
   trace("Print this level-3 trace message");
   }
```

---

logError()、logInfo()、logWarning()

InterChange Server ログ・ファイルにエラー、情報、または警告のメッセージを送信します。

構文

```
void logError(String message)
void logError(int messageNum)
void logError(int messageNum, String param [,...])
void logError(int messageNum, Object[] paramArray)

void logInfo(String message)
void logInfo(int messageNum)
void logInfo(int messageNum, String param [,...])
void logInfo(int messageNum, Object[] paramArray)

void logWarning(String message)
void logWarning(int messageNum)
void logWarning(int messageNum, String param [,...])
void logWarning(int messageNum, Object[] paramArray)
```

パラメーター

message メッセージ・テキスト。

messageNum メッセージ・テキスト・ファイル内のメッセージの番号です。

param 単一のパラメーター。コンマで区切られた、最大 5 つのパラメーターで構成されます。各パラメーターは順次的に解決され、メッセージ・テキスト内のパラメーターになります。

paramArray パラメーターの配列。

戻り値

なし。

例外

なし。

注記

このメソッドは、InterChange Server Express のログ宛先にメッセージを送信します。ログ宛先は、ファイル、ウィンドウ、またはその両方にすることができます。

デフォルトのログ宛先は、ファイル InterchangeSystem.log です。ユーザーは、構成ファイル InterchangeSystem.cfg 内の LOG_FILE のパラメーター値を入力してログ宛先を変更することができます。パラメーター値は、ファイル名、STDOUT (サーバーのコマンド・ウィンドウにログを書き込みます)、またはその両方にすることができます。

メソッドの設定内容は、次のようになります。

- 第 1 の形式は自己完結型で、メッセージの生成に必要なテキストをすべて組み込みます。
- 第 2 の形式は、パラメーターを持たないメッセージを生成します。
- 第 3 の形式は、メッセージ番号とパラメーター値のセットを含みます。
- 第 4 の形式は、パラメーターの配列を使用します。

messageNum パラメーターを取得するメソッドのすべての形式で、メッセージ・ファイルを使用する必要があります。このファイルはメッセージ番号でインデックスが

付けられます。メッセージ・テキスト・ファイルの設定方法の詳細は、537 ページの『付録 A. メッセージ・ファイル』を参照してください。

例

次の例では、getString() を使用してメッセージに記録する属性値を取得し、情報メッセージを記録します。

```
logInfo("Item shipped. CustomerID: "
   + fromCustomerBusObj.getString("CustomerID"));
```

次の例では、マップ・メッセージ・ファイルにテキストが格納されるエラー・メッセージを記録します。メッセージは、メッセージ・ファイルの 10 番で、顧客のラストネーム (LName 属性) と顧客のファーストネーム (FName 属性) という 2 つのパラメーターを取得します。

```
logError(10, customer.get("LName"), customer.get("FName");
```

次の例では、パラメーターの配列を使用してエラー・メッセージを記録します。説明用に、この例ではパラメーターが 2 つのみの配列を使用します。この例では、顧客 ID と顧客名の 2 つの要素を持つ配列 args が宣言されます。それから、12 番のメッセージおよび args 配列内の値を使用して、logError() メソッドはエラーを記録します。

```
Object[] args = {
      fromCustomerBusObj.getString("CustomerID"),
      fromCustomerBusObj.getString("CustomerName");
}

logError(12, args);
```

参照項目

trace()

---

raiseException()

例外を発生させます。

構文

```
void raiseException(String exceptionType, String message)

void raiseException(String exceptionType, int messageNum,
   String parameter[,...])

void raiseException(RunTimeEntityException exception)
```

パラメーター

exceptionType 以下の IBM WebSphere Business Integration Server Express 定義の定数の 1 つです。

AnyException	不特定の例外です。
AttributeException	属性でアクセスする場合の問題です。例えば、String 属性に対して getDouble() を呼び出したり、存在

しない属性に対して getString() を呼び出すコラボレーションがあります。

JavaException	IBM WebSphere Business Integration Server Express API の一部ではない Java コードに関する問題です。
ObjectException	メソッドに渡されたビジネス・オブジェクトが無効でした。または、null オブジェクトにアクセスされました。
OperationException	サービス呼び出しが適切に設定されませんでした。あるいは送信されませんでした。
ServiceCallException	サービス呼び出しに失敗しました。例えば、コネクターやアプリケーションは使用できません。
SystemException	IBM WebSphere Business Integration Server Express システム内の内部エラーです。

message	メソッドの呼び出しに例外メッセージを組み込むテキスト・ストリング。
messageNum	マップ・メッセージ・ファイル内の番号付きメッセージへの参照。
parameters	メッセージ自体のパラメーターの値。メソッドの呼び出しにおいて、最大 5 つのパラメーターで構成されます。
exception	例外オブジェクト変数の名前。

戻り値

なし。

注記

raiseException() メソッドには、以下の 3 つの形式があります。

- メソッドの第 1 の形式は、例外タイプおよびストリングを渡す新しい例外を作成します。この例外を使用して、メッセージをメソッドの呼び出し自体に組み込みます。
- 第 2 の形式は、例外タイプおよびマップ・メッセージ・ファイルに含まれるメッセージへの参照を渡す新しい例外を作成します。メソッドの呼び出しには、コンマで区切られた、最大 5 つのパラメーターを含めることができます。
- 第 3 の形式は、マップが以前に処理した例外オブジェクトを生成します。例えば、変換ステップは例外を取得して、変数に割り当ててから、その他の作業を実行することがあります。最後に、変換ステップは例外を生成します。

注: *messageNum* パラメーターを取得するメソッドのすべての形式で、メッセージ・ファイルを使用する必要があります。このファイルはメッセージ番号でインデ

ックスが付けられます。メッセージ・テキスト・ファイルの設定方法の詳細は、537 ページの『付録 A. メッセージ・ファイル』を参照してください。

例

次の例では、メソッドの第 1 の形式を使用して ServiceCallException タイプの例外を生成します。テキストはメソッドの呼び出しに組み込まれます。

```
raiseException(ServiceCallException,
   "Attempt to validate Customer failed.");
```

次の例は、ServiceCallException タイプの例外を生成します。メッセージ・ファイル内のメッセージは、以下のとおりです。

```
23
Customer update failed for CustomerID={1} CustomerName={2}
```

raiseException() メソッドはメッセージを呼び出して、fromCustomer 変数からメッセージ・パラメーターの値を取得し、raiseException() 呼び出しに渡します。

```
raiseException(ServiceCallException, 23,
   fromCustomer.getString("CustomerID"),
   fromCustomer.getString("CustomerName"));
```

最後の例では、処理済みの例外を生成します。システム定義変数 currentException は、例外を含む例外オブジェクトです。

```
raiseException(currentException);
```

releaseRelConnection()

リレーションシップ・データベースへの接続を解放します。

構文

```
void releaseRelConnection(Boolean doCommit)
```

パラメーター

doCommit このメソッドがデータベース接続を解放する前に DtpConnection.commit() メソッドを呼び出す必要があるかを示すフラグ。

戻り値

なし。

例外

DtpConnectionException - データベース接続の解放中、または要求されたコミットやロールバックが失敗したときにエラーが発生した場合です。

注記

releaseRelConnection() メソッドは、この特定のマップに対する接続を解放します。このメソッドは、以下のように、*doCommit* 引き数の値に基づいてデータベース・トランザクションをコミットまたはロールバックします。

- *doCommit* が true の場合、releaseRelConnection() は、データベースでの操作が正常に完了した後に呼び出されたことを前提とするため、トランザクションを安全にコミットできます。
- *doCommit* が false の場合、releaseRelConnection() は、例外の発生により呼び出されたことを前提とするため、トランザクションをロールバックする必要があります。

releaseRelConnection() は、データベース・トランザクションで選択されたアクションを実行した後に、現行のスレッドが排他的に使用しているデータベース接続を解放します。

参照項目

getRelConnection(), release()

---

trace()

トレース・メッセージを生成します。

構文

```
void trace(String traceMsg)
void trace(int traceLevel, String traceMsg)
void trace(int traceLevel, int messageNum)
void trace(int traceLevel, int messageNum, String param [,...])
void trace(int traceLevel, int messageNum, Object[] paramArray)
```

パラメーター

traceLevel メッセージが生成されるトレース・レベル。

traceMsg トレース・ファイルに出力されるストリング。

messageNum マップ・メッセージ・ファイル内のメッセージを表す番号。

param 単一のパラメーター。コンマで区切られた、最大 5 つの単一のパラメーターを追加できます。

paramArray パラメーターの配列。

注記

trace() メソッドは、トレースがオンになっている場合にマップが出力するメッセージを生成します。このメソッドには、以下の 5 つの形式があります。

- 第 1 の形式は、トレースが任意のレベルに設定されている場合に表示されるストリング・メッセージのみを取得します。
- 第 2 の形式は、トレース・レベルおよびストリング・メッセージを取得します。このメッセージはトレースが指定されたレベル以上に設定されるときに表示されます。
- 第 3 の形式は、トレース・レベルおよび番号を取得します。この番号は、マップ・メッセージ・ファイル内のメッセージを表します。メッセージ・テキスト全体はメッセージ・ファイルに表示されます。トレースが指定されたレベル以上に設定されるときメッセージ・ファイルは、パラメーターなしで表示どおりに出力されます。

- 第 4 の形式は、トレース・レベル、マップ・メッセージ・ファイル内のメッセージを表す番号およびメッセージで使用される 1 つ以上のパラメーターを取得します。パラメーター値を最大 5 つまで送信できます。このパラメーター値を、コンマで区切ってメッセージに使用することができます。
- 第 5 の形式は、トレース・レベル、マップ・メッセージ・ファイル内のメッセージを表す番号およびパラメーター値の配列を取得します。

注: *messageNum* パラメーターを取得するメソッドのすべての形式で、メッセージ・ファイルを使用する必要があります。このファイルはメッセージ番号でインデックスが付けられます。メッセージ・テキスト・ファイルの設定方法の詳細は、537 ページの『付録 A. メッセージ・ファイル』を参照してください。

マップ・プロパティーの一部としてマップのトレース・レベルを設定できます。

例

次の例では、レベル 2 のトレース・メッセージを生成し、メッセージのテキストを指定します。

```
trace (2, "Starting to trace at Level 2");
```

次の例では、トレース・レベルが 2 以上の場合に、マップ・メッセージ・ファイル内でメッセージ 201 を出力します。メッセージには、名前と年の 2 つのパラメーターがあります。このメソッドの呼び出しは、これらのパラメーターの値を渡します。

```
trace(2, 201, "DAVID", "1961");
```

参照項目

logError()、logInfo()、logWarning()

第 10 章 BusObj クラス

この章では、BusObj クラスのオブジェクトに対して操作を行うメソッドについて説明します。

注: BusObj クラスは、コラボレーション開発とマッピングの両方で使用されます。各メソッドの『注記』セクションには、メソッドの使用に関する注意事項が示してあります。

この章の最初の 2 つのセクションでは、これらのメソッドとともにリストされている例外と、階層ビジネス・オブジェクトで属性と子ビジネス・オブジェクトを指定する方法について説明します。残りのセクションでは、表 123 にリストされているメソッドについて説明します。

表 123. BusObj メソッドの要約

メソッド	説明	ページ
copy()	入力ビジネス・オブジェクトからこのビジネス・オブジェクトにすべての属性値をコピーします。	360
duplicate()	このオブジェクトとまったく同じビジネス・オブジェクト (BusObj オブジェクト) を作成します。	360
equalKeys()	ビジネス・オブジェクトのキー属性値を入力ビジネス・オブジェクトの属性値と比較します。	361
equals()	ビジネス・オブジェクトの属性値を、子ビジネス・オブジェクトを含む入力ビジネス・オブジェクトの属性値と比較します。	362
equalsShallow()	ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし子ビジネス・オブジェクトは比較から除外されます。	363
exists()	指定した名前のビジネス・オブジェクト属性があるかどうかを検査します。	363
getBoolean()、getDouble()、getFloat()、getInt()、getLong()、get()、getBusObj()、getBusObjArray()、getLongText()、getString()	ビジネス・オブジェクトから単一の属性の値を検索します。	364
getLocale()	ビジネス・オブジェクト・データのロケールを検索します。	366
getType()	このビジネス・オブジェクトの基になったビジネス・オブジェクト定義の名前を検索します。	366
getVerb()	ビジネス・オブジェクトの動詞を検索します。	367
isBlank()	属性の値がゼロ長ストリングに設定されているかどうかを調べます。	367

表 123. BusObj メソッドの要約 (続き)

メソッド	説明	ページ
isKey()	ビジネス・オブジェクトの属性がキー属性として定義されているかどうかを調べます。	368
isNull()	ビジネス・オブジェクトの属性の値が null かどうかを調べます。	368
isRequired()	ビジネス・オブジェクトの属性が必須属性として定義されているかどうかを調べます。	370
keysToString()	ビジネス・オブジェクトの基本キー属性の値をストリングとして検索します。	370
set()	ビジネス・オブジェクトの属性を、個々のデータ型の指定値に設定します。	371
setContent()	このビジネス・オブジェクトの内容を別のビジネス・オブジェクトに設定します。	372
setDefaultAttrValues()	すべての属性をデフォルト値に設定します。	373
setKeys()	ビジネス・オブジェクトのキー属性の値を、別のビジネス・オブジェクトのキー属性の値に設定します。	373
setLocale()	現行ビジネス・オブジェクトのロケールを設定します。	374
setVerb()	ビジネス・オブジェクトの動詞を設定します。	374
setVerbWithCreate()	子ビジネス・オブジェクトのインスタンスを作成し、動詞を設定します。	375
setWithCreate()	ビジネス・オブジェクトの属性を、個々のデータ型の指定値に設定します。その値に対してオブジェクトが存在しない場合は、オブジェクトが作成されます。	375
toString()	ビジネス・オブジェクト内のすべての属性の値をストリングとして戻します。	376
validData()	指定した値が指定した属性に対して有効な型かどうかを検査します。	377

例外と例外タイプ

例外または例外タイプがリストされているメソッドは、CollaborationException 例外をスローします。例外と例外タイプの両方がリストされているメソッドもあります。例外と例外タイプは CollaborationException オブジェクトに関連していますが、以下のように異なります。

- *例外* は、CollaborationException からサブクラス化されたクラスです。サブクラス化された例外がある場合、マッピングでその例外を使用して、問題の原因をより詳細に判断できます。
- *例外タイプ* は、CollaborationException オブジェクト内のデータです。コラボレーション開発者はこの例外タイプを使用して、Designer ユーザー・インターフェースから例外をキャッチします。BusObj の全ユーザーは、このフィールドを使用して、CollaborationException より詳細な例外クラスがスローされていない場合に、失敗の理由を判断することもできます。

階層ビジネス・オブジェクトをトラバースするための構文

階層ビジネス・オブジェクトをトラバースする必要があるコードを作成している場合、子ビジネス・オブジェクト配列のエレメントで属性を指定できる構文を使用する必要があります。その子ビジネス・オブジェクト配列は、子ビジネス・オブジェクト配列、および他のそのような複雑性のエレメントです。この章では、使用する構文を指定します。

属性の仕様は、以下のとおりです。

```
[[attributeName [index ].]...]attributeName
```

この構文は、以下の形式のいずれかに拡張されます。

```
attributeName
attributeName [index ].attributeName
attributeName [index ]... .attributeName
```

注: ビジネス・オブジェクト属性の名前を作成するときに、ピリオド (.) は使用しないでください。ビジネス・オブジェクト属性の名前にピリオドを付けると、IBM WebSphere Business Integration Server Express マップは、ピリオドを Java のドット演算子として解釈して、特別な意味を付与します。例えば、“attribute.name” は、“name” が “attribute” オブジェクトのフィールドまたはメソッドであると解釈されます。

基本型の属性の指定

次の例では、busObj.get() メソッドを使用して、ビジネス・オブジェクト orderObj から OrderID という名前の基本型属性を検索します。

```
orderObj.get("OrderID");
```

子ビジネス・オブジェクトでの属性の指定

次の例では、orderObj は階層ビジネス・オブジェクトであることを前提としています。オブジェクトの属性の 1 つは CustomerInfo で、単一カーディナリティーの子ビジネス・オブジェクトです。例では、CustomerInfo の CustomerName 属性から顧客名を検索します。

```
orderObj.get("CustomerInfo.CustomerName");
```

子ビジネス・オブジェクトの子での属性の指定

CustomerInfo が orderObj の子で、AddressInfo が CustomerInfo の子である子ビジネス・オブジェクトのチェーンがある場合、以下のように、AddressInfo から市区町村の情報を検索できます。

```
orderObj.get("CustomerInfo.AddressInfo.City");
```

子ビジネス・オブジェクトの配列エレメントでの属性の指定

配列でインデックスを指定することで、配列の子ビジネス・オブジェクトを参照することもできます。配列の最初のエレメントは常にゼロで始まります。例えば、次の例では、配列の 3 番目の子ビジネス・オブジェクトから Quantity 属性の値を検索します。

```
orderObj.get("LineItem[2].Quantity");
```

copy()

入力ビジネス・オブジェクトからこのビジネス・オブジェクトにすべての属性値をコピーします。

構文

```
void copy(BusObj inputBusObj)
```

パラメーター

inputBusObj 属性値が現行のビジネス・オブジェクトにコピーされるビジネス・オブジェクトの名前。

注記

copy() メソッドは、すべての子ビジネス・オブジェクトおよび子ビジネス・オブジェクト配列を含むビジネス・オブジェクト全体をコピーします。このメソッドは、コピーされたオブジェクトへの参照は設定しません。代わりに、すべての属性を複製します。つまり、属性の別のコピーを作成します。

例

次の例では、sourceCustomer に含まれる値を destCustomer にコピーします。

```
destCustomer.copy(sourceCustomer);
```

次の例では、3 つのビジネス・オブジェクト (myBusObj、myBusObj2、および mysettingBusObj) を作成し、mysettingBusObj の値を使用して myBusObj の attr1 属性を設定します。次に、myBusObj のすべての属性を myBusObj2 にクローンします。

```
BusObj myBusObj = new BusObj();
BusObj myBusObj2 = new BusObj();

BusObj mySettingBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);
myBusObj2.copy(myBusObj);
```

このコード・フラグメントを実行すると、myBusObj.attr1 と myBusObj2.attr1 の *両方* が mySettingBusObj ビジネス・オブジェクトに設定されます。ただし、mySettingBusObj を変更すると、myBusObj.attr1 は変更されますが、myBusObj2.attr1 は変更されません。myBusObj2 の属性は copy() を使用して設定されたため、値はクローンされました。したがって、myBusObj2 の attr1 の値は、変更*前* の元の mySettingBusObj.attr1 値のままです。

duplicate()

このオブジェクトとまったく同じビジネス・オブジェクト (BusObj オブジェクト) を作成します。

構文

```
BusObj duplicate()
```

戻り値

複製されたビジネス・オブジェクト。

例外

CollaborationException — duplicate() メソッドは、この例外に対して例外タイプ ObjectException を設定できます。

注記

このメソッドはビジネス・オブジェクトの複製を作成し、それを戻します。このメソッド呼び出しの戻り値は、BusObj タイプの宣言された変数に明示的に割り当てる必要があります。

例

次の例では、destCustomer を作成するために sourceCustomer が複製されます。

```
BusObj destCustomer = sourceCustomer.duplicate();
```

---

equalKeys()

ビジネス・オブジェクトのキー属性値を入力ビジネス・オブジェクトの属性値と比較します。

構文

```
boolean equalKeys(BusObjinputBusObj)
```

パラメーター

inputBusObj このビジネス・オブジェクトと比較されるビジネス・オブジェクト。

戻り値

すべてのキー属性の値が同じ場合は true を戻し、同じでない場合は false を戻します。

例外

CollaborationException — equalKeys() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException - ビジネス・オブジェクトの引き数が無効な場合に設定します。

参照項目

equalsShallow(), equals()

注記

このメソッドは浅く比較を行います。つまり子ビジネス・オブジェクトのキーは比較されません。

例

次の例では、order2 のキー値を order1 のキー値と比較します。

```
boolean areEqual = order1.equalKeys(order2);
```

---

equals()

ビジネス・オブジェクトの属性値を、子ビジネス・オブジェクトを含む入力ビジネス・オブジェクトの属性値と比較します。

構文

```
-boolean equals(Object inputBusObj)
```

パラメーター

inputBusObj このビジネス・オブジェクトと比較されるビジネス・オブジェクト。

戻り値

すべての属性の値が同じ場合は true を戻し、同じでない場合は false を戻します。

例外

CollaborationException — equals() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException - ビジネス・オブジェクトの引き数が無効な場合に設定します。

注記

このメソッドは、ビジネス・オブジェクトの属性値を入力ビジネス・オブジェクトの属性値と比較します。ビジネス・オブジェクトが階層的である場合は、子ビジネス・オブジェクトの属性もすべて比較されます。

注: ビジネス・オブジェクトを Object として渡すと、この equals() メソッドが Object.equals() メソッドをオーバーライドします。

比較時に、null 値は比較対象の任意の値と等価であると見なされるため、null 値に起因して true が戻されないということはありません。

参照項目

equalsShallow(), equalKeys()

例

次の例では、order2 のすべての属性を order1 のすべての属性と比較して、比較の結果を変数 areEqual に割り当てます。子ビジネス・オブジェクトが存在する場合は、その属性も比較されます。

```
boolean areEqual = order1.equals(order2);
```

equalsShallow()

ビジネス・オブジェクトの属性値を、入力ビジネス・オブジェクトの属性値と比較します。ただし子ビジネス・オブジェクトは比較から除外されます。

構文

```
boolean equalsShallow(BusObj inputBusObj)
```

パラメーター

inputBusObj このビジネス・オブジェクトと比較されるビジネス・オブジェクト。

戻り値

すべての属性の値が同じ場合は true を戻し、同じでない場合は false を戻します。

例外

CollaborationException — equalsShallow() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException - ビジネス・オブジェクトの引き数が無効な場合に設定します。

参照項目

equals(), equalKeys()

例

次の例は、order2 の属性を order1 の属性と比較します。ただし子ビジネス・オブジェクトが存在する場合、その属性は比較から除外されます。

```
boolean areEqual = order1.equalsShallow(order2);
```

exists()

指定した名前のビジネス・オブジェクト属性があるかどうかを検査します。

構文

```
boolean exists(String attribute)
```

パラメーター

attribute 属性の名前。

戻り値

属性が存在する場合は true を戻し、存在しない場合は false を戻します。

例

次の例では、ビジネス・オブジェクト order に Notes という属性があるかどうかを検査します。

```
boolean notesAreHere = order.exists("Notes");
```

---

getBoolean()、getDouble()、 getFloat()、getInt()、 getLong()、get()、getBusObj()、 getBusObjArray()、 getLongText()、getString()

ビジネス・オブジェクトから単一の属性の値を検索します。

構文

```
Object get(String attribute)
Object get(int position)

boolean getBoolean(String attribute)
double getDouble(String attribute)
float getFloat(String attribute)
int getInt(String attribute)
long getLong(String attribute)
Object get(String attribute)
BusObj getBusObj(String attribute)
BusObjArray getBusObjArray(String attribute)
String getLongText(String attribute)
String getString(String attribute)
```

パラメーター

attribute 属性の名前。

position ビジネス・オブジェクトの属性リスト内で属性の順序を指定する整数。

戻り値

指定された属性の値。

例外

CollaborationException — 上記の get メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - 属性にアクセスの問題が発生した場合に設定します。例えば、数字で構成されていない String 属性に対してコラボレーションが getDouble() を呼び出した場合や、存在しない属性に対して getString() を呼び出した場合に、この例外を発生させることができます。

注記

get() メソッドは現行ビジネス・オブジェクトから属性値を検索します。属性値のコピー (複製) を戻します。このメソッドは、ソース・ビジネス・オブジェクトのこの属性 に対するオブジェクト参照を戻しません。したがって、ソース・ビジネス・オブジェクトの属性を変更しても、get() が戻す値は変更されません。このメソッドは、呼び出されるたびに属性の新しいコピー (複製) を戻します。

get() メソッドには、以下の形式があります。

- 第 1 の形式では、メソッド名に指定された型の値を戻します。例えば、getBoolean() は boolean 値を、getBusObj() は BusObj 値を、getDouble() は double 値を戻します。ただし、WebSphere Business Integration Server Express の longtext 型は、最大サイズのないストリング・オブジェクトであるため、getLongText() はストリング・オブジェクトを戻します。特定の基本データ型または WebSphere Business Integration Server Express 定義のデータ型の属性を検索するには、これらの形式を使用してください。

  これらのメソッドでは、属性の*名前* を指定することによって属性値にアクセスできます。
- 第 2 の形式、get() では、*不特定* の型の属性値を検索します。戻された値は、属性タイプの該当する値にキャストできます。

  このメソッドでは、属性の*名前*、またはビジネス・オブジェクトの属性リスト内における属性索引の*位置* の*いずれか一方* を指定することによって、属性値にアクセスできます。

例

次の例は、get() がオブジェクト参照の代わりに属性値のコピー (複製) を戻すメカニズムを示しています。

```
BusObj mySettingBusObj = new BusObj();
BusObj myBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);

BusObj Extract = myBusObj.get("attr1");
```

このコード・フラグメントを実行した後に、Extract ビジネス・オブジェクトを変更しても、mySettingBusObj は変更*されません*。これは、get() の呼び出しにより attr1 属性のコピーが戻されたためです。

次の例では、getBusObj() を使用して、customer ビジネス・オブジェクトから顧客の住所を含む子ビジネス・オブジェクトを検索し、変数 address に割り当てます。

```
BusObj address = customer.getBusObj("Address");
```

次の例では、getString() を使用して、CustomerName 属性の値を検索します。ビジネス・オブジェクト変数は sourceCustomer です。

```
String customerName = sourceCustomer.getString("CustomerName");
```

次の例では、getInt() を使用して、変数が item1 と item2 の 2 つのビジネス・オブジェクトから Quantity 値を検索します。次に両方の数量の合計を計算します。

```
int sumQuantity = item1.getInt("Quantity") + item2.getInt("Quantity");
```

次の例では、ビジネス・オブジェクト変数 order から属性 Item を検索します。属性 Item はビジネス・オブジェクト配列です。

```
BusObjArray items = order.getBusObjArray("Item");
```

次の例は、ソース・ビジネス・オブジェクトから CustID 属性値を取得し、これに一致するように宛先ビジネス・オブジェクトの Customer の値を設定します。

```
destination.set("Customer", source.get("CustID"));
```

次の例では、属性リスト内における属性の順序を使用して属性値にアクセスします。

```
for i=0; i<maxAttrCount; i++)
{
        String strValue = (String)myBusObj.get(i);
    ...
```

---

getLocale()

ビジネス・オブジェクトのデータに関連したロケールを検索します。

構文

```
java.util.Locale getLocale()
```

パラメーター

なし。

戻り値

ビジネス・オブジェクトのロケールに関する情報を含む Java ロケール・オブジェクト。このロケール・オブジェクトは `java.util.Locale` クラスのインスタンスでなければなりません。

注記

`getLocale()` メソッドは、ビジネス・オブジェクトのデータに関連したロケールを戻します。このロケールは、コラボレーションが実行されているコラボレーション・ロケールとは、多くの場合異なります。

参照項目

getLocale() (`BaseCollaboration` クラス)、setLocale()

---

getType()

このビジネス・オブジェクトの基になったビジネス・オブジェクト定義の名前を検索します。

構文

```
String getType()
```

戻り値

ビジネス・オブジェクト定義の名前。

注記

このメソッドでは、ビジネス・オブジェクトのタイプは、ビジネス・オブジェクトの作成元であるビジネス・オブジェクト定義の名前です。

戻り値

次の例は、sourceShipTo という名前のビジネス・オブジェクトのタイプを検索します。

```
String typeName = sourceShipTo.getType();
```

次の例は、トリガー・イベントを適切なタイプの新規ビジネス・オブジェクトにコピーします。

```
BusObj source = new BusObj(triggeringBusObj.getType());
```

getVerb()

ビジネス・オブジェクトの動詞を検索します。

構文

```
String getVerb()
```

戻り値

動詞の名前。例えば、Create、Retrieve、Update、または Delete です。

注記

コラボレーション開発では、このメソッドは、複数のタイプの着信イベントを処理するシナリオに役立ちます。シナリオの最初のアクション・ノードが getVerb() を呼び出します。次にそのアクション・ノードからの出力遷移リンクで、戻されたストリングの内容をテストし、それぞれの出力遷移リンクが、使用可能な動詞の 1 つを処理する実行パスの先頭になるようにします。

例

次の例は、orderEvent というビジネス・オブジェクトから動詞を取得し、それを orderVerb という変数に割り当てます。

```
String orderVerb = orderEvent.getVerb();
```

isBlank()

属性の値がゼロ長ストリングに設定されているかどうかを調べます。

構文

```
boolean isBlank(String attribute)
```

パラメーター

attribute　　属性の名前。

戻り値

属性値がゼロ長ストリングの場合は true を戻し、それ例外の場合は false を戻します。

注記

ゼロ長ストリングは、ストリング "" と同等です。ゼロ長ストリングは null とは異なります。null の存在は、isNull() メソッドで検出されます。

コラボレーションが属性値を検索してからその値に対して何らかの操作を行う必要がある場合、値を検索する前に isBlank() と isNull() を呼び出すことで、その属性が値を持っているかどうかを検査できます。

例

次の例では、sourcePaperClip ビジネス・オブジェクトの Material 属性がゼロ長ストリングかどうかを検査します。

```
boolean key = sourcePaperClip.isBlank("Material");
```

isKey()

ビジネス・オブジェクトの属性がキー属性として定義されているかどうかを調べます。

構文

```
boolean isKey(String attribute)
```

パラメーター

attribute　　属性の名前。

戻り値

属性がキー属性である場合は true を戻し、キー属性でない場合は false を戻します。

例

次の例では、customer ビジネス・オブジェクトの CustID 属性がキー属性かどうかを判断します。

```
boolean keyAttr = (customer.isKey("CustID"));
```

isNull()

ビジネス・オブジェクトの属性の値が null かどうかを調べます。

構文

```
boolean isNull(String attribute)
```

パラメーター

attribute　　属性の名前。

戻り値

属性値が null である場合は true を戻し、null でない場合は false を戻します。

注記

null は、ゼロ長ストリングと異なり、値がないことを示します。ゼロ長ストリングは isBlank() を呼び出すことによって検出されます。オブジェクトは、使用する前に isNull() によってテストしてください。なぜなら、オブジェクトが null であると操作が失敗することがあるためです。

以下の状況では、属性値が null であることがあります。

- 属性値が明示的に null に設定された。

  属性値は、set() メソッドを使用して null に設定できます。
- 属性値が設定されていない。

  新しいビジネス・オブジェクトをインスタンス化するときに、すべての属性値が null で初期化されます。ビジネス・オブジェクトの作成後に isNull() を呼び出すまでに属性値が設定されないと、値は null のままです。
- マッピング中に null が挿入された。

  コラボレーションがコネクターから受け取ったビジネス・オブジェクトを処理している場合、マッピング処理により null が挿入された可能性があります。マッピング・プロセスでは、コネクターから受け取ったアプリケーション固有のビジネス・オブジェクトは、コラボレーションによって処理される汎用ビジネス・オブジェクトに変換されます。汎用ビジネス・オブジェクトの属性のうち、アプリケーション固有のオブジェクトにおいて等価な属性のない属性については、それぞれマッピングによって null 値が挿入されます。

ヒント: Java では null オブジェクトに対して操作を行うことができないため、子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列である属性に対して操作を実行する前に、必ず isNull() を呼び出してください。

例

次の例では、sourcePaperClip ビジネス・オブジェクトの Material 属性に null 値があるかどうかを検査します。

```
boolean key = sourcePaperClip.isNull("Material");
```

次の例では、contract1 ビジネス・オブジェクトの CustAddr 属性が検索前は null かどうかを検査します。isNull() 検査が false である、つまり属性が null でない場合にのみ属性の検索が行われます。

```
if (! contract1.isNull("CustAddr"))
   {
   BusObj customerAddress = contract1.getBusObj("CustAddr");
   //do something with the "customerAddress" business object
   }
```

isRequired()

ビジネス・オブジェクトの属性が必須属性として定義されているかどうかを調べます。

構文

```
boolean isRequired(String attribute)
```

パラメーター

attribute　　属性の名前。

戻り値

属性値が必須の場合は `true` を戻し、必須でない場合は `false` を戻します。

注記

属性が必須として定義されている場合、その属性は null でない値を持つ必要があります。

例

次の例は、必須属性の値が null である場合に警告をログに記録します。

```
if ( (customer.isRequired("Address"))
      && (customerBusObj.isNull("Address")) )
   {
   logWarning(12, "Address is required and cannot be null.");
   }
else
   {
   //do something else
   }
```

keysToString()

ビジネス・オブジェクトの基本キー属性の値をストリングとして検索します。

構文

```
String keysToString()
```

戻り値

ビジネス・オブジェクト内のすべてのキー値が、属性の序数値の順序に並べられて連結されたストリング・オブジェクト。

注記

このメソッドの出力には、属性の名前と値が含まれます。複数値は、スペースで区切られて連結された基本キー属性値です。例えば、1 つの基本キー属性 SS# がある場合、これは次のような出力になります。

```
SS#=100408394
```

基本キー属性が FirstName と LastName の場合、これは次のような出力になります。

```
FirstName=Nina LastName=Silk
```

例

次の例は、変数名 fromOrder によって表されるビジネス・オブジェクトのキー属性の値を戻します。

```
String keyValues = fromOrder.keysToString();
```

---

set()

ビジネス・オブジェクトの属性を、個々のデータ型の指定値に設定します。

構文

```
void set(String attribute, Object value)
void set(int position, Object value)

void set(String attribute, boolean value)
void set(String attribute, double value)
void set(String attribute, float value)
void set(String attribute, int value)
void set(String attribute, long value)
void set(String attribute, Object value)
void set(String attribute, String value)
```

パラメーター

attribute	設定する属性の名前。
position	ビジネス・オブジェクトの属性リスト内で属性の順序を指定する整数。
value	属性値。

例外

CollaborationException — set() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException — 属性にアクセスの問題が発生した場合に設定します。

注記

set() メソッドは、現行ビジネス・オブジェクトの属性値を設定します。このメソッドは、属性に値を割り当てるときに、*値* パラメーターに対するオブジェクト参照を設定します。このメソッドは、ソース・ビジネス・オブジェクトから属性値をクローンしません。このため、ソース・ビジネス・オブジェクトの*値* の変更内容は、set() メソッドを呼び出すビジネス・オブジェクトの属性にも適用されます。

set() メソッドには、以下の形式があります。

- 第 1 の形式では、メソッドの第 2 のパラメーターの型で指定された型の値を設定します。例えば、set(String *attribute*, boolean *value*) はブール値によって属性を設定し、set(String *attribute*, double *value*) はダブル値によって属性を設定する、

というように使用します。特定の基本データ型または WebSphere Business Integration Server Express 定義のデータ型の属性を設定するには、この形式を使用してください。

これらのメソッドでは、属性の*名前* を指定することによって属性値にアクセスできます。

- 第 2 の形式では、*不特定* の型の属性値を設定します。属性値として不特定型のデータを送信できるのは、属性値パラメーターの型がオブジェクトであるためです。例えば、BusObj または LongText オブジェクトの属性を設定するには、この形式のメソッドを使用して、BusObj または LongText オブジェクトを属性値として渡します。

  この形式の set() メソッドでは、属性の*名前*、またはビジネス・オブジェクトの属性リスト内における属性索引の*位置* のいずれか一方を指定することによって、属性値にアクセスできます。

例

次の例では、toCustomer の LName 属性を値 Smith に設定します。

```
toCustomer.set("LName", "Smith");
```

次の例は、set() が値を複製するのではなくオブジェクト参照を割り当てるメカニズムを示します。

```
BusObj BusObj myBusObj = new BusObj();
BusObj mySettingBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);
```

このコード・フラグメントを実行すると、myBusObj の attr1 属性が mySettingBusObj ビジネス・オブジェクトに設定されます。mySettingBusObj を変更すると、myBusObj.attr1 も同じように変更されます。これは、set() が attr1 属性を設定するときに、mySettingBusObj へのオブジェクト参照を行うためです。mySettingBusObj の静的コピーは作成されません。

次の例では、属性リスト内での属性の順序を使用して属性値を設定します。

```
for i=0; i<maxAttrCount; i++)
{
        myBusObj.set(i, strValue);
    ...
```

setContent()

このビジネス・オブジェクトの内容を別のビジネス・オブジェクトに設定します。

構文

```
void setContent(BusObj BusObj)
```

パラメーター

BusObj このビジネス・オブジェクトの値を設定するのに値が使用されるビジネス・オブジェクト。

例外

CollaborationException — setContent() メソッドは、この例外に対して次の例外タイプのいずれかを設定できます。

- AttributeException: 属性にアクセスの問題が発生した場合に設定します。
- ObjectException - ビジネス・オブジェクトの引き数が無効な場合に設定します。

例

次の例では、出力オブジェクト ObjOutput1 のインスタンス変数の内容がビジネス・オブジェクト rDstBO[0] の内容に設定されます。

```
ObjOutput1.setContent(rDstBO[0]);
```

setDefaultAttrValues()

すべての属性をデフォルト値に設定します。

構文

```
void setDefaultAttrValues()
```

注記

ビジネス・オブジェクト定義には、属性のデフォルト値を含めることができます。このメソッドは、ビジネス・オブジェクトの属性の値を、定義の中でデフォルトとして指定されている値に設定します。

例

次の例は、PaperClip ビジネス・オブジェクトの値をそのデフォルト値に設定します。

```
PaperClip.setDefaultAttrValues();
```

setKeys()

ビジネス・オブジェクトのキー属性の値を、別のビジネス・オブジェクトのキー属性の値に設定します。

構文

```
void setKeys(BusObj inputBusObj)
```

パラメーター

inputBusObj 別のビジネス・オブジェクトの値を設定するのに値が使用されるビジネス・オブジェクト。

例外

CollaborationException — setKeys() メソッドは、この例外に対して次の例外タイプのいずれかを設定できます。

- AttributeException: 属性にアクセスの問題が発生した場合に設定します。
- ObjectException - ビジネス・オブジェクトの引き数が無効な場合に設定します。

例

次の例は、ビジネス・オブジェクト helpdeskCustomer のキー値をビジネス・オブジェクト ERPCustomer キー値に設定します。

```
helpdeskCustomer.setKeys(ERPCustomer);
```

setLocale()

現行ビジネス・オブジェクトのロケールを設定します。

構文

```
void setLocale(java.util.Locale locale)
```

パラメーター

locale ビジネス・オブジェクトに割り当てるロケールに関する情報を含む Java ロケール・オブジェクト。このロケール・オブジェクトは java.util.Locale クラスのインスタンスでなければなりません。

戻り値

なし。

注記

setLocale() メソッドは、ビジネス・オブジェクトに関連したデータにロケールを割り当てます。このロケールは、コラボレーションが実行されているコラボレーション・ロケールとは異なる場合があります。

参照項目

getLocale()

setVerb()

ビジネス・オブジェクトの動詞を設定します。

構文

```
void setVerb(String verb)
```

パラメーター

verb ビジネス・オブジェクトの動詞。

注記

setVerb() メソッドは、マッピングでのみ使用されます。

注: このメソッドをコラボレーション開発で使用しない でください。コラボレーション開発では、サービス呼び出しのプロパティーを入力することで、出力ビジネス・オブジェクトの動詞を対話式に設定する必要があります。

例

次の例は、ビジネス・オブジェクト contactAddress に動詞 Delete を設定します。

```
contactAddress.setVerb("Delete");
```

setVerbWithCreate()

子ビジネス・オブジェクトのインスタンスを作成し、動詞を設定します。

構文

```
void setVerbWithCreate(String attributeName, String verb)
```

パラメーター

attributeName 作成された子ビジネス・オブジェクトの名前。

verb 設定される動詞。

例外

CollaborationException — setVerbWithCreate() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException — 属性にアクセスの問題が発生した場合に設定します。

注記

attributeName パラメーターで指定された属性のタイプが BusObj で、これが null である場合、その子ビジネス・オブジェクトの新しいインスタンスが作成され、動詞は verb パラメーターの値に設定されます。この子ビジネス・オブジェクトのインスタンスがすでに存在する場合は、動詞のみが設定されます。子ビジネス・オブジェクトがマルチカーディナリティーである場合は、*attributeName* パラメーターは添え字を指定する必要があります。

例

次の例では、childBO 子ビジネス・オブジェクトのインスタンスを作成し、動詞を Create に設定します。

```
myBO.setVerbWithCreate("childBO", "Create");
```

setWithCreate()

ビジネス・オブジェクトの属性を、個々のデータ型の指定値に設定します。その値に対してオブジェクトが存在しない場合は、オブジェクトが作成されます。

構文

```
void setWithCreate(String attributeName, BusObj busObj)
void setWithCreate(String attributeName, BusObjArray busObjArray)
void setWithCreate(String attributeName, Object value)
```

パラメーター

attributeName 設定する属性の名前。

busObj ターゲットの属性に挿入されるビジネス・オブジェクト。

busObjArray ターゲットの属性に挿入されるビジネス・オブジェクト配列。

value ターゲットの属性に挿入されるオブジェクト。このオブジェクトのタイプは、BusObj、BusObjArray、Object のいずれかである必要があります。

例外

CollaborationException — setWithCreate() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException — 属性にアクセスの問題が発生した場合に設定します。

注記

指定するオブジェクトが BusObj であり、ターゲットの属性に複数カーディナリティーの子ビジネス・オブジェクトが含まれる場合、BusObj は BusObjArray に最後の要素として追加されます。ただし、ターゲットの属性に BusObj が含まれている場合は、このビジネス・オブジェクトが元の値に置き換わります。

例

次の例は、ChildAttrAttr という属性を値 5 に設定します。この属性は、myBO の属性の ChildAttr に含まれるビジネス・オブジェクトの中にあります。呼び出し時に childAttr ビジネス・オブジェクトが存在しないと、このメソッド呼び出しによってこのビジネス・オブジェクトが作成されます。

```
myBO.setWithCreate("childAttr.childAttrAttr", "5");
```

---

toString()

ビジネス・オブジェクト内のすべての属性の値をストリングとして戻します。

構文

```
String toString()
```

戻り値

ビジネス・オブジェクトのすべての属性値を含む String オブジェクト。

注記

このメソッドを呼び出した結果のストリングは、例えば以下のようになります。

```
Name: GenEmployee
Verb: Create
Type: AfterImage
Attributes: (Name, Type, Value)

LastName:String, Davis
FirstName:String, Miles
SS#:String, 041-33-8989
Salary:Float, 15.00
ObjectEventId:String, MyConnector_922323619411_1
```

例

次の例は、ビジネス・オブジェクト変数 fromOrder の属性値を含むストリングを戻します。

```
String values = fromOrder.toString();
```

validData()

指定した値が指定した属性に対して有効な型かどうかを検査します。

構文

```
boolean validData(String attributeName, Object value)
boolean validData(String attributeName, BusObj value)
boolean validData(String attributeName, BusObjArray value)
boolean validData(String attributeName, String value)
boolean validData(String attributeName, long value)
boolean validData(String attributeName, int value)
boolean validData(String attributeName, double value)
boolean validData(String attributeName, float value)
boolean validData(String attributeName, boolean value)
```

パラメーター

attributeName 属性。

value 値。

戻り値

true または false (ブール戻り値)

注記

ターゲットの属性 (*attributeName* で指定される) によって渡される値の互換性を検査します。基準は以下のとおりです。

基本型 (String、long、int、double、float、boolean) の場合	値は属性のデータ型に変換可能である必要があります。
BusObj の場合	値のデータ型はターゲットの属性値のデータ型と同じである必要があります。
BusObjArray の場合	値は、属性のタイプと同じ (ビジネス・オブジェクト定義) タイプである BusObj または BusObjArray を指している必要があります。

Object の場合	値のタイプは、String、BusObj、または BusObjArray でなければなりません。対応する検証規則が適用されます。

使用すべきでないメソッド

BusObj クラスのメソッドには、以前のバージョンではサポートされていたが現在のバージョンではサポートされていないものがいくつかあります。これらの*使用すべきでないメソッド* によりエラーが生成されることはありませんが、CrossWorlds では、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。

BusObj クラスの使用すべきでないメソッドを表 124 に示します。これまでに Map Designer Express を使用したことがない場合は、このセクションは無視してください。

表 124. BusObj クラスの使用すべきでないメソッド

以前のメソッド	代替メソッド
getCount()	BusObjArray.size()
getKeys()	keysToString()
getValues()	toString()
not	標準の Java NOT 演算子、"!"
set(BusObj *inputBusObj*)	copy()
入力引き数として子ビジネス・オブジェクトまたは子ビジネス・オブジェクト配列をとるメソッドすべて	子ビジネス・オブジェクトまたはビジネス・オブジェクト配列へのハンドルを取得し BusObj または BusObjArray クラスのメソッドを使用してください。

setVerb() メソッドは、以前のバージョンでは使用すべきでないメソッドのリストに入っていましたが、現在ではマッピングでの使用のためリストからはずされています。このメソッドはコラボレーション内では使用しないでください。

第 11 章 BusObjArray クラス

この章では、IBM WebSphere Business Integration Server Express 定義のクラス BusObjArray のオブジェクトに対して操作を行うメソッドについて説明します。BusObjArray クラスは、ビジネス・オブジェクトの配列をカプセル化します。階層ビジネス・オブジェクトでは、属性のカーディナリティーが n に等しいときに、属性は子ビジネス・オブジェクトの配列への参照になります。BusObjArray クラスに対する操作は、BusObjArray オブジェクト、またはビジネス・オブジェクトの実際の配列を戻します。

注: BusObjArray クラスは、コラボレーション開発とマッピングの両方で使用されます。各メソッドの『注記』セクションには、メソッドの使用に関する注意事項が示してあります。

BusObjArray クラスのメソッドを表 125 に示します。

表 125. BusObjArray メソッドの要約

メソッド	説明	ページ
addElement()	ビジネス・オブジェクト配列にビジネス・オブジェクトを追加します。	380
duplicate()	このビジネス・オブジェクト配列とまったく同じビジネス・オブジェクト配列 (BusObjArray オブジェクト) を作成します。	381
elementAt()	このビジネス・オブジェクト配列での位置を指定することで、単一のビジネス・オブジェクトを検索します。	381
equals()	別のビジネス・オブジェクト配列とこのビジネス・オブジェクト配列を比較します。	382
getElements()	このビジネス・オブジェクト配列の内容を検索します。	382
getLastIndex()	ビジネス・オブジェクト配列から使用可能な最後のインデックスを検索します。	383
max()	このビジネス・オブジェクト配列のすべてのエレメントから、指定した属性の最大値を検索します。	383
maxBusObjArray()	指定した属性の最大値を持つビジネス・オブジェクトを、ビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。	384
maxBusObjs()	指定した属性の最大値を持つビジネス・オブジェクトを、BusObj オブジェクトの配列として戻します。	385
min()	配列の中のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。	386
minBusObjArray()	指定した属性の最小値を持つビジネス・オブジェクトを、BusObjArray オブジェクトとして戻します。	387

表 125. BusObjArray メソッドの要約 (続き)

メソッド	説明	ページ
minBusObjs()	指定した属性の最小値を持つビジネス・オブジェクトを、BusObj オブジェクトの配列として戻します。	388
removeAllElements()	ビジネス・オブジェクト配列からすべてのエレメントを除去します。	389
removeElement()	ビジネス・オブジェクト配列からビジネス・オブジェクト・エレメントを除去します。	389
removeElementAt()	このビジネス・オブジェクト配列の個々の位置にあるエレメントを除去します。	390
setElementAt()	ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を指定します。	390
size()	このビジネス・オブジェクト配列内のエレメントの数を戻します。	391
sum()	ビジネス・オブジェクト配列の中のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。	391
swap()	このビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。この場合、配列の最初のエレメントが 0、2 番目が 1、3 番目が 2 であり、以下同様となることを念頭に置いてください。	392
toString()	このビジネス・オブジェクト配列内の値を単一のストリングとして検索します。	392

注: このクラスの例外処理に関する重要な説明は、358 ページの『例外と例外タイプ』を参照してください。セクションは、BusObjArray と BusObj の例外にのみ適用されます。

addElement()

ビジネス・オブジェクト配列にビジネス・オブジェクトを追加します。

構文

```
void addElement(BusObj element)
```

パラメーター

element 配列に追加するビジネス・オブジェクト。

例外

CollaborationException — addElement() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException: エレメントが無効な場合に設定します。

例

次の例では、getBusObjArray() メソッドを使用して、ビジネス・オブジェクト order から itemList というビジネス・オブジェクトの配列を検索します。配列は items に割り当てられ、items に新しいビジネス・オブジェクトが追加されます。

```
BusObjArray items = order.getBusObjArray("itemList");
items.addElement(new BusObj("oneItem"));
```

---

duplicate()

このビジネス・オブジェクト配列とまったく同じビジネス・オブジェクト配列 (BusObjArray オブジェクト) を作成します。

構文

```
BusObjArray duplicate()
```

戻り値

ビジネス・オブジェクト配列。

例

次の例では、items 配列が複製され、newItems が作成されます。

```
BusObjArray newItems = items.duplicate();
```

---

elementAt()

このビジネス・オブジェクト配列での位置を指定することで、単一のビジネス・オブジェクトを検索します。

構文

```
BusObj elementAt(int index)
```

パラメーター

index 検索される配列のエレメント。配列の最初のエレメントが 0、2 番目が 1、3 番目が 2 であり、以下同様です。

例外

CollaborationException — elementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException: エレメントが無効な場合に設定します。

例

次の例では、items 配列の 11 番目のビジネス・オブジェクトが検索され、Item 変数に割り当てられます。

```
BusObj Item = items.elementAt(10);
```

equals()

別のビジネス・オブジェクト配列とこのビジネス・オブジェクト配列を比較します。

構文

```
boolean equals(BusObjArray inputBusObjArray)
```

パラメーター

inputBusObjArray

このビジネス・オブジェクト配列と比較されるビジネス・オブジェクト配列。

注記

2 つのビジネス・オブジェクト配列の比較とは、エレメントの数とそれらの属性値を検査することです。

例

次の例は、equals() を使用して、条件付きループを設定します (ループの内側は示してありません)。

```
if (items.equals(newItems))
   {
...
   }
```

getElements()

このビジネス・オブジェクト配列の内容を検索します。

構文

```
BusObj[] getElements()
```

例外

CollaborationException — getElements() メソッドは、この例外に対して次の例外タイプを設定できます。

- ObjectException - エレメントの 1 つが無効な場合に設定します。

例

次の例は、items 配列のエレメントを出力します。

```
BusObj[] elements = items.getElements();
for (int i=0, i<elements.length; i++)
   {
   trace(1, elements[i].toString());
   }
```

getLastIndex()

ビジネス・オブジェクト配列から使用可能な最後のインデックスを検索します。

構文

```
int getLastIndex()
```

戻り値

BusObjArray の中の最後のエレメントを指す最後のインデックス。

注記

以前のバージョンでは、size() メソッドを使用していました。つまり、BusObjArray の中の最後の有効なインデックスを検索するときにはビジネス・オブジェクト配列の size() を使用していました。しかし BusObjArray にギャップが含まれる場合、この方法では不正なデータが戻されます。

すべての Java 配列と同様に、BusObjArray はゼロ相対配列です。したがって、size() メソッドは getLastIndex() メソッドより 1 大きい値を戻します。

例

次の例は、ビジネス・オブジェクト配列の中の最後のインデックスを検索します。

```
int lastElementIndex = items.getLastIndex();
```

max()

このビジネス・オブジェクト配列のすべてのエレメントから、指定した属性の最大値を検索します。

構文

```
String max(String attr)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する変数。属性の型は、String、LongText、Integer、Float、および Double のいずれかである必要があります。

戻り値

指定した属性の、ストリング形式での最大値。または、BusObjArray 内のすべてのエレメントについてその属性の値が null である場合は null。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。max() メソッドは、この例外に対して例外タイプ AttributeException を設定できます。

注記

max() メソッドは、BusObjArray の中のビジネス・オブジェクトのうち、指定した属性の最大値を検索します。例えば、3 つの Employee オブジェクトが使用されていて、その属性が Float 型の Salary である場合、最大の給与を表すストリングが戻されます。

BusObjArray 内のエレメントのうち、指定した属性の値が null であるエレメントは無視されます。すべてのエレメントについて指定した属性値が null の場合、null が戻されます。

属性のデータ型が String である場合、max() は字句が最長のストリングである属性値を戻します。

例

```
String maxSalary = items.max("Salary");
```

maxBusObjArray()

指定した属性の最大値を持つビジネス・オブジェクトを、ビジネス・オブジェクト配列 (BusObjArray オブジェクト) として戻します。

構文

```
BusObjArray maxBusObjArray(String attr)
```

パラメーター

attr ビジネス・オブジェクト配列内のビジネス・オブジェクトの属性を参照する String 型、LongText 型、Integer 型、Float 型、または Double 型の変数。

戻り値

BusObjArray または null 形式のビジネス・オブジェクトのリスト。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。maxBusObjArray() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

注記

maxBusObjArray() メソッドは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObjArray オブジェクトによって戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最大値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最長のストリングが戻されます。

例

```
BusObjArray boarrayWithMaxSalary = items.maxBusObjArray("Salary");
```

maxBusObjs()

指定した属性の最大値を持つビジネス・オブジェクトを、BusObj オブジェクトの配列として戻します。

構文

```
BusObj[] maxBusObjs(String attr)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、Integer 型、Float 型、または Double 型の変数。

戻り値

BusObj[] または null 形式のビジネス・オブジェクトのリスト。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。maxBusObjs() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

注記

maxBusObjs() メソッドは、指定した属性の最大値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最大値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最大値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最長のストリングが戻されます。

例

```
BusObj[] bosWithMaxSalary = items.maxBusObjs("Salary");
```

min()

配列の中のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。

構文

```
String min(String attr)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、Integer 型、Float 型、または Double 型の変数。

戻り値

指定した属性の、ストリング形式での最小値。または、BusObjArray 内のすべてのエレメントについてその属性の値が null である場合は null。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。min() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

注記

min() メソッドは、ビジネス・オブジェクト配列の中のビジネス・オブジェクトのうち、指定した属性の最小値を検索します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、それらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

例

```
String minSalary = items.min("Salary");
```

---

minBusObjArray()

指定した属性の最小値を持つビジネス・オブジェクトを、BusObjArray オブジェクトとして戻します。

構文

```
BusObjArray minBusObjArray(String attr)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、Integer 型、Float 型、または Double 型の変数。

戻り値

BusObjArray または null 形式のビジネス・オブジェクトのリスト。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。minBusObjArray() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

注記

minBusObjArray() メソッドは、指定した属性の最小値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクト BusObjArray オブジェクトによって戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、これらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

例

```
BusObjArray boarrayWithMinSalary = items.minBusObjArray("Salary");
```

minBusObjs()

指定した属性の最小値を持つビジネス・オブジェクトを、BusObj オブジェクトの配列として戻します。

構文

```
BusObj[] minBusObjs(String attr)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する String 型、LongText 型、Integer 型、Float 型、または Double 型の変数。

戻り値

BusObj[] または null 形式のビジネス・オブジェクトのリスト。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。minBusObjs() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

注記

minBusObjs() メソッドは、指定した属性の最小値を持つ 1 つ以上のビジネス・オブジェクトを検索し、これらのビジネス・オブジェクトを BusObj オブジェクトの配列として戻します。

例えば、Employee ビジネス・オブジェクトを含むビジネス・オブジェクト配列があり、入力引き数が Float 型の属性 Salary であるとします。このメソッドは、すべての Employee ビジネス・オブジェクトの中で Salary の最小値を調べ、その値を含むビジネス・オブジェクトを戻します。Salary の最小値を持つビジネス・オブジェクトが複数ある場合は、これらのビジネス・オブジェクトすべてが戻されます。

指定した属性が null を含むビジネス・オブジェクトは無視されます。配列内のすべてのビジネス・オブジェクトで値が null である場合、null が戻されます。

属性のデータ型が String のときは、字句が最短のストリングが戻されます。

例

```
BusObj[] bosWithMinSalary = items.minBusObjs("Salary");
```

removeAllElements()

ビジネス・オブジェクト配列からすべてのエレメントを除去します。

構文

```
void removeAllElements()
```

例

次の例は、配列 items のすべてのエレメントを除去します。

```
items.removeAllElements();
```

removeElement()

ビジネス・オブジェクト配列からビジネス・オブジェクト・エレメントを除去します。

構文

```
void removeElement(BusObj element)
```

パラメーター

elementReference
配列のエレメントを参照する変数。

例外

CollaborationException — removeElement() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - エレメントが無効な場合に設定します。

注記

配列からエレメントを削除すると、配列のサイズが変わり、既存のエレメントのインデックスが変更されます。

例

次の例では、ビジネス・オブジェクト配列 items からエレメント Child1 を削除します。

```
items.removeElement(Child1);
```

removeElementAt()

このビジネス・オブジェクト配列の個々の位置にあるエレメントを除去します。

構文

```
void removeElementAt(int index)
```

注記

配列からエレメントを除去すると、配列のサイズが変わり、場合によっては既存のエレメントのインデックスが変更されます。

パラメーター

index エレメントのインデックス。

例外

CollaborationException — removeElementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - エレメントが無効な場合に設定します。

例

次の例は、配列 items の 6 番目のビジネス・オブジェクトを削除します。

```
items.removeElementAt(5);
```

setElementAt()

ビジネス・オブジェクト配列内のビジネス・オブジェクトの値を指定します。

構文

```
void setElementAt (int index, BusObj element)
```

パラメーター

index 配列の位置を表す整数。配列の最初のエレメントが 0、2 番目が 1、3 番目が 2 であり、以下同様です。

inputBusObj 配列エレメントを設定する値を含むビジネス・オブジェクト。

例外

CollaborationException — setElementAt() メソッドは、この例外に対して次の例外タイプを設定できます。

- AttributeException - エレメントが無効な場合に設定します。

注記

このメソッドは、配列内の指定位置にあるビジネス・オブジェクトの値を、入力ビジネス・オブジェクトの値に設定します。

例

次の例は、タイプ Item の新規ビジネス・オブジェクトを作成し、これを配列 items の 4 番目のエレメントとして追加します。

```
items.setElementAt(5, new BusObj("Item"));
```

---

size()

このビジネス・オブジェクト配列内のエレメントの数を戻します。

構文

```
int size()
```

注記

すべての Java 配列と同様に、BusObjArray はゼロ相対配列です。したがって、size() メソッドは getLastIndex() メソッドより 1 大きい値を戻します。

例

次の例は、items のエレメントの数を戻します。

```
int size = items.size();
```

---

sum()

ビジネス・オブジェクト配列の中のすべてのビジネス・オブジェクトの、指定した属性の値を合計します。

構文

```
double sum(String attrName)
```

パラメーター

attr ビジネス・オブジェクト内の属性を参照する変数。指定できる属性のデータ型は、Integer、Float、または Double です。

戻り値

ビジネス・オブジェクトのリストからの、指定した属性の合計。

例外

UnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

UnsupportedAttributeTypeException - 指定した属性のデータ型が、『注記』セクションにあるサポート対象の属性データ型ではない場合です。

上記の例外すべては CollaborationException からサブクラス化されています。sum() メソッドは、これらの例外に対して例外タイプ AttributeException を設定できます。

例

```
double sumSalary = items.sum("Salary");
```

swap()

このビジネス・オブジェクト配列内の 2 つのビジネス・オブジェクトの位置を逆にします。この場合、配列の最初のエレメントが 0、2 番目が 1、3 番目が 2 であり、以下同様となることを念頭に置いてください。

構文

```
void swap(int index1, int index2)
```

パラメーター

index1 スワップしたい一方のエレメントの配列での位置。

index2 スワップしたいもう一方のエレメントの配列での位置。

例

次の例では、swap() を使用して、次の配列の BusObjA と BusObjC の位置を逆にします。

BusObjA	BusObjB	BusObjC

```
swap(0,2);
```

swap() 呼び出しの結果は、次の配列になります。

BusObjC	BusObjB	BusObjA

toString()

このビジネス・オブジェクト配列内の値を単一のストリングとして検索します。

構文

```
String toString()
```

例

次の例では、toString() を使用して items ビジネス・オブジェクト配列の内容を検索し、logInfo() を使用してその内容をログ・ファイルに書き込みます。

```
logInfo(items.toString());
```

第 12 章 CwDBConnection クラス

CwDBConnection クラスは、データベースで SQL 照会を実行するためのメソッドを提供します。照会は、接続プールから取得される接続を通じて実行されます。このクラスをインスタンス化するには、BaseDLM クラスで getDBConnection() を呼び出す必要があります。すべてのマップは BaseDLM から派生またはサブクラス化されるため、getDBConnection() へのアクセス権があります。

表 126 に CwDBConnection クラスのメソッドについて要約します。

表 126. CwDBConnection メソッドの要約

メソッド	説明	ページ
beginTransaction()	現行接続に対して明示的なトランザクションを開始します。	395
commit()	現行接続に関連付けられたアクティブなトランザクションをコミットします。	396
executeSQL()	構文とオプションのパラメーター配列を指定して、静的な SQL 照会を実行します。	399
executePreparedSQL()	構文とオプションのパラメーター配列を指定して、準備済みの SQL 照会を実行します。	398
executeStoredProcedure()	名前とパラメーター配列を指定して、SQL ストアード・プロシージャーを実行します。	401
getUpdateCount()	データベースへの最後の書き込み操作によって影響された行の数を戻します。	402
hasMoreRows()	照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。	402
inTransaction()	現行接続でトランザクションが進行中かどうかを判別します。	403
isActive()	現行接続がアクティブかどうかを判別します。	404
nextRow()	照会結果から次の行を検索します。	404
release()	現行接続の使用を解放して、接続プールに戻します。	405
rollBack()	現行接続に関連付けられたアクティブなトランザクションをロールバックします。	406

beginTransaction()

現行接続に対して明示的なトランザクションを開始します。

構文

```
void beginTransaction()
```

パラメーター

なし。

戻り値

なし。

例外

CwDBConnectionException - データベース・エラーが発生した場合です。

注記

beginTransaction() メソッドは、現行接続における新規の明示的なトランザクションの開始を示します。beginTransaction() メソッド、commit() メソッド、および rollBack() メソッドは、明示的なトランザクションに対してトランザクション境界を管理します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

beginTransaction() を使用して明示的なトランザクションの開始を指定しない 場合、データベースは、各 SQL ステートメントを個別トランザクションとして実行します。

要確認: 接続が明示的なトランザクション・ブラケットを使用する場合は、beginTransaction() のみを使用します。接続が暗黙的なトランザクション・ブラケットを使用する場合は、beginTransaction() を使用すると、CwDBTransactionException 例外が発生します。

明示的なトランザクションを開始する前に、BaseDLM クラスから getDBConnection() メソッドを使用して、CwDBConnection オブジェクトを作成する必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

例

次の例は、トランザクションを使用して、CustDBConnPool の中の接続に関連付けられているデータベースの表に行を挿入する照会を実行します。

```
CwDBConnection connection = getDBConnection("CustDBConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

参照項目

commit(), getDBConnection(), inTransaction(), rollBack()

---

commit()

現行接続に関連付けられたアクティブなトランザクションをコミットします。

構文

```
void commit()
```

パラメーター

なし。

戻り値

なし。

例外

CwDBConnectionException - データベース・エラーが発生した場合です。

注記

commit() メソッドは、現行接続に関連付けられているデータベースに加えられた変更をコミットすることによって、アクティブ・トランザクションを終了します。beginTransaction() メソッド、commit() メソッド、および rollBack() メソッドは、明示的なトランザクションに対してトランザクション境界を管理します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。

要確認: 接続が明示的なトランザクション・ブラケットを使用する場合は、commit() のみを使用します。接続が暗黙的なトランザクション・ブラケットを使用する場合は、commit() を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に commit() (または rollback()) を使用して明示的なトランザクションを終了していない場合、InterChange Server Express は、マップが成功したかどうかに基づいて暗黙的にトランザクションを終了します。マップが成功の場合、ICS は、このデータベース・トランザクションをコミットします。マップが成功しなかった 場合、ICS はデータベース・トランザクションを暗黙的にロールバックします。マップが成功かどうかにかかわらず、ICS は警告を記録します。

明示的なトランザクションを開始する前に、BaseDLM クラスから getDBConnection() メソッドを使用して、CwDBConnection オブジェクトを作成する必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

例

トランザクションのコミットの例については、beginTransaction() の例を参照してください。

参照項目

beginTransaction(), getDBConnection(), inTransaction(), rollBack()

executePreparedSQL()

構文とオプションのパラメーター配列を指定して、準備済みの SQL 照会を実行します。

構文

```
void executePreparedSQL(String query)
void executePreparedSQL(String query, Vector queryParameters)
```

パラメーター

query データベースで実行する SQL 照会のストリング表現。

queryParameters
SQL 照会のパラメーターに渡される引き数の Vector オブジェクト。

戻り値

なし。

例外

CwDBSQLException - データベース・エラーが発生した場合です。

注記

executePreparedSQL() メソッドは、指定した *query* ストリングを準備済み SQL ステートメントとして、現行接続に関連付けられたデータベースに送信します。この照会は、最初に実行されるときにストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL ステートメントを実行し、この準備済みステートメントを executePreparedSQL() に戻します。executePreparedSQL() メソッドは、この準備済みステートメントをメモリーに保管します。executePreparedSQL() は、何度も実行する必要がある SQL ステートメントに使用します。これに対して executeSQL() メソッドは、準備済みステートメントを保管しない ため、一度のみ実行すればよい照会の場合に有効です。

要確認: executePreparedSQL() を使用して照会を実行する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを生成することで、目的のデータベースへの接続を取得する必要があります。

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

  検索されたデータにアクセスするには、hasMoreRows() メソッドと nextRow() メソッドを使用します。
- データベース内のデータを変更する SQL ステートメント
  - INSERT

– DELETE
– UPDATE

接続が明示的なトランザクション・ブラケットを使用する場合は、beginTransaction() を使用して各トランザクションを明示的に開始し、commit() または rollback() を使用してトランザクションを終了する必要があります。

- 準備済みストアード・プロシージャーを実行する CALL ステートメント (ただしこのストアード・プロシージャーでは OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、executeStoredProcedure() メソッドを使用してください。

参照項目

beginTransaction(), commit(), executeSQL(), executeStoredProcedure(), getDBConnection(), hasMoreRows(), nextRow(), rollBack()

---

executeSQL()

構文とオプションのパラメーター配列を指定して、静的な SQL 照会を実行します。

構文

```
void executeSQL(String query)
void executeSQL(String query, Vector queryParameters)
```

パラメーター

query データベースで実行する SQL 照会のストリング表現。

queryParameters
SQL 照会のパラメーターに渡される引き数の Vector オブジェクト。

戻り値

なし。

例外

CwDBSQLException - データベース・エラーが発生した場合です。

注記

executeSQL() メソッドは、指定した *query* ストリングを静的な SQL ステートメントとして、現行接続に関連付けられたデータベースに送信します。この照会はストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式にコンパイルし、SQL ステートメントを実行します。この場合に、実行可能形式は保管されません。executeSQL() は、一度のみ実行する必要がある SQL ステートメントに使用します。これに対して executePreparedSQL() メソッドは、実行可能形式 (準備済みステートメントと呼ばれる) を保管するため、何度も実行する必要がある照会の場合に有効です。

要確認: executeSQL() を使用して照会を実行する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを生成することで、目的のデータベースへの接続を取得する必要があります。

実行できる SQL ステートメントを以下に示します (必要なデータベース許可がある場合)。

- 1 つ以上のデータベース表からデータを要求する SELECT ステートメント

  検索されたデータにアクセスするには、hasMoreRows() メソッドと nextRow() メソッドを使用します。
- データベース内のデータを変更する SQL ステートメント
  - INSERT
  - DELETE
  - UPDATE

  接続が明示的なトランザクション・ブラケットを使用する場合は、beginTransaction() を使用して各トランザクションを明示的に開始し、commit() または rollback() を使用してトランザクションを終了する必要があります。
- ストアード・プロシージャーを静的に実行する CALL ステートメント (ただしこのストアード・プロシージャーでは OUT パラメーターを使用できません)

  OUT パラメーターを使用してストアード・プロシージャーを実行するには、executeStoredProcedure() メソッドを使用してください。

例

次の例は、接続が AccntConnPool 接続プール内にあるアカウンティング・データベースに、行を挿入する照会を実行します。

```
CwDBConnection connection = getDBConnection("AccntConnPool");

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
connection.commit();

// Release the database connection
connection.release();
```

関係表からデータを選択する完全なコード・サンプルは、以下を参照してください。

参照項目

executePreparedSQL(), executeStoredProcedure(), getDBConnection(), hasMoreRows(), nextRow()

executeStoredProcedure()

名前とパラメーター配列を指定して、SQL ストアード・プロシージャーを実行します。

構文

```
void executeStoredProcedure(String storedProcedure,
                  Vector storedProcParameters)
```

パラメーター

storedProcedure
データベースで実行する SQL ストアード・プロシージャーの名前。

storedProcParameters
ストアード・プロシージャーに渡されるパラメーターの Vector オブジェクト。各パラメーターは、CwDBStoredProcedureParam クラスのインスタンスです。

戻り値

なし。

例外

CwDBSQLException - データベース・エラーが発生した場合です。

注記

executeStoredProcedure() メソッドは、指定した *storedProcedure* への呼び出しを、現行接続に関連付けられたデータベースに送信します。このメソッドは、ストアード・プロシージャー呼び出しを準備済み SQL ステートメントとして送信します。つまり、このストアード・プロシージャー呼び出しは、最初に実行されるときにストリングとしてデータベースに送信されます。データベースは、このストリングを実行可能形式 (準備済みステートメントと呼ばれる) にコンパイルし、SQL ステートメントを実行し、この準備済みステートメントを executeStoredProcedure() に戻します。executeStoredProcedure() メソッドは、この準備済みステートメントをメモリーに保管します。

要確認: executeStoredProcedure() を使用してストアード・プロシージャーを実行する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを作成する必要があります。

ストアード・プロシージャーが戻すデータを処理するには、hasMoreRows() メソッドと nextRow() メソッドを使用します。

ストアード・プロシージャーに OUT パラメーターが含まれていない 限り、executeSQL() メソッドまたは executePreparedSQL() メソッドを使用して、このストアード・プロシージャーを実行することもできます。ストアード・プロシージャーが OUT パラメーターを使用する場合、executeStoredProcedure() を使用して実行する必要があります。executeSQL() または executePreparedSQL() の場合と異な

り、ストアード・プロシージャーを実行するのに全 SQL ステートメントを渡す必要はありません。executeStoredProcedure() を使用する場合、ストアード・プロシージャーの名前と、CwDBStoredProcedureParam オブジェクトの Vector パラメーター配列のみを渡す必要があります。executeStoredProcedure() メソッドは、*storedProcParameters* 配列からパラメーターの数を判断し、ストアード・プロシージャーの呼び出しステートメントを構築することができます。

参照項目

executePreparedSQL(), executeSQL(), getDBConnection(), hasMoreRows(), nextRow()

---

getUpdateCount()

データベースへの最後の書き込み操作によって影響された行の数を戻します。

構文

```
int getUpdateCount()
```

パラメーター

なし。

戻り値

最後の書き込み操作によって影響された行の数を表す int を戻します。

例外

CwDBConnectionException - データベース・エラーが発生した場合です。

注記

getUpdateCount() メソッドは、現行接続に関連付けられているデータベース内で、直近の更新操作によって変更された行の数を示します。このメソッドは、UPDATE または INSERT ステートメントをデータベースに送信した後に、その SQL ステートメントによって影響された行の数を判別する必要がある場合に役立ちます。

要確認: このメソッドを使用する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを作成し、CwDBConnection クラスから executeSQL() メソッドまたは executePreparedSQL() メソッドを使用してデータベースを更新する照会を送信する必要があります。

参照項目

executePreparedSQL(), executeSQL(), getDBConnection()

---

hasMoreRows()

照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。

構文

```
boolean hasMoreRows()
```

パラメーター

なし。

戻り値

行がさらに存在する場合は、true を戻します。

例外

CwDBSQLException - データベース・エラーが発生した場合です。

注記

hasMoreRows() メソッドは、現行接続に関連付けられている照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。このような照会には、SELECT ステートメントとストアード・プロシージャーが含まれます。一度に接続に関連付けることのできる照会は 1 つのみです。このため、hasMoreRows() が false を戻す前に別の照会を実行すると、最初の照会のデータが失われます。

参照項目

executePreparedSQL(), executeSQL(), nextRow()

---

inTransaction()

現行接続でトランザクションが進行中かどうかを判別します。

構文

```
boolean inTransaction()
```

パラメーター

なし。

戻り値

現行接続でトランザクションが現在アクティブな場合は true を戻し、それ以外の場合は false を戻します。

例外

CwDBConnectionException - データベース・エラーが発生した場合です。

注記

inTransaction() メソッドは、現行接続にアクティブ・トランザクション (開始されたが終了していないトランザクション) が存在するかどうかを示す boolean 値を戻します。

要確認: トランザクションを開始する前に、BaseDLM クラスから getDBConnection() メソッドを使用して CwDBConnection オブジェクトを作成する必要があります。

参照項目

beginTransaction(), commit(), getDBConnection(), rollBack()

---

isActive()

現行接続がアクティブかどうかを判別します。

構文

```
boolean isActive()
```

パラメーター

なし。

戻り値

現行接続がアクティブな場合は true を戻し、この接続が解放されている場合は false を戻します。

例外

なし。

参照項目

getDBConnection(), release()

---

nextRow()

照会結果から次の行を検索します。

構文

```
Vector nextRow()
```

パラメーター

なし。

戻り値

照会結果の次の行を Vector オブジェクトとして戻します。

例外

CwDBSQLException - データベース・エラーが発生した場合です。

注記

nextRow() メソッドは、現行接続に関連付けられている照会結果からデータの 1 行を戻します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。このような照会には、SELECT ステートメントとストアード・プロシージャーが含まれます。一度に接続に関連付けることのできる照会は 1 つのみです。このため、nextRow() がデータの最後の行を戻す前に別の照会を実行すると、最初の照会の結果が失われます。

参照項目

hasMoreRows(), executePreparedSQL(), executeSQL(), executeStoredProcedure()

---

release()

現行接続の使用を解放して、接続プールに戻します。

構文

```
void release()
```

パラメーター

なし。

戻り値

なし。

例外

CwDBConnectionException

注記

release() メソッドは、マップ・インスタンスによる現行接続の使用を明示的に解放します。解放された接続はその接続プールに戻されます。関連付けられているデータベースへの接続を必要とする他のコンポーネント (マップまたはコラボレーション) は、この接続を使用できるようになります。接続を明示的に解放しないと、現行マップの実行終了時に、マップ・インスタンスによって暗黙的に接続が解放されます。このため、静的変数の中に接続を保管して再使用することはできません。

重要: トランザクションが現在アクティブな場合は、release() メソッドを使用しないでください。暗黙的なトランザクション・ブラケットの場合、ICS は、マップが成功または失敗のいずれかを確認するまでデータベース・トランザクションを終了しません。このため、暗黙的なトランザクション・ブラケットを使用する接続にこのメソッドを使用すると、CwDBTransactionException 例外が発生します。この例外を明示的に処理しないと、アクティブなトランザクションも自動的にロールバックされます。inTransaction() メソッドを使用してトランザクションがアクティブかどうかを判別することもできます。

参照項目

getDBConnection(), inTransaction(), isActive()

---

rollBack()

現行接続に関連付けられたアクティブなトランザクションをロールバックします。

構文

```
void rollBack()
```

パラメーター

なし。

戻り値

なし。

例外

CwDBConnectionException - データベース・エラーが発生した場合です。

注記

rollback() メソッドは、現行接続に関連付けられているデータベースに加えられた変更をロールバックすることによって、アクティブ・トランザクションを終了します。beginTransaction() メソッド、commit() メソッド、および rollBack() メソッドは、明示的なトランザクションに対してトランザクション境界を管理します。このトランザクションには、SQL 照会 (SQL ステートメント INSERT、DELETE、または UPDATE を含む) と、これらの SQL ステートメントの 1 つを含むストアード・プロシージャーとが含まれます。ロールバックが失敗すると、rollback() は CwDBTransactionException 例外をスローして、エラーを記録します。

要確認: 接続が明示的なトランザクション・ブラケットを使用する場合は、rollback() のみを使用します。接続が暗黙的なトランザクション・ブラケットを使用する場合は、rollback() を使用すると、CwDBTransactionException 例外が発生します。接続が解放される前に rollback() (または commit()) を使用して明示的なトランザクションを終了しない場合、マップが成功したかどうかに基づいて InterChange Server Express によって暗黙的にトランザクションが終了されます。マップが成功の場合、ICS は、このデータベース・トランザクションをコミットします。マップが成功しなかった 場合、ICS はデータベース・トランザクションを暗黙的にロールバックします。マップが成功かどうかにかかわらず、ICS は警告を記録します。

明示的なトランザクションを開始する前に、BaseDLM クラスから getDBConnection() メソッドを使用して、CwDBConnection オブジェクトを作成する必要があります。この接続では、明示的なトランザクション・ブラケットを必ず使用してください。

参照項目

beginTransaction(), commit(), getDBConnection(), inTransaction()

第 13 章 CwDBStoredProcedureParam クラス

CwDBStoredProcedureParam オブジェクトは、ストアード・プロシージャーの単一のパラメーターについて説明します。表 127 に、CwDBStoredProcedureParam クラスのメソッドについて要約します。

表 127. CwDBStoredProcedureParam メソッドの要約

メソッド	説明	ページ
CwDBStoredProcedureParam()	ストアード・プロシージャーのパラメーターに関する引き数情報を保持する、CwDBStoredProcedureParam の新しいインスタンスを作成します。	409
getParamType()	現行ストアード・プロシージャー・パラメーターのイン/アウト・タイプを整数定数として検索します。	410
getValue()	現行のストアード・プロシージャー・パラメーターの値を検索します。	411

CwDBStoredProcedureParam()

ストアード・プロシージャーのパラメーターに関する引き数情報を保持する、CwDBStoredProcedureParam の新しいインスタンスを作成します。

構文

```
CwDBStoredProcedureParam(int paramType, String paramValue);

CwDBStoredProcedureParam(int paramType, int paramValue);
CwDBStoredProcedureParam(int paramType, Integer paramValue);
CwDBStoredProcedureParam(int paramType, Long paramValue);

CwDBStoredProcedureParam(int paramType, double paramValue);
CwDBStoredProcedureParam(int paramType, Double paramValue);
CwDBStoredProcedureParam(int paramType, float paramValue);
CwDBStoredProcedureParam(int paramType, Float paramValue);
CwDBStoredProcedureParam(int paramType, BigDecimal paramValue);

CwDBStoredProcedureParam(int paramType, boolean paramValue);
CwDBStoredProcedureParam(int paramType, Boolean paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Date paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Time paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Timestamp paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Blob paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Clob paramValue);

CwDBStoredProcedureParam(int paramType, byte[] paramValue);
CwDBStoredProcedureParam(int paramType, Array paramValue);
CwDBStoredProcedureParam(int paramType, Struct paramValue);
```

パラメーター

paramType 関連するストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプ。

paramValue　ストアード・プロシージャーに送信される引き数値。この値は、以下の Java データ型のいずれかになります。

戻り値

ストアード・プロシージャーの宣言内の 1 つの引き数の引き数情報を保持するための、新規 CwDBStoredProcedureParam オブジェクトを戻します。

例外

なし。

注記

CwDBStoredProcedureParam() コンストラクターは CwDBStoredProcedureParam インスタンスを作成して、ストアード・プロシージャーの 1 つのパラメーターを説明します。この場合のパラメーター情報は以下のとおりです。

- パラメーターのイン/アウト・タイプ

  パラメーターのイン/アウト・タイプは、コンストラクターの最初の引き数によって初期化されます。パラメーターの有効なイン/アウト・タイプのリストについては、表 128 を参照してください。

- パラメーター値

  このパラメーター値は、コンストラクターの 2 番目の引き数によって初期化されます。CwDBStoredProcedureParam クラスには、サポートされるパラメーター値のデータ型ごとにコンストラクターの 1 つの形式が用意されています。

executeStoredProcedure() メソッドに渡すストアード・プロシージャー・パラメーターの Java Vector は、開発者が用意する必要があります。このメソッドは、ストアード・プロシージャーの名前とパラメーター・ベクターからストアード・プロシージャー呼び出しを作成し、現行接続に関連付けられているデータベースにこの呼び出しを送信します。

参照項目

executeStoredProcedure()

---

getParamType()

現行ストアード・プロシージャー・パラメーターのイン/アウト・タイプを整数定数として検索します。

構文

```
int getParamType()
```

パラメーター

なし。

戻り値

関連する CwDBStoredProcedureParam パラメーターのイン/アウト・タイプを戻します。

例外

なし。

注記

getParamType() メソッドは、現行ストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを戻します。イン/アウト・パラメーター・タイプは、ストアード・プロシージャーがパラメーターをどのように使用するかを示します。CwDBStoredProcedureParam クラスは、表 128 に示すように、各イン/アウト・タイプを定数として表します。

表 128. パラメーターのイン/アウト・タイプ

パラメーターのイン/アウト・タイプ	説明	イン/アウト・タイプ定数
IN パラメーター	IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、値を戻すためにそのパラメーターを使用しません。	PARAM_IN
OUT パラメーター	OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーはその値を入力として読み取るのではなく、値を戻すためにそのパラメーターを使用します。	PARAM_OUT
INOUT パラメーター	INOUT パラメーターは*入力および出力* です。つまり、ストアード・プロシージャーはその値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。	PARAM_INOUT

参照項目

CwDBStoredProcedureParam(), getValue()

---

getValue()

現行のストアード・プロシージャー・パラメーターの値を検索します。

構文

```
Object getValue()
```

パラメーター

なし。

戻り値

関連する CwDBStoredProcedureParam パラメーターの値を Java Object として戻します。

例外

なし。

注記

getValue() メソッドは、パラメーター値を Java Object (Integer、Double、または String など) として戻します。OUT パラメーターに戻された値が JDBC NULL の場合、getParamValue() は null 定数を戻します。

参照項目

CwDBStoredProcedureParam(), getParamType()

第 14 章 DtpConnection クラス

DtpConnection クラスは、Data Transformation Package (DTP) の一部です。このクラスは、リレーションシップ・データベースで SQL 照会を実行するためのメソッドを提供します。このクラスをインスタンス化するには、BaseDLM クラスで getRelConnection() を呼び出す必要があります。すべてのマップは BaseDLM から派生またはサブクラス化されるため、getRelConnection() へのアクセス権があります。

要確認: DtpConnection クラスとそのメソッドは、後方互換性のためにのみ サポートされています。これらの*使用すべきでない*メソッド によりエラーが生成されることはありませんが、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。新しいマップ開発では、CwDBConnection クラスとそのメソッドを使用して、データベース接続を確立します。

表 129 に、DtpConnection クラスのメソッドについて要約します。

表 129. DtpConnection メソッドの要約

メソッド	説明	ページ
beginTran()	リレーションシップ・データベースに対して SQL トランザクションを開始します。	413
commit()	リレーションシップ・データベースで現行のトランザクションをコミットします。	414
executeSQL()	CALL ステートメントを指定することで、リレーションシップ・データベースで SQL 照会を実行します。	415
execStoredProcedure()	名前とパラメーター配列を指定して、リレーションシップ・データベースで SQL ストアード・プロシージャーを実行します。	416
getUpdateCount()	リレーションシップ・データベースへの最後の書き込み操作によって影響された行の数を戻します。	417
hasMoreRows()	照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。	418
inTransaction()	リレーションシップ・データベースでトランザクションが進行中かどうかを判別します。	418
nextRow()	照会結果ベクトルの次の行を検索します。	419
rollBack()	リレーションシップ・データベースで現行のトランザクションをロールバックします。	419

beginTran()

リレーションシップ・データベースに対して SQL トランザクションを開始します。

構文

```
void beginTran()
```

パラメーター

なし。

戻り値

なし。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

beginTran() メソッド、commit() メソッド、および rollBack() メソッドは、SQL 照会のトランザクションをサポートします。

トランザクションを開始する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

例

次の例では、トランザクションを使用して、SapCust 関係の中の表に行を挿入する照会を実行します。

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();
```

参照項目

commit(), getRelConnection(), inTransaction(), rollBack()

---

commit()

リレーションシップ・データベースで現行のトランザクションをコミットします。

構文

```
void commit()
```

パラメーター

なし。

戻り値

なし。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

beginTran() メソッド、commit() メソッド、および rollBack() メソッドは、SQL 照会のトランザクションをサポートします。

トランザクションを開始する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

例

次の例では、トランザクションを使用して、SapCust 関係の中の表に行を挿入する照会を実行します。

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();
```

参照項目

beginTran(), getRelConnection(), inTransaction(), rollBack()

---

executeSQL()

CALL ステートメントを指定することで、リレーションシップ・データベースで SQL 照会を実行します。

構文

```
void executeSQL(String query)
void executeSQL(String query, Vector queryParameters)
```

パラメーター

query リレーションシップ・データベースで実行される SQL 照会。

queryParameters
SQL 照会のパラメーターに渡される引き数の Vector オブジェクト。

戻り値

なし。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

executeSQL() を使用して照会を実行する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

実行できる SQL ステートメントは、INSERT、SELECT、DELETE、および UPDATE です。ストアード・プロシージャーを実行することもできます (ただし、このストアード・プロシージャーでは OUT パラメーターを使用できません)。OUT パラメーターを使用してストアード・プロシージャーを実行するには、execStoredProcedure() メソッドを使用してください。

例

次の例では、SapCust 関係の中の表に行を挿入する照会を実行します。

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();

// release the database connection
releaseRelConnection(true);
```

参照項目

execStoredProcedure(), getRelConnection(), hasMoreRows(), nextRow()

---

execStoredProcedure()

名前とパラメーター配列を指定して、リレーションシップ・データベースで SQL ストアード・プロシージャーを実行します。

構文

```
void execStoredProcedure(String storedProcedure,
   Vector storedProcParameters)
```

パラメーター

storedProcedure
リレーションシップ・データベースで実行する SQL ストアード・プロシージャーの名前。

storedProcParameters
ストアード・プロシージャーに渡されるパラメーターの Vector オブジェクト。各パラメーターは、UserStoredProcedureParam クラスのインスタンスです。

戻り値

なし。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

execStoredProcedure() を使用してストアード・プロシージャーを実行する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

このストアード・プロシージャーに OUT パラメーターが含まれていない限り、executeSQL() メソッドを使用してストアード・プロシージャーを実行することもできます。ストアード・プロシージャーが OUT パラメーターを使用する場合、execStoredProcedure() を使用して実行する必要があります。executeSQL() の場合と異なり、ストアード・プロシージャーを実行するのに全 SQL ステートメントを渡す必要はありません。execStoredProcedure() を使用する場合、ストアード・プロシージャーの名前と、UserStoredProcedureParam オブジェクトの Vector パラメーター配列のみを渡す必要があります。execStoredProcedure() メソッドは、*storedProcParameters* 配列からパラメーターの数を判断し、ストアード・プロシージャーの呼び出しステートメントを構築することができます。

参照項目

executeSQL(), getRelConnection(), hasMoreRows(), nextRow()

---

getUpdateCount()

リレーションシップ・データベースへの最後の書き込み操作によって影響された行の数を戻します。

構文

```
int getUpdateCount()
```

パラメーター

なし。

戻り値

最後の書き込み操作によって影響された行の数を表す int を戻します。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

このメソッドを使用する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

このメソッドは、データベースで UPDATE または INSERT ステートメントを送信した後に、その SQL ステートメントによって影響された行の数を判別する必要がある場合に役立ちます。

参照項目

executeSQL(), getRelConnection()

---

hasMoreRows()

照会結果に、処理する必要のある行がさらに存在するかどうかを判別します。

構文

```
boolean hasMoreRows()
```

パラメーター

なし。

戻り値

行がさらに存在する場合は、true を戻します。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

hasMoreRows() メソッドは、現行のリレーションシップ・データベースに関連付けられている照会に、処理する必要のある行がさらに存在するかどうかを判別します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。このような照会には、SELECT ステートメントとストアード・プロシージャーが含まれます。一度に接続に関連付けることのできる照会は 1 つのみです。このため、hasMoreRows() が false を戻す前に別の照会を実行すると、最初の照会のデータが失われます。

参照項目

nextRow(), executeSQL(), getUpdateCount()

---

inTransaction()

リレーションシップ・データベースでトランザクションが進行中かどうかを判別します。

構文

```
boolean inTransaction()
```

パラメーター

なし。

戻り値

トランザクションが進行中の場合は、"True" を戻します。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

トランザクションを開始する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

参照項目

beginTran(), commit(), getRelConnection(), rollBack()

---

nextRow()

照会結果ベクトルの次の行を検索します。

構文

```
Vector nextRow()
```

パラメーター

なし。

戻り値

照会結果の次の行を Vector オブジェクトとして戻します。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

nextRow() メソッドは、現行のリレーションシップ・データベースに関連付けられている照会からデータの 1 行を戻します。このメソッドは、データを戻す照会から結果を検索するときに使用してください。このような照会には、SELECT ステートメントとストアード・プロシージャーが含まれます。一度に接続に関連付けることのできる照会は 1 つのみです。このため、nextRow() がデータの最後の行を戻す前に別の照会を実行すると、最初の照会のデータが失われます。

参照項目

hasMoreRows(), executeSQL(), getUpdateCount()

---

rollBack()

リレーションシップ・データベースで現行のトランザクションをロールバックします。

構文

```
void rollBack()
```

パラメーター

なし。

戻り値

なし。

例外

DtpConnectionException - データベース・エラーが発生した場合です。

注記

beginTran() メソッド、commit() メソッド、および rollBack() メソッドは、SQL 照会のトランザクションをサポートします。

トランザクションを開始する前に、BaseDLM クラスから getRelConnection() メソッドを使用して DtpConnection オブジェクトを作成する必要があります。

参照項目

beginTran(), commit(), getRelConnection(), inTransaction()

第 15 章 DtpDataConversion クラス

ビジネス・オブジェクト・マッピングで最も一般的なタスクの 1 つは、属性値をあるデータ型から別のデータ型に変換することです。このプロセスは、データ変換 と呼ばれます。 DtpDataConversion クラスは、データ変換を実行するための単純な方法を提供します。

java.lang パッケージのデータ型クラスには変換メソッドが含まれていますが、可能な変換がすべてサポートされているわけではありません。 DtpDataConversion クラスは、多数のデータ変換メソッドを 1 つのクラスに集約して、マップで実行される最も一般的な変換をサポートします。getType() メソッドと isOKToConvert() メソッドにより、特定の変換が可能かどうかが簡単に判別できます。

このクラスのメソッドはすべて、静的として宣言されます。表 130 に、DtpDataConversion クラスのメソッドについて要約します。

表 130. DtpDataConversion メソッドの要約

メソッド	説明	ページ
getType()	値のデータ型を判別します。	421
isOKToConvert()	値をあるデータ型から別のデータ型に変換できるかどうかを判別します。	423
toBoolean()	Java オブジェクトを Boolean オブジェクトに変換します。	425
toDouble()	オブジェクト・データ型または基本データ型を Double オブジェクトに変換します。	425
toFloat()	オブジェクト・データ型または基本データ型を Float オブジェクトに変換します。	426
toInteger()	オブジェクト・データ型または基本データ型を Integer オブジェクトに変換します。	427
toPrimitiveBoolean()	String オブジェクトまたは Boolean オブジェクトを基本的な boolean データ型に変換します。	427
toPrimitiveDouble()	オブジェクト・データ型または基本データ型を基本的な double データ型に変換します。	428
toPrimitiveFloat()	オブジェクト・データ型または基本データ型を基本的な float データ型に変換します。	429
toPrimitiveInt()	オブジェクト・データ型または基本データ型を基本的な int データ型に変換します。	429
toString()	オブジェクト・データ型または基本データ型を String オブジェクトに変換します。	430

getType()

値のデータ型を判別します。

構文

```
int getType(Object objectData)
int getType(int integerData)
int getType(float floatData)
int getType(double doubleData)
int getType(boolean booleanData)
```

パラメーター

objectData 任意の Java オブジェクト。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

booleanData 任意の基本的な boolean 変数。

戻り値

受け渡すパラメーターのデータ型を表す整数を戻します。DtpDataConversion クラスで静的および最終として宣言された次の定数の 1 つと比較することで、戻り値を解釈できます。

INTEGER_TYPE データは基本的な int 値または Integer オブジェクトです。

STRING_TYPE データは String オブジェクトです。

FLOAT_TYPE データは基本的な float 値または Float オブジェクトです。

DOUBLE_TYPE データは基本的な double 値または Double オブジェクトです。

BOOL_TYPE データは基本的な boolean 値または Boolean オブジェクトです。

DATE_TYPE データは Date オブジェクトです。

LONGTEXT_TYPE データは LongText オブジェクトです。

UNKNOWN_TYPE データ型は不明です。

例外

なし。

注記

OKToConvert() メソッドの getType() から戻り値を使用して、2 つのデータ型間で変換が可能かどうかを判別できます。

例

```
int conversionStatus = DtpDataConversion.isOKToConvert(
   DtpDataConversion.getType(srcObject),
   DtpDataConversion.getType(destObject));

switch(conversionStatus)
   {
   case DtpDataConversion.OKTOCONVERT:
      // go ahead and convert
      break;
   case DtpDataConversion.POTENTIALDATALOSS:
```

```
      // convert, then check value
      break;
   case DtpDataConversion.CANNOTCONVERT:
      // return an error
      break;
}
```

参照項目

isOKToConvert()

---

isOKToConvert()

値をあるデータ型から別のデータ型に変換できるかどうかを判別します。

構文

```
int isOKToConvert(int srcDatatype, int destDataType)
int isOKToConvert(String srcDataTypeStr, String destDataTypeStr)
```

パラメーター

srcDataType getType() によって戻される整数です。変換したいソース値のデータ型を表します。

destDataType getType() によって戻される整数です。ソース値を変換したいデータ型を表します。

srcDataTypeStr
変換したいソース値のデータ型の名前を含むストリング。使用可能な値は、Boolean、boolean、Double、double、Float、float、Integer、int、および String です。

destDataTypeStr
ソース値を変換したいデータ型の名前を含むストリング。使用可能な値は、Boolean、boolean、Double、double、Float、float、Integer、int、および String です。

戻り値

ソース・データ型の値を宛先データ型の値に変換できるかどうかを指定する整数を戻します。次の定数の 1 つと比較することで、戻り値を解釈できます。定数は、DtpDataConversion クラスで静的および最終として宣言されます。

OKTOCONVERT ソース・データ型から宛先データ型に変換できます。

POTENTIALDATALOSS
変換はできますが、ソース値に変換不可能な文字が含まれる場合や、宛先データ型に合わせて切り捨てる必要がある場合は、データが失われる可能性があります。

CANNOTCONVERT ソース・データ型を宛先データ型に変換することはできません。

例外

なし。

注記

getType() メソッドは、パラメーターとして渡す値のデータ型を表す整数を戻します。2 つの属性間でデータ変換が可能かどうかを判別するには、isOKToConvert() の第 1 の形式と getType() を使用します。*srcDataType* パラメーターと *destDataType* パラメーターを生成するには、isOKToConvert() メソッドを呼び出すときに、ソース属性と宛先属性の両方で getType() を使用します。

メソッドの第 2 の形式では、ソース・データと宛先データのデータ型の名前を含む String 値を受け入れます。データ型がわかっていて、変換を実行できるかどうかを検査する場合は、この形式のメソッドを使用します。

表 131 に、ソース・データ型と宛先データ型の組み合わせごとに可能な変換を示します。表での意味は、次のとおりです。

- OK は、ソース・タイプから宛先タイプに変換できることを示します。データ損失はありません。
- DL は、変換はできますが、ソース値に変換不可能な文字が含まれる場合や、宛先タイプに合わせて切り捨てる必要がある場合に、データ損失が発生する可能性があることを示します。
- NO は、値をソース・データ型から宛先データ型に変換できないことを示します。

表 131. データ型間で可能な変換

ソース	宛先 int、Integer	String	float、Float	double、Double	boolean Boolean	Date	Longtext
int Integer	OK	OK	OK	OK	NO	NO	OK
String	DL[1]	OK	DL[1]	DL[1]	DL[2]	DL	OK
float、Float	DL[3]	OK	OK	OK	NO	NO	OK
double、Double	DL[3]	OK	DL[3]	OK	NO	NO	OK
boolean、Boolean	NO	OK	NO	NO	OK	NO	OK
Date	NO	OK	NO	NO	NO	OK	OK
Longtext	DL[1]	DL[3]	DL[1]	DL[1]	DL[2]	DL	OK

[1]String または Longtext 値を数値型に変換する場合、String または LongText 値には、数値と小数のみ含めることができます。変換の前に、String または LongText 値から通貨記号などのその他の文字を除去する必要があります。除去しないと、DtpIncompatibleFormatException がスローされます。
[2]String または Longtext 値を Boolean に変換する場合、String または Longtext の値は 「true」または「false」でなければなりません。「true」以外のストリング (大文字小文字は関係ありません) は、false と見なされます。
[3] ソース・データ型は宛先データ型より大きい精度をサポートしているため、値が切り捨てられる可能性があります。

例

```
if (DtpDataConversion.isOKToConvert(getType(mySource),
      getType(myDest))== DtpDataConversion.OKTOCONVERT)
   // map these attributes
else
   // skip these attributes
```

参照項目

getType()

---

toBoolean()

Java オブジェクトを Boolean オブジェクトに変換します。

構文

```
Boolean toBoolean(Object objectData)
Boolean toBoolean(boolean booleanData)
```

パラメーター

objectData Boolean に変換したい Java オブジェクト。現在サポートされているオブジェクトは、String のみです。

booleanData 任意の基本的な boolean 変数。

戻り値

Boolean オブジェクトを戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を Boolean に変換できない場合です。

例

```
Boolean MyBooleanObj = DtpDataConversion.toBoolean(MyStringObj);
```

参照項目

getType(), isOKToConvert(), toPrimitiveBoolean()

---

toDouble()

オブジェクト・データ型または基本データ型を Double オブジェクトに変換します。

構文

```
Double toDouble(Object objectData)
Double toDouble(int integerData)
Double toDouble(float floatData)
Double toDouble(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Float、Integer、および String です。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

戻り値

Double オブジェクトを戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を Double に変換できない場合です。

例

```
Double myDoubleObj = DtpDataConversion.toDouble(myInteger);
```

参照項目

getType(), isOKToConvert(), toPrimitiveDouble()

---

toFloat()

オブジェクト・データ型または基本データ型を Float オブジェクトに変換します。

構文

```
Float toFloat(Object objectData)
Float toFloat(int integerData)
Float toFloat(float floatData)
Float toFloat(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Integer、および String です。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

戻り値

Float オブジェクトを戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を Float に変換できない場合です。

例

```
Float myFloatObj = DtpDataConversion.toFloat(myInteger);
```

参照項目

getType(), isOKToConvert(), toPrimitiveFloat()

---

toInteger()

オブジェクト・データ型または基本データ型を Integer オブジェクトに変換します。

構文

```
Integer toInteger(Object objectData)
Integer toInteger(int integerData)
Integer toInteger(float floatData)
Integer toInteger(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Float、および String です。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

戻り値

Integer オブジェクトを戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を Integer に変換できない場合です。

例

```
Integer myIntegerObj = DtpDataConversion.toInteger(myFloat);
```

参照項目

getType(), isOKToConvert(), toPrimitiveInt()

---

toPrimitiveBoolean()

String オブジェクトまたは Boolean オブジェクトを基本的な boolean データ型に変換します。

構文

```
boolean toPrimitiveBoolean(Object objectData)
```

パラメーター

objectData 基本的な boolean データ型に変換したい String または Boolean オブジェクト。

戻り値

基本的な boolean 値を戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を boolean に変換できない場合です。

例

```
boolean MyBoolean = DtpDataConversion.toPrimitiveBoolean(MyStringObj);
```

参照項目

getType(), isOKToConvert(), toBoolean()

---

toPrimitiveDouble()

オブジェクト・データ型または基本データ型を基本的な double データ型に変換します。

構文

```
double toPrimitiveDouble(Object objectData)
double toPrimitiveDouble(int integerData)
double toPrimitiveDouble(float floatData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Float、Integer、および String です。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

戻り値

基本的な double 値を戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を double に変換できない場合です。

例

```
double myDouble = DtpDataConversion.toPrimitiveDouble(myObject);
```

参照項目

getType(), isOKToConvert(), toDouble()

---

toPrimitiveFloat()

オブジェクト・データ型または基本データ型を基本的な float データ型に変換します。

構文

```
float toPrimitiveFloat(Object objectData)
float toPrimitiveFloat(int integerData)
float toPrimitiveFloat(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Float、Integer、および String です。

integerData 任意の基本的な int 変数。

doubleData 任意の基本的な double 変数。

戻り値

基本的な float 値を戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を float に変換できない場合です。

例

```
float myFloat = DtpDataConversion.toPrimitiveFloat(myInteger);
```

参照項目

getType(), isOKToConvert(), toFloat()

---

toPrimitiveInt()

オブジェクト・データ型または基本データ型を基本的な int データ型に変換します。

構文

```
int toPrimitiveInteger(Object objectData)
int toPrimitiveInteger(float floatData)
int toPrimitiveInteger(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Float、Integer、および String です。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

戻り値

基本的な int 値を戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を integer に変換できない場合です。

例

```
int myInt = DtpDataConversion.toPrimitiveInt(myObject);
```

参照項目

getType(), isOKToConvert(), toInteger()

---

toString()

オブジェクト・データ型または基本データ型を String オブジェクトに変換します。

構文

```
String toString(Object objectData)
String toString(int integerData)
String toString(float floatData)
String toString(double doubleData)
```

パラメーター

objectData Java オブジェクト。現在サポートされているオブジェクトは、Double、Float、および Integer です。

integerData 任意の基本的な int 変数。

floatData 任意の基本的な float 変数。

doubleData 任意の基本的な double 変数。

戻り値

String オブジェクトを戻します。

例外

DtpIncompatibleFormatException - ソース・データ型を String に変換できない場合です。

例

```
String myString = DtpDataConversion.toString(myObject);
```

参照項目

getType(), isOKToConvert()

第 16 章 DtpDate クラス

DtpDate クラスは、時刻と日付の値を比較して、形式を指定し、時刻と日付の値のコンポーネントを戻します。

静的 (クラス) メソッドは、クラス名に対して操作を行います。静的メソッドは、ビジネス・オブジェクトのセットを取得して、最も早い/最も遅い日付または最も早い/最も遅い日付を含むビジネス・オブジェクトを戻します。

インスタンス・メソッドは、日付オブジェクトに対して操作を行います。DtpDate コンストラクターに日付値を渡すと、その結果の日付オブジェクトを操作できます。インスタンス・メソッドにより、日付に関連付けられた値を検索、形式設定、および変更できます。日付を処理したい形式を設定することもできます。

データ変換メソッドは、あるアプリケーションがある形式で日付を保管し、別のアプリケーションが別の形式で日付を保管する場合に有効です。例えば、SAP は日付を 26/8/1999 15:23:20 の形式で送信しますが、Clarify は August 26, 1999 15:23:20 の形式で日付を送信する必要があります。

DtpDate クラスに渡される値は、次の規則に従う必要があります。

日	1 から 30 までの数値。日付と時刻のストリングに、月、年、日付の間の分離文字がなく、日付が数値形式の場合、単一の文字の前には、01 のようにゼロ (0) を付ける必要があります。
月	1 から 12 までの数値、January や February などの名前、または Jan や Feb などの省略形の (3 文字の) 月の名前。日付と時刻のストリングに、月、年、日付の間の分離文字がなく、日付が数値形式の場合、単一の文字の前には、01 のようにゼロ (0) を付ける必要があります。
年	4 桁の数値。
時間	01 から 23 までの値。24 時間形式を表します。AM または PM は指定できません。
分	01 から 59 までの数値。
秒	01 から 59 までの数値。

表 132 に、DtpDate クラスのメソッドについて要約します。この表では、静的メソッドとインスタンス・メソッドは分かれていますが、章全体ではアルファベット順になっています。

表 132. DtpDate メソッドの要約

メソッド	説明	ページ
コンストラクター		
DtpDate()	指定した形式に従って、日付を解析します。	435
静的メソッド		
getMaxDate()	ビジネス・オブジェクトのリストから、最も遅い日付を DtpDate オブジェクトとして戻します。	448
getMinDate()	ビジネス・オブジェクトのリストから、最も早い日付を DtpDate オブジェクトとして戻します。	451

表 132. DtpDate メソッドの要約 (続き)

メソッド	説明	ページ
getMaxDateBO()	ビジネス・オブジェクトのリストから、最も遅い日付を含むビジネス・オブジェクトを戻します。	449
getMinDateBO()	ビジネス・オブジェクトのリストから、最も早い日付を含むビジネス・オブジェクトを戻します。	452
インスタンス・メソッド		
addDays()	指定した日数をこの日付に加算します。	437
addWeekdays()	指定した平日の数をこの日付に加算します。	438
addYears()	指定した年数をこの日付に加算します。	439
after()	この日付が、入力パラメーターとして渡された日付に従っているかどうかを検査します。	439
before()	この日付が、パラメーターとして渡された日付より前かどうかを検査します。	440
calcDays()	この日付から別の日付までの日数を計算します。	441
calcWeekdays()	この日付から別の日付までの平日の数を計算します。	441
get12MonthNames()	この日付について、12 か月の現在の短縮名の表記を戻します。	442
get12ShortMonthNames()	この日付について、12 か月の現在の正式名の表記を戻します。	443
get7DayNames()	この日付について、曜日の現在の名前を戻します。	443
getCWDate()	この日付を、IBM 汎用の日付形式に再設定します。	443
getDayOfMonth()	この日付の日を戻します。	444
getDayOfWeek()	この日付の曜日を戻します。	444
getHours()	この日付の時間の値を戻します。	445
getIntDay()	この日付の曜日を整数として戻します。	445
getIntDayOfWeek()	この日付の曜日を戻します。	445
getIntMilliSeconds()	この日付からミリ秒の値を戻します。	446
getIntMinutes()	この日付の分の値を整数として戻します。	446
getIntMonth()	この日付の月を整数として戻します。	446
getIntSeconds()	この日付の秒を整数として戻します。	447
getIntYear()	この日付の年を整数として戻します。	447
getMSSince1970()	1970 年 1 月 1 日 00:00:00 からこの日付までのミリ秒数を戻します。	448
getMinutes()	この日付から分の値を戻します。	453
getMonth()	この日付の月の正式名表記を戻します。	454
getNumericMonth()	この日付から月の値を数値形式で戻します。	454
getSeconds()	この日付から秒の値をストリングとして戻します。	454
getShortMonth()	この日付から月名の短縮名表記を戻します。	455
getYear()	この日付の年の値を戻します。	455
set12MonthNames()	この日付の 12 か月の名前の正式名表記を変更します。	456
set12MonthNamesToDefault()	この日付の 12 か月の名前の正式名表記を、デフォルト値に復元します。	456
set12ShortMonthNames()	この日付の 12 か月の名前の短縮名表記を変更します。	457
set12ShortMonthNamesToDefault()	この日付の 12 か月の名前の短縮名表記を、デフォルト値に復元します。	457
set7DayNames()	この日付の曜日の名前を変更します。	458

表 132. DtpDate メソッドの要約 (続き)

メソッド	説明	ページ
set7DayNamesToDefault()	この日付について、曜日の名前をデフォルト値に復元します。	458
toString()	日付を指定した形式またはデフォルトの形式で戻します。	458

DtpDate()

指定した形式に従って、日付を解析します。

構文

```
public DtpDate()

public DtpDate(String dateTimeStr, String format)

public DtpDate(String dateTimeStr, String format, String[] monthNames,
   String[] shortMonthNames)

public DtpDate(long msSince1970, boolean isLocalTime)
```

パラメーター

dateTimeStr ストリングの形式の日付と時間。

format 日付の形式。詳しくは『注記』を参照してください。

monthNames 12 か月の正式名を表すストリングの配列。null の場合、デフォルト値は January、February、March などです。

shortMonthNames
月の短縮名を表すストリングの配列。このパラメーターが null で *monthNames* が null ではない場合、この値は、Jan、Feb、Mar など、月の正式名の最初の 3 文字になります。

msSince1970 1970 年 1 月 1 日 00:00:00 からのミリ秒数。

isLocalTime 時刻がすでに現地時間の場合は true、それ以外の場合は false に設定します。

戻り値

なし。

例外

DtpDateException - コンストラクターで解析エラーが発生した場合です。これは、日付が指定した形式ではない場合に発生することがあります。

注記

コンストラクターの第 1 の形式は、パラメーターを使用しません。この形式は、システムの現在の日付を新しい DtpDate オブジェクトに割り当てます。
DtpDateException はスローしません。

コンストラクターの第 2 の形式と第 3 の形式は、指定した日付*形式* に従って日付を解析し、日、月、年、時間、分、および秒の値を抽出します。これらの値は、他の DtpDate メソッドを使用して後で取得したり、再設定することができます。

例えば、月は以下の形式のいずれかで取得できます。

- 正式名表記 (デフォルト形式): January、February、March、April、May、June、July、August、September、 October、November、および December
- 数値形式: 1-12
- 各月の名前の最初の 3 文字で構成される短縮名表記:
  Jan、Feb、Mar、Apr、May、Jun、Jul、Aug、Sep、Oct、Nov、Dec

取得されたデータは、他のデータのコンテキストには依存しません。

月の正式名および短縮名の表記は、以下の方法で変更できます。

- それぞれ set12MonthNames() メソッドと set12ShortMonthNames() メソッドを使用する
- パラメーターとしての表記を、DtpDate() コンストラクターの 第 3 の形式に渡す

コンストラクターの第 4 の形式は、1970 年 1 月 1 日 00:00:00 からのミリ秒数を使用します。多くのアプリケーションが、この方法で日付を表します。

日付形式

日付*形式* では、日付は常に時刻より前にあります。時刻はオプションです。日付と時刻のストリングに時刻がない場合、時間、分、および秒の値は、デフォルト値の 00 になります。

日付形式は、以下のように、大文字小文字を区別するキー文字を使用します。

D	日
M	月
Y	年
h	時間
m	分
s	秒

これらのキー文字は、「/」や「-」などの分離文字で区切られることもあります。

例

次の例は、新しい日付オブジェクト aDate、date2、および date3 を作成する DtpDate() コンストラクターを示しています。

```
Dtpdate aDate = new DtpDate("5/21/1997 15:23:01", "M/D/Y h:m:s");
DtpDate date2 = new DtpDate("05211997 152301", "MDY hms");
DtpDate date3 = new DtpDate("Jan 10, 1999 10:00:00", "M D, Y h:m:s");
```

以下の日付形式では、DtpDateException がスローされます。

```
h:m:s D/M/Y
```

addDays()

指定した日数をこの日付に加算します。

構文

```
public DtpDate addDays(int numberOfDays)
```

パラメーター

numberOfDays 整数値。負の数値の場合、新しい日付は、DtpDate の現在のインスタンスの、*numberOfDays* 日前の日付になります。

戻り値

新しい DtpDate オブジェクト。

例外

```
DtpDateException
```

注記

addDays() メソッドは、指定した日数をこの日付に加算します。get() メソッドを使用すると、結果の新しい日付に関する情報を検索できます。戻される DtpDate オブジェクトは、月名、日付形式など、DtpDate の現在のオブジェクトのプロパティーをすべて継承します。

新しい日付は、有効な日付に合わせて調整されます。例えば、1999 年 1 月 29 日 1999 00:00:00 に 5 日を加算すると、1999 年 2 月 03 日 00:00:00 になり、-30 日を加算すると、1998 年 12 月 30 日 00:00:00 になります。

日を加算しても、時刻には影響しません。

例

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   System.out.println("Tomorrow is "
      + tomorrow.getDayOfMonth() + "/"
      + tomorrow.getNumericMonth() + "/"
      + tomorrow.getYear() + " "
      + tomorrow.getHours() + ":"
      + tomorrow.getMinutes() + ":"
      + tomorrow.getSeconds());
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

addWeekdays(), addYears()

addWeekdays()

指定した平日の数をこの日付に加算します。

構文

```
public DtpDate addWeekdays(int numberOfWeekdays)
```

パラメーター

numberOfWeekdays

整数値。負の数値の場合、新しい日付は、現在の DtpDate オブジェクトで表される日付の、*numberOfWeekdays* 日 (平日) 前の日付になります。

戻り値

新しい DtpDate オブジェクト。

例外

DtpDateException

注記

addWeekdays() メソッドは、指定した平日の数をこの日付に加算します。get メソッドを使用すると、結果の新しい日付に関する情報を検索できます。戻される DtpDate は、月名、日付形式など、DtpDate の現在のインスタンスのプロパティーをすべて継承します。

Monday、Tuesday、Wednesday、Thursday、および Friday、または同等の値のみが平日と見なされます。Monday は、最初の曜日と見なされます。

例

```
try
   {
   DtpDate toDay = new DtpDate("8/2/1999 00:00:00", "M/D/Y h:m:s");
   DtpDate fiveWeekdaysLater = toDay.addWeekdays(5);
   // The new date should be 8/9/1999 00:00:00
   System.out.println("Next month is "
      + fiveWeekdaysLater.getDayOfMonth() + "/"
      + fiveWeekdaysLater.getNumericMonth() + "/"
      + fiveWeekdaysLater.getYear() + " "
      + fiveWeekdaysLater.getHours() + ":"
      + fiveWeekdaysLater.getMinutes() + ":"
      + fiveWeekdaysLater.getSeconds());
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

addDays(), addYears()

addYears()

指定した年数をこの日付に加算します。

構文

```
public DtpDate addYears(int numberOfYears)
```

パラメーター

numberOfYears 整数値。負の数値の場合、新しい日付は、現在の DtpDate オブジェクトの、*numberOfYears* 年前の日付になります。

戻り値

新しい DtpDate オブジェクト。

注記

addYears() メソッドは、指定した年数をこの日付に加算します。get() メソッドを使用すると、結果の新しい日付に関する情報を検索できます。戻される DtpDate は、月名、日付形式など、DtpDate の現在のインスタンスのプロパティーをすべて継承します。

例

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getDayOfMonth() + "/"
   + lastYear.getNumericMonth() + "/"
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

参照項目

addDays(), addWeekdays()

after()

この日付が、入力パラメーターとして渡された日付に従っているかどうかを検査します。

構文

```
public boolean after(DtpDate date)
```

パラメーター

date この日付と比較される日付。

戻り値

この日付が渡された日付より後の場合は true を戻し、この日付が渡された日付より前の場合は false を戻します。

例外

DtpDateException

例

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = yesterday.addDays(-1);
   // isAfter should be false.
   boolean isAfter = yesterday.after(today)
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

before()

---

before()

この日付が、パラメーターとして渡された日付より前かどうかを検査します。

構文

public boolean before(DtpDate *date*)

パラメーター

date この日付と比較される日付。

戻り値

この日付が渡された日付より前の場合は true を戻し、この日付が渡された日付より後の場合は false を戻します。

例外

DtpDateException

例

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = yesterday.addDays(-1);
   // isBefore should be true.
   boolean isBefore = yesterday.before(today)
   }
```

```
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

after()

---

calcDays()

この日付から別の日付までの日数を計算します。

構文

public int calcDays(DtpDate *date*)

パラメーター

date この日付と比較される日付。

戻り値

日数を表す int。常に正数です。

例外

DtpDateException

注記

calcDays() メソッドは、この日付から別の日付までの日数の差を計算します。結果は常に自然数の日数です。

19990615 00:30:59 と 19990615 23:59:59 の差は 0 日、19990615 23:59:59 と 19990616 00:01:01 の差は 1 日です。

例

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   int days = today.caldDays(tomorrow);
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

calcWeekdays()

---

calcWeekdays()

この日付から別の日付までの平日の数を計算します。

構文

```
public int calcWeekdays(DtpDate date)
```

パラメーター

date この日付と比較される日付。

戻り値

平日の数を表す int。常に正数です。

例外

```
DtpDateException
```

注記

calcWeekdays() メソッドは、この日付から別の日付までの平日の数を計算します。金曜日と土曜日の差は 0、金曜日と月曜日の差は 1 です。平日の数は、月曜日から金曜日、または同等の値を想定しています。休日が平日の可能性もあるため、平日は営業日と同じとは限りません。

例

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   int days = today.caldWeekdays(tomorrow);
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

calcDays()

---

get12MonthNames()

この日付について、12 か月の現在の正式名の表記を戻します。

構文

```
public String[ ] get12MonthNames()
```

戻り値

12 か月の有効な名前を含む、ストリング・オブジェクトの配列。

例

```
DtpDate toDay = new DtpDate();
String[] toDay.get12MonthNames();
```

参照項目

set12MonthNames(), set12MonthNamesToDefault()

---

get12ShortMonthNames()

この日付について、12 か月の現在の短縮名の表記を戻します。

構文

```
public String[ ] get12ShortMonthNames()
```

戻り値

12 か月の有効な短縮名を含む、ストリング・オブジェクトの配列。

例

```
DtpDate toDay = new DtpDate();
String[] toDay.get12ShortMonthNames();
```

参照項目

set12ShortMonthNames(), set12ShortMonthNamesToDefault()

---

get7DayNames()

この日付について、曜日の現在の名前を戻します。

構文

```
public String[ ] get7DayNames()
```

戻り値

曜日の有効な名前を含む、ストリング・オブジェクトの配列。

例

```
DtpDate toDay = new DtpDate();
String[] toDay.get7DayNames();
```

参照項目

set7DayNames(), set7DayNamesToDefault()

---

getCWDate()

この日付を、IBM 汎用の日付形式に再設定します。

構文

```
public String getCWDate()
```

戻り値

日付を IBM WebSphere Business Integration Server Express 汎用ビジネス・オブジェクト形式で表したストリング。形式は YMD hms です。この形式の例は次のとおりです。

- 19990615 150701
- 19990831 114122

注記

IBM 汎用日付形式は、次の形式を使用します。

YYYYMMDD HHMMSS

例

```
DtpDate toDay = new DtpDate();
String genericDate = toDay.getCWDate();
```

---

getDayOfMonth()

この日付の日を戻します。

構文

```
public String getDayOfMonth()
```

戻り値

01、20、30 など、日を表すストリング。

例

```
DtpDate toDay = new DtpDate();
String dayOfMonth = toDay.getDayOfMonth();
```

参照項目

getIntDay()

---

getDayOfWeek()

この日付の曜日を戻します。

構文

```
public String getDayOfWeek()
```

戻り値

Monday、Tuesday など、曜日を示すストリング。

例

```
DtpDate toDay = new DtpDate();
String dayOfWeek = toDay.getDayOfWeek();
```

参照項目

getIntDayOfWeek()

getHours()

この日付の時間の値を戻します。

構文

```
public String getHours()
```

戻り値

00 から 23 までの時間の値を表すストリング。

例

```
DtpDate toDay = new DtpDate();
String hours = toDay.getHours();
```

getIntDay()

この日付の日を整数として戻します。

構文

```
public int getIntDay()
```

戻り値

日を表す int 値。

例

```
DtpDate toDay = new DtpDate();
int day = toDay.getIntDay();
```

参照項目

getDayOfMonth()

getIntDayOfWeek()

この日付の曜日を整数として戻します。

構文

```
public int getIntDayOfWeek()
```

戻り値

曜日を表す int 値。可能な値は、0 (Monday)、1 (Tuesday)、2 (Wednesday)、3 (Thursday)、4 (Friday)、5 (Saturday)、または 6 (Sunday) です。

例

```
DtpDate toDay = new DtpDate();
int dayOfWeek = toDay.getIntDayOfWeek();
```

参照項目

getDayOfWeek()

---

getIntMilliSeconds()

日付からミリ秒の値を戻します。

構文

```
public int getIntMilliSeconds()
```

戻り値

ミリ秒を表す int 値。範囲は 0 から 999 です。

例

```
DtpDate toDay = new DtpDate();
int millisecs = toDay.getIntMilliSeconds();
```

---

getIntMinutes()

この日付の分の値を整数として戻します。

構文

```
public int getIntMinutes()
```

戻り値

分を表す int 値。範囲は 0 から 59 です。

例

```
DtpDate toDay = new DtpDate();
int mins = toDay.getIntMinutes();
```

参照項目

getMinutes()

---

getIntMonth()

この日付の月を整数として戻します。

構文

```
public int getIntMonth()
```

戻り値

月を表す int 値。範囲は 1 (January) から 12 (December) です。

例

```
DtpDate toDay = new DtpDate();
int month = toDay.getIntMonth();
```

参照項目

getMonth(), getNumericMonth()

---

getIntSeconds()

この日付の秒を整数として戻します。

構文

```
public int getIntSeconds()
```

戻り値

秒を表す int 値。範囲は 0 から 59 です。

例

```
DtpDate toDay = new DtpDate();
int secs = toDay.getIntSeconds();
```

参照項目

getSeconds(), getMSSince1970()

---

getIntYear()

この日付の年を整数として戻します。

構文

```
public int getIntYear()
```

戻り値

年を表す int 値。

例

```
DtpDate toDay = new DtpDate();
int year = toDay.getIntYear();
```

参照項目

getYear()

getMSSince1970()

1970 年 1 月 1 日 00:00:00 からこの日付までのミリ秒数を戻します。

構文

```
public long getMSSince1970()
```

戻り値

整数値。この日付が 1970 年 1 月 1 日 00:00:00 より前の場合、負になる可能性もあります。

例外

```
DtpDateException
```

例

```
try
   {
   DtpDate toDay = new DtpDate();
   long ms = toDay.getMSSince1970();
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

参照項目

```
getSeconds()
```

getMaxDate()

ビジネス・オブジェクトのリストから、最も遅い日付を DtpDate オブジェクトとして戻します。

構文

```
public static DtpDate getMaxDate(BusObjArray boList, String attr,
      String dateFormat)
```

パラメーター

boList ビジネス・オブジェクトのリスト。

attr 比較を実行するときに使用するビジネス・オブジェクトの属性。指定できる属性のデータ型は、Date です。

dateFormat 日付の形式。詳しくは、DtpDate() を参照してください。これが null の場合、日付は 1970 年以降のミリ秒数であると想定されます。

戻り値

最大日付を含む DtpDate オブジェクト。

例外

DtpIncompatibleBOTypeException - リストのビジネス・オブジェクトが、同じビジネス・オブジェクト・タイプではない場合です。

DtpUnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

DtpUnsupportedAttributeTypeException - 指定した属性のデータ型が、上記のリストにあるサポート対象の属性データ型ではない場合です。

これらの例外はすべて、RunTimeEntityException のサブクラスです。

注記

getMaxDate() メソッドは、ビジネス・オブジェクトのリストをスキャンして、日付が最新のビジネス・オブジェクトを検索し、DtpDate オブジェクトの形式でその日付を戻します。

ヒント: このメソッドは静的メソッドです。

日付の評価では、Jan 1, 2004 000000 は Jan 1, 2002 000000 より後で、Jan 1, 2002 000000 は Jan 1, 1999 000000 より後です。

日付情報は、メソッドに渡される属性名に保管されることを前提とします。オブジェクトに null の日付情報がある場合、そのオブジェクトは無視されます。すべてのオブジェクトに null の日付情報がある場合は、null が戻されます。

例

```
try
   {
   DtpDate maxDate = DtpDate.getMaxDate(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

参照項目

getMinDate(), getMaxDateBO()

---

getMaxDateBO()

ビジネス・オブジェクトのリストから、最も遅い日付を含むビジネス・オブジェクトを戻します。

構文

```
public static BusObj[] getMaxDateBO(BusObj[] boList, String attr,
      String dateFormat)
public static BusObj[] getMaxDateBO(BusObjArray boList, String attr,
      String dateFormat)
```

パラメーター

boList ビジネス・オブジェクトのリスト。BusObj の配列、または BusObjArray のインスタンスです。これらのビジネス・オブジェクトは、ビジネス・オブジェクト・タイプが同じでなければなりません。

attr 比較されるビジネス・オブジェクトの属性。指定できる属性のデータ型は、Date です。

dateFormat 日付の形式。詳しくは、DtpDate() を参照してください。これが null の場合、日付は 1970 年以降のミリ秒数であると想定されます。

戻り値

日付が最新のビジネス・オブジェクトの配列。

例外

これらの例外はすべて、RunTimeEntityException のサブクラスです。

DtpIncompatibleBOTypeException - リストのビジネス・オブジェクトが、同じビジネス・オブジェクト・タイプではない場合です。

DtpUnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

DtpUnsupportedAttributeTypeException - 指定した属性のデータ型が、上記のリストにあるサポート対象の属性データ型ではない場合です。

DtpDateException - 日付形式が無効な場合です。

注記

getMaxDateBO() メソッドは、ビジネス・オブジェクトのリストをスキャンして、日付が最新のビジネス・オブジェクトを検索し、そのビジネス・オブジェクトを戻します。複数のビジネス・オブジェクトの最大日付が同じ場合、その日付のオブジェクトがすべて戻されます。

ヒント: このメソッドは静的メソッドです。

最も早い日付の評価では、Jan 1, 2004 000000 は Jan 1, 2002 000000 より後で、Jan 1, 2002 000000 は Jan 1, 1999 000000 より後です。

日付情報は、メソッドに渡される属性名に保管されることを前提とします。オブジェクトに null の日付情報がある場合、そのオブジェクトは無視されます。すべてのオブジェクトに null の日付情報がある場合は、null が戻されます。

例

```
try
   {
   BusObj[] max = DtpDate.getMaxDateBO(bos, "Start Date",
      "D/M/Y h:m:s");
```

```
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

参照項目

getMaxDate(), getMinDateBO()

---

getMinDate()

ビジネス・オブジェクトのリストから、最も早い日付を DtpDate オブジェクトとして戻します。

構文

```
public static DtpDate getMinDate(BusObjArray boList, String attr,
      String dateFormat)
```

パラメーター

boList	ビジネス・オブジェクトのリスト。
attr	比較を実行するときに使用するビジネス・オブジェクトの属性。指定できる属性のデータ型は、Date です。
dateFormat	日付の形式。詳しくは、DtpDate() を参照してください。これが null の場合、日付は 1970 年以降のミリ秒数であると想定されます。

戻り値

最も早い日付を含む DtpDate オブジェクト。

例外

DtpIncompatibleBOTypeException - リストのビジネス・オブジェクトが、同じビジネス・オブジェクト・タイプではない場合です。

DtpUnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

DtpUnsupportedAttributeTypeException - 指定した属性のデータ型が、上記のリストにあるサポート対象の属性データ型ではない場合です。

これらの例外はすべて、RunTimeEntityException のサブクラスです。

注記

getMinDate() メソッドは、ビジネス・オブジェクトのリストをスキャンして、日付が最も早いビジネス・オブジェクトを検索し、DtpDate オブジェクトの形式でその日付を戻します。

ヒント: このメソッドは静的メソッドです。

日付の評価では、Jan 1, 1999 000000 は Jan 1, 2002 000000 より前で、Jan 1, 2002 000000 は Jan 1, 2004 000000 より前です。

日付情報は、メソッドに渡される属性名に保管されることを前提とします。オブジェクトに null の日付情報がある場合、そのオブジェクトは無視されます。すべてのオブジェクトに null の日付情報がある場合は、null が戻されます。

例

```
try
   {
   DtpDate minDate = DtpDate.getMinDate(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

参照項目

getMaxDate(), getMinDateBO()

---

getMinDateBO()

ビジネス・オブジェクトのリストから、最も早い日付を含むビジネス・オブジェクトを戻します。

構文

```
public static BusObj[] getMinDateBO(BusObj[] boList, String attr,
      String dateFormat)
public static BusObj[] getMinDateBO(BusObjArray boList, String attr,
      String dateFormat)
```

パラメーター

boList ビジネス・オブジェクトのリスト。

attr 比較を実行するときに使用するビジネス・オブジェクトの属性。指定できる属性のデータ型は、Date です。

dateFormat 日付の形式。詳しくは、DtpDate() を参照してください。これが null の場合、日付は 1970 年以降のミリ秒数であると想定されます。

戻り値

その日付を持つビジネス・オブジェクトの配列。

例外

DtpIncompatibleBOTypeException - リストのビジネス・オブジェクトが、同じビジネス・オブジェクト・タイプではない場合です。

DtpUnknownAttributeException - 指定した属性が、渡されたビジネス・オブジェクトの有効な属性ではない場合です。

DtpUnsupportedAttributeTypeException - 指定した属性のデータ型が、上記のリストにあるサポート対象の属性データ型ではない場合です。

DtpDateException - 日付形式が無効な場合です。

これらの例外はすべて、RunTimeEntityException のサブクラスです。

注記

getMinDateBO() メソッドは、ビジネス・オブジェクトのリストをスキャンして、日付が最も早いビジネス・オブジェクトを検索し、DtpDate オブジェクトの形式でその日付を戻します。

ヒント: このメソッドは静的メソッドです。

最も早い日付の評価では、Jan 1, 2004 000000 は Jan 1, 2002 000000 より後で、Jan 1, 2002 000000 は Jan 1, 1999 000000 より後です。

日付情報は、メソッドに渡される属性名に保管されることを前提とします。オブジェクトに null の日付情報がある場合、そのオブジェクトは無視されます。すべてのオブジェクトに null の日付情報がある場合は、null が戻されます。

例

```
try
   {
   BusObj[] min = DtpDate.getMinDateBO(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

参照項目

getMinDate(), getMaxDateBO()

getMinutes()

この日付から分の値を戻します。

構文

```
public String getMinutes()
```

戻り値

分を表すストリング。戻り値は 00 から 59 です。

参照項目

getIntMinutes()

getMonth()

この日付の月の正式名表記を戻します。

構文

```
public String getMonth()
```

戻り値

January、February など、月の名前。

参照項目

getIntMonth(), getNumericMonth(), getShortMonth()

getNumericMonth()

この日付から月の値を数値形式で戻します。

構文

```
public String getNumericMonth()
```

戻り値

01、02 など、月を数値形式で表したストリング。

例

```
DtpDate toDay = new DtpDate();
System.out.println("Today is "
   + toDay.getDayOfMonth() + "/"
   + toDay.getNumericMonth() + "/"
   + toDay.getYear() + " "
   + toDay.getHours() + ":"
   + toDay.getMinutes() + ":"
   + toDay.getSeconds());
```

参照項目

getIntMonth(), getMonth()

getSeconds()

この日付から秒の値をストリングとして戻します。

構文

```
public String getSeconds()
```

戻り値

秒を表すストリング。戻り値は 00 から 59 です。

例

```
DtpDate toDay = new DtpDate();
System.out.println("Today is "
   + toDay.getDayOfMonth() + "/"
   + toDay.getNumericMonth() + "/"
   + toDay.getYear() + " "
   + toDay.getHours() + ":"
   + toDay.getMinutes() + ":"
   + toDay.getSeconds());
```

参照項目

getIntSeconds()

---

getShortMonth()

この日付から月名の短縮名表記を戻します。

構文

```
public String getShortMonth()
```

戻り値

Jan、Feb など、短形式で表された月の名前。

例

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getShortMonth() + " "
   + lastYear.getDayOfMonth() + ", "
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

参照項目

getMonth(), set12ShortMonthNames(), set12ShortMonthNamesToDefault()

---

getYear()

この日付の年の値を戻します。

構文

```
public String getYear()
```

戻り値

年を表すストリング。年の値には世紀が含まれます。例えば、1998 や 2004 です。

例

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getDayOfMonth() + "/"
   + lastYear.getNumericMonth() + "/"
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

参照項目

getIntYear()

---

set12MonthNames()

この日付の 12 か月の名前の正式名表記を変更します。

構文

```
public void set12MonthNames(String[] monthNames,
      boolean resetShortMonth)
```

パラメーター

monthNames 12 か月の名前を含むストリングの配列。最初のエレメントは 1 年の最初の月で、最後のエレメントは 1 年の最後の月です。

resetShortMonthNames
デフォルトでは、月の短縮名は、月の正式名の最初の 3 文字です。このフラグを true に設定すると、月の短縮名は、月の新しい正式名に基づいて変更されます。このフラグを false に設定すると、このメソッドは、月の短縮名を変更しません。

戻り値

なし。

例外

DtpDateException - 渡された月の名前が 12 個ではない場合です。

参照項目

get12MonthNames(), set12MonthNamesToDefault()

---

set12MonthNamesToDefault()

この日付の 12 か月の名前の正式名表記を、デフォルト値に復元します。

構文

```
public void set12MonthNamesToDefault()
```

戻り値

なし。

注記

デフォルト名は、January、February、March などです。

参照項目

`get12MonthNames()`, `set12MonthNames()`

---

set12ShortMonthNames()

この日付の 12 か月の名前の短縮名表記を変更します。

構文

```
public void set12ShortMonthNames(String[] shortMonths)
```

パラメーター

shortMonths　　ビジネス・オブジェクトのリスト。

戻り値

なし。

例外

`DtpDateException` - 渡された月の名前が 12 個ではない場合です。

参照項目

`get12ShortMonthNames()`, `set12ShortMonthNamesToDefault()`

---

set12ShortMonthNamesToDefault()

この日付の 12 か月の名前の短縮名表記を、デフォルト値に復元します。

構文

```
public void set12ShortMonthNamesToDefault()
```

戻り値

なし。

注記

月の短縮名は、Jan、Feb、Mar などです。

参照項目

`get12ShortMonthNames()`, `set12ShortMonthNames()`

set7DayNames()

この日付の曜日の名前を変更します。

構文

```
public void set7DayNames(String[] dayNames)
```

パラメーター

dayNames 曜日を含むストリングの配列。最初のエレメントは、Monday と同等でなければなりません。

戻り値

なし。

例外

DtpDateException - 7 個の曜日が指定されていない場合です。

参照項目

get7DayNames(), set7DayNamesToDefault()

set7DayNamesToDefault()

この日付について、曜日の名前をデフォルト値に復元します。

構文

```
public void set7DayNamesToDefault()
```

戻り値

なし。

注記

デフォルト名は、Monday、Tuesday、Wednesday などです。

参照項目

get7DayNames(), set7DayNames()

toString()

日付を指定した形式またはデフォルトの形式で戻します。

構文

```
public String toString()
public String toString(String format)
public String toString(String format boolean twelveHr)
```

パラメーター

format 日付の形式。詳しくは、DtpDate() を参照してください。

twelveHr boolean 値。true に設定した場合は、メソッドが 24 時間ではなく 12 時間で表した時刻を戻すことを指定します。

戻り値

次のような、日付情報を含むストリングです。

```
19990930 053029 PM
```

月の位置の形式に関係なく、出力ストリングは常に 2 文字の整数表現になります (January は 01、December は 12 など)。

例外

DtpDateException - 日付形式が無効な場合です。

例

```
try
   {
   DtpDate toDay = new DtpDate();
   String date = toDay.toString("Y/M/D h:m:s");
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

第 17 章 DtpMapService クラス

サブマップは、別のマップ内から呼び出すマップです。DtpMapService クラスは、サブマップを実行するためのメソッドを提供します。表 133 に、DtpMapService クラスのメソッドについて要約します。

表 133. DtpMapService メソッドの要約

メソッド	説明	ページ
runMap()	指定したマップを実行します。	461

runMap()

指定したマップを実行します。

構文

```
BusObj[] runMap(String mapName, String mapType,
                BusObj[] srcBOs, cwExecCtx)
```

パラメーター

mapName 実行するマップの名前。

mapType 実行するマップのタイプ。DtpMapService クラスに定義されている定数 CWMAPTYPE (IBM WebSphere Business Integration Server Express マップ) のみ を使用します。

srcBOs *mapName* のソース・ビジネス・オブジェクトであるビジネス・オブジェクトの配列。

cwExecCtx 現行マップの実行コンテキストを含む変数。この変数は、Map Designer Express がすべてのマップに対して生成するコードで定義されます。

戻り値

mapName の宛先ビジネス・オブジェクトであるビジネス・オブジェクトの配列を戻します。

例外

MapFailureException - *mapName* の実行中にエラーが発生した場合です。

MapNotFoundException - *mapName* がリポジトリーの中にない場合です。

CxMissingIDException - maintainSimpleIdentityRelationship() を参照してください。

注記

別のマップ内からサブマップを呼び出すには、runMap() メソッドを使用します。サブマップの呼び出しの詳細については、48 ページの『サブマップを使用した変換』を参照してください。

例

次のコードはサブマップを呼び出して、アプリケーション固有の Address ビジネス・オブジェクトを汎用 Address ビジネス・オブジェクトにマップします。

```
// Create the BusObj Array
BusObj[] rSrcBOs = new BusObj[1];
rSrcBOs[0] = MyCustomerObj.MyAddressObj[0];

// Make the call to the map service
OutObjName = DtpMapService.runMap(MyAppAddressToGenAddress,
   DtpMapService.CWMAPTYPE,rSrcBOs,cwExecCtx);
```

参照項目

48 ページの『サブマップを使用した変換』

第 18 章 DtpSplitString クラス

DtpSplitString クラスは、ストリングをトークンに分割または解析して、結果を取得する方法を提供します。このクラスは、複合キー、日付、電話番号などのフォーマット設定されたストリングを操作する場合に有効です。

DtpSplitString は、java.util パッケージの StringTokenizer クラスに似ています。ただし、IBM WebSphere Business Integration Server Express マップを処理する場合、DtpSplitString は、StringTokenizer に比べて次の利点があります。

- DtpSplitString オブジェクト内のトークンはインデックスが付けられます。これにより、関心のある特定のトークンを簡単に抽出できます。例えば、電話番号 (650-555-1111 など) を、区切り文字としてダッシュ (-) を使用して 3 つのトークンに解析する場合、エレメント 0 を参照して市外局番を抽出し、エレメント 1 と 2 を連結して残りの電話番号を作成できます。
- DtpSplitString オブジェクトにより、トークンを双方向にスクロールできます。nextElement() と prevElement() を使用してエレメントをナビゲートする場合、すべてのエレメントが使用可能な状態のままです。

表 134 に、DtpSplitString クラスのメソッドについて要約します。

表 134. DtpSplitString メソッドの要約

メソッド	説明	ページ
DtpSplitString()	DtpSplitString の新しいインスタンスを構成して、ストリングをトークンに解析します。	463
elementAt()	指定した位置の DtpSplitString オブジェクトのエレメントを戻します。	464
firstElement()	位置ゼロの DtpSplitString オブジェクトのエレメントを戻します。	465
getElementCount()	エレメントの総数を含む整数を戻します。	465
getEnumeration()	各ストリングが、解析されたトークンの 1 つであるストリング・オブジェクトの列挙を戻します。	466
lastElement()	DtpSplitString オブジェクトの最後のエレメントを戻します。	466
nextElement()	DtpSplitString オブジェクトの次のエレメントを戻します。	467
prevElement()	DtpSplitString オブジェクトの前のエレメントを戻します。	468
reset()	DtpSplitString オブジェクトの現在の位置番号をゼロにリセットします。	468

DtpSplitString()

DtpSplitString の新しいインスタンスを構成して、ストリングをトークンに解析します。

構文

```
DtpSplitString(String str, String delimiters)
```

パラメーター

str 解析するストリング。

delimiters *str* で使用される区切り文字を含むストリング。複数の区切り文字を使用できますが、各区切り文字の長さは 1 文字でなければなりません。

注記

DtpSplitString() は、指定した区切り文字に基づいて、*str* をエレメントと呼ばれるトークンに解析します。DtpSplitString() を呼び出した後、DtpSplitString クラス・メソッドを呼び出して、特定のエレメントを選択および取得できます。

例

```
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");
```

---

elementAt()

指定した位置の DtpSplitString オブジェクトのエレメントを戻します。

構文

```
String elementAt(int nth)
```

パラメーター

nth DtpSplitString オブジェクトから抽出されるエレメントの位置。最初のエレメントの位置はゼロです。

戻り値

位置 *nth* にあるエレメントを含む String を戻します。

例外

DtpNoElementAtPositionException - *nth* に無効な位置を指定した場合です。

注記

エレメントには、ゼロから始まって、最初から最後まで番号が付けられます。例えば、区切り文字がコンマとスペースの場合、ストリング 「This,is a test」の位置 2 のエレメントは「a」になります。

elementAt() メソッドは、指定した位置のエレメントを戻しますが、現在のエレメントの位置は変更しません。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

//This call returns "a"
public String MyString.elementAt(2);
```

参照項目

getElementCount()

---

firstElement()

位置ゼロの DtpSplitString オブジェクトのエレメントを戻します。

構文

```
String firstElement()
```

戻り値

位置ゼロのエレメントを含む String を戻します。

例外

DtpNoElementAtPositionException - エレメントがない場合です。

注記

DtpSplitString オブジェクトのエレメントには、ゼロから始まって、最初から最後まで番号が付けられます。したがって、最初のエレメントの位置はゼロです。

firstElement() メソッドは、位置ゼロのエレメントを戻しますが、現在のエレメントの位置は変更しません。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the first element containing "This"
String anElement = MyString.firstElement();
```

参照項目

lastElement()

---

getElementCount()

DtpSplitString オブジェクトのエレメントの総数を戻します。

構文

```
int getElementCount()
```

戻り値

エレメントの総数を含む整数を戻します。

注記

エレメントには、ゼロから始まって、最初から最後まで番号が付けられます。 getElementCount() が 6 を戻した場合、最も大きい番号のエレメントは 5 です。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the integer 4
String numElements = MyString.getElementCount();
```

参照項目

firstElement(), lastElement()

---

getEnumeration()

各ストリングが、解析されたトークンの 1 つであるストリング・オブジェクトの列挙を戻します。

構文

```
Enumeration getEnumeration()
```

戻り値

列挙型オブジェクトを戻します。

注記

getEnumeration() メソッドは、DtpSplitString オブジェクトの解析されたトークンを処理するもう 1 つの方法を提供します。列挙型オブジェクトを処理する方法の詳細については、『Java.Util パッケージ』を参照してください。

---

lastElement()

DtpSplitString オブジェクトの最後のエレメントを戻します。

構文

```
String lastElement()
```

戻り値

最後のエレメントを含む String を戻します。

例外

DtpNoElementAtPositionException - エレメントがない場合です。

注記

エレメントには、ゼロから始まって、最初から最後まで番号が付けられます。最後のエレメントは、最も番号の大きいエレメントです。最後のエレメントの位置番号は、getElementCount()-1 と同じです。

lastElement() メソッドは、最後のエレメントを戻しますが、現在のエレメントの位置は変更しません。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the last element, containing "test"
String anElement = MyString.lastElement();
```

参照項目

firstElement(), getElementCount()

---

nextElement()

DtpSplitString オブジェクトの次のエレメントを戻します。

構文

```
String nextElement()
```

戻り値

次のエレメントを含む String を戻します。

例外

DtpNoElementAtPositionException - 次のエレメントがない場合です。

注記

初めて nextElement() を呼び出したときに、位置ゼロのエレメントが戻されます。後続のメソッド呼び出しでは、nextElement() は位置 1、2、3 などのエレメントを戻します。nextElement() と prevElement() を使用して、DtpSplitString オブジェクト内のエレメント (トークン) をナビゲートすることもできます。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()
```

参照項目

prevElement(), reset()

---

prevElement()

DtpSplitString オブジェクトの前のエレメントを戻します。

構文

```
String prevElement()
```

戻り値

前のエレメントを含む String を戻します。

例外

DtpNoElementAtPositionException - 前のエレメントがない場合です。

注記

prevElement() と nextElement() を使用して、DtpSplitString オブジェクト内のエレメント (トークン) をナビゲートすることもできます。初めて nextElement() を呼び出したとき、エレメントの位置はゼロです。以降の nextElement() の呼び出しにより、位置が 1 つずつ増分されます。prevElement() メソッドは、前のエレメントを戻し、エレメントの位置を 1 つずつ減少させます。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()

// This call returns element 0 containing "This"
String anotherElement = MyString.prevElement()
```

参照項目

nextElement()

---

reset()

DtpSplitString オブジェクトの現在の位置番号をゼロにリセットします。

構文

```
void reset()
```

戻り値

なし。

注記

デフォルトのエレメント位置はゼロです。nextElement() を呼び出すたびに、エレメントの位置が 1 つずつ増分されます。prevElement() メソッドは、前のエレメントを戻し、エレメントの位置を 1 つずつ減少させます。reset() を使用すると、現在の位置をゼロにリセットできます。

例

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()

// Reset the position to zero
MyString.reset()

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()
```

参照項目

nextElement(), prevElement()

第 19 章 DtpUtils クラス

DtpUtils クラスは、複数の汎用的な操作を実行します。

表 135 に、DtpUtils クラスのメソッドを要約します。

表 135. DtpUtils メソッドの要約

メソッド	説明	ページ
padLeft()	ストリングを指定した文字で埋めます。	471
padRight()	ストリングを指定した文字で埋めます。	471
stringReplace()	ストリング内のパターンの出現を、別のパターンに置換します。	471
truncate()	この数値を切り捨てます。	473

padLeft()

ストリングを指定した文字で埋めます。

構文

```
public static String padLeft(String src, char padWith, int totalLen)
```

パラメーター

src 埋め込まれるストリング。

padWith 埋め込みに使用される文字。

totalLen ストリングの新しいサイズ (正数)。値が 0、元のストリングのサイズより小さい、または負数の場合、元のストリングが戻されます。

戻り値

新しく埋め込まれたストリング。

注記

ストリングを指定した文字で埋めます。

例

次の呼び出しは、0000012345 を戻します。

```
padLeft("12345", '0', 10);
```

次の呼び出しは、123456 を戻します。

```
padLeft("123456", '0', 5);
```

padRight()

ストリングを指定した文字で埋めます。

構文

```
public static String padLeft(String src, char padWith, int totalLen)
```

パラメーター

src	埋め込まれるストリング。
padWith	埋め込みに使用される文字。
totalLen	ストリングの新しいサイズ (正数)。値が 0、元のストリングのサイズより小さい、または負数の場合、元のストリングが戻されます。

戻り値

新しく埋め込まれたストリング。

注記

ストリングを指定した文字で埋めます。

例

次の呼び出しは、1234500000 を戻します。

```
padRight("12345", '0', 10);
```

次の呼び出しは、123456 を戻します。

```
padRight("123456", '0', 5);
```

---

stringReplace()

ストリング内のパターンの出現を、別のパターンに置換します。

構文

```
public static String stringReplace(String src, String oldpattern,
      String newPattern)
```

パラメーター

src	変更されるストリング。
oldPattern	埋め込みに使用される文字。
newPattern	置換に使用されるストリング・パターン。

戻り値

新しいパターンを含む新しいストリング。

注記

メソッドは、*oldPattern* で指定された値をすべて、*newPattern* で指定された値に置換します。単一文字の置換の場合は、Java String クラスで replace() を使用します。*oldPattern* が見つからない場合は、元の (変更前の) ストリングが戻されます。

例

次のストリングは、youoyou and dad になります。

```
stringReplace("momomom and dad", "mom", "you");
```

truncate()

この数値を切り捨てます。

構文

```
public static double truncate(Object aNumber, int precision)
   throws DtpIncompatibleFormatException

public static double truncate(float aNumber, int precision)
public static double truncate(double aNumber, int precision)

public static int truncate(Object aNumber)
   throws DtpIncompatibleFormatException

public static int truncate(float aNumber)
public static int truncate(double aNumber)
```

パラメーター

aNumber 番号。有効な型は、String、float、および double です。

precision 小数点の右側で除去される桁数。

戻り値

double または int の数値。

注記

このメソッドは、右から始めて、この数値から桁を除去します。

メソッドの最初の 3 つの形式は、右から始めて、小数点の右側の桁を除去することで、数値を切り捨てます。入力した数値が整数の場合は、切り捨てられません。Object の型の数値は、String、Double、または Float でなければなりません。

メソッドの最後の 3 つの形式は、小数点の右側の桁をすべて除去することで、数値を切り捨て、int 値を戻します。

例

次の呼び出しは、123.45 を戻します。

```
truncate("123.4567", 2);
```

次の呼び出しは、123 を戻します。

```
truncate(123.456, 4)
```

第 20 章 IdentityRelationship クラス

この章では、IdentityRelationship クラスのオブジェクトに対して操作を行うメソッドについて説明します。これらのオブジェクトは、一致関係のインスタンスを表します。 IdentityRelationship クラスは、リポジトリー・データベースにアクセスするときに必要な追加機能を提供します。既存の API のセットを、マップ開発者が簡単に使用できるメソッドに結合します。

IdentityRelationship クラスのメソッドのソース・コードが提供されますが、このソース・コードは、IBM WebSphere Business Integration Server Express 環境で現状のまま使用することも、あるいは他の環境に合わせてカスタマイズすることもできます。

IdentityRelationship クラスのメソッドを表 136 に示します。

表 136. IdentityRelationship メソッドの要約

メソッド	説明	ページ
addMyChildren()	指定した子インスタンスを、一致関係の親/子関係に追加します。	475
deleteMyChildren()	指定した親に属する一致関係の親/子関係に対して、指定した子インスタンスを除去します。	477
foreignKeyLookup()	ソース・ビジネス・オブジェクトの外部キーに基づいて、外部関係表でルックアップを実行します。外部関係表に外部キーが存在しない場合、関係インスタンスの検索は失敗します。	478
foreignKeyXref()	ソース・ビジネス・オブジェクトの外部キーに基づいて、リレーションシップ・データベースの関係表でルックアップを実行します。外部キーが存在しない場合は、外部関係表で新しい関係インスタンスを追加します。	480
maintainChildVerb()	マップの実行コンテキストと、親ビジネス・オブジェクトの動詞に基づいて、子ビジネス・オブジェクトの動詞を設定します。	483
maintainCompositeRelationship()	親マップ内から複合一致関係を保守します。	485
maintainSimpleIdentityRelationship()	親マップまたは子マップ内から単純一致関係を保守します。	487
updateMyChildren()	必要に応じて、一致関係の指定された親/子関係で子インスタンスを追加または削除します。	489

注: IdentityRelationship クラスのメソッドはすべて、静的として宣言されます。既存の関係インスタンスから、または IdentityRelationship クラス IdentityRelationship.*method* を参照することで、このクラスの任意のメソッドを呼び出すことができます。*method* は、表 136 のメソッドの名前です。

addMyChildren()

指定した子インスタンスを、一致関係の親/子関係に追加します。

構文

```
public static void addMyChildren(String parentChildRelDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, Object childBusObjList,
      CxExecutionContext map_ctx)
```

パラメーター

parentChildRelDefName
親/子関係定義の名前。

parentParticpntDefName
親/子関係で親ビジネス・オブジェクトを表す参加者定義の名前。

parentBusObj 親ビジネス・オブジェクトを含む変数。

childParticpntDefName
親/子関係で子ビジネス・オブジェクトを表す参加者定義の名前。

childBusObjList
子ビジネス・オブジェクト、または関係に追加されるオブジェクトを含む変数。このパラメーターは、単一の汎用ビジネス・オブジェクト (BusObj)、または汎用ビジネス・オブジェクトの配列 (BusObjArray) にすることができます。

map_ctx マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException

注記

addMyChildren() メソッドは、*childBusObjList* の子インスタンスを、*parentChildRelDefName* 関係定義の関係表に追加します。このメソッドは、固有キーを持つ親ビジネス・オブジェクトを含むカスタム関係で有効です。親ビジネス・オブジェクトに子ビジネス・オブジェクトの追加がある場合は、addMyChildren() を使用して、変更後イメージ (*parentBusObj* にあります) と変更前イメージ (関係表の情報) を比較し、変更後イメージで新しい子オブジェクトを判別します。新しい子オブジェクトごとに、addMyChildren() は、親および子の参加者の関係表 (*parentParticpntDefName* と *childParticpntDefName*) に子インスタンスを追加します。関係表に親ビジネス・オブジェクトが存在しない場合、addMyChildren() は、この親オブジェクトの関係インスタンスを挿入します。

addMyChildren() メソッドでは、Relationship Designer Express で定義された親/子関係が必要です。この種類の関係を作成する方法の詳細については、309 ページの『親/子関係定義の作成』を参照してください。

参照項目

deleteMyChildren(), updateMyChildren()

309 ページの『子インスタンスの管理』

deleteMyChildren()

指定した親に属する一致関係の親/子関係に対して、指定した子インスタンスを除去します。

構文

```
void deleteMyChildren(String parentChildRelDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, Object childBusObjList,
      CxExecutionContext map_ctx)

void deleteMyChildren(String parentChildRefDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, CxExecutionContext map_ctx)
```

パラメーター

parentChildRelDefName
: 親/子関係定義の名前。

parentParticpntDefName
: 親/子関係で親ビジネス・オブジェクトを表す参加者定義の名前。

parentBusObj
: 親ビジネス・オブジェクトを含む変数。

childParticpntDefName
: 親/子関係で子ビジネス・オブジェクトを表す参加者定義の名前。

childBusObjList
: 関係から削除される子ビジネス・オブジェクトを含む変数。このパラメーターは、単一の汎用ビジネス・オブジェクト (BusObj)、または汎用ビジネス・オブジェクトの配列 (BusObjArray) にすることができます。

map_ctx
: マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException

注記

deleteMyChildren() メソッドは、親/子の *parentChildRelDefName* 関係定義から子インスタンスを削除します。このメソッドは、以下の形式をサポートしています。

- メソッドの最初の形式は、親および子の参加者の関係表から、*childBusObjList* の各子ビジネス・オブジェクトに対応する子インスタンスを除去します。子インスタンスを検索し、子オブジェクトの値と名前、および親オブジェクトの値と名前を突き合わせて削除します。
- メソッドの第 2 の形式は、親および子の参加者の関係表から、*parentBusObj* 親オブジェクトの子インスタンスをすべて除去します。子インスタンスを検索し、親オブジェクトの値と名前を突き合わせて削除します。

このメソッドは、固有キーを持つ親ビジネス・オブジェクトを含むカスタム関係で有効です。親オブジェクトが子オブジェクトを除去した場合は、deleteMyChildren() を使用して、変更後イメージ (*parentBusObj* にあります) と変更前イメージ (関係表の情報) を比較し、変更後イメージで除去された子オブジェクトを判別します。子オブジェクトごとに、deleteMyChildren() は、親および子の参加者の関係表 (*parentParticpntDefName* と *childParticpntDefName*) の関係表から対応する子インスタンスを除去します。

deleteMyChildren() メソッドでは、Relationship Designer Express で定義された親/子関係が必要です。この種類の関係を作成する方法の詳細については、309 ページの『親/子関係定義の作成』を参照してください。

参照項目

addMyChildren(), updateMyChildren()

309 ページの『子インスタンスの管理』

---

foreignKeyLookup()

ソース・ビジネス・オブジェクトの外部キーに基づいて、外部関係表でルックアップを実行します。外部関係表に外部キーが存在しない場合、関係インスタンスの検索は失敗します。

構文

```
public static void foreignKeyLookup(String relDefName,
      String appParticpntDefName, BusObj
 appSpecificBusObj,      String appForeignAttr,
       BusObj genericBusObj,
String genForeignAttr, CwExecutionContext map_ctx)
```

パラメーター

relDefName　外部ビジネス・オブジェクトを管理する単純一致関係の名前。

appParticpntDefName
単純一致関係でアプリケーション固有のビジネス・オブジェクトを表す参加者定義の名前。この参加者のタイプは、外部のアプリケーション固有のビジネス・オブジェクトです。

appSpecificBusObj
アプリケーション固有のビジネス・オブジェクトを含む変数。アプリケーション固有のビジネス・オブジェクトには、外部ビジネス・オブジェクトへの参照が含まれます。

appForeignAttr
外部ビジネス・オブジェクトのキー値を含むアプリケーション固有のビジネス・オブジェクトの属性の名前。

genericBusObj *appSpecificObject* のマップ先、またはマップ元の汎用ビジネス・オブジェクトを含む変数。

genForeignAttr
外部ビジネス・オブジェクトへの汎用的な参照を含む、汎用ビジネス・オブジェクトの属性名の名前。

map_ctx マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException

注記

foreignKeyLookup() メソッドは、*AppParticpntDefName* 参加者の関係表で、外部キー・ルックアップを実行します。つまり、*appSpecificBusObj* ビジネス・オブジェクトの外部キーで値が一致する関係インスタンスの外部関係表を検査します。このルックアップが失敗した場合、foreignKeyLookup() メソッドは、宛先ビジネス・オブジェクトの外部キーを null に設定します。外部関係表への行の挿入は行われません (foreignKeyXref() メソッドと同様です)。このメソッドは、インバウンド・マップとアウトバウンド・マップの両方で使用できます。

例

Requisition オブジェクトに対する Clarify_PartRequest では、VendorId フィールドは外部キー・ルックアップです。これは、Purchasing で Vendor Wrapper が呼び出されないためです。ルックアップが失敗した場合に行を挿入したくないので、ここでは foreignKeyXref() メソッドを使用していません。

```
if (ObjCustomerRole.isNull("RoleId"))
   {
   logError(5003, "OrderAssociatedCustomers.RoleId");
   // throw new MapFailureException("OrderAssociatedCustomers.RoleId
   // is null");
   }

try
   {
   IdentityRelationship.foreignKeyLookup("Customer", "SAPCust",
         ObjSAP_OrderPartners, "PartnerId", ObjCustomerRole,
         "RoleId", cwExecCtx);
   }

catch (RelationshipRuntimeException re)
   {
   logWarning(re.getMessage());
   }
if (ObjSAP_OrderPartners.get("PartnerId") == null)
```

```
{
logError(5007, "SAP_OrderPartners.PartnerId",
   "OrderAssociatedCustomers.RoleId", "Customer", "SAPCust",
   strInitiator);
throw new MapFailureException("ForeignKeyLookup failed");
}
```

参照項目

foreignKeyXref()

318 ページの『外部キー参照の実行』

---

foreignKeyXref()

ソース・ビジネス・オブジェクトの外部キーに基づいて、リレーションシップ・データベースの関係表でルックアップを実行します。外部キーが存在しない場合は、外部関係表で新しい関係インスタンスを追加します。

構文

```
public static void foreignKeyXref(String relDefName,
      String appParticpntDefName, String genParticpntDefName,
      BusObj appSpecificBusObj,String appForeignAttr,
      BusObj genericBusObj, String genForeignAttr,
      CxExecutionContext map_ctx)
```

パラメーター

relDefName　外部ビジネス・オブジェクトを管理する単純一致関係名の名前。

appParticpntDefName
単純一致関係でアプリケーション固有のビジネス・オブジェクトを表す参加者定義の名前。この参加者のタイプは、外部のアプリケーション固有のビジネス・オブジェクトです。

genParticpntDefName
単純一致関係で汎用ビジネス・オブジェクトを表す参加者定義の名前。この参加者のタイプは、外部の汎用ビジネス・オブジェクトです。

appSpecificBusbj
外部オブジェクトへの参照を含む、アプリケーション固有のビジネス・オブジェクト。

appForeignAttr
外部ビジネス・オブジェクトのキー値を含むアプリケーション固有のビジネス・オブジェクトの属性の名前。

genericObject　*appSpecificObject* のマップ先、またはマップ元の汎用ビジネス・オブジェクト。

genForeignAttr
外部ビジネス・オブジェクトへの汎用的な参照を含む、汎用ビジネス・オブジェクトの属性名の名前。

map_ctx　マップの実行コンテキスト。マップの実行コンテキストを渡すに

は、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException

注記

foreignKeyXref() メソッドは、*AppParticpntDefName* 参加者の関係表で、外部キー・ルックアップを実行します。つまり、*appSpecificBusObj* ビジネス・オブジェクトの外部キーで値が一致する関係インスタンスの外部関係表を検査します。このルックアップが失敗した場合、foreignKeyXref() メソッドは、アプリケーション固有キーの新しい関係インスタンスを外部関係表に追加します。宛先ビジネス・オブジェクトの外部キーを null に設定することはありません (foreignKeyLookup() メソッドと同様です)。このメソッドは、インバウンド・マップとアウトバウンド・マップの両方で使用できます。

foreignKeyXref() メソッドは、渡された引き数で以下の検証を実行します。

- *relDefName* 関係定義の名前を検証します。
- アプリケーション固有のビジネス・オブジェクトの *particpntDefName* 参加者定義の名前を検証します。
- *relDefName* 関係が一致関係であることを確認します。また、汎用ビジネス・オブジェクトを表す *relDefName* の参加者定義は、IBM WebSphere Business Integration Server Express **管理対象**として定義される必要があります。これらの設定を指定する方法の詳細については、267 ページの『一致関係の定義』を参照してください。

上記の検証のいずれかが失敗した場合、foreignKeyXref() は RelationshipRuntimeException 例外をスローします。

引き数の検証が完了すると、foreignKeyXref() が実行するアクションは、次の情報によって異なります。

- 呼び出しコンテキスト — マップの実行コンテキストでは、*map_ctx* 引き数 (cwExecCtx) の一部として渡されます
- 動詞 — ソース・ビジネス・オブジェクトの場合
  - 呼び出しコンテキスト EVENT_DELIVERY (または ACCESS_REQUEST) と SERVICE_CALL_RESPONSE のアプリケーション固有のビジネス・オブジェクト (*appSpecificBusObj*)
  - 呼び出しコンテキスト SERVICE_CALL_REQUEST と ACCESS_RESPONSE の汎用ビジネス・オブジェクト (*genericBusObj*)

foreignKeyXref() メソッドは、呼び出しコンテキストと動詞の適切な組み合わせに対して、外部関係表での関係インスタンスの基本的な追加をすべて処理します。foreignKeyXref() が行うアクションの詳細については、319 ページの『Foreign Key

Cross-Reference 関数ブロックの使用』を参照してください。表 110 と表 111 に、各呼び出しコンテキストのアクションを示します。

例

Order マップに対する Clarify_SFAQuote では、CustomerId フィールドは外部キー・ルックアップです。これは、Sales Order Processing Collab が Customer Wrapper を呼び出すためです。

```
if (ObjSAP_OrderLineItem.get("SAP_OrderLineObjectIdentifier[0]")
      != null)
   {
   if (ObjSAP_OrderLineItem.getString(
"SAP_OrderLineObjectIdentifier[0].ObjectQualifier").equals("002"))
      {
      BusObj temp = ObjSAP_OrderLineItem.getBusObj(
         "SAP_OrderLineObjectIdentifier[0]");
      if (temp.isNull("ItemId"))
         {
         logWarning(5003,
   "SAP_OrderLineItem.SAP_OrderLineObjectIdentifier[1].ItemId");
         }
      else
         {
         try
            {
            IdentityRelationship.foreignKeyXref(
               "Item",
               "SAPMbasc",
               "CWItba",
               temp,
               "ItemId",
               ObjOrderLineItem,
               "ItemId",
               cwExecCtx);
            }
         catch (RelationshipRuntimeException re)
            {
            logWarning(re.getMessage());
            }

         if (ObjOrderLineItem.get("ItemId") == null)
            {
            logError(5009, "OrderLineItem.ItemId",
         "SAP_OrderLineItem.SAP_OrderLineObjectIdentifier.ItemId",
               "Item",
               "SAPMbasc",
               strInitiator);

            throw new MapFailureException("ForeignKeyXref() failed");
            }
         }
      }
   }
```

参照項目

foreignKeyLookup()

318 ページの『外部キー参照の実行』

maintainChildVerb()

マップの実行コンテキストと、親ビジネス・オブジェクトの動詞に基づいて、子ビジネス・オブジェクトの動詞を設定します。

構文

```
public static void maintainChildVerb (String relDefName,
      String appSpecificParticpntName,
String genericParticpntName,
      BusObj appSpecificObj,
String appSpecificChildObj,
      BusObj genericObj,
String genericChildObj,
      CxExecutionContext map_ctx,
boolean to_Retrieve,
boolean is_Composite)
```

パラメーター

relDefName 子ビジネス・オブジェクトを管理する一致関係名の名前。

appSpecificParticpntName
アプリケーション固有の参加者定義の名前。

genericParticpntName
汎用的な参加者定義の名前。

appSpecificObj
子オブジェクトを含む、アプリケーション固有のオブジェクト。

appSpecificChildObj
アプリケーションの子ビジネス・オブジェクトの名前。

genericObj *appSpecificObject* のマップ先、またはマップ元の汎用ビジネス・オブジェクト。

genericChildObj
汎用子ビジネス・オブジェクトの名前。

ctx 実行コンテキスト。

to_Retrieve SERVICE_CALL_RESPONSE ロジックのフラグ。条件が true の場合は、子ビジネス・オブジェクトの動詞を更新します。false の場合は何もしません。

isComposite 子の参加者が複合キーを使用するかどうかを示すフラグ。条件が true の場合、キーが使用されます。false 場合、キーは使用されません。

戻り値

なし。

例外

RelationshipRuntimeException — この例外がスローされた場合の詳細については、『注記』セクションを参照してください。

ClassCastException

注記

maintainChildVerb() メソッドは、渡された引き数で以下の検証を実行します。

- *relDefName* 関係定義の名前を検証します。
- アプリケーション固有のビジネス・オブジェクト (*appSpecificParticpntName*) と汎用ビジネス・オブジェクト (*genericParticpntName*) の参加者定義の名前を検証します。
- アプリケーション固有のビジネス・オブジェクト (*appSpecificObject*) と汎用ビジネス・オブジェクト (*genericObject*) が null でない ことを確認します。
- *relDefName* 関係が一致関係であることを確認します。また、汎用ビジネス・オブジェクトを表す *relDefName* の参加者定義は、IBM WebSphere Business Integration Server Express **管理対象**として定義される必要があります。これらの設定を指定する方法の詳細については、267 ページの『一致関係の定義』を参照してください。

上記の検証のいずれかが失敗した場合、maintainChildVerb() は RelationshipRuntimeException 例外をスローします。

引き数の検証が完了すると、maintainChildVerb() が実行するアクションは、次の情報によって異なります。

- 呼び出しコンテキスト — マップの実行コンテキストでは、*map_ctx* 引き数 (cwExecCtx) の一部として渡されます。
- 動詞 — ソース・ビジネス・オブジェクトの場合:
  - 呼び出しコンテキスト EVENT_DELIVERY (または ACCESS_REQUEST) と SERVICE_CALL_RESPONSE のアプリケーション固有のビジネス・オブジェクト (*appSpecificObj*)
  - 呼び出しコンテキスト SERVICE_CALL_REQUEST の汎用ビジネス・オブジェクト (*genericObj*)

maintainChildVerb() が行うアクションの詳細については、315 ページの『子動詞設定の決定』を参照してください。表 105 から表 108 に、各呼び出しコンテキストのアクションを示します。

親オブジェクトの子属性に対する変換ステップで、このメソッドを呼び出すことができます。この子オブジェクトは、次のいずれかに参加できます。

- 子ビジネス・オブジェクトが固有キーを使用して関連している場合は、子ビジネス・オブジェクトを変換するサブマップのキー属性に対する変換ステップ。

通常は maintainChildVerb() を使用して、複合一致関係 (maintainCompositeRelationship()) に参加する子オブジェクトの動詞を設定します。ただし、単純一致関係 (maintainSimpleIdentityRelationship()) に参加する子オブジェクトの動詞を設定して呼び出すこともできます。

例

maintainChildVerb() を含める例については、303 ページの『複合一致関係に関連するマップの規則のカスタマイズ』を参照してください。

参照項目

maintainCompositeRelationship(), maintainSimpleIdentityRelationship()

314 ページの『ソース子動詞の設定』

maintainCompositeRelationship()

親マップ内から複合一致関係を保守します。

構文

```
public static void maintainCompositeRelationship(String relDefName,
      String particpntDefName, BusObj appSpecificBusObj,
      Object genericBusObjList, CxExecutionContext map_ctx)
```

パラメーター

relDefName
親属性が参加する、(Relationship Designer Express に定義された) 複合一致関係の名前。

particpntDefName
複合キーを含む参加者の名前。この参加者は常にアプリケーション固有です。

appSpecificBusObj
このマップで使用されるアプリケーション固有のビジネス・オブジェクトを含む変数。このビジネス・オブジェクトは、親ビジネス・オブジェクトです。

genericBusObjList
このマップで使用される汎用ビジネス・オブジェクトを含む変数。各汎用ビジネス・オブジェクトは、汎用親オブジェクトの子ビジネス・オブジェクトに含まれます。このパラメーターは、単一の汎用ビジネス・オブジェクト (BusObj)、または汎用ビジネス・オブジェクトの配列 (BusObjArray) にすることができます。

map_ctx
マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException

CxMissingIDException
動詞 Retrieve と SERVICE_CALL_REQUEST の呼び出しコンテキストを使用してマップを実行しているときに、関係表に参加者が存在しない場合です。コネクターは、ビジネス・オブジェクトをアプリケーションに送信せずに、「サービス呼び出し要求が失敗しました」というメッセージがコラボレーションに送信されます。

注記

maintainCompositeRelationship() メソッドは、*relDefName* 複合一致関係の *particpntDefName* 参加者に関連付けられた関係表を保守します。このメソッドは、参加者が異なるレベル (複合キー) の複数のビジネス・オブジェクトからキーを使用する関係を保守します。

注: maintainCompositeRelationship() メソッドは、子の複合キーが祖父母に依存する場合を処理できません。詳細については、302 ページの『General/APIs/Identity Relationship/Maintain Composite Relationship のアクション』を参照してください。

このメソッドは、*appSpecificObj* 親ビジネス・オブジェクトのすべての子ビジネス・オブジェクトについて繰り返され、*partDefName* 参加者の関係表の関係インスタンスを保守します。メソッドは、受け取った汎用ビジネス・オブジェクトの配列 (*genericObjs*) から関係インスタンス ID を取得します。子インスタンスごとに、maintainCompositeRelationship() は maintainSimpleIdentityRelationship() メソッドを呼び出して、実際の関係表の管理を実行します。
maintainSimpleIdentityRelationship() が実行するアクションは、次の情報によって異なります。

- 呼び出しコンテキスト — マップの実行コンテキストでは、*map_ctx* 引き数 (cwExecCtx) の一部として渡されます。
- 動詞 — ソース・ビジネス・オブジェクトの場合、次のいずれかです。
  - 呼び出しコンテキスト EVENT_DELIVERY (または ACCESS_REQUEST) と SERVICE_CALL_RESPONSE のアプリケーション固有のビジネス・オブジェクト (*appSpecificBusObj*)
  - 呼び出しコンテキスト SERVICE_CALL_REQUEST と ACCESS_RESPONSE の汎用ビジネス・オブジェクト (*genericBusObjList* 配列の 1 つのエレメント)

maintainSimpleIdentityRelationship() が行うアクションの詳細については、288 ページの『一致関係表のアクセス』を参照してください。表 95 から表 99 に、各呼び出しコンテキストのアクションを示します。

複合関係を保守するには、maintainCompositeRelationship() とともに maintainChildVerb() メソッドと updateMyChildren() メソッドを使用します。詳細については、303 ページの『複合一致関係に関連するマップの規則のカスタマイズ』を参照してください。

例

```
// This is an example of a code fragment in a parent map. It maintains
// the relationship table for all instances of a child object type for
// this application-specific parent object.

BusObjArray secondLevel2 =
      (BusObjArray)ObjFirstLevelBusObj2.get("MultiCardChild");

IdentityRelationship.maintainCompositeRelationship(
   "CmposRel",
   "AppSpPrt",
   ObjFirstLevelBusObj2,
   secondLevel2,
   cwExecCtx);
```

```
IdentityRelationship.updateMyChildren(
   "PCRel",
   "Parent",
   ObjFirstLevelBusObj2,
   "Child",
   "MultiCardChild",
   "CmposRel",
   "AppSpPrt",
   cwExecCtx);
```

maintainCompositeRelationship() を含む例については、303 ページの『複合一致関係に関連するマップの規則のカスタマイズ』を参照してください。

参照項目

updateMyChildren(), maintainChildVerb(), maintainSimpleIdentityRelationship()

300 ページの『複合一致関係の使用』

---

maintainSimpleIdentityRelationship()

親マップまたは子マップ内から単純一致関係を保守します。

構文

```
public static void maintainSimpleIdentityRelationship(
      String relDefName, String particpntDefName,
      BusObj appSpecificBusObj, BusObj genericBusObj,
      CxExecutionContext map_ctx)
```

パラメーター

relDefName この属性が参加する、(Relationship Designer Express に定義された) 単純一致関係の名前。

particpntDefName
アプリケーション固有のビジネス・オブジェクトを表す参加者定義の名前。

appSpecificBusObj
このマップで使用されるアプリケーション固有のビジネス・オブジェクトを含む変数。

genericBusObj このマップで使用される汎用ビジネス・オブジェクトを含む変数。

map_ctx マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException
: この例外がスローされた場合の詳細については、『注記』セクションを参照してください。

CxMissingIDException
: 動詞 Retrieve と SERVICE_CALL_REQUEST の呼び出しコンテキストを使用してマップを実行しているときに、関係表に参加者が存在しない場合です。コネクターは、ビジネス・オブジェクトをアプリケーションに送信せずに、「サービス呼び出し要求が失敗しました」というメッセージがコラボレーションに送信されます。

注記

maintainSimpleIdentityRelationship() メソッドは、*relDefName* 単純一致関係の *particpntDefName* 参加者に関連付けられた関係表を保守します。このメソッドは、参加者が同じレベルの複数のビジネス・オブジェクトから固有キーを使用する関係を保守します。

maintainSimpleIdentityRelationship() メソッドは、渡された引き数で以下の検証を実行します。

- *relDefName* 関係定義の名前を検証します。
- アプリケーション固有のビジネス・オブジェクトの *particpntDefName* 参加者定義の名前を検証します。
- アプリケーション固有のビジネス・オブジェクト (*appSpecificBusObj*) と汎用ビジネス・オブジェクト (*genericBusObj*) が null でない ことを確認します。
- *relDefName* 関係が一致関係であることを確認します。また、汎用ビジネス・オブジェクトを表す *relDefName* の参加者定義は、IBM WebSphere Business Integration Server Express **管理対象**として定義される必要があります。これらの設定を指定する方法の詳細については、267 ページの『一致関係の定義』を参照してください。
- 呼び出しコンテキストが有効であることを確認します (有効な呼び出しコンテキストのリストについては、表 94 を参照してください)。
- アプリケーション固有のビジネス・オブジェクトの動詞がサポートされていることを確認します。動詞は、Create、Update、Delete、Retrieve のいずれかでなければなりません。

上記の検証のいずれかが失敗した場合、maintainSimpleIdentityRelationship() は RelationshipRuntimeException 例外をスローします。

引き数の検証が完了すると、maintainSimpleIdentityRelationship() が実行するアクションは、次の情報によって異なります。

- 呼び出しコンテキスト — マップの実行コンテキストでは、*map_ctx* 引き数 (cwExecCtx) の一部として渡されます。
- 動詞 — ソース・ビジネス・オブジェクトの場合:
  - 呼び出しコンテキスト EVENT_DELIVERY (または ACCESS_REQUEST) と SERVICE_CALL_RESPONSE のアプリケーション固有のビジネス・オブジェクト (*appSpecificBusObj*)

– 呼び出しコンテキスト SERVICE_CALL_REQUEST と ACCESS_RESPONSE の汎用ビジネス・オブジェクト (*genericBusObj*)

maintainSimpleIdentityRelationship() メソッドは、呼び出しコンテキストと動詞の組み合わせごとに、参加者と関係インスタンスの基本的な追加と削除をすべて処理します。maintainSimpleIdentityRelationship() が行うアクションの詳細については、288 ページの『一致関係表のアクセス』を参照してください。表 95 から表 99 に、各呼び出しコンテキストのアクションを示します。

次のいずれかの場合に、このメソッドを呼び出すことができます。

- 親オブジェクトのキー属性に対する変換ステップ。
- 子ビジネス・オブジェクトが固有キーを使用して関連している場合は、子ビジネス・オブジェクトを変換するサブマップのキー属性に対する変換ステップ。

単純一致関係を保守するには、maintainSimpleIdentityRelationship() と maintainChildVerb() メソッドを使用します。詳細については、299 ページの『単純一致関係の変換規則の定義』を参照してください。

例

次の例は、インバウンド Clarify_BusOrg-to-Customer マップで、Clarify_BusOrg と汎用 Customer ビジネス・オブジェクト間の単純一致関係を保守します。

```
IdentityRelationship.maintainSimpleIdentityRelationship(
   "CustIdentity",
   "ClarBusOrg",
   ObjClarify_BusOrg,
   ObjCustomer,
   cxExecCtx);
```

maintainSimpleIdentityRelationship() を含む例については、299 ページの『単純一致関係の変換規則の定義』を参照してください。

参照項目

maintainChildVerb()

287 ページの『単純一致関係の使用』

---

updateMyChildren()

必要に応じて、一致関係の指定された親/子関係で子インスタンスを追加または削除します。

構文

```
void updateMyChildren(String parentChildRelDefName,
      String parentParticpntDef, BusObj parentBusObj,
      String childParticpntDef, String childAttrName,
      String childIdentityRelDefName,
      String childIdentityParticpntDefName,
      CxExecutionContext map_ctx)
```

パラメーター

parentChildRelDefName
親/子関係定義の名前。

parentParticpntDefName
親/子関係で親ビジネス・オブジェクトを表す参加者定義の名前。

parentBusObj 親ビジネス・オブジェクトを含む変数。

childParticpntDefName
親/子関係で子ビジネス・オブジェクトを表す参加者定義の名前。

childAttrName 親ビジネス・オブジェクトの属性の名前。親ビジネス・オブジェクトのタイプは、親/子関係に参加する子オブジェクトの名前です。例えば、顧客と住所の関係で、親オブジェクトに Address1 属性が含まれ、この属性がタイプ Address の子ビジネス・オブジェクトである場合、*childAttrName* 属性名は Address1 になります。

childIdentityRelDefName
子ビジネス・オブジェクトが参加する一致関係の名前。

childIdentityParticpntDefName
一致関係で子ビジネス・オブジェクトを表す参加者定義の名前。

map_ctx マップの実行コンテキスト。マップの実行コンテキストを渡すには、cwExecCtx 変数を使用します。この変数は、Map Designer Express によってすべてのマップに定義されます。

戻り値

なし。

例外

RelationshipRuntimeException
この例外がスローされた場合の詳細については、『注記』セクションを参照してください。

注記

updateMyChildren() メソッドは、*parentChildRelDefName* と *childIdentityRelDefName* 関係定義の関係表の子インスタンスを更新します。このメソッドは、親ビジネス・オブジェクトが子オブジェクトの追加または除去によって更新された場合の一致関係で有効です。updateMyChildren() を使用して、変更後イメージ (*parentBusObj* にあります) と変更前イメージ (関係表の情報) を比較し、変更後イメージで新しい、または削除された子オブジェクトを判別します。

注: updateMyChildren() メソッドは、子の複合キーが祖父母に依存する場合を処理できません。詳細については、311 ページの『Update My Children の使用についてのヒント』を参照してください。

updateMyChildren() メソッドは、渡された引き数で以下の検証を実行します。

- *parentChildrelDefName* 関係定義の名前 (最初の引き数) を検証します。

- *parentChildRelDefName* 関係が親/子関係で、*parentParticpntDefName* と *childParticpntDefName* が *parentChildRefDefName* 関係定義の一部であることを確認します。
- *childIdentityRelDefName* 関係が一致関係であることを確認します。また、汎用ビジネス・オブジェクトを表す *childIdentityRelDefName* の参加者定義は、IBM WebSphere Business Integration Server Express 管理対象として定義される必要があります。これらの設定を指定する方法の詳細については、267 ページの『一致関係の定義』を参照してください。
- *childIdentityParticpntDefName* が *childIdentityRefDefName* 関係定義の一部であることを確認します。

上記の検証のいずれかが失敗した場合、updateMyChildren() は RelationshipRuntimeException 例外をスローします。

引き数の検証が完了すると、updateMyChildren() メソッドは、必要に応じて、指定した親ビジネス・オブジェクトに属する子ビジネス・オブジェクトのリストに子を追加、または子を削除します。このメソッドは、親および子の参加者 (*parentParticpntDefName* と *childParticpntDefName*) の関係表に対して、次のタスクのいずれかを実行します。

- 新しい子オブジェクトごとに、updateMyChildren() は子インスタンスを追加します。

  このメソッドは、子の関係表には追加しません。これは、親ビジネス・オブジェクトに現在関連付けられているビジネス・オブジェクトはすべて、maintainCompositeRelationship() が呼び出されたときにすでに保守されているためです。
- 削除された子オブジェクトごとに、updateMyChildren() は対応する子インスタンスを除去します。

  このメソッドは、子の相互参照だけでなく、親/子関係表からも除去します。

updateMyChildren() メソッドでは、Relationship Designer Express で定義された親/子関係が必要です。この種類の関係を作成する方法の詳細については、309 ページの『親/子関係定義の作成』を参照してください。

注: 子ビジネス・オブジェクトに固有キーがある場合、子の参加者の属性は、子オブジェクトの固有キーになります。子オブジェクトに固有キーがない場合、子の参加者の属性はこの非固有キーになります。

例

updateMyChildren() と maintainCompositeRelationship() メソッドを含む例については、maintainCompositeRelationship() の『例』セクションを参照してください。

updateMyChildren() を含む例については、303 ページの『複合一致関係に関連するマップの規則のカスタマイズ』を参照してください。

参照項目

addMyChildren(), deleteMyChildren(), maintainCompositeRelationship(), maintainSimpleIdentityRelationship()

310 ページの『親ビジネス・オブジェクトに対する更新処理』

第 21 章 MapExeContext クラス

MapExeContext クラスは、マップの実行中に有効になるさまざまなランタイム値を照会および設定するためのメソッドを提供します。

表 137 に、MapExeContext クラスのメソッドを要約します。

表 137. MapExeContext メソッドの要約

メソッド	説明	ページ
getConnName()	現行マップ・インスタンスに関連付けられたコネクター名を検索します。	493
getInitiator()	現行マップ・インスタンスに関連付けられた呼び出しコンテキストを検索します。	494
getLocale()	マップの実行コンテキストに関連付けられたロケールを検索します。	495
getOriginalRequestBO()	現行マップ・インスタンスに関連付けられたオリジナル要求ビジネス・オブジェクトを検索します。	496
setConnName()	現行マップ・インスタンスに関連付けられたコネクター名を設定します。	496
setInitiator()	現行マップ・インスタンスに関連付けられた呼び出しコンテキストを設定します。	497
setLocale()	マップの実行コンテキストに関連付けられたロケールを設定します。	497

getConnName()

現行マップ・インスタンスに関連付けられたコネクター名を検索します。

構文

```
String getConnName()
```

パラメーター

なし。

戻り値

コネクター名を含む String を戻します。

例外

なし。

参照項目

setConnName()

getInitiator()

現行マップ・インスタンスに関連付けられた呼び出しコンテキストを検索します。

構文

```
String getInitiator()
```

パラメーター

なし。

戻り値

現行マップ・インスタンスの実行に対する呼び出しコンテキストを表す静的固定変数を戻します。呼び出しコンテキストは、次の値のいずれかです。

EVENT_DELIVERY
: マップされるソース・ビジネス・オブジェクトは、コネクターを介してアプリケーションから InterChange Server Express に送信されます。

ACCESS_REQUEST
: マップされるソース・オブジェクトは、アクセス・クライアントを介してアプリケーションから InterChange Server Express に送信されます。

SERVICE_CALL_REQUEST
: マップされるソース・オブジェクトは、コネクターを介して InterChange Server Express からアプリケーションに送信されます。

SERVICE_CALL_RESPONSE
: マップされるソース・オブジェクトは、サービス呼び出し要求が正常に完了した後に、コネクターを介してアプリケーションから InterChange Server Express に返されます。

SERVICE_CALL_FAILURE
: マップされるソース・オブジェクトは、サービス呼び出し要求が失敗した後に、コネクターを介してアプリケーションから InterChange Server Express に返されます。

ACCESS_RESPONSE
: マップされるソース・オブジェクトは、アクセス・クライアントを介して InterChange Server Express からアプリケーションに返されます。

例外

なし。

注記

呼び出しコンテキストは、マップの実行コンテキストの一部です。マップでの呼び出しコンテキストの使用法の詳細については、207 ページの『マップの実行コンテキストの理解』を参照してください。

例

以下の例では、マップ・ランタイム・イニシエーターと MapExeContext クラスで定義された定数を比較します。

```
String sInitiator = null;
sInitiator = cwMapCtx.getInitiator();
if(sInitiator.equals(MapExeContext.EVENT_DELIVERY))
     logInfo("**************Initiator =  MapExeContext.EVENT_DELIVERY.");
```

参照項目

getOriginalRequestBO(), setInitiator()

---

getLocale()

マップの実行コンテキストに関連付けられたロケールを検索します。

構文

```
Locale getLocale()
```

パラメーター

なし。

戻り値

マップの実行コンテキストの言語と国のコードを含む、ロケール・オブジェクトを戻します。

例外

なし。

注記

このメソッドは、MapExeContext タイプのマップ変数で実行する必要があります。この変数の名前は、cwMapCtx (システムで生成された場合)、またはマップ・コードを自動的に生成しない環境 (コラボレーション内など) でマップを呼び出した場合には、自分で指定します。

例

次の例では、マップの実行コンテキストのロケールを変数に取得して、トレース・ステートメントを使用して報告します。

```
Locale mapLocale = cwMapCtx.getLocale();
String mapLocaleToString = mapLocale.toString();
trace(3, "THE MAP LOCALE IS: " + mapLocaleToString);
```

参照項目

setLocale()

getOriginalRequestBO()

現行マップ・インスタンスに関連付けられたオリジナル要求ビジネス・オブジェクトを検索します。

構文

```
BusObj getOriginalRequestBO()
```

パラメーター

なし。

戻り値

次の表に示すように、マップのオリジナル要求ビジネス・オブジェクトを戻します。

呼び出しコンテキスト	オリジナル要求ビジネス・オブジェクト
EVENT_DELIVERY, ACCESS_REQUEST	アプリケーションから発生したアプリケーション固有のビジネス・オブジェクト
SERVICE_CALL_REQUEST, SERVICE_CALL_FAILURE SERVICE_CALL_RESPONSE	InterChange Server Express から送信された汎用ビジネス・オブジェクト SERVICE_CALL_REQUEST から送信された汎用ビジネス・オブジェクト
ACCESS_RESPONSE	最初にアクセス要求から発生したアプリケーション固有のビジネス・オブジェクト

例外

なし。

注記

オリジナル要求ビジネス・オブジェクトは、マップの実行コンテキストの一部です。 `getOriginalRequestBO()` メソッドは、オリジナル要求ビジネス・オブジェクトを戻します。このビジネス・オブジェクトは、マップの呼び出しコンテキストによって異なります。マップでのこのビジネス・オブジェクトの使用法の詳細については、210 ページの『オリジナル要求ビジネス・オブジェクト』を参照してください。

参照項目

`getInitiator()`

setConnName()

現行マップ・インスタンスに関連付けられたコネクター名を設定します。

構文

```
void setConnName(String connectorName)
```

パラメーター

connectorName コネクターの名前。

戻り値

なし。

例外

なし。

注記

指定したコネクターのコントローラーは、InterChange Server Express で実行されている必要があります。

参照項目

getConnName()

---

setInitiator()

現行マップ・インスタンスに関連付けられた呼び出しコンテキストを設定します。

構文

void setInitiator(String *callingContext*)

パラメーター

callingContext

次の値のいずれかを含むストリング。

EVENT_DELIVERY	マップされるソース・オブジェクトは、コネクターを介してアプリケーションから InterChange Server Express に送信されます。
ACCESS_REQUEST	マップされるソース・オブジェクトは、アクセス・クライアントを介してアプリケーションから InterChange Server Express に送信されます。
SERVICE_CALL_REQUEST	マップされるソース・オブジェクトは、コネクターを介して InterChange Server Express からアプリケーションに送信されます。
SERVICE_CALL_RESPONSE	マップされるソース・オブジェクトは、サービス呼び出し要求が正常に完了した後に、コネクターを介してアプリケーションから InterChange Server Express に返されます。

SERVICE_CALL_FAILURE	マップされるソース・オブジェクトは、サービス呼び出し要求が失敗した後に、コネクターを介してアプリケーションから InterChange Server Express に返されます。
ACCESS_RESPONSE	マップされるソース・オブジェクトは、アクセス・クライアントを介して InterChange Server Express からアプリケーションに返されます。

戻り値

なし。

例外

なし。

注記

呼び出しコンテキストは、マップの実行コンテキストの一部です。呼び出しコンテキストは、ソース・ビジネス・オブジェクトがマップされている方向を示します。マップでの呼び出しコンテキストの使用法の詳細については、207 ページの『マップの実行コンテキストの理解』を参照してください。

参照項目

getInitiator()

---

setLocale()

マップの実行コンテキストに関連付けられたロケールを設定します。

構文

```
void setLocale(Locale newLocale)
```

パラメーター

newLocale	マップの実行コンテキストが設定される新しい Locale オブジェクト。

戻り値

なし。

例外

なし。

注記

このメソッドは、MapExeContext タイプのマップ変数で実行する必要があります。この変数の名前は、cwMapCtx (システムで生成された場合)、またはマップ・コードを自動的に生成しない環境 (コラボレーション内など) でマップを呼び出した場合には、自分で指定します。

マップによって作成されるビジネス・オブジェクトのロケールは、マップの実行コンテキストのロケールに影響されます。マップのロジックの一部として、マップの実行コンテキストのロケールを変更すると、新しいロケールがビジネス・オブジェクトにコピーされます。これは、ユーザーが変更可能なロジックの実行が完了したとき (つまり、Map Designer Express のダイアグラムに表示される変換が完了したとき) に行われます。この API を使用すると、ビジネス・オブジェクトのロケールを、マップを入力したときと異なるロケールに変更できます。

例

以下のコードは、新しい Locale オブジェクトを定義して、マップの実行コンテキストを新しいロケール値に設定し、マップの実行コンテキストのロケールを報告します。

```
Locale newLocale = new Locale("ja", "JP");
cwMapCtx.setLocale(newLocale);
trace(3, "THE MAP LOCALE IS NOW: " + cwMapCtx.getLocale().toString());
```

参照項目

getLocale()

使用すべきでないメソッド

MapExeContext クラスのメソッドには、以前のバージョンではサポートされていたが現在のバージョンではサポートされていないものがいくつかあります。これらの *使用すべきでないメソッド* によりエラーが生成されることはありませんが、CrossWorlds では、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。

MapExeContext クラスの使用すべきでないメソッドを表 138 に示します。これまでに Map Designer Express を使用したことがない場合は、このセクションは無視してください。

表 138. 使用すべきでないメソッド、MapExeContext クラス

以前のメソッド	代替メソッド
getGenericBO()	getOriginalRequestBO()

第 22 章 Participant クラス

この章では、Participant クラスのオブジェクトに対して操作を行うメソッドについて説明します。参加者インスタンスは、関係インスタンスで使用されます。各参加者インスタンスには、以下の情報が含まれています。

- 関係定義の名前
- 関係インスタンス ID
- 参加者定義の名前
- 参加者に関連付けられたデータ

Participant クラスは、特定の参加者について以下の各値を設定および検索するためのメソッドを提供します。

表 139 に、Participant クラスのメソッドを要約します。

表 139. Participant メソッドの要約

メソッド	説明	ページ
Participant()	Participant インスタンスを新規に作成します。	501
getBusObj()、getString()、getLong()、getInt()、getDouble()、 getFloat()、getBoolean()	Participant インスタンスに関連付けられたデータを検索します。	503
getInstanceId()	Participant インスタンスが参加している関係の関係インスタンス ID を検索します。	504
getParticipantDefinition()	Participant インスタンスが作成された参加者定義名を検索します。	504
getRelationshipDefinition()	Participant インスタンスが参加している関係定義の名前を検索します。	505
set()	Participant インスタンスに関連付けられたデータを設定します。	505
setInstanceId()	Participant インスタンスが参加している関係のインスタンス ID を設定します。	506
setParticipantDefinition()	Participant インスタンスが作成された参加者定義名を設定します。	506
setRelationshipDefinition()	Participant インスタンスが参加している関係定義を設定します。	507

Participant()

Participant インスタンスを新規に作成します。

構文

関係インスタンスの既存の参加者に新しい参加者インスタンスを追加するには、以下のようにします。

```
Participant(String relDefName,String partDefName,
int instanceId,BusObj partData)
Participant(String relDefName,String partDefName,
 int instanceId,String partData)
Participant(String relDefName,String partDefName,
int instanceId,long partData)
Participant(String relDefName,String partDefName,
int instanceId,int partData)
Participant(String relDefName,String partDefName,
int instanceId,double partData)
Participant(String relDefName,String partDefName,
int instanceId,float partData)
Participant(String relDefName,String partDefName,
int instanceId,boolean partData)
```

関係インスタンスのない新しい参加者インスタンスを作成するには、以下のようにします。

```
Participant(String relDefName,String partDefName, BusObj partData)
Participant(String relDefName,String partDefName, String partData)
Participant(String relDefName,String partDefName, long partData)
Participant(String relDefName,String partDefName, int partData)
Participant(String relDefName,String partDefName, double partData)
Participant(String relDefName,String partDefName, float partData)
Participant(String relDefName,String partDefName, boolean partData)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者を説明する参加者定義の名前。

instanceId 新しい参加者インスタンスを受け取るための関係インスタンスの関係インスタンス ID。

participantData
参加者インスタンスに関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

戻り値

新しい参加者インスタンスを戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

注記

このメソッドは、Participant クラスのコンストラクターです。以下の形式を使用します。

- コンストラクターの第 1 の形式は、*instanceId* で識別される関係インスタンスに新しい参加者インスタンスを追加します。

- 第 2 の形式は、関連する関係インスタンスのない、新しい参加者インスタンスを作成します。この参加者インスタンスを、IdentityRelationship.addMyChildren() または Relationship.create() への引き数として使用して、新しい関係インスタンスを作成できます。Relationship.create() メソッドを使用するには、関係インスタンス ID がないことが要件です。

participantData パラメーターに関連付けるデータは、関係の種類によって異なります。

- 一致関係の参加者インスタンスを作成するには、*participantData* パラメーターとしてビジネス・オブジェクトを使用します。
- 参照関係の参加者を作成するには、*participantData* パラメーターに String、long、int、double、float、boolean のいずれかのデータ型を使用します。

例

```
// create a participant instance with no relationship instance ID
participant p = new Participant(myRelDef,myPartDef,myBusObj);

// create a relationship instance
int relInstanceId = Relationship.addParticipant(p);
```

参照項目

addMyChildren(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

---

getBusObj()、getString()、getLong()、 getInt()、getDouble()、getFloat()、 getBoolean()

Participant インスタンスに関連付けられたデータを検索します。

構文

```
BusObj getBusObj()
String getString()
long getLong()
int getInt()
double getDouble()
float getFloat()
boolean getBoolean()
```

戻り値

この参加者インスタンスに関連付けられたデータを戻します。このデータ値のデータ型は、メソッド名に含まれます。例えば、getBoolean() は boolean 値を、getBusObj() は BusObj 値を、getDouble() は double 値を戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

set(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

getInstanceId()

Participant インスタンスが参加している関係の関係インスタンス ID を検索します。

構文

```
int getInstanceId()
```

戻り値

この Participant インスタンスが参加している関係インスタンスのインスタンス ID を表す整数を戻します。Participant インスタンスが関係インスタンスのメンバーではない 場合、このメソッドは定数 INVALID_INSTANCE_ID を戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

setInstanceId(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

getParticipantDefinition()

Participant インスタンスが作成された参加者定義名を検索します。

構文

```
String getParticipantDefinition()
```

戻り値

この参加者インスタンスに関連付けられた参加者定義の名前を含む String を戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

setParticipantDefinition(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

getRelationshipDefinition()

Participant インスタンスが参加している関係定義の名前を検索します。

構文

```
String getRelationshipDefinition()
```

戻り値

この参加者インスタンスが参加する関係定義の名前を含む String を戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

setRelationshipDefinition(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

set()

Participant インスタンスに関連付けられたデータを設定します。

構文

```
void set(BusObj partData)
void set(String partData)
void set(long partData)
void set(int partData)
void set(double partData)
void set(float partData)
void set(boolean partData)
```

パラメーター

partData Participant インスタンスに関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

戻り値

なし。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

注記

参加者データをビジネス・オブジェクト (BusObj タイプ) になるように設定する場合は、関係定義と参加者定義を先に設定しておく必要があります。参加者データを別のデータ型に設定する場合は、設定を指定する順序は関係ありません。

参照項目

getBusObj()、getString()、getLong()、 getInt()、getDouble()、 getFloat()、getBoolean()，259 ページの『第 7 章 関係定義の作成』，48 ページの『サブマップを使用した変換』

---

setInstanceId()

Participant インスタンスが参加している関係のインスタンス ID を設定します。

構文

```
void setInstanceId(int id)
```

パラメーター

id 関係のインスタンス ID。

戻り値

なし。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

注記

setInstanceId() を使用すると、参加者インスタンスをパラメーターとして Participant() メソッドまたは create() メソッドに渡すときに、関係インスタンス ID が除去されます。この場合は、インスタンス ID を定数 INVALID_INSTANCE_ID に設定します。

例

```
// wipe out the relationship instance ID
myParticipant.setInstanceId(Participant.INVALID_INSTANCE_ID);

// pass the participant instance to the create() method
int newRelId = create(myParticipant);
```

参照項目

getInstanceId()，259 ページの『第 7 章 関係定義の作成』，48 ページの『サブマップを使用した変換』

---

setParticipantDefinition()

Participant インスタンスが作成された参加者定義名を設定します。

構文

```
void setParticipantDefinition(String partDefName)
```

パラメーター

partDefName　Participant インスタンスが作成された参加者定義の名前。

戻り値

なし。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

setParticipantDefinition(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

---

setRelationshipDefinition()

Participant インスタンスが参加している関係定義を設定します。

構文

void setRelationshipDefinition(String *relDefName*)

パラメーター

relDefName　関係定義の名前。

戻り値

なし。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

参照項目

getRelationshipDefinition(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

第 23 章 Relationship クラス

この章では、IBM WebSphere Business Integration Server Express 定義のクラス Relationship のオブジェクトに対して操作を行うメソッドについて説明します。Relationship クラスは*関係インスタンス* と呼ばれる関係のランタイム・インスタンスを操作するためのメソッドを提供します。通常は、一致関係または静的参照としてマップされるビジネス・オブジェクト属性の変換ステップでこのメソッドを使用します。このクラスのメソッドを使用して関係属性をプログラミングする方法の詳細については、48 ページの『サブマップを使用した変換』を参照してください。

このクラスのメソッドのほとんどは、指定したパラメーターのバリエーションをサポートしています。バリエーションは通常、次のガイドラインに従います。

- 関係インスタンスで特定の参加者を識別する場合、通常は、関係定義名、参加者定義名、関係インスタンス ID、および参加者に関連付けられたビジネス・オブジェクトを指定します。
- 関係定義名、参加者定義名、インスタンス ID、およびビジネス・オブジェクトを属性として含む Participant インスタンスを指定することもできます。
- 操作によっては、関係インスタンス ID (新しい関係を作成する場合など) やビジネス・オブジェクト名を省略できることもあります。

ほとんどの場合、(例えば retrieve() 呼び出しの結果として) Participant インスタンスがあれば、各属性を個別に指定しなくても、パラメーターとして Relationship クラス・メソッドにインスタンスを簡単に渡すことができます。

このクラスのメソッドはすべて、静的として宣言されます。既存の関係インスタンスから、または Relationship クラスを参照することでメソッドを呼び出すことができます。

表 140 に、Relationship クラスのメソッドについて要約します。

表 140. Relationship メソッドの要約

メソッド	説明	ページ
静的メソッド		
addParticipant()	新しい参加者を関係インスタンスに追加します。	510
create()	新しい関係インスタンスを作成します。	512
deactivateParticipant()	1 つ以上の関係インスタンスから参加者を非アクティブにします。	513
deactivateParticipantByInstance()	特定の関係インスタンスから参加者を非アクティブにします。	514
deleteParticipant()	1 つ以上の関係インスタンスから参加者インスタンスを除去します。	516
deleteParticipantByInstance()	特定の関係インスタンスから参加者を除去します。	517
getNewID()	関係定義名に基づいて、次に使用可能な関係の関係インスタンス ID を戻します。	518

表 140. Relationship メソッドの要約 (続き)

メソッド	説明	ページ
retrieveInstances()	特定の参加者インスタンスを含むゼロまたは複数の関係インスタンスの関係インスタンス ID のみを検索します。	518
retrieveParticipants()	関係インスタンスからゼロまたは複数の参加者を検索します。	520
updateParticipant()	1 つ以上の関係インスタンスの参加者を更新します。	521
updateParticipantByInstance()	特定の関係インスタンスの参加者を更新します。	522

addParticipant()

新しい参加者を関係インスタンスに追加します。

構文

既存の関係インスタンスに新しい参加者を追加するには、以下のようにします。

```
int addParticipant
 (String relDefName,
String partDefName,
int instanceId,BusObj partData)

int addParticipant
 (String relDefName,
String partDefName,
 int instanceId,String partData)

int addParticipant
 (String relDefName,
String partDefName,int instanceId,
long partData)

int addParticipant
 (String relDefName,
String partDefName,int instanceId,
int partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId,
double partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId, float partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId,
boolean partData)
```

新しい関係インスタンスに参加者を追加するには、以下のようにします。

```
int addParticipant
 (String relDefName,
String partDefName,
      BusObj partData)
int addParticipant
 (String relDefName,
String partDefName,
      String partData)
int addParticipant
 (String relDefName,
String partDefName,
      long partData)
int addParticipant
 (String relDefName,
String partDefName,
      int partData)
int addParticipant
 (String relDefName,
String partDefName,
      double partData)
int addParticipant
 (String relDefName,
String partDefName,
      float partData)
int addParticipant
 (String relDefName,
String partDefName,
      boolean partData)
```

関係インスタンスに既存の参加者インスタンスを追加するには、以下のようにします。

```
int addParticipant(Participant participant)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

instanceId 新しい参加者を受け取るための関係インスタンスの関係インスタンス ID。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

participant 関係に追加する参加者。

戻り値

新しい参加者を受け取るための関係のインスタンス ID を表す整数を戻します。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

注記

メソッドの第 1 の形式は、指定した関係インスタンスに新しい参加者を追加します。それぞれの変形は、参加者に関連するデータのさまざまなデータ型をサポートしています。

第 2 の形式は、関係インスタンスを指定しないため、新しい関係インスタンスを作成して、新しい参加者を追加します。この場合、戻り値は、新しく作成された関係インスタンスのインスタンス ID になります。それぞれの変形は、参加者に関連するデータのさまざまなデータ型をサポートしています。

第 3 の形式は、渡された参加者インスタンスを、参加者インスタンスで指定された関係インスタンスに追加します。参加者インスタンスに関係インスタンス ID がない場合、新しい関係インスタンスが作成され、新しいインスタンス ID が戻されます。

addParticipant() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

create()

---

create()

新しい関係インスタンスを作成します。

構文

```
int create(String relDefName, String partDefName, BusObj partData)
int create(String relDefName, String partDefName, String partData)
int create(String relDefName, String partDefName, long partData)
int create(String relDefName, String partDefName, int partData)
int create(String relDefName, String partDefName, double partData)
int create(String relDefName, String partDefName, float partData)
int create(String relDefName, String partDefName, boolean partData)
int create(Participant participant)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

participant 関係の最初の参加者。

戻り値

新しい関係の関係インスタンス ID を表す整数を戻します。

例外

RelationshipRuntimeException

注記

create() メソッドは、*partDefName* 参加者定義の参加者インスタンスを使用して、新しい関係インスタンスを作成します。 *partData* 引き数を使用して、この新しい

参加者インスタンスのデータを指定できます。このメソッドを呼び出した後に、addMyChildren() を呼び出して、関係インスタンスにさらに参加者を追加できます。

メソッドの最後の形式では、*participant* パラメーターは、関係インスタンス ID を持つことができません。通常は、参加者インスタンスは関係インスタンス ID を持っています。このメソッドは、新しい関係インスタンスを作成するため、参加者インスタンスに関連付けられたインスタンスがまだないことを確認する必要があります。この操作を行うには、(Participant クラスの) setInstanceId() メソッドを使用して、インスタンス ID を INVALID_INSTANCE_ID 定数に設定します。

create() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

addMyChildren(), setInstanceId()

---

deactivateParticipant()

1 つ以上の関係インスタンスから参加者を非アクティブにします。

構文

```
void deactivateParticipant(String relDefName,
String partDefName,
      BusObj partData)

void deactivateParticipant(String
relDefName,
String partDefName,
      String partData)

void deactivateParticipant(String relDefName,
String partDefName,
       long partData)

void deactivateParticipant(String relDefName,
String partDefName,
       int partData)

void deactivateParticipant(String relDefName,
String partDefName,
       double partData)

void deactivateParticipant(String relDefName,
String partDefName,
      float partData)

void deactivateParticipant(String relDefName,
String partDefName,
      boolean partData)

void deactivateParticipant(Participant participant)
```

パラメーター

relDefName　　関係定義の名前。

partDefName 参加者定義の名前。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

participant 関係で非アクティブ化される参加者。

戻り値

なし。

例外

RelationshipRuntimeException

注記

deactivateParticipant() メソッドは、*relDefName* のすべてのインスタンスから参加者を非アクティブ化します。インスタンスでは、*partData* は *partDefName* に関連付けられます。このメソッドは、関係表から参加者を除去しません。関係表に存在するレコードを保持している場合に、参加者を除去するには、このメソッドを使用します。

非アクティブ化された参加者を表示するには、関係表に直接照会します。表名を検索して、特定の関係の情報にアクセスするには、Relationship Designer Express を使用して関係定義を開き、「編集」メニューから「拡張設定」を選択します。これらの設定の詳細については、272 ページの『関係の拡張設定の指定』を参照してください。

重要: deactivateParticipant() は、実際には関係表から参加者の行を除去しないため、参加者を削除するために、このメソッドを機械的に使用しないでください。このメソッドを使用すると、関係表が無駄に大きくなります。

deactivateParticipant() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deleteParticipant(), deactivateParticipantByInstance(), 259 ページの『第 7 章 関係定義の作成』, 48 ページの『サブマップを使用した変換』

---

deactivateParticipantByInstance()

特定の関係インスタンスから参加者を非アクティブにします。

構文

```
void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, BusObj partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, String partData ] )

void deactivateParticipantByInstance(String relDefName,
```

```
       String partDefName, int instanceId [, long partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, int partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, double partData ] )

void deactivateParticipantByInstance(String relDefName,
      String partDefName, int instanceId [, float partData ] )

void deactivateParticipantByInstance(String relDefName,
      String partDefName, int instanceId [, boolean partData ] )
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

instanceId 参加者が属する関係インスタンスの ID。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。これはオプショナル・パラメーターです。

戻り値

なし。

例外

RelationshipRuntimeException - 203 ページの『例外処理』を参照してください。

注記

deactivateParticipantByInstance() メソッドは、関係インスタンス ID *instanceID* が識別する関係インスタンスから、指定された参加者を非アクティブにします。ただし、このメソッドは、関係表から参加者を除去しません。関係表に存在するレコードを保持している場合に、参加者を除去するには、このメソッドを使用します。

非アクティブ化された参加者を表示するには、関係表に直接照会します。表名を検索して、特定の関係の情報にアクセスするには、Relationship Designer Express を使用して関係定義を開き、「編集」メニューから「拡張設定」を選択します。これらの設定の詳細については、272 ページの『関係の拡張設定の指定』を参照してください。

重要: deactivateParticipantByInstance() は、実際には関係表から参加者の行を除去しないため、参加者を削除するために、このメソッドを機械的に使用しないでください。このメソッドを使用すると、関係表が無駄に大きくなります。

deactivateParticipantByInstance() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deleteParticipant(), deactivateParticipant()

---

deleteParticipant()

1 つ以上の関係インスタンスから参加者インスタンスを除去します。

構文

```
void deleteParticipant(String relDefName, String partDefName,
BusObj partData)
void deleteParticipant(String relDefName, String partDefName,
String partData)
void deleteParticipant(String relDefName, String partDefName,
long partData)
void deleteParticipant(String relDefName, String partDefName,
int partData)
void deleteParticipant(String relDefName, String partDefName,
double partData)
void deleteParticipant(String relDefName, String partDefName,
float partData)
void deleteParticipant(String relDefName, String partDefName,
boolean partData)

void deleteParticipant(Participant participant)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

participant 関係から削除する参加者を表す参加者インスタンス。

戻り値

なし。

例外

RelationshipRuntimeException

注記

deleteParticipant() メソッドは、*relDefName* のすべてのインスタンスから指定した参加者を削除します。インスタンスでは、*partData* は *partDefName* に関連付けられ、基本となる関係表から参加者を削除します。

deleteParticipant() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deactivateParticipant(), deleteParticipantByInstance()

deleteParticipantByInstance()

特定の関係インスタンスから参加者を除去します。

構文

```
void deleteParticipantByInstance(String relDefName,
    String partDefName, int instanceId [, BusObj partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, String partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, long partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, int partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, double partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, float partData] )

void deleteParticipantByInstance(String relDefName,
    String partDefName, int instanceId [, boolean partData] )
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

instanceId 参加者が属する関係インスタンスの ID。

partData 参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。これはオプショナル・パラメーターです。

戻り値

なし。

例外

RelationshipRuntimeException

注記

deleteParticipantByInstance() メソッドは、*instanceId* 関係インスタンス ID で識別される関係から参加者インスタンスを削除します。このメソッドは、関係インスタンスと、基本となる関係表から参加者を除去します。

オプションの *partData* パラメーターを指定すると、deleteParticipantByInstance() は、*partData* が *partDefName* 参加者定義に関連付けられている場合のみ、参加者インスタンスを削除します。

メソッドの最後の形式は、参加者インスタンスを唯一のパラメーターとして受け入れます。参加者インスタンスには、関係定義名、参加者定義名、およびインスタンス ID と参加者データのいずれかが含まれている必要があります。

deleteParticipantByInstance() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deactivateParticipant()

---

getNewID()

関係定義名に基づいて、次に使用可能な関係の関係インスタンス ID を戻します。

構文

```
public static int getNewID(String relDefName)
```

パラメーター

relDefName　　関係定義の名前。

戻り値

関係定義名に基づいて、関係インスタンス ID を戻します。

例外

RelationshipRuntimeException

注記

関係インスタンス ID は、一般的な IBM WebSphere Business Integration Server Express 一致関係で汎用 ID として使用できるため、この新しい ID は、汎用対汎用関係の汎用 ID として使用できます。

---

retrieveInstances()

特定の参加者インスタンスを含むゼロまたは複数の関係インスタンスの関係インスタンス ID のみを検索します。

構文

```
int[] retrieveInstances(String relDefName,
String partDefName,
     BusObj partData)

int[] retrieveInstances(String relDefName,
```

```
String partDefName,
      String partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      long partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      int partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      double partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      float partData)

int[] retrieveInstances(String relDefName,
String partDefName,
     boolean partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      BusObj partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      String partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
       long partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      int partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      double partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
       float partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      boolean partData)

int[] retrieveInstances(String relDefName, BusObj partData)
int[] retrieveInstances(String relDefName, String partData)
int[] retrieveInstances(String relDefName, long partData)
int[] retrieveInstances(String relDefName, int partData)
int[] retrieveInstances(String relDefName, double partData)
int[] retrieveInstances(String relDefName, float partData)
int[] retrieveInstances(String relDefName, boolean partData)
```

パラメーター

relDefName　関係定義の名前。

partDefName　参加者定義の名前。

partDefList　参加者定義のリスト。

partData　参加者に関連付けられたデータ。データ型は、BusObj、String、long、int、double、float、boolean のいずれかです。

戻り値

参加者を含む関係のインスタンス ID である整数の配列を戻します。

例外

RelationshipRuntimeException

注記

retrieveInstances() メソッドは、インバウンド・マップで参照関係をインプリメントします。このメソッドは、指定した参加者インスタンス (*partDefList* と *partData*、または *partData* のみ) に関連付けられた関係表から、関係インスタンス ID を取得します。*relDefName* 関係定義に関連付けられた属性のみ を検索します。ビジネス・オブジェクトの他の属性には、入力しません。関係定義に関連付けられた属性は通常、キー属性と、明示的に選択したその他の属性です。関係定義の詳細については、259 ページの『第 7 章 関係定義の作成』を参照してください。

retrieveInstances() で指定したデータの関係インスタンスが見つからなかった場合、例外は生成されません。関係表にデータがない場合でも、ルックアップが適切に実行されなかったとは限りません。retrieveInstances() で値が見つからなかったときに例外を生成したい場合は、メソッドが戻すインスタンス ID の値を検査して、値が null の場合は、MapFailureException を明示的に生成する必要があります。

retrieveInstances() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

addMyChildren(), deactivateParticipant(), deleteParticipant(), retrieveParticipants()

285 ページの『参照関係のマップ変換のカスタマイズ』

---

retrieveParticipants()

関係インスタンスからゼロまたは複数の参加者を検索します。

構文

```
Participant[] retrieveParticipants(String relDefName,
       String partDefName, int instanceId)

Participant[] retrieveParticipants(String relDefName,
       String[] partDefList, int instanceId)

Participant[] retrieveParticipants(String relDefName,
        int instanceId)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

partDefList 参加者定義のリスト。

instanceId 参加者が属する関係インスタンスの関係インスタンス ID。

戻り値

参加者インスタンスの配列を戻します。

例外

RelationshipRuntimeException

注記

retrieveParticipants() メソッドは、アウトバウンド・マップで参照関係をインプリメントします。このメソッドは、指定した *instanceID* 関係インスタンス ID に関連付けられた関係表から参加者インスタンスを取得します。*relDefName* 関係定義に関連付けられた属性のみ を検索します。ビジネス・オブジェクトの他の属性には、入力しません。関係定義に関連付けられた属性は通常、キー属性と、明示的に選択したその他の属性です。関係定義の詳細については、 259 ページの『第 7 章 関係定義の作成』を参照してください。

retrieveParticipants() が RelationshipRuntimeException を生成すると、null 値の *instanceId* を受け取ります。retrieveInstances() メソッドが、一致するインスタンス ID を戻したかどうか保証できない場合は、retrieveParticipants() を呼び出す*前* に、null 値について *instanceId* の値を検査してください。

retrieveParticipants() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

addMyChildren(), deactivateParticipant(), deleteParticipant(), retrieveInstances()

285 ページの『参照関係のマップ変換のカスタマイズ』

---

updateParticipant()

1 つ以上の関係インスタンスの参加者を更新します。

構文

```
void updateParticipant(String relDefName, String partDefName,
BusObj partData)
```

パラメーター

relDefName 関係定義の名前。

partDefName *relDefName* 関係に参加している参加者定義の名前。

partData 参加者に関連付けられたデータ。データ型は BusObj です。

戻り値

なし。

例外

RelationshipRuntimeException

注記

updateParticipant() メソッドは、*relDefName* のインスタンスの *partData* を更新します。*partData* は、*partDefName* に関連付けられます。このメソッドは、指定した参加者に関連付けられたビジネス・オブジェクトの非キー属性を更新します。関係定義に関連付けられた属性のみが更新されます。

updateParticipant() メソッドは、次の *relDefName* 関係の参加者インスタンスをすべて更新します。

- *partDefName* の参加者定義
- *partData* ビジネス・オブジェクトのキー値と一致するキー値

このメソッドは、partData ビジネス・オブジェクトの値を使用して、参加者インスタンスの非キー属性を更新します。関係定義に関連付けられた属性のみが更新されます。

ビジネス・オブジェクトではない キー属性または参加者タイプ (String、long、int、double、float、または boolean など) を変更するには、deleteParticipant() または deactivateParticipant() を使用して参加者を削除してから、addMyChildren() を使用して、新しい参加者を追加する必要があります。

updateParticipant() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deleteParticipant(), deactivateParticipant(), addMyChildren()

---

updateParticipantByInstance()

特定の関係インスタンスの参加者を更新します。

構文

特定の関係インスタンスの参加者を更新するには、以下のようにします。

```
void updateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [ , BusObj partData ] )

void updateParticipantByInstance(Participant participant)
```

パラメーター

relDefName 関係定義の名前。

partDefName 参加者定義の名前。

instanceId 参加者が属する関係を識別する関係インスタンス ID。

partData 参加者に関連付けられたデータ。データ型は BusObj です。このパラメーターはオプションです。

participant 関係で更新される参加者。

戻り値

なし。

例外

RelationshipRuntimeException

注記

updateParticipantByInstance() メソッドは、指定した参加者に関連付けられたビジネス・オブジェクトの非キー属性を更新します。関係定義に関連付けられた属性のみが更新されます。

ビジネス・オブジェクトではないキー属性または参加者タイプ (String、long、int、double、float、または boolean) を変更するには、deleteParticipant() または deactivateParticipant() を使用して参加者を削除してから、addMyChildren() を使用して、新しい参加者を追加する必要があります。

updateParticipantByInstance() メソッドは、静的として宣言されたクラス・メソッドです。既存の関係インスタンスから、または Relationship クラスを参照することでこのメソッドを呼び出すことができます。

参照項目

deleteParticipant(), deactivateParticipant(), addMyChildren()

使用すべきでないメソッド

Relationship クラスのメソッドには、IdentityRelationship クラスに移動されたものもあります。これらの*使用すべきでないメソッド* によりエラーが生成されることはありませんが、CrossWorlds では、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。

Relationship クラスの使用すべきでないメソッドを表 141 に示します。

表 141. Relationship クラスの使用すべきでないメソッド

以前のメソッド	代替メソッド
addMyChildren()	IdentityRelationship クラスの addMyChildren()
deleteMyChildren()	IdentityRelationship クラスの deleteMyChildren()
maintainCompositeRelationship()	IdentityRelationship クラスの maintainCompositeRelationship()
maintainSimpleIdentityRelationship()	IdentityRelationship クラスの maintainSimpleIdentity Relationship()
updateMyChildren()	IdentityRelationship クラスの updateMyChildren()

第 24 章 UserStoredProcedureParam クラス

UserStoredProcedureParam クラスは、ストアード・プロシージャーへの引き数値を処理するためのメソッドを提供します。ストアード・プロシージャーは、リレーションシップ・データベースで実行します。UserStoredProcedureParam オブジェクトは、ストアード・プロシージャーの単一のパラメーターについて記述します。

要確認: UserStoredProcedureParam クラスとそのメソッドは、後方互換性のためにのみ サポートされています。これらの*使用すべきでないメソッド* によりエラーが生成されることはありませんが、これらのメソッドを使用せずに、既存のコードを新しいメソッドに移行することを推奨します。使用すべきでないメソッドは、今後のリリースで除外されることがあります。新しいマップ開発では、CwDBStoredProcedureParam クラスとそのメソッドを使用して、ストアード・プロシージャーへの引き数を提供します。

表 142 に、UserStoredProcedureParam クラスのメソッドについて要約します。

表 142. UserStoredProcedureParam メソッドの要約

メソッド	説明	ページ
UserStoredProcedureParam()	ストアード・プロシージャーのパラメーターに関する引き数情報を保持する、UserStoredProcedureParam の新しいインスタンスを作成します。	526
getParamDataTypeJavaObj()	このストアード・プロシージャー・パラメーターのデータ型を、Integer、Double、または String などの Java Object として検索します。	526
getParamDataTypeJDBC()	このストアード・プロシージャー・パラメーターのデータ型を、整数の JDBC データ型として検索します。	527
getParamIndex()	このストアード・プロシージャー・パラメーターのインデックス位置を検索します。	528
getParamIOType()	このストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを検索します。	528
getParamName()	ストアード・プロシージャー・パラメーターの名前を検索します。	529
getParamValue()	ストアード・プロシージャー・パラメーターの値を検索します。	530
setParamDataTypeJavaObj()	データ型をこのストアード・プロシージャー・パラメーターの Java Object として設定します。	531
setParamDataTypeJDBC()	データ型をこのストアード・プロシージャー・パラメーターの JDBC データ型として設定します。	531
setParamIndex()	このストアード・プロシージャー・パラメーターのインデックス位置を設定します。	532
setParamIOType()	このストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを設定します。	532
setParamName()	ストアード・プロシージャー・パラメーターの名前を設定します。	533
setParamValue()	ストアード・プロシージャー・パラメーターの値を設定します。	534

UserStoredProcedureParam()

ストアード・プロシージャーのパラメーターに関する引き数情報を保持する、UserStoredProcedureParam の新しいインスタンスを作成します。

構文

```
UserStoredProcedureParam(int paramIndex, String paramType,
   Object paramValue, String ParamIOType, String paramName)
```

パラメーター

paramIndex ストアード・プロシージャーの宣言で、関連するパラメーターのインデックス位置。インデックスの番号付けは、1 から始まります。

paramType 関連するパラメーターの (Java Object としての) データ型。

paramValue ストアード・プロシージャーに送信される引き数値。

ParamIOType 関連するパラメーターのイン/アウト・タイプ。有効なタイプは、次のとおりです。「IN」- パラメーター値は*入力専用* です。「INOUT」- パラメーター値は*入力および出力* です。「OUT」- パラメーター値は*出力専用* です。

paramName Vector 配列を作成する後続のステートメントで使用される、引き数の名前。

戻り値

ストアード・プロシージャーの宣言内の位置 *argIndex* で引き数の引数情報を保持するための、新規 UserStoredProcedureParam オブジェクトを戻します。

例外

DtpConnectionException - パラメーターが無効な場合です。

getParamDataTypeJavaObj()

このストアード・プロシージャー・パラメーターのデータ型を、Integer、Double、または String などの Java Object として検索します。

構文

```
String getParamDataTypeJavaObj()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam パラメーターのデータ型を Java Object として戻します。

例外

なし。

注記

Java Object は、UserStoredProcedureParam オブジェクトに保管されるパラメーター・データ型の 2 つの表記の 1 つです。Java Object データ型を取得するには getParamDataTypeJavaObj() を使用します。Java Object データ型を使用する理由は、以下のとおりです。

- IN (および INOUT) パラメーターの場合、パラメーター値を Java Object として指定する必要があります。したがって、パラメーターのデータ型を Java Object として指定する方が整合性がとれます。
- execStoredProcedure() メソッドは、Vector パラメーター配列のパラメーターを送信します。Vector オブジェクトには、Java Object であるエレメントのみ含めることができます。

参照項目

getParamDataTypeJDBC(), setParamDataTypeJavaObj()

---

getParamDataTypeJDBC()

このストアード・プロシージャー・パラメーターのデータ型を、整数の JDBC データ型として検索します。

構文

```
int getParamDataTypeJDBC()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam パラメーターのデータ型を JDBC データ型として戻します。

例外

なし。

注記

JDBC データ型は、UserStoredProcedureParam オブジェクトに保管されるパラメーター・データ型の 2 つの表記の 1 つです。JDBC データ型は整数値で、以下を含んでいます。

- java.sql.Types.INTEGER
- java.sql.Types.VARCHAR
- java.sql.Types.DOUBLE
- java.sql.Types.DATE

これらのデータ型は、java.sql.Types で定義されます。

推奨: JDBC データ型ではなく、Java Object データ型を使用することを推奨します。ただし、マッピング API は JDBC を内部的に使用するため、getParamDataTypeJDBC() を使用して UserStoredProcedureParam オブジェクトから値を取得できます。

参照項目

getParamDataTypeJavaObj(), setParamDataTypeJDBC()

---

getParamIndex()

このストアード・プロシージャー・パラメーターのインデックス位置を検索します。

構文

```
int getParamIndex()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam パラメーターのインデックス位置を戻します。

例外

なし。

注記

ストアード・プロシージャー・パラメーターのインデックス位置は、ストアード・プロシージャー宣言のパラメーター・リスト内でのインデックスの位置です。最初のパラメーターのインデックス位置は 1 です。インデックス位置は、ストアード・プロシージャーに指定されるリテラル・パラメーターは参照しません。

参照項目

setParamIndex()

---

getParamIOType()

このストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを検索します。

構文

```
String getParamIOType()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam パラメーターのイン/アウト・タイプを戻します。

例外

なし。

注記

イン/アウト・パラメーター・タイプは、ストアード・プロシージャーがパラメーターをどのように使用するかを示します。次のうちの 1 つのストリング表記を使用できます。

- IN パラメーター

  IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、値を戻すためにそのパラメーターを使用しません。getParamIOType() は、イン/アウト・パラメーター・タイプを「IN」として戻します。

- INOUT パラメーター

  INOUT パラメーターは*入力および出力* です。つまり、ストアード・プロシージャーはその値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。getParamIOType() は、イン/アウト・パラメーター・タイプを「INOUT」として戻します。

- OUT パラメーター

  OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーはその値を入力として読み取るのではなく、値を戻すためにそのパラメーターを使用します。getParamIOType() は、イン/アウト・パラメーター・タイプを「OUT」として戻します。

参照項目

setParamIOType()

---

getParamName()

ストアード・プロシージャー・パラメーターの名前を検索します。

構文

```
String getParamName()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam オブジェクトから、パラメーターの名前を戻します。

例外

なし。

注記

パラメーターの名前は通知専用です。エラー・メッセージとデバッグのためにのみ使用されます。パラメーター名は、ストアード・プロシージャー・パラメーターにアクセスする必要はありません。これは、ストアード・プロシージャーには、ストアード・プロシージャー宣言のインデックス位置を使用してアクセスするためです。

参照項目

setParamName()

---

getParamValue()

ストアード・プロシージャー・パラメーターの値を検索します。

構文

```
Object getParamValue()
```

パラメーター

なし。

戻り値

関連する UserStoredProcedureParam パラメーターの値を Java Object として戻します。

例外

なし。

注記

getParamValue() メソッドは、パラメーター値を Java Object (Integer、Double、または String など) として戻します。OUT パラメーターに戻された値が JDBC NULL の場合、getParamValue() は null 定数を戻します。

参照項目

setParamValue()

setParamDataTypeJavaObj()

データ型をこのストアード・プロシージャー・パラメーターの Java Object として設定します。

構文

```
void setParamDataTypeJavaObj(String paramDataType)
```

パラメーター

paramDataType Java Object としての、パラメーターのデータ型。

例外

DtpConnectionException - 入力データ型がサポートされていない場合です。

注記

Java Object は、UserStoredProcedureParam オブジェクトに保管されるパラメーター・データ型の 2 つの表記の 1 つです。データ型を Java Object として設定するには、setParamDataTypeJavaObj() を使用します。Java Object データ型を使用する理由は、以下のとおりです。

- IN (および INOUT) パラメーターの場合、パラメーター値を Java Object として指定する必要があります。したがって、パラメーターのデータ型を Java Object として指定する方が整合性がとれます。
- execStoredProcedure() メソッドは、Vector パラメーター配列のパラメーターを送信します。Vector オブジェクトには、Java Object であるエレメントのみ含めることができます。

参照項目

getParamDataTypeJavaObj(), setParamDataTypeJDBC()

setParamDataTypeJDBC()

データ型をこのストアード・プロシージャー・パラメーターの JDBC データ型として設定します。

構文

```
void setParamDataTypeJDBC(int paramDataType)
```

パラメーター

paramDataType JDBC タイプとしての、パラメーターのデータ型。

例外

DtpConnectionException - 入力データ型がサポートされていない場合です。

注記

すべての UserStoredProcedureParam オブジェクトに、Java Object データ型と JDBC データ型の 2 つの表記があります。Java Object データ型を使用する理由は、以下のとおりです。

- IN (および INOUT) パラメーターの場合、パラメーター値を Java Object として指定する必要があります。したがって、パラメーターのデータ型を Java Object として指定する方が整合性がとれます。
- execStoredProcedure() メソッドは、Vector パラメーター配列のパラメーターを送信します。Vector オブジェクトには、Java Object であるエレメントのみ含めることができます。

参照項目

getParamDataTypeJDBC(), setParamDataTypeJavaObj()

---

setParamIndex()

このストアード・プロシージャー・パラメーターのインデックス位置を設定します。

構文

```
void setParamIndex(int paramIndex)
```

パラメーター

paramIndex　ストアード・プロシージャー・パラメーターのインデックス位置。

注記

ストアード・プロシージャー・パラメーターのインデックス位置は、ストアード・プロシージャー宣言のパラメーター・リスト内でのインデックスの位置です。最初のパラメーターのインデックス位置は 1 です。インデックス位置は、ストアード・プロシージャーに指定されるリテラル・パラメーターは参照しません。

参照項目

getParamIndex()

---

setParamIOType()

このストアード・プロシージャー・パラメーターのイン/アウト・パラメーター・タイプを設定します。

構文

```
void setParamIOType(String paramIOType)
```

パラメーター

paramIOType　ストアード・プロシージャー・パラメーターの入出力タイプ

注記

イン/アウト・パラメーター・タイプは、ストアード・プロシージャーがパラメーターをどのように使用するかを示します。タイプは次のいずれかです。

- IN パラメーター

  IN パラメーターは*入力専用* です。つまり、ストアード・プロシージャーはその値を入力として受け入れますが、値を戻すためにそのパラメーターを使用しません。IN パラメーターの場合、イン/アウト・パラメーター・タイプを 「IN」に設定します。

- INOUT パラメーター

  INOUT パラメーターは*入力および出力* です。つまり、ストアード・プロシージャーはその値を入力として受け入れ、また値を戻すときにもそのパラメーターを使用します。INOUT パラメーターの場合、イン/アウト・パラメーター・タイプを「INOUT」に設定します。

- OUT パラメーター

  OUT パラメーターは*出力専用* です。つまり、ストアード・プロシージャーはその値を入力として読み取るのではなく、値を戻すためにそのパラメーターを使用します。OUT パラメーターの場合、イン/アウト・パラメーター・タイプを「OUT」に設定します。

参照項目

`getParamIOType()`

---

setParamName()

ストアード・プロシージャー・パラメーターの名前を設定します。

構文

```
void setParamName(String paramName)
```

パラメーター

paramName　　ストアード・プロシージャー・パラメーターの名前。

注記

パラメーターの名前は通知専用です。エラー・メッセージとデバッグのためにのみ使用されます。パラメーター名は、ストアード・プロシージャー・パラメーターにアクセスする必要はありません。これは、ストアード・プロシージャーには、ストアード・プロシージャー宣言のインデックス位置を使用してアクセスするためです。

参照項目

`getParamName()`

setParamValue()

ストアード・プロシージャー・パラメーターの値を設定します。

構文

```
void setParamValue(Object paramValue)
```

パラメーター

paramValue ストアード・プロシージャー・パラメーターの値。値は、Java Object (Integer、Double、または String など) でなければなりません。

注記

パラメーター値を Java Object として設定する必要があります。

参照項目

getParamValue()

第 4 部 付録

付録 A. メッセージ・ファイル

各マップには、メッセージ・ファイルを関連付けることができます。メッセージ・ファイル には、そのマップの例外メッセージとロギング・メッセージのテキストが格納されます。メッセージ・ファイル内の各メッセージは、固有 ID によって識別されます。メッセージのテキストには、パラメーター と呼ばれるプレースホルダー変数が入っていることがあります。

マップのメッセージを生成するメソッドには、ユーザーに対して表示するメッセージ・テキストの生成方法が 2 通りあります。メソッド呼び出しのコーディングに、以下のものを組み込むことができます。

- メッセージのテキスト
- 外部メッセージ・ファイルに含まれているメッセージ・テキストの参照

一般に、マップでテキストそのものを生成するより、メッセージ・ファイルを参照する方が望ましいやり方で、保守、管理、および国際化対応に適しています。

この章では、メッセージ・ファイル、メッセージ・ファイルの機能、およびメッセージ・ファイルの設定方法について説明します。内容は、次のとおりです。

『メッセージの位置』	537
541 ページの『マップ・メッセージのフォーマット』	541
541 ページの『メッセージのパラメーター』	541
542 ページの『ファイルの保守』	542
543 ページの『メッセージ・ファイルを使用する操作』	543

メッセージの位置

すべてのメッセージ・ファイルは、IBM WebSphere Business Integration Server Express 製品ディレクトリー内の以下のディレクトリーにあります。

```
DLMs¥messages
```

注: 本書では、ディレクトリー・パスの規則として円記号 (¥) を使用します。UNIX システムの場合、円記号の代わりにスラッシュ (/) を使用します。すべての WebSphere Business Integration Express 製品のパス名は、ご使用のシステムで WebSphere Business Integration Express 製品がインストールされているディレクトリーを基準とした相対パス名です。

マップに関するメッセージの生成に使用できるメッセージ・ファイルとして、以下の 3 種類のファイルがあります。

- マップ固有のメッセージ・ファイル *mapName_locale*txt。ここで、*mapName* はマップ名であり、*locale* はマップの定義が行われたロケールです。

  マップのメッセージは、Map Designer Express の「メッセージ」タブに表示され、マップ定義の一部としてリポジトリーに保管されます。マップをコンパイルすると、Map Designer Express によってメッセージの内容が抽出され、実行時に

使用できるように、メッセージ・ファイルが作成 (または更新) されます。メッセージ・ファイルの名前のフォーマットは、以下のとおりです。

MapName_locale.txt

例: LegacyAddress_to_CwAddress というマップがあり、このマップが作成されたロケールが米国の英語ロケールである場合、Map Designer Express は LegacyAddress_to_CwAddress_en_US.txt という名前のメッセージ・ファイルを作成し、それを ProjectName\Maps\Messages ディレクトリーに保管します。このマップが InterChange Server Express に配置されるときには、DLMs¥messages ディレクトリーに配置されます。

- UserMapMessages.txt メッセージ・ファイル。

  このファイルに、WebSphere Business Integration Express によって定義された「安全な」範囲のメッセージ番号を新しく追加することができます (表 143 を参照)。例えば、Oracle マップに対するメッセージを作成する場合、メッセージに 6101 から 6200 までの番号を割り当てることができます。WebSphere Business Integration Express の汎用メッセージ・ファイル (次で説明する CWMapMessages.txt) にすでに定義済みのメッセージ番号を使用したり、既存のメッセージ・テキストをユーザーが選択したテキストに変更することもできます。UserMapMessaages.txt ファイルは、WebSphere Business Integration Express メッセージ・ファイルよりも先に 検索されるので、このファイルに追加した内容は、WebSphere Business Integration Express メッセージ・ファイル内のメッセージをオーバーライドします。

- WebSphere Business Integration Express の汎用メッセージ・ファイル CWMapMessages.txt (WebSphere Business Integration Express によって提供される)。

  他の 2 つのメッセージ・ファイルのいずれも*参照しない* マップでは、このメッセージ・ファイルを参照する必要があります。表 143 に、WebSphere Business Integration Express によって割り当てられ、汎用メッセージ・ファイルに格納されているメッセージ番号を示します。

  重要: WebSphere Business Integration Express の汎用メッセージ・ファイル CwMapMessages.txt の内容は変更しないでください。汎用メッセージに変更を加えるには、汎用メッセージ・ファイルを UserMapMessage.txt メッセージ・ファイル内にコピーしてから、カスタマイズしてください。

これら 3 つのファイルは、マップ固有のファイルから汎用ファイルまでの用途に使用できます。任意のマップが使用できるメッセージは、WebSphere Business Integration Express 提供の汎用ファイル内にあります。他の 2 つのファイルは、作成するマップの必要に応じてメッセージをカスタマイズするためのオプションとなります。

要確認: InterChange Server Express は、始動時に UserMapMessages.txt ファイルおよび CWMapMessages.txt ファイルをメモリーに読み込みます。UserMapMessages.txt に変更を加えた場合は、その変更内容がマップで有効になるように InterChange Server Express を再始動する必要があります。

表 143. *CwMapMessages.txt* のメッセージ

メッセージ番号	メッセージ・テキスト	メッセージの用途
5000	Mapping - Value of the primary key in the source object is null. Map execution stopped.	ソース・オブジェクトの基本キーが null の場合に使用されます。ソース・オブジェクトの基本キーに基づいた関係メソッドのいずれかが呼び出される前に、ソースの基本キーが null であるかどうかのチェックを必ず実行する必要があります。基本キーが null の場合、エラーが表示され、マップの実行が停止されます。
5001	Mapping - RelationshipRuntimeException. Map execution stopped.	以下のいずれかで `RelationshipRuntimeException` がキャッチされた場合に使用されます。 • 関数ブロック – General/APIs/Identity Relationship/Maintain Simple Identity Relationship – General/APIs/Identity Relationship/Maintain Composite Relationship • マッピング API – `maintainSimpleIdentityRelationship()` – `maintainCompositeRelationship()`
5002	Mapping - CxMissingIDException. Map execution stopped.	以下のいずれかで `CxMissingIDException` がキャッチされた場合に使用されます。 • 関数ブロック – General/APIs/Identity Relationship/Maintain Simple Identity Relationship – General/APIs/Identity Relationship/Maintain Composite Relationship • マッピング API – `maintainSimpleIdentityRelationship()` – `maintainCompositeRelationship()`
5003	Mapping - Data in the {1} attribute is missing.	関数ブロック Foreign Key Lookup (`foreignKeyLookup()`) または Foreign Key Cross-Reference (`foreignKeyXref()`) を使用する前に、ソース属性が null になっていた場合に使用されます。これらの関係メソッドが呼び出される前に、ソース属性が null であるかどうかのチェックを必ず実行する必要があります。基本キーが null の場合、エラーが表示され、マップの実行が停止されます。
5007	Mapping - ForeignKeyLookup() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution stopped.	関数ブロック Foreign Key Lookup (`foreignKeyLookup()`) を使用した後、宛先属性が null になった場合に使用されます。マップの実行を停止する必要があります。
5008	Mapping - ForeignKeyLookup() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution continued.	関数ブロック Foreign Key Lookup (`foreignKeyLookup()`) を使用した後、宛先属性が null になった場合に使用されます。マップの実行を続行する必要があります。
5009	Mapping - ForeignKeyXref() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution stopped.	関数ブロック Foreign Key Cross-Reference (`foreignKeyXref()`) を使用した後、宛先属性が null になった場合に使用されます。マップの実行を停止する必要があります。

マップがメッセージ番号を参照するとき、以下の順にメッセージ・ファイルが検索されます。

1. マップ固有のメッセージ・ファイル*mapName_locale*. txt (*mapName* はマップ名) が検索される。
2. ファイル UserMapMessages.txt が検索される。
3. WebSphere Business Integration Express の汎用メッセージ CWMapMessages.txt が検索される。

表 144 に、CwMapMessages.txt ファイル内の各メッセージが使用される状態を示したコード例を示します。

表 144. CwMapMessages.txt メッセージのコード例

メッセージ番号	コード例
5000	`ObjContract.setVerb(ObjSAP_Contract.getVerb()); if (ObjSAP_Contract.get("ContractId") == null) { logError(5000); throw new MapFailureException( "Data in the primary key is missing"); }`
5001	`try { IdentityRelationship.maintainSimpleIdentityRelationship("Contract", "SAPCntr", ObjSAP_Contract, ObjContract, cwExecCtx); } catch (RelationshipRuntimeException e1) { logError(5001); throw new MapFailureException("RelationshipRuntimeException"); } catch (CxMissingIDException e2) { logError(5002); throw new MapFailureException("CxMissingIDException"); }`
5002	上記のコード例を参照してください。
5003	`if (ObjSAP_Contract.get("CustomerId") == null) { logError(5003, "CustomerId"); throw new MapFailureException("CustomerId is null"); }`
5007	`try { IdentityRelationship.foreignKeyLookup ("Customer", "OracCust", ObjOracle_OrderImport, "customer_id", ObjOrder, "CustomerId", cwExecCtx); } catch (RelationshipRuntimeException e) { logWarning(e.toString()); } if (ObjOracle_OrderImport.get("customer_id") == null) { logError(5007, "customer_id", "CustomerId", "Customer", "OracCust", strInitiator); throw new MapFailureException("foreignKeyLookup() failed."); }`
5008	`try { IdentityRelationship.foreignKeyLookup ("Customer", "OracCust", ObjOracle_OrderImport, "customer_id", ObjOrder, "CustomerId", cwExecCtx); } catch (RelationshipRuntimeException e) { logWarning(e.toString()); } if (ObjOracle_OrderImport.get("customer_id") == null) { logError(5008, "customer_id", "CustomerId", "Customer", "OracCust", strInitiator); }`

表 144. CwMapMessages.txt メッセージのコード例 (続き)

メッセージ番号	コード例
5009	try { IdentityRelationship.foreignKeyXref ("Customer", "OracCust", "CWCust", ObjOracle_OrderImport, "customer_id", ObjOrder, "CustomerId", cwExecCtx); } catch (RelationshipRuntimeException e) { logWarning(e.toString()); } if (ObjOracle_OrderImport.get("customer_id") == null({ logError(5009, "customer_id", "CustomerId", "Customer", "OracCust", strInitiator); throw new MapFailureException("foreignKeyXref() failed."); }

マップ・メッセージのフォーマット

WebSphere Business Integration Express では、メッセージの整合性を保証するため、メッセージ・フォーマットを作成しました。このセクションでは、このフォーマットの以下の内容について説明します。

- 『メッセージ・フォーマット』
- 『メッセージのパラメーター』
- 542 ページの『コメント』

注: マップ固有のメッセージ・ファイルの変更は、Map Designer Express の「メッセージ」タブから行う必要があります。ファイルに直接変更を加えないでください。Map Designer Express は、マップ固有のメッセージ・ファイルに加えられたカスタム変更を、マップに保管されているメッセージで上書きします。ただし、メッセージ・ファイル UserMapMessages.txt と CWMapMessages.txt に関しては、ファイルに直接変更を加えても問題ありません。

メッセージ・フォーマット

各メッセージのフォーマットは次のとおりです。

MessageNum
Message

メッセージ番号 (*MessageNum*) およびメッセージ自体 (*Message*) は、別の行にある必要があります。この場合、各行の終わりには改行が必要です。

例: 図 136 に示すように、マップのメッセージには番号 23 と確認されたメッセージがあり、このメッセージのテキストに {1} と {2} で示された 2 つのプレースホルダー変数が含まれています。

```
23
Customer ID {1} could not be changed: {2}
```

図 136. サンプル・メッセージ

メッセージのパラメーター

マップが特定のメッセージを表示するメソッドを呼び出すとき、メッセージの識別番号と、考えられる追加パラメーターをメソッドに渡します。このメソッドはこの

識別番号を使用して、メッセージ・ファイルから正しいメッセージを探し出し、追加パラメーターの値をメッセージ・テキストのプレースホルダー変数に挿入します。

考えられるすべての状況ごとに別のメッセージを作成する必要はありません。代わりに、パラメーターを使って実行時に変化する値を表します。パラメーターを使用することにより、1 つのメッセージで複数の状況に対処することが可能になり、メッセージ・ファイルの増大を防ぐことができます。

パラメーターは常に、{*number*} のように中括弧で囲まれた番号として表示されます。メッセージに追加したい各パラメーターには、以下のように中括弧で囲んだ番号をメッセージのテキストに挿入します。

```
message text {number} more message text.
```

例: 図 136 のメッセージ 23 の場合を再度考えてみます。マップがこのメッセージを表示または記録する必要がある場合、マップはこのメッセージの識別番号 (23) と 2 つの追加パラメーターを該当するメソッドに渡します。

- パラメーター 1 は顧客 ID 番号 (6701) になります。
- パラメーター 2 は「greater than maximum length」などの説明のための付加的なテキストが入った String 変数になります。

このメソッドは、正しいメッセージを探し出し、メッセージのプレースホルダーのパラメーター値を置換し、以下のメッセージを表示または記録します。

```
Customer ID 6701 could not be changed: greater
than maximum length
```

メッセージ・テキストでは、欠落しているエントリーとその ID をパラメーターとして記述できるので、ハードコーディングされたストリングとして組み込む代わりに、顧客 ID と説明テキストの任意のペアに対して同じメッセージを使用できます。

コメント

メッセージ・ファイルの各コメントの先頭には、ポンド記号 (#) を付けます。

例: コメントは次のようになります。

```
# Message file for the Address business object map.
```

推奨: ファイルの先頭に、一連のコメント行を短縮ヘッダーとして挿入することをお勧めします。このヘッダー・データには、マップの名前や、ファイルの作成者およびファイルの作成日などの情報を組み込みます。

ファイルの保守

ユーザー・サイトでは、マップ・メッセージをフィルター操作して、問題を解決できる人に E メールや E メール・ページャーで通知するためのプロシージャーを管理者が設定する場合があります。そのため、エラー番号とその番号に対応する意味がマップの最初のリリースの後で変更がないようにする必要があります。

推奨: エラー番号に対応するテキストを変更することはできますが、テキストの意味を変更したりエラー番号を割り当て直すことは避けてください。エラー番号に対応する意味を変更する必要がある場合は、必ずその変更を文書化してそのマップのユーザーに通知してください。

メッセージ・ファイルを使用する操作

メッセージ・ファイルは、幾つかのタイプの操作で使用されるメッセージのテキストを格納しています。 25 ページの表 9 に、メッセージ・ファイルを使用する操作のタイプと、これらの操作を実行する BaseDLM クラスのメソッドを示します。

表 145. メッセージを生成する操作

操作	関数ブロック	メソッド
例外の生成	General/APIs/Maps/Exception/ Raise Map Exception	raiseException()
ロギング	General/Logging and Tracing/Log Information ID	logInfo()
	General/Logging and Tracing/Log error ID	logError()
	General/Logging and Tracing/Log warning ID	logWarning()
トレース	General/Logging and Tracing/Trace/Trace on Level	trace()

このセクションでは、メッセージを生成する操作のうち、マップの実行に影響を与えるものについて説明します。

例外の生成

raiseException() メソッドには幾つかの形式があります。以下は、一般的に使用される構文の例です。

```
raiseException(String exceptionType,
   int messageNum, String param[,...])
```

この構文では、1 つから 3 つの *param* String パラメーターを指定できます。したがって、1 つの raiseException() 呼び出しに、最大で 5 つまでのコンマで区切ったパラメーターを指定できます。

この例では、メッセージ番号 23 を使用して新しい例外を発生させ、顧客 ID およびストリングが 2 つのパラメーターとしてメッセージに渡されます。

```
raiseException(AttributeException, 23,
   fromCustomer.getString("CustomerID"),
   "greater than maximum length");
```

図 136 に、メッセージ 23 のテキストを、メッセージ・ファイルに格納されている形で示してあります。

ロギング・メッセージ

マップは、管理者に関係があると考えられる事態が発生するたびにメッセージを記録できます。マップは、メッセージを記録する場合、BaseDLM クラスの

logInfo()、logWarning()、および logError() の各メソッドを使用します。各メソッドには、さまざまなメッセージの重大度レベルが関連付けられています。

重大度レベル

メッセージを記録するためには、そのメッセージの重大度に対応するメソッドを呼び出す必要があります。表 146 に、重大度レベルとそれぞれのレベルに対応するメソッドを示します。

表 146. メッセージ・レベル

重大度レベル	メソッド	説明
情報	logInfo()	情報のみ。ユーザーは何もする必要はありません。
警告	logWarning()	問題に関する情報を示します。このレベルは、ユーザーが解決する必要がある問題には使用しないでください。
エラー	logError()	ユーザーが調べる必要がある重要な問題を示します。

メッセージ・ファイルの使用

すべてのマップには、1 つ以上のメッセージ・ファイルが関連付けられている必要があります。マップでカスタム・メッセージが使用されない場合、そのメッセージはシステム・マップ・メッセージ・ファイル CWMapMessages.txt にあるメッセージです。マップでカスタマイズしたメッセージが使用される場合、そのマップはマップ固有のメッセージ・ファイル (Map Designer Express の「メッセージ」タブで入力されたメッセージから生成される) を持っています。詳細については、537 ページの『メッセージの位置』を参照してください。

マップがエラーを記録する場合、エラー・メッセージのテキストにはマップのメッセージ・ファイルが使用されます。

例: 次の例では、マップのメッセージ・ファイルにテキストが格納されるエラー・メッセージを記録します。エラー・メッセージ 10 のテキストは、以下のようにメッセージ・ファイルに表示されます。

```
10
Credit report error for {1}, {2}.
```

メッセージを記録するコードは、以下のようになります。

```
logError(10, customer.get("LName"), customer.get("FName");
```

logError() メソッドが実行されると、メッセージ 10 のテキストがログ・ファイルに書き込まれます。この場合、パラメーター 1 および 2 は顧客のラストネームおよびファーストネームに置換されます。

例: 顧客名が John Davidson である場合、記録されるメッセージは、以下のようになります。

```
Credit report error for Davidson, John.
```

理想的なメッセージ・ロギングの原則

メッセージを作成する場合、管理者がロギング機能を使用する方法は大変重要です。

重大度レベルの割り当て: メッセージに対するエラー・レベルの割り当ては、厳密に行う必要があります。 IBM システムの E メール通知機能では、エラー・メッセージや致命エラー・メッセージの生成が検出されたとき、指定された人 (通常は管理者) にメッセージを送信します。管理者は、この IBM システム E メール通知機能を使用しますが、さらに、エラーの発生時にページを送信する E メール・ページャーにリンクすることもできます。エラー・レベルをメッセージに正確に割り当てることにより、重大メッセージの数を減らすことができます。

メッセージの変更: 例えば、テキストをわかりやすくしたり、拡張するために、メッセージのテキストをいつでも変更できます。ただし、ある一定のタイプのエラーにメッセージ番号を割り当てた後は、その番号の変更をしないということが重要です。多くの管理者はスクリプトを使用してログ・メッセージをフィルターに掛けています。ただし、これらのスクリプトはメッセージ番号に依存しています。このため、メッセージ・ファイルの番号の意味を変更しないことが重要です。番号が変更されると、ユーザーがメッセージを喪失したり、間違ったメッセージを受信することがあります。

情報メッセージを使用するタイミング: `logInfo()` メソッドを使用して、ユーザー独自のデバッグ用一時メッセージを作成できます。ただし、開発が完了したら、これらのデバッグ・メソッド呼び出しは除去するようにしてください。

コラボレーションの通常操作を文書化する場合は、`logInfo()` メソッドを使用しないようにしてください。このメソッドを使用すると、管理者のログ・ファイルが関係のないメッセージでいっぱいになってしまいます。代わりに、`trace()` メソッドを使用して、管理者用のデバッグの詳細情報を記録してください。

トレース・メッセージの追加

トレース・メッセージをマップに追加することにより、マップ・インスタンスの実行時にそのアクションの詳細な説明が生成されるようにすることができます。トレース・メッセージは、ユーザー独自のデバッグや、管理者によるオンサイトのトラブルシューティングに役立ちます。

トレース・メッセージは、そのトレース・メッセージに含まれるログ・メッセージとは異なり、デフォルトでは削除可能となっています。ただし、そのログ・メッセージを削除することはできません。通常、トレース・メッセージの方がより詳細で、一定の状況下でのみ表示されるようになっています。例えば、あるユーザーがマップのトレース・レベルを意図的にゼロより大きい番号に構成した場合などがあります。トレース・メッセージとログ・メッセージは別のファイルに送信できます。

トレース・メッセージをマップに追加して、そのマップに固有の操作を報告できます。マップがトレース・ファイルに書き込むことが可能な情報のタイプを以下に示します。

- マップが特定の変換ステップを開始または終了した時点でのビジネス・オブジェクトのキー値
- 実行経路で特定の分岐が採用する決定内容

トレース・レベルの割り当て

各トレース・メッセージには、1 から 5 までのトレース・レベルを関連付ける必要があります。トレース・レベルは通常、詳細の度合いと相互に関係があります。一般にレベル 1 のメッセージより、レベル 2 のメッセージの方がより詳細で、レベル 3 のメッセージの方がレベル 2 より詳細となっています。このため、レベル 1 のトレースをオンにした場合、レベル 5 のメッセージより詳細ではないメッセージが表示されます。ただし、レベルは必要なレベルに設定できます。

推奨: レベルの割り当てに関する提案事項を以下に示します。

- すべてのトレース・メッセージに対して同じレベルを割り当てることができます。
- 詳細の度合いに応じて、トレース・レベルを割り当てることができます。
- 関係するビジネス・オブジェクトに応じて、メッセージ・レベルを割り当てることができます。レベル 1 のトレース・メッセージを特定のビジネス・オブジェクトに関連させ、レベル 2 のトレース・メッセージを別のビジネス・オブジェクトに関連させるという指定が可能です。

特定のレベルでのトレースをオンにすると、特定のレベルに関するメッセージと、これ以下のレベルに関するメッセージが表示されます。例えば、レベル 2 のトレースの場合は、レベル 2 とレベル 1 に関するメッセージが表示されます。

ヒント: トレース・レベルは、文書に記録して、ユーザーがトレースを行うときに使用すべきレベルがわかるようにしてください。

トレース・メッセージの生成

例: 以下は、メッセージと、メッセージを生成するメソッドの呼び出しの例です。メッセージは、以下のようにメッセージ・ファイルに表示されます。

```
20
Begin transformation on {1} attribute: value = {2}
```

このメソッドの呼び出しは、属性 LName の値を取得し、この値を使用してメッセージのパラメーターを置換します。コードは、以下のようにマップに表示され、このメッセージは、ユーザーがトレースをレベル 3 に設定したときに表示されます。

```
trace(3, 20, "LName", customer.get("LName"));
```

トレース・レベルの設定

図 137 に、Map Designer Express の「マップ・プロパティー」ダイアログの「一般」タブを示します。 (「マップ・プロパティー」ダイアログの表示方法については、64 ページの『マップ・プロパティー情報の指定』を参照してください。) このダイアログで、トレース・メッセージのトレース・レベルを設定することができます。

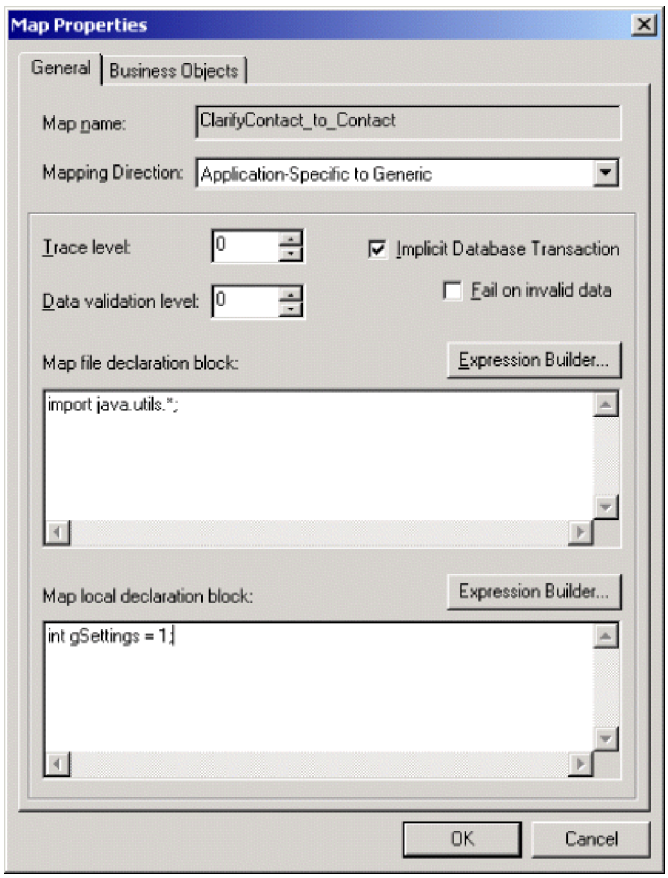

図 137. マップのトレース・レベル

マップの開発者が、546 ページの『トレース・レベルの割り当て』の説明に従って、マップ生成のトレース機能を要求できるレベルを作成します。

注: アクティブなマップのトレース・レベルを変更する場合は、マップを停止してから再始動した後、新しいトレース・レベルを有効にする必要があります。System Manager の「コンポーネント」メニューを使用して、マップを停止して始動します。

Map Designer Express の「マップ・プロパティー」ダイアログでトレース・レベルを設定すると、このマップのマップ定義に従って、マップのすべての インスタンスを対象に、トレース・レベルを設定することになります。マップのすべてのインスタンスにトレース・レベルを設定する場合、System Manager の「マップ・プロパティー」ウィンドウから行うこともできます。

付録 B. 属性プロパティー

表 147 に、ビジネス・オブジェクト定義の属性のプロパティーを示します。

表 147. 属性プロパティー

プロパティー	説明
Name	属性に含まれるデータの型を表す名前。名前は 80 文字までの英数字と下線です。スペースやその他の句読点を含めることはできません。
Type	属性のデータ型。基本型は、String、Boolean、Double、Float、Integer、および Date です。属性が子ビジネス・オブジェクトを参照する場合は、子ビジネス・オブジェクト定義の名前を指定します。子ビジネス・オブジェクトを参照する属性は、複合属性と呼ばれます。
IsKey	boolean 値 (`true` または `false`)。これがキー属性かどうかを指定します。キー属性は、定義から作成されたビジネス・オブジェクトを一意に識別します。各ビジネス・オブジェクト定義には、少なくとも 1 つのキー属性があります。
IsForeignKey	boolean 値 (`true` または `false`)。これが外部キー属性かどうかを指定します。
MaxLength	属性に含まれる最大バイト数を表す整数。制限がないことを指定するには、ゼロ (`0`) を入力します。
AppSpecificInfo	表のフィールド名や、属性に対応するフォームなど、特定のアプリケーションの属性に関する情報を提供するストリング。コネクターは、オブジェクトを処理するときにこの情報を使用します。
DefaultValue	ランタイム値がない場合にこの属性に割り当てられる値。
IsRequired	boolean 値 (`true` または `false`)。この属性の値がビジネス・オブジェクトの作成に必要かどうかを指定します。
ContainedObjectVersion	子ビジネス・オブジェクト定義のバージョン番号。IBM WebSphere System Manager では、この値は Type Version 名の下に表示されます。
Relationship	親ビジネス・オブジェクトと子ビジネス・オブジェクト間の関係。現在のリリースでは、有効な関係は Containment のみです。
Cardinality	この属性が参照する子ビジネス・オブジェクトの数。属性が 1 つの子ビジネス・オブジェクトのみを参照する場合、値は 1 になります。属性が複数の子ビジネス・オブジェクトを参照できる場合、値はリテラル `n` になります。

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。

IBM
IBM ロゴ
AIX
CrossWorlds
DB2
DB2 Universal Database
Lotus
Lotus Domino
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

MMX および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

System Manager には、Eclipse Project (http://www.eclipse.org/) により開発されたソフトウェアが含まれています。

WebSphere Business Integration Server Express V4.3 および WebSphere Business Integration Server Express Plus V4.3

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アウトバウンド・マップ 4, 5
  外部キー参照 479, 481
  カスタマイズ例 313
  参照関係 286, 287, 521
  テスト 105, 108
  マップ文書の 69
アクセス・クライアント 94, 209, 290
「値を設定」変換 19, 41, 42, 59, 62, 90, 114
宛先ビジネス・オブジェクト 3, 6, 15, 202
  関係 245
  実行順序 18, 62, 85, 90
  動詞 40, 312
  動詞の設定 40
  「ビジネス・オブジェクト」ウィンドウ 40
  表示 11, 20, 29, 67
  変数 185
  マップへの追加 36, 38, 39
アプリケーション固有のビジネス・オブジェクト 3
暗黙的なトランザクション・ブラケット 237
  接続の解放 241
  デフォルト 237
  トランザクションのスコープ 238
一時変数 187
一致関係 247, 251
  関係インスタンス ID 254
  クラス 475
  子動詞を維持する 315
  子ビジネス・オブジェクト 269
  子ビジネス・オブジェクトの削除 477, 489
  子ビジネス・オブジェクトの追加 475, 489
  参加者の作成 503
  種類 247, 267
  静的 278
  静的参照 312
  定義 267, 270, 287, 299, 300
  定義済み 246, 247, 267
  テスト 104
「移動」変換 19, 25, 41, 43, 59, 62, 90
イベントにより起動されたフロー 208, 209
インバウンド・マップ 4, 5
  外部キー参照 321, 479, 481
  カスタマイズ例 312
  参照関係 286, 520
  テスト 105, 106, 108
インバウンド・マップ (続き)
  マップ文書の 69
エラー
  コンパイル 89, 92
  ランタイム 109
エラー・メッセージ 350, 544
親子関係 309
  子インスタンスの削除 477, 489
  子インスタンスの追加 475, 489
  定義 269
  定義済み 309
オリジナル要求ビジネス・オブジェクト 210, 293, 295, 298, 320, 322, 496

[カ行]

階層ビジネス・オブジェクト
  最上位の比較 363
  すべてを比較 362
  トラバース 359
外部キー 318, 478, 480, 549
外部キー参照 59, 318
カスタム変換 19, 25, 41, 54, 55, 59, 114, 190, 281
環境変数
  CLASSPATH 178, 180
  JCLASSES 179
  PATH 12
関係 251, 255
  概要 245, 258
  カスタム 324
  コードのインプリメント 281
  最適化 277
  始動 267
  処理 281
  静的 277, 278
  静的参照 172
  タイプ 246, 273
  定義済み 245
  停止 267
  テスト 103
  動的 277
  非一致 246
  変換 54, 59, 281
  命名規則 252, 266
  例外 203
関係インスタンス 252, 255
  位置 253
  インスタンス ID の検索 518
  クラス 8, 253, 475, 509
  子オブジェクトの削除 477
  作成 324, 512

関係インスタンス *(続き)*
参加者の検索 520
参加者の更新 521, 522
参加者の削除 325, 516, 517
参加者の作成 325, 502
参加者の追加 510
参加者の非アクティブ化 325, 513, 514
定義済み 252
ランタイム・データ 271
関係インスタンス ID 254
一致関係 254
参加者インスタンス 504, 506
参加者の検索 518
参加者の更新 523
参加者の削除 517
参加者の非アクティブ化 515
参照関係 254
重複 204
次の検索 518
定義済み 254
関係開発 257
関係定義 251, 252
アンロード 340
位置 251
一致 287, 299, 300
親子 309
拡張設定 268, 273
コピー 271, 272
削除 277
作成 266
参照 270, 283
従属オブジェクト 342
定義済み 8, 251, 259
名前 505, 507
名前変更 272
表示 261
変更 267
保管 267
命名規則 252, 266
リスト 262
ロード 340
ID 267, 270
関係表 252, 253
位置 252, 253, 273, 275, 276, 277
一致関係 288
インデックス・サイズ 301
外部 318, 479, 481
外部キー・ルックアップ 478, 480
キャッシング 277
作成 269, 271, 275
参加者 514, 515
参照関係 171, 270, 283
参照の実行 312
照会 329
定義済み 253
内容 255
関係表 *(続き)*
名前 253, 274, 283
表スキーマ 271, 273, 275
複合一致関係 301
変更 210, 331, 480
MaxLength 属性 301
関係リポジトリー・ファイル 340
関数ブロック 123, 126
移動 160, 165
カスタム JAR ライブラリーの追加 174
関係のインプリメントに使用 281
使用例 157, 164, 171
接続リンクの追加 160, 165
ドラッグ・アンド・ドロップ 159, 164
General/APIs/Business Object 128
General/APIs/Business Object Array 133
General/APIs/Business Object/Array 132
General/APIs/Business Object/Constants 132
General/APIs/Database Connection 134
General/APIs/Identity Relationship 136
General/APIs/Maps 138
General/APIs/Maps/Constants 138
General/APIs/Maps/Exception 139
General/APIs/Participant 140
General/APIs/Participant/Array 142
General/APIs/Participant/Constants 142
General/APIs/Relationship 143
General/Date 145
General/Date/Formats 147
General/Logging and Tracing 147
General/Logging and Tracing/Log Error 148
General/Logging and Tracing/Log Information 148
General/Logging and Tracing/Log Warning 149
General/Logging and Tracing/Trace 150
General/Mapping 150
General/Math 151
General/Properties 153
General/Relationship 153
General/String 153
General/Utilities 156
General/Utilities/Vector 157
JAR ライブラリーのプロパティーのカスタマイズ 176
Web サービス 180
関数ブロックを使用した関係のインプリメント 281
一致関係 299
複合一致関係 301
Foreign Key Cross-Reference 319, 323
Foreign Key Lookup 318, 323
Maintain Child Verb 299, 303, 305, 315, 317
Update My Children 306, 311
キー属性 247, 549
一致関係 268
外部 318, 478, 480, 549
単一 247, 268, 488
複合 249, 268, 300, 486

キー属性値
検査 368
ストリングとして検索 370
設定 373
比較 361
キーボード・ショートカット 32
クイック表示モード (Activity Editor) 117
グラフィックス・ツールバー (Activity Editor) 122
「ズームアウト」 122
「ズームイン」 122
「進む」 122
「ホーム」 122
「戻る」 122
「1 レベル上へ」 122
グラフィック表示 (Activity Editor) 114, 115
クイック表示モード 117
「コンテンツ」ウィンドウ 117
デザイン・モード 115
「プロパティー」ウィンドウ 117
「ライブラリー」ウィンドウ 116
警告メッセージ 350, 544
「結合」変換 19, 25, 41, 44, 59, 62, 90, 114
検索
ストリングとしてのビジネス・オブジェクト配列の値 392
ストリングとしてのビジネス・オブジェクト・キー属性値 370
配列からのビジネス・オブジェクトの 381
ビジネス・オブジェクト属性 364
ビジネス・オブジェクト動詞 367
ビジネス・オブジェクト配列のエレメントの数 391
ビジネス・オブジェクト配列の最後のインデックス 383
ビジネス・オブジェクト配列の最小値 386, 387, 388
ビジネス・オブジェクト配列の最大値 383, 384, 385
ビジネス・オブジェクト配列の内容 382
ビジネス・オブジェクト・タイプ 366
マップ名 347
検索、リンクされていない属性 79
検索および置換、テキストの 81
コール・トリガー・フロー 209
子インスタンスを管理するための関数ブロック 309
コネクター
名前の検索 493
名前の設定 496
マッピング要求の開始 94, 209, 290
コピー
関係定義 271, 272
参加者定義 271, 272
属性 43, 58
ビジネス・オブジェクト 360
子ビジネス・オブジェクト
アドレッシング 187
一致関係 269
親子関係からの除去 477, 489
親子関係への追加 475, 489
カーディナリティー 269, 549
関係に応じたカスタマイズ 304
子ビジネス・オブジェクト *(続き)*
関係に応じたカスタマイズの例 304
サブマップ 49, 50
属性のコメント 59
単一カーディナリティー 187, 211, 212
テスト 97
動詞 314
動詞の設定 483
バージョン番号 549
複数カーディナリティー 49, 187, 212
マッピング 211
「コンテキスト」メニュー (Relationship Designer Express)
アクセス 265
「インデックスを変更」 270
コンテキスト・メニュー (宛先データ、属性)
「ブレークポイントをクリア」 100
「ブレークポイントを設定」 98
コンテキスト・メニュー (宛先データ、メイン・オブジェクト)
「縮小」 98
「保管」 102
コンテキスト・メニュー (ソース・データ、子オブジェクト)
「インスタンスを削除」 97
「インスタンスを追加」 96, 97
「すべてのインスタンスを削除」 97
コンテキスト・メニュー (ソース・データ、メイン・オブジェクト)
「保管」 96
「リセット」 96
「ロード」 98
コンテキスト・メニュー (ビジネス・オブジェクト・ウィンドウ)
「削除」 40
「プロパティー」 186
コンテキスト・メニュー (ビジネス・オブジェクト・ブラウザー)
「コピー」 39
「すべて最新表示」 20
コンテキスト・メニュー (ビジネス・オブジェクト・ペイン)
「ビジネス・オブジェクトを削除」 83
「ビジネス・オブジェクトを追加」 38
コンテキスト・メニュー (変換)
「新規ウィンドウで開く」 30
「ソースを表示」 30
「開く」 30
コンテキスト・メニュー (マップ・ワークスペース)
「削除」 83
「出力オブジェクトとして貼り付け」 39
「入力オブジェクトとして貼り付け」 39
「ビジネス・オブジェクトを追加」 38
「マップ・プロパティー」 64
コンテキスト・メニュー (Activity Editor)
アクセス 121
「記述を追加」 121
「行に移動」 90
「コメントを追加」 121
式ビルダー 199, 220

コンテキスト・メニュー (Activity Editor) *(続き)*
「新規定数」 121
「対応しない区切り文字を検査」 89
「予定を追加」 121
「ラベルを追加」 121
「Add To My Collection」 121
コンテキスト・メニュー (Map Designer Express)、アクセス 30

[サ行]

サーバー・コンポーネント管理ビュー
関係プロパティーの更新 279
マップ・プロパティーの更新 66, 203, 207
サブマップ 48, 53, 216, 223
一致関係 300, 308
キー・マップ 25
検証 62, 90
コードへのアクセス 114
子ビジネス・オブジェクト 49, 50
コンパイル 50, 91, 213, 217, 222
作成 50, 213, 217, 221
式ビルダー 219
実行コンテキスト 207
使用 48
条件 52, 216
属性のコメント 59
多対 1 220
定義済み 48
複数カーディナリティー 212
変換コード 19
命名規則 50
呼び出し 50, 213, 216, 217, 219, 222, 461
「サブマップ」変換 41
参加者 255, 257
定義済み 245
命名規則 255, 266
参加者インスタンス 256
関係インスタンス ID 257, 501, 504, 506, 518
関係インスタンスからの検索 520
関係インスタンスへの追加 510
関係定義 257, 501, 505, 507
クラス 257, 501
更新 521, 522
コンストラクター 501
削除 325, 516, 517
作成 325, 501, 512
参加者定義 257, 501, 504, 506
データ 257, 274, 501, 503, 505
定義済み 256, 501
内容 257
非アクティブ化 513, 514
ID 256
参加者インスタンス ID 256
参加者タイプ 256, 266
データ 246, 256, 283
参加者タイプ *(続き)*
ビジネス・オブジェクト 256, 266, 268, 287, 300
Data 266, 270
「参加者タイプ」ウィンドウ 262, 263, 266, 270
参加者定義 255
位置 255
拡張設定 269, 274
コピー 271, 272
作成 266
定義済み 255
名前 504, 506
名前変更 272
命名規則 255, 266
参照関係 246, 282, 287
関係インスタンス ID 254
コード 285, 520, 521
参加者タイプ 256, 270, 283
参加者の作成 503
静的 172, 278
定義 270, 283
定義済み 246, 270, 282
テスト 107
例 246, 283
式ビルダー 199, 219
自動マッピング
設定 72
属性 72
同義語の作成 78
同義語の使用、例 79
ビジネス・オブジェクト 72
プレフィックスまたはサフィックスの追加 72
マップの作成 72
マップの反転 76
例 74
使用すべきでないメソッド
BusObj クラス 378, 499
DtpConnection クラス 328, 413
Relationship クラス 523
UserStoredProcedureParam クラス 525
情報メッセージ 350, 544, 545
「新規定数」 121, 125, 167, 168
数値、切り捨て 473
ステータス・バー (Activity Editor) 123
ストアード・プロシージャー
関係インスタンス 271, 274
実行 231, 333, 399, 400, 401, 416
照会結果 232, 403, 405
ストアード・プロシージャー・パラメーター 234
値 234, 335, 411, 530, 534
インデックス位置 335, 528, 532
イン/アウト・パラメーター・タイプ 234, 335, 410, 528, 532
オブジェクトの作成 234, 335, 409, 526
名前 335, 529, 533
Java Object タイプ 335, 526, 531
Java オブジェクトから JDBC へのマッピング 236, 338

ストアード・プロシージャー・パラメーター *(続き)*
JDBC から Java オブジェクトへのマッピング 338
JDBC データ型 335, 527, 531
ストリング関数ブロック 153
使用例 159
「置換」 154
Append Text 153
If 153
Is Empty 154
Is NULL 154
Left Fill 154
Left String 154
Lower Case 154
Object To String 154
Repeat 154
Right Fill 154
Right String 155
Substring by position 155
Substring by value 155
Text Equal 155
Text Equal Ignore Case 155
Text Length 155
Trim Left 155
Trim Right 155
Trim Text 155
Upper Case 156, 159
ストリングの分割
オブジェクトに解析されたトークンの処理 466
解析されたストリングの作成 463
現在の位置番号のリセット 468
指定した位置のエレメントの取得 464
ストリングからの最後のエレメントの取得 466
ストリングからの最初のエレメントの取得 465
ストリングからの次のエレメントの取得 467
ストリングからの前のエレメントの取得 468
ストリングのエレメント数の取得 465
ストリング変換 199
大文字テキストへの変換 199
ストリング処理 200
ブランクの検査 202
null の検査 202
静的関係 277, 278
静的参照 312
静的参照関係 172
接続
アクティブかどうかを判別 241, 404
オープン 224, 329
解放 241, 405
取得 223, 345
トランザクション・プログラミング・モデル 237, 238, 346
接続プール 223, 241, 346, 405
接続リンク、関数ブロックに追加 160, 166
設定
子ビジネス・オブジェクト属性の動詞 375
配列内のビジネス・オブジェクトの値 390
ビジネス・オブジェクト属性 371, 375
設定 *(続き)*
ビジネス・オブジェクト属性のデフォルト値 373
ビジネス・オブジェクト動詞 374
ビジネス・オブジェクトのキー属性の値 373
ビジネス・オブジェクトの内容 372
「設定」ダイアログ (Map Designer Express) 29
「一般」タブ 18, 19, 23, 56, 61, 85, 86, 92, 185
「キー・マップ」タブ 25, 43, 45, 47, 51, 54
「検証」タブ 24
「自動マッピング」タブ 25, 74
ゼロ長ストリング 367
ソース・ビジネス・オブジェクト 3, 6, 202
テスト 95
「ビジネス・オブジェクト」ウィンドウ 39
表示 11, 20, 29, 67
変数 185
マップへの追加 35, 38
「相互参照」変換 19, 25, 41, 53, 90, 114
および ACCESS_REQUEST 呼び出しコンテキスト 290
および ACCESS_RESPONSE 呼び出しコンテキスト 298
および SERVICE_CALL_ RESPONSE 呼び出しコンテキスト 294
およびEVENT_DELIVERY 呼び出しコンテキスト 290
およびSERVICE_CALL_ FAILURE 呼び出しコンテキスト 297
およびSERVICE_CALL_ REQUEST 呼び出しコンテキスト 292
関係に応じた定義 288
検証 62
呼び出しコンテキストでの振る舞い 290
属性
宛先 6, 19
アプリケーション固有の情報 549
依存関係 85
拡張設定 275
関係 54, 59, 208, 281, 282
キーの検査 368
結合 44
検索 41, 58, 79
検証 62, 90
コメント 19, 40, 59, 68, 82
最大長 549
指定 359
自動マッピング 72
自動マッピングの同義語の作成 78
ソース 18
データ型 18, 19, 40, 377, 549
名前 18, 19, 40, 549
必要 370, 549
プロパティー 549, 551
分割 46
変換でのアドレッシング 184
リンクされていない 21, 58, 70, 79
列名 275
属性値
検索 364

属性値 *(続き)*
検証 205
合計 391
コピー 43, 58
最小値の検索 386, 387, 388
最大値の検索 383, 384, 385
ストリングとして検索 376
設定 371, 375
ゼロ長ストリング 367
データ型の検証 377
デフォルト 41, 96, 373, 549
デフォルト値の設定 373
ブランク 367
null 326, 368

[タ行]

「ダイアグラム」タブ (Map Designer Express) 20
一時ビジネス・オブジェクト 189
カスタム変換 54
キー・マップ 25
サブマップの呼び出し 51
属性値の設定 42
属性の移動 43
属性の結合 44
属性の表示 41
属性の分割 46
デフォルトの表示 26
「ビジネス・オブジェクト」ウィンドウ 20, 29, 39, 185
ビジネス・オブジェクトの追加 38
ビジネス・オブジェクト変数 185
ビジネス・オブジェクト・ブラウザー 20, 26, 29
変換の削除 83
マップ・ワークスペース 20, 189
単一マップ・マップ・テーブル 68
単純一致関係 247, 248, 267, 287
親マップ 299
子レベル 299
サブマップ 300
参加者タイプ 287
「相互参照」変換の定義 288
定義 268, 269, 287, 299
変換規則の定義 299
メイン・マップ 299
例 247
maintainChildVerb() 299, 317
maintainSimpleIdentityRelationship() 288, 487
置換、テキストの 81
「ツール」メニュー (Activity Editor) 121
「コードを編集」 121
式ビルダー 121
「対応しない区切り文字を検査」 121
「変換」 121
「ツール」メニュー (Map Designer Express) 30
自動マッピング 30
マップの反転 30
「ツール」メニュー (Map Designer Express) *(続き)*
Business Object Designer Express 30
Map Designer Express 30
Process Designer Express 30
Relationship Designer Express 30
「ツール」メニュー (Relationship Designer Express) 265
Business Object Designer Express 265
Map Designer Express 265
Process Designer Express 265
「Relationship Manager」 265
データ型
属性 549
判別 421
変換が可能かどうかを判別 423
データ形式変更パッケージ 10
データ妥当性検査 202, 205
データベース
最後の書き込みによって影響された行 402
照会 225, 229, 402, 404
照会の実行 224, 398, 399, 401
接続 223, 241, 345
データの処理 225
変更 228, 229
データ変換 43, 421
クラス 421
有効な変換 424
Boolean オブジェクト 425
boolean データ型 427
Double オブジェクト 425
double データ型 428
Float オブジェクト 426
float データ型 429
int データ型 429
Integer オブジェクト 427
Java.lang メソッド 421
String オブジェクト 430
「テーブル」タブ (Map Designer Express) 18, 20
一時ビジネス・オブジェクト 189
カスタム変換 54
サブマップの呼び出し 51
実行順序の指定 86
出力ウィンドウ 17
属性値の設定 42
属性の移動 43
属性の結合 44
属性の分割 46
属性変換テーブル 18, 82
デフォルトの表示 26
「ビジネス・オブジェクト」ペイン 20, 26, 29, 38, 83, 189
ビジネス・オブジェクトの追加 38
ビジネス・オブジェクト変数 185
「ビジネス・オブジェクトを削除」 83
変換の削除 82
テキストを検索 79
デザイン・モード (Activity Editor) 115, 116

テスト実行　94
　一時停止　98, 101
　結果の表示　102
　始動　102
　準備　95
　初回　95
　テスト・データの作成　95
　ブレークポイント　98, 101
　2 回目以降　97
「デバッグ」メニュー (Map Designer Express)　29
　「拡張」　29
　「終了」　29, 103
　「ステップオーバー」　29, 102
　「すべてのブレークポイントをクリア」　30, 100
　「接続」　29, 103
　「続行」　29, 101
　「テスト実行」　29, 101
　「テスト実行を停止」　29
　ブレークポイント　99
　「ブレークポイント」　29
　「ブレークポイントの切り替え」　29, 98
デフォルト属性値　41, 373, 549
同義語の作成
　自動マッピングの　78
　編集　78
動詞
　検索　367
　設定　40, 59, 213, 215, 221, 312, 374, 483
　定義済み　40
　テスト実行　96, 97
　動詞ベース・ロジック　192
動詞ベース・ロジック　192
動的関係　277
トランザクション
　暗黙的　237
　開始　238, 239, 332, 395, 413
　管理　228, 237
　コミット　238, 239, 241, 332, 339, 396, 414
　進行中かどうかを判別　240, 403, 418
　スコープ　238
　定義済み　237, 332
　明示的　237
　ロールバック　239, 332, 406, 419
トレース　545, 547
　コード例　546
　メッセージの生成　546
　レベル　546
トレース・メッセージ　355, 543, 545, 547
　生成　546
　追加　545
　トレース・レベルの設定　546
　へのトレース・レベルの割り当て　546
トレース・レベル　350, 546

［ナ行］

内容ベースのロジック　190
名前属性プロパティー　549

［ハ行］

パッケージ
　データ形式変更　10
　マップ・ユーティリティー　177, 178
　Java パッケージをインポート　174
　java.lang　421
　java.util　226, 330, 463
反転マップ
　区切り文字の指定　76
　自動作成　76
　変換規則　76
　例　76
非一致関係　246
比較
　キー属性値　361
　ビジネス・オブジェクト属性値　362, 363
　ビジネス・オブジェクト配列　382
ビジネス・オブジェクト
　値の設定　390
　一時　187
　インスタンス名　39, 184
　階層のトラバース　359
　キー値の設定　373
　キー属性　368
　キー属性値の検索　370
　キー属性値の比較　361
　コピー　360
　削除　16, 83, 261
　自動マッピング　72
　属性値の検索　364, 376
　属性値の設定　371, 372, 375
　属性値の比較　362, 363
　属性のデータ型の検証　377
　追加　16, 261
　動詞の検索　367
　配列内でのスワッピング　392
　配列に追加　380
　汎用　3, 196, 210
　ビジネス・オブジェクト定義　366
　ビジネス・オブジェクト配列から検索　381
　ビジネス・オブジェクト配列から除去　389, 390
　ビジネス・オブジェクト配列内の数　391
　必須属性　370
　複製　360
　プロパティー　186
　変換でのアドレッシング　184
　変数　184
　リストの最新表示　20
　null 属性　368
ビジネス・オブジェクト定義、名前の検索　366

ビジネス・オブジェクト配列
インデックス 90, 97, 185
エレメントの位置を逆にする 392
エレメントの除去 389, 390
エレメントの設定 390
最後のインデックスの検索 383
最小属性値の検索 386, 387, 388
サイズの検索 391
最大属性値の検索 383, 384, 385
ストリングとして値を検索 392
すべてのエレメントを除去 389
属性値を合計 391
内容の検索 382
ビジネス・オブジェクトを検索 381
ビジネス・オブジェクトを追加 380
複製 381
別のビジネス・オブジェクトと比較 382
日付の形式設定
形式に従った日付の解析 435
現在日付 198, 435
時間の値の取得 445
指定された形式またはデフォルト形式での取得 458
月の値の取得 446, 454
月の正式名の使用 442, 456
月の短縮名の使用 443, 457
月名の取得 454, 455
日の取得 444, 445
年の取得 447, 455
汎用形式 196, 443
日付間の日数の計算 441
日付間の平日の数の計算 441
日付の比較 439, 440
日付への日数の追加 437
日付への年数の追加 439
日付への平日の数の追加 438
秒の値の取得 447, 454
分の値の取得 446, 453
曜日の取得 444, 445
曜日名の使用 443, 458
リストからの最新の日付の取得 448, 449
リストからの最も早い日付の取得 451, 452
例 195
1/1/70 から日付までのミリ秒数の取得 448
CrossWorlds 日付形式への再設定 443
必須属性 370, 549
「表示」メニュー (Activity Editor) 120
「移動」 120
クイック表示モード 120
「コンテンツ」ウィンドウ 121
「ズームアウト」 120
「ズームイン」 120
「ステータス・バー」 121
「設定」 121
「ツールバー」 121
デザイン・モード 120
「倍率指定ズーム」 120
「表示」メニュー (Activity Editor) *(続き)*
「プロパティー」ウィンドウ 121
「ライブラリー」ウィンドウ 120
「表示」メニュー (Map Designer Express) 26, 29
「出力ウィンドウ」 17, 26, 29
「出力をクリア」 17, 26, 29, 93
「ステータス・バー」 17, 26, 29
「設定」 22, 29
「ダイアグラム」 21, 26, 29, 41
「ツールバー」 26, 29
「ビジネス・オブジェクト」ペイン 20, 26, 29
「Server Pane」 20, 26, 29
「表示」メニュー (Relationship Designer Express) 263, 264
「参加者タイプ」 264, 266
「ステータス・バー」 263
「ツールバー」 263
「ツリーを縮小」 265
「ツリーを展開」 264
標準ツールバー (Activity Editor) 122
「アクティビティーを保管」 122
「切り取り」 122
「コピー」 122
「削除」 122
「貼り付け」 122
「ヘルプ」 122
「Print Activity」 122
標準ツールバー (Map Designer Express) 27
「印刷」 82
「検索」 79
「新規マップ」 34
表示 26, 29
「マップをファイルから開く」 63
「マップをファイルに保管」 57
「マップをプロジェクトから開く」 62
「マップをプロジェクトに保管」 55
標準ツールバー (Relationship Designer Express) 263
「関係を保管」 267
「新規関係」 266
「新規参加者」 266
「表示」 263, 265
「ファイル」メニュー (Activity Editor) 119
「印刷」 120
「印刷設定」 120
「印刷プレビュー」 120
「閉じる」 120
「保管」 119, 192, 200
「ファイル」メニュー (Map Designer Express) 28
「印刷」 28, 82
「印刷設定」 28, 82
「印刷プレビュー」 28, 82
「コンパイル」 28, 64, 91, 95, 213, 217, 222
「削除」 28, 84
「サブマップでコンパイル」 28, 91
「終了」 28, 64
「新規」 28, 34
「すべてコンパイル」 28, 91

「ファイル」メニュー (Map Designer Express) *(続き)*
「閉じる」 28, 64
「開く」 28, 62, 63
「別名保管」 28, 56, 57
「保管」 28, 55, 57
「マップ文書を作成」 28, 70
「マップ文書を表示」 28, 71
「マップを検証」 28, 90
「ファイル」メニュー (Relationship Designer Express) 264
「関係定義の保管」 267, 272
「参加者定義を追加」 264, 266
「新規」 264
「新規関係定義」 266
「すべて保管」 264
「プロジェクトへの切り替え」 264
「保管」 264
複合一致関係 249, 251, 267, 300, 311
子インスタンスの管理 309
サブマップ 308
参加者タイプ 300
定義 268, 269, 300
マップの規則のカスタマイズ 303
メイン・マップ 304
maintainChildVerb() 304, 317
maintainCompositeRelationship() 302, 485
複合一致関係の保守のための関数ブロック 301
複数マップ・マップ・テーブル 68
複製
ビジネス・オブジェクト 360
ビジネス・オブジェクト配列 381
ブランク属性値 367
ブレークポイント 98, 101
プロジェクト 16, 260
作業 16
処理 260
ブラウズ 261
マップのオープン 17
マップの保管 17, 55, 261
プロジェクトのブラウズ 261
「プロジェクトへの切り替え」(Relationship Designer Express) 261
「分割」変換 19, 25, 41, 46, 59, 62, 90, 114
「ヘルプ」メニュー (Activity Editor) 121
ドキュメンテーション 121
「ヘルプ・トピック」 121
「ヘルプ」メニュー (Map Designer Express) 30
「ドキュメンテーション」 30
「ヘルプ・トピック」 30
Map Designer Express に関する 30
「ヘルプ」メニュー (Relationship Designer Express) 265
「ドキュメンテーション」 265
「ヘルプ・トピック」 265
Relationship Designer Express に関する 265
変換 6, 19, 41, 55, 190, 202
値を設定 19, 41, 42
宛先属性 19
変換 *(続き)*
移動 19, 41, 43
概要 5
カスタム 19, 41, 54
関係属性 54, 59, 281
関係に応じた定義 281, 303
完全性のチェック 58
結合 19, 41, 44
検証 62, 90
コードの検査 89
コンテキスト・メニュー 30
サブマップ 19, 41
実行順序 18, 62, 85, 90
ストリング 199
選択 28
ソース属性 18
ソース・データを検証する 205
相互参照 19, 41, 53
属性のアドレッシング 184
内容ベースのロジック 190
反転マップ 76
日付の形式設定 195
標準 19, 41, 114
分割 19, 41, 46
変数 187
マップ定義ファイル内の 57
マップ文書 67
変換コード
位置 91
欠落 58
検査 89
削除 82
自動更新モード 55, 114
生成 48
対応しない区切り文字 89
テキストの検索 79
パッケージをインポート 177
表示 71
プログラミング考慮事項 324
例外の処理 203
変換ステップ 6, 19, 34, 40, 82, 90
「編集」メニュー (Activity Editor) 120
「行に移動」 90, 120
「切り取り」 120
「検索」 120
「コピー」 120
「削除」 120
「全選択」 120
「置換」 120
「貼り付け」 120
「元に戻す」 120
「やり直し」 120
「編集」メニュー (Map Designer Express) 28
「行を挿入」 28
「現在の選択範囲を削除」 28, 40, 82, 83
「検索」 29, 41, 58, 79

「編集」メニュー (Map Designer Express) *(続き)*
「全選択」 28
「置換」 29, 81
「ビジネス・オブジェクトを削除」 28, 83
「ビジネス・オブジェクトを追加」 28, 38, 186, 187
「マップ・プロパティー」 29, 64, 110, 178, 186, 206
「編集」メニュー (Relationship Designer Express) 264
「拡張設定」 264, 268, 270, 273, 276
「切り取り」 264
「コピー」 264, 272
「削除」 277
「名前変更」 264
「貼り付け」 264, 272
変数 184, 190
一時 187
グローバル 187, 190, 202
宣言 189
ビジネス・オブジェクトの 184
cwExecCtx 207, 461, 476, 477, 479, 481, 485, 487, 490
strInitiator 193
ポリモアフィック・マップ 86

[マ行]

マッピング
概要 3
サポート 3
自動 72
単純 5
ツール 9, 10
定義済み 3
反転 72
標準 58, 111, 326
ポリモアフィック 86
マッピング API 10, 58
関係クラス 11
ビジネス・オブジェクト・クラス 11
ユーティリティー・クラス 11
BaseDLM クラス 11
BusObj クラス 11
BusObjArray クラス 11, 379
CwDBConnection クラス 11, 395
CwDBStoredProcedureParam クラス 11, 409
DTP クラス 10
DtpConnection クラス 10, 11, 413
DtpDataConversion クラス 10, 421
DtpDate クラス 10, 433
DtpMapService クラス 10, 461
DtpSplitString クラス 10, 463
DtpUtils クラス 10, 471
IdentityRelationship クラス 11, 475
MapExeContext クラス 11, 493
Participant クラス 11, 501
Relationship クラス 11, 509
UserStoredProcedureParam クラス 11, 525
マッピング関数ブロック 150
マッピング関数ブロック *(続き)*
Run Map 150
Run Map with Context 151
マッピングの役割 66
マップ
印刷 82
オープン 61
開発ファイル 13
基本クラス 345
現行 55, 61, 91, 190, 347
検証 23, 55, 62, 90
コーディング 113, 241
コンパイル 17, 21, 56, 91, 93, 95, 178
削除 84
作成 33
実行コンテキスト 207
実行の表示 94, 102
自動作成 72
処理 61
定義済み 3, 8, 15
テスト 94, 103
デバッグ 103, 109
閉じる 64
名前 6, 37, 65, 347
名前変更 56
反転マップの作成 76
ファイルへの保管 57
プロジェクトへの保管 55
保管 21, 37, 55, 83
ポリモアフィック 86
マップ文書 67
命名 37
モジュール性の向上 48
例外 203
HTML バージョン 67
Web サービスの使用 181
XML 版 57
マップ開発 12, 15
マップ定義 5, 7
アンロード 87
位置 5
作成 34
「新規マップ」ウィザード 34
定義済み 5
マップ定義ファイル内の 57
命名規則 6
ロード 87
マップの実行
一時停止 98, 101
関係インスタンス 252, 256
実行順序 18, 62, 85, 90
続行 101, 102
テスト実行 94
トランザクション 241, 339, 346, 349
表示 94, 102
マップ・インスタンス 8

マップの実行 *(続き)*
目的 208
マップの実行コンテキスト 207
オリジナル要求ビジネス・オブジェクト 210, 293, 295, 298, 320, 322, 496
クラス 207, 493
呼び出しコンテキスト 208, 494, 495, 498, 499
cxExecCtx 207
マップの自動化
同義語の作成 78
反転マップの作成 76
マップの自動作成 72
マップ文書 67
マップ・インスタンス 8
オリジナル要求ビジネス・オブジェクト 496
クラス 8, 345
コネクター名 493, 496
再使用 189, 202
実行コンテキスト 8, 207
始動 207, 547
データ検証レベル 206
定義済み 8
停止 207, 547
トランザクション・プログラミング・モデル 238, 349
トレース・レベル 547
内容 8
呼び出しコンテキスト 494, 497
マップ・プロパティー 11, 64
サーバー・コンポーネント管理ビューからの更新 66, 203, 207
トレース・レベル 356, 546
「マップ・ファイル宣言ブロック」 178
「マップ・ローカル宣言ブロック」 178
ランタイム 66
DataValidationLevel 110, 206
「マップ・プロパティー」ダイアログ (Map Designer Express)
「一般」タブ 65, 178, 190, 202, 206, 238, 347, 546
「ビジネス・オブジェクト」タブ 185, 186, 189
マップ・ユーティリティー・パッケージ 177, 178
マップ・リポジトリー・ファイル 87
明示的なトランザクション・ブラケット 237
接続の解放 241
トランザクションのスコープ 239
命名規則
関係定義 252, 266
参加者定義 255, 266
マップ 6
メッセージ 21
位置 21, 537
重大度 544, 545
テキスト 541
内部のパラメーター 537, 541
番号 541
フォーマット 541
変更 545
5000 539, 540
メッセージ *(続き)*
5001 539, 540
5002 539, 540
5003 539, 540
5007 539, 540
5008 539, 540
5009 539, 541
メッセージ・ファイル 537, 547
位置 14, 537
概要 537
コメント 542
使用 195, 542, 544
使用する操作 543
定義済み 537
どれを使用するかの選択 537
表示 21
フォーマット 541
保守 542
CWMapMessages.txt 195, 538
mapName.txt 540
mapName_locale.txt 537
UserMapMessages.txt 538, 540

[ヤ行]

呼び出しコンテキスト 208
値の検査 193
一致関係 290
カスタム関係 324
検索 494
設定 497
テスト 103
基づくロジック 193
例 211
ACCESS_REQUEST 208, 290
ACCESS_RESPONSE 208, 290
EVENT_DELIVERY 208, 290
SERVICE_CALL_FAILURE 208, 290
SERVICE_CALL_REQUEST 208, 290
SERVICE_CALL_RESPONSE 208, 290
strInitiator 193

[ラ行]

リポジトリー
関係のエクスポート 339
関係表のロケーション 329
マップのエクスポート 87
リレーションシップ・データベース 252, 253
リレーションシップ・データベース 252
位置 14, 252, 253, 273, 275, 276
最後の書き込みによって影響された行 417
照会の実行 329
処理する必要のあるさらなる行の照会 418
接続 329, 339, 348

リレーションシップ・データベース *(続き)*
切断 354
タイプ 274, 275
次の行の検索 419
トランザクションが進行中かどうかを判別 418
変更 331
ユーザー・アカウント 273, 275
SQL 照会 10, 415
例外
関係 203
生成 352, 543
タイプ 358
定義済み 203, 358
CollaborationException クラス 358
CwDBTransactionException 239, 241, 347, 396, 397, 405, 406
RelationshipRuntimeException クラス 203
例外処理 203, 205
例外タイプ 358
ロギング 110, 195, 543, 545
原則 544
重大度レベル 544
メッセージを送信するメソッド 350, 543, 544
例 544
レベル 545
論理演算子 378

A

ACCESS_REQUEST 呼び出しのコンテキスト 208, 209, 290
オリジナル要求ビジネス・オブジェクト 210, 496
検索 494
設定 497
テスト 105, 108
Create 動詞 291, 315, 320, 324, 325
Delete 動詞 291, 315, 326
foreignKeyXref() 319, 481
getOriginalRequestBO() 496
maintainChildVerb() 315, 484
maintainCompositeRelationship() 302, 486
maintainSimpleIdentityRelationship() 290, 488
Retrieve 動詞 291, 315, 320
Update 動詞 291, 315, 320
ACCESS_RESPONSE 呼び出しコンテキスト 208, 210, 290
オリジナル要求ビジネス・オブジェクト 210, 298, 496
検索 494
設定 498
foreignKeyXref() 322, 481
getOriginalRequestBO() 496
maintainCompositeRelationship() 302, 486
maintainSimpleIdentityRelationship() 298, 489
updateMyChildren() 311
Activity Editor 113
アクセス 30, 43, 46, 48, 52, 114
アクティビティーの保管 161, 171
「値を設定」変換 43, 114
Activity Editor *(続き)*
カスタム変換 55
関数ブロック 123, 126, 309
キーボードのショートカット 119
「記述を追加」 121, 169
クイック表示モード 117, 118
グラフィック表示 114, 115
「結合」変換 46, 114
「コメント」 125
「コメントを追加」 121
コンテキスト・メニュー 121
「コンテンツ」ウィンドウ 117
コンポーネントのグループ化 126
「サブマップ」変換 52, 114
始動 113, 114
使用例 157, 162, 171
「新規定数」 121, 125, 168
ステータス・バー 123
「設定」ダイアログ 55
説明 125
「相互参照」変換 114
「対応しない区切り文字を検査」 89
タイトル・バー 115
「ツール」メニュー 121
ツールバー 121
デザイン・モード 115, 117
「表示」メニュー 120
「ファイル」メニュー 119
「プロパティー」ウィンドウ 117
「分割」変換 48, 114
文書表示域 115
「ヘルプ」メニュー 121
「編集」メニュー 120
メイン表示 114
メインメニュー 119
「予定」 125
「予定を追加」 121
「ライブラリー」ウィンドウ 116
「ラベル」 125
「ラベルを追加」 121
レイアウト 114
「Add To My Collection」 121
「Connection Links」 124
Java 表示 114, 117
「Resize」ラベル 125
Web サービスの使用 180
addDays() メソッド 196, 437
addElement() メソッド 380
addMyChildren() メソッド 204, 324, 475, 503, 524
addParticipant() メソッド 325, 510
addWeekdays() メソッド 196, 438
addYears() メソッド 196, 439
after() メソッド 196, 439
AnyException 例外 352
APIs/Business Object Array の関数ブロック 133
Add Element 133

APIs/Business Object Array の関数ブロック *(続き)*
Duplicate 133
Equals 133
Get Element At 133
Get Elements 133
Get Last Index 133
Is Business Object Array 133
Max attribute value 133
Min attribute value 133
Remove All Elements 134
Remove Element 134
Remove Element At 134
Set Element At 134
Size 134
Sum 134
Swap 134
To String 134
APIs/Business Object 関数ブロック 128
値を設定 132
Copy 128
Duplicate 128
Equal Keys 128
Equals 128
Exists 129
Get Boolean 129
Get Business Object 129
Get Business Object Array 129
Get Business Object Type 129
Get Double 129
Get Float 129
Get Int 129
Get Long 129
Get Long Text 130
Get Object 130
Get String 130
Get Verb 130
Is Blank 130
Is Business Object 130
Is Key 130
Is Null 130
Is Required 130
Iterate Children 131
Key to String 131
New Business Object 131
Set Content 131
Set Default Attribute Values 131
Set Keys 131
Set Value with Create 131
Set Verb 131
Set Verb with Create 131
Shallow Equals 132
To String 132
Valid Data 132
APIs/Business Object/Array の関数ブロック 132
GetBusObj At 132
New Business Object Array 132
APIs/Business Object/Array の関数ブロック *(続き)*
Set BusObj At 132
Size 132
APIs/Business Object/Constants の関数ブロック 132
動詞 Create 132
動詞 Delete 132
動詞 Retrieve 133
動詞 Update 133
APIs/Database Connection の関数ブロック 134
Begin Transaction 134
Commit 134
Execute Prepared SQL 135
Execute Prepared SQL with Parameter 135
Execute SQL 135
Execute SQL with Parameter 135
Execute Stored Procedure 135
Get Database Connection 135
Get Database Connection with Transaction 135
Get Next Row 135
Get Update Count 135
Has More Rows 135
In Transaction 135
Is Active 136
Release 136
Roll Back 136
APIs/Identity Relationship の関数ブロック 136
Add My Children 136
Delete All My Children 136
Delete My Children 136
Foreign Key Cross-Reference 137
Foreign Key Lookup 137
Maintain Child Verb 137
Maintain Composite Relationship 137
Maintain Simple Identity Relationship 137
Update My Children 138
APIs/Maps の関数ブロック 138
Get Adapter Name 138
Get Calling Context 138
Get Original Request Business Object 138
APIs/Maps/Constants の関数ブロック 138
Calling Context ACCESS_REQUEST 138
Calling Context ACCESS_RESPONSE 138
Calling Context EVENT_DELIVERY 138
Calling Context SERVICE_CALL_FAILURE 139
Calling Context SERVICE_CALL_REQUEST 139
Calling Context SERVICE_CALL_RESPONSE 139
APIs/Maps/Exception の関数ブロック 139
Raise Map Exception 139
Raise Map Exception 1 139
Raise Map Exception 2 139
Raise Map Exception 3 139
Raise Map Exception 4 140
Raise Map Exception 5 140
Raise Map RunTimeEntity Exception 140
APIs/Participant の関数ブロック 140
Get Boolean Data 140

APIs/Participant の関数ブロック *(続き)*
Get Business Object Data 140
Get Double Data 140
Get Float Data 140
Get Instance ID 140
Get Int Data 141
Get Long Data 141
Get Participant Name 141
Get Relationship Name 141
Get String Data 141
New Participant 141
New Participant in Relationship 141
Set Data 141
Set Instance ID 141
Set Participant Definition 142
Set Relationship Definition 142
APIs/Participant/Array の関数ブロック 142
Get Participant At 142
New Participant Array 142
Set Participant At 142
Size 142
APIs/Participant/Constants の関数ブロック 142
APIs/Participant/Constants の関数ブロック、Participant INVALID_INSTANCE_ID 142
APIs/Relationship の関数ブロック 143, 281
Add Participant 143
Add Participant Data 143
Add Participant Data to New Relationship 143
Create Relationship 143
Create Relationship with Participant 143
Deactivate Participant 143
Deactivate Participant By Data 143
Deactivate Participant By Instance 143
Deactivate Participant By Instance Data 144
Delete Participant 144
Delete Participant By Instance 144
Delete Participant By Instance Data 144
Delete Participant with Data 144
Get Next Instance ID 144
Retrieve Instances 144
Retrieve Instances for Participant 144
Retrieve Participants 145
Retrieve Participants with ID 145
Update Participant 145
Update Participant By Instance 145
Update Participant By Instance Data 145
AppSpecificInfo 属性プロパティー 549
AttributeException 例外 352

B

BaseDLM クラス 8, 11, 345, 356
定義済み 345
メソッドの要約 345
getDBConnection() 345
getName() 347
BaseDLM クラス *(続き)*
getRelConnection() 348
implicitDBTransactionBracketing() 349
isTraceEnabled() 350
logError() 350
logInfo() 350
logWarning() 350
releaseRelConnection() 354
trace() 355
before() メソッド 196, 440
beginTransaction() メソッド 239, 395
beginTran() メソッド (使用すべきでない) 332, 413
Boolean クラス 549
ストアード・プロシージャーのパラメーター型 237, 338, 409
データ型の判別 422
変換 425
有効な変換 424
Boolean への変換 428
boolean データ型
ストアード・プロシージャーのパラメーター型 237, 409
属性値の検索 364
属性の設定 371
データ型の判別 422
変換 427
有効なデータの検査 377
有効な変換 424
Boolean への変換 425
BOOL_TYPE 定数 422
BusObj クラス 11, 357, 378
使用すべきでないメソッド 378
定義済み 357
メソッドの要約 357
copy() 360
duplicate() 360
equalKeys() 361
equalsShallow() 363
equals() 362
exists() 363
getCount() 378
getKeys() 378
getLocale() 366, 374
getType() 366
getValues() 378
getVerb() 367
isBlank() 367
isKey() 368
isNull() 368
isRequired() 370
keysToString() 370
not() 378
setContent() 372
setDefaultAttrValues() 373
setKeys() 373
setVerbWithCreate() 375
setVerb() 374

BusObj クラス *(続き)*
setWithCreate() 375
set() 371, 378
toString() 376
validData() 377
BusObjArray クラス 11, 379, 393
定義済み 379
メソッドの要約 379
addElement() 380
duplicate() 381
elementAt() 381
equals() 382
getElements() 382
getLastIndex() 383
maxBusObjArray() 384
maxBusObjs() 385
max() 383
minBusObjArray() 387
minBusObjs() 388
min() 386
removeAllElements() 389
removeElementAt() 390
removeElement() 389
setElementAt() 390
size() 391
sum() 391
swap() 392
toString() 392

C

calcDays() メソッド 196, 441
calcWeekdays() メソッド 196, 441
CALL ステートメント 231, 233, 334, 399, 400, 415
CANNOTCONVERT 定数 423
Cardinality 属性プロパティー 549
catch ステートメント 111, 215, 326
CLASSPATH 環境変数 178, 180
CLASSPATH 設定 178
CollaborationException クラス 358
CollabUtils.jar ファイル 178
commit() メソッド (CwDBConnection) 239, 396
commit() メソッド (DtpConnection) 332, 354, 414
ContainedObjectVersion 属性プロパティー 549
copy() メソッド 360, 378
Create 動詞
関係インスタンス 324, 325
条件付設定 312
foreignKeyXref() 320, 322
maintainChildVerb() 315, 317
maintainCompositeRelationship() 302
maintainSimpleIdentityRelationship() 291, 293, 295, 298
create() メソッド 204, 324, 503, 506, 512
CwDBConnection クラス 11, 395, 407
オブジェクトの作成 224, 345
行にアクセスするためのメソッド 225
CwDBConnection クラス *(続き)*
ストアード・プロシージャーを呼び出すメソッド 231
トランザクションを管理するためメソッド 239
メソッドの要約 395
beginTransaction() 395
commit() 396
executePreparedSQL() 398
executeSQL() 399
executeStoredProcedure() 401
getUpdateCount() 402
hasMoreRows() 402
inTransaction() 403
isActive() 404
nextRow() 404
release() 405
rollBack() 406
CwDBStoredProcedureParam クラス 11, 234, 409, 412
コンストラクター 409
メソッドの要約 409
getParamType() 410
getValue() 411
CwDBStoredProcedureParam() コンストラクター 234, 409
CwDBTransactionException 例外 239, 241, 347, 396, 397, 405, 406
cwExecCtx 変数 207, 461, 476, 477, 479, 481, 485, 487, 490
CWMapMessages.txt メッセージ・ファイル 195, 538
CWMAPTYPE 定数 461
CxMissingIDException 例外 539

D

DataValidationLevel マップ・プロパティー 110, 206
Date クラス 237, 409, 422, 424, 549
Date の関数ブロック 145
使用例 164
Add Day 145
Add Month 145
Add Year 146
Date After 146
Date Before 146
Date Equals 146
Format Change 146, 164
Get Day 146
Get Month 146
Get Year 147
Get Year Month Day 147
Now 147
Date/Formats の関数ブロック 147
yyyyMMdd 147
yyyyMMdd HHmmss 147
yyyy-MM-dd 147
DATE_TYPE 定数 422
deactivateParticipantByInstance() メソッド 325, 514
deactivateParticipant() メソッド 325, 513
DefaultValue 属性プロパティー 549
DELETE ステートメント 228, 331, 399, 400

Delete 動詞
関係インスタンス 326
foreignKeyXref() 322
maintainChildVerb() 315, 317
maintainCompositeRelationship() 302
maintainSimpleIdentityRelationship() 291, 293, 295, 298
deleteMyChildren() メソッド 477
deleteParticipantByInstance() メソッド 325, 517
deleteParticipant() メソッド 325, 516
Designer ツールバー (Map Designer Express) 27
「検証」 91
「コンパイル」 91
「ステップオーバー」 102
「すべての属性」 26
「すべてのブレークポイントをクリア」 100
「続行」 101
「テスト実行」 101
「ビジネス・オブジェクトを追加」 38
表示 26, 29
「ブレークポイントの切り替え」 98
「リンクされていない属性のみ」 26
「リンクされている属性のみ」 26
Double クラス 549
最小値の取得 386, 387, 388
最大値の取得 383, 384, 385
ストアード・プロシージャーのパラメーター型 237, 338, 409
データ型の判別 422
変換 425
有効な変換 424
Double への変換 428
Float への変換 426, 429
Integer への変換 427, 430
String への変換 430
double データ型
ストアード・プロシージャーのパラメーター型 237, 409
属性値の検索 364
属性の設定 371
データ型の判別 422
変換 428
有効なデータの検査 377
有効な変換 424
Double への変換 426
Float への変換 426, 429
Integer への変換 427, 430
String への変換 430
DOUBLE_TYPE 定数 422
DtpConnection クラス (使用すべきでない) 10, 11, 413, 420
オブジェクトの作成 348
行にアクセスするためのメソッド 330
ストアード・プロシージャーを呼び出すメソッド 333
トランザクションを管理するためメソッド 332
メソッドの要約 413
beginTran() 413
commit() 414
execStoredProcedure() 416
DtpConnection クラス (使用すべきでない) *(続き)*
executeSQL() 415
getUpdateCount() 417
hasMoreRows() 418
inTransaction() 418
nextRow() 419
rollBack() 419
DtpDataConversion クラス 10, 421, 431
定義済み 421
メソッドの要約 421
CANNOTCONVERT 423
getType() 421
isOKToConvert() 423
OKTOCONVERT 423
POTENTIALDATALOSS 423
toBoolean() 425
toDouble() 425
toFloat() 426
toInteger() 427
toPrimitiveBoolean() 427
toPrimitiveDouble() 428
toPrimitiveFloat() 429
toPrimitiveInt() 429
toString() 430
DtpDate クラス 10, 433, 459
規則 433
メソッドの要約 433
addDays() 437
addWeekdays() 438
addYears() 439
after() 439
before() 440
calcDays() 441
calcWeekdays() 441
DtpDate() 435
get12MonthNames() 442
get12shortMonthNames() 443
get7DayNames() 443
getCWDate() 443
getDayOfMonth() 444
getDayOfWeek() 444
getHours() 445
getIntDayOfWeek() 445
getIntDay() 445
getIntMilliSeconds() 446
getIntMinutes() 446
getIntMonth() 446
getIntSeconds() 447
getIntYear() 447
getMaxDateBO() 449
getMaxDate() 448
getMinDateBO() 452
getMinDate() 451
getMinutes() 453
getMonth() 454
getMSSince1970() 448

DtpDate クラス *(続き)*
getNumericMonth() 454
getSeconds() 454
getShortMonth() 455
getYear() 455
set12MonthNamesToDefault() 456
set12MonthNames() 456
set12ShortMonthNamesToDefault() 457
set12ShortMonthNames() 457
set7DayNamesToDefault() 458
set7DayNames() 458
toString() 458
DtpDateException 例外 196
DtpDate() コンストラクター 196, 198, 435
DtpMapService クラス 10, 461, 462
メソッドの要約 461
runMap() 461
DtpSplitString クラス 10, 463, 469
定義済み 463
メソッドの要約 463
DtpSplitString() 463
elementAt() 464
firstElement() 465
getElementCount() 465
getEnumeration() 466
lastElement() 466
nextElement() 467
prevElement() 468
reset() 468
DtpSplitString() コンストラクター 463
DtpUtils クラス 10, 471, 473
メソッドの要約 471
padLeft() 471
padRight() 471
stringReplace() 472
truncate() 473
duplicate() メソッド 360, 381

E

elementAt() メソッド 381, 464
Enumeration クラス 226, 330
equalKeys() メソッド 361
equalsShallow() メソッド 363
equals() メソッド 362, 382
EVENT_DELIVERY 呼び出しコンテキスト 208, 209, 290
オリジナル要求ビジネス・オブジェクト 210, 496
検索 494
設定 497
テスト 105, 108
Create 動詞 291, 315, 320, 324, 325
Delete 動詞 291, 315, 326
foreignKeyXref() 319, 481
getOriginalRequestBO() 496
maintainChildVerb() 315, 484
maintainCompositeRelationship() 302, 486
EVENT_DELIVERY 呼び出しコンテキスト *(続き)*
maintainSimpleIdentityRelationship() 290, 488
Retrieve 動詞 291, 315, 320
Update 動詞 291, 315, 320
updateMyChildren() 311
execStoredProcedure() メソッド (使用すべきでない) 333, 334, 416
executePreparedSQL() メソッド 229, 231, 232, 398
executeSQL() メソッド (CwDBConnection) 224, 231, 232, 399
executeSQL() メソッド (DtpConnection) 329, 333, 334, 415
executeStoredProcedure() メソッド 231, 233, 401
exists() メソッド 363

F

firstElement() メソッド 465
Float クラス 549
最小値の取得 386, 387, 388
最大値の取得 383, 384, 385
ストアード・プロシージャーのパラメーター型 237, 338, 409
データ型の判別 422
変換 426
有効な変換 424
Double への変換 426, 428
Float への変換 429
Integer への変換 427, 430
String への変換 430
float データ型
ストアード・プロシージャーのパラメーター型 237, 409
属性値の検索 364
属性の設定 371
データ型の判別 422
変換 429
有効なデータの検査 377
有効な変換 424
Double への変換 426, 428
Float への変換 426
Integer への変換 427, 430
String への変換 430
FLOAT_TYPE 定数 422
Foreign Key Cross-Reference 関数ブロック 319, 323
Foreign Key Lookup 関数ブロック 318, 323
foreignKeyLookup() メソッド 59, 318, 326, 327, 478, 539
foreignKeyXref() メソッド 59, 319, 326, 327, 328, 480, 539

G

get12MonthNames() メソッド 196, 442
get12ShortMonthNames() メソッド 196, 443
get7DayNames() メソッド 196, 443
getConnName() メソッド 493
getCount() メソッド (使用すべきでない) 378
getCWDate() メソッド 196, 443
getDayOfMonth() メソッド 196, 444

getDayOfWeek() メソッド 196, 444
getDBConnection() メソッド 223, 238, 345, 346
getElementCount() メソッド 465
getElements() メソッド 382
getEnumeration() メソッド 466
getGenericBO() メソッド (使用すべきでない) 499
getHours() メソッド 196, 445
getInitiator() メソッド 207, 494
getInstanceId() メソッド 204, 504
getIntDayOfWeek() メソッド 196, 445
getIntDay() メソッド 196, 445
getIntMilliSeconds() メソッド 446
getIntMinutes() メソッド 196, 446
getIntMonth() メソッド 196, 446
getIntSeconds() メソッド 196, 447
getIntYear() メソッド 196, 447
getKeys() メソッド (使用すべきでない) 378
getLastIndex() メソッド 383
getLocale() メソッド 366, 374, 495
getMaxDateBO() メソッド 196, 449
getMaxDate() メソッド 196, 448
getMinDateBO() メソッド 196, 452
getMinDate() メソッド 196, 451
getMinutes() メソッド 196, 453
getMonth() メソッド 195, 454
getMSSince1970() メソッド 196, 448
getName() メソッド 347
getNewID() メソッド 518
getNumericMonth() メソッド 196, 454
getOriginalRequestBO() メソッド 207, 496, 499
getParamDataTypeJavaObj() メソッド (使用すべきでない) 335, 339, 526
getParamDataTypeJDBC() メソッド (使用すべきでない) 335, 339, 527
getParamIndex() メソッド (使用すべきでない) 335, 528
getParamIOType() メソッド (使用すべきでない) 335, 528
getParamName() メソッド (使用すべきでない) 335, 529
getParamType() メソッド 234, 410
getParamValue() メソッド (使用すべきでない) 335, 530
getParticipantDefinition() メソッド 504
getRelationshipDefinition() メソッド 505
getRelConnection() メソッド (使用すべきでない) 329, 348, 416, 417
getSeconds() メソッド 196, 454
getShortMonth() メソッド 195, 455
getType() メソッド 366, 421
getUpdateCount() メソッド (CwDBConnection) 228, 402
getUpdateCount() メソッド (DtpConnection) 332, 417
getValues() メソッド (使用すべきでない) 378
getValue() メソッド 234, 411
getVerb() メソッド 367
getYear() メソッド 196, 455

H

hasMoreRows() メソッド (CwDBConnection) 225, 232, 402
hasMoreRows() メソッド (DtpConnection) 330, 418

I

IdentityRelationship クラス 11, 253, 475, 492
  メソッドの要約 475
  addMyChildren() 475, 524
  deleteMyChildren() 477
  foreignKeyLookup() 478
  foreignKeyXref() 480
  maintainChildVerb() 483
  maintainCompositeRelationship() 485, 524
  maintainSimpleIdentityRelationship() 487, 524
  updateMyChildren() 489, 524
implicitDBTransactionBracketing() メソッド 238, 349
import ステートメント 178
IN パラメーター 234, 235, 411
INOUT パラメーター 234, 411
INSERT ステートメント 228, 284, 331, 398, 400, 402
int データ型
  ストアード・プロシージャーのパラメーター型 236, 409
  属性値の検索 364
  属性の設定 371
  データ型の判別 422
  変換 429
  有効なデータの検査 377
  有効な変換 424
  Double への変換 426, 428
  Float への変換 426, 429
  Integer への変換 427
  String への変換 430
Integer クラス 549
  最小値の取得 386, 387, 388
  最大値の取得 383, 384, 385
  ストアード・プロシージャーのパラメーター型 236, 338, 409
  データ型の判別 422
  変換 427
  有効な変換 424
  Double への変換 426, 428
  Float への変換 426, 429
  Integer への変換 430
  String への変換 430
INTEGER_TYPE 定数 422
inTransaction() メソッド (CwDBConnection) 239, 240, 403
inTransaction() メソッド (DtpConnection) 332, 418
INVALID_INSTANCE_ID 定数 504, 506, 513
isActive() メソッド 241, 404
isBlank() メソッド 367
IsForeignKey 属性プロパティー 549
IsKey 属性プロパティー 549
isKey() メソッド 368
isNull() メソッド 368

isOKToConvert() メソッド 423
IsRequired 属性プロパティー 549
isRequired() メソッド 370
isTraceEnabled() メソッド 350

J

JAR ライブラリー
  関数ブロックとしてのインポート 174
  表示設定のカスタマイズ 176
Java Development Kit (JDK) 12, 177
Java 演算子、NOT 378
Java クラス
  オブジェクト 364, 371, 377
  Boolean 425, 549
  Date 422, 549
  Double 425, 428, 549
  Enumeration 226, 330
  Float 426, 429, 549
  Integer 427, 430, 549
  java.sql.Types 236, 338
  StringTokenizer 463
  Vector 225, 233, 330, 399, 404, 410
Java コンパイラー (javac) 91
Java ステートメント
  catch 111, 326
  import 178
  try 111, 326
Java ツールバー (Activity Editor) 123
  「行に移動」 123
  式ビルダー 123
  「テキストを検索」 123
  「元に戻す」 123
  「やり直し」 123
  「Java コードを編集」 123
Java 表示 (Activity Editor) 114
  クイック表示モード 118
  デザイン・モード 117
  WordPad 117
JavaException 例外 353
java.lang パッケージ 421
java.sql.Types クラス 236, 338
java.util パッケージ 226, 330, 463
JCLASSES 環境変数 179

K

keysToString() メソッド 370, 378

L

lastElement() メソッド 466
logError() メソッド 195, 350, 543, 544
Logging and Tracing の関数ブロック 147
  Log error 147
Logging and Tracing の関数ブロック *(続き)*
  Log error ID 147
  Log information 147
  Log information ID 147
  Log warning 147
  Log warning ID 148
  Trace 148
Logging and Tracing/Log Error の関数ブロック 148
  Log error ID 1 148
  Log error ID 2 148
  Log error ID 3 148
Logging and Tracing/Log Information の関数ブロック 148
  Log information ID 1 148
  Log information ID 2 149
  Log information ID 3 149
Logging and Tracing/Log Warning の関数ブロック 149
  Log warning ID 1 149
  Log warning ID 2 149
  Log warning ID 3 149
Logging and Tracing/Trace の関数ブロック 150
  Trace ID 1 150
  Trace ID 2 150
  Trace ID 3 150
  Trace on Level 150
logInfo() メソッド 110, 111, 223, 350, 543, 544, 545
logWarning() メソッド 350, 544
long データ型 237, 364, 371, 377, 409
LongText クラス
  最小値の取得 386, 387, 388
  最大値の取得 383, 384, 385
  属性値の検索 364
  属性の設定 372
  データ型の判別 422
  有効な変換 424
LONGTEXT_TYPE 定数 422

M

Maintain Child Verb の関数ブロック 303, 315, 317
maintainChildVerb() メソッド 303, 315, 318, 483
  実行される検証 484
maintainCompositeRelationship() メソッド 485
  アクション 302
  エラー・メッセージ 326, 539
  使用すべきでないバージョン 524
  属性のコメント 59
maintainSimpleIdentityRelationship() メソッド 487
  エラー・メッセージ 326, 539
  実行される検証 488
  使用すべきでないバージョン 524
  属性のコメント 59
Map Designer Express 8, 15, 61
  概要 15
  起動 16
  機能 27
  「検索」コントロール・ペイン 27, 58, 79, 80

Map Designer Express *(続き)*
検索機能 79
コンテキスト・メニュー 30
「サブマップ」ダイアログ 51, 213, 214, 216, 218
「サブマップ」ダイアログの呼び出し条件 218
始動 16
自動マッピング 72
終了 28, 64
出力ウィンドウ 17, 22, 26, 27, 29, 92, 94
「新規マップ」ウィザード 34, 37
ステータス・バー 17, 26, 29
生成されたファイル 13
設定 22
タブ・ウィンドウ 9, 17, 61
「ツール」メニュー 30
ツールバー 26, 27, 31
データ変換 43
「テスト」タブ 21, 26, 94
「デバッグ」メニュー 29
反転マップ 72
「ビジネス・オブジェクト」ウィンドウ 20, 29, 39, 185
「ビジネス・オブジェクト」ペイン 20, 26, 38, 83, 189
「ビジネス・オブジェクトを削除」ダイアログ 83
「ビジネス・オブジェクトを追加」ダイアログ 38
ビジネス・オブジェクト・ブラウザー 20, 26, 29
「表示」メニュー 29
「ファイル」メニュー 28
「複数の属性」ダイアログ 18, 44
「ブレークポイント」ダイアログ 100
プログラム・ツールバー 26, 27, 29
プロジェクトでの作業 16
「ヘルプ」メニュー 30
「編集」メニュー 28
「マップを削除」ダイアログ 84
「マップを含むファイルを開く」ダイアログ 63
「マップを別名保管」ダイアログ 37, 56
マップ・ワークスペース 20, 38, 189
メインウィンドウ 17, 25
メイン・コンポーネント 17
「メッセージ」タブ 21, 26, 537
メニュー 27
レイアウト 17
「Open Map from Project」ダイアログ 62
MapExeContext クラス 11, 493, 501
使用すべきでないメソッド 499
メソッドの要約 493
呼び出しコンテキスト定数 208
getConnName() 493
getGenericBO() 499
getInitiator() 494
getLocale() 495
getOriginalRequestBO() 496
setConnName() 496
setInitiator() 497
setLocale() 498
MapFailureException 例外 326
mapName.txt メッセージ・ファイル 540
mapName._locale.txt メッセージ・ファイル 537
Math の関数ブロック 151
Absolute value 151
Ceiling 151
Divide 151
Equal 151
Floor 151
Greater than 151
Greater than or Equal 151
Less than 151
Less than or equal 152
Maximum 152
Minimum 152
Minus 152
Multiply 152
Not a Number 152
Not Equal 152
Number to String 152
Plus 152
Round 153
String to Number 153
maxBusObjArray() メソッド 384
maxBusObjs() メソッド 385
MaxLength 属性プロパティー 549
max() メソッド 383
MAX_CONNECTION_POOLS 構成パラメーター 274, 276
Message
5000 326
5001 326
5002 326
5003 326
5007 327
5008 327
5009 327, 328
minBusObjArray() メソッド 387
minBusObjs() メソッド 388
min() メソッド 386

N

nextElement() メソッド 467
nextRow() メソッド (CwDBConnection) 225, 232, 404
nextRow() メソッド (DtpConnection) 330, 419
NOT 演算子 378
not() 378
null 属性値 326, 368

O

Object クラス 364, 371, 377
ObjectEventId 属性 58, 90, 96, 104
ObjectException 例外 353
OKTOCONVERT 定数 423
OperationException 例外 353

OUT パラメーター 232, 234, 235, 411, 412

P

padLeft() メソッド 471
padRight() メソッド 471
PARAM_IN 定数 235, 411
PARAM_INOUT 定数 411
PARAM_OUT 定数 235, 411
Participant クラス 11, 253, 257, 501, 507
  定義済み 501
  メソッドの要約 501
  getInstanceId() 504
  getParticipantDefinition() 504
  getRelationshipDefinition() 505
  Participant() 501
  setInstanceId() 506
  setParticipantDefinition() 506
  setRelationshipDefinition() 507
  set() 505
Participant() コンストラクター 501, 506
PATH 環境変数 12, 91
POTENTIALDATALOSS 定数 423
prevElement() メソッド 468
Properties の関数ブロック 153
Properties の関数ブロック、Get Property 153

R

Relationship Designer Express 8
  概要 259
  「拡張設定」ダイアログ 268, 273, 276, 277, 278
  起動 259
  機能 263
  「グローバル・デフォルト設定」ダイアログ 276
  始動 259
  ステータス・バー 263, 265
  「ツール」メニュー 265
  ツールバー 263, 265
  「表示」メニュー 264
  「ファイル」メニュー 264
  プログラム・ツールバー 260
  プロジェクトの処理 260
  「ヘルプ」メニュー 265
  「編集」メニュー 264
  メインウィンドウ 263
  メニュー 264
  レイアウト 261
Relationship Manager 285
Relationship クラス 11, 253, 509, 525
  ガイドライン 509
  使用すべきでないメソッド 523
  定義済み 509
  メソッドの要約 509
  addParticipant() 510
Relationship クラス (続き)
  create() 512
  deactivateParticipantByInstance() 514
  deactivateParticipant() 513
  deleteParticipantByInstance() 517
  deleteParticipant() 516
  getNewID) 518
  retrieveInstances() 518
  retrieveParticipants() 520
  updateParticipantByInstance() 522
  updateParticipant() 521
Relationship 属性プロパティー 549
Relationship の関数ブロック 153, 281
  使用例 173
  Maintain Identity Relationship 153
  Static Lookup 153, 173
RelationshipRuntimeException クラス 106, 203, 326, 539
releaseRelConnection() メソッド (使用すべきでない) 354
release() メソッド 241, 405
removeAllElements() メソッド 389
removeElementAt() メソッド 390
removeElement() メソッド 389
repos_copy ユーティリティー 87, 339
reset() メソッド 468
Retrieve 動詞 485, 488
  foreignKeyXref() 320, 322
  maintainChildVerb() 315, 317
  maintainCompositeRelationship() 302
  maintainSimpleIdentityRelationship() 291, 293, 295, 298
retrieveInstances() メソッド 59, 286, 312, 518
retrieveParticipants() メソッド 59, 287, 312, 520
rollBack() メソッド (CwDBConnection) 239, 406
rollBack() メソッド (DtpConnection) 332, 419
runMap() メソッド 53, 87, 207, 213, 216, 220, 222, 323, 461

S

SELECT ステートメント 225, 229, 329, 398, 400, 403, 405
ServiceCallException 例外 353
SERVICE_CALL_FAILURE 呼び出しコンテキスト 208, 209, 290
  オリジナル要求ビジネス・オブジェクト 210, 496
  検索 494
  設定 498
  汎用ビジネス・オブジェクト 210
  getOriginalRequestBO() 496
  maintainCompositeRelationship() 302
  maintainSimpleIdentityRelationship() 297
SERVICE_CALL_REQUEST 呼び出しコンテキスト 208, 209, 290
  オリジナル要求ビジネス・オブジェクト 210, 496
  検索 494
  設定 497
  テスト 106, 109
  汎用ビジネス・オブジェクト 210
  Create 動詞 293, 315, 322

SERVICE_CALL_REQUEST 呼び出しコンテキスト *(続き)*
Delete 動詞 293, 315, 322
foreignKeyXref() 321, 481
getOriginalRequestBO() 496
maintainChildVerb() 315, 484
maintainCompositeRelationship() 302, 485, 486
maintainSimpleIdentityRelationship() 292, 488, 489
Retrieve 動詞 293, 315, 322
Update 動詞 293, 316, 322
updateMyChildren() 311
SERVICE_CALL_RESPONSE 呼び出しコンテキスト 208, 209, 290
一致関係 106
オリジナル要求ビジネス・オブジェクト 210, 297, 496
検索 494
設定 497
テスト 107, 109
汎用ビジネス・オブジェクト 106, 210
Create 動詞 295, 317, 320, 325
Delete 動詞 295, 317, 326
foreignKeyXref() 319, 481
getOriginalRequestBO() 496
maintainChildVerb() 316, 484
maintainCompositeRelationship() 302, 486
maintainSimpleIdentityRelationship() 294, 488
Retrieve 動詞 295, 317, 320
Update 動詞 295, 317, 320
updateMyChildren() 311
set12MonthNamesToDefault() メソッド 196, 456
set12MonthNames() メソッド 196, 456
set12ShortMonthNamesToDefault() メソッド 196, 457
set12ShortMonthNames() メソッド 196, 457
set7DayNamesToDefault() メソッド 196, 458
set7DayNames() メソッド 196, 458
setConnName() メソッド 496
setContent() メソッド 372
setDefaultAttrValues() メソッド 373
setElementAt() メソッド 390
setInitiator() メソッド 207, 497
setInstanceId() メソッド 506
setKeys() メソッド 373
setLocale() メソッド 498
setParamDataTypeJavaObj() メソッド (使用すべきでない) 335, 339, 531
setParamDataTypeJDBC() メソッド (使用すべきでない) 335, 339, 531
setParamIndex() メソッド (使用すべきでない) 335, 532
setParamIOType() メソッド (使用すべきでない) 335, 532
setParamName() メソッド (使用すべきでない) 335, 533
setParamValue() メソッド (使用すべきでない) 335, 534
setParticipantDefinition() メソッド 506
setRelationshipDefinition() メソッド 507
setVerbWithCreate() メソッド 375
setVerb() メソッド 215, 308, 374, 378
setWithCreate() メソッド 375
set() メソッド 371, 378, 505
size() メソッド 383, 391
SQL 照会 10, 223, 241, 328, 339
さらなる行の検査 225, 402, 418
実行 224, 329, 398, 399, 401, 415
準備済み 229, 398
静的 224, 399
次の行の検索 225, 404
start_server.bat ファイル 178, 180
Static Lookup の関数ブロック、使用例 173
String クラス 549
最小値の取得 386, 387, 388
最大値の取得 383, 384, 385
ストアード・プロシージャーのパラメーター型 236, 338, 409
属性値の検索 364
属性の設定 371
データ型の判別 422
変換 430
有効なデータの検査 377
有効な変換 424
Boolean への変換 425, 428
Double への変換 426, 428
Float への変換 426, 429
Integer への変換 427, 430
stringReplace() メソッド 472
Strings
あるパターンを別のパターンに置換 472
指定した文字での埋め込み 471
StringTokenizer クラス 463
STRING_TYPE 定数 422
strInitiator 組み込み変数 193
sum() メソッド 391
swap() メソッド 392
System Manager 11
関係カテゴリー 278
「コンポーネント」メニュー 92, 267, 547
接続プール 224
プロジェクトからのマップのオープン 62
マップのコンパイル 92
「マップ・プロパティー」ウィンドウ 66, 203, 207, 238, 547
Map Designer Express の始動 16
Relationship Designer Express の始動 259
SystemException 例外 353

T

toBoolean() メソッド 425
toDouble() メソッド 425
toFloat() メソッド 426
toInteger() メソッド 427
toPrimitiveBoolean() メソッド 427
toPrimitiveDouble() メソッド 428
toPrimitiveFloat() メソッド 429
toPrimitiveInt() メソッド 429
toString() メソッド 196, 376, 378, 392, 430, 458

trace() メソッド 355, 543, 545
truncate() メソッド 473
try ステートメント 111, 215, 326
Type 属性プロパティー 549

U

UNKNOWN_TYPE 定数 422
Update My Children の関数ブロック 310
UPDATE ステートメント 228, 331, 399, 400, 402
Update 動詞
  条件付設定 312
  foreignKeyXref() 320, 322
  maintainChildVerb() 315, 316, 317
  maintainCompositeRelationship() 302
  maintainSimpleIdentityRelationship() 291, 293, 295, 298
updateMyChildren() メソッド 311, 489, 524
updateParticipantByInstance() メソッド 522
updateParticipant() メソッド 521
Upper Case の関数ブロック、使用例 159
UserMapMessages.txt メッセージ・ファイル 538, 540
UserStoredProcedureParam クラス (使用すべきでない) 11, 525, 534
  コンストラクター 526
  メソッドの要約 525
  getParamDataTypeJavaObj() 526
  getParamDataTypeJDBC() 527
  getParamIndex() 528
  getParamIOType() 528
  getParamName() 529
  getParamValue() 530
  setParamDataTypeJavaObj() 531
  setParamDataTypeJDBC() 531
  setParamIndex() 532
  setParamIOType() 532
  setParamName() 533
  setParamValue() 534
UserStoredProcedureParam() コンストラクター (使用すべきでない) 335, 526
Utilities の関数ブロック 156
Utilities の関数ブロック *(続き)*
  Catch Error 156
  Catch Error Type 156
  Condition 156
  Loop 156
  Move Attribute in Child 156
  Raise Error 156
  Raise Error Type 156
Utilities/Vector の関数ブロック 157
  Add Element 157
  Get Element 157
  Iterate Vector 157
  New Vector 157
  Size 157
  To Array 157

V

validData() メソッド 377
Vector クラス 330
  executeStoredProcedure() 233, 399, 410
  nextRow() 225, 404

W

Web サービス
  マップでの使用 181
  マップでの使用例 181
  Activity Editor での使用 181
  Activity Editor へのエクスポート 180

[特殊文字]

.bo ファイル拡張子 14, 96, 98, 102
.class ファイル拡張子 14, 91
.cwm ファイル拡張子 14, 57, 63, 64
.jar ファイル拡張子 88
.java ファイル拡張子 14, 91, 92
.txt ファイル拡張子 14, 537