IBM Security Directory Integrator
バージョン 7.2.0.1

リファレンス・ガイド

SC88-8414-03
(英文原典：SC27-2707-03)

IBM Security Directory Integrator
バージョン 7.2.0.1

リファレンス・ガイド

SC88-8414-03
(英文原典：SC27-2707-03)

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、 823 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。

注: 本書は、*IBM Security Directory Integrator* ライセンス・プログラムのバージョン 7.2.0.1 (5724-K74)、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。

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： SC27-2707-03
IBM Security Directory Integrator
Reference Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

本書について xv
資料および用語集へのアクセス xv
アクセシビリティー xvii
技術研修. xviii
サポート情報 xviii
適切なセキュリティーの実践に関する注意事項 xviii

第 1 章 概要. 1

第 2 章 コネクター 3
コネクター・インターフェース 3
サポートされるモードの欄の凡例 7
コネクターの再利用 7
Active Directory 変更検出コネクター 9
Active Directory 内での変更の追跡 9
Active Directory 内で削除されたオブジェクト 10
Active Directory 内で移動されたオブジェクト 11
objectGUID をオブジェクト ID として使用. . 11
変更検出 11
オフラインおよびページングされた結果の事例 12
Active Directory 変更検出コネクターの使用. . . 13
ディレクトリーに対するコネクターの認証 . . 13
エラー・フロー 14
構成. 15
AssemblyLine コネクター. 17
構成. 18
コネクターの使用 19
属性マッピング (スキーマ) とモード. . . . 19
AssemblyLine パラメーター 22
Axis Easy Web Service サーバー・コネクター. . . 22
WSDL ファイルのホスティング 24
スキーマ 25
構成. 26
コネクターの操作 28
Axis2 Web サービス・サーバー・コネクター . . . 29
Axis1 と Axis2 のコンポーネントの比較 . . . 29
SOAP エンコード方式のサポート 30
RPC/エンコード・モデルの普及度 30
コネクターの使用 31
サポートされているメッセージ交換パターン 32
SOAP 障害 33
SOAP ヘッダー 33
HTTP トランスポート層 33
WSDL の生成 34
スキーマ 34
構成. 39
セキュリティーと認証. 41
暗号化 41
認証. 41
許可. 42
CCMDB コネクター 42
CCMDB コネクターのアーキテクチャー. . . . 43
データ表現モード 44
IdML モード 44
ネイティブ・モード 44
スキーマの比較 44
CCMDB コネクターの動作モード 47
AddOnly モード 47
更新モード 48
削除モード 48
イテレーター・モード. 48
ルックアップ・モード. 48
構成. 49
例 49
コマンド行コネクター. 50
一部のオペレーティング・システムでのネイティブ・エンコード出力 50
引用符で囲む場合のワード 51
構成. 51
例 51
データベース・コネクター 52
構成. 52
導入済み資産コネクター 53
デプロイ済み資産コネクターの使用 54
デプロイ済み資産コネクターのアーキテクチャー 54
デプロイ済み資産コネクターの動作モード . . 55
AddOnly モード 55
イテレーター・モード. 55
ルックアップ・モード. 55
削除モード 55
構成. 55
例 56
TCP/URL の直接スクリプト記述 56
TCP 56
URL 57
Domino/Lotus Notes コネクター. 57
セッション・タイプ 57
インストール後の構成. 61
ローカル・クライアント・セッションの作成 61
ローカル・サーバー・セッションの作成. . . 62
IIOP セッションの作成 62
ネイティブ API 呼び出しのスレッド化 63
ncso.jar ファイル 63
サーバー側 64
「-v」コマンド行オプション. 64
サーバー API getServerInfo メソッド . . . 64
AssemblyLine の実行、IIOP セッション . . . 64
コンポーネントの可用性の報告. 65
コンポーネントのバージョン表. 65
コンポーネント・コンボ・ボックス . . . 65

「入力マップ」からのデータ・ソースへの接続 65
「classes」フォルダー 65
Domino 変更検出コネクター 66
コネクターの使用 67
文書の識別 67
削除された文書 67
同期の最小間隔 67
データベースのレプリカへの切り替え . . 68
コネクターから戻される項目の構造 . . . 68
$$UNID および $$NoteID 属性 68
$$ChangeType 属性 69
同期状態の値 69
コネクターの同期状態へのアクセス . . . 70
項目のフィルター処理 70
ソート 70
IBM Domino Server のシステム時刻を使用 71
大規模な Domino データベース (.nsf ファイル) の処理 71
API 71
IBM Security Directory Integrator に必要なセットアップ 72
IBM Domino に必要なセットアップ 72
Domino Server で必要なタスク 72
必須特権 72
構成 73
Domino 変更検出コネクターのトラブルシューティング 75
互換性 78
Lotus Domino ユーザー・コネクター 78
Domino Server への展開と接続 80
構成 80
旧バージョンからのパラメーター・マイグレーション 82
セキュリティー 82
Domino Server とクライアントの間での暗号化の構成 82
認証 83
許可 84
Domino ユーザー・コネクターの使用 84
イテレーター・モード 84
ルックアップ・モード 85
AddOnly モード 85
更新モード 88
削除モード 90
Domino ユーザー属性 (ユーザー文書項目) のリスト 93
IBM Domino Server (Unix/Linux) 95
例 96
Lotus Domino AdminP コネクター 96
管理要求への署名 96
スキーマ 97
構成 98
Lotus Notes コネクター 100
既知の制限事項 100
セッション・タイプ 101
IIOP を使用した接続 102
構成 102
UNID のサポート 104
RichText 属性のサポート 104
スクリプトの例 105
RichText の制限 105
割り当て量およびファイル所有権の設定 . . 106
セキュリティー 106
TIM DSMLv2 コネクター 107
更新モードおよび削除モードでのルックアップのスキップ 108
ITIM サーバーでのコネクターの使用方法 . . . 108
HTTPS (SSL) のサポート 108
構成 108
DSMLv2 SOAP コネクター 109
サポートされるコネクター・モード 110
拡張操作 111
SOAPAction ヘッダー 112
構成 112
DSMLv2 SOAP サーバー・コネクター 113
拡張操作 114
構成 114
EIF コネクター 116
IBM Tivoli Netcool/OMNIbus の概要 116
Tivoli Enterprise Console の概要 117
Event Integration Facility (EIF) の概要 117
スキーマ 118
イテレーター・モード 118
AddOnly モード 118
構成 118
ファイル・コネクター 119
構成 119
ファイル管理コネクター 120
コネクターの使用 121
ディレクトリー構造のトラバース 121
シンボリック・リンク 123
リンク基準での fullPath の指定 123
ファイルまたはディレクトリーの更新 . . . 124
ファイルおよびディレクトリーの強制削除 125
空のファイルおよびディレクトリーの作成 125
スキーマ 126
構成 127
例 128
フォーム入力コネクター 128
コネクターの使用 129
構成 129
FTP クライアント・コネクター 130
SSL サポート 130
文字エンコード 132
構成 132
Generic Log Adapter コネクター 133
アダプター構成ファイル 134
構成ファイルでの複数のアウトプッターの使用 134
構成 135
TDIOutputter の構成 135
コネクターの使用 136

スキーマ 136
HTTP クライアント・コネクター 137
モード 137
ルックアップ・モード 138
特殊属性 138
文字エンコード 140
構成 140
例 141
HTTP サーバー・コネクター 142
コネクターの構造とワークフロー 143
コネクター・クライアントの認証 143
チャンク転送エンコード 144
構成 144
コネクター・スキーマ 146
IBM Security Access Manager コネクター 147
コネクター・モード 148
更新モードおよび削除モードでのルックアップのスキップ 148
構成 148
IBM Security Access Manager の Java ランタイムの構成 148
IBM Security Access Manager ポリシー・サーバーへのセキュア通信の構成 149
SSL の構成 149
コネクターの構成 150
コネクターの使用 152
AddOnly モード 153
更新モード 155
削除モード 157
ルックアップ・モード 158
イテレーター・モード 159
トラブルシューティング 160
コネクター入力属性の詳細 160
ユーザー 160
グループ 162
ポリシー 163
ドメイン 165
SSO Credentials 165
SSO Resource 166
SSO Resource Group 166
IBM Security Access Manager v2 コネクター . . . 166
Registry Direct API のデプロイ 167
構成 169
パラメーター 169
属性マップ 169
トラブルシューティング 171
IBM Security Directory Integrator 変更ログ・コネクター 173
属性のマージ動作 173
分散 TDS と z/OS TDS での変更ログの違い 174
構成 175
ITIM Agent コネクター 178
ITIM Agent コネクターのための SSL のセットアップ 179
構成 179
既知の問題 180
IBM MQ コネクター 181
JDBC コネクター 181
コネクターの構造とワークフロー 181
JDBC ドライバーについて 182
DB2 への接続 183
Informix Dynamic Server への接続 186
Oracle への接続 186
SQL Server への接続 188
Sybase Adaptive Server への接続 188
Derby への接続 189
IBM solidDB への接続 189
ODBC データベース・パスの指定 190
スキーマ 190
構成 191
リンク基準の構成 195
更新または削除モードでのルックアップのスキップ 195
SELECT、INSERT、UPDATE、および DELETE ステートメントのカスタマイズ 196
メタデータ・オブジェクト 196
リンク・オブジェクト (リンク基準) 197
便利なオブジェクト 197
準備済みステートメントをオフにするオプション 198
カスタムの準備済みステートメント 198
追加の JDBC コネクター関数 200
パラメーター置換を使用不可または使用可能にする API 201
準備済みステートメントの仕様を可能にする API 202
タイム・スタンプ 202
埋め込み 203
ストアード・プロシージャーの呼び出し . . . 203
SQL データベース: 特殊文字を含む列名 . . . 204
準備済みステートメントの使用 204
複数項目時 204
追加の組み込み再接続ルール 205
JMS コネクター 205
JMS メッセージ・フロー 206
IBM WebSphere MQ および JMS/非 JMS メッセージ・コンシューマー 207
JMS メッセージのタイプ 207
テキスト・メッセージ 208
オブジェクト・メッセージ 209
バイト・メッセージ 209
イテレーター・モード 210
ルックアップ・モード 210
AddOnly モード 211
Call/Reply モード 211
JMS のヘッダーとプロパティー 211
構成 213
例 217
外部システム構成 217
トラブルシューティング 223
JMS パスワード・ストア・コネクター 224
コネクターのワークフロー 224

累積メッセージの強制転送 226
メッセージ・セキュリティー 226
PKCS7 暗号化のサポート 227
メッセージの署名 227
メッセージの暗号化 228
証明書の管理 228
証明書の構造 228
証明書の作成 228
例の使用法 229
スキーマ 231
構成 231
JMS ドライバー 233
IBM WebSphere MQ Everyplace ドライバー 234
IBM WebSphere MQ ドライバー 234
JMX コネクター 235
コネクター・スキーマ 235
構成 236
JNDI コネクター 238
構成 238
変更操作の設定. 240
変更インターフェースの呼び出し. 241
属性への値の追加 241
属性値の置換 242
属性の除去 242
属性からの特定の属性値の除去 242
変更操作 242
更新モードおよび削除モードでのルックアップのスキップ 243
LDAP コネクター 243
modrdn 操作の検出と処理 245
構成 245
仮想リスト・ビュー制御. 250
LDAP コネクターでのメモリーの問題の取り扱い 250
再接続機能の組み込みルール 251
z/OS における SDBM バックエンドの検索 . . 251
LDAP コネクターのメソッド (API) 252
LDAP の比較 252
属性への値の追加 252
属性値の置換 253
属性値の除去 253
すべての属性値の除去 253
属性の追加または置換のデフォルト・アクション用の構成エディターのフラグ. 254
再バインド 254
更新モードおよび削除モードでのルックアップのスキップ 254
LDAP グループ・メンバー・コネクター 255
大きな Active Directory グループの処理 . . . 255
LDAP グループ項目 256
データ・ソース・スキーマ 256
構成 257
LDAP サーバー・コネクター 257
スクリプト記述. 258
LDAP メッセージ戻り値の戻し 258
エラー処理 259
構成 259
ログ・コネクター 260
スキーマ 260
構成 260
ロガー構成画面. 261
Apache Log4J ロガー. 261
Java Util ロガー 267
JLOG ロガー 268
Lotus Notes コネクター 269
メールボックス・コネクター 269
スキーマ 270
コネクターの使用 272
イテレーター・モード 272
ルックアップ・モード 273
削除モード 273
AddOnly モード 274
更新モード 274
構成 275
メモリー・キュー・コネクター 276
メモリー・キューのコンポーネント 277
ワークフローの概要 277
構成 277
メモリー・キューへのプログラマチックなアクセス 278
メモリー・ストリーム・コネクター 279
構成 280
プロパティー・コネクター 280
構成 281
コネクターの使用 282
プロパティー・ファイルのフォーマット . . . 283
QRadar コネクター 284
QRadar コネクター・パラメーター 285
QRadar コネクターのセットアップ 287
LEEF スキーマへの入力データのマッピング . . 289
QRadar ログ・ソースのセットアップ 291
ソリューションの検証 292
RAC コネクター 293
構成 295
コネクターの使用 296
AddOnly モード 296
イテレーター・モード 297
スキーマ 298
RDBMS 変更検出コネクター 298
構成 299
変更表の形式 301
DB2 での変更表の作成 302
Oracle での変更表の作成 302
変更表とトリガーの作成 (MS SQL) 303
Informix での変更表の作成とトリガー 305
変更表とトリガーの作成 (SYBASE) 306
例 308
SCIM コネクター 308
構成 308
スクリプト・コネクター. 310
定義済みスクリプト・オブジェクト 310
関数 311

構成 313
例 314
サーバー通知コネクター 314
暗号化 315
認証 315
SSL 認証 315
ユーザー名とパスワードによる認証 316
構成 316
スキーマ 318
シンプル Tpae IF コネクター 319
共通基盤 320
統合フレームワーク 321
Maximo ビジネス・オブジェクト 322
MIF オブジェクト構造 323
コネクターの使用 324
オブジェクト構造サービスの使用 324
エンタープライズ・サービスの使用 325
MBO パラメーター 325
コネクター・モード 326
イテレーター・モード 326
AddOnly モード 329
更新モード 330
削除モード 330
ルックアップ・モード 331
スキーマ 331
エラー処理 332
外部システムの構成 332
構成 333
例 337
SNMP コネクター 337
構成 338
例 339
SNMP サーバー・コネクター 339
コネクター・スキーマ 339
構成 340
Sun Directory 変更検出コネクター 341
属性のマージ動作 342
構成 342
システム・キュー・コネクター 346
構成 346
セキュリティー、認証、および許可 348
暗号化 348
認証 348
ユーザー名とパスワードによる認証 . . . 348
SSL 証明書ベースの認証 348
許可 349
システム・ストア・コネクター 349
構成 351
コネクターの使用 353
TADDM 変更検出コネクター 354
TADDM 変更検出 355
デルタ・タグ・サポート 355
TADDM 変更検出コネクターのデータ・ソース・スキーマ 356
構成 357
TADDM コネクター 360
TADDM のデータ表現モデル 360
Common Data Model 361
データ表現フォーマット 362
TADDM モデル 362
IT レジストリー・モデル 362
IdML ブックのフォーマット 363
TADDM での統一スキーマの実装 363
明示属性名とクラス・タイプ 363
暗黙属性 363
識別可能なデータ 363
データ表現モード 364
IdML モード 364
ネイティブ・モード 365
スキーマの比較 365
システム属性 366
コネクターの使用 366
基本構成 366
MSS サポート 367
TADDM の照会 368
追加の属性の取得 369
特定のモデル・オブジェクトの検索 370
TADDM からの構成アイテムと関係の読み取り 371
モデル・オブジェクトの削除 371
新規モデル・オブジェクトの作成 372
既存のモデル・オブジェクトの更新 373
デルタ・モードのサポート 374
TADDM コネクターのデータ・ソース・スキーマ 375
インストール後の作業 376
TADDM コネクターのトラブルシューティング 376
TADDM コネクターの使用時に AccessException がスローされる 377
IdML モードで TADDM からデータを読み取る際に例外がスローされる 377
構成 378
TCP コネクター 380
イテレーター・モード 380
AddOnly モード 381
構成 381
TCP サーバー・コネクター 382
構成 382
コネクター・スキーマ 382
タイマー・コネクター 384
構成 384
Tpae IF 変更検出コネクター 384
Tpae IF 変更検出コネクターのアーキテクチャー 385
デルタ・タグ 387
Maximo サーバーの構成 389
HTTP エンドポイントの作成 389
HTTP エンドポイントの割り当て 389
イベント・リスナーの有効化 390
クーロン・タスクの構成 390
構成 391
例 392
Tpae IF コネクター 393

コネクターの使用 393
イテレーター・モード 394
AddOnly モード、更新モード、削除モード 397
ルックアップ・モード 399
エラー処理 400
外部システム構成 401
構成 402
例 405
URL コネクター 406
構成 406
サポートされる URL プロトコル. 406
Web サービス受信側サーバー・コネクター . . . 407
WSDL ファイルのホスティング 407
スキーマ 408
構成 409
コネクターの操作 411
Windows ユーザー/グループ・コネクター 412
前提条件 412
構成 413
リンク基準の構成 413
その他. 414
ユーザーおよびグループのアカウント名 414
新規ユーザーの作成 415
ユーザー・パスワードの設定 415
ユーザー 1 次グループ/グローバル・グループのメンバーシップの設定 415
グループの操作. 415
文字セット 416
例 416
Windows ユーザー/グループ・コネクターの機能仕様とソフトウェア要件. 416
ユーザーおよびグループ・データの抽出 . . 417
ユーザーおよびグループ・データの追加 . . 417
グループ・メンバーシップの変更. 417
ユーザーおよびグループ・データの変更 . . 418
ユーザーおよびグループ・データの削除 . . 418
z/OS LDAP 変更ログ・コネクター 418
属性のマージ動作 419
構成 419
関連情報 421

第 3 章 パーサー 423

基本パーサー 423
文字エンコード変換 423
CBE パーサー 425
パーサーの使用. 425
CBE 入力マップおよび出力マップの属性 . . . 426
構成 428
CSV パーサー 429
構成 429
スキーマ 430
DSMLv1 パーサー. 431
構成 431
例 431
DSMLv2 パーサー. 432
モード. 432
操作 433
変更要求 433
変更応答 433
検索要求 434
検索応答 434
追加要求 435
追加応答 435
削除要求 436
削除応答 436
ModifyDN 要求. 436
ModifyDN 応答. 437
比較要求 437
比較応答 437
認証要求 438
認証応答 438
拡張要求 439
拡張応答 439
エラー応答 439
バイナリー属性とストリング以外の属性 . . . 440
オプション属性. 440
Base64 エンコードであると想定される DSMLv2 コントロール 440
結果コードおよび結果記述の設定. 441
複数の属性の変更 441
構成 442
例 443
サーバー・モードでの DSMLv2 AddRequest の構文解析 443
クライアント・モードでの DSMLv2 SearchRequest の作成 444
固定レコード・パーサー. 445
JSON パーサー. 445
パーサーの使用. 445
例 446
構成 449
HTTP パーサー. 449
構成 449
スキーマ 450
一般ヘッダーのフィールド 452
エンティティー・ヘッダーのフィールド . . 452
要求ヘッダーのフィールド 453
応答ヘッダーのフィールド 455
文字セット/エンコード 456
HTTP Cookie を使用する方法 457
LDIF パーサー 457
LDIF 入力の読み取り. 458
LDIF 出力の書き込み. 458
構成 459
行リーダー・パーサー 461
構成 461
スクリプト・パーサー 461
オブジェクト 462
結果オブジェクト 462
項目オブジェクト 462
inp オブジェクト 462
out オブジェクト 463

パーサー・オブジェクト 463
コネクター・オブジェクト 463
関数 (メソッド) 463
構成 464
スキーマ 464
例 465
単純なパーサー 465
構成 465
SOAP パーサー 466
項目の例 466
SOAP 文書の例 466
構成 467
パーサー固有の呼び出し 467
例 467
SPMLv2 パーサー 468
操作 468
追加要求 469
追加応答 469
変更要求 469
変更応答 470
削除要求 470
削除応答 471
ルックアップ要求 471
ルックアップ応答 471
検索要求 472
検索応答 472
バイナリー属性とストリング以外の属性 . . 473
属性の操作のタグ付け 473
検索フィルターの機能 474
構成 475
例 476
単純 XML パーサー 476
構成 477
単純 XML パーサーにおける文字エンコード 478
例 479
XML パーサー 481
構成 481
パーサーの使用 483
XML 構造内のナビゲーション 483
読み取り時のナビゲーション 483
書き込み時のナビゲーション 486
XML の読み取り 488
XML の書き込み 489
XML パーサーにおける文字エンコード . . . 489
例 491
XSD スキーマの使用 491
定義済み XSD スキーマ URI 491
XSD が提供されていない 491
スキーマの構成 491
XSD スキーマの例 493
XML SAX パーサー 493
構成 495
文字エンコード 496
XSL ベース XML パーサー 496
構成 496
パーサーの使用 498
IBM Security Directory Integrator 内部フォーマット 498
例 498
ユーザー定義パーサー 500

第 4 章 関数コンポーネント 501

Castor Java to XML 関数コンポーネント 502
構成 503
関数コンポーネントの使用 503
Castor XML to Java 関数コンポーネント 504
構成 505
関数コンポーネントの使用 506
XMLToSDO 関数コンポーネント 507
例 507
構成 508
マイグレーション 509
SDOToXML 関数コンポーネント 510
構成 511
関数コンポーネントの使用 512
マイグレーション 512
AssemblyLine 関数コンポーネント 513
構成 513
関数コンポーネントの使用 514
Java クラス関数コンポーネント 517
スキーマ 517
構成 518
パーサー関数コンポーネント 518
構成 518
関数コンポーネントの使用 519
スクリプト記述関数コンポーネント 519
構成 520
関数コンポーネントの使用 520
オブジェクト 520
CBE 関数コンポーネント 521
CBE (Common Base Event) 522
CEI (Common Event Infrastructure) 522
入力属性および出力属性 522
構成 522
CBE ログ XML の生成 523
CEI サーバーへのイベントの発行 524
関数コンポーネント API 524
SendEMail 関数コンポーネント 525
スキーマ 526
構成 527
メモリー・キュー関数コンポーネント 528
構成 529
関数コンポーネントの使用 529
Axis Java から Soap への変換関数コンポーネント 530
構成 530
関数コンポーネントの入力 531
関数コンポーネントの出力 531
関数コンポーネントの使用 531
カスタム・シリアライザー/デシリアライザー 533
シリアライゼーション/デシリアライゼーションの問題 533
WrapSoap 関数コンポーネント 534

構成 535
関数コンポーネントの入力 536
関数コンポーネントの出力 536
関数コンポーネントの使用 536
InvokeSoap WS 関数コンポーネント 537
認証 537
構成 537
関数コンポーネントの入力 538
関数コンポーネントの出力 539
関数コンポーネントの使用 539
Axis Soap から Java への変換関数コンポーネント 541
構成 541
関数コンポーネントの入力 542
関数コンポーネントの出力 542
関数コンポーネントの使用 542
Axis2 WS クライアント関数コンポーネント . . . 543
関数コンポーネントの使用 544
サポートされているメッセージ交換パターン . . 544
SOAP ヘッダー 545
スキーマ 545
構成 548
Axis EasyInvoke Soap WS 関数コンポーネント . . 549
構成 549
セキュリティーと認証 550
関数コンポーネントの入力 551
関数コンポーネントの出力 551
関数コンポーネントの使用 551
複素数タイプ・ジェネレーター関数コンポーネント 552
構成 553
関数コンポーネントの入出力 554
トラブルシューティング 554
デルタ関数コンポーネント 554
構成 554
関数コンポーネントの使用 556
例 557
リモート・コマンド行関数コンポーネント 557
構成 558
関数コンポーネントの入力 560
関数コンポーネントの出力 560
関数コンポーネントの使用 561
z/OS TSO/E コマンド行関数コンポーネント . . . 564
構成 564
パラメーター 564
関数コンポーネントの使用 565
エラー・フロー 565
関数コンポーネントの入力 566
関数コンポーネントの出力 566
認証 566
許可 567
必須の仮名ファイル 567
関数コンポーネントのネイティブ部分のセットアップ 571
ファイル転送関数コンポーネント 573
ファイル転送関数コンポーネントのアーキテクチャー 574
ファイル転送関数コンポーネントのデータ・ソース・スキーマ 574
ファイル転送の方向 575
ターゲット・システムの構成 576
Windows システム 576
Cygwin システム 577
UNIX および Linux システム 577
AS400 システム 578
構成パラメーター 578
ソース・オプション 578
拡張ソース・オプション 579
ターゲット・オプション 580
拡張ターゲット・オプション 580
拡張オプション 581

第 5 章 SAP ABAP Application Server Component Suite 583

Component Suite のインストール 583
ソフトウェア要件 583
SAP Java Connector の構成 584
Component Suite for SAP ABAP Application Server の検証 585
バージョン番号の確認 586
アンインストール 586
SAP ABAP Application Server の関数コンポーネント 587
構成 588
パラメーター 588
関数コンポーネントの入力 589
関数コンポーネントの出力 590
関数コンポーネントの使用 591
コマンド行 RFC Invoker の使用 591
SAP ABAP Application Server ユーザー・レジストリー・コネクター 592
更新モードおよび削除モードでのルックアップのスキップ 593
構成 594
パラメーター 594
SAP ABAP Application Server ユーザー・レジストリー・コネクターの使用 597
IBM Security Directory Integrator 項目スキーマ 598
AddOnly モード 598
更新モード 599
削除モード 599
ルックアップ・モード 600
イテレーター・モード 600
サポートされないトランザクション操作 . . 601
ABAP エラーの処理 602
SAP ABAP Application Server Business Object Repository コネクター 602
更新モードおよび削除モードでのルックアップのスキップ 606
構成 606
SAP ABAP Application Server Business Object Repository コネクターの使用 609

IBM Security Directory Integrator 項目スキーマ 610
AddOnly モード 611
更新モード 612
削除モード 613
ルックアップ・モード 613
イテレーター・モード 614
サポートされないトランザクション操作 . . 614
ABAP エラーの処理 614
ALE Intermediate Document (IDOC) Connector for SAP ABAP Application Server および SAP ERP . . 615
インストール 617
構成 617
IDOC Server パラメーター 617
IDOC Client の構成パラメーター 617
一般構成パラメーター 619
SAP ALE IDOC コネクターの使用 620
IBM Security Directory Integrator スキーマ 620
XML 属性の構文解析. 623
SAP ALE 分散モデルでの構成 627
SAP ABAP Application Server Component Suite のトラブルシューティング. 628
SAP ABAP Application Server Component Suite の補足情報 630
ユーザー・レジストリー・コネクター XML インスタンス文書の例 630
ユーザー・レジストリー・コネクター XML の XSchema 632

第 6 章 資産統合スイート 641
CDM コンポーネント 641
属性 641
クラス. 642
インターフェース 642
関係 643
命名および識別. 643
IT レジストリー 644
スイートのコンポーネント 645
IdML のオープン関数コンポーネント 645
スキーマ 649
構成 649
IdML のクローズ関数コンポーネント 651
スキーマ 651
構成 651
IdML の分割関数コンポーネント 651
スキーマ 652
構成 653
IdML CI および関係コネクター 653
スキーマ 655
構成 656
IdML パーサー 657
スキーマ 657
構成 658
データ・クレンジング関数コンポーネント. . . 658
スキーマ 659
構成 659
IT レジストリーの初期化関数コンポーネント 659
スキーマ 660
構成 661
IT レジストリー CI および関係コネクター . . 662
設計時の命名規則の妥当性検査 665
スキーマ 666
構成 667
it_registry.properties ファイル 668
例 669
IT レジストリー・データベースのセットアップ 670
トラブルシューティング. 672

第 7 章 スクリプト言語 675
JavaScript 675
Java と JavaScript 675

第 8 章 オブジェクト 677
AssemblyLine コネクター・オブジェクト 677
属性オブジェクト 677
例 678
新しい属性オブジェクトの作成 678
属性への値の追加 678
属性の値のスキャン 678
コネクター・インターフェース・オブジェクト . . 679
項目オブジェクト 679
FTP オブジェクト. 681
例 681
メイン・オブジェクト 681
検索 (基準) オブジェクト 682
オペランド 682
例 683
shellCommand オブジェクト 683
状況オブジェクト 683
システム・オブジェクト. 683
タスク・オブジェクト 683
COMProxy オブジェクト 684
コード例 684

第 9 章 IBM Security Directory Integrator スケジューラー 687
構成 687
タイマー 687
キープアライブ. 688

付録 A. AssemblyLine シーケンス . . 691
構成 691

付録 B. パスワード同期プラグイン. . . 693

付録 C. AssemblyLine フロー・チャート 695

付録 D. サーバー API 697
ローカル・サーバーおよびリモート・サーバーの API インターフェース 699
サーバー API 構造 699

セキュリティー. 700
サーバー API の構成. 702
サーバー API プロパティーの構成 702
ユーザー・レジストリーのセットアップ . . . 702
リモート・クライアント構成 702
サーバー API の使用. 704
ローカル・セッションの作成 704
リモート・セッションの作成 705
構成インスタンスの処理. 705
実行中の構成インスタンスへのアクセス . . 705
構成インスタンスの開始. 706
構成インスタンスの停止. 706
サーバー API と構成の初期設定の同期化 . . 707
構成ファイル内の任意指定の構成インスタンス ID 707
AssemblyLine の処理 710
構成内で使用可能な AssemblyLine へのアクセス 710
実行中の AssemblyLine へのアクセス . . . 711
AssemblyLine の開始 712
手動モードでの AssemblyLine の開始 . . . 712
リスナーを使用した AssemblyLine の開始 712
コンポーネント・シミュレーションによる AssemblyLine の開始 714
AssemblyLine の停止 714
構成の編集 714
IBM Security Directory Integrator 構成フォルダー 714
編集のためのロード 715
構成のロック 715
一時構成インスタンスを使用した編集のためのロード 718
構成ファイル・パスの代わりにソリューション名を使用する. 718
構成更新に関するサーバー API イベント . . 719
システム・キューの処理. 719
サーバー API を介したシステム・キューへのアクセス 721
システム・キューへのメッセージの入力 . . 721
システム・キューからのメッセージの取得 721
トゥームストーン・マネージャーの処理 . . . 721
グローバル一意識別子 (GUID). 721
サーバー API によるトゥームストーン・マネージャーのサポート 722
AssemblyLine トゥームストーンへのカスタム・メッセージの追加 724
IBM Security Directory Integrator プロパティーの処理 725
サーバー API イベント通知のための登録 . . . 726
サーバー・シャットダウン・イベント . . . 728
サーバー API カスタム・イベント通知. . . 728
ログ・ファイルへのアクセス 729
サーバー情報 730
セキュリティー・レジストリーの使用 731
カスタム・メソッドの起動 731
JMX レイヤー 733
JMX レイヤーへのローカル・アクセス 734
JMX レイヤーへのリモート・アクセス 734
MBean およびサーバー API オブジェクト. . . 735
JMX 通知 735
JMX の例 - IBM Security Directory Integrator および MC4J 構成 736
IBM Security Directory Integrator 側 736
リモート・サーバー API および JMX のセットアップ 736
コマンド行からの IBM Security Directory Integrator サーバーの始動 737
MC4J 側 737
以前のバージョンとの互換性 739
シナリオ概要 739
現行バージョンのサーバーを使用する目的での V6.0 サーバー API クライアントの移植に関するガイドライン 740
v6.0 サーバーと現行バージョン・サーバーの両方を使用できるサーバー API クライアントのインプリメントに関するガイドライン . . 741
RMI リモート・オブジェクトの使用. . . 741
シリアライズ可能クラスの使用 742
構成の編集 743
認証メカニズム. 743
IBM Security Directory Integrator サーバーのバージョンの確認 743
サーバー API の変更内容 744
既知の問題 748

付録 E. REST サーバー API 751

REST サーバー API のアーキテクチャー 752
リソース階層のナビゲーション 752
アルゴリズムの例 754
サーバー・フィード 755
サーバー情報項目 755
サーバー制御項目 756
カスタム通知項目 756
構成フィード 757
構成ディレクトリー項目. 757
構成ファイル項目 758
ConfigInstance フィード 759
サーバー・リスナー・フィード 767
トゥームストーン・フィード 768
リスナー・トランスポート・チャネル 769
スキーマ 771
内容定義 771
JSON を使用するポリモアフィズム 772
外部システム構成 773

付録 F. アダプターを使用した新規コンポーネントの作成 775

IBM Security Directory Integrator アダプターのインプリメンテーションを可能にする機能 776
AL の操作 776
スイッチ/ケース・コンポーネント 777
コネクターの柔軟な初期化 777

フローでのイテレーターの使用 778
消費のためのアダプターのパブリッシュ . . . 778
AssemblyLine でのアダプターの使用. 779
IBM Security Directory Integrator アダプターでの操作の使用 779
アダプター操作のコネクター・モードへのマッピング 780
アダプターにおける各操作のためのコードのインプリメント 780
$initialization 操作を使用したアダプター構成 781
リンク基準について 781
属性マッピング. 781
状況の表示 783
照会スキーマのインプリメント 783
デルタ・モード. 783
エラー処理 784

付録 G. Java での独自コンポーネントのインプリメント 785

コンポーネント開発のサポート資料 785
コネクターの開発 785
コネクターの Java ソース・コードのインプリメント 785
コネクターでのパーサーの使用 792
コネクターからのロギング 794
コネクターのソース・コードの作成 795
コネクターの GUI 構成フォームのインプリメント 795
tdi.xml ファイルのフォーマット 796
基本コンポーネントの定義 796
インストール・ロケーション 797
フォーム記述 798
コンポーネント/フォームの関連性 . . . 798
フォーム/構成のバインディング 798
フォーム定義 798
フォームのスクリプト 805
例 805
コネクターの再接続ルールの定義. 806
コネクターのパッケージ化とデプロイ 807
関数コンポーネントの開発 807
関数コンポーネントの Java ソース・コードのインプリメント 807
関数コンポーネントのソース・コードの作成 . . 809
関数コンポーネントの GUI 構成フォームのインプリメント 809
関数コンポーネントのパッケージ化とデプロイ 809
パーサーの開発. 810
パーサーの Java ソース・コードのインプリメント 810
パーサーのソース・コードの作成. 812
パーサーの GUI 構成フォームのインプリメント 813
パーサーのパッケージ化とデプロイ 813
追加ロガーの作成 813
ロギング・インターフェースの理解 814
ログ・インターフェースの構成 815
ロガーの外部構成 816
ロガーの内部構成 816
ロガー API 817
com.ibm.di.server.Log 818
com.ibm.di.log.LogInterface 818
com.ibm.di.server.Log 820

特記事項. 823

索引 827

本書について

この資料には、IBM® Security Directory Integrator に含まれるコンポーネントを使用したソリューション開発に必要な情報が記載されています。

IBM Security Directory Integrator の各コンポーネントは、ユーザー・ディレクトリーおよびその他のリソースの管理を担当するネットワーク管理者用に設計されています。IBM Security Directory Integrator と IBM Security Directory Server の両方のインストールおよび使用に関する実務経験を持っていることを想定しています。

本書は、IBM Security Directory Integrator を使用したソリューションの開発、インストール、および管理を担当するユーザーも対象としています。読者は、開発したソリューションの接続先となるシステムの概念や管理方法について習熟している必要があります。そのようなシステムには、ソリューションに応じて、以下の製品、システム、概念のうち 1 つ以上が含まれます (これらに限定されてはいません)。

- IBM Security Directory Server
- IBM Security Identity Manager
- IBM Java™ ランタイム環境 (JRE) または Oracle Java ランタイム環境
- Microsoft Active Directory
- Windows および UNIX オペレーティング・システム
- セキュリティー管理
- Hypertext Transfer Protocol (HTTP)、HyperText Transfer Protocol Secure (HTTPS)、および Transmission Control Protocol/Internet Protocol (TCP/IP) を含むインターネット・プロトコル (IP)
- Lightweight Directory Access Protocol (LDAP) およびディレクトリー・サービス
- サポートされるユーザー・レジストリー
- 認証と許可の概念
- SAP ABAP アプリケーション・サーバー

資料および用語集へのアクセス

以下は英語のみの対応となります。オンラインでアクセス可能な IBM Security Directory Integrator バージョン 7.2.0.1 ライブラリーおよび関連資料の説明をお読みください。

このセクションの内容は以下のとおりです。

- 『IBM Security Directory Integratorライブラリー』内の資料のリスト
- xvii ページの『オンライン資料』へのリンク。
- xvii ページの『IBM Terminology Web サイト』へのリンク

IBM Security Directory Integratorライブラリー

IBM Security Directory Integrator ライブラリーでは以下の資料を入手できます。

- *IBM Security Directory Integrator バージョン 7.2.0.1 Federated Directory Server 管理ガイド*

  Federated Directory Server コンソールを使用して、データ統合ソリューションの設計、インプリメント、管理を行う方法について記載されています。また、System for Cross-Domain Identity Management (SCIM) プロトコルとインターフェースを使用して ID を管理する方法についても記載されています。
- *IBM Security Directory Integrator バージョン 7.2.0.1 スタートアップ・ガイド*

  IBM Security Directory Integrator の解説および概要です。対話の作成の例と、IBM Security Directory Integrator の実践学習を含んでいます。
- *IBM Security Directory Integrator バージョン 7.2.0.1 ユーザーズ・ガイド*

  IBM Security Directory Integrator の使用方法が記載されています。Security Directory Integrator デザイナー・ツール (構成エディター) を使用したソリューションの設計や、コマンド行からの既製ソリューションの実行について説明しています。また、インターフェース、概念、および AssemblyLine の作成に関する情報も記載されています。
- *IBM Security Directory Integrator バージョン 7.2.0.1 インストールおよび管理者ガイド*

  インストール、旧バージョンからのマイグレーション、ロギング機能の構成、および IBM Security Directory Integrator のリモート・サーバー API の基礎となるセキュリティー・モデルについて記載されています。ソリューションのデプロイおよび管理方法が含まれています。
- *IBM Security Directory Integrator バージョン 7.2.0.1リファレンス・ガイド*

  IBM Security Directory Integrator の個々のコンポーネント (コネクター、関数コンポーネント、パーサーなど、AssemblyLine のビルディング・ブロック) に関する詳細情報が記載されています。
- *IBM Security Directory Integrator バージョン 7.2.0.1 Problem Determination Guide*

  問題の識別および解決を支援する IBM Security Directory Integrator のツール、リソース、および技法に関する情報が記載されています。
- *IBM Security Directory Integrator バージョン 7.2.0.1 メッセージ・ガイド*

  IBM Security Directory Integrator に関連付けられたすべての情報、警告、およびエラー・メッセージのリストが記載されています。
- *IBM Security Directory Integrator バージョン 7.2.0.1 パスワード同期プラグイン*

  5 つの IBM パスワード同期プラグイン (Windows 用 Password Synchronizer、Sun Directory Server 用 Password Synchronizer、IBM Security Directory Server 用 Password Synchronizer、Domino® 用 Password Synchronizer、UNIX および Linux 用 Password Synchronizer) それぞれのインストールおよび構成について詳細に説明しています。また、LDAP パスワード・ストアと JMS パスワード・ストアの構成手順についても説明します。
- *IBM Security Directory Integrator バージョン 7.2.0.1 リリース・ノート*

IBM Security Directory Integrator に関する新規機能および最新情報で、本書に記載していないものを掲載しています。

オンライン資料

IBM では、製品のリリース時および資料の更新時に、以下の場所に製品資料を掲載しています。

IBM Security Directory Integrator ライブラリー
製品資料サイト (http://www-01.ibm.com/support/knowledgecenter/SSCQGF/welcome) には、ライブラリーのウェルカム・ページとナビゲーションが表示されます。

IBM Security Systems Documentation Central
IBM Security Systems Documentation Central には、すべての IBM Security Systems 製品ライブラリーのアルファベット順のリストと、各製品のそれぞれのバージョンのオンライン資料へのリンクが掲載されています。

IBM Publications Center
IBM Publications Center サイト (http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) には、必要なすべての IBM 資料を見つけるのに役立つカスタマイズ検索機能が用意されています。

関連情報

IBM Security Directory Integrator の関連情報は以下の場所で入手できます。

- IBM Security Directory Integrator では、Oracle の JNDI クライアントを使用しています。JNDI クライアントについては、「*Java Naming and Directory Interface™ Specification*」(http://download.oracle.com/javase/7/docs/technotes/guides/jndi/index.html) を参照してください。
- IBM Security Directory Integrator に関する疑問点を解決するために有用な情報が https://www-947.ibm.com/support/entry/myportal/over-accesspubsview/software/security_systems/tivoli_directory_integratorに記載されています。

IBM Terminology Web サイト

IBM Terminology Web サイトは、製品ライブラリーの用語を 1 つの場所にまとめたものです。Terminology Web サイトには、http://www.ibm.com/software/globalization/terminology からアクセスできます。

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。この製品では、インターフェースの読み上げおよびナビゲートを行うための支援技術を使用できます。また、マウスの代わりにキーボードを使用して、グラフィカル・ユーザー・インターフェースのすべての機能を操作できます。

詳しくは、「*Directory Integrator の構成*」のアクセシビリティーに関する付録を参照してください。

技術研修

以下は英語のみの対応となります。技術研修の情報については、IBM Education Web サイト (http://www.ibm.com/software/tivoli/education) を参照してください。

サポート情報

IBM サポートは、コード関連の問題、およびインストールまたは使用方法に関する短時間の定型質問に対する支援を提供します。IBM ソフトウェア・サポート・サイトには、http://www.ibm.com/software/support/probsub.html から直接アクセスできます。

トラブルシューティング では、以下が詳細に説明されています。

- IBM サポートに連絡する前に収集する情報。
- IBM サポートに連絡するためのさまざまな方法。
- IBM Support Assistant の使用方法。
- 問題を自分自身で特定して解決するための手順と問題判別のためのリソース。

適切なセキュリティーの実践に関する注意事項

IT システムのセキュリティーでは、企業内および企業外からの不適切なアクセスの防止、検出、およびそれらのアクセスへの対応により、システムおよび情報を保護する必要があります。不適切なアクセスにより、情報が改ざん、破壊、盗用、または悪用されたり、ご使用のシステムの損傷または他のシステムへの攻撃のための利用を含む悪用につながる可能性があります。完全に安全と見なすことができる IT システムまたは IT 製品は存在せず、また単一の製品、サービス、またはセキュリティー対策が、不適切な使用またはアクセスを防止する上で、完全に有効となることもありません。 IBM のシステム、製品およびサービスは、包括的なセキュリティーの取り組みの一部となるように設計されており、これらには必ず追加の運用手順が伴います。また、最高の効果を得るために、他のシステム、製品、またはサービスを必要とする場合があります。IBM は、システム、製品、またはサービスが、悪意のある行為または不正な行為から影響を受けないこと、またはこれらの行為がお客様の企業に影響を与えないことを保証しません。

第 1 章 概要

ここに示す手順を使用して、「IBM Security Directory Integrator バージョン 7.2 リファレンス・ガイド」にアクセスし、この資料を利用することができます。

本書の内容を補足する各種の例を利用するには、インストール・パッケージを参照して必要なファイルをダウンロードする必要があります。

これらのサンプル・ファイルは、インストール・ディレクトリー (通常は `TDI_install_dir/`) の `examples` ディレクトリーにあります。

IBM Security Directory Integrator に付属するサンプルは現状のままで提供され、正式な IBM サポートは付帯していません。

第 2 章 コネクター

IBM Security Directory Integrator に組み込まれているすべてのコネクター・インターフェースのリストを利用できます。コネクター・インターフェースは、コネクターの中で、そのコネクターが処理するデータ・ソースとの通信を行う実際のロジックをインプリメントしている部分です。

コネクターの可用性およびリファレンス

必要に応じて、ユーザー独自のコネクター・インターフェースも作成できます。AssemblyLine はこれらのコネクターをラップして、AssemblyLine コネクターとして使用できるようにします。

コネクターを AssemblyLine に正しくデプロイするには、コネクターをあらかじめ構成しておく必要があります。コネクターに設定されているモードに応じて、多数のコネクターにさまざまなパラメーター・セットが存在します。例えば、イテレーター・モードで有効なパラメーターは AddOnly モードでは必要でないためパラメーター・リストに表示されません。

以下に示すすべての AssemblyLine コネクターは、コネクター・インターフェースのメソッドおよびプロパティーに加えて、`com.ibm.di.server.`*`AssemblyLineComponent`* に記述されているメソッドにアクセスできます。メソッドの資料については、JavaDocs (CE から「**ヘルプ**」->「**ようこそ**」->「**JavaDocs**」を選択) を参照してください。

コネクター・インターフェース

ここでは、コネクターのリストとそれぞれに対応するリンクのリストを示します。

サポートされるモードのリストについては、7 ページの『サポートされるモードの欄の凡例』を参照してください。

以下に示す各コネクター・インターフェースについては、このセクションに示す概要説明を参照してください。

9 ページの『Active Directory 変更検出コネクター』
I

17 ページの『AssemblyLine コネクター』
I

22 ページの『Axis Easy Web Service サーバー・コネクター』
S

29 ページの『Axis2 Web サービス・サーバー・コネクター』
S

42 ページの『CCMDB コネクター』
A D I L U

50 ページの『コマンド行コネクター』
A I C

52 ページの『データベース・コネクター』
A D I L U Δ

53 ページの『導入済み資産コネクター』
A D I L

56 ページの『TCP/URL の直接スクリプト記述』
custom

96 ページの『Lotus Domino AdminP コネクター』
I A

66 ページの『Domino 変更検出コネクター』
I

78 ページの『Lotus Domino ユーザー・コネクター』
A D I L U

109 ページの『DSMLv2 SOAP コネクター』
A D I L U C Δ

113 ページの『DSMLv2 SOAP サーバー・コネクター』
S

116 ページの『EIF コネクター』
A I

119 ページの『ファイル・コネクター』
A I

120 ページの『ファイル管理コネクター』
A D I L U

128 ページの『フォーム入力コネクター』
I

130 ページの『FTP クライアント・コネクター』
A I

133 ページの『Generic Log Adapter コネクター』
I

137 ページの『HTTP クライアント・コネクター』
A I L C

142 ページの『HTTP サーバー・コネクター』
I S

181 ページの『IBM MQ コネクター』
A I L C

173 ページの『IBM Security Directory Integrator 変更ログ・コネクター』
I

653 ページの『IdML CI および関係コネクター』
C

662 ページの『IT レジストリー CI および関係コネクター』
I L C

178 ページの『ITIM Agent コネクター』
A D I L U

107 ページの『TIM DSMLv2 コネクター』
A D I L U

181 ページの『JDBC コネクター』
A D I L U Δ

205 ページの『JMS コネクター』
A I L C

224 ページの『JMS パスワード・ストア・コネクター』
I

235 ページの『JMX コネクター』
I

238 ページの『JNDI コネクター』
A D I L U Δ

243 ページの『LDAP コネクター』
A D I L U Δ

255 ページの『LDAP グループ・メンバー・コネクター』
I

257 ページの『LDAP サーバー・コネクター』
S

260 ページの『ログ・コネクター』
A

100 ページの『Lotus Notes コネクター』
A D I L U

269 ページの『メールボックス・コネクター』
A D I L U

276 ページの『メモリー・キュー・コネクター』
A I

279 ページの『メモリー・ストリーム・コネクター』
A I

280 ページの『プロパティー・コネクター』
A I U L D

293 ページの『RAC コネクター』
A I

298 ページの『RDBMS 変更検出コネクター』
I

602 ページの『SAP ABAP Application Server Business Object Repository コネクター』 A D I L U

592 ページの『SAP ABAP Application Server ユーザー・レジストリー・コネクター』 A D I L U

310 ページの『スクリプト・コネクター』

custom

スクリプト・コネクターはユーザー自身が作成するものであり、ユーザーがスクリプトに書き込んだモードを提供します。

314 ページの『サーバー通知コネクター』

A I

319 ページの『シンプル Tpae IF コネクター』

A D I L U

337 ページの『SNMP コネクター』

A I L

339 ページの『SNMP サーバー・コネクター』

S

341 ページの『Sun Directory 変更検出コネクター』

I

346 ページの『システム・キュー・コネクター』

A I

349 ページの『システム・ストア・コネクター』

A D I L U

354 ページの『TADDM 変更検出コネクター』

I

360 ページの『TADDM コネクター』

A D I L U Δ

380 ページの『TCP コネクター』

A I

382 ページの『TCP サーバー・コネクター』

I S

147 ページの『IBM Security Access Manager コネクター』

A I D L U

384 ページの『タイマー・コネクター』

I

384 ページの『Tpae IF 変更検出コネクター』

S

393 ページの『Tpae IF コネクター』

A D I L U

406 ページの『URL コネクター』

A I

407 ページの『Web サービス受信側サーバー・コネクター』

S

412 ページの『Windows ユーザー/グループ・コネクター』
A D I L U

418 ページの『z/OS LDAP 変更ログ・コネクター』
I

サポートされるモードの欄の凡例

サポートされるモードの欄の凡例リストをここで参照できます。

- A – AddOnly
- D – 削除
- I – イテレーター
- L – ルックアップ
- U – 更新
- Δ – デルタ
- C – Call/Reply
- S – サーバー
- +– 新しいバージョンのサポートが存在します。

コネクターの再利用

コネクターの再利用とその機能については、ここに記載されている情報を通じて詳しく知ることができます。

コネクターをインスタンス化すると、通常、一定量のリソースが特定のシステム (接続オブジェクト、セッション・オブジェクト、結果セットなど) との通信用に割り振られます。同じタイプの複数のコネクターを同一のシステムに接続するときは、基礎になるリソースを共用すると合理的です。つまり、特定のシステムとの単一の接続を複数のコネクターで再利用するようにします。

IBM Security Directory Integrator では、AssemblyLine 内でコネクターを再利用することができます。特定の AssemblyLine ごとに、既に構成されているコネクターを同一の AssemblyLine から再利用できます。

IBM Security Directory Integrator サーバーについては、コネクターを再利用する際、1 つの物理コネクター・オブジェクトがインスタンス化され、複数の論理コネクターがそのオブジェクトを共用します。

構成に関しては、コネクター再利用はマスターとスレーブの関係です。再利用される側 (「マスター」) のコネクターに接続およびパーサーの完全な構成が設定され、再利用する側のすべてのコネクターにはマスター・コネクターへの参照が設定されます。再利用する側のすべてのコネクターは、再利用される側であるコネクターの接続およびパーサーの設定を共用します。接続およびパーサーの設定は再利用する側のコネクターに対して固定ですが、別に構成される他の特定の機能もあります (いずれかのパラメーターが別に構成されていない場合は、マスター・コネクターからパラメーターが継承されます)。

- 入出力マップ
- リンク基準

- フック
- デルタ設定
- 再接続設定

通常、コネクターは同じモード (イテレーターとサーバーは除く) であれば問題なく再利用できます。つまり、例えばルックアップ・モードで必要なのみ何度でも安全にコネクターを再利用できます。

コネクターを複数の異なるモードで再利用すると、問題が発生する可能性があります。共用の物理コネクター・オブジェクトの初期化と終了が行われるのは一度のみです。そのため、コネクターの初期化と終了の手順は、サポートされているすべてのモードで同じである必要があります。

各種のモードで再利用できる IBM Security Directory Integrator コネクターのリストを以下に示します。

- Lotus Domino ユーザー・コネクター
- DSMLv2 SOAP コネクター
- HTTP クライアント・コネクター
- IBM MQ コネクター
- ITIM Agent コネクター
- JDBC コネクター
- JMS コネクター
- JNDI コネクター
- LDAP コネクター
- Lotus® Notes コネクター
- メールボックス・コネクター
- プロパティー・コネクター
- SAP ABAP Application Server Business Object Repository コネクター
- SAP ABAP Application Server ユーザー・レジストリー・コネクター
- スクリプト・コネクター (ユーザーが指定する Javascript による)
- SNMP コネクター
- システム・キュー・コネクター
- システム・ストア・コネクター
- 旧 Tivoli® Access Manager コネクター
- TCP コネクター
- TIM DSMLv2 コネクター
- URL コネクター
- Windows ユーザー/グループ・コネクター
- TADDM コネクター

このリストにないコネクターは同じ AssemblyLine 内で再利用できません。それは、再利用する意味がないか、またはコネクターの内部ロジックで再利用が許可されていないことが理由です。

AssemblyLine で再利用するためのコネクターの構成については、「*Directory Integrator の構成*」を参照してください。構成済み AssemblyLine では、再利用される側のコネクターは、「@」が名前の前に付加されて表示されます。

Active Directory 変更検出コネクター

Active Directory 変更検出コネクターは、変更された Active Directory オブジェクトを報告するため、他のリポジトリーを Active Directory と同期させることができます。この情報を使用することができます。

Active Directory 変更検出コネクター (以降 ADCD コネクターと記述) は、LDAP コネクターの特殊インスタンスです。

変更済みオブジェクトの検索には、LDAP プロトコルが使用されます。

このコネクターを実行すると、他のリポジトリーと Active Directory とを同期するために必要なオブジェクトの変更を、コネクターがオフライン中に発生したものと、コネクターがオンラインで作動中に発生したものの両方について報告します。

このコネクターでは、項目レベルでのみデルタ・タグもサポートされています。

ADCD コネクターはイテレーター・モードで動作します。

Active Directory 内での変更の追跡

ここに記載されている情報を読むことで、Active Directory 内での変更を追跡できるようになります。

IBM Security Directory Integrator や他の一部の LDAP サーバーとは異なり、Active Directory には変更ログは用意されていません。

ADCD コネクターは、**uSNChanged** という Active Directory の属性を使用して、変更されたオブジェクトを検出します。

Active Directory の各オブジェクトには **uSNChanged** 属性があり、この属性はディレクトリーに対してグローバルな USN (更新シーケンス番号) オブジェクトに対応しています。Active Directory オブジェクトが作成、変更、または削除されると、常にグローバル・シーケンス・オブジェクトの値が増分され、その新しい値がオブジェクトの **uSNChanged** 属性に割り当てられます。

AssemblyLine の反復 (コネクターのメソッド `getNextEntry()` の呼び出し) ごとに、コネクターは Active Directory 内の変更されたオブジェクトを 1 つずつ戻します。コネクターは、変更された Active Directory オブジェクトをそのままの状態で (現在の属性すべてを含めて) 引き渡し、オブジェクト変更のタイプとして更新 (追加か変更) または削除を報告します。コネクターは、そのオブジェクト内でどの属性が変更されたか、および属性変更のタイプについては報告しません。

コネクターが保持する同期状態はユーザー・プロパティー・ストアに保存されます。変更されたオブジェクトを報告するたびに、コネクターは、中断と再始動が行われた場合に正しい箇所から処理を続行するのに必要な USN 番号を保存します。

ADCD コネクターは、始動された時点で IBM Security Directory Integrator のユーザー・プロパティー・ストアから、最新の ADCD コネクター・セッションで保存されたこの USN 値を読み取ります。

Active Directory 内の変更の追跡に関する MSDN からの情報は、ここを参照してください。 uSNChanged 属性を使用する変更のポーリングについては、ここを参照してください。

Active Directory 内で削除されたオブジェクト

ここに記載されている情報を読むことで、Active Directory 内で削除されたオブジェクトの扱い方について知ることができます。

オブジェクトがディレクトリーから削除されると、Active Directory は以下のステップを実行します。

- オブジェクトの **isDeleted** 属性を TRUE に設定します。isDeleted==TRUE であるオブジェクトは、Tombstone と呼ばれます (IBM Security Directory Integrator のトゥームストーンとは関係ありません)。
- Active Directory が必要としない属性をすべて削除します。一部のキー属性、例えば **objectGUID**、**objectSID**、**nTSecurityDescriptor**、**uSNChanged** などは保持します。
- 削除されたオブジェクトのコンテナー (ディレクトリー区画内の非表示のコンテナー) に Tombstone を移動します。

削除が実行された後しばらくしてから、Tombstone (削除されたオブジェクト) のガーベッジ・コレクションを行います。「cn=Directory Service,cn=Windows NT,cn=Service,cn=Configuration,dc=ForestRootDomain」オブジェクトの 2 つの設定により、どの Tombstone がいつ削除されるかが決まります。

- 「ガーベッジ・コレクション間隔」により、ドメイン・コントローラー上でのガーベッジ・コレクションの間隔 (時間数) が決まります。デフォルトの設定は 12 時間、最小の設定は 1 時間です。
- 「Tombstone の有効期間」により、Tombstone がガーベッジ・コレクションの対象になるまで存続する日数が決まります。デフォルトの設定は 60 日、最小の設定は 2 日です。

上記の仕様は、削除されたオブジェクトを処理する必要のある同期化処理について、以下の要件が求められることを意味します。

- 同期は、Active Directory の「Tombstone の有効期間」の設定よりも短いインターバルで実行する必要があります。
- 同期中のオブジェクト ID には、**objectGUID** 属性を使用する必要があります。オブジェクトの **distinguishedName** 属性 (ディレクトリー・ツリー内でのオブジェクトの位置を一意的に識別する) は利用できません。オブジェクトが削除されると、ディレクトリー・ツリー内でのオブジェクトの位置が変更され (削除されたオブジェクトのコンテナーに移動される)、古い識別名 (DN) は完全に失われるためです。ただし、**objectGUID** 属性は、変更されることはありません。同期中に削除されたオブジェクトが検出された場合は、他のリポジトリーで同じ **objectGUID** を持つオブジェクトを検索して、検出されたオブジェクトを削除する必要があります。

Active Directory 内で移動されたオブジェクト

ここに記載されている情報を通じて、Active Directory 内の移動されたオブジェクトの扱い方について知ることができます。

オブジェクトが Active Directory ツリー内の 1 つの場所から別の場所に移動されると、そのオブジェクトの **distinguishedName** 属性が変更されます。このオブジェクト変更が、オブジェクトの **uSNChanged** 属性の新規に増分された値に基づいて検出された場合、この変更は他の何らかの変更操作のように見えます。オブジェクトの古い識別名 (DN) に関する情報がないためです。

移動されたオブジェクトを適切に処理する必要のある同期化処理では、**objectGUID** 属性を使用します。オブジェクトが移動されても、この属性は変化しません。同期させるリポジトリー内で **objectGUID** 属性による検索を実行すれば正しいオブジェクトが見つかるため、新旧の識別名を比較することで、オブジェクトが移動されたかどうかを検査できます。

objectGUID をオブジェクト ID として使用

ここに記載されている情報を読むことで、objectGUID をオブジェクト ID として使用する方法を理解することができます。

Active Directory 内の変更を追跡するときは、LDAP 識別名 (DN) ではなく、**objectGUID** 属性をオブジェクト ID として使用してください。これは、Active Directory 内でオブジェクトが削除または移動されると、そのオブジェクトの識別名が失われるためです。それに対して、**objectGUID** 属性は常に保持されているため、オブジェクトの識別に利用できます。

ADCD コネクターから項目の変更が報告されたときには、他のリポジトリー内で **objectGUID** 値による検索を実行して、変更または削除する必要のあるオブジェクトを特定します。したがって、他のリポジトリーでは **objectGUID** 属性を同期し、その値を格納する必要があります。

変更検出

ADCD コネクターは、変更されたオブジェクトを **uSNChanged** 属性値の履歴に従って検出および報告します。変更されたオブジェクトは、**uSNChanged** 値の小さいものが、**uSNChanged** 値の大きいものより前に報告されます。

このコネクターは、`(usnChanged>=X)` というタイプの LDAP 照会を実行します (X は、現在の同期状態を表す USN 番号)。検索操作ではソートおよびページという LDAP v3 コントロールを使用して、変更の履歴を提供し、大規模な結果セットを処理できるようにします。削除されたオブジェクトも検索結果に含めることを指定するには、削除済み項目の表示という LDAP v3 要求コントロール (OID「1.2.840.113556.1.4.417」) を使用します。

ADCD コネクターは、変更されたオブジェクトをすべて連続的に報告します。中断、開始時期と停止時期、およびコネクターがオンラインまたはオフラインのどちらの状態のときに変更が発生したかは関係ありません。コネクターが保持する同期状態はユーザー・プロパティー・ストアに保管されます。変更されたオブジェクトを報告するごとに、コネクターは、中断と再始動があった場合に正しい箇所から処理を続行するのに必要な USN 番号を保管します。

報告する変更がなくなった時点で (タイムアウト値に基づいて判断される)、コネクターはデータの終わりを通知し、処理を停止します。

検索する変更済みオブジェクトがなくなると、Active Directory コネクターは Active Directory での新たなオブジェクト変更を待ちながら循環します。**スリープ間隔**パラメーターは、コネクターが新たな変更を待つときの 2 つの連続したポーリング間の秒数を指定します。コネクターは、新しい Active Directory オブジェクトが検索されるか、タイムアウト時間 (**タイムアウト**・パラメーターにより指定) が満了するまで、ループします。タイムアウト時間が満了した場合は、Active Directory コネクターは、戻す項目が残っていないことを示す **NULL** 項目を戻します。新しい Active Directory オブジェクトが検索されると、そのオブジェクトは前に述べたように処理され、Active Directory コネクターは新しい項目を戻します。

古いバージョンのコネクターでは、追加された項目と変更された項目の両方を更新された項目として報告していました。現在、このコネクターは追加と変更を区別し、それぞれの操作を別々に報告します (詳しくは、『オフラインおよびページングされた結果の事例』を参照)。

ADCD コネクターは、変更された Active Directory オブジェクトをすべての現行属性とともにそのまま引き渡します。どのオブジェクト属性が変更されたか、およびオブジェクトが何回変更されたかは判別されません。オブジェクトに対するすべての中間の変更内容は完全に失われます。Active Directory コネクターが報告する各オブジェクトは、そのオブジェクトに対して実行されたすべての変更の累積効果を表します。ただし、Active Directory コネクターは、複製データ・ソース上で実行する必要のあるオブジェクト変更のタイプを認識し、複製データ・ソースにおいてそのオブジェクトの更新または削除を行う必要があるかどうかを報告します。

注: 取得できるのは、ユーザーが読み取り許可を持っているオブジェクトと属性のみです。このコネクターは、ユーザーが読み取り許可を持っていないオブジェクトまたは属性については、それが Active Directory 内に存在していても、取得しません。そのような場合、ADCD コネクターは、そのオブジェクトまたは属性が Active Directory 内に存在しない場合と同じ動作をします。

オフラインおよびページングされた結果の事例

ここに記載されている情報を使用することで、オフラインだったときの変更検出の結果である場合、および変更検出においてページングされた結果が適用される場合に対処することができます。

コネクターがオフラインのとき、または**ページングされた結果**が使用可能になっているときに初期検索要求が行われ、変更された項目を含むページがまだ取得されていない場合は、その項目に対して行われた複数の変更がマージされます。つまり、コネクターは 1 つの項目のみを受け取り、その中に、その項目に対して適用されたすべての操作の結果が含まれます。

このような場合、Active Directory に項目が**追加**され、その後**削除**されたときは、コネクターは、Active Directory と同期しているリポジトリーに追加されていない項目について、「**削除**」の操作を報告します。これは、重大な制約事項ではありません。IBM Security Directory Integrator コネクターの削除モードでは、最初に削除対象の項目が存在するかどうかが検査され、その項目が存在しない場合は「一致なし

の場合」フックが呼び出されるためです。このフック内に、その種の不要な削除を処理または無視するコードを記述できます。

別のシナリオとして、項目が**追加**され、その後に**変更**された場合があります。その場合、コネクターはその項目について「**追加**」の操作を報告します。項目には、追加後に行われたすべての変更が含まれます。

考えられるその他すべての場合、戻される項目にはすべての変更が含まれ、その項目に対して最後に行われた操作を使用してタグが付けられます。

Active Directory 変更検出コネクターの使用

Active Directory 変更検出コネクターは、以下に示す情報を通じて使用することができます。

コネクターから引き渡される各項目には **changeType** 属性が含まれており、その属性の値は「add」(新しく作成されたオブジェクトの場合)、「modify」(変更されたオブジェクトの場合)、または「delete」(削除された Active Directory オブジェクトの場合) のいずれかです。各項目には、objectGUID 値を表す次の 2 つの属性も含まれています。

- **objectGUID** – 属性。この属性の値は 16 バイトのバイト配列で、対応する Active Directory オブジェクトの 128 ビットの objectGUID を表しています。
- **objectGUIDStr** – 属性。この属性の値は、128 ビットの objectGUID の 16 進値を表すストリングです。この値は、{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx} の形式で引き渡されます (各 x は 1 つの 16 進数字です)。

移動または削除されたオブジェクトを検出して処理する必要がある場合は、オブジェクト ID として、LDAP 識別名ではなく **objectGUID** 値を使用する必要があります。LDAP 識別名はオブジェクトが移動または削除されると変わりますが、**objectGUID** 属性は常に変わりません。オブジェクトの **objectGUID** 属性を複製データ・ソースに保管し、この属性を使用してオブジェクトを検索してください。

注: Active Directory 内の削除されたオブジェクトは、構成可能な期間 (デフォルトは 60 日) が経過するまで存続し、その後で完全に除去されます。削除済みオブジェクトを完全に失ってしまうのを防ぐには、より頻繁に増分同期を行うようにしてください。

ADCD コネクターには、同期プロセス中のどの時点でも割り込むことができます。このコネクターは同期プロセスの状態を IBM Security Directory Integrator のユーザー・プロパティー・ストアに保管するので (各項目の検索後)、次回 Active Directory コネクターを開始したときには、割り込まれた位置から正しく同期が続行されます。

ディレクトリーに対するコネクターの認証

さまざまなバージョンの LDAP プロトコルが、さまざまな認証メソッドをサポートしています。そのさまざまなメソッドを参照するには、以下に示す情報をお読みください。

LDAP v2 は、無名、単純、Kerberos v4 の 3 つをサポートしています。LDAP v3 は、無名、単純、および SASL 認証をサポートしています。AD 変更検出コネクターは、無名、単純、および SASL の認証をサポートしています。

無名 この認証メソッドを設定すると、クライアントの接続先のサーバーは、クライアントがどのようなユーザーであるかを認識または関知しません。サーバーは、このようなクライアントに対して、認証されていないユーザー用に構成されたデータへのアクセスを許可します。 ADCD コネクターは、ユーザー名が入力されない場合にこの認証メソッドを自動的に指定します。このタイプの認証が選択されていて、ユーザー名が指定された場合、ADCD コネクターは認証メソッドを自動的に「単純」に設定します。

単純 この場合、クライアント (ADCD コネクター) から完全修飾の識別名とクライアント・パスワードを LDAP サーバーに平文で送信する必要があります。パスワードがネットワークから読み取られる可能性があるため、このことは問題となります。 LDAP サーバーで SSL プロトコルがサポートされている場合は、パスワードの露出を防ぐために、このタイプの認証を SSL プロトコルと組み合わせて使用することができます。単純モードが指定されていて、ユーザー名が指定されない場合、ADCD コネクターは認証モードを自動的に「無名」に設定します。無名モードが選択されていても、ユーザー名が指定された場合は ADCD コネクターにより認証モードが「単純」に設定されます。

SASL LDAP サーバーへの接続時、クライアント (ADCD コネクター) は Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。この認証メカニズムを使用するには、SASL メカニズムを使用するように手動で構成する必要があります。そのためには、「**追加のプロバイダー・パラメーター**」オプションを使用して追加の JNDI パラメーターを指定します。SASL 認証、JNDI によってサポートされている SASL メカニズム、および JNDI で使用できる構成可能 SASL パラメーターについて詳しくは、次の URL を参照してください。http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html

注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

以下にパラメーターの例を示します。この例では、「**追加のプロバイダー・パラメーター**」パラメーターを追加して、SASL 認証メソッドが選択されている場合に DIGEST-MD5 を使用するように LDAP コネクターを構成しています。

```
java.naming.security.authentication:DIGEST-MD5
```

エラー・フロー

変更検出でスローされたエラー・フローのリストを表示することができます。

- 戻された項目に参照が含まれている場合は PartialResultException がスローされます。Active Directory サーバーは外部ドメインへの参照を生成することができます。ただし、生成された参照に従うことができるようにするのはクライアントの責任です。Active Directory はその処理を実行しません。Active Directory は参照

コントロールをサポートしていません。したがって、Active Directory 変更検出コネクターも参照に従うことはできません。

- 「**LDAP URL**」パラメーターが欠落している場合は、例外がスローされます。
- システム・ストアへの USN 値の保存時に問題があった場合、コネクターは例外をスローします。

構成

Active Directory 変更検出コネクターを構成するときは、以下にリストするパラメーターについて注意が必要です。

このコネクターは、以下のパラメーターを必要とします。

LDAP URL
アクセスする Active Directory サービスの LDAP URL を指定します。LDAP URL の形式は、ldap://*hostname*:*port* または ldap://*server_IP_address*:*port* です。例えば、**ldap://localhost:389**。

注: デフォルトの LDAP ポート番号は 389 です。SSL を使用しているときは、デフォルトの LDAP ポート番号は 636 です。

ログイン・ユーザー名
サービスに対する認証に使用する識別名を指定します。例えば、**cn=administrator,cn=users,dc=your_domain,dc=com**。

注: 無名認証を使用する場合は、このパラメーターをブランクのままにしておく必要があります。

ログイン・パスワード
信任状 (パスワード) を指定します。

注: 無名認証を使用する場合は、このパラメーターをブランクのままにしておく必要があります。

認証メソッド
使用する認証メソッドを指定します。指定できる値は、以下のとおりです。

- 無名 (認証を使用しない)
- 単純 (弱い認証 (平文パスワード) を使用)

SSL の使用
Active Directory との LDAP 通信に Secure Sockets Layer を使用するかどうかを指定します。

追加のプロバイダー・パラメーター
複数の追加パラメーターを JNDI レイヤーに受け渡せるようにします。名前:値のペアとして 1 行に 1 つずつ指定します。

バイナリー属性
ストリングではなくバイナリー値として解釈するパラメーターのリストを指定します。このパラメーターのデフォルト値は **objectGUID objectSid** です。

LDAP 検索ベース
変更の有無を調べるためにポーリングされる Active Directory サブツリーを

指定します。既に削除されているオブジェクトを検出する必要がある場合は、検索ベースとして Active Directory ネーミング・コンテキストを使用する必要があります。例えば、**dc=your_domain,dc=com**。

ページ・サイズ
この要求が戻すページあたりの項目数を指定します (デフォルト値は 500)。

イテレーター状態キー
IBM Security Directory Integrator のユーザー・プロパティー・ストアに現在の同期状態を格納するパラメーターの名前を指定します。これは、IBM Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。「**削除**」ボタンを使用すると、この情報をユーザー・プロパティー・ストアから削除できます。

開始位置
EOD または **0** を指定します。**EOD** は、コネクターの開始後に発生した変更のみを報告することを意味します。**0** は、完全同期の実行を意味します。つまり、Active Directory サービス内の使用可能なすべてのオブジェクトを報告します。このパラメーターは、「**イテレーター状態キー**」パラメーターによって指定されたパラメーターがユーザー・プロパティー・ストアに存在しない場合にのみ有効です。

状態キーの維持
コネクターの状態をいつシステム・ストアに書き込むかを決定します。デフォルト (推奨される設定でもある) は「**サイクルの終わり**」です。以下のいずれかを選択できます。

読み取り後
AssemblyLine の残りの部分を続行する前、Active Directory 変更ログから項目を読み取るときにシステム・ストアを更新します。

サイクルの終わり
AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、変更ログ番号を使用してシステム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。代わりに、AssemblyLine の特定の時点において、ADCD コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保存する必要があります。

通知の使用
Active Directory での新規変更の待機時に通知を使用するかどうかを指定します。使用不可の場合、コネクターは新規変更をポーリングします。

使用可能にすると、コネクターはスリープまたはタイムアウトせず、代わりに Active Directory サーバーからの変更通知イベント (*サーバー検索通知コントロール* (OID 1.2.840.113556.1.4.528)) を待機します。スリープ間隔とタイムアウトのパラメーターは無視されます。

タイムアウト
コネクターが次の変更済み Active Directory オブジェクトを待つ最大秒数を指定します。このパラメーターが **0** の場合は、コネクターは無期限に待機

します。タイムアウト 秒数内に次の変更済み Active Directory オブジェクトを検索しなかった場合は、コネクターは、戻す項目が残っていないことを示す空 (**NULL**) 項目を戻します。デフォルトは 5 です。

スリープ間隔
コネクターが連続したポーリング間でスリープする秒数を指定します。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

コメント
ユーザーのコメントを入力します。

注: タイムアウトまたはスリープ間隔の値を変更すると、そのピアは自動的に有効な値に調整されます (例えば、タイムアウトをスリープ間隔よりも大きくした場合、未編集の値は変更された値にあわせて調整されます)。調整はフィールド・エディターがフォーカスを失うときに実行されます。

関連情報

243 ページの『LDAP コネクター』,
341 ページの『Sun Directory 変更検出コネクター』,
173 ページの『IBM Security Directory Integrator 変更ログ・コネクター』,
418 ページの『z/OS LDAP 変更ログ・コネクター』,
Windows 2000 および
Windows Server 2003 で
Active Directory 内のオブジェクト属性の変更をポーリングする方法。

AssemblyLine コネクター

AssemblyLine コネクターを使用すると、ワークフローへの AssemblyLine の統合を容易に行うことができるようになります。AssemblyLine コネクターは、この作業を標準的かつ分かりやすい方法で行う手段を提供します。AssemblyLine コネクターは AssemblyLine の実行に関係するスクリプト記述の多くをラップします。

AssemblyLines は、多くの場合、複合関数として他の AssemblyLines から呼び出されます。特定のタスクを実行し、入力パラメーターと出力パラメーターをマッピングする呼び出しをセットアップする作業は、スクリプト環境において時間がかかる場合があります。AssemblyLine コネクターは、インライン実行に AssemblyLine の手動サイクル・モードを使用し、内部的に 513 ページの『AssemblyLine 関数コンポーネント』 を使用して処理を行います。

AssemblyLine コネクターはイテレーター・モードのみをサポートしています。ただし、AssemblyLine 操作をサポートする別の AssemblyLine を呼び出す場合は例外です。詳細については、「*Directory Integrator の構成*」の『AssemblyLine 操作』および 775 ページの『付録 F. アダプターを使用した新規コンポーネントの作成』を参照してください。

このコネクターを使用して可能になったサーバー対サーバーの接続機能は、接続されたシステムにアクセスすることを IBM Security Directory Integrator 開発者に許可する一方で、コネクターの稼働パラメーターにアクセスすることや、同じ物理サー

バー上に新たな関数をデプロイすることでそのサーバーの可用性に影響を与えることは禁止したいと管理者が考えている場合などに、セキュリティー上の問題を解決します。

構成

AssemblyLine コネクターを構成するときは、以下にリストするパラメーターについて注意が必要です。

このコネクターは、以下のパラメーターを必要とします。

AssemblyLine
このコネクターの制御のもとで実行される AssemblyLine の名前。ドロップダウン・リストから選択するか、名前を入力します。

AssemblyLine パラメーター
「*パブリッシュ済み AssemblyLine の初期化パラメーター*」にあるターゲット AssemblyLine の「AL の操作」スキーマ・タブで定義されている AssemblyLine パラメーター。これは実際の操作ではありません。このスキーマで定義された名前は、この AssemblyLine を呼び出す AssemblyLine コネクターまたは AL 関数コンポーネントを構成するときに使用されます。「パブリッシュ済み AssemblyLine の初期化パラメーター」スキーマの名前の「情報」フィールドは、パラメーターのツールチップ (説明) として使用されます。

ターゲット AssemblyLine が作成されると、ユーザーが指定したこれらすべての属性および値は、どのコネクターが初期化されるよりも前に、ターゲット AssemblyLine に渡されます。これらの属性と値のペアにアクセスするには、`task.getOpEntry().getAttribute("<name from schema>")` のようにスクリプトを記述するか、操作フォルダーにある式エディターを使用します。

リモート・サーバー
AssemblyLine を定義および実行する IBM Security Directory Integrator サーバー。ローカルのインスタンスはブランク、または *host[:port]* を使用します。

構成インスタンス
リモート・サーバーにある構成インスタンスの ID またはパス。

カスタム鍵ストア
鍵ストア構成に標準の `javax.net.ssl.` プロパティーではなくカスタムの `api.remote.server.` Java プロパティーを使用する場合は、このボックスにチェック・マークを付けます。これにチェック・マークを付けた場合は、`global.properties` の以下のプロパティーも関連します (IBM Security Directory Integrator の『global.properties』も参照してください)。

api.client.keystore
クライアント証明書が含まれている鍵ストア・ファイルを指定します。

api.client.keystore.pass
api.client.keystore に指定した鍵ストア・ファイルのパスワードを指定します。

api.client.key.pass
api.client.keystore に指定した鍵ストア・ファイルに格納されている秘密鍵のパスワードです。このプロパティーが指定されていないと、代わりに api.client.keystore.pass に指定されたパスワードが使用されます。

api.truststore
IBM Security Directory Integrator Server 公開証明書が含まれている鍵ストア・ファイルを指定します。

api.truststore.pass
api.truststore に指定した鍵ストア・ファイルのパスワードを指定します。

シミュレート
設定した場合、呼び出された AssemblyLine が外部システムと対話するときに、その AssemblyLine のシミュレーション構成が使用されます。

ロギングの共用
設定した場合、呼び出された AssemblyLine はこのコネクターと同じロギングを使用します。デフォルトでは、このパラメーターは設定されていません。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

コネクターの使用

AssemblyLine コネクターは、以下に示す情報とリンクを通じて使用することができます。

AssemblyLine コネクターは、ターゲット AssemblyLine から取得する結果セットを反復処理します。ターゲット AssemblyLine は常に、AssemblyLine コネクターによる手動サイクル・モードで同期的に実行されます。ターゲット AssemblyLine は、スレッドに対してローカルでも、リモート・サーバー上でも (サーバー API を使用) 構いません。

機能の大部分は 513 ページの『AssemblyLine 関数コンポーネント』 コンポーネントにインプリメントされており、AssemblyLine コネクターは発生しているエラーを単にリダイレクトする点に注意してください。

属性マッピング (スキーマ) とモード

ここに記載されている情報を参照して、AssemblyLine コネクターの属性マッピングおよびモードを構成します。

AssemblyLineConnector は、ターゲット AssemblyLine で使用可能な操作に基づいて、選択可能なコネクター・モード (イテレーターなど) を動的に報告します。ターゲット AssemblyLine は任意の操作名を定義でき、定義した操作名はコネクターのモード・ドロップダウン・リストに表示されます。操作またはモードが標準モード名でない場合は、暗黙で CallReply モードが内部的に使用されます (つまり、UI は CallReply と同等のレイアウトに変更され、queryReply メソッドが AssemblyLineConnector で呼び出されます)。カスタム・コネクターの開発をさらに

支援するために、AssemblyLine コネクターでは以下に示す操作名に特別な重要度が与えられています。 ScriptConnector の場合、操作名は機能名と同じであり、ConnectorInterface メソッド名とも同じです。

表 1. 操作名

計算モード	**必要な操作**
イテレーター	getNextEntry selectEntries
AddOnly	putEntry
ルックアップ	findEntry
更新	findEntry modEntry putEntry
削除	findEntry deleteEntry
CallReply	queryReply
該当なし	initialize terminate *これら 2 つはオプションの操作ですが、存在する場合は呼び出されます。*

これらのうち 1 つ以上が存在する場合、AssemblyLine コネクターはサポートされているモードを操作に基づいて計算し、ターゲット AssemblyLine はアダプター・モードであると見なされます。通常モードとアダプター・モードの違いは、AssemblyLine コネクターがターゲット AssemblyLine の操作を呼び出す方法です。

モードが内部コネクター・インターフェース・メソッドで終了し、その際に対応する操作が定義されていない場合、例外がスローされます。この例外は CallReply モードで、これがすべての非標準モードのデフォルトになります。

例として、ターゲット AssemblyLine が findEntry を操作としてインプリメントしている場合、UI には選択可能なモードとして「ルックアップ」が表示されます。AssemblyLine コネクターは、AssemblyLine によって呼び出されると、findEntry 操作を呼び出すことによって「ネイティブ」メソッド (例えば findEntry) をターゲット AssemblyLine に直接転送します。別の例として、「削除」では、AssemblyLine コネクターは findEntry を呼び出し、続いて deleteEntry を呼び出して削除操作を実行します。通常モード (例えばターゲット AssemblyLine で DeleteUser 操作が定義されている場合) では、AssemblyLine コネクターは単に DeleteUser 操作を呼び出すだけで、削除操作全体をターゲット AssemblyLine に任せます。ターゲット AssemblyLine では標準モードを操作として定義できますが、そのような定義はお勧めしません。一部の操作は、モード操作を完了するのに複数の操作が必要であり (例えば削除と更新では、削除または更新の前に findEntry を呼び出す)、そのため正常に機能しません。

AssemblyLine コネクターがアダプター・モードの AssemblyLine で操作を呼び出すと、出力属性マップからターゲット AssemblyLine の作業項目に結果が渡されます。リンク基準が必要な場合、AssemblyLine コネクターは `com.ibm.di.server.SearchCriteria` インスタンス・オブジェクトをターゲット AssemblyLine の op 項目に検索として追加します。ターゲット AssemblyLine は、`task.getOpEntry().getObject("search")` を呼び出すことによってこのオブジェクトを取得できます。

ターゲット AssemblyLine からの結果は、常に作業項目に伝えられます。この項目が AssemblyLine コネクターの *conn* 項目となり、入力属性マップに渡されます。このルールに対する 1 つの例外は、結果の作業項目に「conn」という属性が含まれているときです。この属性が存在する場合、AssemblyLine コネクターは戻された作業項目の属性をすべて無視し、*conn* 属性を操作の結果として使用します。 *conn* 項目は任意の数の項目オブジェクトを含むことができます。これは通常、findEntry から NULL または複数の項目が戻された場合に使用されます。*conn* 属性に値がない場合、それは NULL を戻すことと同等であり、*on-no-match* フックが呼び出されます。 *conn* 属性に複数の値がある場合、AssemblyLine コネクターはすべての項目を「複数項目時」配列に追加します。それによって AssemblyLine は「*複数項目時*」フックをトリガーし、`getFindEntryCount()` および `getNextFindEntry()` メソッドを使用して重複項目を使用可能にします。 *conn* 属性に `com.ibm.di.entry.Entry` タイプでないオブジェクトが含まれている場合は、エラーがスローされます。

ターゲット AssemblyLine の操作がネイティブのコネクター・インターフェース・メソッド (getnext や selectentries など) 経由で呼び出されると、AssemblyLine コネクターにより、事前に定義された属性名を持つ作業項目が提供されます。これらの属性を以下に示します。

conn
このコネクターとアダプターとの間で渡される 1 つ以上の項目。有効な値は、NULL、単一項目、項目の配列または集合 (複数の項目が存在する場合) です。

search
ユーザーが指定した検索基準。

current
現在のターゲット・オブジェクト。既存の項目を変更する場合に、この属性が使用されます。

逆に、ターゲット AssemblyLine がこれらのメソッドや操作のデータを返す場合は、これらの属性名を持つ項目も返す必要があります。

AssemblyLine コネクターのターゲット AssemblyLine に操作が定義されていない場合、AssemblyLine コネクターはイテレーターを唯一のモードとして報告します。AssemblyLine コネクターはターゲット AssemblyLine で操作を呼び出さず、単にターゲット AssemblyLine の `executeCycle(work)` を呼び出して次の入力項目を取得します。

スキーマ・ディスカバリー

AssemblyLine コネクターのスキーマを取得するには、その前に正しいターゲット AssemblyLine を使用してコネクターが構成され、使用するモードが選択されている必要があります。スキーマのディスカバーを容易にするための考慮事項は以下のとおりです。

1. AL コネクターを構成するときに選択されたモードと完全に一致する操作があるかどうかを確認するために、ターゲット AL が検査されます。この名前が例えば「myCleverOps」である場合、「myCleverOps」という操作の有無が検査されます。見つかった場合は、操作のスキーマが AL コネクターに返されます。
2. 名前が派生した名前 (イテレーター、AddOnly など) である場合、呼び出される実際の操作が 20 ページの表 1 に示した操作であっても、同じことが行われます。派生した名前の操作が存在しない場合、呼び出される実際の操作のスキーマがあれば、そのスキーマが使用されます。

上に示した 2 つのステップのどちらでも一致が見つからなかった場合 (例えば、不明な操作を使用した場合など)、スキーマは「querySchema」という操作から取得されます。この操作が呼び出されることはありません。この操作は、AL コネクターで取得できるスキーマを定義するためにだけ使用されます。

値として「*」を使用すると、**すべての**属性をマップできます。

AssemblyLine パラメーター

ターゲット AssemblyLine には、タスク制御ブロック (TCB) をパラメーターとして渡すことができます。以下に示す情報を参照してください。同様に、リンク先の情報もご参照ください。

このパラメーターは実行時に生成されるものであり、AssemblyLine コネクターはこれを利用してターゲット AssemblyLine にパラメーターを渡します。これは、ターゲット AL の「パブリッシュ済み AssemblyLine の初期化パラメーター」操作を「**AssemblyLine パラメーター**」パラメーターと組み合わせて使用してパラメーターを指定する代わりの方法です。

関連情報

775 ページの『付録 F. アダプターを使用した新規コンポーネントの作成』.

Axis Easy Web Service サーバー・コネクター

Axis Easy Web Service サーバー・コネクターの処理には、以下に示す情報を使用します。

Axis Easy Web Service サーバー・コネクターは、IBM Security Directory Integrator Web Services Suite に含まれています。このコネクターは、AxisSoapToJava および AxisJavaToSoap の各関数コンポーネントをインスタンス化、構成、および使用するという点で、 407 ページの『Web サービス受信側サーバー・コネクター』を単純化したバージョンです。

注: このコンポーネントで使用されている Axis ライブラリーの制約により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

提供される機能は、407 ページの『Web サービス受信側サーバー・コネクター』をホストする AssemblyLine 内でこの 2 つの関数コンポーネントを連結して構成した場合と同じです。このコネクターを使用すると、SOAP 要求を構文解析する前や SOAP 応答をシリアライズした後にカスタム処理をフックする必要がなくなります。つまり、Axis によって提供される処理とバインディングに従う必要がありますが、セットアップと使用が単純になります。

Axis Easy Web Service サーバー・コネクターはサーバー・モードのみで稼働します。

AssemblyLine では、操作項目 (op 項目) がサポートされています。op 項目の属性 *$operation* には、AssemblyLine により実行された現行操作の名前が含まれています。さまざまな Web サービス操作を容易に処理できるようにするため、Axis Easy Web Service サーバー・コネクターにより op 項目の *$operation* 属性が設定されます。

Axis Easy Web Service サーバー・コネクターは、AssemblyLine の入出力スキーマに基づく WSDL ファイルの生成をサポートしています。 IBM Security Directory Integrator では AssemblyLine により複数操作がサポートされるため、WSDL を生成すると、複数の操作を持つ Web サービス定義が生成されることがあります。操作の命名に関するルールを以下に示します。

- 6.1 より前のバージョンの IBM Security Directory Integrator 構成ファイルでは、1 つの入力スキーマと 1 つの出力スキーマのみがデフォルト操作スキーマとして参照されていました。6.1 より前のバージョンの IBM Security Directory Integrator 構成を使用する場合は、IBM Security Directory Integrator 6.0 の場合と同様に、1 つの操作のみが生成され、この操作の名前として AssemblyLine 名が設定されます。
- IBM Security Directory Integrator 構成では、「Default」という名前の操作が存在する場合、これに対応する WSDL ファイルの操作の名前には AssemblyLine 名が設定されます。
- IBM Security Directory Integrator 構成では、「Default」という名前の操作と AssemblyLine 名を持つ操作が存在する場合、 WSDL ファイルでは両方の操作の名前が維持されます。
- 上記に該当しない場合はすべて、WSDL ファイルの操作には AssemblyLine 構成での名前が使用されます。

このコネクターの構成は比較的単純です。コネクターは、受信した SOAP 要求を解析し、その内容を (HTTP 固有のデータとともに) イベント項目に格納し、属性マッピングのためにその情報を AssemblyLine に提示します。応答フェーズで作業項目 (SOAP 応答の Java 表現が格納されている項目) がコネクターに戻されると、コネクターはその応答をシリアライズして、Web サービス・クライアントに戻します。

このコネクターは SOAP 要求を受信すると、この受信した SOAP 要求を解析し、op 項目の $operation 属性を設定します。操作の名前は、SOAP エンベロープの Body エレメントでネストしているエレメントの名前により決定します。SOAP メッ

セージを解析するために SAX パーサーが使用されます。SAX パーサーは、DOM パーサーと比較するとパフォーマンスのオーバーヘッドが少なくて済みます。

SOAP メッセージにはいくつかのタイプがあります。

- RPC スタイルの SOAP メッセージを使用する場合は、エレメント名と操作名が同一になります。
- 文書スタイルの SOAP メッセージを使用する場合は、以下の 2 つの状況が考えられます。
  - ラップされた文書スタイルの SOAP メッセージ (この場合 SOAP メッセージ本体が RPC スタイルの SOAP メッセージに類似している) を使用する。これは、SOAP Body エレメントでネストしているエレメントに SOAP 本体の内容をラップすることで実現します。このエレメントの名前は、SOAP 操作名と同一です。
  - 標準またはラップ解除文書スタイルの SOAP メッセージを使用する。この場合は SOAP 操作の概念が定義されません。つまり、SOAP メッセージが SOAP メッセージ交換の一部になっています。この場合、コネクターは、op 項目の $operation 属性を、SOAP Body エレメント内でネストされたエレメントの名前に設定します。また、IBM Security Directory Integrator の開発者/デプロイ担当者は、IBM Security Directory Integrator ソリューションによってこの状況が正しく処理されることを確認する必要があります。標準またはラップ解除文書スタイルの SOAP メッセージを使用する場合は、op 項目の $operation 属性の値に依存しないことをお勧めします。

WSDL ファイルのホスティング

Axis Easy Web Service サーバー・コネクターは、*wsdlRequested* というコネクター属性を AssemblyLine に提供します。ここに記載されている情報を使用することで、WSDL ファイルをホストできるようになります。

HTTP 要求が到着したとき、要求された HTTP リソースの末尾に「?WSDL」がある場合、コネクターは *wsdlRequested* 属性の値を **true** に設定し、「**WSDL ファイル**」パラメーターで指定されたファイルの内容を *soapResponse* というコネクター属性に読み取ります。「?WSDL」がない場合、この属性の値は **false** に設定されます。

したがって、この属性の値を利用すると、純粋な SOAP 要求と、WSDL ファイルを求める HTTP 要求を区別できます。AssemblyLine では、分岐コンポーネントを使用して、ロジックのうち適切な部分のみを実行できます。つまり、(1) WSDL ファイルを求める要求を受信した場合、AssemblyLine は何らかのオプション・ロジックを実行するか、異なる WSDL ファイルを読み取って、それを Web サービス・クライアントに返送することも、あるいは単にデフォルトの処理に任せることもできます。(2) SOAP 要求を受信した場合、AssemblyLine はその SOAP 要求を処理します。別の方法として、適切な場所 (スクリプト・コンポーネント内、AssemblyLine の最初のコネクターのフック内など) に `system.skipEntry();` 呼び出しをプログラミングして、以降の処理をスキップし、チャネル応答処理に直接進むこともできます。

SOAP 要求に対して必要な応答を提供する処理は、AssemblyLine の役割です。

このコネクターは、次の共通メソッドをインプリメントしています。

```
public String readFile (String aFileName) throws IOException;
```

このメソッドをスクリプト・コンポーネント内の IBM Security Directory Integrator JavaScript で使用すると、ローカル・ファイル・システムにある WSDL ファイルの内容を読み取ることができます。その場合 AssemblyLine は、WSDL の内容を *soapResponse* 属性に戻し、それにより Web サービス・クライアントに戻します (WSDL に対する要求を受信した場合)。

スキーマ

Axis Easy Web Service サーバー・コネクターには、入力と出力の 2 つのスキーマがあります。詳細については、以下に記載されている情報を参照してください。

入力スキーマ

表 2. Axis Easy Web Service サーバー・コネクターの入力スキーマ

属性	値
host	タイプは String です。要求が送信された先のホスト名が含まれています。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
requestObjArray	オブジェクトの配列として表される soapRequest。 SoapToJava 関数コンポーネントの実行方法で java.lang.String を Object[] に変換します。
requestedResource	要求された HTTP リソース。
soapAction	SOAP アクションの HTTP ヘッダー。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
soapFault	SOAP エラーが発生した場合、`org.apache.axis.AxisFault` がこの属性に格納されます。
soapRequest	txt/XML または DOMElement フォーマットの SOAP 要求。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
soapResponse	SOAP 応答メッセージ。wsdlRequested が true の場合、soapResponse は WSDL ファイルの内容に設定されます。
wsdlRequested	WSDL ファイルが要求される場合、このパラメーターは true、要求されない場合は false です。
http.username	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。値は接続されているクライアントのユーザー名です。
http.password	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。値は接続されているクライアントのパスワードです。

出力スキーマ

表 3. Axis Easy Web Service サーバー・コネクターの出力スキーマ

属性	値
responseContentType	応答タイプ。
responseObjArray	オブジェクトの配列として表される soapRequest。 soapResponse は JavaToSoap 関数コンポーネントを使用してここから値を取得し、 Object[] を java.lang.String に変換します。
soapFault	SOAP エラーが発生した場合、`org.apache.axis.AxisFault` がこの属性に格納されます。
soapResponse	SOAP 応答メッセージ。wsdlRequested が true の場合、soapResponse は WSDL ファイルの内容に設定されます。

表 3. *Axis Easy Web Service* サーバー・コネクターの出力スキーマ *(続き)*

属性	値
wsdlRequested	WSDL ファイルが要求される場合、このパラメーターは true、要求されない場合は false です。
http.credentialsValid	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。構文はブール値で、true の場合クライアント認証が成功します。HTTP 基本認証を使用する場合、AssemblyLine がこのパラメーターの値を設定します。

構成

Axis Easy Web Service サーバー・コネクターを構成するときは、以下にリストするパラメーターについて注意が必要です。

パラメーター

TCP ポート
Web サービスがクライアント接続を listen するポート。

接続バックログ
着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

WSDL ファイル
このパラメーターは必須で、タイプはストリングです。このパラメーターの値は、この Web サービスを記述した WSDL 文書の完全なファイル・システム・パスである必要があります。

SOAP 操作
WSDL ファイルに記述されている SOAP 操作の名前です。

複素数タイプ
このパラメーターは必須ではありませんが、指定する場合は、完全修飾の Java クラス名 (パッケージ名を含む) のリストにします。このリストでは各エレメント (Java クラス) の間に、コンマ、セミコロン、スペース、復帰、改行の記号を 1 つ以上挿入して区切ります。

OP 項目にタグを付ける
このパラメーターをチェックすると (つまり「true」)、実行操作が AssemblyLine/WSDL の公開操作のリストにある場合にのみ、コネクターによって op 項目にタグが付けられます。WSDL 内で操作が見つからない場合は、SOAP フォールト・メッセージが生成され、クライアントに戻されます。

注: IBM Security Directory Integrator 6.0 では AxisEasyWSServerConnector を使用するには「**SOAP 操作**」パラメーターを設定する必要がありました。IBM Security Directory Integrator では、「**OP 項目にタグを付ける**」パラメーターが「true」に設定されていると、AxisEasyWSServerConnector は「**SOAP 操作**」パラメーターに指定されている名前ではなく、抽出された操作名を使用します。この場合「**SOAP 操作**」パラメーターは必須パラメーターではないため、ブランクのままにしておくことができます。

SSL の使用
これをチェックした場合、サーバーは SSL (https) 接続のみを受け入れま

す。SSL のパラメーター (鍵ストアなど) は、IBM Security Directory Integrator インストール・フォルダーにある global.properties ファイルに Java システム・プロパティーの値として指定します。

クライアント認証を必要とする
このコネクターでクライアントがクライアント SSL 証明書による認証を実行することが必要であるかどうかを指定します。このパラメーターの値が **true** の場合 (つまりチェックした場合)、クライアントはクライアント SSL 証明書による認証を実行せず、コネクターはクライアント接続を失います。このパラメーターの値が **true** であり、かつクライアントがクライアント SSL 証明書による認証を実行する場合は、コネクターはクライアント要求の処理を続行します。このパラメーターの値が **false** の場合は、クライアントがクライアント SSL 証明書による認証を実行するかどうかに関係なく、コネクターはクライアント要求を処理します。

認証レルム
認証が要求される場合にクライアントに送られる基本レルムです。デフォルトは「IBM Security Directory Integrator」です。

HTTP 基本認証 (BA) を使用します。
このコネクターでは HTTP 基本認証がサポートされます。アクティブにするには、「**HTTP 基本認証 (BA) を使用**」チェック・ボックスをチェックします。アクティブにすると、サーバーは信任状が既に送信されているかどうかを検査し、送信されていない場合はクライアントに許可要求を送信します。クライアントから必要な信任状が送信された後に、コネクターによって「http.username」と「http.password」という 2 つの属性が設定されます。この 2 つの属性にはそれぞれ、クライアントのユーザー名とパスワードが含まれています。AssemblyLine がこのユーザー名とパスワードが有効であるかどうかを検査します。クライアントが正常に許可された場合は、作業項目属性「http.credentialsValid」を true に設定する必要があります。クライアントが許可されなかった場合は、作業項目属性「http.credentialsValid」を false に設定する必要があります。クライアントが許可されなかった場合は、サーバーから「Not Authorized」HTTP メッセージが送信されます。

コメント
ここにはユーザー独自のコメントを入力します。

詳細ログ
これをチェックした場合、追加のログ・メッセージが生成されます。

WSDL 出力ファイル名
「WSDL 生成」ボタンをクリックしたときに生成される WSDL ファイルの名前です。このパラメーターは、WSDL 生成ユーティリティーのみが使用し、コネクターの実行中は使用しません。

Web サービス・プロバイダー URL
Web サービス・クライアントが Web サービス要求を送信する先のアドレスです。このパラメーターも、WSDL 生成ユーティリティーのみが使用し、コネクターの実行中は使用しません。

「**WSDL 生成**」ボタンをクリックすると、WSDL 生成ユーティリティーが実行されます。

WSDL 生成ユーティリティーは、入力として、生成する WSDL ファイルの名前と、Web サービスのプロバイダーの URL (Web サービスの場所) を使用します。このユーティリティーは、コネクターが組み込まれている AssemblyLine の入力パラメーターと出力パラメーターを抽出し、その情報を使用して入力 WSDL メッセージおよび出力 WSDL メッセージの WSDL 部分を生成します。「初期作業項目」スキーマと「結果項目」スキーマの項目属性それぞれについて、「ネイティブ構文」列に属性の Java タイプ (例えば、`java.lang.String`) を指定することが必須です。このユーティリティーで生成した WSDL ファイルを、後から手動で編集することもできます。

生成される WSDL の中で定義される SOAP 操作の操作スタイルは、*rpc* です。

WSDL 生成ユーティリティーでは、複素数タイプの <types...>...</types> セクションを WSDL 内に生成することができません。

コネクターの操作

コネクターの操作を実行するには、以下に示す情報とリンクを使用します。

HTTP/SOAP 要求間の情報交換に使用される Axis Easy Web Service サーバー・コネクターの属性の概要については、25 ページの『スキーマ』を参照してください。

このコネクターは、受信した SOAP 要求メッセージを解析し、その SOAP 要求の Java 表現を *requestObjArray* コネクター属性に格納します。コネクターは、文書スタイルおよび RPC スタイルの SOAP メッセージをいずれも解析できるほか、(a) 文書スタイルの SOAP 応答メッセージ、(b) RPC スタイルの SOAP 応答メッセージ、および (c) SOAP フォールト応答メッセージを生成することもできます。生成されるメッセージのスタイルは、「**WSDL ファイル**」コネクター・パラメーターで指定された WSDL によって決まります。

このコネクターでは、SOAP 要求メッセージの解析と、SOAP 応答メッセージの生成が可能です。生成する応答メッセージには、WSDL 文書の *<types>* セクションで定義されている複素数タイプの値を含めることができます。これを行うために、このコネクターは、(1)「**複素数タイプ**」コネクター・パラメーターに、SOAP 操作に対する要求および応答パラメーターとして使用される複素数タイプをインプリメントする Java クラスすべての名前が含まれていること、(2) それらの Java クラスのクラス・ファイルが IBM Security Directory Integrator の Java クラス・パスに配置されていることを必要とします。

SOAP 要求の解析中に解析コードから例外がスローされた場合、このコネクターは SOAP フォールト・オブジェクト (`org.apache.axis.AxisFault`) を生成して、そのオブジェクトを *soapFault* コネクター属性に格納します。

このコネクターでは、エンコードされた SOAP 応答メッセージの解析と生成に、「リテラル」エンコード方式および SOAP セクション 5 エンコード方式の両方を使用できます。生成される SOAP 応答メッセージのエンコード方式は、「**WSDL ファイル**」コネクター・パラメーターで指定された WSDL によって決まります。

AssemblyLine 処理の最後のチャネル応答フェーズにおいて、このコネクターは、SOAP 応答メッセージの Java 表現 (*Object[]*) をマップするために作業 項目の

responseObjArray 属性から取得します。その後、コネクターは、SOAP 応答メッセージをシリアライズし、HTTP 応答にラップして、Web サービス・クライアントに戻します。

関連情報

407 ページの『Web サービス受信側サーバー・コネクター』.

Axis2 Web サービス・サーバー・コネクター

Axis2 Web サービス・サーバー・コネクターを使用して、HTTP または HTTPS を使用してアクセス可能な SOAP Web サービスを提供できます。このことについては、ここに記載されている情報を通じて詳しく知ることができます。

このようなサービスのロジックは、IBM Security Directory Integrator AssemblyLine としてインプリメントされ、既存の IBM Security Directory Integrator コンポーネントを利用するものと想定されます。

コネクター名は、基礎になる Axis2 Java ライブラリー (http://ws.apache.org/axis2/) に基づいています。

WSDL 1.1 (http://www.w3.org/TR/wsdl/) 文書および WSDL 2.0 (http://www.w3.org/TR/wsdl20/) 文書の両方がサポートされています。

SOAP 1.1 および SOAP 1.2 の両方のプロトコルがサポートされています。使用できるのはリテラル SOAP メッセージのみで、エンコード SOAP メッセージはサポートされていません。これは、基礎になる Axis2 ライブラリー (バージョン 1.4.0.1) の制限です。

Axis2 Web サービス・サーバー・コネクターではサーバー・モードのみがサポートされています。

Axis1 と Axis2 のコンポーネントの比較

ここに記載されている情報を使用することで、Axis1 と Axis2 のコンポーネントを比較した場合の違いについて詳しく知ることができます。

通常、Axis1 コンポーネントを使用する必要があるのは、次のような場合のみです。

- SOAP エンコードのサポートが必要な場合
- 階層構造を持つ属性ではなく、カスタム生成 (複素数タイプ・ジェネレーター FC による) の Java タイプを使用する場合

その他すべての場合は、次の理由により Axis2 コンポーネントを使用する必要があります。

- SOAP 1.2 および WSDL 2.0 をサポートしている
- スキーマ・ディスカバリー (「querySchema」) に対応
- SOAP ヘッダーの操作が簡単
- SOAP 障害を制御できる
- Axis2 は発展し続けているので、将来の拡張が可能

SOAP エンコード方式のサポート

ここに記載されている情報とリンクを使用することで、SOAP エンコード方式のサポートについて理解することができます。

WSDL1.1 文書でのバインディングは、サービスをメッセージング・プロトコル (特に SOAP メッセージング・プロトコル) にバインドする方法を記述しています。WSDL SOAP バインディングは、リモート・プロシージャー・コール (RPC) スタイルのバインディングか、または文書スタイルのバインディングとすることができます。SOAP バインディングはエンコードでの使用またはリテラルでの使用もできます。そのため、次の 4 つのスタイル/使用モデルがあります。

1. RPC/エンコード
2. RPC/リテラル
3. 文書/エンコード
4. 文書/リテラル

詳しくは、http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/ を参照してください。

IBM Security Directory IntegratorAxis2 コンポーネントでのスタイル/使用モデルのサポートは次のとおりです。

1. **RPC/エンコード**は、Axis2 ライブラリーの制限によりサポートされていません。RPC/エンコードのバインディングは WS-I 基本プロファイル (http://www.ws-i.org/Profiles/BasicProfile-1.1.html#Consistency_of_style_Attribute) に準拠していません。
2. **RPC/リテラル** – サポートされています。
3. **文書/エンコード**はサポートされませんが、まったく使用されないため問題ありません。また、WS-I に準拠していません。
4. **文書/リテラル** – サポートされています。

WSDL 2.0 では、すべてが文書/リテラル・モデルと同様です (すべてのメッセージは XML スキーマのようなタイプ言語を直接使用して定義される)。そのため、Axis2 コンポーネントとの問題はありません。 RPC 呼び出しについては、WSDL 2.0 では、RPC 呼び出しに適したメッセージを設計するための一連のルールが定義されています。詳しくは、http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/74bae690-0201-0010-71a5-9da49f4a53e2 を参照してください。

RPC/エンコード・モデルの普及度

ここに記載されているリンクを使用することで、RPC/エンコード・モデルの普及度について理解することができます。

Web サービスの主要なすべてのフレームワークは、文書/リテラルのメッセージをサポートしています。一般的なフレームワークのほとんどは、RPC/エンコードをある程度サポートしているため、開発者はエンコード専用のサービスの作成に RPC/エンコードを引き続き使用できます。その結果、実動使用でどのくらいの数の Web サービスが SOAP エンコード・メッセージでのみ動作するかを見積もることは困難です。ただし、傾向としては、RPC/エンコードから文書/リテラルへと移行しつつあり

ます。これは、SOAP エンコードの仕様では、100% のインターオペラビリティーが保証されておらず、RPC/エンコードのインプリメンテーションでベンダーごとに相違があるためです。

一部の一般的なフレームワークでのエンコードのサポートに関するいくつかの参照先は、以下のとおりです。

- Microsoft SOAP Toolkit - Microsoft SOAP Toolkit (COM コンポーネントへの Web サービスのアクセスの提供に使用) は、デフォルトで RPC/エンコードを使用します (文書/リテラルもサポートします)。詳しくは、http://msdn2.microsoft.com/en-us/library/ms995793.aspx を参照してください。

  この製品は Microsoft により非推奨となっており、このサポートは 2004 年 7 月 1 日に停止しています。http://msdn2.microsoft.com/en-us/library/aa286526.aspx
- Microsoft .NET (WSE/WCF) - .NET のバージョン 1.1 以降、デフォルトは文書/リテラルですが、RPC/エンコードも許可されています。詳しくは、http://msdn2.microsoft.com/en-us/library/dkwy2d72(VS.71).aspx を参照してください。
- Glassfish (オープン・ソース・ベースの Sun Java System Application Server)、JBossWS、および OracleAS: これらはすべて RPC/エンコードをある程度サポートします (文書/リテラルは完全にサポートされています)。http://wiki.apache.org/ws/StackComparison

代替案

RPC/エンコード・モデルを使用する必要がある場合、古い Web サービス・スイートを使用できます。また、サービスおよび SOAP メッセージに追加情報がある場合、HTTP コンポーネントおよび XML パーサーを使用してソリューションを作成できます。

コネクターの使用

Axis2 Web サービス・サーバー・コネクターは、以下に示す情報を通じて使用することができます。

Axis2 Web サービス・サーバー・コネクターは、「**WSDL First**」という開発方法用に設計されています。つまり、このことは、このコネクターには Web サービスを記述している WSDL 文書が必要であることを意味しています。これにより、コネクターでは、クライアントが予想している Web サービスの振る舞いがどのようなものかが認識されます。したがって、Web サービスのインプリメンテーションは、WSDL によって概略が示されたモデルに準拠している必要があります。(代替案として、ロジックを先にインプリメントし、そのインプリメンテーションにふさわしい WSDL をコネクターに作成させます。) この設計を選択する理由は、既に確立されている WSDL 記述に準拠することによって、IBM Security Directory Integrator が既存の通信モデルに適合しやすくなるためです。

既存の Assembly Line が Web サービス・インターフェースを経由して公開される必要がある状況のために、IBM Security Directory Integrator はいくつかの基本的な WSDL 生成機能を提供します (34 ページの『WSDL の生成』を参照)。

WSDL 文書は、複数のインターフェース (WSDL 1.1 の用語では ポート・タイプ) を記述できます。インターフェースはそれぞれ一連の操作をグループ化します。Axis2 Web サービス・サーバー・コネクターのインスタンス 1 つにつき、インターフェースを 1 つのみインプリメントできます。 AssemblyLine のロジックが、異なる操作を区別できるようにするために、コネクターは操作の名前 (修飾名のローカルの部分) を操作項目 (op-entry) の $operation 属性に渡します。AssemblyLine 操作および操作項目について詳しくは、「*Directory Integrator の構成*」を参照してください。

注:

1. **SOAP のバージョン**は、クライアントによって異なります。クライアントが SOAP 1.1 要求を送信する場合、コネクターは SOAP 1.1 応答を返信します。クライアントが SOAP 1.2 要求を送信する場合、コネクターは SOAP 1.2 応答を返信します。WSDL 文書からの SOAP のバージョン設定は無視されます。
2. コネクターは、着信 SOAP メッセージまたは発信 SOAP メッセージの **XML スキーマの妥当性検査**を実行しません。
3. コネクターは、**HTTP POST** 要求に対してのみ SOAP 処理を行います。その他の HTTP 要求は、Assembly Line のロジックでの処理のために残されます。
4. コネクターは、SOAP 応答を SOAP 要求の応答としてのみ生成します。 HTTP 要求に本体が含まれない場合、コネクターによって SOAP 応答は生成されません。
5. Assembly Line は、`http.body` 出力属性を指定することによって、すべての要求の応答をオーバーライドできます。 Assembly Line によって、オーバーライドする `http.body` 属性が提供される場合、コネクターによって SOAP 応答は生成されません。
6. 特殊な場合には、IBM Security Directory Integrator サーバーの Log4j 構成ファイル (`etc/log4j.properties`) で、基礎となる Axis2 ライブラリーのログ・レベルを構成できます。

サポートされているメッセージ交換パターン

Axis2 Web サービス・サーバー・コネクターでは、以下にリストするメッセージ交換パターン (WSDL 2.0 の用語で記述) がサポートされています。

In-Only
: サーバーは、クライアントから SOAP 要求を受信し、SOAP 応答を生成しません。対応する WSDL 1.1 の用語は、「片方向操作」です。

In-Out
: サーバーは SOAP 要求を受信し、SOAP 障害または通常の SOAP メッセージのいずれかで応答します。対応する WSDL 1.1 の用語は、「要求/応答操作」です。

Robust-In-Only
: サーバーは SOAP 要求をクライアントから受信し、SOAP 障害で応答するか、または SOAP メッセージをまったく使用しないで応答します。このメッセージ交換パターンに対応する WSDL 1.1 の用語はありません。

メッセージ交換パターンについて詳しくは、以下を参照してください。

http://www.w3.org/TR/wsdl20-adjuncts/#patterns
http://www.w3.org/TR/wsdl#_porttypes

注: サーバーで SOAP 応答が生成されない場合でも、HTTP 応答がサーバーからクライアントに返信されます。その場合、HTTP 応答の本体には、SOAP メッセージは含まれません。

SOAP 障害

クライアントの要求に応答して SOAP 障害を生成するよう、Axis2 Web サービス・サーバー・コネクターに指示できます。以下に示すコネクター属性を使用することでこれを実行できます。

- $faultCode
- $faultCodeNamespacePrefix
- $faultCodeNamespaceURI
- $faultReason
- $faultNode
- $faultRole
- $faultDetail

これらおよびその他の属性の説明について詳しくは、『スキーマ』を参照してください。

SOAP 障害について詳しくは、以下を参照してください。

http://www.w3.org/TR/soap12-part1/#soapfault
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383507

SOAP ヘッダー

コネクターは、特殊使用または拡張使用のため、分析用に SOAP 要求の SOAP ヘッダーへのアクセスを提供しています。ここに記載されている情報とリンクを使用することで、SOAP ヘッダーについて詳しく知ることができます。

また、ユーザー定義の SOAP ヘッダーを応答に含めることもできます。

ユーザー定義の SOAP ヘッダーは、通常の SOAP メッセージおよび SOAP 障害の両方に影響を与えることに注意してください。

属性の説明について詳しくは、セクション 34 ページの『スキーマ』を参照してください。

HTTP トランスポート層

ここに記載されている情報とリンクを使用することで、HTTP トランスポート層について詳しく知ることができます。

Axis2 Web サービス・サーバー・コネクターは、 142 ページの『HTTP サーバー・コネクター』を HTTP トランスポートとして使用します。

特殊使用または拡張使用の場合に、HTTP サーバー・コネクターが HTTP 要求および HTTP 応答に対して提供するコントロールを利用できます。

要求の HTTP ヘッダーを分析し、応答の HTTP ヘッダーを設定できます。

また、応答の HTTP の本体全体をオーバーライドすることもできます。Axis2 Web サービス・サーバー・コネクターは、SOAP メッセージを HTTP POST 要求からのみ解析します。HTTP GET 要求は、SOAP エンジンによって処理されず、そのような場合には、独自のロジックを自由にインプリメントできます。例えば、`?wsdl` で終わる URI を持つ HTTP GET 要求を受信した場合に、WSDL 文書を返すことができます。

WSDL の生成

WSDL の生成を実行するには、以下に示す手順を使用します。

Axis2 WS サーバー・コネクターが機能するには、WSDL 文書が必要です。動作する AssemblyLine はあっても、WSDL 文書がない場合、このコネクターのインスタンスを使用して、IBM Security Directory Integrator で WSDL 文書を生成できます。生成された WSDL 文書のサービス名に、AssemblyLine の名前が設定されます。

WSDL の生成機能は、初心者ユーザーがクイック・スタートとして使用することを目的としています。Web サービスに慣れている場合は、WSDL 文書を自分で設計するか、または少なくとも、生成された WSDL 文書を完全に検査してから、実動使用に移行することを強くお勧めします。

WSDL ファイルを生成するには、以下の手順を実行します。

1. WSDL 文書を生成する AssemblyLine に、Axis2 WS サーバー・コネクターのインスタンスを追加します。
2. 「**WSDL 出力ファイル名**」、「**Web サービス・プロバイダー URL**」、および「**WSDL バージョン**」のパラメーターを入力します。
3. 「**WSDL 生成**」ボタンを押します。

スキーマ

Axis2 Easy Web Service サーバーのスキーマとその属性については、以下に示す情報とリンクを通じて把握することができます。

入力スキーマ

トランスポート関連の属性について詳しくは、142 ページの『HTTP サーバー・コネクター』の文書を参照してください。

`http.content-type` や `http.content-length` などの属性を入力マップに追加して、SOAP 要求のこれらのパラメーターを AssemblyLine のロジック内で使用できます。

他に役立つ属性として、サーバーによって受信される要求のタイプ (GET または POST) を保持する `http.method` があります。コネクターは POST 要求のみを解析するため、この属性の値がチェックされ、GET 要求の場合には、特定の戻り値 (例えば、サービスを記述する WSDL 文書または HTML 文書) を設定できます。

http.SOAPAction も、重要な HTTP ヘッダーであり、これは Web サービスで重要になります。SOAP メッセージの HTTP バインディングで設定され、その値は URI です。一部の SOAP バインディングでは SOAPAction は必要なく、この属性は省略されます。

HTTP 要求に埋め込まれている SOAP メッセージ

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
 ...
    </soapenv:Body>
</soapenv:Envelope>
```

着信メッセージなど、WSDL 固有の属性について詳しくは、『Axis2 WS クライアント関数コンポーネント』の『スキーマ』セクションを参照してください。

$soapHeader

着信 SOAP メッセージの SOAP ヘッダーが含まれる階層属性。

例えば、以下のような SOAP メッセージについて考えてみます。

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

 <soapenv:Header
  xmlns:myns="http://www.my.com">
  <myns:myheader />
 </soapenv:Header>

 <soapenv:Body><payload /></soapenv:Body>

</soapenv:Envelope>
```

$soapHeader 階層属性は、以下の XML 要素の DOM 表現です。

```
<$soapHeader
 xmlns:myns="http://www.my.com">
 <myns:myheader />
</$soapHeader>
```

$soapHeader 要素と SOAP メッセージからのヘッダー要素との唯一の違いは、**$soapHeader** 要素には、関連付けられているネーム・スペースがないことです。つまり、どのネーム・スペースを使用するかを考える時間が節約されます (ヘッダー要素のネーム・スペースは、SOAP のバージョンごとに異なります)。

出力スキーマ

トランスポート関連の属性について詳しくは、142 ページの『HTTP サーバー・コネクター』の文書を参照してください。

HTTP 属性を使用すると、SOAP 応答の特性を設定するだけでなく、AssemblyLine の動作を変更できます。例えば、属性 http.body がコネクターの出力マップにマップされる場合、その値は SOAP 応答として直接設定され、Axis2 エンジンは応答の

生成に使用されません。同様の手法が、出荷されているサンプルの最初で使用されています。そこでは、http.method の値がチェックされ、GET 要求の場合には、サービスを記述する WSDL ファイルの内容が http.body 属性に設定されています。POST 要求が受信されると、SOAP 応答が組み立てられて送信されます。

他に役立つ HTTP 属性として、http.status があります。 AssemblyLine のロジックに従って、コネクターおよびその値セットの出力マップに、この属性をマップできます。この方法によって、サーバーが送信する HTTP 応答の状況を変更できます。OK の場合は「200」、Forbidden の場合は「403」、Not Found の場合は「404」などと設定します。

着信メッセージなど、WSDL 固有の属性について詳しくは、『Axis2 WS クライアント関数コンポーネント』の『スキーマ』セクションを参照してください。

$authResult

このオプションの属性は、現行クライアントの認証の結果を表します。

この属性が「true」に設定されている (大/小文字を区別しない) 場合、クライアントの認証は成功していると考えられます。そうでない (この属性が欠落しているか、またはその他の値が設定されている) 場合には、コネクターは認証が失敗していると見なし、エラーの HTTP 応答をクライアントに返します。

認証について詳しくは、 41 ページの『認証』を参照してください。

$soapHeader

階層属性。その子要素は、発信 SOAP メッセージの SOAP ヘッダーに追加されます。

例えば、以下のような SOAP メッセージについて考えてみます。

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

 <soapenv:Header
  xmlns:myns="http://www.my.com">
  <myns:myheader />
 </soapenv:Header>

 <soapenv:Body><payload/></soapenv:Body>

</soapenv:Envelope>
```

$soapHeader 階層属性が以下の XML 要素の DOM 表現であるとします。

```
<$soapHeader
 xmlns:otherns="http://www.other.com">
 <otherns:otherheader>
  This is an example of a user-defined SOAP header.
 </otherns:otherheader>
</$soapHeader>
```

上記の SOAP メッセージを **$soapHeader** 階層属性の内容と結合する場合、その結果は以下の SOAP メッセージになります。

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

 <soapenv:Header
  xmlns:myns="http://www.my.com"
  xmlns:otherns="http://www.other.com">
```

```
  <myns:myheader />
  <otherns:otherheader>
   This is an example of a user-defined SOAP header.
  </otherns:otherheader>
 </soapenv:Header>

 <soapenv:Body><payload/></soapenv:Body>

</soapenv:Envelope>
```

$faultCode

コネクターが SOAP 障害応答を生成する場合は、必須属性です。

$faultCode 属性は、**$faultCodeNamespacePrefix** および **$faultCodeNamespaceURI** と組み合わせて使用するために設計されています。これら 3 つの属性を合わせると、修飾名を完全に定義できます。つまり、ネーム・スペースの URI ($faultCodeNamespaceURI)、ローカルの部分 ($faultCode)、およびネーム・スペースの接頭部 ($faultCodeNamespacePrefix) です。この修飾名は、SOAP 障害のコードを表します。

SOAP 1.2 を使用する場合、**$faultCode** 属性は修飾名のローカルの部分として使用されます。この属性は、**Code** 要素 (**Fault** 要素の子) の子である **Value** 要素の内部で確認できます。

例えば、**$faultCode** 属性にストリング「mycode」が含まれ、**$faultCodeNamespacePrefix** および **$faultCodeNamespaceURI** の属性が欠落している場合、SOAP メッセージの本体の内部の障害要素は、以下のとおりです。

```
<soapenv:Fault xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
 <soapenv:Code>
  <soapenv:Value>mycode</soapenv:Value>
 </soapenv:Code>
 ...
</soapenv:Fault>
```

一方、以下のように 3 つの属性をすべて設定した場合

```
$faultCodeNamespaceURI = http://www.w3.org/2003/05/soap-envelope
$faultCode = Sender
$faultCodeNamespacePrefix= env
```

SOAP 障害は以下のようになります。

```
<soapenv:Fault xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <soapenv:Code>
  <soapenv:Value>env:Sender</soapenv:Value>
 </soapenv:Code>
 ...
</soapenv:Fault>
```

SOAP 1.1 を使用する場合、**$faultCode** 属性は $faultcode 要素 (**Fault** 要素の子) の内容として使用されます。

例えば、以下の属性値を設定した場合

```
$faultCodeNamespaceURI = http://www.my.com
$faultCode = myfaultcode
$faultCodeNamespacePrefix = mypref
```

SOAP 本体の内部の障害要素は、以下のとおりです。

```
<soapenv:Fault xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:mypref="http://www.my.com">
 <faultcode>mypref:myfaultcode</faultcode>
 ...
</soapenv:Fault>
```

一般的に、独自の SOAP 障害コードを作成するよりも、以下のような事前定義の SOAP 障害コードをいくつか使用する方が好まれます。

http://www.w3.org/TR/soap12-part1/#soapfault (SOAP 1.2 用)
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383507 (SOAP 1.1 用)

$faultCodeNamespaceURI

このオプションの属性は、SOAP 障害コードのネーム・スペースの URI を表します。**$faultCode** 属性および **$faultCodeNamespacePrefix** 属性と組み合わせて使用されます。詳しくは、**$faultCode** 属性の説明を参照してください。

$faultCodeNamespacePrefix

このオプションの属性は、SOAP 障害コードのネーム・スペースの接頭部を表します。**$faultCode** 属性および **$faultCodeNamespaceURI** 属性と組み合わせて使用されます。詳しくは、**$faultCode** 属性の説明を参照してください。

$faultReason

コネクターが SOAP 障害応答を生成する場合は、必須属性です。

$faultReason 属性には、SOAP 障害を人間に分かりやすく記述した内容が含まれている必要があります。SOAP 1.2 を使用する場合、**$faultReason** 属性の値は、最初の Text 要素 (Fault 要素内の Reason 要素の子) の内容として使用されます。

例えば、**$faultReason** 属性を「myreason」に設定した場合、SOAP 障害要素は、以下のとおりです。

```
<soapenv:Fault xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  ...
  <soapenv:Reason>
   <soapenv:Text
    xml:lang="ja-jp"
    xmlns:xml="http://www.w3.org/XML/1998/namespace">
    myreason
   </soapenv:Text>
  </soapenv:Reason>
  ...
</soapenv:Fault>
```

言語は常に「en-US」に設定されていることに注意してください。SOAP 1.1 を使用する場合、**$faultReason** 属性の値は、**Fault** 要素内の **faultstring** 要素の内容として使用されます。

例えば、**$faultReason** 属性を「myreason」に設定した場合、SOAP 障害要素は、以下のとおりです。

```
<soapenv:Fault xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  ...
  <faultstring>myreason</faultstring>
  ...
</soapenv:Fault>
```

$faultNode

このオプションの属性は URI であり、障害を生成する SOAP ノードを表します。

$faultNode 属性は、SOAP 1.2 での **Node** 要素、および SOAP 1.1 での **faultactor** 要素に対応します。

$faultRole

このオプションの属性は URI であり、障害が発生した時点でノードが操作していた役割を表します。

$faultRole 属性は、SOAP 1.2 の **Role** 要素に対応します。

SOAP の役割について詳しくは、http://www.w3.org/TR/soap12-part1/#soaproles を参照してください。

SOAP 1.1 を使用する場合、この属性の値は無視されます。

$faultDetail

このオプションの階層属性は、障害を記述するアプリケーション固有の追加情報を表します。

$faultDetail 階層属性の最初の子要素は、SOAP 1.2 の **Detail** 要素、または SOAP 1.1 の **detail** 要素の内容として使用されます。

例えば、**$faultDetail** 階層属性が以下の XML 要素の DOM 表現である場合

```
<$faultDetail>
  <myfaultdata>some information here</myfaultdata>
</$faultDetail>
```

SOAP 障害要素は、以下のとおりです (SOAP 1.2 を使用)。

```
<soapenv:Fault xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  ...
  <soapenv:Detail>
   <myfaultdata>some information here</myfaultdata>
  </soapenv:Detail>
  ...
</soapenv:Fault>
```

SOAP 1.1 を使用すると、以下のようになります。

```
<soapenv:Fault xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  ...
  <detail>
   <myfaultdata>some information here</myfaultdata>
  </detail>
  ...
</soapenv:Fault>
```

構成

Axis2 Web サービス・サーバー・コネクターには、提供されたパラメーターがあります。

WSDL URL

Web サービスの記述が含まれる WSDL 文書のロケーション。コネクターは WSDL 文書に設定されているエンドポイント・アドレス情報を無視します。ポートのリスニング、SSL および HTTP 基本認証などの機能は、コネクター・パラメーターを使用してのみ制御できます。

サービス

WSDL 文書内のサービス記述の名前。このパラメーターには、関連付けられたスクリプトがあり、そのスクリプトは、WSDL 文書内の使用可能なサービスの記述をすべてリストするものです。スクリプトを最初に設定するには、**WSDL URL** パラメーターが必要です。

WSDL 文書に単一サービスの記述が含まれている場合、このパラメーターを空白にしておくことができます。一方、WSDL 文書に複数のサービスが記述されている場合、このパラメーターは必須です。

TCP ポート

着信要求を listen する TCP ポート (デフォルト・ポートは **80**)。

接続バックログ

着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

HTTP 基本認証 (BA)

使用可能にした場合 (デフォルトでは使用不可)、クライアントに対して HTTP 基本認証が実施されます。

認証レルム

HTTP 基本認証を要求するときにクライアントに送信する認証レルム。デフォルトは「IBM Security Directory Integrator」です。

SSL の使用

使用可能にすると (デフォルトでは使用不可)、コネクターはクライアントに対して SSL を使用するように要求します。非 SSL 接続要求は失敗します。

SSL を使用すると、コネクターはデフォルトの IBM Security Directory Integrator サーバー SSL 設定 (証明書、鍵ストア、およびトラストストア) を使用します。

クライアント認証を必要とする

使用可能にした場合 (デフォルトでは使用不可)、コネクターは SSL 使用時にクライアント認証を実施します。つまり、コネクターはクライアントに対し、構成済み *Directory Integrator の構成* トラストストアと突き合わせることができるクライアント・サイド SSL 証明書を提供するよう求めます。このパラメーターは、直前のパラメーター (SSL の使用) が使用可能に設定されている場合にのみ有効です。

WSDL ファイル生成プログラムのパラメーター

以下のパラメーターは、WSDL ファイルの生成に関連しています (ランタイム時には使用されません)。

WSDL 出力ファイル名

生成される WSDL ファイルの名前。このパラメーターは、WSDL ファイル生成ユーティリティーが実行されたときに生成される WSDL ファイルの名前を指定します。

Web サービス・プロバイダー URL

Web サービス・クライアントが Web サービス要求を送信する先のアドレス。この値は、WSDL 生成ユーティリティーによってのみ使用されます。

WSDL バージョン
生成される WSDL 文書のバージョン。以下の値から選択できます。

- **1.1** – WSDL 1.1 用
- **2.0** – WSDL 2.0 用

WSDL 生成
WSDL 生成ユーティリティーを実行するためのボタン。出力は「**WSDL 出力ファイル名**」パラメーターで指定されているファイルに送信されます。

セキュリティーと認証

セキュリティーと認証には 3 つのメソッドがあります。暗号化、認証、および許可です。ここに記載されている情報を通じてこれについて詳しく知ることができます。

暗号化

Axis2 Web サービス・サーバー・コネクターは、SSL/TLS の使用により、トランスポート・レベルのセキュリティーをサポートしています。暗号化の詳細については、以下に示す情報とリンクを通じて把握することができます。

SSL をオンにするには、「**SSL を使用**」コネクター・パラメーターを true に設定します。

SSL クライアント認証をオンにするには、「**クライアント認証を必要とする**」コネクター・パラメーターを true に設定します。

コネクター・パラメーターについて詳しくは、 39 ページの『構成』を参照してください。

認証

Axis2 Web サービス・サーバー・コネクターの HTTP 基本認証は、デフォルトで使用不可になっています。認証の詳細については、以下に示す情報とリンクを通じて把握することができます。

HTTP 基本認証をオンにするには、「**HTTP 基本認証 (BA)**」コネクター・パラメーターを true に設定します。また、「**認証レルム**」に認証レルムの名前を設定します。クライアントはそのレルムに対して認証するようプロンプトが出されます。

コネクター・パラメーターについて詳しくは、 39 ページの『構成』を参照してください。

以下のコネクター属性は、HTTP 基本認証に関連しています。

入力スキーマ

http.remote_user
HTTP クライアントの要求で指定されているユーザー名です。

http.remote_pass
HTTP クライアントの要求で指定されているパスワードです。

出力スキーマ

$authResult

認証プロセスの結果です。コネクターに関連付けられている AssemblyLine で、ユーザー定義の認証ロジックの結果に従ってこの属性を設定する必要があります。

実際の認証ロジックは、例えば、データベースまたは LDAP サーバーに対してクライアント信任状を確認することなどによって、関連付けられている Assembly Line でインプリメントされる必要があります。

許可

許可の詳細については、以下に示す情報とリンクを通じて把握することができます。

コネクターによって提供されているユーザー名 (41 ページの『認証』を参照) に基づき、コネクターの関連付けられている AssemblyLine のカスタム許可ロジックをインプリメントできます。

関連情報

TDI_install_dir/examples/axis2_web_services の例。

CCMDB コネクター

CMDB コネクターは、IBM Tivoli Change and Configuration Management Database (CCMDB) 内の構成アイテム (CI) および CI 間の関係の読み取り、書き込み、削除、または検索実行するのに使用できます。

注: このコネクターは推奨されません。IBM Security Directory Integrator の将来のバージョンでは削除されます。

CCMDB は、縦割りの組織間の連携を円滑にすることを目的として設計されたユーザー・インターフェースとワークフローを使用して、変更および構成管理プロセスの管理、監査、調整を行うための統合化生産性向上ツールおよびデータベースです。CCMDB には、多数のデータベースの論理的な集合体として機能し、IT インフラストラクチャー・リソースについての重要な情報を提供するデータベースが組み込まれています。提供される情報には、キー属性とその構成、他のインフラストラクチャー・リソースとの物理的関係および論理的関係などが含まれます。

CCMDB コネクターは、JDBC を使用してデータベースに接続し、DB2® データベースのみをサポートします。このコネクターは、queries.xml 構成ファイルを使用して静的 SQL ステートメントを組み込み、データをデータベースから取得するかデータベースに保管します。

CCMDB コネクターでは、階層データ・ソース・スキーマを使用して情報を表します。スキーマは、指定された成果物タイプ (実際の構成アイテムや関係など) および指定されたクラス・タイプによって異なります。

CCMDB について詳しくは、http://publib.boulder.ibm.com/infocenter/tivihelp/v10r1/topic/com.ibm.ccmdb.doc_7.1.1/overview/c_ccmdb_overview.html を参照してください。

CCMDB コネクターのアーキテクチャー

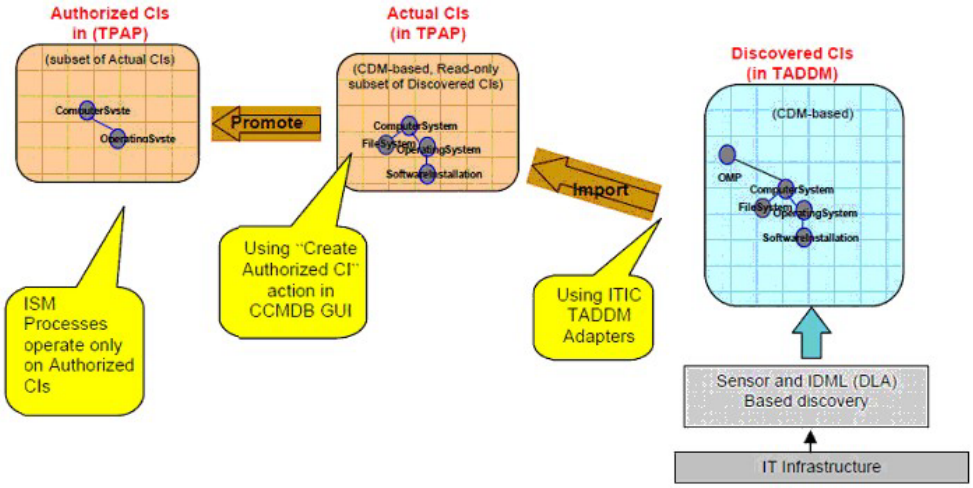
CCMDB コネクターのアーキテクチャーの処理には、以下に示す情報とリンクを使用します。

CCMDB データ層には、構成アイテム、プロセス成果物、およびこれらのオブジェクト間の関係を保持する 3 つのデータ・スペースが含まれます。このデータ層は、ディスカバーされた環境の依存関係マッピング、および以下を定義する許可された構成アイテムの仕様を提供します。

- 制御および管理する構成アイテムの特定の側面および特性
- 変更要求 (RFC) などのプロセス成果物との関係

CCMDB コネクターでは、これら 3 つのすべてのデータ・スペースで Common Data Model (CDM) がサポートされます。CDM は、整合性のあるデータ定義の共有と、顧客のビジネス環境における管理対象のリソースとコンポーネントに関するセキュリティー管理製品間でのデータ交換をサポートするために使用される論理的な情報モデルです。以下の図に、CCMDB ソリューションの 3 つのデータ・スペース、そのインターオペラビリティー、およびプロセス成果物などの他のデータ構造との関係を示します。

CDM について詳しくは、http://publib.boulder.ibm.com/infocenter/tivihelp/v10r1/index.jsp?topic=/com.ibm.taddm.doc_7.2/SDKDevGuide/c_cmdbsdk_understandingdatamodel.html を参照してください。



CCMDB コネクターは、現状の CI データ・スペースに保管された構成アイテムを処理します。現状の CI は、ディスカバーされた CI データ・スペース (現状の CI データ・スペースにコピーされている) の構成アイテムおよび関係のサブセットです。現状の CI データ・スペースでは、システムは、ディスカバーされた CI データ・スペースのデータのサブセットを処理します。システムには、構成管理または変更管理の一部として CI 監査などのプロセス管理とサービス管理のすべての機能を実行するためのデータが依然として含まれています。

データ表現モード

CCMDB コネクターでは、提供された 2 つのデータ表現モードがサポートされます。

- IdML モード - CDM 対応システム間での整合性のあるデータ転送用に作成された統一スキーマ
- ネイティブ・モード - 直接データ表現

構成エディターで、CCMDB コネクターで使用するデータ表現モードを選択できます。

IdML モード:

IdML は、データ表現モードの 1 つとして、以下に示す情報を通じて使用することができます。

IdML モードでは、すべての属性は CDM 名の先頭が大文字になり、cdm: という接頭部を付けて表現されます。すべての関係は、2 つの部分、つまり関係と関連項目で構成されます。関連クラスには、関係の方向を示す情報が含まれます。そのため、sys.ComputerSystem の関係 runson は、cdm-rel:runsOn . cdm-src:sys.OperatingSystem に変更されます。最初の部分 (cdm-rel:runsOn) は、接頭部 cdm-rel からもわかるように、関係を記述しています。2 番目の部分は、関連クラス・タイプ sys.OperatingSystem とその接頭部 (関連項目が関係のソースである場合) を表しています。

IdML モードの場合:

- すべての CI 属性は、cdm: 接頭部を使用します。
- すべての関係には、以下の 2 つの部分が含まれます。
  - cdm-rel: という接頭部が付いた関係名。
  - 接頭部 cdm-src: (項目が関係のソースである場合) または接頭部 cdm-trg: (項目が関係のターゲットである場合) が付いた関連構成アイテム

ネイティブ・モード:

ネイティブ・モードでは、すべての構成アイテムおよび関係は、内部データ・モデルに従って表されます。

コネクターは、構成アイテムの GUID を生成しません。入力データに指定された ID 値に依存します。

スキーマの比較:

ソフトウェアがインストールされたオペレーティング・システムについては、次に示すネイティブ・モードおよび IDML モードのデータ構造例を参照してください。

IdML モード	ネイティブ・モード

<pre>{
"@ClassType": "sys.OperatingSystem",
"@Guid": "46E8E8DE2946319190AF5B70BDCF4A60",
"cdm:Guid": "46E8E8DE2946319190AF5B70BDCF4A60",
"cdm:CreatedBy": "system",
"cdm:DisplayName": "192.168.1.1",
"cdm:LastModifiedBy": "system",
"cdm:LastModifiedTime": 1.222355157147E12,
"cdm:OSConfidence": 15.0,
"cdm:OSName": "D-Link embedded",
"cdm-rel:installedOn": {
"cdm-trg:sys.ComputerSystem": {
"@ClassType": "sys.ComputerSystem",
"@Guid": "272A0D2B9DE73C8A9C86740263CD39FA",
"cdm:Guid": "272A0D2B9DE73C8A9C86740263CD39FA",
"cdm:Fqdn": "192.168.1.1",
"cdm:Name": "192",
"cdm:Signature": "192.168.1.1",
"cdm:Type": "ComputerSystem",
"cdm:ContextIp": "NULL_CONTEXT",
"cdm:CreatedBy": "system",
"cdm:DisplayName": "192.168.1.1",
"cdm:LastModifiedBy": "system",
"cdm:LastModifiedTime": "1.222355157147E12"
}
},.....</pre>
	<pre>{
	"actciname": "192.168.1.1",
	"actcinum": "192.168.1.1~22335",
	"bidiflag": 3.0,
	"changeby": "SYSTEM",
	"changedate": "2008-09-25 11:05:57.0",
	"classification": "SYS.OPERATINGSYSTEM",
	"classstructureid": "1695",
	"createdby": "system",
	"displayname": "192.168.1.1",
	"guid": "46E8E8DE2946319190AF5B70BDCF4A60",
	"hasld": 0,
	"langcode": "EN",
	"lastmodifiedby": "system",
	"lastmodifiedtime": 1.222355157147E12,
	"lastscandt": "2008-09-25 11:05:57.0",
	"osconfidence": 15.0,
	"osname": "D-Link embedded",
	"installedon": {
	"relation": {
	"ancestorci": "192.168.1.1",
	"relationnum": "RELATION.INSTALLEDON",
	"sourceci": "192.168.1.1~22335",
	"sourceciguid":
	"46E8E8DE2946319190AF5B70BDCF4A60",
	"swapped": "1",
	"targetci": "192.168.1.1~22333",
	"targetciguid":
	"272A0D2B9DE73C8A9C86740263CD39FA",
	"target": {
	"actciname": "192.168.1.1",
	"actcinum": "192.168.1.1~22333",
	"changeby": "SYSTEM",
	"changedate": "2008-09-25 11:05:57.0",
	"classstructureid": "1625",
	"guid": "272A0D2B9DE73C8A9C86740263CD39FA",
	"hasld": "0",
	"langcode": "EN",
	"lastscandt": "2008-09-25 11:05:57.0",
	"fqdn": "192.168.1.1",
	"name": "192",
	"signature": "192.168.1.1",
	"type": "ComputerSystem",
	"bidiflag": "3.0",
	"contextip": "NULL_CONTEXT",
	"createdby": "system",
	"displayname": "192.168.1.1",
	"lastmodifiedby": "system",
	"lastmodifiedtime": "1.222355157147E12"
	}
	}
	},...</pre>

構成エディターの「**IdML モード**」オプションを使用することで、これらの 2 つのデータ表現モードを切り替えることができます。現在のスキーマの構造を確認するには、CCMDB コネクターの「*スキーマの照会*」機能を使用します。

installedOn 関係については、ネイティブ・モードおよび IDML モードの以下のデータ構造の例を参照してください。

IdML モード	ネイティブ・モード

<pre>{
"cdm-rel:installedOn": {
 "cdm-src:sys.zOS.ZOS": {
  "@ClassType": "sys.zOS.ZOS",
  "@Guid": "4E043C3C223B38B9AC647FE699B83365",
  "cdm:Guid": "4E043C3C223B38B9AC647FE699B83365",
  "cdm:CreatedBy": "system",
  "cdm:DisplayName": "OMO1",
  "cdm:Label": "OMO1-SYSPLEXO",
  "cdm:LastModifiedBy": "administrator",
  "cdm:LastModifiedTime": "1.176320919296E12",
  "cdm:SourceToken": "OMO1-ZOS",
  "cdm:FQDN": "PTHOMO1.PERTHAPC.AU.IBM.COM",
  "cdm:Name": "PTHOMO1.PERTHAPC.AU.IBM.COM",
  "cdm:OSName": "OMO1",
  "cdm:VersionString": "01.08.00",
  "cdm:IPLParmDataset": "SYS8.IPLPARM",
  "cdm:IPLParmDevice": "E81A",
  "cdm:IPLParmMember": "LOAD00",
  "cdm:IPLParmVolume": "$$SR81",
  "cdm:IPLTime": "1.174551129E12",
  "cdm:JESNode": "PTHAPO0",
  "cdm:MasterCatalogDataset":
  "CATALOG.MASTER.SYSPLEXO",
  "cdm:MasterCatalogVolume": "O$SY01",
  "cdm:NetID": "AUIBMQXP",
  "cdm:NetidSSCP": "AUIBMQXP.OMO1CDRM",
  "cdm:PrimaryJES": "JES2",
  "cdm:ProcessCapacityUnits": "3.0",
  "cdm:ProcessingCapacity": "52.0",
  "cdm:SMFID": "OMO1",
  "cdm:SSCP": "OMO1CDRM",
  "cdm:SysResVolume": "$$SR81"
 },
 "cdm-trg:sys.zOS.ZVMGuest": {
  "@ClassType": "sys.zOS.ZVMGuest",
  "@Guid": "A97257B6DA5434E69F2C47D34FC115ED",
  "cdm:Guid": "A97257B6DA5434E69F2C47D34FC115ED",
  "cdm:Name": "PTHOMO1",
  "cdm:ProcessCapacityUnits": "3.0",
  "cdm:ProcessingCapacity": "52.0",
  "cdm:Type": "IpDevice",
  "cdm:VMID": "PTHOMO1-PTHVM8",
  "cdm:CreatedBy": "administrator",
  "cdm:DisplayName": "PTHOMO1",
  "cdm:Label": "PTHOMO1-PTHVM8",
  "cdm:LastModifiedBy": "administrator",
  "cdm:LastModifiedTime": "1.176317254312E12",
  "cdm:SourceToken":
  "PTHOMO1-PTHVM8-ES64-PTHES6-VMGuest"
 }
}
}</pre>
	<pre>{
	"ancestorci": "1625",
	"relationnum": "RELATION.INSTALLEDON",
	"sourceci": "OMO1-SYSPLEXO~1828",
	"sourceciguid": "4E043C3C223B38B9AC647FE699B83365",
	"swapped": 0,
	"targetci": "PTHOMO1-PTHVM8~1829",
	"targetciguid": "A97257B6DA5434E69F2C47D34FC115ED",
	"source": {
	 "actciname": "OMO1-SYSPLEXO",
	 "actcinum": "OMO1-SYSPLEXO~1828",
	 "changeby": "ADMINISTRATOR",
	 "changedate": "2007-04-11 15:48:39.0",
	 "classstructureid": "1761",
	 "guid": "4E043C3C223B38B9AC647FE699B83365",
	 "lastscandt": "2007-04-11 15:48:39.0",
	 "createdby": "system",
	 "displayname": "OMO1",
	 "label": "OMO1-SYSPLEXO",
	 "lastmodifiedby": "administrator",
	 "lastmodifiedtime": "1.176320919296E12",
	 "sourcetoken": "OMO1-ZOS",
	 "fqdn": "PTHOMO1.PERTHAPC.AU.IBM.COM",
	 "name": "PTHOMO1.PERTHAPC.AU.IBM.COM",
	 "osname": "OMO1",
	 "versionstring": "01.08.00",
	},
	"target": {
	 "actciname": "PTHOMO1-PTHVM8",
	 "actcinum": "PTHOMO1-PTHVM8~1829",
	 "changeby": "ADMINISTRATOR",
	 "changedate": "2007-04-11 14:47:34.0",
	 "classstructureid": "1995",
	 "guid": "A97257B6DA5434E69F2C47D34FC115ED",
	 "lastscandt": "2007-04-11 14:47:34.0",
	 "name": "PTHOMO1",
	 "type": "IpDevice",
	 "createdby": "administrator",
	 "displayname": "PTHOMO1",
	 "label": "PTHOMO1-PTHVM8",
	 "lastmodifiedby": "administrator",
	 "lastmodifiedtime": "1.176317254312E12",
	 "sourcetoken":
	  "PTHOMO1-PTHVM8-ES64-PTHES6-VMGuest"
	}
	}</pre>

注:

1. このデータ構造の例では、単純化するために、いくつかのファイル・システム属性はスキップされています。
2. ネイティブ・モードでは、関係にはソース項目とターゲット項目しかないため、スキーマには、IdML モードよりも多くの属性があります (例えば、`relationnum`、`swapped`、など)。

CCMDB コネクターの動作モード

CCMDB コネクターがサポートしているのは、AddOnly、削除、更新、イテレーター、およびルックアップの各動作モードです。成果物タイプおよびスキーマ別のサポート対象動作モードについては、以下に示す表を参照してください。

	成果物タイプ			
モード	現状の CI (ネイティブ・スキーマ)	現状の CI (IdML スキーマ)	関係 (ネイティブ・スキーマ)	関係 (IdML スキーマ)
イテレーター	はい	はい	はい	はい
ルックアップ	はい	はい	はい	いいえ
AddOnly	はい	はい	はい	はい
更新	はい	はい	いいえ	いいえ
削除	はい	はい	はい	はい

注: コネクター動作モード、属性マッピング、およびリンク基準について詳しくは、「*Directory Integrator の構成*」で『一般概念』のセクションを参照してください。

AddOnly モード

構成アイテムまたは関係を追加するには、構成エディターで、属性の値を指定するための属性マッピングをセットアップします。

AddOnly 操作モードでは、「項目」で使用可能な属性を使用して、現状の構成アイテムまたは関係が作成されて、データベースに挿入されます。

以下の表で、AddOnly モードでの操作について説明します。

表 4. 現状の構成アイテム

成果物	操作
ルート構成アイテム	項目が存在しない場合は項目をデータベースに追加し、存在する場合は例外をスローします。
関係	関係を追加します。
関連構成アイテム	項目が存在する場合はスキップします。 存在しない項目を追加します。

表 5. 関係

成果物	操作
関係	指定された関係が存在しない場合は、その関係をデータベースに追加し、存在する場合は例外をスローします。
関連構成アイテム	項目が存在する場合はスキップします。 存在しない項目を追加します。

更新モード

構成アイテムまたは関係を更新するには、構成エディターで、更新する属性の値を提供するためのリンク基準を指定して出力属性マッピングをセットアップします。

更新モードでは、「項目」で使用可能な属性を使用して、現状の構成アイテムまたは関係がデータベースで更新されます。

以下の表で、更新モードでの操作について説明します。

表 6. 現状の構成アイテム

成果物	操作
ルート構成アイテム	項目を更新します。
関係	関係を上書きします。
関連構成アイテム	項目が存在する場合はスキップします。 存在しない項目を追加します。

削除モード

構成アイテムまたは関係を削除するには、構成エディターで、属性のリンク基準および入力属性マッピングを指定します。

削除モードでは、「項目」で使用可能な属性を使用して、現状の構成アイテムまたは関係がデータベースから削除されます。

以下の表で、削除モードでの操作について説明します。

表 7. 現状の構成アイテム

成果物	操作
ルート構成アイテム	指定項目をデータベースから削除します。
関係	データベースから関係を削除します。
関連構成アイテム	削除操作をスキップします。

表 8. 関係

成果物	操作
関係	指定された関係をデータベースから削除します。
関連構成アイテム	削除操作をスキップします。

イテレーター・モード

現状の構成アイテムと、構成エディターで指定したクラス・タイプでの当該構成アイテム間の関係を読み取るのには、CCMDB コネクターが使用できます。

ルックアップ・モード

CCMDB コネクターは、一致する属性を検索するのに使用します。

必要な構成アイテム/関係クラスに基づいて、メソッド呼び出しに渡された選択基準に関係するタイプの項目が構成されます。

注: IdML スキーマは、関係レベルで属性を定義しません。

構成

CCMDB コネクターの構成パラメーターは、以下に示すように使用することができます。

成果物タイプ
このパラメーターを使用して、CCMDB コネクターで処理するリソース・タイプ (**構成アイテム**または**関係**) を指定します。

クラス・タイプ
このパラメーターを使用して、処理される構成アイテムまたは関係のタイプを指定します。

サポートされるクラスを選択するには、構成エディターで「**選択...**」ボタンをクリックします。

IdML モード
このパラメーターを使用して、IdML 互換データを使用して処理を行うかどうかを指定します。

ソース関係
このパラメーターを使用して、読み取り項目のソース関係をロードするかどうかを指定します。

ターゲット関係
このパラメーターを使用して、読み取り項目のターゲット関係をロードするかどうかを指定します。

JDBC URL
このパラメーターを使用して、データベースに接続するための JDBC URL を指定します。

JDBC ドライバー
このパラメーターを使用して、データベースに接続するための JDBC ドライバーを指定します。

ユーザー名
このパラメーターを使用して、データベースにログインするための有効なユーザー ID を指定します。

パスワード
このパラメーターを使用して、ユーザー ID に関連するパスワードを指定します。

コメント
このパラメーターを使用して、コメントを追加します。データの構文解析時には、このコメントは無視されます。

詳細ログ
このパラメーターを使用して、詳細なログ・メッセージを生成します。

例

例を読むには、以下に示すパスを使用します。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/CCMDBConnector ディレクトリーにあります。

コマンド行コネクター

コマンド行コネクターを使用すると、コマンド行から出力を読み取ること、またはコマンド行の標準入力にデータをパイピングすることができます。コマンド引数はそれぞれスペース文字で区切られ、引用符は無視されます。コマンドはローカル・マシンで実行されます。

注: 独立したシェルが取得されるのではないため、リダイレクト文字 (| > など) は無効です。リダイレクトを使用するには、適切なパラメーター・セットを指定したシェル・スクリプト (UNIX) またはバッチ・コマンド (DOS) を作成してください。例えば、Windows システムでは次のように入力します。

```
cmd /c dir
```

ディレクトリーの内容がリストされます。

コネクターでは、イテレーター・モード、AddOnly モード、および CallReply モードがサポートされます。

イテレーター・モードと AddOnly モードでは、「**コマンド行**」パラメーターに指定されたコマンドが、コネクター初期化中にターゲット・システムに発行されます。これは、このコマンドが AssemblyLine の存続期間において 1 回のみ発行されることを意味します。

ただし CallReply モードでは、出力属性マッピング (呼び出しフェーズ) の後から入力属性マッピング (応答フェーズ) の前までの間において、AssemblyLine を反復するたびにこのコマンドがターゲット・システムに発行されます。このモードでは、**command.line** 属性に、実行するコマンドを指定する必要があります。コマンドの実行後、**command.output** 属性で出力結果を確認できます。

コマンド行コネクターにパーサーが接続している場合は、この結果出力が解析されます。

一部のオペレーティング・システムでのネイティブ・エンコード出力

一部のオペレーティング・システムでのネイティブ・エンコード出力については、以下に示す情報を通じて把握することができます。

コマンド行コネクターを使用して Windows オペレーティング・システムでプログラムを実行すると、プログラムの出力が DOS コード・ページを使用してエンコードされることがあります。通常、Windows プログラムは Windows コード・ページを使用するため、これによって予期しない結果を引き起こす場合があります。 DOS コード・ページは Windows コード・ページとは異なるため、コマンド行コネクターのパーサーの文字エンコードを、ユーザーの地域の正しい DOS コード・ページ (例: `cp850`) に設定する必要がある場合があります。

423 ページの『文字エンコード変換』も参照してください。

引用符で囲む場合のワード

Linux/Unix システムでは、字句的に重要な文字を含む可能性があるパラメーターを引用符で囲む処理を実行する機能がこのコネクターに組み込まれています。ここに記載されている情報を通じてこれについて詳しく知ることができます。

「**sh の使用**」パラメーターをチェックすると、IBM Security Directory Integrator では sh プログラム (標準 Linux シェルなど) を使用してコマンド行が実行され、sh により引用符で囲む処理が実行されます。オペレーティング・システムに sh がない場合は、このボックスをチェックしないでください。

sh を使用しない場合、Unix/Linux プラットフォーム上でコマンド行コネクターを実行すると、このコネクターでは引用符で囲まれているパラメーターが指定されているコマンド行が正しく処理されません。次のコマンドを例に説明します。

```
Report -view fileView -raw -where "releaseName = 'ibmdi_60'
     and nuPathName like 'src/com/ibm/di%' "
```

このコマンドでは `"releaseName = 'ibmdi_60' and nuPathName like 'src/com/ibm/di%' "` という句を 1 つのパラメーターとして指定する必要がありますが、指定されていません。これは、IBM Security Directory Integrator が Java ランタイムの exec() メソッドを使用して、すべてのコマンドをスペースで分割し、すべての引用符を無視するためです。引用符に基づいて分割するようにする必要があります。(可能であれば)「**sh の使用**」をチェックすると、この問題が解決します。

構成

このコネクターは、リストされたパラメーターを必要とします。

コマンド行
: 実行するコマンド行。イテレーター・モードと AddOnly モードのみで使用されます。

sh の使用
: 使用可能に設定されている場合 (デフォルトでは使用不可)、コネクターに対して sh のような解析を使用するよう指示します。このパラメーターが true に設定されている場合は特に、コネクターは、二重引用符で囲まれ、スペースが含まれているコマンド行引数を正しく解析できます。

  この機能は、「sh」シェル・コマンド・インタープリターを備えたオペレーティング・システム (通常、UNIX 型のオペレーティング・システム) でのみ使用可能です。

詳細ログ
: このフィールドをチェックすると、追加のログ・メッセージが生成されます。

パーサー
: 項目の解釈または生成を行うパーサー。

例

コマンド行コネクターの例を参照するには、以下に示すパスとリンクを使用します。

IBM Security Directory Integrator インストール済み環境の *TDI_install_dir*/examples/commandLine_connector ディレクトリーを参照してください。

関連情報

557 ページの『リモート・コマンド行関数コンポーネント』

データベース・コネクター

データベース・コネクターは、データベースの構成に、より使いやすいユーザー・インターフェースを提供することを目的としています。ここに記載されている情報とリンクを使用することで、データベース・コネクターについて詳しく知ることができます。

データベース・コネクターは、181 ページの『JDBC コネクター』を単純化したバージョンです。詳細パラメーターはコネクター構成フォームからすべて削除され、最もよく使用されるパラメーターのみが使用可能となります。

コネクター・モードなどの動作については、181 ページの『JDBC コネクター』の該当セクションを参照してください。 特に、データベース・システムへの接続を設定できるようにするためのドライバー・ライブラリーの配置先については、182 ページの『JDBC ドライバーについて』を参照してください。ただし、データベース・コネクターは JDBC URL 作成プロセスの一部を自動化します。

構成

データベース・コネクターは、提供されたパラメーターを必要とします。

データベース・タイプ
このセクションでは、ホスト名、ポート、およびデータベース・パラメーターの入力が求められます。データベース・タイプのドロップダウン選択に基づいて、「`jdbcDriver`」および「`jdbcSource`」パラメーターが生成されます。

ユーザー名
このユーザー名を使用してデータベースにサインオンします。このユーザーからアクセス可能な表のみが表示されます。これは、JDBC コネクターの「`jdbcuser`」パラメーターを反映します。

パスワード
このユーザーのサインオンに使用するパスワード。

スキーマ
使用するデータベースの表内のスキーマ。ブランクのままにすると、jdbcLogin (「**ユーザー名**」パラメーター) の値が使用されます。

注: IBM Security Directory Integrator の資料では、アクセスしているオブジェクトのデータ定義を意味する用語として「スキーマ」が使用されています。ただし、RDBMS におけるスキーマという用語には、1 つの ID (ユーザー名) でグループ化されたデータ定義、表、およびオブジェクトのコレクション全体という別の意味があります。この特定のコネクターにおける特定のパラメーターについては、RDBMS におけるスキーマの意味で使用しています。

表名 このコネクターでアクセスする表の名前。データベースへのサインオンが可能な場合、「**選択**」ボタンを使用して、アクセス可能な表のデータベースを照会できます。

テーブルの自動作成

コネクターがいずれかの出力モード (AddOnly または更新) で構成されている場合は、詳細セクションに以下の追加オプションがあります。

「**テーブルの自動作成**」オプションでは、コネクターの属性マップおよびスキーマに基づいて、コネクターによりシンプルなテーブルが作成されます。これは、データベースにテーブルが存在しない場合のみ実行されます。

テーブルの自動作成時、コネクターは最初に属性マップから列名を派生させます。属性マップが空の場合は、スキーマを使用して列名のリストを取得します。列名が決まると、各列名で SQL *CREATE TABLE* ステートメントが生成されます。スキーマに列名の定義がある場合、列の構文を決定するために使用されます。スキーマには、列の構文を決定する 2 つのパートがあります。最初に、「ネイティブ構文」が指定されている場合は、それがそのまま使用されます。次に、提供されるネイティブ・スキーマがない場合、コネクターは「Java クラス」を使用して構文を派生させます。スキーマの「Java クラス」フィールドに、以下の値のいずれかを指定します。

表 9. Java クラスから SQL タイプへのマッピング

値	生成される SQL タイプ
整数または java.lang.Integer	INT
ストリングまたは java.lang.String	VARCHAR(255)
Double または java.lang.Double	DOUBLE
日付または java.util.Date	TIMESTAMP

列に関するスキーマ情報がない場合、または値が認識されない場合、コネクターは生成された「表の作成ステートメント」で「VARCHAR(255)」を使用します。

導入済み資産コネクター

データ統合についての導入済み資産コネクターは、IBM Tivoli Asset Management for IT データベースで使用することができます。IBM Tivoli Asset Management for IT により、IT 環境を効果的かつ効率的に管理することができます。

デプロイ済み資産コネクターは、効率的なデータの変換および検証のために、JDBC を使用してデータベースに接続します。デプロイ済み資産コネクターの主要な機能は、以下のとおりです。

- デプロイ済み資産データをデータベースに追加します。
- デプロイ済み資産データをデータベースから取得します。
- 指定した検索基準に一致するデプロイ済み資産データを検索して取得します。
- デプロイ済み資産データをデータベースから削除します。
- データ統合のために、デプロイ済み IT 資産タイプ (コンピューター、プリンター、またはネットワーク・デバイス) を選択できます。

注: 現在サポートされているのは DB2 データベースのみです。

デプロイ済み資産コネクターの使用

導入済み資産コネクターのアーキテクチャー、サポートされる動作モード、および実行については、以下に示す情報を使用して把握することができます。

デプロイ済み資産コネクターのアーキテクチャー

導入済み資産コネクターのアーキテクチャーについては、以下に示す情報を通じて、詳細な洞察を取得できます。

デプロイ済み資産コネクターでは、デプロイ済み IT 資産データは、階層的に示されます。例えば、タイプ「コンピューター」の資産には、Domainname、Biosname、Ramsize などの単純な属性を含めることができます。また、以下の階層データ・モデルに表すように、1 つ以上の関連 CPU を含めることもできます。

```
{
"Assetclass": "COMPUTER",
"Biosdate": "1998-03-26 00:00:00.0",
"Biosname": "Intel 4A4LL0X0.10A.0027.P09",
"Biospnp": 0,
"Biosversion": "4A4LL0X0.10A.0027.P09",
"Changedate": "2005-02-03 14:23:16.0",
"Createdate": "2005-02-03 14:23:16.0",
"Description": "KLINGON_EMPIRE administrator enterprise 00400542C346
     192.168.100.95/24",
"Domainname": "enterprise",
"Hwdetectiontool": "Maximo Discovery 4.5",
"Hwlastscandate": "2003-06-26 16:52:00.0",
"Logonname": "administrator",
"Makemodel": "Dell Dimension XPS D266",
"Manufacturer": "Dell",
"Nodename": "KLINGON_EMPIRE 00400542C346",
"Ramdescription": "Total Ram",
"Ramsize": 64.00,
"Ramtotalslots": 0,
"Ramunit": "MB",
"Ramunusedslots": 0,
"Serialnumber": "NB7V7",
"Smbios": 0,
"Supportssnmp": 1,
"Supportswmi": 1,
"Swdetectiontool": "Maximo Discovery 4.5",
"Systemrole": "Network PC",
"CPUList": {
 "Processor": {
  "Changedate": "2005-02-03 14:23:17.0",
  "Createdate": "2005-02-03 14:23:17.0",
  "Currspeed": "0.00",
  "Makemodel": "Pentium II",
  "Manufacturer": "Intel",
  "Maxspeed": "266.00",
  "Serialnumber": "Source ID: 963",
  "Speedunit": "MHz",
  "CPUVariant": {
   "Processorname": "Pentium II",
   "Processorvar": "Pentium II",
  },
  "manufacturer-info": {
   "Manufacturername": "Intel",
   "Manufacturervar": "Intel",

  }
```

```
 }
},
....
}
```

デプロイ済み資産コネクターの動作モード

導入済み資産コネクターでは、以下に示す動作モードをサポートしています。

- AddOnly モード
- イテレーター・モード
- ルックアップ・モード
- 削除モード

「属性マップ」および「リンク基準」の使用について詳しくは、「*Directory Integrator の構成*」のセクション『一般概念』を参照してください。

AddOnly モード:

AddOnly モードでは、デプロイ済み資産コネクターを使用して、デプロイ済み資産データをデータベースに追加できます。

指定される資産タイプ (コンピューター、ネットワーク・デバイス、またはネットワーク・プリンター) について、コネクターの「出力マップ」で属性をマップする必要があります。

イテレーター・モード:

イテレーター・モードでは、デプロイ済み資産コネクターを使用して、指定した資産タイプ (コンピューター、ネットワーク・デバイス、ネットワーク・プリンターなど) のルート・レベルの資産およびその関係を読み取ることができます。

ルックアップ・モード:

ルックアップ・モードでは、デプロイ済み資産コネクターを使用して、一致する資産を検索できます。

資産を検索するには、選択した資産タイプの属性を「リンク基準」として指定します。例えば、NodeID が 100 の資産を検索するには、`NodeID equals 100` を「リンク基準」として設定します。

削除モード:

削除モードでは、デプロイ済み資産コネクターを使用して、既存の資産をデータベースから削除できます。資産を削除するには、選択した資産タイプの属性を「リンク基準」として指定します。

構成

導入済み資産コネクターには、以下に示すパラメーターが使用できます。

資産タイプ

このパラメーターを使用して、デプロイ済み IT 資産タイプ (コンピューター、ネットワーク・デバイス、ネットワーク・プリンターなど) を指定します。

JDBC URL
このパラメーターを使用して、データベースに接続するための JDBC URL を指定します。例えば jdbc:db2://10.1.1.1:50005/MAXDB71。

JDBC ドライバー
このパラメーターを使用して、データベースに接続するための JDBC ドライバーを指定します。

ユーザー名
このパラメーターを使用して、データベースにログインするための有効なユーザー ID を指定します。例えば、IBM Tivoli Asset Management for IT のデフォルトのインストール済み環境では maximo を使用します。

パスワード
このパラメーターを使用して、ユーザー ID に関連するパスワードを指定します。

参照のロード
ルート・レベルの資産のみをロードする必要があるかどうかを示します。

例

導入済み資産コネクターの例を参照するには、以下に示すパスとリンクを使用します。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/DPAConnector ディレクトリーにあります。

TCP/URL の直接スクリプト記述

コネクターを使用せずに、URL オブジェクトまたは TCP ポートに直接アクセスすることもできます。プロローグに挿入できるコードの例を以下に示します。

TCP

TCP ポートに直接アクセスするためのプロローグに挿入できるコードの例を以下に示します。

```
// This example creates a TCP connection to www.example_page_only.com
  and asks for a bad page

var tcp = new java.net.Socket ( "www.example_page_only.com", 80 );
var inp = new java.io.BufferedReader ( new java.io.InputStreamReader
  ( tcp.getInputStream() ) );
var out = new java.io.BufferedWriter ( new java.io.OutputStreamWriter
  ( tcp.getOutputStream() ) );

task.logmsg ("Connected to server");

// Ask for a bad page
out.write ("GET /smucky¥r¥n");
out.write ("¥r¥n");

// When using buffered writers always call flush to make sure data
  is sent on connection
out.flush ();

task.logmsg ("Wait for response");
```

```
var response = inp.readLine ();

task.logmsg ( "Server said: " + response );
```

URL

URL ポートに直接アクセスするためのプロローグに挿入できるコードの例を以下に示します。

```
// This example uses the java.net.URL object instead of the raw
  TCP socket object

var url = new java.net.URL("http://www.example_page_only.com");
var obj = url.getContent();

var inp = new java.io.BufferedReader ( new java.io.InputStreamReader
  ( obj ) );
while ( ( str = inp.readLine() ) != null ) {
task.logmsg ( str );
}
```

Domino/Lotus Notes コネクター

以下に示す情報とリンクを使用することで、IBM Domino/Lotus Notes コネクターの接続およびインストールができるようになります。

Domino Server または Notes システムに接続するには、使用可能な接続タイプ (Notes の用語では「セッション・タイプ」) について検討します。これらのコネクターを操作するには、 Domino/Notes のクライアント・ライブラリーをインストールする必要があり、どのクライアント・ライブラリーをインストールするか (61 ページの『インストール後の構成』も参照) の決定は、必要なセッション・タイプによって決まります。

セッション・タイプ

セッション・タイプの詳細については、以下に示す情報を使用して把握することができます。

ローカル・クライアント・セッション

ローカル・クライアント・セッションによる Lotus Domino Server の呼び出しは、Notes ユーザー ID に基づいています。

Notes Client をローカルにインストールする必要があります。このセッション・タイプでは、Notes.jar ファイルが *TDI_install_dir*/jars/3rdparty/IBM フォルダー内に存在しており、ローカル・クライアント・バイナリーが PATH システム環境変数に指定されている必要があります。

ローカル・サーバー・セッション (Domino ローカル・セッション)

このタイプのセッションの作成時には、ローカル・サーバーの ID ファイルが使用されます。

セッションを作成する Notes API メソッドの host パラメーターは NULL でなければなりません。セッション作成メソッドの NULL サーバー・パラメーターなどの現行サーバーへの参照は、ローカルの IBM Domino 環境が指示されていることを意味します。ローカル・クライアント・セッションを作成する場合は、user パラメーターも NULL にする必要があります。これにより、Notes ユーザー ID を使用することが指示されます。

ローカル・サーバーはセッション作成時にのみ使用されます。ただし、ローカル環境に接続しているサーバーの名前を指定することで、これらのサーバーにアクセスできます。この名前は、lotus.domino.Session クラスの *getDatabase* メソッドの 1 番目のパラメーターとして指し示されます。

ローカル・サーバー・セッションを使用するには、IBM Security Directory Integrator がインストールされているマシンに Domino Server をインストールする必要があります。また、このサーバー・インストールへのパスも、PATH システム環境変数に追加されている必要があります。

IIOP セッションと IOR パラメーター

IIOP セッションはネットワーク・ベースのセッションであり、リモート Domino Server によってクライアント要求が処理されます。

IIOP セッションが指定されている場合は、コネクターは認証に IBM Domino ユーザー名およびそのユーザーのインターネット・パスワードを使用します。ユーザーのユーザー名とインターネット・パスワードは、コネクターのパラメーターです。システムのローカル・ユーザー ID と必ずしも同一ユーザーである必要はありません。

IOR は、Domino Java API が Domino Server との IIOP セッションを確立するために必要なテキスト・ストリングです。 Lotus Notes コネクターと同様、クライアントはストリング IOR をデコードして、それをリモート・セッションの確立に使用します。これは、diiop_ior.txt と呼ばれるファイルに含まれています。

以下のような変更によって、IOR ストリングが不整合になる場合があります。

- DIIOP ポート番号の変更
- DIIOP ポートの使用可能化または使用不可化
- TCP/IP アドレスの変更

IIOP セッションを作成するには、次の 2 とおりの方法があります。

IOR ストリングを明示的に指定する

IBM Security Directory Integrator 6.0 Domino 変更検出コネクターが使用するセッション作成メソッドは、Domino HTTP タスクから IOR ストリングを取得します。IBM Security Directory Integrator Domino/Lotus コネクターの現行バージョンでは、パラメーター「**IOR ストリング**」は外部化されています。このパラメーターはオプションです。このパラメーターが指定されていないか、または値が指定されていない場合は、IBM Security Directory Integrator 6.0 と同様の方法で IIOP セッションが作成されます。コネクター構成にこのパラメーターが指定されている場合は、Domino Java API の以下のメソッドがセッションの作成に使用されます。

```
static public Session createSessionWithIOR(String IOR,
        String user, String passwd)
        throws NotesException

    static public Session createSessionWithIOR(String IOR,
        String args[], String user, String passwd)
        throws NotesException
```

このコネクター・パラメーターを指定すると、次に説明する 2 つの点でコネクターが向上します。

- コネクターが機能するようにする目的で IBM Domino HTTP タスクを実行する必要がなくなるため、コネクターのセットアップの要件が減ります。
- コネクターが機能するのは、IBM Domino HTTP タスクが SSL ポートを使用するよう構成されている場合のみです。

HTTP タスクから IOR ストリングを取得する

この方法では、HTTP タスクを使用して Domino Server から IOR ストリングを取得するため、コネクターが「**HTTP ポート**」パラメーターを使用します。IBM Domino Server へのセッションを作成するためにコネクターがローカル・クライアントを使用する場合は、このポートは考慮されません。

IIOP セッションの作成時に、SSL が使用されることがあります。コネクターは最初に IOR ストリングを HTTP タスクから取得することを試行し、その後、ホスト・パラメーターの代わりに、取得した IOR を指定することによって、ストリングの配列をパラメーターとして受け入れるセッション作成メソッドの使用を試行します。

SSL が使用されている場合、コネクターは次に示すメソッドを使用してセッション作成を試行します。

```
static public Session createSessionWithIOR(String ior,
     String user, String passwd)
      throws NotesException
```

SSL が使用されていない場合は、コネクターは次に示すメソッドを使用してセッション作成を試行します。

```
static public Session createSession(String host, String args[],
     String user, String password)
     throws NotesException
```

この場合「**HTTP ポート**」パラメーターの値がホストに追加されます。このメソッドは、このポートで実行される Lotus Domino HTTP タスクから IOR ストリングを取得しようとします。このタスクでは、このポートを SSL 実行ポートとして使用することはできません。

次に示すセッション・タイプでは `ncso.jar` ファイルが *TDI_install_dir*`/jars/3rdparty/IBM` フォルダー内に存在しており、ローカル・サーバー・バイナリーが PATH システム環境変数に指定されている必要があります。

コネクター別のサポートされるセッション・タイプ

表 10. サポートされる Lotus Domino/Lotus Notes セッション・タイプ

サポートされるセッション ► コネクター ■	ローカル・クライアント・セッション	ローカル・サーバー・セッション	IIOP セッション
Domino 変更検出コネクター	はい	いいえ	はい
Lotus Domino ユーザー・コネクター	はい	はい	はい

表 10. サポートされる *Lotus Domino/Lotus Notes* セッション・タイプ *(続き)*

Lotus Notes コネクター	はい	はい	はい
Lotus Domino AdminP コネクター	いいえ	はい	はい

注: SSL 用の IBM Domino API は JSSE 準拠ではなく、IBM Domino に固有のものです。つまり、Lotus Domino 変更検出コネクターの SSL 構成では IBM Security Directory Integrator トラストストアと鍵ストアは使用されません。Domino 変更検出コネクターの SSL 構成では、DIIOP プロセスが (Domino Server で) 開始されるたびに生成される `TrustedCerts.class` ファイルが、IBM Security Directory Integrator のクラスパス (ibmditk または ibmdisrv) に指定されている必要があります。クラスパスに含まれるローカル・パスに `TrustedCerts.class` をコピーするか、またはインストールされている Domino の `Lotus¥Domino¥Data¥Domino¥Java` をクラスパスに含める必要があります。

ファイルは、`ncso.jar` ファイルをロードするのと同じクラス・ローダーによってロードされる必要があります。 `ncso.jar` ファイルが IBM Security Directory Integrator のシステム・クラス・ローダーによってロードされるため、`TrustedCerts.class` ファイルはシステム・クラス・ローダーによってロードされる必要があります。「classes」フォルダーの `TrustedCerts.class` ファイルを廃棄することによって、このことを簡単に実行できます。詳しくは、65 ページの『「classes」フォルダー』を参照してください。

64 ビット版のオペレーティング・システムでのローカル・セッション

一般的に、32/64 ビット版 JVM を使用する IBM Security Directory Integrator は、対応する 32/64 ビット版の Domino/Notes インストールを使用してのみ、ローカル・セッション (クライアントまたはサーバー) を確立できます。

	32 ビット版 Lotus Domino/Notes	64 ビット版 Lotus Domino*
32 ビット版 JVM	はい	いいえ
64 ビット版 JVM	いいえ	はい

(*) 現時点 (バージョン 8.5) では、Lotus は 64 ビット版の IBM Notes クライアントを提供していません。

ローカル・クライアントおよびローカル・サーバーのセッションでは、Lotus の Java API (Notes.jar) が使用されます。この Java API は、Notes/Domino インストールのネイティブ・ライブラリーに依存しています (このため、IBM Security Directory Integrator と同じマシンに Notes Client またはDomino Server のいずれかがインストールされている必要があります)。ほとんどの最新のオペレーティング・システムでは、32 ビット版または 64 ビット版のいずれかのネイティブ・バイナリーを使用するプロセスを許可していますが、両方は許可していません。32 ビット版の実行可能ファイルを使用してプロセスを開始する場合、そのプロセスでは、32 ビット版のネイティブ・ライブラリーのみを使用できます。

ほとんどの 64 ビット版のプラットフォームでは、32 ビット版のアプリケーションをインストールして使用できます。(64 ビット版のオペレーティン

グ・システム上で) 32 ビット版の Domino Server (Notes Client) へのローカル・サーバー (ローカル・クライアント) セッションを使用する場合、32 ビット版の JVM を使用した IBM Security Directory Integrator を使用する必要があります。

注: Lotus Domino/Notes コネクターは、Lotus Domino R8 および Lotus Domino R8.5.x でサポートされています。

インストール後の構成

IBM Security Directory Integrator をインストールした後に、いくつかの構成ステップを実行して、Lotus Notes/Domino コネクターを実行できるようにする必要があります。これについては、ここに記載されている情報を参照することで、詳しく知ることができます。

(コネクターが使用される) Lotus Domino Server または Lotus Notes Client のバージョンがサポートされていることを確認します。

コネクターが Lotus Notes Client マシンにデプロイされている場合は、Lotus Notes Client マシンでのみこれらのステップを実行する必要があり、Lotus Notes Client が接続されている Lotus Domino Server マシンでは実行しません。

コネクターが Lotus Domino Server マシンにデプロイされている場合は、その Lotus Domino Server マシンでこれらのステップを実行する必要があります。

Lotus には、notes.jar という名前の Java ライブラリーが付属します。このライブラリーは、IBM Domino Server にアクセスするネイティブ呼び出しへのインターフェースを備えています (ネットワークを使用する場合もあります)。このライブラリーは、Domino Server または Notes クライアントがインストールされているフォルダーにあります (C:¥Lotus¥Domino、C:¥Lotus¥Notes など)。 Lotus Domino と Lotus Notes ではバイナリーが異なるため、ローカル・クライアント・セッションまたはローカル・サーバー・セッションのいずれを作成するのかによって設定を変える必要があります。

ローカル・クライアント・セッションの作成

ここに記載されている手順を使用することで、ローカル・クライアント・セッションを作成できるようになります。

- *TDI_install_dir*/jars/3rdparty/IBM または global.properties (または solution.properties) の com.ibm.di.loader.userjars プロパティーで指定されているフォルダーに notes.jar をコピーします。
- *TDI_install_dir*/jars フォルダーおよびそのサブフォルダーに ncso.jar ファイルがないことを確認してください。
- ibmditk (または ibmditk.bat) および ibmdisrv (または ibmdisrv.bat) シェル・スクリプト内の PATH 環境変数にローカル Lotus Notes バイナリーのパスを追加します (C:¥Lotus¥Domino、C:¥Lotus¥Notes)。次に、2 つのシェル・スクリプトの一番下の Java 実行可能ファイルに提供されているパラメーターのリストに以下のパラメーターを追加します。

  ```
  -Djava.library.path="%PATH%"
  ```

- 以下の例外が発生することがあります。

```
CTGDJE010E Connector was unable to initialize local Notes session to Domino Server.
Exception encountered: java.lang.Exception: Native call SECKFMSwitchToIDFile failed with error:
     code 259, 'File does not exist'
```

Lotus Domino Server への認証に使用される Notes ID ファイルを IBM Security Directory Integrator のソリューション・ディレクトリーにコピーする必要があります。通常、このファイルは、*Notes_install_dir*/Data/ フォルダーにあります。この問題の原因は、notes.ini ファイルが ID ファイルを指す方法にあります。絶対パスではなく相対パスが使用されている (KeyFilename=user.id) ため、コンポーネントは、Lotus Notes がインストールされているデータ・フォルダーではなく現行ディレクトリーである IBM Security Directory Integrator のソリューション・ディレクトリーを検索します。

ローカル・サーバー・セッションの作成

ローカル・サーバー・セッションを作成する場合は、以下に示す手順を実行します。

- *TDI_install_dir*/jars フォルダーおよびそのサブフォルダーに ncso.jar ファイルがないことを確認してください。
- *TDI_install_dir*/jars/3rdparty/IBM または global.properties (または solution.properties) の com.ibm.di.loader.userjars プロパティーで指定されているフォルダーに notes.jar をコピーします。
- ibmditk (または ibmditk.bat) および ibmdisrv (または ibmdisrv.bat) シェル・スクリプト内の PATH 環境変数にローカル Lotus Domino バイナリーのパス (C:¥Lotus¥Domino など) を追加します。次に、2 つのシェル・スクリプトの一番下の Java 実行可能ファイルに提供されているパラメーターのリストに以下のパラメーターを追加します。

  ```
  -Djava.library.path="%PATH%"
  ```

注: Notes API の実装方法により、*TDI_install_dir*/jars/3rdparty/IBM には ncso.jar または notes.jar のいずれか 1 つの jar のみが存在します。両方ある場合は、コネクターの動作が予測不能になります。

IIOP セッションの作成

IIOP セッションを作成する場合は、以下に示す手順を実行します。

- *TDI_install_dir*/jars フォルダーおよびそのサブフォルダーに notes.jar ファイルがないことを確認してください。
- *TDI_install_dir*/jars/3rdparty/IBM または global.properties (または solution.properties) の com.ibm.di.loader.userjars プロパティーで指定されているフォルダーに ncso.jar をコピーします。
- ネイティブ・ライブラリーの制限により、ibmditk (または ibmditk.bat) および ibmdisrv (または ibmdisrv.bat) シェル・スクリプト内の PATH 環境変数にローカル Lotus Domino バイナリー (C:¥Lotus¥Domino など) を追加する必要があります。
- 以下の例外が発生することがあります。

  ```
  CTGDJE010E Connector was unable to initialize local Notes session to Domino Server.
  Exception encountered: java.lang.Exception: Native call SECKFMSwitchToIDFile failed with error:
       code 259, 'File does not exist'
  ```

Lotus Domino Server への認証に使用される Notes ID ファイルを IBM Security Directory Integrator のソリューション・ディレクトリーにコピーする必要があります。通常、このファイルは、*Notes_install_dir*/Data/ フォルダーにあります。この問題の原因は、notes.ini ファイルが ID ファイルを指す方法にあります。絶対パスではなく相対パスが使用されているため (KeyFilename=user.id)、各コンポーネントは、Notes のインストール済み環境のデータ・フォルダーではなく、現行ディレクトリーである IBM Security Directory Integrator のソリューション・ディレクトリーを検索します。

ネイティブ API 呼び出しのスレッド化

ネイティブ API 呼び出しのスレッド化の詳細については、以下に示す情報を使用して把握することができます。

コネクターが含まれている AssemblyLine を IBM Security Directory Integrator Server が実行する場合、この AssemblyLine は単一のスレッドで実行されます。このスレッドは、AssemblyLine コネクターにアクセスする唯一の AssemblyLine スレッドです。Lotus Notes API の初期化、項目の選択、項目の反復、およびコネクターの終了が 1 つのワーカー・スレッドにより実行されます。

Notes API の要件の 1 つとして、**ローカル・セッション**の使用時には、Notes API 関数を呼び出す前に、Notes API 関数を実行する各スレッドが NotesThread オブジェクトを初期化する必要があります。構成エディター GUI スレッドでは NotesThread オブジェクトが初期化されないため、これが原因で IBM Notes 例外が発生します。

NotesThread オブジェクトを初期化する方法は複数あります。コネクターによる初期化方法は、ローカル・セッションの作成時に NotesThread.sinitThread() メソッドを呼び出す方法です。

このため、Lotus Notes および Lotus Domino のすべてのコネクターでは、Lotus Notes ランタイムの初期化とすべての Lotus Notes API 関数の呼び出しに、コネクター固有の内部スレッドが使用されます。内部スレッドは、コネクターの初期化時に作成および開始され、コネクターの終了時に停止します。コネクターはすべてのネイティブ IBM Notes API 呼び出しの実行をこの内部スレッドに委任します。内部スレッド自体は、他のスレッドから送信されるネイティブ の IBM Notes API 呼び出し要求を待機し、この要求を処理します。

一般にこのインプリメンテーションでは、コネクターで構成エディター GUI 機能とマルチスレッド・アクセスがサポートされます。Lotus Notes コネクターは、ローカル・セッションが作成されている場合に Lotus Notes ランタイムを初期化します。

ncso.jar ファイル

IBM Security Directory Integrator の Notes/Domino コンポーネントが IIOP セッションを使用するには、ncso.jar ファイルが存在している必要があります。

IBM Security Directory Integrator 6.1 以降、ncso.jar は IBM Security Directory Integrator 製品に付属しません。 IBM Security Directory Integrator Lotus/Domino コンポーネントが適切に機能するためには、このファイルを手動で準備する必要があります。

ただし、Lotus Domino Server には ncso.jar ファイルが付属しています。このファイル (通常は Windows プラットフォームの <Domino_root>¥Data¥domino¥java¥ncso.jar) は、Domino のインストール済み環境から取得して IBM Security Directory Integrator_root¥jars¥3rdparty¥IBM フォルダーに保管することができます。これにより、初期化時に IBM Security Directory Integrator サーバーがこのファイルをロードできるようになります。インストールされている IBM Security Directory Integrator には ncso.jar はないため、既存の IBM Security Directory Integrator 6.0 機能は次のように変更されます。

サーバー側

以下に示すセクションを通じて、サーバーの側面のうち 2 つについての洞察を取得できます。

「-v」コマンド行オプション:

「-v」コマンド行オプションの詳細については、以下に示す情報を使用して把握することができます。

IBM Security Directory Integrator サーバーの「-v」コマンド行オプションは、すべての IBM Security Directory Integrator コンポーネントのバージョンを表示します。ncso.jar ファイルは IBM Security Directory Integrator の一部としてインストールされないため、ncso.jar をインストールされている Domino Server または Notes から取得しない場合は、次のようなメッセージが表示されます (ncso.jar に依存しないコンポーネントのバージョンは正しく表示されます)。

```
ibmdi.DominoUsersConnector:
com.ibm.di.connector.dominoUsers.DominoUsersConnector:
2006-03-03: CTGDIS001E The version number of the Connector is undefined
```

サーバー API getServerInfo メソッド:

サーバー API getServerInfo メソッドの詳細については、以下に示す情報を使用して把握することができます。

サーバー API には、IBM Security Directory Integrator コンポーネントのバージョン情報を要求するためのメソッド (Session.getServerInfo) があります。ncso.jar を使用するコネクターに関するバージョン情報がサーバー API から要求され、この jar がインストールされている Domino Server または Notes から取得されていない場合は、エラーがスローされます。例えば、次のようなスクリプトを使用してローカル・サーバー API にアクセスするとします。

```
session.getServerInfo().getInstalledConnectors()
```

この場合、以下のエラーが表示されます。

```
18:16:12  CTGDKD258E Could not retrieve version info for class
'com.ibm.di.connector.DominoChangeDetectionConnector'.:
java.lang.NoClassDefFoundError: lotus.domino.NotesException
```

AssemblyLine の実行、IIOP セッション

インストールされている Domino Server または Notes から ncso.jar ファイルを取得していないと、(IIOP セッションを使用する) コネクターを使用する AssemblyLine の実行が失敗し、NoClassDefFoundError 例外がスローされます。

コンポーネントの可用性の報告

以下に示すセクションで、使用可能なコンポーネントとして報告されているものについてのリストを示します。

コンポーネントのバージョン表:

すべてのインストール済み IBM Security Directory Integrator コンポーネントの各バージョンを記載した表 (「**サーバー**」パネルで表示されるサーバーのコンテキスト・メニュー・オプション「**インストール済みコンポーネントの表示**」から使用可能) が表示されます。

インストールされている Domino/Notes から `notes.jar` と `ncso.jar` のどちらも取得されていない場合は、Notes/Domino コネクターのいずれのコンポーネント・バージョンもこのテーブルに表示されません。

コンポーネント・コンボ・ボックス:

コンポーネント・コンボ・ボックスの使用中には、以下に示す指示により対処できます。

IBM Domino/Notes のインストール済み環境から `notes.jar` と `ncso.jar` のどちらも取得されていない場合は、「新規オブジェクトを挿入します」ボックス (このボックスが使用可能な場合は、「**コンポーネントの挿入... (Insert Component...)**」を選択するとアクティブになります) に IBM Notes/Domino コネクターの既存の IBM Security Directory Integrator コネクター・モードが (サポート対象のものだけでなく) すべて表示されます。

「入力マップ」からのデータ・ソースへの接続:

作成されるセッションに関係なく、インストールされている IBM Notes/Domino から jar ライブラリーを取得していない場合は、Notes/Domino コネクターの「入力マップ」タブからデータ・ソースへの接続を試行すると、コネクターをロードできないというエラーが表示されます。

「classes」フォルダー

「classes」フォルダーに対する処理には、以下に示す情報とパスを使用します。

TDI_Install_Folder`/classes` に配置されているこのフォルダーには、システム・クラス・ローダーによってロードされたユーザー提供のクラス・ファイルが含まれています。このフォルダーを使用して、システム・クラス・ローダーでロードする必要があるカスタム・クラスを指定できます。

クラス・ファイルをシステム・クラス・ローダーでロードするには、このフォルダーにクラス・ファイルをコピーする必要があります。クラスが Java パッケージ内にある場合は、クラス・ファイルを対応する「classes」フォルダーに含める必要があります。例えば、クラス・ファイルが Java パッケージ `com.ibm.di.classes` に含まれている場合は、このクラス・ファイルを *TDI_Install_Folder*`/classes/com/ibm/di/classes` フォルダー内に含める必要があります。

「classes」フォルダー内のクラス・ファイルのみがロードされます。つまり、このフォルダーに配置されていても、jar ファイルはロードされません。クラスが jar ファイルにパッケージ化されている場合は、このクラスを jar ファイルから「classes」フォルダーに抽出する必要があります。

Domino 変更検出コネクター

Domino 変更検出コネクターは、Domino サーバー上で維持されているデータベースに対して変更が発生したことを IBM Security Directory Integrator が検出できるようにします。

IBM Domino 変更検出コネクターは、IBM Domino Server 上のデータベース (NSF ファイル) 内で発生した変更を取得します。その後、どの Domino 文書が変更されたかを報告するため、他のリポジトリーを Domino と同期することができます。

注:

1. Notes のアーキテクチャーのため、このコネクターにはネイティブ・ライブラリーが必要となります (両方のセッション・タイプ IIOP および LocalClient に対して)。このため、このコネクターは、Windows プラットフォームでのみサポートされます。ローカル・クライアント・ライブラリーおよび Domino サーバーのインストールのパスを IBM Security Directory Integrator サーバー開始スクリプト `ibmdisrv.bat` の PATH 変数の定義に追加する必要があります。
2. このコネクターで使用可能なセッション・タイプの概要については、『コネクター別のサポートされるセッション・タイプ』を参照してください。

このコネクターを実行すると、他のリポジトリーと Lotus Domino データベースとを同期するために必要なオブジェクトの変更が、コネクターがオフライン中に発生したものと、コネクターの実行中に発生したものの両方について報告されます。

Lotus Domino 変更検出コネクターはイテレーター・モードで稼働し、項目レベルのみの文書変更を報告します。

AssemblyLine が反復するたびに、Lotus Domino 変更検出コネクターは Domino データベース内で変更のあった文書オブジェクトを 1 つずつ配信します。コネクターは、変更された IBM Domino 文書オブジェクトをそのままの状態で (現在の項目すべてを含めて) 配信し、オブジェクトの変更のタイプ (文書の追加、変更、または削除) も報告します。コネクターは、文書内で変更された項目、または項目の変更タイプについては報告しません。コネクターは、文書の変更を検索した後、文書を解析し、文書の項目すべてを新規の項目オブジェクトに項目属性としてコピーします。そして、コネクターはこの項目オブジェクトを戻します。

このコネクターでは、項目レベルでのみデルタ・タグがサポートされています。

このコネクターは IBM パスワード同期プラグインとともに使用できます。IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。

コネクターは、同期の状態を IBM Security Directory Integrator マシン上にローカルに格納します。開始時、コネクターは、最後の同期点から処理を続行し、その点以後の変更をすべて、コネクターがオフライン中に発生した変更も含めて報告します。

注: 構成画面で「ソートして配信」チェック・ボックスをチェックしていない場合、変更された文書が引き渡される順序は、発生順でも、その他の特定の順序でもありません。詳細については、 70 ページの『ソート』を参照してください。このオプションを使用しない場合、後で変更された文書がそれより前に変更された文書より先に引き渡される場合や、その逆の場合が発生する場合があります。

報告する変更がなくなった時点で、コネクターはデータの終わりを通知し、処理を停止します。ただし、すべての変更の報告を完了した後も実行を終了せずに活動状態のままにして、IBM Domino に対して変更のポーリングを繰り返すように構成することも可能です。

コネクターの使用

IBM Domino 変更検出コネクターは、以下に示す情報を通じて使用することができます。

文書の識別:

IBM Domino 変更検出コネクターは、IBM Domino 文書のユニバーサル ID (UnID) を取得します。UnID 値を使用して、コネクターから報告された文書の変更を追跡することができます。

例えば、文書の削除が報告された場合は、その文書の UnID 値を使用して、同期化しているリポジトリー内から削除するべきオブジェクトを検索します。IBM Domino ユーザー (ユーザー文書) を同期化している場合は、ユーザーの名前が変更された時点でそれを検出する必要があります。ユーザーが名前変更された (ユーザー文書のフルネーム項目が変更された) 時点で、コネクターはこれを「変更」操作として報告します。もう一方のリポジトリー内のオブジェクトを UnID で検索すると、元のオブジェクトが見つかります。そのオブジェクトから古いフルネーム属性を読み取り、新しいフルネーム値と比較します。これにより、ユーザーが名前変更されたかどうかを判別できます。

削除された文書:

Domino データベースから削除された文書は、「削除スタブ」オブジェクトを利用して追跡できます。これは、以下に示す情報を使用して実行することができます。

削除スタブは、削除された文書のユニバーサル ID と Note ID のみ提供し、それ以外は提供しません。これにより、コネクターが削除された文書を検出した場合は、文書項目が何も含まれていない項目が戻されることになります。ただし、コネクター自体によって以下の項目属性のみ追加されています。

- $$UNID
- $$NoteID
- $$ChangeType

同期の最小間隔:

各データベースには、「指定日数間に変更がない文書を削除する。指定日数」パラメーターがあります。この値より古い削除スタブは除去されます。削除された文書を処理する場合は、このパラメーターの値より短い間隔で同期化を行う (このコネクターを実行する) 必要があります。

Domino R8.0 および Domino R8.5 ではいずれも、Domino Administrator からこのパラメーターにアクセスできます。データベースをオープンした後、メニューから「**ファイル**」->「**複製**」->「**このアプリケーションのオプション...**」を選択し、「**スペースセーバー**」を選択します。パラメーターの名前は「**指定日数間に変更がない文書を削除する。指定日数**」です。

このパラメーターのデフォルト値は 90 日です。

データベースのレプリカへの切り替え:

UnID は、同一データベースのどのレプリカにおいても同じです。そのため、オリジナルのデータベースが破損した場合や使用できなくなった場合には、IBM Domino データベースの別のレプリカに切り替えることができます。

ただし、文書のタイム・スタンプはレプリカごとに異なります。このため、レプリカへの切り替えを行う際には、完全同期を実行する必要があります (「イテレーター状態キー」には新しいキーを使用し、「開始位置」パラメーターには「データの始まり」を設定します)。この結果、文書の追加や削除 (他のリポジトリーに既に適用済みのもの) が大量に報告される可能性がありますが、更新の欠落を確実に回避できます。

コネクターから戻される項目の構造:

コネクターから戻される項目の構造については、以下に示す情報を使用して把握することができます。

文書に含まれるすべての項目が、その元の名前を付けた項目属性にマップされます。

すべてのデータ値は java.util.Date オブジェクトとして戻されます。

また、以下の項目属性がコネクター自体によって追加されます (これらの値は文書項目としては利用不可)。

- $$UNID – 文書のユニバーサル ID (『$$UNID および $$NoteID 属性』を参照)
- $$NoteID – 文書の Note ID (『$$UNID および $$NoteID 属性』を参照)
- $$ChangeType – 文書変更のタイプ (『$$ChangeType 属性』を参照)
- $$DateCreated – この文書が作成された時刻を表す java.util.Date オブジェクト (この属性が利用可能なのは削除されていない文書のみ)
- $$DateModified - この文書が最後に変更された時刻を表す java.util.Date オブジェクト (この属性が利用可能なのは削除されていない文書のみ)

$$UNID および $$NoteID 属性:

ユニバーサル ID (UnID) は、IBM Domino 文書を一意的に識別する値です。コネクターは、NoteID 文書値も戻します。この詳細については、以下に示す情報を通じて把握することができます。

特定の文書のすべてのレプリカは同じ UnID を持っており、文書が変更されても UnID は変わりません。同期処理の際にオブジェクトを追跡するためには、この値を使用してください。ユニバーサル ID の値は、コネクターから引き渡される項目オ

ブジェクトの $$UNID 属性にマップされます。$$UNID 属性の値は 32 文字のストリングで、各文字は 16 進数字 (0 から 9、A から F) を表します。

NoteID 文書値は、現行データベースのコンテキスト内でのみ固有です (この文書のレプリカはでは、通常、NoteID が異なります)。コネクターは、$$NoteID 項目属性によって NoteID を引き渡します。この属性の値は、最大 8 文字までの 16 進文字で構成されるストリングです。

$$ChangeType 属性:

$$ChangeType という名前の属性が、Lotus Domino 変更検出コネクターから戻される項目すべてに追加されます。この属性の値については、以下に示す情報を通じて把握することができます。

$$ChangeType 属性には、次のいずれかの値が設定されます。

- **add** – 報告された文書が Lotus Domino データベースに新しく追加された文書であるという意味。
- **modify** – 報告された文書が、既存の文書を変更したものであるという意味。
- **delete** - 報告された文書が Domino データベースから削除されたという意味。

同期状態の値:

システム・ストアに保管される一部の値は、現在の同期状態を表しています。格納される値の詳細については、以下のセクションで把握することができます。

コネクターは、始動時にこれらの値を読み取り、適切な箇所から変更の報告を続行します。

コネクターがどのモードで実行されていても、2 つの同期状態値がユーザー・プロパティー・ストアに格納されます。この 2 つの値は項目オブジェクトに以下の名前の属性として格納され、それぞれの意味は以下のとおりです。

- ***SYNC_TIME*** – この属性は、コネクターの次回のポーリングでの「開始時刻」を表す java.util.Date オブジェクトです。つまり、コネクターの次回のポーリングでは、この時刻以後に発生したデータベースの変更のみが戻されます。開始条件が *データの始まり* と指定された特殊ケースでは、java.lang.String 値「NULL_DATE」が格納されます。
- ***SYNC_CHECK_DOCS*** – この属性は java.lang.Boolean オブジェクトで、コネクター固有のシステム・ストア・テーブル (下記を参照) 内に処理済みの文書があるかどうかをチェックする必要性の有無を示します。この属性は、コネクターの「状態キーの維持」パラメーターを「*読み取り後*」に設定している場合にのみ使用します。コネクターの「状態キーの維持」パラメーターを「*サイクルの終わり*」に設定している場合、この属性の値は常に「**false**」になります。

実行されているコネクターは、ユーザー・プロパティー・ストアに値を格納するだけでなく、システム・ストア内にもコネクターに固有のテーブルを作成します (まだ作成されていない場合)。このテーブルの名前は、「***domch_***」に「***イテレーター状態キー***」コネクター・パラメーターの値を連結したものになります。このコネクター固有のテーブルには、以下の特性を持つ値が格納されます。

- キーは、java.lang.String オブジェクトとして既に配信済みの変更された文書の UnID です。
- 値は、次回のポーリングのための日時を表す java.util.Date オブジェクトで、通常はこの文書がコネクターから配信された時刻と同じですが、UnID が削除済み文書に対応している場合は java.lang.String 定数「NULL_DATE」が格納されます。

コネクター固有のテーブルは、コネクターが同期セッションを正常に完了するたびにクリアされます。

同じ IBM Security Directory Integrator サーバー上で実行されている Lotus Domino 変更検出コネクターの各インスタンスに対して、異なるコネクター固有のテーブルがシステム・ストア内に存在します。

コネクターの同期状態へのアクセス:

コネクターがオフライン中には、次回のコネクターの実行で使用する「開始時刻」の日時にアクセスできます。この日時はユーザー・プロパティー・ストアに格納されています。

以下に示すのは、次回の同期の日時値を取得する方法です。

```
var syncTime = system.getPersistentObject("dcd_sync");
var sinceDateTimeAttribute = syncTime.getAttribute("SYNC_TIME");
var sinceDateTime = sinceDateTimeAttribute.getValue(0);
if (sinceDateTime.getClass().getName().equals("java.util.Date")) {
main.logmsg("Start date: " + sinceDateTime);
}
else {
main.logmsg("Start date: Start Of Data");
}
```

「dcd_sync」は、「**イテレーター・ストア・キー (Iterator Store Key)**」コネクター・パラメーターによって指定された値です。

以下に示すのは、次回の同期の開始日時を設定する方法です。

```
var syncTime = system.newEntry();
syncTime.setAttribute("SYNC_TIME", new java.util.Date()); //current time
syncTime.setAttribute("SYNC_CHECK_DOCS", new java.lang.Boolean("false"));
system.setPersistentObject("dcd_sync", syncTime);
```

項目のフィルター処理:

このバージョンのコネクターでは、文書のフィルター処理は実施されません。作成、変更、または削除されたすべてのデータベース文書がコネクターから報告されます。フィルター処理が必要な場合は、コネクターのフック内に自分でスクリプトを記述してください。

ソート:

変更文書は、最終変更日時ごとにソートして配信できます。この場合、構成画面の「ソートして配信」チェック・ボックスを選択します。

注: ソートを使用すると、メモリー使用量や CPU 時間に関してパフォーマンスが低下します。このため、ソートが必要かどうかを十分検討する必要があります。

IBM Domino Server のシステム時刻を使用:

IBM Domino 変更検出コネクターは、IBM Domino データベース内での変更を検出する際に、最終変更のタイム・スタンプを使用します。ここに記載されている情報を通じてこれについて詳しく知ることができます。

コネクターの状態には、IBM Domino Server のシステム時刻で読み取ったタイム・スタンプ値が組み込まれます。したがって、コネクターの実行中に、またはコネクターの実行と次回の実行との間に Domino Server のシステム時刻を変更すると、コネクターが誤った処理をする場合があります (変更の報告が欠落または反復する、誤った変更タイプが報告される、など)。

大規模な Domino データベース (.nsf ファイル) の処理:

大規模な IBM Domino データベース (.nsf ファイル)の処理においては、以下に示す情報が使用できます。

コネクターには、大量の物理メモリーが必要な場合があります。例えば、1,000,000 個以上の文書を含む巨大なデータベースを処理する場合などです。特に、完全同期を実行する場合は、大量の物理メモリーが必要になります。これは、コネクターが同期セッションの実行中に取得した全文書の UnID をメモリー内に保持するためです。例えば、データベースに約 1,000,000 個の変更された文書が含まれている場合は、512 MB の物理メモリーで十分です (メモリーを消費する他のプロセスが実行されていない場合)。この量のメモリーが利用可能でない場合は、IBM Security Directory Integrator が使用できるメモリーを増やすことが可能です。

また、「ソートして配信」パラメーターに注意してください。このパラメーターを有効にすると、パフォーマンスに重大な影響を与えます。

API:

IBM Domino 変更検出コネクターでは、これに固有の 2 つのメソッドをサポートしています。以下にそのメソッドを示します。

```
    /**
     * This method saves the synchronization state of the Connector.
     * This method should be called by users (using script component) whenever
     * they want to save the synchronization state.
     *
     * @throws Exception if the synchronization fails.
     */
 public void saveStateKey() throws Exception;

 /**
  * Skip the current document. Use this method to skip problem documents when
  * the Connector will otherwise die with an exception.
  * <p>
  * For example use the following script in the "Default On Error" hook of
  * the Connector:
  *
  * <pre>
  * thisConnector.connector.skipCurrentDocument();
  * </pre>
  *
  * </p>
  *
  *
  * @throws Exception
```

```
 *             If the Notes thread is not running or the Notes thread
 *             encounters an error while processing the command.
 */
public void skipCurrentDocument()throws Exception;
```

IBM Security Directory Integrator に必要なセットアップ

IBM Security Directory Integrator で必要なセットアップについて把握するには、以下に示すリンクを使用します。

必要なライブラリーに関する問題と、発生する可能性のあるライブラリー間の競合については、『コネクター別のサポートされるセッション・タイプ』と以下のセクションを参照してください。

IBM Domino に必要なセットアップ

Domino Server で必要なタスクおよび特権は、以下に示す情報を通じて使用することができます。

Domino Server で必要なタスク:

このコネクターは、リストされた Domino Server タスクが Domino Server 上で開始済みであることを必要とします。

- HTTP Web サーバー
- DIIOP サーバー

これらの Domino Server タスクがサーバー上で開始されていない場合、コネクターの処理は失敗します。

必須特権:

IBM Domino 変更検出コネクターの必須特権については、以下に示す情報を通じて把握することができます。

Domino 変更検出コネクターは、Domino Server に対して 2 つのセッションを作成します。1 つはローカル・ユーザー ID ファイルを使用してローカル Notes クライアント・コードを経由するセッション、もう 1 つはインターネット・ユーザー・アカウントを使用するリモート IIOP セッションです (この 2 つのセッションを確立するために同一の Domino ユーザーを使用することもできますが、必須ではありません)。これらのセッションに使用するアカウントには、以下の特権が必要です。

ローカル・ユーザー ID のアカウント

コネクターは Lotus Notes Client のユーザー ID ファイルを使用して、Lotus Domino Server に接続します。このため、コネクターを Lotus Domino Server にアクセスできるよう適切にセットアップするには、Lotus Notes Client のユーザー ID ファイルが必要となります。ユーザー ID ファイルがローカルに配置されている IBM Domino ユーザーには、変更をポーリングする IBM Domino データベースのアクセス制御リスト (ACL) 内に少なくとも「読者」アクセス権が構成されている必要があります。

この構成を行うには、Domino Administrator の「ファイル」タブで、変更をポーリングするデータベースを右クリックし、「アクセス制御」->「管理...」を選択します。このユーザー ID ファイルに対応するユーザー名がリストに含まれていない場合は、「追加...」ボタンをクリックして、そのユーザ

ー名をリストに追加します。リスト内でそのユーザー名を選択し、そのユーザーに対して「読者」以上 (つまり、「読者」、「作成者」、「編集者」、「設計者」、または「管理者」) のアクセスが設定されていることを確認します。

IIOP セッション用のインターネット・アカウント

このコネクターは、IIOP セッションを作成するために、Lotus Domino インターネット・ユーザー名とパスワードを必要とします。そのインターネット・ユーザーには、変更をポーリングする IBM Domino データベースのアクセス制御リスト (ACL) 内に少なくとも「読者」アクセス権が構成されている必要があります。

この構成を行うには、Domino Administrator の「ファイル」タブで、変更をポーリングするデータベースを右クリックし、「アクセス制御」->「管理...」を選択します。このインターネット・ユーザーがリストに含まれていない場合は、「追加...」ボタンをクリックして、そのインターネット・ユーザーをリストに追加します。リスト内でインターネット・ユーザー名を選択し、そのユーザーに対して「読者」以上(つまり、「読者」、「作成者」、「編集者」、「設計者」、または「管理者」) のアクセスが設定されていることを確認します。

構成

IBM Domino 変更検出コネクターには、以下に示すパラメーターがあります。

セッション・タイプ

コネクターで IIOP セッションを作成するのか、それとも LocalClient 呼び出しを実行するのかを指定します。これはドロップダウン・リストです。デフォルト値は「IIOP」です。

「**セッション・タイプ**」が「**LocalClient**」の場合は、「**IOR ストリング**」、「**HTTP ポート**」、「**ユーザー名**」、および「**SSL の使用**」の各パラメーターが無視されます。

Domino Server の IP アドレス

変更をポーリングするデータベースが存在する Lotus Domino Server の IP アドレス (またはホスト名)。

IOR ストリング

IIOP セッションの作成に使用される IOR ストリング。このパラメーターは、Lotus Domino Server からこの値を要求する代わりにオプションで使用できます。

HTTP ポート

Domino Server の HTTP タスクが実行されているポート。デフォルト値は 80 です。

ユーザー名

IIOP セッション認証のユーザー名。このユーザー名は、ユーザーのユーザー文書の「ユーザー名」フィールドの最初の値と一致する必要があります。

パスワード

IIOP セッション認証のユーザー・パスワード。このパスワードは、ユーザーのユーザー文書の「インターネット・パスワード」フィールドと一致する必要があります。

SSL の使用

クライアント・サイドの証明書を使用して Lotus Domino Server との暗号化された通信を使用可能にします。このパラメーターは、IIOP セッションにのみ関連します。

データベース

変更をポーリングする IBM Domino データベースのファイル名 (「names.nsf」など)。

イテレーター状態キー

IBM Security Directory Integrator のユーザー・プロパティー・ストアに現在の同期状態 (つまり、最終変更) を格納するパラメーターの名前を指定します。この名前は、IBM Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。

「**削除**」ボタンは、このパラメーターの値に対応するすべての同期状態をクリアします。クリックすると、ユーザー・プロパティー・ストアだけでなく、システム・ストアにあるコネクターに固有のテーブルからも、キーと値のペアが削除されます。

開始位置

開始条件のタイプ。このパラメーターは、「**イテレーター状態キー**」によって指定された永続パラメーターがブランクか、またはユーザー・プロパティー・ストアに存在しない場合にのみ考慮されます。これは、次のいずれかです。

データの開始

データベースからすべての文書を取得して、完全同期を実行します。

データの終わり

将来の変更 (コネクターを開始した後に行われる変更) のみを取得します。

特定の日付

「**開始日**」パラメーターに指定された値 (日付) 以降に発生した変更を取得します。

デフォルト値は「**データの開始**」です。

注: このパラメーターは、「**イテレーター状態キー**」によって指定された永続パラメーターがユーザー・プロパティー・ストアに存在しない場合にのみ考慮されます。

開始日 コネクターは、ここに指定されている日時以降に変更された文書を取得します。このパラメーターは、「**開始位置**」パラメーターが「**特定の日付**」に設定され、以下の日時フォーマットが受け入れられる場合にのみ使用されます。

- **yyyy-MM-dd HH:mm:ss.SSS** — 例えば、2002-05-23 16:39:07.628 (年が 4 桁、月が 2 桁、日が 2 桁、24 時間制の時が 2 桁、分が 2 桁、秒が 2 桁、ミリ秒が 3 桁)。Lotus Notes の日時の実際の精度は 10 ミリ秒であるため、注意してください。
- **yyyy-MM-dd HH:mm:ss** — 例えば、2002-05-23 16:39:07
- **yyyy-MM-dd** — 例えば、2002-05-23

これは、「**イテレーター状態キー**」に指定された永続パラメーターがブランクか、またはシステム・ストアに存在せず、「**開始位置**」パラメーターが「**特定の日付**」に設定されている場合にのみ有効です。

状態キーの維持

コネクターの状態をシステム・ストアに保管する方式を制御します。デフォルトは「**サイクルの終わり**」です。以下のいずれかを選択できます。

読み取り後

AssemblyLine の残りの部分を続行する前、Lotus Domino 変更ログから項目を読み取るときにシステム・ストアを更新します。この操作モードは、旧バージョンの IBM Security Directory Integrator では「一度だけ保証された配信」と呼ばれていました。

サイクルの終わり

AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、システム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。この場合、AssemblyLine 内の適切な時点において、Domino 変更検出コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保管する必要があります。

タイムアウト

変更された次の文書をコネクターが待機する最大秒数を指定します。このパラメーターが 0 の場合は、コネクターは無期限に待機します。待機状態に入ってからタイムアウト秒数内に変更された次の文書オブジェクトを取得しなかった場合、コネクターは NULL 項目を戻します (戻す項目がなくなったことを示す)。

スリープ間隔

変更を取得するための連続したポーリング間で、コネクターがスリープする秒数を指定します。

ソートして配信

チェックすると、変更文書が最終変更日時ごとにソートして配信されます。チェックしない場合、変更文書はランダムな順序で配信されます。変更文書の数が多い場合は、ソートによって IBM Security Directory Integrator パフォーマンスの速度が低下します。

詳細ログ

追加のログ・メッセージを使用可能にする場合は、チェックします。

Domino 変更検出コネクターのトラブルシューティング

IBM Domino 変更検出コネクターは、以下に示す情報を通じてトラブルシューティングすることができます。

1. **問題:** IBM Domino 変更検出コネクターを使用する AssemblyLine を実行すると、次のような例外が発生して AssemblyLine の処理が失敗する。*NotesException: Could not get IOR from Domino Server:* 。ここで、*<domino_server_ip>* はアクセスしようとしている IBM Domino Server の IP アドレス (すなわち、「**Domino Server の IP アドレス**」コネクター・パラメーターの値) です。

   解決策: この例外は、Domino Server 上で HTTP Web サーバー・タスクが実行されていないことを示します。アクセスしようとしている Domino Server 上で HTTP Web サーバー・タスクを始動後、AssemblyLine を再度開始してください。.

2. **問題:** IBM Domino 変更検出コネクターを使用する AssemblyLine を実行すると、パスワード・プロンプトでユーザー ID のパスワードを入力した直後に、次のような例外が発生して AssemblyLine の処理が失敗する。*NotesException: Could not open Notes session: org.omg.CORBA.COMM_FAILURE: java.net.ConnectException: Connection refused: connect Host: <domino_server_ip> Port: XXXXX vmcid: 0x0 minor code: 1 completed: No*。ここで、<domino_server_ip> はアクセスしようとしている IBM Domino Server の IP アドレス (すなわち、「**Domino Server の IP アドレス**」コネクター・パラメーターの値) です。

   解決策: この例外は、Domino Server 上で DIIOP サーバー・タスクが実行されていないことを示します。アクセスしようとしている Domino Server 上で DIIOP サーバー・タスクを始動後、AssemblyLine を再度開始してください。

   このメッセージが表示される別の理由は、適切に設定されていない Lotus Domino Server の完全修飾ホスト名です (localhost または 127.0.0.1 のままになっているなど)。この問題を解決するには、「IBM Domino Administrator」を開始して、使用しているサーバーを開いて「**構成**」タブに移動し、「サーバー」-> 「現在のサーバー文書」を編集します。この文書の「**基本**」タブで、「完全なインターネットホスト名」の正しい値を追加し、文書を保存してサーバーを再起動します。

3. **問題:** Domino 変更検出コネクターが変更を取得しているときに、次のような例外が発生する。*Exception in thread "main" java.lang.OutOfMemoryError*

   解決策: この例外は、IBM Security Directory Integrator の Java 仮想マシンが使用可能なメモリー (JVM の最大ヒープ・サイズ) が不足していることを示します。一般に、Java 仮想マシンは、使用可能メモリーのすべてを使用するわけではありません。IBM Security Directory Integrator の JVM が使用できるメモリーを増やすには、以下の手順を実行します。

   IBM Security Directory Integrator インストール・ディレクトリー内の ibmdisrv.bat ファイルを編集して、ヒープ・メモリー・サイズ・パラメーター (-Xms および -Xmx) を変更します。詳しくは、「トラブルシューティング」を参照してください。

   注: -Xms は初期ヒープ・サイズ (バイト数)、-Xmx は最大ヒープ・サイズ (バイト数) です。これらの値は、必要に応じたサイズに設定できます。

4. **問題:** 実際には削除されていないにもかかわらず、すべてのデータベース文書が ***削除された*** とコネクターが報告する。

   解決策: ローカル・ユーザー ID ファイルのユーザーに、変更をポーリングするデータベースに対して必要な特権が与えられていません。 72 ページの『必須特権』の説明に従って、必要なユーザー権限を与えてください。

5. **問題:** Domino 変更検出コネクターを使用する AssemblyLine を実行すると、次のような例外が発生する。*java.lang.UnsatisfiedLinkError: <IBM Security Directory Integrator_install_folder>¥libs¥domchdet.dll: Can't find dependent libraries* ここで、*<IBM Security Directory Integrator_install_folder>* は IBM Security Directory Integrator がインストールされているフォルダーです。

   注: IBM Security Directory Integrator サーバーをコマンド・プロンプトから実行した場合は、この例外メッセージが出力される前に、「This application has failed to start because nNOTES.dll was not found. Re-installing the application may fix this problem.」というメッセージがポップアップ・ダイアログ・ボックスに表示されます。

   解決策: この例外メッセージやポップアップ・ダイアログ・ボックスが表示されるのは、コネクターが Notes のダイナミック・リンク・ライブラリーを検出できないためです。可能性の高い原因は、ibmditk.bat または ibmdisrv.bat に誤った Lotus Notes ディレクトリーが指定されているか、まったく指定されていないことです。したがって、ibmditk.bat と ibmdisrv.bat の両方について、PATH 環境変数に指定された Lotus Notes ディレクトリーが正しいことを確認する必要があります。詳しくは、 72 ページの『IBM Security Directory Integrator に必要なセットアップ』を参照してください。

6. **問題:** 一部の文書に無効なデータが含まれているため、コネクターが例外をスローして停止する。このような文書は、データベース全体のごくわずかな部分で構成されています。問題のある文書をスキップして、繰り返しを継続します。

   解決策: コネクターの「エラー発生時のデフォルト」フックをオーバーライドします。`skipCurrentDocument()` メソッドを使用して、コネクターが問題のある文書をスキップするよう、コネクターの内部原票カウンターを増分します。また、`system.skipEntry()` メソッドを使用して、現行サイクルをスキップするよう AssemblyLine に指示します。コネクターが文書の読み取りに失敗したため、このサイクルには意味のあるデータが提供されません。

   「エラー発生時のデフォルト」フックに入力するスクリプトは、非致命的エラー (文書に無効フィールドが含まれているなど) と致命的エラー (Domino Server が実行されていないなど) で区別する必要があります。致命的エラーが発生した後にコネクターが繰り返しを継続しないようにする必要があります。以下に、「エラー発生時のデフォルト」フックのスクリプト例を示します。このスクリプトでは、無効な日付を含む文書のみがスキップされます。

```
var ex = error.getObject("exception");
var goOn = false;
if (ex != null) {
 if (ex.getMessage().indexOf("Invalid date") != -1) {
  goOn = true;
 }
}
if (goOn) {
```

```
 thisConnector.connector.skipCurrentDocument();
 system.skipEntry();
} else {
 throw "Fatal error: "+error;
}
```

7. **問題:** Lotus Domino 変更検出コネクターを使用する AssemblyLine を実行すると、以下の例外で AssemblyLine が失敗する。

   ```
   java.lang.Exception: CTGDJE010E Connector was unable to initialize local Notes session to
   Domino Server. Exception encountered: java.lang.Exception: Native call SECKFMSwitchToIDFile
   failed with error: code 259, 'File does not exist'.
   ```

   解決策: この問題の詳細は、61 ページの『インストール後の構成』セクション内の『ローカル・クライアント・セッションを作成する場合』と『IIOP セッションを作成する場合』のそれぞれのパラグラフで確認できます。

互換性:

IBM Domino 変更検出コネクターの互換性について把握するには、以下に示すリンクを使用します。

必要なライブラリーを使用してこのコネクターをセットアップする方法と、その他の Lotus Domino/Notes® コネクターとの相互作用については、『コネクター別のサポートされるセッション・タイプ』のセクションを参照してください。

関連情報

Java セッション・オブジェクトへのアクセス、
Java クラスを使用する文書へのアクセス。

Lotus Domino ユーザー・コネクター

Lotus Domino ユーザー・コネクターを使用すると、Lotus Domino ユーザー・アカウントにアクセスし、それらのユーザー・アカウントを管理することができます。これにより、リストされたアクションを実行できます。

- Domino ディレクトリからのユーザー文書とその項目の取得
- Lotus Domino ユーザーの作成と登録
- 管理要求データベースへの管理要求の通知による (IBM Domino 管理プロセスを介した) IBM Domino ユーザー削除操作の開始
- Domino ディレクトリのユーザー文書の変更によるユーザーの変更
- 「アクセス拒否グループ」へのユーザーの追加/削除によるユーザーの「無効化/有効化」
- Lotus Domino ユーザーの「ルックアップ」の実行

現時点では、このコネクターではユーザー再認証プロセスはサポートされていません。

リモート・サーバーとローカル・マシンのどちらの IBM Domino Server にもアクセスできます。

このコネクターは、イテレーター、ルックアップ、AddOnly、更新、および削除の各モードで動作し、以下の操作を実行できます。

イテレーター

アドレス帳から取り出したすべてのユーザー文書 (またはフィルターにより選別したサブセット) に対して、繰り返しを行います。

このコネクターは、「ドミノディレクトリー」データベースの個人文書を対象として繰り返しを行います。すべてのユーザー文書 (フィルターが設定されている場合はフィルター基準を満たすもの) が項目オブジェクトとして引き渡され、すべての文書項目 (添付文書を除く) が項目属性に変換されます。

コネクターが戻す項目には、ユーザー文書項目に対応する属性の他に、コネクター自体が作成した追加属性が含まれています。以下の表でこれらの属性を説明します。コネクターから派生した属性であり、Domino の「ネイティブ」の属性でないことを示すため、属性名の先頭に「DER_」が付いています。

表 11. 派生属性

属性名	タイプ	値
DER_IsEnabled	ブール値	*true* – ユーザーが「アクセス不可グループのみ」グループに属していない場合。 *false* – ユーザーが少なくとも 1 つの「アクセス不可グループのみ」タイプのグループに属している場合。

ルックアップ

一定の基準を満たすユーザー文書を探して検索します。

ルックアップ・モードでは、コネクターが実行する検索のタイプは「**フルテキスト検索**」パラメーターの値によって決まります。

- 「**フルテキスト検索**」が「**true**」: コネクターは、「ユーザー」ビュー内でフルテキスト検索を行います。

  注: 「フルテキスト検索」は、フルテキスト索引付きデータベースとフルテキスト索引なしデータベースのどちらにも有効です。ただし、フルテキスト索引なしのデータベースの場合は、検索効率は低下します。

  また、データベースのフルテキスト索引が更新されないことがあり、その場合は、検索結果は実際のデータベースの内容と一致しないことがあります。
- 「**フルテキスト検索**」が「**false**」: コネクターは Lotus の式を使用して通常のデータベース検索を実行します。コネクターは、この公式にエレメント (Form = "Person") を自動的に追加するため、検索対象はユーザー文書のみに限定されます。

AddOnly

Lotus Domino Server に新規ユーザーを登録し、それらのユーザー文書を作成します。このように操作することで、ユーザー登録時にメール・テンプレートを指定できます。テンプレートが指定されないと、コネクターは引き続き IBM Security Directory Integrator 6.0 バージョンのコネクターとして機能します (デフォルト・テンプレートを使用します)。

更新 ユーザーのユーザー文書の変更、ユーザーの有効化/無効化、既存 (インター

ネット) ユーザーの登録、および「アクセス拒否グループ」でのユーザー名の追加/削除による有効化/無効化を実行します。

削除 ユーザー削除の要求を Domino Server 管理要求データベースにポストします。

このコネクターは IBM パスワード同期プラグインとともに使用できます。IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。

注: Domino ユーザー・コネクターを使用するには、Notes のリリースが 7.0、8.0、または 8.5.x、および Domino Server バージョン 7.0、8.0、または 8.5.x でなければなりません。

Domino Server への展開と接続

Domino Server コネクターへの展開と接続には、以下に示す情報とリンクを使用します。

必要なライブラリーのセットアップと発生する可能性のあるライブラリー間の競合についての詳細は、『コネクター別のサポートされるセッション・タイプ』を参照してください。

Domino Server マシンへのデプロイ

Domino Server がインストールされているマシンに Domino ユーザー・コネクターがデプロイされている場合は、サポートされている認証メカニズムであるインターネット・パスワード認証と Notes ID ファイル認証の両方を使用できます。

Notes クライアント・マシンへのデプロイ

IBM Notes Client がインストールされているマシンに IBM Domino ユーザー・コネクターがデプロイされている場合は、IBM Notes ID ファイル認証のみを使用できます。

ローカル・サーバー接続を認証するには、IBM Domino では、ユーザーのショート・ネームおよびインターネット・パスワードが必要です (これらはコネクターのパラメーターです)。

構成

コネクターに必要なパラメーターのリストを示します。

セッション・タイプ

コネクターで IIOP セッションを作成するのか、それとも LocalClient 呼び出しを実行するのかを指定します。これはドロップダウン・リストです。デフォルト値は「IIOP」です。

「**セッション・タイプ**」が「**LocalClient**」の場合は、「**IOR ストリング**」、「**HTTP ポート**」、「**ユーザー名**」、および「**SSL の使用**」の各パラメーターが無視されます。詳しくは、82 ページの『旧バージョンからのパラメーター・マイグレーション』を参照してください。

Domino Server の IP アドレス

「Lotus Domino ディレクトリ」データベースをホストする Lotus Domino Server の IP アドレス (またはホスト名)。

このパラメーターが指定されていないかまたは空の場合は、ローカル・マシンが使用されます。この動作により、6.1 以前の IBM Security Directory Integrator 構成ファイルとの互換性が確保されます。

IOR ストリング
IIOP セッションの作成に使用される IOR ストリング。このパラメーターは、Lotus Domino Server からこの値を要求する代わりにオプションで使用できます。

HTTP ポート
Domino Server の HTTP タスクが実行されているポート。デフォルト値は 80 です。

ユーザー名
Domino Server へのログインまたは認証に使用するユーザー名。「**セッション・タイプ**」に「**LocalClient**」の認証が使用されている場合は無視されます。詳しくは、 83 ページの『認証』を参照してください。

パスワード
ユーザー名のパスワード、または Notes ID ファイル認証を使用する場合はこのファイルに関連付けられているパスワード。詳しくは、 83 ページの『認証』を参照してください。

SSL の使用
クライアント・サイドの証明書を使用して Lotus Domino Server との暗号化された通信を使用可能にします。このパラメーターは、IIOP セッションにのみ関連します。

ドミノディレクトリー・データベース
Lotus Domino ディレクトリ・データベース (従来の「アドレス帳」データベース) の名前。通常、「names.nsf」 (デフォルト) になります。

フルテキスト検索
このパラメーターは、コネクターがイテレーター・モードまたはルックアップ・モードで構成されている場合に使用します。チェックすると、コネクターは**ユーザー**・ビューとフルテキスト検索を使用してユーザー文書にアクセスします。チェックしないと、コネクターは通常のデータベース検索を使用します。この場合、コネクターは**フォーム項目**の値が **Person** である文書のみにアクセスすることにより、自動的にデータベース検索をユーザー文書のみに絞ります。このパラメーターの影響を受けるのは、イテレーター・モードとルックアップ・モードのみです。

フルテキスト・フィルター
この値が考慮されるのは、**フルテキスト検索**が使用可能である場合のみです。このパラメーターには、イテレーター・モードのコネクターが戻すユーザー文書をフィルターに掛ける、フルテキスト照会が含まれます。NULL または空のストリングの場合は、フィルター操作は行われません。デフォルト値は空です。

公式フィルター
この値が考慮されるのは、**フルテキスト検索**が使用不可である場合のみです。このパラメーターには、イテレーター・モードのコネクターが戻すユーザーをフィルターに掛ける公式が含まれます。コネクターは、この公式に自動的に以下のスニペットを追加します。

```
"& Form = "Person""
```

これにより、検索がユーザー文書のみに制限されます。デフォルト値は空です。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

旧バージョンからのパラメーター・マイグレーション

旧バージョンからのパラメーター・マイグレーションについては、以下に示す情報を使用して把握することができます。

IBM Security Directory Integrator には、このコネクター用の「**セッション・タイプ**」パラメーターが、他の IBM Lotus Notes/Domino コネクターとの調整の一環として含まれています。以前は「認証メカニズム」パラメーターを使用して構成されていた機能は、「**セッション・タイプ**」パラメーターによってカバーされています。

- 構成ファイルに「**セッション・タイプ**」パラメーターがある場合は、このパラメーターの値が、「**認証メカニズム**」パラメーターの有無にかかわらず常に使用されます。
- ただし、「**セッション・タイプ**」パラメーターが使用されず、「**認証メカニズム**」パラメーターがある場合は、マッピングが行われ、以前のバージョンとの互換性が保証されます。
  - 「**Notes ID ファイル**」は「LocalClient」にマッピングされます。
  - 「**インターネット・パスワード**」は「LocalServer」にマッピングされます。

セキュリティー

IBM Domino ユーザー・コネクターのセキュリティーについて把握するには、以下に示す情報とパスを使用します。

IBM Security Directory Integrator が IBM Domino Server にアクセスできるようにするには、「**Domino Administrator**」->「**構成**」->「**現在のサーバー文書**」->「**セキュリティー**」->「**Java/COM の制限 (Java/COM Restrictions)**」からそれを有効にしなければならない場合があります。IBM Security Directory Integrator で使用するアカウントとして構成されたユーザー・アカウントは、「**制限付き Java/Javascript/COM の実行**」と「**制限なし Java/Javascript/COM の実行**」の下にリストされているグループに属している必要があります。

Domino Server とクライアントの間での暗号化の構成:

IBM Domino ユーザー・コネクターで使用可能なポート暗号化オプションについて把握します。

IBM Notes Client マシンで IBM Domino ユーザー・コネクターを実行している場合は、IBM Notes Client マシンと IBM Domino Server マシンの間で通信が行われます。

この通信を暗号化するには、Lotus Domino または Lotus Notes (あるいはこの両方) でポート暗号化を使用します。 2 つのオプションがあります。

IBM Domino Server 通信ポートの暗号化

サーバー設定のみを構成するためセットアップが容易ですが、Notes Client を使用する標準ユーザーを含むすべてのクライアントとの通信に影響します。

1. Lotus Domino Administrator で「**設定**」を選択します。
2. 右側のパネルから「**サーバー/サーバーポート... (Server/Server Ports...)**」を選択します。
3. 使用する通信ポートごとに、「**通信ポート**」リストからポートを選択して「**ネットワークデータの暗号化**」オプションをチェックします。
4. 「**OK**」をクリックします。
5. IBM Domino サーバーを再始動して、変更を有効にします。

Notes 通信ポートの暗号化

この暗号化は、暗号化を必要としない他の IBM Notes Client には影響しません。

1. Lotus Notes で、「**ファイル**」->「**設定**」->「**ユーザー設定...**」に移動します。
2. 左側のナビゲーション・パネルから「**ポート**」を選択します。
3. 使用する通信ポートごとに、「**通信ポート**」リストからポートを選択して「ネットワークデータの暗号化」オプションをチェックします。
4. 「**OK**」をクリックします。
5. Lotus Notes を再始動して変更を有効にします。

認証:

IBM Domino ユーザー・コネクターは、IBM Domino ディレクトリー (アドレス帳データベース) にアクセスするために IBM Domino ユーザーを偽装します。認証については、以下に示す情報を通じて把握することができます。

Domino ユーザー・コネクターでは、インターネット・パスワード認証と Notes ID ファイル・ベースの認証という 2 種類の認証メカニズムがサポートされています。

インターネット・パスワード認証 – IIOP およびローカル・サーバー・セッションでの使用

この認証メカニズムでは、Lotus Domino ユーザーの短縮名とインターネット・パスワードが使用されます。コネクター構成パラメーター「**ユーザー名**」と「**パスワード**」に、Lotus Domino ユーザーの短縮名とインターネット・パスワードを指定する必要があります。

IBM Domino ユーザー・コネクターでは、IBM Domino ディレクトリーに基づいて、ローカル呼び出しを実行するためにインターネット・セッション・オブジェクトを作成する目的で、この認証メカニズムが使用されます。この認証メカニズムを使用するには、Lotus Domino Server がローカル・マシンにインストールされている必要があります。

Notes ID ファイル認証 – ローカル・クライアント・セッションでの使用

この認証メカニズムでは、現在構成されているデフォルトの IBM Notes ID ファイルとそのパスワードが使用されます。ローカル・クライアント・セッ

ションは、パスワード・パラメーターを使用して作成されます。このパラメーターの値が Notes ユーザー ID と一致する場合にアクセスが付与されます。

現在構成されているデフォルトの IBM Notes ID ファイルは、IBM Notes Client 構成の一部です。一般に Lotus Notes Client では、現在構成されているデフォルトの Lotus Notes ID ファイルのパスが `notes.ini` ファイルに保管され、次回の Lotus Notes Client 始動時にこの Lotus Notes ID ファイルがデフォルトで使用されます。

コネクター構成パラメーター「**パスワード**」に Lotus Notes ID ファイルのパスワードを指定する必要があります。

IBM Domino ユーザー・コネクターでは、IBM Notes ユーザー ID に基づいて、ローカル呼び出しを実行するためにセッション・オブジェクトを作成する目的で、この認証メカニズムが使用されます。この場合、IBM Domino Server または IBM Notes Client がローカルにインストールされている必要があります。

この認証メカニズムは、IBM Notes Client マシンと IBM Domino Server マシンの両方で使用できます。Domino Server マシンでこのメカニズムが使用される場合は、サーバー ID ファイルが使用されます。一般に、サーバー ID ファイルにはパスワードまたは空のパスワードは記述されていません。このため、通常はコネクター構成パラメーター「**パスワード**」を空白のままにします。ただし、サーバー ID ファイルにパスワードが記述されている場合は、コネクター構成パラメーター「**パスワード**」の値としてそのパスワードを指定してください。

許可:

Lotus Domino Server は、Lotus Domino ディレクトリー (アドレス帳データベース) のアクセス・コントロール・リストを使用して、コネクターが使用する Lotus Domino ユーザーに必要なデータベース、文書、またはフィールドへのアクセス権限が実際に設定されているかどうかを検証します。許可について把握するには、以下に示す情報が使用できます。

コネクターを使用して Lotus Domino ユーザーの姓名のいずれかまたは両方を変更した場合は、名前を変更する前にユーザーがアクセス権限を持っていたデータベースのアクセス・コントロール・リスト (ACL) を手動で更新する必要があります。これにより、新しいユーザー名が認識されます。

Domino ユーザー・コネクターの使用

IBM Domino ユーザー・コネクターは、以下に示す動作モードをサポートしています。

イテレーター・モード:

このコネクターは、**アドレス帳**データベースのユーザー文書を対象として繰り返しを行います。ここに記載されている情報を通じてイテレーター・モードについて詳しく知ることができます。

すべてのユーザー文書 (フィルターが設定されている場合はフィルター基準を満たすもの) が項目オブジェクトとして引き渡され、すべての文書項目 (添付文書を除く) が項目属性に変換されます。

コネクターが戻す項目には、ユーザー文書項目に対応する属性の他に、コネクターが計算した値を持つ追加 (派生) 属性が含まれています。以下に、このような派生属性のリストを示します。

DER_IsEnabled
(ブール) ユーザーが使用可能であるか使用不可であるかを指定します。

- **true**: ユーザーが **Deny List only** グループに属していない場合。
- **false**: ユーザーが少なくとも 1 つの **Deny List only** グループに属している場合。

ルックアップ・モード:

ルックアップ・モードでは、コネクターはユーザー文書の検索を行い、検索のタイプは**フルテキスト検索**パラメーターの値によって決まります。

- **フルテキスト検索 = true**: コネクターはユーザー・ビュー内でフルテキスト検索を実行します。フルテキスト検索は、フルテキスト索引付きデータベースとフルテキスト索引なしデータベースのいずれにも有効です。ただし、フルテキスト索引なしのデータベースの場合は、検索効率は低下します。また、データベースのフルテキスト索引が更新されないことがあり、その場合は、検索結果は実際のデータベースの内容と一致しません。
- **フルテキスト検索 = false**: コネクターは Lotus 公式を使用して、通常のデータベース検索を実行します。コネクターは、この公式にエレメント (Form = "Person") を自動的に追加するため、検索対象はユーザー文書のみに限定されます。

単純リンク基準を使用する場合は、正規名 (CN=UserName/O=Org) および短縮名 (UserName/Org) の両方の値を使用して、ユーザーのフルネームを指定することができます。コネクターは、指定された値を自動的に処理し、必要があれば変換します。

拡張リンク基準を使用する場合は、十分に注意して、次に示す正しい形式でユーザーのフルネームを指定する必要があります。

- フルテキスト検索の場合: 短縮名 (UserName/Org) を使用します。
- 通常データベース検索の場合: 正規名 (CN=UserName/O=Org) を使用します。

AddOnly モード:

AddOnly モードでは、常に新しいユーザー文書が**アドレス帳**データベースに追加されます。この詳細を把握するには、次に示す情報と表を参照してください。

追加プロセスでは、属性マッピングにより提供されるどのような属性でも受け入れられますが、Lotus Domino が正しいユーザー処理を行うためには、属性名は、Lotus Domino が処理する**項目**名と一致している必要があります。このコネクターはユーザーのみを対象として操作を行うため、常に属性の **Type** および **Form** の値を **Person** に設定します。したがって、属性マッピング・プロセスでこれらの属性に設定されている値はすべて指定変更されます。ユーザー文書を正しく作成するには、

LastName の IBM Domino ユーザー属性が必要です。既存の値がコネクターによって自動的にハッシュされるのでなければ、HTTPPassword 属性は必須ではありません。

属性の固定スキーマに応じて、コネクターは新規ユーザーを登録できます。以下の表は、これらの属性、*conn* 項目内の各属性の有無に応じたコネクターの動作、および各属性の値を示しています。

属性名	タイプ	登録のために必要か?	値
REG_Perform	ブール値	はい	true に設定すると、コネクターはユーザー登録を行います。 この属性が存在しないか、値が false である場合は、他の REG_ 属性の存在の有無やその値にかかわらず、コネクターはユーザー登録を行いません。
REG_IdFile	ストリング	はい	登録する ID ファイルの絶対パスが含まれています。以下に例を示します。 `c:¥¥newuserdata ¥¥newuser.id`
REG_UserPw	ストリング	いいえ	ユーザーのパスワード。
REG_Server	ストリング	いいえ	ユーザーのメール・ファイルを含むサーバーの名前。 この属性が存在しない場合は、現行のコネクターの Lotus Domino セッションから値が取得されます。 コネクターが Lotus Notes Client マシンで実行され、ユーザーを登録している場合は、この属性を指定して、サーバーに新規登録ユーザー用のメール・ファイルを作成する必要があります。
REG_CertifierIDFile	ストリング	はい	認証者 ID ファイルへの絶対ファイル・パス。
REG_CertPassword	ストリング	はい	認証者 ID ファイルのパスワード。 **注:** ユーザーを登録しているときに認証者パスワードに誤りがあると、ポップアップ・ウィンドウが表示されます。認証者パスワードは必ず正しく指定してください。
REG_Forward	ストリング	いいえ	ユーザーのメール・ファイル用の転送ドメイン。
REG_AltOrgUnit	<String> のベクトル	いいえ	ID ファイルを作成するときに使用する組織単位の代替名。
REG_AltOrgUnit Lang	<String> のベクトル	いいえ	ID ファイルを作成するときに使用する組織単位の代替言語名。

属性名	タイプ	登録のために必要か?	値
REG_CreateMailDb	ブール/ストリング	いいえ	true – メール・データベースを作成します。 false – メール・データベースを作成しません (セットアップ時に作成されます)。 この属性が存在しない場合は、デフォルト値として false が使用されます。この属性が true の場合は、MailFile 属性を有効なパスにマップする必要があります。
REG_MailTemplateFile	ストリング	いいえ	コネクターがメール・ファイルの作成に使用する Lotus Notes テンプレート・データベースのファイル名。この属性がない場合は、デフォルトのメール・テンプレートが使用されます。
REG_MailTemplateServer	ストリング	いいえ	メール・テンプレート・データベース (「REG_MailTemplateFile」で指定された) が常駐する Lotus Domino Server マシンの IP アドレスまたはホスト名。この属性がない場合は、ローカルの Lotus Domino Server マシンが使用されます。
REG_MailDbInherit	ブール/ストリング	いいえ	true – 作成されるユーザー・メール・データベースは、メール・テンプレート・データベース設計の変更を継承します。 false – 作成されるユーザー・メール・データベースは、メール・テンプレート・データベース設計の変更を **継承しません**。 この属性が存在しない場合は、デフォルト値として「false」が想定されます。
REG_StoreIDInAddress Book	ブール/ストリング	いいえ	true – ID ファイルをサーバーの IBM Domino ディレクトリーに保管します。 false – ID ファイルをサーバーの IBM Domino ディレクトリーに保管しません。 この属性が存在しない場合は、デフォルト値として「false」が想定されます。
REG_Expiration	日付	いいえ	ID ファイルを作成するときに使用する有効期限の日付。この属性が存在しないか、値が NULL である場合は、デフォルト値として「現在日 + 2 年」が使用されます。
REG_IDType	整数/ストリング	いいえ	作成する ID ファイルのタイプ。0: フラット ID を作成します。1: 階層 ID を作成します。2: 認証者 ID がフラットか階層かに合わせて ID を作成します。 この属性が存在しない場合は、デフォルト値として 2 が使用されます。

属性名	タイプ	登録のために必要か?	値
REG_Is NorthAmerican	ブール/ストリング	いいえ	true – ID ファイルは North American です。 false – ID ファイルは North American ではありません。 この属性が存在しない場合は、デフォルト値として true が使用されます。
REG_MinPassword Length	整数/ストリング	いいえ	REG_MinPasswordLength 値は、ユーザーが以後パスワードを変更する際に必要とされる最小パスワード長を定義します。ユーザーを登録するときに使用するパスワードは、REG_MinPasswordLength 値の制約を受けません。 この属性が存在しない場合は、デフォルト値として 0 が使用されます。
REG_OrgUnit	ストリング	いいえ	ID ファイルを作成するときに使用する組織単位。この属性が存在しない場合は、デフォルト値として " " が使用されます。
REG_RegistrationLog	ストリング	いいえ	ID ファイルを作成するときに使用するログ・ファイル。この属性が存在しない場合は、デフォルト値として " " が使用されます。
REG_Registration Server	ストリング	いいえ	ID ファイルを作成するときに使用するサーバー。この属性が使用されるのは、作成された ID がサーバーの Lotus Domino Directory に保管される場合か、新規ユーザー用のメール・データベースが作成される場合のみです。
REG_StoreID InAddressBook	ブール/ストリング	いいえ	true - ID ファイルをサーバーの IBM Domino ディレクトリーに保管します。false - ID ファイルをサーバーの IBM Domino ディレクトリー に保管しません。この属性が存在しない場合は、デフォルト値として false が使用されます。

ユーザー登録が成功するためには、「**登録のために必要**」フィールドが「**はい**」に設定されている属性が必要です。ユーザー登録の成功には、これらの REG_ 属性の他に、**LastName** の IBM Domino ユーザー属性も必要です。

REG_Perform が **true** に設定されているときに、登録に必要な他の属性のいずれかが欠落している場合は、コネクターは例外をスローし、問題を説明するメッセージを出します。

更新モード:

操作項目と属性項目については、以下に示す情報とリンクを通じて把握することができます。

更新モードでは、以下が行われます。

1. IBM Domino で、更新する項目が検索されます。

2. 該当項目が見つからない場合は、『AddOnly モード』の項で説明した AddOnly 操作が実行されます (必要な REG_ 属性が指定されている場合は、ユーザー登録も行われます)。
3. 該当項目が見つかった場合は、変更操作が行われます。

ユーザーを変更するときは、Lotus Domino ユーザー・コネクターは、**アドレス帳**データベース内のユーザー文書を、指定された属性で変更します。変更プロセスでは、属性マッピングにより提供される属性をすべて受け入れますが、Lotus Domino が正しいユーザー処理を行うためには、属性名は、Lotus Domino が処理する**項目**名と一致している必要があります。Lotus Domino ユーザー・プロパティーのリストについては、93 ページの『Domino ユーザー属性 (ユーザー文書項目) のリスト』を参照してください (ただしすべてのプロパティーがリストされているわけではありません)。

このコネクターはユーザーのみを対象として操作を行うため、属性マッピング・プロセスで何が行われるかにかかわらず、属性の **Type** および **Form** は変更しません (これらの値はいずれも **Person** です)。**HTTPPassword** 属性が指定されている場合は、既存の値がコネクターによって自動的にハッシュされます。

ユーザーを変更するプロセスでは、Domino ユーザー・コネクターは、ユーザーを使用不可および使用可能にするオプションを提供します。ユーザーを使用不可にするには、「**アクセス不可グループのみ**」グループにそのユーザーの名前を追加します。(「**アクセス不可グループのみ**」グループについては、IBM Domino の資料を参照してください。http://www.lotus.com/products/domdoc.nsf にアクセスして、**Domino Document Manager 3.5** リンクをクリックしてください。) ユーザーを使用可能にするには、すべての **Deny List only** グループからそのユーザーの名前を削除します。

コネクターは、**conn** 項目の有無、および以下の項目属性の値に応じて、ユーザーを使用不可または使用可能にします。

ACC_SetType
(整数/ストリング) この属性が存在しない場合は、アクションは何も行われず、ユーザーは現在の使用不可/使用可能の状況を維持します。この属性が指定されている場合は、その値が検査されます。

- **0**: コネクターは**ユーザーを使用不可にする**操作を行います (**ACC_DenyGroupName** 属性に指定されているグループにそのユーザーの名前が追加されます)。
- **1**: コネクターは**ユーザーを使用可能にする**操作を行います (その他の値の場合、例外が出されます)。

ACC_DenyGroupName
(ストリング) ユーザーを使用不可にするときに、そのユーザーの名前を追加する **Deny List only** グループの名前。**ACC_SetType** の値が **0** の場合は、**ACC_DenyGroupName** 属性が必須になります。この属性が欠落しているか、存在しない **Deny List only** グループを値に指定する場合、例外がスローされます。**ACC_SetType** 属性が欠落しているか、その値が **1** である場合は、**ACC_DenyGroupName** 属性は不要であり、その値は無視されます。

このコネクターは、変更時にユーザー登録も行うことができます。登録を行うかどうかの決定には、AddOnly モードの場合と同じ規則が適用されます。同じ属性スキーマが使用され、すべての REG_ 属性が同じ意味を持ちます。

REG_ 属性に基づき登録を行うことが決定された場合に、以下のような状況が生じることがあります。

- ユーザーはまだ登録されていない (例えば、インターネットまたは Web ユーザーを登録し、そのユーザーが IBM Notes Client を使用してログオンおよび作業ができるようにしようとしている場合)。ユーザーは登録され、新規 ID ファイルの作成などが行われます。
- ユーザーは既に登録されている。この場合、ユーザーは再登録されます。例えば、Lotus Domino 登録値が新規に指定された値でリセットされます。新規 ID ファイルも作成されます。

注:

1. 変更時にユーザー登録を行う場合は、コネクターの**変更の計算**オプションをオフにします。オンにされていると、**変更の計算**機能によって、ある種のユーザー登録に必要な属性がクリアされ、登録が失敗する場合があります。
2. 変更時にユーザー登録を行う場合は、登録後のユーザーのフルネームを事前に確認することが必要です。そして、**conn** 項目内の **FullName** 属性にその値を指定する必要があります (これは、多くの場合スクリプトにより構成します)。これは、IBM Domino の登録プロセスに関する詳細な知識が必要になるため、あまり便利とはいえません。ただし、ユーザーの正しいフルネームを事前に設定しておかないと、既存ユーザーではなく新規ユーザーが登録されてしまう可能性があります。
3. 変更時にユーザーを登録する場合は、**conn** 項目の **FirstName** 属性に、登録するユーザーの FirstName の値を設定する必要があります。**FirstName** 属性を指定しないと、新規ユーザーが作成される可能性があります。

削除モード:

このコネクターは、IBM Domino 管理プロセスを使用してユーザーを削除します。このことと、構成する必要があるパラメーターの詳細については、以下に示す情報を通じて把握することができます。

コネクターは、**システム管理要求**データベースに**アドレス帳内削除**要求をポストします。**アドレス帳内削除**タイプの要求が IBM Domino 管理プロセスで処理されるたびに、管理要求の送付と処理、および管理プロセスがユーザーを削除するために実行するアクションという処理チェーン全体が起動されます。**アドレス帳内削除**管理要求を送付すると、IBM Domino Administrator を使用して手動でユーザーを削除するのと同じ結果になります。特に、次のことがあげられます。

- 管理要求の処理時間は、Lotus Lotus Domino Server 構成によって異なります。
- 要求された削除のタイプによっては、一連の管理要求に、管理者の承認が必要な要求が含まれることがあります (例えば、ユーザーのメール・ファイルを削除するための**ファイル削除承認**要求)。

このコネクターでは、開始するユーザー削除ごとに調整を行うことができます。構成可能なパラメーターは、以下のとおりです。

メール・ファイルの削除
次のいずれかのオプションを指定できます。

- メール・ファイルを削除しない。
- 個人文書内で指定されているメール・ファイルのみを削除する。
- 個人文書およびすべてのレプリカの中で指定されているメール・ファイルを削除する。

グループへの追加
ユーザーを削除するときに、いずれかのグループにそのユーザーの名前を追加するかどうかを指定します。**はい**の場合は、そのグループの名前も指定します。このオプションは、通常、削除するユーザーを「**アクセス不可グループのみ**」グループに追加して、そのユーザーがサーバーにアクセスすることを拒否するために使用します。

上記の削除パラメーターには、それぞれデフォルト値がありますが、これらは IBM Domino ユーザー・コネクターが提供する API を使用して変更することもできます。IBM Domino ユーザー・コネクターのインスタンスが作成されるときに (特に各 AssemblyLine の開始時)、パラメーターは以下のデフォルト値に設定されます。

メール・ファイルの削除
メール・ファイルを削除しない。

グループへの追加
削除時にユーザーの名前をどのグループにも追加しない。

これらのデフォルト値が、使用する削除のタイプに適合している場合は、削除についての特別な構成は必要ありません。必要なのは、削除コネクターで正しいリンク基準を指定することです。

ただし、IBM Domino ユーザー・コネクターが提供する API を使用して、実行時にデフォルト値を変更することもできます (スクリプトを使用)。

int getDeleteMailFile()
「メール・ファイルの削除」パラメーターのデフォルト値のコードを戻します。

- **0**: メール・ファイルを削除しない。
- **1**: ユーザー文書内で指定されているメール・ファイルのみを削除する。
- **2**: ユーザー文書およびすべてのレプリカの中で指定されているメール・ファイルを削除する。

void setDeleteMailFile (int deleteType)
「メール・ファイルの削除」パラメーターのデフォルト値を設定します。
deleteType メソッドのパラメーターには、希望する値のコードを含める必要があります (コードは getDeleteMailFile() に示したとおりです)。

String getDeleteGroupName ()
「グループへの追加」パラメーターのデフォルト値を戻します。

- NULL: **ユーザーの名前をどのグループにも追加しない**ことを意味します。
- NULL 以外の値: ユーザーの名前を追加するグループの名前。

void setDeleteGroupName (String groupName)

「グループへの追加」パラメーターのデフォルト値を設定します。

- NULL: 削除時にユーザーの名前をどのグループにも追加しないことを指定します。
- NULL 以外のストリング値: 削除時にユーザーの名前を追加するグループの名前を指定します。

削除パラメーターのデフォルト値は、別の値に変更されるかまたはコネクター・インスタンス (オブジェクト) が破棄されるまでは、このコネクターが行うすべての削除操作に適用されます。

以下に、上記のメソッドを使用したシナリオを示します。

- **Before Delete** フック内のスクリプト・コードで、**work** オブジェクトおよび **conn** オブジェクト (および検査を要するその他のすべてのもの) を検査し、特定の決定ロジックに従い、**setDeleteMailFile** および **setDeleteGroupName** を使用して特定のユーザー削除それぞれを調整する。
- 削除対象のすべてのユーザーを 1 つのパターンに従って削除する必要がある (そして特定のユーザー削除それぞれについて調整を行う必要はない) 場合、AssemblyLine プロローグ内のスクリプト・コードで、setDeleteMailFile および setDeleteGroupName メソッドを使用し、プロセス全体についての値を設定できる。

また、**conn** 項目内で以下の属性を設定することによって、削除パラメーターを操作するという方法もあります。

DEL_DeleteMailFile

(整数/ストリング)

conn 項目内にこの属性がない場合は、**メール・ファイルの削除**のデフォルト値が使用されます。

conn 項目内にこの属性が存在する場合は、その値によって、現行の削除のみについての**メール・ファイルの削除**パラメーターの値が決まります。

- **0**: メール・ファイルを削除しない。
- **1**: ユーザー文書内で指定されているメール・ファイルのみを削除する。
- **2**: ユーザー文書およびすべてのレプリカの中で指定されているメール・ファイルを削除する。

DEL_DeleteGroupName

(ストリング)

conn 項目内にこの属性がない場合は、**グループへの追加**のデフォルト値が使用されます。

conn 項目内にこの属性が存在する場合は、その値によって、現行の削除のみについての**グループへの追加**パラメーターの値が決まります。

- NULL: ユーザーの名前をどのグループにも追加しないことを指定します。
- NULL 以外のストリング値: ユーザーの名前を追加するグループの名前を指定します。

conn 項目内で **DEL_DeleteMailFile** 属性および **DEL_DeleteGroupName** 属性を使用すると、現行の削除についてのみ、それぞれに対応する削除パラメーターのデフォルト値が指定変更されます。

conn 項目内で **DEL_DeleteMailFile** 属性および **DEL_DeleteGroupName** 属性を設定するには、**Before Delete** フック内でスクリプトを使用します。ただし、スクリプトを使用して属性を追加するのは必ずしも便利な方法ではないため、多くの場合、デフォルトの削除パラメーターを使用し、API を使用してそれを変更する方が簡単です。

Domino ユーザー属性 (ユーザー文書項目) のリスト

以下に示すリスト (すべてを網羅していない可能性があります) は、サーバーがユーザーについて操作する際に IBM Domino が認識して処理する Domino ユーザー文書項目のそれです。これらの項目について詳しくは、Lotus Domino の資料を参照してください。

コネクターで追加、変更、削除、またはルックアップ操作を行うときは、項目属性名に下記のリストに示されているものと同じ名前を使用してください。

- AltFullName
- AltFullNameLanguage
- AltFullNameSort
- Assistant
- AvailableForDirSync
- CalendarDomain
- CellPhoneNumber
- CcMailUserName
- Certificate
- CheckPassword
- Children
- City
- ClientType
- Comment
- CompanyName
- country
- Department
- DocumentAccess
- EmployeeID
- EncryptIncomingMail
- FirstName
- Form
- FullName
- HomeFAXPhoneNumber
- HTTPPassword

- InternetAddress
- JobTitle
- LastName
- Level0
- Level0_1
- Level0_2
- Level0_3
- Level1
- Level1_1
- Level1_2
- Level1_3
- Level2
- Level2_1
- Level2_2
- Level2_3
- Level3
- Level3_1
- Level3_2
- Level3_3
- Level4
- Level4_1
- Level4_2
- Level4_3
- Level5
- Level5_1
- Level5_2
- Level5_3
- Level6
- Level6_1
- Level6_2
- Level6_3
- LocalAdmin
- Location
- MailAddress
- MailDomain
- MailFile
- MailServer
- MailSystem
- Manager
- MessageStorage

- MiddleInitial
- NetUserName
- NoteID
- OfficeCity
- OfficeCountry
- OfficeFAXPhoneNumber
- OfficeNumber
- OfficePhoneNumber
- OfficeState
- OfficeStreetAddress
- OfficeZIP
- Owner
- PasswordChangeDate
- PasswordChangeInterval
- PasswordGracePeriod
- PersonalID
- PhoneNumber
- PhoneNumber_6
- SametimeServer
- ShortName
- Spouse
- State
- StreetAddress
- Suffix
- Title
- Type
- WebSite
- x400Address
- Zip

IBM Domino Server (Unix/Linux)

スクリプトの PATH 定義の後、開始行の前に、以下に示す 2 行を追加することができます。

Lotus Domino ユーザー・コネクターを Lotus Domino Server for Unix/Linux で使用する場合は、*ibmditk* スクリプトと *ibmdisrv* スクリプトを更新する必要があります。

```
LD_LIBRARY_PATH=Domino_binary_folder
export LD_LIBRARY_PATH
```

ここで、*Domino_binary_folder* は Domino ネイティブ・ライブラリーを含むフォルダーです (例えば、Solaris の場合は /opt/lotus/notes/latest/sunspa、Linux の場合は /opt/lotus/notes/latest/linux)。

IBM Domino ユーザーで IBM Security Directory Integrator を始動します (**root** を使用しないでください)。Lotus Domino Server のインストール時に変更しない限り、Lotus Domino ユーザーは **notes** と呼ばれます。

例

IBM Domino ユーザー・コネクターの例を参照するには、以下に示すパスとリンクを使用します。

関連情報

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/DominoUserConnector ディレクトリーにあります。

『Lotus Domino AdminP コネクター』,

100 ページの『Lotus Notes コネクター』,

IBM Domino でのユーザー登録

Java を使用した IBM Domino でのユーザー登録

Lotus Domino AdminP コネクター

IBM Domino AdminP コネクターは、以下に示す情報を通じて使用することができます。

Lotus Domino 管理プロセスは、数多くのルーチン管理用タスクを自動化するプログラムです。例えば、ユーザー・アカウントを削除する場合、管理プロセスでは、Lotus Domino ディレクトリおよび ACL でそのユーザーの名前を見つけて削除するなど、そのユーザーに関する必要な削除を行います。 Lotus Domino 管理要求データベースに要求を配置すると、管理プロセスによって、必要なアクションがすべて実行されます。

Domino AdminP コネクターは、特殊バージョンの Notes コネクターです。拡張された Lotus Domino AdminP コネクターには、Lotus Domino データベースに文書を追加しているときにフィールドに署名する機能があります。Lotus Notes コネクターと比較すると、「データベース」パラメーターは表示されません (このパラメーターは常に IBM Domino 管理要求データベース「admin4.nsf」に設定されています)。この IBM Domino AdminP コネクターには、「**管理要求タイプ**」という新規パラメーターがあります。

Lotus Domino AdminP コネクターでは、以下のコネクター・モードがサポートされています。

- イテレーター: 管理要求のすべてまたはフィルターにより選別したサブセットに対して繰り返しを行います。
- AddOnly – 管理要求を作成し、署名します。

管理要求への署名

Lotus Domino 管理要求を管理プロセスでさらに処理できるようにするには、admin4.nsf データベースに追加する前に署名する必要があります。文書に署名する

と、ユーザー ID の固有部分が署名済みの注に添付され、署名者が作成者であることが識別できます。この詳細については、以下に示す情報を通じて把握することができます。

署名しない場合、以下のエラーが表示されます。

```
All of the required fields in the request have not been signed.

Cause of error - An unauthorized person or a non-Domino program edited a posted request.
This indicates a failed security attack.
```

Domino AdminP コネクターの特別なコーディングによって、 Domino 文書が保存される前に、文書のすべての項目が署名されます。

注: 文書に署名できるよう、Lotus Domino 管理者にも「無制限にメソッドおよび操作を実行する」権利があります。これを実現するには、「**サーバー**」->「**セキュリティー**」->「**Run Unrestricted methods & operations**」リストに administrator/IBM などのアカウントを追加して、Domino Administrator を使用します。

スキーマ

IBM Domino AdminP コネクターのスキーマおよび 2 つのタイプの管理要求について把握します。

Lotus Domino AdminP コネクターには、一連の事前定義管理要求スキーマがあります。これらのスキーマは、その他すべてのコネクターの入出力スキーマと同様に、構成ファイル `tdi.xml` で説明されています。ただし、相違点が 2 つあります。

- スキーマの名前は、「入力」や「出力」ではなく、指定されている要求名です。
- スキーマには、常に、ProxyAction 属性を指定する必要があります。この属性は、Lotus Domino 管理要求のタイプを識別します。この属性が欠落している場合、この要求タイプに対するフィルター操作が行われません。

現在、このコネクターの構成には、バンドルされている管理要求が 2 種類だけあります。これらの管理要求には、以下の定義があります。

ユーザーの名前変更

表 12. ユーザーの名前変更スキーマ

属性	説明
Form	要求のフォーム。「AdminRequest」にする必要があります。
ProxyAction	作成される要求の ID に対応します。RenameUser の場合、ID は「118」です。
ProxyAuthor	要求の作成者。管理特権を持っている必要があります (CN=administrator/O=IBM など)。
ProxyNameList	変更される Lotus Domino ユーザーの名前。
ProxyNewWebFirstName	ユーザーの新しい名前。
ProxyNewWebLastName	ユーザーの新しい姓。
ProxyNewWebMI	ユーザーの新しいミドルネーム。
ProxyNewWebName	追加される新しいユーザー名。
ProxyProcess	要求のプロセス。「AdminP」にする必要があります。
ProxyServer	ターゲット・サーバー。通常、「*」です。
ProxyWebNameChangeExpires	要求の有効期限。デフォルトは 21 日間です。

表 12. ユーザーの名前変更スキーマ (続き)

属性	説明
Type	要求のタイプ。「AdminRequest」にする必要があります。

グループの名前変更

表 13. グループの名前変更スキーマ

属性	説明
Form	要求のフォーム。「AdminRequest」にする必要があります。
ProxyAction	作成される要求の ID に対応します。RenameGroup の場合、ID は「40」です。
ProxyAuthor	要求の作成者。管理特権を持っている必要があります (CN=administrator/O=IBM など)。
ProxyNameList	変更される Lotus Domino グループの名前。
ProxyNewGroupName	グループの新規名。
ProxyProcess	要求のプロセス。「AdminP」にする必要があります。
ProxyServer	ターゲット・サーバー。通常、「*」です。
ProxyWebNameChangeExpires	要求の有効期限。デフォルトは 21 日間です。
Type	要求のタイプ。「AdminRequest」にする必要があります。

以下のスキーマは、構成エディターで「**属性のディスカバー**」アクションを実行すると戻されます。

すべて

属性は定義されていません。

「ユーザーの名前変更」スキーマおよび「グループの名前変更」スキーマは、必要な値のサンプルを使用して、Lotus Domino ディレクトリのユーザーまたはグループの名前変更に必要なフィールドを定義します。その他のスキーマ (「すべて」) は空で、他のタイプの要求に使用されます。他のタイプの要求を使用するには、有効な属性および対応する新規ドロップダウン項目を持つ新しいスキーマを追加する必要があります。

構成

このコネクターは、提供されたパラメーターを必要とします。

セッション・タイプ
IIOP または **LocalServer** のいずれかを指定できます。 101 ページの『セッション・タイプ』を参照してください。

Domino Server の IP アドレス
Lotus Domino Server の IP ホスト名またはアドレス。また、IOR:<xxx> ストリングを指定して、HTTP 経由でこの自動ディスカバリーを回避することもできます。詳しくは、IOR ストリングに関するセクションを参照してください。

HTTP ポート
コネクターが IIOP セッションを作成するために IOR ストリングを IBM Domino HTTP タスクから取得するときに、このパラメーターを使用します。

ユーザー名
IIOP セッションとローカル・サーバー・セッションで使用されるユーザー名。

パスワード
IIOP セッションとローカル・サーバー・セッションで使用されるインターネット・パスワード。

SSL の使用
このフラグをチェックすると、コネクターは暗号化された IIOP 接続を要求します。このフラグが意味を持つのは、セッション・タイプが IIOP の場合のみです。 SSL を使用するための要件の 1 つに、DIIOP プロセスが開始されるたびに生成される TrustedCerts.class ファイルが、クラスパス内に含まれていなければならないという条件があります。 CLASSPATH に組み込まれているローカル・パスに TrustedCerts.class をコピーするか、またはクラスパス内に IBM Domino をインストールしたシステムの ¥Lotus¥Domino¥Data¥Domino¥Java を含める必要があります。

管理要求タイプ
管理要求のタイプ。リストは、構成ファイルから取得されます。使用可能な値は、「ユーザーの名前変更」、「グループの名前変更」、および「すべて」です。デフォルト値は「すべて」です。

RichText 項目のサポート
チェックすると、項目内と同様に Lotus Domino RichTextItems をマッピングできます。そうしない場合、それらはプレーン・テキストに変換されます。

重要: RichTextItems は順序付け可能ではなく、このオプションを有効にすると、属性が別の Lotus Domino データベースにマッピングされた場合やリモートで使用された場合に例外が発生します。

Domino Server 名
データベースが配置されているサーバーの名前。現在接続しているサーバー (「**Domino Server の IP アドレス**」で指定されているサーバー) を使用する場合は、ブランクのままにしておきます。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

関連情報

78 ページの『Lotus Domino ユーザー・コネクター』,

100 ページの『Lotus Notes コネクター』

Java API。

Lotus Notes コネクター

Lotus Notes コネクターは、Lotus Domino データベースへのアクセスを提供します。これにより、以下に示すタスクを実行できます。

- Notes データベースからの文書とその項目の取得
- 文書の作成
- 文書のフィールドの変更
- 文書の削除
- Notes 文書の「ルックアップ」の実行

注: Lotus Notes コネクターを使用するには、Lotus Notes のリリースが 7.0 以上であることが必要です。

既知の制限事項

ローカル・クライアント・モードまたはローカル・サーバー・モードのみを使用する Lotus Notes コネクター: IBM Security Directory Integrator 構成エディターを使用して Lotus Notes データベースに接続できないことがあります。 こうした制限の詳細については、以下に示す情報を通じて把握することができます。

IBM Notes コネクターが IBM Notes API に対してパスワードを提供する場合であっても、IBM Notes コネクターがユーザーにパスワードを求めるプロンプトを出すことがあります。このプロンプトは標準出力に書き込まれ、ユーザーからの入力は標準入力から読み取られます。このプロンプト表示は Lotus Notes API により実行されるものであり、IBM Security Directory Integrator の制御の範囲外です。

- IBM Security Directory Integrator サーバーを実行すると、標準入力と標準出力の両方がコンソールに接続されるため、ユーザーに対してプロンプトが表示され、ユーザーがパスワードを入力することができます。 IBM Notes コネクターは制御を再獲得し、実行を続けます。これは、このコネクターが予期したとおりに働いているということです。
- IBM Security Directory Integrator 構成エディターを実行すると、標準入力および標準出力はコンソールから切断されるため、ユーザーに対して何も表示されず、応答も入力できません。接続操作は、無期限にユーザー入力を待ってハングすることがあります。

セッション・タイプが「**LocalClient**」である場合は、Notes または Designer クライアントを始動し、「**ファイル**」->「**セキュリティー**」->「**ユーザーセキュリティ**」パネルでフラグを設定することにより、他のアプリケーションがその接続を使用できるようにすることが可能です。「基本 (Security Basics)」をクリックし、「ログインとパスワードの設定」で「**他の Notes ベースのプログラムでパスワードプロンプトを表示しない (セキュリティが低下)**」を選択します。この場合、IBM Notes コネクター (つまり IBM Notes API) は、指定されたパスワードを無視し、IBM Notes または Designer クライアントが確立した現行セッションを再使用します。IBM Security Directory Integrator がこのセッションを再使用するには、IBM Notes または Designer クライアントが実行されている必要があります。

注: いったん DIIOP モードに切り替えて AssemblyLine を構成してから、再びローカル・クライアント・モードまたはローカル・サーバー・モードに戻し、IBM Security Directory Integrator サーバーで AssemblyLine を実行することができます。

セッション・タイプ

IBM Lotus Notes コネクターでは、3 つのタイプのセッションをサポートしています。これらの詳細については、以下に示すセクションに記載されています。

サポートされるセッション・タイプを以下に示します (ライブラリー、セットアップ、および他の Lotus Domino コネクターとの非互換性についての詳細は、「コネクター別のサポートされるセッション・タイプ」も参照してください)。

IIOP このセッション・タイプでは、Domino Server への TCP 接続が使用されます。Lotus Notes コネクターは、HTTP および IIOP を使用して IBM Domino Server にアクセスするため、これらのサービスが開始されていて、Lotus Notes コネクターを実行しているホストからアクセスできることを確認してください。

LocalClient

このセッション・タイプは、Lotus Notes または Designer のローカル・インストールを使用します。 Lotus Notes コネクターは、ローカル・クライアントが使用している ID ファイルを使用します。

このセッション・タイプでは、「**ユーザー名**」パラメーター (dominoLogin) は無視されます。**パスワード** (dominoPassword) は、使用されている ID ファイル内のパスワードと一致していなければなりません。一致しない場合は、ローカルの IBM Notes Client はパスワードを求めるプロンプトを出します。このセッション・タイプでは、「**Domino Server の IP アドレス**」、「**IOR ストリング**」、「**HTTP ポート**」、および「SSL の使用」の各パラメーターも無視されます。

注: この操作は困難な場合があります。例えば、コンソールから切り離された標準入力または標準出力を使用して AssemblyLine を実行する場合です。AssemblyLine は常にコマンド行ウィンドウで実行して、ローカル・クライアントがパスワードを求めるかどうかを確認するようにしてください。テストの結果によれば、ローカル・クライアントは正しい**パスワード**・パラメーターを無視して、常にパスワードを求めるプロンプトを出します。プロンプトが出されないようにする 1 つの方法は、次のステップを実行することです。

1. IBM Notes または Designer クライアントを始動します。
2. 「**ファイル**」->「**ツール**」->「**ユーザー ID**」メニューを選択します。
3. 「**他の Lotus Notes ベースのプログラムでパスワードプロンプトを表示しない**」をチェックします。

LocalServer

これは **LocalClient** と同じですが、ローカルの IBM Domino Server インストールを使用します。異なるのは、有効な**ユーザー名**および一致する**パスワード**を指定できるという点です。ただし、このセッション・タイプでは、「**Domino Server の IP アドレス**」、「**IOR ストリング**」、「**HTTP ポート**」、および「SSL の使用」の各パラメーターも無視されます。

IIOP を使用した接続

このコネクターは、IIOP を使用して Domino Server と通信できます。IBM Domino Server との IIOP セッションを確立するには、コネクターは、サーバー上で IIOP プロセスを検出するための IOR ストリングを必要とします。

IBM Notes コネクターを構成するときに、ホスト名の他に、オプションとして、サーバーが配置されているポート番号を指定します。この ***hostname*:*port*** ストリングは、実際には、コネクターが IOR ストリングを検索する IBM Domino Server の http サービスのアドレスです。そして、この IOR ストリングを使用して、サーバーの IIOP サービス (diiop) との IIOP セッションが作成されます。この http サービスが必要なのは、IOR ストリングを検出するためだけです。この操作は非常に単純です。コネクターは、IOR ストリングが含まれているものと予期される **/diiop_ior.txt** という名前の文書を、IBM Domino http サーバーに要求します。***hostname*:*port*** の指定をこのストリングで置き換えれば、最初のステップを省略し、http サーバーへの依存を回避することができます。通常、diio_ior.txt ファイルは、Lotus Domino Server インストール・ディレクトリー内の `data/domino/html` ディレクトリーに入っています。正確な場所については、Lotus Administrator の中の Web 構成を調べてください。

最初のステップを確認するには、http://*hostname*:*port*/diiop_ior.txt という URL を参照してください。ここで、*hostname* はホスト名、*port* は IBM Domino Server のポート番号です。ユーザーは、IOR: *numbers* という文書を受け取ります。このような応答を受け取った場合は、最初のステップは確認されたことになります。この操作が失敗した場合は、サーバー上の HTTP 構成を調べて、無名アクセスを許可するように設定されていることと、プロセスが実行されていることを確認してください。

注: IIOP セッションの構成時に、構成エディターでコネクターの構成の使用可能なデータベースをブラウズするには、Lotus Domino Server でこのブラウズがサポートされ、使用可能なデータベース、ビュー、フォーム、およびエージェントのリストを使用してさまざまな制御を取り込むことができる必要があります。データベースでブラウズを使用可能にする Domino Server 設定は、**「サーバー文書」->「インターネット・プロトコル (IP)」->「HTTP」タブ ->「HTTP クライアントからのデータベース参照を許可」**にあります。「はい」に設定し、Lotus Domino Server を再始動する必要があります。

構成

このコネクターは、提供されたパラメーターを必要とします。これらのパラメーターとその値については、以下に示すセクションで把握することができます。

セッション・タイプ

IIOP、**LocalClient**、または **LocalServer** のいずれかを指定できます。 101 ページの『セッション・タイプ』を参照してください。

Domino Server の IP アドレス

Lotus Domino Server の IP ホスト名またはアドレス。また、IOR:<xxx> ストリングを指定して、HTTP 経由でこの自動ディスカバリーを回避することもできます。詳しくは、IOR ストリングに関するセクションを参照してください。

HTTP ポート
コネクターが IIOP セッションを作成するために IOR ストリングを IBM Domino HTTP タスクから取得するときに、このパラメーターを使用します。

ユーザー名
IIOP セッションとローカル・サーバー・セッションで使用されるユーザー名。セッション・タイプが **LocalClient** である場合、これは無視されます。

パスワード
IIOP セッションとローカル・サーバー・セッションで使用されるインターネット・パスワード。ローカル・クライアント・セッションの場合は、Notes ID ファイルのパスワードです。

SSL の使用
このフラグをチェックすると、コネクターは暗号化された IIOP 接続を要求します。このフラグが意味を持つのは、セッション・タイプが IIOP の場合のみです。 SSL を使用するための要件の 1 つに、DIIOP プロセスが開始されるたびに生成される TrustedCerts.class ファイルが、クラスパス内に含まれていなければならないという条件があります。 CLASSPATH に組み込まれているローカル・パスに TrustedCerts.class をコピーするか、またはクラスパス内に IBM Domino をインストールしたシステムの ¥Lotus¥Domino¥Data¥Domino¥Java を含める必要があります。

RichText 項目のサポート
チェックすると、項目内と同様に Lotus Domino RichTextItems をマッピングできます。そうしない場合、それらはプレーン・テキストに変換されます。

重要: RichTextItems は順序付け可能ではなく、このオプションを有効にすると、属性が別の Lotus Domino データベースにマッピングされた場合やリモートで使用された場合に例外が発生します。

Server **データベース**が配置されているサーバーの名前。現在接続しているサーバー (「**Domino Server の IP アドレス**」で指定されているサーバー) を使用する場合は、ブランクのままにしておきます。

データベース
使用するデータベースの名前。

文書選択
データ・ソースを繰り返すときに使用する選択肢 (このパラメーターは **イテレーター**・モードの場合にのみ使用します)。有効な Lotus Notes 選択ステートメントを使用する必要があります。アドレス帳から項目を選択するには、次の選択ステートメントを使用します。

```
Select Form="Person"
```

常に公式検索を使用
このフラグは、ビューが設定されておらず、アクセスするデータベースがフルテキスト索引付きデータベースである場合に使用します。このフラグをチェックすると、コネクターは、データベースが索引付きかどうかにかかわら

ず、公式ステートメントを使用します。ビューは公式検索ステートメントをサポートしていないため、ビューを指定した場合は、常にフルテキスト検索が使用されます。

データベース・ビュー
使用するデータベース・ビュー。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

メモリーの節約
検索ビューの反復時にメモリーを節約する必要があるかどうかを示します。

UNID のサポート

UNID は、Notes 文書の普遍的に固有な ID です。この ID は、データベース・レプリカ間でも一意です。検索条件に UNID を追加するには、以下に示す手順を実行します。

Lotus Notes API では、Lotus Notes/Domino 検索機能に渡される検索フィルターで直接 UNID を使用することはできません。このため、検索条件で UNID を使用するオプションを追加すると、以下のようになります。

- 「リンク基準」に UNID がある場合は、基準のその他すべてのフィールドがコネクターによって無視され、文書検索が UNID によってのみ実行されます。
- 複数の UNID が指定されている場合は、最初の一致が採用されます。
- 一致操作が等しくない場合は、例外がスローされます。
- 指定された UNID で文書が見つからない場合は、例外がスローされます。
- 指定された UNID で文書が見つかった場合、コネクターはその文書が削除スタブではないかどうかをチェックします。削除スタブではない場合、文書は処理されます。
- 「リンク基準」に UNID がない場合、検索フィルターは IBM Security Directory Integrator 6.0 と同様に作成されます。

ここで説明されている機能は、以前のバージョンとの互換性の問題の原因とはなりません。これは、Lotus Notes API では、公式/検索フィルターで UNID を使用できないためです。

RichText 属性のサポート

Lotus Domino 文書には、タイプ `lotus.domino.RichTextItem` の項目が含まれています。通常、読み取り時には、このような項目は、項目の属性マップでプレーン・テキストとして受け取られるか、または逆に、Domino 文書にプレーン・テキストとして書き込まれます。コネクターでは、このような項目の追加機能がサポートされます。

イテレーター・モード
「**RichText 項目のサポート**」パラメーターをチェックして、コネクターが Lotus Domino データベースから文書を読み取るときに lotus.domino.RichTextItem が見つかった場合、項目に lotus.domino.RichTextItem として配置するか、または項目にストリングとして配置するようコネクターの動作を変更します。

追加/更新モード

Lotus Domino 文書を追加/更新する場合、項目に lotus.domino.RichTextItem オブジェクトを提供し、その文書に書き込まれるようにするオプションがあります。この RichTextItem は、AssemblyLine の他の場所から受け取るか、またはスクリプトを使用して作成できます。

スクリプトの例:

RichTextItem の作成および RichTextItem からの添付ファイルの抽出について、数例のスクリプトを示します。

RichTextItem の作成

```
var rti = NotesConnectorName.connector.getDominoSession().getDatabase("", <database_name>)
     .createDocument().createRichTextItem("Body");
var header = NotesConnectorName.connector.getDominoSession().createRichTextStyle();

header.setBold(lotus.domino.RichTextStyle.YES);
header.setColor(lotus.domino.RichTextStyle.COLOR_YELLOW);
header.setEffects(lotus.domino.RichTextStyle.EFFECTS_SHADOW);
header.setFont(lotus.domino.RichTextStyle.FONT_ROMAN);
header.setFontSize(14);

rti.appendStyle(header);
rti.appendText("Sample text which will be formatted with the above style.");

work.setAttribute("Body", rti);
```

RichTextItem からの添付ファイルの抽出

```
var doc = NotesConnectorName.connector.getDominoSession().getDatabase("", <database_name>)
     .getAllDocuments().getFirstDocument();
      if (doc.hasEmbedded()) {
        var body = doc.getFirstItem("Body");
        var rtnav = body.createNavigator();
        if (rtnav.findFirstElement(
        lotus.domino.RichTextItem.RTELEM_TYPE_FILEATTACHMENT)) {
          do {
            var att = rtnav.getElement();
            var path = "c:¥¥Files¥¥" + att.getSource();
            att.extractFile(path);
            main.logmsg(path + " extracted");
          } while (rtnav.findNextElement());
        }
        else
          main.logmsg ("No attachments");
      }
      else
        main.logmsg ("No attachments or embedded objects");
```

詳細および例については、http://www-128.ibm.com/developerworks/lotus/documentation/dominodesigner/ にある Domino の資料を参照してください。

RichText の制限:

IBM Lotus Notes コネクターの RichText 属性には、一定の制限があります。この詳細を把握するには、以下に示すセクションをお読みください。

lotus.domino.RichTextItem クラスはシリアライズ可能ではなく、このタイプの項目は、RMI を介して転送できません。これにより、このタイプのオブジェクトにリモート・サーバー API を介してアクセスすると、java.io.NotSerializableException が発生します。非シリアライズ可能オブジェクトが項目内に到達すると、項目全体が非直列化可能となります。

注: これは、lotus.domino.DateTime クラスでも同様です。
また、このシリアライゼーションの制限によって、lotus.domino.RichTextItems のさまざまな Lotus Domino データベース間での転送が制限されます。このため、RichTextItem の追加/更新は、*同じデータベース* の別の RichTextItem または IBM Domino API を使用したスクリプトで同じデータベース用に作成された RichTextItem でのみ可能となります。

割り当て量およびファイル所有権の設定

以下に示すサンプル・コードは、割り当て量およびファイル所有権の設定に使用できます。

コネクターは、Lotus Notes データベース項目を直接操作できますが、データベース・プロパティーを操作することはできません。ただし、スクリプト・コンポーネントおよび構成された Lotus Notes コネクターを使用し、スクリプトによってデータベース割り当て量を設定できます。以下のサンプル・コードでは、ファイル・サイズ割り当て量が設定され、必要な MailOwnerAccess が ACL に書き込まれます。

```
//NotesIterator is the NotesConnector name in the AssemblyLine
var db = NotesIterator.connector.getDominoDatabase(null);
//uses the public getDominoDatabase(...) method of the NotesConnector class;
//giving null for method parameter will return the database configured in the Connector
main.logmsg("Old quota: " + db.getSizeQuota());
//should print the old database size quota
db.setSizeQuota(5000);
//sets the size quota to 5000KB
main.logmsg("New quota: " + db.getSizeQuota());
//will print the new database size quota in kilobytes, i.e. 5000
var acl = db.getACL();
//get the database access control list
var ACLEntry = acl.createACLEntry("DesiredNotesUser", lotus.domino.ACL.LEVEL_MANAGER);
//create new ACL Entry
ACLEntry.setUserType(lotus.domino.ACLEntry.TYPE_PERSON); //set user type equal to Person
acl.save();
//save the access control list
```

セキュリティー

IBM Lotus Notes コネクターのセキュリティー設定については、以下に示す情報を通じて把握することができます。

IBM Security Directory Integrator が Domino Server にアクセスできるようにするには、「**Domino Administrator**」->「**セキュリティー**」->「**IIOP 制限 (IIOP restriction)**」を使用する必要があります。IBM Security Directory Integrator が使用するものとして構成したユーザー・アカウントは、「**制限付き Java/Javascript の実行**」および「**制限なし Java/Javascript の実行**」の下にリストされているグループのいずれかに属していなければなりません。

IBM Domino Web サーバーは、無名アクセスを許可するように構成する必要があります。このように構成しておかないと、現行バージョンの Lotus Notes コネクターは Lotus Domino IIOP サーバーに接続できません。

注: IBM Notes アドレス帳の **HTTPPassword** フィールドを暗号化する場合は、次のコードを AssemblyLine に追加します。

```
var pwd = "Mypassword";
var v = dom.connector.getDominoSession().evaluate
("@Password(¥"" + pwd + "¥")" ) ;
ret.value = v.elementAt(0);
```

このコードは、IBM Domino のパスワード暗号化ルーチンを使用して、変数 *pwd* を暗号化します。このコードは、IBM Domino が提供する **@Password** 関数を使用してストリングを暗号化する任意の場所で使用できます。このコードを使用する最も適切な場所の 1 つは、**HTTPPassword** 属性の**出力マップ**です。

関連情報

Lotus Notes に関する Wikipedia。

TIM DSMLv2 コネクター

このコネクターは、IBM Security Identity Manager との通信を必要とするソリューションで使用することができます。

ITIM サーバーは、ITIM 専用バージョンの DSMLv2 を使用する通信インターフェースを提供します。この ITIM 専用バージョンの DSMLv2 は、DSMLv2 仕様に完全に準拠しているわけではありません。このため、コネクター名は、*TIM DSMLv2* コネクター です。

このコネクターは、以下の両方の目的で使用します。

- プロビジョニング・データの取得
- プロビジョニング・データのフィード

この場合、ITIM 専用の DSMLv2 通信インターフェースを使用します。

サポートされる ITIM のバージョンは 5.0 以上です。

Directory Services Markup Language v1.0 (DSMLv1) を使用すると、ディレクトリー構造情報を XML 文書として表現できます。DSMLv2 はさらに進歩し、ディレクトリーの照会や更新 (そしてこれらの操作の結果) を XML 文書として表現するメソッドを提供します。

注: このコネクターは、ITIM とともに使用するよう*特別に設計* されています。一般的な使用の場合、このコネクターではなく、DSMLv2Soap Connector または DSMLv2SoapServer Connector あるいはその両方を使用します。

HTTP を介して DSML を使用して IBM Security Identity Manager サーバーのリポジトリーに接続する TIM DSMLv2 コネクター。

このコネクターは、DSMLv2 ITIM イベント・ハンドラー (ITIM 4.5 で導入) に接続します。このイベント・ハンドラーは、DSMLv2 サーバーとして機能する ITIM を使用してデータを ITIM にインポートできます。したがって、サポートされているのは ITIM サーバー 4.5 以上のみです。TIM DSMLv2 コネクターは、ITIM サーバーに接続して通信するために、ITIM DSML JNDI ドライバー「dsml2.jar」を使用します。DSMLv2 コネクターのデプロイメントは、JNDI 照会を使用して ITIM リポジトリーと対話します。

このコネクターがサポートしているのは、**AddOnly**、**削除**、**イテレーター**、**ルックアップ**、および**更新** の各モードです。

更新モードおよび削除モードでのルックアップのスキップ

TIM DSMLv2 コネクターでは、更新モードまたは削除モードでの「**ルックアップのスキップ**」一般オプションがサポートされています。

このオプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。正常に機能させるためには、「**名前**」パラメーター (例えば、LDAP の場合は $dn など) を指定する必要があります。

ITIM サーバーでのコネクターの使用方法

ITIM サーバーに接続するには、以下に示すパスを使用できます。また、ここで述べる制限事項にも注意が必要です。

ITIM サーバーに接続する際は、TIM DSMLv2 コネクターに次の URL を指定する必要があります。http://<*ITIM_Server_host:ITIM_Server_port*>/enrole/dsml2_event_handler。例えば、「http://192.168.113.12:9080/enrole/dsml2_event_handler」となります。

ITIM サーバーと対話するとき、TIM DSMLv2 コネクターの各モードには次のような制限があります。

- イテレーター・モード – 指定した JNDI フィルターに一致する項目が 1 つの場合のみ処理が行われます。フィルターに一致する項目が複数ある場合は、項目は戻されません。
- ルックアップ、更新、および削除 – 指定したリンク基準で見つかった項目が 1 つの場合のみ正しく処理が行われます。リンク基準が複数の項目に一致する場合、コネクターはリンク基準に一致した項目が 1 つもない場合と同じ動作をします。

ITIM サーバーと対話するときの JNDI 照会およびフィルターは、GUI から使用する場合も、スクリプト記述 (例えば、拡張検索基準) 内で使用する場合も、すべて括弧で囲む必要があります。例えば、「(uid=user1)」のようにします。

HTTPS (SSL) のサポート

DSMLv2 サーバーとの間でセキュアな HTTPS 接続を使用するためには、指定するプロバイダー URL の先頭を「https://」にします。また、サーバーの証明書を IBM Security Directory Integrator のトラストストアに組み込む必要があります。

構成

TIM DSMLv2 コネクターは、提供されたパラメーターを必要とします。以下に示すリンクを使用して、追加情報を参照することもできます。

プロバイダー URL
接続の URL。

参照 LDAP サーバーが見つけた参照をどのように処理するかを指定します。指定できる値は、以下のとおりです。

- **follow**: 自動的に参照に従います。
- **ignore**: 参照を無視します。

- **throw**: 参照が見つかった時点で ReferralException をスローします。これはエラー・フックの中で処理する必要があります。

認証メソッド
認証メソッド。

ログイン・ユーザー名
プリンシパル名 (ユーザー名など)。

ログイン・パスワード
信任状 (パスワードなど)。

名前パラメーター
AssemblyLine 項目内のどのパラメーターを項目の名前に使用するかを指定します。これは、追加、変更、および削除操作で使用され、読み取りまたは検索操作時に戻されます。このパラメーターを指定しない場合は、「$dn」が使用されます。

プロバイダー・パラメーター
プロバイダーに受け渡す追加のプロバイダー・パラメーターのリスト。それぞれの「parameter:value」は別々の行に指定してください。

検索ベース
ディレクトリーの繰り返しの際に使用する検索ベース。識別名を指定します。ディレクトリーによっては、ブランク・ストリングを指定することができます。その場合は、デフォルトにより、サーバーについて設定されている処置がとられます。ディレクトリー・サービスによっては、そのディレクトリー内の有効な識別名を指定しなければならないこともあります。

検索フィルター
ディレクトリーの繰り返しの際に使用する検索フィルター。

検索範囲
データ・ソースの繰り返しの際に使用する検索範囲。指定できる値は、以下のとおりです。

- subtree: 検索ベースおよびその下位にあるすべてのレベルを検索します。
- onelevel: 検索ベースの直接の子のみ検索します。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

関連情報

178 ページの『ITIM Agent コネクター』,

DSML ID フィード。

DSMLv2 SOAP コネクター

DSMLv2 SOAP コネクターは、DSMLv2 標準をインプリメントします。以下にリストするタスクの実行には、このコネクターが役立ちます。

- DSML サーバーに対する DSMLv2 要求の実行。
- DSML SOAP バインディングを使用するためのオプションの提供。

- HTTP 要求の作成と HTTP 応答の解析のための HTTP パーサーの内部インスタンス化、構成、および使用。
- DSMLv2 要求メッセージの作成と DSMLv2 応答メッセージの解析のための DSMLv2 パーサーの内部インスタンス化、構成、および使用。

サポートされるコネクター・モード

コネクター・モードにより、コネクターが要求する DSML 操作のタイプが決定します。DSMLv2 SOAP コネクターは、以下に示すモードをサポートします。

AddOnly

DSMLv2 SOAP コネクターは DSMLv2 addRequest を送信し、DSMLv2 addResponse メッセージを受信します。

イテレーター

DSMLv2 SOAP コネクターは、現行コネクター構成から取得された検索ベース、検索フィルター、および検索範囲が設定された DSMLv2 searchRequest 操作を送信します。DSML サーバーは複数の searchResultEntry エレメントが含まれている DSMLv2 searchResponse メッセージを戻します。コネクターは DSML searchResultEntry エレメントを循環処理し、各エレメントを個別の AssemblyLine の反復で引き渡します。

ルックアップ

DSMLv2 SOAP コネクターは、コネクターのリンク基準から作成された検索フィルターが設定された DSMLv2 searchRequest を送信します。DSML サーバーから、検出された項目として DSMLv2 searchResponse メッセージが戻されます。searchResponse メッセージに複数の searchResultEntry エレメントが含まれている場合は、「**複数項目時**」フックでこれらのエレメントを処理する必要があります。

削除 コネクターはリンク基準に基づいて DSML deleteRequest を作成および送信します。DSML サーバーから deleteResponse メッセージが戻されます。

更新 作業項目の `$dn` 属性が更新対象項目の `$dn` 属性と同等である場合に、コネクターは modifyRequest DSMLv2 要求を送信し、modifyResponse 応答を受信します。それ以外の場合は、modDnRequest 要求を DSML サーバーに送信し、modDnResponse 応答を受信します。

デルタ デルタ・モードでは、項目タグに基づいて、呼び出すコネクター・メソッドと送信する DSMLv2 要求を AssemblyLine が決定します。属性レベルでのデルタ・タグは DSMLv2 パーサーにより処理され、その結果作成される DSMLv2 要求にデルタ情報が組み込まれます。

DSMLv2 SOAP コネクターは、modEntry メソッドに「newrdn」属性があるかどうかを検出し、ある場合はターゲット $dn の rdn を新しい値で置き換えます。その後、modDnRequest 要求が DSML サーバーに送信され、modDnResponse 応答を受信します。

CallReply

CallReply モードでは、コネクターが作業項目を DSMLv2 パーサーに受け渡し、DSMLv2 パーサーにより作成された DSMLv2 メッセージを送信します。DSMLv2 サーバーからの応答は直接 DSMLv2 パーサーに受け渡され、作成された項目がコネクターにより戻されます。コネクターでは DSMLv2 エレメントが自動的に設定されることはないため、正しい要求タイプを割り

当てる必要があります。 CallReply モードでは、DSMLv2 拡張操作を送信できます。詳しくは、『拡張操作』を参照してください。

拡張操作

CallReply モードの DSMLv2 SOAP コネクターは、DSMLv2 拡張操作を送信できます。このことについては、ここに記載されている情報を通じて詳しく知ることができます。

拡張操作は操作 ID (OID) により識別されます。例えば、IBM Security Directory Server のログ・ファイルの一部を取得するための拡張操作の OID は `1.3.18.0.2.12.22` です。

拡張操作には value プロパティーも指定できます。このプロパティーは、対応する操作の入力データが格納されるデータ構造です。拡張操作の `value` プロパティーのエンコードは Basic Encoding Rules (BER) でなければなりません。また、DSMLv2 メッセージでは base-64 エンコードでなければなりません。DSMLv2 SOAP コネクターのユーザーは、BER エンコードの value プロパティーについてのみ責任を持ちます。 DSMLv2 メッセージの作成時に、コネクターにより自動的にデータが Base 64 でエンコードされます。

BER のエンコードとデコードには、BEREncoder と BERDecoder という 2 つのクラスが使用されます。これらのクラスは `com.ibm.asn1` パッケージに含まれています。

DSMLv2 拡張操作要求の送信と応答の処理を次の例に示します。

1. 以下のスクリプト・コードを属性 `dsml.extended.requestvalue` の出力マップに配置します。

```
enc = new Packages.com.ibm.asn1.BEREncoder();
serverFile = 1;  //slapdErrors log file

nFirstLine = new java.lang.Integer(7200);
nLastLine = new java.lang.Integer(7220);

seq_nr = enc.encodeSequence();
enc.encodeEnumeration(serverFile);

enc.encodeInteger(nFirstLine);
enc.encodeInteger(nLastLine);

enc.endOf(seq_nr);
var myByte = enc.toByteArray();

ret.value = myByte;
```

2. 以下のスクリプト・コードをコネクターの「`CallReply 後`」フックに配置します。

```
var ba = conn.getAttribute("dsml.response").getValue(0);
bd = new Packages.com.ibm.asn1.BERDecoder(ba);

main.logmsg("SLAPD log file:");
main.logmsg(new java.lang.String(bd.decodeOctetString()));
```

SOAPAction ヘッダー

DSMLv2 SOAP コネクターは、デフォルトで常に、SOAPAction ヘッダーに対して空のヘッダーを送信します。ここに記載されている情報を使用することで、SOAPAction ヘッダーについて詳しく知ることができます。

SOAP 状態に関する OASIS 標準は、「各 SOAP 要求本体には、単一の batchRequest が含まれます。SOAP ノードは、「SOAPAction」ヘッダー・フィールドで、SOAP 要求の <body> 内の最上位エレメントのエレメント名を示す必要があります。このヘッダーは空でも有効ですが、必要に応じて値を設定できる必要があります。また、一部のベンダーは、DSML 定義でこのヘッダーを必須と定義しています (例として Sun 社を挙げることができます。http://docs.oracle.com/cd/E19261-01/820-2765/6nebir7ld/index.html を参照)。

必要に応じて、「**SOAPAction ヘッダー**」パラメーターを使用して、SOAPAction ヘッダーを自身で作成できます。

構成

DSMLv2 SOAP コネクターには、以下に示すパラメーターが使用できます。

DSMLv2 サーバー URL
DSMLv2 サーバーの URL を指定します。

認証メソッド
HTTP 認証のタイプを指定します。HTTP 認証のタイプが「無名」に設定されている場合は、認証は実行されません。「HTTP 基本認証 (BA)」が指定されている場合は、「ユーザー名」と「パスワード」パラメーターに指定されているユーザー名とパスワードを使用して HTTP 基本認証が実行されます。

ユーザー名
HTTP 基本認証に使用されるユーザー名。

パスワード
HTTP 基本認証に使用するパスワード

バイナリー属性
コネクターにバイナリー属性として処理させる属性を、コンマで区切ったリストとして指定します。このパラメーターのデフォルト属性リストを以下に示します。これらの属性は変更できます。

- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedData
- thumbnailPhoto
- thumbnailLogo
- userPassword
- userCertificate
- authorityRevocationList

- certificateRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- objectGUID
- objectSid

検索ベース
繰り返し時の検索開始点を指定します。

検索フィルター
繰り返し時に使用する LDAP フィルターを指定します。

検索範囲
繰り返し時に使用する検索範囲。指定できる値は、以下のとおりです。

- subtree
- onelevel

デフォルトはサブツリーです。

SOAP バインディング
このパラメーターを使用可能にすると、コネクターは SOAP DSML メッセージを送受信します。それ以外の場合は、DSML メッセージは SOAP によりラップされません。

SOAPAction ヘッダー
SOAP バインディングが使用可能な場合に組み込む SOAPAction ヘッダー値。デフォルト・ヘッダー値は空です。

詳細ログ
デバッグ・メッセージを有効にします。このパラメーターはすべての IBM Security Directory Integrator コンポーネントで共通です。

DSMLv2 SOAP サーバー・コネクター

DSMLv2 SOAP サーバー・コネクターは、HTTP を介して DSMLv2 要求を listen します。コネクターは要求を受け取るとその要求を解析し、解析した要求を処理のために AssemblyLine ワークフローに送信します。結果は HTTP を介してクライアントに送り返されます。以下に示すタスクは、DSMLv2 SOAP サーバー・コネクターで実行できます。

DSMLv2 SOAP サーバー・コネクターは、以下の操作を実行できます。

- DSML サーバーに対する DSMLv2 要求の実行。
- DSML SOAP バインディングを使用するためのオプションの提供。
- HTTP 要求の作成と HTTP 応答の解析のための HTTP パーサーの内部インスタンス化、構成、および使用。
- DSMLv2 要求メッセージの作成と DSMLv2 応答メッセージの解析のための DSMLv2 パーサーの内部インスタンス化、構成、および使用。
- 個別スレッドでの各イベントの処理。これにより、コネクターは複数の DSMLv2 イベントを並列実行できます。

拡張操作

DSMLv2 SOAP サーバー・コネクターでは、拡張操作がサポートされています。以下に示す情報を使用することで、この操作を実行し、ヘルパー・クラスを使用することができます。

拡張操作の value プロパティーは、DSMLv2 メッセージから自動的に base-64 デコードされます。次に、この値をデコードするための Basic Encoding Rules (BER) をプローブする必要があります。また、dsml.response 項目属性により表される responseValue プロパティーを BER でエンコードする必要があります。DSMLv2 応答の作成および送信時に、コネクターによりデータが自動的に base-64 でエンコードされます。

データを BER でエンコードおよびデコードするために使用できる 2 種類のヘルパー・クラスを以下に示します。

- com.ibm.asn1.BEREncoder
- com.ibm.asn1.BERDecoder

注: このコネクターでは拡張操作のスキーマを自動的に判別できません。拡張操作要求の構造を記述するメタデータがありません。

IBM Security Directory Server ログの一部を戻す拡張操作要求の例を以下に示します。

```
var name = work.getString("dsml.extended.requestname");
   var ba = work.getAttribute("dsml.extended.requestvalue").getValue(0);

decoder = new Packages.com.ibm.asn1.BERDecoder(ba);
iSecuence = decoder.decodeSequence();
fileNumber = decoder.decodeEnumeration();
firstLine = decoder.decodeIntegerAsInt();
lastLine = decoder.decodeIntegerAsInt();

main.logmsg("Operation: " + name);
main.logmsg("File: " + fileNumber);
main.logmsg("First line: " + firstLine);
main.logmsg("Last line: " + lastLine);

// send the response, assuming this sample string is the log file content
var str = new java.lang.String("Apr 13 16:18:18 2005
     Entry cn=chavdar kovachev,o=ibm,c=us already exists.");

enc = new Packages.com.ibm.asn1.BEREncoder();
enc.encodeOctetString(str.getBytes());
myByte = enc.toByteArray();

work.setAttribute("dsml.response", myByte);
work.setAttribute("dsml.responseName", "1.3.18.0.2.12.23");
work.setAttribute("dsml.resultdescr", "success");
```

構成

DSMLv2 SOAP サーバー・コネクターの構成には、以下にリストするパラメーターを使用することができます。

DSML ポート
DSMLv2 SOAP サーバー・コネクターが listen する TCP ポートを指定します。

接続バックログ
着信接続の最大キュー長を指定します。接続要求の着信時にキューがいっぱいである場合、接続は拒否されます。

HTTP 基本認証 (BA)
クライアントが HTTP 基本認証を提供する必要があるかどうかを決定します。

認証レルム
HTTP 基本認証を要求するときにクライアントに送信する認証レルムを指定します。

バイナリー属性
コネクターにバイナリー属性として処理させる属性を、コンマで区切ったリストとして指定します。

このパラメーターのデフォルト属性リストを以下に示します。これらの属性は変更できます。

- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedData
- thumbnailPhoto
- thumbnailLogo
- userPassword
- userCertificate
- authorityRevocationList
- certificateRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- objectGUID
- objectSid

SSL の使用
これをチェックすると、コネクターの初期化時に Secure Sockets Layer (SSL) が使用されます。

クライアント認証を必要とする
これをチェックすると、コネクターでは SSL を使用したクライアント認証が必要となります。

チャンク転送エンコード
これをチェックすると、応答メッセージの HTTP BODY が一連のチャンクとして転送されます。

SOAP バインディング
これをチェックすると、コネクターは SOAP DSML メッセージを送受信します。それ以外の場合は、DSML メッセージは SOAP によりラップされません。

詳細ログ
デバッグ・メッセージを有効にします。このパラメーターはすべての IBM Security Directory Integrator コンポーネントで共通です。

EIF コネクター

EIF Connector として知られる Event Integration Facility については、以下に示す情報を通じて把握することができます。

IBM Security Directory Integrator では、Netcool/OMNIbus や IBM Tivoli Enterprise Console などのエンタープライズ・システムとの統合プロセスで、Event Integration Facility (EIF) の機能を使用します。EIF によって IBM Security Directory Integrator は、Netcool/OMNIbus イベント管理システムでイベントとして認識されるアラートおよび状況情報を作成し、送信できます。

EIF コネクターによって IBM Security Directory Integrator は、EIF イベント・メッセージを送受信でき、 TEC や Netcool/Omnibus などの EIF 対応システムとの双方向通信を可能にします。

送信は、コネクターの AddOnly モードを使用して行われ、イベントの読み取りは、イテレーター・モードで処理されます。

IBM Tivoli Netcool/OMNIbus の概要

IBM Tivoli Netcool®/OMNIbus はサービスレベル管理 (SLM) システムで、さまざまなネットワーク・データ・ソースからエンタープライズ全体のイベント情報を収集し、オペレーターおよび管理者にこの情報の単純化したビューを示します。以下にリストするタスクを実行できます。

この情報は、以下のように処理できます。

- オペレーターに割り当てます。
- ヘルプ・デスク・システムに渡します。
- データベースにログインします。
- リモート Tivoli Netcool/OMNIbus システムで複製します。
- 特定のアラートへの自動応答のトリガーに使用します。

また、Tivoli Netcool/OMNIbus は、リモート・ロケーションのドメインが制限されたさまざまなネットワーク管理プラットフォームからの情報を集約できます。既存の管理システムおよびアプリケーションとともに使用すると、Tivoli Netcool/OMNIbus は、デプロイメント時間を最小化し、従業員が既存のネットワーク管理スキルを使用できるようにします。

Tivoli Netcool/OMNIbus は、メモリー内のハイパフォーマンス・データベースのアラート通知を追跡し、個別に構成可能なフィルターおよびビューを使用して、ユーザーに必要な情報を示します。Tivoli Netcool/OMNIbus 自動化機能は、管理アラートに対するインテリジェント処理を実行できます。

IBM Tivoli Netcool/OMNIbus 資料 Web サイトは、http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?toc=/com.ibm.netcool_OMNIbus.doc/toc.xml です。

Tivoli Enterprise Console の概要

IBMTivoli Enterprise Console (TEC) 製品は、ルール・ベースのイベント管理アプリケーションで、システム、ネットワーク、データベース、およびアプリケーションの管理を統合して、組織における IT サービスの最良の可用性を実現します。その機能について以下に示します。

Tivoli Enterprise Console® 製品の特徴は、以下のとおりです。

- コンピューター・エンタープライズに関する集中型のグローバル・ビューを提供します。
- 応答しないデータベース・サーバー、失われたネットワーク接続、正常に完了したバッチ処理ジョブなど、一般的な管理イベントを収集、処理し、このようなイベントに自動的に応答します。
- 他の Tivoli ソフトウェア・アプリケーション、Tivoli パートナー・アプリケーション、カスタム・アプリケーション、ネットワーク管理プラットフォーム、リレーショナル・データベース・システムなど、さまざまなソースからのアラームおよびイベントの中央コレクション・ポイントとして機能します。

Tivoli Enterprise Console 製品では、以下のようにして、IT 環境の高ボリューム・イベントを効果的に処理します。

- 重要性レベルによるイベントの優先順位付け
- 冗長イベントまたは低優先順位イベントのフィルター操作
- イベントの別のソースのイベントとの関連付け
- 特定のイベントを参照し、処理する担当者の決定
- エスカレーション、通知、トラブル・チケットのオープンなど、必要に応じた自動修正アクションの開始
- ホストの識別および事前定義イベント・グループの保守モードのホストからのイベントのグループ化

この製品およびそのコンポーネントの詳細は、「*IBM Tivoli Enterprise Console ユーザーズ・ガイド V3.9*」(SC88-9603-00) を参照してください。

Event Integration Facility (EIF) の概要

IBM Tivoli Event Integration Facility (EIF) は、イベント・コンソールのイベント配布および統合ポイントです。Tivoli Event Integration Facility ツールキットを使用すると、イベント・アダプターを作成して、IBM Tivoli Enterprise Console 環境に統合することができます。この機能の詳細については、以下に示す情報を通じて把握することができます。

IBM Tivoli Enterprise Console アダプターは、統合リンクです。アダプターは、イベントの収集、ローカル・フィルター操作の実行、関連イベントのイベント・コンソールの適切な形式への変換、イベントのイベント・サーバーへの転送を行います。システム、Tivoli ソフトウェア・アプリケーション、およびサード・パーティー・アプリケーションには、さまざまなアダプターがあります。既存のアダプターでサポートされていないソース (サード・パーティー・アプリケーションやカスタム・アプリケーションなど) をモニターするには、Tivoli Event Integration Facility を使用して、そのソースのアダプターを作成します。

Tivoli Event Integration Facility を使用して、以下を実行できます。

- イベント・サーバーに送信して処理するイベント情報を指定します。
- アダプターを作成して、イベント情報をフィルター操作および変換し、イベント・サーバーに転送します。
- 状態相関を使用して、ソース付近のイベントをフィルター操作し、関連付けます。
- イベントを受信できるアプリケーションを作成します。

詳細は、「*IBM Tivoli Event Integration Facility ユーザーズ・ガイド バージョン 3.8*」(GC88-8826-01) を参照してください。

スキーマ

EIF コネクターのスキーマについては、以下に示す情報を通じて把握することができます。

「**スキーマ・ファイル**」(eifSchemaFile) 構成パラメーターに Netcool ゲートウェイ・マッピング・ファイルが指定されている場合は、このファイルから EIF コネクター・スキーマが取得されます。詳細については、『構成』を参照してください。

イテレーター・モード

EIF コネクターは、イテレーター・モードの場合、提供された構造に適合する項目が AssemblyLine に供給されます。

属性名	値
className	ストリング
slotname	ストリング
slotname	ストリング
...	

slotname は、イベントの受信で指定されているスロットの名前です。

AddOnly モード

EIF コネクターは、AddOnly モードの場合、リモート・システムにイベントを送信します。送信されるイベントは、項目として指定されます。コネクターは、提供される項目が提供された構造に適合することを必要とします。

属性名	値
className	ストリング
slotname	ストリング
slotname	ストリング
...	

構成

EIF コネクターの構成には、以下に示すパラメーターが使用できます。

EIF 構成ファイル
このテキスト・フィールドには、構成ファイルのパスまたは基礎にあるライ

ブラリーで認識されるプロパティーのテキストのブロックが含まれます。プロパティーを指定するときは、このフィールドの先頭の文字に「!」を使用する必要があります。

スキーマ・ファイル
このオプションのストリング・パラメーターには、スキーマの取得に使用される Netcool EIF ゲートウェイのマッピング・ファイルのパスが含まれます。指定されたファイルが見つからない、開くことができない、または読み取ることができない場合は、エラー・メッセージが記録され、「msg」属性のみがスキーマに追加されます (以前の動作)。

エラー時に中断
このブール値パラメーターは、初期に接続を確立できない場合にコネクターがエラーをスローするかどうかを指定します。デフォルト値は *true* です。

GetNext タイムアウト
この数値パラメーターは、送信されるイベント・メッセージの待機時間 (秒単位) を指定します (-1: 永続的に待機、0: 待機せず次の使用可能なメッセージを取得、N: 待機する秒数)。デフォルト値は -1 です。

終了タイムアウト
この数値パラメーターは、リモート・サーバーとの接続をクローズするときの待機時間 (秒単位) を指定します。デフォルト値は 120 です。

詳細ログ
ログの詳細なメッセージが必要な場合は、このパラメーターをチェックします。

ファイル・コネクター

ファイル・コネクター (以前は「ファイル・システム・コネクター」と呼ばれていた) は、動作するためにパーサーを必要とするトランスポート・コネクターです。ここに記載されている情報を使用することで、それについて詳しく知ることができます。

ファイル・コネクターは、それを実行しているシステムで使用可能なファイルの読み取りと書き込みを行います。 1 つのファイルの同時使用は、ロック・メカニズムを使用して制御されます。

注: このコネクターは、イテレーター・モードまたは AddOnly モードでのみ使用できます。または、受動状態でこれらのモードに対応する操作を行うためにのみ使用できます。

構成

ファイル・コネクターの構成には、以下に示すパラメーターが使用できます。

ファイル・パス
読み取りまたは書き込みを行うファイルの名前。

タイムアウト (秒単位)
このパラメーターを正数で指定すると、コネクターはファイルからの読み取りの際に使用可能なデータを待ちます (つまり、コネクターはイテレーター・モードです)。待機時間を無制限に設定する場合は **0** (ゼロ) を指定し、

ファイルの終わりを通知するまでの待機時間を無制限以外に設定する場合は、その待機時間 (秒単位) を指定します。このパラメーターを **0** (ゼロ) に設定すると、コネクターは、UNIX スタイルの **tail -f** コマンドをシミュレートします。

ファイルに対するロックを要求した場合 (「**ファイルのロック**」パラメーターを使用)、この「タイムアウト」パラメーターは、ロックを獲得するまで待機する時間を指定します。このパラメーターが未指定の場合、または負の数値を指定した場合は、「永久に待機」します。

出力時に付加

このパラメーターを設定した場合、書き込みのためにファイルをオープンするときに、上書きではなく追加が行われます。

ファイルのロック

書き込み時には、書き込み先のファイルに対して排他ロックを獲得します。読み取り時には、共用ロックを獲得します。

ロックは、コネクターが初期化される時点で獲得され、コネクターがクローズされる時点で解放されます。

1 つのコネクター (A) がファイルに対して排他ロックを獲得した後、別のコネクター (B) が同じファイルをオープンしようとした場合、コネクター B はロックが解放されるのを待機するか、エラーをスローします。コネクター B が「ファイルのロック」パラメーターをチェックしている場合、待機します。「ファイルのロック」パラメーターをチェックしていなかった場合、エラーがスローされます。

ロック・メカニズムは、オペレーティング・システムに依存します。ファイルのロックはデッドロックの原因となることがあります。特に、AssemblyLine ごとに複数のファイルをロックする場合は、その可能性が高くなります。

詳しくは、http://docs.oracle.com/javase/6/docs/api/java/nio/channels/FileChannel.html#lock() を参照してください。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

パーサー

「パーサー」タブでは、「継承元:」ボタンでパーサーを選択し、ファイルの内容にアクセスするパーサーの名前を構成します。

関連情報

406 ページの『URL コネクター』.

ファイル管理コネクター

ファイル管理コネクターは、実行されているシステムで使用可能なファイル・システム構造およびファイル・システム・メタデータの読み取りおよび変更を行います。より具体的には、ファイルおよびディレクトリーを作成、検索、および削除できます。

このコネクターは、ユーザー定義のロケーションからディレクトリー構造内を反復処理して、ディスカバーしたファイル (ディレクトリー) を返すことができます。さらに、ファイル (およびディレクトリー) を検索、名前変更、および削除したり、それらをファイル・システム上の他のロケーションに移動またはコピーしたりすることができます。また、空のファイルおよびディレクトリーを作成するためにも使用できます。

このコネクターは、ファイルの現状の内容に対して操作を行いません。 119 ページの『ファイル・コネクター』 を使用して、これをコネクター駆動ループ・コンポーネントと結合することで、指定したフォルダーおよびそのサブフォルダーからすべてのファイル (または、IdMLs などの特定のタイプのサブセット) の内容を読み取る AssemblyLine を作成できます。

コネクターの使用

このセクションを通じて、ファイル管理コネクターのさまざまな使用法を理解することができます。

ディレクトリー構造のトラバース

ディレクトリー構造内を反復処理する際、あるいは特定のファイルまたはディレクトリーを検索する際に、コネクターは、指定されたツリーを再帰的にトラバースします。ここに記載されている情報を使用することで、それについて詳しく知ることができます。

イテレーター・モードでは、ディスカバーされるすべてのファイルまたはディレクトリーが返されます。一方、ルックアップ・モードでは、リンク基準に一致するファイルのみが返されます。複数の一致には、「**複数項目時**」フックを使用します。

開始ディレクトリーは、コネクターの構成タブにある「**ディレクトリー・パス**」パラメーターで設定します。

注: 「**ディレクトリー・パス**」は、UNC パスおよびマップされたネットワーク・ドライブを、Windows OS の有効な開始ディレクトリーとして受け入れます。
トラバースするファイルおよびディレクトリーのカウントを制限するために、反復の深さを指定できます。「**深さ**」パラメーターをブランクのままにした場合は、コネクターは、開始ディレクトリーからすべてのサブディレクトリーをトラバースします。値にゼロを指定した場合は、開始ディレクトリーのみを反復処理します。正の値は、コネクターがディレクトリー構造内をトラバースできる深さを示します。

例えば、「**深さ**」パラメーターに値 1 を指定した場合は、開始ディレクトリーとレベル 1 のサブディレクトリーのみが反復処理されます。

```
/startDirectory
  |
  |--- /dir1
  |   |
  |   |--- file1
  |   |--- file2
  |
  |--- /dir2
      |
      |--- /dir3
        |
```

```
    |--- file3
    |--- file4
^  ^  ^
  Lvl-0 Lvl-1 Lvl-2
```

この場合は、コネクターは、dir1、file1、file2、dir2、および dir3 を返しますが、file3 および file4 は深さが 2 であるため返されません。

また、「深さ」パラメーターを使用することで、シンボリック・リンクによる無限再帰を回避できます。例えば、親にリンクされた子サブディレクトリーがある場合に、この問題が生じる可能性があります。「**深さ**」パラメーターが指定されていない場合は、コネクターは、基盤オペレーティング・システムで制限されるまで、構造を無限に反復処理します。ディレクトリーの反復処理を固定の深さに設定することで、このような再帰が制限されることを保証できます。

「**フィルター**」パラメーターを使用して、返されるファイルおよびディレクトリーのセットを削減することができます。デフォルトでは、提供のフィルターは *グロブ式構文*を使用しますが、高度な使用法として、正規表現もサポートされます。この動作を切り替えるには、コネクターの構成で、「**正規表現の使用**」パラメーターを使用可能または使用不可にする必要があります。

グロブ・パターンはストリングとして指定され、他のストリング (ディレクトリーやファイルの名前など) に対して突き合わされます。グロブ構文は、以下に示す複数の単純なルールに従います。

- 単独で使用されるアスタリスク「*」は、任意の数 (ゼロ個も含む) の文字に一致します。
- 連続する 2 つのアスタリスク「**」は、「*」と同様に機能しますが、ディレクトリーの境界を横断します。この構文は、通常、完全なパスに一致させるために使用されます。
- 疑問符「?」は、任意の 1 文字のみに一致します。
- 中括弧は、サブパターンのコレクションを指定します。例を示します。
  - {sun,moon,stars} は、「sun」、「moon」、または「stars」に一致します。
  - {temp*,tmp*} は、「temp」または「tmp」で開始するすべてのストリングに一致します。
- 大括弧は、単一の文字の集合か、または文字の範囲 (ハイフン文字「-」を使用した場合) を表します。例えば、英字の場合、以下のようになります。
  - [a, e, i, o, u] は、任意の小文字の母音と一致します
  - [0-9] は、任意の数字と一致します。[A-Z] は、任意の大文字と一致します。
  - [a-z,A-Z] は、任意の大文字または小文字と一致します。

  大括弧内の「*」、「?」、および「¥」は、それ自身と一致します。
- その他のすべての文字はその文字に一致します。
- 「*」、「?」などの特殊文字に一致させる場合は、円記号「¥」を使用することで特殊文字をエスケープできます。例えば、「¥¥」は単一の円記号 (¥) に一致し、「¥?」は疑問符 (?) に一致します。

グロブ (ワイルドカード) 構文には次のような例があります。

- `*.html` – 開始ディレクトリー内のみの、.html で終了するすべてのストリングに一致します。
- `**.html` – 任意のサブディレクトリー内の、.html で終了するすべてのストリングに一致します。
- `new**.txt` – new で開始し、.txt で終了する任意の相対パスに一致します (例えば、New Folder¥output.txt)。
- `???` – ちょうど 3 つの文字または数字から成るすべてのストリングに一致します。
- `*[0-9]*` – 数値が含まれたすべてのストリングに一致します。
- `*.{htm,html,pdf}` – .htm、.html、または .pdf で終了する任意のストリングに一致します。
- `a?*.java` – a で開始し、少なくとも 1 つの文字または数字が続き、.java で終了する任意のストリングに一致します。
- `{foo*,*[0-9]*}` – foo で開始する任意のストリング、または数値を含む任意のストリングに一致します。

Java における正規表現の説明については、http://java.sun.com/developer/technicalArticles/releases/1.4regex/ を参照してください。

シンボリック・リンク

ここに記載されている情報を参照して、シンボリック・リンクを使用することができます。

コネクターはシンボリック・リンク (ハード・リンクではない) を検出し、見つかった場合は、入力マップの `isSymbolicLink` 属性を *true* に設定します。

さらに、ディレクトリーがシンボリック・リンクで、オプション「シンボリック・リンクのフォロー」が使用可能になっている場合、コネクターは、そのディレクトリーの内容を反復処理します。

注: コネクターは、Windows のシンボリック・リンクを検出しません。これは、Java 1.6 仮想マシンおよび Windows の制限事項です。

リンク基準での fullPath の指定

この属性を使用すると、検索に必要な時間を短縮することができます。ここに記載されている情報を参照して、リンク基準で fullPath を指定してください。

`fullPath` 属性は、すべてのファイルまたはディレクトリーの固有 ID です。この属性をリンク基準で指定し、「equal」一致条件を使用した場合、コネクターはディレクトリー・ツリーの検索をスキップし、リンク基準の残りと、このファイルまたはディレクトリーとの突き合わせを試みます。

例: コネクターの開始ディレクトリー・パスが `/user` に設定され、`/user/home/file.txt` に設定された属性 `fullPath` と *false* に設定された **isDirectory** でリンク基準が構成されている場合、これらの条件に一致可能なファイルは 1 つのみになります。そのため、コネクターはディレクトリー・ツリーを検索せず、指定されたファイルが基準の残りと一致するかどうかの検査のみを行います。一致した場合、このファイルの詳細が含まれた項目が返されます。

ファイルまたはディレクトリーの更新

ここに記載されている情報を使用することで、ファイルまたはディレクトリーを更新できるようになります。

更新モードでは、コネクターは、項目の 3 つの一般属性 (優先度が高いものから順に、fullPath、parent、および name など) を変更できます。ただし、一回の変更で変更できるのは、これらのうち 1 つのみです。複数の属性を指定した場合、コネクターは、最も優先度が高いものを対象にします。これらの属性とともに content を使用すると、更新対象のファイルに指定内容を書き込むようにコネクターに指示します。出力マップで (fullPath、parent、および name を指定せずに) content のみを指定した場合は、ファイルの既存内容が上書きされます。

注: コネクターに fullPath 属性と name 属性が指定されず、content 属性のみが指定されたときに、リンク基準に何も一致しなかった場合は、ファイルは作成されず、エラーが返されます。

name 属性のみを変更した場合は、コネクターは、移動操作ではなく、ローカルの名前変更を実行します。

以下に、考えられるファイルおよびディレクトリーの更新の明細を示します。

- ファイル (例えば、C:¥test¥a¥file.txt) を更新する必要がある場合は、以下のオプションがあります。
  - fullPath (例えば、C:¥b¥file2.txt) が指定されている場合は、ファイルは移動され、元の「C:¥test¥a¥file.txt」は更新されて「C:¥b¥file2.txt」になります。
  - そうではない場合は、parent (例えば、C:¥b) が指定されている場合は、ファイルは移動され、元の「C:¥test¥a¥file.txt」は更新されて「C:¥b¥file.txt」になります。
  - 最後に、name (例えば、file3.txt) のみが指定されている場合は、ファイルは名前変更され、元の「C:¥test¥a¥file.txt」は更新されて「C:¥test¥a¥file3.txt」になります。
- ディレクトリー (例えば、C:¥test¥a¥dir) を更新する必要がある場合は、以下のオプションがあります。
  - fullPath (例えば、C:¥b¥dir2) が指定されている場合は、内容は移動され、元の「C:¥test¥a¥dir」は更新されて「C:¥b¥dir2」になり、内容は元のものと同じになります。
  - そうでない場合は、parent (例えば、C:¥b) が指定されている場合は、ディレクトリー全体が移動され、元の「C:¥test¥a¥dir」は更新されて「C:¥b¥dir」になり、内容は元のものと同じになります。
  - 最後に、name (例えば、dir3) のみが指定されている場合は、ディレクトリーは単に名前変更され、元の「C:¥test¥a¥dir」は更新されて「C:¥test¥a¥dir3」になります。

注:

1. オプション「**オリジナルを保持**」を選択した場合は、移動または名前変更の代わりに、コピー操作が実行されます。
2. シンボリック・リンクであるか、シンボリック・リンクが含まれたディレクトリーをコピーまたは移動すると、各シンボリック・リンクは無効になります。

ファイルおよびディレクトリーの強制削除

削除モードでは、ファイル管理コネクターは、ディスカバーされたファイルまたはディレクトリーを削除します。ただし、読み取り専用ファイルまたは空ではないディレクトリーの場合、削除操作は失敗します。この場合、ファイルまたはディレクトリーを削除するには、次の 2 つのオプションがあります。

- 構成エディターで「**強制削除**」オプションを設定する。この場合は、デバッグ・モードが使用可能になっていれば、ログに明示的なメッセージが記録されます。
- コネクターの「**削除エラー**」フックに独自のロジックを追加する。この方法は、強制削除するファイルまたはディレクトリーを実行時に決定するために使用されます。

  以下に実装できるフックの例を示します。

```
var file = conn.file.getValue(0);
if (file.isFile() || file.listFiles().length > 5) {
   // 読み取り専用ファイルおよびサブエレメント数が 5 個未満のフォルダーを削除
   FileManagementConnector.connector.forceDelete();
}
```

注: 「強制削除」が使用可能になっている場合は、シンボリック・リンクまたはシンボリック・リンクが含まれるディレクトリーを削除すると、シンボリック・リンクだけではなく、リンクの参照先の実際のディレクトリーも削除されます。ディレクトリーの削除を強制すると、「**シンボリック・リンクのフォロー**」オプションは考慮されません。

空のファイルおよびディレクトリーの作成

AddOnly モードでは、コネクターは、空のファイルまたはディレクトリーを作成できます。また、新規ファイルまたはディレクトリーのパスに、欠落しているすべてのディレクトリーを作成します。ここに記載されている情報を通じて、空のファイルおよびディレクトリーの作成方法を理解することができます。

コネクターの出力マップに `isDirectory` 属性が指定されていない場合は、「**ファイルの作成**」チェック・ボックスを使用して、新規ファイルまたはディレクトリーを作成する必要があるかどうかを指定します。また、ファイルまたはディレクトリーの完全修飾名を指定する必要もあります。ファイルは属性 `fullPath`、`parent`、および `name` によって特徴付けられるため、以下のオプションが存在します。

- `fullPath` を指定した場合は、コネクターはそれを逐語的に使用して、ファイルまたはディレクトリーを作成します。他の 2 つの属性は無視されます。
- `parent` 属性と `name` 属性の両方を設定した場合は、これらの属性を絶対パスとして使用して、ファイルまたはディレクトリーが作成されます。
- `name` 属性のみを設定した場合は、コネクターは構成の「**ディレクトリー・パス**」パラメーターを使用して、新規ファイルまたはディレクトリーの完全修飾名を形成します。

`content` 属性を使用して、そのファイルの初期内容を指定できます。内容がストリング・オブジェクトの場合は、`charSet` 属性により、当該ストリングのシリアライゼーションに使用する文字セットを指定します。

AddOnly モードでサポートされる最後の属性は isReadOnly です。ファイル管理コネクターは、シンボリック・リンクおよび隠しファイル/ディレクトリーを作成できません。これは、そのような操作が、基盤のプラットフォームまたはファイル・システムに依存するためです。

スキーマ

ここに記載されている情報を通じて、スキーマとその属性について知ることができます。

入力スキーマ

file - java.io.File
この属性には、ファイル・システム上の実ファイルまたはディレクトリーを指す Java ファイル・オブジェクトが含まれます。

name - java.lang.String
この属性には、ファイルまたはディレクトリーのローカル名が含まれます。

parent - java.lang.String
この属性には、ファイルまたはディレクトリーの親パスが含まれます。

fullPath - java.lang.String
この属性には、ファイルまたはディレクトリーの正規パスが含まれます。この属性は、シンボリック・リンクの場合は、parent 属性と name 属性の組み合わせとは異なります。

isReadOnly - java.lang.Boolean
この属性は、ファイルまたはディレクトリーが読み取り専用かどうかを示します。

注: この属性は、基盤オペレーティング・システムがファイル許可を処理する方法に依存します。例えば、UNIX ベースのシステムの root ユーザーは、すべてのファイルおよびディレクトリーに対する絶対的な権限を備えています。そのため、root としてログインした場合は、このコネクターを実行する際、isReadOnly の値は常に false になります。

isHidden - java.lang.Boolean
この属性は、ファイルまたはディレクトリーが非表示であるかどうかを示します。

isDirectory - java.lang.Boolean
この属性は、入力項目がファイルであるかまたはディレクトリーであるかを示します。値 *NULL* は、基盤のファイル・システムに問題があることを示します。

isSymbolicLink - java.lang.Boolean
この属性は、ファイルまたはディレクトリーがシンボリック・リンクかどうかを示します (Windows では機能しません)。

lastModified - java.util.Date

length - java.lang.Long
この属性には、ファイルまたはディレクトリーの長さが含まれます。

出力スキーマ

fullPath - java.lang.String
この属性には、ファイルまたはディレクトリーの正規パスが含まれます。

parent - java.lang.String
この属性には、ファイルまたはディレクトリーの親パスが含まれます。

name - java.lang.String
この属性には、ファイルまたはディレクトリーのローカル名が含まれます。

content - java.lang.Object
この属性には、ファイルの新規初期内容が含まれます。バイト配列またはストリングを指定できます。

charSet - java.lang.String
この属性は、ストリング・オブジェクトである内容をシリアライズする際に使用する文字セットを示します。

isDirectory - java.lang.Boolean
この属性は、出力項目がファイルであるかまたはディレクトリーであるかを示します。この属性は、ファイルまたはディレクトリーの作成時には必須ですが、既存のファイルまたはディレクトリーの更新時には使用できません。

isReadOnly - java.lang.Boolean
この属性は、ファイルまたはディレクトリーが読み取り専用かどうかを示します。この属性はオプションです。また、既存のファイルまたはディレクトリーの変更時には適用外です。

構成

ここに記載されているパラメーターを使用することで、ファイル管理コネクターを構成できるようになります。

ファイル管理コネクターには、以下のパラメーターがあります。

ディレクトリー・パス
コネクターによって開始点として使用されるディレクトリー・パス。

ファイルだけを返す
ディレクトリー・ツリーの反復時にコネクターがファイルのみを返すように制限します。デフォルト値は *false* です。このパラメーターは、コネクターのイテレーター・モードに適用されます。

注: これにより、ファイル・コネクターが直接、入力項目を使用できるようになります。

深さ　反復するディレクトリー・ツリーの深さを設定します。値をブランクにすると (デフォルト)、すべてのサブディレクトリーが再帰的にトラバースされます。値が 0 の場合は、「**ディレクトリー・パス**」のターゲットであるディレクトリーの内容のみが対象になります。このパラメーターは、コネクターの AddOnly モードに適用されません。

フィルター
返されるファイル/ディレクトリーを制限するために使用するフィルター。「グロブ」と正規表現の両方がサポートされます。デフォルトでは、**グロブ (ワイルドカード)** 構文が想定されます。詳しくは、121 ページの『ディレ

クトリー構造のトラバース』セクションを参照してください。このパラメーターは、コネクターの AddOnly モードに適用されません。

正規表現の使用
正規表現の構文を使用するには、「フィルター」フィールドの動作を変更します。デフォルト値は *false* です。このパラメーターは、コネクターの AddOnly モードに適用されません。

シンボリック・リンクのフォロー
使用可能にすると、コネクターは、シンボリック・リンク・ディレクトリーの内容を返すことができるようになります。デフォルト値は *false* です。このパラメーターは、コネクターの AddOnly モードに適用されません。

強制削除
空ではないディレクトリーと読み取り専用ファイルを強制的に削除するように、コネクターを設定します。このパラメーターは、コネクターの削除モードに適用されます。

オリジナルを保持
見つかったファイルまたはディレクトリーに対して (移動または名前変更の代わりに) コピー・アクションを実行するようにコネクターを設定します。このパラメーターは、コネクターの更新モードに適用されます。

ファイルの作成
コネクターの出力マップに `isDirectory` 属性が指定されていない場合は、ファイルまたはディレクトリーをデフォルトで作成するかどうかを決定します。このパラメーターのデフォルト値は、*true* です。このパラメーターは、コネクターの AddOnly モードに適用されます。

コメント
このパラメーターは、ユーザーのコメントをすべて保持できます。このコンポーネントの操作中は考慮されません。

詳細ログ
詳細なログ・メッセージが必要な場合は、このパラメーターをチェックします。

例

ここに記載されているパスを使用することで、ファイル管理コネクターの例にアクセスできます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*`/examples/FileManagementConnector` ディレクトリーにあります。

フォーム入力コネクター

このコネクターは、コネクターのパラメーターとして指定された項目を AssemblyLine に供給します。ここに記載されている情報を参照して、フォーム入力コネクターについて詳しく知ることができます。

通常のコネクターと同様に動作しますが、別個の入力ファイルはありません。概念的には、このコネクターは、構成ファイルの一部として実際に格納されているテスト・ケースをテスト用 AssemblyLine に供給するときに便利です。また、

AssemblyLine の内部でデータを解析する必要があり、その結果として繰り返し処理の対象となる一連の項目が取得されるときにも、このコンポーネントが非常に便利です。この場合は、フォーム入力コネクターをコネクター・ループに付加してから、バイト・ストリームをロー・データ・テキスト・パラメーターにマップすることができます。

このコネクターは、イテレーター・モードのみをサポートします。

コネクターの使用

このコネクターを使用すると、コネクターにパラメーターとして提供されたロー・データを AssemblyLine に送ることができます。このコネクターは、構成されたパーサーを使用してロー・データを解析し、有効な項目を作成して基礎になる AssemblyLine に渡します。

構成

ここに記載されているパラメーターを使用することで、フォーム入力コネクターを構成できるようになります。

フォーム入力コネクターには、以下の 2 つのパラメーターがあります。

無限ループ

このパラメーターは、入力データのループ処理を可能にします。このパラメーターを有効にすると、コネクターは入力されたロー・データを無限に反復処理します。これは、AssemblyLine コンポーネントのストレス・テストを実行するときに便利です。

ロー・データ・テキスト

UTF-8 フォーマットで保存された入力項目です。このパラメーターは、`setParam()` メソッドを使用して実行時に設定できます。このパラメーターのデフォルト値は以下のとおりです。

```
first:John
last:Smith
.
id:2
first:Jane
last:Doe
```

このテキストは単純なパーサーを使用して簡単に解析することができ、結果の項目ダンプは次のようになります。

```
CTGDIS003I *** 項目のダンプを開始します
Operation: generic
Entry attributes:
 last (replace): 'Smith'
 first (replace): 'John'
 id (replace): '1'
CTGDIS004I *** 項目のダンプを完了しました

CTGDIS003I *** 項目のダンプを開始します
Operation: generic
Entry attributes:
 last (replace): 'Doe'
 first (replace): 'Jane'
 id (replace): '2'
CTGDIS004I *** 項目のダンプを完了しました
```

FTP クライアント・コネクター

FTP クライアント・コネクターはトランスポート・コネクターの 1 つであり、このコネクターが正しく機能するにはパーサーが必要です。このコネクターを使用すると、ファイルのリストまたはディレクトリーのリストのいずれかであるデータ・ストリームの読み取りまたは書き込みを行うことができます。FTP クライアント・コネクターは、ファイル転送に使用する手段というよりも、リモート読み取り/書き込み機能と考える方が妥当です。

このコネクターは、RFC959 に基づき FTP 受動モードをサポートしています。受動モードでは、ファイル転送でのデータ接続開始側が逆になります。通常、(クライアントからのコマンドの後に) サーバーがクライアントへのデータ接続を開始しますが、受動モードではクライアントがデータ接続を開始できます。これにより、クライアントがファイアウォールで保護されている場合のファイル転送が容易になります。

注:

1. イテレーター・モードでは、**get** および **list** 操作がサポートされます。AddOnly モードでは、**put** 操作がサポートされます。
2. このコネクターは、バイナリー・ファイルを転送するためのものではありません。

適切に構成した場合、このコネクターは FTP over SSL (FTPS) 接続をサポートし、セキュア転送を実現します。

SSL サポート

FTP クライアント・コネクターは FTPS をサポートし、セキュア転送を実行できます。この場合、FTP で使用される制御チャネルまたはデータ・チャネル (あるいはその両方) を暗号化するために、標準 FTP プロトコルの下位 SSL/TLS レイヤーが使用されます。 FTPS には、以下に示すような 2 つの一般的な使用方法があります。

- *暗黙的 FTPS* は広く実装されているスタイルであり、クライアントがデフォルトの 21 とは別の制御ポートに接続し、SSL ハンドシェークが実行されてから FTP コマンドが送信されます。 FTPS セッション全体が暗号化されます。暗黙的 FTPS ではネゴシエーションは許可されず、クライアントは FTPS サーバーの TLS/SSL ハンドシェークによるユーザー確認にただちに応答する必要があります。制御チャネルが暗号化されていない場合は、以降のデータ・チャネルも暗号化されていないことが必要であり (非 SSL)、制御チャネルが暗号化されている場合は、以降のデータ・チャネルはクリアのままでも暗号化されていてもかまいません。 Internet Assigned Numbers Authority (IANA) が公式にポート 990 を FTPS 制御チャネル・ポートとして指定し、ポート 989 を FTPS データ・チャネル・ポートとして指定しています。
- *明示的 FTPS* または *FTPES*。この方法では、クライアントはポート 21 で平文を使用して接続し、FTP セットアップ時またはそれ以降の任意の時点でセキュア TLS 接続をネゴシエーションできます。ネゴシエーションが失敗した場合、サーバーは暗号化されていない FTP を許可することがあります。暗号化されたデータ・チャネルと制御チャネルでの暗号化は、クライアントによっていつでも設定または破棄できます。

上述のとおり、FTP プロトコルは 2 つのチャネルを使用して動作します。制御 (コマンド) チャネルは FTP サーバーにコマンドを送信するために使用され、データ・チャネルはデータ転送に使用されます。細分性を高めるために、FTP クライアント・コネクターでは各チャネルに対して SSL サポートをオンにすることができます。

「**セキュリティー**」パラメーターを使用して、「**なし**」、「**制御チャネルで SSL を使用**」、「**制御チャネルおよびデータ・チャネルで SSL を使用**」の各オプションを指定できます。1 つ目のパラメーターは、SSL サポートが提供されず、セキュリティーの利点が期待できないことを示します。

「**制御チャネルで SSL を使用**」を選択した場合は、制御 (コマンド) チャネルで SSL が使用されます。この場合、FTP サーバーによって使用される証明書を IBM Security Directory Integrator のトラストストア (このトラストストアは `solution.properties` ファイル内の `javax.net.ssl.trustStore` プロパティーで設定されます) に追加する必要があります。このようにすることで、クライアントはサーバーを認証でき、正常に通信できます。また、このオプションを使用するときは、コネクターによって使用されるポートをサーバーが FTP/SSL 接続に使用するポート (デフォルトは 990) に変更することを忘れないでください。

SSL サポートを提供するもう 1 つのオプションが「**制御チャネルおよびデータ・チャネルで SSL を使用**」です。これを選択すると、クライアントは制御チャネルを保護する以外にも、セキュア・データ・チャネルをネゴシエーションしようとします。これは、「PBSZ 0」コマンドおよび「PROT P」コマンドをサーバーに送信することによって行われます。 PBSZ コマンドは、データ接続上で送受信されるアプリケーション・レベルのエンコード・データ用の最大バッファー・サイズを定義します。ただし、TLS/SSL はデータのブロック化を処理するため、「0」パラメーターが使用されます。もう 1 つのコマンド (PROT) では、FTP データ接続に使用される保護を定義し (ここで、「P」パラメーターは秘密 - TLS/SSL が使用されることを示します)、それによって完全性と機密性の保護が提供されます。

SSL オプション「*明示モード*」は、接続で SSL オプションのいずれか (例えば、制御での SSL の使用や制御およびデータでの SSL の使用) が選択されている場合にのみ有効です。このモードは動作を変更し、リモート FTP サーバーへの初期接続は SSL ソケットではありません。次に、接続の確立後に、制御チャネルで SSL がネゴシエーションされます。リモート FTP サーバーがネゴシエーションを拒否し、それにより、コネクターはセッションを打ち切ります。

「**セキュリティー**」パラメーターでは、FTP クライアント・コネクター用に許可されている一連のセキュリティー・オプションをリストします。ただし、コネクターがスクリプトを使用して作成されている場合は、もう 1 つ別のオプションがあります。コネクターのセキュリティー・パラメーターはコネクターが FTP サーバーに接続するときに引数として渡されるので (例えば、`connect(String host,String user, String password, boolean useSSLonCommandChannel, boolean useSSLonDataChannel)` のようになります)、SSL をデータ・チャネルで使用可能にし、制御チャネルでは使用可能にしない、ということが可能です。この構成は、クライアントがプレーン・テキスト・メッセージを送信してサーバーの SSL/TLS ポートに接続しなければならないことを示します。この接続の試みは確実に失敗し、そ

のため FTP クライアント・コネクターはこのケースの有無を確認し、AssemblyLine が開始されるとエラー・メッセージが表示されます。

上述のとおり、FTP クライアント・コネクターはアクティブ および受動 の 2 つモードで動作できます。受動モードでは、FTP サーバーは FTP クライアント・コネクターからの接続 (コマンド・チャネルおよびデータ・チャネルに対する) を待機します。接続が行われると、サーバーは証明書をクライアントに送信し、SSL 通信が可能になります。アクティブ・モードの場合、コマンド・チャネルについての状況は同じですが、クライアントが接続 (データ・チャネルに対する) を listen します。通常の場合、そのためにはクライアントが証明書を妥当性検査のためにサーバーに送信する必要があります。この問題に対処するために、SSL セッションはクライアント・モードで実行されます。つまり、SSL の役割が逆になります (TCP サーバーはクライアントとして動作し、TCP クライアントがサーバーとして動作します。この場合もサーバーは証明書をクライアントに送信します)。これは `setUseClientMode(true)` メソッドによって実現されます。

文字エンコード

FTP クライアント・コネクターは、読み取りと書き込みのために構成済みのパーサーを使用します。ここに記載されている情報を参照して、文字エンコードを実行することができます。

したがって、データの FTP サーバーからの読み取りおよび FTP サーバーへの書き込みは、このパーサーの「**文字エンコード**」パラメーターを使用して行われます。このパラメーターが指定されていない場合は、IBM Security Directory Integrator を実行しているプラットフォームのデフォルトの文字エンコードが使用されます。

構成

ここに記載されているパラメーターを使用することで、FTP クライアント・コネクターを構成できるようになります。

FTP ホスト名
コネクター接続先 FTP サーバーが存在する場所のホスト名または IP アドレス。

FTP ポート
FTP TCP ポート (デフォルトは **21**)。

ログイン・ユーザー
ログイン・ユーザー名。

ログイン・パスワード
ログイン・パスワード。

操作　意図された操作。ファイルの読み取り (イテレーター) の場合は **get**、ファイルの書き込み (AddOnly) の場合は **put**、ディレクトリーのリスト作成 (イテレーター) の場合は **list** を指定します。

リモート・パス
アクセスする初期リモート・ディレクトリー (list の場合) または初期リモート・ファイル (get/put の場合)。

転送モード

ASCII またはバイナリー。サポートされているモードは ASCII のみです。

受動モード

このチェック・ボックスをチェックすると、FTP クライアント・コネクターがアクティブ・モードではなく受動モードで FTP サーバーに接続することが指定されます。IPv6 では*常に* 受動モードが使用されるため、IPv6 接続ではこのパラメーターは無視されます。

セキュリティー

選択したオプションに応じて、FTP クライアント・コネクターは SSL セキュア接続を使用しないか、制御チャネルに対して使用するか、または制御チャネルとデータ・チャネルの一方または両方に使用します。使用可能な値は以下のいずれかです。

- なし
- SSL_control_channel - 制御チャネルで SSL を使用
- SSL_control_data_channels - 制御チャネルおよびデータ・チャネルで SSL を使用

明示モード SSL (FPES)

このチェック・ボックスを有効にすると、非 SSL ソケット経由で SSL セッションのネゴシエーションが実行されます。クリアすると (また、SSL が使用可能になっていると)、制御チャネルに対して SSL ソケットが暗黙的に作成されます。ftpSecurity パラメーターの値を「なし」にすると、このチェック・ボックスは使用不可になります。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

「パーサー」ペインから必須パーサーを選択します。例えば、リストを作成する場合や 1 つのファイルのみをコピーする場合は、Line Reader パーサーが便利です。選択ダイアログをアクティブにするには、左上の「**パーサーの選択**」ボタンをクリックします。

関連情報

681 ページの『FTP オブジェクト』,
406 ページの『URL コネクター』.

Generic Log Adapter コネクター

Generic Log Adapter コネクターは、ログ・ファイルを処理して、AssemblyLine に供給される Common Base Event (CBE) オブジェクトに変換します。ここに記載されている情報とリンクを使用することで、Generic Log Adapter コネクターについて詳しく知ることができます。

注: このコネクターは推奨されません。IBM Security Directory Integrator の将来のバージョンでは削除されます。

このコネクターは、IBM オートノミック・コンピューティング・ツールキットの一部である Generic Log アダプター (GLA) テクノロジーを使用してログ・ファイルを処理し、ログ・ファイルの内容を Common Base Event 形式に変換します。IBM Redbook の「A Practical Guide to the IBM Autonomic Computing Toolkit」には、Generic Log Adapter を構成および使用する方法に関する情報が記載されています。

アダプター構成ファイル

アダプター構成エディター Eclipse プラグインを使用して外部で準備されたアダプター構成ファイルは、Generic Log Adapter コネクターと組み合わせて使用されます。ここに記載されている情報を使用することで、それについて詳しく知ることができます。

この構成ファイルは、Generic Log Adapter コネクターが実行時に Common Base Event オブジェクトを作成するために使用するパーサー・ルールを作成するためのツールを提供します。つまり、この構成ファイルには、処理されるログ・ファイルについての情報が含まれています。このファイルからすべての CBE オブジェクトが後で作成されます。オブジェクト解析ロジックもアダプター構成ファイル内に実装されます。 Eclipse GLA プラグインにはこのような構成ファイルの例が複数あります。それらの構成ファイルは、いくつかのよく知られているアプリケーション・ログ・ファイルを処理するように作成されています。 Eclipse のユーザー・インターフェースを使用して独自の構成ファイルを作成することもできます。

作成済みの構成ファイルを使用する場合と新規構成ファイルを作成する場合のどちらの場合も、*TDIOutputter* という専用のアウトプッターを構成する必要があることに注意してください。これが必要なのは、CBE オブジェクトが作成されると、TDIOutputter は作成された CBE オブジェクトを Generic Log Adapter コネクターに送信するためです (その後、GLA コネクターは通常のマッピング・メカニズムを使用して CBE オブジェクトを AssemblyLine に送信できます)。

構成ファイルでの複数のアウトプッターの使用

ここに記載されている情報を通じて、構成ファイルで複数のアウトプッターを使用する方法を理解することができます。

複数の TDIOutputter を使用することはできません。アダプター構成ファイル内に複数の TDIOutputter が構成されている場合、Generic Log Adapter コネクターが TDIOutputter から相関 ID を取得しようとしたときに例外がスローされます。複数の TDIOutputter が構成されている場合、その構成ファイルを使用する Generic Log Adapter コネクターは、複数の TDIOutputter のうちどれを使用すればよいかを判別できず、正しい TDIOutputter から CBE オブジェクトを取得することができません。

ただし、TDIOutputter でない場合は複数のアウトプッターを定義することが可能です。例えば、TDIOutputter と FileOutputter を組み合わせることができます。この場合、すべての CBE オブジェクトが、ファイルと Generic Log Adapter コネクターの両方に送信 (および保存) されます。

構成

ここに記載されているパラメーターを使用することで、Generic Log Adapter コネクターを構成できるようになります。

Generic Log Adapter コネクターを構成するには、有効なアダプター構成ファイルが必要です。ファイルのパスがコネクターの「**構成ファイル・パス**」パラメーターに設定されている必要があります。構成ファイルの妥当性が検査され、有効なアダプター構成ファイルでない場合は例外がスローされます。

構成ファイル・パス
アダプター構成ファイルが配置されている場所を決定します。構成ファイルには、ログ・ファイルとその処理方法についての情報がすべて含まれています。

詳細ログ
このパラメーターをチェックすると、より多くの情報がログに記録されます。

TDIOutputter の構成

TDIOutputter を使用するようにアダプター・ファイルを構成するには、Eclipse の GLA ユーザー・インターフェース (Eclipse GLA プラグイン) を使用します。

Eclipse ユーザー・インターフェースを使用してアウトプッターを構成する方法は、以下に説明するとおりです。

1. アダプター構成ファイルを編集用に開きます。Eclipse プラグインによって構成ファイルの内容が表示されます。
2. 「アダプター」->「構成」->「コンテキスト・インスタンス」を選択します。
3. 「コンテキスト・インスタンス」を右クリックします。
4. 「追加」->「ロギング・エージェント・アウトプッター」を選択します。これでアウトプッターを表示して構成できるようになります。
5. アウトプッターのタイプとして非宣言を選択します。
6. 「説明」フィールドに選択した項目の説明を入力します。
7. 「アウトプッター (Outputter)」を右クリックし、「追加」->「プロパティー」を選択します。
8. プロパティーに「*tdi_correlation_id*」と名前を付けます。
9. このプロパティーの値として、任意の**固有**値を使用します。この値は、この構成ファイルを使用する Generic Log Adapter コネクターの相関 ID になります。
10. 値が入力されない場合、TDIOutputter はデフォルト値を使用して、アダプター構成ファイルの開始を試みたコネクターを登録します。
11. 「アダプター」->「コンテキスト」->「基本コンテキストの実装」を選択して右クリックします。
12. 「追加」->「ロギング・エージェント・アウトプッター」を選択します。
13. 名前と説明のフィールドに入力します。
14. 「実行可能クラス」フィールドに「com.ibm.di.connector.gla.TDIOutputter」と入力します。

15. 役割がアウトプッターに設定されていることを確認します。
16. 「役割のバージョン」フィールドに数字を追加します (例: 1.0.0)。
17. 固有の ID について「参照」をクリックし、ステップ 2 から 7 で作成したばかりのアウトプッターを選択します。
18. これでアウトプッターが構成できたので、このアダプター構成ファイルを Generic Log Adapter コネクターで使用できます。

コネクターの使用

Generic Log Adapter コネクターを構成するには、有効なアダプター構成ファイルが必要です。

ファイルのパスがコネクターの「**構成ファイル・パス**」パラメーターに設定されている必要があります。

Generic Log Adapter コネクターが開始すると、GLA インスタンスがコネクター内部の別スレッドで開始されます。アダプターを別のスレッドで開始すると、GLA によるログ・ファイル全体の処理が終了するよりも前に繰り返し処理を開始できます。コネクターは CBE オブジェクトを受け取るとオブジェクトをキューに格納し、その際 FIFO (先入れ先出し法) 方式で要素の順序を指定します。キューにエレメントがないときにコネクターがキューからエレメントを取得する必要がある場合、コネクターは NULL 値を戻すのではなく、エレメントが使用可能になるまで待機します (つまり、コネクターはブロックします)。一方、キューがいっぱいのときにエレメントをキューに追加する必要がある場合は、キューに使用可能なスペースができるまでブロックします。

データの終わりや GLA アダプター・エラーなどの条件は、特殊なメッセージによってキュー内で処理されるため、このコネクターは、IBM Security Directory Integrator コネクターにおいて想定される方法で動作することができます。

繰り返し処理の際、CBE オブジェクトがキューから 1 つずつ読み取られ、Generic Log Adapter コネクターに配信されます。CBE オブジェクト自体は、作業項目の「rawCBEObject」と呼ばれる属性に格納されます。作業項目には CBE 属性も設定されます。

複数のコネクター・インスタンスが同時に実行される状況を処理できるようにするために、正しい CBE オブジェクトを正しい Generic Log Adapter コネクターに送信するメカニズムが必要です。これは、TDIOutputter 構成で固有の相関 ID パラメーターを使用することで実現できます。 Generic Log Adapter コネクターは、アダプター構成ファイルを開始する前に、TDIOutputter 構成から相関 ID を取得します (コネクターは実際にアダプター構成ファイルを解析します)。次に、コネクターは内部 TDIOutputter テーブルにその ID を登録します。TDIOutputter は、生成された CBE オブジェクトを送信する準備ができると、そのオブジェクトの相関 ID を取得し、この ID を使用してテーブルに登録されている Generic Log Adapter コネクターを選択します。

スキーマ

ここに記載されている情報およびリンクを通じて、Generic Log Adapter コネクターのスキーマについて詳しく知ることができます。

TDIoutputter キュー・オブジェクトから読み取られた未処理のロー CBE オブジェクトは、次の属性で使用可能であり、*作業* 項目にマップすることができます。

属性名	説明
$rawCBE	この属性には、処理されたアプリケーション・ログ・ファイルの結果である単一の CBE オブジェクトが保持されます。 CBE オブジェクトの数は、アダプター構成ファイルでのパーサーの構成によって異なります。

残りの属性は、 425 ページの『CBE パーサー』の資料にある出力マップ・スキーマ定義に示された仕様に従います。

関連情報

DB2 ログ・ファイルの処理例 (*TDI_install_dir*/examples/glaconnector ディレクトリー)、

293 ページの『RAC コネクター』,

521 ページの『CBE 関数コンポーネント』,

GLA ユーザーズ・ガイド。

HTTP クライアント・コネクター

HTTP クライアント・コネクターは、URL コネクターより、HTTP セッションに対する高い制御能力を提供します。この HTTP コネクターを使用すると、事前定義済みの属性を使用して HTTP ヘッダーと本文を設定できます。また、ユーザーは、データを戻すサーバーへのどのような要求でも、属性として使用できます。

このコネクターでは、サーバーからの要求に基づく SSL プロトコルを使用したセキュア接続がサポートされています (URL に「https://」接頭部を使用してサーバーにアクセスする場合など)。サーバーがクライアント・サイド証明書を要求する場合は、これらの証明書を *トラブルシューティング* トラストストアに追加し、global.properties または solution.properties でこのトラストストアを構成する必要があります。これに関する詳細は、「*インストールと管理*」の『IBM Security Directory Integrator コンポーネントのクライアント SSL の構成』セクションを参照してください。

注: HTTP クライアント・コネクターは、拡張リンク基準をサポートしていません (「IBM Security Directory Integrator」の『拡張リンク基準』を参照)。

モード

HTTP クライアント・コネクターは、以下に示す 4 つの異なる AssemblyLine モードで使用できます。

イテレーター

このコネクターを呼び出すたびに、このコネクター用に構成されている同じ URL が要求されます。したがって、コネクターの構成にパーサーを組み込んでいない限り、コネクターは永続的に同じページを要求して実行を続けま

す。パーサーを組み込んである場合は、パーサーは接続から最後の項目が読み取られた時点で通知を送り、コネクターは最終的に AssemblyLine を終了させます。

ルックアップ
このモードでは、コネクターは、ルックアップ関数が呼び出されるたびに 1 つのページを要求します。検索基準には、要求するページまたは URL を指定したり、任意の数のパラメーターを組み込むことができます。これらのパラメーターは、すべて要求パラメーターとして基本 URL に追加されます。

AddOnly
このモードでは、コネクター要求はイテレーター・モードとほとんど同じように実行されます。

Call/Reply
このモードでは、コネクターには入力属性マップと出力属性マップの 2 つの属性マップがあります。AssemblyLine がコネクターを呼び出すと、出力マップ操作が行われ、続いて入力マップ操作が行われます。

ルックアップ・モード

ルックアップ・モードでは、以下に示すように、検索基準を設定することにより動的に要求 URL を変更することができます。

- 基準が 1 つしかなく、属性の名前が **url** である場合は、その基準の中で指定した値が要求 URL として使用されます。

  ```
  url equals $url
  ```

- 複数の基準があるか、または唯一の基準が **url** 以外のものである場合は、コネクター構成で要求 URL として指定されている url に、すべての属性名と値が付加されます。

  基本 URL: http://www.example_page_only.com/lookup.cgi

  検索基準:

  ```
  name equals john
  mail equals doe.com
  ```

  結果の URL: http://www.example_page_only.com/lookup.cgi?name=john &mail=doe.com

- ルックアップ関数はオペランドを無視します。したがって、**equals** ではなく **contains** を指定したとしても、コネクターは equals を使用したときと同じように URL を作成します。

特殊属性

HTTP クライアント・コネクターの使用中は、ここに示す特殊属性について注意が必要です。

このコネクターをイテレーター・モードまたはルックアップ・モードで使用するときは、以下に示す属性またはプロパティーのセットがコネクター (*conn*) 項目に戻されます。

http.responseCode
整数オブジェクトとしての HTTP 応答コード。

200 OK —-> 200

http.responseMsg
ストリング・オブジェクトとしての HTTP 応答メッセージ。

200 OK —-> OK

http.content-type
戻される http.body のコンテンツ・タイプ (存在する場合)。

http.content-encoding
戻される http.body のエンコード (存在する場合)。

http.content-length
http.body のバイト数。

http.body
このオブジェクトは、戻された本文のバイトを読み取るために使用できる、java.io.InputStream クラスのインスタンス/サブクラスです。

```
var body = conn.getObject ("http.body");
var ch;

while ( (ch = body.read()) != -1 ) {
 task.logmsg ("Next character: " + ch);
}
```

InputStream クラスおよび各メソッドについては、Javadocs を参照してください。

http.body.response
コネクターが AddOnly モードで動作している場合、サーバーの http.body パーツからの応答はこの属性で使用可能になり、アウトバウンド・コールに指定された http.body 属性は修正されません。アウトバウンド・コールで http.body 属性に値を指定しなかった場合、サーバーから戻った時点で、http.body 属性は http.body.response 属性と同じになります。

http.text-body
http.content-type が text/ というシーケンスで始まっている場合は、コネクターは本文がテキスト・データであると判断し、http.body ストリーム・オブジェクトをこの属性に読み込みます。

このコネクターを AddOnly モードで使用しているときは、コネクターはヘッダー名に **http.** が付いている属性をすべて送信します。したがって、要求のコンテンツ・タイプを設定するには、属性に **http.content-type** という名前を付け、通常どおりに値を指定します。特殊な属性の 1 つに **http.body** があり、これには、ストリングまたは任意の java.io.InputStream または java.io.Reader サブクラスを含めることができます。

どのモードの場合も、コネクターは必ず **http.responseCode** 属性および **http.responseMsg** 属性を設定します。AddOnly モードでは、コネクターに渡される **conn** オブジェクトはこれらの属性の取り込み先オブジェクトなので特別です。これらの属性にアクセスするには、コネクターの「**追加後**」フック内の値を取得する必要があります。

文字エンコード

HTTP クライアント・コネクターは、内部的に HTTP パーサーを使用して、指定された URL 宛てに作成されたソケットの入力および出力ストリームを解析します。文字エンコードについては、ここに記載されている情報を通じて詳しく知ることができます。

これに使用されるデフォルトの文字エンコードは ISO-8859-1 です。

HTTP クライアント・コネクターに対してパーサーが構成されている場合は、このパーサーが、指定された文字エンコードを使用して `http.body` 属性を書き込むために使用されます。パーサーが指定されていない場合には、プラットフォームのデフォルトの文字エンコードが使用されます。

`http.body` 属性の文字エンコードを明示的に指定するには、HTTP クライアント・コネクターの「**コンテンツ・タイプ**」パラメーターを使用します。詳しくは、『構成』を参照してください。

構成

ここに記載されているパラメーターを使用することで、HTTP クライアント・コネクターを構成できるようになります。

このコネクターには、以下のパラメーターがあります。

HTTP URL
要求する HTTP ページ。

注: `https://` アドレスを使用する場合は、証明書のインポートも必要になることがあります (サーバーで自己署名証明書を使用する場合など)。必要な場合は、「**証明書の取得**」をクリックし、指定された URL でサーバーにアクセスして、サーバーの証明書をトラストストアにインストールしてください。

要求メソッド
ページを要求するときに使用する HTTP メソッド。詳しくは、http://www.w3.org/Protocols/HTTP/Methods.html を参照してください。

ユーザー名
このパラメーターが設定されている場合は、HTTP 許可ヘッダーは、このパラメーターと**パスワード**・パラメーターを使用して設定されます。

パスワード
ユーザー名 を指定した場合に使用されます。

プロキシー
これを指定した場合は、URL の中で指定されているホストに直接接続する代わりに、プロキシー・サーバーに接続します。形式は *proxyhost*:*port* です (例えば、`proxy:8080`)。ここで、`proxy` は *proxyhost* の名前で、`8080` は使用する *port* 番号です。

プロキシー・サーバー・ユーザー名
使用するプロキシー・サーバーで認証が必要な場合は、プロキシー・サーバーに対して認証を行うためのユーザー名を指定します。

プロキシー・サーバーのパスワード
指定されたプロキシー・サーバー・ユーザー名のパスワードを指定します。

HTTP 本文のファイル
ファイルの絶対パス。ファイル内容は、HTTP 本文として HTTP メッセージにコピーされます。これは、パーサーの処理として可能な設定をすべて指定変更します。

コンテンツ・タイプ
これを設定した場合、送信されるファイル (「**HTTP 本文のファイル**」パラメーターで指定したもの)、または項目内に存在する他の *HTTP Body* 属性 (前述の HTTP 属性を参照) の *http.content-type* として使用されます。

応答 HTTP 本文のファイル
ファイルの絶対パス。応答 HTTP メッセージの本文は、ファイルにコピーされます。

タイムアウト
サーバーへの接続とサーバーからの応答の受信という各操作のタイムアウト (秒単位)。タイムアウトをゼロにすると、無期限タイムアウトと解釈されます。タイムアウト時間が切れると、`java.net.SocketTimeoutException` が発生します (詳しくは、`java.net.Socket` のオンライン資料を参照)。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

「パーサー」ペインからパーサーを選択します。パーサーを選択するには、左上の「**パーサーの選択**」ボタンをクリックします。パーサーを指定した場合は、データを送信するときに、**http.body** のコンテンツを生成するためにそのパーサーが使用されます。パーサーは、名前が **http.** で始まっていない属性を持つ項目を取得します。また、このパーサー (指定されている場合) は、データの受信時に、追加の解析を行うために **http.body** を取得します。ただし、メッセージ本文には別のメッセージは含まれないため、system:/Parsers/ibmdi.HTTP は指定しないでください。

例

属性マップの中で提供された割り当てを使用して、ファイルのコンテンツを HTTP サーバーに通知することができます。

```
// Attribute assignment for "http.body"
ret.value = new java.io.FileInputStream ("myfile.txt");

// Attribute assignment for "http.content-type"
ret.value = "text/plain";
```

コネクターによって、**http.content-length** 属性が計算されます。したがって、この属性を指定する必要はありません。

関連情報

`

406 ページの『URL コネクター』,
『HTTP サーバー・コネクター』,
449 ページの『HTTP パーサー』.

HTTP サーバー・コネクター

IBM Security Directory Integrator には、着信 HTTP 接続を listen し、HTTP サーバーのように動作する HTTP サーバー・コネクターがあります。ここに記載されている情報を使用することで、これについて詳しく知ることができます。

コネクターは要求を受け取るとその要求を解析し、解析した要求を処理のために AssemblyLine ワークフローに送信します。結果は HTTP クライアントに戻されます。デフォルトでは、戻される結果のコンテンツ・タイプは「text/html」です。

このコネクターは、サーバー・モードとイテレーター・モードをサポートします。サーバー・モードが推奨モードです。

- サーバー・モードでは、接続が承認されるとコネクターは AssemblyLine のクローンを作成し、要求をクローンに渡します。親プロセスは新しい着信接続の待機を再開します。
- イテレーター・モードでは、このようなクローン作成は行われず、要求は現行の AssemblyLine 自体で処理されます。ただし、要求が処理されると、接続 (したがって、AssemblyLine) は終了します。これは目的に沿わない場合があります。

パーサーが指定されている場合、コネクターは **post** 要求を処理し、指定されたパーサーを使用して内容を解析します。**get** 要求はパーサーを使用しません。パーサーが指定されていないときに **post** 要求を受信した場合は、**post** データの内容は属性 (**postdata**) として戻り項目に戻されます。

HTTP サーバー・コネクターは、パーサーが指定されていない場合は、ibmdi.HTTP を内部パーサーとして使用します。

コネクターは、次のようにして、URL 要求を解析し、項目を取り込みます。

http://localhost:8888/path?p1=v1&p2=v2

```
http.method : 'GET'
http.Host   : 'localhost:8888'
http.base   : '/path'
http.qs.p1  : 'v1'
http.qs.p2  : 'v2'
```

http://localhost:8888/?p1=v1&p2=v2

```
http.method : 'GET'
http.Host   : 'localhost:8888'
http.base   : '/'
http.qs.p1  : 'v1'
http.qs.p2  : 'v2'
```

POST 要求が使用される場合は、リクエスターはこの接続を介してデータも送信しているものと見なされます。コネクターは、「**パーサー**」パラメーターの値に応じて次の処理を実行します。

Parser present
HTTP 入力ストリームを使用して、パーサーをインスタンス化します。コネクターは getNext をパーサーの getEntry に委任し、パーサーが戻すものをそのまま戻します。

Parser not present
ポスト・データの内容を **http.body** というコネクター属性に入れます。

コネクターが AssemblyLine から getNext 要求を受信したときに、取り出すデータが何も残っていなければ、HTTP クライアントとのセッションはクローズされます。例えば、パーサーが NULL 値を戻した場合や、パーサーが存在しない場合に getNext が 2 回目に呼び出されるときです。

コネクターの構造とワークフロー

HTTP サーバー・コネクターは、HTTP クライアントから HTTP 要求を受信し、HTTP 応答を戻します。前述したように、デフォルトのコンテンツ・タイプ・ヘッダーは「text/html」に設定されます。この設定をオーバーライドするには、コネクターからクライアントに結果が戻される前に、項目属性 `http.content-type` に適切な値を設定します。

AssemblyLine は HTTP サーバー・コネクターを初期化した後に、コネクターの `getNextClient()` メソッドを呼び出します。このメソッドは、クライアント要求が着信するまでブロックします。要求を受信すると (しかもサーバー・モードが選択されている場合)、コネクターがコネクター自身の新規インスタンスを作成します。この新規作成インスタンスは AssemblyLine に渡され、この AssemblyLine によりコネクター・インスタンスの新規 AssemblyLine スレッドが作成されます。この設計機能では、それぞれのイベントを個別のスレッドで処理できるため、HTTP サーバー・コネクターは複数の HTTP イベントを並列処理できます。次に AssemblyLine は新規スレッドでこの新規コネクター・インスタンスに対する `getNextEntry()` メソッドを呼び出します。`getNextEntry()` 呼び出しから戻される各項目は、HTTP クライアントからの個別の HTTP 要求を表します。`getNextEntry()` から戻される項目ごとにコネクターの `replyEntry(Entry conn)` メソッドが呼び出され、対応する HTTP 応答をクライアントに送信します。

コネクター・クライアントの認証

「HTTP 基本認証 (BA)」パラメーターは、HTTP クライアントがネットワーク経由でこのコネクターにアクセスする際にクライアント認証を強制するかどうかを制御します。ここに示す方式による実装方法を理解することができます。

HTTP サーバー・コネクターに HTTP 基本認証をインプリメントするには、次の 2 とおりの方法があります。

1. **認証コネクターを使用する**

   これは、HTTP EventHandler (IBM Security Directory Integrator の現行バージョンには存在しない) との互換性を確保するためのメカニズムです。コネクター・パラメーターである「**認証コネクター**」は、リンク基準として指定された HTTP 基本認証データのユーザー名とパスワードとともに、ルックアップ・モードで使用される IBM Security Directory Integrator コネクターを指定します。

- ルックアップから項目が戻された場合は、認証が正常に完了していると見なされ、HTTP サーバー・コネクターはクライアントの要求の処理に進むことができます。
- ルックアップで項目が検出できない場合は、クライアントは認証されず、要求は処理されません。

2. **スクリプト認証**

   このメカニズムではある程度のコーディングが必要ですが、より高性能であり、ユーザーが独自のスクリプトを使用して認証をインプリメントできます。このメカニズムは、「**認証コネクター**」パラメーターが NULL または空の場合にのみ使用できます。

   コネクターは、公開コネクター・メソッドである `getUserName()` と `getPassword()` により、「GetNext 後」フックのユーザー名とパスワードの値をユーザーが使用できるようにします。認証メカニズムのインプリメントは、ユーザーが行います。認証コードは「GetNext 後」フックに格納する必要があります。認証が失敗した場合は、AssemblyLine フックからコネクターの `rejectClientAuthentication()` メソッドを呼び出す必要があります。以下に示すサンプル認証スクリプト・コードを検討してください。

```
var httpServerConn = thisConnector.connector;
var username = httpServerConn.getUserName();
var password = httpServerConn.getPassword();

//perform verification here
successful = true;

if (!successful) {
 httpServerConn.rejectClientAuthentication();
}
```

チャンク転送エンコード

ここに記載されている情報を通じて、チャンク転送エンコードの手順について理解することができます。

パラメーター「**チャンク転送エンコード**」を使用可能にすると、コネクターは HTTP BODY を一連のチャンクとして書き込みます。

チャンク・エンコードを使用する場合は、ユーザーが各チャンクに対してコネクターの `putEntry(entry)` メソッドを呼び出します。指定される項目の「http.body」属性の値が、HTTP チャンクとして送信されます。反復が終了すると、AssemblyLine によりコネクターの `replyEntry(entry)` メソッドが自動的に呼び出されます。このメソッドは、(「http.body」属性が指定されている場合に)最後のデータ・チャンクを書き込み、チャンク・シーケンスを終了します。

HTTP サーバー・コネクターに対してパーサーが指定されている場合、これはパーサーから戻されたストリームです。このストリームは、`putEntry(entry)` または `replyEntry(entry)` の呼び出しのたびに、HTTP チャンクとして送信されます。

構成

ここに記載されているパラメーターを使用することで、HTTP サーバー・コネクターを構成できるようになります。

TCP ポート
着信要求を listen する TCP ポート (デフォルト・ポートは **80**)。

接続バックログ
着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

コンテンツ・タイプ
アウトバウンド・データに使用するデフォルトの HTTP コンテンツ・タイプ。 この値をオーバーライドするには、作業オブジェクトの http.content-type 属性を指定します。デフォルトは text/html です。

プロパティーとしての TCP データ
このチェック・ボックスをチェックすると (デフォルト)、conn 項目オブジェクトの getProperty() メソッドを使用して TCP 接続プロパティーにアクセスできます。チェックしないと、TCP 接続プロパティーが項目属性として表示されます。

プロパティーとしての HTTP ヘッダー
このチェック・ボックスをチェックすると、conn 項目オブジェクトの getProperty() メソッドを使用してすべての HTTP ヘッダーにアクセスできます。チェックしないと、すべての HTTP ヘッダーが項目属性として表示されます。

HTTP 基本認証 (BA)
使用可能にした場合 (デフォルトでは使用不可)、クライアントに対して HTTP 基本認証が実施されます。

認証レルム
HTTP 基本認証を要求するときにクライアントに送信する認証レルム。デフォルトは「IBM Security Directory Integrator」です。

認証コネクター
このドロップダウン・リストでは、オーセンティケーター・コネクターが指定されます。指定するコネクターがコネクター・ライブラリー内に存在しており、ルックアップ・モードで使用するよう構成されている必要があります。

このパラメーターを指定すると、このサービスにアクセスしようとするが認証データを送信しないすべてのクライアント (例: Web ブラウザー) に対し、HTTP サーバー・コネクターが認証要求を発行します。クライアントがユーザー名/パスワードを送信すると、HTTP サーバー・コネクターは、認証コネクターのルックアップ・メソッドを呼び出し、ユーザー名とパスワードの属性を渡します。したがって、認証コネクターは、$username および $password 属性を指定したリンク基準を使用して構成する必要があります。一般的なリンク基準は、以下のようになります。

```
username equals $username
password equals $password
```

検索が失敗した場合、HTTP サーバー・コネクターは要求を拒否し、認証要求をクライアントに送り返します。検索が正常に完了した場合は、HTTP サーバー・コネクターは要求を処理します。

オーセンティケーター・コネクターにより戻された項目にアクセスするには、イベント項目の「auth.entry」プロパティーを使用します。

クライアント認証と、代替方法での認証コネクターの使用についての詳細は、 143 ページの『コネクター・クライアントの認証』を参照してください。

SSL の使用

使用可能にすると (デフォルトでは使用不可)、コネクターはクライアントに対して SSL を使用するように要求します。非 SSL 接続要求は失敗します。

SSL を使用すると、コネクターはデフォルトの IBM Security Directory Integrator サーバー SSL 設定 (証明書、鍵ストア、およびトラストストア) を使用します。

クライアント認証を必要とする

使用可能にした場合 (デフォルトでは使用不可)、コネクターは SSL 使用時にクライアント認証を実施します。つまり、コネクターはクライアントに対し、構成済み IBM Security Directory Integrator トラストストアと突き合わせることができるクライアント・サイド SSL 証明書を提供するよう求めます。このパラメーターは、直前のパラメーター (SSL の使用) が使用可能に設定されている場合にのみ有効です。

チャンク転送エンコード

これをチェックすると、メッセージの HTTP BODY が一連のチャンクとして転送されます。 144 ページの『チャンク転送エンコード』を参照してください。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

注: 「**パーサー**」構成ペインからパーサーを選択できます。「パーサー」ペインがアクティブなときに右下隅の「*継承元:*」ボタンをクリックします。

コネクター・スキーマ

ここに記載されているリストを参照することで、HTTP サーバー・コネクターがサポートする属性について知ることができます。

入力属性

- http.* - 任意の HTTP ヘッダー。
- http.Authorization - HTTP 許可タイプ。
- http.base - HTTP 基本パラメーター。
- http.body - HTTP 要求の BODY。
- http.content-length - http.body のバイト数。
- http.content-type - HTTP コンテンツのタイプ (text/plain、text/xml など)。
- http.method - HTTP メソッド・タイプ。有効値は GET/POST/PUT です。
- http.qs.* - 照会ストリング・パラメーター。
- http.remote-pass - リモート・ユーザーのパスワード。

- http.remote-user - リモート・ユーザーのユーザー名。
- auth.entry - オーセンティケーター・コネクターにより戻される項目。
- tcp.inputstream - ソケット入力ストリーム。
- tcp.outputstream - ソケット出力ストリーム。
- tcp.remoteIP - リモート IP アドレス。
- tcp.remotePort - リモート・ポート。
- tcp.remoteHost - リモート・ホスト名。
- tcp.localIP - ローカル IP アドレス。
- tcp.localPort - ローカル・ポート。
- tcp.localHost - ローカル・ホスト名。
- tcp.socket - ロー・ソケット・オブジェクト。

出力属性

- http.body - HTTP 応答の BODY。
- http.content-type - HTTP コンテンツのタイプ。
- http.redirect - 指定の場所へクライアントをリダイレクトする。
- http.status - 戻された操作の状況コード。

関連情報

406 ページの『URL コネクター』,
137 ページの『HTTP クライアント・コネクター』,
449 ページの『HTTP パーサー』.

IBM Security Access Manager コネクター

ここに記載されている情報を使用することで、IBM Security Access Manager コネクターについて知ることができます。

IBM Security Access Manager の IBM Security Directory Integrator コネクターを使用すると、IBM Security Access Manager のユーザー・アカウント、グループ、ポリシー、ドメイン、SSO リソース、SSO リソース・グループ、および (IBM Security Access Manager から見て) 外部のアプリケーションに対する SSO ユーザー資格情報のプロビジョニングと管理を行うことができます。このコネクターは、IBM Security Access Manager Java API を使用します。

このコネクターの主な機能と利点は以下のとおりです。

- IBM Security Access Manager のユーザー・アカウント、グループ、ポリシー、ドメイン、SSO リソース、SSO リソース・グループ、および SSO ユーザー信任状に関する作成、読み取り、更新、および削除のサポート。

注: このコネクターは IBM Security Access Manager 6 Java API を使用して、ターゲットとするオブジェクトの属性を操作します。そのため、このコネクターでは、v5.1 Runtime Environment (RTE) に関する JRE のサポートの制限により、IBM Security Access Manager 5.1 をサポートできません。IBM Security Access Manager バージョン 6.0 および 6.1 のみがこのコネクターでサポートされます。

IBM Security Access Manager サーバーとの SSL 通信がサポートされています。

注: IBM Security Access Manager v2 コネクターは、IBM Security Directory Integrator の V7.2.0.1 以降に付属するもう 1 つのコネクターです。このコネクターを使用すると、IBM Security Access Manager のユーザーおよびグループに対する IBM Security Access Manager Registry Direct API を使用したプロビジョニングと管理を行うことができます。詳細については、166 ページの『IBM Security Access Manager v2 コネクター』を参照してください。

コネクター・モード

このコネクターでは、ルックアップ、イテレーター、更新、AddOnly、および削除の各モードがサポートされています。

それぞれのモードの具体的な使用法については、152 ページの『コネクターの使用』を参照してください。

更新モードおよび削除モードでのルックアップのスキップ

IBM Security Access Manager コネクターでは、更新モードまたは削除モードでの「**ルックアップのスキップ**」一般オプションがサポートされます。ここに記載されているリンクを使用することで、これについて詳しく知ることができます。

このオプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。

有効なリンク基準が設定されている必要があります。つまり、155 ページの『更新モード』および 157 ページの『削除モード』の各セクションにある必須属性の表にそれぞれ定義されているように、必須の属性がコネクターのリンク基準に定義されている必要があります。

構成

AssemblyLine でこのコネクターを使用するには、IBM Security Access Manager バージョン 6.x をターゲット・マシンにインストールする必要があります。また、IBM Security Access Manager Java Runtime Environment (JRTE) も IBM Security Directory Integrator と同じマシン上にインストールされている必要があります。

IBM Security Access Manager の Java ランタイムの構成

このコネクターは IBM Security Access Manager Java API を使用するため、IBM Security Access Manager Runtime for Java が IBM Security Directory Integrator マシンにインストールされている必要があります。ここに記載されている情報を使用することで、構成について詳しく知ることができます。

IBM Security Access Manager Runtime for Java を IBM Security Directory Integrator マシンにインストールして構成する方法について詳しくは、「*IBM Security Access Manager インストール・ガイド*」を参照してください。

構成ユーティリティー (**pdjrtecfg**) にパラメーターを入力する場合は、以下のようにします。

- ポリシー・サーバーとレジストリー・サーバーの両方が実行中であることを確認します。
- IBM Security Directory Integrator が稼働していないことを確認します。
- IBM Security Directory Integrator の JVM がパスに含まれていることを、以下のようなコマンドを使用して確認します (UNIX/Linux の場合)。

  ```
  export PATH=/opt/IBM/TDI/V7.2/jvm/jre/bin:$PATH
  ```

- IBM Security Directory Integrator JRE ディレクトリーのロケーションを指定します。例えば、Linux マシンでのデフォルトの IBM Security Directory Integrator JVM ディレクトリーは以下のとおりです。

  ```
  /opt/IBM/TDI/V7.2/jvm/jre
  ```

- 構成タイプに完全を指定します。例えば、ディレクトリー Policy_Director/sbin から、以下のコマンドを (1 行で) 入力します。

  ```
  pdjrtecfg -action config -host ISAM_host_name -port 7135
    -java_home "/opt/IBM/TDI/V7.2/jvm/jre" -config_type full
  ```

  ここで、*ISAM_hostname* は IBM Security Access Manager Policy Server がインストールされているホストの名前です。「*Access Manager Runtime for Java の構成が進行中です (Configuration of Access Manager Runtime for Java is in progress)*」というメッセージが表示されます。これには数分かかる場合があります。完了すると、「*Access Manager Runtime for Java の構成が正常に完了しました (Configuration of Access Manager Runtime for Java completed successfully)*」というメッセージが表示されます。

IBM Security Access Manager ポリシー・サーバーへのセキュア通信の構成

IBM Security Directory Integrator マシン上で **SvrSslCfg** ユーティリティーを実行することによって、IBM Security Directory Integrator と IBM Security Access Manager のポリシー・サーバーと許可サーバーの間のセキュアな通信を構成し、IBM Security Directory Integrator が許可済みの IBM Security Access Manager Java アプリケーションになるようにすることができます。

例えば、ディレクトリー *TDI_install_dir*/jvm/jre/bin から、以下のコマンドを (1 行で) 入力します。このコマンドは、IBM Security Directory Integrator' の Java 実行可能プログラムで実行する必要があります。

```
/opt/IBM/TDI/V7.2/jvm/jre/bin/java com.tivoli.pd.jcfg.SvrSslCfg -action config
   -admin_id sec_master -admin_pwd password -appsvr_id appsvr -host ISAM_host_name
   -mode remote -port 999 -policysvr policy_svr:7135:1 -authzsvr auth_svr:7136:1
   -cfg_file cfg_file_name -key_file keyfile_name -cfg_action create
```

SvrSslCfg ユーティリティーについて詳しくは、「*IBM Security Access Manager Authorization Java Classes デベロッパーズ・リファレンス*」(特に、『*付録 A*』) を参照してください。

SSL の構成

ここに記載されている手順によって、オプションで新規の自己署名証明書を作成し、その証明書を使用するように IBM Security Directory Integrator を構成することができます。

1. IBM Security Directory Integrator 構成エディターを開きます。

2. ツールバーから「**KeyManager**」を選択します。 IBM Key Manager ツールが開きます。
3. 「**鍵データベース・ファイル**」を選択し、「**新規**」を選択します。
4. 鍵データベース・タイプとして「JKS」を選択します。
5. 該当する「**ファイル名**」、および該当する「**ロケーション**」を入力します。「**OK**」をクリックします。
6. 「**パスワード**」を入力します。確認のため、パスワードを再度入力します。「**OK**」をクリックします。
7. 鍵データベース内容のセクションで、「**個人証明書**」を選択します。「**新規自己署名**」をクリックします。

   注: 代わりに、既存の証明書を使用することもできます。既存の証明書を使用するには、「**エクスポート/インポート**」をクリックして該当する証明書をインポートします。
8. 該当する「**鍵ラベル**」、該当する「**組織**」、およびその他の必要な情報を入力します。「**OK**」をクリックします。
9. IBM Key Manager を閉じます。
10. IBM Security Directory Integrator 構成エディターで、「**サーバー・ストアのブラウズ**」を選択してから、構成するサーバー・ストア (通常は Default.tdiserver) の「 **オープン**」をクリックします。 「**ソリューション・プロパティー**」をダブルクリックすると、ソリューション・プロパティー・テーブルが開きます。
11. パラメーター `javax.net.ssl.trustStore` を見つけます。上記のステップ 5 で作成した鍵データベース・ファイルの値を入力します。
12. パラメーター `javax.net.ssl.trustStorePassword` を見つけます。上記のステップ 6 で入力したパスワードの値を入力します。
13. パラメーター `javax.net.ssl.trustStoreType` を見つけます。「jks」を入力します。
14. パラメーター `javax.net.ssl.keyStore` を見つけます。上記のステップ 5 で作成した鍵データベース・ファイルの値を入力します。
15. パラメーター `javax.net.ssl.keyStorePassword` を見つけます。上記のステップ 6 で入力したパスワードの値を入力します。
16. パラメーター `javax.net.ssl.keyStoreType` を見つけます。「jks」を入力します。
17. 「**クローズ**」をクリックして、ソリューション・プロパティー・テーブルを閉じます。ソリューション・プロパティーへの変更は、関連する `solution.properties` ファイルに保存されます。
18. IBM Security Directory Integrator 構成エディターを閉じます。

SSL の構成について詳しくは、「*IBM Security Directory Integrator 管理者ガイド*」を参照してください。

コネクターの構成

IBM Security Access Manager 用の IBM Security Directory Integrator コネクターは、Assembly Line に直接追加できます。このセクションでは、使用可能な構成パラメーターをリストします。

ISAM ID

コネクターは、このユーザー名と「パスワード」パラメーターに指定されているパスワードを使用して IBM Security Access Manager へのログオンを試行します。デフォルト値: *sec_master*

ISAM パスワード

このパラメーターの値が考慮されるのは、パラメーター「**ISAM ID**」に非ブランク値が設定されている場合のみです。ログオン操作に使用するパスワードを指定します。デフォルト値はブランクです。

ドメイン

IBM Security Access Manager のドメインを指定します。デフォルト値は「デフォルト」です。このパラメーターの隣にある「ドメイン」ボタンを押すと、IBM Security Access Manager サーバーが照会され、ドメインのリストが表示されます。このリストから適切なドメインを選択できます。コネクターは、パラメーター「**ISAM ID**」および「**ISAM パスワード**」を使用して IBM Security Access Manager へのログオンを試行します。

ISAM プログラム名

IBM Security Access Manager がこのコネクターを識別するために使用する名前。デフォルト値: *IDI*

ISAM 構成ファイル

SvrSslCfg 構成ユーティリティーによって作成される IBM Security Access Manager 構成ファイルのファイル・パス名。

項目タイプ

以下のいずれかの値に設定する必要があります。

- *User* (コネクターが IBM Security Access Manager のユーザーによって構造化されたデータを操作することを指定)
- *Group* (コネクターが IBM Security Access Manager のグループによって構造化されたデータを操作することを指定)
- *Policy* (コネクターが IBM Security Access Manager のユーザー・ポリシーによって構造化されたデータを操作することを指定)
- *Domain* (コネクターが IBM Security Access Manager のドメインによって構造化されたデータを操作することを指定)
- *SSOCred* (コネクターが IBM Security Access Manager の SSO 信任状によって構造化されたデータを操作することを指定)
- *SSOResource* (コネクターが IBM Security Access Manager の SSO リソースによって構造化されたデータを操作することを指定)
- *SSResourceGroup* (コネクターが IBM Security Access Manager の SSO リソース・グループによって構造化されたデータを操作することを指定)

ユーザー/グループのフィルター処理

「User」オブジェクト・タイプまたは「Group」オブジェクト・タイプの選択に使用されるフィルター・ストリングを定義する、オプションのコネクター属性。このパラメーターは、上記の 2 つの項目タイプのいずれかのイテレーター・モードにのみ使用されます。デフォルトでは、この属性は空であり、フィルタリングされないことを意味します。

レジストリーからのユーザー/グループのインポート

チェックすると、IBM Security Access Manager は**追加**操作時に、ユーザー・レジストリーにユーザーを作成するのではなく、ユーザー・レジストリーからユーザー/グループをインポートします。更新モードの場合、ユーザー/グループは**追加**操作によってのみインポートされ、**変更**操作ではインポートされません。

レジストリーからのユーザー/グループ/ドメインの削除

チェックすると、IBM Security Access Manager は削除操作時に、ユーザー・レジストリーからユーザー/グループ/ドメインを削除します。

UserName、RegistryUID、Firstname、および Lastname の各属性は、インポートする IBM Security Access Manager アカウントの正しい LDAP レジストリー・ユーザー名をこの操作で検出するために必須です。

更新モードの場合、ユーザー/グループは追加操作によってのみインポートされ、変更操作ではインポートされません。

詳細ログ

このフィールドをチェックすると、追加のデバッグ・ログ・メッセージが生成されます。

コネクターの使用

このセクションでは、サポートされている IBM Security Directory Integrator コネクターの各モードでのこのコネクターの使用方法について説明します。またこのセクションでは、コネクターによりサポートされる IBM Security Directory Integrator 項目スキーマについても説明します。

注: コネクターが Assembly Line で実行されると、IBM Security Access Manager コンテキストが、コネクターの*初期化* メソッドで作成されます。パフォーマンス上の理由から、IBM Security Access Manager コネクター・インスタンスごとにコンテキストが作成されないように、AssemblyLine 内で IBM Security Access Manager コネクターがキャッシュ (プール) される必要があります。 AssemblyLine 内のコネクターのキャッシュは、IBM Security Directory Integrator で構成できます。詳細については、「*IBM Security Directory Integrator ユーザーズ・ガイド*」を参照してください。

コネクターが IBM Security Access Manager のポリシー・オブジェクトを操作するように構成する場合は、**AddOnly** モードまたは**更新**モードのコネクターをフィードする作業項目の属性値を指定する際に、特別な考慮が必要です。ポリシー・オブジェクト属性は、関連するポリシー項目ごとにグループ化されています。各属性はセットに分割可能ですが、そのポリシー項目の個別の属性のいずれかを更新またはいずれかに適用する値は、属性のセットそれぞれに必要になります。例えば、ポリシー項目「アカウント有効期限日付」を操作する場合は、属性 AcctExpDateEnforced、AcctExpDateUnlimited、および AcctExpDate のそれぞれに値を指定する必要があります。その後、アカウント有効期限日付のこれらの属性のいずれかを変更する場合は、3 つの属性のそれぞれ、および UserName 属性に値を再度指定する必要があります。

以下の表に、ポリシー項目およびその属性グループの定義を示します。

表 14. ポリシー項目

ポリシー項目	必要なポリシー項目属性のセット
アカウント有効期限日付	AcctExpDateEnforced、AcctExpDateUnlimited、AcctExpDate
アカウント使用不可時間	AcctDisableTimeEnforced、AcctDisableTimeUnlimited、AcctDisableTime
アカウント・パスワード・スペース	PwdSpacesAllowedEnforced、PwdSpacesAllowed
アカウント最大パスワード使用日数	MaxPwdAgeEnforced、MaxPwdAge
アカウント最大反復文字数	MaxPwdRepCharsEnforced、MaxPwdRepChars
アカウント最小英字数	MinPwdAlphasEnforced、MinPwdAlphas
アカウント最小非英字数	MinPwdNonAlphasEnforced、MinPwdNonAlphas
アカウントのアクセス可能時間	TodAccessEnforced、AccessibleDays、AccessStartTime、AccessEndTime、AccessTimezone
アカウント最小パスワード長	MinPwdLenEnforced、MinPwdLen
アカウント最大ログイン失敗数	MaxFailedLoginsEnforced、MaxFailedLogins
アカウント最大同時 Web セッション数	MaxConcWebSessionsEnforced、MaxConcWebSessions、MaxConcWebSessionsUnlimited、MaxConcWebSessionsDisplaced

AddOnly モード

ここに記載されている情報とリンクを使用することで、AddOnly モードで処理できるようになります。

AddOnly モードでデプロイされた場合、コネクターは、IBM Security Directory Integrator のデータベース内にデータの範囲を作成することができます。このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**により、以下の属性のマッピングが定義される必要があります。これらの属性は、コネクター・スキーマを照会することによっても検索できます。

注:

1. アスタリスク (*) が付いた属性は、必須です。
2. すべての属性の詳細については、 160 ページの『コネクター入力属性の詳細』を参照してください。
3. ポリシー項目およびその必須ポリシー項目属性を取り扱う場合は、表 14 に規定されている警告に注意してください。

表 15. AddOnly モードの項目タイプ別属性

項目タイプ	属性
User	UserName*
	RegistryUID*
	FirstName*
	LastName*
	Description
	Password*
	IsAccountValid
	IsPasswordValid

表 15. AddOnly モードの項目タイプ別属性 (続き)

項目タイプ	属性
	IsSSOUser
	NoPasswordPolicyOnCreate
	MaxFailedLogins
	MaxConcWebSessions
	Groups (多値属性): ユーザーがグループのメンバーではない必要があります。
Group	GroupName*
	RegistryGID*
	CommonName
	Description
	ObjectContainer
	Users (多値属性): グループにユーザーが含まれていない必要があります。
Policy	UserName*
	AcctExpDateEnforced
	AcctExpDateUnlimited
	AcctExpDate
	AcctDisableTimeEnforced
	AcctDisableTimeUnlimited
	AcctDisableTimeInterval
	PwdSpacesAllowedEnforced
	PwdSpacesAllowed
	MaxPwdAgeEnforced
	MaxPwdAge
	MaxPwdRepCharsEnforced
	MaxPwdRepChars
	MinPwdAlphas
	MinPwdNonAlphasEnforced
	MinPwdNonAlphas
	TodAccessEnforced
	AccessibleDays
	AccessStartTime
	AccessEndTime
	AccessTimezone
	MinPwdLenEnforced
	MinPwdLen
	MaxFailedLoginsEnforced
	MaxFailedLogins
	MaxConcWebSessions
	MaxConcWebSessionsEnforced

表 15. *AddOnly* モードの項目タイプ別属性 (続き)

項目タイプ	属性
	MaxConcWebSessionsUnlimited
	MaxConcWebSessionsDisplaced
Domain	DomainName*
	Description
SSO Credentials	UserName*
	ResourceName*
	ResourceType*
	ResourceUser*
	ResourcePassword*
SSO Resource	SSOResourceName*
	Description
SSO Resource Group	SSOResourceGroupName*
	Description
	SSOResources (多値属性)

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

更新モード

ここに記載されている情報とリンクを使用することで、更新モードで処理できるようになります。

更新モードでデプロイされたコネクターは、IBM Security Access Manager データベースの既存のデータを変更できます。コネクターを IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**により、以下の属性のマッピングが定義される必要があります。また、以下の属性は、コネクター・スキーマに照会して取得することもできます。

更新時にユーザー/グループをインポートする場合:

- IBM Security Access Manager アカウントが IBM Security Access Manager に存在しない場合、このアカウントはレジストリーからのみインポートできます。
- **ReplaceUsersOnUpdate** フラグまたは **ReplaceroupsOnUpdate** フラグ (160 ページの『コネクター入力属性の詳細』を参照) が「true」に設定されている場合、ユーザーはグループのメンバーとして追加されますが、そのユーザーが既にメンバーである場合は例外がスローされます。
- インポート時の更新で変更されない属性は、**description** 属性のみです。レジストリーからインポートされた IBM Security Access Manager のその他の属性は、変更されます。

ポリシー項目およびその必須ポリシー項目属性を取り扱う場合は、 153 ページの表 14 に規定されている警告に注意してください。

アスタリスク (*) が付いた属性は、必須です。

表 16. 更新モードの項目タイプ別属性

項目タイプ	属性
User	UserName*
	Description
	パスワード
	IsAccountValid
	IsPasswordValid
	IsSSOUser
	MaxFailedLogins
	MaxConcWebSessions
	Groups (多値属性)
Group	GroupName*
	Description
	ReplaceUsersOnUpdate
	Users (多値属性)
Policy	UserName*
	AcctExpDateEnforced
	AcctExpDateUnlimited
	AcctExpDate
	AcctDisableTimeEnforced
	AcctDisableTimeUnlimited
	AcctDisableTimeInterval
	PwdSpacesAllowedEnforced
	PwdSpacesAllowed
	MaxPwdAgeEnforced
	MaxPwdAge
	MaxPwdRepCharsEnforced
	MaxPwdRepChars
	MinPwdAlphas
	MinPwdAlphasEnforced
	MinPwdNonAlphasEnforced
	MinPwdNonAlphas
	TodAccessEnforced
	AccessEndTime
	AccessibleDays
	AccessStartTime

表 16. 更新モードの項目タイプ別属性 (続き)

項目タイプ	属性
	AccessTimezone
	MinPwdLenEnforced
	MinPwdLen
	MaxFailedLoginsEnforced
	MaxFailedLogins
	MaxConcWebSessions
	MaxConcWebSessionsEnforced
	MaxConcWebSessionsUnlimited
	MaxConcWebSessionsDisplaced
Domain	DomainName*
	Description
SSO Credentials	UserName*
	ResourceName*
	ResourceType*
	ResourceUser
	ResourcePassword
SSO Resource	未サポート
SSO Resource Group	SSOResourceGroupName*
	SSOResources (多値属性)

また、コネクターの**リンク基準**で上記の必須フィールドを定義する必要があります。AssemblyLine はコネクターの `findEntry()` メソッドを呼び出して指定のユーザーの存在を確認するため、AssemblyLine にはリンク基準が必要です。リンク基準に定義されているこの属性の値は、**出力マップ**のエレメントの値に一致している必要があります。

リンク基準でサポートされている唯一の演算子は、**完全一致等号**です。ワイルドカード検索基準は、サポートされていません。このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

削除モード

ここに記載されている情報とリンクを使用することで、削除モードで処理できるようになります。

削除モードでデプロイされたコネクターは、IBM Security Access Manager データベースの既存のデータを削除できます。コネクターを AssemblyLine の **Flow** セクションに追加する必要があります。

アスタリスク (*) が付いた属性は、必須です。

表 17. 削除モードの項目タイプ別属性

項目タイプ	属性
User	UserName*
Group	GroupName*
Policy	UserName*
Domain	DomainName*
SSO Credentials	UserName*
	ResourceName*
	ResourceType*
SSO Resource	SSOResourceName*
SSO Resource Group	SSOResourceGroupName*

コネクターの**リンク基準**で必須属性を定義する必要があります。AssemblyLine はコネクターの `findEntry()` メソッドを呼び出して指定のユーザーの存在を確認するため、AssemblyLine にはリンク基準が必要です。

リンク基準でサポートされている唯一の演算子は、**完全一致等号**です。ワイルドカード検索基準は、サポートされていません。このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

ルックアップ・モード

ここに記載されている情報とリンクを使用することで、ルックアップ・モードで処理できるようになります。

ルックアップ・モードでデプロイされているコネクターは、必要な IBM Security Access Manager データの詳細をすべて取得できます。コネクターを AssemblyLine の **Flow** セクションに追加する必要があります。コネクターの**リンク基準**で必須属性を定義する必要があります。

アスタリスク (*) が付いた属性は、必須です。

表 18. ルックアップ・モードの項目タイプ別属性

項目タイプ	属性
User	UserName*
Group	GroupName*
Policy	UserName*

表 18. ルックアップ・モードの項目タイプ別属性 (続き)

項目タイプ	属性
Domain	DomainName*
SSO Credentials	UserName*
	ResourceName*
	ResourceType*
SSO Resource	SSOResourceName*
SSO Resource Group	SSOResourceGroupName*

コネクターの `findEntry()` メソッドは、主要な実行コードです。**リンク基準**でサポートされている唯一の演算子は、完全一致等号です。ワイルドカード検索基準は、サポートされていません。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターは一度に 1 つの項目のみを戻します。

イテレーター・モード

ここに記載されている情報とリンクを使用することで、イテレーター・モードで処理できるようになります。

イテレーター・モードでデプロイされたコネクターは、IBM Security Access Manager データベースの各データ入力項目の詳細を取得し、これらの詳細を AssemblyLine に対して使用可能にします。

このモードでデプロイされた IBM Security Directory Integrator は、最初にコネクターの `selectEntries()` メソッドを呼び出し、IBM Security Access Manager データベースの全データ入力項目のリストを取得し、キャッシュに入れます。項目タイプが **User** または **Group** で、フィルター属性が指定されている場合、このリストにはフィルター操作された項目が含まれます。Assembly Line は次にコネクターの `getNextEntry()` メソッドを呼び出します。このメソッドは、リスト内でキャッシュされている現行名を指し示すポインターを維持します。

フィルター属性のワイルドカードがサポートされているのは、**User** および **Group** 項目タイプのみです。

- アスタリスクを使用して、**UserName** ワイルドカード検索パターンを作成できます。 **UserName** パターンは、ユーザーの **UserName** 属性とゼロ文字以上一致する文字のストリングとして解釈されます。 **UserName** パターンの先頭、真ん中、または最後にアスタリスクを配置できます。また、**UserName** には、複数のアスタリスクを含めることができます。
- アスタリスクを使用して、**GroupName** ワイルドカード検索パターンを作成できます。 **GroupName** パターンは、グループの **GroupName** 属性とゼロ文字以上一致する文字のストリングとして解釈されます。 **GroupName** パターンの先頭、真ん中、または最後にアスタリスクを配置できます。また、**GroupName** には、複数のアスタリスクを含めることができます。

トラブルシューティング

リストされた理由のいずれかのために発生する可能性のある問題について、理解することができます。

IBM Security Access Manager コネクターが正しくインストールされていない
構成を確認し、必要に応じて再インストールを実行してください。

スキーマ照会の問題
IBM Security Directory Integrator GUI でコネクターを使用してスキーマ照会を実行すると、データ・ソースへの接続試行時に例外が発生することがあります。この例外は無視できます。その後、スキーマを**ディスカバー**するボタンを使用する操作は正常に実行されます。コネクターでは、*次の項目の取得* スタイルのスキーマ照会はサポートされていません。コネクターは、スキーマ・ディスカバリーの通常の IBM Security Directory Integrator スタイルをサポートします。

既に AssemblyLine に含まれているコネクターのモードの変更
テスト中に、AssemblyLine 内のコネクターのモード変更操作が成功しない場合があることが判明しました。コネクターがその本来のモードで稼働しているように見えていても、AssemblyLine エラーが発生することがあります。このような状況が発生した場合は、コネクターを削除し、新しいモードでコネクターを AssemblyLine に追加してください。

コネクター入力属性の詳細

このセクションでは、コネクター入力属性について説明します。

ユーザー

コネクター入力属性のリストを参照することができます。

表 19. コネクター入力属性

属性	説明	例	デフォルト
UserName	ユーザー名	maryl	
RegistryUID	LDAP ユーザーの Distinguished Name (DN)	cn=mary、o=companyabc、c=au	
FirstName	ユーザーの名前	Mary	
LastName	ユーザーの姓	Lou	
Description	説明	Contractor	
Password	ユーザーのパスワード (「NoPasswordPolicyOnCreate」属性が FALSE に設定されている場合、パスワードは IBM Security Access Manager の現行パスワード・ポリシーに従う必要があります。)	m3ry10u	
IsAccountValid	TRUE に設定すると、アカウントがアクティブになります。FALSE に設定すると、アカウントが非アクティブのままになります。	TRUE または FALSE	TRUE

表 19. コネクター入力属性 (続き)

属性	説明	例	デフォルト
IsPasswordValid	ユーザーが次のログインでパスワードを変更する場合は、FALSE に設定します。パスワードを変更しない場合は、TRUE に設定します。	TRUE または FALSE	TRUE
IsPDUser	IBM Security Access Manager の PD ユーザー・フラグ。	TRUE または FALSE	
IsSSOUser	TRUE を設定すると、このユーザーはシングル・サインオン (SSO) 機能を使用できます。FALSE を設定すると、この機能を使用できません。	TRUE または FALSE	FALSE
NoPasswordPolicy OnCreate	FALSE を設定すると、「Password」属性に対してパスワード・ポリシーが適用されるため、パスワードは、最初に作成されたときに、パスワード・ポリシー設定に従っているか確認されます。TRUE を設定すると、パスワードの作成時にパスワード・ポリシーは適用されません。	TRUE または FALSE	TRUE
MaxFailedLogins	アカウントが使用可能になる前にユーザーが失敗できるログインの最大数を設定します。	8	10
MaxConcWebSessions	許可される並行 Web セッションの最大数を設定します。	3	0
Groups (多値属性)	これは多値属性です。多値属性の設定方法については、「*IBM Security Directory Integrator ユーザーズ・ガイド*」を参照してください。この属性にリストされているグループは、すべて IBM Security Access Manager に有効なグループとして存在する必要があります。	Groups1 -> itSpecialists Groups2 -> programmers	

表 19. コネクター入力属性 (続き)

属性	説明	例	デフォルト
ReplaceGroupsOnUpdate	更新モードでは、この属性が TRUE に設定されている場合、ユーザーは、現在メンバーであるすべてのグループから削除されます。このユーザーは、Groups 属性の値として指定されている各グループのメンバーとして追加されます。 この属性が FALSE に設定されている場合、変更時に、現在ユーザーが含まれているグループが変更され、各 Groups 属性値の操作に基づいてユーザーが追加または削除されます。その結果、Groups 属性値の操作が AttributeValue.AV_ADD に設定されている場合、ユーザーがグループに追加されます。Group 属性値の操作が AttributeValue.AV_DELETE に設定されている場合、ユーザーがグループから削除されます。 追加モードでは、ReplaceGroupsOnUpdate フラグは無視されます。また、このフラグは、IBM Security Access Manager ユーザーとなるユーザーが見つからないときに更新が追加操作に戻る場合にも無視されます。	TRUE または FALSE	TRUE

グループ

ここに記載された Group 属性のリストを参照することができます。

表 20. Group 属性

属性	説明	例
GroupName	グループ名	programmers
RegistryGID	LDAP グループの DistinguishedName (DN)	cn=programmers、cn=SecurityGroups、secAuthority=Default
CommonName	LDAP 共通名 (CN)	programmers
Description	グループの説明	Fulltime Programmers
IsPDGroup	IBM Security Access Manager の PD グループ・フラグ。	TRUE または FALSE
ObjectContainer	IBM Security Access Manager のオブジェクト・コンテナー。	
ユーザー	これは多値属性です。多値属性の設定方法については、「*IBM Security Directory Integrator ユーザーズ・ガイド*」を参照してください。この属性にリストされているユーザーは、すべて IBM Security Access Manager に有効なユーザーとして存在する必要があります。	Users1 -> maryl Users2 -> johnd

表 20. Group 属性 (続き)

属性	説明	例
ReplaceUsersOnUpdate	更新モードでは、この属性には、グループのメンバーシップがどのように変更されたかを示すブール値フラグが指定されます。TRUE に設定されている場合は、グループのすべてのメンバーが削除され、ユーザーのリストが、Users 属性で削除されたユーザーを置き換える値として提供されます。 この属性が FALSE に設定されている場合、変更時に、グループのユーザーが User 属性値の操作に基づいて変更されます。その結果、User 属性値の操作が AttributeValue.AV_ADD に設定されている場合、ユーザーがメンバーとしてグループに追加されます。 User 属性値の操作が AttributeValue.AV_DELETE に設定されている場合、ユーザーがグループのメンバーシップから削除されます。 デフォルト値は TRUE です。 追加モードでは、ReplaceUsersOnUpdate フラグは無視されます。また、このフラグは、IBM Security Access Manager グループとなるグループが見つからないときに更新が追加操作に戻る場合にも無視されます。	TRUE または FALSE

ポリシー

ここに記載された Policy 属性のリストを参照することができます。

表 21. Policy 属性

属性	説明	例
UserName	ポリシーが設定されるユーザー名。有効な IBM Security Access Manager ユーザーである必要があります。	maryl
AcctExpDateEnforced	TRUE の場合、アカウント有効期限が適用されます。	TRUE または FALSE
AcctExpDateUnlimited	TRUE の場合は、アカウント有効期限が無期限に設定されます。	TRUE または FALSE

表 21. Policy 属性 (続き)

属性	説明	例
AcctExpDate	ユーザー・アカウントの有効期限日付を設定します。 この属性のタイプは、java.util.Date または java.lang.String である必要があります。ストリング値が指定されている場合、必要な日付ストリング・フォーマットは「yyyyMMdd」です。「yyyy」は 4 桁の年、「MM」は 2 桁の月、「dd」は 2 桁の日です。したがって、2009 年 12 月 31 日の値は 20091231 となります。	「IBM Security Access Manager Java API Reference」を参照してください。
AcctDisableTimeEnforced	TRUE の場合、アカウント使用不可時間が適用されます。	TRUE または FALSE
AcctDisableTimeUnlimited	TRUE の場合は、アカウント使用不可時間が無期限に設定されます。	TRUE または FALSE
AcctDisableTimeInterval	アカウント使用不可時間間隔を設定します。	「IBM Security Access Manager Java API Reference」を参照してください。
PwdSpacesAllowedEnforced	TRUE の場合、「PwdSpacesAllowed」属性の値が適用されます。	TRUE または FALSE
PwdSpacesAllowed	TRUE の場合、パスワードにスペースを使用できます。	TRUE または FALSE
MaxPwdAgeEnforced	TRUE の場合、パスワードの最大使用日数値が適用されます。	TRUE または FALSE
MaxPwdAge	パスワードの最大使用日数を設定します。	「IBM Security Access Manager Java API Reference」を参照してください。
MaxPwdRepCharsEnforced	TRUE の場合、パスワードの最大反復可能文字数が適用されます。	TRUE または FALSE
MaxPwdRepChars	パスワードの最大反復可能文字数を設定します。	5
MinPwdAlphasEnforced	TRUE の場合、許可される英数字の最小文字数が適用されます。	TRUE または FALSE
MinPwdAlphas	許可される英数字の最小文字数を設定します。	6
MinPwdNonAlphasEnforced	TRUE の場合、許可される非英数字の最小文字数が適用されます。	TRUE または FALSE
MinPwdNonAlphas	許可される非英数字の最小文字数を設定します。	3
TodAccessEnforced	TRUE の場合、ユーザーのアクセス時間設定が適用されます。	TRUE または FALSE
AccessibleDays	ユーザー・アカウントのアクセス可能日数を設定します。	「IBM Security Access Manager Java API Reference」を参照してください。
AccessStartTime	ユーザー・アカウントのアクセス開始時刻を設定します。	「IBM Security Access Manager Java API Reference」を参照してください。

表 21. Policy 属性 (続き)

属性	説明	例
AccessEndTime	ユーザー・アカウントのアクセス終了時刻を設定します。	「IBM Security Access Manager Java API Reference」を参照してください。
AccessTimezone	ユーザー・アカウントのタイム・ゾーンを設定します。	「IBM Security Access Manager Java API Reference」を参照してください。
MinPwdLenEnforced	TRUE の場合、パスワードの最小長が適用されます。	TRUE または FALSE
MinPwdLen	パスワードの最小長を設定します。	8
MaxFailedLoginsEnforced	TRUE の場合、最大ログイン失敗設定が適用されます。	TRUE または FALSE
MaxFailedLogins	ユーザーの最大ログイン失敗数を設定します。	8
MaxConcWebSessions	許可される並行 Web セッションの最大数を設定します。	3
MaxConcWebSessionsEnforced	TRUE の場合、並行 Web セッションの最大数設定が適用されます。	TRUE または FALSE
MaxConcWebSessionsUnlimited	TRUE の場合、並行 Web セッションの最大数ポリシーが「無制限」に設定されます。	TRUE または FALSE
MaxConcWebSessionsDdisplaced	TRUE の場合、並行 Web セッションの最大数ポリシーが「差し替え」に設定されます。	TRUE または FALSE

ドメイン

ここに記載された Domain 属性のリストを参照することができます。

表 22. Domain 属性

属性	説明	例
DomainName	ドメインの名前	MyDomain
Description	ドメインの説明	Sample domain name

SSO Credentials

ここに記載された SSO Credentials のリストを参照することができます。

表 23. SSO Credentials 属性

属性	説明	例
UserName	信任状が設定されるユーザーの名前	maryl
ResourceName	SSO リソース名 (有効な IBM Security Access Manager SSO リソース項目である必要があります)。	myResource1
ResourceType	このリソースが単一リソースかリソース・グループかを指定します。	指定できる値は、「Web Resource」および「Resource Group」のみです。
ResourceUser	リソース・ユーザー名を設定します。	marylou
ResourcePassword	指定したリソースのユーザー名パスワードを設定します。	b1ddy4

SSO Resource

ここに記載された SSO Resource のリストを参照することができます。

表 24. SSO Resource 属性

属性	説明	例
SSOResourceName	シングル・サインオン (SSO) リソース名	MyResource1
Description	説明	Development Server 1

SSO Resource Group

ここに記載された SSO Resource Group のリストを参照することができます。

表 25. SSO Resource Group 属性

属性	説明	例
SSOResourceGroupName	シングル・サインオン (SSO) リソース・グループ名	MyResourceGroup1
Description	説明	All Development Servers
SSOResources	これは多値属性です。多値属性の設定方法については、「*IBM Security Directory Integrator* ユーザーズ・ガイド」を参照してください。この属性にリストされている SSO リソースは、すべて IBM Security Access Manager に有効な SSO リソースとして存在する必要があります。	SSOResources1 -> myResource1 SSOResources2 -> myResource2

関連情報

e-business の Access Manager (Access Manager for e-business)

IBM Security Access Manager v2 コネクター

IBM Security Access Manager v2 コネクターでは、IBM Security Access Manager Registry Direct API を使用して IBM Security Access Manager のユーザーとグループのプロビジョニングと管理を行うことができます。

IBM Security Access Manager Registry Direct API は、許可サーバーやポリシー・サーバーを介さずに、下層の Security Access Manager レジストリーに直接アクセスします。さらに、これを使用すると、基礎になっているレジストリー・ユーザー属性のほとんどにアクセスでき、従来の IBM Security Access Manager Java API を通じて使用できる属性にもアクセスできます。この API は、属性の読み取り専用グローバル・サインオン (シングル・サインオン・リソース信任状) をサポートします。グローバル・サインオン対応になっているユーザーを、作成したり、使用可能または使用不可に設定したり、削除したりすることはありません。

IBM Security Access Manager v2 コネクターは、Registry Direct API を通じて提供されるアクセス方式を使用します。

この API の特徴を、以下に示します。

- ポリシー・サーバーへの依存 (Single Point of Failure) を取り除きます。

- より多くの属性にアクセスできるようにします。
- パフォーマンスとスケーラビリティーを向上させます。

IBM Security Access Manager v2 コネクターは、以下のモードでのユーザーとグループの管理をサポートします。

- イテレーター
- AddOnly
- 更新
- ルックアップ
- 削除

注: IBM Security Access Manager v2 コネクターでは、グローバル・サインオン・ユーザーの追加、変更、削除はいずれもサポートされません。これらのユーザーについては、イテレーター・モードまたはルックアップ・モードを使用して読み取りのみを行うことができます。グローバル・サインオン・ユーザーの追加、変更、削除を行うには、147 ページの『IBM Security Access Manager コネクター』を使用します。

Registry Direct API のデプロイ

IBM Security Access Manager v2 コネクターを構成する前に、IBM Security Access Manager Registry Direct API をデプロイして、そのプロパティーを構成する必要があります。

始める前に

- IBM Security Directory Integrator バージョン 7.2.0.1 (`ISAMConnector.jar` を含む) をインストールします。
- IBM Security Access Manager バージョン 6.1.1 以降をインストールし、ユーザー・レジストリーが統合するように構成します。
- スタンドアロン構成を許可するように構成された IBM Security Access Manager 認証 ID。

  注: スタンドアロン構成では、LDAP へのアクセスおよび管理更新の実行に使用される LDAP ID を手動で作成する必要があります。アクセス・マネージャーを使用して、LDAP ID を作成します。以下に例を示します。

  ```
  pdadmin sec_master> user create -no-password-policy
  testapi cn=testapi,o=ibm,c=us testapi api passw0rd
  ( SecurityGroup ivacld-servers remote-acl-users )
  ```

  詳しくは、IBM Security Access Manager Registry Direct Java API の資料を参照してください。

このタスクについて

`ISAMConnector.jar` ファイルは *tdi_install_dir*`/jars/connectors` ディレクトリー内にあります。

ここで、

tdi_install_dir は、IBM Security Directory Integrator のインストール・ディレクトリーです。*tdi_solution_dir* は、IBM Security Directory Integrator ソリューション・ディレクトリーです。これはインストール時に選択され、*tdi_install_dir*/bin/defaultSolDir スクリプトに記載されています。

com.tivoli.pd.rgy.jar ファイルには、IBM Security Access Manager の Registry Direct API ライブラリーと、API 構成ファイルを作成するツールが含まれています。

以下の手順では、IBM Security Access Manager サーバーが IBM Security Directory Integrator Server から見てリモート側にあることを前提とします。

手順

1. IBM Security Access Manager Registry Direct API JAR ファイルを、IBM Security Directory Integrator サーバーで使用できるようにします。以下のいずれかの方法を選択してください。
   - *ISAM_install_dir*/java/export/rgy/com.tivoli.pd.rgy.jar を *tdi_install_dir*/jars/3rdParty/IBM ディレクトリーにコピーする。
   - *ISAM_install_dir*/java/export/rgy/com.tivoli.pd.rgy.jar を、solution.properties 内の **com.ibm.di.loader.userjars** プロパティーで指定された IBM Security Directory Integrator ディレクトリーにコピーする。以下の設定はデフォルト値を示します。

     ```
     # com.ibm.di.loader.userjars=c:¥myjars
     ```

     この行のコメントを外し、参照されるディレクトリー名を作成する必要があります。
2. IBM Security Access Manager 構成ツール RgyConfig を使用して、構成ファイルを作成します。
   a. IBMJava ランタイム環境を使用するために *tdi_install_dir*/jvm/jre ディレクトリーに移動します。
   b. 次のコマンドを実行します。

      ```
      java –cp jar_file_path/com.tivoli.pd.rgy.jar
           com.tivoli.pd.rgy.util.RgyConfig properties_file_destination
           create Default Default "ldaphostname:389:readwrite:5"
           "DN" DN_password
      ```

      Security Access Manager Registry Direct API の構成オプションについて詳しくは、『Configuration options』を参照してください。

      例　前提:
      - 現行ディレクトリーは *tdi_install_dir*/jvm/jre です。
      - com.tivoli.pd.rgy.jar ファイルを *tdi_install_dir*/jars/3rdParty/IBM にコピーしました。
      - sec_master の有効な DN は、cn=SecurityMaster,secAuthority=Default です。

      ```
      java.exe -cp tdi_install_dir/jars/3rdParty/IBM/com.tivoli.pd.rgy.jar
           com.tivoli.pd.rgy.util.RgyConfig sam4sdi.properties
           create Default Default "ldapSamServer:389:readwrite:5"
           "cn=SecurityMaster,secAuthority=Default" secret
      ```

sam4sdi.properties ファイルがローカル・ディレクトリーに作成されます。

3. 新しく作成されたプロパティー・ファイルを *sdi_solution_dir*/LDAPSync ディレクトリーにコピーします。
4. IBM Security Directory Integrator を再始動します。

構成

IBM Security Access Manager v2 コネクターをインストールしたら、その構成を行うことができます。構成するには、そのコネクターを AssemblyLine に追加するか、ご使用の IBM Security Directory Integrator プロジェクトのリソースの「**コネクター**」 フォルダーに追加します。

削除モード、ルックアップ・モード、および更新モードにはリンク基準が必要です。リンク基準は IBM Security Access Manager v2 コネクターの「**リンク基準**」タブで作成できます。ユーザーには principalName を使用し、グループには cn を使用する必要があります。

パラメーター

IBM Security Access Manager v2 コネクターのパラメーターを使用して、IBM Security Access Manager からユーザーとグループを管理できます。

「IBM Security Access Manager v2 コネクター」パネルの「**接続**」タブで以下のパラメーターを使用して、 IBM Security Access Manager v2 コネクターを構成します。

ISAM ドメイン
統合する IBM Security Access Manager ドメインの名前。

構成ファイル
com.tivoli.pd.rgy.until.RgyConfig ツールを使用して作成される IBM Security Access Manager API 構成ファイルへのパス。

項目タイプ
項目のタイプ (ユーザーまたはグループ)。

名前検索フィルター
IBM Security Access Manager ユーザー・アカウントの **principalName** 属性、またはグループの場合は **cn** 属性に対する検索フィルターとして使用される値。

このパラメーターではワイルドカード文字のアスタリスク (*) がサポートされます。例えば、ab* を指定すると、先頭が ab であるすべての項目が返されます。このパラメーターはイテレーター・モードの場合にのみ使用できます。

属性マップ

IBM Security Access Manager v2 コネクターのスキーマは、選択するエンティティー・タイプによって異なります。

IBM Security Access Manager ユーザー・エンティティーの属性

ユーザー・エンティティー・タイプには、以下のスキーマ属性があります。属性の中には、ユーザーに関連付けられている LDAP の個人項目に書き込まれるものと、ユーザー・アカウント自体に固有のものがあります。

cn 関連付けられている LDAP 項目の cn (共通名) を指定します。この属性は必須です。

description
関連付けられている LDAP 個人項目について説明します。

memberOf
ユーザーが 1 つ以上のグループのメンバーであることを示します。

この属性には、1 つ以上のグループの **cn** 値、またはそれらの値を参照する **secDN** が含まれている必要があります。グループのメンバーシップを管理するための別の手段として、グループの **member** 属性を使用する方法があります。

principalName
ユーザーを一意的に識別します。この固有値は、このユーザーのログイン資格情報になります。この属性は必須です。

secAcctValid
IBM Security Access Manager のユーザー・アカウントが有効かどうかを示します。この値は、ストリングかブール値 (例えば、`true` や `'true'`) です。

secDN
このユーザーに関連付けられている LDAP 項目の DN (識別名) を指定します。この属性は必須です。

secPwdValid
ユーザーのパスワードが有効かどうかを示します。

基礎になっている IBM Security Directory Server でパススルー認証 (PTA) を実施できるようにするには、このブール値のフラグを `true` に設定してください。

注: パスワードを変更すると、**secPwdValid** の値は自動的に `true` にリセットされます。例えば、更新モードを使用して AssemblyLine で **userPassword** の値を設定すると、**secPwdValid** の値は `true` に設定されます。

sn 関連付けられている LDAP 項目の sn (姓) を指定します。この属性は必須です。

userPassword
ユーザーのパスワードを書き込みます。この値は平文で指定する必要があります。

ディレクトリー内に IBM Security Access Manager ユーザーと LDAP 個人項目の両方を作成する場合、この属性は必須です。この属性が必須なのは、作成される項目に対して API によってポリシー検査が適用されるためです。ただし、個人項目が既存のものである場合 (個人項目はコネクターによって追加されます)、そのユーザーは作成されるのではなくインポートされます。この場合、**userPassword** は必須でなくなります。例えば、Federated Directory Server plug-in for IBM Security Access Manager でこのコネクターが使用される場合、**userPassword** 属性をマップする必要はありません。詳細については、../di_fg/fdsisamplugin.ditaを参照してください。

IBM Security Access Manager グループ・エンティティーの属性

グループ・エンティティー・タイプには、以下のスキーマ属性があります。グループ自体に固有の属性は、**cn** と **secDN** のみです。その他の属性は、関連付けられている LDAP グループ項目に関するものです。

cn IBM Security Access Manager グループを識別します。これは関連付けられている LDAP グループ項目の cn (共通名) でもあります。この属性は必須です。

description
関連付けられている LDAP グループ項目について説明します。

member
LDAP の個人項目またはグループ項目、あるいはこの両方を参照する 1 つ以上の DN 値を含みます。グループのメンバーシップを管理するための別の手段として、個々のユーザーの **memberOf** 属性を使用する方法があります。

secDN
このグループに関連付けられている LDAP 項目の DN を指定します。この属性は必須です。

トラブルシューティング

IBM Security Access Manager v2 コネクターのトラブルシューティングを行うには、一般的なエラーに関する説明を参照してください。

構成 URL file:/X:/TDI/LDAPSync/ISAM_API.properties を読み取ることができません。(Unable to read in the configuration URL: file:/X:/TDI/LDAPSync/ISAM_API.properties.)
構成ファイルとしてラベル付けされた IBM Security Access Manager v2 コネクター・パラメーターには、IBM Security Access Manager API プロパティー・ファイルのパスとファイル名が含まれていなければなりません。この API プロパティー・ファイルは **com.tivoli.pd.rgy.util.RgyConfig** ツールを使用して生成されます。

IBM Security Access Manager ドメイン <*DomainName*> が存在しません。(The IBM Security Access Manager domain <*DomainName*> does not exist.)
IBM Security Access Manager v2 コネクターの「接続」タブまたは API プロパティー・ファイルに指定されているドメイン名が無効です。

この識別名はレジストリー内の既存の項目にマップされていません。(The distinguished name does not map to an existing entry in the registry.)
secDN 値が IBM Security Directory Server ディレクトリー・ツリーの既存のブランチにマップされていません。属性のマッピングが正しいことを確認してください。

指定された識別名 (secDN) は存在しません。(The specified distinguished name (secDN) does not exist.)
secDN 値が IBM Security Directory Server ディレクトリー・ツリーの既存のブランチにマップされていません。属性のマッピングが正しいことを確認してください。

無効なグループ ID または識別名が指定されました。(An invalid group identification or Distinguished Name (DN) was specified.)

グループ ID または DN 値が無効です。例えば、グループを作成しているときに使用される **cn** 属性値が無効です。属性のマッピングが正しいことを確認してください。

ID <*id*> の IBMSecurity Access Manager エンティティーがドメインに存在しません。(There is no IBM Security Access Manager entity in the domain with ID <*id*>.) グループを作成する際は、**member** 属性に既存の IBM Security Access Manager ユーザー・エンティティーとグループ・エンティティーの ID が含まれていなければなりません。その ID が含まれていないと、これらの値はスキップされ、このエラーがログに記録されます。

項目が見つかりませんでした。(Entry was not found.)

IBM Security Access Manager v2 コネクター用にセットアップされたリンク基準によって項目が見つかりませんでした。

グループが見つかりません。(Group not found.)

IBM Security Access Manager ユーザーを作成する際は、**memberOf** 属性に既存グループの ID が含まれていなければなりません。その ID が含まれていないと、これらの値はスキップされ、このエラーがログに記録されます。

AddOnly モードの出力マップに userPassword がない場合、コネクターは NULL ポインター例外を発行します (Connector gives null pointer exception when userPassword is missing in output map of the AddOnly mode)

ディレクトリー内に IBM Security Access Manager ユーザーと LDAP 個人項目の両方を作成する場合、userPassword 属性は必須です。この属性が必須なのは、作成される項目に対して API によってポリシー検査が適用されるためです。ただし、個人項目が既存のものである場合 (個人項目はコネクターによって追加されます)、そのユーザーは作成されるのではなくインポートされます。この場合、userPassword は必須でなくなります。例えば、IBM Security Access Manager の Federated Directory Server プラグインでこのコネクターが使用される場合、**userPassword** 属性をマップする必要はありません。

secPwdValid パスワードにマップされた値が false だった場合でも、このパスワードが true として作成されます。(The secPwdValid password is written as true even when the value mapped to it was false.)

IBM Security Access Manager ユーザーの **secValidPwd** 属性は、**userPassword** 属性が変更されると必ず true に設定されます。

詳しくは、以下のリンクを参照してください。

- フロー・フックを使用したフローのカスタマイズ (Federated Directory Server がどのようにこのコネクターを使用するのかについて説明します)。
- Federated Directory Server Plug-in for IBM Security Access Manager
- IBM Security Access Manager の資料
- IBM Security Access Manager Registry Direct Java API の資料
- IBM Security Access Manager コネクター

IBM Security Directory Integrator 変更ログ・コネクター

IBM Security Directory Integrator 変更ログ・コネクターの詳細については、以下に示す情報を通じて把握することができます。

IBM Security Directory Integrator 変更ログ・コネクターは、LDAP コネクターの特殊インスタンスです。IBM Security Directory Integrator 変更ログ・コネクターには、変更ログを繰り返すためのロジックが含まれています。戻されるさまざまな属性には、**変更**属性が含まれます。この属性は `java.lang.String` タイプであり、パーサー FC および LDIF パーサーを使用して読み取ることができる増分 LDIF を含みます。この手法では、変更された項目の「objectClass」に対する変更も取得できます (コネクターから戻される objectClass 属性は、変更ログ項目自体の objectClass 属性です)。

このコネクターはバッチ型実行で使用できます。その場合は、特定の変更番号で開始され、最後の**変更ログ**項目の後で終了します。また、定期的に次の**変更ログ**項目をチェックするためのタイマー値を指定して、連続モードで実行することもできます。

このコネクターは変更ログ項目を読み取り、1 回の繰り返しのたびに変更ログ・カウンターを自動的に 1 ずつ増加させます。存在しない変更ログ項目を読み取ろうとした場合は、このコネクターは一定期間 (**スリープ間隔**) スリープ状態に入ります。新規項目を待機する合計時間が「**タイムアウト**」値を超過すると、コネクターは、呼び出し元に **NULL** 値 (繰り返しの終了) を返します。

また、このコネクターは「**通知の使用**」オプションを公開します。このオプションは、コネクターがポーリングまたは通知メカニズムを使用して新規 IDS 変更を取得するかどうかを指定します。「*false*」に設定すると、コネクターは新規変更をポーリングします。このパラメーターを「*true*」に設定した場合、既存のすべての変更が処理されると、コネクターはその他の処理をブロックし、IBM Security Directory Integrator からの非送信請求イベント通知を待機します。通知メカニズムを使用する場合は、コネクターはスリープせずにタイムアウトします。

このコネクターでは、項目レベル、属性レベル、および属性値レベルでデルタ・タグがサポートされています。 LDIF パーサーが属性レベルと属性値レベルでデルタ・サポートを提供します。

コネクターにより、サーバーの変更ログの *modrdn* 操作が検出されます。詳しくは、245 ページの『modrdn 操作の検出と処理』を参照してください。

属性のマージ動作

属性のマージ動作については、ここに記載されている情報を通じて知ることができます。

旧バージョンの IBM Security Directory Integrator では、変更ログ項目の属性と実際のディレクトリー項目の変更された属性の間で、変更ログ・コネクターのマージが行われます。変更された属性を検出できないため、これにより問題が発生します。現行バージョンのコネクターには、このような状態を解決するためのロジックがあります。このロジックは、パラメーター「**マージ・モード**」によって設定されます。モードは以下のとおりです。

- **変更ログおよび変更データのマージ**: コネクターは、変更ログ項目の属性と実際のディレクトリー項目の変更された属性をマージします。これは旧来の実装であり、以前のバージョンとの互換性を確保します。
- **変更データのみを戻す**: 変更/追加された属性のみを戻し、変更ログ・イテレーターとデルタ・モードを容易にします。これはデフォルト設定です。旧バージョンの IBM Security Directory Integrator で開発された、または旧バージョンから移行した構成の場合、同じ動作を確保するために手動で「**変更ログおよび変更データのマージ**」を選択する必要が生じることがあります。
- **両方とも戻す**: 実際のディレクトリー項目の変更された属性を含む項目および変更ログ項目の属性を含む追加属性「changelog」を戻します。これにより、2 つのセットの属性を簡単に区別できます。

デルタ・タグはすべてのマージ・モードでサポートされているので、大量のスクリプトを使用しなくても、異なる LDAP サーバー間で項目を転送できます。

分散 TDS と z/OS TDS での変更ログの違い

分散 TDS と z/OS® TDS との間での変更ログの違いについては、以下に示すリストを参照してください。

注: IBM Security Directory Integrator バージョン 7.2 以降では z/OS オペレーティング・システムはサポートされません。

パスワード・ポリシーの運用属性に対する変更内容が `cn=changelog` に記録される方法は、z/OS 上の IBM Security Directory Integrator の場合と、その他のプラットフォーム上で稼働する分散 IBM Security Directory Integrator の場合では、いくつかの点で異なります。現在分かっている動作の違いは以下のとおりです。

1. userpassword の変更

   `userpassword` の変更操作は、`pwdfailuretime`、`pwdreset`、`pwdaccountlockedtime`、`pwdgraceusetime`、および `pwdexpirationswarned` などの属性をディレクトリー内の項目から削除します。また、`pwdchangedtime` も更新されます。

   分散 TDS は、これらの更新と変更ログ項目の `userpassword` 値の置換を LDIF で記録します。

   z/OS TDS は、生成された運用属性の削除を省略して、`userpassword` の置換のみを LDIF で記録します。

   また、パスワード変更により、項目の `pwdhistory` 属性も条件付きで更新できます。 この変更は z/OS TDS では記録されないことが分かっています。分散 TDS で実際に記録されることを示すテスト・データはありませんが、記録されることが予想されます。
2. 変更ログ LDIF のパスワード値

   z/OS TDS は、セキュリティー上の理由で実際の値を非表示にする代わりに、値を「`userpassword: *ComeAndGetIt*`」と表示します。

分散 TDS には、userpassword 値がそのまま表示されます。パスワード暗号化が使用されていないテスト出力しかないため、実際のパスワードが「平文」で表示されることに注意してください。パスワード暗号化がアクティブな場合、ほぼタグ付きの状態で、暗号化された値が表示されます。

3. ユーザー項目の追加

   ユーザーに対して有効なポリシーが、ユーザー項目を新規項目のケースであると示した場合パスワードを含むユーザー項目の追加操作により、true の値を持つ pwdreset 属性が条件付きで追加されます。

   分散 TDS は、追加用の変更ログ項目の LDIF に「PWDRESET: true」を含みますが、z/OS TDS にはありません。

4. 猶予ログインによる認証

   パスワードの期限が切れても「猶予」ログインが許可されている場合、認証 (バインドまたは比較操作のいずれか) が成功して、pwdgraceusetime 属性の追加値がユーザー項目に追加されます。分散 TDS は、この値を項目に追加される単一値として記録します。z/OS TDS は、古い値と新しい値をすべてリスト表示して、pwdgraceusetime 属性の値セット全体の置換としてこの値を記録します。

構成

IBM Security Directory Integrator 変更ログ・コネクターの構成には、以下に示すパラメーターを使用します。

LDAP URL
接続の LDAP URL (ldap://*host*:*port*)。

ログイン・ユーザー名
サーバーへの認証に使用する LDAP 識別名。匿名アクセスの場合はブランクにしておいてください。

ログイン・パスワード
信任状 (パスワード)。

認証メソッド
LDAP 認証のタイプ。これは、次のいずれかです。

- **無名** - この認証メソッドが設定されている場合、クライアントが接続されているサーバーでは、そのクライアントがどのクライアントであるかを認識しません (クライアント情報を必要としません)。サーバーは、このようなクライアントが、非認証ユーザー用に構成されたデータにアクセスすることを許可します。ユーザー名が指定されない場合、コネクターは、自動的にこの認証メソッドを指定します。ただし、このタイプの認証が選択され、**ログイン・ユーザー名**と**ログイン・パスワード**が指定された場合、コネクターは、自動的に認証メソッドを単純に設定します。
- **単純**: **ログイン・ユーザー名**と**ログイン・パスワード**を使用します。**ログイン・ユーザー名**と**ログイン・パスワード**を指定しない場合は、無名と見なされます。SSL プロトコルを使用して LDAP サーバーと通信するようにコネクターが構成されている場合を除いて、コネクターは、完全修飾された識別名 (DN) およびクライアント・パスワードを平文で送信することに注意してください。

- **CRAM-MD5** - これは SASL 認証メカニズムの 1 つです。LDAP サーバーは接続に際して、特定のデータを LDAP クライアント (すなわち、このコネクター) に送信します。すると、クライアントは MD5 暗号化方式を使用して暗号化された応答をパスワードとともに送信します。その後、LDAP サーバーはクライアントのパスワードをチェックします。CRAM-MD5 がサポートされるのは LDAP v3 サーバーのみです。IBM Security Directory Integrator のどのサポート対象バージョンでもサポートされません。
- **SASL** - クライアント (このコネクター) は、LDAP サーバーへの接続時に Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。このタイプの認証の稼働パラメーターは、「**追加のプロバイダー・パラメーター**」オプションを使用して指定する必要があります。例えば、DIGEST-MD5 認証をセットアップするには、以下のパラメーターを「追加のプロバイダー・パラメーター」フィールドに追加する必要があります。

  ```
  java.naming.security.authentication:DIGEST-MD5
  ```

  SASL 認証およびパラメーターについて詳しくは、http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html を参照してください。

  注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

SSL の使用
「SSL の使用」が **true** である (チェック・マークが付いている) 場合、コネクターは SSL を使用して LDAP サーバーに接続します。これに従ってポート番号の変更が必要になることがあります。

変更ログ・ベース
変更ログが保持される検索ベース。標準 DN は **cn=changelog** です。

追加のプロバイダー・パラメーター
複数の追加パラメーターを JNDI レイヤーに受け渡せるようにします。名前:値のペアとして 1 行に 1 つずつ指定します。

イテレーター状態キー
最後に行われた変更のときに処理が停止し、再開するように、現在の変更ログ番号を IBM Security Directory Integrator のユーザー・プロパティー・ストアに格納するパラメーターの名前を指定します。これは、IBM Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。この値を空のままにすると、AssemblyLine の再始動のたびに、コネクターは変更の開始時に処理を開始します。「**削除**」ボタンを押すと、この情報はユーザー・プロパティー・ストアから削除されます。

開始位置
開始変更番号を指定します (デフォルト値: 1)。各変更ログ項目に **changenumber=intvalue** という名前が付けられ、コネクターはこのパラメーターに指定された番号から開始し、自動的に 1 つずつ番号が増加します。

特殊値 **EOD** は、変更ログの終わりから開始することを意味します。このパラメーターは、イテレーターの状態が空または保存されていない場合にのみ使用されます。

「**照会**」ボタンを押すと、最初と最後の変更番号がサーバーから取得されます。

状態キーの維持

コネクターの状態をシステム・ストアに保管する方式を制御します。デフォルト (推奨設定) は「**サイクルの終わり**」です。以下のいずれかを選択できます。

読み取り後

AssemblyLine の残りの部分を続行する前、Directory Server の変更ログから項目を読み取るときにシステム・ストアを更新します。

サイクルの終わり

AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、変更ログ番号を使用してシステム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。この場合、AssemblyLine 内の特定の時点において、IBM Security Directory Server 変更ログ・コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保管する必要があります。

マージ・モード

変更ログ項目の属性と実際のディレクトリー項目の変更された属性のマージに使用されるメソッドを管理します。デフォルトは「**変更データのみを戻す**」であり、以下のいずれかを選択できます。

変更ログおよび変更データのマージ

コネクターにより、変更ログ項目の各属性が実際のディレクトリー項目の変更された属性とマージされます。このオプションでは、古いバージョンの IBM Security Directory Integrator の動作を選択し、以前のバージョンとの互換性を維持します。

変更データのみを戻す

変更または追加された属性のみが戻されます。

両方とも戻す

実際のディレクトリー項目の変更された属性を返すと共に、変更ログ属性を持つ項目が含まれている「変更ログ」と呼ばれる追加属性も返します。

通知の使用

IBM Security Directory Server での新規変更の待機時に通知を使用するかどうかを指定します。有効にすると、コネクターはスリープもタイムアウトもせず (さらに対応するパラメーターは無視され)、代わりに IBM Security Directory Server からの通知イベントを待ちます。

バッチ検索

IDS 変更ログでの検索の実行方法を指定します。チェックされていない場合は、コネクターは増分ルックアップを実行します (後方互換モード)。これが

チェックされており、サーバーで「ソート制御」がサポートされている場合は、「changenumber>=>=*some_value*」という照会を使用して検索が実行されます。これは、最後に実行した検索に対応しています。このパラメーターは、次に説明する「**ページ・サイズ**」パラメーターとの組み合わせで使用することで機能します。デフォルトでは、このオプションはチェックされていません。

ページ・サイズ
IDS が項目を戻すページのサイズを指定します (デフォルト値は 500)。このパラメーターは、「**バッチ検索**」が *true* に設定されている (チェック・マークが付けられている) 場合にのみ使用されます。

タイムアウト
コネクターが次の変更ログ項目を待つ秒数を指定します。デフォルトは 0 です。このデフォルト値では、無期限で待機します。

スリープ間隔
コネクターが各ポーリング間でスリープする秒数を指定します。デフォルトは 60 です。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

注: タイムアウトまたはスリープ間隔の値を変更すると、そのピアは自動的に有効な値に調整されます (例えば、タイムアウトをスリープ間隔よりも大きくした場合、未編集の値は変更された値にあわせて調整されます)。調整はフィールド・エディターがフォーカスを失うときに実行されます。

関連情報

Change log management for a directory server instance、
243 ページの『LDAP コネクター』,
9 ページの『Active Directory 変更検出コネクター』,
341 ページの『Sun Directory 変更検出コネクター』,
418 ページの『z/OS LDAP 変更ログ・コネクター』.

ITIM Agent コネクター

ITIM Agent コネクターは、IBM Security Identity Manager の JNDI ドライバーを使用して ITIM Agent に接続します (JNDI ドライバーは DAML プロトコルを使用します)。したがって、ITIM Agent コネクターは、DAML プロトコルをサポートするすべての ITIM Agent に接続できます。ここに記載されている情報を通じて ITIM Agent コネクターについて詳しく知ることができます。

このコネクター自体は、接続先の ITIM Agent の特定のスキーマを認識していません。このコネクターは、JNDI 項目の作成、読み取り、更新、および削除という基本的な機能を提供します。

ITIM Agent コネクターは、イテレーター、ルックアップ、AddOnly、更新、および削除の各モードをサポートしています。

このコネクターは、ITIM 5.1 リリースのクライアント・ライブラリー enroleagent.jar を使用します。

ITIM Agent コネクターのための SSL のセットアップ

ここに記載されている情報と例を参照して、ITIM Agent コネクター用に SSL をセットアップすることができます。

enroleagent.jar クライアント・ライブラリーでは SSL 認証に JSSE (Java ベースの鍵ストア/トラストストア) が使用されるため、SSL 関連証明書の詳細を global.properties/solution.properties に指定する必要があります。以前のバージョンの ITIM Agent コネクターでは、「CA 証明書ファイル (CA Certificate File)」パラメーターに証明書名を指定する必要がありました。最初に ITIM Agent の証明書を IBM Security Directory Integrator トラストストアにインポートする必要があります。

例えば、以下のコマンドでは servercertificate.der ファイルが tim.jks にインポートされます。

```
keytool –import –file servercertificate.der –keystore tim.jks
```

証明書のインポート後に、global.properties/solution.properties ファイルの「server authentication」セクションにこのトラストストアを指定する必要があります。

```
## server authentication

javax.net.ssl.trustStore=E:¥IBMDirectoryIntegrator¥tim.jks
{protect}-javax.net.ssl.trustStorePassword=<jks_keystore_password>
javax.net.ssl.trustStoreType=jks
```

注: 現行バージョンでは global.properties または solution.properties の JKS トラストストアに指定されている証明書が使用されるため、ITIM Agent コネクターの「CA 証明書ファイル (CA Certificate File)」プロパティーは存在しません。

構成

ここに記載されているパラメーターを使用することで、ITIM Agent コネクターを構成できます。

Agent URL
ITIM Agent に接続するために使用される URL。「https://<agent_ip_address>:<port>」という形式です (例えば、https://localhost:45580)。

ユーザー名
ITIM Agent の構成で指定されたユーザー名。ITIM Agent に対する認証のためにコネクターが使用します。

パスワード
ITIM Agent の構成で指定されたパスワード。ITIM Agent に対する認証のためにコネクターが使用します。

接続の再試行数
失敗した接続について再試行する回数を指定します (最初の接続の試行も含む)。値を指定しない場合、ITIM JNDI ドライバーはデフォルト値の 3 を使用します。

検索フィルター

イテレーター・モードで使用するフィルターの式。値を指定しない場合、デフォルトのフィルター「(objectclass=*)」が使用され、すべての項目が戻されます。

詳細ログ

このパラメーターをチェックすると、追加のログ・メッセージが生成されます。

既知の問題

このセクションでは、ITIM Agent コネクターでの作業中に発生する可能性のある既知の問題について参照できます。

このコネクターのテストは、少数の ITIM Agent について短期間で実施されました。ルックアップに関していくつかの問題が見つかっています。基礎にある Agent のインプリメンテーションの制約が原因です。

場合により、単純な JNDI 検索から戻される結果が期待どおりでないことがあります。例えば、Windows 2000 Agent を使用している場合に、Guest ユーザー・アカウントを見つける JNDI 検索「(eruid=Guest)」が複数の項目を戻すことがあります。あるいは、Red Hat Linux Agent を使用している場合に、「root」グループを見つける検索「(erLinuxGroupName=root)」が空の結果セットを戻すことがあります。

このようなケースに対する次善策は、オブジェクト・クラスを指定した拡張検索フィルターを使用することです (「(&(eruid=)(objectclass=<classname>))」)。 Windows 2000 Agent の場合は「&(eruid=Guest)(objectclass=erW2KAccount))」という検索フィルターになり、Red Hat Linux Agent の場合は「(&(eruid=root)(objectclass=erLinuxGroup))」という検索フィルターになります。

この次善策は、ルックアップに関するすべての問題で有効とは限りません。例えば、Windows の「Administrators」グループを検索するために、Windows 2000 Agent で「(erW2KGroupName=Administrators)」を使用すると、空の結果セットが戻されます。「(&(eruid=Administrators)(objectclass=erW2KGroup))」という拡張検索フィルターも、空の結果セットを戻します。

ルックアップで問題が発生した場合は、次のように対処してください。

1. 最新バージョンの Agent を使用していることを確認します。
2. 前述の次善策を試します。
3. 次善策が有効でない場合は、Agent のスキーマを調べて、項目の識別に利用できる他の属性を検索します。

項目の識別に利用できる Agent のスキーマの他の属性の例をいくつか紹介します。

- 前述の Windows の「Administrators」グループの検索では、erW2KGroupName 属性の代わりに erW2KGroupCommonName 属性を利用できます。「(erW2KGroupCommonName=Administrators)」というフィルターは、正常に機能して「Administrators」グループ項目を取得できます。
- LDAP-X Agent では、デフォルトの eruid 属性を使用して LDAP ユーザー (erXLdapAccount クラス) の検索を試行すると、処理が失敗します。この場合は、項目の識別に cn 属性を利用することができます。

関連情報

DAML/DSML プロトコル、
107 ページの『TIM DSMLv2 コネクター』.

IBM MQ コネクター

IBM MQ コネクターは、205 ページの『JMS コネクター』 の特殊インスタンスです。

JDBC コネクター

JDBC コネクターは、さまざまなシステムへのデータベース・アクセスを提供します。 JDBC を使用してシステムに到達するには、システム・プロバイダーが提供する JDBC ドライバーが必要です。

このドライバーは、jar または zip ファイルの形式で製品とともに提供されます。これらのファイルは、パス内に配置するか、インストールされている IBM Security Directory Integrator の `jars/` ディレクトリーにコピーする必要があります。コピーしない場合、「Unable to load T2 native library」といった不可解なメッセージが出力されることがあります。これは、ドライバーがクラスパス上に見つからなかったことを示しています。

また、この jar または zip ファイルにあるどのクラスが JDBC ドライバーをインプリメントするのかを判断する必要があります。この情報は、「**JDBC ドライバー**」パラメーターに設定します。

JDBC コネクターには、SELECT、INSERT、UPDATE、および DELETE ステートメントを入力するための複数行入力フィールドがあります。構成が完了している JDBC コネクターでは、コネクター自体の自動生成ステートメントの代わりに、これらのステートメントの値が使用されます。この値は、完全な SQL ステートメントを生成するパラメーター置換モジュールにより拡張されたテンプレートです。このテンプレートは、コネクター構成、*searchcriteria* オブジェクト、および *conn* オブジェクトにアクセスできます。コネクターは*作業* オブジェクトの内容を認識しないため、置換で *作業* オブジェクトを使用することはできません。コネクター構成では他のプロバイダー・パラメーターもサポートされています。

JDBC コネクターでは、AddOnly、更新、削除、ルックアップ、イテレーター、およびデルタの各モードがサポートされています。

原則的に、このコネクターは SSL プロトコルを使用してセキュア接続を処理できますが、この SSL サポートをセットアップするために、ドライバー固有の構成手順が必要になる場合があります。詳しくは、製造元のドライバー資料を参照してください。

コネクターの構造とワークフロー

JDBC コネクターは、コネクターの初期化中に指定のデータ・ソースへの接続を確立します。このことについては、ここに記載されている情報を通じて詳しく知ることができます。

指定のデータ・ソースへの接続確立中に、追加のプロバイダー・パラメーターがあるかどうかが調べられ、指定されている場合は設定されます。接続初期化中には、自動コミット・フラグ設定の処理および設定も行われます。

JDBC コネクターは、事前定義のマッピング・テーブルを使用して SQL ステートメントを内部で作成します。AddOnly、更新、削除、イテレーター、およびルックアップの各モードでのコネクター・フローの振る舞いは、他のコネクターと同じです。

また、このコネクターではデルタ・モードがサポートされています。JDBC コネクターのデルタ機能は、ALComponent (すべてのコネクターに共通の汎用ビルディング・ブロック) により処理されます。ALComponent がルックアップを実行し、デルタ項目をターゲット項目に適用してから更新を実行して、その後で正しいデータベース操作を決定します。次に、コネクターが操作の内容に対応して追加、変更、または削除の SQL ステートメントを使用します。

JDBC ドライバーについて

このセクションに記載されている情報を通じて、JDBC ドライバーについて理解することができます。さらに、インストールについて、および JDBC ドライバーのインストール中に注意すべき事項についても知ることができます。

JDBC コネクターがリレーショナル・データベースにアクセスするためには、Java クラス・ライブラリーに含まれる一連のサブルーチンまたはメソッドである*ドライバー* にアクセスする必要があります。このライブラリーは、IBM Security Directory Integrator の *CLASSPATH* に存在する必要があります。存在しない場合、IBM Security Directory Integrator はコネクターの初期化時にライブラリーをロードできないため、リレーショナル・データベースと通信できません。IBM Security Directory Integrator によって使用できる JDBC ドライバー・ライブラリーのインストールは、ライブラリーを *TDI_install_dir*/jars ディレクトリーまたはその下位にある任意のディレクトリー (例: *TDI_install_dir*/jars/local) にコピーする方法で行うことをお勧めします。

注:

1. 一部のドライバーには、ネイティブ・コード (通常は .dll ファイルまたは .so ファイル内に記述) が含まれている場合があります。実行時に IBM Security Directory Integrator が取得できるように、これらのファイルを PATH 変数に追加する必要があります。
2. クラス名の重複には注意してください。CLASSPATH 内の他のライブラリーのクラスと重複するクラスがライブラリーに含まれる場合、どちらのクラスがロードされるか不確定になります。
3. ライブラリーは、すべてのユーザーから読み取り可能である必要があります。
4. ライブラリーを使用するアプリケーションは、ライブラリーのインストール後に再始動する必要があります (構成エディター、IBM Security Directory Integrator サーバー)。

JDBC 経由で RDBMS にアクセスするには、次の 4 つの基本的な方法があります (ドライバー・タイプと呼ばれることもあります)。

1. JDBC API を他のデータ・アクセス API へのマッピングとして実装するドライバー (Open Database Connectivity (ODBC) など)。このタイプのドライバーは、通常、ネイティブ・ライブラリーに依存しているため、移植性が制限されています。Type 1 のドライバーの例としては、JDBC-ODBC ブリッジ・ドライバーがあります。このドライバーは通常、JVM の一部であるため、IBM Security Directory Integrator のクラスパスで個別に指定する必要がありません。

   ODBC の構成については、 190 ページの『ODBC データベース・パスの指定』を参照してください。

   注: JDBC-ODBC ブリッジは、IBM によって製品とともに出荷される、プラットフォーム依存の各種 JVM のいずれにも存在する可能性があります。ただし、IBM が JDBC-ODBC ブリッジをサポートするのは、Windows プラットフォームのみです。また、専用のネイティブ (「Type 4」) ドライバーと比較すると、パフォーマンスが低い可能性があります。市販の ODBC/JDBC ブリッジを使用できます。JDBC-ODBC ブリッジが必要な場合は、市販のブリッジの購入を検討してください。また、JDBC-ODBC ブリッジ・ドライバーに関する説明 (http://java.sun.com/products/jdbc/driverdesc.html) も参照してください。
2. Java プログラミング言語で記述された部分とネイティブ・コードで記述された部分があるドライバー。このようなドライバーは、接続先データ・ソースに固有のネイティブ・クライアント・ライブラリーを使用します。また、ネイティブ・コードのために移植性が制限されています。
3. Pure Java クライアントを使用し、データベースに依存しないプロトコルを使用してミドルウェアと通信するドライバー。ミドルウェア・サーバーはクライアントの要求をデータ・ソースに伝えます。
4. 特定のデータ・ソース用にネットワーク・プロトコルを実装する Pure Java ドライバー。クライアントは、データ・ソースに直接接続します。

Windows 上の JDBC-ODBC ブリッジを例外として、ここでは IBM Security Directory Integrator で Type 4 のドライバーのみを使用します。その他のタイプのドライバーについても、サポート対象の各データベースのコンテキストで、理解を深めるために説明します。

JDBC Type 3 および Type 4 のドライバーは、ネットワーク・プロトコルを使用してバックエンドと通信します。通常、これは TCP/IP 接続を意味します。この接続は単純な TCP/IP ソケット接続、またはドライバーがサポートする場合は Secure Socket Layer (SSL) 接続にすることができます。

注: カスタムの準備済みステートメントを使用する場合は、JDBC の使用ドライバーが JDBC 3.0 に準拠していることを確認します。ドライバーは JDBC 2.0 のみを実装するため、IBM solidDB® 6.5 には既知の問題があります。このデータベースを操作するとき「**カスタムの SQL 準備済みステートメントを使用**」オプションが使用可能な場合、`java.lang.NullPointerException` がスローされます。

DB2 への接続

ここに記載されている情報とリンクを使用することで、JDBC コネクターを DB2 に接続する方法について理解することができます。

IBM Security Directory Integrator にバンドルされている JDBC および SQLJ 用の IBM ドライバーは、http://www-306.ibm.com/software/data/db2/java から入手できます。このドライバーは、JDBC 1.2、JDBC 2.0、JDBC 2.1、および JDBC 3.0 に準拠しています。

IBM DB2 用の JDBC ドライバーに関する情報は、オンラインで参照可能です。構成のための出発点と実例は、DB2 Developer のマニュアルのセクション『JDBC アプリケーションによるデータ・ソースへの接続方法』にあります。このドライバーは、ご使用の目的に合う場合と合わない場合があります。

ドライバーのライセンス交付

z/Series および iSeries® 用の DB2 を除いて、このドライバーでは DB2 データベース・システム用の追加のライセンス交付が必要ありません。適切なライセンス・ファイル (`db2jcc_license_cu.jar`) が既に含まれています。ドライバーが z/Series および iSeries の 2 つのシステムと通信するには、DB2 Connect™ 製品を入手してそのライセンス・ファイル `db2jcc_license_cisuz.jar` を *jars/3rdparty/IBM* ディレクトリーにコピーする必要があります。また、このドライバーはネイティブ・コンパイルされたコードを持つファット・クライアント (.dll/.so) であるため、これらのライブラリーを使用するためには DB2 Connect のインストール・パスを PATH 変数に追加する必要があります。

JDBC ドライバーのアーキテクチャーに基づいて、DB2 JDBC ドライバーは 4 つのタイプに分けられます。

1. DB2 JDBC Type 1

   DB2 ODBC (JDBC ではない) ドライバー。JDBC-ODBC ブリッジ・ドライバーを使用して接続します。このドライバーは、原則的にこれ以上は使用されません。

   JDBC Type 1 のドライバーは、JDBC 1.2、JDBC 2.0、および JDBC 2.1 で使用できます。

   ODBC の構成については、190 ページの『ODBC データベース・パスの指定』を参照してください。

2. DB2 JDBC Type 2

   DB2 JDBC Type 2 ドライバーは、非常によく知られており、しばしば *app* ドライバーと呼ばれます。app ドライバーという名前は、このドライバーがローカルの DB2 クライアントを介してリモート・データベースへのネイティブ接続を実行するという概念、およびパッケージ名 (COM.ibm.db2.jdbc.app.*) が元になっています。

   言い換えれば、JDBC 呼び出しを行っているアプリケーションが実行されているマシン上に DB2 がインストールされている必要があります。JDBC Type 2 ドライバーは、Java とネイティブ・コードを組み合わせているので、一般に Java のみの Type 3 または Type 4 インプリメンテーションよりも良好なパフォーマンスを実現します。

このドライバーのインプリメンテーションでは、ネイティブ・プラットフォームの C ライブラリーにバインドされた Java 層を使用します。Type 2 ドライバーは非常に高いパフォーマンスと完全な機能を提供するため、J2EE プログラミング・モデルを使用しているプログラマーは Type 2 ドライバーに引きつけられます。Type 2 ドライバーは、J2EE サーバーでの使用も認定されています。

このタイプのドライバーのインプリメンテーション・クラス名は、*com.ibm.db2.jdbc.app.DB2Driver* です。

JDBC Type 2 ドライバーを使用して、JDBC 1.2、JDBC 2.0、および JDBC 2.1 をサポートすることができます。

3. DB2 JDBC Type 3

   JDBC Type 3 ドライバーは Pure Java インプリメンテーションであり、DB2 JDBC アプレット・サーバーを提供するミドルウェアと通信する必要があります。このドライバーは、Java アプレットが DB2 データ・ソースにアクセスできるようにするために設計されました。このドライバーを使用するアプリケーションは、DB2 クライアントがインストールされている別のマシンと通信することができます。

   JDBC Type 3 ドライバーは、net ドライバーと呼ばれることもよくあります。これは、パッケージ名 (COM.ibm.db2.jdbc.net.*) が元になっています。

   このタイプのドライバーのインプリメンテーション・クラス名は、*com.ibm.db2.jdbc.net.DB2Driver* です。

   JDBC Type 3 ドライバーは、JDBC 1.2、JDBC 2.0、および JDBC 2.1 とともに使用できます。

4. DB2 JDBC Type 4

   JDBC Type 4 ドライバーも Pure Java インプリメンテーションです。 JDBC Type 4 ドライバーには、分散リレーショナル・データベース体系アプリケーション・リクエスター (Distributed Relational Database Architecture™ Application Requester (DRDA® AR)) 機能が組み込まれているので、このドライバーを使用するアプリケーションは、DB2 クライアントと接続インターフェースを取る必要がありません。

   このタイプのドライバーのインプリメンテーション・クラス名は、*com.ibm.db2.jcc.DB2Driver* です。

   このドライバーの最新バージョン (9.1) は、SSL 接続をサポートします。そのためには、「**追加のプロバイダー・パラメーター**」フィールドのプロパティーを設定する必要があります。詳しくは、http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.apdv.java.doc/doc/rjvdsprp.htmを参照してください。着信 SSL 接続を受け入れるようにターゲット・データベースをセットアップする必要があります。

JDBC コネクターの照会スキーマから例外がスローされる場合、または JDBC テーブルに対する追加/更新操作が BLOB データ・タイプで失敗する場合は、データベース管理者に連絡し、スキーマの取得に必要なストアード・プロシージャーをイン

ストールするよう要請してください。 Java から DB2 へのアクセスについて詳しくは、「Linux、UNIX、および Windows 用の DB2 UDB における Java 開発の概要」も参照してください。

Informix Dynamic Server への接続

ここに記載されている情報とリンクを使用することで、Informix Dynamic Server に接続することができます。

Informix® Client SDK をインストールする場合は、Informix ODBC ドライバーもインストールします。このドライバーを使用すると、JDBC-ODBC ブリッジ・ドライバーを使用できるようになります。このドライバーを実動環境で使用することは推奨されません。ODBC の構成については、 190 ページの『ODBC データベース・パスの指定』を参照してください。

ただし、Informix JDBC ドライバー、バージョン 3.0 の使用をお勧めします。これは Pure Java (Type 4) ドライバーで、分散トランザクションのサポートが強化されており、IBM WebSphere® Application Server と連携するように最適化されています。

Java プログラミング言語で記述されたインターフェースとクラスの集合からなります。ドライバーには組み込み SQL/J が含まれており、Java での組み込み SQL をサポートしています。

このドライバーのインプリメンテーション・クラスは *com.informix.jdbc.IfxDriver* です。 Informix ドライバーのインストール方法については、http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp?topic=/com.ibm.conn.doc/jdbc_install.htm を参照してください。

Oracle への接続

このセクションには、Oracle ベースの JDBC ドライバーのリストが記載されています。

JDBC ドライバーのアーキテクチャーに基づいて、以下のタイプのドライバーを Oracle から入手可能です。

1. Oracle JDBC Type 1

   Oracle ODBC (JDBC ではない) ドライバー。JDBC-ODBC ブリッジ・ドライバーを使用して接続します。Oracle は ODBC ドライバーを提供していますが、ブリッジ・ドライバーは提供していません。代わりに、JVM の一部であるデフォルトの JDBC-ODBC ブリッジを使用するか、または http://java.sun.com/products/jdbc/drivers.html から JDBC-ODBC ブリッジ・ドライバーを入手します。この構成は正常に機能しますが、JDBC Type 2 または Type 4 のドライバーはより多くの機能を提供し、速度も向上します。

   ODBC の構成については、 190 ページの『ODBC データベース・パスの指定』を参照してください。

2. Oracle JDBC Type 2

   Type 2 ドライバーには 2 つの種類があります。

   • JDBC OCI クライアント・サイド・ドライバー

このドライバーは、Java ネイティブ・メソッドを使用して、基礎となる C ライブラリー内のエントリー・ポイントを呼び出します。この C ライブラリーは OCI (Oracle Call Interface) と呼ばれ、Oracle データベースと対話します。JDBC OCI ドライバーでは、ドライバーと同じバージョンの Oracle クライアントがインストールされている必要があります。ネイティブ・メソッドを使用すると、JDBC OCI ドライバーがプラットフォーム固有になります。Oracle は、Solaris、Windows、およびその他多数のプラットフォームをサポートします。つまり、Oracle JDBC OCI ドライバーは C ライブラリーに依存しているので、Java アプレットには適していないということになります。バージョン 10.1.0 以降の JDBC OCI ドライバーは、OCI Instant Client 機能を使用してインストールできるので、完全な Oracle クライアントのインストールは不要です。詳しくは、Oracle Call Interface を参照してください。

- JDBC サーバー・サイド内部ドライバー

  このドライバーは、Java ネイティブ・メソッドを使用して、基礎となる C ライブラリー内のエントリー・ポイントを呼び出します。この C ライブラリーは Oracle サーバー・プロセスの一部で、Oracle 内の内部 SQL エンジンと直接通信します。ドライバーは、内部関数呼び出しを使用して SQL エンジンにアクセスするので、ネットワーク・トラフィックが発生しません。これによって、Java コードは可能な限り速い方法でサーバー上で稼働して基礎となるデータベースにアクセスすることができます。これは、同じデータベースへのアクセス以外には使用できません。

3. Oracle JDBC Type 4

   さらに、Type 4 ドライバーには 2 つの種類があります。

   - JDBC シン・クライアント・サイド・ドライバー

     このドライバーは Java を使用して直接 Oracle に接続します。このドライバーは、独自の TCP/IP ベースの Java ソケット・インプリメンテーションを使用して、Oracle の SQL*Net Net8 および TTC アダプターをインプリメントします。JDBC シン・クライアント・サイド・ドライバーを使用するには、Oracle クライアント・ソフトウェアのインストールは不要ですが、サーバーに TCP/IP リスナーを構成する必要があります。このドライバーは Java で記述されているので、プラットフォームに依存しません。 JDBC シン・クライアント・サイド・ドライバーは、どんなブラウザーにも Java アプリケーションの一部としてダウンロードすることもできます。クライアント・ブラウザー内で実行する場合、アプレットがサーバーへの Java ソケット接続を開くことをブラウザーで許可する必要があります。

     これは、最もよく使用されるドライバーです。一般に、TCP/IP 以外のネットワークのサポートなどの OCI 固有の機能を使用する必要がある場合以外は、JDBC シン・ドライバーを使用します。

     このドライバーの現在のインプリメンテーション・クラスは *oracle.jdbc.driver.OracleDriver* です。

   - JDBC シン・サーバー・サイド・ドライバー

     このドライバーは Java を使用して直接 Oracle に接続します。このドライバーは Oracle データベースで内部的に使用され、JDBC シン・クライアント・

サイド・ドライバーと同じ機能を提供しますが、Oracle データベース内部で稼働するのでリモート・データベースへのアクセスに使用されます。このドライバーは Java で記述されているので、プラットフォームに依存しません。クライアント・アプリケーションのシン・ドライバーとサーバー内部のシン・ドライバーのどちらを使用しても、コードに違いはありません。

Java から Oracle へのアクセスの詳細については、Java、JDBC、およびデータベース Web サービス、および Oracle JDBC の FAQ も参照してください。

SQL Server への接続

ここに記載されている情報を使用して、SQL Server に接続する方法を理解することができます。

JDBC 用の Microsoft SQL Server 2008 ドライバーは、JDBC 1.22、JDBC 2.0、および JDBC 3.0 の仕様をサポートします。これは Type 4 ドライバーです。

このドライバーのインプリメンテーション・クラスは *com.microsoft.sqlserver.jdbc.SQLServerConnection* です。このインプリメンテーション・クラスは、ドライバー・ファイル `sqljdbc.jar` に含まれています。このファイルは、通常、MS SQL Server 2008 のインストール済み環境の `<Microsoft SQL Server 2008-Install-Dir>¥sqljdbc_1.1.1501.101_enu¥sqljdbc_1.1¥enu¥sqljdbc.jar` にあります。

また、その他のサード・パーティー製のドライバーを使用して Microsoft SQL Server に接続することもできます。

適切な選択肢としては、GNU LGPL で配布されている jTDS JDBC 3.0 ドライバーなどがあります。これは Type 4 ドライバーであり、Microsoft SQL Server 6.5、7、2000、および 2005 をサポートします。jTDS は JDBC 3.0 と 100% の互換性があり、前方のみおよびスクロール可能/交信可能な結果セット、並行 (完全独立) ステートメントをサポートし、すべての `DatabaseMetaData` メソッドおよび `ResultSetMetaData` メソッドをインプリメントします。jTDS は、http://jtds.sourceforge.net から自由にダウンロード可能です。この Web サイトでは、このドライバーに関する詳細情報を入手できます。

Sybase Adaptive Server への接続

ここに記載されている情報を使用することで、Sybase Adaptive Server に接続できるようになります。

Sybase の jConnect for JDBC ドライバーは、Adaptive Server Enterprise、Adaptive Server Anywhere、Adaptive Server IQ、および Replication Server など、すべての Sybase 製品ファミリーへのハイパフォーマンスのネイティブ・アクセス (Type 4) を提供します。

jConnect for JDBC は、Java JDBC 標準のインプリメンテーションです。jConnect for JDBC は JDBC 1.22 および JDBC2.0 をサポートしており、さらに JDBC 3.0 に制限付きで準拠しています。複数階層環境や異機種混合環境において、Java 開発者にネイティブ・データベース・アクセスを提供します。jConnect for JDBC は、前

もってクライアントをインストールすることなく、すぐにダウンロードして、IBM Security Directory Integrator のようなシン・クライアント Java アプリケーションで使用できます。

このドライバーのインプリメンテーション・クラス名は *com.sybase.jdbc3.jdbc.SybDriver* です。

また、その他のサード・パーティー製のドライバーを使用して Sybase に接続することもできます。

適切な選択肢としては、GNU LGPL で配布されている jTDS JDBC 3.0 ドライバーなどがあります。これは Type 4 ドライバーであり、Sybase 10、11、12、15.1 をサポートします。jTDS は JDBC 3.0 と 100% の互換性があり、前方のみおよびスクロール可能/交信可能な結果セット、並行 (完全独立) ステートメントをサポートし、すべての `DatabaseMetaData` メソッドおよび `ResultSetMetaData` メソッドをインプリメントします。jTDS は、http://jtds.sourceforge.net から自由にダウンロード可能です。この Web サイトでは、このドライバーに関する詳細情報を入手できます。

Derby への接続

ここに記載されている情報を使用することで、Derby に接続できるようになります。

Derby は、IBM DB2 をモデルにして開発されたリレーショナル・データベースで、すべて Java で記述されています。このデータベース製品およびそのドライバーは、IBM Security Directory Integrator にバンドルされています。ネットワーク・ドライバーは Type 4 ドライバー (ネイティブ Java コード) です。

このドライバーのインプリメンテーション・クラス名は *org.apache.derby.jdbc.ClientDriver* です。

Derby の使用時に JDBC URL を構成する方法の詳細については、「Derby Developer's Guide」の『*Conventions for specifying the database paths*』を参照してください。

IBM solidDB への接続

この情報を使用することで、IBM solidDB に接続できるようになります。

IBM solidDB は、Derby と比較して拡張されたパフォーマンスを提供するリレーショナル・メモリー内データベースです。そのため、システム・ストアとして Derby データベースの代わりに使用され、依存するコンポーネントのパフォーマンスが向上します。

IBM solidDB で提供されるドライバーは、Type 4 です (Java で完全実装)。このドライバーは、*SolidDB_install_dir*`/jdbc/SolidDriver2.0.jar` からデータベース・インストールで入手できます。

IBM solidDB について詳しくは、http://publib.boulder.ibm.com/infocenter/soliddb/v6r3/index.jsp を参照してください。

注: IBM solidDB のドライバーは JDBC 3.0 には対応しておらず、JDBC 2.0 のみを実装しています。そのため、カスタムの準備済みステートメントを使用すると問題が発生する可能性があります。

ODBC データベース・パスの指定

JDBC-ODBC ブリッジ (Windows システムのみでサポートされます) を使用する ODBC 接続を使用する場合は、ODBC ドライバーで使用する必要があるデータベースまたはファイル・パスを指定できます (ODBC ドライバーで許可される場合)。

このタイプの構成を使用すれば、コネクターが使用する各データベースまたはファイル・パスのデータ・ソース名を定義する必要がなくなります。

jdbcDriver

sun.jdbc.odbc.JdbcOdbcDriver

jdbcSource

jdbc:odbc:*driver name*;DBQ=*path*

このパラメーターの構文は、以下の条件によって異なります。

MS Access がインストールされている場合

ODBC データ・ソース制御パネルを開き、「**ユーザー DSN**」タブを選択します。このテーブルには、JDBC ソース・パラメーターで使用できるドライバー名が表示されます。例えば、MS Access データベース (C:¥Documents and Settings¥*username*¥My Documents¥mydb.mdb) にアクセスするには、JDBC ソースに次の値を指定します。

```
jdbc:odbc:MS Access Database;dbq=C:¥Documents and Settings¥username¥My Documents¥mydb.mdb
```

MS Access がインストールされていない場合

MS Access がインストールされておらず、Windows システムを使用している場合は、次の値を使用します。

```
jdbc:odbc:Driver={MS Access Driver
 (*.mdb)};dbq=C:¥Documents and Settings¥username¥My Documents¥mydb.mdb
```

あるいは、Windows のシステム DSN ユーティリティーを使用します。このユーティリティーは、「**管理ツール**」**->**「**データ・ソース (ODBC)**」から使用できます。システム DSN を定義したら、jdbcSource パラメーターを次のように使用します。

```
jdbc:odbc:myDSNNameHere
```

ユーティリティーで取得するドライバー・リストを確認します。JDBC URL は、このリストで記述されているとおりに正確に指定してください。

スキーマ

JDBC コネクター

イテレーター・モードとルックアップ・モードでは、JDBC コネクター・スキーマは、指定された表についてデータベースから読み取られたメタデータ情報に依存します。表名が指定されていない場合、スキーマは SQL Select/Lookup ステートメン

ト (定義されている場合のみ。 196 ページの『SELECT、INSERT、UPDATE、および DELETE ステートメントのカスタマイズ』を参照) を使用して検索されます。

AddOnly、削除、更新、およびデルタ・モードの場合、 JDBC コネクター・スキーマは、指定された表についてデータベースから読み取られたメタデータ情報に依存します。

構成

ここに記載されているパラメーターを使用して JDBC コネクターを構成することができます。

このコネクターは、以下のパラメーターを必要とします。

JDBC URL

JDBC プロバイダーの資料を参照してください。一般的な RDBMS システムの代表的な URL を以下に示します。

表 26. JDBC URL の例

RDBMS	接続 URL の例
IBM DB2 (DRDA ドライバーを使用)	「jdbc:db2://hostname:port/dbname」
Informix Dynamic Server 11.7	「jdbc:informix-sqli://hostname:port/dbname:informixserver=<Informix Server Name>」
Oracle (「シン・ドライバー」を使用)	「jdbc:oracle:thin:@hostname:1521:SID」、「host:port:sid」構文を使用、TNSListener はポート 1521 での接続を受け入れる
Microsoft SQL Server (Microsoft のドライバーを使用)	「jdbc:sqlserver://hostname:1433;databasename=dbname;」、 SQL Server はポート 1433 で接続を listen
Sybase 15.5 (v. 10 以降の旧バージョンも含む)、jConnect 6.05 を使用	「jdbc:sybase:Tds:hostname:port/」
Derby	「jdbc:derby://hostname:port/<server path to database>;options」
IBM solidDB7.0	「jdbc:solid://hostname:port」

JDBC ドライバー

JDBC ドライバーのインプリメンテーション・クラス名。デフォルト値の sun.jdbc.odbc.JdbcOdbcDriver は JDBC-ODBC ブリッジを示していますが、これは実動使用にはお勧めできません。別のタイプのドライバーを使用できるデータベースの場合、一般的なドライバーのインプリメンテーション・クラス名は以下のとおりです。

表 27. ドライバーのインプリメンテーション・クラス名

RDBMS	ドライバーのインプリメンテーション・クラス名
IBM DB2、タイプ 2 または 4	com.ibm.db2.jcc.DB2Driver
Oracle、タイプ 4	oracle.jdbc.driver.OracleDriver
Informix Dynamic Server 11.7	com.informix.jdbc.IfxDriver
Microsoft SQL Server、タイプ 4	com.microsoft.sqlserver.jdbc.SQLServerDriver
Sybase 15.5 (v. 10 以降の旧バージョンも含む)	com.sybase.jdbc3.jdbc.SybDriver
Derby	org.apache.derby.jdbc.ClientDriver
IBM solidDB7.0	solid.jdbc.SolidDriver

182 ページの『JDBC ドライバーについて』および 52 ページの『データベース・コネクター』も参照してください。

ユーザー名
このユーザー名を使用してデータベースにサインオンします。このユーザーからアクセス可能な表のみが表示されます。

パスワード
このユーザーのサインオンに使用するパスワード。

スキーマ
使用するデータベースの表内のスキーマ。ブランクのままにすると、jdbcLogin (「**ユーザー名**」パラメーター) の値が使用されます。

注: IBM Security Directory Integrator の資料では、アクセスしているオブジェクトのデータ定義を意味する用語として「スキーマ」が使用されています。ただし、RDBMS におけるスキーマという用語には、1 つの ID (ユーザー名) でグループ化されたデータ定義、表、およびオブジェクトのコレクション全体という別の意味があります。この特定のコネクターにおける特定のパラメーターについては、RDBMS におけるスキーマの意味で使用しています。

表名 操作する表またはビュー。このパラメーターは、コネクターがルックアップ・モードまたは更新モードで稼働している場合のみ使用されます。**SQL SELECT** パラメーターが指定されていない場合は、イテレーター・モードのコネクターは、このパラメーターも使用してデフォルトの SELECT ステートメントを構成します。

選択... このボタンをクリックすると、使用できる表名のリストが表示され、そこから選択した表名が「**表名**」フィールドに入力されます。この機能は、基礎にあるデータベースでサポートされている場合のみ使用できます。例えば、ODBC を使用している Microsoft Access はこの機能をサポートしていません。

NULL 値を戻す
このチェック・ボックスにチェック・マークを付けると、NULL 値属性は空の値を戻します。チェック・マークを外したままにすると、定義済みの NULL 動作が実行されます。デフォルトの NULL 動作が実行されると、NULL を受け取る属性が除去されます。

コミット
いつデータベース・トランザクションをコミットするかを制御します。オプションは以下のとおりです。

- **各データベース操作後** (デフォルト)
- **各データベース操作後 (SELECT を含む)**
- **コネクターのクローズ時**
- **手動**
- **サイクルの終わり**

手動は、必要に応じて、JDBC コネクターの *commit()* メソッドまたは *rollback()* を呼び出す必要があることを意味します。

注: 「**各データベース操作後 (SELECT を含む)**」というオプションは、トランザクションでデータベース表を読み取り操作のみの目的で選択した場合でもデータベース表をロックするデータベース (DB2 など) 用に提供されました。

準備済みステートメントを使用する
このチェック・ボックスの値により、準備済みステートメントまたはステートメントのいずれを使用するかが決定されます。このチェック・ボックスが選択されている場合、JDBC コネクターは準備済みステートメントを使用し、選択されていない場合はステートメントを使用します。デフォルトはチェック・マークあり (「*true*」) で、SQL ステートメントのプリコンパイル、および正常状態へのフォールバックが試行されます。

カスタムの SQL 準備済みステートメントを使用
カスタムの指定された SQL ステートメントが準備されたステートメントかどうか (true または false) を指定します。デフォルトではチェックは外されています (false)。 198 ページの『準備済みステートメントをオフにするオプション』も参照してください。

SQL SELECT
繰り返しの対象の項目を選択するときに実行する SELECT ステートメント (イテレーター・モード)。このパラメーターをブランクのままにした場合は、デフォルトの構成 (SELECT * FROM TABLE) が使用されます。 196 ページの『SELECT、INSERT、UPDATE、および DELETE ステートメントのカスタマイズ』を参照してください。

SQL Select パラメーターの右にある「**...**」というマークのボタンを押すと、リンク基準フォームを記入および適切な SQL where 節を生成できる「リンク基準」ダイアログが表示されます。

「**追加**」ボタンをクリックすると、選択基準を作成する行が追加されます。「**いずれかと一致**」チェック・ボックスは、デフォルトの AND 式ではなく OR 式を生成します。これは、片方向のヘルパーであることに注意してください。 SQL SELECT パラメーターに既存のものはすべて、生成された式で置換されます。 SQL SELECT パラメーターが「where」節を含む場合は、where 節だけが置換されます。

SQL ルックアップ
ルックアップに使用するカスタム SQL ステートメント (ルックアップ、更新、および削除の各モードで使用されます)。

SQL 挿入
AddOnly モードまたは Delta モードでのデータベースへの挿入操作に使用するカスタム SQL ステートメント。

SQL 更新
更新モードまたは Delta モードでのデータベースの更新操作に使用するカスタム SQL ステートメント。

SQL 削除
削除モードまたは Delta モード使用時に使用するカスタム SQL ステートメント。

セッション・ステートメントの変更

このパラメーターは、ALTER SESSION コマンドを指定するための複数行フィールドです。次に示すのは ALTER SESSION コマンドの例です。

```
"SET NLS_FORMAT 'YYYY-MM-DD'"
```

追加のプロバイダー・パラメーター

追加の JDBC プロバイダー・パラメーター (name:value を各行に 1 つずつ)。このパラメーターでは、JDBC プロバイダーでサポートされる追加パラメーターを指定できます。サポートされているパラメーターについてドライバーの資料を参照し、サポートされているパラメーターを使用します。以下に DB2 固有の例を示します。

```
securityMechanism:KERBEROS_SECURITY
loginTimeout:20
readOnly:true
```

日付形式

ストリングとして供給された日付を解析するために使用する書式制御ストリング。事前定義書式制御ストリングのリストからストリングを選択するか、または独自のストリングを指定できます。

挿入の埋め込みを使用不可に設定

このチェック・ボックスの値により、特定のモードの挿入操作に対して埋め込みを使用不可にするかどうかが決定されます。デフォルトでは、このチェック・ボックスはオフになっています。つまり、埋め込みは使用不可になって**いません**。言い換えれば、AddOnly モード、更新モード、およびデルタ・モードでは埋め込みが使用可能です。

更新の埋め込みを使用不可に設定

このチェック・ボックスの値により、特定のモードの更新操作に対して埋め込みを使用不可にするかどうかが決定されます。デフォルトでは、このチェック・ボックスはオフになっています。つまり、埋め込みは使用不可になって**いません**。言い換えれば、更新モードおよびデルタ・モードの更新操作では埋め込みが使用可能です。

ルックアップの埋め込みを使用不可に設定

このチェック・ボックスの値により、特定のモードのルックアップ操作に対して埋め込みを使用不可にするかどうかが決定されます。デフォルトでは、このチェック・ボックスはオフになっています。つまり、埋め込みは使用不可になって**いません**。言い換えれば、Lookup モード、更新モード、削除モード、およびデルタ・モードのルックアップ操作では埋め込みが使用可能です。

テーブルの自動作成

コネクターがいずれかの出力モード (AddOnly または更新) で構成されている場合は、詳細セクションに以下の追加オプションがあります。

「**テーブルの自動作成**」オプションでは、コネクターの属性マップおよびスキーマに基づいて、コネクターによりシンプルなテーブルが作成されます。これは、データベースにテーブルが存在しない場合のみ実行されます。

テーブルの自動作成時、コネクターは最初に属性マップから列名を派生させます。属性マップが空の場合は、スキーマを使用して列名のリストを取得します。列名が決まると、各列名で SQL *CREATE TABLE* ステートメントが

生成されます。スキーマに列名の定義がある場合、列の構文を決定するために使用されます。スキーマには、列の構文を決定する 2 つのパートがあります。最初に、「ネイティブ構文」が指定されている場合は、それがそのまま使用されます。次に、提供されるネイティブ・スキーマがない場合、コネクターは「Java クラス」を使用して構文を派生させます。スキーマの「Java クラス」フィールドに、以下の値のいずれかを指定します。

表 28. Java クラスから SQL タイプへのマッピング

値	生成される SQL タイプ
整数または java.lang.Integer	INT
ストリングまたは java.lang.String	VARCHAR(255)
Double または java.lang.Double	DOUBLE
日付または java.util.Date	TIMESTAMP

列に関するスキーマ情報がない場合、または値が認識されない場合、コネクターは生成された「表の作成ステートメント」で「VARCHAR(255)」を使用します。

コネクター・フラグ

特定の動作を使用可能にするフラグのリスト。

{ignoreFieldErrors}

このフラグは、フィールドの値の取得結果がエラーになる場合に、例外をスローする (コネクターの *Fail EventHandler を呼び出す) 代わりに Java 例外オブジェクトを値として戻すように、コネクターに指示します。

詳細ログ

このフィールドをチェックすると、追加のログ・メッセージが生成されます。

リンク基準の構成

ルックアップ、削除、更新、デルタの各モードのコネクターの構成で指定されているリンク基準を使用することができます。これらのモードは、データベースとの対話に使用される SQL 照会の WHERE 節を指定するために使用されます。

IBM Security Directory Integrator オペランド**等しい**は、SQL 照会では等号 (=) に変換され、**包含**、**先頭文字**、および**終了文字**の演算子は **LIKE** 演算子にマップされます。

更新または削除モードでのルックアップのスキップ

ここに記載されている情報を使用して、更新モードまたは削除モードでのルックアップのスキップについて知ることができます。

JDBC コネクターでは、更新または削除モードでの「**ルックアップのスキップ**」一般オプションがサポートされています。このオプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。コネクターの特殊なコードによって、更新または削除の実行時に影響を受ける項目の適切な数値が取得されます。

SELECT、INSERT、UPDATE、および DELETE ステートメントのカスタマイズ

JDBC コネクターには、SQL テンプレートの SQL 操作を実行する前に、この SQL テンプレートを拡張できる機能があります。テンプレートは 5 種類の操作で使用できます。このセクションを参照することで、これらの操作を理解することができます。

表 29. SQL 操作

操作	コネクター・パラメーター名	説明	モード
SELECT	SQL SELECT	イテレーター・モードで使用されます (検索条件なし)。	イテレーター
INSERT	SQL 挿入	データ・ソースへの項目の追加時に使用されます。	更新、AddOnly、Delta
UPDATE	SQL 更新	データ・ソースの既存の項目の変更時に使用されます。	更新、Delta
DELETE	SQL 削除	データ・ソースの既存の項目の削除時に使用されます。	削除、Delta
LOOKUP	SQL ルックアップ	WHERE 文節を含む SELECT ステートメント。データ・ソースの検索時に使用されます。	ルックアップ、削除、更新

特定の操作のテンプレートが定義されていない場合 (NULL または空の場合など)、JDBC コネクターでは、独自の内部テンプレートが使用されます。

操作のテンプレートが定義されている場合は、このテンプレートから完全かつ有効な SQL ステートメントが生成される必要があります。テンプレートでは、標準のパラメーター置換オブジェクト (`mc`、`config`、`work`、`Connector` など)、コネクターに対して構成されているテーブルの JDBC スキーマ、およびその他いくつかの便利なオブジェクトを参照できます。

注: ルックアップ操作のテンプレートには、照会によって戻されるエレメントをフィルター処理する WHERE 文節が含まれている場合があります。ただし、コネクターがルックアップ、更新、または削除モードである場合、「**リンク基準**」パラメーターは、実行される照会用の WHERE 文節のアセンブルに使用されるため、必須です。**リンク基準**が省略されると、以下の例外がスローされます。

```
java.lang.Exception: CTGDIS143E No criteria can be built from input (no link criteria specified).
    at com.ibm.di.server.SearchCriteria.buildCriteria(Unknown Source)
```

したがって、構成が作成され、その構成でルックアップ・テンプレートの WHERE 文節が使用される場合は、必要ない場合でも**リンク基準**を提供する必要があります。コネクターは単にリンク基準を無視し、テンプレート照会が使用されます。不要な「ダミー」条件を**リンク基準**で追加しなくて済むようにするための解決策は、「**カスタム・スクリプトで基準を作成**」オプションをチェックし、表示されたスクリプト領域を空のままにすることです。

メタデータ・オブジェクト

ここに記載されている情報と例を参照して、メタデータについて理解することができます。

JDBC フィールド・タイプの情報は、メタデータと呼ばれる項目オブジェクトとして提供されます。メタデータ項目オブジェクトの各属性はフィールド名に対応しており、値はフィールドの対応するタイプです。例えば、以下の定義が含まれているテーブルがあるとします。

```
CREATE TABLE SAMPLE (
  name varchar(255),
  age numeric(10),
)
```

このテーブルは、パラメーター置換時に次の方法で参照されることがあります。

```
{javascript<<EOF
      metadata = params.get("metadata");
      if (metadata.getAttribute("name").equals("varchar"))
   return "some sql statement";
     else
  return "some other sql statement";
EOF
}
```

リンク・オブジェクト (リンク基準)

リンク・オブジェクトとは、リンク基準項目の配列です。各項目のフィールドでは、構成に基づいてリンク基準が定義されます。ここに記載されている情報とリンクを使用することで、リンク・オブジェクトについて詳しく知ることができます。

リンク基準の値はリンク ・オブジェクトに含まれています。構成リンク基準が *cn equals john doe* として定義されている場合は、テンプレートは以下の置換式を使用してこの情報にアクセスできます。

```
link[0].name → "cn"
link[0].match → "="
link[0].value → "john doe"
link[0].negate → false
```

SELECT 操作の完全なテンプレートは以下のようになります。

```
SELECT * FROM {config.jdbcTable} WHERE {link[0].name} = '{link[0].value}'
```

便利なオブジェクト

以下の表に記載されている情報を使用することで、便利なオブジェクトについて詳しく知ることができます。

JavaScript コードを使用せずに WHERE 文節または列名リストを生成する操作は、容易ではありません。簡易オブジェクトとして、JDBC コネクターは UPDATE および INSERT ステートメントで使用された列名を、列として使用可能にします。これは、SELECT ステートメントおよび LOOKUP ステートメントには適用されません。この値は、複数の列名をコンマで区切ったリストです。操作を単純化するため、テキストの WHERE 文節として「whereClause」を使用できます。両方の句の使用法を以下の例に示します。

```
SELECT {columns} from {config.jdbcTable} WHERE {whereClause}
```

例: *SELECT a,b,c from TABLE-A WHERE a > 1 AND b = 2*

表 30. 各種ステートメントで使用可能な情報

オブジェクト	SELECT	LOOKUP	INSERT	DELETE	UPDATE
config	はい	はい	はい	はい	はい
Connector	はい	はい	はい	はい	はい
metadata	いいえ	可能性あり	可能性あり	はい	はい
conn	いいえ	いいえ	はい	はい	はい
columns	いいえ	いいえ	はい	はい	はい
link	いいえ	はい	いいえ	はい	はい
whereClause	いいえ	はい	いいえ	はい	はい

準備済みステートメントをオフにするオプション

JDBC コネクターで準備済みステートメントをオフにするオプションがあります。それについて詳しく知るには、このセクションの情報を参照してください。

JDBC コネクターは接続 RDBMS サーバーで SQL ステートメントを効率的に実行するため *PreparedStatement* を使用します。ただし、JDBC ドライバーで PreparedStatements がサポートされていない場合があります。フォールバック・メカニズムとして、JDBC コネクターの構成で構成パラメーター (構成パネルで「**準備済みステートメントを使用する**」ラベルが付けられた jdbcPreparedStatement) を使用できます。この構成パラメーターは、JDBC コネクターが PreparedStatements を使用するかどうかを指定するブール値フラグです。このフラグが設定されている場合、コネクターは PreparedStatement を使用し、例外発生時には通常のステートメント (java.sql.Statement) にフォールバックします。このフラグが設定されていない場合は、JDBC コネクターは SQL 照会の実行時に通常のステートメントを使用します。IBM Security Directory Integrator ソリューションの開発者は、このチェック・ボックスにより、PreparedStatements の使用が原因で問題が発生した場合に対処することができます。チェック・ボックスはデフォルトでチェックされます。これはつまり、JDBC コネクターが PreparedStatement を使用するということです。

JDBC コネクターの `findEntry`、`putEntry`、`deleteEntry`、および `modEntry` メソッドは、usePreparedStatement フラグの値を調べ、PreparedStatements または Statements のどちらを使用するかを判別します。コネクター構成でこのフラグが指定されていない場合 (古いバージョンの構成など) は、このパラメーター値はデフォルトで「*true*」に設定されます。これにより、マイグレーションで問題または影響が発生しません。

カスタムの準備済みステートメント

ここに記載されている情報を使用して、カスタムの準備済みステートメントについて知ることができます。

カスタム SQL ステートメントなしで JDBC コネクターを使用すると、コネクターは JDBC ターゲット・データベースへのアクセスをより速くするために内部で*準備済みステートメント*を使用します。コネクターは、コネクター操作を実行するためのシンプルな SQL 準備済みステートメント (例えば、`SELECT * from TABLE WHERE x = ?`) を構築してから、JDBC API を使用してステートメントのプレースホルダー (疑問符) に値を指定します。これにより、値に対して複雑な文字列エンコードを行わなくても、データベースにさまざまな Java オブジェクトを簡単に提供できます。

ユーザーは、JDBC コネクターが作成する SQL ステートメントを指定変更しなければならない場合があります。これには、SQL 挿入や更新などの構成パラメーターが関与しています。ユーザーは特定の操作に使用する正確な SQL ステートメントを指定できます。SQL ステートメントは JDBC ドライバーにスタンドアロン・ステートメントとして指定されます。これは、基本的に SQL ステートメントがこのステートメントで使用される列の値を含むことを意味します。また、ユーザーが、構成自体の列の値も含めたステートメントを、値の複雑な文字列エンコードも含めて、構築する必要があることも意味します。

カスタムの準備済みステートメント機能により、ユーザーは適切な準備済みステートメントを使用できるようになりました。これにより、エンコードを行う必要がなくなったため、カスタム・ステートメントは以前より簡単に作成できるようになりました。また、ステートメントが呼び出し間で変わらない場合、準備済みステートメントが再利用されるため、実行速度がより速くなります。

JDBC コネクターには、「**カスタムの準備済みステートメントを使用する**」というパラメーターがあります。 準備済みステートメントを使用可能、または使用不可にするには、チェック・ボックスを使用します。これは、デフォルトで false となっています。このチェック・ボックスがチェックされている場合、カスタム SQL フィールドで正確な構文を使用する必要があります。SQL ステートメントでは、引き続き定数を使用できますが、ステートメントのすべての疑問符 (?) を正しくエスケープするか、または準備済みステートメントのプレースホルダーに式を指定しておく必要があります。出力マップ内の属性のリスト、またはその他の一般的な式から選択できるようにするコード補完ヘルパーを立ち上げるには、「?」を入力するか、Ctrl キーと <スペース> キーを一緒に使用します。式を選択して **Enter** キーを押すと、エディターが 2 つの疑問符 (?) 間にそのストリングを挿入します。何がプレースホルダー式として解釈されているかについて知らせるために、以下のように構文で強調表示されている点にも注意してください。

```
Select * from table where modified_date > ?{javascript return new java.util.Date()}
 and something_else < ?{conn.a}
```

また、プレースホルダーを通常持たない場合であっても、ステートメントでの式の使用は可能であることに注意してください。準備済みステートメントは、API を使用して指定することもできます。これが、最も使用しやすいオプションになる場合もあります。詳しくは、202 ページの『準備済みステートメントの仕様を可能にする API』を参照してください。 JDBC 2.0 ドライバー (特に IBM solidDB) が組み込まれた JDBC コネクターを使用する場合は、189 ページの『IBM solidDB への接続』も参照してください。

バックグラウンド

上記機能の必要性についてより詳しく説明するため、以下について考えてみます。

カスタム SQL ステートメントの現在のフォーマットでは、ユーザーが完全な SQL ステートメントを入力する必要があります。この方法は、手間がかかるとともに、値の性質が複雑である場合やバイナリーである場合には、記述することがほぼ不可能になります。 JDBC コネクターには、ユーザーによる準備済みステートメント・モードと現在のプレーン・ストリング・モードの切り替えが可能なオプション (**カスタムの準備済みステートメントを使用する**) があります。準備済みのステートメント・モードは、カスタム SQL ステートメントに別の構文を適用します。準備済

みステートメント・モードを選択すると、準備済みステートメント以外のモードが選択された場合同様、ストリングに対して置換は実行されません。

例として、シンプルな SELECT ステートメントを使用して、完全な SQL ステートメントの構築に関する問題を示します。

```
SELECT * FROM TABLE_NAME WHERE modified_date > 03/04/09
```

このステートメントは、特定の日付より後の modifed_date にフィルター処理する where 節を含んでいます。この例は、SQL ステートメントの問題を示しています。日付は 2009 年 3 月 3 日または 2009 年 4 月 4 日のどちらですか ?完全な SQL ステートメントの構築によりさらに問題が発生する場合があります。

「**カスタムの準備済みステートメントを使用する**」オプションでは、ユーザーは若干修正された SQL 準備済みステートメントを使用できます。JDBC 用語における SQL 準備済みステートメントは、値のプレースホルダーを含む完全な SQL ステートメントです。プレースホルダーは疑問符 (?) で、ランタイム時に値に置換されます。

```
SELECT * FROM TABLE_NAME WHERE modified_date > ?
```

ただし、JDBC コネクターは各プレースホルダーにどの値を指定するかを把握する必要があります。ランタイム時にどの値が入力されるのかを指定する機能も提供すると同時に、準備済みステートメントを構文的にできるだけ正確にするため、準備済みステートメントの構文が以下のように若干修正されます。

```
SELECT * FROM TABLE_NAME WHERE modified_date > {expression}
```

この構文は有効な準備済みステートメント構文ではありませんが、JDBC コネクターはこの文字列を解析して、ステートメントを実行する前に「{expression}」を 1 つの疑問符 (?) と置換します。「{expression}」は IBM Security Directory Integrator の式で、準備済みステートメントのプレースホルダーに値を指定します。カスタム SQL ステートメントのテキスト・フィールド・エディターは、ユーザーによるステートメントの構築を支援する追加機能を提供します。

注: カスタムの準備済みステートメントを使用する場合、適用可能な場合、ユーザーは WHERE 節も指定する必要があります。

追加の JDBC コネクター関数

このコネクターは、すべてのコネクターが公開する標準関数に加えて、ユーザーがスクリプト内で使用できる他のいくつかの関数も公開します。ここに記載されている情報を使用して、それらを呼び出す方法を理解することができます。

それらの関数を呼び出すには、特殊変数 *thisConnector* を使用して、`thisConnector.commit();` — のように記述します (コネクター内でスクリプトを記述できる場所から呼び出す場合は常に)。

commit()
保留中のデータベース操作をすべてコミットします。

execSQL (string)
任意の SQL コマンドを開始します。失敗した場合はエラー・ストリングを戻します。

execSQLSelect (string)

SQL SELECT コマンドを開始します。失敗した場合はエラー・ストリングを戻します。

getNextSQLSelectEntry ()

execSQLSelect を開始した後で、このメソッドを使用して、結果セットから次の項目を取得することができます。

この関数が正しく機能するためには、コネクターの「**表名**」パラメーターが空であることが必要です。

rollback()

最後の *commit()* 以後に実行されたデータベース操作をすべてバックアウトします (コミットが手動で行われた場合と、自動コミットによって行われた場合の両方を含む)。

上記の関数は、このコネクターの項目および属性マッピングの通常フローを妨げることはありません。

パラメーター置換を使用不可または使用可能にする API

JDBC コネクターでは、JDBC コネクターによって実行される SQL ステートメントのパラメーター置換を使用不可または使用可能にする API が公開されています。このことについては、ここに記載されている情報を通じて詳しく知ることができます。

上記のサブセクションでは、JDBC コネクターが、実行する挿入、更新、および削除の各 SQL コマンドのパラメーター置換機能にアクセスする方法について説明しています。場合によっては、これが原因で問題が発生します。その理由は、カスタマイズ SQL がサブストリング (「{」で始まり「}」で終わる) で終わる可能性があり、パラメーター置換メカニズムによって処理すべきではないのに処理されるためです。

```
/**
 * set enableParamSubstitute parameter
 *
 *
 */
public void setParameterSubstitution(boolean val)
{
 enableParamSubstitute = val;
}

/**
 * Returns value of enableParamSubstitute parameter
 *
 *
 */
public boolean getParameterSubstitution()
{
  return enableParamSubstitute ;
}
```

API を使用して不要なパラメーター置換を回避する代わりに、エスケープ文字を使用できます。

この場合のエスケープ文字は「¥」です。「¥」が {ArgumentIndex} や {TDIReference} の直前に置かれた場合 (つまり、¥{ArgumentIndex} や

¥{TDIReference} の場合)、パラメーター置換は行われません (処理されません)。実行されるのはエスケープ文字の除去のみで、パラメーター置換は発生しません。例えば、¥{TDIReference} を処理すると単に {TDIReference} となります。

準備済みステートメントの仕様を可能にする API

パワー・ユーザーの場合、API を使用して、使用する正しい準備済みステートメントおよびすべての値をより簡単に指定できます。以下の新しいメソッドが JDBC コネクターに追加されました。

```
public PreparedStatement setPreparedModifyStatement(String preparedSql)
public PreparedStatement setPreparedDeleteStatement(String preparedSql)
public PreparedStatement setPreparedInsertStatement(String preparedSql)
public PreparedStatement setPreparedFindStatement(String preparedSql)
public PreparedStatement setPreparedSelectStatement(String preparedSql)
```

これらのメソッドにより、ユーザーは例えば以下のようなコードを使って、特殊な選択を実行できます。

```
ps = thisConnector.connector.setPreparedSelectStatement
     ("Select * from tableName where fieldName = ? and field2= ?")
ps.setInteger(1, someValue)
ps.setObject(2, someObject)
```

その他の例については、メソッドの Javadoc を参照してください。

タイム・スタンプ

日付と時刻の両方を含むタイム・スタンプ値を保管する場合は、ここに示す詳細に注意しなければなりません。

次のような属性マッピングを使用できるように、必ず **java.sql.Timestamp** タイプのオブジェクトを指定するようにしてください。

```
ret.value = java.sql.Timestamp(java.util.Date().getTime());
```

何らかの理由でテーブルに DATE フィールドを格納する際に問題が発生した場合、例えば Oracle エラー「**ORA-01830: date format picture ends before converting entire input string**」が発生した場合などに、**java.sql.Timestamp** タイプが役立ちます。通常、ストリング形式の日付/時刻値を保管する場合、「**日付形式**」パラメーターを使用してストリングを基礎となるデータベースが予期する DATE タイプに変換しますが、このパラメーターと、ストリング形式の日付/時刻値の間に不一致があると、問題が発生します。

問題のトラブルシューティングを行うには、以下の手順で操作します。

- データ・パターン構成を調べます。
- IBM Security Directory Integrator によるこのフィールドの認識方法を確認します (コネクターの「スキーマ」タブを調べます)。ここで、JDBC ドライバーが Oracle Data タイプを java.sql.TimeStamp または java.sql.Date タイプに変換するとします (また、java.util.Date と java.sql.Date では、特に精度の点で違いがある点に注意してください)。例えば java.sql.Timestamp タイプに変換する場合、前述の構成を次のように指定します。

  ```
  ret.value = java.sql.Timestamp(java.util.Date().getTime());
  ```

  これがトラブルシューティングに有効であるかどうかを確認します。有効である場合は、以下を使用できます。

```
ret.value = java.sql.Timestamp(system.parseDate(work.getString("yourDate"),
            "yyyyMMddHHmmssz").getTime());
```

- 上記のいずれも有効でない場合は、コネクターを詳細ログ・モードに切り替え、コネクターがデータベースからスキーマを取得できるかどうかを確認します。取得できない場合、コネクターは準備済みステートメントを使用しません。この場合、準備済みステートメントを使用すると効率性が低下し、エラーが発生する可能性が高くなります。このため、コネクターの「**スキーマ**」構成パラメーターが正しく設定されていることを確認する必要があります。

埋め込み

ここに記載されている情報を使用することで、埋め込みと、それを使用可能または使用可能にする方法について詳しく知ることができます。

従来、JDBC コネクターでは、データの長さが CHAR データ・タイプの列の幅よりも小さい場合は、追加するデータを CHAR データ・タイプ列に埋め込みます。これは以前のデフォルトの動作であり、埋め込みを構成するオプションはありませんでした。

UTF-8 文字セットが登場したことで、このデフォルトの動作が予期しない動作をもたらす可能性があります。これは、コネクターでは UTF-8 データの正確な長さを判別できないためです。そのため、予測できない数の空白が追加される結果となり、データの長さが列の幅よりも大きくなり、その結果、データベースから例外がスローされます。

この問題を回避するために、コネクターが実行するさまざまな操作 (AddOnly、ルックアップ、更新、および削除の各モードで実行される挿入、更新、およびルックアップの操作) に対する埋め込みを任意で使用不可にするオプションを提供しています。この機能を選択するパラメーターについては、191 ページの『構成』セクションを参照してください。

UTF-8 データの場合は、埋め込みを使用不可にする必要があります。Latin 1 文字の場合は、埋め込みを使用可能または使用不可にすることができます。

ストアード・プロシージャーの呼び出し

JDBC コネクターの「getConnection()」メソッドにより、コネクターの初期化が正常に完了すると作成される JDBC 接続オブジェクトにアクセスできるようになります。このことについては、このセクションで詳しく知ることができます。

例えば、AL に DBconn という名前の JDBC コネクターがあるとします。

```
var con = DBconn.getConnector().getConnection();
```

これにより JDBC 接続オブジェクト (java.sql.Connection のインスタンス) にアクセスできるようになります。

注: コネクター自体の内部から呼び出される場合は、*thisConnector* 変数も使用できます。

次のコード例に、データベースに対してストアード・プロシージャーを呼び出す方法を示します。

```
// Stored procedure call
command = "{call DBName.dbo.spProcedureName(?,?)}";

try {
    cstmt = con.prepareCall(command);

    // Assign IN parameters (use positional placement)
    cstmt.setString(1, "Christian");
    cstmt.setString(2, "Chateauvieux");

    cstmt.execute();

    cstmt.close();
    // Security Directory Integrator will close the connection,
    // but you might want to force a close now.
    DBConn.close();
}

catch(e) {
    main.logmsg(e);
}
```

SQL データベース: 特殊文字を含む列名

特殊文字を含む名前を持つ列があるときに、AddOnly モードまたは更新モードを使用する場合は、次のようにしてください。

1. 更新コネクターまたは AddOnly コネクターの属性マップに進みます。
2. コネクター属性の名前 (作業属性ではありません) を、**name-with-dash** から **"name-with-dash"** に変更します (引用符を追加)。

この機能を使用する必要があるかどうかは、使用している JDBC ドライバーによっても異なりますが、標準の MS Access 2000 ではこの問題が生じます。

準備済みステートメントの使用

JDBC

このセクションでは、コネクターがどのように SQL 照会を作成するかについて説明します。このような内部の働きについて確認する必要がない場合は、このセクションをスキップできます。

データベースに対して、コネクターは、状況に応じて準備済みステートメントまたは動的照会を使用します。

- データベースからスキーマ定義を取得する場合は、コネクターは準備済みステートメントを使用します。 198 ページの『準備済みステートメントをオフにするオプション』も参照してください。
- その他の場合は、コネクターは動的 SQL 照会を作成します。

複数項目時

ここに記載されている情報とリンクを使用することで、複数項目を処理できるようになります。

コネクターで複数項目を戻すリンク基準が設定されている場合に何が発生するかについて詳しくは、 695 ページの『付録 C. AssemblyLine フロー・チャート』を参照してください。

削除モードまたは更新モードの JDBC コネクターについて setCurrent() メソッドを使用し、特に他のロジックを追加しなかった場合は、リンク基準を満たすすべての項目が削除または更新されます。

追加の組み込み再接続ルール

JDBC コネクターでは、IBM Security Directory Integrator に組み込まれている再接続エンジンが利用されています。ここに記載されている情報を通じてこれについて詳しく知ることができます。

このエンジンによって提供される標準の動作に加えて、JDBC コネクターには追加の組み込みルールが多数用意されています。コネクター固有の組み込みルールによって、java.sql.SQLException がスローされた場合や、正規表現を使用して評価される以下のメッセージが例外に含まれている場合は、再接続が実行されます。

- ^I/O.*
- ^Io.*
- ^IO.*
- ^ORA-01089.*
- ^Closed Connection.*

これらのルールは、コネクター構成の「**接続エラー**」ペインに表示されます。

関連情報

52 ページの『データベース・コネクター』

JMS コネクター

ここに記載されている情報とリンクを使用することで、JMS コネクターの機能について知ることができます。

「JMS」とは Java Message Service を指し、JMS コネクターは JMS 標準を使用して実装されたメッセージ・キューにタップできるコネクターです。JMS について詳しくは JMS のチュートリアル、API については JMS の仕様および API の資料を参照してください。

JMS コネクターの機能を以下に示します。

- Java メッセージ・サービス製品を使用して受け渡されるネイティブ項目オブジェクトの通信を有効にします。
- JMS メッセージ・ヘッダーとプロパティーをサポートします。
- JMS バスでの各種タイプのデータ (テキスト・メッセージ、オブジェクト・メッセージ、バイト・メッセージ) の送信をサポートします。
- ユーザーに対し、各種 JMS システムへ接続するための独自の Java コード (JMS イニシエーター・クラス) の作成を許可します。
- ユーザーに対し、各種 JMS システムへ接続するための JavaScript の作成を許可します。
- IBM MQ 以外のメッセージ・キューへの接続をサポートします。
- `acknowledge()` メソッドによる自動確認と手動確認をサポートします。

JMS コネクターは、IBM MQ Server やバンドルされている MQe などの JMS ベースのシステムへのアクセスを提供します。このコネクターの一部が事前に構成されているものは、「**IBM MQ コネクター**」という名前で存在しています。このコネクターでは JMS サーバー・タイプが非表示であり、「IBMMQ」に事前設定されています。

JMS コネクターを稼働させるために、ご使用の IBM Security Directory Integrator システムに対して実行する必要がある操作については、『特定トピック』の項を参照してください。

このコネクターは、Java Message Server 製品を使用してネイティブ項目オブジェクトおよび XML テキストの両方を受け渡しできる通信を可能にします。

JMS コネクターは、JMS メッセージ・プロパティーをサポートしています。JMS コネクターがメッセージを受け取ると、その JMS メッセージからのプロパティーが conn オブジェクトに取り込まれます (これらのプロパティーにアクセスするには、項目クラスの getProperty() メソッドおよび setProperty() メソッドを参照してください)。**conn** オブジェクトのプロパティーは、**jms.** の接頭部で始まり、その後に JMS メッセージ・プロパティー名が続きます。プロパティーは、JMS メッセージからの値を保持します。ユーザーは、メッセージを送信するときにプロパティーを設定でき、これらのプロパティーは送信する JMS メッセージに渡されます。JMS コネクターは、**conn** オブジェクトをスキャンして **jms.** で始まるプロパティーを見つけ、**conn** プロパティーからそれに対応する JMS メッセージ・プロパティーを設定します。

- JMS: correlationID=12 ——> **conn** jms.correlationID=12
- **conn**:jms.inReplyTo=12 ——> JMS:inReplyTo=12

conn オブジェクトを使用できるフックは限られています。「」の『conn オブジェクト』を参照してください。

JMS メッセージ・フロー

JMS コネクターが送信および受信するものは、すべて JMS メッセージです。JMS コネクターにより IBM Security Directory Integrator 項目オブジェクトから JMS メッセージへの変換とこの逆の変換が行われます。ここに記載されている情報を使用することで、それについて詳しく知ることができます。

各 JMS メッセージには、事前定義済みの JMS ヘッダー、ユーザー定義のプロパティー、そして、何らかの本文 (テキスト、バイト配列、またはシリアライズ Java オブジェクト) が含まれています。

JMS バスとの通信を大幅に容易にする JMS コネクターの一部としてのメソッドに、acknowledge() があります。メソッド acknowledge() は、**自動確認**のチェックを外してあるときに、JMS セッションの消費済みメッセージをすべて明示的に確認するために使用します。コネクターの acknowledge() を呼び出すことにより、コネクターは、メッセージの配達先セッションで消費されたすべてのメッセージを確認します。**自動確認**がチェックされている場合は、acknowledge() の呼び出しは無視されます。

受信メッセージの確認応答には十分な注意を払う必要があります。前述したように、JMS コネクターでは**自動確認**を使用せずに、AssemblyLine の JMS コネクターの直後にスクリプト・コネクターを挿入して、JMS コネクターの acknowledge() メソッドを呼び出すのが最良の方法です。こうすると、保管されるシステム・ストア内の関連メッセージ情報と JMS キュー通知との間のウィンドウが最小になります。このウィンドウで障害が発生した場合は、メッセージは再度受信されます。

逆に、**自動確認**に依存すると、キューのメッセージを検索 (および確認) した時点から、項目にマップされたメッセージの内容がシステム・ストアにしっかり保管されるまで存続するウィンドウが作成されます。このウィンドウで障害が発生した場合は、メッセージは失われるため、大きな問題となることがあります。

注: 自動確認がオンになっている場合に構成エディターで JMS コネクターを構成すると、問題が発生することがあります。これは、このような状況である限り、「**スキーマ**」->「**接続**」->「**GetNext**」、または入力マップから高速ディスカバーを使用してスキーマ・ディスカバリーのプロセスを行うと、メッセージが受信および消費される (つまり、入力キューからなくなる) ことが原因です。これによって、意図しない副次作用が起こる場合があります。このような状況を回避するには、スキーマ検出の前に**自動確認**をオフにします。ただし、こうした動作を望む場合は、オンに戻すことを忘れないでください。

IBM WebSphere MQ および JMS/非 JMS メッセージ・コンシューマー

JMS コネクターは、IBM WebSphere MQ へのメッセージ送信時に、提供されたメッセージを読み取るクライアントに応じて、2 種類のモードでメッセージを送信できます。

- メッセージが非 JMS クライアントにより読み取られる (デフォルト)
- メッセージが JMS クライアントにより読み取られる

デフォルトでは、コネクターはメッセージが非 JMS クライアントにより読み取られるようにメッセージを送信します。この 2 つのモードの主な相違点は、メッセージが非 JMS クライアントにより読み取られる予定である場合は JMS プロパティーが無視される点です。したがって、これらのプロパティーに対する後続のルックアップでは一致するものが見つかりません。

「JMS クライアントによる読み取り」モードに切り替えるには、「特定のドライバー属性」パラメーター値に、(その他の指定属性とは別に) `mq_nonjms=false` という行を含める必要があります。

JMS メッセージのタイプ

JMS 環境では、JMS バス上で異なるタイプのデータを送信できます。このコネクターは、3 つのデータ・タイプを認識します。

この 3 つのタイプとは、テキスト・メッセージ、バイト・メッセージ、およびオブジェクト・メッセージと呼ばれるものです。最も制約の少ないストラテジーはテキスト・メッセージを使用すること (例: `jms.usetextmessages=true`) であり、この場合は IBM Security Directory Integrator 以外のアプリケーションで、JMS コネクターにより生成されるメッセージを読み取ることができます。

JMS バスを介して他 IBM Security Directory Integrator サーバーと通信する場合、BytesMessage を使用すると、非常に単純な方法で項目オブジェクト全体を宛先に送信できます。これは、特に、テキストでは簡単に表せない特殊な Java オブジェクトが項目オブジェクトに含まれている場合にも便利です。ほとんどの Java オブジェクトは、ストリング表現を戻す toString() メソッドを提供しますが、その逆が可能なことは非常にまれです。また、toString() メソッドは必ずしも有用な情報を戻すとは限りません。例えば、バイト配列のストリング表現は、次のようになります。

```
"[B@<memory-address>"
```

テキスト・メッセージ

テキスト・メッセージはテキストの本文を伝達します。テキスト自体の形式は定義されていないため、事実上何でも構いません。ここに記載されている情報を通じて、テキスト・メッセージについて詳しく知ることができます。

このタイプのメッセージを送受信するとき、コネクターは、ユーザーがパーサーを指定しているかどうかに応じて、次の 2 つのいずれかを行います。

- パーサーを指定してある場合は、コネクターはそのパーサーを呼び出してテキスト・メッセージを解釈し、属性、ヘッダー、およびプロパティーを戻します。メッセージを送信するときは、指定した **conn** オブジェクトがパーサーに渡されて、テキスト本文部分が生成されます。したがって、さまざまな形式のデータを JMS バスに乗せて簡単に送信できます (例えば、LDIF パーサー、XML パーサーなどを使用)。また、Simple Object Access Protocol (SOAP) パーサーを使用して、JMS バスを介して SOAP 要求を送信することもできます。
- パーサーを定義していない場合は、テキスト本文はメッセージと呼ばれる属性に入れて戻されます。メッセージを送信するときは、コネクターは、指定されたメッセージ属性を使用して JMS テキスト本文部分を設定します。

```
var str = work.getString ("message");
task.logmsg ("Received the following text: " + str );
```

多様な形式のテキスト・メッセージ (XML、LDIF、CSV など) を受信することが予測される場合は、パーサー・パラメーターをブランクのままにしておき、テキスト・メッセージがどのような形式かを自分で判断する必要があります。形式が分かれば、次のように system.parseObject(*parserName*, *data*) 構文を使用して解析することができます。

```
var str = work.getString ("message");
// code to determine format
if ( isLDIF )
   e = system.parseObject( "ibmdi.LDIF", str );
else if ( isCSV )
   e = system.parseObject ( "ibmdi.CSV", str );
else
   e = system.parseObject ( "ibmdi.XML", str );
}
// Dump parsed entry to logfile
task.dumpEntry ( e );
```

Textmessage の使用フラグは、メッセージを送信するときにこのメソッドを使用するかどうかを、コネクターに指示します。

オブジェクト・メッセージ

オブジェクト・メッセージは、シリアライズ Java オブジェクトを含むメッセージです。シリアライズ Java オブジェクトは、特定の形式のバイト・ストリームに変換された Java オブジェクトで、これを使用することにより、受信側でのオブジェクトの復活が可能になります。ここに記載されている情報を使用することで、オブジェクト・メッセージを処理できるようになります。

テストの結果では、送信側と受信側の両方で JMS サーバーが Java クラス・ライブラリーを利用できる状態にあれば、この機能が正しく働くことが分かっています。一般に、java.lang.String オブジェクトは問題を起こすことはありませんが、他の Java オブジェクトについては問題が発生する可能性があります。このため、JMS コネクターでは、オブジェクト・メッセージの生成はしませんが、この種のメッセージを受信することはできます。オブジェクト・メッセージを受信すると、コネクターは次の 2 つの属性を戻します。

java.object
この属性は Java オブジェクトを保持するもので、このオブジェクトにアクセスするには、**作業**項目または **conn** 項目内で getObject メソッドを使用します。

java.objectClass
この属性は、Java オブジェクトのクラス名 (ストリング) を保持する簡易属性です。

```
var obj = work.getObject ("java.object");
obj.anyMethodDefinedForTheObject ();
```

受信されるのはこれらのメッセージのみです。

バイト・メッセージ

バイト・メッセージは、任意のバイト配列を伝達するメッセージです。ここに記載されている情報と例を使用することで、バイト・メッセージについて詳しく知ることができます。

JMS コネクターは、**Textmessage の使用**フラグが **false** の場合にこのタイプのメッセージを生成します。このコネクターは、指定された項目をシリアライズしてバイト配列にし、バイト・メッセージとしてメッセージを送信します。バイト・メッセージを受信すると、このコネクターは、まずバイト配列を非シリアライズして項目オブジェクトにしようとします。これが失敗した場合は、そのバイト配列がメッセージ属性に入れて戻されます。このオブジェクトにアクセスするには、**作業**項目または **conn** 項目内で getObject メソッドを使用する必要があります。

```
var ba = work.getObject ("message");
for ( i = 0; i < ba.length; i++)
   task.logmsg ( "Next byte: " + ba [ i ] );
```

このタイプのメッセージが生成されるのは、**Textmessage の使用**が **false** である (チェックされていない) 場合のみです。

イテレーター・モード

メッセージ・セレクターは、式が含まれているストリングです。ここに記載されている構文を使用することで、イテレーター・モードについて詳しく知ることができます。

式の構文は、SQL92 条件式構文のサブセットに基づいています。NewsTYpe プロパティーに値「Sports」または「Opinion」が設定されているメッセージを選択するメッセージ・セレクターの例を以下に示します。

```
NewsType = 'Sports' OR NewsType = 'Opinion'
```

ルックアップ・モード

このコネクターは、ルックアップ・モードをサポートします。このモードでは、JMS キュー内で一致するメッセージを検索することができます (ルックアップ・モードではトピック (Pub/Sub) はサポートされていません)。このセクションを参照することで、ルックアップ・モードについて詳しく知ることができます。

リンク基準は、キュー上の一致するメッセージを選択するための JMS ヘッダーおよびプロパティーを指定します。

拡張リンク基準の場合は、JMS 仕様 (http://java.sun.com/products/jms) で記述されているメッセージ選択仕様に従う必要があります。JMS コネクターは SQL フィルター仕様を再使用して (JMS メッセージ選択は SQL92 のサブセットです)、メッセージ選択ストリングを作成します。生成されたメッセージ・フィルター・ストリングを見るには、デバッグ・モードをオンにしてください。

ルックアップを行うには、基本的に 2 つの方法があります。

- キュー内で非破壊検索を行う (**QueueBrowser** を使用)。JMS キューからメッセージを削除せずに、一致するメッセージが戻されます。
- 一致するすべての項目を JMS キューから削除する。

いずれを使用するかを設定するには、コネクター構成内で**ルックアップの除去**フラグを設定します。トピック接続の場合は、トピック上のメッセージはサブスクライバーがそれを受信した時点で常に削除されるため、**ルックアップの除去**フラグは適用されません。ただし、ルックアップ・モードでは**永続サブスクライバー**・フラグに留意します。このフラグが設定されている場合は、JMS サーバーは、ユーザーが切断された後もトピックで送信されたメッセージをすべて保持します。

JMS コネクターでも、他のコネクターの場合と同様に、AssemblyLine 設定の中で、戻される項目の最大数を指定することができます。ルックアップで単一のメッセージのみが検索されるようにするには、AssemblyLine 設定の中で、**戻される重複項目の最大数 = 1** を指定します。**戻される重複項目の最大数**を **1** に設定すると、JMS キュー内に一致するメッセージがいくつあっても、一致する項目を一度に 1 つずつ検索することができます。

JMS バスは非同期であるため、JMS コネクターでは、ルックアップ機能がどの時点でメッセージ検索を停止するかを指示するためのパラメーターを使用できます。この種のパラメーターは 2 つあります。1 つは JMS キューを何回照会するかをコネクターに指示し、もう 1 つは、照会中に新しいメッセージを待つ時間を指示します。再試行回数に **10**、タイムアウトに **1000** を指定したとすれば、コネクターは

JMS キューを 10 回照会し、1 回ごとに 1 秒ずつ新規メッセージを待ちます。この時間内にメッセージが受信されない場合は、コネクターは戻ります。照会中にメッセージを受信した場合は、コネクターはさらに追加のメッセージの有無をチェックします (これにはタイムアウトはありません)。これは、キューからメッセージが戻されなくなるか、または、受信したメッセージの数が AssemblyLine で定義されている**戻される重複項目の最大数**の限界に達するまで続けられます。この方法には、ルックアップ操作により、その時点で使用可能なメッセージのみが検索されるという利点があります。

AddOnly モード

このモードでは、AssemblyLine の反復のたびに JMS コネクターから JMS サーバーに項目が送信されます。トピックを使用する場合はメッセージがパブリッシュされ、キューを使用する場合は、メッセージがキューに入れられます。

Call/Reply モード

ここに記載されている情報を使用して、Call/Reply モードで JMS コネクターを使用する方法を理解することができます。

このモードでは、コネクターは、**入力属性マップ**と**出力属性マップ**の両方があります。AssemblyLine がコネクターを呼び出すと、出力マップ操作が行われ、続いて入力マップ操作が行われます。JMS コネクターには、QueueRequestor クラスを使用する `queryReply()` というメソッドがあります。QueueRequestor コンストラクターには、非トランザクション QueueSession と宛先キューが渡されます。応答のための TemporaryQueue が作成され、`request()` メソッドが提供されます。このメソッドは要求メッセージを送信して応答を待機します。

JMS のヘッダーとプロパティー

JMS メッセージは、ヘッダー、プロパティー、および本文から構成されます。ヘッダーはプロパティーとは異なる方法でアクセスされ、旧バージョンではこれは使用できませんでした。現バージョンでは、ヘッダーとプロパティーをどのように扱うかを指定することができます。

JMS ヘッダー

JMS ヘッダーは、すべてのメッセージ内に存在する事前定義された名前付きの値です (値は NULL の場合もあります)。このコネクターは、以下に挙げる JMS ヘッダー名をサポートしています。

JMSCorrelationID
(ストリング) アプリケーションは、他のアプリケーションで使用するためにこのヘッダーを設定します。

JMSDeliveryMode
(整数) このヘッダーは JMS プロバイダーが設定するもので、配送モードを示します。

JMSExpires
(long 型) 値が 0 の場合は、メッセージの有効期限がないことを意味します。その他の値は、メッセージがキューから削除されるまでの有効期限を示します。

JMSMessageID

(ストリング) 固有のメッセージ ID。これは必須フィールドではなく、値は NULL でも構いません。

JMS プロバイダーはユーザー指定のメッセージ ID を使用しないことがあるため、コネクターは、メッセージを送信した後で **$jms.messageid** という特殊プロパティーを設定します。これは、ユーザーが常にそのメッセージ ID を使用できるようにするためです。この値を検索するには、**After Add** フックの中で `conn.getProperty("$jms.messageid")` を使用します。

JMSPriority

(整数) メッセージの優先順位。

JMSTimestamp

(long 型) メッセージが送信された時刻。

JMSType

(ストリング) メッセージのタイプ。

JMSReplyTo

(宛先) 送信側が応答の宛先として予期しているキュー/トピック。メッセージを受信するときは、この値はプロバイダー固有の宛先インターフェース・オブジェクトであり、通常これは内部のキュー・オブジェクトまたはトピック・オブジェクトです。メッセージを送信するときは、着信宛先オブジェクトを再使用するか、この値を有効なトピック/キュー名に設定する必要があります。この値が **NULL** (例えば、値のない属性) か、ストリング `"%this%"` である場合は、コネクターは独自のキュー/トピックを値として使用します。この方式と、キュー/トピックを明示的に設定する方式との違いは、ユーザーがコネクター構成のキュー/トピック名を変更した場合に、属性割り当てを更新する必要がないという点です。

現行バージョンには、制約事項が 1 つあります。それは、ユーザーが現在接続しているものと同じタイプの接続に対する応答しか要求できないという点です。つまり、トピックに関するメッセージを発行してキューへの応答を要求すること、またはその逆を行うことはできないということです。

このヘッダーに応答することは必須条件ではないため、メッセージの受信側がこのフィールドを完全に無視したとしても何の弊害も生じません。

上記のヘッダーはすべてプロバイダーが設定するもので、JMS ドライバーによって出力メッセージ用に処理される場合があります。構成画面では、すべてのヘッダーを属性として戻すことを指定することも、関心のあるヘッダーのリストを指定することもできます。すべてのヘッダーには、**jms.** という接頭部付きの名前が付けられます。また、JMS ヘッダー名は必ず **JMS** というストリングで始まるという点にも注意してください。したがって、**jms.JMS** で始まるプロパティー名は決して使用しないでください。このような名前は、ヘッダーとして解釈されてしまいます。

動作モードに応じて、JMS コネクターは、以下の追加プロパティーをその **conn** 項目に設定します。

messageType

このプロパティーは、読み取られたメッセージのタイプまたは書き込まれる

メッセージのタイプを保持します。その値は、「**メッセージ・タイプの選択**」構成パラメーターを上書きします。例を示します。

```
var messgeType = conn.getProperty("$jms.messageType ");
```

メッセージ

このプロパティーは、読み取られたメッセージを保持します。元のメッセージには、「**GetNext 後**」(イテレーター・モード)、「**ルックアップ後**」(ルックアップ・モード)、または「**CallReply 後**」(CallReply モード) の各フックからアクセスできます。

messageid

このプロパティーは、書き込まれたメッセージの ID を保持します。この ID には、AddOnly モードの「**追加後**」フックからアクセスできます。

JMS プロパティー

このコネクターの旧バージョンでは、項目オブジェクトと JMS メッセージの間ですべての JMS プロパティーがコピーされていました。現行リリースでは、ユーザー定義のすべてのプロパティーを属性として戻すことをコネクターに指示することも、関心のあるプロパティーのリストを指定することもできるため、この動作を詳細に設定できるようになりました。すべてのプロパティーには、他の属性と区別するために、**jms.** という接頭部が付けられます。プロパティーのリストをブランクのままにし、**属性としての JMS プロパティー**・フラグのチェックを外してある場合の動作は、旧バージョンの場合と同じになります。ユーザーは JMS ヘッダーと JMS プロパティーの両方を設定できます。後方互換モードを使用する場合は、次の例のように、**Before Add** フックの中で項目プロパティーを設定する必要があります。

```
conn.setProperty ( "jms.MyProperty", "Some Value" );
```

属性としての JMS プロパティー・フラグをチェックするか、プロパティーのリストを指定した場合は、JMS プロパティーを属性として指定する必要があります。そのための方法の 1 つは、**jms.** 接頭部を使用して属性を属性マップに追加することです。例えば、**jms.MyProperty** を属性マップに追加すると、**MyProperty** という名前の JMS プロパティーができることになります。

構成

このコネクターの名前は JMS Pub/Sub コネクターです。このコネクターにはここで提供されたパラメーターが必要です。

ブローカー

JMS サーバーの URL を指定します。このパラメーターを使用して、ActiveMQ、MQe、および ESB 初期設定ファイルを提供できます。

注:

1. ESB の値のフォーマットは、`hostname:port:sib_endpoint` です。
2. ActiveMQ ドライバーを使用する場合は、`vm://localhost` アドレスを使用します。 VM トランスポートでは、クライアントは、ネットワーク通信のオーバーヘッドなしで、VM 内で相互に接続できます。使用される接続はソケット接続ではなく、直接的なメソッドの起動であるため、ハイパフォーマンスの組み込みメッセージング・システムが実現します。

3. ActiveMQ JMS プロバイダーが IBM Security Directory Integrator サーバーの始動時にシステム・キューとして起動されない場合は、`vm://localhost?brokerConfig=xbean:etc/activemq.xml` ブローカー・パラメーターを使用して始動することができます。

サーバー・チャネル
MQ サーバー用に構成されたチャネルの名前。このパラメーターは、IBM WebSphere MQ Server とともに JMS コネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

SSL 接続を使用
SSL 接続に必要なパラメーターおよび構成設定を使用可能にします。

SSL サーバー・チャネル
SSL を使用して MQ サーバーにアクセスするために構成されたチャネルの名前。このパラメーターは、IBM WebSphere MQ Server とともに JMS コネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

キュー・マネージャー
MQ サーバー用に定義されたキュー・マネージャーの名前。IBM MQ 以外の場合は INITIAL_CONTEXT_FACTORY。

SSL CipherSuite
MQ サーバー・チャネルの構成時に選択された暗号に対応する暗号スイート名。このパラメーターは、IBM WebSphere MQ Server とともに JMS コネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

ユーザー名
JMS へのアクセス認証用のユーザー名。

パスワード
JMS へのアクセスを認証するためのパスワード。

接続タイプ
キューまたは**トピック** (トピックは **Pub/Sub** と呼ばれることもあります) のいずれに接続するかを指定します。

トピック/キュー
メッセージを交換する相手のトピック/キュー。

耐久トピック・サブスクライバー
トピック (Pub/Sub) **接続タイプ**のみに関連します。**true** の場合、コネクターは永続サブスクライバーを作成します。これは、コネクターがオフラインのときも、サーバーがトピックに関するメッセージを後で検索できるように保管しておくことを意味します。

クライアント ID
トピック接続に使用するクライアント ID (耐久の場合は必須)。

メッセージ選択フィルター
トピック/キューからメッセージを選択するためのメッセージ・フィルターを指定します。これはイテレーター・モードでのみ使用されます。

GetNext タイムアウト

イテレーター・モードにおいて、新規項目を待つ時間 (ミリ秒単位)。**-1** は**永久**を示します。

値がゼロの場合は、JMS コネクターはメッセージを即時に受信して戻ります。そのため、キュー/トピックにメッセージが存在しない場合、または読み取り操作が遅すぎる場合は、メッセージは受信されません。

JMS サーバー・タイプ

JMS サーバー・タイプを選択します。 JMS ドライバーのインターフェースをインプリメントするクラスのフルネーム。

特定のドライバー属性

name=value という形式のドライバー属性。以下に例を示します。

```
QUEUE_FACTORY_NAME=primaryQCF または
TOPIC_FACTORY_NAME=primaryTCF
```

JMS ドライバー・スクリプト

このパラメーターには、JMS プロバイダー固有オブジェクトの初期化に使用される JavaScript コードを指定します。このパラメーターの内容は、ハッシュ・テーブル・キー名「jsscript」を使用して構成済み JMS ドライバーに渡されます。このパラメーターは JMS スクリプト・ドライバーにより使用されます。JMS スクリプト・ドライバーは、このパラメーターの内容を Javascript として実行します。この「jsscript」という名前は、JMS スクリプト・ドライバーに渡されるハッシュ・テーブルのキーとして使用されます。JMS コネクターとともに使用するよう MQe または MQ ドライバーが構成されている場合は、このパラメーターの内容は無視されます。JMS スクリプト・ドライバーとは異なるサード・パーティーの JMS ドライバーが構成されている場合は、ほとんどの状況でこのパラメーターの内容は無視されます。

このパラメーターの JavaScript コードの構造と、このコードの実行環境については、「*インストールと管理*」のシステム・キューに関するセクション内の『JMS Script Driver』というラベルの付いたセクションおよび 346 ページの『システム・キュー・コネクター』を参照してください。

自動確認

true の場合は、このコネクターは自動的に各メッセージの確認を送ります。**false** の場合は、ユーザーが手動で JMS メッセージの受信を確認する必要があります (コネクターの acknowledge() メソッドを使用)。このパラメーターが**オフ**の場合は、JMS CLIENT_ACKNOWLEDGE モードが使用されます。

メッセージ・タイプの選択

このパラメーターを使用して、JMS バスに送信されるメッセージのタイプを指定します。メッセージは、TextMessage、BytesMessage、または ObjectMessage にすることができます。

属性としての JMS ヘッダー

true の場合、イテレーター・モードおよびルックアップ・モードでは、すべての JMS ヘッダーが属性として戻されます (**jms.** の接頭部が付けられます)。AddOnly モードでは、**jms.JMS** で始まる属性はすべて JMS ヘッダー

と見なされます。その結果、これらの属性は JMS ヘッダーとして設定され、メッセージの送信前に項目オブジェクトから削除されます。

注: 設定できるヘッダーはわずかであり、設定したとしても JMS プロバイダーが必ずそのヘッダーを使用するとは限りません。

特定 JMS ヘッダー
属性としての JMS ヘッダーと同様ですが、リストされる JMS ヘッダーがヘッダーとして扱われる点が異なります。ヘッダーは 1 行に 1 つずつ指定してください。

属性としての JMS プロパティー
true の場合、イテレーター・モードおよびルックアップ・モードでは、すべての JMS プロパティーが属性として戻されます (**jms.** の接頭部が付けられます)。AddOnly モードでは、**jms.** で始まる属性はすべて JMS プロパティーと見なされます。その結果、これらの属性は JMS プロパティーとして設定されます。

特定 JMS プロパティー
属性としての JMS プロパティーと同様ですが、リストされる JMS プロパティーがプロパティーとして扱われる点が異なります。プロパティーは 1 行に 1 つずつ指定してください。

ルックアップの除去
true の場合は、ルックアップで検出された各メッセージはキューから削除されます。

注: AssemblyLine 設定の中で**戻される重複項目の最大数**パラメーターを使用して、ルックアップで項目が 1 つのみ戻すようにできます。
false の場合は、メッセージは通常どおりに戻されますが、キューから削除はされません。

ルックアップ再試行回数
ルックアップ機能が、一致するメッセージを見つけるためにキューを検索する回数。

ルックアップ・タイムアウト
コネクターが、1 回のルックアップ照会で新しいメッセージを待つ時間 (ミリ秒)。このパラメーターを使用するのは、**ルックアップの除去**が **true** に設定されている場合のみです。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

トランザクション・セッションの使用
このパラメーターが `true` に設定されている場合、トランザクション・セッションが使用されます。コネクター・メソッドの rollback() および commit() は、このトランザクション内のすべてのメッセージをロールバックまたはコミットするために使用できます。コネクターが閉じられると、commit() は自動的に呼び出されます。このパラメーターが `true` に設定されている場合、「自動確認」パラメーターは無視されます。

「**パーサー...**」ペインからパーサーを選択できます。このペインでパーサーを選択するには、右下の「継承元」ボタンをクリックします。パーサーを指定すると、JMS テキスト・メッセージはこのパーサーを使用して解析されます。このパーサーは、JMS コネクターが受信したメッセージを処理し、また、JMS コネクターがメッセージを送信するときにテキスト・メッセージを生成するために使用されます。

例

このセクションに記載されている例を使用することで、JMS コネクターについて詳しく知ることができます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/SoniqMQ ディレクトリーにあります。

IBM Security Directory Integrator では、Sonic MQ の JMS スクリプト・ドライバーの例が提供されています。このサンプルでは、IBM Security Directory Integrator JMS コンポーネント (JMS コネクター、システム・キュー) で JMS プロバイダーとして SonicMQ サーバーを使用する方法を示します。

ディレクトリー *TDI_install_dir*/examples/was_jms_ScriptDriver で、 JMS コネクターおよび JMS スクリプト・ドライバーが組み込まれた WebSphere デフォルトの JMS プロバイダーの使用方法を示す例を確認できます。

外部システム構成

このコネクターがアクセスする外部 JMS システムの構成は、このコネクターに固有ではありません。このコネクターがアクセスする外部 JMS システムは、他の JMS クライアントに対して構成する場合と同様に構成されている必要があります。

IBM WebSphere MQ

IBM WebSphere MQ を JMS プロバイダーとして使用するには、以下の *jar* ファイルを IBM WebSphere MQ インストール・フォルダーから *TDI_install_dir*¥jars¥3rdparty¥IBM ディレクトリーにコピーします。

IBM WebSphere MQ v6.0 の場合

- com.ibm.mqjms.jar
- com.ibm.mq.jar
- jms.jar
- connector.jar
- dhbcore.jar
- jta.jar

IBM WebSphere MQ v7.0 の場合

- com.ibm.mqjms.jar
- com.ibm.mq.jmqi.jar
- jms.jar
- dhbcore.jar

IBM MQ への接続を受け入れる際には、以下のケースが考えられます。

1. ユーザー名とパスワードの両方が指定されている場合には、ユーザーは認証されます。匿名接続が許可されていない場合には、これが必要です。
2. ユーザー名のみが指定されている場合は、MQ サーバーは、ユーザーがホスト・システム上に存在して、mqm グループに属しているかどうかを検証します。
3. 資格情報が指定されていない場合は、匿名接続が試行されます。

そのため、IBM MQ と連携するように構成された JMS コネクターで資格情報を指定する際には、以下のオプションを使用できます。

- ユーザー名とパスワードの両方を指定する – 非匿名接続の場合はこれが必要です (ケース 1)。
- ユーザー名のみを指定する – ユーザー名の検証ではこれが必要です (ケース 2)。
- ユーザー名とパスワードをブランクのままにする – MQ ドライバーは、IBM Security Directory Integrator サーバーが実行されているのと同じコンテキストのユーザー名を取得して、検証のために送信します (ケース 2)。
- ユーザー名にシングル・スペースを指定し、パスワードを指定しない – 匿名接続用 (ケース 3)。

SSL の使用可能化

SSL (Secure Socket Layer) プロトコルを使用すると、MQ キュー・マネージャーとのセキュアな通信が可能になります。SSL プロトコルを使用可能にするには、MQ サーバーおよび JMS コネクターを IBM Security Directory Integrator 構成で調整する必要があります。以下のステップでは、サンプルのセットアップについて説明します。

IBM WebSphere MQ v6.0 および v7.0 用の SSL セキュリティーの構成

証明書の管理

GUI を使用してローカル・コンピューターで SSL 証明書を管理するには、IBM 鍵管理 (iKeyman) を使用します。

1. 鍵データベース・ファイルを作成します。

   iKeyman を開始し、「**鍵データベース・ファイル**」>「**新規**」を選択します。「鍵データベース・タイプ」は **CMS** である必要があります。ファイルの名前と場所を選択できますが、これらはキュー・マネージャーの鍵リポジトリー属性で後で設定する必要があることを覚えておいてください。「**ファイルに対してパスワードを隠しておく (Stash the password to a file)**」オプションをチェックし、ファイルへのアクセス時に使用するパスワードを指定します。
2. 証明書を取得します。

   CA (Certification Authority) の証明書を請求することもできますが、この例では自己署名証明書を使用します。「**作成**」>「**新規自己署名証明書**」を選択し、フォームを完了します。「鍵ラベ

ル」属性値の形式は
<ibmwebspheremq<aQueueManagerNameinLowerCase>>
(「ibmwebspheremqmyqueuemanager」の場合) とする必要があります。

3. 作成された証明書を今後使用するために抽出します。

   「**証明書の抽出**」ボタンを使用して、名前、場所、およびデータ・タイプを指定し、「**OK**」をクリックします。

キュー・マネージャーでの SSL の構成

この構成を行うために、IBM WebSphere MQ エクスプローラーを使用します。

キュー・マネージャーの鍵リポジトリーを設定します。

使用しているキュー・マネージャー **> 「プロパティー」 >** 「**SSL**」を選択し、「鍵リポジトリー」属性の値を変更します。この値は、『証明書の管理』セクションのステップ 1 での鍵データベース・ファイルの名前および場所にする必要があります。ただし、拡張子 .kdb は含めません。

SSL チャネルの構成

1. ***使用しているキュー・マネージャー*** **> 「拡張」 > 「チャネル」 >** ***使用しているチャネル名*** を選択します。右クリックして、「**プロパティー**」 **>** 「**SSL**」を選択し、SSL 暗号仕様を設定します (この例では「NULL_MD5」に設定)。これにより、メッセージ送信時に使用する暗号化方式およびハッシュ関数が指定されます。
2. 証明書を証明書の所有者の名前でフィルター処理します。

   証明書には、証明書の所有者の識別名 (DN) が含まれています。指定した値と一致する所有者の識別名 (DN) の属性を持つ証明書のみを受け取るようにチャネルを任意で構成することができます。これを行うには、「**これらの値と一致する識別名を持つ証明書のみを受け入れる**」チェック・ボックスを選択します。
3. キュー・マネージャーとの接続の開始側を認証します。

   別の相手がキュー・マネージャーへの SSL 対応接続を開始した場合、キュー・マネージャーは ID の証明として開始側に個人用証明書を送信する必要があります。開始側が独自の個人用証明書を送信しない場合はキュー・マネージャーで接続を拒否するように、キュー・マネージャーのチャネルをオプションで構成することもできます。これを行うには、チャネルのプロパティー・ダイアログの SSL ページで、「**接続開始側の認証**」リストの「**必須**」を選択します。この例では、この追加確認は必要はないので、「**オプション**」を選択します。

JMS コネクター用の SSL セキュリティーの構成

1. JMS コネクター構成の追加設定
   a. 「**SSL 接続を使用**」をチェックします。

b. 上記のステップ『SSL チャネルの構成』で構成した「SSL サーバー・チャネル」を指定します。

c. 使用するキュー・マネージャーを指定します。

d. 「**SSL CipherSuite**」プルダウン・リストから「**SSL_RSA_WITH_NULL_MD5**」オプションを選択します。

2. デジタル証明書を IBM Security Directory Integrator トラストストアに追加します。

この操作で iKeyman を再び使用します。

a. 証明書の追加

SSL 接続が確立されると、キュー・マネージャーによってその証明書が初期ハンドシェークの一環として送信され、受信した証明書を検証するために IBM Security Directory Integrator トラストストアが確認されます。証明書が検証されないと、接続は終了します。

既存のトラストストア・ファイル `testServer.jks` を編集するか、または IBM 鍵管理ツールを使用して新しい Java 鍵ストアを作成します。その後、コンボ・ボックスから「**署名者証明書**」オプションを選択し、「**追加**」をクリックします。ステップ 3 で抽出した証明書を保存した場所にブラウズし、それを選択します。ラベルに関するプロンプトが表示された場合は、ステップ 2 と同じラベル (ibmwebspheremqmyqueuemanager) を使用します。

「**接続開始側の認証**」オプションの「 **必須** 」を選択した場合は、独自の個人用自己署名証明書を IBM Security Directory Integrator 鍵ストアに作成し、それをキュー・マネージャーの鍵データベース・ファイルに署名者の証明書として追加する必要があります。これらのステップは、上記で示したステップと同じです。この新しい証明書は、コネクターによってキュー・マネージャーに SSL ハンドシェークの一環として送信され、証明書が存在しないと接続は終了されます。

前述のように、「**接続開始側の認証**」->「**オプション**」を選択した場合は、この手順は不要です。

b. `solution.properties` ファイルを変更します。

新しい鍵ストアを作成した場合、または既存の鍵ストアの場所を変更した場合は、`solution.properties` に記載する必要があります。例を示します。

```
javax.net.ssl.trustStore=C:¥¥Program Files¥¥IBM¥¥TDI¥¥V7.2¥¥jmsTrustStore
 javax.net.ssl.trustStorePassword=
 javax.net.ssl.trustStoreType=jks

 javax.net.ssl.keyStore=C:¥¥Program Files¥¥IBM¥¥WebSphere MQ¥¥Java¥¥bin¥¥jmsKeyStore
 javax.net.ssl.keyStorePassword=changeit
 javax.net.ssl.keyStoreType=jks
```

これらの変更は、IBM Security Directory Integrator を開始する前に行う必要があります。

追加情報:
「**SSL 接続を使用**」チェック・ボックスのチェック・マークを外した場合、以下のフィールドは保存された構成で保持されますが、以降の非 SSL 接続では使用されなくなります。

1. SSL サーバー・チャネル
2. キュー・マネージャー
3. SSL CipherSuite

「**SSL 接続を使用**」チェック・ボックスのチェック・マークが付いていない場合は、「**サーバー・チャネル**」に指定された値が使用されます。

IBM WebSphere MQ および JMS コネクターでの文字エンコードに関する考慮事項

複数の IBM WebSphere MQ サーバーを異なるプラットフォーム上にインストールしていると、文字セットの相違による問題が発生する場合があります。このような問題を解決する場合に考慮すべきキーポイントを以下に示します。

- JMS コネクターで JMS API の IBM MQ インプリメンテーションを使用します。これにより、クライアント側 (つまり、MQGET の実行時) での文字セット変換がデフォルトで使用可能になり、この動作を変更するインターフェースが提供されなくなります。いわゆるデータ変換は、メッセージの文字セットが宛先の文字セットと異なる場合に発生します。
- 明示的に設定していない場合は、インストール先のプラットフォームのデフォルトの文字セットがすべてのキュー・マネージャーで使用されます。例えば、Linux では UTF-8 になります。
- キュー・マネージャーにメッセージを挿入する場合は、キュー・マネージャーの文字セットと同じ文字セットでメッセージがエンコードされるようにします。これにより、ある文字セットでエンコードされたメッセージが、それとは異なる文字セットのメッセージを想定しているキュー・マネージャーに挿入されないようになります。
- 使用している IBM WebSphere MQ のバージョンが、使用している複数の文字セット間の変換をサポートしているかどうかを確認します。

具体的なシナリオに対するソリューションおよび次善策については、 223 ページの『トラブルシューティング』セクションを参照してください。

IBM WebSphere MQ でのデータ変換について詳しくは、以下の Web ソースを参照してください。

- http://www-01.ibm.com/support/docview.wss?uid=swg27005729&aid=1
- http://publibfp.boulder.ibm.com/epubs/pdf/csqzaw12.pdf
- http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzae05.pdf
- http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzak05.pdf
- http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC34658300

Lotus Expeditor Microbroker
既存の Lotus Expeditor Microbroker インストール済み環境がある場合は、それを JMS プロバイダーとして使用できます。ただし、Microbroker には

さまざまなバージョンおよびデプロイメントがあります。以下に、Microbroker を使用するために実行したいくつかのステップを例として挙げます。

1. IBM Security Directory Integrator で使用する場合は、「**JMS サーバー・タイプ**」(jms.driver) パラメーターに値 `com.ibm.di.systemqueue.driver.IBMMB` を使用します。パスワード・プラグイン用に JMS パスワード・ストアを構成する場合は、「**JMS サーバー・タイプ**」(jms.driver) パラメーターに値 `com.ibm.di.plugin.pwstore.jms.driver.IBMMB` を使用する必要があります。
2. 必要な .jar ファイルを ibmdisrv クラスパスに追加します。IBM Security Directory Integrator は、以下のバージョンの Microbroker jar でテストされています。
   - com.ibm.micro.client.nl_3.0.0.1-20081111.jar
   - com.ibm.micro.client_3.0.0.1-20081111.jar
   - com.ibm.micro.utils.extended_3.0.0.1-20081111.jar
   - com.ibm.micro.utils.nl_3.0.0.1-20081111.jar
   - com.ibm.micro.utils_3.0.0.1-20081111.jar
   - com.ibm.mqttclient.jms.nl_3.0.0.1-20081111.jar
   - com.ibm.mqttclient.jms_3.0.0.1-20081111.jar
   - com.ibm.mqttclient.nl_3.0.0.1-20081111.jar
   - com.ibm.mqttclient_3.0.0.1-20081111.jar
   - com.ibm.msg.client.osgi.jms_1.0.0.0.jar

     注: 以下の jar ファイルをこのファイルからアンパックする必要があります。
     - com.ibm.msg.client.commonservices.jar
     - com.ibm.msg.client.jms.jar
     - com.ibm.msg.client.provider.jar
     - com.ibm.msg.client.jms.internal.jar
   - com.ibm.msg.client.osgi.nls_1.0.0.0.jar

   注:

   a. Microbroker が異なるマシンにインストールされている場合、リストした jar ファイルを *TDI_install_dir*/jars フォルダーまたはそのサブフォルダーとは異なるローカル・フォルダーにコピーする必要があります。

   b. リストした jar ファイルにパックされた jar ファイルが含まれている場合は、これらの jar ファイルについてもアンパックし、IBM Security Directory Integrator サーバーのクラスパスに含める必要があります。
3. *TDI_install_dir*/jars/3rdparty/IBM/ibmjms.jar を IBM Security Directory Integrator サーバー (ibmditk) のクラスパスに追加します。

C:¥MB_jars フォルダーに必要な jar ファイルがすべて含まれていると想定した場合の、変更された ibmdisrv クラスパスの例を以下に示します。

```
"%TDI_JAVA_PROGRAM%" -classpath "%TDI_HOME_DIR%¥jars¥3rdparty¥IBM¥ibmjms.jar;
C:¥MB_jars¥com.ibm.msg.client.osgi.jms_1.0.0.0.jar;C:¥MB_jars¥
     com.ibm.msg.client.osgi.nls_1.0.0.0.jar;
C:¥MB_jars¥com.ibm.msg.client.commonservices.jar;C:¥MB_jars¥com.ibm.msg.client.jms.jar;
C:¥MB_jars¥com.ibm.msg.client.provider.jar;C:¥MB_jars¥com.ibm.msg.client.jms.internal.jar;
C:¥MB_jars¥com.ibm.micro.client.nl_3.0.0.1-20081111.jar;C:¥MB_jars¥
     com.ibm.micro.client_3.0.0.1-20081111.jar;
fC:¥MB_jars¥com.ibm.micro.utils.extended_3.0.0.1-20081111.jar;C:¥MB_jars¥
     com.ibm.micro.utils.nl_3.0.0.1-20081111.jar;
C:¥MB_jars¥com.ibm.micro.utils_3.0.0.1-20081111.jar;C:¥MB_jars¥
     com.ibm.mqttclient.jms.nl_3.0.0.1-20081111.jar;
C:¥MB_jars¥com.ibm.mqttclient.jms_3.0.0.1-20081111.jar;C:¥MB_jars¥
     com.ibm.mqttclient.nl_3.0.0.1-20081111.jar;
C:¥MB_jars¥com.ibm.mqttclient_3.0.0.1-20081111.jar;
%TDI_HOME_DIR%¥IDILoader.jar" %ENV_VARIABLES% com.ibm.di.loader.IDILoader com.ibm.di.server.RS %*
```

注: .jar ファイルの上記のリスト (1 つの長い行を表示上の理由から改行しています) は、ご使用の Microbroker バージョンとは異なる場合があります。必要な .jar ファイルのリストについては、Microbroker の資料を参照してください。

IBM WebSphere MQ Everyplace®

バンドルされた IBM WebSphere MQ Everyplace を JMS プロバイダーとして使用する場合は、IBM Security Directory Integrator のインストール後に追加の *.jar* ファイルをコピーする必要はありません。

トラブルシューティング

JMS コネクターのトラブルシューティングを行うときは、以下にリストする点について注意が必要です。

メッセージを交換する複数の IBM WebSphere MQ サーバーが異なるプラットフォームにインストールされているシステムでは、送信されたメッセージが受信時に壊れる場合があります。

例えば、2 つの MQ サーバーがあり、z/OS プラットフォーム上の MQ サーバーから、Linux プラットフォーム上の別の MQ サーバーにメッセージを送信する次のシナリオを想定します。Linux MQ サーバーから受け取ったメッセージが正しくない場合、z/OS プラットフォームと Linux プラットフォームのデフォルトの文字セットが異なることが原因で文字セット変換に問題がある可能性があります。このような問題に対処する場合に使用できるソリューションの一部を以下に示します (最も望ましいソリューションから降順で示しています)。

注: IBM Security Directory Integrator バージョン 7.2 以降では z/OS オペレーティング・システムはサポートされません。

1. z/OS MQ サーバーにメッセージを送信する前に、z/OS MQ キュー・マネージャーの文字セットを使用してメッセージをエンコードします。
2. 予期されるメッセージと同じ文字セットを使用するように、z/OS MQ キュー・マネージャーを構成します。
3. 正しい z/OS および Linux の文字セットに関して以下の次善策を使用します。
   a. 拡張マッピングを使用して **message** 属性をマップします。
   b. 以下のスクリプトを使用します。

      ```
      ret.value = new java.lang.String(conn.getString("message").getBytes(z/OS_charset), Linux_charset);
      ```

   c. 構成を実行します。

注: この次善策は、説明したシナリオの場合にのみ適用できます。3 つを超える MQ サーバーが存在するシステムでは、より複雑なメッセージ・デコードが必要となる場合があります。

JMS パスワード・ストア・コネクター

JMS パスワード・ストア・コネクターを使用して、MQ キュー・マネージャーおよび IBM WebSphere MQe キュー・マネージャーに接続します。

前のリリースでは、このコネクターは MQe パスワード・ストア・コネクターと呼ばれていました。現行バージョンの IBM Security Directory Integrator では、JMS パスワード・ストア・コネクターと呼ばれています。このように名前が変更されたのは、IBM Security Directory Integrator の JMS ドライバーのプラグ可能アーキテクチャーを使用できるようになったためです。これは、このコネクターが、MQe キュー・マネージャーだけでなく、IBM WebSphere WebSphere MQ キュー・マネージャーにもすぐに接続できることを意味します。さらに、接続の確立に JMS ドライバーを提供している限り、ユーザー指定のあらゆるキュー・マネージャーに接続できます。

JMS パスワード・ストア・コネクターでは、イテレーター・モードのみがサポートされています。

JMS パスワード・ストアでは、PKCS7 暗号化を使用してパスワード変更通知メッセージに署名して暗号化してから、このメッセージを JMS パスワード・ストア・コネクターに送信することができます。

注:

1. IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。
2. IBM Security Directory Integrator コンポーネントをデプロイし、MQe Mini-Certificate 認証アクセスを利用できます。これらの MQe 機能を使用するには、IBM WebSphere MQ Everyplace 2.0.1.7 以上と、IBM WebSphere MQ Everyplace Server Support ES06 をダウンロードしてインストールする必要があります。証明書認証アクセスを使用することで、匿名 MQe クライアントのキュー・マネージャーまたはアプリケーションが JMS パスワード・ストア・コネクターに対してパスワード変更要求を送信することを防止できます。
3. IBM MQ Everyplace では、IP バージョン 6 アドレッシングはサポートされていません。このため、JMS パスワード・ストア・コネクターが MQe にアクセスする際には、従来の IPv4 アドレスのみを使用できます。
4. IBM MQ Everyplace は、このバージョンの IBM Security Directory Integrator では非推奨であり、将来のバージョンで除去される予定です。適切な軽量のメッセージ・キューは、その時提供される予定です。

JMS パスワード・ストア・コネクターでは、複数のパスワード・ストアからのメッセージの受信がサポートされています。

コネクターのワークフロー

JMS パスワード・ストア・コネクターのワークフローを以下に示します。

1. JMS パスワード・ストア・コネクターは、ローカル MQe キュー・マネージャー上 (MQe JMS ドライバーを使用している場合) または外部キュー・マージャー上 (MQe JMS ドライバー以外を使用している場合) のいずれかの事前定義済みキューのメッセージを要求します。これらのメッセージは JMS インターフェースを使用して取得されます。
2. 取得されたメッセージが確認および/または暗号化解除されます (このステップはオプションです)。
3. メッセージが解析され、項目オブジェクトが作成されます。この項目オブジェクトの属性は、ユーザー ID、パスワード値、およびパスワード更新のタイプを表します。
4. この新規に作成された項目オブジェクトが IBM Security Directory Integrator AssemblyLine に渡されます。

JMS パスワード・ストア・コネクターは、初期化時に次のことを実行します。

MQe JMS ドライバーを使用する場合

- コネクターは、MQe キュー・マネージャーを開始し (MQe キュー・マネージャーがまだ開始されていない場合)、実行中のキュー・マネージャーに対する参照を取得します。
- ストレージ・コンポーネントへの接続を開始し、JMS パスワード・ストア・コネクターのキュー・マネージャーが稼働中であり、通知受信の準備が整ったことをストレージ・コンポーネントに通知します。

MQe JMS ドライバー以外を使用する場合

特定の JMS ドライバーが初期化され、外部キュー・マネージャーとの接続が確立されます。

コネクターは、パスワード更新メッセージの取得時に、次の 3 つのモードのいずれかで作動できます。

待機なし

QueueManager キュー内でパスワード更新メッセージを取得できるかどうかを検査します。**取得できる (yes)** の場合は、このモードはメッセージを検索して解析します。**取得できない (no)** の場合は、このモードでは **NULL** が戻され、イテレーターの終了が通知されます。

待機するミリ秒数

指定したミリ秒の間、パスワード更新メッセージが QueueManager キューに表示されるのを待機します。パスワード更新メッセージが表示されると、このモードはメッセージを検索して解析します。パスワード更新メッセージが表示されない場合は、このモードは **NULL** を戻し、イテレーターの終了を通知します。

永久に待機

コネクターは、QueueManager キューにパスワード更新メッセージが表示されるまで待機します。**NULL** が戻されることはありません。このモードで作動する場合は、コネクターを外部から停止する必要があります。

デフォルトでは、コネクターは QueueManager JMS キューから受け取るすべてのメッセージに対して自動的に確認を行います。ただし、「**自動確認**」パラメーターの選択を解除すれば、この動作を変更できます。そのように設定した場合は、メッセ

ージの確認通知を自分で出す必要があるため、AssemblyLine 内の適切な場所でコネクターの acknowledge() メソッドを呼び出してください。コネクターの acknowledge() メソッドを呼び出すたびに、その時点までにコネクターから引き渡されたメッセージすべてを確認できます。

累積メッセージの強制転送

ここに記載されている情報を使用して、MQe を使用する JMS パスワード・ストアからの累積メッセージの強制転送を実行する方法を知ることができます。

MQe ベースのパスワード・ストアの累算メッセージは、IBM Security Directory Integrator に自動的に転送されません。このような累算メッセージを強制転送するには、JMS パスワード・ストア・コネクターの「**ストレージ通知サーバー**」パラメーター、および JMS パスワード・ストアの「mqe.notify.port」パラメーターを使用します。

以下で説明します。

JMS パスワード・ストアを MQe で使用する場合、パスワード・ストア側のキュー・マネージャーと IBM Security Directory Integrator 側のキュー・マネージャーという 2 つの MQe キュー・マネージャーが関与します。パスワード・ストア側では、IBM Security Directory Integrator 側のローカル MQe キューを指す、リモート MQe キューが構成されています。

メッセージは、両方のキュー・マネージャーが作動可能である場合にのみ転送されます。IBM Security Directory Integrator が実行されていない場合は、パスワード・ストアのキュー・マネージャーで、到着するメッセージが累積されます。MQe では、通常、リモートのキュー・マネージャーが作動可能になったことを自動的に検出できません。そのため、Directory Integrator がオンラインに戻っても、累積されたメッセージは新しいメッセージがパスワード・ストアに到着するまで転送されません。

パスワード・ストアの累積メッセージを JMS パスワード・ストア・コネクターで「プル」できるようにする特別な機能があります。この機能は、コネクターの「**ストレージ通知サーバー**」パラメーターおよび JMS パスワード・ストアの「mqe.notify.port」パラメーターを使用して構成できます。コネクターは、初期化されると、累積メッセージの送信を開始することをパスワード・ストアに通知します。現時点では、「プッシュ」の代替機能はないことに注意してください。そのため、パスワード・ストアは、IBM Security Directory Integrator が実行中であるかどうかを定期的に確認することはしません。

メッセージ・セキュリティー

セキュリティー上の観点から見ると、JMS パスワード・ストア・コネクターは、次の 3 つのモードでメッセージを受け取ることができます。

- プレーン・テキスト・メッセージ

  メッセージは、JMS パスワード・ストアと JMS パスワード・ストア・コネクターとの間でプレーン・テキストとして転送されます。このため、メッセージ・ベースのセキュリティーは適用されません。
- IBM Security Directory Integrator 6.1.1 より前の PKI 暗号化メッセージ

この機能はオプションです。このオプションが使用されると、以下の操作のために .jks ファイルの証明書が使用されます。

– JMS パスワード・ストアによる受信メッセージの暗号化

– JMS パスワード・ストア・コネクターによるメッセージの暗号化解除

注: IBM Security Directory Integrator 6.1.1 以降、この暗号化は非推奨になりました。PKCS7 カプセル化には、暗号化など、メッセージをより安全に転送するための方法が用意されているためです。

- PKCS7 カプセル化メッセージ

  IBM Security Directory Integrator 6.1.1 以降、JMS パスワード・ストアと JMS パスワード・ストア・コネクターは PKCS7 をサポートするようになりました。これには、署名と暗号化の両方が含まれています。

カプセル化に PKCS7 を使用するかどうかはオプションです。デフォルトでは、オフになっています。PKCS7 を使用する場合、JMS パスワード・ストアと JMS パスワード・ストア・コネクターの両方を、PKCS7 を使用するように構成します。ただし、PKCS7 を使用すると、PKI 暗号化は許可されません。PKCS7 が暗号化をサポートしているためです。

PKCS7 暗号化のサポート

JMS パスワード・ストアでは、PKCS7 暗号化を使用してパスワード変更通知メッセージに署名して暗号化してから、このメッセージを JMS パスワード・ストア・コネクターに送信することができます。ここに記載されている情報を通じてこれについて詳しく知ることができます。

PKCS7 カプセル化の使用はオプションです。デフォルトではオフになっています。署名および暗号化は両方とも、機能するために証明書を必要とします。PKCS7 を使用する場合、古いバージョンの IBM Security Directory Integrator で使用可能な PKI ベースの古い暗号化メカニズムとの互換性はありません。

PKCS7 オプションを活動化すると、PKCS7 によって、受信した各メッセージの署名が、その署名者の証明書をトラストストア内の証明書と比較することで検証されます。これらが一致した場合、メッセージ・シグニチャーが検証されます。シグニチャーの検証が成功した場合、このコネクターではメッセージが受け入れられ、コネクター側の証明書の秘密鍵によってメッセージが暗号化解除されます。

注: PKCS7 を使用する必要がある場合、PKCS7 を使用するためのセットアップは JMS パスワード・ストア・コネクターと JMS パスワード・ストア (複数のストアが使用されている場合は、それらすべて) の両側で必要です。片側のみ PKCS7 を使用するように構成されている場合、エラーが発生します。
証明書は .jks ファイルに格納されます。コネクター側にも JMS パスワード・ストア側にも、それぞれ .jks ファイルがあります。

メッセージの署名

署名を使用して、メッセージの送信者が主張する正当性を検証することができます。

この検証では、JMS パスワード・ストア・コネクターにおいて、パスワード変更通知メッセージの送信側が実際に信頼できる JMS パスワード・ストアであることを検証する必要があります。

複数のパスワード・ストアから単一の JMS パスワード・ストア・コネクターにメッセージを送信することができます。この場合、コネクターを構成することによって、信頼できるパスワード・ストアそれぞれの公開鍵がこのコネクターの .jks ファイルに含まれるようにする必要があります。

メッセージの暗号化

暗号化は、パスワード・ストアでこのコネクターの公開鍵を使用してメッセージを暗号化することにより実現できます。コネクター側では、メッセージを暗号化解除するためにコネクターの秘密鍵を使用します。

証明書の管理

PKCS7 機能を処理するために .jks ファイルが必要です。このファイルには、JMS パスワード・ストア・コネクターの証明書だけではなく、このコネクターにメッセージを送信するすべてのパスワード・ストアの証明書も含まれている必要があります。証明書の管理については、ここに記載されている情報を通じて知ることができます。

JMS パスワード・ストア・コネクターの証明書は自己署名の個人証明書であり、この秘密鍵は、パスワード・ストアからのメッセージを暗号化解除するために使用されます。

パスワード・ストアの証明書は、信頼できる署名者証明書であり、各 JMS パスワード・ストアの .jks ファイルにあります。受信されるすべてのメッセージは、それに添付されている公開鍵と .jks ファイルにある有効なものと比較することにより検証されます。これらが一致した場合、メッセージ・シグニチャーが証明書に対して検証され、メッセージはコネクターの秘密鍵を使用して暗号化解除されます。

証明書の構造:

証明書は .jks ファイルに格納されます。コネクター側にもパスワード・ストア側にも、それぞれ対応する .jks ファイルがあります。これら 2 つの .jks ファイルには、PKCS7 を使用できるように、提供されたリストが含まれている必要があります。

JMS パスワード・ストア .jks ファイル

- 信頼できる署名者証明書としてのコネクターの公開鍵
- パスワード・ストアの公開鍵と秘密鍵の鍵ペア

JMS パスワード・ストア・コネクター .jks ファイル

- 信頼できる署名者証明書としての信頼できるパスワード・ストアそれぞれの公開鍵
- コネクターの公開鍵と秘密鍵の鍵ペア

証明書の作成:

ここに記載されている手順に従って、証明書の作成方法を理解することができます。

.jks ファイルの処理に使用する基本ツールは ikeyman.exe です。 ikeyman.exe は、IBM Security Directory Integrator とともに配布される JVM で使用可能なツールです。

このツールは、*TDI_install_dir*¥jvm¥jre¥bin にあります。ここで、*TDI_install_dir* は IBM Security Directory Integrator のインストール・ディレクトリーです。必要な鍵ストア/トラストストアの .jks ファイルを作成するには、以下のステップに従ってください。

1. **.jks ファイルの作成**

   新規 .jks ファイルを作成するには、「**鍵データベース・ファイル**」->「**新規**」をクリックし、名前およびファイル・パスを入力して JKS を選択します。パスワードを入力するように促されます。パスワードは覚えておきます。後でコンポーネントを設定するときに必要です。MQePasswordStore 用と JMS パスワード・ストア・コネクター用の少なくとも 2 つのファイルを作成する必要があります。
2. **証明書の作成**

   新規証明書を作成するには、証明書リスト上部のドロップダウン・メニューをクリックし、「**個人証明書**」を選択します。次に、「**新規自己署名...**」をクリックして、適切な情報を入力します。
3. **証明書の転送**

   最後のステップは、前のステップで作成した MQePasswordStore の JKS の自己署名証明書を JMS パスワード・ストア・コネクターの JKS に追加し、その逆の追加も行うことです。このために、証明書を DER バイナリー・データとして抽出する必要があります。「**証明書の抽出...**」をクリックし、「**データ・タイプ**」->「**DER バイナリー・データ**」を選択します。名前を入力して適切な場所にそれを保存し、もう一方の .jks ファイルを開きます。「**追加...**」をクリックし、DER 抽出データのファイルを探します (注: 必ず「**署名者証明書**」リストを選択してから、新規証明書を追加してください)。

注: IBM Security Directory Integrator の PKCS7 インプリメンテーションでは、.jks ファイルのセット以外のパスワードで保護された証明書はサポートされません。

例の使用法

ここに記載されている例を使用することで、JMS パスワード・ストア・コネクターの構成について知ることができます。

以下の例では、構成された JMS パスワード・ストア (「*パスワード同期プラグイン*」を参照) を処理するために、JMS パスワード・ストア・コネクターをどのように構成するかを示します。パラメーター「**PKCS7**」はチェックされています。つまり、PKCS7 暗号化/証明書オプションは使用可能にされています。

.jks ファイルへのパスであるパラメーター「**PKCS7 鍵ストア・ファイル**」は、C:¥dev¥di611_061025a¥certs¥mqeconnpkcs7.jks です。このファイルには、自己署名証明書だけではなく、JMS パスワード・ストアの信頼できる署名者証明書も含まれている必要があります (必要な証明書の作成の詳細については、 228 ページの

『証明書の作成』を参照してください)。この例の場合、パラメーター「**MQeConnector 証明書の別名**」は「*mqeconn*」と指定されています。

例で必要なため、2 つの .jks ファイル「mqepkcs7.jks」と「mqeconnpkcs7.jks」を作成します。手順は以下のとおりです。

1. iKeyman.exe を開き、「**鍵データベース・ファイル**」->「**新規...**」をクリックします。
2. ファイルの場所を選択します。上の例では、.jks ファイルを C:¥dev¥061025a¥certs に mqeconnpkcs7.jks という名前で保存しています。「**OK**」ボタンをクリックすると、パスワードを入力するように促されます。例の他のデータと互換性を保つため、パスワードとして「*secret*」と入力します。
3. 次のステップは、JMS パスワード・ストア・コネクターの証明書を作成することです。このため、ドロップダウン・メニューから「*個人証明書*」を選択し、「**新規自己署名...**」をクリックします。鍵ラベルは .jks ファイル内の証明書の別名です。「*mqeconn*」と設定します。その他のオプションは、デフォルト値のままでかまいません。
4. 前のステップで作成した自己署名証明書「mqeconn」を同じフォルダー C:¥dev¥di611_061025a¥certs に DER データとして抽出します。証明書に対応した名前 (例えば、mqeconn) を選択します。このファイルは後で、JMS パスワード・ストアの .jks ファイルに JMS パスワード・ストア・コネクターの証明書をインポートするために使用します。
5. 1 から 4 までのステップを繰り返します。ただし、.jks ファイルの場所は C:¥Program Files¥IBM¥DiPlugins¥mqepkcs7.jks にし、パスワードは同じ「*secret*」にします。 JMS パスワード・ストア証明書では、鍵ラベルの値は「*mqestore*」に設定し、同じディレクトリー C:¥Program Files¥IBM¥DiPlugins¥ に mqestore.der として抽出します。
6. 作成した両方の .jks ファイル間でそれぞれの証明書を交換する必要があります。 mqepkcs7.jks ファイルが開いているので、これに、mqeconnpkcs7.jks から抽出された DER バイナリー・データを先にインポートします。ドロップダウン・リストから「*署名者証明書*」を選択し、「**追加...**」をクリックします。表示されたポップアップ・ウィンドウで、データ・タイプとして「*バイナリー DER データ*」を選択し、.der ファイルが保存されている場所 C:¥dev¥di611_061025a¥certs を参照します。mqeconn.der ファイルを選択して、「**OK**」をクリックします。インポートされた証明書のラベルは必須です。JMS パスワード・ストアのプロパティー・ファイルのプロパティー「pkcs7MqeConnectorCertificateAlias」に、この値を指定する必要があるので、混乱を避けるため、もう一方の .jks ファイルのものと同じ別名、この場合は mqeconn にするのが賢明です。
7. 同じ手順を mqeconnpkcs7.jks ファイル (このコネクターの必須証明書を保持する鍵ストア) にも実行する必要があります。最初に、「**鍵データベース・ファイル**」->「**オープン...**」をクリックして、正確な場所を表示し、.jks ファイルを開きます。これまでの指示に正確に従っている場合、このファイルへのパスは C:¥dev¥di611_061025a¥certs になります。再度、パスワード入力のプロンプトが出されます。この後、ステップ 6 を新規パラメーターを使用して繰り返します。場所は C:¥Program Files¥IBM¥DiPlugins であり、証明書の名前は mqestore.der です。便宜上、名前は「*mqestore*」にします。このステップで、この例は完了です。

スキーマ

JMS パスワード・ストア・コネクターを操作するときは、ここに記載されているスキーマを使用できます。

JMS パスワード・ストア・コネクターでは、以下の固定属性構造 (スキーマ) の IBM Security Directory Integrator 項目オブジェクトが構成されます。

UserID
単一のストリング値が含まれます。

UpdateTypes
次のストリング値のいずれかが含まれます。

- **replace** (パスワード値の置換操作)
- **add** (パスワード値の追加操作)
- **delete** (パスワード値の削除操作)

Passwords
多値属性。各値は、パスワード値を表すストリングです。

Timestamp
パスワードが変更された時刻。`yyyyMMddHHmmss.SZ` という形式のストリングです。

CustomData
`pwsync.props` で定義されているカスタム・ストリング。

構成

ここに記載されているパラメーターを使用することで、JMS パスワード・ストア・コネクターを構成できるようになります。

GetNext タイムアウト
コネクターが、QueueManager キューにおけるパスワード更新メッセージの表示を待機する時間をミリ秒で指定します。**-1** は永久に待機することを指定し、**0** はメッセージを取得できない場合に即時に戻る (NULL を返す) ことを指定します。

ストレージ通知サーバー
JMS パスワード・ストア・コネクターからの通知を listen するストレージ・コンポーネント・サーバーを *host*:*port* の形式で指定します。ポートのデフォルト値は **41002** です。ホストは、パスワード・シンクロナイザーおよびストレージ・コンポーネントを配置するマシンの IP アドレスでなければなりません。

複数のストレージ・コンポーネント・サーバーが存在することがあります。その場合は、サーバーごとに別々の行に指定します。

ブローカー
JMS サーバーの URL。IPv6 アドレスを使用する場合は、このパラメーターに IPv6 JMS サーバー・アドレスと JMS サーバー・ポートの両方を指定する必要があります。このパラメーターを使用して、MQe 初期化ファイルを提供することもできます。

JMS サーバー・タイプ

JMS ドライバー・インターフェースをインプリメントするクラスのフルネーム。以下のいずれかの値を選択できます。

- IBMMQ
- IBMMQe
- ActiveMQ

特定のドライバー属性

name=value という形式のドライバー属性。以下に例を示します。

```
QUEUE_FACTORY_NAME=primaryQCF または
TOPIC_FACTORY_NAME=primaryTCF
```

サーバー・チャネル

MQ サーバー用に構成されたチャネルの名前。このパラメーターは、IBM WebSphere MQ Server とともに JMS パスワード・コネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

キュー・マネージャー

MQ サーバー用に定義されたキュー・マネージャーの名前。IBM MQ 以外の場合は INITIAL_CONTEXT_FACTORY。

ユーザー名

JMS へのアクセス認証用のユーザー名。

パスワード

JMS へのアクセスを認証するためのパスワード。

クライアント ID

キュー接続に使用するクライアント ID。

SSL 接続を使用

SSL 接続に必要なパラメーターおよび構成設定を使用可能にします。

SSL サーバー・チャネル

SSL を使用して MQ サーバーにアクセスするために構成されたチャネルの名前。このパラメーターは、IBM WebSphere MQ Server とともに JMS コネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

SSL CipherSuite

MQ サーバー・チャネルの構成時に選択された暗号に対応する暗号スイート名。このパラメーターは、IBM WebSphere MQ Server とともにコネクターを使用する場合にのみ適用されます。このパラメーターは、以前のバージョンとの互換性を確保する目的で構成に残されています。

自動確認

これをチェックした場合は、各メッセージが自動的に確認されます。チェックしない場合は、コネクターの `acknowledge()` メソッドを使用して手動でメッセージを確認する必要があります。デフォルトでは選択されています。

メッセージの暗号化解除
ストレージ・コンポーネントがパスワード更新メッセージを暗号化し、そのメッセージをコネクターが暗号化解除する必要がある場合は、このフィールドをチェックします。

鍵ストア・ファイル
パスワード・データの暗号化解除に使用する JKS ファイルのパス (「メッセージの暗号化解除」フィールドが選択されている場合にのみ考慮されます)。

鍵ストア・ファイルのパスワード
JKS ファイルのパスワード (「メッセージの暗号化解除」フィールドが選択されている場合にのみ考慮されます)。

鍵ストア証明書の別名
JKS ファイルからの鍵の別名 (「メッセージの暗号化解除」フィールドが選択されている場合にのみ考慮されます)。

鍵ストア証明書のパスワード
秘密鍵の取得に使用するパスワード。これを指定しない場合は、秘密鍵の取得に「**鍵ストア・ファイルのパスワード**」が使用されます (「メッセージの暗号化解除」フィールドが選択されている場合にのみ考慮されます)。

PKCS7
PKCS7 カプセル化を使用するかどうかを示します。デフォルト値は使用不可です。このパラメーターが使用可能になっている場合は、PKCS7 機能に関連する他のすべてのパラメーターを指定します。

PKCS7 鍵ストア・ファイル
JMS パスワード・ストア・コネクターの JKS の名前を含む絶対パスです。Windows プラットフォームでファイル・パスを指定する場合、単一スラッシュ「¥」の代わりにダブルスラッシュ「¥¥」を使用する必要はありません。

PKCS7 鍵ストア・ファイルのパスワード
JKS ファイルのパスワードのプレーン・テキスト値 (MQePasswordStore のプロパティーでは暗号化バージョンが必須)。

MQeConnector 証明書の別名
.jks ファイルに保存するときの、拡張子が付いていない、JMS パスワード・ストア・コネクター証明書の別名。

詳細ログ
詳細なログ・メッセージが必要な場合は、このフィールドをチェックします。

JMS ドライバー

JMS パスワード・ストア・コネクターには、IBM WebSphere MQ Everyplace ドライバーと IBM WebSphere MQ ドライバーという 2 つのドライバーがあります。このセクションを参照することで、これらのドライバーについて知ることができます。

IBM WebSphere MQ Everyplace ドライバー

ここに記載されている情報を使用することで、JMS プロバイダーを使用するためのパラメーターを構成および設定できるようになります。

JMS パスワード・ストア・コネクターの JMS プロバイダーとして MQe を使用するには、「**JMS サーバー・タイプ**」構成パラメーターに「IBMMQE」を設定し、`global.properties` または `solution.properties` の「systemqueue.jmsdriver.name」プロパティーに「com.ibm.di.systemqueue.driver.IBMMQe」を設定する必要があります。

IBM WebSphere MQ Everyplace ドライバーには以下の 1 つのパラメーターがあります。

- **mqe.file.ini** - このパラメーターの値は MQe 初期設定ファイルの絶対ファイル名にする必要があります。

例えば、MQe を使用するために JMS パスワード・ストア・コネクターの構成が必要な場合、以下の行を `global.properties` または `solution.properties` に含める必要があります。

```
systemqueue.jmsdriver.param.mqe.file.ini=TDI_install_folder/MQePWStore/pwstore_server.ini
```

これは MQe 構成ユーティリティーによって MQe 初期設定ファイルが作成されるデフォルトの場所です。

注: JMS パスワード・ストア・コネクターの JMS プロバイダーとして MQe を使用するには、MQe キュー・マネージャーが作成されている必要があります。これを作成するには、IBM Security Directory Integrator に組み込まれている MQe 構成ユーティリティーを使用します。詳しくは、「*インストールと管理*」の『MQe Configuration Utility』を参照してください。

IBM WebSphere MQ ドライバー

ここに記載されている情報を使用することで、JMS プロバイダーを使用するためのパラメーターを構成および設定できるようになります。

JMS パスワード・ストア・コネクターの JMS プロバイダーとして MQ を使用するには、「**JMS サーバー・タイプ**」構成パラメーターを「IBMMQ」に設定する必要があります。

IBM WebSphere MQ ドライバーには以下のパラメーターがあります。

- **ブローカー** (jms.broker) - MQ サーバーのアドレス (IP アドレスおよび TCP ポート番号)。この値の例は、「192.168.113.54:1414」です。
- **サーバー・チャネル** (jms.serverChannel) - MQ サーバー・インスタンス用に構成されたサーバー・チャネルの名前。
- **キュー・マネージャー** (jms.qManager) - MQ サーバー・インスタンス用に定義されたキュー・マネージャーの名前。
- **SSL Cipher Suite** (jms.sslCipher) - MQ サーバー・チャネルの構成時に選択された暗号に対応する暗号スイートの名前。この値の例は、「SSL_RSA_WITH_RC4128_MD5」です。
- **SSL 接続を使用** (jms.sslUseFlag) - MQ サーバー・インスタンスへの接続に SSL を使用するかどうかを指定します。有効な値は **true** および **false** です。

IBM WebSphere MQ サーバーの構成方法については、当該資料を参照してください。

関連情報

346 ページの『システム・キュー・コネクター』,
「*インストールと管理*」の
システム・キューに関するセクション、
「*パスワード同期プラグイン*」の
『JMS Password Store』、
205 ページの『JMS コネクター』.

JMX コネクター

JMX コネクターは、JMX 1.2 および JMX Remote API 1.0 仕様を採用しています。このコネクターでは、標準 JMX 機能のみを使用します。これについて詳しく知るためには、ここに記載されている情報を参照してください。

JMX コネクターでは、その構成に応じて、ローカル JMX 通知かリモート JMX 通知のいずれかを listen して報告できます。

AssemblyLine の開始時に JMX コネクターが初期化されます。このコネクターの初期化時に、コネクター・パラメーターに基づいてローカル通知かリモート通知のいずれを報告するかが判別されます (単一実行でローカル通知とリモート通知の両方を報告することはできません)。次に、コネクターは該当する MBean サーバーへのローカル参照またはリモート参照を取得し、コネクター・パラメーターに指定されている JMX 通知を登録します。

`getNextEntry()` メソッドで、コネクターは通知待機中に AssemblyLine をブロックします。通知を受信すると、コネクターの `getNextEntry()` メソッドから AssemblyLine に項目 (通知詳細が格納されている項目) が戻されます。

正常に完了した 2 つの `getNextEntry()` 呼び出しの間に受信された通知はバッファーに入れられます。これにより、通知が失われることが防止されます。`getNextEntry()` の呼び出し時にバッファーに通知が入っている場合は、コネクターは AssemblyLine をブロックせずに、最初にバッファーに入れられた通知を即時に戻します。

このコネクターは、イテレーター・モードのみで稼働します。

コネクター・スキーマ

ここに示す使用可能な属性 (入力属性マップ) を使用することで、JMX スキーマを理解することができます。

event.originator
: タイプ com.ibm.di.connector.JMXConnector の JMX コネクター・オブジェクト

event.type
: タイプ java.lang.String の通知タイプ

event.rawNotification
JMX コネクターが受け取るロー JMX 通知インスタンス (javax.management.Notification)。この通知をブロードキャストするコンポーネントが javax.management.Notification を拡張し、このサブクラスに何らかの追加データを含めた場合は、このプロパティーを使用してその追加情報を検索できます。

event.timestamp
タイプ java.lang.Long の通知タイム・スタンプ。通知が作成された時刻を表します。

event.sequenceNumber
通知シーケンス番号 (java.lang.Long)。ソース・オブジェクト内での通知のシーケンス番号を表します。これは、通知ソースのコンテキストにおいて特定の通知インスタンスを識別するシリアル番号です。通知モデルでは、通知が送信時と同じ順序で受信されることは想定されていません。このシーケンス番号は、受信した通知をソートする目的で使用できます。

event.message
通知のメッセージ (java.lang.String)。

event.mbean.objectName
登録または登録抹消された MBean のオブジェクト名 (javax.management.ObjectName)。このプロパティーが使用できるのは、event.type が JMX.mbean.registered または JMX.mbean.unregistered の場合のみです。 ObjectName は MBean 名 (および MBean 名のワイルドカード) を表します。ドメインとすべてのキーおよび値の組み合わせが固有でなければなりません。(つまり、MBean 名全体が固有でなければなりません。)

event.mbean.name
MBean オブジェクト名のストリング表現 (java.lang.String)。このプロパティーが使用できるのは、event.type が JMX.mbean.registered または JMX.mbean.unregistered の場合のみです。

event.userData
JMX 通知ユーザー・データ (java.lang.Object)。

event.source
この通知が最初に発生した MBean オブジェクト名 (javax.management.ObjectName)。

構成

ここに記載されているパラメーターのリストを使用することで、JMX コネクターを構成できるようになります。

モード このパラメーターは、JMX コネクターがローカル JMX 通知かリモート JMX 通知のどちらを listen するかを決定します。JMX Remote API 1.0 仕様に基づき、コネクターはリモート JMX 通知を登録して listen します。

このパラメーターの選択可能な値 (ドロップダウン・リスト) は、*リモート* と *ローカル* です。

「ローカル」を選択すると、コネクターはローカル Java 仮想マシン内の MBeanServer に登録されている MBean により発行された通知のみを listen します。

「リモート」を選択すると、コネクターは JMX Remote API 1.0 仕様に基づいてリモート JMX システムに接続し、そのリモート・システムの Java 仮想マシン内の MBean サーバーに登録されている MBean により発行された通知を登録します。

リモート JMX URL
このパラメーターは、「モード」パラメーターが「リモート」に設定されている場合にのみ有効です。これは、リモート JMX システムへの接続に使用される JMX URL です。正確には、この URL はリモート MBean サーバーの始動時に指定され、リモート・クライアントがこの MBean サーバーに接続するために使用します。

このパラメーターの値の例は、「service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector」です。

デフォルト値は「service:jmx:rmi://localhost/jndi/jmx」です。

すべての MBean を listen する
使用可能なすべての MBean をこのコネクターで登録するか (チェックした場合)、または「**listen する MBean**」コネクター・パラメーターに指定されている MBean のみを登録するか (チェックを外した場合) を指定します。デフォルトでは、このパラメーターはチェックされています。

listen する MBean
MBean オブジェクト名のリストを指定します。このとき、1 行にオブジェクト名を 1 つずつ指定します。このリストにより、コネクターが通知対象として登録する MBean が指定されます。MBean オブジェクト名が指定されていない場合 (リストが空の場合) は、すべての MBean から発行される通知が報告されます。1 つ以上の MBean 名が指定されている場合は、指定されている MBean から発行される通知のみが報告されます。

通知タイプ
JMX 通知のタイプ。Java クラスと混同しないでください。Java クラスは汎用通知オブジェクトの特性です。このタイプはブロードキャスター・オブジェクトにより割り当てられ、特定の通知の意味を伝達します。このタイプは、通知オブジェクトの「ストリング」フィールドとして指定されます。このストリングは、任意数のドットで区切られたコンポーネントとして解釈されます。これにより、通知タイプの命名にはユーザー定義の任意の構造を使用できます。

JMX コネクターが listen する JMX 通知のタイプを指定します。コネクターは、タイプが指定されていない通知を報告しません。 それぞれの通知タイプは 1 つずつ個別の行に入力してください。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

JMX コネクターは、接続に SSL プロトコルを使用できます。リモート JMX システムで SSL 接続のみが受け入れられる場合、トラストストアが適切に構成されてい

れば、JMX コネクターは SSL 接続を自動的に確立します。つまり、global.properties または solution.properties の javax.net.ssl.trustStore、javax.net.ssl.trustStorePassword、および javax.net.ssl.trustStoreType プロパティーに適切な値が設定されている必要があります。

関連情報

JMX に関する Wikipedia、
JMX のスタートアップ・ガイド、
JMX のチュートリアル、
JMX を使用した ITM による IBM Security Directory Integrator の管理 (Managing IBM Security Directory Integrator with ITM using JMX)。

JNDI コネクター

JNDI コネクターは、さまざまの JNDI サービスへのアクセスを提供します。ここに記載されている情報とリンクを使用して、これについてさらに詳しく知ることができます。

これは、*javax.naming* および *javax.naming.directory* パッケージを使用して、さまざまなディレクトリー・サービスを処理します。特定のシステムに到達するためには、そのシステム用の JNDI ドライバー (例えば、LDAP 用の com.sun.jndi.ldap.LdapCtxFactory など) をインストールする必要があります。ドライバーは、通常、1 つ以上の jar または zip ファイルとして配布されます。これらのファイルは、Java ランタイムから到達できる場所 (例えば、*TDI_install_dir*/lib/ext ディレクトリーなど) に置いてください。

このコネクターでは、属性レベルでデルタ・タグがサポートされています。つまり、AssemblyLine の前のコネクターから属性レベルでデルタ情報が提供されると、JNDI コネクターはこの情報を使用してターゲット JNDI ディレクトリーで必要な変更を行うことができます。

LDAP サーバーの照会のために JNDI コネクターを使用するとき、検索基準に一致する項目の数が、LDAP サーバーで設定されている最大制限値よりも大きい場合、*SizeLimitExceededException* が発生することがあります。この状態を回避するには、LDAP サーバーの最大結果制限値を増やすか、または java.naming.batchsize プロバイダー・パラメーターにサーバーの最大制限値よりも小さい値を設定します。java.naming.batchsize パラメーターの詳細については、http://java.sun.com/products/jndi/tutorial/ldap/search/batch.html を参照してください。

構成

JNDI コネクターでは、以下のリストに示すパラメーターが必要です。

JNDI ドライバー
JNDI ドライバーのクラス名 (JNDI ネーミング・ファクトリー)。

プロバイダー URL
接続のための URL。例えば LDAP ドライバーの場合は ldap://host です。

認証メソッド
LDAP 認証のタイプ。これは、次のいずれかです。

- **無名** - この認証メソッドが設定されている場合、クライアントが接続されているサーバーでは、そのクライアントがどのクライアントであるかを認識しません (クライアント情報を必要としません)。サーバーは、このようなクライアントが、非認証ユーザー用に構成されたデータにアクセスすることを許可します。ユーザー名が指定されない場合、コネクターは、自動的にこの認証メソッドを指定します。ただし、このタイプの認証が選択され、**ログイン・ユーザー名**と**ログイン・パスワード**が指定された場合、コネクターは、自動的に認証メソッドを単純に設定します。
- **単純**: **ログイン・ユーザー名**と**ログイン・パスワード**を使用します。**ログイン・ユーザー名**と**ログイン・パスワード**を指定しない場合は、無名と見なされます。SSL プロトコルを使用して LDAP サーバーと通信するようにコネクターが構成されている場合を除いて、コネクターは、完全修飾された識別名 (DN) およびクライアント・パスワードを平文で送信することに注意してください。
- **CRAM-MD5** - これは SASL 認証メカニズムの 1 つです。LDAP サーバーは接続に際して、特定のデータを LDAP クライアント (すなわち、このコネクター) に送信します。すると、クライアントは MD5 暗号化方式を使用して暗号化された応答をパスワードとともに送信します。その後、LDAP サーバーはクライアントのパスワードをチェックします。CRAM-MD5 がサポートされるのは LDAP v3 サーバーのみです。 のどのサポート対象バージョンでもサポートされません。
- **SASL** - クライアント (このコネクター) は、LDAP サーバーへの接続時に Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。このタイプの認証の稼働パラメーターは、「**追加のプロバイダー・パラメーター**」オプションを使用して指定する必要があります。例えば、DIGEST-MD5 認証をセットアップするには、以下のパラメーターを「追加のプロバイダー・パラメーター」フィールドに追加する必要があります。

  ```
  java.naming.security.authentication:DIGEST-MD5
  ```

  SASL 認証およびパラメーターについて詳しくは、http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html を参照してください。

  注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

ログイン・ユーザー名
プリンシパル名 (**ユーザー名**など)。

ログイン・パスワード
信任状 (**パスワード**など)。

SSL の使用
LDAP サーバーとの通信に Secure Sockets Layer (SSL) を使用します。

名前パラメーター
AssemblyLine 項目内のどのパラメーターを項目の名前に使用するかを指定

します。これは、追加、変更、および削除操作で使用され、読み取りまたは検索操作時に戻されます。このパラメーターを指定しない場合は、**$dn** が使用されます。

検索ベース
ディレクトリーの繰り返しの際に使用する検索ベース。識別名を指定します。ディレクトリーによっては、ブランク・ストリングを指定することができます。その場合は、デフォルトにより、サーバーについて設定されている処置がとられます。ディレクトリー・サービスによっては、そのディレクトリー内の有効な識別名を指定しなければならないこともあります。

検索フィルター
ディレクトリーの繰り返しの際に使用する検索フィルター。

検索範囲
データ・ソースの繰り返しの際に使用する検索範囲。指定できる値は、以下のとおりです。

subtree
検索ベースおよびその下位にあるすべてのレベルの項目を戻します。

onelevel
検索ベースの直下にある項目のみを戻します。

参照 LDAP サーバーが見つけた参照をどのように処理するかを指定します。指定できる値は、以下のとおりです。

- **follow** – 自動的に参照に従います。
- **ignore** – 参照を無視します。
- **throw** – 参照が見つかった時点で ReferralException をスローします。 これはエラー・フックの中で処理する必要があります。

追加のプロバイダー・パラメーター
プロバイダーに受け渡す追加のプロバイダー・パラメーターのリスト。それぞれの *parameter*:*value* は別々の行に指定してください。以下に例を示します。

```
java.naming.batchsize=100
```

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

変更操作の設定

ここに記載されているステップを使用することで、JNDI コネクターに変更操作値を設定できます。

JNDI コネクターでは、このコネクターが変更モードのときに**変更操作**値を設定できます。また、**変更操作**を設定する代わりに、単純なコネクター・インターフェースを使用して、属性値や属性を直接追加、除去、または置換することもできます。

変更操作を設定するための構成エディターは提供されていません。変更モードで次のインターフェースを使用して、JNDI コネクターの作業項目の各属性に操作値を手動で追加する必要があります。

di.com.ibm.di.entry.Attribute.setOper(char operation) 操作

di.com.ibm.di.entry.Attribute.ATTRIBUTE_DELETE
この定数は、指定された属性値を属性から削除します。

その結果、属性には元の値セットから指定された値セットを除いた値セットが含まれます。値が指定されない場合は、属性全体を削除します。属性が存在しないか、指定された値セットの一部またはすべてのメンバーが存在しない場合は、そのことが無視されて操作が成功するか、存在しないことを示す例外がスローされます。属性に少なくとも 1 つの値が必要な場合は、最後の値を除去すると、その属性も除去されます。

di.com.ibm.di.entry.Attribute.ATTRIBUTE_REPLACE
この定数は、属性を指定された値に置換します。

属性が既に存在する場合は、この定数はすべての既存の値を新たに指定された値に置換します。属性が存在しない場合は、この定数は属性を作成します。値が指定されない場合は、この定数は属性のすべての値を削除します。属性に少なくとも 1 つの値が必要な場合は、最後の値を除去すると、その属性も除去されます。これはデフォルトの変更操作です。

di.com.ibm.di.entry.Attribute.ATTRIBUTE_ADD
この定数は、属性に指定された値を追加します。

属性が存在しない場合は、この定数は属性を作成します。その結果、属性には指定された値セットと元の値セットの結合が含まれます。

変更インターフェースの呼び出し

ここに記載されている情報を使用することで、インターフェースの変更を呼び出すための様々な操作を実行できるようになります。

属性への値の追加:

ここに記載されているコード例とステップを使用することで、値を属性に追加することができます。

```
public void addAttributeValue(String moddn, String modattr, String modval)
```

例外をスローします。ここで、

- *moddn*: 属性値を追加する DN
- *modattr*: 値を追加する属性の名前
- *modval*: *modattr* に追加する値

例えば、`"cn=bob"` を `"cn=mygroup"` の **members** 属性に追加する場合は、次のメソッドを使用します。

```
thisConnector.connector.addAttributeValue("cn=mygroup","members","cn=bob");
```

基礎にある変更操作が失敗すると、例外がスローされます。

属性値の置換:

ここに記載されているコード例とステップを使用することで、属性の値を置換することができます。

```
public void replaceAttributeValue(String moddn, String modattr, String modval)
```

例外をスローします。ここで、

- *moddn*: 属性値を追加する DN
- *modattr*: 値を置換する属性の名前
- *modval*: *modattr* に追加する値

例えば、"cn=mygroup" の **members** 属性を "cn=bob" のみに置換する場合は、次のメソッドを使用します。

```
thisConnector.connector.replaceAttributeValue("cn=mygroup","members","cn=bob");
```

基礎にある変更操作が失敗すると、例外がスローされます。

属性の除去:

ここに記載されているコード例とステップを使用することで、属性を削除することができます。

```
public void removeAttribute(String moddn, String modattr)
```

例外をスローします。ここで、

- *moddn*: すべての属性値を除去する DN
- *modattr*: すべての値を除去する属性名

例えば、"cn=mygroup" の **members** 属性を除去する場合は、次のメソッドを使用します。

```
thisConnector.connector.removeAttribute("cn=mygroup","members");
```

基礎にある変更操作が失敗すると、例外がスローされます。

属性からの特定の属性値の除去:

ここに記載されているコード例とステップを使用することで、属性から特定の属性値を削除することができます。

```
public void removeAttributeValue(String moddn, String modattr, String modval)
```

例外をスローします。ここで、

- *moddn*: 属性値を除去する DN
- *modattr*: 値を除去する属性名
- *modval*: 指定した属性から除去する値

基礎にある変更操作が失敗すると、例外がスローされます。

変更操作

以下の表に記載されている値を使用することで、JNDI コネクターに変更操作値を設定できます。

変更操作は、変更要求ごとに設定できます。これにより、変更要求項目内のすべての属性に対する**変更操作**が適切な変更操作値に設定されます。プロパティー値とマッチングする変更操作値は、次のとおりです。

プロパティー値 (ストリング)	変更操作値
delete	di.com.ibm.di.entry.Attribute. ATTRIBUTE_DELETE
add	di.com.ibm.di.entry.Attribute. ATTRIBUTE_ADD
replace	di.com.ibm.di.entry.Attribute. ATTRIBUTE_REPLACE

このプロパティーは、次のスクリプトから **modOperation** プロパティーを設定することにより、コネクターの実行時にいつでも設定できます。

```
conn.setProperty("modOperation","delete");
```

注: このプロパティーは、上記で定義されたインターフェースの動作には影響しません。ただし、`di.com.ibm.di.entry.Attribute.setOper(char operation)` を使用すると、既存の**変更操作**設定が上書きされます。

更新モードおよび削除モードでのルックアップのスキップ

「ルックアップのスキップ」オプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。

JNDI コネクターでは、更新モードまたは削除モード時の「**ルックアップのスキップ**」一般オプションがサポートされます。正常に機能させるためには、「名前」パラメーター (例えば、LDAP の場合は $dn など) を指定する必要があります。

関連情報

JNDI の概要、
JNDI のチュートリアル、
JNDI の FAQ、
『LDAP コネクター』.

LDAP コネクター

LDAP コネクターは、さまざまな LDAP ベースのシステムへのアクセスを提供します。このコネクターは、LDAP バージョン 2 および 3 の両方をサポートしています。JNDI 接続の階層の上に構築されます。

このコネクターは IBM パスワード同期プラグインとともに使用できます。IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。

他のほとんどのコネクターと異なり、LDAP ディレクトリーにオブジェクトを挿入するときは、他のコネクターで指定する属性の他に、オブジェクト・クラス属性と **$dn** 属性も指定する必要があるという点に注意してください。次に示すコードをプロローグに挿入すると、後で使用できる **objectClass** 属性が定義されます。

```
// This variable used to set the object class attribute
var objectClass = system.newAttribute ("objectclass");
objectClass.addValue ("top");
objectClass.addValue ("person");
objectClass.addValue ("inetorgperson");
objectClass.addValue ("organizationalPerson");
```

これで、次の割り当てを使用することにより、LDAP コネクターは **objectclass** という属性を持つことができます。

```
ret.value = objectClass
```

person クラスがどのような属性を持つかを確認するには、 http://java.sun.com/products/jndi/tutorial/ldap/schema/object.html を参照してください。

これを見ると、更新または追加コネクター内で **sn** および **cn** 属性を指定する必要があることが分かります。

また、LDAP コネクターでは、識別名に対応する **$dn** 属性も必要です。**iuid** という名前の work オブジェクトの属性として属性マップ内に **$dn** を作成するには、次のようなコードを使用します。

```
var tuid = work.getString("iuid");
ret.value = "uid= " + tuid + ",ou=people,o=example_name.com";
```

注:

1. 項目の更新に加えて移動も行いたい場合以外は、更新モードでの変更には、**$dn** および **objectclass** の 2 つの特殊属性は含まれません。
2. ディレクトリーに接続できない場合は、構成の「**SSL の使用**」フラグが、ディレクトリーが要求している状態に合わせて設定されていることを確認してください。
3. ルックアップを行うときは、コネクター属性として **$dn** を使用し、識別名を使用してルックアップを行ってください。**$dn** と他の属性の両方を使用する単純リンク基準は指定しないでください。この場合、DN で「等しい」比較を使用すれば単純ルックアップを実行できます。
4. ある種のサーバーには、すべてのデータの選択ができないようにするサイズ制限パラメーターがあります。これが使用されている場合、イテレーターは最初の *n* 個の項目しか戻さないため、不便な場合があります。一部のサーバー (例えば、Netscape/iPlanet) では、ユーザーがマネージャーとして認証されていれば、このサイズ制限を超過することができます。
5. 1 回の実行でディレクトリー全体を戻すサーバー (例えば非ページ検索) の場合は、クライアント側でメモリーの問題を引き起こすことがよくあります。 250 ページの『LDAP コネクターでのメモリーの問題の取り扱い』を参照してください。
6. **コネクター・フラグ**に **deleteEmptyStrings** の値が含まれている場合は、LDAP コネクターは、各属性について空のストリング値を除去します。これにより、属性に値がまったく含まれなくなることがあります (例えば、空の値セット)。属性

に空の値セットがある場合は、変更操作により、その属性はディレクトリー内の項目から削除されます。空の属性は許可されないため、追加操作では組み込まれません。空でない場合、項目変更では属性値は置換されます。

7. ルックアップ・モードで「baselevel」検索範囲を使用して **rootdse** 検索を実行する場合、**objectClass** の値が * である (objectClass equals *) と指定するリンク基準を追加して、「検索ベース」フィールドをブランクのままにする必要があります。イテレーター・モードで同じ検索を実行するには、「検索ベース」をブランクのままにし、「検索フィルター」に「objectClass=*」を設定する必要があります。
8. ルックアップ・モードで「baselevel」検索範囲を使用して通常検索を実行する場合、指定した「検索ベース」に基づく有効なリンク基準を追加する必要があります (例えば、検索ベース: cn=MyName,o=MyOrganization,c=MyCountry、リンク基準: sn equals MySurName)。

modrdn 操作の検出と処理

ここに記載されている情報とリンクを使用することで、modrdn 操作を検出および処理できるようになります。

一部の変更ログ・コネクター (IDS 変更ログ・コネクターおよび Sun Directory 変更検出コネクター) では、*modrdn* 操作が基礎となる LDAP サーバーの変更ログで提供されるため、この操作を検出できます。この操作が検出された場合、変更ログ・コネクターによって項目が *modify* 操作でタグ付けされます。操作が modrdn である場合、変更ログ属性には「newrdn」属性が含まれます。 LDAP コネクターは、「newrdn」属性が存在するかどうかを modEntry メソッドで検出し、存在する場合は、ターゲットの $dn の rdn を新しい値で置換して、コンテキスト名前変更操作を実行します。

注: IBM Security Directory Integrator v7.0 より前のデルタ・モードでの LDAP 構成では、modrdn 操作は汎用として扱われ、処理されませんでした。現在では、この操作は *modify* として処理されます。また、現在の構成では、「newrdn」属性がある場合、$dn は名前変更されます。

構成

このコネクターでは、提供されたパラメーターが必要です。ただし、モードによっては、一部のパラメーターは使用不能であり、表示されません。

LDAP URL
接続の LDAP URL (ldap://*host*:*port*)。

ログイン・ユーザー名
サーバーへの認証に使用する識別名。

ログイン・パスワード
信任状 (パスワード)。

検索ベース
使用する検索ベース。識別名を指定します。ディレクトリーによっては、ブランク・ストリングを指定することができます。その場合は、デフォルトにより、サーバーについて設定されている処置がとられます。ディレクトリ

ー・サービスによっては、そのディレクトリー内の有効な識別名を指定しなければならないこともあります。デフォルト値は「<o=orgname>」です。

検索フィルター
ディレクトリーの繰り返しの際に使用する検索フィルター。このパラメーターは、イテレーター・モードでのみ使用されますが、スキーマ・ディスカバリーを支援するためにすべてのモードで表示されます。

「**検索フィルター**」フィールドの右にある「**...**」というマークのボタンを押すと、リンク基準フォームを記入および LDAP 検索フィルターを生成できる「リンク基準」ダイアログが表示されます。

「**追加**」ボタンをクリックすると、選択基準を作成する行が追加されます。「**いずれかと一致**」チェック・ボックスは、デフォルトの AND 式ではなく OR 式を生成します。これは、片方向ヘルパーであることに注意してください。構成に既存のものはすべて、生成された式で置換されます。

検索範囲
このパラメーターは、コネクターが AddOnly モードの場合は使用されません。指定できる値は、以下のとおりです。

subtree
検索ベースおよびその下位にあるすべてのレベルの項目を戻します。

onelevel
検索ベースの直下にある項目のみを戻します。

baselevel
検索ベースにより指定された項目のみを戻します。

デフォルト値は subtree です。

サイズ制限
検索または繰り返しの結果として戻す最大項目数。**0 = 無制限**です。

時間制限
項目の検索を行う最大時間 (秒数)。**0 = 無制限**です。

ページ・サイズ
これを指定した場合は、LDAP コネクターはページ・モードの検索を試行します。ページ・モードでは、ディレクトリー・サーバーは、1 つのチャンク内のすべての項目を戻すのではなく、特定数の項目 (ページと呼ばれる) を戻します。このオプションをサポートしていないディレクトリー・サーバーもあります。デフォルト値は 0 です。ページ・モードが使用不可であることを表します。

属性のソート
サーバー側のソートを指定するパラメーター。Netscape/iPlanet 4.2 では使用できません。

注: サーバー上のひずみが増加します。

認証メソッド
LDAP 認証のタイプ。これは、次のいずれかです。

- **無名** - この認証メソッドが設定されている場合、クライアントが接続されているサーバーでは、そのクライアントがどのクライアントであるかを認識しません (クライアント情報を必要としません)。サーバーは、このようなクライアントが、非認証ユーザー用に構成されたデータにアクセスすることを許可します。ユーザー名が指定されない場合、コネクターは、自動的にこの認証メソッドを指定します。ただし、このタイプの認証が選択され、**ログイン・ユーザー名**と**ログイン・パスワード**が指定された場合、コネクターは、自動的に認証メソッドを単純に設定します。
- **単純**: **ログイン・ユーザー名**と**ログイン・パスワード**を使用します。**ログイン・ユーザー名**と**ログイン・パスワード**を指定しない場合は、無名と見なされます。SSL プロトコルを使用して LDAP サーバーと通信するようにコネクターが構成されている場合を除いて、コネクターは、完全修飾された識別名 (DN) およびクライアント・パスワードを平文で送信することに注意してください。
- **CRAM-MD5** - これは SASL 認証メカニズムの 1 つです。LDAP サーバーは接続に際して、特定のデータを LDAP クライアント (すなわち、このコネクター) に送信します。すると、クライアントは MD5 暗号化方式を使用して暗号化された応答をパスワードとともに送信します。その後、LDAP サーバーはクライアントのパスワードをチェックします。CRAM-MD5 がサポートされるのは LDAP v3 サーバーのみです。IBM Security Directory Server のどのサポート対象バージョンでもサポートされません。
- **SASL** - クライアント (このコネクター) は、LDAP サーバーへの接続時に Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。このタイプの認証の稼働パラメーターは、「**追加のプロバイダー・パラメーター**」オプションを使用して指定する必要があります。例えば、DIGEST-MD5 認証をセットアップするには、以下のパラメーターを「追加のプロバイダー・パラメーター」フィールドに追加する必要があります。

  ```
  java.naming.security.authentication:DIGEST-MD5
  ```

  SASL 認証およびパラメーターについて詳しくは、http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html を参照してください。

  注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

SSL の使用
これをチェックすると、LDAP サーバーとの通信に Secure Sockets Layer が使用されます。

参照 LDAP サーバーが見つけた参照をどのように処理するかを指定します。指定できる値は、以下のとおりです。

- **follow** – 自動的に参照に従います。
- **ignore** – 参照を無視します。

- **throw** – 参照が見つかった時点で ReferralException をスローします。 これはエラー・フックの中で処理する必要があります。

コネクター・フラグ
特定の動作を使用可能にするフラグ。

deleteEmptyStrings
このフラグを使用すると、コネクターは、ディレクトリーを更新する前に、値として空ストリングのみを含んでいる属性を削除します。LDAP バージョン 3 サーバーを使用している場合は、属性の値を空ストリングにすることはできないため、必ずこのフラグを使用してください。

追加のプロバイダー・パラメーター
追加の JNDI プロバイダー・パラメーター。コロンで区切った *name*:*value* のペアを 1 行に 1 つずつ入れた形式を使用してください。

戻り属性
戻す属性のリスト (1 行に付き 1 属性)。このパラメーターを空のままにした場合は、すべての操作不可 (ユーザー) 属性が戻されます。操作可能属性 (modifyTimestamp など) を戻すには、操作可能属性を明示的にリストする必要があります。

バイナリー属性
バイナリーとして扱われる属性のリスト。1 行に 1 つずつ属性を入力してください。このパラメーターを指定しなかった場合は、デフォルトの属性リストが使用されます。デフォルトのリストは以下のとおりです。

- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedData
- thumbnailPhoto
- thumbnailLogo
- userPassword
- userCertificate
- authorityRevocationList
- certificateRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- objectGUID
- objectSid

注: 1 つの AssemblyLine が持つことのできるバイナリー属性のリストは 1 つのみです。1 つの AssemblyLine 内に複数の LDAP コネクターがあるときに、デフォルトのリストを変更する必要がある場合は、最後のコネクターで、その AssemblyLine 内のすべての LDAP コネクター用のバイナリー属性のリストを定義してください。

AD パスワードの自動マップ

LDAP を使用して、Active Directory におけるユーザーのパスワードを追加または更新するために使用します。これをチェックした場合は、LDAP パスワード (**userPassword** という名前である必要がある **conn** 属性) が別の名前 (**unicodePwd**) にマップされます。**unicodePwd** には、コネクターが変換を行う特殊な形式があります。

注: ADAM には不要です。

LDAP トレース・ファイル

LDAP BER パケットをファイルにトレースします。これはデバッグ時に便利です。

属性のソート

サーバー側のソートを指定するパラメーター。Netscape/iPlanet 4.2 では使用できません。

注: これによりサーバー上のひずみが増加します。

仮想リスト・ビュー・ページ・サイズ

繰り返しのために仮想リスト・ビューを使用します。これは一部のサーバーには効果がある可能性がありますが、テストの結果、一部のサーバー (例えば、Netscape/iPlanet 4.2) の速度が極度に低下することが示されています。ただし、メモリー不足が生じた場合の次善策としては利用できます。 250 ページの『仮想リスト・ビュー制御』も参照してください。

名前変更のシミュレート

サーバーで名前変更がサポートされていない場合に、**削除**および**追加**操作でシミュレートします。

属性の追加 (置換しない)

このオプションは、項目の変更時に LDAP コネクターのデフォルトの動作を変更します。

このチェック・ボックスをチェックすると、LDAP コネクターは **DirContext.ADD_ATTRIBUTE** 制約を設定します。このチェック・ボックスにチェックしない場合、LDAP コネクターは **DirContext.REPLACE_ATTRIBUTE** 制約を設定します。

LDAP 接続に **DirContext.ADD_ATTRIBUTE** 制約を設定すると、AssemblyLine で処理される任意の属性に新しい値が追加されます。したがって、この制約を慎重に使用しないと、項目に同じ値が繰り返し追加されてしまいます。また、対象の属性が一価である場合に、例外が発生することがあります。**DirContext.REPLACE_ATTRIBUTE** を設定すると、動作は旧 LDAP コネクターと同じになります (デフォルト動作)。つまり、属性のすべての値が作業項目の内容で置換されます。

運用属性の設定

このパラメーターを有効にすると、IBM Security Directory Server で運用属性の設定と変更を行うことができます。サーバーで名前変更がサポートされていない場合に、削除または追加操作でシミュレートします。

コメント

このパラメーターを使用して、コメントを追加します。

詳細ログ

このフィールドをチェックすると、追加のログ・メッセージが生成されます。

仮想リスト・ビュー制御

以下に示す情報とリンクを使用して、仮想リスト・ビュー制御を利用するために、Booster Pack をダウンロードし、Booster Pack で作業を行うことができます。

仮想リスト・ビュー・コントロールを IBM Security Directory Integrator で使用するには、Sun Microsystems から JNDI/LDAP Booster Pack をダウンロードしてください (http://java.sun.com/products/jndi/downloads/index.html)。Booster Pack をダウンロードしたら、IBM Security Directory Integrator を開始する前に、Booster Pack 内の「ldapbp.jar」を *TDI_install_dir*¥jars フォルダーにコピーする必要があります。仮想リスト・ビュー・コントロールを使用するが、「ldapbp.jar」が使用できない場合は、AssemblyLine が失敗し、該当するエラー・メッセージが表示されます。

LDAP コネクターでのメモリーの問題の取り扱い

サーバーには、検索結果の全体を一度に戻すものがあり (例えば非ページ検索)、その場合はしばしばメモリーの問題が発生します。ここに記載されている情報を使用して、このような問題に対処する方法を理解することができます。

この状況は、ユーザーには IBM Security Directory Integrator でメモリー・リークが起きているように見えることがありますが、これは、サーバーが次から次へと送り込んでくる項目を Directory Integrator が次々に処理しているために、そう見えるだけです。

Active Directory などの LDAP サーバーは、**ページ検索**拡張機能をサポートしており、ページ (一度に戻すオブジェクト数) を検索できます。これは、大規模な戻りセットを取り扱うための望ましい方法です (詳細については、**ページ・サイズ・**パラメーターを参照してください)。サーバーがページ検索をサポートしているかどうかは、LDAP コネクターの「構成」タブで、**ページ・サイズ・**パラメーターの右側にあるボタンをクリックすることにより、いつでも確認できます。

ページ・サイズ・パラメーターがサポートされていない場合は、クライアントはサーバーから送られてくる大量のデータに対してほとんど成すすべがなく、問題が発生する可能性があります。以下、この問題への対応策をいくつか紹介します。

- 仮想リスト・ビューを行うための**仮想リスト・ビュー・ページ・サイズ・**パラメーターを参照してください。この方法は、使用している LDAP サーバーによっては、効果がある場合とない場合があります。
- ディレクトリーがメモリーに収まるサイズのものであることが分かっている場合は、JavaVM が使用できるメモリーの量を増やすことができます。「」の付録『仮想マシンが使用できるメモリーの増加』を参照し、AIX® にデプロイされている LDAP コネクターに関する現行の問題に特に注意してください。
- この問題に対する一般的な解決策は、サーバー固有のユーティリティーを使用して、LDAP データベースを LDIF ファイルまたは他の形式のファイルにダンプし、ファイルまたは URL コネクターを使用してそのファイルの読み取りまたは繰り返しを行うことです。プロローグの中で (system.shellCommand によりコネクターが活動化される前に) コマンド行を開始して、LDIF エクスポートを生成

し、AssemblyLine がそのファイルを読み取れるようにできます。この方法は、実施が可能であれば効果的な解決策となります。大きいディレクトリー全体を繰り返すモードを使用している場合は、バッチ処理も考慮に入れてください。

- 場合によっては、IBM Security Directory Integrator を使用して、ディレクトリー検索をファイルにダンプすることもできます。これは、ファイルへの高速書き込みにより、IBM Security Directory Integrator が入力側の供給量に対応するのに十分な量のデータにアクセスできるためです (データの量と供給速度に応じて異なります)。AssemblyLine が 1 項目を処理するのに長時間を要する場合 (例えば、他のディレクトリーを更新している場合) は、早期に項目のあふれが生じることになります。ただし、このソリューションは時間への依存度が大きいため、他にもっと効果的な方法がある場合は使用しないようにしてください。

再接続機能の組み込みルール

ここに記載されている情報を通じて、再接続機能の組み込みルールを使用できます。

コネクターには、IBM Security Directory Integrator v6.1.1 フィックスパック 2 における再接続処理のために実装されたロジックがあります。
`javax.naming.ServiceUnavailableException` がメッセージにかかわらずスローされる場合、コネクター特有の組み込みルールによって再接続を実行できます。

v7.1 より前のバージョンの IBM Security Directory Integrator では、コネクターにループがあり、初期接続の確立が 10 回試行されていました。このループは一部のサーバーに関する問題に対処していましたが、サーバーがダウンした場合、初期接続確立の失敗により非常に時間がかかる場合があるという副次作用がありました。v7.1 以降、この「ループ」が再接続ルールに移行しました。この方法により、再接続試行の有無と、試行回数を指定できます。以前のバージョンとの互換性のため、初期再接続は使用可能になっています。

z/OS における SDBM バックエンドの検索

z/OS で SDBM バックエンドを検索するために LDAP コネクターを使用する場合、以下の点を考慮する必要があります。

注: IBM Security Directory Integrator バージョン 7.2 以降では z/OS オペレーティング・システムはサポートされません。

1. z/OS SDBM (LDAP) サービスでユーザー・プロファイルのリストを取得するために、イテレーター・モードで LDAP コネクターを使用する場合、デフォルトで DN 属性のみが戻されます。その他の属性は、入力マップに「*」属性が指定されている場合でも戻されません。これは LDAP コネクターの既知の制限事項です (最初から意図されたものではありません)。すべての属性を取得するには、AssemblyLine を構成することによって、最初に、イテレーター・モードで LDAP コネクターを使用して DN を取得し、続いて、その DN をリンク基準に使用して (つまり、リンク基準に「$dn EQUAL $$dn」を設定して)、ルックアップ・モードで LDAP コネクターを使用します。

注: 取得する DN の有効範囲を決定するため、イテレーター・コネクターの構成 (「構成」タブ ->「検索フィルター」) で「presence」フィルターを使用し、続いて、ルックアップ・モードの LDAP コネクターのリンク基準でも同値のフィルターを使用します。

2. イテレーター/ルックアップのフローの処理が z/OS 上の SDBM バックエンドで機能しない 3 つのユーザー・プロファイルがあります。
   - $dn 'racfid=irrmulti,profiletype=user,sysplex=sysb'
   - $dn 'racfid=irrsitec,profiletype=user,sysplex=sysb'
   - $dn 'racfid=irrcerta,profiletype=user,sysplex=sysb'

   これらのプロファイルに関しては、検索によりエラー「ICH30001I ユーザーが見つかりません (ICH30001I UNABLE TO LOCATE USER)」または「ICH31005I 検索基準に一致する項目がありません (ICH31005I NO ENTRIES MEET SEARCH CRITERIA)」が発生します。エラーが発生する理由は、これらのユーザーが、実ユーザーではないため、検索の対象に含めてはならないからです。SDBM バックエンドでは、大文字の要求を発行する「listuser」が実際には行われるため、これらのプロファイルは検出されません。これは予期された動作です。

LDAP コネクターのメソッド (API)

このセクションは、LDAP コネクターで使用可能なメソッドの一部を理解するために役立ちます。

完全な API リファレンスは JavaDoc 形式で用意されており、構成エディターの「**ヘルプ**」->「**ようこそ**」画面で、「**JavaDoc**」リンクを選択すると表示できます。

LDAP の比較

ここに記載されているコードを使用すると、LDAP を比較できます。

```
public boolean compare(String compdn, String attname, String attvalue)
  throws Exception
```

ここで、

- *compdn*: 属性を比較する DN。
- *attname*: 比較する属性の名前。
- *attvalue*: 比較の元となる *attvalue* の値。

値が等しい場合は、**true** が戻されます。値が等しくない場合は、**false** が戻されます。例えば、**cn=joe,o=ibm** の userpassword 属性が **secret** と等しいかどうか判別する場合は、メソッド `compare("cn=joe,o=ibm", "userpassword", "secret")` を使用します。

属性への値の追加

このメソッドを使用すると、指定された値を属性に追加できます。

```
public void addAttributeValue(String moddn, String modattr, String modval)
  throws Exception
```

ここで、

- *moddn*: 属性値を追加する DN。
- *modattr*: 値を追加する属性の名前。

- *modval*: *modattr* に追加する値。

例えば、**cn=mygroup** の **members** 属性に **cn=bob** を追加する場合は、メソッド `addAttributeValue("cn=mygroup", "members", "cn=bob")` を使用します。

基礎にある変更操作が失敗すると、java.langException がスローされます。

属性値の置換

このメソッドを使用すると、属性値を置換できます。

```
public void replaceAttributeValue(String moddn, String modattr, String modval)
  throws Exception
```

ここで、

- *moddn*: 属性値を置換する DN。
- *modattr*: 値を置換する属性の名前。
- *modval*: *modattr* において置換する値。

例えば、**cn=mygroup** の **members** 属性を **cn=bob** のみに置換する場合は、メソッド `replaceAttributeValue("cn=mygroup", "members", "cn=bob")` を使用します。

基礎にある変更操作が失敗すると、java.langException がスローされます。

属性値の除去

このメソッドを使用すると、指定された値を属性から削除できます。

```
public void removeAttributeValue(String moddn, String modattr, String modval)
  throws Exception
```

ここで、

- *moddn*: 属性値を除去する DN。
- *modattr*: 値を除去する属性の名前。
- *modval*: *modattr* から除去する値。

例えば、DN が **cn=mygroup** の **members** 属性から **cn=bob** という値を除去する場合は、メソッド `removeAttributeValue("cn=mygroup", "members", "cn=bob")` を使用します。

基礎にある変更操作が失敗すると、java.langException がスローされます。

すべての属性値の除去

このメソッドを使用すると、指定された属性のすべての値を削除できます。

```
public void removeAllAttributeValues(String moddn, String modattr)
  throws Exception
```

ここで、

- *moddn*: 属性値を除去する DN。
- *modattr*: すべての値を除去する属性の名前。

例えば、**cn=mygroup** の **members** 属性からすべての値を除去する場合は、メソッド `removeAllAttributeValues("cn=mygroup", "members")` を使用します。

基礎にある変更操作が失敗すると、java.langException がスローされます。

属性の追加または置換のデフォルト・アクション用の構成エディターのフラグ

ここに記載されている手順に従って、属性の追加または置換のためのデフォルト・アクションに対し、構成エディターでフラグを立てることができます。

LDAP コネクターの構成エディターには、「**属性の追加 (置換しない)**」というチェック・ボックスがあります。このオプションは、項目の変更時に LDAP コネクターのデフォルトの動作を変更します。

このチェック・ボックスをチェックすると、LDAP コネクターは `DirContext.ADD_ATTRIBUTE` 制約を設定します。このチェック・ボックスにチェックしない場合、LDAP コネクターは `DirContext.REPLACE_ATTRIBUTE` 制約を設定します。

LDAP 接続に `DirContext.ADD_ATTRIBUTE` 制約を設定すると、AssemblyLine で処理される任意の属性に新しい値が追加されます。したがって、この制約を慎重に使用しないと、項目に同じ値が繰り返し追加されてしまいます。また、対象の属性が一価である場合に、例外が発生することがあります。`DirContext.REPLACE_ATTRIBUTE` を設定すると、動作は旧 LDAP コネクターと同じになります (デフォルト動作)。つまり、属性のすべての値が作業項目の内容で置換されます。

このフラグを設定するのは、通常、グループを処理している場合です。**グループ** (属性) に**メンバー** (値) を追加する場合は、他のすべての値は削除しないはずです。

以前の動作では、属性が新規の値に置換されていました。この動作は、デフォルトとして残っています。

注: このプロパティーは、スクリプトから `addAttribute` プロパティーを設定することにより、コネクターの実行時にいつでも設定できます。次のようなコマンドを使用します。

```
work.setProperty("addAttribute", true)
```

注: このプロパティーは、上記で定義された `addAttributeValue` および `replaceAttributeValue` メソッドの動作には影響しません。

再バインド

このセクションを通じて、再バインド・メソッドについて知ることができます。

LDAP コネクターにある `rebind()` メソッドを利用すると、仮想ディレクトリーなどのソリューションや、着信する認証要求 (サポートされるいずれかのプロトコルを使用) を LDAP にマップするソリューションを容易に構築できます。詳しくは、Javadoc を参照してください。

更新モードおよび削除モードでのルックアップのスキップ

ここに記載されている情報を通じて、更新モードおよび削除モードでのルックアップのスキップについて知ることができます。

LDAP コネクターでは、更新モードまたは削除モード時の「**ルックアップのスキップ**」一般オプションがサポートされます。このオプションを選択した場合、実際の

更新操作および削除操作よりも前に検索が行われることはありません。正常に機能させるためには、$dn パラメーターを指定する必要があります。

関連情報

238 ページの『JNDI コネクター』,
9 ページの『Active Directory 変更検出コネクター』,
341 ページの『Sun Directory 変更検出コネクター』,
173 ページの『IBM Security Directory Integrator 変更ログ・コネクター』
418 ページの『z/OS LDAP 変更ログ・コネクター』,
LDAP に関する Wikipedia。

LDAP グループ・メンバー・コネクター

LDAP グループ・メンバー・コネクターを使用して、LDAP グループのメンバーを検索することができます。

このコンポーネントは、グループ項目自体ではなく、グループ・メンバーのユーザー項目を返します。プロパティーを使用して、含まれているグループおよび親/祖先グループの情報にアクセスできます。

LDAP グループ・メンバー・コネクターでは、イテレーター・モードがサポートされ、LDAP グループ・メンバーのユーザー項目が返されます。このコネクターでは、ネストされたグループもサポートされます。LDAP グループ・メンバー・コネクターは、 243 ページの『LDAP コネクター』を拡張します。

大きな Active Directory グループの処理

LDAP グループ・メンバー・コネクターは、大きな Active Directory グループを自動的に処理します。以下のセクションで、その仕組みについて知ることができます。

その仕組み

このコネクターは、Active Directory サーバーがグループ・メンバー・リストを反復するために必要な属性名構文を備えています。Active Directory がメンバー・リストのフラグメントを返す際に、返される属性名は、返されるメンバーの範囲でエンコードされます。例えば、グループ項目に 700 のメンバーが含まれていて、最初の読み取りで `member;range=0-499` 属性が返された場合は、`range` 部分は、属性値として返されるメンバーシップ・リストの部分を示します。LDAP グループ・メンバー・コネクターはこの情報を保持し、現在の範囲が完了すると、`member;range=500-999` 属性を要求することで、次のメンバー・バッチを処理します。メンバー・リストの最後のフラグメントを返す際に、属性名は、最終範囲について「`*`」文字でエンコードされます (例えば、`member;range=500-*`)。

LDAP グループ・メンバー・コネクターの繰り返し処理と同時にグループのメンバーが削除された場合、結果は予測不能となります。コネクターはグループ内の位置に基づいてメンバーを取得しますが、ユーザーの削除によって残りのユーザーの位置が移動するため、一部のメンバーが表示されなくなる場合があります。

LDAP グループ項目

LDAP グループ・メンバー・コネクターは、以下に示す基準に基づいて LDAP グループ項目を処理します。

- 項目には、member 属性、uniquemember 属性、または ibm-memberGroup 属性が含まれている必要があります。トピック 257 ページの『構成』の『**グループ・メンバー属性**』パラメーターを参照してください。
- LDAP グループ・メンバー・コネクターは、循環ネストを検出および処理するために、既に処理されたグループ項目のメモリー内キャッシュを保持します。
- 単一の反復処理で同じユーザー項目が複数回返されます。例えば、ネストされたグループに同じメンバーがある場合です (「**1 回だけユーザーを返す**」チェック・ボックスを選択している場合を除く)。

  注: デルタ機能を使用して、各メンバーが単一の反復で 1 回のみ返されるように、返されるユーザー項目をフィルターに掛けることができます。デルタ・モニタリングに含める必要がある属性は、$dn のみです。詳しくは、『デルタ機能』セクションを参照してください。

データ・ソース・スキーマ

イテレーター・モードでは、LDAP グループ・メンバー・コネクターは、接続された LDAP サーバーからユーザー項目を読み取ります。以下の表に示す情報を使用して、返される項目ごとのプロパティーを理解することができます。

データ・ソース・スキーマは、読み取り項目のオブジェクト・クラスに依存し、接続して項目を読み取ることで検出できます。例えば、入力マップまたは出力マップの「**接続**」ボタンおよび「**次を読み込む**」ボタンを使用します。イテレーター・モードでコネクターを操作する方法について詳しくは、「IBM Security Directory Integrator」の『イテレーター・モード』セクションを参照してください。

LDAP グループ・メンバー・コネクターは、返される項目ごとに、以下のプロパティーも返します。

プロパティー	説明
groupHierarchy	GroupEntry の配列 (以下のセクションで説明)。添字 0 は含まれているグループで、残りはネストの順序での祖先です。
group	IBM Security Directory Integrator 項目オブジェクトには、メンバー・リストの現在のグループが含まれます。グループ項目の ObjectClass に関係なく、返されるグループ・メンバーの識別名が設定された member 属性が含まれます。同期での使用に適するように、この項目には、デルタ操作コードのタグが付けられています。例えば、デルタ・モードの LDAP コネクターを使用して、グループ・メンバーのリストを増分的に更新できます。
groupEntry	現在反復されている、ユーザーの GroupEntry。

LDAP グループ・メンバー・コネクターによって返されるグループ項目オブジェクトには、以下のスキーマ定義があります。

```
public static class GroupEntry {
 Entry entry;
 String dn;
 Attribute groupMembers;
 int groupIndex;
 ArrayList<String> nestedGroups;

 public String getGroupDN();
 public Entry getGroupEntry();
 public Attribute getMembers();
 public boolean hasMoreMembers();
}
```

LDAP v3 スキーマについて詳しくは、http://www.ietf.org/rfc/rfc2256.txt を参照してください。

構成

以下に示すパラメーターを使用して、LDAP グループ・メンバー・コネクターを構成することがでます。

LDAP グループ・メンバー・コネクターは、 243 ページの『LDAP コネクター』に基づきます。そのため、LDAP コネクターの構成パラメーターは、LDAP グループ・メンバー・コネクターにも適用されます。LDAP グループ・メンバー・コネクターに固有の構成パラメーターについて、このセクションで説明します。

ネストされたグループの展開
このパラメーターを使用して、ネストされたグループを展開するか無視するかを指定します。

1 回だけユーザーを返す
このパラメーターを使用不可にすると、複数のグループのメンバーシップが読み取られるため、同じユーザーが複数回返されます。有効にすると、ユーザーは 1 回のみ返されます。

グループ・メンバー属性
このパラメーターを使用して、メンバーを含む属性の名前を指定します。値は名前のコンマ区切りのリストである必要があります。デフォルト値は `member,uniquemember,ibm-memberGroup` です。

関連情報

243 ページの『LDAP コネクター』

LDAP サーバー・コネクター

LDAP サーバー・コネクターは、構成でセットアップされている既知のポート (通常は 389) 上の LDAP クライアントから LDAP 接続要求を受け入れます。以下に示す情報を使用して、これについて詳しく知ることができます。

LDAP サーバー・コネクターは、サーバー・モードのみで作動し、自身のコピーを生成して、この接続が LDAP クライアントによってクローズされるまで、受け入れた接続を管理します。

このコネクターは IBM パスワード同期プラグインとともに使用できます。IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。

接続上で受け取った各 LDAP メッセージは、LDAP サーバー・コネクターのロジックを 1 サイクル実行します。メイン・スレッドは、他の LDAP クライアントからの同様の LDAP 要求の listen に戻ります。この時点で属性マッピングが行われ、LDAP.operation などの該当する属性が *work* オブジェクトにマップされます。

AssemblyLine の残りの部分が実行され、サイクルが応答チャネルまで到達すると、マップされた属性から戻りメッセージが構築されて、クライアントに送り返されます。LDAP 検索コマンドの場合は、ユーザーは **add** メソッドを呼び出して、クライアントに送信するデータ構造を構築します。LDAP サーバー・コネクターは、既存の接続上で次の LDAP コマンドの listen に戻ります。

LDAP 操作の値は、LDAP サーバー・コネクターの **conn** 項目内の *LDAP.operation* 属性に指定します。有効な値は、**SEARCH**、**BIND**、**UNBIND**、**COMPARE**、**ADD**、**DELETE**、**MODIFY**、および **MODIFYRDN** です。LDAP メッセージは、指定された LDAP 操作に対して多数の属性を提供します。

スクリプト記述

以下に示す情報を使用して、LDAP サーバー・コネクターのスクリプト記述について知ることができます。

AssemblyLine で LDAP サーバー・コネクターの後に続く部分では、LDAP メッセージが要求している結果を判断する作業を行う必要があります。基本的な LDAP 操作 (**SEARCH**、**BIND**、**UNBIND**、**COMPARE**、**ADD**、**DELETE**、**MODIFY**、および **MODIFYRDN**) は、LDAP サーバー・コネクターのスクリプト環境で値として指定して、スクリプト記述を容易にします。例えば、**LDAP.operation** を **BIND** にします。ユーザー・コードは、LDAP サーバー・コネクター内で **add (entry)** メソッドを呼び出して、検索結果項目をクライアントに送信します。項目は、有効な LDAP 属性名に特殊属性 **$dn** (項目の識別名) を付けた形式にする必要があります。

LDAP メッセージ戻り値の戻し

LDAP

AssemblyLine 内のユーザー提供のコードが各要求に応答するには、**ldap.status**、**ldap.matcheddn**、および **ldap.errormessage** 項目属性を設定します。**ldap.matcheddn** と **ldap.errormessage** はオプションです。

AssemblyLine の応答チャネル・フェーズで、LDAP サーバー・コネクターは*作業*項目の一部の属性をフォーマット設定して戻します。これらの属性を以下に示します。

- **LDAP.status**
- **LDAP.errormessage**

注: ストリングのみがサポートされます。**resultCode** は、デフォルトで **0** (成功) に設定されます。成功以外を示す **resultCode** は、ユーザーが個々に設定する必要があります。

エラー処理

受け取ったメッセージが LDAP v3 形式に準拠していない場合、LDAP サーバー・コネクターは接続を終了し、エラーを記録します。

注: LDAP サーバー・コネクターは、着信した属性に対していかなる検証も行いません。したがって、すべての操作またはパラメーター値が受け入れられます。

構成

ここに記載されているパラメーターを使用することで、LDAP サーバー・コネクターを構成できるようになります。

LDAP ポート
このコネクターが listen する TCP ポート。デフォルト値の中から 1 つ選択するか、独自のポート番号を指定します。

SSL の使用
これをチェックした場合、サーバー・コネクターは SSL 接続のみを受け入れます。

注: ソリューションのインプリメンテーションによっては、ポート番号の変更が必要な場合もあります。

文字エンコード
文字エンコード方式を指定します。デフォルトは **UTF-8** です。

バイナリー属性
バイナリーとして扱われる属性のリスト (バイナリー属性は、ストリングではなくバイト配列として戻されます)。1 行に 1 つずつ属性を入力してください。

注: 1 つの AssemblyLine が持つことのできるバイナリー属性のリストは 1 つのみです。1 つの AssemblyLine 内に複数の LDAP コネクターがあるときに、デフォルトのリストを変更する必要がある場合は、最後のコネクターで、その AssemblyLine 内のすべての LDAP コネクター用のバイナリー属性のリストを定義してください。

コメント
ユーザー自身が使用するためのコメント。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

関連情報

243 ページの『LDAP コネクター』

ログ・コネクター

ログ・コネクターは他のコネクターとは大きく異なり、ソース/ターゲット・システムについては意図されていません。ここに記載されている情報を使用することで、これについて詳しく知ることができます。

このコネクターは、IBM Security Directory Integrator ロギング機能への代替アクセス方法を提供する目的でのみ作成されています。詳しくは、「*インストールと管理*」の『Logging and debugging』を参照してください。

ログ・コネクターを使用すると、より単純な方法でロギング・ユーティリティーを使用でき、必要なスクリプトも少なくなります。このコネクターは AssemblyLine 内のどのポイントにも挿入でき、動的に使用可能/使用不可を切り替えることができます。このコネクターの導入前は、AssemblyLine のログ・オブジェクトを呼び出すスクリプト・コードを追加する必要がありました。 AssemblyLine のログ・オブジェクトを使用すると、このログ・オブジェクトに関連付けられているすべてのロガーにログ・メッセージが追加されます。同様に、AssemblyLine にロガーを追加 (AssemblyLine のロギング構成画面を使用) すると、IBM Security Directory Integrator の内部ロギングもログ出力チャネルにマージされ、不要なメッセージでログがいっぱいになる場合があります。ログ・コネクターを使用すると、ログ・チャネルに書き込まれるメッセージをすべて明示的に制御できます。

このコネクターは、AddOnly モードのみをサポートします。

スキーマ

ログ・コネクターのスキーマは、提供された定義済み属性のフラット構造になっています。

属性	説明
message	ログに記録されるメッセージ。
level	ログに記録されるメッセージのオプションのレベル。
exception	オプションの *java.lang.Exception*。

例外が存在する場合、レベル・パラメーターは無視され、`LogInterface.error(msg, exception)` が呼び出されます。

例外もレベルも存在しない場合、`LogInterface.info(msg)` が呼び出されます。

構成

ログ・コネクターのコネクター構成は、選択するロガー・コンポーネントに関連付けられている形式で表示されます。ここに記載されているパラメーターを使用することで、ログ・コネクターを構成できるようになります。

ロガーの選択

構成画面の「ロガーの選択」機能では、インストール済みログ・コンポーネントのフォルダーから、定義済みの構成を選択します。この構成は、実際のロガー・インスタンス化で使用するために、コネクター構成にコピーされます。有効なカテゴリーおよび選択項目は以下のとおりです。

Apache Log4J ロガー

これは昔から普及している、機能が豊富なカテゴリーです。

有効な Apache Log4J ロガーは以下のとおりです。

- ConsoleAppender
- CustomAppender
- DailyRollingFileAppender
- FileAppender
- NTEventLog
- FileRollerAppender
- SystemLogAppender
- SyslogAppender

Java Util ロガー

- カテゴリー・ベース構成
- FileHandler (java.util.logging)

JLOG ロガー

- カテゴリー・ベース構成
- FileHandler (com.ibm.log.FileHandler)

ロガーを定義する代わりに、標準装備のロギング機能と `LogConfigItem` 構成オブジェクトを使用することもできます。この方式を採用すると、Log4J による `log4j.properties` ファイルの検索を回避できます。詳しくは、813 ページの『追加ロガーの作成』を参照してください。

ロガー構成画面

以降の各セクションに示す情報を使用して、3 つのタイプのロガーについて詳しく知ることができます。

Apache Log4J ロガー:

Apache Log4j ロギング・ユーティリティーは IBM Security Directory Integrator に組み込まれています。Log4j では、IBM Security Directory Integrator で使用するロガーを構成するためにプロパティー・ファイルを使用します。以下に示す情報とリンクを使用して、これについて詳しく知ることができます。

Log4J API によってロガーを取得するには、`log4j.properties` ファイル内のロガー定義に一致するカテゴリー名が必要です。デフォルトでは、IBM Security Directory Integrator により、Log4J が*ソリューション・ディレクトリー* `/etc/log4j.properties` を使用するように構成されます。構成例については「*インストールと管理*」を参照してください。

レイアウト

ほとんどのロガーでは、出力ログのレイアウトを指定できます。以下のレイアウトを選択できます。

- パターン・レイアウト – パターンを使用して出力メッセージのフォーマットを設定します (以下の『パターン』を参照)。

- 単純レイアウト – ログ・レベルに続けて「 – 」および実際のログ・メッセージを出力します。
- HTML レイアウト – HTML テーブルでログを出力します。
- XML レイアウト – log4j.dtd (http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-files/log4j.dtd) に従ってログ・イベントを出力します。

パターン

多くのロガーでは、パターンを使用してログに出力する出力ストリングのフォーマットを設定します (Pattern.ConversionPattern)。このパターンのフォーマットについては、それぞれのロギング・ユーティリティーの資料に記載されています。これらのパターンで使用可能な役立つ変換文字をいくつか、以下のリストに抜粋します。

パターンとは、計算される値によって置換される特殊な構成が含まれたストリングです。パーセント記号 (%) の後に以下のいずれかの文字を使用すると、計算された値が挿入されます。

文字	効果
m	呼び出し元からのログ・メッセージの出力に使用されます。
c	ロギング・イベントのカテゴリーの出力に使用されます。このカテゴリー変換指定子の後には、大括弧で囲んだ 10 進定数である精度指定子をオプションで指定できます。
d	ロギング・イベントの日付の出力に使用されます。この日付変換指定子の後には、中括弧で囲んだ日付形式の指定子を追加できます。例えば、%d{HH:mm:ss,SSS} や %d{dd MMM yyyy HH:mm:ss,SSS} などです。日付形式の指定子が指定されていない場合は、ISO8601 形式であると見なされます。
F	ロギング要求が実行されたファイル名の出力に使用されます。
n	プラットフォームに固有の行末文字。
p	ログに記録されるイベントの優先度。
t	ログ・イベントを生成したスレッドの名前 (例えば、AssemblyLine 名など)。
%	単一の % 記号を出力します。

カテゴリー・ベース構成:

以下の情報を使用して、カテゴリー・ベース構成について知ることができます。

カテゴリー・ベース構成のロガーでは、ロガーの作成が Log4J によって代行されます。 Log4j は、ファイル `log4j.properties` を使用して一致するカテゴリーを検索し、プロパティー・ファイルの定義に従ってロガーを返します。

カテゴリー
使用するカテゴリーを入力します。デフォルトは空です。

ConsoleAppender:

コンソール Appender は、標準出力/エラーのストリームに書き込みます。パラメーターは以下のとおりです。

レイアウト
Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。

- パターン (デフォルト)
- 単純
- HTML
- XML

パターン
ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

CustomAppender:

1 つ以上の Java プロパティーにカスタム Appender が定義されている場合に、カスタム Appender を使用することができます。

`custom.appender.` で開始されるプロパティーには、`com.ibm.di.log.CustomAppenderInterface` をインプリメントする Appender クラスを指定する値があると予想されます。パラメーターは以下のとおりです。

Appender パラメーター
パラメーターの形式は未定義であり、このテキスト・フィールドを構文解析する Appender のインプリメンテーションに依存します。デフォルトで、このフィールドは空です。

DailyRollingFileAppender:

日次ローリング・ファイル Appender は、毎日ログ・ファイルを交替します。以下に示すパラメーターを使用して、この処理を実行することができます。

出力ファイルはローリングされると、ベース名と日付パターン・ストリングで構成された名前 (例: `filename.yyyy-mm-dd`) が付与されます。パラメーターは以下のとおりです。

ファイル・パス
基本ログ・ファイルのパス名を指定します。デフォルト値は空です。

ファイルに付加
このボックスをチェックするとログの初期化時にファイルに追加され、チェックを外すと上書きされます。デフォルトではチェックは外されています。

日付パターン
ログ・ファイル名の接尾部になる日付パターンを指定し、それによってロールオーバーの発生時間 (例: ファイル名の変更時間) を制御します。以下の定義済みの値から選択するか、またはユーザー独自の値を指定できます。

- .'yyyy-MM

- '.'yyyy-MM-dd
- '.'yyyy-MM-dd-HH
- '.'yyyy-MM-dd-HH-mm
- '.'dd-MM-yyyy

デフォルト値は '.'yyyy-MM-dd

レイアウト

Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。

- パターン (デフォルト)
- 単純
- HTML
- XML

パターン

ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

文字エンコード

出力文字のエンコードを指定します (例: UTF-8)。デフォルト値は空です。

FileAppender:

ファイル Appender は、出力ファイルにメッセージを書き込みます。以下に示すパラメーターを使用できます。

ファイル・パス

ログ・ファイルのファイル・パスを指定します。デフォルト値は空です。

ファイルに付加

このボックスをチェックするとログの初期化時にファイルに追加され、チェックを外すと上書きされます。デフォルトではチェックは外されています。

レイアウト

Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。

- パターン (デフォルト)
- 単純
- HTML
- XML

パターン

ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)

- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

文字エンコード
出力文字のエンコードを指定します (例: UTF-8)。デフォルト値は空です。

NTEventLog:

NT イベント・ロガーは、Windows NT イベント・ログにメッセージを書き込みます。以下に示すパラメーターを使用できます。

ソース 通常はロギングを実行しているアプリケーションのタイトルです。デフォルト値は「itdi」です。

レイアウト
Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。
- パターン (デフォルト)
- 単純
- HTML
- XML

パターン
ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。
- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

FileRollerAppender:

ファイル・ローラーは、1 から**バックアップ・ファイル数**までの順序番号を使用して毎日ログを交替します。以下に示すパラメーターを使用できます。

ファイル・パス
このロガーは、<ファイル・パス>.1 に書き込み、指定されたバックアップ・ファイル数になるまで既存のログ・ファイルの拡張子を増分します。デフォルト値は空です。

バックアップ・ファイル数
最も古いログ・ファイルを削除する前の、保持するファイルの数を指定します。デフォルト値は 5 です。

レイアウト
Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。
- パターン (デフォルト)
- 単純
- HTML
- XML

パターン

ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

文字エンコード

出力文字のエンコードを指定します (例: UTF-8)。デフォルト値は空です。

SystemLogAppender:

システム・ログ Appender は、system_logs/{ConfigId}/{AL,EH}_X (ここで、X は実行中の AL/EH の名前) で検出されたログ・ファイルに書き込みます。以下に示すパラメーターを使用できます。

パターン

ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

文字エンコード

出力文字のエンコードを指定します (例: UTF-8)。デフォルト値は空です。

SyslogAppender:

Syslog Appender は、syslogd デーモンに書き込みます。 syslogd デーモンは、ほとんどの UNIX システムでの標準ロギング・ユーティリティーです。以下に示すパラメーターを使用できます。

ホスト名 IP アドレス

syslog デーモンのホスト名または IP アドレスを指定します。デフォルト値は、127.0.0.1 です。

Syslog 機能

ログに記録されたメッセージに使用する機能名を指定します。使用可能な値は以下のいずれかです。

- kern
- user
- mail
- daemon
- auth
- syslog
- lpr
- news

- cron
- authpriv
- ftp
- local0
- local1
- local2
- local3
- local4
- local5
- local6
- local7

デフォルト値は、**local7** です。

機能ストリングを出力

機能の出力が必要な場合にチェックします。デフォルトでは、これはチェックされています。

レイアウト

Appender のレイアウトを選択します。使用可能な値は以下のいずれかです。

- パターン (デフォルト)
- 単純
- HTML
- XML

パターン

ログのフォーマットを定義する置換マスクを指定します。「レイアウト」が「パターン」の場合にのみ使用します。以下の定義済みのパターンから選択するか、またはユーザー独自のパターンを指定できます。

- %d{ISO8601} %-5p [%c] - %m%n (デフォルト)
- %d{HH:mm:ss} %p [%t] - %m%n
- %p [%t] %c %d{HH:mm:ss,SSS} - %m%n

Java Util ロガー:

これらのロガーは、標準 Java VM の一部です。 Java Util ロガーのレイアウトはフォーマッターと呼ばれています。 IBM Security Directory Integrator では、提供されたフォーマッターがサポートされています。

- 単純 – 人間が読めるフォーマットでログ・イベントの要約を出力します。通常、この要約は 1 行または 2 行になります。
- XML – Java ロギング API によって指定されたフォーマットでログ・ファイルを出力します (DTD の説明については、付録 A を参照してください)。

カテゴリー・ベース構成:

カテゴリー・ベース構成のロガーでは、ロガーの作成が Java Util ロギングによって代行されます。 JUL は、その `lib/logging.properties` ファイルを使用して一致するカテゴリーを検索し、プロパティー・ファイルの定義に従ってロガーを返します。

カテゴリー
使用するカテゴリーを入力します。デフォルトは空です。

FileHandler:

ファイル Appender は、出力ファイルにメッセージを書き込みます。「制限」パラメーターによってファイル・サイズの上限が設定されている場合、ログ・ファイルは最大サイズに到達すると交替されて、新規ファイルが作成され、ロギングは継続します。以下に示すパラメーターを使用できます。

ファイル名
ログ・ファイル名のパターンを指定します (java.util.logging.FileHandler を参照してください)。

付加 このボックスをチェックすると、ログの初期化時にロガーによりファイルに追加されます。デフォルトではチェック・マークが外されていて、既存のファイルに上書きされます。

フォーマッター
使用するフォーマッターを選択します。使用可能な値は以下のいずれかです。

- 単純
- XML

制限 ロギングが交替されるまでのファイルの最大サイズを指定します。デフォルトは空であり、サイズが無限であることを表します。

カウント
ログの交替が発生した後に保持するファイルの数を指定します。デフォルトは空であり、交替はせず、1 つのファイルのみを使用することを意味します。

JLOG ロガー:

JLOG ロガーは IBM Security Directory Integrator に組み込まれています。 JLOG のレイアウトはフォーマッターと呼ばれています。 IBM Security Directory Integrator では、提供されたフォーマッターがサポートされています。

- 単純
- CBE101Formatter – Common Base Event v1.0.1 XML のエントリーとしてログ・イベントをフォーマットします。
- EnhancedFormatter – 以下の例のように、ログ・イベントの各フィールドを個別の行に出力してログ・イベントをフォーマットします。

```
Date:    1999.07.16  11:20:56.842
Class:   com.ibm.log.samples.LogSample
Method:  messageSample
```

カテゴリー・ベース構成:

カテゴリー・ベース構成のロガーでは、ロガーの作成が JLOG によって代行されます。ここに記載されている情報を使用することで、これについて詳しく知ることができます。

JLOG は、その `jlog.properties` ファイルを使用して一致するカテゴリーを検索し、プロパティー・ファイルの定義に従ってロガーを返します。

カテゴリー
使用するカテゴリーを入力します。デフォルトは空です。

FileHandler:

ファイル Appender は、出力ファイルにメッセージを書き込みます。以下に示すパラメーターを使用できます。

ファイル・パス
ログ・ファイルのファイル・パスを指定します。デフォルトは空です。

ファイルに付加
このボックスをチェックすると、ログの初期化時にロガーによりファイルに追加されます。デフォルトではチェック・マークが外されていて、既存のファイルに上書きされます。

フォーマッター
このロガーに使用するフォーマッターを選択します。使用可能な値は以下のいずれかです。

- CBE101
- 拡張
- 単純

Lotus Notes コネクター

Lotus Notes コネクターについて知るには、以下に示すリンクを使用できます。

結合された Lotus Notes セクションの 100 ページの『Lotus Notes コネクター』 を参照してください。

メールボックス・コネクター

メールボックス・コネクターは、インターネット・メール・ボックス (POP3 または IMAP) へのアクセスを提供します。メールボックス・コネクターは、AddOnly、イテレーター、ルックアップ、更新、削除の各モードで使用できます。

メールボックス・コネクターは、使用頻度の高いヘッダーには事前定義済みの属性名を使用します。これより多くの情報が必要な場合は、**mail.message** プロパティーを使用してネイティブ・メッセージ・オブジェクトを検索してください。

初期化時に、コネクターはサーバーのメールボックスから取得可能なメール・メッセージをすべて取得し、内部コネクター・バッファーに格納します。後でコネクターは、`getNextEntry()` を呼び出すたびに (つまり反復のたびに) 1 つずつメッセー

ジを取得します。バッファーのメッセージをすべて取得した後の動作は、パラメーター「**ポーリング間隔**」により制御されます。 275 ページの『構成』を参照してください。この点は、このコネクターの以前のインプリメンテーションと異なります。

IMAP プロトコルが指定されている場合は、メールボックス・コネクターはサーバーのメールボックスでのメッセージの追加と削除についての通知を登録します。コネクターはメッセージがメールボックスに追加されたという通知を受信すると、このメッセージをコネクターの内部バッファーに追加します。コネクターはメッセージがメールボックスから除去されたという通知を受信すると、このメッセージをコネクターの内部バッファーから除去します。

Addonly を除くサポートされているすべてのモード (イテレーター、更新、ルックアップ、削除) において、構成でフォルダーが指定されていない場合、メールボックス・コネクターはメールボックス内のすべてのフォルダーで繰り返されます。その他の場合として、指定したフォルダーのサブフォルダーで繰り返すオプションがあります。両方の場合において、メールボックスを参照するときに除外するフォルダーをコンマ区切りのリストで指定できます。

注:

1. 1 ユーザー ID に付き 1 接続のみサポートされます。ユーザーが、スキーマ・タブを使用しているときに切断するのを忘れ、その後で AssemblyLine を実行すると、接続拒否エラーが生じることになります。
2. メールボックス・コネクターは、拡張リンク基準をサポートしていません (「」の『拡張リンク基準』を参照)。

スキーマ

以下に示す情報を使用して、メールボックス・コネクターのスキーマについて知ることができます。

入力マップ

メールボックス・コネクターは、以下に示す事前定義済みの属性およびプロパティーを使用します。これらの属性およびプロパティーは、入力マップにあります。

mail.from
送信者ヘッダー

mail.to **宛先** (受信者) ヘッダー

mail.cc CC 宛先ヘッダー

mail.replyto
返信先メール・アドレス

mail.subject
サブジェクト・ヘッダー

mail.messageid
メッセージ ID ヘッダー

mail.messagenumber
メッセージの内部番号

mail.sent
メッセージが送信された日付

mail.received
メッセージが受信された日付

mail.body
1 パーツ・メッセージの場合、この属性にはメッセージ本文が含まれます。

mail.bodyparts
複数パーツ・メッセージの場合、この属性には javax.mail.Part オブジェクトが含まれます。

mail.message
これは、項目に戻されるメッセージを表す javax.mail.Message です。

mailbox.message
これは、項目に戻されるメッセージを表す `javax.mail.Message` です。これは、**mail.message** に格納されているオブジェクトと同一です。この属性により、廃止されたメールボックス EventHandler との互換性が確保されます。

mail.originator
コネクター・オブジェクト。

event.originator
コネクター・オブジェクト。これは、**mail.originator** に格納されているオブジェクトと同一です。この属性により、廃止されたメールボックス EventHandler との互換性が確保されます。

mail.session
Java セッション・オブジェクト (javax.mail.Session)。

mailbox.session
Java セッション・オブジェクト (javax.mail.Session)。これは、**mail.session** に格納されているオブジェクトと同一です。この属性により、廃止されたメールボックス EventHandler との互換性が確保されます。

mail.store
メッセージ・ストア・オブジェクト (javax.mail.Store)。

mailbox.folder
フォルダー・オブジェクト (javax.mail.Folder)。これは、**mail.folder** に格納されているオブジェクトと同一です。この属性により、廃止されたメールボックス EventHandler との互換性が確保されます。

mail.operation
mail.message に関連した操作。POP3 接続の場合は、*既存* 項目のみが報告されます。 IMAP 接続の場合、このプロパティーの値は *new* または *deleted* になります。

mailbox.operation
mailbox.message に関連した操作。 これは、**mail.operation** に格納されているオブジェクトと同一です。この属性により、廃止されたメールボックス EventHandler との互換性が確保されます。

出力マップ

メールボックス・コネクターは、作業項目 (出力属性マップ) から以下の属性を取得します。

mail.from
送信者ヘッダー

mail.to **宛先** (受信者) ヘッダー

mail.subject
サブジェクト・ヘッダー

mail.messageid
メッセージ ID ヘッダー

mail.messagenumber
メッセージの内部番号

mail.addMessage
オブジェクト `javax.mail.Message` または `javax.mail.Message[]` のホルダーであり、AddOnly モードにおける指定したメールボックス・フォルダーへのメッセージの追加に使用されます。

Flag.Answered
javax.mail.Flags.Flag.ANSWERED のブール値。

Flag.Deleted
javax.mail.Flags.Flag.DELETED のブール値。

Flag.Draft
javax.mail.Flags.Flag.DRAFT のブール値。

Flag.Recent
javax.mail.Flags.Flag.RECENT のブール値。

Flag.Seen
javax.mail.Flags.Flag.SEEN のブール値。

コネクターの使用

以下に示す情報を参照して、コネクターを各種のモードで使用することができます。

イテレーター・モード

このモードでは、コネクターは指定したフォルダー (デフォルトでは INBOX) のすべてのメッセージに対して繰り返されます。各メッセージは項目に変換され、その属性がフローの後続ステップで使用可能になります。ここに記載されている情報を通じて、これについてさらに知ることができます。

接続しているバックエンド・メール・サーバーによっては、メッセージ本体と直接対話できない場合があります。これは、サーバーが、単一の本文部分ではなく複数の本文部分をサポートしているためです。この場合、すべての本文部分は多値属性としてアクセスできます (例:

```
work.getAttribute("mail.bodyparts").getValue(N).getContent();
```

ここで、*N* はメッセージ本体の数であり、番号ゼロは最初のメッセージ本体を表します)。サーバーが複数の本文部分を提供していない場合、メッセージ本体は属性 *mail.body* に入ります。

フォルダーが指定されていない場合、コネクターはメールボックスのすべてのフォルダーで繰り返されます。パラメーターを使用して、繰り返し時に特定のフォルダーがスキップされるように構成できます。これらのフォルダーの名前は、コンマで区切ってテキスト・ボックスに入力します。また、チェック・ボックスとして定義される別のパラメーターを使用すると、指定したフォルダーのサブフォルダー全体でもコネクターを繰り返すかどうかを指定できます。 POP3 が選択された場合は、このオプションが使用できないことに注意してください。これは、POP3 プロバイダーが単一のフォルダー「INBOX」を提供するためです。「**メール・フォルダー**」パラメーターが空のままである場合、コネクターは INBOX フォルダー内のメッセージで繰り返されます。

ルックアップ・モード

このモードでは、検索される項目はコネクターで定義された LinkCriteria に基づきます。ここに記載されている情報を通じて、これについてさらに知ることができます。

メッセージが検出されない場合は、例外がスローされます。このモードはイテレーター・モードと類似していますが、LinkCriteria を定義しない限り、メール・フォルダー内のメッセージで繰り返すことはできません。作業項目にマップされた属性は、単一のメッセージが検出された場合に充てんされます。複数の項目が戻された場合、AssemblyLine の実行は停止します。このような場合のロジックを「複数項目時」フックに設定することにより、これを回避できます。

メールボックス・コネクターをルックアップ・モードで使用している場合、検索可能なヘッダーは以下のとおりです。

- mail.from
- mail.to
- mail.cc
- mail.subject
- mail.messageid
- mail.messagenumber

削除モード

このモードでは、戻される項目は定義された LinkCriteria に基づきます。ここに記載されている情報を通じて、これについてさらに知ることができます。

1 つのメッセージが検出されると、そのメッセージは削除されます。メッセージが検出されない場合は、「一致なしの場合」のフックが (定義されていれば) 呼び出されるか、例外がスローされて実行が停止します。複数のメッセージが検出された場合は、「複数項目時」のフックが (定義されていれば) 呼び出されるか、例外がスローされます。

メールボックス・コネクターを削除モードで使用している場合、検索可能なヘッダーは以下のとおりです。

- mail.from
- mail.to
- mail.cc
- mail.subject
- mail.messageid
- mail.messagenumber

AddOnly モード

AddOnly モードは、指定したメールボックス・フォルダーにメッセージを入れる場合に使用します。この目的で使用する場合は、最初にメール・サーバー、メールの信任状、プロトコル・タイプ (IMAP のみ)、および新規メッセージが配信されるフォルダーを構成する必要があります。

このモードは IMAP プロトコルでのみ使用可能です。これは、POP3 プロトコルではメッセージの追加がサポートされていないためです。 Java Mail API の POP3 プロバイダーでの制約事項について詳しくは、http://java.sun.com/products/javamail/javadocs/com/sun/mail/pop3/package-summary.html を参照してください。

コネクターの構成で定義されているフォルダーが存在しない場合は、パラメーター「createFolder」が考慮されます。指定されたフォルダーが検査され、それが存在しない場合は、この名前で新規のフォルダーが作成されます。新規のメッセージは、属性 *mail.addMessage* でコネクターに配信されます。この属性は、タイプ `javax.mail.Message` のオブジェクト、またはそのタイプの配列に渡されます。コネクターはメールボックスに接続し、指定されたフォルダーにメッセージを追加します。

更新モード

更新モードでは、メールボックス・コネクターは指定されたメール・フォルダー内のメッセージのフラグを変更できます。サポートされているフラグは、Answered、Deleted、Draft、Recent、および Seen です。これらのパラメーターは、コネクターに、その出力マップの属性として渡すことができます。

そのため、各フラグおよび対応する属性は、ブール値型です。各フラグは、メソッド `javax.mail.Message.setFlag(...)` を使用して操作されます。

更新対象のフラグおよび新規の値は、作業項目内に指定します。更新対象のフラグがメッセージ・ストアでサポートされていない場合は、操作が失敗したフラグ属性を含めて、メッセージがログに記録されます。その後、コネクターはその他のフラグの更新を続行します。

メールボックス・コネクターを更新モードで使用している場合、検索可能なヘッダーは以下のとおりです。

- mail.from
- mail.to
- mail.cc
- mail.subject
- mail.messageid

- mail.messagenumber

構成

ここに記載されているパラメーターを使用することで、メールボックス・コネクターを構成できるようになります。

メール・サーバー
メールボックスのホストとなる POP/IMAP メール・サーバー。これには、次の例のように、スペースで区切ってポート番号を含めることができます (*url port*)。以下に例を示します。

```
domino.raleigh.ibm.com 110
```

SSL の使用
チェックすると、コネクターで SSL 接続が使用されます。チェックしないと、コネクターでは非 SSL 接続が使用されます。

メール・プロトコル
pop3 または **imap** を指定します。

ユーザー名
ユーザー名。

パスワード
ユーザー名のパスワード。

メール・フォルダー
メール・サーバー上のユーザーのメール・フォルダーの名前を指定します。

ユーザーのメール・フォルダーには、メール・サーバー上のユーザーのメール・メッセージが保管されます。 POP3 メール・プロトコルを使用している場合は、このパラメーターの値として「INBOX」を指定する必要があります。 IMAP メール・プロトコルを使用している場合には、メール・サーバーの任意の既存メール・フォルダーを指定できます。

フォルダーの作成
このボックスをチェックすると、パラメーター「**メール・フォルダー**」で指定されたメールボックスがない場合に、そのメールボックスが作成されます。 AddOnly モードでのみ適用されます。

サブフォルダーの取得
イテレーター・モードにおいて、指定したフォルダーのサブフォルダー全体でコネクターが繰り返されるかどうかを指定します。 AddOnly モードを除くすべてのモードに適用されます。

除外フォルダー
メールボックスでの繰り返しで除外されるフォルダーをコンマ区切りのリストで指定します。 AddOnly モードを除くすべてのモードに適用されます。

ポーリング間隔 (秒単位)
コネクターが新着メール・メッセージの確認でメール・サーバーをポーリングするまでのスリープ秒数を指定します。

メールボックス・コネクター・バッファーに格納されていたすべてのメール・メッセージが AssemblyLine によりコンシュームされた後で、コネクターは特定の期間スリープしてからメール・サーバーに再接続し、新着メッセ

ージがあるかどうかを確認します。つまり、コネクターは新着メール・メッセージ確認のためにサーバーをポーリングします。

特殊値「-1」を指定すると、コネクターは初回ポーリング以降は新着メール・メッセージ確認のポーリングを実行しません。つまり、 AssemblyLine はコネクターが初回ポーリングで取得したすべてのメッセージをコンシュームした後で終了します。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

メモリー・キュー・コネクター

メモリー・キュー (MemQueue)・コネクターは、メモリー・キュー機能 (別名 MemBufferQ) に対する読み取りおよび書き込みを実行するコネクターに類似した機能を提供します。以下に示す情報とリンクを使用して、これについて詳しく知ることができます。

これは、メモリー・キューにアクセスするためにスクリプトを記述するという方法の代替手段であり、 528 ページの『メモリー・キュー関数コンポーネント』 (関数コンポーネント) の拡張機能です。

コンポーネント間の通信に使用されるオブジェクトは永続的なものではなく、大規模な結果セットが戻されても処理できません。例えば、*ldapsearch* 操作から大量のデータが戻された場合などです。この問題を解決するために、スレッド・セーフな内部メモリー・キューを AL コンポーネント間の通信用データ構造として使用することができます。そのデータ構造に、バッファーが x% フル/空/データ利用可能などの時点でトリガーされるロジックを組み込むこともできます。

同一のキューに複数の読み取りプロセスと書き込みプロセスがある場合もあります。各書き込みプロセスは、データを追加する前にロックを取得する必要があります。書き込みプロセスは、読み取りプロセスがアクセスできるようにロック解除する必要があります。イテレーター・モードでのコネクターには、読み取りロックを解除するタイミングを決定するパラメーター (**単一の読み取り後**、**AL サイクル終了**、または**コネクターのクローズ**) があります。

このコネクターは、AddOnly モードとイテレーター・モードのみをサポートします。

注:

1. このコネクターは非永続的であるため、代わりに 346 ページの『システム・キュー・コネクター』を使用することが推奨されます。このコネクターは、永続オブジェクト・ストレージを備えた基礎となる Java Messaging Service (JMS) 機能に基づいているためです。
2. メモリー・キュー ・コネクターがイテレーター・モードの場合、コネクターは構成されたキューから読み取ります。キューが存在しない場合は、作成されます。この動作を防ぐには、システム・プロパティー `tdi.memq.create.queue.default=false` を設定する必要があります。このように

設定することで、IBM Security Directory Integrator は以前のバージョンと同様に、イテレーター・モードでキューが存在しない場合は例外をスローします。

このコネクターは、JavaScript からセットアップされたメモリー・キュー・パイプと組み合わせて使用することもできます。ただし、メモリー・キュー・コネクターにより作成されたメモリー・キュー・パイプは、コネクターを閉じると終了する点に注意してください。

メモリー・キュー・バッファーは FIFO タイプのデータ構造であり、追加操作と読み取り操作が同時に発生しても構いません。一方で追加が発生し、もう一方で読み取りが発生した場合、このキューはパイプとして機能します。読み取りが発生すると、キューからデータが除去されます。

メモリー・キュー・バッファーでは、しきい値に達した場合の、システム・ストアを使用したオーバーフロー・ストレージ (利用可能なランタイム・メモリーの関数) が提供されます。

メモリー・キューのコンポーネント

メモリー・キューには、ウォーターマークとページの 2 つのコンポーネントがあります。以下のセクションで、これらのコンポーネントについて詳しく知ることができます。

ウォーターマーク
これはキューの最大サイズです。操作の詳細は、構成のページを参照してください。

ページ コネクターは、一連のオブジェクトを、システム・ストアに書き込む前にバッファーに入れることができます。コネクターが使用するバッファーのことを、ページといいます。ユーザーは、各ページに含めることのできるオブジェクトの数を指定できます。これは、ページ・サイズ・パラメーターを使用して指定されます。ページの使用は、コネクターがシステム・ストアとの間でオブジェクトをより効果的に読み取り/書き込みする場合に役立ちます。

ワークフローの概要

メモリー・キュー・バッファーは、オブジェクトが含まれるページのキューです。以下に示す情報を使用して、メモリー・キュー・バッファーのワークフローの概要を理解することができます。

特定のしきい値 (「ウォーターマーク」) に到達すると、新しくスレッドが作成されて、別のページ・バッファーへの書き込みが開始されます。ページがフルになると、そのページをメイン・キューまたはシステム・ストアのいずれかに転送します。ページがメイン・キューから読み取られると、システム・ストアからメイン・キューに次の 1 ページが転送されます。その処理と同時に、そのページがシステム・ストアから削除されます。

構成

以下に示すパラメーターを使用して、メモリー・キュー ・コネクターを構成することができます。

インスタンス
構成インスタンスの名前。これが NULL の場合、現在のインスタンスが想定されます。

キュー 作成するキューまたはパイプの名前。

読み取りタイムアウト
キュー内に項目が見つからない場合に、制御を戻すまで待機する間隔 (ミリ秒)。

イテレーター読み取りロックの解放
コネクターがイテレーター・モードである場合、指定したメモリー・キューの読み取りロックが解放されるタイミングがこれによって決定されます。選択可能な値は、**単一の読み取り時** (デフォルト)、**AL サイクル終了**、および**コネクターのクローズ**のいずれかです。

使用メモリー・パーセント
これにより、メモリー・キューで利用することのできるメモリーのパーセンテージが決定されます。デフォルト値は 50 です。

ウォーターマーク
これは、オブジェクトをシステム・ストアに格納するときのしきい値です。ページが実際に書き込まれる時期は「**ページ・サイズ**」パラメーターによって決定されるため、「ウォーターマーク」は「ページ・サイズ」の倍数にする必要があります。

ページ・サイズ
1 ページ内の項目数。デフォルト値は 100 です。

データベース名
システム・ストアのデータベース名: システム・ストア・データベースへの JDBC URL (または、デフォルトのシステム・ストアの場合はブランク)。

ユーザー名
システム・ストア・データベースに接続するためのユーザー名。

パスワード
データベースにサインオンするときに使用するパスワード。

テーブル名
ページングに使用するシステム・ストア表の名前。

詳細ログ
詳細なログ・メッセージが必要な場合は、チェックします。

メモリー・キューへのプログラマチックなアクセス

JavaScript からメモリー・キューへは、コネクターを使用した方法だけでなく、直接アクセスすることもできます。以下に示す方法を使用して、メモリー・キューにプログラマチックにアクセスすることができます。

1. パイプを新規に作成する方法には以下の 2 種類の方法があります。
   a. ページングを使用不可にする: **newPipe**(String instName,String pipeName,int watermark) // Does not require any DB related entries

b. ページングを使用可能にする: newPipe(String instName,String pipeName,int watermark,int pagesize) // Requires DB initialization

ページングが使用可能な場合のサンプル・スクリプトを以下に示します。

```
var memQ=system.newPipe( "inst","Q1",1000,10) ;
memq.initDB(dbName, jdbcLogin, jdbcPassword, tableName);  // Required to Initialize DB
memQ.write(conn);
```

2. **getPipe**(String instName,String pipeName)
3. **purgeQueue()**

   サンプル・スクリプトは次のようになります。

```
var q =system.getPipe("Inst1","Q1") ;
q.purgeQueue();
```

4. **deletePipe**(String pipeName)

   例:

```
var q =system.getPipe("Inst1","Q1");
q.deletePipe();
```

API 呼び出しを使用してメモリー・キューからの読み取りを実行するサンプル・スクリプトを以下に示します。

```
var memQ=system.getPipe( "inst","Q1") ;
var size=memQ.size();

for(var count=0;i<=size;count++){
 main.logmsg(memQ.read());
}
```

メモリー・ストリーム・コネクター

メモリー・ストリーム・コネクターは、どのような Java ストリームとの間での読み書きでもできますが、ほとんどの場合はメモリーへの書き込みのために使用されます。そして、後でメモリーからフォーマット済みデータを検索することができます。以下に示す情報を使用して、このコネクターの各動作モードと動作について知ることができます。

割り振られたバッファーへは必要に応じてアクセス/検索します。

注: メモリー・ストリームがローカル JVM に限定されているため、別の JVM (同一マシンおよび別のマシン) で実行中のタスクとデータを交換することはできません。

このコネクターは、イテレーター・モード、AddOnly モード、または受動状態でのみ使用できます。このコネクターの動作は、どのように初期化されているかによって決まります。

initialize(null)

これはデフォルトの動作です。コネクターはメモリーに書き込み、getDataBuffer() メソッドを使用してフォーマット済みデータを検索できます。このメソッドはメモリー・ストリーム・コネクターの中でのみ使用できます。コネクター名が MM である場合、次に示すコードはどこででも使用できます (例えば、プロローグ、エピローグ、すべてのフック、スクリプト・コンポーネントのほか、属性マッピングの中でも使用できます)。

```
var str = MM.connector.getDataBuffer();
// use str for something.
// To clear the data buffer and ready the Connector
  for more output, re-initialize
MM.connector.initialize(null);
```

initialize(Reader r)
コネクターは **r** から読み取ります。これは、ストリームからの読み取りを行いたい場合に使用できます。

initialize(Writer w)
コネクターは **w** に書き込みます。

initialize(Socket s)
コネクターは、ソケット **s** からの読み取りおよび書き込みの両方を行うことができます。

注: 他のデータ・ストリームからの読み取りまたはそこへの書き込みを開始する場合以外は、再初期化はしないでください。コネクター・インターフェース・オブジェクトを使用する場合は、679 ページの『コネクター・インターフェース・オブジェクト』を参照してください。このコネクターには、getDataBuffer() という追加メソッドがあります。

構成

ここに記載されているパラメーターを使用することで、メモリー・ストリーム・コネクターを構成できるようになります。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

パーサー
出力をフォーマットする、または入力を解析するパーサーの名前。

プロパティー・コネクター

ここに記載されている情報を使用することで、プロパティー・コネクターについて知ることができます。

IBM Security Directory Integrator のソリューションは、1 つ以上の IBM Security Directory Integrator 構成ファイル (XML 形式) にパッケージ化されています。その構成ファイルには、エンドポイント接続、データ・フロー、およびその他の機能のホストに関する設定が含まれています。構成ファイルには、ソリューションの作成に必要なすべての設定を保持できますが、構成の動作を変更するために、構成ファイル外部のデータ・ソース (標準 Java プロパティー、IBM Security Directory Integrator の外部プロパティー、および IBM Security Directory Integrator のシステム・ストア・プロパティーなど) の使用が必要な場合が多くあります。

プロパティー・ストアは、*key=value* 形式の構成情報を保持するために使用されます。プロパティー・コネクターは、このようなストアの処理に使用され、各プロパティーの読み取り/書き込みの操作、および特定のプロパティー値の暗号化/暗号化解除を実行します。一般的な `global.properties` および `solution.properties` が、このようなプロパティー・ストアの例です。

個々のプロパティー・ストアは、「**プロパティー鍵**」および「**暗号化**」パラメーターによって個々の証明書を使用して暗号化できます。これにより、サーバー証明書とは異なる証明書を使用して、ファイル内のプロパティーおよび必要な場合はファイル全体の両方を暗号化/暗号化解除できます。この証明書は複数のデベロッパーが 1 つのプロジェクトで作業中で、信任状を共有できない場合に役立ちます。

このコネクターは、内部メモリー・バッファーを使用してプロパティー・ファイル内のすべてのプロパティーを保持します。また、このコネクターを使用して JavaVM システム・プロパティー・オブジェクトにアクセスすることもできます。

このコネクターは、イテレーター、AddOnly、更新、ルックアップ、および削除の各モードをサポートします。

構成

ここに記載されているパラメーターを使用することで、プロパティー・コネクターを構成できるようになります。

コレクション・パス/URL
コレクション・タイプがファイル/URL の場合に読み取り/書き込むプロパティー・ファイルを指定します。このパラメーターは、コレクション・タイプがファイル/URL の場合に必須です。

作成 チェック・ボックスであり、チェックした場合 (デフォルト)、ファイルが自動的に作成されます。このチェック・ボックスが空で、ファイルが欠落している場合、例外がスローされます。

暗号化 チェックすると、入力したパスワードを使用して、このコレクションのプロパティーが暗号化されます。デフォルトではチェックは外されています (つまり、「false」)。

暗号アルゴリズム
「**暗号化**」が TRUE であるか、またはストリームに個別の暗号化値が含まれている場合に使用する暗号アルゴリズムです。 IBM Security Directory Integrator サーバーの暗号化を使用する場合は「server」を指定します。デフォルトの暗号は、 プロパティー `com.ibm.di.server.encryption.transformation` で `global.properties` または `solution.properties` に指定されています。

「**プロパティー鍵**」パラメーターを指定した場合、このパラメーターはその鍵で使用するアルゴリズムを指定します。keyalias が指定されていない場合、このパラメーターが、ファイル全体を暗号化するときに使用するアルゴリズムを指定します。この場合、「server」という語は IBM Security Directory Integrator サーバーの暗号化を使用することを意味し、それ以外は、アルゴリズムの鍵として「**パスワード**」パラメーターからのパスワードが使用されます。

パスワード
ストリーム/プロパティー値の暗号化/暗号化解除時に使用する秘密鍵。

「**プロパティー鍵**」が指定されず、「**暗号化**」にチェック・マークが付けられていて、「**暗号アルゴリズム**」が「サーバー」ではない場合にのみ使用されます。

プロパティー鍵

プロパティー・ファイル内の暗号化値を個別に暗号化または暗号化解除する場合に使用されるサーバー鍵ストアでの証明書の名前。「**暗号化**」パラメーターが true に設定されている場合、この証明書もプロパティー・ファイル全体の暗号化または暗号化解除に使用されます。このパラメーターが設定されている場合、「**パスワード**」パラメーターの値を上書きすることに注意してください。

このパラメーターは、ドロップダウン・リストです。ドロップダウン・リストはサーバー鍵ストアにある名前が自動的に入力されます。

自動再書き込み (AutoRewrite)

true の場合は、自動暗号化値が検出されるとコネクターがコンテンツを書き戻します。

このパラメーターが true に設定されている場合、任意の値が自動的に暗号化されていると、コレクションが即時に書き戻されます。プロパティーのプロパティー名の前に「{protect}-」のマークが付けられている場合、その値は暗号化されていないと自動的に暗号化されます。このパラメーターが true に設定されていない場合は、プログラムを使用してコレクションを書き戻す必要があります。

詳細ログ

このボックスをチェックすると、追加のログ・メッセージが生成されます。

コネクターの使用

プロパティー・コネクターは、標準の .property ファイル、Java プロパティー、またはシステム・ストア・ユーザー・プロパティー・ストアへの接続に使用できます。また、読み取り/書き込み中の各ストアの暗号化/暗号化解除を提供します。

このコネクターの標準的な動作では、その URL によって指定された .property ファイルに接続されます。これは、コネクターの「**コレクション**」パラメーターを設定し、「ユーザー定義」の各プロパティーを構成することにより、実現されます。

ただし、システム定義のプロパティー・ストアである、JVM (「java」) プロパティー、ユーザー・プロパティー・ストア、global.properties および solution.properties にアクセスすることもできます。これを実行するには、コネクターの collectionType プロパティーを設定する必要があります。このプロパティーは、構成画面には表示されませんが、以下のスクリプトで設定できます (例えば、「**Prolog**」->「**初期化前**」フックに挿入します)。

- **JVM プロパティー**: `thisConnector.connector.setParam("collectionType", "Java-Properties");`
- **ユーザー・プロパティー・ストア**: `thisConnector.connector.setParam ("collectionType", "System-Properties");`
- **global.properties**: `thisConnector.connector.setParam("collectionType", "Global-Properties");`
- **solution.properties**: `thisConnector.connector.setParam("collectionType", "Solution-Properties");`

注: これらのプロパティーのコレクションは、構成ブラウザーにおいて指定の構成ファイルの「プロパティー」フォルダーに表示されるものです。これらは、構成エディターを使用して変更可能することができます。これにより、アクセスにこのコネクターを使用する必要がなくなり、実行時にこれらのプロパティーのコレクション内で任意のプロパティーを変更できます。

これらのストアはすべて同じ Java VM 内で共用され、システム・ストアに書き込む AssemblyLine は、同じストアから読み取る Java VM 内のその他のすべての AssemblyLine に影響することになります。

global および solution のストア内のすべてのプロパティーは、開始時に IBM Security Directory Integrator サーバーによって Java プロパティー・ストアにその順序で伝搬されます。ここで重要となるのは、global および solution のストアを個別に処理できるようになること、およびこれらのファイルへの変更も (許可されている場合は) 可能になることです。各プロパティー・ストアには、構成インスタンス内で固有である固有の名前が付与されます。 IBM Security Directory Integrator サーバーで複数の構成インスタンスが実行される場合、それらのインスタンスは、Java、global、solution、およびすべてのシステム・ストアのプロパティー・ストア (system など) を共用しますが、それ以外はすべて構成インスタンスに対してローカルです。

注: コネクターを使用して外部プロパティーを処理する場合に、明示的な「コミット」を呼び出すことなく、暗号化されたプロパティーを自動的に書き戻すには、「**自動再書き込み** 」パラメーターを *true* に設定する必要があります。

プロパティー・コネクターのリンク基準は、「key equals keyvalue」という形式の単一の基準にのみすることができます (ここで、keyvalue は検索するキー値です)。これより高度な検索はできません。

プロパティー・ファイルのフォーマット

以下に示す例とリンクを使用して、プロパティー・ファイルのフォーマットを理解することができます。

```
# comment
' comment
// comment
!include filename
!merge filename

[{protect}-]keyword <colon | equals> [{encr}]value
```

注:

1. オプションの接頭部 {protect}- は、値が暗号化されているか、または値を暗号化すべきかのいずれかを示します。値の先頭が文字シーケンス {encr} である場合、値が既に暗号化されていることを意味します。
2. 「!include」では、現在のプロパティー・マップに条件なしで書き込まれたプロパティーの外部ファイル/URL が読み込まれます。
3. 「!merge」では、プロパティーがまだ存在していない場合に現在のプロパティー・マップに書き込まれる (非破壊書き込み) プロパティーの外部ファイル/URL が読み込まれます。
4. IBM Security Directory Integrator は現在、鍵と値のペアのプロパティー・ファイルで等号「=」またはコロン「:」のいずれか最初の方を分離文字として使用して

います。したがって、プロパティー名およびプロパティー値での等号またはコロンの使用はサポートされません。IBM Security Directory Integrator バージョン 6.0 以前のプロパティー・ファイルの鍵/値分離文字は、「:」だけです。そのため、V6.0 以前から移行されたプロパティー・ファイルは編集が必要になる場合があります。

構文検査は、プロパティー・コネクター、IBM Security Directory Integrator サーバー、および CE によって直接読み込まれるプロパティー・ファイルに対して使用されます。プロパティー・ファイルのフォーマットに準拠していない非ブランク行がある場合は、例外がスローされます。

プロパティー・ファイルのヘッダー

プロパティー・ファイルの最初の 1 行から 2 行は、以下のストリングで始まります。

```
##{PropertiesConnector}
```

このストリングは、この行がプロパティー・ファイルが書き込まれるたびに再書き込みされるヘッダーであることを表します。

最初の行は以下のようになります。

```
##{PropertiesConnector} savedBy=user, saveDate=date
```

ここで、user はファイルを保存したユーザーの名前で、date はファイルが保存された日付です。

ファイルの書き込み時に「**プロパティー鍵**」パラメーターが指定されていた場合、次の行は以下のようになります。

```
##{PropertiesConnector} encryptionKey=keyAlias
```

ここで、 keyAlias は「**プロパティー鍵**」パラメーターの値です。

関連情報

「*Directory Integrator の構成*」の『プロパティー・ストア』。

QRadar コネクター

IBM Security Directory Integrator の QRadar コネクターを使用して、サポートされないイベント・ソースを QRadar と統合します。

QRadar は、次世代のセキュリティー情報およびイベント管理のソリューションです。これは、Device Support Module (DSM) を介して各種のログ・ソースから取得するイベント情報を使用します。この情報は、Log Event Extended Format (LEEF) と呼ばれるフォーマットでなければなりません。LEEF の現行バージョンは 1.0 です。

QRadar コネクターは、以下のパラメーターを必要とします。

- Syslog 入力を介した LEEF フォーマットのイベント
- Universal LEEF DSM を介したファイル・インポート

QRadar コネクターは、サポートされないイベント・ソースと QRadar との統合を容易化するように設計されています。入力データ・フィールドから LEEF V1.0 スキーマの属性へのマッピングにより、有効な LEEF イベント情報を作成できます。イベント・データの読み取りまたは受信のために構成されたコネクターを使用して IBM Security Directory Integrator AssemblyLine を作成できます。その後に QRadar コネクターが続きます。QRadar コネクターによって、必要な LEEF 出力が生成されます。

この方法で作成されたイベントを QRadar で使用するためには、事前にそのイベントを QRadar でマップし、適切なカテゴリー化を使用できるようにする必要があります。詳しくは、「QRadar 資料」を参照してください。

QRadar コネクター・パラメーター

QRadar コネクターの「**接続**」タブでパラメーターを設定し、LEEF ファイルの作成方法を指定します。

LEEF フォーマットの syslog メッセージを直接 QRadar に送信するには、「**syslog への出力**」オプションを選択します。

後で QRadar へのインポートが可能な LEEF フォーマットのファイルを作成するには、「**syslog への出力**」オプションをクリアします。このセクションの例では、これらのメッセージを syslog に直接送信することを想定しています。

「**syslog への出力**」オプションを選択した場合は、以下のパラメーターを使用できます。

ホスト名
Syslog メッセージの送信先となるシステムのホスト名または IP アドレスを指定します。

このパラメーターは必須です。

ポート
Syslog メッセージの送信に使用し、QRadar が listen するポートを指定します。

Severity
Syslog メッセージの重大度設定を指定します。

以下の値を使用できます。

- alert
- critical
- debug
- emergency
- error
- informational
- notice
- warning

機能
Syslog メッセージに使用する機能名を指定します。

以下の値を使用できます。

- kernel
- user
- mail
- system daemons
- security/authorization
- internal syslogs
- line printer subsystem
- network news subsystem
- UUCP subsystem
- clock daemon
- security/authorization messages
- FTP daemon
- NTP subsystem
- log audit (note 1)
- log alert (note 1)
- clock daemon (note 2)
- local0
- local1
- local2
- local3
- local4
- local5
- local6
- local7

日付形式マスク
マップの対象の LEEF 属性の日付値に適用する Java SimpleDateFormat マスク (例えば、devTime など) を指定します。

このパラメーターのデフォルト値は MMM dd yy HH:mm:ss であり、Oct 16 12 15:15:57 のようなストリングが作成されます。

詳細ログ
デバッグを目的として、コネクターによってログ出力に Syslog メッセージを表示するかどうかを示します。

コメント
このコンポーネントに関するテキスト情報を格納します。

「**syslog への出力**」オプションをクリアした場合は、コネクターによって LEEF インポート・ファイルが作成され、以下のパラメーターを使用できます。

ファイル・パス
LEEF 出力の書き込み先ファイルのパスを指定します。書き込まれるイベントの数が、**「ファイルごとの最大イベント数」**パラメーターで設定された値を超えた場合、 すべてのファイルに 3 桁の数値が付加されます。この数値は 000 (最初のファイル用) から開始します。

このパラメーターは必須です。

日付形式マスク
マップの対象の LEEF 属性の日付値に適用する Java SimpleDateFormat マスク (例えば、devTime など) を指定します。

このパラメーターのデフォルト値は MMM dd yy HH:mm:ss であり、Oct 16 12 15:15:57 のようなストリングが作成されます。

ファイルごとの最大イベント数
出力を複数の LEEF ファイルに分割するかどうかを指定します。

出力を分割するには、この値をゼロより大きい値 (>0) に設定します。このファイルには、**「ファイル・パス」**パラメーターで定義されたファイル名を使用して名前が付けられ、000 (最初のファイル用) から開始する 3 桁の数値が付加されます。

値を入力しない場合、またはゼロ以下の値 (<= 0) を入力した場合は、すべてのイベントが 1 つのファイルに書き込まれます。このファイルには、**「ファイル・パス」**パラメーターで指定された名前が使用されます。

デフォルトでは、このパラメーターは設定されず、単一の出力ファイルが作成されます。

詳細ログ
コネクターがデバッグ情報をログに出力したかどうかを示します。

コメント
このコンポーネントに関するテキスト情報を格納します。

これらのパラメーターを構成した後で、コネクターの入力マップまたは出力マップの「スキーマ」ペインで「**接続**」をクリックします。標準 LEEF フィールドのリストが返されます。このリストには、各フィールドのタイプおよびそれが必須またはオプションのいずれであるかが示されます。

必須の 4 つの属性のみが出力マップに表示されている必要があります。これらの属性は、1 つのイベントに対して書き込まれる LEEF ヘッダーの一部であり、LEEF 出力を作成するためにマップされている必要があります。スキーマ・リストをスクロールダウンすると、以下の必須属性が表示されます。

- LEEFHeader_EventID
- LEEFHeader_Product
- LEEFHeader_ProductVersion
- LEEFHeader_Vendor

それぞれの必須属性の名前の後に続く [1..1] の表記は無視してもかまいません。

QRadar コネクターのセットアップ

入力ファイルを解析するために、QRadar コネクターを使用して AssemblyLine をセットアップできます。

このタスクについて

QRadar コネクターは、IBM Security Directory Integrator バージョン 7.2.0.1 以降で使用できます。IBM Security Directory Integrator をインストールすると、QRadarConnector.jar ファイルが *tdi_install*/jars/connectors ディレクトリーにコピーされます。

以下の手順では、構成エディターで AssemblyLine とコネクターを作成し構成する方法を理解していることが前提となっています。IBM Security Directory Integrator の資料の『始めに』および『構成』のセクションを参照してください。

手順

1. この手順のために、フィールドを区切るためにセミコロン (;) を用いる CSV 形式のサンプル入力ファイルを作成するか使用します。そのファイルに Alerts.csv という名前を付けます。

```
SYSTEM;MANUFACTURER;MACADDRESS;SYSTEMVRS;PORT;HOSTNAME;IPSOURCE;WHEN;ALERTID;ACCOUNT
StreamPort;TT Sys;1F:9D:A7:9B:29:78;1.5.12;2332;matrix.net;213.162.242.251;
     Fri Apr 27 13:04:09 GMT+1 2012;A00398988;WRST
E112-B;Sun;64:C0:2A:7F:6A:5A;2.3.17;3566;matrix.net;195.89.246.157;
     Fri Apr 27 13:04:09 GMT+1 2012;ABN107441;SWCHW
AccessGate;Oracle;87:F3:D2:33:A8:32;5.1.6;3962;abc.com;105.168.129.139;
     Fri Apr 27 13:04:09 GMT+1 2012;AL662162;GRCO
StreamPort;IBM;1C:D8:B2:BD:29:DD;8.2.10;8597;ccrd.comgroup.eu;140.62.226.198;
     Fri Apr 27 13:04:09 GMT+1 2012;ABN861291;TEL5
NetViewer;Elektron;65:70:22:50:FB:CB;5.7;1177;sil2.devops.crund.com;102.204.120.233;
     Fri Apr 27 13:04:09 GMT+1 2012;A00897609;FDDLR
Auth Grid;HP;E0:C0:52:03:BE:ED;4.0.16;9957;fldrs.omnicom.net;94.23.123.47;
     Fri Apr 27 13:04:09 GMT+1 2012;ABN739017;GRMM
Facilities Monitor;Cisco;EC:E0:CB:85:16:1F;2.1.18;3434;fldrs.omnicom.net;112.192.157.23;
     Fri Apr 27 13:04:09 GMT+1 2012;CRT852913;GRCO
Omnisys;Cisco;09:1E:EA:54:B8:C7;2.3.17;6555;baynter.org;80.189.199.43;
     Fri Apr 27 13:04:09 GMT+1 2012;A00344678;ABCO
```

2. IBM Security Directory Integrator 構成エディターで「**AssemblyLine**」を作成します。
3. 「ナビゲーター」ペインから「**QRadar コネクター**」をドラッグして AssemblyLine に追加します。 コネクターを AssemblyLine に追加する方法について詳しくは、IBM Security Directory Integrator の資料の『コネクター』セクションを参照してください。 AssemblyLine 内の「**QRadarConnector**」コンポーネントが青色で表示されます。これは、ライブラリー内のコネクターから構成を継承していることを示します。
4. CSV ファイルを読み取るために、イテレーター・モードの「**ファイル・システム**」コネクターを AssemblyLine に追加します。
5. ファイルのデコードを処理するために、ファイル・コネクターの「**CSV パーサー**」を選択します。
6. コネクターの名前を変更します。 例えば、Read Alerts file に名前を変更します。
7. この Read Alerts file コネクターを、AssemblyLine の「**フィード**」セクションに配置します。 このイテレーター・コネクターはファイル全体を読み取り、処理のために、ユーザーが「**データ・フロー**」セクションに入力した任意のコンポーネントに解析データを渡します。
8. 「**接続**」タブで、「**ファイル・パス**」にサンプルの Alerts.csv ファイルを指定します。

9. ファイルのスキーマをディスカバーするために、「**入力マップ**」タブで、「**接続**」をクリックします。

   このアクションは、正しいパーサーを選択したことを確認する場合にも役立ちます。「**接続**」をクリックすると、コネクターは接続の初期化を行い、接続されたシステムのスキーマを照会します。スキーマ情報を提供した RDBMS、LDAP ディレクトリー、またはその他のシステムを使用している場合は、使用可能な属性のリストを表示できます。
10. 「**次へ**」をクリックし、「**サンプル値**」列を、入力ファイルから解析された次の項目でリフレッシュします。
11. 値を参照して、コネクターがそのファイルを読み取り、解析できることを確認します。
12. 入力マップをセットアップします。

    これらのフィールドは、AssemblyLine での処理用にはまだ選択されていません。コネクターから AssemblyLine へのデータの受け渡し方法を記述するマッピング・ルールを作成する必要があります。

    次のいずれかの操作を実行します。
    - スキーマ領域から属性をドラッグして、マッピング領域にドロップします。
    - マッピング領域で「**追加**」をクリックします。

    この例の場合、すべてのスキーマ属性を処理のために AssemblyLine に移動させる必要があります。「**追加**」をクリックしてから、「**すべての属性をマップ**」するためにワイルドカード・マップ (*) を指定します。

    これで、CSV ファイルの各行からのすべてのフィールドが作業項目の属性として返されるようになります。各属性には、値の取得元のフィールド名が保持されます。

次のタスク

QRadar コネクター・パラメーターを構成します。この例では、以下の設定を構成します。

- 「**syslog への出力**」オプションを選択します。
- 「**ホスト名**」 (サンプル値: `localhost`) を指定します。
- 「**ポート**」 (サンプル値: `514`) を指定します。
- 「**重大度**」 (サンプル値: `debug`) を指定します。
- 「**機能**」 (サンプル値: `mail`) を指定します。
- 「**詳細ログ**」オプションを選択します。

LEEF スキーマへの入力データのマッピング

入力データを LEEF スキーマにマッピングすることによって、QRadar に送信される Syslog メッセージに書き込む値を指定します。

手順

1. 「**出力マップ**」タブで、「スキーマ」ペインの「**接続**」をクリックして LEEF スキーマをディスカバーします。
2. 入力フィールドの必須属性を LEEF スキーマの属性にマップします。マッピング・ルールを作成するために、スキーマ領域から属性を選択し、その属性を左側にドラッグします。
3. 必須属性に加えて、「スキーマ」ペインの以下の属性をマップします。

   devTime
   デバイス時刻。イベント・ログを提供するホストから生成される未加工のイベント日時です。

   dst
   イベント宛先の IP アドレス。

   dstMAC
   16 進形式でのイベント宛先の MAC アドレス。

   dstPort
   イベントの宛先ポート。
4. QRadar コネクターの「**出力マップ**」に追加したマッピング・ルールを編集することにより、これらの属性のマッピングを変更します。

   各マッピング・ルールは、マッピング後に表示されるターゲット属性名と、この属性の値の割り当てから構成されます。

   「**出力マップ**」において、割り当てはマッピング・ルールの左側に表示され、ターゲット属性名は右側に表示されます。

   例えば、「スキーマ」ペインから「**出力マップ**」に `LEEFHeader_EventID` 属性をドラッグした場合、新規ルールのターゲット属性名は、選択したスキーマ属性と以下のように同じになります。

   - **割り当て**: `work.LEEFHeader_EventID`
   - **コンポーネント属性名**: `LEEFHeader_EventID`
5. マッピング・ルールを修正します。

   「スキーマ」ペインからマップに属性をドラッグすると、デフォルトの割り当てが定義されます。この割り当ては、作業項目の属性と同じ名前の属性として定義されます。

   入力ファイルから読み取られているフィールドを参照するように、このデフォルトの割り当てを変更する必要があります。

   a. 割り当ての値をダブルクリックしてスクリプト・エディターを開きます。
   b. 作業項目属性の名前を変更して、対応する入力フィールドと一致させます。

   作業項目は、イテレーター・コネクターによって読み取られる値を保持するデータ・バケットです。これは、*work* という名前の変数としてスクリプト記述に使用できます。

   注: 「CTRL + スペース」キーを押すと、入力属性のリストを表示できます。また、`work` と入力しても同じリストが表示されます。

トピック 287 ページの『QRadar コネクターのセットアップ』で説明されているサンプル・シナリオの完全なマップを以下に示します。

表 31. 出力マップ

割り当て	コンポーネント属性
work.ALERTID	LEEFHeader_EventID
work.SYSTEM	LEEFHeader_Product
work.SYSTEMVRS	LEEFHeader_ProductVersion
work.MANUFACTURER	LEEFHeader_Vendor
work.WHEN	devTime
work.IPSOURCE	dst
work.MACADDRESS	dstMAC
work.PORT	dstPort

次のタスク

QRadar ログ・ソースをセットアップします。

QRadar ログ・ソースのセットアップ

QRadar がソースから Syslog メッセージを受信できるようにするには、専用のログ・ソースを構成する必要があります。

このタスクについて

重要: 「**ログ・ソース・タイプ**」パラメーターと「**プロトコル構成**」パラメーターを正しく設定する必要があります。そうしないと、ユーザーが送信した Syslog イベントが正しく受信および解析されません。詳しくは、「QRadar 資料」を参照してください。

手順

1. 「QRadar SIEM」コンソールにログオンします。
2. 「**管理**」タブをクリックします。
3. 「**データ・ソース**」>「**イベント**」セクションで「**ログ・ソース**」をクリックします。
4. 「**追加**」をクリックして、ログ・ソースを作成します。
5. 以下の最低限のパラメーターを設定します。

   ログ・ソース名
   ログ・ソースのタイトルを入力します。この名前はログ・アクティビティー・ウィンドウに表示されます。

   ログ・ソースの説明
   ログ・ソースの説明を入力します。

   ログ・ソース・タイプ
   イベントのフォーマットを指定します。「Universal LEEF」値を選択します。

「Universal LEEF」値を選択しない場合、QRadar は、QRadar コネクターを介して送信された Syslog メッセージを解析できません。

プロトコル構成
このログ・ソースのプロトコルを選択します。「Syslog」値を選択します。これは、QRadar コネクターが使用するプロトコルです。

ログ・ソース ID
IBM Security Directory Integrator サーバーの IP アドレスを入力します。

使用可能
このオプションを選択してログ・ソースを使用可能にします。

6. 「**保管**」をクリックします。
7. 「QRadar SIEM」コンソールの「**管理**」タブで、「**変更のデプロイ**」をクリックして新しいログ・ソースをアクティブにします。

次のタスク

IBM Security Directory Integrator および QRadar の統合ソリューションをテストします。『ソリューションの検証』を参照してください。

ソリューションの検証

QRadar コネクターの構成ステップが完了し、属性マップをセットアップした後に、ソリューションのテストを行い、イベントが QRadar に送信されることを確認できます。

手順

1. IBM Security Directory Integrator 「構成エディター」で AssemblyLine を開きます。
2. 「AssemblyLine エディター」ウィンドウで「**コンソールで実行**」をクリックします。 AssemblyLine が開始され、AssemblyLine によってログに記録されている出力が表示されます。
3. ログ出力に、AssemblyLine コネクターで実行された各種操作のメトリックが含まれていることを確認します。 例えば、以下のサンプル出力は、入力ファイルから 8 行が読み取られ、その後 QRadar コネクターによって書き込まれたことを示しています。

```
[Read Alerts file] Get:8
[QRadarConnector] Add:8
```

4. QRadar に送信されたイベントを確認するために、「QRadar SIEM」コンソールにログオンします。
5. 「**ログ・アクティビティー**」タブをクリックします。
6. 「**ログ・アクティビティー**」にある表に、Syslog イベントが含まれていることを確認します。

次のタスク

ログ結果が予期したものではない場合は、以下の情報を参照してトラブルシューティングを行います。

- IBM Security Directory Integrator 「構成エディター」において、QRadar コネクターの「**接続**」タブで、「**詳細ログ**」オプションが選択されていることを確認します。このオプションが選択されていない場合、QRadar は、書き込まれた実際の LEEF イベントのログ出力を取得しません。
- QRadar イベントに対する着信 Syslog メッセージのマッピングを構成する必要があります。このマッピングが構成されていない場合、QRadar の「**ログ・アクティビティー**」ページにおいて、イベントは「**イベント名**」列で「不明」と表示されます。
- 「**ログ・アクティビティー**」ページにイベントが表示されない場合は、表示が一時停止になっていないことを確認してください。「**ビュー**」リストから「**リアルタイム (ストリーミング)**」を選択し、すべてのフィルターを削除することによって、ライブ・フィードを表示していることを確認できます。それでもイベントが表示されない場合は、コネクターの QRadar ホスト名と Syslog ポートの設定が正しいことを確認してください。
- 「**ログ・アクティビティー**」の下にイベントは表示されるが、ログ・ソースの名前が表示されない場合、「**ログ・ソース ID**」の値に誤りがある可能性があります。IP アドレスが Syslog パケットのアドレスに一致しない場合は、それらは代わりに汎用ログ・ソースによって処理されます。この場合、解析は実行されず、ソース IP 列と宛先 IP 列は、受信したパケットの送信側 IP アドレスにデフォルトで設定されます。ログ・ソースの「**ログ・ソース ID**」パラメーターにこの値を指定する必要があります。
- 「**ログ・アクティビティー**」の下にログ・ソースの名前が正しく表示されているが、それでもなおソース IP 列と宛先 IP 列に IBM Security Directory Server の IP アドレスが表示される場合があります。この場合、「**ログ・ソース・タイプ**」に「Univeral LEEF」を選択していることを確認してください。そうでない場合、解析は失敗します。

RAC コネクター

以下に示す情報とリンクを使用して、RAC コネクターとそのプロパティーについて理解することができます。

注: このコネクターは推奨されません。IBM Security Directory Integrator の将来のバージョンでは削除されます。

「RAC」は、リモート・エージェント・コントローラーを意味しますが、このテクノロジーの現在の名前はエージェント・コントローラー です。

エージェント・コントローラーとは、そのドメイン下のエージェントとクライアント・アプリケーションが対話できるようにするサーバーです (hthttp://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.tptp.platform.agentcontroller.doc.user%2Fconcepts%2Fac%2Fc_ac_ovr.html)。

Generic Log Adapter (GLA) は、専有のログおよびトレース・データを Common Base Event 形式に変換します (http://www.ibm.com/developerworks/library/specification/ws-cbe/)。 Generic Log Adapter が必要になる理由は、ログ・ファイルの読み取りは煩雑であり、すべてのタイプのログに対するパーサーの作成は拡張が容易ではなく、いずれにしてもカスタマイズされる傾向があるということです。 GLA はエー

ジェント・コントローラーのエージェントとして動作し、各クライアントがリモートのアプリケーション・ログをモニターできるようにします。

エージェント・コントローラーおよび Generic Log Adapter について詳しくは、http://www.eclipse.org/tptp/home/documents/index.php を参照してください。

RAC コネクターでは、以下のように RAC に対するデータの読み取りおよび書き込みが可能です。

- **AddOnly** モードの RAC コネクターは、ロギング・エージェントを使用して Common Base Event をパブリッシュします。このモードの RAC コネクターは、521 ページの『CBE 関数コンポーネント』のインスタンスを使用して、入力スキーマ属性を単一 Common Base Event オブジェクトに変換します。

  このコネクターは、ローカルのエージェント・コントローラーにエージェントとして登録され、AssemblyLine から受信した Common Base Event オブジェクトを送信します。

  RAC コネクターでは、その初期化時にリモートのエージェント・コントローラーが実行されている必要がありません。ローカルのエージェント・コントローラーが起動されるとすぐに、ロギング・エージェントが登録され、クライアントでモニターできるようになります。

  重要な点は、ローカルのエージェント・コントローラーがアクティブではない場合でも、このコネクターはエラーを報告しないことです。

  このコネクターは、ローカル・マシンにエージェント・コントローラーがインストールされていない場合であってもエラーを報告しません。その場合はもちろん、ロギング・エージェントは登録されません。
- **イテレーター**・モードの RAC コネクターは、リモート・ロギング・エージェントのクライアントとして機能します。

  つまり、コネクターは、リモート・マシン上のエージェント・コントローラーと通信して、特定のロギング・エージェントのハンドルを取得し、Common Base Event オブジェクトの形式でのログ・データの受信を開始します。

  リモートのエージェント・コントローラーが停止すると、コネクターはエージェント・コントローラーからの応答を待機してハングします (これは、現在のクライアント・ライブラリーの実装のためです)。したがって、再接続のロジックはすべて使用できません。

  このコネクターは、内部キューを使用して、入力される Common Base Event (CBE) を保存します。結果として、このコネクターは、キューにイベントを保持したままにできるため、エージェント・コントローラーの停止後でも CBE の取り出しを継続できます。エージェント・コントローラーのクライアント・ライブラリーによる制約事項のため、Common Base Event オブジェクトが 8 KB を超える大きさの XML にシリアライズされた場合、このコネクターはそのオブジェクトを処理できません。大きさが 8 KB を超えるデータ部分が受信されると、コネクターはそれ以上のイベントを処理せず、リモート・ロギング・エージェントが停止するまで待機します。これは、クライアント・ライブラリーのインプリメンテーションの制限です。

終了時に、コネクターはリモート・エージェントから切り離されます。この手順が何らかの理由 (例: JVM が強制終了された) でスキップされると、その他のクライアントはエージェントをモニターできなくなります。さらに、エージェントは、モニターされ続けていると認識したままになります。

例えば、構成エディターから実行されたコネクターがあり、AssemblyLine を手動で停止すると、そのコネクターをロギング・エージェントから切り離す機会がなくなります。そのため、そのコネクターが再実行された場合、エージェント・コントローラー (およびエージェント) はエージェントが既にモニターされていると認識しているため、コネクターはそのエージェントからデータを受信しません。

特定の時間においてロギング・エージェントをモニターできるクライアントは 1 つのみであるため、イテレーター・モードにある 2 つの RAC コネクターが同時に同じエージェントを指し示すことがないようにしてください。

構成

構成パネルに表示されるコネクターの名称は、「RAC コネクター」です。ここに記載されているパラメーターを使用することで、RAC コネクターを構成できるようになります。

リモート・ロギング・エージェント名
イテレーター・モードで使用されます。

モニターするリモート・ロギング・エージェントの名前です。

エージェント・コントローラー・ホスト
イテレーター・モードで使用されます。

リモート・エージェント・コントローラーのホストです。デフォルト値は「localhost」です。

エージェント・コントローラー・ポート
イテレーター・モードで使用されます。

リモート・エージェント・コントローラーのポートです。デフォルト値は 10006 です。

受信キュー・サイズ
イテレーター・モードで使用されます。

コネクターが読み取りを管理する前に受信イベントをバッファーに入れるキューのサイズです。デフォルト値は 1024 です。

非活動エージェントのデータを待機
イテレーター・モードで使用されます。

リモート・エージェントが終了した後の各データ受信のタイムアウト (秒) です。このタイムアウトに到達すると、エージェントのデータは空になったと見なされ、コネクターは終了します。デフォルト値は 5 です。

接続タイムアウト
エージェント・コントローラーへの接続のソケット・タイムアウト (秒) です。デフォルト値は 5 です。

ロギング・エージェント名
AddOnly モードで使用されます。

ローカル・エージェント・コントローラー内のロギング・エージェントの名前です。デフォルト値は「tdi_logging_agent」です。

モニター対象の待機

AddOnly モードで使用されます。

モニター対象エージェントの、データを RAC に送信する前の待機時間 (秒) です。ゼロの場合は無期限に待機します。デフォルト値は 0 です。

詳細ログ

チェックすると、追加のログ・メッセージが生成されます。

コネクターの使用

ここでは、RAC コネクターで使用できるさまざまな動作モードについて説明します。

AddOnly モード

AddOnly モードで動作する場合は、IBM Security Directory Integrator サーバー上の最初の RAC コネクターによって、ロギング・エージェントがローカルのエージェント・コントローラーに登録されるようにする必要があります。

AssemblyLine から受信されるすべての Common Base Event オブジェクトは、XML としてシリアライズされ、ロギング・エージェントに書き込まれます。ロギング・エージェントは、IBM Security Directory Integrator サーバーのマスター・プロセスが作動している限り、作動し続けます。このエージェントはその存続期間中は、それを登録したコネクターが既にクローズされている場合でも、クライアントによるモニターが可能です。ただし、IBM Security Directory Integrator サーバーが停止 (または異常終了) した場合、エージェント・コントローラー (RAC) は IBM Security Directory Integrator のロギング・エージェントの登録を強制終了します。

コネクターは、ロギング・エージェントへのデータ書き込みを開始する前に、指定された時間の間、モニター側クライアントの到着を待機します。特殊な場合には、無期限に待機することもできます。これは、コネクター・パラメーター「**モニター対象の待機**」によって指定されます。クライアントがエージェントのモニターを開始すると、エージェントはエージェント・コントローラーへのデータ転送を開始します。次に、エージェント・コントローラーはクライアントにデータを送信します。

コネクターによるそれぞれの書き込み試行前に待機が発生します。

待機時間の期限に到達し、まだモニター側クライアントが到着しない場合、コネクターは例外をスローします。ただし、コネクターの待機中にクライアントがエージェントのモニターを開始した場合、その待機は中断され、エージェントはエージェント・コントローラーへのデータ転送を開始します。

コネクター・パラメーター「**モニター対象の待機**」の値によっては、クライアントによるエージェントのモニター開始をコネクターが無期限に待機する可能性があります。これにより、AssemblyLine 全体が無期限にブロックされる場合があります。この理由から、以下のコネクター・メソッドが用意されています。

```
public boolean isLogging();
```

このメソッドは、このコネクターからのデータをモニター/listen しているクライアントがある場合は *true* を戻し、それ以外の場合は *false* を戻します。このメソッドは、JavaScript を使用してアクセス可能であり、コネクター・オブジェクト (つまり、`thisConnector.islogging()`) で呼び出すことができます。

このメソッドを使用すると、AssemblyLine の実行がコネクターに到達したときにそのコネクターによるブロックが発生するかどうかを検出できます。ブロッキングは好ましくないが、データの損失は許容できない場合に、メソッド isLogging() が false を返す場合は、データをキュー (IBM Security Directory Integrator メモリー・キューなど) に一時的に保管する解決策をインプリメントすることも考えられます。

AddOnly モードに対するインストール後の構成

RAC コネクターの AddOnly モードでは、エージェント・コントローラーのバイナリー (.dll、.so) がオペレーティング・システムの動的ライブラリー・ローダーで使用可能になっている必要があります。これを実現するための推奨方法は、エージェント・コントローラーのバイナリー・フォルダーを、Windows プラットフォームの場合は PATH 環境変数に、Linux プラットフォームの場合は LD_LIBRARY_PATH 環境変数に含めることです。これは、グローバルに実行することも、または IBM Security Directory Integrator サーバーのプロセスに対してのみ実行することもできます。例を示します。

- Windows の場合: PATH 環境変数を「マイ コンピュータ」->「プロパティ」->「詳細設定」->「環境変数」で変更します。エージェント・コントローラーのライブラリーへの必要なパスを追加します。
- Linux の場合: 以下のような行を開始スクリプト (ibmdisrv および ibmditk) の PATH 定義の後、開始行の前に追加します。

  ```
  LD_LIBRARY_PATH=/AgentController/lib
  export LD_LIBRARY_PATH
  ```

  ユーザー独自の LD_LIBRARY_PATH エレメントがある場合は、それらを LD_LIBRARY_PATH 定義に追加します。

イテレーター・モード:

イテレーター・モードの RAC コネクターは、リモート・エージェント・コントローラーのクライアントとして機能します。ここに記載されている情報を通じて、これについてさらに知ることができます。

コネクターはエージェント・コントローラーに接続して、ロギング・エージェントのハンドルを取得します。エージェントの名前はコネクターの構成に指定されています。ハンドルを取得後、コネクターはロギング・エージェントのモニターを開始します。モニター時には、コネクターはロギング・エージェントによって生成されたデータを受信します。

データ受信は、エージェント・コントローラーのクライアント・ライブラリーによって非同期に処理され、そのキューに入れられます。コネクターはデータ受信の発生時に通知を受け、コネクターがキューから読み取ったときに、受信されるバイナ

リー・データがバッファーに入れられます。キューはブロッキングを行い、データがない場合はコネクターが待機し、キューに空き領域がない場合はデータ・プロセッサーが待機するようにします。

受信されるバイナリー・データには、UTF-8 のエンコードで XML としてシリアライズされた CommonBaseEvent オブジェクトが含まれています。さらに、CommonBaseEvent はバッファーからデコードされ、入力マップによりコネクターで使用可能になります。

コネクターによるエージェント・コントローラーへのアクセス時に、指定した名前のアクティブなエージェントがない場合、コネクターはそのエージェントが登録されるまで待機します。

コネクターがイベントを listen している間のある時点で、そのエージェントの登録が解除されると、コネクターは同じ名前の別のエージェントが現れるまで待機します。基本的に、コネクターはエージェント・コントローラーへの接続に失敗する場合を除いて、停止することはありません。

コネクターは、現在の AssemblyLine の反復でコネクターによって取得された Common Base Event オブジェクトへのアクセスを提供する、以下のようなメソッドを公開します (コネクターの「getNextEntry」メソッドによって処理される最終イベント)。

```
public CommonBaseEvent getCurrentCbeObject();
```

スキーマ:

RAC コネクターのスキーマについて知るには、以下のリンクを使用できます。

コネクターは、内部的に 521 ページの『CBE 関数コンポーネント』を使用し、その特定の FC のスキーマを使用します。

関連情報

エージェント・コントローラー: 概要、アーキテクチャー、管理、および構成
(Agent Controller: overview, architecture, administration and configuration)、
TPTP データ収集フレームワーク。Java/C++ を使用したエージェントおよびクライアントの開発方法
(TPTP Data Collection Framework. How to develop agents and clients using Java/C++)、
ロギング・エージェントによるアプリケーションのモニタリング
(Monitoring an application with logging agents)、
ログおよびトレース・アナライザー、
133 ページの『Generic Log Adapter コネクター』.

RDBMS 変更検出コネクター

RDBMS 変更検出コネクターにより、IBM Security Directory Integrator で特定の RDBMS 表内の変更を検出できるようになります。現在は、Oracle、DB2、MS SQL、Informix、および Sybase の各データベースの表に対してセットアップ・シナリオが提供されています。

RDBMS には、選択されたデータベース表上で行われた変更を外部に通知するための共通のメカニズムはありません。このことに対処するため、IBM Security Directory Integrator では、一部の RDBMS メカニズム (トリガーやストアード・プロシージャーなど) が、ターゲット表内の変更されたレコードごとに 1 つのレコードを持つ個別の変更表を保持できます。シーケンス番号も、同じメカニズムによって保持されます。

LDAP 変更検出コネクターと同様に、RDBMS 変更検出コネクターは特定の形式で構成された変更表と通信し、そのことによって他のシステムに変更を伝搬できます。形式は IBM DB2 Information Integrator (バージョン 8) で使用するものと同じであり、IBM Security Directory Integrator ユーザーは、DB2II を使用してそのような表を作成したり、別の方法で表を作成するオプションを選択できます。RDBMS 変更検出コネクターは、シーケンス番号を常に追跡することにより、変更表を使用して最後の繰り返し以降の変更のみを報告します。

RDBMS 変更検出コネクターは、JDBC を使用して特定の RDBMS 表に接続します。 JDBC ドライバーの問題について詳しくは、181 ページの『JDBC コネクター』を参照してください。

RDBMS 変更検出コネクターは、イテレーター・モードのみで作動します。

このコネクターでは、項目レベルでのみデルタ・タグがサポートされています。

RDBMS 変更検出コネクターは、特定のフィールドを読み取り、変更表内の新規の変更を判別します (301 ページの『変更表の形式』を参照してください)。 RDBMS 変更検出コネクターは、次の変更表レコードを読み取るか、または最初の変更表レコードを検索します。 RDBMS 変更検出コネクターによって変更表内でデータが検出されない場合、RDBMS 変更検出コネクターは最大待機時間を超過したかどうかを確認します。最大待機時間を超過した場合、RDBMS 変更検出コネクターは、**NULL** を戻して反復の終了をシグナル通知します。変更表内でデータが検出されず、最大待機時間は超過していない場合、RDBMS 変更検出コネクターは、指定された秒数 (**ポーリング間隔**) だけ待機した後、次の変更表レコードを読み取ります。

RDBMS 変更検出コネクターが変更表にデータを戻した場合、コネクターはユーザー・プロパティー・ストア (このタイプの永続的情報用に調整されたシステム・ストア内の領域) にある **nextchangelog** 番号を増分して更新します。

項目が戻されるたびに、制御情報 (カウンター、操作、時刻/日付) が項目プロパティーに移動します。変更表内の制御情報以外のすべてのフィールドは、そのまま属性として項目にコピーされます。項目オブジェクト操作 (**getOperation** により戻される) は、対応する変更ログ操作 (追加、削除、または変更) に設定されます。

原則として、このコネクターは SSL プロトコルを使用してセキュア接続を処理できますが、この SSL サポートをセットアップするには、ドライバー固有の構成手順を実行する必要があります。詳しくは、製造元のドライバーの資料を参照してください。

構成

以下に示すパラメーターを使用して、RDBMS の変更検出の構成を設定することができます。

JDBC URL
JDBC プロバイダーの資料を参照してください。これは、ターゲット・データベースへの JDBC URL です。

ユーザー名
コネクターが RDBMS にサインオンするときのユーザー ID。このユーザーが使用できる表のみが表示されます。

パスワード
ユーザーのパスワード。ユーザー名/パスワードの認証メカニズムを使用した RDBMS に対する認証に使用されます。

スキーマ
モニターするデータベースの表のスキーマ (所有者)。これをブランクのままにした場合は、**ユーザー名**パラメーターの値が使用されます。

JDBC ドライバー
JDBC ドライバーのクラス名。このパラメーターのデフォルト値は `com.ibm.db2.jcc.DB2Driver` です。

表名　変更をモニターする表またはビュー。

処理済みの行を除去
次のポーリング試行の前に処理済みの表行をすべて除去する場合に選択します。このクリーンアップは、イテレーター状態が続く場合に実行されます。

イテレーター状態キー
IBM Security Directory Integrator のユーザー・プロパティー・ストアに現在の同期状態を格納するパラメーターの名前を指定します。これは、IBM Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。

「**削除**」ボタンを使用すると、この状態情報がユーザー・プロパティー・ストアから削除されます。

開始位置
このパラメーターは、「**イテレーター状態キー**」がプロパティー・ストアに見つからないか、ブランクのままである場合にのみ考慮されます。このパラメーターは、コネクターが項目の読み取りを開始する、「変更表」内のレコードの位置を示します。このパラメーターでは、1 から EOD (データの終わり -「変更表」内の最終レコードの番号) までの値が許可されます。このパラメーターに指定された入力値が無効である場合は、実行時に該当する例外がスローされます。

状態キーの維持
コネクターの状態をシステム・ストアに保管する方式を制御します。デフォルトおよび推奨の設定は「**サイクルの終わり**」であり、以下のいずれかを選択できます。

読み取り後
RDBMS サーバー変更ログから項目を読み取った時点で、AssemblyLine の残りの部分を続行する前に、システム・ストアを更新します。

サイクルの終わり

AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、システム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。この場合、AssemblyLine 内の特定の時点で、RDBMS 変更検出コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保管する必要があります。

スリープ間隔

IBM Security Directory Integrator が変更表の各ポーリング間で待機する時間 (秒単位) を指定します。

タイムアウト

新規の変更を待機する時間 (秒単位) を指定します。値 **0** (ゼロ) を指定すると、コネクターは無期限に待機します。

コミット

いつデータベース・トランザクションをコミットするかを制御します。オプションは以下のとおりです。

- **各データベース操作後**
- **コネクターのクローズ時**
- **手動**

手動の場合は、ユーザーが `commit()` を呼び出す必要があります。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

変更表の形式

以下の例を使用して、NAME フィールドと EMAIL フィールドが含まれている表から変更を収集する変更表を表示することができます。太字のエレメントは、すべての変更ログ表に共通です。この例の構文は、Oracle 用のものです。

```
IBMSNAP_COMMITSEQ is used as our changelog-nr.
IBMSNAP_OPERATION takes on of the values I (Insert), U (Updated)  or D (Deleted).
CREATE TABLE "SYSTEM"."CCDCHANGELOG"
(
IBMSNAP_COMMITSEQ   RAW(10)   NOT NULL,
IBMSNAP_INTENTSEQ   RAW(10)   NOT NULL,
IBMSNAP_OPERATION   CHAR(1)   NOT NULL,
IBMSNAP_LOGMARKER   DATE      NOT NULL,
NAME   VARCHAR2 ( 80 )  NOT NULL,
EMAIL   VARCHAR2 ( 80 )
)#
```

コネクター内部で使用されている **ibmsnap_commitseq** 列名がデータベース内の実際の列と正確に一致していない場合、RDBMS 変更検出コネクターは機能しません。これは、RDBMS 変更検出コネクターによる反復対象データベース内のデータ・オブジェクトで大/小文字の区別が有効になっている場合にのみ該当します。

これに対処するため、列名はコネクター構成パラメーターとして外部化されています。これにより、DBA はデータベース表と同じ大/小文字で **ibmsnap_commitseq** を容易に設定できます。ただし、「コネクター構成」タブにはこのパラメーターは表示されません。このパラメーターを構成するには、RDBMS 変更検出コネクターの

初期化前 フックに手動でこれを設定する必要があります。これにより、複数の RDBMS 変更検出コネクターが、コネクターによる反復対象の変更表に設定されている列名値のコピーをそれぞれ所有できます。以下に例を示します。

```
myConn.connector.setParam("rdbms.chlog.col","IBMSNAP_COMMITSEQ");
```

これは、**ibmsnap_commitseq** 列の名前を **IBMSNAP_COMMITSEQ** に設定します。デフォルトでは小文字が使用されます。

DB2 での変更表の作成

以下に示す例では、前述したように DB2 データベースでトリガーを作成して、変更表を保持します。

```
connect to your_db

drop table email
drop table ccdemail

create table email ( ¥
 name varchar(80), ¥
 email varchar(80) ¥
)

create table ccdemail ( ¥
 ibmsnap_commitseq integer, ¥
 ibmsnap_intentseq integer, ¥
 ibmsnap_logmarker date, ¥
 ibmsnap_operation char, ¥
 name varchar(80), ¥
 email varchar(80) ¥
)

drop sequence ccdemail_seq
create sequence ccdemail_seq

create trigger t_email_ins after insert on email referencing new as n ¥
 for each row mode db2sql ¥
   INSERT INTO ccdemail VALUES (nextval for ccdemail_seq, 0,
   CURRENT_DATE, 'I', n.name, n.email )

create trigger t_email_del after delete on email referencing old as n ¥
 for each row mode db2sql ¥
   INSERT INTO ccdemail VALUES (nextval for ccdemail_seq, 0,
   CURRENT_DATE, 'D', n.name, n.email )

create trigger t_email_upd after update on email referencing new as n ¥
 for each row mode db2sql ¥
   INSERT INTO ccdemail VALUES (nextval for ccdemail_seq, 0,
   CURRENT_DATE, 'U', n.name, n.email )
```

Oracle での変更表の作成

ユーザー名を「ORAID」とした場合、以下の変更表の例では、NAME フィールドおよび EMAIL フィールドが含まれる表から変更が収集されます。太字体のエレメントは、すべての変更表に共通です。太字の項目は、最後は項目プロパティーとなる追加の制御情報です。以下に示すコード例を使用して、Oracle での変更表の作成方法を詳しく知ることができます。

```
-- create source email table in Oracle.
---This will be the table that the RDBMS Change Detection Connector will detect changes on.
CREATE TABLE ORAID.EMAIL
(
 NAME VARCHAR2(80),
 EMAIL VARCHAR2(80)
);
-- Sequence generators used for Intentseq and commitseq
CREATE SEQUENCE ORAID.SGENERATOR001
MINVALUE 100 INCREMENT BY 1 ORDER;

CREATE SEQUENCE ORAID.SGENERATOR002
```

```
MINVALUE 100 INCREMENT BY 1 ORDER;

-- create change table and index for email table
CREATE TABLE ORAID.CCDEMAIL
(
 IBMSNAP_COMMITSEQ   RAW(10)   NULL,
 IBMSNAP_INTENTSEQ   RAW(10)   NOT NULL,
 IBMSNAP_OPERATION   CHAR(1)   NOT NULL,
 IBMSNAP_LOGMARKER   DATE      NOT NULL,
 NAME VARCHAR2( 80 ),
EMAIL VARCHAR2( 80 )
);

CREATE UNIQUE INDEX ORAID.IXCCDEMAIL ON ORAID.CCDEMAIL
(
IBMSNAP_INTENTSEQ
);

-- create TRIGGER to capture INSERTs into email
CREATE TRIGGER  ORAID.EMAIL_INS_TRIG
AFTER INSERT ON ORAID.EMAIL
FOR EACH ROW BEGIN INSERT INTO ORAID.CCDEMAIL
( NAME,
  EMAIL,
  IBMSNAP_COMMITSEQ,
  IBMSNAP_INTENTSEQ,
  IBMSNAP_OPERATION,
  IBMSNAP_LOGMARKER )
 VALUES (
  :NEW.NAME,
  :NEW.EMAIL,
  LPAD(TO_CHAR(ORAID.SGENERATOR001.NEXTVAL),20,'0'),
  LPAD(TO_CHAR(ORAID.SGENERATOR002.NEXTVAL),20,'0'),
  'I',
  SYSDATE);END;


-- create TRIGGER to capture DELETE ops on email
CREATE TRIGGER  ORAID.EMAIL_DEL_TRIG
AFTER DELETE ON ORAID.EMAIL
FOR EACH ROW BEGIN INSERT INTO ORAID.CCDEMAIL
( NAME,
  EMAIL,
  IBMSNAP_COMMITSEQ,
  IBMSNAP_INTENTSEQ,
  IBMSNAP_OPERATION,
  IBMSNAP_LOGMARKER)
 VALUES
( :OLD.NAME,
  :OLD.EMAIL,
  LPAD(TO_CHAR(ORAID.SGENERATOR001.NEXTVAL),20,'0'),
  LPAD(TO_CHAR(ORAID.SGENERATOR002.NEXTVAL),20,'0'),
  'D',
  SYSDATE);END;


-- create TRIGGER to capture UPDATEs on email
CREATE TRIGGER  ORAID.EMAIL_UPD_TRIG
AFTER UPDATE ON ORAID.EMAIL
FOR EACH ROW BEGIN INSERT INTO ORAID.CCDEMAIL
( NAME,
  EMAIL,
  IBMSNAP_COMMITSEQ,
  IBMSNAP_INTENTSEQ,
  IBMSNAP_OPERATION,
  IBMSNAP_LOGMARKER )
 VALUES (
  :NEW.NAME,
  :NEW.EMAIL,
  LPAD(TO_CHAR(ORAID.SGENERATOR001.NEXTVAL),20,'0'),
  LPAD(TO_CHAR(ORAID.SGENERATOR002.NEXTVAL),20,'0'),
  'U',
  SYSDATE);END;
```

変更表とトリガーの作成 (MS SQL)

以下に示すコード例を使用して、MS SQL での変更表とトリガーの作成方法を詳しく知ることができます。

```
-- Source table msid.email.
-- This will be the table that the RDBMS Change Detection Connector will detect changes on.
CREATE TABLE msid.email
(
 NAME   VARCHAR (80),
 EMAIL   VARCHAR (80)
 );

-- CCD table to capture changes. The RDBMS Change Detection Connector uses the CCD table to capture
-- all the changes in the source table. This table needs to be created in the following format.
CREATE TABLE msid.ccdemail
(
 IBMSNAP_MSTMSTMP timestamp,
 IBMSNAP_COMMITSEQ   BINARY(10)   NOT NULL,
 IBMSNAP_INTENTSEQ   BINARY(10)   NOT NULL,
 IBMSNAP_OPERATION   CHAR(1)      NOT NULL,
 IBMSNAP_LOGMARKER   DATETIME     NOT NULL,
 NAME   VARCHAR (80),
 EMAIL   VARCHAR (80)
);
```

また、E メール表に対して実行される挿入操作、更新操作、および削除操作を収集するトリガーを作成する必要があります。

```
CREATE TRIGGER  msid.email_ins_trig ON msid.email
FOR INSERT AS
BEGIN
 INSERT INTO msid.ccdemail
(NAME,
 EMAIL,
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER )
 SELECT
 NAME,
 EMAIL,
  @@DBTS,
 @@DBTS,
 'I',
 GETDATE() FROM inserted
END;
```

注:：@@DBTS により、現行データベースの現行タイム・スタンプ・データ・タイプの値が戻されます。このタイム・スタンプは、データベース内で固有であることが保証されています。

```
-- creating DELETE trigger to capture delete operations on email table
CREATE TRIGGER  msid.email_del_trig ON msid.email
FOR DELETE AS
BEGIN
 INSERT INTO msid.ccdemail
(
 NAME,
 EMAIL,
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER
)
 SELECT
 NAME,
 EMAIL,
 @@DBTS,
 @@DBTS,
 'D',
 GETDATE() FROM deleted
END;#

-- creating UPDATE trigger to capture update operations on email table
CREATE TRIGGER  msid.email_upd_trig ON msid.email
FOR UPDATE AS
BEGIN
 INSERT INTO msid.ccdemail
(
 NAME,
 EMAIL,
```

```
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER
)
 SELECT
 NAME,
 EMAIL,
 @@DBTS,
 @@DBTS,
 'U',
 GETDATE() FROM inserted
END;
```

Informix での変更表の作成とトリガー

以下に示すコード例を使用して、Informix での変更表の作成方法を詳しく知ることができます。

```
-- Create Source table infxid.email.
-- This will be the table that the RDBMS Change Detection Connector
-- will detect changes on.
CREATE TABLE infxid.email
(
NAME VARCHAR(80),
EMAIL VARCHAR(80)
);

-- create ccdemail table to capture DML operations on email table
CREATE TABLE infxid.ccdemail
(
IBMSNAP_COMMITSEQ   CHAR(10)   NOT NULL,
IBMSNAP_INTENTSEQ   CHAR(10) NOT NULL,
IBMSNAP_OPERATION   CHAR(1)   NOT NULL,
IBMSNAP_LOGMARKER   DATETIME YEAR TO FRACTION(5) NOT NULL,
NAME   VARCHAR(80),
EMAIL   VARCHAR(80)
);

--Create sequence generators
CREATE SEQUENCE infxid.SG1
MINVALUE 100 INCREMENT BY 1;
CREATE SEQUENCE infxid.SG2
MINVALUE 100 INCREMENT BY 1;

-- procedure to capture INSERTs into email table
CREATE PROCEDURE infxid.email_ins_proc
(
NNAME  VARCHAR(80),

NEMAIL  VARCHAR(80)
)

DEFINE VARHEX CHAR(256);

 INSERT INTO infxid.ccdemail
(NAME,
 EMAIL,
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER )
 VALUES
(NNAME,
 NEMAIL,
 infxid.SG1.NEXTVAL,
 infxid.SG2.NEXTVAL,
 'I',
 CURRENT YEAR TO FRACTION(5));END PROCEDURE;

-- now create the trigger for INSERTs into ccdemail
CREATE TRIGGER  infxid.email_ins_trig
 INSERT ON infxid.email
 REFERENCING NEW AS NEW FOR EACH ROW( EXECUTE PROCEDURE
 infxid.email_ins_proc
( NEW.NAME,
  NEW.EMAIL
) );
```

```
-- create procedure to capture DELETEs on email table
CREATE PROCEDURE infxid.email_del_proc
(
 ONAME  VARCHAR(80),
 OEMAIL  VARCHAR(80)
 );

 INSERT INTO infxid.ccdemail
(NAME,
 EMAIL,
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER )
 VALUES
(ONAME,
 OEMAIL,
 infxid.SG1.NEXTVAL,
 infxid.SG2.NEXTVAL,
 'D',
 CURRENT YEAR TO FRACTION(5));END PROCEDURE;

-- create DELETE trigger
CREATE TRIGGER  infxid.email_del_trig
 DELETE ON infxid.email
 REFERENCING OLD AS OLD FOR EACH ROW( EXECUTE PROCEDURE
 infxid.email_del_proc
(OLD.NAME,
 OLD.EMAIL
) );

-- create PROCEDURE to capture updates
CREATE PROCEDURE infxid.email_upd_proc
(
 NNAME  VARCHAR(80),
 NEMAIL  VARCHAR(80)
);
 INSERT INTO infxid.ccdemail
(NAME,
 EMAIL,
 IBMSNAP_COMMITSEQ,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER)
 VALUES
(NNAME,
 NEMAIL,
 infxid.SG1.NEXTVAL,
 infxid.SG2.NEXTVAL,
 'U',
 CURRENT YEAR TO FRACTION(5));END PROCEDURE;

-- create TRIGGER to capture UPDATES
CREATE TRIGGER  infxid.email_upd_trig
 UPDATE ON infxid.email
 REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW( EXECUTE PROCEDURE
 infxid.email_upd_proc
(NEW.NAME,
 NEW.EMAIL
 ) );
```

変更表とトリガーの作成 (SYBASE)

以下に示すコード例を使用して、SYBASE での変更表の作成方法を詳しく知ることができます。

```
-- Create Source table sybid.email.
-- This will be the table that the RDBMS Change Detection Connector will detect changes on.
CREATE TABLE sybid.EMAIL
(
 NAME   VARCHAR (80),
 EMAIL   VARCHAR (80)
)

-- Create CCD table to captures changes on email table
CREATE TABLE sybid.CCDEMAIL
(
```

```
 IBMSNAP_TMSTMP TIMESTAMP,
 IBMSNAP_COMMITSEQ  NUMERIC(10)   IDENTITY,
 IBMSNAP_INTENTSEQ   BINARY(10)   NOT NULL,
 IBMSNAP_OPERATION   CHAR(1)      NOT NULL,
 IBMSNAP_LOGMARKER   DATETIME     NOT NULL,
 NAME   VARCHAR(80),
 EMAIL   VARCHAR(80)
)

-- Create TRIGGER to capture INSERTs on email table
CREATE TRIGGER  sybid.EMAIL_INS_TRIG ON sybid.EMAIL
FOR INSERT AS
BEGIN
 INSERT INTO sybid.CCDEMAIL
(NAME,
 EMAIL,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER )
 SELECT
 NAME,
 EMAIL,
@@DBTS,
 'I',
 GETDATE() FROM inserted
END

NOTE: @@DBTS is a special database variable that yields the next database timestamp value

-- create TRIGGER to captures DELETE ops on EMAIL table
CREATE TRIGGER  sybid.EMAIL_DEL_TRIG ON sybid.EMAIL
FOR DELETE AS
BEGIN
 INSERT INTO sybid.CCDEMAIL
(
 NAME,
 EMAIL,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER
)
 SELECT
 NAME,
 EMAIL,
 @@DBTS,
 'D',
 GETDATE() FROM deleted
END

-- create TRIGGER to capture UPDATEs on email
CREATE TRIGGER  sybid.EMAIL_UPD_TRIG ON sybid.EMAIL
FOR UPDATE AS
BEGIN
  DECLARE @COUNTER INT
  SELECT @COUNTER=COUNT(*) FROM deleted
  IF @COUNTER>1
  BEGIN
  DECLARE @NAME  VARCHAR ( 80 )
  DECLARE @EMAIL  VARCHAR ( 80 )
  DECLARE insertedrows CURSOR FOR SELECT * FROM inserted
  OPEN insertedrows
  WHILE 1=1 BEGIN
  FETCH insertedrows INTO
  @NAME,
  @EMAIL
   IF @@fetch_status<>0 BREAK
  ELSE INSERT INTO sybid.CCDEMAIL
 (
  NAME,
  EMAIL,
  IBMSNAP_INTENTSEQ,
  IBMSNAP_OPERATION,
  IBMSNAP_LOGMARKER
 )
 VALUES
 (
  @NAME,
  @EMAIL,
  @@DBTS,
 'U',
```

```
GETDATE()
)
END
 DEALLOCATE insertedrows
END ELSE INSERT INTO sybid.CCDEMAIL(
 NAME,
 EMAIL,
 IBMSNAP_INTENTSEQ,
 IBMSNAP_OPERATION,
 IBMSNAP_LOGMARKER
)
SELECT
 I.NAME,
 I.EMAIL,
 @@DBTS,
'U',
GETDATE() FROM inserted I
END
```

例

RDBMS 変更検出コネクターについて詳しく知るには、以下に示すコード例を使用できます。

この例は、ディレクトリー *TDI_install_dir*/examples/RDBMS の下に記載されています。この例では、リモート・データベースで表の変更を検出する RDBMS 変更検出コネクターの機能が示されています。現在の例は、IBM DB2 での動作だけを対象に設計されています。

SCIM コネクター

System for Cross-Domain Identity Management (SCIM) プロトコルは、Web 上の ID データのプロビジョニングと管理を行うためのアプリケーション・レベルの REST プロトコルです。ここに記載されている情報を使用することで、SCIM コネクターについて詳しく知ることができます。

このプロトコルは、核となる ID リソース (ユーザーおよびグループ) とカスタム・リソース拡張の作成、変更、取得、ディスカバリーをサポートします。

SCIM コネクターは、JavaScript と HTTP クライアント・コネクターを使用して SCIM プロトコルを実装します。

構成

ここに記載されているパラメーターを使用することで、SCIM コネクターを構成できるようになります。

SCIM コネクターでは、以下のパラメーターが使用されます。

SCIM サーバー URL
SCIM サーバーの URL を指定します。このパラメーターは必須です。

リソース・エンドポイント
リソース・エンドポイントを指定します。コア SCIM スキーマから「ユーザー」または「グループ」を選択することも、ユーザー定義のリソース・エンドポイントを選択することもできます。

ユーザー名
SCIM サーバーで HTTP 基本認証を行うためにコネクターが使用するユーザー名を指定します。

パスワード
指定されたユーザー名のパスワードを指定します。

「拡張」セクションでは、以下のパラメーターを使用することができます。

更新メソッド
SCIM サーバーでエントリーを更新する際に使用するメソッドを指定します。以下のオプションから選択することができます。

- **指定された項目でのパッチ**: PATCH メソッドを使用して項目を SCIM サーバーに送信します。
- **すべての項目を置換**: PUT メソッドを使用して項目を SCIM サーバーに送信します。

属性フィルター
サーバーが返す必要のある属性のコンマ区切りリストを指定します。このパラメーターに値を指定しなかった場合、デフォルトはフィルターなし (すべてのリソースを受け取る) になります。

プロキシー・サーバー
プロキシー・サーバーを使用して接続する場合は、ホスト・プロキシー・サーバーとポート番号 (*proxyhost:port*) を指定します。このパラメーターに値を指定しなかった場合、プロキシー・サーバーは使用されません。

プロキシー・サーバー・ユーザー名
使用するプロキシー・サーバーで認証が必要な場合は、プロキシー・サーバーに対して認証を行うためのユーザー名を指定します。

プロキシー・サーバーのパスワード
指定されたプロキシー・サーバー・ユーザー名のパスワードを指定します。

ソート基準
SCIM サーバーでソート機能を実装する場合に、結果のソートで使用される属性を指定します。このパラメーターに値を指定しなかった場合、デフォルトはソートなしになります。

ソート順
ソート順を昇順または降順として指定します。このパラメーターが使用されるのは、ソート機能を実装する場合だけです。このパラメーターに値を指定しなかった場合、昇順で結果がソートされます。

スクリプト
SCIM コネクターの稼働方法を制御します。スクリプトを変更すると予期しない結果が発生する可能性があるため、スクリプトを変更する場合は注意が必要です。

関連情報

『*Federated Directory Server の管理*』セクションの『System for Cross-domain Identity Management』

SCIM Web サイト (www.simplecloud.info)

スクリプト・コネクター

スクリプト・コネクターを使用すると、JavaScript で独自のコネクターを作成できます。

スクリプト・コネクターが機能するためには、いくつかの関数をインプリメントする必要があります。このコネクターを繰り返しのためにのみ使用する (例えば、検索や更新ではなく読み取りを行う) ことを予定している場合は、2 つの関数のみで目的を果たすことができます。このコネクターを完全な機能を備えたコネクターとして使用する場合は、すべての関数をインプリメントする必要があります。これらの関数は、パラメーターを使用しません。ホストとして稼働するコネクターとスクリプトの間のデータの受け渡しには、事前定義済みのオブジェクトが使用されます。これらの定義済みオブジェクトの 1 つに**結果**オブジェクトがあり、これは状況情報を伝達するために使用されます。いずれかの関数に入る時点では、**状況**フィールドは**通常**に設定されており、その場合ホスト・コネクターは呼び出しを続行します。**入力終了**または**エラー**を通知するには、このオブジェクト内の**状況**フィールドおよび**メッセージ**・フィールドを設定します。他の 2 つのスクリプト・オブジェクト、**項目**オブジェクトと**検索**オブジェクトは、関数に入るときに定義されます。

注: スクリプト・コネクターまたはパーサーを変更すると、スクリプトは、それが入っているライブラリーから構成ファイルにコピーされます。これには、スクリプトをカスタマイズできるという利点がありますが、AssemblyLine が新規バージョンを認識しないという欠点もあります。

一つの次善策としては、AssemblyLine から旧スクリプト・コネクターを除去してから再導入する方法があります。

汎用コンテナーの場合、スクリプト・コネクターはユーザー自身が JavaScript で作成するものであり、スクリプトで作成したモードを提供します。「」の『JavaScript コネクター』を参照してください。

サポートされるモードのリストについては、 7 ページの『サポートされるモードの欄の凡例』を参照してください。

スクリプト・ベースのコネクターでは、IBM Security Directory Integrator と同じライブラリーに対して直接 Java 呼び出しを実行すると、問題が発生する原因になることがあります。新バージョンの IBM Security Directory Integrator によりライブラリーが (異なるセマンティクスにより) 更新されたか、または最後にコネクターを使用した時点以降にライブラリーがアップグレードされている可能性があります。

定義済みスクリプト・オブジェクト

以下のセクションで、定義済みのスクリプト・オブジェクトを参照できます。

main 実行中の構成インスタンス (RS オブジェクト)。

task このコネクターが含まれている AssemblyLine。

system UserFunctions オブジェクト。

結果オブジェクト

setStatus (コード)

- **0**: 入力終了

- **1**: 状況 OK
- **2**: エラー

setMessage (テキスト)
エラー・メッセージ

構成オブジェクト
このオブジェクトにより、この AL コンポーネントの構成、およびその入出力スキーマにアクセスできます。このオブジェクトの getSchema() メソッドには 1 つのブール値パラメーターがあり、このパラメーターが *true* の場合は入力スキーマを戻し、false の場合はユーザー に対して出力スキーマが取得されることに注意してください。

項目オブジェクト

項目オブジェクトは、コネクター (または関数コンポーネントのスクリプトを作成する場合は関数) の **conn** 項目に対応しています。

詳しくは、 679 ページの『項目オブジェクト』を参照してください。

検索オブジェクト
検索オブジェクトにより、searchCriteria オブジェクト (リンク基準設定に基づいて作成されたオブジェクト) にアクセスできます。詳しくは、 682 ページの『検索 (基準) オブジェクト』を参照してください。

コネクター・オブジェクト

このコネクターへの参照です。

これは、以下のようなコードを使用して、関数 findEntry() で検出された複数の項目を戻す場合などに役立ちます。

```
function findEntry() {
 connector.clearFindEntries();
 // Use the search object fo find Entries, and
 for ( entry = all Entries found) {
  connector.addFindEntry(entry)
 }
 if (connector.getFindEntryCount() == 1)
  result.setStatus(1);
 else
  result.setStatus(0);
}
```

関数

スクリプト・コネクターによってインプリメントできる関数を以下に示します。関数にはまったく呼び出される可能性のないものもいくつかありますが、このような関数は、呼び出し元にその関数がサポートされていないことを知らせるエラー通知コードとともに挿入しておくことをお勧めします。

initialize
この関数は、コネクターを初期化します。他のどの関数よりも先に呼び出され、基本パラメーターを初期化したり接続を確立するコードが組み込まれている必要があります。

selectEntries
この関数は、順次読み取り用としてコネクターを準備するために呼び出されます。このコネクターが呼び出されるのは、通常、コネクターが AssemblyLine 内でイテレーターとして使用される場合です。

getNextEntry

この関数は、入力セット内の次のエントリーから属性と値を項目オブジェクトに取り込みます。戻す項目がなくなると、コネクターは**結果**オブジェクトを使用して、呼び出し元に入力終了を知らせます。

findEntry

findEntry 関数は、接続されているシステム内で、**検索**オブジェクトに指定されている基準と一致する項目を検索するために呼び出されます。コネクターが一致項目を 1 つのみ検出した場合は、コネクターは**項目**オブジェクトにデータを取り込みます。該当項目が 1 つも見つからない場合は、コネクターは、**結果**オブジェクトの中に、項目を検出できなかったことを知らせるエラー・コードを設定します。一致する項目が複数検出された場合は、コネクターは重複項目の配列を取り込むことができます。それ以外の場合は、該当項目が見つからない場合と同じプロシージャーが行われます。

modEntry

この関数は、接続されているシステム内の既存項目を変更するために呼び出されます。新しい項目データは**項目**オブジェクトから提供され、どの項目を変更するかは**検索**オブジェクトで指定されます。一部のコネクターは、通知なしで**検索**オブジェクトを無視し、**項目**オブジェクトを使用してどの項目を変更するかを決定することがあります。

putEntry

この関数は、接続されているシステムに**項目**オブジェクトを追加します。

deleteEntry

この関数は、接続されているシステム内の既存項目を削除するために呼び出されます。削除する項目は**検索**オブジェクトに指定します。一部のコネクターは、通知なしで**検索**オブジェクトを無視し、**項目**オブジェクトを使用してどの項目を削除するかを決定することがあります。

queryReply

この関数は、コネクターを Call/Reply モードで使用する場合に呼び出されます。

querySchema

querySchema() 関数は、このコネクターのスキーマを検索するために使用されます。これをインプリメントする場合は、見つかった列/属性のそれぞれについて、**項目**オブジェクトのベクトルを戻します。querySchema() 関数は、属性マップで「オープン/照会」を実行する場合にのみ呼び出されます (高速ディスカバリー・ボタンをクリックしても呼び出されません)。

スキーマ・ディスカバリーをサポートするため、スクリプト・コネクターまたは関数コンポーネントには次のようなコードを組み込むことができます。

```
function querySchema() {
  config.getSchema(true).newItem("name-in");
  config.getSchema(true).newItem("address-in");
  config.getSchema(false).newItem("name-out");
  config.getSchema(false).newItem("address-out");
 }
```

これにより、入力スキーマと出力スキーマにそれぞれ 2 つの項目が作成されます。詳しくは、SchemaConfig および SchemaItemConfig API (Javadoc) を調べてください。

terminate
この関数は、コネクターがそのタスクを完了したときに呼び出されます。処理中に取得されたリソース (例えば、ロックや接続など) を開放するコードが組み込まれている必要があります。

各種モードに基づいて、インプリメントする必要がある関数の最小要件があります。

表 32. 必須の関数

モード	インプリメントする必要がある関数
イテレーター	selectEntries() getNextEntry()
AddOnly	putEntry()
ルックアップ	findEntry()
削除	findEntry() deleteEntry()
更新	findEntry() putEntry() modEntry()
CallReply	queryReply()

構成

ここに記載されているパラメーターを使用することで、スクリプト・コネクターを構成できるようになります。

スクリプトの編集...
このボタンは、独自のスクリプト・コードを作成できるウィンドウを開きます。空の骨組みが表示されます。

グローバル状態を保持
このパラメーターがチェックされている場合 (デフォルト)、スクリプトで定義されたグローバル変数がコネクター終了後も保持され、コネクターが再度初期化されたとき最後の値が表示されます。

注: このパラメーターがチェックされていると (デフォルト)、このコネクターの動作が IBM Security Directory Integrator バージョン 7.1.1 以降で変更されます。コネクターとともにグローバル変数を再初期化する必要がある場合、このパラメーターのチェックを解除、または `initialize()` メソッド内部にこれらの変数を設定する必要があります。

外部ファイル
実行時に外部スクリプト・ファイルを組み込む場合は、それらのファイルを指定します。ファイルは 1 行に 1 つずつ指定してください。これらのファイルは、スクリプトの前に開始されます。

グローバル・スクリプトの組み込み
スクリプト・ライブラリーからグローバル・スクリプトを組み込みます。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

例

スクリプト・コネクターについてさらに詳しく知るには、以下に示す例を使用できます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/script_connector ディレクトリーにあります。

関連情報

461 ページの『スクリプト・パーサー』,
519 ページの『スクリプト記述関数コンポーネント』,
「リファレンス」の『JavaScript コネクター』

サーバー通知コネクター

サーバー通知コネクターは、IBM Security Directory Integrator 通知システムへのインターフェースです。以下に示す情報を使用して、これについて詳しく知ることができます。

このコネクターは、サーバー API 通知を listen、報告、および発行します。このコネクターには、IBM Security Directory Integrator サーバーで実行される各種プロセス (AssemblyLine 停止および開始プロセスのイベントなど) をモニターする機能と、カスタム・サーバー通知を発行する機能があります。

サーバー通知コネクターでは、イテレーター・モード、および AddOnly モードがサポートされています。

イテレーター・モード

イテレーター・モードのサーバー通知コネクターには、それがどのように構成されているかに応じて、ローカルまたはリモートのいずれかのサーバー API 通知を listen し、報告する機能がありますが、同一のコネクター・セッションで両方に対して実行することはできません。

「ローカル」接続タイプは、通知を送信する IBM Security Directory Integrator サーバーと同一の JVM でコネクターが実行されている場合に使用します。

「リモート」接続タイプは、異なる JVM で実行されているリモートの IBM Security Directory Integrator サーバーにコネクターが接続する場合に使用します。

AddOnly モード

AddOnly モードのコネクターは、ローカルまたはリモートのいずれかのサーバー API セッションを使用して、サーバー API のカスタム (つまり、ユーザー定義の) 通知を送信しますが、同一のコネクター・セッションで両方には送信しません。

「ローカル」接続タイプは、通知を送信する IBM Security Directory Integrator サーバーと同一の JVM でコネクターが実行されている場合に使用します。

「リモート」接続タイプは、異なる JVM で実行されているリモートの IBM Security Directory Integrator サーバーにコネクターが接続する場合に使用します。

通知オブジェクトの作成に必要なデータは、AssemblyLine によってコネクターに受け渡される *conn* 項目から取得されます。コネクターはこの項目にある固定名の属性を検索してそれらの値を取得し、それらの値を使用して通知オブジェクトを作成し、サーバー API を使用してこの通知オブジェクトを発行します。固定名の属性について詳しくは、『スキーマ』のセクションを参照してください。

各サーバー API 通知では対応する JMX 通知も発行されるため、AddOnly モードのサーバー通知コネクターは間接的に JMX 通知も送信します。カスタム通知について詳しくは、318 ページの『スキーマ』のセクションを参照してください。

暗号化

以下に示す情報を使用して、サーバー通知コネクターを使用した暗号化について理解することができます。

サーバー通知コネクターでは、接続タイプが「**リモート**」に設定されている場合に Secure Sockets Layer (SSL) を使用できます。リモート IBM Security Directory Integrator サーバーで SSL 接続のみが受け入れられる場合、ローカル IBM Security Directory Integrator サーバーでトラストストアが適切に構成されていれば、サーバー通知コネクターは SSL 接続を自動的に確立します。SSL 使用時には、コネクターはサーバー API SSL セッションを使用します。このセッションは、SSL を介して RMI を実行します。

トラスト・ストア

リモート IBM Security Directory Integrator サーバーが通知を発行すると、ローカル IBM Security Directory Integrator サーバーとの新規 SSL 接続が確立され、この新規 SSL 接続のセッションを確立するためにローカル IBM Security Directory Integrator サーバーは (トラストストアを使用して) リモート IBM Security Directory Integrator サーバー SSL 証明書を信頼する必要があるため、ローカル IBM Security Directory Integrator サーバー上にトラストストアが必要です。トラストストアを構成するには、ファイル `global.properties` または `solution.properties` 内にあるプロパティー `javax.net.ssl.trustStore`、`javax.net.ssl.trustStorePassword` および `javax.net.ssl.trustStoreType` に適切な値を設定します。

認証

以降の各セクションに示す情報を使用して、認証について詳しく知ることができます。

SSL 認証

サーバー通知コネクターを使用する場合は、クライアント SSL 証明書を使用して認証を行うことができます。

これを実行できるのは、 SSL を使用し、SSL クライアント証明書がクライアント上で必要となるようにリモートの IBM Security Directory Integrator サーバー API が構成されている場合だけです。ローカル IBM Security Directory Integrator サーバーでトラスト・ストアが適切に構成されている必要があります。

ユーザー名とパスワードによる認証

使用するユーザー名とパスワードを、コネクター・パラメーターとして設定することができます。この場合、コネクターはサーバー API のユーザー名/パスワード認証メカニズムを使用します。

サーバー通知コネクターでは、サーバー API のユーザー名/パスワード認証メカニズムを使用できます。 SSL が使用されていて、ユーザー名とパスワードがコネクター・パラメーターとして指定されている場合は、コネクターはリモート IBM Security Directory Integrator サーバーに対する認証に SSL クライアント証明書ではなく、指定されたユーザー名とパスワードを使用します。

構成

ここに記載されているパラメーターを使用することで、サーバー通知コネクターを構成できるようになります。

接続タイプ

サーバー通知コネクターがローカルまたはリモートのいずれのサーバー API 通知を listen および発行するかを指定します。このパラメーターの有効値は「remote」と「local」です。「local」を選択すると、コネクターはローカル Java 仮想マシン内でのみ通知を listen および発行します。「リモート」を選択すると、コネクターはリモート IBM Security Directory Integrator サーバー・システムに接続し、そのリモート・システムの Java 仮想マシン内で通知を登録および発行します。

RMI URL

リモート IBM Security Directory Integrator サーバー・システムへの接続に使用するリモート・メソッド呼び出し (RMI) URL を指定します。このパラメーターは、 connectionType パラメーターが「remote」に設定されている場合にのみ有効です。このパラメーターの値の例を以下に示します。

```
rmi://127.0.0.1:1099/SessionFactory
```

ユーザー名

コネクターが IBM Security Directory Integrator サーバーに対する認証に使用するユーザー名を指定します。このパラメーターは、「**接続タイプ**」パラメーターが「*remote*」に設定されている場合にのみ考慮されます。

パスワード

このパラメーターでは、コネクターが IBM Security Directory Integrator サーバーに対する認証に使用するパスワードを指定します。このパラメーターは、「**接続タイプ**」パラメーターが「*remote*」に設定されている場合にのみ考慮されます。

構成インスタンス ID のフィルター

コネクターがイベント通知をフィルタリングするために使用する構成インスタンス ID を指定します。このパラメーターを指定すると、コネクターはこ

の構成インスタンス ID がある通知のみを報告します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

通知 ID のフィルター
コネクターがイベント通知をフィルタリングするために使用する通知 ID を指定します。このパラメーターを指定すると、コネクターは指定された通知 ID が含まれている通知のみを報告します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

タイムアウト (秒)
通知を待機する最大秒数を指定します。このタイムアウト期間が経過すると、コネクターは終了します。このパラメーターの値を「0」に設定すると、コネクターは無期限に待機します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

すべてのサーバー API イベントを受信する
コネクターが「di.*」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

すべての構成インスタンス・イベントを受信する
コネクターが「di.ci.*」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

構成インスタンス開始イベントを受信する
コネクターが「di.ci.start」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

構成インスタンス停止イベントを受信する
コネクターが「di.ci.stop」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

構成更新イベントを受信する
コネクターが「di.ci.file.updated」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

すべての AssemblyLine イベントを受信する
コネクターが「di.al.*」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

AssemblyLine 開始イベントを受信する
コネクターが「di.al.start」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

AssemblyLine 停止イベントを受信する
コネクターが「di.al.stop」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

サーバー・シャットダウン・イベントを受信する
コネクターが「di.server.stop」通知を受信するかどうかを指定します。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

カスタム通知を使用する
コネクターが追加の通知またはカスタム通知を受信するかどうかを指定します。これをチェックすると、追加/カスタムの通知を「**カスタム通知タイ**

プ」のコネクター・パラメーターに指定できます。このパラメーターは、コネクター・モードがイテレーターの場合にのみ有効です。

カスタム通知タイプ
サーバー通知コネクターが listen および報告する、追加またはカスタムのサーバー API 通知の通知タイプを指定します。それぞれの通知タイプは 1 つずつ個別の行に入力してください。このパラメーターは、「**カスタム通知を使用する**」パラメーターが true である場合にのみ有効であり、コネクター・モードがイテレーターである場合にのみ考慮されます。

デバッグ
デバッグ・メッセージを有効にします。このパラメーターは、すべての IBM Security Directory Integrator コンポーネントに対してグローバルに定義されます。

スキーマ

ここに記載されている情報を使用することで、スキーマについて知ることができます。

イテレーター・モード

イテレーター・モードのサーバー通知コネクターでは、入力属性マップに以下の属性が設定されます。

event.rawNotification
通知イベント・オブジェクト (com.ibm.di.api.DIEvent)。このオブジェクトには、IBM Security Directory Integrator サーバーのコアで生成されたサーバー API イベントに関する完全な情報が含まれます。

event.type
通知イベントのタイプ (java.lang.String)。この属性により、発生事項が指定されます。この属性の値の例には、「di.al.start」があります。

event.id
通知 ID (java.lang. String)。この属性により、イベントのソース (つまり、このイベントが発生した IBM Security Directory Integrator のコンポーネント) が指定されます。この属性の値の例には、「AssemblyLines/tcp」があります。

event.userData
通知のユーザー・データ (java.lang.Object)。オプションのユーザー定義の情報であり、その目的は、イベントの発生理由、イベントの発生場所など、イベントに関する詳細情報を伝達することです。この属性は、このようなユーザー・データがイベント生成時に実際に受け渡された場合にのみ使用可能です。

AssemblyLine のイベントの場合、この属性には AssemblyLine の固有コードが含まれ、このコードは、そのイベントを生成した AssemblyLine インスタンスを明白に特定するために使用できます (例:「1709375019」)。

userData オブジェクトのタイプが com.ibm.di.entry.Entry である場合、生成される出力項目にもその属性がマップされるため、Assembly Line でその属

性に直接アクセスでき (例: `conn.getAttribute ("event.userData.hostname")`)、その属性を使用可能にするためにスクリプトを追加する必要はありません。

event.configInstanceId
構成インスタンス ID (java.lang.String)。このイベントを開始した、ロード済みで実行中の構成インスタンスの ID。この属性の値の例には、「C__dev_assembly_TCPServer.xml」があります。

event.dateCreated
この通知が作成された時間および日付が日付オブジェクトに格納されます (java.util.Date)。

AddOnly モード

AddOnly モードのサーバー通知コネクターでは、出力属性マップから以下の属性が受信されます。

event.type
通知イベントのタイプ (java.lang.String)。この属性により、カスタム通知でシグナル通知するイベントが指定されます。これはユーザー定義のイベントであるため、任意のストリングにできます。この属性の値の例には、「myAL.DBRecord.Committed」があります。この属性は、conn 項目に必須です。この属性が conn 項目にない場合、コネクターは例外をスローします。

この属性に対してユーザーが指定した値には、通知オブジェクトの作成時にコネクターによって、「user.」の接頭部が付加されます。例えば、ユーザーによって受け渡されるタイプが「process.X.completed」である場合、ブロードキャストされるイベントのタイプは「user.process.X.completed」になります。

event.id
通知 ID (java.lang. String)。この属性により、イベントのソース (つまり、このイベントが発生した IBM Security Directory Integrator のコンポーネント) が指定されます。これはユーザー定義のイベントであるため、「myAssemblyLine_5」などの任意のストリングにできます。この属性は、conn 項目に必須です。この属性が conn 項目にない場合、コネクターは例外をスローします。

event.userData
通知のユーザー・データ (java.lang.Object)。オプションのユーザー定義の情報であり、その目的は、イベントの発生理由、イベントの発生場所など、イベントに関する詳細情報を伝達することです。この属性は、*conn* 項目では任意指定です。

シンプル Tpae IF コネクター

共通基盤 (Tpae) (「基本サービス」とも呼ばれる) はコア Java クラスのコレクションであり、Java アプリケーション作成の基盤として使用されます。ここに記載されている情報とリンクを使用することで、これについて詳しく知ることができます。

統合フレームワークは Tpae の機能の 1 つで、標準統合オブジェクト (オブジェクト構造およびインターフェース) と、アウトバウンド/インバウンド・オブジェクトが含まれています。シンプル Tpae IF コネクターは、IBM Security Directory Integrator を Tpae 統合フレームワークに接続して、情報を交換します。

シンプル Tpae IF コネクターは、統合フレームワークに対して読み取りおよび書き込みを行います。Maximo® ビジネス・オブジェクト (MBO) がサポートされ、統合オブジェクトを使用して処理されます。このコネクターは、インポートまたはエクスポートされるオブジェクトを検証するために、MBO 層を使用します。シンプル Tpae IF コネクターは、各種 AssemblyLine モード (イテレーター、AddOnly、更新、ルックアップ、削除など) で使用できます。

共通基盤

Tpae アーキテクチャーを使用することには利点があります。多くの Java アプリケーションで使用されるコア機能を、アプリケーションでコーディングする必要がないことです。

各アプリケーションは、コア機能をコーディングするのではなく、基本クラスに依存してコア機能を提供します。Tpae 層はミドルウェアであり、ユーザーによってアプリケーションとして直接使用されることはありません。主な機能は、以下のとおりです。

- 勘定科目一覧
- サイト
- 組織
- レポート
- ユーザー
- セキュリティー・グループ
- ワークフロー
- 管理アプリケーション
- 構成アプリケーション

いくつか Tpae アプリケーションを以下に示します。

- Maximo Asset Management (MAM)
- Tivoli Service Request Manager® (TSRM)
- Tivoli Asset Management for IT (TAMIT)
- Change and Configuration Management Database (CCMDB)

Tpae に基づいて作成されたアプリケーションは、ユーザー、役割、およびグループの管理に、基盤システムのセキュリティー・ツールを使用します。Tpae の主要な機能を以下に示します。

- 統合フレームワーク (旧 Maximo Enterprise Adapter (MEA))
- カスタム・クーロン・タスクのスケジューリングおよび処理
- カスタム・ワークフロー処理
- カスタム E メール・リスナー
- Birt J2EE レポート作成

- アプリケーション・サーバー・セキュリティー統合 (LDAP)
- 動的データ取得

統合フレームワーク

統合フレームワーク (IF) は、システムとフレームワーク・アプリケーション間の統合を容易にする一連のアプリケーションです。ここに記載されている情報を通じてこれについて詳しく知ることができます。

IF は、基本共通基盤の一部であり、Tpae を使用するすべての主要製品で使用可能です。例えば、MAM、および SRM です。

IF は Tpae の一部です。これは XML ベースの統合フレームワークであり、XML ファイルと区切りファイルの両方がサポートされます。IF により、Tpae 共通アーキテクチャーを使用してアプリケーション・サーバーで実行されているアプリケーションと外部システムの間でのデータの同期および統合が可能になります。IF により、各種通信プロトコルを使用して、データを同期的および非同期的に交換できます。

IF は、一連のアウトバウンド (チャネル) およびインバウンド (サービス) 統合インターフェースを備えています。以下のような複数の通信方式がサポートされます。

- XML ファイル/フラット・ファイル
- データベース・インターフェース・テーブル
- XML over HTTP
- Web サービス
- Java Message Service (JMS) メッセージング

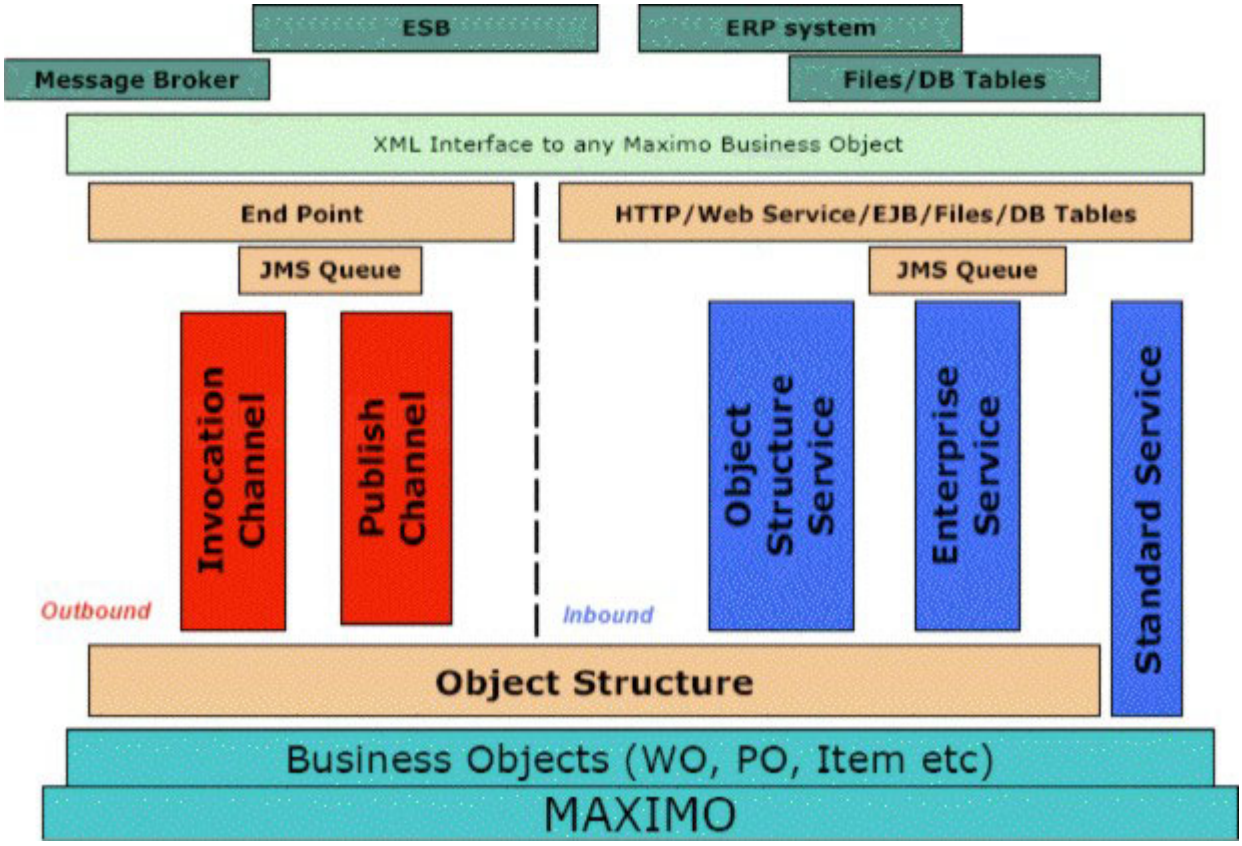


図 1. Tpae 統合フレームワーク

以下の IF サービスを使用して、Tpae 製品と外部システムを統合できます。

標準サービス
細分化されたオブジェクト固有サービスを提供します。例えば、資産の移動や状況変更です。このサービスは、アノテーション付きの Java メソッドでのみ使用可能です。標準サービスを追加するには、コードの変更が必要になります。

オブジェクト構造サービス
一般的な挿入、更新、削除、および照会の機能を提供します。このサービスは、キューイングおよびカスタマイズの層が不要な場合に使用します。

エンタープライズ・サービス
一般的な挿入、更新、削除、および照会の機能を提供します。このサービスは、キューイングおよびカスタマイズ (Java/XSL) の層が必要な場合に使用します。

IF により、データをアプリケーション (MAM または CCMDB) との間で入力/出力したり、データを外部システムとの間で入力/出力したりすることができます。シンプル Tpae IF コネクターは、IF 機能を使用してデータを統合します。

Maximo ビジネス・オブジェクト

Maximo ビジネス・オブジェクト (MBO) は、一連のフィールドおよびビジネス・ルールを定義し、1 つ以上の Maximo データベース表を更新します。

複数のオブジェクト構造が同じ MBO を使用している場合は、各構造定義でこれらの詳細が繰り返されます。

IF は MBO を使用して、基盤となっている表に対して、データを抽出したり、データをロードしたりします。MBO は、受信データに対して 1 つ以上のビジネス・ルールを適用します。データに対するルールの適用が失敗した場合は、MBO は必要な操作を実行できません。例えば、発注書の状況の変更や新規ワークフロー・プロセスの挿入などです。 MBO 層は、データを Tpae に統合する際に使用されます。

MIF オブジェクト構造

MIF オブジェクト構造 (MOS) は、外部システムとの間で送受信される統合メッセージの内容を構成する 1 つ以上のサブレコードで構成されます。

各サブレコードには、MBO からのフィールドが含まれます。 MBO と、対応するサブレコードの名前は同じです。MOS には、任意の数のサブレコードを含めることができます。オブジェクト構造は階層的にして、オブジェクト構造内のサブレコード・ペア間の親子関係を表すことができます。最上位の MBO は、1 次オブジェクトまたはルート MBO と呼びます。

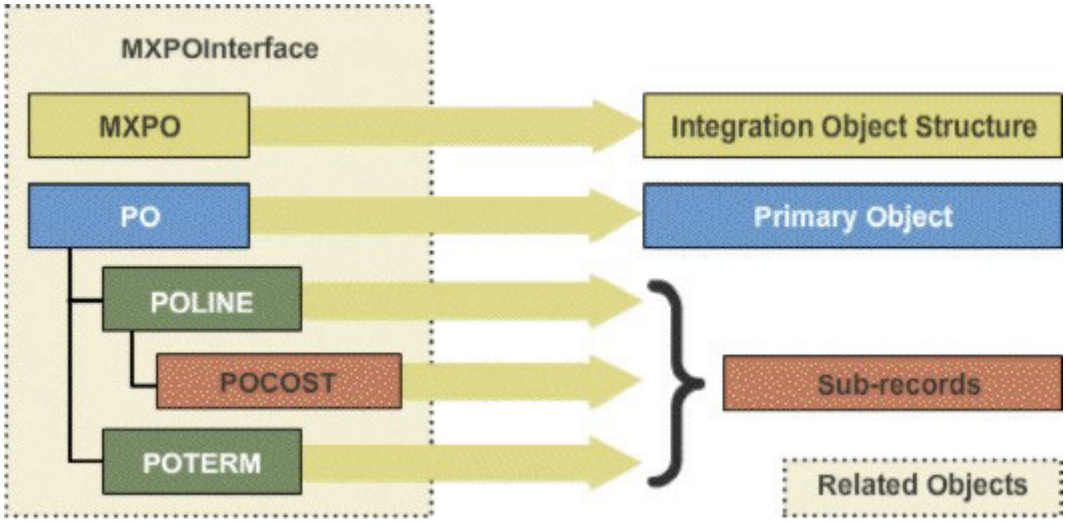


図 2. 発注書の事前定義オブジェクト構造

オブジェクト構造は、アウトバウンドおよびインバウンドのアプリケーション・データの処理のために IF が使用する共通データ層です。単一のオブジェクト構造のメッセージ内容を使用して、インバウンドとアウトバウンド両方のメッセージ処理をサポートできます。標準サービスおよび REST API (322 ページの図 1 を参照) は、オブジェクト構造層を通過しません。

注: Maximo 6 では、オブジェクト構造は、統合オブジェクトと呼ばれていました。Maximo 7 では、オブジェクト構造は、統合オブジェクト構造または MIF オブジェクト構造 (MOS) と呼びます。

コネクターの使用

シンプル Tpae IF コネクターは XML over HTTP を使用して、データを IF に統合します。その統合には、2 つのタイプがあります。それぞれのタイプについて、以下のセクションで説明します。

- 『オブジェクト構造サービスの使用』
- 325 ページの『エンタープライズ・サービスの使用』

オブジェクト構造サービスの使用

オブジェクト構造サービスは、指定したオブジェクト構造に対して提供された操作を実行する機能を備えています。ここに記載されている情報を使用することで、オブジェクト構造サービスを使用できるようになります。

- **更新**
- **照会**
- **作成**
- **同期**
- **削除**
- パブリッシュ
- 呼び出し

シンプル Tpae IF コネクターでは、上のリストで示されている操作がサポートされます。オブジェクト構造サービスは、作成、読み取り、更新、または削除の各操作に使用でき、コネクターのデフォルト動作になっています。これらのサービスは、オブジェクト構造アプリケーションによって管理されます。Maximo には、複数の事前定義オブジェクト構造 (例えば、以下の MXASSET) が付属しています。

表 33. MXASSET の例

MBO	親オブジェクト	ロケーション・パス	関係
ASSET		ASSET	
ASSETMETER	ASSET	ASSET/ASSETMETER	INT_ASSETMETER
ASSETUSERCUST	ASSET	ASSET/ASSETUSERCUST	ASSETUSERCUST
ASSETSPEC	ASSET	ASSET/ASSETSPEC	ASSETSPECCLASS

コネクターは、資産を読み取る際に、関係がネストされたエレメントとして表された以下の XML ファイルのような構造を受信します。

```
<ASSET>
 <ASSETNUM>7112</ASSETNUM>
 -
 <ASSETMETER>
  <METERNAME>RUNHOURS</METERNAME>
  -
 </ASSETMETER>
 <ASSETMETER>
  <METERNAME>KILOMETERS</METERNAME>
  -
 </ASSETMETER>
 -
</ASSET>
```

エンタープライズ・サービスの使用

エンタープライズ・サービスは、操作とオブジェクト構造を関連付けます。エンタープライズ・サービスは、指定されたオブジェクト構造で実行される操作を定義します。これらの操作は、Maximo から、HTTP を介した XML メッセージとして要求できます。エンタープライズ・サービスは、リストされた機能を備えています。

- キュー・サポート
- 処理クラス
- ユーザー出口
- スキップのためのルール
- フローのカスタマイズ

統合のためにエンタープライズ・サービスを使用するには、キューおよび外部システム情報を構成する必要があります。これらのサービスは、エンタープライズ・サービス・アプリケーションによって管理されます。

外部システムは、外部システム・アプリケーションによって管理されます。外部システムは、Maximo とのアウトバウンドまたはインバウンドのデータの同期に関係する特定の外部アプリケーションを識別します。外部アプリケーションで使用可能なすべてのエンタープライズ・サービスを定義します。以下の図に、基本的な概念を示します。

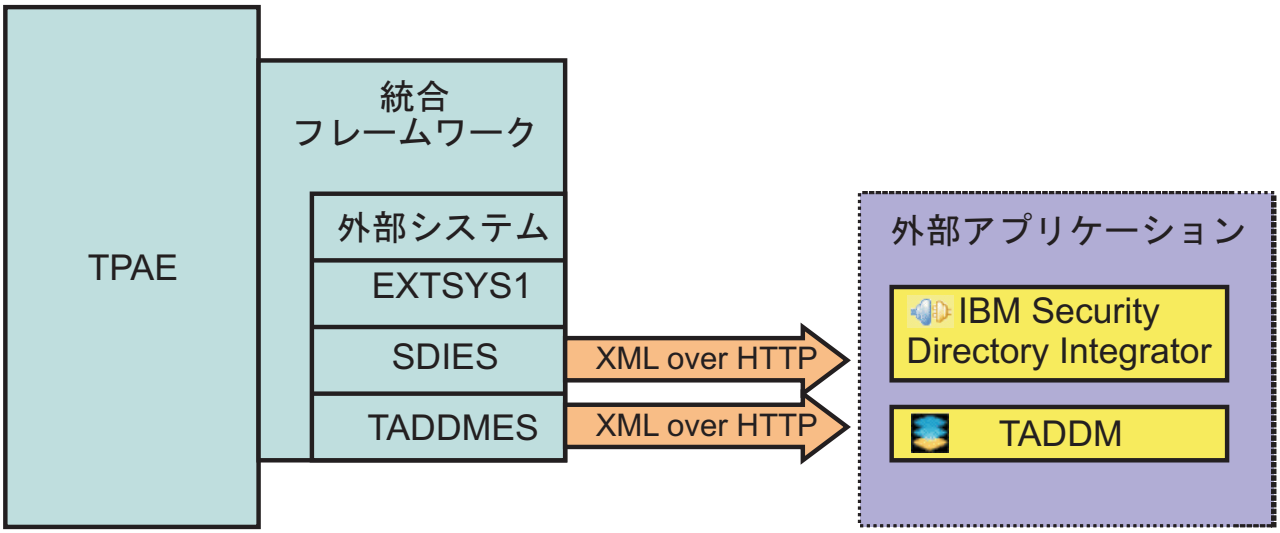


図 3. Tpae エンタープライズ・サービスの例

注: エンタープライズ・サービス・パラメーターが使用可能になり、コネクターで使用できるのは、「**外部システム**」パラメーターを指定した場合のみです。この場合は、オブジェクト構造サービスの代わりに、エンタープライズ・サービスが統合に使用されます。

MBO パラメーター

MBO パラメーターを使用した作業を行うには、以下に示す例を参照できます。

シンプル Tpae IF コネクターは、フラット項目の場合にのみ機能します。オブジェクト構造は、階層的に配置された複数の MBO で構成されます。そのため、コネクターは、*一度に 1 つの MBO のみ* を処理できます。構成エディターの「**MBO**」フィールドに MBO 名を指定する必要があります。このパラメーターが定義されていない場合は、コネクターは、オブジェクト構造のルート MBO を処理します。選択した MBO は、指定したオブジェクト構造の一部でなければなりません。

例えば、事前定義オブジェクト構造の MXASSET は、ASSET、ASSETMETER、ASSETUSERCUST、ASSETSPEC などの MBO で構成されます。

以下の構文で MBO パラメーターを使用します。

```
<Top-Level MBO>[@<Child MBO Level 1>[@<Child MBO Level 2>[@<Child MBO Level N>]]]
```

例:

MBO パラメーターの値	選択した MBO
ASSET	ASSET
ASSET@ASSETMETER	ASSETMETER
ASSET@ASSETSPEC	ASSETSPEC

選択した MBO は、すべての コネクター・モードで使用されます。

指定したオブジェクト構造内で使用可能な MBO のリストを取得するには、`TpaeIFConnector.getMboList()` メソッドを使用します。このメソッドについて詳しくは、Javadoc を参照してください。

コネクター・モード

シンプル Tpae IF コネクターは、各種モード (イテレーター、AddOnly、更新、ルックアップ、削除など) で動作します。以降の各セクションに示す情報を使用して、各モードについて詳しく知ることができます。

イテレーター・モード:

イテレーター・モードでは、コネクターは、照会 XML 要求を IF サーバーに送信して、照会 XML 応答を受信します。詳しくは、以下に示す例を参照してください。

例えば、Maximo は、事前定義の MXASSET オブジェクト構造に対する照会操作の結果として、以下の XML を返します。

```
<ASSET>
 <ASSETNUM>7111</ASSETNUM>
 <BUDGETCOST>1000.0</BUDGETCOST>
 <ASSETSPEC>
  <ASSETATTRID>RAMSIZE</ASSETATTRID>
  <MEASUREUNITID>MBYTE</MEASUREUNITID>
  <NUMVALUE>512.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
  <ALNVALUE />
  <ASSETATTRID>DISKSIZE</ASSETATTRID>
  <MEASUREUNITID>GBYTE</MEASUREUNITID>
  <NUMVALUE>100.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
```

```
  <ASSETATTRID>PROSPEED</ASSETATTRID>
  <MEASUREUNITID>GHZ</MEASUREUNITID>
  <NUMVALUE>1.5</NUMVALUE>
  -
 </ASSETSPEC>
 -
</ASSET>
<ASSET>
 <ASSETNUM>7115</ASSETNUM>
 <BUDGETCOST>1500.0</BUDGETCOST>
 <ASSETSPEC>
  <ASSETATTRID>RAMSIZE</ASSETATTRID>
  <MEASUREUNITID>MBYTE</MEASUREUNITID>
  <NUMVALUE>2048.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
  <ALNVALUE />
  <ASSETATTRID>DISKSIZE</ASSETATTRID>
  <MEASUREUNITID>GBYTE</MEASUREUNITID>
  <NUMVALUE>250.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
  <ASSETATTRID>PROSPEED</ASSETATTRID>
  <MEASUREUNITID>GHZ</MEASUREUNITID>
  <NUMVALUE>3.2</NUMVALUE>
  -
 </ASSETSPEC>
 -
</ASSET>
```

照会は、2 つの資産 (それぞれ 3 つの資産仕様を持つ) を返します。結果の項目オブジェクトは、MBO パラメーターに定義された値によって異なります。

MBO パラメーターが ASSET の場合は、結果は、以下の属性名および値が含まれた 2 つの項目オブジェクトになります。

項目	ASSETNUM	BUDGETCOST
1	7111	1000.0
2	7115	1500.0

MBO パラメーターが ASSET@ASSETSPEC の場合は、結果は、以下の属性および値が含まれた 6 つの項目オブジェクトになります。

項目	ASSET NUM	BUDGET COST	ASSETSPEC@ ASSETATTRID	ASSETSPEC@ MEASUREUNITID	ASSETSPEC@ NUMVALUE
1	7111	1000.0	RAMSIZE	MBYTE	512.0
2	7111	1000.0	DISKSIZE	GBYTE	100.0
3	7111	1000.0	PROSPEED	GHZ	1.5
4	7115	1500.0	RAMSIZE	MBYTE	2048.0
5	7115	1500.0	DISKSIZE	GBYTE	350.0
6	7115	1500.0	PROSPEED	GHZ	3.2

イテレーター・モードでの照会基準

コネクターは、イテレーター・モードでのみ「**照会基準**」パラメーターを使用して、反復の結果セットをフィルターに掛けます。

注: オブジェクト構造の MBO の上位 2 レベルから照会値を選択します。例えば、ASSET、ASSETSPEC、ASSETMETER などの MBO の属性を選択します。

演算子属性

演算子属性は、以下の形式で、フィールドの値を他の 1 つ以上の値と比較します。

operator = oper。ここで、oper は、以下のいずれかの値です。

表 34. 演算子の値

Oper	説明
=	等しい
!=	等しくない
<	より小
<=	より小または等しい
>	より大
>=	より大または等しい
SW	先頭文字
EW	終了文字

「より小」および「より大」の属性は数値および日付フィールドでのみ使用します。

例:

IT 以外のすべてのタイプの資産を検索するには、以下の形式の照会を使用します。

```
<ASSET>
  <ASSETTYPE operator="!=">IT</ASSETTYPE>
</ASSET>
```

フィールド選択

フィールド・ベースの照会では、フィールドの値が、XML フィールドの指定された値と比較されます。値に大文字と小文字の区別はありません。

例:

以下の照会は、VENDOR が ATI で STATUS が OPERATING の資産を検索します。

```
<ASSET>
  <VENDOR operator="=">ATI</VENDOR>
  <STATUS operator="=">OPERATING</STATUS>
</ASSET>
```

以下の照会は、VENDOR に ATI が含まれ、STATUS に OPER が含まれる資産を検索します。

```
<ASSET>
  <VENDOR>ATI</VENDOR>
  <STATUS>OPERATING</STATUS>
</ASSET>
```

以下の照会では、指定されたタグが付いていない資産が検索されます。最初の照会では、演算子属性を使用し、2 番目の照会では、比較に完全一致突き合わせ値を使用しています。

```
<ASSET>
  <ASSETTAG operator="NULL"></ASSETTAG>
</ASSET>
```

```
<ASSET>
  <ASSETTAG>NULL</ASSETTAG>
</ASSET>
```

以下の照会は、テキスト 711 で開始する資産番号の資産を検索します。

```
<ASSET>
  <ASSETNUM operator="SW">711</ASSETNUM>
</ASSET>
```

以下の照会は、SQL IN 節に相当する集合を使用して、状況が NOT READY または OPERATING の資産を検索します。

```
<ASSET>
  <STATUS>NOT READY, OPERATING</STATUS>
</ASSET>
```

範囲選択

照会では、特定の範囲内の値を持つレコードを検索することができます。フォーマットは、選択基準が無制限であるか、範囲の上限および下限があるかによって異なります。

例:

以下の照会では、BUDGETCOST が 1000 ドルを超える資産が検索されます。

```
<ASSET>
  <BUDGETCOST operator="&gt;">1000</BUDGETCOST>
</ASSET>
```

以下の照会では、BUDGETCOST が 1000 ドルより大きく 20000 ドルより小さい資産が検索されます。

```
<ASSET>
  <BUDGETCOST operator="&gt;">1000</BUDGETCOST>
  <BUDGETCOST operator="&lt;">20000</BUDGETCOST>
</ASSET>
```

注: 照会には、同じ属性に対して最大で 2 つの参照を含めることができます。

AddOnly モード:

ここに記載されている情報を使用することで、AddOnly モードについて知ることができます。

シンプル Tpae IF コネクターを使用して項目を追加する場合、「必須」のマークが付いている属性を指定します。Tpae では、空ストリングも許可されます。指定されていない必須属性がある場合、コネクターが例外をスローして、追加操作が失敗します。

注: 子の MBO を追加する場合は、親が IF 内に存在していることを確認してください。

AddOnly モードでの MBO パラメーター

MBO パラメーターのターゲットがオブジェクト構造のルート MBO の場合、コネクターは「**CREATE エンタープライズ・サービス**」パラメーターを使用します。

MBO パラメーターのターゲットが任意のレベルのオブジェクト構造の子 MBO である場合、コネクターは「**UPDATE エンタープライズ・サービス**」パラメーターを

使用します。作成対象の MBO パラメーターのターゲットとなる MBO を除き、MBO のキー属性 (オブジェクト構造のルート MBO まで) を、既存のレコードへの参照と共に指定してください。

例えば、事前定義されたオブジェクト構造 MXASSET は、ASSET MBO と ASSETMETER MBO を公開します。したがって、資産の計測値を作成するには、少なくとも以下の属性を使用して、作業項目を指定する必要があります。

属性名	値
ASSETNUM	1001
SITEID	BEDFORD
ASSETMETER@METERNAME	RUNHOURS

ASSETNUM と SITEID は、既存の資産を識別します。ASSETMETER@METERNAME は、新規計測値の名前です。

更新モード:

ここに記載されている情報を使用することで、更新モードについて知ることができます。

シンプル Tpae IF コネクターで項目を変更する場合は、「必須」のマークが付いた属性だけを指定してください。指定されていない必須属性がある場合、コネクターが例外をスローして、変更操作が失敗します。

注: 出力マップ内で MBO の固有キーが変更された場合、そのキーの値は Maximo から読み取られた元の値で上書きされます。このキー値により、MBO が変更されます。さらに、固有属性は変更できないことを知らせるデバッグ・メッセージが表示されます。

削除モード:

ここに記載されている情報を使用することで、削除モードについて知ることができます。

シンプル Tpae IF コネクターを使用して項目を削除する場合は、「必須」のマークが付いた属性だけを指定してください。指定されていない必須属性がある場合、コネクターが例外をスローして、削除操作が失敗します。

注: すべての固有属性を指定した場合でも、削除が失敗することがあります。この失敗は、Maximo オブジェクト間の関係によるものです。

削除モードでの MBO パラメーター

MBO パラメーターのターゲットがオブジェクト構造のルート MBO または子 MBO の場合、コネクターは「**SYNC エンタープライズ・サービス**」パラメーターを使用します。すべての MBO のキー属性 (オブジェクト構造のルート MBO まで) を、既存のレコードへの参照と共に指定してください。

例えば、事前定義されたオブジェクト構造 MXASSET は、ASSET MBO と ASSETMETER MBO を公開します。したがって、資産計測値を削除するには、以下の属性を使用して作業項目を指定します。

属性名	値
ASSETNUM	11430
SITEID	BEDFORD
ASSETMETER@METERNAME	RUNHOURS

ASSETNUM と SITEID は、既存の資産を識別します。ASSETMETER@METERNAME は、削除する計測値を識別します。

ルックアップ・モード:

Maximo 内の特定のレコードを検索するには、そのレコードを一意的に識別する属性を使用してリンク基準を指定します。

注: 選択された MBO の固有属性には、構成エディター内で「必須」のマークが付けられます。
例:

以下の属性は、Maximo 内の資産を一意的に識別します。

属性名	値
ASSETNUM	1001
SITEID	BEDFORD

以下の属性は、Maximo 内の資産計測値を一意的に識別します。

属性名	値
ASSETNUM	1001
SITEID	BEDFORD
ASSETMETER@METERNAME	RUNHOURS

シンプル Tpae IF コネクターは、タイプが AND のリンク基準と、以下の一致演算子のみをサポートしています。

- 「=」– バイナリー等号演算子
- "!" –バイナリー不等号演算子。両方のオペランドが異なっていなければなりません。

スキーマ

返される項目のスキーマは、選択されたオブジェクト構造と MBO に応じて異なります。以下に示す情報を参照すると、これについて詳しく知ることができます。

各 MBO には、その MBO を作成、更新、または削除する場合に指定しなければならない固有属性があります。これらの属性には、構成エディター内で「必須」のマークが付けられます。

エラー処理

シンプル Tpae IF コネクターは、通常のサーバー・フックを使用して、発生するすべての例外を処理します。障害を処理できない場合は、対応する AssemblyLine エラー・フックが開始されます。以下に示す例外は、このコネクター特有の例外です。

- MxConnectorRuntimeException
  - MxConnConfigException
- MxConnectorException
  - MxConnIOException
    - MxConnHttpException
    - MxConnTimeoutException
  - MxConnSchemaException
    - MxConnExcedentSizeException
    - MxConnTypeConvertionException
  - MxConnXmlParsingException

シンプル Tpae IF コネクターでの Assembly Line で障害が発生した場合、以下のようにしてエラーに関する追加情報を取得できます。

1. 「エラー発生時のデフォルト」フックに以下のコードを追加します。コネクターには mxConn という名前を付けます。

```
task.logmsg("ERROR", "An exception occurred.");
mxConn.connector. extractMaximoException(error);
task.dumpEntry(error);
```

2. 例外が発生すると、次のメッセージが表示されます。

```
19:31:44  CTGDIS003I *** 項目のダンプを開始します
19:31:44   Operation: generic
19:31:44   Entry attributes:
19:31:44   exception (replace): 'com.ibm.di.connector.maximo.exception.
           MxConnHttpException: response: 404 - Not Found'
19:31:44   targetUrl (replace): 'http://9.156.6.14/meaweb/schema
           /service/MXPersonService.xsd'
19:31:44  class (replace): 'com.ibm.di.connector.maximo.
          exception.MxConnHttpException'
19:31:44  operation (replace): 'update'
19:31:44  status (replace): 'fail'
19:31:44  connectorname (replace): 'AddPerson'
19:31:44  body (replace): 'Error 404: BMXAA1513E -
          Cannot obtain resource /meaweb/schema/service/MXPersonService.xsd.'
19:31:44  responseCode (replace): '404.0'
19:31:44  responseMessage (replace): 'Not Found'
19:31:44  message (replace): 'The HTTP server did not returned "HTTP OK".'
19:31:44  CTGDIS004I *** 項目のダンプを完了しました
```

注: task.dumpEntry(error) は、エラーに関する情報を出力します。

外部システムの構成

以下に示す手順を使用して、XML スキーマ定義の生成方法を知ることができます。

XML スキーマ定義の生成

コネクターを初めて使用する場合は、以下の手順を実行します。

1. システム構成タスクの実行権限を持つ管理者として Maximo にログオンします。

2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**オブジェクト構造**」を選択して、オブジェクト構造アプリケーションを開きます。
3. 使用するオブジェクト構造ごとに、以下の手順を繰り返します。
   a. 「**リスト**」タブで、オブジェクト構造の名前 (MXASSET など) を検索します。

      検索するには、「フィルター」を開き、「**オブジェクト構造**」列の「**フィルター**」フィールドにオブジェクト構造の名前または名前の一部を入力します。次に ENTER を押します。
   b. オブジェクト構造名をクリックして、オブジェクト構造のレコードを開きます。
   c. 「**アクションの選択**」メニューから、「**スキーマの生成/XML の表示**」を選択します。

      操作ごとにスキーマを生成するかどうかを尋ねるメッセージ・ボックスが開きます。
   d. 「**OK**」をクリックします。「XML の表示」ダイアログ・ボックスが開きます。
   e. 「**一覧**」タブに戻るには、「**OK**」をクリックします。

構成

ここに記載されているパラメーターを使用することで、シンプル Tpae IF コネクターを構成できるようになります。

基本 URL

このパラメーターを使用して、Tpae 製品にメッセージを送信するための URL のリストを指定します。Tpae が IBM Security Directory Integrator と同じシステム上にある場合は、`http://localhost` を使用します。それ以外の場合は、Tpae サーバーの IP アドレスを使用します。Tpae アプリケーションにログインする場合と同じポートを使用してください。例えば、ログイン URL が `http://192.168.80.128:9080/maximo/webclient/login/login.jsp` の場合、ポート 9080 を使用します。リストでは、URL 間の分離文字としてスペースを使用します。

注: 高可用性を実現するには、単一の URL でなく URL のリストを使用する必要があります。リストの先頭のサーバーが例外をスローすると、2 番目の URL が使用され、有効なサーバーが見つかるまで順に同様の処理が行われます。最後の URL も無効だった場合は、例外がスローされて接続が失敗します。

必要な認証

このパラメーターを使用して、HTTP 要求ヘッダーにユーザー ID とパスワードを組み込みます。このパラメーターは、「`HTTP 基本認証 (BA)`」オプションが有効になっているサーバーでのみサポートされます。

ユーザー ID

このパラメーターを使用して、Tpae アプリケーションにログインするための有効なユーザー ID を指定します。

注: 構成エディターの「**ユーザー ID**」フィールドは、「**認証が必要**」チェック・ボックスを選択した場合にのみ使用可能になります。

パスワード
このパラメーターを使用して、Tpae アプリケーションにログインするための有効なパスワードを指定します。

注: 構成エディターの「**パスワード**」フィールドは、「**認証が必要**」チェック・ボックスを選択した場合にのみ使用可能になります。

オブジェクト構造
このパラメーターを使用して、統合に使用されるオブジェクト構造の名前を指定します。各オペレーティング・システムには事前定義された MBO のセットが含まれているため、このパラメーターにより、**MBO** パラメーターに指定できる値の範囲が制限されます。

MBO このパラメーターを使用して、統合に使用される MBO オブジェクトの名前を指定します。このパラメーターを指定しなかった場合、コネクターは、指定されたオブジェクト構造のルート・オブジェクトを使用します。このパラメーターについて詳しくは、325 ページの『MBO パラメーター』セクションを参照してください。以下の 2 つのフィールドが **MBO** パラメーターに関連付けられます。

MBO の取得
有効な MBO パラメーターを選択するには、以下の手順を実行します。

1. 「**MBO の取得**」ボタンをクリックして、指定されたオブジェクト構造について指定可能な MBO 名のリストを取得します。
2. MBO パラメーターとして、リストから名前を選択します。

クリア コネクターは、MBO リストの表示にかかる時間を最小限にするために、使用されたすべてのオブジェクト構造のスキーマを内部のキャッシュに保存します。「**クリア**」ボタンをクリックすると、保存されたすべてのスキーマを削除することができます。オブジェクト構造のスキーマが変更された場合 (XSD の生成) や、このスキーマのローカル表現の更新が必要な場合は、この操作を使用すると便利です。スキーマのキャッシュをクリアすると、その後、別のオペレーティング・システムに対する「MBO の取得」スクリプトの呼び出しで遅延が生じます。遅延の原因は、各スキーマを再度サーバーから取得する必要があることです。

注: 「**クリア**」ボタンをクリックすると、設計時に使用されたスキーマ・キャッシュがクリアされます。構成エディターがサーバー上で AssemblyLine を実行すると、別のスキーマ・キャッシュが作成されます。コネクターで `clearSchemaCache()` メソッドを呼び出すと、このスキーマを削除することができます。例えば、「**初期化後**」フックに `thisConnector.connector.clearSchemaCache();` というテキストを追加すると、コネクターを使用する前にキャッシュをクリアすることができます。

照会基準
このパラメーターを使用して、反復の結果セットをフィルタリングします。

このパラメーターには、イテレーター・モード用の選択基準が含まれています。照会は XML 構文で指定します。単一値または値の範囲に基づいてレコードを選択できます。

注: 照会基準は、オブジェクト構造のルート MBO にのみ適用されます。
照会基準パラメーターのフォーマットを以下に示します。

```
<MBO>
  <FIELD1 operator="oper"> </FIELD1>
  <FIELD2> </FIELD2>
  ...
</MBO>
```

ここで、

MBO - 検索対象のビジネス・オブジェクト。
FIELD - MBO フィールドの名前。
oper - 検索のための条件演算子。

詳細と照会基準の例については、326 ページの『イテレーター・モード』を参照してください。

照会属性
サーバーに送信された XML の照会エレメントに追加される属性。

ページ・サイズ
このパラメーターを使用して、Tpae から取得するレコードの数を制限します。コネクターは、照会基準によって選択されたすべてのレコードを取得するための複数の要求を作成します。

注: ページ・サイズは、オブジェクト構造のルート MBO にのみ適用されます。
例えば、Maximo のデータベースに 1000 個の資産があり、ページ・サイズが 100 と定義されている場合、事前定義された MXASSET オブジェクト構造に対する照会は 10 個の要求によって実行されます。 デフォルト値は 100 です。

フィールド・サイズの検証
このパラメーターを使用して、テキスト・フィールドが最大サイズを超えた場合にエラーをスローします。構成エディターで「**フィールド・サイズの検証**」チェック・ボックスが選択されていない場合は、テキストが切り捨てられます。

XML 文字検証
このパラメーターを使用して、XML コンテンツの構文解析を実行する前に、無効な Unicode 文字を XML コンテンツから削除します。

IF バージョン
このパラメーターを使用して、サーバーと交換される各メッセージに含まれていなければならない IF のバージョンを指定します。構成エディターで、適切なデフォルト値が提供されます。

トランザクション言語
このパラメーターを使用して、複数言語に対応したフィールドの内容値を提供するためのトランザクション言語を指定します。デフォルト値は EN で

す。言語の頭字語の詳細なリストについては、ISO 639-1 alpha-2 コード (http://www.loc.gov/standards/iso639-2/php/English_list.php) を参照してください。

選択可能な項目は、以下のとおりです。

- DE - ドイツ語
- EN - 英語
- ES - スペイン語
- FR - フランス語
- IT - イタリア語
- KO - 韓国語
- PT - ポルトガル語
- ZH - 中国語

タイムアウト

このパラメーターを使用して、IF サーバーと通信します。接続を確立する前、または使用可能なデータを読み取る前にタイムアウト時間が経過すると、`MxConnTimeoutException` がスローされます。タイムアウトの値をゼロ (デフォルト) にすると、無限タイムアウトと見なされます。

外部システム

このパラメーターを使用して、外部システムの名前を指定します。外部システムは、作成、更新、削除、または照会操作のエンタープライズ・サービスを、選択されたモード用にグループ化して公開します。

MAXOBJECT/MAXATTRIBUTE オブジェクト構造

このパラメーターを使用して、`MAXOBJECT` MBO と `MAXATTRIBUTE` MBO を公開するオブジェクト構造の名前を指定します。このオブジェクト構造を使用して、属性の最大許容サイズなど、補足的な MBO のメタデータを取得します。デフォルト値は `MXOBJECTCFG` です。

MAXOBJECT/MAXATTRIBUTE QUERY エンタープライズ・サービス

このパラメーターを使用して、`MAXOBJECT` オブジェクト構造に対する*照会*操作を実行するエンタープライズ・サービスの名前を指定します。このオブジェクト構造を使用して、属性の最大許容サイズなど、補足的な MBO のメタデータを取得します。

注: このパラメーターは、「**外部システム**」パラメーターを指定した場合のみ有効です。
デフォルト値は `MXMaxObjectQuery` です。

CREATE エンタープライズ・サービス

このパラメーターを使用して、指定されたオブジェクト構造に対する*作成*操作を実行するエンタープライズ・サービスの名前を指定します。

注: このパラメーターは、「**外部システム**」パラメーターを指定し、コネクターが AddOnly 動作モードまたは更新動作モードになっている場合のみ有効です。

SYNC エンタープライズ・サービス
このパラメーターを使用して、指定されたオブジェクト構造に対する*同期*操作を実行するエンタープライズ・サービスの名前を指定します。

注: このパラメーターは、「**外部システム**」パラメーターを指定し、コネクターが削除動作モードになっている場合のみ有効です。

QUERY エンタープライズ・サービス
このパラメーターを使用して、指定されたオブジェクト構造に対する*照会*操作を実行するエンタープライズ・サービスの名前を指定します。

注: このパラメーターは、「**外部システム**」パラメーターを指定した場合のみ有効です。

UPDATE エンタープライズ・サービス
このパラメーターを使用して、指定されたオブジェクト構造に対する*更新*操作を実行するエンタープライズ・サービスの名前を指定します。

注: このパラメーターは、「**外部システム**」パラメーターを指定し、コネクターが更新動作モードになっている場合のみ有効です。

コメント
このパラメーターを使用して、コメントを追加します。データの構文解析時には、このコメントは無視されます。

詳細ログ
このパラメーターを使用して、詳細なログ・メッセージを生成します。

例

シンプル Tpae IF コネクターを使用した作業を行うには、以下に示す例を使用できます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/SimpleTpaeIFConnector ディレクトリーにあります。

関連情報

641 ページの『第 6 章 資産統合スイート』,
393 ページの『Tpae IF コネクター』,
384 ページの『Tpae IF 変更検出コネクター』.

SNMP コネクター

このコネクターは、ネットワーク上で送信される SNMP トラップを listen し、SNMP PDU 内のすべてのエレメントの名前と値を示す項目を戻します。このコネクターで作業するときは、以下に示す点に注意が必要です。

注:

1. クライアント・モードでは、最初の待機時間が 5 秒間、その後 1 回ごとに倍増して間隔を長くしながら、5 回再試行されます。その間に応答が受信されないと、タイムアウトが生じます。

2. SNMP トラップを送信することが目的である場合は、`system.snmpTrap()` メソッドを使用できます。
3. SNMP コネクターは、拡張リンク基準をサポートしていません (「」の『拡張リンク基準』を参照)。

構成

ここに記載されているパラメーターを使用することで、SNMP コネクターを構成できるようになります。

このコネクターは、以下のパラメーターを必要とします。

コミュニティー・ストリング
コネクターをテストする場合は、**public** を使用します。

モード トラップ・リスナーまたはクライアント。クライアント・モードの場合は、このコネクターを、AddOnly モード (SNMP Set)、ルックアップ・モード (SNMP Get)、またはイテレーター・モード (Walk) で使用できます。

トラップ・リスナーの場合は繰り返しのみが可能で、ローカル・ホスト上でトラップが listen されます。

SNMP トラップ・ポート
トラップ・モードのポート。クライアント・モードでは使用されません。

トラップ待機タイムアウト
トラップ・モードでのタイムアウト。次のプロトコル・データ単位 (PDU) を待つミリ秒数。この値が 0 以下の場合は、コネクターは無期限に待機します。

SNMP ホスト (get 用)
クライアント・モードでの Get の場合のみ使用されます。トラップ・モードでは使用されません。

SNMP ポート
クライアント・ポート (クライアント・モードの場合)。トラップ・モードでは使用されません。

SNMP Walk OID (繰り返し)
クライアント・モードのイテレーター・コネクターでのみ使用されます。ウォークする OID ツリーを指定します。

SNMP バージョン
クライアント・モードでの get/walk 用のデフォルト・バージョン。トラップ・モードでは使用されません。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

注: このコネクターではリンク基準の取り扱いが他と異なります。ルックアップ・モードでは、このコネクターは get 要求を実行し、要求された oid について **oid/値** を戻します。リンク基準は、**oid**、**サーバー**、**ポート**、および**バージョン**を指定します。例えば、リンク基準は、**"oid" = "1.1.1.1.1.1.1"** のようになります。

例

SNMP コネクターを使用した作業を行うには、以下に示す例を使用できます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/snmpTrap ディレクトリーにあります。

関連情報

SNMP V1: RFC1155、RFC1157
SNMP V2: RFC1901、RFC1907、RFC2578
SNMP V3: RFC3411、RFC3412。

SNMP サーバー・コネクター

SNMP サーバー・コネクターは SNMP v1 をサポートしています。SNMP v2 では、SNMP v2 認証および暗号化機能を除いた部分をサポートしています。以下に示す情報を使用して、このコネクターについて詳しく知ることができます。

このコネクターは SNMP TRAP メッセージをサポートしていません。

SNMP サーバー・コネクターはサーバー・モードのみで稼働します。使用するトランスポート・プロトコルは TCP ではなく UDP です。UDP は信頼性が低いトランスポート・プロトコルであり、信頼性が低いトランスポート・プロトコルでは SSL を実行できません。このため、コネクターは SSL を使用してトランスポート層を保護することができません。

SNMP サーバー・コネクターは (サーバー・モードの他のコネクターとは対照的に) DatagramSocket を使用します。このため、接続という概念はありません。SNMP サーバー・コネクターは、ネットワーク上の複数の SNMP マネージャーから SNMP パケットを受信する単一 DatagramSocket を使用します。

getNextClient() メソッドでは、SNMP パケットを受信するまでソケットが receive() メソッドでブロックします。パケットを受信すると、コネクターはコネクター自体の新規インスタンスを作成し、受信パケットを子コネクターに渡し、子コネクターを戻します。

getNextEntry() メソッドは SNMP 要求パケット属性を抽出して、これらの属性を *conn* 項目に設定します。これで入力属性マッピングを実行できます。

replyEntry() メソッドが *conn* 項目から属性を抽出して SNMP 応答パケットを作成し、クライアントにこのパケットを戻します。出力属性マッピングを使用して *conn* 項目にデータを取り込む必要があります。

replyEntry() メソッドは親コネクターの DatagramSocket を使用して応答を送り戻します。親コネクターの DatagramSocket はすべての子コネクターにより共有されるため、DatagramSocket へのアクセスは同期化されます。

コネクター・スキーマ

SNMP サーバー・コネクターでは、入力属性マップで提供された属性が使用可能になります。

snmp.operation
呼び出された SNMP 操作を表す java.lang.String オブジェクト。サポートされる操作のタイプは GET、GETNEXT、および SET です。

snmp.community
ネットワーク管理システム (NMS) グループのアクセス環境を定義します。コミュニティー内の NMS は、同一管理可能ドメイン内に存在しているものとされています。適切なコミュニティー名を認識していないデバイスは SNMP 操作からあらかじめ除外されるため、コミュニティー名は弱い認証として機能します。

snmp.remoteip
SNMP クライアントの IP アドレス (ドット表記)。

snmp.errorcode
特定のエラーおよびエラー・タイプを示します。このフィールドは応答操作によってのみ設定されます。その他の操作では、このフィールドはゼロに設定されます。

snmp.errorindex
エラーを特定のオブジェクト・インスタンスに関連付けます。このフィールドは応答操作によってのみ設定されます。その他の操作では、このフィールドはゼロに設定されます。

snmp.request-id
SNMP 要求を応答に関連付けます。

snmp.PDU
プロトコル・データ単位。SNMP PDU には特定のコマンド (Get、Set など) と、トランザクションに使用されるオブジェクト・インスタンスを示すオペランドが含まれています。

snmp.oid
OID とは、ターゲット・システム内で読み取りまたは変更される特定の変数または属性を示す、MIB 構造のアドレスです。 GET には OID のリストを含めることができ、SET にはターゲット・システム内のそれらの変数に対して設定される対応する値も含めることができます。ただし、多くの場合、SNMP 配置で使用するのは SNMP メッセージごとに 1 つの OID のみです。

snmp.oidvalue
1 つの OID の対応値が含まれています。これはストリング表現です。

snmp.oidvalue.raw
1 つの OID の対応値が含まれています。これはオブジェクト表現です。

構成

ここに記載されているパラメーターを使用することで、SNMP サーバー・コネクターを構成できるようになります。

UDP ポート
コネクターが (1) 着信 SNMP 要求パケットを受信し、(2) SNMP 応答パケットを送信する、UDP ポートを指定します。デフォルト値は 161 です。これは、SNMP GET/SET 操作の標準ポートです。

コミュニティーの検査

SNMP コミュニティー名を指定します。適切なコミュニティー名を認識していないデバイスは SNMP 操作からあらかじめ除外されるため、SNMP コミュニティー名は、弱い認証として機能します。

設定すると、コネクターはこのコミュニティー・ストリングに一致しないメッセージをすべて廃棄します。 ブランクの場合、コネクターはすべてのコミュニティー・ストリングを許可します。

デフォルト値は「public」です。

詳細ログ

使用可能に設定されていると、詳細なログ・メッセージが生成されます。

Sun Directory 変更検出コネクター

以下に示す情報とリンクを使用して、Sun Directory 変更検出コネクターについて知ることができます。

Sun Directory 変更検出コネクターは、LDAP コネクターの特殊インスタンスです。このコネクターは、以前は Netscape/iPlanet 変更ログ・コネクターと呼ばれていました。

Sun/iPlanet Directory Server 5.0 では、変更ログの形式が専有の形式に変更されました。旧バージョンの iPlanet Directory Server では、LDAP を使用して変更ログにアクセスできました。現在は、変更ログはサーバーによる内部使用専用になっています。変更ログの読み取りが必要なアプリケーションがある場合は、以前のバージョンとの互換性を維持するために iPlanet Retro Change Log Plug-in を使用する必要があります。

Sun/iPlanet Directory Server を Retro Changelog モードで常に実行できるわけではないため、コネクターは次の 2 種類の配信モードで実行できます。

1. *変更ログ・モード* – このモードでは、コネクターは変更ログを反復し (iPlanet Retro Change Log Plug-in により可能)、すべての項目の配信が完了すると、新規変更があるかどうかを確認するためポーリングするか、または変更通知を使用します。
2. *リアルタイム・モード* – このモードでは、通知として受け取った変更のみが配信され、オフラインの変更は失われます。このモードではコネクターは変更ログを使用しません。この配信モードは、変更ログをサポートしていない Sun/Netscape/iPlanet Server で必要となります。

このコネクターは、次の 2 種類の動作モードでデルタ・タグをサポートします。

- **変更ログ**・モードでは、項目レベル、属性レベル、および属性値レベルでデルタ・タグがサポートされます。LDIF パーサーが属性レベルと属性値レベルでデルタ・サポートを提供します。
- **リアルタイム**・モードでは、項目レベルでのみデルタ・タグ付けが実行されます。

コネクターにより、サーバーの変更ログの *modrdn* 操作が検出されます。詳しくは、245 ページの『modrdn 操作の検出と処理』を参照してください。

属性のマージ動作

古いバージョンの IBM Security Directory Integrator では、Sun Directory 変更検出コネクターのマージは、変更ログ項目の各属性と実際のディレクトリー項目の変更された属性の間で実行されます。

変更された属性を検出できないため、これにより問題が発生します。現行バージョンのコネクターには、このような状態を解決するためのロジックがあります。このロジックは、パラメーター「**マージ・モード**」によって設定されます。モードは以下のとおりです。

- **変更ログおよび変更データのマージ**: コネクターは、変更ログ項目の属性と実際のディレクトリー項目の変更された属性をマージします。これは旧来の実装であり、以前のバージョンとの互換性を確保します。
- **変更データのみを戻す**: 変更/追加された属性のみを戻し、変更ログ・イテレーターとデルタ・モードを容易にします。これはデフォルト設定です。旧バージョンの IBM Security Directory Integrator で開発された、または旧バージョンから移行した構成の場合、同じ動作を確保するために手動で「**変更ログおよび変更データのマージ**」を選択する必要が生じることがあります。
- **両方とも戻す**: 実際のディレクトリー項目の変更された属性を含む項目および変更ログ項目の属性を含む追加属性「changelog」を戻します。これにより、2 つのセットの属性を簡単に区別できます。

デルタ・タグはすべてのマージ・モードでサポートされているので、大量のスクリプトを使用しなくても、異なる LDAP サーバー間で項目を転送できます。

「**リアルタイム**」モードでは、LDAP 検索ベースが「cn=changelog」と異なる場合、コネクターがディレクトリー項目の変更された属性を特定できず、マージ・モードのパラメーターの値に関係なく、出力項目が同一のままになってしまうことに注意してください。リアルタイム・モードで、サーバーで変更ログがサポートされ、検索ベースが「cn=changelog」に設定されている場合は、選択したマージ・モードに応じて出力項目がマージされます。

構成

ここに記載されているパラメーターを使用することで、Sun Directory 変更検出コネクターを構成できるようになります。

LDAP URL
接続の LDAP URL (ldap://*host*:*port*)。

ログイン・ユーザー名
サーバーへの認証に使用する LDAP 識別名。匿名アクセスの場合はブランクにしておいてください。

ログイン・パスワード
信任状 (パスワード)。

イテレーター状態キー
IBM Security Directory Integrator のユーザー・プロパティー・ストアに現在の同期状態を格納するパラメーターの名前を指定します。これは、IBM

Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。

「**削除**」ボタンを押すと、この状態情報がユーザー・プロパティー・ストアから削除されます。

変更番号で開始

開始変更番号を指定します。各変更ログ項目には **changenumber=intvalue** という名前が付けられ、コネクターはこのパラメーターに指定した番号から開始され、自動的に 1 つずつ番号が増加します。特殊値 **EOD** は、変更ログの終わりから開始することを意味します。

このパラメーターは、イテレーター状態がブランクまたは保存されていない場合にのみ使用されることに注意してください。

「**照会**」ボタンを押すと、最初および最終の変更番号がサーバーから取得されます。

認証メソッド

LDAP 認証のタイプ。これは、次のいずれかです。

- **無名** - この認証メソッドが設定されている場合、クライアントが接続されているサーバーでは、そのクライアントがどのクライアントであるかを認識しません (クライアント情報を必要としません)。サーバーは、このようなクライアントが、非認証ユーザー用に構成されたデータにアクセスすることを許可します。ユーザー名が指定されない場合、コネクターは、自動的にこの認証メソッドを指定します。ただし、このタイプの認証が選択され、**ログイン・ユーザー名**と**ログイン・パスワード**が指定された場合、コネクターは、自動的に認証メソッドを単純に設定します。
- **単純**: **ログイン・ユーザー名**と**ログイン・パスワード**を使用します。**ログイン・ユーザー名**と**ログイン・パスワード**を指定しない場合は、無名と見なされます。SSL プロトコルを使用して LDAP サーバーと通信するようにコネクターが構成されている場合を除いて、コネクターは、完全修飾された識別名 (DN) およびクライアント・パスワードを平文で送信することに注意してください。
- **CRAM-MD5** - これは SASL 認証メカニズムの 1 つです。LDAP サーバーは接続に際して、特定のデータを LDAP クライアント (すなわち、このコネクター) に送信します。すると、クライアントは MD5 暗号化方式を使用して暗号化された応答をパスワードとともに送信します。その後、LDAP サーバーはクライアントのパスワードをチェックします。CRAM-MD5 がサポートされるのは LDAP v3 サーバーのみです。 のどのサポート対象バージョンでもサポートされません。
- **SASL** - クライアント (このコネクター) は、LDAP サーバーへの接続時に Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。このタイプの認証の稼働パラメーターは、「**追加のプロバイダー・パラメーター**」オプションを使用して指定する必要があります。例えば、DIGEST-MD5 認証をセットアップするには、以下のパラメーターを「追加のプロバイダー・パラメーター」フィールドに追加する必要があります。

```
java.naming.security.authentication:DIGEST-MD5
```

SASL 認証およびパラメーターについて詳しくは、http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html を参照してください。

注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

SSL の使用

「SSL の使用」が **true** である場合は、コネクターは SSL を使用して LDAP サーバーに接続します。これに従ってポート番号の変更が必要になることがあります。

変更ログ/通知ベース

変更ログが保持される検索ベースを指定します。標準 DN は **cn=changelog** です。「リアルタイム」配信モードの通知コンテキストとも呼ばれます。

追加のプロバイダー・パラメーター

複数の追加パラメーターを JNDI レイヤーに受け渡せるようにします。名前:値のペアとして 1 行に 1 つずつ指定します。

状態キーの維持

コネクターの状態をシステム・ストアに保管する方式を制御します。デフォルトおよび推奨の設定は「**サイクルの終わり**」であり、以下のいずれかを選択できます。

読み取り後

Sun Directory Server 変更ログから項目を読み取った時点で、AssemblyLine の残りの部分を続行する前に、システム・ストアを更新します。

サイクルの終わり

AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、変更ログ番号を使用してシステム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。この場合、AssemblyLine 内の特定の時点において、iPlanet Directory Server 変更ログ・コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保管する必要があります。

マージ・モード

変更ログ項目の属性と実際のディレクトリー項目の変更された属性のマージに使用されるメソッドを管理します。デフォルトは「**変更データのみを戻す**」であり、以下のいずれかを選択できます。

変更ログおよび変更データのマージ

コネクターにより、変更ログ項目の各属性が実際のディレクトリー項目の変更された属性とマージされます。このオプションでは、古いバージョンの IBM Security Directory Integrator の動作を選択し、以前のバージョンとの互換性を維持します。

変更データのみを戻す

変更または追加された属性のみが戻されます。

両方とも戻す

接頭部が「changelog.」である変更ログ属性の項目、およびディレクトリー項目の変更された属性が戻されます。

配信モード

変更ログまたは (リアルタイム) 通知項目のいずれを使用するかを指定します。LDAP サーバーで変更ログが維持されていない場合は、「**リアルタイム**」のみが使用可能なオプションです。デフォルトは「**変更ログ**」です。

通知の使用

Sun Directory Server での新規変更の待機時に通知を使用するかどうかを指定します。使用可能に設定すると、コネクターはスリープもタイムアウトもせずに (対応するパラメーターは無視され)、代わりに Sun Directory Server からの通知イベントを待ちます。

バッチ検索

変更ログでの検索の実行方法を指定します。チェックされていない場合は、コネクターは増分ルックアップを実行します (後方互換モード)。これがチェックされており、サーバーで「ソート制御」がサポートされている場合は、「changenumber>=some_value」という照会を使用して検索が実行されます。これは、最後に実行した検索に対応しています。. デフォルトでは、このオプションはチェックされていません。

タイムアウト

コネクターが次の変更ログ項目を待つ秒数を指定します。デフォルトは 0 です。このデフォルト値では、無期限で待機します。

スリープ間隔

コネクターが各ポーリング間でスリープする秒数を指定します。デフォルトは 60 です。

詳細ログ

このフィールドをチェックすると、追加のログ・メッセージが生成されます。

注: タイムアウトまたはスリープ間隔の値を変更すると、そのピアは自動的に有効な値に調整されます (例えば、タイムアウトをスリープ間隔よりも大きくした場合、未編集の値は変更された値にあわせて調整されます)。調整はフィールド・エディターがフォーカスを失うときに実行されます。

関連情報

Sun Directory Server の標準変更ログ、
Sun Directory Server の Retro Changelog、
243 ページの『LDAP コネクター』,
9 ページの『Active Directory 変更検出コネクター』,,
173 ページの『IBM Security Directory Integrator 変更ログ・コネクター』
418 ページの『z/OS LDAP 変更ログ・コネクター』.

システム・キュー・コネクター

システム・キューは、Java Message Service (JMS) と類似したサブシステムを IBM Security Directory Integrator に提供します。汎用メッセージと IBM Security Directory Integrator 項目オブジェクトを保管し、IBM Security Directory Integrator サーバーと AssemblyLine の間で転送します。以下に示す情報を使用して、システム・キュー・コネクターについて詳しく知ることができます。

システム・キュー・コネクターは、AssemblyLine がシステム・キューとのインターフェースをとるためのメカニズムです。システム・キューおよびその構成について詳しくは、「*インストールと管理*」のシステム・キューのセクションを参照してください。

システム・キュー・コネクターは、イテレーター・モードと AddOnly モードで AssemblyLine とともに使用できます。

- イテレーター・モードでは、コネクターは指定されているメッセージ・キューから IBM Security Directory Integrator 項目オブジェクトを取得します。
- AddOnly モードでは、コネクターは指定されているメッセージ・キューに IBM Security Directory Integrator 項目オブジェクトを格納します。

注: 2 つの JMS クライアントが同一の JMS キューから同時にメッセージを取得すると、エラーが発生することがあります。同一の JMS キューから同時にメッセージを取得する複数のシステム・キュー・コネクター・インスタンスを使用するソリューションは使用しないでください。ただし、あるシステム・キュー・コネクター・インスタンスによるキューへの書き込み操作と、コネクターの別のインスタンスによる同じキューからの読み取り操作が同時に発生することは可能です。

システム・キュー・コネクターはサーバー API を使用してシステム・キューにアクセスします。このコネクターは、サーバー API のローカル・インターフェースとリモート・インターフェースの両方を使用します。これにより、リモート・コンピューター上で実行中の IBM Security Directory Integrator システム・キューをコネクターで操作できるようになります。リモート・コンピューターで作動するコネクターの機能は、リモート JMS サーバーに接続するシステム・キューの機能と結合して、かなり複雑なデプロイメントのシナリオに使用することが可能になります。例えば、マシン A にデプロイされた IBM Security Directory Integrator サーバーとシステム・キュー・コネクターは、リモート・サーバー API を使用してマシン B 上の IBM Security Directory Integrator サーバーおよびシステム・キューと連携して機能し、さらに、マシン C 上にデプロイされた JMS サーバーとインターフェースを取ります。

IBM Security Directory Integrator では、ActiveMQ は、デフォルトのシステム・キューとして使用され、インストール・プロセスで使用可能です。ActiveMQ のデフォルト設定は、`<TDI インストール・ディレクトリー>/etc/activemq.xml` ファイルで変更できます。

構成

ここに記載されているパラメーターを使用することで、システム・キュー・コネクターを構成できるようになります。

接続タイプ

このパラメーターは、システム・キュー・コネクターがローカル IBM Security Directory Integrator サーバーまたはリモート IBM Security Directory Integrator サーバーのシステム・キューのいずれを操作するかを定義します。このパラメーターの有効値は「*local*」および「*remote*」です。

- 値「*local*」を指定すると、コネクターがローカル・サーバー API インターフェースを使用し、ローカル IBM Security Directory Integrator サーバー上のシステム・キューを操作します。これはデフォルトです。

  値「*remote*」を指定すると、コネクターがリモート・サーバー API インターフェースを使用し、リモート IBM Security Directory Integrator サーバー上のシステム・キューを操作します。その場合は、RMI URL パラメーターが必要であり、コネクター構成 (ローカル IBM Security Directory Integrator サーバーのコネクター・パラメーターと SSL 構成の両方) がリモート IBM Security Directory Integrator サーバーの構成と一致する必要があります。

RMI URL

リモート IBM Security Directory Integrator サーバー・システムへの接続に使用するリモート・メソッド呼び出し (RMI) URL です。このパラメーターの値の例 (およびデフォルト) を以下に示します。

```
rmi://127.0.0.1:1099/SessionFactory
```

このパラメーターは、「`connectionType`」パラメーターが「`remote`」に設定されている場合にのみ有効です。

ユーザー名

サーバー API のユーザー名とパスワードによる認証メカニズムを使用したリモート IBM Security Directory Integrator サーバーに対する認証に使用されます。このパラメーターは、「`connectionType`」パラメーターが「`remote`」に設定されている場合にのみ有効です。

パスワード

リモート IBM Security Directory Integrator サーバーに対する認証に使用されます。このパラメーターは、「`connectionType`」パラメーターが「*remote*」に設定されている場合にのみ有効です。

キュー名

コネクターが処理する JMS キューの名前を指定します。イテレーター・モードでは、コネクターはこのキューから項目オブジェクトを取得します。AddOnly モードでは、コネクターはこのキューに項目オブジェクトを格納します。

タイムアウト

NULL 項目オブジェクトが戻されるまでコネクターが待機する秒数を指定します。このパラメーターに値 0 (ゼロ) を指定すると、キューに使用可能な項目オブジェクトがない場合にはコネクターが即時に戻ります。このパラメーターに負の値を指定すると、コネクターはキューで項目オブジェクトが使用可能になる時点まで無期限に待機します。デフォルト値は -1 です。

詳細ログ

このパラメーターは、デバッグ・メッセージを有効にします。このパラメーターはすべての IBM Security Directory Integrator コンポーネントに対してグローバルに定義されます。

セキュリティー、認証、および許可

以降の各セクションに示す情報を使用して、暗号化、認証、許可について理解し、これらを実行することができます。

暗号化

以下のセクションを参照すると、システム・キュー・コネクターの暗号化について知ることができます。

接続タイプが「リモート」に設定されていて、リモートの IBM Security Directory Integrator サーバーが Secure Sockets Layer (SSL) を使用するように構成されている場合、ローカルの IBM Security Directory Integrator サーバーのトラストストアが正しく構成されていれば、システム・キュー・コネクターは SSL を使用します。SSL 使用時には、コネクターはサーバー API SSL セッションを使用します。このセッションは、SSL を介して RMI を実行します。

注: 標準 JMS ドライバーでは、MQ のドライバーのみが、追加設定なしで SSL をサポートします。 MQe JMS ドライバーは、ローカルのキュー・マネージャーでのみ動作します。これは、MQe アーキテクチャーでの規定です。 JMS Script Driver は、対応するユーザー提供の JavaScript がサポートするものすべてをサポートする汎用ドライバーです。

認証

以下のセクションを参照すると、システム・キュー・コネクターの認証について知ることができます。

ユーザー名とパスワードによる認証:

以下のセクションを参照すると、システム・キュー・コネクターの認証について知ることができます。

システム・キュー・コネクターでは、リモート・サーバー API のユーザー名/パスワード認証メカニズムを使用できます。コネクター自体が認証をインプリメントするわけではありません。サーバー API に指定されるユーザー名とパスワードは、コネクター構成パラメーターとして構成されています。

SSL 証明書ベースの認証:

システム・キュー・コネクターは、クライアント SSL 証明書を使用して認証を実行できます。以下のセクションを参照すると、システム・キュー・コネクターの SSL 証明書ベースの認証について知ることができます。

これを実行できるのは、 SSL を使用し、SSL クライアント証明書がクライアント上で必要となるようにリモートの IBM Security Directory Integrator サーバー API が構成されている場合だけです。ローカル IBM Security Directory Integrator サーバーでトラスト・ストアが適切に構成されている必要があります。

SSL を使用しており、ユーザー名とパスワードがコネクター・パラメーターとして指定されている場合は、コネクターはリモート IBM Security Directory Integrator サーバーに対する認証に SSL クライアント証明書ではなく、指定されたユーザー名とパスワードを使用します。

許可

以下のセクションを参照すると、システム・キュー・コネクターの許可について知ることができます。

システム・キュー・コネクターが IBM Security Directory Integrator サーバーへの接続を確立するサーバー API セッションに対して、サーバー API の許可メカニズムが適用されます。サーバー API では、システム・キュー・コネクターは認証されると、IBM Security Directory Integrator システム・キューを使用できます。

関連情報

205 ページの『JMS コネクター』,

「*インストールと管理*」の『システム・キュー』、

TDI_install_dir/examples/SonicMQ の Sonic MQ 使用例 SystemQueue_SonicMQ_example.xml、

TDI_install_dir/examples/was_jms_ScriptDriver の WebSphere デフォルトの JMS プロバイダー使用例 SystemQConn_jmsScriptDriver_example.xml。

システム・ストア・コネクター

システム・ストア・コネクターは、基礎となるシステム・ストアへのアクセスを提供します。以下に示す情報とコードを使用して、これについて詳しく知ることができます。

システム・ストア・コネクターの主な用途は、**項目**オブジェクトをシステム・ストアの表に格納することです。ただし、このコネクターを使用して外部の Derby、DB2 9.7、Oracle、Microsoft SQL*Server、または IBM solidDB データベースに接続することも可能であり、システム・ストアとして構成されたデータベースのみに接続が限られるわけではありません。各**項目**オブジェクトは、キー属性と呼ばれる固有値により識別されます。

指定されたデータベース内に表がまだ存在しない場合は、システム・ストア・コネクターは表を新規に作成します。存在しない表に対して繰り返すと、(空の) 表が作成され、イテレーターは値を戻しません。

システム・ストア・コネクターは次のような SQL ステートメントを使用して表を作成し、その表に基本キー制約を設定します (Derby 構文)。

```
"CREATE TABLE {0} (ID VARCHAR(VARCHAR_LENGTH) NOT NULL, ENTRY BLOB );
ALTER TABLE {0} ADD CONSTRAINT {0}_PRIMARY Primary Key (ID);"
```

このコネクターは、事前設定された SQL ステートメントを多数の一般的なデータベースに入力しますが、適切なステートメントに変更することも可能です。その他のデータベース場合は、ユーザー独自の等価な SQL ステートメント (必要に応じて

複数のステートメント) を「**表の作成ステートメント (Create Table Statement)**」パラメーターに指定して入力する必要があります。このパラメーターは空にできません。空にすると、例外がスローされます。

注:

1. VARCHAR_LENGTH の値は、プロパティー・ストア (TDI-P) 内で設定されている `com.ibm.di.store.varchar.length` プロパティーから取得されます。VARCHAR_LENGTH のデフォルトは 512 に設定されています。この値を変更するには、プロパティー・ストアの `com.ibm.di.store.varchar.length` に値を設定します。
2. IBM Security Directory Integrator 6.0 との互換性を確保するため、`tdi.pesconnector.return.wrapped.entry` という別の属性があります。IBM Security Directory Integrator `global.properties` ファイルでこのプロパティーを定義し、*true* に設定している場合は、IBM Security Directory Integrator は旧バージョンでの動作に戻ります。例えば、`findEntry()` メソッド (イテレーター、ルックアップ、および更新の各モードでシステムにより使用されるメソッド) は [ENTRY: <Instance of Entry object containing Attributes passed by user>] という形式の項目オブジェクトを戻します。IBM Security Directory Integrator 6.0 では、渡された元の属性を取得するには、次のような JavaScript コードを作成する必要がありました。

   ```
   Entry e = (Entry)conn.getAttribute("ENTRY");
   ```

   このコードの挿入位置の後では、*e* にはシステム・ストアへの書き込み時に渡された元の属性が含まれていました。入力属性マップ・フックでこの処理を実現するには、*e* の属性を*作業* 項目に慎重にマップするか、あるいはこのコネクターの後にスクリプト・コンポーネントを使用しました。スクリプト・コンポーネントを使用する場合は、前述の JavaScript サンプル (conn を *work* に置換) を使用して*作業* 項目の複合属性 *entry*をアンパックしました。

   現在のバージョンの IBM Security Directory Integrator では、デフォルトで項目がアンラップされるため、ユーザーが渡すすべての属性を、項目の属性として直接使用できます。上記のスクリプトを使用する必要はありません (`tdi.pesconnector.return.wrapped.entry` 属性を *true* に設定している場合を除く)。
3. システム・ストア・コネクターは、AddOnly、更新、削除、イテレーター、ルックアップの各モードで動作します。 ただし、AddOnly、更新、および削除の各操作は、デルタ・テーブルおよびプロパティー・ストア・テーブルに対しては許可されていません。

このコネクターは、単純リンク基準と拡張リンク基準の両方をサポートしています。

原則としてこのコネクターは、その基礎となる JDBC コネクターと同様に SSL プロトコルを使用したセキュア接続を処理できます。ただし、SSL サポートをセットアップするには、ドライバー固有の構成手順を実行する必要があります。詳しくは、製造元のドライバー資料を参照してください。

構成

ここに記載されているパラメーターを使用することで、システム・ストア・コネクターを構成できるようになります。

データベース
データベースの位置。これはオプショナル・パラメーターです。ブランクのままにすると、global.properties ファイルの com.ibm.di.store.database プロパティーで構成されているとおりのシステム・ストアが使用されます。その値は「**ストア**」の「**システム・ストアの表示**」画面に表示されます。

ユーザー名
指定したデータベースへの JDBC 接続を確立するために使用するユーザーの名前。このユーザーが使用できる表のみが表示されます。これを指定しない場合、global.properties ファイルで設定されている com.ibm.di.store.jdbc.user プロパティーの値がデフォルト値として使用されます。

パスワード
指定したデータベースへの JDBC 接続を確立するために使用するユーザーのパスワード。これを指定しない場合、global.properties ファイルで設定されている com.ibm.di.store.jdbc.password プロパティーの値がデフォルト値として使用されます。

鍵属性名
項目の固有値を提供する属性名。このパラメーターは必須です。

注: 複数のキー属性の名前を「+」記号で区切って指定することもできます。システム・ストア・コネクターは、それらの属性の値を連結して単一の varchar(255) キーを生成し、固有キーとします。

選択モード
「**すべて**」、「**既存**」、または「**削除済み**」を指定します。「**既存**」および「**削除済み**」の各キーワードを使用するには、コネクターがシステム・ストア内のデルタ・テーブルを参照する必要があります。イテレーターでデルタが使用可能な場合、AssemblyLine はデータベースにシーケンス・プロパティーを格納し、ソースから読み取った各項目にシーケンス番号を追加します。このパラメーターは、デルタ・テーブルでのみ使用されます。

注: IBM Security Directory Integrator 6.0 以降でのデルタ・テーブル名は、「IDI_DS_」という接頭部を、「デルタ構成」タブの「デルタ・ストア」フィールドで指定された *ID* に付加したものになります。

表名 項目を格納する表の名前。このパラメーターは必須です。システム・ストア・コネクターは、指定された表名を使用して表を作成します (まだ存在しない場合)。

注:

1. コネクターの「コネクター構成」タブにある「選択」ボタンをクリックすると、接続されたデータベースにある表のリストが表示されます。「ユーザー名」フィールドに指定したユーザーが使用可能な表のみがリストに含まれます。

2. コネクターの「コネクター構成」タブにある「削除」ボタンをクリックすると、選択した表を削除できます。理想的には、AL を実行した後、システム・ストア・コネクターによって作成された表を削除する場合に、この「削除」ボタンを使用します。これは、デルタ・ テーブルでは利用できません。
3. 表名は、アクセスするデータベースに対して有効な名前にする必要があります。ほとんどの場合は、先頭は文字で開始する必要があり、それ以外の部分には文字、数字、および下線 (_) を使用できます。

JDBC ドライバー

このパラメーターは、JDBC ドライバーの Java クラス名を含みます (以前のバージョンにあるようなデータベース名の代わり。既存の構成は、自動的に移行されます)。

パラメーターが空のまま、または提供されたオプションのいずれかを選択した場合、「表の作成ステートメント」パラメーターは、使用されるデータベースのデフォルトの「CREATE TABLE」ステートメントによって初期化されます。**JDBC ドライバー**が指定されていない場合、システム・ストア設定で構成された JDBC ドライバーが「**表の作成ステートメント**」の正しい値を取得するときに使用されます。

このパラメーターにより、システム・ストア設定を変更することなく、システム・ストア・コネクターが別のシステム・ストア・データベースに接続できます。

指定できる値は、以下のとおりです。

- org.apache.derby.jdbc.ClientDriver
- org.apache.derby.jdbc.EmbeddedDriver
- com.ibm.db2.jcc.DB2Driver
- oracle.jdbc.OracleDriver
- com.microsoft.sqlserver.jdbc.SQLServerDriver
- solid.jdbc.SolidDriver

デフォルト値は空です。

表の作成ステートメント

選択したデータ・ソース内にテーブルを作成するために使用する「CREATE TABLE」SQL ステートメント。接続先に選んだデータベースに対応する正しい「CREATE TABLE」ステートメントを入力する必要があります。それを入力しないと、表が欠落している場合のコネクターでの表の作成に失敗します。

クローズ時に表を削除

この値を **true** に設定した場合、システム・ストア・コネクターによって作成された表は、コネクターが終了した時点で除去されます。

SQL SELECT

繰り返しの対象の項目を選択するときに実行する SELECT ステートメント。 WHERE 文節を指定します。これは、イテレーター・モードでデータ・セットを戻すための検索フィルターとして使用されます。このパラメー

ターをブランクのままにした場合は、デフォルトの構成 (SELECT * FROM TABLE) が使用されます (TABLE は「表名」フィールドに指定された名前)。

コミット
いつデータベース・トランザクションをコミットするかを制御します。オプションは以下のとおりです。

- **各データベース操作後**
- **コネクターのクローズ時**
- **手動**
- **サイクルの終わり**

手動は、システム・ストア・コネクターの *commit()* メソッドを呼び出すか、またはロジックで必要な場合は *rollback()* を呼び出す必要があることを意味します。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

コネクターの使用

システム・ストア・コネクターは、システム・ストア内に作成された表へのアクセスを提供します。以下に示す手順を使用して、コネクターの使用法について知ることができます。

システム・ストアは、JDBC ドライバーが使用可能な、どのデータベース・サーバーにも配置できます。また、システム・ストアで Derby が使用されている場合、Derby は組み込みモード (IBM Security Directory Integrator サーバー・プロセス内部) またはネットワーク・モードのいずれで実行するようにも構成できます。コネクターは、グローバルに定義されたパラメーターを解決して、デフォルトのシステム・ストアへの接続を取得できます。別のデータベースへの接続を構成するには、少なくとも以下のパラメーターを明示的に入力する必要があります。**データベース**、**ユーザー名**、**パスワード**、または **JDBC ドライバー**。

システム・ストアの各種構成についてデータベースおよび JDBC ドライバーを指定する正しい方法は、以降で説明します。

注: 以下の例は、Windows 向けになっています。

組み込み Derby サーバーをシステム・ストアとして構成した環境でのシステム・ストア・コネクターの使用

データベース: f:¥Program Files¥IBM¥IBMDirectoryIntegrator/Derby
JDBC ドライバー: org.apache.derby.jdbc.EmbeddedDriver

注: 組み込みモードで稼働する Derby サーバーは、自動的に始動され、指定したデータベースが (存在する場合) データベース内にブートされます。指定したデータベースが存在しない場合は、新しいデータベースが指定の場所に作成されます。

ネットワーク・モードの Derby サーバーをシステム・ストアとして構成した環境でのシステム・ストア・コネクターの使用

データベース: jdbc:derby://localhost:1527/E:¥TDI¥TDISysStore;create=true
JDBC ドライバー: org.apache.derby.jdbc.ClientDriver

注:

1. 重要なのは、データベースの URL 内に「create=true」フラグを指定することです。こうすると、指定のデータベースが存在しない場合に、それが作成されます。この指定は、Derby をネットワーク・モードで実行するように構成した場合に必要です。
2. ネットワーク・モードで稼働する Derby サーバーは、手動で始動する必要があります。Derby サーバーをネットワーク・モードで始動する方法について詳しくは、「*インストールと管理*」のシステム・ストア に関するセクションを参照してください。

DB2 9.7 サーバーをシステム・ストアとして構成した環境でのシステム・ストア・コネクターの使用

データベース: jdbc:db2://machine-name:50000/testDB
JDBC ドライバー: com.ibm.db2.jcc.DB2Driver

注:

1. DB2 インスタンスと DB2 データベースは、システム・ストアとして使用するより前に作成しておく必要があります。
2. 指定したインスタンスは、データベースの URL に指定したポート上で実行中であることが必要です。

関連情報

以下の場所にあるコネクター使用例:
TDI_install_dir/examples/systore、および *TDI_install_dir*/examples/SystemStore。
「*インストールと管理*」のシステム・ストア に関するセクション。

TADDM 変更検出コネクター

TADDM は、構成アイテム (CI) と、構成アイテム間の関係をサポートする管理ツールです。この構成情報は、ビジネス組織のアプリケーション・インフラストラクチャー全体をスキャンすることができる、定期的な自動ディスカバリー機能によって収集されます。ここに記載されている情報とリンクを使用することで、これについて詳しく知ることができます。

TADDM 変更検出コネクターを使用して、IBM Tivoli Application Dependency Discovery Manager (TADDM) データベースと通信し、TADDM Java API (TADDM SDK) を使用して変更を直接検出します。

TADDM 変更検出コネクターは、イテレーター・モードでのみ機能します。

収集されるデータには、デプロイ済みのソフトウェア・コンポーネント、物理サーバー、ネットワーク・デバイス、仮想 LAN、およびホスト・データがあります。

初期スキャンの実行が完了すると、その後のすべてのディスカバリーでは、インフラストラクチャー内で発生した新しい変更 (存在する場合) が検出されます。TADDM 変更検出コネクターは、TADDM データベース全体をスキャンするのではなく、これらの変更を直接取得します。

TADDM 変更検出コネクターは TADDM コネクター をベースとしているため、以下の共通機能をこのコネクターと共有しています。

- すべての CDM 関連コンポーネントとデータを交換するために、以下の両方の統一スキーマをサポートします。
  - ネイティブ・モード - 直接データ表現
  - IdML モード - CDM 対応システム間での整合性のあるデータ転送用に作成された統一スキーマ
- 構成アイテムとそれらの項目間の関係をサポートします。
- システム属性 (`$classType`、`$id`、`$cycle` など) を使用します。
- 各構成アイテムに関する追加情報 (所有管理ソフトウェア・システム、拡張属性、ドメイン属性など) を取得します。
- `iterate-all` 機能を使用して、TADDM 内に存在するすべての CDM クラス・タイプを反復処理します。

TADDM 変更検出

以下に示す情報を使用して、TADDM の変更検出について理解することができます。

TADDM 変更検出コネクターは、指定された時間間隔 (例えば「01 Jan 2010 00:00:00」から「10 May 2010 02:00:00」まで) での変更を検出します。間隔が指定されていない場合、コネクターは有効な最初の日付 (01 Jan 1970 00:00:00) から変更を検出し、それ以降のすべての変更を返します。

TADDM 変更検出コネクターでは、時間間隔内の個別のポイントで変更が取得されます。待機時間が 180 秒の場合、報告された変更は、この間隔の終わりに、開始時の内容と比較して表示されます。したがって、ある構成アイテムが作成されてから複数回更新されても、出力に表示されるのは 1 つの作成イベントと 1 つの更新イベントだけになります。構成アイテムの内容は待機間隔の終わりに更新されるため、両方のイベントで同じ属性が表示されます。構成アイテムが作成され、その後削除されると、待機間隔の開始時にも終了時にもその項目は存在しないことになるため、イベントは何も返されません。さらに詳細な検出が必要な場合は、「スリープ間隔」パラメーターの値を小さくしてください。

デルタ・タグ・サポート

TADDM 変更検出コネクターには、項目レベルのデルタ・タグ機能が組み込まれています。このコネクターは、変更タイプに応じて項目操作だけを設定します。以下に示すコード例を使用して、これについて詳しく知ることができます。

例を示します。

```
*** 項目のダンプを開始します
 Operation: delete
 Entry attributes:
 guid (replace): '30EFCB75FDAF3B3F92274803BAE6FB01'
 $classType (replace): 'sys.linux.LinuxUnitaryComputerSystem'
 $id (replace): '30EFCB75FDAF3B3F92274803BAE6FB01'
```

この例では、削除済みのモデル・オブジェクトと、その GUID が表示されます。追加または変更されたオブジェクトの場合は、属性リストも表示されます。この例では IdML モードを使用しています。

```
*** 項目のダンプを開始します
 Operation: add
 Entry attributes:
 $classType: 'sys.ComputerSystem'
 $id:  'CBEEDF3618633CDFA56039E39AE833FF',
 cdm:Guid: 'CBEEDF3618633CDFA56039E39AE833FF',
 cdm:Signature: 'testSignature',
 cdm:CreatedBy: 'administrator',
 cdm:LastModifiedTime: 1279790297569,
 cdm:DisplayName: 'testDisplayName'
```

項目操作は、以下のスクリプトに示すように、get または set することができます。

```
var entryOperation = work.getOperation(); //get entry operation

work.setOperation("modify");   //set entry operation
work.setOp ("m");
```

デルタ・タグとその他のデルタ機能について詳しくは、「*Directory Integrator の構成*」の『デルタ機能』セクションを参照してください。

TADDM 変更検出コネクターのデータ・ソース・スキーマ

以下に示す表で、TADDM 変更検出コネクターのスキーマを参照することができます。

このセクションでは、TADDM 変更検出コネクターの入力スキーマについて説明します。

入力スキーマ

以下の表に、入力マップの属性のリストを示します。

注: すべての属性がすべての状況で存在しているわけではありません。

表 35. 入力スキーマ

属性名	説明
$cycle	階層項目モデルの循環 (ループ) を禁止します。
$id	項目の固有 ID を保持します (TADDM GUID や IdML ID など)。
$classType	読み取り項目の CDM/TADDM クラス名を保持します。
cdm:ManagedSystemName フォーマットと managedSystemName フォーマット	明示属性 IdML モードに応じて、両方のフォーマットを使用できます。
cdm-rel:installedOn.cdm-trg:sys.ComputerSystem フォーマットと親フォーマット	暗黙属性 IdML モードに応じて使用できます。

表 35. 入力スキーマ (続き)

属性名	説明
$mss とその子	MSS 情報 (使用可能な場合) を保持し、そのパラメーター・オプションが有効になります。
$domain	ドメイン属性を保持します。ドメイン属性は、エンタープライズ TADDM インフラストラクチャーで役立つ場合があります。
ext:attrName フォーマットと cdm-ext: attrName フォーマット	拡張属性 追加の非 CDM フラット・データを保持します。両方のフォーマットがサポートされます。

構成

ここに記載されているパラメーターを使用することで、TADDM 変更検出コネクターを構成できるようになります。

成果物タイプ
このパラメーターを使用して、コネクターで処理するリソース・タイプ (「**構成アイテム**」または「**関係**」) を指定します。

クラス・タイプ
このパラメーターを使用して、処理される構成アイテムまたは関係のタイプを指定します。

注: TADDM からの読み取り時にこのパラメーターが指定されていない場合、イテレーターまたはルックアップの動作モードでは、すべての構成アイテムまたは関係がトラバースされます。

TADDM への書き込み時にこのパラメーターが指定されていない場合は、実行時に適切なクラス・タイプが指定されます。

サポートされるクラスを選択するには、構成エディターで「**選択**」ボタンをクリックします。

IdML モード
このパラメーターを使用して、IdML 互換データを使用して処理を行うかどうかを指定します。

深さ このパラメーターを使用して、TADDM からモデル・オブジェクトを読み取る際にトラバースする関係のレベルを指定します。

ホスト名
このパラメーターを使用して、TADDM サーバーのホスト名を指定します。

ポート このパラメーターを使用して、TADDM サーバーに接続するためのポート番号を指定します。このパラメーターが指定されていない場合は、TADDM SDK のデフォルトのポート番号が使用されます。

ユーザー名
このパラメーターを使用して、TADDM サーバーにログインするための有効なユーザー ID を指定します。

パスワード

このパラメーターを使用して、TADDM サーバーのユーザー ID に関連するパスワードを指定します。

TADDM SDK

このパラメーターを使用して、TADDM SDK のロケーションを指定します。

注: TADDM SDK は、IBM Security Directory Integrator インスタンスが TADDM コネクターと共に実行されているシステムと同じシステム上に存在している必要があります。

また、TADDM サーバーのバージョンと TADDM SDK のバージョンも一致している必要があります。例えば、TADDM 7.1.2 サーバーを使用する場合、IBM Security Directory Integrator インスタンスが TADDM コネクターと共に実行されているシステムと同じシステム上に、TADDM 7.1.2 SDK が存在している必要があります。

TADDM SDK を探すには、構成エディターの「**選択**」ボタンをクリックします。

イテレーター状態キー

このパラメーターを使用して、コネクターによって処理された最後の変更を保管します。コネクターを再始動すると、この保管されたタイム・スタンプから処理が再開されます。現在保管されているタイム・スタンプを削除して最初から変更検出を再開するには、構成エディターの「**削除**」ボタンを押してください。

開始位置

このパラメーターを使用して、「イテレーター状態キー」パラメーターがブランクの場合にコネクターが項目の読み取りを開始するタイム・スタンプを指定します。このパラメーターをブランクのままにすると、「01/01/1970 00:00:00」以降のすべての変更が検出されます。EOD (データの終わり) オプションを選択すると、コネクターの開始後の変更が返されます。

過去のタイム・スタンプを指定するには、現在の TADDM サーバー時刻を参照点として使用します。現在の TADDM サーバー時刻を取得するには、「**時刻の確認**」ボタンをクリックします。現在のサーバー時刻より先の値を指定すると、現在時刻を使用して変更が検出されます。

作成 このパラメーターを使用して、作成されたモデル・オブジェクトの変更を検出するかどうかを指定します。

更新 このパラメーターを使用して、更新されたモデル・オブジェクトの変更を検出するかどうかを指定します。

削除 このパラメーターを使用して、削除されたモデル・オブジェクトの変更を検出するかどうかを指定します。

状態キーの維持

「**状態キーの維持**」パラメーターにより、タイム・スタンプをシステム・ストアに保存するタイミングを指定します。選択項目は以下のとおりです。

- **読み取り後** - 変更を読み取るたびにシステム・ストアが更新されます。

- **サイクルの終わり** - AssemblyLine のすべてのコネクターと他のコンポーネントの評価と実行が完了した時点で、イテレーター状態キーを使用してシステム・ストアが更新されます。
- **手動** - システム・ストアは、コネクター状態の詳細では更新されません。フックまたは AssemblyLine から、`taddm-cd-connector-name.connector.saveStateKey()` メソッドを使用する必要があります。

スリープ間隔

このパラメーターを使用してスリープ間隔を指定します。この間隔が経過すると、新しい変更がないかどうかの照会が、TADDM サーバーに対して再び実行されます。このパラメーターをブランクのままにすると、デフォルト値 (180 秒) が使用されます。

タイムアウト

このパラメーターを使用して、コネクターが待機する最大タイムアウト時間 (秒数) を指定します。この時間が経過すると、次の変更検出処理が実行されます。値にゼロを指定すると、コネクターは無限に待機するか、データベース内で変更が生じるまで待機します。このパラメーターがブランクの場合、値にゼロが指定されたものと見なされます。

例えば、ポーリング間隔が 180 秒でタイムアウトが 300 秒の場合、コネクターは変更を確認してから 3 分間待機し、再び変更を確認して、変更がなければ終了します。

MSS GUID

このパラメーターを使用して、TADDM サーバーからの管理ソフトウェア・システム (MSS) GUID を指定します。MSS GUID は、読み取り時にフィルターとして使用したり、書き込み時に新規構成アイテムに追加したりすることができます。

MSS GUID を選択するには、構成エディターの「**MSS の選択**」ボタンをクリックします。

SSL の使用

このパラメーターを使用して、TADDM サーバーで SSL 接続を確立する必要があるかどうかを指定します。

Domain 属性

このパラメーターを使用して、ホストまたはポートなどのドメイン情報を使用してコネクター入力を拡張するかどうかを指定します。

注: ドメイン・データは、エンタープライズ TADDM インフラストラクチャーでのみ使用可能です。

拡張属性

このパラメーターを使用して、非 CDM データを特定の構成アイテムに対して使用するかどうか、特定のクラス・タイプの項目についてのみ定義するかどうかを指定します。

MSS 情報

このパラメーターを使用して、読み取り項目に関連する MSS についての情報を返すかどうかを指定します。MSS に関連付けられていないスタンドアロン項目も使用できます。

関連情報

『TADDM コネクター』,
641 ページの『第 6 章 資産統合スイート』.

TADDM コネクター

以下に示す情報とリンクを使用して、TADDM コネクターと、TADDM コネクターが使用する技法について理解することができます。

TADDM コネクターを使用して、TADDM Java API (TADDM SDK) を使用する IBM Tivoli Application Dependency Discovery Manager (TADDM) と通信します。

TADDM は、構成アイテム (CI) と、構成アイテム間の関係をサポートする管理ツールです。この構成情報は、ビジネス組織のアプリケーション・インフラストラクチャー全体をスキャンすることができる、定期的な自動ディスカバリー機能によって収集されます。収集されるデータには、デプロイ済みのソフトウェア・コンポーネント、物理サーバー、ネットワーク・デバイス、仮想 LAN、およびホスト・データがあります。また、TADDM は、ディスカバーされたデータ、変更とバージョンのトラッキング、およびレポート生成機能を一元化したトポロジー・ビューも提供します。

TADDM は、以下のいずれかの技法を使用して、他のアプリケーションが TADDM のデータベースにデータをロードしたり、TADDM のデータベースからデータを取り出したりできるようにします。

- TADDM API による Java アプリケーションの作成
- XML ファイル (IdML ブック) の処理と TADDM コマンド・ライン・インターフェースの使用
- TADDM SOAP または REST API の呼び出し

TADDM コネクターは、IBM データ統合イニシアチブの 1 つである Common Data Model (CDM) を使用して、転送データを表します。詳細については、 641 ページの『第 6 章 資産統合スイート』を参照してください。

TADDM コネクターは、AddOnly、削除、イテレーター、ルックアップ、更新、およびデルタの各動作モードをサポートします。

注:

TADDM コネクターの JAR ファイルは、*tdi_install_dir*/Jars フォルダー (JAR のロードに通常の IBM Security Directory Integrator ローダーが使用される場合) には存在しません。TADDM コネクターは OSGI モデルの概念を使用し、OSGI ローダーによってロードされます。

TADDM のデータ表現モデル

TADDM は、循環型階層モデルを使用して TADDM の情報を表現します。以下に示す情報と例を使用して、TADDM のデータ表現モデルについて詳しく知ることができます。

例えば OperatingSystem 項目は、Name、Release、FQDN などの単純属性と、オペレーティング・システムのインストール先である、関連する ComputerSystem 項目を持つことができます。ComputerSystem 項目は、以下の階層データ・モデルで図示される子属性として、1 つ以上の関連する CPU 属性を持つことができます。

```
OperatingSystem
 Name=Windows
 Release=1.2.3
 FQDN=www.myfqdn.com
 ComputerSystem (Parent)
  SerialNumber=1234567
  Manufacturer=IBM
  CPU (Parent)
   CPUSpeed=2500000000
   IndexOrder=5
  CPU (Parent)
   CPUSpeed=2500000000
   IndexOrder=5
   ExternalCache=1000
```

TADDM データ・モデルをサポートするために、TADDM コネクターは IBM Security Directory Integrator 項目の階層機能を使用します。

上記のデータ構造では、OperatingSystem 項目の属性 (Name など) と ComputerSystem 項目が、項目の子属性になります。CPU 属性は ComputerSystem の子属性となり、以下同様の関係になります。属性の配置について詳しくは、365 ページの『スキーマの比較』のセクションを参照してください。

以下の表に、3 つのデータ表現モデルの違いをリストします。

	項目と関係	正しい属性名とクラス名	明示属性	暗黙属性
TADDM	はい	いいえ	はい	はい
IT レジストリー	はい	はい	はい	一部のみ対応
IdML	はい	はい	はい	いいえ

Common Data Model

インフラストラクチャー内での項目とその関係を表す論理モデルを、Common Data Model (CDM) と呼びます。TADDM Common Data Model により、サポートされる項目のタイプと関係 (またはクラス・タイプ)、およびリンクされる方法が定義されます。また、このモデルにより、各クラス・タイプによってサポートされる属性とその命名規則も指定されます。

項目の属性は、その項目に関する情報を提供します。ComputerSystem 項目は、シリアル番号と製造元の詳細を保持します。これら 2 つの属性は、番号またはストリングなどの単純値を取り、明示属性と呼ばれます。

暗黙属性は、項目内での関係を表すために使用されます。以下の図に示すように、OperatingSystem は ComputerSystem にインストールされ、CDM の定義に従って、installedOn 関係を OperatingSystem (ソースとして) および ComputerSystem (ターゲットとして) と共有します。OperatingSystem 項目内の関係はその Parent 属性によって表され、ComputerSystem 項目には OSInstalled 暗黙属性が含まれます。

図 4. Common Data Model

クラス・タイプのそれぞれの*命名規則* は、1 つ以上の暗黙属性と明示属性で構成され、項目を識別する固有の方法を定義します。IdML コンシューマーに項目を登録する場合、リソースを一意的に識別して拒否されないようにするために、項目の命名規則を少なくとも 1 つは満たす属性を指定する必要があります。命名規則の属性は、*識別* 属性とも呼ばれます。上の例では、OperatingSystem の識別属性は、Name (明示) と Parent (暗黙) です。一意的に識別するには、Name と ComputerSystem の両方の属性が必要です。

データ表現フォーマット

以下のセクションで、製品で使用されるさまざまなデータ表現フォーマットについて理解することができます。

TADDM モデル:

以下に示す情報を使用して、TADDM モデル、使用される命名形式、および CDM 名について知ることができます。

TADDM には、*暗黙* 属性と*明示* 属性、構成アイテム、およびこれらの関係が含まれています。属性名は小文字で始まりますが、名前が 2 文字以上の大文字で始まる場合は例外です。例えば、Name は name に変更されますが、OSInstalled は変更されません。

TADDM は、クラス・タイプに対して以下の命名フォーマットを使用します。

- クラス・タイプのショート・ネーム。例えば、sys.ComputerSystem ではなく、ComputerSystem になります。
- TADDM ネーム・スペースの完全修飾名。例えば、com.collation.platform.topology.sys.ComputerSystem のようになります。

ショート・ネームを使用すると名前の競合が発生する場合は、クラス・タイプに CDM 名が使用されます。例えば、ショート・ネーム SSLSettings は、app.SSLSettings と app.lotus.SSLSettings の両方に一致します。この場合に、CDM 名が使用されます。

IT レジストリー・モデル:

IT レジストリー・モデルについては、ここに記載されている情報を通じて知ることができます。

IT レジストリー・モデルでは、**暗黙の naming** 属性の場合を除き、明示属性名だけがサポートされます。このモデルでは、項目は 1 回の操作で登録されます。例えば OperatingSystem 属性の場合、そのネーミング・コンテキスト (ComputerSystem) を明示属性と一緒に指定する必要があります。登録を行うには、一時的な属性を使用します。一時的な属性には、他の項目の固有 ID (GUID) または識別属性を格納することができます。

ただし、暗黙の非識別属性はサポートされません。この場合、項目間の関係を追加することにより、属性の内容が実装されます。

IdML ブックのフォーマット:

IdML ブックは、特定の管理ソフトウェア・システムにおける項目と関係の情報が格納された XML ファイルです。属性名とクラス名は CDM モデルに一致します。以下のセクションで、これについて詳しく知ることができます。

IdML ブック・フォーマットには、暗黙属性はありません。このモデルでは、情報はファイルに保管されます。このモデルでは、ComputerSystem 項目の明示属性を XML エレメントとして追加し、次に OperatingSystem の明示属性を追加して、これらの属性間の関係を示す XML エレメントを指定することができます。IdML ブックを別のシステムにインポートするには、まずブック全体の内容を解析します。このタスクは、一括ロード・ユーティリティーに委任できます。

TADDM での統一スキーマの実装

TADDM コネクターは、前のセクションで説明したさまざまなデータ・フォーマットの相違をすべて統一した、新しいスキーマを使用します。基本的な相違点や問題に対して、以下の各セクションで説明する解決策を使用することができます。

統一スキーマには、名前の変換や、命名規則の取得と検証などの追加処理が必要です。そのため、TADDM コネクターには **IdML モード**と呼ばれるオプション機能があります。

明示属性名とクラス・タイプ:

すべての名前は、それに対応する CDM バージョンと一致するように変更されます。以下の命名規則に従います。

- 属性名は大文字で始まります (Signature など)。
- 関係タイプの名前は小文字になり、ネーム・スペースは持ちません (installedOn など)。
- クラス・タイプの名前は大文字で始まり、ネーム・スペースを持ちます (app.web.GenericWebServer など)。

暗黙属性:

それぞれの暗黙属性名は、関係タイプ、関連するクラス・タイプ、および関係の方向で構成されます。以下に示す例を参照すると、暗黙属性を理解することができます。

例えば、sys.ComputerSystem の OSRunning 暗黙属性は、sys.OperatingSystem 項目に対して*逆* 方向 (つまり OperatingSystem から ComputerSystem への方向) の runsOn 関係にすることができます。

識別可能なデータ:

データ構造の例で示しているように、TADDM データ (CDM フォーマット) は階層型であり、リンクされます。ここに記載されている情報を使用することで、識別可能なデータについて知ることができます。

CPU などの項目を読み取る場合、その項目が ComputerSystem にリンクされ、それがさらに OperatingSystem にリンクされていることがあります。この場合、1 つの項目タイプを読み取るために、多くの属性を取得しなければならなくなります。CPU だけが必要な場合は、「**深さ**」パラメーターに 1 を設定して、その明示属性を取得します。暗黙属性に含まれているのは ComputerSystem の GUID だけです。IdML モードでの保管を選択している場合、このデータは識別できません。

CPU が定義された ComputerSystem は GUID だけを持つため、いずれの命名規則にも適合しません。この問題を解決するには、新しいスキーマに対して追加処理を行います。取得された各項目が命名規則に対して検証され、いずれの命名規則にも一致しない場合はスキップされます。例えば、ComputerSystem はスキップされ、そのコンテキストで CPU が識別された場合はこれもスキップされます。ただし、CPU が他の属性を持っている場合は、取得された結果に CPU が残ります。「**深さ**」パラメーターの値は増やすこともできるため、それによって ComputerSystem 項目を一意的に定義することができます。

データ表現モード

TADDM コネクターでは、以下の 2 つのデータ表現モードを使用できます。

- ネイティブ・モード - 直接データ表現
- IdML モード - CDM 対応システム間での整合性のあるデータ転送用に作成された統一スキーマ

TADDM で作業する場合は、ネイティブ・モードを使用します。例えば、CSV ファイルから読み取ったデータをインスタンス登録する場合などです。すべてのコンピューター・システムを IdML ファイルにエクスポートするには、IdML モードを使用してください (このシナリオでは IdML コネクターを使用します)。

注: このセクションで説明するモードは、AddOnly やイテレーターなどの標準的な IBM Security Directory Integrator コネクター動作モードとは異なります。TADDM コネクターには、実際の IBM Security Directory Integrator モードに関係なくスキーマを変更するスイッチがあります。

IdML モード:

以下に示す情報と例を使用して、IdML モードについて理解することができます。

IdML モードでは、すべての明示属性は CDM 名の先頭が大文字になり、cdm: という接頭部を付けて表現されます。例えば、属性 managedSystemName は cdm:ManagedSystemName になります。暗黙属性は、関係と関連クラスという 2 つの部分で構成されます。関連クラスには、関係の方向を示す情報が含まれます。 したがって、sys.ComputerSystem の属性 OSRunning は、cdm-rel:runsOn . cdm-src:sys.OperatingSystem に変更されます (わかりやすくするため、2 つの部分の間にスペースを追加してあります)。最初の部分 (cdm-rel:runsOn) は、接頭部 cdm-rel からもわかるように、関係を記述しています。2 番目の部分は、関連クラス・タイプ sys.OperatingSystem とその接頭部 (関連項目が関係のソースである場合) を表しています。

これとは逆のシナリオの場合、sys.OperatingSystem は Parent という暗黙属性を持ちます。統合モデルでは、この属性は cdm-rel:runsOn . cdm-trg:sys.ComputerSystem に変更されます。

IdML モードの場合:

- すべての明示属性とクラス・タイプで cdm: 接頭部が使用されます。
- すべての暗黙属性は、以下の 2 つの部分で構成されます。
  - cdm-rel: という接頭部が付いた関係名。
  - cdm-src: (その項目が関係のソースの場合) または cdm-trg: (その項目が関係のターゲットの場合) のいずれかの接頭部が付いた関連クラス・タイプ。

ネイティブ・モード:

ネイティブ・モードでは、クラス・タイプと属性名は変更されません。ここに記載されている情報を通じて、これについてさらに知ることができます。

標準 TADDM データに、項目のすべての暗黙属性が $implicit の子属性として追加されるだけです。例えば、明示属性 OSInstalled がある場合、これは $implicit.OSInstalled に変更されます。

スキーマの比較:

ネイティブ・モードと IdML モードを比較したデータ構造の例を以下に示します。これは、ソフトウェアがインストールされたオペレーティング・システムを示しています。

IdML モード	ネイティブ・モード
{ cdm:Name=Windows cdm:Release=3.3.4 cdm-rel:installedOn { cdm-trg:sys.ComputerSystem { cdm:Signature=12345 cdm:Manufacturer=IBM cdm:Fqdn=www.sample.com cdm-rel:contains { cdm-trg:sys.CPU { cdm:IndexOrder=2 cdm:ExternalCache=1000 cdm:CPUSpeed=2500000000 } cdm-trg:sys.CPU { cdm:IndexOrder=1 cdm:CPUSpeed=1500000000 } } } cdm-src:app.SoftwareInstallation { cdm:ProductName=Notes cdm:ManufacturerName=IBM cdm:InstalledLocation=C:¥notes } } }	{ name=Windows release=3.3.4 $implicit { parent { signature=12345 manufacturer=IBM fqdn=www.sample.com $implicit { CPU { indexOrder=2 externalCache=1000 CPUSpeed=2500000000 } CPU { indexOrder=1 CPUSpeed=1500000000 } } } softwareInstallation { productName=Notes manufacturerName=IBM installationLocation=C:¥notes } } }

注: このデータ構造の例では、いくつかのシステム属性が省略されています。システム属性について詳しくは、 366 ページの『システム属性』のセクションを参照してください。

構成エディターの「**IdML モード**」オプションを使用することで、これらの 2 つのデータ表現モードを切り替えることができます。現在のスキーマの構造を確認するには、TADDM コネクターのスキーマの照会 機能を使用します。

システム属性

TADDM コネクターは、それぞれ異なる状況で使用される 3 つのシステム属性をサポートしています。ここに記載されている情報とリンクを使用することで、システム属性について詳しく知ることができます。

TADDM からデータを読み取ると、返された各オブジェクト (ルート・オブジェクトと関連オブジェクト) の `$classType` 属性にタイプが格納されています。IdML モードが使用可能になっている場合、CDM クラス・タイプ (`sys.ComputerSystem` など) が使用されます。ネイティブ・モードでは、TADDM クラス・タイプ (`com.collation.platform.model.topology.sys.ComputerSystem` など) が使用されます。 AddOnly モードでは、この属性を使用して、登録済みの項目タイプを実行時に変更することができます。詳しくは、 372 ページの『新規モデル・オブジェクトの作成』セクションを参照してください。

`$id` 属性は、処理済みの項目の固有 ID を保持します。TADDM からデータを読み取ると、この属性に項目の GUID (属性 `guid/cdm:Guid` と同じ値) が格納されます。TADDM にデータを書き込む際に、それが別のシステムからのデータ (IdML ブックなど) の場合、`$id` 属性には、対応する IdML エレメントの ID 属性 (特殊フォーマットを持たない単純な ID) が格納されます。AddOnly モードでは、この属性を使用して循環を解決し、データ重複を防止します。

`$cycle` 属性は、TADDM データ内でループを検出する場合に使用します。最初の項目の内容をリストすると、暗黙の転送属性によって 2 番目の項目が取得されます。2 番目の項目は最初の項目に対してリンクされるため、これが原因でループが発生します。このような状況を解決するには、階層パスの上位のどこにも現在の項目が見つからないことを確認します。見つかった場合は、`$id` だけが `$cycle` 属性の値として保持されます。

コネクターの使用

ここに記載されているセクションを通じて、コネクターの使用方法を理解することができます。

基本構成

以下に示す情報とリンクを使用して、TADDM コネクターの基本構成を理解することができます。

TADDM コネクターは、TADDM サーバーとの通信を TADDM Java API に依存しています。そのため、構成エディターで TADDM SDK のパスを指定する必要があります。この SDK には、JAR ライブラリー、構成ファイル、および TADDM モデルの資料が含まれています。

TADDM サーバーのホスト名は、ユーザー名およびパスワードと一緒に指定する必要があります。TADDM サーバーが listen するポート番号が指定されていない場合、*taddm-sdk*`/etc/collation.properties` ファイルに指定されているプロパティー

com.collation.api.port の値が使用されます。デフォルトのポート番号は 9530 です。SSL 接続の場合は、プロパティー com.collation.api.ssl.port の値が使用されます。

クラス・タイプを手動で入力することも、構成エディターの「**選択**」ボタンをクリックして、TADDM でサポートされるクラス・タイプを選択することもできます。「**IdML モード**」チェック・ボックスを選択すると、リストされるタイプには cdm: という接頭部が付きます。ネイティブ・モードの場合、これらのタイプに接頭部は付きません。「**IdML モード**」フィールドをブランクのままにすると、TADDM データベース内のすべての項目が (タイプに関係なく) 反復されることになります。詳しくは、371 ページの『TADDM からの構成アイテムと関係の読み取り』セクションを参照してください。

構成エディターの「**成果物タイプ**」選択リストから「*構成アイテム*」リソース・タイプを選択した場合、「**選択**」ボタンをクリックすると、項目クラス・タイプだけがリストされます。「*関係*」リソース・タイプを選択した場合は、既知の関係だけが表示されます。このフィルター・オプションは、ネイティブ・モードのみでサポートされます。IdML モードの場合、関係はサポートされず、「**成果物タイプ**」フィールドが表示されます。

```
{
    guid=D234B679309DCE
    hostname=www.sample.com
    VMID=12
    $implicit {
        hostSystem {
            guid=6859GA5934B1
            signature=4530093
            serialNumber=12345
            $implicit {
                CPU {
                    indexOrder=12
                    CPU Speed=10000
                    guid=0BCA35EF1
                }
            }
        }
     }
}
```

「**深さ**」パラメーターを使用して、TADDM からモデル・オブジェクトを読み取る際にトラバースする関係のレベルを指定します。デフォルトでは、このレベルは設定されず、無制限の照会が実行されます。関連データだけを取得するには、このパラメーターの値を設定する必要があります。

例えば、データ構造に表示される ComputerSystem のデータを考えてみます。この例ではネイティブのスキーマ構文を使用しますが、IdML バージョンにも同じ規則が適用されます。考えられる状況は以下のとおりです。

- 深さが負の値の場合 – TADDM コネクターからエラーが返されます。
- 深さがゼロの場合 - 照会対象項目の GUID だけが返されます。この例では、項目 guid=D234B679309DCE だけが返されます。
  注: システム属性も使用することができます。
- 深さを 1 に設定した場合 - ComputerSystem 項目のすべての明示属性が返され、関連する ComputerSystem の GUID も guid=6859GA5934B1 まで返されます。
- 深さを 2 に設定した場合 - 最初の ComputerSystem の属性はすべて返されますが、2 番目では明示属性だけが返されます。$implicit.hostSystem.$implicit.CPU 属性の場合、その GUID のみ (つまり guid=0BCA35EF1 までのすべて) が返されます。
- 深さを 3 以上に設定した場合 - この例におけるすべてのデータが返されます。

MSS サポート

TADDM コネクターでは、TADDM で作業する際に、管理ソフトウェア・システム (MSS) を指定することができます。ここに記載されている情報とリンクを使用することで、MSS サポートについて詳しく知ることができます。

IBM Security Directory Integrator のユーザー・インターフェースを使用すると、使用可能な MSS をすべてリストして、必要な GUID を選択することができます。ユーザー・インターフェースの詳細については、378 ページの『構成』セクションを参照してください。

TADDM から (例えばイテレーター・モードやルックアップ・モードで) データを読み取る場合、選択した MSS によって登録されている項目だけが取得または検索されます。

新規項目を TADDM に追加する場合 (AddOnly モード)、特定の MSS に項目を登録中であることが TADDM に通知されます。その後でこの項目を読み戻す場合、構成エディターで「**MSS 情報**」チェック・ボックスが選択されている場合は、項目内の MSS データをリストする必要があります。

注: MSS について詳しくは、369 ページの『追加の属性の取得』セクションを参照してください。

TADDM の照会

以下に示す情報とコードを使用して、TADDM の照会方法を理解することができます。

クラス・タイプを指定すると、TADDM コネクターはそのタイプから項目を取得します。TADDM コネクターが MQL 照会を作成し、この照会が TADDM データベースに対して実行されます。MQL は、データベースからデータを取得するために使用される TADDM 独自の技法です。MQL は、すべての SQL 機能をサポートしているわけではありませんが、返される項目の数を制限する必要がある場合に使用すると便利です。以下の照会は、項目のクラス・タイプが指定されている場合にのみ構成されます。

```
SELECT * FROM class-type
```

深さが 0 の場合、生成された照会は以下のように変更されます。

```
SELECT guid FROM class-type
```

MQL は、複雑な WHERE 節と INNER JOIN もサポートしています。イテレーター・モードの場合、TADDM コネクターは構成エディターの「**MQL 選択**」フィールドを介して、これらの機能を使用します。このフィールドに照会を入力すると、「**クラス・タイプ**」パラメーターが指定変更されます。以下の照会例では、「Windows」という名前を持つ OperatingSystem がすべて返されます。

```
SELECT * FROM OperatingSystem WHERE OSName == 'Windows'
```

複雑な照会では、例えば、OracleInstance のポートに一致するポートを持つすべての Db2Server の名前を返し、さらに IBM で製造された ComputerSystem で稼働する各 Db2Server を返すことができます。

```
SELECT Db2Server.* FROM Db2Server, OracleInstance
  WHERE Db2Server.port == OracleInstance.port
 AND DB2Server.host.manufacturer contains 'IBM'
```

この照会では、暗黙属性と内部結合の両方を使用しています (「$implicit」属性は使用していません)。照会言語について詳しくは、TADDM MQL の資料を参照してください。

「**MQL 選択**」パラメーターは、パラメーター置換をサポートしています。次のオブジェクトを使用できます。

- Connector - TADDM コネクターのメソッドにアクセスする場合に使用します。
- config - コネクターを構成する場合に使用します。
- mc - metamerge を構成する場合に使用します。

また、「**MQL 選択**」フィールドはオートコンプリート機能をサポートし、「リンク基準」ダイアログ・ボックスは、条件を通常の MQL ステートメントに転送します。

「**MQL 選択**」フィールドは、IdML モードに依存しています。IdML モードが有効になっていると、追加のフィルタリング機能がオンになります。 364 ページの『IdML モード』セクションで説明している統一スキーマの定義に従って、クラス・タイプと属性名 (暗黙と明示) の両方を指定することができます。

```
 SELECT cdm:Signature FROM cdm:sys¥.ComputerSystem
WHERE cdm-reln:virtualizes.cdm-trg:sys¥.ComputerSystem.cdm:NumCPUs == '3'
```

この照会は、3 つの CPU を持つコンピューター・システム上に存在するすべての仮想コンピューター・システムのシグニチャーを返します。

注: クラス・タイプ名はすべてエスケープされます (例: `cdm:sys¥.ComputerSystem`)。この要件は重要です。なぜなら、IBM Security Directory Integrator におけるドットは属性間の分離文字を表すため、ドットをエスケープしないと、`ComputerSystem` が `sys` エレメントのサブエレメントとして解釈されるからです。

「**フェッチ・サイズ**」パラメーターは、TADDM からデータを読み取る際に使用されるバッファーのサイズを示します。このサイズを変更することにより、IBM Security Directory Integrator ソリューションのパフォーマンスを微調整することができます。例えば、コネクターが TADDM サーバーから 100 項目を (深さ 1 で) 読み取る際に、フェッチ・サイズが 100 を超えていれば、バルク要求を 1 回実行するだけですべてのデータが取得されます。

追加の属性の取得

TADDM コネクターが提供する以下の追加属性を使用することができます。

表 36. 追加の属性

属性	説明
Domain 属性	この属性には、読み取り項目が属する TADDM ドメインに関する詳細が含まれています。この属性は、エンタープライズ TADDM サーバーに対して照会を実行する場合のみ使用できます。この属性を使用して、データを提供する TADDM サーバーと、エンタープライズ・アーキテクチャー内のその他すべてのサーバーを区別することができます。 ドメイン属性 (ポートやホスト名など) は、存在する場合、`$domain` 属性の下に表示されます。

表 36. 追加の属性 (続き)

属性	説明
拡張属性	これらの属性は、各項目に付加できます。この属性により、項目の整合性を損なうことなく、CDM モデル外部の情報を保管できます。例えば、コメント属性として使用される、読み取り項目の短い説明テキストを保持する拡張属性などが考えられます。IdML モードでは、このような拡張属性には `cdm-ext:` という接頭部が付きます。ネイティブ・モードの場合、接頭部は `ext:` だけです。
明示的な関係	この機能がサポートされるのは、IdML モードの場合だけです。 TADDM には、以下の 2 つのタイプの関係があります。 • 暗黙タイプ - 読み取り項目の暗黙属性によって定義されます。 • 明示タイプ - 既存の項目の間に追加されます。 **注:** 暗黙的な関係と明示的な関係はオーバーラップする場合があるため、返された項目に明示的な関係を追加する前に、TADDM コネクターにより、同じ関係が既に暗黙タイプとして追加されていないかどうかが確認されます。このアクションにより、項目でのデータ重複を防止し、整合性を保つことができます。明示的な関係は、構成されている検索の深さに従います。そのため、明示的な関係に従うと、結果として深さの制限がバイパスされ、その関係が無視されることになります。

MSS 情報	このオプションを使用すると、`$mss` システム属性の下に MSS 情報が返されます。この情報には、読み取り項目を登録した各 MSS の詳細が含まれています。そのため、複数の MSS エレメントが存在する場合があります。以下の例は、IdML モードで 2 つの MSS を持つ属性を示しています。
	`{`
	`  $mss {`
	`  cdm:process.ManagementSoftwareSystem {`
	`  cdm:MSSName=MSS1`
	`  cdm:Guid=12BA843..2FF`
	`  }`
	`  cdm:process.ManagementSoftwareSystem {`
	`  cdm:Guid=12BA843..2FF`
	`  cdm:Hostname=www.sample.com`
	`  cdm:ProductName=TDI`
	`  cdm:ManufacturerName=IBM`
	`  }`
	`  }`
	`}`

特定のモデル・オブジェクトの検索

ルックアップ動作モードの場合、TADDM コネクターは、TADDM MQL 機能を使用して特定の項目を検索します。項目を検索するには、構成エディターでリンク基準を指定するか、カスタム・スクリプトを使用して拡張基準を作成します。

結果の MQL 照会を作成するための式は以下のとおりです。

```
SELECT * FROM class-type WHERE criteria
```

「**深さ**」パラメーターの値がゼロ (0) の場合、「*」の代わりに guid 属性を使用します。

MQL には拡張基準機能が組み込まれているため、IBM Security Directory Integrator リンク基準の一致条件がすべてサポートされ、対応する MQL の同義語にマップされます。

同様に、「**MQL 選択**」パラメーターに対しても、「**IdML モード**」が有効になっていれば、ルックアップ・モードの基準で統一スキーマ名がサポートされます。CDM クラス・タイプでは、app.db.db2.Db2Server や sys.CPU などのように「.」を使用しないでください。

TADDM からの構成アイテムと関係の読み取り

TADDM コネクターで読み取る項目のタイプを指定する必要があります。

TADDM は、テーブル構造で項目を公開します。この構造では、項目のクラス・タイプがテーブル名を表します。詳しくは、368 ページの『TADDM の照会』セクションを参照してください。

TADDM コネクターは、イテレーター・モードとルックアップ・モードの両方で、明示的に構成されたクラス・タイプを使用しない処理をサポートし、すべての TADDM 項目をトラバースします。この機能を使用して、すべての項目を取得するか (イテレーター・モードの場合)、TADDM データ全体に対して一般的な検索を実行することができます (ルックアップ・モードの場合)。

この動作は、構成済みの TADDM サーバーに対する複数の照会 (各タイプに 1 つずつ) によって実行されるため、いずれかの照会が失敗しても TADDM コネクターは終了せず、デバッグ・メッセージをログに記録して次の照会が続行されます。

TADDM データベースからすべての項目を読み取る場合は、「CTGDJN095E Unable to get the IdML compliant name of attribute 'attr-name' from class 'class-name')」という例外が発生すると、TADDM コネクターが途中で終了します (IdML モードが使用可能になっている場合)。詳しくは、376 ページの『TADDM コネクターのトラブルシューティング』セクションを参照してください。

モデル・オブジェクトの削除

削除動作モードを使用して、不要な項目と関係を TADDM から消去することができます。

削除動作モードを使用して、不要な項目と関係を TADDM から消去します。このモードでは、ルックアップ操作を実行してから、ディスカバーされた項目の削除が試行されます。削除操作の場合、TADDM API で必要になるのは項目の GUID だけであるため、コネクターは $id システム属性を使用します。

複数の一致項目が検出された場合は、「**複数項目時**」フックが開始されます。この場合、ディスカバーされた項目を連続して削除することができます。また、検出された項目のすべての $id を最初にディスカバーされた項目に追加して、コネクターに提供することもできます。コネクターが複数の ID を検出した場合、これらの ID は 1 回の TADDM 要求ですべて削除されます。

フック・スクリプトは、以下のようになります。

```
var first = thisConnector.getFirstDuplicateEntry();

var next = thisConnector.getNextDuplicateEntry();
while (next != null)
{
 var id = next.getString("$id");
 first.addAttributeValue("$id", id);
 next = thisConnector.getNextDuplicateEntry();
}

thisConnector.setCurrent(first);
```

新規モデル・オブジェクトの作成

ここに記載されている情報を使用することで、新規モデル・オブジェクトを作成できるようになります。

AddOnly 動作モードの場合、TADDM コネクターは TADDM 内にオブジェクトを作成します。CDM の命名規則に従い、TADDM は、オブジェクト作成プロセスとオブジェクト更新プロセスを同様の処理とみなします。提供された属性が既存の TADDM オブジェクトの属性と同じであれば、属性は単に更新されます。

項目を登録するには、識別属性を指定し、関連項目でそれらの属性を使用できるようにする必要があります。あるいは、識別属性を指定する代わりに、関連項目の `$classType` 属性と `guid` 属性を設定することもできます。こうしたモデル・オブジェクトを作成するには、このデータだけでは不十分ですが、この場合、TADDM コネクターはそのモデル・オブジェクトを処理しようとします。該当のデータが検出されると、作成済みのモデル・オブジェクトに追加されます。GUID ルックアップ関数を利用するには、`$classType` 属性と `$id` 属性のみを考慮します。この機能を使用して、TADDM 内の 2 つの既存の項目間の明示的な関係を追加することができます。

`$classType` システム属性は実行時に変更できるため、作成された項目のタイプを TADDM コネクターで切り替えることができます。AddOnly モードでは、`$cycle` システム属性も使用できます。この属性は、関連項目が現行項目の階層パスの上位で既に指定されている場合に、その関連項目の識別属性の重複を回避するために使用すると便利です。元の場所からすべての情報を取得するには、`$cycle` 属性内の項目の固有 ID を設定する必要があります。

新規のモデル・オブジェクトが TADDM に登録されると、その GUID が *conn* 項目を使用して返されます。この値は、「**追加後**」フックで以下のスクリプト・コードを使用すると、取得することができます。

```
var guid = conn.getString("$id");
```

設計時の命名規則の検証

TADDM コネクターには、出力マップにマップされた属性を、選択された CDM クラス・タイプの命名規則に対して検証するためのサポートが用意されています。

注: TADDM コネクターは、TADDM に接続され、指定のクラス・タイプを持つように構成されている必要があります。このように構成されていない場合、「エラー・ログ」ビューにエラーが記録されます。

構成エディターで、コネクターの出力マップにある「**妥当性検査**」ボタンをクリックすると、すべての属性が検証され、結果が「**問題**」ビューに表示されます。指定したクラスの命名規則ごとに 1 つのメッセージがあります。少なくとも 1 つの命名規則が満たされると、「問題」ビューのすべてのメッセージに情報のマークが付きます。どの命名規則も満たしていない場合は、メッセージにエラーのマークが付きます。各メッセージには、命名規則を満たすために追加または削除する必要がある属性が表示されます。

既存のモデル・オブジェクトの更新

更新動作モードでは、既存のモデル・オブジェクトを更新することができます。

TADDM では、オブジェクトの作成とオブジェクトの更新は同じプロセスです。詳しくは、372 ページの『新規モデル・オブジェクトの作成』セクションを参照してください。ただし、既存のモデル・オブジェクトを更新する際に考慮すべき追加の特性がいくつかあります。

単一値の暗黙属性を更新しようとした場合、その属性が存在していないと、TADDM は属性を作成します。存在している場合は、その属性が更新されます。

元の項目	項目の更新	結果の項目

{
$classType=sys.ComputerSystem
Guid=A027...2ECB
displayName=777-888
signature=777-888
$implicit {
OSRunning{
$classType=sys.OperatingSystem
$id=3F62F...B5AD
guid=3F62...B5AD
displayName=976063427
FQDN=fqdn/976063428
}
}
}
	{
	$implicit {
	OSRunning{
	displayName=976063429
	}
	}
		{
		$classType=sys.ComputerSystem
		Guid=A027...2ECB
		displayName=777-888
		signature=777-888
		$implicit {
		OSRunning{
		$classType=sys.OperatingSystem
		guid=3F62...B5AD
		displayName=976063429
		FQDN=fqdn/976063428
		}
		}
		}

複数値の暗黙属性のいずれかの値を更新するには、その値の GUID を出力マップに指定する必要があります。これを指定しないと、既存の値が更新されずに、新しい値が追加されることになります。

GUID を指定した場合の項目の更新は、以下のようになります。

元の項目	項目の更新	結果の項目

{
$classType=sys.ComputerSystem
Guid=A027...2ECB
displayName=777-888
signature=777-888
$implicit {
OSInstalled{
$classType=sys.OperatingSystem
guid=3F62...B5AD
displayName=976063427
FQDN=fqdn/976063428
}
}
}
	{
	$implicit {
	OSInstalled{
	guid=3F62...B5AD
	displayName=976063429
	}
	}
		{
		$classType=sys.ComputerSystem
		Guid=A027...2ECB
		displayName=777-888
		signature=777-888
		$implicit {
		OSInstalled{
		$classType=sys.OperatingSystem
		guid=3F62...B5AD
		displayName=976063429
		FQDN=fqdn/976063428
		}
		}
		}

GUID を指定しない 場合の項目の更新は、以下のようになります。

元の項目	項目の更新	結果の項目

{
$classType=sys.ComputerSystem
Guid=A027...2ECB
displayName=777-888
signature=777-888
$implicit {
OSInstalled{
$classType=sys.OperatingSystem
guid=3F62...B5AD
displayName=976063427
FQDN=fqdn/976063428
}
}
}
	{
	$implicit {
	OSInstalled{
	displayName=976063429
	FQDN=fqdn/976063429
	}
	}
		{
		$classType=sys.ComputerSystem
		Guid=A027...2ECB
		displayName=777-888
		signature=777-888
		$implicit {
		OSInstalled{
		$classType=sys.OperatingSystem
		guid=3F62...B5AD
		displayName=976063427
		FQDN=fqdn/976063428
		}
		OSInstalled{
		$classType=sys.OperatingSystem
		guid= 6030...E574
		displayName=976063429
		FQDN=fqdn/976063429
		}
		}
		}

また、TADDM コネクターを使用して明示属性の値を NULL に更新することにより、既存のモデル・オブジェクトから明示属性を削除することができます。値を NULL に更新するには、出力マップでデフォルトの NULL 動作を「**NULL 値を戻す**」に変更します (構成エディターの「**詳細の表示**」ボタンをクリック)。

注: 階層項目を正確に比較することはできないため、TADDM コネクターは「変更の計算」機能をサポートしていません。

デルタ・モードのサポート

ここに記載されている情報を使用することで、デルタ・モードのサポートについて知ることができます。

デルタ・モードの場合、TADDM コネクターは特定のデルタ情報を受け取ります。コネクターは、受け取った情報に基づいて、指定された項目に対する追加、更新、または削除の操作を実行します。

注: 項目を更新または削除するには、リンク基準を設定する必要があります。例えば、基準 `$id equals $$id` を削除タグ付きの項目と一緒に指定すると、TADDM で項目が削除されます。

TADDM コネクターのデータ・ソース・スキーマ

TADDM コネクターの入力スキーマと出力スキーマを、以下に示す各表で参照できます。

入力スキーマ

以下の表に、入力マップの属性のリストを示します。

注: すべての属性がすべての状況で存在しているわけではありません。

表 37. 入力スキーマ

属性名	説明
`$cycle`	階層項目モデルの循環 (ループ) を禁止します。
`$id`	項目の固有 ID を保持します (TADDM GUID や IdML ID など)。
`$classType`	読み取り項目の CDM/TADDM クラス名を保持します。
`cdm:ManagedSystemName` フォーマットと `managedSystemName` フォーマット	明示属性 IdML モードに応じて、両方のフォーマットを使用できます。
`cdm-rel:installedOn.cdm-trg:sys.ComputerSystem` フォーマットと親フォーマット	暗黙属性 IdML モードに応じて使用できます。
`$mss` とその子	MSS 情報 (使用可能な場合) を保持し、そのパラメーター・オプションが有効になります。
`$domain`	ドメイン属性を保持します。ドメイン属性は、エンタープライズ TADDM インフラストラクチャーで役立つ場合があります。
`ext:attrName` フォーマットと `cdm-ext: attrName` フォーマット	拡張属性 追加の非 CDM フラット・データを保持します。両方のフォーマットがサポートされます。

出力スキーマ

以下の表に、出力マップの属性のリストを示します。

注: すべての属性がすべての状況で存在しているわけではありません。

表 38. 出力スキーマ

属性名	説明
`$cycle`	階層項目モデルの循環 (ループ) を禁止します。
`$id`	項目の固有 ID を保持します (TADDM GUID や IdML ID など)。
`$classType`	読み取り項目の CDM/TADDM クラス名を保持します。

表 38. 出力スキーマ (続き)

属性名	説明
cdm:ManagedSystemName フォーマットと managedSystemName フォーマット	明示属性 IdML モードに応じて、両方のフォーマットを使用できます。
cdm-rel:installedOn.cdm-trg.sys.ComputerSystem フォーマットと親フォーマット	暗黙属性 IdML モードに応じて使用できます。

インストール後の作業

TADDM コネクターは TADDM SDK に依存しています。TADDM API JAR ファイルは、IBM Security Directory Integrator には付属していません。TADDM コネクターの使用を開始する前に、以下に示すタスクを実行して、サポートされていないモード (サーバー・モードや CallReply モードなど) が使用されることを防ぎ、サポートされているクラス・タイプのリストを取得する必要があります。

1. TADDM サーバーから TADDM SDK の圧縮アーカイブ・ファイルをコピーし、IBM Security Directory Integrator が稼働しているシステムに解凍します。TADDM SDK の圧縮ファイルは、*taddm-home*/dist/sdk に格納されています。

   TADDM 7.2 SDK 用に作成されるディレクトリー構造は以下のとおりです。

   ```
   /sdk
    |--/adaptor - TADDM Discovery Library Adaptor 1.0 を格納
    |--/bin - 便利なシェル・スクリプトとバッチ・ファイルを格納
    |--/dla - IBM® Discovery Library IdML Certification Tool を格納
    |--/doc - 英語版の PDF ファイルと他の文書ファイルを格納
    |--/etc - 構成プロパティーを格納
    |--/examples - サンプルを格納
    |--/lib - サーバーとクライアントのランタイム・ライブラリーを格納
    |--/log - ランタイム・ログ・スキーマの XML スキーマを格納
   ```

2. 以下の TADDK API JAR ファイルを IBM Security Directory Integrator のクラス・パスにコピーします。

   - *taddm-sdk*/lib/taddm-api-client.jar
   - *taddm-sdk*/lib/platform-model.jar

   TADDM 7.1.2 の場合、JAR ファイルは *taddm-sdk*/clientlib に格納されます。これらのファイルを、IBM Security Directory Integrator の /jars ディレクトリー *TDI_install_dir*/jars/3rdparty/IBM にコピーするか、solution.properties の com.ibm.di.loader.userjars プロパティーで指定された場所にコピーします。

注: IBM Security Directory Integrator サーバー上で稼働するすべての TADDM コネクターで使用できる TADDM SDK は 1 つだけです。これは、TADDM SDK による制限です。TADDM SDK では、そのパスがシステム・プロパティーとして設定されている必要があります。バージョンの異なる TADDM サーバー (TADDM 7.1 と TADDM 7.2 など) で、2 つの TADDM コネクターを同時に使用することはできません。

TADDM コネクターのトラブルシューティング

以下の情報を使用して、TADDM コネクターを使用する際に発生する可能性のある問題のトラブルシューティングを行います。

TADDM コネクターの使用時に AccessException がスローされる

以下に示す情報を使用して、「AccessException のスロー」エラーのトラブルシューティングの方法を知ることができます。

問題

TADDM コネクターを初めて開始する前に、別のコンポーネントによって制限的なセキュリティー・マネージャーが設定されていると、例外がスローされます。TADDM コネクターは、マネージャーがすべて許可することを前提としているため、制限的なマネージャーが設定されている場合は正常に機能することができません。

ソリューション

以下のいずれかのタスクを実行します。

- IBM Security Directory Integrator サーバーを再始動し、最初に TADDM コネクターを実行します。
- デフォルトで、すべてのアクセス権を付与します。

すべてのアクセス権を付与するには、以下の手順を実行します。

1. *TDI_install_dir*/jvm/jre/lib/security/java.policy ファイルを編集します。
2. 以下の項目を指定します。

```
grant {
      permission java.security.AllPermission;
};
```

注: この変更は、別の Java ポリシー・ファイルに対しても行うことができます (ファイルが正しくセットアップされている場合)。詳しくは、ポリシー・ファイルに関する説明を参照してください。

IdML モードで TADDM からデータを読み取る際に例外がスローされる

以下に示す情報を使用して、例外スローのエラー (IdML モードでの TADDM からのデータ読み取り時) のトラブルシューティングの方法を知ることができます。

問題

この問題は、TADDM の暗黙属性のストレージで不整合があることが原因で発生します。通常は、TADDM の各暗黙属性には、それに対応する関係の名前が含まれています。この情報が存在しない場合、TADDM コネクターは読み取りデータの正しい IdML モデルを構築できなくなるため、例外がスローされます。この問題が発生するのは、クラス `meta.UserDataMeta` とクラス `process.CompositeAttributeDef` の `Parent` 暗黙属性の場合だけです。

ソリューション

ネイティブ・モードに切り替えると、変換を回避することができます。ネイティブ・モードに切り替えるには、構成エディターの「**IdML モード**」チェック・ボックスをクリアします。

この例外は、TADDM の全項目に対して反復/ルックアップを実行した場合にスローされます。他の CDM 対応コンポーネント (IdML コネクターなど) を処理する場合は、IdML モードにする必要があります。このシナリオでは、TADDM コネクターの構成で「イテレーター・エラー/ルックアップ・エラー」フックを指定変更する必要があります。

構成

ここに記載されているパラメーターを使用することで、TADDM コネクターを構成できるようになります。

基本的な構成情報については、 366 ページの『基本構成』セクションを参照してください。

成果物タイプ
このパラメーターを使用して、TADDM コネクターで処理するリソース・タイプ (「**構成アイテム**」または「**関係**」) を指定します。

クラス・タイプ
このパラメーターを使用して、処理される構成アイテムまたは関係のタイプを指定します。

注: TADDM からの読み取り時にこのパラメーターが指定されていない場合、イテレーターまたはルックアップの動作モードでは、すべての構成アイテムまたは関係がトラバースされます。

TADDM への書き込み時にこのパラメーターが指定されていない場合は、実行時に適切なクラス・タイプが指定されます。

サポートされるクラスを選択するには、構成エディターで「**選択**」ボタンをクリックします。

IdML モード
このパラメーターを使用して、IdML 互換データを使用して処理を行うかどうかを指定します。

深さ このパラメーターを使用して、TADDM からモデル・オブジェクトを読み取る際にトラバースする関係のレベルを指定します。

ホスト名
このパラメーターを使用して、TADDM サーバーのホスト名を指定します。

ポート このパラメーターを使用して、TADDM サーバーに接続するためのポート番号を指定します。このパラメーターが指定されていない場合は、TADDM SDK のデフォルトのポート番号が使用されます。

ユーザー名
このパラメーターを使用して、TADDM サーバーにログインするための有効なユーザー ID を指定します。

パスワード
このパラメーターを使用して、TADDM サーバーのユーザー ID に関連するパスワードを指定します。

TADDM SDK

このパラメーターを使用して、TADDM SDK のロケーションを指定します。

注: TADDM SDK が、IBM Security Directory Integrator インスタンスが TADDM コネクターと共に実行されているシステムと同じシステム上に存在していることを確認してください。

TADDM SDK を探すには、構成エディターの「**選択...**」ボタンをクリックします。

MSS GUID

このパラメーターを使用して、TADDM サーバーからの管理ソフトウェア・システム (MSS) GUID を指定します。MSS GUID は、読み取り時にフィルターとして使用したり、書き込み時に新規構成アイテムに追加したりすることができます。

MSS GUID を選択するには、構成エディターの「**MSS の選択**」ボタンをクリックします。

MQL 選択

カスタム MQL 照会を使用した TADDM の照会でこのパラメーターを使用し、リンク基準を指定することによって、返される構成アイテムの数を制限します。2 つのクラス・タイプ間の結合操作を実行する際にも、このパラメーターを使用できます。

SSL の使用

このパラメーターを使用して、TADDM サーバーで SSL 接続を確立する必要があるかどうかを指定します。

フェッチ・サイズ

このパラメーターを使用して、TADDM からデータを取得する際に使用するバッファー・サイズを指定します。

Domain 属性

このパラメーターを使用して、ホストおよびポートなどのドメイン情報を使用してコネクター入力を拡張するかどうかを指定します。

注: ドメイン・データは、エンタープライズ TADDM インフラストラクチャーでのみ使用可能です。

拡張属性

このパラメーターを使用して、非 CDM データを特定の構成アイテムに対して使用するかどうか、特定のクラス・タイプの項目についてのみ定義するかどうかを指定します。

明示的な関係

このパラメーターを使用して、読み取り項目の明示的な関係を検査するかどうかを指定します。このオプションは、IdML モードでのみ使用可能です。

MSS 情報

このパラメーターを使用して、読み取り項目に関連する MSS についての情報を返すかどうかを指定します。MSS に関連付けられていないスタンドアロン項目も使用できます。

関連情報

354 ページの『TADDM 変更検出コネクター』,
641 ページの『第 6 章 資産統合スイート』.

TCP コネクター

TCP コネクターは、トランスポート用に TCP ソケットを使用するトランスポート・コネクターです。TCP コネクターは、イテレーター・モードおよび AddOnly モードでのみ使用できます。

イテレーター・モード

イテレーター・モードで使用する場合、TCP コネクターは、特定のポート上で着信 TCP 呼び出しを待ちます。このモードを使用するときは、以下に示すプロパティーを使用できます。

接続が確立されると、**getnext** メソッドが以下のプロパティーを持つ項目を戻します。

socket TCP ソケット・オブジェクト (例えば、TCP 入力および出力ストリーム)

in ソケットの入力ストリームを使用する BufferedReader のインスタンス

out ソケットの出力ストリームを使用する BufferedWriter のインスタンス

in および **out** オブジェクトは、TCP 接続からのデータの読み取り、または TCP 接続へのデータの書き込みのために使用されます。例えば、次のようにして、単純なエコー・サーバーをインプリメントすることができます (コードを **GetNext 後**フックに書き込みます)。

```
var ins = conn.getProperty("in");
var outs = conn.getProperty("out");
var str = ins.readLine();
outs.write("You said==>"+str+"<==");
outs.flush();
```

BufferedWriter を使用しているため、データが実際にこの接続で発信されるようにするために、out.flush() メソッドを呼び出すことが重要です。

パーサーを指定した場合は、BufferedReader はそのパーサーに渡され、パーサーがストリーム上で送信されるデータを読み取って解釈します。その結果、戻される項目には、上記にリストしたプロパティー (**socket**、**in**、および **out**) の他に、パーサーが割り当てる属性が組み込まれます。

TCP コネクターが *Listen Mode=true* として構成されている場合は、*getnext* メソッドに対する各呼び出し間で接続がクローズされます。*Listen Mode=false* の場合は、リモート・ホストへの接続は、TCP コネクターがアクティブの間は (例えば、AssemblyLine が終了するまで) オープンのままになっています。

注: このコネクターの listen モード・パラメーターは、382 ページの『TCP サーバー・コネクター』の動作で構成しないでください。この動作のコネクターは、TCP 要求の受信を (必要に応じて複数並行して) 受け入れるのにより適しているコネクタ

ーです。*Listen Mode=true* に関連付けられている機能は非推奨であり、将来のバージョンのコネクターでは削除され、発信接続のみにコネクターを使用するよう構成できるようになります。

AddOnly モード

以下に示す情報を使用して、TCP コネクターの AddOnly モードでの使用方法について知ることができます。

TCP コネクターがこのモードになっているときは、デフォルトのインプリメンテーションでは、項目は、有用とは言えないストリング形式で書き込まれます。通常は、特定出力を行うには、パーサーを指定するか、**追加の指定変更**フックを使用します。「**追加の指定変更**」フックでは、次のように、コネクター・インターフェースの getReader() および getWriter() メソッドを呼び出して、**in** または **out** オブジェクトにアクセスします。

```
var in = mytcpconnector.connector.getReader();
var out = mytcpconnector.connector.getWriter();
```

また、**追加前**フックおよび**追加後**フックを使用して、パーサーからの出力の前後にヘッダーまたはフッターを挿入することもできます。

構成

ここに記載されているパラメーターを使用することで、TCP コネクターを構成できるようになります。

TCP ホスト
接続するリモート・ホスト (**serverMode = false** の場合)。

TCP ポート
serverMode の値に応じて、接続または listen する TCP ポート番号。

SSL の使用
これをチェックすると、コネクターは接続時に Secure Sockets Layer (SSL) をデプロイします。

listen モード
(非推奨。TCP サーバー・コネクターを使用してください) **true** の場合は、繰り返し操作で着信要求が listen されます。 **false** の場合は、繰り返し操作ではリモート・サーバーに接続されます。

SSL によるクライアント認証を要求する
(非推奨) これをチェックすると、listen モードで SSL が使用可能である場合 (着信接続を listen している場合) にクライアント認証が必要になります。

接続バックログ
(非推奨) 着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

「パーサー」ペインからこのコネクターのパーサーを選択できます。パーサーを選択するには、左上の「**パーサーの選択**」ボタンをクリックします。

関連情報

119 ページの『ファイル・コネクター』,
56 ページの『TCP/URL の直接スクリプト記述』,
『TCP サーバー・コネクター』
406 ページの『URL コネクター』.

TCP サーバー・コネクター

このコネクターは、サーバー・モードとイテレーター・モードでのみ使用できます。

サーバー・モードでは、このコネクターは指定されたポート上で着信 TCP 接続を待機し、着信要求を取り扱うための新規のスレッドを作成します。新規のスレッドが開始されると、元のサーバー・モードのコネクターは listen モードに戻ります。その新規作成のスレッドが完了すると、そのスレッドは停止し、TCP 接続はクローズされます。

イテレーター・モードでは、コネクター単一スレッドであり、ローカル・マシンの IP アドレスと指定のポートでの接続を待機します。接続を受信したら、クライアントにより接続が終了されるまで、コネクターは受信データに基づいて項目を生成します。

構成

ここに記載されているパラメーターを使用することで、TCP サーバー・コネクターを構成できるようになります。

TCP ポート
着信接続を listen する TCP ポート。

接続バックログ
着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

SSL の使用
これをチェックすると、コネクターは接続時に Secure Sockets Layer (SSL) をデプロイします。

クライアント認証を必要とする
これをチェックすると、コネクターはクライアントに対し、構成済み IBM Security Directory Integrator トラストストアと突き合わせるためのクライアント・サイド SSL 証明書を提供するよう求めます。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

コネクター・スキーマ

このコネクターでは、入力属性マップで以下に示すプロパティーを使用できます。

tcp.originator
コネクター・オブジェクト。

event.originator
コネクター・オブジェクト。これは、**tcp.originator** に格納されているオブジェクトと同一です。この属性により、TCP ポート EventHandler (現時点では廃止)との互換性が確保されます。

tcp.inputstream
TCP ソケット入力ストリーム (java.io.InputStream)

event.inputstream
TCP ソケット入力ストリーム (java.io.InputStream)。これは、**tcp.inputstream** に格納されているオブジェクトと同一です。この属性により、TCP ポート EventHandler (廃止) との互換性が確保されます。

tcp.outputstream
TCP ソケット出力ストリーム (java.io.OutputStream)。

event.outputstream
TCP ソケット出力ストリーム (java.io.OutputStream)。これは、**tcp.outputstream** に格納されているオブジェクトと同一です。この属性により、TCP ポート EventHandler (廃止) との互換性が確保されます。

tcp.remoteIP
リモート IP アドレス (ドット表記)。

tcp.remotePort
リモート TCP ポート番号。

tcp.remoteHost
リモート・ホスト名。

tcp.localIP
ローカル IP アドレス (ドット表記)。

tcp.localPort
ローカル TCP ポート番号。

tcp.localHost
ローカル・ホスト名。

tcp.socket
TCP ソケット・オブジェクト (java.net.Socket)。

TCP サーバー・コネクターでは、出力属性マップは使用されません。このコネクターは、操作完了時にクライアント・アプリケーションとの接続を終了します。

tcp.inputstream 属性値と **tcp.outputstream** 属性値はそれぞれ、AssemblyLine において、クライアント要求を読み取り、応答を書き込むためにスクリプトにより使用されるものです。

関連情報

380 ページの『TCP コネクター』.

タイマー・コネクター

タイマーは、指定された時間が経過するまで待機し、スリープ状態から戻ると、AssemblyLine を再開します (つまり、新しいサイクルを開始します)。このコネクターは、イテレーター・モードでのみ使用できます。

属性マッピングで作業項目にマップできる属性は 1 つで、`java.util.Date` タイプのタイム・スタンプです。この属性には、コネクターがサイクルを開始した時刻が入ります。

デルタ機能をこのコネクターとともに使用することは、あまり意味がありません。

構成

ここに記載されているパラメーターを使用することで、タイマー・コネクターを構成できるようになります。

月　タイマー・コネクターを実行する月を選択します (* = 任意)

日　タイマー・コネクターを実行する日 (* = 任意)

曜日　タイマー・コネクターを実行する曜日を選択します (* = 任意)

時間　タイマー・コネクターを実行する時刻の時間 (* = 任意)。複数の値をコンマで区切って指定できますが、これらの値は昇順に並べる必要があります。

分　タイマー・コネクターを実行する時刻の分。複数の値をコンマで区切って指定できますが、これらの値は昇順に並べる必要があります。

スケジュール
これは、コネクターの実行時刻を設定する UNIX crontab スタイルの「**スケジュール**」パラメーターです。このパラメーターを指定すると、その他すべてのタイミング・パラメーターが上書きされます。このパラメーターは主に、実行する複数の週日を指定するために使用されます。例えば、月曜日から金曜日の 03:45 にコネクターが実行されるように指定するには、スケジュール・パラメーターを「* * 2,3,4,5,6 3 45」と設定します。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

コメント
このパラメーターは、ユーザーのコメントをすべて保持できます。コネクター操作中は無効となります。

Tpae IF 変更検出コネクター

以下に示す情報とリンクを使用して、Tpae IF 変更検出コネクターについて知ることができます。

Tpae IF 変更検出コネクターを使用して、共通基盤 (Tpae) 統合フレームワークからの変更通知を受け取ります。

Tpae IF 変更検出コネクターは、Maximo ベースのシステムから、HTTP 要求用の構成可能な TCP ポートで変更通知を受け取ります。Maximo サーバーは、変更通知を HTTP POST 要求として送信するように構成されています。

Tpae IF 変更検出コネクターは、階層項目を返し、サーバー・モードのみをサポートします。指定されたポートにクライアントが接続すると、AssemblyLine がイテレーター・モードで動作して、変更通知を送信中の複数のクライアントをコネクターで同時に処理できるようにします。

Tpae IF 変更検出コネクターのアーキテクチャー

ルート MBO は、子 MBO と共に、要求の HTTP 本文に送信されます。以下に示す例を使用して、Tpae IF 変更検出コネクターのアーキテクチャーを理解することができます。

事前定義されたオブジェクト構造 MXPERSON

```
 の変更通知の例を以下に示します。
<PublishMXPERSON xmlns="http://www.ibm.com/maximo" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" creationDateTime=
"2010-08-26T10:50:36+03:00"
 transLanguage="EN" baseLanguage="EN" messageID="1282809036703888707
"maximoVersion="7 1 20090627-0754 V7115-149" event="1">
 <MXPERSONSet>
  <PERSON action="Replace">
   <DISPLAYNAME changed="1">Fred Rogers</DISPLAYNAME>
   <FIRSTNAME changed="1">Fred2</FIRSTNAME>
   <ADDRESSLINE1>179 Woodtree Lane</ADDRESSLINE1>
   <CITY>Arlington</CITY>
   ...
   <PHONE>
    <ISPRIMARY>1</ISPRIMARY>
    <PHONEID>145833</PHONEID>
    <PHONENUM>(617) 643-1933</PHONENUM>
    <TYPE>WORK</TYPE>
   </PHONE>
   <EMAIL>
    <EMAILADDRESS>fred.rogers@warpspeed.net</EMAILADDRESS>
    <EMAILID>146115</EMAILID>
    <ISPRIMARY>1</ISPRIMARY>
    <TYPE>WORK</TYPE>
   </EMAIL>
  </PERSON>
 </MXPERSONSet>
</PublishMXPERSON>
```

次に、XML パーサーを使用してルート MBO の XML 表現が解析され、ルート MBO だけが取得されます。例えば、PublishMXPERSON エレメントは、XSD スキーマ定義で示しているように、MXPERSONSet エレメントと PERSON エレメントを持つことができます。

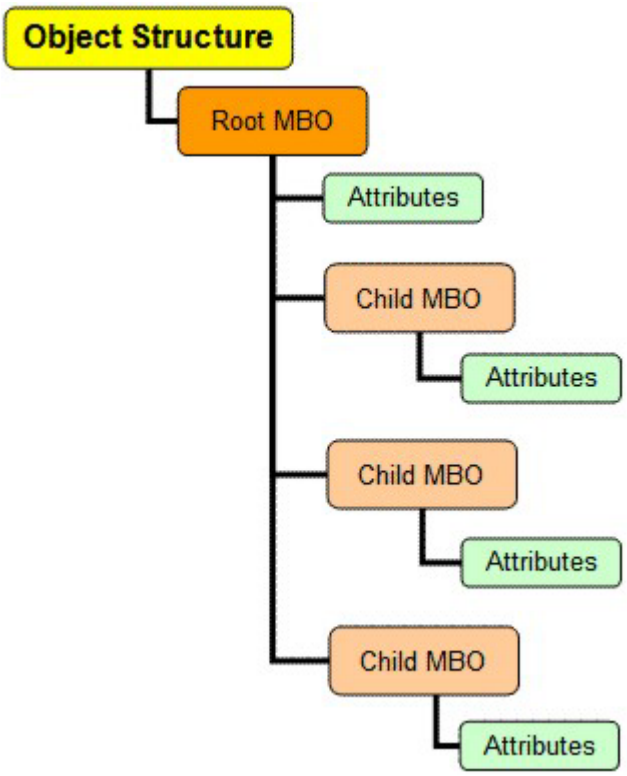


図 5. Maximo MBO

結果の項目は、以下の図のようになります。

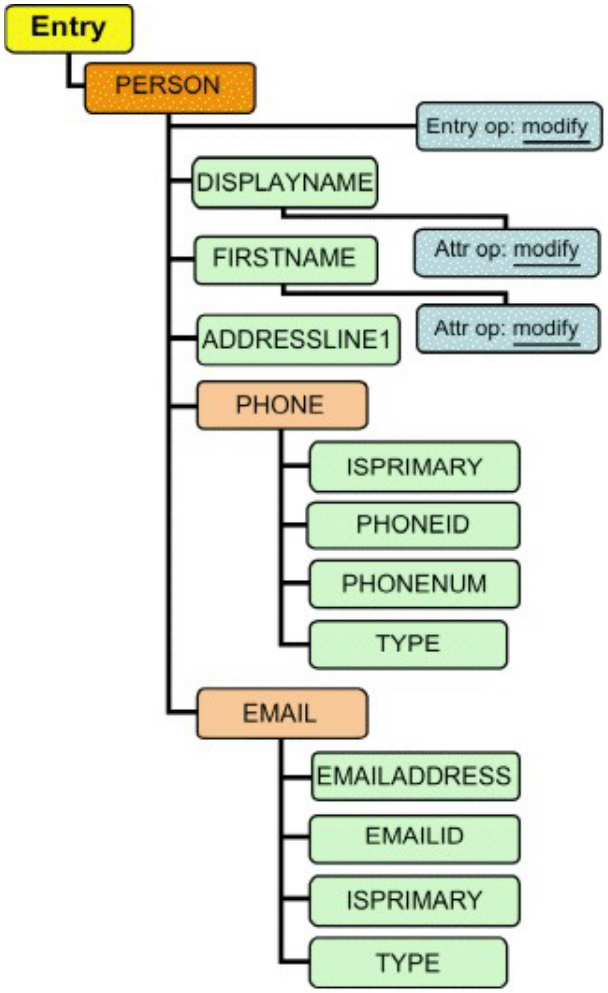


図 6. Maximo MBO から解析される項目

返された項目には、上の図に示すように、すべての子 MBO (E メール・アドレスや電話番号など) が含まれています。

注: 変更通知が送信されるのは、ルート MBO の属性と構造に対する変更の場合のみです。変更内容には、次のようなものが考えられます。

- ルート MBO の追加属性、変更属性、または削除属性
- 子 MBO の追加属性または削除属性

子 MBO の属性に対する変更は、変更通知のトリガーにはならず、作業項目で指定された属性操作によるタグ付けもされません。

デルタ・タグ

Tpae IF 変更検出コネクターは、項目レベルと属性レベルのデルタ・タグを提供します。このコネクターは、受信した項目にタグ付けをし、変更通知を解析します。属性レベルのデルタ・タグは、ルート MBO の変更された属性と追加された属性に対してのみ使用できます。

以下は、前のセクションで示した項目例のデルタ・タグの出力例です。

```
{
 "#type": "modify",
 "#count": 11,
 "PERSON": {
  "#type": "replace",
  "#count": 0,
  "@xmlns": "http://www.ibm.com/maximo",
  "@action": "Replace",
  "FIRSTNAME": {
   "#type": "modify",
   "#count": 1,
   "@changed": "1",
   "#": "Fred"
   },
  "DISPLAYNAME": {
   "#type": "modify",
   "#count": 1,
   "@changed": "1",
   "#": "Fred Rogers"
  },
  "ADDRESSLINE1": [
   "#type": "replace",
   "#count": 1,
   "#": "179 Woodtree Lane"
  ],
  "PHONE": {
   "#type": "replace",
   "#count": 0,
   "ISPRIMARY": [
    "#type": "replace",
    "#count": 1,
    "#": "1"
   ],
   "PHONEID": [
    "#type": "replace",
    "#count": 1,
    "#": "145833"
   ],
   "PHONENUM": [
    "#type": "replace",
    "#count": 1,
    "#": "(617) 643-1933"
   ],
   "TYPE": [
    "#type": "replace",
    "#count": 1,
    "#": "WORK"
   ]
  },
  "EMAIL": {
   "#type": "replace",
   "#count": 0,
   "EMAILADDRESS": [
    "#type": "replace",
    "#count": 1,
    "#": "fred.rogers@warpspeed.net"
   ],
   "EMAILID": [
    "#type": "replace",
    "#count": 1,
    "#": "146115"
   ],
   "ISPRIMARY": [
    "#type": "replace",
    "#count": 1,
    "#": "1"
   ],
   "TYPE": [
    "#type": "replace",
    "#count": 1,
    "#": "WORK"
   ]
  }
 }
}
```

Maximo の操作名を保持する action プロパティーを使用して、項目の操作が決まります。アクションは、追加、削除、置換、変更のいずれかです。置換アクションと変更アクションは、項目の modify 操作として解釈されます。

Maximo サーバーの構成

ここに記載されている情報を使用することで、Maximo サーバーを構成できるようになります。

HTTP エンドポイントの作成

ここに記載されている手順を使用することで、HTTP エンドポイントを作成できるようになります。

1. Maximo に管理者としてログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**エンドポイント**」を選択して、エンドポイント・アプリケーションを開きます。
3. 「**アクションの選択**」メニューから「**ハンドラーの追加/変更**」を選択するか、「**新規エンドポイント**」をクリックします。
4. 新規エンドポイントに有効な名前を指定します。
5. HTTP ハンドラーを選択し、以下のパラメーターの値を指定します。
   a. USERNAME – リモート・コンピューターに接続するユーザー名
   b. PASSWORD – 上記のユーザー名に関連付けるパスワード
   c. CONNECTIONTIMEOUT – 接続時のタイムアウト
   d. READTIMEOUT – 読み取り時のタイムアウト
   e. URL – リモート・システム上のファイルの URL (例: http://9.156.6.14/test.xml)
   f. HTTPMETHOD – POST 要求を選択
   g. HTTPEXIT – カスタム Java クラスの名前

エンドポイントとハンドラーについて詳しくは、「*Maximo Asset Management 7.1 統合ガイド*」の『エンドポイントおよびハンドラー』のセクションを参照するか、ご使用の Maximo ベースの製品の資料を参照してください。

HTTP エンドポイントの割り当て

以下に示す手順の説明に従って、作成済みの HTTP ハンドラーを外部システム、またはパブリッシュ・チャネルに割り当てる必要があります。

1. Maximo に管理者としてログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**外部システム**」を選択して、外部システム・アプリケーションを開きます。
3. 使用可能な外部システムを選択します。
4. 「**エンドポイント**」フィールドに HTTP エンドポイント名を指定します。

   注: すべてのパブリッシュ・チャネルでこのエンドポイントが使用されます。
5. オプション: 「**パブリッシュ・チャネル**」タブに移動し、指定されたパブリッシュ・チャネルのエンドポイントを入力します。

注: オブジェクト構造の変更通知を受け取るには、すべてのパブリッシュ・チャネルを使用可能にする必要があります。

イベント・リスナーの有効化

イベント・リスナーは、オブジェクト構造のルート MBO の変更を listen します。変更を検出すると、アウトバウンド・メッセージが作成され、アウトバウンド・キューに追加されます。選択したパブリッシュ・チャネルに対してイベント・リスナーを有効にするには、ここで提供された手順を実行します。

1. Maximo に管理者としてログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」→「**パブリッシュ・チャネル**」を選択して、パブリッシュ・チャネル・アプリケーションを開きます。
3. 使用するオブジェクト構造のパブリッシュ・チャネルを選択します (例えば、MXPERSON オブジェクト構造の場合は MXPERSONInterface)。
4. 「**リスナーを有効にする**」チェック・ボックスが選択されていない場合は、「**アクションの選択**」メニューから「**イベント・リスナーを有効にする**」を選択します。

特定のオブジェクト構造に対するパブリッシュ・チャネルの名前を探すには、以下の手順を実行します。

1. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」→「**パブリッシュ・チャネル**」を選択して、パブリッシュ・チャネル・アプリケーションを開きます。
2. 「**拡張検索**」ボタンをクリックして、オブジェクト構造の名前を指定します。

   指定したオブジェクト構造のパブリッシュ・チャネルが存在しない場合は、メッセージが表示されます。

クーロン・タスクの構成

各パブリッシュ・チャネルのハンドラーは、指定された間隔で起動するように構成されたクーロン・タスクを使用して開始できます。

変更通知メッセージは、sqout キューをデフォルトで使用します。デフォルト設定の場合、JMSQSEQCONSUMER クーロン・タスクは、アウトバウンド・キュー (sqout) と順次インバウンド・キュー (sqin) に対してポーリングを行います。キュー内に未処理のインバウンド・メッセージとアウトバウンド・メッセージが残らないようにするため、クーロン・タスクの該当するインスタンス (SEQQIN と SEQQOUT) をアクティブにしてください。必要に応じて、クーロン・タスクの実行間隔を変更することができます。

クーロン・タスクを構成するには、以下の手順を実行します。

1. Maximo に管理者としてログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**システム構成**」-> 「**プラットフォームの構成**」->「**クーロン・タスクのセットアップ**」 を選択して、クーロン・タスク・セットアップ・アプリケーションを開きます。
3. JMSQSEQCONSUMER クーロン・タスクを選択します。
4. アウトバウンド・メッセージの処理に使用する SEQQOUT インスタンスが使用可能になっていることを確認します。
5. 必要に応じて、クーロン・タスクの実行間隔を変更します。

デフォルトは 30 秒です。この場合、クーロン・タスクは 30 秒ごとに出力キューを検査して、未処理のメッセージがあるかどうかを調べます。未処理のメッセージは、パブリッシュ・チャネル用に指定されたハンドラーに送られます。

注: この手順でのキュー名とクーロン・タスク・インスタンスは、すべて、Maximo Asset Management 7.1 のデフォルトの名前です。他のシステムでは、異なるキューとクーロン・タスクを使用するように構成されている場合があります。この手順で説明しているオブジェクトが見つからない場合は、システム管理者に問い合わせてください。

構成

ここに記載されているパラメーターを使用することで、Tpae IF 変更検出コネクターを構成できるようになります。

TCP ポート
このパラメーターを使用して、クライアントが HTTP 要求を listen するためのポート番号を指定します。

接続バックログ
このパラメーターを使用して、着信接続要求用のキューの最大長を指定します。このキューがいっぱいになると、それ以後の接続要求は拒否されます。

HTTP 基本認証 (BA)
このパラメーターを使用して、認証処理に HTTP 基本認証メカニズムが必要かどうかを指定します。

認証レルム
このパラメーターを使用して、HTTP 基本認証の要求時にクライアントに送信される認証レルムを指定します。

SSL の使用
このパラメーターを使用して、クライアント接続の受け入れに SSL を使用するかどうかを指定します。

クライアント認証を必要とする
このパラメーターを使用して、SSL 接続でクライアント認証が必要かどうかを指定します。

JDBC URL
このパラメーターを使用して、データ・ソースの JDBC URL を指定します。

JDBC ドライバー
このパラメーターを使用して、接続に必要な JDBC ドライバー・クラスを指定します。

ユーザー名
このパラメーターを使用して、JDBC システムに接続するための有効なユーザー名を指定します。

パスワード
このパラメーターを使用して、JDBC システムに接続するためのユーザー ID に関連するパスワードを指定します。

スキーマ
このパラメーターを使用して、データ・ソースのスキーマを指定します。

外部システム
このパラメーターを使用して、変更通知を HTTP POST 要求として送信する外部システムのコンマ区切りリストを指定します。

このパラメーターを指定した場合、外部システムからの停止したメッセージのみが検出され、必要に応じて処理されます。このパラメーターが空の場合、すべての外部システムからのメッセージが検出され、必要に応じて処理されます。

エラー発生時のアクション
このパラメーターを使用して、指定された外部システムから送信されたすべての保留メッセージに対して実行するアクションを指定します。指定できる値は、以下のとおりです。

- なし - 停止したメッセージが検出された場合、警告メッセージだけを表示します。
- 再試行 - 指定された外部システムからの停止したメッセージの状況を「RETRY」に変更し、Maximo に再処理を指示します。
- 削除 - 指定された外部システムから、停止したメッセージを削除します。

注: Maximo のメッセージ再処理アプリケーションを使用して、メッセージの再処理と削除を行うことができます。

エラー・チェック間隔
このパラメーターを使用して、最後に受信した変更通知と最初のエラー・チェックとの間の時間間隔を秒数で指定します。この間隔により、エラー・チェックの間隔も指定されます。この間隔を 0 に設定すると、エラー・チェックは実行されません。

詳細ログ
このパラメーターを使用して、詳細なログ・メッセージを生成します。

コメント
このパラメーターを使用して、コメントを追加します。データの構文解析時には、このコメントは無視されます。

例

ここに記載されている例を使用することで、Tpae IF 変更検出コネクターを理解できるようになります。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/TpaeIFCDConnector ディレクトリーにあります。

関連情報

641 ページの『第 6 章 資産統合スイート』,
393 ページの『Tpae IF コネクター』,
319 ページの『シンプル Tpae IF コネクター』.

Tpae IF コネクター

以下に示す情報とリンクを使用して、Tpae IF コネクターを構成することができます。

共通基盤 (Tpae) (「基本サービス」とも呼ばれる) はコア Java クラスのコレクションであり、Java アプリケーション作成の基盤として使用されます。統合フレームワークは Tpae の機能の 1 つで、標準統合オブジェクト (オブジェクト構造およびインターフェース) と、アウトバウンド/インバウンド・オブジェクトが含まれています。Tpae IF コネクターは、IBM Security Directory Integrator を Tpae 統合フレームワークに接続して情報を交換します。

Tpae IF コネクターは、統合フレームワークとの間で読み取りと書き込みを行います。Maximo ビジネス・オブジェクト (MBO) がサポートされ、統合オブジェクトを使用して処理されます。このコネクターは、インポートまたはエクスポートされるオブジェクトを検証するために、MBO 層を使用します。

Tpae IF コネクターは階層項目を処理することができ、319 ページの『シンプル Tpae IF コネクター』に基づいて動作します。Tpae IF コネクターは、イテレーター、AddOnly、更新、ルックアップ、削除などのさまざまな AssemblyLine モードで使用できます。

コネクターの使用

Tpae IF コネクターは、シンプル Tpae IF コネクターに関連付けられ、階層項目を処理します。以下に示す表から、この 2 つの Tpae コネクターの相違点を知ることができます。

このコネクターは、オブジェクト構造サービスとエンタープライズ・サービスによる統合をサポートします。このコネクターは、オブジェクト構造全体の読み取りまたは書き込みを行います。MBO を指定する必要はありません。

以下の表に、シンプル Tpae IF コネクターと Tpae IF コネクターの比較を示します。

表 39. Tpae コネクターの相違点

基準	シンプル Tpae IF コネクター	Tpae IF コネクター
データ・フォーマット	フラット項目	階層項目
リンク基準	一致演算子の `equals` と `not equals` だけをサポートします。	すべての一致演算子をサポートします。 階層型のリンク基準名をサポートします。MBO の上位 2 レベルの属性のみ指定できます。
スキーマ	選択された MBO のスキーマを表示します。	選択されたオブジェクト構造の階層スキーマを表示します。*
サービス	作成、更新、削除、および照会の各操作について、オブジェクト構造サービスとエンタープライズ・サービスをサポートします。	同期操作と照会操作について、オブジェクト構造サービスとエンタープライズ・サービスをサポートします。 **注:** 同期操作を使用するサービスには、作成、追加、および削除の各操作がカプセル化されています。

イテレーター・モード

イテレーター・モードの場合、Tpae IF コネクターは、照会 XML 要求を IF サーバーに送信し、照会 XML 応答を受信します。以下に示す例を使用して、この場合の XML 応答について知ることができます。

例えば、Maximo は、事前定義された MXASSET オブジェクト構造に対する照会操作の結果として、以下の XML 応答を返します。

```
<ASSET>
 <ASSETNUM>7111</ASSETNUM>
 <BUDGETCOST>1000.0</BUDGETCOST>
 <ASSETSPEC>
  <ASSETATTRID>RAMSIZE</ASSETATTRID>
  <MEASUREUNITID>MBYTE</MEASUREUNITID>
  <NUMVALUE>512.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
  <ASSETATTRID>DISKSIZE</ASSETATTRID>
  <MEASUREUNITID>GBYTE</MEASUREUNITID>
  <NUMVALUE>100.0</NUMVALUE>
  -
 </ASSETSPEC>
 <ASSETSPEC>
  <ASSETATTRID>PROSPEED</ASSETATTRID>
  <MEASUREUNITID>GHZ</MEASUREUNITID>
  <NUMVALUE>1.5</NUMVALUE>
  -
 </ASSETSPEC>
 -
</ASSET>
```

上記の XML 表現は、以下のように IBM Security Directory Integrator 項目オブジェクトに変換されます。

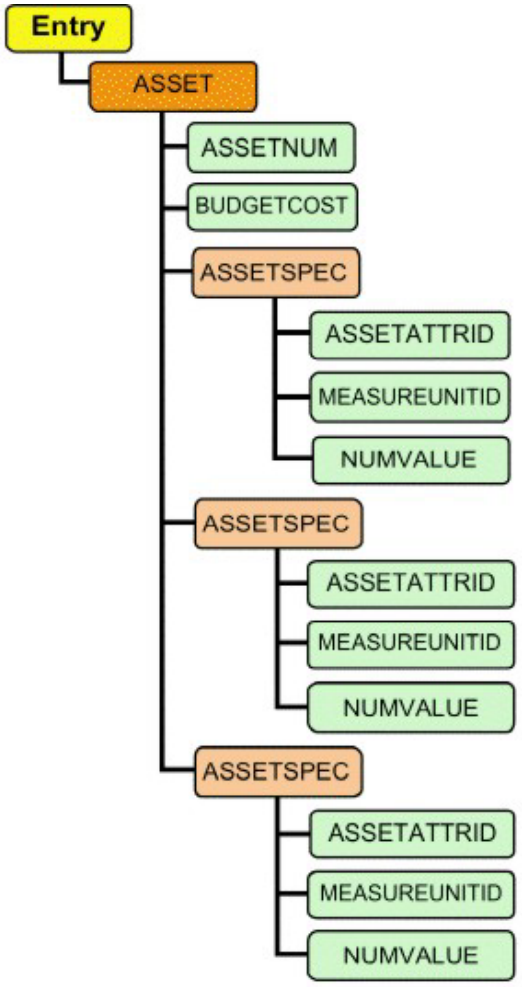


図 7. Maximo からの階層項目オブジェクト

返された項目には、すべての子 MBO (ツリー型の資産仕様) が含まれています。項目の `toString()` メソッドを使用して、この階層のストリング表現全体を表示することができます。

イテレーター・モードでの照会基準

Tpae IF コネクターは、イテレーター・モードの場合のみ、「**照会基準**」パラメーターを使用して反復の結果セットをフィルタリングします。

注: オブジェクト構造の MBO の上位 2 レベルから照会値を選択します。例えば、`ASSET`、`ASSETSPEC`、`ASSETMETER` などの MBO の属性を選択します。

演算子属性

演算子属性は、以下の形式で、フィールドの値を他の 1 つ以上の値と比較します。

`operator = oper`。ここで、`oper` は、以下のいずれかの値です。

表 40. 演算子の値

Oper	説明
=	等しい

表 40. 演算子の値 (続き)

Oper	説明
!=	等しくない
<	より小
<=	より小または等しい
>	より大
>=	より大または等しい
SW	先頭文字
EW	終了文字

「より小」および「より大」の属性は数値および日付フィールドでのみ使用します。

例:

IT 以外のすべてのタイプの資産を検索するには、以下の形式の照会を使用します。

```
<ASSET>
  <ASSETTYPE operator="!=">IT</ASSETTYPE>
</ASSET>
```

フィールド選択

フィールド・ベースの照会では、フィールドの値が、XML フィールドの指定された値と比較されます。値に大文字と小文字の区別はありません。

例:

以下の照会は、VENDOR が ATI で STATUS が OPERATING の資産を検索します。

```
<ASSET>
  <VENDOR operator="=">ATI</VENDOR>
  <STATUS operator="=">OPERATING</STATUS>
</ASSET>
```

以下の照会は、VENDOR に ATI が含まれ、STATUS に OPER が含まれる資産を検索します。

```
<ASSET>
  <VENDOR>ATI</VENDOR>
  <STATUS>OPERATING</STATUS>
</ASSET>
```

以下の照会では、指定されたタグが付いていない資産が検索されます。最初の照会では、演算子属性を使用し、2 番目の照会では、比較に完全一致突き合わせ値を使用しています。

```
<ASSET>
  <ASSETTAG operator="NULL"></ASSETTAG>
</ASSET>

<ASSET>
  <ASSETTAG>NULL</ASSETTAG>
</ASSET>
```

以下の照会は、テキスト 711 で開始する資産番号の資産を検索します。

```
<ASSET>
  <ASSETNUM operator="SW">711</ASSETNUM>
</ASSET>
```

以下の照会は、SQL IN 節に相当する集合を使用して、状況が NOT READY または OPERATING の資産を検索します。

```
<ASSET>
  <STATUS>NOT READY, OPERATING</STATUS>
</ASSET>
```

範囲選択

照会では、特定の範囲内の値を持つレコードを検索することができます。フォーマットは、選択基準が無制限であるか、範囲の上限および下限があるかによって異なります。

例:

以下の照会では、BUDGETCOST が 1000 ドルを超える資産が検索されます。

```
<ASSET>
  <BUDGETCOST operator="&gt;">1000</BUDGETCOST>
</ASSET>
```

以下の照会では、BUDGETCOST が 1000 ドルより大きく 20000 ドルより小さい資産が検索されます。

```
<ASSET>
  <BUDGETCOST operator="&gt;">1000</BUDGETCOST>
  <BUDGETCOST operator="&lt;">20000</BUDGETCOST>
</ASSET>
```

注: 照会には、同じ属性に対して最大で 2 つの参照を含めることができます。

AddOnly モード、更新モード、削除モード

Tpae IF コネクターは、オブジェクト構造サービスとエンタープライズ・サービスの両方を使用して MBO を変更します。Tpae IF コネクターを構成するには、構成パラメーターを 1 つだけ使用できます。

オブジェクト構造サービスまたはエンタープライズ・サービスには、作成、更新、削除の各操作をカプセル化する機能があります。更新モードの場合、コネクターは属性の操作を検査して、Tpae に送信される結果の XML ペイロードに適切なアクションを設定します。追加、変更、または削除操作に属性がタグ付けされていない場合、アクションは設定されません。

以下の例は、デルタ・タグが付けられた変更対象項目と、Tpae IF サーバーに送信される対応する要求を示しています。

```
{
  "#type": "generic",
  "#count": 7,
  "PERSON": {
    "#type": "replace",
    "#count": 0,
    "PERSONID": [
"#type": "replace",
"#count": 1,
"#replace": "JOHN"
    ],
    "FIRSTNAME": [
"#type": "replace",
"#count": 1,
"#replace": "John"
    ],
    "LASTNAME": [
"#type": "replace",
"#count": 1,
"#replace": "Jones"
    ],
    "PHONE": {
"#type": "add",
"#count": 0,
"PHONENUM": [
    "#type": "replace",
    "#count": 1,
    "#replace": "0888776455"
 ],
"TYPE": [
    "#type": "replace",
    "#count": 1,
    "#replace": "WORK"
],
"ISPRIMARY": [
    "#type": "replace",
    "#count": 1,
    "#replace": "1"
]
},
    "PHONE": {
"#type": "delete",
"#count": 0,
"PHONENUM": [
    "#type": "replace",
    "#count": 1,
    "#replace": "555244458"
]
    },
    "EMAIL": {
"#type": "replace",
"#count": 0,
"EMAILADDRES": [
    "#type": "replace",
    "#count": 1,
    "#replace": "jjones@mail.com"
 ]
}
    }
```

```
<?xml version='1.0' encoding='UTF-8'?>
<SyncMXPERSON
    xmlns="http://www.ibm.com/maximo"
    creationDateTime="2010-11-05T16:44:20+02:00"
    transLanguage="EN"
    messageID="1288968260482"
    maximoVersion="7 1 Harrier 072 7100-001">
    <MXPERSONSet>
 <PERSON action="Change">
    <PERSONID>JOHN</PERSONID>
    <FIRSTNAME>John</FIRSTNAME>
    <LASTNAME>Jones</LASTNAME>
    <PHONE action="Add">
      <PHONENUM>0888776455</PHONENUM>
      <TYPE>WORK</TYPE>
      <ISPRIMARY>1</ISPRIMARY>
    </PHONE>
    <PHONE action="Delete">
      <PHONENUM>555244458</PHONENUM>
    </PHONE>
    <EMAIL>
      <EMAILADDRESS>
   jjones@mail.com
      </EMAILADDRESS>
      <ISPRIMARY>1</ISPRIMARY>
    </EMAIL>
 </PERSON>
    </MXPERSONSet>
</SyncMXPERSON>
```

AddOnly モードと更新モードでは、追加または更新対象の MBO のすべての固有属性に値を指定してください。属性では、空のストリングも値としてサポートされま

す。削除モードでは、ルート MBO のすべての固有属性の値を指定してください。いずれかの固有キーが指定されていない場合、コネクターが例外をスローして操作が失敗します。

更新モードでは、「ルックアップのスキップ」機能が Tpae IF コネクターでサポートされます。 「ルックアップのスキップ」機能により、オブジェクト構造を更新または削除する前に、オブジェクト構造の照会をスキップすることができます。照会操作のたびに、ネットワーク経由で HTTP 要求が送信されます。AssemblyLine を更新する場合、または複数の項目を削除する場合は、「ルックアップのスキップ」を使用可能にするとパフォーマンスが向上します。

注: 更新モードでは、「ルックアップのスキップ」が無効になっていて、出力マップで MBO の固有キーが変更されている場合、その値は元の値で上書きされます。Maximo から値が読み取られ、MBO が変更されます。その際、固有属性は変更できないことを知らせるデバッグ・メッセージが表示されます。 「ルックアップのスキップ」が無効になっている場合、MBO は新規と見なされて追加されます。

ルックアップ・モード

Maximo 内の特定のレコードを検索するには、そのレコードを一意的に識別する属性を使用したリンク基準を指定することができます。

例

以下の属性は、Maximo 内の資産を一意的に識別します。

属性名	値
ASSET.ASSETNUM	1001
ASSET.SITEID	BEDFORD

以下の属性は、Maximo 内の資産計測値を一意的に識別します。

属性名	値
ASSET.ASSETNUM	1001
ASSET.SITEID	BEDFORD
ASSET.ASSETMETER@METERNAME	RUNHOURS

項目を検索する場合、さまざまな一致演算子を使用することができます。「**複数項目時**」フックが指定されていないと、エラーがスローされます。

Tpae IF コネクターは、タイプ AND のリンク基準のみをサポートしています。サポートされる一致演算子を以下の表にリストします。

表 41. 一致演算子

一致演算子	詳細
equals	この演算子をストリング・タイプの属性と一緒に使用すると、辞書の語順に従ってストリングが比較されます。 例えば、a は c より前に来るため、apple は carrot より小さいとみなされます。
less than	
less or equals	
greater than	
greater or equals	

表 41. 一致演算子 (続き)

一致演算子	詳細
contains	この演算子を使用できるのは、ストリング・タイプの属性だけ です。
starts with	
ends with	
not equals	

注: 1 つの属性に 3 つ以上の基準を指定すると、例外がスローされます。
リンク基準では、特定のオブジェクト構造階層の上位 2 レベルの MBO から属性が選択されます。そのため、以下のような階層リンク基準を指定できます。

ASSET.SITEID	equals	BEDFORD
ASSET.ASSETUSERCUST.PERSONID	equals	MAXADMIN
ASSET.ASSETMETER.METERNAME	contains	HOURS

上の表の ASSETUSERCUST と ASSETMETER は、両方とも ASSET MBO の子 MBO の第 1 レベルです。

エラー処理

Tpae IF コネクターは、発生したすべての例外を、通常のサーバー・フックを使用して処理します。障害を処理できない場合は、対応する AssemblyLine エラー・フックが開始されます。このコネクターに特有の例外を以下に示します。

- MxConnectorRuntimeException
  - MxConnConfigException
- MxConnectorException
  - MxConnIOException
    - MxConnHttpException
    - MxConnTimeoutException
  - MxConnSchemaException
    - MxConnExcedentSizeException
    - MxConnTypeConvertionException
  - MxConnXmlParsingException

Tpae IF コネクターで AssemblyLine に障害が発生した場合、以下のようにしてエラーに関する追加情報を取得できます。

1. 「エラー発生時のデフォルト」フックに以下のコードを追加します。コネクターには mxConn という名前を付けます。

```
task.logmsg("ERROR", "An exception occurred.");
mxConn.connector. extractMaximoException(error);
task.dumpEntry(error);
```

2. 例外が発生すると、次のメッセージが表示されます。

```
19:31:44  CTGDIS003I *** 項目のダンプを開始します
19:31:44   Operation: generic
19:31:44   Entry attributes:
19:31:44   exception (replace): 'com.ibm.di.connector.
           maximo.exception.MxConnHttpException:
           response: 404 - Not Found'
19:31:44   targetUrl (replace): 'http://9.156.6.14/meaweb/schema
           /service/MXPersonService.xsd'
```

```
19:31:44   class (replace): 'com.ibm.di.connector.maximo.exception
          .MxConnHttpException'
19:31:44  operation (replace): 'update'
19:31:44  status (replace): 'fail'
19:31:44  connectorname (replace): 'AddPerson'
19:31:44  body (replace): 'Error 404: BMXAA1513E - Cannot obtain
          resource /meaweb/schema/service/MXPersonService.xsd.'
19:31:44  responseCode (replace): '404.0'
19:31:44  responseMessage (replace): 'Not Found'
19:31:44  message (replace): 'The HTTP server did not returned "HTTP OK".'
19:31:44  CTGDIS004I *** 項目のダンプを完了しました
```

注: task.dumpEntry(error) は、エラーに関する情報を出力します。

外部システム構成

以下に示す手順を使用して、外部システムの構成について知ることができます。

XML スキーマ定義の生成

コネクターを初めて使用する場合は、以下の手順を実行します。

1. システム構成タスクを実行する管理者として Maximo にログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**オブジェクト構造**」を選択して、オブジェクト構造アプリケーションを開きます。
3. 使用するオブジェクト構造ごとに、以下の手順を繰り返します。
   a. 「**リスト**」タブで、オブジェクト構造の名前 (MXASSET など) を検索します。

      検索するには、「フィルター」を開き、「**オブジェクト構造**」列の「**フィルター**」フィールドにオブジェクト構造の名前または名前の一部を入力します。ENTER を押します。
   b. オブジェクト構造名をクリックして、オブジェクト構造のレコードを開きます。
   c. 「**アクションの選択**」メニューから、「**スキーマの生成/XML の表示**」を選択します。
   d. 「**OK**」をクリックします。「XML の表示」ダイアログ・ボックスが表示されます。
   e. 「**一覧**」タブに戻るには、「**OK**」をクリックします。

エンタープライズ・サービスの作成

Tpae IF コネクターは、オブジェクト構造サービスとエンタープライズ・サービスの両方をサポートしています。「**外部システム**」パラメーターを指定する場合は、以下のパラメーターにエンタープライズ・サービスの名前を指定する必要があります。

- **QUERY エンタープライズ・サービス** - このパラメーターには、指定されたオブジェクト構造にバインドされたサービスを指定する必要があります。
- **SYNC エンタープライズ・サービス** - このパラメーターには、指定されたオブジェクト構造にバインドされたサービスを指定する必要があります。
- **MAXOBJECT/MAXATTRIBUTE QUERY エンタープライズ・サービス** - このパラメーターを使用して、オブジェクト構造のメタデータ情報を取得します。このパラメーターには、MAXOBJECT/MAXATTRIBUTE MBO が含まれているオブジェクト構造にバインドされたサービスを指定する必要があります。

Tpae IF コネクターは、オブジェクト構造サービスでなくエンタープライズ・サービスでこれらのパラメーターを使用して、オブジェクト構造の照会または変更を行います。

指定された外部システムを使用してオブジェクト構造用のエンタープライズ・サービスを作成するには、以下の手順を実行します。

1. システム構成タスクを実行する管理者として Maximo にログオンします。
2. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**オブジェクト構造**」を選択して、オブジェクト構造アプリケーションを開きます。
3. 「**新規エンタープライズ・サービス**」をクリックして、エンタープライズ・サービスを作成します。
4. 以下のパラメーターの詳細を指定します。
   a. **エンタープライズ・サービス** – エンタープライズ・サービスの固有の名前。
   b. **アダプター** – エンタープライズ・サービスで使用されるアダプターの名前。デフォルトの名前は Maximo です。
   c. **オブジェクト構造** – エンタープライズ・サービスに関連付けるオブジェクト構造の名前。
   d. **操作** – 操作のタイプ。デフォルトの操作は「同期」です。「同期」オプションには、作成、削除、更新の各機能が含まれます。Tpae IF コネクターの場合、照会操作または同期操作についてのみ、エンタープライズ・サービスを作成することができます。
5. エンタープライズ・サービスを保存します。
6. ナビゲーション・ツールバーの「**移動**」メニューから「**統合**」->「**外部**」を選択して、外部システム・アプリケーションを開きます。
7. 外部システムとその「**エンタープライズ・サービス**」タブを選択します。
8. 「**新規行**」をクリックして、新しく作成されたエンタープライズ・サービスの名前を入力します。

構成

ここに記載されているパラメーターを使用することで、Tpae IF コネクターを構成できるようになります。

基本 URL

このパラメーターを使用して、Tpae 製品にメッセージを送信するための URL のリストを指定します。Tpae が IBM Security Directory Integrator と同じシステム上にある場合は、`http://localhost` を使用します。それ以外の場合は、Tpae サーバーの IP アドレスを使用します。Tpae アプリケーションにログインする場合と同じポートを使用してください。例えば、ログイン URL が `http://192.168.80.128:9080/maximo/webclient/login/login.jsp` の場合、ポート 9080 を使用します。リストでは、URL 間の分離文字としてスペースを使用します。

注: 高可用性を実現するには、単一の URL でなく URL のリストを使用する必要があります。リストの先頭のサーバーが例外をスローすると、2 番目

の URL が使用され、有効なサーバーが見つかるまで順に同様の処理が行われます。最後の URL も無効だった場合は、例外がスローされて接続が失敗します。

ユーザー ID

このパラメーターを使用して、Tpae アプリケーションにログインするための有効なユーザー ID を指定します。

パスワード

このパラメーターを使用して、Tpae アプリケーションにログインするための有効なパスワードを指定します。

オブジェクト構造

このパラメーターを使用して、統合に使用されるオブジェクト構造の名前を指定します。構成エディターの「**クリア**」ボタンは、「オブジェクト構造」パラメーターに関連付けられています。

クリア コネクターは、MBO リストの表示にかかる時間を最小限にするために、使用されたすべてのオブジェクト構造のスキーマを内部的に保存します。「**クリア**」ボタンをクリックすると、保存されたすべてのスキーマを削除することができます。オブジェクト構造のスキーマが変更された場合 (XSD の生成) や、このスキーマのローカル表現の更新が必要な場合は、この操作を使用すると便利です。スキーマのキャッシュをクリアした後は、別のオペレーティング・システムに対する「MBO の取得」スクリプトの呼び出しで遅延が生じます。これは、各スキーマをサーバーから再び取得する必要があるためです。

注: 「**クリア**」ボタンをクリックすると、設計時に使用されたスキーマ・キャッシュがクリアされます。構成エディターがサーバー上で AssemblyLine を実行すると、別のスキーマ・キャッシュが作成されます。コネクターで `clearSchemaCache()` メソッドを開始すると、このスキーマを削除することができます。例えば、「**初期化後**」フックに `thisConnector.connector.clearSchemaCache();` というテキストを追加すると、コネクターを使用する前にキャッシュをクリアすることができます。

照会基準

このパラメーターを使用して、反復の結果セットをフィルタリングします。このパラメーターには、イテレーター・モード用の選択基準が含まれています。照会は XML 構文で指定します。単一値または値の範囲に基づいてレコードを選択できます。

注: 照会基準では、オブジェクト構造の上位 2 レベルの MBO から属性が選択されます。
照会基準パラメーターのフォーマットを以下に示します。

```
<MBO>
  <FIELD1 operator="oper"> </FIELD1>
  <FIELD2> </FIELD2>
  ...
</MBO>
```

ここで、

MBO - 検索対象のビジネス・オブジェクト。
FIELD - MBO フィールドの名前。
oper - 検索のための条件演算子。

ページ・サイズ
このパラメーターを使用して、Tpae から取得するレコードの数を制限します。コネクターは、照会基準によって選択されたすべてのレコードを取得するための複数の要求を作成します。

注: ページ・サイズは、オブジェクト構造のルート MBO にのみ適用されます。
例えば、Maximo のデータベースに 1000 個の資産があり、ページ・サイズが 100 と定義されている場合、事前定義された MXASSET オブジェクト構造に対する照会は 10 個の要求によって実行されます。 デフォルト値は 100 です。

フィールド・サイズの検証
このパラメーターを使用して、テキスト・フィールドが最大サイズを超えた場合にエラーをスローします。構成エディターで「**フィールド・サイズの検証**」チェック・ボックスが選択されていない場合は、テキストが切り捨てられます。

XML 文字検証
このパラメーターを使用して、XML コンテンツの構文解析を実行する前に、無効な Unicode 文字を XML コンテンツから削除します。

IF バージョン
このパラメーターを使用して、サーバーと交換される各メッセージに含まれていなければならない IF のバージョンを指定します。構成エディターで、適切なデフォルト値が提供されます。

トランザクション言語
このパラメーターを使用して、複数言語に対応したフィールドの内容値を提供するためのトランザクション言語を指定します。デフォルト値は EN です。言語の頭字語の詳細なリストについては、ISO 639-1 alpha-2 コード (http://www.loc.gov/standards/iso639-2/php/English_list.php) を参照してください。

選択可能な項目は、以下のとおりです。

- DE - ドイツ語
- EN - 英語
- ES - スペイン語
- FR - フランス語
- IT - イタリア語
- KO - 韓国語
- PT - ポルトガル語
- ZH - 中国語

タイムアウト
このパラメーターを使用して、IF サーバーと通信します。接続を確立する前、または使用可能なデータを読み取る前にタイムアウト時間が経過する

と、MxConnTimeoutException がスローされます。タイムアウトの値をゼロ (デフォルト) にすると、無限タイムアウトと見なされます。

外部システム
このパラメーターを使用して、外部システムの名前を指定します。外部システムは、作成、更新、削除、または照会操作のエンタープライズ・サービスを、選択されたモード用にグループ化して公開します。

MAXOBJECT/MAXATTRIBUTE オブジェクト構造
このパラメーターを使用して、MAXOBJECT MBO と MAXATTRIBUTE MBO を公開するオブジェクト構造の名前を指定します。このオブジェクト構造を使用して、属性の最大許容サイズなど、補足的な MBO のメタデータを取得します。デフォルト値は MXOBJECTCFG です。

MAXOBJECT/MAXATTRIBUTE QUERY エンタープライズ・サービス
このパラメーターを使用して、MAXOBJECT オブジェクト構造に対する*照会*操作を実行するエンタープライズ・サービスの名前を指定します。このオブジェクト構造を使用して、属性の最大許容サイズなど、補足的な MBO のメタデータを取得します。

注: このパラメーターは、「**外部システム**」パラメーターを指定した場合のみ有効です。
デフォルト値は MXMaxObjectQuery です。

QUERY エンタープライズ・サービス
このパラメーターを使用して、指定されたオブジェクト構造に対する*照会*操作を実行するエンタープライズ・サービスの名前を指定します。

注: このパラメーターは、「**外部システム**」パラメーターを指定した場合のみ有効です。

SYNC エンタープライズ・サービス
このパラメーターを使用して、指定されたオブジェクト構造に対する*同期*操作を実行するエンタープライズ・サービスの名前を指定します。このサービスを使用して、選択したコネクター・モードで指定されたオブジェクト構造に対する作成、更新、および削除操作を実行します。

注: このパラメーターは、「**外部システム**」パラメーターを指定した場合のみ有効です。

コメント
ここには、ユーザー独自のコメントを入力します。コメントは、このコンポーネントの操作中は考慮されません。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

例

Tpae IF コネクターを理解するには、以下に示す例を使用できます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/TpaeIFConnector ディレクトリーにあります。

関連情報

641 ページの『第 6 章 資産統合スイート』,
319 ページの『シンプル Tpae IF コネクター』,
384 ページの『Tpae IF 変更検出コネクター』.

URL コネクター

URL コネクターはトランスポート・コネクターの 1 つであり、このコネクターが正しく機能するにはパーサーが必要です。このコネクターは、URL で指定されているストリームをオープンします。このコネクターは、AddOnly モードとイテレーター・モードで使用できます。

注: プロキシー・サーバーを強制適用するファイアウォールを通して実行された場合、URL コネクターは機能しません。URL コネクターは、正しいプロキシー・サーバー・セットを持っている必要があります。

原則として、このコネクターは SSL プロトコルを使用したセキュア通信を処理できますが、この SSL サポートをセットアップするには、ドライバー固有の構成手順を実行する必要があります。

構成

ここに記載されているパラメーターを使用することで、URL コネクターを構成できるようになります。

このコネクターは、以下のパラメーターを必要とします。

URL 公開する URL (例えば、http://*host*/file.csv)。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

「パーサー構成」ペインから、ストリームを操作するパーサーを選択します。パーサーを選択するには、右下の「**継承元:**」ボタンをクリックします。

サポートされる URL プロトコル

使用可能なサポートされる URL プロトコルを以下に示します。

- HTTP
- HTTPS

関連情報

119 ページの『ファイル・コネクター』,
380 ページの『TCP コネクター』,
56 ページの『TCP/URL の直接スクリプト記述』.

Web サービス受信側サーバー・コネクター

Web サービス受信側サーバー・コネクターは、IBM Security Directory Integrator Web Services Suite に含まれています。以下に示す情報とリンクを使用して、Web サービス受信側サーバー・コネクターについて知ることができます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

このコネクターは、基本的には HTTP サーバーですが、HTTP 経由での SOAP 要求にサービスを提供する目的に特化されています。これはサーバー・モードのみで稼働します。

AssemblyLine では、操作項目 (op 項目) がサポートされています。op 項目の属性 *$operation* には、AssemblyLine により実行された現行操作の名前が含まれています。他の Web サービス操作を容易に処理できるようにするため、Web サービス受信側サーバー・コネクターにより op 項目の *$operation* 属性が設定されます。

Web サービス受信側サーバー・コネクターは、AssemblyLine の入出力スキーマに基づく WSDL ファイルの生成をサポートしています。 IBM Security Directory Integrator では AssemblyLine により複数操作がサポートされるため、WSDL を生成すると、複数の操作を持つ Web サービス定義が生成されることがあります。操作の命名に関するルールを以下に示します。

- 6.1 より前のバージョンの IBM Security Directory Integrator 構成ファイルでは、1 つの入力スキーマと 1 つの出力スキーマのみがデフォルト操作スキーマとして参照されていました。6.1 より前のバージョンの IBM Security Directory Integrator 構成を使用する場合は、IBM Security Directory Integrator 6.0 の場合と同様に、1 つの操作のみが生成され、この操作の名前として AssemblyLine 名が設定されます。
- 現行バージョンの IBM Security Directory Integrator の構成では、「Default」という名前の操作が存在する場合、これに対応する WSDL ファイルの操作名には、AssemblyLine の名前が設定されます。
- 現行バージョンの IBM Security Directory Integrator 構成では、「Default」という名前の操作と AssemblyLine 名を持つ操作が存在する場合、WSDL ファイルでは両方の操作の名前が維持されます。
- 上記に該当しない場合はすべて、WSDL ファイルの操作には AssemblyLine 構成での名前が使用されます。

WSDL ファイルのホスティング

以下に示す情報を使用して、WSDL ファイルのホスティングについて知ることができます。

Web サービス受信側サーバー・コネクターは、*wsdlRequested* というコネクター属性を AssemblyLine に提供します。

HTTP 要求が到着したとき、要求された HTTP リソースの末尾に「?WSDL」がある場合、コネクターは *wsdlRequested* 属性の値を **true** に設定します。それ以外の場合、この属性の値は **false** に設定されます。

この属性の値は、受信した要求が SOAP 要求なのか WSDL ファイルを求める要求なのかを AssemblyLine に通知します。これを利用すると、AssemblyLine で、純粋な SOAP 要求と WSDL ファイルを求める HTTP 要求を区別できます。AssemblyLine では、分岐コンポーネントを使用して、ロジックのうち必要な部分のみを実行できます。つまり、(1) WSDL ファイルを求める要求を受信した場合、AssemblyLine は WSDL ファイルを読み取って、それを Web サービス・クライアントに返送します。(2) SOAP 要求を受信した場合、AssemblyLine はその SOAP 要求を処理します。別の方法として、適切な場所 (スクリプト・コンポーネント内、AssemblyLine の最初のコネクターのフック内など) に `system.skipEntry();` 呼び出しをプログラミングして、それ以降の処理をスキップすることもできます。

SOAP 要求または WSDL ファイルを求める要求に対して必要な応答を提供する処理は、AssemblyLine の役割です。

このコネクターは、次の共通メソッドをインプリメントしています。

```
public String readFile (String aFileName) throws IOException;
```

このメソッドをスクリプト・コンポーネント内の IBM Security Directory Integrator JavaScript で使用すると、ローカル・ファイル・システムにある WSDL ファイルの内容を読み取ることができます。その場合 AssemblyLine は、WSDL の内容を *soapResponse* 属性に戻し、それにより Web サービス・クライアントに戻します (WSDL に対する要求を受信した場合)。

スキーマ

Web サービス受信側サーバー・コネクターについてよく理解するために、以下のスキーマの説明を参照できます。

入力スキーマ

表 42. Web サービス受信側サーバー・コネクターの入力スキーマ

属性	値
host	タイプは String です。要求が送信された先のホスト名が含まれています。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
requestedResource	要求された HTTP リソース。
soapAction	SOAP アクションの HTTP ヘッダー。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
soapRequest	txt/XML または DOMElement フォーマットの SOAP 要求。このパラメーターは「wsdlRequested」が false の場合のみ設定されます。
wsdlRequested	WSDL ファイルが要求される場合、このパラメーターは true、要求されない場合は false です。
http.username	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。値は接続されているクライアントのユーザー名です。
http.password	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。値は接続されているクライアントのパスワードです。

出力スキーマ

表 43. Web サービス受信側サーバー・コネクターの出力スキーマ

属性	値
responseContentType	応答タイプ。デフォルトの応答タイプは「text/xml」です。これは、SOAP メッセージで使用されます。
soapResponse	SOAP 応答メッセージ。wsdlRequested が true の場合、soapResponse は WSDL ファイルの内容に設定されます。
http.credentialsValid	この属性を使用できるのは、HTTP 基本認証が使用可能な場合のみです。クライアントが HTTP 基本認証のユーザー名とパスワードを入力する場合、この属性を true または false に設定する必要があります。これは、コネクターではなく、コネクターを使用している AssemblyLine によって実行されます。値が true の場合、クライアントが正しく認証され、アクセス権限が付与されます。値が false の場合、ユーザーは認証されず、HTTP の「権限を持たない」コネクター応答は戻されます。

構成

ここに記載されているパラメーターを使用することで、Web サービス受信側サーバー・コネクターを構成できるようになります。

パラメーター

TCP ポート
サービスが実行されている (listen している) ポート番号です。

接続バックログ
着信接続指示 (接続要求) の最大キュー長を表します。接続指示の着信時にキューがいっぱいであると、接続は拒否されます。

SOAP メッセージを次のタイプとして入力
AssemblyLine への SOAP 要求メッセージの入力タイプを指定します。このドロップダウン・リストでは、「**ストリング**」または「*DOMElement*」のいずれかを選択できます。

SOAP メッセージを次のタイプとして戻す
AssemblyLine からの SOAP 応答メッセージの出力タイプを指定します。このドロップダウン・リストでは、「**ストリング**」または「*DOMElement*」のいずれかを選択できます。

OP 項目にタグを付ける
このパラメーターをチェックすると (「true」の場合)、現行実行操作が AssemblyLine/WSDL の公開操作のリストにない場合でも、コネクターにより op 項目にタグが付けられます。大/小文字が適切に処理されるかどうかは、IBM Security Directory Integrator ソリューションのインプリメンテーションによって決まります。

SSL の使用
これをチェックすると、サーバーは SSL (https) 接続だけを許可します。SSL のパラメーター (鍵ストアなど) は、IBM Security Directory Integrator のインストール・フォルダーに格納されている `global.properties` ファイル内の Java システム・プロパティーの値として指定されています。

クライアント認証を必要とする

このコネクターでクライアントがクライアント SSL 証明書による認証を実行することが必要であるかどうかを指定します。このパラメーターの値が **true** の場合 (つまりチェックした場合)、クライアントはクライアント SSL 証明書による認証を実行せず、コネクターはクライアント接続を失います。このパラメーターの値が **true** であり、かつクライアントがクライアント SSL 証明書による認証を実行する場合は、コネクターはクライアント要求の処理を続行します。このパラメーターの値が **false** の場合は、クライアントがクライアント SSL 証明書による認証を実行するかどうかに関係なく、コネクターはクライアント要求を処理します。

認証レルム

認証が要求された場合にクライアントに送られる基本レルム。

HTTP 基本認証 (BA) を使用します。

このコネクターでは HTTP 基本認証がサポートされます。アクティブにするには、「HTTP 基本認証 (BA) を使用」チェック・ボックスをチェックします。アクティブにすると、サーバーは信任状が既に送信されているかどうかを検査し、送信されていない場合はクライアントに許可要求を送信します。クライアントから必要な信任状が送信された後に、コネクターによって「http.username」と「http.password」という 2 つの属性が設定されます。この 2 つの属性にはそれぞれ、クライアントのユーザー名とパスワードが含まれています。AssemblyLine がこのユーザー名とパスワードが有効であるかどうかを検査します。クライアントが正常に許可された場合は、作業項目属性「http.credentialsValid」を true に設定する必要があります。クライアントが許可されなかった場合は、作業項目属性「http.credentialsValid」を false に設定する必要があります。クライアントが許可されなかった場合は、サーバーから「Not Authorized」HTTP メッセージが送信されます。

コメント

ここにはユーザー独自のコメントを入力します。

詳細ログ

これをチェックした場合、追加のログ・メッセージが生成されます。

WSDL 出力ファイル名

「**WSDL 生成**」ボタンをクリックしたときに生成される WSDL ファイルの名前です。このパラメーターは、WSDL 生成ユーティリティーのみが使用し、コネクターの実行中は使用しません。

Web サービス・プロバイダー URL

Web サービス・クライアントが Web サービス要求を送信する先のアドレスです。このパラメーターも、WSDL 生成ユーティリティーのみが使用し、コネクターの実行中は使用しません。

「**WSDL 生成**」ボタンをクリックすると、WSDL 生成ユーティリティーが実行されます。

WSDL 生成ユーティリティーは、入力として、生成する WSDL ファイルの名前と、Web サービスのプロバイダーの URL (Web サービスの場所) を使用します。このユーティリティーは、コネクターが組み込まれている AssemblyLine の入力パラメーターと出力パラメーターを抽出し、その情報を使用して入力 WSDL メッセ

ージおよび出力 WSDL メッセージの WSDL 部分を生成します。「初期作業項目」スキーマと「結果項目」スキーマの項目属性それぞれについて、「ネイティブ構文」列に属性の Java タイプ (例えば、`java.lang.String`) を指定することが必須です。このユーティリティーで生成した WSDL ファイルを、後から手動で編集することもできます。

生成される WSDL の中で定義される SOAP 操作の操作スタイルは、*rpc* です。

WSDL 生成ユーティリティーでは、複素数タイプの <types...>...</types> セクションを WSDL 内に生成することができません。

コネクターの操作

コネクター操作を扱うときは、以下のリストに示す点に注意が必要です。

Web サービス受信側サーバー・コネクターは、以下の情報を HTTP/SOAP 要求から取り出してコネクターの *conn* 項目の属性に格納し、作業項目にマップする準備をします (408 ページの『スキーマ』 も参照してください)。

- 要求が送信された先のホスト名 (ローカル・ホスト)。*host* 属性に格納されます。
- 要求された HTTP リソース。*requestedResource* 属性に格納されます。
- *soapAction* HTTP ヘッダーの値。*soapAction* 属性に格納されます。
- 「**SOAP メッセージ入力タイプ**」関数コンポーネント・パラメーターの値が「**ストリング**」である場合、SOAP 要求メッセージは `java.lang.String` オブジェクトとして *soapRequest* 属性に格納されます。
- 「**SOAP メッセージ入力タイプ**」関数コンポーネント・パラメーターの値が「**DOMElement**」である場合、SOAP 要求メッセージは `org.w3c.dom.Element` オブジェクトとして *soapRequest* 属性に格納されます。
- WSDL ファイルが要求されたかどうか。*wsdlRequested* 属性に入ります。要求された場合 (この値が **true** の場合)、その他の属性は設定されません。

AssemblyLine の応答ステージに達した時点で、このコネクターは、テキスト XML 形式または *DOMElement* 形式の SOAP 応答メッセージを作業 項目の *soapResponse* 属性から取得する必要があります (出力用のマッピングのため)。

- 「**SOAP メッセージの戻りタイプ**」関数コンポーネント・パラメーターの値が「**ストリング**」である場合、SOAP 応答メッセージは、AssemblyLine によって `java.lang.String` オブジェクトとして *soapResponse* 属性に格納されている必要があります。
- 「**SOAP メッセージの戻りタイプ**」関数コンポーネント・パラメーターの値が「**DOMElement**」である場合、SOAP 応答メッセージは、AssemblyLine によって `org.w3c.dom.Element` オブジェクトとして *soapResponse* 属性に格納されている必要があります。

その後、コネクターは、SOAP 応答メッセージを HTTP 応答にラップして、Web サービス・クライアントに戻します。

関連情報

22 ページの『Axis Easy Web Service サーバー・コネクター』.

Windows ユーザー/グループ・コネクター

Windows ユーザー/グループ・コネクター (旧バージョンの IBM Security Directory Integrator では NT4 コネクターと呼ばれていました) は、Windows NT セキュリティー・データベースを使用して作動します。ここに記載されている情報とリンクを使用することで、それについて詳しく知ることができます。

このコネクターは、Windows ユーザーとグループ (Windows NT セキュリティー・データベースの 2 つの基本エンティティー) を取り扱います。このコネクターは、ローカル Windows NT マシン、1 次ドメイン・コントローラー・マシン、および他のドメインの 1 次ドメイン・コントローラー・マシン上にある Windows NT セキュリティー・データベースの読み取りおよび変更の両方を行うことができます。

注: このコネクターは Windows NT API に依存し、Windows プラットフォームでのみ稼働します。

このコネクターは、Win32 API for Windows NT および Windows 2000/2003 のユーザー・アカウントおよびグループ・アカウントを使用して、Windows NT4 および Windows 2000 SAM データベースに接続するように設計されています。ユーザーは Windows 2000 SAM データベースに接続できますが、このコネクターは、NT4 との互換性を持つ属性のみを読み書きします (つまり、Windows ユーザー/グループ・コネクターには NT4 属性からなる定義済みの静的属性マップ・テーブルがあります)。したがってこのコネクターでは、Windows 2000/2003 のネイティブ属性またはユーザー定義の属性はサポートされません。

このコネクターの詳細な機能仕様、アーキテクチャーの説明、およびハードウェアとソフトウェアの要件については、416 ページの『Windows ユーザー/グループ・コネクターの機能仕様とソフトウェア要件』を参照してください。

前提条件

Windows ユーザー/グループ・コネクターを正しく実行し、そのすべての機能を取得するには、ローカル管理者グループのメンバーであるユーザーが所有するプロセス内でコネクターを実行する必要があり、また、コネクターにドメイン・コントローラーおよびその他のドメイン (アクセスする場合) へのログオン特権が設定されている必要があります。

コネクターの「**ユーザー名**」および「**パスワード**」パラメーターが、上記の要件を満たすアカウントを指定するように設定されている場合は、この前提条件を省略できます。

Windows ユーザー/グループ・コネクターは、次のモードで稼働するように設計されインプリメントされています。

- イテレーター
- ルックアップ
- AddOnly
- 削除
- 更新

注: このコネクターでは、拡張リンク基準はサポートされていません (「」の『拡張リンク基準』を参照)。

構成

以下に示すパラメーターを使用して、Windows ユーザー/グループ・コネクターを構成することができます。

コンピューター名
このコネクターを実行するマシンの名前 (例えば **ntserver01**) またはそのマシンの IP アドレス (例えば **212.52.2.218**)。IBM Security Directory Integrator を実行しているマシンは、ターゲット・システムと同じドメインまたはワークグループの中になければなりません。

ユーザー名
これをブランクのままにした場合は、指定したマシンへのログオンは行われず、コネクターは、IBM Security Directory Integrator が実行されているプロセスの特権を持ちます。このパラメーターに何らかの値を指定した場合、コネクターは、このユーザー名、および「**パスワード**」パラメーターで指定されているパスワードを使用して、「**コンピューター名**」のマシンへのログオンを試行します。

パスワード
このパラメーターの値が考慮されるのは、「**ユーザー名**」パラメーターに非ブランク値が設定されている場合のみです。その場合にログオン操作に使用するパスワードを指定します。

項目タイプ
設定できる値は、**User** (コネクターがユーザーにより構造化されたデータを操作することを指定)、または **Group** (コネクターがグループにより構造化されたデータを操作することを指定) のいずれかです。

ページ・サイズ
コネクターがユーザーおよびグローバル・グループを検索するときに、Windows NT または Active Directory が 1 つのチャンクとして戻す項目 (ユーザーおよびグローバル・グループ) の数を指定します。指定できる値の範囲は **1** から **100** です。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

リンク基準の構成

Windows ユーザー/グループ・コネクターをルックアップ・モード、更新モード、および削除モードで使用するときは、リンク基準を構成します。このコネクターは、1 つの項目を一意的に識別するリンク基準のみをサポートします。形式は厳密であり、この形式と一致しないリンク基準を渡すと、以下の例外が発生します。

```
Unsupported Link Criteria structure.
```

必要なリンク基準構造は、項目タイプに応じて以下のとおりです。

ユーザー

Windows ユーザー/グループ・コネクターの「**項目タイプ**」パラメーターが **User** に設定されています。このパラメーターは、以下のような 1 つの行から成ります。

- コネクター属性は **UserName** に設定します。
- オペランドは **equals** に設定します。
- 値は、ユーザー・アカウントの名前 (例えば**ユーザー名**) に設定するか、ユーザー・アカウントを受け取るテンプレートにより構成します。

グループ

Windows ユーザー/グループ・コネクターの**項目タイプ**・パラメーターが **Group** に設定されています。このパラメーターは、以下に示す 2 行から成ります。

1. 1 行目
   - コネクター属性は **GroupName** に設定します。
   - オペランドは **equals** に設定します。
   - 値は、グループ・アカウントの名前 (例えば**グループ名**) に設定するか、グループ・アカウントを受け取るテンプレートにより構成します。
2. 2 行目
   - Windows ユーザー/グループ・コネクター属性は **IsGlobal** に設定します。
   - オペランドは **equals** に設定します。
   - 値は、**True** (1 行目に指定したグループ・アカウントがグローバルであることを示す)、または **False** (グループ・アカウントがローカルであることを示す) に設定します。グローバル・グループ・アカウントかローカル・グループ・アカウントかを示す **True** または **False** の値を受け取るテンプレートにより構成することもできます。

その他

Windows ユーザー/グループ・コネクターで作業を行うときは、以下のセクションに示す情報を考慮してください。

ユーザーおよびグループのアカウント名:

以下に示す情報を使用して、ユーザーとグループのアカウント名を参照できます。

ドメイン・コントローラー・マシン上

ユーザーおよびグループが検索され、次の形式でアクセスされます。

USER_NAME,*GROUP_NAME*

非ドメイン・コントローラー・マシン上

ローカル・ユーザーおよびグループが検索され、次の形式でアクセスされます。

USER_NAME,*GROUP_NAME*

グローバル・グループおよびドメイン・ユーザー (非ドメイン・コントローラー・マシン上のローカル・グループのメンバーでもよい) が検索され、次の形式でアクセスされます。

DOMAIN_NAME¥GLOBAL_GROUP_NAME,DOMAIN_NAME¥USER_NAME

新規ユーザーの作成:

新規ユーザーを作成するときには、以下に示す属性を使用できます。

Windows ユーザー/グループ・コネクターで新規ユーザーを作成するときに、以下の属性のいずれかが省略されているか、または **NULL** 値が割り当てられている場合は、次のように自動的にデフォルト値が割り当てられます。

Flags アカウントは、**通常アカウント**および**ユーザー・パスワードの有効期限なし**としてマークされます。

AccountExpDate
アカウントの有効期限なしを示す値が設定されます。

LogonHours
時間制限なし (例えば、ユーザーは常にログオンできる) を示す値が設定されます。

ユーザー・パスワードの設定

ユーザー・パスワードの値は検索できないことに注意してください。Windows は、パスワード値を読み取り不能な形式で保管します。

AssemblyLine が、ある Windows マシンから別のマシンにユーザーをコピーする場合は、ユーザーは **Password** 属性値を手動で設定する必要があります。

値のない **Password** 属性を渡してユーザーを追加すると、空のパスワードを持つユーザーが作成されることになります。

値のない **Password** 属性を渡してユーザーを変更した場合は、旧パスワードがそのまま使用されます。

ユーザー 1 次グループ/グローバル・グループのメンバーシップの設定

すべてのドメイン・ユーザーは、その 1 次グループのメンバーである必要があります。

これは、**PrimaryGroup** 属性に設定する値が **GlobalGroups** 属性内になければならないことを意味します。 PrimaryGroup 属性に値がない場合は、「ドメイン・ユーザー」が設定されます。

グループの操作

一部の属性に対する削除、名前変更、および変更の操作のみを実行することができます。

Windows 固有の定義済みグループがいくつかあります。また、これらのグループでは使用できない操作がいくつかあります。これらの定義済みグループに対して無効な操作を試行すると、例外がスローされます。

以下に示すのは、インストールされている Windows に従って編成されたこれらのグループのリストです。

ドメイン・コントローラー:
- グローバル・グループ
  - ドメイン管理者
  - ドメイン・ユーザー
- ローカル・グループ
  - 管理者
  - ユーザー
  - ゲスト
  - バックアップ・オペレーター
  - レプリケーター
  - アカウント・オペレーター
  - 印刷オペレーター
  - サーバー・オペレーター

非ドメイン・コントローラー:
- ローカル・グループ
  - 管理者
  - ユーザー
  - ゲスト
  - バックアップ・オペレーター
  - レプリケーター
  - パワー・ユーザー

文字セット

Unicode がサポートされています。

例

Windows ユーザー/グループ・コネクターについてさらに詳しく知るには、以下に示す例を使用できます。

ご使用の IBM Security Directory Integrator インストール済み環境の *TDI_install_dir*/examples/NT4 ディレクトリーにナビゲートします。

Windows ユーザー/グループ・コネクターの機能仕様とソフトウェア要件

以下に示す情報とリンクを使用して、Windows ユーザー/グループ・コネクターの機能の仕様とソフトウェア要件について知ることができます。以降の各セクションでは、機能について詳しく説明します。

Windows ユーザー/グループ・コネクターは、以下に示すように、Windows の定義および制約条件に従って、Windows システムのユーザー管理とグループ管理の両方に Windows ユーザー/グループ・データベースへのアクセスをインプリメントしま

す。追加のバックグラウンド情報については、「Overview of Users and Groups」および「Managing local and remote Users and Groups」を参照してください。

機能

ユーザーおよびグループ・データの抽出

ユーザー情報とグループ情報の読み取りについては、以下に示す情報を参照できます。

Windows ユーザー/グループ・コネクターは、Windows ユーザー/グループ・リポジトリーからユーザー情報とグループ情報の両方を読み取ります。これには、グループおよびユーザーのメタデータ、および関係情報 (例えば、**users** グループおよび **groups** グループのメンバーシップ) が含まれます。コネクターは、ローカルおよびドメインのユーザー・データまたはグループ・データを読み取ります。データは、Windows から読み取られ、編成されて、IBM Security Directory Integrator で予定しているコンテナーに入れられます。

ユーザーおよびグループ・データの追加

ユーザー情報とグループ情報の追加については、以下に示す情報を参照できます。

Windows ユーザー/グループ・コネクターは、ローカル・マシンとドメイン・コントローラーの両方にユーザー情報とグループ情報を追加します。ドメイン・コントローラーを操作しているときは、コネクターはローカル・グループとグローバル・グループの両方を作成することができます。ドメイン・コントローラー以外のマシンを操作するときには、コネクターは、Windows が設定するセキュリティー上の制約条件に従い、ローカル・グループのみを作成することができます。

グループ・メンバーシップの変更

ここに記載されている情報を使用することで、グループ・メンバーシップを変更できるようになります。

Windows ユーザー/グループ・コネクターは、ローカル・グループおよびグローバル・グループの両方についてグループ・メンバーシップを変更します。グループへのメンバーの追加は、Windows NT のセキュリティー制限に従って、次のように行うことができます。

- グローバル・グループには、そのグループのドメイン内のユーザーのみをメンバーとして含めることができます。
- ローカル・グループには、そのドメインまたは他の任意のトラステッド・ドメイン内のグローバル・グループおよびユーザーを、メンバーとして含めることができます。ただし、ローカル・グループに他のローカル・グループを含めることはできません。
- ローカル・マシンには、特定グループのメンバーとなっていないユーザーが存在することができます。
- ドメイン・コントローラー上の各ユーザーは、同じ 1 次グループに属していなければなりません。ユーザーの 1 次グループは、ドメイン内のどのグローバル・グループでも構いません。ユーザーの 1 次グループは変更できますが、ユーザーは常に自分の 1 次グループのメンバーです。

ユーザーおよびグループ・データの変更

Windows ユーザー/グループ・コネクターは、ローカル・マシンおよびドメイン・コントローラーの両方の外部プロパティーおよびグループ・プロパティーを変更します。

ドメイン・コントローラーに接続されているときは、コネクターは、ローカル・グループとグローバル・グループの両方のプロパティーを変更することができます。

ユーザーおよびグループ・データの削除

Windows ユーザー/グループ・コネクターは、ローカル・マシンおよびドメイン・コントローラーの両方からユーザーとローカル・グループを削除することができます。ドメイン・コントローラーを操作しているときは、コネクターはローカル・グループとグローバル・グループの両方を削除することができます。

z/OS LDAP 変更ログ・コネクター

以下に示す情報とリンクを使用して、z/OS LDAP 変更ログ・コネクターについて知ることができます。

注: IBM Security Directory Integrator バージョン 7.2 以降では z/OS オペレーティング・システムはサポートされません。

z/OS LDAP 変更ログ・コネクターは、LDAP コネクターの特殊インスタンスです。z/OS ディレクトリー・サーバーとともに使用され、TCP/IP を使用した LDAP プロトコル (「zLDAP」) を使用してアクセスするよう構成されています。

パスワード・ポリシーの運用属性に対する変更内容が `cn=changelog` に記録される方法は、z/OS 上の IBM Security Directory Server の場合と、その他のプラットフォーム上で稼働する分散 IBM Security Directory Server の場合では、いくつかの点で異なります。 2 つのバージョンの動作における既知の違いについて詳しくは、174 ページの『分散 TDS と z/OS TDS での変更ログの違い』を参照してください。

このコネクターでは、項目レベル、属性レベル、および属性値レベルでデルタ・タグがサポートされています。 LDIF パーサーが属性レベルと属性値レベルでデルタ・サポートを提供します。

このコネクターは、RACF® (リソース・アクセス管理機能) LDAP サーバーの変更ログから、変更内容をインターセプトすることができます。RACF は z/OS のセキュリティー・マネージャーで、ユーザー名およびパスワードを含むデータベースを維持します。このデータベースに対する変更は、IBM Security Directory Server などの LDAP サーバーの変更ログに記録することができます。このサーバーの変更ログには、GDBM LDAP インターフェースおよび RACF データベース自体、つまり SDBM インターフェースを介してアクセスできます。このコネクターは、異なる z/OS マシンまたはその他の分散プラットフォーム上の LDAP サーバー全体にわたる機密情報 (ユーザー名、パスワードなど) の変更の伝播に適しています。

コネクターにより、サーバーの変更ログの *modrdn* 操作が検出されます。詳しくは、245 ページの『modrdn 操作の検出と処理』を参照してください。

属性のマージ動作

以下に示す情報を使用して、属性のマージ動作と処理のモードについて知ることができます。

旧バージョンの IBM Security Directory Integrator では、変更ログ項目の属性と実際のディレクトリー項目の変更された属性の間で、z/OS LDAP 変更ログ・コネクターのマージが実行されます。変更された属性を検出できないため、これにより問題が発生します。現行バージョンのコネクターには、このような状態を解決するためのロジックがあります。このロジックは、パラメーター「**マージ・モード**」によって設定されます。モードは以下のとおりです。

- **変更ログおよび変更データのマージ**: コネクターは、変更ログ項目の属性と実際のディレクトリー項目の変更された属性をマージします。これは旧来の実装であり、以前のバージョンとの互換性を確保します。
- **変更データのみを戻す**: 変更/追加された属性のみを戻し、変更ログ・イテレーターとデルタ・モードを容易にします。これはデフォルト設定です。旧バージョンの IBM Security Directory Integrator で開発された、または旧バージョンから移行した構成の場合、同じ動作を確保するために手動で「**変更ログおよび変更データのマージ**」を選択する必要が生じることがあります。
- **両方とも戻す**: 実際のディレクトリー項目の変更された属性を含む項目および変更ログ項目の属性を含む追加属性「changelog」を戻します。これにより、2 つのセットの属性を簡単に区別できます。

デルタ・タグはすべてのマージ・モードでサポートされているので、大量のスクリプトを使用しなくても、異なる LDAP サーバー間で項目を転送できます。

構成

ここに記載されているパラメーターを使用することで、z/OS 変更ログ・コネクターを構成できるようになります。

LDAP URL
接続の LDAP URL (ldap://*host*:*port*)。

ログイン・ユーザー名
サーバーへの認証に使用する LDAP 識別名。匿名アクセスの場合はブランクにしておいてください。

ログイン・パスワード
信任状 (パスワード)。

認証メソッド
LDAP 認証のタイプ。これは、次のいずれかです。

- **無名** - この認証メソッドが設定されている場合、クライアントが接続されているサーバーでは、そのクライアントがどのクライアントであるかを認識しません (クライアント情報を必要としません)。サーバーは、このようなクライアントが、非認証ユーザー用に構成されたデータにアクセスすることを許可します。ユーザー名が指定されない場合、コネクターは、自動的にこの認証メソッドを指定します。ただし、このタイプの認証が選択され、**ログイン・ユーザー名**と**ログイン・パスワード**が指定された場合、コネクターは、自動的に認証メソッドを単純に設定します。

- **単純**: **ログイン・ユーザー名**と**ログイン・パスワード**を使用します。**ログイン・ユーザー名**と**ログイン・パスワード**を指定しない場合は、無名と見なされます。SSL プロトコルを使用して LDAP サーバーと通信するようにコネクターが構成されている場合を除いて、コネクターは、完全修飾された識別名 (DN) およびクライアント・パスワードを平文で送信することに注意してください。
- **CRAM-MD5** - これは SASL 認証メカニズムの 1 つです。LDAP サーバーは接続に際して、特定のデータを LDAP クライアント (すなわち、このコネクター) に送信します。すると、クライアントは MD5 暗号化方式を使用して暗号化された応答をパスワードとともに送信します。その後、LDAP サーバーはクライアントのパスワードをチェックします。
  CRAM-MD5 がサポートされるのは LDAP v3 サーバーのみです。 のどのサポート対象バージョンでもサポートされません。
- **SASL** - クライアント (このコネクター) は、LDAP サーバーへの接続時に Simple Authentication and Security Layer (SASL) 認証メソッドを使用します。このタイプの認証の稼働パラメーターは、「**追加のプロバイダー・パラメーター**」オプションを使用して指定する必要があります。例えば、DIGEST-MD5 認証をセットアップするには、以下のパラメーターを「追加のプロバイダー・パラメーター」フィールドに追加する必要があります。

  ```
  java.naming.security.authentication:DIGEST-MD5
  ```

  SASL 認証およびパラメーターについて詳しくは、http://java.sun.com/products/jndi/tutorial/ldap/security/sasl.html を参照してください。

  注: すべてのディレクトリー・サーバーですべての SASL メカニズムがサポートされているわけではなく、デフォルトで SASL メカニズムが無効になっている場合もあります。詳しくは、接続先のディレクトリー・サーバーの資料および構成オプションを確認してください。

SSL の使用
「SSL の使用」が **true** である (チェック・マークが付いている) 場合、コネクターは SSL を使用して LDAP サーバーに接続します。これに従ってポート番号の変更が必要になることがあります。

変更ログ・ベース
変更ログが保持される検索ベース。標準 DN は **cn=changelog** です。

追加のプロバイダー・パラメーター
複数の追加パラメーターを JNDI レイヤーに受け渡せるようにします。名前:値のペアとして 1 行に 1 つずつ指定します。

イテレーター状態キー
IBM Security Directory Integrator のユーザー・プロパティー・ストアに現在の同期状態を格納するパラメーターの名前を指定します。これは、IBM Security Directory Integrator のユーザー・プロパティー・ストアの特定のインスタンスに格納されるすべてのパラメーターに対して固有名である必要があります。

開始位置
開始変更番号を指定します。各変更ログ項目には **changenumber=intvalue**

という名前が付けられ、コネクターはこのパラメーターに指定した番号から開始され、自動的に 1 つずつ番号が増加します。特殊値 **EOD** は、変更ログの終わりから開始することを意味します。

状態キーの維持
コネクターの状態をシステム・ストアに保管する方式を制御します。デフォルトは「**サイクルの終わり**」です。以下のいずれかを選択できます。

読み取り後
AssemblyLine の残りの部分を続行する前、Directory Server の変更ログから項目を読み取るときにシステム・ストアを更新します。

サイクルの終わり
AssemblyLine のすべてのコネクターおよびその他のコンポーネントの評価と実行が完了した時点で、変更ログ番号を使用してシステム・ストアを更新します。

手動 このコネクターの状態情報を使用したシステム・ストアの自動更新をオフにします。この場合、AssemblyLine 内の特定の時点において、z/OS LDAP 変更ログ・コネクターの *saveStateKey()* メソッドを手動で呼び出し、状態を保管する必要があります。

マージ・モード
変更ログ項目の属性と実際のディレクトリー項目の変更された属性のマージに使用されるメソッドを管理します。デフォルトは「**変更データのみを戻す**」です。指定できる値は、以下のとおりです。

- **変更ログおよび変更データのマージ** - 7.0 よりも前の実装。以前のバージョンとの互換用。
- **変更データのみを戻す**: 変更/追加された属性のみを戻します。
- **両方とも戻す**: 実際のディレクトリー項目の変更された属性および変更ログ属性を持つ項目を含む追加属性「changelog」を戻します。

タイムアウト
コネクターが次の変更ログ項目を待つ秒数を指定します。デフォルトは 0 です。このデフォルト値では、無期限で待機します。

スリープ間隔
コネクターが各ポーリング間でスリープする秒数を指定します。デフォルトは 60 です。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

注: タイムアウトまたはスリープ間隔の値を変更すると、そのピアは自動的に有効な値に調整されます (例えば、タイムアウトをスリープ間隔よりも大きくした場合、未編集の値は変更された値にあわせて調整されます)。調整はフィールド・エディターがフォーカスを失うときに実行されます。

関連情報

ここに記載されているリンクを参照することで、z/OS 変更ログ・コネクターについて詳しく知ることができます。

243 ページの『LDAP コネクター』,
9 ページの『Active Directory 変更検出コネクター』,
173 ページの『IBM Security Directory Integrator 変更ログ・コネクター』,
341 ページの『Sun Directory 変更検出コネクター』.

第 3 章 パーサー

ストリーム・ベースのコネクターと共にパーサーを使用して、コネクターのバイト・ストリーム経由で送受信される内容を解釈したり生成したりすることができます。

「**スキーマのディスカバー**」をクリックすると、パーサーが呼び出しコネクターと協働して、基礎データ・ストリームのスキーマを見つけます。

解析しようとしているバイト・ストリームが選択したパーサーに適合しない場合は、ユーザーは `sun.io.MalformedInputException` エラーを受け取ります。このエラー・メッセージは、例えば、「**スキーマ**」タブを使用してファイルをブラウズしているときなどに表示されます。

基本パーサー

ここに記載されているリンクを参照して、指定されたパーサーのリストについて理解することができます。

- 425 ページの『CBE パーサー』
- 429 ページの『CSV パーサー』
- 431 ページの『DSMLv1 パーサー』
- 432 ページの『DSMLv2 パーサー』
- 445 ページの『固定レコード・パーサー』
- 449 ページの『HTTP パーサー』
- 657 ページの『IdML パーサー』
- 445 ページの『JSON パーサー』
- 457 ページの『LDIF パーサー』
- 461 ページの『行リーダー・パーサー』
- 461 ページの『スクリプト・パーサー』
- 465 ページの『単純なパーサー』
- 466 ページの『SOAP パーサー』
- 468 ページの『SPMLv2 パーサー』
- 476 ページの『単純 XML パーサー』
- 481 ページの『XML パーサー』
- 493 ページの『XML SAX パーサー』
- 496 ページの『XSL ベース XML パーサー』

文字エンコード変換

IBM Security Directory Integrator の記述言語である Java では、Unicode (2 バイト) 文字セットがサポートされます。ここに記載されている情報とリンクを使用することで、文字エンコード変換を実行できます。

AssemblyLine およびコネクター内で処理されるストリングおよび文字は、常に Unicode であると想定されます。多くのコネクターは、文字エンコードを使用するための何らかの手段を備えています。ローカル・システムでテキスト・ファイルを読み取るときには、Java は、既にデフォルトの文字エンコード変換を確立しています。このデフォルトの変換は、実行しているプラットフォームによって異なります。

IBM Security Directory Integrator サーバーの **-n** コマンド行オプションは、サーバーが新規ファイルの作成時に使用する構成ファイルの文字セットを指定します。また、構成ファイルにはこの文字セットの指定子も組み込まれるため、後でファイルを読み取るときに、ファイルの内容を正しく解釈できます。

ただし、テキスト・ファイルからのデータの読み取り、またはそこへのデータの書き込みを行うとき、ファイルによって文字エンコードの方式が異なる場合がしばしばあります (これは、別のオペレーティング・システムが実行されているマシンで作成されたファイルを読み取る場合に発生する可能性があります)。パーサーを必要とするコネクターは、通常、パーサー構成の「**文字セット**」パラメーターを受け入れます。このパラメーターを設定する場合は、IANA Charset Registry に準拠し、Java ランタイム内にある許容変換テーブルの 1 つに設定してください。このパラメーターを設定しない場合は、ほとんどのパーサーはローカル文字エンコード方式を使用します。一部のパーサーには、特定のデフォルト文字エンコード方式があります。個々のパーサーの詳細については、この資料を参照してください。

UTF-8、UTF-16、または UTF-32 でエンコードされている場合、一部のファイルでは、ファイルの先頭にバイト・オーダー・マーカー (BOM) が含まれることがあります。BOM の目的は、InputStream の文字へのエンコードに使用されるアルゴリズムを検索することです。BOM は文字 `0xFEFF` のエンコードです。これは、使用されるエンコードのシグニチャーとして役立ちます。IBM Security Directory Integrator ファイル結合子は、 BOM を認識しません。また、以下の IBM Security Directory Integrator パーサーも BOM を認識しません。

- CSV パーサー
- 固定パーサー
- HTTP パーサー
- LDIF パーサー
- 行リーダー・パーサー
- スクリプト・パーサー
- 単純なパーサー

BOM を含むファイルの読み取りを試行しており、パーサーがこの読み取りの処理方法を認識していない場合、使用不可能なデータが戻されないようにするために、以下のコードをコネクターの**選択前**などのフックに追加する必要があります。

```
var bom = thisConnector.connector.getParser().getReader().read(); // skip the BOM = 65279
if (bom != -1 && bom != 65279) {
//make sure that we are skipping the BOM and not any other meaningful character.
 throw "Invalid BOM";
}
```

このコードでは、パーサーに対して正しい文字セットを指定していると仮定して、BOM を読み取り、スキップします。この回避策は、パーサーが BOM を認識または処理しない場合または一般に BOM をスキップする必要がある場合にのみ必要となります。

HTTP プロトコルに何らかの処理が必要となります。詳細は、 449 ページの『HTTP パーサー』で 456 ページの『文字セット/エンコード』にある HTTP パーサーについての説明で文字セット・エンコードに関するセクションを参照してください。

可用性

IANA Charset Registry (http://www.iana.org/assignments/character-sets) を参照してください。

Windows コンピューターの共通文字セットは CP850 です。

CBE パーサー

パーサーは、コネクターの構成の一部として入力ストリームまたは出力ストリーム (あるいはその両方) に接続することができます。AssemblyLine フローで別のコンポーネントとして別個にデプロイする必要はありません。

このパーサーは、 481 ページの『XML パーサー』を拡張します。また、このパーサーは、指定された属性として CBE オブジェクトに基づいた XML を書き込む代わりに、入力ストリームから XML を読み取り、この XML を CBE オブジェクトに変換するよう設計されています。このパーサーの操作は 521 ページの『CBE 関数コンポーネント』と似ていますが、パーサーとしてパッケージされている点が異なります。

例えば、一部の AssemblyLine で受信したすべての CBE オブジェクトをファイルに保存する必要がある場合があります。この場合、 119 ページの『ファイル・コネクター』を CBE パーサーとともに AddOnly モードで使用して、CBE オブジェクト自体を出力マップの「event」属性に受け渡します。

CBE パーサーが構成されたイテレーター・モードでファイル・コネクターを使用すると、CBE オブジェクトを再度読み取ることができ、入力マップの「event」属性でこのオブジェクトを受け取ります。

パーサーの使用

CBE パーサーは、XML から読み取る場合、すべての標準 CBE 属性および入力マップの属性としての CBE オブジェクトを戻します。ここに記載されている情報を参照して、パーサーの使用方法について理解することができます。

CBE パーサーは、XML に書き込む場合、CBE オブジェクトが出力マップの「event」属性に設定されることを予期します。「event」が設定されていない場合、CBE パーサーは、少なくとも、出力マップに必要な CBE 属性がすべて設定されていることを予期します。設定されていない場合は、エラーがスローされます。

このパーサーに受け渡される XML は、CBE 仕様および CBE スキーマに対応している必要があります。

CBE 入力マップおよび出力マップの属性

CBE 仕様の構造は複雑です。属性を定義できるようにするため、一部のコンポーネントの属性マップでは、属性のドット表記がサポートされています。

属性とその推奨値について詳しくは、http://www-128.ibm.com/developerworks/autonomic/books/fpy0mst.htm#ToC_91 の「IBM Autonomic Computing Toolkit Developer's Guide」を参照してください。

以下の「CBE 属性」に、CBE イベント・オブジェクトの作成に使用する属性や出力マップの event 属性または eventXml 属性に提供される CBE オブジェクトから書き込まれる属性を示します。

表 44. CBE 属性

属性名	属性タイプ	説明
CreationTime	ストリング	イベントが作成された時刻。デフォルト値は、CBEGeneratorFC でイベント・オブジェクトが作成された時刻です。
GlobalInstanceId	ストリング	イベントの 1 次 ID。ユーザーから渡されない場合は固有の ID が生成されます。
Message	ストリング	オプションのプロパティー
Severity	整数	オプションのプロパティー。値の範囲は 0 から 70 です。
ExtensionName	ストリング	オプションのプロパティー。
SequenceNumber	整数	オプションのプロパティー
RepeatCount	整数	オプションのプロパティー。
ElapsedTime	整数	RepeatCount が設定されていない場合は、オプションのプロパティー。
Priority	整数	オプションのプロパティー。値の範囲は 0 から 100 です。
situation.		***これを表記することで、状態オブジェクトのプロパティーが定義されます。***
situation.CategoryName	ストリング	状態のタイプを記述します。定義されている値を以下に示します。 • StartSituation • StopSituation • ConnectSituation • ConfigureSituation • RequestSituation • FeatureSituation • DependencySituation • CreateSituationDestroy • SituationReportSituation • AvailableSituation • OtherSituation このプロパティーは必須です。

表 44. CBE 属性 (続き)

属性名	属性タイプ	説明
situation.reasoningScope		状態の有効範囲を記述します。これは必須プロパティーです。
availableSituation.operationDisposition	ストリング	CategoryName が AvailableSituation の場合に必要です。
availableSituation.processingDisposition	ストリング	CategoryName が AvailableSituation の場合に必要です。
availableSituation.availabilityDisposition	ストリング	CategoryName が AvailableSituation の場合に必要です。
configureSituation.successDisposition	ストリング	CategoryName が ConfigureSituation の場合に必要です。
connectSituation.successDisposition	ストリング	CategoryName が ConnectSituation の場合に必要です。
connectSituation.situationDisposition	ストリング	CategoryName が ConnectSituation の場合に必要です。
createSituation.successDisposition	ストリング	CategoryName が CreateSituation の場合に必要です。
dependencySituation.dependencyDisposition	ストリング	CategoryName が DependencySituation の場合に必要です。
destroySituation.successDisposition	ストリング	CategoryName が DestroySituation の場合に必要です。
featureSituation.featureDisposition	ストリング	CategoryName が FeatureSituation の場合に必要です。
reportSituation.reportCategory	ストリング	CategoryName が ReportSituation の場合に必要です。
requestSituation.successDisposition	ストリング	CategoryName が RequestSituation の場合に必要です。
requestSituation.situationQualifier	ストリング	CategoryName が RequestSituation の場合に必要です。
startSituation.successDisposition	ストリング	CategoryName が StartSituation の場合に必要です。
startSituation.situationQualifier	ストリング	CategoryName が StartSituation の場合に必要です。
stopSituation.successDisposition	ストリング	CategoryName が StopSituation の場合に必要です。
stopSituation.situationQualifier	ストリング	CategoryName が StopSituation の場合に必要です。
otherSituation.any	ストリング	CategoryName が OtherSituation の場合に必要です。 **注:** XML が CBE 1.0.1 スキーマ (<someTag> 任意のテキスト値 </someTag> など) に準拠するよう、この値は XML エレメントとして表される必要があります。この値は、1 つのエレメントのみにラップされている必要があります (例えば、<someTag>val</someTag> <anotherTag>val2</anotherTag> は有効な値ではありません)。
SCI.		***この表記により、ソース・コンポーネントの識別が記述されます。CommonBaseEvent の必須プロパティーです。***
SCI.location	ストリング	
SCI.locationType	ストリング	
SCI.executionEnvironment	ストリング	
SCI.component	ストリング	
SCI.subcomponent	ストリング	
SCI.componentIdType	ストリング	
SCI.componentType	ストリング	
RCI.		***この表記では、レポーター・コンポーネントのコンポーネント識別情報が記述されます。これは必須プロパティーではありません。***
RCI.location	ストリング	
RCI.locationType	ストリング	

表 44. CBE 属性 (続き)

属性名	属性タイプ	説明
RCI.executionEnvironment	ストリング	
RCI.component	ストリング	
RCI.subcomponent	ストリング	
RCI.componentIdType	ストリング	
RCI.componentType	ストリング	
X.		***この表記では、ExtendedDataElement (EDE) が記述されます。これは必須プロパティーではありません。***
X.attributeName	ストリング	`attribute Name` という名前とユーザー定義値を持つ EDE を作成します。
X.attributeName.childAttributeName	ストリング	`attribute Name` という名前の EDE と `childAttributeName` という名前の子エレメントを作成します。

AssemblyLine で CBE FC と CBE パーサーのいずれかを使用して CBE オブジェクトを処理する場合は、入出力マップを扱う必要があります。

表 45. 入力マップの属性

属性名	属性タイプ	説明
Event	org.eclipse.hyades.logging.events.cbe.CommonBaseEvent	属性から変換された CBE オブジェクト。
eventXml	ストリング	変換された CBE オブジェクトの XML 表現。
テーブル (426 ページの表 44) からすべての CBE 属性を組み込みます。		

表 46. 出力マップ属性

属性名	属性タイプ	説明
Event	org.eclipse.hyades.logging.events.cbe.CommonBaseEvent	属性に変換される CBE オブジェクト。
eventXml	ストリング	CBE オブジェクトに解析され、その後属性に変換される XML。
テーブル (426 ページの表 44) からすべての CBE 属性を組み込みます。		

構成

ここに記載されているパラメーターを使用することで、CBE パーサーを構成できるようになります。

文字エンコード

読み取りまたは書き込み時に使用される文字エンコードを設定します。デフォルト値は UTF-8 です。

XML を読み取る場合は、入力ソースにエンコードが定義されていない場合にのみ、このパラメーターが使用されます。

このパーサーは XML パーサーを拡張したものであるため、XML パーサーの場合と同じ考慮事項が適用されます。

XML 宣言の省略
設定すると、このパーサーは、出力ストリームで XML 宣言ヘッダーを省略します。

XML の妥当性検査
設定すると、このパーサーは、CBE 仕様に基づく XSD スキーマを使用して読み取り XML の妥当性を検査します。

詳細ログ
チェックすると、このパーサーは追加のログ・メッセージを出力します。

関連情報

521 ページの『CBE 関数コンポーネント』,

481 ページの『XML パーサー』,

476 ページの『単純 XML パーサー』.

CSV パーサー

Comma Separated Values (CSV) パーサーは、CSV 形式のデータの読み取りおよび書き込みを行います。ここに記載されている情報を使用することで、CSV パーサーを操作できるようになります。

注: 構成エディターでは、パラメーターはコネクターの「**パーサー**」タブで設定されます。フィールド分離文字として TAB を使用する場合は、**¥t** を使用する必要がありますが、複数のフィールド名を指定するときは、フィールド名の間に実際のタブ文字を使用する必要があります。

出力では、多値属性の値は最初の 1 つだけが出力されます。

構成

このパーサーは、以下のパラメーターを必要とします。

フィールド分離文字
各列を分離するために使用する文字 (通常はコンマまたはセミコロン) を指定します。指定しない場合は、パーサーは読み取り時には自動判別を試み、書き込み時にはコンマを使用します。エスケープ文字として円記号 (¥) を使用すると、印刷不能文字を指定できます。例えば、(¥t) はタブ文字を表します。

ソート・フィールド
ヘッダー・フィールドをアルファベット順 (昇順) に書き込むには、このオプションをチェックします。デフォルトは false、つまりチェックは外されています。

フィールド名
パーサーが読み取りおよび書き込みを行う必要がある各列の名前を指定します。指定しない場合は、パーサーは最初の行を読み取り、その値をフィールド名として使用します。フィールド名の間にフィールド分離文字を使用することも、別個の行にそれぞれの名前を指定することもできます。

引用符の使用可能

このパラメーターを **true** に設定すると (つまり、チェックすると)、書き込み時にフィールドが以前のバージョンと同じ条件で引用符で囲まれて出力されます。このとき、引用符付きフィールドの中の引用符は 2 つ繰り返されます。

注: 引用符の使用可能を **false** に設定すると、フィールドはそのまま出力され、問題が発生する場合があります。

このパラメーターを **true** に設定すると、読み取り時にフィールドを囲む引用符が取り除かれ、パーサーが引用符で囲まれた属性 (列分離文字を含む属性など) を読み取ることができます。このパラメーターが **false** に設定されている場合は、引用符で区切られたフィールドが入力に含まれていると、パーサーによって予期しない値が戻されます。

すべてのフィールドを引用符で囲む

フィールドに引用符、分離文字、または改行が含まれている場合、すべてのフィールドを個別に引用符で囲みます。

ヘッダーの書き込み

このパラメーターのデフォルト値は **true** です。**ヘッダーの書き込み**を設定すると、パーサーによって出力される最初の行に、列区切り文字で分離されたすべてのフィールド名が出力されます。

BOM の書き込み

バイト・オーダー・マーク (BOM) をファイルに書き込みます。これを有効にするには、「**ヘッダーの書き込み**」パラメーターを true に設定する必要があります。

長い行のログ

1 行の最大バイト数を定義します。この最大数より長い行の行番号がログに記録されます。

余りを最後のフィールドに結合

チェックすると、行にある定義フィールド数を超えるすべての余分のフィールドを、新しい「余り」フィールドに結合します。

これらのフィールド、および暗黙的にそのフィールド数が、「**フィールド名**」パラメーターを使用して、またはこのパラメーターが存在しない場合はファイルの最初の行を使用して定義されます。

文字エンコード

使用される文字エンコード。 423 ページの『文字エンコード変換』も参照してください。

詳細ログ

このフィールドをチェックすると、追加のログ・メッセージが生成されます。

スキーマ

CSV パーサー・スキーマについては、ここに記載されている情報を通じて知ることができます。

CSV パーサーが入出力コネクター・マップに提供するスキーマは、このパーサーの「**フィールド名**」構成パラメーターの値から取得されます。パーサーは、このパラメーターのフィールドをコネクターのマップに単にコピーします。これにより、パーサーからすべてのフィールドを 1 つずつ対応するコネクター・マップにコピーしないで済みます。

DSMLv1 パーサー

DSMLv1 パーサーを使用して、XML 文書の読み取りおよび書き込みを行うことができます。このパーサーは、スキーマ項目を警告なしに無視します。

構成

ここに記載されているパラメーターを使用することで、DSMLv1 パーサーを構成できるようになります。

このパーサーは、以下のパラメーターを必要とします。

DN 属性
識別名 DSML 属性 (**$dn**) に使用する属性。

DSML 接頭部
XML エレメントが DSML ネーム・スペースに属することを指示する接頭部。デフォルトは **dsml** です。

DSML ネーム・スペース URI
このネーム・スペースを識別する URI。デフォルトは http://www.dsml.org/DSML です。

XML 宣言の省略
これをチェックした場合は、出力ストリームから XML 宣言が省かれます。

文書の妥当性検査
これをチェックした場合は、このパーサーは DTD/Schema 妥当性検査パーサーを要求します。

ネーム・スペースを意識
これをチェックした場合は、このパーサーはネーム・スペース認識パーサーを要求します。

文字エンコード
使用される文字エンコード。

このパーサーは単純 XML パーサーを拡張したものであるため、文字エンコードに関して同じ注意事項が適用されます。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

例

ここに記載されている例は、DSML 文書を動的に生成する方法を示しています。

```
var dsml = system.getParser ( "ibmdi.DSML" );
var entry = system.newEntry();
```

```
entry.setAttribute ("$dn", "uid=johnd,o=doe.com");
entry.setAttribute ("mail", "john@doe.com");
entry.setAttribute ("uid", "johnd");
entry.setAttribute ("objectclass", "top");
entry.addAttributeValue ("objectclass", "person");

dsml.setOutputStream ( new java.io.StringWriter() );
// Uncomment if you dont want the "<?xml version= ...." header
// dsml.setOmitXMLDeclaration ( true );
dsml.initParser();
dsml.writeEntry ( entry );
dsml.closeParser();

var result = dsml.getXML();
task.logmsg ( result );
```

以下の例は、スクリプトを使用して DSML 文書を読み取る方法を示しています。

```
var dsml = system.getParser ("ibmdi.DSML");
dsml.setInputStream ( new java.io.FileInputStream("dirdata.dsml") );
dsml.initParser ();

var entry = dsml.readEntry();
while ( entry != null ) {
 task.dumpEntry ( entry );
 entry = dsml.readEntry();
}
```

関連情報

476 ページの『単純 XML パーサー』,
466 ページの『SOAP パーサー』,
『DSMLv2 パーサー』.

DSMLv2 パーサー

Directory Services Markup Language v1.0 (DSMLv1) を使用すると、ディレクトリー構造情報を XML 文書として表現できます。

DSMLv2 はさらに進歩し、ディレクトリーの照会や更新 (そしてこれらの操作の結果) を XML 文書として表現するメソッドを提供します。DSMLv2 文書は、さまざまな方法で使用できます。IBM Security Directory Integrator には、DSMLv2 の要求メッセージおよび応答メッセージを解析、作成できるパーサーがあります。

DSMLv2 パーサーは、DSMLv2 バッチ要求または DSMLv2 バッチ応答によって初期化されます。項目の読み取りまたは書き込みを行う 1 回の呼び出しごとに、バッチ要求またはバッチ応答に含まれている個々の DSML 要求または応答が解析、作成されます。

このパーサーでは、項目レベルと属性レベルでデルタ・タグがサポートされています。 441 ページの『複数の属性の変更』も参照してください。

モード

DSMLv2 パーサーをサーバー・モードまたはクライアント・モードのいずれかで操作できます。

- サーバー・モードでは、このパーサーは、DSMLv2 要求を読み取って解析し、DSMLv2 応答を書き込んで作成します。
- クライアント・モードでは、このパーサーは、DSMLv2 応答を読み取って解析し、DSMLv2 要求を書き込んで作成します。

操作

ここに提供されているセクションを通じて、DSMLv2 パーサーによってサポートされる操作を表示できます。

DSMLv2 パーサーは、**変更**、**追加**、**削除**、**検索**、**ModifyDN**、**比較**、**認証 (Auth)**、および **拡張 (Extended)** の各操作をサポートしています。

重要: 以下に示す ITIM DSML ライブラリーの IBM Security Directory Integrator 6.0 DSMLv2 パーサー・カスタム・ヘルパー・オブジェクトのサポートは廃止されました。

- dsml.request: すべての要求操作のオブジェクト。
- dsml.response: すべての応答操作のオブジェクト。

これらの属性のいずれかを使用する構成がある場合は、その構成を編集し、これらの属性への参照を削除してください。古いバージョンでロー要求オブジェクトおよび応答オブジェクトを介して使用可能であったデータは、DSMLv2 パーサーにより引き渡されるその他の属性では使用できません。

変更要求

変更要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「modifyRequest」に設定されます。
dsml.base	DSML の modifyRequest エレメントの「dn」という XML 属性を保持します。
$dn	DSML の modifyRequest エレメントの「dn」という XML 属性を保持します。

また、それぞれの変更項目について、DSML の「modification」エレメントの「name」という XML 属性で指定された IBM Security Directory Integrator 属性が作成されます。この属性の値は、DSML の「modification」エレメントで指定された値になります。また、IBM Security Directory Integrator 属性の操作は、DSML の「modification」エレメントの「operation」という XML 属性に従って設定されます。

変更応答

変更応答に対して、パーサーを使用して、ここに記載された構造の項目を解析 (読み取り時) および作成 (書き込み時) できます。

属性	値
dsml.operation	modifyResponse
$dn	DSML の modifyResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。

属性	値
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の addResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

検索要求

検索要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「searchRequest」に設定されます。
$dn	DSML の compareResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.base	DSML の searchRequest エレメントの「dn」という XML 属性を保持します。
dsml.scope	DSML の searchRequest エレメントの「scope」という属性の値を保持します。
dsml.filter	DSML 要求の filter エレメントに対応する LDAP フィルター。
dsml.attributes	この属性の値は、DSML 要求の attributes エレメントにリストされた属性名をエレメントとして保持したベクトルまたはストリング。
dsml.derefAliases	DSML の searchRequest エレメントの「derefAliases」という属性の値を保持します。
dsml.sizeLimit	DSML の searchRequest エレメントの「sizeLimit」という属性の値を保持します。
dsml.timeLimit	DSML の searchRequest エレメントの「timeLimit」という属性の値を保持します。
dsml.typesOnly	DSML の searchRequest エレメントの「typesOnly」という属性の値を保持します。

検索応答

検索応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「searchResponse」に設定されます。
$dn	DSML 応答の「searchResultDone」という DSML エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
resultEntries	多値属性。この属性のそれぞれの値は、対応する「searchResultEntry」エレメントの「attr」エレメントに対応した属性を持つ IBM Security Directory Integrator 項目です。

属性	値
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。

追加要求

追加要求に対してパーサーを使用して、指定された構造の項目を解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「addRequest」に設定されます。
dsml.base	DSML の「addRequest」エレメントの「dn」という XML 属性を保持します。
$dn	DSML の「addRequest」エレメントの「dn」という XML 属性を保持します。

また、DSML の各「attr」エレメントについて、DSML の「attr」エレメントの「name」という XML 属性で指定された IBM Security Directory Integrator 属性が作成されます。この属性の値は、DSML の「attr」エレメントで指定された値になります。

追加応答

追加応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「addResponse」に設定されます。
"$dn	DSML の addResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の addResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

削除要求

削除要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「deleteRequest」に設定されます。
dsml.base	DSML の delRequest エレメントの「dn」という XML 属性を保持します。
$dn	DSML の delRequest エレメントの「dn」という XML 属性を保持します。

削除応答

削除応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「deleteResponse」に設定されます。
$dn	DSML の delRequest エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の addResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

ModifyDN 要求

ModifyDN 要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「modDnRequest」に設定されます。
dsml.base	DSML の modDNRequest エレメントの「dn」という XML 属性を保持します。
$dn	DSML の modDNRequest エレメントの「dn」という XML 属性を保持します。
newrdn	DSML の modDNRequest エレメントの「newrdn」という XML 属性を保持します。
dsml.newSuperior	DSML の modDNRequest エレメントの「newSuperior」という XML 属性を保持します。
dsml.deleteOldRDN	DSML の modDNRequest エレメントの「deleteoldrdn」という XML 属性を保持します。

ModifyDN 応答

ModifyDN 応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「modDnResponse」に設定されます。
$dn	DSML の modDNResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の addResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

比較要求

比較要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「compareRequest」に設定されます。
dsml.base	DSML の compareRequest エレメントの「dn」という XML 属性を保持します。
$dn	DSML の compareRequest エレメントの「dn」という XML 属性を保持します。
dsml.compare_name	DSML 応答の assertion エレメントの「name」という XML 属性を保持します。
dsml.compare_value	DSML 応答の assertion エレメントの値を保持します。

比較応答

比較応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「compareResponse」に設定されます。
$dn	DSML の compareResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.compare_result	比較の結果として一致が見つかったかどうかに応じて「true」または「false」のいずれか。このパーサーを使用して DSML 応答を作成する場合は、この属性が必須です。この属性の値に応じて、パーサーは正しい結果コード値を設定します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。

属性	値
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の addResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

認証要求

認証要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「authRequest」に設定されます。
dsml.principal	DSML の authRequest エレメントの「principal」という XML 属性を保持します。

認証応答

認証応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「authResponse」に設定されます。
$dn	DSML の authResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の authResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。

拡張要求

拡張要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「extendedRequest」に設定されます。
dsml.extended.requestname	DSML の extendedRequest エレメントの「requestName」という XML 属性を保持します。
dsml.extended.requestvalue	DSML の extendedRequest エレメントの「requestValue」という XML 属性を保持します。

拡張応答

拡張応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「extendedResponse」に設定します。
$dn	DSML の extendedResponse エレメントの「matchedDN」という XML 属性を保持します。
dsml.resultcode	DSML 応答の「resultCode」という XML エレメントの「code」という XML 属性を保持します。
dsml.resultdescr	DSML 応答の「resultCode」という XML エレメントの「descr」という XML 属性を保持します。
dsml.error	この属性が存在することはエラー状態を示します。この属性は、DSML 応答の「errorMessage」という XML エレメントの値を保持します。
dsml.exception	DSML 応答の「resultCode」XML エレメントの XML 属性「code」と「descr」にデータを自動的に挿入するために使用される javax.naming.NamingException オブジェクトを保持します。この属性が指定されると、「dsml.resultcode」または「dsml.resultdescr」項目属性に設定されている値はすべて無視され、これらの値は例外オブジェクトを使用して取得されたデータに置換されます。
dsml.referral	DSML の extendedResponse エレメントのすべての参照 URI が格納されているベクトルを保持します。
dsml.responseName	DSML の extendedResponse エレメントの「responseName」という XML 属性を保持します。
dsml.response	「extendedResponse」操作からの応答を表す文字列が格納されているバイト配列を保持します。

注: すべての無効な XML 文字 (XML 仕様による) は、この属性が DSML に直列化する前に「dsml.error」項目属性から削除されます。

エラー応答

エラー応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
dsml.operation	「errorResponse」に設定されます。

属性	値
dsml.errorType	DSML 応答の XML エレメントの「type」という XML 属性の値を保持します。「notAttempted」、 「couldNotConnect」、 「connectionClosed」、「malformedRequest」、 「gatewayInternalError」、「authenticationFailed」、「unresolvableURI」、 または「other」のいずれかである必要があります。
dsml.message	DSML 応答の「message」という XML エレメントのテキスト値を保持します。
dsml.details	DSML 応答の「detail」という XML エレメントの値を保持します。

バイナリー属性とストリング以外の属性

DSMLv2 パーサーで作業をするときは、ここに記載されている点について注意が必要です。

DSML メッセージの解析時に、パーサーの「**バイナリー属性**」パラメーターによってバイナリーとしてタグ付けされた属性は、Base64 でデコードされます。つまり、DSML メッセージのストリング値は Base64 でデコードされて Java のバイト配列に入れられます。

DSML メッセージの作成時に、値が Java バイト配列である属性はすべて、DSML メッセージに書き込まれる前に、Base64 でエンコードされてストリングに変換されます。

DSML メッセージの作成時に、値のタイプがストリングでも Java バイト配列でもない属性が渡されると、その値はオブジェクトの toString() メソッドの呼び出しによってストリングに変換され、そのストリング値が DSML メッセージに書き込まれます。

オプション属性

ここに示すオプション属性が指定されている場合、パーサーはすべての DSMLv2 要求および応答でこれらの属性を解析 (読み取り時) および作成 (書き込み時) します。

属性	値
dsml.requestID	DSMLv2 の requestID 属性に対応します。
dsml.controls	ロー `com.ibm.ldap.dsml.DsmlControl` オブジェクトのベクトルを保持します。

Base64 エンコードであると想定される DSMLv2 コントロール

読み取り時に、パーサーは DSMLv2 コントロールの値が Base64 エンコードであると予期します。ここに記載されているコード例を使用して、DSMLv2 コントロールを操作できます。

例えば、以下のようなコントロール・エレメントではなく、

```
<control type="1.2.840.113556.1.4.619" criticality="true">
 <controlValue xsi:type="xsd:string">mycontrolvalue</controlValue>
</control>
```

以下のようなコントロール・エレメントを提供する必要があります。

```
<control type="1.2.840.113556.1.4.619" criticality="true">
 <controlValue xsi:type="xsd:base64Binary">bXljb250cm9sdmFsdWU==</controlValue>
</control>
```

これは、IBM Security Directory Integrator (com.ibm.ldap.dsml.*) の基本 DSML ライブラリーの制約です。DSMLv2 XML スキーマ (http://www.oasis-open.org/committees/dsml/docs/DSMLv2.xsd) により、controlValue エレメントを xsd:anyType にすることができます。ただし、IBM DS DSMLv2 ライブラリー (IBM Security Directory Server 6.1 から取得) では xsi:type 属性が無視され、値の base64 デコードが常に試行されます。

結果コードおよび結果記述の設定

ここに記載されている情報を使用して、結果コードおよび結果記述を設定することができます。

DSML 応答メッセージの dsml.resultcode 属性を設定するときに利用できるタイプは、java.lang.Integer と、ストリング値として整数が入っている java.lang.String です。この値は、DSML の resultCode エレメントの「code」という XML 属性 (整数値) に対応します。DSMLv2 仕様では、この属性は必須です。

オプションとして、dsml.resultdescr 属性を DSML 応答メッセージに設定することもできます。この値は、DSML の resultCode エレメントの「descr」という XML 属性に対応します。DSMLv2 仕様では、この属性は必須ではありません。この属性に値を割り当てると、その値がそのまま DSML 応答に配置されます。値の検証は実行されず (列挙型ストリングでは実行される)、必須の整数属性 dsml.resultcode とこの値が対応しているかどうかの検査も実行されません。

DSML の resultCode エレメントの XML 属性「code」と「descr」も、DSML 応答メッセージの dsml.exception 項目属性によって設定できます。この属性は javax.naming. NamingException オブジェクトのみを受け入れます。dsml.exception 属性が設定されている場合は、DSML の resultCode エレメントの XML 属性「code」と「descr」も、例外オブジェクトから抽出された新規の値により上書きされます。例えば dsml.exception 属性に javax.naming.AuthenticationException オブジェクトが設定されている場合は、「code」属性は LDAP コード 49 に、「descr」属性は LDAP 記述「inappropriateAuthentication」に設定されます。

複数の属性の変更

情報およびデータ・フロー・ルールを使用して、複数の属性の変更を処理できます。

DSMLv2 パーサー (および LDIF パーサー) では、1 つの属性に対する複数変更はサポートされていません。変更の値は属性に累積され、最終変更の操作が属性の操作タグとして設定されます。したがってパーサーは、変更を項目にマージする必要があります。このとき、マージされた属性変更が、変更操作における属性の変更と同等になるようにします。これを行うには、属性レベルでの IBM Security Directory Integrator 固有のタグである `Attribute.ATTRIBUTE_MOD` と、AttributeValue レベル・タグである `AttributeValue.AV_ADD`、`AttributeValue.AV_DELETE` を使用します。

属性オブジェクトで変更を累積する際のデータ・フロー・ルールを以下に示します。

- 「追加」変更では、AttributeValue.AV_ADD によって値が属性に追加されます。また、属性にまだ Attribute.ATTRIBUTE_REPLACE タグが付いていない場合には、Attribute.ATTRIBUTE_MOD というタグが付けられます。
- 以前の変更においてこの属性に既に Attribute.ATTRIBUTE_REPLACE というタグが付けられている場合は、このタグは変更されません。
- 「削除」変更では、AttributeValue.AV_DELETE によって値が属性に追加されます。また、属性にまだ Attribute.ATTRIBUTE_REPLACE タグが付いていない場合には、Attribute.ATTRIBUTE_MOD というタグが付けられます。「削除」変更で属性のそれぞれの値について Attribute.ATTRIBUTE_REPLACE というタグが属性に付けられている場合は、属性にその値が含まれていると、その値が属性から除去されます。
- 値がない「削除」変更では、以前の変更の属性値がクリアされ、属性に Attribute.ATTRIBUTE_REPLACE というタグが付けられます。
- 「置換」変更では、以前の変更の属性値がクリアされ、新規の値が追加されます。また、属性に Attribute.ATTRIBUTE_REPLACE というタグが付けられます。

構成

ここに記載されているパラメーターを使用することで、DSMLv2 パーサーを構成できるようになります。

このパーサーは、以下のパラメーターを必要とします。

文字エンコード

XML 文字エンコード (UTF-8、ASCII など) を指定します。

このパーサーは単純 XML パーサーを拡張したものであるため、文字エンコードに関して同じ注意事項が適用されます。

モード パーサーをサーバー・モードとクライアント・モードのいずれで実行するかを指定します。指定できる値は「サーバー」および「クライアント」です。サーバー・モードでは、要求が読み取られ、応答が書き込まれます。クライアント・モードでは、要求が書き込まれ、応答が読み取られます。

バイナリー属性

パーサーにバイナリー属性として処理させる属性を、コンマで区切ったリストとして指定します。

デフォルトでは、以下の属性がバイナリーとして指定されています (このリストは自分で変更できます)。

- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedData
- thumbnailPhoto
- thumbnailLogo
- userPassword
- userCertificate

- authorityRevocationList
- certificateRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- objectGUID
- objectSid

エラー発生時

BatchRequest エレメントには XML 属性 onError を格納できます。この属性は、要求エレメント処理中のサーバーによる障害への対応を決定します。有効な値は「終了」と「再開」です。デフォルト値は exit です。

処理 バッチ要求の「processing」DSML 属性の値を設定します。

応答順序

サーバーによる BatchResponse 内の個々の応答の順序付け方法を指定します。このパラメーターの値は「順次」と「順不同」です。デフォルト値は「順次」です。「応答順序」の値が「順次」に設定されている場合、サーバーは、個々の応答がそれぞれの要求に対応した順序になっている BatchResponse を戻す必要があります。

XML 宣言の省略

XML 宣言の省略を使用可能にするか使用不可にするかを指定します。デフォルトでは、このパラメーターは使用不可です。

インデント出力

チェックされている場合は、ステートメント行の深さに基づいて出力がインデントされます。これは形式的なもののみであり、出力ファイルの内容のセマンティックには影響しません。

SOAP バインディング

オンにすると、パーサーが SOAP DSML メッセージを処理および作成します。それ以外の場合は、DSML メッセージは SOAP によりラップされません。

詳細ログ

チェックすると、より詳細なログ・メッセージが生成されます。

例

このセクションに記載されている例を使用することで、DSMLv2 パーサーについての理解を深めることができます。

サーバー・モードでの DSMLv2 AddRequest の構文解析

DSMLv2 パーサーが「サーバー」(読み取り) モードで稼働するよう構成され、ここに記載されている DSMLv2 要求が DSMLv2 パーサーに渡される場合、項目オブジェクトを生成します。

```
<batchRequest onError="exit" processing="sequential"
    responseOrder="sequential" xmlns="urn:oasis:names:tc:DSML:2:0:core"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <addRequest requestID = "3" dn="cn=chavdar kovachev,o=ibm,c=us">
        <attr name="objectclass">
            <value>person</value>
        </attr>
        <attr name="telephoneNumber">
```

```
            <value>555</value>
        </attr>
        <attr name="sn">
            <value>kovachev</value>
        </attr>
        <attr name="cn">
            <value>chavdar kovachev</value>
        </attr>
    </addRequest>
</batchRequest>
```

以下の属性を持つ項目オブジェクト。

- sn: 'kovachev'
- $dn: 'cn=chavdar kovachev,o=ibm,c=us'
- telephoneNumber: '555'
- objectclass: 'person'
- dsml.operation: 'addRequest'
- dsml.requestID: '3'
- cn: 'chavdar kovachev'
- dsml.base: 'cn=chavdar kovachev,o=ibm,c=us'

クライアント・モードでの DSMLv2 SearchRequest の作成

DSMLv2 パーサーが「クライアント」(書き込み) モードで稼働するよう構成され、ここに記載されている属性を持つ項目がパーサーに渡される場合、項目を生成します。

- $dn: "o=ibm,c=us"
- dsml.derefAliases: 'neverDerefAliases'
- dsml.sizeLimit: '0'
- dsml.operation: 'searchRequest'
- dsml.timeLimit: '0'
- dsml.typesOnly: 'false'
- dsml.requestID: '7'
- dsml.attributes: '[cn, sn]'
- dsml.scope: 'wholeSubtree'
- dsml.base: 'o=ibm,c=us'
- dsml.filter: '(sn=*)'

DSMLv2 要求:

```
<?xml version="1.0" encoding="UTF-8"?>
<batchRequest onError="exit" processing="sequential"
    responseOrder="sequential" xmlns="urn:oasis:names:tc:DSML:2:0:core"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <searchRequest requestID="7" derefAliases="neverDerefAliases"
        dn="o=ibm,c=us" scope="wholeSubtree" sizeLimit="0"
        timeLimit="0" typesOnly="false">
       <filter>
            <present name="sn"/>
       </filter>
       <attributes>
            <attribute name="cn"/>
            <attribute name="sn"/>
       </attributes>
    </searchRequest>
</batchRequest>
```

固定レコード・パーサー

固定レコード・パーサーは、固定長のテキスト・レコードの読み取りおよび書き込みを行います。ここに記載されているパラメーターを使用することで、固定レコード・パーサーを構成できるようになります。

構成

このパーサーは、以下のパラメーターを必要とします。

列記述 これは複数行パラメーターであり、それぞれにフィールド名、オフセット、および長さを指定します。例を示します。

```
field1, 1, 12
field2, 13, 4
field3, 17, 3
```

これらのフィールド名は、スキーマ・ディスカバリーが実行されると表示されます。

注: オフセットは 1 で開始されます。0 などの無効な値は例外を引き起こす場合があります。

値のトリム
チェックすると、読み取られるフィールドから先頭および末尾のスペースが除去されます。

文字エンコード
使用される文字エンコード。 423 ページの『文字エンコード変換』も参照してください。

詳細ログ
このフィールドをチェックすると、追加のログ・メッセージが生成されます。

JSON パーサー

JSON パーサーを使用して、JavaScript Object Notation (JSON) フォーマットで項目の読み取りおよび書き込みを行うことができます。

JSON は、軽量のデータ交換フォーマットであり、JavaScript プログラミング言語のサブセットです。 JSON は、以下の 2 つの構造を使用して作成されます。

- 値の番号付きリスト (配列)
- 名前/値ペアのコレクション (オブジェクト)

JSON フォーマットについて詳しくは、http://json.org を参照してください。

パーサーの使用

JSON パーサーは、JSON フォーマットのデータの読み取りおよび書き込みを行います。ここに記載されている情報を通じて、JSON パーサーの使用方法を理解することができます。

IBM JavaScript エンジンの JSON フレームワークが、Java オブジェクトと JSON オブジェクトのマッピングに使用されます。JSON フレームワークの `JsonFactory`

インスタンスは、IBM Security Directory Integrator の項目オブジェクトまたは属性オブジェクトと、基本的な JSON 型 (オブジェクト、ストリング、配列、数値、ブール値など) との間のマッピングを実行します。

JSON 型の項目または属性へのマッピング

オブジェクト

- オブジェクトは、項目または属性にマップされます。親オブジェクトが NULL の場合は、項目が使用されます。親オブジェクトが NULL ではない場合は、属性が使用されます。階層内の最上位のオブジェクトは、項目にマップされます。
- 属性ベースのオブジェクトには、名前付き属性が含まれます。親オブジェクトの値は属性である必要があり、オブジェクトのプロパティー名および値を表します。
- 項目ベースのオブジェクトには、属性ベースのオブジェクトと同様に、名前付き属性が含まれます。

配列

- 配列は、属性の値にマップされます。配列内の値は、属性の値として現在のターゲットに追加されます。
- 値 (例えば、ストリング、数値、ブール値) およびオブジェクトが含まれた配列は、単純な値および属性オブジェクトが含まれた属性になり、配列内の無名オブジェクトを表します。

項目または属性の JSON 型へのマッピング

項目 項目はオブジェクトにマップされ、各属性はオブジェクト内のプロパティーを表します。

属性 属性は配列またはオブジェクトになります。属性に含まれている内容により、以下のようになります。

- 属性オブジェクトと他のオブジェクト (例えば、ストリングや日付) の両方が含まれている場合は、属性は配列になります。
- すべての子属性に同じ名前が付いている属性オブジェクトが含まれている場合は、属性はオブジェクトの配列になります。

JSON でのデータ・マッピングは対称的です。マップされて書き出された JSON データは、同じ結果になります。ただし、プロパティーの順序は、読み取られるデータと完全には一致しないことがあります。

例

ここに示す JSON データ例では、単純な値、配列、およびオブジェクトがネストされています。

```
{
 "id": "0001",
 "type": "donut",
 "name": "Cake",
 "ppu": 1.55,
 "batters":
  {
   "batter":
    [
     { "id": "1001", "type": "Regular" },
     { "id": "1002", "type": "Chocolate" },
```

```
    { "id": "1003", "type": "Blueberry" },
    { "id": "1004", "type": "Devil's Food" }
   ]
  },
 "topping":
  [
   { "id": "5001", "type": "None" },
   { "id": "5002", "type": "Glazed" },
   { "id": "5005", "type": "Sugar" },
   { "id": "5007", "type": "Powdered Sugar" },
   { "id": "5006", "type": "Chocolate with Sprinkles" },
   { "id": "5003", "type": "Chocolate" },
   { "id": "5004", "type": "Maple" }
  ],
 "simplearray":
  [
   "first value",
   "second value",
   "third value"
  ]
}
```

上記の JSON データは、以下の項目構造にマップされます。

```
Entry.toDeltaString()

{
 "#type": "generic",
 "#count": 8,
 "batters": [
  "#type": "replace",
  "#count": 1,
  "#replace": ""batters": {
 "batter": {
  "batter": {
   "id": "1001",
   "type": "Regular"
  },
  "batter": {
   "id": "1002",
   "type": "Chocolate"
  },
  "batter": {
   "id": "1003",
   "type": "Blueberry"
  },
  "batter": {
   "id": "1004",
   "type": "Devil's Food"
  }
 }
}"
 ],
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "0001"
 ],
 "name": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Cake"
 ],
 "ppu": [
  "#type": "replace",
  "#count": 1,
  "#replace": 1.55
 ],
 "topping": {
  "#type": "replace",
  "#count": 0,
  "topping": {
   "#type": "replace",
   "#count": 0,
   "id": [
    "#type": "replace",
    "#count": 1,
    "#replace": "5001"
   ],
```

```
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "None"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "5002"
 ],
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Glazed"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "5005"
 ],
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Sugar"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "5007"
 ],
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Powdered Sugar"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "5006"
 ],
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Chocolate with Sprinkles"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
  "#type": "replace",
  "#count": 1,
  "#replace": "5003"
 ],
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "Chocolate"
 ]
},
"topping": {
 "#type": "replace",
 "#count": 0,
 "id": [
```

```
    "#type": "replace",
    "#count": 1,
    "#replace": "5004"
   ],
   "type": [
    "#type": "replace",
    "#count": 1,
    "#replace": "Maple"
   ]
  }
 },
 "type": [
  "#type": "replace",
  "#count": 1,
  "#replace": "donut"
 ],
 "simplearray": [
  "#type": "replace",
  "#count": 3,
  "#replace": "first value",
  "#replace": "second value",
  "#replace": "third value"
 ]
}
```

構成

ここに記載されているパラメーターを使用することで、JSON パーサーを構成できるようになります。

圧縮出力

このパラメーターを使用して、圧縮モードでデータを表示します。圧縮モードの場合、フォーマットされていない単一行に JSON データが書き込まれます。これは、デフォルト・モードです。

文字エンコード

このパラメーターを使用して、データの読み取りまたは書き込みの際に使用する文字エンコードを指定します。

詳細ログ

このパラメーターを使用して、詳細なログ・メッセージを生成します。

コメント

このパラメーターを使用して、コメントを追加します。データの構文解析時には、このコメントは無視されます。

HTTP パーサー

HTTP パーサーは、HTTP 仕様に従ってバイト・ストリームを解釈します。ここに提供されているリンクを使用することで、これについて詳しく知ることができます。

このパーサーは、HTTP クライアント・コネクターおよび HTTP サーバー・コネクターにより使用されます。

構成

ここに記載されているパラメーターを使用することで、HTTP パーサーを構成できるようになります。

プロパティーとしてのヘッダー
設定した場合は、ヘッダー値は**プロパティーとして取得**され、**プロパティーとして設定**されます。設定しない場合は、ヘッダー値は**属性として読み取られ**、**属性として戻され**ます。

クライアント・モード
設定した場合は、パーサーはクライアント HTTP 応答モードで動作します。設定しない場合は、パーサーはサーバー・モードで動作します。この設定は、パーサーが出力ストリームを書き込んでいる場合にのみ関係します。

文字エンコード
使用される文字エンコード。 456 ページの『文字セット/エンコード』も参照してください。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

スキーマ

ここに記載されているスキーマおよび対応する属性を使用して、HTTP パーサーを構成できます。

HTTP パーサーは、以下の属性を *作業項目* (入力属性マップおよび出力属性マップ) に設定します。構成パラメーター「**プロパティーとしてのヘッダー**」が使用可能になっている場合、以下で説明するすべての属性は項目プロパティーとして構成されるため、このスキーマは役に立ちません。

http.method
Request-URI によって識別されるリソース上で実行されるメソッド。このメソッドでは大/小文字が区別されます (デフォルトは **GET**)。HTTP メソッドについて詳しくは、http://www.w3.org/Protocols/HTTP/Methods.html を参照してください。

http.base
要求が適用されるリソースを識別する URI。

http.responseCode
要求を把握し対応するための試行に関する 3 桁の整数による結果コード。この属性またはプロパティーは、クライアント・モードでは必須です。

http.responseMsg
短いテキストによる応答コードの説明。この属性またはプロパティーは、クライアント・モードでは必須です。

http.body
メッセージ本体。要求または応答メッセージに関連付けられたエンティティー本体を運ぶために使用されます。メッセージ本体は、**http.Transfer-Encoding** ヘッダー・フィールドで示されているように、転送コーディングが適用された場合にのみ、エンティティー本体と異なります。読み取りの際、このオブジェクトは、データのコンテンツ・タイプに応じて、java.lang.StringBuffer のインスタンス、char[]、または byte[] となります。

http.bodyAsString

`http.bodyAsString` 属性は、メッセージの本体をストリング・フォーマットで返すために使用します。メッセージからストリング文字への変換は、コンテンツ・タイプの文字エンコードを使用してエンコードされます。

http.bodyAsBytes

`http.bodyAsBytes` 属性は、メッセージの本体をバイトで返すために使用します。

http.url

使用する URL。この属性またはプロパティーは、クライアント・モードでは必須です。

http.remote_user

ユーザー名 (要求メッセージの **http.Authorization** ヘッダーのフィールドに存在する場合)。

http.remote_pass

パスワード (要求メッセージの **http.Authorization** ヘッダーのフィールドに存在する場合)。

http.status

サーバー・モードでの書き込みの場合に使用されます。 デフォルトは **200 OK** です。HTTP 応答メッセージの状況表示行の作成に使用されます (http://tools.ietf.org/html/rfc2616#section-6.1 を参照)。HTTP 応答の Status-Code (3 桁の数字)、およびシングル・スペース文字で区切られた HTTP 応答の Reason-Phrase が含まれている必要があります。例えば、「201 Created」となります。別の方法として、以下の事前定義値のいずれかを使用することもできます。

- **OK** または **200 OK**: 200 OK 応答を戻します。
- **FORBIDDEN** または **401 Forbidden**: 401 Forbidden 応答を戻します。この応答は、**http.auth-realm** 属性またはプロパティーを使用します。
- **NOT FOUND** または **404 File Not Found**: 404 File Not Found 応答を戻します。

http.auth-realm

追加認証を要求するときに使用されます。 デフォルト値は **IBM-Directory-Integrator** です。

http.redirect

サーバー・モードで書き込みを行っているときに、この属性またはプロパティーに値が存在する場合は、その値を指し示すリダイレクト・メッセージが送信されます。

http.qs.*

サーバー・モードで読み取りを行っているときの照会ストリングのパート。キーは、**http.qs** の後に続く名前のパートです。この値は属性またはプロパティーに含まれます。

http.* **http.** で始まるその他のすべての属性またはプロパティーは、書き込み時にヘッダー行を生成するために使用します。読み取りの際には、ヘッダーは、名前が **http.** で始まりその後にヘッダーの名前が続く属性またはプロパティーに入れられます。

一般ヘッダーのフィールド

一般ヘッダー・フィールドのリストをここで見ることができます。

http.Cache-Control
要求/応答チェーン上のすべてのキャッシング・メカニズムが従う必要があるディレクティブを指定するときに使用します。

http.Connection
送信者は特定の接続に対して希望するオプションを選択できます。以降の接続でプロキシーによる通信を行うことはできません。

http.Date
メッセージが発信された日時を表します。フィールド値は HTTP 日付で、次の形式になります: 1*2DIGIT month 2*4DIGIT。

http.Pragma
要求/応答チェーン上のすべての受信者に適用されるインプリメンテーション固有のディレクティブを組み込むときに使用します。すべてのプラグマ・ディレクティブが、プロトコルの視点からオプションの動作を指定します。

http.Trailer
特定のヘッダーのフィールド・セットが、チャンク転送コーディングでエンコードされたメッセージのトレーラーに存在することを示します。

http.Transfer-Encoding
送信側と受信側の間で安全にメッセージ本体を転送するために、メッセージ本体に適用された変換のタイプ (存在する場合) を指定します。これは、転送コーディングがエンティティーではなくメッセージのプロパティーである場合、コンテンツ・コーディングとは異なります。

http.Upgrade
クライアントがサポートする追加の通信プロトコルを指定し、プロトコルの切り替えが適切であるとサーバーが判断した場合に使用できるようにします。このフィールドは 101 コード (Switching Protocols) 内で使用されます。

http.Via
クライアントがサポートする追加の通信プロトコルを指定し、プロトコルの切り替えが適切であるとサーバーが判断した場合に使用できるようにします。このフィールドは 101 コード (Switching Protocols) 内で使用されます。

http.Warning
メッセージに反映されていない場合がある、メッセージの状況または変換に関する追加情報を送るために使用します。次のような形式になります: `3DIGIT-warn-code SP warn-agent SP warn-text [SP warn-date]`。

エンティティー・ヘッダーのフィールド

エンティティー・ヘッダー・フィールドのリストをここで見ることができます。

http.Allow
Request-URI によって識別されるリソースがサポートするメソッド・セットのリスト。このフィールドの目的は、リソースに関連付けられている有効な

メソッドを受信者に伝えることに限定されます。Allow ヘッダーのフィールドは 405 (Method Not Allowed) 応答に存在します。

http.content-encoding
メディア・タイプの修飾子として使用します。存在する場合、その値はエンティティー本体に適用された追加のコンテンツ・コーディングを示します。これにより、http.Content-Type フィールドで参照されるメディア・タイプを取得するために適用する必要があるデコード・メカニズムも示されます。

http.Content-Language
囲まれたエンティティーに関する対象者の自然言語を示します。エンティティー本体で使用されているすべての言語と等しいとは限らないことに注意してください。

http.content-length
受信者に送信されるエンティティー本体のサイズ (オクテットの 10 進数)、または HEAD メソッドの場合は要求が GET であるときに送信されたエンティティー本体のサイズを示します。この属性またはプロパティーは、読み取り時には戻され、書き込み時には無視されます。これはパーサーによって再計算されます。

http.Content-Location
メッセージで囲まれたエンティティーが、要求されたリソースの URI (絶対 URI または相対 URI) とは別の場所からアクセス可能な場合に、そのエンティティーのリソース・ロケーションを提供するために使用することがあります。

http.Content-MD5
エンティティー本体のエンドツーエンドのメッセージ整合性チェック (MIC) を提供するための、エンティティー本体の MD5 ダイジェストです。

http.Content-Range
エンティティー本体の一部分とともに送信して、その部分をどの完全なエンティティー本体に適用する必要があるかを指定します。

http.content-type
受信者に送信されるエンティティー本体のメディア・タイプ、または HEAD メソッドの場合は要求が GET であるときに送信されたメディア・タイプを示します。

http.Expires
応答が失効したと見なされる日時を指定します。

http.Last-Modified
起点サーバーが認識しているバリアントの最終変更日時を示します。形式は HTTP 日付です。

要求ヘッダーのフィールド

要求ヘッダー・フィールドのリストをここで見ることができます。

http.Accept
応答で受け入れ可能な目的のメディア・タイプ・セットを指定するために使用します。

http.Accept-Charset
応答で受け入れ可能な文字セットを指定するために使用します。

http.Accept-Encoding
応答で受け入れ可能なコンテンツ・コーディングを指定するために使用します。

http.Accept-Language
要求への応答として優先される自然言語のセットを指定するために使用します。

http.authorization
要求されるリソースのレルムに関するユーザー・エージェントの認証情報を含む信任状で構成されています。

http.Expect
クライアントで必要となる特別なサーバー動作を指定するために使用します。

http.From
指定した場合、ユーザー・エージェントの要求を制御する人物 (ユーザー) のインターネット E メール・アドレスが含まれています。

http.Host
ユーザーまたは参照リソースが提供する元の URI (通常は HTTP URL) から取得された、要求されるリソースのインターネット・ホストとポート番号を指定します。ホストのフィールド値は、元の URL によって指定される起点サーバーまたはゲートウェイの命名機関を表している必要があります。

http.If-Match
条件付きとするメソッドとともに使用します。前にリソースから取得した 1 つ以上のエンティティーを持つクライアントは、関連付けられたエンティティーのタグのリストを If-Match ヘッダー・フィールドに格納することで、いずれかのエンティティーがカレントであることを確認できます。この機能の目的は、最小限のトランザクション・オーバーヘッドで、キャッシュされた情報の効率的な更新を可能にすることです。更新要求の場合にも、正しくないバージョンのリソースを誤って修正することを防止するために使用できます。特別な場合として、値「*」はリソースのあらゆるカレント・エンティティーに一致します。

http.If-Modified-Since
条件付きとするメソッドとともに使用します。要求されたバリアントが、このフィールドで指定された時刻以降変更されていない場合、エンティティーはサーバーから戻されず、代わりに 304 (not modified) 応答がメッセージ本体なしで戻されます。形式は HTTP 日付です。

http.If-None-Match
条件付きとするメソッドとともに使用します。前にリソースから取得した 1 つ以上のエンティティーを持つクライアントは、関連付けられたエンティティーのタグのリストを If-None-Match ヘッダー・フィールドに格納することで、いずれのエンティティーもカレントでないことを確認できます。

http.If-Range
クライアントのキャッシュにエンティティーの部分コピーが含まれており、

エンティティー全体の最新コピーをキャッシュに格納したい場合は、条件付き GET とともに Range 要求ヘッダーを使用することができます。要求されたエンティティーが変更されていない場合は、クライアントで欠けている部分が送信されます。変更されている場合は、新しいエンティティー全体が送信されます。HTTP 日付が含まれている場合もあります。

http.If-Unmodified-Since
条件付きとするメソッドとともに使用します。要求されたリソースが、このフィールドで指定した時刻以降変更されていない場合、サーバーは If-Unmodified-Since ヘッダーが存在していないかのように要求された操作を実行します。要求されたバリアントが指定された時刻以降に変更されている場合、サーバーは要求された操作を実行せず、412 コード (Precondition Failed) を戻します。形式は HTTP 日付です。

http.Max-Forwards
要求を次のインバウンド・サーバーに転送できるプロキシーまたはゲートウェイの数を制限するために、TRACE メソッドと OPTIONS メソッドを使用したメカニズムを提供します。

http.Proxy-Authorization
クライアントが、認証を必要とするプロキシーに対してクライアント自体やそのユーザーを識別するための情報を提供できます。要求されるリソースのプロキシーおよび/またはレルムのユーザー・エージェントの認証情報を含む信任状で構成されています。

http.Range
HTTP 要求 (GET メソッドを使用) から戻された結果エンティティーのどの範囲 (バイト) が受信されるかを指定します。

http.Referer
クライアントは、サーバーの便宜を図るために、Request-URI の取得元のリソースのアドレス (URI) を指定することができます。

http.TE
どの拡張転送コーディングを応答で受け取るか、およびチャンク転送コーディングのトレーラー・フィールドを受け入れるかどうかを指定します。

http.User-Agent
要求を発信したユーザー・エージェントに関する情報が含まれています。

応答ヘッダーのフィールド

応答ヘッダー・フィールドのリストをここで見ることができます。

http.Accept-Ranges
サーバーがリソースの範囲要求を受け入れることを示しますが、このフィールドが欠落していても受け入れます。

http.Age
発信元のサーバーでの応答 (または再確認) の生成後に経過した時間数の送信側による見積もりを伝えます。

http.ETag
要求されたバリアントのエンティティー・タグの現行値を提供します。このエンティティー・タグは、同じリソースの他のエンティティーとの比較に使用される場合があります。

http.Location
要求の完了または新しいリソースの識別のために、Request-URI 以外のロケーションに受信者をリダイレクトするために使用します。このフィールドの値は、単一の絶対 URI で構成されています。

http.Proxy-Authenticate
407 (Proxy Authentication Required) 応答の一部として含まれています。このフィールドの値は、認証方式を示すチャレンジ、および Request-URI のプロキシーに適用可能なパラメーターで構成されています。

http.Retry-After
503 (Service Unavailable) 応答とともに使用して、要求したクライアントがサービスを使用できない予測期間を示すことができます。このフィールドをいずれかの 3xx (リダイレクト) 応答とともに使用して、リダイレクト要求を発行する前にユーザー・エージェントに待機を依頼する最小時間を示すこともできます。このフィールドの値は、HTTP 日付、または応答時間後の秒数 (10 進整数) のいずれかになります。

http.Server
要求を処理するために発信元サーバーが使用するソフトウェアに関する情報が含まれています。

http.Vary
完全に識別された要求ヘッダー・フィールドのセットを示し、応答が新鮮な間は、以降の要求に応答するために再確認せずにその応答をキャッシュが使用できるかどうかを示します。キャッシュできない応答や失効した応答の場合、Vary フィールドの値によって、表現の選択に使用された基準に関する情報がユーザー・エージェントに通知されます。

http.WWW-Authenticate
401 (Unauthorized) 応答メッセージに含まれています。このフィールドの値は、Request-URI に適用可能な認証方式およびパラメーターを示す、少なくとも 1 つのチャレンジで構成されています。

文字セット/エンコード

ここに記載されている情報を通じて、文字セット/エンコードの扱い方について知ることができます。

http.body 属性が java.io.File オブジェクトである場合は、そのファイルがそのまま送信され、文字変換は実行されません。

文字セットについて詳しくは、 423 ページの『文字エンコード変換』も参照してください。

読み取り時の文字セット

読み取り時のデフォルトの文字セット・エンコードは **iso-8859–1** です。このエンコードよりも、このコネクターの構成ペインにある「**文字エンコード**」パラメーター

の方が優先されます。また、その characterSet パラメーターよりも、「Content-type: text/plain; charset=iso-8859–1」というタイプのヘッダーの方が優先されます。最適なパフォーマンスと互換性を得るためには、このヘッダーを設定する必要があります。

送信時の文字セット

読み取り時のデフォルトの文字セット・エンコードは **iso-8859–1** です。このエンコードよりも、このコネクターの構成ペインにある「**文字エンコード**」パラメーターの方が優先されます。テキスト・メッセージを送信するときには、送信する項目に「http.content-type」という名前の属性を組み込み、「Content-type: text/plain; charset=iso-8859-1」という形式のテキスト値を設定しておく必要があります。デフォルトが使用されるのは、この属性が存在しない場合のみです。

HTTP Cookie を使用する方法

HTTP Cookie は HTTP ヘッダーであり、その構文は HTTP State Management Mechanism 標準 (RFC 2109、RFC 2965) に準拠しています。ここに記載されている情報と例を使用することで、HTTP Cookie の使用について知ることができます。

IBM Security Directory Integrator の HTTP コンポーネントは、Cookie ヘッダーの特別な処理は実行しません。Cookie を使用する場合は、各 Cookie ヘッダーの内容を自分で解釈する必要があります。

Cookie を HTTP 応答に設定するには、「Set-cookie」HTTP ヘッダーを使用します。例を示します。

```
work.setAttribute("http.Set-Cookie", "myname=myvalue; expires=Sat, 15-Jan-2011 13:23:56 GMT;
      path=/; domain=.ibm.com");
```

Cookie を HTTP 要求に設定するには、「Cookie」HTTP ヘッダーを使用します。例を示します。

```
work.setAttribute("http.Cookie", "myname=myvalue; myname2=myvalue2");
```

関連情報

423 ページの『文字エンコード変換』
137 ページの『HTTP クライアント・コネクター』,
142 ページの『HTTP サーバー・コネクター』.

LDIF パーサー

LDIF 形式を使用して、ディレクトリー情報、またはディレクトリー項目に加えた一連の変更の説明を伝達することができます。

LDIF ファイルは、行の分離文字で区切られた一連のレコードで構成されています。レコードは、ディレクトリー項目を説明する連続する行、またはディレクトリー項目に加えた一連の変更を説明する連続した行で構成されています。LDIF ファイルでは、一連のディレクトリー項目、およびディレクトリー項目に適用された一連の変更のいずれか (両方ではありません) を指定します。

ディレクトリーを変更する LDAP 操作 (add、delete、modify、moddn、および modrdn) と、LDIF 形式で記述された変更レコードのタイプ

(「add」、「delete」、「modify」、「moddn」、および「modrdn」) との間には 1 対 1 の相関関係があります。これは意図的に行われている対応であり、これにより、LDIF 変更レコードからプロトコル操作への直接変換が可能になります。

LDIF パーサーは、LDIF スタイルのデータの読み取りおよび書き込みを行います。LDIF パーサーは通常、LDAP ディレクトリーとの間でファイル交換を行うために使用されます。

LDIF パーサーは、MIME BASE64 でエンコードされたストリングを正しく解析し、書き込みます。そして、必要に応じて、BASE64 エンコードを行います。例えば、属性値の後に末尾のスペースがある場合、このパーサーは、他の LDIF パーサーがそのスペースを確実に取得できるように、BASE64 で属性をエンコードします。

注: 準拠する LDIF ファイルは、常に**文字エンコード**が UTF-8 に設定されていることが必要です。**文字エンコード**・パラメーターは、BASE64 エンコード・ストリングをエンコードまたはデコードするときにも適用されます。

BASE64 エンコードは、デコードの方法を知らない限り、文字化けしたテキストのように見えます。

このパーサーでは、項目レベル、属性レベル、および属性値レベルでデルタ・タグと互換性のあるタグが処理/提供されます。属性レベルでのデルタ・タグは DSMLv2 パーサーにより処理されます。 441 ページの『複数の属性の変更』を参照してください。

LDIF パーサーは「newrdn」属性が存在するかどうかを writeEntry メソッドで検出し、存在する場合は、変更タイプを「modify」ではなく「modrdn」に設定します。特定のコネクターが「modrdn」操作を処理する方法については、 245 ページの『modrdn 操作の検出と処理』 も参照してください。

LDIF 入力の読み取り

LDIF パーサーを使用する場合、読み取りの際に入力の行が 1 つずつ読み取られ、各入力に対してここに記載された確認が行われます。

- 「dn」キーが読み取られた場合、この鍵が構成済みの「dnAttributeName」属性の値に設定されている。
- 属性に「:」で始まる値が含まれている場合、指定されたエンコードを使用してバイト配列として読み取られている。

キー「changetype」が検出され、その値が「modify」、「moddn」、または「modrdn」に等しい場合、項目はそれに応じてタグ付けされたデルタです。

「add」、「replace」、または「delete」のいずれかのキーが検出された場合、項目の属性はそれに応じてタグ付けされています。

LDIF 出力の書き込み

ここに記載されている情報を使用して、LDIF パーサーの使用中に LDIF 出力を書き込むことができます。

最初の書き込み時に、「**バージョン番号**」パラメーターが選択されているかどうかが確認され、選択されている場合は、テキスト「version: 1」が最初の行に書き込まれます。これは、出力が RFC 2489 LDIF 仕様に準拠していることを意味します。その後、「dn」キーが「dnAttributeName」属性 (存在する場合) の値とともに追加されます。

項目がタグ付きデルタの場合、対応する changetype キーが、項目の操作および属性に応じて値「add」、「modify」、「modrdn」、または「delete」とともに追加されます。ただし、パラメーター「**記述レコードのみ**」が設定されている場合は、項目がタグ付きデルタの場合でも変更レコードは書き込まれません。

項目の属性がタグ付きデルタの場合、対応する操作 (「add」、「replace」、または「delete」) が属性の値とともに出力に追加されます。

構成

ここに記載されているパラメーターを使用することで、LDIF パーサーを構成できるようになります。

DN 属性名
LDIF「dn」行に使用する属性名。

バージョン番号
チェックした場合は、RFC 2849 で規定されているように、出力の始めにバージョン属性が表示されます。 デフォルトでは、このパラメーターは **On** です。

注: LDIF パーサーは、「**バージョン番号**」パラメーターを使用して LDIF バージョン番号を抑止できます。

バイナリー属性
バイナリーとして扱われる追加属性を指定する必要がある場合は (バイナリー属性はストリングではなくバイト配列として戻されます)、このパラメーターに追加属性を指定します。 デフォルトで、バイナリーと見なされるのは以下の属性です。

- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedData
- thumbnailPhoto
- thumbnailLogo
- userPassword
- userCertificate
- authorityRevocationList
- certificateRevocationList
- crossCertificatePair
- x500UniqueIdentifier

- objectGUID
- objectSid

文字エンコード

使用される文字エンコード。デフォルトは UTF-8 です。 423 ページの『文字エンコード変換』も参照してください。

記述レコードのみ

設定した場合は、記述レコードのみが書き込まれます。デフォルトでは、このパラメーターは **Off** です。

LDIF ファイルには、「変更レコード」または「記述レコード」が含まれている場合があります。変更レコードには、項目に必要な変更が記述されています。記述レコードでは、項目が単に記述されています。

レコードが変更レコードであるかどうかを確認する簡単な方法は、「dn」行の直後の 2 行目として「changetype」行が含まれていることを確認することです。

正しい LDIF ファイルには、変更レコードのみが含まれるか、または記述レコードのみが含まれています。

LDIF パーサーは作業項目の命令コードを使用して、変更レコードを書き込むか、または単に記述レコードを書き込むかを判断します。つまり、作業項目に汎用でない操作が含まれる場合、変更レコードを書き込む必要があると判断されます。これは非常に便利ですが、すべての場合に最適な方法とはいえません。

使用可能なデルタが設定されたコネクターからの作業項目であっても、LDIF ファイルに記述レコードのみを含める必要がある場合があります。例えば、LDIF ファイルを使用するシステムで読み取りできるレコードが記述レコードである場合です。

このフラグが設定されると、項目の操作に関係なく、記述レコードのみが生成されるようになります。操作が Delete の場合はその項目に関する書き込みは行われませんが、その他の場合は、項目に含まれる属性値が記述レコードとして書き込まれます。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

言語タグのサポート

言語タグのサポートが設定されている場合、属性名に言語タグが含まれます。

関連情報

http://www.ietf.org/rfc/rfc2849.txt

行リーダー・パーサー

行リーダー・パーサーは、単一行のデータを読み取ります。読み取られた行は単一の属性として戻されます。 ここに記載されている情報とリンクを使用して、行リーダー・パーサーについて詳しく知ることができます。

他に **linenumber** という属性もあり、これには **1** から始まる行番号が入ります。

注: 行リーダー・パーサーは、テキスト・ファイルのみをコピーする場合に使用します。バイナリー・ファイルをコピーする場合は、バイナリー・ファイルをコピーする方法について、681 ページの『例』 の FTP オブジェクトの例を参照してください。

行リーダー・パーサー は、テキスト・ファイルのみを読み取るときに便利です。

構成

ここに記載されているパラメーターを使用することで、行リーダー・パーサーを構成できるようになります。

属性名 直前に読み取ったか、書き込もうとしているテキストの行を含む属性の名前を指定します。デフォルトは **line** です。

文字エンコード
使用される文字エンコード。423 ページの『文字エンコード変換』も参照してください。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

スクリプト・パーサー

スクリプト・パーサーを使用すると、JavaScript で独自のパーサーを作成できます。

スクリプト・パーサーが機能するためには、いくつかの関数をインプリメントする必要があります。これらの関数は、パラメーターを使用しません。

注: スクリプト・パーサーのスクリプトは、別の JavaScript エンジンで実行されます。これは、スクリプトが使用可能または設定済みのいずれの変数にも通常の AssemblyLine フックでアクセスできないことを意味します。
ホストとして稼働するコネクターとスクリプトの間のデータの受け渡しには、事前定義済みのオブジェクトが使用されます。これらの定義済みオブジェクトの 1 つに**結果**オブジェクトがあり、これは状況情報を伝達するために使用されます。いずれかの関数に入る時点では、状況フィールドは**通常**に設定されており、その場合ホスト・パーサーは呼び出しを続行します。入力終了またはエラーを通知するには、このオブジェクト内の状況フィールドおよびメッセージ・フィールドを設定します。**項目**オブジェクトは、**writeEntry** の呼び出し時に移植され、**readEntry** 関数で移植されると予期されます。項目を読み取るときは、ストリームからの文字データの読み取りのために **inp** BufferedReader オブジェクトを使用できます。項目を書き込むときは、ストリームへの文字データの書き込みのために **out** BufferedWriter オブジェクトを使用できます。

ユーザーは、構成に独自のパラメーターを追加し、**parser** オブジェクトを使用してそれらのパラメーターを取得することができます。

注: スクリプト・コネクターまたはパーサーを使用するとき、スクリプトは、それが入っているライブラリーから構成ファイルにコピーされます。これには、スクリプトをカスタマイズできるという利点がありますが、AssemblyLine が新規バージョンを認識しないという欠点もあります。

この欠点に対処するには、AssemblyLine から旧スクリプト・パーサーを除去し、再導入します。

オブジェクト

ここに記載されている共通オブジェクトのリスト (これらは AssemblyLine の場合と同じです) を使用して、スクリプト・パーサーを操作できます。

main 実行中の構成インスタンス (RS オブジェクト)。

task このパーサーが含まれる AssemblyLine。

system `UserFunctions` オブジェクト。

config このエレメントの構成。つまり、このパーサーです。

スクリプト・パーサーからアクセスできるオブジェクトは、以下のみです。

結果オブジェクト

ここに記載されているパラメーターを使用して、結果オブジェクトを設定できます。

setStatus
コード

- **0**: 入力終了
- **1**: 状況 OK
- **2**: エラー

setMessage
テキスト

項目オブジェクト

ここに記載されているパラメーターを使用して、項目オブジェクトを設定できます。

addAttributeValue (name, value)
属性に値を追加します。

getAttribute (name)
指定した属性を戻します。

使用できるすべてのメソッドのリスト (パラメーターおよび戻り値を含む) は、Javadoc (*TDI_install_dir*/docs/api/com/ibm/di) にあります。

inp オブジェクト

ここに記載されているパラメーターを使用して、inp オブジェクトを設定できます。

read() ストリーム内の次の文字を戻します。

readLine()
入力ストリーム内の次の CRLF 終了行を戻します。

out オブジェクト

ここに記載されているパラメーターを使用して、out オブジェクトを設定できます。

write (str)
出力ストリームにストリングを書き込みます。

writeln (str)
CRLF を伴うストリングを出力ストリームに書き込みます。

パーサー・オブジェクト

ここに記載されているパラメーターを使用して、パーサー・オブジェクトを設定できます。

getParam(str)
パラメーター名 **str** に関連したパラメーター値を戻します。

setParam(str, value)
パラメーター **str** の値を **value** に設定します。

logmsg(str)
パラメーター **str** をログ・ファイルに書き込みます。

すべてのメソッドのリストは、インストール・パッケージにあります。

コネクター・オブジェクト

詳しくは、インストール・パッケージに収められている Javadoc の資料を参照してください。

関数 (メソッド)

パーサーは、ここで提供された関数を提供する必要があります (IBM Security Directory Integrator で予期する使用法に関連します)。

readEntry()
入力ストリームから次の論理項目を読み取り、**項目**オブジェクトに取り込みます。この関数は、add_only の状況のみで呼び出されるパーサーには不要です。

writeEntry()
項目オブジェクトの内容を出力ストリームに書き込みます。この関数は、読み取りのみに使用されるパーサーには不要です。

closeParser ( )
closeParser 関数がインプリメントされている場合、Connector.close の呼び出し時に呼び出されます。例を示します。

```
function closeParser ( )
{
  task.logmsg("CLOSE CALLED.");
}
```

flush() flush 関数は、connector.getParser( ).flush( ) メソッドによりパーサーの flush

が呼び出される場合に呼び出されます。これらのメソッドをインプリメントすると、パーサーのメソッドがオーバーライドされます。例を示します。

```
function flush ( )
{
  task.logmsg("FLUSH CALLED.");
}
```

querySchema()

querySchema 関数は、パーサーの親、つまりこのパーサーが構成されているコネクターによって呼び出されます。この関数を使用して基本データ・ソースのスキーマを検出し、入出力マップを取り込みます。詳しくは、『スキーマ』を参照してください。

構成

ここに記載されているパラメーターを使用することで、スクリプト・パーサーを構成できるようになります。

外部ファイル

実行時に外部スクリプト・ファイルを組み込む場合は、それらのファイルを指定します。ファイルは 1 行に 1 つずつ指定してください。これらのファイルは、スクリプトの前に実行されます。

グローバル・スクリプトの組み込み

スクリプト・ライブラリーからスクリプトを組み込みます。

文字エンコード

使用される文字エンコード。423 ページの『文字エンコード変換』も参照してください。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

スクリプト

実行するユーザー定義スクリプト。

スキーマ

ここに記載されている情報を使用して、スキーマについて理解することができます。

サンプルの *querySchema* 関数が、構成パラメーター「**スクリプト**」で提供されています。テキスト・ファイルから一度に 1 行ずつ読み取るデフォルト・ケースを想定しているため、この関数から戻されるスキーマにはタイプ java.lang.String の「line」という名前の 1 つのフィールドのみが含まれます。特定の動作が必要な場合は、この関数をオーバーライドする必要があります。

この関数には、以下の 2 つのスクリプト・オブジェクトからアクセスできる 2 つの定義済みオブジェクトがあります。

list これはベクトル・オブジェクトです。querySchema(Object) 関数は、項目オブジェクトをこのベクトルに追加する必要があります。

Source これは、querySchema(Object) 関数が呼び出されたときにこの関数に渡されるオブジェクト・パラメーターです。

有効なスキーマの照会を作成するには、少なくとも 1 つの「name」という名前の属性、およびオプションの属性「syntax」または「extsyntax」を含む項目とともに、定義済みの **list** オブジェクトを取り込む必要があります。これは、*項目* オブジェクトを作成し、その addAttributeValue 関数を呼び出して目的の値を属性に設定することで実行できます。

スキーマの取得に成功したら、以下の値のいずれかが設定された定義済みの結果オブジェクトの関数 setStatus(int) を呼び出す、3 つのタイプの終了コードを設定できます。

- 0: 入力終わり
- 1: 状況 OK
- 2: エラー

結果に関してより詳細な情報を設定するために、テキスト形式の 1 つのパラメーターを扱う setMessage(String) 関数を使用できます。終了コードが 1 の場合にのみ、querySchema(Object) によってスキーマが戻され、それ以外の場合は NULL が戻されます。

例

ここに記載されているパスとリンクを使用して、スクリプト・パーサーについて詳しく知ることができます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/script_connector ディレクトリーにあります。

関連情報

310 ページの『スクリプト・コネクター』,
519 ページの『スクリプト記述関数コンポーネント』
「リファレンス」の『JavaScript パーサー』。

単純なパーサー

単純なパーサーは、項目の読み取りおよび書き込みを行います。ここに記載されている情報を使用することで、単純なパーサーについて詳しく知ることができます。

形式は、***attributename*:*value*** のペアの複数行です。ここで、*attributename* は属性名、*value* は値です。

多値属性には複数の行を使用します。単一のピリオドがある行は項目の終わりを示します。*value* の中の **¥r** および **¥n** は、CR および LF のエンコードです。

構成

ここに記載されているパラメーターを使用することで、単純なパーサーを構成できるようになります。

文字エンコード

使用される文字エンコード。 423 ページの『文字エンコード変換』も参照してください。

詳細ログ

このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

SOAP パーサー

SOAP パーサーを使用して、SOAP XML 文書の読み取りおよび書き込みを行うことができます。

このパーサーは、単純かつ直接的な方法で、SOAP XML 文書を項目オブジェクトに、そして項目オブジェクトを SOAP XML 文書に変換します。 XML 文書を作成するとき、パーサーは項目からの属性を使用して文書を作成します。**SOAP_CALL** 属性には、SOAP 呼び出しの値が含まれます。同様に、読み取りの際には、この属性は **SOAP-ENV:Body** タグの後の最初のタグを示すように設定されます。そして、項目内の各属性について、その名前と値を持つタグが作成されます。文書の読み取りの際には、**SOAP_CALL** タグの下の各タグは項目オブジェクト内の属性に変換されます。

注: Web サービスのコネクターを操作するときに、属性名の最初の 1 文字には、特殊文字 (**[0 から 9] [- ' ( ) + , . / = ? ; ! * # @ $ %]** など) を使用することはできません。また、特殊文字 (**[' ( ) + , / = ?;! * # @ $ %]** など) を属性名の中で使用することもできません。これは、Web サービスは SOAP (つまり XML) を基礎として作成されているためです。XML は、タグ内の文字として $ を受け入れません。

以下の例は、読み取りおよび書き込みの際の項目および SOAP XML 文書を示しています。

項目の例

ここに記載されている例を使用することで、項目を理解できるようになります。

```
*** Begin Entry Dump
 SOAP_CALL: 'updateLDAP'
 mail: ('john@doe.com)'
 uid: 'johnd'
*** End Entry Dump
```

SOAP 文書の例

ここに記載されている例を使用することで、SOAP 文書を理解できるようになります。

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="(http://schemas.xmlsoap.org/soap/envelope/)"
 xmlns:xsi="(http://www.w3.org/1999/XMLSchema-instance)"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:updateLDAP xmlns:ns1="" SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
<uid xsi:type="xsd:string">johnd</uid>
<mail xsi:type="xsd:string">john@doe.com</mail>
</ns1:updateLDAP>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

構成

ここに記載されているパラメーターを使用することで、SOAP パーサーを構成できるようになります。

XML 宣言の省略
出力ストリーム内の XML 宣言ヘッダーを省略します。

文書の妥当性検査
DTD/XSchema 妥当性検査 XML パーサーを要求します。

ネーム・スペースを意識
ネーム・スペース認識 XML パーサーを要求します。

文字エンコード
使用される文字エンコード。デフォルトは UTF-8 です。 423 ページの『文字エンコード変換』も参照してください。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

パーサー固有の呼び出し

SOAP パーサーには、ユーザーのスクリプトからアクセスできます。このスクリプトでは、このパーサーを動的にロードし、SOAP 文書の読み取りまたは書き込みを行うメソッドを呼び出します。

以下の例は、項目から XML 文書を生成する方法を示しています。

```
var e = system.newEntry();
e.setAttribute ("soap_call", "updateLDAP");
e.setAttribute ("uid", "johnd");
e.setAttribute ("mail", "(john@doe.com)");

// Retrieve the XML document as a string
var soap = system.getParser ("ibmdi.SOAP");
soap.initParser();
var soapxml = soap.getXML ( e );

task.logmsg ( "SOAP XML Document" );
task.logmsg ( soapxml );

// Write to a file
var soap = system.getParser("ibmdi.SOAP");
soap.setOutputStream ( new java.io.FileOutputStream("mysoap.xml") );
soap.writeEntry ( e );
soap.close();

// Read from file
soap.setInputStream ( new java.io.FileInputStream ("mysoap.xml") );
var entry = soap.readEntry();

// Read from string (from soapxml generated above)
var entry = soap.parseRequest( soapxml );
task.dumpEntry ( entry );
```

例

ここに記載されているパスを使用して、SOAP パーサーの例にアクセスできます。

ご使用の IBM Security Directory Integrator システムの *TDI_install_dir*/examples/soap ディレクトリーにあります。

SPMLv2 パーサー

SPML バージョン 2 (SPMLv2) は、実際のプロビジョニング・データを定義するためにさまざまなデータ・モデルを使用できるコア・プロトコル [SPMLv2] を定義します。ここに記載されている情報を使用することで、これについて理解することができます。

データ・モデルと SPML コア仕様の組み合わせは、プロファイルと呼ばれます。どのプロファイルが使用されるかは、参加する側によってアプリケーションのフロー以外で交渉して選択されますが、SPML の使用には、特定のプロファイルの使用が必要となります。

DSML v2 プロトコル [DSMLV2] は、Web サービスを使用して LDAP タイプの操作を実行するように設計されています。DSML V2 プロトコルでは、同期要求/応答セマンティクス、および属性/値のペアに基づくデータ・モデルを定義します。DSML V2 では、属性/値ペアのスキーマ・メカニズムを定義しません。

SPMLv2 パーサーは、SPMLv2 DSMLv2 プロファイルをサポートしています。これは、SPMLv2 メッセージの解析と作成を行う IBM Security Directory Integrator パーサー・コンポーネントです。このパーサーは、個々の SPMLv2 要求と応答を解析したり、SPMLv2 要求と応答を書き込んだりすることを目的としています。

SPMLv2 パーサーは、「(SPML) v2 - DSML v2 Profile」仕様で規定されているように、コア操作をサポートします。明示的な IBM Security Directory Integrator 項目スキーマが、サポートされている各操作に対して定義されています。

このパーサーは 481 ページの『XML パーサー』 を拡張したものであり、すべての SPML メッセージをメモリーに作成しなくても、大量の要求/応答を読み取ることができます。さらに、このパーサーは OpenSPML 2.0 Toolkit の最上部にインプリメントされています。

このパーサーは、バッチ・メッセージを読み取り、書き込むことができます。以下のタイプのツールキットが使用されています。

```
org.openspml.v2.msg.spmlbatch.BatchRequest;
org.openspml.v2.msg.spmlbatch.BatchResponse;
```

各 **readEntry** 呼び出しでは、このパーサーはバッチ・メッセージに含まれている個々の要求または応答を表す項目を戻します。**writeEntry** では、このパーサーは適切なバッチ・メッセージに個々の要求または応答を書き込みます。

操作

適合するプロバイダーは、コア XSD で定義されているすべての操作をインプリメントする必要があります。コア操作は指定されたとおりです。

- 追加 (追加要求および追加応答)
- 変更 (変更要求および変更応答)
- 削除 (削除要求および削除応答)
- ルックアップ (ルックアップ要求およびルックアップ応答)

このパーサーは以下の検索操作もサポートしています。

- 検索 (検索要求および検索応答)

追加要求

追加要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Add」に設定されます。
spml.operation.type	「Request」に設定されます。
spml.containerID	addRequest エレメントのサブエレメントとして containerID エレメントが存在する場合は、containerID エレメントの ID 属性の値に設定されます。
spml.containerID.targetID	containerID エレメントに targetID 属性が存在する場合は、ID 属性の値に設定されます。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

また、DSML の attr エレメントそれぞれについて、DSML の attr エレメントの「name」という XML 属性で指定された名前を付けた属性名。この属性の値は、DSML の attr エレメントで指定された値になります。

追加応答

追加応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Add」に設定されます。
spml.operation.type	「Response」に設定されます。
spml.psoID	「psoID」エレメントが応答で使用可能な場合は、「psoID」エレメントの ID 属性値に設定されます。
spml.pso.targetID	プロバイダーが複数のターゲットをサポートしている場合は、psoID エレメントの targetID 属性値に設定されます。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。
spml.errorCode	追加要求が失敗した場合に作成されます。この属性の値は、失敗の特徴を示している必要があります。
spml.status	AddResponse エレメントの状況属性の値を保持します。
spml.errorMessages	要求された操作の状況または失敗に関する追加情報を提供するストリング・オブジェクトの配列。

変更要求

変更要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

変更操作では、変更されたオブジェクトの ID が変更される場合があることに注意してください。

属性	値
spml.operation	「Modify」に設定されます。
spml.operation.type	「Request」に設定されます。

属性	値
spml.psoID	ID 属性の値に設定されます。変更要求には、ターゲットに存在し、プロバイダーによって公開されるオブジェクトを識別する <psoID> エレメントが常に含まれている必要があります。
spml.pso.targetID	プロバイダーが 1 つのターゲットのみをサポートしている場合、この属性は指定されない場合があります。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

また、変更項目それぞれについて、DSML の modification エレメントの「name」という XML 属性で指定された名前を付けた属性。この属性の値は、DSML の modification エレメントで指定された値になります。また、IBM Security Directory Integrator 属性の操作は、DSML の modification エレメントの「operation」という XML 属性と同じに設定されます。

変更応答

変更応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Modify」に設定されます。
spml.operation.type	「Response」に設定されます。
spml.psoID	プロバイダーが要求されたオブジェクトの変更に成功した場合、 <modifyResponse> には <pso> エレメントが含まれている必要があります。 <pso> には、<modifyRequest> の「returnData」属性で指定した、要求されたオブジェクトの (XML 表現の) サブセットが含まれています。
spml.pso.targetID	プロバイダーが 1 つのターゲットのみをサポートしている場合、この属性は指定されない場合があります。
spml.status	ModifyResponse エレメントの状況属性の値。
spml.errorCode	要求が失敗した場合に作成されます。この属性の値は、失敗の特徴を示している必要があります。この属性の値は、SPML 仕様による事前定義値の 1 つである場合があります。
spml.errorMessages	要求された操作の状況または失敗に関する追加情報を提供するストリング・オブジェクトの配列。この属性はオプションです。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

削除要求

削除要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Delete」に設定されます。
spml.operation.type	「Request」に設定されます。
spml.psoID	<psoID> エレメントの「ID」属性の値に設定されます。「削除要求」には常に PSO ID が含まれている必要があります。
spml.pso.targetID	プロバイダーが 1 つのターゲットしかサポートしない場合は、属性を指定しなくてもかまいません。

属性	値
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

削除応答

削除応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Delete」に設定されます。
spml.operation.type	「Response」に設定されます。
spml.errorCode	要求が失敗した場合に作成されます。この属性の値は、失敗の特徴を示している必要があります。この属性の値は、SPML 仕様による事前定義値の 1 つである場合があります。
spml.status	DeleteResponse エレメントの状況属性の値を保持します。
spml.errorMessages	要求された操作の状況または失敗に関する追加情報を提供するストリング・オブジェクトの配列。この属性はオプションです。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

ルックアップ要求

ルックアップ要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Lookup」に設定されます。
spml.operation.type	「Request」に設定されます。
spml.psoID	<psoID> エレメントの「ID」属性の値に設定されます。ルックアップ要求は常に PSO ID を指定している必要があります。
spml.pso.targetID	プロバイダーが 1 つのターゲットしかサポートしない場合は、属性を指定しなくてもかまいません。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

ルックアップ応答

ルックアップ応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Lookup」に設定されます。
spml.operation.type	「Response」に設定されます。
spml.psoID	<psoID> エレメントの「ID」属性の値に設定されます。
spml.pso.targetID	<psoID> エレメントの「targetID」属性と同じです。プロバイダーが 1 つのターゲットしかサポートしていない場合は、指定しなくてもかまいません。
spml.status	LookupResponse エレメントの状況属性の値を保持します。

属性	値
spml.requestID	各要求内の「requestID」属性の合理的な固有値。「requestID」値はグローバルに固有である必要はありません。「requestID」に必要なのは、未処理の各要求を識別するのに十分な一意性だけです。
spml.errorMessage	要求された操作の状況または失敗に関する追加情報を提供するストリング・オブジェクトの配列。この属性はオプションです。

検索要求

検索要求に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Search」に設定されます。
spml.operation.type	「Request」に設定されます。
spml.scope	検索範囲
spml.containerID	検索要求の「basePsoID」エレメントの「ID」属性の値を含みます。
spml.containerID.targetID	検索要求の「basePsoID」エレメントの「targetID」属性の値を含みます。
spml.attributeDescription	検索要求の「attributes」エレメントにリストされている属性の名前を保持するストリング値を持つ多値属性。
spml.filter.substrings.name	Filter の Substrings エレメントの Name の値を含みます。
spml.filter.substrings.initial	Filter の Substrings エレメントの Initial エレメントの値。
spml.filter.substrings.any	Filter の Substrings エレメントの Any エレメントの値を含む多値属性。
spml.filter.substrings.final	Filter の Substrings エレメントの Final の値を含みます。
spml.filter	Filter エレメントの値を Attribute 階層オブジェクトとして含みます。これは、v7.0 階層オブジェクトを使用します。

検索フィルターについて詳しくは、 474 ページの『検索フィルターの機能』を参照してください。

検索応答

検索応答に対して、指定された構造の項目を使用して、パーサーによって解析 (読み取り時) および作成 (書き込み時) することができます。

属性	値
spml.operation	「Search」に設定されます。
spml.operation.type	「Response」に設定されます。
spml.resultEntries	その各値が、個々の「<spml:pso>」エレメントの「<spml:data>¥attr」エレメントに対応している属性を持つ項目である多値属性。
spml.errorCode	要求が失敗した場合に作成されます。この属性の値は、失敗の特徴を示している必要があります。この属性の値は、SPML 仕様による事前定義値の 1 つである場合があります。
spml.status	SearchResponse エレメントの状況属性の値を保持します。
spml.errorMessages	要求された操作の状況または失敗に関する追加情報を提供するストリング・オブジェクトの配列。この属性はオプションです。
spml.requestID	未処理の各要求を識別する、合理的な固有値です。

バイナリー属性とストリング以外の属性

ここに記載されている情報を通じて、バイナリー属性とストリング以外の属性について詳しく知ることができます。

SPML メッセージの解析時に、「**バイナリー属性**」パーサー・パラメーターによってバイナリーとしてタグ付けされた属性は、Base64 でデコードされます。つまり、SPML メッセージのストリング値は Base64 でデコードされて Java のバイト配列に入れられます。

Java バイト配列を値として持つ属性はすべて、SPML メッセージの作成時には Base64 でエンコードされてストリングに変換されてから SPML メッセージに書き込まれます。

SPML メッセージの作成時に、値のタイプがストリングでも Java バイト配列でもない属性が渡されると、その値はその属性オブジェクトの `toString()` メソッドを呼び出すことによってストリングに変換され、このストリング値が SPML メッセージに格納されます。

属性の操作のタグ付け

ここに記載されている情報およびコード例を使用して、操作属性にタグを付けることができます。

SPMLv2 パーサーは、さまざまな変更操作を処理する場合、SPMLv2 文書内の「*dsml:modification*」操作属性の値に従って項目属性にタグを付けます。

例えば、以下の SPML 構造を読み取る場合を考えてみます。

```
<modifyRequest xmlns='urn:oasis:names:tc:SPML:2:0' returnData='everything'>
 <psoID ID='CN=DnsUpdateProxy,OU=Groups,DC=2k3,DC=dom'/>
 <modification>
  <dsml:modification xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='member' operation='delete'>
   <dsml:value>CN=Eric Clapton,CN=Users,DC=2k3,DC=dom</dsml:value>
  </dsml:modification>
  <dsml:modification xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='member2' operation='add'>
   <dsml:value>CN=Eric Adams,CN=Users,DC=2k3,DC=dom</dsml:value>
  </dsml:modification>
  <dsml:modification xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='member3' operation='replace'>
   <dsml:value>CN=Joey DeMaio,CN=Users,DC=2k3,DC=dom</dsml:value>
  </dsml:modification>
 </modification>
</modifyRequest>
```

この SPML の抽出部分を解析して項目にする場合は、「member」、「member2」、および「member3」という名前の 3 つの属性をマップする必要があります。各属性は、対応する操作 (すなわち、delete、add、replace) でタグ付けされます。変更操作の値には、以下のスクリプトを使用してもアクセスすることができます。

```
work.getAttribute("member").getOperation();
```

項目を解析して SPML 文書にする場合は、作業項目で提供された属性操作を用いて属性にタグが付けられます。この変更操作が設定されていない場合は、デフォルトである「replace」が使用されます。あるいは、以下のスクリプトを使用して手動で設定することもできます。

```
work.getAttribute("member").setOperation("add");
```

検索フィルターの機能

ここに記載された情報を通じて、検索フィルター機能について詳しく知ることができます。

IBM Security Directory Integrator の旧バージョンでは、SPMLv2 パーサーは、検索要求でのみ Filter の Substrings エレメントをサポートしており、属性 spml.filter.substrings.name、spml.filter.substrings.initial、spml.filter.substrings.final、および spml.filter.substrings.any を使用して Substrings サブエレメントの値を含めていました。

SPMLv2 パーサーの現行バージョンは、以下に示した、DSMLv2 で提供されている上記以外のフィルター機能を解析することができます。

- **not** - 含まれているフィルター項目の否定
- **and** - 複数のフィルター項目を含む論理「積」
- **or** - 複数のフィルター項目を含む論理「和」
- **equalityMatch** - 同等で一致していることを示すフィルター項目
- **approxMatch** - ほぼ一致していることを示すフィルター項目
- **extensibleMatch** - 拡張一致を示すフィルター項目
- **greaterOrEqual** - 以上の場合に一致を示すフィルター項目
- **lessOrEqual** - 以下の場合に一致を示すフィルター項目
- **present** - 指定された属性が存在していることを示すフィルター項目

読み取り

パーサーは、Filter エレメントの読み取り時に、それに対応する階層属性を「spml.filter」という名前で作成します。すべてのフィルター項目に対して、それぞれに個別の属性オブジェクトが作成され、子として付加されます。それに応じて、フィルター項目に含まれているすべてのサブエレメントまたは別のフィルター項目に対して、個別の属性が作成され、その子として追加されます。

注: DSMLv2 スキーマに従うと、**not** エレメントには 1 つのフィルター項目しか含めることができません。

属性「spml.filter.substrings.name」、「spml.filter.substrings.initial」、「spml.filter.substrings.final」、および「spml.filter.substrings.any」は、Filter エレメントに単一の Substrings エレメントが含まれている場合にしか作成されません。それ以外の場合は、spml.filter 階層属性が作成されます。

制限: SPMLv2 パーサーは、読み取り時に extensibleMatch エレメントの「matchingRule」および「dnAttributes」属性を読み取ることができません (これらの属性の値は常にデフォルト値である false です)。これは、基礎となる Open SPML 2.0 ライブラリーが matchingRule と dnAttributes を属性ではなく値として読み取ろうとするためです。ただし、これらの属性を書き込む場合は正しく書き込まれます。

書き込み

書き込み時に、提供されている項目に「smpl.filter」属性が存在している場合はこの属性が使用され、「spml.filter.substrings.name」、「spml.filter.substrings.initial」、「spml.filter.substrings.final」、および「spml.filter.substrings.any」属性は無視されます。「smpl.filter」属性が存在していない場合は、「spml.filter.*」属性が使用されます (以前のバージョンとの互換性)。

「spml.filter」階層属性、およびこの属性から生成された対応する XML を以下に示します。

表 47. 「spml.filter」階層属性

「spml.filter」属性	Filter エレメント

'"and": {
"or": {
"substrings": {
"name": "cn",
"initial": "J"
},
"not": {
"and": {
"lessOrEqual": {
"name": "roomnumber",
"value": "2000"
},
"greaterOrEqual": {
"name": "roomnumber",
"value": "3000"
}
}
},
"approxMatch": {
"name": "sn",
"value": "Smith"
}
},
"equalityMatch": {
"name": "objectClass",
"value": "inetorgperson"
}
}'
	<dsml:filter xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'>
	<dsml:and>
	<dsml:or>
	<dsml:substrings name='cn'>
	<dsml:initial>J</dsml:initial>
	</dsml:substrings>
	<dsml:not>
	<dsml:and>
	<dsml:lessOrEqual name='roomnumber'>
	<dsml:value>2000</dsml:value>
	</dsml:lessOrEqual>
	<dsml:greaterOrEqual name='roomnumber'>
	<dsml:value>3000</dsml:value>
	</dsml:greaterOrEqual>
	</dsml:and>
	</dsml:not>
	<dsml:approxMatch name='sn'>
	<dsml:value>Smith</dsml:value>
	</dsml:approxMatch>
	</dsml:or>
	<dsml:equalityMatch name='objectClass'>
	<dsml:value>inetorgperson</dsml:value>
	</dsml:equalityMatch>
	</dsml:and>
	</dsml:filter>

構成

ここに記載されているパラメーターを使用することで、SPMLv2 パーサーを構成できるようになります。

バイナリー属性

パーサーによってバイナリー属性 (必要に応じて Base64 デコード/エンコードされています) として処理される属性がコンマ区切り文字で区切られているリストを指定します。

このパラメーターは、変更可能な属性のデフォルト・リストを持っています。これらの属性は、「photo」、 「personalSignature」、「audio」、「jpegPhoto」、 「javaSerializedData」、 「thumbnailPhoto」、「thumbnailLogo」、 「userPassword」、「userCertificate」、「authorityRevocationList」、「certificateRevocationList」、「crossCertificatePair」、 「x500UniqueIdentifier」、 「objectGUID」、および「objectSid」です。

文字エンコード
読み取りまたは書き込み時に使用される文字エンコード。デフォルトは UTF-8 です。このパーサーは XML パーサーを拡張したものであるため、XML パーサーの場合と同じ考慮事項が適用されます。

詳細ログ
この項目にチェック・マークを付けると、詳細ログが生成されます。

例

ここに記載されているパスを使用して、SPMLv2 パーサーの例にアクセスできます。

このパーサーの使用方法の例が *TDI_install_dir*/examples/SPMLv2Parser ディレクトリーにあります。

関連情報

432 ページの『DSMLv2 パーサー』

単純 XML パーサー

単純 XML パーサーは XML 文書の読み取りと書き込みを行います。単純 XML パーサーは、深くても 2 レベルまでの深度の XML データを扱います。ここに記載されている情報とリンクを使用することで、単純 XML パーサーについて詳しく知ることができます。

このパーサーは、Apache の Xerces ライブラリーおよび Xalan ライブラリーを使用します。このパーサーでは、**xmldom** というスクリプト・オブジェクトを使用して XML 文書にアクセスできます。**xmldom** は、org.w3c.dom.Document インターフェースのインスタンスです。このインターフェースの詳細については、http://docs.oracle.com/javase/6/docs/api/ を参照してください。

XML 文書からのノードの検索と選択は、XPathAPI (http://xml.apache.org/xalan-j/apidocs/index.html) を使用して、スクリプト内のその Java クラスにアクセスして行うこともできます。**selectNodeList** (**システム**・オブジェクト内の便利なメソッド) を使用すると、XML 文書からサブセットを選択できます。

DTD タグが存在している場合、単純 XML パーサーはコネクターの初期化時に文書型定義 (DTD) 検証を実行しようとします。

ユーザーが自分で XML 文書を解釈または生成するには、コネクターの指定変更関数を使用します。AssemblyLine のフック定義の「**GetNext の指定変更**」または「**GetNext 成功**」を使用して、必要なスクリプトを作成します。指定変更しない場

合、パーサーは、項目オブジェクト・モデルに類似した非常に単純な XML 文書の読み取りまたは書き込みを行います。デフォルトのパーサーでは、深度 2 レベルまでの XML ファイルの読み取りまたは書き込みしかできません。また、多値属性も読み取ります。ただし、データを「スキーマ」タブでブラウズしているときに表示されるのは、多値属性のうち 1 つのみです。

setAttribute など一部のメソッドは、IBM Security Directory Integrator の**項目** と、**xmldom.createElement** から戻されるオブジェクトの両方で使用できるという点に注意してください。これらの関数は、同じ名前またはシグニチャーを持ちます。**xmldom** のオブジェクトと IBM Security Directory Integrator のオブジェクトを混同しないようにしてください。

注:

1. このパーサーは、IBM Security Directory Integrator 7.0 よりも前のリリースでは「XML パーサー」と呼ばれていました。IBM Security Directory Integrator 7.0 では、単純 XML パーサーという名前に変更され、新しい XML パーサーが追加されました。481 ページの『XML パーサー』を参照してください。この新しいパーサーでは数多くの改善がなされており、メインの IBM Security Directory Integrator XML パーサーになっています。
2. (4 MB を超える) 大きな XML ファイルを読み取るか、(14 MB を超える) 大きな XML ファイルを書き込むと、Java VM がメモリー不足に陥る可能性があります。この問題の対処方法については、「」の『仮想マシンが使用できるメモリーの増加』を参照してください。別の方法として、481 ページの『XML パーサー』 または 493 ページの『XML SAX パーサー』 を使用することもできます。
3. このパーサーは、空の項目を警告なしに無視します。
4. CDATA の属性を読み取るとき、値からブランク・スペースが切り取られることはありません。ただし、CDATA 以外の属性からは空白が切り取られます。
5. **$** などの一部の文字は、XML タグでは無効です。XML パーサーを使用するときは、属性名にこのような文字を使用しないでください。使用すると、無効な XML が作成されることがあります。
6. LDAP ディレクトリーまたは LDIF ファイルから読み取る場合は、通常、識別名 (DN) が **$dn** という名前の属性で返されます。$dn は XML 文書用の有効なタグではないため、識別名を変更せずにこの属性を XML ファイルにマップしようとすると失敗します。明示的マッピングを行う場合は、出力コネクター内で、"$dn" を "dn" (または特殊文字を含まないその他の値) に変更する必要があります。暗黙的マッピング (例えば、*、または AssemblyLine の「**構成...**」タブの「**AssemblyLine 設定**」で「**自動的にすべての属性をマップ**」をチェックする) を行う場合は、識別名 (例えば $dn) を別の名前に変換するように XML パーサーを構成することができます。例えば、**GetNext 前**フックに次のような行を追加します。

```
conn.setAttribute("dn", work.getAttribute("$dn"));
conn.removeAttribute("$dn");
```

構成

ここに記載されているパラメーターを使用することで、単純 XML パーサーを構成できるようになります。

ルート・タグ
ルート・タグ (出力)。

項目タグ
項目用の項目タグ (出力)。

値タグ 項目属性用の値タグ (出力)。

文字エンコード
使用される文字エンコード。『単純 XML パーサーにおける文字エンコード』を参照してください。

XML 宣言の省略
これをチェックした場合は、出力ストリームから XML 宣言が省かれます。

文書の妥当性検査
これをチェックした場合は、このパーサーは DTD/Schema 妥当性検査パーサーを要求します。

ネーム・スペースを意識
これをチェックした場合は、このパーサーはネーム・スペース認識パーサーを要求します。

インデント出力
このフィールドにチェック・マークを付けると、出力はインデント処理されます。

注: このテキストをプログラムで処理する場合 (人間が解釈できるようにする必要がない場合) には、このパラメーターを選択解除することをお勧めします。そうすれば、スペースや改行が不必要に出力に挿入されることがありません。

詳細ログ
このパラメーターをチェックすると、より詳細なログ・メッセージが生成されます。

単純 XML パーサーにおける文字エンコード

単純 XML パーサーをデプロイするときに使用するデフォルトの推奨文字エンコードは UTF-8 です。この文字エンコードを使用すれば、ほとんどの場合に XML データの整合性が保持されます。異なるエンコード方式を使用する必要がある場合、パーサーは各種のエンコード方式を提供された方式で処理します。

- パーサーは、ファイルを読み取る際に、以下の順にエンコードを探します。
  1. IBM Security Directory Integrator の「CharacterSet」構成パラメーターが設定されている場合、エンコードはこのパラメーターで指定されている値に設定されます。ただし、指定されているエンコードが UTF-32 または UTF-16 の場合は、検査 #2 が試行され、成功するとこの検査は上書きされます。
  2. XML 宣言に「encoding」属性が存在しているかどうかについて XML が検査されます。最初に、BOM が存在しているかどうかについて XML が検査されます。存在している場合は、その BOM で指定されているエンコードを使用して、XML 宣言からエンコード属性が取り出されます。存在していない場合

は、JRE のデフォルトのエンコードを使用してこの属性が取り出されます。XML 宣言の「encoding」属性が見つかった場合は、その値が使用されます。

3. IBM Security Directory Integrator CharacterSet が設定されておらず、XML 宣言に「encoding」属性が見つからなかった場合、BOM エンコードが設定されていればそのエンコードが使用されます。
4. 上記のいずれにも該当しない場合は、JRE のデフォルトのエンコードが使用されます。

- 出力時には、パーサーは、文字エンコードを指定した XML ヘッダーを書き込みます。これは、パーサーの構成で指定されているエンコード方式です。構成で何も指定されていない場合は、UTF-8 を使用します。

例

ここに記載されている例を使用することで、単純 XML パーサーについて詳しく知ることができます。

追加の指定変更フック

```
var root = xmldom.getDocumentElement();
var entry = xmldom.createElement ("entry");
var names = work.getAttributeNames();

for ( i = 0; i < names.length; i++ ) {
 xmlNode = xmldom.createElement ("attribute");
 xmlNode.setAttribute ( "name", names[i] );
 xmlNode.appendChild ( xmldom.createTextNode ( work.getString(
   names[i] ) ) );
 entry.appendChild ( xmlNode );
}
root.appendChild ( entry );
```

選択後フック

```
//
// Set up variables for "override getnext" hook
//

var root = xmldom.getDocumentElement();
var list = system.selectNodeList ( root, "//Entry" );
var counter = 0;
```

GetNext の指定変更フック

```
//
// Note that the Iterator hooks are NOT called when we override the
  getnext function
// Initialization done in After Select Entries hook


var nxt = list.item ( counter );

if ( nxt != null ) {
   var ch = nxt.getFirstChild();
   while ( ch != null ) {
      var child = ch.getFirstChild();
      while (child != null ) {
        // Use the grandchild's value if it exist, to be able to
    read multivalue attributes
 grandchild = child.getFirstChild();
 if (grandchild != null)
   nodeValue = grandchild.getNodeValue();
      else nodeValue = child.getNodeValue();
 // Ignore strings containing newlines, they are just fillers
      if (nodeValue != null && nodeValue.indexOf('¥n')
    == -1) {
            work.addAttributeValue ( ch.getNodeName(), nodeValue );
      }
 child = child.getNextSibling();
      }
```

```
      ch = ch.getNextSibling();
   }

   result.setStatus (1); // Not end of input yet
   counter++;
} else {
   result.setStatus (0); // Signal end of input
}
```

前の例では、以下のような項目を含むファイルが解析されます。

```
<DocRoot>
  <Entry>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <title>Engineer</title>
  </Entry>
  <Entry>
    <firstName>Al</firstName>
    <lastName">Bundy</lastName>
    <title">Shoe salesman</title>
  </Entry>
</DocRoot>
```

入力が上記のものではなく、以下のようなものであったとします。

```
<DocRoot>
  <Entry>
    <field name="firstName">John</field>
    <field name="lastName">Doe</field>
    <field name="title">Engineer</field>
  </Entry>
  <Entry>
    <field name="firstName">Al</field>
    <field name="lastName">Bundy</field>
    <field name="title">Shoe salesman</field>
  </Entry>
</DocRoot>
```

この場合は、属性名はフィールド・ノードの属性から検索できるため、**GetNext の指定変更**フックの中で次のようなコードを使用します。

```
var nxt = list.item ( counter );

if ( nxt != null ) {
 var ch = nxt.getFirstChild();
 while ( ch != null ) {
  if(String(ch.getNodeName()) == "field") {
   attrName = ch.getAttributes().item(0).getNodeValue();
   nodeValue = ch.getFirstChild().getNodeValue();
   work.addAttributeValue ( attrName, nodeValue );
  }
  ch = ch.getNextSibling();
 }

 result.setStatus (1); // Not end of input yet
 counter++;
} else {
 result.setStatus (0); // Signal end of input
}
```

このパッケージ例では、**GetNext の指定変更**フックおよび**追加の指定変更**フックを使用して、深度が 2 レベルを超える XML を読み取れるように基本の単純 XML パーサーの機能を拡張する方法を示しています。

関連情報

481 ページの『XML パーサー』,
493 ページの『XML SAX パーサー』,

496 ページの『XSL ベース XML パーサー』,
466 ページの『SOAP パーサー』,
431 ページの『DSMLv1 パーサー』.

XML パーサー

この XML パーサーは、IBM Security Directory Integrator v7.0 で初めて導入されました。ここに記載されている情報とリンクを使用することで、XML パーサーについて詳しく知ることができます。

このパーサーは StAX (JSR-173) 仕様の XLXP インプリメンテーションを使用します。 StAX はカーソル・ベースの XML パーサーで、XML の読み取りと書き込みの両方の能力を持っています。

注: IBM Security Directory Integrator の旧バージョンで使用されていた従来の DOM ベースのパーサーは名前が変更され、現在は 476 ページの『単純 XML パーサー』として使用できます。新しい XML パーサーは、古いコンポーネントの置き換えと見なされているので、古い構成をマイグレーションして、新規のパーサーを使用してください。

コネクターは XML パーサーを使用して、ソースの XML から IBM Security Directory Integrator 項目オブジェクトを取得するか、または IBM Security Directory Integrator 項目オブジェクトを XML として出力します。XML パーサーは内部ではカーソル・ベースの StAX パーサーを使用します。IBM Security Directory Integrator XML パーサーの旧バージョン (現在は単純 XML パーサーという名前になっています) では、XML の構文解析には DOM メカニズムが使用されていました。StAX インプリメンテーションの主要な利点は、IBM Security Directory Integrator パーサーは DOM のように XML 構造全体をメモリーにロードする必要がないため、はるかに高速であるという点です。StAX インプリメンテーションはメモリー効率がよいため、きわめて大きな XML 構造を IBM Security Directory Integrator ソリューションで扱うことが想定されている場合により適しています。

XML データを解析する際のこのメモリー効率のよいメカニズムの唯一の欠点は、ランダム・エレメント・アクセスを利用できないという点です。その理由は、すべての StAX は、XML 構造全体を走査して一度に 1 つのエレメントを取り出すからです。取り出された各エレメントは、IBM Security Directory Integrator XML パーサーの構成に応じて、スキップするか、XML から取り出される各エレメントを表す属性を持つ項目に挿入することができます。

構成

ここに記載されているパラメーターを使用することで、XML パーサーを構成できるようになります。

単純 XPath
エレメントを検出してそれらを項目として解釈するために使用される値 (XPath に類似した式) を含みます。このパラメーターは、書き込まれる XML 文書の構造を表示する場合にも使用されます。

項目タグ
XML パーサーに渡された各項目をラップするエレメントの名前を保持します。

値タグ XML パーサーに渡された各属性値をラップするエレメントの名前を保持します。

ネーム・スペース・マップの接頭部
パイプ文字 (|) で区切られた <prefix>=<namespace> の間のマッピング。接頭部が $ で始まっている場合、その接頭部はデフォルトのネーム・スペース宣言と見なされます。デフォルト値は「<prefix>=<namespace>」です。

XSD スキーマ・ロケーション
表示のみを目的として使用されるスキーマ・ロケーション。

文字エンコード
読み取りまたは書き込み時に使用される文字エンコード。デフォルトは UTF-8 です。 489 ページの『XML パーサーにおける文字エンコード』も参照してください。

静的属性宣言
属性および接頭部を宣言するために使用されます。属性および接頭部は、「**単純 XPath**」パラメーターから読み取られた静的エレメントとともに書き込まれます。これはテキスト域であり、デフォルトは以下のとおりです。

```
<!-- this is an example for statically declared XML attributes/namespaces -->
<!-- DocRoot xmlns="defaultNS" attr1="val2">
 <Entry xmlns:p1="p1NS" p1:attr2="val2" />
</DocRoot-->
```

読み取り中に反復 XML 宣言を無視
XML 宣言があれば、常に最初の XML 宣言を認知する場合にチェック・マークを付けます。これより後の他の宣言はすべて無視されます。デフォルトではチェックは外されています。

合体 これにチェック・マークが付けられている場合、パーサーは隣接する文字データ・セクションを合体します。デフォルトではチェックは外されています。

書き込み時に XML 宣言を省略
出力に XML 宣言を書き込まない場合は、これにチェック・マークを付けます。既存の XML ファイルに付加する場合に役立ちます。デフォルトではチェックは外されています。

複数ルート文書
これにチェック・マークが付けられている場合は、各項目はスタンドアロン・エレメントとして出力されます。これにより、複数ルート文書が作成されます。デフォルトではチェックは外されています。

インデント出力
このフィールドにチェック・マークを付けると、XML 出力がインデントされます。デフォルト値はチェック・マーク付きです。

書き込み時に無効な XML 文字を許可
このチェック・ボックスを選択すると、無効な XML 文字が XML タグに含まれます。選択していない場合は、XML 文書の書き込み中に例外がスローされます。

詳細ログ

より詳細なログ・メッセージを生成する場合は、これにチェック・マークを付けます。

パーサーの使用

XML パーサーを使用して、さまざまな操作を実行できます。ここに記載されている詳細を参照できます。

XML 構造内のナビゲーション

XML パーサーが認識するのは非常に単純な XPath 式です。パーサーは、この式に従って、エレメントそれ自体を表す単一の属性オブジェクトを含む項目、またはパーサーのラップ/アンラップ機能が利用されている場合には、複数の属性オブジェクトを含む項目を見つけてそれを返します。ここに記載されている情報を使用することで、ナビゲーションについて知ることができます。

現在の XPath のインプリメンテーションには、XPath 式で参照されるエレメントを正確に特定するための (オブジェクト・モデルでの) ランダム・エレメント・アクセスが必要です。StAX パーサーはこの機能 (ランダム・エレメント・アクセス) を備えていないため、以下のような単純 XPath 式でしか動作できません。

- /root/container1/container2/entry
- /root/prefix:container/entry
- /root/$prefix:container/
- /root/*/entry
- /root/prefix:*
- /root/$prefix:*/entry

読み取り時のナビゲーション:

XML の構造がきわめて複雑な場合は、いくつかの単純なパスを提供できます。

各 XPath 式は、パイプ文字 (「|」) を使用して前の式から分離されます。各式は XML 文書内でのエレメントの検索に使用されます。デフォルトでは XML パーサーは、単純 XML パーサーと同じように、深度が 2 レベルの XML を処理できます。さらに XML パーサーは、任意の深度の複雑な階層構造を操作するための方法も提供します。詳しくは以下の 2 つのセクションを参照してください。

単純 XML

これは XML を構文解析するためのデフォルトの方法です。単純 XML パーサーとまったく同様に、このパーサーは以下のような XML 構造を構文解析することができます。

```
<?xml version="1.0" encoding="UTF-8" ?>
<DocRoot>
    <Entry>
        <telephoneNo>
                <ValueTag>555-888-8888</ValueTag>
                <ValueTag>555-999-9999</ValueTag>
        </telephoneNo>
        <User>Jill Vox</User>
    </Entry>
</DocRoot>
```

単純モードの場合、XML パーサーは、単純なフラットなデータ構造 (項目) を返すために、階層内のエレメントの一部が取り除かれるようにします。このパーサーの振る舞いは、以下の 3 つのパラメーターによって制御されます。

1. **単純 XPath** (xpath.expr) フィールド –「entry.tag」パラメーターによって指定されたエレメントが存在しているか検索されるコンテナー・エレメントへのパスを指定する場合に使用されます。このフィールドは、デフォルトでは入力 XML のルート・エレメントを見つけるよう構成されています。
2. **項目タグ** (entry.tag) フィールド – 戻される項目を表すエレメントの名前を指定するために使用されます。

   注: このパラメーターは、パーサーが単純な構文解析を行うのか、拡張構文解析を行うのかを指定します。このパラメーターが空の場合、XML パーサーは拡張構文解析を実行します。
3. **値タグ** (value.tag) フィールド – 多値属性の 1 つの値を保持するエレメントの名前を指定するために使用されます。

   注: 「**entry.tag**」パラメーターが空の場合、このパラメーターは使用されません。

注: 「**xpath.expr**」パラメーターを「**ns.map**」パラメーターと一緒に使用すると、エレメントの一部をフィルタリングすることができます。詳しくは、『拡張 XML』セクションを参照してください。

XML パーサーは、これらのパラメーターのデフォルト値を使用して、上述の XML の例を簡単に構文解析することができ、以下のデータを含む項目が返されます。

```
{
 "telephoneNo": [
  "555-888-8888",
  "555-999-9999"
 ],
 "User": "Jill Vox"
}
```

ここでは「Entry」エレメントは除去されており、ValueTag エレメントは「telephoneNo」属性の値として取得されています。

注: 読み取り前に入力 XML の構造が不明の場合は、「**entry.tag**」パラメーターの値を除去できます。このようにして XML 全体が一度に読み取られ、XML 構造がどのようになっているのかが示されます。返された情報に基づき、XML 構造を突き合わせるようパーサーを再構成することができます。

拡張 XML

XML パーサーは、「entry.tag」パラメーターが空の場合にはこのモードで実行します。

見つかったエレメントごとに、単一の属性オブジェクトのみが作成されます。 XML パーサーは、サイクルごとに、XML 文書内で見つかったエレメントに対応する属性を 1 つのみ含む項目オブジェクトを返します。いずれかの XPath 式に一致するエレメントが XML 文書で見つからなかった場合、XML パーサーは null を返します。

パーサーが XML でデータを見つける方法を構成するパラメーターは 2 つあります。

1. 「**単純 XPath**」(xpath.expr) パラメーター – 希望するデータを含んでいるエレメントへのパスを指定するために使用されます。このパラメーターは必須です。
2. **ネーム・スペース・マップの接頭部** (ns.map) フィールド – 接頭部およびネーム・スペースを宣言するために使用されます。後で分かるように、これは必須パラメーターではありませんが、これにより、より柔軟に特定のデータを見つけることができます。

これらの 2 つのパラメーターを詳しく説明するために、次の例を考えてみましょう。

```
<?xml version="1.0" encoding="UTF-8" ?>
<root xmlns="defaultNS" xmlns:pref1="prefix1NS">
 <pref1:container xmlns:pref2="prefix2NS" attribute1="attrValue1" pref1:attribute2="attrValue2">
  <pref2:entryElement>
   <someData />
  </pref2:entryElement>
  <pref2:entryElement xmlns:pref2="prefix3NS">
   <moreData />
  </pref2:entryElement>
 </pref1:container>
</root>
```

取得する必要のある目的のデータは、いずれかの項目エレメントにあるものとします。各項目エレメントを最も簡単に取得するには、以下のエレメントを指定します。

```
xpath.expr: /root/container/entryElement
```

反復のたびに単一の entryElement が得られます。この例の場合、2 つの entryElement エレメントを取得するには 2 回の反復が必要になります。このエレメントの接頭部またはネーム・スペースを指定しなかった場合、パーサーは単純 XPath 式で提供されたローカル名を使用してエレメントを突き合わせます。

ただし、2 つの entryElement はそれぞれ別のネーム・スペースに属しているため、これらは別のものであることに気付きます。希望するデータは、「prefix3NS」ネーム・スペースに属する entryElement であるとします。前の構成を使用すると、得られるのは必要でないデータ (つまり、最初の entryElement) です。希望するエレメントがどこに属しているのかをパーサーに知らせる必要があるため、このような場合に ns.map を使用します。以下に、2 番目のエレメントのみを取得する方法を示します。

```
xpath.expr: /root/container/pref2:entryElement
ns.map: pref2=prefix3NS
```

ここでパーサーは、エレメントのローカル名 (すなわち、entryElelement) とネーム・スペースを突き合わせます。 ns.map フィールドで pref2 を指定しなかった場合、パーサーは突き合わせを行う際に xpath.expr 式で見つかった接頭部とローカル名のみを使用します。pref2 を ns.map で再定義すると、最新の定義が使用され、以前の定義は無視されます。

```
xpath.expr: /root/container/p1:entryElement | /root/container/p2:entryElement
ns.map: p1=prefix2NS | p2=prefix3NS
```

この場合、接頭部は無視され、エレメントのローカル名とネーム・スペースのみが考慮されます。

以下の式があるとします。

```
xpath.expr: /$defPref:root/container/pref2:entryElement
ns.map: pref2=prefix3NS | $defPref= defaultNS
```

この式の意味は 2 番目の構成例と同じです。ただし、式 $defPref は、ルート・エレメントはデフォルトのネーム・スペースである「defaultNS」に属していることをパーサーに示します。これが役立つのは、デフォルトのネーム・スペースが、ある場所にある XML で事前定義されている場合です。この例では、パーサーはローカル名とネーム・スペースしか突き合わせませんが、パーサーはそれが検査する XML エレメントは、デフォルトのネーム・スペース (すなわち、接頭部を持たないネーム・スペース) に属していることを期待します。以下の式を見てみます。

```
xpath.expr: /$defPref:root/$somePref:container/pref2:entryElement
ns.map: pref2=prefix3NS | $somePref=prefix1NS | $defPref= defaultNS
```

この式は、どの entryElement エレメントも返しません。

XML パーサーは、ワイルドカードを使用して XML ツリーをナビゲートすることができます。サポートされているワイルドカードは、アスタリスク文字 (「*」) です。このワイルドカード文字は、XML のあるエレメントのローカル名を置き換えるために使用されます。以下の構成が設定されているものとします。

```
xpath.expr: /root/container/*
```

この式は、ローカル名が「container」であるエレメントの下にある各エレメントを取り出すため、合計で 2 回の反復が行われます。この結果は、xpath.expr が「/root/container/pref2:*」に設定され、かつ pref2 が ns.map フィールドで定義されていない場合と同じです。

以下の構成があるとします。

```
xpath.expr: /root/container/p1*
ns.map: p1=prefix2NS
```

これは、container エレメントの下にあり、かつ prefix2NS ネーム・スペースに属するすべてのエレメントを取り出します。この場合、取り出されるエレメントは container エレメントの最初の子だけです。

注: 「*:localName」、「local*」、「pref:*Name」などのワイルドカード操作は実行できません。アスタリスク文字は、エレメントのローカル名にしか置き換わりません。

書き込み時のナビゲーション:

ここに記載されている情報とパスの例を使用することで、XML パーサーの書き込み時のナビゲーションについて詳しく知ることができます。

「**単純 XPath**」(xpath.expr) パラメーターの主な目的は、項目データを挿入する場所を指定することです。このパラメーターは、デフォルトでは単一のワイルドカード (「*」) に設定されています。デフォルト値が変更されていない場合、パーサーは DocRoot という名前のルート・エレメントを持つ XML を出力します。したがって、このパラメーターの値を除去して複数ルート文書にするか、またはアスタリスクを具体的な値で置き換えるかを選択することができます。

例えば、以下のパスが設定されているとします。

```
xpath.expr: /root/container/entry | /otherRoot/otherContainer/moreElements
```

この場合、パーサーは以下の構造を作成します。

```
<root>
 <container>
  <entry>
   /* The Entries go here. */
  </entry>
 </container>
</root>
```

ここで、root、container、および entry の各エレメントは静的です。その理由は、これらのエレメントは、パーサーに入力として渡されるどの項目にも属していないからです。これらの静的エレメントは、パーサーの構成に応じて、サイクルごとに書き込んで各項目をラップするか、またはすべての項目をラップすることができます。

注: 最初のパスのみが使用され、残りは無視されます。

XML パーサーが出力モードの場合に「**単純 XPath**」(xpath.expr) パラメーターでアスタリスクを使用すると、パーサーは最初のアスタリスクの前のパスのみを考慮します。例えば、式が以下のようであったとします。

```
xpath.expr: /root/container/*/entry
```

これは、次の式を指定したかのように見なされます。

```
xpath.expr: /root/container
```

このルールの例外となる唯一の式は以下のとおりです。

```
xpath.expr: *
```

これは、以下が指定されたかのように読まれます。

```
xpath.expr: DocRoot
```

この **xpath.expr** は、ルート・エレメントのみを構成するパラメーターと考えることができます。パーサーは、各項目出力を XML としてラップする単一エレメントを宣言することができます。このエレメントは、**entry.tag** フィールドで構成できます。このフィールドに値がない場合、各項目をラップするエレメントは出力されません。このパラメーターは、デフォルトではある値を持っているため、出力ストリームには追加のエレメントが書き込まれます。パーサーは、単純な多値属性の各値を含むエレメントの名前を構成するための便利なフィールドも提供しています。この名前は **value.tag** フィールドで構成することができ、デフォルトでは *ValueTag* に設定されています。しかし、この値が除去されていて、そのような属性を出力するようパーサーに要求があった場合には、各値が「value」という名前のエレメントに挿入されます。

注:

1. 「**value.tag**」パラメーターは、「**entry.tag**」パラメーターが空でない場合にのみ考慮されます。そのパラメーターが空の場合、多値属性の値はラップされません。
2. **entry.tag** も **value.tag** もワイルドカードをサポートしていません。ワイルドカードが提供された場合は、結果として例外がスローされます。

それらの静的エレメントでいくつかの属性また接頭部を宣言するために、**静的属性宣言** (static.decl) フィールドを使用できます。『拡張 XML』セクションで使用した XML を出力したい場合は、以下の構成を使用する必要があります。

```
xpath.expr: /root/pref1:container/
static.decl: <root xmlns="defaultNS" xmlns:pref1="prefixNS">
   <pref1:container xmlns:pref2="prefix2NS" attribute1="attrValue1" pref1:attribute2="attrValue2" />
 </root>
```

この例から、**static.decl** は、静的なルートに出力する必要のある属性とネーム・スペースを XML を使用してマークアップしていることが分かります。この XML 構造は、得られた xml 構造に一致する必要があることに注意してください。このフィールドには、項目タグ・エレメントに関する情報も含めることができます。

上の例でパラメーター

```
entry.tag: Entry
```

を追加すると、以下のようにそのレベルにいくつかの属性/ネーム・スペースを追加することができます。

```
static.decl: <root xmlns="defaultNS" xmlns:pref1="prefixNS">
 <pref1:container xmlns:pref2="prefix2NS" attribute1="attrValue1" pref1:attribute2="attrValue2" />
  <Entry xmlns:="otherDefaultNS" pref1:attribute3="attrValue3" />
 </pref1:container>
</root>
```

xpath.expr パスの各エレメントの場合とまったく同様に、接頭部を持つよう entry.tag と value.tag の両方を定義することができます。これら 2 つの違いは、value.tag エレメントの接頭部は、使用する前に定義する必要があるという点です。これを行うには、static.decl フィールドを使用するか、またはこのバージョンで提供されている階層項目構造を使用します。static.decl フィールドには entry.tag が含まれているため、現在のところ value.tag をこのフィールドに含めることはできません。

XML の読み取り

ここに記載されている情報を使用することで、XML の読み取りについて知ることができます。

パーサーは、項目に対する要求があるたびに、InputStream からデータを読み取り、そのデータを項目オブジェクトとして取り出します。XML で見つかった各エレメントは、`org.w3c.dom.Element` インターフェースをインプリメントする Attribute クラスで表されます。XML で見つかった各属性は、`org.w3c.dom.Attr` インターフェースをインプリメントする Property クラスで表されます。XML で見つかった各 CDATA は、`org.w3c.dom.CDATASection` インターフェースをインプリメントする AttributeValue クラスで表されます。

XLXP プロジェクトの StAX インプリメンテーションは DTD 検証も XSD 検証もサポートしていないため、検証を行うことはできません。

XML パーサーは、複数ルート XML 文書に複数の XML 宣言がない限り、複数ルート XML 文書を読み取ることができます。複数の XML 宣言がある場合は、「**読み取り時に XML 宣言の繰り返しを無視する (Ignore repeating XML declarations when reading)**」チェック・ボックスにチェック・マークが付けられている場合に限り、XML を読み取ることができます。ただし、この場合は、XML 宣言が繰り返されていないか文書が二重検査されるため、パーサーのパフォーマンスが影響を受けます。

注:

1. 「**読み取り時に XML 宣言の繰り返しを無視する (Ignore repeating XML declarations when reading)**」チェック・ボックスを有効にすると、(最初の XML 宣言を除く) すべての XML 宣言が無視されます。つまり、CDATA セクションに XML 宣言が含まれていても、その宣言は無視されます。これを回避するために、項目が取り出された後に CDATA セクションを手動で修正することができます。
2. XML パーサーは、『XML パーサーにおける文字エンコード』の説明に従って適切なエンコードを見つけようとします。

取り出された項目のストリング表現を使用することができ、`getCurrentEntryAsXMLString()` メソッドを使用してアクセスすることができます。

XML の書き込み

ここに記載されている情報を通じて、XML の書き込み方法を理解することができます。

パーサーは、項目オブジェクトを出力ストリームに書き込むよう要求されると、(単純 XML パーサーの振る舞いと同様にして) XML に組み込まれる各オブジェクトの `toString()` メソッドを呼び出します。各項目は出力ストリームに個別にフラッシュされ、システム障害の場合は、最後にフラッシュされた項目が安全に送信されます。

このパーサーは、**複数ルート文書**オプションにチェック・マークを付けるすることにより、各項目を個別のルートに出力することができます。これにより、各項目が独自の静的ルートを持つ複数ルート文書が生成されます。

XML を既存の XML に付加する場合は、「**XML 宣言を書き込まない (No XML declaration when writing)**」パラメーターにチェック・マークを付けると便利です。このパラメーターにチェック・マークを付けると (有効にすると)、通常は XML 文書の先頭に挿入される XML 宣言を省略するようパーサーが指示を受けます。

XML パーサーを使用して、XML エレメントに変換される項目属性名に無効な XML 文字がないかを検査できます。XML 文字を XML 文書に書き込む場合は、「**書き込み時に無効な XML 文字を許可**」パラメーターを使用可能にします。XML 仕様 (XML 1.0 標準) に従って、無効な XML 文字が項目属性内に見つかると、例外がスローされます。詳しくは、http://www.w3.org/TR/2000/REC-xml-20001006#NT-Char を参照してください。

XML タグ名では、以下は許可されません。

- 特殊文字 (! " # $ % & ' () * + , / ; < = > ? @ [¥] ^ ` { | } ~ など) およびスペース文字が含まれている
- 先頭文字が -、.、または数字である

XML パーサーにおける文字エンコード

ここに記載されているメソッドを使用して、XML パーサーでの文字エンコードを実行できます。

XML パーサーは、エンコードの名前を設定するために使用できる「**文字エンコード**」というパラメーターを持っています。このパラメーターを設定すると、このエンコードを使用して、初期化時にパーサーに渡された InputStream がデコードされます。このパラメーターがブランク (空ストリング) 以外の場合は、その値に関係なくこのパラメーターが使用されます。例えば、InputStream が UTF-16BE でエンコードされていて、先頭にバイト・オーダー・マーク (BOM) があり、「**文字エンコード**」パラメーターが「UTF-16BE」に設定されている場合、パーサーはその BOM シーケンスを認識することができるようになり、自動的にそれをスキップします。「**文字エンコード**」パラメーターが (InputStream のエンコードと互換性のない) 別のエンコードに設定されている場合は、不適切なエンコードが指定されていることを示す例外がスローされます。

InputStream またはファイルのエンコードが不明の場合は、パーサーにエンコードを見つけさせることができます (可能な場合)。以下に、エンコードが構成で明示的に指定されていない場合に (つまり、「**文字エンコード**」パラメーターが空の場合に) パーサーが XML のエンコードを見つけるために従う順序を示します。

1. パーサーは BOM がないかチェックします。BOM が見つかった場合、パーサーはその BOM によって提供される情報を使用して InputStream をデコードします。(BOM に基づいて) 認識可能なエンコードは、UTF-8、UTF-16LE、UTF-16BE、UTF-32LE、UTF-32BE です。

   注: パーサーは、UTF-32 のシーケンスに類似した、一般的でない (反転された) 4 バイト・シーケンスを認識しません。この場合は、(「**文字エンコード**」パラメーターを使用する) 明示的な構成が必要になります。
2. InputStream またはファイルが BOM シーケンスを提供しておらず、明示的な構成が設定されていない場合は、パーサーはエンコードを推測して、XML 宣言のエンコード属性値を読み取ろうとします。特定のエンコードを読み取るために、UTF-8、UTF-16BE、UTF-16LE、UTF-32BE、UTF-32LE、および IBM-1047 (EBCDIC の変種) の各エンコードが使用されます。エンコードが見つかった場合、その値を使用して InputStream またはファイルの残りがデコードされます。

   注: XML 宣言は文書の最初の行で設定する必要があり、最初の文字から開始しなければなりません。
3. 「**文字エンコード**」パラメーターが設定されていない場合で、BOM も XML 宣言も見つからないとき (または、XML 宣言に「encoding」属性がないとき) は、パーサーはデフォルトのエンコードである UTF-8 を使用します。

設計時にエンコードが分かっている場合は、XML パーサーの構成でそのエンコードを明示的に設定することをお勧めします。これにより、エンコードの検索が行われないため、パーサーの初期化プロセスのパフォーマンスが向上します。

パーサーは、書き込み用に初期化されるときに (出力モード)、「**文字エンコード**」パラメーターが明示的に割り当てられることを予期します。そのような割り当てが行われなかった場合、パーサーは UTF-8 をデフォルト・エンコードとして使用します (BOM シーケンスのない UTF-8)。BOM と互換性のある何らかのエンコードが明示的に指定されている場合 (UTF-8、UTF-16BE、UTF-16LE、UTF-32BE、UTF-32LE)、パーサーはストリームの先頭で BOM シーケンスを設定します。

例

ここに記載されているパスを使用して、XML パーサーの例にアクセスできます。

XML パーサーの機能を使用して、IBM Security Directory Integrator がどのようにさまざまな XML 文書を処理できるのかを示す例は、*TDI_install_dir*/examples/xmlparser2 内にあります。詳しくは readme.txt ファイルを参照してください。

XSD スキーマの使用

ここに記載されている情報を使用して、XSD スキーマを処理できます。

定義済み XSD スキーマ URI

IBM Security Directory Integrator XML パーサーの構成中に、XSD スキーマを指すパラメーターを使用することができます。

パーサーは、そのスキーマを要求されると、XSD からスキーマを読み取り、それを表示します。ただし、これには、パーサーが適切に構成されている必要があります。ナビゲーション・パスが設定されていない場合は、スキーマを取り出すことができません。その場合は、項目を読み取ってサンプル・スキーマを見つけることができます。

XSD が提供されていない

ここに記載されている情報を通じて、XSD が提供されていない場合の XML パーサーの操作について学習することができます。

XSD は提供されていないものの、パーサーの構成が適切な場合 (つまり、ナビゲーション・パスが設定されている場合)、パーサーは XML からスキーマのロケーション情報を抽出しようとします。見つかったすべてのスキーマは、目的のエレメントのスキーマがないか検査されます。XML 文書内にスキーマが見つからなかった場合、返された項目の内容を表示するために、項目を読み取る (つまり、XML の一部を読み取る) 必要があります。これがすべてのスキーマ照会に対するデフォルトの振る舞いです。ただし、返された項目の構造は、次のサイクルで読み取られる別の項目と同じではない場合があります。つまり、表示されるスキーマが完全である、あるいは有効であることは保証できません。

スキーマの構成

ここに記載されている情報を使用してスキーマを構成できます。

目的のエレメントのスキーマを表示するには、そのエレメントへのパス、およびそのエレメントに対応するスキーマへのパスを構成する必要があります (スキーマへのパスはオプションです。『XSD が提供されていない』を参照してください)。複数のエレメントまたはスキーマ・パス (あるいはその両方) がある場合は、すべてのパスを垂直バー (「|」文字) で区切る必要があります。スキーマ・パスが入力されているかどうかに関係なく、パーサーは XML 文書内にスキーマ・ロケーションがないかを検査します。XML から抽出された最初のスキーマが主スキーマとして選択されます。スキーマが返されない場合は、ユーザーが構成した最初のスキーマが主スキーマになります。

スキーマがパーサー構成で構成されている場合は、対応するネーム・スペースでそのスキーマを宣言できます。構成は以下のとおりです。

```
namespace1 schema1 | namespace2 schema2 | noNamespaceSchema | ...
```

ネーム・スペースは、スキーマ・ファイルが XSD スキーマ内のどのタイプに属しているのかを決定する場合に使用されます (そのことがスキーマそれ自体で指定されている場合)。

主スキーマの利点

これが、入力したエレメントの有無を検査される最初のスキーマになります。主スキーマに情報が見つからなかった場合は、入力された、または XML から抽出された他のスキーマが検査されます。主スキーマには、入力した XPath のルート・エレメントを含むスキーマを使用することをお勧めします。

スキーマの構文解析に使用されるライブラリーは、XSD スキーマの作成時には速度が遅くなります。この理由から、すべてのパスはマップ内に保持され、初めてスキーマが必要になったときにスキーマが作成され、後でそれが必要になる場合に備えてマップに保持されます。

結果

エレメントのパスに入力されたエレメントごとに項目が作成されます。各項目には、「Name」と「Type」という 2 つの属性が含まれます。「Name」は、照会するエレメントの名前です。「Type」は、スキーマ内で見つかったエレメントの対応するタイプです。スキーマ内で見つかったタイプがプリミティブ・タイプの場合は (つまり、提供されているスキーマにそのタイプに対応する定義が見つからない場合は)、その名前が「Type」属性の値になります。見つかったタイプがプリミティブ・タイプでない 場合は (すなわち、提供されているスキーマにそのタイプに対応する定義がある場合は)、名前 (見つかったすべてのエレメントと属性の名前) および値 (対応する属性またはエレメントのタイプ) を持つ属性を含む新規の項目が、「Type」属性の値として挿入されます。この場合も、タイプがプリミティブでない場合は、まったく同じようにして新規の項目が作成されて入力されます。これにより、項目はツリー状の構造として現れます。

すべてのエレメントのスキーマが見つかると、作成された項目が Vector オブジェクトに挿入され、このオブジェクトがスキーマ照会の結果として返されます。

注: このスキーマは、現在の構成エディターではフラットに表示されます。スキーマの照会機能のテスト目的にはこれで十分です。

インディケーター

- 順序インディケーター – スキーマ・インディケーターには、**すべて**、**選択**、および**シーケンス**という 3 つの可能性があります。これらのインディケーターの 1 つがスキーマ・ファイル内で見つかると、「#indicator」という名前のプロパティーが結果エントリーで設定されます。このプロパティーの値は、スキーマで見つかったインディケーターです。項目内部の情報は、このインディケーターに従っていなければなりません。
- 出現インディケーター – これらのインディケーターは、対応するエレメントの属性として設定されます。以下の「属性」セクションを参照してください。

属性

スキーマ・エレメントに属性が含まれている場合、それらの属性は、そのエレメントのスキーマの書き込み先の項目に対応するプロパティー「#attributes」内のマップに保持されます。

XSD スキーマの例

ここに記載されている XSD スキーマの例を参照できます。

スキーマ名: Order.xsd

スキーマ・パス: /path/Order.xsd

Order.xsd の内容:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            xmlns="urn:nonstandard:XSD_Schema"
            targetNamespace="urn:nonstandard:XSD_Schema" xmlns:stako="Stako">
    <xsd:element name="order" type="Order" />
    <xsd:complexType name="Order">
        <xsd:all>
            <xsd:element name="user" type="User" minOccurs="1" maxOccurs="1" />
            <xsd:element name="products" type="Products" minOccurs="1" maxOccurs="1" />
        </xsd:all>
    </xsd:complexType>
    <xsd:complexType name="User">
        <xsd:all>
            <xsd:element type="xsd:string" name="deliveryAddress" />
            <xsd:element name="fullname">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength value="30" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
        </xsd:all>
    </xsd:complexType>
    <xsd:complexType name="Products">
        <xsd:sequence>
            <xsd:element name="product" type="Product" minOccurs="1" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Product">
        <xsd:attribute name="id" type="xsd:long" use="required" />
        <xsd:attribute name="quantity" type="xsd:positiveInteger" use="required" />
    </xsd:complexType>
</xsd:schema>
```

ユーザーおよび製品スキーマを表示するためのパーサーの構成:

```
Simple XPath: /Order/User | /Order/Products
XSD Schema Location: /path/Order.xsd
```

結果

```
[[Name:user, Type:[fullname:[xsd:string:[xsd:maxLength:30]], deliveryAddress:string]],
 [Name:products, Type:[product:[quantity:xsd:positiveInteger, id:xsd:long]]]]
```

これは、結果として戻される Vector の `toString()` メソッド表現です。すべて 1 行で表示されます。

XML SAX パーサー

XML SAX パーサーは、Apache Xerces ライブラリーを基にしています。このパーサーを使用して、DOM ベースの XML パーサーがメモリーの制約のために処理できないような大きいサイズの XML 文書を読み取ることができます。

このパーサーは、構成で指定された「グループ・タグ」で囲まれたデータを取り出して 1 つの項目を作成し、データ内にある属性を設定します。複数のグループ・タグを指定する場合は、各タグ名をコンマで区切ります。これにより、SAX パーサーは、指定されたすべてのタグで中断します。複数のグループ・タグを指定している場合、SAX パーサーは first-in-win アプローチを使用します。このアプローチでは、最初に検出されたグループ・タグが、そのグループを閉じるタグになります。例えば、グループ・タグとして A と B があり、文書の構造が、B は A の子となっている場合、A がその項目を閉じるタグになります (A は B よりも前に見つかっており、優先されるためです)。

あるグループ・タグが見つかった場合、その下にグループ・タグがネストして出現しても、それらのタグは現在の項目には影響を与えません。

定義されているグループ・タグがない場合は、XML 文書全体が単一の項目として戻されます。

項目の属性名は、タグ名を分離文字「@」で囲んだ形で作成されます。例えば、次のような XML ファイルについて考えてみます。

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRoot>
 <Entry>
  <Company>
   <Name incorporated="yes">IBM Corporation</Name>
   <Country>USA</Country>
  </Company>
 </Entry>
 <Entry>
  <Company>
   <Name incorporated="no">Smith Brothers</Name>
   <Country>USA</Country>
  </Company>
 </Entry>
</DocRoot>
```

「Entry」をグループ・タグとして使用すると、上記の XML 文書から次の 2 つの項目が生成されます。

項目 1

```
Attribute name: DocRoot@Entry@Company@Name
Attribute value: IBM Corporation
Attribute name: DocRoot@Entry@Company@Name#incorporated
Attribute value: yes
Attribute name:DocRoot@Entry@Company@Country
Attribute value: USA
```

項目 2

```
Attribute name: DocRoot@Entry@Company@Name#incorporated
Attribute value: Smith Brothers
Attribute name: DocRoot@Entry@Company@Name#incorporated
Attribute value: no
Attribute name:DocRoot@Entry@Company@Country
Attribute value: USA
```

構成の中で「接頭部の除去」の値を指定すれば、属性名を短縮できます。例えば、上の例で、「接頭部の除去」の値として「DocRoot@Entry@Company」を指定すると、項目に含まれる属性は次のようになります。

```
Attribute name: Name
Attribute value: IBM Corporation
Attribute name: Name#incorporated
Attribute value: yes
Attribute name: Country
Attribute value:  USA
...
```

コネクターが初期化されると、XML パーサーは、DTD タグが存在するかどうか、文書タイプ定義 (DTD) 検査を実行しようとします。パーサーは、多値属性も読み取ります。ただし、データを「スキーマ」タブでブラウズしているときに表示されるのは、多値属性のうち 1 つのみです。

XML ファイルに項目タグのネストがある場合、最も外側の項目タグに囲まれたすべての項目タグは、通常の XML タグとして扱われます。例えば、

```
<entry>
 <entry>
  <company>IBM</company>
 </entry>
</entry>
```

この場合、項目には次のような属性が組み込まれます。

```
attribute name: entry@entry@company
attribute value: IBM
```

構成

ここに記載されているパラメーターを使用することで、XML SAX パーサーを構成できるようになります。

グループ・タグ
: 項目を囲む XML グループ・タグの名前。複数のタグは、各タグ名をコンマで区切って指定します。または、このパラメーターが指定されていない場合 (および XML 文書全体が単一の項目として戻される場合) は、ルート・タグを使用します。

接頭部の除去
: 属性名から除去する接頭部を指定します。

属性を無視
: グループ・タグとその子の属性を無視するようにパーサーに依頼します。

文字エンコード
: 使用される文字エンコード。デフォルトは UTF-8 です。 496 ページの『文字エンコード』も参照してください。

文書の妥当性検査
: 使用されている DTD/XSchema に基づく妥当性検査を要求するには、このフィールドをチェックします。

XSD 検証の使用
: このフィールドにチェック・マークを付けた場合は、XML ファイルの検証に DTD ではなく XSD が使用されます。

ネーム・スペースを意識
: XML パーサーにネーム・スペースを意識させるには、このフィールドをチェックします。

読み取りタイムアウト
: データを何も受け取らない場合にパーサーが停止するまでの秒数。

詳細ログ
: このフィールドをチェックすると、追加のログ・メッセージが生成されます。

文字エンコード

文字エンコードについては、ここに記載されている情報を通じて知ることができます。

XML SAX パーサーをデプロイするときに使用するデフォルトの推奨文字エンコードは UTF-8 です。この文字エンコードを使用すれば、ほとんどの場合に XML データの整合性が保持されます。異なるエンコード方式を使用する必要がある場合、パーサーは各種のエンコード方式を次のようにして処理します。

パーサーは、ファイルを読み取る際に、以下の順にエンコードを探します。

1. パーサーの「CharacterSet」構成パラメーターが設定され、UTF-8 に設定されていない場合、エンコードはこのパラメーターで指定されている値に設定されます。ただし、指定されているエンコードが UTF-32 または UTF-16 の場合は、検査 #2 が試行され、成功するとこの検査は上書きされます。
2. XML 宣言に「encoding」属性が存在しているかどうかについて、解析中の XML が検査されます。XML 宣言の「encoding」属性が見つかった場合は、その値が使用されます。
3. 上記のいずれにも該当しない場合は、JRE のデフォルトのエンコードが使用されます (通常は UTF-8)。

関連情報

481 ページの『XML パーサー』,

476 ページの『単純 XML パーサー』,

『XSL ベース XML パーサー』.

XSL ベース XML パーサー

XSL ベースの XML DOM パーサーを IBM Security Directory Integrator で利用すると、任意のフォーマットの XML 文書を、ユーザーから提供された XSL を使用して解析し、属性と値のペアとして項目オブジェクトに格納することができます。ここに記載されている情報を通じて、プロセスについて詳しく知ることができます。

XSL ベースのパーサーは、すべての種類の XML フォーマットの読み取りを容易に行うために必要です。特に、XML のうち特定の部分のみを必要とするユーザーは、必要な部分を選別するための XSL を記述できます。このパーサーは、入力 XML および IBM Security Directory Integrator 内部フォーマットを表現するためにメモリー内に解析ツリーを作成します。入力 XML から生成した DOM 文書を XSL を利用して変換し、IBM Security Directory Integrator 内部フォーマット用の出力 DOM を生成します。このパーサーは、変換を実行するために javax 変換ライブラリーを使用します。

構成

ここに記載されているパラメーターを使用して、XSL ベースの DOM XML パーサーを構成できます。

入力 XSL ファイルを使用します
入力 XSL ファイルを使用するか、「入力 *XSL*」フィールドにキー入力した XSL を使用するかを指定するチェック・ボックス。

入力 XSL ファイル名
ユーザー XML を IBM Security Directory Integrator 内部フォーマットに変換するためのテンプレート・マッチング・ルールが指定された入力 XSL ファイル。

入力 XSL
編集可能な領域。入力 XSL 全体をユーザーがキー入力するか、貼り付けることができます。

出力 XSL ファイルを使用
出力 XSL ファイルを使用するか、「*出力 XSL*」フィールドにキー入力した XSL を使用するかを指定するチェック・ボックス。

出力 XSL ファイル名
IBM Security Directory Integrator の内部フォーマットをユーザー XML に変換するためのテンプレート・マッチング・ルールを持つ出力 XSL ファイル。

出力 XSL
編集可能な領域。出力 XSL 全体をユーザーがキー入力するか、貼り付けることができます。

文字エンコード
読み取りまたは書き込みの際に使用する文字エンコード。デフォルトは UTF-8 です。

このパーサーは単純 XML パーサーを拡張したものであるため、文字エンコードに関して同じ注意事項が適用されます。

XML 宣言の省略
出力ストリーム内の XML 宣言ヘッダーを省略するには、これをチェックします。

文書の妥当性検査
DTD/XSchema による妥当性検査を XML パーサーに要求するには、これをチェックします。

ネーム・スペースを意識
XML パーサーにネーム・スペースを意識させるには、これをチェックします。

インデント出力
これをチェックすると、出力が適切にインデントされるため、読みやすくなります。出力を別のプログラムによって処理する場合は、このオプションをオフにすることをお勧めします。

詳細ログ
詳細なデバッグ情報をログに書き込むかどうかを指定します。

パーサーの使用

ここに記載されている情報を通じて、XSL ベースのパーサーを使用することができます。

このパーサーは、イテレーター または *AddOnly* モードのファイル・システム・コネクターで使用できます。XSL ベース DOM XML パーサーでは、ユーザーが以下の事項を指定する必要があります。

- 入力 XSL ファイル (イテレーター・モードのファイル・システム・コネクターで使用する場合)。XML を IBM Security Directory Integrator 内部フォーマットに変換するために使用されます。
- 出力 XSL ファイル (AddOnly モードのファイル・システム・コネクターで使用する場合)。IBM Security Directory Integrator 内部フォーマットを元のフォーマットに変換するために使用されます。

XSL 変換では、XSLT プロセッサーが XML 文書と XSLT スタイル・シートの両方を読み取ります。プロセッサーは、XSLT スタイル・シートにある指示に基づいて、新しい XML 文書、または XML 文書の断片を出力します。パーサーは、確実性を高めるために、XSL ファイルについて基本的な検証を実行します。また、パーサーは、コネクターから提供されたファイルの文書およびネーム・スペースについて妥当性検査を実行することもできます (オプション)。このパーサーは、ファイル・システム・コネクターとともに使用できます。このパーサーは、入力および出力 XSL に指定されているフォーマットで XML ファイルの読み取りと書き込みが可能であるという意味で、読み取りと書き込みの両方をサポートします。また、次の妥当性検査がオプションとして提供されています。

- 文書の妥当性検査
- ネーム・スペースを意識

IBM Security Directory Integrator 内部フォーマット

ここに記載されている例を通じて、IBM Security Directory Integrator 内部フォーマットの使用方法を理解することができます。

```
<DocRoot>
 <Entry>
  <attribute_name>
   <value_tag>attribute_value</value_tag>
   <value_tag>attribute_value</value_tag>
   <value_tag>attribute_value</value_tag>
  </attribute_name>
  <attribute_name>
   <value_tag>attribute_value</value_tag>
  </attribute_name>
     -
     -
     -
 </Entry>
 <Entry>
     -
     -
     -
 </Entry>
 -
</DocRoot>
```

例

ここに記載されている例を使用することで、XSL ベースの XML パーサーについて詳しく知ることができます。

入力 xml: birds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Class>
<Order Name="TINAMIFORMES">
        <Family Name="TINAMIDAE">
            <Species Scientific_Name="Tinamus major">  Great Tinamou.</Species>
            <Species Scientific_Name="Nothocercus">Highland Tinamou.</Species>
            <Species Scientific_Name="Crypturellus soui">Little Tinamou.</Species>
            <Species Scientific_Name="Crypturellus cinnamomeus">
      Thicket Tinamou.</Species>
            <Species Scientific_Name="Crypturellus boucardi">Slaty-breasted
      Tinamou.</Species>
            <Species Scientific_Name="Crypturellus kerriae">Choco Tinamou.</Species>
        </Family>
    </Order>
<Order Name="GAVIIFORMES">
        <Family Name="GAVIIDAE">
            <Species Scientific_Name="Gavia stellata">Red-throated Loon.</Species>
            <Species Scientific_Name="Gavia arctica">Arctic Loon.</Species>
            <Species Scientific_Name="Gavia pacifica">Pacific Loon.</Species>
            <Species Scientific_Name="Gavia immer">Common Loon.</Species>
            <Species Scientific_Name="Gavia adamsii">Yellow-billed Loon.</Species>
        </Family>
    </Order>
</Class>
```

入力 XSL: birds.XSL

```
<?xml version="1.0" ?>
<XSL:stylesheet xmlns:XSL="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <XSL:output method="xml" indent="yes" />
<XSL:template match="Class">
 <DocRoot>
  <XSL:for-each select="Order">
     <XSL:variable name="order"><XSL:value-of select="@Name" />
     </XSL:variable>
     <XSL:for-each select="Family">
     <Entry>
      <Attribute name="Order">
   <Value><XSL:value-of select="$order" /></Value>
  </Attribute>
     <Attribute name="Family">
   <Value><XSL:value-of select="@Name" /></Value>
     </Attribute>
     <Attribute name="Species">
      <XSL:for-each select="Species">
             <Value><XSL:value-of select="." /></Value>
      </XSL:for-each>
     </Attribute>
     </Entry>
  </XSL:for-each>
    </XSL:for-each>
 </DocRoot>
</XSL:template>
</XSL:stylesheet>
```

birds.xsl は、birds.xml を IBM Security Directory Integrator 内部フォーマットに変換します。その内部フォーマットから、属性と値のペアを設定した項目オブジェクトを形成できます。

関連情報

476 ページの『単純 XML パーサー』

「XML Bible」(XSL に関するセクション)
http://www.ibiblio.org/xml/books/bible2/sections/ch17.html

W3C Document Object Model
http://www.w3.org/DOM/

Effective XML processing with DOM and XPath in Java
http://www.ibm.com/developerworks/xml/library/x-domjava/

ユーザー定義パーサー

IBM Security Directory Integrator に既に組み込まれているパーサーの他に、ユーザーは独自のパーサーを作成してシステムに追加することができます。

ユーザー定義パーサーの一例として、正規表現の解析が可能なパーサーが Examples ディレクトリーに提供されています。ご使用の IBM Security Directory Integrator インストール済み環境の *TDI_install_dir*/examples/regexp_parser ディレクトリーに移動します。このサンプルは、Java で作成されました。

ユーザー定義パーサーのもう一つの例は、*TDI_install_dir*/examples/script_parser にあります。これは、スクリプトを使用してパーサーを作成する方法を示しています。このサンプルの詳細については、「」の『JavaScript パーサー』を参照してください。

第 4 章 関数コンポーネント

以下に示す情報を使用して、関数コンポーネントについて理解することができます。

関数コンポーネント (FC) は、コネクターおよびパーサーと並んで、IBM Security Directory Integrator を構成する、もう 1 つのタイプのビルディング・ブロックです。関数コンポーネントは、スコープの面ではコネクターに類似していますが、コネクターはデータ・ソースに固有のものであるのに対して、関数コンポーネントはデータ・ソースに依存しないという点で異なります。さらに正確に言えば、関数コンポーネントは、カスタム・ロジックや外部メソッドを簡単にラップできる AssemblyLine コンポーネントであり、構成エディター (CE) 内では分かりやすくて「コネクターに類似した」ユーザー・インターフェースを提供します。

また、関数コンポーネントにはモードがありません。つまり、AssemblyLine 内で関数コンポーネントを構成するとき、動作するモードを指定する必要がありません。関数コンポーネントの機能は、AssemblyLine から関数コンポーネントの `perform()` メソッドを呼び出したときに実行されます。

以降で説明するコンポーネントの多くは、IBM Security Directory Integrator のモジュラー Web サービス・アーキテクチャーの中で完全な Web サービス・ソリューション (クライアント・サイドとサーバー・サイドの両方) を構築する手段を提供します。

IBM Security Directory Integrator で提供されている関数コンポーネントは以下のとおりです。

- 513 ページの『AssemblyLine 関数コンポーネント』
- 549 ページの『Axis EasyInvoke Soap WS 関数コンポーネント』
- 543 ページの『Axis2 WS クライアント関数コンポーネント』
- 530 ページの『Axis Java から Soap への変換関数コンポーネント』
- 541 ページの『Axis Soap から Java への変換関数コンポーネント』
- 502 ページの『Castor Java to XML 関数コンポーネント』
- 504 ページの『Castor XML to Java 関数コンポーネント』
- 521 ページの『CBE 関数コンポーネント』
- 651 ページの『IdML のクローズ関数コンポーネント』
- 552 ページの『複素数タイプ・ジェネレーター関数コンポーネント』
- 554 ページの『デルタ関数コンポーネント』
- 573 ページの『ファイル転送関数コンポーネント』
- 587 ページの『SAP ABAP Application Server の関数コンポーネント』
- 537 ページの『InvokeSoap WS 関数コンポーネント』
- 659 ページの『IT レジストリーの初期化関数コンポーネント』
- 517 ページの『Java クラス関数コンポーネント』
- 528 ページの『メモリー・キュー関数コンポーネント』

- 518 ページの『パーサー関数コンポーネント』
- 557 ページの『リモート・コマンド行関数コンポーネント』
- 519 ページの『スクリプト記述関数コンポーネント』
- 510 ページの『SDOToXML 関数コンポーネント』
- 525 ページの『SendEMail 関数コンポーネント』
- 534 ページの『WrapSoap 関数コンポーネント』
- 507 ページの『XMLToSDO 関数コンポーネント』
- 510 ページの『SDOToXML 関数コンポーネント』

Castor Java to XML 関数コンポーネント

Castor はオープン・ソースのデータ・バインディング・フレームワークです。このコンポーネントを使用すると、XML 文書に定義されているデータに、オブジェクト・データ・モデルを通してアクセスできるようになります。

各種の XML ソリューション (例えば Web サービス) で要件となることが多いものとして、複素数データ・タイプやカスタム・データ・タイプの処理があります。

IBM Security Directory Integrator には、Java から XML へのバインディング機能と、XML から Java へのバインディング機能が内蔵されていますが、この機能を使用すると、Web サービス・ツールキットからは独立して、複素数データ・タイプやカスタム・データ・タイプを処理することができます。これは特に、各種の Web サービス・ツールキットにおけるバインディングの制限に対処する手段として役立ちます。

Castor は、ほとんどすべての「Bean 型の」Java オブジェクトを XML との間でマーシャルできます。マーシャル/アンマーシャルのプロセスでは Castor のデフォルトのイントロスペクション・モデル (Java のリフレクションに基づくインプリメンテーションで、Castor はこれを利用してデータをマーシャルおよびアンマーシャルする方法を決定する) を使用できますが、マッピング・ルールを定義している castor XML マッピング・ファイルを使用してこのプロセスを制御およびカスタマイズすることも可能です。

IBM Security Directory Integrator の観点からは、ユーザーが XML マッピング・ファイルを作成して、カスタム・データを XML との間でどのようにマップするのかを指定できます。

Castor を利用すると、特に Castor 向けに設計されていない XML 文書でも、処理対象以外の XML 部分をスキップすることにより、処理が可能になります。ここで問題となる制限事項は、Castor では、ある XML ノードをスキップした場合に、スキップしたその XML ノードのサブツリーに属するノードも処理できなくなるということです。この制限があるために、大規模で複雑な XML 文書からランダムな部分を抽出する場合 (現実的に予想されるケースです) に非常に不便です。その理由から、IBM Security Directory Integrator の Castor 関数コンポーネントは、XPath 照会を利用して XML の特定の部分を指定する機能を提供しています。

CastorJavaToXML 関数コンポーネントは、Castor 0.9.5.4 を使用します。Castor ライブラリーの文書と情報は、Castor プロジェクトの Web サイト (http://www.castor.org/) から入手できます。

構成

ここに記載されているパラメーターを使用することで、Castor Java To XML 関数コンポーネントを構成できるようになります。

パラメーター

Castor マッピング・ファイル
Java オブジェクトまたは*項目* オブジェクトを XML にシリアライズする方法を定義する XML マッピング・ファイル (Castor 構文によって定義する)。

このパラメーターで指定されるマッピング・ファイルには、*TDI_install_dir*/etc/di_castor_mapping.xml ファイルで定義されているマッピング・ルールを常に組み込む必要があります。したがって、このパラメーターの値として「etc/di_castor_mapping.xml」を指定するか、指定したマッピング・ファイルにそれらのルールが確実に含まれるようにする (例えば、Castor の include 節を使用して di_castor_mapping.xml のルールを別のマッピング・ファイルに組み込む) ことが必要です。

XML のルート・エレメント
生成される XML のルート・エレメントの名前。これを空のままにすると、ルート・エレメントの名前は「Entry」になります。

属性名の使用
これをチェックすると、属性の名前が XML エレメント名として使用されます。それ以外の場合は、XML エレメントはマッピング・ファイルで指定された名前になります。このパラメーターは、項目モードでのみ有効です。

XML の戻りタイプ
このドロップダウン・リストでは、戻りタイプを指定します。この指定は、入力オブジェクトが*項目* タイプのオブジェクト以外である場合に有効です。有効な値は「**String**」および「**DOMelement**」です。

詳細ログ
追加のログ・メッセージを生成する場合は、チェックします。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの使用

ここに記載されている情報を参照して、このコンポーネントの使用方法について理解することができます。

CastorJavaToXML 関数コンポーネントは、Java オブジェクトまたは*項目* オブジェクトから XML 文書を作成します。

この関数コンポーネントは、*項目* オブジェクトおよびカスタム Java オブジェクトの両方について動作可能です。

関数コンポーネントに入力として*項目* オブジェクトを渡すと、*項目* オブジェクトが戻されます。この動作モードのことを、**項目**モードといいます。

項目オブジェクトではない Java オブジェクトを渡すと、この関数コンポーネントは Castor のシリアライゼーション機能を使用して、渡されたオブジェクトをシリアライズし、それが結果の XML になります。この動作モードのことを、**非項目**モードといいます。

項目モード

- **項目**モードでは、入力として渡した項目オブジェクトの各属性がマーシャルされて、結果の XML エレメントのルートの下に配置されます。
- 「**XML の戻りタイプ**」パラメーターを「**DOMelement**」に設定した場合、結果の項目オブジェクトには *xmlDOMElement* という名前の属性が 1 つ組み込まれ、その値は `org.w3c.dom.Element` オブジェクトとしてマーシャルされた XML エレメントです。
- 「**XML の戻りタイプ**」パラメーターを「**String**」に設定した場合、結果の項目オブジェクトには *xmlString* という名前の属性が 1 つ組み込まれ、その値は `java.lang.String` オブジェクトとしてマーシャルされた XML エレメントです。

非項目モード

- 「**XML の戻りタイプ**」パラメーターを「**DOMelement**」に設定した場合、結果の XML エレメントは `org.w3c.dom.Element` オブジェクトとして戻されます。
- 「**XML の戻りタイプ**」パラメーターを「**String**」に設定した場合、XML エレメントは `java.lang.String` オブジェクトとして戻されます。

Castor XML to Java 関数コンポーネント

ここに記載されている情報とリンクを使用することで、Castor XML To Java 関数コンポーネントについて詳しく知ることができます。

CastorXMLtoJava 関数コンポーネントは 502 ページの『Castor Java to XML 関数コンポーネント』とミラー・イメージの関係にあり、そのセクションの内容がここにも同じように当てはまります。

具体的には、CastorXMLtoJava 関数コンポーネントは、XML 文書から*項目* オブジェクトまたは一般的な Java オブジェクトを作成します。また、XML 文書を非シリアライズする際に XML ツリーの特定の部分からデータを取得するオプションも提供しています。

この関数コンポーネントは、XML ノード/サブツリーから Java オブジェクト (カスタム Java クラスも可) を作成する方法を指定する Castor のマッピング・メカニズムに加えて、XML 文書から項目属性にデータを取り込む方法を指定する独自のロジックも提供しています。XPath 照会を使用することにより、XML 文書のどの部分を Castor API に渡して非シリアライズするかを指定できます。

このアプローチは、IBM Security Directory Integrator のコンテキストで使用が容易というだけでなく、カスタムの XML 文書 (例えば、他のシステムによって生成された XML 文書) を処理する際に、より強力な機能を提供します。XPath 照会を利

用すると、XML のどの部分を処理対象にするかを指定する (また、それらの部分を項目属性に直接取得する) ことや、どの部分が処理に無関係で、処理する必要がないかを指定することが可能になります。また、Castor の XML マッピング・ファイルを簡単に記述できます。XML 文書内で処理対象の部分についてのみマッピング・ルールを記述し、XML 文書全体についてルールを記述する必要はないためです。

CastorXMLToJava 関数コンポーネントは、Castor 0.9.5.4 を使用します。Castor ライブラリーの文書と情報は、Castor プロジェクトの Web サイト (http://www.castor.org/) から入手できます。

構成

ここに記載されているパラメーターを使用して、Castor XML To Java 関数コンポーネントを構成することができます。

パラメーター

Castor マッピング・ファイル

XML を Java オブジェクトにマッピングする方法を定義する XML マッピング・ファイル (Castor 構文によって定義されます)。

このパラメーターで指定されるマッピング・ファイルには、*TDI_install_dir*/etc/di_castor_mapping.xml ファイルで定義されているマッピング・ルールを常に組み込む必要があります。したがって、このパラメーターの値として「etc/di_castor_mapping.xml」を指定するか、指定したマッピング・ファイルにそれらのルールが確実に含まれるようにする (例えば、Castor の include 節を使用して di_castor_mapping.xml のルールを別のマッピング・ファイルに組み込む) ことが必要です。

属性の指定

<AttributeName>,<XPath query>[,<type>] の形式で、各行に 1 つの属性を指定します。このパラメーターは、関数コンポーネントを項目モードで使用する場合にのみ有効です。ここで、

<AttributeName>
項目属性の名前を指定します。

<XPath query>
XML のどの部分をアンマーシャルして、この属性に値として割り当てるかを指定します。

<type> 属性のタイプが複合 Java クラスではなく、次のいずれかの基本データ・タイプである場合は、常に指定する必要があります。*string*、*date*、*boolean*、*integer*、*long*、*double*、*float*、*big-decimal*、*byte*、*short*、*character*、*strings* (ストリングの配列)、*chars* (文字の配列)、*bytes* (バイトの配列)。これらのケースでは、Castor における制限により、ユーザーがタイプを指定する必要があります。Castor は、これらのタイプがスタンドアロン・オブジェクトにマップされるとき (他のオブジェクトのメンバーにマップされるのではないとき)、それらのタイプを処理できません。そのため、この関数コンポーネントは、タイプを認識して、Castor が正しいオブジェクトを生成するように特別な処置をとる必要があります。

XML の入力タイプ
このドロップダウン・リストでは、関数コンポーネントが入力 XML データを **DOMElement** オブジェクトの形式で受け入れるか、**String** として受け入れるかを指定します。

詳細ログ
追加のログ・メッセージを生成する場合は、チェックします。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの使用

ここに記載されている情報を参照して、このコンポーネントと対応するモードの使用方法を理解することができます。

CastorXMLToJava 関数コンポーネントは、XML 文書から*項目* オブジェクトまたは一般的な Java オブジェクトを作成します。項目オブジェクトとカスタム Java オブジェクトの両方について動作可能です。

関数コンポーネントに入力として項目オブジェクトを渡すと、項目オブジェクトが戻されます。この動作モードのことを、**項目**モードといいます。

関数コンポーネントに入力して項目オブジェクト以外のオブジェクト (**String** または **DOMelement**) を渡すと、Castor によってアンマーシャルされた Java ロー・オブジェクトが戻されます。この動作モードのことを、非項目モードといいます。

項目モード

- 「**XML の入力タイプ**」パラメーターに **DOMelement** を指定した場合、この関数コンポーネントは、*xmlDOMElement* という名前の属性にタイプ `org.w3c.dom.Element` の値が設定された項目オブジェクトを入力として想定します。
- 「**XML の入力タイプ**」パラメーターに **String** を指定した場合は、*xmlString* という名前の属性にタイプ `java.lang.String` の値が設定された項目オブジェクトが入力として想定されます。
- 生成される出力は*項目* オブジェクトで、その属性は、「**属性の指定**」パラメーターとマッピング・ファイルにより指定されている、アンマーシャルされた XML エレメントです。

非項目モード

- 「**XML の入力タイプ**」パラメーターに **DOMelement** を指定した場合、この関数コンポーネントは `org.w3c.dom.Element` オブジェクトを入力として想定します。
- 「**XML の入力タイプ**」パラメーターに **String** を指定した場合は、`java.lang.String` オブジェクトが入力として想定されます。

XMLToSDO 関数コンポーネント

EMF XMLToSDO 関数コンポーネントは、XML 文書を SDO オブジェクトに変換します。これらの SDO オブジェクトは、XML 構造に類似したツリー形式の構造で接続されています。ここに記載されている情報を参照して、XMLToSDO 関数コンポーネントについて詳しく理解することができます。

注: XMLToSDO 関数コンポーネントは、IBM Security Directory Integrator バージョン 7.2 以降は非推奨になりました。将来のバージョンで削除される予定です。

XML エレメントごとに、XML 属性データ・オブジェクトが作成されます。次に、一部のデータ・オブジェクトに対して IBM Security Directory Integrator 項目属性が作成されます。項目属性の名前は、IBM Security Directory Integrator 属性が表すエレメントの祖先エレメントの名前で構成されています。後続の XML エレメント名は、「@」文字で区切られます。XML 属性を表す場合は、「#」記号を分離文字として使用して、XML エレメントの名前にこの XML 属性の名前が付加されます。

すべての属性名は、XML ルートを表す「DocRoot」テキストで始まります。項目属性値には次の 2 つのタイプがあります。

- XML エレメント/属性値がプリミティブ・タイプ (java.lang.String、java.lang.Integer、java.lang.Boolean など) の場合は、標準 Java ラッパー。
- XML エレメントが複合 XML 構造の場合は、サービス・データ・オブジェクト。これは、タイプ org.eclipse.emf.ecore.sdo.EDataObject のオブジェクトです。

共通する親エレメントを持つ XML エレメントは、兄弟と呼ばれます。同じ名前を持つ兄弟エレメントは、多値の IBM Security Directory Integrator 属性項目にグループ化されます。

注: XML エレメントの祖先エレメントに同名の兄弟がある場合、この XML エレメントの属性は作成されません。このようなエレメントへは、兄弟エレメントを表現する多値属性を介してのみアクセスできます。

例

ここに記載されている例を使用して、EMF XMLToSDO 関数コンポーネントで以下の XML ファイルを処理する方法について説明します。

```
<?xml version="1.0">
<database name="Persons">
 <description>This is a sample database</description>
 <person>
  <name>Ivan</name>
  <age>21</age>
 </person>
 <person>
  <name>George</name>
  <age>32</age>
 </person>
</database>
```

EMF XMLToSDO 関数コンポーネントが例に示されている XML ファイルを処理すると、次の属性を持つ項目が作成されます。

- DocRoot – XML ルートを表すサービス・データ・オブジェクト。

- DocRoot@database -「database」XML エレメントを表すサービス・データ・オブジェクト。
- DocRoot@database#name –「database」XML エレメントの「name」XML 属性を表す java.lang.String オブジェクト。
- DocRoot@database@description -「description」XML エレメント (「database」XML エレメントの子) を表す java.lang.String
- DocRoot@database@person – 個々の「person」XML エレメントを表すサービス・データ・オブジェクトを値として持つ多値属性。

この場合、XML 文書には複数の XML エレメント person が同一レベルで存在するため、「DocRoot@database@person@name」は有効な IBM Security Directory Integrator 属性ではありません。

EMF XMLToSDO 関数コンポーネントでは、ネーム・スペースを接頭部として使用できるオプションがあります。ネーム・スペース接頭部オプションは、項目属性名の XML エレメント名部分に、対応するネーム・スペースを接頭部として追加することを指定します (例えば、「DocRoot@namespace1:database@namespace2:person」)。

構成

ここに記載されているパラメーターを使用することで、XMLToSDO 関数コンポーネントを構成できるようになります。

XSD ファイル

XML スキーマ (XSD) ファイルの位置を指定します。XML スキーマ・ファイルは、XML 文書の読み取り時、EMF Ecore モデルの生成時、およびスキーマ・ディスカバー機能で使用されます。このパラメーターは必須です。

指定する XML スキーマ・ファイルの拡張子は「.xsd」でなければなりません。

ネーム・スペースを使用する

生成される項目属性名で XML エレメントと属性のネーム・スペースを設定するかどうかを指定します。このパラメーターにチェック・マークを付けると、「namespaceMap」パラメーターで定義されている接頭部、または「namespaceMap」パラメーターで接頭部が定義されていない場合はネーム・スペース URI が、XML エレメントおよび属性の接頭部として使用されます。

このパラメーターでは、スキーマ・ディスカバー機能で項目属性名の接頭部として XML ネーム・スペースを使用するかどうかも指定されます。

ネーム・スペース・マップ

ネーム・スペース接頭部とネーム・スペース URI のマッピングを定義します。それぞれのペアを 1 つずつ新規行に指定します。接頭部と URI は等号で区切られます (例:「ibm=http://www.ibm.com」)。接頭部と URI のどちらでも、先頭の空白と末尾の空白は無視されます。

このパラメーターは、「useNamespaces」パラメーターが true に設定されている場合にのみ有効です。

入力 XML のタイプ
入力 XML 文書のタイプを指定します。java.lang.String オブジェクトまたは org.w3c.dom.Element オブジェクトを指定できます。

エンコード
XML との間でのエンコード変換に使用する文字セットを指定します。ブランクのままにすると、デフォルトのシステム文字セットが使用されます。

デバッグ
デバッグ・メッセージを有効にします。

マイグレーション

ここに記載されている情報と手順を参照して、XMLToSDO 関数コンポーネントの場合のマイグレーションを実行することができます。

EMF XMLToSDO 関数コンポーネントと SDOToXML 関数コンポーネントには、IBM Security Directory Integrator 6.0 Castor 関数コンポーネントとの互換性はありません。Castor 関数コンポーネントを使用するソリューションで EMF XMLToSDO 関数コンポーネントおよび EMF SDOToXML 関数コンポーネントを使用できるようにするには、Castor 関数コンポーネントを再インプリメントする必要があります。Castor XML To Java 関数コンポーネントでは、マッピング・ファイルがサポートされています。このマッピング・ファイルを使用して、複合カスタム XML を解析し、複合カスタム Java オブジェクトに変換する方法を指定できます。この機能は、EMF XMLToSDO 関数コンポーネントではサポートされていません。以下の概略的なガイドラインに従うと、IBM Security Directory Integrator 6.0 構成を再インプリメントして EMF XMLToSDO 関数コンポーネントを使用できるようになります。

1. EMF XMLToSDO 関数コンポーネントを AssemblyLine に挿入します。
2. パラメーターを適切に設定します。
3. AssemblyLine の EMF XMLToSDO 関数コンポーネントの直後にスクリプト・コンポーネントを挿入します。
4. このスクリプト・コンポーネント内で、EMF XMLToSDO 関数コンポーネントから戻される SDO DataObject から必要なデータを抽出し、必要なカスタム Java オブジェクトにデータを取り込む Javascript コードを記述します。

Castor XML To Java 関数コンポーネントでは XML の特定部分を項目属性にマップできるようにするメカニズムがサポートされていました。EMF XMLToSDO 関数コンポーネントでは、この機能はサポートされていません。EMF XMLToSDO 関数コンポーネントは常に XML 全体を解析して項目属性にマップします。ただし EMF XMLToSDO 関数コンポーネントの入力属性マップを使用すると、必要な属性のみをマップできるため、Castor XML To Java 関数コンポーネントの動作をエミュレートできます。

Castor Java To XML 関数コンポーネントでは、複合 Java オブジェクトを XML (エレメント/属性名など) にシリアライズする方法を指定するマッピング・ファイルがサポートされていました。. EMF SDOToXML 関数コンポーネントは XML スキーマ・ファイルに基づいてシリアライズされて XML になります。つまり、エレメントおよび属性の名前などは関数コンポーネント・パラメーターとして指定された XML スキーマ・ファイルで指定されます。

SDOToXML 関数コンポーネント

EMF SDOToXML 関数コンポーネントにより、サービス・データ・オブジェクトを XML に変換することができます。このコンポーネントは XML スキーマ定義を使用して Ecore モデルを作成します。

注: SDOToXML 関数コンポーネントは、IBM Security Directory Integrator バージョン 7.2 以降は非推奨になりました。将来のバージョンで削除される予定です。

この関数コンポーネントが受信する項目では、属性が XML 文書を表します。項目属性値のタイプは、プリミティブ・タイプを表す Java クラス、または複合 XML エレメントを表すサービス・データ・オブジェクト (org.eclipse.emf.ecore.sdo.EDataObject) です。

項目属性名は、EMF XMLToSDO 関数コンポーネントでの属性名の構成と完全に同じ方法で XML 階層を記述します。すべての属性名は、XML ルートを表す「DocRoot」で始まります。XML 階層でこれよりも下のエレメントは、「@」文字により分離されます。IBM Security Directory Integrator 項目属性が XML 属性を表す場合は、「#」文字を使用して XML 属性名とこの属性を含む XML エレメントの名前が分離されます。

渡される IBM Security Directory Integrator 項目に、実データに対応する項目属性のみが含まれていることがあります。例えば、項目に「DocRoot@database@person」属性は含まれているが「DocRoot@database」属性は含まれていない場合、EMF SDOToXML 関数コンポーネントは、作成する XML 文書に XML エレメント「database」を自動的に作成します。EMF SDOToXML 関数コンポーネントは XML スキーマを使用して、指定された XML エレメントまたは属性の祖先にあたる XML エレメントをすべて追跡および作成します。

項目に含まれている属性で指定される XML エレメントが、項目属性により指定される他の XML エレメントに含まれていることがあります。例えば、項目に「DocRoot@database@person」属性と「DocRoot@database」属性の両方が含まれている場合です。この場合、ルートに最も近い属性から処理が開始され、順次処理されます。最後に処理されるエレメントは、他のすべての属性に含まれている最も固有な XML エレメントとなります。この処理順序により、大きな XML コンテキストにおいて特定の詳細を変更することができます。

例えば、「DocRoot@database@person」エレメントのみを変更し、「DocRoot@database」エレメントの他の部分は変更しないでおく場合は、EMF XMLToSDO 関数コンポーネントを使用して XML 文書を読み取り、「DocRoot@database」属性をマップし、EMF SDOToXML 関数コンポーネントにその属性をそのまま提供することができます。次に、XML エレメント「person」に対して実行する特定の更新が含まれている「DocRoot@database@person」属性も提供します。EMF SDOToXML 関数コンポーネントは、最初に「DocRoot@database」を処理し、読み取った XML にその内容全体を適用します。次に「database」エレメントの子「person」を、「DocRoot@database@person」項目属性の内容で指定変更します。

多値属性と、そのエレメントの子またはその他の後続エレメントを指定する属性が組み合わせて指定されている場合は、関数コンポーネントは兄弟 XML エレメント

のうちこの後続エレメントの適用対象を判別できないため、エラーが出されます (例外がスローされます)。例えば「DocRoot@database@person」が指定されており、これに 2 つの値が含まれており (したがって、2 つの XML エレメント「person」が同一レベルで指定されており)、かつ「DocRoot@database@person@name」も指定されている場合、この関数コンポーネントは既存の 2 つの「person」エレメントのうちのいずれに「name」エレメントが適用されるのか分かりません。項目属性のエレメント名には、XML ネーム・スペースが接頭部として使用されています。

エレメント名には、接頭部としてネーム・スペース URI または「namespaceMap」パラメーターで定義されている接頭部が使用されます。

例えば、以下の XML 文書を作成するとします。

```
<?xml version="1.0"?>
<database xmlns="www.ibm.com" xmlns:tmp="www.tmp.com" name="employees">
 <person>
  <name>Ivan</name>
  <tmp:age>21</tmp:age>
 </person>
</database>
```

この場合、以下の IBM Security Directory Integrator 項目を EMF SDOToXML 関数コンポーネントに渡すことができます。

- DocRoot@ibm:database#ibm:name
- DocRoot@ibm:database@ibm:person@ibm:name
- DocRoot@ibm:database@ibm:person@www.tmp.com:age

使用されているネーム・スペース接頭部では、「namespaceMap」パラメーターで「ibm」接頭部が「www.ibm.com」に設定されており、「www.tmp.com」には接頭部が定義されていないことが想定されています (このため、「name」属性では直接使用されていす)。

構成

ここに記載されているパラメーターを使用することで、SDOToXML 関数コンポーネントを構成できるようになります。

XSD ファイル
このパラメーターは XML スキーマ・ファイルの場所を指定します。XML スキーマ・ファイルは、XML 文書の生成処理とスキーマ・ディスカバー機能で使用されます。このパラメーターは必須です。指定する XML スキーマ・ファイルの拡張子は「.xsd」でなければなりません。

ネーム・スペースを使用する
スキーマ・ディスカバー機能で項目属性名の接頭部として XML ネーム・スペースを使用するかどうかが指定されます。このパラメーターにチェック・マークを付けると、「namespaceMap」パラメーターで定義されている接頭部、または「namespaceMap」パラメーターで接頭部が定義されていない場合はネーム・スペース URI が、XML エレメントおよび属性の接頭部として使用されます。

ネーム・スペース・マップ
ネーム・スペース接頭部とネーム・スペース URI のマッピングを定義します。それぞれのペアを 1 つずつ新規行に指定します。接頭部と URI は等

号で区切られます (例:「ibm=http://www.ibm.com」)。接頭部と URI のどちらでも、先頭の空白と末尾の空白は無視されます。

XML の戻りタイプ
関数コンポーネントにより戻される XML 文書のタイプを指定します。java.lang.String オブジェクトまたは org.w3c.dom.Element オブジェクトを指定できます。

エンコード
XML との間でのエンコード変換に使用する文字セットを指定します。ブランクのままにすると、デフォルトのシステム文字セットが使用されます。

デバッグ
デバッグ・メッセージを有効にします。

関数コンポーネントの使用

ここに記載されている情報を通じて、SDOToXML 関数コンポーネントの使用方法を理解することができます。

マイグレーション

以下の手順により、SDOToXML 関数コンポーネント上でマイグレーションを実行することができます。

EMF XMLToSDO 関数コンポーネントと SDOToXML 関数コンポーネントには、IBM Security Directory Integrator 6.0 Castor 関数コンポーネントとの互換性はありません。このため、Castor 関数コンポーネントを使用するソリューションで EMF XMLToSDO 関数コンポーネントおよび EMF SDOToXML 関数コンポーネントを使用できるようにするには、Castor 関数コンポーネントを再インプリメントする必要があります。Castor XML To Java 関数コンポーネントでは、マッピング・ファイルがサポートされていました。このマッピング・ファイルを使用して、複合カスタム XML を解析し、複合カスタム Java オブジェクトに変換する方法を指定できます。この機能は、EMF XMLToSDO 関数コンポーネントではサポートされていません。ただし、以下の概略的なガイドラインに従うと、このような IBM Security Directory Integrator 6.0 構成を再インプリメントして EMF XMLToSDO 関数コンポーネントを使用できるようになります。

1. EMF XMLToSDO 関数コンポーネントを AssemblyLine に挿入します。
2. パラメーターを適切に設定します。
3. AssemblyLine の EMF XMLToSDO 関数コンポーネントの直後にスクリプト・コンポーネントを挿入します。
4. このスクリプト・コンポーネント内で、EMF XMLToSDO 関数コンポーネントから戻される SDO DataObject から必要なデータを抽出し、必要なカスタム Java オブジェクトにデータを取り込む Javascript コードを記述します。

Castor XML To Java 関数コンポーネントでは XML の特定部分を項目属性にマップできるようにするメカニズムがサポートされていました。EMF XMLToSDO 関数コンポーネントでは、この機能はサポートされていません。EMF XMLToSDO 関数コンポーネントは常に XML 全体を解析して項目属性にマップします。ただし EMF

XMLToSDO 関数コンポーネントの入力属性マップを使用すると、必要な属性のみをマップできるため、Castor XML To Java 関数コンポーネントの動作をエミュレートできます。

Castor Java To XML 関数コンポーネントでは、複合 Java オブジェクトを XML (エレメント/属性名など) にシリアライズする方法を指定するマッピング・ファイルがサポートされていました。. EMF SDOToXML 関数コンポーネントは XML スキーマ・ファイルに基づいてシリアライズされて XML になります。つまり、エレメントおよび属性の名前などは関数コンポーネント・パラメーターとして指定された XML スキーマ・ファイルで指定されます。

AssemblyLine 関数コンポーネント

AssemblyLine 関数コンポーネント (AL FC) は、他の AssemblyLine の呼び出しをコンポーネントにラップします。これにより、その AssemblyLine の実行方法、および発生する可能性のある結果への対処方法などをある程度制御できます。

AL FC は、サーバー API を使用して AL を呼び出し、管理します。このコンポーネントは、RMI を介してサーバー API とのサーバー接続を確立し、サーバーとのセッションを作成します。

構成

ここに記載されているパラメーターを使用することで、AssemblyLine 関数コンポーネントを構成できるようになります。

AssemblyLine
この関数コンポーネントのターゲットにできる定義済み AssemblyLine が列挙されたドロップダウン・リスト。

サーバー
AssemblyLine を実行する IBM Security Directory Integrator サーバー。内部サーバーには「*Local*」またはブランク、リモート・サーバーには「*hostname[:port]*」を使用します。

構成インスタンス
リモート・サーバーを使用するときの構成インスタンスを指定します。

実行モード
次の 3 つの利用可能なモードを列挙したドロップダウン・リスト。

実行後結果を待機する
この関数コンポーネントの Javadoc の説明に従って実行結果をピックアップできます。一般に、空の項目オブジェクトを使用して関数コンポーネントを呼び出します。戻された項目オブジェクトの value 属性に、ターゲット AL への参照が含まれています。

バックグラウンドで実行する
このモードでは、AssemblyLine を非同期に開始し、結果を待機しません。

手動 (サイクル・モード)
AssemblyLine を 1 サイクルの間のみ実行します。

カスタム鍵ストア
「api.remote.server」を使用する場合にチェック・マークを付けます。鍵ストア構成のための標準の「javax.net.ssl.」プロパティーの代わりとなる java プロパティー。

TCB 属性を使用する
これにチェック・マークを付けると、関数コンポーネントは「$tcb.」接頭部が付いている属性を TCB へのパラメーターとして解釈し、項目から除去します。

シミュレート
呼び出された AssemblyLine をシミュレート・モードで実行する場合は、これにチェック・マークを付けます。これは、呼び出された AssemblyLine は、外部システムと対話するときに、そのシミュレーション構成を使用することを意味しています。

ロギングの共用
これにチェック・マークを付けた場合、呼び出された AssemblyLine はこのコンポーネントと同じロギングを使用します。

操作　ターゲットの AssemblyLine で定義されている、使用可能な公開操作から選択します。「**照会**」ボタンを使用すると、ターゲットの AL からのスキーマ (入力属性または出力属性) の取り出しが試みられます。詳しくは、17 ページの『AssemblyLine コネクター』および 775 ページの『付録 F. アダプターを使用した新規コンポーネントの作成』を参照してください。

AssemblyLine パラメーター
適切な AL が選択された場合、このフィールドはその AL の初期化パラメーターへのアクセスを提供します。

注: リモート AL がその構成、つまり *$initialization* スキーマで初期化パラメーターを定義していなかった場合、このフィールドは空のままです。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの使用

この関数コンポーネントは、ローカル・サーバー上またはリモート・サーバー上の AssemblyLine の呼び出しと管理を行うためのハンドラー・オブジェクトを提供します。

この関数コンポーネントを構成するには、呼び出す AL を選択し、その AL を実行するように定義されているサーバーを指定し (ブランクまたは「local」は、FC を実行しているこのサーバー上で AL を実行することを示す)、AL が所属する構成インスタンスを指定します。また、空白のパラメーター値は、この AL が関数コンポーネント自体を含む同じ構成インスタンスにあることを意味します。

また、実行モードも選択します (詳しくは、「*Directory Integrator の構成*」の『AL のサイクル・モード』を参照)。実行モードは 3 つありますが (「実行して結果を待機」、「バックグラウンドで実行」、「手動 (サイクル・モード)」)、スクリプトか

ら AL を開始する標準的な方法は最初の 2 つのオプションです (AL の join() メソッドを呼び出すスクリプトと、呼び出さないスクリプトとのいずれの場合も)。

最初の 2 つのモードを選択すると、ターゲットで AL が自身のスレッドで独自 (スタンドアロン) に実行されます。3 番目のモード (サイクル・モード) では、ターゲット AL が関数コンポーネントによって制御されます。関数コンポーネントが呼び出されるたびに、関数コンポーネントによって AL が 1 サイクル実行されます。関数コンポーネントは、AssemblyLine をスタンドアロン・モードで実行するとき、ターゲット AL への参照を保持します。これは、main.startAL() を呼び出す場合と同じです。また、関数コンポーネントは、実行中または停止した AL の状況を戻すこともできます。この状況は、NULL または空の項目パラメーターを使用して関数コンポーネントの perform() メソッドを呼び出すことにより取得できます。戻された項目オブジェクトでは、「value」という属性にターゲット AL への参照が入っています。関数コンポーネントに NULL 値を渡すと、戻り値はターゲット AL への実際の参照になります (これも、main.startAL() を呼び出す場合と同じ)。

また、下記の特定のストリング・コマンド値を指定して関数コンポーネントを呼び出すと、ターゲット AL に関する情報を取得できます。

perform("target")	ターゲット AL のオブジェクト参照子を戻します。
perform("active")	ターゲットの AL の状況に応じて、「active」、「aborted」、または「terminated」のいずれかを戻します。
perform("error")	状況が "aborted" のとき、java.lang.Exception オブジェクトを戻します。
perform("result")	現在の結果項目オブジェクトを戻します。
perform("stop")	アクティブなターゲット AL の終了を試行し、呼び出しが成功しなかった場合にエラーをスローします。

実行モードとして「実行して結果を待機」を指定した場合は、perform() を呼び出すたびにターゲット AL が開始され、実行の完全な状況 (例えば、ターゲット AL への参照、状況およびエラー・オブジェクト) が戻されることに注意してください。これは、他のすべての場合とは異なり、initialize() メソッドを呼び出してもターゲット AL が開始されません。このモードの関数コンポーネントを、項目オブジェクトを指定して呼び出す場合は、上記のキーワードのうち 1 つ以上を「command」という属性に (上の表で説明したキーワードをコンマで区切ったリストに連結して) 指定して項目オブジェクトに組み込むことができます。戻される項目オブジェクトは、前述と同じ値で取り込まれます。そのため、必要なコマンドを使用して perform() を複数回呼び出すのではなく、項目オブジェクトの属性としてすべてのキーワードを持つ項目を作成し、perform() への 1 回の呼び出しで解放することができます。

```
var e = system.newEntry();
e.setAttribute("command", "target, status");
// In this example, fc references a Function Interface.
// If this was an AL Function instead, then fc.callreply(e)
// would be done.
var res = fc.perform(e);
task.logmsg("The status is: " + res.getString("status"));
```

「手動」モードを指定して関数コンポーネントで AL を実行すると、項目オブジェクトを指定して関数コンポーネントを呼び出すたびに、ターゲット AL で 1 サイクルが実行されます。戻される項目オブジェクトは、サイクルの終了時点での結果作業項目です。ターゲットの AL が完了すると、NULL 項目が戻されます。サイク

ルの実行でエラーが起きた場合は、そのエラーが関数コンポーネントによって再度スローされます (したがって、スクリプトの中で try-catch ブロックを使用する必要があります)。

ターゲットの AL には、異なる 2 つの方法でパラメーターを提供できます。

タスク呼び出しブロック (TCB) を使用して

fc.getTCB() メソッドを使用し、戻された TCB オブジェクトでパラメーターを設定することができます。このオブジェクトは、次回 AssemblyLine がこの FC によって開始されたときに使用されます。戻された TCB では、コネクター・パラメーターのみを設定しなければなりません。その理由は、この関数オブジェクトは実行モードと初期作業項目を上書きする可能性があるからです。

特殊な属性を使用する

TCB パラメーターを設定するためのもう 1 つの方法では、特定の接頭部「$tcb.」で変数を定義する必要がある出力属性マップを使用します。これらの属性が項目で見つかると、それらの属性は TCB に移動させられ、その項目から除去されます。これが動作するのは、関数コンポーネントが呼び出されるたびに (つまり、実行と待機の完了のたびに) 関数コンポーネントが AssemblyLine を実行する場合です。属性名「$tcb.accumulator」は、アキュムレーターの設定に使用されます。

AL FC の出力マップでは、以下の $tcb.* 属性を使用することができます。

- コンポーネント・パラメーター `$tcb.`*`connectorName`*`.`*`paramName`*。`connectorName` は AssemblyLine コンポーネントの名前、`paramName` は構成パラメーターの内部名です。例えば、FileConnector という名前のファイル・コネクターがあり、ファイル・パスを設定したい場合は、`$tcb.FileConnector.filePath` 属性をマップします。
- アキュムレーター `$tcb.accumulator` は、AssemblyLine によって生成された作業項目を収集するためのアキュムレーターを指定します。

関数コンポーネントの「入力マップ」および「出力マップ」タブの「照会」(「クイック・ディスカバリー」) ボタンを使用すると、次のような方法で、呼び出される AL のスキーマの判別を試行できます。

1. 「**操作**」パラメーターが設定されている場合、関数コンポーネントは、その操作の「入力」および「出力」マップで定義されているすべての属性を取得します。
2. AL でスキーマが定義されている場合は (AL の「呼び出し/戻り」タブ)、その情報が使用されます。
3. スキーマが定義されていない場合、関数コンポーネントは、AL 内のすべてのコネクターについて入力マップと出力マップを調べて、AL のスキーマを「推測」します。

ターゲットの AL では、「querySchema」という名前の操作の入力属性および出力属性を使用して AL FC (または AssemblyLine コネクター) にスキーマが提供されている場合には、その操作を定義することができます。

AssemblyLine へのパラメーターの引き渡し

以下の例に、AssemblyLine 関数コンポーネントを使用して、さまざまな AssemblyLine にパラメーターを渡す方法を示します。

1. AssemblyLine 関数コンポーネントの呼び出し前フックで、以下に示すように TCB を宣言します。

```
tcb = thisComponent.getFunction().getTCB();
// gets the current TCB of the running AssemblyLine.
tcb.setALSetting("dn", work.getString("$dn"));
// dn is the attribute name that TCB holds. The second parameter in
// tcb.setALSetting(<>,<>) is the actual value that needs to be passed to the target AssemblyLine.
```

2. 構成エディターで、「**TCB 属性を使用する**」チェック・ボックスを選択します。
3. ターゲット AssemblyLine で、マップされる変数 (入力マップ) について、以下のコードを使用して、渡されたパラメーターを取得します。

```
var mydn = task.getConfigStr("dn");
//task.getConfigStr(<transfer_name>), the transfer_name should match the name used in Step 1.
```

関連情報

17 ページの『AssemblyLine コネクター』,
775 ページの『付録 F. アダプターを使用した新規コンポーネントの作成』.

Java クラス関数コンポーネント

IBM Security Directory Integrator では、スクリプト・コード内で Java オブジェクトを使用して、IBM Security Directory Integrator では直接提供されない特定の操作を実行することができます。

Java オブジェクトを構成し、パラメーターを適切なクラスにマップする必要がある場合に、Java オブジェクトのメソッドを呼び出すことが困難なことがあるため、Java クラス関数コンポーネントを使用することで、スクリプトでの Java オブジェクトの使用が容易になります。Java クラス関数コンポーネントにより、構成エディターで Java クラスとメソッドを選択できます。また、この関数コンポーネントはパラメーターからメソッドへの変換とマッピングを実行します。

スキーマ

ここに記載されている情報を参照して、AssemblyLine 関数コンポーネントのスキーマを理解することができます。

Java クラス関数コンポーネントのスキーマは動的であり、選択された Java クラスとメソッドを反映します。この関数コンポーネントはまた、ターゲット Java クラス/メソッドのシグニチャーに一致するようパラメーターを動的に変換します。

パラメーター変換

ほとんどの一般的なタイプではパラメーター変換が実行されます。ただしこの関数コンポーネントでは、すべての Java クラス・オブジェクトを対象とした変換は提供されません。サポートされないオブジェクトについては、Java クラス関数コンポー

ネントを呼び出す前に、これらを明示的に作成しておく必要があります。Java クラス関数コンポーネントがパラメーター変換で認識するオブジェクトの表を以下に示します。

パラメーター・タイプ	注記
整数	オブジェクトおよびプリミティブ・タイプの両方
長	オブジェクトおよびプリミティブ・タイプの両方
倍精度 (Double)	オブジェクトおよびプリミティブ・タイプの両方
浮動 (Float)	オブジェクトおよびプリミティブ・タイプの両方
短精度 (Short)	オブジェクトおよびプリミティブ・タイプの両方
バイト (Byte)	オブジェクトおよびプリミティブ・タイプの両方
文字	オブジェクトおよびプリミティブ・タイプの両方
ブール	オブジェクトおよびプリミティブ・タイプの両方
日付	DateFormat での定義に基づくデフォルトの日付形式からの変換のみ
ストリング	

上記のタイプ以外に、Java クラス関数コンポーネントではプリミティブ配列および java.util.Collection オブジェクトへの変換も試行されます。

構成

Java クラス関数コンポーネントでは、ここに記載されているパラメーターが使用されます。

Jar/クラス・ファイル
このパラメーターでは、Java クラスが記述されているファイルを指定します。

Java クラス
Java クラスの完全修飾名を指定します。このパラメーターは必須です。

メソッド
Java クラスで呼び出すメソッドを指定します。

パーサー関数コンポーネント

パーサー関数コンポーネントにより、パーサーを AssemblyLine コンポーネントにラップすることができます。これにより、AssemblyLine データ・フローの任意の場所にパーサーを挿入できるようになります。

パーサー関数コンポーネントの複数インスタンスを利用すると、2 つ以上のレイヤーから成るプロトコルをデコードするのに役立ちます。

構成

ここに記載されているパラメーターを使用することで、パーサー関数コンポーネントを構成できるようになります。

動作モード
パーサーの動作モード: パーサーからの項目の読み取り、パーサーへの項目書き込み (結果を戻す)。

結果を String で戻す
これをチェックすると、関数は *Bytearray* オブジェクトではなく String オブジェクトを戻します。

文字セット
値をストリング・オブジェクトとして戻す場合に使用する文字セット。デフォルトは UTF-8 です。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

パーサー関数コンポーネントには「**パーサー**」タブもあります。「パーサー」タブを使用すると、データ・ストリーム・レコードを解釈または生成するために使用するパーサーを選択して構成することができます。

関数コンポーネントの使用

この関数コンポーネントを使用してパーサーを選択し、パーサーのモードを設定することができます。

有効なモードは、入力 (**読み取り**) または出力 (**書き込み**) です。

読み取りモードでは、「*value*」という名前の属性を (出力マップで) 提供する必要があります。この属性は、ストリング・オブジェクト、ファイル・オブジェクト、リーダー・オブジェクト、または java.io.InputStream オブジェクトのいずれかで、パーサーの入力として使用されます。関数コンポーネントは、解析済みの属性を持つ *Entry* オブジェクト (conn) を戻します。これらの属性は、入力マップに使用できます。

書き込みモードでは、関数コンポーネントは出力マップによって渡された属性を持つ項目を受け取り、パーサーをその項目に適用して、「*value*」という名前の属性で戻りバイト・ストリームを提供します。この属性は、「構成」タブで「**結果を String で戻す**」チェック・ボックスが選択されている場合は java.lang.String です。それ以外の場合は、*bytearray* です。

スクリプト記述関数コンポーネント

コネクターやパーサーと同様、IBM Security Directory Integrator を使用することにより、スクリプト記述を使用して関数コンポーネントを完全にプログラミングできます。

そのためには、スクリプト記述された関数コンポーネントが提供しているテンプレートを利用します。

注: スクリプト記述関数コンポーネントのスクリプトは、個別の JavaScript エンジンで実行されます。これは、スクリプトが使用可能または設定済みのいずれの変数にも通常の AssemblyLine フックでアクセスできないことを意味します。

構成

スクリプト関数コンポーネントは、すべてのロジックが「スクリプト」ペインにあるため、比較的簡単に構成することができます。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの使用

関数コンポーネントの大半は「スクリプト」ペインにあります。そのペインで、関数コンポーネントを構成するロジックを設定する必要があります。

プログラミング支援機能として、有効な関数コンポーネントを作成するのに必要な関数の覚え書となるスタブ関数が用意されています。これには、以下のものがあります。

initialize (fc,obj)
この関数は、この関数コンポーネントが組み込まれている AssemblyLine の初期化フェーズ中に呼び出されます。このメソッドが AssemblyLine 関数コンポーネントから呼び出された場合、「*obj*」パラメーターは null です。

terminate (fc)
この関数は、この関数コンポーネントが組み込まれている AssemblyLine の終了フェーズ中に呼び出されます。ここでは、リソースの解放などを行います。

perform (fc,obj)
これは、実際の処理を実行する関数で、AssemblyLine 内のこの関数コンポーネントを配置した位置から AssemblyLine によって呼び出されます。「*obj*」パラメーターは、このメソッドが AssemblyLine 関数コンポーネントから呼び出された場合は、マップされた属性を含む項目です。

これらは、3 つの主要な関数インターフェース・メソッドに相当します。各メソッドには、この ScriptedFC である関数パラメーターが渡されます。

オブジェクト

共通オブジェクトのリスト (AssemblyLine の場合と同じリストです) を以下に示します。

main 実行中の構成インスタンス (RS オブジェクト)。

task このパーサーが含まれる AssemblyLine。

system `UserFunctions` オブジェクト。

config このエレメント、つまりこの関数コンポーネントの構成。`config.getParent()` は、AssemblyLine 関数コンポーネントの FunctionConfig になり、属性マッピングなどを含みます。

スクリプト・パーサーからアクセスできるオブジェクトは、以下のみです。

関連情報

310 ページの『スクリプト・コネクター』,
461 ページの『スクリプト・パーサー』,
「リファレンス」の『JavaScript コネクター』

CBE 関数コンポーネント

CBE 関数コンポーネントを使用して、Common Base Event (CBE) イベント・オブジェクトを生成することができます。

これらのイベント・オブジェクトは、CBE ログに書き込んだり (CBE ログは、Autonomic Computing Toolkit の Log and Trace Analyzer を使用して表示と管理を行うことができます)、IBM Common Event Infrastructure (CEI) サーバーに出力したりすることができます。あるいは、CBE イベントを listen する外部アプリケーションにこれらのイベント・オブジェクトを送信することもできます。

IBM Security Directory Integrator で関数コンポーネントを使用する場合は、作業項目属性を CBE 関数コンポーネントの出力マップで公開されている標準の CBE 属性にマップする必要があります。これにより、AssemblyLine が実行されると、CBE 関数コンポーネントは CBE イベント・オブジェクト (および CBE イベント XML) を作成して、それを AssemblyLine の*作業* 項目に戻します。

また、CBE オブジェクトを「event」属性にマップする、または CBE オブジェクトを表す XML を OutputMap の「eventXml」属性にマップすることもできるため、CBE 関数コンポーネントは AL の実行時にすべての標準 CBE 属性を取り出して、それらを*作業* 項目に戻します。

イベントは、状態の発生結果として送信されたメッセージ・データをカプセル化します。複雑な IT システムにおいてアプリケーション間でイベントが交換されることで、システムのさまざまな面での相互運用、通信、および各自アクティビティーの調整が可能になります。エンタープライズ管理および e-ビジネス・コミュニケーションの基本的な側面 (パフォーマンス・モニター、セキュリティー、および信頼性など)と、e-ビジネス・コミュニケーションの基本部分 (注文追跡など) は、これらのイベントの実行可能性と精度に基づいています。

Common Base Event は、エンタープライズ管理およびビジネス・アプリケーション・イベントの新たな標準として定義されています。Common Base Event の定義によって、特定の状態発生時に一般情報をパブリッシュするためのプロパティーを提供することで、データの完全性が確保されます。Common Base Event により提供されるこの一般情報は、3 タプルと呼ばれます。

3 タプルを構成するエレメントを以下に示します。

- 特定の状態をレポートするコンポーネントの識別
- 状態に影響を受けたコンポーネントの識別 (状態を報告するコンポーネントと同じ場合がある)
- 状態そのもの

CBE (Common Base Event)

Common Base Event (CBE) は、ロギング機能、管理機能、問題判別機能、オートノミック・コンピューティング機能、e-ビジネス機能をサポートする異種エンタープライズ・コンポーネント間の効果的な相互通信を促進します。

CEI (Common Event Infrastructure)

Common Event Infrastructure (CEI) は、ビジネス、システム、ネットワークのさまざまな CBE 形式イベントについて作成、送信、維持、配布を行うための、整合性のある統一された一連の API とインフラストラクチャーの IBM インプリメンテーションです。

CEI はオートノミック・コンピューティング部門の CBE 仕様に基づいています。この仕様は、デバイスやソフトウェアがトランザクションなどのアクティビティーを追跡するために使用するイベント情報の標準形式を定義しています。

CEI は、基本イベント管理サービスを必要としているアプリケーションにそれらのサービスを提供することを目的とする埋め込み可能なテクノロジーです。このイベント・インフラストラクチャーは、複数の異種のソースからの未加工イベントの統合と永続化、およびイベント・コンシューマーへのそれらのイベントの配布のための統合ポイントとして機能します。イベントは Common Base Event モデルを使用して表されます。このモデルは、エンタープライズ管理およびビジネス・アプリケーションによって使用されることを目的とする、イベントの共通表現を定義する標準です。この標準は IBM Autonomic Computing Architecture Board により開発されたもので、ロギング、トレース、管理、およびビジネス・イベントのエンコードを、XML ベースの共通フォーマットを使用してサポートし、さまざまなアプリケーションから発生するさまざまなタイプのイベントを関連付けることを可能にします。

入力属性および出力属性

ここに記載されているリンクを参照して、入力属性と出力属性について詳しく理解することができます。

この関数コンポーネントの入力属性および出力属性は、 425 ページの『CBE パーサー』の入力属性および出力属性と同じです。

構成

CBE 関数コンポーネントは、ここに記載されているパラメーターを使用します。

ロガーの名前
ロガーの名前。これはオプション属性です。定義しないと、デフォルトとして LocalHostIP が使用されます。

モード これは、この関数コンポーネントが、CBE オブジェクトを項目から戻すか、または項目を CBE オブジェクトから戻すかを指定します。指定できる値は、以下のとおりです。

項目 -> CBE
これは、IBM Security Directory Integrator 7.1.1 よりも前に使用されていたデフォルト・モードです。このモードでは、OutputMap の必

須属性を提供し、CBE オブジェクトを InputMap の「event」属性 (「eventXml」属性の XML 表現でもあります) で受け取る必要があります。

CBE -> 項目
このモードでは、ユーザーは CBE イベント・オブジェクトを属性に変換することができます。プレーンな Java オブジェクトのとしての OutputMap の「*event*」属性、または XML 表現としての「*eventXml*」属性のいずれにおいても、CBE オブジェクトが予期されています。したがって、戻される属性は InputMap で提供されます。タグの値 (<situationType> someValue </situationType>) がそのまま受け取られるというのは、XML に固有のものです。つまり、値に改行またはタブが含まれていると、これらの文字は同じように属性に戻されます。

XML の妥当性検査
XML の妥当性検査を、CBE 仕様の XSD スキーマと照合して行うかどうかを指定します。デフォルトは「true」です。

デバッグ
デバッグ・メッセージを有効にします。このパラメーターは、すべての IBM Security Directory Integrator コンポーネントに対してグローバルに定義されます。

CBE ログ XML の生成

既存のログ・ファイルを解析し、CBE に準拠する新しいログ・ファイルを生成するソリューションを作成することができます。あるいは、自社製品を直接 IBM Security Directory Integrator と通信させて、ログを CBE 準拠フォーマットで生成することができます。

IBM Security Directory Integrator ユーザーの主要なニーズの 1 つは、CBE に準拠したログを生成できる機能を自社製品に持たせることにより、IBM Autonomic Computing Toolkit の Log and Trace Analyzer (LTA) などの他の CBE ログ・アナライザーを使用して、一貫性のある共通した方法でさまざまなシステムのレポートを生成し、ログを分析できるようにすることです。

シナリオがどのようなものであれ、CBE イベントを生成するには CBE 関数コンポーネントを使用する必要があります。これは、以下の手順を使用して達成することができます。

1. ログに記録する必要がある属性を AssemblyLine の*作業* 項目内に挿入します。これは、いくつかの既存の製品ログを解析したり、履歴データベースを読み取ったりなどして行うことができます。
2. これらの属性は、CBE 関数コンポーネントの*出力マップ* 操作を使用して CBE 関数コンポーネントに挿入され、CBE 関数コンポーネントは、さまざまな属性がユーザーごとに渡された値として設定されている CBE イベント・オブジェクトのインスタンスを生成します。
3. (CBE 関数コンポーネントからのイベント生成の後の) フックの 1 つで、**getCBELogXML( )** API (CBE 関数コンポーネントで公開されている API) を呼び出し、新規に作成された *event* オブジェクトを渡すことができます。その結果出力されるストリングは、Hyades CBE ロギング・フォーマットに準拠した

XML フラグメントです。**getCBELogXML( )** API から受け取ったストリングを (例えば) 作業項目に再び設定するには、work.setAttribute( ) API を呼び出します。

```
var cbe = work.getObject("event");
var xmlString = com.ibm.di.fc.cbe.CBEGeneratorFC.getCBELogXml(cbe,false);
work.setAttribute("logXML",xmlString);
```

4. 次に、LineReader パーサーとファイル・コネクターを使用して、この新規属性 (CBE ログ XML が格納されている属性) を任意のログ・ファイルに書き込むことができます。

CEI サーバーへのイベントの発行

CBE 関数コンポーネントを利用して、IBM CEI サーバー・コンポーネントとの間で直接 CBE イベントを発行/受信することができます。

現在のところ、IBM CEI サーバーは IBM WebSphere Process Server バージョン 6.0 に同梱されているコンポーネントです。外部の Java アプリケーション (IBM WebSphere Application Server の内部では稼働しない) の場合、CEI サーバーにイベントを発行する唯一の方法は、https://cs.opensource.ibm.com/projects/mainstream/ で入手可能な TEC Web サービスを使用する方法です。

この Web サービスは、WS Notification を利用して外部アプリケーションから CBE イベントを受信し、次に IBM CEI SDK を利用してこれらの CBE イベントを CEI サーバーに送信します。現在、この Web サービスは、イベントを消費したりサブスクライブしたりするための手段を提供していません。WebSphere によって WS Notification の標準化されたインプリメンテーションがリリースされれば、TEC チームが提供を検討する可能性があります。

以下の手順は、IBM CEI サーバーにイベントを発行するためのソリューションを構成する方法を示しています。

1. 必須の属性を AssemblyLine の *作業* 項目に挿入します。これは、いくつかの既存の製品ログを解析したり、履歴データベースを読み取ったりなどして行うことができます。
2. これらの属性は、CBE 関数コンポーネントの出力マップ操作を使用して CBE 関数コンポーネントに挿入され、CBE 関数コンポーネントは、さまざまな属性がユーザーごとに渡された値として設定されている CBE イベント・オブジェクトのインスタンスを生成します。
3. イベント・オブジェクトが IBM Security Directory Integrator の Axis Web サービス関数コンポーネントに渡されます。この関数コンポーネントは CommonBaseEvent オブジェクトをシリアライズし、それを SOAP を介して (ユーザー定義のポートおよび WSDL アドレス上の) CEI Web サービスに送信します。
4. CEI Web サービスはこのイベントを CEI サーバーに送信します。

関数コンポーネント API

CBE 関数コンポーネントは、ここに記載されているメソッドを公開します (このコンポーネントの Javadoc も参照してください)。

public String convertCBEEventToXML (CommonBaseEvent event) throws Exception

このメソッドは、CommonBaseEvent オブジェクトを XML ストリング・オブジェクトに変換します。この XML は、入力マップの「eventXml」属性でもデフォルトで使用することができます。

public String getCBELogXML (CommonBaseEvent event, boolean isCompleteXML)

このメソッドは、org.eclipse.hyades.logging.java.CommonBaseEventLogRecord クラスの externalizeCanonicalXmlDocString( ) および externalizeCanonicalXmlString( ) API のラッパーです。このメソッドは、CBE ログ XML を取得する場合に使用できます。戻された XML ストリングが完全な XML 文書であるか、それとも単に XML の断片であるかは、isCompleteXML フラグで決定されます。

詳しくは、 http://archive.eclipse.org/tptp/4.2.0/javadoc/Platform/public/org/eclipse/hyades/logging/java/CommonBaseEventLogRecord.htmlを参照してください。

523 ページの『CBE ログ XML の生成』も参照してください。

public static String mapCbeToEntry (CommonBaseEvent cbe, Entry entry)

この静的メソッドは、Common Base Event オブジェクトのフィールドを IBM Security Directory Integrator 項目の属性にマップします。このプロセスは、CBE 関数コンポーネントの「perform」メソッドによるプロセスとは逆のプロセスです。マップ後の項目のすべての属性は、java.lang.String 型になります。

このメソッドには IBM Security Directory Integrator の Javascript を介してアクセスできます。

関連情報

- 425 ページの『CBE パーサー』

- A Practical Guide to the IBM Autonomic Computing Toolkit
- examples/cbe_demo にある IBM Security Directory Integrator を使用した Common Base Event 生成のサンプル

SendEMail 関数コンポーネント

SendEMail 関数コンポーネントは、JavaMail API を使用して E メールを送信します。

SendEMail 関数コンポーネントは Simple Mail Transfer Protocol (SMTP) サーバーに接続し、E メールを複数の受信者に送信したり、オプションで複数のファイルを E メールに添付したりできます。また、Multipurpose Internet Mail Extensions (MIME) タイプが異なる複数のファイルを添付することもできます。

注: 多くの Web ベース E メール・サービスには、HTTP 経由でブラウザーからのみアクセスできます。SendEMail 関数コンポーネントを使用してこのようなサービスにアクセスすることはできません。

スキーマ

SendEMail 関数コンポーネントは、SMTP サーバーを使用して E メールを送信します。この関数コンポーネントを操作するには、構成パラメーターを使用するか、属性をマップします。ここでは、入力スキーマと出力スキーマについて説明します。

出力スキーマ

attachments

多値属性。それぞれの値は、E メールに追加される添付ファイルを指定します。それぞれの値は、絶対ファイル・パスか、作業ディレクトリーを起点とする相対ファイル・パスです。この属性が存在していると、添付ファイル関数コンポーネント・パラメーターの値が指定変更されます。

異なる MIME タイプを持つ各ファイルを添付するには、添付ファイルの名前の後にその添付ファイルの MIME タイプを「>」で区切って追加します。

例を示します。

```
SomeDocument.pdf>application/pdf
```

body メールの本文テキストを含むストリング・オブジェクト。項目属性が必要です。この属性が存在しない場合は、例外がスローされます。

from この属性は、メールの「*送信者*」フィールドの内容を指定します。この属性が存在する場合、この属性は「**送信者**」関数コンポーネント・パラメーターの値を指定変更します。

recipients

この属性は、メールの受信者をコンマで区切ったリストでなければなりません。この属性が存在する場合、この属性は「**受信者**」関数コンポーネント・パラメーターの値を指定変更します。

smtpServerHost

この属性は、メールの送信に使用される SMTP サーバーのアドレスを指定します。この属性が存在する場合、この属性は「**SMTP サーバー・ホスト**」関数コンポーネント・パラメーターの値を指定変更します。

smtpServerPort

この属性は、メールの送信に使用される SMTP サーバーのポートを指定します。この属性が存在する場合、この属性は「**SMTP サーバー・ポート**」関数コンポーネント・パラメーターの値を指定変更します。

Subject

この属性は、メールの件名を指定します。この属性が存在する場合、この属性は「**件名**」関数コンポーネント・パラメーターの値を指定変更します。このフィールドの値は、属性としてマップして提供するか、または「**件名**」関数コンポーネント・パラメーターで提供する必要があります。それ以外の場合は、例外がスローされます。

replyTo

この属性は、メッセージ・オブジェクトの「reply-to」フィールドを指定します。この属性には、コンマで区切られたメール・アドレスの配列を表すストリング・オブジェクトが含まれています。このストリング・パラメーターは

InternetAddress オブジェクトに変換されます。その後、作成されたアドレスは、メッセージ・オブジェクトの setReplyTo メソッドを使用して出力メッセージに設定されます。

入力スキーマ

status メールが正常に送信された場合 (つまり、SMTP サーバーによって受信された場合)、この属性は、値「OK」を含む java.lang.String オブジェクトになります。

構成

SendEMail 関数コンポーネントでは、ここに記載されているパラメーターが使用されます。

SMTP サーバー・ホスト
このパラメーターでは、メールを送信する SMTP サーバーのアドレスを指定します。このパラメーターを設定しない場合は、「smtpServer」項目属性をマップする必要があります。

SMTP サーバー・ポート
このパラメーターは、メールを送信する SMTP サーバーのポートを指定します。このパラメーターを設定しない場合は、「smtpServerPort」項目属性をマップする必要があります。マップされた「smtpServerPort」項目属性は、このパラメーターが設定されていてもこのパラメーターよりも優先されます。

ユーザー名
このパラメーターは、SMTP 認証に使用するユーザー名です。SMTP サーバーでユーザー名/パスワード認証が不要な場合は、このパラメーターに値を入力しないでください。

パスワード
このパラメーターは、SMTP 認証に使用されるパスワードです。

SSL の使用
このパラメーターにチェック・マークを付けると、関数コンポーネントは Secure Sockets Layer (SSL) を使用して SMTP サーバーと通信します。

送信者 E メール内の「**送信者**」フィールドの内容を指定します。このパラメーターを設定しない場合は、from 項目属性をマップする必要があります。from パラメーターにはスペースを含めることはできません。

受信者 このパラメーターは、受信者のアドレスをコンマで区切ったリストです。このパラメーターを設定しない場合は、「recipients」項目属性をマップする必要があります。

件名 E メールの件名を指定します。

添付ファイル
この多値パラメーターを使用すると、メッセージと一緒に含めたい任意のファイルをいくつでも添付することができます。それぞれの値は、絶対ファイル・パスか、作業ディレクトリーを起点とする相対ファイル・パスです。個

々の添付ファイルごとに異なる MIME タイプを設定する場合は、ファイル名の後に MIME 添付タイプを追加します。MIME タイプとファイル名は、> 文字で区切ります。例を示します。

```
SomeDocument.pdf>application/pdf
```

MIME コンテンツ・タイプ
このパラメーターを指定すると、E メール本文の MIME コンテンツ・タイプを設定できます。デフォルト値は `text/plain` です。

MIME 文字セット
エンコードされた単語およびテキスト部分に使用する MIME 文字セットを指定します。ブランクのままにすると、デフォルトのシステム文字セットが使用されます。サポートされているエンコードについては、http://java.sun.com/j2se/1.5/docs/guide/intl/encoding.doc.html を参照してください。

返信先 このパラメーターは、「返信先」アドレスをコンマで区切ったリストです。このパラメーターを設定しない場合は、「`replyTo`」項目属性をマップする必要があります。このパラメーターはオプションです。

デバッグ
デバッグ・メッセージを有効にします。このパラメーターは、すべての IBM Security Directory Integrator コンポーネントに対してグローバルに定義されます。

メモリー・キュー関数コンポーネント

多くの場合、MemQueue 関数コンポーネントと呼ばれています。メモリー・キュー関数コンポーネントは、(API に存在する) IBM Security Directory Integrator メモリー・バッファー・パイプの機能をカプセル化し、それを構成するための GUI を提供します。ここに記載されている情報とリンクを使用することで、メモリー・キュー関数コンポーネントについて詳しく知ることができます。

この関数コンポーネントには、ロー関数コンポーネントと、構成エディター (GUI) コンポーネントという 2 つの部分があります。ロー関数コンポーネントは、メモリー・バッファー・パイプに対する呼び出しをカプセル化します。GUI は、メモリー・バッファー・パイプの動作を構成する手段をユーザーに提供します。この関数コンポーネントは、メモリー・バッファー・パイプへの参照を戻すか、メモリー・バッファー・パイプの読み取り/書き込みを行います。

同一のキューに複数の読み取りプロセスと書き込みプロセスがある場合もあります。各書き込みプロセスは、データを追加する前にロックを取得する必要があります。読み取りプロセスがロックにアクセスするには、その前に書き込みプロセスがロックを解放する必要があります。

注: このリリースでは、メモリー・キュー関数コンポーネントを直接使用することは推奨されません。新規パイプを作成し、このパイプにデータを追加し、パイプにデータを書き込むには、276 ページの『メモリー・キュー・コネクター』を使用するか、または 683 ページの『システム・オブジェクト』を直接使用する方法が推奨されており、かつこの方法の方が容易です。この機能の API は、システム・オブジェクトで公開されています。

構成

ここに記載されているパラメーターを使用して、MemoryQueue 関数コンポーネントを構成することができます。

インスタンス名
メモリー・バッファー・パイプの作成先 IBM Security Directory Integrator インスタンスの名前。これをブランクにすると (デフォルト)、現在のインスタンスが想定されます。

パイプ名
選択したインスタンス内に作成されるメモリー・バッファー・パイプの名前。

使用するメモリーのパーセンテージ
これにより、メモリー・キューで利用することのできるメモリーのパーセンテージが決定されます。デフォルトは 50 です。

ウォーターマーク
これは、オブジェクトをシステム・ストアに格納するときのしきい値です。ページが実際に書き込まれる時期は「**ページ・サイズ**」パラメーターによって決定されるため、「ウォーターマーク」は「ページ・サイズ」の倍数にする必要があります。

ページ・サイズ
1 ページ内の項目数。

データベース名
使用する外部データベースの JDBC URL、またはデフォルトのシステム・ストアの場合はブランク (デフォルト)。

ユーザー名
使用するデータベースのログイン・ユーザー名。

パスワード
使用するデータベースのログイン・パスワード。

テーブル名
ページング用に使用する表。

詳細ログ
追加のログ・メッセージが必要な場合は、チェックします。

関数コンポーネントの使用

ここに記載されている情報とリンクを参照して、メモリー・キュー関数コンポーネントの使用方法を理解することができます。

683 ページの『システム・オブジェクト』にある *getFunction(string name)* というメソッドは、関数コンポーネントの初期化済みのインスタンスを戻します。戻されたオブジェクトは、次のような呼び出しを実行するために使用できます。

単純な呼び出しを使用するには、以下を使用します。

```
MemBufferQ pipe = system.getFunction("ibmdi.MemQueueFC").perform(null);
```

Entry を使用して呼び出すには、以下を使用します。

```
var inp = system.newEntry();
inp.setAttribute ("test", "this is a sample entry");
MemBufferQ pipe = system.getFunction("ibmdi.MemQueueFC").perform(inp);
```

メモリー・バッファー・キュー関数コンポーネントは、NULL の項目オブジェクトを渡されるとメモリー・バッファー・パイプへの参照を戻し、空の項目オブジェクトを渡されるとメモリー・バッファー・パイプに対して読み取り操作を実行し、空でない項目オブジェクトを渡されるとメモリー・バッファー・キューに対して書き込み操作を実行します。

戻された MemBufferQ オブジェクトに含まれている 2 つのメソッド `purgeQueue()` と `deletePipe()` は、オブジェクト管理に役立ちます。

関連情報

276 ページの『メモリー・キュー・コネクター』

346 ページの『システム・キュー・コネクター』

Axis Java から Soap への変換関数コンポーネント

Axis Java から SOAP への変換関数コンポーネント (FC) は、IBM Security Directory Integrator Web Services Suite に含まれます。このコンポーネントは、Web サービスのクライアント側と、Web サービスのサーバー側の両方で使用できます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

このコンポーネントは、項目または Java オブジェクトを受け取り、SOAP 要求 (クライアント上) または応答 (サーバー上) メッセージを作成します。 SOAP メッセージ全体を提供するだけでなく、処理とカスタマイズを容易にするために SOAP ヘッダーと SOAP 本体を別々に提供することもできます。

コンポーネントは、RPC スタイルと文書スタイルの両方をサポートします。

構成

以下に示すパラメーターを使用して、Axis Java から Soap への変換関数コンポーネントを構成することができます。

パラメーター

WSDL URL
サービスを記述している WSDL 文書の URL。

SOAP 操作
WSDL ファイルに記述されている SOAP 操作の名前。

XML の戻りタイプ
このドロップダウン・リストでは、結果を String として戻すか、DOM エレメントとして戻すかを指定します。

複素数タイプ
このパラメーターはオプションです。これを指定する場合は、完全修飾され

た Java クラス名 (パッケージ名を含む) のリストを指定する必要があります。このリストでは、各エレメント (Java クラス) を 1 つ以上の記号 (コンマ、セミコロン、スペース、復帰、および改行) で区切ります。

モード この関数コンポーネントが SOAP **要求**を生成するか (クライアント上にデプロイされた場合)、SOAP **応答**を生成するか (サーバー上にデプロイされた場合) を指定するフラグ。

多重参照を使用する
このパラメーターは、RPC スタイルの Web サービスが使用される場合にのみ有効です。文書スタイルの Web サービスが使用される場合は、このパラメーターは生成される SOAP メッセージに影響しません。

これをチェックした場合は、RPC スタイルの Web サービスが使用され、生成される SOAP メッセージでは多重参照が使用されます。これを**チェックしない**場合に RPC スタイルの Web サービスが使用されると、生成される SOAP メッセージでは多重参照が**使用されません**。.

操作パラメーター
このパラメーターは、属性名のリストです。このリストでは、各属性名を 1 つ以上の記号 (コンマ、セミコロン、スペース、復帰、および改行) で区切ります。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの入力

Axis Java から Soap への変換関数コンポーネントでは、以下に示す入力を使用することができます。

項目 または *Java* オブジェクト。これら以外のものを渡すと、例外がスローされます。

関数コンポーネントの出力

Axis Java から Soap への変換関数コンポーネントでは、以下に示す出力を使用することができます。

SOAP メッセージ全体用の属性、SOAP ヘッダー用の属性、SOAP 本体用の属性という 3 つの属性が組み込まれた項目オブジェクト。SOAP メッセージ、本体、ヘッダーは、「**XML の戻りタイプ**」パラメーターの指定に応じて、XML ストリングか DOM オブジェクトのいずれかです。

関数コンポーネントの使用

この関数コンポーネント (FC) を使用すると、SOAP メッセージの Java 表現をシリアライズして、その SOAP メッセージの XML 表現に変換することができます。

- この関数コンポーネントに (a) タイプ `org.apache.axis.AxisFault` の値を設定した *soapFault* 属性を組み込んだ項目オブジェクト、または (b) タイプ `org.apache.axis.AxisFault` の Java オブジェクトを渡した場合、関数コンポー

ネントは、渡された *AxisFault* オブジェクトに格納されている情報を入れた SOAP フォールト・メッセージを生成します。

- 関数コンポーネントの「**XML の戻りタイプ**」パラメーターの値が **String** である場合、SOAP 応答メッセージは *xmlString* 属性に格納されます (関数コンポーネントに*項目* を渡した場合)。ただし、関数コンポーネントの「**XML の戻りタイプ**」パラメーターの値が **DOMElement** である場合、生成された SOAP メッセージは *xmlDOMElement* 属性に格納されます (関数コンポーネントに *項目* を渡した場合)。この関数コンポーネントに *Java* オブジェクト の配列 (Object[]) を渡した場合、関数コンポーネントの戻り値は `java.lang.String` オブジェクト (関数コンポーネントの「**XML の戻りタイプ**」パラメーターが **String** の場合)、または `org.w3c.dom.Element` オブジェクト (そのパラメーターの値が **DOMElement** の場合) のいずれかになります。
- 渡された *soapFault* 属性のタイプが `org.apache.axis.AxisFault` でない場合、例外がスローされます。
- 関数コンポーネントの「**操作パラメーター**」パラメーターの値に含まれる各項目は、属性の名前です。関数コンポーネントに渡す項目にはそれらの属性が存在している必要があります。いずれかの属性が欠落していると、例外がスローされます。
- 関数コンポーネントに *Java* オブジェクト の配列 (Object[]) を渡した場合、関数コンポーネントは、その配列に Java オブジェクトが格納されている順序で配列から各オブジェクトを取り出して SOAP 操作に渡します。関数コンポーネントに *項目* を渡した場合、SOAP 操作に渡されるパラメーターの順序と値は、関数コンポーネントの「**操作パラメーター**」パラメーターの値によって決まります。
- 関数コンポーネントの「**操作パラメーター**」パラメーターの値に含まれる項目の順序により、属性値がパラメーターとして SOAP 操作に渡される順序が決まります。
- この関数コンポーネントは、(a) 文書スタイルの SOAP メッセージ、(b) RPC スタイルの SOAP メッセージ、および (c) SOAP フォールト・メッセージを生成することが可能です。生成されるメッセージのスタイルは、関数コンポーネントの「**WSDL URL**」パラメーターで指定された WSDL によって決まります。
- この関数コンポーネントは、SOAP メッセージ生成のエンコードに、「リテラル」エンコード方式および SOAP セクション 5 エンコード方式の両方を使用できます。生成されるメッセージのエンコードは、関数コンポーネントの「**WSDL URL**」パラメーターで指定された WSDL によって決まります。
- 「**多重参照を使用する**」パラメーターが異なる内容を指定することがありますが、このパラメーターは RPC スタイルのメッセージにのみ適用されます。文書スタイルの Web サービスが使用される場合、このパラメーターは生成される SOAP メッセージに影響しません。これをチェックした場合は、RPC スタイルの Web サービスが使用され、生成される SOAP メッセージでは多重参照が使用されます。これを**チェックしない**場合に RPC スタイルの Web サービスが使用されると、生成される SOAP メッセージでは多重参照が**使用されません**。

  注: 「**多重参照を使用する**」パラメーターは、Axis ライブラリーを使用してこの関数コンポーネントをインプリメントしたために導入されました。現時点では、Axis Java から Soap への変換関数コンポーネントでは RPC スタイルのメッセージをシリアライズするときに、生成される SOAP で XML href/多重参照が使用されます。これにより、Axis C++ ライブラリーが中断します。このため、href/多重

参照をオンまたはオフに切り替えるための Axis Java から SOAP への変換関数コンポーネント構成パラメーターが提供されています。

- この関数コンポーネントは、WSDL 文書の <types> セクションで定義されている複素数タイプの値を含む SOAP メッセージを生成できます。これを行うために、この関数コンポーネントは、(1) 関数コンポーネントの「**複素数タイプ**」パラメーターに、SOAP 操作に対するパラメーターとして使用される複素数タイプをインプリメントする Java クラスすべての名前が含まれていること、(2) それらの Java クラスのクラス・ファイルが IBM Security Directory Integrator の Java クラス・パスに配置されていることを必要とします。
- この関数コンポーネントに*項目* オブジェクトを渡した場合、関数コンポーネントは、生成した SOAP メッセージ・ヘッダーと SOAP メッセージ本体を (生成した SOAP メッセージ全体とは別に) 属性として*項目* オブジェクトに入れて戻します。関数コンポーネントの「**XML の戻りタイプ**」パラメーターの値が **String** である場合、SOAP ヘッダーと SOAP 本体はそれぞれ *soapHeaderString* 属性と *soapBodyString* 属性に `java.lang.String` オブジェクトとして格納されます。関数コンポーネントの「**XML の戻りタイプ**」パラメーターの値が「**DOMElement**」である場合、SOAP ヘッダーと SOAP 本体はそれぞれ *soapHeaderDOMElement* 属性と *soapBodyDOMElement* 属性に `org.w3c.dom.Element` オブジェクトとして格納されます。

カスタム・シリアライザー/デシリアライザー

シリアライゼーションでは、Java オブジェクトを XML エレメントに変換することができます。デシリアライゼーションでは、XML エレメントを Java オブジェクトに変換できます。

AxisJavaToSoap 関数コンポーネントと AxisSoapToJava 関数コンポーネントには、XML タイプから Java タイプへのマッピングをカスタム・シリアライザー/デシリアライザーに登録するためのメソッドがあります (デフォルトでは、すべての複素数タイプが Axis の org.apache.axis.encoding.ser.BeanSerializer/ org.apache.axis.encoding.ser.BeanDeserializer によりシリアライズ/シリアライズ解除されます)。

```
/**
   * This method is analogous to the 'registerTypeMapping' method in org.apache.axis.client.Call.
   * It can be used for configuring serialization/deserialization of Java types, for which the
   * default serializer/deserializer (org.apache.axis.encoding.ser.BeanSerializer/
   * org.apache.axis.encoding.ser.BeanDeserializer) is not suitable.
   */
    public void registerTypeMapping(Class javaType,
                                    QName xmlType,
                                    SerializerFactory serializerFactory,
                                    DeserializerFactory deserializerFactory)
```

このメソッドは、以下のような JavaScript を使用して、「初期化後」プロローグ関数コンポーネント・フックで呼び出すことができます。

```
var myClass = java.lang.Class.forName("mypackage.MyClass");
var myQName = new javax.xml.namespace.QName("http://www.myserver.com", "MyClass");
var mySerializerFactory = new mypackage.MySerializerFactory();
var myDeserializerFactory = new mypackage.MyDeserializerFactory();

myFC.getFunction().registerTypeMapping(myClass, myQName, mySerializerFactory, myDeserializerFactory);
```

シリアライゼーション/デシリアライゼーションの問題

以下に示す情報と例を使用して、シリアライゼーションとデシリアライゼーションの問題について詳しく知ることができます。

デフォルトでは、すべての複素数タイプが Axis の org.apache.axis.encoding.ser.BeanSerializer/org.apache.axis.encoding.ser.BeanDeserializer によりシリアライズ/シリアライズ解除されます。これらのデフォルトのシリアライザーおよびデシリアライザーは、特定のまれなケースには適していません。シリアライゼーション/デシリアライゼーションのエラーが発生した場合、カスタム・シリアライザー/デシリアライザーの指定が必要となる可能性があります。

以下に、既知のケースの 1 つを示します。

XML スキーマ *list* タイプ

WSDL 文書の XML スキーマが、*list* タイプになるようにエレメントを以下のように定義する場合。

```
<s:simpleType name="MyListType">
 <s:list>
   <s:simpleType>
     <s:restriction base="s:string">
       <s:enumeration value="One" />
       <s:enumeration value="Two" />
       <s:enumeration value="Three" />
     </s:restriction>
   </s:simpleType>
 </s:list>
</s:simpleType>
```

... 以下のようなエラーが発生します。

```
at org.apache.axis.encoding.ser.ArrayDeserializer.characters(ArrayDeserializer.java:502)
 at org.apache.axis.encoding.DeserializationContext.characters(DeserializationContext.java:966)
 at org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.java:177)
 at org.apache.axis.message.MessageElement.publishToHandler(MessageElement.java:1141)
 at org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
 at org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
 at org.apache.axis.client.Call.invoke(Call.java:2467)
 at org.apache.axis.client.Call.invoke(Call.java:2366)
 at org.apache.axis.client.Call.invoke(Call.java:1812)
 at com.ibm.di.fc.webservice.AxisEasyInvokeSoapWS.perform(Unknown Source)
```

問題の原因は、`org.apache.axis.encoding.ser.ArrayDeserializer` が *xsd:list* タイプに適していないことです。`org.apache.axis.encoding.ser.SimpleListDeserializer` デシリアライザーを代わりに使用する必要があります。AxisJavaToSoap/AxisSoapToJava 関数コンポーネントの「**初期化後**」フックで以下のようなスクリプトを使用することで、問題を解決できます。

```
var javaType = java.lang.Class.forName("[Ljava.lang.String;");
var xmlType = new javax.xml.namespace.QName("http://www.example.com", "MyListType");;
var serializerFactory = new org.apache.axis.encoding.ser.SimpleListSerializerFactory(javaType, xmlType);
var deserializerFactory = new org.apache.axis.encoding.ser.SimpleListDeserializerFactory(javaType, xmlType);
thisConnector.getFunction().registerTypeMapping(javaType, xmlType, serializerFactory, deserializerFactory);
```

関連情報

541 ページの『Axis Soap から Java への変換関数コンポーネント』

WrapSoap 関数コンポーネント

WrapSoap 関数コンポーネント (FC) は、IBM Security Directory Integrator Web Services Suite に含まれます。ここに記載されている情報とリンクを使用することで、WrapSoap 関数コンポーネントについて詳しく知ることができます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

このコンポーネントは、SOAP 本体およびオプションで SOAP ヘッダーを含む完全な SOAP メッセージの生成に使用されます。

このようなコンポーネントは、ユーザーが SOAP 本体の内容をカスタマイズする場合や、完全に自力で (例えば Castor のバインディングを使用して) 作成する場合に役立ちます。このコンポーネントは、SOAP 本体および SOAP ヘッダーの内容や、SOAP エンベロープ、ヘッダー、本体の XML エレメントの属性 (通常はネーム・スペース宣言) を受け入れて、完全な SOAP メッセージを作成します。

このコンポーネントは、実際には、SOAP データを完全な SOAP メッセージにラップするために String オブジェクトまたは DOM オブジェクトを操作するという、エラーの発生しやすい作業をユーザーが行わないで済むようにするヘルパー関数コンポーネントです。

構成

ここに記載されているパラメーターを使用して、ラップ Soap 関数コンポーネントを構成することができます。

パラメーター

SOAP 本体と SOAP ヘッダーを次のタイプとして入力する
このドロップダウンは、SOAP 本体および SOAP ヘッダーの入力値が String オブジェクト (つまり、`java.lang.String`) として渡されるのか、DOM オブジェクト (`org.w3c.dom.Node`) として渡されるのかを指定します。

SOAP メッセージを次のタイプとして戻す
このドロップダウンでは、完全な SOAP メッセージを String として戻すか、DOM オブジェクトとして戻すかを指定します。

Header タグおよび Body タグを組み込む
属性で渡される SOAP 本体に <Body> タグが含まれているかどうか、および属性で渡される SOAP ヘッダーに <Header> タグが含まれているかどうかを指定します。

SOAP エンベロープに追加する属性
SOAP エンベロープの XML エレメントに組み込む XML 属性とその値を指定します。

SOAP エンベロープに追加するネーム・スペース宣言
SOAP エンベロープに追加するネーム・スペース宣言を指定します。

SOAP 本体に追加する属性
SOAP 本体の XML エレメントに組み込む XML 属性とその値を指定します。

SOAP 本体に追加するネーム・スペース宣言
SOAP 本体に追加するネーム・スペース宣言を指定します。

SOAP ヘッダーに追加する属性
SOAP ヘッダーの XML エレメントに組み込む XML 属性とその値を指定します。

SOAP ヘッダーに追加するネーム・スペース宣言
SOAP ヘッダーに追加するネーム・スペース宣言を指定します。

詳細ログ
追加のログ・メッセージを生成する場合は、チェックします。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの入力

ラップ Soap 関数コンポーネントに対して、ここに記載されている入力を使用することができます。

項目 オブジェクト。SOAP ヘッダー用に 1 つの属性と (オプション)、SOAP 本体用に 1 つの属性を組み込みます。

これら以外のものを渡すと、例外がスローされます。

関数コンポーネントの出力

ラップ Soap 関数コンポーネントに対して、ここに記載されている出力を使用することができます。

完全な SOAP メッセージを含む*項目* オブジェクト。

関数コンポーネントの使用

この関数コンポーネントによって処理されて返される項目のタイプと形式を使用することができます。このタイプと形式は、以下に記載するように、指定されたパラメーターによって大きく異なります。

- 関数コンポーネントの「**SOAP の本体とヘッダーの入力タイプ**」パラメーターが **String** である場合、SOAP 本体は *soapBodyString* 属性に入れて渡し、SOAP ヘッダーは *soapHeaderString* 属性に入れて渡します。関数コンポーネントの「**SOAP の本体とヘッダーの入力タイプ**」パラメーターが **DOMElement** である場合、SOAP 本体は *soapBodyDOMElement* 属性に入れて渡し、SOAP ヘッダーは *soapHeaderDOMElement* 属性に入れて渡します。
- 関数コンポーネントの「**SOAP メッセージの戻りタイプ**」パラメーターが **String** である場合、完全な SOAP メッセージは *xmlString* 属性に入れて戻されます。そのパラメーターに **DOMElement** が指定されている場合は、完全な SOAP メッセージは *xmlDOMElement* 属性に入れて戻されます。
- 「**... に追加する属性**」というパラメーターにはそれぞれ、作成する SOAP メッセージのターゲット SOAP メッセージ・エレメント (エンベロープ、ヘッダー、または本体) のタグに追加する XML 属性のリストを指定します。属性と値の各ペアは、他のペアとの間にスペース、コンマ、セミコロン、復帰、または改行のいずれかの記号を挿入して区切ります。属性と値のペアでは、属性名と属性値を等号「=」で区切ります。
- 「**... に追加するネーム・スペース宣言**」というパラメーターにはそれぞれ、作成する SOAP メッセージの SOAP メッセージ・エレメント (エンベロープ、ヘッダー、または本体) のタグに追加する XML ネーム・スペース宣言のリストを指定します。ネーム・スペース接頭部と値の各ペアは、他のペアとの間にスペー

ス、コンマ、セミコロン、復帰、または改行のいずれかの記号を挿入して区切ります。接頭部と値のペアでは、ネーム・スペースの接頭部とネーム・スペースの値を等号「=」で区切ります。

InvokeSoap WS 関数コンポーネント

InvokeSoap WS 関数コンポーネントを使用して、呼び出しの入力メッセージを指定して Web サービス呼び出しを実行することができます。

Axis InvokeSoapWS 関数コンポーネント (FC) は、IBM Security Directory Integrator Web Services Suite に含まれます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

組み込み SOAP 解析機能はありませんが、541 ページの『Axis Soap から Java への変換関数コンポーネント』および 530 ページの『Axis Java から Soap への変換関数コンポーネント』と組み合わせて使用することで、完全な Web サービス・ソリューションを実現します。

InvokeSoapWS 関数コンポーネントには、完全な SOAP 要求メッセージが必要です。 SOAP メッセージを指定してこの関数コンポーネントが呼び出されると、関数コンポーネントはこのメッセージを使用してリモート Web サービス操作を呼び出します。この関数コンポーネントは SOAP 応答メッセージを戻します。ただし、この関数コンポーネントは SOAP 応答メッセージを戻すだけであり、XML-Java バインディングは実行しません (つまり、SOAP 応答メッセージは解析されません)。

認証

InvokeSoapWS 関数コンポーネントは HTTP 基本認証 (BA) メソッドをサポートしています。ここに記載されている情報を参照して、このメソッドについて詳しく理解することができます。

ユーザー名とパスワードのパラメーターに値が入力されると、(HTTP 基本認証を使用する場合の HTTP 仕様に基づいて) HTTP ヘッダー・フィールド「authorization」に、適切な信任状が設定されます。ユーザー名とパスワードは送信前に関数コンポーネントによりエンコードされます。エンコードとして Base64 が使用され、エンコードが InvokeSoapWS 関数コンポーネント内部で実行されます。

構成

ここに記載されているパラメーターを使用することで、InvokeSoapWS 関数コンポーネントを構成できるようになります。

パラメーター

WSDL URL
サービスを記述している WSDL 文書の URL。これは必須パラメーターです。指定しないと、初期化時に例外がスローされます。

SOAP 操作
WSDL ファイルに記述されている SOAP 操作の名前です。これは必須パラメーターです。指定しないと、初期化時に例外がスローされます。

プロバイダー URL
Web サービス・プロバイダーの URL。WSDL の値を置き換えます。このパラメーターは、動的プロバイダー切り替えを使用可能にする目的で提供されています。このパラメーターが**空**の場合は WSDL の値が使用されます。

ログイン・ユーザー名
HTTP 基本認証を使用してサーバーに送信されるログイン・ユーザー名。サーバーで許可が必要な場合は、この値と次の値 (ログイン・パスワード) を使用してクライアントが認証されます。エンコードとして Base64 が使用され、エンコードが InvokeSoapWS 関数コンポーネント内部で実行されます。

ログイン・パスワード
HTTP 基本認証を使用してサーバーに送信されるログイン・パスワード。サーバーで許可が必要な場合は、この値と前の値 (ログイン・ユーザー名) を使用してクライアントが認証されます。

SOAP メッセージを次のタイプとして入力
このドロップダウン・リストでは、SOAP 要求メッセージを String として関数コンポーネントに渡すか、DOM オブジェクトとして渡すかを指定します。このパラメーターは必須です。

SOAP メッセージを次のタイプとして戻す
このドロップダウン・リストでは、SOAP 応答メッセージを String として戻すか、DOM オブジェクトとして戻すかを指定します。このパラメーターは必須です。このパラメーターが指定されていないか、または無効な値が指定されていると、初期化時に例外がスローされます。また、SOAP 要求メッセージがこのパラメーターにより指定されるフォーマットに従っていない場合は、エラーとなります。ただし、片方向 Web サービス操作を呼び出す場合はこれは無視されます。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの入力

ここに記載されている項目を InvokeSoapWS 関数コンポーネントの入力として使用することができます。

項目、`java.lang.String` オブジェクト、または `org.w3c.dom.Element` オブジェクトは、完全な SOAP 要求メッセージを含みます。

これら以外のものを渡すと、例外がスローされます。

関数コンポーネントに項目を渡す場合、関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値に **String** を設定してあるときには、その項目の *xmlString* 属性に SOAP 要求メッセージを格納する必要があります。関数コンポー

ネントに*項目* を渡す場合、関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値に **DOMElement** を設定してあるときには、*xmlDOMElement* 属性に SOAP 要求メッセージを格納する必要があります。

関数コンポーネントに*項目* 以外のオブジェクト (String または Element のいずれか) を渡す場合、関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値に **String** を設定してあるときには、SOAP 要求メッセージを `java.lang.String` オブジェクトとして渡す必要があります。関数コンポーネントに項目以外のオブジェクト (String または Element のいずれか) を渡す場合、関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値に **DOMElement** を設定してあるときには、SOAP 要求メッセージを `org.w3c.dom.Element` オブジェクトとして渡す必要があります。

関数コンポーネントの出力

ここに記載されている項目を InvokeSoapWS 関数コンポーネントの入力として使用することができます。

SOAP メッセージ全体用の属性、SOAP ヘッダー用の属性、SOAP 本体用の属性という 3 つの属性が組み込まれた項目オブジェクト。SOAP メッセージ、本体、ヘッダーは、「**SOAP メッセージの戻りタイプ**」パラメーターの指定に応じて、XML ストリングか DOM オブジェクトのいずれかです。次のセクション『関数コンポーネントの使用』を参照してください。

関数コンポーネントの使用

この関数コンポーネントを使用して SOAP 要求メッセージを送信し、SOAP 応答メッセージを受信することにより、Web サービスを呼び出すことができます。

- 関数コンポーネントに*項目* を渡した場合、関数コンポーネントの「*SOAP メッセージの戻りタイプ*」パラメーターの値に *String* が設定されていると、SOAP 応答メッセージは *xmlString* 属性に格納されます。ただし、関数コンポーネントの「*SOAP メッセージの戻りタイプ*」パラメーターの値に *DOMElement* が設定されていると、SOAP 応答メッセージは *xmlDOMElement* 属性に格納されます。
- また、この関数コンポーネントに*項目* オブジェクトを渡した場合、関数コンポーネントは、SOAP 応答のヘッダーと SOAP 応答の本体を (SOAP 応答メッセージ全体とは別に) 属性として*項目* オブジェクトに入れて戻します。関数コンポーネントの「**SOAP メッセージの戻りタイプ**」パラメーターの値が **String** である場合、SOAP ヘッダーと SOAP 本体はそれぞれ *soapHeaderString* 属性と *soapBodyString* 属性に `java.lang.String` オブジェクトとして格納されます。関数コンポーネントの「**SOAP メッセージの戻りタイプ**」パラメーターの値が **DOMElement** である場合、SOAP ヘッダーと SOAP 本体はそれぞれ *soapHeaderDOMElement* 属性と *soapBodyDOMElement* 属性に `org.w3c.dom.Element` オブジェクトとして格納されます。
- この関数コンポーネントに*項目* 以外のオブジェクトを渡した場合、関数コンポーネントの戻り値は `java.lang.String` オブジェクト (関数コンポーネントの「*SOAP メッセージの戻りタイプ*」パラメーターが *String* の場合)、または `org.w3c.dom.Element` オブジェクト (そのパラメーターの値が *DOMElement* の場合) のいずれかになります。

- この関数コンポーネントは、SOAP メッセージの送信と受信に、「リテラル」エンコード方式および SOAP セクション 5 エンコード方式の両方を使用できます。
- この関数コンポーネントは、WSDL 文書の <types> セクションで定義されている複素数タイプの値を含む SOAP メッセージを送信および受信できます。
- この関数コンポーネントは、SOAP 要求メッセージの「*soapAction*」HTTP ヘッダーに、指定された SOAP 操作 (関数コンポーネントの「*SOAP 操作*」パラメーターに名前が指定されているもの) に対して WSDL 文書 (関数コンポーネントの「*WSDL URL*」パラメーターに位置が指定されているもの) で指定されている値を設定します。
- この関数コンポーネントは SOAP 要求メッセージを、「*WSDL URL*」パラメーターに指定されている Web サービス・アドレスに HTTP 経由で送信します。「*WSDL URL*」パラメーターが指定されていないかまたは空の場合は、指定の SOAP 操作 (その名前は、関数コンポーネント・パラメーター「*SOAP 操作*」で指定されています)の WSDL 文書 (その位置は、関数コンポーネント・パラメーター「*WSDL URL*」で指定されています) で指定されている Web サービス・アドレスが使用されます。
- この関数コンポーネントには、ユーザー名とパスワードのパラメーターがあります。これらのパラメーターが指定されている場合は、関数コンポーネントにより基本許可ヘッダーが設定され、サーバーに送信されます。指定されたユーザー名とパスワードを base64 エンコード方式を使用してエンコードします。このエンコード操作は、InvokeSoapWS 関数コンポーネント外部で実行されます。

片方向の Web サービス操作のサポート

WSDL 1.1 には、Web サービス・エンドポイントがサポート可能な 4 つの伝送プリミティブがあります。

一方向 エンドポイントはメッセージを受信します。

要求/応答
エンドポイントはメッセージを受信し、相関するメッセージを送信します。

送信請求/応答
エンドポイントはメッセージを送信し、相関するメッセージを受信します。

通知 エンドポイントはメッセージを送信します。

WSDL では伝送プリミティブが操作と呼ばれています。(この点についての詳細は、http://www.w3.org/TR/wsdl#_porttypes を参照してください。)

InvokeSoapWS 関数コンポーネントでは、**要求/応答**および**片方向** Web サービス操作のみがサポートされています。初期化フェーズにおいて、InvokeSoapWS 関数コンポーネントは構成されている WSDL 文書を読み取り、指定されている SOAP 操作が片方向操作であるかどうかを確認します。一方向操作でない場合は、要求/応答であると想定されます。

要求/応答操作を記述した WSDL の断片のサンプルを以下に示します。

```
<operation name="myRequestResponseOperation">
 <input message="myInputMessage"/>
 <output message="myOutputMessage"/>
</operation>
```

片方向操作を記述した WSDL サンプル・フラグメントを以下に示します。

```
<operation name="myOneWayOperation">
 <input message="myInputMessage"/>
</operation>
```

注: 片方向 Web サービス操作では、サーバー応答は行われません。クライアントが要求メッセージを送信しますが、(フォールト・メッセージの場合も含め) サーバーは応答しません。このため、InvokeSoapWS は片方向 SOAP 操作の呼び出し時に応答を戻しません。この関数コンポーネントの「perform」メソッドに *項目* 引数が渡されると (例えば、関数コンポーネントが AssemblyLine の一部として実行される場合)、この関数コンポーネントは**空の項目**を戻します。この関数コンポーネントの perform メソッドに java.lang.Object が渡されると (例えば、関数コンポーネントがスクリプトによって実行される場合)、この関数コンポーネントは **null** を戻します。

関連情報

549 ページの『Axis EasyInvoke Soap WS 関数コンポーネント』

Axis Soap から Java への変換関数コンポーネント

このコンポーネントは、Web サービスのクライアント側と、Web サービスのサーバー側の両方で使用できます。ここに記載されている情報とリンクを使用することで、これについて詳しく知ることができます。

Axis SOAP から Java への変換関数コンポーネント (FC) は、IBM Security Directory Integrator Web Services Suite に含まれます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

この関数コンポーネントは、Axis のメカニズムを利用して、SOAP 応答 (クライアント上の場合) または SOAP 要求 (サーバー上の場合) を解析し、Java オブジェクトに変換します。その意味で、AxisJavaToSoap 関数コンポーネントと相補的な関係にあるコンポーネントです。この関数コンポーネントに SOAP 応答/要求メッセージを引き渡すと、解析された Java オブジェクトが、スタンドアロンの Java オブジェクトとして、または項目オブジェクトにカプセル化した形で戻されます。

このコンポーネントは、RPC スタイルと文書スタイルの両方をサポートします。

構成

以下に示すパラメーターを使用して、Axis SOAP から Java への変換関数コンポーネントを構成することができます。

パラメーター

WSDL URL
: サービスを記述している WSDL 文書の URL。

SOAP 操作
: WSDL ファイルに記述されている SOAP 操作の名前。

SOAP メッセージを次のタイプとして入力
このドロップダウン・リストでは、SOAP メッセージを String として指定するか、DOM オブジェクトとして指定するを設定します。

複素数タイプ
このパラメーターはオプションです。これを指定する場合は、完全修飾された Java クラス名 (パッケージ名を含む) のリストを指定する必要があります。このリストでは、各エレメント (Java クラス) を 1 つ以上の記号 (コンマ、セミコロン、スペース、復帰、および改行) で区切ります。

モード この必須パラメーターは、値「**要求**」または「**応答**」のいずれかをとります。このパラメーターの値は、この関数コンポーネントが SOAP 要求メッセージまたは SOAP 応答メッセージのどちらを解析するかを指定します。

詳細ログ
これをチェックした場合、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

関数コンポーネントの入力

以下に示す項目を、Axis Java から Soap への変換関数コンポーネントの入力として渡すために使用することができます。

完全な SOAP メッセージを表す*項目* または *Java オブジェクト*。

これら以外のものを渡すと、例外がスローされます。

関数コンポーネントの出力

以下に示す項目を、Axis Java から Soap への変換関数コンポーネントの出力として渡すために使用することができます。

SOAP 要求/応答の Java 表現が含まれる*項目* または *Java オブジェクト*。

関数コンポーネントの使用

この関数コンポーネントは、SOAP メッセージを解析し、Java オブジェクトに変換します。実行できるタスクのリストをここで参照できます。

- この関数コンポーネントに SOAP フォールト・メッセージを渡して解析させると、タイプ `org.apache.axis.AxisFault` の Java オブジェクトが戻されます。
- この関数コンポーネントが `org.apache.axis.AxisFault` オブジェクトを戻す場合、入力として*項目* を渡されたケースでは、そのオブジェクトを *soapFault* 属性に入れて戻します。入力として *java.lang.Object* を渡されたケースでは、`org.apache.axis.AxisFault` オブジェクトを戻します。
- 関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値が **String** である場合、解析対象の SOAP メッセージは *xmlString* 属性 (タイプ `java.lang.String`) から読み取られます (関数コンポーネントに*項目* を渡した場合)。関数コンポーネントの「**SOAP メッセージ入力タイプ**」パラメーターの値が **DOMElement** である場合、解析対象の SOAP メッセージは *xmlDOMElement* 属性 (`org.w3c.dom.Element` オブジェクト) から読み取られます (関数コンポーネントに項目を渡した場合)。

- この関数コンポーネントに *Java* オブジェクト を渡した場合、`java.lang.String` オブジェクト (関数コンポーネントの「**SOAP メッセージの入力タイプ**」パラメーターが **String** の場合) または `org.w3c.dom.Element` オブジェクト (関数コンポーネントの「**SOAP メッセージの入力タイプ**」パラメーターの値が **DOMElement** の場合) のいずれかとして渡された Java オブジェクトの値が、解析対象の SOAP メッセージであると想定されます。
- この関数コンポーネントは、(a) 文書スタイルの SOAP メッセージ、(b) RPC スタイルの SOAP メッセージ、および (c) SOAP フォールト・メッセージを解析することが可能です。
- この関数コンポーネントは、「リテラル」エンコード方式および SOAP セクション 5 エンコード方式のいずれでエンコードされた SOAP メッセージでも解析できます。
- この関数コンポーネントは、WSDL 文書の <types> セクションで定義されている複素数タイプの値を含む SOAP メッセージを解析できます。これを行うために、この関数コンポーネントは、(1) 関数コンポーネントの「**複素数タイプ**」パラメーターに、SOAP メッセージで使用される複素数タイプをインプリメントする Java クラスすべての名前が含まれていること、(2) それらの Java クラスのクラス・ファイルが IBM Security Directory Integrator の Java クラス・パスに配置されていることを必要とします。
- この関数コンポーネントに*項目* を渡した場合、解析されたメッセージが SOAP フォールト・メッセージ以外であると、この関数コンポーネントは、SOAP 操作の出力パラメーターの名前と一致する項目属性の出力パラメーターを戻します。

関連情報

533 ページの『カスタム・シリアライザー/デシリアライザー』
530 ページの『Axis Java から Soap への変換関数コンポーネント』

Axis2 WS クライアント関数コンポーネント

Axis2 WS クライアント関数コンポーネントは、Web サービス・クライアントです。これを使用して、実行中のサービスを呼び出すことができます。この関数コンポーネントは、Apache Axis2 ライブラリーを使用して、要求を Web サービスに送信し、Web サービスから応答を受信します。

WSDL 1.1 (http://www.w3.org/TR/wsdl/) 文書および WSDL 2.0 (http://www.w3.org/TR/wsdl20/) 文書の両方がサポートされています。

SOAP 1.1 および SOAP 1.2 の両方のプロトコルがサポートされています。使用できるのは*リテラル* SOAP メッセージのみで、エンコード SOAP メッセージはサポートされていません。これは、基礎になる Axis2 ライブラリー (バージョン 1.4.0.1) の制限です。

WSDL URL
Web サービス呼び出しに使用される WSDL ファイルのロケーション。

サービス
呼び出されるサービスの名前 (WSDL ファイルに書き込まれています)。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望するサービス名を選択することができるドロップダウ

ン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初に WSDL ファイルを指定する必要があります。これは、このスクリプトが、そのファイル内のサービスを表示するためです。

操作
呼び出される SOAP 操作の名前。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望する操作を選択することができるドロップダウン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初にサービスを指定する必要があります。これは、このスクリプトの目的がサービスに関連付けられている操作を表示することであるためです。

エンドポイント
呼び出される Web サービスに対応するエンドポイントの名前 (WSDL ファイルに書き込まれています)。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望するエンドポイント名を選択することができるドロップダウン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初にサービスを指定する必要があります。これは、このスクリプトが、サービスに関連付けられているエンドポイントを表示するためです。

ユーザー名
HTTP 基本認証呼び出しに使用されるユーザー名。

パスワード
HTTP 基本認証呼び出しに使用されるパスワード。

接続タイムアウト
接続タイムアウトをミリ秒で指定します。 デフォルトは 60000 (1 分) です。

このコネクターを構成する場合は、最初に WSDL ファイルを構成し、その後にサービスを選択します。次に、SOAP 操作とエンドポイントを選択します。

関数コンポーネントの使用

以下に示す情報を使用して、Axis2 WS クライアント関数コンポーネントの使用法を理解することができます。

Axis2 WS クライアント関数コンポーネントは、構成済みの Web サービスの操作を呼び出し、HTTP 転送プロトコルを使用してその応答を戻します。この関数コンポーネントは、WSDL 文書内の入力メッセージ・エレメントの後に指定された IBM Security Directory Integrator 階層属性にそのペイロードが含まれているものと想定します。Axis2 WS クライアント関数コンポーネントは、WSDL 文書内の出力メッセージ・エレメントの後に指定された別の階層属性で応答を戻します。

In-Only 操作の場合、応答階層オブジェクトは作成されず、空の項目が戻されます。

サポートされているメッセージ交換パターン

サポートされるメッセージ交換パターンのリストを以下で参照できます。

Axis2 Web サービス・クライアント関数コンポーネントは、以下のメッセージ交換パターンをサポートしています (詳しくは、 29 ページの『Axis2 Web サービス・サーバー・コネクター』を参照してください)。

In-Only
Axis2 ライブラリーは、WSDL ファイルではこのパターンが http://www.w3.org/ns/wsdl/in-only であることを要求します。

In-Out Axis2 ライブラリーは、WSDL ファイルではこのパターンが http://www.w3.org/ns/wsdl/in-out であることを要求します。

Robust-In-Only
Axis2 ライブラリーは、WSDL ファイルではこのパターンが http://www.w3.org/ns/wsdl/robust-in-only であることを要求します。

SOAP ヘッダー

ここに記載されているリンクを使用することで、SOAP ヘッダーについて知ることができます。

33 ページの『SOAP ヘッダー』を参照してください。

スキーマ

Axis2 コンポーネント (Axis2WSClientFC および Axis2WSServerConector) は、階層構造を持つ属性の形式でデータを送受信します。これらの属性は、スクリプト・コンポーネントなどのコンポーネントで作成できますが、各属性の構造は、そのコンポーネントにパラメーターとして提供される WSDL 文書によって異なります。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
targetNamespace="http://www.example.com/HelloService.wsdl" >
...
<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc | document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation ... >
<input>
                <soap:body use="literal|encoded"? namespace="uri"?>
            </input>
            <output>
                <soap:body use="literal|encoded"? namespace="uri"?>
            </output>
        </operation>
</binding>
...
</definitions>
```

この文書が WSDL 1.1 標準に準拠している場合には、以下の 2 つのオプションがあります。

1. SOAP バインディングの操作の**スタイル**を **rpc** にできます。

   次に、追加のエレメントを使用して、SOAP 本体に配置されるデータをラップします。

   ラップされるメッセージが要求 の場合、このラッパー・エレメントには操作名と同じ名前が付けられます。この操作名は、バインディングの soap:body エレメントで定義されているオプションのネーム・スペース属性、またはこの属性が存在しない場合は、wsdl 定義の **targetNamespace** と同じ値を持つ**ネーム・スペース**を持ちます。

   ラップされるメッセージが要求ではなく、ラッパーが応答 メッセージに対応している場合は、その名前は、操作名に「Response」という語を付加して形成され、上と同様にしてネーム・スペースが形成されます。各メッセージ・パーツ

(パラメーター) はラッパーの下に表示され、呼び出しの対応するパラメーターと同一の名前のアクセサーによって表されます。パーツは呼び出しのパラメーターと同じ順序で配置されます。

WSDL スニペット:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
  targetNamespace="http://www.example.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.example.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<message name="SayHelloRequest">
<part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
<part name="greeting" type="xsd:string"/>
</message>
<portType name="Hello_PortType">
<operation name="sayHello">
<input message="tns:SayHelloRequest"/>
<output message="tns:SayHelloResponse"/>
</operation>
</portType>
<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
...
</binding>
...
```

Web サービスに渡される階層属性を作成するためのスクリプト (*request*: *SayHelloRequest*) は以下のとおりです。

```
var wrapper = work.createElementNS("http://www.example.com/HelloService.wsdl" ,"ns:sayHello");
var message = work.createElement("firstName");
message.appendChild(work.createTextNode("My Text Value"));
wrapper.appendChild(message);
work.setAttribute(wrapper);
```

また SOAP 要求は以下のようになります。

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" >
   <soap:Header/>
   <soap:Body>
      <ns:sayHello xmlns:ns="http://www.example.com/HelloService.wsdl">
         <firstName>My Text Value </firstName>
      </ns:sayHello>
   </soap:Body>
</soap:Envelope>
```

(例えば、Axis2WSServerConnctor から) Web サービスの応答を収集する場合、スクリプトは以下のようになります。

```
var wrapper = work.createElementNS("http://www.example.com/wsdl/HelloService.wsdl", "ns:sayHelloResponse");
var message = work.createElement("greeting");
message.appendChild(work.createTextNode("Returned Value"));
wrapper.appendChild(message);
work.setAttribute(wrapper);
```

また SOAP 応答は以下のようになります。

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" >
   <soap:Header/>
   <soap:Body>
      <ns:sayHelloResponse  xmlns:ns="http://www.example.com/wsdl/HelloService.wsdl">
         <greeting>Returned Value </greeting>
      </ns:sayHelloResponse>
   </soap:Body>
</soap:Envelope>
```

2. 操作のスタイルを**文書**にできます。

追加のラッパーが存在していないため、メッセージ・パーツは直接 SOAP の Body エレメントの下に現れます。必要なことは、メッセージ・パーツの名前と

同じ名前を持つ属性と、渡す必要のあるデータ (この例ではストリング) を保持する、メッセージ・パーツの子エレメントを作成することだけです。

同一の WSDL スニペットの場合:

```
...
<message name="SayHelloRequest">
<part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
<part name="greeting" type="xsd:string"/>
</message>
<portType name="Hello_PortType">
<operation name="sayHello">
<input message="tns:SayHelloRequest"/>
<output message="tns:SayHelloResponse"/>
</operation>
</portType>
<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
...
</binding>
...
```

要求属性を作成するためのスクリプトは以下のようになります。

```
var message = work.createElement("firstName");
message.appendChild(work.createTextNode("My Text Value"));
work.setAttribute(message);
```

また SOAP 要求は以下のようになります。

```
<soapenv:Envelope xmlns:soapenv=" http://www.w3.org/2003/05/soap-envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <firstName>My Text Value </firstName>
   </soapenv:Body>
</soapenv:Envelope>
```

(例えば、Axis2WSServerConnctor から) Web サービスの応答を収集する場合、スクリプトは以下のようになります。

```
var message = work.createElement("greeting");
message.appendChild(work.createTextNode("Returned Value"));
work.setAttribute(message);
```

また SOAP 応答は以下のようになります。

```
<soapenv:Envelope xmlns:soapenv=" http://www.w3.org/2003/05/soap-envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <greeting>Returned Value </greeting>
   </soapenv:Body>
</soapenv:Envelope>
```

追加情報については、http://www.w3.org/TR/wsdl/ のセクション 3.5 を参照してください。

提供された WSDL ファイルが WSDL 2.0 標準に準拠している場合は、以下のようになります。

状態は文書操作スタイルに類似しています。つまり、追加のラッパーは不要です。

属性の階層は、WSDL ファイルの *types* ブロックによって定義されているメッセージの構造に準拠している必要があります。

SOAP ヘッダー情報については、33 ページの『SOAP ヘッダー』を参照してください。

Axis2WSClientFC は、詳細ログ・モードの場合、受信した SOAP 応答の HTTP ヘッダーをログに記録することができます。これにより、この情報に簡単にアクセスすることができます。また、「HTTP」属性をこの関数コンポーネントの入力マップでマップして、SOAP 要求の特定のヘッダーを設定することができます。HTTP ヘッダーについて詳しくは、 142 ページの『HTTP サーバー・コネクター』および 29 ページの『Axis2 Web サービス・サーバー・コネクター』を参照してください。

構成

以下に示すパラメーターを使用して、Axis2WSClientFC を構成することができます。

Axis2 WS クライアント関数コンポーネントには以下のパラメーターがあります。

WSDL URL
Web サービス呼び出しに使用される WSDL ファイルのロケーション。

サービス
呼び出されるサービスの名前 (WSDL ファイルに書き込まれています)。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望するサービス名を選択することができるドロップダウン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初に WSDL ファイルを指定する必要があります。これは、このスクリプトが、そのファイル内のサービスを表示するためです。

操作
呼び出される SOAP 操作の名前。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望する操作を選択することができるドロップダウン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初にサービスを指定する必要があります。これは、このスクリプトの目的がサービスに関連付けられている操作を表示することであるためです。

エンドポイント
呼び出される Web サービスに対応するエンドポイントの名前 (WSDL ファイルに書き込まれています)。このパラメーターには、スクリプトが割り当てられたボタンがあり、このスクリプトによって、希望するエンドポイント名を選択することができるドロップダウン・ボックスが表示されます。ただし、このスクリプトが機能するためには、最初にサービスを指定する必要があります。これは、このスクリプトが、サービスに関連付けられているエンドポイントを表示するためです。

ユーザー名
HTTP 基本認証呼び出しに使用されるユーザー名。

パスワード
HTTP 基本認証呼び出しに使用されるパスワード。

接続タイムアウト
接続タイムアウトをミリ秒で指定します。 デフォルトは 60000 (1 分) です。このコネクターを構成する場合は、最初に WSDL ファイルを構成し、その後にサービスを選択します。次に、SOAP 操作とエンドポイントを選択します。

関連情報

TDI_install_dir/examples/axis2_web_services の例。

Axis EasyInvoke Soap WS 関数コンポーネント

Axis EasyInvokeSoapWS 関数コンポーネント (FC) は、IBM Security Directory Integrator Web Services Suite に含まれます。以下に示す情報を使用して、Axis EasyInvokeSoapWS 関数コンポーネント (FC) について詳しく知ることができます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

これは「簡易版」の Web サービス呼び出しコンポーネントです。専用の構成画面を持つスタンドアロン関数コンポーネントですが、内部的には以下の 3 つの関数コンポーネントと同じ関数を実行します。AxisJavaToSoap、InvokeSoapWS、および AxisSoapToJava。

提供される機能は、AssemblyLine 内でこれら 3 つの関数コンポーネントを連結して構成した場合と同じです。この関数コンポーネントを使用すると、カスタム処理をフックする可能性がなくなります。つまり、Axis によって提供される処理とバインディングに従いますが、セットアップと使用が単純になります。

この関数コンポーネントで処理できないシナリオがあることに注意してください。代わりに前述の AxisJavaToSoap、 InvokeSoapWS、および AxisSoapToJava の組み合わせを使用する必要があります。処理できないシナリオは以下のとおりです。

- SOAP ヘッダーの処理が必要
- シリアライゼーション/デシリアライゼーションの問題 (シリアライゼーション/デシリアライゼーションの問題、およびこれらの問題を処理するカスタム・シリアライザー/デシリアライザーの登録方法について詳しくは、533 ページの『シリアライゼーション/デシリアライゼーションの問題』を参照してください)

構成

ここに記載されているパラメーターを使用することで、Axis EasyInvokeSoapWS 関数コンポーネントを構成できるようになります。

パラメーター

WSDL URL
サービスを記述している WSDL 文書の URL。これは必須パラメーターです。指定しないと、初期化時に例外がスローされます。

SOAP 操作
WSDL ファイルに記述されている SOAP 操作の名前です。これは必須パラメーターです。指定しないと、初期化時に例外がスローされます。

ログイン・ユーザー名
HTTP 基本認証を使用してサーバーに送信されるログイン・ユーザー名。サーバーで許可が必要な場合は、この値と次の値 (ログイン・パスワード) を

使用してクライアントが認証されます。エンコードとして Base64 が使用され、エンコードが InvokeSoapWS 関数コンポーネント内部で実行されます。

ログイン・パスワード
サーバーに送信されるログイン・パスワード。サーバーで許可が必要な場合は、この値と前の値 (ログイン・ユーザー名) を使用してクライアントが認証されます。

複素数タイプ
このパラメーターはオプションです。これを指定する場合は、完全修飾された Java クラス名 (パッケージ名を含む) のリストを指定する必要があります。このリストでは、各エレメント (Java クラス) を 1 つ以上の記号 (コンマ、セミコロン、スペース、復帰、および改行) で区切ります。

操作パラメーター
SOAP 操作のパラメーター名を順番に並べたリスト。SOAP 操作にパラメーターが必要で、関数コンポーネントに*項目* を渡す場合は、このパラメーターが必須です。SOAP 操作にパラメーターが不要で、関数コンポーネントに*項目* を渡す場合は、このパラメーターを空にしなければなりません。*項目* を渡さない場合、このパラメーターの内容は処理に影響を与えません。

- このパラメーターを指定する場合は、このパラメーターに属性名のリストを含める必要があります。このリストでは、各属性名を 1 つ以上の記号 (コンマ、セミコロン、スペース、復帰、または改行) で区切ります。
- このリストに含める属性名は、関数コンポーネントに渡す*項目* に存在している必要があります。属性が 1 つでも欠落していると、例外がスローされます。
- このリストに含まれる項目の順序により、属性値がパラメーターとして SOAP 操作に渡される順序が決まります。

タイムアウト
データを取り出すための時間 (秒単位。小数部でミリ秒を指定することもできます。0.001 未満であるか、または明示的に 0 に設定されている場合は、永久に待機します)。デフォルト値は 60 です。

詳細ログ
このボックスをチェックすると、追加のログ・メッセージが生成されます。

コメント
ここにはユーザー独自のコメントを入力します。

セキュリティーと認証

以下に示す情報を使用して、認証の実行方法について知ることができます。

AxisEasyInvokeSoapWS 関数コンポーネントは HTTP 基本認証を使用します。この関数コンポーネントの「ユーザー名」パラメーターおよび「パスワード」パラメーターが提供された場合は、この情報がサーバーに送信されます。サーバーは、認証を要求している場合は、ユーザー名とパスワードに対応するこれら 2 つのパラメーターを受け取ります。

関数コンポーネントの入力

以下に示す項目を、AxisEasyInvokeSoapWS 関数コンポーネントの入力として使用することができます。

Web サービスの入力データを表す*項目* または *Java* オブジェクト。これら以外のものを渡すと、例外がスローされます。

関数コンポーネントの出力

以下に示す項目を、AxisEasyInvokeSoapWS 関数コンポーネントの出力として使用することができます。

Web サービスの出力データを表す*項目* または *Java* オブジェクト。

関数コンポーネントの使用

この関数コンポーネント (FC) は、HTTP 上の SOAP Web サービスを呼び出すための比較的簡単な方法を提供します。以下に示す情報を参照すると、この通信について詳しく知ることができます。

コミュニケーション・フローを以下に示します。

Web サービス・クライアント <–> AxisEasyInvokeSoapWS 関数コンポーネント
<–> org.apache.axis.client.Call <–> Web サービス

使用する際には、以下に注意する必要があります。

- この関数コンポーネントに *Java Object* の配列 (*Object[]*) を渡した場合、関数コンポーネントは、その配列に Java オブジェクトが格納されている順序で配列から各オブジェクトを取り出して SOAP 操作に渡します。この関数コンポーネントに *項目* を渡した場合、SOAP 操作に渡されるパラメーターの順序と値は、関数コンポーネントの「SOAP 操作」パラメーターの値によって決まります。
- この関数コンポーネントは、文書スタイルの SOAP メッセージおよび RPC スタイルの SOAP メッセージの生成と解析のほか、SOAP フォールト・メッセージの解析を実行できます。生成されるメッセージのスタイルは、関数コンポーネントの「WSDL URL」パラメーターで指定された WSDL によって決まります。
- この関数コンポーネントは、SOAP メッセージを生成および解析するときのエンコードとして、「リテラル」エンコード方式および SOAP セクション 5 エンコード方式の両方を使用できます。生成されるメッセージのエンコードは、関数コンポーネントの「WSDL URL」パラメーターで指定された WSDL によって決まります。
- この関数コンポーネントは、WSDL 文書の <types> セクションで定義されている複素数タイプの値を含む SOAP メッセージを生成および解析できます。これを行うために、この関数コンポーネントは、(1)「複素数タイプ」関数コンポーネント・パラメーターに、SOAP 操作へのパラメーターとして使用される複素数タイプをインプリメントするすべての Java クラスの名前が含まれていること、(2) それらの Java クラスのクラス・ファイルが IBM Security Directory Integrator の Java クラス・パスに配置されていることを要求します。
- この関数コンポーネントに*項目* オブジェクトを渡した場合、サーバーから戻された SOAP 応答メッセージが SOAP フォールト・メッセージ以外であり、SOAP 操作の出力パラメーターが 1 つのみであると、この関数コンポーネントは、

return 属性のパラメーターを戻します。(Axis 1.1 仕様に基づき、SOAP 操作の出力パラメーターが 1 つのみの場合はこのパラメーターが操作の戻り値として解釈されます。SOAP 操作に複数の出力パラメーターがある場合は、操作の戻りタイプが void であると解釈されます。)

- この関数コンポーネントに*項目* オブジェクトを渡した場合、サーバーから戻された SOAP 応答メッセージが SOAP フォールト・メッセージ以外であり、かつ SOAP 操作の出力パラメーターが複数存在すると、この関数コンポーネントは、SOAP 操作の出力パラメーターの名前と一致する*項目*属性の出力パラメーターを戻します。
- この関数コンポーネントに SOAP 操作の入力パラメーターと *Object[]* が戻された場合、サーバーから戻された SOAP 応答メッセージが SOAP フォールト・メッセージ以外であると、*Object[]* タイプの結果が生成されます。結果の 1 番目のエレメントは SOAP 操作の戻り値 (操作が void の場合は NULL) であり、その他のパラメーターは操作の出力パラメーターです。

  Object[] –> AxisEasyInvokeSoapWS 関数コンポーネント –> Object[]

  または

  項目 –> AxisEasyInvokeSoapWS 関数コンポーネント –> 項目
- この関数コンポーネントには、ユーザー名とパスワードのパラメーターがあります。これらのパラメーターが指定されている場合、関数コンポーネントにより基本許可ヘッダーが設定され、サーバーに送信されます。指定されたユーザー名とパスワードがエンコードされます。使用されるエンコード方式は base64 であり、InvokeSoapWS 関数コンポーネント内部で実行されます。
- タイムアウト・フィールドは、呼び出しが応答を待つ時間を指定します。タイムアウトになると例外がスローされ、Web サービスが時間内に応答しなかったことをユーザーに通知します。 Axis 1.4 で使用されるデフォルトのタイムアウト値は 60 秒です。「timeOut」パラメーターは、小数点以下 3 桁に書式設定された double 型の値を受け取ります。単位は秒です。

関連情報

537 ページの『InvokeSoap WS 関数コンポーネント』

複素数タイプ・ジェネレーター関数コンポーネント

複素数タイプ・ジェネレーター関数コンポーネントは、IBM Security Directory Integrator Web Services Suite に含まれます。ここに記載されている情報とリンクを参照して、複素数タイプ・ジェネレーター関数コンポーネントについて詳しく理解することができます。

注: このコンポーネントで使用されている Axis ライブラリーの制限により、WSDL (http://www.w3.org/TR/wsdl) バージョン 1.1 の文書のみがサポートされます。また、サポートされるメッセージ交換プロトコルは SOAP 1.1 です。

この関数コンポーネントは、WSDL の内部スキーマまたは WSDL が参照しているスキーマで定義された複合データ・タイプをインプリメントする Java クラス・ファイルを含む JAR ファイルを生成するために使用します。生成した JAR ファイル

は、それらの複合データ・タイプを含む SOAP メッセージをシリアライズおよび解析するために、他の Web サービス関数コンポーネントが利用できます。

この関数コンポーネントは、例えば AssemblyLine に組み込んで「実行する」ことが想定されていないため、注意してください。この関数コンポーネントは、次のようにして使用します。

1. AssemblyLine 内に配置する。
2. パラメーターに情報を入力する。
3. 「**複素数タイプの生成**」ボタンをクリックして JAR ファイルを作成する。

必要な JAR ファイルを生成した後、この関数コンポーネントは使用不可にするか、AssemblyLine から完全に削除することができます。この関数コンポーネントは、いかなるランタイム機能も提供していません。

構成

ここに記載されているパラメーターを使用して、複素数タイプ・ジェネレーター関数コンポーネントを構成することができます。

パラメーター

WSDL URL
このパラメーターの値には、WSDL ファイルの位置を示す有効な URL ストリングまたはファイル・システム・パス (絶対パスまたは相対パス) のいずれかを指定します。

WSDL2Java オプション
このパラメーターの値には、Axis WSDL2Java ユーティリティーに対するオプションを、コマンド行の場合と同じようなリストとして指定します。関数コンポーネントは、WSDL から Java ソース・ファイルを生成するときに、このオプション・リストを WSDL2Java ユーティリティーに渡します。これらのオプションは、WSDL2Java ユーティリティーのデフォルトの動作を変更するために使用できます。

JAR ファイル名
このパラメーターの値には、作成する JAR ファイルの名前 (絶対パスまたは相対パス) を指定します。

JDK パス
Java Development Kit (JDK) のインストールへのパスです。空のままにすると、ユーティリティーは Java コンパイラー `javac` ツールおよび `jar` ツールがシステムの実行可能ファイル・パスにあると想定します。

注: この関数コンポーネントのインプリメンテーションには、バージョン 1.4 以上の JDK が必要です。

Java ソース・ファイルの生成
このボックスにチェック・マークを付けた場合 (デフォルト)、関数コンポーネント・ユーティリティーは、指定された WSDL から Java ソース・ファイルを生成します。チェックを外した場合、関数コンポーネント・ユーティリティーは Java ソース・ファイルの生成をスキップし、コンパイルと JAR 作成のみを実行します。このパラメーターを false に設定する (つまり、チ

ェックを外す) 方法は、複素数タイプのインプリメンテーションを自分で記述する場合や、自動生成された Java ソース・ファイルを変更する場合に役立ちます (このパラメーターを true に設定すると、手動で編集/作成した Java ソース・ファイルがすべて上書きされます)。

関数コンポーネントの入出力

ここに記載されている情報を参照して、複素数タイプ・ジェネレーター関数コンポーネントの入出力について詳しく理解することができます。

この関数コンポーネントの JAR 作成ユーティリティーを実行するには、「**複素数タイプの生成**」ボタンをクリックします。

- Java **ソース**・ファイルの出力先、およびその後の読み取り元は「*<installation_folder>*/temp/ComplexTypesJavaFiles」です。このフォルダーが存在しない場合は、自動的に作成されます。
- Java クラス・ファイルの出力先、およびその後の読み取り元は「*<installation_folder>*/temp/ComplexTypesClassFiles」です。このフォルダーが存在しない場合は、自動的に作成されます。

注: どの出力ファイル (Java ソース・ファイル、Java クラス・ファイル、JAR ファイル) を作成する場合も、以前に生成されたファイルは削除されます。

トラブルシューティング

ComplexTypesGenerator 関数コンポーネントからエラー・メッセージ・ボックスが表示された場合に、発生したエラーに関してさらに情報が必要なときは、提供された手順を実行します。

1. *log4j.logger.com.ibm.di.admin* ロガー (*<installation_directory>*/log4j.properties にある) のログ・レベルを DEBUG に変更します。例えば、*log4j.logger.com.ibm.di.admin=WARN* という行を、*log4j.logger.com.ibm.di.admin=DEBUG* に変更します。
2. 構成エディターを再始動します。
3. ComplexTypesGen ユーティリティーを再び実行します。

デルタ関数コンポーネント

通常のコネクター・モード以外のデルタ・サービスに対して、ルックアップ・モード、追加モード、削除モードでデルタ関数コンポーネントを操作することができます。

デルタ関数コンポーネントでは、デルタ機能を AssemblyLine 内の任意の場所で実行できます。この方法により、デルタの変更を計算してそれらをデルタ・ストアに適用する前に、入力ソースから読み込まれた項目を変更できます。

構成

ここに記載されているパラメーターを使用することで、デルタ関数コンポーネントを構成できるようになります。

固有属性名

特定のデータ・ソースで固有値を保持する属性の名前です。「**重複デルタ鍵を許可**」が有効になっている場合を除いて、重複鍵を持つデータ・ソースをデルタ関数の対象とすることはできません。

デルタ・ストア

以降の実行との差を検出できるようにするため、このコネクターの前回の実行からのデルタ情報を保持するシステム・ストア内のテーブルです。 このパラメーターが空の場合、「AssemblyLines」リテラル・ストリングから作成されたデフォルトの名前、AssemblyLine 名、およびコンポーネント名が使用されます (例えば、「`AssemblyLines_AL1_DeltaFunc`」)。

削除済み項目の読み取り

これをチェックした場合、イテレーターが繰り返しを完了したとき、つまり入力を終了したときに、AssemblyLine は削除された項目を AssemblyLine の実行に投入します。演算コードは、この項目が入力ソースで削除されたことを示します。「削除済み項目の除去」フラグも使用可能にしない限り、削除タグが付いた項目がデルタ・ストアから削除されないことに注意してください。

削除済み項目の除去

これをチェックした場合、入力ソースから削除された項目がデルタ・ストアから削除され、以降の実行で再び検出されることはなくなります。

未変更項目を戻す

これをチェックした場合、この実行で変更されなかった項目があれば、AssemblyLine に投入されます。

コミット

入力全体について繰り返した結果、いつデルタ・ストアへの変更をコミットするかを選択します。選択項目には以下のものがあります。

- 各データベース操作後
- AL サイクル終了時
- コネクターのクローズ時
- 自動コミットなし

デフォルトは、「**各データベース操作後**」です。

行のロック

デルタ・ストアへの接続のトランザクション分離レベルを選択します。詳しくは、「*Directory Integrator の構成*」の『イテレーター・モードのデルタ機能 (Delta feature for Iterator mode)』セクションの『行のロック (Row Locking)』サブセクションを参照してください。指定できる値は、以下のとおりです。

- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

デフォルトは、**READ_COMMITTED** です。

高速アルゴリズム

これをチェックした場合、より多くのメモリー使用量を消費しても、高速アルゴリズムを使用して変更を計算するように AssemblyLine が設定されます。基本的に、変更されていない項目をデルタ・ストアへ書き込むことはできません。その代わり、「**削除済み項目の読み取り**」が *true* に設定されている場合、メモリーにキーが記録されます。

重複デルタ鍵を許可

AssemblyLines の長時間実行中に、変更ログ/変更検出コネクターのデルタ機能を使用可能にしていると、1 つの項目を 2 回以上変更することがあります。これらを変更すると、項目の受信が 2 回目になり、重複デルタ鍵の例外がスローされる原因となります。このパラメーターにチェック・マークを付けた場合、重複鍵属性を持つ項目 (「**固有属性名**」パラメーターで指定) をデルタが有効なイテレーター・コネクターで処理できるようになります。

属性リスト

属性のコンマ区切りのリストで、変更の処理中にどの属性を検出または無視するかを指定します。リスト表示された属性の変更は、それらを無視または検出するかどうかを指定する「**変更検出モード**」パラメーターの影響を受けます。このパラメーターについて詳しくは、「*Directory Integrator の構成*」の『イテレーター・モードのデルタ機能 (Delta feature for Iterator mode)』セクションの『特定の属性でのみ変更を検出または無視 (Detect or ignore changes only in specific attributes)』サブセクションを参照してください。

変更検出モード

属性を検出または無視する変更を指定します。このパラメーターについて詳しくは、「*Directory Integrator の構成*」の『イテレーター・モードのデルタ機能 (Delta feature for Iterator mode)』セクションの『特定の属性でのみ変更を検出または無視 (Detect or ignore changes only in specific attributes)』サブセクションを参照してください。

関数コンポーネントの使用

この関数コンポーネントを使用して、AssemblyLine 内の任意の場所でデルタ検出を実行したり、ロジックを適用したりすることができます。

デルタ関数コンポーネントには、イテレーター・モードにあるコネクターで使用可能なデルタ・タブと同じ機能が備わっています (「*Directory Integrator の構成*」の『デルタ』を参照してください)。

デルタ関数コンポーネントは関数コンポーネントであるため、実行には項目が必要です。したがって、入力ソースがデータの終わりに到達すると、デルタ・コンポーネントには処理するものがなくなります。そのため、「**削除済み項目の読み取り**」パラメーターを選択すると、コンポーネントに空のダミーの項目がいくつか提供される場合のみ、削除された項目がデルタ関数コンポーネントによって戻されます。これらの項目は、削除された項目を戻し始めるように関数コンポーネントに通知します。

デルタが使用可能なイテレーター・コネクターと同様、デルタ関数コンポーネントには、Assembly Line 実行の最後にデルタ統計が表示されます (例えば、`"Add:3, Modify:1, CallReply:5, Skip:2, Nochange:2"`)。

例

ここに記載されている例を使用して、デルタ関数コンポーネントについて詳しく理解することができます。

以下の例では、「**削除済み項目の読み取り**」パラメーターが使用可能な場合に、デルタ関数コンポーネントを使用して入力ソースと IBM Security Directory Integrator デルタ・ストアを同期する方法を示します。入力ソースから削除された項目は、*delete* デルタ操作のマークが付けられ、AssemblyLine へ戻されます。

実行の手順は以下のとおりです。

1. 入力データ・ソースを繰り返すイテレーター・モードにあるコネクターを追加します。
2. スクリプト・コネクターを最後のコネクターとして「フィード」セクションに追加します。
3. 次のように、このコードを getNextEntry() メソッドに追加します。

```
r = task.getResult();

if (r != null){
 if (r.size() > 0){
  entry = system.newEntry();
  result.setStatus(1);        // OK
 } else {
  result.setStatus(0);        // end of input
 }
    }
```

4. 作業項目を変更するカスタム・ロジックまたはコンポーネントを、いくつか追加します。
5. デルタ関数コンポーネントを追加して、それが削除済み項目について繰り返すように構成します。
6. AssemblyLine を開始します。
7. 入力データ・ソースの一部の項目を削除します。
8. AssemblyLine を再度開始して、*delete* というタグが付けられた削除済み項目を受け取ります。

注: 最後に、デルタ関数コンポーネントは空のダミー項目を戻します。この項目をチェックすると、スクリプト・コネクターの getNextEntry() メソッドを使用して、どの時点でダミー項目が戻るのを停止するかを判別できます。削除済み項目がなく、「**削除済み項目の読み取り**」が使用可能な場合は、空の項目も戻されます。

リモート・コマンド行関数コンポーネント

リモート・コマンド行関数コンポーネント (リモート CLFC) では、リモート・マシンに対してコマンド行システム呼び出しを実行できます。ここに記載されている情報を使用することで、リモート・コマンド行関数コンポーネントについて詳しく知ることができます。

設計とインプリメンテーションでは RXA ツールキット v2.2 を使用してリモート・マシンに接続し、コマンドを実行して結果を戻します。戻された出力を解析して、一度に 1 つずつ値をコンシュームし、実行したコマンドの問題を検出できます。

リモート・コマンド行関数コンポーネントは、RSH、REXEC、SSH、AS400、または Windows プロトコルのいずれかを使用してリモート・マシンに接続できます。使用するプロトコルを選択できますが、デフォルト値「ANY」のままにしておくと、関数コンポーネントは、接続が正常に完了するまで、使用可能なプロトコルを 1 つずつ使用してリモート・マシンへの接続を試行します。

この関数コンポーネントによって使用される RXA ライブラリーは、対話式の SSH セッションのみをサポートしています。非対話式の SSH セッションはサポートされて**いません**。

リモート・マシンの情報 (ホスト名、ユーザー名、パスワードなど) を指定する必要があります。SSH プロトコルを使用して接続が確立される場合は、認証にパスワードを使用する代わりに、鍵ストア名とパスフレーズを指定できます。

注: SSH 接続は一般に Linux/UNIX ホストに関連しています。ただし、Cygwin および Cygwin *openssh* パッケージを Windows ターゲット・マシンにインストールすることで、Windows ターゲットでも SSH プロトコルを使用できます。

構成

ここに記載されているパラメーターを使用して、リモート・コマンド行関数コンポーネントを構成することができます。

ターゲット・マシンのホスト名
ターゲット・マシンのホスト名 (アドレス)。このパラメーターは必須です。

リモート・ユーザー
ターゲット・マシンの管理特権が付与されているユーザーの名前。

パスワード
ターゲット・マシンのユーザー (「**リモート・ユーザー**」で指定するユーザー) のパスワード。鍵ストアを使用した SSH 接続および RSH 接続の場合は、このパラメーターはオプションです。

鍵ファイル・パス
OpenSSH 秘密鍵ファイルへの絶対パス。このパラメーターはオプションであり、SSH 接続の場合にのみ使用されます。

パスフレーズ (パスワード)
上記の「**鍵ストア・パス**」パラメーターにより指定される鍵ストアで秘密鍵を保護するパスフレーズ (パスワード)。

接続プロトコル
「ANY」、「SSH」、「RSH」、「REXEC」、「AS400」、および「WIN」から選択します。これにより、リモート・マシンへの接続時に使用するプロトコルが指定されます。詳しくは、561 ページの『関数コンポーネントの使用』を参照してください。

ポート ターゲット・マシンへの接続に使用されるポート。

コマンド
ターゲット・マシンで実行するコマンド。出力属性「command.line」が指定されている場合、これは指定変更されます。これは、「command.line」属性が出力マップで提供されていない場合は、必須パラメーターです。

標準入力ソース・ファイル (ローカル)
指定されたコマンドへの標準入力として使用するローカル・システム上のファイルのパス。このパラメーターはオプションです。

標準入力の宛先ディレクトリー (リモート)
「**標準入力ソース・ファイル (ローカル)**」で指定した標準入力ファイルをコピーするターゲット上の既存の宛先ディレクトリーのパス。「**標準入力ソース・ファイル (ローカル)**」に値を指定した場合に宛先として値を指定しないと、リモート・マシンに一時ディレクトリーがランダムに作成されます。ファイルは一時的にコピーされる点に注意してください。コマンドの実行が完了すると、リモート・マシン上のコピーは削除されます。

標準入力ソース・ファイルをターゲット・システムの文字セットに変換する
チェック・マークを付けると、標準入力ソース・ファイルはターゲット・システムの文字セットに変換されます。チェック・マークを外した場合は、ファイルの現行のエンコードが維持されます。

タイムアウト (ミリ秒)
適切な CPU タイムアウト期間 (ミリ秒)。指定された期間内に操作が完了しないと、操作は取り消されます。このパラメーターはオプションです。指定されていない場合、または値 0 (ゼロ) を指定した場合は、無制限、つまり処理時間制限はなくなります。

注: タイムアウトは、リモート・コマンド行関数コンポーネント・プロセスの CPU クロック時間であり、プロセス開始後の実際の経過時間ではありません。処理集中型でないコマンドは、処理時間制限に達していない場合は指定の時間内にタイムアウトすることはありません。

初期接続タイムアウト (ミリ秒)
ターゲット・システムとの初期接続のタイムアウト期間を定義するオプションの「リモート CLFC」パラメーター。これは AS400 ターゲットには影響しません。

AS400 への SSL を有効にする
このパラメーターは、AS400 接続で SSL 接続を行うかどうかを決定します。チェック・マークを付けた場合、AS400 ターゲットとの SSL 接続が試行されます (AS400 ターゲット・システムに SSL がインストールおよび構成されている必要があります)。デフォルトではチェックは外されています。

AS400 プロキシー・サーバー名
このパラメーターは、必要であれば AS400 プロキシー・サーバーを定義します。

AS400 プログラムを実行する
AS400 接続に使用するコマンド実行のタイプを定義するオプションの「リモート CLFC」パラメーター。AS400 プログラムの拡張子は .PGM です。これらの AS400 プログラムの引数は、項目属性「**command.args**」を使用して指定できます。

デフォルトではチェックは外されています。

RXA 内部ロギングを有効にする
これを有効にすると、RXA 内部ロガーによって AssemblyLine ログ・ファイル内にログ・メッセージを生成できます。

AS400 コマンド行引数の文字エンコード
AS400 コマンド行引数に使用する文字エンコード。指定されない場合は、JVM のデフォルトの文字エンコードが使用されます。この構成パラメーターはオプションで、「**AS400 プログラムを実行する**」パラメーターが設定されている場合にのみ適用されます。この値をターゲット AS400 ボックスの適切なエンコードに設定しないと、リモート・マシン上の JVM のエンコードが、AS400 プログラムが実行される AS400 マシンのデフォルト・エンコードに一致しない場合は、コマンド行パラメーター・ストリングが破損する可能性があります。

詳細ログ
これを有効にすると、デバッグ・ログ・メッセージが生成されます。

関数コンポーネントの入力

リモート・コマンド行関数コンポーネントの「構成」画面で構成するパラメーターの一部は、入力マップの作業項目からマップされる属性としても指定できます。何らかの値が指定されている属性が存在する場合は、ここにリストされている「構成」画面のパラメーターよりも優先されます。

command.line
ターゲット・マシンで実行するコマンド。この属性は、出力マップで定義する必要があり、「構成」タブで定義されている「**コマンド**」パラメーターと置き換わります。

command.args
それぞれの値がコマンド行引数である多値属性。AS400 プログラムを実行する場合は必須です。

command.args.delim
この属性は、コマンド/プログラム引数の区切り文字を指定します。指定されていない場合のデフォルトは、単一の空白文字です。

stdin.source
java.io.String タイプの属性であり、指定されたコマンドの標準入力として使用するローカル・マシン上のファイルのパスを表します。

stdin.destination
java.io.String タイプの属性であり、リモート・マシンに保管する転送ファイルのパスを表します。

つまり、*command.line* という属性が入力項目オブジェクトに指定されている場合、構成エディターで入力されたすべてのコマンドは無視されます。これにより、AssemblyLine 内の他のコンポーネントによりリモート・コマンド行関数コンポーネントを繰り返し呼び出し、異なるコマンドを実行することができます。

関数コンポーネントの出力

ここに記載されている属性を使用して出力を処理することができます。

リモート・コマンド行関数コンポーネントは、前述のように *command.line* 属性または「**コマンド**」構成パラメーターで指定されているコマンドの実行が完了すると、属性マッピングで以下の属性を使用可能にします。

command.returnCode (int)
リモート・コマンド実行結果の戻りコード。

command.error (String)
コマンド実行時に生成された標準エラー・メッセージ (生成されている場合)。

command.out (String)
コマンド実行時に生成された標準出力メッセージ (生成されている場合)。

関数コンポーネントの使用

ここに記載されている情報とリンクを参照して、リモート・コマンド行関数コンポーネントの使用方法について理解することができます。

他の IBM Security Directory Integrator コンポーネント (コネクターや他の関数コンポーネントなど) が含まれている AssemblyLine 内でリモート・コマンド行関数コンポーネントを使用することができます。正しく機能するようにするには、構成エディターを使用してリモート・コマンド行関数コンポーネントを正しく構成する必要があります。初期化時にリモート・マシンとの接続が確立され、次に `perform()` メソッドが呼び出されると (通常、AssemblyLine 内でこの関数コンポーネントに達した時点)、ターゲットでコマンドが実行されます。

`perform()` メソッドは、完了時に前述の 3 つの出力属性 (*command.returnCode*、*command.error*、および *command.out*) を含む項目オブジェクトを戻します。これにより、AssemblyLine 内の後続のその他の IBM Security Directory Integrator コンポーネントでこれらの属性を使用することができるようになります。

この関数コンポーネントを使用して、メッセージのリストを標準出力 (ディレクトリー・リストなど) で戻すコマンドを実行する場合、*command.out* String オブジェクトから個々の項目を抽出し、一度に 1 つずつ処理するためには、リモート・コマンド行関数コンポーネントを他の IBM Security Directory Integrator コンポーネント (パーサーなど) と組み合わせて使用する必要があります。

ターゲット・システムの構成

ターゲット・マシンは、以下の要件を満たしている必要があります。

Windows ターゲット

WIN プロトコルの使用: Windows XP ターゲットでは、Remote Execution and Access が機能できるようにするため、簡易ファイルの共有を無効にする必要があります。単純ネットワーキングでは、すべてのログインが「ゲスト」として認証されます。ゲスト・ログインでは、Remote Execution and Access が機能するために必要な権限がありません。

簡易ファイルの共有を無効にするには、「Windows エクスプローラ」を開始して「**ツール**」->「**フォルダ オプション**」をクリックします。「**表示**」タブを選択して、設定のリストをスクロールし、「**簡易ファイルの共有を使**

用する」を見つけます。「**簡易ファイルの共有を使用する**」の横のチェックを外し、「**適用**」と「**OK**」をクリックします。

Windows XP には、インターネット接続ファイアウォール (ICF) と呼ばれる標準装備のファイアウォールがあります。デフォルトでは、Windows XP Service Pack 2 以外の Windows XP システムでは ICF は無効になっています。Windows XP Service Pack 2 ではデフォルトで有効になっています。Windows XP ターゲットでいずれかのファイアウォールが有効になっていると、Remote Execution and Access からのアクセスがブロックされます。Service Pack 2 では、Windows ファイアウォール構成の「例外」タブの「ファイルとプリンタの共有」ボックスを選択して、アクセスを許可できます。

Remote Execution and Access がターゲット・マシンでコマンドとスクリプトを実行できるようにするため、ターゲット・マシンではリモート・レジストリー管理が有効になっている必要があります (これはデフォルト構成です)。

Remote Execution and Access が適切に機能するには、デフォルトでは非表示の管理ディスク共有 (C$、D$ など) が必要です。

Cygwin ターゲット

SSH プロトコルの使用: リモート Windows コンピューターへの SSH ログインを使用するには、http://cygwin.com から Cygwin をダウンロードし、アプリケーションのターゲットとなる各 Windows マシンにインストールする必要があります。 Cygwin に関する詳細な資料は、http://cygwin.com から入手可能です。

Cygwin ターゲットで Remote Execution and Access アプリケーションを使用するには、インストールされているデフォルトの Cygwin には含まれていない追加 Cygwin パッケージを最大 2 つまでインストールする必要があります。http://cygwin.com で、Cygwin パッケージの *net* カテゴリーにある openssh をダウンロードしてインストールします。openssh には、Cygwin ターゲットでの SSH ログインをサポートするために必要な ssh デーモンが含まれています。もう 1 つのパッケージである cygrunsrv は、*admin* カテゴリーにあります。このパッケージは、ssh デーモンを Windows サービスとして実行する機能を提供します。ssh デーモンをサービスとして実行しない場合は、このパッケージはオプションです。

MKS ターゲット

MKS ツールキットは、Windows マシンで Cygwin の代わりに使用されます。詳しくは http://www.mkssoftware.com/ を参照してください。Windows のコマンド行から MKS を使用するには、PATH 環境変数に *MKS_Installation*/bin へのパスを追加します。MKS のデフォルトでは、SSH はユーザー名パスワード認証を使用するよう構成されています。パスワード不要認証をセットアップするには、(MKS Toolkit およびほとんどの UNIX システムで使用可能な) ssh-keygen ユーティリティーを使用して公開鍵と秘密鍵のペアを生成し、接続先のマシンにそれらをコピーする必要があります。

(MKS Toolkit のセキュア・シェル・サービス (sshd) などの) セキュア・シェルの OpenSSH バージョンから得られるセキュア・シェル・サービスまたはデーモンに接続する場合は、プロトコル・バージョン 1 の RSA 鍵をホ

ストの ~/.ssh/authorized_keys ファイルに添付し、プロトコル・バージョン 2 の RSA 鍵と DSA 鍵をホストの ~/.ssh/authorized_keys2 ファイルに添付する必要があります。ここで、~/ はリモート・ホスト上のアカウントのホーム・ディレクトリーです。

UNIX および Linux ターゲット

SSH、**RSH**、または **REXEC** プロトコルの使用: この関数コンポーネントが使用する RXA ツールキットには、UNIX マシン用の SSH コードはありません。SSH プロトコルを使用してアクセスするすべてのターゲットで、SSH がインストールされており、有効になっていることを確認してください。OpenSSH 3.71 以上に含まれているセキュリティー拡張機能は、これより前のリリースには含まれていません。

RXA は、リモート・アクセス・プロトコル (rsh、rexec、ssh) がすべて無効になっている UNIX ターゲットには接続できません。

Solaris を除くすべての UNIX 環境では、UNIX 環境のターゲット・シェルとして Bourne シェル (sh) が使用されます。Solaris ターゲットでは、sh で問題が発生したため、代わりに Korn シェル (ksh) が使用されます。

RXA がパスワード認証を使用して Linux およびその他の SSH ターゲットと通信できるようにするには、ターゲット・マシン上のファイルである /etc/ssh/sshd_config ファイルを編集して以下のように設定する必要があります。

```
PasswordAuthentication yes (the default is 'no')
```

この設定変更後、以下のコマンドを使用して SSH デーモンを停止してから再始動します。

```
/etc/init.d/sshd stop
/etc/init.d/sshd start
```

rsh / rexec 接続プロトコルを使用する場合は、IBM Security Directory Integrator サーバーが、特権ユーザー (Unix の場合は root、Windows の場合は管理者特権を持つユーザー) として実行している必要があります。この要件が満たされていない場合、接続は失敗します。rsh および rexec 接続プロトコルでは、ソース・ポートに「高い信頼性」が要求されますが (ポート番号は 1024 未満)、これらのプラットフォームでは、信頼性の高いポート接続の作成は特権ユーザーに制限されています。

ローカル・マシンとターゲットとの間でのパスワード認証または鍵ストアを使用した SSH の構成方法についての詳細は、http://www.openssh.com の該当する OpenSSH 資料を参照してください。

AS400 ターゲット

AS400 ターゲットでは、IBM Toolbox for Java を適切な JRE と共にインストールする必要があります。IBM Toolbox for Java は、JAR ファイルが *TDI_install_dir*/jars/3rdParty/IBM ディレクトリーに配置される IBM Security Directory Integrator サーバーにも必要です。コマンドおよびプログラム自体は、iSeries システム上の QSYS ライブラリーの下に配置されている必要があります。

AS400 SSL 接続オプションを有効にする場合は、自己署名証明書に追加の構成が必要になります。この場合、署名証明書を Java Security CA 証明書

ストア (*jre_directory*/lib/security/cacerts) に追加する必要があります。詳しくは、http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html を参照してください。

関連情報

50 ページの『コマンド行コネクター』

z/OS TSO/E コマンド行関数コンポーネント

この関数コンポーネントを使用して、特権 z/OS コマンド (RACF コマンド、ACF2 コマンド、TopSecret コマンドを含む) を IBM Security Directory Integrator から発行できるようにするためのニーズに対応することができます。

注: IBM Security Directory Integrator バージョン 7.2 以降では z/OS オペレーティング・システムはサポートされません。

構成

ここに記載されているパラメーターを使用して、z/OS TSO コマンド行関数コンポーネントを構成することができます。

パラメーター

この関数コンポーネントを正常に機能させるには、z/OS 環境に対して一部のパラメーターを設定する必要があります。

パートナー TP 名
APPC TP プロファイルに指定されているパートナー TP 名を指定します。このパラメーターは必須です。

宛先 LU 名
APPC 構成ファイルに指定されている宛先 LU 名を指定します。これを NULL または空にした場合は、デフォルトとして定義されている LU が使用されます。

ソース LU 名
APPC 構成ファイルに指定されているソース LU 名を指定します。これを NULL または空にした場合は、デフォルトとして定義されている LU が使用されます。

APPC モード
APPC 会話のモードを指定します。これを NULL または空にした場合は、ソース LU で指定されているデフォルト・モードが使用されます。

ユーザー名
会話を行うために ID が使用されるユーザー。

これを NULL または空にした場合、会話の Security_Type は **ATB_SECURITY_SAME** になり、IBM Security Directory Integrator が始動されたときの ID がデフォルト・プロファイルとともに使用されます。それ以外の場合、会話の Security_Type は **ATB_SECURITY_PROGRAM** になり、TSO コマンドは、指定したユーザー ID のもとで、そのユーザーのプロファイルを使用して実行されます。

ユーザー・パスワード

会話を行うために ID が使用されるユーザーのパスワード。「**ユーザー名**」パラメーターを指定した場合にのみ、このパラメーターが使用されます。

これを NULL または空にした場合は、指定したユーザーに、REXX スクリプトがデプロイされたシステムに対する代理権限が付与されている場合に限り、会話が成功します。

詳細ログ

これをチェックすると、追加のログ・メッセージが生成されます。

コメント

ここにはユーザー独自のコメントを入力します。

関数コンポーネントの使用

このコンポーネントは、渡されたコマンドを実行する場合にのみ使用することができます。このコンポーネントがシェル・コマンドを作成したり、実行中のコマンドに関連するビジネス・ロジックを理解したりすることはありません。

z/OS TSO コマンド行関数コンポーネントでは、TSO/E シェル・コマンドを実行できます。

この関数コンポーネントに入力としてコマンド行を渡すと、そのコマンドの実行状況と生成された出力が戻されます。アーキテクチャー上、この関数コンポーネントは、Java 層、USS 共用ライブラリー、および REXX スクリプト・コンポーネントで構成されています。Java 層はコマンドを共用ライブラリーに渡し、共用ライブラリーはそのコマンドを APPC を使用して REXX スクリプトに渡し、REXX スクリプトが TSO/E コマンドを実行して結果を戻します。

高水準の特定のビジネス・ロジックを、この関数コンポーネントの上に構築することができます。例えば、RACF ユーザーを管理するコネクターやアダプターなどです。このコネクターで、正しい RACF コマンド (追加、変更などに対応するコマンド) を構成し、内部でこの関数コンポーネントを使用してこれらのコマンドを実行します。

エラー・フロー

以下に、z/OS TSO コマンド行関数コンポーネントが例外をスローする場合のシナリオを示します。

- z/OS パラメーターの取得中にエラーが発生した場合。
- APPC 会話を割り振ることができなかった場合。
- TSO コマンドを実行できなかった場合。

  次のメッセージがログに記録された場合:「CTGDKB012E TSO コマンドを実行できませんでした。コマンドは戻りコード 26 を返しました。」この場合、以下のような理由が考えられます。

  1. サービス理由: 43

     ATB80043I 呼び出し側プログラムがユーザー ID もパスワードも指定しなかったか、または代理許可検査が失敗したか、あるいはその両方が発生しました。

有効なユーザー名とパスワードを両方とも指定したことを確認してください。

2. サービス理由: 49

   ATB80049I「Local_LU_name」パラメーターで指定された値が、システムの基本 LU の名前でも NOSCHED LU の名前でもありません。

   D APPC,LU,ALL コマンドの出力で「BASE=YES」として定義されている別の LU を探してください。

3. サービス理由: 100

   サービス: ATBRCVW

   ATB80100I VTAM® マクロ APPCCMD から: 1 次エラーの戻りコード: 0018、2 次エラーの戻りコード: 0000、センス・コード: 08640001。

   指定されたユーザー ID には、このコマンドの実行権限、または TDIEXEC REXX スクリプトを含むデータ・セットの実行権限がありません。

- TSO コマンドのコマンド出力を取り出せず、TSO 戻りコードが null の場合。
- 終了時の会話の割り振りの解除中にエラーが発生した場合。
- 無効なパラメーターを使用していずれかの関数が呼び出された場合。

関数コンポーネントの入力

ここに記載されている項目を、z/OS TSO コマンド行関数コンポーネントの入力として使用することができます。

項目 オブジェクトの *command* という名前の属性に、実行するべき TSO/E コマンドを値として設定したもの。

関数コンポーネントの出力

z/OS TSO コマンド行関数コンポーネントの項目オブジェクトとその属性を以下に示します。

commandOutput
TSO/E コマンドを実行した際の出力が入ります。

tsoCommandReturnCode
TSO/E コマンドの戻りコードが入ります。

appcReturnCode
APPC の戻りコードが入ります。

認証

ここに記載されている情報を参照して、z/OS TSO コマンド行関数コンポーネントの認証処理を実行することができます。

APPC 会話は、**Security_Same** および **Security_Progam** という 2 つのモードで実行できます。

Security_Program モードで会話を行えるかどうかは、関数コンポーネントの「**ユーザー名**」パラメーターに NULL 値以外が指定されているかどうかによって決まります。

許可

REXX スクリプトは、TSO コマンドを実際に実行するコンポーネントです。ここに記載されている情報を参照して、このスクリプトについて詳しく理解することができます。

IBM Security Directory Integrator の TSO コマンド実行の許可/不許可は、z/OS TSO コマンド行関数コンポーネントの構成に指定したユーザー ID の特権によって決まります。

REXX スクリプトで TSO コマンドを実行できる機能が悪用される機会を最小限に抑えるには、次のようなオプションのデプロイメント方針を適用することができます。

REXX スクリプト用に特定のデータ・セットを作成します。このデータ・セットには REXX スクリプトのみが含まれ、他のメンバーは含まれません。RACF では、このデータ・セットへのアクセスを、そのスクリプトの実行を許可するユーザーのみに制限します。その上で、それと同じユーザー (複数可) を、z/OS TSO コマンド行コンポーネントの構成に指定する必要があります。

REXX スクリプトへのアクセスを制限するその他のオプションとして、APPC により提供されるアクセスを制限するという方法があります。

- 会話要求が受諾される論理装置 (LU) を制限することができます。例えば、REXX スクリプトにローカル・システムからアクセスする場合は、リモート呼び出しからはアクセス不能な LU 内に TP プロファイルを配置します。
- REXX スクリプトに関連付けられた TP と会話する要求を許可するユーザーの数を制限することができます。例えば、TP にアクセスできる特別なユーザーを作成することが可能です。

必須の仮名ファイル

APPC/MVS 呼び出しでは、実際の呼び出し、特性、変数などに仮名が使用されます。ここに記載されている情報を参照して、仮名ファイルについて詳しく理解することができます。

例えば、APPC/MVS 呼び出しの「return_code」パラメーターを仮名の atb_ok にできます。仮名 atb_ok の整数値は 0 です。APPC/MVS は、さまざまな言語および通信呼び出しに対して仮名とそれに対応する整数値を定義しているヘッダー・ファイル・データ・セット SYS1.SIEAHDR.H で、いくつかの仮名ファイルを提供しています。

ATBPBREX 仮名ファイルは APPC/MVS 呼び出し用に提供されています。この仮名ファイルには、トランザクション・プログラムを REXX で簡単に作成するための REXX 割り当てステートメントが含まれています。TDIEXEC REXX スクリプトは、内部的にこの仮名ファイルを使用しているため、基礎となる z/OS 環境ではヘッダー・ファイル・データ・セット SYS1.SIEAHDR.H で ATBPBREX ファイルが使用可能になっている必要があります。

このファイルが z/OS 環境に存在していない場合は、このファイルを別の場所からコピーするか、次の ATBPBREX ファイルのサンプルを使用して作成することができます。

```
/****START OF SPECIFICATIONS*****************************************/
/*                                                                   */
/*01* MODULE-NAME = ATBPBREX                                         */
/*                                                                   */
/*02* DESCRIPTIVE-NAME = Interface Declaration File for LU 6.2       */
/*                       Protocal Boundary Interface - REXX          */
/*                                                                   */
/*02* COMPONENT = APPC Component (SCACB)                            */
/*                                                                   */
/*01* PROPRIETARY STATEMENT=                                        */
/****PROPRIETARY_STATEMENT********************************************/
/*                                                                   */
/*                                                                   */
/*  LICENSED MATERIALS - PROPERTY OF IBM                             */
/*  THIS EXEC IS "RESTRICTED MATERIALS OF IBM"                       */
/*  5647-A01 (C) COPYRIGHT IBM CORP. 1998                            */
/*                                                                   */
/*  STATUS= HBB6606                                                  */
/*                                                                   */
/*  EXTERNAL CLASSIFICATION:  GUPI                                   */
/*                                                                   */
/*  END OF EXTERNAL CLASSIFICATION                                   */
/*                                                                   */
/****END_OF_PROPRIETARY_STATEMENT**************************************/
/*                                                                   */
/*                                                                   */
/*01* DISCLAIMER =                                                  */
/*                                                                   */
/*   THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    */
/*   COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR     */
/*   ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM */
/*   TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS         */
/*   WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.             */
/*                                                                   */
/*01* FUNCTION = LU 6.2 REXX pseudonym file                         */
/*                                                                   */
/*01* METHOD OF ACCESS:                                             */
/*                                                                   */
/*  If you are using interpreted REXX provided by TSO/E the          */
/*  EXECIO command should be used to read this file.                 */
/*                                                                   */
/*01* DISTRIBUTION LIBRARY: AIEAHDR                                 */
/*                                                                   */
/*01* CHANGE-ACTIVITY:                                              */
/*                                                                   */
/*  FLAG LINEITEM  FMID    DATE  ID  COMMENT                         */
/*  $01=OY54027  HBB4420 920505 PDI8: MAKE PART AVAILABLE IN HBB4420 */
/*  $P1=PKB0817  HBB4430 920729 PDI8: Support of Conversation State  */
/*      constants.                                                   */
/*  $L1=APPCP    HBB6603 960105 PDE6: APPC/MVS PC support            */
/****END OF SPECIFICATIONS*********************************************/
  /* *****************************************************************/
  /*  Conversation State Values                                @P1A*/
  /* *****************************************************************/
     atb_initialize_state           = 2                     /*@P1A*/
     atb_send_state                 = 3                     /*@P1A*/
     atb_receive_state              = 4                     /*@P1A*/
     atb_send_pending_state         = 5                     /*@P1A*/
     atb_confirm_state              = 6                     /*@P1A*/
     atb_confirm_send_state         = 7                     /*@P1A*/
     atb_confirm_deallocate_state   = 8                     /*@P1A*/
     atb_defer_receive_state        = 9                     /*@L1A*/
     atb_defer_deallocate_state     = 10                    /*@L1A*/
     atb_sync_point_state           = 11                    /*@L1A*/
     atb_sync_point_send_state      = 12                    /*@L1A*/
     atb_sync_point_dealloc_state   = 13                    /*@L1A*/

  /* *****************************************************************/
  /*  Conversation Type Values                                       */
  /* *****************************************************************/
     atb_basic_conversation = 0
     atb_mapped_conversation = 1
  /* *****************************************************************/
  /*  Data Received Values                                           */
  /* *****************************************************************/
     atb_no_data_received = 0
     atb_data_received = 1
     atb_complete_data_received = 2
     atb_incomplete_data_received = 3
  /* *****************************************************************/
```

```
/*  Deallocate Type Values                                          */
/* *****************************************************************/
   atb_deallocate_sync_level = 0
   atb_deallocate_flush =1
   atb_deallocate_confirm = 2
   atb_deallocate_abend   = 3
/* *****************************************************************/
/*  Error Direction Values                                          */
/* *****************************************************************/
   atb_receive_error      = 0
   atb_send_error         = 1
/* *****************************************************************/
/*  Fill Values                                                     */
/* *****************************************************************/
   atb_fill_ll            = 0
   atb_fill_buffer        = 1
/* *****************************************************************/
/*  Lock Values                                                     */
/* *****************************************************************/
   atb_locks_short        = 100
   atb_locks_long         = 101
/* *****************************************************************/
/*  Prepare to Receive Type Values                                  */
/* *****************************************************************/
   atb_prep_to_receive_sync_level  = 0
   atb_prep_to_receive_flush       = 1
   atb_prep_to_receive_confirm     = 2
/* *****************************************************************/
/*  Notify Type Values                                              */
/* *****************************************************************/
   atb_notify_type_none   = '00000000'X
   atb_notify_type_ecb    = '00000001'X
/* *****************************************************************/
/*  Request To Send Received Values                                 */
/* *****************************************************************/
   atb_req_to_send_not_received    = 0
   atb_req_to_send_received        = 1
/* *****************************************************************/
/*  Return Code Values                                              */
/* *****************************************************************/
   atb_ok                 = 0
   atb_allocate_failure_no_retry   = 1
   atb_allocate_failure_retry      = 2
   atb_conversation_type_mismatch  = 3
   atb_pip_not_specified_correctly = 5
   atb_security_not_valid = 6
   atb_sync_lvl_not_supported_lu   = 7                        /*@L0A*/
   atb_sync_lvl_not_supported_pgm  = 8
   atb_tpn_not_recognized = 9
   atb_tp_not_available_no_retry   = 10
   atb_tp_not_available_retry      = 11
   atb_deallocated_abend           = 17
   atb_deallocated_normal = 18
   atb_parameter_error    = 19
   atb_product_specific_error      = 20
   atb_program_error_no_trunc      = 21
   atb_program_error_purging       = 22
   atb_program_error_trunc         = 23
   atb_program_parameter_check     = 24
   atb_program_state_check         = 25
   atb_resource_failure_no_retry   = 26
   atb_resource_failure_retry      = 27
   atb_unsuccessful                = 28
   atb_deallocated_abend_svc       = 30
   atb_deallocated_abend_timer     = 31
   atb_svc_error_no_trunc          = 32
   atb_svc_error_purging           = 33
   atb_svc_error_trunc             = 34
   atb_take_backout                 = 100                     /*@L0A*/
   atb_deallocated_abend_bo         = 130                     /*@L0A*/
   atb_deallocated_abend_svc_bo     = 131                     /*@L0A*/
   atb_deallocated_abend_timer_bo   = 132                     /*@L0A*/
   atb_resource_fail_no_retry_bo    = 133                     /*@L0A*/
   atb_resource_failure_retry_bo    = 134                     /*@L0A*/
   atb_deallocated_normal_bo        = 135                     /*@L0A*/
/* *****************************************************************/
/*  Reason Code Values                                               @L0A*/
/* *****************************************************************/
   atb_invalid_vote_read_only      = 1                        /*@L0A*/
   atb_invalid_wait_for_outcome    = 2                        /*@L0A*/
```

```
    atb_invalid_action_if_problems  = 3                     /*@L0A*/
    atb_extract_exit_not_specified  = 4                     /*@L0A*/
    atb_extract_exit_failed         = 5                     /*@L0A*/
    atb_no_active_tp                = 6                     /*@L0A*/
    atb_service_error               = 7                     /*@L0A*/
  /* ****************************************************************/
  /*  Return Control Values                                         */
  /* ****************************************************************/
    atb_when_session_allocated      = 0
    atb_immediate          = 1
    atb_when_conwinner_allocated    = 100
  /* ****************************************************************/
  /*  Security Type Values                                          */
  /* ****************************************************************/
    atb_security_none      = 100
    atb_security_same      = 101
    atb_security_program   = 102
  /* ****************************************************************/
  /*  Send Type Values                                              */
  /* ****************************************************************/
    atb_buffer_data        = 0
    atb_send_and_flush     = 1
    atb_send_and_confirm   = 2
    atb_send_and_prep_to_receive    = 3
    atb_send_and_deallocate         = 4
  /* ****************************************************************/
  /*  Status Received Values                                        */
  /* ****************************************************************/
    atb_no_status_received = 0
    atb_send_received      = 1
    atb_confirm_received   = 2
    atb_confirm_send_received       = 3
    atb_confirm_dealloc_received    = 4
    atb_take_syncpt                 = 5                   /* @L0A*/
    atb_take_syncpt_send            = 6                   /* @L0A*/
    atb_take_syncpt_dealloc         = 7                   /* @L0A*/
  /* ****************************************************************/
  /*  Sync Level Values                                             */
  /* ****************************************************************/
    atb_none               = 0
    atb_confirm            = 1
    atb_syncpt             = 2                          /* @L0A*/
  /* ****************************************************************/
  /*  Set Syncpt Options Values                               @L0A*/
  /* ****************************************************************/
    atb_syncpt_options_nochange     = 0                     /*@L0A*/
    atb_vote_read_only_no           = 1                     /*@L0A*/
    atb_vote_read_only_yes          = 2                     /*@L0A*/
    atb_wait_for_outcome_no         = 1                     /*@L0A*/
    atb_wait_for_outcome_yes        = 2                     /*@L0A*/
    atb_action_if_problems_commit   = 1                     /*@L0A*/
    atb_action_if_problems_backout  = 2                     /*@L0A*/
```

このファイルを作成することになった場合は、新規の FB 80 z/OS データ・セットを割り振り、別のデータ・セットを使用するよう TDIEXEC スクリプトを編集する必要があります。例えば、ROOT.MYATBREX という名前のデータ・セットを作成した場合は、TDIEXEC を以下のように編集する必要があります。

```
...
/*********************************************************************/
 /*  Get psuedonym definition file for REXX for the LU6.2 Verbs      */
 /*  sys1.sieahdr.h(atbpbrex)                                        */
 /*********************************************************************/

"alloc f(datain) da('ROOT.MYATBREX') shr reuse"
"execio * diskr datain (stem  linelist.  finis"
do x = 1 to linelist.0
  interpret linelist.x
end
drop linelist.
"free f(datain)"
...
```

注: 提供されている ATBPBREX ファイルは 1 つの例にすぎないため、すべての z/OS 環境で使用できるとは限りません。

関数コンポーネントのネイティブ部分のセットアップ

TSO コマンド行関数コンポーネントを使用する前に、REXX スクリプトを z/OS データ・セット上にデプロイし、それに合わせて APPC を構成する必要があります。

z/OS TSO コマンド行関数コンポーネントには TDIEXEC という名前の REXX スクリプトが含まれています。このスクリプトは、TSO/E コマンドを実行してコマンドの出力を戻します。

この REXX スクリプトは、それを呼び出す場所の FB 80 z/OS データ・セットにコピーする必要があります。

z/OS TSO コマンド行関数コンポーネントには TDITP.jcl という名前の JCL が含まれています。この JCL は、REXX スクリプト用の TP プロファイルを定義します。

その JCL を z/OS のシステム環境に合わせてカスタマイズし、実行します。

この関数コンポーネントをデプロイする手順の詳細は次のとおりです。

1. JCL および REXX スクリプトを配置する PDS データ・セットを確認します (または、割り振ります)。JCL と REXX スクリプトは、同じデータ・セット内に配置することも、別々のデータ・セット内に配置することもできます。

   TDITP.jcl という JCL と TDIEXEC という REXX スクリプトは、IBM Security Directory Integrator インストール・フォルダーの tso_fc サブフォルダーにあります。
2. REXX スクリプトと JCL をサンプル・ライブラリーから作成済みの PDS データ・セットにコピーします。

   この操作を行うには、例えば、TSO シェルまたは ISPF のメニュー 6 から以下のコマンドを実行します。

   ```
   OGET 'TDI_install_dir/tso_fc/TDIEXEC' '<TDIEXEC_dataset>(TDIEXEC)'
   OGET 'TDI_install_dir/tso_fc/TDITP.jcl' '<TDITP.jcl_dataset>(TDITP)'
   ```
3. ご使用の環境に合わせて TDITP.jcl をカスタマイズします。

   カスタマイズを行うには、TDITP.jcl JCL 内にある指示に従います。基本的には、以下の名前を指定する必要があります。

   - TDIEXEC REXX スクリプトが存在するデータ・セットの名前。ステップ 1 で識別または割り振られたデータ・セットの名前を指定します。
   - APPC TP プロファイル・データ・セットの名前。各 APPC トランザクション・プログラム (TP) では APPC/MVS に対して TP プログラムが定義されています。これらの定義は、APPC TP プロファイル・データ・セットに保管されています。このデータ・セットのデフォルト名は SYS1.APPCTP ですが、インストールごとにカスタマイズできます。
   - トランザクション・スケジューラー定義クラスの名前。デフォルトでは、ASCH 構成ファイルが USER.PARMLIB データ・セット内にあります。ASCHPMxx (「xx」には、例えば ASCHPM00 や ASCHPM1A などが入ります) という名前の PARMLIB メンバーをブラウズし、定義済みの任意のクラスの名前を使用します。

'CLASSADD CLASSNAME(MYCLASS) MSGLIMIT(1000) MAX(10) MIN(1) RESPGOAL(1)' に類似した定義を追加することにより、トランザクション・イニシエーターの独自のクラスを作成することもできます。この定義は、'SET ASCH=xx' システム・コマンドを使用して活動化する必要があります。ここで、「xx」は、ASCHPMxx USER.PARMLIB メンバーの最後の 2 文字です。

注: 開始済みのトランザクション・イニシエーターの最小数は、同時に実行しているトランザクションの予期されている数に対応している必要があります。

4. APPC および ASCH (トランザクション・スケジューラー) が開始済みであることを確認します。APPC/MVS およびトランザクション・スケジューラー用の APPC および ASCH アドレス・スペースが、システムでアクティブになっていることを確認します。このことを確認するには、'D APPC' コマンドおよび 'D ASCH' コマンドを実行します。

   これらのコマンドのいずれかまたは両方が実行していない場合は、'S APPC,SUB=MSTR,APPC=xx' コマンドと 'S ASCH,SUB=MSTR,ASCH=xx' コマンドを使用してそれらのコマンドを開始します。「xx」は、APPCPMxx および ASCHPMxx USER.PARMLIB メンバーの最後の 2 文字です。

5. 指定した LU が存在しており、アクティブになっていることを確認します。「destLuName」および「srcLuName」パラメーターが既存の LU を指定しているか、null に設定されている場合は、デフォルトの LU が定義されている必要があります。

   このことは、APPC 構成ファイルを検査することによって確認できます。このファイルは、デフォルトでは USER.PARMLIB データ・セット内にあります。APPCPMxx (「xx」には、例えば APPCCPM00 や APPCPM1A などが入ります) という名前の PARMLIB メンバーには、すべてのローカル LU の定義が含まれています。トランザクション・スケジューラーの基本論理装置には、「BASE」というマークが付けられています。

   注: そのファイルで定義されている LU 名は、USER.VTAMLST メンバー内にある APPC/MVS の VTAM アプリケーション定義に対応している必要があります。

   例として、以下に BASELU 論理装置の定義を示します。

   ファイル: USER.PARMLIB(APPCPM00)

```
LUADD
  ACBNAME(BASELU)
  BASE
  SCHED(ASCH)
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)
SIDEINFO
  DATASET(SYS1.APPCSI)
```

   ファイル: USER.VTAMLST(A01APPC)

```
        VBUILD TYPE=APPL
BASELU  APPL  ACBNAME=BASELU,                                  X
              APPC=YES,                                        X
              AUTOSES=0,                                       X
              DDRAINL=NALLOW,                                  X
              DLOGMOD=#BATCH,                                  X
              DMINWNL=5,                                       X
              DMINWNR=5,                                       X
              DRESPL=NALLOW,                                   X
```

```
DSESLIM=10,                                          X
LMDENT=19,                                           X
MODETAB=LOGMODES,                                    X
PARSESS=YES,                                         X
```

注: LU が定義されている場合でも、その LU がまだアクティブになっていない可能性があります。LU が実際にアクティブであるかどうかを確認するには、'D APPC,LU,ALL' システム・コマンドを実行します。これにより、すべての LU に関する情報が表示されます。特定の LU をアクティブにするには、VTAM コマンド VARY ACT を使用します (例えば 'V NET,ACT,ID=A01APPC'。ここで「A01APPC」は VTAMLST メンバーの名前)。

6. 変更済みの TDITP.jcl を実行依頼します。

   これを ISPF から実行依頼するには、JCL 名の前に「sub」を付けて入力します。

注:

1. ISPF からシステム・コマンドを実行するには、「システム表示および検索機能」に移動し、メニュー「s」とコマンドの先頭に「/」を付けて入力します (例えば、'/s APPC,SUB=MSTR')。
2. 前述のデータ・セットのいずれかがご使用のシステムに存在していない場合は、システム・プログラマーまたはネットワーク管理者に連絡して支援を受けてください。

ファイル転送関数コンポーネント

ファイル転送関数コンポーネントを使用して、指定されたソース・システムからターゲット・システムにファイルを転送することができます。ソース・システムからターゲット・システムにファイルを転送するには、指定されたプロトコルを参照してください。

- FTP プロトコル

  FTPCleint API は、FTP プロトコルを使用して接続を確立してファイルを転送するために使用します。構成エディターまたは入力項目属性を使用して、接続を確立するためのパラメーター (ユーザー名やパスワードなど) の値を指定できます。

- RXA でサポートされるプロトコル

  Remote Execution and Access (RXA) Toolkit 2.3.0.1 のリモート・アクセス・インターフェースを使用して、ファイル転送操作用の接続を確立することもできます。クラスパスに以下の JAR ファイルが必要です。

  – jlanclient.jar
  – ssh.jar
  – rxa-langpack.jar
  – remoteaccess.jar
  – jt400.jar (AS400 接続の場合)

  RXA Toolkit およびそのサポートされるプロトコルについて詳しくは、https://cs.opensource.ibm.com/frs/?group_id=1639&release_id=7640 を参照してください。

ファイル転送関数コンポーネントのアーキテクチャー

提供された図はファイル転送操作を実行するために RXA Toolkit または FTP クライアント・インターフェースを使用したファイル転送関数コンポーネントの論理ビューを示します。

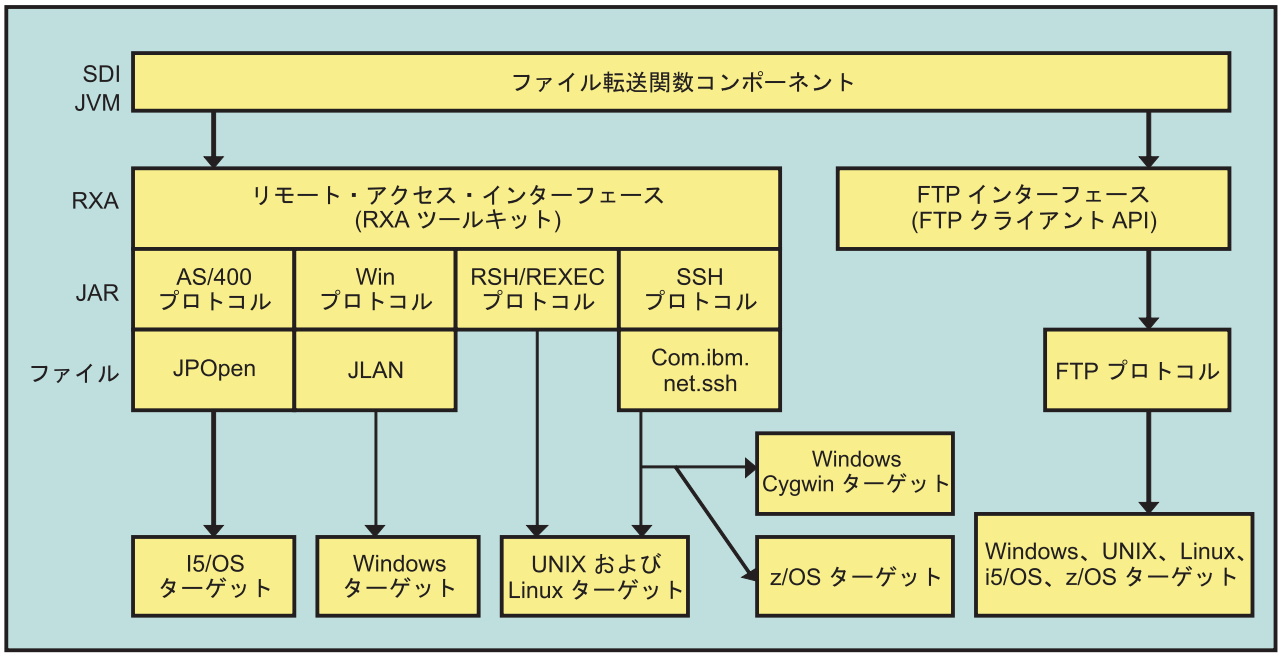


図 8. ファイル転送関数コンポーネントのアーキテクチャー

ファイル転送関数コンポーネントのデータ・ソース・スキーマ

このセクションを参照して、ファイル転送関数コンポーネントの入力スキーマと出力スキーマについて理解することができます。

出力スキーマ

以下の表に、出力マップの属性のリストを示します。

表 48. 出力スキーマ

属性名	説明
source.protocol	ソースの「プロトコル」パラメーターの値をオーバーライドします。
source.path	ソースの「パス」パラメーターの値をオーバーライドします。
source.hostname	ソースの「ホスト名」パラメーターの値をオーバーライドします。
source.port	ソースの「ポート」パラメーターの値をオーバーライドします。
source.user	ソースの「ユーザー名」パラメーターの値をオーバーライドします。

表 48. 出力スキーマ (続き)

属性名	説明
source.password	ソースの「パスワード」パラメーターの値をオーバーライドします。
source.keystore	ソースの「鍵ストア」パラメーターの値をオーバーライドします。
source.passphrase	ソースの「パスフレーズ (パスワード)」パラメーターの値をオーバーライドします。
target.protocol	ターゲットの「プロトコル」パラメーターの値をオーバーライドします。
target.path	ターゲットの「パス」パラメーターの値をオーバーライドします。
target.hostname	ターゲットの「ホスト名」パラメーターの値をオーバーライドします。
target.port	ターゲットの「ポート」パラメーターの値をオーバーライドします。
target.user	ターゲットの「ユーザー名」パラメーターの値をオーバーライドします。
target.password	ターゲットの「パスワード」パラメーターの値をオーバーライドします。
target.keystore	ターゲットの「鍵ストア」パラメーターの値をオーバーライドします。
target.passphrase	ターゲットの「パスフレーズ (パスワード)」パラメーターの値をオーバーライドします。

入力スキーマ

以下の表に、入力マップの属性のリストを示します。

表 49. 入力スキーマ

属性名	説明
$tempFilePath	リモート間のファイル転送操作で作成された一時ファイル・パスの値が含まれます。

ファイル転送の方向

以下の表に示すように、ファイル転送関数コンポーネントを使用してさまざまな方向にファイルを転送することができます。

表 50. ファイル転送の方向

ファイル転送の方向	説明	プロトコル
ローカルからローカル	以下の場合、ファイルは、ローカル・コンピューター上のあるロケーションから別のロケーションに転送されます。 • 構成エディター内の「ソース」セクションと「ターゲット」セクションの両方の「ホスト名」パラメーターが空である。 • source.hostname 属性と target.hostname 属性の両方の入力項目が空である。	ファイル API

表 50. ファイル転送の方向 (続き)

ファイル転送の方向	説明	プロトコル
ローカルからリモート	以下の場合、ファイルは、ローカル・コンピューター上のロケーションからリモート・コンピューター上の別のロケーションに転送されます。 • 構成エディター内の「ソース」セクションの「ホスト名」パラメーターおよび source.hostname 属性の入力項目が空である。 • 構成エディター内の「ターゲット」セクションの「ホスト名」パラメーターまたは target.hostname 属性の入力項目が空ではない。	RXA インターフェースまたは FTP クライアント API
リモートからローカル	以下の場合、ファイルは、リモート・コンピューター上のロケーションからローカル・コンピューター上の別のロケーションに転送されます。 • 構成エディター内の「ソース」セクションの「ホスト名」パラメーターまたは source.hostname 属性の入力項目が空ではない。 • 構成エディター内の「ターゲット」セクションの「ホスト名」パラメーターおよび target.hostname 属性の入力項目が空である。	RXA インターフェースまたは FTP クライアント API
リモートからリモート	以下の場合、ファイルは、リモート・コンピューター上のロケーションから別の/同一のリモート・コンピューター上の別のロケーションに転送されます。 • 構成エディター内の「ソース」セクションと「ターゲット」セクションの両方の「ホスト名」パラメーターが空ではない。 • source.hostname 属性と target.hostname 属性の入力項目が空ではない。 この場合、リモート・コンピューターとの接続が確立されます。ソース・リモート・コンピューターからのファイルが、ローカル・コンピューター上の一時ロケーションで受信されます。受信したファイルは、一時ロケーションからターゲット・リモート・コンピューターに転送されます。	RXA インターフェースまたは FTP クライアント API

ターゲット・システムの構成

指定されたセクションに記載されているプラットフォーム上のリモート・システムにファイルを転送するための前提条件である、外部システムの構成に従う必要があります。

Windows システム

ターゲットの Windows コンピューターについて、以下の要件が満たされていることを確認する必要があります。

- Windows XP ターゲット・システムで「簡易ファイルの共有」機能を使用不可にします。「簡易ファイルの共有」機能を使用不可にするには、以下のようにします。
  1. Windows エクスプローラーを開きます。
  2. 「**ツール**」->「**フォルダー オプション**」を選択します。
  3. 「**表示**」タブをクリックします。

4. 「**ファイルの簡易共有を使用する (推奨)**」チェック・ボックスを選択します。

- Windows ファイアウォール設定を構成します。

  Windows XP には、組み込みのインターネット接続ファイアウォール (ICF) が含まれています。デフォルトでは、Windows XP システム上の ICF は使用不可になっています。Windows XP Service Pack 2 では、ファイアウォールが使用可能に設定されています。Windows XP または Vista のターゲットでいずれかのファイアウォールが使用可能になっている場合は、RXA によるアクセス試行がブロックされます。XP Service Pack 2 では、「Windows ファイアウォール」構成の「**例外**」タブにある「**ファイルとプリンタの共有**」チェック・ボックスを選択すると、アクセスを許可できます。
- RXA がコマンドおよびスクリプトを実行できるように、ターゲット・コンピューターのリモート・レジストリー管理を使用可能にします。
- デフォルトの非表示の管理ディスク共有 (C$ や D$ など) が RXA の適切な操作のために使用可能になっている状態にします。

Cygwin システム

ターゲットの Cygwin コンピューターについて、以下の要件が満たされていることを確認する必要があります。

- リモート Windows コンピューターへの SSH ログインを使用するために、Windows ターゲット・コンピューターに Cygwin をインストールします。Cygwin は http://cygwin.com からダウンロードできます。
- Cygwin ターゲット・コンピューターで RXA アプリケーションを使用するために、以下の追加の Cygwin パッケージをインストールします。
  - 「Net」カテゴリーから `openssh` パッケージをインストールします。`openssh` パッケージには、`ssh` デーモンが含まれています。このデーモンは、ターゲット Cygwin コンピューターで SSH ログインをサポートするために必要です。
  - 「Admin」カテゴリーから `cygrunsrv` パッケージをインストールします。`cygrunsrv` パッケージは、`ssh` デーモンを Windows サービスとして実行するために使用されます。

UNIX および Linux システム

ターゲットの UNIX コンピューターと Linux コンピューターについて、以下の要件が満たされていることを確認する必要があります。

- SSH プロトコルを使用してアクセスするターゲット・コンピューターで、SSH がインストールされ、使用可能な状態になっていることを確認します。
- UNIX ターゲット・コンピューターでは、Bourne シェル (sh) を使用します。
- Solaris ターゲット・コンピューターでは、Korn シェル (ksh) を使用します。
- パスワード認証を使用した、Linux およびその他の SSH ターゲット・システムとの RXA 通信のために、以下のステップを使用して、`/etc/ssh/sshd_config` ファイルを編集します。
  1. `PasswordAuthentication` オプションを `yes` に設定します。
  2. 以下のコマンドを使用して SSH デーモンを停止して再始動します。

     ```
     /etc/init.d/sshd stop
     /etc/init.d/sshd start
     ```

AS400 システム

ターゲットの AS400 コンピューターについて、以下の要件が満たされていることを確認する必要があります。

- ターゲット AS400 システムに、適切な JRE とともに IBM Toolbox for Java をインストールします。
- IBM Security Directory Integrator サーバーに IBM Toolbox for Java をインストールします。JAR ファイルは、このサーバーの *TDI_install_dir*/jars/3rdParty/IBM ディレクトリーに保管されます。
- セキュア接続のために、ターゲット AS400 システムで SSL を構成します。

  V5R3 システムで SSL をセットアップする説明については、
  http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/rzahh/sslcert.htm
  を参照してください。

構成パラメーター

以下の各セクションでは、ファイル転送関数コンポーネントのパラメーターについて説明します。

一時ディレクトリー
このパラメーターを使用して、リモートからリモートのファイル転送操作時にファイルを一時的に保管するためのディレクトリー・ロケーションを指定します。

一時ファイルの削除
このパラメーターを使用して、一時ディレクトリー内のファイル (リモートからリモートのファイル転送操作時に保管される) を削除するかどうかを指定します。

サブディレクトリーを含む
このパラメーターを使用して、ファイル転送操作でソースまたはターゲットのロケーション内のファイルを検索する際にサブディレクトリーを含めるかどうかを指定します。

コメント
このパラメーターを使用して、コメントを追加します。データの構文解析時には、このコメントは無視されます。

詳細ログ
このパラメーターを使用して、詳細なログ・メッセージを生成します。

ソース・オプション

ソース・オプションのリストを以下に示します。

プロトコル
このパラメーターを使用して、ソース・コンピューターに接続するためのプロトコルを指定します。

ソース・ファイル・パス
このパラメーターを使用して、ターゲット・コンピューターに転送するソース・ファイルの絶対パスまたは共有パスを指定します。

ホスト名
このパラメーターを使用して、ソース・コンピューターのホスト名を指定します。

ポート このパラメーターを使用して、ソース・コンピューターのポート番号を指定します。ポート番号を指定しなかった場合は、指定したプロトコルのデフォルト・ポート値が使用されます。

ユーザー名
このパラメーターを使用して、ソース・コンピューターにログインするための有効なユーザー ID を指定します。

パスワード
このパラメーターを使用して、ソース・コンピューターのユーザー ID に関連付けられたパスワードを指定します。

鍵ストア
このパラメーターを使用して、ソース・コンピューターの鍵ストア・ファイル・パスを指定します。

パスフレーズ (パスワード)
このパラメーターを使用して、鍵ストア・ファイルを開くためのパスフレーズを指定します。

拡張ソース・オプション

ここに記載されている拡張ソース・オプションのリストを使用することができます。

FTP 受動モード
FTP サーバーに受動モードで接続する必要があるかどうかを指定します。

注: IPv6 接続では、アクティブ・モードのみを使用するため、このパラメーターは無視されます。

FTP セキュリティー
このパラメーターを使用して、FTP 接続のセキュリティー・タイプおよびセキュリティー・レベルを指定します。

FTP 明示モード SSL (FTPES)
このパラメーターを使用して、非 SSL ソケット (FTPES) で SSL セッションをネゴシエーションするかどうかを指定します。

(RXA) 操作のタイムアウト
このパラメーターを使用して、RXA でサポートされるプロトコルを使用したファイル転送操作のタイムアウト期間 (ミリ秒) を指定します。

(RXA) ファイル・エンコードの変換
このパラメーターを使用して、ソース・ファイルをターゲット・コンピューターの文字セットに変換する必要があるかどうかを指定します。

AS400 への SSL を有効にする
このパラメーターを使用して、ターゲット AS/400® システムで SSL 接続を確立する必要があるかどうかを指定します。

注: ターゲット AS/400 システムで SSL をインストールして構成する必要があります。

AS400 プロキシー・サーバー名
このパラメーターを使用して、AS/400 プロキシー・サーバーのホスト名または IP アドレスを指定します。

ターゲット・オプション

ターゲット・オプションのリストを以下に示します。

プロトコル
このパラメーターを使用して、ターゲット・コンピューターに接続するためのプロトコルを指定します。

ターゲット・ディレクトリー・パス
このパラメーターを使用して、ファイルを転送する必要がある宛先のターゲット・ディレクトリーの絶対パスまたは共有パスを指定します。

ターゲット・ディレクトリーの作成
このパラメーターを使用して、ターゲット・コンピューター上にディレクトリーを作成するかどうかを指定します。

ホスト名
このパラメーターを使用して、ターゲット・コンピューターのホスト名を指定します。

ポート このパラメーターを使用して、ターゲット・コンピューターのポート番号を指定します。ポート番号を指定しなかった場合は、指定したプロトコルのデフォルト・ポート値が使用されます。

ユーザー名
このパラメーターを使用して、ターゲット・コンピューターにログインするための有効なユーザー ID を指定します。

パスワード
このパラメーターを使用して、ターゲット・コンピューターのユーザー ID に関連付けられたパスワードを指定します。

鍵ストア
このパラメーターを使用して、ターゲット・コンピューターの鍵ストア・ファイル・パスを指定します。

パスフレーズ (パスワード)
このパラメーターを使用して、鍵ストア・ファイルを開くためのパスフレーズを指定します。

拡張ターゲット・オプション

ここに記載されている拡張ターゲット・オプションのリストを使用することができます。

FTP 受動モード
FTP サーバーに受動モードで接続する必要があるかどうかを指定します。

注: IPv6 接続では、アクティブ・モードのみを使用するため、このパラメーターは無視されます。

FTP セキュリティー
このパラメーターを使用して、FTP 接続のセキュリティー・タイプおよびセキュリティー・レベルを指定します。

FTP 明示モード SSL (FTPES)
このパラメーターを使用して、非 SSL ソケット (FTPES) で SSL セッションをネゴシエーションするかどうかを指定します。

(RXA) 操作のタイムアウト
このパラメーターを使用して、RXA でサポートされるプロトコルを使用したファイル転送操作のタイムアウト期間 (ミリ秒) を指定します。

(RXA) ファイル・エンコードの変換
このパラメーターを使用して、ソース・ファイルをターゲット・コンピューターの文字セットに変換する必要があるかどうかを指定します。

AS400 への SSL を有効にする
このパラメーターを使用して、ターゲット AS/400 システムで SSL 接続を確立する必要があるかどうかを指定します。

注: ターゲット AS/400 システムで SSL をインストールして構成する必要があります。

AS400 プロキシー・サーバー名
このパラメーターを使用して、AS/400 プロキシー・サーバーのホスト名または IP アドレスを指定します。

拡張オプション

拡張オプションのリストを以下に示します。

FTP 転送モード
このパラメーターを使用して、FTP でのファイル転送モード (ASCII またはバイナリーなど) を指定します。

ASCII モードは、テキスト・ファイルをあるフォーマットから別のフォーマットに自動的に変換する場合に使用します。例えば、UNIX ファイル・システムでは、ファイル内の行は改行 (LF) で終了します。Windows および DOS ファイルでは、行は復帰 (CR) と改行 (LF) で終了します。

注: ASCII モードでバイナリー・ファイルを送信すると、バイナリー・ファイルの構造が破損します。

バイナリー・モードは、ファイルを元の形式で転送する場合に使用します。

RXA 内部ロギングを有効にする
このパラメーターを使用して、RXA 内部ロギングを AssemblyLine ログ・ファイルに送信するかどうかを指定します。

第 5 章 SAP ABAP Application Server Component Suite

IBM Security Directory Integrator と SAP ABAP Application Server との統合を実現するために必要な、ここに記載されている手順を使用することができます。

IBM Security Directory Integrator の各コンポーネントは、ユーザー・ディレクトリーおよびその他のリソースの管理を担当するネットワーク管理者用に設計されています。このセクションでは、ユーザーが IBM Security Directory Integrator と SAP ABAP Application Server の両方を実際にインストールおよび使用した経験を有していることを前提としています。

このセクションでは、IBM Security Directory Integrator と SAP ABAP Application Server のインストールと構成が完了しており、ネットワーク上でこれらの製品が稼働していることを前提としています。統合に関連して必要な場合を除き、これらの製品のインストールと構成についての詳細は説明しません。

Component Suite のインストール

ここに記載されている IBM Security Directory Integrator Component Suite for SAP ABAP Application Server のソフトウェア要件およびインストール手順を使用できます。

ソフトウェア要件

ここに記載されているコンポーネントをターゲット・マシンに追加して、Component Suite のインストールを完了できます。

IBM Security Directory Integrator をインストールすると、この Component Suite もインストールされます。追加コンポーネントは以下のとおりです。

- SAP Java Connector (JCo) バージョン 2.1.6、2.1.7、2.1.8。これらのバージョンは、SAP ABAP Application Server でサポートされる SAP JCo のバージョンです。

IBM Security Directory Integrator Component Suite for SAP ABAP Application Server は、IBM Security Directory Integrator と SAP JCo の共通のオペレーティング・システム・プラットフォームでサポートされています。 IBM Security Directory Integrator でサポートされているオペレーティング・システムについては「IBM Security Directory Integrator 管理者ガイド」を、SAP JCo でサポートされているプラットフォームについては SAP Web サイトを参照してください。SAP JCo には、他にも以下の前提条件があります。

Windows

SAP JCo ライブラリーには、MS 8.0 C/C++ ランタイムが必要です。説明については、 SAP ノート 684106 を参照してください。次善策として、msvcr71.dll、msvcp71.dll、 mfc71.dll、mfc71u.dll などを、その他の Windows コンピューター (32 および 64 ビット・バージョンの DLL が使

用可能です。コピーが必要なバージョンは、ご使用の SAP DLL のバージョンと一致している必要があります) からコピーできます。

Linux 最新バージョンの libstdc++、libgcc、および compat-libstdc++ が必要です。C++ Runtime 6.0 (libstdc++.so.6) に関する情報は、 SAP ノート 1021236 にあります。

ライセンス交付を受けた SAP ABAP Application Server 利用者は、SAP Web サイトから JCo をダウンロードできます。SAP サポートへログインするための有効なアカウントとパスワードが必要です。このアカウントとパスワードは、SAP サポートに要求して入手できます。SAP ABAP Application Server のサポートされているバージョンがネットワーク環境内のノードにインストールされ、稼働している必要があります。SAP ABAP Application Server インスタンスと、IBM Security Directory Integrator Component Suite for SAP ABAP Application Server をホスティングしているマシンとの間に TCP/IP ネットワーク接続が必要です。

SAP ABAP Application Server のサポートされているバージョンは以下のとおりです。

- SAP ABAP Application Server v6.20
- SAP ABAP Application Server v6.40
- SAP ABAP Application Server v7.0

SAP Java Connector の構成

ここに記載されている手順を使用して、SAP Java Connector を構成できます。

IBM Security Directory Integrator と Component Suite for SAP ABAP Application Server のホストとなるマシンに JCo をダウンロードし、使用可能な状態になったら、次の手順で Jco をインストールし、IBM Security Directory Integrator 用に構成します。

1. JCo 配布パッケージをターゲット・マシンのディレクトリーに unzip します。例を示します。

   ```
   /SapJco216
   ```

2. `installation.html` ファイルを開き、ご使用のオペレーティング・システムのインストール手順に従います。例を示します。

   ```
   /SapJco216/docs/jco/installation.html
   ```

3. 以下の項目をネットワーク・サービス・ファイルに追加します。

   - `sapdpNN 32NN/tcp`
   - `sapgwNN 33NN/tcp`

   ここで、*NN* は、IBM Security Directory Integrator Component Suite for SAP ABAP Application Server と接続する SAP システムの SAP インスタンス ID です。

4. SAP JCo パッケージ・ディレクトリーから `sapjco.jar` を *IBM Security Directory Integrator_HOME*`/jars` フォルダーにコピーします。

5. 615 ページの『ALE Intermediate Document (IDOC) Connector for SAP ABAP Application Server および SAP ERP』 を使用する場合は、同じロケーションにも `sapidoc.jar` と `sapidocjco.jar` をコピーする必要があります。

6. **Windows マシンのみ該当する手順:** librfc32.dll および sapjcorfc.dll を *IBM Security Directory Integrator_HOME*/libs フォルダーにコピーします。

注:

1. ネットワーク・サービス・ファイルは %system_root %¥system32¥drivers¥etc¥services (Windows 32) または /etc/services (UNIX) にあります。
2. IBM Security Directory Integrator Component Suite for SAP ABAP Application Server を使用する前に、sapjco.jar が CLASSPATH に含まれており、sapjcorfc.{dll/so} と librfc*.{dll/so} がロード可能なライブラリーのパスに含まれていることを確認してください。

Component Suite for SAP ABAP Application Server の検証

以下の表に記載されている詳細を使用することで、Component Suite for SAP ABAP Application Server を検証できます。

IBM Security Directory Integrator Component Suite for SAP ABAP Application Server を検証するには、以下のようにします。

次の表 51 では、Component Suite に関連するファイルと、IBM Security Directory Integrator のシステム・インストーラーによりこれらのファイルがインストールされる場所を説明します。

表 51. IBM Security Directory Integrator Component Suite のインストール・ロケーション

ファイル名	説明
SapR3BorConnector.jar	*IBM Security Directory Integrator_HOME*/jars/connectors
SapR3UserConnector.jar	*IBM Security Directory Integrator_HOME*/jars/connectors
SapR3RfcFC.jar	*IBM Security Directory Integrator_HOME*/jars/functions
index.html (すべての SAP コンポーネントの Javadoc)	*IBM Security Directory Integrator_HOME*/docs/api/
bapi_user_actgroups_assign.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_actgroups_delete.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_change.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_create.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_delete.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_getdetail_postcall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_getdetail_precall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_getlist_postcall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_getlist_precall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_profiles_assign.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_user_profiles_delete.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_employee_dequeue.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_employee_enqueue.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_employee_getdata_postcall.xsl	*IBM Security Directory Integrator_HOME*/xsl

表 51. *IBM Security Directory Integrator Component Suite* のインストール・ロケーション (続き)

ファイル名	説明
bapi_employee_getdata_precall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_change.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_create.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_delete.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_getdetail_postcall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_getdetail_precall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_getdetailedlist_postcall.xsl	*IBM Security Directory Integrator_HOME*/xsl
bapi_persdata_getdetailedlist_precall.xsl	*IBM Security Directory Integrator_HOME*/xsl

注: コネクターは XSL スタイルシートを使用して操作を実行しますが、デフォルトでは相対パスを使用してスタイルシートを見つけます (例えば xsl/bapi_user_getlist_precall.xsl)。IBM Security Directory Integrator ソリューション・ディレクトリーを使用する場合、デフォルトでは相対パスが使用されることを認識しておくことが重要です。その結果、以下のいずれかを実行する必要が生じます。

1. *TDI_install_dir*/xsl フォルダーを IBM Security Directory Integrator ソリューション・ディレクトリーにコピーします。
2. ソリューション・ディレクトリーを IBM Security Directory Integrator インストール・ディレクトリーになるように設定します。
3. ソリューション・ディレクトリーを構成しないことを選択します。

IBM Security Directory Integrator ソリューション・ディレクトリーに XSL フォルダーがない場合、SAP コネクターを使用しようとすると、以下のようなエラーが発生します。

```
com.ibm.di.connector.sapr3.user.UserRegistryConnectorException: CTGDIK019E The Connector detected an
 exception during initialization. The message is: 'CTGDIK008E The configured XSL file named
 'xsl/bapi_user_getdetail_precall.xsl' does not exist.'.
```

バージョン番号の確認

ここに記載されている手順を使用して、この統合パッケージのコンポーネント・ソフトウェアのバージョン番号を確認できます。

1. IBM Security Directory Integrator を始動し、「**ヘルプ**」をクリックします。
2. 「**IBM Security Directory Integrator コンポーネント**」を選択します。
3. 以下のコンポーネントのバージョン番号が表示されます。
   - **ibmdi.SapR3RfcFC**
   - **ibmdi.SapR3UserRegConnector**
   - **ibmdi.SapR3BorConnector**

アンインストール

ここに記載されている手順を使用することで、ターゲット・システムから IBM Security Directory Integrator Component Suite for SAP ABAP Application Server を削除できます。

1. 現在実行中で、IBM Security Directory Integrator Components for SAP ABAP Application Server の 1 つを使用している IBM Security Directory Integrator Assembly Line を停止します。
2. *IBM Security Directory Integrator_HOME*/_uninstsap にあるアンインストール実行プログラムを実行し、プロンプトに従って操作します。
3. ネットワーク・サービス・ファイル (Windows 32 の場合は %system_root%¥system32¥drivers¥etc¥services、UNIX の場合は /etc/services) から以下の項目を除去します。
   - sapdp*NN* 32*NN*/tcp
   - sapgw*NN* 33*NN*/tcp

   ここで、*NN* は、IBM Security Directory Integrator Component Suite for SAP ABAP Application Server に接続している SAP システムの SAP インスタンス ID です。
4. インストール時に作成された SAP JCo (*SAP_JCO_HOME*) ディレクトリーを除去します。
5. インストール時に *SAP_JCO_HOME*/docs/jco/installation.html の指示に従って作成した環境変数項目と追加項目を除去します。
6. sapjco.jar を *TDI_install_dir*/jars フォルダーから除去します。
7. **Windows マシンのみに該当する手順:** librfc32.dll および sapjcorfc.dll の各ファイルを、*IBM Security Directory Integrator_HOME*/libs フォルダーから削除します。

SAP ABAP Application Server の関数コンポーネント

ここに記載されている情報を使用して、SAP ABAP Application Server 用の IBM Security Directory Integrator 関数コンポーネントを操作できます。

このセクションでは、SAP ABAP Application Server 用 IBM Security Directory Integrator 関数コンポーネントについて説明します。

SAP ABAP Application Server 用の関数コンポーネントは、SAP JCo を使用して SAP ABAP Application Server システム上の RFC を呼び出します。この関数コンポーネントでは、任意の RFC を呼び出すことができます。

RFC 関数コンポーネントのアーキテクチャーの概要を以下の図に示します。

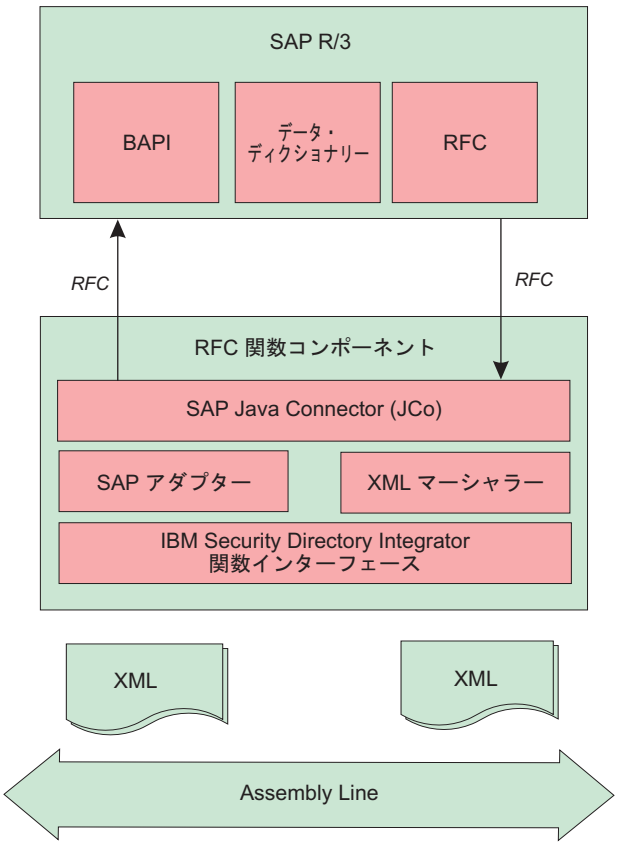


図 9. RFC 関数コンポーネントのアーキテクチャー概要

SAP ABAP Application Server 用関数コンポーネントを使用する前に、SAP JCo をダウンロードしてインストールしておく必要があります (詳しくは『ソフトウェア要件』を参照)。

構成

ここに記載されているパラメーターを使用して、SAP ABAP Application Server を構成できます。

SAP ABAP Application Server (SAP ABAP AS) 用関数コンポーネントを直接 Assembly Line に追加する場合は、クライアント接続用の以下の構成パラメーターを使用できます。これらのパラメーターは、従来の SAP GUI のログオン・パラメーターに非常に類似しています。ランタイム名を括弧で囲んで示します。

パラメーター

上記で説明したパラメーターのリストを使用することができます。

ABAP AS クライアント (client)
SAP 接続のための SAP ABAP AS ログオン・クライアント。例: 100。

ABAP AS ユーザー (user)
SAP 接続のための SAP ABAP AS ログオン・ユーザー。

パスワード (passwd)
SAP 接続のための SAP ABAP AS ログオン・パスワード。

ABAP AS システム番号 (sysnr)
SAP 接続のための SAP ABAP AS システム番号。例: 00。

ABAP AS ホスト名 (ashost)
SAP 接続のための SAP ABAP AS アプリケーション・サーバー名。

ゲートウェイ・ホスト (gwhost)
SAP 接続のためのゲートウェイ・ホスト名。

RFC トレース (trace)
RFC API トレースを使用可能にするには、*1* に設定します。使用可能に設定されると、SAP RFC API は個別の `rfc_nnnn.trc` ファイル (*nnnn* は RFC API により割り当てられた値を表す) を IBM Security Directory Integrator の作業ディレクトリーに作成します。このオプションは、RFC 呼び出しの問題を診断する際に役立ちます。このオプションを設定すると、コネクターと SAP ABAP AS との間のアクティビティーとデータがログに書き込まれます。実動デプロイメントの場合はゼロ (*0*) に設定してください。

関数コンポーネントをプログラマチックに使用する場合は、追加の構成パラメーターが使用可能になります。追加のパラメーターについての詳細は、配布パッケージの `SapR3RfcFC` Java Doc を参照してください。

関数コンポーネントの入力

ここに記載されている関数コンポーネントの入力のリストを使用して、SAP ABAP Application Server 関数コンポーネントを構成できます。

perform() メソッドは **Entry** オブジェクトを受け入れます。これら以外のものを渡すと、例外がスローされます。**項目**オブジェクトには 2 つの属性があります。

- **requestType**
- **request**

この関数コンポーネントでは、3 種類の呼び出しスタイルがサポートされています。

- XML 文書
- XML ストリング
- 多値属性

使用するスタイルを指定するには、**requestType** に次のいずれかの値を設定します。

- *xmlDomDocument*
- *xmlString*
- *multiValuedAttributes*

属性 **request** の値のタイプを以下に示します。

- `org.w3c.dom.Document` (**requesttype** が *xmlDomDocument* の場合)
- `java.lang.String` (**requesttype** が *xmlString* の場合)
- `com.ibm.di.entry.Attribute` (**requesttype** が *multiValuedAttributes* の場合)

request の値は、RFC の要求データを以下のいずれかとして表します。

- XML ストリング
- DOM 文書
- 多値属性 (多値属性呼び出しを使用するサンプル JavaScript については、Javadoc を参照)

上記以外の値を指定すると、例外がスローされます。

request のタイプが org.w3c.dom.Document の場合:
関連付けられている値は、ABAP RFC XML シリアライゼーションの仕様に準拠した XSchema が含まれている org.w3c.dom.Document でなければなりません。

request のタイプが java.lang.String の場合:
関連付けられている値は XML ストリングでなければなりません。DOM パーサーによりストリング値が解析されます。その XSchema も「Serialization of ABAP Data in XML」の仕様に準拠している必要があります。

request が多値属性の場合:
属性 **request** の 1 番目の値はタイプ java.lang.String である必要があり、RFC の名前が含まれていなければなりません。属性 **request** の 2 番目の値は com.ibm.di.entry.Attribute である必要があり、この値には、SAP RFC パラメーターの追加属性が、RFC のインポート・パラメーターとテーブル・パラメーターの名前を表す一連のネストされた多値属性として含まれています。パラメーターの名前は、「Serialization of ABAP Data in XML」のルールに基づいてエンコードする必要があります (名前には、不正な形式の XML を生成する可能性がある文字は含まれません)。

多値属性スタイルを使用して関数コンポーネントを呼び出す方法の例を以下に示します。

```
var rfc = system.newAttribute("BAPI_SALESORDER_GETLIST");
var attr1 = system.newAttribute("CUSTOMER_NUMBER");
attr1.addValue("0000000016");
var attr2 = system.newAttribute("SALES_ORGANIZATION");
attr2.addValue("AU01");
rfc.addValue(attr1);
rfc.addValue(attr2);
var entry = system.newEntry();
var reqAttr = entry.newAttribute("request");
reqAttr.addValue(rfc);
entry.setAttribute("requestType", "multiValuedAttributes");
var result = fc.perform(entry);
```

関数コンポーネントの出力

ここに記載されている関数コンポーネントの出力のリストを使用して、SAP ABAP Application Server 関数コンポーネントを構成できます。

関数コンポーネントの出力は、次の 2 つの属性を持つ**項目**オブジェクトです。

- **responseType**。応答タイプを示します。
- **response**。RFC 応答を DOM 文書、XML ストリング、またはネストした多値属性のいずれかとして示します。

属性 **responseType** には、入力要求タイプに対応した java.lang.String 値が含まれています。

項目の属性 responseType の値が *xmlDomDocument* の場合
属性 **response** の値は、RFC 応答を含む org.w3c.dom.Document です。

項目の属性 responseType の値が *xmlString* の場合
属性 **response** の値は、RFC 応答を含む XML `java.lang.String` です。

項目の属性 responseType の値が *multiValuedAttributes* の場合
属性 **response** の値はネストした多値属性です。1 番目の値は、呼び出された RFC の名前を含む `java.lang.String` であり、2 番目の値には RFC の結果が一連のネスト多値属性として含まれています。

関数コンポーネントの使用

関数コンポーネントを使用して、SAP ABAP Application Server システムの指定の RFC を呼び出すことができます。

Assembly Line に挿入するか、またはスクリプトから直接呼び出すことができます。呼び出し側は、戻された項目オブジェクトに、RFC 呼び出しが原因で発生したエラーがあるかどうかを確認する必要があります。

JavaScript から RFC を呼び出す際に利用できるコードの例を以下に示します。

```
var counter = 0;
var fc = system.getFunction("ibmdi.SapR3RfcFC");
var myentry;
var docResponse;

fc.setParam(fc.PARAM_CONFIG_CLIENT, "100");
fc.setParam(fc.PARAM_CONFIG_USER, "TIVOLI");
fc.setParam(fc.PARAM_CONFIG_PASSWORD, "******");
fc.setParam(fc.PARAM_CONFIG_SYSNUMBER, "11");
fc.setParam(fc.PARAM_CONFIG_LANGUAGE, "E");
fc.setParam(fc.PARAM_CONFIG_APPLICATION_SERVER, "kimala");

fc.initialize(null);
var rfc = new java.lang.String("<BAPI_COMPANYCODE_GETLIST/>");
var myentry = system.newEntry();
var attr = myentry.newAttribute(fc.PARAM_INPUT_TYPE);
attr.addValue(fc.PARAM_VAL_STRING);

attr = myentry.newAttribute(fc.PARAM_INPUT);
attr.addValue(rfc);
var myresponse = fc.perform(myentry);

//system.dumpEntry(myresponse);
fc.terminate();
```

注: initialize() を呼び出す前に構成パラメーターを設定する必要があります。また、クリーンアップのために **terminate()** を呼び出す必要があります。

コマンド行 RFC Invoker の使用

ここに記載されている手順を使用して、コマンド行 RFC Invoker を使用できます。

有効な RFC XML 要求の作成を支援するツールとして、コマンド行ユーティリティーが用意されています。コマンド行ユーティリティーは IBM Security Directory Integrator 環境の外部から呼び出すことができ、SAP ABAP Application Server システムに対して実行される RFC XML 要求を表す XML ファイルを読み取ることができます。

コマンド行ユーティリティーを呼び出すには、最初に以下の jar を CLASSPATH 環境変数に追加します。

- *TDI_install_dir*`/jars/functions/SapR3RfcFC.jar`
- *TDI_install_dir*`/jars/common/tdiresource.jar`

- *TDI_install_dir*/jars/3rdparty/IBM/icu4j_4_2.jar

次に、以下のコマンドを使用して呼び出します。

```
TDI_install_dir/jvm/bin/java com.ibm.di.fc.sapr3rfc.RfcXmlInvoker -f
   [input XML file] -o [output XML file] -p
   [JCO Connection properties file]
```

注:

1. 上記の手順では、 584 ページの『SAP Java Connector の構成』 の手順を既に完了していることを前提としています。sapjco.jar が CLASSPATH に含まれており、sapjcorfc.{dll/so} と librfc*.{dll/so}がロード可能なライブラリーのパスに含まれていることが重要です。
2. AIX の場合、Java 実行可能ファイルのパスは *TDI_install_dir*/jvm/jre/bin/java.exe です。

JCO プロパティー・ファイルの内容は、SAP システムの SAP ABAP AS クライアント接続パラメーターを表します。このプロパティー・ファイルの値の例を以下に示します。

```
jco.client.client=R/3 CLIENT
jco.client.user=R/3 USER NAME
jco.client.passwd=R/3 USER PASSWORD
jco.client.sysnr=R/3 SYSTEM NUMBER
jco.client.ashost=R/3 APPLICATION SERVER HOSTNAME OR IP ADDRESS
jco.client.trace=RFC API TRACE: 1 = ON; 0 = OFF
```

SAP ABAP Application Server ユーザー・レジストリー・コネクター

ここに記載されている情報を使用することで、IBM Security Directory Integrator SAP ABAP Application Server ユーザー・レジストリー・コネクターを構成し、操作することができます。

このコンポーネントを使用すると、(SAP ABAP Application Server から見て) 外部のアプリケーションへの SAP ユーザー・アカウントのプロビジョニングと管理が可能になります。このコネクターは、IBM Security Directory Integrator Function Component for SAP ABAP Application Server (以降 RFC 関数コンポーネントと呼びます) の汎用 RFC 呼び出し機能を使用します。このコネクターは、RFC 関数コンポーネントを使用することにより SAP ユーザー・アカウントの属性を管理できます。それを行うためにこのコンポーネントは、RFC ABAP コードを外部の SAP ABAP Application Server クライアント・アプリケーションとして実行します。

このコネクターでは、SAP ユーザー・アカウントとその関連属性をプロビジョニングするための拡張可能な汎用フレームワークがサポートされています。これは、ユーザー・アカウント情報の XML 表現を定義することで実現します。この XML は、XSL スタイル・シート変換 (XSLT) により RFC 要求に変換されます。このコネクターのデフォルトの機能を使用する場合は、カスタム RFC ABAP コードをターゲットの SAP ABAP Application Server インスタンスにデプロイする必要はありません。

このコネクターの主な機能と利点は以下のとおりです。

- SAP ユーザーの作成、読み取り、更新、および削除 (C.R.U.D) の各操作をサポートします。

- XSL 変換により SAP ABAP Application Server RFC 実行の動作を変更できます。
- このコネクターと SAP ABAP Application Server の間のコンパイル時間の依存関係が最小限に抑えられています。このコネクターは生成された RFC プロキシー・コードを使用しません。RFC 関数コンポーネントを動的プロキシーとして使用します。

このコネクターでは、以下の IBM Security Directory Integrator コネクター・モードがサポートされています。

- AddOnly
- 更新
- 削除
- ルックアップ
- イテレーター

SAP User Registry のコンポーネント設計を以下の図に示します。

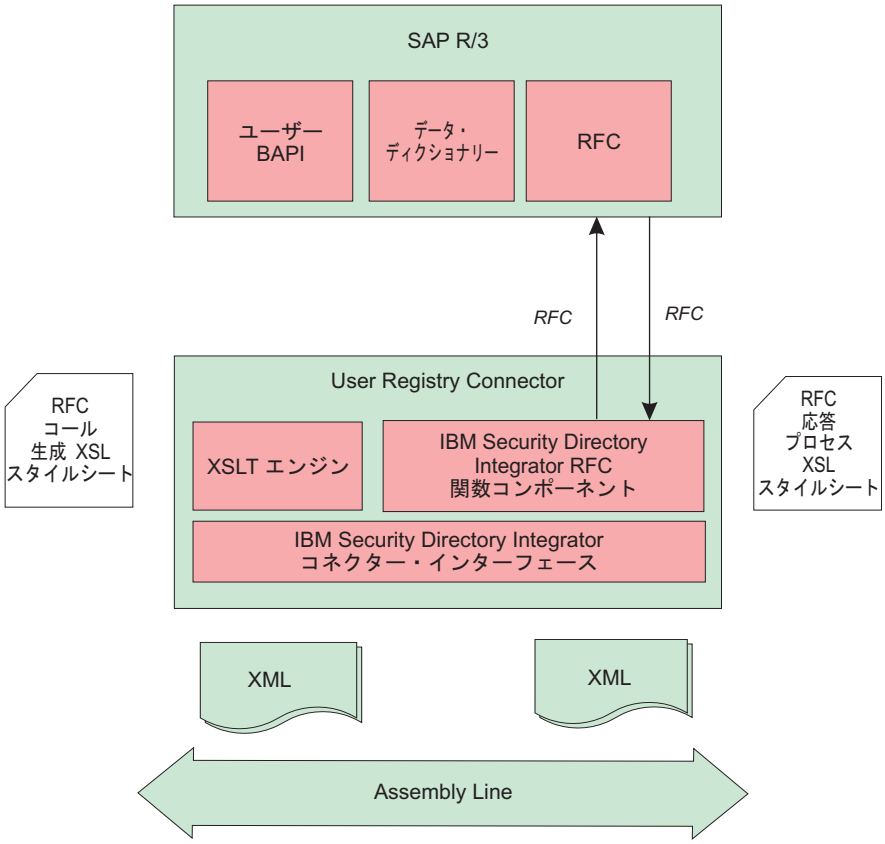


図 10. SAP User Registry のコンポーネント設計

更新モードおよび削除モードでのルックアップのスキップ

ルックアップのスキップ・オプションを使用して、検索を回避できます。

SAP ABAP Application Server ユーザー・レジストリー・コネクターでは、更新モードまたは削除モードでの「**ルックアップのスキップ**」一般オプションがサポートされています。このオプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。

これが機能するには、コネクターのリンク基準で **sapUserName** 属性を定義する必要があります。

構成

SAP ABAP Application Server ユーザー・レジストリー・コネクターを Assembly Line に直接追加できます。このセクションでは、SAP ABAP Application Server クライアント接続と XSL スタイル・シート動作に使用可能な構成パラメーターをリストします。ランタイム名を括弧で囲んで示します。

パラメーター

ここに記載されているパラメーターを使用することで、SAP ABAP Application Server ユーザー・レジストリー・コネクターを構成できます。

ABAP AS クライアント (client)
SAP 接続のための SAP ABAP AS ログオン・クライアント (例えば *100*)。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS ユーザー (user)
SAP 接続のための SAP ABAP AS ログオン・ユーザー。これは RFC 関数コンポーネントに直接渡されます。

パスワード (passwd)
SAP 接続のための SAP ABAP AS ログオン・パスワード。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS システム番号 (sysnr)
SAP 接続のための SAP ABAP AS システム番号 (*100* など)。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS ホスト名 (ashost)
SAP 接続のための SAP ABAP Application Server名。これは RFC 関数コンポーネントに直接渡されます。

ゲートウェイ・ホスト (gwhost)
SAP 接続のためのゲートウェイ・ホスト名。これは RFC 関数コンポーネントに直接渡されます。

RFC トレース (trace)
RFC API トレースを使用可能にするには、*1* に設定します。使用可能にすると、 SAP RFC API は個別の `rfc_nnnn.trc` ファイルを IBM Security Directory Integrator の作業ディレクトリーに作成します。このオプションは、RFC 呼び出しの問題を診断する際に役立ちます。このオプションを設定すると、コネクターと SAP ABAP Application Server との間のアクティビティーとデータがログに書き込まれます。実動デプロイメントの場合はゼロ (*0*) に設定してください。

オプション RFC 接続パラメーター
他のオプション RFC 接続パラメーターのリストを定義する際に使用しま

す。この構成リストの値は、各接続パラメーターをスペース文字で区切った key=value 形式のリストです。例えば、以下に示すストリング値により、SAP Gateway Service が「sapgw00」に設定され、SAP GUI が使用可能になります。

```
"gwserv=sapgw00 use_sapgui=1"
```

利用可能なオプション SAP Java コネクター・パラメーターを以下に示します。

- ユーザー名の別名 (alias_user)
- SAP メッセージ・サーバー (mshost)
- ゲートウェイ・サービス (gwserv)
- ログオン言語 (lang)
- RFC トレース: 1 (有効) または 0 (無効) (trace)
- SAP 表記の初期コード・ページ (codepage)
- セキュア・ネットワーク接続 (SNC) モード: 0 (オフ) または 1 (オン) (snc_mode)
- SNC パートナー。例えば、p:CN=R3、O=XYZ-INC、C=EN (snc_partnername)。
- SNC セキュリティー・レベル: 1 から 9 (snc_qop)
- SNC 名。デフォルト SNC パートナーをオーバーライドする (snc_myname)
- SNC サービスを提供するライブラリーのパス (snc_lib)
- SAP R/3 名 (r3name)
- SAP アプリケーション・サーバー・グループ (group)
- 外部サーバー・プログラムのプログラム ID (tpname)
- 外部サーバー・プログラムのホスト (tphost)
- リモート・ホストのタイプ: 2 = R/2、3 = R/3、E = 外部 (type)
- ABAP デバッグの有効化: 0 または 1 (abap_debug)
- リモート SAP グラフィカル・ユーザー・インターフェースの使用 (0/1/2) (use_sapgui)
- ログオン後の SSO チケットの取得/取得不可 (1 または 0) (getsso2)
- 指定の SAP Cookie バージョン 2 をログオン・チケットとして使用する (mysapsso2)
- 指定の X509 証明書をログオン・チケットとして使用する (x509cert)
- オープン時のログオン・チェックの有効化/無効化: 1 (有効) または 0 (無効) (lcheck)
- 32 ビット Windows での SAPLOGON の定義ストリング (saplogon_id)
- 外部認証 (PAS) のデータ (extiddata)
- 外部認証 (PAS) のタイプ (extidtype)
- 接続のアイドル・タイムアウト (秒) で、この時間を経過すると R/3 により接続が閉じられます。 正の値のみが許可されます (idle_timeout)。
- dsr サポートの有効化 (1) または無効化 (0) (dsr)

RFC 関数コンポーネント名 (sapr3.userconn.rfcFC)

IBM Security Directory Integrator に登録されている RFC 関数コンポーネントの名前。このオプションは、IBM サポートの指示があった場合にのみ変更してください。デフォルト値は以下のとおりです。

```
ibmdi.SapR3RfcFC
```

追加モード・スタイルシート (sapr3.userconn.putStylesheets)

AddOnly モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapUserXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

```
xsl/bapi_user_create.xsl, xsl/bapi_user_actgroups_assign.xsl,
   xsl/bapi_user_profiles_assign.xsl
```

更新モード・スタイルシート (sapr3.userconn.modifyStylesheets)

更新モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapUserXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルトの XSL リストは以下のとおりです。

```
xsl/bapi_user_change.xsl, xsl/bapi_user_actgroups_assign.xsl,
   xsl/bapi_user_profiles_assign.xsl
```

削除モード・スタイルシート (sapr3.userconn.deleteStylesheets)

削除モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapUserXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

```
xsl/bapi_user_delete.xsl
```

ルックアップ・モード事前スタイルシート (sapr3.userconn.findPreStylesheet)

特定のユーザーのユーザー属性をすべて取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**更新**、**削除**、および **ルックアップ**の各モードでデプロイするときに、この構成値を設定する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

```
xsl/bapi_user_getdetail_precall.xsl
```

ルックアップ・モード事後スタイルシート (sapr3.userconn.findPostStylesheet)

コネクターからのユーザー XML フォーマット応答の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**更新**、**削除**、および **ルックアップ**の各モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**ルックアップ・モード事前ス**

タイルシート構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

xsl/bapi_user_getdetail_postcall.xsl

項目選択の事前スタイルシート (sapr3.userconn.selectEntriesPreStylesheet)

SAP からすべてのユーザー名を取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

xsl/bapi_user_getlist_precall.xsl

項目選択の事後スタイルシート (sapr3.userconn.selectEntriesPostStylesheet)

getNextEntry() 処理のユーザー XML の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**項目選択の事前スタイルシート**構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

xsl/bapi_user_getlist_postcall.xsl

イテレーター・モード事前スタイルシート (sapr3.userconn.getNextPreStylesheet)

特定のユーザーのユーザー属性をすべて取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

xsl/bapi_user_getdetail_precall.xsl

イテレーター・モード事後スタイルシート (sapr3.userconn.getNextPostStylesheet)

コネクターからのユーザー XML フォーマット応答の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**イテレーター・モード事前スタイルシート**構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。この構成パラメーターは、IBM サポートの指示がある場合にのみ変更してください。デフォルト値は以下のとおりです。

xsl/bapi_user_getdetail_postcall.xsl

詳細ログ

これをチェックすると、追加のログ・メッセージが生成されます。このオプションを使用可能に設定すると、コネクターがデータとアクティビティーをログに記録します。

SAP ABAP Application Server ユーザー・レジストリー・コネクターの使用

ここに記載されている方法によって、SAP ABAP Application Server ユーザー・レジストリー・コネクターを使用できます。

このセクションでは、IBM Security Directory Integrator コネクターの各モードでのこのコネクターの使用法を説明します。またこのセクションでは、コネクターによりサポートされる IBM Security Directory Integrator 項目スキーマについても説明します。

注: デフォルトの XSL スタイルシート・ファイル名の値は、IBM Security Directory Integrator AssemblyLine 実行ディレクトリーを起点にした相対パスです。場合によっては、デフォルト・ファイル名値の前に XSL ファイルのインストール場所の完全修飾名を追加する必要があります。このような変更は、IBM Security Directory Integrator Component Suite for SAP ABAP Application Server のインストール先 (または AssemblyLine の実行元) が、インストールされている IBM Security Directory Integrator とは別のディレクトリーである場合に行うことがあります。

IBM Security Directory Integrator 項目スキーマ

ここに記載されているスキーマを使用することで、ユーザー・レジストリー・コネクターについて知ることができます。

User Registry Connector では、2 つの固定 IBM Security Directory Integrator 項目属性のみがサポートされています。スキーマを使用するには、IBM Security Directory Integrator 構成ツールの **スキーマ・ディスカバー**機能 (「**接続**」ボタン) を使用します。属性スキーマを以下に説明します。

表 52. IBM Security Directory Integrator スキーマ

属性名	タイプ	説明
sapUserXml	java.lang.String	SAP ユーザーの属性を表すストリング。 XSchema の定義については、 632 ページの『ユーザー・レジストリー・コネクター XML の XSchema』を参照してください。 コネクターを **AddOnly**、**更新**、および**削除**モードでデプロイしている場合は、この属性と値が**出力マップ**に含まれている必要があります。 コネクターを**ルックアップ**・モードおよび**イテレーター**・モードでデプロイするときには、この属性と値が**入力マップ**で使用可能です。
sapUserName	java.lang.String	特定の SAP ユーザーの名前を表すストリング。コネクターでは、**リンク基準**を構成する目的でこの属性がサポートされています。

AddOnly モード

SAP ABAP Application Server ユーザー・レジストリー・コネクターを AddOnly モードで使用すると、SAP データベース内に新規ユーザーを作成できます。

このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**により、**sapUserXml** コネクター属性のマッピングが定義される必要があります。この属性の値は、SAP に追加されるユーザーの詳細を表します。この値は、構成されている各 XSLT ファイルに定義順に適用されます。XSLT 変換により生成される各 RFC XML 要求は、RFC 関数コンポーネントにより実行されます。これは、コネクターにより内部管理されています。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

更新モード

SAP ABAP Application Server ユーザー・レジストリー・コネクターを更新モードで使用すると、SAP データベース内の既存のユーザーを変更できます。

このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**により、**sapUserXml** コネクター属性のマッピングが定義される必要があります。この属性の値は、SAP で変更されるユーザーの詳細を表します。この値は、構成されている各 XSLT ファイルに定義順に適用されます。XSLT 変換により生成される各 RFC XML 要求は、RFC 関数コンポーネントにより実行されます。これは、コネクターにより内部管理されています。

また、コネクターの**リンク基準**で **sapUserName** 属性を定義する必要があります。AssemblyLine はコネクターの **findEntry()** メソッドを呼び出して指定のユーザーの存在を確認するため、AssemblyLine には**リンク基準**が必要です。**リンク基準**で定義されている **sapUserName** の値は、**sapUserXml** の値の中の `<sapUserName>` XML エレメント値に一致している必要があります。**リンク基準**で定義されるパラメーターはすべて、XSLT スタイル・シート・パラメーターとして渡されます。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。スタイル・シートではこのパラメーターを使用する必要はありません。

リンク基準でサポートされている唯一の演算子は、**完全一致等号**です。ワイルドカードを使用した検索条件はサポートされていません。これは、RFC lookup メソッドでは現在ワイルドカードがサポートされていないためです。コネクターは重複項目を戻しません。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

注: このモードでは、役割およびプロファイルの割り当てを変更できます。コネクターに提供される XML に **sapRoleList** または **sapProfileList** が含まれていると、コネクターは SAP での現行割り当てを完全に削除および置換します。つまり、提供される XML には、操作実行後に存在している必要がある完全な割り当てが記述されている必要があります。これは、役割に関連する日付範囲についても適用されます。既存の割り当てを追加または削除するのではなく、既に割り当てられている役割の日付範囲を変更することを目的としている場合は、役割への新規日付範囲の割り当ての完全なリストを XML に指定する必要があります。SAP のデフォルト日付値が受け入れられない場合は、すべての役割に日付範囲が設定されている必要があります。

削除モード

SAP ABAP Application Server ユーザー・レジストリー・コネクターを削除モードで使用すると、SAP データベースから既存のユーザーを削除できます。

このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。コネクターの**リンク基準**で **sapUserName** 属性を定義する必要があります。AssemblyLine はコネクターの **findEntry()** メソッドを呼び出して指定のユーザーの存在を確認するため、AssemblyLine には**リンク基準**が必

要です。**リンク基準**で定義されるパラメーターはすべて、XSLT スタイル・シート・パラメーターとして渡されます。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。スタイル・シートではこのパラメーターを使用する必要はありません。

リンク基準でサポートされている唯一の演算子は、完全一致等号です。ワイルドカードを使用した検索条件はサポートされていません。これは、RFC lookup メソッドでは現在ワイルドカードがサポートされていないためです。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

ルックアップ・モード

SAP ABAP Application Server ユーザー・レジストリー・コネクターを使用して、特定の SAP ユーザーのすべての詳細を取得できます。

このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。コネクターの**リンク基準**で **sapUserName** 属性を定義する必要があります。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。コネクターは、属性 **sapUserXml** の XML ストリング値を取り込みます。この属性は、コネクターの**入力マップ**内の AssemblyLine で使用できます。

コネクターの **findEntry()** メソッドは、主要な実行コードです。**ルックアップ・モード事前スタイルシート**で構成されている XSLT 変換の実行結果を使用して RFC を実行し、特定のユーザーの詳細をすべて取得します。次に、**ルックアップ・モード事後スタイルシート**で構成されている XSLT 変換を使用して RFC 実行結果が変換されます。

リンク基準でサポートされている唯一の演算子は、完全一致等号です。ワイルドカードを使用した検索条件はサポートされていません。これは、RFC lookup メソッドでは現在ワイルドカードがサポートされていないためです。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターは一度に 1 つの項目のみを戻します。

イテレーター・モード

SAP ABAP Application Server ユーザー・レジストリー・コネクターを使用して、SAP データベースの各ユーザーの詳細を取得し、それらの詳細を AssemblyLine が使用できるようにすることができます。

項目選択の事前スタイルシート、**項目選択の事後スタイルシート**、**イテレーター・モード事前スタイルシート**、および **イテレーター・モード事後スタイルシート**の XSLT スタイル・シートを構成する必要があります。

このモードでデプロイされた IBM Security Directory Integrator AssemblyLine は、最初にコネクターの **selectEntries()** メソッドを呼び出し、SAP データベース内の全ユーザー名のリストを取得し、キャッシュに入れます。AssemblyLine は次にコネクターの **getNextEntry()** メソッドを呼び出します。このメソッドは、リスト内でキャッシュされている現行名を指し示すポインターを維持します。このメソッドは、こ

の名前を使用して RFC を呼び出し、ユーザーの詳細をすべて取得します。RFC 実行結果が XSLT 変換によりフォーマットされ、コネクターにより **sapUserXml** の値として設定され、戻されます。

サポートされないトランザクション操作

ここに記載されている情報を使用することで、サポートされないトランザクション操作について理解することができます。

現在、このコネクターおよび IBM Security Directory Integrator のどちらにおいても、SAP ABAP Application Server とのトランザクションはサポートされていません。このセクションでは、これに伴う既知の問題の一部について説明します。

結果として SAP と連携して書き込み操作が行われるモード (**AddOnly**、**更新**、および**削除**) でコネクターがデプロイされている場合は、操作が部分的にしか完了しないことがあります。これは、操作を完了するために、RFC 要求を生成する XSL スタイル・シートが複数必要な場合に発生します。RFC 要求の 1 つが失敗すると、その結果としてその後実行される RFC 要求が失敗することがあります。コネクターはすべての XSL 変換とその結果として生じる RFC 呼び出しを、最大限の努力原則に基づいて実行を試みます。

AddOnly を使用して SAP でユーザー・アカウントを作成するケースを考えてみましょう。1 番目のスタイル・シートでは、BAPI_USER_CREATE を求める RFC 要求が生成されます。2 番目のスタイル・シートでは、BAPI_USER_ACTGROUPS_ASSIGN を求める RFC 要求が生成されます。3 番目のスタイル・シートでは、BAPI_USER_PROFILES_ASSIGN を求める RFC 要求が生成されます。3 番目の要求が失敗すると、ユーザーは作成されますが、このユーザーにプロファイルが割り当てられないことがあります。

もう 1 つのケースとして、既に SAP に存在しているユーザーを作成しようとする場合があります。1 番目のスタイル・シートを実行した結果、BAPI_USER_CREATE が呼び出されます。この呼び出しでは ABAP アプリケーション・レベルのエラーが発生し、結果が戻されます (これは API またはインフラストラクチャーのエラーとは異なります)。コネクターはこれをログに記録します。次に、コネクターは後続のスタイル・シートと RFC 呼び出しを処理し、役割とプロファイルをユーザーに割り当てようとします。ユーザーは既に存在しているため、役割とプロファイルの割り当てが成功します。

前述のケースでは、1 番目の RFC の後にコネクターが処理を停止するか、または役割とプロファイルの割り当てを続行する (新規作成ユーザーに対し IBM Security Directory Integrator ユーザーが既に存在していることが予期される) かのいずれかになります。必要な振る舞いが、最初に発生した RFC エラーの後で停止することである場合は、IBM Security Directory Integrator AssemblyLine を追加構成することで、この要件を満たすことができます。**ルックアップ**・モードの 2 番目のコネクター・インスタンスを、**AddOnly** モードのインスタンスよりも前にデプロイします。**ルックアップ**・モードのコネクターは、作成するユーザーが存在しているかどうかにという条件に応じて、AssemblyLine を停止または続行するためのカスタム JavaScript コードを補助できます。

ABAP エラーの処理

SAP ABAP Application Server ユーザー・レジストリー・コネクターを使用して SAP の BAPI/RFC 関数を呼び出すと、コネクターのモード操作を実行できます。

場合によっては、XML 入力から BAPI/RFC 関数に渡されるデータが原因で、ABAP データ検証エラーが発生することがあります。この例としては、郵便番号の値が国/地域内で無効である場合があります。BAPI/RFC 関数は RFC の RETURN パラメーターに検証チェックの結果を入れて戻します。

このコネクターは、AssemblyLine が RFC 戻り状況を使用できるようにすることを目的として設計されています。このコネクターでは、ABAP エラー/警告が、スローする例外に解釈または変換されることはありません。コネクターは **urcAbapErrorCache** という名前のスクリプト Bean を登録します。この Bean は、すべてのコネクター・モードで登録され、コネクター・フック内でアクセスできます。この Bean は **AbapErrorCache** のインスタンスです。コネクター・フックのスクリプト・コードでは、必要に応じて臨時アクションを実行するときにこの情報を使用できます。各コネクター・メソッドの実行前にキャッシュがリセットされます。

スクリプト・コードの例を以下に示します。詳しくは、配布パッケージに含まれている Javadoc を参照してください。

```
var errs = urcAbapErrorCache.getLastErrorSet();
if (errs.size() > 0) {
  task.logmsg("********** There were ABAP Errors **********");
  for (var i = 0; i < errs.size(); ++i) {
   var errInfo = errs.get(i);
   task.logmsg("The message is: " + errInfo.getMsg());
   task.logmsg("The message number is: " + errInfo.getMsgNum().toString());
  }
}

var warns = urcAbapErrorCache.getLastWarningSet();
if (warns.size() > 0) {
  task.logmsg("********** There were ABAP Warnings *********");
  for (var i = 0; i < warns.size(); ++i) {
   var errInfo = warns.get(i);
   task.logmsg("The message is: " + errInfo.getMsg());
   task.logmsg("The message number is: " + errInfo.getMsgNum().toString());
  }
}
```

SAP ABAP Application Server Business Object Repository コネクター

ここに記載されている情報を使用することで、IBM Security Directory Integrator SAP ABAP Application Server Business Object Repository コネクターを構成および操作することができます。

SAP 人事管理モジュールには、さまざまなビジネス機能が含まれています。主な機能領域は、給与計算、勤怠管理、および一般人材マスター・データ管理におけるビジネス・ニーズに対応しています。

データの面で SAP HR システムの基本となるのが、*インフォタイプ* です。インフォタイプは、関連属性をまとめた論理グループです。SAP では、多数のデフォル

ト・インフォタイプが定義されています。これらのインフォタイプは番号範囲を使用してグループ化および識別されています。以下の表にこれらの番号範囲を示します。

表 53. インフォタイプの番号範囲

番号範囲	**HR サブモジュール**
0000 から 0999	HR マスター・データ
1000 から 1999	人事計画
2000 から 2999	勤怠管理
4000 から 4999	採用管理
9000 から 9999	カスタム拡張

このように多数のインフォタイプがあるため、SAP HR のすべての統合要件すべてに対応できる IBM Security Directory Integrator コネクターを 1 つだけ設計することは非常に困難です。SAP では Business API (BAPI) を介した HR データ・リポジトリーへのアクセスがサポートされています。BAPI は、ビジネス・オブジェクト・リポジトリー (BOR) 内のオブジェクトに接続します。これにより、汎用 BOR コネクターが設計およびインプリメントされています。このコネクターは、あらゆる BOR オブジェクトのあらゆるメソッドを呼び出すことができます。このコネクターは、特定の BOR オブジェクトにより管理されるデータの XML 表現を提示します。このコネクターを使用するには、一連の XSL スタイル・シートを構成し、特定の BOR オブジェクトのクラス ID 名を指定する必要があります (実際には XSL スタイル・シートにより XML データ表現が定義されます)。

コネクターのコンポーネント設計を以下の図に示します。

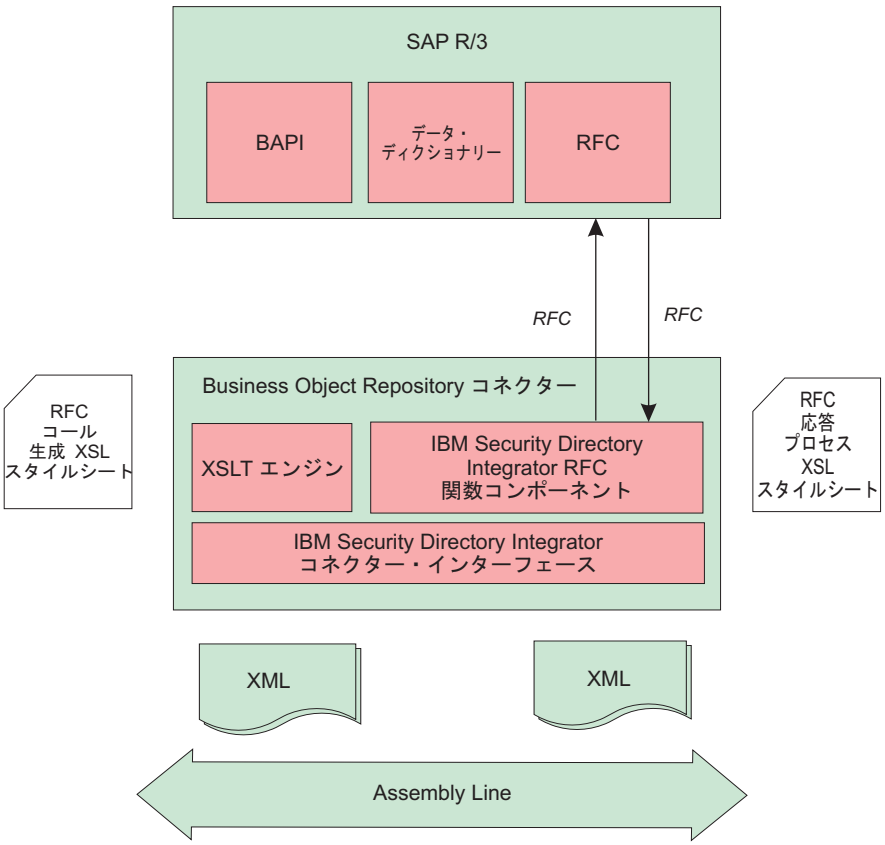


図 11. SAP ABAP Application Server Business Object Repository コネクターのコンポーネント設計

IBM Security Directory Integrator は、コネクターによる HR 個人データ情報 (インフォタイプ 0002) の管理を可能にする XSL スタイルシートのサンプル・セットを提供しています。スタイル・シートは、PERSDATA BOR オブジェクトの BAPI RFC メソッドを呼び出すようセットアップされています。このコネクターは、SAP ABAP Application Server 用の IBM Security Directory Integrator 関数コンポーネントの汎用 RFC 呼び出し機能を使用します。

このコネクターの主な機能と利点は以下のとおりです。

- SAP HR データに対する作成、読み取り、更新、および削除 (C.R.U.D) の各操作をサポートします。
- XSL 変換により SAP ABAP Application Server RFC 実行の動作を変更できます。
- このコネクターと SAP ABAP Application Server の間のコンパイル時間の依存関係が最小限に抑えられています。このコネクターは生成された RFC プロキシー・コードを使用しません。RFC 関数コンポーネントを動的プロキシーとして使用します。
- ABAP または Java のカスタム・コーディングが不要です (ただし特定の新機能をサポートするためにカスタム・コードが必要なことがあります)。

このコネクターは、以下に示す IBM Security Directory Integrator の標準コネクター・モードをサポートしていますが、各モードで機能を提供する場合は、標準の BAPI メソッドを使用します。

- AddOnly
- 更新
- 削除
- ルックアップ
- イテレーター

コネクター・モードと BAPI メソッドのマッピングの例を以下の表に示します。

表 54. マッピングの例

コネクター・モード	**BAPI メソッド**
追加	Create、CreateFromData
更新	Change
削除	削除
ルックアップ	Get、GetDetail
イテレーター	GetList、Get、GetDetailedList

キー・フィールドおよび XML 表記

BOR オブジェクトのキー・フィールドは、コネクターにより特別な方法で処理されます。これは、BOR オブジェクト・データの XML 表現に反映されます。

要求 XML と応答 XML を処理するための代替 XSL スタイル・シートを定義できますが、代替スタイル・シートでは **sapBorObjIdentifier** というエレメントがサポートされている必要があります。このエレメントは、**ルックアップ**・モードと**イテレーター**・モードで項目が戻されるときに、コネクターの Java コードにより処理されます。**sapBorObjIdentifier** は XML 内の任意の位置に記述できます。このエレメントには、特定の BOR オブジェクトのキー・フィールド名に一致する名前のタグを持つエレメントが含まれています。

HR 個人データ情報 XML の一般的なフォームを以下に示します。

```
<sapPersonalData>
 <sapBorObjIdentifier>
  <EmployeeNumber>00000001</EmployeeNumber>
  <SubType />
  <ObjectID />
  <LockIndicator />
  <ValidityEnd>99991231</ValidityEnd>
  <ValidityBegin>19740320</ValidityBegin>
  <RecordNumber>000</RecordNumber>
 </sapBorObjIdentifier>
 <personalDataDetail>
  <title>1</title>
  <firstname></firstname>
  <lastname></lastname>
  <nameAtBirth />
  <knownAs></knownAs>
  <surnamePrefix />
  <gender></gender>
  <dateOfBirth></dateOfBirth>
  <birthPlace />
  <stateOfBirth />
  <countryOfBirth />
  <maritalStatus></maritalStatus>
  <numberOfChildren></numberOfChildren>
  <religion />
  <language></language>
  <languageCode></languageCode>
```

```
  <nationality></nationality>
  <idNumber />
 </personalDataDetail>
</sapPersonalData>
```

更新モードおよび削除モードでのルックアップのスキップ

ルックアップのスキップ・オプションを使用して、検索を回避できます。

SAP ABAP Application Server Business Object Repository コネクターでは、更新モードまたは削除モードでの「**ルックアップのスキップ**」一般オプションがサポートされています。このオプションを選択した場合、実際の更新操作および削除操作よりも前に検索が行われることはありません。

このことが機能するためには、HR 個人データ情報 (インフォタイプ 0002) に対して、以下の属性がリンク基準で定義されている必要があります。

- EmployeeNumber
- ValidityBegin
- ValidityEnd

構成

ここに記載されているパラメーターを使用して、SAP ABAP Application Server Business Object Repository コネクターを構成できます。

BOR Connector for SAP ABAP Application Server (SAP ABAP AS) は、Assembly Line に直接追加できます。次のセクションでは、SAP クライアント接続と XSL スタイルシート動作に使用可能な構成パラメーターをリストします。ランタイム名を括弧で囲んで示します。

パラメーター

ABAP AS クライアント (client)
SAP 接続のための SAP ABAP AS ログオン・クライアント (例えば *100*)。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS ユーザー (user)
SAP 接続のための SAP ABAP AS ログオン・ユーザー。これは RFC 関数コンポーネントに直接渡されます。

パスワード (passwd)
SAP 接続のための SAP ABAP AS ログオン・パスワード。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS システム番号 (sysnr)
SAP 接続のための SAP ABAP AS システム番号 (*100* など)。これは RFC 関数コンポーネントに直接渡されます。

ABAP AS ホスト名 (ashost)
SAP 接続のための SAP ABAP AS アプリケーション・サーバー名。これは RFC 関数コンポーネントに直接渡されます。

ゲートウェイ・ホスト (gwhost)
SAP 接続のためのゲートウェイ・ホスト名。これは RFC 関数コンポーネントに直接渡されます。

RFC トレース (trace)

RFC API トレースを使用可能にするには、*1* に設定します。使用可能にすると、SAP RFC API は個別の `rfc_nnnn.trc` ファイルを IBM Security Directory Integrator の作業ディレクトリーに作成します。このオプションは、RFC 呼び出しの問題を診断する際に役立ちます。このオプションを設定すると、コネクターと SAP ABAP AS との間のアクティビティーとデータがログに書き込まれます。実動デプロイメントの場合はゼロ (*0*) に設定してください。

オプション RFC 接続パラメーター

他のオプション RFC 接続パラメーターのリストを定義する際に使用します。この構成リストの値は、各接続パラメーターをスペース文字で区切った key=value 形式のリストです。例えば、以下に示すストリング値により、SAP Gateway Service が「sapgw00」に設定され、SAP GUI が使用可能になります。

```
"gwserv=sapgw00 use_sapgui=1"
```

利用可能なオプション SAP Java コネクター・パラメーターを以下に示します。

- ユーザー名の別名 (alias_user)
- SAP メッセージ・サーバー (mshost)
- ゲートウェイ・サービス (gwserv)
- ログオン言語 (lang)
- RFC トレース: 1 (有効) または 0 (無効) (trace)
- SAP 表記の初期コード・ページ (codepage)
- セキュア・ネットワーク接続 (SNC) モード: 0 (オフ) または 1 (オン) (snc_mode)
- SNC パートナー。例えば、p:CN=R3、O=XYZ-INC、C=EN (snc_partnername)。
- SNC セキュリティー・レベル: 1 から 9 (snc_qop)
- SNC 名。デフォルト SNC パートナーをオーバーライドする (snc_myname)
- SNC サービスを提供するライブラリーのパス (snc_lib)
- SAP R/3 名 (r3name)
- SAP アプリケーション・サーバー・グループ (group)
- 外部サーバー・プログラムのプログラム ID (tpname)
- 外部サーバー・プログラムのホスト (tphost)
- リモート・ホストのタイプ: 2 = R/2、3 = R/3、E = 外部 (type)
- ABAP デバッグの有効化: 0 または 1 (abap_debug)
- リモート SAP グラフィカル・ユーザー・インターフェースの使用 (0/1/2) (use_sapgui)
- ログオン後の SSO チケットの取得/取得不可 (1 または 0) (getsso2)
- 指定の SAP Cookie バージョン 2 をログオン・チケットとして使用する (mysapsso2)

- 指定の X509 証明書をログオン・チケットとして使用する (x509cert)
- オープン時のログオン・チェックの有効化/無効化: 1 (有効) または 0 (無効) (lcheck)
- 32 ビット Windows での SAPLOGON の定義ストリング (saplogon_id)
- 外部認証 (PAS) のデータ (extiddata)
- 外部認証 (PAS) のタイプ (extidtype)
- 接続のアイドル・タイムアウト (秒) で、この時間を経過すると R/3 により接続が閉じられます。 正の値のみが許可されます (idle_timeout)。
- dsr サポートの有効化 (1) または無効化 (0) (dsr)

BOR クラス名 (sapr3.conn.borObjName)
このコネクターが統合する BOR クラスの名前。BOR クラスの名前は、SAP でトランザクション BAPI を使用することで使用可能になります。この値は、スキーマ照会実行時に BOR オブジェクトのキー・フィールド名を取得するときに使用されます。

RFC 関数コンポーネント名 (sapr3.conn.rfcFC)
IBM Security Directory Integrator に登録されている RFC 関数コンポーネントの名前。このオプションは、IBM サポートの指示があった場合にのみ変更してください。デフォルト値は以下のとおりです。

```
ibmdi.SapR3RfcFC
```

追加モード・スタイルシート (sapr3.conn.putStylesheets)
AddOnly モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。

更新モード・スタイルシート (sapr3.conn.modifyStylesheets)
更新モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。

削除モード・スタイルシート (sapr3.conn.deleteStylesheets)
削除モードでのデプロイ時にコネクターにより実行される XSLT スタイルシート・ファイルのリスト。実行時には、コンテナー項目内に含まれている XML に各スタイル・シートが適用されます。XSL は**sapXml** 属性の値に適用されます。テキスト・ボックス内で、XSL スタイル・シート・ファイル名を 1 つずつ新規行に入力する必要があります。

ルックアップ・モード事前スタイルシート (sapr3.conn.findPreStylesheet)
特定のユーザーのユーザー属性をすべて取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**更新**、**削除**、および **ルックアップ**の各モードでデプロイするときに、この構成値を設定する必要があります。

ルックアップ・モード事後スタイルシート (sapr3.conn.findPostStylesheet)
コネクターからのユーザー XML フォーマット応答の作成時にコネクター

により実行される XSLT スタイル・シート・ファイル。コネクターを**更新**、**削除**、および **ルックアップ**の各モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**ルックアップ・モード事前スタイルシート**構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。

項目選択の事前スタイルシート (sapr3.conn.selectEntriesPreStylesheet)
SAP からすべてのユーザー名を取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。

項目選択の事後スタイルシート (sapr3.conn.selectEntriesPostStylesheet)
getNextEntry() 処理のユーザー XML の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**項目選択の事前スタイルシート**構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。

イテレーター・モード事前スタイルシート (sapr3.conn.getNextPreStylesheet)
特定のユーザーのユーザー属性をすべて取得可能な RFC XML 要求の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。

イテレーター・モード事後スタイルシート (sapr3.conn.getNextPostStylesheet)
コネクターからのユーザー XML フォーマット応答の作成時にコネクターにより実行される XSLT スタイル・シート・ファイル。コネクターを**イテレーター**・モードでデプロイするときに、この構成値を設定する必要があります。XSLT では、**イテレーター・モード事前スタイルシート**構成からの XSLT 実行結果として、実行された RFC からの応答 XML が変換されます。

詳細ログ
これをチェックすると、追加のログ・メッセージが生成されます。このオプションを使用可能に設定すると、コネクターがデータとアクティビティーをログに記録します。

SAP ABAP Application Server Business Object Repository コネクターの使用

ここに記載されている情報を使用して、SAP ABAP Application Server Business Object Repository コネクターを操作できます。

このセクションでは、IBM Security Directory Integrator でサポートされている各コネクター・モードでこのコネクターを使用する方法について詳しく説明します。またこのセクションでは、コネクターによりサポートされる IBM Security Directory Integrator 項目スキーマについても説明します。

注: デフォルトの XSL スタイルシート・ファイル名の値は、IBM Security Directory Integrator AssemblyLine 実行ディレクトリーを起点にした相対パスです。場合によっては、デフォルト・ファイル名値の前に XSL ファイルのインストール場所の完全

修飾名を追加する必要があります。このような変更は、IBM Security Directory Integrator Component Suite for SAP ABAP Application Server のインストール先 (または AssemblyLine の実行元) が、インストールされている IBM Security Directory Integrator とは別のディレクトリーである場合に行うことがあります。

IBM Security Directory Integrator 項目スキーマ

IBM Security Directory Integrator 項目スキーマを使用して、項目スキーマ属性の詳細を理解することができます。

BOR コネクターでは、ネイティブ属性 **sapXml** がサポートされています。**sapXml** の値は、BOR オブジェクトの属性を表す XML ストリング表現です。その他の属性は、特定の BOR オブジェクトのキー・フィールド名を反映します。これらは、コネクターが**ルックアップ**、**削除**、または**更新**モードでデプロイされている場合に、IBM Security Directory Integrator **リンク基準**を定義できるようにサポートされています。

スキーマは、IBM Security Directory Integrator 構成ツールのスキーマ照会機能で使用可能です。属性スキーマを以下に説明します。

表 55. 項目スキーマ属性

属性名	タイプ	説明
sapXml	java.lang.String	SAP BOR オブジェクトの属性を表すストリング。コネクターが **AddOnly** モードまたは**更新**モードでデプロイされている場合は、この属性と値が**出力マップ**に存在している必要があります。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
EmployeeNumber	java.lang.String	個人データ情報インフォタイプ 0002 固有です。8 桁の従業員番号です。コネクターが **ルックアップ**、**更新**、および**削除**モードでデプロイされている場合は、この属性と値が**リンク基準**に含まれている必要があります。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
Subtype	java.lang.String	個人データ情報インフォタイプ 0002 固有です。4 文字の個人データ・サブタイプです。コネクターが **ルックアップ**、**更新**、および**削除**モードでデプロイされている場合は、この属性と値が**リンク基準**に含まれている必要があります。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
ObjectID	java.lang.String	個人データ情報インフォタイプ 0002 固有です。他のすべてのキー・フィールドが同一であるインフォタイプのオブジェクト ID (2 文字) です。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
LockIndicator	java.lang.String	個人データ情報インフォタイプ 0002 固有です。SAP でマスター・データ・レコードがロックされているかどうかを示す 1 文字のフラグです。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。

表 55. 項目スキーマ属性 (続き)

属性名	タイプ	説明
ValidityEnd	java.lang.String	個人データ情報インフォタイプ 0002 固有です。8 桁の日付値 (YYYYMMDD)。コネクターが **ルックアップ**、**更新**、および**削除**モードでデプロイされている場合は、この属性と値が**リンク基準**に含まれている必要があります。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
ValidityBegin	java.lang.String	個人データ情報インフォタイプ 0002 固有です。8 桁の日付値 (*YYYYMMDD*)。コネクターが **ルックアップ**、**更新**、および**削除**モードでデプロイされている場合は、この属性と値が**リンク基準**に含まれている必要があります。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。
RecordNumber	java.lang.String	個人データ情報インフォタイプ 0002 固有です。2 桁の値。コネクターが**ルックアップ**および**イテレーター**・モードでデプロイされている場合は、この属性は**入力マップ**で使用可能です。

AddOnly モード

Add Only モードでコネクターを使用して、SAP データベースに新規オブジェクトを作成することができます。

コネクターを IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**で、**sapXml** コネクター属性のマッピングを定義する必要があります。この属性の値は、SAP に追加されるオブジェクトの詳細を表します。この値は、構成されている各 XSLT ファイルに定義順に適用されます。XSLT 変換により生成される各 RFC XML 要求は、RFC 関数コンポーネントにより実行されます。これは、コネクターにより内部管理されています。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

HR 個人データ情報 (インフォタイプ 0002) の場合は、有効な従業員番号が存在している必要があります。一般的な XML フォームを以下に示します。必須エレメントは **EmployeeNumber**、**ValidityBegin**、および **ValidityEnd** です。

```
<sapPersonalData>
 <sapBorObjIdentifier>
  <EmployeeNumber>00000001</EmployeeNumber>
  <SubType />
  <ObjectID />
  <LockIndicator />
  <ValidityEnd>99991231</ValidityEnd>
  <ValidityBegin>19740320</ValidityBegin>
  <RecordNumber>000</RecordNumber>
 </sapBorObjIdentifier>
 <personalDataDetail>
  <title></title>
  <firstname></firstname>
  <lastname></lastname>
  <nameAtBirth />
  <knownAs>Torpedo</knownAs>
  <surnamePrefix />
  <gender>1</gender>
  <dateOfBirth></dateOfBirth>
  <birthPlace />
```

```
  <stateOfBirth />
  <countryOfBirth />
  <maritalStatus></maritalStatus>
  <numberOfChildren></numberOfChildren>
  <religion />
  <language></language>
  <languageCode></languageCode>
  <nationality></nationality>
  <idNumber />
 </personalDataDetail>
</sapPersonalData>
```

更新モード

コネクターを更新モードで使用して、SAP データベースの既存のオブジェクトを変更できます。

更新モードでデプロイされたコネクターは、データベース内の既存のデータを変更することができます。このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**出力マップ**で、**sapXml** コネクター属性のマッピングを定義する必要があります。この属性の値は、SAP で変更されるユーザーの詳細を表します。この値は、構成されている各 XSLT ファイルに定義順に適用されます。XSLT 変換により生成される各 RFC XML 要求は、RFC 関数コンポーネントにより実行されます。これは、コネクターにより内部管理されています。

また、コネクターの**リンク基準**に BOR オブジェクトの一部のキー・フィールドが必要です。AssemblyLine は、コネクターの **findEntry()** メソッドを呼び出して特定のオブジェクトの存在を確認するため、AssemblyLine は**リンク基準**を必要とします。**リンク基準**で定義されるパラメーターはすべて、XSLT スタイル・シート・パラメーターとして渡されます。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

HR 個人データ情報 (インフォタイプ 0002) の場合は、以下の属性が**リンク基準**で定義されている必要があります。

- EmployeeNumber
- ValidityBegin
- ValidityEnd

これらの属性は XSL スタイル・シートにパラメーターとして渡されるため、XML では必須ではありません。一般的な XML フォームを以下に示します。

```
<sapPersonalData>
 <sapBorObjIdentifier>
  <SubType />
  <ObjectID />
  <LockIndicator />
  <RecordNumber>000</RecordNumber>
 </sapBorObjIdentifier>
 <personalDataDetail>
  <title></title>
  <firstname></firstname>
  <lastname></lastname>
  <nameAtBirth />
  <knownAs>Torpedo</knownAs>
  <surnamePrefix />
  <gender></gender>
  <dateOfBirth></dateOfBirth>
  <birthPlace />
```

```
  <stateOfBirth />
  <countryOfBirth />
  <maritalStatus></maritalStatus>
  <numberOfChildren></numberOfChildren>
  <religion />
  <language></language>
  <languageCode></languageCode>
  <nationality></nationality>
  <idNumber />
 </personalDataDetail>
</sapPersonalData>
```

削除モード

コネクターを削除モードで使用することで、SAP データベースから既存のオブジェクトを削除できます。

このコネクターは、IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。**削除**モードでは、コネクターは**リンク基準**のみに依存します。**リンク基準**で定義されるパラメーターはすべて、XSLT スタイル・シート・パラメーターとして渡されます。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターには一度に 1 つの項目のみを指定できます。

HR 個人データ情報 (インフォタイプ 0002) の場合は、以下の属性が**リンク基準**で定義されている必要があります。

- EmployeeNumber
- ValidityBegin
- ValidityEnd

ルックアップ・モード

コネクターをルックアップ・モードで使用して、特定の SAP オブジェクトの詳細をすべて取得できます。

コネクターを IBM Security Directory Integrator AssemblyLine の **Flow** セクションに追加する必要があります。コネクターの**リンク基準**でコネクターのキー・フィールド属性を定義する必要があります。重複する**リンク基準**名が指定されると、コネクターは最後に指定された値を使用します。コネクターは、「**sapXml**」属性の XML ストリング値を取り込み、コネクターの**入力マップ**で AssemblyLine がこの属性を使用できるようにします。キー・フィールドの名前と値も**入力マップ**で使用可能になります。

コネクターの **findEntry()** メソッドは、主要な実行コードです。**ルックアップ・モード事前スタイルシート**で構成されている XSLT 変換の実行結果を使用して RFC を実行し、特定のユーザーの詳細をすべて取得します。次に、**ルックアップ・モード事後スタイルシート**で構成されている XSLT 変換を使用して RFC 実行結果が変換されます。

このコネクターでは、重複項目または複数項目はサポートされていません。コネクターは一度に 1 つの項目のみを戻します。

HR 個人データ情報 (インフォタイプ 0002) の場合は、以下の属性が**リンク基準**で定義されている必要があります。

- EmployeeNumber
- ValidityBegin
- ValidityEnd

イテレーター・モード

コネクターをイテレーター・モードで使用して、SAP データベースの各オブジェクトの詳細を順番に取得し、AssemblyLine がこれらの詳細を使用できるようにすることができます。

項目選択の事前スタイルシート、**項目選択の事後スタイルシート**、**イテレーター・モード事前スタイルシート**、および **イテレーター・モード事後スタイルシート**の XSLT スタイル・シートを構成する必要があります。

このモードでデプロイされた IBM Security Directory Integrator AssemblyLine は、最初にコネクターの **selectEntries()** メソッドを呼び出して、SAP データベース内の (特定の BOR オブジェクトの) すべてのキー・フィールドの名前と値のリストを取得し、キャッシュに格納します。AssemblyLine は次にコネクターの **getNextEntry()** メソッドを呼び出します。このメソッドは、リスト内でキャッシュされている現行キー・フィールドを指し示すポインターを維持します。このメソッドはこのキー・フィールド情報を使用して RFC を呼び出し、オブジェクトのすべての詳細を取得します。RFC 実行結果は XSLT 変換によりフォーマットされ、**sapXml** の値として設定され、コネクターにより戻されます。キー・フィールドの名前と値も**入力マップ**で使用可能になります。

サポートされないトランザクション操作

ここに記載されている情報を使用することで、サポートされないトランザクション操作について理解することができます。

結果として SAP での書き込み操作が行われるモード (**AddOnly**、**更新**、および**削除**) でコネクターがデプロイされている場合は、操作が部分的にしか完了しないことがあります。これは、操作を完了するために、RFC 要求を生成する XSL スタイル・シートが複数必要な場合に発生します。RFC 要求の 1 つが失敗すると、その結果としてその後実行される RFC 要求が失敗することがあります。

ABAP エラーの処理

SAP ABAP Application Server Business Object Registry コネクターを使用して、SAP の BAPI/RFC 関数を呼び出し、コネクター・モード操作を実行することができます。

場合によっては、XML 入力から BAPI/RFC 関数に渡されるデータが原因で、ABAP データ検証エラーが発生することがあります。BAPI/RFC 関数は RFC の「RETURN」パラメーターに検証チェックの結果を入れて戻します。

このコネクターは、AssemblyLine が RFC 戻り状況を使用できるようにすることを目的として設計されています。このコネクターでは、ABAP エラー/警告が、スローする例外に解釈または変換されることはありません。コネクターは **borcAbapErrorCache** という名前のスクリプト Bean を登録します。この Bean は、すべてのコネクター・モードで登録され、コネクター・フック内でアクセスできます。この Bean は **AbapErrorCache** のインスタンスです。コネクター・フック

のスクリプト・コードでは、必要に応じて臨時アクションを実行するときにこの情報を使用できます。各コネクター・メソッドの実行前にキャッシュがリセットされます。

スクリプト・コードの例を以下に示します。詳しくは、配布パッケージに含まれている Javadoc を参照してください。

```
var errs = borcAbapErrorCache.getLastErrorSet();
if (errs.size() > 0) {
  task.logmsg("********** There were ABAP Errors *********");
  for (var i = 0; i < errs.size(); ++i) {
   var errInfo = errs.get(i);
   task.logmsg("The message is: " + errInfo.getMsg());
   task.logmsg("The message number is: " + errInfo.getMsgNum().toString());
  }
}

var warns = borcAbapErrorCache.getLastWarningSet();
if (warns.size() > 0) {
  task.logmsg("********** There were ABAP Warnings *********");
  for (var i = 0; i < warns.size(); ++i) {
   var errInfo = warns.get(i);
   task.logmsg("The message is: " + errInfo.getMsg());
   task.logmsg("The message number is: " + errInfo.getMsgNum().toString());
  }
}
```

ALE Intermediate Document (IDOC) Connector for SAP ABAP Application Server および SAP ERP

ここに記載されている情報を使用することで、SAP ABAP Application Server または ERP システムから送信された ALE IDOC を処理するための IBM Security Directory Integrator コネクターを構成および操作することができます。

SAP システムでは、Application Link Enabling (ALE) は中核となる統合テクノロジーの 1 つです。このテクノロジーは、Intermediate Document (IDOC) として知られている階層データ文書の交換に関わるものです。SAP へのインバウンドと SAP へのアウトバウンドという 2 つのシナリオがあります。このコネクターのこのリリースは、SAP からのアウトバウンドである IDOC と、IBM Security Directory Integrator へのインバウンドである IDOC とのみ統合されています。この文書では、IBM Security Directory Integrator へのインバンドについて、インバウンドという用語を使用します。IBM Security Directory Integrator が IDOC サーバーとして稼働している場合、SAP システムは常に IDOC クライアントになります。IDOC は非同期イベントとして IBM Security Directory Integrator に送信され、IDOC を受信した IBM Security Directory Integrator は、目的の処理を行うために IDOC データを AssemblyLine にプッシュします。これは非同期通信であるため、IBM Security Directory Integrator はクライアントである SAP システムには応答を返しません。SAP システム・クライアントと IBM Security Directory Integrator との間のデータの整合性を保証するために、SAP TID 管理機能が使用されます。非同期通信のため、このコネクターは、イテレーター・モードのみをサポートしています。

SAP システムに ALE を構成する場合の中心となるタスクは、分散モデルと呼ばれるものの作成です。SAP システムには標準として使用できる定義済みの分散モデルが数多くありますが、カスタマイズ可能な独自の分散モデルを作成するための機能もあります。このコネクターは主に、選択した SAP 分散モデル内で論理システムとして振る舞う外部アプリケーションとして使用されます。このセクションで示している例では、カスタムの SAP HR 分散モデル、および定義済みの SAP 集中ユー

ザー管理 (CUA) 分散モデルへの統合を定義します。もちろんこのコネクターを使用して、他の任意の SAP 分散モデルに統合し、SAP FI/CO や SAP PP などの他の SAP モジュールのマスター・データにアクセスすることができます。 SAP モジュールについて詳しくは、SAP のヘルプ・サイト (http://help.sap.com) にアクセスしてください。IDOC インターフェースを持つほとんどすべての SAP マスター・データ・ビジネス・オブジェクトは、この方法で交換できます。

SAP 分散モデルの作成または構成の中心となるのは、サポートする IDOC メッセージ・タイプです。このコネクターが提供するのは、IDOC の XML バージョンであり、これは適宜に解析する必要があります。IDOC XML データの解析を容易にするために、このコネクターは IBM Security Directory Integrator の XML パーサーを使用できるようになっています。DOM パーサー、SAX パーサー、または XSLT パーサーから選択できます。これらを使用すると、ビジネス目的に必要なデータを IDOC から抽出することができます。例えば、SAP HR IDOC メッセージ・タイプ HRMD_A から特定のインフォタイプを抽出し、自動プロビジョニングの目的で IBM Security Directory Integrator に送信することができます。

以下の図は、IDOC クライアントである SAP システムと IDOC サーバーである IBM Security Directory Integrator との対話を示したものです。

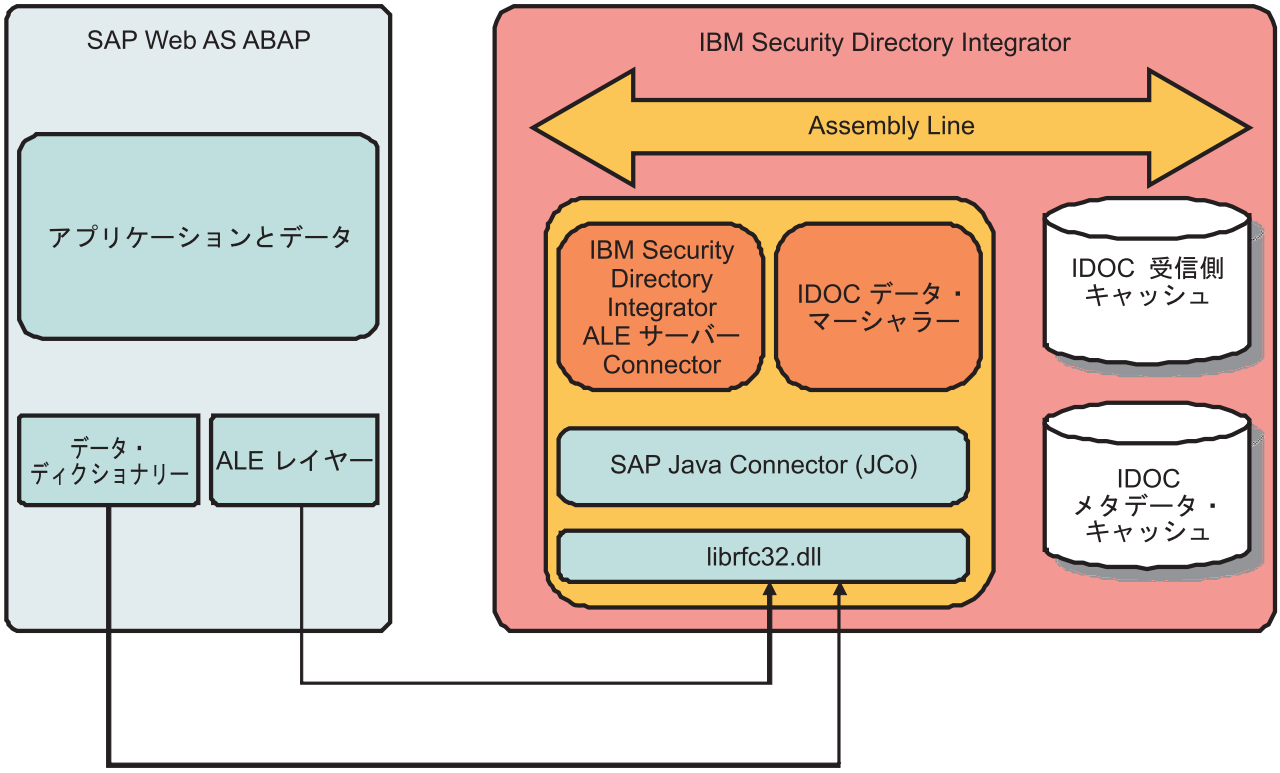


図 12. IDOC クライアントである SAP システムと IDOC サーバーである IBM Security Directory Integrator との対話

インストール

ここに記載されている情報を使用することで、SAP ALE IDOC コネクターをインストールできます。

SAP ALE IDOC Connector for SAP ABAP Application Server は、IBM Security Directory Integrator インストール・パッケージの一部です。ただし、このコネクターが正しく動作するためには、いくつかの SAP クラス・ライブラリーを取得して、`sapjco.jar` ファイルと一緒にインストールする必要があります。これについては、Component Suite 全体に対するインストール手順で概説されています。

- sapidoc.jar
- sapidocjco.jar

構成

ここに記載されているセクション・リストを使用してコネクターを構成できます。

SAP ALE IDOC Connector for SAP ABAP Application Server は Assembly Line に直接追加できます。

IDOC Server パラメーター

ここに記載されているパラメーターを使用することで、ALE IDOC コネクターを構成できるようになります。

IDOC Server SAP Gateway Host
必須の RFC サーバー接続属性。SAP Gateway であるホスト・システムを定義します。

IDOC Server SAP Gateway Service
必須の RFC サーバー接続属性。SAP Gateway Service を定義します。

IDOC Server Program ID
必須の RFC サーバー接続属性。サーバーを SAP Gateway に登録するために、必要な TCP/IP RFC 宛先の構成で使用されるサーバーのプログラム ID を定義します。

IDOC Server Unicode 接続かどうか
オプションの RFC サーバー接続属性。SAP Gateway であるホスト・システムを定義します。

IDOC Server のオプション接続パラメーター
オプションの RFC サーバー接続属性。他のオプション RFC 接続パラメーターのリストを定義するために使用されます。この構成リストの値は、各接続パラメーターをスペース文字で区切った key=value 形式のリストです。例えば、以下のストリング値は、SAP システム番号を「00」に設定し、RFC トレース・メカニズムを使用可能にします。

```
"jco.server.trace=1 jco.server.sysnr=00"
```

IDOC Client の構成パラメーター

ここに記載されている IDOC Client の構成パラメーターを使用して、IDOC クライアントを構成できます。

IDOC Client 番号
SAP システム・クライアントを定義する必須の RFC クライアント接続属性。ログオン・クライアントを定義する 3 桁のストリング値 (例えば、「000」または「100」) で構成されます。

IDOC Client ユーザー
SAP ユーザー・アカウントのログオン ID を定義する、必須の RFC クライアント接続属性。一般にこれは、SAP ユーザー・アカウント・タイプの通信または CPIC です。ただし、ダイアログ・タイプを使用することは可能です。

IDOC Client パスワード
SAP ユーザー・アカウントのパスワードを定義する必須の RFC クライアント接続属性。

IDOC Client Lang
ログオン言語を定義する必須の RFC クライアント接続属性。

IDOC Client ホスト名
ターゲットの SAP システムのホスト名を定義する必須の RFC クライアント接続属性。

IDOC Client システム番号
ターゲットの SAP システムのシステム ID (SID) を定義する必須の RFC クライアント接続属性。

IDOC Client SAP Gateway Service
SAP Gateway Service を定義する、オプションの RFC クライアント接続属性。ほとんどの場合、この値は「IDOC Server SAP Gateway Service」の値と同じです。

IDOC Client SAP Gateway Host
SAP Gateway のホスト名を定義する、オプションの RFC クライアント接続属性。

IDOC Client 最大接続数
内部接続プールでサポートされる RFC 接続の最大数を定義する必須の RFC クライアント接続属性。

IDOC Client のオプション接続パラメーター
他のオプション RFC 接続パラメーターのリストを定義するために使用される、オプションの RFC クライアント接続属性。この構成リストの値は、各接続パラメーターをスペース文字で区切った key=value 形式のリストです。例えば以下のストリング値は、RFC トレース・メカニズムをオンにし、SAP GUI が IBM Security Directory Integrator と同じホストにインストールされている場合に、その SAP GUI を使用可能にします。

```
"jco.client.trace=1 jco.client.use_sapgui=1"
```

アクセス可能な、SAP Java コネクターのオプション・パラメーターを以下に示します。

- ユーザー名の別名 [jco.client.alias_user]
- SAP メッセージ・サーバー [jco.client.mshost]
- RFC トレース [jco.client.trace]

- SAP 表記の初期コード・ページ [jco.client.codepage]
- セキュア・ネットワーク接続 (SNC) モード: 0 (オフ) または 1 (オン) [jco.client.snc_mode]
- SNC パートナーで、例えば、p:CN=R3、O=XYZ-INC、C=EN [jco.client.snc_partnername]
- SNC セキュリティー・レベル: 1 から 9 [jco.client.snc_qop]
- SNC 名で、デフォルトの SNC パートナーを指定変更します [jco.client.snc_myname]
- SNC サービスを提供するライブラリーのパス [jco.client.snc_lib]
- SAP R/3 名 [jco.client.r3name]
- SAP アプリケーション・サーバーのグループ [jco.client.group]
- 外部サーバー・プログラムのプログラム ID [jco.client.tpname]
- 外部サーバー・プログラムのホスト [jco.client.tphost]
- リモート・ホストのタイプ: 2 = R/2、3 = R/3、E = 外部 [jco.client.type]
- ABAP デバッグの有効化: 0 または 1 [jco.client.abap_debug]
- リモートの SAP グラフィカル・ユーザー・インターフェースを使用する (0/1/2) [jco.client.use_sapgui]
- ログオン後の SSO チケットの取得/取得不可 (1 または 0) [jco.client.getsso2]
- 指定の SAP Cookie バージョン 2 をログオン・チケットとして使用する [jco.client.mysapsso2]
- 指定の X509 証明書をログオン・チケットとして使用する [jco.client.x509cert]
- オープン時のログオン・チェックの有効化/無効化: 1 (有効化) または 0 (無効化) [jco.client.lcheck]
- 32 ビット Windows での SAPLOGON の定義ストリング [jco.client.saplogon_id]
- 外部認証 (PAS) のデータ [jco.client.extiddata]
- 外部認証 (PAS) のタイプ [jco.client.extidtype]
- 接続のアイドル・タイムアウト (秒) で、この時間を経過すると R/3 により接続が閉じられます。正の値のみが許可されます。dsr サポート [jco.client.dsr] を有効化する場合は (1)、無効化する場合は (0)

一般構成パラメーター

ここに記載されているパラメーターを使用することで、コネクターを構成できるようになります。

IDOC As は XML のみ

IDOC の XML 値属性のみが必要かどうかを定義する一般属性。「いいえ」に設定した場合、IDOC 制御データは、IDOC ごとに、生成された項目内の独立した属性として設定されます。「はい」に設定した場合、IDOC には、IDOC の内容を XML 値ストリングとして含む属性 (idoc.xml) が 1 つだけ作成されます。

SAP RFM 要求を処理するかどうか
コネクターが、それに対して行われたリモート関数モジュール (RFM) 呼び出しも処理するかどうかを定義する一般属性。「はい」に設定した場合、XML 値ストリングとしての RFM の内容である属性 (rfm.xml) を 1 つ含む 1 つの項目に、すべての RFM 呼び出しが追加されます。「いいえ」に設定した場合、IDOC サーバーに対する RFM 要求は無視されます。

注: RFM 呼び出しに対して実行される処理では、単に RFM が XML 値属性として提供されるだけです。IDOC サーバーは現在、RFM 呼び出しのエクスポート引数とテーブル引数の取り込みを試行しません。それらが必要な場合は、コネクターの機能拡張で提供することができます。これを行う場合は IBM サポートに連絡してください。考慮する必要があるのは、ALE 分散モデルの内部プロセスの一部を形成する RFM 呼び出しだけです。

IDOC と RFM XML のいずれを構文解析するか
XML 値属性 idoc.xml および rfm.xml のどちらに対して構文解析を試行するのかを定義する一般属性。AssemblyLine 構成内のコネクターと対話するには、いずれかの使用可能な IBM Security Directory Integrator パーサーを構成する必要があります。

JCo ミドルウェア・トレース・ロギングを使用可能に設定
有効な JCo トレース・ログを使用可能にし、AssemblyLine のロギングとトレースに組み込むかどうかを定義する一般属性。

JCo ミドルウェアのトレース・レベル
JCo ミドルウェアのトレース・レベルを定義する一般属性。

JCo ミドルウェアのトレース・ファイル・パス
JCo ミドルウェアのトレース・ファイルを作成するディレクトリーを定義する一般属性。RFM 要求を XML の内容を含むファイルとして保管する場合にも使用されます。

SAP ALE IDOC コネクターの使用

ここに記載されている情報を使用することで、SAP ALE IDOC コネクターをイテレーター・モードで処理できます。

また、コネクターでサポートされる IBM Security Directory Integrator スキーマについても説明します。

IBM Security Directory Integrator スキーマ

ここに記載されている表内のデータを使用することで、SAP ALE IDOC コネクター・スキーマについて詳細に理解できるようになります。

コネクターのスキーマは、個々の IDOC を表す項目を AssemblyLine に提供する際の中心になります。IDOC それ自体は、3 つのデータ・セクションを含んでいます。これらのセクションは、「制御データ」、「セグメント・データ」、および「状況データ」です。このデータを IBM Security Directory Integrator で表す最も簡単かつ効率的な方法は、必要なデータに関して簡単に分析できる XML フォーマットを使用する方法です。制御データはアクセスが容易であり、役立つスタンドアロン情報を提供できるため、このデータを個々の属性として使用することもできま

す。制御データをスタンドアロン属性として実動可能にする、または実動不可にするためには、構成パラメーター「IDOC AS は XML のみ」が使用されます。

コネクターはリモート関数モジュール要求を受け取ることもできるため、データを 1 つ以上の属性で表すための要件が存在します。現在、RFM の内容は単一の XML 値属性として使用することができます。RFM XML 値属性を実動可能にする、または実動不可にするためには、構成パラメーター「SAP RFM 要求を処理するかどうか」が使用されます。

このコネクターで使用できるスキーマの定義を以下の表に示します。

表 56. SAP ALE IDOC コネクター・スキーマ

属性名	属性の説明	属性の構文
idoc.tid	SAP システム・クライアントによって提供された関連 TID 値を値として持つ入力スキーマ属性。	java.lang.string
idoc.xml	XML フォーマットの完全 IDOC を値とする入力スキーマ属性。	java.lang.string
idoc.segments.xml	XML フォーマットの完全なセグメント階層を値として持つ入力スキーマ属性。この XML には制御属性は含まれていません。	java.lang.string
idoc.ctrl.ArchiveKey	IDOC 制御データのアーカイブ・キーを表す値 (「ARCKEY」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.Client	IDOC 制御データのクライアントを表す値 (「MANDT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.CreationDate	IDOC 制御データの作成日を表す値 (「CREDAT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.CreationTime	IDOC 制御データの作成時刻を表す値 (「CRETIM」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.Direction	IDOC 制御データの方向を表す値 (「DIRECT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDIMessage	IDOC 制御データの EDI メッセージを表す値 (「REFMES」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDIMessageGroup	IDOC 制御データの EDI メッセージ・グループを表す値 (「REFGRP」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDIMessageType	IDOC 制御データの EDI メッセージ・タイプを表す値 (「STDMES」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDIStandardFlag	IDOC 制御データの EDI 標準フラグを表す値 (「STD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDIStandardVersion	IDOC 制御データの EDI 標準バージョンを表す値 (「STDVRS」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.EDITransmissionFile	IDOC 制御データの EDI 伝送ファイルを表す値 (「REFINT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string

表 56. SAP ALE IDOC コネクター・スキーマ (続き)

属性名	属性の説明	属性の構文
idoc.ctrl.ExpressFlag	IDOC 制御データ Express フラグを表す値 (「EXPRSS」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.IDocCompoundType	IDOC 制御データの IDOC 複合タイプを表す値 (「DOCTYP」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.IDocNumber	IDOC 制御データの IDOC 番号を表す値 (「DOCNUM」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.IDocSAPRelease	IDOC 制御データの IDOC SAP リリースを表す値 (「DOCREL」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.IDocType	IDOC 制御データの IDOC タイプを表す値 (「IDOCTYP」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.IDocTypeExtension	CIM タイプまたはカスタマー拡張タイプとしても知られている IDOC 制御データの IDOC タイプ拡張を表す値 (「CIMTYP」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.MessageCode	IDOC 制御データのメッセージ・コードを表す値 (「MESCOD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.MessageFunction	IDOC 制御データのメッセージ関数を表す値 (「MESFCT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.MessageType	IDOC 制御データのメッセージ・タイプを表す値 (「MESTYP」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.OutputMode	IDOC 制御データの出力モードを表す値 (「OUTMOD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientAddress	IDOC 制御データの受信側アドレスを表す値 (「RCVSAD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientLogicalAddress	IDOC 制御データの受信側論理アドレスを表す値 (「RCVLAD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientPartnerFunction	IDOC 制御データの受信側パートナー関数を表す値 (「RCVPFC」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientPartnerNumber	IDOC 制御データの受信側パートナー番号を表す値 (「RCVPRN」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientPartnerType	IDOC 制御データの受信側パートナー・タイプを表す値 (「RCVPRT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.RecipientPort	IDOC 制御データの受信側ポートを表す値 (「RCVPOR」フィールドの値) を持つ入力スキーマ属性。	java.lang.string

表 56. SAP ALE IDOC コネクター・スキーマ (続き)

属性名	属性の説明	属性の構文
idoc.ctrl.SenderAddress	IDOC 制御データの送信側アドレスを表す値 (「SNDSAD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.SenderLogicalAddress	IDOC 制御データの送信側論理アドレスを表す値 (「SNDLAD」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.SenderPartnerFunction	IDOC 制御データの送信側パートナー関数を表す値 (「SNDPFC」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.SenderPartnerNumber	IDOC 制御データの送信側パートナー番号を表す値 (「SNDPRN」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.SenderPartnerType	IDOC 制御データの送信側パートナー・タイプを表す値 (「SNDPRT」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.SenderPort	IDOC 制御データの「送信側ポートを戻す」を表す値 (「SNDPOR」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.Serialization	IDOC 制御データのシリアライゼーションを表す値 (「SERIAL」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.Status	IDOC 制御データの状況を表す値 (「STATUS」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
idoc.ctrl.TableStructureName	IDOC 制御データのテーブル構造名を表す値 (「TABNAM」フィールドの値) を持つ出力スキーマ属性。	java.lang.string
idoc.ctrl.TestFlag	IDOC 制御データのテスト・フラグを表す値 (「TEST」フィールドの値) を持つ入力スキーマ属性。	java.lang.string
rfm.xml	XML フォーマットの RMF™ 要求の完全な内容を表す値を持つ入力スキーマ属性。	java.lang.string

`java.lang.String` 型の属性は任意の長さにできます。

XML 属性の構文解析

ここに記載されている情報を使用することで、XML 属性の構文解析を実行できるようになります。

コネクターに対するメインの操作モードは、IDOC または RFM の完全な内容を表す XML 値属性を実動使用することです。そのため、このデータを処理するための最適な方法は、コネクターに添付されているいずれかの使用可能な IBM IBM Security Directory Integrator XML パーサーを使用する方法です。生成される XML はネスト構造であるため、DOM パーサーは推奨されませんが、使用することは可能です。推奨されるパーサーは、コネクターと統合する SAP ALE 分散モデルのタイプに応じて、SAX パーサーまたは XSLT パーサーとなります。コネクターが複数の IDOC メッセージ・タイプを処理する必要がある場合、または RFM 要求を処理するようコネクターが構成されている場合は、SAX パーサーが推奨されます。これは、IDOC メッセージ・タイプ、および RFM XML が異なれば XML スキーマ

も異なるためです。SAX パーサーは、さまざまな XML スキーマで XML 値を処理できる唯一の IBM Security Directory Integrator パーサーです。この処理を行うには、SAX パーサーの「グループ」構成パラメーターが特定の値を持たないよう、パーサーを構成します。これには、特定のルート・エレメントを定義する必要がないという効果があります。コネクターが処理する IDOC のタイプが 1 つのみであることがはっきりしている場合は、XSLT パーサーを使用できます。XSLT パーサーを使用すると、コネクター項目属性と作業項目属性との間のより完全なマッピングが可能になります。例えば、SAP HR マスター・データの受信側となるようコネクターが構成されていた場合は、メッセージ・タイプが HRMD_A である IDOC のみが得られることが期待できます。このコネクターが開発された時点では、このメッセージ・タイプの最新バージョンは HRMD_A06 でした。したがって、以下のような XSL を使用すれば、IDOC の内容を解析して、必要なデータが得られました。

```
<XSL:stylesheet xmlns:XSL="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <XSL:output method="XML" indent="yes" />

  <XSL:template match="HRMD_A06">
    <DocRoot>
      <Entry>
        <XSL:apply-templates select="./IDOC"/>
      </Entry>
    </DocRoot>
  </XSL:template>

  <XSL:template match="IDOC">
    <XSL:apply-templates select="./EDI_DC40"/>
    <XSL:apply-templates select="./E1PLOGI"/>
  </XSL:template>

  <XSL:template match="EDI_DC40">
    <Attribute name="IDOC_CTRL_DOCNUM">
      <XSL:for-each select="DOCNUM">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="IDOC_CTRL_MANDT">
      <XSL:for-each select="MANDT">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="IDOC_CTRL_DOCREL">
      <XSL:for-each select="DOCREL">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="IDOC_CTRL_IDOCTYP">
      <XSL:for-each select="IDOCTYP">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="IDOC_CTRL_SNDPOR">
      <XSL:for-each select="SNDPOR">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="IDOC_CTRL_RCVPOR">
      <XSL:for-each select="RCVPOR">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
  </XSL:template>
```

```
  <XSL:template match="E1PLOGI">
    <XSL:apply-templates select="./E1PITYP"/>
  </XSL:template>

  <XSL:template match="E1PITYP">
    <XSL:apply-templates select="./E1P0002"/>
    <XSL:for-each select="E1P0105">
      <Attribute name="PR_COMM_SUBTY">
        <XSL:for-each select="SUBTY">
          <Value>
             <XSL:value-of select="." />
          </Value>
        </XSL:for-each>
      </Attribute>
      <Attribute name="PR_COMM_USRID">
        <XSL:for-each select="USRID">
          <Value>
             <XSL:value-of select="." />
          </Value>
        </XSL:for-each>
      </Attribute>
      <Attribute name="PR_COMM_USRID_LONG">
        <XSL:for-each select="USRID_LONG">
          <Value>
             <XSL:value-of select="." />
          </Value>
        </XSL:for-each>
      </Attribute>
    </XSL:for-each>
  </XSL:template>

  <XSL:template match="E1P0002">
    <Attribute name="PR_PERNR">
      <XSL:for-each select="PERNR">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="PR_LASTNAME">
      <XSL:for-each select="NACHN">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="PR_FIRSTNAME">
      <XSL:for-each select="VORNA">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="PR_BIRTHDATE">
      <XSL:for-each select="GBDAT">
        <Value>
           <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
  </XSL:template>

</XSL:stylesheet>
```

IDOC メッセージ・タイプが常に USERCLONE メッセージ・タイプになる場合には、以下のような XSL を使用すれば、必要な属性マッピングを取得することができます。

```
<XSL:stylesheet xmlns:XSL="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <XSL:output method="XML" indent="yes" />

  <XSL:template match="USERCLONE05">
    <DocRoot>
      <Entry>
        <XSL:apply-templates select="./IDOC"/>
      </Entry>
    </DocRoot>
```

```
</XSL:template>

<XSL:template match="IDOC">
  <XSL:apply-templates select="./EDI_DC40"/>
  <XSL:apply-templates select="./E1BPBNAME"/>
  <XSL:apply-templates select="./E1BPLOGOND"/>
  <XSL:apply-templates select="./E1BPADDR3"/>
  <XSL:apply-templates select="./E1BPLOGOND"/>
  <XSL:apply-templates select="./E1BPUSCOMP"/>
</XSL:template>

<XSL:template match="EDI_DC40">
  <Attribute name="TDI_DOCNUM">
    <XSL:for-each select="DOCNUM">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_MANDT">
    <XSL:for-each select="MANDT">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_DOCREL">
    <XSL:for-each select="DOCREL">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_IDOCTYP">
    <XSL:for-each select="IDOCTYP">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_USERCLONE">
    <XSL:for-each select="USERCLONE">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_SNDPOR">
    <XSL:for-each select="SNDPOR">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
  <Attribute name="TDI_RCVPOR">
    <XSL:for-each select="RCVPOR">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
</XSL:template>

<XSL:template match="E1BPBNAME">
  <Attribute name="TDI_BAPIBNAME">
    <XSL:for-each select="BAPIBNAME">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
  </Attribute>
</XSL:template>

<XSL:template match="E1BPLOGOND">
  <Attribute name="TDI_CLASS">
    <XSL:for-each select="CLASS">
      <Value>
         <XSL:value-of select="." />
      </Value>
    </XSL:for-each>
```

```
    </Attribute>
    <Attribute name="TDI_TZONE">
      <XSL:for-each select="TZONE">
        <Value>
          <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
  </XSL:template>

  <XSL:template match="E1BPADDR3">
    <Attribute name="TDI_FIRSTNAME">
      <XSL:for-each select="FIRSTNAME">
        <Value>
          <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <Attribute name="TDI_LASTNAME">
      <XSL:for-each select="LASTNAME">
        <Value>
          <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
    <XSL:apply-templates select="./E1BPADDR1"/>
  </XSL:template>

  <XSL:template match="E1BPADDR1">
    <Attribute name="TDI_E_MAIL">
      <XSL:for-each select="E_MAIL">
        <Value>
          <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
  </XSL:template>

  <XSL:template match="E1BPUSCOMP">
    <Attribute name="TDI_COMPANY">
      <XSL:for-each select="COMPANY">
        <Value>
          <XSL:value-of select="." />
        </Value>
      </XSL:for-each>
    </Attribute>
  </XSL:template>

</XSL:stylesheet>
```

SAP ALE 分散モデルでの構成

コネクターを SAP システム上の論理システムとしてセットアップして、そのコネクターを SAP ALE 分散モデルの一部にすることができます。

この場合、以下の 2 つのアクションが必要となります。

1. TCP/IP タイプの RFC 宛先が作成され、この宛先でコネクターが外部プログラムとして登録されます。RFC 宛先で提供されているプログラム名が、コネクター構成パラメーターである「IDOC Server Program ID」に提供したのと同じ値であることを確認します。次に、RFC 接続をテストするために、コネクターが IBM Security Directory Integrator AssemblyLine で稼働していることを確認します。この AssemblyLine は、コネクターしか持たない必要最小限の AssemblyLine で、生成された作業項目属性をダンプするためのスクリプト・コンポーネントである可能性があります。SAP GUI トランザクション SM59 を使用して RFC 宛先を作成します。
2. ステップ 1 で作成された RFC 宛先と同じ名前を持つ論理システムが作成されます。これは SAP GUI トランザクション SALE を使用して行われます。実際

に現実の SAP システム・クライアントを表している他の SAP 論理システムの場合のように、クライアントを論理システムに割り当てる必要はありません。

配置された論理システムは、SAP ALE 分散モデルの他の論理システムの場合とほとんど同じように使用することができます。コネクターは、SAP HR マスター・データ分散モデルの一部、および定義済みの SAP CUA 分散モデルの一部としてテストされています。他の SAP ALE 分散モデルを使用していて他の問題が発生した場合は、IBM サポートに連絡してください。

SAP ABAP Application Server Component Suite のトラブルシューティング

ここに記載されている情報を使用することで、SAP ABAP Application Server Component Suite をトラブルシューティングできます。

SAP Java が正しくインストールされていない

インストール状況を確認し、必要に応じて再インストールを実行してください。

使用できない XSL スタイルシート

コネクターは XSL スタイルシートを使用してそれぞれの操作を実行します。ソリューション・ディレクトリーで XSL フォルダーを使用できない場合に発生する可能性のある問題については、この注を参照してください。

sapjco.jar が欠落している

SAP ABAP Application Server RFC FC を使用しようとすると、以下のメッセージのようなエラーが発生する場合:

```
13:01:58  Error in: InitConnectors: java.lang.ClassCastException:
     java.lang.NoClassDefFoundError

java.lang.ClassCastException: java.lang.NoClassDefFoundError
```

この場合、SAP JCo が正しくインストールされていない可能性があります。sapjco.jar が *IBM Security Directory Integrator_Home*/jars ディレクトリー内にあることを確認します。 584 ページの『SAP Java Connector の構成』の説明を参照してください。

librfc32.dll が欠落している

SAP ABAP Application Server FC を使用しようとして、次のメッセージのようなエラーが表示されたとします。「*指定されたパスでダイナミック・リンク・ライブラリー LIBRFC32.dll が見つかりませんでした (The dynamic linked library LIBRFC32.dll could not be found in the specified path)*」。Windows マシンの場合は、librfc32.dll が *IBM Security Directory Integrator_Home*/libs ディレクトリー内にあるか確認してください。Solaris および AIX マシンでは、librfccm.{o/so} がロード可能なライブラリーのパスに追加されていることを確認します。

古いバージョンの librfc32.dll

以下のタイプのエラーが発生することがあります。

```
java.lang.ClassCastException: java.lang.ExceptionInInitializerError
```

この場合は、使用している librfc32 のバージョンが古く、JCo 2.1.6 との互換性がありません。パスに他の librfc32 が含まれていないことを確認し

てください。また、システム・パスに含まれているすべての librfc32*.{dll/so} のバージョンが、6403.3.81.4751 以上であることを確認してください。

```
15:13:44  [YourAssemblyLine] BEGIN   selectEntries

15:13:45  [YourAssemblyLine] handleException: initialize,
java.lang.ClassCastException: java.lang.ExceptionInInitializerError

15:13:45  [YourAssemblyLine] initialize

java.lang.ClassCastException: java.lang.ExceptionInInitializerError
     at com.ibm.di.script.ScriptEngine.call(Unknown Source)
     at com.ibm.di.connector.ScriptConnector.selectEntries(Unknown Source)
     at com.ibm.di.server.AssemblyLineComponent.initialize(Unknown Source)
     at com.ibm.di.server.AssemblyLine.initConnectors(Unknown Source)
     at com.ibm.di.server.AssemblyLine.msInitConn(Unknown Source)
     at com.ibm.di.server.AssemblyLine.executeMainStep(Unknown Source)
     at com.ibm.di.server.AssemblyLine.executeMainLoop(Unknown Source)
     at com.ibm.di.server.AssemblyLine.executeAL(Unknown Source)
     at com.ibm.di.server.AssemblyLine.run(Unknown Source)
```

RFC_ERROR_SYSTEM_FAILURE: ユーザーに接続しない状態での画面出力

コネクターからこのメッセージが戻される場合は、詳細について SAP ノート 49730 を参照してください。

スキーマ照会の問題

IBM Security Directory Integrator GUI でコネクターを使用してスキーマ照会を実行すると、データ・ソースへの接続試行時に例外が発生することがあります。この例外は無視できます。その後、スキーマを**ディスカバー**するボタンを使用する操作は正常に実行されます。

コネクターでは、*次の項目の取得* スタイルのスキーマ照会はサポートされていません。コネクターでは、「データ・ソースのスキーマのディスカバー」 (「**接続**」ボタン) スタイルのスキーマ・ディスカバリーがサポートされます。

ユーザー・レジストリー会社コード割り当て

XML エレメント <companyKeyName> に関連付けられている値が SAP 内の有効な会社コードを表す値でない場合、またはこの値がそもそも指定されていない場合は、SAP により構成済みデフォルト値が割り当てられます。

既に AssemblyLine に含まれているコネクターのモードの変更

テスト中に、AssemblyLine 内のコネクターのモード変更操作が成功しない場合があることが判明しました。コネクターがその本来のモードで稼働しているように見えていても、AssemblyLine エラーが発生することがあります。このような状況が発生した場合は、コネクターを削除し、新しいモードでコネクターを AssemblyLine に追加してください。

関数コンポーネントと SE37 RFC テスト機能の相異点

場合によっては、RFC 関数コンポーネントの動作が、SAP のトランザクション SE37 から利用できる*Test Function Feature* での特定の RFC の実行時の動作と多少異なります。また、SAP テスト機能により値がドイツ語短縮形の内部値に自動的に変換されることがあります (BAPI_SALESORDER_GETLIST など)。したがって、**ルックアップ**・モードおよび**イテレーター**・モードのコネクターから戻される一部の値が、SAP 汎用モジュール・テスト機能により戻される値と多少異なることがあります。パラメーター値を設定するために入力 XML ファイルを指定する必要がある場合は、内部値 (**ルックアップ**・モードおよび**イテレーター**・モードのコネクターから戻される値と同じ形式) を指定します。

RFC 関数コンポーネントでは、文字ストリング・タイプの値を最大長まで埋め込む操作は行われません。

User Registry Connector に関する警告

場合によっては、アプリケーション・レベルの ABAP 警告が SAP から戻された結果、重大度の警告メッセージがコネクターによってログに記録されることがあります。**イテレーター**・モードで稼働している User Registry Connector によりログに記録される警告メッセージの例を以下に示します。

```
15:50:10  [newGetUsers] W: Unable to read the address (69) (D:¥Program
   Files¥IBM¥IBMDirectoryIntegrator¥xsl¥bapi_user_get_detail_precall.xsl)

15:50:10  [newGetUsers] W: Unable to determine the company (76) (D:¥Program
   Files¥IBM¥IBMDirectoryIntegrator¥xsl¥bapi_user_get_detail_precall.xsl)
```

ほとんどの場合、このような警告メッセージは無視できます。

更新モードの User Registry Connector

このモードで稼働するコネクターでは、**リンク基準** に **sapUserName** 属性が定義されており、**sapUserXml** 属性に関連付けられている値の中に XML エレメントとして <sapUserName> が含まれていることが予期されます。この両方の **sapUserName** の値が一致している必要があります。コネクターはこれらの値が等しいかどうかを検証しません。

SAP でのパスワードの動作

SAP での新規ユーザーの作成後、または既存のユーザーのパスワード変更後に、次回ログオン時に SAP からそのユーザーに対し、パスワードをリセットするよう求めるプロンプトが出されます。これは SAP トランザクション SU01 またはコネクターからユーザーが作成または変更された場合に発生する SAP の標準動作です。

HR コネクターでの HR 個人データの削除

コネクターまたは SAP トランザクション PA30 を使用して個人データ入力項目を削除しようとすると、失敗することがあります。失敗メッセージには「*Record cannot be deleted (time constraint 1)*」と示されます。現時点では、この問題の解決策は判明していません。

SAP ABAP Application Server Component Suite の補足情報

以下に記載されているユーザー・レジストリー・コネクターのサンプル・コードおよびスキーマを使用できます。

ユーザー・レジストリー・コネクター XML インスタンス文書の例

ここに記載されているサンプル・コードを使用することで、ユーザー・レジストリー・コネクター XML インスタンス文書の例を表示できます。

```
<User>
      <sapUserName></sapUserName>
      <sapUserPassword></sapUserPassword>
      <sapUserAlias>
          <aliasName></aliasName>
      </sapUserAlias>
      <sapAddress>
          <title></title>
          <academicTitle></academicTitle>
          <firstName></firstName>
          <lastName></lastName>
          <namePrefix></namePrefix>
          <nameFormat></nameFormat>
          <nameFormatRuleCountry></nameFormatRuleCountry>
```

```
    <isoLanguage></isoLanguage>
    <language></language>
    <searchSortTerm></searchSortTerm>
    <department></department>
    <function></function>
    <buildingNumber></buildingNumber>
    <buildingFloor></buildingFloor>
    <roomNumber></roomNumber>
    <name></name>
    <name2></name2>
    <name3></name3>
    <name4></name4>
    <city></city>
    <postCode></postCode>
    <poBoxPostCode></poBoxostCode>
    <poBox></poBox>
    <street></street>
    <streetNumber></streetNumber>
    <houseNumber></houseNumber>
    <country></country>
    <countryIso></countryIso>
    <region></region>
    <timeZone></timeZone>
    <primaryPhoneNumber></primaryPhoneNumber>
    <primaryPhoneExtension></primaryPhoneExtension>
    <primaryFaxNumber></primaryFaxNumber>
    <primaryFaxExtension></primaryFaxExtension>
</sapAddress>
<sapCompany>
    <companyNameKey></companyNameKey>
</sapCompany>
<sapDefaults>
    <startMenu></startMenu>
    <outputDevice></outputDevice>
    <printTimeAndDate></printTimeAndDate>
    <printDelete></printDelete>
    <dateFormat></dateFormat>
    <decimalFormat></decimalFormat>
    <logonLanguage></logonLanguage>
    <cattTestStatus></cattTestStatus>
    <costCenter></costCenter>
</sapDefaults>
<sapLogonData>
    <validFromDate></validFromDate>
    <validToDate></validToDate>
    <userType></userType>
    <userGroup></userGroup>
    <accountId></accountId>
    <timeZone></timeZone>
    <lastLogonTime></lastLogonTime>
    <codeVerEncryption></codeVerEncryption>
</sapLogonData>
<sapSncData>
    <printableName></printableName>
    <allowUnsecure></allowUnsecure>
</sapSncData>
<sapUserGroupList>
    <group>
        <name></name>
    </group>
    <group>
        <name></name>
    </group>
</sapUserGroupList>
<sapParameterList>
    <parameter>
        <parameterId></parameterId>
        <parameterValue></parameterValue>
    </parameter>
    <parameter>
        <parameterId></parameterId>
        <parameterValue></parameterValue>
    </parameter>
</sapParameterList>
<sapUserEmailAddressList>
    <email>
        <defaultNumber></defaultNumber>
        <smtpAddress></smtpAddress>
       <isHomeAddress></isHomeAddress>
        <sequenceNumber></sequenceNumber>
```

```
            </email>
            <email>
                <defaultNumber></defaultNumber>
                <smtpAddress></smtpAddress>
                <isHomeAddress></isHomeAddress>
                <sequenceNumber></sequenceNumber>
            </email>
        </sapUserEmailAddressList>
        <sapRoleList>
            <role>
                <name></name>
                <validFromDate></validFromDate>
                <validToDate></validToDate>
            </role>
            <role>
                <name></name>
                <validFromDate></validFromDate>
                <validToDate></validToDate>
            </role>
        </sapRoleList>
        <sapProfileList>
            <profile>
                <name></name>
            </profile>
           <profile>
                <name></name>
            </profile>
        </sapProfileList>
    </User>
```

ユーザー・レジストリー・コネクター XML の XSchema

以下に示すユーザー・レジストリー・コネクター XML の XSchema を使用できます。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="User">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="sapUserName" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="sapUserPassword" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="sapUserAlias" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapAddress" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="sapCompany" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapDefaults" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapLogonData" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapSncData" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapUserGroupList" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapParameterList" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapUserEmailAddressList" minOccurs="0"
                     maxOccurs="1"/>
                <xsd:element ref="sapRoleList" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="sapProfileList" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="academicTitle">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="20"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="accountId">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="12"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="aliasName">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="40"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
```

```
<xsd:element name="allowUnsecure">
 <xsd:simpleType >
  <xsd:restriction base="xsd:boolean">
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="buildingFloor">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="buildingNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="cattTestStatus">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="companyNameKey">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="42"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="costCenter">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="8"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="dateFormat">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="decimalFormat">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="defaultNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="department">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="email">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="defaultNumber" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="smtpAddress" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="isHomeAddress" maxOccurs="1" minOccurs="0"/>
            <xsd:element ref="sequenceNumber" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="firstName">
```

```
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="function">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="group">
    <xsd:complexType>
        <xsd:sequence>
   <xsd:element name="name">
    <xsd:simpleType >
     <xsd:restriction base="xsd:string">
      <xsd:maxLength value="12"></xsd:maxLength>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="isHomeAddress">
 <xsd:simpleType >
  <xsd:restriction base="xsd:boolean">
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="isoLanguage">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="2"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="language">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="lastLogonTime">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:minLength value="8"></xsd:minLength>
   <xsd:maxLength value="8"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="lastName">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="logonLanguage">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="1"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="name">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="name2">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
```

```
 </xsd:simpleType>
</xsd:element>
<xsd:element name="name3">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="name4">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="40"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="nameFormat">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="2"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="nameFormatRuleCountry">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="3"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="namePrefix">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="20"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="outputDevice">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="4"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="parameter">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="parameterId" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="parameterValue" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="parameterId">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="20"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="parameterValue">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="18"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="poBox">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="postCode">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="primaryFaxExtension">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="primaryFaxNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="30"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="primaryPhoneExtension">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="primaryPhoneNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="30"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="printDelete">
 <xsd:simpleType >
  <xsd:restriction base="xsd:boolean">
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="printTimeAndDate">
 <xsd:simpleType >
  <xsd:restriction base="xsd:boolean">
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="printableName">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="255"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="profile">
    <xsd:complexType>
        <xsd:sequence>
   <xsd:element name="name">
    <xsd:simpleType >
     <xsd:restriction base="xsd:string">
      <xsd:maxLength value="12"></xsd:maxLength>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="region">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="3"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="role">
    <xsd:complexType>
        <xsd:sequence>
   <xsd:element name="name">
    <xsd:simpleType >
     <xsd:restriction base="xsd:string">
      <xsd:maxLength value="30"></xsd:maxLength>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
            <xsd:element ref="validFromDate" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="validToDate" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
</xsd:element>
<xsd:element name="roomNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="10"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="sapAddress">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="academicTitle" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="firstName" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="lastName" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="namePrefix" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="nameFormat" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="nameFormatRuleCountry" minOccurs="0"
                 maxOccurs="1"/>
            <xsd:element ref="isoLanguage" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="language" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="searchSortTerm" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="department" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="function" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="buildingNumber" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="buildingFloor" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="roomNumber" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="name" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="name2" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="name3" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="name4" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="postCode" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="poBox" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="street" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="region" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="timeZone" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="primaryPhoneNumber" minOccurs="0"
                 maxOccurs="1"/>
            <xsd:element ref="primaryPhoneExtension" minOccurs="0"
                 maxOccurs="1"/>
            <xsd:element ref="primaryFaxNumber" minOccurs="0"
                 maxOccurs="1"/>
            <xsd:element ref="primaryFaxExtension" minOccurs="0"
                 maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapCompany">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="companyNameKey" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapDefaults">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="startMenu" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="outputDevice" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="printTimeAndDate" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="printDelete" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="dateFormat" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="decimalFormat" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="logonLanguage" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="cattTestStatus" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="costCenter" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapLogonData">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="validFromDate" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="validToDate" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="userType" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="userGroup" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="accountId" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="timeZone" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="lastLogonTime" minOccurs="0" maxOccurs="1"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapParameterList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" ref="parameter"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapProfileList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" ref="profile"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapRoleList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" ref="role"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapSncData">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="printableName" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="allowUnsecure" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapUserAlias">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="aliasName" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapUserEmailAddressList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" ref="email"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapUserGroupList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" ref="group"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="sapUserName">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="12"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="sapUserPassword">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="8"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="searchSortTerm">
 <xsd:simpleType >
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="20"></xsd:maxLength>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="sequenceNumber">
 <xsd:simpleType >
  <xsd:restriction base="xsd:nonNegativeInteger">
   <xsd:totalDigits value="3"></xsd:totalDigits>
  </xsd:restriction>
 </xsd:simpleType>
```

```
    </xsd:element>
    <xsd:element name="smtpAddress">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="241"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="startMenu">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="20"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="street">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="60"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="timeZone">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="6"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="title">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="30"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="userGroup">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="12"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="userType">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="1"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="validFromDate">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="10"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="validToDate">
     <xsd:simpleType >
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="10"></xsd:maxLength>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
</xsd:schema>
```

第 6 章 資産統合スイート

資産統合スイートを使用することで、IT レジストリーに格納されるデータを実社会でのエンティティーに対応させるための編成方法や、エンティティー間の関係を当該データにより定義する方法を指定することができます。

IBM は、疎結合の多品種環境における統合を実現するための手段として、資産データ統合イニシアチブを開始しました。その主な目的は以下のとおりです。

- 製品全体で一貫したデータ表記
- 共通メカニズムによる改良データの共有
- データの冗長性の削減

このイニシアチブにより、CDM (Common Data Model)、IT レジストリー (Data Integration Service)、および IdML (Identity Markup Language) が開発されました。

注: 用語「IT レジストリー」は、IT リソースの集中処理を可能にする IBM 製品のことを指します。この用語は正式に承認されたものではなく、今後変更される可能性があることに注意してください。

Common Data Model (CDM) は、IT レジストリーに格納された情報の一般的特性を定義する、一貫性のある統合された論理データ・モデルです。CDM では、使用する管理アプリケーションを簡単に使えるようにする手段の管理情報を表します。

CDM コンポーネント

CDM を使用することで、使用中のすべての論理モデル (CIM、BPEL、ITIL、SNA、TMf など) から情報を組み込んで、1 つの一貫したモデルに統合することができます。

統一モデリング言語 (UML) に基づいて、CDM ではエンティティーに関する管理情報 (管理エレメントまたは構成アイテムと呼ばれます) およびこれらのエンティティー間の関係を表します。CDM および関連資料については、IBM Security Directory Integrator 製品 DVD に組み込まれている Tivoli CDM の Web サイトを参照してください。Web サイトにアクセスするには、IBM Security Directory Integrator の DVD から `CDMWebsite.zip` ファイルをコピーして unzip します。CDM は、UML からその概念のほとんどを流用しており、 モデルのコンテンツは IBM Rational® Software Architect などの UML 開発ツールで使用可能です。

属性

属性を最も基本的な粒度で使用することで、アトミック・データを、UML で定義したとおりに、属性として表現することができます。

属性は、関連データ・タイプ、使用可能なデフォルト値を持ち、さらに属性が単一な値または複数の値のどちらを取るかの仕様も含まれています。特定のデータ・タイプおよび列挙では、属性が含むことのできる実際の値は制限されます。CDM 内の属性はすべてグローバルに定義されます。つまり、同じ名前を持つ属性は、使用さ

れるコンテキストにかかわらず、同じ意味を持ちます。これは、さまざまな環境および状況におけるイベントなどの属性の定義および使用に一貫性を持たせるためです。

属性の例を以下に示します。

- Manufacturer
- MemorySize
- PrimaryOwner
- PrimaryMacAddress

クラス

属性は、CDM 内で、実社会のアイテム (コンピューター、ユーザー、ビジネス・プロセスなど) に対応するエンティティーにグループ化することができます。この属性のグループ化は、クラスと呼ばれます。

属性は CDM 内部で、コンピューター、ユーザー、またはビジネス・プロセスなど、実社会のアイテムに対応するエンティティーにグループ化されます。この属性のグループ化は、クラスと呼ばれます。CDM のクラスは、属性を複数のクラスで共有できる単一継承階層に配置されます。

クラスは抽象になる場合があります。抽象クラスはエンティティーの一般的特性を含みますが、これらのクラスのインスタンスを作成することはできません。クラス階層のルートである `ModelObject` は、抽象クラスの一例です。CDM のクラスの大部分は具象で、これらのインスタンスを IT レジストリーに作成することができます。

CDM のクラス階層は、クラス `ConfigurationItem` ではなく、`ModelObject` にルートされることに注意してください。CI に加え、その他の種類のデータも IT レジストリーに格納され、CDM を使用してモデル化されます。

インターフェース

CDM 内で多重継承の目的を満足させるには、インターフェースを使用することができます。

実社会に一般的に発生する多くの状況は、人々が多重継承を使用する原因となります。多重継承は UML ではサポートされていますが、CDM ではサポートされていません。これらの状況を処理するため、CDM にはインターフェースの概念が組み込まれています。これは、クラス階層の任意の場所にあるクラス定義に「組み込む」ことができる属性の一貫したコレクション (または一貫した関係元または関係先) です。CDM がメソッドではなくデータだけを組み込むことを除けば、これは Java によるインターフェースの処理方法と似ています。

インターフェース自体は、その他のインターフェースから派生し、別の継承階層を形成できます。ただし、インターフェース階層は複数のルートを持つことはできますが、派生階層にインターフェースとクラスを混在させることはできません。クラスはその他のクラスからのみ、インターフェースはその他のインターフェースからのみ派生できます。

関係

関係は、クラスとインターフェースとの関連を確立するのに使用できます。

IT レジストリーの最も重要な目的の一つは、実社会におけるエンティティー間の関係を保管することです。したがって、CDM はクラス間およびインターフェース間の関係の定義に大きく重点を置き、特定の意味論的な意味をリレーションシップに割り当てます。例えば、「runsOn」という関係は、ソフトウェアの一部を特定の環境で実行するという事実を表す場合があります。

CDM 内の関係は、アソシエーションと呼ばれる UML 内の同様の概念と関連付けられますが、その概念は異なります。アソシエーションは、UML 内のクラス間の意味的リンクです。1 つの例として、実現 (realization) があります。ここでは、1 つのエンティティーは特定のインターフェースを使用可能にします。UML ではユーザーにアソシエーションの意味を表すように強制することはありません。2 つのエンティティー間に単純に線を引くだけです。CDM では、すべてのアソシエーション (汎化および実現 (realization) 以外) に名前および型が付けられます。アソシエーションの名前には、対応する意味を与えます。したがって、アソシエーションは関係を形成します。同じ名前を持つアソシエーションはすべて、同じ意味を持ちます。

命名および識別

IT レジストリー内のエンティティーの ID に一貫性を持たせるには、命名規則を使用することができます。

エンティティー間の関係の表現および保管に加えて、IT レジストリーはエンティティー間の相関メカニズムを提供します。例えば、2 つの管理製品が 1 つのコンピューター・システムを検出して、それらを異なる名前で呼ぶ場合があります。そのため、単一のエンティティーとしてこれを表すのは重要なことです。IT レジストリー内のエンティティーの識別に一貫性を持たせるため、CDM はエンティティーの各タイプ (各クラス) がどのように識別されるかを正式に定義します。その方法として、モデルは*命名規則*を使用します。

命名規則は、識別特性を提供する属性、エンティティーの識別に必要とされる属性の組み合わせ、および識別を一意にするコンテキストをリストします。命名規則の 2 つの例を以下に示します。

- 「Manufacturer」、「MachineType」、「Model」、および「SerialNumber」を組み合わせることにより、ある 1 つのコンピューターが一意に識別されます。
- 論理ディスクの「DriveLetter」により、オペレーティング・システムのコンテキスト内でディスクが一意に識別されます。

IT レジストリー内の相関は、これらの規則の一貫した使用、および同じタイプのインスタンスを識別する規則の把握によって促進されます。同じインスタンスに対して複数の名前が生じた場合、それらはエイリアスと呼ばれ、IT レジストリーは重複を単一インスタンスとして示します。命名規則を使用した、名前の一貫した形成により、IT レジストリー (またはアプリケーション) にインスタンスのグローバル一意識別子 (GUID) として知られる有用なバイナリー・トークンを生成することもできるようになります。

IT レジストリー

IT レジストリーは、管理情報処理の概念を実装し、管理対象データの処理方法を取得するのに使用できます。

CDM における管理情報 (表記、命名、および識別) の処理方法に関する概念を取り入れるため、 IBM Security Directory Integrator は IT レジストリーを使用します。作業は以下の 2 つのパートで構成されます。

- 登録済みリソースおよび CDM メタデータ (CDM クラスの定義、関係、命名規則など) を含む集中データベース。IBM Security Directory Integrator では、デフォルトでそのようなデータベースがセットアップ済みで、リモートで使用可能になっていると想定していますが、ローカル・インスタンスのセットアップに必要なすべてのものも提供します。詳しくは、 670 ページの『IT レジストリー・データベースのセットアップ』を参照してください。
- データ統合プロセスに関するさまざまなアクティビティーの実行に役立つ Java API を提供する一連のサービス。 IBM Security Directory Integrator は、そのコネクター、関数コンポーネント、およびパーサーの一部において、これらの Java API を直接利用します。

IT レジストリーで提供される最も重要な機能は、命名および調整です。2 つの製品が同じリソースを管理しますが、製品の機能に基づいてリソースを別々に識別することを想定します。さらに、2 つの製品は協調的な方法で効率的にまとめて作業することはできません。各製品にはリソースのデータのサブセットしかないため、各製品はそれがまったく同じリソースであるということが分かりません。この状況を解決するため、ユーザーは IBM Security Directory Integrator と IT レジストリーの組み合わせを使用できます。IBM Security Directory Integrator は、各製品と通信して、そのデータを取得し、Java API を使用して IT レジストリーにそれを登録します。このようにし、IT レジストリーはリソースの命名、表記、および保管を処理します。さらに、使用可能な命名規則を採用し、また 2 つの製品が同じリソースについて「通信」しているかどうかを確認します。このようにし、単一リソースのみを IT レジストリーに保持し、両方の製品から取得した情報を組み込みます。

このシナリオで起こり得るもう 1 つの問題は、2 つの製品で別々の用語を使用して同じエンティティーを記述できることです (例えば、「IBM」および「IBM Corporation」の両方を使用してリソースのメーカーを表します)。これは、シンプルなストリング・マッピング機能によって、IT レジストリーで処理されます。鍵となる用語「IBM」の場合、受け入れ可能な一連の表記があり (例えば、「IBM」、「IBM Corporation」、「IBM Corp」)、入力した値がこのいずれかと一致すると、鍵となる用語が戻されます。そのため、元の値は取り除かれています。このソリューションの使用により、その他複数の製品で使用可能なリソース表記が、よりクリアで、スモール化され、一貫したものになります。

命名および調整に加えて、IBM Security Directory Integrator は Common Data Model メタデータの記入に IT レジストリーを使用します。リソースの登録時に、ユーザーはそれらのクラスおよび属性を指定する必要があります。CDM でサポートされる (または必要とされる) クラスや属性を知らないと、登録することができません。そのため、IBM Security Directory Integrator は IT レジストリーのメタデータ機能を使用して必要な定義を取得し、このプロセスにおけるユーザーの負担が大幅に軽減されます。

IBM Security Directory Integrator は、ここで説明されている機能を使用するコンポーネントを提供して、その統合ソリューションでのコンポーネントの使用を許可します。詳しくは、『スイートのコンポーネント』を参照してください。

IT レジストリーは、一体的にリソースを処理するための最適な方法を提供します。ただし、 Common Data Model を使用するソフトウェア・コンポーネント間のコミュニケーション技術がもう 1 つあります (Discovery Library Adapter)。これらは、管理ソフトウェア・システム (または OMP) に固有のメカニズムを活用して、リソースおよびリソース関係に関する特定の詳細を抜き出すランタイム・コンポーネントです。これらは次に、この情報を IdML スキーマ準拠のファイルに変換します。したがって、Discovery Library Adapter の目的は、ビジネス・アプリケーションを含む、ビジネスおよびインフラストラクチャー・プロセスをサポートするリソースおよび関係の現在のセットを検出および保持することです。

IdML (Identity Markup Language) は、Discovery Library XML スキーマの仕様で、管理ソフトウェア・システム (MSS) のデータ・セットを保存するための標準的な方法、および管理対象リソースの作成、更新、および削除という操作のグループを定義する操作セットを提供します。IdML について詳しくは、『IdML のオープン関数コンポーネント』セクションを参照してください。

スイートのコンポーネント

コンポーネントのリストとその詳細を以下に示します。

IdML のオープン関数コンポーネント

IdML のオープン関数コンポーネントは、IdML ファイル (またはブック) を作成し読み取るのに使用できます。

これは、IdML スイートの最初のコンポーネントであり、IBM Security Directory Integrator 関数コンポーネント、コネクター、パーサーのセットです。この関数コンポーネントの特定の使用は、以下の 2 つのポイントに要約されます。

- スイートのその他のコンポーネントがアクセスできるように、IdML ブックを作成して静的に共有します。別の方法として、既存のブックを取得することもできます (ただし、他のオープン IdML 関数コンポーネントで現在使用されていない場合)。このため、コンポーネントはブック名という、実際の IdML ブックをマッピングする識別子を使用することで、その他のコンポーネントがルックアップを行うことができます (コンポーネントがその名前を認識している場合)。初心者ユーザーのため、値が関数コンポーネントの構成パネル (またはその出力マップ) で設定されていない場合は、デフォルトのブック名 (空ストリング) が使用されます。

  IdML のオープン関数コンポーネントのみがブックを共有できるため、ブックが必要なくなった場合、静的マップからのブックの開放も行います。データ破損を防ぐため (同じブックで作業する 2 つの IdML のオープン関数コンポーネントの場合)、このコンポーネントは関連のブックに排他ロックをかけることで、そのブックが開放されるまで、その他の IdML のオープン関数コンポーネントによる使用を防ぎます。

- 取得した IdML ブックを開きます。コンポーネントは、ブックへのアクセスを受け付けると、そのブックを開こうとします。その他のコンポーネントで使用された一部の構成パラメーターを設定し、IdML 文書の見出しを書き込みます。IdML は XML 派生言語のため、以下に示すように、統一スキーマに従います。

```
<?xml version="1.0" encoding="UTF-8"?>
<idml:idml xmlns:idml="http://www.ibm.com/xmlns/swg/idml"
  xmlns:cdm="http://www.ibm.com/xmlns/swg/cdm"
  xmlns:xsi"http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/xmlns/swg/idml idml.xsd">
  <idml:source.IdMLSchemaVersion="0.8">
   <cdm:process.ManagementSoftwareSystem id="id_value" CDMSchemaVersion="2.9.3" >
    <cdm:MSSName>ibm-cdm:///CDMMSS/identifier</cdm:MSSName>
    <cdm:Label>label_value</cdm:Label>
    <cdm:ProductName>product_value</cdm:ProductName>
    <cdm:ManufacturerName>manufacturer_value</cdm:ManufacturerName>
   </cdm:process.ManagementSoftwareSystem>
  </idml:source>
  <idml:operationSet opid="1">
   OPERATIONS
   ...
   ...
  </idml:operationSet>
</idml:idml>
```

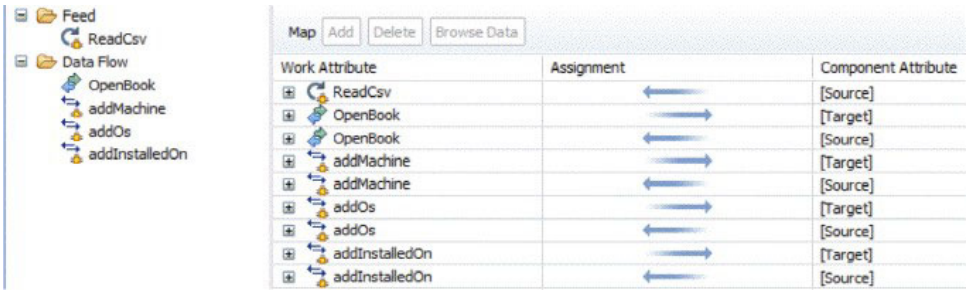
  各 IdML ブックには 2 つのパートがあり、その 1 つは idml:source (見出しとしても参照されます) で、もう 1 つは 1 つ以上の idml:operationSet です。

  source パートは、(operationSet パートで) IdML にリストされたリソースを管理する MSS (管理ソフトウェア・システム) の識別情報を提供するように設計され、もう 1 つの operationSet はこれらのリソースで何を実行すべきかを判断します。オープン IdML 関数コンポーネントは、ブックを開くときに、IdML に MSS データ (名前、ラベル、メーカーなど) を追加する役割があります。IdML のオープン関数コンポーネントが既に開いているブックを取得している場合、MSS のソース情報の追加は行われませんが、代わりにユーザーへのメッセージが記録され、AssemblyLine 内の次のコンポーネントへ実行を渡します。

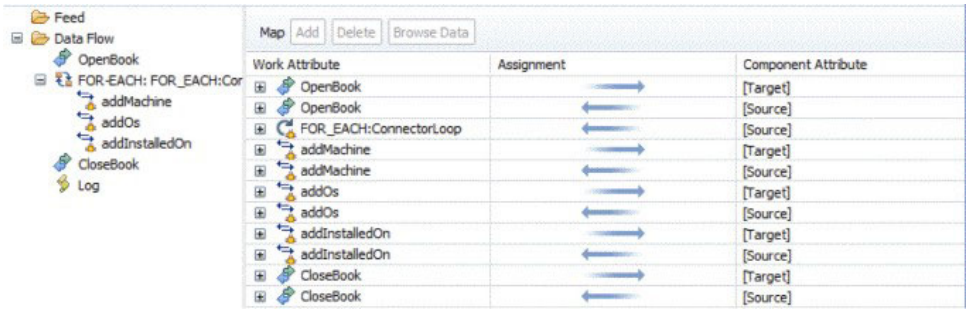
この関数コンポーネントの追加機能は、開いている IdML ブックが閉じられるように保証することです (終了タグ </idml:idml> が追加されます)。デフォルトで、これは 651 ページの『IdML のクローズ関数コンポーネント』の機能ですが、通常のフィード/フロー AssemblyLine の場合、IdML のオープン関数コンポーネントによる実行が可能です。

実際に、IdML スイートが参加できる AssemblyLine のタイプには以下の 2 つがあります。

1. 通常のフィード/フロー AssemblyLine。例えば、データは AssemblyLine の「フィード」セクションにイテレーター・モードのコネクターから読み込まれ、操作の「フロー」へ渡されます (下の図を参照してください)。「フロー」セクション内のコンポーネントはすべて繰り返し実行されるため、IdML のクローズ関数コンポーネントの使用が許可されず、各パス上のブックを閉じようとします。したがって、そのタスクは繰り返し実行できるため、IdML のオープン関数コンポーネントによって実行されます。その結果、それが最初の AL の反復中に IdML ブックを開き、続いてユーザーへの情報メッセージのみを記録し、AL が終了するとブックを閉じます。



2. ループ型の AssemblyLine。ループ・コンポーネントは、コネクターによってフィードされ、ループ範囲のコンポーネントについて繰り返し反復して、コネクターから読み取られるさまざまなデータを各コンポーネントに渡します。この場合、AssemblyLine は反復のみを実行するだけですが、ループ内部のコンポーネントは何度も実行されます。ここで、IdML ブックのクローズが IdML のクローズ関数コンポーネントによって実行され、生成されたブックに関する追加情報がユーザーに提供されます (例えば、IdML ファイルまたはそのコンテンツへの絶対パス)。



オープン IdML 関数コンポーネントは、IdML の以下の 2 つの記憶領域タイプをサポートします。

- ファイルとして保管される標準の IdML ブック。この方法では IdML がメモリー内で累算されないため、メモリー効率がより高くなりますが、ファイルに直接フラッシュされます。
- IdML スニペット。メモリーに保管される有効な IdML 文書です。この方法は、IdML ファイルをスキップして、そのコンテンツを直接戻します。この方法は、ファイルに保管するよりもメモリー消費量が多くなります。

IdML のオープン関数コンポーネントの構成に応じて、いずれか 1 つを生成できます (保管されるファイルの場合、ユーザーは、ファイルが保管される場所または現在のソリューション・ディレクトリーが使用される場所を指定する必要があります)。

前述のとおり、関数コンポーネントは拡張構成オプションとして、デフォルトのブックが使用されないように、カスタムのブック名を受け入れます。これは、デフォルトのブックが既に使用中の場合、特に便利です。次に、使用する IdML コンポーネントに別の名前を指定するだけで、AssemblyLine を実行できます。さらに、使用されたブックの名前を関数コンポーネントの出力マップで指定変更することで、ランタイム時に新しい名前を使用することができます。これは、 AL の「フロー」セクション (上記を参照) がさまざまなユーザーからの要求 (例えば、一部のサーバ

ー・コネクター経由) に応答している場合にとても役立ちます。このような場合、デフォルトのブックを使用して数人のユーザーが IdML を要求すると、最初の要求だけが成功して、その他の要求は成功した要求が終了するまで待機するか、エラー・メッセージを受け取ります。この問題を解決するため、ユーザーに IdML スイートによってブック名として使用される固有 ID をその要求に入力するよう依頼することができます。

もう 1 つの拡張オプションは、通常のデルタ IdML ではなく、リフレッシュ IdML を生成することです。IdML 用語には、以下の 2 つの IdML のタイプがあります。

- デルタ IdML。ブックのインポートにおいて、IdML にリストされた操作が、構成管理データベース、すなわち CMDB (例えば、Tivoli Application Dependency Discovery Manager) の既存のデータのみを修正することを意味します。したがって、文書にはデータベースに追加されるリソース (CREATE 操作)、データが更新されるリソース (MODIFY 操作)、および削除するリソース (DELETE 操作) を指定できます。
- リフレッシュ IdML。上記のデルタ IdML と異なり、CMDB へのリソースの追加だけが可能であることを意味する CREATE 操作だけが、このタイプに含まれます。データベース内の既存のリソースがクリアされ、IdML からの 1 つのリソースだけが残される点が異なります。

また、関数コンポーネントは、DL 認証ツールを使用して、ファイルまたはメモリー内のいずれかで生成された IdML を認証できます。

この関数コンポーネントのもう 1 つの詳細は、Common Data Model (CDM) の処理方法です。IdML に MSS の情報を追加するため、基本的に MSS が通常の構成アイテムであるため、コンポーネントで作業しているユーザーは、必要な MSS 属性の名前を把握しておく必要があります (例えば、`cdm:MSSName` および `cdm:Hostname` など)。この情報は、CDM によって指定されます。実際には、以下に示す CDM メタデータの 2 つの独立したソースが、このコンポーネントでサポートされます。

- IBM Security Directory Integrator に同梱の JAR ファイルのフォームのローカル・コピー (`idml_cert.jar`)。そこに、CDM クラス名および属性が IdML コンポーネントによって解釈可能な Java クラスとして保管されています。
- 関数コンポーネントによる接続および必要な属性名の取得が可能なリモート IT レジストリー システム。この方法が選択された場合、ユーザーは、すべての AL にアクセス可能な `etc/it_registry.properties` ファイルで指定された一般的な IT レジストリー の信任状を使用、またはそのソリューション専用のカスタム信任状を設定できます。 IT レジストリーのプロパティーについて詳しくは、 668 ページの『it_registry.properties ファイル』を参照してください。

制約として、 `ManagementSoftwareSystem` クラス (別名は識別属性) の任意の命名規則で必要とされることを示さずに、すべての属性が表示されることが重要です。使用を容易にするために、IdML のオープン関数コンポーネントは、マップする属性 `cdm:MSSName`、またはその属性がない場合は `cdm:Hostname`、`cdm:Manufacturer`、および `cdm:ProductName` の組み合わせを予測します (`process.ManagementSoftwareSystem` クラスの 2 つの命名規則に必要な識別属性があります)。いずれもない場合、例外がスローされます。

IdML のオープン関数コンポーネントで必要とされる CDM 属性を処理する場合、いくつかの重要なポイントがあります。*CDM Version* パラメーターは、IdML の生

成時に使用される CDM メタデータを指定します。したがって、その値を設定するために提供されたボタンを使用するのが最良の方法です。
process.ManagementSoftwareSystem の使用可能な CDM 属性は、スキーマ照会機能を使用したときにリストされます。そのほとんどは MSS のプロパティーを指定しますが、その他の面においても影響を与えます。例えば、$id 属性は、アプリケーション・コードとホスト名の属性を連結することによって形成されるデフォルトの MSS ID を指定変更し、生成された IdML ファイルの名前を変更します。

また、cdm:MSSName 属性が明示的に指定されない場合、 MSS の固有名の設定に cdm:Hstname、cdm:ManufacturerName、および cdm:ProductName が使用されます。いずれの属性も指定されない場合、コンポーネントによって例外がスローされます。

最後に、cdm:SourceToken 属性を使用して、作成された MSS のソース・トークンを指定できます。これにより、MSS との通信方法を指定できます (例えば、MSS への接続および追加情報の照会に使用できる HTTP URL を組み込むことができます)。

スキーマ

IdML のオープン関数コンポーネントのスキーマについて、以下に記載します。

出力スキーマ

$idmlBookName
: このコンポーネントで使用されるブック名を指定変更するには、この属性を使用します。

applicationCode
: この関数コンポーネントの構成パネルから「**アプリケーション・コード**」パラメーターの値を指定変更するには、この属性を使用します。その値は、このパネルまたは「applicationCode パネル」パラメーターのいずれかで指定される必要があります。いずれも指定されない場合、生成された IdML ファイルの名前 (ファイルとして保存された場合) はアプリケーション・コードではなく空ストリングで始まります。

MSS の CDM 属性
: これらの名前は、使用された CDM から取得されます。有効な IdML 文書を作成するには、cdm:MSSName 属性、または cdm:Hostname、cdm:ProductName、および cdm:ManufacturerName の組み合わせのいずれかをマップする必要があります。

構成

ここに記載されているパラメーターを使用することで、IdML のオープン関数コンポーネントを構成できるようになります。

IdML の保管
: ドロップダウン・リストで、この関数コンポーネントによって開かれたブックの保管メカニズムを決定します。

ディレクトリー名
: IdML ブックが保管されるディレクトリーの名前。 ブックがファイルとし

て保管される場合にのみ適用されます。フィールドがブランクのままの場合、ソリューション・ディレクトリーが使用されます。

アプリケーション・コード
IdML の先頭部分に登録される MSS のアプリケーション・コード。 これは、コンポーネントの出力マップで指定することもできます (その場合、この値は指定変更されます)。

ホスト名
IdML の先頭部分に登録される MSS のホスト名。 これは、コンポーネントの出力マップで指定することもできます (その場合、この値は指定変更されます)。

CDM バージョン
IdML によって使用される Common Data Model バージョン。 この形式は <version>.<release>.<modifier> です。バージョンを確認するには、フィールドの横にあるスクリプト・ボタンを使用します。

ブック名
このコンポーネントによって開く IdML ブックの名前。 ブランクのままにすると、デフォルトの IdML ブックが開かれます。

リフレッシュ
これがリフレッシュ IdML であるか、デルタ IdML であるかを決定します。デフォルトではチェックは外されています (*false*)。

妥当性検査
生成された IdML に対して妥当性検査ツールを使用可能にします。デフォルトではチェックは外されています (*false*)。

CDM に IT レジストリーを使用
関数コンポーネントが、CDM メタデータについて IT レジストリーに依存するかどうかを決定します。デフォルトではチェックされています (*true*)。

JDBC URL
IT レジストリー・データベースへの接続に使用される JDBC URL。 JDBC パラメーターをすべて指定すると、「**テスト接続**」ボタンを使用して IT レジストリー・データベースへの JDBC 接続をテストできます。

JDBC ドライバー
IT レジストリー・データベースへの接続に使用されるデータベース・ドライバー。

ユーザー名
IT レジストリー・データベースへの接続時に使用されるユーザー名。

パスワード
IT レジストリー・データベースへの接続時に使用されるパスワード。

関連情報

IdML のオープン関数コンポーネントは、CDM メタデータ処理用のパッケージを公開します (`com.ibm.di.fc.idml.md`)。詳しくは、Javadocs を参照してください。

IdML のクローズ関数コンポーネント

既に開いている IdML ブックのクローズには、このコンポーネントを使用します。

ブックが既に閉じていた場合は、ユーザーへのメッセージが記録されます。閉じる手順が正常に実行されると、IdML のクローズ関数コンポーネントは $idmlBook 属性でブックの追加情報を提供できるようになり、ブックは静的に共有されなくなります。IdML 文書への絶対パス (ファイルとして保存されている場合)、または実際のコンテンツ (メモリー内の IdML スニペットの場合) のいずれかが組み込まれます。

IdML スイートからのすべてのコンポーネントのように、この関数コンポーネントは「ブック名」パラメーターを受け入れ、その値に応じて異なるブックをルックアップします。使用されるブック名も、$idmlBookName 属性によってランタイム時に指定変更できます。

IdML のクローズ関数コンポーネントは、反復するたびに IdML ブックを閉じようとするため、通常のフィード/フロー AssemblyLines で使用できません。詳しくは、645 ページの『IdML のオープン関数コンポーネント』のセクションを参照してください。

スキーマ

IdML のクローズ関数コンポーネントのスキーマについて、以下に記載します。

出力スキーマ

$idmlBookName
このコンポーネントで使用されるブック名を指定変更するには、この属性を使用します。

入力スキーマ

$idmlBook
生成された IdML のタイプに応じて、この属性は IdML ファイルへの絶対パス (ファイルとして保存された IdML の場合)、または IdML の完全なコンテンツ (メモリー内の IdML スニペットの場合) のいずれかが組み込まれます。

構成

IdML のクローズ関数コンポーネントの構成には、以下に記載するパラメーターを使用します。

ブック名
このコンポーネントによってクローズする IdML ブックの名前。ブランクのままにすると、デフォルトの IdML ブックがクローズされます。

IdML の分割関数コンポーネント

IdML の分割関数コンポーネントは、生成される IdML ファイルのサイズを制限する必要がある場合に使用できます。

したがって、ファイルとして保存された IdML 文書にのみ適用されます。メモリー内の IdML ブックの場合、その使用は例外の原因となります。

関数コンポーネントは、条件のいずれかが満たされたときは IdML 文書のクローズを試み、次のデータが保存される新しい文書を開きます。これらの操作は 1 つのブックのコンテキストで実行されるため、ユーザーからは、1 つのブックが繰り返しスムーズに表示されるように見え、複数のブックが開いて閉じられるようには見えません。作成される IdML ファイルは、それぞれが有効な IdML 文書となり、MSS データの見出しが組み込まれます。IdML ファイル名は現在時刻を使用して生成されるため、ローリングされるファイルの名前パターンを指定する必要はありません。

関数コンポーネントが IdML ブックをローリングした場合、クローズされたファイルの名前が、その入力マップの属性 (`$idmlFileName`) として戻されます。そうでない場合、この属性の値は NULL となります。IdML ブックが IdML のオープン関数コンポーネントによって「*妥当性検査*」オプションで構成された場合、ローリングされるファイルはそれぞれクローズ時に検証されます。

IdML のローリング関数コンポーネントは、以下に示す 2 つの条件に基づいてその機能の実行を試みます。

- 成果物カウント: IdML 文書に追加された構成アイテム (CI) または関係の数。このカウントに到達 (または超過) した場合、ローリングが実行されます。

  注: MSS 自体は、この関数コンポーネントによって CI と見なされません。
- ファイル・サイズ: 指定のサイズに到達した IdML ファイルのキロバイト (KB) での実際のサイズ。

  注: ファイルのクローズによっていくつかのタグを追加する必要があるため、結果ファイルのサイズがこの値を数バイト超過する可能性があることに注意してください。

上記条件のデフォルト値は 0、つまり成果物カウントおよびファイル・サイズは無制限です。ローリング条件に特定の値を指定せずに、このコンポーネントを AssemblyLine に追加する場合、関数コンポーネントは生成された IdML ファイルのローリングを試行しません。

IdML スイートからのすべてのコンポーネントのように、この関数コンポーネントは「ブック名」パラメーターを受け入れ、その値に応じて異なるブックをルックアップします。使用されるブック名も、`$idmlBookName` 属性によってランタイム時に指定変更できます。

スキーマ

IdML の分割関数コンポーネントのスキーマについて、以下に記載します。

出力スキーマ

$idmlBookName
このコンポーネントで使用されるブック名を指定変更するには、この属性を使用します。

入力スキーマ

$idmlFileName
この関数コンポーネントがその機能を実行した場合、この属性は既にクロー

ズ済みの IdML フルパスを組み込みます。そうでなく、ローリングが発生しない場合、この属性は *NULL* 値となります。

構成

ここに記載されているパラメーターを使用することで、IdML の分割関数コンポーネントを構成できるようになります。

成果物カウント

1 つの IdML ファイルに保管する成果物の最大数を決定します (0 は無制限と見なされます)。

ファイル・サイズ (KB)

1 つの IdML ファイルに書き込む成果物の最大キロバイト数を決定します (0 は無制限と見なされます)。

ブック名

このコンポーネントがローリングする IdML ブックの名前。ブランクのままにすると、デフォルトの IdML ブックが使用されます。

IdML CI および関係コネクター

このコネクターは、成果物 (構成アイテム (CI) と関係のいずれか) を IdML ブックに追加するのに使用できます。

したがって、ユーザーはこの構成で目的の成果物タイプおよび追加されたアイテムのクラスを指定する必要があります。このタスクを容易にするため、ユーザーは使用されている CDM でサポートされるクラス名を直接検出することができます。

IdML のオープン関数コンポーネントのように、CDM メタデータをローカル JAR ファイルから取得、またはリモートの IT レジストリー・システムに接続して取得することができます。コネクターの構成パネルの「拡張」セクションで、こららのオプションのいずれかを選択できます。その他のユーザビリティー機能では、使用される CDM のバージョンを確認して、リモートの IT レジストリー・システムへの接続をテストできます (使用されている場合)。IT レジストリーの場合、そのシステムへの接続方法に関する情報も指定できます (例えば、JDBC URL、ドライバー、ユーザー名、およびパスワード)。これらのフィールドがブランクのままの場合、`etc/it_registry.properties` ファイルで指定したデフォルト値が使用されます。

IdML のローリング関数コンポーネントでは、以下のパラメーターが使用されます。例えば、 IT レジストリー CDM が使用されている場合で、CI の属性をリストしているとき、そのクラスの特定の属性を取得するだけでなく (JAR メタデータが使用されたまま)、その親クラスの属性も取得します。これは、JAR 定義が使用される時よりも応答が若干遅くなる原因となります。この機能は、コネクターの構成パネルに関係タイプをリストする時に最も顕著となります。IT レジストリー CDM が使用されている場合、ソースとして動作できる CI のクラス、および選択された関係のターゲットを指定することで、追加情報を確認することができます。これは、必要な関係の制限を知らない場合にとても役立ちますが、動作が非常に遅くなり、JAR CDM が使用されている場合よりも長い時間がかかります (クラスの制限がない場合は関係タイプだけがリストされます)。

命名規則の制限は、このコネクターにも有効です。選択された CI クラスの属性をすべて表示できますが、命名規則の一部である属性 (別名は識別属性)、および何の

規則かを正確に判断することはできません。したがって、CI の必須属性と一致させるには、CDM でそれらの情報を検索する必要があります。

IdML スイートのすべてのコンポーネントのように、このコネクターは「ブック名」パラメーターを受け入れ、その値に応じて異なるブックをルックアップします。使用されるブック名も、$idmlBookName 属性によってランタイム時に指定変更できます。

コネクターの出力マップで提供されるもう 1 つの重要なパラメーターは、$operation 属性です。この属性は、IdML ファイルが CMDB にインポートされた時、指定した CI/関係で実行される動作を決定します。 これは、CMDB (CREATE 操作)、更新 (MODIFY)、または削除 (DELETE) のいずれかに追加できます。CREATE、MODIFY、および DELETE (大/小文字を区別しない) のこれらの値を、$operation 属性に設定できます。使用されている IdML がリフレッシュとして開いている場合 (セクション 645 ページの『IdML のオープン関数コンポーネント』を参照してください)、CREATE 操作だけがサポートされ、別の値を渡すと例外の原因となることに注意してください。$operation 属性に値を指定しない場合、デフォルトで CREATE 値が使用されます。

IdML 操作の設定に関するその他のオプションでは、コネクターへ使用可能なデルタの作業項目が渡されます。IdML CI および関係コネクターは「デルタ認識」であるため、これは、項目に設定されるデルタ操作を解釈して、 IdML 操作にその値をマップします。マッピングは、以下のように非常に直接的です。

表 57. IdML 操作マッピングへのデルタ・コード

デルタ操作	**IdML 操作**
ADD (Entry.OP_ADD)	CREATE
MODIFY (Entry.OP_MOD)	MODIFY
DELETE (Entry.OP_DEL)	DELETE

提供されたデルタ操作は、常に $operation 属性の値を指定変更することを覚えておいてください。

このコネクターの出力マップのスキーマを照会すると、選択された CI/関係クラスの属性がリストされます。 CI の場合、その属性には $id および cdm:SourceToken 属性が含まれます。 $id では、CI の ID のデフォルト値を指定変更して、カスタム値を指定できます (ID として使用可能な任意のストリング)。デフォルトの ID は整数識別子で、成果物が IdML ブックに追加されるたびに値が増加します。これにより、カスタム値を指定した場合は既存の値をオーバーラップしないようにする必要がありますが、ID の固有性は確保されます。IdML ブックに同じ ID を持つ複数の CI がある場合、 CMDB へのインポート時に問題が発生する原因となる可能性があります。ID は関係に追加する CI の決定に使用されるため、同じ ID を持つ CI が複数ある場合、破損の原因となります。そのような問題は、IdML のオープン関数コンポーネントのブック認証機能を使用可能にすることで、早期に検出することができます。

$id 属性は、別の IdML CI および関係コネクターに直接マップできるように、コネクターの入力マップによって戻され、関係を追加するように構成されます。

cdm:SourceToken 属性は、追加された CI の固有 ID の指定に使用できます。$id 属性が IdML ブックで固有である一方、cdm:SourceToken は MSS の領域全体で固有である必要があります。これは、後で使用して IT レジストリーで CI を一意的に識別および特定することができます。

IdML CI および関係コネクターを使用して関係のスキーマを照会することにより、以下に示す 2 つの属性のみが戻されます。

- source: これは、関係のソースである CI の ID です。
- target: これは、関係のターゲットである CI の ID です。

これらの属性はいずれも、関係の定義時に指定される必要があります。指定しない場合、例外がスローされます。

IdML CI および関係コネクターでサポートされる属性には、*拡張属性*という 1 つの追加タイプがあります。通常の属性とは対照的に、これらは IdML XML 文書内の成果物のエレメントのサブエレメントに保存され、各構成アイテムの追加情報を提供するために使用されます。これらは、CI の CDM スキーマに準拠する必要がないため、ユーザーはこれらを使用して特定のデータをマップすることができます。そのような属性は、テンプレート cdm:extattr:AttributeName に続けられ (例えば、 cdm:extattr:Testing)、自身の名前で識別されます。

IdML CI および関係コネクターは、IdML スニペットの処理時に追加機能を提供します。これは、 HTTP サーバー・コネクターを使用して、生成された IdML を送信するよう AssemblyLine が構成されている場合、特に便利です。通常は、IdML のクローズ関数コンポーネントの $idmlBook 属性にマップされ、呼び出し元に戻される前に、IdML 全体を累算する必要があります。ただし、IdML CI および関係コネクターは、IdML ブックの現在のコンテンツを取得して、HTTP サーバー・コネクターを使用してそれを送信するために使用可能なメソッド resetBook() を提供します。この方法により、応答はチャンクされ、呼び出し元は IdML 全体が完了するのを待機する必要がなくなりますが、準備出来次第データが取得されます。ファイルとして保存された IdML ブックにこの方法が呼び出される場合、*NULL* が返されます。

スキーマ

IdML CI および関係コネクターのスキーマについて、以下に記載します。

出力スキーマ

$idmlBookName
: このコネクターで使用されるブック名を指定変更するには、この属性を使用します。

$operation
: この属性は、指定された CI/関係で実行される IdML 操作を決定します。値が指定されない場合は、「CREATE」が使用されます。また、タグ付きデルタの項目がコネクターに渡された場合、そのデルタ操作は $operation 属性で指定された操作を指定変更します。

CI/関係の CDM 属性
: 有効な IdML 文書を作成するため、これらの名前は使用された CDM から取得されます。

$id この属性は、CI のスキーマを照会する時のみ表示されます。IdML 文書で固有の ID が含まれます。

cdm:SourceToken
この属性は、CI のスキーマを照会する時のみ表示されます。リソースを管理する MSS の領域全体で固有の ID が含まれます。

source この属性は、関係のスキーマを照会する時のみ表示されます。これは、関係のソースである CI の ID です。

target この属性は、関係のスキーマを照会する時のみ表示されます。これは、関係のターゲットである CI の ID です。

入力スキーマ

$id この属性には、作成された構成アイテムに与えられた ID が含まれます。この値は、ブックで指定された内部カウンターからの値、またはコネクターの出力マップで指定された値のいずれかになります。コネクターが関係を作成していない場合、この属性は *NULL* 値となります。

構成

ここに記載されているパラメーターを使用することで、IdML Ci および関係コネクターを構成できるようになります。

成果物タイプ
このコネクターが IdML ファイルに追加する成果物のタイプをドロップダウン・リストで決定します。

クラス・タイプ
作成される構成アイテムまたは関係のタイプ。事前定義されたタイプの 1 つを入力するには、「**選択...**」ボタンを使用します。

ブック名
このコネクターによって使用される IdML ブックの名前。 ブランクのままにすると、デフォルトの IdML ブックが使用されます。

CDM に IT レジストリーを使用
関数コンポーネントが、CDM メタデータについて IT レジストリーに依存するかどうかを決定します。テスト接続を使用して、IT レジストリー・システムへ到達できることを検証できます。

JDBC URL
IT レジストリー・データベースへの接続に使用される JDBC URL。 JDBC パラメーターをすべて指定すると、「**テスト接続**」ボタンを使用して IT レジストリー・データベースへの JDBC 接続をテストできます。

JDBC ドライバー
IT レジストリー・データベースへの接続に使用されるデータベース・ドライバー。

ユーザー名
IT レジストリー・データベースへの接続時に使用されるユーザー名。

パスワード
IT レジストリー・データベースへの接続時に使用されるパスワード。

IdML パーサー

IdML パーサーを使用して、IdML ファイルの内容を解析することができます。

IdML のオープン関数コンポーネント、IdML CI および関係コネクター、IdMl のクローズ関数コンポーネント、および IdML の分割関数コンポーネントが、IdML 文書の作成に使用される一方、IdML パーサーを IdML 文書の読み取りのみに使用することができます。IdML ファイルおよびスニペットの処理には、XML パーサーを使用します。

IdML XML のスキーマについて詳しくは、セクション 645 ページの『IdML のオープン関数コンポーネント』を参照してください。

IdML XML の MSS セクションは、コンポーネントの最初の反復中のみ解析され、受信された MSS データは、以降の反復ごとに戻されます。反復のたびに、パーサーも単一の成果物 (CI または関係のいずれか) を operationSet セクションから読み取り、その属性を解析して、戻された項目にそれらをマップします。`cdm:Manufacturer`、`cdm:SourceToken` など通常の CDM 属性同様、パーサーは成果物の拡張属性を読み取り、それらを戻します。唯一の違いは、それらの属性が `cdm:AttributeName` ではなく、`cdm:extattr:ExtendedAttributeName` という名前になることです。

IdML パーサーに渡される入力ファイル/スニペットは、デルタまたはリフレッシュ IdML のいずれかになります。IdML タイプについて詳しくは、セクション 645 ページの『IdML のオープン関数コンポーネント』を参照してください。このタイプは、パーサーの入力マップの `$idmlType` 属性の値を決定します。サポートされる値は、DELTA および REFRESH (大/小文字を区別しない) です。

パーサーによって読み取られる成果物のタイプに応じて、`$artifactType` 属性は CI (構成アイテム用) または関係のいずれかになります。値はいずれも、大/小文字が区別されません。

同様に、クラス・タイプは入力マップの `$classType` 属性にマップされます。この属性の値は、CI の場合は`cdm:ComputerSystem`、`cdm:OperatingSystem` などになり、関係の場合は `cdm:installedOn`、`cdm:runsOn` などになります。

入力マップの `$operation` 属性の値は、 CREATE、MODIFY、または DELETE (大/小文字を区別しない) となります。この値は、成果物の読み取り元であった IdML 文書の操作エレメントに基づいて決定します。

スキーマ

IdML パーサーのスキーマについて、以下に記載します。

出力スキーマ

このパーサーには、出力スキーマがありません。

入力スキーマ

$operation

この属性は、IdML XML スキーマでの操作の値を決定するために使用されます。

$idmlType
この属性は、指定された IdML XML が通常 (デルタ) またはリフレッシュ IdML ファイルのどちらなのかを判断します。

$classType
この属性は、CI または関係の成果物のクラス・タイプを決定します。

$artifactType
この属性は、それが CI または関係のどちらなのかを判断します。

$cdmVersion
この属性は、IdML XML が使用している CDM バージョンの値を含んでいます。

$id この属性は、CI の *id* 属性の値を含んでいます。

mss.*attributeName*
MSS 属性はすべてマップされ、それらの名前には `mss` というトークンが先頭に付けられます。

cdm:SourceToken
この属性は、CI の *sourceToken* 属性の値を含んでいます。

source この属性は、関係の *source* 属性の値を含んでいます。

target この属性は、関係の *target* 属性の値を含んでいます。

構成

ここに記載されているパラメーターを使用することで、IdML パーサーを構成できるようになります。

文字エンコード
使用される文字エンコード。

詳細ログ
追加のログ・メッセージを使用可能にする場合は、このパラメーターをチェックします。

コメント
このパラメーターは、ユーザーのコメントをすべて保持できます。関数コンポーネント操作中は無効となります。

データ・クレンジング関数コンポーネント

以下に示す点については、データ・クレンジング関数コンポーネントを使用することで対処できます。

注: このコンポーネントは非推奨です。今後のバージョンの IBM Security Directory Integrator からは削除される予定になっています。

データ・クレンジング関数コンポーネントは IT レジストリーを使用して、その `$inputString` 属性として指定されたストリングの値をクレンジングします。ただし、この操作を試行する前に、この値が属している CDM 属性のタイプを指定する必要があります (例えば、`cdm:Manufacturer`、`cdm:ProductName` など)。関数コンポーネントの構成パネルにある「**選択...**」ボタンを押すと、IT レジストリーでサポートされている CDM 属性のすべてのタイプのリストを簡単に表示できます。

提供されたストリングにクレンジング規則が存在しない場合、そのストリングは関数コンポーネントの入力マップに変更されずに戻されます。

スキーマ

データ・クレンジング関数コンポーネントのスキーマについて、以下に記載します。

出力スキーマ

$cdmAttributeType
関数コンポーネントの構成パネルで指定された属性タイプの値を指定変更するには、この属性を使用します。

$inputString
この属性には、クレンジングする必要があるストリングが組み込まれます。

入力スキーマ

$cleansedString
この属性は、関数コンポーネントの出力マップで指定された `$inputString` 属性のクレンジングされた値を含みます。この属性にクレンジング規則がない場合、元のストリングが戻されます。

構成

ここに記載されているパラメーターを使用することで、データ・クレンジング関数コンポーネントを構成できるようになります。

属性タイプ
クレンジングする必要があるストリングの Common Data Model 属性タイプ。事前定義された属性タイプの 1 つの値を入力するには、「**選択...**」ボタンを使用します。

IT レジストリーの初期化関数コンポーネント

IT レジストリーの初期化関数コンポーネントは、管理ソフトウェア・システム (MSS) を IT レジストリー・データベースに登録し、その登録した MSS の GUID を入力マップ属性として返すのに使用することができます。

注: このコンポーネントは非推奨です。今後のバージョンの IBM Security Directory Integrator からは削除される予定になっています。

このコンポーネントは、CMDB 間のデータ転送に通常使用される IdML ファイルをバイパスします。IT レジストリーでバルク・ロードする必要があるそのようなファイルを作成する代わりに、このコンポーネントは IT レジストリー・データベースにそのデータを直接インポートできます。IdML スイート内の同等のコンポーネント同様、 IT レジストリーの初期化関数コンポーネントは、静的なブックの共有を使用します (この場合、IT レジストリー・データベースで実行される操作のセッションを意味します)。これを使用することにより、 IT レジストリー・コンポーネントは実行される操作に関する情報を共有します。ブックを使用しているコンポーネントによってリフレッシュ操作が実行されるかどうかを示すフラグ (つまり、それらで登録された成果物のみがデータベースに保持され、古い成果物はすべて削除されます)、および IT レジストリーの初期化関数コンポーネントが実行された時刻を

マーク付けるタイム・スタンプ (リフレッシュ中、データベースで新しい成果物から古いものを分けるのに使用されます) が含まれています。どちらのブックを使用するかを判断するには、関数コンポーネントの構成パネルにある「ブック名」フィールドの値またはその出力マップの $itRegistryBookName 属性の値のいずれかを指定します。

このコンポーネントは IT レジストリー・データベースに接続するため、etc/it_registry.properties ファイルで指定された値を使用します (JDBC URL、JDBC ドライバー、ユーザー名、およびパスワード)。これらのプロパティーをより簡単に修正するため、各コンポーネントにはその拡張構成セクションの値を上書きするためのフィールドがあります。IT レジストリーのプロパティーについて詳しくは、 668 ページの『it_registry.properties ファイル』を参照してください。

関数コンポーネントの構成パネルに表示されている「**CDM に IT レジストリーを使用**」フラグにより、ローカルの JAR システムまたは IT レジストリー・システムを使用して CDM のメタデータ定義を取得するかどうかが指定されます。

関数コンポーネントの出力マップで「接続」ボタンを押すと、その出力スキーマを取り込むことができます。この方法により、process.ManagementSoftwareSystem クラスでサポートされるすべての CDM 属性が表示され、その名前を正確に把握していなくても、これらの属性に直接マップできます。

IdML コンポーネントと同様に、コネクターのスキーマを照会する際に、すべての属性がリストされます。指定する必要がある最小限のサブセットを判別するには、コネクターの出力マップを検証します。詳しくは、 665 ページの『設計時の命名規則の妥当性検査』を参照してください。

使用を容易にするために、IT レジストリーの初期化関数コンポーネントは、マップする属性 cdm:MSSName、またはその属性がない場合は cdm:Hostname、cdm:Manufacturer、および cdm:ProductName の組み合わせを予測します (process.ManagementSoftwareSystem クラスの 2 つの命名規則に必要な識別属性があります)。いずれもない場合、例外がスローされます。

スキーマ

IT レジストリーの初期化関数コンポーネントのスキーマについて、以下に記載します。

出力スキーマ

$itRegistryBookName
: このコンポーネントで使用されるブック名を指定変更するには、この属性を使用します。

MSS の CDM 属性
: これらの名前は、使用された CDM から取得されます。有効な IdML 文書を作成するには、cdm:MSSName 属性、または cdm:Hostname、cdm:ProductName、および cdm:ManufacturerName 属性の組み合わせのいずれかをマップする必要があります。

入力スキーマ

$mssGuid

この属性は、登録された MSS の GUID を含む com.ibm.tivoli.namereconciliation.guid.Guid オブジェクトです。

構成

IT レジストリーの初期化関数コンポーネントの構成には、以下に示すパラメーターを使用できます。

CDM バージョン

このパラメーターは、version.release.modifier という形式 (ドットで区切られた 3 つの整数)で Common Data Model のバージョンを指定します。このバージョンは、IT レジストリー・データベースの CDM バージョンと一致する必要があります。CDM のバージョンが同じであれば、他の CDM リリースまたは修飾子は以前のバージョンとの互換性があります。

詳細ログ

追加のログ・メッセージを使用可能にする場合は、このパラメーターをチェックします。

コメント

このパラメーターは、ユーザーのコメントをすべて保持できます。関数コンポーネント操作中は無効となります。

ブック名

「ブック名」パラメーターは、IT レジストリー・コンポーネント間で静的に共有され、それぞれコンテキスト内で修正を実行します。各コンポーネントは、その構成でブック名を指定して (またはデフォルトのブックを使用するフィールドをブランクのままにします)、対応する MSS GUID を取得します。この属性は、関数コンポーネントの構成パネルまたは出力マップのいずれかで指定される必要があります。

リフレッシュ

これが選択された場合 (つまり、*true*)、ブック (ブック名) で IT レジストリー・データベースのリフレッシュが実行されます。デフォルトは *false* です。

CDM に IT レジストリーを使用

関数コンポーネントが、CDM メタデータについて IT レジストリーに依存するかどうかを決定します。

JDBC URL

IT レジストリー・データベースへの接続に使用される JDBC URL。

JDBC ドライバー

IT レジストリー・データベースへの接続に使用されるデータベース・ドライバー。

ユーザー名

IT レジストリー・データベースへの接続時に使用されるユーザー名。

パスワード

IT レジストリー・データベースへの接続時に使用されるパスワード。

JDBC 関連のパラメーターはすべて、そのデフォルト値を etc/it_registry.properties ファイルから派生させます。

IT レジストリー CI および関係コネクター

IT レジストリー CI および関係コネクターは、CI (構成アイテム) と CI 間の関係について追加、更新、削除、検索、または反復を行うのに使用できます。

注: このコンポーネントは非推奨です。今後のバージョンの IBM Security Directory Integrator からは削除される予定になっています。

リストされた操作をすべて実行するため、このコネクターは IT レジストリー・データベースを直接操作します。コネクターの構成パネルで「**成果物タイプ**」パラメーターを使用することで、その操作が構成アイテムまたは関係で実行されるかどうかを指定することができます。さらに、成果物の正確なクラス名を把握する必要がなく、 CDM によってサポートされているクラスを直接検出することができます (構成パネルの「**選択...**」ボタンを使用)。

IdML CI および関係コネクターのように、CDM メタデータをローカル jar ファイルから取得、またはリモートの IT レジストリー・システムに接続して取得できます。コネクターの構成パネルの「拡張」セクションで、こららのオプションのいずれかを選択できます。その他のユーザビリティー機能では、リモートの IT レジストリー・システムへの接続をテストできます (使用されている場合)。IT レジストリーの場合、IT レジストリーへの接続方法に関する情報も指定できます (例えば、JDBC URL、ドライバー、ユーザー名、およびパスワード)。これらのフィールドがブランクのままの場合、etc/it_registry.properties ファイルで指定したデフォルト値が使用されます。

選択された CDM ソースも、コネクターの出力マップおよび入力マップのスキーマ照会時に表示される属性に影響を与えます。例えば、 IT レジストリー CDM が使用されている場合で、CI の属性をリストしているとき、そのクラスの特定の属性を取得するだけでなく (JAR メタデータが使用されたまま)、その親クラスの属性も取得します。これは、JAR CDM が使用される時よりも応答が少し遅くなる原因となります。この機能は、コネクターの構成パネルに関係タイプをリストする時に最も顕著となります。IT レジストリー CDM を使用して、ソースおよび選択された関係のターゲットとして動作できる CI のクラスを指定することで、追加情報を確認することができます。これは、必要な関係の制限を知らない場合にとても役立ちますが、動作が非常に遅くなり、JAR CDM が使用されている場合よりも長い時間がかかります (クラスの制限がない場合は関係タイプのみがリストされます)。

IdML コンポーネントと同様に、コネクターのスキーマを照会する際に、すべての属性がリストされます。指定する必要がある最小限のサブセットを判別するには、コネクターの出力マップを検証します。詳しくは、 665 ページの『設計時の命名規則の妥当性検査』を参照してください。

このコネクターは「**ブック名**」パラメーターを受け入れ、その値に応じて異なるブックをルックアップします。使用されるブック名も、$itRegistryBookName 属性によってランタイム時に指定変更できます。

IT レジストリー CI および関係コネクターが CallReply モードの場合、構成アイテムおよび関係の両方とそれらの属性を登録/修正することができます。ただし、ユー

ザーが CI を登録すると、IT レジストリーが現在のリリースで非識別の属性をサポートしていないため、CI は識別属性のみを提供します。

コネクターでは、抽象リソースの登録がサポートされます。リソースは通常の CI と似ていますが、クラス・タイプ process.AbstractResource のものです。また、リソースは特定の項目の情報を保持せず、その情報が見つかる別の MSS への参照を保持します。そのような CI を登録する際には、以下を指定します。

- cdm:ExternalSystemName および cdm:ExternalIdentity - 他の MSS にリンクします
- cdm:SourceToken - MSS のレルムの CI を一意的に識別します

例えば、TADDM と TBSM の両方が、複数のコンピューター・システムの情報を受信すると、TADDM は使用可能なすべての詳細を保持し、TBSM は必要な属性のみを保持します。そのため、TBSM がコンピューター・システムの 1 つに関わる関係を IT レジストリーに登録する場合、必要な命名規則を満たさないため、操作が失敗します。この場合、TBSM は項目を抽象リソースとして登録することを選択できます。TBSM は、それを認識している別のシステムの詳細のみを指定し、関係が IT レジストリーに登録されます。最終的に、TADDM は関係を読み取る際に、CI の 1 つが抽象リソースであることを検出し、詳細が見つかる MSS として自身を認識します。TADDM は CI を見つけることができ、その関係を整合性のある方法で保管します。

CallReply モードで正しく動作するため、実行する操作に関する詳細についてこのコネクターは IT レジストリーの初期化関数コンポーネントに依存します。このデータは共有ブックを介してアクセスされ、成果物が「リフレッシュ」されているかどうかを識別するために使用されるリフレッシュ操作およびタイム・スタンプをコネクターが実行するかどうかを決定するフラグを含んでいます。リフレッシュ操作について詳しくは、セクション 645 ページの『IdML のオープン関数コンポーネント』を参照してください。

このモードの場合、コネクターの出力マップで提供されるもう 1 つの重要なパラメーターは、$operation 属性です。この属性は、IT レジストリー・データベースに登録されると、指定した CI/関係で実行される操作を決定します。 これは、IT レジストリー・データベースに追加 (CREATE 操作)、更新 (MODIFY)、または削除 (DELETE) できます。 CREATE、MODIFY、および DELETE (大/小文字を区別しない) のこれらの値を、 $operation 属性で設定できます。使用されているブックがリフレッシュとして開いている場合 (セクション 659 ページの『IT レジストリーの初期化関数コンポーネント』を参照してください)、CREATE 操作のみがサポートされ、その他の値を渡すと例外の原因となることに注意してください。$operation 属性に値を指定しない場合、デフォルトで CREATE 値が使用されます。

IT レジストリー操作の設定に関するその他のオプションでは、コネクターへ使用可能なデルタの作業項目が渡されます。IT レジストリー CI および関係コネクターは「デルタ認識」であるため、これは、項目に設定されるデルタ操作を解釈して、IdML 操作にその値をマップします。マッピングは、以下のように非常に直接的です。

表 58. IdML 操作マッピングへのデルタ・コード

デルタ操作	IdML 操作
ADD (Entry.OP_ADD)	CREATE

表 58. IdML 操作マッピングへのデルタ・コード (続き)

デルタ操作	IdML 操作
MODIFY (Entry.OP_MOD)	MODIFY
DELETE (Entry.OP_DEL)	DELETE

提供されたデルタ操作は、常に $operation 属性の値を指定変更することを覚えておいてください。

注: 現行バージョンの IT レジストリーの制約により、UPDATE 操作は IT レジストリー CI および関係コネクターではサポートされません。UPDATE を指定しようとすると、例外がスローされます。

このコネクターの出力マップのスキーマを照会すると、選択された CI/関係クラスの属性がリストされます。

IT レジストリー CI および関係コネクターを使用して関係のスキーマを照会すると、以下に示す 2 つの属性のみが戻されます。

- source: これは、関係のソースである管理対象エレメントの GUID です。
- target: これは、関係のターゲットである CI の GUID です。

いずれの属性も指定されない場合、例外がスローされます。

構成アイテムを削除するには、$operation 属性の DELETE 値とともに、一連の識別属性をこのコネクターへ渡す必要があります。構成アイテムが複数の**管理ソフトウェア・システム**で所有されている場合、これはデータベースから削除されません。その代わり、CI と**管理ソフトウェア・システム**間の関連付けのみが削除され、MSS コンテキストから解放します。同様に、指定された MSS の関係を解放/削除するには、 MSS の GUID と、ソースおよびターゲット CI の GUID が必要となります。 $classType 属性では、作成した構成アイテム/関係のタイプを実行時に変更できます。

CI および関係の読み取りを容易に行うため、イテレーター・モードが提供されています。IT レジストリー CI および関係コネクターがこのモードの場合、以下の 3 つの条件に基づいて、構成アイテムを戻します。

- *属性フィルター* – 一連のキー値のペアで、戻される CI を制限するフィルターを指定します。各ペアは、CDM 属性およびその必要な値を表します。したがって、これらの属性およびその値を持つ CI のみが、コネクターによって戻されます。フィルターが空の場合、CI はすべて取り出されます。キー値のペアはコンマで区切られ、代入演算子「=」を使用します。例: cdm:Model=T61p、cdm:Manufacturer=IBM、cdm:Fqdn=www.ibm.com。
- *日付フィルター* – 短形式で有効な日付を予測するテキスト・フィールド (例えば、US ローカルの場合、MM/DD/YY となります)。「日付フィルター」が空の場合、エラーがスローされます。

  注: 「属性フィルター」または「日付フィルター」のいずれかを、単一のコネクターで使用できます。デフォルトでは「属性フィルター」が使用可能となっていますが、「**日付フィルターを使用可能にする**」オプションをチェックすることで、「日付フィルター」に切り替えることができます。

- *MSS 名* – MSS の名前 (またはそのホスト名、メーカー、および製品名の組み合わせ) を受け入れるテキスト・フィールド。デフォルトで、このフィールドはブランクのままになっています。つまり、その他のフィルター基準と一致するすべての構成アイテムおよび関係は、それらが属している MSS を確認することなく戻されます。ただし、このフィールドに MSS 名を入力すると、その管理ソフトウェア・システムの成果物のみが戻されます。MSS 名を入力する代わりに、「**MSS 名の取得**」ボタンを押して IT レジストリー・データベースにある MSS をすべて表示し、必要なもののみを直接選択することもできます。

  注: このパラメーターは、「属性フィルター」および「日付フィルター」に関係なく使用できます。
- *isDeleted* - classType または日付フィルターに基づいて、削除された CI を返します。
- *有効範囲* - クラスのみを返す必要があるのか、そのサブクラスも返す必要があるのかを暗黙指定します。

ルックアップ・モードを使用して、IT レジストリー・データベースに保存されているものと同じ識別属性をすべて持つ構成アイテムまたは関係を戻すことができます。検索はコネクターのリンク基準で指定された条件を使用して実行され、条件が具体的でない場合、複数の構成アイテムが戻される場合があります。この場合、「**複数項目時**」フックを使用可能にして、その状態を処理するためのカスタム・ロジックをいくつか追加する必要があります。

複雑なリンク基準を指定する場合、条件の間には必ず AND 論理演算のみを使用して、EQUALS 演算子のみを使用してください (例えば、「`cdm:Model=T61p AND cdm:Manufacturer=IBM`」は有効な基準です)。イテレーター・モードの場合同様、「MSS 名」フィルターに値を指定することで、戻される CI を制限できます。

コネクターがルックアップ・モードまたはイテレーター・モードで動作している場合、`$classType` 属性には、現在読み取られている構成アイテムまたは関係のタイプが含まれます。データを IT レジストリーから読み取る際に、すべての項目または関係が反復されます。これは、イテレーター・モードまたはルックアップ・モードで、構成エディターの「**クラス・タイプ**」フィールドを空のままにすることで実現されます。この機能を使用して、IT レジストリー・データベース内のすべての構成アイテムを反復したり (例えば、一括エクスポート)、すべての項目/関係に対して (特定のソース GUID を持つ関係を返す) 検索を実行したりすることができます。

設計時の命名規則の妥当性検査

設計時の命名規則の妥当性を検査するには、以下に示す情報を使用します。

IT レジストリーの初期化関数コンポーネントおよび IT レジストリー・コネクターでは、選択した CDM クラス・タイプの命名規則に照らした、出力マップでマップされる属性の妥当性検査がサポートされます。妥当性検査を行うには、コンポーネント (例えば、データベース・パラメーター) を構成し、CI の「**クラス・タイプ**」を指定する必要があります。

注: DIS 1.2 ドライバーの制限のため、妥当性検査が動作するようにするには、構成エディターで「CDM に IT レジストリーを使用」オプションを使用不可にしてください。

構成エディターで、コネクターの出力マップにある「**妥当性検査**」ボタンをクリックすると、すべての属性が検証され、結果が「**問題**」ビューに表示されます。指定したクラスの命名規則ごとに 1 つのメッセージがあります。少なくとも 1 つの命名規則が満たされると、「**問題**」ビューのすべてのメッセージに情報のマークが付きます。どの命名規則も満たしていない場合は、メッセージにエラーのマークが付きます。各メッセージは、規則を満たすために追加または削除する必要がある属性などの詳細を示します。

スキーマ

IT レジストリー CI および関係コネクターのスキーマについて、以下に記載します。

出力スキーマ

$itRegistryBookName
このコンポーネントで使用されるブック名を指定変更するには、この属性を使用します。

$classType
この属性を使用して、コネクターの UI で構成される、作成した項目/関係のデフォルト・クラス・タイプをオーバーライドできます。

$operation
この属性は、指定された CI/関係で実行される IT レジストリー操作を決定します。値が指定されない場合は、「CREATE」が使用されます。また、タグ付きデルタの項目がコネクターに渡された場合、そのデルタ操作は `$operation` 属性で指定された操作を指定変更します。

CI/関係の CDM 属性
有効な CI を作成するため、これらの名前は使用された CDM から取得されます。

$mssGuid
この属性は、CI/関係の登録に使用される MSS の GUID を保持します (この方法により、成果物は管理ソフトウェア・システムと関連付けられます)。これは、`com.ibm.tivoli.namereconciliation.guid.Guid` オブジェクトです。

入力スキーマ

$guid この属性には、作成された構成アイテムに与えられた GUID が含まれます。これは、`com.ibm.di.fc.itregistry.ConfigurationItemId` ラッパー、`com.ibm.tivoli.namereconciliation.guid` GUID、またはそのストリング表現を受け入れます。その値は、構成アイテムの作成時にのみ生成され、関係の場合の値は *NULL* となります。

$classType
読み取られた構成アイテムまたは関係のクラス・タイプ。

$managementSoftwareSystem
この属性には、現在の CI と関連付けられた管理対象ソフトウェア・システムの詳細が含まれます。これは HashMap の配列です。

CI/関係の CDM 属性
これらの名前は、構成アイテムの反復またはルックアップ時に (イテレーター・モードおよびルックアップ・モードで)、使用される IT レジストリーから取得されます。

$aliasGuid
現在の CI の別名 GUID の配列。

構成

ここに記載されているパラメーターを使用することで、IT レジストリー CI および関係コネクターを構成できるようになります。

成果物タイプ
このコネクターが IT レジストリーデータベースに追加する成果物のタイプをドロップダウン・リストで決定します。

クラス・タイプ
作成される構成アイテムまたは関係のタイプ。事前定義されたタイプの 1 つを入力するには、「**選択...**」ボタンを使用します。

ブック名
このコネクターによって使用される IT レジストリー・ブックの名前。 ブランクのままにすると、デフォルトの IT レジストリー ブックが使用されます。

MSS 名
IT レジストリー・データベースにある管理ソフトウェア・システムの名前。存在しない場合は、メーカー名、製品名、およびホスト名の組み合わせが表示されます。「**選択...**」ボタンを使用すると、事前定義された名前から選択して入力できます。

CDM に IT レジストリーを使用
関数コンポーネントが、CDM メタデータについて IT レジストリーに依存するかどうかを決定します。IT レジストリーまたは JAR ファイルによって提供される CDM のバージョンを確認するには、「**CDM バージョンの取得**」ボタンを使用します。

JDBC URL
IT レジストリー・データベースへの接続に使用される JDBC URL。 JDBC パラメーターをすべて指定すると、「**テスト接続**」ボタンを使用して IT レジストリー・データベースへの JDBC 接続をテストできます。

JDBC ドライバー
IT レジストリー・データベースへの接続に使用されるデータベース・ドライバー。

ユーザー名
IT レジストリー・データベースへの接続時に使用されるユーザー名。

パスワード
IT レジストリー・データベースへの接続時に使用されるパスワード。

日付フィルターを使用可能にする
このチェック・ボックスは、IT レジストリー・データベースから CI を取

り出すために、「**日付フィルター**」または「**属性フィルター**」を使用するかどうかを決定します。これをチェックした場合、「日付フィルター」が使用可能になります。

日付フィルター
変更日付がフィルターよりも新しい構成アイテムのみが返されます。フィルターの形式は、(M/D/yy h:mm a) です。

属性フィルター
これは、戻された構成アイテムのフィルター処理に使用される識別属性のリストです。CI の場合は命名属性、関係の場合はソースおよびターゲット属性を入力します。属性は、「,」で区切る必要があります。

isDeleted
このパラメーターを使用して、削除された構成アイテムをフィルターに掛けます。

有効範囲
このパラメーターを使用して、構成アイテムに基づいて、クラスおよびサブクラスを制限します。

it_registry.properties ファイル

it_registry.properties ファイルの使用法は、以下に示す情報を参照することで把握できます。

IdML コンポーネント、IdML のオープン関数コンポーネント、IdML CI および関係コネクターは、必要な CDM メタデータを取得するため、IT レジストリー・システムへの接続方法に関する情報を必要とします。同じ情報が IT レジストリー・コンポーネント、IT レジストリーの初期化関数コンポーネント、IT レジストリー CI および関係コネクターでも必要とされ、IT レジストリー・データベースの情報の登録や CDM メタデータの取得に使用されます。したがって、これらのコンポーネントはすべて、そのパネルに構成フィールドが組み込まれ、IT レジストリー情報を指定することができます。

ただし、ほとんどの場合、1 つの IT レジストリー・システムのみを常に使用できるため、このデータに共通の場所がある場合はさらに容易になります。したがって、IBM Security Directory Integrator は、「JDBC URL」、「JDBC ドライバー」、「ユーザー名」、および「パスワード」プロパティーを組み込みこむことができ、これらがデフォルト値としてすべての IdML および IT レジストリー・コンポーネントで使用されるプロパティー・ファイル「*TDI_solution_dir*/etc/it_registry.properties」を提供します。

it_registry.properties ファイルの形式は以下のとおりです。

```
it_registry.jdbcUrl=
it_registry.jdbcDriver=
it_registry.dbUsername=
it_registry.dbPassword=
```

これらのプロパティーのいずれかに別の値が必要な場合は、it_registry.properties ファイルを編集 (すべてのコンポーネントに変更を適用)、または新しい値を必要としているコンポーネントの構成パネルでローカルにその値を設定します。

例

IT レジストリーとIdML スイートを使用した CI/関係の作成には、以下に例示する手順を使用できます。

IT レジストリー・スイートを使用して CI/関係を作成するステップは以下のとおりです。

1. IT レジストリーの初期化関数コンポーネントを追加します。これにより、 IT レジストリー・データベースに必要な MSS が登録されます。この IT レジストリーの初期化関数コンポーネントを以下のように構成します。
   - このコンポーネントで使用されている **CDM バージョン**を手動で指定、または「**CDM バージョンの取得**」ボタンを使用します。
   - 関数コンポーネントの出力マップで必要とされる CDM 属性をマップして、これらの属性に値を指定します。「`cdm:MSSName`」または「`cdm:Hostname + cdm:Manufacturer + cdm:ProductName`」のいずれかを指定する必要があることに注意してください。これらの名前を検索するには、関数コンポーネントの出力マップにある「**接続**」ボタンをクリックします。
   - 関数コンポーネントの入力マップで `$mssGuid` 属性をマップします。
   - オプションで、構成パネルの「拡張」セクションで「**ブック名**」を指定します (ブランクのままにした場合、デフォルトのブックが使用されます)。
2. *CallReply* モードで、IT レジストリー CI および関係コネクターを追加します。
3. この IT レジストリー CI および関係コネクターを以下のように構成します。
   - その構成パネルで、「**成果物タイプ**」を選択します (CI または関係のいずれか)。
   - 次に、その成果物の「**クラス・タイプ**」を選択します。サポートされるクラス・タイプをすべてリストするには、「**選択...**」ボタンを使用します。
   - IT レジストリーの初期化関数コンポーネントで指定したものと同じ「**ブック名**」を入力します (IT レジストリーの初期化関数コンポーネントに「**ブック名**」を指定していない場合は、ブランクのままにしておきます)。
   - IT レジストリーの初期化関数コンポーネントによって戻された `$mssGuid` を、 IT レジストリー・コネクターの出力マップにマップします。
   - 関数コンポーネントの出力マップで必要とされる CDM 属性をマップして、これらの属性に値を指定します。これらの名前を検索するには、コネクターの出力マップにある「**接続**」ボタンをクリックします。
   - `$operation` 属性に有効な値を指定します。サポートされる値は、CREATE、MODIFY、および DELETE (大/小文字を区別しない) です。選択された操作が CREATE の場合、CI/関係が IT レジストリー・データベースに登録されます。
   - CI が追加された場合、入力マップから **$guid** 属性をマップします (関係の場合、この属性は NULL です)。この属性は、もう 1 つの IT レジストリー CI および関係コネクターで使用され、関係が登録されます。
   - 関係が追加された場合は、必ずコネクターの出力マップでソースおよびターゲット属性をマップして、**$guid** 属性から取得できるその値に他の登録済み CI の GUID を指定してください。

IdML スイートを使用して CI/関係を作成するステップは以下のとおりです。

(IT レジストリー・スイート用のステップとの相違部分のみ説明します)。

1. IT レジストリーの初期化関数コンポーネントではなく、IdML のオープン関数コンポーネントを使用して IdML ファイルを作成します。
2. IdML のオープン関数コンポーネントは、IdML ブックを介して IdML コネクターと情報を共有します。したがって、両方のコンポーネントに同じ「**ブック名**」を指定します。
3. コンポーネントを構成するステップはどちらもとても似ています。
4. IdML のオープン関数コンポーネントの入力マップには `$mssGuid` が存在しないため、この属性をマップする必要はありません。
5. `$operation` 属性および CDM 属性の使用方法は同じです。
6. IdML CI および関係コネクターの出力は `$id` 属性 (`$guid` ではない)、つまり IdML ブックの成果物の固有 ID です。これは、その他の IdML CI および関係コネクターにマップされる `$guid` と同じ方法を使用して、関係を作成します。

注: IdML コンポーネントに依存する AssemblyLine を IT レジストリー・コンポーネントを使用して 1 つに変換する方法について詳しくは、IdML の例を参照してください。

IT レジストリー・データベースのセットアップ

IT レジストリー・データベースのセットアップの詳細を以下に示します。

前述したように、CDM コンポーネント (IdML および IT レジストリー・スイートの両方) は、 CDM メタデータの 2 つの異なるソース (ローカルの JAR ファイルおよびリモートの IT レジストリー・システム)に依存します。上級者の場合、IBM Security Directory Integrator は 3 つ目の選択、すなわちローカルの IT レジストリー・データベースのセットアップを提供します。その主な目的は、 IT レジストリー の CDM メタデータ定義のローカル・コピーを提供することにより、リモートの IT レジストリー・システムへの接続が常に必要とされないためです。ただし、その初期設定後にローカルの IT レジストリーが完全に操作可能となり、構成アイテムおよび関係の登録にも使用できるようになります。IT レジストリー 1.1 およびその使用方法について詳しくは、 641 ページの『第 6 章 資産統合スイート』を参照してください。

セットアップに必要なすべてのファイルは、IBM Security Directory Integratorに同梱のインストール DVD および eAssemblies にあります。注意が必要な前提条件は以下のとおりです。

オペレーティング・システム

IT レジストリー・データベースを作成するためのセットアップ・スクリプトは、 Maximo でサポートされるプラットフォーム上でサポートされます (以下のリストを参照)。

注: ローカルの IT レジストリー・データベースが必要な場合、以下のオペレーティング・システムのいずれかを使用する必要があります。

- AIX 5.3 (iSeries / pSeries) および AIX 5L™ 6.1 (iSeries / System p®)

- HP-UX 11i v2 (PA-RISC) および HP-UX 11i v3 (PA-RISC)
- RHEL 4 (x86-32)、RHEL 4 (zSeries /System z)、および RHEL 5 (zSeries /System z)
- Solaris 9 (SPARC) および Solaris 10 (SPARC)
- SUSE (SLES) 9.0 (zSeries /System z) および SUSE (SLES) 10.0 (zSeries /System z)
- Windows Server 2003 Datacenter Edition (オプション) (x86-32 および x86-64)、Windows Server 2003 Enterprise Edition (x86-32 および x86-64)、 Windows Server 2003 Standard Edition (x86-32 および x86-64)、Windows Server 2003 Standard x64 Edition (x86-32 および x86-64)、Windows Vista (x86-32 および x86-64)、Windows XP Professional (x86-32 および x86-64)

データベース

バージョン 1.1 の IT レジストリーは、Maximo でサポートされているデータベースをサポートします。そのため、セットアップ・スクリプトは、これらのサポートされるデータベースにのみ提供されます。サポートされるデータベースの名前およびバージョンは以下のとおりです。

- MS SQL バージョン 2008 Standard Edition または Enterprise Edition (Windows のみ)
- IBM DB2 8.2 + FP 7/1/07 以降、DB2 UDB ESE 9.1 + FP 7/1/07 以降
- Oracle 9i v2、Oracle 10 Rel1、Oracle 10 Rel2、および Oracle 11i for xSeries Linux

ローカルの IT レジストリー・データベース (適切なデータベースが既にインストールされていると仮定) をセットアップするステップは以下のとおりです。

1. IBM Security Directory Integrator のインストール DVD を開き、`it_registry_dbscript.zip` アーカイブをコピーします。このアーカイブには、IT レジストリー・データベースのセットアップに必要なファイルがすべて含まれています。
2. アーカイブを適切な場所 (例えば、`C:¥temp`) に抽出して、その内容を表示します。
3. そこで、サポートされるデータベース・タイプに対応する 3 つのアーカイブ (`disDb2Setup.zip`、`disMssqlSetup.zip`、および `disOracleSetup.zip`) を検索します。
4. 使用を計画しているデータベースのアーカイブ (例えば、`disDb2Setup.zip`) を抽出します。このアーカイブには、 `disDb2Setup.bat` (Windows システム用) および `disDb2Setup.sh` (UNIX/Linux システム用) という 2 つのセットアップ・スクリプトと、データベースのスキーマおよびその CDM メタデータ定義 (例えば、`disDb2Views.sql` および `disDb2Schema.sql`) を含む SQL ステートメント・ファイルを保持する `/sql` という名前の 1 つのフォルダーが含まれています。
5. DB2 コマンド・ウィンドウ (DB2 のインストール・フォルダーの BIN ディレクトリーにある `db2cmd.exe`) から disSetupDb2.bat(.sh) を実行します。データベースの名前 (ない場合は作成されます)、ユーザー名、およびパスワードを入力する必要があります。コマンドは以下のようになります。

```
disSetupDb2.bat –d db_name –u username –p password
```

IT レジストリー・データベースの作成中に問題が発生した場合は、スクリプトのディレクトリーの `logs/` サブフォルダーで作成されたログ・ファイルを確認します。

トラブルシューティング

資産統合スイートの処理でトラブルシューティングを実行するには、以下に示す情報を使用できます。

ロックされたブック

IdML コンポーネントを使用している AssemblyLine が異常終了して、正常にシャットダウンできない (その `terminate()` メソッドが呼び出せない) 場合、使用中だったブックが永続的にロックされる原因となります。この AL の以降の呼び出しには、必要なブックが既に使用中であるというエラー・メッセージが表示されます。この問題を解決するには、 IdML コンポーネントに新しいブック名を指定、または IBM Security Directory Integrator サーバーを再始動します。

データベースのエラー

IT レジストリー・コンポーネントでの作業中に、以下に示す下位レベルのデータベース・エラーが発生する場合があります。

```
com.ibm.tivoli.namereconciliation.common.NrsDatabaseException:
3001. An unexpected database system error has occurred.
```

表示されたメッセージは極めて一般的なため、この問題をさらにデバッグするために、IT レジストリーによるロギングおよびトレースを可能にする必要があります。これは、以下のステップを実行することで実現できます。

1. ロギング・アクティビティーを構成するためのロギング・プロパティー・ファイルを作成します。このファイルは、ロギングのセットアップに使用される標準形式に従い、必要なロギング・レベル、ハンドラー、およびハンドラー設定を指定します。以下に、ファイルの例を示します。

```
############################################################
#   Default Logging Configuration File
#
# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during VM startup.
handlers= com.ibm.tivoli.dataintegration.common.logging.
DISLogFileHandler, com.ibm.tivoli.dataintegration.common.
logging.DISTraceFileHandler
# Default global logging level. The valid logging levels are:
SEVERE (highest value), WARNING, INFO, CONFIG, FINE, FINER, FINEST
.level= FINEST
# DIS Log File Handler
# default file output is in user's home directory.
com.ibm.tivoli.dataintegration.common.logging.DISLogFileHandler.pattern
 = logs/dis%u.log
com.ibm.tivoli.dataintegration.common.logging.DISLogFileHandler.
limit = 5000000
com.ibm.tivoli.dataintegration.common.logging.DISLogFileHandler.
count = 1000
com.ibm.tivoli.dataintegration.common.logging.DISLogFileHandler.
formatter = java.util.logging.SimpleFormatter
# DIS Trace File Handler
# default file output is in user's home directory.
com.ibm.tivoli.dataintegration.common.logging.DISTraceFileHandler.
pattern = logs/dis%u.trace
com.ibm.tivoli.dataintegration.common.logging.DISTraceFileHandler.
limit = 5000000
com.ibm.tivoli.dataintegration.common.logging.DISTraceFileHandler.
count = 1000
com.ibm.tivoli.dataintegration.common.logging.DISTraceFileHandler.
formatter = java.util.logging.SimpleFormatter
# Facility specific properties.
# Provides extra control for each logger.
```

```
#com.ibm.tivoli.namereconciliation.api.level = FINE
#com.ibm.tivoli.namereconciliation.service.level = FINE
#com.ibm.tivoli.namereconciliation.service.plugins.level = FINE
#com.ibm.tivoli.namereconciliation.service.plugins.cdm.level = FINE
#com.ibm.tivoli.datacleanser.level = FINE
#com.ibm.tivoli.dataintegration.metadata.level = FINE
```

太字のパス (**logs/dis%u/log** および **logs/dis%u.trace**) に注意してください。これらのパスは、IT レジストリー・ログおよびトレース・ファイルが保存される場所とファイル名の形式を決定します。また、この構成のロギング・レベルが FINEST に設定されるため、発生しているイベントがすべて記録されます。これは、最下位レベルでのみ記録される一部のデータベース・エラーの検出に必要です。Java ロギングおよび構成ファイルについて詳しくは、http://www.oracle.com/technology/pub/articles/hunter_logging.html を参照してください。

ファイルを `dis.logging.properties` として保存し、それを IBM Security Directory Integrator のソリューション・ディレクトリーに配置します。

2. IBM Security Directory Integrator の開始時にロギング構成ファイルとして作成されたファイルを組み込むように、ibmdisrv.bat スクリプトを変更します。 Windows システムの場合、以下に示す変更を行います (ここで、**太字**のテキストは追加する必要があるものです)。

```
rem Take the supported env variables and pass them to Java program
set LOG_4J=-Dlog4j.configuration="file:etc¥log4j.properties"
set DIS_LOG=-Djava.util.logging.config.file=dis.logging.properties
set ENV_VARIABLES=%LOG_4J% %DIS_LOG%
"%TDI_JAVA_PROGRAM%" -classpath "%TDI_HOME_DIR%¥IDILoader.jar"
%ENV_VARIABLES% com.ibm.di.loader.ServerLauncher %*
```

Linux/UNIX システムの場合、変更はとても似ており、以下のように**太字**で示された変更を ibmdisrv.sh で行います。

```
# Log4j configuration file
LOG_4J=-Dlog4j.configuration=file:etc/log4j.properties
DIS_LOG=-Djava.util.logging.config.file=dis.
logging.properties
"$TDI_JAVA_PROGRAM" $TDI_MIXEDMODE_FLAG -cp "$TDI_HOME_DIR/
IDILoader.jar" "$LOG_4J" "$DIS_LOG"
com.ibm.di.loader.ServerLauncher "$@" &
```

`dis.logging.properties` へのパスは、ファイルを置いた場所に応じて異なる可能性があります。このファイルが IBM Security Directory Integrator のソリューション・ディレクトリー内に格納されている場合、必要なのはそのファイル名だけです。

3. IBM Security Directory Integrator を開始します。ログ/トレース・ファイルが、ソリューション・ディレクトリーの `/logs` フォルダーに現れます。

注: あるいは、DIS ログを IBM Security Directory Integrator サーバー・ログにリダイレクトするために、DIS ハンドラーの代わりに、com.ibm.di.log.ITRegistryHandler を構成できます。

第 7 章 スクリプト言語

ここに記載されている情報を読むことで、スクリプト言語について知ることができます。

このバージョンの IBM Security Directory Integrator で使用可能なスクリプト言語は、Rhino 互換拡張機能を持つ IBM JavaScript Engine (IBMJS)によりインプリメントされている JavaScript のみです。これまでに VBScript、PerlScript、または BeanShell を使用している場合は、JavaScript への変換が必要です。

ご使用の JavaScript コードが Rhino インプリメンテーションに追加された特定の拡張機能を使用している場合は、等価な IBM JavaScript Engine 機能にそのコードを変更する必要が生じる場合があります。例えば、`org.mozilla.javascript.Synchronizer` クラスを使用して同期化を行っていた場合は、このクラスのコンストラクターが Rhino スクリプト・エンジンに属する `org.mozilla.javascript.Scriptable` オブジェクトを必要とするため、このクラスによる同期化は動作しません。IBMJS では、**同期された**キーワードを提供します。使用方法の例については、681 ページの『メイン・オブジェクト』を参照してください。

JavaScript

JavaScript を使用する際には、考慮すべき問題があります。これには、以下のものがあります。

- 「」の『Java + Script ≠ JavaScript』
- 「」の『Java と JavaScript ストリングにおける文字/ストリングのデータ』
- 『Java と JavaScript』

Java と JavaScript

JavaScript では Java オブジェクトにアクセスできます。 IBM Security Directory Integrator の内部オブジェクトはすべて Java オブジェクトであるため、これは非常に便利です。

ただし、JavaScript で予約語または演算子として使用されている語を名前とするメソッドを持つ Java オブジェクトがあると、問題が生じます。このような場合、JavaScript インタープリターは、Java メソッドを呼び出さずに、予約語を処理しようとします。

このような例として、delete メソッドを持つ **java.io.File** クラスが挙げられます。**delete** は JavaScript の演算子でもあります。したがって、以下の呼び出しは失敗します。

```
var myFile = java.io.File("file.txt"); myFile.delete();
```

代わりに、以下のいずれかの呼び出しを実行できます。

- `myFile['delete']();`

これは、Java メソッドには配列エレメントとしてアクセスできるという事実を利用しています。

- `system.deleteFile("file.txt");`

  これは、システム・ライブラリーに **deleteFile** メソッドがあることを利用してうまく機能します。

第 8 章 オブジェクト

ここに記載されている情報を参照して、オブジェクトの詳細について理解することができます。

このセクションで説明するオブジェクトは、ご使用のインストールの *root_directory*/docs/api ディレクトリーにある Javadoc の中で詳細に説明されています。利用可能なメソッドについては、Javadoc を確認してください。Javadoc を表示するには、構成エディターで「**ヘルプ**」->「**ようこそ**」画面を選択し、「**Javadoc**」リンクを選択します。

AssemblyLine コネクター・オブジェクト

AssemblyLine コネクター・オブジェクトは、コネクター・インターフェースへの追加機能を提供するラッパーです。

コネクター・インターフェースには、AssemblyLine コネクターからコネクター・オブジェクトとしてアクセスできます。

注: AssemblyLine コネクター名を使用できるのみでなく、JavaScript コードで「thisConnector」という名前を使用して、現在実行中の AssemblyLine コネクター・オブジェクトを常に参照できます。

AssemblyLine コネクターは、AssemblyLine 内に定義するフック関数を呼び出すコネクターであると同時に、属性マッピングを行うコネクターでもあります。AssemblyLine 内の各 AssemblyLine コネクターに名前を付けると、コネクターは自動的にその名前でスクリプト内で使用できるようになります。ある AssemblyLine コネクターに **ldap** という名前を付けると、この名前は**スクリプト・オブジェクト名**としても使用されます。コネクターに名前を付けるときには、JavaScript 変数として使用できる名前を必ず指定してください。通常は、空白文字および特殊文字は使用しないでください。

AssemblyLine コネクターは、com.ibm.di.server.AssemblyLineComponent で説明されているメソッドおよびプロパティーを持っています。

属性オブジェクト

通常、属性オブジェクトは項目オブジェクトに含まれています。ここに記載されている情報を使用することで、属性オブジェクトについて詳しく知ることができます。

属性は、関連の値を持つ名前付きオブジェクトです。属性の各値は、それぞれ何らかのタイプの 1 つの Java オブジェクトに対応します。属性名に大文字小文字の区別はありませんが、名前の一部にスラッシュ (/) を含めることはできません。一部のコネクター (例えばデータベースにアクセスするものなど) では、他にも不適当と見なされる文字があるということを忘れないでください。できれば、属性名には英数字のみを使用してください。

属性マップにより属性にコネクターの値が移植された場合は、その値は、コネクターが提供した値と同じデータ・タイプになります。

詳細については、インストール・パッケージに含まれている Javadocs の資料を参照してください (com.ibm.di.entry.Attribute クラス)。

例

ここに記載されている例を参照して、属性オブジェクトを使用して実行されるさまざまなタスクについて理解することができます。

新しい属性オブジェクトの作成

ここに記載されているコード例を参照して、新しい属性オブジェクトを作成することができます。

```
var attr =  system.newAttribute("AttributeName");
```

この例では、**AttributeName** という名前の属性オブジェクトが作成され、この属性に **attr** 変数が割り当てられます。初期作成時には、この属性には値はないという点に注意してください。**attr** 変数を使用して、新規に作成したこの属性にアクセスし、対話することができます。

属性への値の追加

ここに記載されているコード例を参照して、属性オブジェクトに値を追加することができます。

```
attr.addValue("value 1");
attr.addValue("value 2");
```

この例では、**attr** 属性にストリング値 **"value 1"** および **"value 2"** が追加され、その結果複数値属性が作成されます。**addValue(obj)** を連続して呼び出すことにより、同じ順序で属性に値が追加されます。

属性の値のスキャン

ここに記載されているコード例を参照して、属性の値をスキャンすることができます。

```
var values = attr.getValues();
for (i=0; i<values.length; i++) {
  task.logmsg("Value " + i + " —> " + values[i]);
}
```

この例では、単一値か複数値かにかかわらず、属性オブジェクトが処理されます。単一値属性と複数値属性の間には、実質的な違いはありません。すべての属性は、0 個以上の値を持つことができます。したがって、単一値属性は、単に値の少ない複数値属性であると言えます。

関連情報

679 ページの『項目オブジェクト』.

コネクター・インターフェース・オブジェクト

コネクター・インターフェース・オブジェクトを取得するには、コネクター・インターフェースを明示的にロードするか (system.loadConnector)、または名前付き AssemblyLine コネクターの .connector を取得します (myConnector.connector)。ここに記載されている情報を参照して、コネクター・インターフェース・オブジェクトの詳細を理解することができます。

通常、AssemblyLine 内のスクリプトを書くときは、他のメソッドのセットにアクセスできる AssemblyLine コネクター・オブジェクトを使用します。

コネクター・インターフェースについては、Javadoc の中のコネクターに関するセクションで詳しく説明されています。詳細については、インストール・パッケージに含まれている Javadocs の資料を参照してください (`com.ibm.di.connector.Connector`)。

メソッド

よく使用されるメソッドには、以下のものがあります。

getNextEntry()
次の入力項目を戻します。

putEntry (entry)
entry を追加または挿入します。

modEntry (entry, search)
search に指定されている項目を **entry** の内容で変更します。

deleteEntry (entry, search)
search で識別される **entry** を削除します。

findEntry (search)
search で識別される項目を検索します。該当項目が見つからない場合は、**NULL** 値が戻されます。

findEntry (attribute, value)
`attribute equals value` の条件を使用して検索を行い、見つかった項目を戻します。該当項目が見つからない場合、または複数の該当項目が見つかった場合は、NULL 値が戻されます。

項目オブジェクト

項目オブジェクトは、AssemblyLine により使用されます。ここに記載されている情報を参照して、スクリプトで使用できる項目オブジェクトとグローバル項目インスタンスについて詳しく理解することができます。

項目オブジェクトは、属性とプロパティーを持つ Java オブジェクトです。属性には不特定数の値が含まれます。プロパティーには単一の値が含まれます。詳細については、インストール・パッケージに含まれている Javadocs の資料を参照してください (`com.ibm.di.entry.Entry`)。

スクリプト記述内で使用できるグローバル項目インスタンス

conn AssemblyLine 内のコネクターに含まれるローカル・ストレージ・オブジェ

クト。このオブジェクトは、コネクターの存続期間の属性マッピング・フェーズの間のみ存在します。「」の『属性マッピング』を参照してください。

work AssemblyLine のデータ・コンテナー・オブジェクト。したがって、すべてのコネクター内で、AssemblyLine からこのオブジェクトにアクセスできます。

current
更新モードおよびデルタ・モードのコネクターでのみ使用できます。このオブジェクトには、追加操作か変更操作かを判別するためにコネクターがデータ・ソースから読み取る項目が格納されます。

error 以下の属性内にエラー状況情報を保持する項目オブジェクト。

status **(java.lang.String)**
例外がまったくスローされない場合は **ok** (この場合、これがエラーの唯一の属性になります)。例外がスローされる場合は **fail** で、以下の属性も使用できます。

exception **(java.lang.Exception)**
スローされた **java.lang.Exception** (またはその後続クラス) オブジェクト

class **(java.lang.String)**
例外クラス (**java.lang.Exception** またはその後続の一部) の名前

message **(java.lang.String)**
例外のエラー・メッセージ

operation **(java.lang.String)**
コネクターの操作タイプ (例えば、AddOnly、更新、ルックアップ、イテレーターなど)

connectorname **(java.lang.String)**
フックが呼び出されたコネクターの名前

thisScriptObject
以下の属性を持つ項目オブジェクト。

AssemblyLine
実行中の AssemblyLine の名前。使用不可の場合は NULL。

Component
実行中の Component の名前。使用不可の場合は NULL。

HookName
実行中のフックの翻訳名。フックが実行されていない場合は NULL。

InternalHookName
実行中のフックの内部名。フックが実行されていない場合は NULL。

属性 マップされている属性名。属性がマップされていない場合は NULL。

マップ 属性がマップされている場合はストリング「入力」または「出力」。マップされていない場合は NULL。

関数 スクリプト・コネクター、スクリプト・パーサー、またはスクリプト記述された関数コンポーネントで呼び出される関数名。それ以外の場合は NULL。

スクリプトで使用する場合の最も簡単な方法は、以下のようになります。

```
task.logmsg("We are now executing " + thisScriptObject)
```

これにより、すべての使用可能な属性が項目に単に出力されます。

関連情報

677 ページの『属性オブジェクト』.

FTP オブジェクト

FTP オブジェクトは、スクリプト・オブジェクトとして使用できます。ここに記載されている情報を参照して、詳細な資料にアクセスすることができます。

このオブジェクトは、FTP クライアント・コネクターで必要な機能が得られない場合に役立ちます。 Javadocs にある `com.ibm.di.protocols.FTPBean` に関する詳細説明を参照してください。

例

ここに記載されているコード例を使用して、FTP オブジェクトについて理解することができます。

```
var ftp = system.getFTP();

if ( ! ftp.connect ("ftpserver", "username",
  "password") )
{
 task.logmsg ("Connect failed: " +
  ftp.getLastError());
}

ftp.cd ("/home/user1");
var list = ftp.dir();

while ( list.next() )
{
 if (list.getType() == 1)
  task.logmsg ("Directory: " +
   list.getName());
 else
  task.logmsg ("File: " + list.getName());
}

ftp.setBinary();
ftp.get ("remotefile", "c:¥¥localfile");
ftp.put ("c:¥¥localfile", "remotefile");
```

メイン・オブジェクト

ここに記載されている情報を使用して、メイン・オブジェクトとメイン・オブジェクトで使用されるメソッドについて詳しく理解することができます。

メイン・オブジェクトは、トップレベル・スレッドです (Javadocs にあるインターフェース `RSInterface` を参照)。このオブジェクトには、AssemblyLine の動作を操作するためのメソッドがあります。 最も一般的なメソッドには、以下のものがあります。

void dump(object)
ログ・ファイルにオブジェクトをダンプします。オブジェクトが**項目**オブジェクトの場合は、オブジェクトをログ・ファイルにダンプします。それ以外の場合は、クラス名と object.toString() のみをダンプします。

void logmsg (String loglevel, String msg)
`logmsg()` メソッドの代替版で、ログ・レベル・パラメーターを指定できます。ログ・レベルとして有効な値は、
「FATAL」、「ERROR」、「WARN」、「INFO」、「DEBUG」で、ログ Appender で利用可能なログ・レベルに応じて使用します。認識できない値はすべて「DEBUG」として扱われます。

startAL (name, initial-work-entry)、startAL (name, runtime-provided-Connector)、startAL (name, initial-work-entry, runtime-provided-Connector)、startAL (name, java.util.Vector)
name パラメーターに指定されている AssemblyLine を開始します。「」の『IBM Security Directory Integrator の概念』の『AssemblyLine』を参照してください。

何らかの理由で、複数の AssemblyLine 間の同期を行う必要がある場合は、IBM Javascript エンジンの synchronized キーワードを使用できます。このキーワードを使用して、一部の共通オブジェクトを同期することができます。例えば、AssemblyLine がすべて同じ構成インスタンスで実行されている場合は、以下のように `main` オブジェクトで同期することができます。

```
synchronized(main) {
  task.logmsg("Inside the synchronized block")
 }
```

検索 (基準) オブジェクト

ここに記載されている情報を参照して、検索 (基準) オブジェクトについて理解することができます。

検索 (基準) オブジェクトは、汎用検索基準を指定するために、AssemblyLine およびコネクターで使用されます。Javadocs にある `com.ibm.di.server.SearchCriteria` を参照してください。検索基準をどのように解釈してどのようにコネクター固有の検索に変換するかは、コネクターによって異なります。検索基準は非常に単純な多値オブジェクトであり、各値には、属性、オペランド、および値を指定します。

オペランド

以下のオペランドは、検索基準オブジェクトで使用するために定義されたオペランドです。

= 等しい

~ 含む

^ これで始まる

$ 終了文字

! 等しくない

例

ここに記載されているコード例を使用して、検索オブジェクトについて理解することができます。

```
for ( i = 0; i < search.size(); i++ ) {
 var sc = search.getCriteria ( i );
 task.logmsg ( sc.name + sc.match + sc.value );
}
```

shellCommand オブジェクト

ここに記載されている情報とコード例を使用して、shellCommand オブジェクトを処理することができます。

shellCommand オブジェクトには、コマンド行プロセスの結果が入ります。

Microsoft Windows プラットフォームでは、シェル・コマンドは開始されますが、そこから出力または状況を取得することはできません。使用可能なメソッドについては、Javadocs にある `com.ibm.di.function.ExecuteCommand` を参照してください。

例を示します。

```
var cmd = system.shellCommand ("/bin/ls -l");
if ( cmd.failed() ) {
 task.logmsg ( "Command failed: " + cmd.getError());
} else {
 task.logmsg ( cmd.getOutputBuffer() );
}
```

状況オブジェクト

状況オブジェクトは、AssemblyLine のコネクターとエラー・コードに関する情報のコンテナーとして使用することができます。

これは **task.getStats()** の同義語です。

システム・オブジェクト

ここに記載されている情報を使用することで、システム・オブジェクトについて詳しく知ることができます。

システム・オブジェクトは、すべてのスクリプト記述コンテキストの中でスクリプト可能オブジェクトとして使用できるもので、基本関数セットを提供します。Java オブジェクトは `com.ibm.di.function.UserFunctions` ですが、スクリプト・オブジェクト **system** にリンクされています。すべてのメソッドのリストは、Javadocs にあります。

タスク・オブジェクト

ここに記載されている情報を使用することで、タスク・オブジェクトについて詳しく知ることができます。

タスク・オブジェクトは、com.ibm.di.server.TaskInterface をインプリメントするクラスのインスタンスで、現行の実行スレッドを表します。

- AssemblyLine の場合は、これは AssemblyLine スレッドで、これにより AssemblyLine 固有の情報およびメソッドにアクセスできます。Javadocs にある com.ibm.di.server.AssemblyLine クラスを参照してください。

COMProxy オブジェクト

COMProxy オブジェクトによって、Java から COM 自動化コンポーネントを呼び出すことができます。

JNI (Java Native Interface) は、COM ライブラリーおよび Win32 ライブラリーにネイティブ呼び出しを行います。 COMProxy では、OLE (Object Linking and Embedding) オートメーションをラップ (実行時バインディングとも呼ばれる) で利用して、COM オブジェクト/インターフェースへの呼び出しを実行します。COMProxy は、Moniker URL もサポートしています。COMProxy インスタンスへのハンドルを取得するには、system.createCOMInstance( ) メソッドを使用します。

注:

1. COMProxy に関する詳細な資料は、com.ibm.di.automation パッケージの下の Javadoc にあります。
2. COMProxy オブジェクトは ADSI 呼び出しをサポートしていません。

COMProxy オブジェクトについては、例で説明します。ここで説明しているコネクターは、VBScript で記述された古い Outlook コネクターを JavaScript で再インプリメントしたものです。このコードは、*TDI_install_dir*/examples/MSOutlook ディレクトリーにあります。

以下の例は、COMProxy を使用して Outlook 連絡先を操作する方法を示しています。これは ibmdi.scriptconnector の例であり、追加、イテレーター、更新、ルックアップ、および削除の各モードをサポートするスクリプト・コネクターを作成する方法を示しています。

独自のスクリプト・コネクターを作成し、データを入力する場合のために、スクリプト・コードを以下に示します。*msoutlook.xml* ファイルは、コネクターが既に入力されている IBM Security Directory Integrator 構成ファイルです。msoutlook.xml を開くと、スクリプト情報を含む **msoutlook** というスクリプト・コネクターが見つかります (「構成」->「スクリプト」)。

継承可能なコネクターのリストに MSOutlook.jar ファイルが表示されたら、このファイルを *TDI_install_dir*//jars/connectors ディレクトリーにコピーすることもできます。

コード例

ここに記載されているコード例を使用して、ComProxy オブジェクトについて理解することができます。

```
//
// This script implements all the necessary functions for accessing
// the Contacts register in MS Outlook.
// Assumes that the number of entries in contact folder is constant for the run
```

```
ol = system.createCOMInstance("Outlook.Application");

ns = COMProxy.call(ol,"GetNameSpace","MAPI");

contacts = COMProxy.call(ns.toObject(),"getDefaultFolder",10);

var item;
var counter = 0;
var oldstring="";

var decode="";

var outlookEntry = system.newEntry();

function selectEntries(){
 counter = 0;
}

function getNextEntry (){
 ol = system.createCOMInstance("Outlook.Application");

 ns = COMProxy.call(ol,"GetNameSpace","MAPI");

 contacts = COMProxy.call(ns.toObject(),"getDefaultFolder",10);

 items = COMProxy.call(contacts.toObject(),"Items");

 count = COMProxy.get(items.toObject(),"count");

 counter++;
 if(counter > count){
  result.setStatus(0);
  result.setMessage("End of input");
 }else{
  item = COMProxy.call(items.toObject(),"Item",counter);
  populateEntry();
 }
}

function findEntry (){
 flt = "[" + search.getFirstCriteriaName() + "] = " + search.getFirstCriteriaValue();
 items = COMProxy.call(contacts.toObject(),"Items");
 item = COMProxy.call(items.toObject(),"Find",flt);
 if (item == null){
  result.setStatus(0)
  result.setMessage("Not found" + "--->["+ flt + "]");
 }
 else
  populateEntry();
}

function modEntry (){
    populateItem();
    COMProxy.call(item.toObject(),"Save");
}

function deleteEntry (){
 COMProxy.call(item.toObject(),"Delete");
}

function putEntry (){
 items = COMProxy.call(contacts.toObject(),"Items");
 item = COMProxy.get(items.toObject(),"Add");
 if(item==null){
  result.setStatus(2)
  result.setMessage("Unabled to create item");
  return;
 }
 oldString = entry.getString("FullName");
 COMProxy.put(item.toObject(),"FileAs",oldString);
 populateItem();
 COMProxy.call(item.toObject(),"Save");
}

function populateEntry (){
 entry.setAttribute("FileAs", COMProxy.get(item.toObject(),"FileAs"));
 entry.setAttribute("FullName", COMProxy.get(item.toObject(),"FullName"));
 entry.setAttribute("Email1Address", COMProxy.get(item.toObject(),"Email1Address"));
 entry.setAttribute("Birthday", COMProxy.get(item.toObject(),"Birthday"));

 entry.setAttribute("BusinessAddress", COMProxy.get(item.toObject(),"BusinessAddress"));
 entry.setAttribute("BusinessTelephoneNumber",
        COMProxy.get(item.toObject(),"BusinessTelephoneNumber"));
 entry.setAttribute("BusinessFaxNumber", COMProxy.get(item.toObject(),"BusinessFaxNumber"));
 entry.setAttribute("CompanyName", COMProxy.get(item.toObject(),"CompanyName"));
 entry.setAttribute("JobTitle", COMProxy.get(item.toObject(),"JobTitle"));

 entry.setAttribute("HomeAddress", COMProxy.get(item.toObject(),"HomeAddress"));
 entry.setAttribute("HomeTelephoneNumber", COMProxy.get(item.toObject(),"HomeTelephoneNumber"));
 entry.setAttribute("HomeFaxNumber", COMProxy.get(item.toObject(),"HomeFaxNumber"));
```

```
 entry.setAttribute("MobileTelephoneNumber", COMProxy.get(item.toObject(),"MobileTelephoneNumber"));

 entry.setAttribute("Categories", COMProxy.get(item.toObject(),"Categories"));
 entry.setAttribute("LastModificationTime", COMProxy.get(item.toObject(),"LastModificationTime"));
        outlookEntry = entry.clone(entry);
}

function populateItem (){
 outlookEntry.merge(entry);
        COMProxy.put(item.toObject(),"FileAs", outlookEntry.getString("FileAs"));
 COMProxy.put(item.toObject(),"FullName", outlookEntry.getString("FullName"));
 COMProxy.put(item.toObject(),"Email1Address", outlookEntry.getString("Email1Address"));
 COMProxy.put(item.toObject(),"BusinessAddress", outlookEntry.getString("BusinessAddress"));
 COMProxy.put(item.toObject(),"BusinessTelephoneNumber",
          outlookEntry.getString("BusinessTelephoneNumber"));
 COMProxy.put(item.toObject(),"BusinessFaxNumber",outlookEntry.getString("BusinessFaxNumber"));
 COMProxy.put(item.toObject(),"JobTitle", outlookEntry.getString("JobTitle")) ;
 COMProxy.put(item.toObject(),"CompanyName", outlookEntry.getString("CompanyName")) ;
 COMProxy.put(item.toObject(),"HomeAddress",outlookEntry.getString("HomeAddress")) ;
 COMProxy.put(item.toObject(),"HomeTelephoneNumber", outlookEntry.getString("HomeTelephoneNumber")) ;
 COMProxy.put(item.toObject(),"HomeFaxNumber", outlookEntry.getString("HomeFaxNumber")) ;
 COMProxy.put(item.toObject(),"Categories", outlookEntry.getString("Categories"));
 if (outlookEntry.getString("Birthday")!=null && !outlookEntry.getString("Birthday").equals(" "))
  COMProxy.put(item.toObject(),"Birthday", outlookEntry.getString("Birthday"));
}
```

関連情報

310 ページの『スクリプト・コネクター』.

第 9 章 IBM Security Directory Integrator スケジューラー

IBM Security Directory Integrator スケジューラーを使用して、構成情報で指定された AssemblyLine またはシーケンスを、指定された時刻に自動的に開始することができます。

構成エディターのユーザー・インターフェースを使用して、スケジューラーを作成 (「**ファイル**」 ->「**新規**」 ->「**スケジューラー**」をクリック) して、指定した時刻に AssemblyLine を実行できます。IBM Security Directory Integrator サーバーが構成ファイルをロードすると、`ibmdisrv -c myconfig.xml –d` コマンドを実行することにより、定義されたスケジューラーが開始されます (`myconfig.xml` は、スケジュールを含むエクスポートされた構成ファイルです)。構成インスタンスを停止すると、関連付けられたすべてのスケジューラーも停止します。構成エディターから、以下に示すように、スクリプトにスケジュールを含めることで、そのスケジュールを開始、停止、または一時停止できます。

- main.startScheduler("mySchedule") - mySchedule というスケジュールを開始します。
- main.shutdownScheduler("mySchedule") - mySchedule というスケジュールを停止します。
- main.stopSchedulers() - ロードされたすべてのスケジュールを停止します。

以下の 2 つのタイプのスケジューラーを定義できます。

- タイマー - 指定した時刻に AssemblyLine を開始する場合に使用します。
- キープアライブ - IBM Security Directory Integrator サーバーの始動時に AssemblyLine を開始し、その後は AssemblyLine が停止するたびに AssemblyLine を再始動する場合に使用します。

スケジュールを再始動または編集するには、構成エディター・ウィンドウの左側にあるナビゲーター・パネルの「**プロジェクト**」 ->「**リソース**」 ->「**スケジュール**」の下で、編集するスケジュールをダブルクリックします。

構成

このセクションでは、IBM Security Directory Integrator スケジューラーの構成パラメーターについて説明します。

タイマー

ここに定義されている属性を使用することで、IBM Security Directory Integrator スケジューラーを構成できます。

AssemblyLine

このパラメーターを使用して、実行する AssemblyLine またはシーケンスの名前を指定します。

使用可能な AssemblyLine またはシーケンスについてサーバーに照会するには、構成エディターで「**照会**」ボタンをクリックします。

開始時刻

このパラメーターを使用して、AssemblyLine を開始するようにスケジュールする時刻を指定します。

構成エディターで「**テスト**」ボタンをクリックすると、現行値で AssemblyLine が実行される 10 回分が表示されます。

注: AssemblyLine のインスタンスが既に稼働している場合、スケジューラーによって AssemblyLine の新規インスタンスが開始されることはありません。

AssemblyLine パラメーター

このパラメーターを使用して、初期パラメーターを AssemblyLine に指定します。

操作　このパラメーターを使用して、AssemblyLine の操作のタイプを指定します。

使用可能な AssemblyLine 操作を照会するには、構成エディターで「**照会**」ボタンをクリックします。

サーバー

このパラメーターを使用して、AssemblyLine を実行する必要があるサーバーの名前を指定します。

注: 内部サーバーの場合は `local/blank`、リモート・サーバーの場合は `hostname[:port]` を使用します。

構成インスタンス

このパラメーターを使用して、AssemblyLine が定義されているリモート・サーバーを使用する際の構成インスタンスを指定します。

障害によるスケジューラーの停止

このパラメーターを使用して、AssemblyLine が失敗した場合にスケジューラーを停止するかどうかを指定します。

失敗 AssemblyLine

このパラメーターを使用して、選択された AssemblyLine が失敗した場合に実行する AssemblyLine を指定します。

キープアライブ

ここに記載されている属性を使用することで、IBM Security Directory Integrator スケジューラーを構成できます。

AssemblyLine

このパラメーターを使用して、実行する AssemblyLine またはシーケンスの名前を指定します。

使用可能な AssemblyLine またはシーケンスについてサーバーに照会するには、構成エディターで「**照会**」ボタンをクリックします。

最小秒数

このパラメーターを使用して、AssemblyLine が実行する必要がある最小時間 (秒) を指定します。この指定時間内に AssemblyLine が終了した場合は、その AssemblyLine が失敗したものとみなされます。

失敗 AssemblyLine
このパラメーターを使用して、選択された AssemblyLine が失敗した場合に実行する AssemblyLine を指定します。

AssemblyLine パラメーター
このパラメーターを使用して、初期パラメーターを AssemblyLine に指定します。

操作 このパラメーターを使用して、AssemblyLine の操作のタイプを指定します。

使用可能な AssemblyLine 操作を照会するには、構成エディターで「**照会**」ボタンをクリックします。

構成インスタンス
このパラメーターを使用して、AssemblyLine が定義されているリモート・サーバーを使用する際の構成インスタンスを指定します。

Server このパラメーターを使用して、設計時に、リモート・サーバー上の構成インスタンスをリストするためのサーバー名を指定します。

付録 A. AssemblyLine シーケンス

AssemblyLine シーケンスを使用して、最小の条件付き機能で AssemblyLine を実行することができます。

構成エディターのユーザー・インターフェースを使用して、AssemblyLine シーケンスを作成し、実行します。AssemblyLine シーケンスを作成するには、「**ファイル**」->「**新規作成**」 ->「**シーケンス**」をクリックします。

また、以下の例に示すように、スクリプトを使用して AssemblyLine シーケンスを開始することもできます。

```
main.startSequence("MySequence");
```

ここで、`mySequence` は AssemblyLine シーケンスの名前です。

また、IBM Security Directory Integrator スケジューラーを使用して、AssemblyLine シーケンスを開始することもできます。

構成エディターで以下をクリックします。

- 「**AssemblyLine の追加**」ボタン。AssemblyLine をシーケンスに追加します。
- 「**スクリプトの追加**」ボタン。スクリプトを追加します。
- 「**削除**」ボタン。シーケンスから AssemblyLine またはスクリプトを削除します。
- 「**ログ**」ボタン。ロギングを構成するためのウィンドウを開きます。
- 「**コンソールで実行**」ボタン。構成エディターから AssemblyLine シーケンスを実行します。

シーケンスを再開または編集するには、「構成エディター」ウィンドウの左側にある「ナビゲーター」パネルで「**プロジェクト**」 ->「**リソース**」 ->「**シーケンス**」を選択し、編集する AssemblyLine シーケンスをダブルクリックします。

構成

ここに記載されているパラメーターを使用することで、AssemblyLine シーケンスを構成できるようになります。

AssemblyLine
このパラメーターを使用して、実行する AssemblyLine またはシーケンスの名前を指定します。

使用可能な AssemblyLine またはシーケンスについてサーバーに照会するには、構成エディターで「**照会**」ボタンをクリックします。

障害によるシーケンスの停止
このパラメーターを使用して、この AssemblyLine が失敗した場合にシーケンスを停止するかどうかを指定します。AssemblyLine がバックグラウンドで実行されている場合は、このパラメーターは適用外です。

JavaScript の共有

このパラメーターを使用して、JavaScript エンジンをシーケンス内の AssemblyLine で共有するかどうかを指定します。以下の場合、スクリプト・エンジンは共有されません。

- AssemblyLine がバックグラウンドで実行されている。
- AssemblyLine が別のサーバーで実行されている。

ロギングの共用

このパラメーターを使用して、シーケンスと AssemblyLine 間のロギングを共用するかどうかを指定します。

バックグラウンドで実行する

このパラメーターを使用して、シーケンス内の次の AssemblyLine に進む前に AssemblyLine が終了するまでシーケンスが待機する必要があるかどうかを指定します。

作業項目の継承

このパラメーターを使用して、バックグラウンドで実行されていない前の AssemblyLine から作業項目を継承して、現在の AssemblyLine の初期作業項目として使用するかどうかを指定します。

AssemblyLine パラメーター

このパラメーターを使用して、初期パラメーターを AssemblyLine に指定します。

操作　このパラメーターを使用して、AssemblyLine の操作のタイプを指定します。

使用可能な AssemblyLine 操作を照会するには、構成エディターで「**照会**」ボタンをクリックします。

サーバー

このパラメーターを使用して、AssemblyLine を実行する必要があるサーバーの名前を指定します。

注: 内部サーバーの場合は `local/blank`、リモート・サーバーの場合は `hostname[:port]` を使用します。

構成インスタンス

このパラメーターを使用して、AssemblyLine が定義されているリモート・サーバーを使用する際の構成インスタンスを指定します。

付録 B. パスワード同期プラグイン

IBM Security Directory Integrator を使用して構築されるパスワード同期ソリューションを使用して、複数のシステムでのパスワード変更をインターセプトできます。

IBM Security Directory Integrator には、異種ソフトウェア混合環境においてユーザー・パスワードを同期するソリューションをインプリメントするためのインフラストラクチャーと、すぐに利用可能な多数のコンポーネントがあります。

インターセプトされた変更は、以下に直接適用できます。

- 同一ソフトウェア・システム
- 異なるソフトウェア・システム集合

インターセプトしたパスワードを必要なシステムに伝搬するよう構成されている IBM Security Directory Integrator AssemblyLine により、同期が実現します。

パスワード同期ソリューションを構成するコンポーネントは、Password Synchronizer、パスワード・ストア、コネクター、および AssemblyLine です。Password Synchronizer、パスワード・ストア、およびコネクターは、IBM Security Directory Integrator に組み込まれているすぐに使用できるコンポーネントです。そのため、これらのコンポーネントをデプロイして構成すると、パスワードをインターセプトして IBM Security Directory Integrator からこのパスワードにアクセスできるようにするソリューションをインプリメントすることができます。

以降のセクションでは、現在使用可能な特殊パスワード同期コンポーネントについて説明します。

Password Synchronizer

Windows XP/Vista 用 Password Synchronizer
Windows のログイン・パスワードの変更をインターセプトします。

IBM Security Directory Server 用 Password Synchronizer
IBM Security Directory Server パスワードの変更をインターセプトします。

Sun Directory Server 用 Password Synchronizer
Sun ONE Directory Server パスワードの変更をインターセプトします。

IBM Domino 用 Password Synchronizer
IBM Notes ユーザーの HTTP パスワードの変更をインターセプトします。

UNIX および Linux 用 Password Synchronizer
UNIX および Linux のユーザー・パスワードの変更をインターセプトします。

パスワード・ストア

LDAP パスワード・ストア
インターセプトされたユーザー・パスワードを LDAP ディレクトリー・サーバーに保管するために必要な機能を提供します。

JMS パスワード・ストア
JMS パスワード・ストア (正式には MQ Everyplace Password Store) は、インターセプトされたユーザー・パスワードを JMS プロバイダーのキュー (IBM Security Directory Integrator などのあらゆる JMS クライアントが情報を読み取ることができる場所) に格納するために必要な機能を提供します。

Log パスワード・ストア
Log パスワード・ストアは、標準のパスワード・ストアが実行するすべてのアクションをログに記録するためにのみ使用されます。このパスワード・ストアは、Java プロキシーおよびネイティブ・プラグインが正常に通信していることを検証する場合に役立ちます。

専門化されたコネクター

JMS パスワード・ストア・コネクター
パスワード更新メッセージを IBM WebSphere MQ Everyplace から取得して IBM Security Directory Integrator に送信するために必要な機能を提供します。

IBM Security Identity Manager の統合
「*パスワード同期プラグイン*」では、IBM Security Identity Manager と以下の Password Synchronizer を統合するために必要なステップについても詳しく説明しています。

- Sun Directory Server Password Synchronizer
- IBM Security Directory Server Password Synchronizer
- Windows Password Synchronizer
- UNIX および Linux 用 Password Synchronizer

IBM パスワード同期プラグインのインストールと構成について詳しくは、「*パスワード同期プラグイン*」を参照してください。

付録 C. AssemblyLine フロー・チャート

AssemblyLine フロー・チャートを使用することで、特定のコンポーネントでの実行状態エレメント (フックなど) や項目オブジェクト (Conn、Work、Current、Error など) が使用可能な場所を表示することができます。

フロー・チャートは、PDF 文書 ftp://public.dhe.ibm.com/software/security/products/SDI/docs/7201/SDI_FlowDiagrams.pdf に記載されています。

付録 D. サーバー API

IBM Security Directory Integrator サーバー API に用意されている一連のプログラミング呼び出しを使用して、ソリューションを開発したり、ローカルまたはリモートで IBM Security Directory Integrator サーバーとやり取りしたりできます。

また、Java Management Extensions (JMX) インターフェースを介してサーバー API 呼び出しを公開する管理レイヤーも組み込まれています。

サーバー API の呼び出しを使用して、以下の操作を実行できます。

- IBM Security Directory Integrator サーバーに関する情報の取得
- サーバーにインストールされているコンポーネントに関する情報の取得
- サーバーによりロードされた構成の読み取り、変更、および書き込み
- サーバーでの新規構成の作成とロード
- AssemblyLine の開始、照会、および停止
- AssemblyLine の手動サイクル
- サーバー・イベント通知の登録と受信
- AssemblyLine のログ・メッセージの登録および受信

すべての呼び出しは、IBM Security Directory Integrator サーバー JVM からローカルに実行するか、または別の JVM (ローカルまたはリモート・ネットワーク・マシン) から RMI を介してリモートに実行することができます。

ローカル・アクセス

AssemblyLine フックへのスクリプトの組み込み、Java でインプリメントされてサーバーにデプロイされた新規コンポーネント (コネクター、関数コンポーネント) からの API の使用などがこのタイプのアクセスに該当します。

リモート・アクセス

このタイプのアクセスでは、IBM Security Directory Integrator にリモート接続し、IBM Security Directory Integrator 内でのプロセスを管理し、IBM Security Directory Integrator を基盤としたビジネス・ロジックを構築するソリューションをインプリメントできます。IBM Security Directory Integrator 専用アプリケーションまたは IBM Security Directory Integrator を使用して目標を達成するアプリケーションなどがあります。

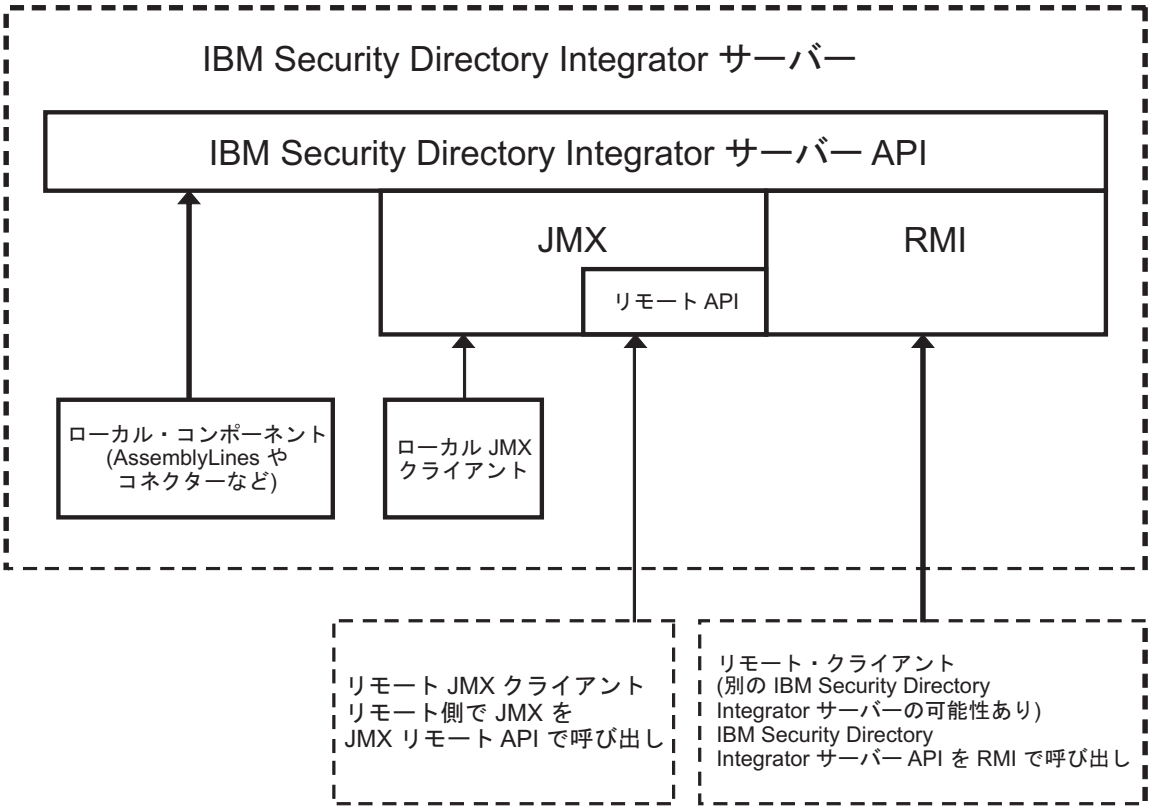
サーバー API の管理レイヤーは、JMX を介してサーバー API 呼び出しを公開します。これによりサーバー管理機能が実現し、JMX と対話する管理インフラストラクチャーに IBM Security Directory Integrator を接続することができます。JMX インターフェースには以下の方法でアクセスできます。

- ローカル (JMX 1.2 仕様の定義に基づく)
- リモート (JMX Remote API 1.0 仕様の定義に基づき RMI を使用)

サーバー API 内部エンジンから発行される通知は、JMX 通知としても使用可能です。

サーバー API (JMX Remote API を含む) へのリモート・アクセスを保護するため、クライアント認証とサーバー認証で SSL を使用します。

IBM Security Directory Integrator サーバー API への各種アクセス方法を以下の図に示します。



サンプル・ユース・ケース

このサンプル・シナリオでは、クライアント (スタンドアロン Java アプリケーションなど) が IBM Security Directory Integrator サーバーの AssemblyLine を開始する必要があります。クライアントはサーバー API を使用できます。この場合、サーバー API RMI クライアント・ライブラリーを使用し、RMI インターフェースを介してサーバー API にリモート・アクセスします。

700 ページの『セキュリティー』で説明するセキュリティー・モデルに基づき、最初にクライアントはクライアント自体の証明書またはカスタム認証を使用して、リモート IBM Security Directory Integrator サーバーへのセッションを作成します。クライアント証明書がサーバーのトラストストアに格納されている場合、またはカスタム認証が成功した場合には、サーバーでのクライアント認証が成功します。認証が成功すると、サーバー API メソッドを呼び出すためのエントリー・ポイントを表すオブジェクトがクライアントに渡されます。クライアントはこのオブジェクトを使用して、開始する必要がある AssemblyLine を指定するパラメーターを渡して AssemblyLine 開始のための呼び出しを実行します。

サーバー API はメソッドを実際に実行する前に、クライアントにこのメソッドの実行権限があるかどうかを調べます。つまり、SSL チャネルの確立に使用されたクラ

イアント証明書、またはカスタム認証用に提供されている信任状を使用して、クライアントの識別情報が判別されます。クライアントに対しこの AssemblyLine の開始が許可されている場合はメソッドが実行され、AssemblyLine が開始されます。それ以外の場合は、メソッドは実行されず、クライアントにこの操作を実行する権限がないことを示すエラー (例外) が戻されます。

ローカル・サーバーおよびリモート・サーバーの API インターフェース

ここに記載されている情報を使用することで、ローカル・サーバーおよびリモート・サーバーの API インターフェースを処理できます。

サーバー API には、ローカルで使用するインターフェース・セットとリモートで使用するインターフェース・セットの 2 つがあります。この 2 つのインターフェース・セットに組み込まれている呼び出しと機能は同一ですが、この 2 つのインターフェース・セットはそれぞれ異なる Java パッケージに含まれています。

パッケージ com.ibm.di.api.local にはローカル・アクセスのためのインターフェースが含まれており、com.ibm.di.api.remote には RMI を介したサーバーへのリモート・アクセスのためのインターフェースが含まれています。

ローカル・インターフェースおよびリモート・インターフェースとそのメソッドの詳細な仕様は、IBM Security Directory Integrator に付属の Javadoc 文書に記載されています (この文書は IBM Security Directory Integrator インストール先ルート・フォルダーの下の docs/api フォルダーにあります)。

リモート・パッケージのすべてのインターフェースは java.rmi.Remote を拡張し、すべてのメソッドは java.rmi.RemoteException をスローします。一方でローカル・アクセスのためのインターフェースは java.rmi.Remote を拡張せず、メソッドは java.rmi.RemoteException をスローしません。これはローカル・アクセスとリモート・アクセスのために個別のインターフェース・セットが必要である理由の 1 つであり、これによりコーディングが容易になります。

サーバー API 構造

ここに記載されている情報を使用することで、サーバー API 構造について理解することができます。

ローカル・インターフェースとリモート・インターフェースの構造は同一です。以降の文では、 Java インターフェース名のみを使用します。この内容は、ローカル (com.ibm.di.api.local) およびリモート (com.ibm.di.api.remote) サーバー API Java パッケージの両方のインターフェースに適用されます。

サーバー API のエントリー・ポイントは SessionFactory インターフェース (ローカルの場合は com.ibm.di.api.local.SessionFactory、リモートの場合は com.ibm.di.api.remote.SessionFactory) です。

SessionFactory インターフェースには、createSession() および createSession(Username, Password) という 2 つのメソッドがあります。これらのメソッドは、ユーザー/エンティティーのための API セッションを作成します。このユーザー/エンティティーは、この API セッションを呼び出し、タイプ Session

のオブジェクトを戻します。このセッション・オブジェクトが、サーバー API の呼び出しへの今後のアクセスを提供します。

このセッション・オブジェクトを使用して、サーバー情報の取得、サーバー自体の停止、既存の構成インスタンスの取得、新規構成インスタンスのロードまたは新規作成を実行できます。一部のセッション・オブジェクトの呼び出しでは、他のサーバー API オブジェクトが戻されます。例えば、`startConfigInstance(String aConfigUrl)` は `ConfigInstance` オブジェクトを戻します。`ConfigInstance` オブジェクトは、構成インスタンスで実行中の AssemblyLine に対して構成データ構造へのアクセスを提供し、新規 AssemblyLine を開始するための呼び出しを実行します。一部の呼び出しでは、サーバー API オブジェクトが戻されることもあります。例えば `startAssemblyLine(String aAssemblyLineName)` は、AssemblyLine で別の操作を照会および実行するときに使用できる `AssemblyLine` オブジェクトを戻します。

つまり、セッション・オブジェクトとは、サーバー API オブジェクト階層へのアクセスを提供するオブジェクトです。すべてのサーバー API 呼び出しは、 セッション・オブジェクトに対して直接実行するか、またはセッション・オブジェクトを介して直接的または間接的に取得されたオブジェクトに対して実行します。

セキュリティー

ここに記載されているメソッドとオプションを使用することで、サーバー API に対するセキュリティーを実行できます。

セッション・オブジェクトの取得中に認証が実行されます。取得されると、セッション・オブジェクト、またはこの セッション・オブジェクトを介して直接的または間接的に取得されたその他のサーバー API オブジェクトに対して呼び出されるすべてのメソッドは、セッション・オブジェクトを取得したユーザーの ID で実行されます。

メソッド呼び出しごとに許可が実行されます。サーバーは要求された呼び出しを実行する前に、現行セッションに関連付けられている ID に、この呼び出しの実行が許可されているかどうかを判別します。

以下の認証オプションが使用可能です。

SSL ベース認証 (V6.0 で使用可能なメカニズム)

このオプションは、`api.remote.ssl.client.auth.on=true` の場合にのみ有効です (また、`api.on=true`、`api.remote.on=true`、`api.remote.ssl.on=true` である必要があります)。

サーバー API ユーザー・レジストリー内の SSL 証明書ユーザー ID に割り当てられている権限に基づいて、ユーザーが許可されます。

注: SSL を使用しており、リモート・クライアント・アプリケーションがサーバー API リスナー・オブジェクトを使用する場合は、このクライアント・アプリケーションに、IBM Security Directory Integrator サーバーにより信頼されるアプリケーション自体の証明書が必要です (これは、SSL クライアント認証のセットアップに類似しています)。IBM Security Directory Integrator サーバーに信頼されるクライアント証明書がない場合、リスナ

ー・オブジェクトは機能せず、リモート・クライアント・アプリケーションは IBM Security Directory Integrator サーバーから通知を受信できません。

「ユーザー名/パスワード」ベースの認証

このオプションは、`api.custom.authentication` に JavaScript 認証ファイルが設定されている場合にのみ有効です。

この認証方式は、SSL が使用されているかどうか、および SSL クライアント認証が使用されているかどうかに関係なく動作します。サーバー API ユーザー・レジストリー内でユーザー名 (username) ユーザーに割り当てられている権限に基づいて、ユーザーが許可されます。

認証フックの Javascript の例を示す目的で、サンプル認証フック Javascript ファイルが提供されています。このサンプル Javascript は、実際の IBM Security Directory Integrator 認証フックを作成する場合の基礎として使用できます。

認証フックで LDAP サーバー (IBM Security Directory Server や Active Directory など) を使用してクライアント要求を認証する方法を示すサンプルの JavaScript は、IBM Security Directory Integrator サーバー・フォルダー examples/auth_ldap に格納されています。このサンプル・ファイルの名前は ldap_auth.js です。

LDAP 認証

IBM Security Directory Integrator サーバー API は LDAP 認証をサポートします。これにより、利用者は既にユーザー ID とパスワードが保管されている既存の LDAP インフラストラクチャーを利用できます。

LDAP 認証を使用するには、global.properties/solution.properties で該当するプロパティーを構成する必要があります。これらのプロパティーについては、「管理者ガイド」で説明します。

ホスト・ベース認証

このオプションは、`api.remote.ssl.on=false` の場合にのみ有効です。この場合、`api.remote.nonssl.hosts` プロパティーに指定されているすべてのホストから提供されるユーザー名/パスワードを使用せずにサーバー API セッションをオープンする操作が正常に認証され、管理権限が付与されます。`api.remote.nonssl.hosts` プロパティーは global.properties/solution.properties ファイルに指定できます。

注: この認証は、独立したトラステッド環境において、迅速なプロトタイピングとデモ目的でのみ使用することを強くお勧めします。

JAAS 認証

サーバー API では、JAAS 認証がサポートされています。既に JAAS 認証モジュールがある場合は、それを IBM Security Directory Integrator で使用することができます。

JAAS 認証を使用するには、`global.properties` または `solution.properties` で適切なプロパティーを構成し、JAAS ログオンをインストールする必要があります。

注: IBM Security Directory Integrator は、JAAS 認証モジュールの構成を行いません。認証モジュールがインプリメントされており、正しく構成されているという理解に基づいて動作します。IBM Security Directory Integrator は単にモジュールを使用するだけです。

サーバー API の構成

ここにリストされているファイルに、関連するシステム・プロパティーを指定して、サーバー API を構成できます。

ファイルは、`global.properties` (または `solution.properties`) およびユーザー・レジストリー・ファイルです。

サーバー API プロパティーの構成

ここに記載されている情報とリンクを使用することで、サーバー API プロパティーを構成することができます。

サーバー API エンジンを構成するには、`global.properties` ファイル (ソリューション・フォルダーを使用する場合は `solution.properties` ファイル) で一連のプロパティーを設定します。サーバー API の構成方法については、「*インストールと管理*」内のセキュリティーおよび IBM Security Directory Integrator に関するセクションである、「サーバー API のアクセス・セキュリティー」セクションを参照してください。

ユーザー・レジストリーのセットアップ

ここに記載されている情報とリンクを使用することで、ユーザー・レジストリーをセットアップできるようになります。

ユーザー・レジストリーのセットアップ、ユーザー役割の割り当て、ユーザー・レジストリー・ファイルの暗号化/暗号化解除の説明と例については、「*インストールと管理*」の『セキュリティーと IBM Security Directory Integrator』のセクションを参照してください。

リモート・クライアント構成

ここに記載されている前提条件とステップを使用することで、リモート・クライアントを構成できます。

このセクションでは、リモート・サーバー API を使用するリモート・クライアントの要件について説明します。

前提条件:

クライアント・サイドに Java 7.0.4 以降が必要です。

クライアントの構成:

1. リモート・サイドの CLASSPATH に以下の jar ファイルが組み込まれている必要があります。
   - jars/common/diserverapi.jar
   - jars/common/diserverapirmi.jar
   - jars/3rdparty/others/log4j-1.2.15.jar

- jars/common/miconfig.jar
- jars/common/miserver.jar
- jars/common/mmconfig.jar
- jars/common/tdiresource.jar
- jars/3rdparty/IBM/icu4j_4_2.jar
- jars/3rdparty/IBM/ITLMToolkit.jar
- jars/3rdparty/IBM/jlog.jar

これらの jar ファイルは IBM Security Directory Integrator インストール済み環境からコピーできます。

2. サーバー API によってインプリメントされるソリューションで IBM Security Directory Integrator 以外のカスタム・オブジェクトを使用する場合 (項目の属性値が接続を介して転送される場合など)、クライアント・サイドでも対応する Java クラスが使用可能になっている必要があります。これらのクラスはシリアライズ可能であり、クライアント JVM の CLASSPATH に組み込まれている必要があります。

リモート・クライアントの SSL 構成

リモート・クライアントで SSL を構成するには、次の 2 つの方法があります。

サーバー API 固有の SSL プロパティーの使用

Java システム・プロパティー `api.client.ssl.custom.properties.on` が `true` に設定されている場合は、以下の IBM Security Directory Integrator API 固有 Java システム・プロパティーを使用して SSL を構成します。

- **api.client.keystore** – クライアント証明書が含まれている鍵ストア・ファイルを指定します。
- **api.client.keystore.pass** – *api.client.keystore* に指定する鍵ストア・ファイルのパスワードを指定します。
- **api.client.keystore.type** – *api.client.keystore* オプション・プロパティーで指定されている鍵ストア・ファイルのタイプを指定します。指定しないと、JVM のデフォルトの鍵ストア形式が使用されます。
- **api.client.key.pass** – *api.client.keystore* に指定する鍵ストア・ファイルに格納されている秘密鍵のパスワードです。このプロパティーが指定されていないと、*api.client.keystore.pass* に指定されるパスワードが代わりに使用されます。
- **api.client.truststore** – IBM Security Directory Integrator サーバーの公開証明書を含む鍵ストア・ファイルを指定します。
- **api.client.truststore.pass** – *api.truststore* で指定される鍵ストア・ファイルのパスワードを指定します。
- **api.client.truststore.type** – *api.client.truststore* オプション・プロパティーで指定されている鍵ストア・ファイルのタイプを指定します。指定しないと、JVM のデフォルトの鍵ストア形式が使用されます。

クライアント・アプリケーションで、このアプリケーションが使用する別の SSL チャネルの構成に標準 Java SSL プロパティーを使用している場合は、サーバー API 固有の SSL プロパティーを使用すると便利です。

これらのプロパティーをコマンド行で JVM 引数として指定できます。以下に例を示します。

```
java MyTDIServerAPIClientApp
    -Dapi.client.ssl.custom.properties.on=true
    -Dapi.client.truststore=C:¥TDI¥serverapi¥testadmin.jks
    -Dapi.client.truststore.pass=administrator
    -Dapi.client.keystore=C:¥TDI¥serverapi¥testadmin.jks
    -Dapi.client.keystore.pass=administrator
```

この例では、IBM Security Directory Integrator に同梱の`testadmin.jks` 鍵ストア・ファイルを参照しています。このサンプル・ファイルには、クライアントの秘密鍵と IBM Security Directory Integrator サーバーの公開鍵の両方が含まれているため、鍵ストアおよびトラストストアの両方として使用される点に注意してください。

標準 SSL Java システム・プロパティーの使用

Java システム・プロパティー `api.client.ssl.custom.properties.on` が指定されていないか、または `false` に設定されている場合は、SSL チャネルの構成に標準 JSSE システム・プロパティーが使用されます。標準 JSSE 手順に従って、クライアント・アプリケーションにより使用される鍵ストアとトラストストアを構成します。

これらのプロパティーをコマンド行で JVM 引数として指定できます。以下に例を示します。

```
java MyTDIServerAPIClientApp
-Djavax.net.ssl.keyStore=C:¥TDI¥serverapi¥testadmin.jks
-Djavax.net.ssl.keyStorePassword=administrator
-Djavax.net.ssl.trustStore=C:¥TDI¥serverapi¥testadmin.jks
-Djavax.net.ssl.trustStorePassword=administrator
```

サーバー API の使用

サーバー API を使用して、ローカル・セッションおよびリモート・セッションを作成できます。

ローカル・セッションの作成

ここに記載されているサンプル・コードを使用することで、ローカル・セッションを作成できます。

IBM Security Directory Integrator サーバー JVM で実行される Java コード (新規コネクター、またはスクリプトを使用してアクセスする Java クラスなど) を作成し、サーバー API 呼び出しを実行する場合は、ローカル・サーバー API セッションが必要です。

ローカル・サーバー API セッションを取得するには、以下を呼び出します。

```
import com.ibm.di.api.APIEngine;
import com.ibm.di.api.local.Session;

...

Session session = APIEngine.getLocalSession();
```

`getLocalSession()` は `com.ibm.di.api.APIEngine` クラスの静的メソッドです。このメソッドは `com.ibm.di.api.local.Session` オブジェクトを新規に作成して戻します。戻されたこのセッションには管理権限があるため、すべてのサーバー API 呼び出しを実行できます。

スクリプト・コンテキストでのサーバー API へのアクセス

スクリプト・コンテキスト (AssemblyLine フックなど) からサーバー API へアクセスするには、session スクリプト・オブジェクトを呼び出します。 IBM Security Directory Integrator サーバーは com.ibm.di.api.APIEngine.getLocalSession() メソッドを呼び出してセッション・オブジェクトを登録します。

リモート・セッションの作成

ここに記載されているサンプル・コードを使用して、リモート・セッションを作成できます。

リモート・サーバー API を使用するクライアント・アプリケーションは、最初に IBM Security Directory Integrator サーバーに接続してサーバー API セッションを取得する必要があります。

RMI SessionFactory オブジェクトをルックアップし、サーバー API セッションを取得するには、以下の Java コードを使用します。

```
import com.ibm.di.api.remote.Session;
import com.ibm.di.api.remote.SessionFactory;

...

SessionFactory sessionFactory = (SessionFactory) Naming.lookup("rmi://<TDI_Server_host>:
 <TDI_Server_RMI_port>/SessionFactory");

Session session = sessionFactory.createSession();
```

TDI_Server_host と *TDI_Server_RMI_port* を、IBM Security Directory Integrator サーバーのホストと RMI ポートにそれぞれ置き換える必要があります。以下に例を示します。

```
Naming.lookup("rmi://127.0.0.1:1099/SessionFactory")
```

ローカル・セッション・オブジェクトとリモート・セッション・オブジェクトから提供される呼び出しは同一です。以下に示すすべての例では、セッションを既に取得しており、リモート・コンテキストで操作することを前提としています。つまり、サーバー API インターフェースのリモート・バージョンが使用されます。

構成インスタンスの処理

構成インスタンスを使用すると、AssemblyLine、コネクター、パーサー、関数コンポーネントの構成の照会、AssemblyLine の開始、実行中の AssemblyLine へのアクセス、ログ・ファイルの照会などの各種タスクを実行できます。

構成インスタンスは、IBM Security Directory Integrator サーバーにロードされた構成と関連するサーバー・オブジェクトを表します。各 AssemblyLine は構成インスタンスのコンテキストで実行されます。

実行中の構成インスタンスへのアクセス

IBM Security Directory Integrator サーバーで実行中のすべての構成インスタンスにアクセスするには、以下のコードを実行します。

```
ConfigInstance[] configInstances = session.getConfigInstances();
for (int i=0; i<configInstances.length; i++) {
// do something with configInstances[i]
}
```

getConfigInstances() メソッドから、サーバーで実行中のすべての構成インスタンスを表す構成インスタンス・サーバー API オブジェクトの配列が戻されます。

構成インスタンスの開始

ここに記載されている情報とサンプル・コードを使用することで、構成インスタンスを開始できます。

IBM Security Directory Integrator サーバーに新規構成をロードするには、新規構成インスタンスを開始し、このインスタンスが XML 構成ファイルを指し示すように設定する必要があります。

```
ConfigInstance configInstance = session.startConfigInstance("testconfig.xml");
```

これにより testconfig.xml 構成ファイルがロードされ、構成に関連する新規構成インスタンス・オブジェクトが開始されます。構成インスタンス・オブジェクトを取得したら、このオブジェクトを使用して、構成自体の変更、AssemblyLine の開始、または構成インスタンスが不要になった時点でのサーバーでの構成インスタンスの停止を実行できます。

単一の構成ファイルから複数の構成インスタンスを開始する必要がある場合 (インスタンスごとに異なるプロパティー・セットを使用する場合など) は、各インスタンスに固有の実行名を付与する必要があります。

```
ConfigInstance configInstance = session.startConfigInstance("testconfig.xml", true, null,
 "myrunname", "mystore=mynewstore.properties");
```

上記の呼び出しを行うと、「testconfig.xml」ファイルから「myrunname」という実行名の新規構成インスタンスが開始します。この実行名は、この構成インスタンスの ID として使用されます。さらに、インスタンスのプロパティー・ストア「mystore」がリダイレクトされて、「mynewstore.properties」ファイルから内容がロードされます。

構成インスタンスの停止

ここに記載されている情報とサンプル・コードを使用することで、構成インスタンスを停止できます。

構成インスタンス・サーバー API オブジェクトへの参照がある場合は、以下のように呼び出すことで構成インスタンスを停止できます。

```
configInstance.stop();
```

構成インスタンス・オブジェクトの参照については、以下のオプションがあります。

- 構成インスタンスを開始した位置からの参照を維持する (例: `configInstance = session.startConfigInstance("testconfig.xml");`)
- 構成 ID を使用して `session.getConfigInstance (String aConfigId)` を呼び出し、構成インスタンス・オブジェクトを取得する。構成 ID は、サーバーで実行されている各構成インスタンスの固有 ID です。対応するサーバー API 構成インスタンス・オブジェクトが作成されると、この ID がサーバー API により作成されます。`configInstance.getConfigId()` を呼び出し、構成インスタンス・オブジェクトを介して構成 ID を取得できます。

- 実行中のすべての構成インスタンスを反復し、必要なインスタンスを見つけます。`session.getConfigInstances()` は実行中のすべての構成インスタンスの配列を戻します。

サーバー API と構成の初期設定の同期化

ここに記載されている情報とサンプル・コードを使用することで、サーバー API と構成の初期設定を同期化できます。

サーバー API 呼び出しが作成されたとき、構成インスタンスによって完全に構成ファイルがロードされていない場合は、NULL オブジェクトが返されます。IBM Security Directory Integrator では、タイムアウト間隔は **api.config.timeout** プロパティーを使用して構成できます。デフォルトでは、このプロパティーは 2 分です。この時間間隔内に構成がロードされない場合、例外がスローされます。

構成ファイル内の任意指定の構成インスタンス ID

ここに記載されている情報を使用することで、構成ファイル内の任意指定の構成インスタンス ID を処理できます。

構成インスタンスは、IBM Security Directory Integrator サーバーにロードされた構成および関連するサーバー・オブジェクトを表しています。各 AssemblyLine は構成インスタンスのコンテキストで実行されます。構成インスタンスを使用すると、AssemblyLine、コネクター、パーサー、関数コンポーネントの構成の照会、AssemblyLine の開始、実行中の AssemblyLine へのアクセス、およびログ・ファイルの照会を実行できます。

ソリューション名と実行名 - 構成インスタンス ID:

IBM Security Directory Integrator サーバーによってロードされると、構成ファイルは実行中の**構成インスタンス**になります。各構成インスタンスには、固有の**構成 ID** があります。同時に実行中の 2 つの構成インスタンスが同じ構成 ID を持つことはできません (構成 ID は、IBM Security Directory Integrator 内で実行中の構成インスタンスを一意的に識別します)。

構成インスタンスによって構成ファイルが開始すると、IBM Security Directory Integrator サーバーはまず構成ファイルに**ソリューション名** (ソリューション・インターフェースの構成フィールドにある構成フィールド) が定義されているかどうかを検査します。ソリューション名が存在し、空でない場合、サーバーはこの名前を構成インスタンス ID として使用します。ソリューション名が存在しないか空の場合、IBM Security Directory Integrator サーバーは自動的に構成 ID を生成します。

例えば、絶対ファイル名を持つ構成ファイル「C:/IBM/TDI/configs/rs.xml」が IBM Security Directory Integrator サーバーにロードされており、このファイルのソリューション名が「mysoluname」に設定されている場合、生成される構成インスタンスの ID は「mysoluname」になります。同じ構成でソリューション名が定義されていない場合、構成インスタンス ID は「C__IBM_TDI_configs_rs.xml」などのようになります。

注: IBM Security Directory Integrator サーバー API を使用するクライアントは、自動的に生成された構成インスタンス ID を透過的なエンティティーとして認識する必要があります。これらの ID を生成するアルゴリズムは将来変更されるため、ID

の生成方法を推測しようとすることはできません。唯一保証されているのは、自動生成された構成 ID を持つ構成インスタンスが過去に存在した場合、後で同じ構成 ID を使用してトゥームストーンやシステム・ログなどの特定の成果物にアクセスすることができます。ただし、同じ構成ファイルが再度実行された場合、新しく生成された構成インスタンスに前と同じ自動生成された ID が付けられる保証はありません。
一般に、構成インスタンスの開始時に、クライアントが構成ファイルへのパスだけを指定している場合、構成 ID は構成ファイルの属性 (存在する場合はソリューション名、または絶対ファイル名) のみに基づいて生成されます。結果として、追加の情報が指定されない場合、構成ファイルを構成インスタンスとしてロードできるのは 1 回だけになります (それ以上ロードすると構成インスタンス ID の競合が発生します)。

同じ構成ファイルから複数の構成インスタンスをロードする必要がある場合、クライアントは各インスタンスに対して固有の**実行名**を指定する必要があります。構成インスタンスの開始時に実行名が指定された場合、実行名はこのインスタンスの構成インスタンス ID として使用されます。したがって、実行名は既に実行中のどの構成インスタンスの ID とも一致しない必要があります。

各ソリューション名および各実行名は、IBM Security Directory Integrator サーバーが現在実行中のプラットフォーム上で有効なファイル名である必要があります。このような制限がある理由は、システム・ログなどの構成インスタンス固有の情報を保存するときに構成インスタンス ID (ソリューション名または実行名から派生) が使用されるからです。ファイル・システムの問題を防ぐために、IBM Security Directory Integrator は実行名やソリューション名に以下の記号を使用することが禁止されています: ¥ / : * " < > | ?

注:

1. 上記のいずれかの記号が含まれるソリューション名で構成インスタンスが開始されると、IBM Security Directory Integrator サーバーは問題のある記号を自動的に下線に置き換え、警告をログに記録します。クライアントが、上記のいずれかの記号が含まれる実行名で構成インスタンスを開始しようとすると、API の呼び出しは失敗し、例外がスローされます。
2. 有効なファイル名の定義はファイル・システムによって異なるため、上記の記号を使用しないようにするだけでは、実行名 (ソリューション名) が有効なファイル名であることを保証できません。サーバー API がこのような記号を禁止するポリシーは、絶対的な保護ではなく、ベストエフォートとしての検査と考える必要があります。結果的に、実行名 (またはソリューション名) が有効なファイル名ではない構成インスタンスを開始することは可能です。このようなインスタンスは、システム・ログなどの機能に依存する場合、ファイル・システム関連の問題に至ります。

他には、このような名前を使用する構成インスタンスについては、ソリューション名および実行名はファイル・システムの絶対パスではなくユーザー・レジストリーにある必要があります。

絶対ファイル名「C:/IBM/TDI/configs/rs.xml」を持つ構成ファイルがあるとします。下の表に、ユーザー・レジストリー内のこのファイルから起動された構成インスタ

ンスを参照する方法を説明しています。この表では、構成ファイルにソリューション名があるかどうか、および構成インスタンスが実行名を使用して開始されるかどうかが考慮されています。

ソリューション名	実行名	ユーザー・レジストリー内のセクション
-	-	[CONFIG]:C:/IBM/TDI/configs/rs.xml
-	myrunname	[CONFIG]:myrunname
mysoluname	-	[CONFIG]:mysoluname
mysoluname	myrunname	[CONFIG]:myrunname

ユーザー・レジストリー内の許可は、構成ファイルごとではなく、構成インスタンスごとに割り当てられている点に注意することが重要です。

構成ファイル・パスの代わりにソリューション名を使用する

注: サーバー API を使用して編集できる構成ファイルは、このフォルダーにあるものだけです。

構成インスタンスおよびサーバー API のチェックイン/チェックアウト機能を開始する IBM Security Directory Integrator 6.1 および以前のリリースでは、構成ファイルの URL (ファイル・パス) を指定する必要がありました。現在のバージョンの IBM Security Directory Integrator では、対応するソリューション名で同じサーバー API インターフェースのメソッドを渡すことができるため、これは不要になりました。AMC や CLI などのサーバー API クライアントでは、分かりにくい構成ファイルパスの代わりに、分かりやすいソリューション名を受け入れるようになったため、ユーザーにとっては便利です。

構成ファイル・パスは、ソリューション名よりも優先度が高くなっています。つまり、例えば構成インスタンスを開始するメソッドにストリング (構成ファイル・パスまたは対応するソリューション名のいずれか) が渡され、構成ファイル・パスが有効な場合、メソッドはこの構成ファイルへの参照としてこの値を扱います。ストリングとして同一の構成ファイルとソリューション名がある場合、構成ファイル・パスが優先されます。この動作によって、ソリューション名がない以前のバージョンの IBM Security Directory Integrator との互換性が確保されます。

IBM Security Directory Integrator サーバーの起動時に、ソリューション名を使用して参照できるのは、IBM Security Directory Integrator の configs フォルダー (`global.properties` または `solution.properties` ファイルのパラメーター **api.config.folder** で指定) にある構成と、ソリューション・ディレクトリーにある構成のみです。

configs フォルダーでソリューション名をスキャンする

IBM Security Directory Integrator サーバーは、始動時に configs フォルダー (`global.properties` または `solution.properties` の **api.config.folder** プロパティーで指定) で、configs フォルダーにある構成ファイルのソリューション名をスキャンします。次に、サーバーは、構成ファイル・パスの代わりにソリューション名を使用できるように、ソリューション名を構成ファイル・パスにマッピングする内部マップを作成します。

configs フォルダーをスキャンする際に使用するルールを以下に示します。

1. ファイル名の拡張子が「.cfg」の場合 – ファイル名を返す (非常に古いスタイルの構成)
2. 構成ドライバーによって構成を正常にロードできる場合は、ソリューション名を調べる
3. 構成ドライバーによって構成をロード**できない** 場合、
   a. ファイル名の拡張子が「.xml」の場合 – ファイル名を返す
   b. 別の拡張子の場合 – ファイルを無視し、何も返さない

この結果、IBM Security Directory Integrator サーバーの始動方法によって以下の状態になります。

セキュア・モードの IBM Security Directory Integrator サーバー	**通常モードの IBM Security Directory Integrator サーバー**
PKI で暗号化された構成 – ソリューション名が表示される (存在する場合)	PKI で暗号化された構成 – ファイル名が表示される (拡張子が .cfg または .xml の場合)
暗号化されていない構成 – ファイル名が表示される (拡張子が .cfg または .xml の場合)	暗号化されていない構成 –IBM Security Directory Integrator ソリューション名が表示される (存在する場合)
パスワードで暗号化された構成 – ファイル名が表示される (拡張子が .cfg または .xml の場合)	パスワードで暗号化された構成 – ファイル名が表示される (拡張子が .cfg または .xml の場合)
SDI 以外の構成ファイル (その他、テキストまたはバイナリー) – ファイル名が表示される (拡張子が .cfg または .xml の場合)	SDI 以外の構成ファイル (その他、テキストまたはバイナリー) – ファイル名が表示される (拡張子が .cfg または .xml の場合)

有効な構成ファイル (XML フォーマットまたは cfg フォーマット) 以外のファイルを configs フォルダーに保管しないことをお勧めします。構成ファイル以外のファイルを解析しようとすると、エラーが報告される場合があり、ファイルは無視されます。これは、サーバーの正常な運用には影響しません。

AssemblyLine の処理

ここに記載されている詳細を使用することで、AssemblyLine を処理できます。

構成内で使用可能な AssemblyLine へのアクセス

既に構成インスタンス・オブジェクトへの参照がある場合は、構成インスタンス全体の構成データ・ストラクチャーを表す MetamergeConfig オブジェクトを取得してから、使用可能な AssemblyLine を取得する必要があります。

```
import com.ibm.di.config.interfaces.MetamergeConfig;
import com.ibm.di.config.interfaces.MetamergeFolder;
import com.ibm.di.config.interfaces.AssemblyLineConfig;

...

MetamergeConfig configuration = configInstance.getConfiguration();
MetamergeFolder configFolder =
    configuration.getDefaultFolder(MetamergeConfig.ASSEMBLYLINE_FOLDER);
String[] assemblyLineNames = configFolder.getNames();
if (assemblyLineNames != null) {
    for (int i=0; i<assemblyLineNames.length; i++) {
        System.out.println(assemblyLineNames[i]);
```

```
// get the AssemblyLine configuration object
AssemblyLineConfig alConfig =
    configuration.getAssemblyLine(assemblyLineNames[i]);
// do something with alConfig ...
```

上記のコード・ブロックは、構成内のすべての AssemblyLine 名を標準出力に出力するものであり、AssemblyLine 構成オブジェクトの取得方法を示します。AssemblyLine 構成オブジェクトを使用して、AssemblyLine で構成されているコネクターとそのパラメーターなどの詳細情報を取得できます。

MetamergeConfig、MetamergeFolder、および AssemblyLineConfig インターフェースは、サーバー API インターフェースではありません。これらのインターフェースは、IBM Security Directory Integrator 構成ドライバーに属しています (例の import 文節を参照)。また、これらはリモート・オブジェクトではありません。configInstance.getConfiguration() が実行されると、MetamergeConfig オブジェクトがシリアライズされ、接続を介して転送されます。次にそのオブジェクトのローカル・コピーがコードによって処理されます。

実行中の AssemblyLine へのアクセス

特定の構成インスタンスのアクティブ AssemblyLine を取得するか、または実行中のすべての構成インスタンスの IBM Security Directory Integrator サーバー上でのアクティブ AssemblyLine をすべて取得できます。

特定の構成インスタンスのアクティブ AssemblyLine の取得:

構成インスタンス・オブジェクトへの参照が必要です。現在構成インスタンスで実行中のすべての AssemblyLine を戻すコードを以下に示します。

```
AssemblyLine[] assemblyLines = configInstance.getAssemblyLines();
for (int i=0; i
for (int i=0; i<assemblyLines.length; i++) {
    System.out.println(assemblyLines[i].getName());

    // do someting with assemblyLines[i]
}
```

IBM Security Directory Integrator サーバー全体でのアクティブ AssemblyLine の取得: サーバー上で実行中のすべての AssemblyLine を取得するには、以下のコードを実行します。

```
AssemblyLine[] assemblyLines = session.getAssemblyLines();
for (int i=0; i<assemblyLines.length; i++) {
    System.out.println(assemblyLines[i].getName());

    // do someting with assemblyLines[i]

    // which Config Instance this AssemblyLine belongs to?
    ConfigInstance alConfigInstance = assemblyLines[i].getConfigInstance();
}
```

これは特定の構成インスタンスに対して実行されるのではなく、セッション・レベルで実行される点に注意してください。実行中の AssemblyLine が属する構成インスタンスを確認する必要がある場合は、AssemblyLine オブジェクトを介して親構成インスタンス・オブジェクトへの参照を取得できます。

各種 AssemblyLine プロパティー、AssemblyLine 構成オブジェクト、AssemblyLine ログ、AssemblyLine 結果項目を取得する場合、または AssemblyLine を停止する場合は、AssemblyLine サーバー API オブジェクトを使用します。

AssemblyLine の開始

AssemblyLine を開始するには、この AssemblyLine が属する構成インスタンス・オブジェクトを使用します。

開始する AssemblyLine の名前を把握しておく必要があります。

```
AssemblyLine assemblyLine = configInstance.startAssemblyLine("MyAssemblyLine");
```

また、新規に開始された AssemblyLine インスタンスへの参照も受け取ることができます。

手動モードでの AssemblyLine の開始

AssemblyLine を手動で実行するためにサーバー API を使用できます。

手動モードでは、AssemblyLine は固有のスレッドで実行されません。AssemblyLine の開始時に AssemblyLine が初期化されるだけです。AssemblyLine オブジェクトの executeCycle() メソッドが呼び出されると、AssemblyLine の反復処理が同期的に実行されます。この呼び出しにより現行スレッドがブロックされ、AssemblyLine 反復が完了すると結果項目オブジェクトが戻されます。

TestAL AssemblyLine を手動モードで開始し、この AssemblyLine の反復処理を 3 回実行するコードを以下に示します。各反復処理の結果項目は標準出力に出力されます。

```
AssemblyLineHandler alHandler = configInstance.startAssemblyLineManual("TestAL", null);
Entry entry = null;
for (int i=0; i<3; i++) {
  entry = alh.executeCycle();
  System.out.println("TestAL entry: " + entry);
}
alHandler.close();
```

構成インスタンス・オブジェクトの `startAssemblyLineManual(String aAssemblyLineName, Entry aInputData)` メソッドは AssemblyLine を手動モードで開始し、タイプ `com.ibm.di.api.remote.AssemblyLineHandler` のオブジェクトを戻します。このオブジェクトを使用して、AssemblyLine の反復処理を手動で実行し、初期作業項目と各種タスク呼び出しブロック・パラメーターを渡し、AssemblyLine サーバー API オブジェクトへの参照を取得し、処理が完了した時点で AssemblyLine を終了することができます。

AssemblyLine の実行時の動作を模倣するには、`NULL` が戻されるまで `executeCycle()` を呼び出します。

リスナーを使用した AssemblyLine の開始

ここに記載されている情報を使用することで、リスナーを使用して AssemblyLine を開始できます。

サーバー API で AssemblyLine を開始すると、それぞれの AssemblyLine の反復で通知を受信し、結果項目を配信する特定の AssemblyLine リスナーを登録できます。また、これは AssemblyLine 終了時の場合も同様です。このメカニズムにより、AssemblyLine をリモート・アプリケーションから開始でき、AssemblyLine により作成されるすべての項目を容易に受信できます。また、AssemblyLine リスナーは、AssemblyLine の実行中にログに記録されたすべてのメッセージを配信します。

使用するリスナー・クラスは、com.ibm.di.api.remote.AssemblyLineListener インターフェース (ローカル・アクセスの場合は com.ibm.di.api.local.AssemblyLineListener) をインプリメントしている必要があります。

以下のメソッドを指定する必要があります。

- assemblyLineCycleDone(Entry aEntry) – このメソッドは、AssemblyLine の各反復の終わりに呼び出されます。aEntry パラメーターは、AssemblyLine 反復の結果項目を表します。
- assemblyLineFinished() – このメソッドは、AssemblyLine 終了時にサーバー API により呼び出されます。
- messageLogged(String aMessage) – このメソッドは、AssemblyLine ロガーによりメッセージがログに記録されるたびに、サーバー API により呼び出されます。したがって、AssemblyLine により生成されるログ・メッセージにリアルタイムでリモート・アクセスできます。

受信したすべての項目とすべての AssemblyLine ログ・メッセージを標準出力にのみ出力するサンプル AssemblyLine リスナー・クラスを以下に示します。

```
import com.ibm.di.api.DIException;
import com.ibm.di.api.remote.AssemblyLineListener;
import com.ibm.di.entry.Entry;
import java.rmi.RemoteException;

public class MyRemoteALListener implements AssemblyLineListener {

  public void assemblyLineCycleDone(Entry aEntry)
    throws DIException, RemoteException
  {
    System.out.println("AssemblyLine iteration: " + aEntry.toString());
    System.out.println();
  }

  public void assemblyLineFinished()
    throws DIException, RemoteException
  {
    System.out.println("AssemblyLine terminated.");
    System.out.println();
  }

  public void messageLogged(String aMessage)
    throws DIException, RemoteException
  {
    System.out.println("AssemblyLine log message: " + aMessage);
    System.out.println();
  }
}
```

AssemblyLine リスナー・クラスのインプリメント後に、リスナー・オブジェクトをインスタンス化し、AssemblyLine 開始時にこのインスタンスを渡す必要があります。

```
MyRemoteALListener alListener = new MyRemoteALListener();
configInstance.startAssemblyLine("TestAL", null,
 AssemblyLineListenerBase.createInstance(alListener,true), true);
```

startAssemblyLine(String aAssemblyLineName, Entry aInputData, AssemblyLineListener aListener, boolean aGetLogs) メソッドは、AssemblyLine 名、初期作業項目、リスナー・オブジェクトを指定し、ログ・メッセージを受信するかどうかを指定します。aGetLogs が false の場合は、messageLogged(String aMessage) リスナー・メソッドはサーバー API により呼び出されません。

リモート・コンテキストでリスナーを登録する場合は、特定のリスナーを AssemblyLine ベース・リスナー・クラスでラップする必要があります。これは、サーバー・サイドにないカスタム・リスナー Java クラスと、サーバー API 通知メカニズムを連結するために必要です。
com.ibm.di.api.remote.impl.AssemblyLineListenerBase クラスの静的メソッド createInstance(AssemblyLineListener aListener, boolean aSSLon) を呼び出して、ベース・リスナー・クラスを作成します。使用するリスナー・クラスを表すオブジェクトを提供し、サーバーとの通信に SSL を使用するかどうかを指定する必要があります (サーバー・サイドでのサーバー API の構成方法を指定する必要があります。指定しないと、リスナーとの通信が失敗します)。

コンポーネント・シミュレーションによる AssemblyLine の開始

ここに記載されている情報を使用することで、コンポーネント・シミュレーションを使用して AssemblyLine を開始できます。

AssemblyLine のシミュレーション・フラグを true に設定し、AssemblyLine におけるコンポーネントの動作をシミュレートすることを指定します。シミュレーション機能の詳細は、「IBM Security Directory Integrator」で説明されています。ここでは、シミュレーション・フラグの設定方法のみ説明します。

```
usertcb.setProperty(com.ibm.di.server.AssemblyLine.TCB_SIMULATE_MODE, Boolean.TRUE);
```

「usertcb」は TaskCallBlock オブジェクトです。このオブジェクトを使用して AL を開始します。

AssemblyLine の停止

ここに記載されているコード行を実行することで、AssemblyLine を停止できます。

AssemblyLine を停止するには、AssemblyLine オブジェクトへの参照が必要です。AssemblyLine 開始時から AssemblyLine オブジェクトへの参照を維持するか、またはすべての実行中 AssemblyLine を反復し、必要な AssemblyLine を見つけます。コード例:

```
assemblyLine.stop();
```

構成の編集

ここに記載されている情報を使用することで、各種の構成設定を編集できます。

IBM Security Directory Integrator 構成フォルダー

ここに記載されている情報を使用することで、参照およびロードにどのファイルが使用可能かについて知ることができます。

IBM Security Directory Integrator サーバー・プロパティー `api.config.folder` が IBM Security Directory Integrator サーバー構成ファイル `global.properties` で使用可能です。このプロパティーは、ローカル・ディスク上のフォルダーを指定します。サーバー API には、このフォルダーまたはサブフォルダーにある構成を参照およびロードするための呼び出しがあります。例を示します。

```
api.config.folder=configs
```

つまり、「*<TDI_root>*/configs」およびそのサブフォルダーに保管されているすべての構成ファイルを、サーバー API から (ローカルまたはリモートで) 参照およびロードできます。

サーバー API には、`api.config.folder` プロパティーで指定されたフォルダー内のファイルとフォルダーを参照するための新しい呼び出しがあります。

編集のためのロード

ここに記載されている情報を使用することで、編集のためのファイルのロード・プロセスについて知ることができます。

IBM Security Directory Integrator 6.0 では、構成を編集する前に、対応する構成インスタンスが IBM Security Directory Integrator サーバーで既に開始されている必要がありました。その後、構成オブジェクトを取得し、(変更されている可能性のある) 構成オブジェクトの設定を元に戻し、ディスクに構成を保管する API 呼び出しが実行されました。

IBM Security Directory Integrator の現行バージョンでは、アクティブな構成インスタンスの構成オブジェクトは変更できません。サーバー API ユーザーは、引き続きアクティブな構成インスタンスの構成オブジェクトを取得できますが、構成オブジェクトを設定してディスクに保管する以下の呼び出しを、通常の実行中に構成オブジェクトに対して実行すると、例外がスローされます。

- `ConfigInstance.setConfiguration(MetamergeConfig configuration)`
- `ConfigInstance.saveConfiguration()`
- `ConfigInstance.saveConfiguration(boolean aEncrypt)`

一時構成インスタンスを使用して編集用に構成をロードすると、構成に適用される変更をテストする目的で `setConfiguration(...)` メソッドを実行できます。ただし `saveConfiguration(...)` メソッドを実行すると例外がスローされます。IBM Security Directory Integrator は、編集用に構成をロードするための新規サーバー API 呼び出しと、編集した構成をディスクに保管するための新規サーバー API 呼び出しを提供します。

構成のロック

ここに記載されている情報を使用することで、構成のロックのプロセスを理解できるようになります。

サーバー API は、編集用にロードされたすべての構成を内部追跡します。別のサーバー API ユーザーが既に編集用にロードされている構成を要求すると、メソッド呼び出しは失敗し、例外がスローされます。構成が編集用に現在ロード (ロック) されているかどうかを確認する新規サーバー API 呼び出しが追加されました。

構成を編集用にロードしたユーザーがこの構成を保管するか、または更新を取り消すと、この構成のロックが解除されます。サーバー API には、構成ロックのタイムアウト値を指定できるオプションがあります。構成のタイムアウト値に到達するとロックが解除されるため、構成をロックしたユーザーがこの構成を保管するには、構成を再ロードする必要があります。

IBM Security Directory Integrator サーバー構成ファイル global.properties に新規プロパティー「api.config.lock.timeout」が追加されました。このプロパティーは、タイムアウト値を分数で指定します。このプロパティーを空のままにするか、または値 0 を設定すると、タイムアウトが設定されません。このプロパティーのデフォルト値は 0 です。タイムアウト・ロジックは IBM Security Directory Integrator サーバーの新規スレッドによりインプリメントされます。このスレッドは、「api.config.lock.timeout」に 0 より大きい値が設定される場合にのみアクティブになり、期限切れロックがあるかどうかを 30 秒ごとに調べ、期限切れロックがある場合はそのロックを解除します。

編集用にロードされた構成に対するロックを強制解除する特殊な呼び出しがサーバー API に追加されました。この呼び出しを実行できるのは、admin の役割が付与されているサーバー API ユーザーのみです。

すべての構成は、IBM Security Directory Integrator サーバー構成フォルダーを基準にした構成ファイルの相対ファイル・パスにより識別されます。

メソッド・パラメーターとして指定されるすべてのパスは、IBM Security Directory Integrator サーバー構成フォルダーを基準にした相対パスです。

以下の新規呼び出しがローカル/リモートのサーバー API セッション・オブジェクトと JMX インターフェースに追加されました。

- `public boolean releaseConfigurationLock(String aRelativePath) throws DIException;`

  指定されている構成のロックの解除は管理者により実行されます。この呼び出しは、admin の役割が付与されているユーザーのみが実行できます。
- `public boolean undoCheckOut(String aRelativePath) throws DIException;`

  指定された構成のロックを解除し、実行中の変更をすべて打ち切ります。この呼び出しは、構成ロックがタイムアウトになっていない場合に、既に構成をチェックアウトしたユーザーのみが実行できます。
- `public ArrayList listConfigurations(String aRelativePath) throws DIException;`

  指定されたフォルダー内のすべての構成のファイル名のリストを戻します。戻される構成ファイルのパスは、IBM Security Directory Integrator サーバー構成フォルダーを基準にした相対パスです。
- `public ArrayList listFolders(String aRelativePath) throws DIException;`

  指定されたフォルダーの子フォルダーのリストを戻します。
- `public ArrayList listAllConfigurations() throws DIException;`

  IBM Security Directory Integrator サーバー構成フォルダーのディレクトリー・サブツリーのすべての構成のファイル名のリストを戻します。戻される構成ファイルのパスは、IBM Security Directory Integrator サーバー構成フォルダーを基準にした相対パスです。
- `public MetamergeConfig checkOutConfiguration(String aRelativePath) throws DIException;`

指定された構成をチェックアウトします。構成を表す MetamergeConfig オブジェクトを戻し、サーバー上でその構成をロックします。

- `public MetamergeConfig checkOutConfiguration(String aRelativePath, String aPassword) throws DIException;`

  指定されたパスワード保護構成をチェックアウトします。構成を表す MetamergeConfig オブジェクトを戻し、サーバー上でその構成をロックします。

- `public void checkInConfiguration(MetamergeConfig aConfiguration, String aRelativePath) throws DIException;`

  指定された構成を保管し、ロックを解除します。チェックアウト時に一時構成インスタンスが開始されている場合は、この構成インスタンスも停止されます。

- `public void checkInConfiguration(MetamergeConfig aConfiguration, String aRelativePath, boolean aEncrypt) throws DIException;`

  指定された構成を暗号化して保管し、ロックを解除します。チェックアウト時に一時構成インスタンスが開始されている場合は、この構成インスタンスも停止されます。

- `public void checkInAndLeaveCheckedOut(MetamergeConfig aConfiguration, String aRelativePath) throws DIException;`

  指定された構成をチェックインし、この構成をチェックアウトしたままの状態にします。構成のロックのタイムアウトがリセットされます。

- `public MetamergeConfig createNewConfiguration(String aRelativePath, boolean aOverwrite) throws DIException;`

  新規の空の構成を作成し、即時にチェックアウトします。指定されたパスの構成が既に存在しており、aOverwrite パラメーターが false に設定されている場合は、操作が失敗し、例外がスローされます。

- `public ConfigInstance checkOutConfigurationAndLoad(String aRelativePath) throws DIException;`

  指定された構成をチェックアウトし、サーバーで一時構成インスタンスを開始します。この構成インスタンスは、構成がチェックアウトされた時点、または構成のロックの期限が切れた時点で停止されます。このメソッドは ConfigInstance オブジェクトを戻します。ConfigInstance オブジェクトを使用して MetamergeConfig オブジェクトを取得できます。

- `public ConfigInstance checkOutConfigurationAndLoad(String aRelativePath, String aPassword) throws DIException;`

  指定されたパスワード保護構成をチェックアウトし、サーバーで一時構成インスタンスを開始します。この構成インスタンスは、構成がチェックアウトされた時点、または構成のロックの期限が切れた時点で停止されます。このメソッドは ConfigInstance オブジェクトを戻します。ConfigInstance オブジェクトを使用して MetamergeConfig オブジェクトを取得できます。

- `public ConfigInstance createNewConfigurationAndLoad(String aRelativePath, boolean aOverwrite) throws DIException;`

新規の空の構成を作成し、即時にチェックアウトし、サーバーに一時構成インスタンスをロードします。指定されたパスの構成が既に存在しており、aOverwrite パラメーターが false に設定されている場合は、操作が失敗し、例外がスローされます。この一時構成インスタンスは、構成がチェックアウトされた時点、または構成のロックの期限が切れた時点で停止されます。このメソッドは ConfigInstance オブジェクトを戻します。ConfigInstance オブジェクトを使用して MetamergeConfig オブジェクトを取得できます。

- `public boolean isConfigurationCheckedOut(String aRelativePath) throws DIException;`

  指定された構成がサーバーでチェックアウトされているかどうかを調べます。

一時構成インスタンスを使用した編集のためのロード

ここに記載されている情報を使用することで、一時構成インスタンスを使用した編集のためのロードの必要性およびプロセスについて知ることができます。

これは、特殊な編集用ロード・メカニズムです。構成を編集用にロードするときに、一時構成インスタンスも開始される点が異なります。これにより、構成とその AssemblyLine を開発中にテストできます。これは、IBM Security Directory Integrator 構成エディターなどの開発ツールを使用する場合に特に便利です。

構成インスタンスは、構成のロックが解放された時点、または構成のロックの有効期限が切れた時点で自動的に停止されます。

一時構成インスタンスは、サーバーでの一般的な実行時間の長い構成インスタンスから独立しています。構成 rs.xml の通常の構成インスタンスがサーバー上で実行中である場合に、一時構成インスタンスを使用して rs.xml 構成を編集用にロードできます。この結果、通常の長時間実行 rs.xml 構成インスタンスに加え、rs.xml ファイルから新しい一時構成インスタンスが開始されます。

一時構成インスタンスを使用して編集用にロードされる構成には、同じロック・メカニズムが適用されます。つまり、編集用のロード時に一時構成インスタンスを使用しているかどうかにかかわらず、編集用に構成をロードできるのは 1 回のみです。

構成ファイル・パスの代わりにソリューション名を使用する

ここに記載されているメソッドを確認することで、構成ファイル・パスの代わりにソリューション名を使用できます。

従来、構成インスタンスの開始、サーバー API のチェックイン/チェックアウト機能では、構成ファイルの URL (ファイル・パス) を指定する必要がありました。現在のバージョンの IBM Security Directory Integrator では、対応するソリューション名で同じサーバー API インターフェースのメソッドを渡すことができるため、これは不要になりました。AMC や CLI などのサーバー API クライアントでは、分かりにくい構成ファイルパスの代わりに、ユーザーから分かりやすいソリューション名を指定できるので、ユーザーにとっては便利です。

構成ファイル・パスのソリューション名に対する優位性

構成ファイル・パスは、ソリューション名よりも優先度が高くなっています。つまり、構成インスタンスを開始するメソッドにストリング (構成ファイル・パスまたは対応するソリューション名のいずれか) が渡され、構成ファイル・パスが有効な場合、メソッドはこの構成ファイルへの参照としてこの値を扱います。したがって、ストリングとして同一の構成ファイルとソリューション名がある場合、構成ファイル・パスが優先されます。この動作によって、ソリューション名がない以前のバージョンの IBM Security Directory Integrator との互換性が確保されます。

configs フォルダー

ソリューション名で参照できるのは、IBM Security Directory Integrator サーバーの起動時に、IBM Security Directory Integrator の *configs* フォルダー内に存在する構成ファイルだけです。

ソリューション名、実行名の詳細と構成方法については、「*リファレンス*」の『構成ファイル内の任意指定の構成インスタンス ID』も参照してください。

構成更新に関するサーバー API イベント

ロックされていた構成を IBM Security Directory Integrator サーバーに保管することにより、サーバー API イベント `di.ci.file.updated` を実行できます。

サーバー API イベント `di.ci.file.updated` は、ロックされていた構成が IBM Security Directory Integrator サーバーに保管されるたびに実行されます。

この通知により、サーバー API クライアントに対し、使用中の構成が変更されたこと (最新バージョンを反映するための再ロードなど) が通知されます。

システム・キューの処理

ここに記載されている情報とリンクを使用することで、システム・キューを処理できます。

システム・キューとは、IBM Security Directory Integrator の内部オブジェクトと IBM Security Directory Integrator コンポーネントで汎用目的のキューとして使用できる IBM Security Directory Integrator サーバー・モジュールです。システム・キューの目的は、JMS プロバイダーに接続し、一般メッセージと IBM Security Directory Integrator 項目オブジェクトを JMS メッセージ・キューから取得し、JMS メッセージ・キューに入れる機能を提供することです。システム・キューは各種 IBM Security Directory Integrator JMS ドライバーを使用して各種 JMS プロバイダーに接続できます。システム・キューの詳細については、「*インストールと管理*」の、システム・キューのセクションを参照してください。

システム・キュー機能は、ローカル/リモートのサーバー API インターフェースとサーバー API の JMX レイヤーを介して公開されます。ローカル IBM Security Directory Integrator サーバーの Java 仮想マシンで稼働する IBM Security Directory Integrator コンポーネントとサブシステムは、ローカル・サーバー API インターフェースを使用してシステム・キューと対話することが予期されています。リモート・サーバー API クライアント・アプリケーション、およびリモート IBM Security Directory Integrator サーバーの Java 仮想マシンで稼働する IBM Security

Directory Integrator コンポーネントとサブシステムは、リモート・サーバー API インターフェースを使用することが予期されています。

IBM Security Directory Integrator サーバー API JMX レイヤーには SystemQueue MBean が含まれています。この MBean は、SystemQueue への JMX アクセスを提供します。JMX クライアントは SystemQueue JMX MBean にアクセスし、JMX を介してシステム・キューを操作できます。

サーバー API からシステム・キューにアクセスするには、その前にシステム・キューが適切に構成されている必要があります。システム・キューを構成する簡単な手順を以下に示します。

- JMS プロバイダーをセットアップします。

  IBM Security Directory Integrator には、すぐに使用可能な MQ Everyplace JMS プロバイダーがあります。*mqeconfig* コマンド行ユーティリティーを使用して MQe キュー・マネージャーをセットアップできます (*mqeconfig* ユーティリティーはインストールされている IBM Security Directory Integrator の jars/plugins サブフォルダーにあります)。

  mqeconfig.props 構成ファイルを変更します。
  - MQe キュー・マネージャーを配置するフォルダーを指定します。

    ```
    serverRootFolder=C:¥¥TDI¥¥MQePWStore
    ```
  - IBM Security Directory Integrator サーバーの IP アドレスを指定します。

    ```
    serverIP=127.0.0.1
    ```
  - 構成オプションを設定し、MQe キュー・マネージャーを作成します。

    ```
    mqeconfig mqeconfig.props create server
    ```
  - テスト用のキューを作成します。

    ```
    mqeconfig mqeconfig.props create queue myqueue
    ```
- `global.properties` または `solution.properties` でシステム・キューと JMS プロバイダーを構成します。
  - システム・キューの使用を有効にします。

    ```
    systemqueue.on=true
    ```
  - システム・キューの JMS ドライバーを MQ Everyplace に設定します。

    ```
    systemqueue.jmsdriver.name=com.ibm.di.systemqueue.driver.IBMMQe
    ```
  - MQe キュー・マネージャーの構成ファイルを設定します (これは mqeconfig ユーティリティーにより生成されたファイルです)。

    ```
    systemqueue.jmsdriver.param.mqe.file.ini=C:¥TDI¥MQePWStore¥pwstore_server.ini
    ```

:

注: スタンドアロン Java プログラムがサーバー API を介してシステム・キューを適切に操作できるようにするため、JMS インプリメンテーションをプログラムの CLASSPATH に含める必要があります。 IBM Security Directory Integrator とともに配布される JMS インプリメンテーション (`jars/3rdparty/IBM/ibmjms.jar`) を使用できます。

サーバー API を介したシステム・キューへのアクセス

ここに記載されているコードを使用して、サーバー API を介してシステム・キューにアクセスできます。

サーバー API セッションが開始されたら、以下の方法でシステム・キューにアクセスできます。

```
import com.ibm.di.api.remote.SystemQueue;
...
SystemQueue systemQueue = session.getSystemQueue();
```

システム・キューへのメッセージの入力

ここに記載されているコードを使用することで、システム・キューにメッセージを入力できます。

「myqueue」というキューにテキスト・メッセージを入れるコードを以下に示します (この呼び出しでは、指定されたキューは自動的に作成されません。最初にキューを手動で作成する必要があります)。

```
systemQueue.putTextMessage("myqueue", "mytextmessage");
```

システム・キューからのメッセージの取得

ここに記載されているコードを使用することで、システム・キューからメッセージを取得できます。

「myqueue」というキューからテキスト・メッセージを取り出すコードを以下に示します (キューが存在している必要があります)。このメソッド呼び出しは、メッセージが取り出し可能になるまで最大 10 秒間待機します。

```
String textMessage = systemQueue.getTextMessage("myqueue", 10);
```

トゥームストーン・マネージャーの処理

ここに記載されている情報を使用することで、トゥームストーン・マネージャーを処理できます。

以前のバージョンの IBM Security Directory Integrator では、強制終了された AssemblyLine や構成は追跡されませんでした。したがって、管理者は AssemblyLine の最終実行時点を確認するには、それぞれの AssemblyLine のログを参照する以外に方法がありませんでした。AssemblyLine を開始するバンドラーは、AssemblyLine の終了後にはその状況を照会できませんでした。

この解決策として導入されたトゥームストーン・マネージャーは、各 AssemblyLine と構成のレコード (「トゥームストーン」) を終了時に作成します。このレコードには、終了状況およびその他の情報が含まれており、後でサーバー API を介してこれらの情報を要求できます。

グローバル一意識別子 (GUID)

サーバー API により作成されたグローバル一意識別子 (GUID) を使用することで、構成インスタンスおよび AssemblyLine インスタンスを一意に識別できます。

GUID は、特定の IBM Security Directory Integrator サーバーによりこれまでに作成された構成インスタンス、AssemblyLine、または EventHandler の各インスタンスの (古いバージョンの IBM Security Directory Integrator の) 固有ストリング値です。

GUID は、構成インスタンス/AssemblyLine のオブジェクト・ハッシュ・コードを表すストリングと、構成インスタンス/AssemblyLine の開始時刻 (ミリ秒) を表すストリングを連結したものとして定義されます。

構成インスタンスおよび AssemblyLine のサーバー API インターフェースで、メソッド `String getGlobalUniqueID ();` が使用可能です。

AssemblyLine および構成インスタンスのサーバー API 停止イベントで、フィールド GlobalUniqueID が使用可能です。

サーバー API によるトゥームストーン・マネージャーのサポート

ここに記載されている情報とサンプル・コードを使用することで、トゥームストーン・マネージャーのサーバー API サポートを提供できます。

トゥームストーンについて

サーバー API に新規クラス `com.ibm.di.api.Tombstone` が追加されました。このクラスのインスタンスは、トゥームストーン・オブジェクトを表します。Tombstone クラスの公開インターフェースを以下に示します。

```
public class Tombstone implements Serializable {

    public int getComponentTypeID ()

    public int getEventTypeID ()

    public java.utilDate getStartTime ()

    public java.utilDate getTombstoneCreateTime ()

    public String getComponentName ()

    public String getConfigID ()

    public int getExitCode ()

    public String getErrorDescription ()

    public String getGUID ()

    public Entry getStat ()

    public String getUserMessage ()

}
```

トゥームストーンの取得

トゥームストーンは、トゥームストーン・マネージャーを使用して検索できます。トゥームストーン・マネージャーへは、サーバー API から以下のようにアクセスできます。

```
import com.ibm.di.api.remote.TombstoneManager;
...
TombstoneManager tombstoneManager = session.getTombstoneManager();
```

トゥームストーン・マネージャーでは、特定のトゥームストーンを検索できます。先週作成されたすべてのトゥームストーンを反復処理するコードを以下に示します。

```
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DATE, -7);

Tombstone[] tombstones = tombstoneManager.getTombstones(calendar.getTime(), new Date());
```

```
for (int i = 0; i < tombstones.length; ++i) {
System.out.println("Tombstone found for : "+tombstones[i].getComponentName());
System.out.println("¥t GUID : "+tombstones[i].getGUID());
System.out.println("¥t statistics : "+tombstones[i].getStatistics());
}
```

特定の AssemblyLine のすべてのトゥームストーンを検索するには、以下のようにします (この例では AssemblyLine は「myline」、構成 ID は「C__TDI_myconfig.xml」です)。

```
Tombstone[] alTombstones = tombstoneManager.getAssemblyLineTombstones("AssemblyLines/myline",
 "C__TDI_myconfig.xml");
```

トゥームストーン・マネージャーを照会するための新規サーバー API 呼び出しを以下に示します。これらの呼び出しは、com.ibm.di.api.local.TombstoneManager インターフェースのメソッドです。

- `Tombstone getTombstone (String aGUID)`

  指定された GUID により一意的に識別される 1 つのトゥームストーン・オブジェクトを戻します。

- `Tombstone[] getAssemblyLineTombstones (String aAssemblyLineName, String aConfigID)`

  指定された AssemblyLine の使用可能なトゥームストーンをすべて戻します。

- `Tombstone[] getAssemblyLineTombstones (String aAssemblyLineName, String aConfigID, java.util.Date aStartTime, java.util.Date aEndTime)`

  指定された AssemblyLine の使用可能なトゥームストーンであり、かつ aStartTime と aEndTime により指定される間隔内のタイム・スタンプを持つトゥームストーンをすべて戻します。

- `Tombstone[] getConfigInstanceTombstones (String aConfigID)`

  指定された構成インスタンスの使用可能なトゥームストーンをすべて戻します。

- `Tombstone[] getConfigInstanceTombstones (String aConfigID)`

  指定された構成インスタンスの使用可能なトゥームストーンをすべて戻します。

- `Tombstone[] getTombstones (java.util.Date aStartTime, java.util.Date aEndTime)`

  aStartTime と aEndTime により指定される間隔内のタイム・スタンプを持つ使用可能なトゥームストーンをすべて戻します。

トゥームストーンの削除

不要になったトゥームストーンは削除します。

過去 1 週間のトゥームストーンをすべて削除するコードを以下に示します。

```
tombstoneManager.deleteTombstones(7);
```

古いトゥームストーン・レコードを削除するためのサーバー API 呼び出しを以下に示します。

- `int deleteTombstones (int aDays)`

指定された日数よりも古いトゥームストーンをすべて削除します。削除したトゥームストーン・レコードの数を戻します。

- `int keepMostRecentTombstones (int aMostResentToKeep)`

  このメソッドの実行後には、*aMostRecentToKeep* の数の最新トゥームストーン・レコードのみが維持され、その他のレコードはすべて削除されます。削除したトゥームストーン・レコードの数を戻します。

- `int deleteALTombstones (String aAssemblyLineName, String aConfigID)`

  指定された AssemblyLine のトゥームストーンをすべて削除します。削除したトゥームストーン・レコードの数を戻します。

- `int deleteALTombstones (String aAssemblyLineName, String aConfigID, int aDays)`

  指定された AssemblyLine のトゥームストーンのうち、指定された日数よりも古いトゥームストーンをすべて削除します。削除したトゥームストーン・レコードの数を戻します。

- `int keepMostRecentALTombstones (String aAssemblyLineName, String aConfigID, int aMostResentToKeep)`

  このメソッドの実行後には、指定された AssemblyLine のトゥームストーン・レコードのうち、*aMostRecentToKeep* の数の最新トゥームストーン・レコードのみが維持され、その他のレコードはすべて削除されます。削除したトゥームストーン・レコードの数を戻します。

- `int deleteCITombstones (String aConfigID)`

  指定された構成インスタンスのトゥームストーンをすべて削除します。削除したトゥームストーン・レコードの数を戻します。

- `int deleteCITombstones (String aConfigID, int aDays)`

  指定された構成インスタンスのトゥームストーンのうち、指定された日数よりも古いトゥームストーンをすべて削除します。削除したトゥームストーン・レコードの数を戻します。

- `int keepMostRecentCITombstones (String aConfigID, int aMostResentToKeep)`

  このメソッドの実行後には、指定された構成インスタンスのトゥームストーン・レコードのうち、*aMostRecentToKeep* の数の最新トゥームストーン・レコードのみが維持され、その他のレコードはすべて削除されます。削除したトゥームストーン・レコードの数を戻します。

- `boolean deleteTombstone (String aGUID)`

  指定された GUID のトゥームストーンを削除します。指定された GUID のトゥームストーン・オブジェクトが検出され、削除された場合にのみ true を戻します。

AssemblyLine トゥームストーンへのカスタム・メッセージの追加

ここに記載されている情報とサンプル・コードを使用することで、AssemblyLine トゥームストーンにカスタム・メッセージを追加できます。

タスク・スクリプト・オブジェクトは、AssemblyLine コンテキストにおける AssemblyLine オブジェクトを表します。このため、スクリプト作成時にこのオブジェクトを使用できます。

タスク・オブジェクトのインターフェースは拡張されており、この AssemblyLine のトゥームストーンの UserMessage フィールドに保管されるカスタム・メッセージを設定するためのメソッドがあります。この新しいメソッドのシグニチャーへアクセスするには、以下に示すようにタスク・スクリプト・オブジェクトを使用します。

```
task.setTombstoneUserMessage(String aUserMessage);
```

AssemblyLine スクリプトからこのメソッドを使用して、AssemblyLine トゥームストーンの追加情報を提供できます。

トゥームストーンのユーザー・メッセージは、以下の方法で取得できます。

```
String userMessage = tombstone.getUserMessage();
```

注: ConfigInstance トゥームストーンにはユーザー定義メッセージを設定できません。

IBM Security Directory Integrator プロパティーの処理

ここに記載されている情報とサンプル・コードを使用することで、IBM Security Directory Integrator プロパティーを処理できます。

リモート・クライアントがプロパティー (またはストア)を照会、取得、設定するためには、リモート・クライアントに対してサーバーの `TDIProperties` オブジェクトへのリモート参照を提供する必要があります。リモート・クライアントは `com.ibm.di.api.remote.ConfigInstance` の以下のメソッドにより、`com.ibm.di.api.remote.TDIProperties` インターフェース・リモート参照を取得できます。

```
public TDIProperties getTDIProperties() throws DIException,RemoteException;
```

ローカル・サーバー API インターフェースでも同様のインターフェースとインプリメンテーションが使用可能です。

インターフェース・メソッドの説明については、IBM Security Directory Integrator Javadoc を参照してください。

特定の構成インスタンスの使用可能なプロパティー・ストアをすべてリストする例を以下に示します。

```
TDIProperties tdiProperties = configInstance.getTDIProperties();

List stores = tdiProperties.getPropertyStoreNames();
Iterator it = stores.iterator();
System.out.println("Available property stores :");
while (it.hasNext()) {
 String storeName = (String) it.next();
 System.out.println("¥t"+storeName);
}
```

個々のプロパティーを取得するには、それぞれの名前を使用します。グローバル・プロパティー・ストア (global.properties) で使用可能なすべてのプロパティーを出力するコードを以下に示します。

```
String storeName = "Global-Properties";
System.out.println(storeName+" store contents :");
String[] storeKeys = tdiProperties.getPropertyStoreKeys(storeName);
for (int i = 0; i < storeKeys.length; ++i) {
System.out.println("¥t"+storeKeys[i]+" : "+ tdiProperties.getProperty(storeName, storeKeys[i]));
}
```

プロパティーの値の変更と新規プロパティーの作成は、以下のように実行できます。

```
tdiProperties.setProperty(storeName, "mykey", "myvalue");
```

プロパティー・ストアからプロパティーを除去するコードを以下に示します。

```
tdiProperties.removeProperty(storeName, "mykey");
```

プロパティー・ストアの変更 (新規プロパティーの追加、プロパティー値の変更、プロパティーの除去) を反映するには、変更をコミットする必要があります。

```
tdiProperties.commit();
```

JMX レイヤー API

メソッド getTDIProperties( ) は、com.ibm.di.api.jmx.mbeans.ConfigInstanceMBean クラスに含まれています。JMX クライアントはこのクラスを使用してjavax.management.ObjectName インターフェースへの参照を取得できます。

サーバー API イベント通知のための登録

サーバー API イベント通知メカニズムを使用することで、サーバー API イベント通知を登録できます。

サーバー API には、サーバー・イベント (構成インスタンスと AssemblyLine の開始と停止など) のイベント通知メカニズムがあります。これにより、ローカル・クライアント・アプリケーションとリモート・クライアント・アプリケーションがイベント通知を登録し、各種イベントに対応することができます。

通知を登録して受信する必要があるアプリケーションは、DIEventListener インターフェースをインプリメントするリスナー・クラス(リモート・アプリケーションの場合は com.ibm.di.api.remote.DIEventListener、ローカル・アクセスの場合は com.ibm.di.api.local.DIEventListener) をインプリメントする必要があります。このクラスは、サーバー・イベントの処理を担当します。DIEventListener インターフェースの handleEvent(DIEvent aEvent) メソッドに、サーバー・イベントを処理するコードを挿入する必要があります。複数の handleEvent(DIEvent aEvent) メソッド・インプリメンテーションを使用して、必要な数のリスナー・クラスをインプリメントし、これらのリスナー・クラスをすべてイベント・リスナーとして登録できます。イベント・オブジェクトのみをログに記録するリスナーの例を以下に示します。

```
import java.rmi.RemoteException;

import com.ibm.di.api.DIEvent;
import com.ibm.di.api.DIException;
import com.ibm.di.api.remote.DIEventListener;

public class MyListener implements DIEventListener
{
  public void handleEvent (DIEvent aEvent) throws DIException, RemoteException
  {
```

```
    System.out.println("TDI Server event: " + aEvent);
    System.out.println();
  }
}
```

リスナーをインプリメントしたら、このリスナーをサーバー API に登録する必要があります。ただし、リモート・アプリケーションをインプリメントする場合は、リスナー・オブジェクトをサーバー API に登録する前に実行すべき手順として、インプリメントしたリスナーをラップするベース・リスナー・オブジェクトをインスタンス化して使用する必要があります。ベース・リスナー・クラスにより、同一 Java クラスがサーバー上で使用可能でない場合でも、各自のリスナー・クラスを使用できます。

```
DIEventListener myListener = new MyListener();
DIEventListener myBaseListener = DIEventListenerBase.createInstance(myListener, true);
```

ベース・リスナー・オブジェクトは同一 *DIEventListener* インターフェースを実装しますが、そのクラスは既にサーバーに存在しており、ローカル・クライアント・サイドのリスナー・クラスとサーバーとを結びつける役割を果たします。ベース・リスナー・オブジェクトを作成するには、*com.ibm.di.api.remote.impl.DIEventListenerBase* クラスの静的メソッド *createInstance(DIEventListener aListener, boolean aSSLon)* を呼び出します。 1 番目のパラメーター *aListener* は実際のリスナー・オブジェクトを表し、2 番目のパラメーターはサーバー API が SSL を使用するかどうかを指定します (これは、このリスナー・オブジェクトで SSL を使用するかどうかをユーザーが選択するためのオプションではありません。ここでは、サーバー API がサーバー・サイドでどのように構成されているかをユーザーが指定する必要があります。指定しないと、このリスナーの通信が失敗します)。

リスナー・オブジェクト (リモート・アクセスの場合はベース・リスナー) が準備できている場合は、セッション・オブジェクトを使用してイベント通知を登録できます。

```
session.addEventListener(myBaseListnener, "di.*", "*");
```

セッション・オブジェクトの *addEventListener(DIEventListener aListener, String aTypeFilter, String aIdFilter)* メソッドによりリスナーが登録されます。1 番目のパラメーター *aListener* はリスナー・オブジェクト (リモート・アクセスの場合はベース・リスナー・オブジェクト) であり、*aTypeFilter* と *aIdFilter* では受信するイベントのタイプを指定できます。

- *aTypeFilter* は、受信するイベント・オブジェクトのタイプを指定します。現在サポートされているイベントは以下のとおりです。
  - **di.ci.start** – 構成インスタンスが開始されました。
  - **di.ci.stop** – 構成インスタンスが停止しました。
  - **di.al.start** – AssemblyLine が開始されました。
  - **di.al.stop** – AssemblyLine が停止しました。
  - **di.ci.file.updated** – 構成ファイルが変更されました。
  - **di.server.stop** – IBM Security Directory Integrator サーバーがシャットダウンしました。

  `di.al.start` などの特定のイベント・タイプを指定するか、または「`*`」ワイルドカードを使用してフィルターを指定することができます。例えば `di.al.*` と指定

すると、AssemblyLine に関連するすべてのサーバー・イベントを listen する目的でリスナーが登録され、* または NULL というタイプ・フィルターを指定すると、すべてのイベントを listen する目的でリスナーが登録されます。

- *aIdFilter* は、*aTypeFilter* が「*」または NULL に設定されていない場合にのみ有効です。これにより、イベントに関連するオブジェクトに基づいてイベントがフィルターされます。AssemblyLine の場合は AssemblyLine 名、構成インスタンスの場合は構成インスタンス ID です。例えば、`addEventListener(myListnener, "di.al.start", "MyAssemblyLine")` を使用してリスナーを登録すると、「MyAssemblyLine」という AssemblyLine が開始された時点でのみイベントが送信され、その他のサーバー・イベントは受信しません。

特定の時点でサーバー API に既に登録されているリスナーからのイベント通知の受信を停止するには、リスナーを登録抹消する必要があります。この処理を実行するには、登録時と同じセッション・オブジェクトを使用し、以下のように呼び出します。

```
session.removeEventListener(myListener);
```

サーバー・シャットダウン・イベント

新しく追加されたサーバー API イベント通知を使用することで、サーバー・シャットダウン・イベントをシグナル通知できます。

このイベントは、ローカル・コンテキストとリモート・コンテキストにおいて、サーバー API クライアントと JMX クライアントに対して使用可能です。サーバー API と JMX 通知レイヤーのどちらでも、このイベント・タイプは「di.server.stop」です。このイベント・オブジェクトは、追加ユーザー・データとしてサーバーのブート時刻を通知します。

サーバー API カスタム・イベント通知

カスタム、つまりユーザー定義のイベント通知を送信するために新たに追加されたサーバー API 機能を使用できます。

以下の新規呼び出しがローカル/リモートのサーバー API セッション・オブジェクトと *DIServer* MBean に追加されました。これにより、JMX コンテキストからもアクセスできます。

```
public void sendCustomNotification (String aType, String aId, Object aData)
```

このメソッドを呼び出すと、新規ユーザー定義イベント通知がブロードキャストされます。このメソッドに渡す必要があるパラメーターの内容は、標準サーバー API 通知の個別パラメーターと同一です。 aType パラメーターは、イベントのタイプを指定します。ユーザーにより指定される値には、接頭部 `user.` が追加されます。例えば、ユーザーから渡されたタイプが `process.X.completed` の場合、イベント・ブロードキャストのタイプは `user.process.X.completed` になります。クライアント・アプリケーションですべてのカスタム・イベントを登録するには、`user.*` というタイプ・フィルターを指定します。このイベントの発生元オブジェクトを指定するには、aId パラメーターを指定します。標準のサーバー API イベントでは、この値を使用して構成インスタンスまたは AssemblyLine を指定します。aData パラメーターを使用して、このイベントに関連する任意の追加データを渡すことができます。イベントがリモート・コンテキストで送受信される場合は、このオブジェクトはシリアライズ可能である必要があります。

ログ・ファイルへのアクセス

ここに記載されている情報とサンプル・コードを使用することで、ログ・ファイルにアクセスできます。

712 ページの『リスナーを使用した AssemblyLine の開始』では、リスナーを使用して AssemblyLine のログ・メッセージを作成と同時にリアルタイムで取得する方法について説明しました。

サーバー API には、AssemblyLine により作成されるログ・ファイルに直接アクセスするためのもう 1 つのメカニズムがあります。このメカニズムでは、AssemblyLine の SystemLog ロガーにより生成されるログ・ファイルのみにアクセスできます。

ログ・ファイルにアクセスするために、AssemblyLine のサーバー API オブジェクトを参照する必要はありません。また、既に終了している AssemblyLine の古いログにもアクセスできます。

まず、SystemLog オブジェクトを取得します。

```
SystemLog systemLog = session.getSystemLog();
```

次に、AssemblyLine により生成されたすべてのログ・ファイルを要求できます。

```
String[] alLogFileNames = systemLog.getALLogFileNames("C__Dev_TDI_rs.xml", "TestAL");
if (alLogFileNames != null) {
  System.out.println("Availalbe AssemblyLine log files:");
  for (int i=0; i<allLogFileNames.length; i++) {
    System.out.println(alLogFileNames[i]);
  }
}
```

getALLogFileNames(String aConfigId, String aALName) メソッドには、構成 ID と AssemblyLine の名前が渡されます (構成 ID についての詳細は、706 ページの『構成インスタンスの停止』を参照)。これにより、指定された AssemblyLine の実行時に生成されたすべてのログ・ファイルの名前が格納された配列が戻されます。

AssemblyLine の最終実行にのみ関心がある場合は、最終実行のログ・ファイルの名前のみを戻すサーバー API 呼び出しがあります。

```
String lastALLogFileName = systemLog.getALLastLogFileName("C__Dev_TDI_rs.xml", "TestAL");
System.out.println("AssemblyLine last log file name: " + lastALLogFileName);
```

ログ・ファイル名が判明したら、そのログ・ファイルの実際の内容を取得できます。

```
String alLog = systemLog.getALLog("C__Dev_TDI_rs.xml", "TestAL", lastALLogFileName);
System.out.println("TestAL AssemblyLine log: ");
System.out.println(alLog);
```

ログ・ファイルが非常に大きい場合には、ログの最後の部分のみを取得できます。ログ・ファイルの最終 10 KB 部分のみを取得することを指定するサンプル・コードを以下に示します。

```
String alLog = systemLog.getALLogLastChunk("C__Dev_TDI_rs.xml", "TestAL", lastALLogFileName, 10);
System.out.println("Last 10K of the TestAL AssemblyLine log: ");
System.out.println(alLog);
```

サーバー API には、古いログ・ファイルをクリーンアップ (削除) するメソッドもあります。

特定の日付よりも古いすべてのログ・ファイル (すべての構成、すべての AssemblyLine) を削除できます。1 週間以上経過しているログ・ファイルをすべて削除するサンプル・コードを以下に示します。

```
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DATE, -7);
systemLog.cleanAllOldLogs(calendar.getTime());
```

ログ・ファイルのクリーンアップ基準として、各 AssemblyLine のログ・ファイルの数も使用できます。すべての AssemblyLine のログ・ファイルのうち、最新の 5 つのログ・ファイルを除くすべてのログ・ファイルを削除するには、以下のように指定します。

```
systemLog.cleanAllOldLogs(5);
```

また、AssemblyLines のログ・ファイルのみ、または特定の AssemblyLine のログ・ファイルを削除することができます。この場合、前述の 2 つの基準 (日付とログ・ファイル数) を使用できますが、さらに AssemblyLine の名前を指定するか、またはすべての AssemblyLine のみを対象とする呼び出しを使用できます。すべてのログをクリーンアップするメソッドのシグニチャーと説明については、`com.ibm.di.api.remote.SystemLog` または `com.ibm.di.api.local.SystemLog` インターフェースの Javadoc を参照してください。

サーバー情報

IBM Security Directory Integrator サーバー自体に関する各種情報 (サーバー・バージョン、IP アドレス、オペレーティング・システム、ブート時刻、およびサーバーにインストールされており使用可能なコネクター、パーサー、関数コンポーネントについての情報など) を、サーバー API を使用して取得できます。

このような情報にアクセスできるようにするオブジェクトが `ServerInfo` です。セッション・オブジェクトを使用して `ServerInfo` オブジェクトを取得します。

```
ServerInfo serverInfo = session.getServerInfo();
```

次に、サーバー環境の詳細情報を取得して出力できます。

```
System.out.println("Server IP address: " + serverInfo.getIPAddress());
System.out.println("Server host name: " + serverInfo.getHostName());
System.out.println("Server boot time: " + serverInfo.getServerBootTime());
System.out.println("Server version: " + serverInfo.getServerVersion());
System.out.println("Server operating system: " + serverInfo.getOperatingSystem());
```

サーバーにインストールされており使用可能なすべてのコネクターのリストも出力できます。

```
String[] connectorNames = serverInfo.getInstalledConnectorsNames();
System.out.println("Connectors available on the Server: ");
for (int i=0; i<connectorNames.length; i++) {
  System.out.println(connectorNames[i]);
}
```

インストールされている各コネクターの詳細情報 (説明とバージョンを含む) を出力できます。

```
String[] connectorNames = serverInfo.getInstalledConnectorsNames();
for (int i=0; i<connectorNames.length; i++) {
  System.out.println("Installed connector: ");
  System.out.println("    name: " + connectorNames[i]);
  System.out.println("    description: " + serverInfo.getConnectorDescription(connectorNames[i]));
  System.out.println("    version: " + serverInfo.getConnectorVersionInfo(connectorNames[i]));
  System.out.println();
}
```

その他のコンポーネント (パーサーおよび関数コンポーネント) に関する情報も同様の方法で取得できます。

セキュリティー・レジストリーの使用

セキュリティー・レジストリー (特殊なサーバー API オブジェクト) を使用することで、ユーザーに付与されている権限と、ユーザーが特定のアクションを実行できるかどうかを照会できます。

これは、アプリケーションがアプリケーション自体の認証および許可のロジックを作成している場合に役立ちます。例えば、アプリケーションが IBM Security Directory Integrator サーバーとの通信に単一管理ユーザーを内部で使用しており、アプリケーション固有のユーザーと権利のセットを管理している場合などです。

セキュリティー・レジストリー・オブジェクトを使用できるのは、admin の役割が付与されているユーザーのみです。これはセッション・オブジェクトを使用して取得されます。

```
SecurityRegistry securityRegistry = session.getSecurityRegistry();
```

さまざまなユーザー権限を検査できます。例えば、`securityRegistry.userIsAdmin ("Stan")` は、Stan に admin 役割が付与されている場合に true を返します。`securityRegistry.userCanExecuteAL ("User1", "rs.xml", "TestAL")` は、Stan が構成「rs.xml」から AssemblyLine「TestAL」を実行できる場合にのみ true を返します。

使用可能なすべてのメソッドについては、
`com.ibm.di.api.remote.SecurityRegistry` のJavadoc を参照してください。

カスタム・メソッドの起動

ここに記載されているメソッドを使用することで、カスタム・メソッドの起動を実行できます。

場合によっては、独自の機能をインプリメントし、サーバー API からローカルおよびリモートの両方でこの機能にアクセスできるようにする必要があります。IBM Security Directory Integrator 6.0 のサーバー API ではこの処理がサポートされていましたが、IBM Security Directory Integrator クラスパスに独自の JAR ファイルをドロップし、リモート・サーバー API から RMI を使用せずにこの JAR ファイルにアクセスできるようにするため、この処理を単純化する必要がありました。

2 つのメソッドが以下のインターフェースに導入されました。

- `com.ibm.di.api.remote.Session`
- `com.ibm.di.api.local.Session`

2 つのメソッドは以下のとおりです。

```
public Object invokeCustom(String aCustomClassName, String aMethodName, Object[] aParams)
 throws DIException;
```

および

```
public Object invokeCustom(String aCustomClassName, String aMethodName,
 Object[] aParamsValue, String[] aParamsClass)
 throws DIException;
```

どちらのメソッドも、クラス名、メソッド名、およびメソッド・パラメーターにより記述されるカスタム・メソッドを呼び出します。

これらのメソッドは、カスタム・クラスの静的メソッドのみを呼び出すことができます。カスタム・クラスの静的メソッドは、カスタム・クラスのオブジェクトをインスタンス化してから、カスタム・クラスのインスタンス・メソッドを呼び出すことができるため、これは制限ではありません。

2 つのメソッドの主な違いは、`invokeCustom(String, String, Object[], String[])`メソッドでは呼び出し時にパラメーターのタイプを (`paramsClass` String 配列で) 明示的に設定する必要がある点です。これは、カスタム・クラスからカスタム・メソッドを呼び出すが、メソッドを呼び出すときに NULL パラメーター値を指定する場合に便利です。パラメーターの値は NULL であるため、そのタイプを判別できず、呼び出すメソッドを判別できません。

NULL 値を指定してカスタム・メソッドを呼び出す必要がある場合は、`invokeCustom(String, String, Object[], String[])` メソッドを使用してください。このメソッドでは、メソッド・パラメーターのタイプと正確な順序を表すストリング配列のエレメントによって、該当メソッドが判別されます。ユーザーが `invokeCustom(String, String, Object[])` を使用し、オブジェクト配列に NULL 値を格納すると、例外がスローされます。

プリミティブ・タイプの処理

これらのメソッドでは、プリミティブ・タイプのパラメーターが指定されたメソッドの呼び出しはサポートされていません。Java のすべてのプリミティブ・タイプには、プリミティブ・タイプの代わりに使用できるラッパー・クラスがあります。

パラメーターを取らないカスタム・メソッド

パラメーターを取らないメソッドを呼び出す必要がある場合は、`paramsValue` オブジェクト配列を NULL に設定する必要があります (他のメソッドを使用する場合は `paramsClass` ストリング配列も指定する必要があります)。

エラー

これらのメソッドを使用するときには、さまざまな例外が発生することがあります。ローカル・セッションとリモート・セッションではこの 2 つのメソッドがサポートされていますが、サーバー API JMX ではサポートされていません。

カスタム呼び出しの有効化/無効化

`invokeCustom()` メソッドの使用を有効または無効にできます (デフォルトでは無効)。このためには、global.properties ファイルで `api.custom.method.invoke.on` という名前のプロパティーを `true` または `false` に設定します。このプロパティーの値が true に設定されている場合は、ユーザーはこれらのメソッドを使用できます。

カスタム呼び出し対象クラスの指定

これらのサーバー API メソッドにより呼び出すことができるクラスには制限があります。global.properties ファイルの `api.custom.method.invoke.allowed.classes` というプロパティーに、これらのメソッドが呼び出すことができるクラスのリストが指定されています。

これらのメソッドを使用する場合に、許可クラス・リストにないクラスが呼び出されると、例外がスローされます。このプロパティーの値は、完全修飾クラス名をコンマ、セミコロン、またはスペースで区切ったリストです。

サンプル

これらのプロパティーの値の例を以下に示します。

```
api.custom.method.invoke.on=true
api.custom.method.invoke.allowed.classes=com.ibm.MyClass,com.ibm.MyOtherClass
```

この例の 1 番目の行では、カスタム呼び出しが有効であり、2 つの invokeCustom() メソッドを使用できることが指定されています。2 番目の行では、呼び出すことができるクラスが指定されています。この場合は com.ibm.MyClass クラスと com.ibm.MyOtherClass クラスのみを呼び出すことができます。2 つの invokeCustom() メソッドのいずれかを使用して別のクラスを呼び出すと、例外がスローされます。

デフォルト

api.custom.method.invoke.on プロパティーのデフォルト値は false です。つまり、ユーザーは 2 つの invokeCustom() メソッドを使用できず、これらのメソッドのいずれかが呼び出されると例外がスローされます。

api.custom.method.invoke.allowed.classes のデフォルト値は空であり、クラスを呼び出すことはできません。つまり、カスタム呼び出しが有効であっても、2 つの invokeCustom() メソッドでクラスを呼び出すことはできません。

詳細な例

jar ファイルに以下のクラスがパッケージされており、この jar ファイルが IBM Security Directory Integrator の「jars」フォルダーに保管されているとします。

```
public class MyClass {
 public static Integer multiply(Integer a, Integer b) {
  return new Integer(a.intValue() * b.intValue());
 }
}
```

global.properties IBM Security Directory Integrator 構成ファイルに以下の行が記述されているとします。

```
api.custom.method.invoke.on=true
api.custom.method.invoke.allowed.classes=MyClass
```

この場合、クライアント・アプリケーションでは「MyClass」の「multiply」メソッドをサーバー API セッションで以下のように呼び出すここができます。

```
Integer result = (Integer) session.invokeCustom(
                           "MyClass",
                          "multiply",
          new Object[] {new Integer(3), new Integer (5)});
```

JMX レイヤー

JMX レイヤーを使用することで、JMX インターフェースを介してローカルおよびリモート (JMX リモート API 1.0 を使用) の両方ですべてのサーバー API 呼び出しを公開できます。

サーバー API には JMX レイヤーがあります。

ローカル・アクセスおよびリモート・アクセスが可能となるようにサーバー API の JMX レイヤーをオンにしてセットアップする方法については、IBM Security Directory Integrator の資料で「インストールと管理」セクションの『リモート・サーバー』セクションを参照してください。

JMX レイヤーへのローカル・アクセス

ここに記載されているサンプル・コードを使用することで、JMX レイヤーへのローカル・アクセスを取得できます。

ローカル・サーバー JVM から JMX MBeanServer オブジェクトへの参照を取得するには、以下のように呼び出します。

```
import com.ibm.di.api.jmx.JMXAgent;
import javax.management.MBeanServer;

...

MBeanServer jmxMBeanServer = JMXAgent.getMBeanServer();
```

com.ibm.di.api.jmx.JMXAgent クラスの getMBeanServer() 静的メソッドから戻される MBeanServer JMX オブジェクトは、サーバー API の JMX レイヤーのすべての MBean へのエントリー・ポイントを表します。また、戻された MBeanServer オブジェクトを使用して JMX 通知を登録できます。

注: getMBeanServer() メソッドが呼び出された時点でサーバー API の JMX レイヤーが初期化されていないと、例外がスローされます。

JMX レイヤーへのリモート・アクセス

ここに記載されている情報とサンプル・コードを使用することで、JMX レイヤーへのリモート・アクセスを取得できます。

サーバー API へのリモート JMX アクセスは、JMX Remote API 1.0 仕様に基づいてインプリメントされています。

リモート・アクセスには以下の JMX サービス URL を使用する必要があります。

```
service:jmx:rmi://<SDI_Server_host>/jndi/rmi://<SDI_Server_host>:<SDI_Server_RMI_port>/jmxconnector
```

<SDI_Server_host> と <SDI_Server_RMI_port> を、それぞれ IBM Security Directory Integrator サーバーのホストと RMI ポートに置き換える必要があります (例: service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector)。

リモート JMX 接続の確立方法を示すサンプル・コードを以下に示します。

```
import javax.management.MBeanServerConnection;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

...

JMXServiceURL jmxUrl = new
  JMXServiceURL("service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector");
JMXConnector jmxConnector = JMXConnectorFactory.connect(jmxUrl);
MBeanServerConnection jmxMBeanServer = jmxConnector.getMBeanServerConnection();
```

ローカル JMX アクセスと同様に、*MBeanServerConnection* オブジェクトは、サーバー API の JMX レイヤーのすべての MBean と通知へのエントリー・ポイントです。

例えば、JMX サーバーで使用可能なすべての MBean をリストできます。

```
Iterator mBeans = jmxMBeanServer.queryNames(null, null).iterator();
while (mBeans.hasNext()) {
  System.out.println("MBean: " + mBeans.next());
}
```

MBean およびサーバー API オブジェクト

ここでは、Mbean の使用について知ることができるほか、使用可能なサーバー API オブジェクトのリストを確認できます。

JMX レイヤーでは、サーバー API オブジェクトが MBean によりラップされます。ただし MBean へのアクセスは簡単です。`MBeanServerConnection` オブジェクトを使用して MBean を直接ルックアップできます。

MBean レイヤーにはセッション・オブジェクトはありません (セッションとセキュリティー検査は、RMI セッションにより管理されます)。サーバー API セッション・オブジェクトに存在する構成インスタンスの作成、開始、および停止のためのメソッドは、JMX レイヤーの DIServer MBean にあります。

特定の時点における IBM Security Directory Integrator サーバーでの使用可能なサーバー API MBean のリストは以下のようになります。

- `ServerAPI:type=ServerInfo,id=192.168.113.222`
- `ServerAPI:type=ConfigInstance,id=C__Dev_TDI_11_11_fp1_rs.xml`
- `ServerAPI:type=AssemblyLine,id=AssemblyLines/longal.618794016`
- `ServerAPI:type=DIServer,id=winserver`
- `ServerAPI:type=SystemLog,id=SystemLog`
- `ServerAPI:type=SecurityRegistry,id=SecurityRegistry`
- ServerAPI:type=Notifier,id=Notifier

それぞれの構成インスタンスまたは AssemblyLine は MBean にラップされます。構成インスタンスまたは AssemblyLine が開始されると、MBean が自動的に作成され、構成インスタンスまたは AssemblyLine が終了すると、この MBean が自動的に除去されます。

使用可能なすべての MBean とそのメソッドおよび属性については、Java パッケージ `com.ibm.di.api.jmx.mbeans` の Javadoc を参照してください。

JMX 通知

ここに記載されている情報とリンクを使用することで、JMX 通知を処理できます。

サーバー API の JMX レイヤーには、すべてのサーバー API イベントのローカル/リモート通知機能があります (719 ページの『システム・キューの処理』を参照)。

Notifier MBean を使用して JMX 通知を登録する必要があります。

JMX 通知タイプは、サーバー API 通知と完全に同一です。

- di.ci.start – 構成インスタンスが開始されました。
- di.ci.stop – 構成インスタンスが停止しました。
- di.al.start – AssemblyLine が開始されました。
- di.al.stop – AssemblyLine が停止しました。
- di.ci.file.updated – 構成ファイルが変更されました。
- di.server.stop – IBM Security Directory Integrator サーバーがシャットダウンしました。

JMX の例 - IBM Security Directory Integrator および MC4J 構成

この例を使用することで、MC4J からサーバー API JMX レイヤーにアクセスできるように MC4J および IBM Security Directory Integrator をセットアップする方法について知ることができます。

IBM Security Directory Integrator 側

ここに記載されている情報を使用することで、リモート・サーバー API および JMX のセットアップについてと、コマンド行からの IBM Security Directory Integrator サーバーの開始について知ることができます。

リモート・サーバー API および JMX のセットアップ:

ここに記載されているサンプル・コードを使用することで、リモート・サーバー API および JMX をセットアップできます。

global.properties ファイルまたは solution.properties ファイルで以下のプロパティーを設定します (長い 16 進値がこの資料の端からはみ出ている可能性があります)。

```
## Server API properties
## ---------------------

api.on=true
api.user.registry=serverapi/registry.txt
api.user.registry.encryption.on=false

api.remote.on=true
api.remote.ssl.on=false
api.remote.ssl.client.auth.on=true
api.remote.naming.port=1099
# api.remote.server.ports=8700-8900
api.truststore=testserver.jks
{protect}-api.truststore.pass={encr}L79kdqak1afKdAyuCZBMi1GqY
/DPfD1Ipo020CVAGx/OROE2JBUTgZxLjqADXSZJgM3dHg2aWlCRwB+is
/WQa+dSVwT2hpA2kT11T7svqnIESYlcfbSg8xWxcNACdtHmdZoF7
aKSJ1cunDAxNCk0xfvMN+hXV8GK/PrneMLs1YY=

## Specifies a list of IP addresses to accept non SSL connections
from (host names are not accepted).
## Use space, comma or semicolon as delimiter between IP addresses.
This property is only taken into account
## when api.remote.ssl.on is set to false.
## api.remote.nonssl.hosts=

api.jmx.on=true
api.jmx.remote.on=true
```

注: 構成を容易にする目的で、SSL はオフになっています。

プロパティー api.remote.server.ports は RMI サービスで使用されるポートを指定します。このプロパティーは、サーバーとクライアントの間にファイアウォールが存在し、ファイアウォールがデフォルトの範囲 (8700 から 8900) の変更を必要としているときに、変更するために使用できます。

コマンド行からの IBM Security Directory Integrator サーバーの始動:

ここに記載されているサンプル・コードを使用することで、IBM Security Directory Integrator サーバーをコマンド行から開始できます。

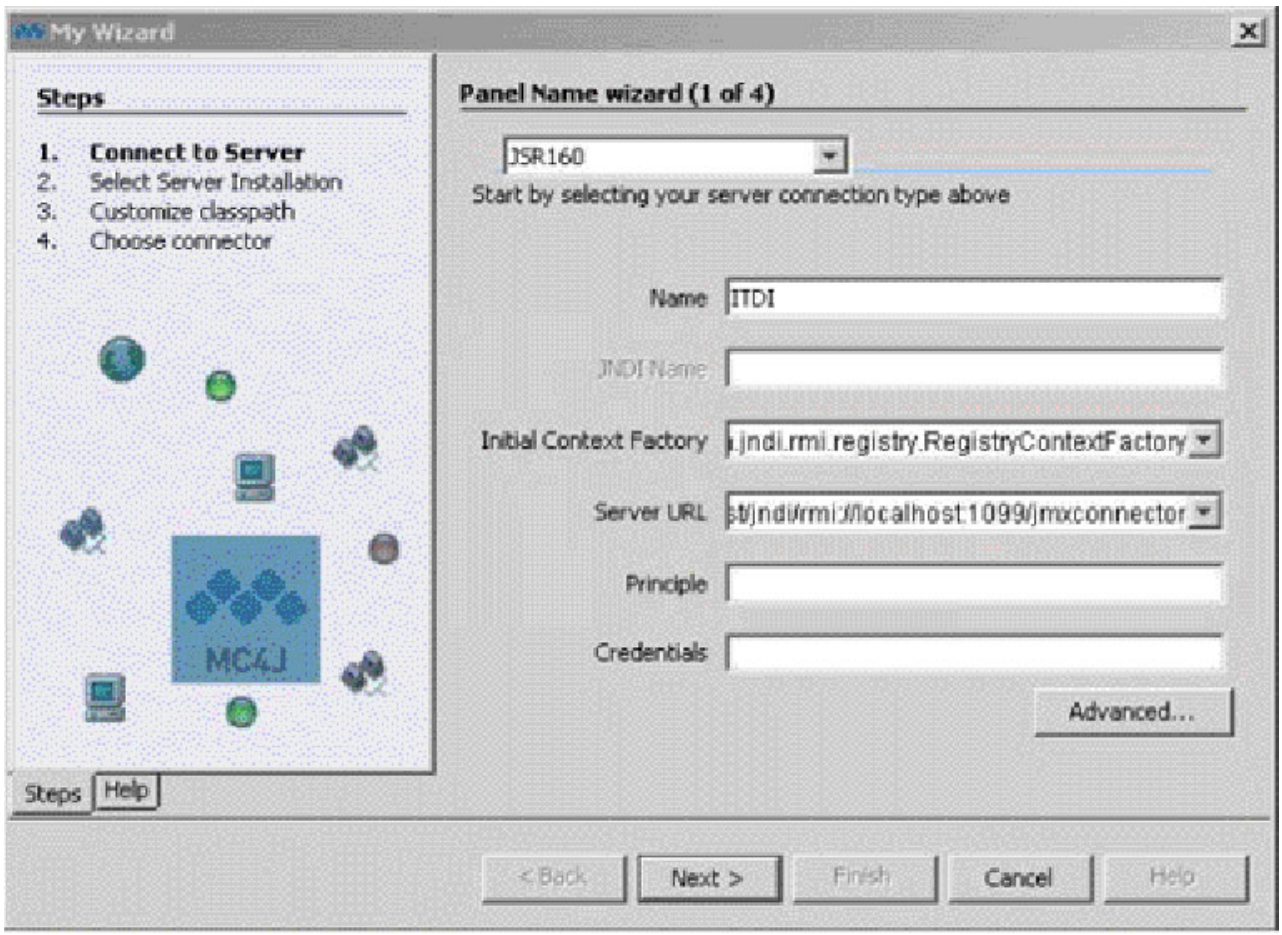
```
D:¥TDI>ibmdisrv -d

CTGDKD435I Remote API successfully started on port:1099, bound to:'SessionFactory'.
     SSL およびクライアント認証は使用不可に設定されています。
CTGDKD111I JMX Remote Server Connector started at:
     service:jmx:rmi://localhost/jndi/rmi://localhost:1099/jmxconnector.
```

MC4J 側

ここに記載されている手順を使用することで、MC4J 側を処理できます。

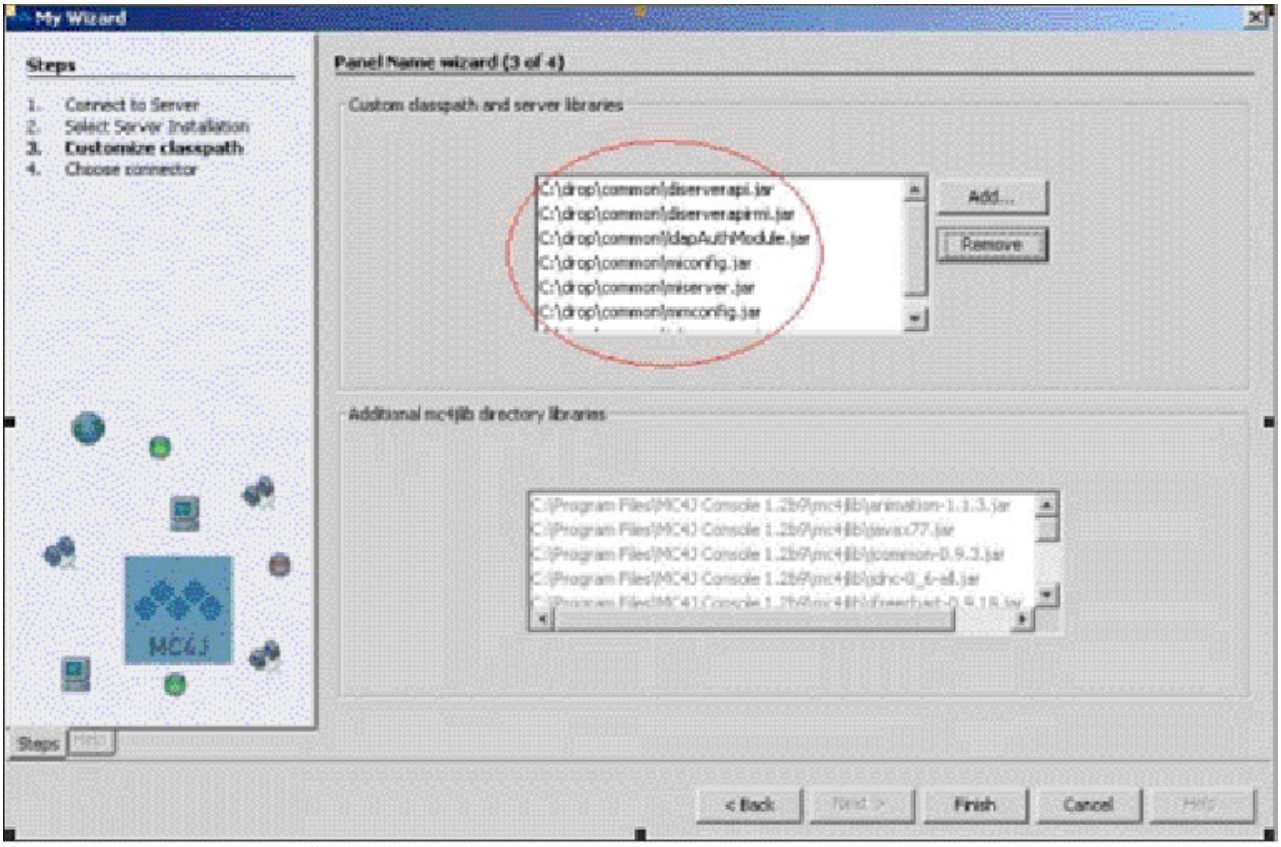
1. MC4J を http://sourceforge.net/projects/mc4j/ からダウンロードしてインストールします。
2. 「**Connect to server ...**」ウィザードを開始します。



3. 「*Name*」フィールドに **SDI** と入力します。

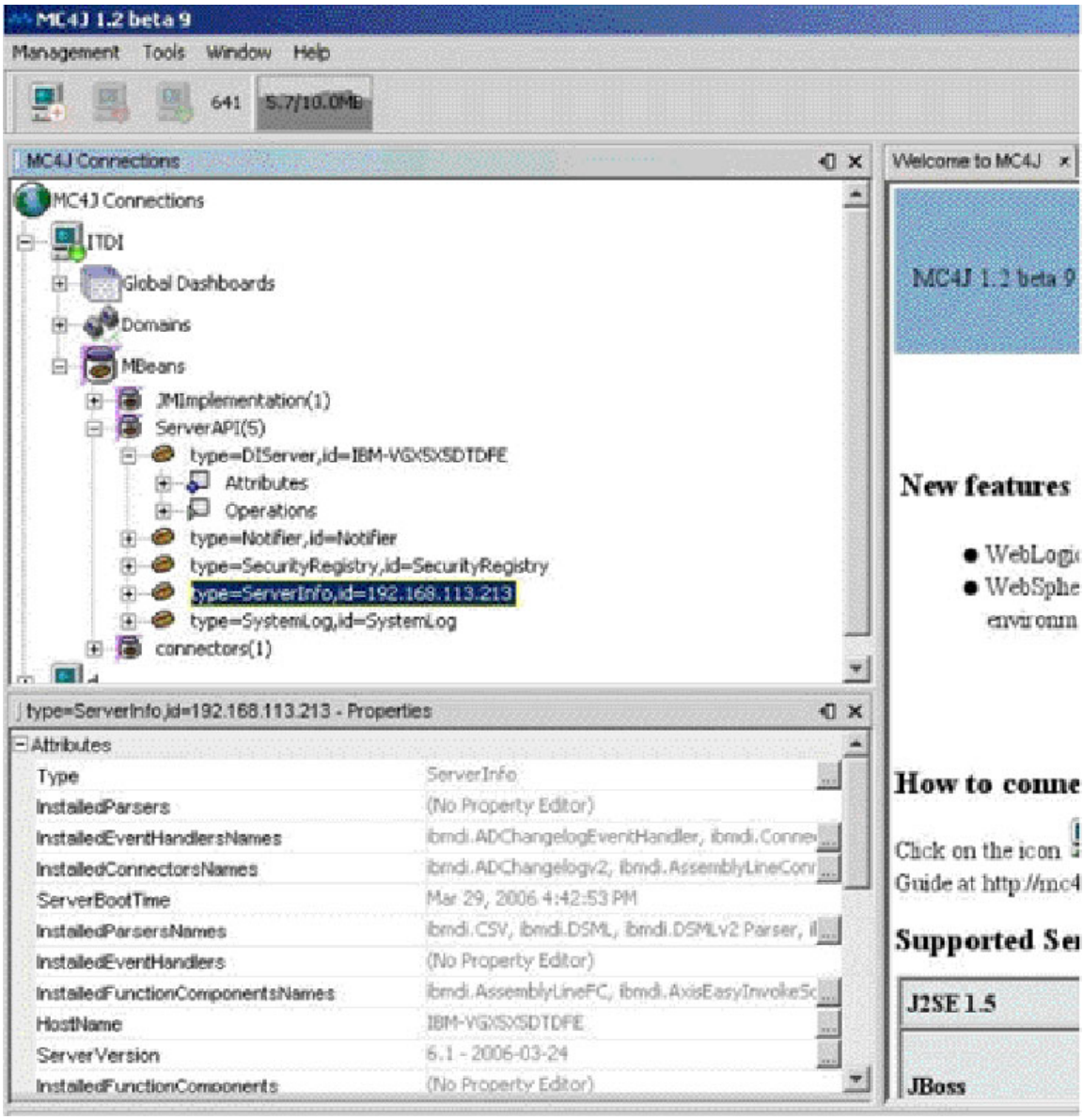
4. 「**Server URL**」テキスト・ボックスに、IBM Security Directory Integrator サーバーの始動時にダンプされた JMX 接続 URL を貼り付けます。

   注: IBM Security Directory Integrator と MC4J が異なるマシンにインストールされている場合は、ローカル・ホストを IBM Security Directory Integrator マシン IP アドレスに置き換えてください。
5. 「**Next**」を選択します。



6. 「**カスタム・クラスパスおよびサーバー・ライブラリー (Custom classpath and server libraries)**」リストに、<*TDI_install_dir*>¥jars¥common フォルダーのすべての JAR ファイルを追加します。
7. また、以下の 3 つの jar ファイルも追加します。
   - <*TDI_home*>¥jars¥3rdparty¥others¥log4j-1.2.15.jar
   - <*TDI_home*>¥jars¥3rdparty¥IBM¥icu4j_4_2.jar
   - <*TDI_home*>¥jars¥3rdparty¥IBM¥ITLMToolkit.jar
   - 「**Finish**」を選択します。

これで、MC4J が IBM Security Directory Integrator サーバーに接続されました。



以前のバージョンとの互換性

ここに記載されている情報を使用することで、JMX レイヤーの以前のバージョンとの互換性について詳しく理解できます。

シナリオ概要

ここに記載されている表内の情報を使用することで、互換性マトリックスの概要について理解できます。

インストールされている IBM Security Directory Integrator 6.0 を新規バージョンの IBM Security Directory Integrator にアップグレードする場合、以下のいずれかのシナリオに該当することがあります。

表 59. 互換性マトリックス

IBM Security Directory Integrator **サーバーのバージョン** **クライアントのバージョン**	**6.0**	**6.1**
6.0	OK	ほとんどの場合、クライアントを 6.1 に移植する必要がある。『現行バージョンのサーバーを使用する目的での V6.0 サーバー API クライアントの移植に関するガイドライン』を参照。
6.1	OK。ただし注意が必要。 741 ページの『v6.0 サーバーと現行バージョン・サーバーの両方を使用できるサーバー API クライアントのインプリメントに関するガイドライン』を参照。	OK

現行バージョンのサーバーを使用する目的での V6.0 サーバー API クライアントの移植に関するガイドライン

ここに記載されている情報とリンクを使用することで、IBM Security Directory Integrator サーバーの現行バージョンを使用するために V6.0 サーバー API クライアントを移植する際のガイドラインについて知ることができます。

ポーティングが必要な場合

現行バージョンの IBM Security Directory Integrator サーバー API での最も重要な変更点は、構成の編集方法です。IBM Security Directory Integrator での構成の編集に関する詳細については、「サーバー API の使用」の「構成の編集」セクションを参照してください。

現行バージョンの IBM Security Directory Integrator でのサーバー API の変更内容は、IBM Security Directory Integrator 6.0 サーバー API クライアントの移植に関係します。

- 構成編集関連メソッドの動作が多少変更されています。これについては、 746 ページの表 62の「変更されたメソッド」で説明します。
- IBM Security Directory Integrator 6.0 クライアントで、「推奨されないメソッド」にリストされているメソッド呼び出しを使用している場合は、代わりに IBM Security Directory Integrator の新しいメソッドを使用するように設定し直す必要があります。

IBM Security Directory Integrator 6.0 クライアントを設定し直すもう 1 つの理由は、現行バージョンの IBM Security Directory Integrator で導入された機能を使用できるようにするためです。

- 現行バージョンの IBM Security Directory Integrator で導入された各種インターフェースについては、「新規サーバー API インターフェース」で説明します。

- 一部の IBM Security Directory Integrator 6.0 インターフェースに新規メソッドが追加されました。新規メソッドのリストについては、「新規メソッド」を参照してください。

IBM Security Directory Integrator 6.0 クライアントを移植する際のもう 1 つの重要な考慮事項は、シリアライズ可能クラスの使用についてです。詳細は、 742 ページの『シリアライズ可能クラスの使用』に記載されています。サーバー API が使用する主なシリアライズ可能クラスは、IBM Security Directory Integrator 構成インターフェース・クラスです。新規のシリアライズ可能クラスは、 747 ページの表 64 の「新規シリアライズ可能クラス/インターフェース」にリストされています。構成インターフェースの詳細なリファレンスは、IBM Security Directory Integrator に付属の Javadoc に含まれています。

ポーティングがオプションである場合

IBM Security Directory Integrator 6.0 クライアントが構成編集機能を使用せず、かつ新規 IBM Security Directory Integrator サーバー API 機能を使用する必要がない場合は、IBM Security Directory Integrator 6.0 クライアントを変更する必要はありません。

v6.0 サーバーと現行バージョン・サーバーの両方を使用できるサーバー API クライアントのインプリメントに関するガイドライン

ここに記載されている情報とリンクを使用することで、IBM Security Directory Integrator サーバーの V6.0 と現行バージョンの両方を使用できるサーバー API クライアントをインプリメントできます。

サーバー API における機能拡張では以前のバージョンとの互換性が確保されているため、サーバー API クライアント・アプリケーションで、IBM Security Directory Integrator 6.0 サーバーに対しては IBM Security Directory Integrator 6.0 のすべての機能を使用できます。また、IBM Security Directory Integrator サーバーに対しては IBM Security Directory Integrator の現行バージョンのすべての新機能を使用することができます。これを実現するには、サーバー API クライアントが IBM Security Directory Integrator サーバーのバージョンを確認し、該当するバージョン固有コードを実行するようにします。この例は、 743 ページの『IBM Security Directory Integrator サーバーのバージョンの確認』に記載されています。

サーバー API クライアントと IBM Security Directory Integrator サーバーの間でデータを共用する 2 つの主要な方法を以下に示します。

- RMI リモート・オブジェクトの使用
- シリアライズ可能クラスの使用

RMI リモート・オブジェクトの使用:

ここに記載されている情報とリンクを使用することで、インプリメンテーションの実行時に RMI リモート・オブジェクトを使用できます。

この場合、サーバー API クライアントは、現行バージョンの IBM Security Directory Integrator のリモート・クラスから生成されたリモート・オブジェクト・スタブを使用します。これらのスタブには、IBM Security Directory Integrator バージョン 6.0 のリモート・クラスの既存のメソッドすべてと、IBM Security Directory

Integrator の現行バージョンで使用可能なメソッドが含まれています (新たに導入されたメソッドについては、『新規メソッド』で説明されています)。

- IBM Security Directory Integrator の現行バージョンで使用可能なメソッドを IBM Security Directory Integrator 6.0 サーバーに対して使用することはできません。これらの新規メソッドを IBM Security Directory Integrator 6.0 サーバーに対して使用しないようにするため、クライアント・サーバー API アプリケーションにより、事前にサーバーのバージョンを確認する操作が実行されます。
- 「推奨されないメソッド」で説明しているメソッドは、IBM Security Directory Integrator 6.0 サーバー以外では使用できません。推奨されないメソッドが現行バージョンの IBM Security Directory Integrator サーバーで呼び出されると、例外がスローされます。

シリアライズ可能クラスの使用:

シリアライズ可能クラスを使用する際に、ここに記載されている情報とリンクを使用できます。

IBM Security Directory Integrator 6.0 から、サーバー API のシリアライズ可能クラスと IBM Security Directory Integrator のシリアライズ可能クラスが大幅に変更されました。したがってこれらのクラスは、IBM Security Directory Integrator 6.0 と現行バージョンの IBM Security Directory Integrator では異なります。これらのクラスは大幅に変更されましたが、シリアライズの点から、以前のバージョンとの互換性を確保しています。つまり、IBM Security Directory Integrator 6.0 のシリアライズ可能クラスは、Java RMI エンジンを介して現行バージョンのシリアライズ可能クラスと相互運用できます。

Java RMI エンジンは、クラスのシリアル・バージョン UID を調べ、シリアライズ可能クラスに互換性があるかどうかを判別します。2 つのクラスのクラス・シリアル・バージョン UID が同一の場合は、RMI エンジンはこの 2 つのクラスが互換であると判断します。IBM Security Directory Integrator のシリアライズ可能クラスのシリアル・バージョン UID は、「シリアライズ可能クラスの serialVersionUID」に記載されています。現行バージョンのシリアライズ可能クラスは IBM Security Directory Integrator 6.0 のシリアライズ可能クラスと互換であるため、これらのクラスのシリアル・バージョン UID は変更されていません。

『新規シリアライズ可能クラス/インターフェース』に、現行バージョンの IBM Security Directory Integrator で使用可能なクラスとインターフェースがリストされています。これらのクラスとインターフェースは IBM Security Directory Integrator 6.0 では使用不可であるため、IBM Security Directory Integrator 6.0 サーバーに対して使用することはできません。両方のリリースにおけるシリアライズ可能クラスのメソッド・シグニチャーの変更についての詳細は、IBM Security Directory Integrator Javadoc を参照してください。 IBM Security Directory Integrator 6.0 で使用できないメソッドを IBM Security Directory Integrator 6.0 サーバーに対して使用することはできません。

サーバー API クライアントがサード・パーティーのシリアライズ可能クラスまたはカスタム・ユーザー・シリアライズ可能クラスを使用する場合の最良の方法は、サーバーとクライアントでこれらのクラスが同一であるようにすることです。何らかの理由でこれらのシリアライズ可能クラスが同一クラスの異なるバージョン (互換)

である場合は、両方のバージョンに同一の serialVersionUID が設定されていれば、クライアントは機能します。シリアライズ可能クラスの維持と変更についての詳細は、以下を参照してください。

http://java.sun.com/j2se/1.5.0/docs/api/java/io/Serializable.html
http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serialTOC.html
http://www-03.ibm.com/developerworks/blogs/page/woolf?entry=serialization_and_serial_version_uid

構成の編集:

構成の編集を実行する際に、ここに記載されている情報とリンクを使用できます。

現行バージョンの IBM Security Directory Integrator での構成編集は、IBM Security Directory Integrator 6.0 での構成編集と大幅に異なります。このため、IBM Security Directory Integrator 6.0 サーバーと以降のバージョンのサーバーとの両方の IBM Security Directory Integrator 構成をコーディングおよび編集する場合には、特に注意が必要です。このコードは IBM Security Directory Integrator バージョン固有です。このため、IBM Security Directory Integrator サーバー・バージョンを確認してコードを分岐する必要があります。これについては、『IBM Security Directory Integrator サーバーのバージョンの確認』で説明します。

認証メカニズム:

認証メカニズムを使用するときは、以下に示す説明について注意が必要です。

ユーザー名/パスワードに基づく認証メカニズムと LDAP 認証メカニズムは、IBM Security Directory Integrator 6.0 ではサポートされていません。このため、com.ibm.di.api.remote.SessionFactory インターフェースの createSession (String aUserName, String aPassword) メソッドを IBM Security Directory Integrator 6.0 サーバーに対して呼び出すと失敗します。

IBM Security Directory Integrator サーバーのバージョンの確認:

ここに記載されているサンプル・コードを使用することで、IBM Security Directory Integrator サーバーのバージョンを確認できます。

一般に、サーバー API クライアント・コードの大部分は IBM Security Directory Integrator 6.0 サーバーと現行バージョンの IBM Security Directory Integrator サーバーで共通しています。ただし場合によっては、IBM Security Directory Integrator 6.0 以降のバージョン固有コードが必要となります。これらのバージョン固有のコード部分は、サーバー・バージョンの確認を必要とします。IBM Security Directory Integrator サーバー・バージョンを取得および使用する方法を示すサンプル・コードを以下に示します。

```
import com.ibm.di.api.remote.Session;
import com.ibm.di.api.remote.ServerInfo;

...

ServerInfo serverInfo = session.getServerInfo();
if (serverInfo == null) {
    throw new Exception("Server version information is not available!");
}

String serverVersion = serverInfo.getServerVersion();
```

```
if (serverVersion.startsWith("6.1")) {
    // TDI 6.1 specific code
}
else if (serverVersion.startsWith("6.0")) {
    // TDI 6.0 specific code
}
else {
    throw new Exception("Unsupported TDI server version: " + serverVersion);
}
```

サーバー API の変更内容

ここでは、サーバー API の変更内容を確認できます。

表 60. 新規サーバー API インターフェース

名前	説明
SystemQueue	SystemQueue へのサーバー API アクセス
TDIProperties	外部プロパティー・ストアのラッパー
TombstoneManager	トゥームストーンの読み取り機能と削除機能へのアクセス

表 61. 新規メソッド

名前	説明
AssemblyLine	
String getGlobalUniqueID ()	AssemblyLine GUID を戻します。GUID は、特定の IBM Security Directory Integrator サーバーによりこれまでに作成された各コンポーネントに設定されている固有のストリング値です。
ConfigInstance	
String getGlobalUniqueID ()	構成インスタンス GUID を戻します。GUID は、特定の IBM Security Directory Integrator サーバーによりこれまでに作成された各コンポーネントに設定されている固有のストリング値です。
String[] getConnectorPoolNames ()	構成インスタンスのすべてのコネクター・プール名を戻します。
int getConnectorPoolSize (String aConnectorPoolName)	指定されたコネクター・プールのサイズを戻します。
int getConnectorPoolFreeNum (String aConnectorPoolName)	指定されたコネクター・プール内のフリー・コネクターの数を戻します。
PoolDefConfig getConnectorPoolConfig (String aConnectorPoolName)	コネクター・プール構成オブジェクトを戻します。
int purgeConnectorPool (String aConnectorPoolName)	プールを最小サイズに縮小するため、未使用のコネクターが解放されます。
TDIProperties getTDIProperties()	現在の構成に関連する TDIProperties オブジェクトを戻します。
Session	
void shutDownServer (int aExitCode)	指定した終了コードで IBM Security Directory Integrator サーバーをシャットダウンします。
TombstoneManager getTombstoneManager ()	TombstoneManager オブジェクトを戻します。このオブジェクトを使用してトゥームストーンの照会と消去を実行できます。

表 61. 新規メソッド (続き)

名前	説明
boolean isSSLon ()	現在のセッションで SSL が使用されているかどうかを検査します。
boolean releaseConfigurationLock(String aRelativePath)	指定されている構成のロックの解除は管理者により実行されます。この呼び出しは、admin の役割が付与されているユーザーのみが実行できます。
boolean undoCheckOut(String aRelativePath)	指定された構成のロックを解除し、実行中の変更をすべて打ち切ります。この呼び出しは、構成ロックがタイムアウトになっていない場合のみに、既に構成をチェックアウトしたユーザーのみが実行できます。
ArrayList listConfigurations(String aRelativePath)	指定されたフォルダー内のすべての構成のファイル名のリストを戻します。戻される構成ファイルのパスは、サーバー構成コードベース・フォルダーを基準にした相対パスです。
ArrayList listFolders(String aRelativePath)	指定されたフォルダーの子フォルダーのリストを戻します。
ArrayList listAllConfigurations()	サーバー構成コードベース・フォルダーのディレクトリー・サブツリーにあるすべての構成のファイル名のリストを戻します。戻される構成ファイルのパスは、IBM Security Directory Integrator サーバー構成コードベース・フォルダーを基準にした相対パスです。
MetamergeConfig checkOutConfiguration (String aRelativePath)	指定された構成をチェックアウトします。構成を表す MetamergeConfig オブジェクトを戻し、サーバー上でその構成をロックします。
MetamergeConfig checkOutConfiguration (String aRelativePath, String aPassword)	指定されたパスワード保護構成をチェックアウトします。構成を表す MetamergeConfig オブジェクトを戻し、サーバー上でその構成をロックします。
ConfigInstance checkOutConfigurationAndLoad (String aRelativePath)	指定された構成をチェックアウトし、サーバーで一時構成インスタンスを開始します。
ConfigInstance checkOutConfigurationAndLoad (String aRelativePath, String aPassword)	指定された構成をチェックアウトし、サーバーで一時構成インスタンスを開始します。
void checkInConfiguration (MetamergeConfig aConfiguration, String aRelativePath)	指定された構成を保管し、ロックを解除します。チェックアウト時に一時 ConfigInstance が開始されている場合は、この構成インスタンスも停止されます。
void checkInAndLeaveCheckedOut (MetamergeConfig aConfiguration, String aRelativePath)	指定された構成をチェックインし、この構成をチェックアウトしたままの状態にします。構成のロックのタイムアウトがリセットされます。
void checkInConfiguration (MetamergeConfig aConfiguration, String aRelativePath, boolean aEncrypt)	指定された構成を暗号化して保管し、ロックを解除します。チェックアウト時に一時構成インスタンスが開始されている場合は、この構成インスタンスも停止されます。
MetamergeConfig createNewConfiguration (String aRelativePath, boolean aOverwrite)	新規の空の構成を作成し、即時にチェックアウトします。指定されたパスの構成が既に存在しており、aOverwrite パラメーターが false に設定されている場合は、操作が失敗し、例外がスローされます。

表 61. 新規メソッド (続き)

名前	説明
ConfigInstance createNewConfigurationAndLoad (String aRelativePath, boolean aOverwrite)	新規の空の構成を作成し、即時にチェックアウトし、サーバーに一時構成インスタンスをロードします。指定されたパスの構成が既に存在しており、aOverwrite パラメーターが false に設定されている場合は、操作が失敗し、例外がスローされます。
boolean isConfigurationCheckedOut (String aRelativePath)	指定された構成がサーバーでチェックアウトされているかどうかを調べます。
void sendCustomNotification (String aType, String aId, Object aData)	カスタムのユーザー定義通知をすべての登録リスナーに送信します。
SystemQueue getSystemQueue()	リモート・サーバー API API SystemQueue 表現オブジェクトを取得します。
String getConfigFolderPath()	リモート・サーバーの api.config.folder プロパティーの値を完全パスとして取得します。設定しない場合は、空ストリングが戻されます。
Object invokeCustom (String aCustomClassName, String aMethodName, Object[] aParams)	指定されたクラスの指定されたメソッドを呼び出します。
Object invokeCustom (String aCustomClassName, String aMethodName, Object[] aParamsValue,String[] aParamsClass)	指定されたクラスの指定されたメソッドを呼び出します。
SessionFactory	
Session createSession (String aUserName, String aPassword)	指定されたユーザー名とパスワードを使用してセッション・オブジェクトを作成します。

表 62. 変更されたメソッド

名前	説明
ConfigInstance	
void setConfiguration (MetamergeConfig aConfiguration)	IBM Security Directory Integrator の現行バージョンでは、特定のクライアントが一時構成インスタンスを使用して同一の構成を既にチェックアウトしている場合にのみ、このメソッドを呼び出すことができます。

表 63. 推奨されないメソッド (IBM Security Directory Integrator の現行バージョン・サーバーに対して使用すべきでないメソッド。IBM Security Directory Integrator 6.0 サーバーに対してこれらのメソッドを使用する場合は、まったく問題ありません)

名前	説明
ConfigInstance	
void saveConfiguration ()	save の代わりに CheckIn メソッドを使用してください。
void saveConfiguration (boolean aEncrypt)	save の代わりに CheckIn メソッドを使用してください。
void setExternalProperties (ExternalPropertiesConfig aExPropConfig)	TDIProperties を使用してください。
void setExternalProperties (String aKey, ExternalPropertiesConfig aExPropConfig)	TDIProperties を使用してください。
void saveExternalProperties ()	TDIProperties を使用してください。

表 64. 新規シリアライズ可能クラス/インターフェース

名前	説明
com.ibm.di.api.Tombstone	5178569311755396746L
com.ibm.di.api.CIEvent	5178569311755396746L
com.ibm.di.config.interfaces.NamespaceEvent	-1857414661726671152L
com.ibm.di.config.interfaces.OperationConfig	2715909691453046036L
com.ibm.di.config.interfaces.PoolDefConfig	-12523719385177656O6L
com.ibm.di.config.interfaces.PoolInstanceConfig	55949197177690302 91L
com.ibm.di.config.interfaces.PropertyManager	4280805548502266432L
com.ibm.di.config.interfaces.PropertyStoreConfig	-26209296775588336 40L
com.ibm.di.config.interfaces.ReconnectConfig	-79356289472614776 28L
com.ibm.di.config.interfaces.TDIProperties	-33614718378886772 77L
com.ibm.di.config.interfaces.TDIPropertyStore	198251115520372634L
com.ibm.di.config.interfaces.TombstonesConfig	-32601026863913324 34L

表 65. シリアライズ可能クラスの serialVersionUID

名前	状況	serialVersionUID
com.ibm.di.api.ALEvent	以前のバージョンと互換性がある	56317722569736929 72L
com.ibm.di.config.base.ALMappingConfigImpl	以前のバージョンと互換性がある	27124936574507107 88L
com.ibm.di.server.ALState	以前のバージョンと互換性がある	6699383122608684 91L
com.ibm.di.config.base.AssemblyLineConfigImpl	以前のバージョンと互換性がある	2715909691453046036L
com.ibm.di.entry.Attribute	以前のバージョンと互換性がある	66758817449018603 29L
com.ibm.di.config.base.AttributeMapConfigImpl	以前のバージョンと互換性がある	-26190155381786656 84L
com.ibm.di.entry.AttributeValue	以前のバージョンと互換性がある	100100L
com.ibm.di.config.base.BaseConfigurationImpl	既知の問題 (748 ページの『既知の問題』を参照)	-731697997925312 5005L
com.ibm.di.config.base.BranchConditionImpl	以前のバージョンと互換性がある	-40917732335838179 12L
com.ibm.di.config.base.BranchingConfigImpl	以前のバージョンと互換性がある	-10135888843811339 44L
com.ibm.di.config.base.CallConfigImpl	以前のバージョンと互換性がある	-46974584978353290 96L
com.ibm.di.config.base.CallParamConfigImpl	以前のバージョンと互換性がある	57880211547147417 67L
com.ibm.di.config.base.CheckpointConfigImpl	以前のバージョンと互換性がある	-83423698815234684 83L
com.ibm.di.config.base.ConfigCache	以前のバージョンと互換性がある	-33112557315041744 16L
com.ibm.di.config.base.ConfigStatistics	以前のバージョンと互換性がある	-12716454573849112 49L
com.ibm.di.config.base.ConnectorConfigImpl	以前のバージョンと互換性がある	40933764562122300 00L
com.ibm.di.config.base.ConnectorSchemaConfigImpl	以前のバージョンと互換性がある	930161291800752910L
com.ibm.di.config.base .ConnectorSchemaItemConfigImpl	以前のバージョンと互換性がある	-16655981947572957 69L
com.ibm.di.config.base.ContainerConfigImpl	以前のバージョンと互換性がある	-41340044095926940 52L
com.ibm.di.config.base.DeltaConfigImpl	以前のバージョンと互換性がある	-72501284845880240 17L
com.ibm.di.api.DIEvent	以前のバージョンと互換性がある	-86645334774524912 19L
com.ibm.di.entry.Entry	以前のバージョンと互換性がある	-59614245293786257 29L

表 65. シリアライズ可能クラスの *serialVersionUID (続き)*

名前	状況	serialVersionUID
com.ibm.di.config.interfaces .ExternalPropertiesDelegator	既知の問題 (『既知の問題』を参照)	7725187425731381660L
com.ibm.di.config.base.ExternalPropertiesImpl	以前のバージョンと互換性がある	-5837658758525300221L
com.ibm.di.config.base.FormConfigImpl	以前のバージョンと互換性がある	-8761349695805705052L
com.ibm.di.config.base.FormItemConfigImpl	以前のバージョンと互換性がある	-7825109041707716857L
com.ibm.di.config.base.FunctionConfigImpl	以前のバージョンと互換性がある	5778585850194005910L
com.ibm.di.config.interfaces.GlobalRef	以前のバージョンと互換性がある	366178307603105225L
com.ibm.di.config.base.HookConfigImpl	以前のバージョンと互換性がある	-1300997546910640256L
com.ibm.di.config.base.HooksConfigImpl	以前のバージョンと互換性がある	-9160883008989377612L
com.ibm.di.config.interfaces.InheritanceLoopException	以前のバージョンと互換性がある	-5977834080357995975L
com.ibm.di.config.base.InheritConfigImpl	以前のバージョンと互換性がある	90155321639831994871
com.ibm.di.config.base.InstanceConfigImpl	以前のバージョンと互換性がある	-7052997089129596762L
com.ibm.di.config.base.LibraryConfigImpl	以前のバージョンと互換性がある	-6737181973806281819L
com.ibm.di.config.base.LinkCriteriaConfigImpl	以前のバージョンと互換性がある	-92068565361720118211
com.ibm.di.config.base.LinkCriteriaItemImpl	以前のバージョンと互換性がある	-9525392489206104521
com.ibm.di.config.base.LogConfigImpl	以前のバージョンと互換性がある	33714110721856251701
com.ibm.di.config.base.LogConfigItemImpl	以前のバージョンと互換性がある	629975046478880897 1L
com.ibm.di.config.base.LoopConfigImpl	以前のバージョンと互換性がある	-81745410745104814181
com.ibm.di.config.base.MetamergeConfigImpl	以前のバージョンと互換性がある	-33636953306859679041
com.ibm.di.config.xml.MetamergeConfigXML	以前のバージョンと互換性がある	-44031697115790297651
com.ibm.di.config.base.MetamergeFolderImpl	以前のバージョンと互換性がある	61075867535231402201
com.ibm.di.config.base.NamespaceConfigImpl	以前のバージョンと互換性がある	986964857890827079L
com.ibm.di.config.base ParserConfigImpl	以前のバージョンと互換性がある	549722149479980009 9L
com.ibm.di.config.base.PropertyConfigImpl	以前のバージョンと互換性がある	-26209296775588336401
com.ibm.di.config.base.RawConnectorConfigImpl	以前のバージョンと互換性がある	84390497169641194601
com.ibm.di.config.base.SandboxConfigImpl	以前のバージョンと互換性がある	-3993201241553733141
com.ibm.di.config.base.SchemaConfigImpl	以前のバージョンと互換性がある	17788160951047851341
com.ibm.di.config.base.SchemaItemConfigImpl	以前のバージョンと互換性がある	51688019478113765661
com.ibm.di.config.base.ScriptConfigImpl	以前のバージョンと互換性がある	-7747686242551793890L
com.ibm.di.api.remote.impl.rmi .SSLRMIClientSocketFactory	以前のバージョンと互換性がある	50830175460314203841
com.ibm.di.server.TaskCallBlock	以前のバージョンと互換性がある	115072761837771375L
com.ibm.di.server.TaskStatistics	以前のバージョンと互換性がある	20985180463768895851
com.ibm.di.api.remote.impl.rmi.RMISocketFactory	以前のバージョンと互換性がある	-32006528589297123031

既知の問題

サーバー API の処理時に発生する、クラス内の既知の問題について説明します。

com.ibm.di.config.interfaces.ExternalPropertiesDelegator

ここに記載されている情報を使用することで、
com.ibm.di.config.interfaces.ExternalPropertiesDelegator クラスの制限の概要を知ることができます。

com.ibm.di.config.interfaces.ExternalPropertiesDelegator クラスは、
com.ibm.di.config.interfaces.ExternalPropertiesConfig インターフェースのインプリメンテーション・クラスです。また、
com.ibm.di.config.interfaces.ExternalPropertiesDelegator クラスは、
com.ibm.di.config.base.BaseConfigurationImpl クラスを拡張します。

サーバー API クライアント・コードは、クラスではなくインターフェースを扱います。このため、サーバー API クライアント・ソース・コードでは ExternalPropertiesDelegator クラスは直接参照されません。IBM Security Directory Integrator 6.0 クライアントは、IBM Security Directory Integrator 6.0 サーバーから ExternalPropertiesConfig オブジェクトを取得できますが、このクライアントには、構成インスタンス・オブジェクト (com.ibm.di.api.remote.ConfigInstance) に対して setExternalProperties(String aKey, ExternalPropertiesConfig aExPropConfig) または setExternalProperties(ExternalPropertiesConfig aExPropConfig) を呼び出してサーバーの外部プロパティーを変更することはできないという制限があります。現行バージョンの IBM Security Directory Integrator クライアントから IBM Security Directory Integrator 6.0 サーバーに対してこれらのメソッドのいずれかを呼び出すと、その呼び出しは失敗します。

com.ibm.di.config.base.BaseConfigurationImpl

前述の com.ibm.di.config.interfaces.ExternalPropertiesDelegator の問題は、
com.ibm.di.config.base.BaseConfigurationImpl にも同様に該当します。(前者は後者の拡張です。)

付録 E. REST サーバー API

RESTful インターフェースを介して、非 Java ベースのクライアントから IBM Security Directory Integrator サーバーにアクセスする場合は、REST サーバー API を使用することができます。

REST は Representational State Transfer の略です。このインターフェースは、構成、ConfigInstance、AssemblyLine、PropertyStore、およびリスナーなど、既に定義済みのリソースの表記を指定することで、すべての呼び出しを既存のサーバー API に委任します。REST サーバー API ではトランスポートに HTTP/1.1 を使用し、XML および JSON の両方の構文でリソース表記を使用できるようにします。

注: RESTful インターフェースの現行バージョンでは、既存のリモート/ローカル API の完全なミラーリングを行うことはできません。最も一般的に使用されるリモート/ローカル API のみが RESTful インターフェースを介してエクスポートされています。

REST (Representational State Transfer) は、ワールド・ワイド・ウェブのような分散ハイパーメディア・システムのためのソフトウェア・アーキテクチャーのスタイルの 1 つです。REST スタイルのアーキテクチャーは、一方の複数のクライアントともう一方の 1 つのサーバーで構成されています。クライアントはサーバーへの要求を開始し、サーバーはその要求を処理して適切な応答を返します。要求および応答は、リソース表記の転送に応じて異なります。リソースは、基本的に、処理可能な任意の一貫した意味のある概念にすることができます。通常、リソースは、リソースの現在の状態または想定される状態を収集する文書で表されます。

場合によっては、クライアントはアプリケーション状態間の移行中、または「停止」状態になります。停止状態のクライアントはユーザーとの対話が可能ですが、ロードは作成せず、また、一連のサーバー上またはネットワーク上でのクライアントごとのストレージの取り込みは行いません。

クライアントは、新しい状態への移行準備が整ったときに、要求の送信を開始します。1 つ以上の要求が未処理の場合、クライアントは移行中と見なされます。各アプリケーションの状態の表記には、次回にクライアントが新しい状態の移行を開始するように選択したときに使用可能なリンクが含まれます。

本来、REST は HTTP のコンテキスト内に記述されていましたが、そのプロトコルに限られるわけではありません。RESTful アーキテクチャーは、他のアプリケーション層のプロトコルによって、有効な表現の状態の転送に基づいた、一定の規則に従う豊富な用語がアプリケーション用に既に提供されている場合、それらのプロトコルをベースとして使用できます。RESTful アプリケーションは、明確に定義された既存のインターフェースと、選択されたネットワーク・プロトコルで提供されるその他の組み込み機能を最大限に利用し、また、新しいアプリケーション固有の機能の追加を最小限に抑えます。

REST サーバー API のアーキテクチャー

ここに記載されている情報を使用することで、REST サーバー API のアーキテクチャーを理解できます。

REST サーバー API は、リソース階層の構造化に、サービス、フィード、および項目の各オブジェクトを使用するため、Atom Syndication プロトコルに基づいています。

注: Atom Syndication 仕様では、項目に Author 要素、Summary 要素、および Title 要素を含める必要があります。項目が示すリソースに上記の要素が適していない場合、また、Author と Title の両方の要素を含める必要がある Atom フィードに適用できない場合に、アプリケーションはこれらの要素を含まない項目を使用します。

REST サーバー API では、IBM Security Directory Integrator サーバーがサポートするすべてのリソースを表す他のオブジェクトも定義します。リソースのデフォルト表記は JSON ですが、クライアント・アプリケーションではそれぞれ REST サーバーを使用して XML を処理できます。この API は、適切な HTTP ヘッダーを使用して、要求ごとのレベルで構成可能です。

リソース階層のナビゲーション

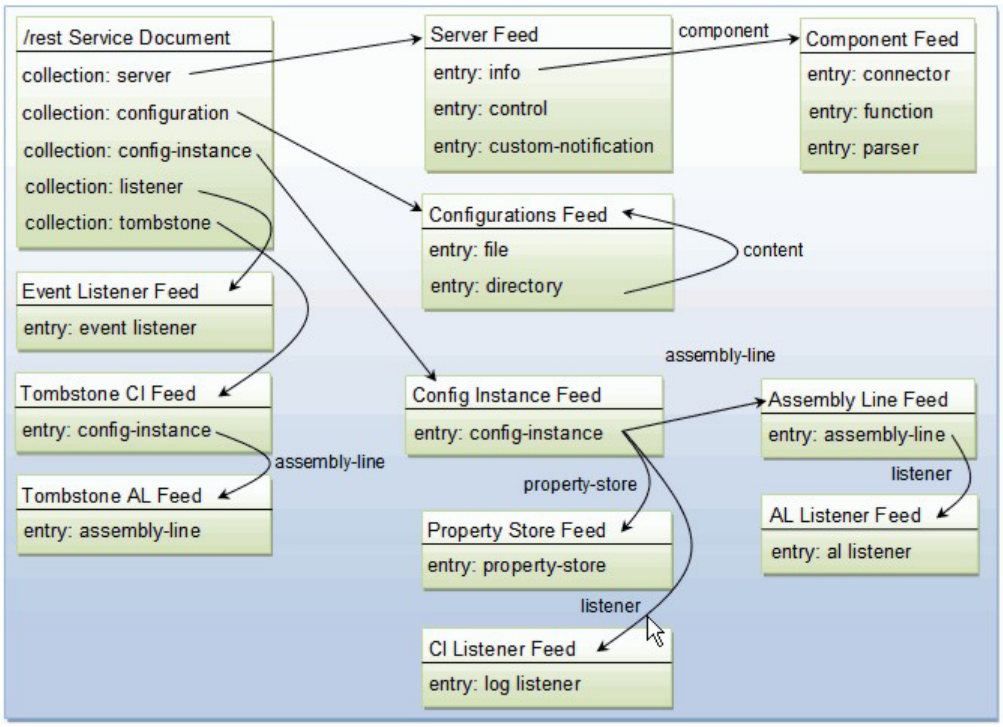
ここに記載されている情報を使用して、リソース階層をナビゲートできます。

REST サーバー API には `http://{host}:{port}/rest` URL で表される単一アクセス・ポイントがあります。各部の意味は次のとおりです。

- `host` - IBM Security Directory Integrator が実行されているシステムの名前または IP アドレスを指定します。
- `port` - 組み込み Web コンテナーが listen するポートを指定します。

API 内の残りのリソースは自由にトラバースできません。クライアント・アプリケーションは、Atom 項目内に Atom リンクとして用意されている URL を使用し、リソース階層をトラバースしてそれらの階層上で操作します。

以下の図は、使用可能なリソースと、クライアント・アプリケーションが階層のナビゲートの際に従う必要があるリンクを示したものです。



注: この図には重要な Atom リソースのみが示されています。

URL.method()[qs1, qs2] 表記は、照会パラメーターを使用して、指定された URL に対する HTTP メソッドの呼び出しを表すために使用されます。 URL はドットで区切られた一連の名前として表されます。パス内のそれぞれの名前は以下のいずれかのオプションを使用して、先行するすべての名前の解決後に、使用可能なリソースを表します。

- リソースのリストでは、サブリソースのカテゴリー (コレクション・リストを含むサービス・ドキュメント、Atom 項目リストを含む Atom フィードなど) を使用して、そのタイプが一意的に識別されます。例を示します。
  - サーバー・フィードを表すために、サービス・ドキュメントでカテゴリー「server」のコレクションが確認され、このサービス・ドキュメントは URL: {server} のように指定されます。
  - サーバー・フィードでカテゴリー「info」の Atom 項目を表すには、URL: {server}.{info} 表記を使用します。
- Atom 項目には他のリソースへのリンクが含まれています。それらのリソースを表すには、Atom リンクで「rel」属性の値が使用されます。例えば、コンポーネント・フィードを表すには、URL: {server}.{info}.component 表記を使用します。

{server}.{info}.component.{connector}.get() ステートメントは、コネクター項目のリソースを表す Atom 項目の URL に HTTP GET 要求が送信されることを示します。詳しくは、754 ページの『アルゴリズムの例』を参照してください。

アルゴリズムの例

標準的なブラウザー・アプリケーションがサーバーから特定のコネクター情報を取得する際に使用する、ここに記載されているアルゴリズムの例を使用することができます。

1. サーバー・フィード URL を取得します。
   a. HTTP GET 要求をエントリー・ポイント URL に送信します。
   b. Atom サービス・ドキュメントを分析して、以下を含む collection 要素を見つけます。
      - 「category」要素
      - 値が「server」の「term」属性
      - 値が「http://www.ibm.com/xmlns/prod/tdi/rest#resource」の「scheme」属性
   c. 「collection」要素の「href」属性の値を取得します。
2. サーバー情報項目 URL を取得します。
   a. HTTP GET 要求をサーバー・フィード URL に送信します。
   b. Atom フィードを分析して、以下を含む category 要素の Atom 要素を見つけます。
      - 値が「info」の「term」属性
      - 値が「http://www.ibm.com/xmlns/prod/tdi/rest#server」の「scheme」属性
   c. Atom 要素を分析して、値が「self」の「rel」属性を含む「link」要素を見つけます。
   d. 「link」要素の「href」属性の値を取得します。
3. コンポーネント・フィード URL を取得します。
   a. HTTP GET 要求をサーバー情報項目 URL に送信します。
   b. Atom 要素を分析して、値が「component」の「rel」属性を含む「link」要素を見つけます。
   c. 「link」要素の「href」属性の値を取得します。
4. 必要なコネクター項目を見つけます。
   a. コンポーネント・フィードの内容を、その URL に HTTP GET 要求を送信して取得します。
   b. 以下を含む「category」要素のすべての Atom 項目を見つけます。
      - 値が「connector」の「term」属性
      - 値が「http://www.ibm.com/xmlns/prod/tdi/rest#component」の「scheme」属性
   c. 実行可能ないずれかの方法で (例えば、「title」要素をフィルタリングして) 一連の Atom 項目から Atom 項目を選択します。
5. コネクター・ディスクリプター文書を取得します。
   a. 選択した Atom 項目を分析して、「content」要素を見つけます。
   b. 「content」要素に「src」属性が含まれている場合は、その属性の値で指定された URL に HTTP GET 要求を送信します。
   c. 含まれていない場合は、必要な文書が「content」要素に組み込まれます。

サーバー・フィード

サーバー・フィードを使用して、IBM Security Directory Integrator サーバーに関する情報および制御を提供することができます。

有効な操作については、このセクションで説明します。

操作	{server}.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「server」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」

サーバー情報項目

IBM Security Directory Integrator サーバーの詳細を提供するサーバー情報項目を使用することができます。

操作	{server}.{info}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「info」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#server」
Atom コンテンツ	application/com.ibm.di.api.server.info+json;type=serverInfo, application/com.ibm.di.api.server.info+xml

コンポーネント・フィード

コンポーネント・フィードでは、サーバーにインストールされる、コネクター、関数コンポーネント、パーサーなどの、すべての IBM Security Directory Integrator コンポーネントのリストが提供されます。

操作	{server}.{info}.component.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「component」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#server」

コネクター項目

操作	{server}.{info}.component.{connector}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「connector」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#component」
Atom コンテンツ	application/com.ibm.di.api.component+json;type=connectorDescriptor, application/com.ibm.di.api.component+xml

関数項目

操作	{server}.{info}.component.{function}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「function」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#component」
Atom コンテンツ	application/com.ibm.di.api.component+json;type=functionDescriptor, application/com.ibm.di.api.component+xml

パーサー項目

操作	{server}.{info}.component.{parser}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「parser」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#component」
Atom コンテンツ	application/com.ibm.di.api.component+json;type=parserDescriptor, application/com.ibm.di.api.component+xml

サーバー制御項目

IBM Security Directory Integrator サーバーの詳細を提供するサーバー制御項目を使用することができます。

操作	{server}.{control}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「control」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#server」

サーバーのシャットダウン

操作	{server}.{control}.shutdown.post()
応答コンテンツ・タイプ	application/com.ibm.di.api.server.control+json;type=shutdown, application/com.ibm.di.api.server.control+xml

カスタム通知項目

ここでカスタム通知項目を表示し、通知を送信できます。

操作	{server}.{custom-notification}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「custom-notification」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#server」

通知の送信

操作	server}.{custom-notification}.notify.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.server.notification+json;type=customNotification, application/com.ibm.di.api.server.notification+xml
詳細	`CustomNotification` タイプでは、「`data`」要素内でのペイロードの (「`type`」属性を使用する) 指定が可能です。以下の値が認識されます。 • `text/*` – ペイロードはストリング・オブジェクトとして解析されます。「`type`」属性がない場合は、この値がデフォルト値になります。 • `application/octet-stream` – Base64 エンコードのバイト配列。これは、デコードされて、そのまま送信されます。 • `application/octet-stream+object` – Base64 エンコードのバイト配列。これは、デコードされ、シリアライズ解除されてからオブジェクトとして送信されます。 正常応答には本文やロケーション・ヘッダーは含まれません。 HTTP コードは 204 (No Content) です。

構成フィード

構成フィードを通じて、デプロイ済みの IBM Security Directory Integrator 構成を含むサーバー API 構成ディレクトリーにアクセスできます。

REST サーバー API では、Atom 項目を使用して構成ファイルが表されますが、サブディレクトリーは (Atom フィードのコンテキスト内の) Atom 項目としても、(ディレクトリー・コンテンツのリスト時に) Atom フィードとしても表されます。

定義済み URL 構文で複数レベルの階層を表すには、以下の単一レベルの表記を使用します。

- {configuration}.{directory} – この構文は、階層内のディレクトリーを表す Atom 項目を意味します (サーバー API 構成ディレクトリーの直接の子である必要はありません)。
- {configuration}.{directory}.content – この構文は、階層内のディレクトリー・コンテンツを表す Atom フィードを意味します (サーバー API 構成ディレクトリーの直接の子である必要はありません)。
- {configuration}.{file} – この構文は、階層内の構成ファイルを表す Atom 項目を意味します (サーバー API 構成ディレクトリーの直接の子である必要はありません)。

操作	{configuration}.get() / {configuration}.{directory}.content.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「configuration」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	サーバー API 構成ディレクトリー内、または任意の子の構成ディレクトリー内の子項目のリストを取得します。

操作	{configuration}.post() / {configuration}.{directory}.content.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=createConfig, application/com.ibm.di.api.configuration+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「file」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#configuration」
詳細	指定された構成データで構成ファイルを作成します。 構成ファイルのパスは、HTTP POST 要求の送信先である、Atom フィードで表される基本ディレクトリーの相対パスです。このパスは、CreateConfig オブジェクトに指定される名前から取得されます。 成功時には、新しく作成された構成ファイルの Atom 項目を指すロケーション・ヘッダーとともに HTTP コード 201 が返されます。応答本体にはその Atom 項目のコピーが含まれます。

構成ディレクトリー項目

構成ディレクトリー項目としてここに記載されている情報を使用することができます。

操作	{configuration}.{directory}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml

Atom カテゴリー	term=「directory」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#configuration」
詳細	特定の構成ディレクトリーを表す Atom 項目を取得します。この項目には、ディレクトリー・コンテンツは表示されません。コンテンツにアクセスするには、「content」リンクを使用して、Atom フィード表記を取得します。

構成ファイル項目

構成ファイル項目としてここに記載されている情報を使用することができます。

操作	{configuration}.{file}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「file」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#configuration」
詳細	特定の構成ファイルを表す Atom 項目を取得します。

操作	{configuration}.{file}.delete()
詳細	IBM Security Directory Integrator サーバー上のデプロイ済み構成ファイルを削除します。

構成ファイルの編集

クライアント・アプリケーションでは、デプロイ済みの構成ファイルを編集することができます。単一の構成ファイルへの複数のクライアント・アクセスを調整するには、構成ファイルをロックする必要があります。REST サーバー API を使用すると、構成ファイルのロックを取得して (ファイルがまだロックされていないと仮定)、ファイルのロックを保持している間に複数の変更を送信し、他のクライアントで使用するためにロックを解除することができます。

操作	{configuration}.{file}.lock.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=configLock, application/com.ibm.di.api.configuration+xml
応答コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=configLock, application/com.ibm.di.api.configuration+xml
詳細	デプロイ済み構成ファイルのロックを要求します。configPassword はロックの作成時にのみ使用され、既にロックされている構成ファイルの更新中は無視されます。 成功時には、構成ファイルがロックされ、HTTP コード 201 (Created) が返されます。ロケーション・ヘッダーには構成ロック・リソースへのリンクが含まれ、本文にはロックの表記が含まれます。 構成が既にロックされている場合、要求は失敗し、HTTP コード 409 (Conflict) が返されます。構成ファイルは、以下の状態になるまでロックされたままになります。 • DELETE HTTP 操作を使用して、明示的にロックが解除された。 • サーバー API ロック・タイムアウトが IBM Security Directory Integrator サーバーで構成されており、そのタイムアウト時間が経過した。

操作	{configuration}.{file}.lock.get()

応答コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=configLock, application/com.ibm.di.api.configuration+xml
詳細	成功時には、応答本体にロック・オブジェクトの表記が含まれます。以下の HTTP コードと共にエラーが返されます。 • 404 (Not found) – 別の管理者ユーザー、またはサーバー API の自動ロック解除機能により、明示的にロックが解除され、それ以降は取得されていません。 • 403 (Forbidden) – ロックは別のユーザーによって以前に取得されています。

操作	{configuration}.{file}.lock.put()
要求コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=configLock, application/com.ibm.di.api.configuration+xml
応答コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=configLock, application/com.ibm.di.api.configuration+xml
詳細	構成を保護するロックを更新します。この要求ではロックは解除されません。成功時には、応答本体に更新済みロック・オブジェクトの表記が含まれます。以下の HTTP コードと共にエラーが返されます。 • 404 (Not found) – 別の管理者ユーザー、またはサーバー API の自動ロック解除機能により、明示的にロックが解除され、それ以降は取得されていません。 • 403 (Forbidden) – ロックは別のユーザーによって以前に取得されています。

操作	{configuration}.{file}.lock.delete()
詳細	ロックを解除します。 別の管理者ユーザー、またはサーバー API の自動ロック解除機能により、明示的にロックが解除され、それ以降はロックが取得されていない場合、HTTP コード 404 (Not found) と共にエラーが返されます。

ConfigInstance フィード

構成フィードを使用して、IBM Security Directory Integrator サーバーで開始されたサーバー API ConfigInstance オブジェクトにアクセスできます。

操作	{config-instance}.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「config-instance」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	ConfigInstance オブジェクトを Atom 項目として取得します。

操作	{config-instance}.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.configuration+json;type=startCI, application/com.ibm.di.api.configuration+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「config-instance」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」

詳細	サーバーで ConfigInstance を開始します。StartCI 項目を使用すれば、次の 2 つの方法で構成を指定することができます。 • configRef – 構成ファイルの Atom 項目 ID を指定します。 • solution – 構成ファイルを使用せずに構成データを指定します。 成功時には、新しく作成された ConfigInstance Atom 項目を指すロケーション・ヘッダーとともに HTTP コード 201 (Created) が返されます。応答本体にはその Atom 項目のコピーが含まれます。

ConfigInstance 項目

操作	{config-instance}.{config-instance}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「config-instance」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の ConfigInstance を表す Atom 項目を取得します。 この ConfigInstance のロード元の実際の構成ファイルを指すリンクが提供されます。この ConfigInstance が一時的である場合、リンクは提供されません。

操作	{config-instance}.{config-instance}.delete()
詳細	ConfigInstance を同期的に停止します。

ConfigInstance 構成

操作	{config-instance}.{config-instance}.config.get()
応答コンテンツ・タイプ	application/com.ibm.di.configuration+json;type=solution, application/com.ibm.di.configuration+xml
詳細	ConfigInstance の開始時に使用した構成データを取得します。

AssemblyLine フィード

操作	{config-instance}.{config-instance}.assembly-line.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「assembly-line」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の ConfigInstance オブジェクトの AssemblyLine フィードを取得します。

操作	{config-instance}.{config-instance}.assembly-line.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.assembly-line+json;type=startAL, application/com.ibm.di.api.assembly-line+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「assembly-line」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」

詳細	サーバーで AssemblyLine を開始します。startAL オブジェクトには、ConfigInstance 構成から取得された AssemblyLine の名前が含まれます。 成功時には、新しく作成された AssemblyLine Atom 項目を指すロケーション・ヘッダーとともに HTTP コード 201 (Created) が返されます。応答本体にはその Atom 項目のコピーが含まれます。

AssemblyLine 項目

操作	config-instance}.{config-instance}.assembly-line.{assembly-line}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「assembly-line」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の AssemblyLine を表す Atom 項目を取得します。 スキーム「`http://www.ibm.com/xmlns/prod/tdi/rest#assembly-line`」の用語「`active`」を含むカテゴリーは、AssemblyLine がまだアクティブであることを示します。 スキーム「`http://www.ibm.com/xmlns/prod/tdi/rest#assembly-line`」の用語「`manual`」を含むカテゴリーは、AssemblyLine が手動モードで開始されたことを示します。関係が `handle` のリンクも使用可能です。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.delete()
詳細	AssemblyLine を同期的に停止します。

AssemblyLine 構成

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.config.get()
応答コンテンツ・タイプ	application/com.ibm.di.configuration+json;type=assemblyLine, application/com.ibm.di.configuration+xml
Atom コンテンツ	AssemblyLine の開始時に使用した構成データを取得します。

AssemblyLine ログ

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.log.get()
応答コンテンツ・タイプ	text/plain
詳細	AssemblyLine のログを取得します。

AssemblyLine 状況

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.status.get()
応答コンテンツ・タイプ	application/com.ibm.di.api.assembly-line+json;type=taskStatistics, application/com.ibm.di.api.assembly-line+xml
Atom カテゴリー	AssemblyLine の状況を取得します。

AssemblyLine 結果項目

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.result.get()
応答コンテンツ・タイプ	application/com.ibm.di.api.entry+json;type=entry, application/com.ibm.di.api.entry+xml
詳細	AssemblyLine 結果項目を取得します。

手動による AssemblyLine の処理

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.handle.get()
応答コンテンツ・タイプ	application/com.ibm.di.api.assembly-line+json;type=alHandle, application/com.ibm.di.api.assembly-line+xml
詳細	state 属性と「resultEntry」要素 (使用可能な場合) のみを含む「ALHandle」オブジェクトを取得します。このオブジェクトは、AssemblyLine を手動モードで制御する際に使用されます。 「state」属性に指定できる値は次のとおりです。 • init – AssemblyLine が作成されます。サイクルはまだ処理されていません。 • processing – AssemblyLine は (PUT を使用して) サイクルを実行するように要求されましたが、完了しませんでした。 • done – AssemblyLine は、完了したサイクルを実行するように要求されました。 • closed – AssemblyLine ハンドルは既に終了しています。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.handle.put()
要求コンテンツ・タイプ	application/com.ibm.di.api.assembly-line+json;type=alHandle, application/com.ibm.di.api.assembly-line+xml
応答コンテンツ・タイプ	application/com.ibm.di.api.assembly-line+json;type=alHandle, application/com.ibm.di.api.assembly-line+xml
詳細	新しいサイクルを実行するように要求します。 手動による AssemblyLines では、サイクルの実行に長い時間がかかります。API は、クライアント呼び出しで返すスレッドを作成します。 **注:** この特定の操作は、HTTP 仕様で定義されている「べき等」ではありません。このメソッドの結果は必ずしも同じとは限りません。正常応答には、HTTP コード 200 と、ALHandle オブジェクトのスナップショットが含まれます。ALHanlde オブジェクトには、AssemblyLine サイクルの実行結果が含まれます (呼び出しで返される前に完了した場合)。実行が完了していない場合は、ALHandling 状態が processing に設定されます。 別のサイクルがまだ実行されている間にこの操作を実行すると、HTTP コード 409 (Conflict) が返され、要求は無視されます。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.handle.delete()
詳細	ALHandle オブジェクトをクローズします。 **注:** ハンドルのクローズは必須です。ハンドルは、Assembly Line の Atom 項目の明示的な削除の際にもクローズされます。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.listener.post()

要求コンテンツ・タイプ	application/com.ibm.di.api.listener+json;type=assemblyLineListener, application/com.ibm.di.api.listener+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「al」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
詳細	AssemblyLine リスナーを登録して、通知を受信します。

AssemblyLine コンテキストのスクリプト評価

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.script.post()
要求コンテンツ・タイプ	text/plain
応答コンテンツ・タイプ	application/com.ibm.di.api.entry+json;type=entry, application/com.ibm.di.api.entry+xml
詳細	AssemblyLine のコンテキストで JavaScript を実行します。実行したスクリプトで値が返される場合、応答コードは 200 (OK) になります。値が返されない場合は、204 (No Content) コードが返されます。 スクリプトで返された値が IBM Security Directory Integrator 項目の場合は、そのまま返されます。それ以外の場合は、新しい IBM Security Directory Integrator 項目が作成され、値は「value」属性に配置されます。

AssemblyLine リスナー・フィード

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.listener.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「listener」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の AssemblyLine オブジェクトのリスナー・フィードを取得します。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.listener.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.listener+json;type=assemblyLineListener, application/com.ibm.di.api.listener+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「al」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
詳細	AssemblyLine リスナーを登録して、通知を受信します。

AssemblyLine リスナー項目

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.listener.{al}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「al」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
Atom コンテンツ	application/com.ibm.di.api.listener+json;type=assemblyLineListener, application/com.ibm.di.api.listener+xml
詳細	特定の AssemblyLine リスナーを表す Atom 項目を取得します。

操作	{config-instance}.{config-instance}.assembly-line.{assembly-line}.listener.{al}.delete()
詳細	AssemblyLine リスナー・オブジェクトを登録抹消します。

PropertyStore フィード

操作	{config-instance}.{config-instance}.property-store.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「property-store」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の ConfigInstance オブジェクトの PropertyStore フィードを取得します。

操作	{config-instance}.{config-instance}.property-store.post()
要求コンテンツ・タイプ	application/com.ibm.di.configuration+json;type=propertyStore, application/com.ibm.di.configuration+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「property-store」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の ConfigInstance の PropertyStore を作成します。 成功時には、新しく作成された PropertyStore Atom 項目を指すロケーション・ヘッダーとともに HTTP コード 201 (Created) が返されます。応答本体にはその Atom 項目のコピーが含まれます。

PropertyStore 項目

操作	{config-instance}.{config-instance}.property-store.{property-store}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「property-store」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	• 特定の PropertyStore を表す Atom 項目を取得します。 • スキーム「`http://www.ibm.com/xmlns/prod/tdi/rest#property-store`」の用語「`default`」を含むカテゴリーは、PropertyStore がデフォルト項目であることを示します。 • スキーム「`http://www.ibm.com/xmlns/prod/tdi/rest#property-store`」の用語「`password`」を含むカテゴリーは、PropertyStore がパスワードであることを示します。 • スキーム「`http://www.ibm.com/xmlns/prod/tdi/rest#property-store`」の用語「`modified`」を含むカテゴリーは、PropertyStore は更新されたがコミットされていないことを示します。

操作	{config-instance}.{config-instance}.property-store.{property-store}.put()
要求コンテンツ・タイプ	application/json;type=entry, application/atom+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「property-store」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」

詳細	• PropertyStore のタイプを更新します。 • スキーム「http://www.ibm.com/xmlns/prod/tdi/rest#property-store」の用語「default」を含むカテゴリーが要求の Atom 項目にある場合、PropertyStore にはデフォルト項目としてマークが付けられます。 • スキーム「http://www.ibm.com/xmlns/prod/tdi/rest#property-store」の用語「password」を含むカテゴリーが要求の Atom 項目にある場合、PropertyStore にはパスワードとしてマークが付けられます。 **注:** オプションが設定されていない場合、PropertyStore フラグの設定は解除されません。PropertyStore に対する default フラグの設定を解除するには、別の PropertyStore にフラグを設定します。

操作	{config-instance}.{config-instance}.property-store.{property-store}.delete()
詳細	PropertyStore を削除します。

PropertyStore プロパティー

操作	{config-instance}.{config-instance}.property-store.{property-store}.properties.get()
応答コンテンツ・タイプ	application/com.ibm.di.api.property-store+json;type=properties, application/com.ibm.di.api.property-store+xml
詳細	PropertyStore のすべてのプロパティーを取得します。

操作	{config-instance}.{config-instance}.property-store.{property-store}.properties.put()
要求コンテンツ・タイプ	application/com.ibm.di.api.property-store+json;type=properties, application/com.ibm.di.api.property-store+xml
詳細	1 つまたは複数のプロパティーの値を設定して、PropertyStore のインクリメンタル更新を要求します。 **注:** プロパティーは明示的に削除されます。プロパティーを削除するために、要求ではプロパティーとその名前を指定する必要があります (値は指定しません)。API は、要求リストに存在しないプロパティーは削除しません。成功時には、空の応答本体とともに HTTP コード 204 (No Content) が返されます。

単一のプロパティー

操作	{config-instance}.{config-instance}.property-store.{property-store}.properties.get()[name]
応答コンテンツ・タイプ	application/com.ibm.di.api.property-store+json;type=property, application/com.ibm.di.api.property-store+xml
詳細	「name」照会パラメーターで指定されているプロパティーを取得します。名前は指定されているが、プロパティーが見つからない場合は、HTTP コード 404 が返されます。

操作	{config-instance}.{config-instance}.property-store.{property-store}.properties.put()[name, encrypt, commit]
要求コンテンツ・タイプ	text/plain

詳細	• 「name」照会パラメーターで指定されているプロパティーの値を設定します。該当するプロパティーが見つからない場合は、新しいプロパティーが作成されます。 • 「encrypt」パラメーターにブール値を指定して、プロパティー値の暗号化を切り替えます。デフォルト値は「false」です。 • 「commit」– 保留中の変更をすべてこの要求でコミットする必要があるかどうかを指定します。デフォルト値は 『false』 です。名前が指定されていない場合は、HTTP コード 404 が返されます。 • 成功時には、空の応答本体と共に HTTP コード 204 (No Content) が返されます。

操作	{config-instance}.{config-instance}.property-store.{property-store}.properties.delete()[name, commit]
詳細	• name 照会パラメーターで指定されているプロパティーを削除します。 • 「commit」– 保留中の変更をすべてこの要求でコミットする必要があるかどうかを指定します。デフォルト値は false です。 • 名前が指定されていない場合は、HTTP コード 404 が返されます。 • 成功時には、空の応答本体と共に HTTP コード 204 (No Content) が返されます。

ConfigInstance リスナー・フィード

操作	{config-instance}.{config-instance}.listener.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「listener」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	特定の ConfigInstance オブジェクトのリスナー・フィードを取得します。

操作	{config-instance}.{config-instance}.listener.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.listener+json;type=assemblyLineListener, application/com.ibm.di.api.listener+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「log」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
詳細	ConfigInstance リスナーを登録して、通知を受信します。

ConfigInstance リスナー項目

操作	{config-instance}.{config-instance}.listener.{log}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「log」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
Atom コンテンツ	application/com.ibm.di.api.listener+json;type=logListener, application/com.ibm.di.api.listener+xml
詳細	特定の ConfigInstance リスナーを表す Atom 項目を取得します。

操作	{config-instance}.{config-instance}.listener.{log}.delete()
詳細	ConfigInstance リスナー・オブジェクトを登録抹消します。

サーバー・リスナー・フィード

ここに記載されている情報を使用して、サーバー・リスナー・フィードを操作できます。

操作	{listener}.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「listener」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	サーバー通知のリスナー・フィードを取得します。

操作	{listener}.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.listener+json;type=diEventListener, application/com.ibm.di.api.listener+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「event」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
詳細	リスナーを登録して、サーバー通知を受信します。

操作	{listener}.post()
要求コンテンツ・タイプ	application/com.ibm.di.api.listener+json;type=configFileListener, application/com.ibm.di.api.listener+xml
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「config-file」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
詳細	リスナーを登録して、構成ファイルの変更通知を受信します。

サーバー・リスナー項目

操作	{listener}.{event}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「event」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
Atom コンテンツ	application/com.ibm.di.api.listener+json;type=diEventListener, application/com.ibm.di.api.listener+xml
詳細	特定のサーバー・リスナーを表す Atom 項目を取得します。

操作	{listener}.{config-file}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「event」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#listener」
Atom コンテンツ	application/com.ibm.di.api.listener+json;type=configFileListener, application/com.ibm.di.api.listener+xml
詳細	特定の構成ファイル・リスナーを表す Atom 項目を取得します。

操作	{listener}.{event}.delete() or {listener}.{config-file}.delete()
詳細	サーバー/構成ファイル・リスナー・オブジェクトを登録抹消します。

トゥームストーン・フィード

ここに記載されている情報を使用して、トゥームストーン・フィードを操作できます。

操作	{tombstone}.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「tombstone」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
Atom コンテンツ	トゥームストーンが記録されている ConfigInstance Atom 項目、またはトゥームストーンが記録されている子の AssemblyLine がある ConfigInstance Atom 項目がすべて表示されているトゥームストーン・フィード・リストを取得します。

ConnfigInstance 項目

操作	{tombstone}.{config-instance}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「config-instance」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	ConfigInstance Atom 項目に対して記録されているトゥームストーンと、子の AssemblyLine トゥームストーンの両方にアクセスするための ConfigInstance Atom 項目を取得します。

操作	{tombstone}.{config-instance}.delete()
詳細	選択された ConfigInstance のすべてのトゥームストーンと、子の AssemblyLine のすべてのトゥームストーンを削除します。

ConfigInstance トゥームストーン・フィード

操作	{tombstone}.{config-instance}.tombstone.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「tombstone」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	対応する ConfigInstance のトゥームストーン Atom フィードを取得します。詳細には、特定の ConfigInstance のトゥームストーンのリストが含まれます。

ConnfigInstance トゥームストーン項目

操作	{tombstone}.{config-instance}.tombstone.{tombstone}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「tombstone」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
Atom コンテンツ	application/com.ibm.di.api.tombstone+json;type=tombstone, application/com.ibm.di.api.tombstone+xml
詳細	特定の ConfigInstance のトゥームストーン Atom 項目を取得します。

操作	{tombstone}.{config-instance}.tombstone.{tombstone}.delete()
詳細	特定の ConfigInstance の選択したトゥームストーンを削除します。

AssemblyLine フィード

操作	{tombstone}.{config-instance}.assembly-line.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「assembly-line」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	トゥームストーンが記録されたすべての AssemblyLine Atom 項目のトゥームストーン・フィード・リストを取得します。

AssemblyLine 項目

操作	{tombstone}.{config-instance}.assembly-line.{assembly-line}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「assembly-line」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	記録されたトゥームストーンにアクセスするための AssemblyLine Atom 項目を取得します。

操作	{tombstone}.{config-instance}.assembly-line.{assembly-line}.delete()
詳細	選択した AssemblyLine のすべてのトゥームストーンを削除します。

AssemblyLine トゥームストーン・フィード

操作	{tombstone}.{config-instance}.assembly-line.{assembly-line}.tombstone.get()
応答コンテンツ・タイプ	application/json;type=feed, application/atom+xml
Atom カテゴリー	term=「tombstone」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
詳細	対応する AssemblyLine のトゥームストーン Atom フィードを取得します。この詳細には、特定の AssemblyLine のトゥームストーンのリストが含まれます。

AssemblyLine トゥームストーン項目

操作	{tombstone}.{config-instance}.assembly-line.{assembly-line}.tombstone.{tombstone}.get()
応答コンテンツ・タイプ	application/json;type=entry, application/atom+xml
Atom カテゴリー	term=「tombstone」, scheme=「http://www.ibm.com/xmlns/prod/tdi/rest#resource」
Atom コンテンツ	application/com.ibm.di.api.tombstone+json;type=tombstone, application/com.ibm.di.api.tombstone+xml
詳細	特定の AssemblyLine のトゥームストーン Atom 項目を取得します。

操作	{tombstone}.{config-instance}.assembly-line.{assembly-line}.tombstone.{tombstone}.delete()
詳細	特定の AssemblyLine の選択したトゥームストーンを削除します。

リスナー・トランスポート・チャネル

ここに記載されている情報を使用することで、リスナー・チャネルおよびトランスポート・メカニズムのタイプについて知ることができます。

リスナーにはそれぞれメッセージのトランスポート方法を表すオブジェクトが含まれます。トランスポート・メカニズムのタイプは次のとおりです。

- プッシュ・チャネル
- ポーリング・チャネル

プッシュ・チャネル

プッシュ・チャネルでは HTTP プロトコルを使用してメッセージを配信します。メッセージを配信するために、登録済みリスナー内のプッシュ・チャネル・オブジェクトには、メッセージの配信先である宛先 URL を指定するフィールドが含まれています。クライアントは HTTP サーバーを始動して、その HTTP サーバー上の場所を指す URL で構成されたプッシュ・チャネルを持つリスナーを登録します。メッセージごとに、構成済みリスナーは指定されたロケーションに HTTP POST 要求を送信します。

メイン・サーバーとの通信が何らかのエラーにより停止した場合、リスナーはフォールバック・サーバー (提供されている場合) への通知を試行します。プッシュ・チャネルを使用すれば、このような状態が発生したときにリアルタイム通知を簡単に配信できます。

プッシュ・チャネルを持つリスナーには、用語「push」およびスキーム「http://www.ibm.com/xmlns/prod/tdi/rest#listener」を含む Atom カテゴリーもあります。

ポーリング・チャネル

ポーリング・チャネル・メカニズムは、通知をほぼすぐにリアルタイム配信する場合に使用されます。

このメカニズムは、クライアント・サイドが要求するまでメッセージをバッファーに入れる JMS サーバーの使用に依存します。クライアントは、ポーリング構成を持つ新規リスナーを登録します。リスナーが正常に登録されると、クライアントは関係が「poll」の Atom リンクを受信します。その URL に対する HTTP GET 要求には各メッセージが含まれます。

クライアントは、メッセージをまとめて配信するようにプッシュ・チャネルを構成する場合に、JMS サーバーを使用してメッセージをバッファーに入れることができます。このメカニズムにより、ネットワーク・トラフィックが削減され、イベント・ソースが短時間で多くのイベントを生成する場合のクライアント/サーバーの処理も単純化されます。小数のイベントを生成する他のイベント・ソースの場合は、単一イベントを返すか、あるいはタイムアウト値を増やして複数のイベントを少量のまとまりで返すようにそれらを構成します。プッシュ・チャネルを持つリスナーには、用語「poll」およびスキーム「http://www.ibm.com/xmlns/prod/tdi/rest#listener」を含む Atom カテゴリーもあります。

スキーマ

REST サーバー API では、デフォルトのメッセージ形式として JSON 構文を使用します。ここに記載されている HTTP 要求ヘッダーを指定して、XML を使用することができます。

- コンテンツ・タイプ – サーバーに送信した本文のメディア・タイプを示します。
- 同意 – クライアント・アプリケーションがサポートするメディア・タイプを指定します。

応答の場合、REST サーバー API には、返される本文 (存在する場合) のタイプを指定するためのコンテンツ・タイプ HTTP ヘッダーが組み込まれます。タイプがコンテンツ・タイプ・ヘッダーに明示的に指定されていない場合は、本文のエンコードに UTF-8 が使用されます。

内容定義

REST API で使用される (末尾が +xml の) 各カスタム XML メディア・タイプを、1 つ以上の XML スキーマ文書を使用して定義できます。

対応する (末尾が +json の) JSON メディア・タイプでは、XML に定義されている同じスキーマを再使用します。以下の表で、各メディア・タイプに対応する XSD 文書について説明します。

注: デフォルトでは、REST サーバー API のスキーマはすべて http://<host>:<port>/schema/ にあります。この表では基本のロケーション名のみが使用されています。

メディア・タイプ	内容	スキーマ
application/json application/atom+xml	Atom Synd オブジェクト (項目、フィード) を定義します。	http://tools.ietf.org/html/rfc4287
application/atomsvc+json application/atomsvc+xml	Atom Synd オブジェクト (サービス) を定義します。	http://tools.ietf.org/html/rfc4287
application/com.ibm.di.configuration+json application/com.ibm.di.configuration+xml	サーバー構成オブジェクトを定義します。	config/solution.xsd
application/com.ibm.di.api.server.info+json application/com.ibm.di.api.server.info+xml	サーバー API 情報オブジェクトを定義します。	api/server-info.xsd
application/ com.ibm.di.api.server.control+json application/com.ibm.di.api.server.control+xml	サーバー API 制御オブジェクトを定義します。	api/server-control.xsd
application/ com.ibm.di.api.server.notification+json application/ com.ibm.di.api.server.notification+xml	サーバー API 通知オブジェクトを定義します。	api/notification.xsd
application/com.ibm.di.api.component+json application/com.ibm.di.api.component+xml	サーバー API コンポーネント・オブジェクトを定義します。	api/component.xsd
application/com.ibm.di.api.configuration+json application/com.ibm.di.api.configuration+xml	構成を制御するサーバー API オブジェクトを定義します。	api/configuration.xsd

メディア・タイプ	内容	スキーマ
application/com.ibm.di.api.assembly-line+json application/com.ibm.di.api.assembly-line+xml	サーバー API AssemblyLine オブジェクトを定義します。	api/assembly-line.xsd
application/com.ibm.di.api.property-store+json application/com.ibm.di.api.property-store+xml	サーバー API PropertyStore オブジェクトを定義します。	api/property-store.xsd
application/com.ibm.di.api.entry+json application/com.ibm.di.api.entry+xml	サーバー API 項目オブジェクトを定義します。	api/entry.xsd
application/com.ibm.di.api.listener+json application/com.ibm.di.api.listener+xml	サーバー API リスナー・オブジェクトを定義します。	api/listener.xsd

JSON を使用するポリモアフィズム

基本 XML タイプのみ使用して XML タイプの階層を定義する際に、このスキーマを使用できます。

XML スキーマでは、相互に継承する XML タイプを定義します。 JSON では、オブジェクトによって、タイプに関する追加情報が伝達されません。REST サーバー API には、JSON オブジェクトのタイプを指定するプロパティー「@type」が必要です。

ConfigInstance を開始するための XML コンテンツ

```
<startCI xmlns="http://www.ibm.com/xmlns/prod/tdi/72/api"
configRef="http://localhost:1098/rest/config/e%3AReadFile"
 keepAlive="true"
 password="myConfigPasswd"
 runName="ReadFile_1">
 <logListner>
  <pollChannel waitTimeout="60" batchCap="1" />
 </logListener>
</startCI>
```

この例では、新しい runName で開始し、完了時もアクティブのままにしておく既存構成を要求しています。また、ConfigInstance の開始時に LogListener を添付するよう要求しています。リスナーは、クライアントによって要求されるまで、サーバー・サイドでメッセージをバッファーに入れておくポーリング・チャネルを使用します。実際の日付は、以下の例に示されているように、JSON 形式で表されます。

```
{
"configRef" : "http://localhost:1098/rest/config/e%3AReadFile",
"keepAlive" : true,
"password" : "myConfigPasswd",
"runName" : "ReadFile_1",
 "logListner" : {
  "@type" : "logListener",
  "channel" : {
   "@type" : "pollChannel",
"waitTimeout" : 60,
"batchCap" : 1
}
}
}
```

構文は api/configuration.xsd に定義したものと同じです。ただし、「@type」プロパティーが 2 つ追加されており、これらは XSD 文書で指定されていません。

「@type" : "logListener」プロパティーは、オブジェクトが、グローバル XML 要素名「"logListener"」と同じタイプであることを示しています。プロパティー名はプロパティーの基本タイプ (「LogListener」) を示しています。

次の「@type」プロパティーでは、「channel」プロパティーがローカル要素名であり、タイプが基本タイプ (抽象「TransportChannel」) を指していることがわかります。対応する XSD に定義されているグローバル XML 要素名 (「pollChannel」) を使用して、使用するタイプを指定してください。

ルート・オブジェクトには「@type」プロパティーはありません。ルート・オブジェクトのタイプは、開始されている操作で推測されます。ただし、送信するオブジェクトのタイプを指定する場合は、HTTP 要求ヘッダー「Content-Type」を使用できます。ヘッダーの値は「application/
com.ibm.di.api.configuration+json;type=startCI」です。タイプには、対応する XSD 文書に定義されているグローバル要素名が使用されます。

外部システム構成

ポーリング・チャネルをトランスポート・リスナー・イベント用のメカニズムとして使用するために、IBM Security Directory Integrator サーバーを JMS サーバーに対して構成することができます。

REST サーバーでは、IBM Security Directory Integrator JMSDriver アーキテクチャーを再使用して、リモート JMS サーバーへの接続を取得します。以下のプロパティーを設定する必要があります。

- api.rest.jmsdriver.name – JMSDriver のクラス名を指定します。例を示します。

  ```
  api.rest.jmsdriver.name=com.ibm.di.systemqueue.driver.ActiveMQ
  ```

- api.rest.jmsdriver.auth.username – リモート JMS サーバーへの接続を確立する際に使用するユーザー名を指定します。
- api.rest.jmsdriver.auth.password – リモート JMS サーバーへの接続を確立する際に使用するパスワードを指定します。

デフォルトでは、REST サーバー API は、IBM Security Directory Integrator にバンドルされている JMS サーバーを使用するように構成されます。

付録 F. アダプターを使用した新規コンポーネントの作成

IBM Security Directory Integrator においては、アダプターの概念は、開発者が AssemblyLine (AL) の方法論によってカスタム・コネクターを新たに作成できるようにするメカニズムとして使用することができます。

別の方法として、Java または JavaScript で開発する場合もあります。アダプターは、他の IBM Security Directory Integrator 開発者に簡単に配布することができ、従来の組み込み標準コネクターと同じように簡単に使用することができます。

アダプターによって、開発者は、潜在的に複雑なビジネス・ロジックを含むカスタム・コネクターや IBM Security Directory Integrator 開発コミュニティーに提供するカスタム操作を作成する場合に、IBM Security Directory Integrator のすべての機能を使用することができます。

組み合わせてアダプターに使用できる多数の新機能があります。これらの新機能について、以下のセクションで説明します。これらの機能はすべて、アダプターの概念に使用する以外にも役立つので、各機能について他の正式な資料を読むことをお勧めします。

IBM Security Directory Integrator アダプターをインプリメントして使用するためのアクティビティー・フローの概要を以下に示します。

1. Anne は、サポートされているコネクター・モード (イテレーターや削除など) および必要に応じたカスタム・モードをインプリメントするアダプター AssemblyLine (カスタム開発 ERP システムなど) を開発しています。
2. Anne は、別の IBM Security Directory Integrator 開発者である Pete にスタンドアロン・ファイルとして配布できるパッケージにその AL をパブリッシュします。
3. Pete はこのパッケージを自分の IBM Security Directory Integrator 開発環境にコピーします。Pete が定期的に開発している IBM Security Directory Integrator ソリューションで再利用したいと考えている他のコンポーネントを加えた、リソース/ライブラリー・モデルがこの場合は最適です。
4. Pete は、17 ページの『AssemblyLine コネクター』を使用してアダプターを呼び出すことで、Anne のアダプターを自分の AL で他の IBM Security Directory Integrator コネクターと同様に使用することができます。
5. Anne は、上記のステップを実行することで自分のアダプターを改良し、新しいバージョンをパブリッシュすることができます。

以下の図は、左側の Pete の AL が、Anne のアダプターを標準のコネクター・ルックアップ ・モードとカスタムの *Disable_acct* モードの両方で使用しているところを示しています。この図では、アダプターは Pete の AL の 2 つの場所で使用されていますが、バックエンド・ターゲット・システムへの影響を削減するために、アダプターの 1 つのインスタンスのみが開始されています。通常のコネクターと同様、アダプターは AL 内で共有し、プールすることができ、さらには AL 間で共有することもできます。

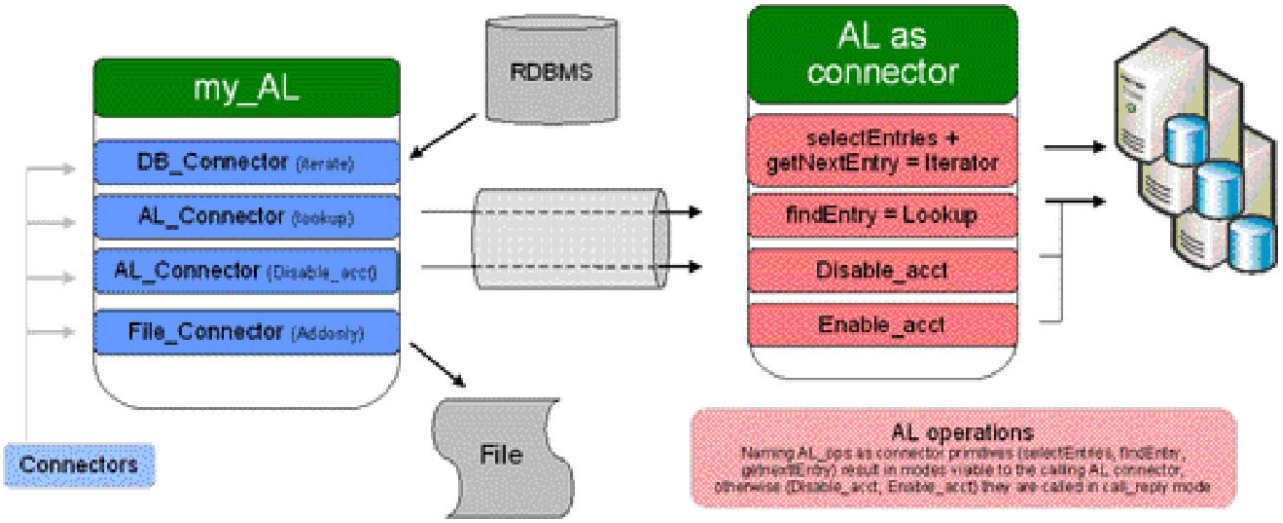


図 13. アダプターの使用法の概要

IBM Security Directory Integrator アダプターのインプリメンテーションを可能にする機能

アダプターを作成し使用するために利用できる機能について、もう少し詳しく見ていきます。これらの機能の大半は、アダプターの概念に特化して開発されているわけではないため、アダプター以外に使用される多くのユース・ケースがあります。

AL の操作

1 つの AL 内で、任意の数の *操作* を定義することができます。

これらの操作は、7.0 以前の呼び出し/戻りスキーマの AL (現在は「デフォルト」操作と呼ばれています) と似ていますが、任意の数の操作を作成できるようになりました。AL が API、スクリプト、AL FC、または AL コネクターから呼び出されて実行される場合、操作はその操作の必須属性を設定して指定するようになりました。ランタイム時に、AL は操作が呼び出されたことを認識します。これを照会し、AL 内のフローを適切に調整することができます。

注: AssemblyLine の操作を定義する場合は、「デフォルト」操作についても定義する場合を除き、明示的操作を指定する AssemblyLine を呼び出す必要があります。そうしないと、AssemblyLine によって例外がスローされます。

これらの操作によって、AL コネクターのエントリー・ポイントが提供され (以下の 779 ページの『AssemblyLine でのアダプターの使用』セクションで説明しています)、アダプターをコネクターとして表示および処理することができます。エントリー・ポイントは、コネクターを Java または JavaScript で開発する場合と同じであり、 785 ページの『付録 G. Java での独自コンポーネントのインプリメント』、および 780 ページの『アダプター操作のコネクター・モードへのマッピング』セクションのテーブルで説明しています。例えば、アダプター開発者が「ルックアップ」モードのみをインプリメントする場合は、「findEntry」操作のみをインプリメント

する必要があります。詳しくは、以下の 779 ページの『IBM Security Directory Integrator アダプターでの操作の使用』セクションを参照してください。

スイッチ/ケース・コンポーネント

Switch/Case コンポーネントを使用すると、定義済みのすべての操作に対するコードを簡単かつ確実にインプリメントできます。

Switch コンポーネントは、従来の開発言語のスイッチ構造に似ている AL コンポーネントです。基本的に、これは If-ElseIf-ElseIf コンポーネントのバリアントです。Switch コンポーネント内では、多数の Case が定義されています。Case には、Switch ステートメントがケースの値に一致した場合に実行される AL コンポーネントが含まれています。Switch コンポーネントの利点の 1 つは、定義されている AL 操作に基づいて case ステートメントを自動的に取り込むことができる点です。

コネクターの柔軟な初期化

ここに記載されている情報を使用することで、コネクターの柔軟な初期化を実行できるようになります。

古いバージョンの IBM Security Directory Integrator では、AL のすべてのコネクターが、AL の初期化段階で初期化されていました。コネクターの動的構成では、一般に、接続の終了、接続パラメーターの変更、および接続の再確立をスクリプト全体で行う必要がありました。代替方法は、コネクターの設定および使用のみをスクリプトで行うことでした。

現在のバージョンの IBM Security Directory Integrator では、オプションで以下のようにコネクターを初期化できます。

オンデマンドで
: 親しみを込めて、「遅延した」初期化と呼ばれることもあります。接続の確立は、AL の初期化時ではなく、AL の実行段階 (フロー) で制御が初めてコネクターに実際に渡されるときに行われます。これは、複合 AL が、コネクターが実際に使用されるときにのみコネクターを初期化することを意味します。

常に
: コネクターは、制御が渡されるたびに初期化されます。この機能は、接続パラメーター (ファイル名や LDAP 信任状など) が、アダプターの呼び出しごとに渡される情報の一部である場合に役立ちます。この機能が役立つ別の利点として、プールされたコネクターを使用する場合、「常に」初期化することで、ランタイム時にプールからコネクター・インスタンスが取得され、使用後に解放されることが挙げられます。基本的に、これにより、AL 全体の「共有/再使用」コネクター・プールがインプリメントされます。

構成の変更時
: 接続は、構成パラメーター (またはパラメーター置換の評価) がコネクターの前回の初期化以降に変更された場合に、再度初期化されます。パラメーター置換機能によって、前よりも大幅に簡単にコネクターを動的に構成できるようになりました。これにより、コネクターの接続パラメーターの変更 (および再接続の強制適用) を AL の内部と外部の両方で簡単に実行できるよう

になりました。例えば、別の AL、またはプロパティーのコマンド行変更によって、ターゲットへの AL の動的再接続を簡単に実行できるようになりました。

フローでのイテレーターの使用

アダプターのイテレーター・モードのインプリメントを容易にするために、イテレーターをフロー自体に配置できるようになりました。これを理解するために、イテレーターがどのように機能するかを簡単に確認します。

イテレーターは、以前は、AL サイクル全体を実行するために「フィード」セクションのみで、またはループ・コンポーネントを機能させるために「フロー」内のみで使用されていました。コネクターの最初の selectEntries() 操作 (アダプターの場合は、以下の『IBM Security Directory Integrator アダプターでの操作の使用』で説明しているように selectEntries 操作) が呼び出されて結果セットが作成され、次に getNextEntry() が呼び出されて、空になるまで結果セットからの読み取りが行われます。「フィード」セクションに対してイテレーターを制限すると、アダプターのイテレーター・コネクターから次のレコードを戻す getNextEntry 操作のインプリメントは非実用的となります。イテレーターをフローで使用すると、getNextEntry 操作でイテレーター・コネクターを利用することができ、操作がはるかに簡単になります。

消費のためのアダプターのパブリッシュ

導入部分に示されているシナリオでは、Anne が自分のアダプター・コンポーネントをパッケージ化し、それを他の人が消費できるようにパブリッシュするのを支援します。

アダプターが開発されると、IBM Security Directory Integrator 開発環境のパブリッシュ・コマンドによって、Anne の AL のパッケージが作成されます。AL のパブリッシュは、アダプター AL と Anne の残りの開発環境との間のすべての継承および依存関係を解決することを意味します。パッケージは、標準のスタンドアロン構成 XML ファイルで構成されています。このファイルには、他の IBM Security Directory Integrator 開発者に送信してそれらの開発者が自分のリソース・ライブラリーに組み込むことができるアダプター・コードのみが含まれています (詳しくは以下を参照)。

このパッケージは、IBM Security Directory Integrator の *Install* ディレクトリーの Packages ディレクトリーに保存することができます。パブリッシュされたアダプターは、使用可能なコネクターとしてコネクター・リストに表示されます。また、そのアダプターを AL コネクターから照会することもできます (『AssemblyLine でのアダプターの使用』セクションを参照)。

この段階で、パッケージとアダプターの違いについて少し混乱されているかもしれません。基本的に、アダプターは、コネクターとして使用することを目的としたパッケージのことです。他のパッケージには、AL FC または AL を実行するための他のメカニズムを使用して呼び出すことのできる AL のみが含まれている場合があります。

AssemblyLine でのアダプターの使用

インターフェース・メカニズムを提供することで、Anne が作成したアダプターを、Pete が自分の AL 内で他のコネクターと同様に利用できるようにします。

AL を別の AL から呼び出す場合使用できるメカニズムは多数存在します。ただし、AL がアダプターとして開発された場合は、主要なメカニズムとして使用するのは、17 ページの『AssemblyLine コネクター』 (AL コネクター) となります。このコネクターは、アダプター形式の AL を処理できます。古いバージョン (6.0 およびそれ以前) の IBM Security Directory Integrator の場合、別の AL の出力に対する繰り返しのみに使用することができました。現在、AL コネクターは、AssemblyLine で使用される場合、呼び出すアダプターを指定することで構成されます。次に、ターゲット・アダプターが操作のために検査され、開発者が使用できるコネクター・モードが決定されます。

便利な機能として、すべてのアダプターが IBM Security Directory Integrator によって自動的にラップされるため、コネクター・リストでコネクターのように表示することができます。これにより、Pete は自分のコネクター・リストからアダプターを直接挿入したり、AL コネクターを挿入し、その後に目的のアダプターを指定して呼び出したりすることができます。

アダプターの構成は、通常のコネクター構成パネルで行われます。ここに表示されるすべてのパラメーターが、アダプターの予約済み操作「$initialization」のスキーマで定義されています。これにより、コネクターの動的構成のために構成パラメーターをアダプターに送信するメカニズムが IBM Security Directory Integrator 開発者に提供されます。

ここでは、777 ページの『コネクターの柔軟な初期化』 が、アダプターを呼び出す AL およびアダプター自体の両方で使用できるという点で、役立ちます。アダプターの初期化として、「オンデマンド」または「常に」を使用すると、呼び出し側の AL は、実行段階で取得した情報を使用してアダプターを構成できるため、より静的なメカニズムを使用して事前に構成する必要がなくなります。

IBM Security Directory Integrator アダプターでの操作の使用

IBM Security Directory Integrator アダプターで操作を取り扱う際に、以下に示す情報とリンクを使用することができます。

IBM Security Directory Integrator コネクター・モードをインプリメントするには、Java または JavaScript でのインプリメントと同様に、すべてのコネクターでインプリメントする必要があるコネクター・プリミティブに対応したアダプターで AL 操作を作成する必要があります。

AL コネクターは、アダプターで定義された操作を検査することで、使用可能なモードを自動的に判断します。アダプターが公開するモードに対応する操作のみをインプリメントする必要があります。以下のテーブル内のいずれにも該当しない操作は、追加のアダプター・モードとして公開され、AL コネクターによる呼び出し/応答モードして実行されます。

アダプター操作のコネクター・モードへのマッピング

IBM Security Directory Integrator コネクターを Java または JavaScript でインプリメントする場合に考慮する必要があるメソッドについて以下に示します。

これらのメソッドと、インプリメントされるコネクター・モードとの間の関係を十分理解するために、785 ページの『付録 G. Java での独自コンポーネントのインプリメント』を参照してください。例えば、アダプターでルックアップ・モードをインプリメントするには、findEntry 操作のみを定義する必要があります。

表 66. 操作とモード

	イテレーター	ルックアップ	AddOnly	更新	削除	デルタ(2)	CallReply	サーバー(3)
initialize	(X)	(X)	(X)	(X)	(X)	(X)	(X)	(X)
querySchema	(X)	(X)	(X)	(X)	(X)	(X)	(X)	(X)
selectEntries	(X)							
getNextEntry	X							X
findEntry		X		X	X	(X)		
modEntry				X		X		
putEntry			X	X		X		(X)
deleteEntry					X	X		
queryReply							X	
getNextClient								X
terminate	(X)	(X)	(X)	(X)	(X)	(X)	(X)	(X)

注:

1. (X) は、オプションを示します。存在する場合に呼び出されます。
2. デルタ・モードは、特別な方法で処理されます。以下のセクションを参照してください。
3. サーバー・モードは、アダプターでは現在サポートされていません。
4. 操作名は大/小文字が区別されます。

アダプターにおける各操作のためのコードのインプリメント

アダプターで操作ごとのコードをインプリメントする際に、以下に示す情報とリンクを使用することができます。

アダプター AL は、公開されているモードのルールに従って、上記のテーブルで示した操作を介して 17 ページの『AssemblyLine コネクター』によって呼び出されます。アダプターでは、呼び出された操作を確認し、対応するコードを AL で実行する必要があります。Switch コンポーネントがこの目的に適していますが、操作とともに AL が呼び出されるたびに設定される op-entry.$operation 属性を使用して条件を作成することで、If-Else コンポーネントを使用することもできます。この属性は、当然のことながら、スクリプトでも使用できます。

Op-entry は、アダプター AL で使用できる項目オブジェクトです。AssemblyLine で作成される *work* オブジェクトに似ていますが、AL サイクルごとにクリアされま

せん。AL がそのライフ・サイクル全体で必要とする属性を格納するために使用されます。次のセクションでは、より詳しい使用方を示します。

$initialization 操作を使用したアダプター構成

アダプターの $initialization 操作のスキーマに定義されているすべての属性が、そのアダプターを呼び出す AL コネクターの構成パネルに表示されます。

これらの属性は、初期化時にアダプターに渡されるため、アダプターでは必要な準備およびターゲット・システムへの接続を実行できます。

$initialization スキーマで定義されている属性は、op-entry 属性の属性としてライフ・サイクル全体でアダプターが使用できます。

アダプターのコネクターは、コネクター・パラメーター・フィールドの式を使用して、これらの属性を指定して構成することができます。例えば、アダプターがその $initialization スキーマで *ou* 属性を定義している場合、アダプターのユーザーには、AL コネクターの構成パラメーターの 1 つとして「ou」が表示されます。次に、このアダプターで LDAP コネクターの検索ベースを以下のように定義できます。

```
cn=...,ou={op-entry.ou}
```

これらの属性は、アダプターの初期化時に使用できるようになります。アダプターの初期化時とは、上記の 777 ページの『コネクターの柔軟な初期化』セクションで説明したいずれかのメカニズムが使用されている場合を除き、デフォルトでは呼び出し側 AL の初期化時と同じです。

リンク基準について

以下に示すコード例を使用して、リンク基準を理解することができます。

AL コネクターで定義されているリンク基準は、op-entry オブジェクトの検索 オブジェクト (*SearchCriteria*) を介してアダプターに渡されます。

個々の基準オブジェクトの抽出は、以下のスクリプト・コードを使用して実行されます。

```
search = Task.getOpEntry().getObject("search");
criteria = search.getCriteria(0); /* index ranges from 0 to search.size() */
name  = criteria.name;      /* target attribute */
match = criteria.match;      /* expression (less, greater, equal.. */
value = criteria.value;      /* value to test the target attribute against through the expression */
negate = criteria.negate;      /* Boolean flag */
```

各基準オブジェクトには、name、match、value、および negate (ブール値) 属性が含まれています。

検索 オブジェクトによって、LDAP、Lotus Domino、および SQL の検索ストリングをそのリンク基準に基づいて作成する便利なメソッドが提供されます。詳しくは、Javadoc を参照してください。

属性マッピング

以下に示す情報とスクリプト・メソッドを使用して、属性のマッピングを実行することができます。

アダプター操作が AL コネクターから呼び出されると、作業項目が属性とともに AL コネクターの出力マップに取り込まれます。次に、AL コネクターは、次のセクションで説明するように、*work* または *conn* オブジェクトのいずれかの属性が戻されることを予期します。

呼び出し AL からアダプターへの処理

アダプターでは以下のようなスクリプト・メソッドを使用して、

```
email = work.getString("email");
```

E メール属性の値を抽出し、アダプター内でのさらなる属性マッピングに使用できるようにします。現実的提案として、AL レベル属性マップ (Attmap) コンポーネントを早期にアダプターに挿入して目的の属性を *conn* から抽出し、アダプターの他の部分で簡単に参照できるように *work* で表示されるようにすることをお勧めします。

アダプターから呼び出し AL への戻りデータ

イテレーター、ルックアップ、および *callreply* の各モードでは、呼び出し側 AssemblyLine にデータが戻され、AL コネクターの出力マップが取り込まれます。呼び出し側 AL に属性を戻す最も簡単な方法は、*work* オブジェクトを使用することです。アダプター・サイクルの最後に *work* に残っている属性はすべて、呼び出し側 AssemblyLine コネクターの入力マップに戻されます。このため、不注意にこれらの属性も戻されてしまわないように、アダプターの最後にある一時 work 属性を除去することが重要となります。以下に例を示します。

```
work.removeAttribute("attributeName");
...
```

このアダプターは、*work* の空の conn オブジェクトを戻すことによって、データの終わりを示しています。空の *work* オブジェクトは、AL コネクターによって空のレコードであると解釈されるだけなので、十分ではありません。イテレーターによってデータの終わりを示すには、以下を使用します。

```
work.newAttribute("conn");
```

ルックアップ・モード

ルックアップ・モードでは、複数のレコードが戻される場合があります。複数のレコードを戻す必要がある場合、アダプターでは、ゼロまたは 1 以上の値の項目タイプを含むことのできる *conn* 項目属性を作業項目に作成する必要があります。このセクションでは、これをアダプターで実現する方法を示すスクリプト・コードをいくつか後述します。

以下の JavaScript と疑似コードの混合では、*findEntry* 操作をインプリメントする (ルックアップ ・モードをインプリメントする) 部分を示しています。ここでの属性は、呼び出し側 AL の AL コネクターに戻すことのできる構造にマップされます。

この例では、複数のレコードを呼び出し側 AL に戻すための 2 つの方法を示しています。右側の例は、*work* がイテレーター・サイクルごとにクリアされるため、work のすべての値がイテレーターの出力マップの結果となり、acc (acc はアキュム

レーターの短縮形) に単一操作で追加できることから、より単純になっています。注目すべき点は、属性の値を *acc* に確実にコピーするために getClone() を使用する必要があることです。

	イテレーター・サイクルごとに work をクリア

`acc = system.newEntry().newAttribute("conn");`
`Loop on iterator (that returns attributes a,b,c from`
`target into work)`
`{`
`/* all of the below would be located in a script`
`component inside the Loop component */`
`temp = system.newEntry();`
`temp.setAttribute(work.getAttribute("a"));`
`temp.setAttribute(work.getAttribute("b"));`
`temp.setAttribute(work.getAttribute("c"));`
`acc.addValue(temp) ;`
`}`
`work.setAttribute("conn", acc);`
	`work.removeAllAttributes();`
	`acc = system.newEntry().newAttribute("conn");`
	`Loop on iterator (that returns attributes a,b,c`
	`from target into work)`
	`{`
	`acc.addValue(work.getClone());`
	`work.removeAllAttributes();`
	`}`
	`work.setAttribute("conn", acc);`

状況の表示

削除されたレコード、変更されたレコード、またはその他の処理が行われたレコードの数を示す属性 *recordsProcessed* を返すことが必要です。

この属性は、*work* オブジェクトでの場合と同様に呼び出し側 AL に戻すことができます。AL コネクターが呼び出し側 AL で 1 つのエラー・フックを呼び出す必要があるというエラー状況を示すには、アダプターが例外をスローする必要があります。これについて詳しくは、エラー処理に関するセクションを参照してください。

照会スキーマのインプリメント

ここに記載されている情報を使用することで、照会スキーマをインプリメントできるようになります。

アダプターのユーザーは、アダプターのスキーマを検出したいと考えます。これは通常、*接続* および*スキーマの照会* のためのボタンが存在する AL コネクターの構成時に行われます。アダプターで静的スキーマをインプリメントする場合、単純なソリューションは、アダプターに *querySchema* 操作を作成し、そこでスキーマを定義することです。*querySchema* 操作で定義されたスキーマは、すべての標準コネクター・モードの共通スキーマとなります。固有のスキーマは、非標準的なモードについて定義できます。例えば、アダプターで「AddUser」操作をインプリメントする場合は、定義された独自のスキーマを持つことができます。

デルタ・モード

デルタ・モードは、デルタ・データ (つまり、項目レベル、属性レベル、または値レベルでの変更用にタグ付けされた項目) を処理するために 2 つの異なるシナリオを使用できるため、他のモードとは若干異なる扱いとなります。

1. ターゲット・システム (またはアダプターでのインプリメント) で変更ベースの修正がサポートされている場合 (例えば、LDAP によって、属性のその他の値を指定しなくても、特定の項目の特定の属性で個々の値を更新できるなど)。これらのシステムは「デルタに強い」として定義されており、アダプターがタグ付き項目を処理できることを示します。

2. その他のシステムでは、削除、追加、または一連の検索のいずれかを実行し、適切な変更をレコードに適用してから、修正内容に関してレコード全体を書き戻すことで、デルタ・モードをシミュレートできます。これは、AL コネクターが基本的なアダプター・プリミティブを使用して実行できることですが、アダプターでは必要な機能であることを示す必要があります。

アダプターでデルタ動作を使用可能化するには、最初にデルタ操作を定義する必要があります。次のオプションでは、属性 *deltaSavvy* をデルタ・スキーマに作成します。deltaSavvy 属性がないと、AL コネクターは上記で説明したようにデルタ・モードをシミュレートします。deltaSavvy 属性が設定されると、AL コネクターは最初に *findEntry* を呼び出さずに、*modEntry* 操作を直接呼び出します。この場合、タグの属性を検査し、ターゲット・システムに適切なコマンドを適用するのはアダプターの役割となります。

エラー処理

呼び出し側の AssemblyLine がユーザーを AL コネクターのエラー・フックにドロップするように、アダプター・コードで例外をスローすることができます。以下に例を示します。

```
throw new java.lang.Exception ("error message");
```

付録 G. Java での独自コンポーネントのインプリメント

ここでは、このセクションの対象者について説明します。

このセクションは、IBM Security Directory Integrator 用の新しいコネクターや関数コンポーネントを作成する開発者を対象としています。開発者は、IBM Security Directory Integrator の操作について十分に理解しているだけでなく、Java 言語による開発の経験が必要です。

この資料では、パーサーの開発方法については説明せず、解析ロジックはコンポーネント自体にインプリメントされていることを前提とします。このテーマに関しては、別の文書が提供されます。

コンポーネント開発のサポート資料

`DirectoryConnector.java` ファイルには、このチュートリアルを読むときに役立つサンプルの Java コードが含まれています。

このファイルは、*TDI_install_dir/examples/connector_java* ディレクトリーにあります。

このセクションで言及するコア IBM Security Directory Integrator クラスの Javadoc はすべて、インストールした IBM Security Directory Integrator の `/docs/api` フォルダーにあります。この資料は、「**ヘルプ**」->「**ようこそ**」画面を選択し、構成エディターから「**JavaDocs**」リンクを選択することで表示することができます。

コネクターの開発

このセクションに記載されている情報を参照して、コネクターを開発することができます。

コネクターの Java ソース・コードのインプリメント

ここに記載されている情報を参照して、コネクターの Java ソース・コードを実装することができます。

すべての IBM Security Directory Integrator コネクターは、*com.ibm.di.connector.ConnectorInterface* という Java インターフェースをインプリメントしています。 このインターフェースは、IBM Security Directory Integrator 内でコネクターを使用する方法として考えられるものすべてに対応するためにインプリメントする多数のメソッドを提供します。通常、作成するコネクターには、IBM Security Directory Integrator が提供するすべてのオプションが必要というわけではなく、実際には *ConnectorInterface* インターフェースに提示されているメソッドのサブセットのみをインプリメントする必要があります。それを可能にするのが、*com.ibm.di.connector.Connector* クラスです。

com.ibm.di.connector.Connector は *ConnectorInterface* をインプリメントする抽象クラスであり、コネクターのコア機能 (例えば、コネクターの構成の処理など) を含んで

います。また、*ConnectorInterface* にあるメソッドの多くについて、空のインプリメンテーションやデフォルト・インプリメンテーションを提供しています。したがって、*com.ibm.di.connector.Connector* をサブクラス化することからコネクターのインプリメントを開始して、*ConnectorInterface* にあるメソッドのうち、自分にとって有用であり、コネクターに実際に必要なメソッドのみをインプリメントすることができます。

以下でリストするのは、実際のコネクターの基幹を構成する *ConnectorInterface* のメソッドです。通常は、これらのメソッドをインプリメントする必要があります。

コネクターのコンストラクター

コネクターのすべてのモードで必要です。

コンストラクターでは、通常、*setName(...)* メソッドでコネクターの名前を設定し、*setModes(...)* メソッドでコネクターがサポートするモード (イテレーター、ルックアップ、AddOnly、サーバー、デルタなど) を定義します。コネクターのインプリメンテーションの実例は、このパッケージに組み込まれている *DirectoryConnector.java* コネクターを参照してください。

public void initialize (Object object)

このメソッドは、サイクルを開始する前に AssemblyLine から呼び出されます。一般に、コネクターを作成し、プログラマチックにコネクターを使用するユーザーは、コネクターのコンストラクターを呼び出した後、他のいずれかのメソッドを呼び出す前に *initialize(...)* を呼び出す必要があります。

initialize(...) メソッドは、通常、コネクターのパラメーターを読み取り、指定されたパラメーター値に基づいて実際の作業に必要な準備 (接続の作成など) を行います。

public void selectEntries ()

イテレーター・モードで必要です。このメソッドは、コネクターがイテレーター・モードで使用されている場合にのみ、初期化の完了後に呼び出されます。

selectEntries(...) の中には、項目の反復処理を実際に開始する前に実行する必要のあるコードを指定します。コネクターがデータベースを操作する場合、そのコードで結果セットを戻す **SQL SELECT** 照会を実行することができます。コネクターが LDAP ディレクトリーを操作する場合は、そのコードは結果セットを戻す検索操作を実行します。*selectEntries(...)* の結果 (結果セットなど) は、後で *getNextEntry(...)* メソッドが使用して、各呼び出しまたは AssemblyLine の各反復の際に 1 つの項目を戻します。もちろん、項目を反復処理する前に準備が不要というケースもあり (ファイル・システム・コネクターの場合など)、その場合は *selectEntries(...)* をインプリメントする必要はありません。 *com.ibm.di.connector.Connector* をサブクラス化すると、何も行わないデフォルトのインプリメンテーションが継承されます。

public Entry getNextEntry ()

イテレーター・モードで必要です。このメソッドは、コネクターがイテレーター・モードの場合に、AssemblyLine の反復ごとに呼び出されます。

AssemblyLine の残りの部分を供給する 1 つの項目を戻すことが期待されます。

このメソッドをインプリメントするための一般的なガイドラインはありません。すべては、このコネクターがアクセスする情報によって決まります。このメソッドは、接続先のデータ・ソースからデータを取得し、項目オブジェクトを作成し、そのオブジェクトに属性を取り込む必要があります。例えば、データベース・コネクターであれば、表または結果セットから次のレコードを読み取り、そのレコードの各フィールドに対応する属性を組み込んだ項目オブジェクトを作成します。

public Entry findEntry (SearchCriteria search)

ルックアップ、更新、削除の各モードで必要です。コネクターがルックアップ操作を行うとき、AssemblyLine の反復ごとに 1 回呼び出されます。

このメソッドは、構成エディターの GUI で指定された「リンク基準」に基づいて、接続先のシステムから一致するデータを検索します。例えば、データベース・コネクターであれば、リンク基準に基づいて適切な **WHERE** 文節を指定して **SELECT** 照会を実行した後、*getNextEntry()* の場合と同じ方法で、データベース・レコードから項目オブジェクトを作成します。入力パラメーター SearchCriteria の構造については、Java Docs を参照してください。

- 指定されたリンク基準により、1 つの項目のみを検索できた場合は、その項目を戻します。
- 指定されたリンク基準でゼロ項目または複数項目が検索されたとき (つまり、1 つのみの一致以外の場合)、メソッドは **NULL** を戻します。ただし、複数項目が一致した場合にも、見つかった項目は提供されます。「複数項目時」フックからそれらの項目にアクセスできるようにするためです。

コネクターの動作に関する前述の要件を達成するには、次のようなインプリメンテーション・パターンを利用します。項目が見つかるごとに、コネクターの *addFindEntry(...)* メソッドを呼び出します。検索が完了したら、*getFindEntryCount(...)* を呼び出して見つかった項目の数を取得します。その数が 1 の場合は、*getFirstFindEntry(...)* から戻される値を戻します。それ以外の場合は、NULL を戻します。

例えば、データベース・コネクターでは、*modEntry(...)* は、項目パラメーターの属性をデータベース・フィールドとして使用し、検索パラメーターの SearchCriteria を使用して **WHERE** 文節を作成することで、**SQL UPDATE** 照会を実行します。

public void putEntry (Entry entry)

AddOnly および更新の各モードで必要です。コネクターが AddOnly モードで使用されている場合は、AssemblyLine の反復ごとに 1 回呼び出されます。更新モードでは、接続先のデータ・ソースに一致する項目が見つからなかった場合に呼び出されます。

このメソッドの目的は、項目オブジェクト (このメソッドにパラメーターとして渡されたもの) をコネクターのデータ・ソースに追加/保管/格納することです。したがって、データベース・コネクターは、項目オブジェクトの属性の名前と値、および表のフィールドの名前と値を使用して、**INSERT SQL** ステートメントを実行することになります。

public void modEntry (Entry entry, SearchCriteria search, Entry old)
— または —
public void modEntry (Entry entry, SearchCriteria search)
更新モードで必要です。

modEntry(...) メソッドについて説明する前に、更新モードのことを明確にしておく必要があります。AssemblyLine は、更新モードのコネクターを検出すると、まず、指定されたリンク基準を使用してコネクターの *findEntry(...)* メソッドを実行します。*findEntry(...)* で一致する項目が見つからない場合は、コネクターの *putEntry(...)* メソッドを呼び出して、項目をデータ・ソースに追加します。*findEntry(...)* メソッドで一致する項目が 1 つのみの場合は、コネクターの *modEntry(...)* メソッドを呼び出します。最後に、*findEntry(...)* メソッドで複数の項目が見つかった場合は、「複数項目時」フックを実行し、ユーザーの指定内容に応じて、コネクターの呼び出しを一切呼び出さないか、*modEntry(...)* メソッドの代わりとして 1 つの *putEntry(...)* を呼び出します。

上記のとおり、*modEntry(...)* メソッドには 2 つのバリエーションがあります。入力パラメーターが 3 つのものと、2 つのものです。いずれのケースでも、使用するパラメーターは次の 2 つです。1 つは *entry* で、データ・ソースに書き込む準備のできた、出力マップ済みの *conn* 項目です。もう 1 つは *search* で、基礎にあるシステムに変更呼び出しを行うために使用される SearchCriteria です。このメソッドが更新モードのロジックから呼び出された場合 (AssemblyLineComponent の *update(...)* メソッド)、このパラメーターは、リンク基準から作成された実際の SearchCriteria (属性値の評価などの実行後) を参照します。

追加のパラメーターは *old* です。これは、データ・ソースにある元の項目で、現時点での、変更を適用する前の内容を表しています。この情報は、「名前変更」操作など、名前変更を実行するために古い名前が必要な場合に便利です。

この 2 つのメソッドのいずれを使用するかは、開発者が決定します。もちろん、両方をインプリメントすることもできます。作成するコネクターで更新モードをサポートするには、いずれか 1 つで十分です。

データベース・コネクターの場合を考えると、*modEntry(...)* は、項目パラメーターの属性をデータベース・フィールドとして使用し、検索パラメーターのデータを使用して SQL 照会の **WHERE** 文節を作成することで、**SQL UPDATE** 照会を実行します。

public Entry queryReply (Entry entry)
CallReply モードで必要です。コネクターが CallReply モードで使用されるとき、AssemblyLine の反復ごとに 1 回呼び出されます。

このモードは、コネクターが要求/応答通信に参加する場合に適しています。出力マップ済みの *entry* パラメーターには、操作の「呼び出し」または「要求」の部分を実行するのに必要なデータが含まれます。例えば、Web サービス・コネクターは、*entry* の属性に基づいて SOAP 呼び出しを作成し、送信します。その後、このメソッドは、返答/応答データから項目オブジェクトを作成して、戻します。

public void deleteEntry (Entry entry, SearchCriteria search)

削除モードで必要です。

削除モードのコネクターは、*findEntry(...)* を実行して、削除する項目を見つけようとします。*findEntry(...)* メソッドから、1 つのみ項目が戻された場合は、この項目と、ルックアップで使用したリンク基準をパラメーターとして指定して、コネクターの *deleteEntry(...)* メソッドを呼び出します。*findEntry(...)* から 0 個または複数の項目が戻された場合は、対応するコネクター・フックが呼び出されます。スクリプト・コードにユーザーが指定した内容に応じて、それ以上何も実行されないか、AssemblyLineComponent の *setCurrent(entry)* メソッドを使用してユーザー・スクリプトで指定された項目について *deleteEntry(...)* メソッドが呼び出されます。「複数項目時」フックで現在の項目を設定しない限り、それ以上の処理は行われず、制御は AssemblyLine の次のコンポーネントに渡されます。

これまでと同様、データベース・コネクターの例で考えると、*deleteEntry()* は SQL DELETE ステートメントを実行します。

public ConnectorInterface getNextClient()

getNextClient() メソッドはサーバー・モードのコネクターに使用され、クライアント要求を受け入れます。通常、このメソッドはクライアント要求が着信するまでブロックします。コネクターは要求を受信すると、コネクター自体の新規インスタンスを作成して戻します。この新規インスタンスは AssemblyLine に渡され、この AssemblyLine はコネクター・インスタンスの新規 AssemblyLine スレッドを作成します。次に、AssemblyLine は、処理する項目がなくなるまで、新規スレッドの新規コネクターのインスタンスに対して *getNextEntry()* メソッドを呼び出します。*getNextClient()* メソッドが戻り AssemblyLine がクライアント要求を処理するための新規スレッドを作成した直後に、AssemblyLine は *getNextClient()* を再度呼び出し、次のクライアント要求を受け入れます。

サーバー・モードのコネクターは応答が必要なクライアント要求を処理するため、AssemblyLine はその終了時に *replyEntry(...)* コネクター・メソッドを呼び出します。このメソッドを使用して、クライアントへ応答を戻すコードを配置します。コネクターが 1 つの要求に対して複数の応答を戻す必要がある場合は、個別の応答項目を戻すように *putEntry(...)* メソッドをコーディングできます。この場合、AssemblyLine 開発者が、スクリプトを作成することによってコネクターの *putEntry(...)* メソッドを呼び出す必要があります。また、このことについてコネクターの資料に記載されている必要があります。

コネクターをサーバー・モードでインプリメントする際には、外部要求に対するコネクターの強制終了についても注意する必要があります。完了コードを *terminateServer()* メソッドに挿入します。このメソッドは、クライアント要求を受け入れるマスター・コネクター・インスタンスと、クライアント要求を処理する子コネクター・インスタンスに対して呼び出すことができる点を考慮してください。どちらの場合でも、適切な終了処理が必要です。一般に適切な終了処置とは、マスター・サーバー・コネクター・インスタンスからの新規要求の受け入れを停止するが、すべての子コネクターに各自の処理を完了させることです。*terminateServer()* メソッドは、通常、マスター・サーバー・コネクター・インスタンスの *getNextClient()* メソッドによりチェッ

クされたフラグを設定します。強制終了が要求されると、*getNextClient()* メソッドは NULL を戻します。これにより AssemblyLine は、このサーバー・コネクターが終了しており、AssemblyLine はその *getNextClient()* メソッドを今後呼び出さないことを認識します。

public void terminate ()

terminate(...) メソッドは、AssemblyLine がサイクルを完了した後、処理を終了する前に呼び出されます。ここには、終結処理コードを指定します。つまり、接続を解放したり、*initialize(...)* メソッド中やその後の処理中に作成したり、ソースを解放したりします。

ここまでリストしたメソッドは、使用するコネクターに指示を与えるコア *ConnectorInterface* メソッドです。また、インプリメントする必要のあるメソッドは、使用するコネクターでサポートするコネクター・モードに対応するもののみです。

モードからメソッドへのマッピング

コネクターを作成する場合、ユーザーが特定の順序でコネクターのメソッドを呼び出すとは限らないことを考慮に入れる必要があります。つまり、コネクターが特定のメソッドを他のメソッドより先に呼び出す必要がある場合は、各メソッドに対して整合性チェックを実行する必要があります。IBM Security Directory Integrator サーバーから見た場合、特定のコネクター上で呼び出されるメソッドは、モード・パラメーターの値によって決定されます。このセクションでは、メソッドの呼び出し順序についてモードごとに説明しています。AssemblyLine でコネクターを使用する場合、AssemblyLine は常に、最初のメソッドとして `initialize()` を呼び出してから、その他のメソッドを呼び出します。ただし、ユーザーが、このメソッドの呼び出しの前に別の呼び出しを挿入する可能性があります。

モード	メソッド	コメント

イテレーター	`Initialize()`
	`selectEntries()`
	`getNextEntry()`
	`terminate()`
		getNextEntry が入力の終了を知らせるためには NULL を戻す必要があります。

AddOnly	`Initialize()`
	`putEntry()`
	`terminate()`	

ルックアップ	`nitialize()`
	`findEntry()`
	`terminate()`
		項目が複数ある場合は、このメソッドで検出された項目ごとに `clearFindEntries()` と `addFindEntry()` を使用する必要があります。

削除	`Initialize()`
	`findEntry()`
	`deleteEntry()`
	`terminate()`	

更新	`Initialize()`
	`findEntry()`
	`putEntry()`
	`modEntry()`
	`terminate()`
		findEntry によって単一の項目が戻された場合は、`modEntry` が呼び出されます。findEntry によって NULL が戻された場合は、`putEntry` が呼び出されます。

モード	メソッド	コメント
デルタ	Initialize() findEntry() putEntry() modEntry() deleteEntry() terminate()	以下の説明を参照してください。

デルタ・モード・コネクターのインプリメント方法

コネクターに対してデルタ・モードを有効にするには、最初に、サポートされているモードのリストに「デルタ」を追加する必要があります。デルタ・モードは、AssemblyLine でエミュレートすることも、コネクターでインプリメントすることもできるという点で若干特殊です。エミュレートされたデルタ・モードとは、単に、findEntry() メソッドからの戻り値、およびターゲット・システムに書き込まれているものに基づいて、インクリメンタル更新が AssemblyLine によって生成されることを意味しています。ターゲット・システムがインクリメンタル更新をサポートしている場合、デルタ項目オブジェクトを、コネクターの基礎となるプロトコルに変換することによって、コネクターをコード化することができます。後者の場合、isDeltaSupported() メソッドに true を戻すことによってコネクターを構成します。これにより、AssemblyLine は、デルタ項目をコネクターの putEntry()、modEntry()、または deleteEntry() メソッドに直接転送し、findEntry() とデルタ項目を計算するアルゴリズムを省略します。

スキーマの照会の動作

IBM Security Directory Integrator では、すべてのコネクターについて、スキーマ・ディスカバリーのデフォルトの動作がインプリメントされています。このデフォルトの動作は、独自のスキーマ処理ロジックをインプリメントしていないコネクター (つまり、querySchema(Object) メソッドをオーバーライドしないコネクター) によって使用されます。デフォルトの動作は、コネクターで使用されているパーサー (存在する場合) に依存しています。

静的スキーマ

コネクターの静的スキーマはそれぞれ、コネクター自体の静的スキーマと、コネクターが使用するパーサーの静的スキーマによって決定されます。両方のスキーマが、コネクターの入出力マップに表示されます。

静的スキーマとは、コネクターとパーサー用の tdi.xml ファイル内で構成されているスキーマのことです。コネクター、パーサー、または関数の定義ファイルに静的スキーマ定義を追加するには、<Connector>、<Parser>、または <Function> エレメント内に <Schema> タグを追加します。スキーマの名前は、「Input」または「Output」である必要があります。

以下に例を示します。

```
<Connector name="ibmdi.Mailbox">
 <Schema name="Input">
  <SchemaItem>
   <Name>mail.body</Name>
   <Syntax>javax.mail.Multipart</Syntax>
  </SchemaItem>
 </Schema>
 <Schema name="Output">
  <SchemaItem>
   <Name>Flag.Answered</Name>
```

```
    <Syntax>boolean</Syntax>
   </SchemaItem>
  </Schema>
 </Connector>
```

動的スキーマ

このタイプのスキーマには、コネクターの接続先として構成されるシステムとの対話が必要となります。このために、コネクターは、構成済みパーサーに対して、パーサーの構成を使用してスキーマをディスカバーできるかどうかを依頼することができます。

querySchema() のインプリメント

ConnectorInterface は、コネクターで利用可能なさまざまな局面に対応するためのメソッドも提供しています。それらのメソッドも、必要に応じてインプリメントすることができます。例えば、*querySchema(...)* があります。このメソッドは、接続先のデータ・ソースのスキーマを戻します。これをインプリメントした場合、構成エディターは戻り値をコネクターのスキーマとして表示します。

これらの戻り値は、項目オブジェクトのベクトルとして、スキーマ内の列/属性ごとに 1 つずつ格納されます。例えば、データベース・コネクターは、接続先のデータベース表にある各列について、1 つの項目を戻します。

メソッドから戻すベクトル内の各項目には、以下の属性を組み込む必要があります。

name	属性の名前 (column、field など) 必須。
syntax	この属性の構文 (**VARCHAR** や **TIMESTAMP** など)、または期待される値のタイプ。オプション。

指定元: ConnectorInterface の *querySchema*

パラメーター: *source* - スキーマを検索する対象のオブジェクト。通常は NULL。項目またはストリング値で指定できます。

戻り値: 各エンティティーを記述した *com.ibm.di.entry.Entry* オブジェクトのベクトル。または、エラーが発生した場合は java.lang.Exception がスローされます。

コネクターでのパーサーの使用

コネクターが IBM Security Directory Integrator コネクター (com.ibm.di.connectors.Connector) の基本インプリメンテーションを拡張している場合、*initParser()* メソッドを呼び出し、関連付けられているパーサーを初期化することができます。

```
/**
 * Initialize the connector's parser with input and output streams. If the parser
 * has not been loaded then an attempt is made to load it. The input and output objects
 * may be Stream objects (InputStream,OutputStream), java.io.Reader object, String object,
 * java.net.Socket, byte and character array objects.
 *
 * @param  is  The input object.
 * @param  os  the output object.
 * @exception  Any exception thrown by the parser
 * @see    #getParser
 */
public void initParser (Object is, Object os) throws Exception;
```

パーサーが読み取り/書き込み操作に使用する入出力ストリームを用意する必要があります。通常は、コネクターのモードでフローの方向が決まります (*initialize(Object obj)* コネクター・メソッドのコネクター・モードは、「obj」オブジェクトのコネクター・モードとなることに注意してください)。コネクターの初期化時にパーサーを初期化する必要はありませんが、他の場所で初期化する明確な理由がある場合を除き、コネクター初期化時に初期化しておきます。いずれの場合も、ロギング・オブジェクト、デバッグ・フラグ、および他の標準的な IBM Security Directory Integrator オブジェクトやその動作とともに適切に初期化するには、*initParser()* メソッドを呼び出す必要があります。

パーサーはユーザーが選択することができます。あるいは、パーサーの選択を非表示にし、tdi.xml ファイルで構成を指定するか、またはコネクターでパーサーをプログラマチックに構成することができます (あるいはその両方が可能です)。

1. **ユーザーがパーサーを選択するようにします。**

   この場合、コネクターの「tdi.xml」ファイルにある「parserOption」パラメーターの値を「true」に設定する必要があります。このフィールドを定義すると、パーサーの選択は、標準ユーザー・インターフェースを介してユーザーに委任されます (tdi.xml ファイルの parserConfig セクションをデフォルトのパーサーで事前に埋めておくことができます)。以下に、FileSystem コネクターの「tdi.xml」ファイルのスニペットを示します。このスニペットでは、「parserOption」が「Required」と示されています。したがって、コネクターにパーサーが必要であることを意味します (つまり、パーサーが構成内に定義されていないとエラーがスローされます)。

   ```
   <Connector name="ibmdi.FileSystem">
    <Configuration>
    ...
     <parameter name="parserOption">Required</parameter>
    </Configuration>
   </Connector>
   ```

   「parserOption」パラメーターには、「Required」、「Useless」(パーサーは使用不可)、または「Optional」の値を指定できます。

2. **「tdi.xml」ファイルで事前定義されているパーサーを使用します。**

   CSV パーサーから継承する場合など、常に同じパーサーを使用する場合は、「tdi.xml」ファイルに parserConfig セクションを組み込むことができます。

   ```
   <Connector name="myconnector">
    <Configuration>
     <parameter name="parserOption">Required</parameter>
    </Configuration>
    <Parser>
     <InheritFrom>system:/Parsers/ibmdi.CSV</InheritFrom>
     ... Optional parameter values to make the parser functional
    </Parser>
   </Connector>
   ```

3. **実行時にパーサーを構成します。**

   コネクターは、*Connector.getConfiguration()* メソッドを使用して、ConnectorConfig オブジェクトにアクセスします。 ConnectorConfig オブジェクトによって、コネクターの ParserConfig インターフェースを取得することができます。そのオブジェクトを使用して、パーサーを構成した後に *initParser()* メソッドを呼び出します。

```
import com.ibm.di.config.interfaces.ConnectorConfig;

 public void initialize(Object obj) throws Exception {

  // Check mode
  String mode = "" + obj;
  boolean isIterator = mode.equals(ConnectorConfig.ITERATOR_MODE);

  ConnectorConfig cc = (ConnectorConfig)getConfiguration();

  // Get the parser config object
  ParserConfig parser = cc.getParserConfig();

  // -- use the csv parser and set the column separator parameter
  parser.setParameter("parserType", "com.ibm.di.parser.CSVParser");
  parser.setParameter("csvColumnSeparator", "¥t");

  if(isIterator)
   initParser(inputStream, null);
  else
   initParser(null, outputStream);
```

パーサーが初期化された後、*readEntry()* および *writeEntry()* メソッドを呼び出して com.ibm.di.entry.Entry オブジェクトをパーサーで定義されたストリーム・フォーマットに変換したり、その逆の変換を行ったりすることができます。通常、*readEntry()* メソッドは *getNextEntry()* メソッドで、writeEntry メソッドは putEntry メソッドから呼び出します。パーサーのインターフェース・ハンドルは、*getParser()* メソッドを使用して取得します。

4. **オプションのパーサーおよび動的再初期設定を行います。**

   パーサーの有無にかかわらずコネクターが機能する場合は、*hasParser()* メソッドを呼び出してパーサーが構成されているかどうかを確認することができます。

```
if(hasParser())
   doSomething();
```

   コネクターの存続時間中にパーサーの複数インスタンスを使用する場合、パーサーのインターフェースをクローズし、確実にデータが出力ストリームに書き込まれ、システム・リソースが解放されるようにします。どのパーサーを使用するかによってパーサーの再初期化に使用するメソッドは異なりますが、ほとんどのパーサーは以下のメソッド呼び出しで十分です。

```
  // Close parser to release system resources
  if(getParser() != null)
   getParser().closeParser();

  // assuming you just got a new input stream ... reinitialize the parser
  initParser(inputStream, null);
```

   コネクターは、終了時に自動的に *closeParser()* メソッドを呼び出します (コネクターがこのメソッドを使用中の場合)。

コネクターからのロギング

使用可能な最も単純なロギングの方法を以下に示します。

コネクターによって拡張される **com.ibm.di.connector.Connector** クラスには、AssemblyLine の構成済みログ・ファイルへのメッセージの書き込みを可能にする数多くのメソッドがあります。

```
 /**
   * Log a message to the connector's log. The message is prefixed by the connector's
   * name.
   *
   * @param  msg  The message to write to the log
   */
  public void logmsg(String msg)
```

```
/**
 * Log a debug message to the connector's log
 *
 * @param  msg  The message to write to the log
 */
public void debug(String msg)
```

これらのメソッドは、以下のようなコードで呼び出すことができます。

```
logmsg("initializing my connector");
```

これにより、AssemblyLine の構成済みログ Appender に INFO レベルでストリングが発行されます。より詳細なロギングにするために、com.ibm.di.connector.Connector クラスにはこのフィールドもあります。

```
/**
  * The log object for logging messages
  */
  protected  com.ibm.di.server.Log        myLog;
```

この **com.ibm.di.server.Log** クラスには、ロギング用のメソッドが多数あります。したがって、myLog オブジェクトを使用して、以下のようにロギングすることができます。

```
  myLog.logerror("Something very bad happened");
```

これにより、ERROR レベルでメッセージがログに発行されます。さまざまなレベルでのロギングに対応する loginfo() や logfatal() などのメソッドがあります。

コネクターのソース・コードの作成

コネクターのソース・コードを作成するときには、CLASSPATH に、インストールした IBM Security Directory Integrator の jars/commonフォルダーにある jar ファイルを組み込みます。

少なくとも、miserver.jar と miconfig.jar を組み込む必要があります。

注: 作成した Java コードを IBM Security Directory Integrator と統合する際には、IBM Security Directory Integrator を構成する既存のコンポーネントのコレクション (特に *jars* ディレクトリー内のファイル) に注意してください。作成したコードが依存しているユーザーのライブラリー・コンポーネントの 1 つが、IBM Security Directory Integrator システムに含まれる 1 つ以上のコンポーネントとオーバーラップしたり、それらのコンポーネントを破壊したりする場合は、実行中にローダーの問題が発生したことが考えられます。

コネクターの GUI 構成フォームのインプリメント

カスタムの IBM Security Directory Integrator コンポーネントを作成する場合は、コンポーネントが記述された追加のファイルも IBM Security Directory Integrator に提供する必要があります。

このファイルは *tdi.xml* という名前で jar ファイルのルートにあります。このファイルの構文と内容については、本書で説明しています。

このセクションの前半では、tdi.xml ファイルのフォーマットとコンポーネント定義ファイルの最小必要要件について説明します。

このセクションの後半では、フォーム定義とフォーム定義時に使用できる各種オプションを中心に説明します。IBM Security Directory Integrator 構成エディターでは、このフォーム定義を使用してユーザーがコンポーネントを構成できます。フォーム定義の UI オプションは基本的なものであり多少制限がありますが、各自のカスタム Java ベース UI コンポーネントを使用し、フォーム・イベントにスクリプトを関連付けることで、拡張操作を実行できます。

tdi.xml ファイルのフォーマット

ここに記載されているコード例を使用して、tdi.xml ファイルのフォーマットを作成することができます。

このファイルは、IBM Security Directory Integrator 構成ファイルとほぼ同じ XML フォーマットで作成されています。

ファイルのスケルトンは、以下のようになります。

```
<?xml version="1.0" encoding="UTF-8">
<MetamergeConfig version="7.0">
 <Folder name="Connectors">
  <Connector name="CustomConnector">
   <Configuration>
    <parameter name="connectorType">com.acme.CustomConnector</parameter>
    ... more parameters ...
   </Configuration>
  </Connector>
 </Folder>
 <Folder name="Forms">
  <Form name="com.acme.CustomConnector">
   <TranslationFile>CustomConnector</TranslationFile>
   ... many more elements which will be defined later ...
  </Form>
 </Folder>
</MetamergeConfig>
```

これにより、system:/Connectors/CustomConnector という名前のコネクターが定義されます。このコネクターをインプリメントする Java クラスは、`com.acme.CustumerConnector.class` です。

このファイル内のラベルおよび記述は、*locale* 識別子を使用して標準的な方法でプロパティー・ファイルを追加することによってローカライズすることができます。この例では、プロパティー・ファイルは CustomConnector.properties です。このファイルのドイツ語版は CustomConnecter_de.properties、ブラジル・ポルトガル語版は CustomConnector_pt_BR.properties になります。ローカライズされたこれらのファイル内の個々のプロパティーのキーは同じですが、値がローカライズされています。各行の形式は、以下のとおりです。

```
key=value
```

これらのファイルにコメントを含めるには、行の先頭を # (ハッシュ) で開始します。

基本コンポーネントの定義:

ここに記載されている構文を使用して、コンポーネント定義を作成することができます。

コンポーネント定義ファイルを初めて作成する場合は、jar ファイルに含まれているコンポーネントのメイン・セクションを追加します。コンポーネントごとに、最小限でも Java クラスを定義するセクションを追加します。3 つの主要コンポーネントの構文を以下に示します。

表 67. コンポーネント・セクションの構文

コンポーネント・タイプ	セクションの最小限の内容
Connector	<Folder name="Connectors"> <Connector name="*your_name*"> <Configuration> <parameter name="connectorType">*your_javaclass_name*</parameter> </Configuration> </Connector> </Folder>
パーサー	<Folder name="Parsers"> <Parser name="*your_name*"> <parameter name="class">*your_javaclass_name*</parameter> </Parser> </Folder>
関数	<Folder name="Functions"> <Function name="*your_name*"> <Configuration> <parameter name="javaclass">*your_javaclass_name*</parameter> </Configuration> </Function> </Folder>

また、常にコンポーネントごとにフォーム定義を組み込んでください。これにより、構成エディターからフォーム不足のエラーが報告されることを防止できます。コンポーネントに構成可能なパラメーターがない場合は、そのように記述したフォームを組み込んでください。

注: *Form* により参照される現行構成オブジェクトは、常に *connectorConfig/parserConfig/functionConfig* オブジェクトです。主要コンポーネントのパラメーターにアクセスする必要がある場合は、*config.getParent()* メソッドを使用して構成の ConnectorConfig インターフェースなどを取得します。

インストール・ロケーション:

ここに記載されている情報を使用することで、インストール・ロケーションを理解できるようになります。

構成エディターまたはサーバーを開始すると、IBM Security Directory Integrator ローダーと呼ばれるコンポーネントが構成 jar ディレクトリーを対象に実行され、ルート・レベルで「tdi.xml」ファイルが含まれている *.jar/*.zip ファイルを検索します。これらのファイル内のすべての定義が、システムのネーム・スペースに配置されます。

これらのファイルが格納されている場所を以下に示します。

- *TDI_Install_directory*/jars およびその中のサブディレクトリー
- com.ibm.di.loader.userjars プロパティーにより指定されるファイル/ディレクトリー (etc/global.properties)

jar ファイルをこれらのディレクトリーのいずれかに保管すると、構成エディターでコンポーネントが表示されるときに、システム・ネーム・スペースの一部として選択した名前で表示されます。

注: jar ファイルを CLASSPATH または PATH に追加しても、jar ファイルはシステム・テンプレートには組み込まれないため、ユーザーに対して表示されません。

フォーム記述

フォーム記述を使用して、コンポーネントのカスタム入力パネルを準備することができます。

構成エディターのほとんどのユーザー・インターフェースは静的ですが、ほとんどのコンポーネントは、ユーザーが動作を定義できするようにするために固有のユーザー・インターフェースを必要とします。

コンポーネント/フォームの関連性:

コンポーネントの構成を開くとき、フォーム定義によって、構成エディターが構築する入力フィールドとラベルが定義されます。

コンポーネント (コネクター、パーサーなど) とそのフォームは、コンポーネントの Java クラスによってバインドされます。前述の例の場合、connectorConfig の connectorType パラメーターは、コンポーネントのインプリメント・クラス (com.acme.CustomConnector) を定義します。このタイプのコンポーネントがユーザーに対して表示されるときに、構成エディターはインプリメント Java クラスと同名のフォームを検索します。

フォーム/構成のバインディング:

ここに記載されている情報を使用することで、フォーム/構成のバインディングを理解できるようになります。

作成されたフォームには、構成オブジェクトの各パラメーターのバインディング・オブジェクトがあります。これらのバインディング・オブジェクトにより、入力フィールドの初期値が設定されます (構成オブジェクトからその値に対して NULL が戻される場合はフォームのデフォルト値を使用)。また、バインディング・オブジェクトは入力フィールドと構成オブジェクトの間のコントローラーとしても機能します。入力フィールドの値が変更されると、バインディングにより構成オブジェクトが変更されます。また、この逆の処理も行われます。構成オブジェクトの読み取りと更新には構成オブジェクトのプリミティブ (BaseConfiguration.getParameter/setParameter など) が使用されます。バインディング・オブジェクトがプリミティブを使用する代わりに特定のメソッドを呼び出すようにすることもできますが、コンポーネント開発者がこの操作を必要とすることはほとんどありません。

フォーム定義:

ここに記載されているコード例を使用して、フォーム定義について理解することができます。

フォームはコンポーネントと同様の方法で定義されます。3 つの入力フィールドと、いずれかのパラメーターの変更をトラッピングする 1 つのイベント・ハンドラ

ーを備えたフォームの例を以下に示します。フォーム定義は、General と Advanced の 2 つのセクションに分けられます。General セクションには 2 つのパラメーター (「firstParameter」および「$GLOBAL.debug」)、Advanced セクションにはパラメーターが 1 つだけ (「secondParameter」) あります。

ここでは、2 つのパラメーターに対してラベルを 1 つだけ定義しています。$GLOBAL.debug は、IBM Security Directory Integrator のグローバル・パラメーターです。このパラメーターを選択すると、詳細ログが有効になります。

```
<Folder name="Forms">
 <Form name="com.acme.CustomConnector">
  <TranslationFile>CustomConnector</TranslationFile>
  <parameter name="title">title_key</parameter>
  <parameter name="formevents">function firstParameter_changed()
          { form.alert("First param modified"); }</parameter>
  <FormSectionNames>
   <ListItem>General</ListItem>
   <ListItem>Advanced</ListItem>
  </FormSectionNames>
  <FormSection name="General">
   <FormSectionNames>
    <ListItem>firstParameter</ListItem>
    <ListItem>$GLOBAL.debug</ListItem>
   </FormSectionNames>
  </FormSection>
  <Formsection name="Advanced">
   <parameter name="title">Advanced_Title</parameter>
   <parameter name="initiallyExpanded">false</parameter>
   <FormSectionNames>
    <ListItem>secondParameter</ListItem>
   </FormSectionNames>
  </FormSection>
  <FormItem name="firstParameter">
   <parameter name="label">first_param_label</parameter>
  </FormItem>
  <FormItem name="secondParameter">
   <parameter name="label">second_param_label</parameter>
  </FormItem>
 </Form>
</Folder>
```

変換ファイル (CustomConnector_en.properties) には以下が含まれます。

```
title_key=This is the title/heading that appears at the top of the form
Advanced_Title=This is the title heading for the section for Advanced Users
first_param_label=First Param Label
second_param_label=Second Param Label
```

フォームの定義エレメント:

ここに記載されているコード例を使用して、フォーム・エレメントを定義することができます。

FormSection エレメントには、FormSection または FormItem のリストが含まれています。このリストには <FormSectionNames> タグがあります。FormSection には、オプションで FormItem のタイトルと再定義を含めることができます。これらの FormItem は、FormSection が属している Form 内の FormItem を継承します。これにより、例えばその FormItem のツールチップをオーバーライドすることができます。 Form には、FormSection のリストが含まれます。このリストには <FormSectionNames> というタグが付けられています。FormSection のすべてのリストで、$Mode がコネクターの現行モードで置換されます。これにより、コネクターのモードに応じたパラメーターを表示することができます。

以下に示すのは、若干複雑な完全なフォームの例です。

```
<Form name="com.ibm.di.connector.FileConnector">
   <TranslationFile>NLS/idi_conn_filesys</TranslationFile>
   <FormItemNames>
    <ListItem>filePath</ListItem>
    <ListItem>fileAwaitDataTimeout</ListItem>
```

```
 <ListItem>fileAppend</ListItem>
 <ListItem>exclusiveLock</ListItem>
 <ListItem>$GLOBAL.debug</ListItem>
 <ListItem>$GLOBAL.help</ListItem>
</FormItemNames>
<FormSectionNames>
 <ListItem>$Mode-General</ListItem>
 <ListItem>$Mode-Advanced</ListItem>
</FormSectionNames>
<FormSection name="Iterator-General">
 <FormSectionNames>
  <ListItem>filePath</ListItem>
 </FormSectionNames>
 <FormItem name="filePath">
  <parameter name="description">path_desc_in</parameter>
 </FormItem>
 <parameter name="title">General_title</parameter>
</FormSection>
<FormSection name="AddOnly-General">
 <FormSectionNames>
  <ListItem>filePath</ListItem>
  <ListItem>fileAppend</ListItem>
 </FormSectionNames>
 <FormItem name="filePath">
  <parameter name="description">path_desc_out</parameter>
 </FormItem>
 <parameter name="title">General_title</parameter>
</FormSection>
<FormSection name="Iterator-Advanced">
 <FormSectionNames>
  <ListItem>fileAwaitDataTimeout</ListItem>
  <ListItem>exclusiveLock</ListItem>
 </FormSectionNames>
 <FormItem name="exclusiveLock">
  <parameter name="description">exlock_desc_in</parameter>
 </FormItem>
</FormSection>
<FormSection name="AddOnly-Advanced">
 <FormSectionNames>
  <ListItem>exclusiveLock</ListItem>
 </FormSectionNames>
 <FormItem name="exclusiveLock">
  <parameter name="description">exlock_desc_out</parameter>
 </FormItem>
</FormSection>
<FormItem name="exclusiveLock">
 <parameter name="label">exlock_label</parameter>
 <parameter name="description">exlock_desc</parameter>
 <parameter name="syntax">boolean</parameter>
</FormItem>
<FormItem name="fileAppend">
 <parameter name="description">append_desc</parameter>
 <parameter name="label">append_label</parameter>
 <parameter name="syntax">boolean</parameter>
</FormItem>
<FormItem name="fileAwaitDataTimeout">
 <Values>
  <ListItem>-1</ListItem>
  <ListItem>10</ListItem>
  <ListItem>60</ListItem>
 </Values>
 <parameter name="description">time_desc</parameter>
 <parameter name="label">time_label</parameter>
 <parameter name="syntax">DROPEDIT</parameter>
</FormItem>
<FormItem name="filePath">
 <Values>
  <ListItem>&lt;&gt;</ListItem>
 </Values>
  <parameter name="description">path_desc</parameter>
  <parameter name="label">path_label</parameter>
  <parameter name="script">selectFile</parameter>
  <parameter name="scriptLabel">path_sript_label</parameter>
  <parameter name="scripthelp">path_script_help</parameter>
  <parameter name="syntax">DROPEDIT</parameter>
</FormItem>
<parameter name="title">CONN_TITLE</parameter>
</Form>
```

XML タグの定義

- <Form> はフォームを定義します。
- <Form> 内には、以下のようなタグがある場合があります。
  - <TranslationFile> - 変換ファイルの名前を定義します。
  - <FormItemNames> - FormSection を認識しない古い構成エディターで使用する FormItem のリスト。
  - <FormSectionNames> - 表示する FormSection/FormItem のリスト。
  - <FormSection> - FormSection を定義します。
  - <FormItem> - FormItem を定義します。
  - <parameter> - 指定できるパラメーターは以下のとおりです。
    - title - フォームのタイトル。変換ファイルがあり、キーが見つかると変換されます。変換ファイルがまったくないか、キーが見つからない場合は、その値自体が使用されます。
    - description - フォームの詳細な記述。
    - formscript - スクリプト付きのボタンの押下のたびに実行される JavaScript コード。例えば、そのボタンのスクリプトが使用できるメソッドを定義します。
    - formevents - このパラメーターも JavaScript であり、フォームの作成時に実行されます。このパラメーターの目的は、特定の値に設定されているフィールドに応答できるメソッドを定義することです。例えば、他のフィールド用のドロップダウンを取り込みます。フィールドの名前が fieldName で fieldName_changed() メソッドがこのスクリプトにより定義されている場合、fieldName によって値が変更されるとそのメソッドが呼び出されます。formevents パラメーターは、XML ファイルの長い CDATA セクションにすることができます。
- <FormSection> 内には、以下のようなタグがある場合があります。
  - <FormSectionNames> - Form の場合と同様です。
  - <FormItem> - Form の場合と同様ですが、当該 Form 内に同じような名前の FormItem がある場合は、その FormItem から暗黙的に継承します。
  - <parameter> - 以下のパラメーターが認識されます。
    - title - title によって、FormSection が異なった表示になります。
    - description - FormSection の詳細な記述。
    - initiallyExpanded - このパラメーターが false に設定されていると、FormSection は最初は展開されなくなります。デフォルト値は true です。
- <FormItem> 内には、以下のようなタグがある場合があります。
  - <Values> - droplist/dropedit 値リストを指定します。
  - <LocalizedValues> - <Values> の値を、変換ファイルでルックアップされるキーにマップします。
  - <parameter> - 以下の表に示すように、FormItem のパラメーターを定義します。
- <ListItem> リストの項目を定義します。
- <LocalizedValues> 内には、以下のようなタグがある場合があります。

– <Item> - マップ内の 1 つの項目。

- <Item> 内には、以下のようなタグがある場合があります。
  – <Key> - キー。
  – <Value> - 値。
- <LocalizedValues> の例は、以下のとおりです。

```
<LocalizedValues>
 <Item>
  <Key>After every database operation</Key>
  <Value>Localized.After.every.database.operation</Value>
 </Item>
 <Item>
  <Key>After every database operation (Including Select)</Key>
  <Value>Localized.After.every.database.operation.Including.Select</Value>
 </Item>
 <Item>
  <Key>Manual</Key>
  <Value>Localized.Manual</Value>
 </Item>
 <Item>
  <Key>On Connector close</Key>
  <Value>Localized.On.Connector.close</Value>
 </Item>
</LocalizedValues>
```

XML 変換の考慮事項:

ここに記載されているコード例を参照して、XML 変換の考慮事項を理解することができます。

変換された値をプロパティー・ファイルから XML ファイルにマージする代わりに、Form 内には新しいタグ <TranslationFile> があります。構成エディターの使用時に、この変換ファイルの正しいローカル・バージョンが読み込まれて、値が使用されます。

ファイル・コネクターの例: ファイル・コネクターの tdi.xml ファイルには、Form 用にこのタグが含まれています。

```
<TranslationFile>NLS/idi_conn_filesys</TranslationFile>
```

この XML ファイルとすべての NLS/idi_conn_filesys.properties ファイルを JAR ファイルにパッケージします。

パラメーター定義:

ここに記載されているパラメーター定義を使用することができます。

以下に、認識されているパラメーターのうち、FormItem で使用可能なパラメーターを示します。

表 68. FormItem パラメーター

キーワード	説明
label	フォームの左側の列に表示されるラベル (LDAP URL など)。
description	パラメーターのツール説明
default	パラメーターのデフォルト値。デフォルト値はコンポーネント構成自体 (tdi.xml ファイル) の中で指定することをお勧めします。このデフォルト値は、ユーザーが構成エディターを使用してコンポーネントの構成を表示/変更するときにのみ設定されます。

表 68. *FormItem* パラメーター *(続き)*

キーワード	説明
script script2	このパラメーターを指定すると、入力フィールドの右側にボタンが追加されます。このボタンをクリックすると、指定された JavaScript 関数が実行されます。 *Script2* を指定すると、1 番目のボタンの右側に 2 番目のボタンが追加されます。
scriptLabel scriptLabel2	ボタン・テキスト
scriptHelp scriptHelp2	スクリプト・ボタンのツール説明。
syntax	パラメーターの構文を指定します。これは、値を表現するために使用される UI コントロールの選択にも影響します。詳しくは、「構文」を参照してください。
reflect	指定されている場合は、バインディングはこのメソッドを使用してパラメーター値を取得/設定します。バインディングでは、この値に応じて「get」または「set」が値の前に付加されます (例: getName または setName を呼び出すには Name と指定します)。これは、構成オブジェクトがパラメーター値の取得/設定時に特定のロジックを実行する場合にのみ使用されます。コンポーネント構成には get/set プリミティブのみが含まれているため、コンポーネント開発者がこのパラメーターを必要とすることはほとんどありません。

動的値:

フォーム定義を操作する場合は、ここに記載されている動的値のリストを参照してください。

values リストには、静的値と動的値を含めることができます。ドロップダウン・リストにデータを取り込むため、動的値は実行時に拡張されて配列に追加されます。

表 69. *動的値*

値	説明
@ASSEMBLYLINES@	すべての既知の AssemblyLine を配列に追加します。
@CONNECTORS@	すべての既知のコネクターを配列に追加します。
@PARSERS@	すべての既知のパーサーを配列に追加します。
@FUNCTIONS@	すべての既知の関数コンポーネントを配列に追加します。
@ATTRS@	入力マップからのすべての属性を追加します。

構文:

FormItem の構文パラメーターでは、指定された任意の値を使用することができます。

表 70. *構文のパラメーター値*

値	説明
String	これはデフォルトの構文です。テキスト入力用の 1 行のテキスト・フィールドが作成されます。

表 70. 構文のパラメーター値 (続き)

値	説明
パスワード	テキスト入力用パスワード・フィールドが作成されます。ユーザーがパスワード・ストアを構成している場合は、FormUI により構成オブジェクトに値は挿入されませんが、プロパティー参照が挿入される点に注意してください。実際の値はパスワード・ストアに保管されます。 スクリプトまたは Java コードを使用してこのパラメーターを変更する場合は、必ず *BaseConfiguration.setParameter()* ではなく *BaseConfiguration.setProtectedParameter()* を呼び出してください。 setProtectedParameter では、新規プロパティーがまだ作成されていない場合に自動的に新規プロパティーが作成されます。パスワード・ストアが構成されていない場合は、setProtectedParameter は代わりに setParameter を呼び出します。
Boolean	true/false 値を指定するためのチェック・ボックスが作成されます。
Droplist Dropedit	values パラメーターの値からなるドロップダウン。Dropedit は編集可能なバージョンであり、カスタム値を指定するためのテキスト・フィールドもユーザーに提供されます。特殊値については、「動的値」を参照してください。
TextArea	複数行テキスト入力のためのテキスト域コントロールを作成します。
Script	スクリプトを呼び出すボタンを作成します。
Static	表示専用テキスト・ラベルを作成します (TextArea の場合と同じですが読み取り専用)。
EditorWindow	この構文を使用すると、フォームがタブ形式ペインになります。左端のタブには editorwindow 以外のパラメーターが示され、各 editorwindow パラメーターにはエディター入力コントロールを備えた固有のタブが作成されます。入力用の完全な表示領域が必要な場合 (スクリプトなど) に使用します。

表 70. 構文のパラメーター値 (続き)

値	説明

Component	この値を使用すると、複雑な入力メカニズムが必要な場合、または UI の制御を強化する場合に独自の UI コンポーネントを指定できます。component キーワードにフォームに挿入する Java クラス名を指定します。
	
	syntax:component
	component:pub.test.CustomUI
	
	Version 7.x – Eclipse SWT コンポーネント
	
	実行時に FormWidget2 によりクラスがインスタンス化されます。このクラスは、SWT Control サブクラス (コンポジットの子になれるもの) でなければなりません。また、クラスにはコンストラクターも含まれている必要があります。次に例を示します。
	`package pub.test;`
	
	`import org.eclipse.swt.widgets.Composite;`
	`import com.ibm.tdi.eclipse.widgets.FormWidget2;`
	`import com.ibm.di.config.interfaces.BaseConfiguration;`
	
	`public class CustomUI extends Composite {`
	`/*`
	`* form - the FormWidget2 object`
	`* parent – The Composite in which this control is placed`
	`* config - the config object being edited`
	`* paramname - the parameter of config being edited`
	`*/`
	`public CustomUI(FormWidget2 form, Composite parent, BaseConfiguration config,`
	`String paramname) {`
	`super(parent, 0);`
	`}`
	
	`}`
	
	このコンポーネントは、GridLayout を使用してコンポジットに配置されます。カスタム UI オブジェクトの GridData を設定しないでください。これは、この設定が、カスタム・クラス作成後にフォーム・ウィジェットによって作成されるためです。

フォームのスクリプト:

フォーム定義では、スクリプト関数の呼び出しを追加できます。

これらの関数は、フォームのスクリプト・エンジンで実行されます。フォームのスクリプト・エンジンには、以下の事前定義オブジェクトがあります。

- **form** - このフォームを管理している com.ibm.tdi.eclipse.widget.FormWidget2 インスタンスを表します。
- **config** - このフォームの操作対象である構成オブジェクトへのハンドル (例: コネクターの場合は接続構成、パーサーの場合はパーサー構成など)。
- **attributeName** - FormItem の名前。
- **system** - com.ibm.di.function.UserFunctions クラスのインスタンス。

例

ここに記載されているパスを使用して、サンプルにアクセスすることができます。

構成エディターで IBM Security Directory Integrator のコンポーネントを参照し、適切なサンプルを探します。zip/jar ツール (例: winzip、unzip) を使用してコンポーネントの jar ファイル (*TDI_install_dir*/jars/components サブディレクトリー) から、「tdi.xml」ファイルを解凍します。

また、このパッケージの examples/connector_java フォルダーには、ディレクトリー・コネクターの「tdi.xml」ファイルが含まれています。

コネクターの再接続ルールの定義

ここに記載されているルールを使用して、コネクターの再接続ルールの定義を理解することができます。

IBM Security Directory Integrator の再接続機能を活用するために、コネクターの .xml ファイルに、接続が中断された場合のコネクターの応答を調整するルールを組み込むことができます。これらのルールは、IBM Security Directory Integrator サーバーの再接続エンジンの組み込みルールに追加されます。[1]

特定のコネクター用のルールは、以下のように、「connectors」のセクションに「connectorConfig」サブセクションの兄弟として表示されます。

```
<Connector name="CustomConnector">
 <Configuration>
   ... various configuration options ...
 </Configuration>
 <Reconnect>
  <ReconnectRules>
   <Rule>
    <parameter name="exceptionClass">java.sql.SQLException</parameter>
    <parameter name="exceptionMessageRegExp">^I/O.*</parameter>
    <parameter name="action">reconnect</parameter>
   </Rule>
   <Rule>
    <parameter name="exceptionClass">java.sql.SQLException</parameter>
    <parameter name="exceptionMessageRegExp">^Io.*</parameter>
    <parameter name="action">reconnect</parameter>
   </Rule>
   ... more rules go here ...
  </ReconnectRules>
 </Reconnect>
```

各ルールには、以下のようなパラメーターがあります。

```
exceptionClass: fully qualified name of the Java class of the exception
exceptionMessageRegExp: regular expression in Java syntax
action: error or reconnect
```

「exceptionClass」パラメーターおよび「exceptionMessageRegExp」パラメーターはオプションです。指定されていない場合、そのルールは、すべての例外クラスおよびすべての例外メッセージにそれぞれ一致します。

「exceptionMessageRegExp」で使用される正規表現の構文について詳しくは、java.util.regex.Pattern クラスの JavaDoc を参照してください (http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html)。

---

1. IBM Security Directory Integrator の以前のバージョンとの互換性を保持するため、以前のバージョンの動作をエミュレートする 2 つの組み込みルールが用意されています。コンポーネントに対して再接続が完全にオフになっている場合を除いて、java.io.IOException タイプおよび javax.naming.CommunicationException タイプのすべての例外に対して、再接続が試行されます。

例

```
<Connector name="ibmdi.ReconnectTest">
 <Configuration>
  <parameter name="connectorType">com.ibm.di.connector.ReconnectTestConnector</parameter>
   </Configuration>
 <Reconnect>
  <ReconnectRules>
   <Rule>
    <parameter name="exceptionClass">java.io.IOException</parameter>
    <parameter name="exceptionMessageRegExp">.*file not found.*</parameter>
    <parameter name="action">error</parameter>
   </Rule>
   <Rule>
    <parameter name="action">reconnect</parameter>
   </Rule>
  </ReconnectRules>
 </Reconnect>
</Connector>
```

コネクターのパッケージ化とデプロイ

コネクターをパッケージ化してデプロイする場合は、以下の点について注意する必要があります。

コネクターのソース・コードのコンパイルと tdi.xml ファイルの作成が完了すると、コネクターをパッケージ化してデプロイできます。

それには、jar ファイル (通常はコネクターと同じ名前にする) を作成して、以下のファイルを組み込みます。

1. コネクターのクラス・ファイル (複数のこともある)
2. tdi.xml ファイル (jar ファイルのルート)。翻訳済みのファイルを使用している場合は、標準 Java 国際化対応スキーマを使用してそれらのファイルも組み込みます。例えば、ドイツ語には CustomConnector_de.properties、フランス語には CustomConnector_fr.properties、ブラジル・ポルトガル語には CustomConnector_pt_BR.properties となります。

新規コネクターの jar ファイルを作成した後は、その jar ファイルを、IBM Security Directory Integrator をインストールしたシステムの jars/connectors フォルダーに配置するのみです。システムを次回に始動した時点で、新規コネクターも自動的にロードされ、使用可能になります。

関数コンポーネントの開発

このセクションに記載されている情報を参照して、関数コンポーネントを開発することができます。

関数コンポーネント (FC) をインプリメントする際も、コネクターを開発するパターンに従います。実際には、関数コンポーネントのインプリメントの方が簡単です。AssemblyLine のワークフローとの依存関係が少ないためです。

関数コンポーネントの Java ソース・コードのインプリメント

ここに記載されているメソッドを使用して、関数コンポーネントの Java ソース・コードを実装することができます。

コネクターの基礎クラスと同じように、インターフェースをインプリメントする *com.ibm.di.fc.FunctionInterface* 抽象クラスおよび *com.ibm.di.fc.Function* 抽象クラスがあります (この 2 つのクラスの Java ソースは、このパッケージの `fc` フォルダーに入っています)。

通常は、*com.ibm.di.fc.Function* クラスをサブクラス化することで関数コンポーネントをインプリメントします。多くの場合にインプリメントする必要のある重要なメソッドは、以下のとおりです。

public void initialize (Object obj

初期化コードをここに指定します。関数コンポーネントのパラメーターの読み取りや、リソースの割り振りなどを行います。関数コンポーネントを AssemblyLine 内に配置した場合、AssemblyLine は開始時に 1 回 *initialize(...)* メソッドを呼び出します。

関数コンポーネントを作成し、プログラマチックに使用する場合は、関数コンポーネント・オブジェクトを構築してパラメーターを設定した直後、かつ *perform(...)* メソッドを呼び出す前に、*initialize(...)* メソッドを呼び出す必要があります。

public Object perform (Object obj)

perform(...) メソッドは、作成する関数コンポーネントのビジネス・ロジックを実際にインプリメントするものです。コネクターの場合は、さまざまなコネクター・モードがあり、モードごとに別々のメソッド (*getNextEntry()*、*findEntry()* など) をインプリメントするのに対して、関数コンポーネントが実行する内容は *perform(...)* メソッドの中にインプリメントします。

perform(...) メソッドに関する一般的な規則は、データを入力として受け取り、その入力に基づいて出力データを生成するというものです。その他の前提事項はありません。後述しますが、使用する関数コンポーネントは、項目オブジェクトを処理することが必須というわけではありません。

関数コンポーネントを AssemblyLine 内に配置した場合、AssemblyLine は反復ごとに関数コンポーネントの *perform(...)* メソッドを呼び出します。AssemblyLine のコンテキストでは、*perform(...)* メソッドは入力パラメーターとして項目オブジェクト (出力属性マッピング・プロセスにより作成された項目オブジェクトで、*conn* 項目とも呼ばれる) を渡されます。項目オブジェクトも戻されます。AssemblyLine は戻された項目オブジェクトを入力属性マッピング・プロセスに供給し、その結果が AssemblyLine の作業項目に適用されます。言い換えると、戻された項目は、入力マップの *conn* 項目です。作成した関数コンポーネントを AssemblyLine 内に配置できるようにするには、このような「項目を入力し、項目を出力する」という動作をサポートする必要があります。

また、入力として項目以外のオブジェクトを受け取り、出力として項目以外のオブジェクトを戻すように、*perform(...)* メソッドをコーディングすることもできます。これを利用すると、プログラマチックに関数コンポーネントを作成して呼び出すプロセスが容易になります。このメソッドを使用した場合、属性マッピングは行われません。

public void terminate ()

関数コンポーネントの *terminate(...)* メソッドは、AssemblyLine がサイクルを完了した後、処理を終了する前に呼び出されます。ここには、終結処理コ

ードを指定します。つまり、接続を解放したり、*initialize(...)* メソッド中やその後の処理中に作成したり、ソースを解放したりします。

関数コンポーネントをプログラマチックに使用する場合は、関数コンポーネント・インスタンスの使用を終えた後で、*terminate(...)* メソッドを呼び出す必要があります。

関数コンポーネントのソース・コードの作成

関数コンポーネントのソース・コードを作成する際に、インストールされている IBM Security Directory Integrator の `jars` フォルダー内の jar ファイルを `CLASSPATH` に組み込むことができます。

少なくとも、`miserver.jar` および `miconfig.jar` を組み込む必要があります。

注: 作成した Java コードを IBM Security Directory Integrator と統合する際には、IBM Security Directory Integrator を構成する既存のコンポーネントのコレクション (特に *jars* ディレクトリー内のファイル) に注意してください。作成したコードが依存しているユーザーのライブラリー・コンポーネントの 1 つが、IBM Security Directory Integrator システムに含まれる 1 つ以上のコンポーネントとオーバーラップしたり、それらのコンポーネントを破壊したりする場合は、実行中にローダーの問題が発生したことが考えられます。

関数コンポーネントの GUI 構成フォームのインプリメント

関数コンポーネントの GUI は、コネクターと同様の方法でインプリメントされます。「tdi.xml」ファイルを使用して、コネクターの場合と同じ構文を使用して関数コンポーネント構成フォームを記述します。

関数コンポーネントのパッケージ化とデプロイ

関数コンポーネントのパッケージ化とデプロイを行う場合は、ここに記載されている情報を参照してください。

関数コンポーネントをパッケージ化してデプロイする方法は、コネクターをパッケージ化してデプロイする場合と同じです。

以下のファイルを組み込んだ jar ファイルが必要です。

1. 関数コンポーネントのクラス・ファイル (複数のこともある)
2. `tdi.xml` ファイル。jar ファイルのルートに配置します (オプションとして、異なる言語をサポートする場合は、`MyFunction_??.properties` ファイルを含めます)

新しい関数コンポーネントの jar ファイルを作成した後は、その jar ファイルを、IBM Security Directory Integrator をインストールしたシステムの `jars/functions` フォルダーに配置するのみです。次回に IBM Security Directory Integrator が始動された時点で、新しい関数コンポーネントが自動的にロードされ、使用できる状態になります。

パーサーの開発

ここに記載されている情報を使用することで、パーサーを開発できるようになります。

パーサーはトランスポート・コネクターとともに使用され、コネクターのバイト・ストリームを介して送受信される内容を解釈または生成します。ただし、非常に特殊なフォーマットで表されたデータの解析が必要となる場合があります。そのため、独自のパーサーをインプリメントする必要があります。

パーサーの Java ソース・コードのインプリメント

ここに記載されている情報とメソッドを使用して、パーサーの Java ソース・コードを実装することができます。

すべての IBM Security Directory Integrator パーサーは、`com.ibm.di.parser.ParserInterface` Java インターフェースをインプリメントします。 このインターフェースは、インプリメントする、すべてのパーサーに共通なメソッドを数多く提供します。通常、作成するパーサーには、インターフェースが提供するすべてのメソッドをインプリメントする必要はありませんが、メソッドのサブセットのみインプリメントが必要です。この目的のため、 `ParserInterface` をインプリメントする `com.ibm.di.parser.ParserImpl` 抽象クラスを使用できます。`ParserImpl` クラスには、コア・パーサー機能が組み込まれており、独自のパーサーをインプリメントするときにそれをサブクラス化できます。

パーサーには 2 種類あります。1 つはストリームから読み取って項目を戻すパーサー、もう 1 つは項目を取得してストリームに書き込むパーサーです。

パーサーを作成したら、それを構成する必要があります。これには、入出力ストリームの設定、および一部の追加パラメーターの構成 (必要に応じて) が含まれます。通常、これはホスティング・コンポーネント (例えば、コネクター) によって行われます。このジョブが完了したら、次に、今後必要となるリソースが割り振られるパーサーの初期化およびその他の初期化を実行します。通常、ホスティング・コンポーネントはパーサーの初期化メソッドの構成および呼び出しの両方を処理します。次は、パーサーの使用において最も重要である、項目の書き込みまたは読み取りです。ここで、実際の解析が行われます。最後に、コネクターが項目の転送を完了したら、パーサーを終了する必要があります。パーサーを終了するときに、パーサーは入出力ストリームを閉じて、前段階で使用していたリソースを解放します。

パーサーのインプリメンテーションの実例は、IBM Security Directory Integrator に組み込まれている `ExampleParser.java` パーサーを参照してください。多くの場合にインプリメントする必要のある重要なメソッドは、以下のとおりです。

public void setInputStream(InputStream is)

ここでは、パーサーの入力ストリーム属性を設定します。このメソッドはオーバーロードされ、引数として「String」および「Reader」も使用できます。抽象クラス `com.ibm.di.parser.ParserImpl` は、パーサーの入力ストリームを設定する 3 つのメソッド用のインプリメンテーションを提供します。`com.ibm.di.parser.ParserImpl` をサブクラス化すると、これらのメソッドの一部を指定変更したり、スーパー・クラスのデフォルトのインプリメ

ンテーションのままにしておくことができます。ただし、インターフェースをインプリメントする場合は、インターフェース全体にインプリメンテーションを提供する必要があります。

入力ストリームを開いたら、開いたものがそれを閉じる必要があることに注意してください。これは、通常 closeParser() メソッドで行われます。com.ibm.di.parser.ParserImpl 抽象クラスはパーサーの入出力ストリームを閉じるためのデフォルトのインプリメンテーションを提供します。

public void setOutputStream(OutputStream os)

ここでは、パーサーの出力ストリーム属性を設定します。出力ストリームは引数として渡されます。このメソッドには、引数として「Writer」オブジェクトを使用するオーバーロード・バージョンがあります。「setInputStream(...)」同様、抽象クラス com.ibm.di.parser.ParserImpl はパーサーの出力ストリームを設定する両方のメソッド用のインプリメンテーションを提供します。com.ibm.di.parser.ParserImpl をサブクラス化すると、これらのメソッドの一部を指定変更したり、スーパー・クラスのデフォルトのインプリメンテーションのままにしておくことができます。ただし、インターフェースをインプリメントする場合は、インターフェース全体にインプリメンテーションを提供する必要があります。

出力ストリームを開いたら、開いたものがそれを閉じる必要があることに注意してください。これは、通常 closeParser() メソッドで行われます。com.ibm.di.parser.ParserImpl 抽象クラスはパーサーの入出力ストリームを閉じるためのデフォルトのインプリメンテーションを提供します。

public void initParser()

ここには初期化コードを入力します。このメソッドは通常、ホスティング・コンポーネント (例えば、コネクター) によって呼び出され、パーサーを初期化します。

任意のパラメーターや追加のチェーニングされたパーサーの設定、および今後必要となる可能性のあるリソースを割り振ることができます。このメソッドは、インプリメントされたすべてのパーサーに必要なわけではありません。

パラメーターへのアクセス方法の例を以下に示します。このコードのセットは、組み込まれた例「ExampleParser.java」の一部です。

```
str = getParam("attributeName");
    if (str != null && str.trim().length() != 0) {
     attrName = str;
    }
```

このメソッドは、入出力ストリームの設定が行われた後に呼び出されることに注意してください。

public Entry readEntry()

このメソッドは、コネクターのインプリメントに使用される「getNextEntry()」メソッドと同様です。これは、現在の入力ストリームから次の項目を戻します。入力ストリームがなくなった場合、 NULL 値が戻されることが期待されています。ここで、実際の解析が行われます。

入力ストリームが正しく初期化されていることを確認してください。入力ストリームを設定するには、setInputStream(...) メソッドを使用します。リーダー・オブジェクトを取得するには、getReader() メソッドを使用します。

通常、入力ストリームは、ホスティング・コンポーネント (例えば、コネクター) によって初期化されます。

public void writeEntry(Entry entry)
現在の出力ストリームを使用して項目を書き込むには、このメソッドを使用します。書き込まれる項目は、引数として渡されます。ここで、項目のデータを解析して、出力ストリームに正しい形式でこのデータを書き込む必要があります。

ライターを取得するには、getWriter() メソッドを使用します。このメソッドは「java.io.BufferedWriter」を戻します。

通常、出力ストリームは、ホスティング・コンポーネント (例えば、コネクター) によって初期化されます。

public void flush()
このメソッドは通常、一部のホスティング・コンポーネントによって呼び出され、現在の出力ストリームに任意のメモリー内データをフラッシュします。 そのため、データがすべて書き込まれ、このメソッドの呼び出しのメモリー内に何も残っていないことを確認してください。

public void closeParser()
このメソッドは通常、ホスティング・コンポーネント (例えば、コネクター) によって呼び出され、パーサーのリソースをクローズおよび解放します。今後使用することがないと思われるリソースをクローズおよび解放するために、およびパーサーがこれ以上呼び出されない場合に、このメソッドを使用します。ほとんどの場合、これは入出力ストリームですが、追加リソースが使用されていた場合は、そのリソースも解放されます。

com.ibm.di.parser.ParserImpl 抽象クラスは、このメソッド用のインプリメンテーションを提供しますが、インターフェースをインプリメントする場合は、自分でそれを記述する必要があります。

パーサーのソース・コードの作成

パーサーのソース・コードを作成する際に、インストールされている IBM Security Directory Integrator の 「jars」フォルダー内の jar ファイルを CLASSPATH に組み込むことができます。

少なくとも「miserver.jar」および「miconfig.jar」を組み込む必要があります。ソース・コードは、Java 7.0.4 以前でコンパイルする必要があることに注意してください。

注: Java コードを IBM Security Directory Integrator と統合する際には、IBM Security Directory Integrator を構成する既存のコンポーネントのコレクション (特に jars ディレクトリー内のファイル) に注意してください。作成したコードが依存しているユーザーのライブラリー・コンポーネントの 1 つが、IBM Security Directory Integrator システムに含まれる 1 つ以上のコンポーネントとオーバーラップしたり、それらのコンポーネントを破壊したりする場合は、実行中にローダーの問題が

発生したことが考えられます。つまり、IBM Security Directory Integrator に同梱されているサード・パーティー・ライブラリーと競合する可能性があることに注意する必要があります。これにより、 IBM Security Directory Integrator が同じライブラリーの特定のバージョンを使用する場合、ライブラリーの別のバージョンを使用するパーサーを作成することを避ける必要があります。

パーサーの GUI 構成フォームのインプリメント

「tdi.xml」ファイルを使用して、コネクターの場合と同じ構文を使用してパーサー構成フォームを記述することができます。

パーサーの GUI は、コネクターと同様の方法でインプリメントされます。

パーサーのパッケージ化とデプロイ

ここに記載されている情報を参照して、パーサーのパッケージ化とデプロイを行うことができます。

パーサーをパッケージ化してデプロイする方法は、コネクターをパッケージ化してデプロイする場合と同じです。

以下のファイルを組み込んだ jar ファイルが必要です。

1. パーサーのクラス・ファイル
2. jar ファイルのルートに配置される「tdi.xml」ファイル。このファイルは、パーサーの登録に使用されることに注意してください。このファイルがない場合、コードはロードされず、IBM Security Directory Integrator 構成エディターでパーサーを使用できません (ただし、スクリプトから呼び出すことはできます)。

新しいパーサーの jar ファイルを作成した後は、その jar ファイルを、IBM Security Directory Integrator インストールの「jars」フォルダーに配置するのみです。独自のフォルダーを作成して jar ファイルをそこに入れることができますが、通常、パーサーが保存されるのは「jars/parsers」フォルダーです。次回に IBM Security Directory Integrator が始動された時点で、新しいパーサーが自動的にロードされ、使用できる状態になります。

追加ロガーの作成

IBM Security Directory Integrator では、IBM Security Directory Integrator と Log4J 間のハードワイヤード・リンクを切断し、デフォルトで Log4J を起動する構成可能なロギング・クラスに置き換えることができます。

従来、IBM Security Directory Integrator へのログインはサーバー・ベースまたはタスク (AssemblyLine) ベースの Appender によって行われ、その場合、実際のログ出力は Apache Log4J フレームワークに依存しています。 Log4J には、さまざまな出力チャネルと出力フォーマットが用意されていますが、IBM Security Directory Integrator ユーザーが必要とする、重複した追加の出力チャネルを持つ他のロギング・ユーティリティーも用意されています。これらの多くはオープン・ソース・ライブラリーであり、法律上の理由から IBM Security Directory Integrator にバンドルされていません。これらのサード・パーティー製ロギング・ユーティリティーを組み込むために、IBM Security Directory Integrator ロギング・コンポーネントは、IBM Security Directory Integrator と実際のロギング・インプリメンテーション

(LogInterface インプリメンテーションと呼ばれる) 間のプロキシーとして動作するようにモデリングされています。IBM Security Directory Integrator には、以下のように Log4J、JLOG、および java.util.log のインプリメンテーションが付随しています。

表 71. ロギング・ユーティリティー用の LogInterface のインプリメンテーション

ロギング・ユーティリティー	ハンドラー/Appender
Apache Log4J	カテゴリー・ベース構成 *) ConsoleAppender CustomAppender DailyRollingFileAppender FileAppender NTEventLog FileRollerAppender SystemLogAppender SyslogAppender
標準 Java ロギング (java.util.log)	FileHandler
JLOG	カテゴリー・ベース構成 *) FileHandler

*) カテゴリー・ベース構成とは、ロガーの構成がロギング・ユーティリティー固有の外部ファイル (Log4J 用の log4j.properties など) 内で定義されていることを意味しています。

IBM Security Directory Integrator 構成ファイルの構造には、com.ibm.di.config.interfaces.LogConfigItem オブジェクトを保持する最上位フォルダーが含まれています。このフォルダーには、すべての jar および zip ファイル内で「tdi.xml」ファイルをスキャンする、IBM Security Directory Integrator クラス・ローダー (IDILoader) によってデータが取り込まれます。「tdi.xml」ファイルでは、適切なセクションを組み込むことによって、IBM Security Directory Integrator ユーザーおよび CE に対して利用可能になる新規ロガーを定義します。この方法で定義された各ロガーには、IBM Security Directory Integrator CE がカスタム・ロガー・パラメーターの構成のためにユーザーに対して表示するフォーム定義も含まれています。

com.ibm.di.log.Log クラスは、これらの新規ログ・コンポーネントを 1 つ以上使用するように設計されています。各ログ・コンポーネントによって、com.ibm.di.log.LogInterface がインプリメントされます。このクラスでは、LogInterface メソッドがロギング・ユーティリティー・フレームワークの対応するメソッドにマップされます。このログ・コンポーネントには、インスタンス化されるときにバックエンド・ロガーを正しく構成できるように LogConfigItem オブジェクトが指定されます。

ロギング・インターフェースの理解

ここにリストされるロギング・インターフェース・ファイルとオブジェクトを参照します。

IBM Security Directory Integrator ロギング・インターフェースは、以下のファイルおよびオブジェクトで構成されています。

表 72. ロギング・インターフェースのファイルとオブジェクト

オブジェクト	説明
com.ibm.di.config.interfaces.LogConfigItem	これは、定義済み LogInterface インプリメンテーション用の構成オブジェクトです。
com.ibm.di.log.LogInterface	これは、サード・パーティーのロギング・ユーティリティーにアクセスするためにロガーによってインプリメントされるインターフェースです。
com.ibm.di.server.Log	これは、IBM Security Directory Integrator コンポーネントがログ・オブジェクトの作成に使用するユーティリティー・クラスです。
<workdir>/logging.categories	カテゴリーを LogInterface のクラス名にマップするためのオプション・ファイル。このファイルは、デフォルトでは存在しません。
com.ibm.di.log.TDILog4j	Log4J 用の LogInterface のインプリメンテーション。
<installdir>/etc/log4j.properties	IBM Security Directory Integrator の主要コンポーネント・カテゴリー (サーバー、CE、および構成ドライバー) 用の Log4j 構成ファイル。
<installdir>/etc/global.properties	ログ・アクティビティーをグローバルに有効または無効にするためにプロパティーが定義されます。false の場合、IBM Security Directory Integrator ログ・クラス経由のログ呼び出しは廃棄されます。 プロパティー名は、`com.ibm.di.logging.enabled` です。

IBM Security Directory Integrator コンポーネントがロガーを取得する場合は、使用する実際のロギング・ユーティリティーおよび出力を決定するためのカテゴリーを使用して、`com.ibm.di.server.Log` クラスのインスタンスを作成します。これは、以前のバージョンとの互換性を保持するためです。

また、このログ・クラスは、カテゴリーと LogInterface クラス間にマッピングがあるかどうかを調べるために `logging.categories` ファイルを参照します。デフォルトでは、このファイル (デフォルトでは存在しない) に特定のマッピングはなく、ログ・クラスは TDILog4J インプリメンテーションにフォールバックします。

ログ・インターフェースの構成

ここに記載されている情報とロギング・ユーティリティーを使用して、ログ・インターフェースの構成を実行することができます。

ロギング・インターフェースは、作業ディレクトリーで `logging.categories` と呼ばれるファイルを探し、デフォルトの TDILog4J `LogInterface` インプリメンテーションの使用をオーバーライドします。このファイルの各行には、LogInterface インプリメンテーションの Java クラス名を提供する値を持つカテゴリー名が含まれています。

例を示します。

```
*:com.ibm.di.log.TDILog4j
AssemblyLine.AssemblyLines/myAL:com.ibm.di.log.TDILogJUL
```

上の例では、TDILogJUL を使用する myAL という名前の AssemblyLine 以外のすべての AssemblyLine は、TDILog4j フレームワークを使用してログを記録します。現在使用できるロギング・ユーティリティーは以下のとおりです。

- Log4j – `com.ibm.di.log.TDILog4J`
- Java Util Logging – `com.ibm.di.log.TDILogJUL`
- JLOG – `com.ibm.di.log.TDIJLog`

ロガーの外部構成

ロガーの外部構成を実行する場合は、以下の説明を参照してください。

ロギング・ユーティリティーは一般に外部構成ファイルを使用して構成されます。例えば、Log4J は、名前 (カテゴリー) が特定の出力タイプ/フォーマット (例えば、ファイル出力、XML フォーマット) にマップされているプロパティー・スタイル・ファイルを使用します。ユーザーがカテゴリー名を提供してロガー・インスタンスを要求すると、ロギング・ユーティリティーはその構成ファイルを参照して、そのカテゴリー名を解決します。

各 LogInterface インプリメンテーションは、そのロガーの正しい構成を提供するためにプロパティー・ファイルを要求することがあります。具体的には、デフォルトの TDILog4j インプリメンテーションが別のロギング・ユーティリティーに置き換えられた場合、新規のロギング・ユーティリティーは、`etc/log4j.properties` ファイルのリリース・バージョンで定義されているカテゴリーに対応するロガーを提供する必要があります。

ロガーの内部構成

ロガーの内部構成を実行する場合は、以下の説明を参照してください。

IBM Security Directory Integrator の主要なコンポーネントは、カテゴリー名を使用してロガー (サーバー、CE、構成ドライバーなど) を取得し、その外部構成を使用して各ロガーに関する詳細を提供します。ただし、ユーザーは構成エディターを使用して追加のロガーを指定することができます。ユーザーは、AssemblyLine レベルまたはサーバー・レベル、あるいはその両方のレベルでロガーを追加できます。ユーザーがこのようにしてロガーを追加すると、そのロガーに関する詳細が IBM Security Directory Integrator 構成ファイルに格納されます。これにより、ロギング機能およびその外部構成ファイルによることなく、IBM Security Directory Integrator によってロガーのインスタンス化が処理されます。従来は、IBM Security Directory Integrator が定義済みの Log4J ロガーのリストを持っており、ユーザーがそこからロガーを選択していました (815 ページの表 72を参照)。現行バージョンの IBM Security Directory Integrator では、ハードコーディングされたロガーのリストは、システム・ネーム・スペース (ロガー・フォルダー) に外部化されています。

ロガーの構成は、構成ファイル内の最上位フォルダーである **Logger** に格納されます。システム全体で使用できるロガーは、システム・ネーム・スペースで定義されます。このネーム・スペースは、IBM Security Directory Integrator の開始時に `tdi.xml` ファイルから構築されます。

installdir/jars ディレクトリー (およびその他のカスタム・ローダー・ディレクトリー) 内の jar/zip ファイルに含まれている tdi.xml ファイルが、IBM Security Directory Integrator ローダーによってシステム・ネーム・スペースに追加されます。ロギング・コンポーネントは、このファイル内の 2 つの独立したセクションで定義されます。

ロガー・セクションでは、特定のロギング・ユーティリティー (例えば、log4j、ファイル・ロガー) に特定のロガーを指定する LogInterface インプリメンテーションとパラメーターが定義されます。このセクションには、ユーザーに構成可能パラメーターを提供するために使用されるフォーム定義を指す第 2 のパラメーターも含まれています。各ロガーに固有のこのセクションでは、追加のパラメーターが指定されている場合があります。

```
<Logger name="ibmdi.JavaUtilLoggingFile">
 <parameter name="categoryBased">false</parameter>
 <parameter name="com.ibm.di.formName">ibmdi.JavaUtilLoggingFile</parameter>
 <parameter name="com.ibm.di.log.interface">com.ibm.di.log.TDILogJUL</parameter>
 <parameter name="handler">FileHandler</parameter>
</Logger>
```

このフォーム・セクションでは、ロガーの構成可能パラメーターが定義されています。

```
<Form name="ibmdi.JavaUtilLoggingFile">
 <FormItemNames>
   <ListItem>fileName</ListItem>
  <ListItem>formatter</ListItem>
  ... other parameters...
 </FormItemNames>
 <FormItem name="fileName">
   <parameter name="description">The pattern for the log file name</parameter>
   <parameter name="label">File Name</parameter>
    <parameter name="Required">true</parameter>
   <parameter name="script">selectFile</parameter>
   <parameter name="scriptLabel">Select...</parameter>
    <parameter name="scripthelp">Choose the file name to use</parameter>
  </FormItem>
  <FormItem name="formatter">
  <Values>
   <ListItem>Simple</ListItem>
    <ListItem>XML</ListItem>
  </Values>
   <parameter name="description">Choose a SimpleFormatter or a XMLFormatter</parameter>
   <parameter name="label">Formatter</parameter>
   <parameter name="syntax">droplist</parameter>
  </FormItem>
  .... Other FormItem definitions
</Form>
```

tdi.xml ファイルの全体としての構文の骨組みは、以下の例のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<MetamergeConfig>
 <Folder name="Loggers">
  <Logger name="...">
  </Logger>
  </Folder>
  <Folder name="Forms">
  <Form name="...">
  </Form>
 </Folder>
</MetamergeConfig>
```

ロガー API

ここに記載されている情報とリンクを使用することで、新規のユーティリティーを追加することができます。

IBM Security Directory Integrator に新規のロギング・ユーティリティーを追加する場合は、LogInterface インプリメンテーションが作成され、適切なセクションを含む tdi.xml ファイルが提供されます (816 ページの『ロガーの内部構成』を参照)。このインプリメンテーションは、新規ロガーをブートストラップするための静的メソッドも備えている必要があります。

com.ibm.di.server.Log:

ロギング・アクティビティーをグローバルに管理するための 2 つのメソッドが定義されている IBM Security Directory Integrator の主要ロギング・クラスを使用することができます。

このアクティビティーの初期設定は、com.ibm.di.logging.enabled という名前のプロパティーによって定義されています。このクラスをバイパスするロギングは影響を受けません。

```
/**
 * Disables or enables TDI logging. All loggers are affected by this setting.
 */
 public static void setLoggingEnabled(boolean enabled);

/**
 * Returns whether TDI logging is active or disabled.
 */
 public static boolean isLoggingEnabled(boolean enabled);
```

ロギングが最初からオフになっている場合でも (例えば、プロパティーが false に設定されている場合)、ローダーとメインプログラムの初期化中に IBM Security Directory Integrator によって数行がログに書き込まれることがあります。ロギングを完全に除去したい場合は、ロギング・ユーティリティーの構成ファイル (例えば、log4j.properties や jlog.properties など) を変更する必要があります。

com.ibm.di.log.LogInterface:

ここに記載されているコード例を使用して、com.ibm.di.log.LogInterface を処理することができます。

```
package com.ibm.di.log;


/**
 * Defines an Interface to new Loggers.
 * Any Logger we use must adhere to this interface.
 * The Implementation must provide a public constructor with no arguments.
 * After construction either the setCategory() or the addAppender() method will be called.
 */
public interface LogInterface {

 public final static String TYPE = "type";
 public final static String NAME = "name";
 public final static String CONFIG_INSTANCE = "configInstance";
 public final static String TIME = "time";

 /**
  * Set the category for this Logger.
  * This method specifies a category, to allow a category based configuration.
  *
  * @param category The category to use.
  */
 public void setCategory(String category) throws Exception;

 /**
  * Add an Appender to the Logger using the given config. Appender is the
  * org.apache.log4j name, java.util.logging would call it a Handler. 5 月
  * throw an Exception if the config does not make sense.<br/>
  * The params Map may contain these keys to help set up the Appender:
  *
```

```
 * - TYPE: "AssemblyLine", "EventHandler" or ""
 * - NAME: A String with the name of component
 * - CONFIG_INSTANCE: a RSInterface
 * - TIME: a String with the time in milliseconds
 *
 *
 * @param config
 *            The LogConfigItem.
 * @param params
 *            Extra information that may be useful/
 */
public void addAppender(LogConfigItem config, Map params) throws Exception;

/**
 * Log a message with level debug.
 * @param str The string to be logged
 */

public void debug( String str );

/**
 * Log a message with level info.
 * @param str The string to be logged
 */

public void info( String str );

/**
 * Log a message with level warning.
 * @param str The string to be logged
 */

public void warn( String str );

/**
 * Log a message with level error.
 * @param str The string to be logged
 */

public void error( String str );

/**
 * Log a message with level error, and an additional Throwable.
 * @param str The string to be logged
 * @param error The Throwable to be logged
 */

public void error( String str, Throwable error );

/**
 * Log a message with level fatal.
 * @param str The string to be logged
 */

public void fatal ( String str);

/**
 * Log a message with level fatal, and an additional Throwable.
 * @param str The string to be logged
 * @param error The Throwable to be logged
 */

public void fatal ( String str, Throwable error );

/**
 * Log a message with the specified level.
 * @param level The level to use when logging.
 * @param str The string to be logged
 */

public void log (String level, String str );

/**
 * Check if a debug message would be logged.
 * @return true if a debug message might be logged
 */
public boolean isDebugEnabled ();


/**
```

```
  * Free up all resources this logger uses.
  * The logger will not be called anymore.
  */
 public void close();
}
```

com.ibm.di.server.Log:

ここに記載されている情報を使用することで、com.ibm.di.server.Log を理解できるようになります。

ロギング機能を必要とする IBM Security Directory Integrator コンポーネントは、com.ibm.di.server.Log クラスを使用してロガーを取得する必要があります。このクラスは、クライアントと実際のロギング・インプリメンテーションの間のプロキシーです。 IBM Security Directory Integrator コンポーネントは、そのコードにロギングを追加する場合は、定義済みの既存のロギング・カテゴリーを再利用するのか、新規のカテゴリーを作成するのか、それとも独自の LogConfigItem 構成オブジェクトを維持するのかを決定する必要があります。いずれの場合でも、実際のロギングを行うには、最終的にはログ・オブジェクトを使用する必要があります。

コンストラクターと構成:

一般に、ロガーを構成するには 2 つのコンストラクターのいずれか一方を使用します。ロガーが作成されると、setPrefix() メソッドを使用して、すべての発信メッセージに接頭部として付加されるストリングを設定することができます。

この接頭部ストリングは翻訳されません。

カテゴリー名は、ロガーを構成するために使用されます。カテゴリー名は、使用中のロギング・ユーティリティーのプロパティー・ファイルで定義されています。resourceFileName は、メッセージの翻訳時に使用される NLS テーブルを含む、(com.ibm.di.server.ResourceHash によってロードされる) リソースの名前です。

```
/**
 * Create a log object using category as both the name of the resource
 * file and the logger category name (configuration).
 */
public Log(String category);

/**
 * Create a log object using separate values for category and resource name.
 */
public Log(String category, String resourceFileName);


/**
 * Sets a prefix to be prefixed to all messages
 *
 * @param  prefix
 */
public void setPrefix (String prefix);
```

単純なログ・メソッド:

以下に示す単純なログ・メソッドを使用することができます。

便利なように、さまざまなレベル向けのロギング・メソッドのセットが提供されています。

```
public void log<level>(String msg)
```

ここで、**<level>** は以下のロギング・レベルです。

- fine
- debug
- info
- warn
- error
- fatal

これらのメソッドは、メッセージを (すべての接頭部を含めて) そのままの形でロガーにログとして書き込みます。

```
log.setPrefix("PRE");
log.loginfo("Hello");

>> PRE Hello
```

NLS ログ・メソッド:

ログ・メッセージを翻訳する場合は、以下のいずれかのメソッドを使用する必要があります。

- fine(String resid)
- debug(String resid)
- info(String resid)
- warn(String resid)
- error(String resid)
- fatal(String resid)

ここで、**resid** は、ログ・オブジェクトに関連付けられているリソース・ファイル内のリソース ID です。リソース ID の翻訳が見つからなかった場合、そのリソース ID はそのままの形でログ出力で使用されます。これらの各メソッドは、翻訳されたストリング内の置換マーカーの値を提供するための 4 つのバリアントとして用意されています。置換値と一緒にこれらのバリアントの 1 つを使用すると、Log クラスは提供したパラメーターを使用して、ストリングに対して `java.util.text.MessageFormat` を使用します。3 つのメソッドを使用して、1 つ、2 つ、または任意の数の置換値を提供します。

```
public void debug(String res)
public void debug(String res, Object param)
public void debug(String res, Object param1, Object param2)
public void debug(String res, Object[] params)
```

Throwable オブジェクトを使用して独自のエラー・メッセージをログに書き込みたい場合は、`error(String resid, Throwable error)` メソッドを使用します。

翻訳されたメッセージを持つ例外を生成したい場合は、`exception(String resid) throws Exception` メソッドを使用できます。このメソッドは、翻訳されたストリングをそのメッセージとして持つ汎用の java.lang.Exception オブジェクトをスローします。

ストリングを翻訳してそれを他の場所で使用したい場合は、`getString()` メソッドを使用して、ログ・オブジェクトに関連付けられているリソース・ファイルから翻訳済みのストリングを取得することができます。

例:

以下に示す非常に単純は、プロパティー・ファイルに基づいて NLS メッセージを標準の IBM Security Directory Integrator サーバー・ログ (miserver) に書き込む方法を示しています。

この例では、「XXX.properties」がコードと一緒にパッケージされていることを想定しています。

「XXX.properties」の内容:

```
my.resource.id= Hello World
```

```
import com.ibm.di.server.Log;

public class XXX() {
 public XXX() {
  this.log = new Log("miserver";, "XXX");
  this.log.info("my.resource.id");
 }
}
```

上の場合は、ログ・メッセージ「Hello World」が IBM Security Directory Integrator サーバー・ログに書き込まれます。

関連情報

697 ページの『付録 D. サーバー API』,
260 ページの『ログ・コネクター』

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。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。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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 Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 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。

IBM の将来の方向性および指針に関するすべての記述は、予告なく変更または撤回される場合があります。これらは目標および目的を提示するものにすぎません。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

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。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラット

フォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。© Copyright IBM Corp. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、PostScript は、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

IT Infrastructure Library は英国 Office of Government Commerce の一部である the Central Computer and Telecommunications Agency の登録商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Centrino、Intel Centrino ロゴ、Celeron、Xeon、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社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。

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

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。

ITIL は英国 The Minister for the Cabinet Office の登録商標および共同体登録商標であって、米国特許商標庁にて登録されています。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社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。

Cell Broadband Engine は、Sony Computer Entertainment, Inc.の米国およびその他の国における商標であり、同社の許諾を受けて使用しています。

Linear Tape-Open、LTO、LTO ロゴ、Ultrium、および Ultrium ロゴは、HP、IBM Corp. および Quantum の米国およびその他の国における商標です。

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]

アクセシビリティー xvii
アダプター
  柔軟なコネクター 777
  スイッチ/ケース・コンポーネント 777
  フロー内のイテレーター 778
  AL コンポーネント 777
アダプター AL 780
アダプター構成 781
アダプター・コード
  エラー処理 784
アルゴリズムの例
  REST サーバー API 754
暗号化 41
暗号化の構成 82
一般構成
  ALE IDOC コネクター 619
イテレーター・モード 498
  読み取り 310
  getnext メソッド 380
移動されたオブジェクト
  Active Directory 11
インストール・ロケーション
  tdi.xml ファイルのフォーマット 797
インターフェースの変更
  値の置換 241, 242
  値の追加 241
  属性値の削除 242
  属性から値を除去する 241
  属性の除去 241, 242
インプリメント、コード
  アダプター 780
埋め込み
  使用可能化 203
  使用不可化 203
エラー処理
  アダプター・コード 784
  AL コネクター 784
エラー・フロー 14
大きな Active Directory グループ
  処理 255
オブジェクト
  新しい属性オブジェクトの作成 678
オブジェクト *(続き)*
  基準オブジェクト 682
  検索オブジェクト 682
  属性オブジェクト 677, 678
  属性オブジェクトへの値の追加 678
  属性の値のスキャン 678
  の例 678
  COMProxy オブジェクト 684
  Java Native Interface 684
  Javadoc 677
オブジェクト ID 11
オブジェクト構造サービス 397

[カ行]

外部システムの構成
  XML スキーマ定義 332
外部の Derby 349
拡張 XML 483
カスタム・シリアライザー 533
カスタム・デシリアライザー 533
仮想リスト・ビュー制御 250
仮名ファイル 567
関数コンポーネント
  ソース・コード 809
  AssemblyLine 501, 807
  Castor Java から XML への変換 502
  Castor XML To Java 505
  castor XML から Java 504
  GUI 構成フォーム 809
  Java ソース・コード 808
管理ソフトウェア・システム 368
既知の互換性の問題
  com.ibm.di.config.base.BaseConfigurationImpl クラス 751
  com.ibm.di.config.interfaces.ExternalPropertiesDelegator クラス 749
行リーダー・パーサー
  構成 461
  単一属性 461
  の例 461
  パラメーター 461
許可 41, 42, 84
組み込み Derby サーバー 353
組み込み再接続ルール 205
組み込みルール 251
グループ・コネクター 416
グローバル一意識別子 (GUID) 721
グローバル項目インスタンス 679
検索オブジェクト
  オペランド 682
  の例 683
研修 xviii
コード例
  フォーム定義 798
構成
  カテゴリー・ベース 262
  サーバー API プロパティー 702
  ターゲット・システム 576
  パラメーター 15, 18, 26, 564
  編集 714
  ユーザー・レジストリー 702
  リモート・ クライアント 702
  ログ・インターフェース 815
  AssemblyLine コネクター 18
  IBM Security Access Manager v2 コネクター 169
  TDILog4J 815
構成インスタンス 721
  構成の初期設定 707
  サーバー API の同期化 707
  始動 706
  停止 706
構成エディター 370
構成の編集 715
  構成ファイル・パス 718
  ソリューション名 718
構文のパラメーター値
  フォーム定義 803
項目オブジェクト 679
項目のフィルター処理 70
考慮事項 217
固定レコード・パーサー
  構成 445
  パラメーター 445
コネクター 3
  再利用 7
  パーサー 792
  パッケージ化とデプロイ 807
  ロギング 794
  Active Directory 変更検出 9
  Assembly Line 7
  Axis Easy Web Service サーバー 22
  QRadar 284
    検査 292
    セットアップ 288
    パラメーター 285
    マッピング 290
    ログ・ソース 291
  SOAP 22
コネクター操作
  HTTP/SOAP 411

コネクターの Java ソース・コード
インプリメンテーション 785
コネクターの開発
インプリメンテーション 785
開発 785
コネクターの再接続ルール
定義 806
コネクターのソース・コード
作成 795
コネクターの認証
単純 14
ディレクトリー 14
無名 14
SASL 14
コネクター・インターフェース 3, 785
リンク 3
list 3
コネクター・インターフェース・オブジェクト
メソッド 679
コネクター・スキーマ
入力属性マップ 383
コマンド行コネクター
引用符の使用 51
オペレーティング・システム 50
構成 51
ネイティブ・エンコード 出力 50
モード 50
例 52
コンストラクターと構成
NLS テーブル 820
resourceFileName 820
コンポーネント
IdML のオープン関数コンポーネント 645
コンポーネント定義
syntax 797
コンポーネントの可用性
コンポーネントのバージョン 表 65
コンポーネント・コンボ・ボックス 65
「入力マップ」 からの接続 65
「入力マップ」からの接続 65

[サ行]

サーバー API 710, 714, 721
移植 740
一時構成インスタンス 718
イベント通知 728
インプリメンテーション 741
カスタム・メソッドの起動 731
既知の互換性の問題 749, 751
既知の問題 749
構成 702
構成インスタンス 705, 706, 707
サーバー API (続き)
構成インスタンス ID 707
構成更新 719
構成の編集 714, 718, 743
構成のロック 715
構成ファイル 707
構造 699
サーバー API イベント 719
サーバー API オブジェクト 735
サーバー情報 730
サーバー・シャットダウン・イベント 728
システム・キュー 719, 721
シリアライズ可能オブジェクト 741
シリアライズ可能クラス 742
セキュリティー 700
セキュリティー・レジストリー 731
トゥームストーン・マネージャー 721, 722
認証メカニズム 743
バージョンの確認 743
編集のためのロード 715
リモート・セッション 704
リモート・セッションの作成 704, 705
ローカル・セッション 704
ログ・ファイル 729
AssemblyLine 710, 711, 712, 714
AssemblyLine トゥームストーン 725
changes 744
IBM Security Directory Integrator 697
IBM Security Directory Integrator 構成 714
IBM Security Directory Integrator プロパティー 725
JAR ファイル 731
JMS プロバイダー 719
JMX インターフェース 697
JMX 通知 735
JMX の例 736
JMX レイヤー 719, 733, 734
MBean 735
RMI 731
RMI リモート・オブジェクト 741
サーバー API イベント通知
登録 726
サーバー API インターフェース
リモート 699
ローカル 699
サーバー API クライアント・コード 743
サーバー側
サーバー API getServerInfo メソッド 64
-v command-line オプション 64
サーバー通知コネクター
暗号化 315
イテレーター・モード 314, 318
構成 316
スキーマ 318
トラストストア 315
認証 315, 316
パラメーター 316
ユーザー名とパスワードによる認証 315
addonly モード 314, 318
SSL 証明書 316
SSL 認証 315, 316
サーバー・モード 407
再接続エンジン 205
再接続機能 251
削除されたオブジェクト 10
作成、新規パイプ 278
サポート資料
コンポーネント開発 785
資産統合スイート 641, 642, 643, 645, 649, 651, 652, 655, 657, 659, 660, 666
コンポーネント 645
トラブルシューティング 672
CI/関係 669
IdML スイート 669
IT レジストリー 644, 669
システム・オブジェクト
Javadoc 683
システム・キュー
アクセス 721
メッセージの取得 721
メッセージの入力 721
システム・キュー・コネクター
暗号化 348
イテレーター・モード 346
許可 348, 349
構成 347
認証 348
パラメーター 347
ユーザー名とパスワードによる認証 348
addonly モード 346
SSL 証明書ベースの認証 348
システム・ストア 349
システム・ストア・コネクター 349
構成 351
使用法 353
パラメーター 351
柔軟なコネクター
初期化 777
出力マップ 153, 155, 157, 158, 159
照会スキーマ
非標準モード 783
AL コネクター 783

状況オブジェクト
エラー・コード 683
状況の表示
AL アダプター 783
AL コネクター 783
証明書の管理 228
シリアライズ可能クラス
Java RMI エンジン 742
シリアライゼーションの問題 534
新規コンポーネント
アダプターを使用した 775
作成 775
Java 775
シンプル Tpae IF コネクター 323
アーキテクチャー 320
イテレーター・モード 320, 326
エラー処理 332
エラー・フック 332
エンタープライズ・サービスの使用 325
オブジェクト構造サービス 324
外部システムの構成 332
共通基盤 321
更新モード 320, 330
構成 333
コネクターの使用 324
コネクター・モード 326
削除モード 320, 330
スキーマ 331
統合フレームワーク 320, 321
の例 337
パラメーター 333
ルックアップ・モード 320, 331
AddOnly モード 329
addonly モード 320
AddOnly モードでの MBO パラメーター 329
HTTP 324
MBO パラメーター 326
MIF オブジェクト構造 323
XML 324
スキーマ
更新 190
出力マップ 270
デルタ 190
入力マップ 270
addonly 190
delete 190
input 25, 34, 126
output 25, 34, 126
スキーマの比較
ネイティブ・モード 365
IdML モード 365
スクリプト関数コンポーネント
オブジェクト 520
構成 520
スクリプト関数コンポーネント *(続き)*
スクリプト・ペイン 520
パラメーター 520
スクリプト記述関数コンポーネント
スクリプト 519
スクリプト言語 675
Java スクリプト 675
JavaScript 675
スクリプトの例
作成 105
添付ファイルの抽出 105
スクリプト・コネクター
イテレーター・モード 310
構成 313
定義済みスクリプト・オブジェクト 310
の例 314
パラメーター 313
functions 311
スクリプト・パーサー
オブジェクト 462
結果オブジェクト 462
構成 464
項目オブジェクト 462
コネクター・オブジェクト 463
スキーマ 464
の例 465
パーサー・オブジェクト 463
パラメーター 464
メソッド 463
functions 461, 463
inp オブジェクト 463
out オブジェクト 463
スケジューラー 687
ステートメントのカスタマイズ
更新 196
delete 196
insert 196
select 196
ストアード・プロシージャーの呼び出し 203
ストリーム・ベースのコネクター 423
正規表現 205
セキュリティー
SSL ベースの認証 700
設計時の 命名規則 665
前提条件
アカウント名 414
新規ユーザーの作成 414
ソース・コード
作成 809
操作
変更要求 433
属性 153, 155, 157, 158, 159
属性のマッピング
呼び出し側 AL からアダプターへ 782
ルックアップ・モード 782
AssemblyLine コネクター 19

［タ行］

ターゲット・システム
AS400 システム 578
Cygwin システム 577
JRE 578
Linux システム 577
SSH 577
UNIX システム 577
Windows システム 576
タイマー・コネクター
イテレーター・モード 384
構成 384
パラメーター 384
タイム・スタンプ 202
タスク・オブジェクト
クラスのインスタンス 684
単純 XML 483
単純 XML パーサー 476
構成 478
パラメーター 478
文字のエンコード 478
例 479
単純なパーサー
構成 465
項目の書き込み 465
項目の読み取り 465
パラメーター 465
単純なログ・メソッド
ロギング・メソッド 820
追加の JDBC 関数 200
追加ロガー
作成 813
データ表現フォーマット
IdML ブックのフォーマット 363
データ表現モード
ネイティブ・モード 364, 365
IdML モード 364
データベースのレプリカ
切り替え 68
データベース・コネクター
構成 52
JDBC コネクター 52
データ・ クレンジング関数コンポーネント
スキーマ 659
データ・クレンジング関数コンポーネント
構成 659
パラメーター 659
CDM 属性 658

データ・ソース・スキーマ
出力スキーマ 356, 375
入力スキーマ 356, 375
ディレクトリー構造
トラバース 121
デシリアライゼーションの問題 534
デプロイ済み資産コネクター
addonly モード 55
デルタ関数コンポーネント
構成 555
削除モード 554
追加モード 554
デルタ検出 556
の例 557
パラメーター 555
ルックアップ・モード 554
ロジックの適用 556
デルタ・タグ・サポート 355
デルタ・モード
デルタ・データ 783
deltaSavvy 783
findEntry 783
トゥームストーン・マネージャー 721
動作モード
イテレーター 55
addonly 55
delete 55
lookup 55
動的値
フォーム定義 803
導入済み資産コネクター
アーキテクチャー 54
イテレーター・モード 55
構成 55
削除モード 55
サポートされる 動作モード 54
動作モード
イテレーター 55
addonly 55
delete 55
lookup 55
の例 56
ルックアップ・モード 55
JDBC 53
ドライバー実装 191
トラブルシューティング xviii
トランスポート・コネクター 406

[ナ行]

認証 41
ホスト・ベース 700
ユーザー名とパスワードによる認証 316
ユーザー名/パスワード 700
JAAS 700
認証 *(続き)*
LDAP 700
認証メカニズム
LDAP 743
の例
com.ibm.di.log.LogInterface 818
NLS メッセージ 822
tdi.xml 806
xxx.properties 822

[ハ行]

パーサー 425
インプリメント 810
開発 810
作成 810
パッケージ化とデプロイ 813
パッケージ化とデプロイメント 810
CBE パーサー 428
DSMLv1 パーサー 431
jar ファイル 813
パーサー関数コンポーネント
書き込みモード 519
構成 518
パラメーター 518
読み取りモード 519
AssemblyLine コンポーネント 518
パーサーの GUI 構成フォーム
インプリメンテーション 813
tdi.xml ファイル 813
パーサーの Java ソース・コード
インプリメンテーション 810
パーサーのインターフェース 810
Java インターフェース 810
パーサーの使用
コネクター 792
パーサーのソース・コード
作成 812
CLASSPATH 812
jars フォルダー 812
パスワード同期プラグイン
コネクター 693
パスワード・ストア 693
AssemblyLine 693
Password Synchronizer 693
パッケージ化とデプロイ
関数コンポーネント 809
jar ファイル 807
jar フォルダー 813
tdi.xml ファイル 807
パラメーター置換
API 201
パラメーター定義
フォーム定義 802
凡例
サポートされるモードの欄 7
ファイル管理コネクター
空ディレクトリーの作成 125
空のファイルの作成 125
構成 127
使用法 121
シンボリック・リンク 123
スキーマ 126
ディレクトリーの強制削除 125
ディレクトリーの更新 124
トラバース 121
ファイルの強制削除 125
ファイルの更新 124
リンク基準 123
例 128
AssemblyLine 121
fullPath 123
functions 121
ファイル転送関数コンポーネント
アーキテクチャー 574
拡張オプション 581
拡張ソース・オプション 579
拡張ターゲット・オプション 580
構成 576, 578
出力スキーマ 574
ソース・オプション 578
ターゲット・オプション 580
入力スキーマ 574
パラメーター 578
方向 575
FTP プロトコル 573
RXA ツールキット 574
RXA でサポートされるプロトコル 573
ファイル・コネクター
構成 119
パーサー 119
パラメーター 119
ファイル・システム・コネクター 119
フォーム記述
コード例 798
フォーム定義 798
フォーム定義
動的値 803
formitem パラメーター 802
フォーム入力コネクター
イテレーター・モード 128
構成 129
パラメーター 129
AssemblyLine 128, 129
フォームのスクリプト
スクリプト関数 805
フォームの定義エレメント
FormItems 799
FormSection 799
フォーム/構成のバインディング
構成オブジェクト 798

複素数タイプ・ジェネレーター関数コンポーネント
構成 553
パラメーター 553
Axis ライブラリー 552
input 554
JAR ファイル 552
output 554
WSDL 552
プロパティー・コネクター
構成 281
構成ファイル 280
の例 283
パラメーター 281
ファイル・フォーマット 283
プロパティー・ストア 280
.property ファイル 282
プロパティー・ストア 282
変更検出 11
オフラインだったときの結果の場合 12
ページングされた結果の場合 12
変更検出コネクター 13
変更の追跡 9
便利なオブジェクト 197

[マ行]

マッピング、アダプター
コネクター・モード 780
JavaScript 780
メールボックス・コネクター
「一致なしの場合」フック 273
イテレーター 269, 272
更新 269, 272
更新モード 274
構成 275
削除モード 273
スキーマ 270
パラメーター 275
メール・フォルダー 272
リンク基準 273
ルックアップ・モード 273
addonly 269, 272
addonly モード 274
delete 269, 272
lookup 269, 272
POP3 プロバイダー 274
メイン・オブジェクト
method 682
メッセージ交換パターン
サポート対象 32
メッセージの JMS/非 JMS コンシューマー 207
メモリーの問題
ページ・サイズ 250
メモリー・キュー ・コネクター
アクセス 278
構成 278
パラメーター 278
メモリー・キュー関数コンポーネント
構成エディター 528
システム・オブジェクト 529
メモリー・バッファー・パイプ 528
getfunction 529
メモリー・キューのコンポーネント
ウォーターマーク 277
ページ 277
メモリー・キュー・コネクター 276
ワークフロー 277
メモリー・ストリーム・コネクター
イテレーター・モード 279
構成 280
受動モード 279
パラメーター 280
addonly モード 279
モード
イテレーター 148
更新 148
addonly 148
delete 148
lookup 148
文字セット
ユニコード 416
文字のエンコード
変換 424
問題判別 xviii

[ラ行]

ラップ Soap 関数コンポーネント
構成 535
使用法 536
パラメーター 535
input 536
output 536
リスナー・トランスポート・チャネル
プッシュ・チャネル 770
プル・チャネル 770
リモート・アクセス
JMX レイヤー 734
リモート・コマンド行関数コンポーネント 557
構成 558
使用法 561
パラメーター 558
リモート・コマンド行関数コンポーネント 560, 561
input 560
output 561
リンク基準 197, 210, 781
リンク基準の構成
更新モード 413
削除モード 413
ルックアップ・モード 413
ルックアップのスキップ
更新 148, 195
更新モード 243, 254
削除モード 243, 254
delete 148, 195
例外 14
ローカル・アクセス
JMX レイヤー 734
ローカル・クライアント・セッション 61
ローカル・サーバー・セッション 62
ロガー API
LogInterface 818
ロガーの外部
構成 816
LogInterface 816
TDILog4j 816
ロガーの内部
構成 816
Log4J 816
LogInterface 816
ロギング・インターフェース
オブジェクト 815
ファイル 815
ロギング・ユーティリティー用の LogInterface のインプリメンテーション 813
ログ構成画面
コンソール Appender 263
日次ローリング・ファイル Appender 263
ファイル Appender 264
Apache log4J ロガー 261, 263, 264, 265, 266
customappender 263
FileRollerAppender 265
Java Util ロガー 261, 267
カテゴリー・ベース構成 268
ファイル・ハンドラー 268
JLOG ロガー 268
カテゴリー・ベース構成 269
FileHandler 269
JLog ロガー 261
NTEventLog 265
SyslogAppender 266
SystemLogAppender 266
ログ・インターフェース
構成 815
ログ・コネクター
構成 260, 262
スキーマ 260
パラメーター 260
ロギング機能 260

ログ・コネクター *(続き)*
ログ構成画面 261, 263, 264, 265, 266, 267, 268, 269
AssemblyLine 260
ログ・ファイル
アクセス 729

［ワ行］

ワークフロー
ウォーターマーク 277

A

ACL 106
Active Directory 9, 10, 13
ADCD 9
addonly モード 498
追加の指定変更フック 381
AL コネクター 781
AL の操作
デフォルトの操作 776
AssemblyLine 776
ALE IDOC コネクター
構成 617
パラメーター 617
IDOC Client 構成 618
ALE Intermediate Document (IDOC) コネクター 615
Apache Axis2 ライブラリー 543
Apache Log4J フレームワーク 813
Apache Xerces 476
API 202
APPC 571
APPC/MVS 呼び出し 567
Assembly Line 3
AssemblyLine 65, 710, 718
アクセス 710
アダプターのパブリッシュ 778
構成インスタンス 711
構成パネル 779
コネクター 19
始動 712
消費 778
新規アダプターの作成 779
新規コンポーネント 775
停止 714
パラメーター 22
IBM Security Access Manager v2 コネクター 169
assemblyLine 794
AssemblyLine 関数コンポーネント
構成 513
サーバー API 513
パラメーター 513
AssemblyLine 関数コンポーネント *(続き)*
パラメーター変換 517
ハンドラー・オブジェクト 514
AssemblyLine コネクター
イテレーター・モード 17
AssemblyLine コネクター・オブジェクト
コネクター・インターフェース 677
AssemblyLine シーケンス
構成 691
構成エディター 691
パラメーター 691
AssemblyLine トゥームストーン
カスタム・メッセージの追加 725
AssemblyLine の開始
コンポーネント・シミュレーション 714
手動モード 712
リスナー 712
AssemblyLine の反復 211
AssemblyLine フロー・チャート 204
コンポーネント 695
AssemblyLine ベースの Appender 813
AssemblyLine モード
イテレーター 137
addonly 137
call/reply 137
lookup 137
Axis Easy Web Service サーバー
Axis1 29
Axis2 29
Axis Easy Web Service サーバー・コネクター
スキーマ 25
操作 28
Axis EasyInvokeSoapWS 関数コンポーネント
構成 549
パラメーター 549
Axis ライブラリー 549
SOAP 549
WSDL 549
Axis Java To Soap 関数コンポーネント
出力 542
input 542
Axis Java から Soap への変換関数コンポーネント
構成 530
パラメーター 530
input 531
output 531
SOAP ヘッダー 530
SOAP 本体 530
SOAP メッセージ 531
WSDL 530
XML の戻りタイプ 531
Axis Soap To Java 関数コンポーネント
構成 541
パラメーター 541
SOAP 541
SOAP メッセージ 542
WSDL 541
Axis2 Easy Web Service サーバー
構成 39
スキーマ 34
WSDL 39
Axis2 Web サービス・クライアント関数コンポーネント
メッセージ交換パターン 544
Axis2 Web サービス・サーバー
Connector 29
SOAP 29
WSDL 29
Axis2 Web サービス・サーバー・コネクター 33
暗号化 41
許可 42
使用法 31
認証 41
Axis2 WS クライアント関数コンポーネント 543, 544
SOAP ヘッダー 545
Axis2 WS サーバー・コネクター
WSDL の生成 34
Axis2WSClientFC
構成 548
スキーマ 545
パラメーター 548
AxisEasyInvokeSoapWS 関数コンポーネント
認証 550
exception 551
HTTP 550
HTTP Web サービス 551
input 551
output 551
SOAP 551
AxisJavaToSoap 533
AxisSoapToJava 533

B

Base64 エンコード 440
Booster Pack 250
BytesMessage 207

C

call/reply モード
input 211
output 211

Castor Java To XML 関数コンポーネント
構成 503
パラメーター 503
Castor XML To Java
構成 505
項目モード 506
パラメーター 505
非項目モード 506
CastorJavaToXML 関数コンポーネント
項目モード 503
非項目モード 503
CBE 134
CBE (Common Base Event)
AssemblyLine 133
CBE 関数コンポーネント 425
イベントの発行 524
構成 522
パラメーター 522
API 525
CBE パーサー 522
CBE ログ 521
CBE ログ XML 523
CEI サーバー 524
Common Base Event 522
Common Event Infrastructure 522
CBE 属性
input 426
output 426
CBE パーサー
構成 428
出力マップ 425
使用 425
属性 426
パラメーター 428
CCMDB コネクター
アーキテクチャー 43
イテレーター・モード 48
更新モード 48
構成 49
構成管理 42
削除モード 48
スキーマの比較 45
ネイティブ・モード 44, 45
の例 49
ルックアップ・モード 48
addonly モード 47
IdML モード 44, 45
mode 47
CDM 372
明示クラス・タイプ 363
明示属性名 363
CDM コンポーネント
インターフェース 641, 642
関係 641, 643
クラス 641, 642
属性 641
CDM コンポーネント *(続き)*
命名 643
命名 規則 643
命名および識別 641
ID 643
CDM フォーマット
識別可能なデータ 364
changelog 173
Common Data Model 641
ComplexTypesGenerator 関数コンポーネント
トラブルシューティング 554
Component Suite のインストール 583
ComProxy オブジェクト
コード例 684
com.ibm.di.log.LogInterface
の例 818
com.ibm.di.server.Log
ログ・オブジェクト 820
jlog.properties 818
log4j.properties 818
LogConfigItem 構成オブジェクト 820
CSV パーサー
スキーマ 431

D

DB2 349
DOM メカニズム 481
Domino Server コネクター
接続 80
デプロイメント 80
Domino Server タスク 72
Domino のセットアップ
タスク 72
特権 72
Domino ユーザー・コネクター
許可 84
構成 80
動作モード
イテレーター 84, 85, 90
更新 88
ルックアップ・モード 85
addonly 84
addonly モード 85
delete 84
lookup 84
認証 83
の例 96
パラメーター・マイグレーション 82
ユーザー属性 93
ユーザーの作成 78
ユーザー文書の 取得 78
Unix/Linux 95
Domino/Lotus Notes コネクター
インストール 57
DSML ライブラリー 440
DSMLv1 パーサー
構成 431
パラメーター 431
例 431
XML 文書 431
DSMLv2 SOAP コネクター
イテレーター 110
拡張操作 111
更新 110
構成 112
デルタ 110
AddOnly 110
CallReply 110
delete 110
HTTP 109
lookup 110
SOAPAction ヘッダー 112
DSMLv2 SOAP サーバー・コネクター
拡張操作 114
構成 114
AssemblyLine 113
HTTP 113
DSMLv2 XML スキーマ 440
DSMLv2 パーサー 440
エラー応答 439
オプション属性 440
拡張応答 439
拡張要求 439
クライアント・モード 432, 444
結果記述 441
結果コード 441
検索応答 434
検索要求 434, 435, 444
構成 442
サーバー・モード 432, 443
削除応答 436
削除要求 436
ストリング以外の属性 440
操作 433
追加要求 435
データ・フロー・ルール 441
ディレクトリー構造情報 432
認証応答 438
認証要求 438
バイナリー属性 440
パラメーター 442
比較応答 437
比較要求 437
複数の属性の変更 441
変更応答 433
例 443
addrequest 443
modifyDN 応答 437
modifyDN 要求 436
XML 文書 432

E

EIF コネクター 117
  イテレーター・モード 118
  エラー時に 中断 118
  構成 118
  終了 タイムアウト 118
  詳細ログアウト 118
  スキーマ 118
  スキーマ・ファイル 118
  統合 117
  addonly モード 116, 118
  EIF 構成ファイル 118
  eifSchemaFile 118
  GetNext タイムアウト 118
  Netcool 118
  Netcool/OMNIbus 116

F

FormItem
  構文パラメーター 803
FTP オブジェクト
  スクリプト可能オブジェクト 681
  の例 681
FTP クライアント・コネクター
  イテレーター・モード 130
  構成 132
  トランスポート・コネクター 130
  パラメーター 132
  文字のエンコード 132
  addonly モード 130
  RFC959 130
  SSL サポート 130
FTPS 130

G

Generic Log Adapter 293
Generic Log Adapter コネクター 133
  アダプター構成ファイル 134, 136
  構成 135
  構成ファイル 134
  スキーマ 137
  パラメーター 135
  FileOutputter 134
  TDIoutputter 134
GUI 構成フォーム
  インプリメンテーション 809
  インプリメント 795
  コネクター 795
  コンポーネント/フォームの関連性 798
  フォーム記述 798

H

HTTP 407, 537
HTTP クライアント・コネクター
  イテレーター・モード 138
  構成 140
  特殊属性 138
  パラメーター 140
  文字のエンコード 140
  ルックアップ・モード 138
  例 141
  AssemblyLine モード 137
  HTTP セッション 137
  SSL プロトコル 137
HTTP サーバー・コネクター
  イテレーター・モード 142
  構成 145
  コネクター構造 143
  コネクター・クライアント認証 143
  サーバー・モード 142
  スキーマ 146
  属性 146
  チャンク転送エンコード 144
  パラメーター 145
  AssemblyLine 143
HTTP チャンク 144
HTTP 転送プロトコル 544
HTTP トランスポート層 33
HTTP パーサー
  一般ヘッダーのフィールド 452
  エンティティー・ヘッダーのフィールド 452
  応答ヘッダーのフィールド 455
  構成 450
  スキーマ 450, 452, 453, 455
  パラメーター 450
  文字セット 456
  要求ヘッダーのフィールド 453
  encoding 456
  HTTP Cookie 457
  HTTP クライアント・コネクター 449
  HTTP サーバー・コネクター 449

I

IBM
  ソフトウェア・サポート xviii
  Support Assistant xviii
IBM Domino/Lotus Notes コネクター
  サーバー側 64
  セッション・タイプ 57
  「classes」 フォルダー 65
IBM Domino/Lotus コネクター
  インストール後の構成 61
  ネイティブ API 呼び出しのスレッド化 63
IBM Domino/Lotus コネクター *(続き)*
  ローカル・クライアント・セッション 61
  ローカル・サーバー・ セッション 62
  IIOP セッション 62
IBM SDI 6.0 から現行バージョン・サーバーへの移植 740
IBM SDI 内部フォーマット
  の例 498
IBM Security Access Manager
  イテレーター・モード 159
  更新モード 155
  削除モード 157
  ルックアップ・モード 158
  addonly モード 153
IBM Security Access Manager v2 コネクター
  構成 169
  属性マップ
    グループ・エンティティー 169
    ユーザー・エンティティー 169
  トラブルシューティング 171
  パラメーター 169
  ユーザーとグループ 166
  Registry Direct API 167
IBM Security Access Manager コネクター
  機能 147
  モード 148
  利点 147
IBM security access manager コネクター
  更新モード 152
  構成 148
  コネクター入力属性 160
  コネクターの構成 151
  使用法 152
  セキュア通信の構成 149
  ドメイン 165
  トラブルシューティング 160
  入力属性 160
  ポリシー 163
  ルックアップのスキップ 148
  addonly モード 152
  Domain 属性 165
  group 162
  Group 属性 162
  JRTE 148
  Policy 属性 163
  SSL の構成 149
  SSO Credentials 165
  SSO Credentials 属性 165
  SSO Resource Group 166
  SSO Resource Group 属性 166
  SSO Resource 属性 166
  SSO リソース 166
  user 160

IBM Security Directory Integrator
アダプター 776
項目スキーマ 610
の例 806
IBM Security Directory Integrator アダプター
AL コネクター 779
AL の操作 779
IBM Security Directory Integrator スケジューラー
キープアライブ 688
構成 687, 688
構成エディター 687
タイマー 687
パラメーター 687
IBM Security Directory Integrator バージョン 7.2
インストール・ディレクトリー 1
の例 1
IBM Security Directory Integrator 変更ログ・コネクター 173
構成 175
属性のマージ動作 173
パラメーター 175
分散 TDS 174
IBM Tivoli Netcool 116
IBM websphere MQ 217, 223
IBMJS 675
IdML CI および関係コネクター
構成 アイテム 653
スキーマ 655
CDM メタデータ 653
IdML Ci および関係コネクター
構成 656
パラメーター 656
IdML のオープン関数コンポーネント
構成 649
スキーマ 649
パラメーター 649
IdML のクローズ関数コンポーネント 651
構成 651
スキーマ 651
パラメーター 651
IdML の分割関数コンポーネント
構成 653
スキーマ 652
パラメーター 653
IdML 文書 651
IdML パーサー
構成 658
スキーマ 657
パラメーター 658
IdML XML 657
IdML モード 363
IIOP セッション 62, 65
IIOP セッション *(続き)*
ncso.jar ファイル 63
Informix
変更表の作成 305
Informix Dynamic Server 186
InvokeSoap WS 関数コンポーネント
Axis ライブラリー 537
SOAP 537
WSDL 537
InvokeSoap 関数コンポーネント
SOAP メッセージ 539
InvokeSoapWS 関数コンポーネント
構成 537
認証 537
パラメーター 537
input 538, 539
output 538, 539
iPlanet Retro Change Log Plug-in 341
IT レジストリー 641
CDM 644
IT レジストリー CI および関係コネクター
構成 667
構成 アイテム 662
スキーマ 666
パラメーター 667
IT レジストリーの 初期化関数コンポーネント
スキーマ 660
IT レジストリーの初期化関数コンポーネント 665
構成 661
パラメーター 661
CMDBs 659
MSS 659
IT レジストリー・コネクター 665
IT レジストリー・データベース
CDM メタデータ 670
ITIM 107
ITIM Agent コネクター
構成 179
パラメーター 179
DAML プロトコル 178
JNDI 178
SSL のセットアップ 179
ITIM Agent コネクターJNDI
既知の問題 180
ITIM DSML ライブラリー 433
ITIM 専用 107
it_registry.properties ファイル
IdML Ci および関係コネクター 668
IdML コンポーネント 668
IdML のオープン関数コンポーネント 668

J

Java Message Service 346
Java VM 282
Java クラス関数コンポーネント
構成 518
構成エディター 517
パラメーター 518
Java ソース・コード
インプリメンテーション 808
Java のコンポーネント
インプリメンテーション 785
java.object 209
java.objectClass 209
JDBC API 198
JDBC URL 191
JDBC 構成
パラメーター 191
JDBC コネクター 186, 190, 191, 197, 200, 205, 299
埋め込み 203
カスタムの準備済みステートメント 198
クラスパス 182
コネクター構造 182
準備済みステートメント 198, 202, 204
スキーマ 190
ステートメントのカスタマイズ 196
ストアード・プロシージャー 203
タイム・スタンプ 202
ドライバー 182
ドライバーのライセンス交付 184
パラメーター置換 201
複数項目 204
メタデータ・オブジェクト 197
リンク基準の構成 195
リンク・オブジェクト 197
ルックアップのスキップ 195
ワークフロー 182
Derby 189
IBM solidDB 189
jConnect 188
JDBC ドライバー 181
Oracle 186
SQL Server 188
SQL データベース 204
SQLJ 184
Sybase Adaptive Server 188
JDBC ドライバー 186, 353
JMS 機能 205
JMS クライアント 217
JMS コネクター
イテレーター・モード 210, 224
オブジェクト・メッセージ 209
外部システム構成 217

JMS コネクター *(続き)*
機能 205
強制転送 226
テキスト・メッセージ 208
トラブルシューティング 223
バイト・メッセージ 209
プロパティー 211
ヘッダー 211
メッセージ・セキュリティー 226
累算メッセージ 226
ルックアップ・モード 210
例 217
addonly モード 211
call/reply モード 211
functions 205
IBM WebSphere MQ 207
JMS Pub 213
JMS パスワード・ストア・コネクター 224, 225
JMS メッセージ・タイプ 207
JMS メッセージ・フロー 206
JMS サーバー 211
JMS ドライバー
IBM Websphere MQ Everyplace ドライバー 234
IBM Websphere MQ ドライバー 234
JMS パスワード・ストア・コネクター 226, 228
暗号化 228
構成 231
証明書の構造 228
証明書の作成 229
署名 228
スキーマ 231
パラメーター 231
JMS ドライバー 234
PKCS7 228, 229
PKCS7 暗号化 227
.jks ファイル 228
JMS ヘッダー 210
JMS メッセージ・フロー
自動確認 206
JMX クライアント 728
JMX コネクター
構成 236
入力属性マップ 235
パラメーター 236
JMX 1.2 235
JMX Remote API 1.0 235
JMX スキーマ 235
JMX の例
IBM Security Directory Integrator 736
MC4J 構成 736
JMX レイヤー
互換性 739
互換性マトリックス 739
JMX レイヤー *(続き)*
コマンド行 737
サーバー API 733
リモート・アクセス 734
リモート・サーバー API 736
ローカル・アクセス 734
IBM Security Directory Integrator 側 736
MC4J 側 737
JNDI 構成
構成 238
パラメーター 238
JNDI コネクター
インターフェースの変更 241, 242
変更操作 240, 243
ルックアップのスキップ 243
LDAP サーバー 238
JSON パーサー
オブジェクト 445
構成 449
の例 446
配列 445
パラメーター 449
JSON 型と項目とのマッピング 445
JSON 型への項目のマッピング 445
jTDS 188

L

LDAP 173
LDAP グループ・メンバー・コネクター
イテレーター・モード 256
大きな Active Directory グループ 255
構成 257
データ・ソース・スキーマ 256
パラメーター 257
LDAP グループ 255
LDAP グループ項目 256
LDAP サーバー 256
LDAP 構成
構成 245
パラメーター 245
LDAP コネクター
値の削除 253
値の追加 252
オブジェクト・クラス 243
仮想リスト・ビュー制御 250
構成エディター 254
再バインド 254
すべての属性値の削除 253
属性値の置換 253
属性の追加 254
デルタ・モード 245
比較 252
メソッド 252
メモリーの問題 250
LDAP コネクター *(続き)*
ルックアップのスキップ 254
API 252
IBM パスワード同期 243
modrdn 245
SDBM バックエンド 251
LDAP サーバー・コネクター
エラー処理 259
応答チャネル 258
構成 259
スクリプト 258
パラメーター 259
LDAP クライアント 257
LDAP メッセージ戻り値 258
LDAP 照会 11
LDAP 変更検出コネクター 299
ldapsearch 276
ldap.jar 250
LDIF パーサー
構成 459
パラメーター 459
LDAP 操作 457
LDIF 出力 459
LDIF 入力 458
LDIF ファイル 457
Lotus Domino AdminP コネクター
イテレーター 96
グループの名前変更 97
構成 98
サインイン 97
スキーマ 97
パラメーター 98
ユーザーの 名前変更 97
addonly 96
Lotus Domino 変更検出コネクター 68
構成 73
項目の 構造 68
項目の フィルター処理 70
互換性 78
削除された文書 67
システム 時刻 71
使用 67
ソート 70
大規模な データベース 71
同期 状態 70
同期状態の値 69
同期の 最小間隔 68
トラブルシューティング 76
必須 特権 72
必要なセットアップ 72
文書の識別 67
API 71
Domino Server タスク 72
IBM Security Directory バージョン 7.2 66
$$ChangeType 69

Lotus Domino 変更検出コネクター *(続き)*
$$NoteID 68
$$UNID 68
Lotus Domino ユーザー・コネクター
セキュリティー 82
ポート暗号化 82
Lotus Notes コネクター
イテレーター・モード 104
更新モード 108
構成 102
削除モード 108
スクリプトの例 105
制限 100
セキュリティー 106
セッション・タイプ 101
追加/更新モード 104
ファイル所有権の 設定 106
文書の検索 100
文書の削除 100
文書の作成 100
文書の取得 100
文書の変更 100
ルックアップのスキップ 108
割り当て量の設定 106
IIOP 102
IOR 102
Lotus Notes 269
RichText 属性 104, 105
UNID のサポート 104

M

MailOwnerAccess 106
Maximo サーバー
イベント・リスナーの有効化 390
クーロン・タスク 390
構成 389
HTTP エンドポイント 389
Maximo ビジネス・オブジェクト 323
MBean レイヤー 735
MC4J 側
JMX レイヤー 737
MemoryQueue 関数コンポーネント
構成 529
パラメーター 529
MemQueue 276
mode
イテレーター 47
更新 47
addonly 47
delete 47
lookup 47
MQe キュー・マネージャー 224
MQL 368
MS SQL
変更表の作成 304
MSS サポート 368

N

NewsTYpe プロパティー 210
NLS ログ・メソッド
ログ・メッセージの翻訳 821
NT4 コネクター 412

O

objectGUID 11
ODBC データベース 186
ODBC データベース・パス 190
OLE 684
OMNIbus 116
Oracle 349
変更表の作成 302

P

parsers 423
基本パーサー 423
構成エディター 429
文字のエンコード 424
ユーザー定義 500
CBE パーサー 425
CSV パーサー 429
PKCS7 228
PKCS7 暗号化 227
PKCS7 鍵ストア・ファイル 229
PKI ベースの暗号化 227

Q

QRadar
コネクター 284, 285, 288, 290, 291, 292

R

RAC コネクター
イテレーター・モード 293, 296, 297
エージェント・コントローラー 293
構成 295
使用法 296
スキーマ 298
パラメーター 295
addonly モード 293, 296
RDBMS コネクター
構成 300
パラメーター 300
RDBMS 変更検出
変更表の形式 301
RDBMS 変更検出 *(続き)*
変更ログ表 301
RDBMS 変更検出コネクター 299
の例 308
変更表の作成 302, 304, 305, 306
DB2 での変更表 302
Registry Direct API 166
デプロイ 167
resid 821
REST サーバー API
アルゴリズムの例 754
関数項目 755
構成ディレクトリー項目 757
構成ファイル項目 758
構成フィード 757
コネクター項目 755
コンポーネント・フィード 755
サーバー情報項目 755
サーバー制御項目 756
サーバーのシャットダウン 756
サーバー・フィード 755
サーバー・リスナー・フィード 767
通知項目 756
通知の送信 756
トゥームストーン・フィード 768
ポーリング・チャネル 773
リスナー・トランスポート・チャネル 770
リソース階層 752
Atom 項目 757, 768
Atom フィード 757, 768
ConfigInstance フィード 759
HTTP 751, 771
HTTP ヘッダー 752
JMS サーバー 773
JMSDriver アーキテクチャー 773
JSON 752, 772
JSON 構文 771
JSON メディア・タイプ 771
JSON を使用するポリモアフィズム 772
RESTful インターフェース 751
URL 構文 757
XML 752, 771
XML スキーマ 772
XML メディア・タイプ 771
RFC ABAP 592
Rhino 互換性 675
RichText 属性
制限 105
RPC/エンコード・モデル
Axis2 Easy Web Service サーバー 30
RXA ツールキット 557
RXA ライブラリー 557

S

SAP ABAP Application Server Business Object Registry コネクター
  ABAP エラー 614
SAP ABAP Application Server Business Object Repository コネクター
  イテレーター・モード 614
  更新モード 612
  構成 606
  削除モード 613
  使用法 609
  トランザクション操作 614
  パラメーター 606
  ルックアップのスキップ 606
  ルックアップ・モード 613
  Add Only モード 611
  SAP 人事管理 602
SAP ABAP Application Server Component Suite
  アンインストール 587
SAP ABAP Application Server 関数コンポーネント
  関数コンポーネントの出力 590
  関数コンポーネントの入力 589
  構成 589, 590
  使用法 591
  の例 591
SAP ABAP Application Server コンポーネント・スイート 584
  スキーマ 630
  トラブルシューティング 628
  の例 630
  IBM Security Directory Integrator 583
SAP ABAP Application Server ユーザー・レジストリー・コネクター
  イテレーター・モード 600
  更新モード 599
  構成 594
  削除モード 599
  使用法 598
  パラメーター 594
  ルックアップ・モード 600
  AddOnly モード 598
SAP ABAP application server ユーザー・レジストリー・コネクター
  BAPI/RFC 602
SAP ABAP アプリケーション・サーバー 583, 592
  構成 588
  コマンド行 591
  統合パッケージ 586
  トランザクション操作 601
  バージョン番号 586
  パラメーター 588
  ルックアップのスキップ 594
SAP ABAP アプリケーション・サーバー *(続き)*
  Component Suite 585
  RFC Invoker 591
SAP ABAP 関数コンポーネント
  RFC 587
  SAP JCo 587
SAP ALE IDOC コネクター
  インストール 617
  構成 617
  SAP ALE 分散モデル 627
  XML 属性の構文解析 623
SAP ALE IDOC コネクター・スキーマ
  RFM XML 620
  XML 620
SAP ALE 分散モデル
  構成 627
SAP ERP 615
SAP Java connector
  構成 584
SCIM コネクター
  構成 308
  パラメーター 308
  REST プロトコル 308
SDBM バックエンド
  検索 251
  z/OS 251
SDOToXML 関数コンポーネント
  構成 511
  パラメーター 511
  マイグレーション 512
  Ecore モデル 510
  XML スキーマ 510
SendEMail 関数コンポーネント
  構成 527
  出力スキーマ 526
  入力スキーマ 526
  パラメーター 527
  JavaMail API 525
  MIME 525
  SMTP 525
  SMTP サーバー 526
shellCommand オブジェクト
  の例 683
SLM 116
SNMP PDU 337
SNMP TRAP メッセージ 339
SNMP コネクター 337
  構成 338
  の例 339
  パラメーター 338
SNMP サーバー・コネクター 339
  構成 340
  スキーマ 340
  パラメーター 340
SOAP
  エンコード方式のサポート 30
  Axis2 30
  WSDL 30
SOAP XML 文書 466
SOAP 障害
  Axis2 Web サービス・サーバー・コネクター 33
SOAP パーサー 466
  構成 467
  項目の例 466
  固有の呼び出し 467
  パラメーター 467
  文書の例 466
  例 467
SOAP ヘッダー 33
SOAP メッセージ 33
SOAP 要求 407
SPMLv2
  操作 468
SPMLv2 パーサー
  検索フィルターの機能 474
  構成 475
  ストリング以外の属性 473
  操作 469, 470, 471, 472
    検索応答 472
    検索要求 472
    削除応答 471
    削除要求 470
    追加応答 469
    追加要求 469
    変更応答 470
    変更要求 469
    ルックアップ応答 471
    ルックアップ要求 471
  属性操作 473
  の例 476
  バイナリー属性 473
  パラメーター 475
  DSMLv2 468
SQL 照会 204
SQL ステートメント 198
SQL データベース
  列名 204
SQL フィルター仕様 210
SQL92 条件式 210
SSL の使用可能化 217
StAX 481
StAX パーサー 483
Sub コネクター
  構成 213
  パラメーター 213
Sun Directory 変更検出コネクター
  構成 342
  属性のマージ動作 342
  パラメーター 342

Sun Directory 変更検出コネクター *(続き)*
LDAP コネクター 341
Sun/iPlanet Directory Server 5.0 341
SYBASE
変更表の作成 306

T

TADDM API 360
TADDM API JAR 376
TADDM SDK 360
TADDM コネクター 360, 368
暗黙属性 363
インストール後のタスク 376
既存のモデル・オブジェクトの更新 373
構成 366, 378
構成アイテムと関係 371
システム属性 366
使用法 366
スキーマの比較 365
追加属性 369
データ表現フォーマット 362, 363
データ表現モード 364, 365
データ表現モデル 361
データ・ソース・スキーマ 375
デルタ・モードのサポート 374
統一スキーマ 363
トラブルシューティング 377
パラメーター 366, 378
モデル・オブジェクトの削除 371
例外のスロー 377
AccessException のスロー 377
addonly モード 372
CDM 363
CDM フォーマット 364
Common Data Model 361
IT レジストリー・モデル 362
TADDM MQL 370
TADDM モデル 362
TADDM 変更検出
スリープ間隔パラメーター 355
TADDM 変更検出コネクター 355
構成 357
データ・ソース・スキーマ 356
パラメーター 357
TADDM データベース 354
TCP コネクター
イテレーター・モード 380
構成 381
パラメーター 381
addonly モード 380, 381
TCP サーバー・コネクター
イテレーター・モード 382
構成 382
コネクター・スキーマ 383
TCP サーバー・コネクター *(続き)*
サーバー・モード 382
パラメーター 382
TCP のスクリプト記述
直接 56
TCP/URL のスクリプト記述
直接 56
TDILog4J 815
TDIOutputter 134, 135
tdi.xml 795
tdi.xml ファイルのフォーマット
インストール・ロケーション 797
tdi.xml フォーマット
コード例 796
作成 796
Textmessage の使用 209
TIM DSMLv2 コネクター 107
構成 108
HTTPS 108
ITIM サーバー 108
SSL 108
timeout 173
Tivoli Enterprise Console 117
TLS/SSL 130
Tpae IF コネクター
イテレーター・モード 394
エラー処理 400
外部システム構成 401
基本サービス 393
更新モード 397
構成 402
コネクターの相違点 393
削除モード 397
照会 XML 394
使用法 393
の例 405
パラメーター 402
リンク基準 399
ルックアップ・モード 399
addonly モード 397
Maximo 399
XML スキーマ定義 401
Tpae IF 変更検出コネクター
アーキテクチャー 385
構成 391
デルタ・タグ 387
の例 385, 392
パラメーター 391
ルート MBO 387
HTTP 要求 384
Maximo サーバー 389

U

UDP 339
UNID のサポート 104
URL コネクター 406
構成 406
パラメーター 406
HTTP 406
HTTPS 406
URL プロトコル 406
URL のスクリプト記述
直接 57
User Registry Connector 592, 630
IBM Security Directory Integrator スキーマ 598
Xschema 632
UTF-8 478, 496

W

Web サービス受信側サーバー・コネクター 407
構成 409
コネクター操作 411
出力スキーマ 408
入力スキーマ 408
パラメーター 409
Windows NT セキュリティー・データベース 412
Windows ユーザー 416
Windows ユーザーおよびグループ・コネクター 416
グループの操作 415
グループ・アカウント名 414
グループ・メンバーシップの変更 417
構成 413
新規ユーザー 415
前提条件 412
パスワードの設定 415
パラメーター 413
ユーザーおよびグループ・データの削除 418
ユーザーおよびグループ・データの抽出 417
ユーザーおよびグループ・データの追加 417
ユーザーおよびグループ・データの変更 418
ユーザー・アカウント名 414
リンク基準の構成 413
例 416
1 次グループ/グローバル・グループのメンバーシップ 415
Windows ユーザー/グループ・コネクター 412
前提条件 414
WrapSoap 関数コンポーネント
Axis ライブラリー 534
SOAP メッセージ 534
WSDL 534

WrapSoap 関数コンポーネント *(続き)*
XML エレメント 534
WSDL 407, 543, 544
WSDL の生成
Axis2 WS サーバー・コネクター 34
WSDL ファイル 24
ホスティング 24, 407
HTTP 407

X

Xalan ライブラリー 476
XLXP 481
XML DOM パーサー 496
XML SAX パーサー
構成 495
パラメーター 495
文字のエンコード 496
Apache Xerces ライブラリー 494
XML インスタンス文書 630
XML パーサ 481
書き込み時のナビゲーション 483, 486
構成 481
スキーマの構成 491
定義済み XSD スキーマ 491
ナビゲーション 483
の例 491
パラメーター 481
文字のエンコード 490
読み取り時のナビゲーション 483
XML 構造のナビゲーション 483
XML の書き込み 489
XML の読み取り 488
XSD 491
XSD スキーマ 491
XSD スキーマの例 493
XML 変換の考慮事項
構成エディター 802
XML ファイル 802
XMLToSDO 関数コンポーネント
構成 508
項目属性 507
の例 507
パラメーター 508
マイグレーション 509
XML エレメント 507
XMP パーサー
読み取り時のナビゲーション 483
XPath のインプリメンテーション 483
XSL ベースの XML パーサー 496
構成 497
の例 499
パラメーター 497
XSL ベースのパーサー 498
XSLT 498

Z

z/OS
LDAP 変更ログ・コネクター 418
z/OS LDAP 変更ログ・コネクター
属性のマージ動作 419
z/OS TSO コマンド行関数コンポーネント
エラー・フロー 565
許可 567
構成 564
認証 566
input 566
output 566
REXX スクリプト 565, 571
TSO/E シェル・コマンド 565
z/OS TSO/E コマンド行関数コンポーネント
ACF2 564
RACF 564
z/OS 変更ログ・コネクター
構成 419
詳細情報 422
パラメーター 419

[特殊文字]

$initialization 操作
構成パネル 781
AL コネクター 781
.jks ファイル 227, 228
.nsf ファイル 71

Printed in Japan

SC88-8414-03

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21