

**IBM Content
Manager for Multiplatforms/IBM
Information
Integrator for Content**



ワークステーション・アプリケーション・プログラミング・ ガイド

バージョン 8 リリース 2

**IBM Content
Manager for Multiplatforms/IBM
Information
Integrator for Content**



ワークステーション・アプリケーション・プログラミング・ ガイド

バージョン 8 リリース 2

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、547 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM Enterprise Information Portal for Multiplatforms バージョン 8 リリース 2、IBM Content Manager for Multiplatforms (製品番号 5724-B43、5724-B19) に適用されます。また、改訂版などで特に断りのない限り、これ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も適用されます。

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

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

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

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

(URL は、変更になる場合があります)

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

原 典： SC27-1347-01
IBM Content Manager for Multiplatforms/
IBM Information Integrator for Content
Workstation Application Programming Guide
Version 8 Release 2

発 行： 日本アイ・ピー・エム株式会社

担 当： ナショナル・ランゲージ・サポート

第1刷 2003.2

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

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

© Copyright International Business Machines Corporation 1996, 2003. All rights reserved.

© Copyright IBM Japan 2003

目次

本書について	vii
本書の対象読者	vii
本書の詳細情報の入手先	vii
製品パッケージに含まれている情報	viii
Web から使用可能なサポート	ix
ご意見の送付方法	ix
Enterprise Information Portal バージョン 8 の新機能	x
Content Manager バージョン 8 の新機能	xii

Enterprise Information Portal の紹介 . . . 1

カスタマー情報の検索	1
ニーズ	2
ソリューション	2
Enterprise Information Portal ソリューション	2
IBM Enterprise Information Portal for Multiplatforms	
コンポーネント	4
バージョン 8.2 API の新機能	6
バージョン 8.1 API の新機能	7
新規および変更された Java クラス	8
新規および変更された C++ クラス	9

Enterprise Information Portal アプリケーション・プログラミングの概念 . . . 13

コンテンツ・サーバーを介したデータ・アクセスについて	13
動的データ・オブジェクトの概念について	14
動的データ・オブジェクト (DDO)	14
拡張データ・オブジェクト (XDO)	15
マルチメディア・コンテンツの表示	16
コンテンツ・サーバーおよび DDO について	16
DDO/XDO と属性値および項目パーツとの比較	16
永続 ID (PID) の説明	17

統合コンテンツ・サーバーおよび統合検索の処理 . . . 19

統合スキーマ・マッピング	21
統合コンテンツ・サーバー・マッピング・コンポーネントの使用	22
統合照会の実行	22
統合照会の構文	24
統合フォルダーへの照会結果の保管 (Java のみ)	26
システム管理の処理	27
EIP システム管理クライアントのカスタマイズ	27

アプリケーション・プログラミング・インターフェース (API) を使用したプログラミング . . . 29

Java API と C++ API の違い	29
クライアント/サーバー・アーキテクチャーの概要 (Java のみ)	29

Java 環境用のパッケージ	30
プログラミング上のヒント	31
Java 環境のセットアップ (Java のみ)	31
Windows 用の Java 環境変数の設定	31
AIX 用の Java 環境変数の設定	32
Solaris 用の Java 環境変数の設定	32
コンテンツ・サーバーでのリモート・メソッド呼び出し (RMI) の使用	33
C++ 環境のセットアップ (C++ のみ)	33
Windows 用の C++ 環境変数の設定	35
AIX 用の C++ 環境変数の設定	35
C++ プログラムのビルド	36
Windows でのコード・ページ変換用コンソール・サブシステムの設定	37
複数検索オプションの概要	37
トレース	38
テキスト検索エンジンを使用したテキスト照会のトレース	38
パラメトリック照会のトレース	39
例外の処理	39
定数	41
コンテンツ・サーバーへの接続	41
接続の確立	42
クライアント内のコンテンツ・サーバーからの接続および切断	43
コンテンツ・サーバー・オプションの設定および取得	43
コンテンツ・サーバーのリスト作成	43
コンテンツ・サーバー用のエンティティおよび属性のリスト	44
動的データ・オブジェクト (DDO) の使用	47
DKDDO の作成	48
DDO へのプロパティの追加	49
永続 ID (PID) の作成	50
データ項目およびプロパティの処理	51
DKDDO と属性プロパティの取得	53
DDO 全体の表示	55
DDO の削除 (C++ のみ)	57
拡張データ・オブジェクト (XDO) の使用	57
XDO 永続的 ID (PID) の使用	58
XDO プロパティについて	58
DB2、ODBC、および DataJoiner の構成ストリング (C++ のみ)	58
Java プログラミングのヒント	59
C++ プログラミングのヒント	60
DDO のパーツとして使用する XDO のプログラミング	60
独立型 XDO のプログラミング	62
XDO の処理例	66
XML の使用 (Java のみ)	89

I	XML をインポートおよびエクスポートする機能	
I	の拡張	90
	XML 文書のインポート	90
	XML のエクスポート	95
	文書の作成と DKPARTS 属性の使用	95
	フォルダーの作成と DKFOLDER 属性の使用	98
	DKAny の使用 (C++ のみ)	101
	タイプ・コードの使用	102
	DKAny のメモリー管理	102
	コンストラクターの使用	102
	タイプ・コードの入手	103
	DKAny に新しい値を代入する	103
	DKAny の値の代入	103
	DKAny の表示	104
	DKAny の破棄	104
	プログラミング上のヒント	105
	コレクションおよびイテレーターの使用	105
	順次コレクション・メソッドの使用	105
	順次イテレーターの使用	106
	コレクションのメモリー管理 (C++ のみ)	109
	コレクションのソート	110
	統合コレクションおよび統合イテレーターについて	111
	コンテンツ・サーバーの照会	112
	dkResultSetCursor と DKResults との相違点	113
	パラメトリック照会の使用	113
	テキスト照会の使用	119
	結果セット・カーソルの使用	133
	照会を再実行するための結果セット・カーソルの オープンおよびクローズ	134
	結果セット・カーソルの位置の設定および取得	134
	結果セット・カーソルからのコレクションの作成	137
	コレクションの照会	137
	照会の結果の取得	138
	新規照会の評価	138
	結合照会に代わる照会可能コレクションの使用	139

Content Manager バージョン 8.2 の使 用 141

	Content Manager システムの概要	141
	Content Manager の概念	142
	項目	143
	属性	143
	項目タイプ	143
	ルート・コンポーネントと子コンポーネント	144
	オブジェクト	145
	リンクおよび参照	145
	文書	146
	フォルダー	146
	バージョン管理	146
	アクセス制御	148
	Content Manager アプリケーションの計画	152
	アプリケーションの機能の決定	152
	エラーの処理	153
	Content Manager サンプルの使用	154
	保険シナリオ・サンプル	154

	Content Manager アプリケーションの作成	155
	ソフトウェア・コンポーネントの説明	155
	DDO を使用した項目の表示	156
	Content Manager システムへの接続	156
	項目の使用	158
	フォルダーの使用	182
	項目間のリンクの定義	189
	アクセス制御の使用	192
	特権の作成	192
	特権セットの作成	193
	特権セット・プロパティの表示	195
	アクセス制御リスト (ACL) の定義	196
	ACL 情報の検索と表示	198
	項目タイプへの ACL の割り当て	199
	項目への ACL の割り当て	200
	照会言語について	201
	Content Manager サーバーの照会	202
	Content Manager データ・モデルへの照会言語の適 用	203
	パラメトリック検索について	204
	テキスト検索について	205
	オブジェクト・コンテンツの検索	205
	文書の検索	206
	ユーザー定義属性をテキスト検索可能にする	206
	テキスト検索構文について	206
	パラメトリック検索とテキスト検索の組み合わせ	207
	照会の例	210
	照会の例	213
	照会言語について	219
	比較演算子 ("=", "!=", ">", "<", "BETWEEN" およびその他) でのエスケープ・シーケンスの使 用	220
	LIKE 演算子でのエスケープ・シーケンスの使用	220
	拡張テキスト検索 ("contains" および "score" 関 数) でのエスケープ・シーケンスの使用	221
	基本テキスト検索 ("contains-text-basic" および "score-basic" 関数) でのエスケープ・シーケンス の使用	222
	Java および C++ でのエスケープ・シーケンス の使用	224
	照会言語の文法	224
	リソース・マネージャーの使用	227
	リソース・マネージャー・オブジェクトの使用	228
	Content Manager における文書管理	228
	文書管理データ・モデルの作成	230
	文書項目タイプの作成	230
	文書の作成	232
	文書の更新	234
	文書の検索と削除	236
	文書管理データ・モデル内のパーツのバージョン 管理	236
	トランザクションの使用	237
	アプリケーションにおけるトランザクション設計 時の考慮事項	238
	明示的トランザクション使用時の注意	239

トランザクションにおけるチェックインおよびチェックアウトの使用	239
トランザクションの処理	240
プロセスにおける文書ルーティング	241
文書ルーティング・プロセスの概要	241
文書ルーティング・プロセスのセットアップ	242
その他のコンテンツ・サーバーの使用	265
旧バージョンの Content Manager の使用	267
ラージ・オブジェクトの処理	268
旧バージョンの Content Manager のコンテンツを表す DDO の使用	268
文書またはフォルダーの作成、更新、および削除	270
文書またはフォルダーの検索	278
テキスト検索 (テキスト検索エンジン) について	282
コンテンツ別のイメージの検索	306
イメージ検索アプリケーションの使用	308
QBIC での接続の確立	313
イメージ検索サーバーのリスト作成	314
イメージ検索データベース、カタログ、およびフィチャーのリスト作成	315
DDO を使用したイメージ検索情報の表現	318
イメージ照会の処理	319
イメージ検索エンジンの使用	322
検索エンジンを使用した既存の XDO の索引付け	323
結合照会の使用	326
旧バージョンの Content Manager のワークフロー関数とワークバスケット関数について	330
OnDemand の使用	339
OnDemand サーバーおよび文書の表示	339
OnDemand サーバーへの接続と切断	340
OnDemand に関する情報のリスト作成	341
OnDemand 文書の取得	344
OnDemand フォルダー・モードの使用可能化	351
非同期検索	351
検索テンプレートとしての OnDemand フォルダー	352
ネイティブ・エンティティとしての OnDemand フォルダー	352
注釈の作成と変更	352
トレース	352
Content Manager ImagePlus for OS/390 の使用	354
エンティティおよび属性のリスト作成	355
ImagePlus for OS/390 の照会構文	359
Content Manager for AS/400 の使用	361
エンティティ (索引クラス) および属性のリスト作成	362
照会の実行	363
パラメトリック照会の実行	368
Domino.Doc の使用	369
エンティティおよびサブエンティティのリスト作成	372
キャビネット属性のリスト作成	374
Domino.Doc での照会の作成	374
照会構文の使用	375

Extended Search (ES) の使用	376
Extended Search サーバーのリスト作成	376
エンティティ (データベース) および属性 (フィールド) のリスト作成	377
Generalized Query Language (GQL) の使用	381
Extended Search での DDO 項目タイプの識別	382
Extended Search での PID の作成	383
Extended Search 文書の内容の処理	383
文書の検索	388
バイナリー・ラージ・オブジェクト (BLOB) の検索	389
MIME タイプと文書との関連付け	390
Extended Search での統合検索の使用	391
Panagon Image Services の使用 (Java のみ)	391
データ・モデル	392
Panagon Image Services の文書およびページ	392
エンティティおよび属性のリスト作成	393
照会	394
リレーショナル・データベースの使用	397
リレーショナル・データベースへの接続	398
エンティティおよびエンティティ属性のリスト	400
照会の実行	402
カスタム・コンテンツ・サーバー・コネクタの作成	405
カスタム・コンテンツ・サーバー・コネクタの開発	405
FeServerDefBase クラスの使用 (Java のみ)	422

EIP ワークフロー・アプリケーションの構築 423

ワークフロー・サービスへの接続	423
ワークフローの開始	424
ワークフローの終了	425
すべてのワークフローのリスト作成	426
ワークフローの延期	427
ワークフローの再開	428
すべてのワーク・リストのリスト作成	428
ワーク・リストへのアクセス	429
作業項目へのアクセス	430
ワークフロー内の項目の移動	432
すべてのワークフロー・テンプレートのリスト作成	432
ユーザー独自のアクションの作成 (Java のみ)	433

非ビジュアルおよびビジュアル JavaBeans でのアプリケーションの構築 435

Bean の基本概念について	435
ビルダーでの JavaBeans の使用	437
IBM Websphere Studio Application Developer の使用	437
EIP Java Bean の起動	438
非ビジュアル Bean	438
非ビジュアル Bean の構成	439
非ビジュアル Bean の機能の概要	439

非ビジュアル Bean のカテゴリ	440
非ビジュアル Bean を使用する際の考慮事項	444
Bean のトレースとロギング	445
非ビジュアル Bean のプロパティとイベントについて	445
非ビジュアル Bean を使用したアプリケーションのビルド	446
ビジュアル Bean の使用	446
CMBLogonPanel bean	447
CMBSearchTemplateList bean	449
CMBSearchTemplateViewer bean	450
CMBSearchTemplateViewer のフィールドの妥当性検査または編集	451
CMBSearchPanel bean	451
CMBSearchResultsViewer bean	451
ポップアップ・メニューのオーバーライド	453
CMBFolderViewer bean	453
CMBDocumentViewer bean	455
ビューアーの指定	455
デフォルトのビューアー	456
外部ビューアーの立ち上げ	456
CMBItemAttributesEditor bean	456
CMBItemAttributesEditor での変更の拒否	457
CMBVersionsViewer bean	457
ビジュアル Bean の一般的な動作	457
ビジュアル Bean の置換	458
ビジュアル Bean を使用したアプリケーションのビルド	459

Java 文書ビューアー・ツールキットの使用 463

ビューアー・アーキテクチャー	464
文書エンジン	464
注釈エンジン	465
汎用文書ビューアーの作成	465
汎用文書ビューアーのカスタマイズ	465
アプリケーション例	467
スタンドアロン・ビューアー	468
Java アプリケーション	468
シン・クライアント	469
アプレットまたはサーブレット	469
二重モードおよびアプレットまたはサーブレット	470
注釈サービスの使用	470
注釈サービス・インターフェースの使用	471
注釈編集サポートについて	472
注釈サービスを使用したアプリケーションの作成	473
ユーザー・アプリケーションへのカスタム注釈タ	
イプの追加	473

Enterprise Information Portal タグ・ライブラリーおよびコントローラー・サーブレットでの使用 475

タグ・ライブラリーおよびサーブレットのセットアップ	475
タグ・ライブラリーの使用	475
タグ・ライブラリーで使用される規則	476
タグの要約	476
接続関連のタグ	476
スキーマ関連タグ	477
検索関連のタグ	478
項目関連のタグ	478
フォルダー関連のタグ	479
文書関連のタグ	480
EIP コントローラー・サーブレット	480
サーブレットの機能	480
サーブレット参照	482
サーブレット・ツールキット機能マトリックス	487

情報マイニングの使用 489

Bean を使用した情報マイニング・アプリケーションの作成	489
サンプル・ファイルの位置	492
文書のカテゴリ化	493
文書の要約	501
情報の抽出	507
クラスター化	512
Web スペースからの文書のインポート	517
カテゴリによる文書の検索	526
独自のコンテンツ・プロバイダーの作成	531
サービス API の使用	532
情報マイニング・サービス API への接続	533
ライブラリー、分類法、およびカタログの管理	534
情報マイニング・ツールの使用	537
レコードの作成およびカタログへのメタデータの保管	540
文書の検索	542
サーバー・タスクの実行	542
JSP に基づく情報マイニング・アプリケーションの例	544
JSP のインストール	545

特記事項 547

商標	549
----	-----

用語集 551

索引 561

本書について

本書は、Enterprise Information Portal (EIP) バージョン 8 リリース 2 および Content Manager (CM) バージョン 8 リリース 2 に提供される Java™、JavaBeans™、および C++ のアプリケーション・プログラミング・インターフェース (API) の使用方法について説明しています。これらの API および Bean は、異機種混合のコンテンツ・サーバーに保管されているコンテンツにアクセスするアプリケーションを作成するための、ビルディング・ブロックを提供します。

旧バージョンでは、CM と EIP に別々のアプリケーション・プログラミング・ガイドがありました。バージョン 8 リリース 2 では、これら 2 つの製品は多数の API を共用し、多くの場合、同じプログラミングの概念に基づいています。また、Enterprise Information Portal バージョン 8 リリース 2 の新しい機能の多くは、CM バージョン 8 リリース 2 への新しいコネクタを含んでいるので、2 つのガイドが 1 つにまとめられました。

本書では以下の点を記述しています。

- Enterprise Information Portal および CM のアプリケーション・プログラミングの概念 (Java および C++ における動的データ・オブジェクトの概念など) の概要
- CM バージョン 8 リリース 2 コネクタから使用可能な機能の説明
- コンテンツ・サーバーへのその他すべての Enterprise Information Portal コネクタに関する文書
- ビジュアルおよび非ビジュアル JavaBeans の更新内容
- 情報マイニング、IBM® Web Crawler およびワークフローに関するプログラミング情報の更新内容

Content Manager に関する図は、この製品のバージョン 8.1 およびバージョン 8 リリース 2 に適用されます。

本書の対象読者

本書の内容は、以下のスキルの一部またはすべてを持つアプリケーション・プログラマーを対象としています。

- C++、Java、JavaBeans、または HTML のいずれかの経験がある
- リレーショナル・データベースの概念について習熟している
- DDO/XDO プロトコルの知識がある

本書の詳細情報の入手先

製品パッケージには、完全セットの情報が含まれており、システムの計画、インストール、管理、および使用の際に役立ちます。製品の資料およびサポートは、Web 上からでも入手可能です。

製品パッケージに含まれている情報

製品パッケージには、Information Center および PDF 形式の各資料が入っています。

Information Center

製品パッケージには、Information Center が入っており、製品のインストール時にインストールできます。Information Center のインストールについては、「*Content Management System* の計画とインストール」を参照してください。

Information Center には、Content Manager、Enterprise Information Portal、および Content Manager VideoCharger の資料が入っています。トピック関連の情報は、製品別およびタスク別 (例: 管理) に編成されています。用意されているナビゲーション機構および索引の他に、検索機能も検索の助けとなります。

PDF 資料

オペレーティング・システムに合った Adobe Acrobat Reader を使用して、PDF ファイルをオンラインで表示できます。Acrobat Reader がインストールされていない場合は、www.adobe.com の Adobe Web サイトからダウンロードできます。

表 1 は、IBM Content Manager for Multiplatforms に付属している Content Manager 資料を示しています。

表 1. Content Manager 資料

ファイル名	資料タイトル	資料番号
install	<i>Content Management System</i> の計画とインストール ¹	GC88-9200-01
migrate	<i>Content Manager</i> バージョン 8.1 へのマイグレーション	SC88-9202-01
sysadmin	システム管理ガイド	SC88-9201-01

IBM Content Manager for Multiplatforms には、IBM Enterprise Information Portal for Multiplatforms が同梱されています。また、IBM Enterprise Information Portal for Multiplatforms を単独で注文することもできます。表 2 は、製品に付属している Enterprise Information Portal 資料を示しています。

表 2. Enterprise Information Portal 資料

ファイル名	資料タイトル	資料番号
apgwork	ワークステーション・アプリケーション・プログラミング・ガイド ¹	SC88-9205-01
ecliinst	<i>eClient</i> のインストール、構成と管理	SC88-9207-02
eipinst	<i>Enterprise Information Portal</i> の計画とインストール	GC88-9203-01
eipmanag	<i>Enterprise Information Portal</i> の管理	SC88-9204-01
messcode	メッセージとコード ²	SC88-9206-01

表 2. Enterprise Information Portal 資料 (続き)

ファイル名	資料タイトル	資料番号
注:		
1. 「ワークステーション・アプリケーション・プログラミング・ガイド」には、Content Manager と Enterprise Information Portal の両方のアプリケーションのプログラミングに関する情報が記載されています。		
2. 「メッセージとコード」には、Content Manager と Enterprise Information Portal のメッセージとコードが記載されています。		

Web から使用可能なサポート

製品サポートは、Web から使用可能です。以下の製品 Web サイトで「サポート (Support)」をクリックします。

www.ibm.com/software/data/cm/

www.ibm.com/software/data/eip/

この資料は、ソフトコピー形式で製品に付属されるものです。Web 上の製品資料にアクセスするには、製品の Web サイトの「ライブラリー (Library)」をクリックします。

HTML ベースのドキュメンテーション・インターフェースは、Enterprise Documentation Online (EDO) と呼ばれ、Web から使用可能です。ここには、現在、API 参照情報が記載されています。EDO へのアクセス方法については、Enterprise Information Portal ライブラリーの Web ページを参照してください。

ご意見の送付方法

お客様のご意見は、IBM が品質情報を提供する際の貴重な資料となります。本書、あるいはその他の Content Manager または Enterprise Information Portal 資料に関するご意見をお寄せください。ご意見は、以下のいずれかの方法でご送付ください。

- Web からご意見を送付できます。以下の URL の IBM Data Management Online Reader's Comment Form (RCF) ページを参照してください。

www.ibm.com/software/data/rcf

このページを利用して、ご意見を入力し、送信してください。

- comments@vnet.ibm.com へ E メールでご意見を送付してください。製品名、製品のバージョン番号、および資料名および資料番号 (存在する場合) を必ず記載してください。本書の特定の箇所に関するご意見の場合は、本文の位置 (例えば、章およびセクションのタイトル、表の番号、ページ番号、またはヘルプ・トピックのタイトル) を記載してください。

Enterprise Information Portal バージョン 8 の新機能

バージョン 8.2: バージョン 8.2 には、さまざまな機能強化が組み込まれています。バージョン 8.2 は、システム管理ワークフローの機能を強化し、DB2 Universal Database バージョン 8.1 の最新データベース・テクノロジーをサポートしています。これらの主要機能、およびバージョン 8.2 製品に対するその他の機能強化について、次に要約します。

Enterprise Information Portal から IBM Information Integrator for Content への名称変更

Enterprise Information Portal の名称が Information Integrator for Content に変更されました。バージョン 8.2 の資料名は変更されましたが、資料の本文では引き続き製品名を Enterprise Information Portal としています。新しい名称への移行が完了するまで、Web 上で詳細情報を検索する際には引き続き Enterprise Information Portal (または EIP) を使用してください。

DB2 UDB V8.1 のサポート

Enterprise Information Portal V8.2 のサポート。DB2 V8.1 の接続集中機能により、2 層アプリケーションとクライアントのスケラビリティが向上しました。

統合フォルダー

複数のリポジトリにある文書とネイティブ・フォルダーを単一の統合フォルダーに編成し、ワークフロー上でそのフォルダーを開始する機能が eClient に追加されました。また、統合フォルダーを使用すると、EIP 統合データベースに検索結果を永続的に保管でき、ユーザーはこのデータベースから検索結果をいつでも取り出すことができます。これらの統合フォルダーに対して、再索引付けを行わずに、すべての CRUD (作成、検索、更新、および更新) 操作を使用できます。

拡張ワークフロー・コレクション・ポイント

ワークフローが AIX および Solaris で完全サポートされました。ワークフロー・ビルダー、API、コレクション・ポイント・モニター、および JavaBeans のワークフロー機能と使いやすさが改善されました。

Microsoft Visual Studio .NET によるアプリケーションの作成

Enterprise Information Portal 8.1 以降の API は、Microsoft Visual Studio .NET によるコンテンツ管理アプリケーションの作成や、Microsoft Visual Studio .NET を使用して作成したアプリケーションの統合をサポートします。

バージョン 8.1: バージョン 8.1 は、統合性と多機能性を追求した初の製品です。主要機能や、前の Content Manager 製品から改善された機能が多数ありますが、その一例は、文書の高度なカスタマイズを可能にする新しいデータ・モデル構造です。Content Manager 製品のバージョン 8.1 での変更点を次に要約します。

Sun Solaris のサポート

Solaris システムに Java コネクタ、フィーチャー、およびデータベースをインストールできます。

共通システム管理

単一のクライアント・アプリケーションは、Content Manager および Enterprise Information Portal 管理に対して、別々にアクセスできます。

新しいコネクタ

- Content Manager バージョン 8 リリース 1 の ICM コネクタを使用すると、Content Manager バージョン 8 の強力な文書保存機能を利用することができます。
- 新しい C++ Extended Search バージョン 3.7 のコネクタは AIX® 上で実行されます。

改良されたコネクタ

- パラメトリック・テキスト検索は、統合レイヤーから、および直接、Extended Search の接続を介してサポートされます。
- OnDemand コネクタに対する機能強化およびパフォーマンスの改良。以下のものがあります。
 - OnDemand DDO の構造に対する変更。
 - 新たに非対称検索がサポートされました。

IBM Web Crawler

IBM Web Crawler は、Web および Lotus Notes® データベースの情報を検索および要約することのできるフィーチャーです。

ワークフローの機能強化

ワークフローが AIX および Solaris で完全サポートされました。ワークフロー・ビルダー、API、および JavaBeans が提供するワークフロー機能および使用可能度が改善されました。

Information Center

ブラウザー・ベースの Information Center には、Content Manager、Enterprise Information Portal、および Content Manager VideoCharger™ の資料が掲載されています。トピック関連の情報は、製品別およびタスク別（例：管理）に編成されています。用意されているナビゲーション機構および索引の他に、検索機能も検索の助けとなります。

アクセス支援

アクセス支援フィーチャーは、運動障害や視覚障害などの身体障害を持つユーザーが、ソフトウェア製品を問題なく使用できるよう支援します。本製品のアクセス支援の主なフィーチャーを以下に示します。

- マウスの代わりにキーボードからすべてのフィーチャーが操作可能。
- 拡張表示プロパティのサポート。
- 映像および音によって注意を喚起するためのオプション。
- 支援テクノロジーとの互換性。
- オペレーティング・システムのアクセス支援フィーチャーとの互換性。
- アクセス可能な文書フォーマット。

Content Manager バージョン 8 の新機能

バージョン 8.2: バージョン 8.2 には、バージョン 8.1 からのさまざまな機能強化が組み込まれています。バージョン 8.2 は、eClient にワークフロー機能をさらに追加し、リソース管理機能を強化し、DB2 Universal Database バージョン 8.1、Oracle バージョン 8.1.7.4 および Version 9.2.0.1、WebSphere バージョン 5 など、データベースとクライアントの最新テクノロジーをサポートしています。これらの主要機能、およびバージョン 8.2 製品に対するその他の機能強化について、次に要約します。

Enterprise Information Portal から IBM Information Integrator for Content への名称変更

Enterprise Information Portal の名称が Information Integrator for Content に変更されました。バージョン 8.2 の資料名は変更されましたが、資料の本文では引き続き製品名を Enterprise Information Portal としています。新しい名称への移行が完了するまで、Web 上で詳細情報を検索する際には引き続き Enterprise Information Portal (または EIP) を使用してください。

Oracle バージョン 8.1.7.4 または 9.2.0.1 以降のサポート

Content Manager V8.2 は、ライブラリー・サーバーとリソース・マネージャーの両方に保管されているメタデータを管理する、Oracle データベースのサポートを強化しています。Content Manager バージョン 7 の Oracle ユーザー用に、マイグレーション・ツールが付属しています。**注:** Oracle は、Enterprise Information Portal データベース・サーバーのコンテンツを管理しません。

複製 Content Manager V8.2 には、リソース・マネージャーの複製機能が組み込まれています。これは、複数の場所にオブジェクトを保管し、複製リソース・マネージャーによって管理する機能です。オブジェクトのレプリカは、ロード・バランシングを効率化するための LAN キャッシュ・オブジェクトとして機能します。

LAN キャッシュ

Content Manager V8.2 の LAN キャッシュのサポートにより、システム管理者の定義によるローカル・サーバーを使用したアプリケーション透過キャッシングが可能です。

DB2 UDB V8.1 のサポート

Content Manager V8.2 と Enterprise Information Portal V8.2 は、DB2/UDB V8.1 をサポートします。DB2 V8.1 の接続集中機能により、2 層アプリケーションとクライアント (Windows 用 Content Manager V8 クライアントなど) のスケーラビリティが向上しました。DB2/UDB V8.1 では、DB2 Universal Database Text Information Extender (TIE) が Net Search Extender (NSE) に置き換えられました。

WebSphere Application Server バージョン 4 とバージョン 5 のサポート

WebSphere Application Server バージョン 5 は、サーバーの配置、およびデータ・アクセスとデータ管理をどの Web ブラウザーからも実行できる機能を導入しています。

統合フォルダー

複数のリポジトリにある文書とネイティブ・フォルダーを単一の統合フォルダーに編成し、ワークフロー上でそのフォルダーを開始する機能が eClient に追加されました。また、統合フォルダーを使用すると、EIP 統合データベースに検索結果を永続的に保管でき、ユーザーはこのデータベースから検索結果をいつでも取り出すことができます。これらの統合フォルダーに対して、再索引付けを行わずに、すべての CRUD (作成、検索、更新、および更新) 操作を使用できます。

拡張ワークフロー・コレクション・ポイント

ワークフローが AIX および Solaris で完全サポートされました。ワークフロー・ビルダー、API、コレクション・ポイント・モニター、および JavaBeans のワークフロー機能と使いやすさが改善されました。

Microsoft Visual Studio .NET によるアプリケーションの作成

Content Manager と Enterprise Information Portal 8.1 以降の API は、Microsoft Visual Studio .NET によるコンテンツ管理アプリケーションの作成や、Microsoft Visual Studio .NET を使用して作成したアプリケーションの統合をサポートします。

バージョン 8.1: バージョン 8.1 は、統合性と多機能性を追求した初の製品です。主要機能や、前の Content Manager 製品から改善された機能が多数ありますが、その一例は、文書の高度なカスタマイズを可能にする新しいデータ・モデル構造です。Content Manager 製品のバージョン 8.1 での変更点を次に要約します。

パフォーマンスの改良

ライブラリー・サーバーおよびリソース・マネージャーでは DB2 のストアード・プロシージャを使用して DB2 テクノロジーを活用し、ネットワーク・トラフィックを大幅に削減し、パフォーマンスおよびスケールビリティを改良しています。

Sun Solaris のサポート

ライブラリー・サーバーおよびリソース・マネージャーはどちらも Sun Solaris にインストールできます。

拡張データ・モデル

カスタマイズされた複合文書管理ソリューションの基礎として、新しい階層データ・モデルを採用しています。

ワークフローの改良

統合文書ルーティングにより、順次ルーティング、動的ルーティング、およびコレクション・ポイントを使用したワークフローの機能が改良されています。

統合テキスト検索

属性ベースの検索に加えて、クライアント・ユーザーはテキスト中心の文書情報に対して全文検索を実行できるようになりました。テキスト検索機能では、DB2 Universal Database Text Information Extender が使用されます。これは、テキスト検索をセットアップするためのストリームライン・プロセスで使用されます。

共通システム管理

単一のクライアント・アプリケーションは、Content Manager および Enterprise Information Portal に対して、別々にアクセスできます。Content Manager 内では、管理ドメインによりライブラリー・サーバーのサブセクションへの管理アクセスを制限できます。

全機能を持つデスクトップ・クライアントおよび拡張 eClient

クライアントが拡張され、迅速な配置または基幹業務の統合のための優れたアプリケーションが提供されるようになりました。Client for Windows は、インポート中の、統合テキスト検索、文書ルーティング、(単一の子コンポーネント・レベルへの) 階層データ・モデル、バージョン管理、および索引をサポートしています。eClient には、統合テキスト検索、EIP 拡張フロー、バージョン管理、および複数値属性が含まれます。

さらに簡単になったインストール

インストールはサポートされているオペレーティング・システムの間で一貫したものになり、カスタマイズされたインストール情報がインストール CD の計画アシスタントから提供されます。サイレント・インストールおよびコンソール・インストールも可能です。

Information Center

ブラウザー・ベースの Information Center には、Content Manager、Enterprise Information Portal、および Content Manager VideoCharger の資料が掲載されています。トピック関連の情報は、製品別およびタスク別 (例: 管理) に編成されています。用意されているナビゲーション機構および索引の他に、検索機能も検索の助けとなります。

アクセス支援

アクセス支援フィーチャーは、運動障害や視覚障害などの身体障害を持つユーザーが、ソフトウェア製品を問題なく使用できるよう支援します。本製品のアクセス支援の主なフィーチャーを以下に示します。

- マウスの代わりにキーボードからすべてのフィーチャーが操作可能。
- 拡張表示プロパティのサポート。
- 映像および音によって注意を喚起するためのオプション。
- 支援テクノロジーとの互換性。
- オペレーティング・システムのアクセス支援フィーチャーとの互換性。
- アクセス可能な文書フォーマット。

PeopleSoft および Siebel の統合

PeopleSoft および Siebel の両アプリケーションのユーザーは、eClient を使用してこれらのアプリケーションを構成することにより、さまざまなコンテンツ・サーバーに保管されているコンテンツにアクセスできます。

Enterprise Information Portal の紹介

保険会社や金融機関などの書類を中心とした企業の多くは、業務関連の大量のコンテンツを管理します。業務情報の管理とアクセスのための社内ソリューションのニーズは、多くの業界に共通してあります。

コンテンツ・サーバー は、社内でコンテンツを処理したり利用したりするためのメタデータと一緒に、書類、文書、および関連データを保管するためのソフトウェア・システムです。別々のコンテンツ・サーバーを効率的に接続する方法がないと、情報の重複や複数の検索を実行できるよう社員を研修するために、時間と経費を浪費することがあります。

Enterprise Information Portal では、最先端のテクノロジーによって、すべての企業リソースをワークステーションのデスクトップ上で扱うことができます。IBM Enterprise Information Portal for Multiplatforms を利用すると、さまざまなコンテンツ・サーバーを単一のクライアントを通して接続することで、企業の情報およびマルチメディア資産の価値を最大化することができます。IBM Enterprise Information Portal for Multiplatforms を使用すると、クライアント・ユーザーは、接続されたすべてのコンテンツ・サーバーへの高速な同時アクセスが可能になります。ユーザーは、情報マイニング、つまり複数のコンテンツ・サーバー (Web やイントラネットを含む) からの「インテリジェント」検索を実行することもできます。また、ビジネス・プロセス内のワークフロー・タスクを実行することもできます。

IBM Enterprise Information Portal for Multiplatforms では、アプリケーションを企業の業務に合わせてカスタマイズできます。IBM Enterprise Information Portal for Multiplatforms ツールキットを使用して、アプリケーション・プログラマーは、デスクトップのアプリケーションおよび Web ベースのアプリケーションの両方を作成できます。

この章では、IBM Enterprise Information Portal for Multiplatforms の概要を説明します。架空の保険会社の「XYZ 保険」のシナリオを使って、IBM Enterprise Information Portal for Multiplatforms のフィーチャーと機能を示します。

カスタマー情報の検索

XYZ 保険 (XYZ) は、大手の損害保険会社であり、写真、保険料支払請求、証券、精算人のメモ、専門家からの報告書、および他の書類を多量に蓄積しています。

XYZ 保険は、電子書式の医療診断および判定と一緒に、保険契約者に送付したすべてのメモを Lotus® Domino™.Doc ファイル・キャビネットに保存しています。XYZ 社は、長期保存と迅速なアクセスのために、すべての保険申告書、通知書、および請求書を、Content Manager OnDemand サーバーに保存しています。XYZ 社は、保険契約者から受け取ったすべての保険料支払請求フォーム、写真、および書状を、Content Manager for AS/400® のシステム・ホルダーに保管しています。XYZ 社は、DB2 Universal Database™ (DB2® UDB) Data Warehouse Center Information Catalog Manager に、専門家からの報告書を保管しています。また XYZ 社は、広告、宣伝、および新規ビジネスの部署でも利用できるよう、高解像度グラフィック

スなどの企業メディア資産を Content Manager に保管しています。さらに、XYZ 社は、企業内手続きなどの情報をイントラネット上に保管しています。

ニーズ

保険料支払請求、顧客連絡、および保険契約者に対する一般サービスを 1 つのサーバーのコンテンツだけで満足に処理することはできません。担当者は、すべての顧客情報にアクセスする必要があるからです。顧客サービスを拡充するには、担当者は、さまざまなコンテンツ・サーバーに同時にアクセスできなければなりません。XYZ 保険には、情報を検索して取り出せるようにコンテンツ・サーバーと企業イントラネットを接続するためのソリューションが必要です。さらに、ワークフロー・プロセッシングの使用を拡張することが必要です。

事務職から支払要求処理係や代理店にいたるまで、多種多様の社員が文書にアクセスする必要があります。XYZ 社は、ある項目ではアクセスを制限し、別の項目では無制限のアクセスを提供する必要があります。また XYZ 社には、研修の必要性を少なくするための使いやすいインターフェースも必要です。

ソリューション

XYZ 保険は、IBM Enterprise Information Portal for Multiplatforms を導入します。その理由は、その包括的な検索テクノロジーを利用して、すべてのコンテンツ・サーバーに接続して検索し、データを取り出すことができるからです。現在、XYZ 社の顧客対応センターの担当者は、電話連絡を受けてから、1 回の統合検索で保険契約者の必要情報をすべて取り出すことができます。

また XYZ 保険は、Enterprise Information Portal 情報マイニング機能を使用して、社内イントラネットから情報を取り出します。さらに、ワークフロー・プロセスの使用を拡張することを検討しています。

Enterprise Information Portal ソリューション

IBM Enterprise Information Portal for Multiplatforms は包括的な製品です。各コンポーネントが協働して、企業固有の必要に合わせたソリューションを提供します。複数層アーキテクチャーを中心とする IBM Enterprise Information Portal for Multiplatforms には、検索管理用の管理クライアント、検索実行用のクライアント (サンプルとして)、および Content Manager、Content Manager ImagePlus[®] for OS/390[®]、Content Manager OnDemand、Lotus Domino.Doc、DB2 UDB、DB2 DataJoiner[®]、および DB2 Data Warehouse Center Information Catalog Manager など、さまざまな別個のコンテンツ・サーバーに接続するためのコネクタが用意されています。さらに別のコンテンツ・サーバーがあれば、そのための追加のコネクタを作成することもできます。

3 ページの図 1 に、IBM Enterprise Information Portal for Multiplatforms の複数層アーキテクチャーの概念を示します。

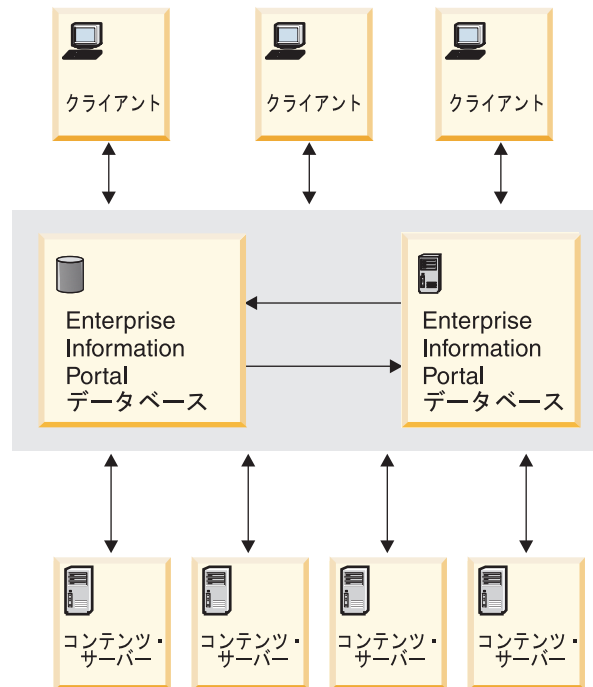


図 1. 複数層アーキテクチャー

IBM Enterprise Information Portal for Multiplatforms のアーキテクチャーでは、クライアント・アプリケーションから 1 つまたは複数のコンテンツ・サーバーに対して単一検索を実行することができます。クライアントが検索を実行するには、IBM Enterprise Information Portal for Multiplatforms 管理者によって定義された検索テンプレートを使用します。

クライアントは検索テンプレートを使用して、統合検索を実行します。統合検索とは、固有属性が、検索テンプレートで使用する統合属性にあらかじめマップされている複数のコンテンツ・サーバーに対して同時に実行される検索です。IBM Enterprise Information Portal for Multiplatforms 検索テンプレートには、検索基準が入っています。これは、各コンテンツ・サーバーの固有の属性にマップされる統合属性を参照します。IBM Enterprise Information Portal for Multiplatforms 管理者は、検索テンプレートを作成します。IBM Enterprise Information Portal for Multiplatforms は、コンテンツ・サーバーの異機種混合インターフェースの共通インターフェースとしてコネクタを提供します。これによってコンテンツ・サーバーは、データ・オブジェクトをクライアントに戻します。

IBM Enterprise Information Portal for Multiplatforms アーキテクチャーには、以下の利点があります。

- 1 回の照会で、e-business トランザクションとカスタマー・サービス・アプリケーションをサポートする複数の異なるコンテンツ・サーバーにアクセスできます。
- 情報マイニング機能によって、Web を含む複数のコンテンツ・サーバーにアクセスできます。
- ワークフロー・プロセスによって、複数の異種コンテンツ・サーバーにわたって分散しているデータにアクセスできます。

- ・ クライアント・アプリケーション開発のサポート。クライアント・アプリケーション、索引、およびデータが別個になっているため、コンテンツ・サーバーでのデータの位置に依存していません。

IBM Enterprise Information Portal for Multiplatforms コンポーネント

ここでは、IBM Enterprise Information Portal for Multiplatforms の各コンポーネントについて説明します。これらのコンポーネントは、IBM Enterprise Information Portal for Multiplatforms 製品の一部として付属しています。

管理データベース

IBM Enterprise Information Portal for Multiplatforms データベースは、DB2 UDB データベースです。これには、EIP およびそのコンポーネントの管理に必要なすべての情報が保管されます。

Enterprise Information Portal バージョン 7.1 データベースへのマイグレーション: Enterprise Information Portal バージョン 8 リリース 2 管理データベースを使用する前に、Enterprise Information Portal バージョン 7.1 からデータをマイグレーションする必要があります。

管理クライアント

システム管理者は、IBM Enterprise Information Portal for Multiplatforms 管理クライアントを使用することにより、下記のことを実行します。

- ・ 統合検索に使う各コンテンツ・サーバーを定義します。
- ・ コンテンツ・サーバー上の固有エンティティおよび固有属性を識別し、それらを統合エンティティにマッピングします。
- ・ 検索テンプレートを作成します。
- ・ 検索テンプレート、情報マイニング機能、およびワークフロー・プロセスにアクセスできるユーザーを識別および管理します。
- ・ ビジネス・ワークフロー・プロセスを定義します。

この情報は、IBM Enterprise Information Portal for Multiplatforms データベース内に保管されます。

便利のため、管理クライアントは、可能な限り IBM Enterprise Information Portal for Multiplatforms データベースと同じワークステーションまたはサーバーにインストールしてください。

その他のワークステーションに管理クライアントをインストールすることも可能であり、その数に制限はありません。そのように構成するには、下記のいずれかを実行する必要があります。

- ・ 管理クライアントがインストールされている各ワークステーション上に DB2 Client サポートをインストールし、DB2 Client Configuration Assistant を使用して、その上のシステム管理データベースへのアクセスを構成します。
- ・ IBM Enterprise Information Portal for Multiplatforms データベースがインストールされている RMI サーバーを始動することによって、リモート・メソッド呼び出し (RMI) を使用します。¥CMBROOT¥cmbclient.ini ファイルの中でこのサーバーが指定されていることを確認してください。この INI ファイルの位置は、

cmbcmenv.properties ファイルの中で CMCOMMON キーワードによってディレクトリーが指定されている場所です。また、このファイルは、cmbcmenv.properties ファイルの中で CMCOMMON_URL キーワードによって指定された URL 上に置かれる場合もあります。

コネクタ

クライアント・アプリケーションは、コネクタ・クラスを使って、コンテンツ・サーバーと IBM Enterprise Information Portal for Multiplatforms データベースにアクセスすることができます。IBM Enterprise Information Portal for Multiplatforms には、以下のコネクタが付属しています。

- リレーショナル・データベース・コネクタ (DB2、JDBC、ODBC)
- 統合コネクタ (IBM Enterprise Information Portal for Multiplatforms データベースへのコネクタ)
- Content Manager バージョン 8 リリース 2
- Content Manager バージョン 7 リリース 1 コネクタ
- Content Manager OnDemand コネクタ
- Content Manager ImagePlus for OS/390 コネクタ
- Content Manager for AS/400 コネクタ
- Lotus Domino.Doc コネクタ
- Extended Search コネクタ

統合コネクタには、IBM Enterprise Information Portal for Multiplatforms データベース用のコネクタ・クラスが入っています。各コンテンツ・サーバー・コネクタには、該当するコネクタ・クラスが入っています。

Java 環境では、Java バージョンのコネクタにはローカルとリモートがあります。C++ では、ローカル・コネクタしかありません。ローカル・コネクタは、種々のコンテンツ・サーバーに直接接続するために使用するコネクタ・クラスの集まりです。ローカル・コネクタの位置は、IBM Enterprise Information Portal for Multiplatforms デスクトップ・クライアント上または IBM Enterprise Information Portal for Multiplatforms RMI サーバー上です。リモート・コネクタは、RMI サーバーまたは RMI サーバー・プール・メンバーを経由してコンテンツ・サーバーに接続するために使用されます。リモート・コネクタを使用すると、コンテンツ・サーバーに直接接続する必要がなくなります。

IBM eClient

eClient は、Content Manager (すべてのプラットフォーム)、Content Manager OnDemand (すべてのプラットフォーム)、および Content Manager ImagePlus for OS/390 に保管された文書にアクセスするための、ブラウザー・ベースのユーザー・インターフェースです。

情報マイニング

情報マイニングは、コンテンツ・サーバー上のテキスト文書に隠れている情報を検出するための、言語サービスを提供します。テキスト文書の処理時に、要約、カテゴリー化、および検索可能なメタデータ (データについての情報) を作成します。IBM Enterprise Information Portal for Multiplatforms には、シン・クライアントで情

報マイニング機能を使用する方法を示すサンプルが付属しています。情報マイニング用に、独自のデスクトップ・クライアントまたはシン・クライアントを作成することができます。

IBM Web Crawler

IBM Web Crawlerでは、Web コンテンツの検索およびインポートが可能です。検索結果およびメタデータは、情報マイニング・ツールを使用して分析および分類が可能です。IBM Web Crawlerは、HTTP、FTP、NTP、Lotus および Domino サーバーをクロールすることができます。

ワークフロー

Enterprise Information Portal を使用すると、業務におけるワークフローを制御することができます。Enterprise Information Portal ワークフロー機能を使用することによって、ワークグループ、部署、または企業のワークフロー・プロセスを定義および実行できます。グラフィカル・ビルダーを使うことにより、Enterprise Information Portal ワークフロー・ビルダー内に包括的でわかりやすい、ワークフロー・プロセスのグラフィカルな表現を作成できます。それによりユーザーは、定義済みワークフロー・プロセスを使用し、プログラマーの開発したクライアントまたは Enterprise Information Portal シン・クライアントのサンプルを使用することにより、それぞれの作業を実行できます。

Content Manager バージョン 7 テキスト検索エンジン

このフィーチャーを使用すると、Content Manager バージョン 7 に保管されている文書の索引付け、検索、および取り出しを自動的に行うことができます。文書を見つけるには、語または句を検索します。

制約事項: Text Search Server およびクライアントは、旧バージョンの Content Manager サーバーでのみ構成および実行することのできる、オプションの Content Manager 機能です。旧バージョンの Content Manager サーバーを使用しない場合、この機能はインストールしないでください。

Content Manager バージョン 7 イメージ検索サーバーおよびクライアント

イメージ検索サーバーは、IBM の QBIC[®] (イメージ・コンテンツによる照会) テクノロジーを使用しており、色やテクスチャーなどの視覚的なプロパティごとにオブジェクトを検索できます。

制約事項: イメージ検索サーバーの機能は、Enterprise Information Portal バージョン 8 と Content Manager バージョン 8 ではサポートされません。バージョン 8.1 Content Manager をお使いの場合は、この機能を引き続き使用できます。

バージョン 8.2 API の新機能

広範囲に渡るコードのサンプル

Content Manager バージョン 8.2 の機能に合わせてコードのサンプルが更新されました。また、コードのサンプルは、読みやすいように枠 (言語別のラベル付き) で囲んで示されています。

DB2 DataJoiner の制限

お使いのコンテンツ・サーバーが DB2 Universal Database バージョン 8.1

を必要とする場合は、バインディングの問題が原因で EIP DataJoiner コネクターが機能しません。これは、DataJoiner 2.1 が DB2 バージョン 7 バインド・ファイル内の SQL ステートメントを認識しないからです。

この問題を解決するには、DataJoiner Web サイト (<http://www.software.ibm.com/data/datajoiner/>) にある FAQ の質問 "Why am I having problems with binding on DB2 Universal Database Version 7 when connecting to a DataJoiner server?" をお読みください。この回答で指示されている、DataJoiner バージョン 2.1.1 サーバーに接続するための手順は次のとおりです。

1. 2 つのバインド・ファイルをダウンロードし、db2cliws_dj.bnd および db2clprp_dj.bnd にリネームします。
2. 以下の DB2 コマンドを入力してください。

```
db2cmd
db2 connect to user using
db2 bind db2cliws_dj.bnd grant public
db2 bind db2clprp_dj.bnd grant public
```

DB2 Data Warehouse Manager Information Catalog Manager の制限

IC コネクターを使用するには DB2 ユニバーサル・データベースのバージョン 7.2 が必要で、DB2 UDB バージョン 8.1 の環境では使用できません。

統合フォルダー (Java のみ)

Enterprise Information Portal バージョン 8 リリース 2 は、統合フォルダーを保管できる特別な統合エンティティを提供しています。この統合フォルダーは、統合照会からの結合された結果を保管できます (例えば、Content Manager からの文書と OnDemand からの関連文書)。この結果は、ワークフローに直接送ることができます。

Microsoft Visual Studio .NET のサポート

Enterprise Information Portal と Content Manager のバージョン 8.2 (およびそれ以降) の API は、Microsoft Visual Studio .NET をサポートしています。

バージョン 8.1 API の新機能

Enterprise Information Portal バージョン 8 リリース 2 では、異なるコンテンツ・サーバーへの前例のないアクセスが可能になりました。新機能およびコンポーネントには次のものがあります。

- XML インポート機能

XML を使用することにより、DDO および XDO を介して Content Manager にコンテンツをインポートおよびエクスポートできるようになりました (Java API を使用)。

- 改善されたインストール手順

- リレーショナル・データベース用の追加のコネクター

Enterprise Information Portal には、DB2 UDB、DB2 DataJoiner、DB2 Data Warehouse Manager Information Catalog Manager、その他のデータベースへの JDBC または ODBC ドライバーによるリレーショナル・データベース・コネクターが用意されました。

- 情報マイニングおよび検索の拡張機能
情報マイニングのテキスト検索の機能が拡張され、特定のカテゴリの文書だけに制限するための柔軟な照会を使用できるようになりました。
- ワークフロー機能
Enterprise Information Portal ワークフロー機能を使用することによって、ワークグループ、部署、または企業のワークフロー・プロセスを定義および実行できます。
- 統合レベルのアクセス制御
Enterprise Information Portal 情報マイニングおよびワークフロー・プロセスへのアクセスを、特権セットおよびアクセス制御リストを使用して制御することができます。付加的なデータ・アクセス制御は、各コンテンツ・サーバーのアクセス制御機能によって管理されます。
- Content Manager の追加サポート
 - コンテンツ・クラスのリスト、追加、検索、更新、および削除
 - オブジェクト・コンテンツの非同期検索

新規および変更された Java クラス

Java 共通クラスは、すべてのコネクタで使用されます。このパッケージには、コネクタが使用するインターフェース (dkDatastore など) および抽象クラス (dkAbstractDatastore および dkXDO など)、また、具象クラス (DKDDO など) が含まれています。

新規クラス:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt
- dkAbstractConfigurationMgmt
- dkAbstractDatastoreAdmin
- dkAbstractDataObjectBase
- dkAbstractPrivilege
- dkAbstractPrivilegeGroup
- dkAbstractPrivilegeSet
- dkAbstractResultSetCursor
- dkAbstractUserDef
- dkAbstractUserMgmt
- dkAttrGroupDef
- dkAuthorizationMgmt
- dkCheckableObject
- DKChildCollection
- dkConfigurationMgmt
- DKLinkCollection
- dkPersistentCheckableObject

- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

変更されたクラス:

- dkAbstractDatastore
- dkAbstractDatastoreDef
- dkAbstractDatastoreExt
- dkAbstractEntityDef
- dkAccessControlList
- dkDataObjectBase
- dkDatastore
- dkDatastoreAdmin
- dkDatastoreDef
- dkDatastoreExt
- DKDDO
- dkEntityDef
- dkPersistentObject
- dkPrivilegeSet
- dkSearchTemplate
- dkSchemaMapping
- dkUserManagement

Enterprise Information Portal バージョン 8 におけるクラスの振る舞いの変更

EIP 8.2 では、一部のクラスおよびクラス・コンポーネントの振る舞いが EIP 7.1 から変更されました。このセクションでは、これらの変更について説明します。詳しくは、「オンライン API 解説書」を参照してください。

- dkIterator: next() はイテレーターを次の項目に送り、その項目を取得します。previous() は前の項目に移動して、その項目を取得します。
- dkXDO: getPidObject() および setPidObject(DKPid pid)

新規および変更された C++ クラス

C++ 共通クラス・パッケージは、すべてのコネクターで使用されます。このパッケージには、コネクターが使用するインターフェース (dkDatastore など) および抽象クラス (dkAbstractDatastore および dkXDO など)、また、具象クラス (DKDDO など) が含まれています。

新規クラス:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt

- dkAbstractConfigurationMgmt
- dkAbstractDataObjectBase
- dkAbstractDatastoreAdmin
- dkAbstractPrivilege
- dkAbstractPrivilegeGroup
- dkAbstractPrivilegeSet
- dkAbstractResultSetCursor
- dkAbstractUserDef
- dkAbstractUserMgmt
- dkAttrGroupDef
- dkAuthorizationMgmt
- dkCheckableObject
- DKChildCollection
- dkConfigurationMgmt
- DKLinkCollection
- dkPersistentCheckableObject
- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

変更されたクラス:

- dkAbstractDatastore
- dkAbstractDatastoreDef
- dkAbstractDatastoreExt
- dkAbstractEntityDef
- dkAccessControlList
- dkDataObjectBase
- dkDatastore
- dkDatastoreAdmin
- dkDatastoreDef
- dkDatastoreExt
- DKDDO
- dkEntityDef
- dkPersistentObject
- dkPrivilegeSet
- dkSchemaMapping
- dkSearchTemplate
- dkUserManagement

Enterprise Information Portal 8.2 におけるクラスの振る舞いの変更

EIP 8.2 では、一部のクラスおよびクラス・コンポーネントの振る舞いが EIP 7.1 から変更されました。このセクションでは、これらの変更について説明します。詳しくは、「オンライン API 解説書」を参照してください。

- AIX 上で DKAny に DKString を使用する方法が変わりました。DKAny から DKString を検索するには、DKAny 変数に DKAny を入れて、これを DKString 変数に割り当てる必要があります。

例: この例では、コンテンツ・サーバーからの dkResultSetCursor fetchNext() API 呼び出しから DDO が検索され、次のように DDO 変数が設定されているものとします。この例では、DDO 内のデータ項目内をループして、ストリング値を検出します。

```
DKDDO *ddo = NULL; DKAny any; DKString strTmp;

unsigned short data_id = 0;
unsigned short count = 0;

count = ddo->dataCount();
for (data_id = 1; data_id <= count; data_id++)
{
    any = ddo->getDataId(data_id);
    if (any.typeCode() == DKAny::tc_string)
    {
        strTmp = any.toString(); // This is how to get a DKString from a DKAny
    }
}
```

- dkXDO をインプリメントするインターフェースが、getPidObject メソッドと setPidObject メソッドに関連して変更されました。

Enterprise Information Portal アプリケーション・プログラミングの概念

Enterprise Information Portal は、オブジェクト指向 (OO) のアプリケーション・プログラミング・インターフェース (API) を提供します。この API を使用して照会アプリケーションを作成し、リレーショナル・データおよびマルチメディア・データにアクセスしてデータを表示できます。この章では、これらの API を Enterprise Information Portal アーキテクチャーに適合させる方法について簡単に概説し、この API の基礎となっているオブジェクト指向のプログラミングの概念について説明します。

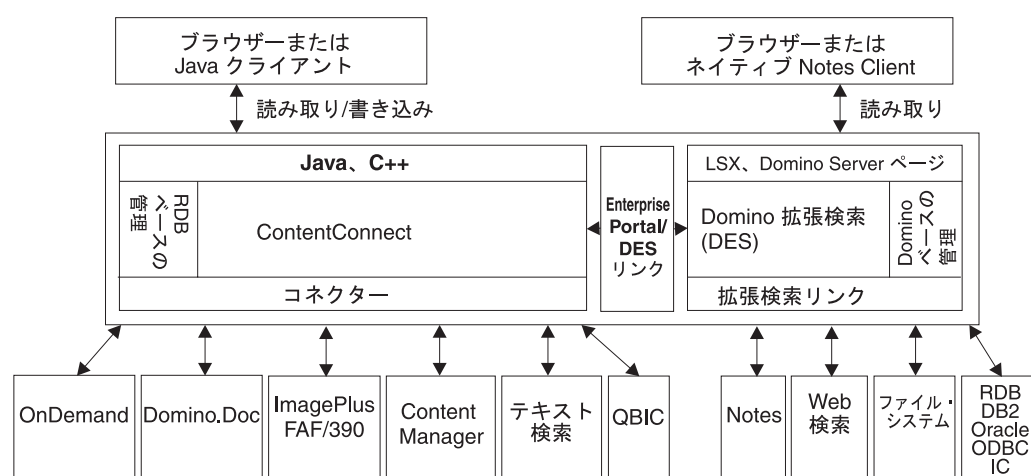


図 2. Enterprise Information Portal の構成

コンテンツ・サーバーを介したデータ・アクセスについて

コンテンツ・サーバーは、DDO/XDO プロトコルと互換性のあるデータ・リポジトリです。コンテンツ・サーバーは、ユーザー・セッション、接続、トランザクション、カーソル、および照会をサポートします。本書で説明しているアプリケーション・プログラミング・インターフェース (API) およびクラス・ライブラリーを使用しているアプリケーションは、DDO の追加、検索、更新、および削除など、コンテンツ・サーバーでサポートされている機能を実行できます。Enterprise Information Portal では、下記のコンテンツ・サーバーがサポートされています。

- Content Manager バージョン 8 リリース 2
- Content Manager バージョン 7 リリース 1
- Domino.Doc
- Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand

- VisualInfo™ for AS/400
- DB2
- DB2 DataJoiner
- 情報カタログ
- DB2 Warehouse Manager Information Catalog Manager
- JDBC/ODBC サーバー

Enterprise Information Portal を使用するアプリケーションは、統合コンテンツ・サーバーを作成できます。これは共通サーバーとして機能します。Enterprise Information Portal データベース・クラスにより、複数のコンテンツ・サーバーにわたる統合検索および更新が可能になります。

Enterprise Information Portal 統合コンテンツ・サーバーおよび各コンテンツ・サーバーは、別々のスキーマを持っています。複数の異種コンテンツ・サーバーを一体化して 1 つの統合システムにするには、変換およびマッピングが必要です。

スキーマ・マッピング機能は、各コンテンツ・サーバーにスキーマ情報を提供します。スキーマ・マッピングによって提供される情報は、統合検索、統合コレクション、および Enterprise Information Portal システム管理の際に使用されます。

Enterprise Information Portal は、スキーマやマッピング、その他の管理情報を、その管理データベースの中に保管します。

動的データ・オブジェクトの概念について

Object Management Groups (OMG) の CORBA Persistent Object Service および Object Query Service Specification に準拠して、Enterprise Information Portal は、動的データ・オブジェクト (DDO) と、その拡張である拡張データ・オブジェクト (XDO) をインプリメントしています。これらは、CORBA Persistent Data Service (PDS) プロトコルの一部です。DDO および XDO の概念は、いずれか 1 つのコンテンツ・サーバーに固有のものではなく、Enterprise Information Portal によってサポートされるデータベース管理システム内のデータ・オブジェクトを表すために使用できます。

動的データ・オブジェクトは、データをコンテンツ・サーバーから取り出したり、そこに保管したりするために使用するインターフェースです。DDO はアプリケーション内に存在し、アプリケーションが終了した後は存在しなくなります。

動的データ・オブジェクト (DDO)

DDO は、コンテンツ・サーバーに依存しないオブジェクトの永続データを表すものです。このオブジェクトの目的は、すべてのデータを単一の永続オブジェクトに入れることです。さらに DDO は、データ・ストアから永続データを検索するか、またはコンテンツ・サーバーに永続データをロードするためのインターフェースです。

DDO には、単一の永続 ID (PID)、オブジェクト・タイプ、および基数がデータ・カウントと呼ばれるデータ項目のセットがあります。それぞれのデータ項目は、名

前、値、ID、1 つ以上のデータ・プロパティ、およびデータ・プロパティ・カウントを持つことができます。それぞれのデータ・プロパティは、ID、名前、および値を持つことができます。

例えば DDO は、データベース・テーブルの行を表して、その列が DDO のデータ項目およびそのプロパティを表すようにすることができます。DDO には、従来のものではないデータ・タイプを表す、1 つ以上の拡張データ・オブジェクト (XDO) を含めることができます。図 3 は、動的データ・オブジェクトおよびデータ項目を示します。

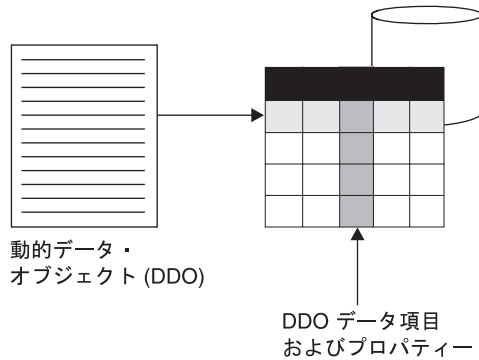


図 3. 動的データ・オブジェクトおよび項目

拡張データ・オブジェクト (XDO)

XDO は、複合マルチメディア・データを表すものです。これには例えば、Content Manager にイメージや文書を保管するリソース項目や、IBM DB2 エクステンダーなどの、リレーショナル・データベースのオブジェクト関連機能によって導入される新しいデータ・タイプなどがあります。

XDO は、複合タイプのマルチメディア・データを保管し、アプリケーションでのデータ・タイプの動作をインプリメントする機能を提供することによって、DDO を補います。XDO は、DDO に含めるかまたは DDO が所有して、複合マルチメディア・データ・オブジェクトを表すことができます。

XDO には、データ・タイプおよび ID などの情報を表すプロパティのセットがあります。XDO は、単独の動的オブジェクトにすることもできます。16 ページの図 4 は、XDO の例を示しています。

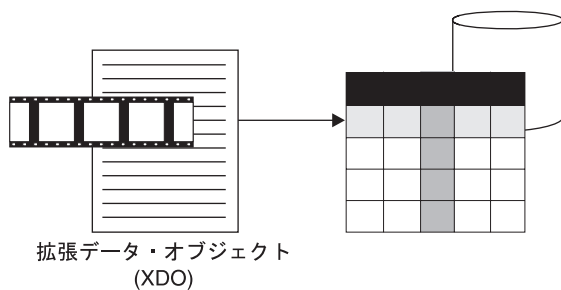


図 4. 拡張データ・オブジェクト (XDO)

マルチメディア・コンテンツの表示

DDO および XDO を使用すれば、どのようなタイプおよび構造のデータ・オブジェクトでも表すことができます。例えば、動画は DDO で表すことができます。この DDO には、複数のデータ項目 (Director_Name または Movie_Title などの動画の属性を表す) や、マルチメディア XDO (ビデオ・クリップまたは静止イメージなど、動画のマルチメディア・データを表す) が含まれます。

Content Manager 8.2 では、DDO はオブジェクト (文書やイメージなど) を記述するすべてのメタデータで構成されます。XDO は、DDO の機能をさらに拡張して、リソース・コンテンツをサポートしています。リソース・コンテンツは、バイナリー・データやテキストからビデオやオーディオのストリームまで、あらゆるタイプのコンテンツです。XDO をインプリメントする (また、XDO でなければならぬ) 項目は、DDO でもあり、DDO に備わっているすべての機能に加えて、XDO に備わっている (備わっていない) 機能をサポートします。

コンテンツ・サーバーおよび DDO について

DDO は、作成されるとコンテンツ・サーバーと動的に関連付けられます。DDO とコンテンツ・サーバーとの関連は、DDO の PID で確立されます。

一般に Enterprise Information Portal アプリケーションは、コンテンツ・サーバーにデータを保管したりそこから取り出したりするために、以下の 5 つのステップを実行します。

1. コンテンツ・サーバーを作成します。
2. コンテンツ・サーバーへの接続を確立します。
3. 操作対象の DDO を作成し、コンテンツ・サーバーとその DDO を関連付けます。
4. 適切なメソッドを使用して DDO を追加、検索、更新、および削除します。
5. 接続をクローズし、コンテンツ・サーバーを破棄します。

DDO/XDO と属性値および項目パーツとの比較

DDO は、Enterprise Information Portal 内の項目に対応します。DDO のオブジェクト・タイプは、項目に関連付けられた項目タイプに対応します。DDO のデータ項目は、項目の属性に対応します。例えば、Content Manager では、項目タイプは属性のセットを使用して作成され、項目は常に項目タイプによって索引付けされます。

DDO は、Enterprise Information Portal 内の項目パーツに対応する 1 つまたは複数の XDO を保持することができます。

永続 ID (PID) の説明

永続 ID (PID) は、コンテンツ・サーバー内の永続オブジェクトを一意的に識別します。DDO の PID は、項目 ID、コンテンツ・サーバー名、およびその他の関連情報から構成されています。DDO をコンテンツ・サーバーに追加すると、固有の PID がシステムによって DDO に割り当てられます。

DDO は、コンテンツ・サーバーに保管するまたはそこから取り出す永続データへの動的インターフェースであるので、異なる DDO が同じ永続データ・エンティティを表す場合があります。したがって、複数の DDO が同じ PID を持つ場合があります。例えば、ある DDO を、コンテンツ・サーバーにデータ・エンティティを移動させて永続的にデータを保管するために作成し、別の DDO を、その同じコンテンツ・サーバーから変更用にチェックアウトする同じデータ・エンティティを保持するために作成することができます。この場合、これら 2 つの DDO は、同じ PID 値を共有します。

統合コンテンツ・サーバーおよび統合検索の処理

統合検索 とは、1 つまたは複数のコンテンツ・サーバーのデータを検索するプロセスのことです。統合検索には、DKDatastoreFed オブジェクトを使用します。統合検索は、統合検索をサポートする dkDatastore、dkDatastoreDef、その他の関連するクラスの特実の実装であるクラスと連携して動作します。特定の統合クラスは、Enterprise Information Portal フレームワークの一部として、他の共通クラス（照会、コレクション、およびデータ・オブジェクトなど）と連携して動作します。

統合されたクラスは、Content Manager ImagePlus for OS/390 や Domino.Doc など、異なるコンテンツ・サーバーに渡って動作します。それらのクラスには、統合検索を実行したり、複数のコンテンツ・サーバーにアクセスしたりするための汎用の関数が含まれています。統合文書モデル と呼ばれるこの共通ビューについて、図 5 に示します。

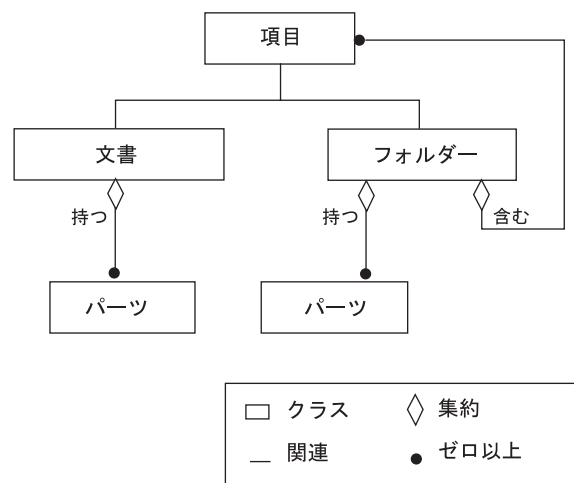


図 5. 統合文書ビュー

項目は文書またはフォルダーのいずれかです。文書には 0 個以上のパーツが含まれます。フォルダーには 0 個以上の項目（文書または他のフォルダー）が含まれます。

すべてのコンテンツ・サーバーが統合文書モデルをサポートするとは限りません。例えば、DB2 データベースには、フォルダーまたはパーツがありません。項目は、DB2 またはその他のリレーショナル・データベースのテーブル内にある行にマップしており、コンテンツ・サーバーが文書やフォルダーをサポートしない場合に使用されます。

一般に、文書はプログラム中で動的データ・オブジェクト (DDO) によって表されます。これは、データをコンテンツ・サーバーに転送したり、またコンテンツ・サーバーからデータを転送したりする自己記述式のデータ・オブジェクトです。DDO 自体の構造は汎用性のあるものであり、さまざまなモデルをサポートします。これ

は統合文書モデルだけに限られているわけではありません。このような柔軟性のおかげで、DDO はデータを異なるコンテンツ・サーバーで、それぞれ独自のデータ・モデルで表すことができます。

エンティティとは、いくつかの属性で構成されるコンテンツ・サーバー・オブジェクトのことです。属性は、コンテンツ・サーバーのメタデータのために使うラベルです。例えば、Domino.Doc コンテンツ・サーバーの中のプロファイル、フィールド、またはキーワードは属性です。

サポートするモデルについて説明するための用語は、コンテンツ・サーバーごとに異なっています。表3に、さまざまなコンテンツ・サーバーで使用する用語と統合モデルとの関係を示します。

表3. 各コンテンツ・サーバーごとの用語の対応

コンテンツ・サーバー	データ・ソース	エンティティ	属性	ビュー
Content Manager 8.2	ライブラリー・サーバー	項目タイプ	属性	項目タイプ・ビューまたは項目タイプ・サブセット
Content Manager 7.1	ライブラリー・サーバー	索引クラス	<ul style="list-style-type: none"> 属性 キー属性 	索引クラス・ビュー
OnDemand	OnDemand サーバー	<ul style="list-style-type: none"> アプリケーション・グループ フォルダー 	<ul style="list-style-type: none"> フィールド 基準 	N/A
ImagePlus	ImagePlus for OS/390 サーバー	エンティティ	属性	N/A
Content Manager for AS/400	Content Manager for AS/400 サーバー	索引クラス	属性	索引クラス・ビュー
Domino.Doc	Domino サーバー	ライブラリー ルーム キャビネット バインダー	プロファイル フィールド キーワード	N/A
Extended Search	Extended Search サーバー	データベース名	データベース名	N/A
リレーショナル・データベース	IBM DB2 UDB、JDBC、ODBC、IBM DB2 DataJoiner	テーブル	列	ビュー

表3. 各コンテンツ・サーバーごとの用語の対応 (続き)

コンテンツ・サーバー	データ・ソース	エンティティ	属性	ビュー
情報カタログ	DB2 Warehouse Manager Information Catalog Manager	索引クラス	プロパティ	
統合コンテンツ・サーバー	マッピング・サーバー	対応する統合エンティティ	対応する統合属性	検索テンプレート
統合フォルダーを保管できる統合コンテンツ・サーバー	サーバー	統合エンティティ	統合属性	統合フォルダー

図6 は統合検索を示しています。統合検索では Enterprise Information Portal 統合コンテンツ・サーバーを使用して、検索テンプレートでの作業を行います。統合コンテンツ・サーバーはその後個々のコンテンツ・サーバーの検索を呼び出し、コンテンツ・サーバーに対して実際の検索を実行します。この関連は、スキーマ・マッピングにより確立されます。

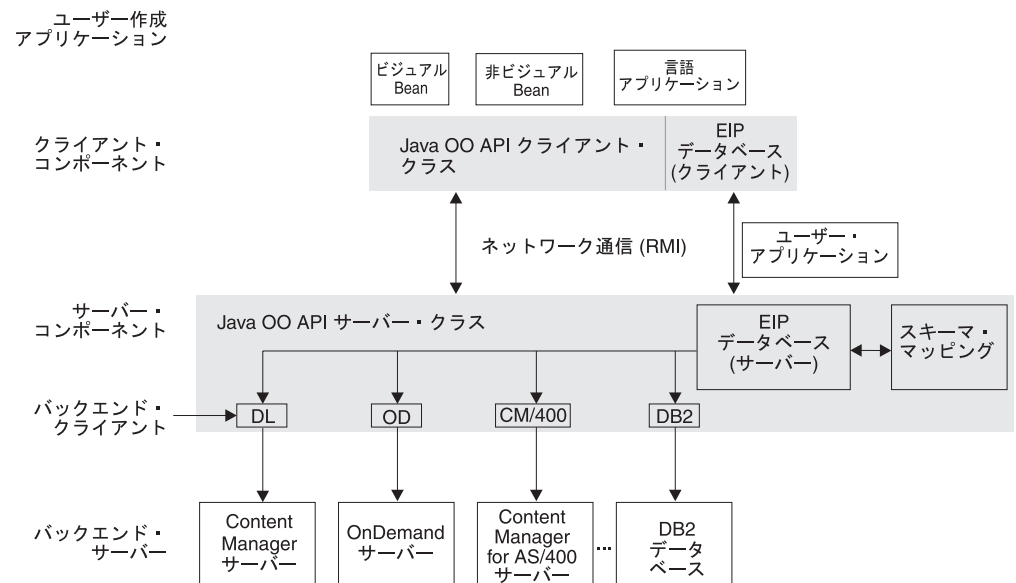


図6. 統合検索の構造

ローカル・コネクターまたはリモート・コネクターを使用して、統合コンテンツ・サーバーをコンテンツ・サーバーに接続でき、この通信には RMI を使用できます。また、API クラスを基礎としてアプリケーションを開発できます。

統合スキーマ・マッピング

スキーマ・マッピング は、コンテンツ・サーバー内のスキーマと、ユーザーがアプリケーション内で処理する項目の構造との間の対応を表します。統合スキーマ は、Enterprise Information Portal 統合コンテンツ・サーバーの概念スキーマです。この統合データ・ストアは、統合コンテンツ・サーバーの概念と、参加している各コンテ

ンツ・サーバーの概念との間の対応を定義します。スキーマ・マッピングは、データの物理的な保管方法と、ユーザーが望むアプリケーションでのデータの処理方法との間の相違を処理します。

マッピング情報は、メモリー内ではスキーマ・マッピング・クラスで表されます。

統合コンテンツ・サーバー・マッピング・コンポーネントの使用

統合コンテンツ・サーバーは、エンティティーおよび属性のマッピングのためのスキーマ・マッピング情報以外に、以下の情報にもアクセスできなければなりません。

ユーザー ID とパスワードのマッピング

単一ログオン機能をサポートするために、Enterprise Information Portal の各ユーザー ID を、各コンテンツ・サーバーの対応するユーザー ID にマッピングできます。

コンテンツ・サーバー登録

各コンテンツ・サーバーを登録することにより、Enterprise Information Portal がそれを見つけてログオンできるようにする必要があります。

ユーザー ID およびコンテンツ・サーバー情報は、Enterprise Information Portal 管理データベースの中に保管されています。

統合照会の実行

API を使用して統合検索を実行するには、まず統合照会ストリングを作成します。その後、下記のさまざまな方法で照会を作成したり実行したりできます。

- 統合照会オブジェクト `DKFederatedQuery` を作成し、それを照会ストリングに渡します。その後そのオブジェクトで `execute` メソッドまたは `evaluate` メソッドを呼び出し、照会を処理することができます。
- 照会ストリングを統合コンテンツ・サーバーの `execute` メソッドまたは `evaluate` メソッドに渡すことにより、照会を直接処理することができます。

照会ストリングは構文解析されて、統合照会の形式になります。この形式は、本質的にコンテンツ・サーバーに依存しない照会の表現です。統合照会形式は、統合検索の入力データです。

グラフィカル・ユーザー・インターフェース (GUI) ベースのアプリケーションからの照会は構文解析する必要がなく、対応する統合照会形式を直接構成することができます。

統合検索の処理において、Enterprise Information Portal は下記のステップを実行します。

- 正規の照会形式を、各コンテンツ・サーバーで実行されるそれぞれ固有の照会に変換します。変換情報は、スキーマ・マッピングから取得します。
- 統合エンティティーおよび統合属性を、それぞれのコンテンツ・サーバーの固有エンティティーおよび固有属性に変換します。この処理では、スキーマ・マッピングに記述されているマッピング・メカニズムおよびスキーマ・メカニズムが使用されます。

- 固有の照会の構成時に、フィルター操作によって関係するデータのみを抽出します。
- 固有の照会を形成し、それを個々のコンテンツ・サーバーに実行依頼します。

各コンテンツ・サーバーは、実行依頼された照会を実行します。結果が統合照会に戻されます。統合照会に戻された結果は、下記のようにして処理することができます。

- マッピング情報に従って、固有エンティティおよび固有属性を、統合エンティティおよび統合属性に変換します。
- 結果をフィルターにかけ、要求したデータだけが含まれるようにします。
- それぞれのコンテンツ・サーバーからの結果をマージして統合コレクションにします。

統合検索の結果は、統合コレクションとして戻されます。イテレーターを作成すれば、それぞれのコレクションのメンバーにアクセスすることができます。イテレーターの `next` メソッドを呼び出すごとに、1 つの `DKDDO` オブジェクトが戻されます。このオブジェクトは、コンテンツ・サーバーに依存しない動的データ・オブジェクトです。

統合コレクションには、照会結果をコンテンツ・サーバーごとに分ける機能があります。統合コレクションで `createMemberIterator` メソッドを呼び出して、順次イテレーターを作成します。この順次イテレーターを使用すれば、メンバー・コレクションである `DKResults` オブジェクトにアクセスし、それらを個別に処理できます。

統合検索のコンポーネントとそれらの関係を、図7 に示します。

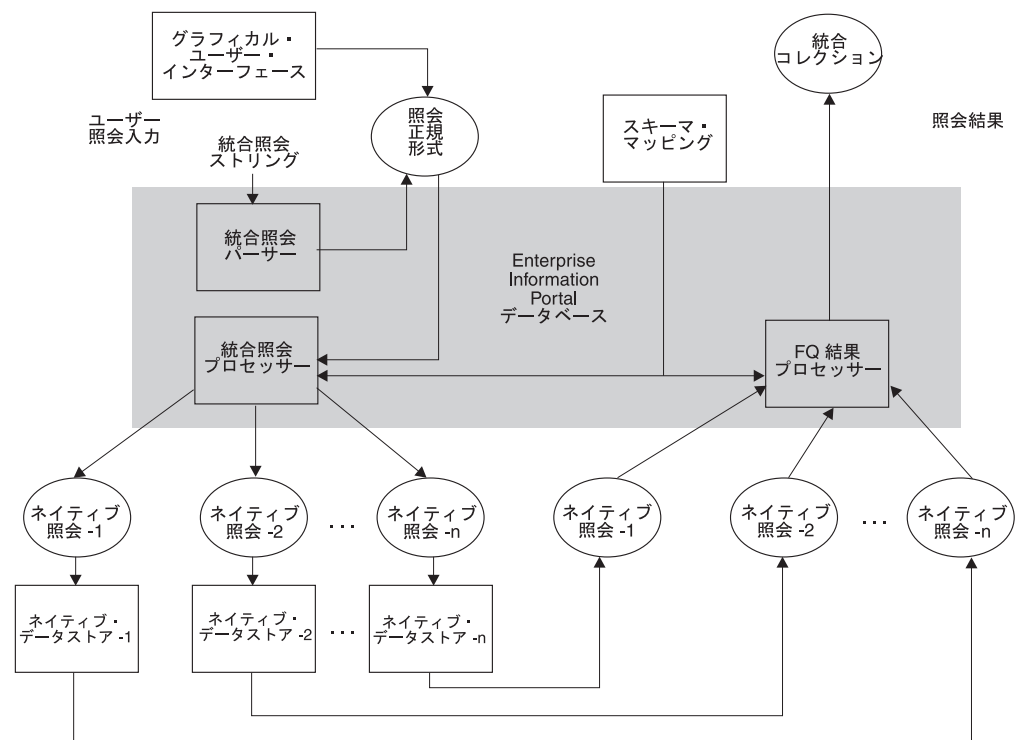


図7. 統合照会処理

統合照会の構文

統合照会を作成する場合は、下記に示す構文を使う必要があります。統合コンテンツ・サーバーは、イメージ照会をサポートしていません。

```
PARAMETRIC_SEARCH=([ENTITY=entity_name,]
                    [MAX_RESULTS=maximum_results,]
                    [COND=(conditional_expression)]
                    [; ...]
                    );
[OPTION=([CONTENT=yes_no_attronly]
        )]

[and

TEXT_SEARCH=(COND=(text_search_expression)
              );
[OPTION=([SEARCH_INDEX={search_index_name | (index_list) };]
        [ASSOCIATED_ENTITY={associated_entity_name});]
        [MAX_RESULTS=maximum_results;]
        [TIME_LIMIT=time_limit]
        )]

]
```

統合検索では NOT 演算子はサポートされていません。

統合照会ストリングの例

LIKE 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_DGSAMP71, MAX_RESULTS = 5,
COND = (fName LIKE '%'))"
```

LIKE 演算子および > 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 5,
COND = (fJTitle LIKE 'Java%' AND fJNumPages > 20) )"
```

LIKE 演算子および < 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 5,
COND = (fJTitle LIKE 'Java%' AND fJNumPages < 20) )"
```

BETWEEN 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJNumPages BETWEEN 5 200) )"
```

MAX_RESULTS をゼロに設定すると、すべての結果が戻されます。

NOTBETWEEN 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJNumPages NOTBETWEEN 5 100) )"
```

IN 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJArticleTitle IN ('Java', 'Multi-Disk B-trees.',
'On Beyond Data.', 'IBM')) )"
```

NOTIN 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJArticleTitle NOTIN ('Java', 'Multi-Disk B-trees.',
'On Beyond Data.', 'IBM')) )"
```


== 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,  
COND = (fJEditorName == 'Harth') )"
```

<> 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,  
COND = (fJSectionTitle <> 'not available') )"
```

AND および OR 演算子を使用した統合パラメトリック照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,  
COND = ((fJTitle LIKE '%Java%') OR ((fJEditorName<>NULL) AND  
(fJArticleTitle LIKE 'Computer%')))) ); OPTION = (CONTENT = YES)"
```

CONTAINS_TEXT_IN_CONTENT 演算子を使用した統合パラメトリック照会

この例では、コンテンツ内のテキストが検索されます。コンテンツは、単語であっても句であってもかまいません。これが有効になるのは、テキスト検索可能な統合エンティティー (FedTextResource) が、Content Manager バージョン 8 のテキスト検索可能な項目タイプ、または Extended Search のテキスト検索可能なエンティティーにマップされている場合のみです。

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource, MAX_RESULTS = 6,  
COND = ( CONTAINS_TEXT_IN_CONTENT 'XML' ) ); OPTION =  
( CONTENT = YES )"
```

CONTAINS_TEXT 演算子を使用した統合パラメトリック照会

属性値内のテキストを検索します。テキストは、単語であっても句であってもかまいません。これが有効になるのは、テキスト検索可能な統合属性 (fJTitle) が、Content Manager バージョン 8 のテキスト検索可能な属性、または Extended Search のテキスト検索可能な属性にマップされている場合のみです。

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,  
COND = (fJTitle CONTAINS_TEXT 'Java') )"
```

統合テキスト照会

コンテンツ内のテキストを検索します。テキストは、単語であっても句であってもかまいません。ASSOCIATED_ENTITY キーワードは、統合エンティティーがテキスト検索可能である場合に限り適用できます。これが有効になるのは、テキスト検索可能な統合エンティティー (FedEntity) が、Content Manager バージョン 8 のテキスト検索可能な項目タイプ、または Extended Search のテキスト検索可能なエンティティーにマップされている場合のみです。

```
"TEXT_SEARCH = ( COND = ('XML') ); OPTION = ( ASSOCIATED_ENTITY=FedEntity )"
```

統合テキスト照会

コンテンツ内のテキストを検索します。これは、単語であっても句であってもかまいません。統合テキスト索引 FedTMINDEX は、Content Manager バージョン 7 Text Miner の検索索引にマップされます。SEARCH_INDEX キーワードは、このタイプのマッピングに対してのみ適用可能です。この条件で語または句を指定するには、テキスト検索をサポートしている Content Manager バージョン 7 サーバーを定義する際に、構成ストリングを GENFEDTEXTQRY=YES に設定する必要があります。

```
"TEXT_SEARCH = ( COND = ('operating system') );  
OPTION = ( SEARCH_INDEX = FedTMINDEX)"
```

統合テキスト照会

Content Manager バージョン 7、Content Manager バージョン 8 および Extended Search に渡って、コンテンツ内のテキストを検索します。これは、単語であっても句であってもかまいません。統合テキスト索引 FedTMINDEX は、Content Manager バージョン 7 Text Miner の検索索引にマップされます。SEARCH_INDEX キーワードは、このタイプのマッピングに対してのみ適用可能です。この条件で語または句を指定するには、テキスト検索をサポートしている Content Manager バージョン 7 サーバーを定義する際に、構成ストリングを GENFEDTEXTQRY=YES に設定する必要があります。ASSOCIATED_ENTITY キーワードは、統合エンティティがテキスト検索可能である場合に限り適用できます。これが有効になるのは、テキスト検索可能な統合エンティティ (FedTextResource) が、Content Manager バージョン 8 のテキスト検索可能な項目タイプ、または Extended Search のテキスト検索可能なエンティティにマップされている場合のみです。

```
"TEXT_SEARCH = ( COND = ( 'operating system' ) ); OPTION =  
( SEARCH_INDEX = FedTMINDEX; ASSOCIATED_ENTITY = FedTextResource;  
MAX_RESULTS = 5 )"
```

OR 演算子を使用した統合パラメトリックおよびテキスト照会

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource,  
AX_RESULTS = 0,COND = (FedTextResourceJTitle LIKE '%test%'  
) ) OR TEXT_SEARCH = ( COND = ('UNIX') );  
OPTION = ( ASSOCIATED_ENTITY =  
FedTextResource; MAX_RESULTS = 4 )"
```

AND 演算子を使用した統合パラメトリックおよびテキスト照会

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource, COND =  
(FedTextResourceJTitle LIKE '%test%') ) AND TEXT_SEARCH =  
( COND = ('UNIX') ); OPTION = ( ASSOCIATED_ENTITY =  
FedTextResource)"
```

OR 演算子を使用した、属性に対する統合パラメトリックおよびテキスト照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal,  
COND = (fJTitle LIKE 'Java%' OR fJArticleTitle  
CONTAINS_TEXT 'Database') );OPTION = ( CONTENT = ATTRONLY )"
```

OR 演算子を使用した、属性に対する統合パラメトリックおよびテキスト照会

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal,  
COND = (fJTitle LIKE 'Java%' OR fJArticleTitle  
CONTAINS_TEXT 'Database') );OPTION = ( CONTENT = YES )"
```

統合フォルダーへの照会結果の保管 (Java のみ)

Enterprise Information Portal バージョン 8 リリース 2 は、統合フォルダーを保管できる特別な統合エンティティを提供しています。この統合フォルダーには、統合照会の結合された結果を保管できます (例えば、Content Manager からの文書と OnDemand からの関連文書)。この結果は、ワークフローに直接送ることができます。

EIP はフォルダーを DDO として保管し、この DDO は DKFolder コレクションに追加できます。また、フォルダーを XDO として DKParts コレクションに保管することもできます。

特別な統合エンティティは、フォルダーを保管するために他のテーブルを使用します。その他の機能 (照会、API、属性など) は、通常の統合エンティティと同じように動作します。特別なエンティティは、DDO PID のみをフォルダーに保管します。

特別な統合エンティティをマップしない場合、統合照会は特別な統合エンティティのみを検索します。特別な統合エンティティを他のネイティブ・エンティティにマップすると、統合照会はこれらのエンティティも加えて検索します。

コード・サンプルについては、CMBROOT¥dk¥samples ディレクトリーを参照してください。

システム管理の処理

Enterprise Information Portal には、システム管理機能にアクセスするためのクラスおよび API が用意されています。特定のクラスについての詳細は、「オンライン API 解説書」を参照してください。

EIP システム管理クライアントのカスタマイズ

EIP システム管理クライアントはシステム管理アプリケーションの拡張をサポートしており、次のカスタム機能を組み込むことができます。

- EIP システム管理クライアントのユーザー・ダイアログとユーザー・グループ・ダイアログを、ユーザー独自のダイアログに置き換えることができます。
- EIP システム管理クライアントの階層に新規ノードを追加することができます。
- システム管理クライアントの「ツール (Tools)」メニューに新規のメニュー項目を追加することができます。

EIP システム管理にログオンする前、およびログオンした後にユーザー出口を呼び出すことができます。

アプリケーション・プログラミング・インターフェース (API) を使用したプログラミング

アプリケーション・プログラミング・インターフェース (API) は、ローカルまたはリモート・データにアクセスして操作するクラスのセットです。このセクションでは、API、複数検索機能のインプリメンテーション、およびインターネット接続について説明します。

API は以下のものをサポートします。

- データ・アクセス用の共通オブジェクト・モデル。
- 異種組み合わせのコンテンツ・サーバー間での複数検索および更新。
- 検索エンジンを組み合わせて使用するための柔軟なメカニズム。例えば、Content Manager のテキスト検索フィーチャー。
- ワークフロー機能。
- 管理機能。
- **Java のみ:** Java アプリケーション用のクライアント/サーバーのインプリメンテーション

Java API のマルチストリーム・サポートは、Windows[®] サーバーのみが完全に対応しています。AIX サーバー、クライアント、および Windows クライアントは、マルチストリームをサポートできません。

Java API と C++ API の違い

IBM Enterprise Information Portal for Multiplatforms Java と C++ API セットとの間には、以下のような違いがあります。

- C++ API で定義されている演算子は、Java API では定義されていません。それらの演算子は、Java 関数としてサポートされています。
- 汎用オブジェクトを表すためには、C++ クラス DKAny の代わりに Java のクラス・オブジェクト (java.lang.Object) が使用されます。
- 共通定数およびグローバル定数は、Java API 内のインターフェース DKConstant に定義されています。C++ では、DKConstant.h に定義されています。
- Java API は、Java のガーベッジ・コレクターを使用します。
- Java 関数 DKDDO.toXML() と DKDDO.fromXML() は、C++ では使用できません。

クライアント/サーバー・アーキテクチャーの概要 (Java のみ)

API は、アプリケーション開発者に便利なプログラミング・インターフェースを提供します。API は、EIP サーバーとクライアントの両方に置くことができます (両方が同じインターフェースを備える)。クライアント API は、Java RMI (リモート・メソッド呼び出し) によってネットワーク経由でサーバーと交信して、データ

にアクセスします。クライアントとサーバーとの間の通信はクラスによって実行されるので、付加的なプログラムを追加する必要はありません。

API クラスは、`server`、`client`、`cs`、および `common` の各パッケージで構成されています。クライアントおよびサーバー・クラスは、同じ API を備えていますが、インプリメンテーションは異なります。

- サーバー・パッケージは、`com.ibm.mm.sdk.server` です。サーバー・パッケージ内のクラスは、統合コンテンツ・サーバーまたはバックエンド・コンテンツ・サーバーと直接通信します。
- クライアント・パッケージは、`com.ibm.mm.sdk.client` です。クライアント・パッケージ内のクラスは、RMI によってサーバー・パッケージ内のクラスと通信します。
- 共通クラスは、クライアントとサーバーの両方によって共有されます。コンテンツがどこにあるのかがアプリケーションには分からない場合があります。例えばアプリケーションは、ある場合にはクライアント上に、別の場合にはサーバー上にあるコンテンツを持つ場合があります。`cs` パッケージは、クライアントとサーバーを動的に接続します。

クライアント・アプリケーションはクライアント・パッケージをインポートし、動的アプリケーションは `cs` をインポートし、サーバー・アプリケーションはサーバー・パッケージをインポートする必要があります。

クライアントとサーバーには同じ API が備えられていますが、クライアント・パッケージには、サーバー・パッケージと通信するために追加の例外項目があります。ただし、一部のコネクタに対してはクライアント/サーバー・インターフェースはサポートされません。例えば、CM V8 ではサポートされていません。

Java 環境用のパッケージ

Enterprise Information Portal API は、`com.ibm.mm.sdk` の一部として、`common`、`server`、`client`、および `cs` の 4 つのパッケージ内に含まれています。

server (`com.ibm.mm.sdk.server`)

コンテンツ・サーバー情報のアクセスと操作

client (`com.ibm.mm.sdk.client`)

リモート・メソッド呼び出し (RMI) による `server` パッケージとの通信

common (`com.ibm.mm.sdk.common`)

サーバー・パッケージ、クライアント・パッケージ、および `cs` パッケージで共通に使用されるクラス

cs (`com.ibm.mm.sdk.cs`)

クライアントまたはサーバーとの動的接続

ご使用のアプリケーションでは、`common` パッケージと共に、ローカル・アプリケーション用の `server` パッケージ、リモート・サーバーにアクセスするアプリケーション用の `client` パッケージ、または `cs` パッケージのいずれかを使用しなければなりません。

プログラミング上のヒント

同じプログラムにクライアント・パッケージとサーバー・パッケージをインポートしないでください。クライアント・アプリケーションを開発する場合には、クライアント・パッケージをインポートしてください。それ以外の場合は、サーバー・パッケージをインポートしてください。コンテンツがある場所が分からない場合には、`cs` パッケージを (サーバーまたはクライアント・パッケージとともに) 使用してください。複数のパッケージをインポートすると、コンパイル・エラーが起きる場合があります。

一部のコネクタのインプリメンテーションには、C コードの呼び出しが含まれています。このようなコネクタの場合は、ピュア Java インターフェースを必要とする Web アプリケーション用のクライアント・パッケージを使用してください。クライアント・パッケージはピュア Java プログラムによって作成されています (サーバー・パッケージには JNI 呼び出しが含まれていることがあります)。

クライアントには例外も必要であるため、`java.rmi.RemoteException` は、アプリケーションがサーバー上で実行するとしてもクライアント上で実行するとしても、アプリケーションに必ずこの例外を付加します。

Java 環境のセットアップ (Java のみ)

Windows、AIX、または Solaris の環境をセットアップする際には、次のパッケージをインポートする必要があります。

サーバー・パッケージ

コンテンツ・サーバーおよびアプリケーションがサーバー側にある場合にインポートします。

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.server`

クライアント・パッケージ

コンテンツ・サーバーおよびアプリケーションがクライアント側にあるときにインポートします。

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.client`

cs パッケージ

コンテンツ・サーバーのロケーションがアプリケーションのロケーションとは異なるときにインポートします。

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.cs`

Windows 用の Java 環境変数の設定

Enterprise Information Portal アプリケーションの開発用にセットアップした環境で、Windows コマンド・プロンプトを開くには、「スタート」→「プログラム」→「IBM Enterprise Information Portal for Multiplatforms 8.2」→「開発ウィンドウ (Development Window)」の順に選択します。環境をセットアップするための別の方法として、Windows コマンド・プロンプトで `cmbenv81.bat` を実行することもできます。

環境変数を変更したい場合は、以下を変更します。

PATH PATH には、必ず `X:%CMBROOT%\DLL` を入れてください (X は Enterprise Information Portal のインストール先ドライブ)。

CLASSPATH

CLASSPATH には、必ず `X:%CMBROOT%\LIB\%xxx` を入れてください (X は Enterprise Information Portal のインストール先ドライブ、xxx は .jar ファイル (例: cmbfed81.jar))。

AIX 用の Java 環境変数の設定

AIX 環境では、シェル・スクリプト `cmbenv81.sh` を使用して、EIP アプリケーション開発のための開発環境をセットアップできます。

スクリプトを使用しない場合は、以下の環境変数を設定する必要があります。

PATH PATH には、必ず `/usr/lpp/cmb/lib` を入れてください。

LIBPATH

LIBPATH には、必ず `/usr/lpp/cmb/lib` を入れてください。

LD_LIBRARY_PATH

LD_LIBRARY_PATH には、必ず `/usr/lpp/cmb/lib` を入れてください。

CLASSPATH

CLASSPATH には、必ず `/usr/lpp/cmb/lib/xxx` を入れてください (xxx は .jar ファイル (例えば cmbfed81.jar))。

オブジェクトが正しく位置合わせされるように、`-qalign=packed` コンパイラー・オプションを使用してください。

Solaris 用の Java 環境変数の設定

Solaris 環境では、シェル・スクリプト `cmbenv81.sh` を使用して、EIP アプリケーションの開発のための開発環境をセットアップできます。

スクリプトを使用しない場合は、以下の環境変数を設定する必要があります。

PATH PATH には必ず `/opt/cmb/lib` を含めてください。

LIBPATH

LIBPATH には必ず `/opt/cmb/lib` を含めてください。

LD_LIBRARY_PATH

LD_LIBRARY_PATH には、必ず `/opt/cmb/lib` を含めてください。

CLASSPATH

CLASSPATH には、必ず `/opt/cmb/lib/xxx` を含めてください。xxx は .jar ファイルです (例えば cmbfed81.jar)。

オブジェクトが正しく位置合わせされるように、`-qalign=packed` コンパイラー・オプションを使用してください。

コンテンツ・サーバーでのリモート・メソッド呼び出し (RMI) の使用

Java API 内のクライアント・クラスは、サーバー・クラスと通信してネットワークを介してデータにアクセスする必要があるため、サーバーとクライアントの両方とも、クライアント / サーバーが実行できる状態でなければなりません。サーバー・マシン上では、RMI サーバーは、指定のポート番号を使用したクライアントからの要求を受信するために、実行状態でなければなりません。クライアント・プログラムには、サーバー名とポート番号が必要です。クライアントとサーバーの間で通信を行うには、クライアント側で接続先のサーバーのポート番号が分かっている必要があります。

RMI サーバーが接続できるコンテンツ・サーバーの数は無制限ですが (システム・リソースによる制限のみ)、それぞれのサーバーは、少なくとも 1 つのコンテンツ・サーバーに接続されなければなりません。マスター RMI サーバーは、サーバー・プール内の他の RMI サーバーを参照することができます。RMI クライアントは、初めてコンテンツ・サーバーを検索する際に、RMI サーバーを始動させます。コンテンツ・サーバーがそこに見つからない場合には、次に RMI プール・サーバーが検索されます。

同じ RMI クライアントが再びコンテンツ・サーバーを検索する場合、そのクライアントはコンテンツ・サーバーが最初に見つかった RMI サーバーを検索します。

RMI サーバーを開始するには、Windows では `cmbregist81.bat` を、AIX または Solaris では `cmbregist81.sh` を使用してください。RMI サーバーを開始する前に、正しいポート番号およびサーバー・タイプを定義してください。RMI サーバーの構成および管理については、「Enterprise Information Portal の計画とインストール」および「Enterprise Information Portal の管理」を参照してください。

C++ 環境のセットアップ (C++ のみ)

Windows または AIX 環境をセットアップするには、ここで説明する設定を確立しなければなりません。34 ページの表 4 に、ライブラリー、AIX 共用、および DLL の要件を示します。

要件: C++ を使用するには、DB2 クライアント・サポートおよび Client Configuration Assistant を、Enterprise Information Portal データベースにアクセスするすべてのリモート・サーバーにインストールする必要があります。データベースへの接続に使用するユーザー ID およびパスワードは、Enterprise Information Portal データベースで使用するユーザー ID およびパスワードと同じものでなければなりません。詳細については、「Enterprise Information Portal の管理」を参照してください。

重要: `cmbcm81x.lib` はリリース・ビルド用、`cmbcm81xd.lib` はデバッグ・ビルド用です。ここで、*x* は Microsoft Visual C++ コンパイラーのバージョン 6 または 7 (.Net) を表します。

表4. 共用オブジェクト、および DLL 環境の情報

コネクター	ライブラリー	Windows DLL	共用オブジェクト (AIX 用)
共通の注: これはコネクターではなく、すべてのコネクターによって使用される API の共通セットです。	cmbcm81x.lib cmbcm81xd.lib	cmbcm81x.dll cmbcm81xd.dll	libcmbcm815.a
Content Manager バージョン 8	cmbicm81x.lib cmbicm81xd.lib	cmbicm81x.dll cmbicm81xd.dll cmbicmfac81x.dll cmbicmfac81xd.dll	libcmbicm815.a libcmbicmfac815.so
旧バージョンの Content Manager	cmbdl81x.lib cmbdl81xd.lib	cmbdl81x.dll cmbdl81xd.dll cmbdlfac81x.dll cmbdlfac81xd.dll de_db2.dll de_db2_d.dll de_ora.dll de_ora_d.dll	libcmbdl815.a libcmbdlfac815.so
統合	cmbfed81x.lib cmbfed81xd.lib	cmbfed81x.dll cmbfed81xd.dll cmbfedfac81x.dll cmbfedfac81xd.dll	libcmbfed815.a libcmbfedfac815.so
DB2 Universal Database バージョン 8.1	cmbdb281x.lib cmbdb281xd.lib	cmbdb281x.dll cmbdb281xd.dll cmbdb2fac81x.dll cmbdb2fac81xd.dll	libcmbdb2815.a libcmbdb2fac815.so
ODBC	cmbodbc81x.lib cmbodbc81xd.lib	cmbodbc81x.dll cmbodbc81xd.dll cmbodbcfac81x.dll cmbodbcfac81xd.dll	サポートされていません。
OnDemand	cmbod81x.lib cmbod81xd.lib	cmbod81x.dll cmbod816xd.dll cmbodfac81x.dll cmbodfac81xd.dll	libcmbod815.a libcmbodfac815.so
ImagePlus for OS/390	cmbip81x.lib cmbip81xd.lib	cmbip81x.dll cmbip81xd.dll cmbipfac81x.dll cmbipfac81xd.dll	サポートされていません。
VisualInfo	cmbv481x.lib cmbv481xd.lib	cmbv481x.dll cmbv481xd.dll cmbv4fac81x.dll cmbv4fac81xd.dll	サポートされていません。
Domino.Doc	cmbdd81x.lib cmbdd81xd.lib	cmbdd81x.dll cmbdd81xd.dll cmbddfac81x.dll cmbddfac81xd.dll	サポートされていません。

表 4. 共用オブジェクト、および DLL 環境の情報 (続き)

コネクター	ライブラリー	Windows DLL	共用オブジェクト (AIX 用)
Domino Extended	cmbdes81x.lib	cmbdes81x.dll	libcmbdes815.a
Search	cmbdes81xd.lib	cmbdes81xd.dll	libcmbodfac815.so
		cmbdesfac81x.dll	
		cmbdesfac81xd.dll	

Windows 用の C++ 環境変数の設定

Enterprise Information Portal アプリケーションの開発用に構成した環境で DOS コマンド・プロンプトを開くには、「スタート」→「プログラム」→「IBM Enterprise Information Portal for Multiplatforms」→「開発ウィンドウ (Development Window)」の順に選択します。別の方法として、DOS コマンド・プロンプトで cmbenv81.bat を実行して、環境をセットアップすることもできます。

環境変数を変更したい場合は、以下を変更します。

PATH

```
set PATH=x:¥CMBROOT¥DLL
```

ここで、*x* は EIP がインストールされているドライブです。

INCLUDE

```
set INCLUDE=x:¥CMBROOT¥INCLUDE
```

ここで、*x* は EIP がインストールされているドライブです。

AIX 用の C++ 環境変数の設定

詳細については、samples ディレクトリーにあるサンプルの MAK ファイルを参照してください。

以下の環境変数を設定してください。

AIX 環境では、3 つのバッチ・ファイルのいずれか 1 つを使用して、ご使用の開発環境をセットアップすることができます。

1. Bourne シェルの場合、cmbenv81.sh を使用します。
2. C シェルの場合、cmbenv81.csh を使用します。
3. Korn シェルの場合、cmbenv81.ksh を使用します。

以下の環境変数を設定してください。

NLSPATH

```
export NLSPATH=${NLSPATH}:/usr/lpp/cmb/msg/En_US/%N
```

PATH

```
export PATH=/usr/lpp/cmb/lib
```

LIBPATH

```
export LIBPATH=/usr/lpp/cmb/lib
```

INCLUDE

```
export INCLUDE=/usr/lpp/cmb/INCLUDE
```

C++ プログラムのビルド

MAK ファイルを作成し、アプリケーションをビルドするには、コンパイラーおよび開発環境の手順に従います。

デバッグ用に使用する C++ API を使ってアプリケーションを構築する場合は、ご使用のアプリケーションを、デバッグ・バージョンの API ライブラリー (つまり、***d.lib** ライブラリー) とリンクさせます。最終実動アプリケーションを構築する場合は、非デバッグ・ライブラリー (***.lib**) とリンクさせます。

Microsoft Visual Studio .NET の使用

Enterprise Information Portal と Content Manager のバージョン 8.2 以降の API は、Microsoft Visual Studio .NET をサポートしています。ただし、Microsoft Visual Studio .NET を使用してアプリケーションを作成する場合は、コンパイル時に適切なライブラリーにリンクする必要があります。このライブラリーは、お使いのコンネクターと Visual Studio C++ のバージョンによって決まります。

MAK ファイルの例 (Visual Studio バージョン 6 と Visual Studio .NET の両方をサポート) が、ICM API サンプルとともに提供されています。

コンパイル時に接続する必要があるライブラリーを判別するには、次の表に示すように、Microsoft Visual Studio C++ 6 をお使いの場合はフォーマット *connector name816.lib* を使用し、Microsoft Visual Studio .NET をお使いの場合は *connector name817.lib* を使用します。

表 5. Microsoft Visual Studio C++ 6 のライブラリー名の例

コネクター	ライブラリー
共通	cmbcm816.lib
Content Manager 8.1 以降	cmbicm816.lib
統合	cmbfed816.lib

表 6. Microsoft Visual Studio .NET のライブラリー名

コネクター	ライブラリー
共通	cmbcm817.lib
Content Manager 8.1 以降	cmbicm817.lib
統合	cmbfed817.lib

Windows でのコード・ページ変換用コンソール・サブシステムの設定

C++

```
#include <DKConstant.h>
#include <DKEnvironment.hpp>

void main(int argc, char *argv[])
{
    // set sub system to console at the beginning of program this
    // will cause the code page that the error messages are returned
    // in by DKExceptions to be converted from the Windows Graphical
    // User Interface (ANSI format) to the Console (OEM format)
    // If this is not specified the default is DK_SS_WINDOWS
    DKEnvironment::setSubSystem(DK_SS_CONSOLE);
    ...
}
```

複数検索オプションの概要

検索は実際、項目またはコンポーネントの任意の部分、項目またはコンポーネント内のテキスト、またはリソース・コンテンツ内のテキストに基づいて実行できます。

Content Manager バージョン 8 にはテキスト・フィーチャーが統合されています。このフィーチャー用に、テキスト検索機能を別途用意する必要はなくなりました (旧バージョンの Content Manager の場合は必要)。205 ページの『テキスト検索について』を参照してください。

複数検索オプションは、以下に示すサポートされている照会タイプのいずれか、またはそれらの組み合わせを使用して特定のコンテンツ・サーバー内を検索したり、直前の検索の結果に対して検索を実行したりするのに使用します。それぞれの検索タイプは、1 つまたは複数の検索エンジンによってサポートされています。複数検索オプションをサポートしないコンテンツ・サーバーもあります。

パラメトリック・サーチ

項目とコンテンツのプロパティ、属性、参照、リンク、フォルダー・コンテンツなどのテキストを検索します。例えば、顧客名を使用して文書を検索する場合にパラメトリック照会を使用します。この照会を行う場合は、照会述部に指定した条件と、コンテンツ・サーバーに保管されているデータ値とが厳密に一致する必要があります。

テキスト検索

DB2 Net Search Engine (NSE) などのテキスト検索エンジンを使用して、`textSearchable(true)` とマークされた任意のテキストを検索します。この照会は、指定したテキスト検索式とおおむね一致するテキスト・フィールドのコンテンツに基づいて行われます。例えば、ある特定の句または語幹が存在する（または存在しない）などです。

注: DB2 Net Search Engine (NSE) は、DB2 の旧バージョンでは DB2 Text Information Extender (TIE) と呼ばれていました。

イメージ検索

イメージ内の特性を検索します。この照会は、指定したイメージ検索式とほぼ一致するイメージのコンテンツに基づいて行われます。例えば、イメージ内に特定の色があるかどうかなどです。

結合検索

パラメトリック検索およびテキスト検索の両方を使用する検索。

Content Manager のみ: CM には検索エンジンが 1 つ備わっており、3 つの選択項目（実行、評価、およびコールバックによる実行）から、検索の実行方法と実行時期を選択できます。表と説明については、SSearchICM のサンプルを参照してください。

トレース

API アプリケーション内で生じる問題を処理するために、トレースおよび例外処理を使用することができます。

テキスト検索エンジンを使用したテキスト照会のトレース

テキスト検索エンジン (TSE) およびそのすべての機能は、旧バージョンの Content Manager でのみ使用できます。Content Manager バージョン 8 にはテキスト・フィーチャーが統合されています。このフィーチャー用に、テキスト検索機能を別途用意する必要はありません。205 ページの『テキスト検索について』を参照してください。

以下の環境変数の設定では、テキスト検索エンジン照会のトレースを、指定したファイルにバイナリ形式で書き込みます。

`CMBTMDSTREAMTRACE=fileName`

(例: Windows の場合は `.\%tm.out`、AIX の場合は `./tm.out`)

以下の環境変数の設定では、テキスト照会の際に使用されるテキスト検索エンジン API 呼び出しのトレースを、指定ファイルに書き込みます。

`CMBTMTRACE=fileName`

以下の環境設定は、指定ファイルにテキスト検索の用語を書き込みます。
CMBTMTerm=*fileName* (例: .%tmterm.out)

注: Content Manager バージョン 8 では、統合されたテキスト検索が使用されます。旧バージョンの Content Manager を使用している場合は、引き続きテキスト検索エンジン (TSE) を使用できます。

パラメトリック照会のトレース

旧バージョンの Content Manager でテキスト検索エンジンを使用する場合は、次の環境変数の設定を使用してパラメトリック照会を作成し、フォルダー・マネージャーに渡します。

CMBDLQRYTRACE=*fileName*

(例: Windows の場合は <.%dlqry.out>、AIX の場合は <./dlqry.out>)

例外の処理

API は問題を検出すると、例外をスロー (throw) します。例外がスローされると、DKException クラスまたはそのサブクラスの例外オブジェクトが 1 つ作成されます。

DKException が作成されると、コネクタ層は、デフォルトのログ作成の構成が使用されているものとして、診断情報をログ・ファイルに書き込みます。EIP API によって使用されるログ・ファイルおよび構成ファイルについての詳細は、「メッセージとコード」を参照してください。

DKException をキャッチ (catch) した場合、エラー・メッセージ、エラー・コード、および実行中に発生したエラー状態が表示されます。エラーが見つかったと、エラー・メッセージが例外がスローされたロケーションについての情報と共に発行されます。エラー ID などの追加情報も示されます。以下のコードは、EIP と CM の場合に行われる、例外のスロー/キャッチ・プロセスの例を示しています。

Java

```
try{
    ... EIP API Operations ...
}
catch (DKException exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("        Name: " + exc.name());
    System.out.println(" Message: " + exc.getMessage());
    System.out.println(" Message ID: " + exc.getErrorId());
    System.out.println("Error State: " + exc.errorState());
    System.out.println(" Error Code: " + exc.errorCode());
    exc.printStackTrace();
    System.out.println("-----");
} catch (Exception exc) {
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("        Name: " + exc.getClass().getName());
    System.out.println(" Message: " + exc.getMessage());
    exc.printStackTrace();
    System.out.println("-----");
}
```

C++

```
try{
    ... EIP API Operations ...
}
catch (DKException &exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    cout << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    cout << "X    !!! Exception !!!    X" << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    cout << "        Name: " << exc.name() << endl;
    cout << " Message ID: " << exc.errorId() << endl;
    cout << "Error State: " << exc.errorState() << endl;
    cout << " Error Code: " << exc.errorCode() << endl;
    // Print API Location(s) Detecting Error
    for(unsigned int j=0; j < exc.locationCount(); j++){ //Print all locations
        const DKExceptionLocation* p = exc.locationAtIndex(j);
        cout << " Location " << j << ": " << p->fileName() << " :: "
            << p->functionName() << '[' << p->lineNumber() << ']' << endl;
    }
    if(exc.textCount()<=0) // Write statement if no locations.
        cout << " Locations: <none> " << endl;
    // Error Message(s)
    for(unsigned int i=0; i < exc.textCount(); i++) // Print All Messages
        cout << " Message " << i << ": " << exc.text(i) << endl;
    if(exc.textCount()<=0) // Notify user if no messages.
        cout << " Messages: <none> " << endl;
    cout << "-----" << endl;
}
```

エラーの検出と処理について詳しくは、CMBROOT¥Samples¥java¥icm または CMBROOT¥Samples¥cpp¥icm にある SConnectDisconnectICM API 実習サンプルを参照してください。

定数

指定される定数は、DK_CM_ (共通定数) または DK_XX_ の形式です。ただし、XX はさまざまなコンテンツ・サーバーを示しています。拡張子 (それぞれの DKDatastore に付加される拡張子) のリストについては、表 7 を参照してください。

DDO 定数を指定する場合、プロパティ・タイプには DK_CM_DATAITEM_TYPE_ ... を使用します (例: DK_CM_DATAITEM_TYPE_STRING)。属性タイプの場合は、DK_CM_...type 定数を使用します (例: DK_CM_INTEGER)。

Java

共通の定数は、DKConstant.java に定義されています。DKConstant.txt では、これらの定数のテキスト版を参照することができます。特定のコンテンツ・サーバー用の定数は、DKConstantXX.java に定義されています。例えば、Content Manager に固有の定数は DKConstantICM.java にあります。

C++

共通の定数は、DKConstant.h に定義されています。定数と対応する値のリストについては、DKConstant2.h を参照してください (ただし、プログラムにこれを組み込まないでください)。特定のコンテンツ・サーバー用の定数は、形式 DKConstantXX.h のヘッダー・ファイルに定義されています。例えば、Content Manager に固有の定数は DKConstantICM.h にあります。

コンテンツ・サーバーへの接続

クラス DKDatastorexx (xx は特定のコンテンツ・サーバー) のオブジェクトはコンテンツ・サーバーとの接続を表し、接続の管理、トランザクション・サポートの提供、およびサーバー・コマンドの実行を行います。具体的な接尾部については、表 7 を参照してください。

表 7. サーバー・タイプおよびクラス名用語

コンテンツ・サーバー	クラス名
Content Manager バージョン 8.2	DKDatastoreICM
旧バージョンの Content Manager	DKDatastoreDL
Content Manager OnDemand	DKDatastoreOD
Content Manager for AS/400 (VisualInfo for AS/400)	DKDatastoreV4
Content Manager ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD

表7. サーバー・タイプおよびクラス名用語 (続き)

コンテンツ・サーバー	クラス名
Extended Search	DKDatastoreDES
Panagon Image Services (FileNET)	DKDatastoreFN
リレーショナル・データベース	DKDatastoreDB2、DKDatastoreJDBC (Java の場合)、DKDatastoreODBC (C++ の場合)

接続の確立

それぞれの DKDatastore.xx クラスは、それに接続したり切断したりするためのメソッドを提供しています。次の例では、ICMNLSDb という名前の Content Manager ライブラリー・サーバー、ユーザー ID ICMADMIN、およびパスワード PASSWORD を使用します。Content Manager については、Content Manager システムへの接続を参照してください。それ以外のコンテンツ・サーバーについては、その他のコンテンツ・サーバーの使用を参照してください。この例では、CM コンテンツ・サーバー用の DKDatastoreICM オブジェクトを作成し、接続し、それを処理し、切断します。

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //Create datastore object
dsICM.connect("ICMNLSDb","ICMADMIN","PASSWORD",""); //Connect to datastore

System.out.println("Connected to datastore dbase: '"+dsICM.datastoreName()+
    "', UserName '"+dsICM.userName()+"'.");

dsICM.disconnect(); // Disconnect from datastore
dsICM.destroy(); // Destroy reference
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SConnectDisconnectICM.java を参照してください。

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM(); //Create datastore object
dsICM->connect("ICMNLSDb","ICMADMIN","PASSWORD",""); //Connect to datastore

cout << "Connected to datastore dbase: '" << dsICM->datastoreName() <<
    "', UserName '" << dsICM->userName() << "'." << endl;

dsICM->disconnect(); //Disconnect from datastore
delete(dsICM); //Destroy reference
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SConnectDisconnectICM.cpp を参照してください。

コンテンツ・サーバーに接続するときには、それぞれのコンテンツ・サーバーの要件を知っている必要があります。例えば、ImagePlus for OS/390 のパスワードの長さは 8 文字以内でなければなりません。

クライアント内のコンテンツ・サーバーからの接続および切断

クライアント・アプリケーションからコンテンツ・サーバーにアクセスするには、42 ページの『接続の確立』にあるものと同じコードを使用し、単に `import com.ibm.mm.sdk.server.*;` を `import com.ibm.mm.sdk.client.*;` に置き換えます。クライアント・アプリケーションは、発生した通信エラーを処理しなければなりません。

コンテンツ・サーバー・オプションの設定および取得

DKDatastorexx でメソッドを使用して、コンテンツ・サーバー上の処理オプションにアクセスするかまたはそれを設定することができます。以下の例は、Content Manager のライブラリー・サーバー上に管理セッションを確立するためのオプションを設定および取得する方法を示しています。オプションのリストとその説明については、「オンライン API 解説書」を参照してください。

ここに示す、Content Manager 内のコンテンツ・サーバー・オプションの設定と取得を行う例では、キャッシングはオフになっています。コンテンツ・サーバーがこのオプションをサポートしている場合は、デフォルト (ON) の使用をお勧めします。
要件: dsICM という名前の変数内に、有効な DKDatastoreICM オブジェクトがすでに作成済みであること。

Java

```
dsICM.setOption(DKConstant.DK_CM_OPT_CACHE,
    new Integer(DKConstant.DK_CM_FALSE));
Object val = dsICM.getOption(DKConstant.DK_CM_OPT_CACHE);
```

C++

```
DKAny inVal = DK_CM_FALSE
DKAny outVal;
dsICM->setOption(DK_CM_OPT_CACHE,inVal);
dsICM->getOption(DK_CM_OPT_CACHE,outVal);
```

コンテンツ・サーバー・オプションを取得する場合、`output_option` は通常は整数ですが、それをオブジェクトになるようにキャストすることができます。

コンテンツ・サーバーのリスト作成

DKDatastorexx は、接続可能なサーバーをリストするためのメソッドを提供します。サーバーのリストは、DKServerDefxx オブジェクトの DKSequentialCollection に戻されます (xx は特定のコンテンツ・サーバー)。

制限事項: Domino.Doc コンテンツ・サーバーは、サーバーをリストするメソッドを提供していません。

DKServerDefxx オブジェクトを取得したら、サーバー名およびサーバー・タイプを取得して、そのサーバー名を使用して接続を確立することができます。

次の例では、接続先として構成されているサーバーのリストを示します。

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); // Create a datastore object
dkCollection coll = dsICM.listDataSources(); // Obtain data source list
dkIterator iter = coll.createIterator(); // Create an iterator
while(iter.more()){ // While there are more
    DKServerDefICM srvrDef = (DKServerDefICM) iter.next();
    System.out.println("Found server '"+srvrDef.getName()+"");
}
```

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM(); // Create a datastore object
dkCollection* coll = (dkCollection*)dsICM->listDataSources(); // Obtain list
dkIterator* iter = coll->createIterator(); // Create an iterator
while(iter->more()){ // While there are more
    DKServerDefICM* srvrDef = (DKServerDefICM*) iter->next()->value();
    cout << "Found server '" << srvrDef->getName() << "' " << endl;
    delete(srvrDef); // Free memory
}
delete(iter); // Free memory
delete(coll);
delete(dsICM);
```

コンテンツ・サーバー用のエンティティおよび属性のリスト

DKDatastore.xx は、コンテンツ・サーバー用にエンティティおよびその属性をリストするためのメソッドを備えています。各属性名は、ネーム・スペースの一部です。デフォルトのネーム・スペースは、ネーム・スペースが指定されていない属性すべてに使用されます。

エンティティのリストは、dkEntityDef オブジェクトの DKSequentialCollection オブジェクトに戻されます。エンティティの属性は、dkAttrDef オブジェクトの DKSequentialCollection オブジェクトに戻されます。dkAttrDef オブジェクトを取得したら、その名前やタイプなどの、属性に関する情報を検索して、その情報を使用して照会を作成することができます。

これら 2 つのメソッドの詳細については、「[オンライン API 解説書](#)」を参照してください。

下記の例は、Content Manager サーバーから項目タイプのリストと属性のリストを取得する方法を示しています。

Java

```
. . .
try {
    DKSequentialCollection pCol = null;
    dkIterator pIter = null;
    DKSequentialCollection pCol2 = null;
    dkIterator pIter2 = null;
    DKServerDefICM pSV = null;
    String strServerName = null;
    String strItemType = null;
    DKComponentTypeDefICM itemTypeDef = null;
    DKAttrDefICM attrDef = null;
    DKDatastoreDefICM dsDefICM = null;
    int i = 0;
    int j = 0;
    // ----- Create the datastore and connect (assumes the
    //      parameters for the connection are previously set)
    DKDatastoreICM dsICM = new DKDatastoreICM();
    dsICM.connect(db,userid,pw,"");
    // ----- List the item types
    pCol = (DKSequentialCollection) dsICM.listEntities();
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        itemTypeDef = (DKComponentTypeDefICM)pIter.next();
        strItemType = itemTypeDef.getName();
        System.out.println("item type name [" + i + "] - " + strItemType);
        System.out.println("    type " + itemTypeDef.getType());
        System.out.println("    itemTypeId " + itemTypeDef.getId());
        System.out.println("    compID " + itemTypeDef.getComponentTypeId());
        //continued . . .
    }
}
```


Java (続き)

```
// ----- List the attributes
pCol2 = (DKSequentialCollection) dsICM.listEntityAttrs(strItemType);
pIter2 = pCol2.createIterator();
j = 0;
while (pIter2.more() == true)
{
    j++;
    attrDef = (DKAttrDefICM)pIter2.next();
    System.out.println("Attr name [" + j + "] - " + attrDef.getName());
    System.out.println("    datastoreType " + attrDef.datastoreType());
    System.out.println("    attributeOf " + attrDef.getEntityName());
    System.out.println("    type " + attrDef.getType());
    System.out.println("    size " + attrDef.getSize());
    System.out.println("    id " + attrDef.getId());
    System.out.println("    nullable " + attrDef.isNullable());
    System.out.println("    precision " + attrDef.getPrecision());
    System.out.println("    scale " + attrDef.getScale());
    System.out.println("    stringType " + attrDef.getStringType());
    System.out.println("    sequenceNo " + attrDef.getSequenceNo());
    System.out.println("    userFlag " + attrDef.getUserFlag());
}
dsICM.disconnect();
}
catch(DKException exc)
{
// ----- Handle the exceptions
```

完全なサンプル SItemTypeRetrievalICM に、項目タイプの定義をリストする方法が示してあります。もう 1 つの完全なサンプル SAttributeDefinitionRetrievalICM には、属性の定義をリストする方法が示してあります。両サンプルは、CMBROOT¥Samples¥java¥icm ディレクトリーにあります。

次の C++ の例は、Content Manager サーバーから索引クラスと属性のリストを取り出す方法を示しています。

C++

```
// Get a collection containing all Item Type Definitions.
DKSequentialCollection* itemTypeColl = (DKSequentialCollection*)
    dsICM->listEntities();
// Accessing each and printing the name & description.
cout << "¥nItem Type Names in System:
    (" << itemTypeColl->cardinality() << ')' << endl;
// Create an iterator to iterate through the collection
dkIterator* iter = itemTypeColl->createIterator();
// while there are still items in the list, continue
while(iter->more()){
    DKItemTypeDefICM* itemType = (DKItemTypeDefICM*) iter->next()->value();
    cout << " - " << itemType->getName() << ": " <<
        itemType->getDescription() << endl;
    delete(itemType); // Free Memory

    cout << endl;
    delete(iter);
    delete(itemTypeColl);
```

詳しくは、SItemTypeRetrievalICM サンプルを参照してください。このサンプルには、項目タイプの定義をリストする方法が示してあります。

SAttributeDefinitionRetrievalICM には、属性の定義をリストする方法が示してあります。これらのサンプルは、CMBROOT¥Samples¥cpp¥icm ディレクトリーにあります。

ヒント: pEnt をすぐに削除しないで、削除する前に apply 関数を使用することにより、コレクション内部の属性定義を削除することができます。詳細については、109 ページの『コレクションのメモリー管理 (C++ のみ)』を参照してください。

C++

```
...  
pCol2->apply(deleteDKAttrDefICM);  
delete pCol2;  
...
```

動的データ・オブジェクト (DDO) の使用

ここでは、DDO の使用方法について説明し、さらに下記の方法について示す例を示します。

1. DKDDO をコンテンツ・サーバーと関連付ける。
2. DKDDO を作成します。
3. DKDDO 属性の永続 ID (PID) を作成する。
4. 属性を追加し、属性プロパティーを定義する。
5. DKDDO をフォルダーまたは文書として定義する。
6. 属性プロパティーの値を設定したり表示したりする。
7. DKDDO プロパティーを検査する。
8. 属性プロパティーを検査する。
9. DKDDO の内容を表示する。
10. DKDDO を削除する。

IBM Enterprise Information Portal for Multiplatforms アプリケーション内の動的データ・オブジェクト (DDO) には、DKDDO クラスを使用してください。DKDDO オブジェクトは、例えば Content Manager 文書やフォルダー、あるいはユーザー定義のオブジェクトなどの項目を表します。DKDDO オブジェクトには属性が含まれます。各属性には、名前、値、およびプロパティーがあります。それぞれの属性はデータ ID によって識別されます。属性に付けられる番号は 1 から始まる連続した番号です。この属性番号がデータ ID です。

属性の名前、値、および特性はさまざまなので、DKDDO は、さまざまなコンテンツ・サーバーや形式に基づくデータを表すことができる柔軟なものとなっています。例えば、Content Manager 内の異なる項目タイプの項目、またはリレーショナル・データベース内の異なるテーブルからの行などです。DKDDO そのものは、特定の 1 つの属性だけに適用されるのではなく、DKDDO 全体に適用されるプロパティーを持つことができます。

DKDDO の属性をコンテンツ・サーバーに設定したり、それらの属性を検索するために、add、retrieve、update、および delete メソッドを呼び出す前に、DKDDO をコンテンツ・サーバーと関連付けます。コンテンツ・サーバーは、DKDDO オブジェクトの作成時にパラメーターとして設定するか、あるいは setDatastore メソッドを呼び出すことにより設定できます。

すべての DKDDO には永続オブジェクト ID (PID) があり、これにはコンテンツ・サーバー内での属性の位置決めの情報が含まれています。

DKDDO の作成

DKDDO には、いくつかのコンストラクターがあります。DKDDO を作成するのに、パラメーターを指定しないでそのコンストラクターを呼び出していました。

Java

```
DKDDO ddo = new DKDDO();
```

C++

```
DKDDO ddo;
```

この DDO ddo は、追加の属性を格納するために動的に拡張する必要があります。コンストラクターをより効率的にするには、必要な属性の正確な数 (例: 10) を渡します。

Java

```
DKDDO ddo = new DKDDO(10);
```

C++

```
DKDDO ddo = new DKDDO((short)10);
```

重要: DKDatastoreICM や DKDatastoreOD などの API では、DKDatastore XX クラスの createDDO() メソッドを使用して DKDDO を作成してください。CM V8 では、これらのメソッドを使用して DDO を作成する必要があります。次の例では、Content Manager バージョン 8 の場合に、コンテンツ・サーバーとオブジェクト・タイプを両方とも渡して DKDDO を作成します。

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //create a CM datastore
DKDDO ddo=dsICM.createDDO("ICMSAMPLE", //create a DDO to hold an object type
DKConstant.DK_CM_DOCUMENT);
```

DDO の作成について詳しくは、CMBROOT¥Samples¥java¥icm にある SItemCreationICM サンプルを参照してください。

C++

```
// create a Content Manager datastore
DKDatastoreICM* dsICM = new DKDatastoreICM();
// create a DDO to hold an object type
DKDDO* ddo = dsICM->createDDO("ICMSAMPLE",
DK_CM_DOCUMENT);
```

DDO の作成について詳しくは、CMBROOT¥Samples¥cpp¥icm にある SItemCreationICM サンプルを参照してください。

その他のコネクタ (旧バージョンの Content Manager など) の場合は、次の例にコンテンツ・サーバーとオブジェクト・タイプを指定して DKDDO を作成できます。

Java

```
DKDatastoreDL dsDL=new DKDatastoreDL(); //create a CM datastore
DKDDO ddo=new DKDDO(dsDL, "DLSAMPLE"); //create a DDO to hold an object type
//DLSAMPLE in dsDL
```

C++

```
// create an earlier Content Manager datastore
DKDatastoreDL dsDL;
// create a DDO to hold an object type DLSAMPLE in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "DLSAMPLE");
```

使用するコンストラクターは、アプリケーションによって異なります。コンストラクターについては、「オンライン API 解説書」を参照してください。

DDO へのプロパティの追加

DDO を表現するための DKDDO オブジェクトを作成する際には、オブジェクトの項目タイプのプロパティ (文書、フォルダー、または項目) を指定する必要があります。

```
このプロパティーは、
DKDatastoreXX.createDDO(itemTypeName,itemPropertyType/SemanticType) メソッ
ドのオプションの 1 つとして渡すことができます。
DKDatastoreICM.createDDO(itemTypeName,itemPropertyType/SemanticType)
```

また、項目タイプのプロパティーを設定せずに DKDDO をすでに作成した場合は、次の例を使用して DDO のタイプを「文書」に設定できます。

Java

```
//----- Add the property that it is a document
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, new Short(DK_CM_DOCUMENT));
```

C++

```
any = DK_CM_DOCUMENT; // it is a document
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);
```

永続 ID (PID) の作成

各 DDO には永続 ID (PID) がなければなりません。PID には、コンテンツ・サーバーの名前、タイプ、ID、およびオブジェクト・タイプに関する情報が含まれています。PID は DDO の永続データの場所を識別します。例えば、旧バージョンの Content Manager コンテンツ・サーバーの場合、PID は項目 ID です。項目 ID は、retrieve、update、および delete 関数の最も重要なパラメーターの 1 つです。Content Manager V8 では、PID は 5 つの部分で構成されます (詳しくは、『Content Manager 8.2 の使用』を参照)。

add 関数の場合、コンテンツ・サーバーは項目 ID を作成してそれを戻します。例えば、DKDatastoreXX.createDDO() メソッドを提供しているコネクタは、DKDDO.add() 操作によって PID オブジェクトを自動的に作成します。

以下の例では、既知の項目を検索するための DDO を作成します。

Java

```
// Given a connected DKDatastoreICM object named "dsICM"
// Create a new DDO
DKDDO ddo = dsICM.createDDO("book",DKConstant.DK_CM_DOCUMENT);
// PID automatically created by the function above.
ddo.add(); // Add the new item to the datastore.
// PID Completed by the System
DKPidICM pid = (DKPidICM) ddo.getPidObject();
```

C++

```
// Given a connected DKDatastoreICM object named "dsICM"  
// Create a new DDO  
DKDDO* ddo = dsICM->createDDO("book",DK_CM_DOCUMENT);  
// PID automatically created by the function above.  
ddo->add(); // Add the new item to the datastore.  
// PID Completed by the System  
DKPidICM* pid = (DKPidICM*) ddo->getPidObject();
```

この例で作成された DDO を検索するには、コンテンツ・サーバーに接続し、`retrieve` 関数を呼び出します。

Content Manager 8.2 は、コネクタに PID クラスがあり、このクラスは `DKPid` のサブクラスです。このサブクラスは `DKPidICM` と呼ばれ、`DKDDO` および `dkResource` に使用される `pid` です。

データ項目およびプロパティの処理

`DKDDO` には、属性および属性のプロパティを `DKDDO` オブジェクトに追加するためのメソッドが用意されています。

項目タイプ `NameOfItemType` にタイプ `integer` の属性 `Name` があり、CM V8 リポジトリ内でヌル可能として定義されているとします。このエンティティの項目を処理する `DKDDO` オブジェクトを作成し、その `DKDDO` に 2 つのデータ項目を追加することができます。

次の例では、項目を作成し、属性プロパティを設定して、Content Manager 内の永続コンテンツ・サーバーに保管します。**要件:** 変数 `dsICM` 内の `DKDatastoreICM` に接続済みであることが前提です。 `varchar S_varchar`、`long integer S_integer`、`short integer S_short`、および `time S_time` を指定して、ユーザー定義の項目タイプ `S_withChild` を定義する必要があります (`SItemTypeCreationICM` API 実習サンプルに定義されているように)。

Java

```
// Create a new item in memory
DKDDO ddo = dsICM.createDDO("S_withChild", DKConstant.DK_CM_DOCUMENT);

// Set attributes (Additional attributes set in SItemCreationICM sample)
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"),
    "abcdefg");
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"),
    new Integer("123"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"),
    new Short("5"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"),
    java.sql.Time.valueOf("10:00:00"));

// Add to datastore
ddo.add();
```

この例の引用元である完全なサンプル・アプリケーション (SItemCreationICM) が CMBROOT¥Samples¥java¥icm ディレクトリーにあります。

C++

```
// Create a new item in memory
DKDDO* ddo = dsICM->createDDO("S_withChild", DK_CM_DOCUMENT);

// Set attributes (Additional attributes set in SItemCreationICM sample)
// NOTE: Values below are automatically converted to type DKAny
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_varchar"),
    DKString("this is a string value"));
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_integer"),
    (long) 123);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_short"),
    (short) 5);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_time"),
    DKTime("10.00.00"));

// Add to datastore
ddo->add();
```

この例の引用元である完全なサンプル・アプリケーション (SItemCreationICM) が CMBROOT¥Samples¥cpp¥icm ディレクトリーにあります。

属性にはプロパティー type を設定しなければなりません。nullable と他のプロパティーはオプションです。

次の例では、DDO の属性値にアクセスします。

Java

```
// Cast operations will enable access as subclass of Object type returned.  
// NOTE: Additional attributes accessed in SItemRetrievalICM sample.
```

```
String attrVal1 = (String) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"));  
Integer attrVal2 = (Integer) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"));  
Short attrVal3 = (Short) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"));  
Time attrVal4 = (Time) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"));  
  
System.out.println("Attr 'S_varchar' value: "+attrVal1);  
System.out.println("Attr 'S_integer' value: "+attrVal2);  
System.out.println("Attr 'S_short' value: "+attrVal3);  
System.out.println("Attr 'S_time' value: "+attrVal4);
```

この例の引用元である完全なサンプル・アプリケーション
(SItemRetrievalICM) が CMBROOT¥Samples¥java¥icm ディレクトリーにあります。

C++

```
// Assignment and cast operations converts values from DKAny to each type.  
// NOTE: Additional attributes accessed in SItemRetrievalICM sample.
```

```
DKString attrVal1 = ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_varchar"))).toString();  
long attrVal2 = (long) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_integer")));  
short attrVal3 = (short) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_short")));  
DKTimestamp attrVal4 = (DKTimestamp) ddo->getData(ddo->dataId(  
    DK_CM_NAMESPACE_ATTR, DKString("S_time")));  
  
cout << "Attr 'S_varchar' value: " << attrVal1 << endl;  
cout << "Attr 'S_integer' value: " << attrVal2 << endl;  
cout << "Attr 'S_short' value: " << attrVal3 << endl;  
cout << "Attr 'S_time' value: " << attrVal4 << endl;
```

この例の引用元である完全なサンプル・アプリケーション
(SItemRetrievalICM) が CMBROOT¥Samples¥cpp¥icm ディレクトリーにあります。

DKDDO と属性プロパティの取得

DKDDO を処理するには、まずそのタイプ (文書、フォルダー、または項目) が分かっている必要があります。次のサンプル・コードは、DDO タイプを判別する方法を示しています。

Java

```
short prop_id = ddo.propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    short type = ((Short) ddo.getProperty(prop_id)).shortValue();
    switch(type) {
        case DK_CM_DOCUMENT:
            // --- process a document
            ....
            break;
        case DK_CM_FOLDER:
            // --- process a folder
        case DK_CM_ITEM:
            // --- Process an item in Content Manager
            ....
            break;
    }
}
```

項目タイプのプロパティへのアクセスについては、
CMBROOT¥Samples¥java¥icm ディレクトリーにある **SItemRetrievalICM API**
実習サンプルを参照してください。

C++

```
unsigned short prop_id =
    ddo->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    unsigned short type = (unsigned short) ddo->getProperty(prop_id);
    switch(type) {
        case DK_CM_DOCUMENT:
            // process document
            ...
            break;
        case DK_CM_FOLDER:
            // process folder
            ...
            break;
    }
}
```

項目タイプのプロパティへのアクセスについては、
CMBROOT¥Samples¥cpp¥icm ディレクトリーにある **SItemRetrievalICM API**
実習サンプルを参照してください。

属性のプロパティを検索するには、以下のようにして属性の **data_id** を取得しなければなりません。その後、プロパティを取り出すことができます。

data_id も **property_id** も 1 から始まります。0 を指定すると例外が発生します。

Java

```
data_id = ddo.dataId("Title"); // get data_id of Title
// ----- Get the number of properties for the attribute
short number_of_data_prop = ddo.dataPropertyCount(data_id);
// ----- Display all data properties belonging to this attribute
//          using a loop; the index starts at 1
for(short i = 1; i <= number_of_data_prop; i++) {
    System.out.println(i + " Property Name = " +
        ddo.getDataPropertyName(data_id,i)
        + " value = " + ddo.getDataProperty(data_id,i));
}
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SItemRetrievalICM を参照してください。

C++

```
// get data_id of Title
data_id = ddo->dataId("Title");
// how many props does it have?
unsigned short number_of_data_prop = ddo->dataPropertyCount(data_id);
// displays all data properties belonging to this attribute
// notice that the loop index starts from 1, where
// 1 <= i <= number_of_data_prop
for (unsigned short i = 1; i <= number_of_data_prop; i++) {
    cout << i << " Property Name = " << ddo->
        getDataPropertyName(data_id, i) << " value = " << ddo->
        getDataProperty(data_id, i) << endl;
}
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SItemRetrievalICM を参照してください。

DDO 全体の表示

アプリケーション開発時には、デバッグのために DKDDO の内容を表示することが必要な場合があります。

Java

```
short number_of_attribute = ddo.dataCount();
short number_of_prop      = ddo.propertyCount();
short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
    System.out.println( k + " Property Name = " + ddo.getPropertyName(k) +
        ",%t    value = " + ddo.getProperty(k));
}
// list data-items and their properties
for (short i = 1; i <= number_of_attribute; i++) {
    System.out.println( i + " Attr. Name = " + ddo.getDataName(i) +
        ",%t    value = " + ddo.getData(i));
    number_of_data_prop = ddo.dataPropertyCount(i);
    for (short j = 1; j <= number_of_data_prop; j++) {
        System.out.println( "%t" + j + " Data Prop. Name = " +
            ddo.getDataPropertyName(i,j)      +
            ",%t    value = " +
            ddo.getDataProperty(i,j));
    }
}
```

DDO (およびすべてのサブコンポーネント) にアクセスして表示する方法の詳細な例については、CMBROOT¥Samples¥java¥icm ディレクトリーにある SItemRetrievalICM と、その printDDO() 静的関数を参照してください。

C++

```
unsigned short number_of_attribute = ddo->dataCount();
unsigned short number_of_prop;
unsigned short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
    cout << k << " Property Name = " << ddo->getPropertyName(k) <<
        ",%t    value = " << ddo->getProperty(k) << endl;
}
// list data-items and their properties
for (unsigned short i = 1; i <= number_of_attribute; i++) {
    cout << i << " Attr. Name = " << ddo->getDataName(i) <<
        << ",%t    value = " << ddo->getData(i) << endl;
    number_of_data_prop = ddo->dataPropertyCount(i);
    for (unsigned short j = 1; j <= number_of_data_prop; j++) {
        cout << "%t" << j << " Data Prop. Name = "
            << ddo->getDataPropertyName(i, j)
            << ",%t    value = " << ddo->getDataProperty(i, j)
            << endl;
    }
}
```

DDO (およびすべてのサブコンポーネント) にアクセスして表示する方法の詳細な例については、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SItemRetrievalICM と、その printDDO() 静的関数を参照してください。

DDO の削除 (C++ のみ)

DKDDO には 2 つの表現があります。1 つはメモリー内にあり、もう 1 つは永続コピーです。DKDDO をメモリーから削除するには、デストラクターを呼び出します。ただし、この処理を行ってもコンテンツ・サーバー内の永続コピーは変更されずに残ります。

コンテンツ・サーバー内の永続コピーを削除するには、`dkddo:del()` 関数を使用します。この関数は、メモリー内の DKDDO の表現には影響しません (属性値は DKAny オブジェクト内にあります)。デストラクターによって `dkCollection` および `dkDataObjectBase` へのオブジェクト参照が削除されます。それには、DKParts、DKFolder、DKDDO、および DKBlob への参照も含まれます。

拡張データ・オブジェクト (XDO) の使用

XDO は、リソース項目や文書パーツなどのリソース・コンテンツを保管できるコンポーネントです。

リソース項目 (XDO) は、非リソース項目 (DDO) を拡張したものです。リソース項目の作成は通常の項目とほとんど同様に行い、通常の項目とまったく同様のリソース項目領域が作成されます。リソース項目のタイプに応じて、XDO をさらに拡張することもできます。

Java

Class Hierarchy

Type	DDO	XDO	Extension
---	----	-----	-----
Lob	DKDDO	-> DKLobICM	
Text	DKDDO	-> DKLobICM	-> DKTextICM
Image	DKDDO	-> DKLobICM	-> DKImageICM
Stream	DKDDO	-> DKLobICM	-> DKStreamICM

Content Manager のみ: CM 8 の場合は、XDO が正しいサブクラスに属している必要があるため、DKDDO の作成には必ず `DKDatastoreICM` の `createDDO()` メソッドを使用する必要があります。これにより、DKDDO 構造の中で重要な情報が自動的にセットアップされ、リソース、CM 文書モデル、フォルダーなどの高度な機能が提供されます。リソース・タイプの項目 (`DKDatastoreICM.createDDO` から戻される) は、XDO 種別に従って正しい XDO またはサブクラスにキャストできます。項目の一般的な作成方法、および `DKDatastoreICM.createDDO()` 関数について詳しくは、`SItemCreationICM` サンプルを参照してください。

バイナリー・オブジェクト用の XDO を作成するには、`DKBlobxx` を使います (`xx` は特定のサーバーを表す接尾部)。例えば、Content Manager の場合は `DKBlobICM`、OnDemand の場合は `DKBlobOD`、ImagePlus for OS/390 の場合は `DKBlobIP` を使用します。`DKBlobxx` オブジェクトを作成する際には、それをコンテンツ・サーバー `DKDatastorexx` に渡さなければなりません。Content Manager の場合は、`DKLobICM` を使用して XDO を作成します。

XDO 永続的 ID (PID) の使用

XDO では、データを永続的に格納するために PID が必要です。XDO を使用してデータを検索したり保管したりするには、DKPidXDOxx を使用し、DKBlobxx の PID を指定しなければなりません。リレーショナル・データベースでは、コンテンツ・サーバーから永続データを検索するために、テーブル、列、およびデータ述部ストリングが必要です。リレーショナル・データベース (RDB) の場合、DKPidXDOxx にテーブル名、列名、およびデータ述部が必要です。

ICM コネクタでは、XDO である dkResource オブジェクトの pid を表すのに DKPidICM を使用します。

XDO プロパティについて

XDO のプロパティを設定するには、DKBlobxx のメソッドを使います。すべてのプロパティがすべてのコンテンツ・サーバーで使用できるとは限りません。ロード時に特定の値が指定されていなければ、プロパティのデフォルト値が設定されます。例えば、旧バージョンの Content Manager では次のデフォルトが使用されます。

RepType (表現タイプ)

デフォルトは FRN\$NULL です。VisualInfo for AS/400 の場合、" " (最初と最後が引用符で囲まれた 8 つのブランク・スペース) を使用しなければなりません。

ContentClass

デフォルトは DK_CM_CC_UNKNOWN です。有効な値については、Enterprise Information Portal の %cmbroot%\include ディレクトリーにある DKConstant2DL.h を参照してください。

AffiliatedType

デフォルトは DK_DL_BASE です。

AffiliatedData

デフォルトは NULL です。

旧バージョンの Content Manager でオブジェクト・コンテンツに正しく索引付けするには、拡張オブジェクト DKSearchEngineInfoDL に、SearchEngine、SearchIndex、および SearchInfo を設定しなければなりません。

Content Manager で XDO を 使用する方法については、158 ページの『項目の使用』を参照してください。

C++ のヒント: ContentClass の有効な値については、Content Manager に含まれているファイル INCLUDE/DKConstant2DL.h を参照してください。

DB2、ODBC、および DataJoiner の構成ストリング (C++ のみ)

ここでは、C++ の場合の DB2、ODBC、および DataJoiner の構成ストリングを定義します。

CC2MIMEFILE=(filename)

cmbcc2mime.ini ファイルを指定します (オプション)。

DSNAME=(content server name)

コンテンツ・サーバー名を指定します (オプション)。このコンテンツ・サーバーが統合 (Federated) で使用される場合、このオプションは自動的に設定されます。

AUTOCOMMIT=ON | OFF

自動コミットのオン/オフを指定します。デフォルトはオフです (オプション)。このコンテンツ・サーバーが統合 (Fed) で使用される場合、自動コミットは常にオンです。これは自動的に設定されます。

ここでは、C++ の場合の DB2、ODBC、および DataJoiner 接続ストリングを定義します。

NATIVECONNECTSTRING=(native connect string)

ネイティブ接続呼び出しに渡すためのネイティブ接続ストリングを指定します (オプション)。

SCHEMA=name

listEntities、listEntityAttrs、listPrimaryKeyNames、および listForeignKeyNames 関数で使用するスキーマを指定します (オプション)。

Java プログラミングのヒント

Content Manager V8 以降の場合、XDO は dkResource オブジェクトです。DKPidICM を使用して、リソース・オブジェクトの PID を表します。

旧バージョンの Content Manager、Content Manager for AS/400、および IP 390 の場合、項目 ID、パーツ ID、および RepType の組み合わせによって XDO を識別します。RDB の場合、XDO を識別するキーはテーブル、列、およびデータ述部のストリングの組み合わせです。独立型 XDO を処理するには、項目 ID とパーツ ID を提供します。システムが XDO に対してデフォルト値を提供するので、RepType はオプションです。

現在のコンテンツをコンテンツ・サーバーに追加するには、DKBlobxx の add メソッドを使います。後でそのオブジェクトを使ってその他のいくつかの操作を実行したい場合には、add の後にパーツ ID 値を検索することができます。

dkXDO に getPidObject() メソッドを使用して、DKPid オブジェクトを取得します。

システムが割り当てたパーツ ID を取得するには、add の後に以下の文を使用することができます。

Java

```
int partID = ((DKPidXDOICM)(axdo.getPidObject())).getPartId();
```

重要: 旧バージョンの Content Manager では、検索マネージャーによって索引付けされるパーツを追加するには有効なパーツ ID が必要です (パーツ ID を 0 には設定できません)。

このリリースでは、dkXDO の 2 つのメソッドが変更されました。つまり、DKPid dkXDO.getPid() は使用が推奨されなくなり、getPidObject に置き換えられました。これらのメソッドは、DKPidXDO を戻すために使用され、現在では、DKPid オブジェクトを戻します。

C++ プログラミングのヒント

Content Manager、VI400[®]、および IP390 の場合、項目 ID、パーツ ID、および RepType を組み合わせて XDO を識別します。リレーショナル・データベースの場合、テーブル名、列名、およびデータ述部の組み合わせが XDO を識別するキーとなります。独立型 XDO の場合は、項目 ID とパーツ ID を指定する必要があります。RepType は、システムがデフォルト値 (FRN\$NULL) を提供するため、オプションです。

add 関数には、パーツ ID を指定する必要があります。後でそのオブジェクトを使ってその他のいくつかの操作を実行したい場合には、add の後にパーツ ID 値を検索することができます。

重要: Content Manager コンテンツ・サーバー上で索引付けするために、検索マネージャー用にパーツを追加する場合は、有効なパーツ ID を持っている必要があり、パーツ ID を 0 に設定することはできません。

DDO のパーツとして使用する XDO のプログラミング

DDO がリソース・コンテンツ・オブジェクトの集合である文書を表している場合、XDO は単一のパーツ・オブジェクトを表します。XDO は、DDO のコンポーネント、または独立型オブジェクトとして扱うことができます。DDO のパーツとして XDO を使用すると、DDO は項目 ID を提供します。独立型オブジェクトとして XDO を使用する場合は、XDO の既存の項目 ID を使用します。

次の例では、文書を作成し、文書パーツを Content Manager に追加します。**要件:** ユーザー定義の項目タイプ S_docModel をシステム内で定義して文書モデルに分類する必要があり、またこの項目タイプは ICMBASE および ICMBASETEXT の両パーツ・タイプをサポートしている必要があります (SItemTypeCreationICM API 実習サンプルに定義されているように)。

Java

```
// Create a document
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Create parts
DKLobICM base = (DKLobICM) dsICM.createDDO("ICMBASE",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM baseText1 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);

// Set parts' MIME type (SResourceItemMimeTypesICM.txt sample)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Load content into parts (SResourceItemCreationICM sample)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load file
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// into memory
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR, DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Add new document to persistent datastore
ddoDocument.add();
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm
ディレクトリーにある SDocModelItemICM.java を参照してください。
SResourceItemCreationICM では、XDO の使用例がさらに示されています。

C++

```
// Create a document
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);
// Create Parts
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
        DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
        DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
        DK_ICM_SEMANTIC_TYPE_BASSETEXT);

// Set parts' MIME type (SResourceItemMimeTypesICM.txt sample)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");
// Load content into parts (SResourceItemCreationICM sample)
// Load the file into memory.
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");
// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
        ddoDocument->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);
// Add new document to persistent datastore
ddoDocument->add();
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SDocModelItemICM.java を参照してください。
SResourceItemCreationICM では、XDO の使用例がさらに示されています。

独立型 XDO のプログラミング

以下の例はすべて Content Manager 8.2 に特有のものです。旧バージョンの Content Manager、およびその他のコンテンツ・サーバーの例については、156 ページの『DDO を使用した項目の表示』、265 ページの『その他のコンテンツ・サーバーの使用』、および CMBROOT¥Samples ディレクトリーにあるサンプル・プログラムを参照してください。

バッファからの XDO の追加

以下の例は、バッファから Content Manager に XDO を追加する方法を示しています。この例では、XDO を作成し、コンテンツをメモリーにロードし、メモリーからコンテンツを永続的に保管します。**要件:** リソースを種別とするユーザー定義の項目タイプ S_lob をシステム内で定義する必要があります (SItemTypeCreationICM API 実習サンプルに定義されているように)。また、リソース・マネージャーと SMS コレクションの定義がセットアップ済みで、項目タイプまたはユーザーのデフォルトとして設定されている必要があります (SResourceMgrDefCreationICM、SSMSCollectionDefCreationICM、SResourceMgrDefSetDefaultICM、および SSMSCollectionDefSetDefaultICM の各サンプル内で行われているように)。

Java

```
// Create an empty resource object
DKLobICM lob = (DKLobICM) dsICM.createDDO("S_lob", DKConstant.DK_CM_DOCUMENT);

// Set the MIME type (SResourceItemMimeTypesICM.txt sample)
lob.setMimeType("application/msword");

// Load content into item's local memory
lob.setContentFromClientFile("SResourceItemICM_Document1.doc");

// Add to datastore with content already in memory
lob.add();
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SResourceItemCreationICM を参照してください。

C++

```
// Create an empty resource object
DKLobICM* lob = (DKLobICM*) dsICM->createDDO("S_lob", DK_CM_DOCUMENT);

// Set the MIME type (SResourceItemMimeTypesICM.txt sample)
lob->setMimeType("application/msword");

// Load content into item's local memory
lob->setContentFromClientFile("SResourceItemICM_Document1.doc");

// Add to datastore With content already in memory
lob->add();
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SResourceItemCreationICM を参照してください。

ファイルからの XDO の追加

次の例では、Content Manager 内のコンテンツ・サーバー (コンテンツをファイルから直接保管する) に XDO を追加します。**要件:** リソースを種別とするユーザー定義の項目タイプ S_lob をシステム内で定義する必要があります (SItemTypeCreationICM API 実習サンプルに定義されているように)。また、リソース・マネージャーと SMS コレクションの定義がセットアップ済みで、項目タイプまたはユーザーのデフォルトとして設定されている必要があります (SResourceMgrDefCreationICM、SSMSCollectionDefCreationICM、SResourceMgrDefSetDefaultICM、および SSMSCollectionDefSetDefaultICM の各サンプル内で行われているように)。

Java

```
// Create an empty resource object
DKTextICM text = (DKTextICM) dsICM.createDDO("S_text", DKConstant.DK_CM_ITEM);

// Set the MIME type (SResourceItemMimeTypesICM.txt sample)
text.setMimeType("text/plain");

// Store content directly from a file
text.add("SResourceItemICM_Text1.txt");
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SResourceItemCreationICM を参照してください。

C++

```
// Create an empty resource object
DKTextICM* text = (DKTextICM*) dsICM->createDDO("S_text", DK_CM_ITEM);

// Set the MIME type (SResourceItemMimeTypesICM.txt sample)
text->setMimeType("text/plain");

// Store content directly from a file
text->add("SResourceItemICM_Text1.txt");
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SResourceItemCreationICM を参照してください。

XDO への注釈オブジェクトの追加

次の例では、Content Manager 内の文書に注釈パーツを追加します。

要件: ユーザー定義の項目タイプ S_docModel をシステム内で定義して文書モデルに分類する必要がある、またこの項目タイプは ICMANNOTATION パーツ・タイプをサポートしている必要があります (SItemTypeCreationICM サンプルに定義されているように)。すでに永続保管されている文書のインスタンスが、ddoDocument 変数に指定されていることを前提とします。また、接続済みの DKDatastoreICM オブジェクトが変数 dsICM に指定されていることも前提です。

Java

```
// Check out / lock the item for update
dsICM.checkOut(ddoDocument);

// Create annotation part
DKLobICM annot = (DKLobICM) dsICM.createDDO("ICMANNOATATION",
                                             DKConstantICM.DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Set annotatoin MIME type (SResourceItemMimeTypesICM.txt sample)
annot.setMimeType("image/bmp");

// Load content into parts (SResourceItemCreationICM sample)
annot.setContentFromClientFile("myAnnotation.bmp");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(annot);

// Save the changes to the persistent datastore
ddoDocument.update();

// Check in / unlock the item after update
dsICM.checkIn(ddoDocument);
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm
ディレクトリーにある SDocModelItemICM を参照してください。

C++

```
// Check out / lock the item for update
dsICM->checkOut(ddoDocument);

// Create annotation part
DKLobICM* annot = (DKLobICM*) dsICM->createDDO("ICMANNOTATION",
                                                DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Set annotatoin MIME type (SResourceItemMimeTypesICM.txt sample)
annot->setMimeType("image/bmp");

// Load content into parts (SResourceItemCreationICM sample)
annot->setContentFromClientFile("myAnnotation.bmp");

// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
    ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)annot);

// Save the changes to the prsistent datastore
ddoDocument->update();

// Check in / unlock the item after update
dsICM->checkIn(ddoDocument);
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm デレクトリーにある SDocModelItemICM を参照してください。

XDO の処理例

以下の例は、独立型 XDO の使用方法を示しています。

XDO の検索、更新、および削除

コンテンツ・サーバー内のオブジェクトの取り出し、更新、または削除を実行するには、オブジェクトを表す XDO に、正しい項目 ID、パーツ ID および RepType を提供してください。

次の例では、Content Manager 内の XDO を検索、更新し、削除します。

要件: リソースを種別とするユーザー定義の項目タイプ S_text をシステム内で定義する必要があります (SItemTypeCreationICM API 実習サンプルに定義されているように)。また、リソース・マネージャーと SMS コレクションの定義がセットアップ済みで、項目タイプまたはユーザーのデフォルトとして設定されている必要があります (SResourceMgrDefCreationICM、SSMSCollectionDefCreationICM、SResourceMgrDefSetDefaultICM、および SSMSCollectionDefSetDefaultICM の各サンプル内で行われているように)。また、コンテンツ・サーバーにすでに存在するリソース項目の PID スtring が、変数 pidString に指定されていることを前提とします。

Java

```
// Given: String pidString

// Re-create Blank DDOs for Existing Item (SItemRetrievalICM sample)
DKLobICM lob = (DKLobICM) dsICM.createDDO(pidString);

// Retrieve the item with the resource content
lob.retrieve(DKConstant.DK_CM_CONTENT_YES);

// Check out / lock the item for update (SItemUpdateICM sample)
dsICM.checkOut(lob);

// Set the new MIME type (SResourceItemMimeTypesICM.txt sample)
lob.setMimeType("application/msword");

// Update datastore with new content
lob.update("SResourceItemICM_Document2.doc");

// Check in / unlock the item after update
dsICM.checkIn(lob);

// Delete item
lob.del();
```

このコード・サンプルは、CMBROOT¥Samples¥java¥icm ディレクトリーにある SResourceItemRetrievalICM、SResourceItemUpdateICM、および SResourceItemDeletionICM から引用したものです。

C++

```
// Given: DKString pidString

// Re-create Blank DDOs for Existing Item (SitemRetrievalICM sample)
DKLobICM* lob = (DKLobICM*) dsICM->createDDO(pidString);

// Retrieve the item with the resource content
lob->retrieve(DK_CM_CONTENT_YES);

// Check out / lock the item for update (SitemUpdateICM sample)
dsICM->checkOut(lob);

// Set the new MIME type (SResourceItemMimeTypesICM.txt sample)
lob->setMimeType("application/msword");

// Update datastore with new content
lob->update("SResourceItemICM_Document2.doc");

// Check in / unlock the item after update
dsICM->checkIn(lob);

// Delete item
lob->del();

// Free Memory
delete(lob);
```

このコード・サンプルは、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある
SResourceItemRetrievalICM、SResourceItemUpdateICM、および
SResourceItemDeletionICM から引用したものです。

XDO 関数の呼び出し

この例では、旧バージョンの Content Manager サーバーを使用して DKBlob クラスをテストする方法を示しています。この例の場合、XDO の項目 ID およびパーツ ID を知っている必要があります。

Java

```
public class txdomiscDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 5;
        String itemId = "GAWCVGGVFUG428UJ";
        String repType = "";
        // Check the number of arguments for main and determine what to do
        if (args.length == 3)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            itemId = args[2];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType + " " + itemId);
        }
        if (args.length == 2)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType);
        }
        if (args.length == 1)
        {
            partId =(short)Integer.parseInt(args[0], 10);
            System.out.println("You enter: java txdomiscDL " + partId );
            System.out.println("The supplied default repType = " + repType);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdomiscDL  ");
            System.out.println("No parameter, following defaults provided:");
            System.out.println("    default partId = " + partId);
            System.out.println("    default repType = " + repType);
            System.out.println("    default itemId = " + itemId);
        }

        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore");
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");

            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            System.out.println("repType=" + apid.getRepType());
            System.out.println("itemid=" + apid.getItemId());
            System.out.println("partId=" + apid.getPartId());
        }
        // continued...
    }
}
```

Java (続き)

```
// ----- Before retrieve
System.out.println("before retrieve:");
System.out.println(" contentclass=" + axdo.getContentClass());
System.out.print(" content length=" + axdo.length());
System.out.println(" (the length of this object instance - in memory)");
System.out.print(" getSize=" + axdo.getSize());
System.out.println(" (get the object size without retrieving object)");
System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
axdo.retrieve();

// ----- After retrieve
System.out.println("after retrieve:");
System.out.println(" contentclass=" + axdo.getContentClass());
System.out.print(" content length=" + axdo.length());
System.out.println(" (the length of this object instance - in memory)");
System.out.print(" getSize=" + axdo.getSize());
System.out.println(" (get the object size without retrieving object)");
System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
System.out.println(" affiliatedType=" + axdo.getAffiliatedType());
if (axdo.getAffiliatedType() == DK_DL_ANNOTATION)
{
    DKAnnotationDL ann =
    (DKAnnotationDL)(axdo.getExtension("DKAnnotationDL"));
    System.out.println("affil pageNumber=" + ann.getPageNumber());
    System.out.println("affil X=" + ann.getX());
    System.out.println("affil Y=" + ann.getY());
}
System.out.println("about to do open()...");
axdo.setInstanceOpenHandler("notepad", true);
int cc = axdo.getContentClass();
if ( cc == DK_DL_CC_GIF)
    axdo.setInstanceOpenHandler("lviewpro", true);
else if (cc == DK_DL_CC_ASCII)
    axdo.setInstanceOpenHandler("notepad", true);
else if (cc == DK_DL_CC_AVI)
    axdo.setInstanceOpenHandler("mplay32 ", true);
axdo.open();
dsDL.disconnect();
    dsDL.destroy();
}
catch (DKException exc)
{
    // ----- Handle the exceptions
}
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    long hsession;
    DKString itemId, repType;
    int partId;
    itemId = "GAWCVGGVFUG428UJ";
    repType = "FRN$NULL";
    partId = 2;

    cout <<"argc is "<<argc<<endl;
    if (argc == 1)
    {
        cout<<"invoke: txdomisc <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdomisc "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }

    cout << connecting Datastore" << endl;
    try
    {
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;
        hsession = (long) (dsDL.connection()->handle());
        cout << "datastore handle" << hsession <<endl;
    }
    // continued...
```

C++ (続き)

```
DKBlobDL* axdo = new DKBlobDL(&dsDL);
DKPidXDODL* apid = new DKPidXDODL;
apid ->setPartId(partId);
apid ->setPrimaryId(itemId);
apid ->setRepType(repType);
axdo ->setPidObject(apid);
cout<<"itemId= "<<axdo->getItemId()<<endl;
cout<<"partId= "<<((DKPidXDODL*) (axdo->getPidObject()))
->getPartId()<<endl;
cout<<"repType= "<<axdo->getRepType()<<endl;

//== before retrieve
cout<<"before retrieve:"<<endl;
cout<<" content class="<<axdo->getContentClass()<<endl;
cout<<" content length="<<axdo->length();
cout<<" (the length of this object instance - in memory)"<<endl;
cout<<" getSize="<<axdo->getSize();
cout<<" (get the object size without retrieving object)"<<endl;
cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
axdo->retrieve();

//== after retrieve
cout<<"after retrieve:"<<endl;
cout<<" content class="<<axdo->getContentClass()<<endl;
cout<<" content length="<<axdo->length();
cout<<" (the length of this object instance - in memory)"<<endl;
cout<<" getSize="<<axdo->getSize();
cout<<" (get the object size without retrieving object)"<<endl;
cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
cout<<" mimeType="<<axdo->getMimeType()<<endl;
int atype = axdo->getAffiliatedType();
cout<<" affiliatedType= "<<axdo->getAffiliatedType()<<endl;
if (atype == DK_DL_ANNOTATION)
{
    DKAnnotationDL* ann=(DKAnnotationDL*)axdo
    ->getExtension("DKAnnotationDL");
    cout <<" pageNumber= "<<ann->getPageNumber()<<endl;
    cout <<" partId= "<<ann->getPart()<<endl;
    cout <<" X="<<ann->getX()<<endl;
    cout <<" Y="<<ann->getY()<<endl;
}
//== open content
int concls = axdo->getContentClass();
if (concls == DK_DL_CC_ASCII)
    axdo->setInstanceOpenHandler("notepad", TRUE);
else if (concls == DK_DL_CC_GIF)
    axdo->setInstanceOpenHandler("lviewpro", TRUE);
else if (concls == DK_DL_CC_AVI)
    axdo->setInstanceOpenHandler("mplay32", TRUE);
axdo->open();
// continued...
```

C++ (続き)

```
        delete apid;
        delete axdo;
        dsDL.disconnect();
        cout<<"datastore disconnected"<<endl;
    }
    catch(DKException &exc)
    {
        cout << "Error id" << exc.errorId() << endl;
        cout << "Exception id " << exc.exceptionId() << endl;
        for(unsigned long i=0;i< exc.textCount();i++)
        {
            cout << "Error text:" << exc.text(i) << endl;
        }
        for (unsigned long g=0;g< exc.locationCount();g++)
        {
            const DKExceptionLocation* p = exc.locationAtIndex(g);
            cout << "Filename: " << p->fileName() << endl;
            cout << "Function: " << p->functionName() << endl;
            cout << "LineNumber: " << p->lineNumber() << endl;
        }
        cout << "Exception Class Name: " << exc.name() << endl;
    }
    cout << "done ..." << endl;
}
```

旧バージョンの Content Manager での XDO メディア・オブジェクトの追加

メディア・オブジェクトが追加されるたびに、FRN\$MEDIA テーブル内にエントリが作成されます。その項目には、メディア・ユーザー・データに関する情報が含まれています。物理メディア・オブジェクトは、ネットワーク・テーブルの中で指定されている VideoCharger コンテンツ・サーバーに保管されます。以下の例では、XDO の項目 ID を知っている必要があります。

Java

```
public class txdoAddVSDL implements DKConstantDL
{
    // ----- Main method
    public static void main(String[] args)
    {
        String fileName = "/icing1.mpg1";           //a media object
        String itemId = "K1A04EWBVHJAV1D7";        //a known itemId
        int partId = 45;
        // ----- Check the arguments for main
        if (args.length == 3)
        {
            fileName = args[0];
            partId = (int)Integer.parseInt(args[1], 10);
            itemId = args[2];
            System.out.println("You enter: java txdoAddVSDL " +
                fileName + " " + partId + " " + itemId);
        }
        if (args.length == 2)
        {
            fileName = args[0];
            partId = (int)Integer.parseInt(args[1], 10);
            System.out.println("You enter: java txdoAddVSDL " +
                fileName + " " + partId );
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 1)
        {
            fileName = args[0];
            System.out.println("You enter: java txdoAddVSDL " + fileName);
            System.out.println("The supplied default partId = " + partId);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdoAddVSDL <filename> <part ID> <item ID>");
            System.out.println("No parameter, following defaults will be provided:");
            System.out.println("    default fileName = " + fileName);
            System.out.println("    default partId = " + partId);
            System.out.println("    default itemId = " + itemId);
        }
        // ----- Processing
        try
        {
            // ----- connect to datastore
            DKDatastoreDL dsDL = new DKDatastoreDL();
            // replace following with your library server, userid, password
            System.out.println("connecting to datastore...");
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");
            // ----- create xdo and pid
            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            axdo.setPidObject(apid);
            // you must use the content class DK_DL_CC_IBMVSS for a media object
            axdo.setContentClass(DK_DL_CC_IBMVSS);
            System.out.println("contentClass=" + axdo.getContentClass());
            System.out.println("partId = " + axdo.getPartId());
        }
        // continued...
    }
}
```

Java (続き)

```
// ----- set up DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS = new DKMediaStreamInfoDL();
aVS.setMediaFullFileName(fileName);
// if fileName contain a list of media segments then use following
//      aVS.setMediaObjectOption(DK_VS_LIST_OF_OBJECT_SEGMENTS);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");
// following are optional, if not set default value will be provided
aVS.setMediaNumberOfUsers(2);
aVS.setMediaAssetGroup("AG");
// ----- same as defined in VideoCharger server
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");
axdo.setExtension("DKMediaStreamInfoDL", (dkExtension)aVS);
System.out.println("about to call add()");
axdo.add();
System.out.println("add successfully....");
System.out.println("after added check for status:");
boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
}
dsDL.disconnect();
dsDL.destroy();
}
catch (DKException exc) {
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception name " + exc.name());
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
catch (Exception exc){
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
}
```

C++

```
void main(int argc, char *argv[])
{
    DKString itemId, fileName;
    int partId;
    itemId = "K1A04EWBVBHJAV1D7";
    partId = 22;
    fileName = "/icing1.mpg1";
    if (argc == 1)
    {
        cout<<"invoke: txdoAddVSDL <fileName> <partId> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default fileName = "<<fileName<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        fileName = DKString(argv[1]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        // connect to datastore
        cout << "Connecting datastore ..." << endl;
        DKDatastoreDL dsDL;
        // replace following with your library server, user ID, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;
        // *** create xdo and pid
        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        axdo ->setPidObject(apid);
        // you must use the content class DK_DL_CC_IBMVSS for a media object
        axdo ->setContentClass(DK_DL_CC_IBMVSS);
        cout <<"itemId= "<<axdo->getItemId()<<endl;
        cout <<"partId= "<<axdo->getPartId()<<endl;
        cout <<"repType= "<<axdo->getRepType()<<endl;
        cout <<"content class="<< axdo->getContentClass()<<endl;
        // continued...
```


C++ (続き)

```
// *** setup DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS;
aVS.setMediaFullFileName(fileName);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");

//following are optional, if not set then default value will be provided
aVS.setMediaNumberOfUsers(1);
aVS.setMediaAssetGroup("AG");
// *** same as defined in VideoCharger server
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");

axdo ->setExtension("DKMediaStreamInfoDL", (dkExtension*)&aVS);
cout <<"about to do add()"<<endl;
axdo ->add();
cout<<"Object added successfully "<<endl;

cout<<"after added check for status:"<<endl;
DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)="<<
        axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;
}
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

XDO メディア・オブジェクトの削除

以下の例は、XDO メディア・オブジェクトを削除する方法を示しています。この例では、XDO の項目 ID、パーツ ID、および RepType (表現タイプ) を知っている必要があります。

Java

```
public class txdoDelVSDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int partId = 45;
        String repType = "";
        String itemId = "K1A04EWBVHJAV1D7";
        if (args.length == 3)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            itemId = args[2];
            System.out.println("You enter: java txdoDelVSDL " +
                + partId + " " + repType + " " + itemId);
        }
        // ----- Check the arguments for main
        if (args.length == 2)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            System.out.println("You enter: java txdoDelVSDL " +
                + partId + " " + repType);
        }

        if (args.length == 1)
        {
            partId =(short)Integer.parseInt(args[0], 10);
            System.out.println("You enter: java txdoDelVSDL " + partId );
            System.out.println("The supplied default repType = " + repType);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdoDelVSDL <part ID> <RepType> <item ID>");
            System.out.println("No parameter, following defaults will be provided:");
            System.out.println("    default partId = " + partId);
            System.out.println("    default repType = " + repType);
            System.out.println("    default itemId = " + itemId);
        }

        // ----- Processing
        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore...");
            // replace following with your library server, userid, password
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");

            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
            System.out.println("isMediaObject?=" + flag2);
        }
        // continued...
    }
}
```

Java (続き)

```
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
    // ----- set delete option for media object
    axdo.setOption(DK_DL_OPT_DELETE_OPTION,
        (Object)new Integer(DK_DL_DELETE_NO_DROPITEM_MEDIA_AVAIL));
    System.out.println("The delete option =" +
        (Integer)(axdo.getOption(DK_OPT_DL_DELETE_OPTION)));
}

System.out.println("about to call del(.. ");
axdo.del();
System.out.println("del successfully.....");
flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
System.out.println("after delete isMediaObject? = " + flag2);
System.out.println("about to call dsDL.disconnect()");
dsDL.disconnect();
dsDL.destroy();
}
// ----- Handle exceptions
catch (DKException exc) {
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception name " + exc.name());
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
catch (Exception exc){
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
}
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "Y68M1I@VYDG8SPQ4";
    partId = 1;
    repType = "FRN$NULL";
    if (argc == 1)
    {
        cout<<"invoke: txdoDelVSDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        itemId = DKString(argv[3]);
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<" "<<argv[2]<<" "
            <<argv[3]<<endl;
    }

    try
    {
        cout << "Connecting datastore ..." << endl;
        // replace following with your library server, user ID, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        apid ->setRepType(repType);
        axdo ->setPidObject(apid);
        cout <<"itemId= "<<axdo->getItemId()<<endl;
        cout <<"partId= "<<((DKPidXDODL*)(axdo->getPidObject()))
            ->getPartId()<<endl;
        DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
        cout <<"isMediaObject? = "<<flag2<<endl;
    }
    // continued...
```

C++ (続き)

```
if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)=
        "<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;

    cout<<"about to set the delete option for media object..."<<endl;
    DKAny delOpt = DK_DL_DELETE_NO_DROPITEM_MEDIA_AVAIL;
    axdo->setOption(DK_DL_OPT_DELETE_OPTION, delOpt);
    DKAny opt;
    axdo->getOption(DK_DL_OPT_DELETE_OPTION, opt);
    long lopt = opt;
    cout<<"The setted delete option = "<<lopt<<endl;

}
cout<<"about to do del()"<<endl;
axdo->del();
cout<<"del successfully..."<<endl;
flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
cout<<"after delete isMediaObject? = "<<flag2<<endl;
delete axdo;
delete apid;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

XDO メディア・オブジェクトの取り出し

以下の例は、XDO メディア・オブジェクトを取り出す方法を示しています。取り出されたオブジェクトには、メディア・オブジェクトそのものではなく、メディア・メタデータのみが含まれています。この例の場合、XDO の項目 ID およびパーツ ID を知っている必要があります。

Java

```
public class txdoretxsDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 45;
        String itemId = "K1A04EWBVHJAV1D7";
        String repType = "";
        System.out.println("Processing using the following values: ");
        System.out.println("    Part Id = " + partId);
        System.out.println("    RepType = " + repType);
        System.out.println("    Item Id = " + itemId);
        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore...");
            // ----- replace following with your library server, userid, password
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");
            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            System.out.println("repType=" + apid.getRepType());
            System.out.println("objectType=" + axdo.getObjectType());
            System.out.println("itemid=" + apid.getItemId());
            System.out.println("partId=" + apid.getPartId());

            boolean flag = axdo.isCategoryOf(DK_DL_INDEXED_OBJECT);
            boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
            System.out.println("isIndexedObject?=" + flag);
            System.out.println("isMediaObject?=" + flag2);
            if (flag)
            {
                DKSearchEngineInfoDL srch = (DKSearchEngineInfoDL)
                    axdo.getExtension("DKSearchEngineInfoDL");
                System.out.println("    serverName=" + srch.getServerName());
                System.out.println("    textIndex=" + srch.getTextIndex());
                System.out.println("    timeStamp=" + srch.getSearchTimestamp());
                System.out.println("    searchIndex=" + srch.getSearchIndex());
                System.out.println("    indexedState=" +
                    axdo.retrieveObjectState(DK_INDEXED_OBJECT));
            }
        }
    }
}
```

Java (続き)

```
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
}
System.out.println("before retrieve.....");
System.out.println(" lob length=" + axdo.length());
System.out.println(" size=" + axdo.getSize());
System.out.println(" createdTimestamp="+axdo.getCreatedTimestamp());
System.out.println(" updatedTimestamp="+axdo.getUpdatedTimestamp());
// ----- Perform the retrieve call
axdo.retrieve();

System.out.println("after retrieve.....");
System.out.println(" lob length=" + axdo.length());
System.out.println(" size=" + axdo.getSize());
System.out.println(" mimeType=" + axdo.getMimeType());
System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
System.out.println("affiliatedType=" + axdo.getAffiliatedType());
if (axdo.getAffiliatedType() == DK_DL_ANNOTATION)
{
    DKAnnotationDL ann =
        (DKAnnotationDL) (axdo.getExtension("DKAnnotationDL"));
    System.out.println("affil pageNumber=" + ann.getPageNumber());
    System.out.println("affil X=" + ann.getX());
    System.out.println("affil Y=" + ann.getY());
}
System.out.println("about to do open()...");
axdo.setInstanceOpenHandler("notepad", true); //default for Windows
int cc = axdo.getContentClass();
if ( cc == DK_DL_CC_GIF)
    axdo.setInstanceOpenHandler("lviewpro ", true); //use lviewpro
else if (cc == DK_DL_CC_AVI)
    axdo.setInstanceOpenHandler("mplay32 ", true); //use mplay32
else if (cc == DK_DL_CC_IBMVSS)
    axdo.setInstanceOpenHandler("iscoview ", true); //use iscoview
axdo.open();

dsDL.disconnect();
dsDL.destroy();
}
catch (DKException exc)
{
    ... ￥￥ handle exceptions and destroy the datastore+
}
}
```


C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "K1A04EWBVHJAV1D7";
    partId = 1;
    repType = "FRN$NULL";
    if (argc == 1)
    {
        cout<<"invoke: txdoRetxsDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        itemId = DKString(argv[3]);
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]
            <<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        cout << "Connecting datastore ..." << endl;
        // replace following with your library server, userid, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        apid ->setRepType(repType);
        axdo ->setPidObject(apid);
        cout <<"itemId= "<<axdo->getItemId()<<endl;
        cout <<"partId= "
            <<((DKPidXDODL*)(axdo->getPidObject()))->getPartId()<<endl;
    }
    // continued...
```

C++ (続き)

```
DKBoolean flag = axdo->isCategoryOf(DK_DL_INDEXED_OBJECT);
DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
cout <<"isIndexed? = "<<flag<<endl;
cout <<"isMediaObject? = "<<flag2<<endl;
if (flag)
{
    DKSearchEngineInfoDL* srchInfo = (DKSearchEngineInfoDL*)
        axdo->getExtension("DKSearchEngineInfoDL");
    cout<<" ServerName="<<srchInfo->getServerName()<<endl;
    cout<<" TextIndex="<<srchInfo->getTextIndex()<<endl;
    cout<<" srchEngine="<<srchInfo->getSearchEngine()<<endl;
    cout<<" srchIndex="<<srchInfo->getSearchIndex()<<endl;
    cout<<" indexedState="<<axdo
        ->retrieveObjectState(DK_DL_INDEXED_OBJECT)<<endl;
}

if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)= "
        <<axdo->retrieveObjectState(DK_DL_MEDIA_OBJECT)<<endl;
}

cout<<"before retrieve..."<<endl;
cout <<" length of lobdata = "<<axdo->length()<<endl;
cout<<" size of lobdata = "<<axdo->getSize()<<endl;
cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<<endl;
axdo->retrieve();
cout<<"after retrieve..."<<endl;
cout <<" length of lobdata = "<<axdo->length()<<endl;
cout <<" mimeType = "<<axdo->getMimeType()<<endl;
cout <<" size of lobdata = "<<axdo->getSize()<<endl;
cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<
    <endl;
// continued...
```

C++ (続き)

```
int atype = axdo->getAffiliatedType();
cout <<"affiliatedType= "<<axdo->getAffiliatedType()<<endl;
if (atype == DK_ANNOTATION)
{
    DKAnnotationDL* ann =
        (DKAnnotationDL*)axdo->getExtension("DKAnnotationDL");
    cout<<"  pageNumber= "<<ann->getPageNumber()<<endl;
    cout<<"  partId= "<<ann->getPart()<<endl;
    cout<<"  X= "<<ann->getX()<<endl;
    cout<<"  Y= "<<ann->getY()<<endl;
}
cout<<"about to do open()..."<<endl;
axdo->setInstanceOpenHandler("notepad", TRUE);
//default use Notepad in Windows
int concls = axdo->getContentClass();
if (concls == DK_DL_CC_GIF)
    axdo->setInstanceOpenHandler("lviewpro", TRUE);
//use lviewpro in Windows
else if (concls == DK_DL_CC_AVI)
    axdo->setInstanceOpenHandler("mplay32", TRUE);
//use mplay32 in Windows
else if (concls == DK_DL_CC_IBMVSS)
    axdo->setInstanceOpenHandler("iscoview", TRUE);
//use iscoview in Windows

axdo->open();

delete axdo;
delete apid;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout<<"Exception Class Name: "<<exc.name()<<endl;
}
cout << "done ..." << endl;
}
```

ストレージ・コレクションへの XDO の追加

ユーザー定義のストレージ・コレクション名と関連付けられている XDO オブジェクトを追加する場合は、拡張オブジェクト `DKStorageManageInfoxx` を使います (xx は特定のサーバーを表す接尾部)。

次の例では、旧バージョンの Content Manager サーバーの DKStorageManageInfoDL が使用されています。Content Manager バージョン 8 以降の場合については 141 ページの『Content Manager バージョン 8.2 の使用』を参照してください。

Java

```
String fileName = "e:¥¥test¥¥notepart.txt"; //file for add
int    partId = 0;                          //let system decide the partId
String itemId = "V5SPB$WBLOHIQ4YI";        //an existing itemId
DKDatastoreDL dsDL = new DKDatastoreDL();    //required datastore
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD",""); //connect to dstore
DKBlobDL axdo = new DKBlobDL(dsDL);         //create XDO
DKPidXDODL apid = new DKPidXDODL();         //create PID
apid.setPartId(partId);                     //set partId
apid.setPrimaryId(itemId);                  //set itemId
axdo.setPidObject(apid);                    //set PID object
axdo.setContentClass(DK_DL_CC_ASCII);       //set ContentClass

// ----- Create the DKStorageManageInfoDL
StorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                     //optional
aSMS.setCollectionName("TESTCOLLECT1");     //already defined in DL SMS
aSMS.setManagementClass("TESTMGT1");        //optional
aSMS.setStorageClass("FIXED");              //optional
axdo.setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo.add(fileName);                         //add from file
System.out.println("after add partId = " + axdo.getPartId());
//display the partId after add
dsDL.disconnect();                         // disconnect from and destroy datastore
dsDL.destroy();
// ----- Handle the exceptions
```

C++

```
DKString fileName="e:¥¥test¥¥notepart.txt"; //file for add
int    partId = 0;                          //let system decide the partId
DKString itemId = "V5SPB$WBLOHIQ4YI";        //an existing itemId
DKString rtype = "FRN$NULL";                //optional
DKDatastoreDL dsDL;                         //required datastore
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD"); //connect to dstore
DKBlobDL* axdo = new DKBlobDL(&dsDL);       //create XDO
DKPidXDODL* apid = new DKPidXDODL();        //create Pid
apid->setPartId(partId);                     //set partId
apid->setPrimaryId(itemId);                  //set itemId
apid->setRepType(rtype);                     //set repType
axdo->setPidObject(apid);                    //set pid object
axdo->setContentClass(DK_DL_CC_ASCII);       //set ContentClass

//---set DKStorageManageInfoDL---
DKStorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                     //optional
aSMS.setCollectionName("TESTCOLLECT1");     //already defined in DL SMS
aSMS.setManagementClass("TESTMGT1");        //optional
aSMS.setStorageClass("FIXED");              //optional
axdo->setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo->add(fileName);                         //add from file
System.out.println("after add partId = " + axdo->getPartId());
//display the partId after add
dsDL.disconnect();                         //disconnect from datastore
System.out.println("datastore disconnected");
```

検索索引付きオブジェクトおよびメディア・オブジェクトを Content Manager に追加する場合の例については、CMBROOT¥Samples ディレクトリーにある以下のコード・サンプルを参照してください。

- TxdoAddBsmsDL
- TxdosAddBsmsDL
- TxdoAddFsmsDL
- TxdosAddFsmsDL
- TxdomAddsmsDL

XDO のストレージ・コレクションの変更

既存の XDO のストレージ・コレクションは変更することができます。拡張オブジェクト DKStorageManageInfoICM をセットアップした後、changeStorage メソッドを呼び出します。

Java

```
System.out.println("about to call changeStorage().....");
axdo.changeStorage();
System.out.println("changeStorage() success.....");
```

C++

```
System.out.println("about to call changeStorage().....");
axdo->changeStorage();
System.out.println("changeStorage() success.....");
```

この例の引用元である完全なサンプル・アプリケーション (TxdoChgSmsICM) は、CMBROOT¥Samples¥java¥dl ディレクトリーまたは CMBROOT¥Samples¥cpp¥dl ディレクトリーにあります。

XML の使用 (Java のみ)

Enterprise Information Portal では、Java API を使用することにより、Content Manager との間で XML 文書のコンテンツを DDO および XDO としてインポートおよびエクスポートする機能をサポートしています。この機能によって、さまざまな情報システムから、それぞれのシステムに対して別個にインターフェースを開発することなく、さまざまなタイプのオブジェクト (データまたはマルチメディア・コンテンツなど) を Content Manager にインポートして保管し、検索することができます。例えば、あるデータベース・システムに保管されているオブジェクトがある場合、それを Enterprise Information Portal の Java API を使用して、XML ファイルに変換し、それから Content Manager にインポートすることができます。Content Manager に保管すれば、他の Content Manager オブジェクトに実行できる操作を、そのオブジェクトに対して行うことができます。

XML をインポートおよびエクスポートする機能の拡張

現在、インポートおよびエクスポート用の XML 関数 `DKDDO.toXML()` と `DKDDO.fromXML()` は、参照属性値、リンク、またはフォルダー・コンテンツのある項目をサポートしていません。エクスポート操作は正常に行われますが、特定のコンテンツ・サーバー内の項目に対する参照として PID が使用されます。その後で別のコンテンツ・サーバーに項目をインポートすると、これらの項目はターゲット・システムに格納されません。これは、PID がインポート元のコンテンツ・サーバーに固有であるからです。これら 2 つのインターフェースは新規項目のみを作成し、同じ項目の複数バージョンはサポートしません。

解決策として、参照される項目の処理、既存項目の検出と上書きなど、このインターフェースによってサポートされない機能を実行するツールを別途作成する必要があります。

EIP のバージョン 8.1 修正パッケージ 1 とバージョン 8.2 は、高度なインポートおよびエクスポート機能を備えた新しいサンプル・ツール (`TImportICM` と `TExportICM`)、およびサンプル・ツール API (`TExportPackageICM`) を提供しています。新しいサンプルの機能は、`SItemXMLImportExportICM` に例示されている `DKDDO.toXML()` と `DKDDO.fromXML()` の両インターフェースを大幅に拡張したものです。

新しいツールは、バージョン 8.1 修正パッケージとバージョン 8.2 に Java サンプルとしてパッケージされています。詳しくは、サンプルのインポート/エクスポート・ツールと API の資料、および `CMBROOT¥Samples¥java¥icm` ディレクトリーにある `TExportPackageICM.doc` を参照してください。

XML 文書のインポート

XML は、標準入力、ファイル、バッファー、および Web アドレス (URL) などの、さまざまなソースからインポートすることができます。XML ファイルを完全にインポートすることもできます。これらのコンストラクターは、XML 文書からコンテンツを抽出し、対応する `DKDDO` と、それに関連した `dkXDO` を作成します。それから `DDO` に対する追加メソッドを呼び出し、オブジェクトを `Content Manager` に追加します。新規の `DDO` は、`Content Manager` バージョン 8 項目タイプまたは旧バージョンの `Content Manager` 索引クラスに属し、`Content Manager` だけに保管することができます。自己参照 XML ファイルをインポートすると、元の XML ファイルを `XDO` として保管することができます。つまり、XML をインポート・プロセスで失うことなく、XML そのものを将来の使用で利用できるようにしておくことができます。

XML からコンテンツをインポートする場合は、`DKDDO` でこれらのメソッドを使用します。

```
toXML(DKNVPair xmlDestination, java.lang.String path, int options)
fromXML(DKNVPair xmlSource, int options)
```

旧バージョンの `Content Manager` で `DKDDO` コンストラクターを使用して XML をインポートすることのないようにしてください。

XML コンテンツのインポート時には、以下の要素に留意してください。

1. Content Manager または旧バージョンの Content Manager のみにインポートできることに注意してください。
2. インポート用のコンテンツを含む XML ファイルは、以下に示す XML 文書タイプ定義に準拠しなければなりません。
3. XML のインポートおよびエクスポートは、Java API によってのみサポートされます。

以下のセクションでは、XML コンテンツをインポートするための前提条件およびメソッドを説明しています。

- XML 文書タイプ定義 (DTD)
- XML 文書でのコンテンツの保管
- さまざまな XML ソースからのコンテンツの抽出
- Content Manager への XML コンテンツのインポート

XML 文書タイプ定義 (DTD)

コンテンツを Content Manager にインポートして XML として保管するには、`ddo.dtd` (`CMROOT¥samples¥java¥dl¥ddo.dtd` にある) に準拠する XML 文書にコンテンツを保管する必要があります。

```
<!ELEMENT ddo (pid?, pidIdStrings*, propertyCount?, property*, dataCount?, dataItem*, xdoValue?)>
<!ATTLIST ddo      entityName CDATA #REQUIRED
                  xmlns      CDATA #FIXED "http://www.omg.org/pub/docs/formal/97-12-12.pdf#ddo/EIP-7.1"
                  xdoType    CDATA #IMPLIED
                  dsType     CDATA #IMPLIED>

<!ELEMENT pid EMPTY>
<!ATTLIST pid      dsType      CDATA #IMPLIED
                  dsName      CDATA #IMPLIED
                  objectType   CDATA #IMPLIED
                  pidString    CDATA #IMPLIED>

<!ELEMENT pidIdStrings EMPTY>
<!ATTLIST pidIdStrings idPosition CDATA #REQUIRED
                    idValue      CDATA #REQUIRED>

<!ELEMENT propertyCount (#PCDATA)>
<!ELEMENT property EMPTY>
<!ATTLIST property      propertyId CDATA #IMPLIED
                    propertyName CDATA #IMPLIED
                    propertyValue CDATA #IMPLIED
                    propertyType  CDATA #IMPLIED>

<!ELEMENT dataCount (#PCDATA)>
<!ELEMENT dataItem (dataPropertyCount?, dataProperty+, (dataValue | dataValues))>
<!ATTLIST dataItem      dataId      CDATA #IMPLIED
                    dataName      CDATA #REQUIRED
                    dataNameSpace  CDATA #IMPLIED>

<!ELEMENT dataPropertyCount (#PCDATA)>
<!ELEMENT dataProperty EMPTY>
<!ATTLIST dataProperty      propertyId CDATA #IMPLIED
                    propertyName CDATA #IMPLIED
                    propertyValue CDATA #IMPLIED
                    propertyType  CDATA #IMPLIED>

<!ELEMENT dataValues (dataValueCount?, dataValue*)>
<!ATTLIST dataValues      collectionName CDATA #IMPLIED>
<!ELEMENT dataValueCount (#PCDATA)>
<!ELEMENT dataValue (#PCDATA | ddo | xdoRef | linkRef)*>
<!ELEMENT linkRef (linkSource, linkTarget, linkItem?)>
<!ATTLIST linkRef linkTypeName CDATA #REQUIRED>
<!ELEMENT linkSource (ddo)>
<!ATTLIST linkSource sameAsParentDDO (yes | no) "no">
<!ELEMENT linkTarget (ddo)>
<!ATTLIST linkTarget sameAsParentDDO (yes | no) "no">
<!ELEMENT linkItem (ddo)>
<!ELEMENT xdoRef (xdoPid, xdoIdStrings*, xdoValue)>
<!-- partId deprecated it is replaced by xdoIdString on an xdoRef -->
<!-- repType deprecated it is replaced by xdoIdString on an xdoRef -->
<!ELEMENT xdoPid EMPTY>
<!ATTLIST xdoPid      dsType      CDATA #REQUIRED
                    dsName      CDATA #IMPLIED
                    xdoType      CDATA #REQUIRED
                    objectType   CDATA #IMPLIED
                    partId      CDATA #IMPLIED
                    repType      CDATA #IMPLIED
                    pidString    CDATA #IMPLIED>

<!ELEMENT xdoIdStrings EMPTY>
<!ATTLIST xdoIdStrings idPosition CDATA #REQUIRED
                    idValue      CDATA #REQUIRED>

<!ELEMENT xdoValue (contentType?, specificInfo*, searchEngineInfo?, smsInfo?, xdoContent?)>
<!ATTLIST xdoValue      refType      CDATA #REQUIRED
                    refEncoding  CDATA #IMPLIED>
```

```

        mimeType      CDATA #REQUIRED
        XML-LINK      CDATA #IMPLIED
        HREF          CDATA #IMPLIED>
<!ELEMENT contentType (#PCDATA)>
<!ELEMENT specificInfo EMPTY>
<!ATTLIST specificInfo
        infoGroup      CDATA #REQUIRED
        infoName       CDATA #REQUIRED
        infoValue      CDATA #REQUIRED>
<!-- searchEngineInfo deprecated it is replaced by specificInfo -->
<!ELEMENT searchEngineInfo EMPTY>
<!ATTLIST searchEngineInfo
        searchEngine    CDATA #REQUIRED
        searchIndex     CDATA #REQUIRED
        searchInfo      CDATA #REQUIRED>
<!-- smsInfo deprecated it is replaced by specificInfo -->
<!ELEMENT smsInfo EMPTY>
<!ATTLIST smsInfo
        smsRetention    CDATA #IMPLIED
        smsCollection   CDATA #IMPLIED
        smsMgmtClass    CDATA #IMPLIED
        smsStorageClass CDATA #IMPLIED
        smsObjServer    CDATA #IMPLIED>
<!ELEMENT xdoContent (#PCDATA)>

```

XML 文書でのコンテンツの保管

XML ファイルは、Content Manager にインポートする、さまざまな文書またはフォルダーとして使用することができます。これらの文書およびフォルダーには、パーツを含めることもできます。以下の例は、まず典型的な XML データ項目 `dataItem` `dataId="1"` を示しています。この値は `Basuki` です。ただし、`DataItem 13` は `dataName DKParts` を使用します。これは自己参照 `XDO` と関係付けられています。

Content Manager のサンプル

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ddo SYSTEM "ddo.dtd">
<ddo entityName="MagXML3" dsType="ICM" xdoType="DKLobICM">
  <pid dsType="ICM" dsName="ICMNLSDb"/>
  <property propertyId="1" propertyName="item-type"
    propertyValue="document"/>
  <dataItem dataName="Classification3">
    <dataProperty propertyName="type" propertyValue="string" />
    <dataValue>B</dataValue>
  </dataItem>
  <dataItem dataName="PublisherName3">
    <dataProperty propertyName="type" propertyValue="string"/>
    <dataValue>PublisherName3</dataValue>
  </dataItem>
  <dataItem dataName="MagEdrXML3" dataNameSpace="CHILD">
    <dataProperty propertyName="type" propertyValue="collection+ddo"/>
  </dataItem>
  <dataValues collectionName="DKChildCollection">
    <dataValue>
      <ddo entityName="MagEdrXML3" dsType="ICM" xdoType="DKLobICM">
        <pid dsType="ICM" dsName="ICMNLSDb"/>
        <dataItem dataName="Sophias3.USPostal3">
          <dataProperty propertyName="type" propertyValue="string"/>
          <dataValue>Title of Book</dataValue>
        </dataItem>
        <dataItem dataName="Sophias3.Association3">
          <dataProperty propertyName="type" propertyValue="string"/>
          <dataValue>Sophias3.Association3</dataValue>
        </dataItem>
      </ddo>
    </dataValue>
  </dataValues>
  <dataItem>
    <xdoValue mimeType="text/plain" refType="file"
      XML-LINK="SIMPLE" HREF="TSophiaBefore.txt" >
    </xdoValue>
  </dataItem>
</ddo>

```

XML が Content Manager にオブジェクトを保管する方法を示すいくつかの例として、`DK\Samples\java\icm\` ディレクトリーにあるサンプルを参照してください。

旧バージョンの Content Manager のサンプル

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ddo SYSTEM "ddo.dtd">
<ddo entityName="DLSAMPLE">
  <pid dsType="DL" dsName="LIBSRVRN"/>
  <property propertyId="1" propertyName="item-type"
    propertyValue="document"/>
  <dataItem dataId="1" dataName="DLSEARCH_Author">
    <dataProperty propertyId="1" propertyName="type"
      propertyValue="string"/>
    <dataValue>Basuki</dataValue>
  </dataItem>
  . . .
  <dataItem dataId="13" dataName="DKParts">
    <dataProperty propertyId="1" propertyName="type"
      propertyValue="collection+xdo"/>
    <dataProperty propertyId="2" propertyName="nullable"
      propertyValue="false"/>
    <dataValues>
      <dataValue>
        <xdoRef>
          <xdoPid dsType="DL" xdoType="DKBlobDL"/>
          <xdoValue refType="self" mimeType="text/xml">
            <contentType>XML</contentType>
          </xdoValue>
        </xdoRef>
      </dataValue>
    </dataValues>
  </dataItem>
</ddo>
```

旧バージョンの Content Manager で XML を使ってオブジェクトを保管する方法については、samples ディレクトリーの次のコード・サンプルを参照してください。

- dlsamp01.xml
- dlsamp02.xml
- dlsamp03.xml
- dlsamp04.xml
- dlsamp05.xml
- dltypes01.xml

さまざまな XML ソースからのコンテンツの抽出

DKDDO メソッドは、標準入力、ファイル、バッファ、および Web アドレス (URL) などの、さまざまな XML ソースからコンテンツを抽出することができます。XML ソースからコンテンツを抽出し、インポート・プロセスを開始するには、DKDDO メソッドを呼び出します。

以下に示すのは、それぞれの XML ソースの例です。

ファイルからの XML:

Java

```
xmlSource = new DKNVPair("FILE", "dlsamp01.xml");
```

バッファからの XML:

Java

```
File file = new File("dlsamp01.xml");
int fileSize = (int) file.length();
byte[] data = new byte[fileSize];
DataInputStream dis = new DataInputStream(new FileInputStream(file));
dis.readFully(data);
String strBuffer = new String(data);
DKNVPair xmlSource = new DKNVPair("BUFFER", strBuffer);
int importOptions=DK_CM_XML_VALIDATION;
```

Web アドレス (URL) からの XML:

Java

```
xmlSource = new DKNVPair("URL", "file:///d://myxml//dlsamp01.xml");
// replace file:///d:// with http://www.webaddress.com/ for URL
Int importOptions=0;
```

Content Manager への XML コンテンツのインポート

以下の例では、次の基本的なステップを順次実行します。

1. DKDDO を作成します。
2. コンテンツ・サーバーを作成してそれに接続します。この場合は Content Manager または旧バージョンの Content Manager です。
3. 新規の DKDDO をコンテンツ・サーバーに追加します。この場合も Content Manager です。
4. dkDataObjectBase fromXML(DKNVPair xmlSource) または dkDataObjectBase fromXML(DKNVPair xmlSource, int options) を使用して XML ソースをインポートします。

結果の DKDDO は、ddo.dtd 仕様に準拠し、Content Manager 項目タイプまたは旧バージョンの Content Manager 索引クラスに属します。

Java

```
// ----- Construct a DDO by importing the XML document
xmlSource = new DKNVPair("FILE", "icmsamp01.xml");
int importOptions = DK_CM_XML_VALIDATION;
DKDDO ddo = new DKDDO();
ds = new DKDatastoreICM();
// ..... connect to the datastore
ddo.setDatastore (ds);
// ----- Add the DDO to the datastore
ddo.add()
// ----- Import the XML
ddo.fromXML(xmlSource, importOptions);
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SItemXMLImportExportICM.java を参照してください。

XML のエクスポート

DDO および XDO データを XML 文書として、Content Manager または旧バージョンの Content Manager からエクスポートすることができます。エクスポートには DKDDO の toXML(DKNVPair xmlDestination, String path, int options) メソッドを使用します。

制限: 項目がリンク、フォルダー・コンテンツ、または参照属性を使用して他の項目を参照している場合は、このサンプルに例示されている DKDDO.toXML() と DKDDO.fromXML() の両インターフェースを使用して、項目を一方のコンテンツ・サーバーからエクスポートし、他方にインポートすることはできません。このインポート操作を行うと新規項目が作成され、既存項目の更新や上書きは行われず、項目の複数バージョンはサポートされません。この機能を拡張するツールについては、90 ページの『XML をインポートおよびエクスポートする機能の拡張』を参照してください。

以下の例は、XML をエクスポートする方法を示しています。

Java

```
DKNVPair xmlDestination = null;
//----- Use this line to export to STDOUT
//xmlDestination = new DKNVPair("STDOUT", null);

//----- Use this line to export to a buffer
//xmlDestination = new DKNVPair("BUFFER", new Object());

//----- Use this line to export to a file
String xmlFile = "export.xml";
if(!fileName.equals("")){
    xmlFile = fileName;
}

String strOS = System.getProperty("os.name");
if (strOS.indexOf("Win") != -1) { // Windows OS
    xmlFile = outputPath + "¥¥" + xmlFile;
} else { // other than Win OS
    xmlFile = outputPath + "/" + xmlFile;
}

xmlDestination = new DKNVPair("FILE", xmlFile);

ddo.toXML(xmlDestination, outputPath, DKConstant.DK_CM_XDO_REFERENCE);
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icmディレクトリーにある SItemXMLImportExportICM.java を参照してください。

文書の作成と DKPARTS 属性の使用

DDO 内の DKPARTS 属性は、文書のパーツの集合を表しています。この属性の値は DKParts オブジェクトで、このオブジェクトは DKPart オブジェクトの集合です。DKPart オブジェクトは、文書パーツ に分類される項目タイプで、リソース・コンテンツを格納します。

Content Manager のみ: 文書は、Content Manager システムおよびユーザーの間で単独の単位として保管、検索、および交換できる項目です。文書 意味タイプが指定された項目は、文書を構成する情報を含んでいることが予期されますが、特定の文書モデルの厳密なインプリメンテーションであるとは限りません。文書 (文書モデルとも呼ばれる) として分類される項目タイプから項目が作成された場合、その項目は文書パーツを含んでおり、Content Manager が提供する文書モデルの特定のインプリメンテーションです。文書として分類される項目タイプは、文書意味タイプまたはフォルダー意味タイプの項目を作成できます。文書パーツには、例えばテキスト、イメージ、スプレッドシートなど、異なるタイプのコンテンツを含めることができます。

次の例では、文書を作成し、文書パーツを Content Manager に追加します。

要件: ユーザー定義の項目タイプ `S_docModel` をシステム内で定義し、文書モデルに分類する必要があります。また、この項目タイプは `ICMBASE` および `ICMBASETEXT` の両パーツ・タイプをサポートしている必要があります (`SItemCreationICM` API 実習サンプルに定義されているように)。

Java

```
// Create a document
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Create parts
DKLobICM base = (DKLobICM) dsICM.createDDO("ICMBASE",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM baseText1 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);

// Set parts' MIME type (SResourceItemMimeTypesICM.txt sample)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Load content into parts (SResourceItemCreationICM sample)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load file
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// into memory
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR, DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Add new document to persistent datastore
ddoDocument.add();
```

パーツを使用した文書の作成については、`CMBROOT¥Samples¥java¥icm` ディレクトリーにある `SDocModelItemICM` API 実習サンプルを参照してください。

C++

```
// Create a document
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);

// Create Parts
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
                                                DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
                                                       DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
                                                       DK_ICM_SEMANTIC_TYPE_BASSETEXT);

// Set parts' MIME type (SResourceItemMimeTypesICM.txt sample)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");

// Load content into parts (SResourceItemCreationICM sample)
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load the file into memory.
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
    ddoDocument->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);

// Add new document to persistent datastore
ddoDocument->add();
```

パーツを使用した文書の作成については、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SDocModelItemICM API 実習サンプルを参照してください。

DDO は、パーツ・コレクション内のすべてのパーツを所有しています。パーツの更新と削除は、文書 DDO を使用して行います。

以下の例は、DDO からパーツを取り出し、パーツにアクセスする方法を示しています。

Java

```
// NOTE: Print function provided in SDocModelItemICM sample

// Get the DKParts object.
short dataid = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
if(dataid==0)
    throw new Exception("No DKParts Attribute Found! Either item type does not
        support parts or the document has not been explicitly retrieved.");
DKParts dkParts = (DKParts) ddoDocument.getData(dataid);

// Go through part list
dkIterator iter = dkParts.createIterator(); // Create an Iterator
while(iter.more()){                        // While there are items left
    DKDDO part = (DKDDO) iter.next();      // Move pointer & return next
    System.out.println("Item Id: "+((DKPidICM)part.getPidObject()).getItemId());
}
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SDocModelItemICM を参照してください。

C++

```
// NOTE: Print function provided in SDocModelItemICM sample

// Get the DKParts object.
short dataid = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if(dataid==0)
    throw DKException("No DKParts Attribute Found! Either item type does not
        support parts or the document has not been explicitly retrieved.");
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(dataid);

// Go through part list
dkIterator* iter = dkParts->createIterator(); // Create an Iterator
while(iter->more()){                          // While there are items
    DKDDO* part = (DKDDO*) iter->next()->value(); // Move pointer & return next
    cout << "Item Id:" << ((DKPidICM*)part->getPidObject())->getItemId()<< endl;
}
delete(iter);                                // Free Memory
```

パーツを使用した文書の作成については、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SDocModelItemICM API 実習サンプルを参照してください。

フォルダーの作成と DKFOLDER 属性の使用

フォルダー DDO では、文書の集合およびそのフォルダーに属する他のフォルダーを表すのに、DKFOLDER 属性を使用します。この属性の値は DKFolder オブジェクトであり、これは DDO の集合です。次に示すように、DKParts 属性が設定されると DKFolder 属性が設定されます。

Content Manager のみ: フォルダーは、フォルダー 意味タイプが指定された、種別と関係ない任意の項目タイプの項目です。フォルダー意味タイプの項目は、非リソース項目の機能、および項目タイプ種別 (文書モデルやリソースなど) に備わっている追加機能のほかに、Content Manager が提供するフォルダー機能を備えています。

す。フォルダーには、文書やサブフォルダーなど任意のタイプの項目をいくつでも含めることができます。フォルダーは、属性別に索引付けされます。

次の例では、フォルダーを作成し、コンテンツを Content Manager に追加します。

要件: ユーザー定義の項目タイプ `S_simple` をシステム内で定義する必要があります (SItemTypeCreationICM API 実習サンプルに定義されているように)。

Java

```
// Create new folder in memory
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Create and save contents to place in folder
DKDDO ddoDocument = dsICM.createDDO("S_simple", DKConstant.DK_CM_DOCUMENT);
DKDDO ddoFolder2 = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
DKDDO ddoItem = dsICM.createDDO("S_simple", DKConstant.DK_CM_ITEM);
ddoDocument.add();
ddoItem.add();
ddoFolder2.add();

// Access the DKFolder attribute
DKFolder dkFolder = (DKFolder)
ddoFolder.getData(ddoFolder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                                   DKConstant.DK_CM_DKFOLDER));

// Add contents to folder
dkFolder.addElement(ddoDocument);
dkFolder.addElement(ddoItem);
dkFolder.addElement(ddoFolder2); // Note, Folders can contain sub-folders.

// Save the folder in the persistent datastore.
ddoFolder.add();
```

フォルダーの作成について詳しくは、CMBROOT¥Samples¥java¥icm フォルダーにある SFolderICM API 実習サンプルを参照してください。

C++

```
// Create new folder in memory
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);

// Create and save contents to place in folder
DKDDO* ddoDocument = dsICM->createDDO("S_simple", DK_CM_DOCUMENT);
DKDDO* ddoFolder2 = dsICM->createDDO("S_simple", DK_CM_FOLDER);
DKDDO* ddoItem = dsICM->createDDO("S_simple", DK_CM_ITEM);
ddoDocument->add();
ddoItem->add();
ddoFolder2->add();

// Access the DKFolder attribute
DKFolder* dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(
    ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER));

// Add contents to folder
dkFolder->addElement(ddoDocument);
dkFolder->addElement(ddoItem);
dkFolder->addElement(ddoFolder2); // Note, Folders can contain sub-folders.

// Save the folder in the persistent datastore.
ddoFolder->add();
```

フォルダーの作成について詳しくは、CMBROOT¥Samples¥cpp¥icm フォルダーにある SFolderICM API 実習サンプルを参照してください。

Content Manager バージョン 8 では、フォルダー項目はフォルダー・コンテンツを所有しません。1 つの項目を複数のフォルダーに追加できます。フォルダーから項目を除去しても、単にシステムによって管理されているフォルダーとコンテンツの関係が分断されるだけです。フォルダー内の項目の更新と削除は別に行う必要があります。Content Manager の旧バージョンでは、DDO がコレクション内のコンテンツを所有します。

以下の例は、DDO からフォルダー・コンテンツを取り出し、コンテンツにアクセスする方法を示しています。

Java

```
// NOTE: Print function provided in SFolderICM API Education Sample

// Get the DKFolder object.
short dataid = folder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                             DKConstant.DK_CM_DKFOLDER);

if(dataid==0)
    throw new Exception("No DKFolder Attribute Found! DDO is either not a
        Folder or Folder Contents have not been explicitly retrieved.");
DKFolder dkFolder = (DKFolder) folder.getData(dataid);

// Access contents
dkIterator iter = dkFolder.createIterator(); // Create an Iterator
while(iter.more()){                          // While there are items left
    DKDDO ddo = (DKDDO) iter.next();          // Move to & return next element
    System.out.println("Item Id: "+((DKPidICM)ddo.getPidObject()).getItemId());
}
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥java¥icm ディレクトリーにある SFolderICM 実習サンプルを参照してください。

C++

```
// NOTE: Print function provided in SFolderICM API Education Sample

// Get the DKFolder object.
short dataid = folder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
if(dataid==0)
    throw DKException("No DKFolder Attribute Found! DDO is either not a Folder
        or Folder Contents have not been explicitly retrieved.");
DKFolder* dkFolder = (DKFolder*)(dkCollection*) folder->getData(dataid);

// Access contents
dkIterator* iter = dkFolder->createIterator(); //Create an Iterator
while(iter->more()){                          //While there are items left
    DKDDO* ddo = (DKDDO*) iter->next()->value(); //Move to & return next element
    cout << "Item Id: " << ((DKPidICM*)ddo->getPidObject())->getItemId()<< endl;
}
```

完全なサンプル・アプリケーションについては、CMBROOT¥Samples¥cpp¥icm ディレクトリーにある SFolderICM 実習サンプルを参照してください。

DKAny の使用 (C++ のみ)

DKAny には、あらゆるオブジェクトが入れられ、そのタイプは実行時に変えることができます。DKAny オブジェクトとしては、下記のタイプのどれも可能です。

- null
- (unsigned) short
- (unsigned) long
- double
- char

- TypeCode
- DKBoolean
- DKString
- DKDate
- DKTime
- DKTimestamp

値が割り当てられていない場合 (DKAny any)、DKAny は必ず NULL になります。
値が割り当てられた後は、DKAny::isNull() は FALSE を返します。

上記以外にも、DKAny オブジェクトは下記のオブジェクト・リファレンス・タイプを入れることができます。

- dkDataObjectBase*
- dkCollection*
- void*

タイプ・コードの使用

typeCode 関数を呼び出すことにより、DKAny オブジェクトの現在のタイプを決定できます。この関数は、TypeCode オブジェクトを返します。つまり、null の場合は tc_null を、short の場合は tc_short を返します。typeCode の完全なリストについては、「オンライン API 解説書」を参照してください。

DKAny のメモリー管理

DKAny に含まれるオブジェクトがオブジェクト・リファレンス・タイプでない場合、含まれるオブジェクトのメモリーは DKAny によって管理されます。オブジェクト参照を呼び出す各種コピー関連操作で作成されるのは、そのポインタのコピーだけです。コピーや削除においては、オブジェクト・リファレンス・タイプを維持する必要があります。

コンストラクターの使用

DKAny には、タイプごとにサポートされるコンストラクターがあります。下記の例は、前の部分に示されているタイプを含む DKAny オブジェクトを作成する方法を示しています。

C++

```
DKAny any1((unsigned short) 10);           // contains unsigned short 10
DKAny any2((long) 200);                     // contains long 200
DKAny any3(DKString("any string"));         // contains DKString
DKAny any4(DKTime(10,20,30));               // contains DKTime
DKAny any5((dkDataObjectBase*) new DKDDO); // contains DKDDO
DKAny any6(new MyObject(5,"abc"));           // contains MyObject
DKAny any7(new DKDDO);                      // shorter form of any5
```

タイプ・コードの入手

DKAny 内のオブジェクトのタイプ・コードを調べるには、typeCode 関数を使います。

C++

```
DKAny::TypeCode type_code;
type_code = any1.typeCode(); // type_code is tc_ushort
type_code = any4.typeCode(); // type_code is tc_time
type_code = any5.typeCode(); // type_code is tc_dobase (object ref)
type_code = any6.typeCode(); // type_code is tc_voidptr since
                             // MyObject is not recognized by DKAny
```

DKAny に新しい値を代入する

新しい値を既存の DKAny オブジェクトに代入するには、等号 (=) 代入演算子を使います。DKAny には、各タイプ・コードごとに代入演算子が用意されています。

C++

```
DKAny any;           // any contains null
long vlong = 300;
DKTimestamp vts(1997,8,28,10,11,12,999);
dkDataObjectBase* dobase =
(dkDataObjectBase*) new DKDDO;
any = vlong;         // any contains long 300
any = vts;           // any contains timestamp
any = dobase;        // any contains ddo
any = new DKDDO;     // any contains ddo
```

DKAny の値の代入

DKAny を標準のタイプに戻すには、キャスト演算子が必要です。例えば、以下のようになります。

C++

```
vlong      = (long) any2;           // sets vlong to 200
DKTime at  = (DKTime) any4;         // sets at to (10,20,30)
DKDDO* ddo = (DKDDO*) ((dkDataObjectBase*) any5); // extract the ddo
dkDataObjectBase* dobase = any7;    // extract the DDO
```

タイプが一致しない場合、無効なタイプ変換の例外になります。したがって、DKAny を標準タイプに変換する前に、タイプ・コードを調べる必要があります。

C++

```
if (any5.typeCode() == DKAny::tc_dobase)
    dobase = (dkDataObjectBase*) any5;
```

DKAny の型を調べるには、下記のような case 文を作成できます。

C++

```
switch(any.typeCode()) {
    case DKAny::tc_short:
        // operation for short
        ...
        break;
    case DKAny::tc_ushort:
        // operation for unsigned short
        ...
        break;
    ... etc.
}
```

DKAny オブジェクトがオブジェクト・リファレンスを含む場合は、DKAny の内容を void ポインターとして入手した後、それを適切なタイプにキャストすることができます。ただしこの操作は、DKAny 内で使用されているタイプ・コードを知っている場合のみ使用してください。

C++

```
// knows exactly any5 contains DKDDO
ddo = (DKDDO*) any5.value();
```

DKAny の表示

cout を使用して、DKAny オブジェクトの内容を表示することができます。

C++

```
cout << any3 << endl; // displays "any string"
cout << any4 << endl; // displays "10:20:30"
cout << any5 << endl; // displays "(dkDataObjectBase*) <address>",
                        // where address is the memory location of the ddo
```

DKAny の破棄

DKAny にオブジェクト・リファレンスを入れることはできますが、オブジェクト・リファレンス・タイプのメモリーを管理することはありません。したがって、そのようなタイプのメモリーは自分で管理する必要があります。下記の例は、DKAny オブジェクトのメモリー管理の例です。

C++

```
DKDDO* ddo = new DKDDO;           // creates a DKDDO in the heap
DKAny anyA((dkDataObjectBase*)ddo);
DKAny* anyB = new DKAny(anyA);     // creates anyB in the heap
                                   // anyA and anyB contains a
                                   // reference to the same ddo

...
delete anyB;                       // delete anyB, does not delete ddo
if (anyA.typeCode() == DKAny::tc_dobase)
    delete ((dkDataObjectBase*) anyA.value()); // deletes the ddo
```

最後の delete 文は、有効範囲から出る前に実行しなければなりません。そうしないと、anyA が削除されてしまい、DDO でメモリー・リークが発生してしまいます。

プログラミング上のヒント

推奨事項: 整数リテラルを DKAny に変換する場合、望ましくないタイプ変換を避けるために明示的にタイプを宣言するのが賢明です。下記のとおりです。

C++

```
any = 10;                          // ambiguous
any = (unsigned long) 10;          // unambiguous
any = (short) 4;                   // unambiguous
```

コレクションおよびイテレーターの使用

dkCollection は、コレクションを処理するメソッドを提供する抽象クラスです。DKSequentialCollection は、dkCollection を具体化したインプリメンテーションです。その他のコレクションは、DKSequentialCollection のサブクラスとしてインプリメントされています。これらのコレクションには、メンバーとしてのデータ・オブジェクトが含まれています。

コレクションのメンバーは、通常は同じタイプのオブジェクトです。しかし、別のタイプのメンバーをコレクションに含めることもできます。

C++ のみ: 新しいメンバーが追加されると、コレクションがそれを所有します。メンバーを取得する場合には、コレクション内の DKAny のポインターを取得することになります。このオブジェクトはコレクションに属します。つまり、その DKAny メンバーのメモリーはコレクションが管理します。DKAny にオブジェクト・リファレンスを入れることは可能ですが、DKAny にオブジェクト・リファレンス・タイプのメモリー管理はできません。ですから、これらのリファレンス・タイプのメモリーは自分で管理しなければなりません。

順次コレクション・メソッドの使用

DKSequentialCollection には、メンバーを追加、検索、除去、および置換するメソッドがあります。それらのほかに、sort メソッドがあります。以下の例は、新規メン

バーをコレクションに追加する方法を説明しています (addElement メソッドは、オブジェクトをパラメーターとして受け取ります)。

Java

```
DKSequentialCollection sq = new DKSequentialCollection();
String str = " first member ";
sq.addElement(str);           // add a new element at the last position
```

C++

```
DKSequentialCollection sq;
DKAny any = DKString(" first member ");
sq.addElement(any);           // add a new element at last position
                                // any will be copied into the collection
                                // you own the original any, the collection
                                // owns the copy
```

順次イテレーターの使用

イテレーターを使用すれば、コレクションのメンバーを網羅することができます。API には、 dkIterator と DKSequentialIterator という 2 つのタイプのイテレーターがあります。

Java

dkIterator は基本イテレーターで、 next、more、および reset メソッドをサポートしています。サブクラス DKSequentialIterator には、 more メソッドが含まれています。コレクションの createIterator メソッドを呼び出すことにより、イテレーターが作成されます。イテレーターを作成したら、setToFirst() メソッドを使用して、最初の項目をポイントします。以下に、イテレーターの使用例を示します。

```
dkIterator iter = sq.createIterator(); // create an iterator for sq
Object member;
iter.setToFirst();
member = iter.at();
while(iter.more()) {                // While there are more members
    member = iter.next();            // move to the next member and get it

    System.out.println(member);
    ....
}
```

C++

イテレーターは、コレクションのメンバーの全部を網羅する反復操作のためのものです。イテレーターには 2 つのタイプがあります。1 つは基本イテレーター `dkIterator` で、これは `next` 関数、`more` 関数、および `reset` 関数をサポートします。もう 1 つはそのサブクラスの `DKSequentialIterator` で、これにはさらに多くの関数が含まれます。イテレーターは、コレクションの `createIterator` 関数を呼び出すことによって作成されます。この関数は、新しいイテレーターを作成し、それを戻します。コレクションに対して反復操作を実行する場合は、下記のようなコードを使います。

```
dkIterator* iter = sq.createIterator(); // create an iterator for sq
DKAny* member;

// while there are more members
// get the current member and
// advance iter to the next member

while(iter->more()) {
    member = iter->next();

    cout << *member << endl; // display it, if you want to
    ...                      // do other processing
}
delete iter;                 // do not forget to delete iter
```

`DKSequentialIterator` には、イテレーターをどちらの方向にも移動させられる追加メソッドがあります。上記の例は、以下のように書き直すことができます。

Java

```
// ----- Create a sequential iterator for sq
DKSequentialIterator iter =
    (DKSequentialIterator) sq.createIterator();
Object member;
iter.setToFirst();
while(iter.more()) {
    member = iter.at(); // get the current member
    ....               // do other processing
    iter.setToNext();  // advance to the next position
}
```

C++

```
DKSequentialIterator* iter = // create an iterator for sq
(DKSequentialIterator*) sq.createIterator();
DKAny* member;
while(iter->more()) {
    member = iter->at(); // get the current member
    ...               // do other processing
    iter->setToNext();  // advance to the next position
}
delete iter;
```

このコードを使用すれば、次のメンバーに移る前に現行メンバーに対してある種の操作を実行することが可能です。例えば、メンバーを新しいもので置換したり、それを削除したりできます。

Java

```
String st1 = "the new first member";
sq.replaceElementAt(st1, iter); // replace current member with a new one
....                          // or
sq.removeElementAt(iter);      // remove the current member
....
```

C++

```
any = DKString("the new first member");

sq.replaceElementAt(any, *iter); // replace current member with a new one
...                             // or
sq.removeElementAt();           // remove the current member
...
```

ヒント: 現行メンバーを除去すると、イテレーターは次のメンバーに進みます。ループ内でメンバーを除去するには、以下の例のように検査をします。現在のメンバーを除去した後に、その次のメンバーをスキップしてしまうのを避けるために、除去の状態を検査してください。

Java

```
....
if (removeCondition == true)
    sq.removeElementAt(iter); // remove current member, do not advance the
                              // iterator since it is advanced to the next
                              // after the removal operation
else
    iter.setToNext();         // if no removal, advance the iterator to the
....                         // next position
```

C++

```
...
if (removeCondition == TRUE)
    sq.removeElementAt(*iter); // remove current member, do not advance iter
                              // since it is advanced to the next after
                              // the removal operation
else
    iter->setToNext();          // no removal, advance the iterator
...                           // to the next position
```


コレクションのメモリー管理 (C++ のみ)

コレクションは、そのメンバー、つまり DKAny オブジェクトのメモリーを管理します。DKAny 内のオブジェクトがオブジェクト・リファレンス・タイプの場合、DKAny オブジェクトに関する同じ規則がここでも当てはまります。下記の操作を実行する時点で、メモリーを管理する必要があります。

- コレクションを破棄する。
- メンバーを置き換える。
- メンバーを削除する。

次の例は、そのような場合にメモリーを管理する方法を示しています。

C++

```
// retrieve the member and hang-on to it
member = iter->at();

// code to handle this member as to prevent memory leaks
if (member->typeCode() == DKAny::tc_dobase) {
    // delete it if no longer needed
    delete ((dkDataObjectBase*) member->value());
}

sq.removeElementAt(*iter);           // remove it from the collection
```

メンバーを削除する代わりに、メンバーを別のコレクションに追加することもできます。replaceElementAt 関数や removeAllElement 関数を使用する場合も、その前に同じようなステップを実行する必要があります。

コレクションを破棄する場合は、その前にその中のメンバーを削除してください。この作業を実行するための関数を作成し、この関数をコレクションの apply 関数に渡すことができます。ここで、DKAttributeDef オブジェクトを含む DKAny オブジェクトのコレクションがあるとします。下記の例は、このコレクションを削除しているところです。

C++

```
DKDatastoreICM dsICM;
...
DKAny any = dsICM.listSchemaAttributes("GRANDPA");
dkCollection* acoll = (dkCollection*) any;
...
acoll->apply(deleteDKAttributeDef);           // use the attributes
delete acoll;                                // deletes all members
```

この例の中の deleteDKAttributeDef は、DKAny オブジェクトをパラメーターとする関数です。これは次のように定義されます。

C++

```
void deleteDKAttributeDef(DKAny& any) {  
    delete ((DKAttributeDef*) any.value());  
    any.setNull(); // good practice  
}
```

コレクションを削除したり、コレクションを削除する前にいくつかのメンバーを除去したりするために、独自の delete 関数を作成することもできます。

DKParts、DKFolder、および DKResults などの既知のコレクションのデストラクターは、これらの必要なクリーンアップのステップを実行するようになっています。しかし、replaceElementAt 関数、removeElementAt 関数、または removeAllElement 関数の実行時には、ストレージ管理は実行されません。

コレクションのソート

指定したキーに基づいて昇順または降順でコレクションのメンバーをソートするには、sort 関数を使います。ソート・オブジェクトと、希望する順序を渡す必要があります。ソート・オブジェクトのインターフェースは、dkSort.java (Java) または dkSort.hpp (C++) 内に定義されています。独自の sort 関数を作成して特定のコレクションをソートすることができます。以下の例は、DDO のコレクションをソートする方法を説明しています。

Java

```
DKResults rs;  
.... // Execute a query to fill DKResults with DDOs  
....  
DKSortDDOId sortId; // Declare the sort function object; sort on item-id  
rs.sort(sortId); // by default, sort in ascending order  
....
```

C++

```
DKResults* rs;  
....  
// Execute a query to fill DKResults with DDOs  
....  
DKSortDDOId* sortId; // Declare the sort function object; sort on item-id  
rs->sort(sortId); // by default, sort in ascending order  
....
```

ヒント: ソート・オブジェクトはスタックの中に作成されるので、それを明示的に削除する必要はありません。関数は再入可能です。つまり、単一のコピーを他の関数との間で共用、再利用および受け渡しできます。

統合コレクションおよび統合イテレーターについて

アプリケーション内の統合コレクションを使用すれば、照会からの結果であるデータ・オブジェクトをコレクションとして処理することができます。統合コレクションは、データ・オブジェクトの間に存在するサブグループの関係を保ちます。

統合コレクションは、DKResults オブジェクトのコレクションです。このコレクションは、DKFederatedQuery の結果を保持するために作成されます。この結果は、異種コンテンツ・サーバーからのもので構成される場合があります。各 DKResults には、特定のコンテンツ・サーバーからの検索結果が含まれます。統合コレクションには、ネスト・コレクションを無限に含めることができます。

統合コレクション内を網羅する処理を実行するには、dkIterator または DKSequentialIterator を作成してそれを使用します。その後、各 DKResults オブジェクト内を網羅するために別の dkIterator を作成し、元となったコンテンツ・サーバーに応じた処理を実行します。

また、統合イテレーター dkFederatedIterator を作成して、どのコンテンツ・サーバーに由来するものかに関係なくコレクションの全メンバーを網羅するために、それを使用するという方法もあります。

制約事項: 統合コレクションを照会することはできません。

図 8 に、DKFederatedCollection の構造および動作を示します。

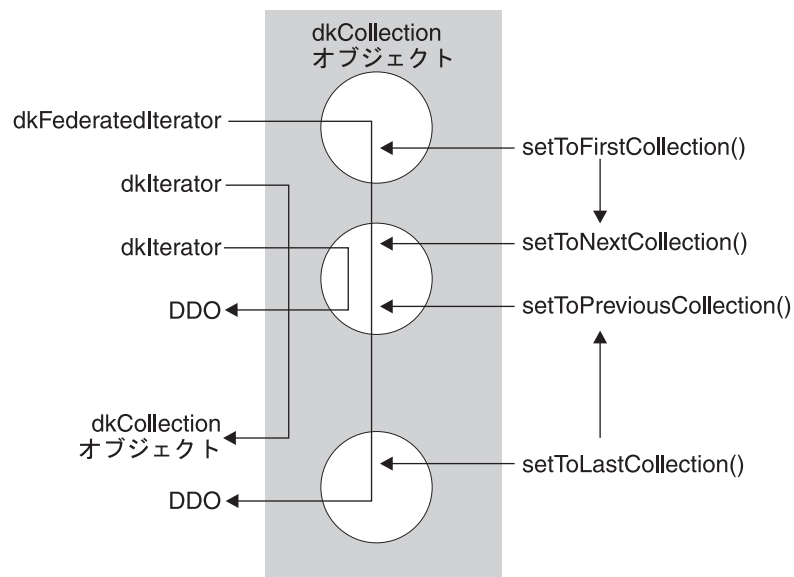


図 8. DKFederatedCollection の構造と動作

図 8 の長方形は DKFederatedCollection を表しており、この中により小さな円、DKResults オブジェクトが含まれています。dkFederatedIterator は、コレクションの境界を超えて処理を実行し、毎回 DDO を戻します。

最初の `dkIterator` は `DKFederatedCollection` のイテレーターで、毎回 `DKResults` オブジェクトを戻します。2 番目の `dkIterator` は 2 番目の `DKResults` オブジェクトのイテレーターで、これは `DKResults` コレクションの各メンバーの DDO を戻します。

`dkFederatedIterator` の `setToFirstCollection` 関数は、その位置を `DKFederatedCollection` の最初の DDO に設定します。この場合、これは最初の `DKResults` コレクション・オブジェクトの最初のエレメントとなります。この時点で `setToNextCollection` 関数が呼び出されると、このメソッドはイテレーターの位置を、2 番目の `DKResults` コレクションの最初の DDO に設定します。

`dkFederatedIterator` の `setToLastCollection` 関数は、そのイテレーターの位置を `DKFederatedCollection` の最後の DDO に設定します。この場合、これは最後の `DKResults` コレクション・オブジェクトの最後のエレメントです。
`setToPreviousCollection` 関数を呼び出すと、このメソッドはイテレーターの位置を、直前の `DKResults` コレクションの最後の DDO に設定します。

コンテンツ・サーバーの照会

コンテンツ・サーバー内を検索すると、`dkResultSetCursor` オブジェクトまたは `DKResults` オブジェクトとして結果を受け取ることができます。サーバーによっては、使用する照会を表す照会オブジェクトを作成し、その後照会オブジェクトの `execute` 関数または `evaluate` 関数を呼び出すことができます。コンテンツ・サーバーの機能により照会オブジェクトは、照会の準備や実行、照会の実行状況のモニター、および結果の保管などの照会処理作業を実行します。一部のコンテンツ・サーバーは、コンテンツ・サーバー・オブジェクトの代わりに照会オブジェクトの使用をサポートしています。旧バージョンの `Content Manager` および統合データ・ストアは、それに該当します。

コンテンツ・サーバーがサポートしている照会オブジェクトには、パラメトリック、テキスト、イメージ、および結合の 4 つのタイプがあります。結合照会は、テキスト照会とパラメトリック照会の両方を組み合わせたものです。すべてのコンテンツ・サーバーが結合照会を実行できるとは限りません。旧バージョンの `Content Manager` は、イメージ照会をサポートしています。

`Content Manager` のパラメトリック照会およびテキスト照会は統合されています。照会オブジェクトは使用しないでください。`Content Manager` における照会方法については、202 ページの『`Content Manager` サーバーの照会』を参照してください。旧バージョンの `Content Manager` サーバーの照会については、『その他のコンテンツ・サーバーの使用』を参照してください。

コンテンツ・サーバーは、照会を実行するために、`execute` と `evaluate` の 2 つのメソッドを使用します。`execute` 関数は `dkResultSetCursor` オブジェクトを戻し、`evaluate` 関数は `DKResults` オブジェクトを戻します。`dkResultSetCursor` オブジェクトは、大きな結果セットを処理するために使用されます。またこれは、結果セット・カーソルの現行位置で `delete` 関数および `update` 関数を実行します。
`fetchNextN` 関数を使用すると、オブジェクトのグループを検索してコレクションに入れることができます。

また、`dkResultSetCursor` を使用して、`close` メソッドおよび `open` メソッドを呼び出すことによって、照会を再実行することもできます。これは、133 ページの『結果セット・カーソルの使用』で説明しています。

`DKResults` には、照会の結果すべてが含まれています。コレクション内の項目の繰り返しは、順方向または逆方向のどちらでも実行できます。また、コレクションを照会したり、そのコレクションを別の照会の有効範囲として使用することもできます。

詳細については、134 ページの『照会を再実行するための結果セット・カーソルのオープンおよびクローズ』を参照してください。

制限事項: `Domino.Doc` コンテンツ・サーバーを照会すると、`DKResults` オブジェクトが戻されます。ただし、その `DKResults` オブジェクトに対して照会を実行したり、それを別の照会の有効範囲として使用することはできません。

dkResultSetCursor と DKResults との相違点

`dkResultSetCursor` コレクションと `DKResults` コレクションには、以下に示す違いがあります。

- `dkResultSetCursor` の動作は、コンテンツ・サーバーと似ています。これに含まれる `DKDDO` は一度に 1 つずつ取り出されるので、これは大きな結果セットで 사용할ことができます。これは、照会を再び実行して最新の結果を入手するのにも使用できます。

制限事項: `dkResultSetCursor` を使用する場合でも、`Domino.Doc` コンテンツ・サーバーで照会を再実行することはできません。

- `DKResults` は、結果セット全体が入り、両方向イテレーターをサポートします。
- `dkResultSetCursor` を長期間にわたってオープンしていると、コンテンツ・サーバーによっては同時ユーザーのパフォーマンスが低下します。

パラメトリック照会の使用

パラメトリック照会とは、照会述部に指定した条件と、コンテンツ・サーバー内に保管されているデータ値との厳密な一致を必要とする照会です。

注: 次に示す照会例は、旧バージョンの `Content Manager` に適用されます。`Content Manager V8` での照会については、201 ページの『照会言語について』を参照してください。

パラメトリック照会ストリングの定式化

照会を作成するには、まず照会ストリングを定式化します。以下の例では、旧バージョンの `Content Manager` における `GP2DLS2` という索引クラスに対する照会を表す照会ストリングを定義しています。`Content Manager` での照会ストリングの例については、210 ページの『照会の例』を参照してください。照会の条件は、属性 `DLSEARCH_DocType` が `null` ではない、すべての文書またはフォルダーの検索です。戻される結果の最大数が 5 に制限されており、`CONTENT` を `YES` に設定されているので、文書またはフォルダーの内容が戻されます。

Java

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS2," +
             "MAX_RESULTS=5," +
             "COND=(DLSEARCH_DocType > null));" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC);";
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,;" +
cmd += "MAX_RESULTS=5,;" +
cmd += "COND=(DLSEARCH_DocType <> NULL));";
cmd += "OPTION=(CONTENT=YES,;" +
cmd += "TYPE_QUERY=DYNAMIC,;" +
cmd += "TYPE_FILTER=FOLDERDOC);";
```

この例では、旧バージョンの Content Manager サーバーがこの照会で動的 SQL を使用し、すべてのフォルダーおよび文書を検索するよう指定しています。コンテンツ・サーバーが異なれば、それが使用する照会ストリングの構文も異なります。統合コンテンツ・サーバーは独自の照会ストリング構文を使用します。詳細については、検索するコンテンツ・サーバーに関する情報、または「オンライン API 解説書」を参照してください。属性名が複数の単語であるか DBCS 言語が使用されている場合は、その属性名をアポストロフィ (') で囲む必要があります。属性値が DBCS の場合、二重引用符 (") で囲む必要があります。

複数の基準によるパラメトリック照会の定式化

パラメトリック照会についての検索基準は複数指定できます。以下の例は、旧バージョンの Content Manager の 2 つの索引クラスに対する照会を指定する方法を示しています。

Java

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS1,MAX_RESULTS=3," +
             "COND=(DLSEARCH_DocType <> null);" +
             "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8," +
             "COND=('First name'!=¥Robert¥));" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC);";
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,MAX_RESULTS=3,";
cmd += "COND=(DLSEARCH_DocType <> NULL);";
cmd += "INDEX_CLASS=DLSAMPLE,MAX_RESULTS=8,";
cmd += "COND=('First name' == ¥\"Robert¥\"));";
cmd += "OPTION=(CONTENT=YES;";
cmd += "TYPE_QUERY=DYNAMIC;";
cmd += "TYPE_FILTER=FOLDERDOC)";
```

パラメトリック照会の実行

照会ストリングを作成したら、次は照会オブジェクトを作成します。コンテンツ・サーバーを表す `DKDatastorexx` には、照会オブジェクトを作成するためのメソッドが含まれています。照会オブジェクトを使用して、照会を実行してその結果を取得します。次の例は、パラメトリック照会オブジェクトを作成し、旧バージョンの Content Manager サーバーで照会を実行する方法を示しています。Content Manager バージョン 8 以降では照会オブジェクトは使用しないでください。照会が実行されると、結果は `DKResults` コレクションに戻されます。

重要: `DKResults` オブジェクトを削除すると、そのメンバーもすべて削除されます。エレメントを二度削除しないようにしてください。

Java

```
// ----- Create the datastore, the query object, and the results set
DKDatastoreDL dsDL = new DKDatastoreDL();
dkQuery pQry = null;
DKResults pResults = null;
DKNameValuePair parms[] = null;
// ----- Connect to the datastore
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
              "MAX_RESULTS=5," +
              "COND=(DLSEARCH_DocType <> NULL));" +
              "OPTION=(CONTENT=YES;" +
              "TYPE_QUERY=STATIC;" +
              "TYPE_FILTER=FOLDERDOC)";
// ----- Create the query using the query string
pQry = dsDL.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Execute the query
pQry.execute(parms);
// ----- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Disconnect when you are through
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (`TSamplePQryDL.java`) が `CMBROOT¥Samples¥java¥dl` ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreDL dsDL;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected libsrv: " <<libsrv<< " userid: "<<userid<<endl;

DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE, ";
cmd += "MAX_RESULTS=5, ";
cmd += "COND=(DLSEARCH_DocType <> NULL)); ";
cmd += "OPTION=(CONTENT=YES); ";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsDL.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsDL.disconnect();
```

完全なサンプル・アプリケーション (TSamplePQryDL.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

コンテンツ・サーバーからのパラメトリック照会の実行

コンテンツ・サーバーを表す DKDatastorexx には、照会を実行するためのメソッド
が含まれています。次の例は、旧バージョンの Content Manager コンテンツ・サー
バーでパラメトリック照会を実行する方法を示しています。照会が実行されると、
結果は dkResultSetCursor オブジェクトに戻されます。

Java

```
// ----- Create the datastore and cursor
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;
// ----- Connect to the content server
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "MAX_RESULTS=5," +
             "COND=((DLSEARCH_DocType <> NULL)" +
             "AND (DLSEARCH_Date >= 1995)))"; +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";

...
// ----- Execute the query using the query string
pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Process query results as you want
...
// ----- When finished with the cursor, delete it, and disconnect
pCur.destroy();
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TExecuteDL.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
...
DKDatastoreDL dsDL;
dkResultSetCursor* pCur = 0;
cout << "Datastore DL created" << endl;
cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
// DKString cmd = "SEARCH=(COND=('DLSEARCH_DocType' == ¥"html¥));";
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,";
cmd += "MAX_RESULTS=5,";
cmd += "COND=(DLSEARCH_DocType <> NULL));";
cmd += "OPTION=(CONTENT=YES);";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC";
cout << "query string " << cmd << endl;
cout << "executing query" << endl;
pCur = dsDL.execute(cmd);
cout << "query executed" << endl;
...
...
if (pCur != 0)
delete pCur;
dsDL.disconnect();
...
```

完全なサンプル・アプリケーション (TExecuteDL.cpp) が

Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから取られています。

コンテンツ・サーバーからのパラメトリック照会の評価

コンテンツ・サーバーを表す DKDatastorexx には、照会を評価するためのメソッドが含まれています。結果は、DKResults コレクションに戻されます。次の例は、旧バージョンの Content Manager コンテンツ・サーバーでパラメトリック照会を評価する方法を示しています。

Java

```
// ----- Create the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
              "COND=((DLSEARCH_Date >= ¥"1995¥") AND " +
              "      (DLSEARCH_Date <= ¥"1996¥")));" +
              "OPTION=(CONTENT=NO;" +
              "      TYPE_QUERY=DYNAMIC;" +
              "      TYPE_FILTER=FOLDERDOC)";

DKNVPair parms[] = null;
DKDDO item = null;
// ----- Create the datastore and connect
// replace following with your library server, user ID,
// password DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Call evaluate, get the results, and create an
// iterator to process them
DKResults pResults =
    (DKResults)dsDL.evaluate(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
    item = (DKDDO)pIter.next();
    ... // ----- Process the DKDDO as appropriate
}
dsDL.disconnect();
dsDL.destroy();
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,";
cmd += "COND=((DLSEARCH_Date >= ¥"1995¥") AND ";
cmd += "      (DLSEARCH_Date <= ¥"1996¥")));" +
        "OPTION=(CONTENT=NO;" +
        "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)";

...
DKAny any = dsDL.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*)element->value();
    // Process the DKDDO
}
delete pIter;
delete pResults;
dsDL.disconnect();
```

テキスト照会の使用

Content Manager バージョン 8 以降では、テキストおよびパラメトリック照会が統合されています。207 ページの『パラメトリック検索とテキスト検索の組み合わせ』を参照してください。

旧バージョンの Content Manager では、テキストおよびパラメトリック検索を実行できます。テキスト検索は、テキスト検索エンジンによって作成されたテキスト索引を照会し、実際の文書テキストを検索します。

テキスト照会ストリングの定式化

テキスト検索は、照会ストリングを定式化することから始めます。以下の例では、TMINDEX テキスト索引に対する照会を表す照会ストリングを作成します。照会ストリングには、UNIX® または member という語を使ってすべてのテキスト文書を検索するための基準が含まれています。戻される結果の最大数は 5 です。

Java

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +  
            "OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5)";
```

C++

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
```

複数の索引に対するテキスト照会の定式化

複数の索引を検索する場合は、テキスト照会を使います。以下の例は、2 つの索引に対する照会を指定する方法を示しています。

重要: 照会で複数のテキスト検索索引を指定する場合、索引は同じタイプでなければなりません。例えば、照会で 2 つの精密索引を指定することはできますが、精密索引と言語索引とを指定することはできません。

Java

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";  
cmd += "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

C++

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +  
            "OPTION=(SEARCH_INDEX=(TMINDEX, INDEX2); MAX_RESULTS=5)";
```

テキスト照会の実行

テキスト照会ストリングを作成したら、次は照会オブジェクトを作成します。コンテンツ・サーバーを表す DKDatastorexx には、照会オブジェクトを作成するためのメソッドが含まれています。結果は、DKResults コレクションに戻されます。照会オブジェクトを使用して、照会を実行してその結果を取得します。以下の例は、テキスト照会オブジェクトを作成し、照会を実行する方法を示しています。

Java

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkQuery pQry = null;
DKResults pResults = null;
DKNVPair parms[] = null;
// ----- Connect to the datastore
//         for example, dsTS.connect("zebra","7502",DK_CTYP_TCPIP);
dsTS.connect(srchSrv,"","");
// ----- Formulate the query string
String cmd = "SEARCH=(COND=(member AND UNIX));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
// ----- Create and execute the query
pQry = dsTS.createQuery(cmd, DK_CM_TEXT_QL_TYPE, parms);
pQry.execute(parms);
// ----- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- When finished, disconnect
dsTS.disconnect();
dsTS.destroy();
```

完全なサンプル・アプリケーション (TSampleTQryTS.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
//dsTS.connect("zebra","7502",DK_CTYP_TCPIP);
dsTS.connect(srchSrv,"","");
cout << "connected to datastore srchSrv: " << srchSrv << endl;

DKString cmd = "SEARCH=";
cmd += "(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=";
cmd += srchIndex;
cmd += ")";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsTS.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsTS.disconnect();
```

完全なサンプル・アプリケーション (TSampleTQryTS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

コンテンツ・サーバーからのテキスト照会の実行

コンテンツ・サーバーを表すのに使用される DKDatastorexx には、照会を実行するためのメソッドがあります。結果は、dkResultSetCursor オブジェクトに戻されます。次の例は、旧バージョンの Content Manager コンテンツ・サーバーに対して、テキスト照会を実行する方法を示したものです。

Java

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the datastore
//         for example, dsTS.connect
("zebra","7502",DK_TS_CTYP_TCPIP);
dsTS.connect(srchSrv,"","");

// ----- Formulate the query string
String cmd = "SEARCH=(COND=(internet OR UNIX));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;" +
             "MAX_RESULTS=5)";

...
// ----- Execute the query and process the results
as appropriate
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);
...
// ----- When finished, delete the cursor and disconnect
pCur.destroy();
dsTS.disconnect();
dsTS.destroy();
```

完全なサンプル・アプリケーション (TExecuteTS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
while (pCur->isValid()) {
    item = pCur->fetchNext();
    if (item != 0) {
        // Process the DKDDO
        ...
        delete item;
    }
}
delete pCur;
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TExecuteTS.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

コンテンツ・サーバーからのテキスト照会の評価

コンテンツ・サーバーを表すのに使用する DKDatastorexx には、照会を実行して DKResults コレクションを戻すための evaluate メソッドがあります。次の例は、旧バージョンの Content Manager コンテンツ・サーバーに対するテキスト照会を評価する方法を示したものです。

Java

```
// ----- Create the datastore and the query string
DKDatastoreTS dsTS = new DKDatastoreTS();
String cmd = "SEARCH=(COND=($MC=$ UN*));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
// ----- Connect to the datastore
dsTS.connect("TM","", ' ');
...
// ----- Call evaluate, get the results, and process
as appropriate
DKResults pResults = (DKResults)dsTS.evaluate(cmd,DK_CM_TEXT_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
    item = (DKDDO)pIter.next();
    // ----- Process the individual DKDDO objects
}
// ----- Disconnect
dsTS.disconnect();
dsTs.destroy();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";

...
DKAny any = dsTS.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*) element->value();
    // Process the DKDDO
    ...
}
delete pIter;
delete pResults;
dsTS.disconnect();
```

一致ハイライト情報の取得

一致情報には、該当する照会でそれぞれ一致する、文書のテキストおよびハイライト情報が含まれます。

照会ストリングを定式化する場合、MATCH_INFO および MATCH_DICT オプションを設定します。MATCH_INFO を YES に設定すると、一致ハイライト情報が戻されます。MATCH_DICT オプションは、辞書を使用してハイライト情報を取得するかどうかを指定します。一致情報は、テキスト照会で戻される DKDDO の DKMATCHESINFO 属性に戻されます。DKMATCHESINFO 属性の値は、DKMatchesInfoTS オブジェクトになります。

一致ハイライト情報の取得は、コンテンツ・サーバーから文書を取得して言語的に分析し、潜在的な一致を判別するので、時間がかかります。このプロセスを実行すると、テキスト照会のパフォーマンスに影響を与えます。

各テキスト照会の結果項目の一致ハイライト情報の取得: 以下の例では、テキスト照会時に各テキスト照会の結果項目の一致ハイライト情報を取得します。MATCH_DICT オプションは NO に設定されているので、辞書は使用されません。

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the content server
//      replace following with your library server,
user ID, password
dsTS.connect("TM","", "", "LIBACCESS=(LIBSRVRN,
FRNADMIN, PASSWORD)");
// ----- Formulate the query string
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
              "OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5; +
              "MATCH_INFO=YES; MATCH_DICT=NO)";

...

pCur = dsTS.execute(cmd, DK_CM_TEXT_QL_TYPE, parms);
DKDDO item = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc = "";
String strSection = "";
String strText = "";
Object anyObj = null;
while (pCur.isValid())
{
    // ----- Get the next DKDDO
    item = pCur.fetchNext();
    if (item != null)
    {
        // continued...
    }
}
```

Java (続き)

```
// ----- Process the DKDDO
for (i = 1; i <= item.dataCount(); i++)
{
    anyObj = item.getData(i);
    if (anyObj instanceof String)
    {
        ...
    }
    else if (anyObj instanceof Integer)
    {
        ...
    }
    else if (anyObj instanceof Short)
    {
        ...
    }
    else if (anyObj instanceof DKMatchesInfoTS)
    {
        pMInfo = (DKMatchesInfoTS)anyObj;
        // ----- process the Match Hightlighting information
        if (pMInfo != null)
        {
            strDoc = pMInfo.getDocumentName();
            numberSections = pMInfo.numberOfSections();
            // ----- loop thru document sections
            for (j = 1; j <= numberSections; j++)
            {
                pMSect = pMInfo.getSection(j);
                strSection = pMSect.getSectionName();
                numberParagraphs = pMSect.numberOfParagraphs();
                // ----- loop thru section paragraphs
                for (k = 1; k <= numberParagraphs; k++)
                {
                    pMPara = pMSect.getParagraph(k);
                    lCCSID = pMPara.getCCSID();
                    lLang = pMPara.getLanguageId();
                    numberTextItems = pMPara.numberOfTextItems();
                    // ----- loop thru paragraph text items
                    for (m = 1; m <= numberTextItems; m++)
                    {
                        pMText = pMPara.getTextItem(m);
                        strText = pMText.getText();
                        // ----- if match found in text item get offset
                        //          and length of match in text item
                        if (pMText.isMatch() == true)
                        {
                            lOffset = pMText.getOffset();
                            lLen = pMText.getLength();
                        }
                        numberNewLines = pMText.numberOfNewLines();
                    }
                }
            }
        }
    }
}
dsTS.disconnect();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5;
MATCH_INFO=YES;MATCH_DICT=NO)";

...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        // Process the DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            switch (anyObj.typeCode())
            {
                case DKAny::tc_string :
                {
                    ...
                    break;
                }
                case DKAny::tc_long :
                {
                    ...
                    break;
                }
                case DKAny::tc_short :
                {
                    ...
                    break;
                }
                case DKAny::tc_dobase :
                {
                    ...
                }
            }
        }
    }
}
// continued...
```

C++ (続き)

```
// process the Match Hightlighting information
pD0Base = a;
pMInfo = (DKMatchesInfoTS*)pD0Base;
if (pMInfo != 0)
{
    strDoc = pMInfo->getDocumentName();
    numberSections = pMInfo->numberOfSections();
    // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo->getSection(j);
        strSection = pMSect->getSectionName();
        numberParagraphs = pMSect->numberOfParagraphs();
        // loop thru section paragraphs
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect->getParagraph(k);
            lCCSID = pMPara->getCCSID();
            lLang = pMPara->getLanguageId();
            numberTextItems = pMPara->numberOfTextItems();
            // loop thru paragraph text items
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara->getTextItem(m);
                strText = pMText->getText();
                // if match found in text item get offset and
                // length of match in text item
                if (pMText->isMatch() == TRUE)
                {
                    lOffset = pMText->getOffset();
                    lLen = pMText->getLength();
                }
                numberNewLines = pMText->numberOfNewLines();
            }
        }
    }
    break;
default :
{
    break;
}
...
delete item;
}
}
delete pCur;
dsTS.disconnect();
```

特定のテキスト照会の結果項目の一致ハイライト情報の取得: 以下の例では、テキスト照会で戻される特定の項目の一致ハイライト情報を取得します。このルーチンに渡される `dkResultSetCursor` は、オープン状態である必要があります。

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";

...
pCur = dsTS.execute(cmd);
DKDDO item = null;
Object anyObj = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc;
String strSection;
String strText;
String strDID = "";
String strXNAME = "";
String strDataName = "";
DKPid pid = null;
while (pCur.isValid())
{
    item = pCur.fetchNext();
    if (item != null)
    {
        pid = item.getPid();
        // Process the DKDDO
        for (i = 1; i <= item.dataCount(); i++)
        {
            anyObj = item.getData(i);
            strDataName = item.getDataName(i);
            if (strDID.equals(""))
            {
                strDID = pid.getId();
            }
            if (strXNAME.equals(""))
            {
                strXNAME = p.getObjectType();
            }
            ...
        }
    }
}
// continued...
```

Java (続き)

```
// Get Match Highlighting Information
pMInfo = dsTS.getMatches(pCur, strDID, strXNAME, false);
strDID = "";
strXNAME = "";
if (pMInfo != null)
{
    strDoc = pMInfo.getDocumentName();
    numberSections = pMInfo.numberOfSections();
    // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo.getSection(j);
        strSection = pMSect.getSectionName();
        numberParagraphs = pMSect.numberOfParagraphs();
        // loop thru section paragraphs
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect.getParagraph(k);
            lCCSID = pMPara.getCCSID();
            lLang = pMPara.getLanguageId();
            numberTextItems = pMPara.numberOfTextItems();
            // loop thru paragraph text items
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara.getTextItem(m);
                strText = pMText.getText();
                // if match found in text item get offset and
                // length of match in text item
                if (pMText.isMatch() == true)
                {
                    lOffset = pMText.getOffset();
                    lLen = pMText.getLength();
                }
                numberNewLines = pMText.numberOfNewLines();
            }
        }
    }
}
dsTS.disconnect();
dsTS.destroy();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
DKString strDID;
DKString strXNAME;
DKString strDataName;
DKPid pid;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        pid = item->getPid();
        // Process the DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            strDataName = item->getDataName(i);
            if (strDataName == "")
            {
                strDID = pid.getId();
            }
            if (strXNAME == "")
            {
                strXNAME = p->getObjectType();
            }
            switch (anyObj.typeCode())
            {
                ...
            }
        }
    }
}
// continued...
```


C++ (続き)

```
// Get Match Highlighting Information
pMInfo = dsTS.getMatches(pCur, strDID, strXNAME, FALSE);
strDID = "";
strXNAME = "";
if (pMInfo != 0)
{
    strDoc = pMInfo->getDocumentName();
    numberSections = pMInfo->numberOfSections();
    // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo->getSection(j);
        strSection = pMSect->getSectionName();
        numberParagraphs = pMSect->numberOfParagraphs();
        // loop thru section paragraphs
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect->getParagraph(k);
            lCCSID = pMPara->getCCSID();
            lLang = pMPara->getLanguageId();
            numberTextItems = pMPara->numberOfTextItems();
            // loop thru paragraph text items
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara->getTextItem(m);
                strText = pMText->getText();
                // if match found in text item get offset and
                // length of match in text item
                if (pMText->isMatch() == TRUE)
                {
                    lOffset = pMText->getOffset();
                    lLen = pMText->getLength();
                }
                numberNewLines = pMText->numberOfNewLines();
            }
        }
    }
    delete pMInfo;
}
...
delete item;
}
}
delete pCur;
dsTS.disconnect();
```

結果セット・カーソルの使用

dkResultSetCursor は、DDO オブジェクトの仮想コレクションを管理するコンテンツ・サーバー・カーソルです。これは、コレクションはエレメントが取り出されるまで実体化されないということを意味します。コレクションは、照会の基準に一致する項目の結果セットです。このカーソルの使用後には、destroy メソッドを呼び出すことによって、使用したメモリーを解放してください。

重要: ここで説明する情報は、Content Manager 8.2 には適用されません。詳細については、201 ページの『照会言語について』を参照してください。

照会を再実行するための結果セット・カーソルのオープンおよびクローズ

結果セット・カーソルは、作成時にはオープン状態です。照会を再実行するには、カーソルをクローズしてから再オープンします。

Java

```
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
...

dkResultSetCursor pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);

pCur.close();
pCur.open();           //re-execute the query
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE)";
cmd += "OPTION=(CONTENT=YES;";
cmd += "TYPE_QUERY=DYNAMIC;";
cmd += "TYPE_FILTER=FOLDERDOC)";
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
// re-execute the query
pCur->close();
pCur->open();
```

結果セット・カーソルの位置の設定および取得

結果セット・カーソルを使用して、カーソル位置の設定および取得を実行できます。以下の例では、照会を作成し、実行します。while ループ内で、カーソル位置は最初の（または次の）有効位置に設定されています。その後、DDO がその位置から取り出されます。カーソルが最後の項目に渡される場合は、fetchObject メソッドからヌルが戻されます。

Java

```
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES);" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC>";

DKNVPair parms[] = null;
DKDDO item = null;
int i = 0;
...
// ----- Execute the query; the result cursor is returned
dkResultSetCursor pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
// ----- Use a while loop to iterate thru the collection
while (pCur.isValid())
{
    pCur.setToNext();
    item = pCur.fetchObject();
    if (item != null)
    {
        i = pCur.getPosition();
    }
}
}
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE)";
cmd += "OPTION=(CONTENT=YES)";
cmd += "TYPE_QUERY=DYNAMIC";
cmd += "TYPE_FILTER=FOLDERDOC>";
pCur = 0;
DKDDO *item = 0;
long i = 0;
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setToNext();
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

以下に示すのは、別の実行方法です。

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_NEXT,a);
    item = pCur.fetchObject();
    if (item != null) {
        i = pCur.getPosition();
    }
}
```

C++

```
DKAny a;
pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setPosition(DK_CM_NEXT,a);
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

項目を繰り返す場合、相対位置を使用することができます。以下の例では、結果セット・カーソル内の他のすべての項目がスキップされます。

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
a = new Integer(2);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_RELATIVE,a); // move cursor 2 positions forward
    item = pCur.fetchObject();          // from the current position
    if (item != null) {                  // (relative)
        i = pCur.getPosition();
    }
}
```

C++

```
DKAny a;
long increment = 2;
pCur = dsDL.execute(cmd);
a = increment;
while (pCur.isValid()) {
    pCur->setPosition(DK_CM_RELATIVE,a);
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

結果セット・カーソルからのコレクションの作成

結果セット・カーソルを使うと、指定した数の項目を結果セット・カーソルからコレクションに取り込むことができます。 `fetchNextN` メソッドの最初のパラメーターには、コレクションに入れる項目数を指定します。最初のパラメーターに 0 を渡すと、すべての項目がコレクションに入れられることになります。

以下の例では、結果セット・カーソルからのすべての項目が、順次コレクションに取り込まれます。 `fItems` が `TRUE` なら、1 つ以上の項目が戻されたということです。

Java

```
DKSequentialCollection seqColl = new DKSequentialCollection();
boolean fItems = false;
int how_many = 0;
fItems = pCur.fetchNextN(how_many,seqColl);
```

C++

```
DKSequentialCollection seqColl;
DKBoolean fItems = FALSE;
long how_many = 0;
fItems = pCur->fetchNextN(how_many,seqColl);
```

コレクションの照会

照会可能コレクション とは、さらに照会を行うことのできるコレクションであり、これによって範囲を限定してさらに精密な結果を得ることができます。照会可能コレクションを具体化したインプリメンテーションが `DKResults` オブジェクトであり、照会評価の結果としてそれが戻されます。 `DKResults` は `dkQueryableCollection` のサブクラスであり、`DDO` のコレクションです。

照会の結果の取得

以下の例は、パラメトリック照会を実行依頼して結果を取得する方法を示しています。結果は DKResults オブジェクトである rs です。前に挙げたコード例を使用して、コレクションを処理し、DDO を取得することができます。

Java

```
// ----- Create a datastore and establish a connection
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Create and execute a query object
String query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null));";
DKParametricQuery pq =
    (DKParametricQuery) dsDL.createQuery(query1,DK_CM_PARAMETRIC_QL_TYPE, null);
pq.execute();
// ----- Get the result
DKResult rs = (DKResults) pq.result();
```

C++

```
// establish a connection
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// create a query object
DKString query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL));";
DKParametricQuery* pq = (DKParametricQuery*)
    dsDL.createQuery(query1,DK_PARAMETRIC_QL_TYPE, NULL);
pq->execute();
DKAny any = pq->result();
DKResult* rs = (DKResults*) any.value();
```

新規照会の評価

照会による結果を照会して、さらに精密なものにすることができます。以下のコードは前の例に基づいており、照会の再評価を示しています。

Java

```
String query2 = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(Subject == 'Mystery'))";
Object obj = rs.evaluate(query2,DK_CM_PARAMETRIC_QL_TYPE, null);
....
```

C++

```
DKString query2="SEARCH=(INDEX_CLASS=GRANDPA, COND=(Subject=='Mystery'))";
any = rs->evaluate(query2,DK_PARAMETRIC_QL_TYPE, NULL);
...
```

2 番目の照会で戻される obj は、絞り込まれた結果を含む DKResults オブジェクトです。両方の照会の結合結果は、以下の場合と同じになります。

```
"SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL AND Subject == 'Mystery'))";
```

満足のゆく結果が得られるまで、照会を繰り返すことができます。あるタイプの照会を開始したら、それに続く照会も同じタイプにする必要があります。異なる照会タイプを使用すると、結果が戻されない場合があります。

以下の例は、順次テキスト照会を示しています。

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM","","","");

// ----- The first query
String tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery1, DK_CM_TEXT_QL_TYPE, null);
tq.execute();
DKResults trs = (DKResults) tq.result();
// ----- The second query
String tquery2 = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
Object obj = trs.evaluate(tquery2, DK_CM_TEXT_QL_TYPE, null);
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","");

DKString tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
    (DKTextQuery*) dsTS.createQuery(tquery1,DK_TEXT_QL_TYPE, NULL);
tq->execute();
any = tq->result();
DKResults* trs = (DKResults*) any.value();

DKString tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
any = trs->evaluate(tquery2,DK_TEXT_QL_TYPE, NULL);
```

2 番目の照会で戻される obj は、絞り込まれた結果を含む DKResults オブジェクトです。両方の照会の結合結果は、以下の場合と同じになります。

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

結合照会に代わる照会可能コレクションの使用

結合照会は、パラメトリック照会とテキスト照会を組み合わせる実行依頼を行える柔軟性を備えています。有効範囲を指定することも、しないことも可能です。ただし、これらすべての照会は、一度に実行依頼しなければなりません。照会可能コレクションを評価する場合のように、一度に 1 つを実行するものではありません。

結合照会は DKResults オブジェクトを戻します。しかし、それに対する別のパラメトリック照会を評価することはできません。すべてのコンテンツ・サーバーで結合照会を使用できるとは限りません。

照会可能コレクションと後続の照会を一緒に評価することにより、直前の照会結果を段階的に絞り込む柔軟性が与えられ、その結果、満足のいく最終結果を得ることができます。後続の照会は、動的にコンテンツ・サーバーをブラウズする場合や、それまでの結果に基づいて次の照会を定式化する場合に便利です。ただし、前もって照会の組み合わせがわかっている場合は、完全な照会を一度に実行依頼するか結合照会を使用するほうが効率的です。

Content Manager バージョン 8.2 の使用

このセクションでは、Content Manager バージョン 8 リリース 2 コネクター (ICM コネクター) のアプリケーション・プログラミング・インターフェース (API) について説明します。ICM コネクターは、Enterprise Information Portal (EIP) フレームワークの拡張機能であるため、この内容について先に進むには、13 ページの

『Enterprise Information Portal アプリケーション・プログラミングの概念』で説明されている EIP フレームワークの概念について理解する必要があります。

ICM コネクター API を使用することにより、Content Manager コンテンツ・サーバーにアクセスするカスタム・アプリケーションを作成して配置することができます。また、API を使用することにより、既存のアプリケーションを Content Manager コンテンツ・サーバーに組み込むこともできます。

このセクションの内容は、以下のとおりです。

- Content Manager システムの概要
- Content Manager の概念
- Content Manager アプリケーションの計画
- Content Manager アプリケーションの作成
- 情報へのアクセスの制御
- トランザクションの処理
- 項目の検索

Content Manager システムの概要

Content Manager システムの主なコンポーネントには、ライブラリー・サーバー、1 つ以上の リソース・マネージャー、および一連のオブジェクト指向アプリケーション・プログラミング・インターフェース (API) があります。Content Manager システムの管理には、Java ベースのシステム管理クライアントも用意されています。

ライブラリー・サーバーには、柔軟性のあるデータ・モデル機能、システムへのセキュア・アクセス、効率的なコンテンツ管理、およびその他のフィーチャーが提供されています。ライブラリー・サーバーは、システム内の項目間の関係を管理し、システム内に構成されているリソース・マネージャーに保管された情報をはじめ、すべてのシステム情報へのアクセスを制御します。

リソース・マネージャーは、スキャン・イメージ、オフィス文書、またはビデオなどのバイナリー・オブジェクトの実際のコンテンツを保管するコンポーネントです。Content Manager VideoCharger やその他の IBM 以外の製品の他のリソース・マネージャーを、Content Manager システムに組み込むことができます。リソース・マネージャーを使用することにより、以下のタスクを行うことができます。

- System Managed Storage (SMS) を使用して、コストのかかる高速メディアから、低速で低コストのメディアにコンテンツを自動的に移動する。
- Web ブラウザーからのリソース・マネージャーへの直接アクセス。

- オブジェクトのすべてまたは一部の検索。
- ライブラリー・サーバーとのデータの同期化。

API により、アプリケーションは Content Manager システムにアクセスできます。Java および C++ 用の API が提供されています。API を使用することにより、アプリケーションは、データ・モデリング、統合パラメトリック検索および統合テキスト検索、サード・パーティーのデータ・アクセスおよびデリバリーなど、すべての Content Manager 機能を利用することができます。

図 9 のダイアグラムは、システム・コンポーネントの関係を示しています。これは、Content Manager システムのインプリメンテーションの 1 つの例に過ぎません。例えば、他のシステム構成では、4 つのリソース・マネージャーが存在するかもしれません。

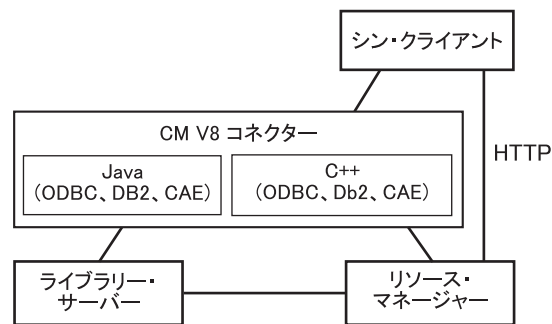


図 9. システム構成

Content Manager の概念

このセクションでは、Content Manager の重要な概念を説明します。プログラミング・タスクに進むには、Content Manager の概念の理解が不可欠です。このセクションで説明する内容は、以下のとおりです。

- 項目。
- 属性。
- 項目タイプ。
- ルート・コンポーネントと子コンポーネント。
- オブジェクト。
- リンクおよび参照。
- 文書。
- フォルダー。
- バージョン管理。
- アクセス制御。
- 文書管理データ・モデル。

項目

項目は、ライブラリー・サーバーが管理する基本エンティティです。項目の例として、保険証券 (policy)、保険料請求 (claim)、電話番号 (phone number) などがあります。項目 は、項目タイプのインスタンスを表す一般的な用語です。オブジェクトが離散的なデジタル・コンテンツである場合、項目はそのオブジェクトの表現となります。項目はオブジェクトではありませんが、項目によりオブジェクトとその検出方法が完全に識別されます。システム内で、項目は、文書やフォルダーなどのオブジェクトを表します。文書などのビジネス・オブジェクトを定義するには、項目定義を操作します。

アプリケーションが項目を作成すると、Content Manager は項目に対していくつかのシステム定義属性を割り当て、ユーザーが独自の属性を定義できるようにします。

システム定義の属性には、作成時のタイム・スタンプ、および項目 ID (item ID) があります。項目 ID は、項目ごとに固有なものとなります。itemID は Content Manager によって保管され、ライブラリー・サーバー内の項目を見つけるために使用されます。アプリケーションを作成する場合、itemID を使用することにより、その項目に関連したすべてのデータにアクセスすることができます。

属性

属性 は、項目の特定の特性またはプロパティ (例: 名前、住所、年齢など) を記述したデータ単位であり、これを基準とした検索によって項目を見付けることができます。

属性をグループ化することにより、属性グループの作成が可能です。例えば住所属性は、郵便番号、都道府県、市区町村、番地を含む属性から構成することができます。

また、複数の値を持つ属性も定義することができます。こうした属性は多値属性と呼ばれ、子コンポーネントとしてインプリメントされます。例えば、証券所有者の自宅や勤務先など、複数の住所を保管することができます。

詳細については、SAttributeDefinitionCreationICM サンプルを参照してください。

項目タイプ

項目タイプ (旧バージョンの Content Manager では索引クラス) は、基本的に、類似した項目を定義して、後から見つけることができるようにするためのテンプレートです。項目タイプは、ルート・コンポーネント、子コンポーネント (ない場合もある) および種別から構成されます。項目タイプは、すべてのコンポーネントおよび関連データが含まれる総合的な構造です。例えば、保険シナリオでは、「保険証券 (policy)」項目タイプには保険証券番号、名前、保険料請求などの属性を持つ項目が含まれます。

Content Manager では、項目タイプは、非リソース、リソース、文書 (文書モデルとも呼ばれる)、および文書パーツの 4 つのタイプに分類されます。非リソース項目タイプは、リソース・マネージャーに保管されないエンティティを表します。リソース項目タイプは、リソース・マネージャーに保管されたオブジェクト (ファイル・システムのファイル、ビデオ・サーバーのビデオ・クリップ、データベース・

テーブルの LOB (ラージ・オブジェクト) など) を表します。文書項目タイプは、単一のリソース項目タイプのように、リソース・コンテンツを含んでいる複数の文書パーツが入ったエンティティを表します。文書パーツ項目タイプは、リソース・マネージャーに保管されているオブジェクトを表しますが、1 つの文書を構成する複数のパーツであり、1 つの文書項目タイプによって格納され、所有されます。Content Manager には、リソース項目タイプの基本セット (LOB、テキスト、イメージ、ストリーム、およびビデオ・オブジェクト) が用意されています。

詳細については、SItemTypeCreationICM サンプルを参照してください。

ルート・コンポーネントと子コンポーネント

項目タイプは、コンポーネント (ルート・コンポーネントと任意の数の子コンポーネント (オプション)) で構成されます。

ルート・コンポーネント は、階層項目タイプの第 1 レベルまたは唯一のレベルです。項目タイプは、システム定義属性およびユーザー定義属性の両方から構成されます。内部的には、最も基本的な項目タイプには 1 つのコンポーネントのみが含まれます。

子コンポーネント は、階層項目タイプのオプションの 2 番目以下のレベルです。子コンポーネントはそれぞれ、その上のレベルと直接関連付けられています。図 10 に、Content Manager メタモデルのダイアグラムを示します。このダイアグラムは、項目の階層を形成するときのルート・コンポーネントおよび子コンポーネントとそれらの関係を示しています。また、リンク、参照、リソース項目、およびリソース・オブジェクトも示します。

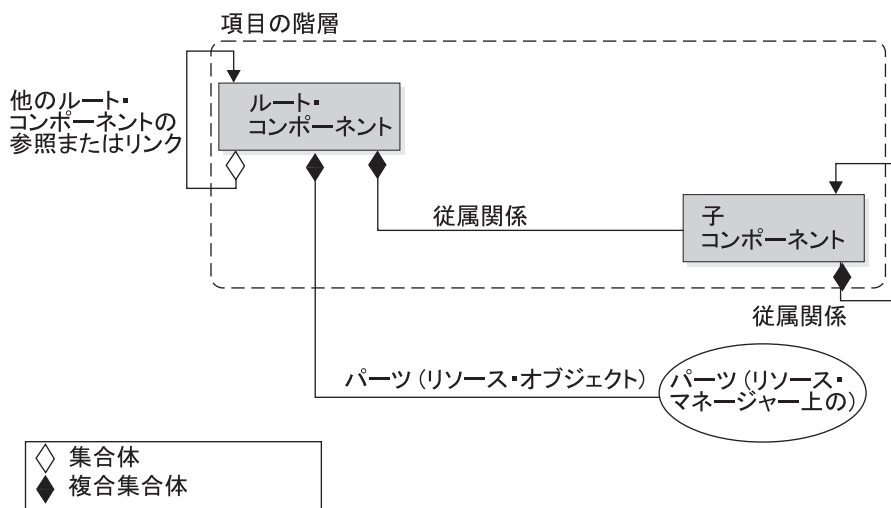


図 10. Content Manager メタモデル: 論理図。

145 ページの図 11 に、ルート・コンポーネントとして「保険証券 (policy)」、子コンポーネントとして「保険料請求 (claim)」、「警察レポート (police report)」、および「損害見積 (damage estimate)」を持つ保険申込書のデータ・モデルの例を示します。

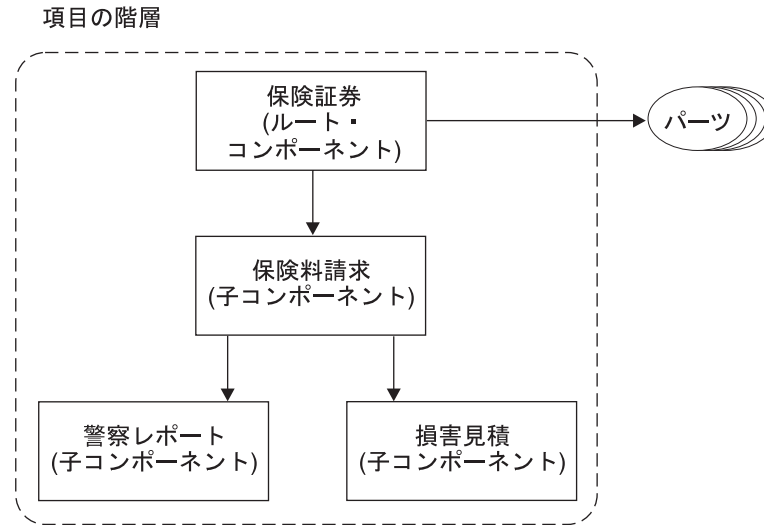


図 11. コンポーネントの階層

Content Manager のデータ・モデルの概念について、詳しくは、SItemTypeCreationICM サンプルを参照してください。

オブジェクト

オブジェクト (リソース・コンテンツとも呼ばれる) は、1 つの単位として保管、検索、および操作できるデジタル・コンテンツです (JPEG イメージ、MP3 オーディオ、AVI ビデオ、およびブックからのテキスト・ブロックなど)。オブジェクトは常にリソース・マネージャーに保管されます。オブジェクトへのアクセスは、ライブラリー・サーバーにより制御されます。

詳細については、SResourceItemCreationICM サンプルを参照してください。

リンクおよび参照

リンクを使用して、項目間の 1 対多の関連をモデル化できます。図 10 に示されているように、リンクは、項目のルート・コンポーネントに関連付けるためのみ使用できます。リンクの参照先は、項目の特定のバージョンではありません。リンクはいくつでも定義できます。リンクを使用すれば、ソースをターゲットに柔軟にリンクできます。リンクは単に 2 つの項目に関連付け、リンク先の項目や、これら 2 つの項目にリンクする他の項目へのアクセス手段になります。リンクの使用法は、ユーザー・アプリケーションによって実行時に決定されます。アプリケーション内ではリンクをいくつでも使用できます。

リンクは、アプリケーション内で使用できるオープンで制限のない柔軟な基本要素です。制限がないので、アプリケーション内でリンクに対して制限を加えることもできます。

リンクの用途の 1 つは、フォルダー関係や包含関係を表すことです。リンクを使用してフォルダーをインプリメントする場合、コンテナはコンテナ内容を所有しないことに注意してください。これは、コンテナ内の項目は、コンテナを削除しても削除されないということです。リンクについて詳しくは、SLinksICM サンプルを参照してください。

参照 は、コンポーネント (ルートまたは子) と別のルート・コンポーネントとの間で指定されます。参照は、設計時に定義されるコンポーネント内の参照属性として表されます。コンポーネント定義には、他のルート・コンポーネントを参照する任意数の参照属性を、コンポーネント・タイプの定義時に指定できます。通常、参照が所有権を示すことはありませんが、必要に応じて、所有関係をインプリメントすることができます。

コンポーネント・タイプに参照を追加すると、そのコンポーネント・タイプの項目は、別の項目を参照できるようになります。DDO の場合、DDO にはその参照の名前によって識別される属性があります。属性の値は、別の DDO に設定することができます。DDO の属性の値は、その参照によって参照される DDO です。

参照は、別のルート・コンポーネントを参照するルート・コンポーネント・タイプと子コンポーネント・タイプの両方で定義することができます。144 ページの図 10 を参照してください。リンクが項目のすべてのバージョンを参照するのに対して、参照は、項目の特定のバージョンを参照します。参照属性について詳しくは、SReferenceAttrDefCreationICM サンプルを参照してください。

文書

システム内に存在できる文書には 2 種類があります。1 つ目のタイプの文書は、意味タイプ「文書」の項目です。この項目は、文書を構成する情報を含んでいることが予期されます。このタイプの「文書」は、独立していることもあれば、データ・モデルに文書モデルをインプリメントした場合にはパーツを格納することもあります。このタイプの文書について詳しくは、SItemCreationICM API 実習サンプルを参照してください。

もう 1 つのタイプの文書は、「文書」として分類される項目タイプ (「文書モデル」とも呼ばれる) から作成された項目です。このタイプの文書は、文書パーツ (Content Manager 文書モデルの特定のインプリメンテーション) を格納しています。文書パーツには、例えばテキスト、イメージ、スプレッドシートなど、さまざまなタイプのコンテンツを組み入れることができます。文書モデルについて詳しくは、SDocModelItemICM サンプルを参照してください。

フォルダー

フォルダー は、任意のタイプの他項目を格納できる項目です。他のフォルダーも格納できます。Content Manager バージョン 8 では、フォルダーの概念は、項目間のリンク関係を使用してインプリメントされます。項目に他の項目を含めて、フォルダー階層と呼ばれる包含階層を形成することができます。例えば、保険証券 (policy) 項目は保険証券 (policy) 項目タイプに属し、複数の保険料請求 (claims) を含むことができます。これにより保険証券 (policy) は、写真、社会保障番号など他の項目を含むフォルダーとなります。あらゆる項目がフォルダーになり、任意の数の他の項目を含むことができるため、フォルダーは非常に柔軟性に富みます。詳細については、SFolderICM サンプルを参照してください。

バージョン管理

バージョン管理とは、項目の子コンポーネントのバージョンも含め、項目の複数バージョンを保管および管理する機能です。バージョン管理規則は、項目タイプを定

義するときに指定します。ある項目タイプをバージョン管理の対象にすると、その項目タイプの項目すべてがバージョン管理の対象になります。

バージョン管理には、常時とアプリケーション別の 2 種類があります。項目を常時バージョン管理の対象にすると、項目が更新されるたびに項目の新規バージョンが自動的に作成され、コンテンツ・サーバーに保管されます。項目をアプリケーション別バージョン管理の対象にすると、ユーザー・アプリケーションによる指定時にのみ新規バージョンが作成されます。

バージョン管理は、Content Manager ライブラリー・サーバーによって処理されます。項目のコンテンツはリソース・マネージャーに保管されますが、項目の各バージョンは RM コンテンツのコピーを個別に所有します。バージョン管理の重要な特性には、以下のものが含まれます。

- バージョン管理には、ルート・コンポーネントおよびその階層全体が含まれます。
- 項目タイプは、「バージョン管理 - 常時 (versioned-always)」、「バージョン管理 - なし (versioned-never) (デフォルト)、および「アプリケーション制御 (application-controlled)」バージョン管理の 3 つのバージョン管理ポリシーのうちの 1 つを持つことができます。
- CM システムに保管された項目のすべてのバージョンは、検索および取り出しが可能です。
- 項目のどのバージョンも、更新および削除することができます。
- 「アプリケーション制御 (application-controlled)」バージョン管理を持つ項目タイプの項目を更新するとき、既存バージョンに更新を適用するか、または更新に基づいて新規バージョンを作成するかをユーザーは選択できます。
- 項目のバージョンごとに、独自の永続 ID (PID) があります。PID には、いくつかのパーツがあり、そのうちの 2 つが現在のコンテキストに関係があります。最初の関係あるパーツは ItemID であり、これはすべてのバージョンの項目において同じです。もう 1 つは、バージョン番号です。項目のバージョンごとに、異なるバージョン番号を持ち、バージョン番号はストリングとして検索および設定できます。以下は、バージョン番号の使用法について説明したサンプルです。

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
String version = pid.getVersionNumber();
....
pid.setVersionNumber(version);
```

- それぞれの項目ごとに限られた数のバージョンのみを保持するように、項目タイプを構成できます。項目の更新が許可された最大数を超えると、システムは保管されている一番古いバージョンを除去し、最新バージョンを作成します。
- バージョン管理可能な項目が再度索引付けされると、その項目の以前のすべてのバージョンは自動的に削除されます。
- 項目の子コンポーネントは、親コンポーネントのバージョンを継承します。
- 子コンポーネント・タイプのバージョンは、親タイプのバージョン管理に従うため、変更できません。
- パーツ・レベルのバージョン管理規則は、項目タイプを表す項目タイプ関係オブジェクトから取得できます。

バージョン管理について詳しくは、SItemUpdateICM サンプルと SItemTypeCreationICM サンプルを参照してください。

アクセス制御

Content Manager のアクセス制御モデルは、以下の基本エレメントから構成されます。

- 特権および特権セット。
- 被制御エンティティ。
- ユーザーおよびユーザー・グループ。
- アクセス制御リスト。

さまざまなアクセス制御エレメントが、次のように機能します。Content Manager のユーザーごとに、一連のユーザー特権が付与されます。これらの特権は、ユーザーが実行できる操作を定義します。ユーザーの有効なアクセス権限が、そのユーザーに対して定義された特権を超えることはありません。

Content Manager のアクセス制御モデルは、被制御エンティティに適用されます。被制御エンティティとは、保護されたユーザー・データの単位です。Content Manager では、被制御エンティティは、項目や項目タイプのレベル、またはライブラリー全体のレベルの場合があります。例えば、項目タイプ・レベルでアクセス制御を実施するために、ACL を項目タイプへバインドすることができます。被制御エンティティでの操作は、アクセス制御リスト (ACL) と呼ばれる 1 つまたは複数の制御規則によって規制されます。Content Manager システム内のすべての被制御エンティティが、ACL へバインドされなければなりません。

ユーザーが項目上で操作を開始すると、システムはユーザーの特権および項目へバインドされた ACL を検査し、ユーザーが項目上でそのような操作を行う権利があるかどうかを判別します。論理的に、項目にアクセスする権利は、同時に、その項目が定義されている項目タイプへのアクセス権を必要とします。図 12 には、システムが、特権および ACL に基づいてユーザーの項目へのアクセス権を判断する方法の例が示されています。

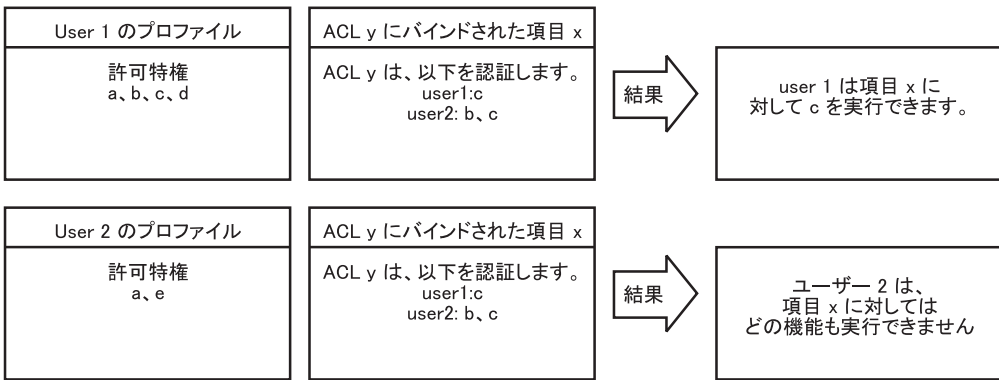


図 12. アクセス制御ダイアグラム

特権および特権セット

特権によりユーザーは、作成や削除など、システムの項目に特定のアクションを実行することができます。Content Manager のすべてのユーザーに、一連のユーザー特権が付与されます。特権は、Content Manager システム内の情報に対してユーザーが実行することができる操作の限度を定義します。ユーザーのアクセス権は、そのユーザーに対して定義されたユーザー特権を超過することはありません。

Content Manager には多数の事前定義された特権が用意されています。これはシステム定義特権と呼ばれ、変更することはできません。また、ユーザーが独自の特権を定義することもでき、これはユーザー定義特権と呼ばれます。ユーザー出口ルーチンを使用して、ユーザー定義特権をアプリケーションに適用します。

すべての特権には、システムにより生成される固有のコードがあり、これを特権定義コードといいます。特権定義コード 0 ~ 999 は、システム定義特権用に予約済みです。コード 1000 以上をユーザー定義特権に使用することができます。

システム定義特権は、システム管理特権とデータ・アクセス特権の 2 つのカテゴリに分けられます。システム管理特権を使用することにより、ユーザー・データをモデル化し、Content Manager システムの管理および保守を行うことができます。システム管理特権は、システムの構成、ライブラリー・サーバー構成の管理、および項目タイプの管理などのタスクを行う上で必要です。データ・アクセス特権は、項目や項目タイプなど、システム・データへのアクセスおよび変更に使用することができます。

ユーザーに割り当てられた特権のグループを特権セット といいます。例えば、ある特権セットに、作成、更新、および削除などの特権を含めることができます。特権セットにより、システム管理を容易に行うことができます。各特権は、これを使用する前に特権セットにグループ化する必要があります。セットに含めることができる特権の数には、制限がありません。

Content Manager の定義済み特権セットには、System Admin 特権などがあります。

AllPrivSet; PrivSetCode: 1

この特権セットを持つユーザーは、すべての Content Manager エンティティに対してすべての機能を実行することができます。この特権セットに含まれる特権は、すべてのシステム定義の特権とユーザー定義の特権です。

NoPrivSet; PrivSetCode: 2

この特権セットを持つユーザーは、すべての Content Manager エンティティに対して、どの機能も実行することができません。この特権セットに含まれる特権はありません。

SystemAdminPrivSet; PrivSetCode: 3

この特権セットを持つユーザーは、すべての Content Manager システム管理機能とデータ・モデル化機能を実行することができます。

ItemAdminPrivSet; PrivSetCode: 4

この特権セットを持つユーザーは、すべての Content Manager データ・モデル化機能と項目アクセス機能を実行することができます。

- システム定義項目タイプ特権
- 項目 SQL 選択特権

- ・ 項目タイプ照会特権
- ・ 項目照会特権
- ・ 項目追加特権
- ・ 項目セットのユーザー定義属性特権
- ・ 項目セットのシステム定義属性特権
- ・ 項目削除特権
- ・ 項目移動特権
- ・ 項目リンク特権
- ・ 項目被リンク特権
- ・ 項目所有特権
- ・ 項目被所有特権
- ・ 項目リンク追加特権
- ・ 項目リンク変更特権
- ・ 項目リンク削除特権
- ・ 項目チェックアウト特権

表 8. 特権コード

特権名	コード
ICMLogon	1
SystemAdmin	40
SystemDefineItemType	45
ItemSQLSelect	121
ItemTypeQuery	122
ItemQuery	123
ItemAdd	124
ItemSetUserAttr	125
ItemSetSysAttr	126

ユーザーおよびユーザー・グループ

ほとんどの企業で、システムに対して同じタイプのアクセス権を必要とするユーザーのグループが存在します。例えば、保険会社の保険引き受け人は全員、請求書項目タイプに対する検索、取り出し、および更新権限を必要とします。保険引き受け人、および共通のアクセス要件を持つその他ユーザーをユーザー・グループにまとめることができます。ただし、あるユーザー・グループを他のユーザー・グループに組み込むことはできません。

ユーザー・グループは、同様のタスクを行う個々のユーザーの便宜的なグループに過ぎません。ユーザー・グループは、ゼロまたは 1 人以上のユーザーで構成されます。ユーザー・グループを特権セットに割り当てることはできません。ユーザー・グループ内の各ユーザーは、特権セットを持ちます。ユーザー・グループを使用することにより、システム内のオブジェクトのアクセス制御リストを容易に作成することができます。ユーザー・グループは、他のグループに属することはできません。

アクセス制御リスト (ACL)

ユーザーが Content Manager システムに項目を作成する場合、ユーザーはこの項目に対する他のユーザーのアクセス権、およびそれらのユーザーがこの項目に対し行うことのできる操作を定義しなければなりません。項目に対しアクセス権を持つユーザー、およびそのユーザーが項目に対して実行することのできる操作のリストを、アクセス制御リスト (ACL) といいます。ACL には、1 つまたは複数の個々のユーザー ID またはユーザー・グループ、およびこれに関連付けられた特権を含めることができます。項目、項目タイプ、およびワーク・リストを ACL に関連付けることができます。特権セットは各ユーザーが持つ最大限のシステム使用権限を定義し、ACL は、項目に対する各ユーザーのアクセス権を制限します。例えば、ACL で写真項目の削除が許可されていても、John の特権セットに削除権限がなければ、John は写真を削除することができません。

被制御エンティティは、ACL コードによって特定の ACL にバインドされます。被制御エンティティに関連付けられた ACL は、バインド済みエンティティの権限を定義し、ユーザー特権を妨げることをないようにします。ACL が適用され、ユーザー特権が調べられます。

アクセス制御規則で指定されるユーザーは、個々のユーザー、ユーザー・グループ、またはパブリックです。解釈は、規則の UserKind フィールドで定義されます。説明しやすくするために、それぞれの規則タイプをユーザー用の ACL 規則、グループ用の ACL 規則、パブリック用の ACL 規則と呼びます。パブリックを指定すると、パブリック用の ACL 規則は、ユーザー特権の検査に通ったすべてのユーザーに対して、バインド済みエンティティの ACL 特権で指定された操作を実行する許可を与えます。バインド済みエンティティに対するパブリック用の ACL 特権は、システム・レベルで構成することができます。バインド済みエンティティをパブリックに開放できるようにするかどうかは、システム全体で構成することができます。構成パラメーターの名前は、PubAccessEnabled (テーブル ICMSTSysControl で定義される) です。使用不可の場合は、そのアクセス制御処理が行われる間、すべてのパブリック用の ACL 規則が無視されます。

同一の ACL 内で、1 人のユーザーが複数の規則タイプで指定される可能性があります。3 つのタイプの優先順位は、高い方から、パブリック用の ACL 規則、ユーザー用の ACL 規則、グループ用の ACL 規則です。ACL 検査を適用する場合、より優先順位の高い規則タイプが検査にパスすると、許可は認可され、処理は停止します。パブリック用の ACL 規則の検査に失敗すると、より優先順位の低い規則タイプの検査が行われます。

ただし、ユーザー用の ACL 規則の検査に失敗すると、検査は停止します。グループ用の ACL 規則は検査されません。ユーザーが個別にユーザー検査を実行すると、そのユーザーはアクセス制御アルゴリズムに基づいたグループ・タイプ・アクセスから除外されるため、続けてグループ・タイプの検査をする必要がなくなります。したがって、個々のユーザー・タイプおよびグループ・タイプのアクセス制御検査は、順次処理ではありません。順次検査でも問題はありますが、どちらか一方のみが処理されます。

ユーザーが個々のユーザー・タイプの検査に失敗すると (あるいは、そのユーザーの規則がアクセス制御テーブル内にないと)、次にグループ・タイプの検査が行われます。ユーザーがいずれかのグループに属しており、特権の検査にパスすると、許

可は認可され、処理は停止します。検査に失敗すると、アクセスが拒否され、処理はやはり停止します。複数のグループ用 ACL 規則で指定されているユーザーは、それらすべての規則の ACL 特権によって許可されます。1 人のユーザーが複数のユーザー用 ACL 規則で指定されることはありません。

CM システムが提供する構成済みの ACL には、SuperUserACL、NoAccessACL、および PublicReadACL があります。

SuperUserACL

この ACL は、CM 構成済みユーザー ICMADMIN のバインド済みエンティティに対するすべての CM 機能 (AllPrivSet) の実行を許可する単一の規則で構成されます。

NoAccessACL

この ACL は、すべての CM ユーザー (パブリック) に対してアクションを一切許可しない (NoPrivSet)、単一の規則で構成されます。

PublicReadACL

この ACL は、すべての CM ユーザー (ICMPUBLIC) に読み取り機能 (ItemReadPrivSet) を許可する、単一の規則で構成されます。これが、ユーザーの DfltACLCode に割り当てられるデフォルト値です。

Content Manager アプリケーションの計画

このセクションでは、Content Manager アプリケーション作成のための要件を示し、Content Manager の動作について説明します。アプリケーションの計画の主要部分は、ビジネスのニーズに合ったデータ・モデルの作成です。Content Manager データ・モデルについて詳しくは、「*Content Management System の計画とインストール*」、および samples ディレクトリーの tSItemTypeCreationICM サンプルを参照してください。

このセクションのトピックは以下のとおりです。

- アプリケーションの機能の決定。
- エラーの処理。

アプリケーションの機能の決定

アプリケーションの開発アプローチは、組織の必要性に応じて異なります。効率的なアプリケーションを開発するには、組織のすべての関係者がアプリケーションの計画と設計に関与する必要があります。計画に関するその他の情報は、「*Content Management System の計画とインストール*」を参照してください。

アプリケーションを作成する前に、以下の質問のすべてまたはほぼすべてに対して解答を用意する必要があります。

- 組織で使用する文書のタイプは何か。
- 既存の文書内のコンテンツのタイプは何か。
- 文書の処理方法はどのようにするか。
- 文書処理を自動化できるか。
- 文書の受信、表示、保管および配布方法はどのようにするか。
- 文書保管後の文書検索の頻度はどれぐらいか。

- 組織で管理する文書の量はどれくらいか。
- ラージ・オブジェクトの保管に使用するストレージ・メディアのタイプは何か。
- その他に組織で使用するアプリケーションはあるか。
- 予定しているユーザー数はどれくらいで、どのタイプのアクセス制御を使用するか。

上記の質問に対する解答に基づき、アプリケーションに組み込む機能を決定します。

エラーの処理

エラーを処理するとき、キャッチしなければならない最も重要な例外は、`DKException` クラスです。例外をプログラム・ロジックに使用しないでください。また、コンテンツ・サーバー内に何かが存在するかどうかを検出する目的で、あるいは純粋な例外事例以外の理由で、例外をキャッチすることに依存しないでください。プログラム・ロジックで例外を使用すると、パフォーマンスが低下し、デバッグおよびサポートには不要なトレース情報およびログ情報が提供されます。

すべての例外情報を、注意深く検討してください。`DKException` にはサブクラスが多数存在し、プログラムによっては、それぞれの例外を個別に処理するのが最善の方法となります。表 9 には、`DKException` 情報が示されています。

表 9. `DKException` 情報

<code>DKException</code>	説明
名前	例外クラス名。サブクラス名を含みます。
メッセージ	特定メッセージによりエラーが説明されます。メッセージは大量の情報を含むことができ、エラーの検出時に、重要な変数の状態をカプセル化している場合があります。
メッセージ ID	固有のメッセージ ID。このエラー・タイプを識別し、上記で使用するコア・メッセージと対応します。
エラー状態	OO API または ライブラリー・サーバーのエラー状態についての追加のエラー情報を含む場合があります。ライブラリー・サーバーがエラーを検出すると、次の 4 つの情報がここにパッケージされます。 <ul style="list-style-type: none"> 戻りコード 理由コード Ext / SQL 戻りコード Ext / SQL 理由コード
エラー・コード	ライブラリー・サーバーの戻りコードを含む場合があります。
スタック・トレース	ユーザー・プログラム内の障害ポイント、および OOAPI によってエラーが最後に検出または処理された正確な場所を示す重要な情報。

Java で作業する場合は、`java.lang.Exception` も処理する必要があります。samples ディレクトリー内の `SConnectDisconnectICM` サンプルに、エラーをキャッチして印刷する方法の実例が示されています。ロギングおよびトレースについての詳細は、「メッセージとコード」を参照してください。

Content Manager サンプルの使用

Content Manager は、Content Manager のキーとなるタスクを完了するために役立つ、コード・サンプルの広範囲のセットを提供します。サンプルは参照情報、プログラミング・ガイド、API の使用例、およびツールを提供しているので、API の理解に非常に役立ちます。

各サンプルは、製品情報センターの「*Online application Programming Reference*」で見ることができます。また、サンプルは、CMBROOT¥Samples¥java¥icm および CMBROOT¥Samples¥cpp¥icm ディレクトリーに入っています。ただし、このディレクトリーにサンプルを入れて使用するには、EIP のインストール時に「Samples and Tools」コンポーネントを選択しておく必要があります。

サンプルを最大限に活用するため、サンプルの README に必ず目を通してください。探している概念やトピックを含むサンプルがすぐに見付かるように、詳細な参照索引が用意されています。すべてのサンプルは入念に文書化され、詳細な概念情報および各タスク・ステップの説明が提供されています。各サンプルに含まれる追加情報は、以下のとおりです。

- サンプル内で示される概念を説明する、詳細なヘッダー情報。
- 前提条件情報およびコマンド行の使用法を含む、サンプル・ファイルの説明。
- アプリケーション内で容易に切り取り、カスタマイズ、および使用できる、完全にコメント化されたコード。
- アプリケーションを開発する際に使用できるユーティリティ機能。

サンプルの README の『始めに (Getting Started)』セクションを読むと、以下の一般タスクの実行方法を迅速に学習することができます。

- データのモデリング。
- サーバーへの接続とエラーの処理。
- 属性および属性グループの定義。
- 参照属性の使用。
- データ・モデルの定義。
- 項目の使用。
- リソース項目の使用。
- フォルダーの使用。
- リンクの使用。
- リソース・マネージャーの定義。
- SMS コレクションの定義。
- 項目の検索。

保険シナリオ・サンプル

Content Manager は、保険会社を例にして「実世界」で可能なインプリメンテーションのコード・サンプルを提供します。保険会社サンプルの作成に使用される情報は、Content Manager のキーとなる機能の説明を分かりやすくするために創作されたものです。保険シナリオを形成するサンプルの完全なリストは、サンプルの README を参照してください。

Content Manager アプリケーションの作成

Content Manager バージョン 8 リリース 2 の機能をインプリメントする API は、ICM コネクタと呼ばれるものにグループ化されています。ICM コネクタ API には ICM という接尾部が付きます (例、DKDatastoreICM)。

このセクションの内容は、以下のとおりです。

- ソフトウェア・コンポーネントの説明。
- DDO を使用した項目の表示。
- Content Manager システムへの接続。
- 項目の使用。

ソフトウェア・コンポーネントの説明

概念上、OO API は以下のサービス・グループに分けられます。

- データおよび文書のモデリング。
- 検索および取り出し。
- データのインポートおよびデリバリー。
- システム管理。
- 文書ルーティング。

データおよび文書のモデリング・モジュールに含まれる API により、基本となる Content Manager 階層データ・モデルにビジネス・データ・モデルをマップすることができます。例えば、保険会社のデータ・モデルには保険証券 (*policy*) が含まれ、これは Content Manager データ・モデル内では基本的に項目となります。データおよび文書のモデリング・モジュール API が提供するインターフェースにより、保険証券 (*policy*) を表す項目を定義することができます。

検索およびリトリブ・モジュールは、文書やフォルダーなど管理対象項目に関する要求を処理します。検索モジュールの API により、Content Manager システムに含まれる項目について結合テキスト検索およびパラメトリック検索を行うことができます。検索結果は、検索結果セット形式でアプリケーションに戻されます。

データのインポートおよびデリバリー・モジュールは、システムへのデータのインポート、および各種メディア経由でのデータのデリバリー (例えばネットワークまたは Web 経由など) を行うための API を提供します。

システム管理モジュールは、効率的なセキュア Content Manager システムの構成および保守のためのインターフェースを提供します。例えば、システム管理 API をアプリケーションに組み込むことにより、システム制御の設定、ユーザー管理、ユーザー特権の割り当て、システムへのアクセス許可などを調整することができます。

文書ルーティング・モジュール API を使用することにより、文書などのビジネス・オブジェクトのルーティングを、ビジネスの必要性に応じて定義されたプロセスを経由して行うことができます。

DDO を使用した項目の表示

アプリケーションを作成する前に、14 ページの『動的データ・オブジェクトの概念について』に記載された DDO/XDO プロトコルの概念について理解する必要があります。このセクションの情報は、Content Manager バージョン 8 リリース 2 に固有の内容です。

DDO は基本的に、属性のコンテナです。属性は、名前、値、およびいくつかのプロパティを持ちます。属性の最も重要なプロパティとして、属性タイプがあります。DDO は、永続的ストレージ内のオブジェクトの位置を示す永続 ID (PID) を持ちます。DDO は、それ自体を配置するいくつかのメソッド、および項目の情報を取り出す、これに対応するメソッドを持ちます。DDO メソッドには、追加、検索、更新、および削除メソッドが含まれます。これらのメソッドにより、Content Manager と同様に、項目のデータを永続的ストレージへ移動したり、永続的ストレージから移動したりすることができます。

メモリー内で、Content Manager 項目は DDO として表現されます。項目属性は、名前、タイプ、および値を持つ DDO 属性として表されます。リンクおよび参照は、特別なタイプの属性として表されます。ただし、リンク属性と参照属性の違いは、参照属性は、他の (単一の) DDO または XDO を参照するのに対し、リンク属性は、(複数の) DDO または XDO のコレクションを参照する点です。XDO は、ラージ・オブジェクト (LOB) の表現に使用されます。

項目への参照 (XDO または他の DDO のいずれかへの) も名前を持ち、そのタイプ・プロパティはオブジェクト参照に設定され、値は参照されるオブジェクトのインスタンスを参照するよう設定されています。子コンポーネントおよびリンクも DDO 属性として設定され、タイプ・プロパティはデータ・オブジェクトのコレクションに、値は DDO のコレクションに設定されます。子コンポーネントの場合、属性の名前は子コンポーネントの名前です。値は、ルート・コンポーネントに属する子コンポーネントのコレクションです。ルート項目が削除されると、そのすべての子コンポーネントも削除されます。

Content Manager システムへの接続

Content Manager アプリケーションを作成するには、最初にサーバーに接続する必要があります。このセクションでは、Content Manager サーバーへの接続および切断を行うさまざまなタスクについて説明します。

Content Manager サーバーにアクセスするためには、アプリケーションはコンテンツ・サーバーを作成する必要があります。このコンテンツ・サーバーは、共通サーバーとして機能します。コンテンツ・サーバーを作成してそれに接続するには、以下のようにします。

1. コンテンツ・サーバー・オブジェクトを作成します。

Java

```
DKDatastoreICM dsICM =new DKDatastoreICM();
```


C++

```
DKDatastoreICM *dsICM =new DKDatastoreICM();
```

2. 接続パラメーターをセットアップします。

Java

```
String database = "icmnlsdb";
String userName = "icmadmin";
String password = "password";
```

C++

```
char * database = "icmnlsdb";
char * userName = "icmadmin";
char * password = "password";
```

3. コンテンツ・サーバーの接続操作を呼び出します。 `databaseNameStr` は接続先のデータベースの名前です。

Java

```
dsICM.connect(databaseNameStr,usridStr,pwStr,"");
```

C++

```
dsICM->connect(database, userName, password, "");
```

システム構成によって、接続が可能なライブラリー・サーバーおよびリソース・マネージャーが複数存在する場合があります。接続が可能なライブラリー・サーバーの名前のリストを表示するには `DKDatastoreICM` を使用して

`listDataSourceNames()` メソッドを呼び出してから、`listDataSources()` メソッドを呼び出します。`listDataSources()` メソッドは現在接続が可能なライブラリー・サーバーをリストします。

ライブラリー・サーバーに接続した後、`DKRMConfiguration` を使用して `listResourceMgrs()` メソッドを呼び出し、そのライブラリー・サーバーに関連付けられたリソース・マネージャーのリストを取得します。

システムから切断するには、コンテンツ・サーバーで `disconnect` 操作を呼び出します。

詳しくは、`SConnectDisconnectICM` を参照してください。これは、上記のコード断片の引用元である完全なサンプルです。

パスワードの変更

ユーザーが新規のライブラリー・サーバー・セッションを開始するごとにパスワードの変更を行えるようにすることができます。パスワードの変更オプションをインプリメントするには、DKDatastoreICM を使用して、changePassword() メソッドを呼び出します。

Java

```
changePassword(String userID, String oldPwd, String newPwd)
```

C++

```
changePassword(const char* userID, const char* oldPwd, const char* newPwd)
```

項目の使用

このセクションでは、項目の作成、更新、および削除に関するプロセスについて説明します。

項目の使用に関する追加情報については、SItemCreationICM サンプルを参照してください。

項目タイプの作成

項目を作成する前に、項目タイプを作成する必要があります。項目タイプは、作成するすべての項目ごとに定義する必要があります。例えば、「保険料請求 (claim)」項目は、項目タイプ「保険証券 (policy)」の子コンポーネントです。

項目タイプを作成する際、項目タイプの種別を定義することができます。項目タイプ種別は、その項目タイプの項目をより細かく識別する項目タイプ内でのカテゴリーです。同じ項目タイプの項目はすべて、同じ項目タイプ種別を持ちます。Content Manager の項目タイプ種別には、item、resourceitem、docmodel、および docpart があります。項目タイプの作成時に項目タイプ種別を指定しない場合は、項目タイプ種別は項目 (DDO) にデフォルト設定されます。事前定義された項目タイプ種別および対応する ID 定数を表 10 にリストします。

表 10. 項目タイプ種別

種別	ID 定数	番号	説明
項目	DK_ICM_ITEMTY PE_CLASS_ITEM	0	標準の項目 (DDO)。
リソース	DK_ICM_ITEMTY PE_CLASS_RESOURCE_ITEM	1	リソース・マネージャーに保管されたデータを記述し、包含するリソース項目。リソース項目の例としては、リソース・マネージャーに保存されたビデオ、イメージ、文書などのデータがあります。

表 10. 項目タイプ種別 (続き)

種別	ID 定数	番号	説明
文書モデル	DK_ICM_ITEMTYPE_CLASS_DOC_MODEL	2	パーツを使用して文書をモデル化する項目。文書は任意数のパーツから構成され、これらは属性 DKConstant.DK_CM_DKPARTS 内に含まれます。
パーツ	DK_ICM_ITEMTYPE_CLASS_DOC_PART	3	文書モデル種別における項目 (パーツ)。

注: 定数は、Java の場合は com.ibm.sdk.common.DKConstantICM.java、C++ の場合は dkicm.DKConstantICM.h にあります。

項目タイプを作成する方法は以下のとおりです (以下のコード例を参照してください)。

1. 項目タイプを作成し、それにコンテンツ・サーバーへの参照を渡します。
2. 項目タイプに名前を付けます。
3. 項目タイプに属性を追加し、nullable、textsearchable、および unique などの修飾子を設定します。
4. 項目タイプを永続コンテンツ・サーバーに追加します。

Java

```
//This example creates an item type definition and names it.
//The item type name must be less than 15 characters in length.
DKItemTypeDefICM bookItemType = new DKItemTypeDefICM(dsICM);
bookItemType.setName("book");
bookItemType.setDescription("This is an example item type name.");
//Below, a text-searchable attribute called book title is added. The
//attribute is defined to require a unique name and also a value. The value
//does not have to be unique.
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_title");
attr.setTextSearchable(true);
attr.setUnique(true);
attr.setNullable(false);
bookItemType.add(attr);
//Here, a book_num_pages attribute is added to the book item
//type with the following characteristics: text searchable, unique,
//and a value is not required.
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_num_pages");
attr.setTextSearchable(false);
attr.setUnique(false);
attr.setNullable(false);
bookItemType.addAttr(attr);
bookItemType.add();
```

C++

```
DKDatastoreICM* pDs;
DKDatastoreICM* pDs;
...
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*)pDs->datastoreDef();
//create new ItemType
DKItemTypeDefICM * bookItemType = new DKItemTypeDefICM(pDs);
bookItemType->setName("book");
bookItemType->setDescription("This is an example item type name.");
//Create new Attribute; add it to datastore and to the ItemType
DKAttrDefICM * attr = (DKAttrDefICM *)dsDefICM->createAttr();
    attr->setName("book_title");
    attr->setType(DK_CM_VARCHAR);
    attr->setSize(80);
    attr->setTextSearchable(TRUE);
    attr->setUnique(TRUE);
    attr->setNullable(FALSE);
//Persist the attribute to the datastore
attr->add();
//Add the newly created attribute to the item type.
bookItemType->addAttr(attr);
//Create new Attribute; add it to datastore and to ItemType
attr = (DKAttrDefICM *)dsDefICM->createAttr();
    attr->setName("book_num_pages");
    attr->setType(DK_CM_INTEGER);
    attr->setTextSearchable(FALSE);
    attr->setUnique(FALSE);
    attr->setNullable(FALSE);
    attr->add();
bookItemType->addAttr(attr);
//Add the entity, bookItemType, to the datastore.
bookItemType->add();
```

詳細については、SitemTypeCreationICM サンプルを参照してください。

項目タイプのリスト

使用可能な定義済みの項目タイプのリストを取得するには、以下のようにします。

1. DKDatastoreICM コンテンツ・サーバーに接続します。
2. コンテンツ・サーバー定義への参照を取得します。
3. コンテンツ・サーバー定義オブジェクトの `listEntityNames` メソッドを呼び出して、項目タイプの名前のストリング配列を取得します。
4. ループを使用して、すべての名前をリストします。

Java

```
String itemTypeNames[] = dsICM.listEntityNames();
DKSequentialCollection itemTypeColl =
    (DKSequentialCollection) dsICM.listEntities();
dkIterator iter = itemTypeColl.createIterator();
while(iter.more()){
    dkEntityDef itemType = (dkEntityDef) iter.next();
    System.out.println(" Item type name : " + itemType.getName()); }
```

C++ の例 1

```
long larraySize = 0;
DKString * itemTypeNames = dsICM->listEntityNames(larraySize);
for (int i = 0; i < larraySize; i++) {
    cout <<(char*)itemTypeNames[i] <<endl;
}
delete [] itemTypeNames;
```

C++ の例 2

```
DKSequentialCollection * itemTypeColl = (DKSequentialCollection *)
dsICM->listEntities();
dkIterator * iter = itemTypeColl->createIterator();
while (iter->more()) {
    dkEntityDef* itemType=(dkEntityDef*)((void*)(*iter->next()));
    cout <<(char*)itemType->getName() < delete(itemType);
}
delete(iter);
delete(itemTypeColl);
```

詳細については、SItemTypeRetrievalICM サンプルを参照してください。

属性の作成

属性を作成するには、以下のようにします。

1. 属性定義オブジェクトを作成します。
2. 名前、記述、タイプ、サイズなどを設定することによって、作成したオブジェクトを記述します。作成可能な属性のタイプの例を以下に示します。
 - DKConstant.DK_CM_BLOB.
 - DKConstant.DK_CM_CHAR.
 - DKConstant.DK_CM_CLOB.
 - DKConstant.DK_CM_DATE.
 - DKConstant.DK_CM_DECIMAL.
 - DKConstant.DK_CM_INTEGER.
 - DKConstant.DK_CM_LONG.
 - DKConstant.DK_CM_SHORT.
 - DKConstant.DK_CM_TIME.
 - DKConstant.DK_CM_TIMESTAMP.
 - DKConstant.DK_CM_VARCHAR.
3. 永続コンテンツ・サーバーに新規の定義を追加します。

Java

```
//This example defines an attribute for the title of a book.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_title");
attr.setDescription("The title of the book.");
attr.setType(DKConstant.DK_CM_VARCHAR);
attr.setSize(100);
attr.add();
//This example defines an attribute for the number of pages in a book.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_num_pages");
attr.setDescription("The number of pages in the book.");
attr.setType(DKConstant.DK_CM_INTEGER);
attr.setMin((short) 0);
attr.setMax((short) 100000);
attr.add();
```

C++

```
//This example defines an attribute for the title of a book.
DKDatastoreICM * dsICM; .....
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_title");
    attr->setDescription("The title of the book.");
    attr->setType(DK_CM_VARCHAR);
    attr->setSize((long) 100);
    attr->add();
//This example defines an attribute for the number of pages in a book.
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_num_pages");
    attr->setDescription("The number of pages in the book.");
    attr->setType(DK_CM_INTEGER);
    attr->setMin((long) 0);
    attr->setMax((long) 100000);
    attr->add();
```

属性の作成について詳しくは、SAttributeDefinitionCreationICM サンプルを参照してください。

属性グループの作成、更新、および削除

属性グループを使用すると、属性の全グループを項目とサブコンポーネントに簡単に追加できます。1つの属性は、0から任意の数の属性グループに所属できます。1つの属性を複数の属性グループに追加できます。1つのデータ項目 (DDO) は複数の属性グループを含むことができます。同じ属性名を DDO で使用できますが、それぞれの属性はネーム・スペースに基づき完全に別にします。属性を属性グループに追加すると、その影響は追加後に作成されたコンポーネント・タイプのみ及びます。以前から存在していたコンポーネント・タイプは変更のないままです。以下の例では、属性グループを作成する方法を説明しています。

Java

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
attrGroup.setName("Book_Details");
//Set a description for the new attribute group
attrGroup.setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM title = (DKAttrDefICM) dsDefICM.retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM publisher = (DKAttrDefICM) dsDefICM.retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup.addAttr(title);
attrGroup.addAttr(publisher);
// add it to the persistent datastore
attrGroup.add();
```

C++

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM* attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
attrGroup->setName("Book_Details");
//Set a description for the new attribute group
attrGroup->setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM* title =
    (DKAttrDefICM *) dsDefICM->retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM* publisher =
    (DKAttrDefICM *) dsDefICM->retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup->addAttr(title);
attrGroup->addAttr(publisher);
// add it to the persistent datastore
attrGroup->add();

delete(attrGroup);
```

属性グループの名前および記述を更新するには、DKAttrGroupDefICM の update() メソッドを呼び出し、新しい属性 ID の配列を指定します。この属性がコンポーネント・タイプにすでに関連付けられている場合は、この属性グループを更新できません。

属性グループを削除する場合にも、DKAttrGroupDefICM クラスを使用します。属性グループを削除した場合、その属性グループの作成に使用された基本属性は、ライブラリー・サーバー内に残ります。属性グループの削除には、以下の例外があります。

- コンポーネント・タイプに関連付けられ、永続する属性グループは、削除できません。
- コンポーネント・タイプに関連付けられ、永続する属性グループからは、属性を除去することはできません。
- コンポーネント・タイプに関連付けられ、永続する属性グループに、属性を追加することはできません。

詳細については、SAttributeGroupDefCreationICM サンプルを参照してください。

コンテンツ・サーバー内の属性のリスト作成

以下の例では、コンテンツ・サーバー内の属性のリストを作成する方法を説明しています。

Java

```
DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM(dsICM.datastoreDef());
//Get a collection containingall Attribute Definitions.
DKSequentialCollection attrDefColl =
    (DKSequentialCollection)dsDefICM.listAttrs();
if ((attrDefColl!=null) &&(attrDefColl.cardinality()>0)){
    //Create an iterator to iterate through the collection
    dkIterator iter = attrDefColl.createIterator();
    while(iter.more()){
        //while there are still items in the list,continue
        dkAttrDef attrDef = (dkAttrDef) iter.next();
        System.out.println("-"+attrDef.getName()+":"+attrDef.getDescription());
    }
}
```

C++

```
DKDatastoreICM * dsICM; .....
DKDatastoreDefICM*dsDefICM =(DKDatastoreDefICM *)dsICM->datastoreDef();
//Get a collection containingall Attribute Definitions.
DKSequentialCollection*attrDefColl =
    (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){
    //Create an iterator to iterate through the collection
    dkIterator*iter =attrDefColl->createIterator();
    while(iter->more()){
        //while there are still items in the list,continue
        dkAttrDef* attrDef =(dkAttrDef *)iter->next()->value();
        cout <<"-"<<attrDef->getName()<<":"<<attrDef->getDescription()<<endl;
        delete(attrDef);
    }
    delete(iter);
    delete(attrDefColl);
}
delete(dsDefICM);
```

項目タイプごとの属性名のリスト作成

以下の例では、項目タイプごとに属性名のリストを作成する方法を説明しています。詳しくは、SItemTypeRetrievalICM API 実習サンプルを参照してください。

Java

```
//Get a collection containing all attr. defs for the Item Type.
DKSequentialCollection attrColl = (DKSequentialCollection)
    dsICM.listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
System.out.println("Attributes of Item Type '"+itemTypeName+":
    (" +attrColl.cardinality()+')');
//Create an iterator to iterate through the collection
dkIterator iter = attrColl.createIterator();
while(iter.more()) {
    //while there are still items in the list, continue
    DKAttrDefICM attr = (DKAttrDefICM)iter.next();
    System.out.println("-"+attr.getName()+":"+attr.getDescription());
}
```

C++

```
//Get a collection containing all Attribute Definitions for the Item Type.
DKSequentialCollection*attrColl =(DKSequentialCollection*)
    dsICM->listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
cout <<"Attributes of Item Type '"<<itemTypeName <<":
    ("<<attrColl->cardinality()<<')'<<
//Create an iterator to iterate through the collection
dkIterator* iter =attrColl->createIterator();
while(iter->more()) {
    //while there are still items in the list, continue
    DKAttrDefICM* attr =(DKAttrDefICM*)iter->next()->value();
    cout <<"-"<<attr->getName()<<":"<<attr->getDescription()<<endl;
    delete(attr);
}
delete(iter);
delete(attrColl);
```

項目の作成

項目は、少なくとも 1 つのルート・コンポーネントがある、コンポーネント DDO のツリー全体です。そのルート・コンポーネント DDO には、項目プロパティ・タイプまたは意味タイプを割り当てる必要があります。項目を作成するときには、それに項目タイプのプロパティを割り当てる必要があります。有効な項目タイプのプロパティは、文書、フォルダー、または項目です。項目タイプのプロパティは、コンテンツ・サーバーの createDDO 機能の 2 次パラメーターとして指定しなければなりません。プロパティ・タイプの値は、DDO のプロパティ、DK_CM_PROPERTY_ITEM_TYPE に保管されます。この項目タイプのプロパティを、項目の構造を記述する総合項目タイプ定義と混同しないでください。166 ページの表 11 に、使用可能な項目のプロパティ・タイプのリストを示します。

表 11. 項目タイプ・プロパティの定義

プロパティ・タイプ	定数	定義
文書	DK_CM_DOCUMENT	項目は文書または保管データを表します。この項目に含まれる情報が、文書を構成する場合があります。この項目は、特定の文書モデルのインプリメンテーションを厳密に意味するものではないため、共通文書として考えることができます。
フォルダー	DK_CM_FOLDER	項目は、コンテンツまたはオブジェクトを含んだり参照したりするオブジェクトを表します。この項目は、特定の文書モデルのインプリメンテーションを厳密に意味するものではないため、共通フォルダーとして考えることができます。
項目 (デフォルト)	DK_CM_ITEM	一般項目。この項目は、システム定義またはユーザー定義の意味タイプに適合しません。

項目は DKDDO として作成されます。システムが DKDatastoreICM の createDDO メソッドを使用して、DKDDO 構造内の重要な情報を自動的にセットアップするので、DKDDO の作成には、常に DKDatastoreICM の createDDO() メソッドを使用してください。

項目を作成する際は、項目を構成するコンポーネントを定義します。例えば、階層項目を作成する場合、子コンポーネントを作成する必要があります。

1. コンテンツ・サーバーの createDDO および createChildDDO メソッドを使用して、項目 DDO を作成し、そのすべての属性およびその他の必須情報を設定します。この例では、ログオンされ接続された DKDatastoreICM オブジェクト (名前は dsICM) を使用します。
2. ルート・コンポーネントを作成します。

Java の例 1

```
DKDDO myDocumentDDO = dsICM.createDDO("EmployeeDoc",
    DKConstantICM.DK_CM_DOCUMENT);
```

Java の例 2

```
DKDDO myFolderDDO = dsICM.createDDO("DeptFolder",
    DKConstantICM.DK_CM_FOLDER);
```

C++ の例 1

```
DKDDO* myDocumentDDO = dsICM->createDDO("EmployeeDoc",DK_CM_DOCUMENT);
```

C++ の例 2

```
DKDDO* myFolderDDO = dsICM->createDDO("DeptFolder",DK_CM_FOLDER);
```

3. ルート・コンポーネント "EmployeeDoc" の下に子コンポーネント、例えば "Dependent" を作成します。

Java

```
DKDDO myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

C++

```
DKDDO* myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

4. 親に子コンポーネントを追加します。

Java

```
short dataid = myDocumentDDO.dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection children =  
    (DKChildCollection) myDocumentDDO.getData(dataid);  
children.addElement(myDDO);
```

C++

```
short dataid = myDocumentDDO->dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection* children =  
    (DKChildCollection*)(dkCollection*)  
myDocumentDDO->getData(dataid);  
children->addElement(myDDO);
```

5. setData メソッドを使用して、DDO を適切な値と共に取り込みます。

Java

```
myDDO.setData(.....);
```

C++

```
myDDO->setData(.....);
```

6. 永続ストアにこの項目を保管します。

Java

```
myDocumentDDO.add();
```

C++

```
myDocumentDDO->add();
```

上記の例の最後のステップで、コンテンツ・サーバー内に文書が作成されます。文書 DDO がコンテンツ・サーバーに追加されると、DKParts コレクション内のすべてのパーツを含む、そのすべての属性が追加されます。文書 DDO がコンテンツ・サーバーに追加されると、すべての子、および DKParts コレクション内のすべてのパーツを含む、文書 DDO の属性すべてが追加されます。

意味タイプ は、項目の使用法または規則を定義します。Content Manager には、いくつかの意味タイプが事前定義されていますが、ユーザー独自の意味タイプを定義することもできます。

意味タイプは、コンテンツ・サーバーの createDDO 機能の 2 次パラメーターとして指定することができます。意味タイプの値は、DDO のプロパティ

DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE 内に保管され、項目プロパティ・タイプと同じ値を含むことができます。ICM コネクターは、フォルダーおよび文書の意味タイプをサポートします。また、独自の意味タイプを作成することもできます。表 12 に、使用可能な意味タイプのリストを示します。

表 12. 定義済みの意味タイプ

意味タイプ	定数	定義
文書	DK_ICM_SEMANTIC_TYPE_DOCUMENT	この項目が文書であることを示します。
フォルダー	DK_ICM_SEMANTIC_TYPE_FOLDER	この項目がフォルダーであることを示します。
注釈	DK_ICM_SEMANTIC_TYPE_ANNOTATION	この項目またはパートが基本部分の注釈であることを示します。
コンテナ	DK_ICM_SEMANTIC_TYPE_CONTAINER	この項目が、他の項目を含むことのできるコンテナであることを示します。コンテナ - コンテナの関係がリンクを使用して表示されます。
履歴	DK_ICM_SEMANTIC_TYPE_HISTORY	この項目またはパートが基本部分の履歴であることを示します。
メモ	DK_ICM_SEMANTIC_TYPE_NOTE	この項目またはパートが基本部分のメモであることを示します。
基本	DK_ICM_SEMANTIC_TYPE_BASE	この項目またはパートが、それに関係する注釈、メモ、または履歴を持つことができる基本部分であることを示します。

表 12. 定義済みの意味タイプ (続き)

意味タイプ	定数	定義
ユーザー定義	ユーザー定義	アプリケーションが解釈するユーザー定義の意味タイプ。

注:

- Java では、定数は DKConstantICM.java の中で定義されます。
- C++ では、定数は DKConstantICM.h の中で定義されます。
- Content Manager バージョン 7 では、意味タイプを関連タイプと呼びます。

リソース項目タイプとして定義されていた項目タイプ内で項目を作成すると、正確な XDO が戻されます。リソースの詳細については、SResourceItemCreationICM サンプルを参照してください。SItemCreationICM サンプルにある項目作成に関する情報を検討すると、役に立つことがあります。

項目属性値の設定と検索

Java

属性値は java.lang.Objects として保管および検索されます。属性値を設定または検索するには、それぞれ DDO の setData および getData() メソッドを使用します。属性タイプに対応するタイプを持つオブジェクトを使用してその値を設定します。

例:

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    nameOfAttrStr),valueObj);
Object obj = ddo.getData(ddo.dataId(DK_CM_NAMESPACE_ATTR,nameOfAttrStr));
```

上記のステートメントは、java.lang.Object valueObj として渡された値に属性値を設定します。nameOfAttrStr は属性のストリング名です。この属性は、この DDO のタイプが定義されるときに、項目タイプに定義および指定されます。valueObj は、設定が必要な値なので、この属性に適正なタイプでなければなりません。

C++

個々の属性に値を設定するには、個々の属性定義名を使用してください。属性グループに属する属性を利用するには、

<Attribute Group Name>.<Attribute Name> という形式を使用してください。

例:

```
//The code snippet below shows the value of the character attribute
//"book_title" for the item represented by the DDO ddoDocument is set to
//the value "Effective C++"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
DKString("book_title")),DKString("Effective C++"));
//The code snippet below shows the value of the integer attribute
//"book_num_pages" for the item represented
//by the DDO ddoDocument is set to the value "250"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
DKString("book_num_pages")), (long)250);
//This code snippet shows how the value of the "book_title" attribute of the
//item represented by the DDO ddoDocument is retrieved into the "title"
//string variable.
DKString title =(DKString) ddoDocument->getData(ddoDocument->
dataId(DK_CM_NAMESPACE_ATTR,DKString("book_title")));
```

項目の属性の変更

項目の属性を変更するには、DDO の setData メソッドを使用して、java.lang.Object を使用して値を指定することにより、必要な値に設定します (次の例を参照)。次の例で、nameOfAttrStr が属性のストリング名、valueObj が値です。

Java

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
nameOfAttrStr),valueObj);
```

C++

```
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,"book_title"),
DKString("More Effective C++"));
```

項目の更新

項目を更新するには、以下のようにします。

1. 項目を検索するか項目を作成して、コンテンツ・サーバーに追加します。
2. 項目、属性、リンク・コレクションなどを変更します。
3. DDO の update 操作を呼び出します。

Java

```
ddo.update();
```

C++

```
ddo->update();
```

項目のバージョン管理を使用可能にすると、現在の項目を更新する代わりに、項目の新規バージョンを作成することができます。以下に例を示します。

Java

```
ddo.update(DKConstant.DK_CM_VERSION_NEW);
```

C++

```
ddo->update(DK_CM_VERSION_NEW);
```

項目の最新バージョンを更新するには、以下の例に示すようなフォーマットを使用します。

Java

```
ddo.update(DKConstant.DK_CM_VERSION_LATEST);
```

C++

```
ddo->update(DK_CM_VERSION_LATEST);
```

また、適切なオプションを使用して、DKDatastoreICM 上で `updateObject` メソッドを呼び出すことによって DDO を更新することもできます。以下に例を示します。

Java

```
// Connected DKDatastoreICM object named ds;  
// DKDDO object named ddo;  
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;  
ds.updateObject(ddo, options);
```

C++

```
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;  
ds->updateObject(ddo, options);
```

重要: 更新メソッドを呼び出す前にその項目を点検し、更新終了後にその項目をもう一度確認しなければなりません。 178 ページの『項目のチェックインとチェックアウト』を参照してください。項目の更新について詳しくは、 SItemUpdateICM API 実習サンプルを参照してください。

リソース項目タイプの定義

リソース項目は、他のシステムで定義された属性を持つ項目です。それらの属性は、項目が表すオブジェクトの位置、タイプ、サイズなどを定義します。オブジェクトは「リソース」と呼ばれることもあり、ビデオ・ファイル、イメージ、ワード・プロセッサ文書などがこれにあたります。詳しい情報は、 SItemTypeCreationICM サンプルを参照してください (samples ディレクトリー内にあります)。

以下のステップでは、リソース項目タイプを定義するプロセスを順に説明します。

1. 接続したコンテンツ・サーバーから、コンテンツ・サーバー定義オブジェクトを取得します。

Java

```
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
```

C++

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
```

2. 新規の項目タイプの定義を作成します。

Java

```
DKItemTypeDefICM itemType = new DKItemTypeDefICM(dsICM);  
itemType.setName("SampleResource");  
itemType.setDescription("Simple Resource Lob Item Type");
```

C++

```
DKItemTypeDefICM* itemType = new DKItemTypeDefICM(dsICM);  
itemType->setName("SampleResource");  
itemType->setDescription("Simple Resource Lob Item Type");
```

3. 属性を追加します。

Java

```
DKAttrDefICM attr =(DKAttrDefICM)dsDefICM.retrieveAttr("S_varchar");
itemType.addAttr(attr);
//Resource classification indicates that this class will
//contain data file
itemType.setClassification
(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

C++

```
DKAttrDefICM * attr = (DKAttrDefICM*) dsDefICM->retrieveAttr("S_varchar");
itemType->addAttr(attr);
// Resource classification indicates that this class will contain
//data file
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

4. この項目タイプにリソースの XDO クラスとタイプを指定します。

Java

```
itemType.setXDOClassName(DKConstantICM.DK_ICM_XDO_LOB_CLASS_NAME);
itemType.setXDOClassID(DKConstantICM.DK_ICM_XDO_LOB_CLASS_ID);
itemType.add();
```

C++

```
itemType->setXDOClassName(DK_ICM_XDO_LOB_CLASS_NAME);
itemType->setXDOClassID(DK_ICM_XDO_LOB_CLASS_ID);
itemType->add();
```

リソース項目の作成

リソース項目の作成は、通常の項目の作成方法と非常によく似ています。XDO は DDO を拡張したもので、リソース項目のタイプに応じて、XDO をさらに拡張することもできます。表 13 に、リソース項目タイプとそれらを作成するために使用したクラス階層を示します。詳しくは、SResourceItemCreationICM サンプルを参照してください (samples ディレクトリーにあります)。

表 13. リソース項目タイプ・クラス階層

タイプ	DDO	XDO	拡張子
LOB	DKDDO ->	DKLobICM	
本文	DKDDO ->	DKLobICM ->	DKTextICM
イメージ	DKDDO ->	DKLobICM ->	DKImageICM
ストリーム	DKDDO ->	DKLobICM ->	DKStreamICM

次のステップでは、リソース項目を作成するプロセスを説明します。リソース・コンテンツを設定して保管する方法は複数あります。次の手順は、項目をコンテンツ・サーバーに追加する時点で、ファイルから直接保管する場合の例です。

1. リソース項目を作成します。リソース項目はすべての意味タイプになる可能性があります。また `DKDatastoreICM::createDDO` 呼び出しは、通常の項目の作成に使用されるのと同じ方法でリソース項目の作成に使用されます。

`DKDatastoreICM.createDDO` から戻されるリソースのタイプは、XDO 分類に基づき適切なサブクラス (この場合は `DKLobICM`) にキャストされます。XDO 分類はこのリソース項目の基になる項目タイプの作成時に定義されます。

Java

```
DKLobICM    lob = (DKLobICM)dsICM.createDDO  
            ("SampleResource",DKConstant.DK_CM_DOCUMENT);
```

C++

```
DKLobICM*    lob = (DKLobICM*)  
dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. コンテンツまたはデータをオブジェクトに設定します。lob の作成後、リソースの MIME タイプを設定できます。この場合、リソースは MS Word 文書です。MIME タイプは、保管されているコンテンツのタイプを記述します。

Java

```
lob.setContentFromClientFile(fileName);  
lob.setMimeType("application/msword");
```

C++

```
lob->setContentFromClientFile(fileName);  
lob->setMimeType("application/msword");
```

MIME タイプについて詳しくは、`SResourceItemMimeTypesICM` サンプルを参照してください。

3. コンテンツ・サーバーにデータを追加します。この場合は、サンプルの Word 文書です。リソース項目コンテンツは、リソース・マネージャーに保管されるので注意してください。

Java

```
lob.add("SResourceItemICM_Document1.doc");
```

C++

```
lob->add("SResourceItemICM_Document1.doc");
```

重要: C++ では、メモリーをクリーンアップする必要があります。

```
delete(lob);
```

詳細については、SItemUpdateICM サンプルを参照してください。

項目の検索

基準セットに一致する項目を検索するには、以下のステップを実行します。

1. DKDatastoreICM オブジェクトに接続します。
2. 適切な照会の処理を行います。照会は、照会言語に準拠してください。詳しくは、201 ページの『照会言語について』を参照してください。
3. 照会結果を DKResult オブジェクトに保管します。

Java

```
DKNVPair options[] = new DKNVPair[3];
options[0] =
    new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No Max (Default)
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,
    new Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
DKResults results = (DKResults)dsICM.evaluate(query,
    DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
```

C++

```
DKNVPair *options = new DKNVPair[3];
// only one result will be returned
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Retrieve only one item
options[1].set(DK_CM_PARM_RETRIEVE , (long)DK_CM_CONTENT_ATTRONLY);
// Last option has to be this value
options[2].set(DK_CM_PARM_END , NULL);
//Note if a query is expected to return more than one result,
//the DKDatastoreICM::execute method
//should be used. The execute method returns a dkResultSetCursor.
DKResults* results = (DKResults*)(dkCollection*)dsICM->evaluate
    (query, DK_CM_XQPE_QL_TYPE, options);
```

詳細については、SSearchICM サンプルを参照してください。

項目の検索

項目を明示的に検索したり、照会によって検索した項目をリフレッシュしたりするには、検索またはリフレッシュする項目の PID オブジェクト、または PID ストリングにアクセスすることが必要です。 ItemID のみが分かっている場合は、照会

を実行して完全な PID 情報を検索できます。次のシナリオは、項目タイプ名と項目 ID が分かっているとして、明示的検索を行いたい場合に DDO を検索する方法を示しています。

Java

1. datastoreICM オブジェクトに接続した後で、適切な項目 ID を使用して項目を検索します。照会の作成については、202 ページの『Content Manager サーバーの照会』を参照してください。
2. 照会結果を DKResults オブジェクトに保管します。以下のコード例の queryStr は、正しい照会言語形式の検索ストリングです。

```
DKResults results = (DKResults)dsICM.evaluate(queryStr, DKConstantICM.DK_CM_XQPE_QL_TYPE, null);
```
3. DKResults にイテレーターを作成し、それを使用して DDO を取得します。これで DDO を 1 つずつ検索することができます。

```
ddo.retrieve();
```
4. 次の例に示すとおり、DDO から取得した PID ストリングがあるとします。

```
DKPidICM pid = ddo.getPidObject();
String pidStr = pid.pidString();
```

PID ストリングを使用して、次のステップを完了すると検索を実行することができます。

1. DKDatastoreICM の createDDO メソッドを使用して DDO を作成します。DKDDO コンストラクターは使用しないでください。以下の例では、dsICM オブジェクトは既に Content Manager データ・ストアに関連付けられています。また、PID は、pidStr という名前のストリングです。

```
DKDDO ddo = dsICM.createDDO(pidStr);
```
2. DDO の retrieve 操作を呼び出します。

```
ddo.retrieve();
```

C++

1. datastoreICM オブジェクトに接続した後で、適切な項目 ID を使用して項目を検索します。照会の作成については、Content Manager サーバーの照会を参照してください。

2. 照会結果を DKResult オブジェクトに保管します。以下のコード例の queryStr は、正しい照会言語形式の検索ストリングです。

```
DKResults* results = (DKResults*)(dkCollection*)
dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE,NULL);
```

3. DKDatastoreICM の createDDO メソッドを使用して DDO を作成します。DKDDO コンストラクターは使用しないでください。以下の例では、dsICM オブジェクトは既に Content Manager データ・ストアに関連付けられています。また、PID は pidStr という名前のストリングです。

```
DKDDO* ddo = dsICM->createDDO(pidStr);
```

4. DDO の retrieve 操作を呼び出します。

```
ddo->retrieve();
```

項目のバージョン管理を使用可能にすると、適切なオプションとともに DKDDO クラスの retrieve メソッドを使用することによって、項目の特定のバージョンを検索することができます。項目の最新バージョンを検索するには、検索オプションとして定数 DK_CM_VERSION_LATEST を使用してください。デフォルトでは (オプションが指定されていない場合は)、ddo.retrieve() を使用して項目の最新バージョンを検索します。

例:

```
DKDDO ddo = ds.createDDO(...);
.... ddo.retrieve(DK_CM_VERSION_LATEST);
```

また、DKDatastoreICM オブジェクト上で retrieveObject メソッドを呼び出すことによって、DDO を検索することもできます。

例:

```
DKDatastoreICM ds =new DKDatastoreICM();
//connect,etc ...
DKDDO ddo =ds.createDDO(itemType,option);
//sets the PID as shown above...
ds.retrieveObject(ddo);
```

項目の検索について詳しくは、SItemRetrievalICM および SResourceItemRetrievalICM API 実習サンプルを参照してください。

項目のバージョン管理を使用可能にした場合、属性および子を含めて項目の最新バージョンを検索するには、以下のフォーマットを使用します。

Java

```
int options =DK_CM_CONTENT_ATTRONLY + DK_CM_CONTENT_CHILDREN +  
DK_CM_VERSION_LATEST;  
ds.retrieveObject(ddo,options);
```

C++

```
DKDDO * ddo;  
long options =  
DK_CM_CONTENT_ATTRONLY +DK_CM_CONTENT_CHILDREN +DK_CM_VERSION_LATEST;  
dsICM->retrieveObject(ddo,options);
```

検索オプションなど、検索について詳しくは、 SItemRetrievalICM サンプルを参照してください。

項目の削除

コンテンツ・サーバーから項目を削除するには、DDO の削除メソッドを使用します。

Java

```
ddoDocument.del();
```

C++

```
ddoDocument->del();
```

DDO には、その項目 ID とオブジェクト・タイプの集合、およびコンテンツ・サーバーに対する有効な接続がなければなりません。

項目の削除について詳しくは、SItemDeletionICM API 実習サンプルを参照してください。

項目のチェックインとチェックアウト

2 名のユーザーが同時に同じ項目に変更を加えることがないように、更新を行う前に項目をチェックアウトする必要があります。項目をチェックアウトすることにより、項目をチェックアウトしたユーザーに排他的な更新権が与えられます。項目をチェックアウトすると、その項目に対して永続書き込みロックがかかります。すべてのバージョンと子コンポーネントを含め、項目全体が 1 単位としてロックおよびアンロックされます。ただし、リンクおよび参照されている項目は、チェックアウトした項目と一緒にロックされません。項目をチェックインすると、項目に対する永続ロックが解放されます。

項目をチェックインおよびチェックアウトするには、以下の例で示すとおり、DKDatastoreICM の checkIn および checkOut 操作を使用します。

1. DKDatastoreICM オブジェクト (名前 dsICM) に接続し、DDO (myDataObject) を使用して、項目をチェックアウトします。

Java

```
dsICM.checkOut(myDataObject);
```

C++

```
dsICM->checkOut(myDataObject);
```

2. 項目をチェックインします。

Java

```
dsICM.checkIn(myDataObject);
```

C++

```
dsICM->checkIn(myDataObject);
```

項目のチェックインおよびチェックアウトについては、SItemUpdateICM API 実習サンプルを参照してください。

項目のバージョン管理プロパティの設定と取得

以下の例では、項目のバージョン管理プロパティを設定し、検索する方法と、その項目のバージョン管理プロパティについて説明します。

Java

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();  
// Accessing version information  
String version = pid.getVersionNumber();  
...  
// Setting the version number in the DDO  
pid.setVersionNumber(version);
```

C++

```
DKPidICM * pid = (DKPidICM *)ddo->getPidObject();
// Accessing version information
DKString version = pid->getVersionNumber();
....
//Setting the version number for the PID associated with an item (DDO).
pid->setVersionNumber((char *)version);
```

バージョン管理プロパティの使用

このセクションでは、バージョン管理プロパティの使用例について説明します。

項目タイプのバージョン管理ポリシーの検索

項目には、その項目のバージョン管理プロパティを含むバージョン管理ポリシーがあります。以下は、使用できる 3 つのバージョン管理プロパティと、バージョン管理ポリシー内でそれぞれのプロパティを表すために使用されている値をリストしています。

DK_ICM_VERSION_CONTROL_ALWAYS

常にバージョン管理が行われる。

DK_ICM_VERSION_CONTROL_NEVER

この項目タイプではバージョン管理はサポートされていない (デフォルト)。

DK_ICM_VERSION_CONTROL_BY_APPLICATION

バージョン管理方式はアプリケーションによって決まる。

Java

```
short versionControlPolicy;
DKItemTypeDefICM item = newDKItemTypeDefICM();
....
versionControlPolicy = item.getVersionControl();
```

C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item = NULL;
...
versionControlPolicy = item->getVersionControl();
```

項目タイプのバージョン管理を設定する

Java

```
DKItemTypeDefICM item = new DKItemTypeDefICM();
short versionControl = DK_ICM_VERSION_CONTROL_ALWAYS;
....
item.setVersionControl(versionControl);
```


C+

```
DKItemTypeDefICM * item = NULL;
short versionControl    = DK_ICM_VERSION_CONTROL_ALWAYS;
...
item->setVersionControl(versionControl);
```

システム内の項目タイプ定義のバージョン管理情報にアクセスする完全なサンプルについては、SItemTypeRetrievalICM API 実習サンプルを参照してください。

項目タイプで利用できるバージョンの最大数を取得する

Java

```
short versionMax;
DKItemTypeDefICM item = new DKItemTypeDefICM();
....
versionMax = item.getVersionMax();
```

C++

```
short versionMax    = 0;
DKItemTypeDefICM * item = NULL;
....
versionMax = item->getVersionMax();
```

項目タイプで利用できるバージョンの最大数を設定する

この例では、この項目タイプを基にした項目の 10 バージョンのみがシステムによって保守されます。

Java

```
short versionMax=10;
DKItemTypeDefICM item = new DKItemTypeDefICM();
...
item.setVersionMax(versionMax);
```

C++

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
....
item->setVersionMax(versionMax);
```

項目タイプ用のバージョン管理タイプの値を設定する

C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

フォルダーの使用

フォルダーは、項目を表す完全にサポートされた DDO です。フォルダーは、項目タイプの完全階層データ構造を持ち、その内部で作成されます。フォルダーは、項目タイプの種別に関係なく、意味タイプ・フォルダーの DDO であり、DKFolder を含む属性 DKConstant.DK_CM_DKFOLDER を持っています。DKFolder オブジェクト DKSequentialCollection には、他の項目を表現するルート・コンポーネント DDO をいくつでも含めることができます。

以下の手順には、情報提供だけを目的としたコード・フラグメントが含まれています。これらのコード・フラグメントは、現在のフォームでは機能しないため、これらをコピーして、アプリケーション・プログラムへ直接貼り付けしないでください。実行できる完全なフォルダー・サンプルは、samples ディレクトリーにある SFolderICM サンプルを参照してください。

フォルダーの作成

実行時の DDO の作成には、DKDatastoreICM.createDDO() メソッドが使用されます。samples ディレクトリー内の SItemCreationICM サンプルに示されているとおり、フォルダー項目の作成時には、項目プロパティー・タイプまたは意味タイプに DKConstant.DK_CM_FOLDER を指定しなければなりません。

フォルダー項目は、DKDatastoreICM.createDDO メソッドを DK_CM_FOLDER 意味タイプと同時に使用して作成されます。フォルダーの作成には、DKConstant.DK_CM_FOLDER を createDDO メソッドの 2 次パラメーターとして指定することもできます。

Java

```
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
```

C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

setData メソッドを使用して、ddoFolder を取り込みます。フォルダー DDO が作成されるとすぐに、永続コンテンツ・サーバーに保管されます。

Java

```
ddoFolder.add();
```

C++

```
ddoFolder->add();
```

フォルダーへのコンテンツの追加

DKFolder に配置されるすべてのコンテンツは、フォルダー DDO 上で add() メソッドまたは update() メソッドを呼び出す前に、コンテンツ・サーバー内に永続している必要があります。コンテンツを永続させるには、コンテンツの DKDDO.add() メソッドを呼び出します。

項目をフォルダーへ追加するには、DKFolder の addElement() 関数を使用します。十分な数のメンバーをフォルダーに追加している場合は、フォルダーとコンテンツの関係を永続ストアに保管するために、追加メソッドまたは更新メソッドを呼び出すことができます。これは、任意の数のフォルダー変更をライブラリー・サーバーへの単一呼び出しへグループ化します。項目をフォルダーへ追加するときに、同じ項目の複数のコピーを DKFolder へ追加しないでください。フォルダーの変更内容はすべて追跡されるので、競合または重複する変更を行わないでください。例えば、フォルダーに同じ項目の複数のコピーを追加しないでください。

フォルダーにコンテンツを追加するには、以下のステップを実行します。フォルダーは、そのコンテンツ項目として別のフォルダー (サブフォルダー) を含むことがあるので注意してください。

Java

1. 新規に 3 つの項目を作成します。これらの項目は、フォルダーにコンテンツとして追加されます。フォルダー・コンテンツは、すべての意味タイプを使用できます。

```
DKDDO ddoDocument=dsICM.createDDO("S_simple",DKConstant.DK_CM_DOCUMENT);  
DKDDO ddoFolder2=dsICM.createDDO("S_simple",DKConstant.DK_CM_FOLDER);  
DKDDO ddoItem=dsICM.createDDO("S_simple",DKConstant.DK_CM_ITEM);
```

2. setData メソッドを使用し DDO を取り込みます。フォルダー・コンテンツとして追加された項目は、データ・ストア内に永続されなければなりません。

```
ddoDocument.add();  
ddoItem.add();  
ddoFolder2.add();
```

3. あらかじめ作成し、永続させたフォルダー DDO の dkFolder 属性を検索します。

```
short dataid = ddoFolder.dataId  
              (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);  
dkFolder = (DKFolder) ddoFolder.getData(dataid);
```

4. フォルダーは更新前に (コンテンツを追加して)、データ・ストアからチェックアウトする必要があります。

```
dsICM.checkOut(ddoFolder);
```

5. 作成してあった項目をフォルダーに追加します。

```
dkFolder.addElement(ddoDocument);  
dkFolder.addElement(ddoItem);  
dkFolder.addElement(ddoFolder2);
```

6. フォルダーへの変更を確定して、その変更を明示的に検査します。

```
ddoFolder.update();  
dsICM.checkIn(ddoFolder);
```

C++

1. フォルダーに追加する項目を作成します。

```
DKDDO * ddoDocument = dsICM->createDDO("book", DK_CM_DOCUMENT);  
DKDDO * ddoFolder2 = dsICM->createDDO("book", DK_CM_FOLDER);  
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
```

2. 作成した項目をデータ・ストアに置きます。

```
ddoDocument->add();  
ddoItem->add();  
ddoFolder2->add();
```

3. フォルダーの DKFolder 属性を検索します。

```
DKFolder * dkFolder;  
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER);  
if (dataid!=0) dkFolder =  
    (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
```

4. フォルダーをチェックアウトします (更新する前にフォルダーをチェックアウトしなければなりません)。

```
dsICM->checkOut(ddoFolder);
```

5. 作成した項目をフォルダーに追加します。

```
dkFolder->addElement(ddoDocument);  
dkFolder->addElement(ddoItem);  
dkFolder->addElement(ddoFolder2); // Folders can contain sub-folders.
```

6. フォルダーを更新します。これによって、暗黙的に、フォルダーをチェックインします (それをアンロックします)。

```
ddoFolder->update();
```

7. フォルダーを明示的にチェックインします。

```
dsICM->checkIn(ddoFolder);
```

フォルダーからのコンテンツの除去

2 つの方法のいずれかで、項目をフォルダーから除去することができます。据え置き除去法 (フォルダー DDO が更新され、複数のコンテンツ・サーバー呼び出しが 1 つに結合されるときに実際の除去が行われる) は、`removeElementXX method(s)` を使用して行います。即時 (複数の呼び出しがコンテンツ・サーバーに行われるのでコストもかかる) 除去法は、`dkFolder.removeMember` メソッドを使用して行うことができます。フォルダーの使用についての詳細は、(samples ディレクトリーにある) `SFolderICM` サンプルを参照してください。

フォルダーからコンテンツを除去するには、以下のステップを完了してください。

Java

1. 項目を除去したいフォルダー DDO をチェックアウトします。

```
dsICM.checkOut(ddoFolder);
```

2. 除去する項目を探します。この場合、前に作成した docItem DDO を検索します。フォルダー・コンテンツ内でこれを繰り返すためにイテレーターを作成します。

```
dkIterator iter = dkFolder.createIterator();
String itemPidString = ddoItem.getPidObject().pidString();
while(iter.more()) {
    // Move the pointer to next element and return that object.
    // Compare the PID trings of the DDO returned from the iterator
    // with the DDO that is to be removed.
    DKDDO ddo =(DKDDO)iter.next();
    if(ddo.getPidObject().pidString().equals(itemPidString)) {
        // The item to be removed is found.
        // Remove it (current element in the iterator)
        dkFolder.removeElementAt(iter);
    }
}
```

3. 変更内容を保持し、フォルダー DDO をチェックインします。

```
ddoFolder.update();
dsICM.checkIn(ddoFolder);
```

C++

1. 項目を作成し、フォルダーに追加します。フォルダーはあらかじめ作成しておいてください。

```
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
ddoItem->add();
DKFolder * dkFolder;
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER);
if (dataid!=0) dkFolder =
    (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
dsICM->checkOut(ddoFolder);
dkFolder->addElement(ddoItem);
ddoFolder->update();
```

2. フォルダーを明示的にチェックインします。

```
dsICM->checkIn(ddoFolder);
```

3. フォルダーのイテレーターを作成します。

```
dkIterator* iter = dkFolder->createIterator();
```

4. 除去する項目が見つかるまでフォルダーのコンテンツ内で繰り返します。
項目を除去します。

```
while(iter->more())
{
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    if ( ((DKPidICM*) ddo->getPidObject())->pidString() ==
        ((DKPidICM*) ddoItem->getPidObject())->pidString() )
    {
        //Found the element to be removed. Remove it
        dkFolder->removeElementAt(*iter);
        // Now that we found the ddoItem, remove it.
    }
}

delete(iter);
```

フォルダーのコンテンツの検索

フォルダー・コンテンツの検索には、検索オプションを設定する必要があります。デフォルトでは検索オプションは設定されません。検索オプションを設定していない場合は、検索が呼び出されて正しいオプションが設定されるまで、オブジェクトに DKFolder または DK_CM_DKFOLDER 属性は含まれません。検索オプションには、以下が組み込まれます。

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

アウトバウンド・リンクが指定された DKFolder 属性コレクションを持つフォルダー項目を検索するには、以下のフォーマットを使用します。

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
```

項目ツリーにリンクおよびフォルダー・コンテンツを含むフォルダー項目を検索するには、以下のフォーマットを使用します。

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ITEMTREE);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

特定の DDO を含むすべてのフォルダーの入手

複数のフォルダーに同じ項目 (DDO) を含めることができます。これにより、項目を別々のアプリケーションやユーザーに関連付けることができます。特定の DDO を現在含んでいるフォルダーを判別するには、次の例を参照してください。

以下のステップでは、前に作成した `ddoDocument` オブジェクトを含むフォルダーの検索方法を示します。フォルダーの使用についての詳細は、`SFolderICM` サンプル (samples ディレクトリー内) を参照してください。

Java

1. データベース拡張オブジェクトを検索します。

```
DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
                        dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
```

2. ddoDocument オブジェクトを含むフォルダーを検索する場合は、拡張オブジェクト上の getFoldersContainingDDO メソッドを呼び出して下さい。このメソッドは、DDO のコレクションを戻します。ここで、戻される各 DDO は ddoDocument オブジェクトを含むフォルダーです。

```
DKSequentialCollection list =
    dsExtICM.getFoldersContainingDDO(ddoDocument);
```

3. 戻されたフォルダーのコレクション内でこれを繰り返すために、イテレーターを作成します。

```
iter =list.createIterator();
while(iter.more()) {
    // Move iterator to next element and return that object.
    DKDDO ddo =(DKDDO) iter.next();
    // Print out the item Id of the folder DDO in order to identify
    // the returned folder object.
    System.out.println("-Item ID: " +
        ((DKPidICM)ddo.getPidObject()).getItemId());
}
```

C++

```
DKDDO* ddoDocument = ....;
//Create datastore object,connect to datastore,create DDO etc.
....
//Create a new datastore extension object from the connected object
DKDatastoreExtICM* dsExtICM = (DKDatastoreExtICM*)
                                dsICM->getExtension(DK_CM_DATASTORE_EXT);
//Retrieve the PID for the created DDO
DKPidICM* pid = (DKPidICM*) ddoDocument->getPidObject();
//Retrieve a list of folders containing the DDO created earlier.
DKSequentialCollection *list =
    dsExtICM->getFoldersContainingDDO(ddoDocument);
//Create a iterator for the returned DDO collection
dkIterator* iter =list->createIterator();
while(iter->more()) {
    DKDDO* ddo =(DKDDO*) iter->next()->value();
    pid = (DKPidICM*) ddo->getPidObject();
    cout << "-Item ID:" << pid->getItemId() << endl;
    delete ddo;
}
delete iter;
```

項目間のリンクの定義

リンクを使用すると、ソース項目をターゲット項目に関連付けることができます。オプションで記述項目を付けることもできます。以下のリンク・エレメントのタイプを指定できる DKLink オブジェクト内にリンクを定義します。

表 14. リンク・データ構造

DKLink	定義
LinkTypeName	リンクのタイプ。
ソース	リンクのソース項目。
ターゲット	リンクのターゲット項目。
LinkItem	記述項目。オプション。

リンクは 1 対多の関係を表しているので、DDO 内の LINK ネーム・スペースの下に、データ項目ごとに DKLinkCollection の値で表示されます。リンクの使用について詳しくは、samples ディレクトリーの SLinksICM サンプルを参照してください。

リンク自体は、ソースまたはターゲットのいずれにも属しません。リンクは単にソースとターゲットを接続するだけです。例えば、ソース A がターゲット B にリンクされている場合、A または B どちらの DDO が参照されるかに関係なく、常に A がソースで B がターゲットとなります。

メモリーでは、ソースおよびターゲット DDO は共に同じ DKLink オブジェクトのコピーを含みます。DKLink オブジェクトにはソースとターゲットの両方への参照が含まれるため、リンクを含む DDO には、それ自体および他の DDO を参照するリンクも含まれます。例えば、ソース A がターゲット B にリンクされている場合、A と B の両方に同じリンクが含まれます。この例を表 15 に示します。

表 15. DKLink 定義の例

A から B に定義された DKLink	DDO A	DDO B
ソースは A	ソースは A	ソースは A
ターゲットは B	ターゲットは B	ターゲットは B

永続ストアでリンクを追加または削除するには、ソースまたはターゲットの 2 つの項目のどちらかに操作を行うだけです。もう一方の項目のリンクは自動的に更新されます。

リンクについての詳細、および実行できる完全なリンク・サンプルについては、samples ディレクトリーにある SLinksICM サンプルを参照してください。

インバウンド・リンクとアウトバウンド・リンク

リンクを使用して特定の DDO (ソースまたはターゲットのいずれか) を参照する場合、このリンクをインバウンド またはアウトバウンド と見なすことができます。個々の DDO からの観点でリンクを考えると、その DDO がターゲットならば、その DDO はリンクをインバウンド・リンクと見なします。一方、そのリンクの相手側にある DDO の観点からは、同じリンクがアウトバウンド・リンクと見なされます。

表 15 の例では、DDO A から DDO B へのリンクはアウトバウンド・リンクで、DDO B への同じリンクはインバウンド・リンクです。

リンク・タイプ名

リンク関係には名前があります。DDO 内で、リンクはリンク・コレクションへグループ化されます。リンク・タイプ名には、システムが定義したものと、独自に定義できるものもあります。ユーザー定義リンク・タイプ名、またはシステム定義リンク・タイプ名は、いくつでも使用できます。システム定義リンク・タイプ名には、次のものがあります。

リンク・タイプ名 **contains**

Java 定数: `DKConstant.DK_ICM_LINKTYPENAME_CONTAINS`

C++ 定数: `DK_ICM_LINKTYPENAME_CONTAINS`

リンク・タイプ名 **DKFolder**

Java 定数 : `DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER`

C++ 定数 : `DK_ICM_LINKTYPENAME_DKFOLDER`

DKFolder リンク・タイプ名が提供され、フォルダー管理のためにシステムによって使用されます。**DKFolder** オブジェクトは、フォルダーの使用を容易にするためのアウトバウンド・リンク・コレクションへの単純化インターフェースです。リンクを定義または削除するために、**DKFolder** リンク・タイプ名を使用しないでください。また、いかなる場合でも **DKFolder** リンク・タイプ名を **DKLinkCollection** と共に使用してはなりません。ただし、リンクを使用してフォルダーを検索するには、**DKFolder** リンク・タイプ名を指定しなければなりません。

リンクについての詳細は、`cmbroot\samples` ディレクトリーの **SLinksICM** サンプルを参照してください。ユーザー定義リンク・タイプの作成については、**SLinkTypeDefinitionCreationICM** サンプルを参照してください。

リンクされた項目の検索

リンクを検索する場合、インバウンド・リンクのみ、アウトバウンド・リンクのみ、または両方のタイプを検索するオプションがあります。インバウンド・リンクとアウトバウンド・リンクを検索するには、検索オプションを設定する必要があります。リンクの検索には、以下のオプションを設定することができます。

Java

- `DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND`
- `DKConstant.DK_CM_CONTENT_LINKS_INBOUND`
- `DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND + DKConstant.DK_CM_CONTENT_LINKS_INBOUND`
- `DKConstant.DK_CM_CONTENT_ITEMTREE`

C++

- `DK_CM_CONTENT_LINKS_OUTBOUND`
- `DK_CM_CONTENT_LINKS_INBOUND`
- `DK_CM_CONTENT_LINKS_OUTBOUND + DK_CM_CONTENT_LINKS_INBOUND`
- `DK_CM_CONTENT_ITEMTREE`

DDO `add()` または `update()` メソッドを呼び出す前に、リンクに関連したすべての項目が永続化されている必要があります。

アクセス制御の使用

Content Manager システム管理クライアントまたは Content Manager API にアクセスして独自の管理プログラムを作成する権限がある場合は、アクセス制御機能を使用して、Content Manager システムの情報へのアクセスを制御することができます。各種アクセス制御 API により、システム全体、密接に関連したシステム操作のセット、または個々の項目に対するアクセスを制御することができます。

アクセス制御 API を使用して行うことのできるタスクには、以下のようなものがあります。

- 特権を作成する。
- アクションのリストをシステム内の情報に関連付ける。
- ライブラリー・サーバーの情報にアクションを実行する許可をユーザーに与える。

特権の作成

特権は、クラス DKPrivilegeICM によって表されます。以下のステップおよびコード例では、新規の特権オブジェクトを作成し、それを永続化し、特権を検索する方法を示します。特権を作成するには、以下のステップを実行します。

Java

1. データ・ストアに接続します。

```
DKDatastoreICM ds = new DKDatastoreICM();
ds.connect("ICMNLSDDB","icmadmin",password,"");
dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();
DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
```
2. DKAuthorizationMgmtICM オブジェクトを検索します。このクラスは、許可管理機能タスクを処理します。

```
DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)
dsAdmin.authorizationMgmt();
```
3. 新規の特権オブジェクトを作成します。

```
DKPrivilegeICM priv = new DKPrivilegeICM(ds);
```
4. 特権オブジェクトに名前を割り当てます。

```
priv.setName("UserPriv");
```
5. 特権オブジェクトに説明を割り当てます。

```
priv.setDescription("This is user-defined privilege");
```
6. 新規特権を許可マネージャー・オブジェクトへ追加します。

```
aclMgmt.add(priv);
```
7. 新規に作成された特権を、許可マネージャー・オブジェクトから検索します。

```
dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
```
8. 特権オブジェクトについての情報を表示します。

```
System.out.println("privilege name = " + aPriv.getName());
System.out.println("privilege description = " + aPriv.getDescription());
```

C++

1. データ・ストア・オブジェクトだけでなく、関連するすべての管理オブジェクトも作成します。

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDb", "icmdmin", "password", "");
//Retrieve the datastore definition object (used to access
//and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore
//administration functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * ac1Mgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. 新規の特権オブジェクトを作成し、そのプロパティを設定します。

```
DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
//Set the name of the privilege object
priv->setName("UserPriv");
//Set the privilege description
priv->setDescription("This is user-defined privilege");
```

3. 許可管理オブジェクトを通じて、データ・ストアに特権を追加します。

```
ac1Mgmt->add(priv);
```

特権セットの作成

特権セットは、クラス `DKPrivilegeSetICM` によって表されます。特権セットの作成を開始する前に、コンテンツ・サーバーへ接続している必要があります。特権セットを作成するには、以下のステップを実行します。

Java

1. 新規の特権セットへ追加するための新規特権を作成（または既存の特権を検索）します。

```
DKPrivilegeICM priv_1 = new DKPrivilegeICM(ds);
priv_1.setName("ItemCheckOut");
DKPrivilegeICM priv_2 = new DKPrivilegeICM(ds);
priv_2.setName("ItemQuery");
DKPrivilegeICM priv_3 = new DKPrivilegeICM(ds);
priv_3.setName("ItemAdd");
```

2. 新規の特権セットを作成します。

```
DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
```

3. 特権セットに名前を割り当てます。

```
privSet1.setName("UserPrivSet");
```

4. 特権セットに説明を割り当てます。

```
privSet1.setDescription("This is a user-defined priv set");
```

5. 特権セットに特権を追加します。

```
privSet1.addPrivilege(priv_1);
privSet1.addPrivilege(priv_2);
privSet1.addPrivilege(priv_3);
```

6. 新規に作成した特権セットを、許可マネージャーへ追加します。

```
AclMgmt.add(privSet1);
```

7. 新規に作成した特権セットについての情報を表示します。

```
DKPrivilegeSetICM aPrivSet = (DKPrivilegeSetICM)
    aclMgmt.retrievePrivilegeSet("UserPrivSet");
System.out.println("privilege set name = " + aPrivSet.getName());
System.out.println("privilege set description=" +
    aPrivSet.getDescription());
dkCollection coll = aPrivSet.listPrivileges();
dkIterator iter = coll.createIterator();
while (iter.more()) {
    DKPrivilegeICM _priv = (DKPrivilegeICM) iter.next();
    System.out.println(" privilege name = " + _priv.getName());
}
```

C++

1. データ・ストア・オブジェクトだけでなく、関連するすべての管理オブジェクトも作成します。

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDb", "icmdmin", "password", "");
//Retrieve the datastore definition object
//(used to access and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore administration
// functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. 3 つの特権を作成し、そのプロパティを設定します。

```
DKPrivilegeICM * priv_1 = new DKPrivilegeICM(ds);
priv_1->setName("ItemCheckOut");
DKPrivilegeICM * priv_2 = new DKPrivilegeICM(ds);
priv_2->setName("ItemQuery");
DKPrivilegeICM * priv_3 = new DKPrivilegeICM(ds);
priv_3->setName("ItemAdd");
```

3. 新規の特権セットを作成し、そのプロパティを設定します。

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("This is a user-defined priv set");
```

4. 新規の特権セットに作成された特権を追加します。

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. 許可管理オブジェクトを使用して、データ・ストアに特権セットを追加します。

```
aclMgmt->add(privSet1);
```

特権セット・プロパティの表示

次の例は、特権セットのプロパティを表示する方法を示しています。

C++

```
//Retrieve a privilege set using its name
DKPrivilegeSetICM * sPrivSet = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("UserPrivSet");
//Display privilege set properties
cout<<"privilege set name = "<< (char *)sPrivSet->getName() << endl;
cout<<"priv set descrip="<< (char *)sPrivSet->getDescription() << endl;
//Retrieve the list of privileges that are a part of this privilege set
dkCollection * coll = sPrivSet->listPrivileges();
dkIterator * iter = coll->createIterator();
while (iter->more())
{
    DKPrivilegeICM* _priv = (DKPrivilegeICM *) (void *) (*iter->next());
    cout<<"  privilege name = "<< (char *)_priv->getName() << endl;
}
delete(iter);
```

アクセス制御リスト (ACL) の定義

Content Manager のアクセス制御モデルは、被制御エンティティに該当します。被制御エンティティとは、保護されたユーザー・データの単位です。被制御エンティティは、個々の項目、項目タイプ、またはライブラリー全体にすることができます。被制御エンティティでの操作は、1 つまたは複数の制御規則で調整されます。ACL はこれらの制御規則用のコンテナです。DKAccessControlListICM クラスは、ACL を表します。Content Manager システム内のすべての被制御エンティティが、ACL へバインドされなければなりません。

次の例は、ACL を定義する方法を示しています。

Java

```
//Define a new DKACLData object.
//A DKACLData class is used to hold ACL related data.
DKACLData aclData1 = new DKACLData();
//set the user group name for the ACL
aclData1.setUserGroupName("ICMADMIN");
//set ACL patron type.
aclData1.setPatronType(DK_CM_USER_KIND_USER);
//Set the privilege set associated with this ACL
aclData1.setPrivilegeSet(privSet_1);
//Create another DKACLData object.
DKACLData aclData2 = new DKACLData();
aclData2.setUserGroupName("ICMPUBLIC");
aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
aclData2.setPrivilegeSet(privSet_2);
//Create a new ACL. A DKAccessControlListICM represents a CM v8.2. ACL
DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
//Assign a new to the newly-created ACL
acl1.setName("UserACL");
//Assign a description to the newly-created ACL
acl1.setDescription("This is a user-defined ACL");
//Add the previously created ACL data objects to the ACL
acl1.addACLData(aclData1);
acl1.addACLData(aclData2);
//Add the newly-created ACL to the authorization manager
aclMgmt.add(acl1);
//Retrieve and display information about the newly created ACL
DKAccessControlListICM acl_1 = (DKAccessControlListICM)
aclMgmt.retrieveAccessControlList("UserACL");
System.out.println("ACL name = " + acl_1.getName());
System.out.println("    desc = " + acl_1.getDescription());
dkCollection coll = acl_1.listACLData();
dkIterator iter = coll.createIterator();
while (iter.more()) {
    DKACLData aclData = (DKACLData) iter.next();
    DKPrivilegeSetICM _privSet = (DKPrivilegeSetICM) aclData.getPrivilegeSet();
    System.out.println("  PrivSet name = " + _privSet.getName());
    System.out.println("  PrivSet desc = " + _privSet.getDescription());
    String usrGrpName = aclData.getUserGroupName();
    System.out.println("    UserGroupName = " + usrGrpName);
    short patronType = aclData.getPatronType();
    System.out.println("    Patron type = " + patronType);
}
```

C++

1. 新規 DKACLDData オブジェクトを定義します。それぞれの DKACLDData オブジェクトは、ACL 関連データを保持するために使用されます。

```
DKACLDData * aclData1 = new DKACLDData();
// set the name of the user group to be associated with this ACL
aclData1->setUserGroupName("ICMADMIN");
// Set the ACL patron type. Can be one of 3 possible
//values:DK_CM_USER_KIND_USER or DK_CM_USER_KIND_GROUP or
//DK_CM_USER_KIND_PUBLIC.
aclData1->setPatronType(DK_CM_USER_KIND_USER);
// Set the privilege set associated with the ACL.
DKPrivilegeSetICM * privSet_1 = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("UserPrivSet");
aclData1->setPrivilegeSet(privSet_1);
```

2. 他の DKACLDdata オブジェクトを作成します。

```
DKACLDData * aclData2 = new DKACLDData();
aclData2->setUserGroupName("ICMPUBLIC");
aclData2->setPatronType(DK_CM_USER_KIND_GROUP);
DKPrivilegeSetICM * privSet_2 = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("PublicPrivSet");
aclData2->setPrivilegeSet(privSet_2);
```

3. 新規の ACL を作成します。この新規 ACL にプロパティ値を設定します。

```
DKAccessControlListICM * acl1 = new DKAccessControlListICM(ds);
//Assign a new name to the newly-created ACL.
acl1->setName("UserACL");
// Assign a description to the newly-created ACL.
acl1->setDescription("This is a user-defined ACL");
```

4. ステップ 3 で作成した ACL データ・オブジェクトを ACL に追加します。

```
acl1->addACLData(aclData1);
acl1->addACLData(aclData2);
```

5. ステップ 3 で作成した ACL を許可マネージャーに追加します。

```
aclMgmt->add(acl1);
```

完全なサンプル・プログラムは、samples ディレクトリーに入っています。

ACL 情報の検索と表示

ACL 情報を検索し表示するために、以下のステップを実行してください。

1. ACL 名を使用して、許可管理オブジェクトから ACL を検索します。

C++

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
    retrieveAccessControlList("UserACL");
cout<<"ACL name = "<< (char *)acl_1->getName() << endl;
cout<<"    desc = "<< (char *)acl_1->getDescription() << endl;
```

2. ACL に関連した ACL データを検索します。

C++

```
dkCollection * coll = acl_1->listACLData();
dkIterator * iter = coll->createIterator();
char szpatronType[12] = {0x00};
while (iter->more())
{
    DKACLData * aclData = (DKACLData *) (void *) (*iter->next());
    DKPrivilegeSetICM * _privSet = (DKPrivilegeSetICM *) aclData->
        getPrivilegeSet();
    cout<<"privSet name = "<< (char *) _privSet->getName() << endl;
    cout<<"privSet desc="<< (char *) _privSet->getDescription() << endl;
    DKString usrGrpName = aclData->getUserGroupName();
    cout<<"    UserGroupName = "<< (char *) usrGrpName << endl;
    short patronType = aclData->getPatronType();
    sprintf(szpatronType, "%d", patronType);
    cout<<"    Patron type    = "<< szpatronType << endl;
}
delete(iter);
```

項目タイプへの ACL の割り当て

ACL が作成されると、それを特定の項目タイプに関連付けることができます。以下のステップに従って、ACL を項目タイプへ割り当てます。

1. 項目タイプを新規にセットアップします。

Java

```
DKItemTypeDefICM it = new DKItemTypeDefICM(dsICM);
//Assign a name to this item type
it.setName("TextResource1");
//Assign a description to this item type
it.setDescription("CMv8.2 Text Resource Item Type.");
//Assign an ACL code to this item type
it.setItemTypeACLCode((int)DK_ICM_ITEMACL_BIND_AT_ITEM);
....
it.add(); // make the item type persistent
...
```

C++

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
// Assign a name to this item type.
itemType->setName("TextResource1");
// Assign a description to this item type.
itemType->setDescription("CMv8.2 Text Resource Item Type.");
```

2. ACL に関連した ACL データを検索します。

Java

```
int itemTypeACLCode = it.getItemTypeACLCode();
// By setting the item type's ACL flag to TRUE(1) we confirm that ACL
// binding is at the item type level. By setting the item type's ACL
// flag was set to FALSE(0), we would be saying that the ACL binding
// is not at the item type level
it.setItemLevelACLFlag(1); // - true
//Determine whether the ACL binding is at the item type level or not
int itemTypeACLFlag = it.getItemLevelACLFlag();
```

C++

```
itemType->setItemTypeACLCode((long) 1);
// By setting the item type's ACL flag to TRUE (1),
//we confirm that ACL binding is at the item type level.
//By setting the item type's ACL flag to FALSE(0),
// we would be saying that the ACL binding is not at the item type level.
itemType->setItemLevelACLFlag((short)1);
itemType->add();
// make the item type persistent.
```

完全なサンプル・プログラムは、samples ディレクトリーに入っています。

項目への ACL の割り当て

項目レベル・アクセス制御を使用可能にするには、ACL を項目へバインドさせる必要があります。これを行うには、以下のコード・フラグメントで示されるように、DDO 上で add() メソッドを使用して項目を作成する必要があります。以下のコード・フラグメントでは、既にコンテンツ・サーバーへの接続があり ACL を作成していることが前提です。完全なプログラムは、samples ディレクトリーに入っています。

Java

```
//Create a new DDO
DKDDO ddoItem =dsICM.createDDO("myItemType",DKConstantICM.DK_CM_ITEM);
//create a new ACL (access contrl list)
DKAccessControlListICM acl1 =new DKAccessControlListICM(ds);
//Assign a name to the ACL
acl1.setName("MyACL");
//Add a new property to the DDO.This property will be
//called DK_ICM_PROPERTY_ACL
int propId =ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
//Set the previously created (or retrieved)ACL as the value of this property
ddoItem.setProperty(propId,"MyACL");
//Persist the DDO into the data store
ddoItem.add();
```

C++

1. 既存の項目タイプに基づき新規の項目 (DDO) を作成します。
`DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);`
2. 新規の ACL (アクセス制御リスト) を作成し、ACL のプロパティを設定します。
`DKAccessControlListICM * acl1 = new DKAccessControlListICM(dsICM);`
3. ACL に名前を割り当てます。
`acl1->setName("MyACL");`
4. DDO に新規のプロパティを追加します。このプロパティは `DK_ICM_PROPERTY_ACL` と呼ばれることになります。
`ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);`
5. このプロパティの値として、上記で作成した ACL を設定します。ユーザーは既存の ACL を検索し、それをこの項目の ACL として使用することも選択できる点に注意してください。
`ddoItem->setProperty(propId, "MyACL");`
6. データ・ストアに DDO を置きます。
`ddoItem->add();`

照会言語について

Content Manager 8.2 では、強力な XML ベースの照会言語が使用できます。アプリケーションが、Content Manager システムに保管された項目を検索する場合、基盤となるエンジンは、照会に基づく検索処理を行います。Content Manager の階層データ・モデルを効率よく全検索するためには、Content Manager 照会言語を使用します。この照会言語には、以下の利点があります。

- 全データ・モデルをサポートする。
- 特定のバージョンや最新バージョンの検索など、バージョンをサポートする。
- コンポーネント・タイプ・ビュー階層内、リンクされた項目間、および参照間の検索が可能。
- パラメトリック検索およびテキスト検索の組み合わせ。
- SORTBY 機能の提供。
- Content Manager アクセス制御の適用。
- W3C XML Query 作業草案のサブセットである XQuery Path Expressions (XQPE) 標準に準拠。
- ハイパフォーマンス検索の実行。

Content Manager の照会言語は、独自開発の言語ではなく、W3C XML Query 作業草案のサブセットである、XQuery Path Expressions (XQPE) に準拠した、XML ベースの照会言語です。この照会言語は階層項目タイプを検索して、項目をす早く容易に見つけます。照会を作成する前に、照会言語の概念、構文および文法について理解する必要があります。

すべての照会が、コンポーネント・タイプ・ビューで実行されます。したがって、照会ストリング内のルート・コンポーネントまたは子コンポーネント用に使用する名前が、システムによって作成されるベース・コンポーネント・タイプ・ビューの名前 (Journal、Journal_Article など) またはユーザー定義のコンポーネント・タイプ・ビューの名前 (My_Journal、My_Book_Section など) のいずれかになります。システム管理では、コンポーネント・タイプ・ビューはコンポーネント・タイプ・サブセットと呼ばれます。

文書、フォルダー、オブジェクトなどの照会を実行依頼すると、要求は Content Manager 照会エンジンに送信され、処理後に適切な SQL へと変換されます。その後ライブラリーは、セキュリティー検査 (ACL) を適用し、照会を実行します。

Content Manager サーバーの照会

Content Manager ライブラリー・サーバーを照会するには、以下のようになります。

1. 検索条件を表す照会ストリングを作成してコンテンツ・サーバーを照会します。
2. 照会ストリングおよび照会オプションを使用して、`evaluate` メソッド、`execute` メソッド、または `executeWithCallback` メソッドを呼び出します。
3. `DKResults` オブジェクト、`dkResultSetCursor` オブジェクト、または `dkCallback` オブジェクトを通じて結果を受け取ります。

照会 API が、照会処理タスク (照会の準備および実行、照会の実行状況のモニター、結果の検索など) を実行します。

論理的に、照会ストリングにはパラメトリック、テキスト、および結合の 3 つの照会タイプのうちの 1 つを含むことができます。パラメトリック照会は、等式および比較などの条件を使用します。テキスト照会は、検索をより強力にするために、テキスト検索機能を使用します。結合照会は、テキスト条件とパラメトリック条件の両方を組み合わせたものです。

照会を実行するには、`evaluate` メソッド、`execute` メソッド、または `executeWithCallback` メソッドのうちのいずれかを使用できます。評価メソッドは、すべての結果を DDO のコレクションとして戻します。このメソッドは、結果セットが小さい場合に便利です。 `execute` メソッドは `dkResultSetCursor` オブジェクトを戻します。これには、次の特性があります。

- `dkResultSetCursor` は、コンテンツ・サーバー・カーソルのように機能します。
- これが戻す DDO は、ユーザーが要求するとブロック単位で検索されるため、これは大きな結果セットにも使用できます。
- `dkResultSetCursor` を使用して、`close` および `open` メソッドを呼び出すことによって、照会を再実行することができます。
- `dkResultSetCursor` を使用して、結果セット・カーソルの現在位置の削除および更新が可能です。

`executeWithCallback` メソッドは、照会を非同期に実行して、結果を指定のコールバック・オブジェクトにブロック単位で送ります。これにより `executeWithCallback` メソッドは、メインの実行スレッドを照会結果の検索タスクから解放します。照会メソッドについて詳しくは、SSearchICM サンプルを参照してください。

DKDatastoreICM クラスのメソッド `evaluate`、`execute`、および `executeWithCallback` については、アプリケーション・プログラミング・リファレンスを参照してください。

Content Manager データ・モデルへの照会言語の適用

照会言語を理解する上で、ライブラリー・サーバーを概念上、XML 文書と見なすことができます。XML 文書による類推は、照会言語の説明のためだけのものです。したがって、ライブラリー・サーバーの XML 表現は単なる仮想表現であり、ライブラリー・サーバーの項目は XML エレメントではなく、照会を実行しても XML エレメントは取得されないことに注意してください。ライブラリー・サーバーの XML 表現において、Content Manager のデータ・モデル・エレメントは、次のように表されます。

項目 一般に、各 CM 項目は、ネストされた XML エレメントによって表され、最上位の XML エレメントがルート・コンポーネントを表し、ネストされた XML エレメントが子のコンポーネントを表します。このように、XML エレメントのネスティングは、コンポーネント階層を表します。

ルート・コンポーネント

ルート・コンポーネントは、XML エレメントの最初のレベルによって表されます。ルート・コンポーネントは、XML 属性の ID ITEMID、STRING COMPONENTID、INTEGER VERSIONID、INTEGER SEMANTICTYPE、およびコンポーネントのその他のユーザー定義属性を持ちます。ライブラリー・サーバーでは、ITEMID は固有です。

子コンポーネント

子コンポーネントは、ネストされた XML エレメントによって表され、属性 STRING ITEMID、STRING COMPONENTID、INTEGER VERSIONID、およびコンポーネントのその他のユーザー定義属性を持ちます。

COMPONENTID のみ、Content Manager コンポーネント内で固有であることに注意してください。ITEMID および VERSIONID は、子のルート・コンポーネントの ITEMID および VERSIONID と全く同じです。

ユーザー定義属性

各ユーザー定義属性は、これを含むコンポーネントを表す XML エレメント内のネストされた XML 属性として表されます。

リンク インバウンド・リンクおよびアウトバウンド・リンクは、CM データ・モデル内の項目自体に含まれていませんが（「リンク (links)」テーブルに別々に保管されている）、照会目的の場合は、それらを、項目を表している XML エレメントに含まれていると概念的に考えると大変便利です。これによって、アプリケーションで照会内に明示的に結合を書き込む必要がなくなります。リンクは、項目間の 1 対多の関係を表しています。この関係は、ルート・コンポーネント間のみであることに注意してください（リンクは、子コンポーネント内を始点または終点にすることはできません）。

項目が始点となるリンクは、IDREF LINKITEMREF、IDREF TARGETITEMREF、および STRING LINKTYPE の以下の属性を持つ、<OUTBOUNDLINK> XML エレメントによって表されます。

LINKITEMREF は、リンクのメタデータを含む項目への参照です。

TARGETITEMREF は、リンクによって示される項目への参照です。

LINKTYPE は、リンクのタイプです。同様に、項目が終点となるリンクは、IDREF LINKITEMREF、IDREF SOURCEITEMREF、および STRING LINKTYPE の以下の属性を持つ、<INBOUNDLINK> XML エlementによって表されます。SOURCEITEMREF は、リンクの始点となる項目への参照です。リンクの意味体系について詳しくは、SLinksICM サンプルと SSearchICM サンプルを参照してください。

参照 (参照属性)

参照属性は、タイプ IDREF の XML 属性によって表されます。参照は、項目またはコンポーネントと、他の項目間の 1 対 1 関係を表しています。したがって、参照のターゲットとして可能なのは、子コンポーネントではなく、ルート・コンポーネントのみです。ただし、参照属性の始点は、ルート・コンポーネントまたは子コンポーネントのいずれからでも可能です。

参照属性は、システム定義 (SYSREFERENCEATTRS)、またはユーザー定義 (以下の照会例では PublicationRef) のいずれかが可能です。参照は、両方向に全探索することができます。

逆方向の参照全探索は上記の逆方向のリンク全探索と同様の方法で実行されます。参照されている項目は、XML 属性 REFERENCER (IDREF タイプ) が含まれる XML Element REFERENCEDBY を持っていると考えられます。REFERENCER は、この項目を参照するコンポーネントを指し示します。これは、逆方向のリンク全探索を行うために SOURCEITEMREF 属性を持つ INBOUNDLINK Element と同じです。

また、意味の削除 (制限、カスケード、NULL 設定、またはアクションなし) もサポートしています。参照属性について詳しくは、SReferenceAttrDefCreationICM サンプルと SSearchICM サンプルを参照してください。

文書 Content Manager が DKFolder として提供しているフォルダー機能は、"DKFolder" (DK_ICM_LINKTYPENAME_DKFOLDER) リンク・タイプの設定済みリンクとして保管されています。それぞれのフォルダーとコンテンツの関係は、フォルダーからのアウトバウンド・リンクと、コンテンツへのインバウンド・リンクとしてモデル化されています。これにより、フォルダーは各リンクのソース、コンテンツは各リンクのターゲットになります。フォルダーを検索および全探索するには、リンクの検索と全探索の意味体系に従って行います。詳細については、SFolderICM および SSearchICM サンプルを参照してください。

パラメトリック検索について

多くの場合、項目は、選択した属性に対して検索を開始することにより検索されます。単一照会では、コンテンツ・サーバー内の項目のシステム定義およびユーザー定義の両方の属性を検査できます。単純な検索条件では、属性名、演算子、および値が文節に結合されます。Content Manager では、パラメトリック検索を実行する多くの比較演算子が提供されます。それらの演算子は、以下のとおりです。

"="

"< "

"<="

">"
">="

"!="

"LIKE"

"NOT LIKE"

"BETWEEN"

"NOT BETWEEN"

"IS NULL"

"IS NOT NULL"

単純な検索条件をブール演算子 AND、OR、および NOT を使用して文節に組み込むことにより、複雑な検索条件を指定することができます。詳細については、照会例を参照してください。

テキスト検索について

Content Manager は、DB2 Universal Database Net Search Extender (NSE) (CM 8.1 では Text Information Extender (TIE) と呼ばれていました) を使用することにより、コンポーネントにテキストを含む属性のテキスト検索と、オブジェクトのテキスト検索の、2 種類のテキスト検索を行います。この 2 種類のテキスト検索の主な違いは、コンテンツの保管方法にあります。属性をテキスト検索可能として定義する場合、その属性の列に含まれるテキストを検索可能とするよう指示します。属性 (列) テキストを検索可能とするため、NSE はテキスト索引を作成します。テキスト索引は、検索対象のテキストに関する情報を含みます。この情報を使用することにより、テキスト検索が効率よく行われます。例えばシステム管理者の Fred が、子コンポーネント・タイプ・ビュー Journal_Article を持つ項目タイプ「ジャーナル (Journal)」を作成する際、テキスト検索を可能にするとします。Journal_Article の属性の 1 つに「タイトル (Title)」があり、Fred はこれをテキスト検索可能にします。引き受け人である Lily が、ワード "Java" を含む「タイトル (Title)」を検索すると、"Java" にヒットする「タイトル (Title)」のテキスト索引が検索されます。

CM 8.1 で使用されている TIE については、DB2 Universal Database Text Information Extender (TIE) の資料を参照してください。CM 8.2 で使用されている NSE については、DB2 Universal Database Net Search Extender (NSE) の資料を参照してください。ICM 照会言語でのテキスト検索に関する説明中では、NSE と TIE のどちらを使用しても構いません。ICM 照会言語の観点からは、NSE と TIE の間でテキスト検索の構文や機能の違いはありません。

オブジェクト・コンテンツの検索

オブジェクトのコンテンツの検索は、やや異なる動作をします。列を直接索引付けする代わりに、システムはリソース・マネージャーにおけるオブジェクトの位置への参照を使用します。NSE は、テキスト索引を作成する際、参照を使用してコンテンツを取り出します。検索を行うエンド・ユーザーは、リソース・マネージャーに保管されたオブジェクトを検索する際に違いに気がつくことはありません。ただし、システム管理者は、検索メカニズムがリソース・マネージャーのコンテンツを見つけることができるよう、テキスト・リソースの項目タイプ・ビューを設定する

必要があります。テキスト検索は、テキスト検索目的のためにリソース・マネージャー上に保管されたコンテンツを参照する、リソース項目タイプの属性 "TIEREF" 上で実行されます。

文書の検索

テキスト検索は、文書パーツのコンテンツ上で実行できます。仮想コンポーネント・タイプ・ビュー "ICMPARTS" は、システム内の全文書の子として照会でサポートされています。"ICMPARTS" コンポーネント・タイプ・ビューの下の "TIEREF" 属性は、テキスト検索用の文書のすべてのテキスト検索可能パーツのコンテンツを参照します。この機能の使用法の詳細については、照会例を参照してください。

ユーザー定義属性をテキスト検索可能にする

DKAttrDeflCM API および DKItemTypeDeflCM API を使用することにより、ユーザー定義属性をテキスト検索可能にすることができます。作成されたテキスト索引のデフォルト・プロパティは、DKTextIndexDeflCM クラスを使用して変更することができます。API について詳しくは、「オンライン API 解説書」、または SItemCreationICM サンプルを参照してください。

テキスト検索構文について

テキスト検索照会は、基本テキスト検索構文または拡張テキスト検索構文のいずれかを使用して実行することができます。

基本テキスト検索

テキスト検索の大多数は数個のワードを列挙するだけで実行されるため、基本 (簡易) テキスト検索構文は、特に、この最も一般的なケースをユーザーにとって容易なものにするために設計されました。構文では、引用句の使用だけでなく、"+" および "-" の使用も可能です。簡易テキスト検索は、"contains-text-basic" 関数および "score-basic" 関数を使用して実行されます。"contains-text-basic" 関数は、属性内部、またはリソースや文書のコンテンツ内部の検索に使用されます。"score-basic" 関数は、"contains-text-basic" 関数と同じ構文を使用し、テキスト検索結果のランクに基づいて結果をソートするために使用されます。"contains-text-basic" 関数は、真の場合は 1、偽の場合は 0 にしてください。これらの関数の使用法については、照会例を参照してください。

基本テキスト検索構文についての追加情報には、以下のものが含まれます。

- 大文字小文字を区別しないテキスト検索の実行 (デフォルトの拡張構文と同様)。大文字小文字の区別をする検索オプションについては、NSE の資料を参照してください。
- 引用内の用語は句であることが前提。
- + (プラス) - (マイナス) の使用
 - + (プラス) = 文書はこのワードを含む必要がある。
 - - (マイナス) = 文書はこのワードを含んではいけない。
 - + または - が指定されていない場合、照会エンジンは、ワードをテキストへ突き合わせるアルゴリズムを使用する。
- ブール演算子 (AND、OR、NOT) は無効で無視される。

- 基本構文内の括弧はサポートされない。
- 有効なワイルドカード
 - ? (疑問符) = 単一文字を表す
 - * (アスタリスク) = 任意数の任意文字を表す

基本テキスト検索について詳しくは、SSearchICM サンプルを参照してください。

拡張テキスト検索

拡張テキスト検索構文を使用すると、ユーザーはテキスト検索用に、より複雑な条件を指定することができます。このテキスト検索は、NSE テキスト検索構文を使用し、接近検索およびファジー検索などの強力な機能を可能にします。拡張テキスト検索構文は、基本テキスト検索に "contains-text-basic" 関数および "score-basic" 関数を使用されるのと同様に、"contains-text" 関数および "score" 関数を使用します。拡張関数へ提供されるストリングは、二重引用符を単一引用符へ変更する (またはその逆) 場合を除き、NSE 構文です。例えば、NSE の CONTAINS (description, 'IBM')=1 条件は、CM 照会言語では contains-text(@description, " 'IBM' ")=1 になります。これは、最小限のエスケープ文字の使用で照会が作成できるという簡易性をサポートするために実行する必要があります。"contains-text" 関数は、真の場合は 1、偽の場合は 0 にしてください。拡張テキスト検索の詳細については、照会例を参照してください。

拡張テキスト検索について詳しくは、SSearchICM サンプルを参照してください。

パラメトリック検索とテキスト検索の組み合わせ

検索は実際、項目またはコンポーネントの任意の部分、項目またはコンポーネント内のテキスト、またはリソース・コンテンツ内のテキストに基づいて実行できます。検索は、以下のいずれかのタイプにすることができます。

パラメトリック・サーチ

項目およびコンポーネント・プロパティ、属性、参照、リンク、フォルダー・コンテンツなどに基づく検索。

テキスト検索

テキスト内の検索。

結合検索

パラメトリック検索およびテキスト検索の両方を使用する検索。

パラメトリック検索とテキスト検索を組み合わせた検索を作成するには、次の手順で行います。

Java

1. ICM データ・ストアを作成し、これに接続します。
2. 複合照会ストリングを生成します。

```
String queryString =  
    "//Journal_Article [Journal_Author/@LastName = ¥'Richardt¥'" +  
    " AND contains-text (@Text, ¥" 'Java' & 'XML' ¥")=1]";
```

3. デフォルトと異なる検索オプションを使用する場合は、DKNVPair 配列を使用してオプションをパッケージします。

```
DKNVPair parms[] = new DKNVPair[3];  
String strMax = "5";  
parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, strMax);  
parms[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,  
    DKConstant.DK_CM_CONTENT_ATTRONLY |  
    DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);  
parms[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
```

4. 複合照会を、evaluate、execute、および executeWithCallback の 3 つの方法のいずれかで実行します。

```
DKResults resultsCollection =  
    (DKResults)dsICM.evaluate(queryString,  
    DKConstant.DK_CM_XQPE_QL_TYPE, parms);
```

5. 結果を処理します。結果の処理プロシージャは、使用した実行メソッドによって異なります。

完全なサンプルと追加資料については、CMBROOT¥Samples¥java¥icm にある SSearchICM API 実習サンプルを参照してください。

C++

1. 検索オプションを指定します。オプションの配列には常に終了エレメントが必要です。例えば、2つのオプションを指定したい場合は、オプションの配列には3つの要素が必要です。

```
DKNVPair * parms = new DKNVPair[3];
DKNVPair * pparm = NULL;
DKString strMax = "5";
DKAny * anyNull = new DKAny();
//Allow a maximum of 5 items to be returned from the search
pparm = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[0] = *pparm;
delete pparm;
//Specify what content is to be retrieved
pparm = new DKNVPair((long)DK_CM_PARM_RETRIEVE,
    DK_CM_CONTENT_ATTRONLY | DK_CM_CONTENT_LINKS_OUTBOUND);
parms[1] = *pparm;
delete pparm;
pparm = new DKNVPair(DK_CM_PARM_END, *anyNull);
parms[2] = *pparm;
delete pparm;
```

2. 検索を実行します。検索を実行する方法が3つあります。

evaluate

すべての結果を集合として戻します。小さなセットの場合に適しています。

execute

呼び出し元が結果に繰り返し使用する、結果のセット・カーソルを戻します。

executeWithCallback

結果のセットに繰り返されるスレッドを作成し、結果のそれぞれのブロック用にコールバック・オブジェクトを呼び出します。呼び出し元は結果を入手するためにコールバック・オブジェクトを使用します。

以下の例では、5つの結果のみが必要なので、DKDatastoreICM.evaluate メソッドが使用されます。

```
DKResults * resultsCollection = (DKResults *) (dkCollection *)
    dsICM->evaluate(queryString, DK_CM_XQPE_QL_TYPE, parms);
```

3. 検索の結果が表示されます。

```
// Create an iterator to go through Results collection.
dkIterator* iter = resultsCollection->createIterator();

cout << "Results:" << endl;
cout << "    - Total: " << results->cardinality() << endl;

while(iter->more())
{
    //Each element in the returned array is an item (DDO)
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    cout << "    - Item ID: " << ((DKPidICM*) ddo->getPidObject())
        ->getItemId() << " (" << ((DKPidICM*) ddo->getPidObject())
        ->getObjectType() << ")" << endl;
}
```

4. クリーンアップします。

```
delete(iter);
delete[] parms;
....
```

完全なサンプルと追加資料については、CMBROOT¥Samples¥cpp¥icm にある SSearchICM API 実習サンプルを参照してください。

照会の例

照会言語の理解を深め照会を記述するため、このセクションでは以下の内容を説明します。

- サンプル・データ・モデル。
- データ・モデルの XML 文書表現。
- サンプル照会。
- 照会言語の文法。

以下のセクションに示すサンプル照会は、211 ページの図 13 で概説する照会例のデータ・モデルを基にしています。サンプル照会を検討する際は、データ・モデルの図を参考にしてください。

代替のデータ・モデルを基にしたその他の照会構文と例については、SSearchICM サンプルを参照してください。

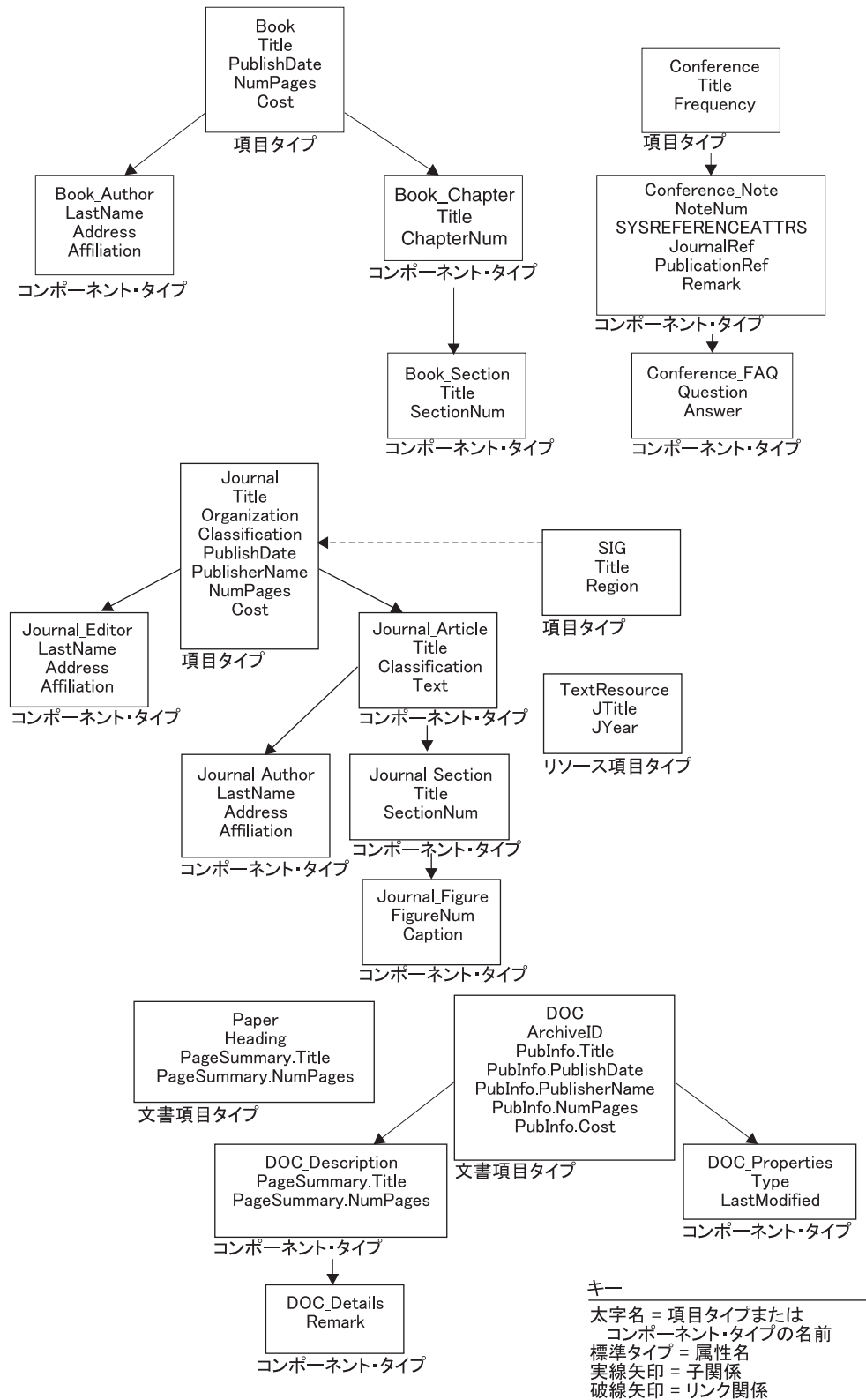


図 13. 照会例のデータ・モデル

以下の XML 文書は、図 13 のデータ・モデルを表したものです。

照会例のデータ・モデルの XML 表記:

```

<Journal (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
          INTEGER SEMANTICTYPE, Title, Organization, Classification,
          PublishDate, PublisherName, NumPages, Cost)>
  <Journal_Editor (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  LastName, Address, Affiliation)>
</Journal_Editor>
... (repeating <Journal_Editor>)

  <Journal_Article (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  Title, Classification, Text)>
    <Journal_Section (STRING ITEMID, STRING COMPONENTID,
                     INTEGER VERSIONID, Title, SectionNum)>
      <Journal_Figure (STRING ITEMID, STRING COMPONENTID,
                      INTEGER VERSIONID, FigureNum, Caption)>
</Journal_Figure>
      ... (repeating <Journal_Figure>)
    </Journal_Section>
    ... (repeating <Journal_Section>)

    <Journal_Author (STRING ITEMID, STRING COMPONENTID,
                    INTEGER VERSIONID, LastName, Address, Affiliation)>
</Journal_Author>
    ... (repeating <Journal_Author>)
  </Journal_Article>
  ... (repeating <Journal_Article>)

  <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                STRING LINKTYPE) >
</OUTBOUNDLINK>
  ... (repeating <OUTBOUNDLINK>)

  <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                STRING LINKTYPE)>
</INBOUNDLINK>
  ... (repeating <INBOUNDLINK>)

  <REFERENCEDBY (IDREF REFERENCER)>
</REFERENCEDBY>
  ... (repeating <REFERENCEDBY>)
</Journal>
...(repeating <Journal>)

<Book (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER
       SEMANTICTYPE, Title, PublishDate, NumPages, Cost)>
  <Book_Author (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
               LastName, Address, Affiliation)>
</Book_Author>
  ... (repeating <Book_Author>)

  <Book_Chapter (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                Title, ChapterNum)>
    <Book_Section (STRING ITEMID, STRING COMPONENTID,
                  INTEGER VERSIONID, Title, SectionNum)>
</Book_Section>
    ... (repeating <Book_Section>)
  </Book_Chapter>
  ... (repeating <Book_Chapter>)

  <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                STRING LINKTYPE) >
</OUTBOUNDLINK>
  ... (repeating <OUTBOUNDLINK>)

  <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                STRING LINKTYPE)>
</INBOUNDLINK>
  ... (repeating <INBOUNDLINK>)

```



```

        <REFERENCEDBY (IDREF REFERENCER)>
        </REFERENCEDBY>
        ... (repeating <REFERENCEDBY>)
    </Book>
    ... (repeating <Book>)

<SIG (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
      INTEGER SEMANTICTYPE, Title, Region)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                  STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                 STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</SIG>
... (repeating <SIG>)

<TextResource (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
               INTEGER SEMANTICTYPE, JTitle, JYear)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                  STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                 STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</TextResource>
... (repeating <TextResource>)

```

照会の例

このセクションにあるサンプル照会は、サンプル・データ・モデル (211 ページの図 13)、およびサンプル XML 文書 (211 ページ) に基づいています。照会例の理解に役立ついくつかのヒントを以下に示します。

- ディレクトリー構造をたどるように、照会ストリングをたどります。
- `"/"` 単一スラッシュは、直接の子関係を示します。
- `"//"` ダブルスラッシュは、子関係または子孫関係を示します。
- `"."` (ドット) は、階層内の現行コンポーネントを示します。
- `".."` (ドット-ドット) は、現行コンポーネントの親を示します。
- `"@"` (アットマーク) は、属性を示します。
- `"[]"` (大括弧) は、条件ステートメントまたはリストを示します。
- `"=>"` (DEREFERENCE 演算子) は、リンク・アクションまたは参照アクションを示します。

- 照会結果は、コンポーネントでなければなりません (例えば、属性がパス内で最後になることはできません)。

その他の例と資料が、SSearchICM サンプルにあります。

例 1: コンポーネントへのアクセス

この照会は、すべてのジャーナルを検索します。

```
/Journal
```

説明: 『/』は、XML 文書の暗黙のルートから始まります。この例では、これはライブラリー・サーバー全体となります。各項目タイプは、このルート以下のエレメントです。LS.xml が上記のモデル全体を含む XML 文書である場合、明示的文書ルートは文書 (LS.xml) となります。

例 2: 属性へのアクセス

この照会は、合計 50 ページあるすべてのジャーナル記事を検索します。

```
/Journal[@NumPages=50]
```

説明: 述部 @NumPages = 50 は、Content Manager 属性『NumPages』が 50 であるすべてのジャーナルについて真を評価します。

例 3: 複数の項目タイプ

この照会は、著者の 1 人として『Williams』を持ち、『XML』で始まるセクション・タイトルを持つすべてのブックまたはジャーナルを検索します。

```
(/Book | /Journal)
[(./Journal_Author/@LastName = "Williams"
OR ./Book_Author/@LastName = "Williams")
AND (./Book_Section/@Title LIKE "XML%"
OR ./Journal_Section/@Title LIKE "XML%")]
```

OR

```
(/Book[./Book_Author/@LastName = "Williams"
AND ./Book_Section/@Title LIKE "XML%"])
| (/Journal[./Journal_Author/@LastName = "Williams"
AND ./Journal_Section/@Title LIKE "XML%"])
```

説明: 上記 2 つの照会では、同じ結果が出ます。『./Journal_Author』は、コンポーネント Journal_Author がパス内の現行コンポーネント (最初のケースの場合、Book または Journal) の直下で検出、または階層内のどこか深い部分で検出されることを意味しています。LIKE 演算子は、ワイルドカード文字 (このケースでは『%』) と結合して使用されることに注意してください。

説明 4: 条件における算術演算

この照会は、45 から 200 までのページ番号を持つすべてのジャーナルを検索します。

```
/Journal[@NumPages BETWEEN 49-4 AND 2*100]
```

説明: 算術演算を実行して、BETWEEN 演算子で使用する結果値を計算することに注意してください。

例 5: 前方方向のリンク全探索

この照会は、『Williams』によって編集され、SIG 内に含まれ、タイトルが『SIGMOD』であるジャーナル内のすべての記事を検索します。

```

/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK
  [@LINKTYPE = "contains"]/@TARGETITEMREF =>
  Journal[Journal_Editor/@LastName = "Williams"]
/Journal_Article

```

説明: これは、前方方向にリンクを追跡する例です。仮想 XML コンポーネント OUTBOUNDLINK およびその属性 TARGETITEMREF は、すべてのジャーナルを、そして最後に基礎をなす Journal_Articles を全探索するために使用されます。パスの最後のコンポーネントは、照会結果として戻されるものです。項目の特定タイプ (この例では Journal) への特定リンク・タイプ (この例では『contains』) のみを全探索することによって、結果を制限することができます。ライブラリー・サーバーの概念的な XML 表記は、インバウンドおよびアウトバウンドのリンクを項目の一部として見るため、アプリケーションが明示的に結合を書き込む必要がなくなるように、非参照演算子を使用することができます。

例 6: 逆方向のリンク全探索

この照会は、著者『Nelson』の記事が含まれる、5 ドル未満のジャーナルを持つタイプのすべての項目を検索します。

```

/Journal[@Cost < 5
AND ../Journal_Author/@LastName = "Nelson"]
/INBOUNDLINK[@LINKTYPE = "contains"]
/@SOURCEITEMREF => *

```

説明: これは、逆方向にリンクを追跡する例です。『=>』の後のワイルドカード『*』により、すべてのタイプの項目が結果として戻されることが保証されます。

例 7: テキスト検索 (contains-text 関数および score 関数)

この照会は、テキスト『Java』およびテキスト『XML』を含む、著者『Richardt』のジャーナル記事を検索します。結果は、テキスト検索スコアの順に並びます。

```

//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))

```

説明: これは、contains-text 関数を使用してテキスト検索を実行する例です。この機能がサポートする構文については、DB2 Net Search Engine (NSE) の資料を参照してください。contains-text 関数は、真の場合は 1、偽の場合は 0 にする必要があります。score 関数は、NSE によって戻されるランク情報を使用します。このケースでは、結果として戻されるジャーナル記事を、SORTBY を使用してソートするために使用されます。

例 8: テキスト検索 (contains-text 関数および属性ソート関数)

この照会は、タイトルに『Design』または『Index』のいずれかのワードを持つすべてのジャーナルを検索し、タイトルの降順に結果をソートします。

```

/Journal
[Journal_Article[contains-text(@Title, " 'Design' | 'Index' ")=1]]
SORTBY (@Title DESCENDING)

```

説明: これは、contains-text 関数を使用してテキスト検索を実行するもう 1 つの例です。このケースでのソート機能は、『Title』属性で DESCENDING 演算子を使用します。SORTBY のデフォルトは、ASCENDING です。

例 9: テキスト検索 (contains-text-basic 関数および score-basic 関数)

この照会は、簡易 (基本) テキスト検索構文を使用して、テキスト『Java』およびテキスト『JDK 1.3』を含むが、テキスト『XML』を含まないすべてのジャーナル記事を検索し、結果をテキスト検索スコアによってソートします。

```
//Journal_Article
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' "))
```

説明: これは、簡易テキスト検索構文を使用してテキスト検索を実行する例です。属性『Title』内に含まれるべきワードまたは句を示すには『+』を使用し、同様に、他のワードまたは句を除外するには『-』を使用します。score-basic 関数は、直前の例の score 関数と同様に機能しますが、簡易構文を使用します。

例 10: リソース項目上でのテキスト検索

この照会は、テキスト『Java』およびテキスト『XML』を含むテキスト・リソース項目タイプ『TextResource』内にあるテキスト・リソースを検索します。

```
/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'
")=1]
```

説明: これは、リソース・マネージャーのリソース内部でテキスト検索を実行する例です。『TIEREF』属性は、項目タイプ『TextResource』が表すリソースの表記として使用されることに注意してください。TIE 構文は、このケースでは contains-text 関数内部で通常どおりに使用されます。この機能がサポートする構文については、DB2 Net Search Engine (NSE) の資料を参照してください。

例 11: 前方方向の参照全探索

この照会は、会議ノートが EIP を含むタイトルを持つブックを参照している会議用のすべての FAQ (よく尋ねられる質問) を検索します。

```
/Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%EIP%"]]
/Conference_FAQ
```

例 12: 前方方向の参照全探索

この照会は、インターネットに関連する会議のノートに参照されるブックのすべての章を検索します。

```
/Conference[@Title LIKE "%Internet%"]
/Conference_Note/@PublicationRef =>
*/Book_Chapter
```

例 13: 逆方向の参照全探索

この照会は、任意のブックを指す参照を持つすべてのコンポーネントを検索します。

```
/Book/REFERENCEDBY/@REFERENCER => *
```

例 14: 逆方向の参照全探索

この照会は、XML に関するブックを参照する会議ノートの下のすべての FAQ (よく尋ねられる質問) を検索します。

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>
Conference_Note/Conference_FAQ
```

説明: 参照属性は Conference_Note コンポーネント内部で発生するため、これは非参照演算子の後の最初のコンポーネントとして現れなければならないコンポーネントであることに注意してください。この照会は、例えば、『=>』演算子が Conference の前にある場合には、空の結果セットを戻します。

例 15: 逆方向の参照全探索

この照会は、注釈に XML を含み、ブックを指す参照を持つすべてのコンポーネントを検索します。

```
/Book/REFERENCEDBY/@REFERENCER =>  
*[@Remark LIKE "%XML%"]
```

例 16: 最新バージョン関数

この照会は、最新バージョンのすべてのジャーナルを検索します。デフォルトでは、照会と一致する指示されたコンポーネント・タイプ・ビューのすべてのバージョンが戻されます。VERSIONID は、すべてのコンポーネント・タイプに含まれるシステム定義の属性です。

```
/Journal[@VERSIONID = latest-version(.)]
```

例 17: 非参照のターゲット上での最新バージョン関数

この照会は、任意の会議のノートで参照されるすべてのブックの最新バージョンを検索します。

```
/Conference/Conference_Note/@SYSREFERENCEATTRS =>  
Book[@VERSIONID = latest-version(.)]
```

例 18: ワイルドカード・コンポーネント上での最新バージョン関数

この照会は、任意のブックを指す参照を持つすべてのコンポーネントの最新バージョンを検索します。

```
/Book/REFERENCEDBY/@REFERENCER => *  
[@VERSIONID = latest-version(.)]
```

例 19: システム定義属性

この照会は、特定の項目 ID を持つすべてのルート・コンポーネントを検索します。

```
/*[@ITEMID =  
"A1001001A01J09B00241C95000"]
```

例 20: 文書モデル上でのテキスト検索

この照会は、パーツのいずれかにワード『XML』を含むすべての文書を検索します。

```
/Doc[contains-text(./ICMPARTS/@TIEREF, " 'XML' ")=1]
```

説明: 照会言語は、「文書」種別の特定の項目タイプの下に含まれるすべての ICM パーツ項目タイプへのアクセスを許可する、仮想コンポーネント『ICMPARTS』を提供します。

例 21: 文書モデル (ICM パーツへのアクセス)

この照会は、ストレージ ID が 555 である文書のすべてのパーツを検索します。

```
/Doc[@ArchiveID = 555]/ICMPARTS/  
@SYSREFERENCEATTRS => *
```

例 22: 文書モデル (ICM パーツへのアクセス)

この照会は、システム内のすべての文書の全パーツを検索します。

```
//ICMPARTS/@SYSREFERENCEATTRS => *
```

説明: 「Doc」 および 「Paper」 の両方の項目タイプがシステム内で 「文書」 として定義されているため、両方からの ICM パーツが結果に戻されます。

例 23: 属性の存在

この照会は、タイトルを持つすべてのルート・コンポーネントを検索します。

```
/*[@Title]
```

説明: ルート・コンポーネントのみが戻される制限をなくすには、以下のよう
に照会をダブルスラッシュを使用して開始するように再書き込みします。

```
//*[@Title]
```

例 24: リテラルおよび式の両方のリスト

この照会は、記事のタイトル、セクションのタイトル、または "IBM Systems Journal" のいずれかに等しいタイトルを持つすべてのジャーナルを検索します。

```
/Journal[@Title = [Journal_Article/@Title,  
./Journal_Section/@Title,"IBM Systems Journal"]]
```

例 25: 数値リテラルのリスト

この照会は、値段が 10 ドル、20 ドル、または 30 ドルのすべてのブックを検索します。

```
/Book[@Cost = [10, 20, 30]]
```

例 26: 照会結果のリスト

この照会は、タイトル 『Star Wars』 を持つすべてのジャーナルまたはブックを検索します。

```
[/Journal, /Book[@Title = "Star Wars"]]
```

例 27: 属性グループ

この照会は、説明が少なくとも 20 ページに渡る文書のすべての詳細を検索します。

```
/Doc[Doc_Description/@PageSummary.NumPages >=  
20]//Doc_Details
```

説明: 属性 (『NumPages』 など) が属性グループ (『PageSummary』 など) に含まれる場合は、その属性を GroupName.AttrName (PageSummary.NumPages など) として参照しなければならないことに注意してください。属性 『@NumPages』 は、Doc_Description の下では検出されません。

INTERSECT/EXCEPT によって入手される中間結果は、算術演算子 (単項 / 2 進) または比較演算子と結合できません。これらは、セット演算子 (UNION/INTERSECT/EXCEPT) によって結合されるか、またはそれぞれ単独で表示できます。

UNION/INTERSECT/EXCEPT の有効な使用例は、以下のとおりです。

1. (/Journal/Journal_Article[@Title = "Content Management"]
EXCEPT
//Journal_Article[@Classification =
"Security"])/Journal_Section

この照会は、EXCEPT の結果が照会全体の結果であるため有効です。結果は、演算子を使用して結合されません。

2. /Journal[(Journal_Editor/@LastName
UNION .//Journal_Author/@LastName) = "Davis"]

この照会は、UNION 演算子に制限がないため有効です。

3. /Journal[Journal_Article[Journal_Section/@Title INTERSECT
.//Journal_Figure/@Caption]/@Title = "Content Management"]

この照会は、INTERSECT の結果が演算子を使用して結合されないため有効です。

4. /Journal[@Title = "VLDB"]
UNION /Journal[@Cost = 20]
INTERSECT /Journal[@Organization = "ACM"]

この照会は、INTERSECT 演算子の結果がセット演算子 (UNION) を使用して結合されるため有効です。

INTERSECT/EXCEPT の無効な使用例は、以下のとおりです。

1. /Journal[(Journal_Editor/@LastName
INTERSECT .//Journal_Author/@LastName) = "Davis"]

この照会は、INTERSECT 演算子の結果が比較演算子 (=) を使用して結合されるため無効です。

2. /Journal[(.//Journal_Section/@SectionNum
EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]

この照会は、EXCEPT 演算子の結果が算術演算子 (+) を使用して結合されるため無効です。

照会言語について

照会言語の拡張機能 (テキスト・ストリングの中のワイルドカード "%" または "_" など) をサポートするには、ワイルドカードが通常文字として取り扱われている場合と、ワイルドカードに特別な意味が与えられている場合とを区別するために、エスケープ・シーケンスが使用されます。ワイルドカード文字を通常文字として使用したい場合は、エスケープ文字を前に置く必要があるため、どの文字がワイルドカードとして使用されているのかを知ることがユーザーにとって重要です。また、エスケープ・シーケンスは単一引用符と二重引用符を処理するためにも、使用されます。

照会に使用されるストリングに、特殊文字 (二重引用符、アポストロフィ) またはワイルドカード文字 (% 記号、下線、星印、疑問符) のいずれか、あるいはデフォルトのエスケープ文字 (円記号) が含まれる場合には、エスケープ・シーケンスを追加する必要があります。この処理は、ストリングが比較条件で使用される場合には最も単純ですが、LIKE 演算子およびテキスト検索関数で使用される場合には、やや複雑になります。特殊文字を適切に扱えば、照会は正しく実行され、照会結果の正確さが保証されます。

重要: 照会時のワイルドカード文字の使用が控え目だと、結果リストのサイズが著しく大きくなることがあります。その結果、パフォーマンスを低下させたり、予期しなかった検索結果を戻すことがあります。

比較演算子 ("=", "!="、">", "<", "BETWEEN" およびその他) のエスケープ・シーケンスの使用

二重引用符 "

二重引用符の前に、別の二重引用符を付けます。

例:

```
//Journal_Article[@Title = "Analysis of ""The Time Machine"" by H.  
G. Wells himself"]
```

記事のタイトルには、二重引用符で囲まれた書名 "The Time Machine" が含まれているので、これらの内部の二重引用符をエスケープさせる必要があります。

単一引用符 (アポストロフィ) '

この場合は、エスケープする必要はありません。

例:

```
/Book[@Title != "Uncle Tom's Cabin"]
```

LIKE 演算子でのエスケープ・シーケンスの使用

二重引用符 "

二重引用符の前に、別の二重引用符を付けます。

例:

```
//Journal_Article[@Title LIKE "Analysis of ""The Time Machine"" %"]
```

記事のタイトルには、二重引用符で囲まれた書名 "The Time Machine" が含まれているので、これらの内部の二重引用符をエスケープさせる必要があります。

単一引用符 (アポストロフィ) '

この場合は、エスケープする必要はありません。

例:

```
/Book[@Title LIKE "Uncle Tom's Cabin"]
```

ワイルドカード ("%", "_")

パーセント記号 "%" はストリング内で任意の数の任意の文字を表すために使用するワイルドカード文字で、LIKE 演算子と共に使用されます。下線 "_" は、単一の任意の文字を表すために使用するワイルドカード文字です。これらのワイルドカード文字を通常文字として扱いたい場合は、次のようにしてください。

1. ワイルドカード文字の前にエスケープ文字を付ける。
2. エスケープ文字を持つ ESCAPE 文節を LIKE 句の後に追加する。

例 A:

```
/Book[@Title LIKE "Plato%%$mposium"]
```

この例は、タイトルのスペルが不確実なブックを検索する場合に、ワイルドカード "%" と "_" をどのように使用するかを示しています。

例 B:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ in query"  
ESCAPE "!"]
```

この例では、検索ストリングに下線 "_" が通常文字として (ワイルドカードではなく) 含まれているので、感嘆符文字 "!" を使用して下線をエスケープすることができます。任意の単一文字をエスケープ文字として使用できます。

例 C:

```
//Journal_Article[@Title LIKE "_sage of underscore ¥_ in%" ESCAPE  
"¥"]
```

この照会では、ワイルドカード文字は、通常文字 ("¥" によってエスケープされる "_") および "Usage" という語の英大文字と小文字バージョンの両方をキャッチするワイルドカード ("_") の両方として、ストリングの複数の末尾をキャッチする "%" と共に使用されています。

例 D:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ on Yahoo!!"  
ESCAPE "!"]
```

エスケープ文字を通常文字として使用することもできます。その場合は、上記の "Yahoo!" を検索する例のように、エスケープ文字の前にエスケープ文字を置きます。

拡張テキスト検索 ("contains" および "score" 関数) でのエスケープ・シーケンスの使用

二重引用符 "

二重引用符の前に、別の二重引用符を付けます。

例:

```
//Journal_Article[contains-text (@Title, " 'Analysis of '"The Time  
Machine'" %' ")=1]
```

記事のタイトルには、二重引用符で囲まれた書名 "The Time Machine" が含まれているので、これらの内部の二重引用符をエスケープさせる必要があります。

単一引用符 (アポストロフィ) '

アポストロフィの前に別のアポストロフィを付けます。用語または句を囲むためにアポストロフィの対を使用するので、拡張テキスト検索では単一のアポストロフィは許可されていません。用語内にアポストロフィが含まれる場合、このアポストロフィを用語または句を終了させるアポストロフィと区別するためにエスケープする必要があります。

例 A:

```
/Book[contains-text (@Title, " 'Uncle Tom''s Cabin' ")=1] SORTBY  
(score (@Title, " 'Uncle Tom''s Cabin' "))
```

Tom's には、2 個のアポストロフィがあります。

例 B:

```
/Book[contains-text (@Title, " ('Greek' & 'Plato''s Symposium') &  
NOT ' Socrates' ")=1] SORTBY (score (@Title, " ('Greek' & 'Plato''s  
Symposium') & NOT ' Socrates' "))
```

Plato's には、2 個のアポストロフィがあります。

ワイルドカード ("%", "_")

拡張構文は、LIKE 演算子と同様に、 "%" および "_" をワイルドカードとして使用します。パーセント記号 "%" は、任意の数の任意の文字を表すために使用するワイルドカード文字です。下線 "_" は、単一の任意の文字を表すために使用するワイルドカード文字です。ワイルドカード文字を通常文字として扱いたい場合は、以下のようにします。

1. ワイルドカード文字の前にエスケープ文字を付けます。
2. エスケープ文字を使用する場合は、各用語の後に ESCAPE 文節を追加します。

例 A:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'  
ESCAPE '!' ")=1] SORTBY (score (@Title, " 'Usage of underscore !_ in  
query' ESCAPE '!' "))
```

この例では、感嘆符マーク "!" が下線前のエスケープ文字として使用されています。

例 B:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'  
ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage of underscore !_ on  
Yahoo!!' ESCAPE '!' | 'War and Peace' ")=1]
```

すべての用語のエスケープ文字が同じだとしても、ワイルドカードをエスケープする場合はテキスト検索ストリングの各用語の後に、ESCAPE 文節を追加しなければなりません。

基本テキスト検索 ("contains-text-basic" および "score-basic" 関数) でのエスケープ・シーケンスの使用

二重引用符 "

二重引用符の前に、別の二重引用符を付けます。

例:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of ""The  
Time Machine"" ")=1]
```

記事のタイトルには、二重引用符で囲まれた書名 "The Time Machine" が含まれているので、これらの内部の二重引用符をエスケープさせる必要があります。ブックのタイトルは、句としてそれを保持するためにアポストロフィで囲みます。

単一引用符 (アポストロフィ) '

アポストロフィの前に別のアポストロフィを付けます。基本テキスト検索構文では、用語にスペースを含めることができるように、用語を単一引用符で囲むことができます。したがって、アポストロフィを二重にすることは、新規用語を開始するアポストロフィと用語内で表示されるアポストロフィを区別するために必要です。

例 A:

```
/Book[contains-text-basic (@Title, "Uncle Tom's Cabin")=1]SORTBY  
(score-basic (@Title, "Uncle Tom's Cabin"))
```

Tom's には、2 個のアポストロフィがあります。

例 B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato's Symposium'  
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek +'Plato's  
Symposium' -Socrates "))
```

Plato's には、2 個のアポストロフィがあります。また Plato's Symposium は句なので、単一引用符で囲みます。

ワイルドカード ("*", "?" および "%")

"*", "?", および "%" 文字をワイルドカードとして使用しない場合は、これらの文字の前に円記号 "%" を付けます。星印 "*" は、contains-text-basic および score-basic 関数用に、基本テキスト検索内で任意の数の任意の文字を表すために使用されるワイルドカード文字です。疑問符 "?" は、単一の任意の文字を表すために使用されるワイルドカード文字です。基本テキスト検索の場合、検索する用語自体にワイルドカード文字が含まれており、そのワイルドカード文字を通常文字として使用したい場合に、照会言語が、エスケープ文字の円記号 "%" を提供します。

例 A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'  
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek  
+'Plato*s*S?mposium' -Socrates "))
```

この例は、用語のスペルが確実にないときの基本テキスト検索の使用方法を示しています。 "*" および "?" 文字はワイルドカードとして扱われており、エスケープされません。

例 B:

```
/Book[contains-text-basic (@Title, "Why forgive%?")=1] SORTBY  
(score-basic (@Title, "Why forgive%?"))
```

この例では、タイトルに通常文字として疑問符 "?" が含まれており、円記号を使用してこの文字をエスケープすることができます。

例 C:

```
//Journal_Section[contains-text-basic (@Title,
"C:¥¥OurWork¥¥IsNeverDone")=1] SORTBY (score-basic (@Title,
"C:¥¥OurWork¥¥IsNeverDone"))
```

検索用語 "C:¥¥OurWork¥¥IsNeverDone" 内でそのまま表示される各円記号は、別の円記号を使用してエスケープしなければなりません。

Java および C++ でのエスケープ・シーケンスの使用

特殊文字 (例えば、二重引用符および円記号) の前に円記号を付けます。

例:

照会:

```
/Book[contains-text-basic (@Title, "Why forgive¥")=1]
```

Java

```
String query = "/Book[contains-text-basic (@Title, ¥"Why forgive¥¥?¥")=1]";
```

C++

```
DKString query ("/Book[contains-text-basic (@Title, ¥"Why forgive¥¥?¥")=1]");
```

内部の二重引用符と疑問符 (?) の前の円記号の前に、円記号が付きます。この処理は、Java および C++ プログラム言語に固有のものです。詳細については、これらの言語の仕様書を参照してください。

照会言語の文法

照会言語の正式な文法は以下のとおりです。

- (* キーワード *)
- AND = ("a" | "A"), ("n" | "N"), ("d" | "D") ;
- ASCENDING = ("a" | "A"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n" | "N"), ("d" | "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
- BETWEEN = ("b" | "B"), ("e" | "E"), ("t" | "T"), ("w" | "W"), ("e" | "E"), ("e" | "E"), ("n" | "N") ;
- DESCENDING = ("d" | "D"), ("e" | "E"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n" | "N"), ("d" | "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
- DIV = ("d" | "D"), ("i" | "I"), ("v" | "V") ;
- EXCEPT = ("e" | "E"), ("x" | "X"), ("c" | "C"), ("e" | "E"), ("p" | "P"), ("t" | "T") ;
- INTERSECT = ("i" | "I"), ("n" | "N"), ("t" | "T"), ("e" | "E"), ("r" | "R"), ("s" | "S"), ("e" | "E"), ("c" | "C"), ("t" | "T") ;
- LIKE = ("l" | "L"), ("i" | "I"), ("k" | "K"), ("e" | "E") ;

- MOD = ("m" | "M"), ("o" | "O"), ("d" | "D") ;
- NOT = ("n" | "N"), ("o" | "O"), ("t" | "T") ;
- OR = ("o" | "O"), ("r" | "R") ;
- SORTBY = ("s" | "S"), ("o" | "O"), ("r" | "R"), ("t" | "T"), ("b" | "B"), ("y" | "Y") ;
- UNION = ("u" | "U"), ("n" | "N"), ("i" | "I"), ("o" | "O"), ("n" | "N") ;
- IS = ("i" | "I"), ("s" | "S");
- NULL = ("n" | "N"), ("u" | "U"), ("l" | "L"), ("l" | "L");
- ESCAPE_KEYWORD = ("e" | "E"), ("s" | "S"), ("c" | "C"), ("a" | "A"), ("p" | "P"), ("e" | "E");
- KEYWORD = (AND | ASCENDING | BETWEEN | DESCENDING | DIV | EXCEPT | INTERSECT | LIKE | MOD | NOT | OR | SORTBY | UNION | IS | NULL) ;
- (* リテラル *)
- DIGIT = ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- NONZERO_DIGIT = ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- Exponent = (e | E), ["+" | "-"], DIGIT, { DIGIT }
- INTEGER_LITERAL = "0" | NONZERO_DIGIT, {DIGIT} ;
- FLOAT_LITERAL = DIGIT, { DIGIT }, ".", { DIGIT }, [Exponent] | ["."], DIGIT, { DIGIT }, [Exponent] ;
- (* UNICODE_CHARACTER は、すべてのユニコード文字およびエスケープ・シーケンスのセットです。その定義は、この文書には含まれません。*) (* スtring・リテラルは、二重引用符で範囲設定され、二重引用符以外の任意の文字を入れることができます。String・リテラルの一部として二重引用符を含めるには、例えば、1 つの二重引用符が別の二重引用符によってエスケープされる、2 つの連続した二重引用符を指定します。これらは、1 つの二重引用符文字として扱われます。*)
- STRING_LITERAL = "'", { (UNICODE_CHARACTER - "'") | ("'", "'") }, "'";
- (* エスケープ・シーケンスは、二重引用符によって区切られる単一文字です。二重引用符自体をエスケープ文字に指定するには、2 つの連続した二重引用符を指定します。つまり、二重引用符を別の二重引用符によってエスケープします。これらは、1 つの二重引用符文字として扱われます。 ESCAPE_CHARACTER の正しい値の完全な説明については、『DB2 SQL 解説書』セクションの『LIKE 述部』を参照してください。*)
- ESCAPE_LITERAL = "'", ((ESCAPE_CHARACTER - "'") | ("'", "'")), "'";
- LETTER = ("a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "_" | "\$") ;
- (* IDENTIFIER は、文字 (a ~ z, A ~ Z)、下線、またはドル記号から始まり、その後、ゼロまたはさらなる文字、下線、ドル記号、または数字 (0 ~ 9) が続きます。単一引用符で囲まれている場合にのみ、キーワードが IDENTIFIER になることがあります。*)

- IDENTIFIER = (LETTER, { LETTER | DIGIT }) - KEYWORD | "'", LETTER, { LETTER | DIGIT }, "'";
- ExpressionWithOptionalSortBy = LogicalOrSetExpression, SORTBY, "(", SortSpecList, ")" | Expression ;
- Expression = LogicalOrSetExpression ;
- SortSpecList = SortSpec, { ",", SortSpec } ;
- SortSpec = Expression, [ASCENDING | DESCENDING] ;
- LogicalOrSetExpression = LogicalOrSetTerm | LogicalOrSetExpression, (OR | UNION | "I" | EXCEPT), LogicalOrSetTerm ;
- LogicalOrSetTerm = LogicalOrSetPrimitive | LogicalOrSetTerm, (AND | INTERSECT), LogicalOrSetPrimitive ;
- LogicalOrSetPrimitive = [NOT], SequencedValue ;
- SequencedValue = ValueExpression ;
- ValueExpression = Comparison ;
- Comparison = ArithmeticExpression | Comparison, CompareOperator, ArithmeticExpression, ESCAPE_KEYWORD, ESCAPE_LITERAL | Comparison, CompareOperator, ArithmeticExpression | Comparison, [NOT], BETWEEN, ArithmeticExpression, AND, ArithmeticExpression | Comparison, IS, [NOT], NULL ;
- ArithmeticExpression = ArithmeticTerm | ArithmeticExpression, ("+" | "-"), ArithmeticTerm ;
- ArithmeticTerm = ArithmeticFactor | ArithmeticTerm, ("*" | DIV | MOD), ArithmeticFactor ;
- ArithmeticFactor = ArithmeticPrimitive | ("+" | "-"), ArithmeticFactor ;
- ArithmeticPrimitive = BasicExpression, OptionalPredicateList | PathExpression ;
- PathExpression = Path | ("/" | "//"), Path | BasicExpression, OptionalPredicateList, ("/" | "//"), Path ;
- Path = Step | Path, ("/" | "//"), Step ;
- Step = NodeGenerator, OptionalPredicateList ;
- NodeGenerator = NameTest | "@", NameTest | "@", NameTest, "=>", NameTest | ".." ;
- OptionalPredicateList = {Predicate} ;
- Predicate ::= [", Expression, "]" ;
- BasicExpression = Literal | FunctionName, "(", OptionalExpressionList, ")" | "(" Expression ")" | ListConstructor | "." ;
- FunctionName = QName ;
- Literal = STRING_LITERAL | INTEGER_LITERAL | FLOAT_LITERAL ;
- OptionalExpressionList = [ExpressionList] ;
- ExpressionList = Expression, { ",", Expression } ;
- ListConstructor = "[", [ListContent], "]" ;
- ListContent = Expression, { ",", Expression } ;
- NameTest = QName | "*" ;
- QName = LocalPart ;

- LocalPart = IDENTIFIER;
- CompareOperator = "=" | "<" | "<=" | ">" | ">=" | "!=" | [NOT] LIKE;

リソース・マネージャーの使用

Content Manager は、管理対象リソース (オブジェクト) のコレクションを制御します。また、必要なストレージおよび階層ストレージ管理 (HSM) インフラストラクチャの管理も行いますが、最初に、リソース・マネージャーが HSM をサポートするよう構成する必要があります。リソース・マネージャーの機能により、ストリーミング、ZIP 圧縮、ZIP 解凍、暗号化、エンコード、トランスコーディング、検索、またはテキスト・マイニングなど、複数タイプのオブジェクトについてタイプ固有のサービスをサポートします。

1 つのリソース・マネージャーが、1 つのライブラリー・サーバーによって独占的に使用されます。Content Manager システムによって配布される各リソース・マネージャーは、固有のデータ・アクセス API の共通サブセットを提供します。これにより、制御側のライブラリー・サーバー、他の Content Manager コンポーネント、およびアプリケーションから、ローカル (同じネットワーク・ノード上) またはリモートでのアクセスが可能になります。

その他のデータ・アクセス API は、リソース・マネージャー独自のクライアント・サポート、または CIFS、NFS、FTP などの標準のネットワーク・アクセス・プロトコルを使用して、リソース・マネージャーへのリモート・アクセスを可能にします。リモート・アクセスには、クライアント / サーバー接続が使用されます。クライアントは、標準の Web サーバーを使用して HTTP により Content Manager リソース・マネージャーと通信します。データのデリバリーは、HTTP、FTP、および FS データ転送プロトコルに基づいて行われます。HTTP により、Content Manager 管理コンテンツにアクセスするアプリケーションまたは Content Manager コンポーネントは、ライブラリー・サーバーおよびリソース・マネージャーと動的に三角関係を形成することができます。この三角形が、アプリケーションと各リソース・マネージャー間の直接のデータ・アクセス・パス、およびライブラリー・サーバーとリソース・マネージャー間の制御パスを形成します。この三角形の概念は、単一ノード構成から地理的に分散した構成に至るまで、あらゆるネットワーク構成に適用することができます。

新しいアーキテクチャーでは、ホスト・ベースのサブシステム、アクセス制御を処理しない単一ユーザー・システム、またはビジネス・ポリシー上アプリケーションによる直接アクセスができない極めて重要な情報を含むシステムなど、アプリケーションが直接アクセスできないリソース・マネージャーにも接続することができます。この場合、このようなリソース・マネージャーへのアクセスは、間接的になります。データ転送のプルおよびプッシュ・パラダイムは、どちらも、Content Manager システムと同期および同期呼び出しによって処理されます。

リソース・マネージャーの構成方法については、「*Content Management System の計画とインストール*」および samples ディレクトリー `cmroot¥samples¥java¥icm` にある `SResourceMgrDefCreationICM` サンプルを参照してください。

リソース・マネージャー・オブジェクトの使用

Content Manager 内のすべての管理対象エンティティは項目と呼ばれます。項目には、文書やフォルダーなど純粋に論理的なエンティティを表すタイプと、ワープロ文書のテキスト・データ、請求書のスキャン・イメージ、または交通事故のビデオ・クリップなど、物理的なデータ・オブジェクトを表すエンティティの 2 つのタイプがあります。オブジェクトには、論理文書に関連した物理データを処理するために必要な特殊な状態と動作があります。

また、リソース・オブジェクトは、ファイル・システムのファイル、ビデオ・サーバーのビデオ・クリップ、および BLOB などを表します。実行時に、リソース・オブジェクトはそれがポイントする物理データへのアクセスに使用されます。このため、Content Manager 内のリソース・オブジェクトにはタイプがあります。つまり、オブジェクトは特定の状態と動作をとります。ライブラリー・サーバーおよびリソース・マネージャーは、オブジェクトの状態を保管するためのスキーマを共有します。Content Manager が提供する基本オブジェクト・タイプには、汎用の BLOB または CLOB、テキスト、イメージ、およびビデオ・コンテンツ・オブジェクトがあります。また、事前定義タイプのサブクラスを作成することもできます。リソース・オブジェクトは、検索および取り出しに使用されるユーザー定義属性を持つ場合もあります。

Content Manager システムの観点から見ると、各オブジェクトは、固有の論理 ID である Uniform Resource Identifier (URI) によって表されます。ライブラリー・サーバーは、URI ネーム・スペースを管理します。ライブラリー・サーバーは要求を受けると、URI を Uniform Resource Locators (URL) にマップします。URL は、物理データへのアクセスを取得するために使用されます。URL がリソース・マネージャーの管理するストレージ域を直接ポイントすることはありません。その代わり、リソース・マネージャーはローカルのネーム・スペースを使用して、論理オブジェクトの名前を物理ファイル名に変換します。オブジェクトの URI は、特定のリソース・マネージャーによって作成されます。ライブラリー・サーバーまたはエンド・ユーザーは、オブジェクト URI (その名前) を提示することができますが、決定はリソース・マネージャーによって行われます。

Content Manager リソース・マネージャー API を使用して、オブジェクトにアクセスできます (保管、検索、更新、削除など)。オブジェクト (ストリーム、マルチキャスト、およびステージ) またはファイル・システムに固有な API を使用できる場合もあります。

リソース・マネージャー・オブジェクトの使用方法については、samples ディレクトリー (CMBROOT¥Samples¥java¥icm、または CMBROOT¥Samples¥cpp¥icm) にある SResourceItemCreationICM サンプルを参照してください。

Content Manager における文書管理

Content Manager は、ビジネス・オブジェクトの管理に使用することのできる柔軟な文書管理データ・モデルをインプリメントしています。データ・モデルの基本エレメントには、フォルダー、文書、およびオブジェクトがあります。

前述したように、文書、フォルダー、およびその他のオブジェクトは、すべて Content Manager システムの項目により表されます。API レベルでは、文書とフォ

ルダの違いは、意味タイプとそれぞれの DKFolder 機能だけです。文書は、その文書を記述した属性またはメタデータで構成され、これには単一値属性 (文書名、日付、件名)、複数值属性 (キーワード)、および複数值属性のコレクション (郵便番号、都道府県、市区町村、番地による住所) が含まれます。

文書管理データ・モデルは、文書パーツを使用して、オブジェクト (リソース項目) と文書を関連付けます。このモデルは、文書を構成する複数のパーツをサポートします。例えば、各ページが異なるパーツの場合があります。アプリケーションが文書を構成するパーツの順序を判別できるようにするために、文書パーツにパーツ番号が保管されます。文書パーツには、オブジェクトへのポインター (参照属性) が含まれ、そのポインターには、そのパーツに関するその他の情報 (MIME タイプ、サイズ、そのパーツが含まれるリソース・マネージャーの ID、そのリソース・マネージャーの集合名など) が含まれます。オブジェクトは、それぞれ異なる属性を持つことができます。例えば、注釈には X 座標および Y 座標があり、注釈ログにはその注釈のテキストの CCSID がある場合などです。

文書管理データ・モデルを理解するため、ユーザーがクライアント・アプリケーションを使用して文書をインポートするケースを考えてみます。

- ウィンドウが表示されます。
- ユーザーは、システムにインポートするファイルの名前を入力 (または選択) します。例えば「警察レポート (police report)」などです。
- 文書タイプ (「メモ (memo)」、「保険料請求 (claim)」、「設計 (design)」) を選択します。
- 新しいウィンドウが開き、ここでは文書を記述する属性を入力することができます。例えば、日付、請求番号、および保険証券番号などです。
- ユーザーは、文書パーツを定義して、属性の値を入力します。例えば、「警察レポート (police report)」は「保険料請求 (claim)」の文書パーツです。
- ユーザーは文書記述の入力を完了し、タスクを終了します。警察レポートが作成されます。

クライアント・アプリケーションは、API または JavaBeans のいずれかを使用して、ライブラリー・サーバーおよびリソース・マネージャーに接続します。システムは、文書を保管するために 2 つの項目 (非リソース項目およびリソース項目) を作成します。警察レポートにはリソース・マネージャーに保管された写真が含まれるため、2 つの項目が作成されます。文書は、単一の API 呼び出しで作成されます。次に、オブジェクトがリソース・マネージャーに保管され、リソース・マネージャーはタイム・スタンプおよびオブジェクトのその他のメタデータを戻します。リソース・マネージャーはオブジェクトの参照属性を作成し、これを文書の子コンポーネントに挿入します。ライブラリー・サーバーへの最後の呼び出しにより子コンポーネントが保管され、属性が更新されます。API に何らかの障害が発生した場合でも文書作成が部分的とならないよう、プロセス全体がトランザクションによって結合されます。

文書を作成した後、これを更新することができます。更新には、メタデータの変更またはコンテンツの変更の 2 種類があります。ライブラリー・サーバーが自動的に新しい項目レコードを作成して次のバージョン番号を付け (項目のバージョン管理が可能な場合)、その項目に関連付けられたすべての子コンポーネントをコピーします。

文書管理データ・モデルの作成

このセクションでは、文書管理データ・モデルに関連したメインタスクの実行を支援します。

- 文書項目タイプの作成。
- 文書の作成。
- 文書の更新。
- 文書の検索と削除。
- 文書管理データ・モデル内のパーツのバージョン管理。

文書項目タイプの作成

Java:

1. 文書 ItemType を ItemType classification = DK_ICM_ITEMTYPE_CLASS_DOC_MODEL で作成します。以下のように設定します。

```
docItemTypeDef.setVersionControl  
    ((short)DKConstantICM.DK_ICM_VERSION_CONTROL_ALWAYS);  
docItemTypeDef.setVersioningType  
    (DKConstantICM.DK_ICM_ITEM_VERSIONING_FULL);  
docItemTypeDef.setDeleteRule(DKConstantICM.DK_ICM_DELETE_RULE_CASCADE);
```

2. ItemType 関係を作成して、文書にパーツを追加します。それぞれのパーツの EntityDef を検索します。

```
Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
```

3. それぞれのパーツごとに、ItemType 関係を作成して、値を設定します。

```
DKItemTypeRelationDefICM itRel = new DKItemTypeRelationDefICM(ds);  
itRel.setTargetItemTypeID(PartName);  
itRel.setDefaultRMCode((short)1);  
itRel.setDefaultACLCode(DKConstantICM.DK_ICM_SUPER_USER_ACL);  
itRel.setDefaultCollCode((short)1);  
itRel.setDefaultPrefetchCollCode((short)1);  
itRel.setVersionControl(DK_ICM_VERSION_CONTROL_NERVER);
```

4. 文書 (ソース) に ItemType 関係を追加します。

```
docItemTypeDef.addItemTypeRelation(itRel);
```

5. 文書 ItemType を永続ストアに追加します。

```
docItemTypeDef.add();
```

C++:

1. コンテンツ・サーバー定義オブジェクトを検索します。

```
DKDatastoreDefICM * dsDefICM = (DKDatastoreDefICM *)dsICM->datastoreDef();
```

2. コンテンツ・サーバー管理オブジェクトを検索します。

```
DKDatastoreAdminICM * pdsAdmin =  
    (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();  
DKItemTypeDefICM *itemType = NULL;  
DKItemTypeRelationDefICM *itemTypeRel = NULL;  
DKAttrDefICM* attr = NULL;
```

3. 文書項目タイプの属性を検索します。その属性が存在しない場合は、作成します。

```

dkAttrDef *pAttr = dsDefICM->retrieveAttr("docTitle1");
if(pAttr == NULL)
{
    attr = new DKAttrDefICM(dsICM);
    attr->setName("docTitle1"); //attribute name column name
    attr->setType(DK_CM_CHAR);
    attr->setSize(100);
    attr->setNullable(false);
    attr->setUnique(false);
    attr->add();
    pAttr = attr;
}
itemType = new DKItemTypeDefICM(dsICM);
itemType->setName("DocModelTest");
itemType->setDescription("This is a test Item Type");
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
itemType->setAutoLinkEnable(false);
itemType->setVersionControl((short)DK_ICM_VERSION_CONTROL_ALWAYS);
itemType->setVersioningType(DK_ICM_ITEM_VERSIONING_FULL);
itemType->addAttr(pAttr);

```

4. 作成した文書と part1 の間の関係を作成します。このためには、まずそれぞれのパーツごとに EntityDef を検索します。

```

DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)
    dsDefICM->retrieveEntity("ICMBASE");
//int part1ITypeid = itemTypePart1->getItemTypeId();
int part1ITypeid = itemTypePart1->getIntId();

```

5. それぞれのパーツごとに、項目タイプ関係を作成して、値を設定します。

```

DKItemTypeRelationDefICM *itemTypeRelPart1=
    new DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart1->setTargetItemTypeID(part1ITypeid);
//sets the default resource manager
itemTypeRelPart1->setDefaultRMCode((short)1);
//sets the default ACL code
itemTypeRelPart1->setDefaultACLCode(1);

```

6. この項目タイプに関連する項目リソースを保管する先のデフォルト・コレクションを設定します。

```

itemTypeRelPart1->setDefaultCollCode((short)1);

```

7. この項目タイプに関連する項目リソースを保管する先のデフォルト事前取り出しコレクションを設定します。

```

itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);
itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);
itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());

```

8. 文書 (ソース) に項目タイプ関係を追加します。

```

itemType->addItemTypeRelation(itemTypeRelPart1);

```

9. 作成した文書と part2 の間の関係を作成します。このためには、まずそれぞれのパーツごとに EntityDef を検索します。

```

DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)
    dsDefICM->retrieveEntity("ICMANNOTATION");
int part2ITypeid = itemTypePart2->getIntId();

```

10. それぞれのパーツごとに、項目タイプ関係を作成して、値を設定します。

```

DKItemTypeRelationDefICM * itemTypeRelPart2= new
    DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart2->setTargetItemTypeID(part2ITypeid);
itemTypeRelPart2->setDefaultRMCode((short)1);
itemTypeRelPart2->setDefaultACLCode(1);
itemTypeRelPart2->setDefaultCollCode((short)1);

```

```
itemTypeRelPart2->setDefaultPrefetchCollCode((short)1);  
  
itemTypeRelPart2->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);  
  
itemTypeRelPart2->setSourceItemTypeID(itemType->getIntId());
```

11. 文書 (ソース) に項目タイプ関係を追加します。

```
itemType->addItemTypeRelation(itemTypeRelPart2);
```

12. ライブラリー・サーバー内の項目タイプの定義を更新します。

```
itemType->add();
```

詳細については、SItemTypeCreationICM サンプルを参照してください。

文書の作成

意味タイプ「文書」を持つ項目はその中に、属性 (他の意味タイプの項目と同じ) および複数の「パーツ」 (他の意味タイプの項目と異なる) が含まれます。以下のステップで、1 つの属性と 1 つの「パーツ」が含まれる項目 (定義済みの文書項目タイプに基づく) の作成プロセスを説明します。以下のステップでは、1 つの属性 "S_varchar" と 1 つのパーツ ("ICMBASE") を持つ項目タイプ "s_simple" は既に定義されていると仮定しています。

Java

1. 文書 DDO を作成します。

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
    DKConstant.DK_CM_DOCUMENT);
short dataId = 0;
String attrValue = "Test";
```

2. 文書の属性を設定します。このとき、項目タイプは属性を 1 つだけ持つと想定しています。

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR, "S_varchar");
ddoDocument.setData(dataId, attrValue);
DKParts parts = null;
// Document's parts
```

3. 文書のパーツ・コレクションをアクセスします。

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
        DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
}
else
{
    parts = (DKParts)ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

4. 事前定義タイプ "ICMBASE" のパーツを作成します。このパーツは、作成済みの文書に追加されます。以下で作成される文書は、パーツを 1 つだけ持つ項目タイプに基づく想定されています。

```
DKLobICM pLobPart = (DKLobICM) dsICM.createDDO("ICMBASE",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setPartNumber(1);
// Set the mime type for added part
pLobPart.setMimeType("text/plain");
String partValue = "This is a base part";
pLobPart.setContent(partValue.getBytes());
```

5. parts コレクションに、作成したパーツを追加します。これは、据え置き保管 (この文書 DDO が永続するまで、変更はデータ・ストアに確定されない) になるので注意してください。

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. データ・ストアに文書を永続させます。

```
ddoDocument.add();
```

C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a new DDO of type DocModelTest and semantic type DK_CM_DOCUMENT
DKDDO* ddoDocument = dsICM->createDDO("DocModelTest",DK_CM_DOCUMENT);
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("docTitle1")),DKString("this is a string value"));
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
    pdsDef->retrieveEntity("DocModelTest");
// Retrieve the collection of DKItemTypeRelationDefICM object for given source
// item type from the persistent store
dkCollection* pRelationColl = itemType->retrieveItemTypeRelations();
dkIterator* pIter = pRelationColl->createIterator();
int noOfPartsToCreate = pRelationColl->cardinality();
DKParts* pPartColl= NULL;
// Create the parts collection for the object if it does not exist
short dataId = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if (dataId ==0) {
    dataId = ddoDocument->addData(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
    pPartColl = new DKParts();
    ddoDocument->setData(dataId,pPartColl);
}
else {
    pPartColl =(DKParts*) (ddoDocument->getData(dataId).value());
    if (pPartColl ==NULL) {
        pPartColl = new DKParts();
        ddoDocument->setData(dataId,pPartColl);
    }
}
int i=0;
DKItemTypeRelationDefICM* itemTypeRelPart =NULL;
DKItemTypeDefICM* pEnt =NULL;
// CV v8 BLOB
DKLobICM* pPart =NULL;
DKString str = "This is to test the document model with two parts";
while(pIter->more()) {
    i=i+1;
    itemTypeRelPart=(DKItemTypeRelationDefICM*) pIter->next()->value();
    pEnt =(DKItemTypeDefICM*)
        ((DKDatastoreDefICM*) pdsDef)->retrieveEntity(
            (long)itemTypeRelPart->getTargetItemTypeID());
    pPart =(DKLobICM*) dsICM->createDDO(pEnt->getName(), DK_CM_RESOURCE);
    pPart->setPartNumber(i);
    pPart->setContent(str);

    DKAny any = (dkDataObjectBase*) pPart;
    pPartColl->addElement(any);
}
//Add the DDO to the datastore
ddoDocument->add();
```

詳細については、SDocModelItemICM サンプルを参照してください。

文書の更新

次のステップでは、意味タイプ「文書」の項目を更新するプロセスを説明します。
次のステップで、新規パーツが追加され、属性値が更新されます。

Java

1. 文書項目の属性値を更新します。

```
String attrValue = "New Value";
short dataId=ddoDocument.dataId
    (DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
```

2. 文書のパーツ・コレクションをアクセスします。

```
DKParts parts      = null;
// Document's parts
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
        DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
}
else
{
    parts = (DKParts)ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

3. 新規パーツのデータを作成します。

```
String partValue = "This is an annotation";
```

4. 定義済みタイプ "ICMANNOTATION" のパーツを作成します。このパーツは、作成済みの文書に追加されます。ここでは、作成中の文書はパーツを 1 つしか持たない項目タイプに基づく想定されています。新規パーツが追加されると、文書はパーツを 2 つ持つことになります。

```
DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setContent(partValue.getBytes());
pLobPart.setPartNumber(2);
```

5. parts コレクションに、作成したパーツを追加します。これは、据え置き保管 (この文書 DDO が永続するまで、変更はデータ・ストアに確定されない) になるので注意してください。

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. データ・ストアに変更した文書を永続させます。

```
ddoDocument.update();
```

C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a Document DDO from a PID string
DKString pidString = ....;
//...
DKDDO* ddoDocument = dsICM->createDDO(pidString);
DKPidICM* pPID = (DKPidICM*) ddoDocument->getPidObject();
DKString* pstrItemType = &pPID->getObjectType();
// Retrieves the definition for the item type "DocModelTest" from the
// persistent datastore. The definition is returned as a the *dkEntityDef
// object that is in turn typecast to a DKItemTypeDefICM object
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
    pdsDef->retrieveEntity(*pstrItemType);
ddoDocument->setData(ddoDocument->dataId(
    DK_CM_NAMESPACE_ATTR,DKString("docTitle1")),
    DKString("this is a new string value"));

DKString updateString = "This is updated part";
DKSequentialIterator* pSeqIter = NULL;
DKLobICM* pPart = NULL;
short dataId = ddoDocument->dataId (DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS);
DKParts* pParts = (DKParts*) &ddoDocument->getData(dataId);
if (pParts != NULL) {
    pSeqIter = (DKSequentialIterator*) pParts->createIterator();
}
else return; // quit
pPart = (DKLobICM *) pSeqIter->next();
// Update the existing part with the new content
pPart->setContent(updateString);
// Add a new part to the Document
DKLobICM* pPart1 =
    (DKLobICM*) dsICM->createDDO ("ICMNOTELOG", DK_CM_RESOURCE);
pPart1->setPartNumber(3);
DKString pTempData = "This is to test the document model with two parts";
pPart1->setContent(pTempData);
DKAny any = (dkDataObjectBase*)(DKDDO*) pPart1;
pParts->addElement(any);
//Update the DDO information in the datastore.
ddoDocument->update();
delete pSeqIter;
```

詳細については、SDocModelItemICM サンプルを参照してください。

文書の検索と削除

文書を検索するには、`ddo.retrieve(option)` を呼び出します。オプションを `DK_ICM_CONTENT_YES` に設定すると、パーツ TOC リストとともにパーツも検索されます。設定しないと、パーツの TOC リストのみが検索されます。

文書を削除するには、`ddo.del()` を呼び出します。文書とそれに付加されたパーツが削除されます。パーツのある文書の検索と削除について詳しくは、SDocModelItemICM API サンプルを参照してください。

文書管理データ・モデル内のパーツのバージョン管理

文書パーツのバージョン管理プロパティは、文書の項目タイプの中で選択した、各パーツ・タイプに対する項目タイプ関係のバージョン管理プロパティによって決まります。文書パーツのバージョン管理特性には、次のものがあります。

- 標準の文書と同様に、パーツは「バージョン管理 - 常時」、「バージョン管理 - なし」(デフォルト)、および「アプリケーション制御」バージョン管理の 3 つのバージョン管理モデルのうちの 1 つを持つことができます。
- 項目タイプが「バージョン管理 - なし」のバージョン管理ポリシーを持つ場合、そのパーツも「バージョン管理 - なし」のバージョン管理ポリシーを持ちます。
- 項目タイプ T が「バージョン管理 - 常時 (versioned-always)」のバージョン・ポリシーおよび項目タイプ T の項目 I を持ち、その属性またはパーツ・コレクションのいずれかを (パーツを追加 / 削除または更新することによって) 変更する場合、項目 I の新規バージョンが作成されます。
- 文書のパーツは、文書自体と異なり、バージョンの最大数を持ちません。
- 重要なパーツ用の項目タイプ関係オブジェクトから、パーツ・レベルのバージョン管理規則を入手することができます (ベース、注、注釈など)。

以下の例は、項目タイプの基本パーツのバージョン管理規則の入手方法を示しています。

Java

```
String itemTypeName="book"; //example item type
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM item =null;
DKDatastoreICM ds = new DKDatastoreICM();
...
item = (DKItemTypeDefICM)ds.datastoreDef.retrieveEntity(itemTypeName;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemRelation.getVersionControl();
```

C++

```
DKString itemTypeName="book"; //example item type
//Specify the part id for which we need versioning information
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM * itemType;
//Retrieve the entity corresponding to the "book" item type
itemType =
(DKItemTypeDefICM *)dsICM->datastoreDef()->retrieveEntity(itemTypeName);
//Retrieve the relation object for the specified part id
DKItemTypeRelationDefICM * itemTypeRelation =
(DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Retrieve the version control policy for the specified part
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

トランザクションの使用

Content Manager では、トランザクションにより、ライブラリー・サーバーとこれに接続するリソース・マネージャー間の一貫性を保つことができます。トランザクションは、ユーザーにより決定される回復可能な作業単位で、ライブラリー・サーバーへの単一の接続を通じて作成される一連の連続した API 呼び出しから構成されます。一連の DKDatastoreICM メソッド呼び出しは、DDO および XDO によって直接または間接的に行われます。

トランザクションの有効範囲と、そのトランザクション内の作業量は、デフォルトでは、単一の API メソッド (暗黙的なトランザクション) によって実行される作業となります。このタイプのトランザクションは、トランザクションのパフォーマンスが最も高い有効範囲であり、推奨されています。ただし、複数のメソッド呼び出し (明示的トランザクション) を組み込むように作業単位の有効範囲を大きくして変更することができますが、このタイプのトランザクションを使用すると、パフォーマンスのオーバーヘッドにつながることがあります。

トランザクションが終了すると、トランザクション全体がコミットまたはロールバックされます。コミットされた場合、トランザクション内で API 呼び出しによって行われたすべての Content Manager サーバーの変更が永続化されます。トランザクションがロールバックされるか失敗した場合、そのトランザクション内で行われた変更はすべてロールバック処理中に元に戻されます。

暗黙的トランザクションの場合は、トランザクションのコミットとロールバックが自動的に行われます。明示的トランザクションが使用されている場合は、トランザクションのコミットは、アプリケーションにより制御されます。一方、トランザクションのロールバックは、アプリケーションによって、または自動的に Content Manager システムによって開始することができます。Content Manager システムは、重大エラーの発生時、またはライブラリー・サーバーとデータベース間のデッドロックを解決する場合に、ロールバックを開始します。

トランザクション内では、コミットされていないリソース・マネージャーの変更は、トランザクションがコミットされるまで、変更を行ったアプリケーションから参照できません。例えば、リソース・マネージャーの項目に変更を加え、それを保管します。トランザクションがコミットされる前にその項目を検索した場合、項目に対して行った変更は反映されません。トランザクションがコミットされるまでは、更新した項目を表示することはできません。

単一のライブラリー・サーバー接続を通じた並行または重複するトランザクションはサポートされません。並行トランザクションを保持するには、ライブラリー・サーバーとデータベースの間に複数の接続を作成するか、クライアント・アプリケーションを使用している場合は、複数のクライアント・プロセスまたはスレッドを開始する必要があります。IBM WebSphere[®] Application Server などのアプリケーションは、プロセス、接続、およびセッションを処理します。

DKDatastoreICM における execute() および executeWithCallback() メソッドは、呼び出されると、自動的にデータベースへの別の接続を作成します。新規のデータベース接続は、その後、照会を実行するために使用されます。照会は、個別のデータベース接続を使用するため、他のコンテンツ・サーバー操作とは別のトランザクション有効範囲も持っています。データベースへの接続は、DKResultSetCursor がクローズされた時点でクローズされます (あるいは、プール化が使用可能な場合は、プールへ戻されます)。

アプリケーションにおけるトランザクション設計時の考慮事項

トランザクションがコミットされる前にクライアント・ノードまたはライブラリー・サーバーに障害が起きた場合、データベース・リカバリー機能がライブラリー・サーバー上のトランザクションを即時にロールバックします。障害が起きた際にリソース・マネージャーに対して行われた変更は、クライアント・ノードとリソ

ース・マネージャーの両方がアクティブであれば、即時に元に戻されます。クライアント・ノード自体に障害が起こった場合、リソース・マネージャーとライブラリー・サーバーの間の整合性を復元するため、リソース・マネージャーを非同期リカバリー・ユーティリティー (Asynchronous Recovery Utility) のサイクルで処理しなければなりません。ユーティリティーを実行する前は、サーバーがまだデータ保全性を持っています。影響を受けるのは、進行中の項目における操作で障害が起きた場合で、RM がリカバリーしないとリジェクトされてしまいます。オブジェクトの更新処理中に障害が起きると、最初の障害から回復するまで、その同じオブジェクトに別の更新を行うことはできません。

リソース・マネージャーに障害が起こったら、非同期リカバリー・ユーティリティーを実行して不整合を除去しなければなりません。OS/390 では、リソース・マネージャーはオブジェクト・アクセス方式 (OAM) などの固有のトランザクション機能を持ちます。この機能は、より適切にリカバリーするために使用されます。

明示的トランザクション使用時の注意

明示的トランザクションの場合は、DKDatastoreICM.startTransaction() と DKDatastoreICM.commit() を使用してトランザクション有効範囲を制御しますが、パーツを持つ Content Manager の文書を使用するアプリケーション開発時、および DKLobICM の作成、検索、更新、および削除 (CRUD) 操作の実行時には、注意してください。これらの操作を実行するときは、トランザクションの終了にできるだけ近いところで CRUD 操作を実行してください。また、トランザクションが長いとデータベースのロック問題が起きる可能性が高くなるので、トランザクションの時間を可能な限り短くする必要があります。

ロック問題は、項目の更新時に最も明らかになり、アプリケーションはトランザクションを即時にコミットしないように選択します。トランザクションがコミットされない限り、更新中の項目は、他のアプリケーションから参照できます。別のユーザーがその項目を使用または表示しようとする、更新トランザクションがコミットされるまで、そのユーザーはロックアウトされます。フォルダーに新規項目を作成する際にも、同じ問題 (データベースのロック) が起きます。他のユーザーからこのフォルダーを参照できるので、そのユーザーが新規項目を検索しようとする、そのユーザーはそのトランザクションがコミットされるまでロックアウトされます。トランザクションのコミットまでの時間が、ユーザーがロックアウトされる時間になります。

データベースのロックを回避する最も効果的なアプローチは、トランザクションのコミットを頻繁に行い、トランザクションの長期実行を回避することです。トランザクション内で CRUD 操作を実行しなければならない場合は、更新中の項目にアクセスする人がいないことを確認してから、これらの操作を実行することをお勧めします。

トランザクションにおけるチェックインおよびチェックアウトの使用

Content Manager は、項目上におけるチェックアウトおよびチェックイン操作をサポートします。チェックアウト操作は、項目への永続的書き込みロックを獲得するために呼び出されます。ユーザーが項目をチェックアウトすると、他のユーザーは同じ項目を検索および表示することはできませんが、更新することはできません。使用

するトランザクション・モード（暗黙的または明示的）に関係なく、項目を更新または再索引付けする前に、チェックアウト操作を呼び出す必要があります。項目の使用が終了したら、永続ロックを解除するためにチェックイン操作を呼び出し、その項目を他のユーザーが更新できるようにします。項目を作成した後、完全にその作業を完了するまで他のユーザーが変更できないように、それをチェックアウト状態で保持するオプションがあります。明示的なトランザクションを使用して項目をチェックアウト（またはチェックイン）した場合、トランザクションがロールバックされると、チェックアウトは取り消されます。暗黙的なトランザクションを使用して項目をチェックアウトすると、そのチェックアウトはコミットされます。項目のチェックインにチェックイン・オプションまたはメソッドを使用するかは、アプリケーション次第です。

トランザクションの処理

トランザクションの有効範囲は、クライアント API の呼び出しによって制御できますが、慎重に設計する必要があります。API 呼び出しのセットをトランザクションにグループ化するには、以下のステップを実行して、明示的にトランザクションを作成する必要があります。

1. DKDatastoreICM クラスの `startTransaction()` メソッドを呼び出します。
DKDatastoreICM メソッドを使用して、トランザクションのすべてのステップを実行します。
2. トランザクションに組み込むすべての API を、その呼び出し順に呼び出します。
3. `commit` または `rollback` メソッドを呼び出して、トランザクションを終了します。

`startTransaction()` と `commit()` または `rollback()` のいずれかの間で行われるすべての API 呼び出しは、1 つのトランザクションとして処理されます。

特に明示されない限り、すべての API をトランザクションに組み込むことができます。詳細については、「オンライン API 解説書」を参照してください。管理 API には、明示的なトランザクションに組み込めないものもあります。例えば、項目タイプを定義または更新するメソッドです。

項目の作成および更新に関連した Content Manager トランザクションに含まれるクラス・メソッドのリストを以下に示します。

DKDatastoreICM.startTransaction()

明示的にトランザクションを開始します。

DKDatastoreICM.commit()

トランザクションでの変更を、データベースにコミットします。

DKDatastoreICM.rollback()

トランザクションでの変更を、データベースからロールバックまたは除去します。

DKDatastoreICM.checkOut()

項目に対する永続的書き込みロックを獲得します。

DKDatastoreICM.checkIn()

以前に獲得された永続的書き込みロックを解放します。

DKDatastoreICM.add()

データベース内に新規項目を作成します。

DKDatastoreICM.updateObject()

項目を更新します。項目は、このメソッドを呼び出す前にチェックアウトされていなければなりません。

DKDatastoreICM.retrieveObject()

データベースから項目を取り出します。

DKDatastoreICM.deleteObject()

データベースから項目を削除します。

DKDatastoreICM.moveObject()

項目を再索引付けします。ある項目タイプから別の項目タイプに、項目を移動します。項目は、このメソッドを呼び出す前にチェックアウトされていなければなりません。

トランザクションに関する情報源としては、SItemUpdateICM サンプルも非常に役立ちます。

プロセスにおける文書ルーティング

Content Manager には、ビジネス・プロセスにおける文書ルーティングを行うための統合文書ルーティング・サービスがあります。文書ルーティング API により、文書ルーティングを使用した新規アプリケーションの作成、または既存のアプリケーションへの文書ルーティング機能の追加を行うことができます。文書ルーティングは、以下の機能を提供します。

- 文書ルーティング・プロセスのすべての項目を同期する (文書ルーティング機能は Content Manager トランザクションに組み込まれている)。
- ユーザーがアクセスできる作業のみの表示。
- 文書作成、修正、およびルーティングの記録を含む単一の監査証跡。

文書ルーティングの基本的な概念と用語については、「システム管理ガイド」を参照してください。また、サンプルも参照してください。サンプルは、文書ルーティングの詳しい情報源として非常に役立ちます。

文書ルーティング・プロセスの概要

文書ルーティングは、プロセス、作業ノード、ワーク・リスト、および作業パッケージから構成されます。システム管理者は、作業ノード、プロセス、およびワーク・リストをシステム管理クライアントを使用して作成します。1 つのプロセスは、複数の作業ノードから構成されます。プロセス内のそれぞれの作業ノードは、プロセス内の独立したステップです。複数の方向に分岐するプロセスを作成することができます。ユーザーは、作業ノードが次にどの分岐に進むかを決定します。ユーザーは、システム管理者が定義した選択可能項目のリストから選択することができます。作業ノードを定義する際は、サーバー出口を定義することができます。作業ノードに入るため、作業ノードを終了するため、およびユーザーに多重定義制限を通知するためにサーバー出口を定義することができます。プロセスが開始される

と、作業パッケージが作成されます。作業パッケージはルーティング・エレメントであり、作業の属性を含みます。作業パッケージの属性は、項目の PID、優先順位、所有者などから構成されます。

コレクション・ポイントは、追加機能を持つ作業ノードです。コレクション・ポイント・ノードにおける作業パッケージは、指定項目タイプの指定数の項目が指定フォルダーに存在する場合に、プロセスの次の作業ノードへと進みます。ワーク・リストは、ユーザーに割り当てられた作業パッケージを定義します。ユーザーは、1 つまたは複数のワーク・リストを持つことができます。各ワーク・リストには、1 つまたは複数の作業ノードを組み込むことができます。ワーク・リスト内の作業パッケージの順序は、優先順位、または日付によって指定することができます。また、ワーク・リスト内の作業ノードの順序も定義することができます。

ワーク・リストを検索する場合、結果にフィルターを掛け、延期された作業を含めるか除外することができます。作業パッケージを通知状態に置くこともできます。通知 状態は、作業パッケージが、管理者の指定した時間より長くノードに置かれている状態です。1 つの作業ノードを複数のワーク・リスト内に置くことができます。ワーク・リストに戻されるパッケージの数は、システム管理者によって定義されます。

文書ルーティングを使用して実行できる基本操作を以下に示します。

- プロセスの開始
- プロセスの終了
- プロセスの続行
- プロセスの延期
- プロセスの再開
- ワーク・リストからの作業の取得
- ワーク・リストからの次項目の取得
- プロセスの定義、更新、および削除
- 作業ノードの定義、更新、および削除
- ワーク・リストの定義、更新、および削除

文書ルーティング・プロセスのセットアップ

アプリケーションへの文書ルーティング機能をインプリメントする際、9 つの API を使用することができます。これら API およびメソッドについての詳細は、「オンライン API 解説書」を参照してください。これら 9 つの文書ルーティング API は以下のとおりです。

DKDocRoutingServiceICM

このクラスは、プロセスの管理メソッド (start、 terminate、 continue、 suspend、 および resume) を提供します。

DKDocRoutingServiceMgmtICM

このクラスは、ヘルパー・クラスの管理メソッド (DKProcessICM、 DKWorkNodeICM、 および DKWorkListICM) を提供します。

DKDocRoutingServiceMgmtICM オブジェクトには DKDocRoutingServiceICM オブジェクトからアクセスすることができます。

DKProcessICM

このクラスは、ライブラリー・サーバー内のプロセスを表します。

DKWorkNodeICM

このクラスはライブラリー・サーバー内の作業ノードを表します。

DKWorkListICM

このクラスは、ライブラリー・サーバー内のワーク・リストを表します。

DKRouteListEntryICM

このクラスは、プロセスが取る経路 (from、to) を定義します。 プロセス・オブジェクト (つまり、DKProcessICM) は、経路エントリー・オブジェクト (つまり、DKRouteListEntryICM) のコレクションを含みます。

DKCollectionResumeListEntryICM

このクラスは、作業ノードの再開リスト内のエントリーを表します。作業ノードには、DKCollectionResumeListEntryICM オブジェクトのコレクションを含めることができます。

DKWorkPackageICM

このクラスは、ライブラリー・サーバー内の作業パッケージを表します。プロセスが開始されると、作業パッケージが作成されます。

DKResumeListEntryICM

このクラスは再開リストを表します。作業パッケージには、再開リストのコレクションを含めることができます。

文書ルーティング・サービス・オブジェクトの作成

以下の例は、文書ルーティング・オブジェクトの作成方法を示します。

Java

```
//The DKDocRoutingServiceMgmtICM object is a helper class that provides
// methods to manage DKProcessICM, DKWorkNodeICM, and DKWorkListICM
// and the meta-data required to define these objects
DKDocRoutingServiceMgmtICM routingMgmt = new
    DKDocRoutingServiceMgmtICM(dsICM);

//The DKDocRoutingServiceICM class provides the core routing services
//like starting, terminating, continuing, suspending, and resuming a process.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);
```

C++

```
//provides methods to manage DKProcessICM, DKWorkNodeICM,  
//and DKWorkListICM and the meta-data required to define these objects  
DKDocRoutingServiceMgmtICM* routingMgmt = new  
    DKDocRoutingServiceMgmtICM(dsICM);  
  
//The DKDocRoutingServiceICM class provides the core routing services  
// such as starting, terminating, continuing, suspending, and resuming  
//a process.  
DKDocRoutingServiceICM* routingService = new  
    DKDocRoutingServiceICM(dsICM);
```

完全なサンプルについては、SDocRoutingDefinitionCreationICM サンプルを参照してください。

新規の標準作業ノードの定義

作業ノードは、文書ルーティングのプロセス定義におけるステップです。いつでもユーザー・アプリケーションを使用して終了できます。その終了基準の妥当性検査については、ユーザー・アプリケーションに任されています。

Java

```
// Create new Work Node Object.  
DKWorkNodeICM workNode1 = new DKWorkNodeICM();  
//Choose a Name that is 15 characters or less length  
workNode1.setName("S_fillClaim");  
//Choose a Description for more information than the name.  
workNode1.setDescription("Claimant Fills Out Claim");  
// Sets the value of the maximum time that an item can spend at this  
// work node (in minutes).  
workNode1.setTimeLimit(100);  
// Specify max number of work packages that can be at this node  
//at any given time  
workNode1.setOverloadLimit(200);  
  
// Set the type to be a regular work node  
workNode1.setType(0);  
  
//Add the new work node definition to the document routing  
//managment object  
routingMgmt.add(workNode1);
```


C++

```
// Create new Work Node Object.
DKWorkNodeICM* workNode1 = new DKWorkNodeICM();
// Choose a Name that is 15 characters or less length
workNode1->setName("ValidateCreditCard");
// Choose a Description for more information than the name.
workNode1->setDescription("Buyer's credit card is validated with
                           the credit card agency");

// Sets the value of the maximum time that an item can spend at
//this work node (in minutes).
workNode1->setTimeLimit(100);
// Specify max number of work packages that can be at this node
// at any given time
workNode1->setOverloadLimit(200);

// Set the type to be a regular work node
workNode1->setType(0);

//Add the new work node definition to the document routing
//management object
routingMgmt->add(workNode1);

// Free memory. This object will no longer be needed.
delete(workNode1);
```

作業ノードの完全な作成例については、SDocRoutingDefinitionCreationICM サンプルを参照してください。

作業ノードのリスト作成

listWorkNodeNames メソッドは、ライブラリー・サーバー内のすべての作業ノード名をリストし、listWorkNodes メソッドは、ライブラリー・サーバー内の作業ノードを表す DKWorkNodeICM オブジェクトのコレクションを戻します。

Java

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new
    DKDocRoutingServiceMgmtICM(dsICM);

// Obtain all Work Nodes in the System.
dkCollection workNodes = routingMgmt.listWorkNodes();
System.out.println("Work Nodes in System: (" + workNodes.cardinality() + ")");
dkIterator iter = workNodes.createIterator();
while(iter.more())
{
    DKWorkNodeICM workNode = (DKWorkNodeICM) iter.next();
    if(workNode.getType() == 0)
        System.out.println(" Normal Node - " + workNode.getName() + ": "
            + workNode.getDescription());
    else
        System.out.println(" Collection Pt - " + workNode.getName() + ": "
            + workNode.getDescription());
}
```

C++

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the collection containing all the work nodes in the system.
dkCollection* workNodes = routingMgmt->listWorkNodes();
if (workNodes && (workNodes->cardinality()>0) )
{
    cout << "Work Nodes in System: (" << workNodes->cardinality() << ")"
        << endl;
    dkIterator* iter = workNodes->createIterator();
    while(iter->more())
    {
        DKWorkNodeICM* workNode = (DKWorkNodeICM*) iter->next()->value();
        if(workNode->getType()==0)
        {
            cout << " Normal Node - " << workNode->getName() << ": " <<
                workNode->getDescription() << endl;
        }
        else
        {
            cout << " Collection Pt - " << workNode->getName() << ": " <<
                workNode->getDescription() << endl;
        }
        delete(workNode);
    }
    delete(iter);
    delete(workNodes);
}
delete(routingMgmt);
```

作業ノードのリスト作成について詳しくは、 SDocRoutingListingICM サンプルを参照してください。

新規コレクション・ポイントの定義

コレクション・ポイントは、特にルーティング・フォルダーに適用されるシステム定義の終了基準を持つ作業ノードです。プロセスがこのポイントを通過して再開または進めることができるように、満たさなければならない要件のセットを指定することができます。

Java

```
// Create a new Work Node Object. This will be
//the collection point
DKWorkNodeICM collectionPoint = new DKWorkNodeICM();

// Choose a Name with 15 characters or less.
collectionPoint.setName("S_gatherAll");

// Choose a Description for more information than the name.
collectionPoint.setDescription("Gather Claim,Police Report,Policy,& Photos");

// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
collectionPoint.setTimeLimit(100);

// Specify max number of work packages that can be at this node.
collectionPoint.setOverloadLimit(200);
// Set the type of node to be a collection point.
collectionPoint.setType(1);

// Create the "Resume" List, which is the list of document types
//that the process must wait for before moving on to the next node.
//A list will be created to hold "resume entries" which are descriptions
//of requirements that must be met before the process can move on.
// Create a List/Collection to hold all Resume Entries.
dkCollection resumeList = new DKSequentialCollection();
// Create as many requirements, or "Resume List Entries", which
//specify what Item Types it must wait for. The process cannot
//pass this collection point unless the specified number of Item
//(DDO) of the specified Item Type reaches this collection point.
DKCollectionResumeListEntryICM resumeRequirement = new
    DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement.setFolderItemTypeName("S_simple");
// Make the collection wait for an Item of the specified Item Type
//to be added to the folder before proceeding.
resumeRequirement.setRequiredItemTypeName("S_autoClaim");
// Specify the number of Items of the specified Item Type that it
//must wait for.
resumeRequirement.setQuantityNeeded(1);
// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList.addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement.setFolderItemTypeName("S_simple");
resumeRequirement.setRequiredItemTypeName("S_policeReport");
resumeRequirement.setQuantityNeeded(1);
// When all requirements (resume list entries) have been added to the
//list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint.setCollectionResumeList(resumeList);
// Add the new collection point definition to the document routing
// managment object
routingMgmt.add(collectionPoint);
```

C++

```
// Create a new Work Node Object.This will be the collection point
DKWorkNodeICM* collectionPoint = new DKWorkNodeICM();
// Choose a Name with 15 characters or less.
collectionPoint->setName("GatherOrderDetails");
// Choose a Description for more information than the name.
collectionPoint->setDescription("Gather all the information related to the
    order, shipping mechanism and shipping address");
// Sets the value of the maximum time that an item can spend at this
//work node (in minutes).
collectionPoint->setTimeLimit(100);
// Specify max number of work packages that can be at this node.
collectionPoint->setOverloadLimit(200);

// Set the type of node to be a collection point.
collectionPoint->setType(1);
// Create the "Resume" List, which is the list of document types
//that the process must wait for before moving on to the next node.
//A list will be created to hold "resume entries" which are descriptions
//of requirements that must be met before the process may move on.

// Create a List / Collection to hold all Resume Entries.
dkCollection* resumeList = new DKSequentialCollection();

// Create as many requirements, or "Resume List Entries", which specify
//what Item Types it must wait for. The process cannot pass this
//collectionpoint unless the specified number of Item (DDO) of the
//specified Item Type reaches this collection point.
DKCollectionResumeListEntryICM* resumeRequirement = new
    DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement->setFolderItemTypeName("book");

// Make the collection wait for an Item of the specified Item Type to
//be added to the folder before proceeding.
resumeRequirement->setRequiredItemTypeName("AnItemType");

//Specify the number of Items of the specified Item Type that
//it must wait for.
resumeRequirement->setQuantityNeeded(1);

// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList->addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement->setFolderItemTypeName("book");
resumeRequirement->setRequiredItemTypeName("AnotherItemType");
resumeRequirement->setQuantityNeeded(1);
resumeList->addElement(resumeRequirement);

// When all requirements (resume list entries) have been added to
//the list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint->setCollectionResumeList(resumeList);
// Add the new collection point definition to the document routing
// management object
routingMgmt->add(collectionPoint);

//Free the memory associated with this collection point
delete(collectionPoint);
```

コレクション・ポイントの定義について詳しくは、
SDocRoutingDefinitionCreationICM サンプルを参照してください。

ワーク・リストの定義

ワーク・リストは、1 つまたは複数の作業ノードで構成されます。ユーザーはその作業ノードから作業パッケージのリストまたは「次の」作業パッケージを入手できます。1 つの作業ノードを複数のワーク・リスト内に置くことができます。ワーク・リストを使用することにより、管理者またはユーザー・アプリケーションは、エンド・ユーザーに連絡することなく作業割り当てを動的に変更することができます。

Java

```
// Create a new work list.
DKWorkListICM workList = new DKWorkListICM();
// Choose a name of 15 characters or less.
workList.setName("S_fillClaimWL");
workList.setDescription("Work List Covering Fill/Submit Claim Work Node.");
//Specify that work packages returned by the work list will be sorted by time
workList.setSelectionOrder(DKConstantICM.DK_ICM_DR_SELECTION_ORDER_TIME);
//Specify that the work packages returned will be the one that are not
// in the suspend state
workList.setSelectionFilterOnSuspend
    (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList.setSelectionFilterOnNotify
    (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

// Specify that at most 100 work packages should be listed in
// this work list
workList.setMaxResult(100);
String[] wnNames = {"S_fillClaim"};
workList.setWorkNodeNames(wnNames);

//Add the new work list definition to the document routing
//management object
routingMgmt.add(workList);
```

C++

```
// Create a new worklist.
DKWorkListICM* workList = new DKWorkListICM();
//Choose a name of 15 characters or less.
workList->setName("ValidateWorkList");
workList->setDescription("worklist Covering the credit card
    validation work node.");

//Specify that work packages returned by the worklist will be
//sorted by time
workList->setSelectionOrder(DK_ICM_DR_SELECTION_ORDER_TIME);

//Specify that the work packages returned will be the one that are not
//in the suspend state
workList->setSelectionFilterOnSuspend(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList->setSelectionFilterOnNotify(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that at most 100 work packages should be listed in this worklist
workList->setMaxResult(100);

DKString* wnNames = new DKString[1];
wnNames[0] = "ValidateCreditCard";
//Add the work node to the worklist
workList->setWorkNodeNames(wnNames,1);

//Add the new worklist definition to the document routing management
//object
routingMgmt->add(workList);

//Free the memory associated with this worklist
delete(workList);
```

ワーク・リストの定義について詳しくは、SDocRoutingDefinitionCreationICM サンプルを参照してください。

ワーク・リストのリスト作成

listWorkListNames メソッドは、ライブラリー・サーバー内のすべてのワーク・リスト名をリストし、listWorkLists メソッドは、ライブラリー・サーバー内のワーク・リストを表す DKWorkListICM オブジェクトのコレクションを戻します。

Java

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain all Work Lists in the System.
dkCollection workLists = routingMgmt.listWorkLists();
System.out.println("Work Lists in System: (" +workLists.cardinality()+")");
dkIterator iter = workLists.createIterator();
while(iter.more())
{
    DKWorkListICM workList = (DKWorkListICM) iter.next();
    System.out.println("    - " +workList.getName()+":
        "+workList.getDescription());
}
```

C++

```
dkCollection* workLists = routingMgmt->listWorkLists(); // Obtain all
// Work Lists in the System.

if (workLists && (workLists->cardinality())>0 )
{
    cout<<"Work Lists in System: ("<<workLists->cardinality()<<")"<<endl;
    dkIterator* iter = workLists->createIterator();
    while(iter->more()){
        DKWorkListICM* workList = (DKWorkListICM*) iter->next()->value();
        cout << " - " << workList->getName() << ": "
            << workList->getDescription() << endl;
        delete(workList); // Free Memory
    }
    delete(iter); // Free Memory
    delete(workLists);
}
```

完全な例については、SDocRoutingListingICM サンプルを参照してください。

新規プロセスおよび関連経路の定義

文書ルーティング・プロセスは、作業パッケージがたどる定義済みの経路です。複数のルーティング・プロセスに複数のノードを再利用でき、またノード間で複数の経路を使用することもできます。

Java

```
// Create a new Process Definition
DKProcessICM process = new DKProcessICM();
process.setName("S_claimProcess");
process.setDescription("Process for an Insurance Claim");

// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries. A simple route
//between two work nodes is specified by associating a 'From' work
//node and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes. A specific route may be selected
// by a user-defined "selection" keyword. Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.
// Create a new connection between two nodes.
DKRouteListEntryICM nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute.setFrom(DKConstantICM.DK_ICM_DR_START_NODE);
nodeRoute.setTo("S_fillClaim");
// Choose any user-defined name for an action that will make the
//transition from the 1st node to the second
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_fillClaim");
nodeRoute.setTo("S_gatherAll");
// Choose any user-defined name for an action that will make the
// transition take place.
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_gatherAll");
nodeRoute.setTo(DKConstantICM.DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make
//the transition take place.
nodeRoute.setSelection("Complete");
// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
// Set the route in the process.
process.setRoute(routes);
// Add the process to the routing Management.
routingMgmt.add(process);
```


C++

```
//A document routing process is the defined routes that a work
//package being routed will follow. Multiple routing processes may
//re-use the same nodes and multiple routes between nodes may be used.

// Create a new Process Definition
DKProcessICM* process = new DKProcessICM();
process->setName("Buy_Book");
process->setDescription("Purchase a book online");
// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection* routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries. A simple route
// between two work nodes is specified by associating a 'From' work node
// and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes. A specific route may be selected
// by a user-defined "selection" keyword. Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.

// Create a new connection between two nodes.
DKRouteListEntryICM* nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute->setFrom(DK_ICM_DR_START_NODE);
nodeRoute->setTo("ValidateCreditCard");
// Choose any user-defined name for an action that will make the transition
// from the 1st node to the 2nd
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("ValidateCreditCard");
nodeRoute->setTo("GatherShippingDetails");

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("GatherOrderDetails");
nodeRoute->setTo(DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Complete");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

// Set the route in the process.
process->setRoute(routes);

// Add the process to the routing Management.
routingMgmt->add(process);

delete(process);
```

完全な例については、SDocRoutingDefinitionCreationICM サンプルを参照してください。

文書ルーティング・プロセスの開始

以下の例は、文書ルーティング・プロセスの開始方法を示します。

Java

```
//First create a document or folder that will be routed.
//An item type of name "s_simple" must be pre-defined before a
// DDO of that name can be created.

DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder.add();

//Create the core document routing service object.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "S_claimProcess" (which must be pre-defined.

//The PID string of the work pkg that will be routed is returned by
//this call.
String workPackagePidStr = routingService.startProcess
("S_claimProcess", ddoFolder.getPidObject().pidString(),1,"icmadmin");
```

C++

```
//First create a document or folder that will be routed.
//An item type of name "book" must be pre-defined before a DDO of
// that name can be created.
DKDDO* ddoFolder = dsICM->createDDO("book", DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder->add();

//Set the priority for this document routing process
int Priority = 1;

//Create the core document routing service object.
DKDocRoutingServiceICM* routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "Buy_Book" (which must be pre-defined.
//The PID string of the work pkg that will be routed is returned by
//this call.
DKString workPackagePidStr=routingService->startProcess("Buy_Book",
((DKPidICM*)
    ddoFolder->getPidObject())->pidString(), Priority, "icmadmin");
```

完全な例については、SDocRoutingProcessingICM サンプルを参照してください。

プロセスの終了

プロセスは、エンド・ノードに到達する前に明示的に終了させることができます。プロセスを終了すると、経路指定中の作業パッケージがシステムから除去されます。プロセスを終了するには、プロセス・インスタンスによって経路指定されている作業パッケージの PID スtringが分かっている必要があります。

Java

```
routingService.terminateProcess(workPackagePidStr);
```

C++

```
routingService->terminateProcess(workPackagePidStr);
```

プロセスの終了について詳しくは、SDocRoutingProcessingICM サンプルを参照してください。

プロセスの続行

continueProcess() メソッドは、指定された作業パッケージの項目 PID により参照される項目を、現行の作業ノードから、選択で決定した次の作業ノードへとルーティングします。ライブラリー・サーバーから指定された作業パッケージが除去され、指定された所有者に対し新規の作業パッケージが作成されます。項目 PID によって参照される項目がチェックインされます (チェックアウトされていた場合)。新規作業パッケージの PID が戻されます。プロセスが終了していた場合は、ヌルが戻されます。

以下のコードの断片では、現行の作業ノードから次の作業ノードに遷移を起こす選択名は「Continue」です。メソッド呼び出しには、現行作業パッケージの作業パッケージ PID Stringを指定する点に注意してください。メソッド呼び出しからは、新規作業パッケージの PID Stringが戻ります。

Java

```
workPackagePidStr = routingService.continueProcess  
(workPackagePidStr, "Continue", "icmadmin");
```

C++

```
char * userName = "icmadmin";  
  
workPackagePidStr = routingService->continueProcess(workPackagePidStr,  
"Continue", userName);
```

完全な例については、SDocRoutingProcessingICM サンプルを参照してください。

プロセスの延期

文書ルーティング・プロセスのインスタンスを、ある時間 (分単位で)、またはインスタンスを再開するために満たさなければならない要件のセットを保留している間、延期することができます。これは、プログラミング環境内のプロセスおよびスレッドとは関係ありません。C++ 実行時環境におけるスレッドまたはプロセスが停止することはありません。

Java

```
dkCollection requirements = new DKSequentialCollection();
//Process will be suspended for 2 minutes.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```

C++

```
dkCollection * requirements = new DKSequentialCollection();
//If no requirements are to be provided and the process is only to be
//suspended for a fixed period of time, the user can also pass in a
//NULL collection to this method.
//dkCollection * requirements = NULL;

//Process will be suspended for 2 minutes.
routingService->suspendProcess(workPackagePidStr, 2, requirements);
delete(requirements);
```

完全な例については、SDocRoutingProcessingICM サンプルを参照してください。

プロセスの再開

延期されたプロセス (延期状態のプロセス) は明示的に再開でき、指定された期間満了後、または定義された要件が満たされた後、暗黙的に再開することもできます。これにより、プロセスは延期状態から抜けて通常のコアを再開します。
`resumeProcess` メソッドは、延期期間が指定された期間に達する前、また再開リストの条件が満たされる前に、指定された作業パッケージの延期フラグを `false` にリセットします。関連する作業項目のルーティングまたはチェックアウトは行われません。

Java

```
routingService.resumeProcess(workPackagePidStr);
```

C++

```
routingService->resumeProcess(workPackagePidStr);
```

プロセスの再開について詳しくは、SDocRoutingProcessingICM サンプルを参照してください。

ワーク・リスト内の作業パッケージ永続 ID スtringのリスト作成

以下のコード例は、指定されたワーク・リスト内のすべての作業パッケージの PID Stringのリストを作成する方法を示します。

Java

```
String[] workPackagePIDs =
    routingService.listWorkPackagePidStrings(workListName,processOwner);

// Print Work Package PIDs
System.out.println("Work Packages in Work List: (+workPackagePIDs.length+));
for(int i=0; i< workPackagePIDs.length; i++)
    System.out.println( " - PID: +workPackagePIDs[i]);
```

C++

```
long arraySize = -1; // Size to be set by the API.
DKString* workPackagePIDs = routingService->
    listWorkPackagePidStrings("workListName",processOwner,arraySize);

// Print Work Package PIDs
cout << "Work Packages in Work List: (" << arraySize << ")" << endl;
for(int i=0; i< arraySize; i++)
    cout << " - PID: " << workPackagePIDs[i] << endl;

delete[] workPackagePIDs; // Free Memory
```

完全な例については、SDocRoutingListingICM サンプルを参照してください。

作業パッケージ情報の検索

文書ルーティング・プロセスのインスタンスが進行中の場合、作業パッケージは、項目 (項目タイプのインスタンス) がルーティング・プロセスを通じて移動する手段となります。作業パッケージには、プロセスと移動中の項目に関する必要な情報がすべて含まれています。作業パッケージは必要に応じてアプリケーションが使用し操作するオブジェクトです。

retrieveWorkPackage メソッドは、指定された作業パッケージ PID (wpPidStringStr) によって参照された DKWorkPackageICM オブジェクトを返します。

Java

```
//Use an established document routing service
//Specifying false in this method call makes sure that the work package
// is not checked out
DKWorkPackageICM workPackage =
    routingService.retrieveWorkPackage(workPackagePidStr,false);
System.out.println("-----");
System.out.println("                Work Package");
System.out.println("-----");
System.out.println(" Process Name: " + workPackage.getProcessName());
System.out.println(" work Node Name: " + workPackage.getWorkNodeName());
System.out.println(" Owner: " + workPackage.getOwner());
System.out.println(" Priority: " + workPackage.getPriority());
System.out.println(" User Last Moved: " + workPackage.getUserLastMoved());
System.out.println(" Time Last Moved: " + workPackage.getTimeLastMoved());
System.out.println(" Suspend State: " + workPackage.getSuspendState());
System.out.println(" Notify State: " + workPackage.getNotifyState());
System.out.println(" Notify Time: " + workPackage.getNotifyTime());
System.out.println(" Resume Time: " + workPackage.getResumeTime());
System.out.println("Work Package Pid: " + workPackage.getPidString());
System.out.println(" Item Pid: " + workPackage.getItemPidString());
```

C++

```
cout << "-----" << endl;
cout << " Work Package" << endl;
cout << "-----" << endl;
cout << " Process Name: " << workPackage->getProcessName() << endl;
cout << " work Node Name: " << workPackage->getWorkNodeName() << endl;
cout << " Owner: " << workPackage->getOwner() << endl;
cout << " Priority: " << workPackage->getPriority() << endl;
cout << " User Last Moved: " << workPackage->getUserLastMoved() << endl;
cout << " Time Last Moved: " << workPackage->getTimeLastMoved() << endl;
cout << " Suspend State: " << workPackage->getSuspendState() << endl;
cout << " Notify State: " << workPackage->getNotifyState() << endl;
cout << " Notify Time: " << workPackage->getNotifyTime() << endl;
cout << " Resume Time: " << workPackage->getResumeTime() << endl;
cout << "Work Package Pid: " << workPackage->getPidString() << endl;
cout << " Item Pid: " << workPackage->getItemPidString() << endl;
```

完全な例については、SDocRoutingProcessingICM サンプルを参照してください。

文書ルーティング・プロセスのリスト作成

以下の例は、文書ルーティング・プロセスのリスト作成方法を示します。

Java

```
//The listProcessNames method lists all process names in the
//Library Server, and the listProcesses method returns a collection
//of DKProcessICM objects representing a process in the Library Server.
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain the list of all running document routing processes
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain list of all routing processes running.
dkCollection processes = routingMgmt.listProcesses();
System.out.println("Running Processes: (" + processes.cardinality() + ")");

dkIterator iter = processes.createIterator();
while(iter.more())
{
    // Move pointer to next element and obtain that next element.
    DKProcessICM proc = (DKProcessICM) iter.next();
    System.out.println(" - " + proc.getName() + ": " + proc.getDescription());
}
```

C++

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the list of all running document routing processes
dkCollection* processes = routingMgmt->listProcesses();
if (processes && (processes->cardinality() > 0))
{
    cout << "Running Processes: (" << processes->cardinality() << ")"
        << endl;
    dkIterator* iter = processes->createIterator();
    while(iter->more())
    {
        DKProcessICM* proc = (DKProcessICM*) iter->next()->value();
        cout << " - " << proc->getName() << ": "
            << proc->getDescription() << endl;
        delete(proc);
    }
    delete(iter);
    delete(processes);
}
delete(routingMgmt);
```

SDocRoutingListingICM サンプルには印刷機能が示されています。

随時ルーティング

以下は、随時ルーティングのプロシーチャー例です。この例では、作業ノード、プロセス、およびワーク・リストをセットアップするためにシステム管理クライアントが使用されます。

1. N1 および N2 など、2 つの作業ノードを作成します。

2. 作業ノード N1 を持つ P1、および作業ノード N2 を持つ P2 の、2 つの単一ノード・プロセスを作成します。

P1 は、以下ようになります。

始点:	アクション:	終点:
START	継続	N1
N1	継続	END

P2 は、以下ようになります。

始点:	アクション:	終点:
START	継続	N2
N2	継続	END

3. 作業ノード N1 を持つ WL1、および作業ノード N2 を持つ WL2 の、2 つのワーク・リストを作成します。

実行時に、以下のステップを実行して、随時ルーティングをインプリメントします。

1. 文書 PID (ABC など) を使用してプロセス P1 を開始します。作業パッケージ WP1 が作成されます。ワーク・リスト WL1 は、作業ノード N1 で作業パッケージ WP1 を表示します。
2. 文書 ABC をプロセス P1 からプロセス P2 へ移動するには、作業パッケージ WP1 を終了し、同じ文書 (ABC) を使用してプロセス P2 を開始します。作業パッケージ WP2 が作成されます。

ワーク・リスト WL2 は、作業ノード N2 で作業パッケージ WP2 を表示します。

その他の例については、SDocRoutingDefinitionCreationICM サンプルを参照してください。

文書ルーティングの照会例

このセクションには、照会の例が含まれています。照会の作成についての詳細は、201 ページの『照会言語について』を参照してください。

- 例 1** 車関係の文書を検索し、関連したアクティブな作業パッケージのみを戻します。

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =  
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =  
0]
```

- 例 2** 車関係の文書を検索し、"AccidentInvestigation" プロセス内にある関連した作業パッケージを戻します。

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =  
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =  
/ROUTINGPROCESS[@PROCESSNAME =  
"AccidentInvestigation"]/@ITEMID]
```

- 例 3** 名前が "Honda" である車関係の文書を検索し、"AccidentInvestigation" プロセス内にある関連した作業パッケージを戻します。


```

/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]

```

例 4 車関係の文書を検索し、“AccidentInvestigation” プロセスの “UnderReview” ステップにある関連した作業パッケージを戻します。

```

/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME = "UnderReview"]

```

例 5 車関係の文書を検索し、“AccidentInvestigation” プロセスの “UnderReview” ステップにある延期された作業パッケージのみを戻します。

```

/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]

```

文書ルーティングの特権の認可

文書ルーティング操作を実行するには、ユーザーに適切な特権がなければなりません。文書ルーティングに関連する特権を、以下の表にリストします。項目の一般的な特権は、プロセス、作業ノード、およびワーク・リストに適用可能です。

表 16. 文書ルーティングの特権

特権	説明	関連 API
ICM_PRIV_ITEM_UPDATE_WORK	作業パッケージに対して以下のことを行うことがユーザーに許可されているかどうかを確認するために使用されます。 優先順位の設定 所有者の設定 再開リストの設定 延期期間の設定	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM_ROUTE_START	プロセスの開始がユーザーに許可されているかどうかを確認するために使用されます。	startProcess
ICM_PRIV_ITEM_ROUTE_END	プロセスの終了がユーザーに許可されているかどうかを確認するために使用されます。	terminateProcess
ICM_PRIV_ITEM_GET_WORKLIST	ワーク・リストから作業パッケージ数を入手することがユーザーに許可されているかどうかを確認するために使用されます。	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM_GET_WORK	作業パッケージの入手がユーザーに許可されているかどうかを確認するために使用されます。	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage

表 16. 文書ルーティングの特権 (続き)

特権	説明	関連 API
ICM_PRIV_ITEM _GET_ASGN_WORK	別のユーザーが所有する作業パッケージの入手が、ユーザーに許可されているかどうかを確認するために使用されます。	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _ROUTE	作業パッケージのルーティングがユーザーに許可されているかどうかを確認するために使用されます。	continueProcess

文書ルーティング用のアクセス制御リストの使用

ACL が、プロセス、作業ノード、ワーク・リストなどの文書ルーティング・エンティティに対して定義されると、エンティティ上で許可される操作は影響を受けます。文書ルーティング・エンティティおよびその関連する特権に ACL が与える影響を、以下の表にリストします。

表 17. アクセス制御リストおよび文書ルーティング

特権	説明	関連 API
プロセス	startProcess	ICM_PRIV_ITEM_ROUTE_START
作業ノード	continueProcess suspendProcess resumeProcess terminateProcess setWorkPackagePriority setWorkPackageOwner	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK
ワーク・リスト	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings checkOutItemInWorkPackage	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK

文書ルーティングの定数

DKConstantICM 内に文書ルーティング定数を定義します。以下は、文書ルーティング定数のリストです。

- public final static int DK_ICM_DR_SELECTION_FILTER_NO = 0;
- public final static in DK_ICM_DR_SELECTION_FILTER_YES = 1;
- public final static int DK_ICM_DR_SELECTION_FILTER_EITHER = 2;
- public final static int DK_ICM_DR_SELECTION_ORDER_PRIORITY = 0;
- public final static int DK_ICM_DR_SELECTION_ORDER_TIME = 1;
- public final static int DK_ICM_DR_MAX_RESULT_ALL = 0;

- `public final static String DK_ICM_DR_START_NODE = "START";`
- `public final static String DK_ICM_DR_END_NODE = "END";`

その他のコンテンツ・サーバーの使用

アプリケーション内でコンテンツ・サーバーに適したコンテンツ・サーバーを定義する場合、dkDatastore クラスを使用します。そのコンテンツ・サーバーが Enterprise Information Portal に対する基本インターフェースになります。各コンテンツ・サーバーには別個のコンテンツ・サーバー・クラスがあります。

コンテンツ・サーバーを作成するには、DKDatastorexx クラスを使います。ここで、xx は特定のコンテンツ・サーバーを示します。 41 ページの表 7 に、それらのクラスを示します。

表 18. サーバー・タイプおよびクラス名用語

コンテンツ・サーバー	クラス名
Content Manager バージョン 8.2	DKDatastoreICM
旧バージョンの Content Manager	DKDatastoreDL
Content Manager OnDemand	DKDatastoreOD
Content Manager for AS/400 (VisualInfo for AS/400)	DKDatastoreV4
Content Manager ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
Panagon Image Services (FileNET)	DKDatastoreFN
リレーショナル・データベース	DKDatastoreDB2、DKDatastoreJDBC (Java 用)、DKDatastoreODBC

コンテンツ・サーバー用にコンテンツ・サーバーを作成する場合には、以下のクラスおよびインターフェースをそれぞれインプリメントします。

dkDatastore

コンテンツ・サーバーを表し、接続、通信、およびコンテンツ・サーバー・コマンドの実行を管理します。 dkDatastore は、照会マネージャー・クラスの抽象バージョンです。これは、evaluate メソッドをサポートしています。

dkDatastoreDef

コンテンツ・サーバーに保管されている項目にアクセスするメソッドを使用します。また、そのエンティティの作成、リスト作成、および削除も行います。これは、dkEntityDefs のコレクションを保守します。このインターフェースの具象クラスの例としては、以下のものがあります。

- DKDatastoreDefDL
- DKDatastoreDefOD

dkEntityDef

エンティティ情報にアクセスするメソッドを使用します。エンティティと属性の作成および削除も行います。このクラスのメソッドは、複数レベル・エンティティへのアクセスをサポートしています。コンテンツ・サーバーがサブエンティティをサポートしていない場合、DKUsageError オブ

ジェクトが生成されます。コンテンツ・サーバーが複数レベル・エンティティーをサポートしている場合、これらのコンテンツ・サーバーのサブクラスに関する例外を上書きするメソッドをインプリメントしなければなりません。dkEntityDef インターフェースの具象クラスの例は、以下のとおりです。

- DKIndexClassDefDL
- DKAppGrpDefOD

図 14 では、エンティティー定義のクラス階層について説明しています。

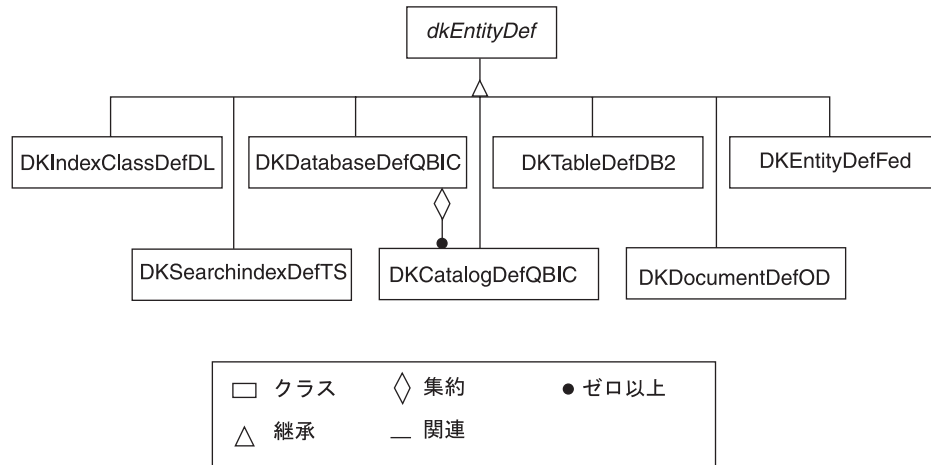


図 14. クラス階層

dkAttrDef

属性情報にアクセスするメソッドや、属性の作成や削除を実行するメソッドを定義します。 dkAttrDef の具象クラスの例は、以下のとおりです。

- DKAttributeDefDL
- DKFieldDefOD

dkServerDef

サーバー情報にアクセスするメソッドを定義します。 dkServerDef の具象クラスの例は、以下のとおりです。

- DKServerDefDL
- DKServerDefOD

dkResultSetCursor

DDO オブジェクトのコレクションを管理するコンテンツ・サーバー・カーソルを作成します。 addObject、deleteObject、および updateObject メソッドを使用するには、コンテンツ・サーバー・オプション DK_CM_OPT_ACCESS_MODE を DK_CM_READWRITE に設定しなければなりません。

dkBlob

各コンテンツ・サーバー内のバイナリー・ラージ・オブジェクト (BLOB) データ型に共通のインターフェースを宣言します。 dkBlob から派生してい

る具象クラスは、この共通インターフェースを共有して、異種コンテンツ・サーバーからの BLOB を処理することができます。dkBlob の具象クラスの例は、以下のとおりです。

- DKBlobDL
- DKBlobOD

データ定義クラスおよびそれらのクラス階層を図 15 に示します。

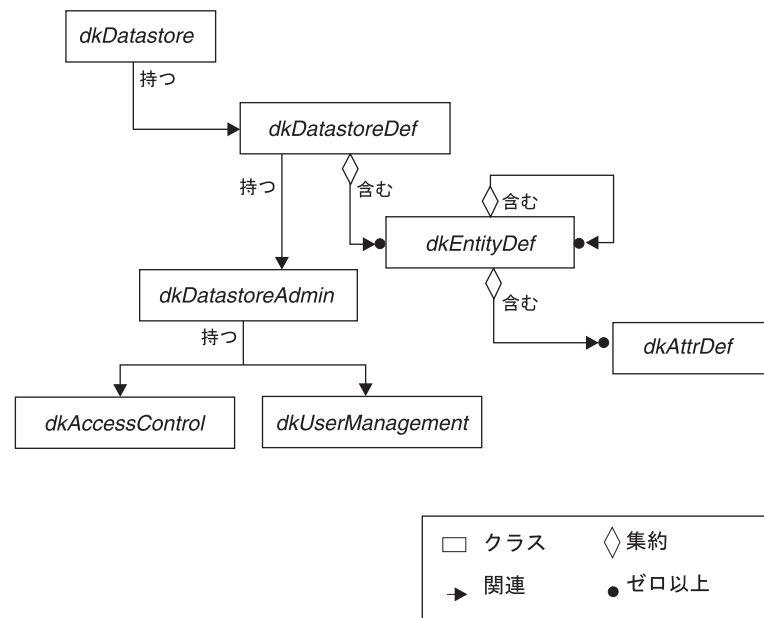


図 15. データ定義クラス階層

dkDatastore およびその他の共通クラスの詳細については、405 ページの『カスタム・コンテンツ・サーバー・コネクタの開発』を参照してください。

旧バージョンの Content Manager の使用

ここでは、Content Manager サーバー内のデータにアクセスする方法、および以下の作業を実行する方法について説明します。

- ラージ・オブジェクトの処理
- DDO の使用
- 検索エンジンでの XDO の使用
- 結合照会の使用
- テキスト検索エンジンの使用
- イメージ検索 (QBIC) の使用
- ワークフローとワークバスケットの使用

ラージ・オブジェクトの処理

旧バージョンの Content Manager コネクタでは、非同期検索を使用してラージ・オブジェクトを部分ごとに検索できます。サンプル・アプリケーションについては、CMBROOT¥dk¥samples ディレクトリーにある TxdoAsyncRetDL を参照してください。

Java ヒープ・サイズの設定 (Java のみ)

Java には、デフォルトの初期ヒープ・サイズと最大ヒープ・サイズに対する制限があります。デフォルトの初期ヒープ・サイズは 1 048 576 バイトであり、デフォルトの最大ヒープ・サイズは 16 777 216 バイトです。ご使用の Java アプリケーション・プログラムがヒープ・サイズより大きいオブジェクトを使おうとすると、プログラムは実行中に失敗します。アプリケーションの最大ヒープ・サイズを大きくするには、Java アプリケーション・プログラムの実行時に `-mx` オプションを使用します。例えば、以下のようになります。

Java

```
java -mx40000000 yourApplication
```

旧バージョンの Content Manager のコンテンツを表す DDO の使用

DKDatastoreDL に関連付けられる DDO には、Enterprise Information Portal 文書のモデル (文書、フォルダー、パーツ、項目 ID、ランクなど) を表す固有の情報があります。この情報にアクセスする方法について、以下に説明します。

DDO のプロパティー

項目のタイプ (文書かフォルダー) は、名前 `DK_CM_PROPERTY_ITEM_TYPE` の下にあるプロパティーです。DDO の項目タイプを取得する場合、以下を呼び出します。

Java

```
DKDDO addo = new DKDDO(dsDL, pid);
Object obj = addo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (obj != null) {
    short item_type = ((Short) obj).shortValue();
}
```

C++

```
DKAny any = cddo->getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (!any.isNull()) {
    unsigned short item_type = (unsigned short) any;
}
... // do something
```

プロパティーが呼び出された後、`item_type` は、文書の場合は `DK_CM_DOCUMENT`、フォルダーの場合は `DK_CM_FOLDER` に同じになります。if 文により、プロパティーが

存在することを確認できます。詳細については、49 ページの『DDO へのプロパティの追加』および 53 ページの『DKDDO と属性プロパティの取得』を参照してください。

永続的 ID (Persistent identifier (PID))

PID には、Enterprise Information Portal 特有の情報のうち重要な部分が入られます。オブジェクト・タイプは DDO の属する索引クラスを示します。PID には、コンテンツ・サーバーからの対応する項目の項目 ID が含まれています。50 ページの『永続 ID (PID) の作成』を参照してください。

文書の表現

文書を表す DDO では、プロパティ `DK_CM_PROPERTY_ITEM_TYPE` が `DK_CM_DOCUMENT` に設定されてます。この PID には、オブジェクト・タイプとして索引クラス名が入ります。PID ID は項目 ID と同じです。

文書内のパーツは DKPartsDL オブジェクトとして表されます。このオブジェクトはバイナリー・ラージ・オブジェクト (BLOB) のコレクションであり、そのおのおのは DKBlobDL オブジェクトとして表されます。

文書 DDO には DKPARTS という固有の属性があり、その値は DKParts オブジェクトです。

文書内の個々のパーツを取得するには、まず DKParts オブジェクトを検索してから、イテレーターを作成し、個々のパーツに処理を繰り返します。文書にパーツがない場合、DKParts はヌルになるか、DKParts の基数はゼロになります。

結合照会 (パラメトリック照会とテキスト照会を組み合わせたもの) に関連した文書は、DKRANK という一時属性を持つことができます。この属性の値は、テキスト検索エンジンによって計算された整数のランクが含まれるオブジェクトになります。

DKParts オブジェクトの作成および処理についての詳細は、270 ページの『文書またはフォルダーの作成、更新、および削除』、278 ページの『文書またはフォルダーの検索』、および 95 ページの『文書の作成と DKPARTS 属性の使用』を参照してください。

フォルダーの表現

フォルダーを表現する DDO のプロパティ `DK_CM_PROPERTY_ITEM_TYPE` は、`DK_CM_FOLDER` と同じです。文書の DDO と同じく、この PID にはオブジェクト・タイプとして索引クラスが含まれ、PID の ID 中には項目 ID が含まれます。

DKFolder オブジェクトは、フォルダーの目録を表します。DKFolder オブジェクトは DDO のコレクションです。各 DDO はフォルダー内の項目 (文書または他のフォルダー) を表しています。フォルダー DDO には DKFOLDER という属性があり、その値は DKFolder オブジェクトです。

フォルダーの個々の DDO メンバーを取得するには、まず DKFolder オブジェクトを検索してから、イテレーターを作成し、それぞれの項目メンバーにアクセスしなければなりません。フォルダーにメンバーがない場合、DKFolder はヌルになりますが、DKFOLDER 属性は、コンテンツ・サーバーによって作成されるフォルダー DDO 内に必ず存在します。

DKFolder オブジェクトの作成および処理の詳細については、『文書またはフォルダーの作成、更新、および削除』、278 ページの『文書またはフォルダーの検索』、および 98 ページの『フォルダーの作成と DKFOLDER 属性の使用』を参照してください。

文書またはフォルダーの作成、更新、および削除

文書およびフォルダーの作成、更新、および削除に関するプロセスについて、以下に説明します。

文書の作成

文書を作成してその永続データをコンテンツ・サーバー中に保管する場合、DDO を作成して、項目 ID 以外の属性（およびその他の情報）をすべて設定する必要があります。項目 ID は、コンテンツ・サーバーによって割り当てられ戻されます。以下の例は、前述のいくつかの例を組み合わせたものです。

Java

```
// ----- Step 1: create a datastore and connect to it
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Step 2: create a document (or folder) DDO
//          and set all its attributes and other required information
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA"); // Set the index-class name it belongs to
DKDDO ddo = new DKDDO(dsDL,pid); // Create a DDO with PID and
...                               // associate it to dsDL

// ----- Step 2.a: add attributes according to index class GRANDPA
Object obj, vstr;
Boolean yes = new Boolean(true);
Boolean no = new Boolean(false);

short data_id = cddo.addData("Title"); // add new attribute "Title"
vstr = new Short(DK_CM_DATAITEM_TYPE_STRING);
// ----- Add type properties VSTRING and nullable
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, no);

data_id = cddo.addData("Subject"); // add new attribute "Subject"
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes);

// ----- Add some more attributes as necessary
....

// ----- Step 2.b: add DKPARTS attribute
DKParts parts = new DKParts(); // create a new DKParts, collection of parts
DKBlobDL blob = new DKBlobDL(dsDL); // create a new XDO blob
DKPidXDODL pidXDO = new DKPidXDODL(); // create PID for this XDO object

pidXDO.setPartId(5); // set part number to 5
blob.setPidObject(pidXDO); // set the PID for the XDO blob
blob.setContentClass(DK_DL_CC_GIF); // set content class type GIF
blob.setRepType(DK_REP_NULL); // set rep type for the part
blob.setContentFromClientFile("choice.gif"); // set the blob's content
blob.setInstanceOpenHandler("xv"); // the viewer program on AIX

parts.addElement(blob); // add the blob to the parts collection

.... // create and add some more blobs to
.... // to the collection as necessary

// ----- Create DKPARTS attribute and set it to refer to the DKParts object
short data_id = ddo.addData(DKPARTS); // add attribute "DKParts"
obj = new Short(DK_CM_COLLECTION_XDO); // add type property
ddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, obj);
ddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes); // add nullable prop
ddo.setData(data_id, parts); // sets the attribute value

// ----- Step 2.c: sets the item type : document
obj = new Short(DK_CM_DOCUMENT);
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// ----- Step 3: make item persistent; add item to the datastore
ddo.add(); // document created in datastore
```

C++

```
// step 1: create a datastore and connect to it
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// step 2: create a document (or folder) DDO
//          and set all its attributes and other required information
//          See the section on "Using DDO"
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// create a DDO with PID and associated with dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// step 2.a: add attributes according to index-class GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add a new attribute named "Title"
unsigned short data_id = cddo->addData("Title");
// add type property VSTRING
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
// add nullable property: non-nullable
any = no;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add a new attribute named "Subject"
data_id = cddo->addData("Subject");

any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
any = yes;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add some more attributes as necessary
// ...

// step 2.b: add DKPARTS attribute
// create a new XDO blob
DKParts* parts = new DKParts;
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO; // create PID for this XDO object

pidXDO.setPartId(5); // set part number to 5
blob->setPid(&pidXDO); // set the PID for the XDO blob
blob->setContentClass(DK_CC_GIF); // set content class type GIF
blob->setRepType(DK_REP_NULL); // set rep type for the part
blob->setContentFromClientFile("choice.gif"); // set the blob's content

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any); // add the blob to the parts collection

... // create and add some more blobs
... // to the collection as necessary
// continued...
```

C++ (続き)

```
// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
any = DK_COLLECTION_XDO;
// add type property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
any = (DKBoolean) TRUE;
// add nullable property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);

// step 2.c: sets the item type : document
any = DK_CM_DOCUMENT;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// step 3: make it persistent
// a document is created in the datastore
ddo->add();
```

上記の例の最後のステップで、コンテンツ・サーバー内に文書が作成されます (情報を含む)。文書 DDO がコンテンツ・サーバーに追加されると、DKParts コレクション内のすべてのパーツを含む、そのすべての属性が追加されます。

フォルダー DDO を追加するのと同じプロセスを使用します。DKFOLDER コレクション・メンバーは、フォルダーのコンポーネントとしてコンテンツ・サーバーに追加されます。フォルダーには、そのメンバー (既存の文書とフォルダー) の目録が含まれます。このため、フォルダー DDO を追加する前に、すべてのフォルダー・メンバーをコンテンツ・サーバー内で作成してください。

Java

同じ文書を、別個のコンテンツ・サーバーに追加することができます。この文書を Content Manager サーバー LIBSRVRN (索引クラスは LIBSV2 で、構造は LIBSV と同じ) に追加する場合、以下の例を使用します。

```
// ----- Create datastore and connect to LIBSRVRN
DKDatastoreDL dsN = new DKDatastoreDL();
dsN.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- Update the PID
pid = ddo.getPidObject();
pid.setObjectType("LIBSV2");           // set the new index class
pid.setPrimaryId("");                  // make the item ID blank
pid.setDatastoreName("LIBSRVRN");      // set the new datastore name
ddo.setPidObject(pid);                 // update the PID
ddo.setDatastore(dsN);                 // re-associate the DDO with dsN
ddo.add();                             // add the DDO
```

C++

同じ文書を、別個のコンテンツ・サーバーに追加することができます。例えば、GRANDPA と同じ構造で索引クラスが GRANDPA2 である文書を、サーバー LIBSRVRN に追加するには、下記のようにします。

```
// create datastore and connect to LIBSRVRN
DKDatastoreDL dsN;
dsN.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// update the Pid
pid = ddo->getPid();
pid.setObjectType("GRANDPA2");           // set the new index-class
pid.setId("");                           // blank the item-id
pid.setDatastoreName("LIBSRVRN");        // set the new datastore name
ddo->setPid(pid);                         // update the PID
ddo->setDatastore(&dsN);                  // re-associate it with dsN
ddo->add();                               // add it
```

文書またはフォルダーの更新

文書またはフォルダーを更新するには、以下のようにします。

1. 項目 ID およびオブジェクト・タイプを設定します。
2. 適切な属性を更新するか、DKParts コレクションへ追加します。
3. update メソッドを呼び出して、変更を保管します。

Java

```
// ----- Update the value of attribute Title
String newTitle = "Accident Report";
short data_id = ddo.getDataByName("Title");
ddo.setData(data_id, newTitle);
ddo.update();
```

C++

```
// update the value of attribute Title
DKAny any = DKString("Guess who is behind all this");
unsigned short data_id = ddo->getDataByName("Title");
ddo->setData(data_id, any);
ddo->update();
```

update メソッドに対する呼び出しの後に、コンテンツ・サーバー中の Title の値が更新されます。この文書中のパーツは、内容に変更が加えられない限り更新されません。update メソッドを呼び出す際には、サーバーへの接続が有効でなければなりません。

フォルダー DDO を更新する場合も、同じステップを実行する必要があります。属性値を更新するか、または DKFolder にエレメントを追加または除去してから、update メソッドを呼び出します。

パーツの更新

文書内のパーツのコレクションは、DKParts オブジェクトを使用して表します。

DKParts は、DKSequentialCollection のサブクラスです。順次コレクション機能の継承に加え、DKParts には 2 つの追加メンバーがあります。1 つはコレクションにパーツを追加し、もう 1 つはコレクションからパーツを除去します。これらのメソッドは、コンテンツ・サーバーへの変更を即時に保管します。

この文書はコンテンツ・サーバー内にすでに存在していなければなりません。

メンバーの追加と除去: 以下の例では、文書にパーツを追加します。

Java

```
DKDDO addo = new DKDDO();    // create a document DDO
DKBlobDL newPart = new DKBlobDL(); // create the new part to be added
....                          // initialized the DDO and new part
DKParts parts = (DKParts) addo.getDataByName(DKPARTS); // get DKParts
parts.addMember(ddo, newPart); // assume none of these values are NULL
```

C++

```
// a document DDO
DKDDO* ddo;
// a new part to be added
DKBlobDL* newPart;
// ddo and newPart are
// initialized somewhere along the line
...
...
// get DKParts
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
// assume none of these values are NULL
parts->addMember(ddo, newPart);
```

コレクションおよびコンテンツ・サーバーから newPart を除去するには、以下を使用します。

Java

```
parts.removeMember(addo, newPart);
```

C++

```
parts->removeMember(ddo, newPart);
```

DKParts 内の removeMember メソッドは、実際に、コンテンツ・サーバーからパーツの永続コピーを削除します。

文書 DDO に対する更新、追加、および除去処理の相違点: DKParts の

addMember メソッドと removeMember メソッドは、コレクションやコンテンツ・サーバーに対してパーツの追加や除去を実行する際に便利です。これらを使用した方が、文書 DDO で update メソッドを使用するより処理が高速です。DDO 上の update メソッドは、DKParts およびその変更したメンバーのすべてを含む、DDO のすべての属性を更新します。ステップは以下のとおりです。

Java

```
....
// ----- Get DKParts, assume it exists and not null
DKParts parts = (DKParts) addo.getDataByName(DKPARTS);
parts.addElement(newPart);           // add a new part to parts
addo.update();                       // updates the whole ddo
....
```

C++

```
...
DKAny any = ddo->getDataByName(DKPARTS);
// get DKParts, assume it exists
DKParts* parts = (DKParts*) any.value();
// assume it is not NULL
any = (dkDataObjectBase*) newpart;
parts->addElement(any);
// updates the whole ddo
ddo->update();
...
```

フォルダーの更新

フォルダー内の文書およびフォルダーのコレクションは、DKFolder オブジェクトを使用して表します。コンテンツ・サーバーでは、フォルダーには、実際のすべてのオブジェクトを保持する代わりに、オブジェクトを参照する目録を保持します。

DKFolder は、DKSequentialCollection のサブクラスです。これには、順次コレクション・メソッドの継承以外に追加メンバーが 2 つあります。1 つはコレクションへのメンバー (文書またはフォルダー) の追加、もう 1 つはコレクションからのメンバーの除去のためのものであり、それらの変更はただちに保管されます。

除去や追加の対象となる文書やフォルダーは、コンテンツ・サーバーにすでに存在するものでなければなりません。

メンバーの追加と除去: 以下の例は、フォルダー DDO に別の文書またはフォルダー DDO を追加しています。

Java

```
DKDDO folderDDO = new DKDDO(); // Created the folder DDO
DKDDO newMember = new DKDDO(); // Create the new DDO to be added
.... // The folder DDO and newMember are
.... // initialized
// ----- Get the DKFolder, assuming it exists, and the value not null
DKFolder folder = (DKFolder) folderDDO.getDataByName(DKFOLDER);
folder.addMember(folderDDO, newMember);
```

C++

```
// a folder DDO
DKDDO* folderDDO;
// a new DDO to be added as a member of this folder
DKDDO* newMember;
... // folderDDO and newMember are
... // initialized somewhere along the line
DKAny any = folderDDO->getDataByName(DKFOLDER);
// get DKFolder, assume it exists
DKFolder* folder = (DKFolder*) any.value();
// assume non NULL
folder->addMember(folderDDO, newMember);
```

別の文書またはフォルダーを追加するには、`newMember` と `folderDDO` の両方がコンテンツ・サーバー中になければなりません。

同じように、コレクションとコンテンツ・サーバーから `newMember` を除去するには、以下の例を使用します。

Java

```
folder.removeMember(folderDDO, newMember);
```

C++

```
folder->removeMember(folderDDO, newMember);
```

重要: フォルダーからメンバーを除去しても、そのメンバーはフォルダーの目録から除去されるだけです。 `removeElementAt` 関数を使用しても、メンバーはメモリーやコンテンツ・サーバーから削除されません。

フォルダー DDO に対する更新、追加、および除去処理の間の相違点: `DKFolder` の `addMember` メソッドと `removeMember` メソッドは、コレクションやコンテンツ・サーバーに対して文書またはフォルダーの追加や除去を行う際に便利です。これらを使用した方が、フォルダー DDO で `update` メソッドを使用するよりも処理が高速です。

DDO での update メソッドは DKFolder およびそのメンバーのすべてを含む DDO のすべての属性を更新するのに対して、addMember メソッドはフォルダーの目録に特定のメンバーを 1 つだけ追加し、removeMember メソッドはフォルダーの目録から特定のメンバーを 1 つだけ除去します。

文書またはフォルダーの削除

コンテンツ・サーバーから文書またはフォルダーを削除するには、DDO の del メソッドを使用します。

Java

```
ddo.del();
```

C++

```
ddo->del();
```

DDO は、項目 ID とオブジェクト・タイプが設定されていなければならない、コンテンツ・サーバーと正しく接続されていなければならない。

上記の文を使用して、フォルダーを削除することもできます。永続データだけが削除され、DDO のメモリー内のコピーは変更されません。したがって、アプリケーション内で後からこの DDO を同じデータ・ストアや別のコンテンツ・サーバーに追加することができます。詳細については、270 ページの『文書の作成』を参照してください。

文書またはフォルダーの検索

(旧バージョンの Content Manager コンテンツ・サーバーを表す) DKDatastoreDL から文書を検索するには、その文書の索引クラス名および項目 ID を知っている必要があります。また、DDO をコンテンツ・サーバーに関連付け、接続を確立しなければなりません。

Java

```
DKDDO ddo = new DKDDO(dsDL,pid);
// ----- Create the datastore and establish a connection
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKPid pid = new DKPid();
pid.setObjectType("Claim"); // set the index-class name it belongs to
pid.setPrimaryId("LN#U5K6ARLGM3DB4"); // set the item-id
// ----- create a DDO with the PID and associated with the datastore

ddo.retrieve(); // retrieve the document
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// set the item-id
pid.setId("LN#U5K6ARLGM3DB4");
// create a DDO with pid and associated to dsDL
DKDDO* ddo = new DKDDO(&dsDL, pid);
// retrieve it
ddo->retrieve();
```

retrieve 呼び出しの後、DDO のすべての属性値は、コンテンツ・サーバーに保管されている永続データの値に設定されます。文書にパーツがある場合、DKPARTS 属性は、DKParts オブジェクトに設定されます。しかし、このコレクション中の個々のパーツの内容は検索されません。 パーツは大きくなる場合があるため、すべてのパーツを一度にメモリーに取り込まないでください。特定のパーツを明示的に取得する方がよいでしょう。

DDO が照会オプション CONTENT=NO を指定したパラメトリック照会の結果である場合、その DDO は空です (属性値がありません)。しかし、取得に必要な情報はすでにすべて設定されています。

パーツの取得

DDO を検索した後、以下のようにして、DKPARTS 属性内に保管されているパーツを取得することができます。

Java

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

C++

```
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
```

この例では、DKPARTS 属性が存在することを前提としています。存在しない場合は、例外が生成されます。まずデータ ID を入手してそれが 0 の場合をテストすることによって属性値を抽出する例については、281 ページの『フォルダーの取得』を参照してください。

個々のパーツを取得するには、イテレーターを作成してコレクション全体に対して処理を実行し、各パーツを取得しなければなりません。 95 ページの『文書の作成と DKPARTS 属性の使用』を参照してください。

Java

```
// ----- Create an iterator and process the part collection members
if (parts != null) {
    DKBlobDL blob;
    dkIterator iter = parts.createIterator();
    while (iter.more()) {
        blob = (DKBlobDL) iter.next();
        if (blob != null) {
            blob.retrieve(); // retrieve the blob's content
            blob.open();
            ....           // other processing, as needed
        }
    }
}
```

C++

```
// create iterator and process the part collection member one by one
if (parts != NULL) {
    DKAny* element;
    DKBlobDL* blob;
    dkIterator* iter = parts->createIterator();
    while (iter->more()) {
        element = iter->next();
        blob = (DKBlobDL*) element->value();
        if (blob != NULL) {
            // retrieve the blob's content
            blob->retrieve();
            // other processing, as needed
            blob->open();
        }
    }
    delete iter;
}
```

パラメトリック照会の DDO 結果と同様、DKParts コレクション内の各パーツの XDO は空です (そのコンテンツ・セットはありません)。しかし、取得に必要な情報はすべてあります。そのコンテンツと関連情報をメモリーに入れるには、以下の retrieve メソッドを呼び出します。

Java

```
blob.retrieve();
```

C++

```
blob->retrieve();
```

フォルダーの取得

フォルダー DDO を取得する方法は、文書 DDO を取得する方法と同じです。取得後、フォルダー DDO には、属性 DKFOLDER を含むすべての属性が設定されています。この属性値は、フォルダー内の DDO メンバーのコレクションである DKFolder オブジェクトに設定します。DKParts オブジェクト内のパーツのように、これらのメンバー DDO には、それらを取得するのに十分な情報のみが含まれます。フォルダー DDO を取得するには、下記のようにします。

Java

```
data_id = ddo.dataId(DKFOLDER);    // get DKFOLDER data-id
if (data_id == 0)                  // folder not found
    throw new DKException(" folder data-item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // get the folder collection

// ----- Create iterator and process the DDO collection members one by one
if (fCol != null) {
    DKDDO item;
    dkIterator iter = fCol.createIterator();
    while (iter.more()) {
        item = (DKDDO) iter.next();
        if (item != null) {
            item.retrieve();    // retrieve the member DDO
            ....                // other processing
        }
    }
}
```

C++

```
// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
// folder not found
if (data_id == 0) {
    DKException exc(" folder data-item not found");
    DKTHROW exc;
}
// get the folder collection
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// create iterator and process the DDO collection member one by one
if (fCol != NULL) {
    DKAny* element;
    DKDDO* item;
    dkIterator* iter = fCol->createIterator();
    while (iter->more()) {
        element = iter->next();
        item = (DKDDO*) element->value();
        if (item != NULL) {
            // retrieve the member DDO
            item->retrieve();
            // other processing
            ...
        }
    }
    delete iter;
}
```

98 ページの『フォルダーの作成と DKFOLDER 属性の使用』も参照してください。

テキスト検索 (テキスト検索エンジン) について

テキスト検索エンジンは、さまざまな照会タイプをサポートしています。

- 『ブール照会』
- 283 ページの『フリー・テキスト照会』
- 283 ページの『ハイブリッド照会』
- 283 ページの『接近照会』
- 284 ページの『グローバル・テキスト検索 (GTR) 照会』

テキスト検索項目 ID、パーツ番号、およびランキング情報 (照会から戻される) を使用して、旧バージョンの Content Manager サーバーから文書を取得する XDO を作成することができます。

テキスト検索エンジンを表すのには、DKDatastoreTS オブジェクトを使用します。テキスト検索エンジンが実際にデータを保管するわけではありません。単に旧バージョンの Content Manager に保管されているデータに索引付けし、それらに対するテキスト検索をサポートするだけです。テキスト検索の結果は、Content Manager 中の文書の位置を示す項目 ID になります。これらの ID を使用して、文書を検索してください。

DKDatastoreTS オブジェクトは、add、update、retrieve、および delete 関数をサポートしません。このコンテンツ・サーバーを照会することはできます。テキスト検索エンジンによって索引付けされた Content Manager にデータを追加することについては、294 ページの『テキスト検索エンジンによって索引付けされるデータのロード』を参照してください。

ブール照会

ブール照会は語と句で構成され、ブール演算子によって区切られます。句は単一引用符 (') で囲んでください。句はリテラル・ストリングとして扱われます。

以下の例で作成する照会ストリングでは、TMINDEX テキスト検索索引中の、語 `www` または句 `web site` を含むテキスト文書がすべて検索されます。

Java

```
String cmd = "SEARCH=(COND=(www OR 'web site'));" +  
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=(www OR 'web site'))";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

フリー・テキスト照会

フリー・テキスト照会は、中括弧 ({ }) で囲まれた語、句、または文で構成されます。語が互いに隣接している必要はありません。以下の例で作成する照会ストリングでは、TMINDEX テキスト検索索引の中から、フリー・テキスト web site を含むテキスト文書がすべて検索されます。

Java

```
String cmd = "SEARCH=(COND=({web site}));" +  
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=({Web site}));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

ハイブリッド照会

ハイブリッド照会は、ブール照会の後にフリー・テキスト照会が続く構成になっています。以下の例で作成する照会ストリングでは、TMINDEX テキスト検索索引の中から、フリー・テキスト web site、そして語 IBM および UNIX を含むテキスト文書がすべて検索されます。

Java

```
String cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));" +  
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=(IBM AND UNIX {Web site}));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

接近照会

接近照会は、同一の文書、段落、または文の中から一連の検索引き数を検索する機能です。以下の例で作成する照会ストリングでは、TMINDEX テキスト検索索引を使用して、同一の段落内に句 rational numbers と語 math が含まれているすべてのテキスト文書が検索されます。

Java

```
String cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));" +  
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)"
```

このタイプの照会には、2 つ以上の検索引き数が必要です。

グローバル・テキスト検索 (GTR) 照会

GTR 照会は、日本語や中国語などの 2 バイト文字セット (DBCS) 言語用に最適化されています。GTR は、1 バイト文字セット (SBCS) 言語もサポートしています。2 バイト文字はすべて単一引用符 (') で囲んでください。検索する句では、指定した文字コードおよび言語を必ず使用するようになしてください。

以下の例は、句 IBM marketing を含むすべてのテキスト文書の GTR 索引の検索を示しています。MATCH キーワードは、句が類似している度合いを示すよう設定されています。

Java

```
String cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ " +  
            "'IBM marketing'))";  
"OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=($CCSID=850, LANG=6011,MATCH=1$ ";  
cmd += "'IBM marketing'))";  
cmd += "OPTION=(SEARCH_INDEX=TMGTRX)";
```

テキスト検索コンテンツ・サーバー・オプション DK_OPT_TS_CCSID (コード化文字セット ID) および DK_OPT_TS_LANG (言語 ID) が正しく設定されていることを確認してください。DK_OPT_TS_CCSID のデフォルトは、DK_CCSID_00850 です。DK_OPT_TS_LANG のデフォルトは、DK_LANG_ENU です。これらの値は、テキスト照会のグローバル・デフォルトとして使用されます。詳細については、「オンライン API 解説書」を参照してください。

以下の例のように、特定の CCSID 情報と LANG 情報を入力することもできます。CCSID と LANG を両方とも指定する必要があります。一方の値を指定しなければ、他方の値も指定することはできません。

DDO を使用したテキスト検索エンジン情報の表現

テキスト検索からの結果を表すには、(DKDatastoreTS オブジェクトと関連付けられた) DDO を使用します。

DKDatastoreTS には、DKDatastoreDL オブジェクトのようなプロパティ項目タイプはありません。ID の形式も異なります。テキスト照会の結果の DDO は、項目内のテキスト・パーツに対応します。以下の標準属性が含まれています。

DKDLITEMID

このテキストが含まれている項目 ID。この項目 ID は、コンテンツ・サーバーから項目全体を取り出すのに使います。

DKPARTNO

このテキスト・パーツのパーツ番号 (整数)。このパーツ番号と項目 ID は、コンテンツ・サーバーからテキスト・パーツを取り出すのに使います。

DKREPTYPE

このテキスト・パーツの表示タイプ。この属性は、項目 ID およびパーツ番号と共に、コンテンツ・サーバーからテキスト・パーツを取り出すのに使います。

DKRANK

このパーツとテキスト照会の結果との関係を指定するランク (整数)。ランクが高いほど一致の程度が大きくなります。詳細については、「オンライン API 解説書」を参照してください。

DKDSIZE

ブール照会の結果の中の語句の出現数を表す整数。詳細については、「オンライン API 解説書」を参照してください。

DKRCNT

ブール検索で一致した数を表す整数。詳細については、「オンライン API 解説書」を参照してください。

テキスト検索 DDO の PID には以下の情報が含まれています。

コンテンツ・サーバー・タイプ

TS。

コンテンツ・サーバー名

コンテンツ・サーバーに接続するために使用される名前。

オブジェクト・タイプ

テキスト検索索引。

ID テキスト検索エンジンの文書 ID。

接続の確立

DKDatastoreTS オブジェクトには、接続用と切断用に 2 つの関数があります。通常は、DKDatastoreTS オブジェクトを作成して接続し、照会を実行してから、終了後に切断します。以下の例は、テキスト検索サーバー TM を使用した 1 つ目の接続関数を示しています。

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM", "", "", "");
.... // run a query
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TConnectTS.java) が CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","", "", "");
... // do some work
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TConnectTS.cpp) が Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから取られています。

以下の例では、ホスト名 apollo、ポート番号 7502、および TCP/IP 通信タイプ DK_CTYP_TCPIP を持つテキスト検索サーバーを使用している 2 番目の接続関数を示しています。

```
dsTS.connect("apollo", "7502", DK_CTYP_TCPIP);
```

以下の例は、ホスト名 apollo、ポート番号 7502、および通信タイプ T (TCP/IP) のテキスト検索サーバーを使用した、最初の接続関数を示しています。

```
dsTS.connect("apollo", "", "", "PORT=7502; COMMTYPE=T");
```

以下の例では、テキスト検索サーバー名 TM、ライブラリー・サーバー LIBSRVR2、ユーザー ID FRNADMIN、およびパスワード PASSWORD を使用した、最初の接続メソッドを示しています。

下記の例では、テキスト検索サーバー名 TM、ライブラリー・サーバー LIBSRVRN、ユーザー ID FRNADMIN、およびパスワード PASSWORD を使用した、最初の接続関数を示しています。

```
dsTS.connect("TM", "", "", "LIBACCESS=(LIBSRVRN, FRNADMIN, PASSWORD)");
```

最後のパラメーター LIBACCESS (接続ストリング) を使用すると、一連のパラメーターを渡すことができます。

ヒント: テキスト検索エンジン接続がタイムアウトにならないようにするには、テキスト検索エンジンに接続し、照会を実行した後すぐに切断してください。接続をオープンしたままにはしないでください。

テキスト検索オプションの取得と設定

テキスト検索にはいくつかのオプションがあり、それらのオプションの設定や取得には、その関数を使います。オプションのリストとその説明については、「オンライン API 解説書」を参照してください。以下の例は、テキスト検索文字コード・セットのオプションの設定と取得の方法を示しています。

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option = new Integer(DK_TS_CCSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSID);
```

C++

```
DKAny input_option = DK_CCSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSID, input_option);
dsTS.getOption(DK_OPT_TS_CCSID, output_option);
```

output_option はオブジェクトですが、通常 Integer にキャストされます。

ヒント: 検索オプションの CCSID と LANG は協働します。一方を指定したら、他方も指定しなければなりません。デフォルト CCSID および LANG は、DKDatastoreTS オプション (DK_OPT_TS_CCSID および DK_OPT_TS_LANG) によって指定されます。コンテンツ・サーバー・オプションとその説明のリストについては、「オンライン API 解説書」を参照してください。

照会項目についての検索オプションは複数指定することができます。検索オプションはコンマで区切ります。複数検索項目の例は、284 ページの『グローバル・テキスト検索 (GTR) 照会』に示されています。

SC (単一必須文字) および MC (オプション文字のシーケンス) 検索オプションのどちらの場合も、まず SC 検索オプションを最初に指定しなければなりません。例えば、\$SC=?,MC=*\$ U?I* のように指定します。

サーバーのリスト作成

DKDatastoreTS オブジェクトは、接続先となるテキスト検索サーバーをリストする関数を備えています。以下の例では、サーバーのリストを取得する方法を示しています。

Java

```
DKServerDefTS pSV = null;
DKIndexTS pIndx = null;
String strServerName = null;
char chServerLocation = ' ';
String strLoc = null;
String strIndexName = null;
String strLibId = null;
int i = 0;
DKDatastoreTS dsTS = new DKDatastoreTS();
System.out.println("list servers");
pCol = (DKSequentialCollection)dsTS.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
    i++;
    pSV = (DKServerDefTS)pIter.next();
    strServerName = pSV.getName();
    chServerLocation = pSV.getServerLocation();
    if (chServerLocation == DK_TS_SRV_LOCAL)
        strLoc = "LOCAL SERVER";
    else if (chServerLocation == DK_TS_SRV_REMOTE)
        strLoc = "REMOTE SERVER";
    System.out.println("Server Name [" + i + "] - " + strServerName +
        " Server Location - " + strLoc);
}
```

完全なサンプル・アプリケーション (TListCatalogTS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strServerName;
char chServerLocation = ' ';
DKString strLoc;
DKServerDefTS *pSV = 0;
long i = 0;
DKAny a;
cout << "list servers" << endl;
a = dsTS.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    pSV = (DKServerDefTS*)((void*)(*pIter->next()));
    strServerName = pSV->getName();
    chServerLocation = pSV->getServerLocation();
    if (chServerLocation == DK_SRV_LOCAL)
    {
        strLoc = "LOCAL SERVER";
    }
    else if (chServerLocation == DK_SRV_REMOTE)
    {
        strLoc = "REMOTE SERVER";
    }
    cout << "Server Name [" << i << "] - " << strServerName
        << " Server Location - " << strLoc << endl;
    delete pSV;
}
delete pIter;
delete pCol;
```

完全なサンプル・アプリケーション (TListCatalogTS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

サーバーのリストは、DKServerInfoTS オブジェクトの DKSequentialCollection に戻されます。DKServerInfoTS オブジェクトを取得すると、サーバーの名前および場所を検索できます。その後、サーバー名を使用して、そのサーバーへの接続を確立することができます。

スキーマのリスト作成

DKDatastoreTS オブジェクトには、スキーマをリストする関数があります。テキスト検索の場合、テキスト検索索引がこれに該当します。下記の例は、索引リストを取得する方法を示しています。

索引のリストは、DKIndexTS オブジェクトの DKSequentialCollection オブジェクトに戻されます。DKIndexTS オブジェクトを取得したら、その名前やライブラリーIDなどの索引に関する情報を取得し、それを使用して照会を作成することができます。

Java

```
tsCol = (DKSequentialCollection) dsTS.listEntities();
tsIter = pCol.createIterator();
int i = 0;
while (tsIter.more()) {
    i++;
    TsIndx = (DKSearchIndexDefTS)tsIter.next();
    strIndexName = TsIndx.getName();
    strLibId = TsIndx.getLibraryId();
    ...          ¥¥ Process the list as appropriate
}
```

完全なサンプル・アプリケーション (TListCatalogTS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strIndexName;
DKString strLibId;
DKServerDefTS *pSV = 0;
DKSearchIndexDefTS *pIndx = 0;
long i = 0;
DKAny a;
cout << "connecting to datastore" << endl;
dsTS.connect("TM","","");
cout << "list search indexes" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsTS.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pIndx = (DKSearchIndexDefTS*)((void*)(*pIter->next()));
    strIndexName = pIndx->getName();
    strLibId = pIndx->getLibraryId();
    cout << "index name [" << i << "] - " << strIndexName
         << " Library - " << strLibId << endl;
    delete pIndx;
}
delete pIter;
delete pCol;
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TListCatalogTS.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。また、コレクションの削除の照会については、109 ページの
『コレクションのメモリー管理 (C++ のみ)』を参照してください。

検索エンジンによる XDO の索引付け

テキスト検索エンジンとイメージ検索を使用してデータ項目を検索する前に、まずデータを索引付けする必要があります。索引には、SearchEngine、SearchIndex、および SearchInfo の 3 つの値が必要です。

SearchIndex の値は、検索サービス名と検索索引名の 2 つの名前の組み合わせです。例えば、システム管理クライアントに TM という名前のテキスト検索サーバーが定義されており、それに関連する TMINDEX という名前の検索索引が定義されている場合、SearchIndex の値は TM-TMINDEX になります。

テキスト検索エンジンによって索引付けされるオブジェクトの場合、SearchEngine の値は SM でなければならず、イメージ検索による照会によって索引付けされるオブジェクトの場合、SearchEngine の値は QBIC でなければなりません (イメージ検索については、306 ページの『イメージ検索の用語と概念について』を参照してください)。

QBIC の場合の SearchIndex は、QBIC データベース名、QBIC カタログ名、および QBIC サーバー名の 3 つの値を組み合わせたものです。例えば、QBIC データベース名が SAMPLEDB、QBIC カタログ名が SAMPLECAT、および QBIC サーバー名が QBICSRV の場合、SearchIndex の正しい値は SAMPLEDB-SAMPLECAT-QBICSRV です。

データをロードする方法、またはフォルダーを作成してデータをロードする方法の例については、CMBROOT¥Samples ディレクトリーにある LoadSampleTSQBICDL と LoadFolderTSQBICDL を参照してください。

重要: 検索エンジンによって索引付けされるパーツ・オブジェクトを追加する際には、表示タイプを設定しないでください。現行では、テキスト検索エンジンはデフォルトの表示タイプ FRN\$NULL を指定した場合に限り稼働します。

テキスト検索エンジンによって索引付けされる XDO の追加: 以下の例は、索引付けされる XDO を追加する方法を示しています。

Java

```
// ----- Declare variables for part ID, item ID, and file
int    partId = 20;
String itemId = "CPPIORH4JBIXWIY0";
String fileName = "g:¥¥test¥¥testsrch.txt";
try {
    DKDatastoreDL dsDL = new DKDatastoreDL(); // create datastore
    ...                                     // connect to datastore
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
    DKBlobDL axdo = new DKBlobDL(dsDL);      // create XDO
    DKPidXDODL apid = new DKPidXDODL();      // create PID
    apid.setPartId(partId);                  // set partId
    apid.setPrimaryId(itemId);               // set itemId
    axdo.setPidObject(apid);                 // setPid to XDO
    axdo.setContentClass(DK_DL_CC_ASCII);    // set ContentClass to text

    // --- set the searchEngine
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo->setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    ...
    // ----- Set file content to buffer area
    axdo.setContentFromClientFile(fileName);
    axdo.add();                             //add from buffer
    ...
    // ----- Display the partId after add
    System.out.println("after add partId = " + ((DKPidXDODL)
        (axdo.getPidObject())).getPartId());

    dsDL.disconnect();                      //disconnect from datastore
    dsDL.destroy();
}
// ----- Catch any exception
catch (...)
```


C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "N2JJBERBQFK@WTVL";
    repType = "FRN$NULL";
    partId = 10;
    if (argc == 1)
    {
        cout<<"invoke: indexPartxsDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    cout << "connecting Datastore" << endl;
    try
    {
        //replace following with your library server, user ID, password
        dsDL.connect("LIBSRVN","FRNADMIN","PASSWORD");

        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        apid ->setRepType(repType);
        axdo ->setPidObject(apid);
        cout<<"itemId= "<<axdo->getItemId()<<endl;
        cout<<"partId= "<<axdo->getPartId()<<endl;
        cout<<"repType= "<<axdo->getRepType()<<endl;
    }

    // continued...
```

C++ (続き)

```
//--- set searchEngine ----  
cout<<"set search engine and setToBeIndexed()"<<endl;  
DKSearchEngineInfoDL aSrchEx;  
aSrchEx.setSearchEngine("SM");  
aSrchEx.setSearchIndex("TM-TMINDEX");  
aSrchEx.setSearchInfo("ENU");  
axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);  
axdo->setToBeIndexed();  
cout<<"setToBeIndexed() done..."<<endl;  
  
delete apid;  
delete axdo;  
dsDL.disconnect();  
cout<<"datastore disconnected"<<endl;  
}  
catch(DKException &exc)  
{  
    cout << "Error id" << exc.errorId() << endl;  
    cout << "Exception id " << exc.exceptionId() << endl;  
    for(unsigned long i=0;i< exc.textCount();i++)  
    {  
        cout << "Error text:" << exc.text(i) << endl;  
    }  
    for (unsigned long g=0;g< exc.locationCount();g++)  
    {  
        const DKExceptionLocation* p = exc.locationAtIndex(g);  
        cout << "Filename: " << p->fileName() << endl;  
        cout << "Function: " << p->functionName() << endl;  
        cout << "LineNumber: " << p->lineNumber() << endl;  
    }  
    cout << "Exception Class Name: " << exc.name() << endl;  
}  
    cout << "done ..." << endl;  
}
```

重要: 検索エンジンによって索引付けされるパーツ・オブジェクトを追加する際には、RepType (表現タイプ) を設定しないでください。テキスト検索エンジンはデフォルトの RepType FRN\$NULL を指定した場合に限り動作します。

テキスト検索エンジンによって索引付けされるデータのロード: テキスト検索エンジンによって索引付けされるデータを Content Manager にロードするには、索引およびテキスト検索索引を両方とも作成しなければなりません。

テキスト検索索引を作成できるようにするには、テキスト検索サーバーが稼働していなければなりません。環境が正しく設定されていることを確認してください。このためには、CMBROOT¥Samples ディレクトリーにあるサンプル TListCatalogDL と TListCatalogTS をカスタマイズし、実行します。

テキスト検索エンジンによって索引付けされている Content Manager 内にパーツを作成するには、57 ページの『拡張データ・オブジェクト (XDO) の使用』を参照してください。

データが Content Manager にロードされた後、DKDatastoreDL クラスの wakeUpService 関数を使うと、その文書が文書キューに入れます。この関数は、検索エンジン名をパラメーターとします。

その後、Content Manager のテキスト検索管理ウィンドウから次の手順で行います。

1. テキスト検索サーバーをダブルクリックします。
2. テキスト検索索引をダブルクリックします。
3. 「索引 (INDEX)」をクリックします。

文書照会上の文書が索引付けされます。索引付けが完了したら、テキスト検索照会を実行できます。

テキスト検索の管理の詳細については、「システム管理ガイド」を参照してください。

テキスト構造化文書サポートの使用

テキスト構造化文書は、タグ付きテキストで構成されます (例: HTML ファイル)。文書モデルが文書の構造を定義し、テキスト検索エンジンはタグの間にある語や句を検索できます。

構造化文書のテキスト照会を実行するには、以下のようにします。

1. 文書モデルを作成します。文書モデルはセクションで構成され、各セクションにはセクション名と使用する文書タグが含まれています。例えば、以下のようになります。

```
<HTML>
<HEAD>
<TITLE>Acme Corp<br></TITLE>
</HEAD>
<BODY>
<H1>Acme Corp<BR></H1>
<P><B>Acme Corp<BR></B><BR>
<P>John Smith <BR>
<P><ADDRESS>Acme Corporation<BR></ADDRESS>
<HR>
<H2>Acme Corp Business Objectives</H2>
<HR>
<P>
<H2><A NAME="Header_Test" HREF="#ToC_Test">Marketing</A></H2>
<P>We need to increase our time to market by 25%.
<P>We need to meet our customers needs.
</BODY>
</HTML>
```

2. Content Manager 文書モデルを使用するテキスト検索索引を作成します。
3. テキスト検索索引の索引付けの規則を設定し、デフォルトの文書形式 (例えば、HTML ファイルの場合は DK_TS_DOCFMT_HTML) を指定します。
4. パーツ・オブジェクトを Content Manager サーバーに追加します。
5. テキスト検索索引の索引付けプロセスを開始します。

次の例では、システム内に定義されている文書モデルをリストする方法を示しています。

Java

```
// ----- Initialize the variables
DKSequentialCollection pCol = null;
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
dkIterator pIter = null;
DKDocModelTS pDocModel = null;
int ccsid = 0;
String strDocModelName = null;
int i = 0;

// ----- Create the datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect(srchSrv,"",' ');

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Get list of document models
pCol = (DKSequentialCollection) dsAdmin.listDocModels("");
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    pDocModel = (DKDocModelTS)pIter.next();
    strDocModelName = pDocModel.getName();
    ccsid = pDocModel.getCCSID();
}
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TListDocModelsTS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKDocModelTS* docModel = 0;
DKSequentialCollection *pCol = 0;
long ccsid = 0;
DKString strDocModelName;
dkIterator *pIter = 0;
long i = 0;
DKAny a;

dsTS.connect(srchSrv,"","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// list document models
pCol = (DKSequentialCollection*)dsAdmin->listDocModels("");
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    docModel = (DKDocModelTS*)((void*)(*pIter->next()));
    strDocModelName = docModel->getName();
    ccsid = docModel->getCCSID();
    delete docModel;
}
delete pIter;
delete pCol;

dsTS.disconnect();
```

完全なサンプル・アプリケーション (TListDocModelsTS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

以下の例は、文書モデルを作成する方法を示しています。

Java

```
// ----- Create datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Describe the document model for text document structure
//         for files like tstruct.htm above
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName(docModelName);
docSection.setName("SAMPTITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Create the document model
dsAdmin.createDocModel("", docModel);

dsTS.disconnect();
dsTS.destroy();
```

その他の例については、CMBROOT¥Samples¥java¥d1 ディレクトリーにある、TCreateDocModelTS.java および TCreateStructDocIndexTS.java を参照してください。

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// Describe the document model for text document structure
// for files like tstruct.htm above

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMP CORPMOD");
docSection->setName("SAMP CORP TITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMP CORP BODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// create doc model
dsAdmin->createDocModel("", docModel);

// delete document model & sections
delete docModel;

dsTS.disconnect();
```

完全なサンプル・アプリケーション (TCreateDocModelTS.cpp) および
(TCreateStructDocIndexTS.cpp) が Cmbroot/Samples/cpp/d1 ディレクトリー
に含まれており、この例はそこから取られています。

以下の例は、文書モデルを使用するテキスト検索索引の索引付け規則を作成し、設定する方法を示しています。

Java

```
// ----- Create the datastore and index rules object
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
DKIndexingRulesTS indexRules = new DKIndexingRulesTS();

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Create the document model instance for indexing rules
DKDocModelTS docModel2 = new DKDocModelTS();
docModel2.setCCSID(DK_TS_CCSID_00850);
docModel2.setName("SAMPCORPMOD");

// ----- Describe the document model for text document structure
//         for files like tstruct.htm above
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName("SAMPCORPMOD");
docSection.setName("SAMPTITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

// ----- Connect to the datastore
dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

DKSearchIndexDefTS pEnt = new DKSearchIndexDefTS((dkDatastore)dsTS);
pEnt.setName("TSTRUCT");
pEnt.setIndexType(DK_TS_INDEX_TYPE_PRECISE);
pEnt.setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);
pEnt.setLibraryId("LIBSUM");
pEnt.setLibraryClientServices("IMLLSCDL");
pEnt.setLibraryServerServices("IMLLSSDL");
String strIndexFileDir = "e:¥¥tsindex¥¥index¥¥tstruct";
// ----- For AIX us the following form for the file
//         String strIndexFileDir = "/home/cltadmin/tsindex/tstruct";
pEnt.setIndexDataArea(strIndexFileDir);
String strWorkFileDir = "e:¥¥tsindex¥¥work¥¥tstruct";
// ----- For AIX us the following form for the file
//         String strWorkFileDir = "/home/cltadmin/work/tstruct";
pEnt.setIndexWorkArea(strWorkFileDir);

// ----- Associate document model with index
pEnt.addDocModel(docModel);

// ----- Create text search index that supports sections
dsDef.add((dkEntityDef)pEnt);

// continued...
```


Java (続き)

```
indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);
```

```
dsAdmin.setIndexingRules(indexRules);
```

```
dsTS.disconnect();
dsTS.destroy();
```

完全なサンプル・アプリケーション (TCreateStructDocIndexTS.java) が CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexingRulesTS* indexRules = new DKIndexingRulesTS();

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// doc model instance for indexing rules
DKDocModelTS* docModel2 = new DKDocModelTS();
docModel2->setCCSID(DK_TS_CCSID_00850);
docModel2->setName("SAMPCORPMOD");

// Describe the document model for text document structure
// for files like tstruct.htm above

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMPCORPMOD");
docSection->setName("SAMPCORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMPCORPBODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

// continued...
```

C++ (続き)

```
dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

DKSearchIndexDefTS* pEnt = new DKSearchIndexDefTS(&dsTS);

pEnt->setName("TSTRUCT");
pEnt->setIndexType(DK_TS_INDEX_TYPE_PRECISE);

// This index is text structure document section enabled
pEnt->setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);

pEnt->setLibraryId("LIBSUM");
pEnt->setLibraryClientServices("IMLLSCDL");
pEnt->setLibraryServerServices("IMLLSSDL");
DKString strIndexFileDir = "e:¥¥tsindex¥¥index¥¥tstruct";
//**** for AIX ****
//DKString strIndexFileDir = "/home/cltadmin/tsindex/index/tstruct";
pEnt->setIndexDataArea(strIndexFileDir);
DKString strWorkFileDir = "e:¥¥tsindex¥¥work¥¥tstruct";
//**** for AIX ****
//DKString strWorkFileDir = "/home/cltadmin/tsindex/work/tstruct";
pEnt->setIndexWorkArea(strWorkFileDir);

// Associate document model with index
pEnt->addDocModel(docModel);

// Create text search index that supports sections
dsDef->add(pEnt);

delete pEnt;

indexRules->setIndexName("TSTRUCT");
indexRules->setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules->setDefaultDocModel(docModel2);
dsAdmin->setIndexingRules(indexRules);

delete indexRules;

dsTS.disconnect();
```

完全なサンプル・アプリケーション (TCreateStructDocIndexTS.cpp) が Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから取られています。

以下の例では、索引付けプロセス (非同期) を開始する方法を示しています。システム管理プログラムを使用することによって、索引付けプロセスを開始し、その状況を検査することができます。

Java

```
// ----- Declare datastore and administration
DKIndexFuncStatusTS pIndexFuncStatus = null;
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Start the indexing process
dsAdmin.startUpdateIndex(indexName);

// ----- Get indexing status
pIndexFuncStatus = dsAdmin.getIndexFunctionStatus(indexName,
    DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);
.... // Process the status as appropriate

// ----- Show the scheduled document queue
System.out.println("*getScheduledDocs "
    + pIndexFuncStatus.getScheduledDocs());

// ----- Show the primary document queue
System.out.println("*getDocsInPrimaryIndex "
    + pIndexFuncStatus.getDocsInPrimaryIndex());

// ----- Shows the secondary document queue
System.out.println("*getDocsInSecondaryIndex " +
    pIndexFuncStatus.getDocsInSecondaryIndex());
System.out.println("*getDocMessages "
    + pIndexFuncStatus.getDocMessages());
if (pIndexFuncStatus.isCompleted() == true)
{
    // ----- Processing if indexing is completed
}

if (pIndexFuncStatus.getReasonCode() != 0)
{
    dsAdmin.setIndexFunctionStatus(indexName,
        DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS, DK_TS_INDEX_ACTID_RESET);
}

dsTS.disconnect();
dsTS.destroy();
```

完全なサンプル・アプリケーション (TIndexingTS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexFuncStatusTS* pIndexFuncStatus = 0;

dsTS.connect(srchSrv,"","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// starts the indexing process
dsAdmin->startUpdateIndex(srchIndex);

// Get indexing status
pIndexFuncStatus = dsAdmin->getIndexFunctionStatus(srchIndex,
    DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);

cout << "***** index status *****" << endl;
cout << "isCompleted " << pIndexFuncStatus->isCompleted() << endl;
cout << "getEnabledId " << pIndexFuncStatus->getEnabledId() << endl;
cout << "getReasonCode " << pIndexFuncStatus->getReasonCode()
    << endl;
cout << "getFuncStopped " << pIndexFuncStatus->getFunctionStopped()
    << endl;
cout << "getStartedLast " << pIndexFuncStatus->getStartedLast()
    << endl;
cout << "getEndedLast " << pIndexFuncStatus->getEndedLast() << endl;
cout << "getStartedExecution " << pIndexFuncStatus->getStartedExecution()
    << endl;
cout << "getScheduledDocs " << pIndexFuncStatus->getScheduledDocs()
    << endl;
cout << "getDocsInPrimaryIndex " << pIndexFuncStatus->getDocsInPrimaryIndex()
    << endl;
cout << "getDocsInSecondIndex " << pIndexFuncStatus->getDocsInSecondIndex()
    << endl;
cout << "getDocMessages " << pIndexFuncStatus->getDocMessages()
    << endl;
if (pIndexFuncStatus->isCompleted() == TRUE)
{
    // indexing completed
}
if (pIndexFuncStatus->getReasonCode() != 0)
{
    dsAdmin->setIndexFunctionStatus(srchIndex,
        DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS,DK_TS_INDEX_ACTID_RESET);
}
delete pIndexFuncStatus;
dsTS.disconnect();
```

完全なサンプル・アプリケーション (TIndexingTS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

状況検査の例については、CMBROOT¥Samples ディレクトリーにある TCheckStatusTS
を参照してください。この例では、キュー要求が、スケジュールされた文書キュー

から 1 次または 2 次キューに移動されたかどうかを検査します。索引付けのエラーが発生したら、テキスト検索インスタンス・ディレクトリー内の imldiag.log ファイルを検査することができます。

以下の例では、前述の定義された文書モデルおよびテキスト検索索引に基づいて、構造化文書テキスト照会を実行する方法を示しています。

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect
dsTS.connect("TMMUF","","","");
// ----- Generate the query string
String cmd = "SEARCH=(COND=($CCSID=850," +
              "DOCMOD=(DOCMODNAME=SAMPCORPMOD," +
              "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));" +
              "OPTION=(SEARCH_INDEX=TMSTRUCT;MAX_RESULTS=5)";
// ----- Execute the query
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);

// ----- Process the results
.....
dsTS.disconnect();
dsTS.destroy();
```

完全なサンプル・アプリケーション (TExecuteStructDocTS.java) が CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreTS dsTS;
dkResultSetCursor* pCur = 0;

dsTS.connect("TMMUF","","","");

DKString cmd = "SEARCH=(COND=($CCSID=850,";
cmd += "DOCMOD=(DOCMODNAME=SAMPCORPMOD,";
cmd += "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));";
cmd += "OPTION=(SEARCH_INDEX=TSTRUCT;MAX_RESULTS=5)";

pCur = dsTS.execute(cmd);

// process the results

dsTS.disconnect();
```

完全なサンプル・アプリケーション (TExecuteStructDocTS.cpp) が Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから取られています。

コンテンツ別のイメージの検索

イメージ・タイプを指定するか、イメージの例を提示すると、イメージ検索サーバーは保管されているイメージを検索できます。

図 16 は、イメージ検索サーバーに接続しているサンプル・アプリケーションを示しています。イメージ検索サーバーでは、イメージ・コンテンツによる照会 (QBIC) のテクノロジーを使用して、類似の色、レイアウト、およびパターンを検索します。

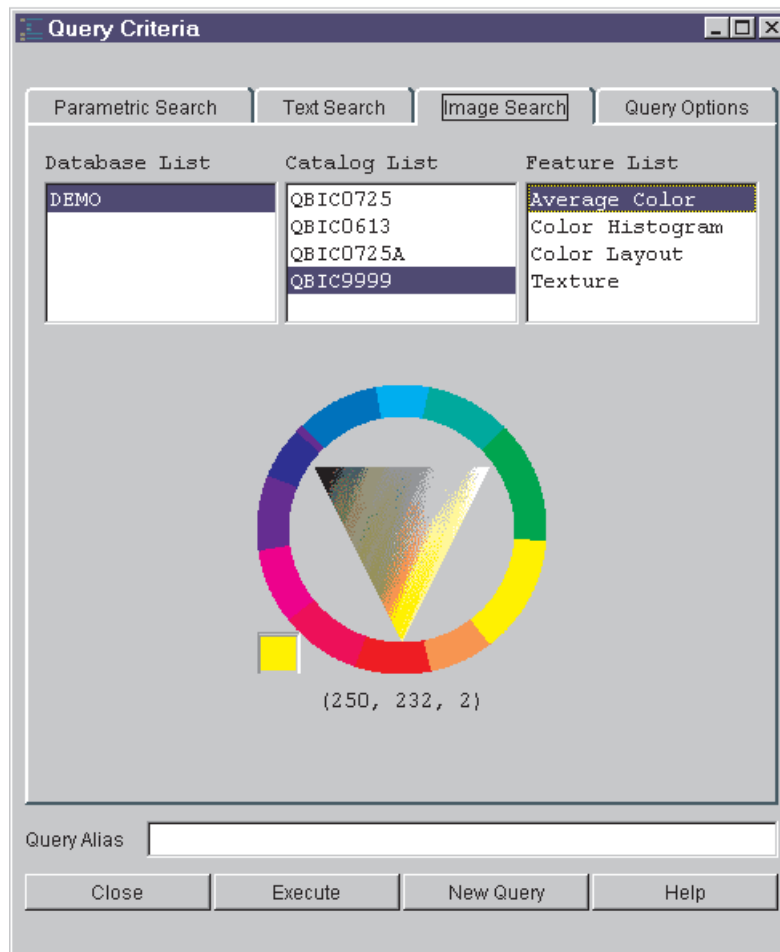


図 16. イメージ検索サンプル・クライアント

イメージ検索の用語と概念について

ここでは、イメージ検索のコンポーネントであるサーバー、データベース、カタログについて説明します。さらに、イメージ検索サーバーと旧バージョンの Content Manager との関係を示します。また、イメージの検索可能ビジュアル特性であるフィーチャーについても説明します。

イメージ検索サーバー、データベース、およびカタログについて: 旧バージョンの Content Manager はイメージを検索する際に、イメージ検索サーバーを使用します。Content Manager アプリケーションは、オブジェクト・サーバー内にイメージ・オブ

ジェクトを保管します。イメージ検索サーバーは、イメージ情報を分析して索引付けします。イメージ検索サーバーは、イメージ自体を保管するわけではありません。

DKDatastoreQBIC オブジェクトによって定義されるコンテンツ・サーバーは、イメージ検索サーバーを表しています。イメージ検索の結果には、Content Manager サーバー中のイメージの場所を記述した ID (項目 ID) が含まれます。これらの ID を、パーツ番号や RepType などの他の結果と一緒に使用して、イメージを取得することができます。

このコンテンツ・サーバーに対する照会は実行することができます。しかし、イメージ検索用のコンテンツ・サーバーでは、追加、更新、検索、および削除操作はサポートしていません。図 17 は、イメージ検索サーバーの例を示しています。

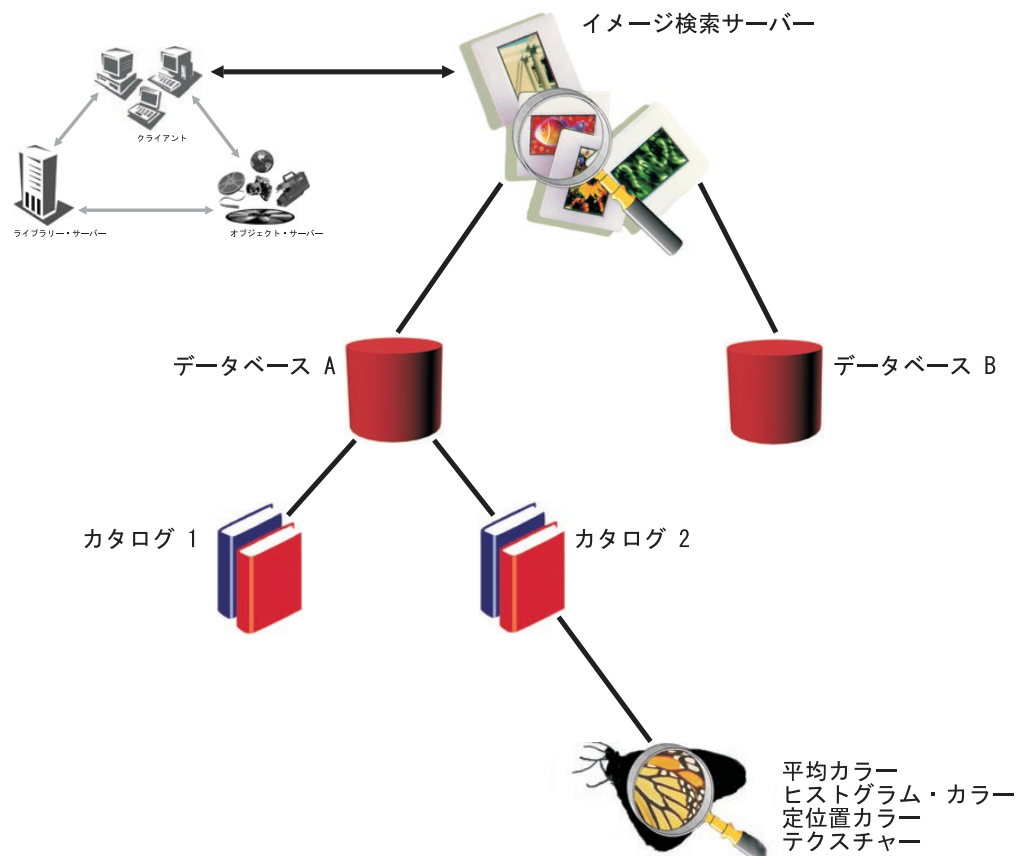


図 17. 旧バージョンの Content Manager システム内のイメージ検索サーバー

イメージ検索サーバーに 1 つまたは複数のデータベースを入れることができます。それぞれのデータベースには、1 つ以上カタログを格納できます。カタログは、イメージの視覚的な特徴 (フィーチャー) に関する情報を保管します。このイメージ検索フィーチャーには、次の 4 種類があります。

- 平均色。
- ヒストグラム色。

- 定位置色 (色レイアウト)。
- テクスチャー。

イメージ検索フィーチャーについて: ここでは、4 つのイメージ検索フィーチャーとその目的について定義します。

平均色 平均色は、主要な色のあるイメージを検索する場合に使います。主要な色が似ているイメージは、平均色も似ています。例えば、赤と黄色の配分の比率が等しいイメージの平均色はオレンジ色になります。

平均の色のフィーチャー名は `QbColorFeatureClass` です。

ヒストグラム色

イメージの色の分布のパーセント比率を測定します。ヒストグラム分析では、イメージ中の異なる色が個別に測定されます。例えば、田園のイメージでは、ヒストグラムの色は青、緑、および灰色の使用率が高くなります。

ヒストグラムの色は、使用されている色の種類が似ているイメージを検索するのに使用してください。ヒストグラムの色のフィーチャー名は `QbColorHistogramFeatureClass` です。

定位置色 (色レイアウト)

定位置色は、イメージ内の指定した領域のピクセルの平均色値を測定します。例えば、中央に明るい赤の物体があるイメージであれば、その部分の定位置色は明るい赤になります。

定位置色のフィーチャー名は `QbDrawFeatureClass` です。

テクスチャー テクスチャーは、特定のパターンを持つイメージを検索するのに使用してください。テクスチャーは、粗度、コントラスト、および方向性を測定します。粗度とは、イメージ内で繰り返される項目のサイズを示します。コントラストとは、イメージ内の輝度の変化の程度を示します。方向性とは、方向がイメージの中で優位を占めているかどうかを示します。例えば木目のイメージは、木目を含む別のイメージとテクスチャーが似ています。

テクスチャーのフィーチャー名は `QbTextureFeatureClass` です。

イメージ検索アプリケーションの使用

イメージ検索クライアント・アプリケーションにより、イメージ照会が作成されて実行され、イメージ検索サーバーによって戻される情報が評価されます。アプリケーションでコンテンツ別にイメージを検索できるようにするには、その前にイメージに索引付けし、イメージ検索データベースにコンテンツ情報を保管しなければなりません。

制限事項: オブジェクト・サーバー中の既存のイメージを索引付けすることはできません。索引付けできるのは、イメージ検索サーバーおよびクライアントのインストール後に、オブジェクト・サーバー内に作成したイメージだけです。 309 ページの図 18 は、クライアントがイメージを取得する例を示しています。

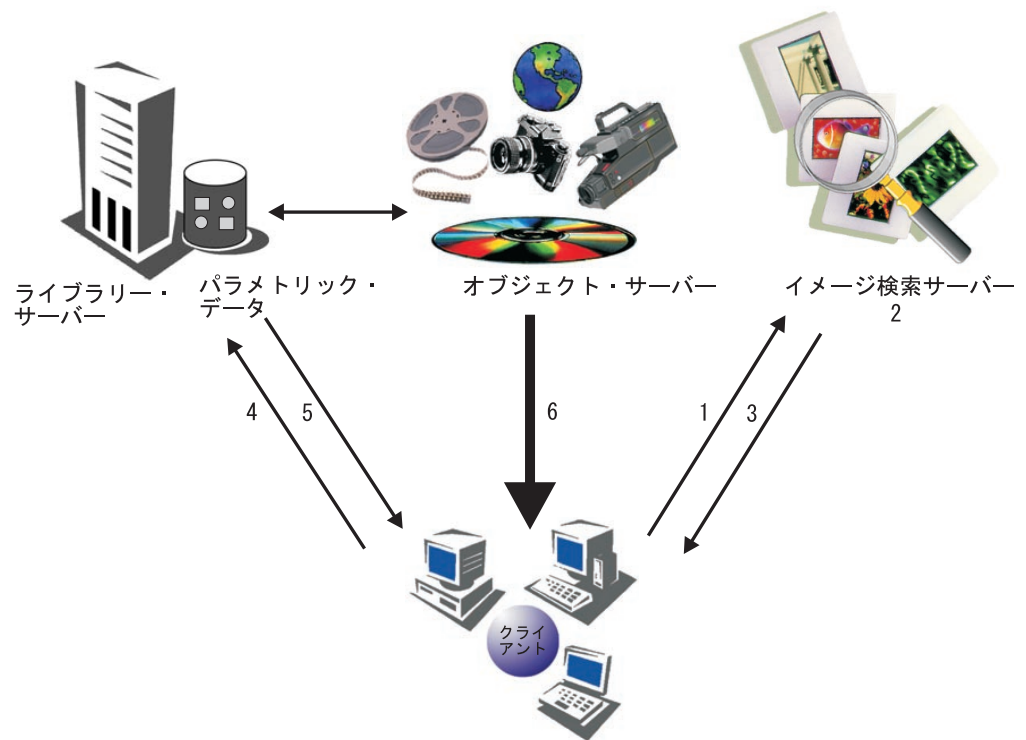


図 18. イメージ検索クライアントがイメージを検索して取得する処理の流れ

イメージ検索を実行するには、以下のようにします。

1. クライアントは、QBIC 照会ストリングを作成し、それをイメージ検索サーバーに送信します。
2. イメージ検索サーバーはその照会ストリングを受け取り、一致するカタログ作成されたイメージを検索します。
3. クライアントは、一致イメージを ID のリストとして受け取ります。個々の一致イメージの ID は以下のものから成ります。
 - 項目 ID。
 - パーツ番号。
 - RepType。
 - ランク。
4. クライアントは、ライブラリー・サーバーのイメージ・パーツと索引情報を要求します。
5. ライブラリー・サーバーは、テキスト記述などの索引情報をクライアントに戻します。ライブラリー・サーバーは、オブジェクト・サーバーに、指定したイメージ・パーツをクライアントに送信するように要求します。
6. オブジェクト・サーバーはイメージ・パーツを送信し、クライアントはそれを受信したことを応答します。

照会の作成

照会を作成する際に、アプリケーションがイメージ検索サーバーに渡す照会ストリングを構成します。イメージ照会とは、検索基準を指定した文字ストリングのことです。検索基準は以下のものから構成されます。

イメージ照会とは、検索基準を指定した文字ストリングのことです。検索基準は以下のものから構成されます。

フィーチャー名

検索で使用するフィーチャー。

フィーチャー値

そのフィーチャーの値。311 ページの表 19 では、照会ストリングで渡すことができるイメージ検索フィーチャーの名前と値を示しています。

フィーチャーの重み

個々のフィーチャーに付けられる相対的な重みまたは重要度。フィーチャーの重みとは、類似度が計算されて照会結果が戻される際に、イメージ検索サーバーによってフィーチャーに付けられる重要度のことです。指定した重みが大きいほど、重要度は高くなります。

最大結果数

照会によって検索されるイメージのタイプを定義することに加え、照会によって戻される一致イメージの最大数を指定することもできます。

照会ストリングの形式は `feature_name value` (`feature_name` はイメージ検索フィーチャー名、`value` はそのフィーチャーに関連付けられた値) です。1 つの照会で複数のフィーチャーを使用する場合は、フィーチャーごとにフィーチャーの名前と値の対を指定しなければなりません。ストリング「and」により個々の対を区切ります。

イメージ検索照会の構文は以下のとおりです。

```
feature_name value  
feature_name value weight
```

1 つの照会内で 1 つのフィーチャーを繰り返すことはできません。1 つの照会内で複数のフィーチャーを指定できます。1 つの照会内で複数のフィーチャーを指定する際は、1 つまたは複数のフィーチャーに重みを割り当てることができます。各フィーチャーの重要度が付けられた照会の形式は、`feature_name value weight` です。ここで `feature_name` はイメージ検索フィーチャー名、`value` はそのフィーチャーに関連した値、`weight` はそのフィーチャーに割り当てられた重みです。`weight` は、キーワード `weight`、等号 (=)、および 0 より大きい実数を組み合わせたものです。

照会によって戻される一致イメージの最大数も指定することができます。最大結果数を指定するには、照会に `and max_results` を付加してください。`max_results` は、キーワード `max`、等号 (=)、および 0 より大きい整数で構成します。311 ページの表 19 はフィーチャー名および値について説明しています。

表 19. イメージ検索照会: 有効なフィーチャー値

フィーチャー名	値
QbColorFeatureClass または QbColor	<p>color = < rgbValue , rgbValue , rgbValue > ここで rgbValue は 0 ～ 255 の整数です。</p> <p>file = < fileLocation , " fileName " > ここで fileLocation は server または client、 fileName はファイルが存在するシステムに適した 形式で指定した完全ファイル・パスです。例え ば、平均色とテクスチャー値を使用して検索を実 行できます。テクスチャー値は、クライアント・ ファイルにあるイメージによって提供されます。 テクスチャーの重みは平均色の 2 倍です。</p> <p>QbColorFeatureClass color= <50, 50, 50> and QbTextureFeatureClass file=<client, "%patterns¥pattern1.gif"> weight=2.0</p>
QbColorHistogramFeatureClass ま たは QbHistogram	<p>histogram = < histList > ここで histList は、1 つ、またはコンマ (,) で区 切られた複数の histClause で構成されます。</p> <p>histClause は (histValue, rgbValue, rgbValue, rgbValue) として指定します。 histValue は 1 ～ 100 の整数 (パーセント値)、 rgbValue は 0 ～ 255 の整数です。</p> <p>file = < fileLocation , " fileName " > ここで fileLocation は server または client、 fileName はファイルが存在するシステムに適した 形式で指定した完全ファイル・パスです。</p>

表 19. イメージ検索照会: 有効なフィーチャー値 (続き)

フィーチャー名	値
QbDrawFeatureClass または QbDraw	<p>description = < " descString " > ここで descString は、ピッカー・ファイルを記述する特殊エンコード・ストリングです。記述ストリングの形式は以下のとおりです。</p> <ol style="list-style-type: none"> 1. Dw,h は、イメージそのものの外部寸法を、幅 w と高さ h で指定します。 2. Rx,y,w,h,r,g,b は、幅 w と高さ h の長方形の配置を、イメージ長方形の左上隅を原点として左上隅が座標 (x,y) になるよう指定します。また、この長方形のカラー値には r (赤)、g (緑)、および b (青) がある必要があります。 3. 区切り文字としてコロン文字 (:) を使用します。 <p>例えば以下のようにすることにより、記述ストリングによって記述した色レイアウト (QbDrawFeatureClass) を検索できます。</p> <p>QbDrawFeatureClass description= <"D100,50:R0,0,50,50,255,0,0"</p> <p>file = < fileLocation , " fileName " > ここで fileLocation は server または client、fileName はファイルが存在するシステムに適した形式で指定した完全ファイル・パスです。</p>
QbTextureFeatureClass または QbTexture	<p>file = < fileLocation , " fileName " > ここで fileLocation は server または client、fileName はファイルが存在するシステムに適した形式で指定した完全ファイル・パスです。</p>

照会の例:

1. 平均の色として赤を検索します。
QbColorFeatureClass color=<255,0,0>
2. 3 色ヒストグラムで、赤が 10%、青が 50%、緑が 40% のものを検索します。
QbColorHistogramFeatureClass histogram=
<(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>
3. 平均の色とテクスチャーの値を使用して検索します。テクスチャー値は、クライアント上のファイル内のイメージによって提供されるものを使用します。テクスチャーの重みは平均色の 2 倍です。
QbColorFeatureClass color=
<50, 50, 50> and QbTextureFeatureClass file=<client, "%patterns%pattern1.gif">
weight=2.0
4. 記述された色レイアウトを検索します。
QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">
5. 平均の色として赤を検索し、戻される一致イメージを 5 つまでに制限します。
QbColorFeatureClass color=<255,0,0> and max=5

照会の実行と検索結果の評価

アプリケーションは、イメージ検索 API を使用して、照会を発行して検索結果を評価します。イメージ検索データベース中の情報がイメージ検索基準と一致すると、一致したイメージの ID が戻されます (複数の場合あり)。この ID は、Content Manager オブジェクト中のイメージ・パーツに対応する動的データ・オブジェクト (DDO) です。

QBIC での接続の確立

イメージ検索には、コンテンツ・サーバーに対する接続および切断用の関数が用意されています。下記の例は、ユーザー ID QBICUSER とパスワード PASSWORD を使用して、QBICSRV という名前のイメージ検索サーバーに接続する方法を示しています。

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // Process as appropriate
dsQBIC.disconnect();
```

完全なサンプル・アプリケーション (TConnectQBIC.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // do some work
dsQBIC.disconnect();
```

完全なサンプル・アプリケーション (TConnectQBIC.cpp) が Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから取られています。

イメージ検索接続を使用すると、アプリケーションからイメージ検索サーバーに接続できます。

接続した後、プログラムではイメージ検索サーバーにアクセスするための関数を使用できるようになります (listDatabases など、イメージ検索カタログに関係しない関数は除く)。処理するためにカタログをオープンするには、openCatalog 関数が必要です。処理が終わると、closeCatalog 関数が呼び出されます。下記の例は、接続、カタログのオープン、カタログのクローズ、および切断の方法を示しています。

Java

```
// ----- Create a QBIC datastore and connect
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- open the catalog
dsQBIC.openCatalog("DEMO", "QBIC0725");
...           // Do some processing
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "QBIC0725");
...           // do some work
dsQBIC.closeCatalog();
dsQBIC.disconnect();
```

イメージ検索サーバーのリスト作成

イメージ検索サーバーには、接続可能なイメージ検索サーバーをリストする関数があります。下記の例は、DKServerInfoQBIC オブジェクトを含むサーバーのリストを取得して、(DKSequentialCollection オブジェクトに) 入れる方法を示しています。DKServerInfoQBIC オブジェクトを取得すると、サーバー名、ホスト名、およびポート番号を取得することができます。

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
.....
DKServerInfoQBIC pSV = null;
String strServerName = null;
String strHostName = null;
String strPortNumber = null;
pCol = (DKSequentialCollection)dsQBIC.listDataSources();
iter = pCol.createIterator();
while (iter.more()) {
    srvDef = (DKServerDefQBIC)iter.next();
    ..... // Process each server as appropriate
}
```

完全なサンプル・アプリケーション (TListCatalogQBIC.java) が CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKServerDefQBIC *pSV = 0;
DKString strServerName;
DKAny a;
long i = 0;
cout << "list servers" << endl;
a = dsQBIC.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    pSV = (DKServerDefQBIC*)((void*)(*pIter->next()));
    strServerName = pSV->getName();
    cout << "Server Name [" << i << "] - " << strServerName << endl;
    delete pSV;
}
delete pIter;
delete pCol;
```

完全なサンプル・アプリケーション (TListCatalogQBIC.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

イメージ検索データベース、カタログ、およびフィーチャーのリスト作成

DKDatastoreQBIC には、イメージ検索サーバー上にあるすべてのイメージ検索データベース、カタログ、およびフィーチャーをリストするための関数があります。このリストは、DKIndexQBIC オブジェクトを含む DKSequentialCollection オブジェクトに戻されます。DKIndexQBIC オブジェクトを取得すると、データベース、カタログ、およびフィーチャー名を取得することができます。下記の例は、データベース、カタログ、およびフィーチャーを取得する方法を示しています。

Java

```
// ----- Create the datastore and connect
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");

// ---- Get the list of servers
col = (DKSequentialCollection)dsQBIC.listDataSources();
iter = col.createIterator();
while (iter.more()) {
    srvDef = (DKServerDefQBIC)iter.next();
    .....    // Process each server as appropriate
}

// ----- Get the list of QBIC Databases
col = (DKSequentialCollection)dsQBIC.listEntities();
iter = col.createIterator();
while (iter.more()){
    dbDef = (DKDatabaseDefQBIC)iter.next();
    // ----- Get the list of catalogs for the database
    col2 = (DKSequentialCollection)dbDef.listSubEntities();
    iter2 = col2.createIterator();
    while (iter2.more()){
        catDef = (DKCatalogDefQBIC)iter2.next();
        // ----- Get the list of features for the catalog
        col3 = (DKSequentialCollection)catDef.listAttrs();
        iter3 = col3.createIterator();
        while (iter3.more()){
            featDef = (DKFeatureDefQBIC)iter3.next();
            .... // Process the features as appropriate
        }
    }
}
dsQBIC.disconnect();
dsQBIC.destroy();
.....
```

完全なサンプル・アプリケーション (TListCatalogQBIC.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKSequentialCollection *pCol2 = 0;
dkIterator *pIter2 = 0;
DKSequentialCollection *pCol3 = 0;
dkIterator *pIter3 = 0;
DKDatabaseDefQBIC *pEntDB = 0;
DKCatalogDefQBIC *pEntCat = 0;
DKString strCatName;
DKString strDBName;
DKString strFeatName;
DKFeatureDefQBIC *pAttr = 0;
DKAny a;
DKAny *pA = 0;
long i = 0;
long j = 0;
long k = 0;
cout << "connecting to datastore" << endl;
dsQBIC.connect("QBICSRV","USERID","PW");
cout << "list databases " << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsQBIC.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pEntDB = (DKDatabaseDefQBIC*)((void*)(*pIter->next()));
    strDBName = pEntDB->getName();
    cout << "database name [" << i << "] - " << strDBName << endl;
    cout << "  list catalogs for DB " << strDBName << endl;
    pCol2=(DKSequentialCollection*)((dkCollection*)pEntDB->listSubEntities());
    pIter2 = pCol2->createIterator();
    j = 0;
    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pEntCat = (DKCatalogDefQBIC*) pA->value();
        strCatName = pEntCat->getName();
        cout << "catalog name [" << j << "] - " << strCatName << endl;
        pCol3=(DKSequentialCollection*)((dkCollection*)pEntCat->listAttrs());
        pIter3 = pCol3->createIterator();
        k = 0;
        while (pIter3->more() == TRUE)
        {
            k++;
            pA = pIter3->next();
            pAttr = (DKFeatureDefQBIC*) pA->value();
            cout << "  Attribute name [" << k << "] - "
                << pAttr->getName() << endl;
            cout << "    datastoreName " << pAttr->datastoreName()
                << endl;
            cout << "    datastoreType " << pAttr->datastoreType()
                << endl;
            cout << "    attributeOf " << pAttr->getEntityName()
                << endl;
            delete pAttr;
        }
    }
}
// continued...
```

C++ (続き)

```
        delete pIter3;
        delete pCol3;
        delete pEntCat;
    }
    cout << " " << j << " features listed for catalog: "
        << strCatName << endl;
    delete pIter2;
    delete pCol2;
    delete pEntDB;
}
delete pIter;
delete pCol;
cout << i << " databases listed" << endl;
dsQBIC.disconnect();
```

完全なサンプル・アプリケーション (TListCatalogQBIC.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

DDO を使用したイメージ検索情報の表現

DKDatastoreQBIC に関連付けられた DDO には、イメージ検索結果を表す特定の情報が含まれています。イメージ照会結果の DDO は、項目内のイメージ・パーツに対応しています。それには以下の標準属性の集合があります。

DKDLITEMID

イメージ・パーツが属する項目の項目 ID。この項目 ID は、コンテンツ・サーバーから項目全体を取り出すのに使います。

DKPARTNO

このイメージ・パーツのパーツ番号 (整数)。これは、項目 ID と共に、コンテンツ・サーバーからこのパーツを取得するのに使います。

DKREPTYPE

表現タイプ (RepType) のストリング。デフォルト値は FRN\$NULL です。この属性は予約済みです。

DKRANK

このパーツとイメージ照会の結果セットとの関係を示すランク (整数)。このランクは 0 ～ 100 の範囲内です。ランクが高いほど一致の程度が大きくなります。

イメージ検索 DDO の PID には以下の情報があります。

コンテンツ・サーバー・タイプ
QBIC。

コンテンツ・サーバー名
コンテンツ・サーバーに接続するために使用されるサーバー名。

ID 結果セット中の DDO のシーケンス番号 (0 から始まる)。

属性値は必ずオブジェクトでなければならないという規則があります。

イメージ照会の処理

ここでは、イメージ照会の実行および評価方法について説明します。

イメージ照会の実行

DKDatastoreQBIC の dkQuery のインスタンスを使用すると、照会を実行して結果を取得するための照会オブジェクトを作成することができます。下記の例は、イメージ照会オブジェクトを作成し、それを実行する方法を示しています。照会が実行されると、結果は DKResults コレクションに戻されます。

Java

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColor color=<255, 0, 0>";
DKNameValuePair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- Open the catalog
dsQBIC.openCatalog("DEMO", "qbic0725");

...    // Process as appropriate

// ----- Create the query and run it
dkQuery pQry = dsQBIC.createQuery(cmd, DK_IMAGE_QL_TYPE, parms);
pQry.execute(parms);
// ----- Get the results and process
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
    // Process the DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
...
```

完全なサンプル・アプリケーション (SampleIQryQBIC.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreQBIC* dsQBIC;  
dsQBIC = new DKDatastoreQBIC();  
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");  
dsQBIC->openCatalog("DEMO", "qbic0725");  
DKAny* element;  
DKDDO* item;  
DKString cmd = "QbColor color=<255, 0, 0>";  
dkQuery* pQry = dsQBIC->createQuery(cmd);  
pQry->execute();  
DKAny any = pQry->result();  
DKResults* pResults = (DKResults*)((dkCollection*)any);  
dkIterator* pIter = pResults->createIterator();  
while (pIter->more())  
{  
    element = pIter->next();  
    item = (DKDDO*)element->value();  
    // Process the DKDDO  
    ...  
}  
delete pIter;  
delete pResults;  
delete pQry;  
dsQBIC->closeCatalog();  
dsQBIC->disconnect();
```

完全なサンプル・アプリケーション (TSampleIQryQBIC.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

コンテンツ・サーバーからのイメージ照会の実行

別の方法として、DKDatastoreQBIC の execute 関数を使用して照会を実行することができます。結果は、dkResultSetCursor オブジェクトに戻されます。下記の例は、コンテンツ・サーバーに対してイメージ照会を実行する方法を示しています。結果は、dkResultSetCursor オブジェクトに戻されます。

Java

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColorFeatureClass color=<255, 0, 0>";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");
// ----- Execute the query from the datastore
dkResultSetCursor pCur = dsQBIC.execute(cmd, DK_IMAGE_QL_TYPE, parms);
while (pCur.isValid())
{
    item = pCur.fetchNext();
    ....    // Process the DKDDO
}
pCur.destroy();
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

完全なサンプル・アプリケーション (TExecuteQBIC.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
cout << "datastore connected" << endl;
dsQBIC->openCatalog("DEMO", "qbic0725");
DKString cmd = "QbColorFeatureClass color=<255, 0, 0>";
dkResultSetCursor* pCur = dsQBIC->execute(cmd);
DKDDO* item = 0;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        // Process the DKDDO
        ...
        delete item;
    }
}
delete pCur;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

完全なサンプル・アプリケーション (TExecuteQBIC.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

コンテンツ・サーバーからのイメージ照会の評価

DKDatastoreQBIC には、照会を評価するための関数もあります。下記の例は、コンテンツ・サーバーからのイメージ照会を評価する方法を示しています。結果は DKResults コレクションに戻されます。

Java

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
DKNameValuePair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");

// ----- Use evaluate to run the query
DKResults pResults=(DKResults) dsQBIC.evaluate(cmd,DK_IMAGE_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
    ...    // Process the DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

C++

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
dsQBIC->openCatalog("DEMO", "qbic0725");
DKAny* element;
DKDDO* item;
DKString cmd = "QbColor color=<255, 0, 0>";
DKAny any = dsQBIC->evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*)any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more())
{
    element = pIter->next();
    item = (DKDDO*)element->value();
    // Process the DKDDO
    ...
}
delete pIter;
delete pResults;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

イメージ検索エンジンの使用

イメージ検索サーバーを使用することにより、平均色、色のパーセント、色レイアウト、およびテクスチャーのいずれか 1 つのフィーチャーに基づく照会を指定することができます。1 つの照会内で複数のフィーチャーを指定することもできます。

照会の結果には、項目 ID、パーツ番号、表現タイプ、およびランキング情報が含まれます。その情報を使うことにより、イメージ・コンテンツ取得のための XDO を作成することができます。

イメージ検索用に索引付けされるデータのロード

Content Manager サーバーにデータをロードしてイメージ検索サーバーによって索引付けするためには、Content Manager 索引クラス、イメージ検索データベース、およびイメージ検索カタログを作成しなければなりません。このデータベースは、イメージ検索カタログのコレクションになります。カタログには、イメージの視覚的フィーチャーに関するデータが含まれています。

索引付けするには、イメージ検索フィーチャーをカタログに追加する必要があります。サポートされているすべてのフィーチャーをカタログに追加してください。

イメージ検索データベースおよびカタログを作成する際には、イメージ検索サーバー稼働していなければなりません。ご使用の環境が正しくセットアップされているかを確認してください。

データが Content Manager にロードされたなら、イメージ・キューにイメージを入れることができます。システム管理プログラムで、「**イメージ・キューの処理 (Process Image Queue)**」を選択してください。索引付けが完了したら、イメージ検索を実行できます。

検索エンジンを使用した既存の XDO の索引付け

指定した検索エンジンを使用して、既存の XDO を索引付けすることができます。以下の例では、DKBlobDL クラスの `setToBeIndexed` 関数を呼び出しています。

Java

```
try
{
    // ----- Create the datastore and connect
    DKDatastoreDL dsDL = new DKDatastoreDL();
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");

    // ----- Create the XDO and PID and set attributes
    DKBlobDL axdo = new DKBlobDL(dsDL);
    DKPidXDODL apid = new DKPidXDODL();
    apid.setPartId(partId);
    apid.setPrimaryId(itemId);
    axdo.setPidObject(apid);

    // ----- Set search engine information
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo.setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    // ----- Call setToBeIndexed on the XDO
    axdo.setToBeIndexed();

    dsDL.disconnect();
    dsDL.destroy();
}
catch (DKException exc)
{
    ... // Handle the DKException
}
catch (Exception exc)
{
    ... // Handle the Exception
}
```


C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "N2JJBERBQFK@WTVL";
    repType = "FRN$NULL";
    partId = 10;
    if (argc == 1)
    {
        cout<<"invoke: indexPartxs <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: indexPartxs "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    cout << "connecting Datastore" << endl;
    try
    {
        //replace following with your library server, userid, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setId(itemId);
        axdo ->setPid(apid);
        axdo ->setRepType(repType);
        cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
        cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
        cout<<"repType= "<<axdo->getRepType()<<endl;
    }

    // continued...
```

C++ (続き)

```
//--- set searchEngine -----
cout<<"set search engine and setToBeIndexed()"<<endl;
DKSearchEngineInfoDL aSrchEx;
aSrchEx.setSearchEngine("SM");
aSrchEx.setSearchIndex("TM-TMINDEX");
aSrchEx.setSearchInfo("ENU");
axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
axdo->setToBeIndexed();
cout<<"setToBeIndexed() done..."<<endl;

delete apid;
delete axdo;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

結合照会の使用

結合照会 は、パラメトリック照会とテキスト照会の組み合わせを実行します。有効範囲は指定することも、しないことも可能です。有効範囲 は、前のパラメトリック照会またはテキスト照会から作成される DKResults オブジェクトです。結果は有効範囲と個々の照会結果の交点になります。したがって、注意深く照会を作成し有効範囲を指定しないと、個々の照会結果に共通部分がなくなって結合照会の結果が意味のないものになってしまう可能性があります。

少なくとも 1 つのパラメトリック照会と 1 つのテキスト照会がある場合、結果 DDO には属性 DKRANK があり、これは文書に属する一致パーツの最高ランクを意味します。

制限事項: 結合照会での照会の場合、検索エンジンとの接続は、照会ごとに別のものを使用しなければなりません。同一の接続を使用して複数の照会を送ることはできません。

パラメトリック照会とテキスト照会の結合

有効範囲を指定しない 1 つのパラメトリック照会と 1 つのテキスト照会で構成される結合照会を実行するには、1 つの結合照会オブジェクトを作成し、入力パラメーターとして 2 つの照会を渡して、結合照会によって実行されるようにします。例えば、以下ようになります。

Java

```
// ----- Create a pre-Version 8.1 Content Manager datastore and connect
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Create a text search datastore and connect
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' '); // TM is a local alias for
...                          // the Text Search Engine server

// ----- Generate the parametric query string and create the query
String pquery = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
    (DKParametricQuery) dsDL.createQuery(pquery, DK_CM_PARAMETRIC_QL_TYPE, null);

// ----- Generate the text query string and create the query
String tquery = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery, DK_CM_TEXT_QL_TYPE, null);

// ----- Create the combined query
DKCombinedQuery cq = new DKCombinedQuery();

// ----- Package the queries in DKNVPair as input parameters
DKNVPair par[] = new DKNVPair[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].setName(DK_PARM_END); // to signal the end of parameter list

// ----- Execute the combined query
cq.execute(par);

// ----- Get the results
DKResults res = (DKResults) cq.result();
if (res != null) {
    ... // process the results
}
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKDatastoreTS dsTS;
// TM is a local alias for the Text Search Engine server
dsTS.connect("TM","", ' ');
// create a parametric query
DKString pquery="SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery* pq =
    (DKParametricQuery*) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, NULL);

// create a text query
DKString tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
    (DKTextQuery*) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, NULL);

// create a combined query
DKCombinedQuery* cq = new DKCombinedQuery();

// package the queries in DKNVPair as input parameters
DKNVPair par[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARM_END);

// execute the combined query
cq->execute(par);

// get the results
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
    // process the results
    ...
}
```

最後の if 文は、DKResults オブジェクトが非ヌルの場合を対象にするために必要です。

有効範囲の使用

有効範囲として使用したい DKResults オブジェクトがある場合、そのオブジェクトを追加の照会パラメーターとして渡します。以下の例は、結合照会の有効範囲として DKResults オブジェクトを使用する方法を示しています。

Java

```
// ----- This scope is the result of a parametric query
DKResults scope;
// ----- This scope is the result of a previous text query
DKResults tscope;

// ----- Package the query in array of DKNVPairs as input parameters
DKNVPair par[] = new DKNVPair[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);

// ----- Execute the combined query
cq.execute(par);
....
```

C++

```
DKResults* scope;    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some parametric query
DKResults* tscope    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some text query

...
// package the query in DKNVPair as input parameters
DKNVPair par[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);
// execute the combined query
cq->execute(par);
...
```

結合照合の結果を、別の結合照合の有効範囲として使用することもできます。さらにその結果を照会することもできます。

ランキング

結合照会に少なくとも 1 つのテキスト照会が含まれている場合、結果の DDO には属性 DKRANK が付けられます。この属性は保管されず、テキスト検索エンジンによって毎回計算されます。ランクの値は、テキスト照会の条件を満たす文書中パーツの最大ランクに対応します。

ヒント

複数のパラメトリック照会と有効範囲がある場合、完全な照会を 1 つ実行する方が効率的です。テキスト照会の場合も同様です。

照会オプション "MAX_RESULTS=nn" は、戻される結果の数を制限します。結果はランク別に降順にソートされるので、通常はこのオプションはテキスト照会の方により適しているといえます。例えばこのオプションを 10 に設定すると、呼び出し側は、一致ランクが高い方から 10 番目までの結果だけを必要としていることになります。

パラメトリック照会の場合は、照会オプション "MAX_RESULTS=nn" の意味は異なります。ランクの概念がないので、呼び出し側は最初の 10 個の結果を得ることになります。この結果には、テキスト照会の結果と共通部分があります。したがって、パラメトリック照会とテキスト照会を結合する際には、パラメトリック照会に照会オプション "MAX_RESULTS=nn" を指定しないようにお勧めします。

旧バージョンの Content Manager のワークフロー関数とワークバスケット関数について

ここでは、旧バージョンの Content Manager のワークフローとワークバスケットの機能について説明します。

旧バージョンの Content Manager ワークフロー・サービスについて

ワークバスケット は、処理したい文書およびフォルダーが入っているコンテナです。ワークフロー は、特定の業務プロセスを表すワークバスケットの順序セットです。フォルダーおよび文書は、ワークフロー内のワークバスケット間を移動します。これによって、アプリケーションで簡単なビジネス・モデルを作成して、そのプロセスを経由することによって処理を完了させることができます。

Content Manager のワークフロー・モデルは、下記の規則に従います。

- ワークバスケットをワークフロー中に入れなくてもよい。
- 1 つのワークバスケットを 1 つまたは複数のワークフローに入れることができる。
- ワークバスケットは、同じワークフロー内に複数入れることができる。
- 文書またはフォルダーは、一度に 1 つのワークフローだけに保管できる。
- 文書またはフォルダーを、ワークフロー中にないワークバスケットに格納できる。

Enterprise Information Portal API には、Content Manager ワークフローを処理するクラスが含まれています。

DKWorkflowServiceDL クラスは、Content Manager のワークフロー・サービスを表します。このクラスには、ワークフロー中の文書またはフォルダーの開始、変更、除去、経路指定、および完了を実行する機能があります。さらに、DKWorkflowServiceDL クラスを使用して、ワークバスケットおよびワークフローについての情報を取得することができます。

DKWorkflowDL クラスおよび DKWorkBasketDL クラスは、それぞれ、ワークフロー項目およびワークバスケット項目のオブジェクト指向の表現です。

接続の確立

ワークフロー・サービスを使用する前に、Content Manager サーバーに接続を確立しなければなりません。コンテンツ・サーバーには、接続および切断用の関数が用意されています。

以下の例は、LIBSRVRN という名前の Content Manager サーバーに、ユーザー ID FRNADMIN およびパスワード PASSWORD を使用して接続する方法を示しています。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
... // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TListWorkFlowWFS.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
... // do some work
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TListWorkFlowWFS.cpp) が Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから取られています。

ワークフローの作成

ワークフローを作成するには、DKWorkflowServiceDL を使います。そのためには、通常、以下の 6 つのステップを実行します。

1. DKWorkflowDL のインスタンスを作成します。
2. ワークフロー名を設定します ("GOLF")。
3. このワークフローにワークバスケットが含まれていないことを示すために、ワークバスケットのシーケンスを設定します ("NULL")。
4. 権限を設定します ("All Privileges")。
5. 後処理を設定します (DK_WF_SAVE_HISTORY)。
6. add() 関数を呼び出します。

この例は、これら 6 つのステップに従ってワークフローを作成しています。

Java

```
// ----- Create the datastore and the CM workflow services
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);

// ----- Set the access option and connect
Object input_option = new Integer(DK_SS_CONFIG);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- Create the CM workflow
DKWorkflowDL newwf = new DKWorkflowDL(wfDL);
newwf.setName("Process claim");
newwf.setWorkBasketSequence((dkCollection *)NULL);
newwf.setAccessList("All Privileges");
newwf.setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf.add();
... // Processing as appropriate
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TCreateDelWorkFlow.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * newwf = new DKWorkflowDL(&wfDL);
newwf->setName("GOLF");
newwf->setWorkBasketSequence((dkCollection *)NULL);
newwf->setAccessList("All Privileges");
newwf->setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf->add();
... // do some work
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TCreateDelWorkFlowWFS.cpp) が Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから取られています。

重要: 通常のユーザー (DK_SS_NORMAL) としてコンテンツ・サーバーに接続した場合、接続後に定義されたワークフローは取得できません。したがって、このサンプルでは DK_SS_CONFIG を使用しています。

ワークフローのリスト作成

下記の例に示すとおり、DKWorkflowServiceDL には、システム内のワークフローをリストするための関数が用意されています。そのリストは、DKWorkflowDL オブジェクトの順次コレクションの中に入れられて戻されます。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get a list of the CM workflows
DKSequentialCollection wfList=(DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
    dkIterator pIter = wfList.createIterator();
    DKWorkflowDL pwf1;
    while (pIter.more())
    {
        pwf1 = (DKWorkflowDL)pIter.next();
        pwf1->retrieve();
        ... // Process as appropriate
    }
}
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TListWorkFlowWFS.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
    (DKSequentialCollection *)wfDL.listWorkFlows();
if (wfList1 != NULL)
{
    dkIterator * pIter1 = wfList1->createIterator();
    DKWorkflowDL * pwf1;
    while (pIter1->more())
    {
        pwf1 = (DKWorkflowDL *)((void*)(*pIter1->next()));
        pwf1->retrieve();
        ... // do some work
        delete pwf1;
    }
}
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TListWorkFlowWFS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

Content Manager ワークバスケットの作成

ワークバスケットを作成するには、DKWorkflowServiceDL を使います。そのためには、通常、以下のステップを実行します。

1. DKWorkBasketDL のインスタンスを作成します。
2. ワークバスケット名を設定します (Hot Items)。

3. 権限を設定します (All Privileges)。
4. add 関数を呼び出します。

下記の例では、これらのステップに従ってワークバスケットを作成しています。通常のユーザー (DK_SS_NORMAL) としてコンテンツ・サーバーに接続した場合、接続後に定義されたワークバスケットは受け取りません。したがって、このサンプルでは DK_SS_CONFIG を使用しています。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
Object input_option = new Integer(DK_SS_CONFIG);
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Create the CM workbasket and set properties
DKWorkBasketDL newwb = new DKWorkBasketDL(wfDL);
newwb.setName("Hot Items");
newwb.setAccessList("All Privileges");
newwb.add();
... // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TCreateDelWorkBasket.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkBasketDL * newwb = new DKWorkBasketDL(&wfDL);
newwb->setName("Hot Items");
newwb->setAccessList("All Privileges");
newwb->add();
... // do some work
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TCreateDelWorkBasket.cpp) が CMBROOT¥Samples¥cpp¥dl ディレクトリーに含まれており、この例はそこから取られています。

ワークバスケットのリスト作成

下記の例に示すとおり、DKWorkflowServiceDL には、システム内のワークバスケットをリストするための関数が用意されています。そのリストは、DKWorkBasketDL オブジェクトの順次コレクションの中に入れられて戻されます。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection wbList=(DKSequentialCollection)wfDL.listWorkBaskets();
if (wbList != null)
{
    dkIterator pIter = wbList.createIterator();
    DKWorkBasketDL pwb1;
    while (pIter.hasMore())
    {
        pwb1 = (DKWorkflowDL)pIter.next();
        pwb1->retrieve();
        ... // do some work
    }
}
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TListWorkBasketWFS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
    (DKSequentialCollection *)wfDL.listWorkBaskets();
if (wbList1 != NULL)
{
    dkIterator * pIter1 = wbList1->createIterator();
    DKWorkBasketDL * pwb1;
    while (pIter1->more())
    {
        pwb1 = (DKWorkBasketDL *)((void*)(*pIter1->next()));
        pwb1->retrieve();
        ... // do some work
        delete pwb1;
    }
}
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TListWorkBasketWFS.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

旧バージョンの Content Manager ワークフロー内の項目のリスト作成

下記の例に示すとおり、DKWorkflowServiceDL には、ワークフロー内の項目 ID を
リストするための関数が用意されています。そのリストは、DKString オブジェクト
の順次コレクションの中に入れられて戻されます。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get the list of CM workflows
KSequentialCollection wfList = (KSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
    dkIterator pIter = wfList.createIterator();
    while (pIter.more())
    {
        DKWorkFlowDL pwf1 = (DKWorkFlowDL)pIter.next();
        // ----- Get the list of items in the CM workflow
        DKSequentialCollection itemList=(DKSequentialCollection)pwf1.listItemIDs();
        if (itemList != null)
        {
            dkIterator iter1 = itemList.createIterator();
            String itemid;
            while (iter1.more())
            {
                itemid = (String)iter1.next();
                // ----- Process the items using the item ID
            }
        }
    }
}
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TListItemWFS.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * wf = new DKWorkflowDL(&wfDL, (char *)itemIDWF);
wf->retrieve;
DKSequentialCollection * pColDoc1 =
    (DKSequentialCollection *)wf->listItemIDs();
if (pColDoc1 != NULL)
{
    dkIterator* pIterDoc1 = pColDoc1->createIterator();
    DKString DocID1;
    while (pIterDoc1->more() == TRUE)
    {
        DocID1 = (DKString)(*pIterDoc1->next());
        ... // do some work
    }
}
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TListItemWFS.cpp) が
Cmbroot/Samples/cpp/dl ディレクトリーに含まれており、この例はそこから
取られています。

旧バージョンの Content Manager ワークフローの実行

DKWorkflowServiceDL には、ワークフローを実行するための関数があります。下記の例は、ワークフロー中の項目を開始し、ワークバスケットに項目を経路指定し、ワークフロー中の項目を完了する方法を示しています。このサンプルを使用するには、以下のように変更を加える必要があります。

- EP8L80R9MHH##QES の代わりに有効な項目 ID を使用する。
- HI7MOPALUPFQ1U47 の代わりに有効なワークフロー ID を使用する。
- E3PP1UZ0ZUFQ1U3M の代わりに有効なワークバスケット ID を使用する。

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
DKString itemID = new String("EP8L80R9MHH#QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID,           // itemID
                      itemIDWF,         // itemIDWB
                      NULL,              // default (1st workbasket)
                      TRUE,              // overload
                      DK_WIP_DEFAULT_PRIORITY // initial_priority
                      );
...                                     // do some work
wfDL.routeWipItem(itemID,               // itemID
                  itemIDWF,             // itemIDWB
                  TRUE,                 // overload
                  DK_NO_PRIORITY_CHANGE // initial_priority
                  );
...                                     // do some work
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
dsDL.destroy();
```

完全なサンプル・アプリケーション (TProcessWFS.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemID = DKString("EP8L80R9MHH#QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID,           // itemID
                      itemIDWF,         // itemIDWB
                      NULL,              // default(1st workbasket)
                      TRUE,              // overload
                      DK_WIP_DEFAULT_PRIORITY // initial_priority
                      );
...                                     // do some work
wfDL.routeWipItem(itemID,               // itemID
                  itemIDWF,             // itemIDWB
                  TRUE,                 // overload
                  DK_NO_PRIORITY_CHANGE // initial_priority
                  );
...                                     // do some work
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
```

完全なサンプル・アプリケーション (TProcessWFS.cpp) が
Cmbroot/Samples/cpp/d1 ディレクトリーに含まれており、この例はそこから
取られています。

OnDemand の使用

Enterprise Information Portal は、Content Manager OnDemand サーバー上のコンテンツにアクセスするためのコネクタおよび関連クラスを提供します。 OnDemand クラスおよび API により、以下を行うことができます。

- OnDemand サーバーとの接続および切断。
- アプリケーション・グループとアプリケーション・グループ・フィールドのリスト作成。
- OnDemand フォルダのリスト作成。
- アプリケーション・グループの照会。
- フォルダまたはアプリケーションのグループ・インターフェースを使用した、文書の検索とリトリブ。
- OnDemand フォルダをネイティブ・エンティティとして扱う統合検索。
- アプリケーション・グループまたはフォルダ・モードにおける、同期および非同期検索。
- OnDemand 文書全体または文書セグメントの検索。
- 指定した OnDemand 文書の論理ビュー・データの取得。
- 指定した OnDemand 文書のリソース・グループの取得。
- 指定した OnDemand 文書の注釈データの検索。
- 注釈の作成と変更。

制限事項: OnDemand は、テキスト検索エンジンと QBIC 検索または結合照会をサポートしていません。

OnDemand サーバーおよび文書の表示

Content Manager OnDemand コンテンツ・サーバーを表す場合は Enterprise Information Portal アプリケーションで DKDatastoreOD を使用し、 OnDemand 文書を DDO として表す場合は DKDDO を使用します。 OnDemand DDO には、以下の情報が含まれています。

- 文書属性名とそれらの値
- 文書データおよび注釈 (DKParts として表現される)
- 文書の論理ビューのコレクション
- リソース・グループ・データ

OnDemand 文書の属性は、プロパティとして DKDDO 内に保管されます。 OnDemand 文書のセグメントおよび注は、DKParts として保管されます。

その他のすべての文書データ (リソース・グループおよびビュー、固定および論理の両方) は、特殊プロパティとして OnDemand DDO に保管されます。以下のプロパティは OnDemand 用に予約されています。

DKViews

論理ビューのコレクション。

DKFixedView

固定ビュー情報が入ります。

DKResource

リソース・グループ・データが入ります。

注:

1. Enterprise Information Portal 管理者は、「初期設定パラメーター (Initialization Parameters)」ページの「追加パラメーター (Additional Parameters)」フィールド内にストリングを指定することによって、 OnDemand コネクターを適切に定義しなければなりません。このストリングは、 ENTITY_TYPE=TEMPLATES;; のようになります。必ず 2 つのセミコロンを入れてください。
2. Enterprise Information Portal バージョン 8.2 では、DKViews、DKFixedView、および DKResource が、それぞれ DKViewDataOD、DKFixedViewDataOD、および DKResourceGrpOD の代わりに使用されます。

OnDemand サーバーへの接続と切断

OnDemand コンテンツ・サーバーにログインするには、 connect メソッドを使用して、サーバー名 (例: ODServer.mycompany.com)、ユーザー ID、およびパスワードを渡します。

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
System.out.println("connecting to datastore ...");
dsOD.connect(ODServer, UserID, Password, "");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");
```

OnDemand サーバーからログアウトするには、 disconnect メソッドを使用します。

Java

```
System.out.println("disconnecting from the datastore ...");
dsOD.disconnect();
dsOD.destroy(); // Finished with the datastore
```

C++

```
cout << "disconnecting from the datastore ..." << endl;
dsOD->disconnect();
delete dsOD;
```


OnDemand に関する情報のリスト作成

OnDemand サーバー用のアプリケーション・グループおよびフォルダーのリストを取得できます。

アプリケーション・グループのリスト作成

DKDatastoreOD の listEntities() メソッドを使用すると、OnDemand のアプリケーション・グループのリストを取得できます。以下の例は、このメソッドを使用する方法について示しています。

Java

```
...
pCol = (DKSequentialCollection) dsOD.listEntities(); //get application groups
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    agDef = (DKAppGrpDefOD)pIter.next();
    strAppGrp = agDef.getName();
    System.out.println("  app grp name[" + i + "]: " + strAppGrp);
    System.out.println("  show attributes for " + strAppGrp + " app grp - ");
    ...
}
```

C++

```
// ----- Show the application groups
// ----- First get the groups
pCol = (DKSequentialCollection*)dsOD.listEntities();
pIter = pCol->createIterator();
int i = 0;
// ----- Process the list
while (pIter->more() == TRUE)
{
    i++;
    agDef = (DKAppGrpDefOD*)((void*)(*pIter->next()));
    strAppGrp = agDef->getName();
    cout << "  app group name[" << i << "]: ==>" << strAppGrp << endl;
    ...
}
```

以下の例は、それぞれのアプリケーション・グループの属性情報の取得について示しています。

Java

```
...
pCol2 = (DKSequentialCollection) dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2.createIterator();
j = 0;

while (pIter2.more() == true)
{
    j++;
    attrDef = (DKFieldDefOD)pIter2.next();
    System.out.println("    Attribute name[" + j + "]: " + attrDef.getName());
    System.out.println("    datastoreType: " + attrDef.datastoreType());
    System.out.println("    attributeOf: " + attrDef.getEntityName());
    System.out.println("    type: " + attrDef.getType());
    System.out.println("    size: " + attrDef.getSize());
    System.out.println("    id: " + attrDef.getId());
    System.out.println("    nullable: " + attrDef.isNullable());
    System.out.println("    precision: " + attrDef.getPrecision());
    System.out.println("    scale: " + attrDef.getScale());
    System.out.println("    stringType: " + attrDef.getStringType());
}

System.out.println("  " + j + " attribute(s) listed for " +
    strAppGrp + " app grp\n");
...
```

C++

```
// ----- Get the attributes for each of the entities(application groups)
pCol2 = (DKSequentialCollection*)dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2->createIterator();
int j = 0;
// ----- List the attributes
while (pIter2->more() == TRUE)
{
    j++;
    attrDef = (DKFieldDefOD*)(void*)(*pIter2->next());
    cout << "attribute name[" << j << "]: ==>" << attrDef->getName() << endl;
    cout << "      datastore type: " << attrDef->datastoreType() << endl;
    cout << "      attribute of: " << attrDef->getEntityName() << endl;
    cout << "      type: " << attrDef->getType() << endl;
    cout << "      size: " << attrDef->getSize() << endl;
    cout << "      ID: " << attrDef->getId() << endl;
    cout << "      precision: " << attrDef->getPrecision() << endl;
    cout << "      scale: " << attrDef->getScale() << endl;
    cout << "      stringType: " << attrDef->getStringType() << endl;
    cout << "      nullable: " << attrDef->isNullable() << endl;
    cout << "      queryable: " << attrDef->isQueryable() << endl;
    cout << "      updatable: " << attrDef->isUpdatable() << endl;
    // ----- Clean up the attribute
    delete attrDef;
}
cout << "  " << j << " attribute(s) listed for the " << strAppGrp
      << " app group\n" << endl;
// ----- Clean up the iterators and collections
if ( pIter2 )
    delete pIter2;
if ( pCol2 )
    delete pCol2;
. . .
```

OnDemand フォルダーのリスト作成

OnDemand コンテンツ・サーバー内のフォルダーのリストを取得するには、listSearchTemplates() 関数を使用できます。

Java

```
...
dsDef = (DKDatastoreDefOD)dsOD.datastoreDef();
pCol = (DKSequentialCollection) dsDef.listSearchTemplates();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    folderName = (String)pIter.next();
    .... // Process the folder as appropriate
}
dsOD.disconnect();
dsOD.destroy();
```

C++

```
. . .
// ----- List the folders
dsDef = (DKDatastoreDefOD*)dsOD.datastoreDef();
pCol = (DKSequentialCollection*)dsDef->listSearchTemplates();
pIter = pCol->createIterator();
i = 0;
// ----- Process the list of folders
while (pIter->more() == TRUE)
{
    i++;
    folderName = (DKString)(*pIter->next());
    cout << "folder name [" << i << "] - " << folderName << endl;
}
// ----- Disconnect
dsOD.disconnect();
. . .
```

OnDemand 文書の取得

OnDemand サーバー内の文書を取得できます。文書をそのパーツおよび属性と共に表示することができます。

特定の文書の検索

次の例では、OnDemand コンテンツ・サーバー内のアプリケーション・グループ (OnDemand Publications) を対象として検索を行います。

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd, DK_CM_SQL_QL_TYPE, parms);
System.out.println("datastore executed query");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();

cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");

DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair( DK_CM_PARM_END, DKAny((long)0) );

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;
if (pCur != 0)
    delete pCur;
```

次の例では、アプリケーション・グループ (OnDemand Publications) を対象として検索を行い、戻された文書を取得します。

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
System.out.println("datastore executed query");

while (pCur.isValid())
{
    DKDDO p = pCur.fetchNext();
    if (p != null)
    {
        String idstr = ((DKPid)p.getPidObject()).pidString();
        System.out.println(" pidString : " + idstr);
        DKPid pid = new DKPid (idstr);
        DKDDO ddoold = p;
        short id, docType = 0;
        if ((id = ddoold.propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
            docType = ((Short)ddoold.getProperty(id)).shortValue();
        if (docType == DK_CM_DOCUMENT)
        {
            System.out.println("create a new DDO with a cloned pid to retrieve!");
            p = dsOD.createDDO(ddoold.getObjectType(), DK_CM_DOCUMENT);
            p.setPidObject(pid);
            try
            {
                dsOD.retrieveObject((dkDataObject)p);
            }
            catch (DKException exc)
            {
                System.out.println("Exception name " + exc.name());
                System.out.println("Exception message " + exc.getMessage());
                System.out.println("Exception error code " + exc.errorCode());
                System.out.println("Exception error state " + exc.errorState());
                exc.printStackTrace();
            }
        }
    }
}
pCur.destroy(); // Finished with the cursor
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair( DK_CM_PARM_END, DKAny((long)0) );

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;

if (pCur != 0)
{
    while (pCur->isValid())
    {
        DKDDO* p = pCur->fetchNext();
        DKDDO* ddoold = p;
        DKString pidStr = ((DKPid*)ddoold->getPidObject())->pidString();
        DKPid* pid = new DKPid(pidStr);
        short id, docType = 0;
        DKAny a;
        if ((id = ddoold->propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
        {
            a = ddoold->getProperty(id);
            if (a.typeCode() == DKAny::tc_ushort)
                docType = (short)(USHORT)a;
            else
                docType = a;
        }
        if (docType == DK_CM_DOCUMENT)
        {
            cout << "create the DDO from the pidstring..." << endl;
            p = dsOD->createDDO(ddoold->getObjectType(), DK_CM_DOCUMENT);
            p->setPidObject(pid);

            dsOD->retrieveObject((dkDataObject*)p);
        }
        delete pid;
        delete ddoold;
    }
    delete pCur;
}
```

文書と、それらのパーツおよび属性の表示

以下の例では、照会で見つかった文書を、そのパーツおよび属性と共に表示しています。

Java

```
//-----For each data item, get the attributes
//-----numDataItems is the number of data items
for (j = 1; j <= numDataItems; j++)
{
    a = p.getData(j)
    strDataName = p.getDataName(j);
    System.out.println("    " + j + ". Attribute Name: " + strDataName );
    System.out.println("        type: " + p.getDataPropertyByName
        (j,DK_PROPERTY_TYPE));

    System.out.println("        nullable: " +
        p.getDataPropertyByName (j,DK_PROPERTY_NULLABLE));
    if (strDataName.equals(DKPARTS) == false &&
        strDataName.equals("DKResource") == false &&
        strDataName.equals("DKViews") == false &&
        strDataName.equals("DKLargeObject") == false &&
        strDataName.equals("DKFixedView") == false &&
        strDataName.equals("DKAnnotations") == false)
    {
        System.out.println("        attribute id: " +
            p.getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID));
    }
    //-----Check for the type of the attribute
    if (a instanceof String)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof Integer)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof Short)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKDate)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKTime)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKTimestamp)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof dkCollection)
    {
        System.out.println("        Attribute Value is collection");
        pCol = (dkCollection)a;
        pIter = pCol.createIterator();
        i = 0;
        while (pIter.more() == true)
        {
            i++;
            a = pIter.next();
            pDO = (dkDataObjectBase)a;
        }
    }
}

// continued...
```


Java (続き)

```
if (pDO.protocol() == DK_XDO)
{
    System.out.println("      dkXDO object " + i + " in collection");
    pXDO = (dkXDO)pDO;
    DKPidXDO pid2 = pXDO.getPidObject();
    System.out.println("          XDO pid string: " +
                      pid2.pidString());
    //----- Retrieve and open instance handler for an XDO
    pXDO.retrieve();
    // pXDO.open();
}
}
}
else if (a != null)
{
    System.out.println("      Attribute Value: " + a.toString());
    if (strDataName.equals("DKResource") ||
        strDataName.equals("DKFixedView") ||
        strDataName.equals("DKLargeObject"))
    {
        pDO = (dkDataObjectBase)a;

        if (pDO.protocol() == DK_XDO)
        {
            System.out.println("      dkXDO object ");
            pXDO = (dkXDO)pDO;
            DKPidXDO pid2 = pXDO.getPidObject();
            System.out.println("          XDO pid string: " +
                              pid2.pidString());
            // Retrieve and open instance handler for an XDO
            pXDO.retrieve();
            // pXDO.open();
        }
    }
}
```

C++

```
DKDDO *p = 0;
DKAny a;

for (j = 1; j <= numDataItems; j++)
{
    a = p->getData(j);
    strDataName = p->getDataName(j);

    cout << " " << j << ". Attribute Name: " << strDataName << endl;
    cout<<"type: "<< p->getDataPropertyByName(j,DK_PROPERTY_TYPE)<<endl;
    cout << "nullable: "
        << p->getDataPropertyByName(j,DK_PROPERTY_NULLABLE) << endl;

    if (strDataName != DK_CM_DKPARTS    &&
        strDataName != "DKResource"    &&
        strDataName != "DKViews"       &&
        strDataName != "DKLargeObject" &&
        strDataName != "DKPermissions" &&
        strDataName != "DKFixedView"   &&
        strDataName != "DKAnnotations")
    {
        cout << "    attribute ID: "
            << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
    }

    if (a.typeCode() == DKAny::tc_string)
    {
        DKString astring = a;
        cout << "    attribute Value (string): " << astring << endl;
    }
    else if . . .
    {
        // ----- Handle each of the other types
    }
    else if (a.typeCode() != DKAny::tc_null)
    {
        cout << "    Attribute Value (non NULL): " << a << endl;
        if (strDataName == "DKResource" ||
            strDataName == "DKFixedView" ||
            strDataName == "DKLargeObject")
        {
            pDO = (dkDataObjectBase*)a;
            if (pDO->protocol() == DK_XDO)
            {
                cout << "    dkXDO object " << endl;
                pXDO = (dkXDO*)pDO;
                pidXDO = (DKPidXDOOD*)pXDO->getPid();
                cout << "    XDO PID string: " << pidXDO->pidString() << endl;
                // ----- Retrieve and open instance handler for an XDO
                pXDO->retrieve();
            }
        }
    }
    else cout << " Attribute Value is NULL" << endl;
}
```

完全なアプリケーションについては、CMBROOT¥samples¥cpp¥od ディレクトリ
ーにある TRetrieveOD.cpp を参照してください。

OnDemand フォルダ・モードの使用可能化

OnDemand フォルダ・モードを使用可能にするには、以下のストリング

ENTITY_TYPE=TEMPLATES

が、接続ストリングまたは構成ストリングの一部として OnDemand コネクターへ渡されなければなりません。例えば、構成ストリングは、以下のようになります。

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

例えば、コネクター・ストリングは、以下のようになります。

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();  
dsOD.connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");  
dsOD->connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

非同期検索

OnDemand コネクターは、統合検索および直接非同期検索の両方をサポートします。非同期検索では、メイン・スレッドを結び付けないため、いつでも検索を取り消すことができます。検索を終了するには、「検索テンプレート・ビューアー (Search Template Viewer)」上の「**検索終了 (Stop Search)**」ボタンを押します。

「検索テンプレート・ビューアー (Search Template Viewer)」上の「最大ヒット数 (max hits)」プロパティは、戻される結果の最大数を制限します。

OnDemand コネクターは、アプリケーション・グループ・モードでの AIX クライアントからの同期および非同期検索もサポートします。

WHERE userid LIKE '%' などの検索基準を使用する場合、クライアントへ戻される文書の結果数によっては、クライアントのマシン上の使用可能なすべてのメモリーが消費される可能性があります。executeWithCallback() メソッドを使用して非同期検索を実行することによって、戻される文書の最大数の値を設定し、いつでも検索を取り消すことができます。

また、結果セットが非常に大きい場合は、Java Virtual Machine (JVM) のデフォルトのスタック・サイズを増やさなければならない場合もあります。Java のスレッドごとのデフォルトのスタック・サイズは 400k で、スタックがオーバーフローする前に 3920 項目を戻すことができます。JVM スタック・サイズを 800k まで増やすと、容量は倍の 7840 項目になります。必要に応じて、JVM スタック・サイズをさらに増やすことができます。

JVM スタック・サイズを増やすには、Java コマンド行オプション `-oss` の後に `nnnK` または `nnM` を使用します。ここで、K はキロバイト、M はメガバイトを意味します。

非同期検索の使用例については、`TRetrieveWithCallback0D`、`TRetrieveFolderWithCallback0D` および `TCallback0D` の各サンプル・プログラムを参照してください。

検索テンプレートとしての OnDemand フォルダー

3 つの EIP ビジュアル JavaBeans である `CMBSearchTemplateList`、`CMBSearchTemplateViewer`、および `CMBSearchResultsView` では、EIP 検索テンプレートが使用されます。`CMBConnection dsType` を `Fed` に設定することによって、統合検索用にこれらの Bean を使用することができます。

OnDemand サーバー上での直接検索用にもこれらの Bean を使用することができます。ログインの前に、以下のようにプロパティを設定します。

```
connection.setDsType("0D");
connection.setServerName(<odserver>);
connection.setConnectionString("ENTITY_TYPE=TEMPLATES");
```

ネイティブ・エンティティとしての OnDemand フォルダー

OnDemand コネクターは、接続ストリング `"ENTITY_TYPE=TEMPLATES"` を指定することによって、OnDemand フォルダーをネイティブ・エンティティとしてマップすることもできます。OnDemand フォルダーをエンティティとして使用すると、統合検索時に役に立つ場合があります。この検索では、OnDemand 用デフォルト・ネイティブ・エンティティである OnDemand アプリケーション・グループよりも、フォルダー定義の処理が容易です。

統合検索の場合、Enterprise Information Portal 管理にサーバー定義を指定します。次に、統合エンティティを定義して OnDemand サーバー上のフォルダーへそれをマップすることができます。

注釈の作成と変更

`CMBDocumentViewer` Bean によって立ち上げられる OnDemand ビューアーを使用する際、`CMBDataManagement` Bean および関連 `CMBAnnotation` クラスを使用することによって OnDemand 文書の注釈を作成、変更、および削除することができます。

トレース

OnDemand コネクターを使用してイベントをトレースすることができます。コネクター `java API` トレースを使用可能にするには、トレース INI ファイル (Java の場

合は cmbodtrace.ini、C++ の場合は cmbodCtrace.ini) を C ドライブのルート (C:¥)、または CMBROOT 変数に指定されているディレクトリーに置きます。AIX の場合は、このファイルを /usr/lpp/cmb/cmgmt に置きます。Solaris の場合は、このファイルを /opt/IBMcmb/cmgmt に置きます。

トレース・ファイルのデフォルト出力ディレクトリーは C:¥Ctrace です。これ以外の場所にトレース情報を書き込むには、トレース INI ファイルを編集します。AIX の場合は、ファイル名をすべて小文字にする必要があるので注意してください。

トレース・ファイルに有効なパス名が指定されていること、および CMBODTRACEDIR を含む行が # 記号で始まっていないことを確認します。次に、サンプルのトレース INI ファイルを示します。

Java

```
#=====
# This is a java property file - not a real INI file!
# *****#
# For windows systems, make sure to use TWO BACK SLASH CHARACTERS (¥¥)
# to separate the directory names!!! =====
# *****#
#
# ***** On windows systems, this file must be located in c:¥ *****
#
# OD Trace File Directory Name property - CMBODTRACEDIR
#
# The CMBODTRACEDIR property defines the directory where the trace files
# will be written to. If the directory name does not exist, it will be
# created.
#
# Please make sure that the directory names are separated by two back slash
# characters to avoid undesirable results.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available. But it is recommended not to change the
# trace output directory name in the middle of an active trace session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_JNI_ONLY
# Trace the entry & exit points in JNI only. Produce the least amount
# of trace.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace the entry and exit points in Java methods and JNI functions.
#
# CMBODTRACESCOPE=JNI_ONLY
# Full trace for the JNI functions only.
#
# If CMBODTRACESCOPE is missing, or set to anything else,
# a full trace will be taken.
#
# To disable the trace, add a leading # character in column 1.
#
# AIX: change the following line to CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/trace
# Sun: change the following line to CMBODTRACEDIR=/opt/IBMcmb/cmgmt/trace
CMBODTRACEDIR=c:¥¥trace
```

C++

```
#=====
# OnDemand Trace INI file
#
# OnDemand Trace File Directory Name key - CMBODTRACEDIR
#
# The CMBODTRACEDIR key defines the directory where the trace files will
# be written to. If the directory name does not exist, it will be created.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available. But it is recommended not to change
# the trace output directory name in the middle of an active trace
# session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace only the entry and exit of all C++ methods and functions.
#
# If CMBODTRACESCOPE is missing, or set to anything else, a full trace
# is taken.
#
# To disable the trace, add a leading # character in column 1 on
# the CMBODTRACEDIR line.
#
[ODCTRACE]
# For AIX: change next line to CMBODTRACEDIR=/usr/lpp/cmb/cmgt/ctrace
CMBODTRACEDIR=D:¥Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

Content Manager ImagePlus for OS/390 の使用

Enterprise Information Portal API は、Content Manager ImagePlus for OS/390 コンテンツ・サーバーを使用する際に、以下の機能をサポートします。

- 1 つまたは複数の ImagePlus サーバーとの接続および切断。
- カテゴリの取得。
- 属性フィールドの取得。
- フォルダーの取得。
- 文書の取得。

アプリケーションで ImagePlus for OS/390 コンテンツ・サーバーを表すには、DKDatastoreIP を使用します。

制限事項: ImagePlus for OS/390 は以下のものをサポートしません。

- テキスト検索エンジンおよび QBIC の検索
- 結合照会
- ワークバスケットおよびワークフロー

エンティティおよび属性のリスト作成

ImagePlus for OS/390 コンテンツ・サーバー用のコンテンツ・サーバーを DKDatastoreIP オブジェクトとして作成し、接続したら、そのコンテンツ・サーバーのエンティティおよび属性を検査することができます。次の例では、ImagePlus for OS/390 コンテンツ・サーバーのエンティティをすべてリストします。

Java

```
// ----- After creating a datastore and connecting
//          dsIP is a DKDatastoreIP object
DKEntityDefIP entDef = null;
DKAttrDefIP attrDef = null;

DKSequentialCollection pCol = (DKSequentialCollection)dsIP.listEntities();
dkIterator pIter = null;

if ( pCol == null )
{
    // ----- Handle if the collection of entities is null
}
else
{
    ... // ----- Process as appropriate
```

完全なサンプル・アプリケーション (TListCatalogIP.java) が CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから取られています。

C++

```
// List entities...
DKEntityDefIP* docDef = 0;
DKAttrDefIP* attrDef = 0;

cout << "---List entities---" << endl;
DKSequentialCollection* pCol = (DKSequentialCollection*)(dsIP.listEntities());
dkIterator* pIter = 0;

if ( pCol == 0 )
{
    cout << "collection of entities is null!" << endl;
}
else
{
    ...
```

完全なサンプル・アプリケーション (TListCatalogIP.cpp) が Cmbroot/Samples/cpp/ip ディレクトリーに含まれており、この例はそこから取られています。

次の例では、DKEntityDefIP の getAttr 関数と listAttrNames 関数を使用して、各エンティティに関連した属性をすべてリストします。

Java

```
// ----- List attributes using listAttrNames and getAttr methods

pIter = pCol.createIterator();
while (pIter.more())
{
    // ----- Iterate over the each entity
    entDef = (DKEntityDefIP)pIter.next();
    System.out.println(" Entity type      : " + entDef.getType() );
    System.out.println(" Entity type name: " + entDef.getName() );

    // ----- Get a list of attributes for the entity
    String[] attrNames = entDef.listAttrNames();
    int count = attrNames.length;
    for (int i = 0; i < count; i++)
    {
        attrDef = (DKAttrDefIP)entDef.getAttr( attrNames[i] );
        System.out.println("  Attr name      : " + attrDef.getName() );
        System.out.println("  Attr id        : " + attrDef.getId() );
        System.out.println("  Entity name    : " + attrDef.getEntityName() );
        System.out.println("  Datastore name: " + attrDef.datastoreName() );
        System.out.println("  Attr type      : " + attrDef.getType() );
        System.out.println("  Attr restrict  : " + attrDef.getStringType() );
        System.out.println("  Attr min val   : " + attrDef.getMin() );
        System.out.println("  Attr max val   : " + attrDef.getMax() );
        System.out.println("  Attr display   : " + attrDef.getSize() );
        System.out.println("  Attr precision: " + attrDef.getPrecision() );
        System.out.println("  Attr scale     : " + attrDef.getScale() );
        System.out.println("  Attr update    ? " + attrDef.isUpdatable() );
        System.out.println("  Attr nullable  ? " + attrDef.isNullable() );
        System.out.println("  Attr queryable? " + attrDef.isQueryable() );
        System.out.println("  ");
    }
}
```


C++

```
// Method 1:
cout << "List attributes using listAttrNames and getAttr functions" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
    docDef = (DKEntityDefIP*)(pIter->next()->value());
    cout << " Document type      : " << docDef->getType() << endl;
    cout << " Document type name: " << docDef->getName() << endl;

    long tmpCount;
    DKString* attrNames;

    // Upon return, tmpCount contains the number of elements in the list.
    attrNames = docDef->listAttrNames(tmpCount);
    for (int i=0; i<tmpCount; i++)
    {
        cout << "   Attr name before lookup " << attrNames[i] << endl;
        attrDef = (DKAttrDefIP*)(docDef->getAttr(attrNames[i]));
        cout << "   Attr name [" << i << "] : " << attrDef->getName() << endl;
        cout << "   Attr id      : " << attrDef->getId() << endl;
        cout << "   Entity name  : " << attrDef->getEntityName() << endl;
        cout << "   Datastore name: " << attrDef->datastoreName() << endl;
        cout << "   Attr type    : " << attrDef->getType() << endl;
        cout << "   Attr restrict : " << attrDef->getStringType() << endl;
        cout << "   Attr min val  : " << attrDef->getMin() << endl;
        cout << "   Attr max val  : " << attrDef->getMax() << endl;
        cout << "   Attr display  : " << attrDef->getSize() << endl;
        cout << "   Attr precision: " << attrDef->getPrecision() << endl;
        cout << "   Attr scale    : " << attrDef->getScale() << endl;
        cout << "   Attr update   ? " << attrDef->isUpdatable() << endl;
        cout << "   Attr nullable ? " << attrDef->isNullable() << endl;
        cout << "   Attr queryable? " << attrDef->isQueryable() << endl;
        cout << "" << endl;
        delete attrDef;
    } // end for

    delete [] attrNames;

} // end while
delete pIter;
```

以下の例は、それぞれのエンティティーに関連付けられた属性のリストを取得する別の方法について示しています。この例では、DKDatastoreIP の listEntityAttrs メソッドを使用します。

Java

```
// --- List attributes using listEntityAttrs method
pIter = pCol.createIterator();
while (pIter.more())
{
    entDef = (DKEntityDefIP)pIter.next();
    System.out.println(" Entity type      : " + entDef.getType() );
    System.out.println(" Entity type name: " + entDef.getName() );

    DKSequentialCollection pAttrCol =
        (DKSequentialCollection)dsIP.listEntityAttrs(entDef.getName());
    if ( pAttrCol == null )
    {
        // ----- Handle if the collection of attributes is null
    }
    else
    {
        dkIterator pAttrIter = pAttrCol.createIterator();
        while (pAttrIter.more())
        {
            attrDef = (DKAttrDefIP)pAttrIter.next();
            System.out.println(" Attr name      : " + attrDef.getName() );
            System.out.println(" Attr id       : " + attrDef.getId() );
            System.out.println(" Entity name    : " + attrDef.getEntityName() );
            System.out.println(" Datastore name: " + attrDef.datastoreName() );
            System.out.println(" Attr type      : " + attrDef.getType() );
            System.out.println(" Attr restrict  : " + attrDef.getStringType() );
            System.out.println(" Attr min val   : " + attrDef.getMin() );
            System.out.println(" Attr max val   : " + attrDef.getMax() );
            System.out.println(" Attr display   : " + attrDef.getSize() );
            System.out.println(" Attr precision: " + attrDef.getPrecision() );
            System.out.println(" Attr scale     : " + attrDef.getScale() );
            System.out.println(" Attr update ?  : " + attrDef.isUpdatable() );
            System.out.println(" Attr nullable ? : " + attrDef.isNullable() );
            System.out.println(" Attr queryable? : " + attrDef.isQueryable() );
            System.out.println(" ");
        }
    }
}
```

C++

```
// Method 2:
cout << "---List attributes using listEntityAttrs function---" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
    docDef=(DKEntityDefIP*)(pIter->next()->value()); //iterator returns DKAny*
    cout << " Document type      : " << docDef->getType() << endl;
    cout << " Document type name: " << docDef->getName() << endl;
    DKSequentialCollection* pAttrCol = (DKSequentialCollection*)
        (dsIP.listEntityAttrs(docDef->getName()));

    if ( pAttrCol == 0 )
    {
        cout << "collection of entity attrs is null for entity "
            << docDef->getName()
            << endl;
    }
    else
    {
        int i=0;
        dkIterator* pAttrIter = pAttrCol->createIterator();
        while (pAttrIter->more())
        {
            i++;
            // ----- The iterator returns a pointer to DKAny
            attrDef = (DKAttrDefIP*)(pAttrIter->next()->value());
            cout << "  Attr name [" << i << "] : " << attrDef->getName() << endl;
            cout << "    Attr id      : " << attrDef->getId() << endl;
            cout << "    Entity name  : " << attrDef->getEntityName() << endl;
            cout << "    Datastore name: " << attrDef->datastoreName() << endl;
            cout << "    Attr type    : " << attrDef->getType() << endl;
            cout << "    Attr restrict: " << attrDef->getStringType() << endl;
            cout << "    Attr min val : " << attrDef->getMin() << endl;
            cout << "    Attr max val : " << attrDef->getMax() << endl;
            cout << "    Attr display : " << attrDef->getSize() << endl;
            cout << "    Attr precision: " << attrDef->getPrecision() << endl;
            cout << "    Attr scale    : " << attrDef->getScale() << endl;
            cout << "    Attr update ? " << attrDef->isUpdatable() << endl;
            cout << "    Attr nullable ? " << attrDef->isNullable() << endl;
            cout << "    Attr queryable? " << attrDef->isQueryable() << endl;
            cout << "" << endl;
            delete attrDef;
        } // end while
        delete pAttrIter;
    }
    delete pAttrCol;
    delete docDef;
} // end while
delete pIter;
}
delete pCol;
```

ImagePlus for OS/390 の照会構文

下記の例は、ImagePlus for OS/390 の照会構文を示しています。

Java

```
SEARCH = (COND=(search_expression), ENTITY={entity_name | mapped_entity_name}  
[, MAX_RESULTS = maximum_results]);  
[OPTION=( [CONTENT={YES | ATTRONLY | NO};] [PENDING={YES | NO};])]
```

C++

```
SEARCH=(COND=(search_expression),ENTITY={entity_name | mapped_entity_name}  
[,MAX_RESULTS=maximum_results]);  
[OPTION=( [CONTENT={YES | ATTRONLY | NO};] [PENDING={YES | NO};])]
```

照会は、以下のパラメーターを使用します。

search_expression

各検索式は 1 つまたは複数の検索基準から成り立っています。検索基準の間ではブール演算子 AND のみを使用できます。

検索基準の形式は以下のとおりです。

{attr_name | mapped_attr_name} operator literal

ここで、

attr_name

検索のベースとするエンティティ属性の名前。

mapped_attr_name

検索のベースとする属性にマッピングされた属性名。

operator

すべての属性は等号 (==) をサポートします。タイプ DATE の属性の場合は、そのほかに次の演算子を使用できます。

> より大きい

< より小さい

>= 以上

<= 以下

literal

リテラル。下記に示すように、数値属性の場合に引用符 (") は使用できません。

FolderType == 9

日付、時刻、およびタイム・スタンプ属性の場合、引用符またはアポストロフィ (') は不要です。しかし、使用することは可能です。例えば、以下のようになります。

ReceiveDate == 1999-03-08

ReceiveDate == '1999-03-08'

ストリング属性の場合、引用符またはアポストロフィ (') は不要ですが、使用することはできません。ストリングにアポストロフィ (') が含まれている場合、ストリングは 2 つのアポストロフィを使用して指定しなければなりません。例えば Folder'1 という値の場合には、下記ようになります。

```
FolderId == 'Folder''1'
```

entity_name

検索対象のエンティティの名前。

mapped_entity_name

検索されるエンティティに対応するエンティティ名。

maximum_results

戻される結果の最大数。

オプション・キーワードには、以下のものがあります。

CONTENT 結果で戻す情報量を制御します。

YES (デフォルト)

文書またはフォルダーの PID、属性、およびその値を設定します。文書にパーツがある場合、XDO PID が設定されます。フォルダー内に文書がある場合、文書 PID が設定されます。

NO 文書またはフォルダーの PID だけを設定します。

ATTRONLY

文書またはフォルダーの PID、属性、およびその値だけを設定します。

PENDING パーツがない保留文書を含めるかどうかを制御します。このオプションは、ENTITY が DOCUMENT に設定されているか、または DOCUMENT にマッピングされているエンティティに設定される場合にのみ適用されます。

YES 結果に保留文書を含めます。

NO (デフォルト)

結果に保留文書を含めません。

Content Manager for AS/400 の使用

Content Manager for AS/400 (VisualInfo for AS/400) 用に提供されている API クラスは、Content Manager 用のものと似ています。

制限事項: Content Manager for AS/400 では、以下はサポートされません。

- テキスト検索エンジンおよび QBIC の検索
- 結合照会
- ワークバスケットおよびワークフロー

エンティティ (索引クラス) および属性のリスト作成

Content Manager for AS/400 コンテンツ・サーバーは、DKDatastoreV4 として表されます。コンテンツ・サーバーを作成してそれに接続すると、Content Manager for AS/400 サーバー用のエンティティ (索引クラス) および属性をリストできます (例を参照)。

Java

```
// ----- After creating a datastore (dsV4) and connecting, get index classes
pCol = (DKSequentialCollection) dsV4.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    icDef = (DKIndexClassDefV4)pIter.next();
    strIndexClass = icDef.getName();
    ... // ---- Process the index classes as appropriate
// ----- Get the attributes
pCol2 = (DKSequentialCollection) dsV4.listEntityAttrs(strIndexClass);
pIter2 = pCol2.createIterator();
j = 0;

    while (pIter2.more() == true)
    {
        j++;
        attrDef = (DKAttrDefV4)pIter2.next();
        ... // ----- Process the attributes
    }
}

dsV4.disconnect();
dsV4.destroy();
```

完全なサンプル・アプリケーション (TListCatalogV4.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
cout << "list index class(es)..." << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsV4.listSchema());
pIter = pCol->createIterator();
i = 0;

while (pIter->more() == TRUE)
{
    i++;
    a = (*pIter->next());
    strIndexClass = a;
    cout << "index class name [" << i << "] - " << strIndexClass << endl;
    cout << " list attribute(s) for " << strIndexClass << " index class:" << endl;
    pCol2 =

(DKSequentialCollection*)((dkCollection*)dsV4.listSchemaAttributes(strIndexClass));
    pIter2 = pCol2->createIterator();
    j = 0;

    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pDef = (DKAttributeDef*) pA->value();
        cout << "    Attribute name [" << j << "] - " << pDef->name << endl;
        cout << "        datastoreType - " << pDef->datastoreType << endl;
        cout << "        attributeOf - " << pDef->attributeOf << endl;
        cout << "        type - " << pDef->type << endl;
        cout << "        size - " << pDef->size << endl;
        cout << "        id - " << pDef->id << endl;
        cout << "        nullable - " << pDef->nullable << endl;
        cout << "        precision - " << pDef->precision << endl;
        cout << "        scale - " << pDef->scale << endl;
        cout << "        string type - " << pDef->stringType << endl;
    }

    cout << " " << j << " attribute(s) listed for "
        << strIndexClass << " index class" << endl;
    pCol2->apply(deleteDKAttributeDef);
    delete pIter2;
    delete pCol2;
}

delete pIter;
delete pCol;
cout << i << " index class(es) listed" << endl;
dsV4.disconnect();
cout << "datastore disconnected" << endl;
```

完全なサンプル・アプリケーション (TListCatalogV4.cpp) が
Cmbroot/Samples/cpp/v4 ディレクトリーに含まれており、このアプリケーションはそこから取られています。

照会の実行

次の例では、Content Manager for AS/400 で照会が実行され、結果が処理されます。

Java

```
// ----- After creating a datastore (dsV4) and connecting, build the
//           query and parameters and execute it
pCur = dsV4.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
...

if (pCur == null)
{
    // ---- Handle if the cursor is null
}

while (pCur.isValid())
{
    p = pCur.fetchNext();
    if (p != null)
    {
        cnt++;
        i = pCur.getPosition();
        System.out.println("\n===== Item " + i + " <=====");
        numDataItems = p.dataCount();
        DKPid pid = p.getPid();
        System.out.println("  pid string: " + pid.pidString());
        k = p.propertyId(DK_CM_PROPERTY_ITEM_TYPE);

        if (k > 0)
        {
            Short sVal = (Short)p.getProperty(k);
            j = sVal.shortValue();
            switch (j)
            {
                case DK_CM_DOCUMENT :
                {
                    ... // Handle if the item is a document ";
                    break;
                }
                case DK_CM_FOLDER :
                {
                    ... // Handle if the item is a folder
                    break;
                }
            }
        }
    }
}

for (j = 1; j <= numDataItems; j++)
{
    a = p.getData(j);
    strDataName = p.getDataName(j);
    ... // Process the attributes as appropriate
    if (strDataName.equals(DKPARTS) == false
        && strDataName.equals(DKFOLDER) == false)
    {
        System.out.println("      attribute id: "
            + p.getDataPropertyByName(j,DK_CM_PROPERTY_ATTRIBUTE_ID));
    }
}
// continued...
```


Java (続き)

```
if (a instanceof String)
{
    System.out.println("        Attribute Value: " + a);
}
else if (a instanceof Integer)
    ... // ---- Handle each type for attribute {
else if (a instanceof dkCollection)
{
    // ---- Handle if attribute value is a collection
    pCol = (dkCollection)a;
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        a = pIter.next();
        pDO = (dkDataObjectBase)a;

        if (pDO.protocol() == DK_CM_PDDO)
        {
            // Process a DDO
            pDDO = (DKDDO)pDO;
            ...
        }
        else if (pDO.protocol() == DK_CM_XDO)
        {
            // Process an XDO
            pXDO = (dkXDO)pDO;
            DKPidXDO pid2 = pXDO.getPid();
            ...
        }
    }
}
else if (a != null)
{
    // Process the attribute
}
else ... // Handle if the attribute is null
}
}
pCur.destroy(); // Delete the cursor when you're done
```

完全なサンプル・アプリケーション (TExecuteV4.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
cout << "executing query..." << endl;
...
pCur = dsV4.execute(cmd);
cout << "  query executed" << endl;
...
cout << "¥n..... Displaying query results ..... ¥n¥n";

...
while (pCur->isValid())
{
    p = pCur->fetchNext();

    if (p != 0)
    {
        cout << "=====> " << "Item " << cnt << " <=====" << endl;
        numDataItems = p->dataCount();
        pid = p->getPid();
        cout << "  Pid String: " << pid.pidString() << endl;
        k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

        if (k > 0)
        {
            a = p->getProperty(k);
            val = a;
            cout << "  *****" << endl;

            switch (val)
            {
                case DK_CM_DOCUMENT :
                {
                    cout << "  Item is a document " << endl;
                    break;
                }
                case DK_CM_FOLDER :
                {
                    cout << "  Item is a folder " << endl;
                    break;
                }
            }

            cout << "  *****" << endl;
        }

        cout << "  Number of Data Items: " << numDataItems << endl;

        for (j = 1; j <= numDataItems; j++)
        {
            a = p->getData(j);
            strDataName = p->getDataName(j);

            switch (a.typeCode())
            {
                case DKAny::tc_string :
                {
                    strData = a;
                    cout << "  attribute name: " << strDataName
                        << ", value: " << strData << endl;
                    break;
                }
            }
        }
    }
}

// continued...
```

C++ (続き)

```
case DKAny::tc_long :
{
    lVal = a;
    cout << " attribute name: " << strDataName
        << ", value: " << lVal << endl;
    break;
}

case DKAny::tc_null :
{
    cout<<" attribute name: "<<strDataName<<" value: NULL "<< endl;
    break;
}

case DKAny::tc_collection :
{
    pdCol = a;
    cout<<strDataName<<" collection name: "<<strDataName << endl;
    cout<<"-----"<<endl;
    pdIter = pdCol->createIterator();
    ushort b = 0;

    while (pdIter->more() == TRUE)
    {
        b++;
        cout << " -----" << endl;
        a = *(pdIter->next());
        pDObase = a;

        if (pDObase->protocol() == DK_PDDO)
        {
            pDDO = (DKDDO*)pDObase;
            cout << " DKDDO object " << b << " in " << strDataName
                << " collection " << endl;
            k = pDDO->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

            if (k > 0)
            {
                a = pDDO->getProperty(k);
                val = a;
                cout << " *****" << endl;

                switch (val)
                {
                    case DK_CM_DOCUMENT :
                    {
                        cout << " Item is a document " << endl;
                        break;
                    }
                    case DK_CM_FOLDER :
                    {
                        cout << " Item is a folder " << endl;
                        break;
                    }
                }
            }
            cout << " *****" << endl;
        }
    }
}

// continued...
```

C++ (続き)

```
        else if (pDObase->protocol() == DK_XDO)
        {
            pXDO = (dkXDO*)pDObase;
            cout << "    dkXDO object " << b << " in " << strDataName
                << " collection " << endl;

        }
    }

    if (pdIter != 0)
    {
        delete pdIter;
    }

    if (b == 0)
    {
        cout << strDataName << " collection has no elements " << endl;
    }

    cout << " -----" << endl;
    break;
}

default:
    cout << "Type is not supported\n";
}

cout<<"type: "<< p->getDataPropertyByName(j,DK_CM_PROPERTY_TYPE)<<endl;
cout<<"nullable: "<< p->getDataPropertyByName(j,DK_CM_PROPERTY_NULLABLE)
    << endl;

    if (strDataName != DKPARTS && strDataName != DKFOLDER)
    {
        cout << "    attribute id: "
            << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
    }
}
cnt++;
delete p;
}
}
cout << "Total Item count is " << cnt-1 << endl;

if (pCur != 0)
    delete pCur;
```

完全なサンプル・アプリケーション (TExecuteV4.cpp) が
Cmbroot/Samples/cpp/v4 ディレクトリーに含まれており、このアプリケーションはそこから取られています。

パラメトリック照会の実行

以下の例では、パラメトリック照会を実行します。

Java

```
// ----- Create the query string and the query object
String cmd = "SEARCH=(INDEX_CLASS=V4DEMO)";
pQry = dsV4.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Run the query
pQry.execute(parms);

System.out.println("number of query results = " + pQry.numberOfResults());

// ----- Processing the query results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
...
```

完全なサンプル・アプリケーション (TSamplePQryV4.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
cout << "query string: " << cmd << endl;
cout << "creating query..." << endl;
pQry = dsV4.createQuery(cmd);
cout << "executing query..." << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "getting query results..." << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsV4.disconnect();
cout << "datastore disconnected" << endl;
delete pQry;
delete pResults;
```

完全なサンプル・アプリケーション (TSamplePQryV4.cpp) が
Cmbroot/Samples/cpp/v4 ディレクトリーに含まれており、このアプリケーション
はそこから取られています。

Domino.Doc の使用

Domino.Doc は、業務文書を編成、管理、および保管し、業務の内外でそれらにアクセスできるようにするための、Lotus Domino ソリューションです。

Domino.Doc は、オープン文書管理 API (ODMA) をサポートしており、これによって ODMA 対応のアプリケーションを使用して、文書を作成、保管、および検索することができます。ODMA は、HTTP または Lotus Notes™ プロトコルを使用して、Domino.Doc サーバーに接続します。

Domino.Doc には、以下の機能があります。

- 1 つまたは複数の Domino.Doc サーバーとの接続および切断。

- バインダーを検索する機能。
- 文書を検索する機能。
- ユーザーが慣れたアプリケーションでの作業を可能にする ODMA 準拠。

制限事項: Domino.Doc は、以下のものをサポートしていません。

- 文書を追加、更新、および削除するメソッド。
- テキスト検索エンジンおよび QBIC の検索。
- 結合照会。
- ワークバスケットおよびワークフロー。

API を使用して Domino.Doc オブジェクトを処理する場合、オブジェクトを戻す式を作成しなければなりません。ここでは、Domino.Doc API の設計、オブジェクトを階層に適合させる方法、および式を作成する方法について説明します。 371 ページの図 19 は、Domino.Doc のオブジェクト・モデルとコンポーネントの関係を示しています。

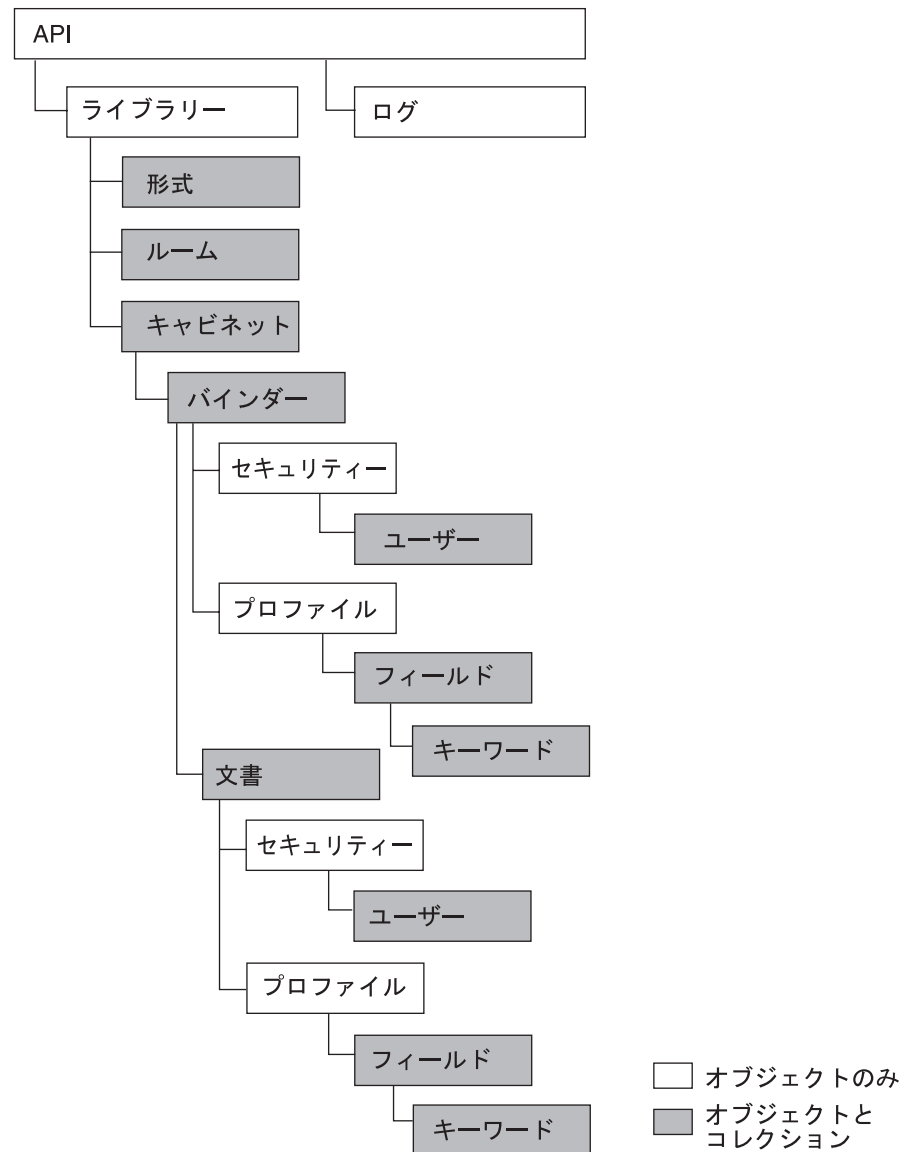


図 19. Domino.Doc オブジェクト・モデル

Domino.Doc に含まれているエレメント (およびそれらを表す API) は、以下のように配置されています。

- ライブラリーには、ルーム (DKRoomDefDD オブジェクト) およびキャビネット (DKCabinetDefDD オブジェクト) が含まれています。
- それぞれのキャビネットには、バインダー (DKBinderDefDD オブジェクト) が含まれています。
- それぞれのバインダーには、プロファイル (DKAttrProfileDefDD オブジェクト) およびセキュリティが含まれています。
- それぞれのバインダーには、文書 (DKDocumentDefDD オブジェクト) が含まれています。
- それぞれの文書には、プロファイル (DKAttrProfileDefDD オブジェクト) およびセキュリティが含まれています。

- それぞれのプロファイルには、フィールド (DKAttrFieldDefDD オブジェクト) が含まれています。
- それぞれのフィールドには、キーワード (DKAttrKeywordDefDD オブジェクト) を含めることができます。

エンティティおよびサブエンティティのリスト作成

次の例では、Domino.Doc のエンティティ (この例ではルーム) とサブエンティティ (この例ではキャビネット、バインダー、および文書) をリストします。

Java

```
...
// ----- Get a list of rooms
dkCollection rooms = dsDD.listEntities();
// ----- Iterate thru the rooms and their subEntities
dkIterator itRooms = rooms.createIterator();
itRooms.reset();
while( itRooms.more() ) {
    ... // Process the rooms and entities
```

完全なサンプル・アプリケーション (TListSubEntitiesDD.java) が CMBROOT¥Samples¥java¥dd ディレクトリーに含まれており、この例はそこから取られています。

C++

```
dkCollection* pColl = domDoc.listEntities();

long nbrEnts = pColl->cardinality();

dkIterator* itEnts = pColl-> createIterator();
while( itEnts->more() )
{ // For each returned dkEntityDef...
    DKRoomDefDD* pEnt = (DKRoomDefDD*)itEnts->next()->value();
    cout << "Room title: " << pEnt->getName() << endl;
    cout << "  Has SubEntities: " << pEnt->hasSubEntities() << endl;

    // print subEntities (Cabinets->Binders->Documents)
    printSubEnts(pEnt, domDoc, 1);

    delete pEnt;
}
delete itEnts;
delete pColl;
```

完全なサンプル・アプリケーション (TListEntitiesDD.cpp) が Cmbroot/Samples/cpp/dd ディレクトリーに含まれており、この例はそこから取られています。

以下の例では、文書属性およびキーワードのリストを取得しています。

Java

```
...
DKAttrProfileDefDD profile = aDocument.getProfile();
dkCollection fields = profile.getFields();

if((fields != null) &&( fields.cardinality() > 0 ))
{
    dkIterator itFields = fields.createIterator();
    while( itFields.more() )
    {
        DKAttrDefDD aField = (DKAttrDefDD)itFields.next();
        // ---- get the keywords
        dkCollection keywords = ((DKAttrFieldDefDD)aField).getKeywords();
        if( keywords != null )
        {
            if( keywords.cardinality() > 0 )
            {
                dkIterator itKeywords = keywords.createIterator();
                while( itKeywords.more() )
                {
                    DKAttrDefDD aKeyword = (DKAttrDefDD)itKeywords.next();
                    // ----- Process the keyword
                }
            }
        }
    }
}
...
```

下記の例では、エンティティ（この場合にはルーム）と関連付けられたサブエンティティ（キャビネット、バインダー、および文書）をリストしています。

C++

```
void printSubEnts( DKEntityDefDD* pEnt, DKDatastoreDD& domDoc, int indents )
{
    // indents: 1=Cabinets; 2=Binders; 3=Documents
    DKString indentation = "";

    for(int i = 0; i < indents; i++)
    {
        indentation += " ";
    }

    if( pEnt->hasSubEntities() )
    {
        dkCollection* pColl = pEnt->listSubEntities();
        long nbrEnts = pColl->cardinality();
        dkIterator* itEnts = pColl-> createIterator();
        while( itEnts->more() )
        {
            DKEntityDefDD* pEnt = (DKEntityDefDD*)itEnts->next()->value();
            cout<< indentation << "SubEntity title: " << pEnt->getName() << endl;
            printSubEnts(pEnt, domDoc, indents+1);
            delete pEnt;
        }
        delete itEnts;
        delete pColl;
    }
    return;
}
```

キャビネット属性のリスト作成

キャビネットは、役立つ属性を含む唯一の項目です。ルームのエンティティ属性をリストしようとしても、コレクション内には何も表示されません。したがって、DKDatastoreDD による検索可能エンティティのリストは、キャビネットのみのリストです。

以下の例では、キャビネットおよびそれらの属性のリストを取得しています。

Java

```
...
dkCollection cabinets = dsDD.listSearchableEntities();
dkIterator itCabinets = cabinets.createIterator();
while( itCabinets.more() )
{
    // ----- For each cabinet, list it's attributes.
    dkEntityDef aCabinet = (dkEntityDef)itCabinets.next();
    cabinetName = aCabinet.getName();
    // ----- List Document Profiles without sub-attributes
    System.out.println("¥n" + Me + ": calling listAttrs for" + cabinetName );
    DKSequentialCollection coll=(DKSequentialCollection) aCabinet.listAttrs();
    ...
}
```

完全なサンプル・アプリケーション (TListAttributes.java) が
CMBROOT¥Samples¥java¥dd ディレクトリーに含まれており、この例はそこから
取られています。

Domino.Doc での照会の作成

照会対象を 1 つのキャビネットに制限したい場合、ENTITY= は、照会ストリング内の最初の語でなければなりません。ENTITY パラメーターおよびその値が欠落していると、ライブラリー全体が検索されます。また、値は引用符 (") で囲まなければなりません (例: "Diane Cabinet")。

QUERY= は必須パラメーターです。

Domino.Doc において照会ストリングは下記のようになります。

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

Domino.Doc コンテンツ・サーバーを照会するには、FTSearch 関数を使います。この関数が効率的に動作するためには、Domino.Doc コンテンツ・サーバーが完全にテキスト索引付けされている必要があります。索引をテストするには、IsFTIndexed プロパティを使用します。索引を作成するには、UpdateFTIndex 関数を使用します。

FTSearch 関数は、コンテンツ・サーバー内のすべての文書を検索します。特定のビュー内にある文書を検索するには、NotesView の FTSearch 関数を使います。特定の文書コレクション内にある文書を検索するには、NotesDocumentCollection の FTSearch 関数を使います。

ソート・オプションを指定しない場合、文書は関連によってソートされます。日付によってソートしたい場合、関連スコアとソート結果は受け取りません。結果の

DocumentCollection を NotesNewsletter インスタンスに渡す場合、使用するソート・オプションに応じて、文書作成日付または関連スコアのいずれかによって結果がソートされます。

照会構文の使用

照会の構文規則は、以下のリストにあります。括弧を使用して、優先順位を変更し、操作をグループ化します。

プレーン・テキスト

プレーン・テキストは、語または句そのものを検索するのに使います。検索キーワードおよび記号は、アポストロフィ (') で囲んでください。

LotusScript リテラル内では、引用符 (") を使用することを忘れないでください。

ワイルドカード

疑問符 (?) は、単語内の任意の位置の単一文字に相当するものを表すために使用します。アスタリスク (*) は、語の任意の位置の 0 ~ n (n は任意の数値) の文字と突き合わせるために使用します。

論理演算子

論理演算子は、検索を拡張または制限するために使用します。演算子とその優先順位は、下記のとおりです。

1. ! (not)
2. & (and)
3. , (accrue)
4. | (or)

これらのキーワードまたは記号のいずれかを使用することができます。

接近性演算子

接近性演算子は、相互に近接している単語を検索するために使用します。それらの演算子は、テキスト全体索引で、語、文、および段落の切れ目を必要とします。それらの演算子は以下のとおりです。

- near
- sentence
- paragraph

フィールド演算子

フィールド演算子は、指定したフィールドへの検索を制限するために使用します。その構文は FIELD *field-name operator* です。ここで *operator* は、テキストおよびリッチ・テキスト・フィールドの場合には CONTAINS で、数値および日付フィールドの場合には記号 =、>、>=、<、<= の 1 つです。

exactcase 演算子

exactcase 演算子は、続く式の検索を、指定した大文字小文字に制限するために使用します。

termweight 演算子

termweight n 演算子は、続く式の関連ランキングを調整するために使用します (n は 0 ~ 100)。

Extended Search (ES) の使用

Enterprise Information Portal は、IBM Extended Search 7.1 (ES 3 ではサポートされません) をサポートしています。Extended Search を使用すれば、以下から文書の照会および検索を行うことができます。

- Lotus Notes データベース。
- NotesPump データベース。
- ファイル・システム。
- Web 検索エンジン。

Enterprise Information Portal のクラスと API は、次の Extended Search 機能をサポートします。

- 1 つまたは複数の ES サーバーとの接続および切断。
- ES サーバーのリスト作成。
- データベースおよびフィールドのリスト作成。
- Generalized Query Language (GQL) を使用した検索の実行。
- 文書の取得。
- AIX における ES C++ クラスおよび API の使用。
- 統合レイヤーから、および直接 ES 接続を経由したパラメトリック・テキスト検索の実行。
- CONTAINS_TEXT および CONTAINS_TEXT_IN_CONTENT テキスト検索演算子の統合レイヤーからの使用。
- Enterprise Information Portal JavaBeans の使用。

制限事項: ES は、以下をサポートしていません。

- 文書の追加、更新、および削除。
- テキスト検索エンジンおよび QBIC の検索。
- 結合照会。
- ワークバスケットおよびワークフロー。

すべての ES 機能は、ES 構成データベースによってアクセスされ制御されます。この構成データベースを使用して、検索するデータ・ソース、ネットワーク・アドレス、アクセス制御情報、および他の関連情報についてのデータベース定義を割り当てます。

Extended Search サーバーのリスト作成

複数の ES サーバーへのアクセスを提供するために、cmbdes.ini というファイルを作成して、そこにサーバー情報を入れることができます。このファイルは、C:\CMBROOT (ここで、C はドライブ名) に保管します。cmbdes.ini ファイルは、下記の形式で、サーバーごとに 1 行が含まれていなければなりません。

```
DATASOURCE=TCP/IP address;PORT=port number
```

ここで、TCP/IP は ES サーバーの TCP/IP アドレス、port number はサーバーにアクセスするために定義されているポート番号 (例: PORT=80) です。

エンティティ (データベース) および属性 (フィールド) のリスト作成

ES サーバーを検索するための照会を作成するには、使用できるデータベース名とフィールド名を知っている必要があります。DKDatastoreDES オブジェクトは、データベースをリストする `listEntities` 関数と、各データベースのフィールドをリストする `listEntityAttrs` 関数を備えています。下記の例は、データベースとそのフィールドを取得する方法を示しています。

Java

```
try {
    DKSequentialCollection pCol = null;
    dkIterator pIter = null;
    DKSequentialCollection pCol2 = null;
    dkIterator pIter2 = null;
    String strDatabase = null;
    DKDatabaseDefDES dbDef = null;
    DKFieldDefDES attrDef = null;
    int i = 0;
    int j = 0;
    DKDatastoreDES dsDES = new DKDatastoreDES();
    System.out.println("connecting to datastore");
    dsDES.connect(libSrv,userid,pw,connect_string);
    System.out.println("datastore connected libSrv " + libSrv + " userid " +
        userid );
    System.out.println("list DES databases");
    pCol = (DKSequentialCollection) dsDES.listEntities();
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        dbDef = (DKDatabaseDefDES)pIter.next();
        strDatabase = dbDef.getName();
        System.out.println("database name [" + i + "] - " + strDatabase);
        System.out.println("list attributes for " + strDatabase + " database");
        pCol2 = (DKSequentialCollection) dsDES.listEntityAttrs(strDatabase);
        pIter2 = pCol2.createIterator();
        j = 0;
        while (pIter2.more() == true)
        {
            j++;
            attrDef = (DKFieldDefDES)pIter2.next();
            System.out.println("Attr name [" + j + "] - " + attrDef.getName());
            System.out.println("    datastoreType " + attrDef.datastoreType());
            System.out.println("    attributeOf " + attrDef.getEntityName());
            System.out.println("    type " + attrDef.getType());
            System.out.println("    size " + attrDef.getSize());
            System.out.println("    id " + attrDef.getId());
            System.out.println("    nullable " + attrDef.isNullable());
            System.out.println("    precision " + attrDef.getPrecision());
            System.out.println("    scale " + attrDef.getScale());
            System.out.println("    stringType " + attrDef.getStringType());
            System.out.println("    queryable " + attrDef.isQueryable());
            System.out.println("    displayName " + attrDef.getDisplayName());
            System.out.println("    helpText " + attrDef.getHelpText());
            System.out.println("    language " + attrDef.getLanguage());
            System.out.println("    valueCount " + attrDef.getNumVals());
        }
        System.out.println("    " + j + " attributes listed for
            " + strDatabase + " database");
    }
    System.out.println(i + " databases listed");
    dsDES.disconnect();
    System.out.println("datastore disconnected");
}
// continued...
```

Java (続き)

```
catch(DKException exc)
{
    System.out.println("Exception name " + exc.name());
    System.out.println("Exception message " + exc.getMessage());
    System.out.println("Exception error code " + exc.errorCode());
    System.out.println("Exception error state " + exc.errorState());
    exc.printStackTrace();
}
```

完全なサンプル・アプリケーション (TListCatalogDES.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```

...
cout << "list entities" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsDES.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pEnt = (DKDatabaseDefDES*)((void*)(*pIter->next()));
    strDBName = pEnt->getName();
    cout << "ndatabase name [" << i << "] - " << strDBName << endl;
    cout << "dispname: " << pEnt->getDisplayName() << endl;
    cout << "helptext: " << pEnt->getHelpText() << endl;
    cout << "lang: " << pEnt->getLanguage() << endl;
    int iVCount = pEnt->getNumVals();
    cout << "NumValus: " << iVCount << endl;
    cout << "datatype: " << pEnt->getDataType() << endl;
    cout << "searchable:" << pEnt->isSearchable() << endl;
    cout << "retrievable" << pEnt->isRetrievable() << endl;
    cout<<"\n  list attributes for "<<strDBName<<" database name"<< endl;
    pCol2 =
    (DKSequentialCollection*)((dkCollection*)dsDES.listEntityAttrs(strDBName));
    pIter2 = pCol2->createIterator();
    j = 0;
    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pAttr = (DKFieldDefDES*) pA->value();
        cout << "Attr name [" << j << "] - " << pAttr->getName() << endl;
        cout << "    datastoreName " << pAttr->datastoreName() << endl;
        cout << "    datastoreType " << pAttr->datastoreType() << endl;
        cout << "    attributeOf " << pAttr->getEntityName() << endl;
        cout << "    type " << pAttr->getType() << endl;
        cout << "    size " << pAttr->getSize() << endl;
        cout << "    id " << pAttr->getId() << endl;
        cout << "    nullable " << pAttr->isNullable() << endl;
        cout << "    precision " << pAttr->getPrecision() << endl;
        cout << "    scale " << pAttr->getScale() << endl;
        cout << "    string type " << pAttr->getStringType() << endl;
        cout << "    display name " << pAttr->getDisplayName() << endl;
        cout << "    help text " << pAttr->getHelpText() << endl;
        cout << "    language " << pAttr->getLanguage() << endl;
        cout << "    isQueryable " << pAttr->isQueryable() << endl;
        cout << "    isRetrievable " << pAttr->isRetrievable() << endl;
        delete pAttr;
    }
    cout << "  " << j << " attributes listed for "
           << strDBName << " database name" << endl;
    delete pIter2;
    delete pCol2;
    delete pEnt;
}
delete pIter;
delete pCol;
cout << i << " entities listed\n" << endl;
...

```

完全なサンプル・アプリケーション (TListCatalogDES.cpp) が
 Cmbroot/Samples/cpp/ES ディレクトリーに含まれており、この例はそこから
 取られています。

Generalized Query Language (GQL) の使用

Extended Search は Generalized Query Language (GQL) を使用して検索を実行します。表 20 は、有効な GQL 式の例を示しています。

表 20. GQL 式

GQL 式	説明
"software"	語 software を含む文書を検索します。
(TOKEN:WILD "exec*")	exec で始まる語を含む文書を検索します。
(AND "software" "IBM")	software と IBM の両方の語を含む文書を検索します。
(START "View" "How")	「表示 (View)」フィールドが How という単語で始まっている文書を検索します。
(EQ "View" "How Do I?")	「表示 (View)」フィールドに、How Do I? というストリングがそのとおり含まれている文書を検索します。
(GT "BIRTHDATE" "19330804")	「生年月日 (BIRTHDATE)」フィールドの値が、1933 年 8 月 4 日より大きい文書を検索します。

ES は照会タイプ DK_DES_GQL_QL_TYPE を使用します。この照会タイプの構文は以下のとおりです。

```
SEARCH=(DATABASE=(db_name | db_name_list | ALL);
          COND=(GQL_expression));
[OPTION=( [SEARCHABLE_FIELD=(fd_name, ...);]
          [RETRIEVABLE_FIELD=(fd_name, ...);]
          [MAX_RESULTS=maximum_results;]
          [TIME_LIMIT=time])]
```

ここで、db_name_list はコンマで区切られているデータベース名 (db_name) のリストで、ALL は、使用可能なすべてのデータベースを検索するということです。デフォルトの検索制限時間は 30 秒です。

以下の例では照会ストリングを使用して、Notes Help データベースを検索し、「表示 (View)」フィールドが How Do I? である文書を探します。さらに、結果は最大で 5 つまでが戻されます。

```
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
             "COND=(EQ ¥"View¥" ¥"How Do I?¥"));" +
             "OPTION=(MAX_RESULTS=5)"
```

この例は、ES に対して GQL 照会を実行します。照会が実行されると、結果は dkResultSetCursor オブジェクトに戻されます。

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
System.out.println("connecting to datastore");
dsDES.connect(libSrv,userid,pw,connect_string);
System.out.println("datastore connected libSrv "+libSrv+" userid "+userid);
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
             "COND=(EQ ¥"View¥" ¥"How Do I?¥"));";
             "OPTION=(MAX_RESULTS=5)";
DKDDO ddo = null;
System.out.println("query string " + cmd);
System.out.println("executing query");
pCur = dsDES.execute(cmd,DK_DES_GQL_QL_TYPE,parms);

System.out.println("cardinality " + pCur.cardinality());
System.out.println("datastore executed query");
System.out.println("process query results");
...
pCur.destroy(); // Finished with the cursor
System.out.println("query results processed");
dsDES.disconnect();
System.out.println("datastore disconnected");
```

完全なサンプル・アプリケーション (TExecuteDES.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;

DKString cmd = "SEARCH=(DATABASE=(Notes Help));";
cmd += "COND=((IN ¥"Subject¥" ¥"your¥"));";
cmd += "OPTION=(MAX_RESULTS=2;TIME_LIMIT=10));";

cout << "query string " << cmd << endl;
cout << "executing query" << endl;
pCur = dsDES.execute(cmd);
cout << "query executed" << endl;
...
```

完全なサンプル・アプリケーション (TExecuteDES.cpp) が
Cmbroot/Samples/cpp/ES ディレクトリーに含まれており、この例はそこから
取られています。

Extended Search での DDO 項目タイプの識別

ES の DDO のタイプは常に DK_CM_DOCUMENT です。DDO の項目タイプを取得する
場合、以下を呼び出します。

Java

```
Object obj = ddo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
short type = ((Short) obj).shortValue();
```

C++

```
DKDDO *p = 0;
ushort k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (k > 0)
{
    DKAny a = p->getProperty(k);
    ushort val = a; // val = DK_CM_DOCUMENT
}
```

Extended Search での PID の作成

永続 ID (PID) には、文書に関する特定の情報が含まれています。オブジェクト・タイプは、その文書が見つかったデータベースを示しています。PID は、データベース名の後に | 文字と文書 ID を付けて作成されます。以下に例を示します。

database name|documentId ()

PID の詳細については、17 ページの『永続 ID (PID) の説明』および 50 ページの『永続 ID (PID) の作成』を参照してください。

Extended Search 文書の内容の処理

DDO 内の各項目は、フィールド、コレクション、または DKParts オブジェクトのいずれかです。

フィールド

単一フィールドのフィールド名は、項目名に含まれています。また、フィールドの値も項目の値に含まれています。フィールドのプロパティは下記のとおりです。

- DK_CM_VSTRING
- DK_CM_FLOAT
- DK_CM_XDOOBJECT
- DK_CM_DATE
- DK_CM_SHORT

コレクション

フィールドに複数の値が含まれている場合、フィールド名は項目名の中にあります。項目値は DKSequentialCollection オブジェクトです。フィールドが BLOB の場合、プロパティには DK_CM_COLLECTION または DK_CM_COLLECTION_XDO を指定できます。

DKParts

文書 DDO には、予約名 DKPARTS が使用されている固有の属性があります。この値は DKParts オブジェクトです。DKPARTS オブジェクトには、

文書に関する Web アドレス (URL) 情報を格納できます。DKPARTS には、文書の URL を表すストリングとして、そのコンテンツと共に XDO を含めることもできます。

以下の例では、DDO の内容を処理します。

Java

```
public static void displayDDO(DKDDO ddo)
    throws DKException, Exception
{
    dkXDO pXDO = null;
    int i = 0;
    int numDataItems = 0;
    short k = 0;
    short j = 0;
    Integer valueCount = null;
    Object value = null;
    String dataName = null;
    dkCollection pCol = null;
    dkIterator pIter = null;
    Object anObject = null;
    numDataItems = ddo.dataCount();
    DKPid pid = ddo.getPidObject();
    System.out.println("pid string " + pid.pidString());

    System.out.println("Number of Attributes " + numDataItems);
    for (j = 1; j <= numDataItems; j++)
    {
        anObject = ddo.getData(j);
        dataName = ddo.getDataName(j);
        System.out.println(j + ": Name " + dataName );
        // determine if data item has a single value or multiple values
        Short type = (Short)ddo.getDataPropertyByName(j, DK_CM_PROPERTY_TYPE);
        k = type.shortValue();
        if (k == DK_CM_COLLECTION)
        {
            {
                pCol = (dkCollection)anObject;
                pIter = pCol.createIterator();
                i = 0;
                while (pIter.more() == true)
                {
                    i++;
                    value = pIter.next();
                    System.out.println("    Value" + i + " " + value);
                }
            }
        }
        else if (k == DK_CM_COLLECTION_XDO)
        {
            {
                pCol = (dkCollection)anObject;
                pIter = pCol.createIterator();
                i = 0;
                while (pIter.more() == true)
                {
                    i++;
                    blob = (DKBlobDES)pIter.next();
                    System.out.println("    Value" + i + " " + blob.getContent());
                }
            }
        }
        else
        {
            System.out.println("    Value " + anObject);
        }
    }
}
```

完全なサンプル・アプリケーション (TExecuteDES.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDDO *p = 0;
dkDataObjectBase *pDOBase = 0;
DKDDO *pDDO = 0;
dkXDO *pXDO = 0;
DKAny a;
ushort j = 0;
ushort k = 0;
ushort val = 0;
ushort cnt = 1;
DKString strData = "";
DKString strDataName = "";
dkCollection* pdCol = 0;
dkIterator* pdIter = 0;
ushort numDataItems = 0;
DKPidXDODES *pidXDO = 0;
DKPid *pid = 0;
DKString strPid;
long pidIdCnt = 0;
long pidIndex = 0;
while (pCur->isValid())
{
    p = pCur->fetchNext();
    if (p != 0)
    {
        cout << "=====> " << "Item " << cnt << " <=====" << endl;
        numDataItems = p->dataCount();
        pid = (DKPid*)p->getPidObject();
        strPid = pid->pidString();
        cout << "pid string " << strPid << endl;
        cout << "pid id string " << pid->getId() << endl;
        strPid = pid->getIdString();
        cout << "pid idString " << strPid << endl;
        pidIdCnt = pid->getIdStringCount();
        cout << "pid idString cnt " << pidIdCnt << endl;
        strPid = pid->getPrimaryId();
        cout << "pid primary id " << strPid << endl;
        pidIndex = 0;
        strPid = pid->getIdString(pidIndex);
        cout << "pid item id " << strPid << endl;
        k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
        if (k > 0)
        {
            a = p->getProperty(k);
            val = a;
            cout << "*****" << endl;
            switch (val)
            {
                case DK_CM_DOCUMENT :
                    cout << "Item is a document " << endl;
                    break;
                default:
                    cout << " Item is not recognized " << endl;
                    break;
            }
            cout << "*****" << endl;
        }
        cout << "Number of Data Items " << numDataItems << endl;
        for (j = 1; j <= numDataItems; j++)
        {
            a = p->getData(j);
            strDataName = p->getDataName(j);
        }
    }
    // continued...
```

C++ (続き)

```
        switch (a.typeCode())
        {
        case DKAny::tc_string :
        {
            strData = a;
            cout << "attribute name : " << strDataName << " value : "
                  << strData << endl;
        }
            break;
        case DKAny::tc_long :
        {
            long l = a;
            cout << "attribute name : " << strDataName << " value : " << l << endl;
        }
            break;
        case DKAny::tc_double :
        {
            double db = a;
            cout << "attribute name : " << strDataName << " value : " << db << endl;
        }
            break;
        case DKAny::tc_timestamp :
        {
            DKTimestamp tt = a;
            cout << "attribute name : " << strDataName << " value : "
            << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
            << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
        }
            break;
        case DKAny::tc_dobase :
        {
            pDOBase = a;
            pXDO = (dkXDO*)pDOBase;
            cout << "attribute name : " << strDataName << " value : " << endl;
            pidXDO = (DKPidXDOES*)pXDO->getPid();
            cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
            cout << "XDO pid docId " << pidXDO->getDocId() << endl;
            cout << "XDO mimetype " << pXDO->getMimeType() << endl;
            ((DKBlobDES*)pXDO)->getContentToClientFile("c:¥¥temp¥¥temp.html", 1);
        }
            break;
        case DKAny::tc_collection :
        {
            pdCol = a;
            cout << strDataName << " collection name : " << strDataName << endl;
            cout << "-----" << endl;
            pdIter = pdCol->createIterator();
            ushort b = 0;
            while (pdIter->more() == TRUE)
            {
                b++;
                cout << " -----" << endl;
                a = *(pdIter->next());
                switch (a.typeCode())
                {
                case DKAny::tc_string :
                {
                    strData = a;
                    cout << "attribute name : " << strDataName << " value : " << strData << endl;
                }
            }
                break;
            }
        }
    }
    // continued...
```

C++ (続き)

```
        case DKAny::tc_long :
        {
            long l = a;
            cout << "attribute name : " << strDataName << " value : " << l << endl;
        }
        break;
        case DKAny::tc_double :
        {
            double db = a;
            cout << "attribute name : " << strDataName << " value : " << db << endl;
        }
        break;
        case DKAny::tc_timestamp :
        {
            DKTimestamp tt = a;
            cout << "attribute name : " << strDataName << " value : "
            << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
            << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
        }
        break;
        case DKAny::tc_dobase :
        {
            pDOBase = a;
            pXDO = (dkXDO*)pDOBase;
            cout << "attribute name : " << strDataName << " value : " << endl;
            pidXDO = (DKPidXDOES*)pXDO->getPid();
            cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
            cout << "XDO pid docId " << pidXDO->getDocId() << endl;
            cout << "XDO mimetype " << pXDO->getMimeType() << endl;
            DKString str = "c:¥¥temp¥¥temp";
            DKString strT = b;
            str = str + strT + ".html";
            ((DKBlobDES*)pXDO)->getContentToClientFile(str, 1);
        }
        break;
    }
    ushort usCount = p->dataPropertyCount(j);
    for (ushort k = 1; k <= usCount; k++)
    {
        a = p->getDataProperty(j, k);
        cout << " property " << k << " " << a << endl;
    }
    }
    if (b == 0)
    {
        cout << strDataName << " collection has no elements " << endl;
    }
    cout << " -----" << endl;
    break;
}
}
ushort usCount = p->dataPropertyCount(j);
for (ushort k = 1; k <= usCount; k++)
{
    a = p->getDataProperty(j, k);
    cout << " property " << k << " " << a << endl;
}
}
cnt++;
delete p;
```

完全なサンプル・アプリケーション (TExecuteDES.cpp) が
Cmbroot/Samples/cpp/ES ディレクトリーに含まれており、この例はそこから
取られています。

文書の検索

DKDatastoreDES オブジェクトから文書を取得するには、文書があるデータベースの
名前、およびその文書 ID を知っている必要があります。また、その DDO をコン
テナツ・サーバーに関連付け、接続を確立しなければなりません。下記の例では、
文書を取得しています。

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKPid pid = new DKPid();
// ----- Primary ID is 'database name' followed by the
//           character '|' followed by the document ID
pid.setPrimaryId("Notes Help" + DK_DES_ITEMID_SEPARATOR + "215e");
pid.setObjectType("Notes Help");
DKDDO ddo = new DKDDO(dsDES, pid);
ddo.retrieve();
```

完全なサンプル・アプリケーション (TRetrieveDDODES.java) が
CMBROOT¥Samples¥java¥d1 ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
p = new DKDDO(&dsDES, "");
DKPid pid2;
pid2.setDatastoreType(dsDES.datastoreType());
pid2.setDatastoreName(dsDES.datastoreName());
pid2.setId("Notes Help|215e");
pid2.setObjectType("");
p->setPidObject((DKPid*)&pid2);
p->retrieve();
...
```

完全なサンプル・アプリケーション (TRetrieveDDODES.cpp) が
Cmbroot/Samples/cpp/ES ディレクトリーに含まれており、この例はそこから
取られています。

バイナリー・ラージ・オブジェクト (BLOB) の検索

DKDatastoreDES オブジェクトから BLOB を検索する場合、文書があるデータベースの名前、BLOB を含んでいる文書 ID、および BLOB を含んでいるフィールド名を知っている必要があります。また、その DDO をコンテンツ・サーバーに関連付け、接続を確立しなければなりません。

以下の例では、ES files というファイル・システム・データベースに、
D:¥desdoc¥README.html という HTML ファイルがあります。その HTML ファイルが含まれているフィールドの名前は Doc\$Content です。次の例では、その HTML ファイルを検索し、それを D:¥DESReadme.html という名前を保管します。

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKBlobDES xdo = new DKBlobDES(dsDES);
DKPidXDODES pid = new DKPidXDODES();
pid.setDocumentId("D:¥¥desdoc¥¥README.html");
pid.setDatabaseName("DES files");
pid.setFieldName("Doc$Content");
xdo.setPidObject(pid);
xdo.retrieve("c:¥¥DESReadme.html");
```

完全なサンプル・アプリケーション (TRetrieveXDODES.java) が
CMBROOT¥Samples¥java¥dl ディレクトリーに含まれており、この例はそこから
取られています。

C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
    cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
cout << "executing retrieve a XDO" << endl;

DKBlobDES* p = new DKBlobDES(&dsDES);
DKPidXDODES pid;
pid.setDocId("D:¥¥desdoc¥¥README.html");
pid.setDatabaseName("ES files");
pid.setFieldName("Doc$Content");
pid.setPrimaryId("ES files|D:¥¥desdoc¥¥README.html");
p->setPidObject((DKPidXD0*)&pid);

p->retrieve("c:¥¥temp¥¥DESReadme.html");

cout << "retrieve executed" << endl;
...
```

完全なサンプル・アプリケーション (TRetrieveXDODES.cpp) が
Cmbroot/Samples/cpp/ES ディレクトリーに含まれており、この例はそこから
取られています。

MIME タイプと文書との関連付け

ES は MIME タイプの識別を直接サポートしてはいません。しかし、Web ブラウザで表示したい XDO の MIME タイプは知っておく必要があります。

文書の MIME タイプを判別する場合、CMBCC2MIME.INI ファイルを使用します。
NotesPump または FileSystem データベースからの ES 照会が BLOB を戻す場合、
CMBCC2MIME.INI ファイルが検索され、MIME タイプをその BLOB に割り当てる

ことが可能かどうか判別されます。デフォルトの MIME タイプは text/html です。 cmbcc2mime.ini.samp というサンプル・ファイルが samples ディレクトリーに入っています。

Extended Search での統合検索の使用

統合照会を作成する場合、ES で使う構文は SQL 構文で使うものによく似ています。統合照会は、ES に実行依頼される前に GQL 構文に変換されます。しかし SQL と GQL の文法には相違点があるため、Enterprise Information Portal がサポートしているのは、SQL 文法のサブセットだけです。

表 21 は、サポートされている比較および論理演算子の、SQL から GQL への変換を要約したものです。

表 21. SQL および GQL 演算子

SQL 演算子	GQL 演算子
AND	AND
OR	OR
NOT	サポートされていません。
IN	サポートされていません。
BETWEEN	BETWEEN
EQ	EQ
NEQ	サポートされていません。
GT	GT
LT	LT
LIKE	サポートされていません。
GEQ	GTE
LEQ	LTE
NOTLIKE	サポートされていません。
NOTIN	サポートされていません。
NOTBETWEEN	サポートされていません。

Panagon Image Services の使用 (Java のみ)

Panagon Image Services (FileNET) 用に提供される Enterprise Information Portal API クラスを使用して、Panagon Image Services サーバーのコンテンツにアクセスすることができます。Panagon Image Services のサポートは、EIP の特殊機能としてのみ利用することができます。

制限事項: Panagon Image Services は、テキスト検索エンジンと QBIC 検索または結合照会をサポートしていません。

データ・モデル

Enterprise Information Portal にはいくつかのデータ・モデリングの概念があり、これはコネクターがサポートするコンテンツ・サーバーの対応するオブジェクトにマップする必要があります。以下の表に、FileNET 環境における EIP データ・モデルを示します。

EIP の概念	FileNET
コンテンツ・サーバー	Panagon Image Services サーバー
サーバー	サーバー
エンティティ	文書クラス
属性	索引
DDO	文書 (フォルダー) FileNET のフォルダーは、文書の一時的な編成に使用されます。これは現在、コネクターにはモデル化されていません。
XDO	ページ・コンテンツ、注釈
DDO の PID	document_id、プライマリー ID として
ページ・コンテンツ XDO の PID	System_serial_num + document_id + page#
注釈 XDO の PID	System_serial_num + document_id + page# + annotation_id

FileNET コネクターは、FileNET のユーザー定義索引または属性のみを戻します。これは、統合エンティティと属性のマッピング、および検索結果の属性表示リストでを使用することができます。以下の表に、FileNET でサポートされる索引または属性データ・タイプの、FileNET コネクターで対応する EIP 属性データ・タイプへのマッピングを示します。

FileNET 索引データ・タイプ	EIP 属性データ・タイプ
数値	double
日付	varchar (max_length は FileNET 索引に定義された最大長に設定)
メニュー	varchar (max_length = 14 (メニュー名の最大長))

EIP の XDO はすべて MIME タイプを取ります。EIP クライアントは MIME タイプを使用して文書の表示方法を決定します。FileNET コネクターは、同じ文書のすべての XDO は同じ MIME タイプであると想定します。XDO の MIME タイプは、それを含む文書の F_DOCFORMAT FileNET システム属性値に応じて設定されます。値がヌルの場合、MIME タイプは image/tiff に設定されます (F_DOCTYPE FileNET システム属性がイメージ・タイプを指定している場合。指定していなければデフォルトの MIME タイプ application/octet-stream に設定される)。

Panagon Image Services の文書およびページ

DDO を使用して Panagon Image Services の文書を表すには、DDO の PID の次の 4 つのフィールドを正しく設定する必要があります。

1. DatastoreType - DKConstantFN で定義された定数 DK_FN_DSTYPE。この値は、DKDatastoreFN の datastoreType() メソッドから取得できます。
2. DatastoreName - FileNET のドメイン編成名。この値は、DKDatastoreFN の datastoreName() メソッドから取得できます。
3. ObjectType - 文書が属す FileNET 文書クラス (EIP のネイティブ・エンティティ名) 名。
4. PrimaryID - FileNET により割り当てられた FileNET 文書番号。

PID が設定されると、アプリケーション・プログラムは DKDatastoreFN または DKDDO クラスの retrieve() メソッドを呼び出して DDO の属性またはコンテンツを取得することができます。

EIP の文書 DDO には、予約名 DKPARTS を持つ特殊属性があります (その値は DKParts オブジェクトです)。DKParts オブジェクトは、バイナリー・ラージ・オブジェクト (BLOB) のコレクションです。文書内の FileNET パーツ (ページ/注釈) は、DKParts コレクションの元素である DKBlobFN オブジェクト (DKXDO のサブクラス) として表されます。

FileNET ページは従来型のページではありません。FileNET の単一ページ (または単一パーツ) 文書は複数の物理ページを含むことができますが、これは FileNET 文書内の単一ページに保管されます。FileNet の複数ページ文書は、Panagon Capture Software またはバッチ入力システムを使用して作成された複数パーツまたは複数ファイルのいずれかを含む文書を参照します。

エンティティおよび属性のリスト作成

Panagon Image Services サーバーは、DKDatastoreFN として表現します。コンテンツ・サーバーを作成してそれに接続したら、Panagon Image Services サーバー用のエンティティ (文書 x クラス) および属性をリストできます。以下の例は、この作業について示しています。

Java

```
DKDatastoreFN dsFN = null;
try {
    DKSequentialCollection pCol = null;
    dkIterator pIter = null;
    DKSequentialCollection pCol2 = null;
    dkIterator pIter2 = null;
    DKServerDefFN pSV = null;
    String strServerName = null;
    String strServerType = null;
    String strEntity = null;
    DKEntityDefFN entityDef = null;
    DKAttrDefFN attrDef = null;
    int i = 0;
    int j = 0;
    dsFN = new DKDatastoreFN();
// ----- Connect to the server using the server name, user ID , and password
dsFN.connect(libSrv, userid, pw, "");
System.out.println("Datastore connected libSrv " + libSrv + " userid "
    + userid );
System.out.println("List entities in server " + libSrv);
pCol = (DKSequentialCollection) dsFN.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    entityDef = (DKEntityDefFN)pIter.next();
    strEntity = entityDef.getName();
    System.out.println("\nEntity name [" + i + "] - " + strEntity +
        ", id = " + entityDef.getId() + ", type = " + entityDef.getType());
    System.out.println(" List attr for the " + strEntity + " entity");
    pCol2 = (DKSequentialCollection) dsFN.listEntityAttrs(strEntity);
    pIter2 = pCol2.createIterator();
    j = 0;
    while (pIter2.more() == true)
    {
        j++;
        attrDef = (DKAttrDefFN)pIter2.next();
        System.out.println(" Attribute name [" + j + "] - " +
            attrDef.getName());
        System.out.println(" datastoreType = " + attrDef.datastoreType());
        System.out.println(" attributeOf = " + attrDef.getEntityName());
        System.out.println(" type = " + attrDef.getType());
        System.out.println(" size = " + attrDef.getSize());
        System.out.println(" id = " + attrDef.getId());
        System.out.println(" nullable = " + attrDef.isNullable());
        System.out.println(" precision = " + attrDef.getPrecision());
        System.out.println(" scale = " + attrDef.getScale());
        System.out.println(" stringType = " + attrDef.getStringType());
    }
    System.out.println(" Total of " + j + " attributes listed for the "
        + strEntity + " entity");
}
System.out.println("\nTotal of " + i + " entities listed for server "
    + libSrv);
// ----- Disconnect and clean up
dsFN.disconnect();
dsFN.destroy();
}
catch(DKException exc)
```

サンプル・アプリケーションについては、CMBROOT¥Samples¥java¥fn ディレクトリーを参照してください。

照会

EIP では、Panagon Image Services について次の 3 つの基本形式の照会をサポートします。

- 照会オブジェクト
- コマンド・ストリング

- DKCQExpr

照会オブジェクトおよびコマンド・ストリングの両形式の照会は、通常、Panagon Image Services サーバーに直接アクセスするアプリケーションで使用されます。EIP 統合コンテンツ・サーバーおよび照会では、個々のコンテンツ・サーバーとのインターフェースに DKCQExpr 形式の照会のみを使用します。

Panagon Image Services サーバーでの照会に関するサンプル・アプリケーションが、CMBROOT¥Samples¥java¥fn ディレクトリーにあります。

照会ストリングの構文およびパラメーター

FileNET コネクターは、DKParametricQuery を使用した照会の実行依頼および実行をサポートします。DKParametricQuery は、コマンド・ストリングによる構成が可能です。コマンド・ストリングは、FileNET フィルター条件の照会仕様を表します。その他の照会オプション (ENTITY_NAME、MAX_RESULTS、CONTENT) および FileNET キー条件 (KEY) は、evaluate() または execute() メソッドに対する (名前、値) ペア (例、DKNVPair) でのパラメーターとして使用できます

値は常にストリング・タイプであり、パラメーター名は DKConstant および DKConstantFN に定義されることに注意してください。戻される結果は、evaluate() の文書 DDO のコレクションおよび、execute() の結果をポイントする結果セットのカーソルです。

コマンドおよび KEY パラメーターの条件式構文は、FileNET WAL PRS_Parse() 関数によりサポートされる FileNET 照会構文に従います。以下の演算子を使用できます。

演算子	使用
AND	論理 AND
OR	論理 OR
NOT	論理否定
<	より小さい
<=	以下
=	等しい
>	より大きい
>=	以上
!=	等しくない
+	加算
-	減算
*	乗算
/	除算
IN RANGE	範囲指定の構文: IN RANGE <i>op constant op constant</i> (<i>op</i> は <, <=, >, または >= で <i>constant</i> は数値またはストリング定数)
LIKE	類似。LIKE 演算子の右辺のパターンは単一引用符文字ストリングとし、これにはワイルドカード文字として ? (すべての文字に一致)、ワイルドカードとして * を含むことができます。

演算子	使用
DEFINED(<i>x</i>)	定義された関数は、所定のカラム名 <i>x</i> が非ヌルの場合に非ゼロを、値がヌルの場合にゼロを返します。

オペランドは、数値定数、ストリング定数、および属性名です。コマンド・ストリングは、FileNET コンテンツ・サーバーを直接検索するのに使用されるため、属性名はすべてローカル FileNET 属性/索引名となります。統合属性名およびスキーマ・マッピングはサポートされず、必要とされません。

数値は `+ddd.dddE+eee` 形式であり、*ddd* は 0 以上の数値、`+` は `+` または `-` (`+` はオプション)、*eee* は 0 ~ 3 桁の指数、`.` は小数点 (オプション)、そして *E* は指数の開始 (オプション) を表します。

ストリング定数は単一引用符で囲み、ストリングに組み込まれた単一引用符は単一引用符を 2 つ続けて表します。

コマンド・ストリングの条件式には、オペランドおよび演算子をいくつでも含むことができます。フィルター内に括弧を使用して優先順位をオーバーライドすることもできます。フィルター条件の例を以下に示します。

```
PROD_PRICE > 100 AND (PROD_DATE > '10/15/92' OR PROD_NAME LIKE 'comp*')
PROD_PRICE >= 100 AND (NOT (PROD_ID < 30))
```

KEY パラメーターの条件式に含むことのできる属性名 (FileNET の検索キーとして定義) および条件は 1 つだけです。FileNET に定義したキー名を使用する必要があり、反転していないカラム名は使用できないことに注意してください。キー・ストリングの例を以下に示します。

```
PROD_PRICE = 100000
PROD_PRICE IN RANGE >= 100000 <= 200000
```

追加の検索オプション

`evaluate()` および `execute()` でサポートされるその他のオプション・パラメーターは以下のとおりです。

(ENTITY_NAME, entity_Name):

照会の有効範囲としエンティティ名 (FileNET 文書のクラス名) を指定します。例:

```
parms[1] = new DKNVPair (DK_FN_PARM_ENTITY_NAME, entity_Name);
```

Panagon Image Services サーバーに実行依頼された実際の照会ストリングに `AND (F_DOCCLASSNUMBER = doc_cls#)` が追加されます。`= doc_cls#` は、指定されたエンティティ名の一致する文書クラス番号です。指定しない場合は、すべての FileNET 文書クラスの文書が検索されます。

(MAX_RESULTS, Max_results)

結果コレクションには、指定された最大数の結果のみ戻されます。値がゼロまたは指定されない場合は、FileNET で許可された最大数の結果が戻されます。例:

```
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, Max_results);
```


(CONTENT, YES / NO / ATTRONLY)

このパラメーターは、コンテンツ・サーバー・レベルで設定された DK_CM_OPT_CONTENT オプションを上書きします。

- YES - 結果の文書のコンテンツが検索される (デフォルト)
- NO - 結果の DDO に文書 ID のみが設定される
- ATTRONLY - 結果の DDO に文書 ID、データ属性およびその値が設定される

例:

```
dsFN.setOption(DK_CM_OPT_CONTENT, DK_CM_CONTENT_YES);
```

例えば、コンテンツ値 YES を指定した場合、結果の文書 DDO には、その PID、オブジェクト・タイプ、プロパティおよびすべての属性が設定されます。また、このオプションでは DKPARTS 属性がコンテンツ・パーツのコレクション (パーツとその注釈の XDO) に設定され、PID 情報が設定されますが、実際のコンテンツは設定されません。CONTENT 値が ATTRONLY の場合、結果の文書 DDO PID、オブジェクト・タイプ、プロパティ、およびすべてのデータ属性が設定されます。CONTENT 値が NO の場合は、結果の文書 DDO PID、オブジェクト・タイプ、およびプロパティが設定されます。文書パーツまたは XDO コンテンツは、必要に応じて DKBlobFN クラスの retrieve() メソッドを使用して明示的に検索することができます。

例外処理およびメッセージ

Panagon Image Services コネクタは、EIP で定義されたものと同じ Java 例外クラスのセットを使用します。例外に組み込まれたテキスト・メッセージも、使用可能な場合は DKMessageID に共通に定義されたメッセージのいずれかを使用します。

例えば、DKUsageError 例外はメッセージを

```
(DKMessage.getMessage(DK_CM_MSG_NOTIMP) + this.getClass().getName() + XXXX, DK_CM_MSG_NOTIMP) として組み込みます。ここで、XXX はプログラムが呼び出そうとするメソッドの名前です。
```

DKDatastoreAccessError 例外は、次の形式で FileNET 固有のメッセージ・テキストを組み込みます。

```
WAL_function_name [IMS_SESSION=sss, ERR_CAT=ccc, ERR_FUN=fff, ERR_NUM=nnn]
```

FileNET コマンド msg (c:%fnsfw%client%bin ディレクトリーにある) を使用すると、さらにネイティブの FileNET エラー・メッセージのテキストを表示することができます。この場合、3 つの数 ccc、fff、および nnn を使用します (例、msg ccc,fff,nnn)。

Panagon Image Services サーバー用に一意的に使用される EIP メッセージ ID は 3401 ~ 3600 となります。

リレーショナル・データベースの使用

Enterprise Information Portal API クラスは、IBM DB2 Universal Database をサポートし、また Java Database Connectivity (JDBC)、Open Database Connectivity (ODBC) (C++ の場合)、または IBM DB2 DataJoiner を使用するその他のリレーショナル・データベースをサポートします。

リレーショナル・データベースへの接続

リレーショナル・データベースを表すには、DKDatastorexx オブジェクトを作成します (xx は、DB2 データベースの場合は DB2、Java Database Connectivity の場合は JDBC、Open Database Connectivity の場合は ODBC)。以下のサンプルでは、DB2 サンプル・データベースに接続します。

Java

```
dsDB2 = new DKDatastoreDB2();
dsDB2.connect("sample", "db2admin", "password", "");
.....
dsDB2.disconnect();
dsDB2.destroy();
```

C++

```
try {      DKDatastoreDB2 dsDB2;
    dsDB2.connect("sample", userid, pw);
    . . .
    dsDB2.disconnect();
}
catch(DKException &exc) . . .
```

実際の接続時には、データベース名を使用してください。

接続ストリング

リレーショナル・データベースに接続する場合、接続ストリングを指定してそれをパラメーターとして渡すことができます。複数の接続ストリングを指定する場合は、それぞれをセミコロン (;) で区切ってください。 接続ストリングの形式は以下のとおりです。

DB2、DataJoiner、および ODBC 用の接続ストリング:

NATIVECONNECTSTRING=(*native connect string*)

接続時にデータベースへ渡すネイティブ接続ストリングを指定します。有効なネイティブ接続ストリングについては、コンテンツ・サーバーに関する情報を調べてください。

SCHEMA=*schema name*

listEntities、listEntityAttrs、listPrimaryKeyNames、listForeignKeyNames メソッドの実行時に使用されるデータベース・スキーマ名を指定します。

JDBC 用の接続ストリング:

SCHEMA=*schema name*

listEntities、listEntityAttrs、listPrimaryKeyNames、listForeignKeyNames メソッドの実行時に使用されるデータベース・スキーマ名を指定します。

構成ストリング

構成ストリングを指定し、それをパラメーターとして接続メソッドに渡すこともできます。複数の構成ストリングを指定する場合は、それぞれをセミコロン (;) で区切ってください。構成ストリングの形式は、以下のとおりです。

DB2、DataJoiner、および ODBC 用の構成ストリング:

CC2MIMEURL=(URL)

URL アドレスとして、cmbcc2mime.ini ファイルを指定します。ファイルの場所に応じて、この形式の構成ストリングか CC2MIMEFILE を使用してください。

CC2MIMEFILE=(filename)

cmbcc2mime.ini ファイルを名前指定します。

DSNAME=(content server name)

コンテンツ・サーバーの名前を指定します。統合照会および他の統合機能の場合、Enterprise Information Portal が自動的にこの名前を設定します。

AUTOCOMMIT=ON | OFF

自動コミットをオンまたはオフに設定します。デフォルトはオフです。統合照会および他の統合機能でこのコンテンツ・サーバーを使用する場合、自動コミットはデフォルトでオンになります。

JDBC 用の構成ストリング:

CC2MIMEURL=(URL)

URL アドレスとして、cmbcc2mime.ini ファイルを指定します。ファイルの場所に応じて、この形式の構成ストリングか CC2MIMEFILE を使用してください。

cmbcc2mime.ini ファイルを名前指定します。

JDBCSEVERURL=(URL)

URL アドレスに cmbjdbcsvrs.ini ファイルを指定します。このファイルには JDBC サーバーのリストが含まれています。

JDBCSEVERFILE=(filename)

ファイル名として JDBC サーバーのリストを含む cmbjdbcsvrs.ini ファイルを指定します。

JBCDRIVER=(JDBC driver)

使用する JDBC ドライバーを指定します。これは、システム管理クライアント・プログラムの使用時には自動的に設定されます。

DSNAME=(content server name)

コンテンツ・サーバーの名前を指定します。統合照会および他の統合機能の場合、Enterprise Information Portal が自動的にこの名前を設定します。

AUTOCOMMIT=ON | OFF

自動コミットをオンまたはオフに設定します。デフォルトはオフです。統合照会および他の統合機能でこのコンテンツ・サーバーを使用する場合、自動コミットはデフォルトでオンになります。

エンティティおよびエンティティ属性のリスト

リレーショナル・データベース用にコンテンツ・サーバーを作成し、それに接続した後、エンティティおよびエンティティ属性のリストを取得することができます。次の例では、リストを取得してステップごとに処理する方法を示しています。

Java

```
// ----- After creating a datastore and connecting, get index classes
pCol = (DKSequentialCollection)dsDB2.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
    i++;
    pSV = (DKServerDefDB2)pIter.next();
    strServerName = pSV.getName();
    .... // Use the server name as appropriate
}
// ----- Connect to datastore
dsDB2.connect(db, userid, pw, "");
if (!schema.equals(""))
{
    dsDefDB2 = (DKDatastoreDefDB2)dsDB2.datastoreDef();
    dsDefDB2.setSchemaName(schema);
    schema = dsDefDB2.getSchemaName();
    System.out.println(" New Schema Name = [" + schema + "]");
}
// ----- List the tables
pCol = (DKSequentialCollection) dsDB2.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    tableDef = (DKTableDefDB2)pIter.next();
    strTable = tableDef.getName();
    // ----- List attributes (columns for the table)
    pCol2 = (DKSequentialCollection) dsDB2.listEntityAttrs(strTable);
    pIter2 = pCol2.createIterator();
    j = 0;
    while (pIter2.more() == true)
    {
        j++;
        colDef = (DKColumnDefDB2)pIter2.next();
        .... // Process the information as appropriate
    }
}
// ----- Commit and disconnect
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

完全な例については、CMBROOT¥Samples¥java¥d1 ディレクトリーにある、TListCatalogDB2.java、TListCatalogJDBC.java、およびTListCatalogDJ.java を参照してください。

C++

```

try {
    DKDatastoreDB2 dsDB2;
    DKString schema;
    DKSequentialCollection *pCol2 = 0;
    dkIterator *pIter = 0;
    dkIterator *pIter2 = 0;
    DKTableDefDB2 *pEnt = 0;
    DKString strServerName;
    DKString strTable;
    DKColumnDefDB2 *pAttr = 0;
    DKDatastoreDefDB2 *dsDefDB2 = 0;
    DKAny a;
    DKAny *pA = 0;
    long i = 0;
    long j = 0;

    // ----- Connect to datastore and set value for schema name
    . . .
    // ----- Create a datastore definition and set the schema name
    dsDefDB2 = (DKDatastoreDefDB2 *) dsDB2.datastoreDef();
    if (schema != "")
    {
        dsDefDB2->setSchemaName(schema);
    }

    // ----- Get a list of the entities (tables)
    pCol = (DKSequentialCollection*)((dkCollection*)dsDB2.listEntities());
    pIter = pCol->createIterator();
    i = 0;
    // ----- List the attributes (columns) for each entity (table)
    while (pIter->more() == TRUE)
    {
        i++;
        pEnt = (DKTableDefDB2*)((void*)(*pIter->next()));
        strTable = pEnt->getName();
        cout << "table name [" << i << "] - " << strTable << endl;
        cout << " list columns for " << strTable << " table" << endl;
        pCol2 =
        (DKSequentialCollection*)((dkCollection*)dsDB2.listEntityAttrs(strTable));
        pIter2 = pCol2->createIterator();
        j = 0;
        while (pIter2->more() == TRUE)
        {
            j++;
            pA = pIter2->next();
            pAttr = (DKColumnDefDB2*) pA->value();
            cout << "    Attr name [" << j << "] - " << pAttr->getName() << endl;
            cout << "        datastoreName " << pAttr->datastoreName() << endl;
            cout << "        datastoreType " << pAttr->datastoreType() << endl;
            cout << "        attributeOf " << pAttr->getEntityName() << endl;
            cout << "        type " << pAttr->getType() << endl;
            cout << "        size " << pAttr->getSize() << endl;
            cout << "        id " << pAttr->getId() << endl;
            cout << "        nullable " << pAttr->isNullable() << endl;
            cout << "        precision " << pAttr->getPrecision() << endl;
            cout << "        scale " << pAttr->getScale() << endl;
            cout << "        string type " << pAttr->getStringType() << endl;
            cout << "        primary key " << pAttr->isPrimaryKey() << endl;
            cout << "        foreign key " << pAttr->isForeignKey() << endl;
            delete pAttr;
        }
    }
    // continued...
}

```

C++ (続き)

```
        delete pIter2;
        delete pCol2;
        delete pEnt;
    }
    delete pIter;
    delete pCol;
    dsDB2.disconnect();
}
catch(DKException &exc)
{
    . . .
}
```

完全な例は、CMBROOT¥Samples¥cpp¥db2、odbc、および dj ディレクトリーの TListCatalogDB2.cpp、TListCatalogODBC.cpp、および TListCatalogDJ.cpp を参照してください。

照会の実行

照会を実行するには、まず照会ストリングを作成してから、その照会を実行する必要があります。以下の例では、照会を実行し、その結果を処理します。

Java

```
// ----- After creating a datastore and connecting, build the
//           query and parameters and execute it
sDB2 = new DKDatastoreDB2();
dkResultSetCursor pCur = null;
DKNVPair parms[] = new DKNVPair[2];
String strMax = "5";
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[1] = new DKNVPair(DK_CM_PARM_END, null);
// ----- Connect to datastore
dsDB2.connect(database, userid, pw, "");
// --- Create the query string
String cmd = "";
cmd = "SELECT * FROM EMPLOYEE";

DKDDO p = null;
DKDDO pDDO = null;
dkXDO pXDO = null;
DKPidXDO pidXDO = null;
int i = 0;
int numDataItems = 0;
short k = 0;
short j = 0;
String strDataName;
dkCollection pCol = null;
dkIterator pIter = null;
Object a = null;
dkDataObjectBase pDO = null;
int cnt = 0;

// continued...
```

Java (続き)

```
// ----- Execute the query
pCur = dsDB2.execute(cmd,DK_CM_SQL_QL_TYPE,parms);
if (pCur == null)
{
    // Handle if the cursor is null
}
while (pCur.isValid())
{
    p = pCur.fetchNext();
    if (p != null)
    {
        cnt++;
        i = pCur.getPosition();
        // Get item information
        numDataItems = p.dataCount();
        DKPid pid = p.getPidObject();
        System.out.println("pid string " + pid.pidString());
        System.out.println("Number of Data Items " + numDataItems);
        for (j = 1; j <= numDataItems; j++)
        {
            a = p.getData(j);
            strDataName = p.getDataName(j);
            // Handle the attributes ;
            if (a instanceof String)
            {
                System.out.println("    Attribute Value " + a);
            }
            ..... // Handle for various types )
            else if (a instanceof dkDataObjectBase)
            {
                pDO = (dkDataObjectBase)a;
                if (pDO.protocol() == DK_PDDO)
                {
                    System.out.println("    DKDDO object ");
                    pid = ((DKDDO)pDO).getPidObject();
                }
                else if (pDO.protocol() == DK_XDO)
                {
                    // dkXDO object
                    pXDO = (dkXDO)pDO;
                    pidXDO = pXDO.getPidObject();
                }
            }
            ..... // Handle for various types
        }
    }
}
// Delete the cursor when you're done, commit and disconnect
pCur.destroy(); // Finished with the cursor
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

完全な例については、CMBROOT¥Samples¥java¥d1 ディレクトリーにある、TExecuteDB2.java、TExecuteJDBC.java、および TExecuteDJ.java を参照してください。

C++

```
try {
    DKDatastoreDB2 dsDB2;
    dkResultSetCursor* pCur = 0;
    DKNVPair par[2];
    DKAny anyValue;
    DKString strMax = "5";
    anyValue = strMax;
    par[0].set(DK_CM_PARM_MAX_RESULTS, anyValue);
    par[1].setName(DK_CM_PARM_END);
    // ---- Create a datastore and connect
    . . .
    // ---- Create a query string containing the select
    DKString qstring = "SELECT * FROM EMPLOYEE";
    // ---- Execute the query
    pCur = dsDB2.execute(qstring, DK_CM_SQL_QL_TYPE, par);
    // ---- Declarations
    DKDDO *p = 0;
    dkDataObjectBase *pDOBase = 0;
    DKDDO *pDDO = 0;
    dkXDO *pXDO = 0;
    DKAny a;
    ushort j = 0;
    ushort k = 0;
    ushort val = 0;
    ushort cnt = 1;
    DKString strData = "";
    DKString strDataName = "";
    dkCollection* pdCol = 0;
    dkIterator* pdIter = 0;
    ushort numDataItems = 0;
    DKString strPid;
    DKPid* pid = 0;
    short sVal = 0;
    long lVal = 0;
    while (pCur->isValid())
    {
        p = pCur->fetchNext();
        if (p != 0)
        {
            cout << "======" << "Item " << cnt << " <======" << endl;
            numDataItems = p->dataCount();
            pid = (DKPid*)p->getPidObject();
            strPid = pid->pidString();
            cout << "pid string " << strPid << endl;
            k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
            if (k > 0)
            {
                a = p->getProperty(k);
                val = a;
                switch (val)
                {
                    case DK_CM_DOCUMENT :
                    {
                        cout << "Item is document " << endl;
                        break;
                    }
                }
            }
        }
    }
    cout << "Number of Data Items " << numDataItems << endl;
```


C++ (続き)

```
for (j = 1; j <= numDataItems; j++)
{
    a = p->getData(j);
    strDataName = p->getDataName(j);
    switch (a.typeCode())
    {
        case DKAny::tc_string :
        {
            strData = a;
            cout << "attribute name : " << strDataName << " value : "
<< strData << endl;
            break;
        }
        // ---- Handle each type in a similar fashion
        . . .
    }
}
// ----- Delete the cursor and disconnect
if (pCur != 0)
    delete pCur;
dsDB2.disconnect();
}
catch(DKException &exc)
. . .
```

完全な例は、CMBROOT¥Samples¥cpp ディレクトリーの TExecuteDB2.cpp、TExecuteODBC.cpp、および TExecuteDJ.cpp を参照してください。

カスタム・コンテンツ・サーバー・コネクタの作成

カスタム・コンテンツ・サーバー (現在 Enterprise Information Portal に組み込まれていない) 用に、ユーザー独自のサーバー定義を作成できます。カスタム・サーバーを EIP に統合する場合は、定義をサポートするユーザー独自の Java クラスまたは C++ クラスを提供する必要があります。

カスタム・コンテンツ・サーバー・コネクタの開発

オブジェクト指向の API フレームワークは、以下の目標を掲げて設計されています。

- 追加のデータ・ストレージ・システムまたはコンテンツ・サーバーを、フレームワークに追加できる。
- コンテンツ・サーバーのあらゆる複合データ・タイプをマッピングできる。
- すべてのコンテンツ・サーバー・データ・アクセス用の共通オブジェクト・モデル。
- さまざまなタイプの検索エンジン (テキスト検索エンジンなど) やイメージ検索 (QBIC) などを組み合わせて使用できる柔軟なメカニズム。
- Java アプリケーション・ユーザー用のクライアント/サーバーのインプリメンテーション。

特定のオブジェクト指向 API については、「オンライン API 解説書」を参照してください。

カスタム・コンテンツ・サーバーを Enterprise Information Portal に統合する場合、以下のことを実行する必要があります。

- `com.ibm.mm.sdk.common` パッケージをインポートします。
- **Java のみ:** 共通フレームワークにアクセスするために、`cmbcm81.jar` (Java) ファイルにリンクします。
- **C++ のみ:** 共通フレームワークにアクセスするために、非デバッグ・バージョンの `cmbcm817.dll` ファイルと、デバッグ・バージョンの `cmbcm8167.dll` ファイルにリンクします。

Enterprise Information Portal データベース・インフラストラクチャー

`dkDatastore` クラスは、Enterprise Information Portal とコンテンツ・サーバーの間の基本インターフェースとして機能します。特定のコンテンツ・サーバーのインプリメンテーション情報を提供するために、各コンテンツ・サーバーには `dkDatastore` クラスをインプリメントする別個のクラスがあります。各コンテンツ・サーバー・タイプは、`DKDatastorexx` というクラスで表されています (`xx` は特定のコンテンツ・サーバーの名前またはタイプ)。41 ページの表 7 に、Enterprise Information Portal で現在提供されているコンテンツ・サーバーのリストを示します。

ユーザーがサーバー定義を作成する場合、EIP システム管理クライアントで、コンテンツ・サーバー用に作成した `DKDatastore` クラスを指定する必要があります。

Enterprise Information Portal の共通クラス

`dkDDO`

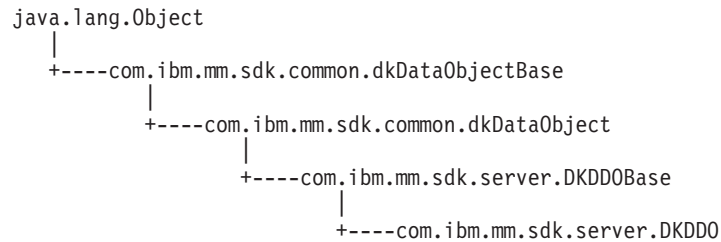
`dkDDO` クラスは、オブジェクトのタイプに関係なく、オブジェクトのデータの定義およびアクセスを行うための表現およびプロトコルを提供します。DDO プロトコルは、オブジェクトの各データ項目の定義、追加、およびアクセスを行うための関数のセットとしてインプリメントされています。このプロトコルを使用することにより、動的にオブジェクトを作成し、コンテンツ・サーバーのタイプに関係なくそれをコンテンツ・サーバーから取得することができます。

コンテンツ・サーバーのインプリメントでは、コンテンツ・サーバー・クラスに登録されているスキーマ・マッピング情報を使用できます。このスキーマは、それぞれの永続データ項目を、コンテンツ・サーバー内の基本的な表現にマッピングします。

DDO には属性のセットがあり、それぞれの属性には、タイプ、値、およびプロパティーが関連付けられています。DDO 自体にも DDO 全体に属するプロパティーがあります。例えば、クラスを Content Manager コンテンツ・サーバー内の項目、または OnDemand 内の文書にマッピングすることができます。

Java

以下の図は、dkDDO クラスの階層を表したものです。



dkXDO

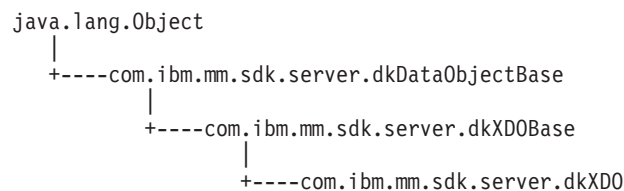
dkXDO クラスは、複雑なユーザー定義型 (UDT) またはラージ・オブジェクト (LOB) を表します。これは、それ自体独立して存在するか、または DDO の一部として存在します。そのため、これは永続 ID (PID) および、create、retrieve、update、そして delete の各関数を持ちます。

dkXDO クラスは、独立したコンテンツ・サーバーの access、create、retrieve、update、および delete 関数を定義することによって、dkXDODBase の共用インターフェースを拡張します。これらの関数を使用することにより、アプリケーションは、関連付けられた DDO クラス・オブジェクトや独立した XDO がなくても、コンテンツ・サーバーとの間でオブジェクト・データの保管や検索を実行できます。

コンテンツ・サーバー内での位置を判別するために、独立型 XDO の PID を設定する必要があります。DDO と一緒に XDO を使用している場合、PID は自動的に設定されます。例えば、クラスを Content Manager コンテンツ・サーバーの項目にマッピングしたり、OnDemand コンテンツ・サーバーの注にマッピングしたりできます。

Java

以下は、dkXDO クラスのクラス階層を示しています。



dkCollection

dkCollection クラスはオブジェクトのコレクションです。dkCollection は照会を評価できません。コレクションには名前が付けられている場合があります (デフォルトの名前は空ストリングです)。例えば、DKParts は DKSequentialCollection のサブクラスであり、DKSequentialCollection は dkCollection のサブクラスです。

DKResults

DKResults は dkQueryableCollection のサブクラスなので、ソートおよび両

方向イテレーターをサポートしており、照会可能です。DKResults クラスの要素・メンバーはオブジェクト (つまり dkDDO クラスのインスタンス) であり、これは照会結果を表します。このクラスが作成するイテレーターは dkSequentialIterator です。

Java

以下に、DKResults クラスのクラス階層を示します。

```
java.lang.Object
|
+----com.ibm.mm.sdk.server.DKSequentialCollection
|
+----com.ibm.mm.sdk.server.dkQueryableCollection
|
+----com.ibm.mm.sdk.server.DKResults
```

dkQuery

dkQuery は、特定のコンテンツ・サーバーと関連付けられている照会オブジェクト用のインターフェースです。このインターフェースをインプリメントするオブジェクトは、コンテンツ・サーバー・クラスによって作成されます。照会の結果は、通常は DKResults オブジェクトです。dkQuery インターフェースの具体的なインプリメンテーション例としては、関連するコンテンツ・サーバーによって作成される DKParametricQuery、DKTextQuery、DKImageQuery があります。

dkCQExpr

dkCQExpr クラスは、複合または結合照会式をあらわします。これには、dkQExpr 照会式ツリーを含めることができます。このツリーには、パラメトリック、テキスト、およびイメージの各照会の組み合わせを含めることができます。各コンテンツ・サーバーで統合検索を使用できるようにする場合、コンテンツ・サーバーはこの dkCQExpr オブジェクトを処理できなければなりません。

dkSchemaMapping

dkSchemaMapping は、コンテンツ・サーバーで統合エンティティと、固有エンティティとの間の関連マッピングを定義するインターフェースです。コンテンツ・サーバーは、照会において統合エンティティおよび統合属性と、その固有エンティティおよび固有属性との間のマッピングを削除してからもう一度マッピングしてから結果を戻すために、このマッピング・クラスを認識している必要があります。

dkDatastore および関連クラス

コンテンツ・サーバーで使用するために、以下の各クラスまたは各インターフェースにつき、具象クラスを 1 つずつインプリメントしなければなりません。

OnDemand サーバーの例では、dkDatastore インターフェースを実装する具象クラスは DKDatastoreOD です。

dkDatastore

dkDatastore は、コンテンツ・サーバーとの接続、そのトランザクションおよびコマンドを表し、それを管理します。これは evaluate 関数をサポートしているので、照会マネージャーのサブクラスと考えることができます。

以下は、dkDatastore インターフェースの主なメソッドです。

connect()

コンテンツ・サーバーと接続します。

disconnect()

コンテンツ・サーバーから切断します。

evaluate()、execute()、executeWithCallback()

コンテンツ・サーバーを照会します。

commit()、rollback()

コンテンツ・サーバーでトランザクションを実行します。

制約事項: コンテンツ・サーバーの中には、これらの関数をサポートしないものもあります。

registerServices()、unregisterServices()

検索エンジンを登録します。

changePassword(userid, oldPasswd, newPasswd)

コンテンツ・サーバーから、現在のログオン・ユーザー ID のログオン・パスワードを変更します。

listDataSources()

ログオン用のコンテンツ・サーバー・ユーザー ID オブジェクトのコレクションを戻します。この関数を使用するためにコンテンツ・サーバーに接続する必要はありません。

listDataSourceNames()

コンテンツ・サーバー名の配列を戻します。

getExtension(String)

コンテンツ・サーバーから dkExtension オブジェクトを取得します。指定した拡張がすでに存在しないものの、コンテンツ・サーバーによってサポートされている場合、新しく作成されたオブジェクトが戻されます。そうでない場合はヌル値が戻されます。

addExtension(String, dkExtension)

新しい拡張オブジェクト (XDO) をこのコンテンツ・サーバーに追加します。

createDDO(String,int)

指定したオブジェクト・タイプおよびフラグに基づいて、データ・オブジェクトを作成します。作成した DDO は、プロパティおよび属性をすべて設定した新しい DKDDO オブジェクトを戻します。呼び出し側プログラムでは、このデータ・オブジェクトの属性値を提供しなければなりません。

以下は、dkDatastore インターフェースのデータ・オブジェクト操作メソッドです。

addObject(dkDataObject)

新しい文書またはフォルダーをコンテンツ・サーバーに追加します。

retrieveObject(dkDataObject)

コンテンツ・サーバーにある文書またはフォルダーを検索します。

deleteObject(dkDataObject)

コンテンツ・サーバーから文書またはフォルダーを削除します。

updateObject(dkDataObject)

コンテンツ・サーバーの文書またはフォルダーを更新します。

moveObject(dkDataObject, String)

あるエンティティから別のエンティティへ、フォルダーまたは文書を移動します。

以下は、dkDatastore インターフェースのスキーマ・マッピング関連のメソッドです。

registerMapping(DKNVPair)

このコンテンツ・サーバーにマッピング情報を登録します。

unRegisterMapping(String)

このコンテンツ・サーバーからのマッピング情報を除去します。

listMappingNames()

このコンテンツ・サーバーからマッピング名の配列を戻します。

getMapping(String)

dkSchemaMapping オブジェクトを戻します。

dkDatastoreDef

dkDatastoreDef インターフェースは、コンテンツ・サーバー情報にアクセスする関数や、そのエンティティの作成、リスト作成、および削除を実行する関数を定義します。これは、dkEntityDef オブジェクトのコレクションを保持します。

表 22 では、dkDatastoreDef インターフェースの具象クラスの例を示しています。

表 22. dkDatastoreDef の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKDatastoreDefICM
OnDemand	DKDatastoreDefOD
Content Manager for AS/400	DKDatastoreDefV4
ImagePlus for OS/390	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Extended Search	DKDatastoreDefDES
IBM DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
旧バージョンの Content Manager	DKDatastoreDefDL

以下は、dkDatastoreDef インターフェースの主なメソッドです。

listEntities()
エンティティをリストします。

listEntityAttrs()
エンティティ属性をリストします。

addEntity()
エンティティを追加します。

getEntity(name)
エンティティを取得します。

各具象クラスには、各コンテンツ・サーバー固有の関数を含めることが可能であり、それらにコンテンツ・サーバーにとって理解しやすい名前を付けることができます。例えば、DKDatastoreDefDL クラスには、以下の固有の関数が含まれています。

- listIndexClassNames()
- listIndexClasses()

DKDatastoreDefOD クラスには、以下の固有の関数が含まれています。

- listAppGrps()
- listAppGrpNames()

dkEntityDef

dkEntityDef クラスは、以下を行うための関数を定義します。

- エンティティ情報へのアクセス。
- 属性の作成と削除。
- エンティティの作成と削除。

dkEntityDef クラスにおいて、サブエンティティに関連するすべての関数は、デフォルトのコンテンツ・サーバーがサブエンティティをサポートしていないことを示す DKUsageError を生成します。しかし、コンテンツ・サーバーがこの種の複数レベル・エンティティをサポートする場合 (例: Domino.Doc など)、そのコンテンツ・サーバーのサブクラスは、例外を上書きする適切な関数をインプリメントしなければなりません。

表 23 では、dkEntityDef クラスの具象クラスの例を示しています。

表 23. dkEntityDef の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKItemTypeDefDL
OnDemand	DKAppGrpDefOD
Content Manager for AS/400	DKIndexClassDefV4
ImagePlus for OS/390	DKEntityDefIP
Domino.Doc	DKCabinetDefDD
Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC

表 23. *dkEntityDef* の具象クラス (続き)

サーバー・タイプ	クラス名
旧バージョンの Content Manager	DKIndexClassDefDL

以下は、*dkEntityDef* クラスの主な関数です。

listAttrs()

エンティティ属性をリストします。

getAttr(String attrName)

指定したエンティティ属性を取得します。

addAttr(DKAttrDef)

属性をエンティティに追加します。

getName()

エンティティの名前を取得します。

setName(String)

エンティティの名前を設定します。

hasSubEntities()

エンティティにサブエンティティが含まれるかどうかを判定します。

getSubEntity(String)

サブエンティティを取得します。

addSubEntity(dkEntityDef)

エンティティにサブエンティティを追加します。

listSubEntities()

エンティティのサブエンティティのリストを取得します。

removeAttr(String)

エンティティからサブエンティティを除去します。

add() コンテンツ・サーバーにエンティティを追加します。

update()

コンテンツ・サーバー内のエンティティを更新します。

retrieve()

コンテンツ・サーバーからエンティティ値を取得します。

del() コンテンツ・サーバーからエンティティを削除します。

dkAttrDef

dkAttrDef クラスは、属性情報にアクセスする関数や、属性を作成および削除する関数を定義します。表 24 では、*dkAttrDef* クラスの具象クラスの例を示しています。

表 24. *dkAttrDef* の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKAttrDefDICM
OnDemand	DKFieldDefOD

表 24. *dkAttrDef* の具象クラス (続き)

サーバー・タイプ	クラス名
Content Manager for AS/400	DKAttrDefV4
ImagePlus for OS/390	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Extended Search	DKFieldDefDES
DB2 Universal Database	DKColumnDefDB2
JDBC	DKColumnDefJDBC
ODBC	DKColumnDefODBC
旧バージョンの Content Manager	DKAttrDefDL

以下は、*dkAttrDef* クラスの主なメソッドです。

listAttrs()

属性をリストします。

getAttr(String attrName)

指定した属性を取得します。

getName()

属性の名前を取得します。

getDescription()

属性の説明を取得します。

add() コンテンツ・サーバーにエンティティを追加します。

dkServerDef

dkServerDef クラスは、各コンテンツ・サーバーのサーバー定義情報を提供します。表 25 では、*dkServerDef* クラスの具象クラスの例を示しています。

表 25. *dkServerDef* の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKServerDefICM
OnDemand	DKServerDefOD
Content Manager for AS/400	DKServerDefV4
Domino.Doc	DKServerDefDD
Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
JDBC	DKServerDefJDBC
ODBC	DKServerDefODBC
旧バージョンの Content Manager	DKServerDefDL

以下は、*dkServerDef* クラスの主な関数です。

setDatastore(dkDatastore ds)

コンテンツ・サーバー・オブジェクトへの参照を設定します。

getDatastore()

コンテンツ・サーバー・オブジェクトへの参照を入手します。

getName()

コンテンツ・サーバーの名前を取得します。

setName(String name)

コンテンツ・サーバーの名前を設定します。

datastoreType()

コンテンツ・サーバーのタイプを入手します。

dkResultSetCursor

dkResultSetCursor は、DDO オブジェクトの仮想コレクションを管理するために使用できる、照会結果セット内のコンテンツ・サーバー・カーソルです。このコレクションは、照会結果の集合です。コレクションの各エレメントは、コンテンツ・サーバーがそのエレメントを取り出すまで作成されません。

以下は、dkResultSetCursor クラスの主な関数です。

isScrollable()

カーソルを前後にスクロールできる場合に TRUE を返します。

isUpdatable()

カーソルを更新できる場合に TRUE を返します。

isValid()

カーソルが有効な場合に TRUE を返します。

isBegin()

カーソルが結果セットの先頭位置にある場合に TRUE を返します。

isEnd()

カーソルが結果セットの終了位置にある場合に TRUE を返します。

isInBetween()

カーソルが、結果セット内のデータ・エレメントとデータ・エレメントの間にある場合に TRUE を返します。

getPosition()

カーソルの現在位置を取得します。

setPosition(int position, Object value)

指定した位置にカーソルを設定します。

setToNext()

結果セットの中の次のエレメントにカーソルを設定します。

fetchObject()

結果セットから現在のエレメントを取り出し、それを DDO として返します。

fetchNext()

結果セットから次のエレメントを取り出し、それを DDO として返します。

findObject(int position, String predicate)

指定した述語を満足するデータ・オブジェクトを検出し、その位置にカーソルを移動し、そのオブジェクトを取り出します。

addObject(DKDDO ddo)

特定の DDO によって表される、同じタイプの新しいエレメントを、コンテンツ・サーバーに追加します。

deleteObject()

現在のエレメントをコンテンツ・サーバーから削除します。

updateObject(DKDDO ddo)

コンテンツ・サーバー内の現在の位置にある現在のエレメントを、指定した DDO を使って更新します。

newObject()

同じタイプのエレメントを作成し、それを DDO として戻します。

open()

カーソルをオープンし、必要なら結果セットを作成するための照会を実行します。

close()

カーソルと結果セットをクローズします。

isOpen()

カーソルがオープンされているなら、TRUE を戻します。

destroy()

カーソルを削除します。これにより、カーソルがガーベッジとして収集される前にクリーンアップ処理を実行できます。

datastoreName()

カーソルが属するコンテンツ・サーバーの名前を入手します。

datastoreType()

カーソルが属するコンテンツ・サーバーを入手します。

handle(int type)

タイプを指定して、結果セット・カーソルと関連付けられた結果セット・ハンドルを取得します。

要件: addObject、 deleteObject、 および updateObject 関数を使用するには、コンテンツ・サーバー・オプション DK_DL_OPT_ACCESS_MODE を DK_READWRITE に設定しなければなりません。

dkBlob

dkBlob は、基本バイナリー・ラージ・オブジェクト (BLOB) データ型用の共通の共用インターフェースを宣言する、抽象クラスです。dkBlob クラスから派生した具象クラスは、この共通の共用インターフェースを共有します。これにより、異機種のコンテンツ・サーバーが引き起こす BLOB のコレクションの、多形的処理が可能になります。dkClob および dkDBClob クラスも具象クラスを持つことが可能です。

表 26 では、dkBlob クラスの具象クラスの例を示しています。

表 26. dkBlob の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKLobICM
OnDemand	DKBlobOD
Content Manager for AS/400	DKBlobV4
ImagePlus for OS/390	DKBlobIP
Domino.Doc	DKBlobDD
Extended Search	DKBlobDES
DB2 Universal Database	DBBlobDB2、DKBlobDB2
JDBC	DKBlobJDBC、DKBlobJDBC
ODBC	DKBlobODBC、DKBlobODBC
旧バージョンの Content Manager	DKBlobDL

以下は、dkBlob クラスの主なメソッドです。

getContent()

オブジェクトの BLOB データを含むバイト配列を戻します。

getContentToClientFile(String afileName, int fileOption)

BLOB データをオブジェクトから指定したファイルにコピーします。

setContent(byte[] aByteArr)

バイト配列の内容により、オブジェクトの LOB データを設定します。

setContentFromClientFile(String afileName)

オブジェクトの LOB データを、ファイル afileName の内容で置換します。

add(String afileName)

指定したファイルの内容をコンテンツ・サーバーに追加します。

retrieve(String afileName)

コンテンツ・サーバーのコンテンツを取り出して、指定したファイルに入れます。

update(String afileName)

オブジェクトおよびコンテンツ・サーバーを、指定したファイルの内容により更新します。

del(boolean flush)

flush が TRUE なら、オブジェクトのデータをコンテンツ・サーバーから削除します。それ以外の場合は、現在のコンテンツが保たれます。

concatReplace(dkBlob aBlob), concatReplace(byte[] aByteArr)

このオブジェクトを別の dkBlob オブジェクトまたはバイト配列に連結します。

length()

オブジェクトの LOB コンテンツの長さを戻します。

indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)

このオブジェクトの中で、検索引き数がオフセット開始位置以降に最初に出現する場所のバイト・オフセットを戻します。

substring(int startPos, int length)

このオブジェクトの LOB データのサブストリングを含むストリング・オブジェクトを戻します。

remove(int startPos, int aLength)

このオブジェクトの LOB データのうち、 *startPos* から始まる *aLength* バイトを削除します。

insert(String aString, int startPos), insert(dkBlob aBlob, int startPos)

このオブジェクトの LOB データのうち、 *startPos* の位置の次に引き数データを挿入します。

open(String afileName)

このオブジェクトの内容をファイル *afileName* にアンロードしてから、デフォルトのファイル・ハンドラーを実行します。

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)

クラス全体のファイル・ハンドラーを、実行可能プログラム名により識別します。また、この関数は、このファイル・オブジェクトに関して、ハンドラーを同期で実行する非同期で実行するかも指示します。

setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)

ファイル・ハンドラーを実行可能プログラム名により識別し、このオブジェクトに関して、それを同期で実行するか非同期で実行するかを指定します。

getOpenHandler()

クラス全体のファイル・ハンドラーの実行プログラム名を取得します。

isOpenSynchronous()

ファイル・ハンドラーの現在の同期設定を戻します。

dkClob

dkClob は、文書などの文字ラージ・オブジェクト (CLOB) データ・タイプを保管するための共用インターフェースを宣言する抽象クラスです。

表 27 に、 dkClob クラスの具象クラスの例を示します。

表 27. dkClob の具象クラス

サーバー・タイプ	クラス名
DB2 Universal Database	DKClobDB2
ODBC	DKClobODBC

以下は、dkClob クラスの主な関数です。

open()

Open() は dkXDOBase から継承したメンバーです。Open() は、dkClob の具象サブクラスによってインプリメントまたはオーバーライドされます。

dkXDO のメンバー: dkXDO& add(), dkXDO retrieve(), dkXDO update(), dkXDO del()

dkXDO からの保護メンバーとして継承されます。必要な場合、これらの保護メンバーは、dkClob の具象サブクラスによってインプリメントまたはオーバーライドされます。

dkClob で定義されているメンバーは、下記のとおりです。

add(String afileName)

指定したファイルの内容をコンテンツ・サーバーに追加します。

retrieve(String afileName)

コンテンツ・サーバーのコンテンツを取り出して、指定したファイルに入れます。

update(String afileName)

オブジェクトおよびコンテンツ・サーバーを、指定したファイルの内容により更新します。

del(DKBoolean flush)

flush が TRUE なら、オブジェクトのデータをコンテンツ・サーバーから削除します。それ以外の場合は、現在のコンテンツが保たれます。

getContentToClientFile(String afileName, int fileOption)

CLOB データをオブジェクトから指定したファイルにコピーします。

setContentFromClientFile(String afileName)

オブジェクトの LOB データを、ファイル *afileName* の内容で置換します。

indexOf(String& aString, long startPos=1), indexOf(dkClob& adkClob, long startpos=1)

このオブジェクトの中で、検索引き数がオフセット開始位置以降に最初に出現する場所のバイト・オフセットを戻します。

substring(long startpos, long length)

このオブジェクトの LOB データのサブストリングを含むストリング・オブジェクトを戻します。

remove(long startpos, long aLength)

このオブジェクトの LOB データのうち、*startPos* から始まる *aLength* バイトを削除します。

insert(DKString aString, long startpos), insert(dkClob& adkClob, long startpos)

このオブジェクトの CLOB データのうち、*startPos* の位置の次に引き数データを挿入します。

open(String afileName)

このオブジェクトの内容をファイル *afileName* にアンロードしてから、デフォルトのファイル・ハンドラーを実行します。

setInstanceOpenHandler(String ahandler, DKBoolean newSynchronousFlag)

ファイル・ハンドラーを実行可能プログラム名により識別し、このオブジェクトに関して、それを同期で実行するか非同期で実行するかを指定します。

setClassOpenHandler(String ahandler, DKBoolean newSynchronousFlag)

クラス全体のファイル・ハンドラーを、実行可能プログラム名により識別します。また、この関数は、このファイル・オブジェクトに関して、ハンドラーを同期で実行する非同期で実行するかも指示します。

getOpenHandler()

クラス全体のファイル・ハンドラーの実行プログラム名を取得します。

isOpenSynchronous()

ファイル・ハンドラーの現在の同期設定を戻します。

dkAnnotationExt

dkAnnotationExt は、すべての注釈オブジェクト用のインターフェース・クラスです。コンテンツ・サーバーが注釈データをサポートしている場合、このインターフェースをインプリメントする必要があります。この注釈オブジェクトは、DKBlobxx クラスの拡張であり、その dkBlob オブジェクトはバイナリー注釈データと DKParts コレクションを表します。

dkDatastoreExt

dkDatastoreExt クラスは、標準のコンテンツ・サーバー拡張クラスを定義します。

表 28 では、dkDatastoreExt クラスの具象クラスの例を示しています。

表 28. dkDatastoreExt の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKDatastoreExtICM
OnDemand	DKDatastoreExtOD
Content Manager for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Extended Search	DKDatastoreExtDES
DB2 Universal Database	DKDatastoreExtDB2
JDBC	DKDatastoreExtJDBC
旧バージョンの Content Manager	DKDatastoreExtDL

以下は、dkDatastoreExt クラスの主な関数です。

getDatastore()

所有コンテンツ・サーバー・オブジェクトへの参照を入手します。

setDatastore(dkDatastore ds)

所有コンテンツ・サーバー・オブジェクトへの参照を設定します。

isSupported(String functionName)

指定した関数名がこの拡張でサポートされているかどうかを判定します。

listFunctions()

拡張に関してサポートされているすべての関数名のリストを取得します。

addToFolder(dkDataObject folder, dkDataObject member)

このフォルダーとコンテンツ・サーバーにメンバーを追加します。

removeFromFolder(dkDataObject folder, dkDataObject member)

このフォルダーおよびコンテンツ・サーバーからメンバーを除去します。

checkout(dkDataObject item)

コンテンツ・サーバーから文書またはフォルダー項目をチェックアウトします。項目がチェックアウトされている間は、その項目に対する排他的更新特権が付与され、他のユーザーは読み取りアクセスだけが可能になります。

checkin(dkDataObject item)

コンテンツ・サーバーから前にチェックアウトした文書またはフォルダー項目をチェックインします。ファイルをチェックインすると、この関数によりすべての書き込み特権が解放されます。

getCommonPrivilege()

特定のコンテンツ・サーバーの共通特権を入手します。

isCheckedOut(dkDataObject item)

ある文書またはフォルダー項目がコンテンツ・サーバーからチェックアウトされたかどうかを判定します。

checkedOutUserid(dkDataObject item)

コンテンツ・サーバーから項目をチェックアウトしたユーザー ID を入手します。

unlockCheckedOut(dkDataObject item)

コンテンツ・サーバーの項目をアンロックします。

changePassword (String userId, String oldPwd, String newPwd)

指定したユーザー ID に関して、コンテンツ・サーバーでのパスワードを変更します。

moveObject (dkDataObject dataObj, String entityName)

あるエンティティから別のエンティティへ *entityName* オブジェクトを移動させます。

retrieveFormOverlay(String id)

書式オーバーレイ・オブジェクトを検索します。

DKPidXDO

DKPidXDO クラスは、コンテンツ・サーバー内の BLOB データの永続識別を表します。

表 29 では、DKPidXDO クラスの具象クラスの例を示しています。

表 29. DKPIdXDO の具象クラス

サーバー・タイプ	クラス名
旧バージョンの Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
Content Manager for AS/400	DKPidXDOV4
ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
JDBC	DKPidXDOJDBC
ODBC	DKPidXDOODBC

dkUserManagement

dkUserManagement クラスは、コンテンツ・サーバーのすべてのユーザー管理関数を表し、それを処理します。

表 30 では、dkUserManagement クラスの具象クラスの例を示しています。

表 30. dkUserManagement の具象クラス

サーバー・タイプ	クラス名
Content Manager	DKUserMgmtICM
Content Manager for AS/400	DKUserMgmtV4
ImagePlus for OS/390	DKUserMgmtIP
旧バージョンの Content Manager	DKUserMgmtDL

DKConstant

すべての共通定数は、DKConstant クラスで定義されます。各コンテンツ・サーバーには、そのコンテンツ・サーバー固有の定数を定義するための、独自の DKConstantxx クラスがあります。

推奨事項: 可能な限り、すべてのコンテンツ・サーバーで共通のメッセージを使用します。

DKMessageId

すべての共通メッセージ ID は、このクラスで定義されます。各コンテンツ・サーバーには、独自のメッセージ ID を定義するための独自の DKMessageIdxx クラスがあります。

推奨事項: 可能な限り、すべてのコンテンツ・サーバーで共通のメッセージを使用してください。

以下のプロパティ・ファイルには、共通警告およびエラー・メッセージが含まれています。

Java の場合:

- DKMessage_en.properties
- DKMessage_es.properties

C++ の場合:

- DKMessage_en_US.properties
- DKMessage_es_ES.properties

各コンテンツ・サーバーには、そのサーバーの警告およびエラー・メッセージ用の独自の DKMessagexx_yy_zz.properties ファイルがあります。

FeServerDefBase クラスの使用 (Java のみ)

FeServerDefBase クラスは抽象クラスで、カスタム・サーバー定義を作成するためには、このクラスを拡張しなければなりません。この基本クラスを拡張する Java クラスには、以下のパラメーターを受け入れてそれらをスーパー・クラスに渡すコンストラクターが必要です。

String connectString

サーバーの接続ストリング。

String[] serverList

定義済みのサーバーのリスト。

String[] associatedServerList

このサーバーに関連付けられたサーバーのリスト (ない場合はヌル)。

String[] serverTypes

定義済みのサーバー・タイプ ID のリスト。

String[] serverTypeDescriptions

定義済みのサーバー・タイプの説明のリスト。

FeServerDefBase クラスを拡張する Java クラスを作成する場合、新しいサーバー・ダイアログのデータを処理する方法を決めなければなりません。同じクラスを使用することもできますし、別のモデル・クラスを使用することもできます。カスタム・コンテンツ・サーバーで、接続ストリングのフィールドよりも多くのフィールドを必要とする場合、追加の関数が正しく実行されるように、モデルとして Enterprise Information Portal データベースおよび Java API を使用しなければなりません。

Enterprise Information Portal Administration プログラムでそのコンテンツ・サーバーが選択されている場合、「新規 (New)」メニューには、Enterprise Information Portal データベースの FASERVERTYPES テーブルに保管されているサーバー・タイプのリストが含まれます。このテーブルには、そのメニュー項目が選択されている場合にインスタンスが生成される Java クラスの名前が含まれています。

パスワード検査をサポートする場合、Java クラスを、Enterprise Information Portal Administration .jar ファイルと同じディレクトリーに置かなければなりません。それにより、動的にその Java クラスのインスタンスを生成し、ユーザー入力パスワードをパラメーターとして指定して verify メソッドを呼び出すことができます。verify メソッドは、有効なパスワードの場合はヌルを、無効なパスワードの場合は情報が含まれているストリングの配列を戻します。

EIP ワークフロー・アプリケーションの構築

EIP クラスおよび API を使用することにより、EIP ワークフロー・サポートを使用するアプリケーションをユーザー独自に作成したり、拡張したりできます。通常は、統合検索を実行し、検索結果 (コンテンツ項目、または複数のコンテンツ項目からなるフォルダー) を使用してワークフローを開始します。API を使用してワーク・リストにアクセスしてから、そのワーク・リスト・コンテンツを表示します。それぞれのアクティビティーが完了すると、ワークフローの処理はワークフロー内の次のアクティビティーに進みます。

ワークフロー・サービスへの接続

アプリケーションで Enterprise Information Portal ワークフローを使用するには、まず DKWorkflowServicesFed のインスタンスを作成し、次にこのインスタンスに接続します。以下の例では、ワークフロー・サービスを開始します。

Java

```
// ----- Create the strings for the name of the
//service, user ID and Password
String wfsrv = "icmnlsdb";
String userid = "icmadmin";
String pw = "password";
// ----- Create a federated datastore
DKDatastoreFed dsFed = new DKDatastoreFed();
dsFed.connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkflowServiceFed svWF =new DKWorkflowServiceFed ();
// ----- Set the datastore in the workflow service
svWF.setDatastore(dsFed);
// ----- Connect to the service
svWF.connect (wfsrv, userid, pw,"");
```

C++

```
// ----- Create the strings for the name of the service, user ID
// ----- and Password
DKString wfsrv = "icmnlsdb";
DKString userid = "icmadmin";
DKString pw = "password";
// ----- Create a federated datastore
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkflowServiceFed* svWF =new DKWorkflowServiceFed ();
// ----- Set the datastore in the workflow service
svWF->setDatastore(dsFed);
// ----- Connect to the service
svWF->connect (wfsrv, userid, pw,"");
```

ワークフロー・サービスの使用が終了したら、disconnect() および delete() 関数を呼び出して切断する必要があります。

Java

```
svWF.disconnect();  
dsFed.disconnect();  
svWF.destroy();  
dsFed.destroy();
```

C++

```
svWF->disconnect();  
dsFed->disconnect();  
delete svWF;  
delete dsFed;
```

ワークフローの開始

ワークフローを作成したら、ワークフローを開始する必要があります。ワークフローを開始するには、次の手順で行います。

1. DKWorkflowFed オブジェクトを作成し、ワークフロー名を設定します。
2. 有効なワークフロー・テンプレート (Enterprise Information Portal ワークフロー・ビルダーで定義されたワークフロー定義) を使用して、ワークフロー・インスタンスを作成します。
3. コンテナ内の PID および優先順位を設定します。
4. ワークフローを開始します。

次に、この手順を使用してワークフローを開始する例を示します。

Java

```
// ----- Create the DKWorkflowFed object and set the name  
DKWorkflowFed WF = new DKWorkflowFed(svWF);  
WF.setName("wf1");  
// ----- Create an instance of a workflow with the workflow template name  
WF.add("WD1");  
// ----- Refresh the workflow object  
WF.retrieve();  
// ----- Construct the container object for the workflow  
DKWorkflowContainerFed con = WF.inContainer();  
// ----- Retrieve the container data  
con.retrieve();  
// ----- Add a PID string referring to an Extended Search document  
con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");  
con.setPriority(100);  
// ----- Update the container  
con.update();  
// ----- Start the workflow  
WF.start(con);
```

C++

```
// - Create the DKWorkflowFed object and set the name
DKWorkflowFed* WF = new DKWorkflowFed(svWF);
WF->setName("wfl");
//Create an instance of a workflow with the workflow template name
WF->add("WD1");
// ----- Refresh the workflow object
WF->retrieve();
// ----- Construct the container object for the workflow
DKWorkflowContainerFed* con = WF.inContainer();
// ----- Retrieve the container data
con->retrieve();
// Add a PID string referring to a content item from Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Assign a priority of 100
con->setPriority(100);
// ----- Update the container
con->update();
// ----- Start the workflow
WF->start(con);
. . .
// When you are done, clean up by deleting the container and workflow
delete con;
delete WF;
```

ワークフローの終了

以下の例に示すように、`terminate()` または `del()` 関数を呼び出すことにより、ワークフローを終了することができます。

Java

```
//-----Retrieve the status of the workflow named WF
WF.retrieve();
int state =WF.state();
//-----Check the status and either terminate or delete
if (state ==DKConstantFed.DK_FED_FMC_PS_RUNNING ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDING)
{
    WF.terminate();
}
if (state ==DKConstantFed.DK_FED_FMC_PS_READY ||
    state ==DKConstantFed.DK_FED_FMC_PS_FINISHED||
    state ==DKConstantFed.DK_FED_FMC_PS_TERMINATED)
{
    WF.del();
}
```

C++

```
/--Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
//-----Retrieve the status of the workflow named WF
WF->retrieve();
int state = WF->state();
//---Check the status and either terminate or delete
if (state == DK_FED_FMC_PS_RUNNING ||
state == DK_FED_FMC_PS_SUSPENDED ||
state == DK_FED_FMC_PS_SUSPENDING)
{
WF->terminate();
}
if (state == DK_FED_FMC_PS_READY ||
state == DK_FED_FMC_PS_FINISHED ||
state == DK_FED_FMC_PS_TERMINATED)
{
WF->del();
}
delete WF;
```

すべてのワークフローのリスト作成

`listWorkFlows()` 関数を使用することにより、ワークフロー・サービスのすべてのワークフローをリストすることができます。以下の例では、`DKWorkflowServiceFed` オブジェクト `svWF` により参照されるワークフロー・サービスのすべてのワークフローの名前および記述をリストしています。

Java

```
// ----- Call the listWorkFlows method
DKSequentialCollection collWF = (DKSequentialCollection)svWF.listWorkFlows();
DKWorkflowFed WF = null;
if (collWF != null)
{
    dkIterator iterWF = collWF.createIterator();
    while (iterWF.more() == true)
    {
        WF = (DKWorkflowFed)iterWF.next();
        WF.retrieve();
        System.out.println("name = " + WF.getName() + " description = "
                           + WF.getDescription());
    }
    iterWF = null;
}
```

C++

```
// ----- Call the listWorkFlows function
DKSequentialCollection *collWF =
    (DKSequentialCollection*)svWF.listWorkFlows();
DKWorkflowFed *WF = NULL;
if (collWF != NULL)
{
    dkIterator *iterWF = collWF->createIterator();
    while (iterWF->more())
    {
        WF = (DKWorkflowFed*)(void*)(*iterWF->next());
        WF->retrieve();
        cout << "name = " + WF->getName()
        << " description = " << WF->getDescription() << endl;
        delete WF;
    }
    delete iterWF;
}
delete collWF;
```

ワークフローの延期

実行中のワークフローを、特定の時間または無期限に延期することができます。以下の例は、特定の時間までワークフローを延期する方法を示しています。

DKTimestamp にヌルを指定した場合、EIP はワークフローを無期限に延期します。

Java

```
// ----- Construct a DKWorkflowFed object
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ----- Call the suspend method if the workflow is in the running state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_RUNNING)
{
    // ----- Suspended until 2000-07-27-16.30.00.000000
    // ----- The timestamp uses the base year 1900; months are
    // ----- numbered 0 to 11
    DKTimestamp suspension = new DKTimestamp(100, 6, 27, 16, 30, 0, 0);
    WF.suspend(suspension);
}
```

C++

```
// ----- Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Call the suspend function if the workflow is in
the running state
if (WF->state() == DK_FED_FMC_PS_RUNNING)
{
    // ----- Suspended until 2000-07-27-16.30.00.000000
    DKTimestamp* suspension = new DKTimestamp(2000, 7,
27, 16, 30, 0, 0);
    WF->suspend(suspension);
    delete suspension;
}
delete WF;
```

ワークフローの再開

resume() 関数を呼び出すことにより、延期したワークフローを再開することができます。次に、延期したワークフローを再開する例を示します。

Java

```
// ----- Construct a DKWorkflowFed object
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ----- Call resume() if the workflow is in the suspended state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
    WF.resume();
}
```

C++

```
// ----- Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Check whether the workflow is suspended and call resume
if (WF->state() == DK_FED_FMC_PS_SUSPENDED)
{
    WF->resume();
}
delete WF;
```

すべてのワーク・リストのリスト作成

ワークフロー・サービスに listWorkLists() 関数を呼び出すことにより、ワークフロー・サービスのすべてのワーク・リストをリストすることができます。以下の例では、DKWorkflowServiceFed のインスタンス svWF により参照されるワークフロー・サービスのすべてのワーク・リストの名前および記述をリストしています。

Java

```
// ----- Call the listWorkLists method
DKSequentialCollection collWL = (DKSequentialCollection)svWF.listWorkLists();
DKWorkListFed WL = null;
if (collWL != null)
{
    dkIterator iterWL = collWL.createIterator();
    while (iterWL.more() == true)
    {
        WL = (DKWorkListFed)iterWL.next();
        WL.retrieve();
        System.out.println("name = " + WL.getName() + " description = "
                           + WL.getDescription());
    }
    iterWL = null;
}
```

C++

```
// ----- Call the listWorkLists function
DKSequentialCollection *collWL =
    (DKSequentialCollection*)svWF.listWorkLists();
DKWorkListFed *WL = NULL;
if (collWL != NULL)
{
    dkIterator *iterWL = collWL->createIterator();
    while (iterWL->more())
    {
        WL = (DKWorkListFed*)(void*)(*iterWL->next());
        WL->retrieve();
        cout << "name = " << WL->getName() << " description = "
              << WL->getDescription() << endl;
        cout << "Threshold = " << WL->getThreshold() << endl;
        delete WL;
    }
    delete iterWL;
}
delete collWL;
```

ワーク・リストへのアクセス

ワーク・リストにアクセスするには、システム管理クライアントを使用して作成したワーク・リストを参照する、DKWorkListFed のインスタンスを作成します。以下の例では、WL0712 というワーク・リストにアクセスし、そのワーク・リスト内に含まれている情報を表示します。

Java

```
// ----- Create the DKWorkListFed
DKWorkListFed WL = new DKWorkListFed(svWF, "WL0712");
WL.retrieve();
// ----- Display information about the worklist
System.out.println ("worklist name = " + WL.getName());
System.out.println ("description = " + WL.getDescription() +
    " owner = " + WL.getOwner() +
    " filter = " + WL.getFilter() +
    " threshold = " + WL.getThreshold() +
    " sort criteria = " + WL.getSortCriteria());
```

C++

```
// ----- Create the DKWorkListFed
DKWorkListFed* WL = new DKWorkListFed(svWF, "WL0712");
WL->retrieve();
// ----- Display information about the worklist
cout << "worklist name = " << WL->getName() << endl;
cout << "description = " << WL->getDescription() <<
    " owner = " << WL->getOwner() <<
    " filter = " << WL->getFilter() <<
    " threshold = " << WL->getThreshold() <<
    " sort criteria = " << WL->getSortCriteria() << endl;
// ----- Delete the worklist when you are done
delete WL;
```

作業項目へのアクセス

DKWorkListFed を作成したなら、作業項目をコレクションとして取得することができます。次に、作業項目を取得する例を示します。

Java

```
// ----- Create a collection and an iterator
DKSequentialCollection coll = (DKSequentialCollection)WL.listWorkItems();
dkIterator iter = (DKSequentialIterator) coll.createIterator ();
Object a;
DKWorkItemFed item;
String nodename;
String workflowname;

// ----- Step through the collections
while (iter.more ())
{
    a = iter.next ();
    item = (DKWorkItemFed) a;
    if (item != null)
    {
        item.retrieve ();
        nodename = item.name ();
        workflowname = item.workFlowName ();
        System.out.println ("workitem node = " + nodename +
                             " workflow name = " + workflowname);
    }
}
iter = null;
```

C++

```
DKSequentialCollection *coll;
dkIterator *iter;
DKWorkItemFed* item;
DKString nodename;
DKString workflowname;
// ----- Create a collection and an iterator
coll = (DKSequentialCollection*)WL->listWorkItems();

if (coll != NULL)
{
    iter = coll->createIterator();
    cout << "listWorkItems" << endl;
    // ----- Step through the collections
    while (iter->more ())
    {
        item = (DKWorkItemFed*)((void*)(*iter->next()));

        if (item != NULL)
        {
            //item.retrieve ();
            nodename = item->name();
            workflowname = item->workFlowName();
            cout << "workitem node = " << nodename
                 << " workflow name = " << workflowname << endl;
            delete item;
        }
    }
    delete iter;
    delete coll;
}
```

ワークフロー内の項目の移動

ワークフローが進展するにつれて、`checkOut()` および `checkIn()` 関数を使用して、あるアクティビティから次のアクティビティへ作業項目を移動します。次の例では、作業項目を移動する方法を示します。作業項目の実行を現在割り当てられているワークフロー・ユーザーだけが、その作業項目のチェックアウトとチェックインを行うことができます。

Java

```
DKWorkItemFed item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item.retrieve();
// ----- Call the checkOut method to lock the workitem
item.checkOut();
// ----- Call the checkIn method
item.checkIn(null);
```

C++

```
DKWorkItemFed* item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item->retrieve();
// ----- Call the checkOut method to lock the workitem
item->checkOut();
// ----- Call the checkIn method
item->checkIn(NULL);
delete item;
```

すべてのワークフロー・テンプレートのリスト作成

`listWorkFlowTemplates()` 関数を呼び出すことにより、ワークフロー・サービスのすべてのワークフロー・テンプレートをリストすることができます。以下の例では、`DKWorkFlowServiceFed` オブジェクト `svWF` により参照されるワークフロー・サービスのすべてのワークフロー・テンプレートの名前および記述をリストしています。

Java

```
// ----- Call the listWorkFlowTemplates method
DKSequentialCollection collWT =
    (DKSequentialCollection)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed WT = null;
if (collWT != null)
{
    dkIterator iterWT = collWT.createIterator();
    while (iterWT.more() == true)
    {
        WT = (DKWorkFlowTemplateFed)iterWT.next();
        WT.retrieve();
        System.out.println("name = " + WT.name() + " description = "
            + WT.description());
    }
    iterWT = null;
}
```

C++

```
// ----- Call the listWorkFlowTemplates function
DKSequentialCollection *collWT =
    (DKSequentialCollection*)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed *WT = NULL;
if (collWT != NULL)
{
    dkIterator* iterWT = collWT->createIterator();
    while (iterWT->more())
    {
        WT = (DKWorkFlowTemplateFed*)(void*)((*iterWT->next()));
        WT->retrieve();
        cout << "name = " << WT->name() << " description = "
            << WT->description() << endl;
        delete WT;
    }
    delete iterWT;
}
delete collWT;
```

ユーザー独自のアクションの作成 (Java のみ)

ワークフロー内で使用できる独自のアクションを作成することができます。アクションを定義し、Enterprise Information Portal 管理のアクション・リストに追加します。アクションを作成するには、アクション・オブジェクト (DKWorkFlowActionFed オブジェクト) を使用します。アクション・オブジェクトは、特定のタスクをクライアント・ノード側で実行する方法について、詳細な指示を記録したメタデータ・コンテナです。アクション・オブジェクト (メタデータ・コンテナ) は指示を記録するだけで、アクション・メタデータに記述されたタスクの呼び出しを開始することはありません。

アクションは、アクション・リスト (DKWorkFlowActionListFed) にグループ化できます。ワークフロー・コンテナは、アクション・リストの名前 (アクション・リストの内容でなく) を格納し、リスト内では作業項目に関係する一連のアクションが関連付けられています。クライアントは、アクション・リストを取得してリスト内のエントリー (アクション) を繰り返し、適切な応答をする必要があります。次のサンプルは、アクションとアクション・リストの使用法を示しています。このサンプルは、次のタスクを実行します。

1. 作業項目を取得します。
2. 作業項目と一緒に送られたコンテナを取得します。
3. コンテナからアクション・リストを取得します。
4. アクション・リストからアクションのリストを入手します。
5. リストに従ってアクションを開始します。

```
wit.retrieve(); // wit is a DKWorkItemFed object
DKWorkFlowContainerFed wcn = wit.inContainer();
wcn.retrieve();
String alName = wcn.getActionList();
DKWorkFlowActionListFed wal = new DKWorkFlowActionListFed(dsFed);
wal.setName(alName);
wal.retrieve();
dkIterator iter = null;
```

```
if ((coll!=null) && (coll.cardinality()>0))
{
    iter = coll.createIterator();
    while (iter.more())
    {
        DKWorkflowActionFed act = (DKWorkflowActionFed) iter.next();
        System.out.println("ACTION = " + act.getCommand());
        Runtime.getRuntime().exec(act.getCommand());
    }
}
else
    System.out.println("NO ACTION DEFINED");
```

非ビジュアルおよびビジュアル JavaBeans でのアプリケーションの構築

この章では、Enterprise Information Portal で提供されている非ビジュアル JavaBeans およびビジュアル JavaBeans について説明します。

EIP JavaBeans は、次のカテゴリーに分けることができます。

非ビジュアル Bean 非ビジュアル Bean を使用して、カスタマイズされたユーザー・インターフェースを必要とする Java アプリケーション、および Web クライアント・アプリケーションを作成できます。非ビジュアル Bean は、デフォルト・コンストラクター、プロパティ、イベント、およびシリアル化可能インターフェースを備えているので、標準の Bean プログラミング・モデルをサポートします。イントロスペクションをサポートするビルダー・ツールで、非ビジュアル Bean を使用できます。

ビジュアル Bean ビジュアル Bean は、カスタマイズ可能な Swing ベースのグラフィカル・ユーザー・インターフェース・コンポーネントです。ビジュアル Bean は、Windows 用の Java アプリケーションの作成に使用します。これらの Bean は、Java ベースのアプリケーションのウィンドウとダイアログ内に配置できます。ビジュアル Bean は非ビジュアル Bean を使用して (データ・モデルとして) 作成されているので、アプリケーションを作成する際には非ビジュアル Bean と組み合わせて使用する必要があります。

Java ビューアー関連 Bean Java ビューアー関連 Bean は、ビジュアル・コンポーネントと非ビジュアル・コンポーネントのセットです。これらの Bean を使用して、文書ビューアーを作成したり、文書を変換したりすることができます。Java ビューアー Bean は、eClient ビューアー・アプレット、および eClient の中間層の両方で使用されます。

Bean の基本概念について

JavaBeans (以後 Bean と呼びます) は、Java 言語で書かれた再使用可能なソフトウェア・コンポーネントで、Bean 対応のビルダー・ツールを使用して操作できます。Bean は再使用可能なので、複雑なコンポーネントを構成したり、新規アプリケーションを作成したり、既存アプリケーションに機能を追加したりするために使用できます。この操作はすべてビルダーを使用して視覚的に行うことも、プログラムから Bean メソッドを呼び出して手動で行うこともできます。

Bean は、基本的には Java クラスです。ほとんどすべての既存プログラミング・コンポーネントと Java クラスを、Bean にすることができます。一般に、プロパティとイベント・インターフェースの定義に関する特定の規則に準拠した Java クラスは、すべて Bean と見なすことができます。

Bean が定義する設計時インターフェースを使用して、アプリケーション設計ツール (ビルダー・ツール) がコンポーネントを照会し、これらのコンポーネントが定義しているプロパティの種類や、コンポーネントが生成したり応答したりするイベントの種類を判別できます。Bean を操作する際に、特別なイントロスペクション・

ツールや構築ツールを使用する必要はありません。パターン・シグニチャーが十分に定義されており、視覚的検査によってこれらのシグニチャーを容易に認識して理解できます。

Bean には次の特性があります。

イントロスペクション

イントロスペクションは、設計時および実行時に Bean がどのように動作するか、ビルダー・ツールが判別および分析するためのプロセスです。Bean は、メソッド・シグニチャーとクラス定義に関する定義済みのパターンを使用してコーディングされるので、これらのパターンを認識できるツールは、Bean を「のぞき込む」ことによってそのプロパティーと動作を判別できます。それぞれの Bean には関連した Bean 情報クラスがあり、このクラスは Bean 自体に関するプロパティー、メソッド、およびイベントの情報を提供します。それぞれの Bean 情報クラスは BeanInfo 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、アプリケーション・ビルダー・ツールに公開する Bean 機能を明示的にリストします。

プロパティー

プロパティーは、Bean の外観と動作を制御します。ビルダー・ツールは、Bean をイントロスペクトしてプロパティーを調べ、これらのプロパティーを操作のために公開します。これにより、Bean のプロパティーを設計時に変更できます。

カスタマイズ

Bean の公開されたプロパティーは、設計時にカスタマイズできます。カスタマイズにより、Bean の外観や動作を変更できます。Bean は、プロパティー・エディターを使用したカスタマイズ、または特殊な高機能の Bean カスタマイザーを使用したカスタマイズをサポートしています。

イベント

Bean は、イベントを使用して他の Bean と通信します。Bean は、イベントを発生させる（つまり、Bean が別の Bean にイベントを送る）ことができます。Bean がイベントを発生させると、その Bean はソース Bean と見なされます。Bean はイベントを受け取ることもでき、この場合の Bean はリスナー Bean と見なされます。リスナー Bean は、対象とするイベントをソース Bean に登録します。ビルダー・ツールは、イントロスペクションを使用して、Bean が送るイベントと受け取るイベントを判別します。

パーシスタンス

Bean は、java.io.Serializable インターフェースをインプリメントすることにより、Java オブジェクトのシリアライゼーションを使用して、カスタマイズの結果として変更された状態を保管および復元します。例えば、アプリケーションがアプリケーション・ビルダーで Bean をカスタマイズすると状態が保管されるので、変更したプロパティーを後で復元できます。

メソッド

すべての Bean メソッドは、他の Java クラスのメソッドと同じです。Bean メソッドは、他の Bean から呼び出すことも、スクリプト言語によって呼び出すこともできます。デフォルトでは、Bean の public メソッドがすべてエクスポートされます。

ビルダーでの JavaBeans の使用

ここでは、IBM Websphere Studio Application Developer、および他のビルダーで JavaBeans を使用方法について説明します。

IBM Websphere Studio Application Developer 以外のビルダーを使用するには、ビルダーが Java 2 をサポートしていることを確認してください。次の説明で指定されている jar を追加するには、新しい jar ファイルの追加に関するビルダーの指示に従ってください。その後、cmb81.jar にある EIP Bean を追加するには、jar からの Bean の追加に関するビルダーの指示に従ってください。

重要: Bean を使用するには、Java JDK 1.3.1 以上が必要です。

CMBROOT¥Samples¥java ディレクトリーに、非ビジュアル Bean のコード・サンプルがあります。

IBM Websphere Studio Application Developer の使用

次のステップを完了すると、非ビジュアル Bean を使用して、Webphere Studio Application Developer にサーブレットおよび JSP ページを構築することができます。

1. Web アプリケーションに Web プロジェクトを作成します。
2. プロジェクトのプロパティー Java Build Path | Libraries に、以下の JAR ファイルを指定します。

```
¥CMBROOT¥lib¥cmb81.jar
¥CMBROOT¥lib¥cmbview81.jar
¥CMBROOT¥lib¥cmbsdk81.jar
¥CMBROOT¥lib¥esclisrv.jar
¥SQLLIB¥java¥db2java.zip
```

3. EIP サーブレットと JSP taglib を使用する予定の場合には、次のファイルを指定しなければなりません。

```
¥CMBROOT¥lib¥cmbservlet81.jar
¥CMBROOT¥lib¥cmbtag81.jar
```

タグ・ライブラリーについては、EIP の JSP taglib 用の taglib.tld ファイルも Web アプリケーションにインポートする必要があります。

- ¥CMBROOT¥lib¥taglib.tld を Web アプリケーションの webApplication¥WEB-INF ディレクトリーにコピーします。
- 以下を追加して、Web アプリケーションの webApplication¥WEB-INF¥web.xml ファイルに taglib を構成します。

```
<taglib>
    <taglib-uri>cmb</taglib-uri>
    <taglib-location>/WEB-INF/taglib.tld</taglib-location>
</taglib>
```

4. 上記のリストされた JAR は J2EE クラスを含むため、通常 ¥Program Files¥IBM¥Application Developer¥plugins ¥com.ibm.etools.websphere.runtime¥lib¥j2ee.jar にある J2EE を組み込む必要があります。

EIP Java Bean の起動

EIP レイヤー内の Bean は、2 とおりの方法で呼び出すことができます。パブリック・インターフェース (public メソッド) を使用して、Bean を直接呼び出すことができます。この場合は、エラー・イベントを示す明示的 Java 例外がスローされます。

セッション単位の Bean の機能呼び出すもう 1 つの方式は、要求イベントと応答イベントを使用して、このタイプの Bean のインスタンスを他の EIP Bean に連結することです。この方式を使用する場合には、次の点に注意してください。

- CMBConnection Bean は、接続要求イベントを listen し、接続応答イベントの発生によって応答します。
- CMBDataManagement Bean は、データ要求イベントを listen し、返答としてデータ応答イベントを発生させます。
- CMBSchemaManagement Bean は、スキーマ要求イベントを listen し、返答としてスキーマ応答イベントを発生させます。
- CMBQueryService Bean は、検索要求イベントを listen し、返答として検索応答イベントを発生させます。
- CMBWorkflowDataManagement Bean は、ワークフロー・データ要求イベントを listen し、返答としてワークフロー・データ応答イベントを発生させます。
- CMBWorkflowQueryService Bean は、ワーク・リスト要求イベントを listen し、返答としてワーク・リスト応答イベントを発生させます。

非ビジュアル Bean

EIP には、あらゆる Java アプリケーションの作成に使用できる非ビジュアル JavaBeans のセットが用意されています。非ビジュアル Bean は、Bean の規則に準拠する Java クラスのセットで、EIP と CM の Java コネクタ・クラスを使用して作成されます。通常はサープレットのビルド、または Java Server Pages (JSP) で使用しますが、コマンド行や Windows アプリケーションでも使用できます。

非ビジュアル Bean を使用する利点は次のとおりです。

- EIP に付属するさまざまなコネクタのために、統合アクセス・メカニズムと共通のプログラミング・モデルを備えています。
- 高い抽象レベルでのプログラミングが可能です。
- 個々のコネクタの複雑な詳細を意識せずに済みます。
- ほとんどの市販開発環境に組み込まれている Bean サポートを利用できます。

Bean を使用すると、基本的なアプリケーションの作成が容易になります。ただし、Bean の使用を開始する前に認識しておく必要がある制限もいくつかあります。Bean の制限は次のとおりです。

- Java API レイヤーにある機能の一部を提供しません (例: 管理機能、構成機能)。
- バッチ・インポートをサポートしません。Bean は、一時的な単一項目タイプのインポート機能のみをサポートします。大量のデータをインポートおよびエクスポートするためのバッチ・プロセスは、コネクタ・インターフェースを使用して作成する必要があります。

- すべてのサーバーがサポートする、すべての機能を提供することはできません。一部の機能はサーバー固有です。例えば、項目内のサブ項目に関連した機能は、Content Manager バージョン 8 のみで利用できる機能に依存します。

上に示した制限は、場合によっては、基礎の Java API へのアクセスを可能にする accessor メソッドを使用して回避できます。

重要: EIP Bean は、Enterprise Java Beans (EJB) ではありません。このため、IBM Websphere などのコンテナが提供する管理環境内で、これらの Bean を直接ホストすることはできません。ただし、非構造化データ・リポジトリへの低レベル層接続メカニズムとして、これらの Bean を EJB 内部から使用することはできます。

非ビジュアル Bean の構成

非ビジュアル Bean には、ローカル構成、リモート構成、および動的構成があります。

local コンテンツ・サーバーに直接接続します。

remote

RMI サーバーを使用して、コンテンツ・サーバーに接続します。

dynamic

cmbcs.ini ファイルに基づいて、ローカルとリモートを動的に切り替えるアプリケーションを使用可能にします。cmbcs.ini ファイルでは、コンテンツ・サーバーがローカルかリモートかを指定します。

非ビジュアル Bean の機能の概要

EIP Bean のプロパティは通常、ストリングや配列などの単純なタイプなので、JSP 内で EIP Bean を使用できます。基本的に、これらの Bean はモデル・ビュー・コントローラー (MVC) 設計パターンを使用してモデル化されるので、Web アプリケーションのモデル・コンポーネントとして機能します。通常、ビュー・コンポーネントは、JSP とサーブレット (EJB サブレット・キット内のサーブレットのような) のコントローラー・コンポーネントで構成されます。EIP 非ビジュアル Bean の機能は、次のとおりです。

- ライブラリー・サーバー内のスキーマ定義へのアクセスを可能にします。
- EIP がサポートするすべてのリポジトリ内にある文書、単純 (非リソース) 項目、およびリソース項目に対する CRUD (create、retrieve、update、delete) メソッドを備えています。
- EIP がサポートするすべてのリポジトリ内にある文書、単純 (非リソース) 項目、およびリソース項目を検索して取得する機能を備えています。
- データ・タイプを表示可能なフォーマットに変換する処理をサポートします。
- EIP がサポートするさまざまなコンテンツ管理リポジトリにわたって、一貫した意味体系を適用する統合レイヤーとして機能します。
- EIP ワークフローと情報マイニング・サービスに備わっている機能を統合して公開します。
- 文書の抽出サービスと変換サービス、および文書注釈管理のサポートを備えています。
- ソート機能と変換機能を備えています。

- 構成 Bean に対して行われる主なアクションに応じて発生するイベントを提供します。例えば、接続イベントと切断イベント、検索結果通知イベント、コンテンツ変更通知イベントなどがあります。

非ビジュアル Bean のカテゴリー

非ビジュアル Bean は、次のカテゴリーに分けることができます。

データ・ストア Bean

これらの Bean は標準的なユーザー・セッションの期間中に存在し、特殊サービスをユーザーに提供します。セッション単位の Bean には、次のものがあります。

- **CMBConnection** この Bean は、バックエンド・サーバーへの接続を維持します。バックエンド・サーバーは、ネイティブ・コンテンツ・サーバーや統合サーバーなどです。この Bean は、すべての JavaBeans を使用するために必要です。
- **CMBSchemaManagement** リポジトリ・メタデータの操作に使用されます。
- **CMBDataManagement** リポジトリ・データの操作に使用されます。
- **CMBQueryService** および **CMBSearchResults** 照会の実行と照会結果の操作に使用されます。
- **CMBWorkflowDataManagement** および **CMBWorkflowQueryService** EIP の高度なワークフロー・プロセスの操作に使用されます。
- **CMBDocRoutingDataManagement** および **CMBDocRoutingQueryService** CM v8 文書ルーティング・プロセスの操作に使用されます。
- **CMBDocumentServices** 文書のストリーミング・サービスと注釈サービスに使用されます。

ヘルパー Bean

ヘルパー Bean は、1 つ以上のセッション単位 Bean のコンテキスト内に存在し、主にデータ値のカプセル化とセッション単位 Bean へのサービスの提供に使用されます。使用できるヘルパー Bean には、次のものがあります。

- **CMBEntity** コンテンツ管理リポジトリ内にあるデータ項目の定義を表します。例えば CM v8 リポジトリの場合、CMBEntity は項目タイプと子コンポーネント定義の両方を表しますが、CM v7 リポジトリの場合、CMBEntity は索引クラスを表します。CMBEntity は CMBSchemaManagement のヘルパー・クラスです。
- **CMBAttribute** リポジトリ内の属性定義を表します。CMBAttribute は CMBSchemaManagement のヘルパー・クラスです。
- **CMBSearchTemplate** 統合検索テンプレートを表します。CMBSearchTemplate は CMBSchemaManagement のヘルパー・クラスです。
- **CMBSTCriterion** 統合検索テンプレートの一部である検索基準を表します。CMBSTCriterion は CMBSchemaManagement のヘルパー・クラスです。

- **CMBItem** 文書、リソース項目、および非リソース項目のインスタンスを表します。CMBItem は CMBDataManagement のヘルパー・クラスです。
- **CMBObject** リソース項目、基本パーツ、およびメモ記録パーツのインスタンスを表します。CMBObject は、BLOB 属性の表現にも使用されます。CMBObject は CMBDataManagement のヘルパー・クラスです。
- **CMBAnnotation** CM v8 リポジトリの場合は注釈パーツ、OnDemand リポジトリの場合はメモのインスタンスを表します。
- **CMBPrivilege** CM または EIP がサポートするリポジトリから、特権に関連した情報を検索するために必要な機能を提供します。

補助 Bean

補助 Bean は、アプリケーションに不可欠ではありませんが、機能の拡張に便利なものです。次に補助 Bean のリストを示します。

- **CMBConnectionPool** CMBConnection Bean にプール・サービスを提供するために使用されます。CMBConnection インスタンスによって使用される DKDatastore インスタンスの維持には、Java API API クラス DKDatastorePool が使用されます。プールを Java API レベルに移すと、サーバーのタイプに応じてインテリジェントにサーバーをプールできるので、コンテンツ・サーバーをより高度に管理できます。
- **CMBUserManagement** ネイティブ・サーバー・ユーザーに対する統合ユーザーのマッピングを管理するために、統合リポジトリへの接続時に使用されます。
- **CMBExceptionSupport** 共通の例外イベント処理のフレームワークを提供します。
- **CMBTraceLog** 共通のトレース・イベント処理のフレームワークを提供し、他の Bean から発生したトレース・イベントのリスナー機能を備えています。

ワークフロー Bean

ワークフロー Bean は、ワークフロー・サービスを提供します。EIP Bean が提供するワークフロー・サービスは、拡張ワークフローと文書ルーティングの 2 種類のワークフロー・システムをサポートします。拡張ワークフロー機能は、MQSeries Workflow を使用して作成されます。一方、文書ルーティングは、CM V8 製品と API セットに組み込まれているワークフロー・システムです。Bean レイヤーは、ワークフロー定義の作成、作成した定義に基づくワークフロー・インスタンスの実行、および実行しているワークフローのインスタンスの管理に必要なオブジェクトをすべて提供します。次の Bean が、拡張ワークフロー・サポートの主要コンポーネントです。

- **CMBWorkflowDataManagement** この Bean は、拡張ワークフロー・インスタンスの作成と操作に使用されます。この Bean のインスタンスは、CMBConnection オブジェクトから取得できます。この Bean は、次のサポートを提供します。
 - ワークフロー・インスタンスの開始、終了、中断、および再開。
 - ユーザー間での作業項目と作業通知の転送。
 - 作業通知の取り消し。

- **CMBWorkflowQueryService** CMBWorkflowQueryService は、拡張ワークフローに関連した情報を照会するためのインターフェースを提供します。この Bean のインスタンスは、CMBConnection オブジェクトから取得できます。この Bean は、次の情報を取得するためのサポートを提供します。
 - システム内のワークフローの情報。
 - アクティブ・ワークフローの一部としてシステム内を移動する作業項目の情報。
 - ワーク・リストに関連した情報。
 - 登録されているすべての作業通知の情報。

次の Bean が、文書ルーティング・サポートの主要コンポーネントです。

- **CMBDocRoutingManagementICM** この Bean は、文書ルーティング・プロセスの作成と管理に使用されます。この Bean のインスタンスは、CMBConnection オブジェクトのインスタンスから入手できます。この Bean は、次の機能のサポートを提供します。
 - 文書ルーティング・インスタンスの開始、終了、中断、および再開。
 - 作業パッケージ内に含まれる CM 項目のチェックアウト。
 - 文書ルーティング・インスタンスによってルーティングされる作業パッケージのプロパティの設定。
- **CMBDocRoutingQueryServiceICM** この Bean は、文書ルーティング・プロセスに関連した情報を照会するためのインターフェースを提供します。この Bean のインスタンスは、CMBConnection オブジェクトのインスタンスから入手できます。この Bean は、次の情報を取得するためのサポートを提供します。
 - 文書ルーティング・システムによってルーティングされている作業パッケージに関連する情報。
 - 現在システム内でアクティブになっているプロセスに関連する情報。
 - システム内に存在するワーク・リストすべての情報。
 - システムに含まれる作業ノードすべて。

情報マイニング Bean

EIP 情報マイニング Bean を使用すると、アプリケーションにテキスト分析とマイニング・テクノロジーを組み込むことができます。情報マイニング Bean には、次の機能があります。

- テキストの要約とカテゴリー化 (文書要約の作成)。
- カテゴリー化 (文書へのカテゴリーの割り当て)。
- 文書から関係のある情報を抽出するためのサポート。
- 文書が書かれている言語を識別するためのサポート。
- 文書コレクション内で類似した文書をクラスター化するためのサポート。
- カタログ全体、または特定のカテゴリーの文書に制限した文書のテキスト検索。
- IBM Web Crawler を使用して継続的に Web から取り出される文書へのアクセスのサポート。

情報マイニング Bean には、次のものがあります。

- **CMBCatalogService** カタログに関連した情報を検索するためのインターフェースを提供します。また、情報マイニング操作によって作成された項目をコンテンツ・サーバーにインポートするためのサポートも提供します。すべての情報マイニング操作は、1つのカタログの範囲内で行われます。それぞれのカタログは分類法と関連しており、分類法はカテゴリーの階層で構成されます。
- **CMBAdvancedSearchService** カタログ内の情報のテキスト検索を実行するためのサポートを提供します。この Bean のメソッドを使用すると、検索の実行対象にするカタログや、テキスト検索から生成される結果の内容とサイズを制御できます。
- **CMBCategorizationService** この Bean は、指定のカタログに基づいて文書のカテゴリーを判別するために使用されます。
- **CMBSummarizationService** 文書の要約を生成するために必要な機能を提供します。
- **CMBClusteringService** コンテンツの類似性に基づいて、文書をクラスターにグループ化するために使用されます。
- **CMBWebCrawlerService** Web Crawler アクションの結果を管理するためのインターフェースを提供します。このインターフェースを使用して、特定の Web スペースに対する Web クロール要求を開始し、結果を管理できます。その後、作成された結果をカテゴリー化して要約し、バックエンドのコンテンツ・サーバーにインポートできます。

文書サービス Bean

文書サービス Bean は、文書ストリーミング・サービスと文書注釈サービスを提供します。文書サービスのサブセクションには、次の Bean が含まれます。

- **CMBDocumentServices** - CMBDocumentServices Bean は、文書のページのレンダリング、変換、再構成に必要な機能など、文書を操作するときに必要なサービスを提供します。
- **CMBDocument** - この Bean は、CMBDocumentServices を使用して文書をロードする際に作成されるエンティティを表します。基本的に、CMBDocument は文書内のページのコンテナです。また、CMBDocument を使用して、文書の特性を制御するプロパティを照会および設定することもできます。
- **CMBPage** - この Bean は、文書内の特定のページを表現します。このクラスの機能を使用して、ページに対して生成できるレンダリング可能なイメージのプロパティを指定および制御できます。
- **CMBPageAnnotation** この Bean は、文書内のページに関連付けることができる注釈をモデル化します。サポートされる注釈はすべて、この Bean のサブクラスによってモデル化されます。また、CMBPageAnnotation 自体は、注釈を付けるページや注釈タイプなどのプロパティを格納します。サブクラスには次のものがあります。
 - CMBArrowAnnotation
 - CMBCircleAnnotation
 - CMBHighlightAnnotation
 - CMBLineAnnotation

- CMBNoteAnnotation
- CMBPenAnnotation
- CMBRectAnnotation
- CMBStampAnnotation
- CMBTextAnnotation

その他の Bean クラス

ここにリストする Bean クラスは、次に説明するようにさまざまな機能を備えています。

- **BeanInfo クラス** BeanInfo インターフェースをインプリメントする別個の関連クラスに、EIP Bean の機能を明示的に公開できます。
- **例外クラス** Bean レイヤーの例外をカプセル化するために使用されます。例外クラスはすべて、基本クラス CMBException から継承されます。CMBException の個々のサブクラスは、特定のエラー条件を示します。それぞれのケースで、例外オブジェクトのプロパティを使用して、例外がスローされる原因になったエラー条件に関する詳細なエラー情報を入手できます。Bean をイベント・ドリブン方式で使用する場合、例外はイベント内でスローされます。
- **イベント・クラスとリスナー・クラス** 標準の Bean イベント・リスナー・モデルをインプリメントします。クラスは、リスナー・インターフェースをインプリメントし、イベントを生成するオブジェクトにそのリスナー・インターフェースを登録することによって、明示的にイベントを要求する必要があります。EIP Bean レイヤーは、スキーマ・アクセス操作、データ・アクセス操作、ワークフロー操作、および検索操作用に、イベントとリスナーの組を提供しています。
- **セッション・リスナー** これらは、セッション単位のレベルで存在するリスナー・クラスです。現在 EIP Bean には、接続要求イベントと接続応答イベントを追跡するセッション・リスナーが存在します。

非ビジュアル Bean を使用する際の考慮事項

非ビジュアル Bean を使用すると、EIP がサポートするコンテンツ管理リポジトリへのアクセスに必要な機能を備えた汎用アプリケーションを作成できます。ここでは、Bean の具体的な使用パターンについていくつかのヒントを示します。

Bean 内の Singleton CMBConnection には、他のセッション単位 EIP Bean のインスタンスにアクセスするためのメソッドがあります。

CMBSchemaManagement や CMBDataManagement などのセッション単位 Bean をこの方法で取得すると、これらの Bean はすでに CMBConnection Bean (取得元である) に連結されているので、接続や切断に関する通知を受けたり、トレースと例外のイベント・ハンドラーを共用したりすることができます。他のセッション単位 Bean のインスタンスは、Bean ごとに 1 つだけ作成されます。これらのメソッドが繰り返し呼び出される場合は、同じインスタンスが戻されます (singleton 設計パターン)。セッション単位 Bean をアプリケーション内で作成し、CMBConnection Bean によって作成しない場合は、使用する CMBConnection Bean にこれらの Bean を連結する必要があります。

Bean のスレッド化に関する考慮事項

CMBConnection Bean の 1 つのインスタンスは、一度に 1 つのスレッド上でのみ使用できます。この制限は、CMBConnection Bean に関連した (関連 Bean の接続プロパティによって) 他の Bean すべてに適用されます。このため、それぞれのスレッドごとに別個の接続を作成する必要があります。あるいは、

CMBConnectionPool Bean を使用すれば、複数のスレッドが接続を取得および解放できます。この場合は、それぞれのスレッドが接続を取得して使用し、解放する必要があります。

すべてのセッション単位 Bean には、取得元となった、または作成後に関連付けられた CMBConnection のインスタンスとの類縁性があります。このため、CMBSchemaManagement などのセッション単位 Bean のインスタンスを一度に使用できるスレッドは 1 つだけです。セッション単位 Bean インスタンスを複数のスレッドが使用する場合は、セッション単位 Bean インスタンスを一度に使用するスレッドが常にただ 1 つであるように、アプリケーション内で明示的な同期を実行する必要があります。

また、すべてのセッション単位 Bean は、CMBConnection Bean によって生成される接続応答イベントを listen します。これにより、CMBConnection Bean インスタンスが関連付けられている基礎のコンテンツ・リポジトリが変更されると、Bean はそのことを認識して適切なアクションを実行できます。

CMBConnection とは異なり、CMBConnectionPool Bean はマルチスレッド用に設計されています。複数のスレッドが、接続オブジェクトの取得と解放に関連したメソッドを同時に呼び出すことができます。プールから取得した接続は CMBConnection のインスタンスで、アクセスは単一スレッドに制限されます。接続プール Bean から取得した接続は、プールに接続を要求する他のスレッドが使用できるように、使用後はすぐにプールに戻す必要があります。

Bean のトレースとロギング

EIP Bean レイヤー内のセッション単位 Bean すべてを対象に、トレースを使用可能にすることができます。CMBConnection Bean のインスタンスに対するトレースを使用可能にすると、スキーマ管理 Bean、データ管理 Bean、照会サービス Bean、ワークフロー Bean など、この接続 Bean から取得した EIP Bean に対するトレースも使用可能になります。

トレースを使用可能にすると、トレース・イベントが発生します。ユーティリティー Bean の CMBTraceLog は、トレース・イベントを listen して、定義されたログ、stdout、stderr、またはウィンドウ (ビジュアル Bean と組み合わせて使用する場合) にトレース・レコードを書き込むことができます。

また、すべてのセッション単位 Bean はトレース・イベントを listen します。Bean のトレース機能は、Java API と同じログ・ファイルにロギング情報を書き込みます (ロギング用に log4j が使用されている場合)。

非ビジュアル Bean のプロパティとイベントについて

各非ビジュアル Bean は、下記のものを提供します。

- インポートされるプロパティ、拒否可能かどうか

プロパティ値は、実行時に `PropertyChange` または `VetoableChange` イベントにより、他の Bean によって決定されます。インポート・プロパティを持つ Bean は、`PropertyChange` または `VetoableChange` イベントを `listen` する必要があります。

- エクスポートされるプロパティ、拒否可能かどうか
非ビジュアル Bean は必須プロパティを持つ場合があります、他のいくつかの Bean がその値を必要とすることがあります。その値が変更された場合、Bean は必ず `PropertyChange` または `VetoableChange` イベントを生成しなければなりません。
- 独立型プロパティ
このプロパティ値を必要とするその他の Bean はありません。
- この Bean が生成するイベント
- この Bean が必要とするイベント

非ビジュアル Bean を使用したアプリケーションのビルド

非グラフィカル・ユーザー・インターフェース (GUI) アプリケーションのサンプル

この部分に示されている例は、非ビジュアル Bean を使用してサンプルの非 GUI アプリケーションを作成します。サンプル・アプリケーションには、`CMBUserManagement` Bean 以外のすべての Bean が含まれています。この例の元となった完全なサンプル・アプリケーション (`DemoSimpleApp1.java`) は、`Cmbroot/Samples/java/beans` ディレクトリにあります。このサンプル・アプリケーションは、下記のことを示しています。以下に行う方法を示します。

1. Enterprise Information Portal (統合) サーバーに接続する
2. 検索テンプレート名のリストを取得する
3. 検索テンプレート名を使用して、検索規準名のリストを取得する
4. 検索テンプレートを選択し、その検索基準を取得する
5. 検索値を入力し、照会を実行依頼する
6. 検索結果 Bean を使用した結果を出力する
7. 結果行を選択し、それを表示する
8. サーバーから切断する

ビジュアル Bean の使用

ビジュアル Bean を使用することによって、Enterprise Information Portal の機能または他のコンテンツ・サーバーを Swing に基づく Java アプリケーションに統合することができます。ビジュアル Bean は、ログオン、検索、結果の表示、文書の更新、およびバージョン情報の表示など、さまざまなアプリケーションに共通な基本タスクを実行します。

各ビジュアル Bean には `Connection` プロパティがあります。このプロパティは、コンテンツ・サーバーへの接続を管理する非ビジュアル Bean、`CMBCConnection` のインスタンスを参照しなければなりません。また、EIP ビジュアル Bean を使用して作成するアプリケーションはいずれも、`CMBCConnection` 非ビジュアル Bean のインスタンスを含んでいなければなりません。

CMBLogonPanel

この Bean は、Enterprise Information Portal または Content Manager バージョン 8.2 (CM 8.2) などのコンテンツ・サーバーにログインするためのパネルを表示します。また、統合ユーザーがコンテンツ・サーバー上のユーザー ID およびパスワードを変更するためのウィンドウも提供します。

CMBSearchResultsViewer

この Bean は検索結果を表示します。検索結果がフォルダーを戻す場合、CMBSearchResultsViewer Bean を使用してフォルダーを「ドリルダウン」し、その内容を見ます。検索結果の項目およびフォルダーは、Windows エクスプローラ・スタイルのウィンドウで表示するように選択してオープンできます。

CMBSearchTemplateList

検索テンプレートをサポートするサーバーの場合、この Bean は、使用可能な検索テンプレートのリストを表示して、テンプレートを選択できるようにします。

CMBSearchTemplateViewer

検索テンプレートをサポートするサーバーの場合、この Bean は、検索テンプレートを表示して、ユーザーが検索基準を入力するためのフィールドを提供します。それらの基準に基づき、検索を実行します。

CMBSearchPanel

すべてのサーバーの検索パネルでは、使用可能なエンティティのリストを表示しており、ユーザーが検索基準を入力するフィールドを提供します。それらの基準に基づき検索を実行します。CMBSearchPanel は、検索テンプレートをサポートしていないコンテンツ・サーバーで検索を実行する場合に役に立ちます。

CMBFolderViewer

Windows Explorer 形式のウィンドウに、1 つまたは複数のフォルダーの内容を表示します。

CMBItemAttributesEditor

ユーザーが索引クラスおよび項目の索引付け属性を更新するためのウィンドウを表示します。

CMBDocumentViewer

1 つまたは複数の文書に対して、それに適したビューアーを立ち上げて表示します。

CMBVersionsViewer

バージョン管理を使用できる場合は、文書のバージョン情報を表示します。

CMBLogonPanel bean

CMBLogonPanel Bean (448 ページの図 20 を参照) は、ユーザーがコンテンツ・サーバーにログインして、ユーザーのマッピングを更新し、パスワードを変更するためのウィンドウを表示します。

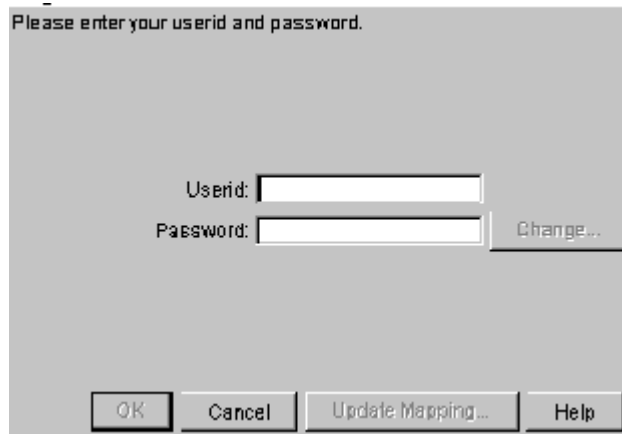


図 20. CMBLogonPanel bean ウィンドウ

CMBLogonPanel Bean において、「**変更 (Change)**」をクリックすると、「**パスワードの変更 (Change Password)**」ウィンドウが表示されます (図 21 を参照)。古いパスワードを入力した後、新しいパスワードを二度入力します。



図 21. 「パスワードの変更 (Change Password)」ウィンドウ

CMBLogonPanel Bean において、ユーザーが Logon ウィンドウの「**マッピングの更新 (Update Mapping)**」をクリックすると、「**ユーザー ID のマッピングの更新 (Update Userid Mapping)**」ウィンドウが表示されます (449 ページの図 22 を参照)。マッピングを更新すると、サーバーで指定しているユーザー ID およびパスワードが更新されます。この機能は、Enterprise Information Portal 統合データベースにログオンする場合にのみ使用可能です。

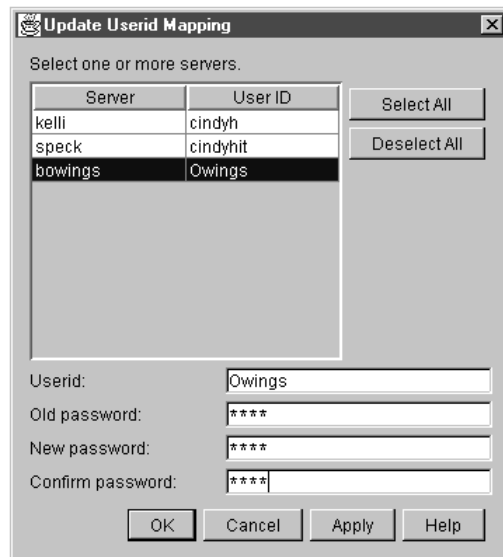


図 22. 「ユーザー ID のマッピングの更新 (Update Userid Mapping)」ウィンドウ

このウィンドウの上部には、すべてのサーバー、および対応するユーザー ID のリストが表示されます。このリストから 1 つまたは複数のサーバーを選択できます。すべてのサーバーを選択するには、「すべて選択 (Select All)」をクリックします。1 つまたは複数のサーバーを選択すると、ユーザーは新しいユーザー ID (およびオプションとしてパスワード) を指定できるようになります。1 つのサーバーだけを選択した場合、そのユーザー ID が「ユーザー ID (Userid)」フィールドに表示されます。複数のユーザー ID を選択した場合、「ユーザー ID (Userid)」フィールドはブランクになります。

「すべてを選択解除 (Deselect All)」

すべてのサーバーの選択を解除します。

「適用 (Apply)」

これをクリックすると、ウィンドウをクローズしないでマッピングおよびパスワードの変更が適用されます。

「OK」

これをクリックすると、変更を受け入れてウィンドウがクローズされます。

「キャンセル (Cancel)」

これをクリックすると、変更を適用しないでウィンドウがクローズされます。

CMBSearchTemplateList bean

CMBSearchTemplateList Bean には 3 つのスタイルがあります。イメージ・スタイル (450 ページの図 23) では、1 つのイメージを選択された項目の背景に使用し、もう 1 つのイメージを選択されていない項目の背景に使用します。450 ページの図 24 は、単純テンプレート・リスト・スタイルを示しています。450 ページの図 25 は、ドロップダウン・テンプレート・リスト・スタイルを示しています。

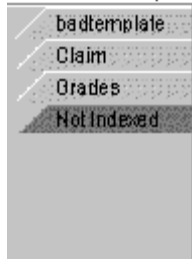


図 23. イメージ・テンプレート・リスト・スタイル



図 24. 単純テンプレート・リスト・スタイル

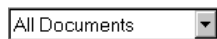


図 25. ドロップダウン・テンプレート・リスト・スタイル

CMBSearchTemplateViewer bean

CMBSearchTemplateViewer Bean (図 26) はウィンドウを表示し、ユーザーはそのウィンドウにおいて、システム管理者によって定義された検索テンプレートに従って検索規準を指定できます。CMBSearchTemplateViewer Bean は検索を開始して CMBSearchResults イベントを生成し、検索の結果を戻します。

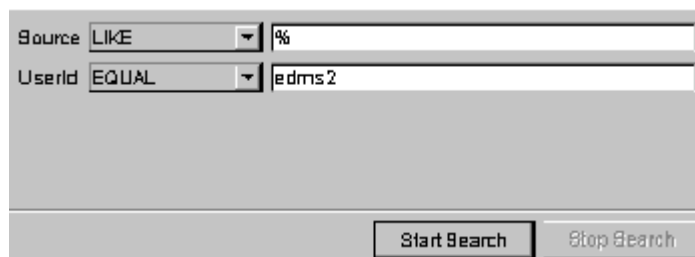


図 26. CMBSearchTemplateViewer Bean

CMBSearchTemplateViewer Bean は「ソース (Source)」、または「ユーザー ID (UserId)」などの検索規準をリストします。各検索基準には、ラベル、演算子、ドロップダウン・ボックス、およびテキスト・フィールドがあります。BETWEEN または NOTBETWEEN 演算子表示には、2 つのテキスト・フィールドがあります。IN または NOTIN 演算子には複数行のテキスト領域があります。各値は、別々の行に入力されなければなりません。

テキスト検索領域

CMBSearchTemplateViewer Bean は、ユーザーが完全テキストまたは索引属性に検索を実行できる領域を含めることができます。テンプレート上の完全テキスト検索領域は、テキスト・フィールドにラベルを付けるだけの単純なものにすることもできます。

照会ストリングをテキスト・フィールドに入力する場合は、フリーまたはブール・テキスト検索の照会構文に一致させなければなりません (DKDatastoreTS クラスを参照)。詳細については、「オンライン API 解説書」を参照してください。

CMBSearchTemplateViewer のフィールドの妥当性検査または編集

ユーザーが入力した検索基準を変更するために、CMBSearchTemplateViewer Bean に対する妥当性検査のロジックを用意することができます。これは、CMBTemplateFieldChangedEvent 用のハンドラーを提供することによって実行します。検索基準の現行値は、このイベントが呼び出される前に `getTemplate` メソッドによって戻された CMBTemplate に保管されます。この規準を検査して、変更することができます。イベント処理が終了すると新しい値が表示されます。

CMBSearchPanel bean

CMBSearchPanel Bean は、ウィンドウを表示し、ユーザーはそのウィンドウで現在のコンテンツ・サーバー上で使用可能なエンティティに従って検索基準を指定できます。CMBSearchPanel Bean は検索を開始して、CMBSearchResultsEvent を生成し、検索結果を戻します。CMBSearchPanel は、ウィンドウ上部のドロップダウン・リストにすべての使用可能なエンティティのリストを表示します。エンティティが選択されると、CMBSearchPanel は、そのエンティティの属性を表示します。各属性には、ラベル、演算子のドロップダウン・ボックス、およびテキスト・フィールドがあります。BETWEEN または NOTBETWEEN などの範囲演算子は 2 つのテキスト・フィールドを所有します。IN あるいは NOTIN 演算子など、複数の値を取る演算子は、複数行のテキスト域を持ちます。各値は、複数行のテキスト域で別々の行に入力しなければなりません。

CMBSearchResultsViewer bean

CMBSearchResultsViewer Bean は、ツリー・ペインと詳細ペインを含むウィンドウに検索結果を表示します。ペインを分ける線をクリックしてドラッグすると、ウィンドウのサイズを変更できます。

452 ページの図 27 は、「検索結果 (Search Results)」フォルダーが選択されている状態の CMBSearchResultsViewer Bean を示しています。

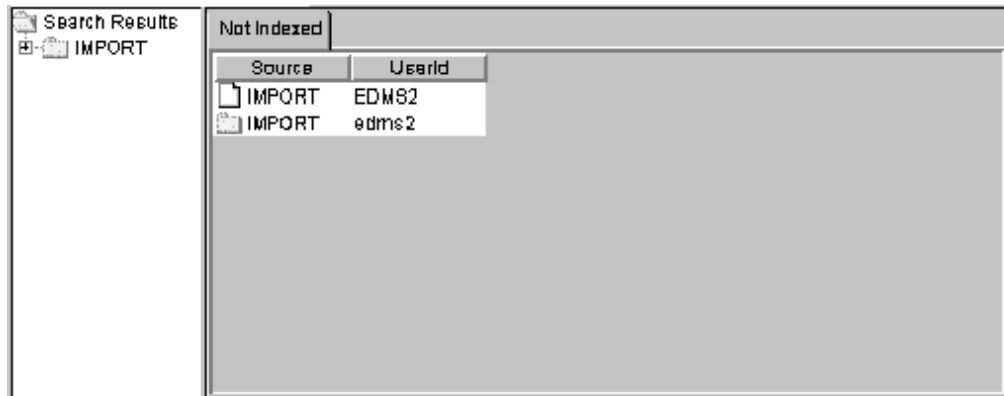


図 27. CMBSearchResultsViewer Bean

CMBSearchResultsViewer ツリー・ペイン

左側のツリー・ペインには、「検索結果 (Search Results)」というラベルが付けられているメイン・フォルダーが示されています。そのフォルダーの下には、検索で見つかった各フォルダーがあります。ツリー・ペインはオプションです。これは `setTreePaneVisible(false)` を使用して `TreePaneVisible` プロパティを設定することにより除去できます。

CMBSearchResultsViewer 詳細ペイン

詳細ペインには、ツリー・ペインで選択されたフォルダーの内容が表示されます。「検索結果 (Search Results)」フォルダーを選択すると、ノートブックにタブが表示され、そこに検索テンプレート名が示されます。「検索結果 (Search Results)」内の別のフォルダーを選択すると、1 つまたは複数のタブが表示されます。それぞれは、フォルダー内の各索引クラスを示しています。タブの名前の形式は、下記のとおりです。

index class @ server

ここで、*index class* は索引クラス名または項目タイプ名を、*server* はコンテンツ・サーバー名を表しています。索引クラスまたは項目タイプかに応じて、テーブルの列の属性の表示が変わります。詳細ペインでは複数選択がサポートされています。 `MultiSelectEnabled` プロパティを `setMultiSelectEnabled(false)` と設定することにより、複数選択をオフにすることもできます。項目タイプが階層的である場合には、子の属性値は子コンポーネント名/属性名のフォームの列ヘッダーを持つテーブルに表示されます。ここで、子コンポーネント名は子コンポーネントの名前、属性名は子コンポーネントの属性の名前です。例えば、項目タイプ *Journal* が *Author* という子コンポーネントを持ち、この子コンポーネント *Author* が属性 *Last Name* を持つ場合は、列ヘッダーは、*Author/Last Name* となります。

ポップアップ・メニュー

表の列見出しを右クリックするとポップアップ・メニューが表示されます。そのメニューにはソート・オプションがあります。「昇順でのソート (Sort Ascending)」をクリックすると、表内の項目が昇順でソートされます。「降順でのソート (Sort Descending)」をクリックすると、項目は降順でソートされます。ツリー・ペインの中の「検索結果 (Search Results)」フォルダー以外のフォルダーを右クリックするか、または詳細ペインの中の文

書かフォルダーを右クリックすると、別のポップアップ・メニューが表示されます。このポップアップ・メニューからツリー・ペイン中のフォルダーの詳細を表示したり、フォルダーの属性を編集したりできます。

オプション: CMBSearchResultsViewer Bean 内のフォルダーの詳細を表示するよりも、CMBViewFolderEvent を使用してください。選択したフォルダーの内容が CMBFolderViewer Bean によって表示されるようにするには、イベントを使います。

ダブルクリック・アクション

ツリー・ペイン内のフォルダー、または詳細ペイン内の項目をダブルクリックすることにより、「表示 (View)」ポップアップ・メニュー項目をクリックするのと同じアクションを実行することができます。デフォルトの項目ポップアップ・メニューを抑止すると、CMBItemActionEvent が発生します。

ポップアップ・メニューのオーバーライド

CMBSearchResultsViewer および CMBFolderViewer のポップアップ・メニューを、別のポップアップ・メニューまたはポップアップ・メニューなしにオーバーライドすることができます。デフォルトのメニューをオフにするには、setDefaultPopupMenu(false) を使用します。

ツリー・ペインでフォルダーを右クリックすると、CMBFolderPopupEvent が生成されます。詳細ペインで項目を右クリックすると、CMBItemPopupEvent が生成されます。ハンドラーを使用して、別のポップアップ・メニューを提供することができます。

CMBFolderViewer bean

CMBFolderViewer Bean は CMBSearchResultsViewer Bean と似たツリー・ペインを表示します。メイン「検索結果 (Search Results)」フォルダーはありません。454 ページの図 28 は、CMBFolderViewer Bean のツリー・ペインおよび詳細ペインを示しています。

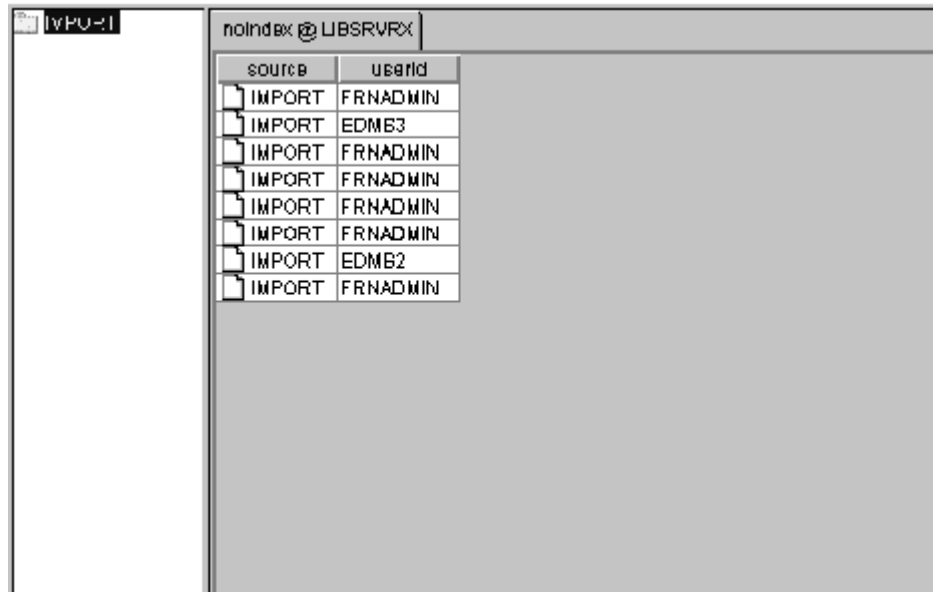


図 28. CMBFolderViewer Bean

CMBFolderViewer Bean は左側のペインにフォルダーのツリーを表示します。右側のペインには、ツリー・ペインで選択されたフォルダーに含まれる文書の表のノートブックが表示されます。サイズ変更が可能なスプリッターがツリー・ペインとノートブック・ペインを区切ります。

CMBFolderViewer ツリー・ペイン

ツリー・ペインにはフォルダーが含まれます。各フォルダーの下にネスト・フォルダーが表示されます。

CMBFolderViewer 詳細ペイン

詳細ペインには、ツリー・ペインで選択されたフォルダーの内容が示されます。この内容は、ノートブックに表示されます。ノートブックには、表中の各項目が索引付けされている各エンティティ（索引クラス、項目タイプ、その他）ごとにタブが設けられています。タブ名の形式は `index class @ server` です。ここで、`index class` は索引クラスの名前を、`server` はサーバーの名前を表しています。各ノートブック内のページは、選択したフォルダー内の文書およびフォルダーを表示する表になっています。表の列を変更すると、索引クラスに応じた属性が表示されます。

ポップアップ・メニュー

フォルダー・ビューアーのポップアップ・メニューの動作は、検索結果ビューアーの動作と同じです。

ダブルクリック・アクション

フォルダー・ビューアーでのダブルクリックは、検索結果ビューアーのダブルクリックと同じです。

CMBDocumentViewer bean

CMBDocumentViewer Bean は、コンテンツ・タイプ固有の文書ビューアーを立ち上げるか、または組み込むことによって文書を表示する機能を提供します。サポートされるビューアーには 2 つのタイプがあります。

1. Java ベースのビューアー。このビューアーはクラス CMBJavaDocumentViewer を拡張したものでなければなりません。
2. 非 Java ビューアー。特定のコンテンツ・タイプのビューアーとして、任意の実行可能プログラムを立ち上げることができます。

Visible プロパティが false に設定されている場合、ビューアーは常に別のウィンドウに表示されます。Visible プロパティが true に設定されていると、可能な場合には、CMBDocumentViewer Bean の表示領域内に表示されます。(現在これは Java ベースのビューアーでのみ可能です。)

CMBJavaDocumentViewer は、CMBDocumentViewer Bean にプラグインした Java ベースの文書ビューアーの提供者によって拡張された抽象クラスです。これらのビューアーは、CMBDocumentViewer Bean の可視スペースか、または画面上の別のウィンドウに文書を表示できます。

CMBDocumentViewer terminate() の呼び出しは、すべての文書のクローズされるイベントが処理されるまで待機します。文書のクローズされるイベント・ハンドラーから terminate() を呼び出すと、デッドロックが発生してプログラムがハングする場合があります。この問題を避けるには、onDocumentClosed(CMBDocumentClosedEvent) イベント・ハンドラーから terminate() を呼び出すときに、SwingUtilities.invokeLater(Runnable) を使用して CMBDocumentViewer.terminate() メソッドを呼び出します。これによって terminate() 呼び出しはイベント・キューの最後に追加され、終了メソッドの呼び出し前に、そのキュー内の他のイベント (他の文書のクローズされるイベント処理など) が継続処理されます。

ビューアーの指定

ビューアーを指定するには 2 つの方法があります。

1. EIP 管理で、MIME タイプとアプリケーションの関連付けエディターを使うことによってビューアーを指定します。これは、「ツール (Tools)」メニューの「**MIME - アプリケーション関連付けエディター (MIME to Appl. Editor)**」を選ぶことによって選択できます。Java ベースのビューアーの場合、アプリケーション名は Java クラス名にし、接尾部として **.class** を付けなければなりません。実行可能な場合、アプリケーション名は実行可能な名前にする必要があります。
2. CMBDocumentViewer の Mime2App プロパティを使用します。このプロパティを、MIME タイプをアプリケーション名にマッピングする Properties オブジェクトのインスタンスに設定することができます。

ビューアーが EIP Administration の MIME タイプ、および Mime2App の使用の両方に指定される場合、Mime2App の使用の指定が優先されます。

デフォルトのビューアー

特定のコンテンツ・タイプに対してビューアーが指定されていない場合、デフォルトのビューアーが立ち上がります。 OnDemand の文書の場合、 OnDemand クライアントが (表示のみのモードで) 立ち上がります。他のすべてのコンテンツ・サーバーの文書は、 Content Manager ビューアーを使用して表示されます。注釈を編集するには、ビューアーの「ファイル (File)」メニューの「文書の編集 (Edit Document)」を選択してください。

外部ビューアーの立ち上げ

アプリケーションを指定して、特定の MIME タイプの文書用の文書ビューアーとして立ち上げるには、 CMBDocumentViewer の Mime2App プロパティを使用します。プロパティ・オブジェクトを、 MIME タイプの名前を実行可能な名前である値にマッピングしている引き数として指定し、 setMime2App を使用してください。

CMBItemAttributesEditor bean

CMBItemAttributesEditor Bean (図 29 を参照) は、索引クラスの表示および変更と、フォルダーまたは文書の属性の索引付けを実行するウィンドウを表示します。

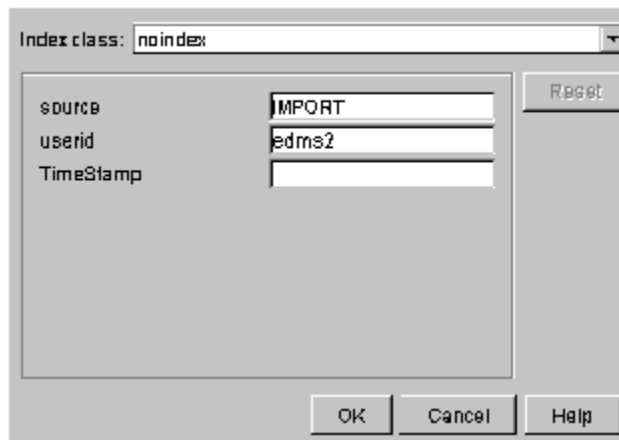


図 29. CMBItemAttributesEditor Bean

使用可能なすべてのエンティティを含むリストが、ウィンドウの上部に表示されます。デフォルトでは、現在のエンティティが選択されています。そのエンティティの属性のリストが、エンティティの下に表示されます。テキスト・フィールド (ファーストネーム、ラストネームなど) には、初期値としてその項目の現行値が入っています。

新しいエンティティを選択すると、直前に選択したエンティティと同じ名前の属性の値が、新しいエンティティの属性のうち名前が類似しているものに伝搬されます。

「リセット (Reset)」をクリックすると、エンティティおよび属性は元の値に戻ります。

「OK」をクリックすると、エンティティおよび属性が更新され、更新前および更新後にイベントが起動されます。このイベントを使用することにより、更新前にフィールドの妥当性検査を実行したり、更新が実行される前に完全でないフィールドを完成させることができます。このイベントは指定した更新を拒否できます。

CMBItemAttributesEditor での変更の拒否

CMBItemAttributesEditor には、ユーザーが入力した属性値を検査したり、それらを変更したり、値が無効な場合には更新を拒否したりするための付加的な妥当性検査ロジックを用意することができます。そのためには、CMBEditRequestedEvent 用のハンドラーを提供して、実行します。

CMBVersionsViewer bean

CMBVersionsViewer Bean は、単一文書または単一項目のバージョン管理属性のテーブルを表示します。表示されているバージョン管理属性には、バージョン番号、作成者のユーザー ID、バージョン作成時のタイム・スタンプ、最終更新者のユーザー ID、および最終更新のタイム・スタンプがあります。バージョン・ビューアーから、項目のさまざまなバージョンを表示したり、項目の属性を更新することができます。

ビジュアル Bean の一般的な動作

ここでは、ビジュアル Bean の共通のプロパティおよび動作について説明します。

プロパティ

ここでは、ビジュアル Bean によって共通に使用される 3 つのプロパティを説明します。

Connection

各 Bean には、CMBConnection 非ビジュアル Bean のインスタンスを参照する、Connection プロパティがあります。操作を正しく実行するためには、ビジュアル Bean にこのプロパティを設定しなければなりません。

CollationStrength

ソートを実行するすべての Bean には、CollationStrength プロパティがあります。CollationStrength プロパティで定義される値は、Java の java.text.Collator クラスで定義されている値と同じです。

ボタンの表示/非表示

すべてのビジュアル Bean 上で表示されるボタンは、表示または非表示にすることができます。setNameButtonVisible プロパティを使用します (Name はボタンの名前)。

保管/復元の構成

CMBSearchTemplateViewer、CMBSearchResultsViewer、および CMBFolderViewer には、loadConfiguration と saveConfiguration という 2 つのメソッドがあります。それらを使用することにより、アプリケーション・セッション間で、フィールド値および列サイズの保管および復元を実行できます。プロパティ・オブジェクト

トは、これらのすべてのメソッドで使用される引き数です。同じプロパティ・オブジェクトを 3 つの Bean すべてで使用できます。保管されるプロパティの名前は Bean 間で同一です。

ヘルプ・イベント

「ヘルプ (Help)」ボタンをクリックするか F1 を押すかしてヘルプを要求すると、各ビジュアル Bean は CMBHelp イベントを生成します。さらに、2 次ウィンドウから F1 または Help を押すと、以下のヘルプ関連のイベントを生成する Bean もあります。

CMBChangePasswordHelpEvent

「パスワードの変更 (Change Password)」ウィンドウで「ヘルプ (Help)」がクリックされた場合

CMBUpdateMappingHelpEvent

「マッピングの更新 (Update Mapping)」ウィンドウで「ヘルプ (Help)」がクリックされた場合

CMBLoginFailedHelpEvent

「サーバー・ログオン失敗 (Server Logon Failed)」ウィンドウで「ヘルプ (Help)」がクリックされた場合

CMBServerUnavailableHelpEvent

「使用不可サーバー (Server Unavailable)」ウィンドウで「ヘルプ (Help)」がクリックされた場合

ヒント: これらのすべてのソースからのヘルプを処理できる 1 つのメソッドは、これらのすべてのイベント用のリスナーを実装する 1 つのクラスを作成することです。onHelp メソッド内では、イベントのソースである Bean を判別し、その Bean に適したヘルプ・テキストを表示するための、追加の論理が必要です。

ビジュアル Bean の置換

ビジュアル Bean を別の Bean または Swing コンポーネントに置換することができます。そのためには、置換するビジュアル Bean のイベントのハンドラーを新しい Bean がインプリメントしなければなりません。また、置換を実行する Bean の少なくともキー・イベントを生成する必要があります。キー・イベントについて、表 31 に示します。

表 31. ビジュアル Bean およびキー・イベント

ビジュアル Bean	キー・イベント
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent
CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Event
CMBItemAttributesEditor	なし

Bean 関数を実装するのに必要なすべてのデータは、Bean が処理するイベントか、または CMBCConnection 非ビジュアル Bean のいずれかから入手できます。

ビジュアル Bean を使用したアプリケーションのビルド

ビジュアル Bean を使用して作成されたサンプル・クライアント・アプリケーションがユーザーのために提供されています。サンプルのソース・ファイルは、`<cmbrout>/samples/java/beans/gui` にあります。また、サンプル・クライアントの詳細とセットアップ要件について、このディレクトリーにある `readme.html` ファイルをお読みください。

以降のセクションでは、アプリケーションをビルドする際にビジュアル Bean をどのように組み合わせるかについて説明します。

ビジュアル Bean の接続

ここでは、ビジュアル Bean を接続し、単純なアプリケーションを作成するシナリオについて説明します。「**検索 (Search)**」ボタンを除いて、すべての Bean はソース Bean の示されているイベントのリスナーとしてターゲット Bean を追加することにより接続されます。例えば、`SearchTemplateList` を `SearchTemplateViewer` に追加する場合、1 行のコードが必要です。検索を立ち上げるためのボタンを追加するには、標準の `JButton` を使用します。ボタンのアクション・イベントから適当なメソッドを呼び出すための内部クラスを作成します。

図 30 の、各 Bean から接続 Bean への線は、その Bean に接続 Bean への参照が含まれていることを示しています。これは、各 Bean の接続プロパティーを設定することによって作成します。例えば、ログオン・パネル Bean から接続 Bean への参照を作成する場合、1 行のコードが必要です。

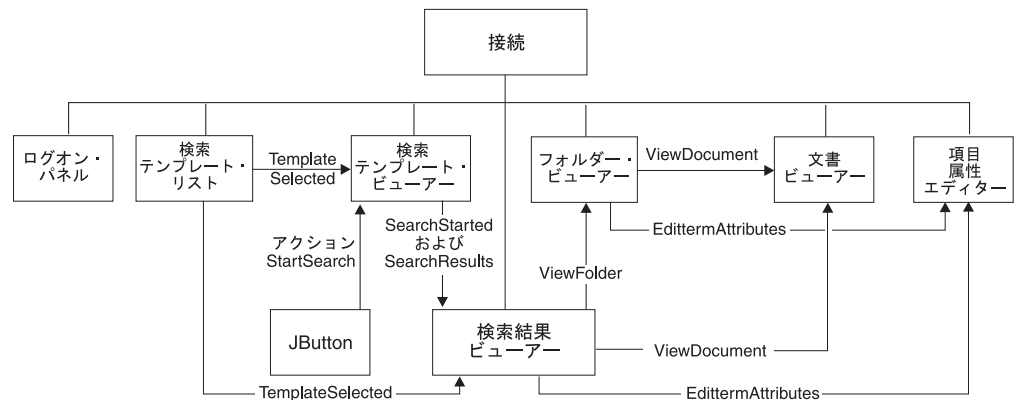


図 30. ビジュアル Bean の接続

図 30 は、9 つの Bean を示しています。JFrame または他のコンテナ Bean は、これらのすべての Bean の親になります。実行時のイベントの順序の例を以下に示します。

1. ユーザー ID およびパスワードをログオン・ウィンドウに入力し、「OK」をクリックします。サーバーへの接続を確立するために、`CMBLogonPanel` Bean が `CMBConnection` Bean の `接続 (connect)` メソッドを呼び出します。
2. 接続 Bean が接続を確立します。`CMBSearchTemplateList` Bean がそのユーザー ID の検索テンプレートのリストを検索し、表示します。(そのためにメソッドを呼び出す必要はありません。`CMBSearchTemplateList` Bean は、`CMBConnection` Bean の適当なイベントを `listen` しています。`CMBConnection`

Bean が `setConnection` メソッドを使用して、その Bean 自体を `CMBSearchTemplateList` に関連付けている場合、その `CMBSearchTemplateList` はリスナーをセットアップします。)

3. リストから検索テンプレートを選択します。`CMBSearchTemplateList` Bean は `CMBTemplateSelectedEvent` を生成します。`CMBSearchTemplateViewer` と `CMBSearchResultsViewer` は両方とも、そのイベントを `listen` します。`CMBSearchTemplateViewer` は適当なテンプレートを表示します。`CMBSearchResultsViewer` は、テンプレートで定義されているように、詳細ペイン内の列を消去し、表示します。
4. ユーザーはテンプレートを完成し、Enter キーを押すか「**検索 (Search)**」をクリックします。「**検索 (Search)**」をクリックすると、アクション・イベント・ハンドラーが `startSearch` メソッドを呼び出します。ユーザーが Enter キーを押すと、`startSearch` メソッドが暗黙のうちに呼び出されます。
5. `CMBSearchTemplateViewer` Bean がテンプレート・フィールドの妥当性検査を実行し、検索を開始できるかどうかを判別します。検索を開始できる場合、`CMBSearchStartedEvent` が生成されます。`CMBSearchResultsViewer` はこの `CMBSearchStartedEvent` を `listen` しており、新しい検索結果の準備として結果を消去します。
6. 検索の進行中、`CMBSearchResultsEvents` が生成され、部分的な検索結果が `CMBSearchResultsViewer` に提供されます。(検索が完了すると、`CMBSearchCompleted` イベントが生成されます。検索の開始時に、「**検索 (Search)**」ボタンが使用不可の場合、このイベントを使用して再び使用可能にできます。)
7. ユーザーは「**検索結果 (Search Results)**」ウィンドウ内のフォルダーを展開して、文書またはフォルダーを選択し、表示することができます。その場合、`CMBViewFolderEvent` または `CMBViewDocumentEvent` が生成されます。`CMBFolderViewer` および `CMBDocumentViewer` Beans はそれぞれのイベントを `listen` しており、フォルダーまたは文書を表示します。
8. `CMBFolderViewer` から、ユーザーは表示する文書を選択できます。表示する文書を選択すると、`CMBViewDocumentEvent` が生成されます。`CMBDocumentViewer` はこのイベントを `listen` しており、適切なビューアでその文書を表示します。
9. 文書またはフォルダーの属性を選択し、`CMBSearchResultsViewer` または `CMBFolderViewer` から更新できます。文書を選択すると、`CMBEditItemAttributesEvent` が生成されます。
10. `CMBItemAttributesEditor` Bean は `CMBEditItemAttributesEvent` を `listen` しています。これは、その項目のエンティティおよび属性を表示します。そのエンティティおよび属性を変更し、「**OK**」をクリックすると、変更が適用されます。

複数のウィンドウまたはダイアログでの Bean の使用

あるウィンドウの Bean から別のウィンドウの Bean へイベントを渡すには、コードを追加しなければなりません。通常、イベントが送られることによってそのウィンドウが表示されます。`EditAttributesDialog` ウィンドウには `ItemAttributesEditor` が含まれます。`CMBEditItemAttributesEvent` が立ち上がると、`SearchFrame` がウィンドウを作成します。


```
// Invoke a secondary dialog for edit attributes
searchResultsViewer.addEditItemAttributesListener(new
CMBEditItemAttributesListener() {
    public void onEditItemAttributes(CMBEditItemAttributesEvent event) {
        EditAttributesDialog editAttributesDialog = new
        EditAttributesDialog(SearchFrame.this,connection,event.getItem());
        editAttributesDialog.setVisible(true);
    }
});
```

通常 CMBItemAttributesEditor Bean に渡される情報が、代わりに引き数としてこのウィンドウのコンストラクターに渡されます。そのコンストラクターでは、下記のプロパティーを設定することにより情報が CMBItemAttributesEditor Bean に渡されます。

```
itemAttributesEditor.setConnection(connection);
```

```
itemAttributesEditor.setItem(item);
```

Java 文書ビューアー・ツールキットの使用

文書ビューアーを使用すると、ユーザーのコンテンツ・サーバーに含まれる文書にアクセスし注釈を付けることができます。EIP Java ビューアー・ツールキットを使用してカスタム文書ビューアーを作成することができます。また、EIP またはスタンドアロンのアプリケーションに統合できる、カスタム・ビューアーのアプレットおよびアプリケーションを作成することもできます。

重要: *viewdata* オプションは、OnDemand バックエンドの場合はサポートされません。

Java ビューアー・ツールキット・クラスには、以下の機能を提供するアクション・オブジェクトが含まれています。

- ページ表示オプション
 - 文書の回転: 右回りに 90 度、左回りに 90 度、および 180 度
 - ズーム: 拡大および縮小
 - スケール: 25%、50%、100%、150%、200%、および 400%
- 反転
- 拡張
- 印刷
- 現行の文書のクローズ
- 全文書のクローズ
- 注釈の作成と編集
 - 書き込む
 - 強調表示する
 - 枠を描く
 - 円を描く
 - 線を描く
 - 矢印を描く
 - テキストを追加する
 - スタンプ
 - 注を追加する
 - 消去する
 - 隠すまたは表示する
 - デフォルトのカーソル・モード
 - 注釈を前面に出す
 - 注釈を背面に送る
 - 注釈プロパティを変更する
 - 注釈操作を取り消す、やり直す
 - 注釈を切り取る、コピーする、貼り付ける、削除する

- 文書を保管する (注釈のみが保管される)
- 文書または文書内のページのナビゲート
 - ページのナビゲート: 先頭ページ、前のページ、指定したページ、次のページ、最終ページ
 - 文書のナビゲート: 最初の文書、前の文書、指定した文書、次の文書、最後の文書
- サムネール
 - サムネールを隠すまたは表示する
 - サムネールを使用したページ・ナビゲーション
 - パンとズーミング

Java ビューアー・ツールキットは、Swing ベースのアプリケーションの作成に使用できる多数の GUI クラスを備えています。また、Swing ベースでない文書表示アプリケーション用に使用できる非 GUI クラスも備えています。

ビューアー・アーキテクチャー

Java ビューアー・ツールキットには、文書ビューアーおよび文書サービスの Bean が含まれています。これらの Bean は、ビューアーを EIP ベースのアプリケーションへ統合する機構を提供します。

ツールキットには、汎用文書ビューアー (Generic Doc Viewer)、ストリーミング文書サービス (Streaming Doc Services)、および注釈サービス (Annotation Services) のクラスも含まれています。これらのクラスによって、Bean を使用できないアプリケーション、例えばスタンドアロン表示、またはコンテンツ・ストアへの接続がローカルではない分散処理状況などでビューアーを使用することができるようになります。

ストリーミング文書サービスは、文書を解析し、ページをレンダリングする文書処理エンジンのセットを管理します。これにより文書ページの操作が可能になります。これらのエンジンは、テキスト、リッチ・テキスト文書、およびオフィス・フォーマットはもちろん、TIFF および IOCA など、さまざまなフォーマットの文書イメージの構文解析を実行します。また、追加の文書エンジンを作成して、それらをビューアー・ツールキット・アーキテクチャーへ接続させ、これ以外のフォーマットをサポートさせたり、文書フォーマット用の代替レンダリングをサポートすることもできます。

注釈サービス・クラスを使用すると、注釈を取り扱うことができます。注釈エンジンは、Content Manager の特定の注釈フォーマットを解析します。追加の注釈エンジンを作成できます。

文書エンジン

EIP とともに提供される文書エンジンには、以下の 4 つがあります。

- MS-Tech 文書エンジン: このエンジンは、ページをイメージとしてレンダリングしながら、一般に IBM Content Manager 上に検出されるコンテンツ・タイプを取

り扱います。サポートされる文書タイプには、TIFF および MO:DCA が含まれます。このエンジンは、また GIF、JPEG、およびプレーン・テキストもサポートします。

- INSO 文書エンジン: このエンジンは Microsoft Office、Lotus SmartSuite、およびその他のオフィス文書フォーマットもサポートします。
- AFP2Web 文書エンジン: このエンジンは AFP を理解し、AFP 文書を HTML または PDF へ変換します。
- Java 文書エンジン: このエンジンは、URL にある文書を、その URL への転送リンクを含んだ HTML に変換します。このエンジンはまた、XSLT を呼び出すことによって、XML を HTML に変換します。

これらのエンジンは共通インターフェースですが、これらへ直接プログラムを組まないでください。その代わりに、文書サービス Bean またはストリーミング文書サービス・クラスによって提供されるインターフェースを使用してください。

これらのエンジンの中には、純正の Java ではなく、移植制限があるものがあります。これにより、いくつかのプラットフォーム上でツールキットの使用が制限される場合があります。MS-Tech エンジンおよび Java エンジンは、純正の Java です。他のエンジンには、Windows プラットフォームに対して使用を制限するプラットフォーム特定ロジックが含まれます。

注釈エンジン

EIP は Content Manager 注釈を処理するために MS-Tech 注釈エンジンを提供しています。MS-Tech エンジンは、Content Manager バージョン 8.2、Content Manager バージョン 8.1、Content Manager バージョン 7、および VI/400 の注釈をサポートします。

汎用文書ビューアーの作成

汎用文書ビューアーを作成するには、最初に、CMBStreamingDocServices インターフェースを使用する CMBGenericDocViewer クラスを使用します。

CMBStreamingDocServices は文書をロードし、レンダーします。

CMBStreamingDocServices は、一連の文書エンジンを使用して、TIFF や MO:DCA など、さまざまな文書フォーマットを変換します。文書内の注釈をロード、編集、および保管するには、CMBAnnotationServices インターフェースを使用します。

汎用文書ビューアーのカスタマイズ

cmbview81.jar ファイルにあるデフォルトの構成ファイル

CMBViewerConfiguration.properties をカスタマイズするか、または新規構成ファイルを作成しても構いません。構成ファイルを作成する場合も

CMBViewerConfiguration.properties をカスタマイズする場合も、同じファイル名を保持し、それをクラス・パス内の cmbview81.jar の前に配置する必要があります。

構成ファイルを作成するには、以下のステップを実行します。

1. ツールバー名によって指定された各ツールバーに、以下の項目を指定する必要があります。このステップでは、メインフレーム上のツールバーの位置を指定します。デフォルトの位置は、NORTH です。リストされた各ツールバーに位置を指定します。

```
Toolbars=[<toolbar_name>[,<toolbar2_name>][,<toolbar3_name>]...]
<toolbar_name>.position={NORTH|SOUTH|EAST|WEST}
```

2. 指定されたツールバーへ追加するアクションを指定します。ツールバー上のアクションとアクションの間の区切り文字として、',' を使用します。

```
<toolbar_name>.tools=[<action_name>[,<action2_name>]
[,<action3_name>]...]
```

```
<action_name>.label=<action_label>
<action_name>.tooltip=<action_tooltip>
<action_name>.icon=<icon_file_name>
<action_name>.key=<key_code>
<action_name>.cursor=<cursor_file>
<action_name>.hotspot=<x,y>
```

3. 新規のポップアップ・メニューは追加できません。ただし、サブメニューおよびメニュー項目を、3 つの事前定義ポップアップ・メニューへ追加できます。

```
<popup_menu_name>.items=[<menuitem_name>[,<menuitem2_name>]
[,<menuitem3_name>]...]
<popup_menu_name>.submenu=[<submenu_name>[,<submenu2_name>]
[,<submenu3_name>]...]
<submenu_name>.label=<submenu_label>
```

すでに適切な構成ファイルを指定してある場合は、いつでもスタンドアロン・ビューアーのアプリケーションまたはアプレットを作成できます。以下のステップを実行するときは、「オンライン API 解説書」を参照してください。

1. CMBStreamingDocServicesCallbacks をインプリメントするプライベート・クラスを作成します。
2. 最初のステップでインプリメントしたコールバックを持つ CMBStreamingDocServices を作成します。
3. CMBAnnotationServicesCallbacks をインプリメントするプライベート・クラスを作成します。
4. インプリメントしたコールバックを持つ CMBAnnotationServices を作成します。
5. CMBGenericDocViewer のインスタンスを作成し、それを CMBStreamingDocServices、CMBAnnotationServices、および構成プロパティ・ファイルを使用して初期化します。デフォルトの構成ファイルを使用したい場合は、プロパティ・ファイルにヌルを渡してください。
6. CMBGenericDocViewer で loadDocument() を呼び出して、文書をロードします。これにより、CMBDocument インスタンスが戻されます。
7. CMBGenericDocViewer で loadAnnotationSet() を呼び出して、任意の注釈をロードします。CMBAnnotationSet オブジェクトが戻されます。CMBAnnotationServices で メソッド setItemHandle(CMBAnnotationSet, CMBItem) を使用します。
8. ビューアーが提供するさまざまなメソッドを使用して、位置、サムネールのサイズ、MDI または SDI ビューなど、ビューアーのルック・アンド・フィールをカスタマイズします。

9. アプリケーションまたはアプレットのメインフレームにビューアーを追加します。
10. 汎用文書ビューアーからアクションを検索してメニュー・バーを作成し、そのメニューをアプリケーションのメインフレームに追加します。
11. 汎用文書ビューアーで `showDocument()` を呼び出して、文書を表示します。
12. 文書の注釈を保管するには、`CMBGenericDocViewer` で `saveAnnotations(CMBDocument)` を呼び出します。
13. 単一文書をクローズするには `closeDocument(CMBDocument)` を、全文書をクローズするには `closeAllDocuments()` を呼び出します。

図 31 は、すべてデフォルト設定を使用した汎用文書ビューアーの例を示しています。



図 31. 汎用文書ビューアー

アプリケーション例

このセクションでは、Java 文書ビューアー・ツールキットをより良く理解するために、ユーザーが作成できるアプリケーションの例を 5 つ挙げて説明します。ツールキットの使用法は以下に挙げた例に限りません。

スタンドアロン・ビューアー

汎用文書ビューアーを使用して、スタンドアロン・ビューアーをインプリメントすることができます。スタンドアロン・ビューアーを使用すると、URL から検索するファイルまたは文書を表示することができます。スタンドアロン・ビューアーは、Web ページ上のパレットとして、または E メールを通じて入手した文書の表示などのために使用できます。図 32 は、スタンドアロン・ビューアーのアーキテクチャーを示しています。

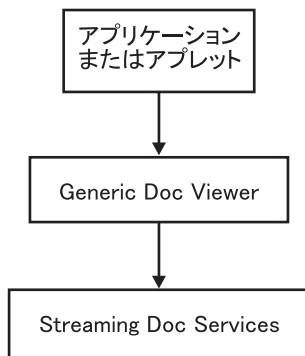


図 32. スタンドアロン・ビューアー

Java アプリケーション

実動 Java アプリケーションを作成するには、CMBDocumentViewer ビジュアル Bean を使用する EIP ビジュアル Bean を使用します。この Bean は、文書表示のために、内部で汎用文書ビューアーを使用します。CMBDocumentViewer Bean は、文書表示のために他のビューアーを立ち上げることもできます。ただし、これらのビューアーは、プラットフォームに依存する可能性があるので注意してください。図 33 は、Java アプリケーション作成のために使用できるアーキテクチャーを示しています。

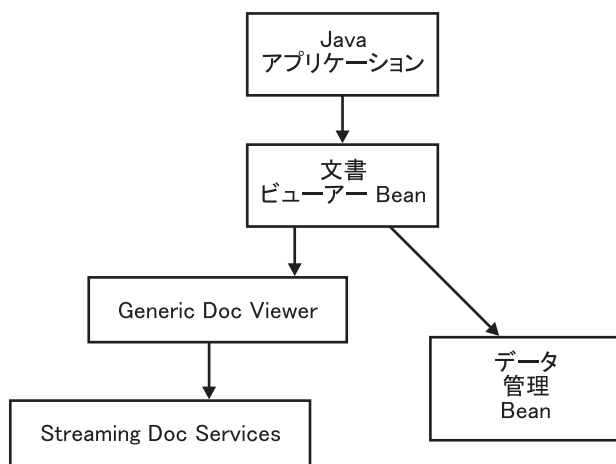


図 33. Java アプリケーション

シン・クライアント

CMBDocumentServices Bean を使用して、Web ベースのアプリケーションでサーバー側の文書変換を実行することができます。ブラウザが扱えないコンテンツ・タイプ (プラグインまたは固有のアプリケーション立ち上げを必要とする文書) から、HTML、GIF、JPEG などのブラウザがネイティブで扱えるコンテンツ・タイプまたは PDF など、プラグインがすぐに使用できるコンテンツ・タイプへ文書を変換することができます。図 34 は、シン・クライアントのアーキテクチャーを示しています。

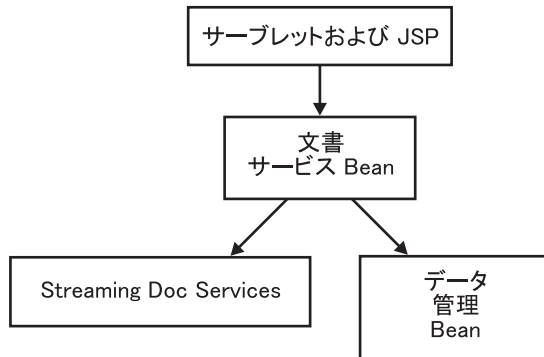


図 34. シン・クライアント

アプレットまたはサーブレット

Web ベースのアプリケーションで、アプレットまたはサーブレットを使用することにより、文書の表示および注釈の編集機能も提供できます。eClient ビューアー・アプレットは、このアーキテクチャーを使用します。アプレットは、汎用文書ビューアーを使用して文書を表示することができます。文書タイプの中には、背景の形式などの共通情報を効率良く共用できるように、いくつかのパーツに分割されてコンテンツ・サーバー上に保管されるものもあります。追加のパーツは、必要なときに汎用文書ビューアーによって要求されます。ビューアーを含んでいるアプレットは、HTTP 要求をサーブレットへ送ることによって要求を満たします。サーブレット側では、CMBDataManagement Bean を使用しているコンテンツ・サーバーが要求された情報を入手します。図 35 は、アプレットまたはサーブレットのアーキテクチャーを示しています。

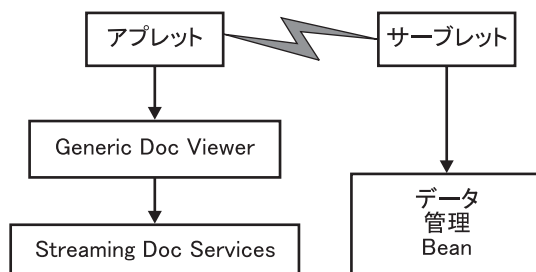


図 35. アプレットまたはサーブレット

二重モードおよびアプレットまたはサーブレット

Web ベースの表示アプリケーション用に提供されたさまざまな例を使用することができます。サーバー上およびアプレット内の `CMBDocumentServices` を使用します。この方法は、文書がアプレット内ではレンダリングできないがサーバー上で変換できる場合に役立ちます。これは、アプレット上およびサーバー上で基礎をなす文書エンジンが、それぞれ異なる機能を持つ場合に生じます。

例えば、サーバーが Windows NT または 2000 で、アプリケーションが OS/2 上で稼働している場合、アプレットがすべての文書タイプをレンダリングできないことがあります。アプレットは、TIFF など、サポートする文書フォーマットをレンダリングします。オフィス・フォーマットなど、アプレットがレンダリングできないタイプの場合には、サーブレットに変換を要求します。アプリケーション・ユーザーから見ると、同じインターフェースが同じ機能を持って表示されます。ただし、サーバー側で文書を変換する時のサーバー・パフォーマンスは、ローカルで文書をレンダリングする場合よりも遅くなる可能性があります。

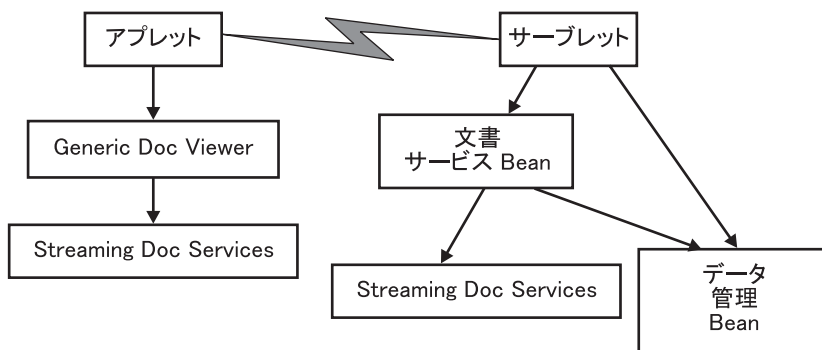


図 36. 二重モード・ビューアー

注釈サービスの使用

Enterprise Information Portal 8.1 Java ビューアー・ツールキットは、文書注釈のレンダリングと変換を行う機能を提供しています。プラグ可能注釈エンジンでは、文書サービスと同様に、アプリケーションで使用して、別のタイプの注釈を解釈できる追加機能を提供しています。471 ページの図 37 は、汎用文書ビューアーと文書サービスと注釈サービスがどのようにかかわり合っているかを図示しています。

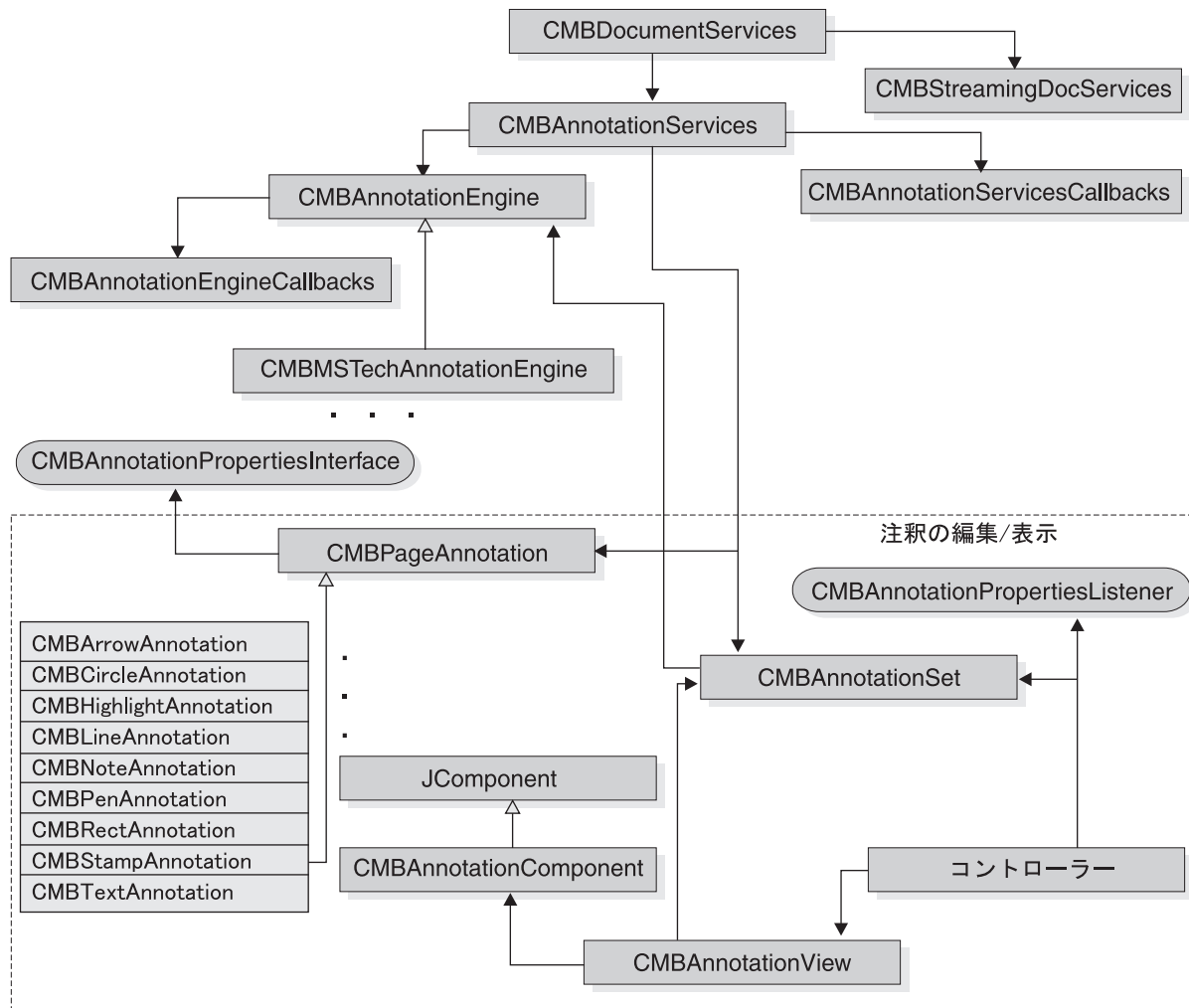


図 37. 注釈サービスと汎用文書ビューアーの関連

注釈サービス・インターフェースの使用

CMBAnnotationServices は、Java ビューアー・ツールキットで使われるメイン・インターフェースを提供します。注釈サービスを使用すると、バックエンドとは別に、注釈サービスを使用している注釈オブジェクトをロードし、操作し、保管できます。注釈を使用する場合は、注釈データをストリームとして渡し、さらに注釈オブジェクトを CMBPageAnnotation インスタンスに変換する適切な注釈エンジンに接続する必要があります。それから、ユーザーは注釈を操作し編集して、元の形式でバックエンドに注釈を保管できます。

472 ページの図 38 は、注釈サービス・クラスの図を示しています。

注釈エンジンをインプリメントするには、抽象クラス CMBAnnotationEngine を拡張しなければなりません。注釈エンジンは CMBAnnotationServicesCallbacks インターフェースと CMBAnnotationEngineCallbacks インターフェースを使用してアプリケーションや注釈サービスと通信します。EIP 8.1 の注釈エンジンは Content Manager 注釈形式のみを理解します。この注釈形式は Content Manager バージョン

7、Content Manager バージョン 8. 1、および VI/400 バックエンドで使用されます。

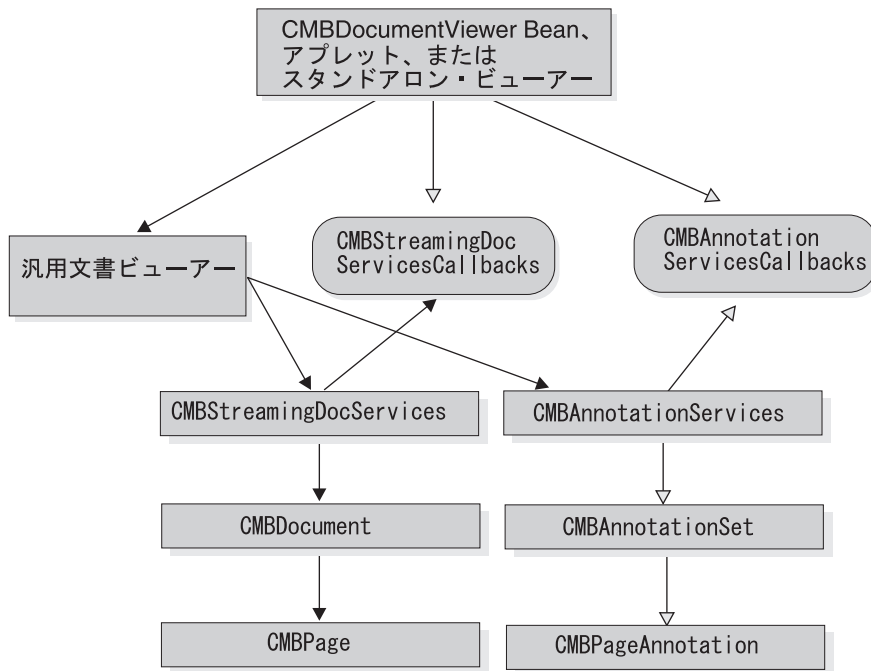


図 38. 注釈サービス・クラスの図

注釈編集サポートについて

Model View Controller (MVC) 設計パターンは、注釈編集機能をインプリメントするために使用されます。 MVC は、注釈データを表すモデルとして機能します。 CMBAnnotationSet には、データで操作するがユーザー・インターフェースを持たないメソッドがあります。 CMBAnnotationSet クラスは、CMBPageAnnotation オブジェクトのリストを維持します。 各文書は、注釈を表す CMBAnnotationSet オブジェクトと関連付けられます。 CMBAnnotationView はモデルからユーザーにデータを表示するビューとして機能します。 CMBAnnotation は、ビュー・コンポーネントについての注釈のすべてのドロー (JComponent) を処理します。

CMBAnnotationComponent は、注釈がドローされるビュー・コンポーネントとして使用できるヘルパー・クラスです。 コントローラーはビューアー・ツールキットの内部機能であり、注釈の作成と編集のためにマウスおよびキーボード・イベントを処理します。

CMBPageAnnotation は、文書のページについて単一注釈を記述する基本クラスです。 図形の注釈について別のタイプを定義する必要がある場合は、CMBPageAnnotation クラスを拡張しなければなりません。 作成できる Content Manager 注釈タイプには、CMBArrowAnnotation、CMBCircleAnnotation、CMBHighlightAnnotation、CMBLineAnnotation、CMBNoteAnnotation、CMBPenAnnotation、CMBRectAnnotation、CMBStampAnnotation、および CMBTextAnnotation の 9 つがあります。

注: 注釈サービスは、GUI アプリケーションで使用できます。

注釈サービスを使用したアプリケーションの作成

このセクションでは、注釈サービス・アプリケーションを作成するためのステップと、それに使用する API について説明します。API 使用法の詳細については、「オンライン API 解説書」を参照してください。

1. CMBAnnotationServicesCallbacks のサブクラスを作成し、注釈コールバックを処理するために抽象メソッドをインプリメントしてください。

2. CMBAnnotationServices のインスタンスを作成します。

```
CMBAnnotationServices annoServices = new  
CMBAnnotationServices(annoServicesCallbacks);
```

3. 注釈ストリームをロードして、CMBAnnotationSet のインスタンスを入手します。

```
CMBAnnotationSet annotationSet = annoServices.loadAnnotationSet(annoStream,  
format, documentResolution, annotationPartNumber );
```

4. 注釈ビューを作成します。

```
CMBAnnotationComponent annoComponent = new CMBAnnotationComponent();  
CMBAnnotationViewer annoView = annoServices.prepareAnnotationView(  
annoComponent, annotationSet );  
annoView.refreshEntireDrawingArea();
```

5. アプリケーションの JFrame または JPanel のような Swing コンテナに注釈コンポーネントを追加します。

6. これで、注釈サービス API を使用して、新規の注釈を追加したり、既存の注釈を編集、または削除できるようになりました。

```
annoServices.prepareToAddAnnotation();  
annoServices.addAnnotation()  
annoServices.removeAnnotation();  
annoServices.reorderAnnotation();  
...
```

7. 変更された注釈を保管します。

```
annoServices.saveAnnotationset(annotationSet);
```

注釈サービスのサンプルが <CMBROOT>%Samples%java%viewer
(TAnnotationEditor.java) ディレクトリに提供されています。

ユーザー・アプリケーションへのカスタム注釈タイプの追加

注釈サービスにカスタム注釈タイプを追加するには、以下のステップを完了してください。API 使用法の詳細については、「オンライン API 解説書」を参照してください。

1. CMBPageAnnotation のサブクラスを作成します。共通クラス TImageAnnotation は CMBPageAnnotation を拡張します。

2. 100 以上の値を持つカスタム注釈のための定数を定義します。1 から 99 の値は予約済みです。

```
public static final int ANN_IMAGE = 101;
```

3. 以下の CMBPageAnnotation のメソッドをオーバーライドします。

```
public void draw(Graphics2D g2);  
public void drawOutline(Graphics2D g2);  
public CMBPropertiesPanel getAnnotationPropertiesPanel();
```

4. `CMBAnnotationPropertiesInterface` インターフェースをインプリメントし、プロパティ・パネルを作成します。ユーザーがカスタム注釈プロパティの編集を選択すると、プロパティ・パネルが表示されます。
5. カスタム注釈プロパティに特定の `set` および `get` メソッドを提供します。
6. カスタム注釈タイプを追加します。

```
annoServices.prepareToAddAnnotation(TImageAnnotation.ANN_IMAGE,  
"TImageAnnotation",1);
```

注釈タイプのサンプルの `TImageAnnotation` が `<CMBROOT>%Samples%.java%viewer` ディレクトリーに組み込まれています。サンプルでは、カスタム注釈タイプを注釈サービスに追加する方法を図示しています。

Enterprise Information Portal タグ・ライブラリーおよびコントローラー・サーブレットでの使用

Enterprise Information Portal には Java Server Pages (JSP) タグ・ライブラリーおよびサーブレットが組み込まれており、これらを使用して Web アプリケーション用の JSP またはサーブレットを作成することができます。タグ・ライブラリーを使用することにより、EIP JavaBeans に作成された JSP における Java スクリプトレットの必要性が減少します。

このタグ・ライブラリーは、model-view-controller 設計による Web アプリケーションのコントローラーとして働くサーブレットを使用し、Bean の初期化やその他のアクションを実行します。

タグ・ライブラリーおよびサーブレットのセットアップ

タグ・ライブラリーとサーブレットを使用するには、それらを IBM WebSphere® Application Server と共に Web サーバーにインストールし、その Web サーバーを構成する必要があります。タグ・ライブラリーとサーブレットのインストール、その構成、および WAR/EAR ファイルの作成については、「*Enterprise Information Portal* の計画とインストール」を参照してください。

タグ・ライブラリーの使用

次の JSP サンプルでは、検索テンプレート・タグを使用して検索テンプレートのリストを取得します。

```
<%@ taglib uri="cmb" prefix="cmb" %>
<%@ page import="com.ibm.mm.beans.*" %>
<jsp:useBean id="connection" scope="session"
              class="com.ibm.mm.beans.CMBConnection" />
<%
    CMBSchemaManagement schema = connection.getSchemaManagement();
    CMBSearchTemplate[] searchtemplates = schema.getSearchTemplate();
    request.setAttribute("searchtemplates", searchtemplates);
%>
<html>
<head>
<title>Search Templates Tag Test</title>
</head>

<body bgcolor="white">
<table border=2 cellspacing=3 cellpadding=3>
<tr>
<td><b>Available Search Templates</b></td>
</tr>
<tr>
<td><%= searchtemplate.getName() %></td>
</tr>
</table>
</body>
</html>
```

taglib ディレクティブは、そのページでの EIP タグ・ライブラリーの使用を宣言し、これに cmb 接頭部を関連付けています。次に、searchtemplates タグが呼び出され、getName() メソッドが各検索テンプレートの名前を戻します。

タグ・ライブラリーで使用される規則

JSP タグには、それが使用または生成する Bean を指定する属性があります。これらの属性が指定されない場合、デフォルト値が使用されます。これらのパラメーターはオプションです。これらの値はローカル変数から、また要求およびセッション有効範囲内の属性から取られます。サーブレット・ツールキットと共に使用される場合、ローカル変数、要求属性、またはセッション属性は適切なデフォルトを含みます。表 32 には、デフォルトの Bean およびこれが想定または配置される有効範囲を示します。これらの規則はサーブレットにも適用されます。タグ・ライブラリーを使用するその他いずれのサーブレットにおいても、この規則に従ってください。

表 32. タグ・ライブラリーの規則

有効範囲	名前	タイプ	説明
アプリケーション	connectionPool	CMBCConnectionPool	セッション間で共用される接続プール Bean
セッション	接続	CMBCConnection	そのセッションの CMBCConnection のインスタンス
セッション	スキーマ	CMBSchemaManagement	スキーマ管理 Bean
セッション	データ	CMBDataManagement	データ管理 Bean
セッション	ユーザー	CMBUserManagement	ユーザー管理 Bean
セッション	照会	CMBQuesryService	照会サービス Bean
セッション	traceLog	CMBTraceLog	トレース・ログ Bean (その他の Bean は、この Bean にトレースを送信する)
セッション	docservices	CMBDocumentServices	文書サービス Bean
要求	項目	CMBItem	最後に操作した項目
要求	項目	CMBItem[]	最後に操作した項目のコレクション
要求	searchTemplate	CMBSearchTemplate	選択した検索テンプレート
要求	searchResutls	CMBSearchResults	前回の検索の結果

タグの要約

次に、タグ・ライブラリーの要約を示します。

接続関連のタグ

```
<cmb:datasources" connection="connection">datasource ...  
</cmb:datasources>
```

使用可能なデータ・ソースに対して反復を行います。

connection

接続を含む CMBConnection 型の変数の名前を指定します。

datasource

データ・ソース名をストリングとして含むストリング変数。

スキーマ関連タグ

<cmb:searchtemplates searchTemplates="searchTemplate"> ...

</cmb:searchtemplates>

使用可能な検索テンプレートに対して反復を行います。

searchTemplates

検索テンプレートを含む CMBSearchTemplate[] 型の配列の名前を指定します。

searchTemplate

検索テンプレートを含む CMBSearchTemplate 型の変数。

<cmb:searchcriteria searchTemplate="searchtemplate">criteria ...

</cmb:searchcriteria>

検索テンプレートの検索基準に対して反復を行います。

searchTemplate

検索テンプレートの名前を指定します。

criteria

検索基準を含む CMBSTCriteria 型の変数。

<cmb:displaycriteria searchTemplate="searchTemplate">criteria ...

</cmb:displaycriteria>

検索テンプレートについて表示可能な検索基準に対して反復を行います。

searchTemplate

検索テンプレートの名前を指定します。

criteria

検索基準を含む CMBSTCriteria 型の変数。

<cmb:allowedoperators criteria="criteria">operator ...

</cmb:allowedoperators>

検索基準で使用するのことができるオペレーターに対して反復を行います。

criteria

検索基準を含む CMBSTCriteria 型の変数の名前を指定します。

operator

オペレーターの値を含むストリング。

<cmb:predefinedvalues criteria="criteria"> value...

</cmb:predefinedvalues>

検索基準の事前定義値に対して反復を行います。

criteria

検索基準を含む CMBSTCriteria 型の変数の名前を指定します。

value 検索基準の事前定義値を含むストリング。

<cmb:entities schema="schema">entity ... </cmb:entities>

使用可能な統合エンティティに対して反復を行います。

schema スキーマを含んでいる CMBSchemaManagement 型の変数の名前を指定します。

entity エンティティの名前を含むストリング。

<cmb:attributes entity="entity" schema="schema">attribute ...

</cmb:attributes>

統合エンティティの統合属性に対して反復を行います。

schema スキーマを含む CMBSchemaManagement 型の変数の名前を指定します。

entity エンティティの名前を指定します。

attribute

属性を含む CMBAttribute 型の変数の名前を保持するストリング。

検索関連のタグ

<cmb:searchresults searchresults="searchResults">item ...

</cmb:searchresults>

検索結果に対して反復を行います。

searchResults

検索結果を含む CMBSearchResults 型の変数の名前を指定します。

item 検索から戻された項目を含む CMBItem 型の変数の名前を含むストリング。

項目関連のタグ

<cmb:itemattributes item="item"> attrname ... attrtype... attrvalue...

</cmb:itemattributes>

項目の属性に対して反復を行います。

item 項目を含む CMBItem 型の変数の名前を指定します。

attrname

属性の名前を含むストリング変数。

attrtype

属性タイプを含むストリング変数。

attrvalue

属性の値を含むストリング変数。

<cmb:itemcontents " data="data" item="item"> content...

</cmb:itemcontents>

項目のコンテンツに対して反復を行います。

data CMBDataManagement 型の変数の名前を指定します。

item 項目を含む CMBItem 型の変数の名前を指定します。

content

項目の内容を含む CMBObject 型の変数。

**<cmb:itemnotelogs " data="data" item="item">notelog ...
</cmb:itemnotelogs>**

項目のメモ記録に対して反復を行います。

data CMBDataManagement 型の変数の名前を指定します。

item 項目を含む CMBItem 型の変数の名前を指定します。

notelog

項目のメモ記録を含む CMBObject 型の変数。

**<cmb:itemprivileges data="data" item="item">privilege ...
</cmb:itemprivileges>**

項目の特権に対して反復を行います。

data CMBDataManagement 型の変数の名前を指定します。

item 項目を含む CMBItem 型の変数の名前を指定します。

privilege

項目の特権を含む CMBPrivilege 型の変数。

**<cmb:itemresources data="datamanagement" item="item"> resource...
</cmb:itemresources>**

項目のリソースに対して反復を行います。

data CMBDataManagement 型の変数の名前を指定します。

item 項目を含む CMBItem 型の変数の名前を指定します。

resource

項目のリソースを含む CMBResources 型の変数。

**<cmb:unmappeditem data="data"
item="item">unmappeditem...</cmb:unmappeditem>**

指定のマップ項目からマップされていない項目を戻します。

data CMBDataManagement 型の変数の名前を指定します。

item マップ項目を含む CMBItem 型の変数の名前を指定します。

unmappeditem

マップされていない項目を含む CMBItem 型の変数。

<cmb:viewdata data="data " item="item">viewdata... </cmb:viewdata>

項目の表示を戻します。

data CMBDataManagement 型の変数の名前を指定します。

item 項目を含む CMBItem 型の変数の名前を指定します。

viewdata

表示可能データを含むための CMBViewData 型の変数。

フォルダー関連のタグ

<cmb:folderitems folder="folder"> item... </cmb:folderitems>

フォルダーのコンテンツに対して反復を行います。

folder フォルダーのコンテンツを含む CMBItem 型の変数の名前を指定します。

item フォルダーを表す CMBItem 型の変数。

文書関連のタグ

<cmb:viewerdocuments docservices="docservices">document ...
</cmb:viewerdocuments>

現在ロードされている文書に対して反復を行います。

docservices

CMBDocumentServices 型の変数の名前を指定します。

document

文書を含む CMBDocument 型の変数。

<cmb:documentpages document="document"> docpage...
</cmb:documentpages>

このタグは文書のページを繰り返します。

document

文書を含む CMBDocument 型の変数の名前を指定します。

docPage

ページを含む CMBPage 型の変数。

EIP コントローラー・サーブレット

EIP は、Web アプリケーションを作成する際に使用できる、プラグ可能アクションを備えたサーブレットを提供します。このサーブレットは model-view-controller 設計による Web アプリケーションのコントローラーとして機能し、アクションを実行して Bean (モデル) を初期化します。また、Bean には、JSP (ビュー) 内で JSP タグを使用して直接または間接的にアクセスが行われます。

次の一般的なアプリケーション・タスクに対するアクションが用意されています。

- ログオンおよびログオフ。
- 検索。
- 文書の作成、検索、変更、および削除。
- フォルダーの作成、およびフォルダーの文書の追加または削除。
- 表示する文書と文書ページの起動。

また、アクションの前後には、サーブレットがコンテンツ・サーバーへの接続の管理などの一般的なタスクを実行します。いずれのアクションの後にも JSP が呼び出され、結果がフォーマットされてブラウザーに戻されます。

ユーザーは、サーブレットのカスタマイズ、新規アクションの追加、および JSP とアクションとの関連付けを行うことができます。

サーブレットの機能

以下に、使用可能なサーブレットの性質をいくつか示します。

接続プール

コントローラー・サーブレットは EIP 接続プールを使用することにより、ハイパフォーマンスの接続管理を行います。セッションに接続が割り振られている時間は、要求に対するものか、セッションのログオン時間に対するもののいずれかとなります。現在のところ接続プールの有効範囲はアプリケーションです。

タイムアウト・セッションのログオン

セッションがタイムアウトになった場合、サーブレットに要求が着信すると、ログオン JSP が表示され、ユーザーは再度ログオンが可能になります。ログオンが正常に行われると、本来の要求が実行されます。

セッション終了時のクリーンアップ

サーブレットは、セッションが終了すると、ログオフまたはタイムアウトのいずれかによってセッションを適切にクリーンアップします。これは、接続が破棄されるか、プールに戻されるということです。サーブレットにより作成された他の EIP Bean はすべて終了され、そのリソースが解放されます。これはガーベッジ・コレクション・サイクルを待たずに行われます。

ロケール

サーブレットは、基本となる Bean にロケールが正しく設定されていることを確認します。このため、メッセージおよび文字ストリングにはロケールの区別があります。

各種 JSP セットの使用

プロパティー・ファイル (デフォルト名は `cmbervletjsp.properties`) には、サーブレット・アクションへの応答に使用する JSP が記述されています。プロパティー・ファイルの位置は、アプリケーション・パラメーターとなります。したがって、各種 JSP セットを使用して複数の異なる Web アプリケーションを作成することができます。

サーブレットの拡張

サーブレットが識別するすべてのアクションは、プロパティー・ファイル (デフォルト名は `cmbervlet.properties`) に定義されます。このファイルを変更することにより、サーブレットのアクションを追加、変更、または削除することができます。新規アクションを追加するには、以下のステップを実行します。

1. アクションを実行するクラスをインプリメントします。このクラスは `com.ibm.mm.servlets.CMBServletAction` を拡張する必要があります。
2. クラス名とアクション名を `cmbervlet.properties` ファイルに追加します。この構文は以下のようになります。

```
actions = list of actionsaction.<action_name>.class = class_name
```

`actions` により、サーブレットにより識別されるアクションがリストされます。それぞれのアクションごとに、プロパティー・ファイル内の各行がアクションのクラスを定義します。例えば、アクション `replay` をクラス `ReplayAction` に追加するには、以下のようになります。

```
actions =... replay
action.replay.class = ReplayAction
```

アクションを置き換えたり、事前定義されたアクションの前後に独自のアクションを指定することもできます。例えば、ログオンの前に独自のアクションを実行し、追加の妥当性検査を行うには、以下のようにします。

```
action.logon.class = MyLogonAction com.ibm.mm.servlets.CMBLogonAction
```

すべての事前定義アクションで使用される命名規則は、`com.ibm.mm.servlets.CMBaction Action` となります。ここで *action* はアクションの名前で、先頭は大文字です。

サーブレット参照

アプリケーション・パラメーター、要求パラメーター、およびプロパティー・ファイルのセットを使用することにより、アプリケーションでコントローラー・サーブレットを使用することができます。

規則

サーブレットは以降のセッションおよび要求の値を定義し、これを他の JSP またはサーブレットで使用することができます。これらの規則は、JSP タグ・ライブラリーに適用されます。これらの規則は、EIP タグ・ライブラリーの場合と同じです。

アプリケーション・パラメーター

サーブレットは、以下のアプリケーション・パラメーターを識別します (代わりの方法として、これらを `cmbServlet.properties` ファイル内に置く方法があります)。

アプリケーション・パラメーター	値	説明
<code>servletPropertiesURL</code>	URL	<code>cmbServlet.properties</code> ファイルの位置。
<code>defaultServerType</code>	Fed、ICM、OD、DL、DES、V4、IP、DD、...	デフォルトのログオン情報。この値と <code>defaultServer</code> 、 <code>defaultUserId</code> 、 <code>defaultPassword</code> は、共用ユーザー ID 環境で使用可能です。ログオンには、ログイン・ページを示す代わりに、デフォルトのログオン情報が使用されます。
<code>defaultServer</code>		デフォルトのログオン情報。
<code>defaultUserId</code>		デフォルトのログオン情報。
<code>defaultPassword</code>		デフォルトのログオン情報。
<code>connectionpool</code>	ブール値: true false	接続プールを使用可能にします。
<code>maxfreeconnection</code>	integer	接続プールで使用可能な接続の最大数
<code>minfreeconnection</code>	integer	接続プールで使用可能な接続の最少数
<code>timeout</code>	integer	フリー接続が切断され、破棄された後の所要時間 (ミリ秒)。

アプリケーション・パラメーター	値	説明
noSessionPage	URL	セッションまたは接続を確立せずにサーブレットが呼び出された場合にログオンで表示するページ。これを使用することにより、ユーザーのログオンが必要な場合であっても、ログオンのプロンプトを出してオリジナルのアクションに戻り、EIP にブックマークを付けたリンクを機能させることができます。
timedOutPage	URL	活動がなく、セッションがタイムアウトになった場合に表示するページ。
serverErrorPage	URL	サーバーへのアクセスでエラーが発生した場合に表示するページ。
connectFailedPage	URL	サーバーへの接続でエラーが発生した場合に表示するページ。この場合、プロンプトを表示して、サーバー用の正しいユーザー ID/パスワードを入力し、再試行するようにできます。
tracelevel	0、1、または 2	トレース・レベルの指示は、以下のとおりです。 <ul style="list-style-type: none"> • 0 - ログなし • 1 - ログ対象: 例外 (デフォルト) • 2 - ログ対象: 例外、アラート・メッセージ、WebSphere Application Server のヘッダーおよび属性、EIP ini ファイル、JVM システム・プロパティ、EIP 内部トレース情報
connectiontype	0、1、または 2	EIP データベースおよびコンテンツ・サーバー・ランタイムの位置。 <ul style="list-style-type: none"> • 0 - ローカル (デフォルト) • 1 - リモート • 2 - 動的
cmbclient	URL	cmbclient.ini の位置
cmbscs	URL	cmbscs.ini の位置。
serviceconnectiontype	0、1、または 2	サービス・ランタイムの位置。 <ul style="list-style-type: none"> • 0 - ローカル (デフォルト) • 1 - リモート • 2 - 動的
cmbsvclient	URL	cmbsvclient.ini の位置。
cmbsvcs	URL	cmbsvcs.ini の位置。
cmbcc2mime	URL	cmbcc2mime.ini の位置。
cachedir	ディレクトリーの名前	文書変換の際に文書をキャッシュするディレクトリー。
jnitrace	ファイルの名前	文書変換で使用する JNI ロジックの JNI トレース情報を書き込むファイル (IBM による診断用)。

アプリケーション・パラメーター	値	説明
conversion	ブール値: true または false	true の場合、(可能な場合は) 中間層でブラウザに表示可能な形式に文書が変換されます。 false の場合、未変換のオリジナル文書がブラウザに送られます。
maxresults	integer	最大戻りヒット数: -1 (デフォルト) は全ヒットを示す。
valuedelimiter	文字	検索基準の値を区切る文字を定義します。デフォルトはロケールに依存し、米国英語の場合はコンマ (,) となります。
conversion.<mimetype>	<none document page >	特定の mimetype の文書を表示するための変換オプション。これは、viewDocument サブレットの振る舞いに影響します。Page は、文書をページ番号付けする試みを意味します。Document は、文書をブラウザで読み取り可能な形式に変換することを意味します。None は、変換なし、すなわちネイティブな形式で文書を戻すことを意味します。
nameseparator	文字	修飾名内で、親コンポーネント属性から子コンポーネント属性を分離する文字を定義します。デフォルトは、地域によって異なり、米国英語ではスラッシュ (/) です。

プロパティ・ファイル

サブレットはプロパティ・ファイル `cmbServlet.properties` を検出します。このファイルは、ここに定義されるアクションを含め、サブレットが使用可能なアクションを定義します。また、使用される JSP ファイルの名前も定義します。

サブレットのプロパティは Web アプリケーション・サーバー (サブレット・エンジン) 上で定義することもできます。構文は、ファイル内で使用されるものと同じです。

`cmbServlet.properties` の内容は、制御サブレットによって `Properties` オブジェクトに保管されます。これは、次の例に示すようにアプリケーション属性 `"cmbServletProperties"` を通じて使用することができます。

```
// check to see if connection pooling is enabled
String name = "connectionpool";
Properties props = (Properties) application.getAttribute(
    "cmbServletProperties");
String value = props.getProperty(name);
// "true" if enabled, "false" otherwise
```

要求パラメーター

サブレットは、以下の要求パラメーターを識別します。応答側の JSP で使用するための追加パラメーターの指定が可能です。

一般

action=action

実行するアクション。指定可能な追加パラメーターは、以下のよう
にアクションにより異なります。

このパラメーターはオプションです。これを指定しない場合、接続
のタイムアウトやログオンのチェックなどの標準のセットアップ
後、サーブレットにより応答ページが実行されます。

reply=<URL>

オプション。 `cmbervlet.properties` ファイルでアクションへの応答
として定義された JSP ではなく、このパラメーターに指定された
JSP に転送します。アクション・パラメーターが指定されず、応答
が指定されている場合は、タイムアウトおよびログオンの接続の検
査などの、標準のセットアップを実行した後で、応答ページがコン
トローラー・サーブレットによって実行されます。

接続関連

action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]

サーバーにログインします。サーバー・タイプ、サーバー名、ユー
ザー ID およびパスワードを指定する必要があります。接続ストリ
ングおよび初期化ストリングは、オプションであり、サーバーのタ
イプに応じて異なります。

action=logoff [endSession=<true|false>]

サーバーからログアウトします。デフォルトではセッションも終了
します。

検索関連

action=searchTemplate template=<> {<criterionname>.op=<> <criterionname>=<>}

指定した検索テンプレートと基準値を使用して検索を実行します。

action=searchEntity entity=<> {attribute.<attrname>.op=<> attribute.<attrname>=<>} [conjunction=<and|or>]

エンティティを使用して検索を実行します。属性値および演算子
も指定することができます。属性値中の複数の値は、アプリケーシ
ョン・パラメーターで指定された値の区切り文字で区切ります。属
性を組み合わせて照会とするには、`and` (デフォルト) または `or` (結
合パラメーターにより指定) を使用します。

action=searchQuery queryString=<> {queryParameter.<parametername>=<>}

指定した照会ストリングを使用して検索を実行TMします。照会構文
は、検索対象のサーバーによって異なります。

任意の数の追加照会パラメーターを指定することができる場合もあ
ります。これもサーバーによって異なります。

項目関連

action=lock itemId=<>

通常、更新中の排他的アクセスのため、項目をロックします。

action=unlock itemId=<>

ロックされた項目をアンロックします。

action=createItem type=<document|folder> entity=<>

{attribute.<attrname>=<attrvalue>}

項目を作成します。ポストされると、コンテンツを作成することができます。

action=retrieveItem itemId=<>

項目の属性およびコンテンツを検索します。これは、サーバーに保管された最新のコンテンツを確実に使用する際に役に立ちます。

action=updateItem itemId=<> [entity=<>]

{attribute.<attrname>=<attrvalue>}

項目の属性を更新します。エンティティを指定した場合、項目が再度索引付けされます。ポストによりサーブレットが起動されると、コンテンツも更新されます。

action=deleteItem itemId=<>

項目を削除します。

action=addContent itemId=<>

項目にコンテンツ・パーツを追加します。コンテンツ・データがポストされます。

action=getContent itemId=<> contentIndex=<>

コンテンツ・パーツを取得して、これをブラウザーに戻します。

action=updateContent itemId=<> contentIndex=<>

項目のコンテンツ・パーツを更新します。コンテンツ・データがポストされます。コンテンツが存在しない場合、コンテンツ・パーツが追加されます。

action=deleteContent itemId=<> contentIndex=<>

指定した項目のコンテンツ・パーツを削除します。

action=addNoteLog itemId=<>

項目のメモ記録を変更します。メモ記録のテキストがポストされます。

action=updateNoteLog itemId=<> notelogIndex=<>

項目のメモ記録を変更します。メモ記録のテキストがポストされます。メモ記録が存在しない場合は、追加されます。

action=deleteNoteLog itemId=<> notelogIndex=<>

項目のメモ記録テキストを削除します。メモ記録のテキストがポストされます。

フォルダー関連

action=addItemToFolder itemId=<> folderId=<>

指定項目を指定したフォルダーに追加します。

action=removeItemFromFolder itemId=<> folderId=<>

指定フォルダーから指定した項目を除去します。

文書関連

`action=viewDocument itemId=<>`

文書を検索して表示します。文書にページ番号が付けられている場合、このアクションは、ビューアーのフレーム・セットを生成する JSP に転送されます。文書にページ番号が付けられていない場合は、このアクションは、文書の実際のコンテンツを戻します。

`action=viewPage itemId=<> page=<> scale=<> rotation=<>`

`annotations=<yes|no>`

文書のページを検索します。

サーブレット・ツールキット機能マトリックス

表 33. サーブレット機能マトリックス

アクション	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logon	Y	Y	Y	Y	Y	Y	Y
logoff	Y	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	Y	N/A
searchEntity	Y	Y	Y	Y	Y	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y	Y
lock	Y	Y	Y	Y	N/A	N/A	N/A
unlock	Y	Y	Y	Y	N/A	N/A	N/A
createItem	Y	Y	N/A	N/A	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y	Y
updateItem	Y	Y	Y	Y	N/A	N/A	N/A
deleteItem	Y	Y	Y	Y	N/A	N/A	N/A
addContent	Y	Y	N/A	N/A	N/A	N/A	N/A
getContent	Y	Y	Y	Y	Y	Y	Y
updateContent	Y	Y	Y	Y	N/A	N/A	N/A
deleteContent	Y	Y	Y	Y	N/A	N/A	N/A
addNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
updateNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
deleteNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
addItemToFolder	Y	Y	Y	N/A	N/A	N/A	N/A
removeItemFrom Folder	Y	Y	Y	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y	Y

Y —— 機能がサポートされる **N/A** —— 機能がサポートされない

注: logon、logoff、getContent、retrieveitem、viewDocument および viewPage アクションはすべてのコンテンツ・サーバーでサポートされています。

表 34. サブレット機能マトリックスの続き

アクション	DES	DB2	JDBC	IC	Fed	FileNet
logon	Y	Y	Y	Y	Y	Y
logoff	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	N/A
searchEntity	N/A	Y	Y	N/A	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y
lock	N/A	N/A	N/A	N/A	Y	N/A
unlock	N/A	N/A	N/A	N/A	Y	N/A
createItem	N/A	Y	Y	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y
updateItem	N/A	Y	Y	N/A	Y	N/A
deleteItem	N/A	Y	Y	N/A	Y	N/A
addContent	N/A	N/A	N/A	N/A	Y	N/A
getContent	Y	Y	Y	Y	Y	Y
updateContent	N/A	N/A	N/A	N/A	Y	N/A
deleteContent	N/A	N/A	N/A	N/A	Y	N/A
addNoteLog	N/A	N/A	N/A	N/A	Y	N/A
updateNoteLog	N/A	N/A	N/A	N/A	Y	N/A
deleteNoteLog	N/A	N/A	N/A	N/A	Y	N/A
addItemToFolder	N/A	N/A	N/A	N/A	N/A	N/A
removeItemFrom Folder	N/A	N/A	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y

Y —— 機能がサポートされる **N/A** —— 機能がサポートされない

注: logon、logoff、getContent、retrieveitem、viewDocument および viewPage アクションはすべてのコンテンツ・サーバーでサポートされています。

情報マイニングの使用

このセクションでは、最初に、情報マイニング Bean を使用したアプリケーションの作成方法について説明し、それ以降のセクションで、独自のコンテンツ・プロバイダーの作成、情報マイニング・サービス API の使用方法、および JSP サンプルの使用方法について説明します。

Bean を使用した情報マイニング・アプリケーションの作成

情報マイニングを使用すると、最新のテキスト分析およびマイニング技術を基にした、強力なアプリケーションを作成することができます。情報マイニング Bean には、以下の機能があります。

- テキストの要約とカテゴリー化 (つまり、文書への要約の割り当て)
- カテゴリー化 (つまり、文書へのカテゴリーの割り当て)
- 情報抽出 (つまり、文書からの関連情報単位の検出および抽出)
- 言語識別 (つまり、文書の作成言語の判別)
- クラスタ化 (つまり、文書コレクション内の類似文書のグループ化)
- 特定カテゴリーの文書に制限可能なテキスト検索
- IBM Web Crawler を使用して継続的に Web から取り出される文書へのアクセス

ご注意

情報マイニング Bean を使用する前に、サービス API と JavaBeans との違いを理解することが重要です。

- 情報マイニング JavaBeans は、RAD (rapid application development) のためのソフトウェア・コンポーネントであり、JavaBeans に準拠しています。コードのいくつかの要素は「結合」されているので、ユーザーが変更することはできません。
- Java サービス API には、アプリケーション作成のための独立したビルディング・ブロックとして、情報マイニングの全機能が含まれています。詳細については、532 ページの『サービス API の使用』を参照してください。

情報マイニング Bean を使用するには、まずその Bean を実行するマシンに Enterprise Information Portal 情報マイニング・コンポーネントをインストールし、構成する必要があります。

情報マイニング Bean には、以下のものがあります。

- **CMBSummarizationService**

文書の要約を作成するために使用可能な非ビジュアル Bean です。これは、3 種類のモードで実行することができ、要約の長さを決めることができます。

- **CMBCategorizationService**

文書のカテゴリを決定するために使用可能な非ビジュアル Bean です。これは、情報構造化ツールを使用して獲得された結果（つまり、そのカテゴリの特徴を示すものとして識別される、作成済みの分類法とメタデータで構成される用語、および、それぞれのカテゴリについて記述するルール）に対して実行されます。1 つの文書を、1 つまたは複数のカテゴリに割り当てることができます。カテゴリ化を行うと、1 つまたは複数のカテゴリが作成され、文書の各カテゴリへの適合度を示す信頼性の値が作成されます。ユーザーは、この信頼性の値と、戻される結果の最大数を設定することができます。

- **CMBInformationExtractionService**

文書から名前、用語、および式を抽出するために使用可能な非ビジュアル Bean です。

- **CMBLanguageIdentificationService**

文書の作成言語の判別を使用するための非ビジュアル Bean です。

- **CMBClusteringService**

文書コレクションを類似の文書またはクラスターのセットに分割するための非ビジュアル Bean です。

- **CMBAdvancedSearchService**

CMBAdvancedSearchService は、検索を実行する非ビジュアル Bean です。拡張検索で検出できるのは、情報マイニング・データ・ストアに保管されている項目のみです。

- **CMBCatalogService**

CMBCatalogService は、永続的な情報マイニング・データ・ストアを保守します。このサービスを使用するには、情報構造化ツールを使用してカタログを作成する必要があります。この Bean を使用すると、情報マイニング・サービスによって作成された結果を保管することができます。もちろん、この情報を検索し、削除することもできます。

- **CMBWebCrawlerService**

CMBWebCrawlerService は、Web スペース、つまり、非同期で稼働している Web Crawler によりクロールされたすべての HTML 文書が保管されているディレクトリーをモニターします。CMBItems（文書を表す EIP ヘルパー・クラス）は、任意の情報マイニング・サービスで利用できるこれらのファイルから作成されます。あるいは、CMBCatalogService を使用して、単に情報マイニング・データ・ストアにインポートされます。

- **CMBInfoMiningAdapter**

CMBInfoMiningAdapter は、異なる情報マイニングのイベント間のスイッチです。また、情報マイニング Bean とその他の Enterprise Information Portal Bean の協働を可能にする、CMBResultEvent のサポートも行います。次の図に示すように、CMBInfoMiningAdapter を使用すると、Bean を組み合わせてさまざまなシナリオを作成することができます。

491 ページの図 39 は、情報マイニング Bean がどのように使われるかを示したものです。

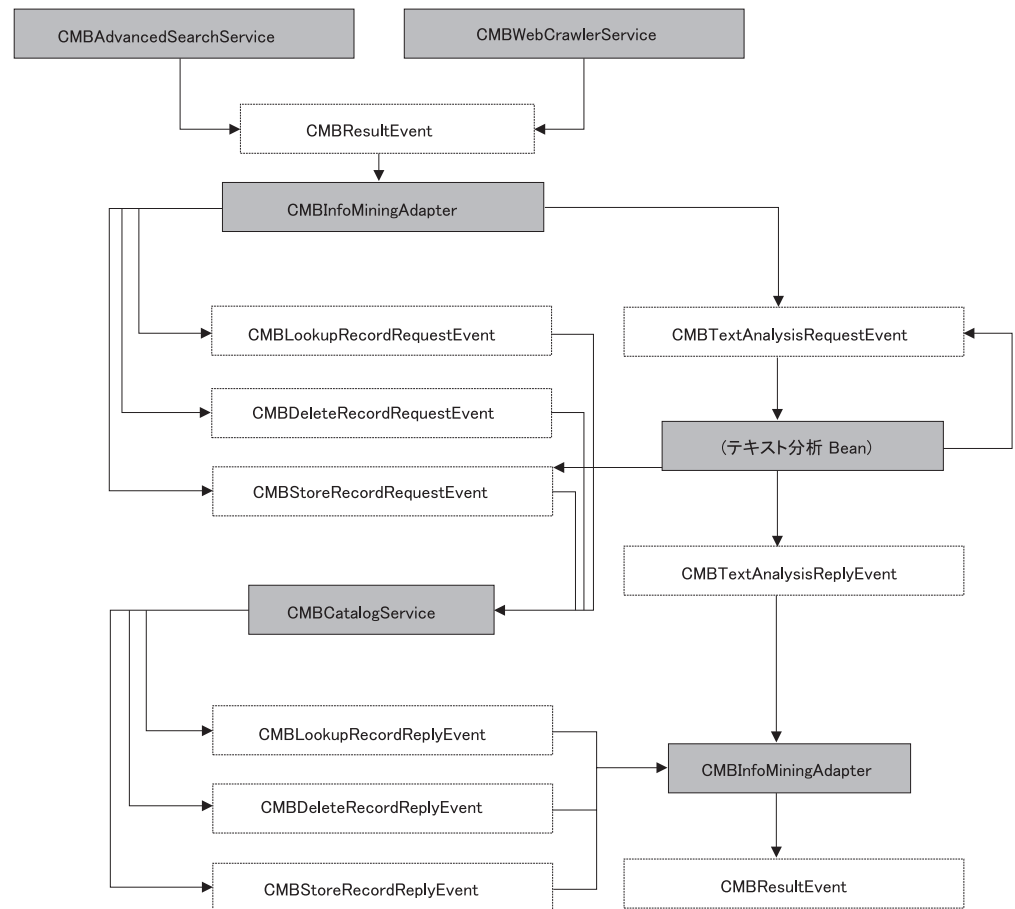


図 39. 情報マイニング Bean

図 40 に、テキスト分析 Bean をリストします。

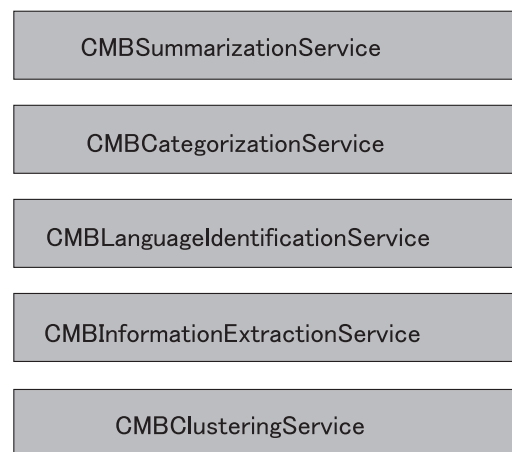


図 40. テキスト分析 Bean

情報マイニング Bean を使用してアプリケーションを作成する方法について、以下のサンプルで説明します。

- **カテゴリー化サンプル:** 情報マイニングの準備として情報を収集します。これは、ライブラリアンの一般的なステップです。 493 ページの図 41を参照してく

ださい。このサンプルでは、EIP コンテンツ・サーバー内の文書を検索するために標準 Enterprise Information Portal (EIP) 検索を実行することから始めます。(Web からの文書の収集方法については、Web Crawler のサンプル内にあります。) 次に、文書の言語が判別され、英語文書がカテゴリ化されます。カテゴリ情報はメタデータ・ストアに保管されます。

- **要約サンプル:** ライブラリアンの別の一般的なステップです。カテゴリ化サンプルの場合と同じように、最初に標準 EIP 検索を行って EIP コンテンツ・サーバー内の文書を検索します。英語文書を識別後、これらの文書を要約し、その要約をメタデータ・ストアに保管します。501 ページの図 43を参照してください。

- **情報抽出サンプル:** これは、情報マイニングのステップです。このサンプルでは、EIP コンテンツ・サーバー内の文書を検索するために標準 EIP 検索を実行することから始めます。その後、名前と用語が英語文書から抽出されます。抽出された情報は、メタデータ・ストアに保管されます。

これについては、507 ページの図 45 で説明されています。

- **クラスター化サンプル:** これは、情報マイニングのステップです。EIP コンテンツ・サーバー内の文書を検索するために標準 EIP 検索を実行することから始めます。次に、英語文書が決定され、手動でクラスター化サービスが起動されます。結果は画面上に表示されます。クラスター化情報は、メタデータ・ストアには保管されません。

これについては、512 ページの『クラスター化』で説明されています。

- **Web Crawler のサンプル:** Web Crawler を使用して文書を取得し、情報をカテゴリ内の検索に使用できるようにします (拡張検索サンプル)。EIP コンテンツ・サーバーからではなくインターネットまたはイントラネットから文書を収集することを除けば、これもカテゴリ化サンプルおよび要約サンプルと類似したライブラリアン・ステップです。カタログから検索結果にカテゴリ情報および要約情報を復元するには、カタログ・サービスを使用してそれらを検索します。

517 ページの図 49を参照してください。

- **拡張検索サンプル:** これは、情報マイニングのステップです。拡張検索を実行して、特定のカテゴリに限定された柔軟な照会を使用して文書を検索することができます。カタログから検索結果にカテゴリ情報および要約情報を復元するには、カタログ・サービスを使用してそれらを検索します。

サンプル・ファイルの位置

この章で説明されているサンプルは、以下のディレクトリーにあります。

サンプル	位置
------	----

カテゴリ化アプリケーション	
---------------	--

<CMBROOT>%samples%java%beans%infomining%categoryization

要約アプリケーション	
------------	--

<CMBROOT>%samples%java%beans%infomining%summarization

情報抽出アプリケーション	
--------------	--

<CMBROOT>%samples%java%beans%infomining%informationExtraction

クラスター化アプリケーション	
----------------	--

<CMBROOT>%samples%java%beans%infomining%clustering
--

拡張検索アプリケーション

<CMBROOT>%samples%java%beans%infomining%advancedSearch

Web Crawler アプリケーション

<CMBROOT>%samples%java%beans%infomining%webcrawler

これらのディレクトリーのそれぞれには、ソース・コードをコンパイルするための compile.bat ファイルが含まれています。アプリケーション・サンプル・ディレクトリーには、サンプル・アプリケーションを実行するための run.bat ファイルも含まれています。

文書のカテゴリー化

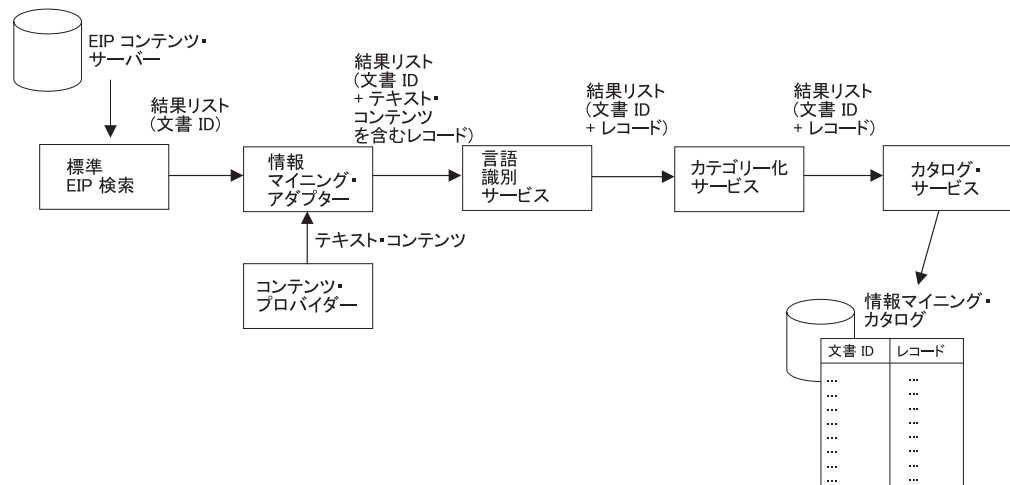


図 41. カテゴリー化サンプル

このサンプルは、標準 EIP 検索で取得した文書をカテゴリー化する方法を示しています。分析結果を保管し、適切な文書をカテゴリー内で検索できるようにします。

図 41 では、標準 EIP 検索は、EIP コンテンツ・サーバー内に保管されている文書に対して行われます。言語識別サービス Bean は、文書の作成言語を判別します。この Bean は、文書 ID を受け取って文書内容を取り出し、その内容を分析して言語を判別します。カテゴリー化サービス Bean は、文書をカテゴリー化します。その後、この情報 (文書 ID およびカテゴリー情報) は、今後のプロセスで使用できるようになります。

このサンプルでは、以下の Bean が使用されます。

- CMBCConnection
- CMBQueryService (標準 EIP 検索を実行するため)
- CMBSearchResults (標準 EIP 検索を実行するため)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBCategorizationService
- CMBCatalogService

このサンプルにおいて、アプリケーションは、

1. Bean を作成します。
2. Bean をカスタマイズします。
3. カテゴリ化サービスを使用して、英語文書を分析します。結果は、カタログに保管されます。
4. 標準 EIP 照会を実行します。
5. 検索結果 (文書のリスト) と検査のカテゴリを表示します。

Java ソースは、以下のとおりです。

Categorization.java の全ソース・コード

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Categorization implements CMBResultListener, CMBExceptionListener
{
    String DEFAULT_CMDBNAME      = "icmnlsdb";
    String DEFAULT_CMDBUSER      = "icmadmin";
    String DEFAULT_CMDBPASSWORD = "password";
    String DEFAULT_SAMPDBNAME    = "eipsampl";
    String DEFAULT_SAMPDBUSER    = "icmadmin";
    String DEFAULT_SAMPDBPASSWORD = "password";

    String CATALOG_NAME          = "Sample";
    String SEARCH_TEMPLATE       = "SearchLongBySource";
    String SEARCH_VALUE          = "ibmpress";
    String SEARCH_CRITERION      = "source";

    public Categorization() throws Throwable
    {
        // creating beans
        CMBConnection samplConnection = new CMBConnection();
        CMBConnection eipConnection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBLanguageIdentificationService languageIdentificationService =
            new CMBLanguageIdentificationService();
        CMBCategorizationService categorizationService =
            new CMBCategorizationService();
    }
}
```

```

CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// reading parameters
BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Sample database      [" + DEFAULT_SAMPDBNAME + "] : ");
String sampDbName = din.readLine();
if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

System.out.print("User ID for " + sampDbName +
" [" + DEFAULT_SAMPDBUSER + "] : ");
String sampUserName = din.readLine();
if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

System.out.print("Password for " + sampDbName +
" [" + DEFAULT_SAMPDBPASSWORD + "] : ");
String sampPasswd = din.readLine();
if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

System.out.print("EIP database      [" + DEFAULT_CMDBNAME + "] : ");
String eipDbName = din.readLine();
if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

System.out.print("User ID for
" + eipDbName + " [" + DEFAULT_CMDBUSER + "] : ");
String eipUserName = din.readLine();
if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

System.out.print("Password for " + eipDbName +
" [" + DEFAULT_CMDBPASSWORD + "] : ");
String eipPasswd = din.readLine();
if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

System.out.print("Catalog      [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("¥n¥nRunning Categorization with the
following settings:");
System.out.println("Sample database      = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database      = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog      = " + catalog + "¥n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.setServiceConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);

```

```

eipConnection.setUserId(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider
    (new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
    (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
    (categorizationService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener
    (languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
    (categorizationService);
categorizationService.addCMBStoreRecordRequestListener
    (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

samplConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
samplConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath
    (catalogService.getTaxonomy().getRootCategory().getPathAsString());
CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion
    (SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new
    CMBSearchRequestEvent(this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNC,
        searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samplConnection.disconnect();
eipConnection.disconnect();
}

```

```

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
        return;

    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)
        return;

    try {
        for(int i = 0; i < cmbItemVector.size(); i++)
        {
            CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

            CMBRecord currentRecord = currentItem.getInfoMiningRecord();

            System.out.println("¥n¥nPID          : " + currentRecord.getPID());
            System.out.println("Categories      : " + currentRecord.getValue("IKF_CATEGORIES"));
        } /* for */
    }
    catch (CMBNoSuchKeyException nske)
    {
        nske.printStackTrace();
    }
}

//implementing com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new Categorization();
        System.exit(0);
    }
    catch(Throwable t)
    {
        t.printStackTrace();
    }
}
}

```

Bean の作成

検索を実行するには、コンテンツ・サーバーとの接続が必要です。接続の確立には、CMBConnection Bean を使います。標準 EIP 検索を実行するには、CMBQueryService Bean および CMBSearchResults Bean が必要です。文書の言語を判別するには、CMBLanguageIdentificationService を使用します。判別された言語は、対応するレコードの IKF_LANGUAGE 属性に保管されます。文書は CMBCategorizationService を使用して 1 つまたは複数のカテゴリーに割り当てられます。さらに、CMBCategorizationService は、該当するレコードにカテゴリー情報を保管します。CMBCatalogService は、項目のカテゴリー情報をカタログに保管し、拡張検索で適切な文書を使用できるようにするために使用されます。検索結果イベントをテキスト分析要求イベントに変換した後、項目保管応答イベントを検索結果イベントに戻すため、2 つのアダプターが必要です。

必要な Bean を作成するコードは、以下のとおりです。

```

        CMBConnection samp1Connection = new CMBConnection();
        CMBConnection eipConnection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBLanguageIdentificationService languageIdentificationService =
            new CMBLanguageIdentificationService();
        CMBCategorizationService categorizationService =
            new CMBCategorizationService();
        CMBCatalogService catalogService = new CMBCatalogService();
        CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
        CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

```

Bean のカスタマイズ

カスタマイズ・セクションで示されているコードは、インストールに応じて調整しなければなりません。接続 Bean に対して、接続するコンテンツ・サーバーの名前、ユーザー ID、および適切なパスワードを指定する必要があります。

カテゴリー化サービス Bean およびカタログ・サービス Bean を実行するには、それらを既存のカタログに関連付ける必要があります。また、カタログ・サービス Bean には、カテゴリー情報が含まれていない項目を割り当てるためのデフォルトのカテゴリーが必要です。

サンプルのためのカスタマイズを実行するコードは、下記のとおりです。

```

        samp1Connection.setServerName(sampDbName);
        samp1Connection.setUserid(sampUserName);
        samp1Connection.setPassword(sampPasswd);
        samp1Connection.setConnectionType
            (CMBConnection.CMB_CONNTYPE_DYNAMIC);
        samp1Connection.setServiceConnectionType
            (CMBConnection.CMB_CONNTYPE_DYNAMIC);

        eipConnection.setServerName(eipDbName);
        eipConnection.setUserid(eipUserName);
        eipConnection.setPassword(eipPasswd);
        eipConnection.setConnectToIKF(true);
        eipConnection.setConnectionType
            (CMBConnection.CMB_CONNTYPE_DYNAMIC);
        eipConnection.setServiceConnectionType
            (CMBConnection.CMB_CONNTYPE_DYNAMIC);

        adapter1.setContentProvider
            (new CMBDefaultContentProvider());
        adapter1.setCatalogName(catalog);
        categorizationService.setCatalogName(catalog);
        catalogService.setCatalogName(catalog);
        adapter2.setCatalogName(catalog);

```

Bean の接続

499 ページの図 42 に、Bean 間でのイベント・フローを示します。

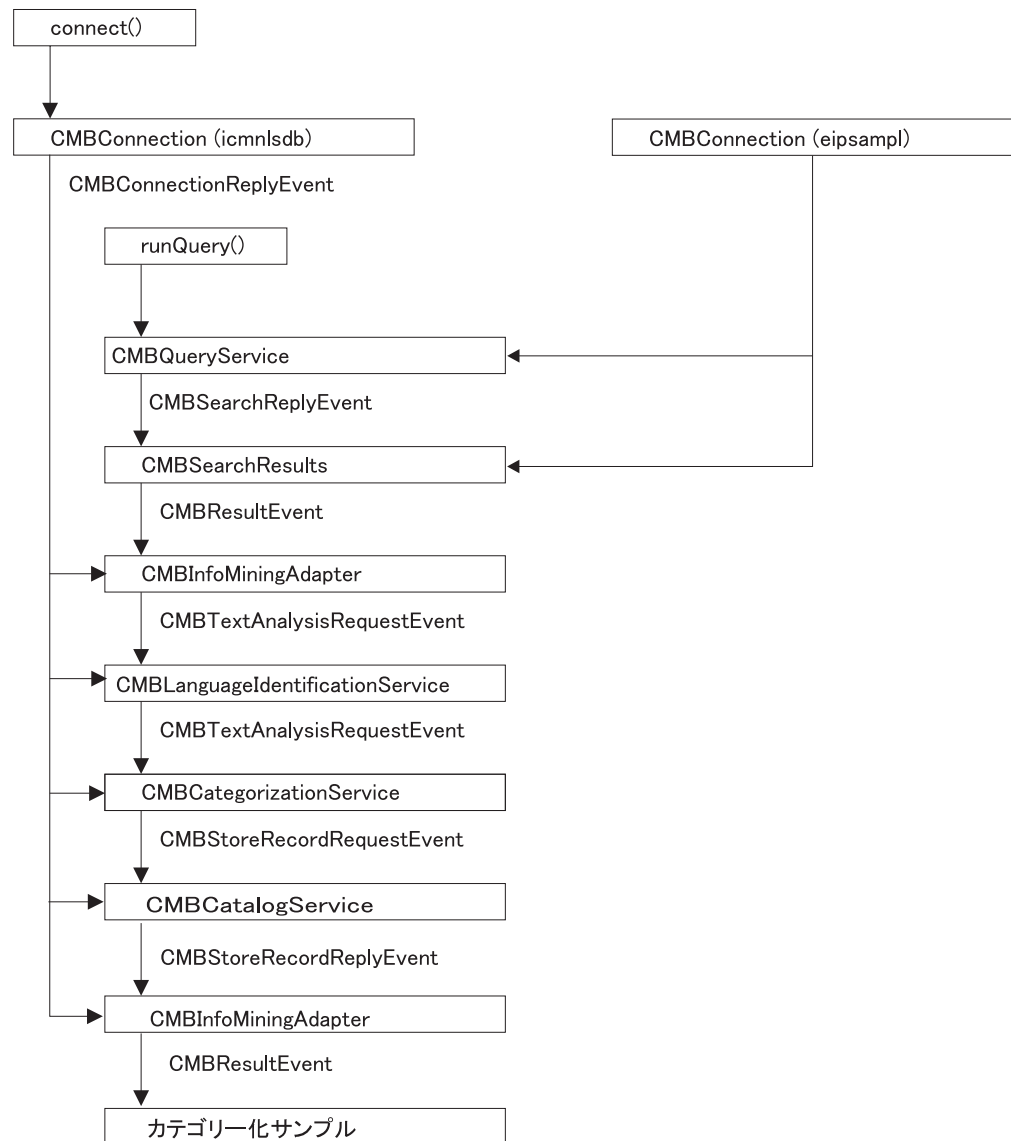


図 42. カテゴリー化サンプル: イベント・フロー

このサンプルで使用される 5 つの Bean は、CMBConnectionReplyEvent を listen し、接続ハンドルを取得します。CMBQueryService Bean は検索を開始し、その結果として、他の Bean にもイベント・フローを開始するイベントが発生します。

Bean を接続するコードは、下記のとおりです。

```

samlConnection.addCMBConnectionReplyListener(queryService);
samlConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
    (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
    (categorizationService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener

```

```

                                (languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
                                (categorizationService);
categorizationService.addCMBStoreRecordRequestListener
                                (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

sampleConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.
    addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

```

例外もイベントとして送られるため、Categorization クラスは、CMBExceptionListener インターフェースをインプリメントすることにより適当なイベントを処理しなければならず、例外に関する通知を受け取るためにその Bean に接続します。

ヒント

カテゴリ化サービス Bean と要約サービス Bean とを任意の順序で使うことによって、文書を分析することができます。これを行う方法については、517 ページの『Web スペースからの文書のインポート』を参照してください。

照会の実行

照会を実行するには、まず CMBConnection Bean で以下のように接続メソッドを呼び出し、コンテンツ・サーバーへの接続を確立する必要があります。

```

sampleConnection.connect();
eipConnection.connect();

```

標準 EIP 検索を開始するには、検索テンプレート、そのテンプレートの基準、比較演算子、そして当然ながら、検索値を指定する必要があります。

下記のコードは、構成に応じて調整する必要があります。

```

catalogService.setDefaultCategoryPath
(catalogService.getTaxonomy().getRootCategory().getPathAsString());
CMBSchemaManagement schema = connection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
                                searchValues);
CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
                                CMB_REQUEST_SEARCH_SYNC, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

```

現行接続をクローズするには、disconnect メソッドを呼び出します。

```

sampleConnection.disconnect();
eipConnection.disconnect();

```


テキスト分析結果の表示

クラス `Categorization` は、検索中に見つかった文書のリスト、および各文書用に作成されたカテゴリ情報を表示できる `CMBResultListener` インターフェースをインプリメントしています。以下のように、`CMBResult` メソッドへの引き数として受け取る `CMBResultEvent` には、`CMBItem` オブジェクトのベクトルが含まれます (各 `CMBItem` オブジェクトは文書を表す)。

```
Vector cmbItemVector = (Vector)e.getData();
```

以下のように、`CMBItem` オブジェクトは文書の `PID` をカプセル化します。

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);  
CMBRecord currentRecord = currentItem.getInfoMiningRecord();  
System.out.println("%n%nPID          : " + currentRecord.getPID());
```

また、以下のテキスト分析の結果も、同様にカプセル化します。

```
System.out.println("Categories : " + currentRecord.getValue("IKF_CATEGORIES"));
```

文書の要約

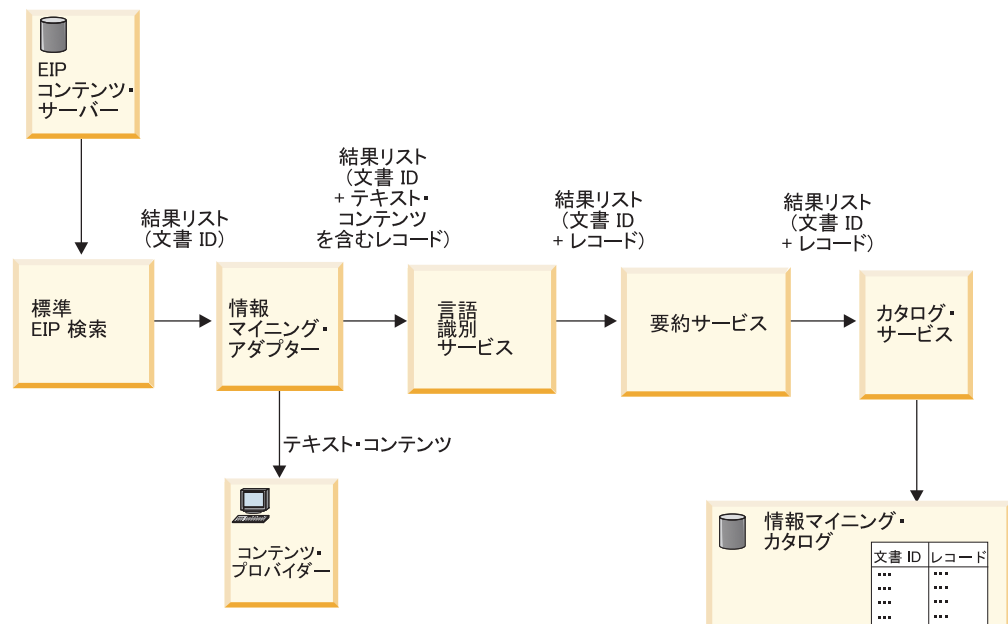


図 43. 要約サンプル

このサンプルは、標準 EIP 検索で取得された文書を要約する方法を示しています。分析結果を保管し、適切な文書を拡張検索できるようにします。

図 43 では、標準 EIP 検索は、EIP コンテンツ・サーバー内に保管されている文書に対して行われます。言語識別サービス Bean は、文書の作成言語を判別します。この Bean は、文書 ID を受け取って文書内容を取り出し、その内容を分析して言語を判別します。要約サービス Bean は英語文書の ID を受け取り、コンテンツ・プロバイダーを使用して検出された文書の内容を取り出し、その要約を作成します。次に、カタログ・サービス Bean が、要約情報をメタデータ・ストアに保管します。その後、この情報 (文書 ID および要約) は、今後のプロセスで使用できるようになります。このサンプルでは、以下の Bean が使用されます。

- CMBConnection
- CMBQueryService (標準 EIP 検索を実行するため)
- CMBSearchResults (標準 EIP 検索を実行するため)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBSummarizationService
- CMBCatalogService

Summarization.java の全ソース・コード

以下に、要約サンプルの全ソース・コードを示します。要約 Bean とカテゴリー化 Bean の動作は類似しているため、詳しくは、493 ページの『文書のカテゴリー化』を参照してください。

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Summarization implements CMBResultListener, CMBExceptionListener
{
    String DEFAULT_CMDBNAME      = "icmnlbdb";
    String DEFAULT_CMDBUSER      = "icmadmin";
    String DEFAULT_CMDBPASSWORD = "password";
    String DEFAULT_SAMPDBNAME    = "eipsampl";
    String DEFAULT_SAMPDBUSER    = "icmadmin";
    String DEFAULT_SAMPDBPASSWORD = "password";

    String CATALOG_NAME          = "Sample";
    String SEARCH_TEMPLATE       = "SearchLongBySource";
    String SEARCH_VALUE          = "ibmpress";
    String SEARCH_CRITERION      = "source";

    public Summarization() throws Throwable
    {
        // creating beans
        CMBConnection samplConnection = new CMBConnection();
        CMBConnection eipConnection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBLanguageIdentificationService languageIdentificationService =
            new CMBLanguageIdentificationService();
    }
}
```

```

CMBSummarizationService summarizationService =
    new CMBSummarizationService();
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// reading parameters
BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Sample database      [" + DEFAULT_SAMPDBNAME + "] : ");
String sampDbName = din.readLine();
if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

System.out.print("User ID for " + sampDbName +
" [" + DEFAULT_SAMPDBUSER + "] : ");
String sampUserName = din.readLine();
if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

System.out.print("Password for " + sampDbName +
" [" + DEFAULT_SAMPDBPASSWORD + "] : ");
String sampPasswd = din.readLine();
if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

System.out.print("EIP database      [" + DEFAULT_CMDBNAME + "] : ");
String eipDbName = din.readLine();
if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

System.out.print("User ID for " + eipDbName
+ " [" + DEFAULT_CMDBUSER + "] : ");
String eipUserName = din.readLine();
if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

System.out.print("Password for " + eipDbName +
" [" + DEFAULT_CMDBPASSWORD + "] : ");
String eipPasswd = din.readLine();
if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

System.out.print("Catalog      [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("¥n¥nRunning Summarization with the following settings:");
System.out.println("Sample database      = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database      = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog      = " + catalog + "¥n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);

```

```

eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider(new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(summarizationService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(summarizationService);
summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);

adapter2.addCMBResultListener(this);

samplConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
samplConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
    getRootCategory().getPathAsString());

CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
    searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
    (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNC,
    searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samplConnection.disconnect();
eipConnection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)

```

```

        return;

        Vector cmbItemVector = (Vector)e.getData();

        if(cmbItemVector == null)
            return;

        try {
            for(int i = 0; i < cmbItemVector.size(); i++)
            {
                CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

                CMBRecord currentRecord = currentItem.getInfoMiningRecord();

                System.out.println("¥n¥nPID      : " + currentRecord.getPID());
                System.out.println("Summary    : " + currentRecord.getValue("IKF_SUMMARY"));
            } /* for */
        }
        catch (CMBNoSuchKeyException nske)
        { nske.printStackTrace();
        }
    }

    //implementing com.ibm.mm.beans.CMBExceptionListener
    public void onCMBException(CMBExceptionEvent e)
    {
        ((Exception)e.getData()).printStackTrace();
    }

    public static void main(String[] args)
    {
        try
        {
            new Summarization();
            System.exit(0);
        }
        catch(Throwable t)
        {
            t.printStackTrace();
        }
    }
}

```

以下に、Bean 間でのイベント・フローを示します。

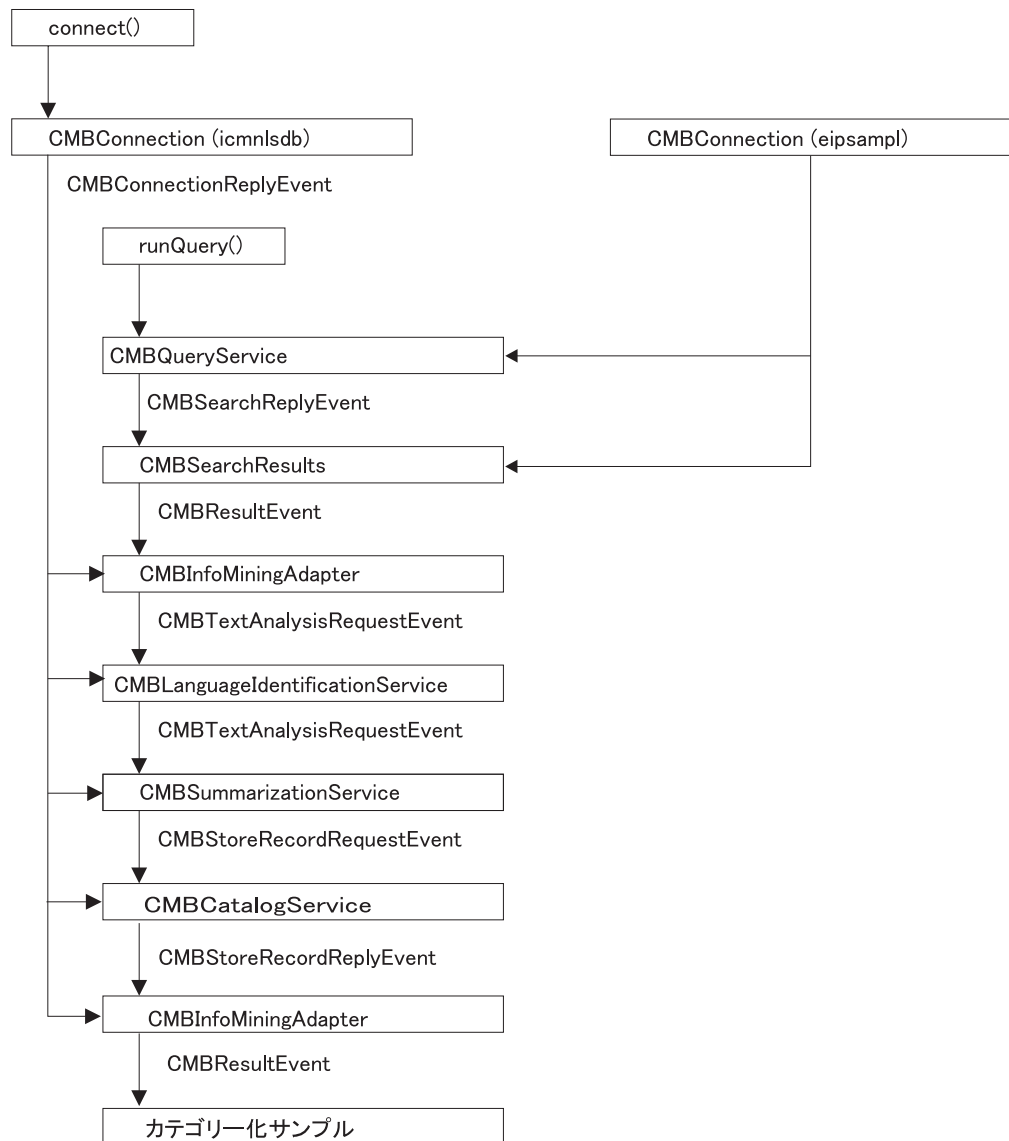


図 44. 要約サンプル: イベント・フロー

情報の抽出

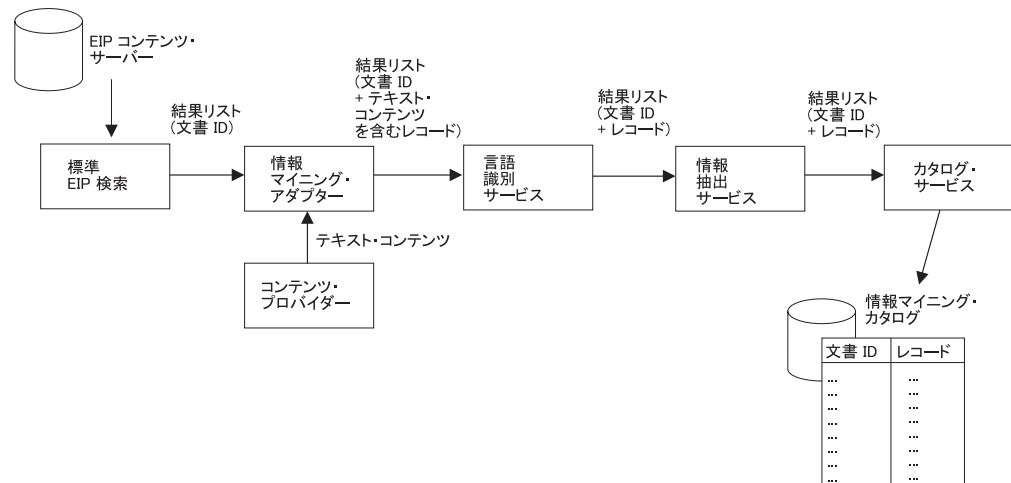


図 45. 情報抽出のサンプル

このサンプルは、標準 EIP 検索で取得された文書から、名前、用語、および式を抽出する方法を示しています。分析結果は保管され、関連文書の取り出し、または個々の文書からの重要情報の抽出に使用できます。

図 45 では、標準 EIP 検索は、EIP コンテンツ・サーバー内に保管されている文書に対して行われます。言語識別サービス Bean は、文書の作成言語を判別します。この Bean は、文書 ID を受け取って文書内容を取り出し、その内容を分析して言語を判別します。情報抽出サービス Bean は英語文書を取り出し、これらの文書から情報を抽出して、その結果をメタデータ・ストアに保管します。これにより、この情報、つまり、文書 ID および抽出された情報を以降の処理で使えるようになります。

このサンプルでは、以下の Bean が使用されます。

- CMBConnection
- CMBQueryService (標準 EIP 検索を実行するため)
- CMBSearchResults (標準 EIP 検索を実行するため)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBInformationExtractionService
- CMBCatalogService

このサンプルにおいて、アプリケーションは、

1. Bean を作成します。
2. Bean をカスタマイズします。
3. 言語識別サービスで文書を分析できるようにするために、Bean を接続します。結果は、カタログに保管されます。
4. 情報抽出サービスを使用して文書を分析します。結果は、カタログに保管されます。

5. 確認のために、検索結果 (文書のリスト) と検査のカテゴリを表示します。

Java ソースは、以下のとおりです。情報抽出 Bean およびカテゴリ化 Bean の動作は類似しているため、詳しくは、493 ページの『文書のカテゴリ化』を参照してください。

InformationExtraction.java の全ソース・コード

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBInformationExtractionService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class InformationExtraction implements
    CMBResultListener, CMBExceptionListener
{
    String DEFAULT_CMDBNAME      = "icmnlsdb";
    String DEFAULT_CMDBUSER      = "icmadmin";
    String DEFAULT_CMDBPASSWORD = "password";
    String DEFAULT_SAMPDBNAME    = "eipsampl";
    String DEFAULT_SAMPDBUSER    = "icmadmin";
    String DEFAULT_SAMPDBPASSWORD = "password";

    String CATALOG_NAME          = "Sample";
    String SEARCH_TEMPLATE      = "SearchLongBySource";
    String SEARCH_VALUE          = "ibmpress";
    String SEARCH_CRITERION     = "source";

    public InformationExtraction() throws Throwable
    {
        // creating beans
        CMBConnection samplConnection = new CMBConnection();
        CMBConnection eipConnection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBLanguageIdentificationService languageIdentificationService =
            new CMBLanguageIdentificationService();
        CMBInformationExtractionService informationExtractionService =
            new CMBInformationExtractionService();
        CMBCatalogService catalogService = new CMBCatalogService();
        CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
        CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

        // reading parameters
        BufferedReader din = new BufferedReader(new InputStreamReader(System.in));
```



```

System.out.print("Sample database          [" + DEFAULT_SAMPDBNAME + "] : ");
String sampDbName = din.readLine();
if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

System.out.print("User ID for " + sampDbName +
" [" + DEFAULT_SAMPDBUSER + "] : ");
String sampUserName = din.readLine();
if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

System.out.print("Password for " + sampDbName +
" [" + DEFAULT_SAMPDBPASSWORD + "] : ");
String sampPasswd = din.readLine();
if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

System.out.print("EIP database          [" + DEFAULT_CMDBNAME + "] : ");
String eipDbName = din.readLine();
if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

System.out.print("User ID for " + eipDbName +
" [" + DEFAULT_CMDBUSER + "] : ");
String eipUserName = din.readLine();
if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

System.out.print("Password for " + eipDbName +
" [" + DEFAULT_CMDBPASSWORD + "] : ");
String eipPasswd = din.readLine();
if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

System.out.print("Catalog          [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("¥n¥nRunning InformationExtraction
                    with the following settings:");
System.out.println("Sample database          = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database          = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog          = " + catalog + "¥n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
samp1Connection.setServerName(sampDbName);
samp1Connection.setUserid(sampUserName);
samp1Connection.setPassword(sampPasswd);
samp1Connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samp1Connection.setServiceConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider(new CMBDefaultContentProvider());

```

```

adapter1.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
    (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
    (informationExtractionService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
    (informationExtractionService);
informationExtractionService.addCMBStoreRecordRequestListener
    (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);

adapter2.addCMBResultListener(this);

samplConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
informationExtractionService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
samplConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
    getRootCategory().getPathAsString());

CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.
    getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION,
    CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
    CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNC, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samplConnection.disconnect();
eipConnection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
        return;

    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)

```

```

        return;

        try {
            for(int i = 0; i < cmbItemVector.size(); i++)
            {
                CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

                CMBRecord currentRecord = currentItem.getInfoMiningRecord();

                System.out.println("¥n¥nPID          : " + currentRecord.getPID());
                System.out.println("Features          : " + currentRecord.getValue("IKF_FEATURES"));
            } /* for */
        }
        catch (CMBNoSuchKeyException nske)
        { nske.printStackTrace();
        }
    }

    // implementing com.ibm.mm.beans.CMBExceptionListener
    public void onCMBException(CMBExceptionEvent e)
    {
        ((Exception)e.getData()).printStackTrace();
    }

    public static void main(String[] args)
    {
        try
        {
            new InformationExtraction();
            System.exit(0);
        }
        catch(Throwable t)
        {
            t.printStackTrace();
        }
    }
}

```

512 ページの図 46 に、Bean 間でのイベント・フローを示します。

このサンプルは、標準 EIP 検索で検索された文書をクラスター化する方法を示しています。クラスター化の前に、文書の言語を識別する必要があります。クラスター化サービスが自動的に収集するのは、EIP 検索から戻された文書のみです。実際のクラスター化のプロセスは、cluster() メソッドを呼び出して、手動で起動する必要があります。

このサンプルでは、以下の Bean が使用されます。

- CMBConnection
- CMBQueryService (標準 EIP 検索を実行するため)
- CMBSearchResults (標準 EIP 検索を実行するため)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBClusteringService

このサンプルにおいて、アプリケーションは、

1. Bean を作成します。
2. Bean をカスタマイズします。
3. 言語識別サービスによって文書を分析して文書の言語を識別し、クラスター化サービスによって、その後のクラスター化に使用できる文書を収集できるよう、Bean を接続します。
4. cluster() メソッドを呼び出して、クラスター化を起動します。(クラス CMBClusterResult の) 結果が画面上に読み取り可能な形式で出力されます。

クラスター化サービスのための Bean は、以下の点を除き、他の情報マイニング Bean と同じです。

- クラスター化サービスが収集するのは、EIP 検索から戻された文書のみです。実際のクラスター化のプロセスは、手動で起動する必要があります。
- カタログは、クラスター化の結果を保管するようには指定されていません。つまり、このシナリオではカタログ・サービスは必要ありません。

Bean の振る舞いについての詳細は、493 ページの『文書のカテゴリ化』を参照してください。

Java ソースは、以下のとおりです。

Clustering.java の全ソース・コード

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;
```

```

import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBClusteringService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBClusterResult;
import com.ibm.mm.beans.infomining.CMBClusterNode;

public class Clustering implements CMBResultListener,
                                   CMBExceptionListener
{
    String DEFAULT_CMDBNAME      = "icmnlsdb";
    String DEFAULT_CMDBUSER      = "icmadmin";
    String DEFAULT_CMDBPASSWORD = "password";
    String DEFAULT_SAMPDBNAME    = "eipsaml";
    String DEFAULT_SAMPDBUSER    = "icmadmin";
    String DEFAULT_SAMPDBPASSWORD = "password";

    String CATALOG_NAME          = "Sample";
    String SEARCH_TEMPLATE       = "SearchLongBySource";
    String SEARCH_VALUE          = "ibmpress";
    String SEARCH_CRITERION      = "source";

    public Clustering() throws Throwable
    {
        // creating beans
        CMBConnection samplConnection = new CMBConnection();
        CMBConnection eipConnection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBLanguageIdentificationService languageIdentificationService =
            new CMBLanguageIdentificationService();
        CMBClusteringService clusteringService = new CMBClusteringService();
        CMBInfoMiningAdapter adapter = new CMBInfoMiningAdapter();

        // reading parameters
        BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Sample database      [" + DEFAULT_SAMPDBNAME + "] : ");
        String sampDbName = din.readLine();
        if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

        System.out.print("User ID for " + sampDbName +
            " [" + DEFAULT_SAMPDBUSER + "] : ");
        String sampUserName = din.readLine();
        if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

        System.out.print("Password for " + sampDbName +
            " [" + DEFAULT_SAMPDBPASSWORD + "] : ");
        String sampPasswd = din.readLine();
        if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

        System.out.print("EIP database      [" + DEFAULT_CMDBNAME + "] : ");
        String eipDbName = din.readLine();
        if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

        System.out.print("User ID for " + eipDbName +
            " [" + DEFAULT_CMDBUSER + "] : ");
        String eipUserName = din.readLine();
        if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

        System.out.print("Password for " + eipDbName +
            " [" + DEFAULT_CMDBPASSWORD + "] : ");
        String eipPasswd = din.readLine();
        if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

        System.out.print("Catalog          [" + CATALOG_NAME + "] : ");

```

```

String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("Running Clustering with the following settings:");
System.out.println("Sample database      = " + sampDbName);
System.out.println("User ID for      " + sampDbName + " = " + sampUserName);
System.out.println("Password for    " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database    = " + eipDbName);
System.out.println("User ID for      " + eipDbName + " = " + eipUserName);
System.out.println("Password for    " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog        = " + catalog + "%n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
samp1Connection.setServerName(sampDbName);
samp1Connection.setUserid(sampUserName);
samp1Connection.setPassword(sampPasswd);
samp1Connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samp1Connection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter.setContentProvider(new CMBDefaultContentProvider());
adapter.setCatalogName(catalog);
clusteringService.setMinClusterCount(2);
clusteringService.setMaxClusterCount(6);
clusteringService.setClusterFeatureCount(5);

// connecting beans
samp1Connection.addCMBConnectionReplyListener(queryService);
samp1Connection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(clusteringService);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter);
adapter.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(clusteringService);

samp1Connection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
clusteringService.addCMBExceptionListener(this);
adapter.addCMBExceptionListener(this);

// running query
samp1Connection.connect();
eipConnection.connect();

CMBSchemaManagement schema = samp1Connection.getSchemaManagement();
CMBSearchTemplate searchTemplate =

```

```

        schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION,
        CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
        (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

// running clustering and printing results
CMBClusterResult clusterResult = clusteringService.cluster();

System.out.println(clusterResult.getClusterCount() +
        " clusters found for " +
        clusterResult.getDocumentCount() +
        " documents:");
CMBClusterNode[] clusterNodes = clusterResult.getClusterNodes();
for(int i = 0; i < clusterNodes.length; i++) {
    CMBClusterNode node = clusterNodes[i];
    String[] features = node.getClusterFeatures();
    String[] names    = node.getDocumentNames();
    String label      = node.getLabel();
    System.out.println("Cluster " + label);
    System.out.print("  Cluster Features : ");
    for(int j = 0; j < features.length; j++) System.out.print(features[j] + " ");
    System.out.println();
    System.out.println("  Documents in cluster :");
    for(int j = 0; j < names.length; j++) System.out.println("    " + names[j]);
}

// disconnecting
samplConnection.disconnect();
eipConnection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    return;
}

// implementing com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new Clustering();
        System.exit(0);
    }
    catch(Throwable t)
    {
        t.printStackTrace();
    }
}
}

```

517 ページの図 48 に、Bean 間でのイベント・フローを示します。

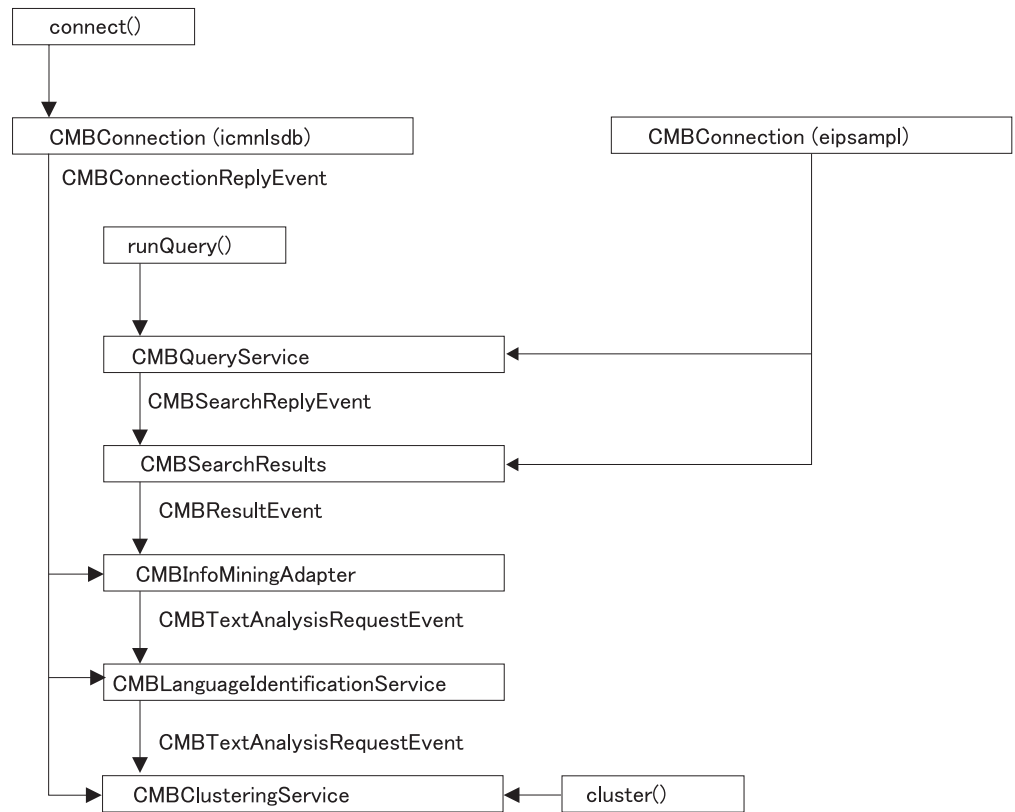


図 48. クラスター化のサンプル: イベント・フロー

Web スペースからの文書のインポート

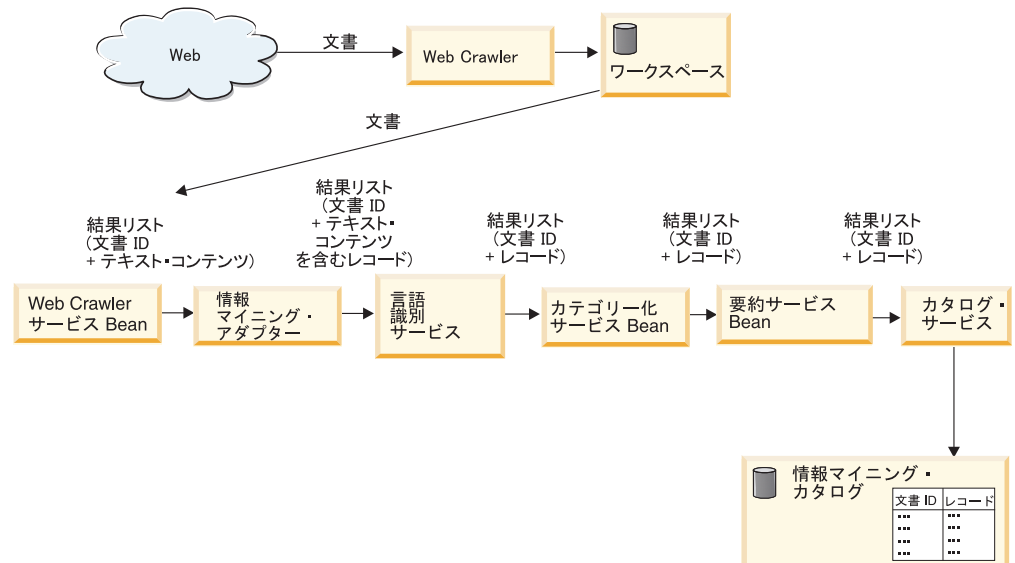


図 49. Web Crawler のサンプル

このサンプルでは、クロールされた文書を情報マイニング・コンポーネントにインポートし、インポートされた文書に対してテキスト分析（言語識別、カテゴリー化、および要約）を実行する方法について説明します。これを実行できるのは、Web

Crawler が実行済みまたは実行中で、新規オブジェクトまたは変更されたオブジェクトが、指定の Web スペース・ディレクトリー (Web Crawler 構成の summaries-dir 属性によって定義される) に保管されている場合のみです。

情報マイニング用に Web Crawler を使用する際には、情報マイニング構成設定値を指定して Web Crawler を実行する必要があります。これらの設定値の説明は、

『Enterprise Information Portal の管理』の情報マイニングの管理に関するセクションにあります。これらの設定値は、情報マイニング・コンポーネント im-crawler-config-sample.xml のサンプル構成ファイルに含まれています。このファイルは、次のディレクトリーにあります。

- Windows の場合:

```
<CMBROOT>%samples%java%beans%infomining%webcrawler%
```

- UNIX (AIX および Solaris) の場合:

```
<CMBROOT>/samples/java/beans/infomining/webcrawler/
```

Web Crawler によってモニターされている Web ページ文書のコピーが指定の Web スペース・ディレクトリーに入れられた後に、CMBWebCrawlerService Bean を使用して、Enterprise Information Portal 情報マイニングにクロール済みの文書をインポートします。

Web Crawler および CMBWebCrawlerService Bean を永続的なプロセスとして実行して文書をモニターし、文書が変更されるごとにその文書をインポートすることができます。CMBResultEvent は、インポートされた文書の数が増えたとき、あるいはモニターされているすべてのファイルがインポートされたときに起動されます。CMBWebCrawlerService Bean は、接続されているすべてのリスナーへの通知を行います。インポートされたファイルは、CMBItems のベクトルとしてイベントに含まれます。

AIX および Solaris 上での CMBWebCrawlerService のアクセス権限。

CMBWebCrawlerService を使用して文書をインポートする際に、対応するファイルはファイル・システムから削除されるか、(archiveEnabled を使用してアーカイブ・オプションをオンにした場合は) disks ディレクトリーから archives ディレクトリーにファイルが移されます。archives ディレクトリーは、disks ディレクトリーと同じレベルにあり (.../webspaces/ikf 内)、そのサブディレクトリー構造も disks ディレクトリーと同じです。CMBWebCrawlerService を使用するには、対応するユーザー ID に次の許可を与える必要があります。

- disks ディレクトリー、およびこのディレクトリー下にあるすべてのサブディレクトリーとファイルの書き込み許可。disks ディレクトリーの場所は、Web Crawler 構成ファイルの summaries-dir に定義されている .../webspaces/ikf/disks です。この許可は、クロール済みの文書とファイルを削除または移動するために必要です。
- アーカイブをオンにした場合は、.../webspaces/ikf の書き込み許可。この許可により、アーカイブを最初に使用するとき archives ディレクトリーを作成できます。
- アーカイブをオンにした場合は、archives ディレクトリーとそのサブディレクトリー、およびすべてのファイルの書き込み許可。この許可により、disks ディレクトリーから移動したファイルを archives ディレクトリー内で作成できます。

インポート操作の変更。 CMBWebCrawlerService Bean で、インポート操作の一部を変更することができます。初期状態では、プログラムはクロール済みの文書を 30 分ごとに検索しますが、この値を変更することができます。また、この Bean は、100 件の文書がインポートされるごとに、すべてのリスナーに CMBResultEvent 通知をスロー (throw) します。この文書の数を決別の数値に変更したり、クロール済みのすべてのファイルが Enterprise Information Portal のセットにインポートされたときに通知をスロー (throw) するように指定することもできます。

CMBWebCrawlerService Java Bean のプロパティ:

pollCycles

ポーリングの回数。

pollMinutes

次のポーリングまでスリープする分数。デフォルトは 30 です。

rootDir

ローカル・ホスト上で %IMY_WEBSPACE% をクロールします。

archiveEnabled

クロール済みファイルを `.../webspaces/ikf/archives` に保管します。デフォルトは `false` で、この場合 CMBWebCrawlerService はクロール済みファイルを処理中に `../webspaces/ikf/disks` から削除し、他の場所へのバックアップは行いません。

pageSize

CMBResultListeners に CMBResultEvent がスロー (throw) されるまでにインポートされる項目の個数。

webSpace

Web Crawler によってモニターされる Web スペース。

他のテキスト分析 Bean の場合と同様に、Web Crawler 検索 Bean の結果は、以降の拡張検索でも使用できます。

このサンプルでは、以下の Bean が使用されます。

- CMBConnection
- CMBWebCrawlerService
- CMBInfoMiningAdapter
- CMBLanguageIdentification
- CMBCategorizationService
- CMBSummarizationService
- CMBCatalogService

このサンプルのアプリケーションは、

1. Bean を作成します。
2. Bean をカスタマイズします。
3. Bean を接続します。
4. Web Crawler サービスを開始します。
5. Crawler の結果およびテキスト分析結果を表示します。

これらの各ステップについては、WebCrawler.java のソースの後で説明します。

WebCrawler.java の全ソース・コード

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBWebCrawlerService;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class WebCrawler implements CMBResultListener, CMBExceptionListener
{
    String CMDBNAME      = "icmnlsdb";
    String CMDBUSER      = "icmadmin";
    String CMDBPASSWORD  = "password";
    String CATALOG       = "Sample";

    String WEBSpace_DIR  = ""; // set to the dir where the web spaces reside
    String WEBSpace_NAME = ""; // set this to the name of your web space

    public WebCrawler() throws Throwable
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
        CMBLanguageIdentificationService languageIdentificationService = new
            CMBLanguageIdentificationService();
        CMBCategorizationService categorizationService = new CMBCategorizationService();
        CMBSummarizationService summarizationService = new CMBSummarizationService();
        CMBCatalogService catalogService = new CMBCatalogService();
        CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
        CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

        // reading parameters
        BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Database [" + CMDBNAME + "] : ");
        String dbName = din.readLine();
        if (dbName.trim().equals("")) dbName = CMDBNAME;

        System.out.print("User name [" + CMDBUSER + "] : ");
        String userName = din.readLine();
        if (userName.trim().equals("")) userName = CMDBUSER;

        System.out.print("Password [" + CMDBPASSWORD + "] : ");
        String passwd = din.readLine();
        if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

        System.out.print("WebSpace directory [" + WEBSpace_DIR + "] : ");
        String webSpaceDir = din.readLine();
        if (webSpaceDir.trim().equals("")) webSpaceDir = WEBSpace_DIR;

        System.out.print("WebSpace name [" + WEBSpace_NAME + "] : ");
        String webSpaceName = din.readLine();
        if (webSpaceName.trim().equals("")) webSpaceName = WEBSpace_NAME;

        System.out.print("Catalog [" + CATALOG + "] : ");
```

```

String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG;

System.out.println("\n\nRunning Summarization with the following settings:");
System.out.println("Database      = " + dbName);
System.out.println("User        = " + userName);
System.out.println("Password    = " + passwd);
System.out.println("Webpace directory = " + webpaceDir);
System.out.println("Webpace name   = " + webpaceName);
System.out.println("Catalog       = " + catalog + "\n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
    connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
    connection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
crawlerService.setRootDirectory(webpaceDir);
crawlerService.setWebSpace(webpaceName);

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// connecting beans
connection.addCMBConnectionReplyListener(crawlerService);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

crawlerService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisRequestListener(summarizationService);
summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
crawlerService.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
connection.connect();
catalogService.setDefaultCategoryPath
(catalogService.getTaxonomy().getRootCategory().getPathAsString());

crawlerService.start();
connection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{

```

```

        if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
            return;

        Vector cmbItemVector = (Vector)e.getData();

        if(cmbItemVector == null)
            return;

        try {
            for(int i = 0; i < cmbItemVector.size(); i++)
            {
                CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

                CMBRecord currentRecord = currentItem.getInfoMiningRecord();

                System.out.println("¥n¥nPID          : " + currentRecord.getPID());
                System.out.println("Categories: " + currentRecord.getValue("IKF_CATEGORIES"));
                System.out.println("Summary    : " + currentRecord.getValue("IKF_SUMMARY"));
            } /* for */
        }
        catch (CMBNoSuchKeyException nske)
        { nske.printStackTrace();
        }
    }

    //implementing com.ibm.mm.beans.CMBExceptionListener
    public void onCMBException(CMBExceptionEvent e)
    {
        ((Exception)e.getData()).printStackTrace();
    }

    public static void main(String[] args)
    {
        try
        {
            new WebCrawler();
            System.exit(0);
        }
        catch(Throwable t)
        {
            t.printStackTrace();
        }
    }
}

```

Bean の作成

情報マイニングではシステムでだれが作業しているかを認識する必要があるので、セキュリティ上の理由からコンテンツ・サーバーへの接続を確立します。これにより、システム内に登録されている人物だけがそのシステムを使用できます。接続の確立には、CMBConnection Bean を使います。CMBWebCrawlerService Bean は、クロール済みの文書を情報マイニング・コンポーネントにインポートしたり CMBResultEvent 通知をスローすることに使用されます。

CMBLanguageIdentification、CMBSummarizationService、および

CMBCategorizationService の各 Bean を使用して、言語識別、要約およびカテゴリー情報が作成されます。2 つのアダプターが、結果イベントをテキスト分析要求イベントに変換した後、レコード応答イベントを検索結果イベントに戻します。

Bean を作成するコードは、以下のとおりです。

```

CMBConnection connection = new CMBConnection();
CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
CMBLanguageIdentificationService languageIdentificationService =
    new CMBLanguageIdentificationService();
CMBCategorizationService categorizationService =
    new CMBCategorizationService();
CMBSummarizationService summarizationService =

```

```

new CMBSummarizationService();
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

```

Bean のカスタマイズ

カスタマイズ・セクションで示されているコードは、インストールに応じて調整しなければなりません。接続するコンテンツ・サーバーの名前、ユーザー ID、および適切なパスワードを指定する必要があります。

Web Crawler サービス Bean に対しては、Web Crawler が Web スペース内でクロールしたオブジェクトを保管または変更した場所であるローカル・ホスト上のルート・ディレクトリーを、既に定義されている Web スペースの名前とともに指定する必要があります。テキスト分析サービス Bean およびカタログ・サービス Bean を実行するには、それらを既存のカタログに事前に関連付ける必要があります。

サンプルのためのカスタマイズを実行するコードは、下記のとおりです。

```

connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
connection.setServiceConnectionType
(CMBConnection.CMB_CONNTYPE_DYNAMIC);
crawlerService.setRootDirectory(webSpaceDir);
crawlerService.setWebSpace(webSpaceName);

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

```

Bean の接続

524 ページの図 50 は、このサンプル内の Bean 間でのイベント・フローを示しています。

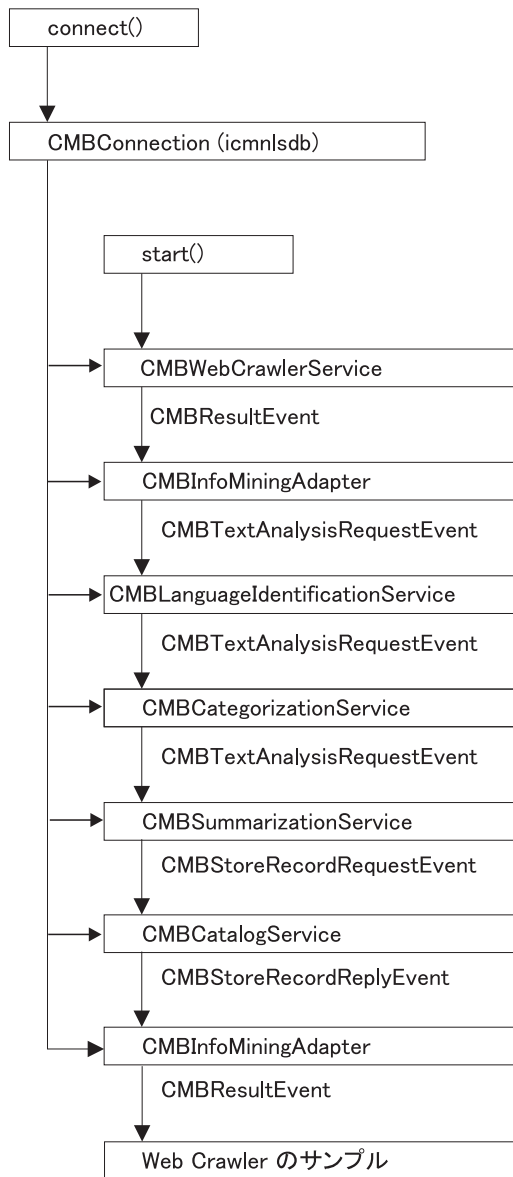


図 50. Web Crawler のサンプル: イベント・フロー

このサンプルで使用する 5 つの Bean は、CMBConnectionReplyEvent を listen し
て接続ハンドルを取得します。CMBWebCrawlerService はクロール・サービスを開
始し、その結果としてその他の Bean にもイベント・フローを開始するイベントが
発生します。

Bean を接続するコードは、下記のとおりです。

```

connection.addCMBConnectionReplyListener(crawlerService);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

crawlerService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.

```



```

        addCMBTextAnalysisRequestListener(categorizationService);
        categorizationService.
            addCMBTextAnalysisRequestListener(summarizationService);
        summarizationService.addCMBStoreRecordRequestListener(catalogService);
        catalogService.addCMBStoreRecordReplyListener(adapter2);
        adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
crawlerService.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

```

例外もイベントとして送られるため、WebCrawler クラスは、CMBExceptionListener インターフェースをインプリメントすることにより適切なイベントを処理しなければならず、例外に関する通知を受け取るためにその Bean に接続します。

Web Crawler サービスの開始

Web Crawler サービスを開始するには、まず CMBCConnection Bean で以下のように接続メソッドを呼び出し、コンテンツ・サーバーへの接続を確立する必要があります。

```
connection.connect();
```

Web Crawler サービスを開始するコードは、以下のとおりです。

```

catalogService.setDefaultCategoryPath
    (catalogService.getTaxonomy().getRootCategory().getPathAsString());
crawlerService.start();

```

現行接続をクローズするには、disconnect() メソッドを呼び出します。

```
connection.disconnect();
```

テキスト分析結果の表示

クラス WebCrawler は、検索中に見つかった文書のリスト、および各文書用に作成されたカテゴリーおよび要約情報を表示できる CMBResultListener インターフェースをインプリメントしています。以下のように、onCMBResult メソッドで引き数として受け取られる CMBResultEvent には、CMBItem オブジェクトのベクトルが含まれます (各 CMBItem オブジェクトは文書を表す)。

```
Vector cmbItemVector = (Vector)e.getData();
```

以下のように、CMBItem オブジェクトは文書の PID をカプセル化します。

```

CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

CMBRecord currentRecord = currentItem.getInfoMiningRecord();

System.out.println("%n%nPID      : " + currentRecord.getPID());

```

また、テキスト分析の結果も、同様にカプセル化します。

次に、以下のようにして要約情報およびカテゴリー情報を取得します。

```

System.out.println("Categories   : " +
    currentRecord.getValue("IKF_CATEGORIES"));
System.out.println("Summary      : " +
    currentRecord.getValue("IKF_SUMMARY"));

```

カテゴリーによる文書の検索

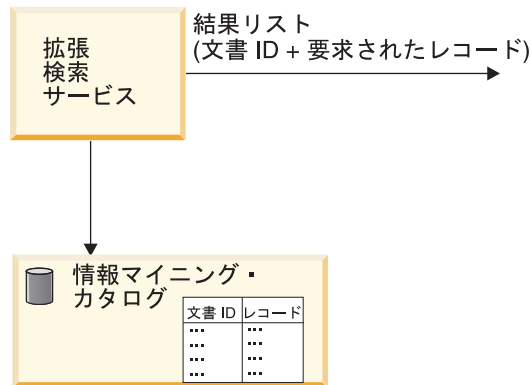


図 51. 拡張検索サンプル

このサンプルでは、拡張検索を実行し、検出された文書内のカテゴリー情報の検索を実行する方法について説明します。カテゴリー化サンプルの場合と同じように、拡張検索で見つけることのできる文書は、この種の検索用に使用可能になっている文書だけです。492 ページの『サンプル・ファイルの位置』を参照してください。前のサンプルでは、標準 EIP 検索が EIP コンテンツ・サーバー全体で実行されたのに対し、このサンプルではデータ・ストア内のメタデータを検索します。

複雑な照会の実行依頼中に問題が生じた場合は、詳細について 542 ページの『パフォーマンス・チューニング』を参照してください。

拡張検索によって結果リストが生成されると、カタログ・サービス Bean は、見つかった文書の ID を使用して、データ・ストア内に保管済みのメタデータを取り出せます。

このサンプルでは、以下の Bean が使用されます。

- CMBConnection
- CMBAdvancedSearchService
- CMBInfoMiningAdapter
- CMBCatalogService

このサンプルのアプリケーションは、

1. Bean を作成します。
2. Bean をカスタマイズします。
3. Bean を接続します。
4. 拡張検索を実行します。
5. 確認のため、検索結果とテキスト分析結果を表示します。

AdvancedSearch.java のソースの後で、最初のステップについてそれぞれ説明します。

AdvancedSearch.java の全ソース・コード

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBException;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBAdvancedSearchService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBCategory;

public class AdvancedSearch implements CMBResultListener,
                                       CMBExceptionListener
{
    String CMDBNAME      = "icmnlsdb";
    String CMDBUSER      = "icmadmin";
    String CMDBPASSWORD = "password";

    String CATALOG      = "Sample";
    String SEARCH_TERM  = "Monitor";

    CMBCatalogService catalogService = null;

    public AdvancedSearch() throws Exception
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBAdvancedSearchService advancedSearchService = new
            CMBAdvancedSearchService();
        catalogService = new CMBCatalogService();

        // reading parameters
        BufferedReader din = new BufferedReader
            (new InputStreamReader(System.in));

        System.out.print("Database [" + CMDBNAME + "] : ");
        String dbName = din.readLine();
        if (dbName.trim().equals("")) dbName = CMDBNAME;

        System.out.print("User name [" + CMDBUSER + "] : ");
        String userName = din.readLine();
        if (userName.trim().equals("")) userName = CMDBUSER;

        System.out.print("Password [" + CMDBPASSWORD + "] : ");
        String passwd = din.readLine();
        if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

        System.out.print("Catalog [" + CATALOG + "] : ");
        String catalog = din.readLine();
        if (catalog.trim().equals("")) catalog = CATALOG;

        System.out.print("Search Term [" + SEARCH_TERM + "] : ");
        String searchTerm = din.readLine();
        if (searchTerm.trim().equals("")) searchTerm = SEARCH_TERM;

        System.out.println("¥n¥nRunning AdvancedSearch with the following settings:");
        System.out.println("Database      = " + dbName);
        System.out.println("User        = " + userName);
        System.out.println("Password    = " + passwd);
    }
}
```

```

System.out.println("Catalog      = " + catalog);
System.out.println("Search Term = " + searchTerm + "¥n");

int key;
do {
    System.out.print("Continue (y/n)? ");
    InputStreamReader inReader = new InputStreamReader(System.in);
    key = inReader.read();
    if (key == 'n') System.exit(0); }
while (key != 'y');

// customizing beans
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
    connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
    connection.setServiceConnectionType
        (CMBConnection.CMB_CONNTYPE_DYNAMIC);

catalogService.setCatalogName(catalog);
advancedSearchService.setCatalogName(catalog);
String[] recordKeys = {"IKF_SUMMARY"};
advancedSearchService.setKeysToBeFetched(recordKeys);
advancedSearchService.setQueryString
    ("(¥"IKF_CONTENT¥" CONTAINS ¥"'" + searchTerm + "'¥")");

// connecting beans
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);

// running query
connection.connect();
advancedSearchService.runQuery();
connection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
        return;

    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)
        return;

    try {
        for(int i = 0; i < cmbItemVector.size(); i++)
        {
            CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

            CMBRecord currentRecord = currentItem.getInfoMiningRecord();

            String summary = (String)currentRecord.getValue("IKF_SUMMARY");

            CMBCategory[] categories =
                catalogService.getCategoriesForRecord(currentItem);
            String categoryString = categories[0].getPathAsString();
            for(int j = 1; j < categories.length; j++)
            {

```

```

        categoryString += ", " + categories[j].getPathAsString();
    }

    System.out.println("\n\nPID      : " + currentRecord.getPID());
    System.out.println("Categories : " + categoryString);
    System.out.println("Summary   : " +
        ((summary == null)? "n/a": summary));
    } /* for */
}
catch (CMBException ex)
{
    ex.printStackTrace();
}
}

// implementing com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new AdvancedSearch();
        System.exit(0);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

Bean の作成

検索を実行するには、コンテンツ・サーバーとの接続が必要です。接続の確立には、CMBConnection Bean を使います。拡張検索を実行するには、CMBAdvancedSearchService Bean が必要です。カテゴリー情報は、CMBCatalogService Bean を使用して取り出すことができます。

Bean を作成するコードは、以下のとおりです。

```

CMBConnection connection = new CMBConnection();
CMBAdvancedSearchService advancedSearchService =
    new CMBAdvancedSearchService();
catalogService = new CMBCatalogService();

```

Bean のカスタマイズ

カスタマイズ・セクションで示されているコードは、インストールに応じて調整しなければなりません。接続するコンテンツ・サーバーの名前、ユーザー ID、および適切なパスワードを指定する必要があります。

拡張検索サービスおよびカタログ・サービス Bean を実行するには、それらを既存のカタログに関連付ける必要があります。

サンプルのためのカスタマイズを実行するコードは、下記のとおりです。

```

connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);

```

```

connection.setServiceConnectionType
    (CMBCConnection.CMB_CONNTYPE_DYNAMIC);

catalogService.setCatalogName(catalog);
advancedSearchService.setCatalogName(catalog);
String[] recordKeys = {"IKF_SUMMARY"};
advancedSearchService.setKeysToBeFetched(recordKeys);
advancedSearchService.setQueryString
    ("(¥"IKF_CONTENT¥" CONTAINS ¥" + searchTerm + "¥)");

```

Bean の接続

図 52 は、このサンプル内の Bean 間でのイベント・フローを示しています。

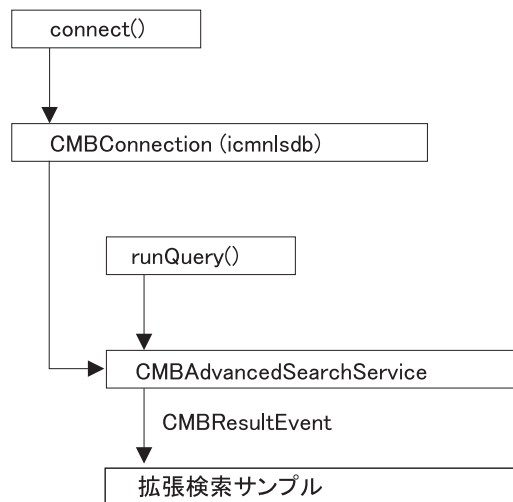


図 52. 拡張検索サンプル: イベント・フロー

このサンプルで使用する 2 つの Bean は、CMBCConnectionReplyEvent を listen し、接続ハンドルを取得します。CMBAdvancedSearchService Bean が検索を開始すると、その結果イベントが発生し、これにより他の Bean にもイベント・フローが発生します。

Bean を接続するコードは、下記のとおりです。

```

connection.addCMBCConnectionReplyListener(catalogService);
connection.addCMBCConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);

```

例外もイベントとして送られるため、AdvancedSearch クラスは、CMBExceptionListener インターフェースを実装することにより適切なイベントを処理しなければならず、例外に関する通知を受け取るためにその Bean に接続します。

照会の実行

照会を実行するには、まず CMBCConnection Bean で以下のように接続メソッドを呼び出し、コンテンツ・サーバーへの接続を確立する必要があります。

```
connection.connect();
```

拡張検索を開始するには、カタログ、検索照会ストリングおよび検索するメタデータを指定する必要があります。

次のようにして拡張検索を開始します。

```
advancedSearchService.runQuery();
```

現行接続をクローズするには、disconnect() メソッドを呼び出します。

```
connection.disconnect();
```

テキスト分析結果の表示

AdvancedSearch クラスは、検索中に見つかった文書をリストしたり、各文書ごとに取り出されたカテゴリー情報を表示したりできる CMBResultListener インターフェースをインプリメントします。以下のように、onCMBResult メソッドで引き数として受け取られる CMBResultEvent には、CMBItem オブジェクトのベクトルが含まれます (各 CMBItem オブジェクトは文書を表す)。

```
Vector cmbItemVector = (Vector)e.getData();
```

以下のように、CMBItem オブジェクトは文書の PID をカプセル化します。

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
```

```
CMBRecord currentRecord = currentItem.getInfoMiningRecord();  
String summary = (String)currentRecord.getValue("IKF_SUMMARY");
```

```
CMBCategory[] categories =  
    catalogService.getCategoriesForRecord(currentItem);  
String categoryString = categories[0].getPathAsString();  
for(int j = 1; j < categories.length; j++)  
{  
    categoryString += ", " + categories[j].getPathAsString();  
}
```

```
System.out.println("¥n¥nPID      : " + currentRecord.getPID());
```

イベント・フロー内にテキスト分析 Bean があれば、その Bean の結果も同じように処理されます。

次に、次のようにしてカテゴリー情報を取得します。

```
System.out.println("Categories      :  
    " + currentRecord.getValue("IKF_CATEGORIES"));
```

クラス CMBItem の getCategories() メソッドによって戻されるベクトルには、クラス CMBCategory のオブジェクトが含まれます。これを使用して、現在のカテゴリーへの絶対パスを判別することができます。

独自のコンテンツ・プロバイダーの作成

独自形式のバイナリー部分からテキストを検索するために独自のフィルターを適用する必要がある場合には、独自のコンテンツ・プロバイダーを作成することができます。

ここでは、ユーザー定義オブジェクト・モデルを処理したり、 `CMBItem` のパーツ内の独自形式を処理したりできる、独自のコンテンツ・プロバイダーを作成する方法について説明します。この作業をサポートするために、情報マイニングには下記の機能が用意されています。

- インターフェース **`CMBContentProvider`**。これは、テキスト分析のために使用されるテキストを判別する方法を認識しているクラスのインターフェースを定義します。
- `CMBInfoMiningUtilities` 内のメソッド **`setContentProvider(CMBContentProvider)`**。これは、`ContentProvider` を設定します。

`CMBContentProvider` インターフェースは、テキスト分析で使用される指定された項目のテキストを戻す 1 つのメソッド `getContent()` を定義します。例えば、以下のようになります。

```
public CMBTextAnalysisDocument getContent(CMBConnection connection, CMBItem item )
throws CMBContentProviderException;
```

- パラメーター `connection`: サーバーへのオープン接続。
- パラメーター `item`: 処理する現在項目。
- 例外 `CMBContentProviderException`: 現在項目の処理中にエラーが発生した場合。
- クラス `CMBTextAnalysisDocument` のオブジェクトとして、テキストを戻します。

使用する `ContentProvider` をシステムに対して指定するには、`CMBInfoMiningUtilities` クラス内のメソッド `setContentProvider(CMBContentProvider)` を使用し、このインターフェースを持つオブジェクトを指定します。例えば、以下のようになります。

```
CMBInfoMiningUtilities.setContentProvider
    (new MyCompaniesLatestGreatestContentProvider());
```

独自のコンテンツ・プロバイダーを開発するには、その出発点としてサンプルのコンテンツ・プロバイダー (`SimpleContentProvider`) を使用することができます。コンテンツ・プロバイダーのサンプルは、
<CMBROOT>%samples%java%beans%infomining%contentprovider にあります。

情報マイニングには、デフォルトのコンテンツ・プロバイダーが用意されています。これは、処理するための個々のパーツを選択できるよう `CMBContentProvider` インターフェースを拡張し、ビデオ・ストリームなど、メモリー内の処理が大きすぎるオブジェクトの処理をしないようにするメカニズムを提供します。

デフォルトのコンテンツ・プロバイダーを登録するには、以下のとおりにします。

```
CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
```

サービス API の使用

情報マイニング・サービス API は、Java API で、情報マイニング機能を EIP サービスとして統合します。情報マイニング・サービス API を使用すると、以下の機能を持つアプリケーションを作成することができます。

- PDF、Microsoft Word および HTML など、異なる文書フォーマットからテキスト内容を取り出す
- テキスト文書に対してテキスト分析を実行し、要約やカテゴリーなどのメタデータを作成する
- 永続レコードに、文書のメタデータを保管する
- レコードを検索する

ご注意

情報マイニング Bean を使用する前に、サービス API と JavaBeans との違いを理解することが重要です。

- JavaBeans には、RAD (rapid application development) のための機能が備わっています。コードのいくつかの要素は「結合」されているので、ユーザーが変更することはできません。詳細については、489 ページの『Bean を使用した情報マイニング・アプリケーションの作成』を参照してください。
- 情報マイニング・サービス API を使用すると、情報マイニング・アプリケーションをより柔軟に作成することができます。コードは独立した「ビルディング・ブロック」として見ることができ、それぞれを組み合わせることにより、特定の要件を満たすことができます。

情報マイニング・サービス API への接続

情報マイニング・サービス API を使用する前に、DKIKServiceFed クラスを使用してサービス・オブジェクトを作成する必要があります。必要としているアプリケーションのタイプに応じて、以下の 3 つのパッケージの 1 つから、クラスをインポートする必要があります。

- サーバー・ホスト上でのアプリケーションを作成する場合は、`import com.ibm.mm.sdk.server.DKIKServiceFed;` というインポート・ステートメントを使用します。
- クライアント・ホスト上でのアプリケーションを作成する場合は、`import com.ibm.mm.sdk.client.DKIKServiceFed;` というインポート・ステートメントを使用します。
- 柔軟性を保ったまま、クライアント上だけでなくサーバー上でも実行可能なアプリケーションを作成する場合は、`import com.ibm.mm.sdk.cs.DKIKServiceFed;` というインポート・ステートメントを使用します。

該当するインポート・ステートメントを使用した後、以下のようして `ikfService` オブジェクトを作成します。

```
DKIKService ikfService = DKIKServiceFed.create();
```

サービス・オブジェクトを作成した後、名前、ユーザー ID、およびパスワードを使用してデータベースに接続することができます。

```
ikfService.connect("databaseName", "userID", "password", null);
```

以上で、情報マイニング・サービス API に接続できたので、以下の 2 つのパッケージのクラスを使用することができます。

- ライブラリー、カタログ、およびレコード・タスクを管理する場合は、**com.ibm.mm.sdk.common.infomining**。
- さまざまなテキスト分析ツールを使用する場合は、**com.ibm.mm.sdk.common.infomining.analysis**。

これらの重要な機能については、以降のセクションを参照してください。

接続を切断するには、以下のとおりにします。

```
ikfservice.disconnect();
```

ライブラリー、分類法、およびカタログの管理

ライブラリー、カタログ、および分類法のタスクを管理するためのクラスは、**com.ibm.mm.sdk.common.infomining** パッケージに含まれています。

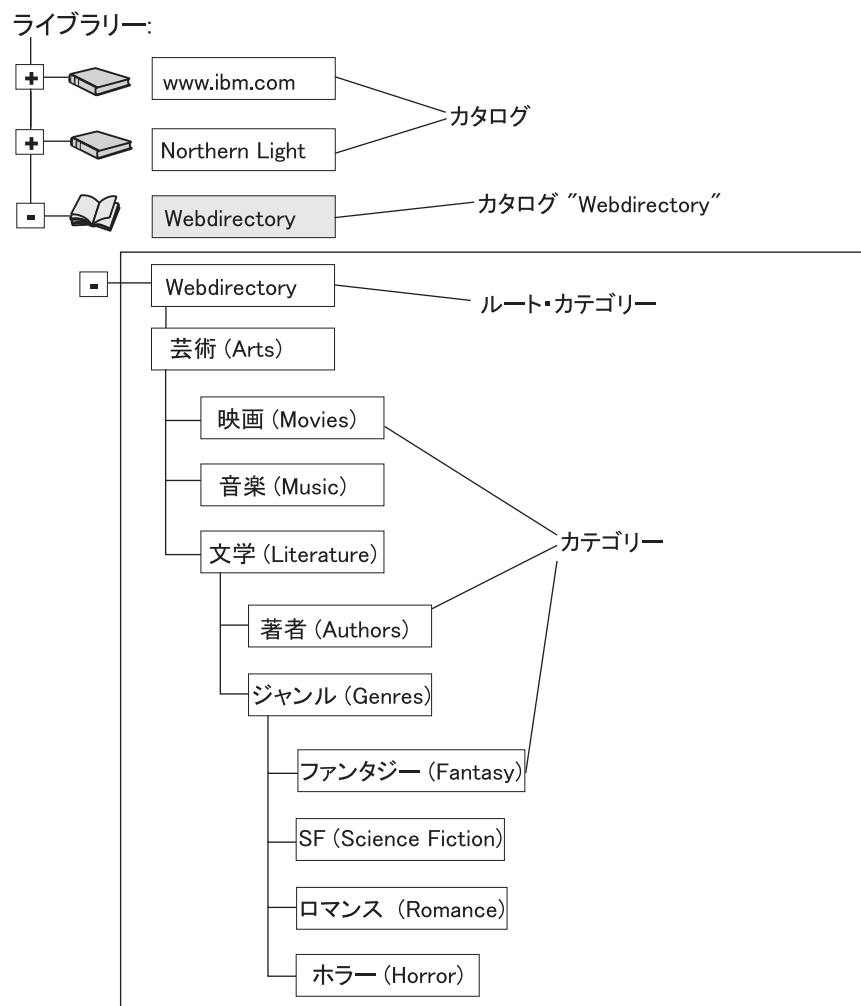


図 53. カタログの例

カタログの例では、ライブラリーに **www.ibm.com**、**Northern Light**、および **Webdirectory** という 3 つのカタログが含まれています。**Webdirectory** のカテゴリは、図で示されているとおり、ツリー構造をしています。

カタログは、Information Structuring Tool (IST) でしか作成できません。

ライブラリーを戻す

ライブラリー・オブジェクトを取得するには、以下のとおりにします。

```
DKIKFLibrary library = ikfService.getLibrary();
```

ライブラリー内の全カタログをリストする

ライブラリー内のすべてのカタログの名前を取得するには、以下のとおりにします。

```
String[] catalogNames = library.getCatalogNames();
```

これは、カタログ名をすべてリストした配列を戻します。この例では、カタログ名 Webdirectory、www.ibm.com および Northern Light が順不同で戻されます。

特定のカタログを戻す

特定のカタログを取得するには、以下のとおりにします。

```
DKIKFCatalog catalog = library.getCatalog(catalogNames[0]);
```

このサンプルでは、前のステップで戻されたリストの最初の名前のカタログが戻されます。この場合は、Webdirectory カatalogが戻されます。

カタログの分類法を戻す

カタログの分類法を取得するには、以下のとおりにします。

```
DKIKFTaxonomy taxonomy = catalog.getTaxonomy();
```

分類法オブジェクトは、カテゴリーを操作するためのさまざまなメソッドを備えています。

ご注意

分類法オブジェクトはデータベースから読み取られ、以降のアクションはすべてこのコピーに対して行われます。

分類法内のすべてのカタログをリストする

分類法内のすべてのカテゴリーを取得するには、以下のとおりにします。

```
DKIKFCategory[] categories = taxonomy.getCategories();
```

これは、ルート・カテゴリーを含む、すべてのカテゴリーをリストした配列を戻します。この例では、ホラー (Horror)、著者 (Author)、音楽 (Music)、文学 (Literature)、Webdirectory、ジャンル (Genres)、SF (Science Fiction)、ファンタジー (Fantasy)、映画 (Movies)、ロマンス (Romance)、および芸術 (Arts) が順不同で戻されます。

ご注意

カテゴリー・オブジェクトはデータベースから読み取られ、以降のアクションはすべてこのコピーに対して行われます。

ルート・カテゴリを戻す

ルート・カテゴリを取得するには、以下のとおりにします。

```
DKIKFCategory rootCategory = taxonomy.getRootCategory();
```

このサンプルでは、前のステップで戻されたリストのルート・カテゴリが戻されます。この場合は、Webdirectory ルート・カテゴリが戻されます。

特定のカテゴリを戻す

特定のカテゴリを戻すには、以下のとおりにします。

```
DKIKFCategory category = taxonomy.getCategory("Webdirectory/Literature/Genres");
```

この例を使用すると、ジャンル (Genres) カテゴリが戻されます。

また、現在分類法オブジェクトで設定されているものと同じ区切り文字 "/" を使用します。

ご注意

分類法オブジェクトでは 3 つの getCategory メソッドを使用することができます。各メソッドでは、以下のようにそれぞれ異なるパラメーターが必要です。

- パスは、指定された文字で区切られたカテゴリ名を持つストリングです。
- パスは、カテゴリ名の配列です。
- 別のカテゴリ・オブジェクト。

詳しくは、「オンライン API 解説書」を参照してください。

現在の分類法を最新のものに更新する

現在の分類法を最新ののものにするには、以下のとおりにします。

```
if(taxonomy.getTimestamp().before(catalog.getTaxonomyLastModified()))
{
    taxonomy = catalog.getTaxonomy();
}
```

ご注意

分類法オブジェクトはデータベースから読み取られ、以降のアクションはすべてこのコピーに対して行われます。したがって、分類法オブジェクトは最新のものであるように確認してください。

カテゴリの子を戻す

カテゴリの子を戻すには、以下のとおりにします。

```
DKIKFCategory[] children = category.getChildren();
```

これは、カテゴリのすべての子をリストした配列を戻します。この例はジャンル (Genres) カテゴリを基にしたものであり、ファンタジー (Fantasy)、SF (Science Fiction)、ロマンス (Romance)、およびホラー (Horror) という子を順不同で戻します。

ご注意

1 つのカテゴリに複数の子が存在している場合がありますが、これらの子よりも下のサブカテゴリは戻されません。

カテゴリの親を戻す

カテゴリの親を戻すには、以下のとおりになります。

```
DKIKFCategory parent = category.getParent();
```

この例はジャンル (Genres) カテゴリを基にしており、文学 (Literature) カテゴリのみを戻します。

特定のカatalog・スキーマを戻し、そのスキーマ内のすべての属性をリストする

特定のカatalog・スキーマを取得するには、以下のとおりになります。

```
DKIKFSchema schema = catalog.getSchema();
Iterator keys = schema.keySet().iterator();

while(keys.hasNext())
{
    String key = (String)keys.next();
    System.out.println("key: " + key + "type: " + schema.getType(key).getName());
}
```

この例では、Webdirectory カatalogが戻されます。

次に、スキーマからキー (属性名) を戻すことができます。詳しくは、「[オンライン API 解説書](#)」を参照してください。

ご注意

スキーマ・オブジェクトはデータベースから読み取られ、以降のアクションはすべてこのコピーに対して行われます。

情報マイニング・ツールの使用

情報マイニング・ツールを使用するためのクラスは、

com.ibm.mm.sdk.common.infomining.analysis パッケージに含まれています。以下の場合にこれらのツールを使用します。

- 文書要約を生成する
- 文書のカテゴリを判別する
- 文書の言語を判別する
- 文書からの名前、用語、式などの情報を抽出する
- 文書セットをクラスター化する

これらのツールは、DKIKFTextDocument クラスのオブジェクトであるテキスト文書进行处理します。テキスト文書オブジェクトを作成するには、以下の 2 つの方法があります。

- 文書内容がすでに Java のストリングとして存在している場合は、DKIKFTextDocument クラスの create メソッド
- 文書内容がフォーマット設定された形式の場合は、DKIKFDocumentFilter クラス
どちらのクラスも、**com.ibm.mm.sdk.common.infomining** パッケージに含まれています。

テキスト文書

文書内容が Java のストリングとして存在している場合にテキスト文書オブジェクトを作成するには、以下のとおりにします。

```
DKIKFTextDocument document = DKIKFTextDocument.create("the document content");
```

ご注意

3 つの異なる create メソッドを使用すると、以下を行うことができます。

- 指定された内容を持つテキスト文書の作成
- 指定された内容と名前を持つテキスト文書の作成
- 指定された内容、名前、および言語を持つテキスト文書の作成

詳しくは、「オンライン API 解説書」を参照してください。

文書のフィルター操作

フォーマット設定された文書からテキスト文書オブジェクトを作成するには、以下のとおりにします。

```
DKIKFDocumentFilter documentFilter = new DKIKFDocumentFilter(ikfService);
byte[] documentBytes = ... //set the bytes of the document from a data source
DKIKFTextDocument document2 = documentFilter.getTextDocument(documentBytes);
```

オリジナルのテキスト文書は、サポートされている形式であれば任意の形式 (例えば、PDF や Microsoft Word 文書) で構いません。

フィルターのエンコード方式を設定するには、文書フィルター・オブジェクトのメソッドを使用します。詳しくは、「オンライン API 解説書」を参照してください。

文書言語の判別

文書の言語を判別するには、以下のとおりにします。

```
DKIKFLanguageIdentifier languageIdentifier =
    new DKIKFLanguageIdentifier(ikfService);
DKIKFLanguageIdentificationResult[] languageResults =
    languageIdentifier.analyze(document);
document.setLanguage(languageResults[0].getLanguage());
```

これは、言語および信頼性の値の結果を含むすべてのオブジェクトを、信頼性の値が高い順にリストした配列で戻します。

ご注意

要約サービスやカテゴリー化サービスなど、他の情報マイニング・ツールを使用するには、テキスト文書上で言語を設定する必要があります。

信頼性の値および言語を戻すには、以下のように言語結果オブジェクトのメソッドを使用します。

```
string language = languageResults[0].getLanguage();  
float confidence = languageResults[0].getConfidence();
```

言語識別サービスおよびデフォルト設定についての詳細は、「オンライン API 解説書」を参照してください。

文書要約の生成

文書要約を作成するには、以下のとおりにします。

```
DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);  
DKIKFSummarizationResult summarizationResult = summarizer.analyze(document);
```

要約を戻すには、以下のとおりにします。

```
string summary = summarizationResult.getSummary();
```

要約サービスおよびデフォルト設定についての詳細は、「オンライン API 解説書」を参照してください。

文書情報の抽出

文書から情報を抽出するには、以下のとおりにします。

```
DKIKFInformationExtractor extractor = new DKIKFInformationExtractor(ikfService);  
DKIKFInformationExtractionResult information = extractor.analyze(document);
```

追加の分析機能を調べるには、情報抽出結果オブジェクトのメソッドを使用します。

```
DKIKFFeature[] features = information.getFeatures();
```

情報抽出サービスおよびデフォルト設定についての詳細は、「オンライン API 解説書」を参照してください。

クラスター化

文書はクラスター化することができます。以下のとおりにすると、類似の文書をグループ化することができます。

```
DKIKFClusterer clusterer = new DKIKFClusterer();  
clusterer.analyze(doc1);  
clusterer.analyze(doc2);  
clusterer.analyze(doc3);  
DKIKFClusterResult clusterResult = clusterer.cluster();
```

クラスターのノード、クラスターの数、および文書の総数を戻すには、クラスター結果オブジェクトのメソッドを使用します。

```
int clusterCount = clusterResult.getClusterCount();  
DKIKFClusterNode[] nodes = clusterResult.getClusterNodes();  
int documentCount = clusterResult.getDocumentCount();
```

クラスター化サービスおよびデフォルト設定についての詳細は、「[オンライン API 解説書](#)」を参照してください。

カテゴリー化

文書にカテゴリーを割り当てるには、以下のとおりにします。

```
DKIKFCategorizer categorizer = new DKIKFCategorizer(catalog);  
DKIKFCategorizationResult[] categorizationResults = categorizer.analyze(document);
```

これは、カテゴリーおよび信頼性の値を含むすべてのカテゴリー化結果オブジェクトを、信頼性の値が高い順にリストした配列で戻します。

信頼性の値およびカテゴリーを戻すには、以下のようにカテゴリライザー結果オブジェクトのメソッドを使用します。

```
DKIKFCategory bestCategory = categorizationResults[0].getCategory();
```

ご注意

カテゴリー化サービスを使用するには、最初に Information Structuring Tool (IST) を使用してカタログを作成し、準備する必要があります。カテゴリー化サービスは、1 つのカタログを使用します。

戻されたカテゴリー・オブジェクトは、分類法オブジェクトには属さないことにも注意してください。親および子のカテゴリーを横断するには、分類メソッドを使用します。

カテゴリー化サービスおよびデフォルト設定についての詳細は、「[オンライン API 解説書](#)」を参照してください。

レコードの作成およびカタログへのメタデータの保管

レコードの作成およびメタデータの保管を行うためのクラスは、**com.ibm.mm.sdk.common.infomining** パッケージに含まれています。以下の場合にこれらのクラスを使用します。

- カatalogに新規レコードを作成する
- カatalogからレコードを取り出す
- レコードのカテゴリーを戻す
- レコード値を更新する
- レコード・カテゴリーの割り当てを更新する
- レコードを削除する

情報マイニング・サービス API が管理できるカタログは、Information Structuring Tool (IST) によって作成されたもののみです。一度作成すると、レコードおよびメタデータを追加することができます。

カタログに新規レコードを作成する

レコードは PID およびカタログ・スキーマを使用して、以下のようにして作成されます。


```
DKIKFRecord record = DKIKFRecord.create("PID", schema);
record.setValue("IKF_TITLE", "This is the title");
record.setValue("IKF_SUMMARY", "This is the document summary");
record.setValue("IKF_CONTENT", "This is the document content");
DKIKFCategory[] categoriesParam = {bestCategory};
catalog.createRecord(record, categoriesParam);
```

この例では、レコードのタイトル、要約、および内容の値が設定されます。レコードは上で設定された値を使用して作成され、1 つまたは複数のカテゴリに割り当てることができます。

カタログからレコードを取り出す

PID を使用してカタログからレコードを取り出すには、以下のように入力します。

```
DKIKFRecord retrievedRecord = catalog.getRecord("PID");
```

レコードのカテゴリを戻す

レコードのカテゴリを戻すには、下記のとおりになります。

```
DKIKFCategory[] recordCategories = catalog.getCategoriesForRecord("PID");
```

カタログ内のレコード値を更新する

レコード値 (例: レコードのタイトル) を更新するには、下記のとおりになります。

```
record.setValue("IKF_TITLE", "This is the new title");
catalog.updateRecord(record);
```

ご注意

3 つの異なる update メソッドを使用すると、以下を行うことができます。

- レコード値のみを持つレコードを更新する (上のタイトルの例を参照)
- レコード値を持ち、カテゴリが割り当てられているレコードを更新する
- カテゴリが割り当てられているレコードのみを更新する (1 つのカテゴリ内の 1 つのレコードが別のカテゴリに追加されている以下の例を参照。)

詳しくは、「オンライン API 解説書」を参照してください。

1 つのカテゴリ内の 1 つのレコードを別のカテゴリに追加する

レコードを別のカテゴリに追加するには、下記のとおりになります。

```
DKIKFCategory oldCategories = catalog.getCategoriesForRecord("PID");
DKIKFCategory newCategory = taxonomy.getCategory("Webdirectory/arts/movies");
List categoriesList = Arrays.asList(oldCategories);
categoriesList.add(newCategory);
catalog.updateRecord("PID",
    categoriesList.toArray(new DKIKFCategory[categoriesList.size()]));
```

この例では、新しい映画 (Movies) カテゴリにレコードが追加されます。元のカテゴリ割り当てを保持する場合は、上の例に示されるように、レコードの更新呼び出しに元のカテゴリを含めます。新規のカテゴリに対してレコードの更新呼び出しを行うと、元のカテゴリ割り当てはすべて除去されます。

レコードの更新には 3 つのメソッドを使用します。詳細については、『カタログ内のレコード値を更新する』を参照してください。

レコードは、ルート・カテゴリーおよび複数のカテゴリーに追加することもできます。詳しくは、「オンライン API 解説書」を参照してください。

保管されているレコードを削除する

レコードを削除するには、以下のとおりにします。

```
catalog.deleteRecord("PID");
```

文書の検索

レコードを検索するためのクラスは、**com.ibm.mm.sdk.common.infomining** パッケージに含まれています。

特定の語を含むレコードを検出する

例えば、カテゴリー「音楽 (Music)」内の「バッハ (Bach)」という語を含むレコードを探すには、以下のとおりにします。

```
String queryString = "(¥IKF_CONTENT¥ contains ¥Bach¥) and  
  (DKIKF_CATEGORY = ¥Webdirectory/Music¥)";  
DKIKFSearchConfiguration searchConfiguration = new DKIKFSearchConfiguration();  
searchConfiguration.setTaxonomy(taxonomy);  
DKIKFSearchResult searchResult =  
  catalog.searchRecords(queryString, searchConfiguration);  
Iterator resultPIDs = searchResult.iterator();
```

カタログ・オブジェクトの `searchRecords` メソッドを実行するには、以下のものがが必要です。

- 照会ストリング
- 検索構成オブジェクト

検索構成オブジェクトは、例えば、検索結果の最大数といった検索のプロパティを指定します。照会ストリングでカテゴリーを使用する場合は、分類法オブジェクトも必要になります。

検索結果は PID ストリングまたはレコードにすることができ、`DKIKFSearchConfiguration` 呼び出しで指定することができます。

パフォーマンス・チューニング

複雑な照会を実行依頼すると、特に照会ストリングに複数の OR 演算子が含まれている場合に、パフォーマンス上の問題が生じることがあります。パフォーマンスを高めるには、deMorgan の法則を使用して OR 式を変換することにより、複雑な照会に OR 演算子が含まれないようにします。単項の + と - の演算子を使用して、複雑なテキスト検索照会を作成します。これらの式を情報マイニング照会言語に変換するためのクラスは、サービス API を使用するアプリケーションの場合は `com.ibm.mm.sdk.common.infomining.DKIKFWebQueryConverter` にあります。Bean を使用するアプリケーションの場合は、`CMBAdvancedSearchService.convertWebQuery` メソッドを使用します。

サーバー・タスクの実行

定義されているタスクのシーケンスをクライアント・アプリケーションから実行する場合は、1 つのサーバー・タスクにこれらの呼び出しをまとめ、それを一度サーバーに送ると、必要に応じて、このサーバー・タスクをクライアントから呼び出す

ことができます。これにより、ネットワーク・トラフィックのレベルが可能な限り低く保たれ、パフォーマンスが大幅に改善されます。

サーバー・タスクは、インターフェース

`com.ibm.mm.sdk.common.infomining.DKIKFServerTask` をインプリメントするオブジェクトです。このオブジェクトはクライアント・アプリケーションからインスタンス生成され、`setServerTask` メソッドを使用してサービス・オブジェクトに設定され、次に `runServerTask` メソッドを使用してサーバー上で実行されます。

下の例では、サーバー・タスク・インターフェースをインプリメントしているクラス `AnalysisTask` のオブジェクトがサービスに設定され、実行されます。

```
...
ikfService.setServerTask(new AnalysisTask(secondCatalog.getName()));
HashMap documentMap = new HashMap();
documentMap.put("PID1", DKIKFTextDocument.create("content1"));
documentMap.put("PID2", DKIKFTextDocument.create("content2"));
documentMap.put("PID3", DKIKFTextDocument.create("content3"));
Map recordMap = (Map)ikfService.runServerTask(documentMap);

Iterator pids = recordMap.keySet().iterator();
while(pids.hasNext())
{
    DKIKFRecord record = (DKIKFRecord)recordMap.get(pids.next());
    System.out.println(record.getPID());
    System.out.println(record.getValue("IKF_LANGUAGE"));
    System.out.println(record.getValue("IKF_SUMMARY"));
}
...
```

文書のマップはサーバー・タスクに渡され、処理されます。サーバー・タスクは、作成されたメタデータ（この場合は文書の言語および要約）を含む指定された文書のレコードのマップを戻します。

サンプルのサーバー・タスクのソース・コードは、以下のようになります。

```
public class AnalysisTask implements DKIKFServerTask {
    private String catalogName;
    private DKIKFSchema catalogSchema;

    public AnalysisTask(String catalogName) {
        this.catalogName = catalogName;
    }

    public Serializable runServerTask(DKIKFService ikfService, Serializable argument)
        throws DKIKFServerTaskException {
        try {
            //the schema is retrieved only once
            if(catalogSchema == null) {
                catalogSchema = ikfService.getLibrary().getCatalog(catalogName).getSchema();
            }

            //preparing the argument, tools, and the map to be returned
            Map documentMap = (Map)argument;
            DKIKFLanguageIdentifier languageIdentifier = new DKIKFLanguageIdentifier(ikfService);
            DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
            HashMap recordMap = new HashMap();
            Iterator pids = documentMap.keySet().iterator();

            //creating a record for each pid
            while(pids.hasNext()) {
                String pid = (String)pids.next();
                DKIKFTextDocument document = (DKIKFTextDocument)documentMap.get(pid);
                DKIKFRecord record = DKIKFRecord.create(pid, catalogSchema);
                String language = languageIdentifier.analyze(document)[0].getLanguage();
                document.setLanguage(language);
                record.setValue("IKF_LANGUAGE", language);
            }
        }
    }
}
```

```

        record.setValue("IKF_SUMMARY", summarizer.analyze(document).getSummary());
        recordMap.put(pid, record);
    }

    //returning the results
    return recordMap;
}
catch(Exception e) {
    throw new DKIKFServerTaskException(e);
}
}
}

```

サーバー・タスクは、レコード作成に必要なスキーマを含むカタログの名前でインスタンス生成されます。メソッド `runServerTask` は、一度だけスキーマを取り出します。このメソッドは文書ごとにレコードを作成し、文書の分析のためのツールを実行し、作成されたメタデータをレコードに保管します。最後に、作成されたレコードがすべて呼び出し側 (クライアント・アプリケーション) に戻されます。

ご注意

上のサンプル・コードでは、メタデータのみが戻され、カタログにはレコードは作成されません。

JSP に基づく情報マイニング・アプリケーションの例

情報マイニングの Java Server Pages (JSP) アプリケーションは、情報マイニング・コンポーネントからカテゴリ内の文書を検索します。見つかった文書をカテゴリ構造で表示します。

JSP サンプル・アプリケーションは、以下のディレクトリーの中にあります。

Windows	<CMBROOT>%samples%jsp%infomining%
AIX	/usr/lpp/cmb/samples/jsp/infomining/
Solaris	/opt/IBMcmb/samples/jsp/infomining/

そのディレクトリーには、下記のファイルが含まれています。

advSearch.jsp

サンプルの最上位のファイル。

このパーツは、拡張検索特有の形式処理コードおよびフォーマット設定指示 (HTML) を提供します。また、データの表示を選択するためのコントローラーとしての役割も果たします。このファイルには、拡張検索に特有の初期設定指示 (特に検索できるカテゴリの可用性) が含まれています。

catView.jsp

このパーツは、表示特有のコードおよび戻される結果のカテゴリ表示に関するフォーマット設定指示 (HTML) を提供します。これには、戻された結果に対して繰り返されるループが含まれていますが、主に含まれているのはフォーマット設定指示です。

classes.jsp

このパーツは、論理および Bean 接続コードを提供します。含まれているのは Java コードだけです。戻された結果を表示するのに使用する、単純なデータ構造クラスがインプリメントされています。

また、戻された結果の検索または操作を実行するイベント・ハンドラーもインプリメントされています。拡張検索から結果リストが戻されると、主要な作業はこのファイルで実行されます。

logon.html サンプルの実行に必要なアカウントおよびカタログ入力。

アカウントおよびカタログ入力は、サーバー名、ユーザー名、パスワード、およびカタログ名で構成されます。

ソース・コードは情報マイニング Bean の使用に関するサンプルです。そのコードの動作に関する説明 (Bean のインスタンス生成、Bean 相互の接続、イベント・ハンドラーを使用した文書の戻し処理など) が含まれています。

JSP を実行するには、Enterprise Information Portal および情報マイニング・コンポーネントがインストールされている必要があります。また、JSP を実行できる Web サーバーも必要です。

JSP アプリケーションの詳細については、
<http://java.sun.com/products/jsp/index.html> を参照してください。

JSP のインストール

JSP を配置する前に、IBM WebSphere Application Server (WAS) がインストール済みで実行されていることを確認してください。

JSP の配置に必要なユーザー・アクセス権限は、次のとおりです。

- Windows の場合: 管理者権限
- AIX の場合: ルート・ユーザー特権
- Solaris の場合: ルート・ユーザー特権

JSP は、ディレクトリー <WAS_Home>%installedApps%JSP.ear%JSP.war に Web アプリケーションとして配置されます。サンプル JSP に変更を加えた場合は、このディレクトリーにある JSP をユーザーが作成した JSP に置き換えます。WAS は、JSP を自動的に再コンパイルします。

JSP 用に WebSphere Application Server を構成する方法については、「Enterprise Information Portal の計画とインストール」を参照してください。

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あり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。

日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
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をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

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、さまざま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。

商標

以下は、IBM Corporation の商標です。

IBM	DisplayWrite	PowerPC
400	e-business	PTX
Advanced Peer-to-Peer Networking	HotMedia	QBIC
AIX	Hummingbird	RS/6000
AIXwindows	ImagePlus	SecureWay
APPN	IMS	SP
AS/400	Micro Channel	VideoCharger
C Set ++	MQSeries	Visual Warehouse
CICS	MVS/ESA	VisualAge
DATABASE 2	NetView	VisualInfo
DataJoiner	OS/2	WebSphere
DB2	OS/390	
DB2 Universal Database	PAL	

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。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

用語集

この用語集では、本システムに固有の用語および略語を定義します。イタリック体の用語は、この用語集の他の場所で定義されています。

[ア行]

アーカイブ (archive). 長期にわたる情報保存に使用する持続ストレージ。通常、記憶装置は非常に安価で、アクセスは遅く、装置障害と自然災害発生時に備え、地理的に異なる場所に保管される。

アクション・リスト (action list). システム管理者またはその他のワークフロー・コーディネーターによって定義され、ワークフロー または文書ルーティング・プロセス内でユーザーが実行することを承認されたアクションのリスト。

アクセス制御 (access control). 許可ユーザーが、許可された方法でアクセスする場合に限り、特定の機能および保管オブジェクトにアクセスできるようにするプロセス。

アクセス制御リスト (access control list). 1 つまたは複数のユーザー ID やユーザー・グループ、およびそれらに関連する特権で構成されるリスト。アクセス制御リストは、Content Manager システム内の項目およびオブジェクトに対するユーザー・アクセスを制御する場合に使用する。アクセス制御リストは、Enterprise Information Portal システム内の検索テンプレートに対するユーザー・アクセスを制御する場合に使用する。

アプリケーション・プログラミング・インターフェース (application programming interface (API)). アプリケーション相互の通信を実行できるようにするソフトウェア・インターフェース。API はプログラム言語の構造体とステートメントの集合で、アプリケーション・プログラム中にコード化することにより、基礎を成すライセンス・プログラムに備えられている特定の機能やサービスを利用できる。

イテレーター (iterator). オブジェクトのコレクションを 1 つずつ処理するために使用するクラスまたは構成。

意味タイプ (semantic type). 項目の使用法または規則。ベース、注釈、および注は、Content Manager によって提供される意味タイプ。ユーザーが独自の意味タイプを定義することもできる。

イメージ・オブジェクト・コンテンツ・アーキテクチャ (Image Object Content Architecture (IOCA)). イメージの交換と表示に使われる構成の集合。

イメージ・コンテンツによる照会 (query by image content (QBIC)). プレーン・テキストではなく、フィーチャーというビジュアル・コンテンツに基づく検索を可能にする照会のテクノロジー。QBIC を使用して、色、テキストチャーなど、可視特性に基づいてオブジェクトを検索できる。

インライン (inline). Content Manager で、オンラインでドライブ中に入っているが、アクティブ・マウント状態ではないオブジェクト。「マウント済み (mounted)」と対比。

永続的 ID (persistent identifier (PID)). 保管場所に関係なくオブジェクトを固有に識別する ID。PID は項目 ID と位置の両方から成る。

エレメント (element). リスト・マネージャーがアプリケーションに割り振るオブジェクト。

オーバーレイ (overlay). 印刷時にページ上の変数データと組み合わせることができる、事前定義されたデータのコレクション (線、陰影、テキスト、枠、ロゴなど)。

オブジェクト (object). 一つの単位として保管、検索、および操作できるデジタル・コンテンツ。例えば、JPEG イメージ、MP3 オーディオ、AVI ビデオ、およびブックからのテキスト・ブロックなど。

オブジェクトのリンクと組み込み (Object Linking and Embedding (OLE)). アプリケーションをリンクしたり埋め込んだりすることで、他のアプリケーションからそのアプリケーションを呼び出せるようにする、Microsoft® 社による仕様。

オブジェクト・サーバー (object server). 「リソース・マネージャー (resource manager)」を参照。

オブジェクト・サーバー・キャッシュ (object server cache). 「リソース・マネージャー・キャッシュ (resource manager cache)」を参照。

【力行】

カーソル (cursor). アプリケーション・プログラムで、行の配列集合中の特定の行を指すために使用される名前付きの制御構造。カーソルは、集合から行を取り出すために使用される。

解放 (release). 延期された基準を項目 から除去すること。延期された項目が解放されるのは、基準が満たされた場合か、または適切な権限を持つユーザーが基準をオーバーライドして手操作で解放する場合である。

拡張データ・オブジェクト (extended data object (XDO)). アプリケーション・プログラムで、保管されている複雑なマルチメディア・オブジェクト がストレージから出し入れされる場合に、そのオブジェクトを総称した表現。XDO は DDO に含まれることが最も多い。

カテゴリー (category). 「項目タイプ (item type)」を参照。

管理クラス (management class). API でマイグレーション・ポリシー に使用する用語。

キー・フィールド (key field). 「属性 (attribute)」を参照。

基数 (cardinality). データベース・テーブルの行数。

基本属性 (base attributes). 個々のオブジェクト に割り当てられる索引の集合。すべての Content Manager オブジェクトは、基本属性 を持つ。

キャッシュ (cache). 主記憶装置より小さくて高速の、特別な用途のバッファ。頻繁にアクセスされるデータのコピーを保持するために使用される。キャッシュの使用によりアクセス時間は削減されるが、メモリー要件が増える可能性がある。「リソース・マネージャー キャッシュ (resource manager cache)」および「LAN キャッシュ (LAN cache)」も参照。

クライアント / サーバー (client/server). 通信において、あるサイトのプログラムが他のサイトのプログラムへ要求を送って応答を待つという、分散データ処理における対話のモデル。要求プログラムがクライアント、応答プログラムがサーバーと呼ばれる。

クライアント・アプリケーション (client application). ユーザー・インターフェースをカスタマイズする目的で、Content Manager API を使用して作成されたアプリケーション。Enterprise Information Portal からコンテンツ・サーバー にアクセスする目的で、オブジェクト指向 API またはインターネット API を使用して作成されたアプリケーション。

クラス (class). オブジェクト指向の設計やプログラミングで、共通の定義でオブジェクトを作成するようインスタンス化できるモデルまたはテンプレート。したがって、プロパティ、操作、および動作も共通になる。オブジェクトはクラスのインスタンスになる。

結合サーチ (combined search). パラメトリック、テキスト、またはイメージのいずれかのタイプの検索 (サーチ) を 1 つまたは複数組み合わせた照会。

検索基準 (search criteria). Content Manager において、保管された項目 を検索するために使用する属性値。Enterprise Information Portal において、管理者が定義する検索テンプレート の特定のフィールド。ユーザー が使用できる選択項目を制限したり定義を詳細化したりする。

検索テンプレート (search template). 管理者が設計した検索基準 から成る、特定のタイプの統合検索のフォーム。管理者は、個々の検索テンプレートにアクセスできるユーザー とユーザー・グループ も識別する。

交換 (interchange). イメージとその索引を 1 つの Content Manager ImagePlus for OS/390 システムから別の ImagePlus システムへ、CIF や CIU を利用してインポートまたはエクスポートする機能。

項目 (item). Content Manager において、項目タイプのインスタンスを表す総称。例えば、項目には、フォルダー、文書、ビデオ、またはイメージなどある。Enterprise Information Portal が管理する最小の情報単位を総称した用語。個々の項目には ID がある。例えば、項目は、フォルダー、または文書 である場合がある。

項目タイプ (item type). 項目などを定義して、後に位置決めをするためのテンプレート。ルート・コンポーネント、ゼロ以上の子コンポーネント、および種別で構成される。

項目タイプ種別 (item type classification). 項目 をさらに識別する項目タイプ 内でのカテゴリー化。同じ項目タイプの項目はすべて、同じ項目タイプ種別を持つ。

Content Manager は、次の項目タイプ種別を提供する。フォルダー、文書、オブジェクト、ビデオ、イメージ、およびテキスト。また、ユーザー独自の項目タイプ種別を定義することもできる。

子コンポーネント (child component). 項目タイプ 階層の、オプションの 2 番目以下のレベル。子コンポーネントはそれぞれ、その上のレベルと直接関連付けられている。

コネクター・クラス (connector class). 特定のコンテンツ・サーバー に固有の API へ標準アクセスを提供する、オブジェクト指向プログラミングのクラス。

コモン・ゲートウェイ・インターフェース (Common Gateway Interface (CGI)). Web サーバーとその外部のプログラムとの間で情報を交換するための規格。外部プログラムを作成する際のプログラム言語は、Web サーバーを実行しているオペレーティング・システムでサポートされているものであればどれでもよい。「CGI スクリプト (CGI script)」を参照。

コレクション (collection). 類似した一連の管理規則を持つオブジェクトのグループ。

混合オブジェクト文書コンテンツ・アーキテクチャー (Mixed Object Document Content Architecture (MO:DCA)). 交換環境中のアプリケーション間や環境間でオブジェクト・データを交換できるように開発された IBM アーキテクチャー。

混合オブジェクト文書コンテンツ・アーキテクチャー・プレゼンテーション (Mixed Object Document Content Architecture-Presentation (MO:DCA-P)). MO:DCA のサブセット・アーキテクチャー。Content Manager ImagePlus for OS/390 ワークステーションに送信される表示用や印刷用の文書を入れるエンベロープとして使用される。

コンストラクター (constructor). プログラム言語では、クラスと同じ名前のメソッドのことで、そのクラスのオブジェクトの作成と初期化に使用される。

コンテナ (container). オブジェクトを保持する、ユーザー・インターフェースの要素。フォルダー・マネージャー では、他のフォルダーまたは文書を含むことができるオブジェクト。

コンテンツ・クラス (content class). 「MIME タイプ (MIME type)」を参照。

コンテンツ・サーバー (content server). マルチメディア・データや業務データ、またそのデータの処理に必要な関連メタデータを保管するためのソフトウェア・システム。コンテンツ・サーバーの例としては、Content Manager や Content Manager ImagePlus for OS/390 がある。

コンポーネント (component). ルート・コンポーネント または子コンポーネント の総称。

[サ行]

サーバー定義 (server definition). Enterprise Information Portal で固有に識別できるようにするための、特定のコンテンツ・サーバー の特性。

サーバー・インベントリ (server inventory). 指定されたコンテンツ・サーバー のネイティブ・エンティティ およびネイティブ属性 の包括的なリスト。

サーバー・タイプ定義 (server type definition). 特定のタイプのカスタム・サーバーが、管理者が識別するように、Enterprise Information Portal でも固有に識別されるために必要な特性のリスト。

作業項目 (work item). 旧バージョンの Content Manager ワークフローおよび Enterprise Information Portal 拡張ワークフローにおいて、ワークフロー 内でアクティブになっている作業アクティビティ。

作業状態 (work state). 個々の作業項目、文書、またはフォルダー の状況。

作業ステップ (work step). ワークフロー または文書ルーティング・プロセス 内の離散ポイントで、個々の作業項目、文書、またはフォルダー はこの点を通過しなくてはならない。

索引 (index). 特定の項目 またはオブジェクト を識別する属性値を追加または編集し、これを後から検索できるようにする。

索引クラス (index class). 「項目タイプ (item type)」を参照。

索引クラス・サブセット (index class subset). 初期の Content Manager において、アプリケーションがフォルダーおよびオブジェクトの保管、取り出し、表示に使用する索引クラス のビュー。

索引クラス・ビュー (index class view). 初期の Content Manager において、索引クラス・サブセット の API で使用する用語。

サスペンドする (suspend). オブジェクト をワークフロー から取り除き、それを活動化させるために必要なサスペンド基準を定義すること。そのオブジェクトを後で活動化させると、処理を再開する。

サブクラス (subclass). 別のクラスから派生するクラス。クラスとサブクラスの間には 1 つ以上のクラスがある場合がある。

参照 (reference). ルートまたは子コンポーネント と別のルート・コンポーネント との間の単一方向の 1 対 1 の結合。「リンク (link)」と対比。

システム管理ストレージ (system-managed storage (SMS)). Content Manager のストレージ管理の方法。システムによりオブジェクトの位置が判別され、オブジェクトのバックアップ、移動、スペース、およびセキュリティが自動的に管理される。

照会ストリング (query string). 照会のプロパティおよびプロパティ値を指定するための文字ストリング。アプリケーションの中で照会ストリングを作成して、照会に渡すことができる。

情報マイニング (information mining). テキストからの重要な情報の抜粋 (要約)、文書の集合からの主要なテーマの検出 (カテゴリー化)、および強力な柔軟性のある照会を使用した適切な文書の検索を実行するための自動化されたプロセス。

シン・クライアント (thin client). ソフトウェアがほとんど、またはまったくインストールされていないが、接続先のネットワーク・サーバーで管理され送達されるソフトウェアに対するアクセス権はあるクライアント。シン・クライアントは、ワークステーションなどの機能の充実したクライアントの代替になる。

スーパークラス (superclass). クラスの派生元のクラス。クラスとスーパークラスの間に 1 つ以上のクラスがある場合がある。

ステージング (staging). 保管オブジェクト を、オフラインまたは優先順位の低い装置から、オンラインまたは優先順位の高い装置に戻すためのプロセス。通常はシステムまたはユーザーの要求に基づいて行われる。ユーザーが永続記憶装置に保管されているオブジェクトを要求すると、作業用コピーがステージング域 に書き込まれる。

ステージング域 (staging area). リソース・マネージャー の作業用ストレージ領域。リソース・マネージャー・キャッシュ ともいう。

ストリーム化データ (streamed data). ネットワーク接続を介して一定の速度で送信されるすべてのデータ。ストリームは、1 つのデータ・タイプまたは複数のタイプの組み合わせである。データ転送率 (ビット / 秒で示される) は、ストリームやネットワークの種類によって異なる。

ストレージ・クラス (storage class). オブジェクトが保管されているメディアのタイプを示す。これは物理ロケーションとは直接関連しないが、デバイス・マネー

ジャー に直接関連付けられる。ストレージ・クラスには、以下のタイプがあります。

DASD

ハード・ディスク

光ディスク

ストリーム

テープ

TSM

ストレージ・グループ (storage group). ストレージ・システムをストレージ・クラスに関連付ける。

ストレージ・システム (storage system). Content Manager システム中のストレージを総称した用語。

「TSM ボリューム (TSM volume)」、**「メディア・アーカイバー (media archiver)」**、および**「ボリューム (volume)」**を参照。

接続マネージャー (connection manager). Content Manager コンポーネントの 1 つで、ライブラリー・サーバーに対する接続を保守するために役立つ。ただし照会ごとに新しい接続を開始するためには使用されない。接続マネージャーには、アプリケーション・プログラミング・インターフェース (API) がある。

属性 (attribute). 項目の一定の特性またはプロパティ (名前、アドレス、経過日数など) を記述し、その項目を探し出すために使用できるデータの単位。属性には、その属性によって保管される情報範囲を指示するタイプ、およびその範囲内の値が含まれる。例えば、マルチメディア・ファイル・システム内のファイルについての、タイトル、実行時間、またはエンコード・タイプなどの情報 (MPEG1、H.263 など)。Enterprise Information Portal については、「統合属性 (federated attribute)」および「ネイティブ属性 (native attribute)」も参照。

属性グループ (attribute group). 1 つまたは複数の属性 の便宜上のグループ分け。例えば、住所には、番地、市区町村、都道府県、および郵便番号という属性が含まれる。

【タ行】

抽象クラス (abstract class). 概念を表現する、オブジェクト指向プログラミングのクラス。このクラスから作成されるクラスは、概念のインプリメンテーションを表現する。抽象クラスのオブジェクトの構成はできない。つまり、インスタンス化は不可能。

データ・ストア (datastore). (1) データを保管する場所 (データベース・システム、ファイル、ディレクトリ

ーなど)を示す総称用語。(2)アプリケーション・プログラムでは、コンテンツ・サーバーの仮想表現。

データ・フォーマット (data format). 「MIME タイプ (MIME type)」を参照。

デステージャー (destager). オブジェクトを、ステージング域からオブジェクトのマイグレーション・ポリシーの第1ステップに移動する、Content Manager リソース・マネージャーの機能。

デバイス・マネージャー (device manager). Content Manager システムにおいて、リソース・マネージャーと1つまたは複数の物理デバイス間のインターフェースのこと。

統合エンティティ (federated entity). 統合属性から成る Enterprise Information Portal メタデータ・オブジェクト。統合エンティティは、1つ以上の統合テキスト索引と関連付けることもできる。

統合検索 (federated search). 1つ以上のコンテンツ・サーバーでデータを同時に検索する、Enterprise Information Portal から発行される照会。

統合コレクション (federated collection). 統合検索の結果であるオブジェクトのグループ。

統合属性 (federated attribute). 1つ以上のコンテンツ・サーバーでネイティブ属性にマップされる Enterprise Information Portal メタデータ・カテゴリー。例えば、統合属性であるポリシー番号 (policy number) は、Content Manager では属性 policy num に、Content Manager ImagePlus for OS/390 では属性 policy ID にマップされる。

統合データ・ストア (federated datastore). 任意の数の特定のコンテンツ・サーバー (Content Manager など)を指す仮想表現。

統合テキスト索引 (federated text index). 1つ以上のコンテンツ・サーバーで1つ以上のネイティブ・テキスト索引にマップされる Enterprise Information Portal メタデータ・オブジェクト。

動的データ・オブジェクト (dynamic data object (DDO)). アプリケーション・プログラムで、ストレージから出し入れされる保管オブジェクトを総称した表現。

独立型システム (stand-alone system). Content Manager システムのコンポーネントがすべて1つのパーソナル・コンピュータにインストールされるよう事前構成された Content Manager システム。

特権 (privilege). 特定のオブジェクトに特定の方法でアクセスする権利。特権には、システムに保管されているオブジェクトの作成、削除、および選択を行う権利が含まれる。特権は、管理者が割り当てる。

特権セット (privilege set). システムのコンポーネントや機能処理するための特権のコレクション。管理者は特権セットをユーザー (ユーザー ID) およびユーザー・グループに割り当てる。

[ナ行]

ネイティブ属性 (native attribute). 特定のコンテンツ・サーバー上で管理され、そのコンテンツ・サーバーにとって固有のオブジェクトの特性。例えば、キー・フィールド policy num は Content Manager コンテンツ・サーバーのネイティブ属性で、フィールド policy ID は Content Manager OnDemand コンテンツ・サーバーのネイティブ属性。

ネイティブ・エンティティ (native entity). 特定のコンテンツ・サーバー上で管理され、ネイティブ属性から成るオブジェクト。例えば、Content Manager 索引クラスは、Content Manager キー・フィールドから成るネイティブ・エンティティである。

ネイティブ・テキスト索引 (native text index). 特定のコンテンツ・サーバー上で管理されるテキスト項目の索引。例えば、Content Manager コンテンツ・サーバーに1つあるテキスト検索索引など。

ネットワーク・テーブル・ファイル (network table file). Content Manager システムのそれぞれのノードのシステム固有の構成情報が含まれているテキスト・ファイル。システムのそれぞれのノードには、ノードを識別し、接続する必要があるノードのリストを表示するネットワーク・テーブル・ファイルがある。

ネットワーク・テーブルの名前は FRNOLINT.TBL。

[ハ行]

パージャー (purger). オブジェクトをシステムから除去する、リソース・マネージャーの機能。

パーツ (part). 「オブジェクト (object)」を参照。

バイナリー・ラージ・オブジェクト (binary large object (BLOB)). サイズが0バイト~2ギガバイトの、バイトのシーケンス。このストリングには関連したコード・ページや文字セットはない。イメージ、オーディオ、およびビデオの各オブジェクトは BLOB の形で保管される。

ハイパーテキスト・マークアップ言語 (Hypertext Markup Language (HTML)). SGML 規格に準拠したマークアップ言語で、主に、ハイパーテキスト・リンクを含むテキスト情報とグラフィック情報のオンライン表示をサポートするために設計された。

パッケージ (package). 関連するクラス およびインターフェースのコレクション。アクセス保護とネーム・スペース管理を提供する。

パトロン (patron). Content Manager API でユーザーに使用する用語。

パラメトリック・サーチ (parametric search). オブジェクトのプロパティに基づき、オブジェクトの照会。

ハンドル (handle). オブジェクトを表す文字ストリング。オブジェクトを検索する場合に使用する。

ヒストリー・ログ (history log). ワークフローの活動履歴を保持するファイル。

ファイル・システム (file system). AIX で、ハード・ディスクのパーティションを区切ってストレージを作る方法。

フィーチャー (feature). イメージ検索サーバーに格納されたビジュアル・コンテンツ情報。さらに、イメージ検索アプリケーションが突き合わせに使用する可視特性。*QBIC* フィーチャーには、平均色、ヒストグラム色、定位置色、およびテクスチャーの 4 種類がある。

フォルダー (folder). フォルダー意味タイプが指定された、種別と関係ない任意の項目タイプの項目。フォルダー意味タイプの項目は、非リソース項目の機能、および項目タイプ種別 (文書 やリソース項目など) に備わっている追加機能のほかに、Content Manager が提供するフォルダー機能を備えている。フォルダーには、文書やサブフォルダーなど任意のタイプの項目をいくつでも含めることができる。フォルダーは、属性 別に索引付けされる。

フォルダー・マネージャー (folder manager). データをオンライン文書およびフォルダーとして管理する Content Manager モデル。フォルダー・マネージャー API は、アプリケーションと Content Manager コンテンツ・サーバーとの間の基本インターフェースとして使用できる。

プロパティ (property). オブジェクトの性質に関する説明。プロパティは変更や修正が可能。プロパティの例としては、例えば書体などがある。

文書 (document). 単独の単位として、Content Manager システムとユーザーとの間で保管、検索、および交換可能な項目。文書意味タイプの項目は、文書を構成する情報を含んでいることが予期されるが、Content Manager 文書モデルのインプリメンテーションであるとは必ずしも限らない。

文書として分類される項目タイプ (Content Manager 文書モデルの特定のインプリメンテーション) から作成された項目は、文書パーツを含んでいる必要がある。文書として分類される項目タイプを使用して、文書意味タイプまたはフォルダー意味タイプの項目を作成できる。

文書パーツには、例えばテキスト、イメージ、スプレッドシートなど、異なるタイプのコンテンツを含めることができる。

文書コンテンツ・アーキテクチャー (document content architecture (DCA)). オフィス・システムのネットワーク内でやりとりされる文書に対して情報の整合性を保証するアーキテクチャー。DCA は、文書の形式と意味を指定するときの規則を提供する。また、変更可能テキスト (変更が可能) と最終形式テキスト (変更が不可) を定義している。

文書タイプ定義 (document type definition (DTD)). 特定クラスの XML 文書用の構造を指定する規則。DTD は、エレメント、属性、および表記とともに構造を定義し、各エレメント、属性、および表記を文書の特定クラス内で使用する方法についての制約を確立する。DTD は、特定のマークアップ言語の構造を完全記述するという点で、データベース・スキーマに類似している。

文書ルーティング・プロセス (document routing process). Content Manager において、文書 またはフォルダー が処理の過程で通過する一連の作業ステップおよびそれらのステップを管理する規則。

ボリューム (volume). システムのオブジェクトが保管されている実際の物理ストレージ装置または物理ストレージ単位を表す。

【マ行】

マイグレーション (migration). (1) コンピューター・システム間でデータ変換せずにデータやソースを移動するプロセス。例えば、新しい操作環境に移動する場合など。(2) 新しいバージョンからリリースのプログラムをインストールして、旧バージョンまたはリリースを置き換えること。

マイグレーション・ポリシー (migration policy). ストレージ・クラス 間でオブジェクト を移動するため

の、ユーザー定義のスケジュール。ストレージ階層中のオブジェクトのグループの保存特性やクラス変換特性を記述する。

マイグレーター (migrator). リソース・マネージャーの機能の 1 つで、マイグレーション・ポリシー を検査し、移動されるようスケジュールされているオブジェクトを次のストレージ・クラス に移動する。

マウント (mount). データ・メディアを操作位置に置くこと。

マウント済み (mounted). Content Manager で、オンラインでドライブ中に入っており、アクティブなマウント 状態であるオブジェクト。「インライン (inline)」と対比。

マシン生成データ構造 (machine-generated data structure (MGDS)). (1) さまざまな Content Manager ImagePlus for OS/390 プログラム間で文字データを渡すための、IBM 構造化データ・フォーマット・プロトコル。(2) イメージから抽出され、汎用データ・ストリーム (GDS) 形式にされたデータ。

マルチメディア (multimedia). コンピューターから表示および制御できるように、異なるメディア・エレメント (テキスト、グラフィックス、オーディオ、静止画、ビデオ、アニメーション) を結合すること。

マルチメディア・ファイル・システム (multimedia file system). ビデオおよびオーディオの保管および送達用に最適化されたファイル・システム。

メソッド (method). Java 設計とプログラミングで、操作によって指定される動作が実装されたソフトウェア。C++ におけるメンバー関数と同義。

メディア・アーカイバー (media archiver). オーディオおよびビデオのストリーム・データの保管に使用する物理装置。VideoCharger はメディア・アーカイバーの一種である。

メディア・サーバー (media server). Content Manager システムの AIX に基づくコンポーネント。ビデオ・ファイルを保管し、これらのファイルにアクセスするときを使う。

目録 (table of contents (TOC)). フォルダーまたはワークバスケット に入れられた、文書 およびフォルダーのリスト。検索の結果は、フォルダーの目録として表示される。

[ヤ行]

ユーザー (user). Content Manager のサービスが必要な人。通常この用語は、アプリケーションの開発者ではなく、Content Manager API を使用するクライアント・アプリケーションの使用者を指す。Enterprise Information Portal では、Enterprise Information Portal 管理プログラム内で識別される人すべて。

ユーザー出口 (user exit). IBM 提供のプログラムのうち、ユーザー出口ルーチンが制御を受け取ることができる場所。

ユーザー出口ルーチン (user exit routine). 事前定義のユーザー出口 で制御を受け取るユーザー作成ルーチン。

ユーザー・グループ (user group). 1 つ以上の定義済み個別ユーザー で成るグループ。単一グループ名によって識別される。

ユーザー・マッピング (user mapping). Enterprise Information Portal のユーザー ID とパスワードを、1 つ以上のコンテンツ・サーバー中の対応するユーザー ID とパスワードに関連付けること。ユーザー・マッピングにより、Enterprise Information Portal と複数のコンテンツ・サーバー にシングル・ログオンできる。

ユーティリティ・サーバー (utility server). データベース・ユーティリティによって、スケジュールリング用に使用される Content Manager コンポーネント。リソース・マネージャー またはライブラリー・サーバー の構成時に、ユーティリティ・サーバーを構成する。ユーティリティ・サーバーの数は、リソース・マネージャーおよびライブラリー・サーバー当たり 1 つである。

[ラ行]

ライブラリー・オブジェクト (library object). 「項目 (item)」を参照。

ライブラリー・クライアント (library client). ライブラリー・システムに低レベルのプログラミング・インターフェースを提供する Content Manager システムのコンポーネント。ライブラリー・クライアントには、ソフトウェア開発者キットの一部を成す API が含まれている。

ライブラリー・サーバー (library server). 項目 上の照会を保管、管理、および処理する Content Manager システムのコンポーネント。

ランク (rank). 指定されたパーツと照会結果の関係を示す整数値。ランクが高いほど一致の度合いが高い。

リソース交換ファイル・フォーマット (Resource Interchange File Format (RIFF)). 異なるタイプのコンピュータ装置上で再生する音またはグラフィックスを保管するために使用する。

リソース・マネージャー. オブジェクトを管理する Content Manager システム。これらのオブジェクトは、ライブラリー・サーバー上に保管された項目によって参照される。

リソース・マネージャー・キャッシュ (resource manager cache). リソース・マネージャーの作業用ストレージ領域。ステー징領域ともいう。

リモート・メソッド呼び出し (Remote Method Invocation (RMI)). プログラミングを分散できる API の集合。ある Java 仮想計算機 (JVM) 中のオブジェクトが、別の JVM 中のオブジェクトのメソッドを呼び出せる。

リンク (link). 2 つの項目 (ソースとターゲット) の間の方向関係。一連のリンクを使用して、1 対多の関連を作ることができる。「参照 (reference)」と対比。

ルート・コンポーネント (root component). 関連システム定義およびユーザー定義の属性から構成される、項目タイプ階層の第 1 レベルまたは唯一のレベル。

レンダリング (render). 通常であればイメージとは無関係なデータを、イメージとして捕らえて表示すること。Content Manager では、ワープロ文書を表示する目的で、それをイメージとしてレンダリングできる。

ローカル・エリア・ネットワーク (local area network (LAN)). 一連の装置が通信のためにお互いに接続されたネットワークで、より大規模なネットワークへ接続することも可能。

[ワ行]

ワークバスケット (workbasket). 処理実行中または処理待機中の文書またはフォルダーのコレクション。ワークバスケットの定義には、そのコンテンツの表示、状況、およびセキュリティを管理する規則が含まれている。

ワークフロー (workflow). 旧バージョンの Content Manager において、文書またはフォルダーが処理の過程で通過する一連のワークバスケット。Enterprise Information Portal において、作業パケット、文書、また

はフォルダーが処理の過程で通過する一連の作業ステップおよびそれらのステップを管理する規則。

例えば、保険料請求の承認 (claims approval) では、個々の保険請求が承認を受けるために経なければならないプロセスを記述する。

ワークフローの状態 (workflow state). ワークフロー全体の状態。

ワークフロー・コーディネーター (workflow coordinator). 旧バージョンの Content Manager で、ワークフロー内の作業項目が、指定されている時間内に処理されなかったという通知を受け取るユーザー。このユーザーは、特定のユーザー・グループに関して選択されるか、またはワークフローの作成時に選択される。

ワーク・パケット (work packet). Enterprise Information Portal バージョン 7.1 において、ある場所から別の場所に経路指定される文書のコレクション。ワーク・リストを介してワーク・パケットにアクセスして処理する。

ワーク・リスト (worklist). ユーザーに割り当てられた作業項目、文書、またはフォルダーの集合。

ワイルドカード文字 (wildcard character). アスタリスク (*) または疑問符 (?) などの特殊文字。1 つまたは複数の文字を表現するために使用することができる。文字または文字セットをワイルドカード文字に置き換えることができる。

A

ADSM. 「Tivoli® Storage Manager」を参照。

API. 「アプリケーション・プログラミング・インターフェース (application programming interface)」を参照。

AVI. 「AVI (Audio Video Interleaved)」を参照。

AVI (Audio Video Interleaved). オーディオ・データおよびビデオ・データをファイルにインターリーブすることができる RIFF (リソース交換ファイル・フォーマット) ファイル仕様。分離トラックには、ファイル装置上の連続アクセスを保持しながら、再生や記録用の代替チャンク内でアクセス可能。

B

BLOB. 「バイナリー・ラージ・オブジェクト (binary large object)」を参照。

C

CGI. 「コモン・ゲートウェイ・インターフェース (*Common Gateway Interface*)」を参照。

CGI スクリプト (CGI script). Web サーバー上で稼働するコンピューター・プログラムで、コモン・ゲートウェイ・インターフェース (*CGI*) を使用して、通常は Web サーバーでは行われないタスク (データベース・アクセスやフォーム処理など) を実行する。CGI スクリプトは、Perl などのスクリプト記述言語で書かれた CGI プログラムである。

CIF. 「*Common Interchange File (CIF)*」を参照。

CIU. 「*Common Interchange Unit (CIU)*」を参照。

Client Application for Windows. Content Manager が備えている Content Manager API で書かれた完全なオブジェクト管理システム。文書とフォルダーの作成、保管、表示、処理、およびアクセス制御をサポートする。ユーザー出口ルーチンを使用してカスタマイズでき、API を使用して部分的に起動することができる。

Common Interchange File (CIF). ImagePlus Interchange Architecture (IPIA) データ・ストリームを 1 つ含んでいるファイル。

Common Interchange Unit (CIU). CIF で使われる、独立した転送単位。これは CIF の一部であり、受信データベースとの関係を示している。1 つの CIF に CIU が複数含まれることもある。

D

DCA. 「文書コンテンツ・アーキテクチャー (*document content architecture*)」を参照。

DDO. 「動的データ・オブジェクト (*dynamic data object*)」を参照。

DTD. 「文書タイプ定義 (*document type definition*)」を参照。

H

HTML. 「ハイパーテキスト・マークアップ言語 (*Hypertext Markup Language*)」を参照。

I

IOCA. 「イメージ・オブジェクト・コンテンツ・アーキテクチャー (*Image Object Content Architecture*)」を参照。

J

JavaBeans. 「Bean」と呼ばれる再使用可能な Java コンポーネントを作成するための、プラットフォーム非依存型のソフトウェア・コンポーネント・テクノロジー。これらの Bean は、作成後、他のソフトウェア・エンジニアが使用したり、Java アプリケーション内で使用したりすることができる。JavaBeans を使用して、ソフトウェア・エンジニアはグラフィカルなドラッグ・アンド・ドロップ型の開発環境で Bean の操作やアセンブルを行える。

Joint Photographic Experts Group (JPEG). (1) デジタル化連続トーン・イメージの圧縮についての規格を定めたグループ。(2) このグループによって開発された静止画の規格。

JPEG. 「*Joint Photographic Experts Group*」を参照。

L

LAN. 「ローカル・エリア・ネットワーク (*local area network*)」を参照。

LAN キャッシュ (LAN cache). リモートのリソース・マネージャーに保管されているオブジェクトのコピーを入れる、ローカルのリソース・マネージャーの一時記憶域。

M

MGDS. 「マシン生成データ構造 (*machine-generated data structure*)」を参照。

MIME タイプ (MIME type). インターネット上で転送されるオブジェクトのタイプを識別するためのインターネット規格。MIME タイプには、さまざまな種類のオーディオ、イメージ、およびビデオが含まれる。オブジェクトごとに 1 つの MIME タイプを持つ。

MO:DCA. 混合オブジェクト文書コンテンツ・アーキテクチャー (*Mixed Object Document Content Architecture*)

MO:DCA-P. 混合オブジェクト文書コンテンツ・アーキテクチャー・プレゼンテーション (*Mixed Object Document Content Architecture—Presentation*)

Multipurpose Internet Mail Extensions (MIME). 「MIME タイプ (*MIME type*)」を参照。

O

OLE. 「オブジェクトのリンクと組み込み (*Object Linking and Embedding*)」を参照。

P

PID. 「永続的 ID (*persistent identifier*)」を参照。

Q

QBIC. 「イメージ・コンテンツによる照会 (*query by image content*)」を参照。

R

README ファイル (README file). プログラムをインストールして実行する前に参照しなければならない、そのプログラムに関連付けられたファイル。README ファイルには、通常、最新の製品情報、インストール情報、または製品使用に関してのヒントが含まれる。

RIFF. 「リソース交換ファイル・フォーマット (*Resource Interchange File Format*)」を参照。

RMI サーバー (RMI server). Java リモート・メソッド呼び出し (*RMI*) 分散オブジェクト・モデルをインプリメントしたサーバー。

S

SMS. 「システム管理ストレージ (*system-managed storage*)」を参照。

T

Tivoli StorageManager (TSM). 異種環境でのストレージ管理とデータ・アクセス・サービスを備えたクライアント / サーバー 製品。さまざまな通信メソッドをサポートし、ファイルのバックアップとストレージを管理する機能を備え、バックアップ操作のスケジューリング機能を備えている。

TOC. 「目録 (*table of contents*)」を参照。

TSM. 「*Tivoli Storage Manager*」を参照。

TSM ボリューム (TSM volume). *Tivoli Storage Manager* によって管理される論理ストレージ。

U

uniform resource locator (URL). コンピューター上、またはインターネットなどのネットワーク内の情報リソースを表現する文字列。この文字列には、情報リソースにアクセスするために使用されるプロトコルの省略名、および情報リソースを見つけるためにプロトコルによって使用される情報が含まれている。例えば、インターネットのコンテキストでは、さまざまな情報リソースにアクセスするために使用されるプロトコルの省略名として、http、ftp、gopher、telnet、および news がある。

X

XDO. 「拡張データ・オブジェクト (*extended data object*)」を参照。

XML. 「*Extensible Markup Language*」を参照。

XML (Extensible Markup Language). SGML から派生した、SGML のサブセットである、マークアップ言語定義のための標準メタ言語。XML は、SGML の複雑であまり使用されない部分を省略したもので、文書タイプの処理、構造情報の作成および管理、さまざまなコンピューター・システム間での構造化情報の伝送および共用を行うためのアプリケーションを非常に容易に作成することができる。XML の使用は、耐久力のあるアプリケーションおよび SGML に必要な処理を必要としない。XML は、World Wide Web Consortium (W3C) の賛助で開発されている。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

【ア行】

アウトバウンド・リンク 187, 190

アクション

作成する 433

リストへのアクセス 433

アクセス制御

規則のリスト (ACL) 151

項目へのリストの割り当て 200

使用 192

ダイアグラム 148

文書ルーティング用のリストの使用
262

リスト (ACL) 148, 151

リスト (ACL) の検索と表示 198

リスト (ACL) の定義 196

Content Manager バージョン 8 148

NoAccessACL 152

PublicReadACL 152

SuperUserACL 152

アプリケーションのビルド

計画 152

注釈サービスの使用 473

ビジュアル Bean 459

非ビジュアル Bean 446

Content Manager バージョン 8 での作成 155

アプリケーション・プログラミング・インターフェース (API)

オンライン・リファレンス 154

概念 13

機能 29

ソフトウェア・コンポーネント 155

Java 29

アーキテクチャー 29

パッケージ 30

複数検索 37

C++ との違い 29

アプリケーション・プログラミング・リファレンス (APR) 154

色検索

定位置 308, 312

テキストチャー 308, 312

ヒストグラム 308, 311, 312

平均 308, 311, 312

意味タイプ

定義 168

定義済みタイプ 168

ユーザー定義 168

一致ハイライト

参照: ハイライト、テキスト照会

イテレーター

統合 111

DKResults 内での位置決め 112

イベント

Bean 444

イメージ検索 306

アプリケーション 308

カタログ 306, 307

カタログのリスト作成 315

検索基準 310

コンテンツ・サーバーからの照会の実行 320

コンテンツ・サーバーからの照会の評価 322

サーバーのリスト作成 314

最大結果数 310

索引付けされるデータのロード 323

照会 319

紹介 6

照会構文 310

照会の作成 310

接続 313

切断 313

属性 318

ダイアグラム 307

データベース 306, 307

データベースのリスト作成 315

定位置色 308, 312

定義 38

テキストチャー 308, 312

統合 24

ヒストグラム色 308, 311, 312

フィーチャー 306, 310

フィーチャーのリスト作成 315

平均色 308, 311, 312

closeCatalog 313

DDO による情報の表現 318

ID 309

listDatabases 313

openCatalog 313

XDO の索引付け 323

イメージ・キューの処理 323

イメージ・コンテンツによる検索 (QBIC)、

参照: イメージ検索

インバウンド・リンク 190

インポート、XML の 90

永続的 ID (PID)

イメージ検索 318

拡張データ・オブジェクト (XDO) 15

紹介 17

動的データ・オブジェクト (DDO) 14

バージョン管理 147

ルーティング 257

エクステンダー 15

エスケープ・シーケンス、照会例 219

エラー・メッセージ

プロパティ・ファイル 421

延期、ワークフロー 427

演算子

パラメトリック・サーチ 204

Domino.Doc 375

FileNET 395

Generalized Query Language

(GQL) 391

ImagePlus for OS/390 照会 360

エンティティ

オブジェクトの移動 420

クラス階層 265

識別 17

スキーマ・マッピング 408

定義用の特定 DK クラス名 411

統合フォルダーの保管 26

統合モデルでのマッピング 20

特権 149

ネイティブ 352, 408

ネイティブとしての OnDemand フォルダ 352

被制御、Content Manager バージョン 8 148

リレーショナル・データベースでのリスト作成 400

Bean 440, 478

CM 8 でのリスト取得 160

CM for AS/400 でのリスト作成 362

Domino.Doc での検索 374

Domino.Doc でのリスト作成 372

Extended Search 377

FileNET での照会 396

FileNET でのリスト作成 393

ImagePlus for OS/390 361

IP OS/390 でのリスト作成 355

オーバーレイ・オブジェクト 420

オープン、結果セット・カーソル 415
オープン文書管理 API (ODMA) 369
オブジェクト
 コンテンツのテキスト検索 205
 属性値の保管 169
 Bean 478
 CM 8 での更新 171
 Content Manager バージョン 8 145
 Domino.Doc での管理 371
オブジェクト管理
 Java 270
 更新 170, 274
 削除 178, 278
 作成する 165, 270

[力行]

カーソル、結果セット
 位置 414
 エレメントの位置指定 414
 エレメントの検索 414
 エレメントの更新
 updateObject 415
 エレメントの削除
 deleteObject 415
 エレメントの追加 415
 オープン 415
 クローズ 415
 更新 414
 コンテンツ・サーバー情報の取得 415
 スクロール 414
 妥当性検査 414
 破棄中 415
 ハンドルの取得 415
ガーベッジ・コレクター、Java 29
階層ストレージ管理 (HSM) 227
拡張検索サンプル、情報マイニングの
 526
拡張データ・オブジェクト、
 参照: XDO
カスタム・コネクター
 開発 405
 システム管理 406
 FeServerDefBase クラスの拡張 422
カタログ、イメージ検索 313
カテゴリー化サンプル、情報マイニングの
 493
環境、設定
 C++ 33
 Java 31
管理
 オブジェクト
 検索 230
 特権 149
管理クライアント
 紹介 4
管理クライアント (続き)
 説明 4
 他のワークステーションへのインスト
 ール 4
管理データベース
 紹介 4
 スキーマ・マッピング 14
 説明 4
キーワード、Domino.Doc 371
基準、検索、Bean 477
キャッシング、コントローラー・サブプレ
 ット 483
キャビネット、Domino.Doc 371
 属性のリスト 374
許可、
 参照: 特権
共通 EIP クラス 406
共通特権、dkDatastoreExt 420
クライアント
 情報マイニング と連動するものの作成
 5
クライアント構成アシスタント 4
クラスター化サンプル、情報マイニングの
 512
クラス名用語 265
グラフィカル・ユーザー・インターフェー
 ス (GUI)、
 参照: Bean、ビジュアル
クローズ、結果セット・カーソル 415
結果セット・カーソル
 関数 414
 Java 133
 オープンおよびクローズ 134
 コレクションの作成 137
 設定および入手 134
結果セット・カーソルのスクロール 414
結合照会
 定義 38
C++
 プログラミング上のヒント 329
 ランキング 329
Java 207, 326
 テキストによるパラメーター 327
 有効範囲の使用 328
検索
 照会言語について 201
 API モジュール 155
 Bean 440, 476, 478
 C++
 パーツ 279
 DKResults クラス 407
 Java 278
 パーツ 279
 フォルダー 281
検索エンジン、登録 409

検索テンプレート、OnDemand
 ビューアー 351
 フォルダーの使用 352
検索の実行 38
検索の評価 38
コードのサンプル、更新 6
コード・ページ変換
 Windows でのコンソール・サブシステ
 ムの設定 37
コールバック、実行 38, 202
コールバックによる検索の実行 38, 202
構成ストリング、リレーショナル・デー
 タベース 58
項目
 コントローラー・サブプレットによる
 管理 486
 参照 156
 属性の変更 170
 チェックインおよびチェックアウト
 178
 ツリー定数 187
 データ・モデル内での表現 203
 統合 19
 バージョン管理 147, 179
 フォルダーからの除去 185
 フォルダーへの追加 183
 フォルダーを介したリンク 146
 文書項目タイプの作成 230
 リンク 189
 リンクされた項目の検索 191
 Bean 440, 476, 478
 CM 8 での検索 175
 CM 8 での属性の設定と検索 169
 Content Manager バージョン 8 143
項目クラス、
 参照: 項目タイプ
項目タイプ
 コンポーネント、ルートおよび子 144
 種別 158
 属性のリスト作成 164
 定義の作成 172
 バージョン管理 147, 180
 複数の照会 214
 リスト作成 160
 Content Manager バージョン 8 143
 Content Manger バージョン 8 での作
 成 158
項目パーツ
 DDO および XDO との比較 16
子コンポーネント
 参照 145
 ダイアグラム 144
 データ・モデル内での表現 203
 バージョン管理 147
 CM 8 での作成 167
 Content Manager バージョン 8 144

子コンポーネント (続き)

DDO 属性としての表現 156

コネクタ

カスタマイズ 405

コントローラー・サブレット値
482

説明 5

ローカルとリモート 5

コレクション

削除 109

統合 111

分類する 110

dkCollection クラス 407

Extended Search 383

コレクションとイテレーター

C++

メモリ管理 109

Java 105

順次イテレーター 106

順次コレクション 105

コレクション・ポイント

説明 242

定義 246

コンソール・サブシステム、Windows で
の設定 37

コンテナ

意味タイプ 168

定数 168

Content Manager バージョン 8 145

コンテンツの更新 418

XDO 66

コンテンツ・サーバー

イメージ検索 306

イメージ照会の実行 320

イメージ照会の評価 322

永続的 ID (PID) 17

オブジェクトの拡張 409

拡張クラス 419

拡張データ・オブジェクト (XDO) 15

カスタマイズ 405

カスタム・コネクタ用の

feServerDefBase の拡張 422

関数名のリスト作成 420

管理オブジェクトの検索 230

共通特権の入手 420

クラス名 265

結果セット・カーソル 266

結果セット・カーソルの所属の識別
415

コントローラー・サブレット値
482

作成する 265

サポートされるコネクタ 5

参照 419

照会 408

紹介 1

コンテンツ・サーバー (続き)

情報へのアクセス 266

書式オーバーレイ・オブジェクトの検
索 420

接続 408

切断 408

データ定義階層 267

データの管理 13

定義 265

定義用の特定 DK クラス 413

テキスト照会の実行 122

テキスト照会の評価 124

統合 19

統合モデルでの登録 22

動的データ・オブジェクト (DDO) 14

トランザクションの実行 409

名前のリターン 409

バイナリー・データの管理 416

パスワード変更 409

パラメトリック照会の実行 116

パラメトリック照会の評価 118

フォルダーからのメンバーの除去 420

フォルダーへのメンバーの追加 420

文書またはフォルダーの移動 410

文書またはフォルダーの検索 409

文書またはフォルダーの更新 410

文書またはフォルダーの削除 410

文書またはフォルダーのチェックアウ
ト 420

文書またはフォルダーの追加 409

マッピング情報の登録 410

ユーザー ID オブジェクトのリターン
409

ユーザー管理クラス 421

リレーショナル・データベース 397

Bean 440

CLOB コンテンツの管理 418

Content Manager for AS/400 361

Content Manager バージョン 8 141

Content Manager、旧バージョン 267

DDO との関係 16

DDO の作成 409

Domino.Doc 369

EIP がサポートするサーバー 13

Extended Search 376

ImagePlus for OS/390 354

OnDemand 339

Panagon Image Services 391

RMI とともに使用 33

XDO クラス 421

コンテンツ・プロバイダー、情報マイニン
グの 531

コントラスト 308

コントローラー・サーバー

デフォルト 482

コントローラー・サブレット

アクション 480, 481

エラー・メッセージを表示するページ
483

拡張 481

規則 482

クリーンアップ 481

サービス・ランタイム 483

再生 482

参照 482

使用 480

セッション管理 481

接続プール 480

接続プール値 482

ツールキット機能マトリックス 487

トレース 483

名前分離文字パラメーター 484

パラメーター 482, 484

プロパティ・ファイル 481, 484

変換パラメーター 484

要求パラメーター

一般 484

検索 485

項目 485

コンテンツの管理 486

接続関連 485

フォルダー 486

文書関連 487

action 485

reply 485

ロケール 481

JSP セット 481

[サ行]

サービス API

カタログの管理 534

カテゴリの割り当て 540

クラスター化 539

検索 542

サーバー・タスクの実行 542

情報マイニング 532

情報マイニング・ツールの使用 537

接続 533

テキスト文書の作成 538

文書言語の判別 538

文書情報の抽出 539

文書のフィルター操作 538

分類法の管理 534

メタデータの保管 540

要約の生成 539

ライブラリーの管理 534

レコードの作成 540

サブレット、

参照：コントローラー・サブレット

再開、ワークフロー 428

再開リスト 243
作業項目
 アクセス 430
作業ノード
 定義 244
 リスト作成 245
 ルーティング 242
作業パッケージ 243, 257
 ワーク・リスト内の PID スtring
 のリスト作成 257
索引
 テキスト検索 120
索引付け
 イメージ検索 323
削除 416
作成する 168
サブエンティティ
 Domino.Doc でのリスト作成 372
算術照会
 演算 214
 構文 226
参照
 照会による全探索 216
 属性タイプ 156
 Content Manager バージョン 8 145
参照属性 204
サンプル・クライアント 306
 アプリケーション 5
サンプル・ファイル
 Content Manager バージョン 8 154
サンプル・ファイルの位置
 情報マイニング 492
式、照会 218, 226
式クラス 408
指数、照会 225
システム管理クライアント
 カスタマイズ 27
 ユーザー出口 27
 ワーク・リストの作成 429
シナリオ、Content Manager バージョン 8
 の保険サンプル 154
初期設定パラメーター・フィールド、
 OnDemand 340
照会
 イメージ検索 310, 319
 イメージ検索照会の構文 310
 エスケープ・シーケンス 219
 結果のリスト 218
 結合式 408
 言語 219
 言語について 201
 言語の文法 224
 構造化文書の検索 295
 構文 213
 算術演算 214
 実行 112

照会 (続き)
 データ・モデル 211
 テキスト検索 206
統合
 処理 22
統合コレクション 111
バージョン 217
評価 112
複合式 408
平均色検索 312
リレーショナル・データベース 402
ルーティングの例 260
例 210, 213
ワイルドカードの例 217
CM for AS/400 363
Content Manager バージョン 8 202
dkQuery クラス 408
Domino.Doc 374
Domino.Doc の構文 375
Extended Search 381
Generalized Query Language
 (GQL) 381
ImagePlus の構文 359
Java 112
 照会オブジェクト・タイプ 112
 テキスト・タイプ 119
 パラメトリック・タイプ 113
 dkResultSetCursor および
 DKResults 113
OnDemand 検索 344
Panagon Image Services 394
照会可能コレクション
 Java 137
 結果の入手 138
 照会可能と詳細化 139
 評価 138
 プログラミング上のヒント 139
照会言語 219
照会の実行 112
照会の評価 112
条件式、FileNET 396
証券の例 155
情報カタログ
 統合モデルからのマッピング 20
 バージョン 8.2 の制限 7
情報センター 154
情報抽出のサンプル、情報マイニングの
 507
情報の検索 257
情報マイニング
 アプリケーションの作成 489
 拡張検索サンプル 526
 カテゴリー化サンプル 493
 クラスター化のサンプル 512
 検索、カテゴリーによる 526
 サービス API の使用 532

情報マイニング (続き)
 サンプル 489
 サンプル・ファイルの位置 492
 情報抽出のサンプル 507
説明 5
独自のコンテンツ・プロバイダー 531
文書サンプルのインポート 517
要約サンプル 501
Bean 442, 489
Bean サポート 439
JSP アプリケーション 544
 Web Crawler のサンプル 517
除去、CLOB 418
書式オーバーレイ・オブジェクト 420
新機能、バージョン 8.2 API の 6
新機能、バージョン 8.2 の 7
診断情報、
 参照: トレース
シン・クライアント 469
随時ルーティング 259
スキーマ、Bean 476
スキーマ・マッピング
 カスタム・コネクター 406
紹介 14
統合モデルでの関連 21
リレーショナル・データベース 398
dkDatastore メソッド 410
dkSchemaMapping クラス 408
スタック・トレース 153
ストリーミング文書サービス 464
ストレージ・コレクション
 XDO の追加 87
 XDO の変更 89
セッション
 サブレット 481
セッション・リスナー、Bean 444
接続 Bean 476
接続プール
 サブレット 480
 サブレット値 482
設定
 サンプル Java アプレットおよびサー
 ブレットの使用 475
 クライアント Java アプリケーショ
 ン 475
 検索サブレット 475
 リモート・アクセス 476
 ローカル側アクセス 476
C++
 NT 上に構築 36
C++ 環境 33
Java
 クライアント / サーバー 29
 クライアントの接続と切断 43
 プログラミング上のヒント 31
 NT 上 31, 32, 35

設定 (続き)

Java 環境 31
相対位置、結果セット・カーソル 136
属性
 イメージ検索 318
 グループ化 143
 項目タイプごとのリスト作成 164
 項目に対する変更 170
 コントローラー・サブレットでの管理 486
 参照 204
 照会 214
 多値 143
 定義 143, 266
 定義用の特定 DK クラス 412
 バージョン管理 177
 プロパティの検索 54
 文書項目タイプを対象とした検索 230
 ユーザー定義 203
 テキスト検索可能にする 206
 リレーショナル・データベースでのリスト作成 400
Bean 440, 478
Bean 内での表示 447, 456
CM 8 でのグループの更新 163
CM 8 での作成 161
CM 8 での設定と検索 169
CM for AS/400 でのリスト作成 362
CM8 でのグループの管理 162
Content Manager バージョン 8 143
DDO および XDO との比較 16
DDO 内でのアクセス 52
DDO による表現 156
DKAny を対象とした削除 109
Extended Search 377
FileNET でのリスト作成 393
IP OS/390 でのリスト作成 355
OnDemand に関するリスト作成 341

[タ行]

タイムアウトになったセッション、サブレット 481
タグ付き文書
 テキスト検索 295
タグ・ライブラリー
 検索関連 478
 項目関連 478
 スキーマ関連 477
 接続関連 476
 フォルダー関連 479
 文書関連 480
単一必須文字 (SC) 287
注釈
 意味タイプ 168
 エンジン 465

注釈 (続き)

カスタマイズ 473
サービス 470
サービス・クラス 464
定数 168
文書 463
編集のサポート 472
Bean 441, 443
dkAnnotationExt クラス 419
FileNET 393
OnDemand 352
XDO へのオブジェクトの追加 64
ツール
 Content Manager バージョン 8 154
追加パラメーター・フィールド、
 OnDemand 340
データ項目、DDO 51
データ定義クラス 267
データ・ストア、
 参照: コンテンツ・サーバー
データ・モデル、照会
 XML 表現 211
データ・モデル、Content Manager バージョン 8
 照会言語の適用 203
定位置色
 照会例 312
 定義 308
 有効な値 312
定数 41
 ルーティング 262
テキスト検索
 永続的 ID (PID) 285
 演算子 206
 オブジェクト・コンテンツの検索 205
 拡張検索 207
 管理関数 282
 基本照会 206
 構造化文書の検索 295, 305
 コンテンツ・サーバーからの照会 122
 コンテンツ・サーバーからの照会の評価 124
 索引付け規則の設定 299
 索引付けプロセスの開始 302
 照会可能コレクション 139
 照会構文 206
 照会ストリングの定式化 120
 照会の実行 120
 照会例 215
 説明 205
 定義 38
 トレース 38
 ハイライト 124
 複数の索引に対する照会 120
 複数の索引の照会 120
 文書の検索 206

テキスト検索 (続き)

文書モデルの作成 297
文書モデルのリスト作成 295
ユーザー定義属性を検索可能にする 206
ワイルドカード 207
Bean 450
OnDemand 339
score-basic 207
テキスト検索エンジン
Java
 接近照会 283
 データのロードと索引付け 294
 ハイブリッド照会 283
 ヒープ・サイズの設定 268
 プール照会 282
 フリー・テキスト照会 283
 ActiveX 286
 GTR 照会 284
テクスチャー
 コントラスト 308
 照会例 312
 粗度 308
 定義 308
 方向性 308
 有効な値 312
テンプレート
 ワークフロー 432
統合 26
 イテレーター 111
 イメージ検索 24
 エンティティ 20
 検索
 関係図 23
 式クラス 408
 紹介 3, 19
 図 21
 プロセス 22
 Extended Search 391
 項目 19
 コレクション 23, 111
 コンテンツ・サーバーの登録 22
 サーバー定義基本クラス 422
 システム管理機能 27
照会
 結果 23
 構文 24
 作成する 22
 処理図 23
 例 24
スキーマ・マッピング 21
属性 20
フォルダー 26
文書モデル 19
マッピング 20
ユーザー ID のマッピング 22

統合 (続き)

- Bean 478
 - Bean のレイヤー 439
- 統合文書モデル 19
- 動的構成、Bean 439
- 動的データ・オブジェクト (DDO)
 - イメージ検索結果 313
 - イメージ検索での表現 318
 - イメージ検索の 永続的 ID 318
 - 永続 ID 17
 - コレクションのソート 110
 - コンテンツ・サーバーとの関係 16
 - 参照 146
 - 紹介 14
 - 属性との比較 16
 - 属性プロパティの検索 54
 - 属性へのアクセス 52
 - 統合 19
 - フォルダー・コンテンツへのアクセス 100
 - 含むすべてのフォルダーの入手 188
 - プロパティ 268
 - プロパティの追加 49
 - マルチメディア・コンテンツを表現する 16
- CM 8 での属性グループ 162
- COBRA 仕様 14
- C++
 - 削除 57
- dkDDO 406
- Extended Search での識別 382
- FileNET での表現 392
- Java 47
 - 作成する 48
 - 情報、テキスト検索エンジン 284
 - 属性、DKPARTS 95, 98
 - 追加 51
 - データ項目値 51
 - デジタル・ライブラリー情報 156, 268
 - 表示 55
 - プロパティ 53
 - PID 50
- retrieveObject 177
- setData による取り込み 167
- XML インポート 94
- XML エクスポート 95

特権

- アクセス制御リスト (ACL) 151
- コード 150
- 構成済み ACL 152
- システム定義 149
- セット 149
- セットの作成 193
- セットのタイプ 149
- セットのプロパティの表示 195

特権 (続き)

- データ・アクセス 149
- 定義済みの Content Manager バージョン 8 セット 149
- ルーティングのための認可 261
- Bean 441, 479
- CM 8 での作成 192
- CM 8 でのユーザーとユーザー・グループ 150
- Content Manager バージョン 8 148, 149
- トラブルシューティング
 - エラー・メッセージ・プロパティ・ファイル 421
 - トレース 38
- トランザクション、Content Manager バージョン 8 237
- 考慮事項 238
- 処理 240
- チェックインおよびチェックアウト 239
- 明示的トランザクション に関する注意 239
- リスト 240
- トレース
 - エラー状態 153
 - コントローラー・サブレット 483
 - サンプル・ツール 153
 - スタック 153
 - テキスト検索 38
 - パラメトリック・サーチ 39
 - 戻りコード・コンテナー 153
- Bean 445, 476
- DKException によるエラーの処理 153
- FileNET 397
- JNI、サブレット 483
- OnDemand 352

[ハ行]

- バージョン 8.2、新機能 7
- バージョン 8.2、API の新機能 6
- バージョン、照会 217
- バージョン管理
 - アプリケーション制御 147
 - 永続的 ID (PID) 147
 - 項目 177
 - 項目タイプ 180
 - 項目に対する設定 179
 - 属性 177
 - バージョン管理ポリシー 180
 - 文書管理データ・モデルのパーツ 236
 - ポリシー 147
- Bean 447, 457
- Content Manager バージョン 8 146

パーツ

- 旧バージョンの CM 269
- 旧バージョンの CM での更新 275
- バージョン管理 236
- Extended Search 383
- 破棄、結果セット・カーソル 415
- バイナリー・データ、
 - 参照: XDO
- バイナリー・ラージ・オブジェクト (BLOB) 416
- クラス 266
- 検索 416
- 更新 416
- コンテンツの管理 416
- 追加 416
- データのコピー 416
- 特定 DK クラス 415
- 特定 DKPidXDO クラス 421
- 長さのリターン 416
- 引き数データの挿入 417
- 非同期オープン 417
- ファイル・ハンドラー の識別 417
- メソッド 416
- 連結 416
- Extended Search での検索 389
- XDO 関数を使用したテスト 68
- XDO の作成 57
- ハイライト、テキスト検索 124
- それぞれの結果に関する情報の取得 125
- 特定の結果に関する情報の取得 129
- バインダー、Domino.Doc 371
- バインディング、DataJoiner 7
- パスワード
 - 統合モデルでのマッピング 22
 - 変更 409
- Bean 内での変更 458
- Content Manager バージョン 8 での作成 158
- パターン 308
- パッケージ
 - クライアントとサーバー (cs) 30
- Java クライアント 30
- Java サーバー 30
- Java での階層 30
- バッチ入力システム、FileNET 393
- バッファ、XDO の追加 62
- パラメトリック・サーチ
 - 演算子 204
- コンテンツ・サーバーからの照会 116
- コンテンツ・サーバーからの照会の評価 118
- 照会ストリングの定式化 113
- 照会の実行 115
- 説明 204
- 定義 38

パラメトリック・サーチ (続き)

統合 24

トレース 39

複数の基準による照会 114

複数の基準の定式化 114

CM for AS/400 での照会 368

Panagon Image Services 395

ハンドル、結果セット・カーソル 415

汎用文書ビューアー

参照: 文書ビューアー・ツールキット

ヒストグラム色

照会例 312

定義 308

有効な値 311

ヒストリー

意味タイプ 168

定数 168

被制御エンティティ 148

ビューアー、Java 文書

参照: 文書ビューアー・ツールキット

表示タイプ 291

非リソース項目

Content Manager バージョン 8 143

ファイルからの XDO の追加 63

フィーチャー

値、イメージ検索 310

イメージ検索 306

重み、イメージ検索 310

最大結果数、イメージ検索 310

名前、イメージ検索 310

フィールド、Domino.Doc 371

フィールド、Extended Search 383

フィルター条件、FileNET 396

フォルダー

アクセス 100

意味タイプ 168

移動 410

旧バージョンの CM での更新 276

旧バージョンの CM での表現 269

旧バージョンの CM のワークフロー 330

検索 409

更新 410

コンテンツの検索 187

コンテンツの除去 185

コンテンツの追加 183

コントローラー・サブレットでの管理 486

削除 410

チェックアウトおよびチェックイン 420

追加 409

特定の DDO を含むすべてのフォルダーの入手 188

ネイティブ・エンティティとしての OnDemand フォルダー 352

フォルダー (続き)

メンバーの除去 420

メンバーの追加 420

Bean 479

Bean 内での表示 447, 453

CM 8 での作成 182

Content Manager の動作 100

Content Manager バージョン 8 146

OnDemand でのモードの使用可能化 351

OnDemand でのリスト作成 343

複数必須文字 (MC) 287

プロセス

延期 256

関連経路の定義 251

再開 256

終了 255

続行 255

リスト作成 258

ルーティング 242

プロセスの延期 256

プロファイル、Domino.Doc 371

文書 168

意味タイプ 168

移動 410

管理データ・モデルのパーツのバージョン管理 236

旧バージョンの CM での管理の相違点 275

旧バージョンの CM のワークフロー 330

検索 409

更新 410

コントローラー・サブレットへのキャッシング 483

削除 410

チェックアウトおよびチェックイン 420

注釈付け 463

追加 409

データ・モデル内での表現 204

テキスト検索 206

表示、

参照: 文書ビューアー・ツールキット

プロセスにおけるルーティング 241

文書項目タイプの作成 230

ルーティング・プロセスの開始 254

Bean 443, 476, 480

Bean 内での表示 447, 455

Bean によるクラスター化 443

CM 8 での管理 228

CM 8 での管理データ・モデルの作成 230

CM 8 での検索 236

CM 8 での更新 234

文書 (続き)

CM 8 での削除

SDocModelItemICM 236

CM 8 での作成 232

Content Manager バージョン 8 143, 146

Domino.Doc 371

文書タイプ定義 (DTD)、XML インポート 91

文書のルーティング、

参照: ルーティング

文書パーツ

項目タイプ 158

定数 158

Content Manager バージョン 8 143

文書ビューアー・ツールキット

アーキテクチャー 464

アプリケーション例 467

アプレットまたはサブレット 469
エンジン

AFP2Web 文書 465

INSO 文書 465

Java 文書 465

MS-Tech 文書 464

関数 463

紹介 463

シン・クライアント 469

スタンドアロン・ビューアー 468

注釈サービス 470

二重モードおよびアプレットまたはサブレット 470

汎用ビューアーの作成 465

汎用文書ビューアーのカスタマイズ 465

ポップアップ・メニュー 466

Java アプリケーション 468

文書モデル、

参照: 文書パーツ

ページ

Bean 480

FileNET 392

平均色

照会例 312

定義 308

有効な値 311

ヘルパー Bean 440

方向性 308

[マ行]

マルチストーリーミング 29

マルチメディア・コンテンツ 16

メタデータ

情報マイニング 5

メッセージ

プロパティ・ファイル 421

メッセージ (続き)

DKException 情報 153

メディア・オブジェクト

旧バージョンの CM での追加 73

検索 82

コード・サンプル名 89

削除 78

メモ

意味タイプ 168

定数 168

メモ、Bean ログ 479

メンバー

除去 275

追加 275

文字ラージ・オブジェクト (CLOB) 418

一部の削除 418

クラス 417

コンテンツの削除 418

コンテンツの取得 418

コンテンツの追加 418

ファイル・ハンドラー 418

モデル・ビュー・コントローラー
(MVC) 439, 472

[ヤ行]

ユーザー

CM 8 でのアクセス制御 150

ユーザー ID

統合モデルでのマッピング 22

ユーザー定義属性 203

テキスト検索可能にする 206

ユーザー出口

システム管理 27

ワークフロー 433

ユーザー特権

Content Manager バージョン 8 148,
149

有効範囲

タグ・ライブラリー 476

トランザクション 237

要約サンプル、情報マイニングの 501

[ラ行]

ラージ・オブジェクト (LOB) 156

旧バージョンの CM での処理 268

ライブラリー

C++ リスト 33

Java 31, 33

Microsoft Visual Studio .NET 36

ライブラリー・サーバー、Content

Manager バージョン 8 141

バージョン管理 147

リスト作成 157

リソース項目

項目タイプ 158

項目タイプの定義 172

定数 158

Content Manager バージョン 8 143

リソース・コンテンツ、

参照: XDO

リソース・マネージャー、Content

Manager バージョン 8 141

オブジェクトの使用 228

使用 227

バージョン管理 147

リスト作成 157

リテラル

ImagePlus for OS/390 360

リテラル、照会 218, 226

リモート・メソッド呼び出し (RMI)

構成 4

コンテンツ・サーバー・コネクター 5

Bean との接続 439

Java API とともに使用 33

Java での開始 33

リレーショナル・データベース

エンティティのリスト作成 400

構成ストリング 399

照会 402

紹介 397

接続 398

接続ストリング 398

属性のリスト 400

統合モデルからのマッピング 20

リンク

インバウンドとアウトバウンド 190

記述項目 189

項目間での定義 189

照会による全探索 214

ソース 189

属性タイプ 156

ターゲット 189

タイプ名 191

タイプ名定数 191

データ・モデル内での表現 203

リンクされた項目の検索 191

Content Manager バージョン 8 145

ルーティング 442

アクセス制御リスト (ACL) 262

開始 254

コレクション・ポイント 242

サービス・オブジェクトの作成 243

再開リスト 243

作業ノード 242

作業ノードのリスト作成 245

作業パッケージ 243

作業パッケージ情報の検索 257

紹介 241

ルーティング (続き)

新規コレクション・ポイントの定義
246

新規の標準作業ノードの定義 244

新規プロセスと関連経路の定義 251

随時 259

セットアップ 242

定数 262

特権の認可 261

プロセスの延期 256

プロセスの再開 256

プロセスの終了 255

プロセスの続行 255

文書ルーティング・プロセスのリスト

作成 258

リスト、ワーク・リスト 250

例 260

ワーク・リスト 243

ワーク・リスト内のパッケージ PID

のリスト作成 257

ワーク・リストの定義 249

Bean 440, 442

ルート・コンポーネント

参照 145

データ・モデル内での表現 203

バージョン管理 147

リンク 145

CM 8 での作成 166

Content Manager バージョン 8 144

ルーム、Domino.Doc 371

例外、処理 39

ログイン、

参照: トレース

ロケール

サブレット 481

[ワ行]

ワークバスケット

規則 330

旧バージョンの CM での作成 333

旧バージョンの CM での識別 337

旧バージョンの CM での定義 330

旧バージョンの CM でのリスト作成
334

ワークフロー

アクセス、ワーク・リスト 429

延期 427

開始 424

旧バージョンの CM での項目 ID の
リスト作成 335

旧バージョンの CM での作成 331

旧バージョンの CM での実行 337

旧バージョンの CM での定義 330

旧バージョンの CM でのリスト作成
332

ワークフロー (続き)

旧バージョンの Content Manager サービス 330
項目の移動 432
作業項目へのアクセス 430
削除 425
終了 425
紹介 6, 423
すべてのテンプレートのリスト作成 432
すべてのワーク・リストのリスト作成 428
接続 423
切断 424
統合フォルダーから結果を送る 26
モデル 330
リスト作成 426
Bean 438
Bean サポート 439
Java アクションの作成 433
ワーク・パケット
紹介 423
ワーク・リスト 243
アクセス 429
項目の移動 432
作業項目へのアクセス 430
定義 249
パッケージ PID のリスト作成 257
リスト作成 250, 428
Bean 438
ワイルドカード
照会 220
テキスト検索 207
Domino.Doc 375

A

addObject 409, 415
addToFolder 420
add、BLOB 416
AIX
C 用の共用オブジェクト 33
RMI サーバーの開始 33
AllPrivSet 149, 152
anyA、DKAny 105
ASCENDING 224
ATTRONLY、ImagePlus for OS/390 361
AUTOCOMMIT、リレーショナル・データベース 399

B

BasicExpression 226
Bean 442

Bean (続き)

イベント・クラスとリスナー・クラス 444
起動 438
基本概念について 435
検索基準 477
検索結果 476
検索テンプレート 476, 477
検索要求 438
項目 476
紹介 435
照会サービス 476
情報マイニング 442
スキーマ 438
スキーマ管理 476
スレッド化 444
セッション・リスナー 444
接続 438, 476
接続プール
CMBCConnectionPool 476
その他のビルダー 437
注釈 441, 443
データ管理 438, 476
データ・ソース
データ・ソース Bean 476
特性
イベント 436
イントロスペクション 436
カスタマイズ 436
パーシスタンス 436
プロパティ 436
メソッド 436
トレース 445
トレース・ログ 476
バージョン管理 447, 457
バッチ・サポート 438
ビジュアル
アプリケーションの作成 459
一般的な動作 457
ウィンドウ内での使用 460
外部ビューアー 456
キー・イベント 458
検索結果ビューアー 451
検索テンプレート・ビューアー 450
検索テンプレート・リスト 449
検索パネル 451
構成の保管/復元 457
紹介 446
照合強度 457
接続 457, 459
専用動作 458
属性エディター 456
置換 458
ツリー・ペイン 452
定義 435

Bean (続き)

ビジュアル (続き)
テキスト検索領域 450
デフォルトのビューアー 456
名前 446
バージョン・ビューアー 457
ビューアーの指定 455
フォルダー・ビューアー 453
文書ビューアー 455
ヘルプ・イベント 458
ボタンの非表示と表示 457
ポップアップ・メニュー 452
ポップアップ・メニューのオーバーライド 453
ログオン・パネル 447
非ビジュアル
アプリケーションの作成 446
イベント 445
カテゴリー 440
構成 439
考慮事項 444
紹介 438
定義 435
フィーチャー 439
プロパティ 445
ビルダーでの使用 437
フォルダー 479
文書 480
文書サービス 443, 476
ヘルパー 440
補助 441
モデル・ビュー・コントローラー (MVC) 439
ユーザー管理 476
要件 437
例外クラス 444
ワークフロー 438, 441
ワーク・リスト 438
BeanInfo クラス 444
CMBSchemaManagement 438
JAR ファイル 437
Java ビューアー関連
定義 435
OnDemand の検索 352
singleton 444
WebSphere Studio Application Developer での作成 437
web.xml 437
BeanInfo 444
BETWEEN 220, 224

BLOB、
参照： バイナリー・ラージ・オブジェクト (BLOB)

C

CC2MIMEFILE、リレーショナル・データベース 399

cc2mime.ini 399

CCSID 287

changePassword 158, 409

checkedOutUserId 420

checkIn、dkDatastoreExt 420

checkOut、dkDatastoreExt 420

CLOB、

参照： 文字ラージ・オブジェクト (CLOB)

cmb81.jar 437

CMBAnnotationPropertiesInterface 474

CMBAnnotationServices 466

CMBAnnotationServicesCallbacks 466

CMBAnnotationSet 466

CMBCC2MIME.INI 390

cmbcc2mime.ini 483

cmbcc2mime.ini.samp 390

cmbclient.ini 4, 483

cmbcm817d.dll 406

cmbcm817.dll 406

cmbcm81d.lib 33

cmbcm81.jar 406

cmbcm81.lib 33

CMBConnection 352, 438, 476

cmbcs.ini 439, 483

CMBDataManagement 438, 476

cmbdes.ini 376

CMBDocumentServices 469, 476

CMBDocumentViewer 447, 455

指定 455

終了 455

CMBFolderViewer 447, 453

CMBGenericDocViewre 466

CMBItem 476

CMBItemAttributesEditor 456

CMBLogonPanel 447

CMBObject 478

CMBPage 480

CMBPageAnnotation 472

CMBQueryService 438, 476

cmbregist81.bat 33

cmbregist81.sh 33

CMBSchemaManagement 476

CMBSearchPanel 451

CMBSearchResults 476

CMBSearchResultsView 352

CMBSearchResultsViewer 447, 451

CMBSearchTemplate 476

CMBSearchTemplateList 352, 447, 449

CMBSearchTemplateViewer 352, 447, 450

CMBServletAction 481

cmbservlet.jsp.properties 481

cmbservlet.properties 481

CMBSTCriterion 477

CMBStreamingDocServices 466

CMBStreamingDocServicesCallbacks 466

cmbsvclient.ini 483

cmbsvcs.ini 483

CMBTraceLog 476

CMBUserManagement 476

cmbview81.jar 465

CMBViewerConfiguration.properties 465

CMCOMMON 4

CMCOMMON_URL 4

cmvcmenv.properties 4

COBRA

Persistent Data Service (PDS) 14

Persistent Object Service 14

CompareOperator 226

Comparison 226

com.ibm.mm.sdk.client 30

com.ibm.mm.sdk.common package 406

com.ibm.mm.sdk.server 30

concatReplace 416

contains-text、テキスト検索 207

contains-text-basic、テキスト検索 206

CONTENT

FileNET 397

ImagePlus for OS/390 361

Content Manager for AS/400

エンティティおよび属性のリスト作成 362

索引クラス 362

紹介 361

照会の実行 363

統合モデルからのマッピング 20

Content Manager バージョン 8

アクセス権の制御 149

アクセス制御 148

アクセス制御の使用 192

アプリケーションの計画 152

アプリケーションの作成 155

オブジェクト 145

基本概念 142

検索照会について 201

項目 143

項目属性の設定と検索 169

項目属性の変更 170

項目タイプ 143

項目タイプの作成 158

項目タイプの属性のリスト作成 164

項目タイプのリスト取得 160

項目の検索 175

Content Manager バージョン 8 (続き)

項目のチェックインおよびチェックアウト 178

項目のリンク 189

子コンポーネント 144

コンテンツ・サーバーの作成 156

コンポーネント図 142

参照 145

サンプル 154

照会 202

紹介 141

照会言語について 201

照会例 210

制御アクセス

アクセス・リスト 151

ユーザー・グループ 150

接続 156

属性 143

属性グループの管理 162

属性の作成 161

統合モデルからのマッピング 20

特権セット 149

トランザクション 237

バージョン管理 146

パスワードの作成 158

非リソース項目 143

フォルダー 146

フォルダーの使用 182

文書 143, 146

文書の管理 228

文書の検索 236

文書の更新 234

文書の削除 236

文書の作成 232

文書のルーティング 241

文書パーツ 143

保険シナリオ・サンプル 154

ライブラリー・サーバー 141

ライブラリー・サーバーおよびリソース・マネージャーの一貫性 237

リソース項目 143

リソース項目タイプの定義 172

リソース・マネージャー 141

リソース・マネージャーの使用 227

リンク 145

ルート・コンポーネント 144

eClient 5

System Managed Storage (SMS) サーバ

ー 141

XML 89

Content Manager、旧バージョン

イメージ検索 6, 306

永続的 ID (PID) 269

紹介 267

統合モデルからのマッピング 20

パーツの更新 275

Content Manager、旧バージョン (続き)

パーツの表現 269
フォルダーの更新 276
フォルダーの表現 269
文書の表現 269
ラージ・オブジェクトの処理 268
ワークフロー 330

eClient 5
XML の保管 93

createChildDDO 166

createDDO 48

cs パッケージ 30

C++

エスケープ・シーケンス 224
エラー・メッセージ・プロパティ・
ファイル 421
環境のセットアップ 33
コネクター 5
新規クラス 9
定数 41
変更されたクラス 10
ライブラリー 33
リレーショナル・データベース構成ス
トリング 58

AIX 用の共用オブジェクト 33

DKAny 101

DKConstant.h 29

DLL ファイル 33

XML サポート 29

D

databaseNameStr 157

DataJoiner、

参照： DB2 DataJoiner

DATASOURCE 376

data_id 54

DB2

クライアント構成アシスタント 4, 33

クライアント・サポート 4, 33

構成ストリング 58

DB2 Data Warehouse Manager Information
Catalog、

参照： 情報カタログ

DB2 DataJoiner 397

構成ストリング 58

バージョン 8.2 の制限 6

DB2 Net Search Engine (NSE) 38

DB2 エクステンダー 15

db2cliws_dj.bnd 7

db2clprj_dj.bnd 7

ddoDocument 64

ddo.dtd 91

DEFINED、 FileNET 395

deleteDKAttributeDef 109

deleteObject 409, 410

del、 BLOB 416

DemoSimpleAppl.java 446

DESCENDING 224

DfltACLCode 152

DIGIT 225

DIV 224

dkAbstractWorkflowUserExit 433

dkAnnotationExt 419

DKAny

型の検査 104

コードの入力、入手 103

コレクション内での使用 105

コレクションの削除 109

タイプ・コンストラクターの使用 102

バージョン 8.2 の変更 11

破棄中 104

表示する 104

プログラミング上のヒント 105

メモリー管理 102

割り当て先 103

割り当て元 103

DKAny 102

dkAttrDef 266

クラス 412

DKAttrFieldDefDD 371

DKAttrGroupDefICM 163

DKAttrKeywordDefDD 371

DKAttrProfileDefDD 371

DKBinderDefDD 371

dkBlob 266

クラス 415

メソッド 416

DKBlobDL

setToBeIndexed 323

DKBlobFN 393

DKCabinetDefDD 371

dkClob

クラス 417

dkCollection 407

DKCollectionResumeListEntryICM 243

DKConstant

定数

共通 421

DKConstantFN 395

DKConstants 41

DKConstant.h 29

DKCQExpr 394

dkCQExpr 408

DKDatastore 265

カスタム・コネクター 406

dkDatastore

紹介 408

addObject 409

changePassword 409

commit 409

connect 408

dkDatastore (続き)

deleteObject 409, 410

disconnect 408

evaluate 408

execute 408

executeWithCallback 408

listDataSourceNames 409

listDataSources 409

listMappingNames 410

moveObject 410

registerMapping 410

registerServices 409

retrieveObject 409

rollback 409

unRegisterMapping 410

unregisterServices 409

updateObject 410

DKDatastoreAccessError 397

DKDatastoreDef

メソッド 410

dkDatastoreDef 265

クラス 410

DKDatastoreDefDL

追加関数 411

DKDatastoreDefOD

追加関数 411

DKDatastoreDES

照会 381

listEntities 377

listEntityAttrs 377

DKDatastoreDL

Java 41

サーバーのリスト 43

スキーマおよびスキーマ属性のリス
ト 44

接続 42

DKDatastoreDL オプション 43

dkDatastoreExt

関数 419

クラス 419

DKDatastoreFN 391

DKDatastoreICM 155

DKDatastoreIP 354

DKDatastoreOD 339

listEntities 341

DKDatastoreQBIC 307

DKDatastoreTS

属性 284

Java 282

サーバーのリスト 287

スキーマのリスト 289

接続 285

DKDatastoreTS オプション 287

DKDatastoreV4 362

DKDatastorexx 265

DKDatstoreIP
listEntityAttrs 357
DKDatstoreOD
listSearchTemplates 343
dkDDO 406
DKDDO、
参照： 動的データ・オブジェクト
(DDO)
DKDLITEMID 284
DKDocRoutingServiceICM 242
DKDocRoutingServiceMgmtICM 242
DKDocumentDefDD 371
DKDSIZE 284
dkEntityDef 265
関数 412
クラス 411
DKEntityDefIP
listAttrNames 355
DkEntityDefIP
getAttr 355
DKException 39, 153
DKFederatedCollection 111
dkFederatedIterator 111
DKFederatedQuery 22
dkFederatedQuery 111
DKFixedView 339
DKFixedViewDataOD 340
DKFolder 269
dkIterator 111
バージョン 8.2 の変更 9
DKLink 189
DKLinkCollection 191
DKLITEMID 318
DKMessageId 421
DKMessageID、FileNET 397
DKMessage_en.properties 421
DKMessage_en_US.properties 421
DKMessage_es.properties 421
DKMessage_es_ES.propertie 421
DKParametricQuery 395
DKPARTNO 284, 318
DKParts
旧バージョンの CM 269
Extended Search 383
DKPidXDO 421
DKPrivilegeSetICM 193
DKProcessICM 242
dkQuery 408
dkQueryableCollection 407
DKRANK 269, 284, 318, 326
DKRCNT 284
DKREPTYPE 284, 318
DKResource 339
DKResourceGrpOD 340
DKResults 407
イテレーターの位置決め 112

dkResultSetCursor 266
関数 414
DKResumeListEntryICM 243
DKRMConfiguration 157
DKRoomDefDD 371
DKRouteListEntryICM 243
dkSchemaMapping 410
DKSequentialCollection 275, 276, 407
dkSequentialIterator 407
dkServerDef 266
関数 413
クラス 413
dkSort 110
DKStorageManageInfo 87
DKString
バージョン 8.2 の変更 11
DKTimestamp
延期、ワークフロー 427
dkUserManagement 421
DkViewOD 340
DKViews 339
DKWorkBasketDL 330
DKWorkflowFed
延期 427
再開 428
DKWorkflowServiceDL 330
DKWorkflowServiceFed
svWf 426
DKWorkflowServicesFed 423
dkWorkflowUserExit 433
DKWorkItemFed
checkIn 432
checkOut 432
DKWorkListFed 429
DKWorkListICM 243, 250
DKWorkNodeICM 243, 245
DKWorkPackageICM 243, 257
dkXDO 418
バージョン 8.2 の変更 9
dkXDOBase 407
open 418
DK_CM_DOCUMENT 269
DK_CM_FOLDER 269
DK_CM_OPT_ACCESS_MODE 266
DK_CM_OPT_CONTENT 397
DK_CM_PROPERTY_ITEM_TYPE 269
DK_CM_READWRITE 266
DK_CM_VERSION_LATEST 177
DK_DES_GQL_QL_TYPE 381
DK_DL_OPT_ACCESS_MODE 415
DK_OPT_TS_CCSID 287
DK_OPT_TS_LANG 287
DK_READWRITE 415
DK_SS_CONFIG 332, 333
DK_SS_NORMAL 332, 333
DK_TS_DOCFMT_HTML 295

DLL ファイル
C++ 33
DLSEARCH_DocType 113
doAction 433
Domino.Doc
エンティティとサブエンティティ
のリスト作成 372
オープン文書管理 API (ODMA) 369
オブジェクト・モデル 370
キャビネット属性のリスト作成 374
照会 374
紹介 369
照会構文 375
統合モデルからのマッピング 20
DKResults オブジェクトの照会 113
DSNAME、リレーショナル・データベー
ス 399
dsType 352

E

eClient
紹介 5
JavaBeans によって作成される 435
EIP ワークフロー・サービス、
参照： ワークフロー
Enterprise Information Portal
永続的 ID (PID) 17
概念 13
拡張データ・オブジェクト (XDO) 15
共通クラス 406
コネクター・クラス 5
コンポーネント 4
サポートされるコンテンツ・サーバー
13
紹介 1
新機能 7
スキーマ・マッピング 14
データベース 14
データベース・インフラストラクチャ
ー 406
動的データ・オブジェクト (DDO) 14
複数層アーキテクチャー 2
文書エンジン 464
編成ダイアグラム 13
保険シナリオ 1
マイグレーション 4
ワークフロー機能 6
Enterprise Java Beans (EJB) 439
ENTITY_TYPE 340, 351
ErrorCode 153
ErrorState 153
ESCAPE、KEYWORD、照会 225
ESCAPE_LITERAL 225
EXCEPT 218, 224
executeWithCallback 408

ExpressionList 226
ExpressionWithOptionalSortBy 225
Extended Search
 エンティティおよび属性のリスト作成 377
 サーバーのリスト作成 376
 紹介 376
 統合検索 391
 統合モデルからのマッピング 20
 内容の処理 383
 フィールドの処理 383
 文書の検索 388
 BLOB の検索 389
 DDO の識別 382
 GQL による照会 381
 MIME タイプの関連付け 390
 PID の作成 383
Extensible Markup Language、
 参照: XML

F

FASERVERTYPES テーブル 422
FeServerDefBase 422
fetchNext 414
fetchObject 414
FileNET、
 参照: Panagon Image Services
findObject 414
FLoadSampleTSQBICDL 291
FLOAT_LITERAL 225
fromXML 90
FTSearch 374
FunctionName 226
F_DOCFORMAT 392
F_DOCTYPE 392

G

Generalized Query Language (GQL)
 式 381
 SQL との違い 391
getCommonPrivilege 420
getContent 416
getContentToClientFile 416, 418
getDatastore 419
getOpenHandler 417, 418
getPosition 414

I

IBM Enterprise Information Portal for
 Multiplatforms
 コンポーネント 4
 管理クライアント 4

IBM Enterprise Information Portal for
 Multiplatforms (続き)
 コンポーネント (続き)
 管理データベース 4
 コネクター 5
 サンプル・クライアント・アプリケーション 5
ICM コネクター、
 参照: Content Manager パージョン 8
ICMLogon 150
ID
 イメージ検索 307, 309
IDENTIFIER 225
ImagePlus for OS/390
 エンティティのリスト作成 355
 紹介 354
 照会構文 359
 照会パラメーター 360
 属性のリスト 355
 統合モデルからのマッピング 20
 eClient 5
imldiag.log 304
IN RANGE 395
INBOUNDLINK 204
indexOf 416, 418
insert、BLOB 417
INTERGER_LITERAL 225
INTERSECT 218, 224
IS 224
isBegin 414
isCheckedOut 420
isEnd 414
IsFTIndexed 374
isInBetween 414
isOpen 415
isOpenSynchronous 417, 419
isScrollable 414
isSupported 419
isUpdatable 414
isValid 414
ItemAdd 150
ItemAdminPrivSet 149
ItemQuery 150
ItemReadPrivSet 152
ItemSetSysAttr 150
ItemSetUserAttr 150
ItemSQLSelect 150
ItemTypeQuery 150

J

j2ee.jar 437
Java
 エスケープ・シーケンス 224
 エラー・メッセージ・プロパティ・ファイル 421

Java (続き)
 ガーベッジ・コレクター 29
 環境のセットアップ 31
 コネクター 5
 新規クラス 8
 定数 41
 パッケージ 30
 文書ビューアー・ツールキット 463
 変更されたクラス 9
 ワークフロー・アクションの作成 433
 DKConstant.h 29
 FeServerDefBase 422
 JVM スタック・サイズの拡大 351
 XML 関数 29
 XML の使用 89
Java Database Connectivity (JDBC) 397
Java Server Pages (JSP)
 サーブレット 481
JavaBeans、
 参照: Bean
java.lang.exception 153
java.lang.objects 169
JDBC DRIVER、リレーショナル・データベース 399
JDBC SERVERS FILE、リレーショナル・データベース 399
JDBC SERVERS URL、リレーショナル・データベース 399
JNI トレース、サーブレット 483
JSP アプリケーション、情報マイニングの
 サンプル 544
 WAS の配置 545
JSP、
 参照: Java Server Pages (JSP)

K

KEY 395

L

LANG 287
length、BLOB 416
LETTER 225
LIKE 220, 224
LIKE、FileNET 395
LinkTypeName 189
ListConstructor 226
ListContent 226
listDataSourceNames 409
listDataSources 157, 409
listEntityNames 160
listFunctions
 dkDatastoreExt 420

listMappingNames 410
listResourceMgrs 157
listWorkflowTemplates 432
listWorkLists 428
LoadFolderTSQBICDL 291
LocalPart 226
LogicalOrSetExpression 225, 226
LogicalOrSetPrimitive 226
LorgicalOrSetTerm 226

M

MATCH_DICT 124
MATCH_INFO 124
MAX_RESULTS、FileNET 396
Microsoft Visual C++ コンパイラー、
参照： Visual C++ コンパイラー
Microsoft Visual Studio .NET 36
MIME タイプ
リソース項目に対する設定 174
Extended Search での関連付け 390
Mime2App プロパティ 455
MOD 224
moveObject 410
moveObject、dkDatastoreExt 420
msg コマンド、FileNET 397

N

nameOfAttrStr 170
NameText 226
NATIVECONNECTSTRING、リレーショナ
ル・データベース 398
NoAccessACL 152
NodeGenerator 226
NONZERO 225
NoPrivSet 149
NOT 224
NULL、照会 225

O

Object Management Group (OMG) 14
OnDemand
アプリケーション・グループの照会
345
アプリケーション・グループのリスト
341
サーバーおよび文書の表現 339
サーバー情報のリスト作成 341
紹介 339
接続 340
切断 340
属性の表示 347
注釈 352

OnDemand (続き)
統合モデルからのマッピング 20
トレース 352
非同期検索 351
フォルダーの検索テンプレートとして
の使用 352
フォルダーのリスト作成 343
フォルダー・モードの使用可能化 351
プロパティ名 339
文書の検索 344
文書の表示 347
Open Database Connectivity (ODBC) 397
構成ストリング 58
open、BLOB 417
open、CLOB 418
OptionalExpressionList 226
OptionalPredicateList 226
OR 224
output_option 43

P

Panagon Capture Software 393
Panagon Image Services
演算子 395
エンティティおよび属性のリスト作
成 393
オプション検索パラメーター 396
クラス 392
サポートされるデータ・タイプ 392
サンプル 395
照会 394
紹介 391
照会構文 395
トラブルシューティング 397
文書クラス 393
文書とページの表現 392
PENDING、ImagePlus for OS/390 361
pEnt、C++ 47
Persistent Data Service
(PDS)、COBRA 14
Persistent Object Service、COBRA 14
Predicae 226
PrivSetCode 149
PublicReadACL 152

Q

QbColor 311
QbColorFeatureClass 308, 311
QbColorHistogramFeatureClass 308, 311,
312
QbDraw 312
QbDrawFeatureClass 308, 312
QbHistogram 311

QbTexture 312
QbTextureFeatureClass 308, 312
QName 226

R

REFERENCEDBY 204
REFERENCER 204
registerMapping 410
registerServices 409
removeAllElement 109
removeFromFolder 420
removeMember 275
remove、BLOB
バイナリー・ラージ・オブジェクト
(BLOB)
一部の除去 417
replaceElementAt 109
ReplayAction、サーブレット 482
retrieveFromOverlay 420
retrieveObject 409
retrieve、BLOB 416
RMI サーバー、
参照： リモート・メソッド呼び出し
(RMI)
rollback 409

S

SAttributeDefinitionCreationICM 143
SConnectDisconnectICM 153
score-basic、テキスト検索 206
テキスト検索
contains-text 207
SDocModelItemICM 146
SDocRoutingDefinitionCreationICM 244,
245, 249, 250, 254
SDocRoutingListingICM 246, 251, 257,
259
SDocRoutingProcessingICM 254, 255, 256,
258
searchTemplate Bean 477
SequencedValue 226
setClassOpenHandler 417, 418
setContent 416
setContentFromClientFile 416, 418
setData 167
setDatastore 419
setInstanceOpenHandler 417, 418
setPosition 414
setToBeIndexed 323
setToFirstCollection 112
setToLastCollection 112
setToNext 414
setToNextCollection 112

setToPreviousCollection 112
SFolderICM 146, 182, 188
SItemCreationICM 146
SItemDeletionICM 178
SItemRetrievalICM 178
SItemTypeCreationICM 144
SItemTypeRetrievalICM 181
SItemUpdateICM 175, 179
SLinksICM 145, 190, 191
SLinkTypeDefinitionCreationICM 191
SORTBY 224, 225
SortSpec 226
SortSpecList 226
SOURCEITEMREF 204
SReferenceAttrDefCreationICM 146, 204
SResourceItemMimeTypeTypesICM 174
SSearchICM 175
Step、照会 226
STRING_LITERAL 225
subString 417, 418
SuperUserACL 152
svWF 426
SYSREFERENCEATTRS 204
System Managed Storage (SMS)、Content
Manager バージョン 8 141
SystemAdmin 150
SystemAdminPrivSet 149
SystemDefineItemType 150

T

taglib.tld 437
TARGETITEMREF 214
TCallbackOD 352
TCheckStatusTS 304
TCP/IP
テキスト検索 286
TExportICM 90
TExportPackageICM 90
text information extender (TIE)
Net Search Engine (NSE) 38
TImageAnnotation 474
TImportICM 90
toXML 90
TRetrieveFolderWithCallbackOD 352
TRetrieveWithCallbackOD 352
TxdoAdd サンプル 89
TxdoAsyncRetDL 268

U

UNICODE_CHARACTER 225
uniform resource identifier (URI) 228
UNION 218, 224
unlockCheckedOut 420

unRegisterMapping 410
unregisterServices 409
UpdateFTIndex 374
updateObject 171, 410
update、BLOB 416

V

valueObj 170
VideoCharger 141
Visual C++ コンパイラー 33
Visual Studio.NET 36

W

W3C XML Query 201
wakeUpService 294
WAL PRS_Parse 395
web crawler
紹介 6
Bean 443
Web Crawler のサンプル、情報マイニン
グの 517
WebSphere Studio Application Developer
Bean の作成 437
web.xml 437
Windows
コンソール・サブシステムの設定 37
C++ DLL ファイル 33
RMI サーバーの開始 33

X

XDO
イメージ検索 での索引付け 323
オブジェクト 145
関数の呼び出し 68
旧バージョンの CM での索引付け
58
クラス 407
クラス・タイプ階層 173
検索 66
更新 66
削除 66, 78
紹介 15
ストレージ・コレクションの変更 89
ストレージ・コレクションへの追加
87
属性との比較 16
注釈オブジェクトの追加 64
テキスト検索 216
特定 DK クラス 421
バッファからの追加 62
ファイルからの追加 63

XDO (続き)

マルチメディア・コンテンツを表現す
る 16
メディア・オブジェクトの追加 73
メディア・オブジェクトの取り出し
82
メンバー 418
ラージ・オブジェクト 156
リソース項目の作成 173
例 66
dkBlob 415
dkXDO 407
FileNET 393
Java 57
索引付け 291
データ・プロパティ 58
独立型 62
プログラミング上のヒント 59, 60
DDO の一部 60
PID 58
MIME タイプの設定 174
XML エクスポート 95

XML

インポート 90
インポート用の文書タイプ定義
(DTD) 91
エクスポート 95
サンプル・ツール 90
紹介 89
照会例のデータ・モデルの表現 211
抽出
バッファ 93
ファイル 93
Web アドレス 94
例 93
Bean タグ・ライブラリー 437
CM へのインポート 94
CM への保管 92
Java サポート 29
W3C 照会 201
XQuery Path Expressions (XQPE) 201



プログラム番号: 5724-B43

Printed in Japan

SC88-9205-01



日本アイ・ビー・エム株式会社
〒106-8711 東京都港区六本木3-2-12