

멀티플랫폼용 IBM Content Manager/
컨텐츠용 IBM Information Integrator



워크스테이션 응용프로그램 프로그래밍 안내서

버전 8 릴리스 2

멀티플랫폼용 IBM Content Manager/
컨텐츠용 IBM Information Integrator



워크스테이션 응용프로그램 프로그래밍 안내서

버전 8 릴리스 2

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 573 페이지의 『주의사항』에 있는 정보를 읽으십시오.

제 2 판(2003년 3월)

이 개정판은 새 개정판에 별도로 명시하지 않는 한, 멀티플랫폼용 IBM Enterprise Information Portal, 멀티플랫폼용 IBM Content Manager(제품 번호 5724-B43, 5724-B19)의 버전 8 릴리스 2 및 모든 후속 릴리스와 수정판에 적용됩니다.

Portions of this product are: Copyright © 1990-2000 ActionPoint, Inc. and/or its licensors, 1299 Parkmoor Drive, San Jose, CA 95126 U.S.A. All rights reserved.

CUP Parser Generator 저작권 표시, 라이선스 및 면책사항은 Scott Hudson, Frank Flannery, C. Scott Ananian에서 그 저작권을 보유합니다. 라이선스 사용자는 추가 비용 없이 CUP Parser Generator 소프트웨어 및 문서를 어떤 용도로든 사용, 복사, 수정 및 배포할 수 있습니다. 단, 위의 저작권 표시를 모든 사본에 표시하고 본 저작권 표시 및 허가 표시와 보증 면책사항을 보조 문서에 표시해야 하며, 이 책의 저자 또는 그 고용주의 이름을 사전 서면 승인 없이 본 소프트웨어와 관련된 광고 또는 선전물에 사용해서는 안 됩니다.

이 책의 저자 및 그 고용주는 본 소프트웨어와 관련하여 상품성 및 특정 목적에의 적합성에 대한 모든 묵시적 보증을 포함하여 어떠한 보증도 하지 않습니다. 이 책의 저자 및 그 고용주는 계약 불이행, 불법 행위 등 손해 발생의 원인을 불문하고 본 소프트웨어의 사용 또는 수행으로 인해 발생한 모든 특별 손해, 간접 손해 또는 결과적 손해 및 용익권과 데이터의 분실 또는 기대했던 이익이 실현되지 못함으로 인하여 발생하는 손해에 대해 일체 책임을 지지 않습니다.

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

목차

이 책에 대하여	ix
이 책의 사용자	ix
추가 정보	ix
제품 패키지에 포함된 정보.	x
웹에서 이용할 수 있는 지원	xi
의견을 보내는 방법	xi
Enterprise Information Portal 버전 8의 새로운 기 능	xii
Content Manager 버전 8의 새로운 기능	xiv
Enterprise Information Portal 소개	1
고객 정보 검색	1
요구사항	2
솔루션	2
Enterprise Information Portal 솔루션	2
멀티플랫폼용 IBM Enterprise Information Portal 구성요소	4
버전 8.2 API의 새로운 기능	6
버전 8.1 API의 새로운 기능	7
신규 및 변경된 Java 클래스	8
신규 및 변경된 C++ 클래스	10
Enterprise Information Portal 응용프로그램 프로 그래밍 개념	13
컨텐츠 서버를 통한 데이터 액세스 이해	13
동적 데이터 오브젝트 개념 이해	14
동적 데이터 오브젝트(DDO).	14
확장 데이터 오브젝트(XDO).	15
멀티미디어 컨텐츠 표현	16
컨텐츠 서버 및 DDO 이해	16
DDO/XDO와 속성값 및 항목 부분 비교	16
지속 식별자(PID) 이해	16
연합 컨텐츠 서버 및 연합 검색에 대한 작업	19
연합 스키마 맵핑	21
연합 컨텐츠 서버 맵핑 구성요소 사용	21
연합 조회 실행	22
연합 조회 구문규칙	23
연합 폴더에 조회 결과 저장(Java 전용)	26
시스템 관리에 대한 작업	27
EIP 시스템 관리 클라이언트 사용자 조정	27

API(Application Programming Interface)로 프로 그래밍	29
Java 및 C++ API 간의 차이점 이해	29
클라이언트/서버 아키텍처 이해(Java 전용)	30
Java 환경을 위한 패키지	30
프로그래밍 팁	31
Java 환경 설정(Java 전용)	31
Windows에 대해 Java 환경 변수 설정	31
AIX에 대해 Java 환경 변수 설정	32
Solaris에 대해 Java 환경 변수 설정	32
컨텐츠 서버에 RMI(Remote Method Invocation) 사용	33
C++ 환경 설정(C++ 전용)	33
Windows용 C++ 환경 변수 설정	35
AIX용 C++ 환경 변수 설정	35
C++ 프로그램 빌드	36
Windows에서 코드 페이지 변환을 위한 콘솔 서 브시스템 설정	37
다중 검색 옵션 이해	37
추적	38
텍스트 검색 엔진을 사용하는 텍스트 조회 추적	38
매개변수식 조회 추적	39
예외 처리	39
상수	41
컨텐츠 서버에 연결	42
연결 설정	42
클라이언트의 컨텐츠 서버에서 연결 및 연결 해제	43
컨텐츠 서버 옵션 설정 및 가져오기	43
컨텐츠 서버 목록	44
컨텐츠 서버의 엔티티 및 속성 목록	45
동적 데이터 오브젝트(DDO)에 대한 작업	48
DKDDO 작성	49
DDO에 등록 정보 추가	51
지속 식별자(PID) 작성	52
데이터 항목 및 등록 정보에 대한 작업	53
DKDDO 및 속성 등록 정보 가져오기	55
전체 DDO 표시	57
DDO 삭제(C++ 전용).	59
확장 데이터 오브젝트(XDO)에 대한 작업	59
XDO 지속 식별자(PID) 사용	60
XDO 등록 정보 이해	60

DB2, ODBC 및 DataJoiner 구성 문자열(C++ 전용)	61	속성	147
Java 프로그래밍 팁.	62	항목 유형	147
C++ 프로그래밍 팁.	62	루트 및 하위 구성요소	148
XDO를 DDO의 부분으로 프로그래밍	63	오브젝트	149
독립형 XDO 프로그래밍	65	링크 및 참조	149
XDO에 대한 작업 예제	69	문서	150
XML에 대한 작업(Java 전용)	91	폴더	150
I XML을 가져오고 내보내는 성능 확장	92	버전화.	151
XML 문서 가져오기	92	액세스 제어.	152
XML 내보내기	97	Content Manager 응용프로그램 계획	156
문서 작성 및 DKPARTS 속성 사용	98	응용프로그램 기능 결정	157
폴더 작성 및 DKFOLDER 속성 사용	101	오류 처리	157
DKAny 사용(C++ 전용)	104	Content Manager 샘플에 대한 작업	158
유형 코드 사용.	105	보험 시나리오 샘플	159
DKAny에서 메모리 관리	105	Content Manager 응용프로그램 작성	159
구성자 사용.	105	소프트웨어 구성요소 이해	159
유형 코드 가져오기	106	DDO를 사용한 항목 표현	160
DKAny에 새 값 할당	106	Content Manager 시스템에 연결.	161
DKAny에서 값 할당.	106	항목에 대한 작업	163
DKAny 표시	108	폴더에 대한 작업	189
DKAny 제거	108	항목 간의 링크 정의	197
프로그래밍 팁	108	액세스 제어에 대한 작업	200
컬렉션 및 반복자 사용	109	사용 권한 작성.	200
순차 컬렉션 메소드 사용	109	사용 권한 세트 작성	202
순차 반복자 사용	109	사용 권한 세트 등록 정보 표시	204
컬렉션의 메모리 관리(C++ 전용).	112	액세스 제어 목록(ACL).	205
컬렉션 정렬.	114	ACL 정보 검색 및 표시	207
연합 컬렉션 및 반복자 이해	114	항목 유형에 ACL 할당	208
컨텐츠 서버 조회	116	항목에 ACL 할당.	209
dkResultSetCursor와 DKResults 간의 차이	117	조회 언어 이해.	211
매개변수식 조회 사용	117	Content Manager 서버 조회	211
텍스트 조회 사용	124	Content Manager 데이터 모델에 조회 언어 적용	212
결과 세트 커서 사용	138	매개변수식 검색 이해.	214
조회를 재실행하기 위해 결과 세트 커서 열기 및 닫기	138	텍스트 검색 이해	215
결과 세트 커서의 위치 설정 및 가져오기	138	오브젝트 콘텐츠 검색.	215
결과 세트 커서에서 컬렉션 작성	141	문서 검색	215
컬렉션 조회.	142	사용자 정의 속성 텍스트 검색 가능하게 만들기	216
조회 결과 얻기.	142	텍스트 검색 구문규칙 이해.	216
새 조회 평가	142	결합된 매개변수식 및 텍스트 검색 작성	217
결합된 조회 대신 조회 가능 컬렉션 사용	144	조회 예제	220
Content Manager 버전 8.2에 대한 작업	145	조회 예제	223
Content Manager 시스템 이해	145	조회 언어 이해.	230
Content Manager 개념 이해	146	비교 연산자와 이스케이프 시퀀스 사용("=", "!=", ">", "<", "BETWEEN" 및 기타)	230
항목	147	LIKE 연산자와 순서 이탈 사용	230

고급 텍스트 검색과 순서 이탈 사용("Include" 및 "Score" 함수)	232
기본 텍스트 검색과 이스케이프 시퀀스 사용 ("contains-text-basic" 및 "score-basic" 함수)	233
Java 및 C++에서 순서 이탈 사용	234
조회 언어 문법.	235
자원 관리자에 대한 작업	238
자원 관리자 오브젝트에 대한 작업	239
Content Manager에서 문서 관리.	240
문서 관리 데이터 모델 작성	241
문서 항목 유형 작성	241
문서 작성	243
문서 갱신	245
문서 검색 및 삭제	247
문서 관리 데이터 모델의 부분 버전화	248
트랜잭션에 대한 작업	249
응용프로그램에서 트랜잭션을 설계할 때 고려해야 할 사항	250
명시적 트랜잭션을 사용할 때의 주의사항	250
트랜잭션에서 체크인 및 체크아웃 사용	250
트랜잭션 처리	251
프로세스를 통한 문서 경로지정	252
문서 경로지정 프로세스 이해	252
문서 경로지정 프로세스 설정	253
기타 콘텐츠 서버에 대한 작업.	275
이전 Content Manager에 대한 작업	277
대형 오브젝트 처리	278
DDO를 사용하여 이전 Content Manager 콘텐츠 표현	278
문서 또는 폴더 작성, 갱신 및 삭제	280
문서 또는 폴더 검색	289
텍스트 검색의 이해(텍스트 검색 엔진)	294
콘텐츠별 이미지 검색.	319
이미지 검색 응용프로그램 사용	321
QBIC에서 연결 설정.	326
이미지 검색 서버 나열	327
이미지 검색 데이터베이스, 카탈로그 및 기능 나열	329
DDO를 사용하여 이미지 검색 정보 표시	331
이미지 조회에 대한 작업	332
이미지 검색 엔진 사용	336
검색 엔진을 사용하여 기존 XDO 색인화	336
결합된 조회 사용	339
이전 Content Manager 워크플로우 및 작업함 기능 이해	343
OnDemand에 대한 작업	353

OnDemand Server 및 문서 표현	354
OnDemand Server에 연결 및 연결 해제	354
OnDemand 정보 나열	355
OnDemand 문서 검색	359
OnDemand 폴더 모드 사용	366
비동기 검색	366
검색 템플릿으로서 OnDemand 폴더.	367
원래의 엔티티로서 OnDemand 폴더.	367
주석 작성 및 수정.	367
추적	368
OS/390용 Content Manager ImagePlus에 대한 작업	370
엔티티 및 속성 나열	371
OS/390용 ImagePlus 조회 구문규칙	375
AS/400용 Content Manager에 대한 작업.	378
엔티티(색인 클래스) 및 속성 나열	378
조회 실행	379
매개변수식 조회 실행.	384
Domino.Doc에 대한 작업	385
엔티티 및 서브엔티티 나열	388
캐비넷 속성 나열	390
Domino.Doc에 조회 빌드	391
조회 구문규칙 사용	391
ES(Extended Search)에 대한 작업	393
ES 서버 나열	393
엔티티(데이터베이스) 및 속성(필드) 나열	394
GQL(Generalized Query Language) 사용	398
ES에서 DDO 항목 유형 식별.	399
ES에 PID 작성	400
ES 문서의 콘텐츠 처리	400
문서 검색	406
BLOB(2진 대형 오브젝트) 검색	406
문서와 MIME 유형 연관	408
ES에서 연합 검색 사용	408
Panagon Image Services에 대한 작업(Java 전용)	408
데이터 모델링	409
Panagon Image Services의 문서 및 페이지	410
엔티티 및 속성 나열	410
조회	412
관계형 데이터베이스에 대한 작업.	415
관계형 데이터베이스에 연결	415
엔티티 및 엔티티 속성 나열	417
조회 실행	420
사용자 조정 콘텐츠 서버 커넥터 작성	423
사용자 조정 콘텐츠 서버 커넥터 개발	423
FeServerDefBase 클래스 사용(Java 전용).	441

EIP 워크플로우 응용프로그램 빌드	443
워크플로우 서비스에 연결	443
워크플로우 시작	444
워크플로우 종료	446
모든 워크플로우 나열	447
워크플로우 일시중단	448
워크플로우 재개	448
모든 작업 목록 나열	449
작업 목록에 액세스	450
작업 항목에 액세스	451
워크플로우의 항목 이동	453
모든 워크플로우 템플릿 나열	454
초치 작성(Java 전용).	455
비주얼 및 비주얼이 아닌 JavaBeans를 사용하여 응용프로그램 빌드.	457
기본 Bean 개념 이해	457
빌더에서 JavaBeans 사용	459
IBM Websphere Studio Application Developer 사용	459
EIP Java Bean 호출.	460
비주얼이 아닌 Bean	460
비주얼이 아닌 Bean 구성	461
비주얼이 아닌 Bean 기능 이해	461
비주얼이 아닌 Bean 카테고리.	462
비주얼이 아닌 Bean 사용 시 고려사항.	467
Bean에서 추적 및 로그 기록	468
비주얼이 아닌 Bean에 대한 등록 정보 및 이벤 트 이해	468
비주얼이 아닌 Bean을 사용하여 응용프로그램 빌드	468
비주얼 Bean에 대한 작업	469
CMBLogonPanel Bean.	470
CMBSearchTemplateList Bean	472
CMBSearchTemplateViewer Bean	472
CMBSearchTemplateViewer의 필드 유효성 확 인 또는 편집	473
CMBSearchPanel Bean.	473
CMBSearchResultsViewer Bean	474
팝업 메뉴 대체.	475
CMBFolderViewer Bean	475
CMBDocumentViewer Bean	477
보기 프로그램 스펙	477
기본 보기 프로그램	478
외부 보기 프로그램 시작	478
CMBItemAttributesEditor Bean	478
CMBItemAttributesEditor의 변경 거부.	479

CMBVersionsViewer Bean	479
비주얼 Bean을 위한 일반 작동	479
비주얼 Bean 대체.	480
비주얼 Bean을 사용하여 응용프로그램 빌드	481
Java 문서 보기 프로그램 툴킷에 대한 작업	485
보기 프로그램 아키텍처	486
문서 엔진	486
주석 엔진	487
일반 문서 보기 프로그램 작성.	487
일반 문서 보기 프로그램 사용자 조정	487
응용프로그램 예제.	489
독립형 보기 프로그램.	490
Java 응용프로그램.	490
Thin 클라이언트	491
애플릿 또는 servlet	491
이중 모드 및 애플릿 또는 servlet	492
주석 서비스에 대한 작업	493
주석 서비스 인터페이스 사용	493
주석 편집 지원의 이해	494
주석 서비스를 사용하여 응용프로그램 빌드	495
응용프로그램에 사용자 조정 주석 유형 추가	496
Enterprise Information Portal 태그 라이브러리 및 제어기 servlet에 대한 작업	497
태그 라이브러리 및 servlet 설정	497
태그 라이브러리 사용	497
태그 라이브러리에 사용되는 규칙.	498
태그 요약	498
연결 관련 태그.	499
스키마 관련 태그	499
검색 관련 태그.	500
항목 관련 태그.	500
폴더 관련 태그.	502
문서 관련 태그.	502
EIP 제어기 servlet	503
servlet이 수행할 수 있는 작업	503
servlet 참조.	504
Servlet 툴킷 기능 매트릭스	509
Information Mining에 대한 작업	511
Bean을 사용하여 Information Mining 응용프로그램 빌드	511
샘플 파일의 위치	515
문서 구분	516
문서 요약	524
정보 추출	530

클러스터링	536
웹 공간에서 문서 가져오기.	541
카테고리별 문서 검색.	550
사용자의 콘텐츠 제공자 빌드	556
서비스 API 사용	557
Information Mining 서비스 API에 연결	557
라이브러리, 분류 및 카탈로그 관리	558
Information Mining 도구 사용	562
레코드 작성 및 카탈로그에 메타데이터 저장	565
문서 검색	567
서버 작업 실행.	568

JSP를 기준으로 한 Information Mining 응용프로	
그램 예제	570
JSP 설치.	571
주의사항	573
상표	575
용어집.	577
색인	587

이 책에 대하여

이 책에서는 Enterprise Information Portal 버전 8 릴리스 2(EIP) 및 Content Manager(CM) 버전 8 릴리스 2와 함께 제공되는 Java™, JavaBeans™ 및 C++ API(Application Programming Interface)의 사용 방법에 대해 설명합니다. API 및 Bean은 이기종 콘텐츠 서버에 저장된 콘텐츠에 액세스하는 응용프로그램을 작성할 수 있도록 빌딩 블록을 제공합니다.

이전 버전에서 CM과 EIP에 대해 별도의 응용프로그램 프로그래밍 안내서가 있었습니다. 버전 8 릴리스 2에서는 두 제품이 다수의 API를 공유하며 여러 가지 동일한 프로그래밍 개념을 기초로 합니다. 또한 Enterprise Information Portal 버전 8 릴리스 2의 여러 가지 새 기능에는 CM 버전 8 릴리스 2의 새 커넥터가 포함되므로 두 개의 안내서가 하나로 통합되었습니다.

이 책에는 다음이 포함되어 있습니다.

- Java 및 C++ 구문의 동적 데이터 오브젝트 개념을 포함하여 Enterprise Information Portal 및 CM 응용프로그램 프로그래밍 개념에 대한 소개
- CM 버전 8 릴리스 2 커넥터를 통해 액세스할 수 있는 기능에 대한 설명
- 콘텐츠 서버에 대한 다른 모든 Enterprise Information Portal 커넥터의 문서
- 비주얼 및 비주얼이 아닌 JavaBeans 갱신
- Information Mining, IBM® Web Crawler 및 워크플로우에 대한 프로그래밍 정보 갱신

Content Manager를 참조하는 그림은 제품의 예비 버전 8.1 및 버전 8 릴리스 2를 모두 나타냅니다.

이 책의 사용자

이 책에서는 다음의 기술 중 일부 또는 모두를 갖추고 있는 응용프로그램 프로그래머를 위해 작성되었습니다.

- C++, Java, JavaBeans 또는 HTML을 사용해 본 경험이 있습니다.
- 관계형 데이터베이스 개념에 익숙합니다.
- DDO/XDO 프로토콜에 대한 지식이 있습니다.

추가 정보

제품 패키지에는 시스템을 계획, 설치, 관리 및 사용하도록 도와주는 완전한 정보 세트가 들어 있습니다. 웹을 통해 제품 문서 및 지원을 이용할 수 있습니다.

제품 패키지에 포함된 정보

제품 패키지에는 Information Center 및 .PDF로 된 각 서적이 들어 있습니다.

Information Center

제품 패키지에는 제품을 설치할 때 설치되는 Information Center가 들어 있습니다. Information Center 설치에 대한 자세한 정보는 *Content Management 시스템 계획 및 설치*를 참조하십시오.

Information Center에는 Content Manager, Enterprise Information Portal 및 Content Manager VideoCharger에 대한 문서가 들어 있습니다. 제품 및 작업(예: 관리)별로 주제별 정보가 구성됩니다. 검색 기능은 탐색 메커니즘 및 색인을 제공할 뿐 아니라, 검색 가능성을 높입니다.

PDF 서적

운영 체제에 맞는 Adobe Acrobat Reader를 사용하여 PDF 파일을 온라인으로 볼 수 있습니다. Acrobat Reader가 설치되지 않았으면 Adobe 웹 사이트 <http://www.adobe.com>에서 다운로드할 수 있습니다.

표 1에서는 멀티플랫폼용 IBM Content Manager에 포함된 Content Manager 서적을 보여줍니다.

표 1. Content Manager 서적

파일 이름	제목	서적 번호
설치	<i>Content Management 시스템 계획 및 설치</i> ¹	GA30-1545-01
이주	<i>Content Manager 버전 8로 이주</i>	SA30-1598-01
sysadmin	<i>시스템 관리 안내서</i>	SA30-1546-01

멀티플랫폼용 IBM Content Manager 주문 시에는 멀티플랫폼용 IBM Enterprise Information Portal도 제공됩니다. 또는 멀티플랫폼용 IBM Enterprise Information Portal을 별도로 주문할 수 있습니다. 표 2에서는 제품에 포함된 Enterprise Information Portal 서적을 보여줍니다.

표 2. Enterprise Information Portal 서적

파일 이름	제목	서적 번호
apgwork	<i>워크스테이션 응용프로그램 프로그래밍 안내서</i> ¹	SA30-1551-01
ecliinst	<i>eClient 설치, 구성 및 관리</i>	SA30-1548-02
eipinst	<i>Enterprise Information Portal 계획 및 설치</i>	GA30-1549-01
eipmanag	<i>Enterprise Information Portal 관리</i>	SA30-1550-01
messcode	<i>메시지 및 코드</i> ²	SA30-1552-01

표 2. Enterprise Information Portal 서적 (계속)

파일 이름	제목	서적 번호
주:		
1. 워크스테이션 응용프로그램 프로그래밍 안내서에는 Content Manager 및 Enterprise Information Portal용 응용프로그램 프로그래밍에 대한 정보가 들어 있습니다.		
2. 메시지 및 코드에는 Content Manager 및 Enterprise Information Portal용 메시지 및 코드가 들어 있습니다.		

웹에서 이용할 수 있는 자원

웹을 통해 제품 지원을 이용할 수도 있습니다. 다음 제품 웹 사이트에서 **Support**를 누르십시오.

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

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

문서는 제품에 소프트웨어로 포함되어 있습니다. 웹에서 제품 문서에 액세스하려면 제품 웹 사이트에서 **Library**를 누르십시오.

또한 EDO(Enterprise Documentation Online)라고 하는 HTML 기반 문서 인터페이스도 웹에서 이용할 수 있습니다. 현재 여기에는 API 참조 정보가 들어 있습니다. EDO 액세스에 대한 정보를 보려면 Enterprise Information Portal 라이브러리 웹 페이지로 찾아가십시오.

의견을 보내는 방법

여러분의 의견은 IBM에게 중요한 정보가 됩니다. 본 서적이거나 기타 Content Manager 또는 Enterprise Information Portal 문서에 대한 의견을 보내주십시오. 다음과 같은 방법으로 의견을 보낼 수 있습니다.

- 웹을 통해 의견을 보내십시오. 다음 IBM Data Management 온라인 고객 의견서(RCF) 페이지를 방문하십시오.

www.ibm.com/software/data/rcf

이 페이지에서 의견을 입력하여 보낼 수 있습니다.

- 전자 우편 comments@vnet.ibm.com으로 의견을 보내주십시오. 반드시 제품 이름, 버전 번호, 책의 제목 및 부품 번호(필요한 경우에는)를 기입하십시오. 특정 텍스트를 언급할 때는 텍스트 위치(예: 장 및 섹션 제목, 표 번호, 페이지 번호 또는 도움말 주제 제목)도 기입하십시오.

Enterprise Information Portal 버전 8의 새로운 기능

버전 8.2: 버전 8.2에는 버전 8.1의 다양한 향상 기능이 포함되어 있습니다. 버전 8.2는 시스템 관리 워크플로우에 더 많은 기능을 추가하고 DB2 Universal Database 버전 8.1 등의 최신 데이터베이스 기술을 지원합니다. 버전 8.2 제품의 이러한 중요 부분 및 기타 향상 기능은 아래 요약되어 있습니다.

Enterprise Information Portal 이름을 컨텐츠용 IBM Information Integrator로 변경

Enterprise Information Portal이 컨텐츠용 Information Integrator로 이름이 바뀌었습니다. 비록 책의 이름이 버전 8.2로 변경되었지만, 책 속의 텍스트는 제품 이름 Enterprise Information Portal을 계속해서 사용합니다. 자세한 정보를 위해 웹을 검색하는 경우, 새로운 이름으로 변환이 완료될 때까지 Enterprise Information Portal 또는 EIP를 계속해서 사용할 수 있습니다.

DB2 UDB V8.1에 대한 지원

Enterprise Information Portal V8.2를 지원합니다. DB2 V8.1의 연결 집중 기능은 상하단부 응용프로그램 및 클라이언트에 대한 증가된 확장성을 제공합니다.

연합 폴더

eClient에는 이제 다중 저장소에서 단일 연합 폴더로 문서 및 원래의 폴더를 구성하는 기능이 있으므로 워크플로우에서 해당 폴더를 시작합니다. 연합 폴더는 또한 사용자가 언제든지 검색 가능한 EIP 연합 데이터베이스에서 검색 결과를 지속적으로 저장하도록 할 수 있습니다. 전체 CRUD(작성, 검색, 갱신 및 삭제) 조작은 재색인화하지 않고도 연합 폴더에 대해 사용할 수 있습니다.

고급 워크플로우 컬렉션 지점

워크플로우는 이제 AIX 및 Solaris에서 완전히 지원됩니다. 워크플로우 빌더, API, Collection Points Monitor 및 JavaBeans는 향상된 워크플로우 기능 및 사용 가능성을 제공합니다.

응용프로그램 빌드용 Microsoft Visual Studio .NET

Enterprise Information Portal 8.1 이상 API는 이제 Content Management 응용프로그램을 작성하기 위해 Microsoft Visual Studio .NET을 지원하거나 Microsoft Visual Studio .NET을 사용하여 빌드된 응용프로그램을 통합하는 데 지원됩니다.

버전 8.1: 버전 8.1은 통합 및 융통성에 대한 Legacy를 시작합니다. 이전 Content Manager 제품의 많은 중요 부분 및 개선점 중 하나는 더 많은 문서 사용자 조정을 허용하는 새 데이터 모델 구조입니다. 버전 8.1의 Content Manager 제품에 대한 변경사항은 아래 요약되어 있습니다.

Sun Solaris에 대한 지원

Solaris 시스템에 Java 커넥터, 기능, 데이터베이스를 설치할 수 있습니다.

공통 시스템 관리

단일 클라이언트 응용프로그램은 Content Manager 및 Enterprise Information Portal 관리에 대한 별도의 액세스를 제공합니다.

새 커넥터

- Content Manager 버전 8 릴리스 1의 ICM 커넥터를 사용하면 Content Manager 버전 8의 강력한 문서 저장 기능을 활용할 수 있습니다.
- 새로운 C++ Extended Search 버전 3.7 커넥터가 AIX®에서 실행됩니다.

향상된 커넥터

- 매개변수식 텍스트 검색이 연합 계층 및 직접 Extended Search 연결을 통해 지원됩니다.
- 다음을 포함하여 OnDemand 커넥터의 기능과 성능이 향상되었습니다.
 - OnDemand DDO의 구조 수정
 - 이제 비동기 검색이 지원됩니다.

IBM Web Crawler

IBM Web Crawler는 사용자가 웹과 Lotus Notes® 데이터베이스의 정보를 검색하고 요약할 수 있도록 해주는 제품입니다.

워크플로우 향상

워크플로우는 이제 AIX 및 Solaris에서 완전히 지원됩니다. 워크플로우 빌더, API 및 JavaBeans는 향상된 워크플로우 기능과 사용 가능성을 제공합니다.

Information Center

브라우저 기반 Information Center에는 Content Manager, Enterprise Information Portal 및 Content Manager VideoCharger™에 대한 문서가 들어 있습니다. 제품 및 작업(예: 관리)별로 주제별 정보가 구성됩니다. 검색 기능은 탐색 메커니즘 및 색인을 제공할 뿐 아니라, 검색 가능성을 높입니다.

내게 필요한 옵션

내게 필요한 옵션 기능은 거동이 불편하거나 시각 장애인과 같이 신체적 장애가 있는 사용자가 소프트웨어 제품을 정상적으로 사용할 수 있도록 합니다. 제품의 주요 내게 필요한 옵션 기능에는 다음과 같은 것들이 있습니다.

- 마우스 대신 키보드를 사용하여 모든 기능을 조작하는 능력
- 확장 표시 등록 정보 지원

- 비디오 및 오디오 경보 신호에 대한 옵션
- 보조 기술과의 호환성
- 운영 체제 내게 필요한 옵션 기능과의 호환성
- 내게 필요한 옵션 문서 형식

Content Manager 버전 8의 새로운 기능

버전 8.2: 버전 8.2에는 버전 8.1의 다양한 향상 기능이 포함되어 있습니다. 버전 8.2는 eClient에 더 많은 워크플로우 기능을 추가하고, 자원 관리 기능을 증가시키며, DB2 Universal Database 버전 8.1, Oracle 버전 8.1.7.4 및 버전 9.2.0.1, 그리고 WebSphere 버전 5를 포함한 최신 데이터베이스 및 클라이언트 기술을 지원합니다. 버전 8.2 제품의 이러한 중요 부분 및 기타 향상 기능은 아래 요약되어 있습니다.

Enterprise Information Portal 이름을 컨텐츠용 IBM Information Integrator로 변경

Enterprise Information Portal이 컨텐츠용 Information Integrator로 이름이 바뀌었습니다. 비록 책의 이름이 버전 8.2로 변경되었지만, 책 속의 텍스트는 제품 이름 Enterprise Information Portal을 계속해서 사용합니다. 자세한 정보를 위해 웹을 검색하는 경우, 새로운 이름으로 변환이 완료될 때까지 Enterprise Information Portal 또는 EIP를 계속해서 사용할 수 있습니다.

Oracle 버전 8.1.7.4 또는 버전 9.2.0.1 이상 지원

Content Manager V8.2는 라이브러리 서버 및 자원 관리자에 모두 저장된 메타데이터를 관리하는 Oracle 데이터베이스에 대한 지원을 추가합니다. 이 주 도구에는 Content Manager 버전 7의 Oracle 사용자가 포함되어 있습니다. 주: Oracle은 Enterprise Information Portal 데이터베이스 서버 컨텐츠를 관리하지 않습니다.

복제 Content Manager V8.2는 자원 관리자 복제를 포함합니다. 여기에는 복제 자원 관리자가 관리하는 다중 위치에 오브젝트를 저장할 수 있는 기능이 있습니다. 오브젝트 복제본은 향상된 로드 밸런싱을 위한 LAN 캐시 오브젝트로서 작동합니다.

LAN 캐시

Content Manager V8.2의 LAN 캐시 지원은 시스템 관리자가 정의한 로컬 서버를 사용하는 응용프로그램 투명 캐싱을 제공합니다.

DB2 UDB V8.1에 대한 지원

Content Manager V8.2 및 Enterprise Information Portal V8.2는 DB2/UDB V8.1을 지원합니다. DB2 V8.1의 연결 집중 기능은 상하단부 응용프로그램 및 클라이언트(예: Content Manager V8 Windows용 클라

이언트)에 대한 증가된 확장성을 제공합니다. DB2/UDB V8.1이 DB2 Universal Database TIE(Text Information Extender)와 NSE(Net Search Extender)를 대체했습니다.

WebSphere Application Server 버전 4 및 버전 5에 대한 지원

WebSphere Application Server 버전 5는 서버 전개와 데이터 액세스 및 모든 웹 브라우저에서의 관리를 소개합니다.

연합 폴더

eClient에는 이제 다중 저장소에서 단일 연합 폴더로 문서 및 원래의 폴더를 구성하는 기능이 있으므로 워크플로우에서 해당 폴더를 시작합니다. 연합 폴더는 또한 사용자가 언제든지 검색 가능한 EIP 연합 데이터베이스에서 검색 결과를 지속적으로 저장하도록 할 수 있습니다. 전체 CRUD(작성, 검색, 갱신 및 삭제) 조작은 재색인화하지 않고도 연합 폴더에 대해 사용 가능합니다.

고급 워크플로우 콜렉션 지점

워크플로우는 이제 AIX 및 Solaris에서 완전히 지원됩니다. 워크플로우 빌더, API, Collection Points Monitor 및 JavaBeans는 향상된 워크플로우 기능 및 사용 가능성을 제공합니다.

응용프로그램 빌드용 Microsoft Visual Studio .NET

Enterprise Information Portal 8.1 이상 API는 이제 Content Management 응용프로그램을 쓰기 위해 Microsoft Visual Studio .NET을 지원하거나 Microsoft Visual Studio .NET을 사용하여 빌드된 응용프로그램을 통합하는 데 지원됩니다.

버전 8.1: 버전 8.1은 통합 및 융통성에 대한 Legacy를 시작합니다. 이전 Content Manager 제품의 많은 중요 부분 및 개선점 중 하나는 더 많은 문서 사용자 조정을 허용하는 새 데이터 모델 구조입니다. 버전 8.1의 Content Manager 제품에 대한 변경사항은 아래 요약되어 있습니다.

성능 향상

라이브러리 서버 및 자원 관리자는 DB2 저장 프로시저 및 레버리지 DB2 기술을 사용하여 네트워크 트래픽을 상당히 감소시키고 성능 및 확장성을 향상시킵니다.

Sun Solaris에 대한 지원

라이브러리 서버와 자원 관리자 모두 Sun Solaris에 설치할 수 있습니다.

확장 데이터 모델

새로운 계층 구조 데이터 모델은 사용자 조정 복합 문서 관리 솔루션을 위한 기초를 제공합니다.

워크플로우 향상

통합된 문서 경로지정을 통해 순차 경로 지정, 동적, 컬렉션 지점으로 워크플로우 성능이 향상되었습니다.

통합 텍스트 검색

클라이언트 사용자는 이제 속성 기반 검색 외에도 텍스트 기반 문서 정보에 대한 전체 텍스트 검색을 수행할 수 있습니다. 이제 텍스트 검색 기능은 텍스트 검색 설정을 간소한 프로세스로 만드는 DB2 Universal Database Text Information Extender를 사용합니다.

공통 시스템 관리

단일 클라이언트 응용프로그램은 Content Manager 및 Enterprise Information Portal에 대한 개별 액세스를 제공합니다. Content Manager 내에서 관리 도메인은 라이브러리 서버의 하위 섹션에 대한 관리 액세스를 제한하는 방식을 제공합니다.

전기능 데스크탑 클라이언트 및 확장 eClient

클라이언트가 향상되어 응용프로그램을 개봉하자마자 사용자에게 신속하게 전개하거나 또는 비즈니스 계열 클라이언트 통합이 제공됩니다. Windows용 클라이언트는 통합 텍스트 검색, 문서 경로지정, 계층 구조 데이터 모델(단일 하위 구성요소 레벨로), 버전화 및 가져오기 시 색인화를 지원합니다. eClient에는 통합 텍스트 검색, EIP 고급 워크플로우, 버전 제어, 여러 값 지정 속성이 포함됩니다.

쉬운 설치

지원되는 여러 운영 체제에 걸쳐 일관된 설치가 이루어지며, Start HereCD의 Planning Assistant는 사용자 조정 설치 정보를 제공합니다. 자동 설치 및 콘솔 설치 또한 제공됩니다.

Information Center

브라우저 기반 Information Center에는 Content Manager, Enterprise Information Portal 및 Content Manager VideoCharger에 대한 문서가 들어 있습니다. 제품 및 작업(예: 관리)별로 주제별 정보가 구성됩니다. 검색 기능은 탐색 메커니즘 및 색인을 제공할 뿐 아니라, 검색 가능성을 높입니다.

내게 필요한 옵션

내게 필요한 옵션 기능은 거동이 불편하거나 시각 장애인과 같이 신체적 장애가 있는 사용자가 소프트웨어 제품을 정상적으로 사용할 수 있도록 합니다. 제품의 주요 내게 필요한 옵션 기능에는 다음과 같은 것들이 있습니다.

- 마우스 대신 키보드를 사용하여 모든 기능을 조작하는 능력
- 확장 표시 등록 정보 지원
- 비디오 및 오디오 경보 신호에 대한 옵션

- 보조 기술과의 호환성
- 운영 체제 내게 필요한 옵션 기능과의 호환성
- 내게 필요한 옵션 문서 형식

PeopleSoft 및 Siebel 통합

PeopleSoft 및 Siebel 응용프로그램의 사용자는 이 응용프로그램을 구성하여 eClient를 사용하는 다양한 콘텐츠 서버에 저장된 콘텐츠에 액세스할 수 있습니다.

Enterprise Information Portal 소개

보험 회사 및 재정 기관과 같이 종이 사용이 많은 수많은 기업에서는 대량의 비즈니스 관련 콘텐츠를 관리합니다. 대부분의 산업에는 비즈니스 정보 관리 및 액세스를 위한 기업 솔루션이 필요합니다.

콘텐츠 서버는 직원이 콘텐츠를 처리하고 작업할 수 있는 메타데이터와 함께 멀티미디어, 비즈니스 양식, 문서 및 관련 데이터를 저장하는 소프트웨어 시스템입니다. 서로 다른 콘텐츠 서버를 효과적으로 연결할 방법이 없을 때, 정보를 복사하거나 직원이 다중 검색을 수행하도록 훈련시키기 위해 시간과 돈을 낭비할 수 있습니다.

Enterprise Information Portal에서는 워크스테이션 데스크탑에서 모든 기업 자원을 가져올 수 있는 최신 기술을 제공합니다. 멀티플랫폼용 IBM Enterprise Information Portal을 사용하면 하나의 클라이언트로 서로 다른 콘텐츠 서버를 연결하여 정보 및 멀티미디어 자산의 가치를 극대화할 수 있습니다. 사용자가 멀티플랫폼용 IBM Enterprise Information Portal 클라이언트를 사용하면 연결된 모든 콘텐츠 서버에 동시에 신속하게 액세스할 수 있습니다. 또한 사용자는 웹 또는 인트라넷을 포함한 콘텐츠 서버에서 Information Mining 또는 “지능형” 검색을 수행할 수 있으며 비즈니스 프로세스 내에서 워크플로우 작업을 수행할 수 있습니다.

멀티플랫폼용 IBM Enterprise Information Portal을 사용하면 기업의 응용프로그램을 사용자 조정할 수 있습니다. 응용프로그램 프로그래머는 멀티플랫폼용 IBM Enterprise Information Portal 툴킷을 사용하여 데스크탑 및 웹 기반 응용프로그램을 모두 작성할 수 있습니다.

이 장에서는 멀티플랫폼용 IBM Enterprise Information Portal의 개요를 제공합니다. 가상의 보험 회사인 XYZ 보험에 대한 시나리오로 멀티플랫폼용 IBM Enterprise Information Portal의 기능 및 기능성에 대해 설명합니다.

고객 정보 검색

대형 화재 보험 및 생명 보험 회사인 XYZ 보험은 방대한 양의 사진, 청구, 보험 증권, 조정자의 메모, 감정서 및 기타 비즈니스 문서를 보유하고 있습니다.

XYZ는 Lotus® Domino™.Doc 파일 캐비닛의 건강 진단 및 평가 전자 양식과 함께 보험 계약자에게 보낸 모든 메모를 보관합니다. XYZ는 장기 저장 및 빠른 액세스를 위해 모든 보험 증서, 주의사항 및 송장을 Content Manager OnDemand 서버에 아카이브합니다. 보험 계약자에게서 받은 모든 청구 양식, 사진 및 서신을 AS/400®용 Content Manager 시스템 폴더에 저장합니다. 감정서는 DB2 Universal Database™(DB2® UDB) Data Warehouse Center Information Catalog Manager에

보관합니다. 고해상도 그래픽과 같은 기업 미디어 자산도 광고, 홍보 및 새로운 비즈니스 분야에서 공유할 수 있는 Content Manager 시스템에 저장합니다. 또한 회사 프로시저와 같은 정보는 회사 인트라넷에 보관합니다.

요구사항

직원이 모든 고객 정보에 액세스해야 하므로 한 서버의 콘텐츠로 청구, 고객의 호출 및 일반 보험 계약자 서비스를 처리할 수는 없습니다. 고객 서비스를 제공하려면 직원은 동시에 다양한 콘텐츠 서버에 액세스해야 합니다. XYZ 보험에는 정보 검색을 위해 콘텐츠 서버와 회사 인트라넷을 연결하는 솔루션이 필요합니다. 또한 워크플로우 처리 사용을 확장하고자 합니다.

청구 담당자부터 조정자 및 조직원에 이르기까지 많은 직원이 문서에 액세스해야 합니다. XYZ에서는 특정 항목에 대한 액세스는 제한하는 반면, 다른 항목에 대한 액세스는 제한 없이 제공해야 합니다. 또한 훈련의 필요성을 줄일 수 있도록 사용이 쉬운 인터페이스도 필요합니다.

솔루션

XYZ 보험은 포괄적인 검색 기술을 사용하여 모든 콘텐츠 서버에 연결함으로써 데이터를 검색할 수 있으므로 멀티플랫폼용 IBM Enterprise Information Portal을 소개합니다. 이제 XYZ 고객 센터 영업대표가 전화를 받으면 한번의 연합 검색으로 모든 필수 보험 계약자 정보를 검색하게 됩니다.

XYZ 보험에서는 또한 Enterprise Information Portal Information Mining 기능을 사용하여 회사의 인트라넷에서 정보를 검색합니다. 워크플로우 처리의 사용도 확장하고자 합니다.

Enterprise Information Portal 솔루션

멀티플랫폼용 IBM Enterprise Information Portal은 포괄적인 제품으로, 해당 구성요소가 엔터프라이즈 특성에 꼭 맞는 솔루션을 제공하기 위해 함께 작동합니다. 다중 아키텍처에 중점을 둔 멀티플랫폼용 IBM Enterprise Information Portal은 검색을 관리하기 위한 관리 클라이언트, 검색을 실행하기 위한 클라이언트(샘플) 및 Content Manager, OS/390®용 Content Manager ImagePlus®, Content Manager OnDemand, Lotus Domino.Doc, DB2 UDB, DB2 DataJoiner® 및 DB2 Data Warehouse Center Information Catalog Manager와 같이 서로 다른 콘텐츠 서버에 연결하기 위한 커넥터를 제공합니다. 콘텐츠 서버를 추가할 경우 추가 커넥터를 작성할 수 있습니다.

3 페이지의 그림 1에서는 멀티플랫폼용 IBM Enterprise Information Portal의 다중 아키텍처 개념을 보여줍니다.

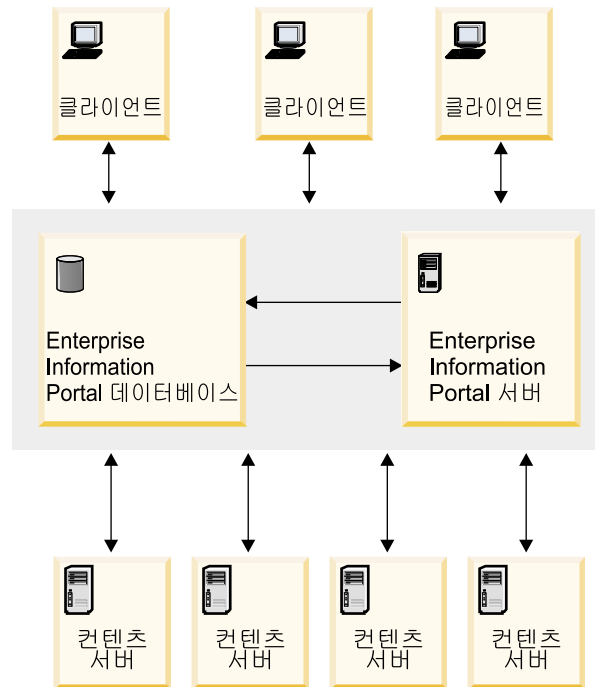


그림 1. 다중 아키텍처

클라이언트 응용프로그램은 멀티플랫폼용 IBM Enterprise Information Portal 구조를 사용하여 하나 이상의 콘텐츠 서버에서 단일 검색을 실행할 수 있습니다. 검색을 수행하기 위해 클라이언트는 멀티플랫폼용 IBM Enterprise Information Portal 시스템 관리자가 정의한 검색 템플리트를 사용합니다.

검색 템플리트를 사용하여 클라이언트는 연합 검색을 수행합니다. 연합 검색은 검색 템플릿에 사용된 연합 속성에 기본 속성이 이전에 맵핑된 여러 콘텐츠 서버에 걸쳐 동시에 실행되는 검색입니다. 멀티플랫폼용 IBM Enterprise Information Portal 검색 템플릿은 각 콘텐츠 서버의 원래 속성과 맵핑된 연합 속성을 참조하는 검색 기준을 포함합니다. 멀티플랫폼용 IBM Enterprise Information Portal 시스템 관리자는 검색 템플릿을 작성합니다. 멀티플랫폼용 IBM Enterprise Information Portal은 콘텐츠 서버의 이기종 인터페이스에 대한 공통 인터페이스로서 커넥터를 제공합니다. 그러면 콘텐츠 서버는 클라이언트에 데이터 오브젝트를 리턴합니다.

멀티플랫폼용 IBM Enterprise Information Portal 아키텍처는 다음 이점을 제공합니다.

- 단일 조회를 사용하여 e-business 트랜잭션 및 고객 서비스 응용프로그램을 지원하는 복수 및 가변 콘텐츠 서버에 대한 액세스
- 웹을 포함하여 복수 콘텐츠 서버에서의 Information Mining 성능
- 복수 및 이기종 콘텐츠 서버의 데이터에 대한 워크플로우 처리 액세스
- 클라이언트 응용프로그램, 색인 및 데이터를 분리하여 모든 콘텐츠 서버에서 데이터의 위치에 관계없이 클라이언트 응용프로그램 개발 지원

멀티플랫폼용 IBM Enterprise Information Portal 구성요소

이 절에서는 멀티플랫폼용 IBM Enterprise Information Portal의 각 구성요소에 대해 설명합니다. 이러한 구성요소는 멀티플랫폼용 IBM Enterprise Information Portal 제품의 일부로 제공됩니다.

관리 데이터베이스

멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스는 DB2 UDB 데이터베이스입니다. 이 데이터베이스는 EIP 및 해당 구성요소를 관리하는 데 필요한 모든 정보를 저장합니다.

Enterprise Information Portal 버전 7.1 데이터베이스 이주: 사용자의 Enterprise Information Portal 버전 8 릴리스 2 관리 데이터베이스를 사용하기 전에 Enterprise Information Portal 버전 7.1로부터 사용자 데이터를 이주시켜야 합니다.

관리 클라이언트

시스템 관리자는 멀티플랫폼용 IBM Enterprise Information Portal 관리 클라이언트를 사용하여 다음 작업을 수행합니다.

- 연합 검색이 가능하도록 각 콘텐츠 서버를 정의합니다.
- 콘텐츠 서버의 원래의 엔티티 및 속성을 식별하여 연합 엔티티에 매핑합니다.
- 검색 템플릿을 작성합니다.
- 검색 템플릿, Information Mining 기능 및 워크플로우 처리에 액세스할 수 있는 사용자를 식별 및 관리합니다.
- 비즈니스 워크플로우 처리를 정의합니다.

이 정보는 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스에 저장됩니다.

관리 클라이언트는 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스와 동일한 워크스테이션 또는 서버에 설치하는 것이 편의상 권장됩니다.

또한 다른 워크스테이션에 원하는 만큼의 관리 클라이언트를 가질 수도 있습니다. 이 구성을 수행하려면 다음 중 하나를 수행해야 합니다.

- DB2 Client 지원이 설치되었으며 DB2 Client Configuration Assistant를 사용하여 관리 클라이언트가 설치된 각 워크스테이션의 시스템 관리 데이터베이스에 대한 액세스를 구성합니다.
- 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스가 설치된 RMI(Remote Method Invocation) 서버를 시작하여 RMI 구성을 사용합니다.
\CMBROOT\cmbclient.ini 파일이 이 서버를 가리키는지 확인해야 합니다. 이 INI 파일의 위치는 cmbcmenv.properties 파일에서 CMCOMMON 키워드가 디렉토리를 지정한 위치입니다. 이 파일의 위치는 cmbcmenv.properties 파일에서 CMCOMMON_URL 키워드가 지정한 URL이 될 수도 있습니다.

커넥터

커넥터 클래스는 클라이언트 응용프로그램이 콘텐츠 서버와 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스에 액세스할 수 있도록 합니다. 멀티플랫폼용 IBM Enterprise Information Portal은 다음 커넥터를 포함합니다.

- 관계형 데이터베이스 커넥터(DB2, JDBC, ODBC)
- 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스에 대한 연합 커넥터
- Content Manager 버전 8 릴리스 2
- Content Manager 버전 7 릴리스 1 커넥터
- Content Manager OnDemand 커넥터
- OS/390용 Content Manager ImagePlus 커넥터
- AS/400용 Content Manager 커넥터
- Lotus Domino.Doc 커넥터
- Extended Search 커넥터

연합 커넥터는 멀티플랫폼용 IBM Enterprise Information Portal 데이터베이스에 대한 커넥터 클래스를 포함합니다. 각 콘텐츠 서버 커넥터에는 해당 커넥터 클래스가 들어 있습니다.

Java 환경에서 커넥터의 Java 버전은 로컬 또는 원격입니다. C++에는 로컬 커넥터 밖에 없습니다. 로컬 커넥터는 다양한 콘텐츠 서버에 직접 연결하는 데 사용하는 커넥터 클래스 세트입니다. 로컬 커넥터는 멀티플랫폼용 IBM Enterprise Information Portal 데스크탑 클라이언트나 멀티플랫폼용 IBM Enterprise Information Portal RMI 서버에 상주합니다. 원격 커넥터는 RMI 서버 또는 RMI 서버 풀 구성원을 통해 콘텐츠 서버에 연결하는 데 사용됩니다. 원격 커넥터를 사용하여 콘텐츠 서버에 직접 연결할 필요성을 제거하십시오.

IBM eClient

eClient는 OS/390용 Content Manager(모든 플랫폼), Content Manager OnDemand(모든 플랫폼) 및 Content ManagerImagePlus에 저장된 문서에 액세스하기 위한 브라우저 기반 사용자 인터페이스입니다.

Information Mining

Information Mining은 언어 서비스를 제공하여 콘텐츠 서버의 텍스트 문서에 숨겨진 정보를 찾습니다. 텍스트 문서 처리 중에 요약, 구분 및 검색할 수 있는 메타데이터(데이터 정보)가 작성됩니다. 멀티플랫폼용 IBM Enterprise Information Portal은 thin 클라이언트에서 Information Mining 기능을 사용하는 방법을 보여주는 샘플을 제공합니다. 데스크탑 클라이언트나 thin 클라이언트를 빌드하여 Information Mining 작업을 수행할 수 있습니다.

IBM Web Crawler

IBM Web Crawler를 사용하여 웹에서 콘텐츠를 검색하고 가져올 수 있습니다. 그런 다음 Information Mining 도구를 사용하여 검색 결과 및 메타데이터를 분석 및 구분할 수 있습니다. IBM Web Crawler는 http, ftp, ntp, lotus 및 domino 서버를 검색할 수 있습니다.

워크플로우

Enterprise Information Portal로 비즈니스에서 작업 흐름을 제어할 수 있습니다. Enterprise Information Portal 워크플로우 기능을 사용하여 작업 그룹, 부서 또는 엔터프라이즈의 워크플로우 처리를 정의 및 실행할 수 있습니다. 그래픽 빌더를 사용하여 Enterprise Information Portal 워크플로우 빌더에서의 워크플로우 처리를 포괄적이고 이해하기 쉬운 그래픽으로 표현할 수 있습니다. 그런 다음 사용자는 직접 개발한 클라이언트나 Enterprise Information Portal의 thin 클라이언트 샘플을 통해 정의된 워크플로우 처리를 사용하여 작업을 수행할 수 있습니다.

Content Manager 버전 7 텍스트 검색 엔진

이 기능을 사용하여 Content Manager 버전 7에 저장된 문서를 자동으로 색인화하고 검색할 수 있습니다. 단어나 구문을 검색하여 문서를 찾을 수 있습니다.

제한사항: 텍스트 검색 서버 및 클라이언트는 예비 버전 8.1 Content Manager 서버로만 구성하고 실행할 수 있는 선택적인 Content Manager 기능입니다. 예비 버전 8.1 Content Manager 서버를 사용하지 않을 경우 이 기능을 설치하지 마십시오.

Content Manager 버전 7 이미지 검색 서버 및 클라이언트

이 기능은 색 및 텍스처와 같은 특정 비주얼 등록 정보로 오브젝트를 검색할 수 있는 IBM QBIC[®](Query By Image Content) 기술을 사용합니다.

제한사항: 이미지 검색 서버의 기능은 Enterprise Information Portal 버전 8 및 Content Manager 버전 8을 둘 다 지원하지 않습니다. 그 기능은 예비 버전 8.1 Content Manager를 사용하여 이용할 수 있습니다.

버전 8.2 API의 새로운 기능

광범위한 코드 샘플

코드 샘플은 Content Manager 버전 8.2 기능에 대해 갱신되었습니다. 또한 모든 코드 샘플은 더 나은 가독성에 대해 내부 상자를 표시합니다(언어로 레이블 됨).

DB2 DataJoiner 제한사항

콘텐츠 서버에 DB2 Universal Database 버전 8.1이 필요한 경우, EIP DataJoiner 커넥터는 바인딩 문제로 인해 작동하지 않습니다. 이는 DataJoiner 2.1이 DB2 버전 7 파인드 파일 내의 SQL문을 인식하지 않기 때문입니다.

이 문제점을 해결하려면 DataJoiner 웹 사이트의 FAQ 질문을 읽으십시오 (<http://www.software.ibm.com/data/datajoiner/>). "DataJoiner 서버 연결 시 DB2 Universal Database 버전 7의 바인드 관련 문제점이 왜 발생합니까?" 거기에 대한 대답은 DataJoiner 버전 2.1.1 서버에 연결하는 다음 단계를 지시합니다.

1. 두 개의 바인드 파일을 다운로드한 후 db2cliws_dj.bnd 및 db2clprj_dj.bnd로 이름을 바꾸십시오.
2. 다음 DB2 명령을 입력하십시오.

```
db2cmd
db2 connect to user using
db2 bind db2cliws_dj.bnd grant public
db2 bind db2clprj_dj.bnd grant public
```

DB2 Data Warehouse Manager Information Catalog Manager 제한사항

IC 커넥터에는 DB2 Universal Database 버전 7.2이 필요하며 DB2 UDB 버전 8.1로 작업하지 않습니다.

연합 폴더(Java 전용)

Enterprise Information Portal 버전 8 릴리스 2는 이제 연합 폴더를 보유할 수 있는 특수 연합 엔티티를 제공합니다. 이 연합 폴더는 연합 조회에서 결합된 결과(예: Content Manager의 문서 및 OnDemand의 관련 문서)를 저장할 수 있습니다. 그런 다음 워크플로우로 직접 결과를 전송할 수 있습니다.

Microsoft Visual Studio .NET 지원

Enterprise Information Portal 및 Content Manager 버전 8.2 (이상) API는 이제 Microsoft Visual Studio .NET을 지원합니다.

버전 8.1 API의 새로운 기능

Enterprise Information Portal 버전 8 릴리스 2는 서로 다른 콘텐츠 서버에 비해 완전히 새로운 액세스를 제공합니다. 새 기능과 구성요소는 다음을 포함합니다.

- XML 가져오기 기능

이제 XML을 사용하여(Java API를 사용하여) DDO 및 XDO를 통해 Content Manager로 콘텐츠를 가져오고 내보낼 수 있습니다.

- 개선된 설치 프로시저

- 관계형 데이터베이스를 위한 추가 커넥터

Enterprise Information Portal은 JDBC 또는 ODBC 드라이버를 통해 DB2 UDB, DB2 DataJoiner, DB2 Data Warehouse Manager Information Catalog Manager 및 기타 데이터베이스에 관계형 데이터베이스 커넥터를 제공합니다.

- 고급 Information Mining 및 검색 기능

Information Mining은 특정 카테고리의 문서로 제한할 수 있는 융통성 있는 조회를 사용하여 고급 텍스트 검색을 제공합니다.

- 워크플로우 기능
Enterprise Information Portal 워크플로우 기능을 사용하여 작업 그룹, 부서 또는 엔터프라이즈의 워크플로우 처리를 정의 및 실행할 수 있습니다.
- 연합 레벨 액세스 제어
사용 권한 세트 및 액세스 제어 목록을 사용하여 Enterprise Information Portal Information Mining과 워크플로우 처리에 대한 액세스를 제어할 수 있습니다. 각 콘텐츠 서버의 액세스 제어 기능을 통해 데이터에 대한 추가 액세스 제어를 관리할 수 있습니다.
- Content Manager에 대한 추가 지원
 - 콘텐츠 클래스의 나열, 추가, 검색, 갱신 및 삭제
 - 오브젝트 콘텐츠의 비동기 검색

신규 및 변경된 Java 클래스

Java 공통 클래스는 모든 커넥터에서 사용됩니다. 이 패키지는 커넥터에서 사용되는 구체적인 클래스(예: DKDDO)뿐만 아니라 인터페이스(예: dkDatastore) 및 추상 클래스(예: dkAbstractDatastore 및 dkXDO)도 포함합니다.

신규 클래스

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

- dkPersistentCheckableObject
- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

변경된 클래스

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

Enterprise Information Portal 버전 8에 대한 클래스 작동 변경사항

EIP 7.1과 EIP 8.2 사이에 일부 클래스 또는 클래스 구성요소의 작동이 변경되었습니다. 이 절에서는 이러한 변경사항에 대해 설명합니다. 자세한 정보는 온라인 *API* 참조서를 참조하십시오.

- dkIterator: next()는 반복자를 다음 항목으로 확장하고 해당 항목을 가져오며, previous()는 이전 항목으로 이동한 후 해당 항목을 가져옵니다.
- dkXDO: getPidObject() 및 setPidObject(DKPid pid)

신규 및 변경된 C++ 클래스

C++ 공통 클래스 패키지는 모든 커넥터에서 사용됩니다. 이 패키지는 커넥터에서 사용되는 구체적인 클래스(예: DKDDO)뿐만 아니라 인터페이스(예: dkDatastore) 및 추상 클래스(예: dkAbstractDatastore 및 dkXDO)도 포함합니다.

신규 클래스

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

변경된 클래스

- dkAbstractDatastore
- dkAbstractDatastoreDef
- dkAbstractDatastoreExt
- dkAbstractEntityDef

- dkAccessControlList
- dkDataObjectBase
- dkDatastore
- dkDatastoreAdmin
- dkDatastoreDef
- dkDatastoreExt
- DKDDO
- dkEntityDef
- dkPersistentObject
- dkPrivilegeSet
- dkSchemaMapping
- dkSearchTemplate
- dkUserManagement

Enterprise Information Portal 8.2에 대한 클래스 작동 변경사항

EIP 7.1과 EIP 8.2 사이에 일부 클래스 또는 클래스 구성요소의 작동이 변경되었습니다. 이 절에서는 이러한 변경사항에 대해 설명합니다. 자세한 정보는 온라인 API 참조서를 참조하십시오.

- AIX에서 DKAny와 함께 DKString 사용이 변경되었습니다. DKAny로부터 DKString을 검색하려면 DKAny를 DKAny 변수로 가져온 후 이를 DKString 변수로 할당해야 합니다.

예제: 이 예제에서는 DDO가 콘텐츠 서버의 dkResultSetCursor fetchNext() API 호출에서 검색되고 아래에 표시된 것처럼 DDO 변수가 설정되었다고 가정합니다. 이 예제에서는 DDO의 데이터 항목을 통해 루프되어 문자열 값을 찾습니다.

```
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
    }
}
```

- getPidObject 및 setPidObject 메소드에 대해 변경된 dkXDO를 구현하는 인터페이스.

Enterprise Information Portal 응용프로그램 프로그래밍 개념

Enterprise Information Portal은 멀티미디어 데이터뿐만 아니라 관계형 데이터에 액세스하고 표시하는 조회 응용프로그램 작성에 사용할 수 있는 객체 지향(OO) API(Application Programming Interface)를 제공합니다. 이 장에서는 이러한 API를 Enterprise Information Portal 아키텍처에 최적화시키는 방법에 대한 간략한 개요를 제공하며, API를 기반으로 하는 객체 지향 프로그래밍 개념에 대해 설명합니다.

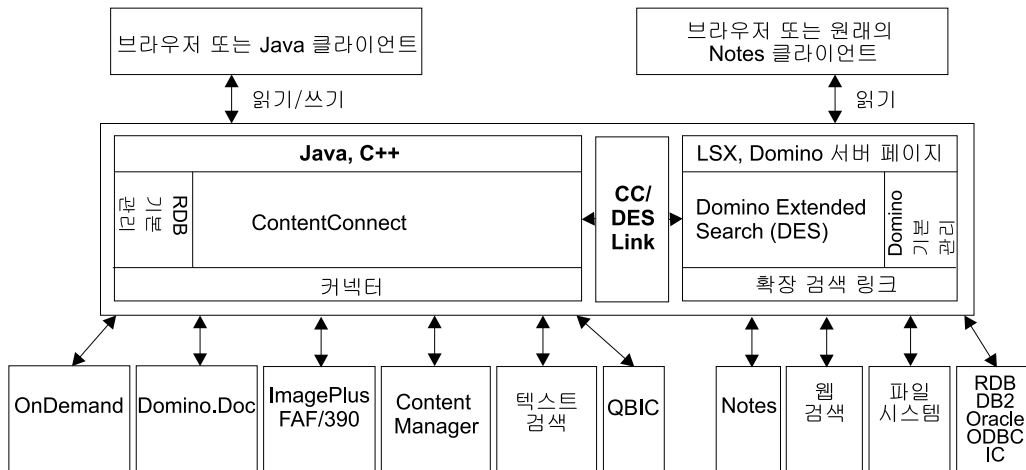


그림 2. Enterprise Information Portal 조직

컨텐츠 서버를 통한 데이터 액세스 이해

컨텐츠 서버는 DDO/XDO 프로토콜과 호환 가능한 데이터 저장소입니다. 컨텐츠 서버는 사용자 세션, 연결, 트랜잭션, 커서 및 조회를 지원합니다. 이 책에서 설명하는 클래스 라이브러리와 API(Application Programming Interface)를 사용하는 응용프로그램은 DDO 추가, 검색, 갱신 및 삭제와 같이 컨텐츠 서버에서 지원하는 기능을 수행할 수 있습니다. Enterprise Information Portal은 다음 컨텐츠 서버를 지원합니다.

- Content Manager 버전 8 릴리스 2
- Content Manager 버전 7 릴리스 1
- Domino.Doc
- Extended Search
- OS/390용 ImagePlus
- Content Manager OnDemand
- AS/400용 VisualInfo™

- DB2
- DB2 DataJoiner
- Information Catalog
- DB2 Warehouse Manager 정보 카탈로그 관리자
- JDBC/ODBC 서버

Enterprise Information Portal을 사용하는 응용프로그램은 공통 서버로 작동하는 연합 콘텐츠 서버를 작성할 수 있습니다. Enterprise Information Portal 연합 클래스를 사용하면 여러 콘텐츠 서버에서 연합 검색, 검색 및 갱신이 가능합니다.

Enterprise Information Portal 연합 콘텐츠 서버 및 각 콘텐츠 서버는 서로 다른 스키마를 가지고 있습니다. 여러 이기종 콘텐츠 서버를 연합 시스템으로 통합하려면 변환 및 맵핑이 필요합니다.

스키마 맵핑 기능은 각 콘텐츠 서버에 스키마 정보를 제공합니다. 스키마 맵핑에서 제공하는 정보는 연합 검색, 연합 콜렉션 및 Enterprise Information Portal 시스템 관리 수행 중에 사용됩니다. Enterprise Information Portal은 해당 관리 데이터베이스의 기타 관리 정보뿐만 아니라 스키마 및 맵핑도 보관합니다.

동적 데이터 오브젝트 개념 이해

OMG(Object Management Group)의 CORBA 지속 오브젝트 서비스 및 오브젝트 조회 서비스 스펙에 따라 Enterprise Information Portal은 동적 데이터 오브젝트(DDO) 및 DDO의 확장판인 확장 데이터 오브젝트(XDO) 구현을 제공합니다. XDO는 CORBA POS(Persistent Data Service) 프로토콜의 일부입니다. DDO 및 XDO의 개념은 하나의 콘텐츠 서버에 고유하지 않으며 Enterprise Information Portal에서 지원하는 모든 데이터베이스 관리 시스템의 데이터 오브젝트 표현에 사용될 수 있습니다.

동적 데이터 오브젝트는 콘텐츠 서버의 내부와 외부로 데이터를 이동하는 데 사용되는 인터페이스입니다. DDO는 응용프로그램 내에 존재하며 응용프로그램 종료 후에는 존재하지 않습니다.

동적 데이터 오브젝트(DDO)

DDO는 오브젝트의 지속 데이터의 콘텐츠 서버 중립 표현입니다. DDO의 목적은 단일 지속 오브젝트의 모든 데이터를 포함하는 것입니다. 또한 콘텐츠 서버에서 지속 데이터를 검색하거나 콘텐츠 서버로 지속 데이터를 로드하기 위한 인터페이스입니다.

DDO는 단일 지속 ID(PID), 오브젝트 유형 및 데이터 항목 세트를 갖고 있으며 DDO의 기본 행수를 데이터 계수라고 합니다. 각 데이터 항목은 이름, 값, ID, 하나 이상의 데이터 등록 정보 및 데이터 등록 정보 계수를 가질 수 있습니다. 각 데이터 등록 정보는 ID, 이름 및 값을 가질 수 있습니다.

예를 들어, DDO는 해당 열이 DDO의 데이터 항목과 등록 정보로 표시되는 데이터베이스 테이블의 행을 나타낼 수 있습니다. DDO는 비전형적인 데이터 유형을 나타내는 확장 데이터 오브젝트(XDO)를 하나 이상 포함할 수 있습니다. 그림 3에서는 동적 데이터 오브젝트 및 데이터 항목을 보여줍니다.

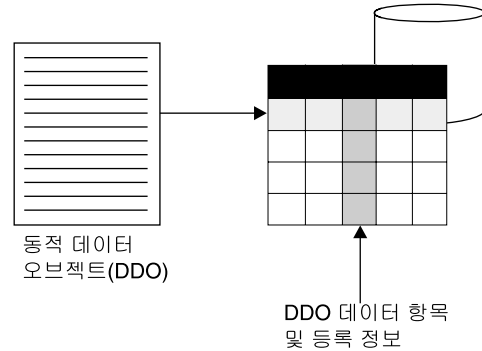


그림 3. 동적 데이터 오브젝트 및 항목

확장 데이터 오브젝트(XDO)

XDO는 Content Manager의 이미지 또는 문서를 저장하는 자원 항목 또는 관계형 데이터베이스의 오브젝트 관련 기능(예: IBM DB2 Extender)에 도입된 새 데이터 유형과 같은 복합 멀티미디어 데이터를 나타냅니다.

XDO는 복합 유형의 멀티미디어 데이터를 저장하고 응용프로그램에 이러한 데이터 유형의 작동을 구현하는 기능을 제공함으로써 DDO를 보완합니다. XDO는 복합 멀티미디어 데이터 오브젝트를 나타낼 수 있도록 DDO에 포함되거나 소유될 수 있습니다.

XDO는 등록 정보 세트를 가지고 있어 데이터 유형 및 ID와 같은 정보를 나타냅니다. 또한 XDO는 독립형 동적 오브젝트일 수도 있습니다. 그림 4에서는 XDO 예제를 보여줍니다.

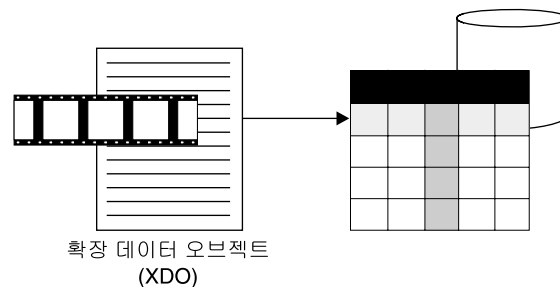


그림 4. 확장 데이터 오브젝트(XDO)

멀티미디어 콘텐츠 표현

DDO와 XDO는 모든 유형 및 구조의 데이터 오브젝트를 나타낼 수 있습니다. 예를 들어, 영화를 DDO로 나타낼 수 있습니다. 이 DDO는 Director_Name 또는 Movie_Title 과 같이 영화 속성을 나타내는 복수의 데이터 항목을 포함하며, XDO는 비디오 클립 또는 정지 이미지와 같이 영화의 멀티미디어 데이터를 나타내는 복수의 데이터 항목을 포함합니다.

Content Manager 8.2에서 DDO는 오브젝트(예: 문서 또는 이미지)를 설명하는 모든 메타데이터로 구성되어 있습니다. XDO는 DDO 기능을 확장하여 자원 콘텐츠를 지원합니다. 자원 콘텐츠는 2진 데이터 또는 텍스트에서 비디오 및 오디오 스트림까지 망라하는 모든 콘텐츠 유형입니다. XDO를 구현하는(그리고 구현해야 하는) 항목은 또한 DDO이며 XDO가 제공하는(그리고 제공해야 하는) 추가 기능에 덧붙여 DDO가 제공하는 모든 기능을 지원합니다.

콘텐츠 서버 및 DDO 이해

DDO를 작성하면 콘텐츠 서버와 동적으로 연결됩니다. DDO와 콘텐츠 서버 사이의 연결은 DDO PID로 설정됩니다.

일반적으로 Enterprise Information Portal 응용프로그램은 아래에 나열된 다섯 가지 단계를 거쳐 콘텐츠 서버의 내부와 외부로 데이터를 이동합니다.

1. 콘텐츠 서버를 작성합니다.
2. 콘텐츠 서버로 연결을 설정합니다.
3. 작업할 DDO를 작성한 다음 콘텐츠 서버와 연결합니다.
4. 해당 메소드를 사용하여 DDO를 추가, 검색, 갱신 및 삭제합니다.
5. 연결을 닫고 콘텐츠 서버를 제거합니다.

DDO/XDO와 속성값 및 항목 부분 비교

DDO는 Enterprise Information Portal의 항목에 해당합니다. DDO의 오브젝트 유형은 항목의 연관된 항목 유형에 해당합니다. DDO의 데이터 항목은 항목 속성에 해당합니다. 예를 들어, Content Manager에서 항목 유형은 속성 세트를 사용하여 작성되고 항목은 항상 항목 유형별로 색인화됩니다.

DDO는 Enterprise Information Portal의 항목 부분에 해당하는 XDO를 하나 이상 보유할 수 있습니다.

지속 식별자(PID) 이해

지속 식별자(PID)는 모든 콘텐츠 서버의 지속 오브젝트를 고유하게 식별합니다. DDO의 PID는 항목 ID, 콘텐츠 서버 이름 및 기타 관련 정보로 구성되어 있습니다. DDO가 콘텐츠 서버에 추가되는 경우, 시스템은 고유 PID를 DDO로 할당합니다.

DDO는 콘텐츠 서버의 내부 또는 외부로 이동된 지속 데이터에 대한 동적 인터페이스 이므로 다른 DDO도 동일한 지속 데이터 엔티티를 나타낼 수 있으며, 따라서 DDO는 동일한 PID를 가질 수 있습니다. 예를 들어, 데이터 엔티티를 콘텐츠 서버로 이동하여 데이터를 지속적으로 저장하기 위한 DDO를 하나 작성한 다음, 수정용으로 동일한 콘텐츠 서버에서 체크아웃한 동일한 데이터 엔티티를 보유할 다른 DDO를 작성할 수 있습니다. 이 경우 두 개의 DDO는 동일한 PID 값을 공유합니다.

연합 콘텐츠 서버 및 연합 검색에 대한 작업

연합 검색은 하나 이상의 콘텐츠 서버에서 데이터를 검색하는 프로세스입니다. 연합 검색에는 DKDatastoreFed 오브젝트를 사용합니다. 연합 검색은 dkDatastore, dkDatastoreDef 및 연합 검색을 지원하는 기타 관련 클래스의 특정 구현인 클래스와 함께 작동합니다. 특정 연합 클래스는 조회, 컬렉션 및 데이터 오브젝트에 대한 클래스와 같은 공통 클래스와 함께 작동하며 Enterprise Information Portal 프레임워크의 일부입니다.

연합 클래스는 OS/390용 Content Manager ImagePlus 또는 Domino.Doc과 같은 여러 콘텐츠 서버 사이에서 작동합니다. 이 클래스는 연합 검색을 위한 일반 기능 세트와 콘텐츠 서버에 대한 액세스를 제공합니다. 연합 문서 모델이라고 하는 이러한 공통 보기가 그림 5에 나와 있습니다.

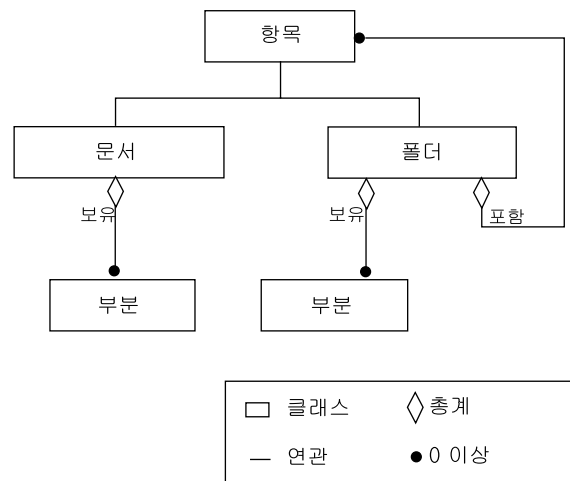


그림 5. 연합 문서 보기

항목은 문서 또는 폴더일 수 있습니다. 문서는 0개 이상의 부분을 포함할 수 있습니다. 폴더는 문서나 다른 폴더일 수 있는 0개 이상의 항목을 가질 수 있습니다.

모든 콘텐츠 서버가 연합 문서 모델을 지원할 수 있는 것은 아닙니다. 예를 들어, DB2 데이터베이스는 폴더 또는 부분을 갖고 있지 않습니다. 항목은 DB2 또는 기타 관계형 데이터베이스 테이블의 행으로 매핑하고, 콘텐츠 서버가 문서 또는 폴더를 지원하지 않는 경우 사용됩니다.

일반적으로 문서는 프로그램에서 동적 데이터 오브젝트(DDO)로 나타납니다. DDO는 데이터를 콘텐츠 서버의 내부 또는 외부로 전송하기 위한 자체 설명 데이터 오브젝트입니다. DDO 자체는 일반 구조를 가지고 있으며 다양한 모델을 지원합니다. 연합 문서

모델로 제한되지 않습니다. 이러한 융통성을 통해 DDO는 각각 자체 데이터 모델을 갖는 다른 콘텐츠 서버의 데이터를 나타낼 수 있습니다.

엔티티는 속성으로 구성된 콘텐츠 서버 오브젝트입니다. 속성은 콘텐츠 서버에서 메타데이터로 사용되는 레이블입니다. 예를 들어, 프로파일, 필드 또는 키워드는 Domino.Doc 콘텐츠 서버의 속성입니다.

각 콘텐츠 서버에는 지원하는 모델을 설명하는 자체 용어가 있습니다. 표 3은 여러 콘텐츠 서버에 사용되는 용어를 연합 모델에 관련시킨 것입니다.

표 3. 각 콘텐츠 서버에 대한 용어 매핑

콘텐츠 서버	데이터 원본	엔티티	속성	보기
Content Manager 8.2	라이브러리 서버	항목 유형	속성	항목 유형 보기 또는 항목 유형 서브세트
Content Manager 7.1	라이브러리 서버	색인 클래스	<ul style="list-style-type: none"> 속성 키 속성 	색인 클래스 보기
OnDemand	OnDemand 서버	<ul style="list-style-type: none"> 응용프로그램 그룹 폴더 	<ul style="list-style-type: none"> 필드 기준 	적용되지 않음
ImagePlus	OS/390용 ImagePlus 서버	엔티티	속성	적용되지 않음
AS/400용 Content Manager	AS/400용 Content Manager 서버	색인 클래스	속성	색인 클래스 보기
Domino.Doc	Domino 서버	라이브러리 방 캐비닛 바인더	프로파일 필드 키워드	적용되지 않음
Extended Search	Extended Search 서버	데이터베이스 이름	데이터베이스 이름	적용되지 않음
관계형 데이터베이스	IBM DB2 UDB, JDBC, ODBC, IBM DB2 DataJoiner	테이블	열	보기
Information Catalog	DB2 Warehouse Manager 정보 카탈로그 관리자	색인 클래스	등록 정보	
연합 콘텐츠 서버	매핑 서버	매핑된 연합 엔티티	매핑된 연합 속성	검색 템플릿
연합 폴더를 보유할 수 있는 연합 콘텐츠 서버	서버	연합 엔티티	연합 속성	연합 폴더

21 페이지의 그림 6에서는 연합 검색을 보여줍니다. 연합 검색은 검색 템플릿을 통해 작업하며 Enterprise Information Portal 연합 콘텐츠 서버를 사용합니다. 그러면

연합 콘텐츠 서버는 콘텐츠 서버에서 실제 검색을 수행하기 위해 개별 콘텐츠 서버에 대한 검색을 호출합니다. 이 연결은 스키마 매핑에 의해 설정됩니다.

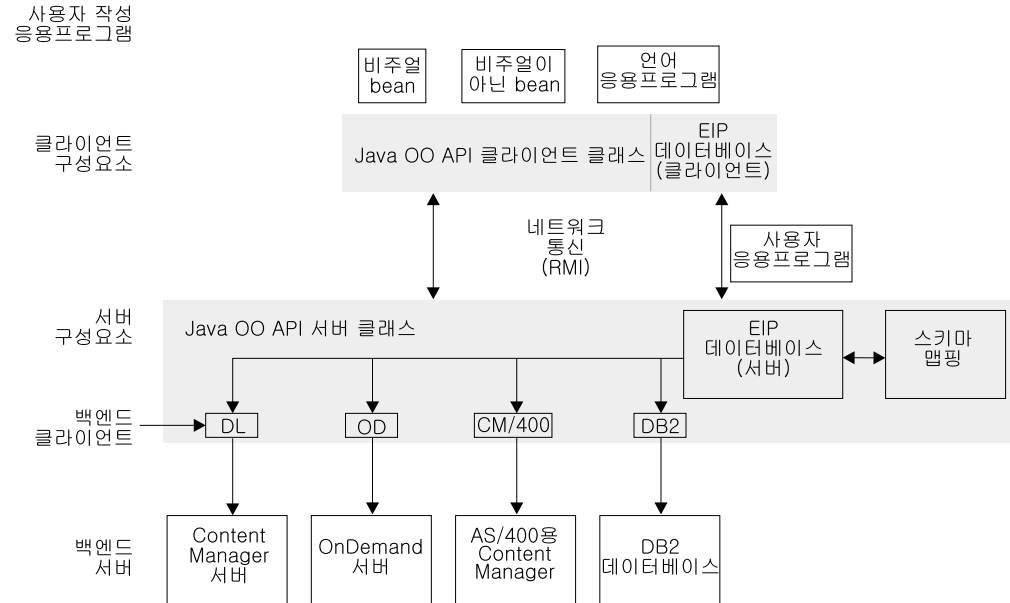


그림 6. 연합 검색의 구조

연합 콘텐츠 서버는 로컬 또는 원격 커넥터를 사용하여 콘텐츠 서버에 연결할 수 있고 이 통신에 대해 RMI를 사용할 수 있습니다. API 클래스의 맨 위에서 응용프로그램을 개발할 수도 있습니다.

연합 스키마 매핑

스키마 매핑은 콘텐츠 서버의 스키마와, 사용자가 응용프로그램에서 처리하려는 항목 구조 사이의 매핑을 나타냅니다. 연합 스키마는 Enterprise Information Portal 연합 콘텐츠 서버의 개념적 스키마이며 연합 콘텐츠 서버의 개념과 각 참여 콘텐츠 서버의 개념 사이에 매핑을 정의합니다. 스키마 매핑은 데이터가 실제로 저장되는 방법과 사용자가 응용프로그램에서 데이터를 처리하려는 방법 간의 차이점을 처리합니다.

매핑 정보는 스키마 매핑 클래스의 메모리에 나타납니다.

연합 콘텐츠 서버 매핑 구성요소 사용

연합 콘텐츠 서버는 엔티티 및 속성을 매핑하기 위한 스키마 매핑 정보 이외에 다음 정보에 대한 액세스도 갖고 있어야 합니다.

사용자 ID 및 암호 매핑

단일 로그인 기능을 지원하려면 Enterprise Information Portal의 각 사용자 ID를 각 콘텐츠 서버의 해당 사용자 ID에 매핑할 수 있어야 합니다.

컨텐츠 서버 등록

Enterprise Information Portal에서 찾아 로그인할 수 있도록 각 컨텐츠 서버를 등록해야 합니다.

사용자 ID 및 컨텐츠 서버 정보는 Enterprise Information Portal 관리 데이터베이스에서 유지보수됩니다.

연합 조회 실행

API를 사용하여 연합 검색을 실행하려면 먼저 연합 조회 문자열을 작성하여 시작하십시오. 그런 다음 여러 가지 방법으로 조회를 작성 및 실행할 수 있습니다.

- 연합 조회 오브젝트 DKFederatedQuery를 작성하고 이 오브젝트에 조회 문자열을 전달한 다음 오브젝트에 대한 실행 또는 평가 메소드를 호출하여 조회를 처리할 수 있습니다.
- 조회 문자열을 연합 컨텐츠 서버의 실행 또는 평가 메소드로 전달하여 조회를 직접 처리할 수도 있습니다.

조회 문자열은 조회의 컨텐츠 서버 중립 표현인 연합 조회 양식으로 구문 분석될 수 있습니다. 연합 조회 양식은 연합 검색에 대한 입력입니다.

조회가 GUI(Graphical User Interface) 기반 응용프로그램에 들어 있으면 조회의 구문 분석이 필요하지 않으며 해당 연합 조회 양식은 직접 구성할 수 있습니다.

연합 검색이 처리됨에 따라 Enterprise Information Portal은 다음 단계를 수행합니다.

- 조회 표준 양식을 각 컨텐츠 서버에서 실행하는 여러 개의 원래의 조회로 변환합니다. 변환 정보는 스키마 맵핑에서 가져옵니다.
- 연합 엔티티 및 속성을 각 컨텐츠 서버에 대한 원래의 엔티티 및 속성으로 변환합니다. 이 프로세스에서는 스키마 맵핑에서 설명한 맵핑 및 변환 메커니즘을 사용합니다.
- 원래의 조회 구성 중 관련 데이터만 필터링합니다.
- 원래의 조회를 형성한 다음 각 컨텐츠 서버에 제출합니다.

각 컨텐츠 서버는 제출된 조회를 실행합니다. 결과는 연합 조회로 리턴되며 다음과 같이 처리할 수 있습니다.

- 맵핑 정보에 따라 원래의 엔티티 및 속성을 연합 엔티티 및 속성으로 변환합니다.
- 결과를 필터링하여 요청된 데이터만 포함합니다.
- 여러 컨텐츠 서버의 결과를 연합 컬렉션에 병합합니다.

연합 검색의 결과는 연합 컬렉션으로 리턴됩니다. 반복자를 작성하여 각 컬렉션 구성원에 액세스할 수 있습니다. 반복자의 다음 메소드에 대한 각 호출은 컨텐츠 서버 중립 동적 데이터 오브젝트인 DADDO 오브젝트를 리턴합니다.

연합 컬렉션은 콘텐츠 서버에 따라 조회 결과를 분리할 수 있는 기능을 제공합니다. 연합 컬렉션에서 createMemberIterator 메소드를 호출하여 순차 반복자를 작성합니다. 이 순차 반복자를 사용하여 DKResults 오브젝트인 각 구성원 컬렉션에 액세스하여 별도로 이를 처리할 수 있습니다.

연합 검색의 구성요소 및 관계가 그림 7에 나와 있습니다.

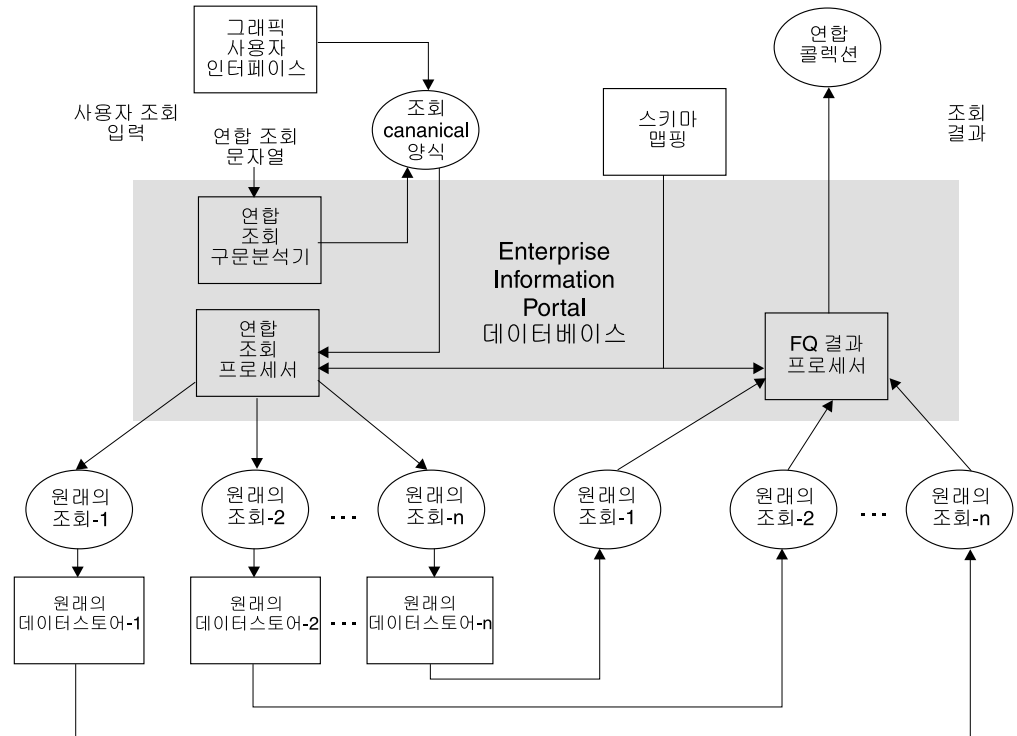


그림 7. 연합 조회 처리

연합 조회 구문규칙

연합 조회를 작성하는 경우 아래와 같이 알맞은 구문규칙을 사용해야 합니다. 연합 콘텐츠 서버는 이미지 조회를 지원하지 않습니다.

```

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]
    )]
]

```

NOT 연산자는 연합 검색에는 지원되지 않습니다.

연합 조회 문자열의 예제

LIKE 연산자를 사용하는 연합 매개변수식 조회

```

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

```

LIKE 및 > 연산자를 사용하는 연합 매개변수식 조회

```

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

```

LIKE 및 < 연산자를 사용하는 연합 매개변수식 조회

```

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

```

BETWEEN 연산자를 사용하는 연합 매개변수식 조회

```

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

```

MAX_RESULTS가 0으로 설정되면 모든 결과를 리턴합니다.

NOTBETWEEN 연산자를 사용하는 연합 매개변수식 조회

```

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

```

IN 연산자를 사용하는 연합 매개변수식 조회

```

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

```

NOTIN 연산자를 사용하는 연합 매개변수식 조회

```

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

```

== 연산자를 사용하는 연합 매개변수식 조회

```

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

```

<> 연산자를 사용하는 연합 매개변수식 조회

```

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

```

AND 및 **OR** 연산자를 사용하는 연합 매개변수식 조회

```
"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 연산자를 사용하는 연합 매개변수식 조회

이 예제에서는 콘텐츠의 텍스트를 검색합니다. 콘텐츠는 단어 또는 구문이 가능합니다. 이 조회는 텍스트 검색 가능 연합 엔티티(FedTextResource)가 Content Manager 버전 8 텍스트 검색 가능 항목 유형 또는 Extended Search 텍스트 검색 가능 엔티티에 매핑된 경우에만 유효합니다.

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

CONTAINS_TEXT 연산자를 사용하는 연합 매개변수식 조회

속성값에서 텍스트를 검색합니다. 텍스트는 단어 또는 구문이 가능합니다. 이 조회는 텍스트 검색 가능 연합 속성(fJTitle)이 Content Manager 버전 8 텍스트 검색 가능 속성 또는 Extended Search 텍스트 검색 가능 속성에 매핑된 경우에만 유효합니다.

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

연합 텍스트 조회

콘텐츠에서 텍스트를 검색합니다. 텍스트는 단어 또는 구문이 가능합니다. 연합 엔티티가 텍스트 검색 가능할 때만 ASSOCIATED_ENTITY 키워드가 적용됩니다. 이 조회는 텍스트 검색 가능 연합 엔티티(FedEntity)가 Content Manager 버전 8 텍스트 검색 가능 항목 유형 또는 Extended Search 텍스트 검색 가능 엔티티에 매핑된 경우에만 유효합니다.

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

연합 텍스트 조회

콘텐츠에서 텍스트를 검색합니다. 텍스트는 단어 또는 구문이 가능합니다. 연합 텍스트 색인 FedTMINDEX는 Content Manager 버전 7 Text Miner 검색 색인에 매핑됩니다. SEARCH_INDEX 키워드는 이러한 유형의 매핑에만 적용됩니다. 조건에 단어나 구문을 지정하려면 텍스트 검색을 지원하는 Content Manager 버전 7 서버를 정의할 때 구성 문자열을 GENFEDTEXTQRY=YES로 설정해야 합니다.

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

연합 텍스트 조회

Content Manager 버전 7, Content Manager 버전 8 및 Extended Search를 통해 콘텐츠에서 텍스트를 검색합니다. 텍스트는 단어 또는 구문이 가능합니다. 연합 텍스트 색인 FedTMINDEX는 Content Manager 버전 7 Text Miner 검색 색인에 매핑됩니다. SEARCH_INDEX 키워드는 이러한 유형의 매핑에만 적용됩니다. 조건에 단어나 구문을 지정하려면 텍스트 검색을 지원하는 Content

Manager 버전 7 서버를 정의할 때 구성 문자열을 GENFEDTEXTQRY=YES 로 설정해야 합니다. 연합 엔티티가 텍스트 검색 가능할 때만 ASSOCIATED_ENTITY 키워드가 적용됩니다. 이 조회는 텍스트 검색 가능 연합 엔티티(FedTextResource)가 Content Manager 버전 8 텍스트 검색 가능 항목 유형 또는 Extended Search 텍스트 검색 가능 엔티티에 매핑된 경우에만 유효합니다.

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

OR 연산자를 사용하는 연합 매개변수식 및 텍스트 조회

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

AND 연산자를 사용하는 연합 매개변수식 및 텍스트 조회

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

OR 연산자를 사용하는 속성에 대한 연합 매개변수식 및 텍스트 조회

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

OR 연산자를 사용하는 속성에 대한 연합 매개변수식 및 텍스트 조회

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

연합 폴더에 조회 결과 저장(Java 전용)

Enterprise Information Portal 버전 8 릴리스 2는 이제 연합 폴더를 보유할 수 있는 특수 연합 엔티티를 제공합니다. 이 연합 폴더는 연합 조회에서 결합된 결과(예: Content Manager의 문서 및 OnDemand의 관련 문서)를 저장할 수 있습니다. 그런 다음 워크 플로우로 직접 결과를 전송할 수 있습니다.

EIP는 DDO로서 폴더를 저장합니다. DDO는 DKFolder 컬렉션에 추가할 수 있습니다. DKParts 컬렉션에서 XDO로서 폴더를 저장할 수도 있습니다.

특수 연합 엔티티는 추가 테이블을 사용하여 폴더를 보유합니다. 다른 모든 기능(예: 쿼리, API 및 속성)은 일반 연합 엔티티로 동일하게 작동합니다. 특수 엔티티만이 폴더에 DDO PID를 저장합니다.

특수 연합 엔티티를 맵핑하지 않도록 선택하는 경우, 연합 조회만이 특수 연합 엔티티를 검색합니다. 특수 연합 엔티티를 다른 원래의 엔티티로 맵핑하도록 선택하는 경우, 연합 조회는 추가적으로 이 엔티티를 검색합니다.

코드 샘플에 대해서는 CMBROOT\dk\samples 디렉토리를 참조하십시오.

시스템 관리에 대한 작업

Enterprise Information Portal은 사용자가 시스템 관리 기능에 액세스할 수 있도록 하는 클래스 및 API를 제공합니다. 특정 클래스에 대한 정보는 온라인 *API* 참조서를 참조하십시오.

EIP 시스템 관리 클라이언트 사용자 조정

EIP 시스템 관리 클라이언트는 사용자 조정 기능을 포함하도록 시스템 관리 응용프로그램의 확장을 지원합니다.

- EIP 시스템 관리 클라이언트의 사용자 및 사용자 그룹 대화 상자를 사용자의 대화 상자로 대체할 수 있습니다.
- EIP 시스템 관리 클라이언트의 계층 구조에 새 노드를 추가할 수 있습니다.
- 시스템 관리 클라이언트의 도구 메뉴에 새 메뉴 항목을 추가할 수 있습니다.

EIP 시스템 관리에 로그인하기 전과 로그인한 후에 사용자 종료를 호출할 수 있습니다.

API(Application Programming Interface)로 프로그래밍

API(Application Programming Interface)는 로컬 또는 원격 데이터에 액세스하여 조작하는 클래스 세트입니다. 이 절에서는 API, 다중 검색 기능의 구현 및 인터넷 연결에 대해 설명합니다.

API는 다음을 지원합니다.

- 데이터 액세스를 위한 공통 오브젝트 모델
- 이기종 콘텐츠 서버의 조합을 통한 다중 검색 및 갱신
- Content Manager 텍스트 검색 기능과 같은 검색 엔진의 조합을 사용하기 위한 융통성 있는 메커니즘
- 워크플로우 기능
- 관리 기능
- **Java 전용:** Java 응용프로그램용 클라이언트/서버 구현

Java API에 대한 멀티스트림 지원은 Windows® 서버에서만 완전히 작동합니다. AIX 서버, 클라이언트 및 Windows 클라이언트는 멀티스트리밍을 지원하지 않습니다.

Java 및 C++ API 간의 차이점 이해

아래 목록에서는 멀티플랫폼용 IBM Enterprise Information Portal Java와 C++ API 세트 간의 차이점에 대해 설명합니다.

- C++ API에 정의된 연산자가 Java API에는 정의되어 있지 않습니다. 이러한 연산자는 Java 기능으로서 지원됩니다.
- Java 클래스 오브젝트(`java.lang.Object`)가 C++ 클래스 `DKAny` 대신 사용되어 일반 오브젝트를 나타냅니다.
- 공통 및 글로벌 상수는 Java API의 인터페이스 `DKConstant`에 정의됩니다. C++에서는 `DKConstant.h`에 있습니다.
- Java API는 Java의 가비지 콜렉터를 사용합니다.
- Java 함수 `DKDDO.toXML()` 및 `DKDDO.fromXML()`은 C++에서 사용할 수 없습니다.

클라이언트/서버 아키텍처 이해(Java 전용)

API는 응용프로그램 작성자에게 편리한 프로그래밍 인터페이스를 제공합니다. API는 EIP 서버 및 클라이언트에 모두 상주할 수 있습니다(둘다 동일한 인터페이스 제공). 클라이언트 API는 Java RMI(Remote Method Invocation)를 통한 네트워크 데이터에 액세스하기 위해 서버와 통신합니다. 클라이언트와 서버 간의 통신은 클래스에서 수행되므로 다른 프로그램을 추가할 필요가 없습니다.

API 클래스는 서버, 클라이언트, cs 및 공통 등의 패키지로 구성되어 있습니다. 클라이언트 및 서버 클래스는 동일한 API를 제공하지만 다른 구현 방법을 가지고 있습니다.

- 서버 패키지는 `com.ibm.mm.sdk.server`입니다. 서버 패키지의 클래스는 연합 또는 백엔드 콘텐츠 서버와 직접 통신합니다.
- 클라이언트 패키지는 `com.ibm.mm.sdk.client`입니다. 클라이언트 패키지의 클래스는 RMI를 통해 서버 패키지의 클래스와 통신합니다.
- 공통 클래스는 클라이언트와 서버에서 공유합니다. 응용프로그램이 콘텐츠가 상주하는 위치를 모르는 경우가 있습니다. 예를 들어, 응용프로그램은 한 시점에는 클라이언트에 상주하는 콘텐츠를 갖고 다른 시점에는 서버에 상주하는 콘텐츠를 가질 수 있습니다. cs 패키지는 클라이언트와 서버를 동적으로 연결합니다.

클라이언트 응용프로그램은 클라이언트 패키지를 가져오고, 동적 응용프로그램은 cs 패키지를 가져오며 서버 응용프로그램은 서버 패키지를 가져와야 합니다.

클라이언트와 서버에 동일한 API가 제공되지만 클라이언트 패키지는 서버 패키지와 통신하므로 추가 예외 항목을 가집니다. 그러나 클라이언트/서버 인터페이스가 모든 커넥터에 지원되지는 않는다는 점에 유의하십시오. 예를 들어, 이 인터페이스는 CM V8에서 지원되지 않습니다.

Java 환경을 위한 패키지

Enterprise Information Portal API는 `com.ibm.mm.sdk: common, server, client` 및 `cs`의 일부로 네 가지 패키지에 포함되어 있습니다.

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

콘텐츠 서버 정보 액세스 및 조작

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

RMI(Remote Method Invocation)를 사용하는 server 패키지와의 통신

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

서버 패키지, 클라이언트 패키지 및 cs 패키지 모두를 위한 공통 클래스

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

클라이언트 또는 서버에 동적으로 연결

응용프로그램은 로컬 응용프로그램의 경우 server 패키지, 원격 서버에 액세스하는 응용프로그램의 경우 client 패키지 또는 cs 패키지와 함께 common을 사용해야 합니다.

프로그래밍 팁

동일한 프로그램으로 클라이언트와 서버 패키지를 가져오지 마십시오. 클라이언트 응용 프로그램을 개발하는 경우에는 클라이언트 패키지를 가져오십시오. 그렇지 않을 경우에는 서버 패키지를 가져오십시오. 콘텐츠가 상주하는 위치를 모르면 서버 또는 클라이언트 패키지와 함께 cs 패키지를 사용하십시오. 여러 패키지를 가져오면 컴파일 오류가 발생할 수 있습니다.

몇몇 커넥터에는 구현 시 C 코드에 대한 호출이 있습니다. 이들 커넥터의 경우에는 순수 Java 인터페이스를 필요로 하는 웹 응용프로그램용 클라이언트 패키지를 사용하십시오. 클라이언트 패키지는 순수 Java 프로그램으로 작성됩니다. 그러면 서버 패키지는 JNI 호출을 포함할 수 있습니다.

클라이언트에는 예외인 `java.rmi.RemoteException`이 필요하므로 응용프로그램이 실행되는 위치가 서버 또는 클라이언트인지에 상관없이 항상 이 예외를 응용프로그램에 첨부해야 합니다.

Java 환경 설정(Java 전용)

Windows, AIX 또는 Solaris 환경 설정 시 다음 패키지를 가져와야 합니다.

서버 패키지

콘텐츠 서버 및 응용프로그램이 서버측에 있는 경우 다음을 가져오십시오.

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

클라이언트 패키지

콘텐츠 서버 및 응용프로그램이 클라이언트측에 있는 경우 다음을 가져오십시오.

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

cs 패키지

콘텐츠 서버 위치가 응용프로그램 위치와 다른 경우 다음을 가져오십시오.

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

Windows에 대해 Java 환경 변수 설정

시작 --> 프로그램 --> 멀티플랫폼 8.2용 IBM Enterprise Information Portal --> 개발 창을 선택하여 Enterprise Information Portal 응용프로그램을 개발하기 위한 환경

경이 설정된 Windows 명령 프롬프트를 열 수 있습니다. 또는 Windows 명령 프롬프트에서 cmbenv81.bat를 실행하여 환경을 설정할 수 있습니다.

환경 변수를 수정하려면 다음을 변경하십시오.

PATH

PATH에 X:\CMBROOT\DLL이 포함되어 있는지 확인하십시오. 여기서 X는 Enterprise Information Portal을 설치한 드라이브입니다.

CLASSPATH

CLASSPATH에 X:\CMBROOT\LIB\xxx가 포함되어 있는지 확인하십시오. 여기서 X는 Enterprise Information Portal을 설치한 드라이브이며 xxx는 .jar 파일(예: cmbfed81.jar)입니다.

AIX에 대해 Java 환경 변수 설정

AIX 환경에서 셸 스크립트인 cmbenv81.sh를 사용하여 EIP 응용프로그램 개발에 대한 개발 환경을 설정할 수 있습니다.

스크립트를 사용하지 않는 경우 다음과 같은 환경 변수를 설정해야 합니다.

PATH

PATH에 /usr/lpp/cmb/lib가 포함되어 있는지 확인하십시오.

LIBPATH

LIBPATH에 /usr/lpp/cmb/lib가 포함되어 있는지 확인하십시오.

LD_LIBRARY_PATH

LD_LIBRARY_PATH에 /usr/lpp/cmb/lib가 포함되어 있는지 확인하십시오.

CLASSPATH

CLASSPATH에 /usr/lpp/cmb/lib/xxx가 포함되어 있는지 확인하십시오. 여기서 xxx는 .jar 파일(예: cmbfed81.jar)입니다.

오브젝트가 제대로 정렬되도록 -qalign=packed 컴파일러 옵션을 사용하십시오.

Solaris에 대해 Java 환경 변수 설정

Solaris 환경에서 셸 스크립트인 cmbenv81.sh를 사용하여 EIP 응용프로그램 개발에 대한 개발 환경을 설정할 수 있습니다.

스크립트를 사용하지 않는 경우 다음과 같은 환경 변수를 설정해야 합니다.

PATH

PATH에 /opt/cmb/lib가 포함되어 있는지 확인하십시오.

LIBPATH

LIBPATH에 /opt/cmb/lib가 포함되어 있는지 확인하십시오.

LD_LIBRARY_PATH

LD_LIBRARY_PATH에 /opt/cmb/lib가 포함되어 있는지 확인하십시오.

CLASSPATH

CLASSPATH에 /opt/cmb/lib/xxx가 포함되어 있는지 확인하십시오. 여기서 xxx는 .jar 파일(예: cmbfed81.jar)입니다.

오브젝트가 제대로 정렬되도록 -qalign=packed 컴파일러 옵션을 사용하십시오.

컨텐츠 서버에 RMI(Remote Method Invocation) 사용

Java API의 클라이언트 클래스가 네트워크를 통해 데이터에 액세스하기 위해서는 서버 클래스와 통신해야 하므로 서버 및 클라이언트가 모두 클라이언트/서버 실행 준비를 해야 합니다. 서버 시스템에서 지정된 포트 번호를 사용하여 클라이언트 요청을 받으려면 RMI 서버가 실행 중이어야 합니다. 클라이언트 프로그램에는 서버 이름 및 포트 번호가 필요합니다. 클라이언트와 서버 간의 통신에서 클라이언트는 연결해야 할 서버의 포트 번호를 알아야 합니다.

RMI 서버는 무제한의 컨텐츠 서버(시스템 자원으로만 제한)에 연결할 수 있지만 각 서버는 최소한 하나의 컨텐츠 서버에 연결되어야 합니다. 마스터 RMI 서버는 서버 풀의 다른 RMI 서버를 참조할 수 있습니다. RMI 클라이언트가 처음 컨텐츠 서버를 검색할 때는 RMI 서버에서 시작합니다. 여기에 컨텐츠 서버가 없으면 그 다음으로 RMI 풀 서버가 검색됩니다.

동일한 RMI 클라이언트가 컨텐츠 서버를 다시 검색하는 경우, 이 클라이언트는 처음 컨텐츠 서버를 찾은 RMI 서버를 검색합니다.

RMI 서버를 시작하려면 Windows에서는 cmbregist81.bat를 사용하고 AIX 또는 Solaris에서는 cmbregist81.sh를 사용하십시오. RMI 서버를 시작하기 전에 올바른 포트 번호와 서버 유형을 정의하십시오. RMI 서버 구성 및 관리에 대한 정보는 *Enterprise Information Portal* 계획 및 설치와 *Enterprise Information Portal* 관리를 참조하십시오.

C++ 환경 설정(C++ 전용)

Windows 또는 AIX 환경을 설정할 때 이 절에 설명되는 설정을 설정해야 합니다. 34 페이지의 표 4에 라이브러리, AIX 공유 및 DLL 요구사항이 나와 있습니다.

요구사항: C++를 사용하려면 Enterprise Information Portal 데이터베이스에 액세스 중인 모든 원격 서버에 DB2 Client 지원 및 Client Configuration Assistant를 설치해야 합니다. 사용자 ID와 암호는 Enterprise Information Portal 데이터베이스에 사용하는 것과 동일한 사용자 ID와 암호여야 합니다. 세부사항은 *Enterprise Information Portal* 관리를 참조하십시오.

경고: cmbcm81x.lib는 릴리스 빌드용이고 cmbcm81xd.lib는 디버그 빌드용입니다. 여기서 x는 Microsoft Visual C++ 컴파일러의 버전 6이나 7(.Net)을 나타냅니다.

표 4. 공유 오브젝트 및 DLL 환경 정보

커넥터	라이브러리	Windows DLL	AIX용 공유 오브젝트
공통 주: 이것은 커넥터가 아닙니다. 이는 모든 커넥터가 사용하는 API의 공통 세트입니다.	cmbcm81x.lib	cmbcm81x.dll	libcmbcm815.a
	cmbcm81xd.lib	cmbcm81xd.dll	
Content Manager 버전 8	cmbicm81x.lib cmbicm81xd.lib	cmbicm81x.dll	libcmbicm815.a
		cmbicm81xd.dll	libcmbicmfac815.so
		cmbicmfac81x.dll	
		cmbicmfac81xd.dll	
이전 Content Manager	cmbdl81x.lib cmbdl81xd.lib	cmbdl81x.dll	libcmbdl815.a
		cmbdl81xd.dll	libcmbdlfac815.so
		cmbdlfac81x.dll	
		cmbdlfac81xd.dll	
		de_db2.dll de_db2_d.dll	
		de_oracle.dll de_oracle_d.dll	
연합	cmbfed81x.lib cmbfed81xd.lib	cmbfed81x.dll	libcmbfed815.a
		cmbfed81xd.dll	libcmbfedfac815.so
		cmbfedfac81x.dll	
		cmbfedfac81xd.dll	
DB2 Universal Database 버전 8.1	cmbddb281x.lib cmbddb281xd.lib	cmbddb281x.dll	libcmbddb2815.a
		cmbddb281xd.dll	libcmbddb2fac815.so
		cmbddb2fac81x.dll	
		cmbddb2fac81xd.dll	
ODBC	cmbodbc81x.lib cmbodbc81xd.lib	cmbodbc81x.dll	지원되지 않음
		cmbodbc81xd.dll	
		cmbodbcfac81x.dll	
		cmbodbcfac81xd.dll	
OnDemand	cmbod81x.lib cmbod81xd.lib	cmbod81x.dll	libcmbod815.a
		cmbod816xd.dll	libcmbodfac815.so
		cmbodfac81x.dll	
		cmbodfac81xd.dll	
OS/390용 ImagePlus	cmbip81x.lib cmbip81xd.lib	cmbip81x.dll	지원되지 않음
		cmbip81xd.dll	
		cmbipfac81x.dll	
		cmbipfac81xd.dll	
VisualInfo	cmbv481x.lib cmbv481xd.lib	cmbv481x.dll	지원되지 않음
		cmbv481xd.dll	
		cmbv4fac81x.dll	
		cmbv4fac81xd.dll	
Domino.Doc	cmbdd81x.lib cmbdd81xd.lib	cmbdd81x.dll	지원되지 않음
		cmbdd81xd.dll	
		cmbddf81x.dll	
		cmbddf81xd.dll	

표 4. 공유 오브젝트 및 DLL 환경 정보 (계속)

커넥터	라이브러리	Windows DLL	AIX용 공유 오브젝트
Domino Extended Search	cmbdes81x.lib cmbdes81xd.lib	cmbdes81x.dll cmbdes81xd.dll cmbdesfac81x.dll cmbdesfac81xd.dll	libcmbdes815.a libcmbodfac815.so

Windows용 C++ 환경 변수 설정

시작 --> 프로그램 --> 멀티플랫폼 8.1용 IBM Enterprise Information Portal --> 개발 창을 눌러 Enterprise Information Portal 응용프로그램을 개발하기 위한 환경이 구성된 DOS 명령 프롬프트를 열 수 있습니다. 또는 DOS 명령 프롬프트에서 CMBenv81.bat를 실행하여 환경을 설정할 수 있습니다.

환경 변수를 수정하려면 다음을 변경하십시오.

PATH

```
set PATH=x:\CMBROOT\DLL
```

여기서 x는 EIP가 설치된 드라이브입니다.

INCLUDE

```
set INCLUDE=x:\CMBROOT\INCLUDE
```

여기서 x는 EIP가 설치된 드라이브입니다.

AIX용 C++ 환경 변수 설정

자세한 정보는 samples 디렉토리의 샘플 MAK 파일을 참조하십시오.

다음 환경 변수를 설정하십시오.

AIX 환경에서 세 가지 일괄처리 파일 중 하나를 사용하여 개발 환경을 설정할 수 있습니다.

1. Bourne 셸의 경우, cmbenv81.sh를 사용하십시오.
2. C 셸의 경우, cmbenv81.sh를 사용하십시오.
3. Korn 셸의 경우, cmbenv81.ksh를 사용하십시오.

다음 환경 변수를 설정하십시오.

NLS path

```
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++ 프로그램 빌드

컴파일러 및 개발 환경에 대한 프로시저에 따라 MAK 파일을 작성한 후 응용프로그램을 빌드하십시오.

C++ API를 사용하여 디버깅에 사용할 응용프로그램을 빌드할 때 응용프로그램을 API 라이브러리의 디버그 버전, 즉 ***d.lib** 라이브러리로 링크하십시오. 최종 프로덕션 응용프로그램을 빌드할 때 디버깅하지 않은 라이브러리(**.lib**)로 링크하십시오.

Microsoft Visual Studio .NET에 대한 작업

Enterprise Information Portal 및 Content Manager 버전 8.2 이상 API는 이제 Microsoft Visual Studio .NET을 지원합니다. 그러나 응용프로그램을 빌드하기 위해 Microsoft Visual Studio .NET을 사용할 경우 컴파일 시 사용 중인 커넥터 및 Visual Studio C++ 버전으로 결정된 해당 라이브러리에 링크해야 합니다.

MAK 파일 예제(Visual Studio 버전 6 및 Visual Studio .NET을 둘 다 지원)는 ICM API 샘플로 제공됩니다.

컴파일 시 연결해야 할 라이브러리를 결정하려면 아래 표에서 설명한 것처럼 Microsoft Visual Studio C++ 6을 사용하는 경우에는 커넥터 이름 816.lib 형식을 사용하고 Microsoft Visual Studio .NET을 사용하는 경우에는 커넥터 이름 817.lib를 사용하십시오.

표 5. Microsoft Visual Studio C++ 6 라이브러리 이름 예제

커넥터	라이브러리
공통	cmbcm816.lib
Content Manager 8.1 이상	cmbicm816.lib
연합	cmbfed816.lib

표 6. Microsoft Visual Studio .NET 라이브러리 이름

커넥터	라이브러리
공통	cmbcm817.lib
Content Manager 8.1 이상	cmbicm817.lib
연합	cmbfed817.lib

Windows에서 코드 페이지 변환을 위한 콘솔 서브시스템 설정

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);
    ...
}
```

다중 검색 옵션 이해

검색은 항목이나 구성요소의 모든 부분, 항목이나 구성요소의 텍스트 또는 자원 콘텐츠의 텍스트를 기반으로 가상적으로 수행될 수 있습니다.

Content Manager 버전 8은 이전 Content Manager와 같이 더 이상 별도의 텍스트 검색 기능이 필요하지 않는 통합 텍스트 기능을 제공합니다. 215 페이지의 『텍스트 검색 이해』를 참조하십시오.

다중 검색 옵션을 사용하여 아래 나열된 하나 이상의 지원 조회를 사용하는 지정된 콘텐츠 서버를 검색하거나 이전 검색 결과를 검색하십시오. 각 검색 유형은 하나 이상의 검색 엔진에 의해 지원됩니다. 모든 콘텐츠 서버가 다중 검색 옵션을 지원하는 것은 아닙니다.

매개변수식 검색

항목 및 구성요소 등록 정보, 속성, 참조, 링크, 폴더 콘텐츠 등의 텍스트를 검색합니다. 예를 들어, 매개변수식 조회를 사용하여 고객의 이름별로 문서를 검색합니다. 조회는 조회 술어에 지정된 조건과 콘텐츠 서버에 저장된 데이터 값이 정확하게 일치해야 합니다.

텍스트 검색

DB2 NSE(Net Search Engine)와 같은 텍스트 검색 엔진을 사용하여 `textSearchable(true)`로 표시된 텍스트를 검색합니다. 조회는 지정된 텍스트 검색 표현식과 대략 일치하는 텍스트 필드의 콘텐츠에 기반합니다. 예를 들어, 특정 구문 또는 어간의 존재 여부를 조회합니다.

주: DB2의 이전 버전에서는 DB2 NSE(Net Search Engine)를 DB2 TIE(Text Information Extender)라고 했습니다.

이미지 검색

이미지 내의 특성을 검색합니다. 조회는 지정된 이미지 검색 표현식과 대략 일치하는 이미지 콘텐츠에 기반합니다. 예를 들어, 이미지에 특정 색상이 있는지를 조회합니다.

결합 검색

매개변수식과 텍스트 검색을 모두 사용하여 검색합니다.

Content Manager 전용: CM은 하나의 검색 엔진과, 검색이 실행되어야 할 방법 및 시기(그리고 결과를 리턴하는 방법 및 시기)에 대해 세 가지 선택사항인 실행, 평가 및 콜백으로 실행을 가집니다. 테이블 및 설명에 대해서는 SSearchICM 샘플을 참조하십시오.

추적

API 응용프로그램에서 발생하는 문제점을 처리하기 위해 추적 및 예외 처리를 사용할 수 있습니다.

텍스트 검색 엔진을 사용하는 텍스트 조회 추적

텍스트 검색 엔진(TSE)과 모든 해당 기능은 이전 Content Manager에서만 사용할 수 있습니다. Content Manager 버전 8은 별도의 텍스트 검색 기능이 필요 없는 통합 텍스트 기능을 제공합니다. 215 페이지의 『텍스트 검색 이해』를 참조하십시오.

다음 환경 변수 설정은 텍스트 검색 엔진 조회에 대한 추적을 2진 형식으로 지정된 파일에 기록합니다.

`CMBTMDSTREAMTRACE=fileName`

(예: Windows용 `.\tm.out` 또는 AIX용 `./tm.out`)

다음 환경 변수 설정은 텍스트 조회 중에 사용된 텍스트 검색 엔진 API 호출에 대한 추적을 지정된 파일에 기록합니다.

`CMBTMTRACE=fileName`

다음 환경 설정은 텍스트 검색 용어를 지정된 파일인 `CMBTMTERM=fileName` (예: `.\tmterm.out`)에 기록합니다.

주: Content Manager 버전 8은 통합 텍스트 검색을 사용합니다. 이전 Content Manager 를 사용하는 경우에는 텍스트 검색 엔진(TSE)을 계속 사용할 수 있습니다.

매개변수식 조회 추적

텍스트 검색 엔진을 사용한 이전 Content Manager의 경우, 폴더 관리자로 전달된 매개변수식 조회를 기록하려면 다음 환경 변수 설정을 사용하십시오.

`CMBDLQRYTRACE=fileName`

(예: Windows용 `<.\dlqry.out>` 또는 AIX용 `<./dlqry.out>`)

예외 처리

API에 문제점이 발생하면 예외를 발생시킵니다. 예외가 발생하면 `DKException` 클래스 또는 해당 서브클래스 중 하나에 대한 예외 오브젝트가 작성됩니다.

`DKException`이 작성되면, 커넥터 계층은 기본 로그 기록 구성이 사용된다고 간주하고 로그 파일에 진단 정보를 기록합니다. EIP API가 사용하는 로그 및 구성 파일에 대한 자세한 정보는 메시지 및 코드를 참조하십시오.

`DKException`이 포착되면 실행 중 발생한 오류 메시지, 오류 코드 및 오류 상태를 볼 수 있습니다. 오류가 포착되면 예외가 발생한 위치와 함께 오류 메시지가 발행됩니다. 오류 ID와 같은 추가 정보도 제공됩니다. 아래 코드에서는 EIP 및 CM용 발생 및 포착 프로세스의 예제를 표시합니다.

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;
}
```

오류 발견 및 처리에 대한 자세한 정보는 CMBROOT\Samples\java\icm 또는 CMBROOT\Samples\cpp\icm에 있는 SConnectDisconnectICM API 교육 샘플을 참조하십시오.

상수

지정된 상수는 DK_CM_(공통 상수) 또는 DK_XX_ 양식을 갖습니다.(여기서 XX는 다른 컨테이너 서버를 나타냅니다.) 확장자(각 DKDatastore에 첨부된 확장자) 목록에 대해서는 42 페이지의 표 7을 참조하십시오.

DDO 상수를 지정하는 경우, 등록 정보 유형에는 DK_CM_DATAITEM_TYPE_ ...(예: DK_CM_DATAITEM_TYPE_STRING)을 사용하십시오. 속성 유형의 경우에는 DK_CM_...type 상수(예: DK_CM_INTEGER)를 사용하십시오.

Java

공통 상수는 DKConstant.java에 정의됩니다. 또한 DKConstant.txt에서 이들 상수의 텍스트 버전을 검토할 수 있습니다. 특정 콘텐츠 서버용 상수는 DKConstantXX.java에 정의됩니다. 예를 들어, Content Manager에 고유한 상수는 DKConstantICM.java에 있습니다.

C++

공통 상수는 DKConstant.h에 정의됩니다. 상수 및 해당 값의 목록에 대해서는 DKConstant2.h를 검토하십시오(그러나 프로그램에는 포함되지 않습니다). 특정 콘텐츠 서버용 상수는 양식 DKConstantXX.h의 헤더 파일에 정의됩니다. 예를 들어, Content Manager에 고유한 상수는 DKConstantICM.h에 있습니다.

콘텐츠 서버에 연결

클래스 DKDatastorexx(여기서 xx는 특정 콘텐츠 서버를 나타냄)의 오브젝트는 콘텐츠 서버에 대한 연결을 표현 및 관리하고 트랜잭션 지원을 제공하며 서버 명령을 실행합니다. 정확한 확장자에 대해서는 표 7을 참조하십시오.

표 7. 서버 유형 및 클래스 이름 용어

콘텐츠 서버	클래스 이름
Content Manager 버전 8.2	DKDatastoreICM
이전 Content Manager	DKDatastoreDL
Content Manager OnDemand	DKDatastoreOD
AS/400용 Content Manager(AS/400용 VisualInfo)	DKDatastoreV4
OS/390용 Content Manager ImagePlus	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
Panagon Image Services(FileNET)	DKDatastoreFN
관계형 데이터베이스	DKDatastoreDB2, DKDatastoreJDBC(Java용) DKDatastoreODBC(C++용)

연결 설정

각 DKDatastorexx 클래스는 연결 또는 연결 해제 메소드를 제공합니다. 다음 예제에서는 ICMNLSDB라는 이전 Content Manager 라이브러리 서버, 사용자 ID ICMADMIN 및 암호 PASSWORD를 사용합니다. Content Manager에 대한 정보는 Content Manager

시스템에 연결을 참조하십시오. 다른 콘텐츠 서버에 대해서는 기타 콘텐츠 서버에 대한 작업을 참조하십시오. 이 예제에서는 CM 콘텐츠 서버용 DKDatastoreICM 오브젝트를 작성하고 연결하여 작업한 후 연결을 해제합니다.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //Create datastore object
dsICM.connect("ICMNLSD", "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
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SConnectDisconnectICM.java를 참조하십시오.

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM(); //Create datastore object
dsICM->connect("ICMNLSD", "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
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SConnectDisconnectICM.java를 참조하십시오.

콘텐츠 서버에 연결할 때는 각 콘텐츠 서버의 요구사항을 알아야 합니다. 예를 들어, OS/390용 ImagePlus의 암호는 8자를 초과할 수 없습니다.

클라이언트의 콘텐츠 서버에서 연결 및 연결 해제

42 페이지의 『연결 설정』의 동일한 코드를 사용하여 클라이언트 응용프로그램에서 콘텐츠 서버에 액세스합니다. 이렇게 하려면 import com.ibm.mm.sdk.server.*;를 import com.ibm.mm.sdk.client.*;로 대체하십시오. 클라이언트 응용프로그램은 발생한 통신 오류를 모두 처리해야 합니다.

콘텐츠 서버 옵션 설정 및 가져오기

DKDatastorexx의 메소드를 사용하여 콘텐츠 서버의 처리 옵션에 액세스하거나 옵션을 설정할 수 있습니다. 다음 예제에서는 Content Manager 라이브러리 서버에 관리 세션

을 설정하는 데 필요한 옵션을 설정하고 가져오는 방법을 보여줍니다. 옵션 목록 및 해당 설명에 대해서는 온라인 *API* 참조서를 참조하십시오.

Content Manager의 콘텐츠 서버 옵션을 설정하고 가져오는 방법에 대한 예제에서 캐싱을 끕니다. 이 옵션을 지원하는 콘텐츠 서버의 경우, 기본값(ON)이 사용되어야 합니다.

요구사항: 올바른 DKDatastoreICM 오브젝트는 이미 dsICM이라는 변수에 작성되었습니다.

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);
```

콘텐츠 서버 옵션을 가져올 때, output_option은 대개 정수이지만 오브젝트로 캐스트할 수 있습니다.

콘텐츠 서버 목록

DKDatastore.xx는 연결할 수 있는 서버를 나열하는 메소드를 제공합니다. 서버 목록은 DKServerDefxx 오브젝트의 DKSequentialCollection에 리턴됩니다.(여기서 xx는 특정 콘텐츠 서버를 식별합니다.)

제한사항: Domino.Doc 콘텐츠 서버는 서버를 나열하는 메소드를 제공하지 않습니다.

DKServerDefxx 오브젝트를 얻은 후에는 서버 이름과 서버 유형을 검색할 수 있으며 서버 이름을 사용하여 연결을 설정할 수 있습니다.

다음 예제에서는 연결하려고 구성한 서버를 나열합니다.

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);
```

컨텐츠 서버의 엔티티 및 속성 목록

컨텐츠 서버의 경우, DKDatastorexx는 엔티티 및 속성을 나열할 수 있는 메소드를 제공합니다. 각 속성 이름은 이름 공간의 일부입니다. 기본 이름 공간은 이름 공간이 지정되지 않은 모든 속성에 사용됩니다.

엔티티 목록은 dkEntityDef 오브젝트의 DKSequentialCollection 오브젝트에 리턴됩니다. 엔티티의 속성은 dkAttrDef 오브젝트의 DKSequentialCollection 오브젝트에 리턴됩니다. dkAttrDef 오브젝트를 확보한 후에는 이름 및 유형과 같은 속성 정보를 검색할 수 있으며 이 정보를 사용하여 조회를 형성할 수 있습니다.

이러한 두 메소드에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

다음 예제에서는 Content Manager 서버로부터 속성 목록뿐만 아니라 항목 유형 목록을 검색하는 방법을 보여줍니다.

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(계속)

```
// ----- 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
```

완전한 샘플인 SItemTypeRetrievalICM에는 항목 유형 정의를 나열하는 방법이 들어 있습니다. 또다른 완전한 샘플인 SAttributeDefinitionRetrievalICM에는 속성 정의를 나열하는 방법이 들어 있습니다. 두 샘플은 모두 CMBROOT\Samples\java\icm 디렉토리에 제공됩니다.

다음 C++ 예제에서는 색인 클래스의 목록을 검색하는 방법과 Content Manager 서버에서 속성을 검색하는 방법을 보여줍니다.

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);
```

자세한 정보는 항목 유형 정의를 나열하는 방법이 들어 있는 SItemTypeRetrievalICM 샘플을 참조하십시오. SAttributeDefinitionRetrievalICM에는 속성 정의를 나열하는 방법이 들어 있습니다. 이 샘플은 CMBROOT\Samples\cpp\icm 디렉토리에 제공됩니다.

팁: 즉시 pEnt를 삭제하는 대신, 이를 연기하고 apply 함수를 사용하여 컬렉션 내부의 속성 정의를 삭제할 수 있습니다. 자세한 정보는 112 페이지의 『컬렉션의 메모리 관리(C++ 전용)』를 참조하십시오.

C++

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

동적 데이터 오브젝트(DDO)에 대한 작업

이 절에서는 DDO를 사용하는 방법에 대해 설명하며, 다음 방법을 학습하는 데 도움을 주는 예제가 들어 있습니다.

1. DKDDO와 콘텐츠 서버를 연결합니다.
2. DKDDO를 작성합니다.
3. DKDDO 속성에 대한 지속 ID(PID)를 작성합니다.
4. 속성을 추가하고 속성 등록 정보를 정의합니다.

5. DKDDO를 폴더 또는 문서로 정의합니다.
6. 속성 등록 정보에 대한 값을 설정 및 봅니다.
7. DKDDO 등록 정보를 점검합니다.
8. 속성 등록 정보를 점검합니다.
9. DKDDO 콘텐츠를 표시합니다.
10. DKDDO를 삭제합니다.

멀티플랫폼용 IBM Enterprise Information Portal 응용프로그램에서는 동적 데이터 오브젝트(DDO)에 대해 DKDDO 클래스를 사용합니다. DKDDO 오브젝트는 Content Manager 문서, 폴더 또는 사용자 정의 오브젝트와 같은 항목을 나타냅니다. DKDDO 오브젝트는 속성을 포함합니다. 각 속성은 이름, 값 및 등록 정보를 갖고 있습니다. 각 속성은 데이터 ID로 식별됩니다. 속성은 1부터 시작하여 차례로 번호가 지정됩니다. 속성 번호는 데이터 ID입니다.

속성의 이름, 값 및 등록 정보는 다를 수 있으므로 DKDDO는 다양한 콘텐츠 서버의 데이터 및 여러 가지 형식의 데이터를 나타낼 수 있도록 융통성 있는 메커니즘을 제공합니다. 예를 들어, Content Manager에 있는 다른 항목 유형의 항목이나 관계형 데이터베이스의 다른 테이블에 들어 있는 행이 있습니다. DKDDO는 하나의 특정 속성 대신 전체 DKDDO에 적용되는 등록 정보를 가질 수 있습니다.

add, retrieve, update 및 delete 메소드를 호출하기 전에 DKDDO를 콘텐츠 서버와 연결하여 해당 속성을 콘텐츠 서버에 배치하거나 검색하십시오. DKDDO 오브젝트를 작성할 때 매개변수로 콘텐츠 서버를 설정하거나 setDatastore 메소드를 호출하여 콘텐츠 서버를 설정합니다.

모든 DKDDO는 콘텐츠 서버의 속성을 찾는 데 필요한 정보가 들어 있는 지속 오브젝트 식별자(PID)를 가지고 있습니다.

DKDDO 작성

DKDDO는 여러 개의 구성자를 가지고 있습니다. 매개변수 없이 해당 구성자를 호출하여 DKDDO를 작성할 수 있습니다.

Java

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

C++

```
DKDDO ddo;
```

더 많은 속성을 적용하려면 이 DDO ddo가 동적으로 증가해야 합니다. 더 효율적인 구성자의 경우, 원하는 속성의 정확한 숫자를 전달하십시오(예: 10).

Java

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

C++

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

중요사항: DKDatastoreICM 및 DKDatastoreOD와 같은 API에서 DKDatastore XX 클래스의 createDDO() 메소드를 사용하여 DKDDO를 작성하십시오. CM V8에서 DDO는 이 메소드를 사용하여 작성해야 합니다. 다음 예제에서는 Content Manager 버전 8용 콘텐츠 서버 및 오브젝트 유형을 둘다 전달하여 DKDDO를 작성합니다.

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 작성에 대한 자세한 정보는 CMBROOT\Samples\java\icm에 있는 SitemCreationICM 샘플을 참조하십시오.

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 작성에 대한 자세한 정보는 CMBROOT\Samples\java\icm에 있는 SItemCreationICM 샘플을 참조하십시오.

다른 커넥터(예: 이전 Content Manager)의 경우, 다음 예제를 사용하여 콘텐츠 서버 및 오브젝트 유형을 제공함으로써 DKDDO를 작성할 수 있습니다.

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");
```

사용하는 구성자는 응용프로그램에 따라 다릅니다. 구성자에 대한 정보는 온라인 API 참조서를 참조하십시오.

DDO에 등록 정보 추가

DDO를 나타내는 DKDDO 오브젝트 작성 시 해당 항목유형 등록 정보(문서, 폴더 또는 항목)를 지정해야 합니다.

이 등록 정보를 DKDatastoreXX.createDDO(*itemTypeName*, *itemPropertyType*/*SemanticType*) 메소드의 옵션 중 하나로 전달할 수 있습니다.

```
DKDatastoreICM.createDDO(itemTypeName,itemPropertyType/SemanticType)
```

또는 해당 항목 유형 등록 정보를 설정하지 않고 이미 DKDDO를 작성한 경우, 다음 예제를 사용하여 DDO 유형을 "document"로 설정할 수 있습니다.

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);
```

지속 식별자(PID) 작성

각 DDO는 지속 식별자(PID)를 가지고 있어야 합니다. PID에는 콘텐츠 서버의 이름, 유형, ID 및 오브젝트 유형에 대한 정보가 들어 있습니다. PID는 DDO의 지속 데이터 위치를 식별합니다. 예를 들어, 이전 Content Manager 콘텐츠 서버에서 PID는 항목 ID입니다. 항목 ID는 retrieve, update 및 delete 함수에서 가장 중요한 매개변수 중 하나입니다. Content Manager V8에서 PID에는 다섯 부분이 있습니다.(자세한 정보는 *Content Manager 8.2에 대한 작업을 참조하십시오*.)

add 함수의 경우, 콘텐츠 서버는 항목 ID를 작성하여 리턴합니다. 예를 들어, DKDatastoreXX.createDDO() 메소드를 제공하는 커넥터는 자동으로 DKDDO.add() 조 작에서 PID 오브젝트를 작성합니다.

다음 예제에서는 알려진 항목을 검색하기 위한 DDO를 작성합니다.

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();
```

컨텐츠 서버에 연결하고 retrieve 함수를 호출하여 예제에서 작성된 DDO를 검색하십시오.

Content Manager 8.2 커넥터에는 DKPid의 서브클래스인 PID 클래스가 있습니다. 이를 DKPidICM이라 하며 DKDDOs 및 dkResources에서 사용하는 PID입니다.

데이터 항목 및 등록 정보에 대한 작업

DKDDO는 DKDDO 오브젝트에 속성 및 속성 등록 정보를 추가할 수 있는 메소드를 제공합니다.

항목 유형 NameOfItemType이 integer 유형의 Name 속성을 지니고 CM V8 저장소에서 널 값 가능하게 정의된다고 가정하십시오. 해당 엔티티의 항목을 처리하기 위한 DKDDO 오브젝트를 작성하고 두 개의 데이터 항목을 DKDDO에 추가하려 합니다.

다음 예제에서는 항목을 작성하고 속성 등록 정보를 설정하며 Content Manager의 지속 컨텐츠 서버에 저장합니다.

요구사항: 변수 dsICM에 연결된 DKDatastoreICM이 있다고 가정하십시오. 사용자 정의 항목 유형 S_withChild는 SItemTypeCreationICM API 교육 샘플에서 정의한 대로 varchar S_varchar, long integer S_integer, short integer S_short 및 time S_time으로 정의되어야 합니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(SItemCreationICM)은
CMBROOT\Samples\java\icm 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(SItemCreationICM)은
CMBROOT\Samples\cpp\icm 디렉토리에 있습니다.

속성에 대해 등록 정보 유형은 반드시 설정해야 하며 널 값 가능 및 다른 등록 정보는
선택적입니다.

다음 예제에서는 DDO의 속성값에 액세스합니다.

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);
```

이 예제가 수행된 완전한 샘플 응용프로그램(SItemRetrievalICM)은
CMBROOT\Samples\java\icm 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(SItemRetrievalICM)은
CMBROOT\Samples\cpp\icm 디렉토리에 있습니다.

DKDDO 및 속성 등록 정보 가져오기

DKDDO 처리 시 문서, 폴더 또는 항목 유형을 먼저 알아야 합니다. 다음 샘플 코드는 DDO 유형 판별 방법에 대해 설명합니다.

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;
    }
}
```

항목 유형 등록 정보에 액세스하는 방법에 대한 자세한 정보는
CMBROOT\Samples\java\icm 디렉토리에 제공되는 SItemRetrievalICM API
교육 샘플을 참조하십시오.

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;
    }
}
```

항목 유형 등록 정보에 액세스하는 방법에 대한 자세한 정보는
CMBROOT\Samples\cpp\icm 디렉토리에 제공되는 SItemRetrievalICM API 교
육 샘플을 참조하십시오.

속성의 등록 정보를 검색하려면 해당 속성의 data_id를 알아야 합니다. 그런 다음 등
록 정보를 검색할 수 있습니다.

data_id 및 property_id는 1에서 시작합니다. 0을 지정하면 예외가 수신됩니다.

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));
}
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SItemRetrievalICM을 참조하십시오.

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;
}
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SItemRetrievalICM을 참조하십시오.

전체 DDO 표시

응용프로그램 개발 중에 디버깅 목적으로 DKDDO의 콘텐츠를 표시해야 하는 경우가 있습니다.

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(및 모든 하위 구성요소)에 액세스하고 인쇄하는 완전한 예제에 대해서는 SItemRetrievalICM 및 CMBROOT\Samples\java\icm 디렉토리의 printDDO() static 함수를 참조하십시오.

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(및 모든 하위 구성요소)에 액세스하고 인쇄하는 완전한 예제에 대해서는 SItemRetrievalICM 및 CMBROOT\Samples\cpp\icm 디렉토리의 printDDO() static 함수를 참조하십시오.

DDO 삭제(C++ 전용)

DKDDO에는 메모리의 표현 및 지속적 복사라는 두 개의 표현이 있습니다. 메모리에서 DKDDO를 삭제하려면 소멸자를 호출하십시오. 콘텐츠 서버에 지속적 복사가 변경되지 않고 남아있다는 점에 유의하십시오.

dkddo:del() 함수로 콘텐츠 서버에서 지속적 복사를 삭제합니다. 메모리의 DKDDO 표현에 영향을 주지 않습니다.(속성값은 DKAny 오브젝트에 있습니다.) 제거 프로그램은 DKParts, DKFolder, DKDDO 및 DKBlob에 대한 참조를 포함한 dkCollection 및 dkDataObjectBase에 대한 오브젝트 참조를 삭제합니다.

확장 데이터 오브젝트(XDO)에 대한 작업

XDO는 자원 콘텐츠(예: 자원 항목 또는 문서 부분)를 저장할 수 있는 구성요소를 나타냅니다.

자원 항목(XDO)은 자원이 아닌 항목(DDO)을 확장합니다. 일반 항목과 같은 자원 항목을 작성합니다. 일반 항목과 같이 자원 항목 영역이 작성됩니다. XDO는 자원 항목 유형에 따라 나중에 추가로 확장될 수 있습니다.

Java

Class Hierarchy

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

Content Manager 전용: CM 8에는 올바른 서브클래스가 될 XDO가 필요하므로 DKDDO는 항상 DKDatastoreICM의 createDDO() 메소드를 사용하여 작성해야 합니다. 그러면 시스템이 자동으로 DKDDO 구조에 중요한 정보를 설정하고 더 큰 기능(예: 자원, CM 문서 모델 및 폴더)을 제공할 수 있습니다. 유형 자원의 항목(DKDatastoreICM.createDDO에서 리턴됨)은 XDO 분류에 따라 올바른 XDO 또는 서브클래스로 캐스트될 수 있습니다. 일반적으로 항목 및 DKDatastoreICM.createDDO() 함수 작성에 대한 자세한 정보는 SItemCreationICM 샘플을 참조하십시오.

2진 오브젝트에 대해 XDO를 작성하려면 DKBlobxx를 사용하십시오. 여기서 xx는 특정 서버를 나타내는 접미부입니다. 예를 들어, Content Manager용 DKBlobICM, OnDemand용 DKBlobOD 또는 OS/390용 ImagePlus에 대해 DKBlobIP를 사용하십시오. DKBlobxx 오브젝트 작성시 콘텐츠 서버 DKDatastorexx로 전달해야 합니다. Content Manager의 경우, DKLobICM을 사용하여 XDO를 작성하십시오.

XDO 지속 식별자(PID) 사용

XDO에는 데이터를 지속적으로 저장하기 위해 PID가 필요합니다. XDO를 사용하여 데이터를 찾아 저장하려면 DKPidXDOxx를 사용하여 DKBlobxx에 PID를 제공해야 합니다. 콘텐츠 서버에서 지속 데이터를 찾으려면 관계형 데이터베이스에는 테이블, 열 및 데이터 술어 문자열이 있어야 합니다. DKPidXDOxx의 경우, 관계형 데이터베이스(RDB), 테이블 이름, 열 이름 및 데이터 술어가 필요합니다.

ICM 커넥터에서 XDO인 dkResource 오브젝트의 PID를 나타내는 데 DKPidICM을 사용하십시오.

XDO 등록 정보 이해

적용할 XDO의 등록 정보를 설정하려면 DKBlobxx 메소드를 사용하십시오. 모든 등록 정보를 모든 콘텐츠 서버에 대해 사용할 수 있는 것은 아닙니다. 로드할 때 특정 값이 지정되어 있지 않으면 해당 등록 정보의 기본값이 설정됩니다. 예를 들어, 다음 기본값은 이전 Content Manager에서 사용됩니다.

RepType(표현 유형)

기본값은 FRN\$NULL입니다. AS/400용 VisualInfo의 경우, 큰따옴표로 묶은 8자리 빈 공백 " "을 사용해야 합니다.

ContentClass

기본값은 DK_CM_CC_UNKNOWN입니다. Enterprise Information Portal의 경우, 올바른 값을 보려면 \cmbroot\include 디렉토리의 DKConstant2DL.h를 참조하십시오.

AffiliatedType

기본값은 DK_DL_BASE입니다.

AffiliatedData

기본값은 NULL입니다.

이전 Content Manager에서 제대로 오브젝트 콘텐츠를 색인화하려면 확장자 오브젝트 DKSearchEngineInfoDL에 SearchEngine, SearchIndex 및 SearchInfo를 설정해야 합니다.

Content Manager의 XDO에 대한 작업에 대해서는 163 페이지의 『항목에 대한 작업』을 참조하십시오.

C++ 팁: ContentClass의 올바른 값에 대해서는 Content Manager에 제공한 INCLUDE/DKConstant2DL.h 파일을 참조하십시오.

DB2, ODBC 및 DataJoiner 구성 문자열(C++ 전용)

이 절에서는 C++ DB2, ODBC 및 DataJoiner 구성 문자열을 정의합니다.

CC2MIMEFILE=(filename)

cmbcc2mime.ini 파일을 지정합니다(선택적).

DSNAME=(content server name)

콘텐츠 서버 이름을 지정합니다(선택적). 이 콘텐츠를 서버를 연합으로 사용하면 이 옵션은 자동으로 설정됩니다.

AUTOCOMMIT=ON | OFF

자동 확약을 설정 또는 해제하도록 지정합니다. 기본값은 해제입니다(선택적). 이 콘텐츠를 서버를 연합으로 사용하면 자동 확약은 항상 설정됩니다. 이 값은 자동으로 설정됩니다.

이 절에서는 C++ DB2, ODBC 및 DataJoiner 콘텐츠 문자열을 정의합니다.

NATIVECONNECTSTRING=(native connect string)

원래의 연결 호출에 전달할 원래의 연결 문자열을 지정합니다(선택적).

SCHEMA=name

listEntities, listEntityAttrs, listPrimaryKeyNames,
listForeignKeyNames 함수에 사용할 스키마를 지정합니다(선택적).

Java 프로그래밍 팁

Content Manager V8 이상의 경우, XDO는 dkResource 오브젝트입니다. DKPidICM 을 사용하여 자원 오브젝트의 PID를 나타냅니다.

이전 Content Manager, AS/400용 Content Manager 및 IP 390의 경우, 항목 ID, 부분 ID 및 RepType의 조합으로 XDO를 식별합니다. RDB의 경우, XDO를 식별하기 위한 키는 테이블, 열 및 데이터 술어 문자열의 조합입니다. 독립형 XDO를 처리하려면 항목 ID와 부분 ID를 제공하십시오. 시스템에서 기본값이 제공되므로 RepType은 선택적입니다.

컨텐츠 서버에 현재 콘텐츠를 추가하려면 xx의 추가 메소드를 사용하십시오. 나중에 해당 오브젝트에 대해 다른 작업을 수행하려고 할 경우, add 다음에 부분 ID 값을 검색할 수 있습니다.

DKPid 오브젝트를 가져오려면 dkXDO의 getPidObject() 메소드를 사용하십시오.

add 이후에 다음 명령문을 사용하여 부분 ID가 할당된 시스템을 얻을 수 있습니다.

Java

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

경고: 이전 Content Manager에서 검색 관리자가 색인화한 부분을 추가하려면 올바른 부분 ID가 필요합니다(부분 ID를 0으로 설정할 수 없음).

이 릴리스에서는 dkXDO의 두 가지 메소드가 수정되었습니다. 즉, DKPid dkXDO.getPid()가 거부되고 getPidObject로 대체되었습니다. DKPid dkXDO.getPidObject() 메소드는 DKPidXDO를 리턴했지만 이제는 DKPid 오브젝트를 리턴합니다.

C++ 프로그래밍 팁

Content Manager, VI400® 및 IP390의 경우, 항목 ID, 부분 ID 및 RepType의 조합으로 XDO를 식별합니다. 관계형 데이터베이스에 대해 테이블 이름, 열 이름 및 데이터 술어의 조합은 XDO를 식별하는 키가 됩니다. 독립형 XDO의 경우, 항목 ID와 부분 ID를 제공해야 합니다. 시스템에서 기본값(FRN\$NULL)이 제공되므로 RepType은 선택적입니다.

add 함수의 경우, 부분 ID를 제공해야 합니다. 나중에 해당 오브젝트에 대해 다른 작업을 수행하려고 할 경우, add 다음에 부분 ID 값을 검색할 수 있습니다.

중요사항: 검색 관리자가 Content Manager 콘텐츠 서버에서 색인화할 부분을 추가할 경우, 올바른 부분 ID가 있어야 하고 부분 ID는 0으로 설정할 수 없습니다.

XDO를 DDO의 부분으로 프로그래밍

문서를 나타내는 DDO를 가지고 있는 경우, XDO는 자원 콘텐츠 오브젝트의 컬렉션인 단일 부분 오브젝트를 나타냅니다. XDO를 DDO의 구성요소나 독립형 오브젝트로 조작할 수 있습니다. DDO의 부분으로 XDO에 액세스하면 DDO는 항목 ID를 제공합니다. XDO를 독립형 오브젝트로 사용할 경우, 해당 XDO에 대해 기존 항목 ID를 사용합니다.

다음 예제에서는 문서를 작성하고 Content Manager에 문서 부분을 추가합니다.

요구사항: S_docModel인 사용자 정의 항목은 문서 모델로 분류된 시스템에 정의되어야 하고 SItemTypeCreationICM API 교육 샘플에서 정의한 대로 ICMBASE 및 ICMBASETEXT 부분 유형을 지원해야 합니다.

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_BASETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASETEXT);

// 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();
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SDocModelItemICM.java를 참조하십시오. SResourceItemCreationICM은 XDO 사용 예제를 추가로 표시합니다.

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_Basetext);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
                                                       DK_ICM_SEMANTIC_TYPE_Basetext);

// 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();
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SDocModelItemICM.java를 참조하십시오. SResourceItemCreationICM은 XDO 사용 예제를 추가로 표시합니다.

독립형 XDO 프로그래밍

다음 예제는 모두 Content Manager 8.2에만 해당합니다. 예를 들어, 이전 Content Manager 및 기타 콘텐츠 서버의 경우, 160 페이지의 『DDO를 사용한 항목 표현』, 275 페이지의 『기타 콘텐츠 서버에 대한 작업』 및 CMBROOT\Samples 디렉토리의 샘플 프로그램을 참조하십시오.

버퍼에서 XDO 추가

이 예제에서는 Content Manager의 버퍼에서 XDO를 추가하는 방법을 보여줍니다. 그러면 XDO를 작성하고 메모리로 콘텐츠를 로드하며 메모리에서 콘텐츠를 지속적으로 저장합니다.

요구사항: 분류 자원의 사용자 정의 항목 유형 S_lob는 SItemTypeCreationICM API

교육 샘플에서 정의한 대로 시스템에 정의되어야 합니다. 또한 자원 관리자 및 SMS 컬렉션 정의는 SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM 및 SSMSCollectionDefSetDefaultICM 샘플에서 수행한 대로 사용자 또는 항목 유형의 기본값으로 설정되어야 합니다.

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();
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SResourceItemCreationICM을 참조하십시오.

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();
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SResourceItemCreationICM을 참조하십시오.

파일에서 XDO 추가

다음 예제에서는 XDO를 Content Manager의 콘텐츠 서버(파일에서 직접 콘텐츠 저장)에 추가합니다.

요구사항: 분류 자원의 사용자 정의 항목 유형 S_lob는 SItemTypeCreationICM API 교육 샘플에서 정의한 대로 시스템에 정의되어야 합니다. 또한 자원 관리자 및 SMS 컬렉션 정의는 SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM,

SResourceMgrDefSetDefaultICM 및 SSMSCollectionDefSetDefaultICM 샘플에서 수행한 대로 사용자 또는 항목 유형의 기본값으로 설정되어야 합니다.

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");
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SResourceItemCreationICM을 참조하십시오.

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");
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SResourceItemCreationICM을 참조하십시오.

XDO에 주식 오브젝트 추가

다음 예제에서는 Content Manager의 문서에 주식 부분을 추가합니다.

요구사항: 사용자 정의 항목 유형 S_docModel은 문서 모델로 분류된 시스템에 정의되어야 하고 SItemTypeCreationICM 샘플에서 정의한 대로 ICMANNOTATION 부분 유형을 지원해야 합니다. ddoDocument 변수에 이미 지속적으로 저장된 문서의 인스턴스를 지정했다고 가정하십시오. 또한 변수 dsICM에 연결된 DKDatastoreICM 오브젝트를 지정했다고 가정하십시오.

Java

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

// Create annotation part
DKLobICM annot = (DKLobICM) dsICM.createDDO("ICMANNOATATION",
                                             DKConstant.ICM_DK_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);
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SDocModelItemICM을 참조하십시오.

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);
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SDocModelItemICM을 참조하십시오.

XDO에 대한 작업 예제

다음 예제에서는 독립형 XDO의 사용 방법을 보여줍니다.

XDO 검색, 갱신 및 삭제

컨텐츠 서버에서 오브젝트를 검색, 갱신 또는 삭제하려면 오브젝트를 나타내는 XDO에 대해 올바른 항목 ID, 부분 ID 및 RepType을 제공하십시오.

다음 예제에서는 Content Manager의 XDO를 검색, 갱신 및 삭제합니다.

요구사항: 분류 자원의 사용자 정의 항목 유형 S_text는 SItemTypeCreationICM API 교육 샘플에 정의한 대로 시스템에 정의되어야 합니다. 또한 자원 관리자 및 SMS 콜렉션 정의는 SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM 및 SSMSCollectionDefSetDefaultICM 샘플에서 수행한 대로 사용자 또는 항목 유형의 기본값으로 설정되어야 합니다. 또는 컨텐츠 서버에 이미 존재하는 자원 항목에 대해 변수 pidString에 PID 문자열을 지정했다고 가정하십시오.

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();
```

이 코드 샘플은 CMBROOT\Samples\java\icm 디렉토리의
SResourceItemRetrievalICM, SResourceItemUpdateICM 및
SResourceItemDeletionICM에서 파생됩니다.

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);
```

이 코드 샘플은 CMBROOT\Samples\cpp\icm 디렉토리의
SResourceItemRetrievalICM, SResourceItemUpdateICM 및
SResourceItemDeletionICM에서 파생됩니다.

XDO 함수 호출

이 예제에서는 이전 Content Manager 서버를 사용하여 DKBlob 클래스를 테스트하는 방법을 보여줍니다. 이 예제의 경우 XDO의 항목 ID와 부분 ID를 알아야 합니다.

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(계속)

```
// ----- 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("  affiliatedTyp=" + 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++(계속)

```
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++(계속)

```
        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;
}
```

이전 Content Manager의 XDO 미디어 오브젝트 추가

추가된 모든 미디어 오브젝트에 대해 FRN\$MEDIA 테이블에 하나의 항목이 작성됩니다. 이 항목은 미디어 사용자 데이터에 대한 정보를 포함합니다. 실제 미디어 오브젝트는 네트워크 테이블에 지정된 VideoCharger 콘텐츠 서버에 저장됩니다. 다음 예제의 경우 XDO의 항목 ID를 알아야 합니다.

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(계속)

```
// ----- 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 = "K1A04EWBVHJAV1D7";
    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++(계속)

```
// *** 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 미디어 오브젝트 삭제

다음 예제에서는 XDO 미디어 오브젝트를 삭제하는 방법을 보여줍니다. 이 예제의 경우 XDO의 항목 ID, 부분 ID 및 RepType(표현 유형)을 알아야 합니다.

Java

```
public class txdoDelVSDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 45;
        String repType = "";
        String itemId = "K1A04EBVHJAV1D7";
        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(계속)

```
        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++(계속)

```
        if (flag2)
        {
            DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
                axdo->getExtension("DKMediaStreamInfoDL");
            cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
            cout<<" mediaType="<<mediaInfo->getMediaCopyRate()<<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 미디어 오브젝트 검색

다음 예제에서는 XDO 미디어 오브젝트를 검색하는 방법을 보여줍니다. 검색된 오브젝트는 미디어 오브젝트 대신 미디어 메타데이터만 포함합니다. 이 예제의 경우 XDO의 항목 ID와 부분 ID를 알아야 합니다.

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_DL_INDEXED_OBJECT));
            }
        }
    }
}
```

Java(계속)

```
        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++(계속)

```
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++(계속)

```
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;
}
```

기억영역 콜렉션에 XDO 추가

사용자 정의 기억영역 콜렉션 이름과 연관된 XDO 오브젝트를 추가하려면 확장자 오브젝트 `DKStorageManageInfoxx`를 사용하십시오. 여기서 `xx`는 특정 서버를 나타내는 접미부입니다.

이전 Content Manager 서버의 경우 다음 예제에서는 `DKStorageManageInfoDL`을 사용합니다. Content Manager 버전 8 이상의 경우 145 페이지의 『Content Manager 버전 8.2에 대한 작업』을 참조하십시오.

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("TESTMG1");                //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("TESTMG1");                //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");
```

Content Manager에 검색 색인화 오브젝트 및 미디어 오브젝트를 추가하는 방법에 대한 예제는 CMBROOT\Samples 디렉토리의 다음 코드 샘플을 참조하십시오.

- TxdoAddBsmsDL
- TxdosAddBsmsDL
- TxdoAddFsmsDL
- TxdosAddFsmsDL
- TxdomAddsmsDL

XDO의 기억영역 콜렉션 변경

기존 XDO의 기억영역 콜렉션을 변경할 수 있습니다. 확장자 오브젝트 DKStorageManageInfoICM을 설정한 후 changeStorage 메소드를 호출하십시오.

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.....");
```

이 예제가 수행된 완전한 샘플 응용프로그램(TxdoChgSmsICM)은 CMBROOT\Samples\java\d1 또는 CMBROOT\Samples\cpp\d1 디렉토리에 있습니다.

XML에 대한 작업(Java 전용)

Enterprise Information Portal은 Java API를 사용하여 DDO 및 XDO로서 XML 문서에서 콘텐츠를 Content Manager로 가져오거나 외부로 내보내는 것을 지원합니다. 이 기능을 사용하면 각 시스템에 대해 별도의 인터페이스를 개발하지 않고 서로 다른 정보 시스템에서 데이터 또는 멀티미디어 콘텐츠와 같은 다양한 Content Manager 오브젝트를 가져오기, 저장 및 검색할 수 있습니다. 예를 들어, 한 데이터 시스템에 오브젝트가 저장되어 있는 경우, Enterprise Information Portal의 Java API를 사용하여 이 오브젝트를 XML 파일로 변환한 다음 Content Manager로 가져올 수 있습니다. Content Manager로 가져온 오브젝트에 대해서는 다른 Content Manager 오브젝트에 수행할 수 있는 작업을 모두 수행할 수 있습니다.

XML을 가져오고 내보내는 성능 확장

현재 XML 가져오기 및 내보내기 함수인 `DKDDO.toXML()` 및 `DKDDO.fromXML()`은 참조 속성값, 링크 또는 폴더 콘텐츠에 항목을 지원하지 않습니다. 가져오기 조작이 성공해도 특정 콘텐츠 서버의 항목에 대한 참조로 PID를 사용합니다. 그런 다음 다른 콘텐츠 서버로 가져온 경우, 이들 항목은 PID 정보가 발생한 콘텐츠 서버에 고유하기 때문에 목표 시스템에 존재하지 않습니다. 이러한 두 개의 인터페이스만이 새 항목을 작성하고 같은 항목의 여러 버전을 지원하지 않습니다.

이를 해결하려면 별도의 도구를 빌드하여 참조 항목을 처리하고, 기존 항목 또는 인터페이스가 지원하지 않는 기타 기능을 발견 및 겹쳐쓰기해야 합니다.

버전 8.1 수정팩 1 및 EIP의 버전 8.2는 새 샘플 도구(`TImportICM` 및 `TExportICM`)와 모든 기능을 갖춘 내보내기 및 가져오기 기능을 수행하는 샘플 도구 API(`TExportPackageICM`)를 제공합니다. 새 샘플 기능은 `SItemXMLImportExportICM`에서 설명한 `DKDDO.toXML()` 및 `DKDDO.fromXML()` 인터페이스를 대폭 확장합니다.

새 도구는 Java 샘플인 버전 8.1 수정팩 1 및 버전 8.2와 함께 패키지로 제공됩니다. 자세한 정보는 샘플 가져오기/내보내기 도구 및 API 문서와 `CMBROOT\Samples\java\icm` 디렉토리에 있는 `TExportPackageICM.doc`을 참조하십시오.

XML 문서 가져오기

표준 입력, 파일, 버퍼 및 웹 주소(URL)를 포함한 다른 원본에서 XML을 가져올 수 있습니다. 또한 XML 파일 전체를 가져올 수도 있습니다. 이러한 구성자는 XML 문서에서 콘텐츠를 추출하여 해당 `DKDDO` 및 이와 연관된 모든 `dkXDO`를 작성합니다. 그러면 `DDO`에서 추가 메소드를 호출하여 오브젝트를 Content Manager에 추가할 수 있습니다. 새 `DDO`는 Content Manager 버전 8 항목 유형 또는 이전 Content Manager 색인 클래스에 속하며 Content Manager에만 저장할 수 있습니다. 자체 참조 XML 파일을 가져오면 원래 XML 파일을 `XDO`로 저장할 수 있습니다. 즉, XML 자체를 나중에 사용할 수 있도록 만들어 가져오기 프로세스를 수행해도 XML이 유실되지 않습니다.

XML에서 콘텐츠를 가져오는 경우 `DKDDO`에서 다음 메소드를 사용하십시오.

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

이전 Content Manager와 함께 `DKDDO` 구성자를 사용하여 XML을 가져오면 거부됩니다.

XML 콘텐츠를 가져올 때 다음 매개변수를 기억하십시오.

1. Content Manager 또는 이전 Content Manager로만 가져올 수 있음을 유념하십시오.

2. 가져올 콘텐츠가 포함된 XML 파일은 아래와 같은 XML 문서 유형 정의를 따라야 합니다.
3. XML 가져오기 및 XML 내보내기는 Java API에 의해서만 지원됩니다

다음 절에서는 XML 콘텐츠를 가져오기 위한 전제조건 및 메소드에 대해 설명합니다.

- XML 문서 유형 정의(DTD)
- XML 문서에 콘텐츠 저장
- 다른 XML 원본에서 콘텐츠 추출
- XML 콘텐츠를 Content Manager로 가져오기

XML 문서 유형 정의(DTD)

컨텐츠를 Content Manager로 가져와 XML로서 저장하려면

CMROOT\samples\java\d1\ddo.dtd에 있는 ddo.dtd를 따르는 XML 문서에 콘텐츠를 저장해야 합니다.

```
<!ELEMENT ddo (pid?, pidIdStrings*, propertyCount?, property*, dataCount?, dataItem*, xdoValue?)>
<!-- 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
-->
<!-- pid EMPTY
      dsType CDATA #IMPLIED
      dsName CDATA #IMPLIED
      objectType CDATA #IMPLIED
      pidString CDATA #IMPLIED
-->
<!-- pidIdStrings EMPTY
      idPosition CDATA #REQUIRED
      idValue CDATA #REQUIRED
-->
<!-- propertyCount (#PCDATA)
      propertyId CDATA #IMPLIED
      propertyName CDATA #IMPLIED
      propertyValue CDATA #IMPLIED
      propertyType CDATA #IMPLIED
-->
<!-- dataCount (#PCDATA)
      dataItem (dataPropertyCount?, dataProperty+, (dataValue | dataValues)*)
-->
<!-- dataItem
      dataId CDATA #IMPLIED
      dataName CDATA #REQUIRED
      dataNameSpace CDATA #IMPLIED
-->
<!-- dataPropertyCount (#PCDATA)
      dataProperty EMPTY
-->
<!-- dataProperty
      propertyId CDATA #IMPLIED
      propertyName CDATA #IMPLIED
      propertyValue CDATA #IMPLIED
      propertyType CDATA #IMPLIED
-->
<!-- dataValues (dataValueCount?, dataValue*)
      collectionName CDATA #IMPLIED
-->
<!-- dataValueCount (#PCDATA)
      dataValue (#PCDATA | ddo | xdoRef | linkRef)*
-->
<!-- linkRef (linkSource, linkTarget, linkItem?)
      linkRef linkTypeName CDATA #REQUIRED
-->
<!-- linkSource (ddo)
      linkSource sameAsParentDDO (yes | no) "no"
-->
<!-- linkTarget (ddo)
      linkTarget sameAsParentDDO (yes | no) "no"
-->
<!-- linkItem (ddo)
      xdoRef (xdoPid, xdoIdStrings*, xdoValue)
-->
<!-- partId deprecated it is replaced by xdoIdString on an xdoRef -->
<!-- repType deprecated it is replaced by xdoIdString on an xdoRef -->
<!-- xdoPid EMPTY
      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 문서에 콘텐츠 저장

XML 파일은 Content Manager로 가져올 문서나 폴더를 여러 가지 방식으로 나타낼 수 있습니다. 이러한 문서 및 폴더는 부분을 포함할 수도 있습니다. 아래 샘플에는 먼저 일반적인 XML 데이터 항목인 dataItem dataId="1"이 표시되어 있으며, 이 값은 Basuki입니다. 그러나 DataItem 13은 dataName DKParts를 사용하며, 이는 자체 참조 XDO에 관련되어 있습니다.

Content Manager에 대한 샘플

```

<?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"/>
    <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>

```

```

        </ddo>
    </dataValue>
</dataValues>
</dataItem>
    <xdoValue mimeType="text/plain" refType="file"
        XML-LINK="SIMPLE" HREF="TSophiaBefore.txt" >
    </xdoValue>
</ddo>

```

XML이 Content Manager를 사용하여 오브젝트를 저장하는 방법에 대한 예제는 DK\Samples\java\icm\ 디렉토리에 있는 샘플을 참조하십시오.

이전 Content Manager에 대한 샘플

```

<?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>

```

XML이 이전 Content Manager를 사용하여 오브젝트를 저장하는 방법에 대한 예제는 sample 디렉토리에 있는 다음 코드 샘플을 참조하십시오.

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

다른 XML 원본에서 콘텐츠 추출

DKDDO 메소드는 표준 입력, 파일, 버퍼 및 웹 주소(URL)를 포함한 다양한 XML 원본에서 콘텐츠를 추출할 수 있습니다. XML 원본에서 콘텐츠를 추출한 다음 가져오기 프로세스를 시작하려면 DKDDO 메소드를 호출하십시오.

다음은 각 XML 원본에 대한 예제입니다.

파일의 XML:

Java

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

버퍼의 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;
```

웹 주소(URL)의 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 Manager로 XML 콘텐츠 가져오기

다음 예제는 아래의 기본 단계를 따릅니다.

1. DKDDO를 작성합니다.
2. 콘텐츠 서버(이 경우에는 Content Manager 또는 이전 Content Manager)를 작성 및 연결합니다.
3. 새 DKDDO를 콘텐츠 서버(이 경우에는 Content Manager)에 추가합니다.
4. dkDataObjectBase fromXML(DKNVPair xmlSource) 또는 dkDataObjectBase fromXML(DKNVPair xmlSource, int options)을 사용하여 XML 원본을 가져옵니다.

이렇게 해서 생성된 DKDDO는 ddo.dtd 스펙을 따르며 Content Manager 항목 유형 또는 이전 Content Manager 색인 클래스에 속합니다.

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);
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SitemXMLImportExportICM.java를 참조하십시오.

XML 내보내기

Content Manager 또는 이전 Content Manager로부터 XML 문서로서 DDO 및 XDO 데이터를 내보낼 수 있습니다. DKDDO의 toXML(DKNVPair xmlDestination, String path, int options) 메소드를 사용하여 내보내십시오.

제한사항: 이 샘플에 설명된 DKDDO.toXML() 및 DKDDO.fromXML() 인터페이스는 다른 항목을 링크, 폴더 콘텐츠 또는 참조 속성을 통해 참조하는 경우 하나의 콘텐츠 서버에서 항목을 내보내어 다른 콘텐츠 서버로 가져오는 데 사용할 수 없습니다. 가져오기 조작은 새 항목을 작성하지만 기존 항목을 갱신 또는 겹쳐쓰기하거나 항목의 여러 버전을 지원하지 않습니다. 이 성능을 확장하는 도구에 대한 정보는 92 페이지의 『XML을 가져오고 내보내는 성능 확장』을 참조하십시오.

다음 예제에서는 XML을 내보내는 방법을 보여줍니다.

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);
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SItemXMLImportExportICM.java를 참조하십시오.

문서 작성 및 DKPARTS 속성 사용

DDO의 DKPARTS 속성은 문서의 부분인 컬렉션을 나타냅니다. 이 속성값은 DKParts 오브젝트이며, 이는 DKPart 오브젝트의 컬렉션입니다. DKPart 오브젝트는 항목 유형을 분류한 문서 부분의 항목이며 자원 콘텐츠를 포함합니다.

Content Manager 전용: 문서는 Content Manager 시스템과 사용자 사이에서 별도 단위로 저장, 검색 및 교환할 수 있는 항목입니다. *document* 의미 유형이 있는 항목은 문서 형성 정보를 포함하는 것으로 예상되지만 엄밀히 특정 문서 모델의 구현을 의미하는 것은 아닙니다. 항목 유형을 분류할 문서(문서 모델이라고도 함)에서 작성된 항목은 그 항목이 Content Manager에서 제공된 문서 모델을 특정 구현하는 문서 부분을 포함한다는 것을 의미합니다. 항목 유형을 분류한 문서는 문서 또는 폴더 의미 유형이 있는 항목을 작성할 수 있습니다. 문서 부분은 텍스트, 이미지 및 스프레드시트를 비롯하여 다양한 유형의 콘텐츠를 포함할 수 있습니다.

다음 예제에서는 문서를 작성하고 Content Manager에 문서 부분을 추가합니다.

요구사항: 사용자 정의 항목 유형인 S_docModel은 문서 모델로 분류된 시스템에 정의되어야 합니다. 또한 SItemTypeCreationICM API 교육 샘플에 정의한 대로 ICMBASE 및 ICMBASETEXT 부분 유형도 지원합니다.

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_BASETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASETEXT);

// 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();
```

부분이 있는 문서 작성에 대한 자세한 정보는 CMBROOT\Samples\java\icm 디렉토리의 SDocModelItemICM API 교육 샘플을 참조하십시오.

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();
```

부분이 있는 문서 작성에 대한 자세한 정보는 CMBROOT\Samples\cpp\icm 디렉토리의 SDocModelItemICM API 교육 샘플을 참조하십시오.

DDO는 부분 컬렉션의 모든 부분을 소유합니다. 문서 DDO를 통해 부분을 갱신 및 삭제하십시오.

다음 예제에서는 DDO에서 부분을 검색 및 액세스하는 방법을 보여줍니다.

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());
}
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SDocModelItemICM을 참조하십시오.

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
```

부분이 있는 문서 작성에 대한 자세한 정보는 CMBROOT\Samples\cpp\icm 디렉토리의 SDocModelItemICM API 교육 샘플을 참조하십시오.

폴더 작성 및 DKFOLDER 속성 사용

DDO 폴더에서 DKFOLDER 속성을 사용하여 폴더에 속한 다른 폴더 및 문서의 컬렉션을 나타냅니다. 이 속성값은 DKFolder 오브젝트이며, DDO 컬렉션입니다. 이는 아래 표시된 대로 DKFolder 속성은 DKParts 속성이 설정된 것처럼 설정됩니다.

Content Manager 전용: 폴더는 분류에 상관없이 *folder* 의미 유형이 있는 모든 항목 유형의 항목입니다. 폴더 의미 유형이 있는 항목은 Content Manager에서 제공한 특정 폴더 기능뿐 아니라 자원이 아닌 모든 항목 성능 및 항목 유형 분류(예: 문서 모델 또는 자원)에서 사용 가능한 추가 기능을 포함합니다. 폴더에는 문서 및 하위 폴더를 비롯한 모든 유형의 항목이 포함되어 있을 수 있습니다. 폴더는 속성으로 색인화됩니다.

다음 예제에서는 폴더를 작성하고 Content Manager에 콘텐츠를 추가합니다.

요구사항: 사용자 정의 항목 유형인 S_simple은 SItemTypeCreationICM API 교육 샘플에 정의한 대로 시스템에 정의되어야 합니다.

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();
```

폴더 작성에 대한 자세한 정보는 CMBROOT\Samples\java\icm 디렉토리의 SFolderICM API 교육 샘플을 참조하십시오.

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();
```

폴더 작성에 대한 자세한 정보는 CMBROOT\Samples\cpp\icm 디렉토리의 SFolderICM API 교육 샘플을 참조하십시오.

Content Manager 버전 8에서 폴더 항목은 폴더 콘텐츠를 소유하지 않습니다. 항목은 여러 폴더에 추가될 수도 있습니다. 폴더에서 항목을 제거하려면 시스템이 관리하는 폴더-콘텐츠 관계를 간단히 구분하면 됩니다. 폴더의 항목은 독립적으로 갱신 및 삭제되어야 합니다. Content Manager의 이전 버전에서 DDO는 컬렉션에 콘텐츠를 소유합니다.

다음 예제에서는 DDO에서 폴더 콘텐츠를 검색 및 액세스하는 방법을 보여줍니다.

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());
}
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\icm 디렉토리의 SFolderICM 교육 샘플을 참조하십시오.

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;
}
```

완전한 샘플 응용프로그램에 대해서는 CMBROOT\Samples\cpp\icm 디렉토리의 SFolderICM 교육 샘플을 참조하십시오.

DKAny 사용(C++ 전용)

DKAny는 런타임 시 유형이 변경될 수 있는 오브젝트를 포함합니다. DKAny 오브젝트는 다음 유형이 될 수 있습니다.

- null
- (부호가 없는) short

- (부호가 없는) long
- double
- char
- TypeCode
- DKBoolean
- DKString
- DKDate
- DKTime
- DKTimestamp

DKAny는 값이 할당되지 않은 경우에는 널(NULL)만이 될 수 있습니다(DKAny any).
값이 할당된 후에는 DKAny::isNull()은 FALSE를 리턴합니다.

DKAny 오브젝트는 위의 유형 뿐만 아니라 다음과 같은 오브젝트 참조 유형도 포함할 수 있습니다.

- dkDataObjectBase*
- dkCollection*
- void*

유형 코드 사용

typeCode 함수를 호출하여 DKAny 오브젝트의 현재 유형을 결정할 수 있습니다. 이 함수는 TypeCode 오브젝트, 즉 널(null)인 경우에는 tc_null, short인 경우에는 tc_short를 리턴합니다. 전체 유형 코드 목록을 보려면 온라인 API 참조서를 참조하십시오.

DKAny에서 메모리 관리

포함된 오브젝트가 오브젝트 참조 유형이 아닌 경우, DKAny는 DKAny에 포함된 오브젝트의 메모리를 관리합니다. 오브젝트 참조를 포함하는 복사 관련 조작은 포인터의 사본만 작성합니다. 복사 및 삭제 중 오브젝트 참조 유형을 계속 추적해야 합니다.

구성자 사용

DKAny는 지원하는 각 유형에 구성자를 제공합니다. 다음 예제에서는 이전 절에 나열된 몇몇 유형을 포함하는 DKAny 오브젝트를 작성하는 방법을 보여줍니다.

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
```

유형 코드 가져오기

DKAny 내부에서 오브젝트 유형 코드를 찾으려면 `typeCode` 함수를 사용하십시오.

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
```

DKAny에 새 값 할당

새 값을 기존의 DKAny 오브젝트에 할당하려면 등호(=) 할당 연산자를 사용하십시오. DKAny는 각 유형 코드에 할당을 제공합니다.

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
```

DKAny에서 값 할당

DKAny를 일반 유형에 다시 할당하려면 캐스트 연산자가 필요합니다. 예를 들면, 다음과 같습니다.

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
```

유형이 일치하지 않으면 올바른지 않은 유형 변환 예외가 발생합니다. 따라서 DKAny를 일반 유형으로 변환하려면 먼저 유형 코드를 점검해야 합니다.

C++

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

다음과 같이 case문을 작성하여 DKAny의 유형을 점검할 수 있습니다.

C++

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

DKAny 오브젝트에 오브젝트 참조가 포함되어 있으면 void 포인터로 DKAny 콘텐츠를 가져온 후 이 콘텐츠를 적절한 유형으로 캐스트할 수 있습니다. 그러나 DKAny 내부에 사용된 유형 코드를 아는 경우에만 이 연산을 사용하십시오.

C++

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

DKAny 표시

cout를 사용하여 DKAny 오브젝트의 콘텐츠를 표시할 수 있습니다.

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 제거

DKAny가 오브젝트 참조를 보유할 수는 있지만 오브젝트 참조 유형에 대한 메모리는 관리할 수 없으므로 다음 유형의 메모리를 관리해야 합니다. 다음 예제에서는 DKAny 오브젝트의 메모리를 관리합니다.

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
```

범위를 종료하기 전에 마지막 delete문이 수행되어야 합니다. 그렇지 않으면 anyA가 삭제되고 DDO는 메모리 누출로 남게 됩니다.

프로그래밍 팁

권장사항: 정수 리터럴을 DKAny로 변환할 때 원하지 않는 유형 변환이 발생하지 않도록 하려면 명시적으로 유형을 지정하는 것이 바람직합니다. 다음과 같이 설정하십시오.

C++

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


컬렉션 및 반복자 사용

dkCollection은 컬렉션에 대한 작업에 필요한 메소드를 제공하는 추상 클래스입니다. DKSequentialCollection은 dkCollection의 구체적인 구현입니다. 다른 컬렉션은 DKSequentialCollection의 서브클래스로 구현됩니다. 이러한 컬렉션은 구성원으로 데이터 오브젝트를 포함합니다.

컬렉션 구성원은 일반적으로 동일한 유형의 오브젝트입니다. 그러나 컬렉션은 다른 유형의 구성원을 포함할 수 있습니다.

C++ 전용: 새 구성원이 추가되면 해당 컬렉션은 이 구성원을 소유합니다. 구성원을 검색할 때 컬렉션 내부의 DKAny 오브젝트로 포인터를 가져옵니다. 이 오브젝트는 컬렉션에 속합니다. 이것은 컬렉션이 DKAny 구성원의 메모리를 관리함을 의미합니다. DKAny 오브젝트는 오브젝트 참조를 보유할 수 있지만 오브젝트 참조 유형에 대한 메모리를 관리할 수 없으므로 이러한 유형의 메모리를 관리해야 합니다.

순차 컬렉션 메소드 사용

DKSequentialCollection은 구성원을 추가, 검색, 제거 및 대체할 수 있는 메소드를 제공합니다. 또한 정렬 메소드도 갖고 있습니다. 다음 예제에서는 컬렉션에 새 구성원을 추가하는 방법에 대해 설명합니다(addElement 메소드는 매개변수로 오브젝트를 사용함).

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
```

순차 반복자 사용

반복자를 사용하여 컬렉션 구성원에 대해 반복을 수행합니다. API는 두 가지 유형의 반복자 dkIterator 및 DKSequentialIterator를 가지고 있습니다.

Java

기본 반복자인 `dkIterator`는 다음, 추가 및 재설정 메소드를 지원합니다. 서브클래스 `DKSequentialIterator`는 추가 메소드를 포함합니다. 컬렉션에서 `createIterator` 메소드를 호출하여 반복자를 작성합니다. 반복자 작성 후 `setToFirst()` 메소드를 사용하여 첫 번째 항목을 지정하십시오. 다음 예제에서는 반복자를 사용하는 방법을 보여줍니다.

```
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++

컬렉션 구성원에 대해 메소드를 반복할 수 있게 하는 반복자가 제공됩니다. 반복자 유형에는 `next`, `more` 및 `reset` 함수를 지원하는 기본 반복자인 `dkIterator`와 더 많은 함수를 포함하는 서브클래스인 `DKSequentialIterator`의 두 가지가 있습니다. 컬렉션에서 `createIterator` 함수를 호출하여 반복자를 작성합니다. 이 함수는 새 반복자를 작성하고 이 반복자를 사용자에게 리턴합니다. 다음 코드를 사용하여 컬렉션에 대해 메소드를 반복하십시오.

```
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`는 반복자를 양방향으로 이동하기 위한 추가 메소드를 제공합니다. 이전 예제는 다음과 같이 다시 작성될 수 있습니다.

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;
```

이 코드는 다음 구성원으로 이동하기 전에 현재 구성원에 대해 몇 가지 연산을 수행할 수 있게 합니다. 이러한 연산은 구성원을 새 구성원으로 대체하거나 제거할 수 있습니다.

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
...
```

팁: 현재 구성원을 제거하면 반복자는 다음 구성원으로 넘어갑니다. 루프 내에서 구성원을 제거할 경우 다음 예제에서와 같이 점검하십시오. 현재 구성원을 제거한 후 다음 구성원을 건너뛰지 않으려면 제거 조건을 점검해야 합니다.

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
```

컬렉션의 메모리 관리(C++ 전용)

컬렉션은 DKAny 오브젝트인 구성원에 대한 메모리를 관리합니다. DKAny 오브젝트 관리 규칙과 동일한 규칙이 여기에도 적용되는데, DKAny 내부의 오브젝트가 오브젝트 참조 유형이면 사용자는 다음의 경우 메모리를 관리해야 합니다.

- 컬렉션 제거.
- 구성원 대체.
- 구성원 대체.

이 예제에서는 이러한 경우에 메모리를 관리하는 방법을 보여줍니다.

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
```

구성원을 삭제하는 대신 이 구성원을 다른 컬렉션에 추가할 수 있습니다.

`replaceElementAt` 및 `removeAllElement` 함수를 사용하기 전에 먼저 유사한 단계를 수행해야 합니다.

컬렉션을 제거하기 전에 해당 구성원을 삭제하십시오. 함수를 기록하여 이 작업을 수행하고 이 함수를 컬렉션의 `apply` 함수로 전달할 수 있습니다. `DKAttributeDef` 오브젝트를 포함하는 `DKAny` 오브젝트 컬렉션이 있다고 가정하십시오. 다음 예에서는 컬렉션을 삭제합니다.

C++

```
DKDatastoreICM dsICM;
...
DKAny any = dsICM.listSchemaAttributes("GRANDPA");
dkCollection* acoll = (dkCollection*) any;
...
acoll->apply(deleteDKAttributeDef);    // use the attributes
delete acoll;                        // deletes all members
```

이 예제에서 `deleteDKAttributeDef`는 매개변수로 `DKAny` 오브젝트를 사용하는 함수입니다. 다음과 같이 정의합니다.

C++

```
void deleteDKAttributeDef(DKAny& any) {
    delete ((DKAttributeDef*) any.value());
    any.setNull();                // good practice
}
```

사용자의 delete 함수를 작성하여 컬렉션을 삭제하거나 컬렉션을 삭제하기 전에 일부 구성원을 제거할 수 있습니다.

DKParts, DKFolder 및 DKResults와 같이 알려져 있는 일부 컬렉션에 대한 제거 프로그램은 다음과 같은 필수 제거 단계를 수행합니다. 그러나 replaceElementAt, removeElementAt 또는 removeAllElement 함수 실행 중에는 기억영역을 관리하지 않습니다.

컬렉션 정렬

지정된 키에 따라 오름차순 또는 내림차순으로 컬렉션 구성원을 정렬하려면 sort 함수를 사용하십시오. 정렬 오브젝트와 원하는 순서를 전달해야 합니다. 정렬 오브젝트를 위한 인터페이스는 dkSort.java(Java) 또는 dkSort.hpp(C++)에 정의됩니다. 사용자는 특정 컬렉션을 정렬할 자체 Sort 함수를 작성할 수 있습니다. 다음 예제에서는 DDO 컬렉션의 정렬 방법에 대해 설명합니다.

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
....
```

팁: 정렬 오브젝트는 스택에 작성되므로 명시적으로 삭제할 필요가 없습니다. 함수는 재 진입하는데, 이것은 단일 사본이 공유되거나 재사용되거나 또는 다른 함수로 전달될 수 있음을 의미합니다.

연합 컬렉션 및 반복자 이해

조회로부터 생성된 데이터 오브젝트를 컬렉션으로 처리하려면 응용프로그램에서 연합 컬렉션을 사용하십시오. 연합 컬렉션은 데이터 오브젝트 간에 존재하는 하위 그룹화 관계를 보존합니다.

의 첫 번째 요소입니다. 이 시점에서 `setToNextCollection` 함수가 호출되면 반복자 위치를 두 번째 `DKResults` 컬렉션의 첫 번째 DDO로 설정합니다.

`dkFederatedIterator`의 `setToLastCollection` 함수는 반복자 위치를 `DKFederatedCollection`의 마지막 DDO로 설정합니다. 이 경우 반복자의 위치는 마지막 `DKResults` 컬렉션 오브젝트의 마지막 요소입니다. `setToPreviousCollection` 함수가 호출되면 반복자 위치를 이전 `DKResults` 컬렉션의 마지막 DDO로 설정합니다.

컨텐츠 서버 조회

`dkResultSetCursor` 또는 `DKResults` 오브젝트에서 컨텐츠 서버를 검색하고 결과를 수신할 수 있습니다. 몇몇 서버의 경우, 조회 오브젝트를 작성하여 조회를 나타낸 후 조회 오브젝트의 `execute` 또는 `evaluate` 함수를 호출할 수 있습니다. 이 컨텐츠 서버의 도움말을 사용하여 조회 오브젝트가 조회 준비와 실행, 조회 실행 상태 모니터 및 결과 저장과 같은 조회 처리 작업을 수행합니다. 일부 컨텐츠 서버는 조회 오브젝트를 대안으로 사용하는 것을 지원합니다. 이전 Content Manager 및 연합 컨텐츠 서버는 이들 중 두 개입니다.

조회 오브젝트를 지원하는 컨텐츠 서버에는 매개변수식, 텍스트, 이미지 및 결합의 네 가지 유형의 조회 오브젝트가 있습니다. 결합된 조회는 텍스트 및 매개변수식 조회로 구성되어 있습니다. 모든 컨텐츠 서버가 결합된 조회를 수행할 수 있는 것은 아닙니다. 이전 Content Manager는 이미지 조회를 지원합니다.

Content Manager에서 매개변수식 조회와 텍스트 조회는 통합되어 있습니다. 조회 오브젝트를 사용해서는 안 됩니다. Content Manager에서 조회하는 방법에 대한 정보는 211 페이지의 『Content Manager 서버 조회』를 참조하십시오. 이전 Content Manager 서버 조회에 대한 정보는 *다른 컨텐츠 서버에 대한 작업을 참조하십시오.*

컨텐츠 서버는 조회를 실행하기 위해 두 가지 메소드(`execute` 및 `evaluate`)를 사용합니다. `execute` 함수는 `dkResultSetCursor` 오브젝트를 리턴하고 `evaluate` 함수는 `DKResults` 오브젝트를 리턴합니다. `dkResultSetCursor` 오브젝트는 결과 세트 커서의 현재 위치에서 큰 결과 세트를 처리하고 `delete`와 `update` 함수를 수행하는 데 사용됩니다. `fetchNextN` 함수를 사용하여 컬렉션에 대한 오브젝트 그룹을 검색할 수 있습니다.

`dkResultSetCursor`는 닫기 및 열기 메소드를 호출하여 조회를 재실행하는 데 사용될 수도 있습니다. 자세한 정보는 138 페이지의 『결과 세트 커서 사용』에 설명되어 있습니다.

`DKResults`에는 모든 조회 결과가 포함됩니다. 컬렉션의 항목에 대해 전방향 또는 역방향으로 반복할 수 있으며 컬렉션을 조회하거나 컬렉션을 다른 조회의 범위로 사용할 수 있습니다.

자세한 정보는 138 페이지의 『조회를 재실행하기 위해 결과 세트 커서 열기 및 닫기』를 참조하십시오.

제한사항: Domino.Doc 콘텐츠 서버를 조회하는 경우, DKResults 오브젝트가 리턴됩니다. 그러나 이 오브젝트를 조회하거나 다른 조회의 범위로 사용할 수는 없습니다.

dkResultSetCursor와 DKResults 간의 차이

dkResultSetCursor 및 DKResults 콜렉션은 다음과 같은 차이점을 가지고 있습니다.

- dkResultSetCursor는 콘텐츠 서버 커서처럼 작동하며, 포함된 DKDDO가 한번에 하나씩 페치되므로 큰 결과 세트에 사용할 수 있습니다. 또한 다시 조회를 실행하는 데 사용하여 최신 결과를 얻을 수도 있습니다.

제한사항: dkResultSetCursor를 사용하더라도 Domino.Doc 콘텐츠 서버에서 조회를 재실행할 수 없습니다.

- DKResults는 전체 결과 세트를 포함하고 양방향 반복자를 지원합니다.
- 긴 시간 동안 dkResultSetCursor를 열어 두면 일부 콘텐츠 서버에서 동시 사용자 성능이 저하될 수 있습니다.

매개변수식 조회 사용

매개변수식 조회는 조회 술어에 지정된 조건과 콘텐츠 서버에 저장된 데이터 값이 정확하게 일치해야 하는 조회입니다.

주: 다음 절의 조회 예제는 이전 Content Manager에 적용됩니다. Content Manager V8에 대한 조회 정보는 211 페이지의 『조회 언어 이해』를 참조하십시오.

매개변수식 조회 문자열 구성

조회를 작성하려면 먼저 조회 문자열을 구성해야 합니다. 다음 예제에서 이전 Content Manager에서 GP2DLS2라는 색인 클래스에 대한 조회를 나타내는 조회 문자열이 정의되어 있습니다. Content Manager의 조회 문자열 예제는 220 페이지의 『조회 예제』를 참조하십시오. 조회 조건은 DLSEARCH_DocType 속성이 null이 아닌 모든 문서 또는 폴더를 검색하는 것입니다. 리턴되는 최대 결과 수는 5로 제한되며 문서 또는 폴더의 콘텐츠가 리턴되도록 콘텐츠를 YES로 설정합니다.

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);";
```

이 예제에서는 Content Manager 서버가 이 조회에 대해 동적 SQL을 사용하고 모든 폴더 및 문서를 검색하도록 지정합니다. 서로 다른 콘텐츠 서버는 서로 다른 조회 문자열 구문규칙을 사용합니다. 연합은 자체 조회 문자열 구문규칙을 가지고 있습니다. 자세한 정보는 온라인 API 참조서 또는 검색하고자 하는 콘텐츠 서버 정보를 참조하십시오. 속성 이름이 둘 이상의 단어로 되거나 DBCS 언어로 되어 있으면 어포스트로피(')로 묶어야 합니다. 속성값이 DBCS로 되어 있으면 큰따옴표(")로 묶어야 합니다.

여러 기준에 대한 매개변수식 조회 구성

매개변수식 조회에 대해 둘 이상의 검색 기준을 지정할 수 있습니다. 다음 예제에서는 이전 Content Manager의 두 개의 색인 클래스에 대해 조회를 지정하는 방법을 보여줍니다.

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);";
```

매개변수식 조회 실행

조회 문자열을 가진 다음 조회 오브젝트를 작성합니다. 콘텐츠 서버를 나타내는 DKDatastorexx에는 조회 오브젝트를 작성하는 메소드가 포함되어 있습니다. 조회 오브젝트를 사용하여 조회를 실행하면 결과가 리턴됩니다. 다음 예제에서는 매개변수식 조회 오브젝트를 작성하고 이전 Content Manager 서버에서 조회를 실행하는 방법을 보여줍니다. Content Manger 버전 8 이상에서 조회 오브젝트를 사용해서는 안됩니다. 조회를 실행하면 DKResults 콜렉션에 결과가 리턴됩니다.

경고: DKResults 오브젝트를 삭제하면 모든 구성원도 삭제됩니다. 요소가 두 번 삭제되지 않도록 주의하십시오.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSamplePQryDL.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSamplePQryDL.cpp)은 Cmbroot/Samples/cpp/dl 디렉토리에 있습니다.

컨텐츠 서버에서 매개변수식 조회 실행

컨텐츠 서버를 나타내는 DKDatastorexx에는 조회를 실행하는 메소드가 포함되어 있습니다. 다음 예제에서는 이전 Content Manager 컨텐츠 서버에서 매개변수식 조회를 실행하는 방법을 보여줍니다. 조회 실행 후 결과는 dkResultSetCursor 오브젝트에 리턴됩니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDL.java)은
CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDL.cpp)은
Cmbroot/Samples/cpp/dl 디렉토리에 있습니다.

컨텐츠 서버에서 매개변수식 조회 평가

컨텐츠 서버를 나타내는 DKDatastorexx에는 조회를 평가하는 메소드가 포함되어 있습니다. 결과는 DKResults 콜렉션에 리턴됩니다. 다음 예제에서는 이전 Content Manager 컨텐츠 서버에서 매개변수식 조회를 평가하는 방법을 보여줍니다.

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\")))";
cmd += "OPTION=(CONTENT=NO;";
cmd += "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();
```

텍스트 조회 사용

Content Manager 버전 8 이상에서 텍스트 조회 및 매개변수식 조회는 통합되어 있습니다. 217 페이지의 『결합된 매개변수식 및 텍스트 검색 작성』을 참조하십시오.

이전 Content Manager에서 텍스트 및 매개변수식 검색을 수행할 수 있습니다. 텍스트 검색은 텍스트 검색 엔진에서 작성한 텍스트 색인을 조회하여 실제 문서 텍스트를 검색합니다.

텍스트 조회 문자열 구성

조회 문자열을 구성하여 텍스트 검색을 시작합니다. 다음 예제에는 TMINDEX 텍스트 색인에 대한 조회를 나타내는 조회 문자열이 작성되어 있습니다. 조회 문자열에는 UNIX® 또는 member라는 단어가 들어 있는 모든 텍스트 문서를 검색하는 기준이 포함됩니다. 리턴되는 최대 결과 수는 5입니다.

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)";
```

여러 색인에 대한 텍스트 조회 구성

텍스트 조회를 사용하여 둘 이상의 색인을 검색할 수 있습니다. 다음 예제에서는 두 개의 색인에 대한 조회를 지정하는 방법을 보여줍니다.

중요사항: 조회에 둘 이상의 텍스트 검색 색인을 지정할 경우, 이 색인은 같은 유형이어야 합니다. 예를 들어, 조회에 두 개의 정밀 색인을 지정할 수 있지만 조회 내에 하나의 정밀 색인과 하나의 언어 색인을 지정할 수는 없습니다.

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)";
```

텍스트 조회 실행

텍스트 조회 문자열을 작성한 다음 조회 오브젝트를 작성합니다. 콘텐츠 서버를 나타내는 DKDatastorexx에는 조회 오브젝트를 작성하는 메소드가 포함되어 있습니다. 결과는 DKResults 콜렉션에 리턴됩니다. 조회 오브젝트를 사용하여 조회를 실행하면 결과가 리턴됩니다. 다음 예제에서는 텍스트 조회 오브젝트를 작성하고 해당 조회를 실행하는 방법을 보여줍니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSampleTQryTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSampleTQryTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

컨텐츠 서버에서 텍스트 조회 실행

컨텐츠 서버를 나타내는 데 사용되는 DKDatastorexx는 조회를 실행하는 메소드를 제공합니다. 결과는 dkResultSetCursor 오브젝트에 리턴됩니다. 다음 예제에서는 이전 Content Manager 서버에서 텍스트 조회를 실행하는 방법을 보여줍니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteTS.java)은
CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteTS.cpp)은
Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

컨텐츠 서버에서 텍스트 조회 평가

컨텐츠 서버를 나타내는 데 사용되는 DKDatastorexx는 조회를 실행하고 DKResults 콜렉션을 리턴하기 위한 평가 메소드를 제공합니다. 다음 예제에서는 이전 Content Manager 컨텐츠 서버에서 텍스트 조회를 평가하는 방법을 보여줍니다.

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();
```

일치 강조표시 정보 얻기

일치 정보는 해당 조회의 모든 일치에 대한 문서의 텍스트 및 강조표시 정보를 포함합니다.

조회 문자열 구성 시 MATCH_INFO 및 MATCH_DICT 옵션을 설정합니다.

MATCH_INFO를 YES로 설정하여 일치 강조표시 정보를 리턴하십시오. MATCH_DICT 옵션은 사전을 사용하여 강조표시 정보를 얻을 수 있는지 여부를 지정합니다. 일치 정보는 텍스트 조회에서 리턴된 DKDDO의 DKMATCHESINFO 속성에 리턴됩니다. DKMATCHESINFO 속성값은 DKMatchesInfoTS 오브젝트입니다.

컨텐츠 서버에서 문서를 검색하고 언어적으로 분석하여 일치 여부를 판별하므로 일치 강조표시 정보를 얻으려면 시간이 많이 걸립니다. 이 프로세스를 실행하면 텍스트 조회의 성능에 영향을 미칩니다.

각 텍스트 조회 결과 항목에 대한 일치 강조표시 정보 얻기: 다음 예제에서는 텍스트 조회 중 각 텍스트 조회 결과 항목에 대한 일치 강조표시 정보를 검색합니다. MATCH_DICT 옵션이 NO로 설정되어 있으므로 사전을 사용하지 않습니다.

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(계속)

```
// ----- 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 Highlighting 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++(계속)

```
// process the Match Hightlighting information
pDOBase = a;
pMInfo = (DKMatchesInfoTS*)pDOBase;
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();
```

특정 텍스트 조회 결과 항목에 대한 일치 강조표시 정보 얻기: 다음 예제에서는 텍스트 조회에서 리턴된 특정 항목에 대한 일치 강조표시 정보를 검색합니다. 이 루틴으로 전달된 `dkResultSetCursor`는 열린 상태에 있어야 합니다.

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TINDEX;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(계속)

```
// 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++(계속)

```
// 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();
```

결과 세트 커서 사용

dkResultSetCursor는 DDO 오브젝트의 가상 컬렉션을 관리하는 콘텐츠 서버 커서입니다. 즉, 컬렉션에서 요소를 페치할 때까지 컬렉션은 구체화되지 않습니다. 컬렉션은 조회 기준에 맞는 항목의 결과 세트입니다. 커서 사용을 완료한 경우, `destroy` 메소드를 호출하여 사용한 메모리를 해제하십시오.

중요사항: 이 절의 정보는 Content Manager 8.2에 적용되지 않습니다. 세부사항은 211 페이지의 『조회 언어 이해』를 참조하십시오.

조회를 재실행하기 위해 결과 세트 커서 열기 및 닫기

결과 세트 커서를 작성하면 열린 상태에 놓여 있습니다. 조회를 재실행하려면 커서를 닫은 다음 다시 여십시오.

Java

```
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES);" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC>";
DKNameValuePair 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();
```

결과 세트 커서의 위치 설정 및 가져오기

결과 세트 커서를 사용하여 현재 위치를 설정 및 가져올 수 있습니다. 다음 예제에서는 조회를 작성 및 실행합니다. `while` 루프 안에서 커서 위치는 첫 번째 또는 그 다음 올

바른 위치로 설정됩니다. 그런 다음 해당 위치에서 DDO를 폐치합니다. 커서가 마지막 항목을 지나면 널(null)이 fetchObject 메소드에서 리턴됩니다.

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;
```

이 작업을 수행하는 다른 방법은 다음과 같습니다.

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;
```

항목을 통해 반복하는 경우 상대적 위치 지정을 사용할 수 있습니다. 다음 예제에서는 결과 세트 커서에 있는 다른 모든 항목을 건너뛵니다.

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;
```

결과 세트 커서에서 컬렉션 작성

결과 세트 커서를 사용하여 결과 세트 커서에 있는 지정된 수의 항목을 컬렉션에 포함시킬 수 있습니다. `fetchNextN` 메소드의 첫 번째 매개변수는 컬렉션에 배치될 항목 수가 얼마나 되는지를 지정합니다. 첫 번째 매개변수에 0을 전달하면 컬렉션에 모든 항목이 배치됨을 나타냅니다.

다음 예제에서 결과 세트 커서의 모든 항목이 순차 컬렉션으로 페치됩니다. `fItems`가 TRUE인 경우, 최소한 하나의 항목이 리턴됩니다.

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);
```

컬렉션 조회

조회 가능 컬렉션은 추가 조회가 가능한 컬렉션으로, 더 작은 세트와 세분화된 결과를 제공합니다. 조회 가능 컬렉션의 구체적인 구현은 조회 평가의 결과로 리턴되는 DKResults 오브젝트입니다. DKResults는 dkQueryableCollection의 서브클래스이고 DDO의 컬렉션입니다.

조회 결과 얻기

다음 예제에서는 매개변수식 조회를 제출하여 결과를 얻는 방법에 대해 설명합니다. 이 결과는 DKResults 오브젝트인 rs에 들어 있습니다. 이전 코드 예제를 사용하여 컬렉션을 처리하고 DDO를 얻을 수 있습니다.

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();
```

새 조회 평가

결과를 조회하여 추가로 조회를 세분화할 수 있습니다. 이전 예제를 기반으로 하는 다음 코드에서는 조회 재평가를 보여줍니다.

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);
...
```

두 번째 조회는 세분화된 결과를 포함하는 DKResults 오브젝트인 obj를 리턴합니다. 두 조회를 결합한 결과는 다음과 같습니다.

```
"SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL AND Subject == 'Mystery'))";
```

만족하는 결과를 얻을 때까지 조회를 반복할 수 있습니다. 한 유형으로 조회를 시작한 후에는 후속 조회도 동일한 유형이어야 합니다. 조회 유형을 혼합할 경우, 결과는 널(null)일 수 있습니다.

다음 예제에서는 순차 텍스트 조회를 보여줍니다.

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);
```

두 번째 조회는 세분화된 결과를 포함하는 DKResults 오브젝트인 obj를 리턴합니다.
두 조회를 결합한 결과는 다음과 같습니다.

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

결합된 조회 대신 조회 가능 컬렉션 사용

결합된 조회는 범위 존재 여부에 관계없이 매개변수식 조회 및 텍스트 조회로 이루어진 결합된 조회를 제출할 수 있는 융통성을 제공합니다. 그러나 이러한 조회는 조회 가능 컬렉션을 평가할 때와 마찬가지로 한번에 하나씩이 아닌 한번에 모두 제출해야 합니다.

결합된 조회는 DKResults 오브젝트를 리턴합니다, 그러나 결합된 조회에 대해 다른 매개변수식 조회를 평가할 수 없습니다. 모든 콘텐츠 서버에서 결합된 조회를 사용할 수는 없습니다.

후속 조회로 조회 가능 컬렉션을 평가하면 만족하는 최종 결과를 얻을 때까지 이전 조회의 결과를 단계별로 세분화할 수 있는 융통성이 제공됩니다. 후속 조회는 콘텐츠 서버를 동적으로 찾아보는 경우와 이전 결과를 기반으로 다음 조회를 구성하는 경우에 유용합니다. 그러나 전체 조회 내용을 미리 알고 있는 경우 완전한 조회를 한번에 제출하거나 결합된 조회를 사용하는 것이 더욱 효율적입니다.

Content Manager 버전 8.2에 대한 작업

이 절에서는 Content Manager 버전 8 릴리스 2 커넥터(ICM 커넥터) API(Application Programming Interface)에 대해 설명합니다. ICM 커넥터는 Enterprise Information Portal(EIP) 프레임워크를 확장한 것이므로 이 정보를 읽기 전에 반드시 13 페이지의 『Enterprise Information Portal 응용프로그램 프로그래밍 개념』에 설명된 EIP 프레임워크 개념을 이해해야 합니다.

ICM 커넥터 API를 사용하여 Content Manager 콘텐츠 서버에 액세스하는 사용자 조정 응용프로그램을 빌드하고 전개할 수 있습니다. API를 사용하여 기존의 응용프로그램을 Content Manager 콘텐츠 서버에 통합할 수 있습니다.

이 절에는 다음과 같은 정보가 들어 있습니다.

- Content Manager 시스템 이해
- Content Manager 개념 이해
- Content Manager 응용프로그램 계획
- Content Manager 응용프로그램 작성
- 정보에 대한 액세스 제어
- 트랜잭션 처리
- 항목 검색

Content Manager 시스템 이해

Content Manager 시스템의 기본 구성요소에는 라이브러리 서버, 하나 이상의 자원 관리자 및 객체 지향 API(Application Programming Interface) 세트가 포함됩니다. Content Manager 시스템을 관리하도록 Java 기반 시스템 관리 클라이언트도 제공됩니다.

라이브러리 서버는 융통성 있는 데이터 모델링 성능, 시스템에 대한 보안 액세스, 효율적인 콘텐츠의 관리 및 기타 기능을 제공합니다. 라이브러리 서버는 시스템의 자원 관리자에 저장된 정보를 포함하여 모든 시스템 정보에 대한 액세스를 제어할 뿐만 아니라 시스템 항목 간의 관계도 관리합니다.

자원 관리자는 스캔된 이미지, 사무용 문서 또는 비디오와 같은 실제 2진 오브젝트 콘텐츠를 저장하는 구성요소입니다. Content Manager VideoCharger 또는 다른 비IBM 제품과 같은 다른 자원 관리자를 Content Manager 시스템에 통합할 수 있습니다. 자원 관리자를 사용하여 다음과 같은 작업을 완료할 수 있습니다.

- SMS(System Managed Storage)를 사용하여 빠르고 비싼 미디어에서 느리고 저렴한 미디어로 콘텐츠를 자동으로 이동

- 웹 브라우저에서 직접 자원 관리자에 액세스
- 오브젝트 전체 또는 일부 검색
- 데이터를 라이브러리 서버와 동기화

API는 Content Manager 시스템에 대한 액세스를 응용프로그램에 제공합니다. API는 Java 및 C++에 사용 가능합니다. 응용프로그램은 API를 사용하여 데이터 모델링, 통합된 매개변수식 검색 및 텍스트 검색, 타사 데이터에 대한 액세스 및 전달과 같은 모든 Content Manager의 기능을 이용할 수 있습니다.

그림 9 도표에 시스템 구성요소가 상호 작용하는 방법이 나와 있습니다. 이는 Content Manager 시스템의 많은 구현 중 하나에 불과합니다. 예를 들어, 다른 시스템 구성에서는 네 개의 자원 관리자가 있을 수 있습니다.

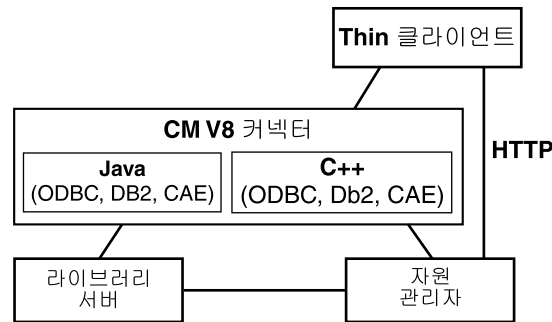


그림 9. 시스템 구성

Content Manager 개념 이해

이 절에서는 중요한 Content Manager 개념에 대해 설명합니다. 프로그래밍 작업을 진행하기 전에 Content Manager 개념을 이해하는 것이 중요합니다. 이 절에서 설명하는 정보는 다음과 같습니다.

- 항목
- 속성
- 항목 유형
- 루트 및 하위 구성요소
- 오브젝트
- 링크 및 참조
- 문서
- 폴더
- 버전화

- 액세스 제어
- 문서 관리 데이터 모델

항목

항목은 라이브러리 서버에서 관리되는 기본 엔티티입니다. 항목의 예로는 보험 증권, 청구, 전화번호 등이 있습니다. 항목은 항목 유형을 가진 인스턴스에 대한 일반 용어입니다. 오브젝트가 디지털 콘텐츠와 별도의 부분인 경우, 항목은 해당 오브젝트를 나타냅니다. 항목은 오브젝트는 아니지만 오브젝트 및 오브젝트 찾는 방법을 완전히 식별합니다. 시스템에서 항목은 문서와 폴더를 포함한 오브젝트를 나타냅니다. 문서와 같은 비즈니스 오브젝트를 정의하기 위해 항목 정의 작업을 수행합니다.

응용프로그램이 항목을 작성하면 Content Manager는 항목에 여러 개의 시스템 정의 속성을 할당하고 사용자 고유의 속성을 정의할 수 있도록 허용합니다.

시스템 정의 속성에는 작성 시간 소인 및 항목 ID가 포함됩니다. 항목 ID는 모든 항목에 대해 고유합니다. 항목 ID는 Content Manager에 저장되며 라이브러리 서버 내에서 항목을 찾는 데 사용됩니다. 응용프로그램 작성 시 항목 ID를 사용하여 항목과 연관된 모든 데이터에 액세스합니다.

속성

속성은 항목의 일정한 특성 또는 등록 정보(이름, 주소, 나이 등)를 설명하는 데이터 단위이며 해당 항목을 찾는 데 사용할 수 있습니다.

속성을 그룹화하여 속성 그룹을 작성할 수 있습니다. 예를 들어, 주소 속성은 번지, 구/군/시, 시/도 및 우편번호를 포함한 속성 그룹으로 구성될 수 있습니다.

다중 값이 있는 속성을 정의할 수도 있습니다. 이러한 속성을 여러 값 지정 속성이라고 하며 이 속성은 하위 구성요소로 구현됩니다. 예를 들어, 보험 증권 소유자에 대해 여러 개의 주소, 집 주소, 직장 주소 등을 저장할 수 있습니다.

자세한 정보는 SAttributeDefinitionCreationICM 샘플을 참조하십시오.

항목 유형

항목 유형(이전 Content Manager 버전의 색인 클래스)은 항목을 정의하고 나중에 이를 찾기 위한 템플릿입니다. 항목 유형은 루트 구성요소, 0 이상의 하위 구성요소 및 분류로 구성됩니다. 항목 유형은 모든 구성요소 및 관련 데이터를 포함하는 전체 구조입니다. 예를 들어, 보험 시나리오에서 보험 증권 항목 유형에는 증권 번호, 가입자 이름, 청구와 같은 속성의 항목이 포함됩니다.

Content Manager에서는 항목 유형을 비자원, 자원, 문서(문서 모델이라고도 함) 및 문서 부분의 네 가지로 분류합니다. 비자원 항목 유형은 자원 관리자에 저장되지 않는 엔티티를 나타냅니다. 자원 항목 유형은 파일 시스템의 파일, 비디오 서버의 비디오 클립,

데이터베이스 테이블의 LOB(대형 오브젝트) 등과 마찬가지로 자원 관리자에 저장된 오브젝트를 나타냅니다. 문서 항목 유형은 단일 자원 항목 유형과 마찬가지로 자원 컨테츠가 있는 문서 부분이 포함된 엔티티를 나타냅니다. 문서 부분 항목 유형은 자원 관리자에 저장된 오브젝트를 나타내지만 문서 항목 유형에 포함된 문서의 일부분입니다. Content Manager는 자원 항목 유형의 기본 세트, 즉 LOB, 텍스트, 이미지, 스트림 및 비디오 오브젝트를 제공합니다.

자세한 정보는 SItemTypeCreationICM 샘플을 참조하십시오.

루트 및 하위 구성요소

항목 유형은 루트 구성요소 하나와 선택적인 하위 구성요소로 구성되어 있습니다.

루트 구성요소는 계층 구조 항목 유형의 첫 번째 레벨 또는 유일한 레벨입니다. 항목 유형은 시스템 및 사용자 정의 속성으로 모두 구성되어 있습니다. 내부적으로 가장 기본적인 항목 유형에는 하나의 구성요소만 포함됩니다.

하위 구성요소는 계층 구조 항목 유형의 선택적인 두 번째 이하 레벨입니다. 각 하위 구성요소는 직접 상위 레벨과 연관됩니다. 그림 10은 Content Manager 메타 모델의 도표를 표시한 것입니다. 루트 및 하위 구성요소를 표시하고 항목 계층 구조 형성 시 이들의 관계를 표시합니다. 또한 도표는 링크, 참조, 자원 항목 및 자원 오브젝트를 표시합니다.

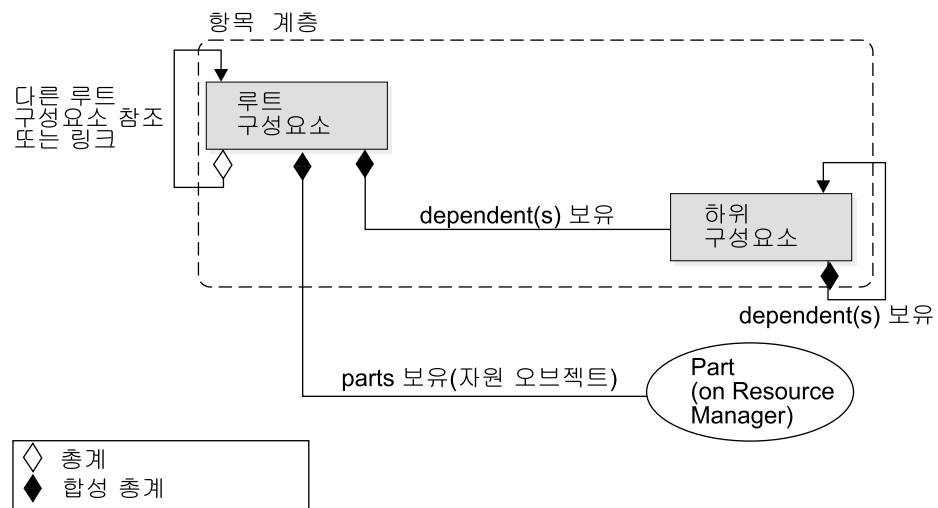


그림 10. Content Manager 메타 모델: 논리 보기.

149 페이지의 그림 11에서는 보험 응용프로그램에 대한 데이터 모델 예제를 보여줍니다. 이 응용프로그램에는 루트 구성요소로 청구, 경찰 보고서가 포함된 보험 증권이 있고, 하위 구성요소로 손상 예상이 있습니다.

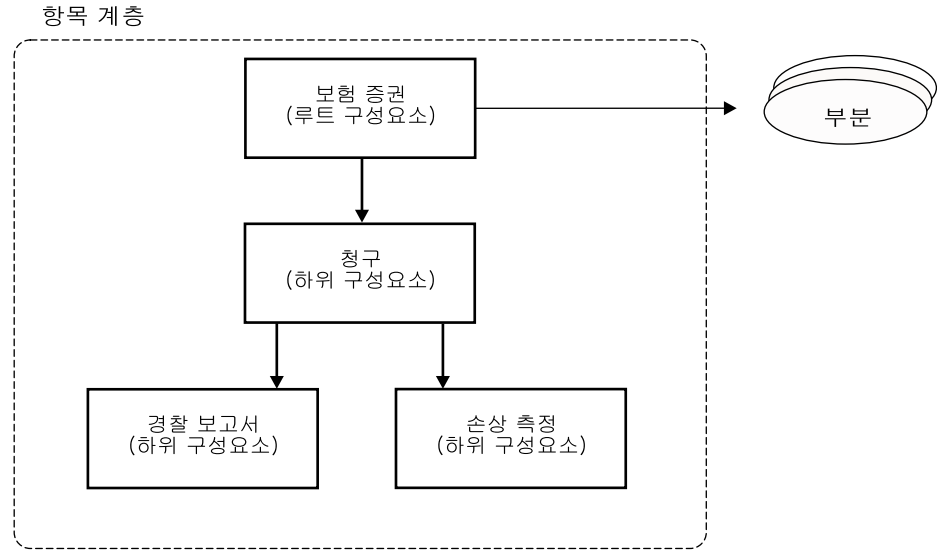


그림 11. 구성요소 계층 구조

Content Manager 데이터 모델링 개념에 대한 자세한 정보는 SItemTypeCreationICM 샘플을 참조하십시오.

오브젝트

오브젝트(자원 콘텐츠라고도 함)는 사용자가 단일 단위(예: JPEG 이미지, MP3 오디오, AVI 비디오 및 책의 텍스트 블록)로 저장, 검색 및 조작할 수 있는 디지털 콘텐츠입니다. 오브젝트는 항상 자원 관리자에 저장됩니다. 오브젝트에 대한 액세스는 라이브러리 서버에 의해 제어됩니다.

자세한 정보는 SResourceItemCreationICM 샘플을 참조하십시오.

링크 및 참조

링크를 사용하여 항목 간 다수 연관에 대해 하나를 모델링할 수 있습니다. 그림 10에 표시된 것처럼, 링크는 항목의 루트 구성요소를 연관시키는 데만 사용될 수 있습니다. 링크는 항목의 특정 버전을 참조하지 않습니다. 모든 링크를 정의할 수 있습니다. 링크는 원본에서 목표로 링크하기에 융통성 있는 방식입니다. 링크는 간단히 두 항목을 연관시키고 링크 항목 또는 두 항목에 링크할 수도 있는 다른 항목에 액세스하는 수단을 제공합니다. 링크의 사용법은 사용자 응용프로그램으로 런타임 시 결정됩니다. 응용프로그램에서 모든 링크를 사용할 수 있습니다.

링크는 개방되어 있고 응용프로그램에서 사용할 수 있는 비제한적이고 융통성 있는 빌딩 블록입니다. 다음과 같이 응용프로그램 내의 링크에 추가 제한사항을 배치하는 옵션이 있습니다.

링크를 사용할 수 있는 방법 중 하나는 폴더링 또는 컨테이너(container)-컨테이너(containee) 관계를 나타내는 것입니다. 링크를 사용하여 폴더링을 구현하도록 선택한

경우, 컨테이너는 컨테이너를 소유하지 않습니다. 즉, 컨테이너의 항목은 컨테이너를 삭제해도 삭제되지 않습니다. 링크에 대한 자세한 정보는 SLinksICM 샘플을 참조하십시오.

참조는 구성요소(루트 또는 하위) 및 다른 루트 구성요소 간에 있습니다. 참조는 설계 시 정의된 구성요소의 참조 속성으로 나타납니다. 구성요소 정의는 기타 루트 구성요소를 참조하는 참조 속성을 구성요소 유형 정의 시 지정한 임의의 수로 가질 수 있습니다. 참조는 일반적으로 소유권을 표시하지 않지만 필요한 경우 소유권 관계를 구현할 수 있습니다.

구성요소 유형에 참조를 추가하면 해당 구성요소 유형의 항목은 다른 항목을 참조할 수 있습니다. DDO는 참조 이름으로 식별되는 속성을 가지고 있습니다. 속성값은 다른 DDO로 설정될 수 있습니다. DDO의 속성값은 참조에 의해 참조되는 DDO입니다.

참조는 루트 및 다른 루트 구성요소 유형을 참조하는 하위 구성요소 유형에 모두 정의할 수 있습니다. 148 페이지의 그림 10을 참조하십시오. 또한 참조는 특정 버전의 항목을 참조할 수 있는 반면, 링크는 모든 버전을 참조합니다. 참조 속성에 대한 자세한 정보는 SReferenceAttrDefCreationICM 샘플을 참조하십시오.

문서

시스템에 있을 수 있는 문서 유형에는 두 가지가 있습니다. 문서의 첫 번째 유형은 의미 유형 "문서"의 항목으로, 여기에는 문서 형성 정보를 포함하는 것으로 예상됩니다. 이 "문서" 유형은 독립형이거나 데이터 모델에서 문서 모델을 구현한 경우 부분을 포함할 수 있습니다. 이 문서 유형에 대한 자세한 정보는 SItemCreationICM API 교육 샘플을 참조하십시오.

문서의 다른 유형은 항목 유형("문서 모델"이라고도 함)을 분류한 "문서"에서 작성된 항목입니다. 이 문서 유형에는 Content Manager 문서 모델의 특정 구현인 문서 부분이 있습니다. 문서 부분은 텍스트, 이미지 및 스프레드시트를 비롯하여 다양한 유형의 콘텐츠를 포함할 수 있습니다. 문서 모델에 대한 자세한 정보는 SDocModelItemICM 샘플을 참조하십시오.

폴더

폴더는 임의 유형의 다른 항목을 포함할 수도 있는 항목입니다. 또한 다른 폴더를 포함할 수도 있습니다. Content Manager 버전 8에서 폴더의 개념은 항목 간 링크 관계를 사용하여 구현됩니다. 항목은 다른 항목을 포함하여 폴더 계층 구조라는 포함 계층 구조를 형성할 수 있습니다. 예를 들어, 보험 증권 항목은 보험 증권 항목 유형에 속하고 여러 개의 청구를 가질 수 있으므로 보험 증권을 사진 및 주민등록번호와 같은 다른 항목을 보유하는 폴더로 만듭니다. 모든 항목이 폴더가 될 수 있고 다른 항목을 무제한으로 포함할 수 있으므로 폴더는 매우 융통성이 있습니다. 자세한 정보는 SFolderICM 샘플을 참조하십시오.

버전화

버전화는 항목의 하위 구성요소 버전을 포함한 여러 항목 버전을 저장하고 유지보수하는 기능입니다. 항목 유형 정의 시 버전 규칙을 지정합니다. 항목 유형이 버전화에 사용 가능한 경우, 해당 항목 유형의 모든 항목이 버전화됩니다.

항상 또는 응용프로그램에 의한 버전화에는 두 가지 유형이 있습니다. 항목이 버전화에 항상 사용 가능한 경우, 항목의 새 버전은 항목이 콘텐츠 서버에 갱신 또는 저장될 때마다 자동으로 작성됩니다. 항목이 응용프로그램에 의한 버전화에 사용 가능한 경우, 시스템만이 사용자 응용프로그램으로 지정된 때의 새 버전을 작성합니다.

버전화는 Content Manager 라이브러리 서버에서 처리됩니다. 콘텐츠가 자원 관리자에 저장된 항목의 각 버전은 RM 콘텐츠의 자체 복사를 가지고 있습니다. 버전화 지원의 주요 특성으로는 다음과 같은 것들이 있습니다.

- 버전화는 루트 구성요소 및 해당 전체 계층 구조와 관련됩니다.
- 항목 유형은 세 가지 가능한 버전화 방침(항상 버전화, 버전화하지 않음(기본값), 응용프로그램 제어 버전화) 중 하나일 수 있습니다.
- CM 시스템에 있는 항목의 모든 버전은 검색 가능합니다.
- 항목의 모든 버전은 갱신 및 삭제할 수 있습니다.
- 응용프로그램 제어 버전화를 가진 항목 유형의 경우, 항목이 갱신되면 사용자는 기존 버전에 갱신사항을 적용하거나 갱신사항에 기초하여 새 버전을 작성하는 옵션을 가집니다.
- 항목의 각 버전은 자체의 지속 식별자(PID)를 가지고 있습니다. PID는 여러 부분으로 되어 있으며 현재 구문에 두 부분이 관련됩니다. 첫 번째 관련 부분은 항목의 서로 다른 모든 버전에 걸쳐 동일한 ItemID입니다. 다른 부분은 버전 번호입니다. 항목의 각 버전은 검색하여 문자열로 설정할 수 있는 서로 다른 버전 번호를 가지고 있습니다. 다음은 버전 번호로 작업하는 방법을 보여주는 샘플입니다.

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
String version = pid.getVersionNumber();
....
pid.setVersionNumber(version);
```

- 항목 유형은 각 항목별 버전의 제한된 숫자만을 유지하도록 구성될 수 있습니다. 항목에 대한 갱신이 허용되는 최대수를 초과하는 경우, 가장 오래된 저장 버전이 삭제되고 시스템은 새 버전을 작성합니다.
- 버전화 가능 항목이 다시 색인화되면 이전의 모든 항목 버전이 자동으로 삭제됩니다.
- 항목의 하위 구성요소는 해당 상위 구성요소의 버전을 상속합니다.
- 하위 구성요소 유형의 버전은 상위 유형의 버전화를 따르기 때문에 변경할 수 없습니다.
- 부분 레벨 버전화 규칙은 유형을 나타내는 항목 유형 관계 오브젝트에서 가져올 수 있습니다.

버전화에 대한 자세한 정보는 SItemUpdateICM 및 SItemTypeCreationICM 샘플을 참조하십시오.

액세스 제어

Content Manager 액세스 제어 모델은 다음의 기본 요소로 구성되어 있습니다.

- 사용 권한 및 사용 권한 세트
- 제어된 엔티티
- 사용자 및 사용자 그룹
- 액세스 제어 목록

여러 액세스 제어 요소는 다음과 같이 작동합니다. 각 Content Manager 사용자에게는 사용자 사용 권한 세트가 부여됩니다. 이 사용 권한은 사용자가 수행할 수 있는 조작을 정의합니다. 사용자의 유효 액세스 권한은 사용자 정의 사용 권한을 초과할 수 없습니다.

Content Manager의 액세스 제어 모델은 제어된 엔티티에 적용됩니다. 제어된 엔티티는 보호된 사용자 데이터 단위입니다. 제어된 엔티티는 Content Manager에서 항목, 항목 유형 또는 전체 라이브러리 레벨일 수 있습니다. 예를 들어, ACL을 항목 유형에 바인드하여 항목 유형 레벨에서 액세스 제어를 시행할 수 있습니다. 제어된 엔티티의 조작은 액세스 제어 목록(ACL)이라는 하나 이상의 제어 규칙으로 규정됩니다. Content Manager 시스템의 제어된 모든 엔티티는 ACL에 바인드되어야 합니다.

사용자가 항목에 대한 조작을 시작하면, 시스템은 항목에 바인드된 사용자 사용 권한 및 ACL을 점검하여 사용자가 항목에 대해 이러한 조작을 수행할 권한을 가지고 있는지 여부를 판별합니다. 논리적으로 볼 때 항목에 액세스할 권한을 가지려면 항목이 정의된 항목 유형에 액세스할 권한 또한 필요합니다. 그림 12에서는 사용 권한 및 ACL에 기초하여 항목에 대한 사용자 액세스 권한을 시스템이 결정하는 방식에 대한 예제를 보여줍니다.

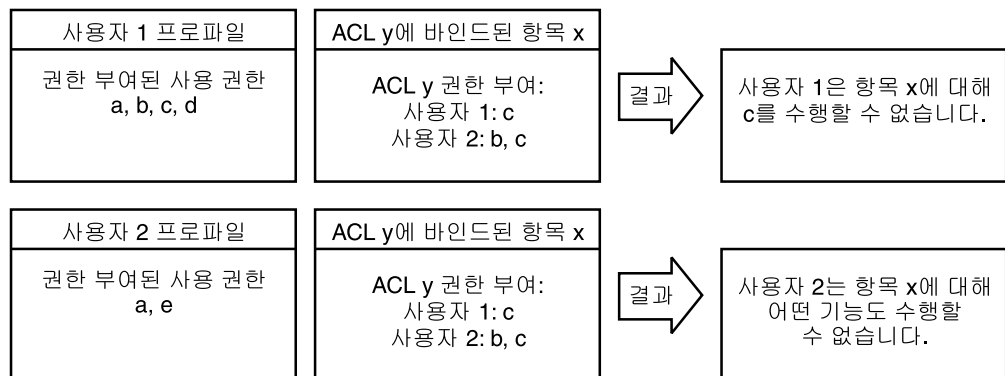


그림 12. 액세스 제어 도표

사용 권한 및 사용 권한 세트

사용 권한은 시스템의 항목에 대해 작성 또는 삭제와 같은 특정 조치를 수행할 수 있게 합니다. 모든 Content Manager 사용자에게는 사용자의 사용 권한 세트가 부여됩니다. 사용 권한은 사용자가 Content Manager 시스템에 있는 정보에 대해 수행할 수 있는 최대 조작을 정의합니다. 사용자의 액세스 권한은 사용자에게 대해 정의된 사용자의 사용 권한을 초과하지 않습니다.

Content Manager는 시스템 정의 사용 권한이라는, 사용자가 변경할 수 없는 다수의 미리 정의된 사용 권한을 제공합니다. 사용자 정의 사용 권한이라는 사용자 고유의 사용 권한을 정의할 수도 있습니다. 사용자 종료 루틴을 사용하여 사용자 응용프로그램에서 사용자 정의 사용 권한을 시행합니다.

모든 사용 권한에는 사용 권한 정의 코드라는 고유한 시스템 생성 코드가 있습니다. 0 - 999의 사용 권한 정의 코드는 시스템 정의 사용 권한에 대해 예약됩니다. 사용자 지정 사용 권한에 대해서는 1000 이상의 코드를 사용할 수 있습니다.

시스템 정의 사용 권한은 시스템 관리 사용 권한과 데이터 액세스 사용 권한의 두 개의 카테고리로 분류됩니다. 시스템 관리 사용 권한을 사용하여 사용자 데이터를 모델링하고 Content Manager 시스템을 관리 및 유지보수할 수 있습니다. 시스템 구성, 라이브러리 서버 구성 관리 및 항목 유형 관리와 같은 작업을 완료하려면 시스템 관리 사용 권한이 필요합니다. 데이터 액세스 사용 권한을 사용하여 항목 및 항목 유형과 같은 시스템 데이터에 액세스하고 변경할 수 있습니다.

사용자에게 할당된 사용 권한 그룹이 사용 권한 세트입니다. 예를 들어, 하나의 사용 권한 세트에는 작성, 갱신 및 삭제 사용 권한이 포함될 수 있습니다. 사용 권한 세트는 시스템 관리를 더욱 용이하게 합니다. 사용 권한을 사용하기 전에 세트로 그룹화해야 합니다. 한 세트에 포함될 수 있는 사용 권한 수에는 제한이 없습니다.

Content Manager의 미리 정의된 사용 권한 세트에는 시스템 관리 사용 권한이 포함됩니다.

AllPrivSet; PrivSetCode: 1

이 사용 권한 세트를 가진 사용자는 모든 Content Manager 엔티티에 대해 모든 기능을 수행할 수 있습니다. 이 사용 권한 세트에 포함된 사용 권한에는 모든 시스템 정의 및 사용자 정의 사용 권한이 포함됩니다.

NoPrivSet; PrivSetCode: 2

이 사용 권한 세트를 가진 사용자는 모든 Content Manager 엔티티에 대해 어떠한 기능도 수행할 수 없습니다. 이 사용 권한 세트에 포함된 사용 권한에는 없음이 포함됩니다.

SystemAdminPrivSet; PrivSetCode: 3

이 사용 권한 세트를 가진 사용자는 모든 Content Manager 시스템 관리 및 데이터 모델링 기능을 수행할 수 있습니다. 이 사용 권한 세트에 포함된 사용 권한에는 다음이 포함됩니다.

ItemAdminPrivSet; PrivSetCode: 4

이 사용 권한 세트를 가진 사용자는 모든 Content Manager 데이터 모델링 및 항목 액세스 기능을 수행할 수 있습니다. 이 사용 권한 세트에 들어 있는 사용 권한에는 다음이 포함됩니다.

- 시스템 정의 항목 유형 사용 권한
- 항목 SQL 선택 사용 권한
- 항목 유형 조회 사용 권한
- 항목 조회 사용 권한
- 항목 추가 사용 권한
- 항목 세트 사용자 정의 attr 사용 권한
- 항목 세트 시스템 정의 attr 사용 권한
- 항목 삭제 사용 권한
- 항목 이동 사용 권한
- 항목 링크 사용 권한
- 항목 링크된 사용 권한
- 항목 고유 사용 권한
- 항목 소유 사용 권한
- 항목 추가 링크 사용 권한
- 항목 변경 링크 사용 권한
- 항목 제거 링크 사용 권한
- 항목 체크아웃 사용 권한

표 8. 사용 권한 코드

사용 권한 이름	코드
ICMLogon	1
SystemAdmin	40
SystemDefineItemType	45
ItemSQLSelect	121
ItemTypeQuery	122
ItemQuery	123
ItemAdd	124
ItemSetUserAttr	125
ItemSetSysAttr	126

사용자 및 사용자 그룹

대부분의 경우 시스템에 대해 동일한 유형의 액세스가 필요한 사용자 그룹을 가지고 있습니다. 예를 들어, 보험 회사의 모든 보험업자에게는 해당 청구 항목 유형에 대한 검색 및 갱신 사용 권한이 필요합니다. 보험업자 및 공통 액세스 요구사항을 가진 기타 사용자를 사용자 그룹으로 그룹화할 수 있습니다. 그러나 하나의 사용자 그룹을 다른 사용자 그룹으로 가져다 놓을 수는 없습니다.

사용자 그룹은 단지 편의를 위해 유사한 작업을 수행하는 각 사용자를 그룹화한 것입니다. 사용자 그룹은 0명 이상의 사용자로 구성되어 있습니다. 사용자 그룹에는 사용 권한 세트를 할당할 수 없습니다. 사용자 그룹의 각 사용자에게는 하나의 사용 권한 세트가 있습니다. 사용자 그룹을 사용하면 시스템의 오브젝트에 대한 액세스 제어 목록을 쉽게 작성할 수 있습니다. 사용자 그룹은 다른 그룹에 속할 수 없습니다.

액세스 제어 목록(ACL)

사용자가 Content Manager 시스템에 항목을 작성하면 해당 사용자는 다른 사용자가 그 항목에 대해 가지게 될 액세스를 정의해야 하며, 해당 항목에 대해 수행할 수 있는 조작을 정의해야 합니다. 항목에 대해 액세스 권한을 가지는 사용자 목록 및 항목에 대해 수행할 수 있는 연산 목록을 액세스 제어 목록(ACL)이라고 합니다. ACL은 하나 이상의 개별 사용자 ID 또는 사용자 그룹 및 연관된 사용 권한을 포함할 수 있습니다. 항목, 항목 유형 및 작업 목록을 ACL에 연관시킬 수 있습니다. 사용 권한 세트는 시스템을 사용할 수 있는 개인의 최대 능력을 정의하고, ACL은 항목에 대한 개인의 액세스 권한을 제한합니다. 예를 들어, ACL에서 사진 항목의 삭제를 허용하더라도 사용 권한 세트에 삭제 사용 권한이 없으면 사진을 삭제할 수 없습니다.

제어된 엔티티는 ACL 코드를 통해 특정 ACL로 바인드됩니다. 제어된 엔티티와 연관될 때 ACL은 바인드된 엔티티의 권한을 정의하고 사용자의 사용 권한을 방해하지 않습니다. ACL이 시행되었고 사용자의 사용 권한이 점검됩니다.

액세스 제어 규칙에 지정된 사용자는 개별 사용자, 사용자 그룹 또는 공용이 될 수 있습니다. 규칙의 UserKind 필드에 의해 해석이 결정됩니다. 설명을 위한 규칙 유형에는 ACL Rule for User, ACL Rule for Group 및 ACL Rule for Public과 같은 이름을 제공할 수 있습니다. 사용자가 사용자의 사용 권한 점검을 통과한 경우, ACL Rule for Public은 public을 지정하여 모든 사용자가 바인드된 엔티티의 ACL 사용 권한에 지정된 조작을 수행할 수 있는 권한을 부여합니다. Public으로 바인드된 엔티티에 대한 ACL 사용 권한이 시스템 레벨에서 구성될 수 있습니다. Public으로 바인드된 엔티티를 여는 기능은 시스템 전체에서 구성될 수 있습니다. 구성 매개변수는 PubAccessEnabled(테이블 ICMSTSysControl에 정의됨)입니다. 모든 ACL Rules for Public이 사용 불가능하게 되면 액세스 제어 처리 중에 무시됩니다.

동일한 ACL 내에 사용자가 둘 이상의 규칙 유형에 지정될 수 있습니다. 최상위에서 최하위로 세 가지 유형의 우선순위는 ACL Rule for Public, ACL Rule for User 및

ACL Rule for Group입니다. ACL 점검을 적용할 때 우선순위가 더 높은 유형이 전달될 경우, 권한이 분석되고 프로세스가 정지됩니다. ACL Rule for Public 점검에 실패한 경우, 우선순위가 더 낮은 규칙 유형에 대해 점검 프로세스가 계속됩니다.

ACL Rule for the User 점검에 실패하면 점검이 정지됩니다. ACL Rule for Group은 점검되지 않습니다. 사용자가 개별 사용자 점검을 수행하는 경우, 해당 사용자는 액세스 제어 알고리즘에 기초하여 그룹 유형 액세스로부터 제외되므로 그룹 유형에 대해 점검을 계속할 필요는 없습니다. 개별 사용자 유형 및 그룹 유형의 액세스 제어 점검은 순차적 프로세스가 아닙니다. 순차적 점검을 수행해도 문제가 없지만 해당 프로세스는 둘 중에 하나를 선택해야 하는 상황입니다.

사용자가 개별 사용자 유형 점검 전달에 실패한 경우(또는 사용자가 액세스 목록 테이블에 규칙을 갖고 있지 않는 경우), 점검 프로세스가 그룹 유형으로 계속됩니다. 사용자가 그룹 중 하나에 속하고 사용 권한 점검이 전달되는 경우, 권한이 분석되고 프로세스가 정지됩니다. 그렇지 않으면 액세스가 거부되고 프로세스도 정지됩니다. 사용자가 둘 이상의 ACL Rule for Group에 지정된 경우, 사용자는 모든 규칙의 ACL 사용 권한 조합에 의해 권한이 부여됩니다. 둘 이상의 ACL Rule for User에 사용자가 결코 지정되지 않습니다.

CM 시스템은 SuperUserACL, NoAccessACL 및 PublicReadACL과 같은 미리 구성된 ACL을 제공합니다.

SuperUserACL

이 ACL은 CM의 미리 구성된 사용자 ICMADMIN이 바인드된 엔티티에 대해 모든 CM 함수(AllPrivSet)를 수행할 수 있는 권한을 부여하는 단일 규칙으로 구성되어 있습니다.

NoAccessACL

이 ACL은 모든 CM 사용자(public)에 대해 조치(NoPrivSet)가 허용되지 않도록 지정하는 단일 규칙으로 구성되어 있습니다.

PublicReadACL

이 ACL은 모든 CM 사용자(ICMPUBLIC)에 대해 읽기 기능(ItemReadPrivSet)이 허용되도록 지정하는 단일 규칙으로 구성되어 있습니다. 이 값이 사용자의 DfltACLCode에 할당된 기본값입니다.

Content Manager 응용프로그램 계획

이 절에서는 Content Manager 응용프로그램을 작성하기 위한 요구사항을 식별하는 것을 도와주며, Content Manager가 작동하는 방법에 대한 정보를 제공합니다. 응용프로그램 계획의 주요 부분은 비즈니스의 요구를 충족시키는 데이터 모델의 작성입니다. Content Manager 데이터 모델에 대한 자세한 정보는 *Content Management 시스템 계획 및 설치*와 샘플 디렉토리의 tItemTypeCreationICM 샘플을 참조하십시오.

이 절에서는 다음 주제에 대해 설명합니다.

- 응용프로그램 기능 결정
- 오류 처리

응용프로그램 기능 결정

응용프로그램을 개발하기 위해 취하는 접근 방법은 조직의 요구사항에 따라 다릅니다. 효율적인 응용프로그램을 생성하려면 조직의 모든 관련 부서가 응용프로그램의 계획 및 설계에 참여해야 합니다. 계획에 대한 자세한 도움말을 보려면 *Content Management 시스템 계획 및 설치*를 참조하십시오.

응용프로그램을 작성하기 전에 다음 모든 질문 또는 대부분의 질문에 대답할 수 있어야 합니다.

- 조직에서 사용하는 문서의 유형은 무엇입니까?
- 기존의 문서에 들어 있는 콘텐츠의 유형은 무엇입니까?
- 문서의 처리 방법은 무엇입니까?
- 문서 처리를 자동화할 수 있습니까?
- 어떤 방법으로 문서를 수신, 표시, 저장 및 분배할 수 있습니까?
- 저장 후 문서 검색 빈도는 얼마입니까?
- 조직에서 관리하는 문서의 양은 어느 정도입니까?
- 대형 오브젝트 저장에 사용할 기억영역 미디어의 유형은 무엇입니까?
- 조직에서 사용하는 다른 응용프로그램이 있습니까?
- 얼마나 많은 사용자와 어떤 유형의 액세스 제어를 가질 계획입니까?

응용프로그램에 포함시킬 기능을 결정하려면 위의 질문에 대한 응답을 사용하십시오.

오류 처리

오류 처리 시 포착해야 할 가장 중요한 예외는 `DKException` 클래스입니다. 프로그램 논리에 예외를 사용하지 말고, 콘텐츠 서버에 무언가 존재하는지 여부 또는 실제로 예외의 경우가 아닌 다른 이유를 감지하는 예외 포착에 의존하지 마십시오. 프로그램 논리에서 예외를 사용하면 성능이 저하되고 추적 및 로그 정보가 디버깅 및 지원용으로 소용 없게 될 수 있습니다.

모든 예외 정보를 주의 깊게 검토하십시오. `DKException`의 서브클래스에는 여러 가지가 있으며 프로그램에 따라 각 예외를 개별적으로 처리하는 것이 가장 좋습니다. 표 9에는 `DKException` 정보가 들어 있습니다.

표 9. `DKException` 정보

<code>DKException</code>	설명
이름	예외 클래스 이름. 서브클래스 이름을 포함합니다.

표 9. *DKException* 정보 (계속)

DKException	설명
메시지	특정 메시지가 오류를 설명합니다. 메시지는 여러 가지 정보를 포함할 수 있으며, 때때로 오류가 발견되었을 때 주요 변수 상태를 요약합니다.
메시지 ID	고유 메시지 ID는 이 오류 유형을 식별하며 위에 사용되는 주요 메시지와 일치합니다.
오류 상태	OO API 또는 라이브러리 서버 오류 상태에 대한 추가 오류 정보를 포함할 수 있습니다. 라이브러리 서버가 오류를 발견하면, 여기에 다음 네 가지 정보가 패키집니다. 리턴 코드 이유 코드 Ext/SQL 리턴 코드 Ext/SQL 이유 코드
오류 코드	라이브러리 서버 리턴 코드를 포함할 수 있습니다.
스택 추적	사용자 프로그램에서 오류가 발생한 지점과 OOAPI가 마지막으로 오류를 발견하거나 처리한 정확한 지점을 나타내는 주요 정보.

Java로 작업할 때는 `java.lang.Exception`도 처리해야 합니다. `samples` 디렉토리의 `SConnectDisconnectICM` 샘플은 오류 포착 및 인쇄 방법을 설명합니다. 로그 기록 및 추적에 대한 자세한 정보는 `메시지 및 코드`를 참조하십시오.

Content Manager 샘플에 대한 작업

Content Manager는 광범위한 코드 샘플 세트를 제공하여 주요 Content Manager 작업을 완료하는 데 도움을 줍니다. 샘플은 참조 정보, 프로그래밍 지침, API 사용 예제 및 도구를 제공하므로 API 교육의 원본이 됩니다.

제품 Information Center의 온라인 응용프로그램 프로그래밍 참조서에서 샘플을 볼 수 있습니다. 또한 샘플은 `CMBROOT\Samples\java\icm` 및 `CMBROOT\Samples\cpp\icm` 디렉토리에 있습니다. 그러나 디렉토리에 샘플을 가지기 위해 EIP 설치 시 샘플 및 도구 구성요소를 선택했어야 함에 유의하십시오.

샘플을 최대한 활용하려면 샘플 Readme를 반드시 읽으십시오. 찾고 있는 개념 또는 주제가 들어 있는 샘플을 빨리 찾는 데 도움을 주는 완전한 참조 색인을 포함하고 있습니다. 각 샘플을 완전하게 설명하게 깊이 있는 개념 정보와 각 작업 단계에 대한 설명을 제공합니다. 각 샘플에 포함된 추가 정보는 다음과 같습니다.

- 샘플에 표시된 개념을 설명하는 자세한 헤더 정보
- 전제조건 정보 및 명령행 사용법을 포함한 샘플 파일에 대한 설명
- 쉽게 자르고 사용자 조정하여 응용프로그램에 사용할 수 있는 완전한 주석을 가진 코드
- 응용프로그램 개발 시 사용할 수 있는 유틸리티 기능

샘플 Readme의 시작하기 절은 다음 일반 작업을 완료하는 방법을 신속하게 배우도록 도와줍니다.

- 데이터 모델링
- 서버 연결 및 오류 처리
- 속성 및 속성 그룹 정의
- 참조 속성에 대한 작업
- 데이터 모델 정의
- 항목에 대한 작업
- 자원 항목에 대한 작업
- 폴더에 대한 작업
- 링크에 대한 작업
- 자원 관리자 정의
- SMS 콜렉션 정의
- 항목 검색

보험 시나리오 샘플

Content Manager에서는 보험 회사를 사용하여 가능한 하나의 "실제 세계" 구현에 대한 코드 샘플을 제공합니다. 보험 회사 샘플을 작성하는 데 사용되는 정보는 주요 Content Manager 기능 설명을 도와주기 위한 의도로만 제작 및 작성되었습니다. 보험 시나리오를 구성하는 전체 샘플 목록은 샘플 Readme를 참조하십시오.

Content Manager 응용프로그램 작성

Content Manager 버전 8 릴리스 2 기능을 구현하는 API는 ICM 커넥터라는 커넥터로 그룹화됩니다. ICM 커넥터 API는 예제 DKDatastoreICM과 같이 ICM이라는 접미부를 갖습니다.

이 절에 들어 있는 정보는 다음과 같습니다.

- 소프트웨어 구성요소 이해
- DDO를 사용한 항목 표현
- Content Manager 시스템에 연결
- 항목에 대한 작업

소프트웨어 구성요소 이해

개념상 OO API를 다음의 서비스 그룹으로 구분할 수 있습니다.

- 데이터 및 문서 모델링
- 검색
- 데이터 가져오기 및 전달

- 시스템 관리
- 문서 경로지정

데이터 및 문서 모델링 모듈에는 비즈니스 데이터 모델을 기본 Content Manager 계층 구조 데이터 모델로 맵핑할 수 있는 API가 포함되어 있습니다. 예를 들어, 보험 회사의 데이터 모델에는 보험 증권이 들어 있는데, 이는 Content Manager 데이터 모델에서는 필수 항목입니다. 데이터 및 문서 모델링 모듈 API는 보험 증권을 나타내는 항목을 정의하기 위한 인터페이스를 제공합니다.

검색 모듈은 문서 및 폴더와 같은 관리 항목에 대한 요청을 처리합니다. 검색 모듈 API를 사용하면 결합된 텍스트 및 매개변수식 검색을 수행하여 Content Manager 시스템에 포함된 항목을 검색할 수 있습니다. 검색 결과는 검색 결과 세트 형식으로 응용프로그램에 리턴됩니다.

데이터 가져오기 및 전달 모듈은 네트워크 또는 웹과 같은 여러 미디어를 통해 데이터를 시스템으로 가져오고 해당 데이터를 전달할 수 있는 API를 제공합니다.

시스템 관리 모듈은 효율적이고 안전한 Content Manager 시스템을 구성 및 유지보수하기 위한 인터페이스를 제공합니다. 예를 들어, 시스템 관리 API를 응용프로그램에 통합하여 시스템 제어 설정을 조정하고 사용자를 관리하며 사용자의 사용 권한을 지정하고 시스템에 대한 액세스를 허용할 수 있습니다.

문서 경로지정 모듈 API는 프로세스를 통해 해당 비즈니스의 요구에 따라 정의된 대로 문서와 같은 비즈니스 오브젝트를 경로지정하는 데 도움이 됩니다.

DDO를 사용한 항목 표현

응용프로그램을 작성하기 전에 14 페이지의 『동적 데이터 오브젝트 개념 이해』에 설명된 DDO/XDO 프로토콜 개념을 이해해야 합니다. 이 절의 정보는 Content Manager 버전 8 릴리스 2에만 해당됩니다.

DDO는 필수 속성 컨테이너입니다. 속성은 이름, 값 및 여러 가지 등록 정보를 가지고 있습니다. 속성의 가장 중요한 등록 정보 중 하나는 속성 유형입니다. DDO는 지속 식별자(PID)를 가지고 있어 오브젝트가 지속 기억영역에 상주하는 위치를 나타냅니다. DDO는 DDO에 데이터를 채우기 위한 몇 가지 메소드와 항목 정보를 검색할 수 있는 메소드를 갖고 있습니다. DDO 메소드에는 add, retrieve, update, 및 delete가 포함됩니다. 이러한 메소드를 사용하여 Content Manager와 같은 지속 기억영역의 내부 및 외부로 항목 데이터를 이동합니다.

메모리에서 Content Manager 항목은 DDO로 나타납니다. 항목 속성은 이름, 유형 및 값을 가진 DDO 속성으로 나타납니다. 링크 및 참조도 속성의 특수한 유형으로 나타남

니다. 그러나 참조 속성은 다른(단일) DDO 또는 XDO를 참조하는 반면, 링크 속성은 DDO 또는 XDO의 컬렉션(복수)을 참조한다는 차이점이 있습니다. XDO는 대형 오브젝트(LOB)를 나타내는 데 사용됩니다.

항목, 즉 XDO 또는 다른 DDO에 대한 참조에는 유형 등록 정보가 오브젝트 참조로 설정된 이름과 참조된 오브젝트의 인스턴스를 참조하도록 설정된 값이 있습니다. 또한 하위 구성요소 및 링크도 유형 등록 정보가 데이터 오브젝트 컬렉션으로 설정되고 값은 DDO 컬렉션으로 설정된 DDO 속성으로 나타납니다. 하위 구성요소의 경우, 속성 이름은 하위 구성요소의 이름입니다. 값은 루트 구성요소에 속하는 하위 구성요소의 컬렉션입니다. 루트 항목을 삭제하면 루트 항목의 모든 하위 구성요소도 삭제됩니다.

Content Manager 시스템에 연결

Content Manager 응용프로그램을 빌드할 때 가장 먼저 수행해야 할 작업 중 하나는 서버에 연결하는 것입니다. 이 절은 Content Manager 서버에 연결 및 연결 해제와 관련된 여러 가지 작업을 수행하는 데 도움이 됩니다.

Content Manager 서버에 액세스하려면 응용프로그램은 공통 서버로 작동하는 콘텐츠 서버를 작성해야 합니다. 콘텐츠 서버를 작성하여 연결하려면 다음을 수행하십시오.

1. 콘텐츠 서버 오브젝트를 작성하십시오.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM();
```

C++

```
DKDatastoreICM *dsICM =new DKDatastoreICM();
```

2. 연결 매개변수를 설정하십시오.

Java

```
String database = "icmnlsdb";  
String userName = "icmadmin";  
String password = "password";
```

C++

```
char * database = "icmnlsdb";  
char * userName = "icmadmin";  
char * password ="password";
```

3. 콘텐츠 서버의 연결 조작을 호출하십시오. `databaseNameStr`은 연결할 데이터베이스의 이름입니다.

Java

```
dsICM.connect(databaseNameStr,usridStr,pwStr,"");
```

C++

```
dsICM->connect(database, userName, password, "");
```

시스템 구성에 따라 여러 개의 라이브러리 서버 및 자원 관리자에 연결할 수 있습니다. 연결할 수 있는 라이브러리 서버 이름 목록을 보려면 `DKDatastoreICM`을 사용하여 `listDataSourceNames()` 메소드와 `listDataSources()` 메소드를 호출하십시오. `listDataSources()` 메소드는 현재 연결에 사용 가능한 라이브러리 서버를 나열합니다.

라이브러리 서버에 연결한 후 라이브러리 서버와 연관된 자원 관리자 목록을 가져오려면 `DKRMConfiguration`을 사용하여 `listResourceMgrs()` 메소드를 호출하십시오.

시스템에서 연결 해제하려면 콘텐츠 서버에서 `disconnect` 연산을 호출하십시오.

자세한 정보는 위의 코드 스니펫이 추출된 전체 샘플인 `SConnectDisconnectICM`을 참조하십시오.

암호 변경

새 라이브러리 서버 세션을 시작할 때마다 사용자가 암호를 변경하게 할 수 있습니다. 암호 변경 옵션을 구현하려면 `DKDatastoreICM`을 사용하여 `changePassword()` 메소드를 호출하십시오.

Java

```
changePassword(String userID, String oldPwd, String newPwd)
```

C++

```
changePassword(const char* userID, const char* oldPwd, const char* newPwd)
```

항목에 대한 작업

이 절에서는 항목 작성, 갱신 및 삭제에 관련된 프로세스에 대해 설명합니다.

항목에 대한 작업 정보는 SItemCreationICM 샘플을 참조하십시오.

항목 유형 작성

항목을 작성하기 전에 항목 유형을 작성해야 합니다. 작성한 모든 항목에 대해 항목 유형을 정의해야 합니다. 예를 들어, claim 항목은 policy 항목 유형의 하위 구성요소입니다.

항목 유형을 작성하면 항목 유형에 대해 분류를 정의할 수 있습니다. 항목 유형 분류는 해당 항목 유형의 항목을 보다 자세히 식별하는 항목 유형 내의 카테고리입니다. 동일한 항목 유형을 가진 모든 항목은 동일한 항목 유형 분류를 갖습니다. Content Manager는 항목, resourceitem, docmodel 및 docpart와 같은 항목 유형 분류를 제공합니다. 항목 유형을 작성할 때 항목 유형 분류를 지정하지 않은 경우, 항목 유형 분류의 기본 값은 항목(DDO)이 됩니다. 미리 정의된 항목 유형 분류 및 해당 ID 상수는 표 10에 나열되어 있습니다.

표 10. 항목 유형 분류

분류	ID 상수	번호	설명
항목	DK_ICM_ITEMTY PE_CLASS_ITEM	0	표준 항목(DDO)
자원	DK_ICM_ITEMTY PE_CLASS_RESOURCE_ITEM	1	자원 관리자에 저장된 데이터를 설명하고 포함하는 자원 항목. 비디오, 이미지, 문서 및 자원 관리자에 기타 아카이브된 데이터가 자원 항목의 예입니다.
문서 모델	DK_ICM_ITEMTY PE_CLASS_DOC_MODEL	2	부분을 사용하여 문서를 모델링하는 항목. 문서는 속성 DKConstant.DK_CM_DKPARTS에 포함된 여러 부분으로 구성됩니다.

표 10. 항목 유형 분류 (계속)

분류	ID 상수	번호	설명
부분	DK_ICM_ITEMTY PE_CLASS_DOC_ PART	3	문서 모델 분류의 항목(부분)

주: 상수는 Java용 com\ibm\mm\sdk\common\DKConstantICM.java 및 C++용 dk\icm\DKConstantICM.h에 있습니다.

항목 유형을 작성하려면 다음을 수행하십시오(아래의 코드 예제 참조).

1. 항목 유형을 작성하여 이를 콘텐츠 서버에 대한 참조로 전달하십시오.
2. 항목 유형에 이름을 지정하십시오.
3. 항목 유형에 속성을 추가하고 nullable, textsearchable 및 unique와 같은 원하는 규정자를 설정하십시오.
4. 항목 유형을 지속 콘텐츠 서버에 추가하십시오.

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();
```

자세한 정보는 SitemTypeCreationICM 샘플을 참조하십시오.

항목 유형 나열

정의되어 사용 가능한 항목 유형 목록을 얻으려면 다음을 수행하십시오.

1. DKDatastoreICM 콘텐츠 서버에 연결하십시오.
2. 콘텐츠 서버 정의에 대한 참조를 가져오십시오.
3. 콘텐츠 서버 정의 오브젝트에서 listEntityNames 메소드를 호출하여 항목 유형 이름에 대한 문자열 배열을 가져오십시오.
4. 루프를 사용하여 모든 이름을 나열하십시오.

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++ 예제 1

```
long larraySize = 0;
DKString * itemTypeNames = dsICM->listEntityNames(larraySize);
for (int i = 0; i < larraySize; i++) {
    cout <<(char*)itemTypeNames[i] <<endl;
}
delete [] itemTypeNames;
```

C++ 예제 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);
```

자세한 정보는 SItemTypeRetrievalICM 샘플을 참조하십시오.

속성 작성

속성을 작성하려면 다음을 수행하십시오.

1. 속성 정의 오브젝트를 작성하십시오.
2. 이름, 설명, 유형, 크기 등을 설정하여 작성하는 오브젝트에 대해 설명하십시오. 작성할 수 있는 속성 유형 예제는 다음과 같습니다.
 - 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. 새 정의를 지속 콘텐츠 서버에 추가하십시오.

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();
```

속성 작성에 대한 자세한 정보는 SAttributeDefinitionCreationICM 샘플을 참조하십시오.

속성 그룹의 작성, 갱신 및 삭제

속성 그룹을 사용하면 전체 속성 그룹을 항목 및 하위 구성요소에 쉽게 추가할 수 있습니다. 속성은 원하는 대로 여러 속성 그룹(0 이상)에 포함될 수 있습니다. 속성은 여러 속성 그룹에 추가될 수 있습니다. 데이터 항목(DDO)은 여러 속성 그룹을 포함할 수 있습니다. 동일한 속성 이름이 DDO에 나타날 수 있지만 각 속성은 이름 공간에 따라 완전히 다릅니다. 속성이 속성 그룹에 추가되면 속성은 추가 후에 작성된 구성요소 유형에만 영향을 미칩니다. 기존 구성요소 유형은 변경되지 않고 그대로 있습니다. 다음 예제에서는 속성 그룹을 작성하는 방법에 대해 설명합니다

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);
```

속성 그룹에 대한 설명 및 이름을 갱신하려면 DKAttrGroupDefICM에서 update() 메소드를 호출하고 새 속성 ID 배열을 제공하십시오. 이 속성 그룹이 이미 구성요소 유형과 연관된 경우, 이 속성 그룹을 갱신할 수 없습니다.

속성 그룹을 삭제하려면 DKAttrGroupDefICM 클래스로 작업해야 합니다. 속성 그룹을 삭제하면 속성 그룹을 구성하는 데 사용되는 기본 속성은 라이브러리 서버에 남아 있습니다. 속성 그룹을 삭제할 때 다음 예외가 적용됩니다.

- 구성요소 유형과 연관되어 있고 지속적인 경우, 속성 그룹을 삭제할 수 없습니다.
- 속성 그룹이 구성요소 유형과 연관되어 있고 지속적인 경우, 속성을 속성 그룹에서 제거할 수 없습니다.
- 속성 그룹이 구성요소 유형과 연관되어 있고 지속적인 경우, 속성을 속성 그룹에 추가할 수 없습니다.

자세한 정보는 SAttributeGroupDefCreationICM 샘플을 참조하십시오.

컨텐츠 서버에 속성 나열

다음 예제에서는 컨텐츠 서버에서 속성 목록을 가져오는 방법에 대해 설명합니다.

Java

```
DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM(dsICM.datastoreDef());
//Get a collection containing all 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 containing all 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);
```

항목 유형에 대한 속성 이름 나열

다음 예제에서는 항목 유형에 대한 속성 이름 목록을 가져오는 방법에 대해 설명합니다. 자세한 정보는 SItemTypeInfoRetrieval API 교육 샘플을 참조하십시오.

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("\nAttributes 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 <<"\nAttributes 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);
```

항목 작성

항목은 최소한 하나의 루트 구성요소를 가진 구성요소 DDO의 전체 트리입니다. 루트 구성요소 DDO는 항목 등록 정보 유형 또는 의미 유형에 할당되어야 합니다. 항목을 작성할 때 적절한 항목 유형을 할당해야 합니다. 올바른 항목 유형 등록 정보는 문서, 폴더 또는 항목입니다. 항목 유형 등록 정보를 콘텐츠 서버의 createDDO 함수에 대한 두 번째 매개변수로 지정해야 합니다. 등록 정보 유형의 값은 DDO의 등록 정보 DK_CM_PROPERTY_ITEM_TYPE에 저장됩니다. 이 항목 유형 등록 정보를 항목의 구조를 설명하는 전체적 항목 유형 정의와 혼동하지 마십시오. 172 페이지의 표 11에서는 사용 가능 항목 등록 정보 유형 목록을 보여줍니다.

표 11. 항목 유형 등록 정보 정의

등록 정보 유형	상수	정의
문서	DK_CM_DOCUMENT	항목은 문서 또는 저장된 데이터를 나타냅니다. 이 항목에 포함된 정보는 문서를 형성할 수 있습니다. 이 항목은 반드시 특정 문서 모델의 구현을 의미하지는 않기 때문에 공통 문서로 간주될 수 있습니다.
폴더	DK_CM_FOLDER	항목은 컨텐츠나 오브젝트를 포함하거나 참조하는 오브젝트를 나타냅니다. 이 항목은 반드시 특정 문서 모델의 구현을 의미하지는 않기 때문에 공통 폴더로 간주될 수 있습니다.
항목(기본값)	DK_CM_ITEM	일반 항목. 시스템 정의 또는 사용자 정의 의미 유형에 맞지 않는 항목.

항목은 DKDDO로 작성됩니다. 항상 DKDatastoreICM의 createDDO() 메소드를 사용하여 DKDDO를 작성하십시오. 시스템은 DKDatastoreICM의 createDDO 메소드를 사용하여 DKDDO 구조의 주요 정보를 자동으로 설정하기 때문입니다.

항목을 작성할 때 해당 항목을 구성하는 구성요소를 정의합니다. 예를 들어, 계층 구조 항목을 작성하도록 선택할 경우 하위 구성요소를 작성해야 합니다.

1. 컨텐츠 서버의 createDDO 및 createChildDDO 메소드를 사용하여 항목 DDO를 작성하고 모든 속성 및 기타 필수 정보를 설정하십시오. 이 예제에서는 dsICM이라는 로그인 및 연결된 DKDatastoreICM 오브젝트를 사용합니다.
2. 루트 구성요소를 작성하십시오.

Java 예제 1

```
DKDDO myDocumentDDO = dsICM.createDDO("EmployeeDoc",
    DKConstantICM.DK_CM_DOCUMENT);
```

Java 예제 2

```
DKDDO myFolderDDO = dsICM.createDDO("DeptFolder",
    DKConstantICM.DK_CM_FOLDER);
```

C++ 예제 1

```
DKDDO* myDocumentDDO = dsICM->createDDO("EmployeeDoc",DK_CM_DOCUMENT);
```


C++ 예제 2

```
DKDDO* myFolderDDO = dsICM->createDDO("DeptFolder",DK_CM_FOLDER);
```

3. 루트 구성요소 "EmployeeDoc" 아래에 하위 구성요소를 작성합니다(예: "Dependent").

Java

```
DKDDO myDDO =dsICM.createChildDDO("EmployeeDoc","Dependent");
```

C++

```
DKDDO* myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

4. 하위 구성요소를 상위 구성요소에 추가하십시오.

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 메소드를 사용하여 DDO를 적당한 값으로 채우십시오.

Java

```
myDDO.setData(.....);
```

C++

```
myDDO->setData(.....);
```

6. 지속적 스토어에 항목을 저장하십시오.

Java

```
myDocumentDDO.add();
```

C++

```
myDocumentDDO->add();
```

이전 예제에서는 마지막 단계에서 콘텐츠 서버에 문서를 작성했습니다. 문서 DDO를 콘텐츠 서버에 추가하는 경우, DKParts 컬렉션 내의 모든 부분을 비롯하여 해당 속성이 모두 추가됩니다. 문서 DDO를 콘텐츠 서버에 추가하는 경우, DKParts 컬렉션 내의 모든 부분과 모든 하위 부분을 비롯하여 해당 속성이 모두 추가됩니다.

의미 유형은 항목에 대한 사용법 또는 규칙을 정의합니다. Content Manager는 일부 미리 정의된 의미 유형을 함께 제공하지만 사용자 고유의 의미 유형을 정의할 수도 있습니다.

의미 유형을 콘텐츠 서버의 createDDO 함수에 대한 두 번째 매개변수로 지정할 수 있습니다. 의미 유형값은 DDO의 등록 정보 DKConstantICM.

DK_ICM_PROPERTY_SEMANTIC_TYPE에 저장됩니다. 이 등록정보는 항목 등록 정보 유형과 동일한 값을 포함할 수 있습니다. ICM 커넥터는 폴더 및 문서 의미 유형을 지원합니다. 사용자 고유의 의미 유형을 작성할 수도 있습니다. 표 12에는 사용 가능한 의미 유형이 나열되어 있습니다.

표 12. 미리 정의된 의미 유형

의미 유형	상수	정의
문서	DK_ICM_SEMANTIC_TYPE_DOCUMENT	이 항목이 문서임을 표시합니다.
폴더	DK_ICM_SEMANTIC_TYPE_FOLDER	이 항목이 폴더임을 표시합니다.
주석	DK_ICM_SEMANTIC_TYPE_ANNOTATION	항목 또는 부분이 기본 부분의 주석임을 표시합니다.

표 12. 미리 정의된 의미 유형 (계속)

의미 유형	상수	정의
컨테이너	DK_ICM_SEMANTIC_TYPE _CONTAINER	이 항목이 다른 항목을 포함하는 컨테이너임을 표시합니다. 링크를 사용하여 컨테이너(container)-컨테이너(containee) 관계를 나타냅니다.
사용 내용	DK_ICM_SEMANTIC_TYPE _HISTORY	이 항목 또는 부분이 기본 부분의 사용 내용을 표시합니다.
메모	DK_ICM_SEMANTIC_TYPE _NOTE	이 항목 또는 부분이 기본 부분의 사용 내용을 표시합니다.
기본	DK_ICM_SEMANTIC_TYPE _BASE	이 항목 또는 부분이 연관된 주석, 메모, 사용 내용을 가지고 있는 기본 부분임을 표시합니다.
사용자 정의	사용자 정의	사용자 정의 의미 유형은 응용프로그램에서 해석합니다.

주:

- Java에서 상수는 DKConstantICM.java에 정의됩니다.
- C++에서 상수는 DKConstantICM.h에 정의됩니다.
- Content Manager 버전 7에서 의미 유형을 관계 유형이라고도 합니다.

자원 항목 유형으로 정의된 항목 유형에 항목을 작성할 경우, 올바른 XDO가 리턴됩니다. 자원에 대한 자세한 정보는 SResourceItemCreationICM 샘플을 참조하십시오. 이 샘플이 유용하다는 것을 알아서 항목 작성에 대한 정보를 SItemCreationICM 샘플에서 검토할 수도 있습니다.

항목 속성 값 설정 및 검색

Java

속성값은 java.lang.Objects로서 저장되고 검색됩니다. 속성값을 설정 또는 검색하려면 DDO의 setData 및 getData() 메소드를 각각 사용하십시오. 속성 유형에 해당하는 유형의 오브젝트를 사용하여 값을 설정하십시오. **예제:**

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,nameOfAttrStr), valueObj);
Object obj = ddo.getData(ddo.dataId(DK_CM_NAMESPACE_ATTR,nameOfAttrStr));
```

위 명령문은 속성값을 java.lang.Object valueObj로 전달된 값으로 설정합니다. nameOfAttrStr은 속성의 문자열 이름입니다. 이 속성은 DDO 항목 유형이 정의되었을 때 항목 유형에서 정의되고 지정되었습니다. valueObj는 반드시 설정해야 하는 값이고 이 속성의 올바른 유형이어야 합니다.

C++

개별 속성의 값을 설정할 때 개별 속성 정의 이름을 사용하십시오. 속성 그룹에 속한 속성에 액세스하려면 <Attribute Group Name>.<Attribute Name> 형식을 사용하십시오. **예제:**

```
//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")));
```

항목의 속성 수정

항목의 속성을 수정하려면 DDO의 setData 메소드를 사용하고 아래 예제에 표시된 것처럼 java.lang.Object를 사용하여 해당 값을 지정함으로써 필수 값으로 설정하십시오. 아래 예제에서 nameOfAttrStr은 속성의 문자열 이름이며 valueObj는 값입니다.

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++"));
```

항목 갱신

항목을 갱신하려면 다음을 수행하십시오.

1. 항목을 검색하거나 항목을 작성한 다음 콘텐츠 서버에 추가하십시오.
2. 항목, 속성, 링크 콜렉션 등을 수정하십시오.
3. DDO의 update 연산을 호출하십시오.

Java

```
ddo.update();
```

C++

```
ddo->update();
```

항목에 버전화 기능이 사용되면 다음 예제에 표시된 것처럼 현재 항목을 갱신하는 대신 항목의 새 버전을 작성할 수 있습니다.

Java

```
ddo.update(DKConstant.DK_CM_VERSION_NEW);
```

C++

```
ddo->update(DK_CM_VERSION_NEW);
```

항목의 최신 버전을 갱신하려면 다음 예제의 형식을 사용하십시오.

Java

```
ddo.update(DKConstant.DK_CM_VERSION_LATEST);
```

C++

```
ddo->update(DK_CM_VERSION_LATEST);
```

또한 아래 예제에 표시된 것처럼 적절한 옵션과 함께 DKDatastoreICM의 updateObject 메소드를 호출하여 DDO를 갱신할 수도 있습니다.

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);
```

중요사항: 갱신 메소드를 호출하기 전에 반드시 항목을 체크아웃하고 갱신을 완료한 다음 항목을 다시 체크인해야 합니다. 185 페이지의 『항목 체크인 및 체크아웃』을 참조하십시오. 항목 갱신에 대한 자세한 정보는 SItemUpdateICM API 교육 샘플을 참조하십시오.

자원 항목 유형 정의

자원 항목은 항목을 나타내는 오브젝트의 위치, 유형, 크기 등을 정의하는 추가 시스템 정의 속성을 가진 항목입니다. 오브젝트를 때때로 "자원"이라고도 하며 비디오 파일, 이미지, 워드 프로세서 문서 등이 될 수 있습니다. 자세한 정보는 SItemTypeCreationICM 샘플(샘플 디렉토리에 있음)을 참조하십시오.

아래 단계는 자원 항목 유형을 정의하는 과정을 안내합니다.

1. 연결된 콘텐츠 서버에서 콘텐츠 서버 정의 오브젝트를 가져오십시오.

Java

```
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
```

C++

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
```

2. 새 항목 유형 정의를 작성하십시오.

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. 속성을 추가하십시오.

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. XDO 클래스 및 이 항목 유형에 대한 자원의 유형을 지정하십시오.

Java

```
itemType.setXDOClassName(DKConstantICM.DK_ICM_XDO_LOB_CLASS_NAME);
itemType.setXDOClassID(DKConstantICM.DK_ICM_XDO_LOB_CLASS_ID);
itemType.add();
```

C++

```
itemType->setXDODClassName(DK_ICM_XDO_LOB_CLASS_NAME);  
itemType->setXDODClassID(DK_ICM_XDO_LOB_CLASS_ID);  
itemType->add();
```

자원 항목 작성

자원 항목 작성은 일반 항목 작성과 아주 비슷합니다. XDO는 DDO를 확장하고 자원 항목 유형에 따라 나중에 추가로 확장될 수 있습니다. 표 13에는 자원 항목 유형과 이들을 작성하는 데 사용되는 클래스 계층 구조가 나와 있습니다. 자세한 정보는 SResourceItemCreationICM 샘플(샘플 디렉토리에 있음)을 참조하십시오.

표 13. 자원 항목 유형 클래스 계층 구조

유형	DDO	XDO	확장자
LOB	DKDDO ->	DKLobICM	
텍스트	DKDDO ->	DKLobICM ->	DKTextICM
이미지	DKDDO ->	DKLobICM ->	DKImageICM
스트림	DKDDO ->	DKLobICM ->	DKStreamICM

아래 단계는 자원 항목을 작성하는 과정을 안내합니다. 자원 콘텐츠를 설정 및 저장하는 방법에는 여러 가지가 있습니다. 다음 프로세스는 항목이 콘텐츠 서버에 추가될 때마다 파일에서 직접 저장하는 방법에 대한 예제입니다.

1. 자원 항목을 작성하십시오. 자원 항목은 임의의 의미 유형이 될 수도 있고, 일반 항목을 작성하는 데 사용된 것과 동일한 방법으로 DKDatastoreICM::createDDO 호출이 자원 항목을 작성하는 데도 사용된다는 점에 유의하십시오. DKDatastoreICM.createDDO에서 리턴된 자원 유형은 올바른 서브클래스(이 경우에는 DKLobICM)로 캐스트되며, 이 자원 항목이 기반으로 하는 항목 유형의 작성 중에 정의된 XDO 분류에 기반을 두고 있습니다.

Java

```
DKLobICM lob = (DKLobICM)dsICM.createDDO  
("SampleResource",DKConstant.DK_CM_DOCUMENT);
```


C++

```
DKLobICM* lob = (DKLobICM*)  
dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. 콘텐츠 또는 데이터를 오브젝트로 설정하십시오. 일단 lob가 작성되면 자원에 대해 MIME 유형을 설정할 수 있습니다. 이 경우 자원은 MS Word 문서입니다. MIME 유형은 저장된 콘텐츠 유형을 설명합니다.

Java

```
lob.setContentFromClientFile(fileName);  
lob.setMimeType("application/msword");
```

C++

```
lob->setContentFromClientFile(fileName);  
lob->setMimeType("application/msword");
```

MIME 유형에 대한 자세한 정보는 SResourceItemMimeTypesICM 샘플을 참조하십시오.

3. 데이터를 콘텐츠 서버에 추가하십시오. 이 경우 샘플 Word 문서입니다. 자원 항목 콘텐츠는 자원 관리자에 저장된다는 점에 유의하십시오.

Java

```
lob.add("SResourceItemICM_Document1.doc");
```

C++

```
lob->add("SResourceItemICM_Document1.doc");
```

중요사항: C++에서 메모리를 제거해야 합니다.

```
delete(lob);
```

자세한 정보는 SItemUpdateICM 샘플을 참조하십시오.

항목 검색

기준 세트에 일치하는 항목을 찾으려면 다음 단계를 완료하십시오.

1. DKDatastoreICM 오브젝트에 연결하십시오.
2. 올바른 조회를 처리하십시오. 조회는 조회 언어를 따라야 합니다. 자세한 정보는 211 페이지의 『조회 언어 이해』를 참조하십시오.
3. 조회 결과를 DKResult 오브젝트에 저장하십시오.

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,
    DKConstant.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);
```

자세한 정보는 SSearchICM 샘플을 참조하십시오.

항목 검색

명시적으로 항목을 검색하거나 조회를 통해 검색된 항목을 화면 갱신하려면 검색하거나 화면 갱신할 항목의 PID 오브젝트 또는 PID 문자열에 액세스해야 합니다. ItemID만을 알 경우, 조회를 수행하여 전체 PID 정보를 검색할 수 있습니다. 항목 유형 이름 및 항목 ID를 가정하면, 다음 시나리오에서는 명시적으로 검색하려는 경우에 DDO를 검색하는 방법을 보여줍니다.

Java

1. datastoreICM 오브젝트에 연결한 후 해당 항목 ID를 사용하여 항목을 검색하십시오. 조회 작성에 대한 자세한 정보는 211 페이지의 『Content Manager 서버 조회』를 참조하십시오.
2. 조회 결과를 DKResults 오브젝트에 저장하십시오. 아래 코드 예제에서 queryStr은 올바른 조회 언어 형식의 검색 문자열입니다. DKResults results = (DKResults)dsICM.evaluate(queryStr,DKConstantICM.DK_CM_XQPE_QL_TYPE, null);
3. DKResults에서 반복자를 작성하고 이를 DDO를 가져오는 데 사용하십시오. 이것은 이제 하나씩 검색할 수 있습니다.
`ddo.retrieve();`
4. 이 예제에 표시된 것처럼 DDO에서 얻은 PID 문자열이 있다고 가정하십시오.
`DKPidICM pid = ddo.getPidObject();
String pidStr = pid.pidString();`

PID 문자열을 사용하여 다음 단계를 완료함으로써 검색을 수행할 수 있습니다.

1. DKDatastoreICM의 createDDO 메소드를 사용하여 DDO를 작성하십시오. DKDDO 구성자를 사용하지 마십시오. 아래 예제에서 dsICM 오브젝트는 이미 Content Manager 데이터스토어에 연결되어 있습니다. 또한 PID는 이름이 pidStr인 문자열입니다.
`DKDDO ddo = dsICM.createDDO(pidStr);`
2. DDO의 검색 연산을 호출하십시오.
`ddo.retrieve();`

C++

1. datastoreICM 오브젝트에 연결한 후 해당 항목 ID를 사용하여 항목을 검색하십시오. 조회 작성에 대한 자세한 정보는 Content Manager 서버 조회를 참조하십시오.
2. 조회 결과를 DKResult 오브젝트에 저장하십시오. 아래 코드 예제에서 queryStr은 올바른 조회 언어 형식의 검색 문자열입니다.

```
DKResults* results = (DKResults*)(dkCollection*)
dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE,NULL);
```

3. DKDatastoreICM의 createDDO 메소드를 사용하여 DDO를 작성하십시오. DKDDO 구성자를 사용하지 마십시오. 아래 예제에서 dsICM 오브젝트는 이미 데이터스토어에 연결되어 있습니다. 또한 PID는 이름이 pidStr인 문자열입니다.

```
DKDDO* ddo = dsICM->createDDO(pidStr);
```

4. DDO의 검색 연산을 호출하십시오.

```
ddo->retrieve();
```

항목에 버전화 기능이 사용되면 적절한 옵션과 함께 DKDDO 클래스의 검색 메소드를 사용하여 항목의 특정 버전을 검색할 수 있습니다. 항목의 최신 버전을 검색하려면 검색 옵션으로 상수 DK_CM_VERSION_LATEST를 사용하십시오. 기본적으로(옵션이 지정되지 않은 경우) ddo.retrieve()는 항목의 최신 버전을 검색합니다.

예제:

```
DKDDO ddo = ds.createDDO(...);
.... ddo.retrieve(DK_CM_VERSION_LATEST);
```

또한 DKDatastoreICM 오브젝트의 retrieveObject 메소드를 호출하여 DDO를 검색할 수도 있습니다.

예제:

```
DKDatastoreICM ds =new DKDatastoreICM();
//connect,etc ...
DKDDO ddo =ds.createDDO(itemType,option);
//sets the PID as shown above...
ds.retrieveObject(ddo);
```

항목 검색에 대한 자세한 정보는 SItemRetrievalICM 및 SResourceItemRetrievalICM API 교육 샘플을 참조하십시오.

항목에 버전화 기능이 사용되고 해당 속성과 하위 항목을 포함하여 최신 버전을 검색하려면 다음 형식을 사용하십시오.

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);
```

검색에 대한 자세한 정보(예: 검색 옵션)는 SItemRetrievalICM 샘플을 참조하십시오.

항목 삭제

컨텐츠 서버에서 항목을 삭제하려면 DDO의 삭제 메소드를 사용하십시오.

Java

```
ddoDocument.del();
```

C++

```
ddoDocument->del();
```

DDO에는 항목 ID와 오브젝트 유형 세트가 있어야 하고 컨텐츠 서버에 올바르게 연결되어 있어야 합니다.

항목 삭제에 대한 자세한 정보는 SItemDeletionICM API 교육 샘플을 참조하십시오.

항목 체크인 및 체크아웃

두 사용자가 동시에 동일한 항목을 변경하지 못하도록 하려면 갱신하기 전에 항목을 체크아웃해야 합니다. 항목을 체크아웃하면 항목을 체크아웃한 사용자에게 독점 갱신 권한이 부여됩니다. 항목을 체크아웃하면 해당 항목은 지속 쓰기 잠금 상태가 됩니다. 전체 항목은 모든 버전 및 하위 구성요소를 비롯하여 전반적으로 잠금 또는 잠금 해제되어 있습니다. 그러나 링크되고 참조된 항목은 항목으로 잠기지 않습니다. 항목의 지속 잠금은 항목을 체크인할 경우 해제됩니다.

항목을 체크인 및 체크아웃하려면 아래 예제에 표시된 것처럼 DKDatastoreICM의 checkIn 및 checkOut 연산을 사용하십시오.

1. dsICM이라는 DKDatastoreICM 오브젝트에 연결한 후 myDataObject라는 DDO를 사용하여 항목을 체크아웃하십시오.

Java

```
dsICM.checkOut(myDataObject);
```

C++

```
dsICM->checkOut(myDataObject);
```

2. 항목을 체크인하십시오.

Java

```
dsICM.checkIn(myDataObject);
```

C++

```
dsICM->checkIn(myDataObject);
```

항목 체크인 및 체크아웃에 대한 자세한 정보는 SItemUpdateICM API 교육 샘플을 참조하십시오.

항목의 버전화 등록 정보 설정 및 가져오기

아래 예제에서는 항목 버전화 등록 정보를 설정하고 가져오는 방법에 대해 설명합니다.

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);
```

버전화 등록 정보에 대한 작업

이 절에서는 버전화 등록 정보에 대한 작업을 도와주는 예제를 제공합니다.

항목 유형의 버전 제어 방침 검색

항목은 버전 제어 방침을 가지고 있고 이 방침에는 항목에 대한 버전화 등록 정보가 포함되어 있습니다. 다음은 사용 가능한 세 가지 버전화 등록 정보 목록과 버전 제어 방침의 각 등록 정보를 나타내는 데 사용되는 값입니다.

DK_ICM_VERSION_CONTROL_ALWAYS

항상 버전화.

DK_ICM_VERSION_CONTROL_NEVER

이 항목 유형에는 버전화가 지원되지 않습니다(기본값).

DK_ICM_VERSION_CONTROL_BY_APPLICATION

응용프로그램이 버전화 설계를 판별합니다.

Java

```
short versionControlPolicy;
DKItemTypeDefICM item = newDKItemTypeDefICM();
....
versionControlPolicy = item.getVersionControl();
```

C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item = NULL;
...
versionControlPolicy = item->getVersionControl();
```

항목 유형에 대한 버전 제어 설정**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);
```

시스템에서 항목 유형 정의에 대한 버전 제어 정보에 액세스하는 전체 샘플에 대해서는 SItemTypeRetrievalICM API 교육 샘플을 참조하십시오.

항목 유형에 허용된 최대수의 버전 가져오기**Java**

```
short versionMax;
DKItemTypeDefICM item = new DKItemTypeDefICM();
....
versionMax = item.getVersionMax();
```


C++

```
short versionMax      = 0;
DKItemTypeDefICM * item = NULL;
....
versionMax = item->getVersionMax();
```

항목 유형에 허용된 최대수의 버전 설정

이 예제에서 시스템은 이 항목 유형에 기초한 항목 중 10개의 버전만 유지보수합니다.

Java

```
short versionMax=10;
DKItemTypeDefICM item = new DKItemTypeDefICM();
...
item.setVersionMax(versionMax);
```

C++

```
short versionMax      = 10;
DKItemTypeDefICM * item = NULL;
....
item->setVersionMax(versionMax);
```

항목 유형의 버전화 유형 값 설정

C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

폴더에 대한 작업

폴더는 항목을 나타내는 완전 지원되는 DDO입니다. 폴더는 작성된 항목 유형의 전체 계층 구조 데이터 구조를 가집니다. 폴더는 의미 유형 폴더의 DDO이고, 항목 유형의

분류에 상관없이 DKFolder를 포함하는 DKConstant.DK_CM_DKFOLDER 속성을 가집니다. DKFolder 오브젝트, DKSequentialCollection은 다른 항목을 표시하는 모든 루트 구성요소 DDO를 포함할 수 있습니다.

아래의 프로시저에 있는 코드 단편은 단지 정보용입니다. 이 코드 단편은 현재 형식으로는 작동하지 않으므로 그대로 복사하여 응용프로그램에 직접 붙여넣지 마십시오. 실행할 수 있는 완전한 폴더 샘플에 대해서는 samples 디렉토리의 SFolderICM 샘플을 참조하십시오.

폴더 작성

DKDatastoreICM.createDDO() 메소드는 런타임 시 DDO를 작성하는 데 사용됩니다. SItemCreationICM 샘플(sample 디렉토리에 있음)에 표시된 것처럼 폴더 항목을 작성할 때 항목 등록 정보 유형 또는 의미 유형은 DKConstant.DK_CM_FOLDER로 지정되어야 합니다.

폴더 항목은 DK_CM_FOLDER 의미 유형의 DKDatastoreICM.createDDO 메소드를 사용하여 작성되었습니다. DKConstant.DK_CM_FOLDER를 createDDO 메소드의 두 번째 매개변수로 지정하여 폴더를 작성할 수 있습니다.

Java

```
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
```

C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

ddoFolder를 채우려면 setData 메소드를 사용하십시오. 일단 폴더 DDO가 작성되면 지속 콘텐츠 서버에 저장됩니다.

Java

```
ddoFolder.add();
```

C++

```
ddoFolder->add();
```

폴더에 콘텐츠 추가

폴더 DDO에서 add() 또는 update() 메소드를 호출하려면 먼저 DKFolder에 있는 모든 콘텐츠가 콘텐츠 서버에서 유지되어야 합니다. 콘텐츠가 유지되게 하려면 DKDDO.add() 메소드를 호출하십시오.

폴더에 항목을 추가하려면 DKFolder의 addElement() 함수를 사용하십시오. 폴더에 충분한 구성원을 추가했으면 add 또는 update 메소드를 호출하여 폴더-콘텐츠 관계를 지속 스토어에 저장할 수 있습니다. 그러면 여러 폴더 수정사항을 묶어서 라이브러리 서버로 한 번 호출합니다. 폴더에 항목을 추가할 때는 동일한 항목의 여러 사본을 DKFolder에 추가하지 마십시오. 모든 폴더 수정사항이 추적되었으므로 수정사항의 중복이나 충돌을 야기시키지 마십시오. 예를 들어, 동일한 항목의 여러 사본을 폴더에 추가하지 마십시오.

폴더에 콘텐츠를 추가하려면 다음 단계를 수행하십시오. 폴더는 다른 폴더(하위 폴더)를 콘텐츠 항목 중 하나로 포함할 수 있습니다.

Java

1. 세 개의 새 항목을 작성하십시오. 이들 항목은 폴더에 콘텐츠로 추가됩니다. 폴더 콘텐츠는 임의의 의미 유형일 수 있습니다.

```
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. DDO를 채우려면 setData 메소드를 사용하십시오. 폴더 콘텐츠로 추가되는 항목은 데이터스토어에서 유지되어야 합니다.

```
ddoDocument.add();  
ddoItem.add();  
ddoFolder2.add();
```

3. 이전에 작성되었고 DDO 폴더에 유지된 dkFolder 속성을 검색하십시오.

```
short dataid = ddoFolder.dataId  
    (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);  
dkFolder = (DKFolder) ddoFolder.getData(dataid);
```

4. 폴더는 갱신하기 전에 콘텐츠를 추가하여 데이터스토어에서 체크아웃해야 합니다.

```
dsICM.checkOut(ddoFolder);
```

5. 이전에 작성된 항목을 폴더에 추가하십시오.

```
dkFolder.addElement(ddoDocument);  
dkFolder.addElement(ddoItem);  
dkFolder.addElement(ddoFolder2);
```

6. 폴더의 변경사항을 파악한 후 변경사항을 체크인하십시오.

```
ddoFolder.update();  
dsICM.checkIn(ddoFolder);
```

C++

1. 폴더에 추가될 항목을 작성하십시오.

```
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. 작성한 항목을 데이터스토어에 유지하십시오.

```
ddoDocument->add();
ddoItem->add();
ddoFolder2->add();
```

3. 폴더에 대한 DKFolder 속성을 검색하십시오.

```
DKFolder * dkFolder;
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER);
if (dataid!=0) dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
```

4. 폴더를 체크아웃하십시오.(폴더는 갱신하기 전에 반드시 체크아웃해야 합니다.)

```
dsICM->checkOut(ddoFolder);
```

5. 작성된 항목을 폴더에 추가하십시오.

```
dkFolder->addElement(ddoDocument);
dkFolder->addElement(ddoItem);
dkFolder->addElement(ddoFolder2); // Folders can contain sub-folders.
```

6. 폴더를 갱신하십시오. 그러면 암시적으로 폴더도 체크인됩니다(잠금 해제).

```
ddoFolder->update();
```

7. 명시적으로 폴더를 체크인하십시오.

```
dsICM->checkIn(ddoFolder);
```

폴더에서 콘텐츠 제거

두 가지 중 하나의 방법으로 폴더에서 항목을 제거할 수 있습니다. 지연된 제거(실제 제거는 DDO 폴더가 갱신되고 여러 콘텐츠 서버 호출이 하나로 결합될 때 완료됨)는 `removeElementXX` 메소드를 사용하여 완료되었습니다. 즉시(그리고 여러 호출이 콘텐츠 서버에서 발생하기 때문에 비용이 드는) 제거는 `dkFolder.removeMember` 메소드를 사용하여 완료될 수 있습니다. 폴더에 대한 작업 정보는 `SFolderICM` 샘플(sample 디렉토리에 있음)을 참조하십시오.

폴더에서 콘텐츠를 제거하려면 다음 단계를 완료하십시오.

Java

1. 항목을 제거하려는 DDO 폴더를 체크아웃하십시오.

```
dsICM.checkOut(ddoFolder);
```

2. 제거할 항목을 찾으십시오. 이 경우 이전에 작성된 docItem DDO를 찾으십시오. 폴더 콘텐츠를 통해 반복하려면 반복자를 작성하십시오.

```
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 strings 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. 변경사항을 유지하고 DDO 폴더를 체크인하십시오.

```
ddoFolder.update();
dsICM.checkIn(ddoFolder);
```

C++

1. 항목을 작성하여 폴더에 추가하십시오. 이전에 폴더를 작성해야 합니다.

```
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. 명시적으로 폴더를 체크인하십시오.

```
dsICM->checkIn(ddoFolder);
```

3. 폴더에 대한 반복자를 작성하십시오.

```
dkIterator* iter = dkFolder->createIterator();
```

4. 제거될 항목을 찾을 때까지 폴더 콘텐츠를 통해 반복하십시오. 항목을 제거하십시오.

```
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);
```

폴더의 콘텐츠 검색

폴더 콘텐츠를 검색하려면 검색 옵션을 설정해야 합니다. 기본적으로 검색 옵션은 설정되지 않습니다. 검색 옵션이 설정되지 않은 경우, 검색이 호출되고 올바른 옵션이 설정될 때까지 오브젝트에는 DKFolder 또는 DK_CM_DKFOLDER 속성이 포함되지 않습니다. 검색 옵션은 다음을 포함합니다.

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

Outbound 링크가 지정된 DKFolder 속성 콜렉션으로 폴더 항목을 검색하려면 다음 형식을 사용하십시오.

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
```

항목 트리에 링크 및 폴더 콘텐츠가 포함되는 폴더 항목을 검색하려면 다음 형식을 사용하십시오.

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ITEMTREE);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

특정 DDO를 포함하는 모든 폴더 얻기

여러 폴더는 동일한 항목(DDO)을 포함할 수 있습니다. 그러면 여러 응용프로그램 또는 사용자와 항목을 연관시킬 수 있습니다. 현재 특정 DDO를 포함하는 폴더를 판별하려면 아래 예제를 참조하십시오.

다음 단계에서는 이전에 작성한 ddoDocument 오브젝트를 포함하는 폴더를 찾는 방법에 대해 설명합니다. 폴더에 대한 작업 정보는 SFolderICM 샘플(sample 디렉토리에 있음)을 참조하십시오.

Java

1. 데이터스토어 확장자 오브젝트를 검색하십시오.

```
DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
                        dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
```

2. ddoDocument 오브젝트가 포함된 폴더를 찾으려면 확장 오브젝트에서 getFoldersContainingDDO 메소드를 호출하십시오. 그러면 리턴된 각 DDO가 ddoDocument 오브젝트를 포함하는 폴더인 DDO 컬렉션을 리턴합니다.

```
DKSequentialCollection list = dsExtICM.getFoldersContainingDDO(ddoDocument);
```

3. 리턴된 폴더 컬렉션을 통해 순환할 반복자를 작성하십시오.

```
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;
```

항목 간의 링크 정의

링크를 사용하여 목표 항목에 대한 원본 항목을 선택적 설명 항목과 연관시킬 수 있습니다. DKLink 오브젝트에 링크를 정의하여 여기서 다음 유형의 링크 요소를 지정할 수 있습니다.

표 14. 링크 데이터 구조

DKLink	정의
LinkTypeName	링크 유형
원본	링크의 원본 항목
목표	링크의 목표 항목
LinkItem	설명 항목(선택적)

링크는 일대다 관계를 나타내므로 DKLinkCollection 값의 LINK 이름 공간 아래에서 데이터 항목으로 DDO를 나타냅니다. 링크에 대한 작업 정보는 samples 디렉토리의 SLinksICM 샘플을 참조하십시오.

링크 자체는 원본 또는 목표에 속하지 않습니다. 링크는 단순히 원본과 목표를 연결합니다. 예를 들어, 원본 A가 목표 B에 링크된 경우, 어떠한 DDO, A 또는 B가 참조되는지에 상관없이 A는 항상 원본이고 B는 항상 목표입니다.

메모리에서 원본 및 목표 DDO는 모두 동일한 DKLink 오브젝트 사본을 포함합니다. DKLink 오브젝트에 원본과 목표 모두에 대한 참조가 포함되기 때문에 링크가 포함된 DDO에는 다른 DDO뿐만 아니라 자체를 참조하는 링크도 포함됩니다. 예를 들어, 원본 A가 목표 B에 링크된 경우, A와 B는 모두 표 15의 예제에 표시된 것처럼 동일한 링크를 포함합니다.

표 15. DKLink 정의 예제

A - B에 대해 정의된 DKLink	DDO A	DDO B
원본은 A임	원본은 A임	원본은 A임
목표는 B임	목표는 B임	목표는 B임

지속 상점에서 링크를 추가하거나 제거할 때는 원본 또는 목표의 두 항목 중 하나에 대해서만 조작을 수행해야 합니다. 링크는 자동으로 다른 항목에 대해 갱신됩니다.

링크 및 실행할 수 있는 완전한 링크 샘플에 대한 자세한 정보는 samples 디렉토리의 SLinksICM 샘플을 참조하십시오.

인바운드 및 아웃바운드 링크

원본 또는 목표와 같은 특정 DDO를 참조하기 위해 링크를 사용할 때 링크가 인바운드 또는 아웃바운드되도록 고려할 수 있습니다. 특정 DDO에서 Perspective의 링크를 고려하는 경우, 해당 DDO가 목표라면 해당 DDO를 인바운드 링크로 고려합니다. 반면, 해당 링크의 반대에 있는 DDO는 해당 Perspective에서 아웃바운드될 동일한 링크를 고려합니다.

표 15의 예제에서 DDO A에서 DDO B로의 링크는 아웃바운드 링크인 반면, DDO B에 대한 동일한 링크는 인바운드 링크입니다.

링크 유형 이름

링크 관계는 이름을 가집니다. DDO 내에서 링크는 링크 컬렉션으로 그룹화됩니다. 시스템 정의 링크 유형 이름이 있으며 사용자 고유의 유형을 정의할 수도 있습니다. 원하는 만큼의 사용자 정의 링크 유형 이름을 사용하거나 시스템 정의 링크 유형 이름을 사용할 수 있습니다. 시스템 정의 링크 유형 이름은 다음을 포함합니다.

링크 유형 이름 포함

Java 상수: `DKConstant.DK_ICM_LINKTYPENAME_CONTAINS`

C++ 상수: `DK_ICM_LINKTYPENAME_CONTAINS`

링크 유형 이름: **DKFolder**

Java 상수: `DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER`

C++ 상수: `DK_ICM_LINKTYPENAME_DKFOLDER`

DKFolder 링크 유형 이름이 제공되며, 이 이름은 시스템이 폴더를 관리하는 데 사용됩니다. DKFolder 오브젝트는 아웃바운드 링크 컬렉션으로의 단순화된 인터페이스로서, 폴더에 대해 작업하기 쉽도록 만들어줍니다. 링크를 정의하거나 삭제하기 위해 DKFolder 링크 유형을 사용해서는 안됩니다. 또한 DKFolder 링크 유형 이름을 DKLinkCollection 컬렉션과 함께 어떤 방식으로든 사용해서는 안됩니다. 그러나 링크를 사용하여 폴더를 검색하려면 DKFolder 링크 유형 이름을 지정해야 합니다.

링크에 대한 자세한 정보는 `cmbroot\samples` 디렉토리의 `SLinksICM` 샘플을 참조하십시오. 사용자 정의 링크 유형 작성에 대한 정보는 `SLinkTypeDefinitionCreationICM` 샘플을 참조하십시오.

링크된 항목 검색

링크를 검색할 때 인바운드만, 아웃바운드만, 또는 둘다 검색하기 위한 옵션이 있습니다. 인바운드 및 아웃바운드를 검색하려면 검색 옵션을 설정해야 합니다. 링크를 검색하기 위해 다음 옵션을 설정할 수 있습니다.

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() 또는 update() 메소드를 호출하기 전에 링크와 연관된 모든 항목은 이미 지속적이라는 점을 기억하십시오.

액세스 제어에 대한 작업

사용자 고유의 관리 프로그램을 작성하기 위해 Content Manager API 또는 Content Manager 시스템 관리 클라이언트에 액세스하는 경우, 액세스 제어 기능을 사용하여 Content Manager 시스템 정보에 대한 액세스를 제어할 수 있습니다. 다양한 액세스 제어 API를 사용하여 전체 시스템, 밀접하게 관련된 시스템의 연산 세트 또는 개별 항목에 대한 액세스를 제어할 수 있습니다.

액세스 제어 API를 사용하여 완료할 수 있는 일부 작업에는 다음이 포함됩니다.

- 사용 권한 작성
- 시스템의 정보와 조치 목록 연관
- 라이브러리 서버의 정보에 대한 조치를 수행하도록 사용자에게 사용 권한 부여

사용 권한 작성

사용 권한은 클래스 DKPrivilegeICM으로 나타냅니다. 다음 단계 및 코드 예제에서는 새 사용 권한 오브젝트를 작성하고 이를 지속적인 오브젝트로 만들고 사용 권한을 검색하는 방법을 보여줍니다. 사용 권한을 작성하려면 아래의 단계를 수행하십시오.

Java

1. 데이터스토어에 연결하십시오.

```
DKDatastoreICM ds = new DKDatastoreICM();  
ds.connect("ICMNLSDb","icmadmin",password,"");  
dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();  
DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
```

2. DKAuthorizationMgmtICM 오브젝트를 검색하십시오. 이 클래스는 권한 작업을 처리합니다.

```
DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)  
dsAdmin.authorizationMgmt();
```

3. 새 사용 권한 오브젝트를 작성하십시오.

```
DKPrivilegeICM priv = new DKPrivilegeICM(ds);
```

4. 사용 권한 오브젝트에 이름을 할당하십시오.

```
priv.setName("UserPriv");
```

5. 사용 권한 오브젝트에 설명을 할당하십시오.

```
priv.setDescription("This is user-defined privilege");
```

6. 권한 관리자 오브젝트에 새 사용 권한을 추가하십시오.

```
aclMgmt.add(priv);
```

7. 권한 관리 오브젝트에서 새로 작성된 사용 권한을 검색하십시오.

```
dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
```

8. 사용 권한 오브젝트에 대한 정보를 표시하십시오.

```
System.out.println("privilege name = " + aPriv.getName());  
System.out.println("privilege description = " + aPriv.getDescription());
```

C++

1. 관련된 모든 관리 오브젝트 및 데이터스토어 오브젝트를 작성하십시오.

```
//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. 새 사용 권한 오브젝트를 작성하고 등록 정보를 설정하십시오.

```
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. 권한 관리 오브젝트를 통해 데이터스토어에 사용 권한을 추가하십시오.

```
aclMgmt->add(priv);
```

사용 권한 세트 작성

사용 권한은 클래스 `DKPrivilegeSetICM`으로 나타냅니다. 사용 권한 세트 작성을 시작하려면 먼저 컨텐츠 서버에 연결되어 있어야 합니다. 사용 권한 세트를 작성하려면 다음 단계를 완료하십시오.

Java

1. 새 사용 권한 세트에 추가할 새 사용 권한(또는 검색 사용 권한)을 작성하십시오.

```
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. 새 사용 권한 세트를 작성하십시오.

```
DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
```

3. 사용 권한 세트에 이름을 할당하십시오.

```
privSet1.setName("UserPrivSet");
```

4. 사용 권한 세트에 설명을 할당하십시오.

```
privSet1.setDescription("This is a user-defined priv set");
```

5. 사용 권한 세트에 사용 권한을 추가하십시오.

```
privSet1.addPrivilege(priv_1);
privSet1.addPrivilege(priv_2);
privSet1.addPrivilege(priv_3);
```

6. 권한 관리자에 새로 작성한 사용 권한 세트를 추가하십시오.

```
AcIMgmt.add(privSet1);
```

7. 새로 작성한 사용 권한 세트에 대한 정보를 표시하십시오.

```
DKPrivilegeSetICM aPrivSet = (DKPrivilegeSetICM)
acIMgmt.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. 관련된 모든 관리 오브젝트 및 데이터스토어 오브젝트를 작성하십시오.

```
//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. 세 개의 사용 권한을 작성하고 등록 정보를 설정하십시오.

```
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. 새 사용 권한 세트를 작성하고 등록 정보를 설정하십시오.

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("This is a user-defined priv set");
```

4. 작성된 사용 권한을 새 사용 권한 세트에 추가하십시오.

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. 권한 관리 오브젝트를 사용하여 데이터스토어에 사용 권한 세트를 추가하십시오.

```
aclMgmt->add(privSet1);
```

사용 권한 세트 등록 정보 표시

아래 예제에서는 사용 권한 세트의 등록 정보를 표시하는 방법을 보여줍니다.

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);
```

액세스 제어 목록(ACL)

Content Manager 액세스 제어 모델은 제어된 엔티티 레벨에 적용됩니다. 제어된 엔티티는 보호된 사용자 데이터 단위입니다. 제어된 엔티티는 개별 항목, 항목 유형 또는 전체 라이브러리일 수 있습니다. 제어된 엔티티의 조작은 하나 이상의 제어 규칙에 의해 조정됩니다. ACL은 이러한 제어 규칙에 대한 컨테이너입니다. DKAccessControlListICM 클래스는 ACL을 나타냅니다. Content Manager 시스템의 제어된 모든 엔티티는 ACL에 바인드되어야 합니다.

아래 예제에서는 ACL 정의 방법을 보여줍니다.

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. 새 DKACLData 오브젝트를 정의하십시오. 각 DKACLData 오브젝트는 ACL 관련 데이터를 보유하는 데 사용됩니다.

```
DKACLData * aclData1 = new DKACLData();  
// 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. 다른 DKACLdata 오브젝트를 작성하십시오.

```
DKACLData * aclData2 = new DKACLData();  
aclData2->setUserGroupName("ICMPUBLIC");  
aclData2->setPatronType(DK_CM_USER_KIND_GROUP);  
DKPrivilegeSetICM * privSet_2 = (DKPrivilegeSetICM *)  
    aclMgmt->retrievePrivilegeSet("PublicPrivSet");  
aclData2->setPrivilegeSet(privSet_2);
```

3. 새 ACL을 작성하십시오. 이 새 ACL에 대한 등록 정보 값을 설정하십시오.

```
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. 3단계에서 작성된 ACL인 데이터 오브젝트를 ACL에 추가하십시오.

```
acl1->addACLData(aclData1);  
acl1->addACLData(aclData2);
```

5. 3단계에서 작성된 ACL을 권한 관리자에 추가하십시오.

```
aclMgmt->add(acl1);
```

완전한 샘플 프로그램은 Samples 디렉토리에 있습니다.

ACL 정보 검색 및 표시

ACL 정보를 검색 및 표시하려면 다음 단계를 완료하십시오.

1. ACL 이름을 사용하여 권한 관리 오브젝트에서 ACL을 검색하십시오.

C++

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
    retrieveAccessControlList("UserACL");
cout<<"ACL name = "<< (char *)acl_1->getName() << endl;
cout<<"    desc = "<< (char *)acl_1->getDescription() << endl;
```

2. ACL과 연관된 ACL 데이터를 검색하십시오.

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);
```

항목 유형에 ACL 할당

일단 ACL이 작성되면 특정 항목 유형에 이를 연관시킬 수 있습니다. 다음은 ACL을 항목 유형에 할당하기 위한 단계입니다.

1. 새 항목 유형을 설정하십시오.

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. ACL과 연관된 ACL 데이터를 검색하십시오.

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.
```

완전한 샘플 프로그램은 Samples 디렉토리에 있습니다.

항목에 ACL 할당

항목 레벨 액세스 제어를 사용하려면 ACL을 항목에 바인드해야 합니다. 이를 위해서는 아래 코드 단편과 같이 DDO의 add() 메소드를 사용하여 항목을 작성해야 합니다. 아래 코드 단편에서는 이미 콘텐츠 서버에 연결되어 있으며 ACL을 작성한 것으로 간주됩니다. 완전한 샘플 프로그램은 Samples 디렉토리에 있습니다.

Java

```
//Create a new DDO
DKDDO ddoItem =dsICM.createDDO("myItemType",DKConstantICM.DK_CM_ITEM);
//create a new ACL (access contrl list)
DKAccessControlListICM ac11 =new DKAccessControlListICM(ds);
//Assign a name to the ACL
ac11.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. 기존 항목 유형을 기반으로 새 항목(DDO)을 작성하십시오.
`DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);`
2. 새 ACL(액세스 제어 목록)을 작성하고 ACL 등록 정보를 설정하십시오.
`DKAccessControlListICM * ac11 = new DKAccessControlListICM(dsICM);`
3. ACL에 이름을 할당하십시오.
`ac11->setName("MyACL");`
4. 새 등록 정보를 DDO에 추가하십시오. 이 등록 정보를 `DK_ICM_PROPERTY_ACL`이라고 합니다.
`ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);`
5. 위에서 작성된 ACL을 이 등록 정보에 대한 값으로 설정하십시오. 사용자는 기존 ACL 검색을 선택하여 이 항목에 대한 ACL로 사용할 수 있음에 유의하십시오.
`ddoItem->setProperty(propId, "MyACL");`
6. 데이터스토어에 DDO를 유지하십시오.
`ddoItem->add();`

조회 언어 이해

이제 Content Manager 8.2에서 강력한 XML 기반 조회 언어를 사용할 수 있습니다. 응용프로그램이 Content Manager 시스템에 저장된 항목을 검색하면 기본 엔진은 조회를 기준으로 검색 처리를 수행합니다. 효율적으로 Content Manager의 계층 구조 데이터 모델을 검토하려면 Content Manager 조회 언어를 사용해야 합니다. 조회 언어에는 다음과 같은 장점이 제공됩니다.

- 완전한 데이터 모델 지원
- 버전 지원(예: 특정 버전 검색 및 최신 버전 검색)
- 링크된 항목 및 참조를 통해 구성요소 유형 보기 계층 구조 내에서 검색 기능
- 매개변수식 검색 및 텍스트 검색의 결합
- SORTBY 기능 제공
- Content Manager 액세스 제어 시행
- W3C XML 조회 작업 드래프트의 서브세트인 XQPE(XQuery Path Expressions) 준수
- 고성능 검색 실행

Content Manager 조회 언어는 소유 언어와는 달리, W3C XML 조회 작업 드래프트의 서브세트인 XQPE(XQuery Path Expressions)를 준수하는 XML 기반 조회 언어입니다. 조회 언어는 계층 구조 항목 유형을 검색하여 항목을 빠르고 쉽게 찾습니다. 조회 작업을 시작하려면 조회 언어 개념, 구문 및 문법을 이해해야 합니다.

모든 조회는 구성요소 유형 보기에서 완료됩니다. 따라서 조회 문자열에서 루트 구성요소 또는 하위 구성요소에 사용되는 이름은 시스템에서 작성한 기본 구성요소 유형 보기 이름(예: Journal, Journal_Article) 또는 사용자 정의 구성요소 유형 보기 이름(예: My_Journal, My_Book_Section)이 될 수 있습니다. sys admin에서 구성요소 유형 보기를 구성요소 유형 서브세트로 호출합니다.

문서, 폴더, 오브젝트 등에 대한 조회를 제출하면 요청은 조회를 처리하여 해당 SQL로 변환하는 Content Manager 조회 엔진으로 전달됩니다. 라이브러리는 보안 점검에서 추가하고(ACL) 조회를 실행합니다.

Content Manager 서버 조회

Content Manager 라이브러리 서버를 조회하려면 다음을 수행하십시오.

1. 검색 조건을 나타내도록 조회 문자열을 작성하여 해당 콘텐츠 서버를 조회하십시오.
2. 조회 문자열 및 조회 옵션과 함께 evaluate, execute 또는 executeWithCallback 메소드를 호출하십시오.
3. DKResults 오브젝트, dkResultSetCursor 또는 dkCallback 오브젝트를 통해 결과를 검색하십시오.

조회 API는 조회 준비 및 조회 실행, 조회 실행 상태 모니터링 및 결과 검색과 같은 조회 처리 작업을 수행합니다.

논리적으로 조회 문자열은 세 가지 유형의 조회(매개변수식, 텍스트, 결합) 중 하나를 포함할 수 있습니다. 매개변수식 조회는 동일 및 비교와 같은 조건을 사용합니다. 텍스트 조회는 텍스트 검색 기능을 사용하여 검색을 보다 강력하게 만듭니다. 결합된 조회는 텍스트 조회와 매개변수식 조건으로 구성됩니다.

조회를 실행하려면 메소드 `evaluate`, `execute` 또는 `executeWithCallback` 중 하나를 사용하십시오. `evaluate` 메소드는 DDO의 콜렉션으로서 모든 결과를 리턴하므로 작은 결과 세트에 유용합니다. `evaluate` 메소드는 다음과 같은 특성을 갖는 `dkResultSetCursor` 오브젝트를 리턴합니다.

- `dkResultSetCursor`는 콘텐츠 서버 커서처럼 작동합니다.
- 사용자가 요청할 때 리턴되는 DDO가 블록으로 검색되므로 대형 결과 세트에 사용할 수 있습니다.
- `dkResultSetCursor`를 닫기 및 열기 메소드를 호출하여 조회를 재실행하는 데 사용할 수 있습니다.
- `dkResultSetCursor`를 결과 세트 커서의 현재 위치를 삭제 및 갱신하는 데 사용할 수 있습니다.

`executeWithCallback` 메소드는 비동기 방식으로 조회를 실행하고 결과를 블록에 지정된 콜백 오브젝트로 전송합니다. `executeWithCallback` 메소드는 조회 결과 검색 작업에서 실행의 기본 스레드를 제거합니다. 조회 메소드에 대한 자세한 정보는 `SSearchICM` 샘플을 참조하십시오. `DKDatastoreICM` 클래스의 `evaluate`, `execute` 및 `executeWithCallback` 메소드에 대해서는 응용프로그램 프로그래밍 참조서를 참조하십시오.

Content Manager 데이터 모델에 조회 언어 적용

조회 언어의 이해를 돕기 위해 개념적으로 라이브러리 서버를 XML 문서로 볼 수 있습니다. XML 문서 유추는 조회 언어를 설명하는 데만 사용됩니다. 따라서 라이브러리 서버의 XML 표현은 가상 표현일 뿐이며 라이브러리 서버의 항목은 XML 요소가 아니므로 조회 수행 시 XML 요소를 얻을 수 없습니다. 라이브러리 서버의 XML 표현에서 Content Manager 데이터 모델 요소는 다음과 같이 나타납니다.

항목 일반적으로 각 CM 항목은 중첩된 XML 요소로 나타납니다. 여기서 최상위 레벨 XML 요소는 루트 구성요소를 나타내고 중첩된 XML 요소는 내림차순으로 구성요소를 나타냅니다. 따라서 XML 요소 중첩은 구성요소 계층 구조를 나타냅니다.

루트 구성요소

루트 구성요소는 XML 요소의 첫 번째 레벨로 나타납니다. 루트 구성요소는 ID

ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER SEMANTICTYPE 및 기타 구성요소의 사용자 정의 속성과 같은 XML 속성을 가지고 있습니다. 라이브러리 서버에서 ITEMID는 고유합니다.

하위 구성요소

하위 구성요소는 중첩된 XML 요소로 나타나며 STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID 및 기타 구성요소의 사용자 정의 속성과 같은 속성을 가지고 있습니다.

COMPONENTID 자체는 Content Manager 구성요소 내에서만 고유하다는 점에 유의하십시오. ITEMID 및 VERSIONID는 하위 항목의 루트 구성요소 ITEMID 및 VERSIONID와 정확히 동일합니다.

사용자 정의 속성

각 사용자 정의 속성은 포함 구성요소를 나타내는 XML 요소 내에 중첩된 XML 속성으로 나타납니다.

링크 인바운드 링크 및 아웃바운드 링크는 CM 데이터 모델에서 항목 자체의 일부가 아니지만(링크 테이블에 별도로 저장되지만), 조회를 위해 항목을 나타내는 XML 요소의 일부로 이를 개념적으로 생각하는 것이 편리합니다. 따라서 응용 프로그램이 조회에 명시적으로 결합을 작성하지 않아도 됩니다. 링크는 항목 간의 일대다 관계를 나타냅니다. 이 관계는 단지 루트 구성요소 사이의 관계라는 점에 유의하십시오.(하위 구성요소에서 링크가 발생하거나 끝날 수 없습니다.)

항목에서 발생하는 링크는 IDREF LINKITEMREF, IDREF TARGETITEMREF 및 STRING LINKTYPE 속성을 가진

<OUTBOUNDLINK> XML 요소로 나타납니다. LINKITEMREF는 링크에 대한 메타데이터가 들어 있는 항목에 대한 참조입니다. TARGETITEMREF는 링크에서 가리키는 항목에 대한 참조입니다. LINKTYPE은 링크 유형입니다. 마찬가지로 항목을 가리키는 링크는 IDREF LINKITEMREF, IDREF SOURCEITEMREF 및 STRING LINKTYPE 속성을 가진 <INBOUNDLINK> XML 요소로 나타납니다. SOURCEITEMREF는 링크가 발생한 항목에 대한 참조입니다. 링크 의미에 대한 자세한 정보는 SLinksICM 및 SSearchICM 샘플을 참조하십시오.

참조(참조 속성)

참조 속성은 IDREF 유형의 XML 속성으로 나타납니다. 참조는 항목이나 구성요소 및 다른 항목 간의 일대일 관계를 나타냅니다. 따라서 참조의 목표는 루트 구성요소만 가능하며 하위 구성요소는 될 수 없습니다. 그러나 참조 속성은 루트 또는 하위 구성요소에서 발생할 수 있습니다.

참조 속성은 시스템에서 정의(SYSREFERENCEATTRS)하거나 사용자가 정의(아래의 샘플 조회에서 PublicationRef)합니다. 참조는 두 방향으로 모두 탐색될 수 있습니다.

두 방향 참조는 위에서 설명한 대로 두 방향 링크와 비슷한 방법으로 수행됩니다. 참조되고 있는 항목을, 이 항목을 참조하는 구성요소를 가리키는 (IDREF 유형의) REFERENCER XML 속성이 들어 있는 REFERENCEDBY라는 XML 요소를 가지고 있는 것으로 개념적으로 생각할 수 있습니다. 이것은 두 방향 링크에 대한 SOURCEITEMREF 속성을 가진 INBOUNDLINK 요소와 비슷합니다.

참조는 또한 삭제 의미를 지원합니다(제한, 계단식, 널(null) 설정). 참조 속성에 대한 자세한 정보는 SReferenceAttrDefCreationICM 및 SSearchICM 샘플을 참조하십시오.

문서 DKFolder를 통해 Content Manager에 제공된 폴더 기능은 "DKFolder" (DK_ICM_LINKTYPENAME_DKFOLDER) 링크 유형의 세트 링크로서 저장됩니다. 각 폴더-컨텐츠 관계는 폴더에서 아웃바운드 링크로서, 그리고 각 링크의 폴더를 원본으로, 컨텐츠를 목표로 하는 컨텐츠에 인바운드 링크로서 모델링합니다. 폴더를 검색 및 탐색하려면 링크를 검색 및 탐색하는 의미를 따르십시오. 자세한 정보는 SFolderICM 및 SSearchICM 샘플을 참조하십시오.

매개변수식 검색 이해

항목은 종종 선택된 속성에 대한 검색을 시작하여 검색됩니다. 단일 조회는 컨텐츠 서버의 항목에 대한 시스템 정의 및 사용자 정의 속성을 둘다 검사할 수 있습니다. 단순 검색 조건은 절로 결합되는 값, 연산자 및 속성 이름으로 구성되어 있습니다. Content Manager에서는 매개변수식 검색을 완료하기 위한 여러 비교 연산자를 제공합니다. 연산자는 다음과 같습니다.

```
"="
"< "
"<="
">"
">="
"!="
"LIKE"
"NOT LIKE"
"BETWEEN"
"NOT BETWEEN"
"IS NULL"
"IS NOT NULL"
```

논리 연산자 AND, OR 및 NOT을 사용하여 단순 검색 조건을 절로 결합하여 복합 검색 조건을 지정할 수 있습니다. 자세한 정보는 조회 예제를 참조하십시오.

텍스트 검색 이해

Content Manager는 이전에 CM 8.1의 Text Information Extender(TIE)로 알려진 DB2 Universal Database Net Search Extender(NSE)를 사용하여 구성요소에 텍스트를 포함하는 속성의 텍스트 검색 및 오브젝트의 텍스트 검색 등의 두 가지 텍스트 검색 유형을 제공합니다. 두 가지 텍스트 검색 유형 간의 주요 차이점은 콘텐츠 저장 방식입니다. 속성을 텍스트 검색 가능하도록 정의하면 해당 속성 열에 포함된 텍스트를 검색할 수 있게 됩니다. 속성(열)을 텍스트 검색 가능하게 하도록 NSE가 텍스트 색인을 작성합니다. 텍스트 색인은 검색될 텍스트에 대한 정보를 보유합니다. 이 정보는 효율적으로 텍스트 검색을 수행하는 데 사용됩니다. 예를 들어, 시스템 관리자인 Fred는 하위 구성요소 유형 보기 Journal_Article을 가진 Journal이라는 항목 유형을 작성하여 텍스트 검색을 가능하게 하려고 합니다. Journal_Article에 대한 속성 중 하나인 Title을 Fred가 텍스트 검색 기능으로 설정했습니다. 보험업자인 Lily가 "Java"라는 단어가 포함된 Title을 검색하면 시스템은 "Java"가 나오는 Title 텍스트 색인을 검색합니다.

CM 8.1에서 사용된 TIE에 대한 정보는 DB2 Universal Database Text Information Extender(TIE) 문서를 참조하십시오. CM 8.2에서 사용된 NSE에 대한 정보는 DB2 Universal Database Net Search Extender(NSE) 문서를 참조하십시오. ICM 조회 언어의 텍스트 검색에 대한 논의에서 NSE 및 TIE는 교환이 가능하도록 사용할 수 있습니다. ICM 조회 언어의 Perspective에서 NSE 및 TIE 간의 텍스트 검색 구문규칙 또는 기능에서는 차이가 없습니다.

오브젝트 콘텐츠 검색

오브젝트 콘텐츠의 검색은 조금 다르게 작동합니다. 시스템은 열을 직접적으로 색인화하는 것 대신에 자원 관리자에서 오브젝트 위치에 대한 참조를 사용합니다. NSE는 참조를 사용하여 텍스트 색인 작성 시 콘텐츠를 폐치합니다. 검색을 수행하는 일반 사용자는 자원 관리자에 저장된 오브젝트를 검색할 때와 별다른 차이점을 느끼지 못합니다. 그러나 시스템 관리자가 자원 관리자의 콘텐츠를 찾는 검색 메커니즘을 위해서는 텍스트 자원 항목 유형 보기를 설정해야 합니다. 텍스트 검색은 자원 항목 유형의 속성 "TIEREF"에 대해 수행됩니다. 이는 텍스트 검색을 위해 자원 관리자에 저장된 콘텐츠를 나타냅니다.

문서 검색

문서 부분의 콘텐츠에 대해 텍스트 검색을 수행할 수 있습니다. 가상 구성요소 유형 보기 "ICMPARTS"는 시스템에 있는 모든 문서의 하위로 조회에서 지원됩니다. "ICMPARTS" 구성요소 유형 보기 아래의 "TIEREF" 속성은 텍스트 검색을 위해 해당 문서의 모든 검색 가능 부분의 콘텐츠를 나타냅니다. 이러한 기능에 대한 특정 사용법은 조회 예제를 참조하십시오.

사용자 정의 속성 텍스트 검색 가능하게 만들기

DKAttrDefICM 및 DKItemTypeDefICM API를 사용하여 사용자 정의 속성 텍스트 검색을 가능하게 만들 수 있습니다. DKTextIndexDefICM 클래스를 사용하여 작성된 텍스트 색인의 기본 등록 정보를 수정할 수 있습니다. API에 대한 자세한 정보는 온라인 API 참조서 또는 SItemTypeCreationICM 샘플을 참조하십시오.

텍스트 검색 구문규칙 이해

기본 텍스트 검색 구문규칙 또는 고급 텍스트 검색 구문규칙을 사용하여 텍스트 검색 조회를 수행할 수 있습니다.

기본 텍스트 검색

대부분의 텍스트 검색은 단지 몇 단어를 하나씩 나열하여 수행되기 때문에, 기본(단순화된) 텍스트 검색 구문규칙은 사용자가 이 작업을 특히 쉽게 수행할 수 있도록 설계되었습니다. 구문규칙에는 따옴표 구문뿐 아니라 "+" 및 "-"를 사용할 수 있습니다.

"contains-text-basic" 및 "score-basic" 함수를 사용하여 단순화된 텍스트 검색이 완료된 "contains-text-basic" 함수를 사용하여 속성 내에서 또는 자원이나 문서 콘텐츠 내에서 검색합니다. "score-basic" 함수는 동일한 구문규칙을 "contains-text-basic" 함수로 사용하며 텍스트 검색 결과의 등급에 기초하여 결과를 정렬하는 데 사용됩니다. 참인지 여부를 점검하려면 "contains-text-basic" 함수를 1로 설정하고, 거짓인지 여부를 점검하려면 0으로 설정하십시오. 이러한 함수가 사용되는 방식에 대해서는 조회 예제를 참조하십시오.

기본 텍스트 검색 구문규칙에 대한 추가 정보에는 다음이 포함됩니다.

- 대소문자 구분 텍스트 검색을 수행합니다(기본적으로 고급 구문규칙과 마찬가지로). 대소문자 구분 검색 옵션에 대해서는 NSE 문서를 참조하십시오.
- 따옴표 내의 용어는 구문으로 간주됩니다.
- + (플러스) - (마이너스)의 사용
 - + (플러스) = 문서는 이 단어를 포함해야 합니다.
 - - (마이너스) = 문서는 이 단어를 포함해서는 안 됩니다.
 - + 또는 -가 지정되지 않은 경우, 조회 엔진은 알고리즘을 사용하여 텍스트에 단어를 일치시킵니다.
- 논리 연산자(AND, OR, NOT)는 유효하지 않으며 무시됩니다.
- 기본 구문규칙의 괄호는 지원되지 않습니다.
- 올바른 와일드카드
 - ? (물음표) = 단일 문자를 나타냅니다.
 - * (별표) = 개수가 정해지지 않은 임의의 문자를 나타냅니다.

기본 텍스트 검색에 대한 자세한 정보는 SSearchICM 샘플을 검색하십시오.

고급 텍스트 검색

고급 텍스트 검색 구문규칙을 사용하면 보다 복잡한 텍스트 검색 조건을 지정할 수 있습니다. 텍스트 검색은 NSE 텍스트 검색 구문규칙을 사용하며, 근접 검색 및 유사 검색과 같은 강력한 기능을 지원합니다. 고급 텍스트 검색 구문규칙은 "contains-text" 및 "score" 함수를, "contains-text-basic" 및 "score-basic" 함수가 기본 텍스트 검색에 사용되는 것과 유사한 방식으로 사용합니다. 고급 함수에 제공된 문자열은 큰따옴표를 작은따옴표로 변경하거나 작은따옴표를 큰따옴표로 변경하는 경우를 제외하고 NSE 구문규칙으로 되어 있어야 합니다. 예를 들어, NSE의 CONTAINS(description, "IBM")=1 조건은 CM 조회 언어의 contains-text(@description, "IBM")=1 이 됩니다. ESC 문자 사용을 최소화한 간단한 조회 작성을 지원하려면 이 작업을 수행해야 합니다. 참인지 여부를 점검하려면 "contains-text-basic" 함수를 1로 설정하고, 거짓인지 여부를 점검하려면 0으로 설정하십시오. 고급 텍스트 검색에 대한 자세한 정보는 조회 예제를 참조하십시오.

고급 텍스트 검색에 대한 자세한 정보는 SSearchICM 샘플을 참조하십시오.

결합된 매개변수식 및 텍스트 검색 작성

검색은 항목이나 구성요소의 모든 부분, 항목이나 구성요소의 텍스트 또는 자원 콘텐츠의 텍스트를 기반으로 가상적으로 수행될 수 있습니다. 다음 유형의 하나를 검색할 수 있습니다.

매개변수식 검색

항목 및 구성요소 등록 정보, 속성, 참조, 링크 또는 폴더 콘텐츠 등을 기반으로 검색합니다.

텍스트 검색

텍스트 내에서 검색합니다.

결합된 검색

매개변수식과 텍스트 검색을 모두 사용하여 검색합니다.

결합된 매개변수식 및 텍스트 검색을 작성하려면 아래 단계를 수행하십시오.

Java

1. ICM 데이터스토어를 작성한 다음 연결하십시오.

2. 결합된 조회 문자열을 생성하십시오.

```
String queryString =  
    "//Journal_Article [Journal_Author/@LastName = \"Richardt\" +  
    \" AND contains-text (@Text, \" 'Java' & 'XML' \")=1]\";
```

3. 기본값과 다른 검색 옵션을 사용하려면 DKNVPair 배열을 사용하여 옵션을 패키징하십시오.

```
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. 세 가지 다른 방식, 즉 evaluate, execute, executeWithCallback 중 하나에
서 결합된 조회를 실행하십시오.

```
DKResults resultsCollection =  
    (DKResults)dsICM.evaluate(queryString,  
    DKConstant.DK_CM_XQPE_QL_TYPE, parms);
```

5. 결과를 처리하십시오. 결과 처리 프로시저는 어떤 실행 메소드를 사용했는지에
따라 달라집니다.

완전한 샘플 및 추가 문서에 대해서는 CMBROOT\Samples\java\icm의 SSearchICM API
교육 샘플을 참조하십시오.

C++

1. 검색 옵션을 지정하십시오. 옵션 배열에 끝 요소가 있어야 함에 유의하십시오. 예를 들어, 두 개의 옵션을 지정하려면 옵션 배열에는 세 개의 요소가 있어야 합니다.

```
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. 검색을 실행하십시오. 검색을 실행하는 방법에는 세 가지가 있습니다.

evaluate

모든 결과를 컬렉션으로 리턴하며, 작은 세트에 적합합니다.

execute

결과 세트 커서를 리턴하며, 호출자는 결과를 반복하는 데 사용합니다.

executeWithCallback

결과 세트를 반복하고 각 결과 블록에 대한 콜백 오브젝트를 호출하는 스레드를 작성합니다. 호출자는 콜백 오브젝트를 사용하여 결과를 가져옵니다.

아래 예제에서 결과 중 다섯 개의 항목만 필요하므로 DKDatastoreICM.evaluate 메소드가 사용됩니다.

```
DKResults * resultsCollection = (DKResults *) (dkCollection *)
dsICM->evaluate(queryString, DK_CM_XQPE_QL_TYPE, parms);
```

3. 검색의 결과를 표시하십시오.

```
// 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;
}

delete(iter);
delete[] parms;
....
```

4. 제거하십시오.

완전한 샘플 및 추가 문서에 대해서는 CMBROOT\Samples\java\icm의 SSearchICM API 교육 샘플을 참조하십시오.

조회 예제

이 절에서는 조회 언어를 보다 잘 이해하고 조회 작성을 시작할 수 있도록 다음 정보를 제공합니다.

- 샘플 데이터 모델
- 데이터 모델의 XML 문서 표현
- 샘플 조회
- 조회 언어 문법

다음 절의 샘플 조회는 221 페이지의 그림 13에 나타난 조회 예제 데이터 모델에 기준합니다. 샘플 조회를 검토할 때 데이터 모델 그림을 참조하십시오.

대체 데이터 모델에 기준한 추가 조회 구문규칙 및 예제에 대해서는 SSearchICM 샘플을 참조하십시오.

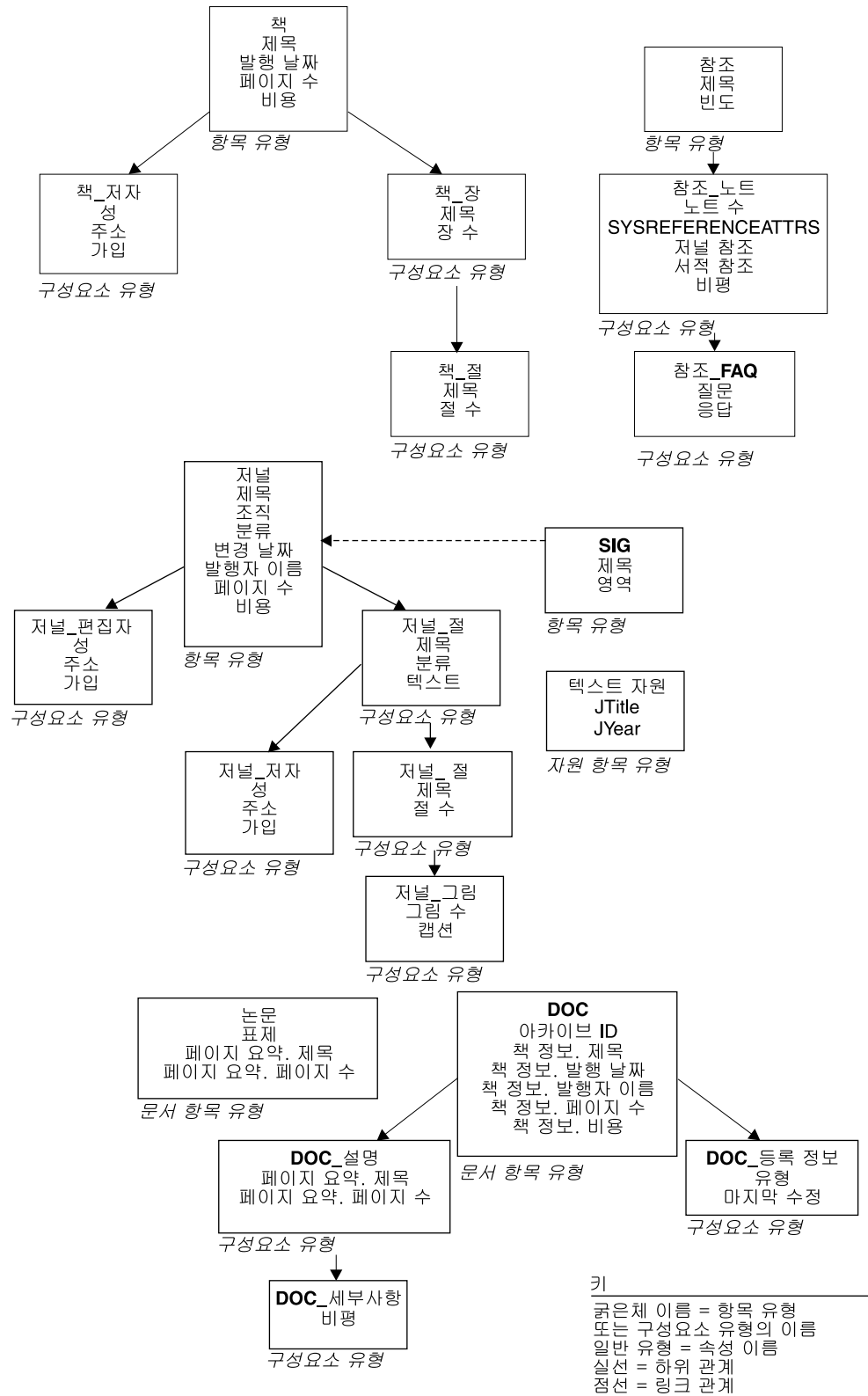


그림 13. 조회 예제 데이터 모델

아래의 XML 문서는 그림 13의 데이터 모델을 나타낸 것입니다.

조회 예제 데이터 모델의 XML 표현:

```
<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>)

```

조회 예제

이 절에 제공된 샘플 조회는 샘플 데이터 모델, 221 페이지의 그림 13 및 222 페이지의 샘플 XML 문서를 기본으로 합니다. 다음은 조회 예제 이해를 도와주는 몇 가지 힌트입니다.

- 디렉토리 구조를 따르듯이 조회 문자열을 따르십시오.
- "/" 단일 슬래시는 직접 하위 관계를 나타냅니다.
- "/" 이중 슬래시는 하위 관계 또는 자손 관계를 나타냅니다.
- "." (점)은 계층 구조의 현재 구성요소를 나타냅니다.
- ".." (점-점)은 현재 구성요소의 상위 구성요소를 나타냅니다.
- "@"(AT 기호)는 속성을 나타냅니다.
- "[]"(사각 대괄호)는 조건문 또는 목록을 나타냅니다.
- "=>"(DEREFERENCE 연산자)는 조치 링크 또는 참조를 나타냅니다.
- 조회 결과는 구성요소여야 합니다.(예를 들어, 속성은 경로의 마지막에 올 수 없습니다.)

추가 예제 및 문서는 SSearchICM 샘플에 제공됩니다.

예 1: 구성요소에 액세스

이 조회는 모든 저널을 찾습니다.

```
/Journal
```

설명: 『/』는 XML 문서의 암시적 루트에서부터 시작하며, 이 경우에는 전체 라이브러리 서버입니다. 각 항목 유형은 루트 아래의 요소입니다. LS.xml이 위에 설명된 대로 전체 모델을 포함하는 XML 문서이면 명시적 문서 루트는 문서(LS.xml)입니다.

예제 2: 속성에 액세스

이 조회는 총 페이지가 50인 모든 저널 기사를 찾습니다.

```
/Journal[@NumPages=50]
```

설명: 술어 @NumPages = 50은 『NumPages』 Content Manager 속성이 50으로 설정된 모든 저널에 대해 참(true)이라고 평가합니다.

예제 3: 다중 항목 유형

이 조회는 저자 중 한 명이 『Williams』이고 섹션 제목이 『XML』로 시작하는 모든 책이나 저널을 찾습니다.

```
(/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%"])
```

설명: 위의 두 조회는 동일한 결과를 생성합니다. 『./Journal_Author』는 구성 요소 Journal_Author가 경로의 현재 구성요소 바로 아래(첫 번째 경우는 Book 또는 Journal) 또는 계층 구조에서 더 깊은 위치에 있어야 함을 의미합니다. LIKE 연산자는 와일드 카드 문자(이 경우 『%』)와 함께 사용된다는 점에 유의하십시오.

예제 4: 조건의 산술 연산

이 조회는 페이지 수가 45에서 200 사이인 모든 저널을 찾습니다.

```
/Journal[@NumPages BETWEEN 49-4 AND 2*100]
```

설명: 산술 연산을 수행하여 BETWEEN 연산자에 사용될 결과값을 계산할 수 있습니다.

예제 5: 정방향 링크 탐색

이 조회는 『Williams』가 편집했으며 제목이 『SIGMOD』인 SIG에 들어 있는 모든 저널 기사를 찾습니다.

```
/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK  
[ @LINKTYPE = "contains" ] / @TARGETITEMREF =>  
Journal[Journal_Editor/@LastName = "Williams"]  
/Journal_Article
```

설명: 다음은 정방향의 다음 링크에 대한 예제입니다. 가상 XML 구성요소 OUTBOUNDLINK 및 해당 속성 TARGETITEMREF는 모든 저널을 탐색한 다음 마지막으로 기본 Journal_Articles를 탐색하는 데 사용됩니다. 경로의 마지막 구성요소는 조회 결과로 리턴된 것입니다. 특정 링크 유형만(이 예에서 『contains』) 특정 항목 유형(이 예에서 Journal)으로 탐색하여 결과를 제한할 수 있습니다. 라이브러리 서버의 개념적 XML 표현은 인바운드 및 아웃바운드 링크를 항목의 일부로 보기 때문에, 참조 해제 연산자를 사용하면 아웃바운드 응용프로그램이 명시적 결합을 작성하지 않도록 할 수 있습니다.

예제 6: 역방향 링크 탐색

이 조회는 저자 『Nelson』의 기사가 있으며 값이 5달러 미만인 저널이 있는 모든 유형의 항목을 찾습니다.

```
/Journal[ @Cost < 5  
AND ../Journal_Author/@LastName = "Nelson"]  
/INBOUNDLINK[ @LINKTYPE = "contains"]  
/ @SOURCEITEMREF => *
```

설명: 다음은 역방향의 다음 링크에 대한 예제입니다. dereference 연산자 『=>』 다음의 와일드 카드 『*』는 어떤 유형의 항목이 결과로 리턴되었는지 확인합니다.

예제 7: 텍스트 검색(contains-text 및 Score 함수)

이 조회는 저자가 『Richardt』이며 텍스트 『Java』 및 텍스트 『XML』이 포함된 저널 기사를 찾습니다. 결과는 텍스트 검색 스코어별로 순서화됩니다.

```
//Journal_Article[Journal_Author/@LastName = "Richardt"  
AND contains-text(@Text, " 'Java' & 'XML' ")=1]  
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

설명: 다음은 contains-text 함수를 사용하여 텍스트 검색을 수행하는 예제입니다. 이 함수가 지원하는 구문규칙에 대해서는 DB2 Net Search Extender(NSE) 문서를 참조하십시오. contains-text 함수는 참인 경우 1이고 거짓인 경우 0입니다. score 함수는 NSE가 리턴한 등급 정보를 사용합니다. 이 정보는 이 경우 SORTBY를 통해 결과 저널 기사를 정렬하는 데 사용됩니다.

예제 8 텍스트 검색(contains-text 및 속성 정렬)

이 조회는 제목에 단어 『Design』 또는 단어 『Index』가 들어 있는 모든 저널을 찾아서 결과를 제목별 내림차순으로 정렬합니다.

```

/Journal
[Journal_Article[contains-text(@Title, " 'Design' | 'Index' ")=1]]
SORTBY (@Title DESCENDING)

```

설명: 다음은 contains-text 함수를 사용하여 텍스트 검색을 수행하는 또다른 예제입니다. 이 경우 정렬은 『Title』 속성에 DESCENDING 연산자를 사용합니다. SORTBY에 대한 기본값은 ASCENDING입니다.

예제 9: 텍스트 검색(contains-text-basic 및 score-basic 함수)

이 조회는 단순화된(기본) 텍스트 검색 구문규칙을 사용하여 텍스트 『Java』 및 텍스트 『JDK 1.3』이 들어 있지만 텍스트 『XML』이 들어 있지 않은 모든 저널 기사를 찾아서 결과를 텍스트 검색 스코어별로 정렬합니다.

```

//Journal_Article
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' "))

```

설명: 다음은 단순화된 텍스트 검색 구문규칙을 사용하여 텍스트 검색을 수행하는 예제입니다. 『Title』 속성에 있어야 하는 단어나 구문을 나타내려면 『+』를 사용하고, 마찬가지로 다른 단어나 구문을 제외시키려면 『-』를 사용합니다. score-basic 함수는 이전 예제의 Score 함수와 유사하게 작동하지만 단순화된 구문규칙을 사용합니다.

예제 10: 자원 항목에서 텍스트 검색

이 조회는 텍스트 『Java』 및 텍스트 『XML』이 들어 있는 텍스트 자원 항목 유형 『TextResource』의 텍스트 자원을 찾습니다.

```

/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'
")=1]

```

설명: 다음은 자원 관리자의 자원 내에서 텍스트 검색을 수행하는 예제입니다. 『TIEREF』 속성은 『TextResource』 유형의 항목으로 나타내는 자원 표현으로 사용됩니다. 이 경우 TIE 구문은 평상시와 같이 contains-text 함수 내에서 사용됩니다. 이 함수가 지원하는 구문규칙에 대해서는 DB2 Net Search Extender(NSE) 문서를 참조하십시오.

예제 11: 정방향 참조 탐색

이 조회는 EIP 제목의 책을 언급하는 회의록에서 모든 자주 묻는 질문을 찾습니다.

```

/Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%EIP%"]]
/Conference_FAQ

```

예제 12: 정방향 참조 탐색

이 조회는 인터넷 관련 회의록에 언급된 책의 모든 장을 찾습니다.

```

/Conference[@Title LIKE "%Internet%"]
/Conference_Note/@PublicationRef =>
*/Book_Chapter

```

예제 13: 역방향 참조 탐색

이 조회는 책을 가리키는 언급이 있는 모든 구성요소를 찾습니다.

```
/Book/REFERENCEDBY/@REFERENCER => *
```

예제 14: 역방향 참조 탐색

이 조회는 XML에 대한 책을 언급하는 회의록에서 모든 자주 묻는 질문을 찾습니다.

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>  
Conference_Note/Conference_FAQ
```

설명: 참조 속성은 Conference_Note 구성요소 내에서 발생하므로 dereference 연산자 다음의 첫 번째 구성요소로 나타나야 하는 구성요소입니다. 이 조회는 가령 Conference 다음에 『 =>』 연산자가 올 경우 빈 결과 세트를 생성합니다.

예제 15: 역방향 참조 탐색

이 조회는 의견란에 XML이 들어 있고 책을 가리키는 언급이 있는 모든 구성요소를 찾습니다.

```
/Book/REFERENCEDBY/@REFERENCER =>  
*[@Remark LIKE "%XML%"]
```

예제 16: 최신 버전 함수

이 조회는 최신 버전의 모든 저널을 찾습니다. 기본적으로 조회에 일치하는 표시된 구성요소 유형 보기의 모든 버전이 리턴됩니다. VERSIONID는 모든 구성요소 유형에 포함된 시스템 정의 속성입니다.

```
/Journal[@VERSIONID = latest-version (.)]
```

예제 17: 참조 해제 목표의 최신 버전 함수

이 조회는 회의록에 언급된 모든 최신 버전 책을 찾습니다.

```
/Conference/Conference_Note/@SYSREFERENCEATTRS =>  
Book[@VERSIONID = latest-version(.)]
```

예제 18: 와일드 카드 구성요소의 최신 버전 함수

이 조회는 책을 가리키는 언급이 있는 모든 최신 버전 구성요소를 찾습니다.

```
/Book/REFERENCEDBY/@REFERENCER => *  
[@VERSIONID = latest-version(.)]
```

예제 19: 시스템 정의 속성

이 조회는 특정 항목 ID를 가진 모든 루트 구성요소를 찾습니다.

```
/*[@ITEMID =  
"A1001001A01J09B00241C95000"]
```

예제 20: 문서 모델의 텍스트 검색

이 조회는 해당 부분에 단어 『XML』을 포함하는 모든 문서를 찾습니다.

```
/Doc[contains-text(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

설명: 조회 언어는 문서 분류의 특정 항목 유형에 포함되어 있는 모든 ICM 부분 항목 유형에 액세스할 수 있도록 하는 『ICMPARTS』 가상 구성요소를 제공합니다.

예제 21: 문서 모델(ICM 부분에 액세스)

이 조회는 기억영역 ID가 555인 문서의 모든 부분을 찾습니다.

```
/Doc[@ArchiveID = 555]/ICMPARTS/  
@SYSREFERENCEATTRS => *
```

예제 22: 문서 모델(ICM 부분에 액세스)

이 조회는 시스템에 있는 모든 문서의 모든 부분을 찾습니다.

```
//ICMPARTS/@SYSREFERENCEATTRS => *
```

설명: Doc 및 Paper 항목 유형 모두 시스템의 문서로 정의되어 있으므로 둘 모두의 ICM 부분이 결과로 리턴됩니다.

예제 23: 속성의 존재

이 조회는 제목을 가진 모든 루트 구성요소를 찾습니다.

```
/*[@Title]
```

설명: 루트 구성요소만이 리턴되도록 하는 제한을 제거하려면, 조회가 이중 슬래시로 시작하도록 재작성하면 됩니다.

```
//*[@Title]
```

예제 24: 리터럴 및 표현식 목록

이 조회는 제목이 기사 제목, 섹션 제목 또는 "IBM Systems Journal"인 모든 저널을 찾습니다.

```
/Journal[@Title = [Journal_Article/@Title,  
./Journal_Section/@Title,"IBM Systems Journal"]]
```

예제 25: 숫자 리터럴 목록

이 조회는 가격이 \$10, \$20 또는 \$30인 모든 책을 찾습니다.

```
/Book[@Cost = [10, 20, 30]]
```

예제 26: 조회 결과 목록

이 조회는 제목이 『Star Wars』인 모든 저널이나 모든 책을 찾습니다.

```
[/Journal, /Book[@Title = "Star Wars"]]
```

예제 27: 속성 그룹

이 조회는 설명의 길이가 최소한 20 페이지인 문서에 대한 모든 세부사항을 찾습니다.

```
/Doc[Doc_Description/@PageSummary.NumPages >=  
20]//Doc_Details
```


설명: 속성(예: 『NumPages』)이 속성 그룹(예: 『PageSummary』)에 포함되어 있는 경우, 해당 속성을 GroupName.AttrName(예: PageSummary.NumPages)으로 나타내야 합니다. 속성 『@NumPages』는 Doc_Description 아래 없습니다.

INTERSECT/EXCEPT로 구한 중간 결과는 산술(단항/2진) 또는 비교 연산자와 결합될 수 없습니다. set 연산자(UNION/INTERSECT/EXCEPT)로 이를 결합할 수도 있고 자체적으로 표시할 수도 있습니다.

UNION/INTERSECT/EXCEPT의 올바른 사용법 예는 다음과 같습니다.

1. (/Journal/Journal_Article[@Title = "Content Management"]
EXCEPT
//Journal_Article[@Classification =
"Security"])/Journal_Section

EXCEPT의 결과는 전체 조회의 결과이므로 이 조회는 올바릅니다. 이 조회는 연산자 사용과 결합되어 있지 않습니다.

2. /Journal[(Journal_Editor/@LastName
UNION .//Journal_Author/@LastName) = "Davis"]

UNION 연산자에는 제한이 없으므로 이 조회는 올바릅니다.

3. /Journal[Journal_Article[Journal_Section/@Title INTERSECT
./Journal_Figure/@Caption]/@Title = "Content Management"]

INTERSECT의 결과는 연산자 사용과 결합되어 있지 않으므로 이 조회는 올바릅니다.

4. /Journal[@Title = "VLDB"]
UNION /Journal[@Cost = 20]
INTERSECT /Journal[@Organization = "ACM"]

INTERSECT 연산자의 결과는 set 연산자(UNION) 사용과 결합되어 있으므로 이 조회는 올바릅니다.

INTERSECT/EXCEPT의 올바른 사용법 예는 다음과 같습니다.

1. /Journal[(Journal_Editor/@LastName
INTERSECT .//Journal_Author/@LastName) = "Davis"]

INTERSECT 연산자의 결과는 비교 연산자(=) 사용과 결합되어 있으므로 이 조회는 올바르지 않습니다.

2. /Journal[(./Journal_Section/@SectionNum
EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]

EXCEPT의 결과는 산술 연산자(+) 사용과 결합되어 있으므로 이 조회는 올바르지 않습니다.

조회 언어 이해

조회 언어의 고급 기능(예: 와일드 카드 "%" 또는 "_" 텍스트 문자열 내부의)을 지원하기 위해, 와일드 카드가 정규 문자로 간주될 때와 와일드 카드 문자의 특수 의미를 가지고 있을 때를 구별하는 데 이스케이프 시퀀스가 사용됩니다. 와일드 카드 문자를 정규 문자로 간주할 때에는 Esc 문자가 앞에 와야 하기 때문에 와일드 카드로 사용된 문자를 알아 두어야 합니다. 또한 이스케이프 시퀀스는 작은따옴표와 큰따옴표를 처리하는 데도 사용됩니다.

문자열에서 특수 문자(큰따옴표, 작은따옴표), 와일드 카드 문자(퍼센트 부호, 밑줄, 별표, 물음표) 또는 기본 Esc 문자(백슬래시)를 포함하는 조회를 사용할 때 이스케이프 시퀀스를 추가해야 합니다. 이는 비교 조건에 사용되는 문자열을 가장 간편하게 처리하는 방법이며, LIKE 연산자 및 텍스트 검색 기능과 관련되어 있습니다. 특수 문자를 제대로 처리해야 조회를 실행하고 정확한 조회 결과를 얻을 수 있습니다.

중요사항: 조회에 와일드 카드 문자를 너무 적게 사용하면 결과 목록 크기가 상당히 커집니다. 크기가 커지면 성능이 저하되고 예상하지 못한 검색 결과를 리턴하기도 합니다.

비교 연산자와 이스케이프 시퀀스 사용("=", "!", ">", "<", "BETWEEN" 및 기타)

큰따옴표 표시 "

큰따옴표 앞에 다른 큰따옴표를 넣습니다.

예제:

```
//Journal_Article[@Title = "Analysis of ""The Time Machine""  
by H. G. Wells himself"]
```

기사 제목에 큰따옴표로 "The Time Machine"이라는 책 제목이 포함되어 있으므로 내부 큰따옴표는 이스케이프되어야 합니다.

작은따옴표 표시(어포스트로피) '

이 경우에 이스케이프할 필요가 없습니다.

예제:

```
/Book[@Title != "Uncle Tom's Cabin"]
```

LIKE 연산자와 순서 이탈 사용

큰따옴표 표시 "

큰따옴표 앞에 다른 큰따옴표를 넣습니다.

예제:

```
//Journal_Article[@Title LIKE "Analysis of ""The Time Machine""  
%"]
```

기사 제목에 큰따옴표로 "The Time Machine"이라는 책 제목이 포함되어 있으므로 내부 큰따옴표는 이스케이프되어야 합니다.

작은따옴표 표시 (어포스트로피) '

이 경우에 이스케이프할 필요가 없습니다.

예제:

```
/Book[@Title LIKE "Uncle Tom's Cabin"]
```

와일드카드("%", "_")

퍼센트 부호("%")는 LIKE 연산자에 사용되는 문자열에서 임의의 문자 수를 나타내는 데 사용하는 와일드 카드 문자입니다. 밑줄("_")은 임의의 문자 하나를 나타내는 데 사용되는 와일드 카드 문자입니다. 이러한 와일드 카드 문자가 정규 문자로 취급되게 하려면 다음을 수행해야 합니다.

1. Esc 문자 앞에 와일드 카드 문자를 둡니다.
2. LIKE 구문 다음의 Esc 문자에 ESCAPE 절을 추가합니다.

예제 A:

```
/Book[@Title LIKE "Plato%s%S_mposium"]
```

이 예제에서는 제목의 철자가 확실하지 않을 때 와일드 카드 "%" 및 "_"을 사용해서 책을 찾는 방법을 보여줍니다.

예제 B:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ in query"  
ESCAPE "!"]
```

이 예제에서는 검색 문자열에 정규 문자(와일드 카드 아님)인 밑줄("_")이 포함되어 있으므로 느낌표("!")를 사용하여 밑줄을 이스케이프할 수 있습니다. 모든 단일 문자는 Esc 문자로 사용될 수 있습니다.

예제 C:

```
//Journal_Article[@Title LIKE "_sage of underscore \_ in%" ESCAPE  
"\"]
```

이 조회에서 문자열의 여러 끝부분을 나타내고 "Usage" 단어를 대문자와 소문자로 모두 표시하기 위해 와일드 카드 문자는 정규 문자("\")로 이스케이프된 "_" 및 와일드 카드("_")로 사용되었습니다.

예제 D:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ on Yahoo!!"  
ESCAPE "!"]
```

정규 문자로 Esc 문자를 사용할 수도 있습니다. 이렇게 하려면 아래 예제의 "Yahoo!" 검색과 같이 Esc 문자를 앞에 넣으십시오.

고급 텍스트 검색과 순서 이탈 사용("Include" 및 "Score" 함수)

큰따옴표 표시 "

큰따옴표 앞에 다른 큰따옴표를 넣습니다.

예제:

```
//Journal_Article[contains-text (@Title, " 'Analysis of ""The Time Machine"" %' ")=1]
```

기사 제목에 큰따옴표로 "The Time Machine"이라는 책 제목이 포함되어 있으므로 내부 큰따옴표는 이스케이프되어야 합니다.

작은따옴표 표시 (어포스트로피) '

작은따옴표 앞에 다른 작은따옴표를 넣습니다. 작은따옴표 세트가 용어나 구문을 묶는 데 사용되므로 하나의 작은따옴표는 고급 텍스트 검색에서는 허용되지 않습니다. 작은따옴표가 단어 내부에 있으면, 작은따옴표는 용어나 구문 끝에 오는 작은따옴표와 구별하기 위해 이스케이프되어야 합니다.

예제 A:

```
/Book[contains-text (@Title, " 'Uncle Tom''s Cabin' ")=1]  
SORTBY (score (@Title, " 'Uncle Tom''s Cabin' "))
```

Tom''s에는 작은따옴표가 두 개 있습니다.

예제 B:

```
/Book[contains-text (@Title, " ('Greek' & 'Plato''s Symposium') & NOT ' Socrates' ")=1]SORTBY (score (@Title, " ('Greek' & 'Plato''s Symposium') & NOT ' Socrates' "))
```

Plato''s에는 작은따옴표가 두 개 있습니다.

와일드카드 ("% ", "_")

LIKE 연산자와 마찬가지로 고급 구문규칙은 "%" 및 "_"을 와일드 카드로 사용합니다. 퍼센트 부호("%")는 임의의 문자 수를 나타내는 데 사용하는 와일드 카드 문자입니다. 밑줄("_")은 임의의 문자 하나를 나타내는 데 사용되는 와일드 카드 문자입니다. 이러한 와일드 카드 문자가 정규 문자로 취급되게 하려면 다음을 수행해야 합니다.

1. 와일드 카드 문자 앞에 Esc 문자를 둡니다.
2. Esc 문자를 사용하는 각 용어 다음에 ESCAPE 절을 추가합니다.

예제 A:

```
/Book[contains-text (@Title, " 'Usage of underscore !_  
in query' ESCAPE '!' ")=1] SORTBY (score (@Title, " 'Usage  
of underscore !_ in query' ESCAPE '!' "))
```

이 예제에서 느낌표("!")는 밑줄 앞에서 Esc 문자로 사용됩니다.

예제 B:

```
/Book[contains-text (@Title, " 'Usage of underscore !_  
in query' ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage  
of underscore !_ on Yahoo!!' ESCAPE '!' |  
'War and Peace' ")=1]
```

모든 용어에 동일한 Esc 문자가 포함되도 와일드 카드를 이스케이프한 텍스트
검색 문자열의 모든 용어 다음에 ESCAPE 절을 추가해야 함에 유의하십시오.

기본 텍스트 검색과 이스케이프 시퀀스 사용("contains-text-basic" 및 "score-basic" 함수)

큰따옴표 표시 "

큰따옴표 앞에 다른 큰따옴표를 넣습니다.

예제:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of '""The  
Time Machine""' ")=1]
```

기사 제목에 큰따옴표로 "The Time Machine"이라는 책 제목이 포함되어 있
으므로 내부 큰따옴표는 이스케이프되어야 합니다. 책 제목은 구문을 유지하기
위해 작은따옴표로 묶여 있습니다.

작은따옴표 표시(어포스트로피) '

작은따옴표 앞에 다른 작은따옴표를 넣습니다. 기본 텍스트 검색 구문규칙은 용
어를 작은따옴표로 묶어 단어 사이에 공백이 들어갈 수 있도록 합니다. 작은따
옴표를 두 개 표시하는 것은 새 용어가 작은따옴표로 시작하는 경우 용어 내
에서 일어날 수 있는 작은따옴표 구분 문제를 해결하는 데 필요합니다.

예제 A:

```
/Book[contains-text-basic (@Title, "Uncle Tom''s Cabin")=1]SORTBY  
(score-basic (@Title, "Uncle Tom''s Cabin"))
```

Tom''s에는 작은따옴표가 두 개 있습니다.

예제 B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato''s Symposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek +'Plato''s
Symposium' -Socrates "))
```

Plato's에는 작은따옴표가 두 개 있고 'Plato's Symposium'은 구문이기 때문에 작은따옴표로 묶여 있습니다.

와일드카드("*", "?" 및 "\")

"*", "?" 및 "\" 문자"가 와일드 카드로 사용되지 않는 경우 이들 문자 앞에 백슬래시("\")가 옵니다. 별표("*")는 contains-text-basic 및 score-basic 함수에 대한 기본 텍스트 검색에서 임의의 문자 수를 나타내는 와일드 카드 문자로 사용됩니다. 물음표("?")는 임의의 문자 하나를 나타내는 데 사용되는 와일드 카드 문자입니다. 기본 텍스트 검색에서 조회 언어는 검색할 용어에 와일드 카드 문자가 포함되어 있고 해당 와일드 카드 문자를 정규 문자로 취급하고 싶을 때 사용할 Esc 문자 백슬래시("\")를 제공합니다.

예제 A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek
+'Plato*s*S?mposium' -Socrates "))
```

이 예제에서는 용어의 철자를 정확히 알지 못할 때 기본 텍스트 검색을 사용하는 방법을 보여줍니다. "*" 및 "?" 문자는 이 경우 와일드 카드를 의미하고 이스케이프되지 않습니다.

예제 B:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1] SORTBY
(score-basic (@Title, "Why forgive\?"))
```

이 예제에서는 제목에 물음표("?")가 정규 문자로 포함되어 있어 백슬래시로 이스케이프될 수 있습니다.

예제 C:

```
//Journal_Section[contains-text-basic (@Title,
"C:\OurWork\IsNeverDone")=1] SORTBY (score-basic (@Title,
"C:\OurWork\IsNeverDone"))
```

검색 용어 "C:\OurWork\IsNeverDone"에서 자연스럽게 발생하는 각 백슬래시는 다른 백슬래시로 이스케이프되어야 합니다.

Java 및 C++에서 순서 이탈 사용

특수 문자(예: 큰따옴표 및 백슬래시) 앞에 백슬래시를 넣습니다.

예제:

조회:

/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]");
```

물음표 앞에 오는 내부 큰따옴표와 백슬래시 앞에 백슬래시가 오는 방법을 주의해서 살펴 보십시오. 이렇게 처리하는 것은 Java 및 C++ 프로그래밍 언어에서 상속된 것입니다. 자세한 정보는 해당 언어 스펙을 참조하십시오.

조회 언어 문법

조회 언어 공식 문법은 다음과 같습니다.

- (* keywords *)
- 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) ;
- (* literals *)
- 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_CHARACTER는 모든 유니코드 문자 및 이스케이프 시퀀스 세트입니다. 그 정의는 이 문서에 포함되어 있지 않습니다. *) (* 문자열 리터럴은 큰따옴표로 분리되며 큰따옴표를 제외한 모든 문자를 포함할 수 있습니다. 큰따옴표를 문자열 리터럴의 일부로 포함시키려면 연속하는 두 개의 큰따옴표를 지정하십시오. 즉, 다른 큰따옴표가 큰따옴표를 이스케이프 처리합니다. 이들은 하나의 큰따옴표 문자로 취급됩니다. *)
- STRING_LITERAL = "'", { (UNICODE_CHARACTER - "'") | ("'", "'') } , "'", ;
- (* 이스케이프 시퀀스는 큰따옴표로 분리되는 단일 문자입니다. 큰따옴표 자체를 Esc 문자로 지정하려면 연속하는 두 개의 큰따옴표를 지정하십시오. 예를 들어, 큰따옴표를 다른 큰따옴표로 이탈합니다. 이들은 하나의 큰따옴표 문자로 취급됩니다. ESCAPE_CHARACTER의 공식적인 값에 대한 자세한 설명은 LIKE 술어에서 DB2 SQL 참조서 섹션을 참조하십시오. *)
- 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" | "_" | "\$") ;

- (* IDENTIFIER는 문자(a-z, A-Z) 또는 밑줄 또는 달러 기호로 시작하고, 0개 이상의 문자, 밑줄, 달러 문자 또는 숫자(0-9)가 뒤따릅니다. 키워드는 작은따옴표로 묶은 경우에만 IDENTIFIER가 될 수 있습니다. *)
- 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;

자원 관리자에 대한 작업

Content Manager 자원 관리자는 관리 자원(오브젝트)의 컬렉션을 제어합니다. 필요한 기억영역 및 HSM(Hierarchical Storage Management) 하부 구조도 관리하지만 먼저 HSM을 지원하기 위해 자원 관리자를 구성해야 합니다. 자원 관리자는 둘 이상의 오브젝트 유형에 대해 스트리밍, 압축, 압축 해제, 암호화, 인코딩, 트랜스코딩, 검색 또는 텍스트 마이닝과 같은 유형별 서비스를 지원하기 위한 기능을 가지고 있습니다.

단일 자원 관리자는 하나의 라이브러리 서버에서 배타적으로 사용됩니다. Content Manager 시스템에서 전달하는 각 자원 관리자는 제어 라이브러리 서버, 다른 Content Manager 구성요소 및 응용프로그램에서 로컬(동일한 네트워크 노드) 또는 원격으로 액세스할 수 있는 원래의 데이터 액세스 API의 공통 서브세트를 제공합니다.

다른 데이터 액세스 API는 자원 관리자의 자체 클라이언트 지원 또는 CIFS, NFS, FTP 등의 표준 네트워크 액세스 프로토콜을 사용하는 자원 관리자에 대한 원격 액세스가 허용됩니다. 원격 액세스의 경우 클라이언트-서버 연결을 사용합니다. 클라이언트는 표준 웹 서버 사용을 통해 HTTP를 사용하여 Content Manager 자원 관리자와 통신합니다. 데이터 전달은 HTTP, FTP 및 FS 데이터 전송 프로토콜을 기반으로 합니다. HTTP를 사용하여 Content Manager 관리 콘텐츠에 액세스해야 하는 Content Manager 구성요소 또는 응용프로그램은 라이브러리 서버 및 자원 관리자와 동적으로 삼각 관계를 형성할 수 있습니다. 이 삼각 관계는 응용프로그램과 각 자원 관리자 사이에 직접 데이터 액세스 경로를 형성하고 라이브러리 서버와 자원 관리자 사이에 제어 경로를 형성합니다. 이러한 개념적인 삼각 관계를 단일 노드 구성에서 지리적으로 분산된 구성에 이르기까지 모든 네트워크 구성에 맵핑할 수 있습니다.

새 아키텍처는 호스트 기반 서브시스템과 같이 응용프로그램에 직접 액세스할 수 없는 자원 관리자, 액세스 제어를 처리하지 않는 단일 사용자 시스템 또는 비즈니스 방침에 따라 응용프로그램의 직접 액세스가 허용되지 않는 매우 중요한 정보가 들어 있는 시스템도 수용합니다. 이 경우 자원 관리자에 대한 액세스는 간접적입니다. Content Manager 시스템은 동기 및 비동기 호출뿐만 아니라 데이터 전송의 풀 및 푸시 패러다임을 모두 수용합니다.

자원 관리자 구성 방법에 대한 정보는 samples 디렉토리 cmbroot/samples/java/icm의 SResourceMgrDefCreationICM 샘플 및 *Content Management 시스템 계획 및 설치*를 참조하십시오.

자원 관리자 오브젝트에 대한 작업

Content Manager 내의 모든 관리 엔티티를 항목이라고 합니다. 항목에는 두 가지 유형이 있습니다. 하나는 문서 또는 폴더와 같은 순수한 논리 엔티티를 나타내는 유형이고, 다른 하나는 워드 프로세싱 문서의 텍스트 데이터, 청구서의 스캔된 이미지 또는 교통 사고 비디오 클립과 같은 실제 데이터 오브젝트를 나타내는 엔티티입니다. 오브젝트는 논리 문서와 연관된 실제 데이터를 처리하는 데 필요한 특수 상태 및 작동을 가지고 있습니다.

또한 자원 오브젝트는 파일 시스템의 파일, 비디오 서버의 비디오 클립 및 BLOB를 나타내기도 합니다. 런타임 시 자원 오브젝트는 자신이 가리키는 실제 데이터에 액세스하는 데 사용됩니다. 이러한 이유로 Content Manager의 자원 오브젝트는 유형을 가지고 있습니다. 즉, 이러한 오브젝트는 특정 상태 및 작동을 가지고 있습니다. 라이브러리 서버 및 자원 관리자는 스키마를 공유하여 오브젝트 상태를 저장합니다. Content Manager에서 제공하는 기본 오브젝트 유형은 일반 BLOB 또는 CLOB, 텍스트, 이미지 및 비디오 콘텐츠 오브젝트입니다. 미리 정의된 유형의 서브클래스를 작성할 수도 있습니다. 자원 오브젝트는 검색에 사용되는 사용자 정의 속성을 가질 수도 있습니다.

Content Manager 시스템 Perspective에서 각 오브젝트는 고유한 논리 식별자인 URI(Uniform Resource Identifier)로 나타납니다. 라이브러리 서버는 URI 이름 공간을 관리합니다. 요청 시 라이브러리 서버는 URI를 URL(Uniform Resource Locators)에 맵핑합니다. URL은 실제 데이터에 대한 액세스 권한을 얻을 때 사용됩니다. URL은 자원 관리자에서 관리하는 기억영역을 직접 가리키지는 않습니다. 대신 자원 관리자는 로컬 이름 공간을 사용하여 논리 오브젝트 이름을 실제 파일 이름으로 변환합니다. 오브젝트 URI는 특정 자원 관리자에서 작성합니다. 라이브러리 서버 또는 일반 사용자는 오브젝트 URI(해당 이름)를 제시할 수 있지만 결정은 자원 관리자가 합니다.

Content Manager 자원 관리자 API를 사용하여 오브젝트에 액세스할 수 있습니다(저장, 검색, 갱신, 삭제 등). 어떤 경우에는 오브젝트(스트림, 멀티캐스트, 스테이지)나 파일 시스템에 고유한 기본 API를 사용할 수 있습니다.

자원 관리자 오브젝트에 대한 작업 방법 정보는 samples 디렉토리인 CMBROOT\Samples\java\icm 또는 CMBROOT\Samples\cpp\icm의 SResourceItemCreationICM 샘플을 참조하십시오.

Content Manager에서 문서 관리

Content Manager는 비즈니스 오브젝트 관리에 사용할 수 있는 융통성 있는 문서 관리 데이터 모델을 구현합니다. 데이터 모델의 기본 요소에는 폴더, 문서 및 오브젝트가 포함됩니다.

앞서 언급한 바와 같이 문서, 폴더 및 기타 오브젝트는 Content Manager 시스템의 항목으로 모두 나타납니다. API 레벨에서 문서와 폴더 간의 유일한 차이점은 의미 유형 및 각각의 DKFolder 기능입니다. 문서는 단일 값 속성(문서 이름, 날짜, 주제), 여러 값 속성(키워드) 및 여러 값 속성의 컬렉션(번지, 구/군/시, 시/도 우편번호로 구성된 주소)을 비롯하여 문서를 설명하는 속성 또는 메타데이터로 구성되어 있습니다.

문서 관리 데이터 모델은 문서 부분을 사용하여 오브젝트(자원 항목)와 문서를 연관시킵니다. 이 모델은 문서 하나를 구성하기 위해 둘 이상의 부분을 지원합니다. 예를 들어, 각 페이지는 별도의 부분이 될 수 있습니다. 응용프로그램이 문서를 구성하는 부분의 순서를 판별하기 위해 부분 번호가 문서 부분에 저장됩니다. 문서 부분에는 부분, 해당 자원 관리자에 있는 컬렉션 이름 등을 포함한 자원 관리자 ID, 크기, MIME 유형과 같은 부분에 대한 기타 정보가 포함된 오브젝트(참조 속성)에 대한 포인터가 들어 있습니다. 모든 오브젝트는 서로 다른 속성을 가질 수 있습니다. 예를 들어, 주식에는 XY 좌표가 있고, 메모 로그에는 메모 텍스트의 CCSID가 있을 수 있습니다.

문서 관리 데이터 모델을 더 잘 이해하려면 사용자가 클라이언트 응용프로그램을 사용하여 문서를 가져오는 다음 시나리오를 살펴 보십시오.

- 사용자에게 창이 표시됩니다.
- 시스템으로 가져올 파일 이름을 입력하거나 선택합니다(예: 경찰 보고서).
- 문서 유형(메모, 청구, 설계)을 선택합니다.
- 새로운 창이 열리며, 여기에 해당 문서를 설명하는 속성을 입력할 수 있습니다(예: 날짜, 청구 번호 및 보험 증권 번호).
- 문서 부분을 정의하고 일부 속성값을 입력합니다. 예를 들어, 경찰 보고서는 청구의 문서 부분입니다.
- 문서 설명 입력을 완료하고 작업을 완료합니다. 보험 증권 보고서가 작성됩니다.

클라이언트 응용프로그램은 API 또는 JavaBeans를 사용하여 라이브러리 서버 및 자원 관리자에 연결합니다. 시스템은 두 개의 항목(비자원 항목 및 자원 항목)을 작성하여 문서를 저장합니다. 보험 증권 보고서에는 자원 관리자에 저장된 사진이 포함되므로 두 개의 항목이 작성됩니다. 문서는 API에 대한 단일 호출로 작성됩니다. 오브젝트는 자원 관리자에 저장되며 자원 관리자는 오브젝트의 시간 소인 및 기타 메타데이터를 리턴함

니다. 자원 관리자는 오브젝트의 참조 속성을 작성하며 문서의 하위 구성요소에 참조 속성을 삽입합니다. 하위 구성요소를 저장하고 속성을 갱신하기 위해 라이브러리 서버에 대한 최종 호출이 이루어집니다. 전체 프로세스는 트랜잭션 단위로 바인드되어 API 장애가 발생할 경우, 문서가 부분적으로 작성되는 결과를 가져오지는 않습니다.

문서를 작성한 후 갱신할 수 있습니다. 메타데이터 변경 또는 콘텐츠 변경의 두 가지 유형의 갱신을 수행할 수 있습니다. 라이브러리 서버는 자동으로 다음 버전 번호(항목의 버전화가 사용될 경우)를 가진 새 항목 레코드를 작성하고 이 항목에 연관된 모든 하위 구성요소를 복사합니다.

문서 관리 데이터 모델 작성

이 절에서는 문서 관리 데이터 모델과 연관된 기본 작업을 완료하는 방법에 대해 설명합니다.

- 문서 항목 유형 작성.
- 문서 작성.
- 문서 갱신.
- 문서 검색 및 삭제.
- 문서 관리 데이터 모델의 부분 버전화.

문서 항목 유형 작성

Java:

1. ItemType 분류 = DK_ICM_ITEMTYPE_CLASS_DOC_MODEL과 함께 ItemType 문서를 작성하고 다음을 설정하십시오.

```
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. 부분을 문서에 추가하려면 ItemType 관계를 작성하십시오. 각 부분에 대해 EntityDef를 검색하십시오.

```
Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
```

3. 각 부분에 대해 ItemType 관계를 작성하고 값을 설정하십시오.

```
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. ItemType 관계를 문서(원본)에 추가하십시오.

```
docItemTypeDef.addItemTypeRelation(itRel);
```

5. 문서 ItemType을 지속 스토어에 추가하십시오.

```
docItemTypeDef.add();
```

C++:

1. 콘텐츠 서버 정의 오브젝트를 검색하십시오.

```
DKDatastoreDefICM * dsDefICM = (DKDatastoreDefICM *)dsICM->datastoreDef();
```

2. 콘텐츠 서버 정의 오브젝트를 검색하십시오.

```
DKDatastoreAdminICM * pdsAdmin = (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();
DKItemTypeDefICM *itemType = NULL;
DKItemTypeRelationDefICM *itemTypeRel = NULL;
DKAttrDefICM* attr = NULL;
```

3. 문서 항목 유형에 대한 속성을 검색하십시오. 속성이 없으면 다음을 작성합니다.

```
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. 방금 작성한 문서 및 part1 간 관계를 작성하십시오. 이렇게 하려면 먼저 각 부분에 대해 EntityDef를 검색하십시오.

```
DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)
    dsDefICM->retrieveEntity("ICMBASE");
//int part1ITypeId = itemTypePart1->getItemTypeId();
int part1ITypeId = itemTypePart1->getIntId();
```

5. 각 부분에 대해 항목 유형 관계를 작성하고 값을 설정하십시오.

```
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. 이 항목 유형에 포함되어 있는 항목 자원이 저장될 기본 컬렉션을 설정하십시오.

```
itemTypeRelPart1->setDefaultCollCode((short)1);
```

7. 이 항목 유형에 포함되어 있는 항목 자원이 저장될 기본 프리페치 컬렉션을 설정하십시오.

```

itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);
itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);
itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());

```

8. 항목 유형 관계를 문서(원본)에 추가하십시오.

```

itemType->addItemTypeRelation(itemTypeRelPart1);

```

9. 방금 작성한 문서 및 part2 간 관계를 작성하십시오. 이렇게 하려면 먼저 각 부분에 대해 EntityDef를 검색하십시오.

```

DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)
    dsDefICM->retrieveEntity("ICMANNOTATION");
int part2ITypeId = itemTypePart2->getIntId();

```

10. 각 부분에 대해 항목 유형 관계를 작성하고 값을 설정하십시오.

```

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. 항목 유형 관계를 문서(원본)에 추가하십시오.

```

itemType->addItemTypeRelation(itemTypeRelPart2);

```

12. 라이브러리 서버에서 항목 유형의 정의를 갱신하십시오.

```

itemType->add();

```

자세한 정보는 SItemTypeCreationICM 샘플을 참조하십시오.

문서 작성

"문서" 의미 유형의 항목은 내부에 속성(다른 의미 유형 항목과 비슷함) 및 여러 "부분" (다른 의미 유형 항목과 비슷하지 않음)을 포함할 수 있습니다. 아래 단계는 하나의 속성 및 하나의 "부분"을 포함하는 항목(미리 정의된 문서 항목 유형을 기반으로 함)을 작성하는 과정을 안내합니다. 아래 단계에서는 "S_varchar"라는 하나의 속성 및 하나의 부분("ICMBASE")이 포함된 "s_simple" 항목 유형이 이미 정의된 것으로 간주합니다.

Java

1. 문서 DDO를 작성하십시오.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
DKConstant.DK_CM_DOCUMENT);
short dataId = 0;
String attrValue = "Test";
```

2. 문서의 속성을 설정하십시오. 이 경우 항목 유형이 하나의 속성만 가지는 것으로 간주합니다.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
DKParts parts = null;
// Document's parts
```

3. 문서의 부분 컬렉션에 액세스하십시오.

```
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. 미리 정의된 유형 "ICMBASE"의 부분을 작성하십시오. 이 부분은 작성된 문서에 추가됩니다. 아래에서 작성된 문서는 하나의 부분만 가진 항목 유형을 기반으로 한다고 간주합니다.

```
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. 작성한 부분을 "부분" 컬렉션에 추가하십시오. 이는 지연된 저장임에 유의하십시오.(변경사항은 DDO 문서가 유지될 때까지 데이터스토어에 확약되지 않습니다.)

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. 문서를 데이터스토어에 유지하십시오.

```
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();
```

자세한 정보는 SDocModelItemICM 샘플을 참조하십시오.

문서 갱신

아래 단계는 "문서" 의미 유형 항목을 갱신하는 과정을 안내합니다. 아래 단계에서 새 부분이 추가되고 속성값이 갱신됩니다.

Java

1. 문서 항목에 대한 속성값을 갱신하십시오.

```
String attrValue = "New Value";
short dataId=ddoDocument.dataId
    (DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
```

2. 문서의 부분 컬렉션에 액세스하십시오.

```
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. 새 부분에 대한 데이터를 작성하십시오.

```
String partValue = "This is an annotation";
```

4. 미리 정의된 유형 "ICMANNOTATION"의 부분을 작성하십시오. 이 부분은 작성된 문서에 추가됩니다. 여기에서 작성된 문서는 하나의 부분만 가진 항목 유형을 기반으로 한다고 간주합니다. 일단 새 항목이 추가되면 문서는 두 부분을 가집니다.

```
DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setContent(partValue.getBytes());
pLobPart.setPartNumber(2);
```

5. 작성한 부분을 "부분" 컬렉션에 추가하십시오. 이는 지연된 저장임에 유의.(변경사항은 DDO 문서가 유지될 때까지 데이터스토어에 확약되지 않습니다.)

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. 변경된 문서를 데이터스토어에 유지하십시오.

```
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;
```

자세한 정보는 SDocModelItemICM 샘플을 참조하십시오.

문서 검색 및 삭제

문서를 검색하려면 ddo.retrieve(옵션)를 호출하십시오. 옵션을

DK_ICM_CONTENT_YES로 설정할 경우, 부분뿐만 아니라 부분 TOC 목록도 검색됩니다. 그렇지 않으면 부분의 TOC 목록만 검색됩니다.

문서를 삭제하려면 ddo.del()을 호출하십시오. 문서 및 첨부된 부분이 삭제됩니다. 부분이 있는 문서의 검색 및 삭제에 대한 자세한 정보는 SDocModelItemICM API 샘플을 참조하십시오.

문서 관리 데이터 모델의 부분 버전화

문서 부분의 버전화 등록 정보는 문서의 항목 유형에 선택된 각 부분 유형에 대해 항목 유형 관계의 버전화 등록 정보로 판별됩니다. 문서 부분의 버전화 특성은 다음을 포함합니다.

- 정규 문서와 마찬가지로 부분은 세 가지의 버전화 모델(항상 버전화, 버전화하지 않음(기본값), 응용프로그램 제어 버전화) 중 하나일 수 있습니다.
- 항목 유형의 버전 방침이 버전화하지 않음(versioned-never)인 경우, 해당 부분의 버전 방침도 버전화하지 않음(versioned-never)입니다.
- 항목 유형 T의 버전 방침이 항상 버전화(versioned-always)이며 항목 유형 T인 항목 I가 있으며 해당 속성이나 부분 컬렉션을 수정한 경우(부분을 추가/삭제 또는 갱신하여), 항목 I의 새 버전이 작성됩니다.
- 문서 자체와는 달리, 문서 부분에는 최대 버전 수가 들어 있지 않습니다.
- 관심 있는 부분(기본, 메모, 주석 등)의 항목 유형 관계 오브젝트로부터 부분 레벨 버전화 규칙을 구할 수 있습니다.

아래 예제에서는 항목 유형의 기본 부분에 대한 버전화 규칙을 가져오는 방법을 보여줍니다.

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();
```

트랜잭션에 대한 작업

Content Manager는 트랜잭션을 사용하여 라이브러리 서버 및 인접한 모든 자원 관리자 사이의 일관성을 유지보수할 수 있습니다. 트랜잭션은 사용자가 결정한 복구 가능한 작업 단위로, 라이브러리 서버에 대한 단일 연결을 통해 이루어진 연속된 API 호출로 구성되어 있습니다. 연속된 DKDatastoreICM 메소드 호출은 DDO 및 XDO를 통해 직간접적으로 이루어집니다.

트랜잭션 범위 및 트랜잭션 내의 작업량은 기본적으로 단일 API 메소드(암시적 트랜잭션)로 수행되는 작업입니다. 이 트랜잭션 유형은 필수이며 최적의 트랜잭션 범위를 수행합니다. 그러나 여러 메소드 호출(명시적 트랜잭션)을 포함하도록 작업 단위의 범위를 크게 변경할 수 있지만 이러한 유형의 트랜잭션을 사용하면 일부 성능에 대해 오버헤드가 발생할 수 있습니다.

하나의 트랜잭션을 종료하면 전체 트랜잭션이 확약 또는 롤백됩니다. 트랜잭션이 확약되는 경우, 트랜잭션 내에서 API 호출에 의한 모든 Content Manager 서버 변경사항은 영구적입니다. 트랜잭션이 롤백되거나 실패하는 경우, 트랜잭션 내에서 이루어진 모든 변경사항은 롤백 처리 중 취소됩니다.

암시적 트랜잭션의 경우, 트랜잭션의 확약과 롤백이 자동으로 완료됩니다. 명시적 트랜잭션이 사용되는 경우, 트랜잭션 확약은 응용프로그램에서 제어하는 반면, 트랜잭션 롤백은 응용프로그램 또는 Content Manager 시스템에서 자동으로 시작할 수 있습니다. Content Manager 시스템은 심각한 오류가 발생하거나 라이브러리 서버와 데이터베이스 사이의 교착 상태를 해결해야 할 경우에 롤백을 시작합니다.

트랜잭션 내에서 확약되지 않은 자원 관리자 변경사항은 트랜잭션이 확약될 때까지 변경사항을 수행한 응용프로그램에 표시되지 않습니다. 예를 들어, 자원 관리자 항목을 변경하고 저장합니다. 트랜잭션이 확약되기 전에 항목을 검색하면 그 항목은 수행한 변경사항을 반영하지 않습니다. 트랜잭션이 확약될 때까지 갱신된 항목을 볼 수 없습니다.

단일 라이브러리 서버 연결을 통한 동시 트랜잭션이나 겹치는 트랜잭션은 지원되지 않습니다. 동시 트랜잭션을 유지보수하려면 라이브러리 서버와 데이터베이스 사이의 복수 연결을 설정하거나, 클라이언트 응용프로그램에 대해 작업 중인 경우에는 복수의 클라이언트 프로세스 또는 스레드를 시작해야 합니다. IBM WebSphere® Application Server와 같은 응용프로그램은 처리, 연결 및 세션을 처리합니다.

DKDatastoreICM의 execute() 및 executeWithCallback() 메소드는 호출되면 자동으로 데이터베이스에 대한 추가 연결을 작성합니다. 새 데이터베이스 연결은 조회를 실행하는 데 사용됩니다. 조회는 별도의 데이터베이스 연결을 사용하므로 다른 콘텐츠 서버 조작과는 다른 별도의 트랜잭션 범위도 가지고 있습니다. DKResultSetCursor가 종료될 때 데이터베이스에 대한 연결도 종료됩니다.(또는 풀이 사용 가능하면 풀로 리턴됩니다.)

응용프로그램에서 트랜잭션을 설계할 때 고려해야 할 사항

트랜잭션이 확약되기 전에 클라이언트 노드 또는 라이브러리 서버가 실패할 경우, 데이터베이스 복구 기능이 라이브러리 서버에서 트랜잭션을 롤백합니다. 클라이언트 노드 및 자원 관리자가 모두 활성화된 경우, 실패하는 동안 자원 관리자에 발생한 변경사항이 즉시 취소됩니다. 클라이언트 노드 자체가 실패하면 자원 관리자와 라이브러리 서버 간의 일관성을 복원하기 위해 비동기 복구 유틸리티 주기를 통해 자원 관리자를 사용해야 합니다. 유틸리티를 실행하기 전에 서버는 계속 데이터 무결성을 유지합니다. 영향을 받는 것은 진행 중인 실패한 항목에 대한 조작이며 RM이 복구될 때까지 거부됩니다. 오브젝트 갱신 진행 중에 실패하면 첫 번째 실패가 조정될 때까지 동일한 오브젝트의 다른 갱신을 금지합니다.

자원 관리자가 실패하면 불일치를 제거하기 위해 비동기 복구 유틸리티를 실행해야 합니다. OS/390에서 자원 관리자는 OAM(오브젝트 액세스 방법) 등 보다 용이하게 복원하는 데 사용되는 원래의 트랜잭션 성능을 가지고 있습니다.

명시적 트랜잭션을 사용할 때의 주의사항

DKDatastoreICM.startTransaction() 및 DKDatastoreICM.commit()를 사용하여 트랜잭션 범위를 제어하는 명시적 트랜잭션인 경우, Content Manager 문서에 대해 작업할 응용프로그램을 개발할 때, 또는 DKLobICM 작성, 검색, 갱신 및 삭제(CRUD) 조작을 수행할 때 주의하십시오. 이러한 조작을 수행할 때 트랜잭션 끝에서 가장 가까운 위치에서 CRUD 조작을 수행해야 합니다. 또한 긴 트랜잭션은 잠재적으로 데이터베이스 잠금 문제를 유발할 수 있기 때문에 트랜잭션을 가능한 한 짧게 유지해야 합니다.

잠금 문제는 항목을 갱신하거나 응용프로그램이 트랜잭션을 즉시 확약하지 않도록 선택한 경우 가장 확실하게 나타납니다. 트랜잭션이 확약되지 않는 동안 갱신된 항목은 다른 응용프로그램에서 여전히 보입니다. 다른 사용자가 항목을 보거나 액세스하려고 시도하면 해당 사용자는 갱신 트랜잭션이 확약될 때까지 잠깁니다. 폴더에서 새 항목을 작성할 때 동일한 문제(데이터베이스 잠금)가 발생합니다. 다른 사용자에게 폴더가 보이고 사용자가 새 항목을 검색하려고 시도하면 해당 사용자는 트랜잭션에 확약될 때까지 잠깁니다. 트랜잭션이 확약되기 전에 걸린 시간이 사용자가 잠기는 시간입니다.

데이터베이스 잠금을 방지하는 가장 좋은 접근 방법은 트랜잭션을 자주 확약해서 오래 실행되지 않게 하는 것입니다. 하나의 트랜잭션에서 CRUD 조작을 수행해야 하는 경우, 갱신 중인 항목에 어떤 사용자도 액세스하지 않을 것으로 예상될 때 해당 조작을 수행하는 것이 좋습니다.

트랜잭션에서 체크인 및 체크아웃 사용

Content Manager는 항목에 대한 체크인 및 체크아웃 조작을 지원합니다. 체크아웃 조작을 호출하여 항목에 대한 지속 쓰기 잠금을 획득합니다. 사용자가 항목을 체크아웃할 때 다른 사용자는 해당 항목을 검색하거나 볼 수는 있어도 갱신할 수는 없습니다. 사용하고 있는 트랜잭션 모드(암시적 및 명시적)에 관계없이 항목을 갱신하거나 다시 색

인화하기에 앞서 체크아웃 조작을 호출해야 합니다. 항목을 완료했으면 체크인 조작을 호출하여 지속적인 잠금을 해제하고 다른 사용자가 항목을 갱신할 수 있도록 합니다. 항목을 작성한 후에는 작업을 완료할 때까지 다른 사용자가 변경할 수 없도록 체크아웃 상태를 계속 유지하는 옵션이 제공됩니다. 명시적 트랜잭션을 사용하여 항목을 체크아웃(또는 체크인)하는 경우, 트랜잭션이 롤백되면 체크아웃이 완료되지 않습니다. 암시적 트랜잭션을 사용하여 항목을 체크아웃하면 체크아웃이 확약됩니다. `checkin` 옵션 또는 메소드를 사용하여 항목을 다시 체크인하는 것은 응용프로그램의 책임입니다.

트랜잭션 처리

트랜잭션 범위는 클라이언트 API 호출로 제어할 수 있지만 주의 깊게 설계되어야 합니다. API 호출 세트를 트랜잭션으로 그룹화하려면 다음 단계를 완료하여 명시적으로 빌드해야 합니다.

1. `DKDatastoreICM` 클래스의 `startTransaction()` 메소드를 호출하십시오.
`DKDatastoreICM` 메소드에 대한 작업을 수행하여 모든 트랜잭션 단계를 완료하십시오.
2. 트랜잭션에 포함할 모든 API를 원하는 순서로 호출하십시오.
3. `commit` 또는 `rollback` 메소드를 호출하여 트랜잭션을 종료하십시오.

`startTransaction()`과 `rollback()` 또는 `commit()` 사이의 모든 API 호출은 하나의 트랜잭션으로 취급됩니다.

모든 API는 별도로 명시되지 않는 한, 트랜잭션에 포함될 수 있습니다. 세부사항은 온라인 API 참조서를 참조하십시오. 일부 관리 API는 명시적 트랜잭션에 포함될 수 없습니다. 예로는 항목 유형을 갱신 또는 정의하는 메소드가 있습니다.

다음은 Content Manager 트랜잭션의 항목 작성 및 갱신에 관련된 클래스 메소드 목록입니다.

`DKDatastoreICM.startTransaction()`

명시적 트랜잭션을 시작합니다.

`DKDatastoreICM.commit()`

데이터베이스에 트랜잭션 변경사항을 확약합니다.

`DKDatastoreICM.rollback()`

데이터베이스에서 트랜잭션 변경사항을 롤백 또는 제거합니다.

`DKDatastoreICM.checkOut()`

항목에 지속 쓰기 잠금을 획득합니다.

`DKDatastoreICM.checkIn()`

이전에 획득한 지속적 쓰기 잠금을 해제합니다.

`DKDatastoreICM.add()`

데이터베이스에 새 항목을 작성합니다.

DKDatastoreICM.updateObject()

항목을 갱신합니다. 이 메소드를 호출하기 전에 항목을 체크아웃해야 합니다.

DKDatastoreICM.retrieveObject()

데이터베이스에서 항목을 검색합니다.

DKDatastoreICM.deleteObject()

데이터베이스에서 항목을 삭제합니다.

DKDatastoreICM.moveObject()

항목을 다시 색인화합니다. 하나의 항목 유형에서 다른 항목 유형으로 항목을 이동합니다. 이 메소드를 호출하기 전에 항목을 체크아웃해야 합니다.

트랜잭션 정보에 대한 다른 대형 원본은 SItemUpdateICM 샘플입니다.

프로세스를 통한 문서 경로지정

Content Manager는 통합 문서 경로지정 서비스를 제공하여 비즈니스 프로세스를 통해 문서를 경로지정할 수 있도록 합니다. 문서 경로지정 API는 문서 경로지정을 사용하여 새 응용프로그램을 빌드하거나 문서 경로지정 기능을 기존의 응용프로그램에 추가할 수 있도록 합니다. 문서 경로지정은 다음과 같은 기능을 제공합니다.

- 문서 경로지정 기능이 Content Manager 트랜잭션에 포함되어 있으므로 문서 경로지정 프로세스에 있는 모든 항목이 동기화됩니다.
- 사용자가 액세스할 수 있는 작업만 제시합니다.
- 단일 감사 추적에는 문서 작성, 수정 및 경로지정에 대한 레코드가 포함됩니다.

기본 문서 경로지정 개념 및 용어에 대해서는 시스템 관리 안내서를 참조하십시오. 또한 샘플이 추가 문서 경로지정 정보의 원본이므로 샘플을 참조하십시오.

문서 경로지정 프로세스 이해

문서 경로지정은 프로세스, 작업 노드, 작업 목록 및 작업 패키지로 구성되어 있습니다. 시스템 관리자는 시스템 관리 클라이언트를 통해 작업 노드, 프로세스 및 작업 목록을 작성합니다. 프로세스는 작업 노드로 구성되어 있습니다. 프로세스에서 각 작업 노드는 해당 프로세스의 개별 단계입니다. 여러 방향으로 나누어지는 프로세스를 작성할 수 있습니다. 사용자는 작업 노드가 다음에 어떠한 분기로 이동하는지를 판별합니다. 사용자는 시스템 관리자가 정의하는 가능한 선택 목록에서 선택할 수 있습니다. 작업 노드를 정의할 때 서버 종료를 정의할 수 있습니다. 과부하 제한에 도달할 때 사용자에게 알려주고, 작업 노드에 진입 및 종료하기 위해 사용자 종료를 정의할 수 있습니다. 프로세스가 시작되면 작업 패키지가 작성됩니다. 작업 패키지는 경로지정 요소이며 작업 속성을 포함합니다. 작업 패키지의 속성은 항목 PID, 우선순위, 소유자 등으로 구성되어 있습니다.

컬렉션 지점은 추가 기능을 가진 작업 노드입니다. 지정된 항목 유형에 대해 지정된 수의 항목이 지정된 폴더에 존재하면 컬렉션 지점 노드의 작업 패키지는 프로세스 상의 다음 작업 노드에서 계속됩니다. 작업 목록은 사용자에게 할당된 작업 패키지를 정의합니다. 하나 이상의 작업 목록을 가질 수 있습니다. 각 작업 목록은 하나 이상의 작업 노드를 포함할 수 있습니다. 우선순위 또는 날짜별로 작업 목록에서 작업 패키지의 순서를 지정할 수 있습니다. 작업 목록에서 작업 노드의 순서도 정의할 수 있습니다.

작업 목록을 검색할 때 일시중단된 작업을 포함하거나 제외하도록 결과를 필터링할 수 있습니다. 작업 패키지는 알림 상태에 있을 수도 있습니다. 알림 상태는 시스템 관리자가 지정한 시간보다 더 오래 노드에 있던 작업 패키지입니다. 작업 노드는 둘 이상의 작업 목록에 있을 수 있습니다. 시스템 관리자가 작업 목록에 리턴된 패키지 수를 정의합니다.

문서 경로지정을 사용하여 수행할 수 있는 기본 조작에는 다음이 포함됩니다.

- 프로세스 시작
- 프로세스 종료
- 프로세스 계속
- 프로세스 일시중단
- 프로세스 재개
- 작업 목록에서 작업 얻기
- 작업 목록에서 다음 항목 얻기
- 프로세스 정의, 갱신 및 삭제
- 작업 노드 정의, 갱신 및 삭제
- 작업 목록 정의, 갱신 및 삭제

문서 경로지정 프로세스 설정

응용프로그램에 문서 경로지정 기능을 구현하는 데 사용할 수 있는 9개의 API가 있습니다. 온라인 API 참조서에 이 API 및 메소드에 대한 자세한 정보가 들어 있습니다. 9개의 문서 경로지정 API는 다음과 같습니다.

DKDocRoutingServiceICM

이 클래스는 프로세스를 관리하기 위한 메소드(시작, 종료, 계속, 일시중단 및 재개)를 제공합니다.

DKDocRoutingServiceMgmtICM

이 클래스는 헬퍼 클래스(DKProcessICM, DKWorkNodeICM 및 DKWorkListICM)를 관리하는 메소드를 제공합니다. DKDocRoutingServiceICM 오브젝트에서 DKDocRoutingServiceMgmtICM 오브젝트에 액세스할 수 있습니다.

DKProcessICM

이 클래스는 라이브러리 서버의 프로세스를 나타냅니다.

DKWorkNodeICM

이 클래스는 라이브러리 서버의 작업 노드를 나타냅니다.

DKWorkListICM

이 클래스는 라이브러리 서버의 작업 목록을 나타냅니다.

DKRouteListEntryICM

이 클래스는 프로세스가 취할 수 있는 경로(시작, 끝)를 정의합니다. 프로세스 오브젝트(즉, DKProcessICM)에는 경로 입력 오브젝트(즉, DKRouteListEntryICM)의 컬렉션이 포함되어 있습니다.

DKCollectionResumeListEntryICM

이 클래스는 작업 노드의 재개 목록에 있는 항목을 나타냅니다. 작업 노드는 DKCollectionResumeListEntryICM 오브젝트의 컬렉션을 포함할 수 있습니다.

DKWorkPackageICM

이 클래스는 라이브러리 서버의 작업 패키지를 나타냅니다. 프로세스가 시작되면 작업 패키지가 작성됩니다.

DKResumeListEntryICM

이 클래스는 재개 목록을 나타냅니다. 작업 패키지에는 재개 목록 컬렉션을 포함할 수 있습니다.

문서 경로지정 서비스 오브젝트 작성

아래 예제에서는 문서 경로지정 오브젝트를 작성하는 방법을 보여줍니다.

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);
```

완전한 샘플에 대해서는 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

새 일반 작업 노드 정의

작업 노드는 문서 경로지정 프로세스 정의에 대한 단계입니다. 사용자 응용프로그램에서 언제든지 종료할 수 있습니다. 사용자 응용프로그램에는 자체 종료 기준을 검증할 책임이 있습니다.

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  
//management 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);
```

작업 메모를 작성하는 완전한 예제에 대해서는 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

작업 노드 나열

listWorkNodeNames 메소드는 라이브러리 서버에 있는 모든 작업 노드 이름을 나열 하며, listWorkNodes 메소드는 라이브러리 서버의 작업 노드를 나타내는 DKWorkNodeICM 오브젝트의 컬렉션을 리턴합니다.

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);
```

작업 노드 나열에 대한 자세한 정보는 SDocRoutingListingICM 샘플을 참조하십시오.

새 컬렉션 지점 정의

컬렉션 지점은 시스템 정의 종료 기준을 가진 작업 노드이며, 특히 경로지정 폴더에 적용할 수 있습니다. 요구사항 세트는 프로세스가 재개되거나 이 지점 이후에 처리되기 전에 부합하도록 지정될 수 있습니다.

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
// management 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);
```

콜렉션 지점 정의에 대한 자세한 정보는 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

작업 목록 정의

작업 목록은 사용자가 작업 패키지 목록 또는 "다음" 작업 패키지를 가져올 수 있는 하나 이상의 작업 노드로 구성되어 있습니다. 작업 노드는 둘 이상의 작업 목록에 있을 수 있습니다. 작업 목록을 사용하면 시스템 관리자 또는 사용자 응용프로그램이 일반 사용자와 접촉하지 않고도 작업 할당을 동적으로 변경할 수 있습니다.

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);
```

작업 목록 정의에 대한 자세한 정보는 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

작업 목록 나열

listWorkListNames 메소드는 라이브러리 서버에 있는 모든 작업 목록 이름을 나열 하며, listWorkLists 메소드는 라이브러리 서버의 작업 목록을 나타내는 DKWorkListICM 오브젝트의 컬렉션을 리턴합니다.

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);
}
```

완전한 예제에 대해서는 SDocRoutingListingICM 샘플을 참조하십시오.

새 프로세스 및 연관된 경로 정의

문서 경로지정 프로세스는 작업 패키지가 따르는 정의된 경로입니다. 여러 경로지정 프로세스는 동일한 노드를 재사용할 수 있고 노드 간 여러 경로를 사용할 수도 있습니다.

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);
```

완전한 예제에 대해서는 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

문서 경로지정 프로세스 시작

아래 예제에서는 문서 경로지정 프로세스를 시작하는 방법을 보여줍니다.

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");
```

완전한 예제에 대해서는 SDocRoutingProcessingICM 샘플을 참조하십시오.

프로세스 종료

프로세스는 종료 노드에 도달하기 전에 명시적으로 종료될 수 있습니다. 프로세스가 종료되면 경로지정된 작업 패키지는 시스템에서 제거됩니다. 요구되는 프로세스를 종료하려면 프로세스 인스턴스로 경로지정된 작업 패키지의 PID 문자열을 알아야 합니다.

Java

```
routingService.terminateProcess(workPackagePidStr);
```

C++

```
routingService->terminateProcess(workPackagePidStr);
```

프로세스 종료에 대한 자세한 정보는 SDocRoutingProcessingICM 샘플을 참조하십시오.

프로세스 계속

continueProcess() 메소드는 지정된 작업 패키지의 항목 PID가 참조하는 항목을, 현재 작업 노드에서 선택에 의해 결정된 다음 작업 노드로 경로지정합니다. 지정된 작업 패키지는 라이브러리 서버에서 제거되며 지정된 소유자에 대해 새 작업 패키지가 작성됩니다. 항목 PID에 의해 참조되는 항목이 체크아웃된 경우 체크인됩니다. 새 작업 패키지의 PID가 리턴됩니다. 프로세스가 종료된 경우 널(null)이 리턴됩니다.

아래 코드 스니펫에서 현재 작업 노드에서 다음 작업 노드까지의 원인이 되는 선택의 이름은 "Continue"입니다. 현재 작업 패키지의 작업 패키지 PID 문자열은 메소드 호출에 지정되어 있습니다. 메소드 호출은 새 작업 패키지의 PID 문자열을 리턴합니다.

Java

```
workPackagePidStr = routingService.continueProcess  
(workPackagePidStr, "Continue", "icmadmin");
```

C++

```
char * userName = "icmadmin";

workPackagePidStr = routingService->continueProcess(workPackagePidStr,
    "Continue", userName);
```

완전한 예제에 대해서는 SDocRoutingProcessingICM 샘플을 참조하십시오.

프로세스 일시중단

문서 경로지정 프로세스의 인스턴스는 일정 시간(분) 동안 일시중단되거나 재개되기 전에 반드시 충족시켜야 할 요구사항 세트로 보류될 수 있습니다. 이것은 프로그래밍 환경의 프로세스 및 스레드와 관련이 없습니다. C++ 런타임 환경에서의 스레드 또는 프로세스는 정지되지 않습니다.

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);
```

완전한 예제에 대해서는 SDocRoutingProcessingICM 샘플을 참조하십시오.

프로세스 재개

일시중단된 프로세스(일시중단된 상태의 프로세스)는 명시적으로 재개될 수 있습니다. 또는 지정된 시간의 만기 후 또는 일부 정의된 요구사항이 충족된 후 암시적으로 재개될 수 있습니다. 이렇게 되면, 일시 중단된 상태에 있지 않은 프로세스가 일반 조작으로 리턴됩니다. resumeProcess 메소드는 일시중단이 지정된 지속 시간에 도달하고 재개 목

록이 충족하기 전에 지정된 작업 패키지의 일시중단 플래그를 거짓으로 재설정합니다.
연관된 작업 항목의 경로지정 또는 체크아웃이 수행되지 않습니다.

Java

```
routingService.resumeProcess(workPackagePidStr);
```

C++

```
routingService->resumeProcess(workPackagePidStr);
```

프로세스 재개에 대한 자세한 정보는 SDocRoutingProcessingICM 샘플을 참조하십시오.

작업 목록에서 작업 패키지 지속 식별자(PID) 문자열 나열

다음 코드 샘플에서는 지정된 작업 목록의 모든 작업 패키지에 대해 PID 문자열을 나열하는 방법을 보여줍니다.

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
```

완전한 예제에 대해서는 SDocRoutingListingICM 샘플을 참조하십시오.

작업 패키지 정보 검색

문서 경로지정 프로세스의 인스턴스가 진행 중일 때 작업 패키지는 경로지정 프로세스를 통해 이동하는 항목(항목 유형의 인스턴스)을 통한 운반 장비입니다. 작업 패키지는 전송 중인 프로세스 및 항목에 대해 필요한 모든 정보를 포함합니다. 작업 패키지는 응용프로그램이 필요에 따라 사용하고 조작할 수 있는 오브젝트입니다.

retrieveWorkPackage 메소드는 지정된 작업 패키지 PID(wpPidStringStr)가 참조하는 DKWorkPackageICM 오브젝트를 리턴합니다.

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;
```

완전한 예제에 대해서는 SDocRoutingProcessingICM 샘플을 참조하십시오.

문서 경로지정 프로세스 나열

다음 예제에서는 문서 경로지정 프로세스를 나열하는 방법을 보여줍니다.

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);
```

인쇄 기능은 SDocRoutingListingICM 샘플에 제공됩니다.

임시 경로지정

다음은 임시 경로지정 예제 프로시저입니다. 예를 들어, 시스템 관리 클라이언트는 작업 노드, 프로세스 및 작업 목록을 설정하는 데 사용됩니다.

1. 예를 들어, 두 작업 노드 N1 및 N2를 작성하십시오.

2. P1은 하나의 작업 노드 N1을 가지고 P2는 하나의 작업 노드 N2를 가지도록 두 노드 프로세스 P1 및 P2를 작성하십시오.

P1은 다음과 같습니다.

시작:	조치:	끝:
START	계속	N1
N1	계속	END

P2는 다음과 같습니다.

시작:	조치:	끝:
START	계속	N2
N2	계속	END

3. WL1은 하나의 작업 노드 N1을 가지고 WL2는 하나의 작업 노드 N2를 가지도록 두 작업 목록 WL1 및 WL2를 작성하십시오.

런타임 시 임시 경로지정을 구현하려면 다음 단계를 완료하십시오.

1. 문서 PID(예: ABC)를 가진 프로세스 P1을 시작하십시오. 작업 패키지 WP1이 작성됩니다. 작업 목록 WL1은 작업 노드 N1에서 작업 패키지 WP1을 표시합니다.
2. 문서 ABC를 프로세스 P1에서 프로세스 P2로 이동하려면 작업 패키지 WP1을 종료하고 동일한 문서(ABC)로 프로세스 P2를 시작하십시오. 작업 패키지 WP2가 작성됩니다.

작업 목록 WL2는 작업 노드 N2에서 작업 패키지 WP2를 표시합니다.

추가 예제를 보려면 SDocRoutingDefinitionCreationICM 샘플을 참조하십시오.

문서 경로지정 예제 조회

이 절에서는 예제 조회를 포함합니다. 조회 작성에 대한 자세한 정보는 211 페이지의 『조회 언어 이해』를 참조하십시오.

예제 1 자동차 문서를 찾고 활성화된 관련 작업 패키지만을 리턴합니다.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG = 0]
```

예제 2 자동차 문서를 찾고 "AccidentInvestigation" 프로세스에 있는 연관된 작업 패키지를 리턴합니다.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID]
```

예제 3 이름이 "Honda"인 자동차 문서를 찾고 "AccidentInvestigation" 프로세스에 있는 연관된 작업 패키지를 리턴합니다.

```

/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]

```

예제 4 자동차 문서를 찾고 "AccidentInvestigation" 프로세스의 "UnderReview" 단계에 있는 연관된 작업 패키지를 리턴합니다.

```

/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME = "UnderReview"]

```

예제 5 자동차 문서를 찾고 "AccidentInvestigation" 프로세스의 "UnderReview" 단계에 있는 일시중단된 작업 패키지만을 리턴합니다.

```

/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]

```

문서 경로지정에 대한 사용 권한 부여

사용자가 문서 경로지정 조작을 수행하려면 적절한 사용 권한을 가지고 있어야 합니다. 문서 경로지정과 연관된 사용 권한은 다음 테이블에 나열되어 있습니다. 프로세스, 작업 노드 및 작업 목록에는 항목에 대한 일반 사용 권한이 적용됩니다.

표 16. 문서 경로지정 사용 권한

사용 권한	설명	관련 API
ICM_PRIV_ITEM_UPDATE_WORK	사용자가 작업 패키지에 대해 다음을 수행할 권한이 있는지 여부를 확인하는 데 사용됩니다. 우선순위 설정 소유자 설정 재개 목록 설정 일시중단에 대한 지속 시간 설정	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM_ROUTE_START	사용자가 프로세스를 시작할 권한이 있는지 여부를 확인하는 데 사용됩니다.	startProcess
ICM_PRIV_ITEM_ROUTE_END	사용자가 프로세스를 종료할 권한이 있는지 여부를 확인하는 데 사용됩니다.	terminateProcess
ICM_PRIV_ITEM_GET_WORKLIST	사용자가 작업 목록에서 작업 패키지 또는 계수를 가져올 권한이 있는지 여부를 확인하는 데 사용됩니다.	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM_GET_WORK	사용자가 작업 패키지를 가져올 권한이 있는지 여부를 확인하는 데 사용됩니다.	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage

표 16. 문서 경로지정 사용 권한 (계속)

사용 권한	설명	관련 API
ICM_PRIV_ITEM_GET_ASGN_WORK	사용자가 다른 사용자가 소유한 작업 패키지를 가져올 권한이 있는지 여부를 확인하는 데 사용됩니다.	getNextWorkPackagePidString getNextWorkPackagegetCount listWorkPackagePidStrings
ICM_PRIV_ITEM_ROUTE	사용자가 작업 패키지를 경로지정할 권한이 있는지 여부를 확인하는 데 사용됩니다.	continueProcess

- `public final static String DK_ICM_DR_END_NODE = "END";`

기타 콘텐츠 서버에 대한 작업

dkDatastore 클래스를 사용하여 응용프로그램에서 해당 콘텐츠 서버에 적합한 콘텐츠 서버를 정의할 수 있습니다. 콘텐츠 서버는 Enterprise Information Portal에 대한 기본 인터페이스입니다. 각 콘텐츠 서버는 별도의 콘텐츠 서버 클래스를 가지고 있습니다.

콘텐츠 서버를 작성하려면 DKDatastorexx 클래스를 사용하십시오. 여기서 xx는 특정 콘텐츠 서버를 나타냅니다. 42 페이지의 표 7에 이러한 클래스가 나와 있습니다.

표 18. 서버 유형 및 클래스 이름 용어

콘텐츠 서버	클래스 이름
Content Manager 버전 8.2	DKDatastoreICM
이전 Content Manager	DKDatastoreDL
Content Manager OnDemand	DKDatastoreOD
AS/400용 Content Manager(AS/400용 VisualInfo)	DKDatastoreV4
OS/390용 Content Manager ImagePlus	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
Panagon Image Services(FileNET)	DKDatastoreFN
관계형 데이터베이스	DKDatastoreDB2, DKDatastoreJDBC(Java용), DKDatastoreODBC

콘텐츠 서버에 대한 콘텐츠 서버 작성 시 다음 클래스 및 인터페이스를 각각 구현하십시오.

dkDatastore

콘텐츠 서버를 나타내며 연결, 통신 및 콘텐츠 서버 명령의 실행을 관리합니다. dkDatastore는 조회 관리자 클래스를 추상화한 것입니다. 평가 메소드를 지원합니다.

dkDatastoreDef

콘텐츠 서버에 저장된 항목에 액세스하는 메소드를 사용합니다. 또한 해당 엔티티를 작성, 나열 및 삭제합니다. dkEntityDefs의 컬렉션을 유지보수합니다. 이 인터페이스에 대한 구체적인 클래스의 예제는 다음과 같습니다.

- DKDatastoreDefDL
- DKDatastoreDefOD

dkEntityDef

엔티티 정보에 액세스하는 메소드를 사용합니다. 또한 엔티티와 속성을 작성 및 삭제합니다. 이 클래스의 메소드는 다중 레벨 엔티티에 대한 액세스를 지원합니다.

니다. 콘텐츠 서버가 서브엔티티를 지원하지 않는 경우, DKUsageError 오브젝트를 생성합니다. 콘텐츠 서버가 다중 레벨 엔티티를 지원하는 경우, 이러한 콘텐츠 서버의 서브클래스에 대한 예외를 겹쳐쓰는 메소드를 구현해야 합니다. dkEntityDef 인터페이스에 대한 구체적인 클래스의 예제는 다음과 같습니다.

- DKIndexClassDefDL
- DKAppGrpDefOD

엔티티 정의에 대한 클래스 계층 구조가 그림 14에 나와 있습니다.

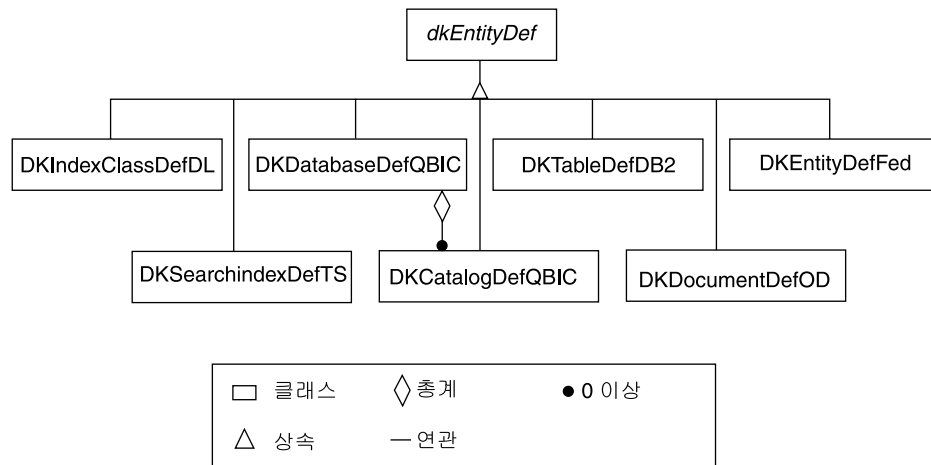


그림 14. 클래스 계층 구조

dkAttrDef

속성 정보에 액세스하고 속성을 작성 및 삭제하도록 메소드를 정의합니다. dkAttrDef에 대한 구체적인 클래스의 예제는 다음과 같습니다.

- DKAttributeDefDL
- DKFieldDefOD

dkServerDef

서버 정보에 액세스하도록 메소드를 정의합니다. dkServerDef에 대한 구체적인 클래스의 예제는 다음과 같습니다.

- DKServerDefDL
- DKServerDefOD

dkResultSetCursor

DDO 오브젝트 컬렉션을 관리하는 콘텐츠 서버 커서를 작성합니다. addObject, deleteObject 및 updateObject 메소드를 사용하려면 콘텐츠 서버 옵션 DK_CM_OPT_ACCESS_MODE를 DK_CM_READWRITE로 설정하십시오.

dkBlob

각 콘텐츠 서버의 BLOB(2진 대형 오브젝트) 데이터 유형에 공통적인 공용 인

터페이스를 선언합니다. dkBlob에서 파생된 구체적인 클래스는 이 공통 인터페이스를 공유하여 이기종 콘텐츠 서버에서 BLOB를 처리할 수 있도록 합니다. dkBlob에 대한 구체적인 클래스의 예제는 다음과 같습니다.

- DKBlobDL
- DKBlobOD

데이터 정의 클래스 및 클래스 계층 구조가 그림 15에 나와 있습니다.

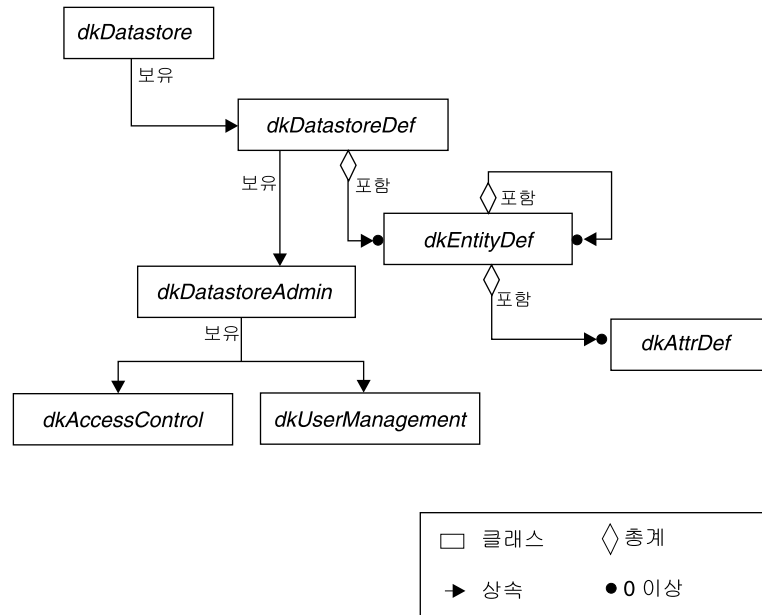


그림 15. 데이터 정의 클래스 계층 구조

dkDatastore 및 다른 공통 클래스에 대한 자세한 정보는 423 페이지의 『사용자 조정 콘텐츠 서버 커넥터 개발』을 참조하십시오.

이전 Content Manager에 대한 작업

이 절에서는 Content Manager 서버의 데이터에 액세스하는 방법 및 다음 작업을 수행하는 방법에 대해 설명합니다.

- 대형 오브젝트 처리
- DDO 사용
- 검색 엔진에서 XDO 사용
- 결합된 조회 사용
- 텍스트 검색 엔진 사용
- 이미지 검색 사용(QBIC)
- 워크플로우 및 작업함 사용

대형 오브젝트 처리

이전 Content Manager 커넥터에서 비동기 검색을 사용하여 대형 오브젝트를 하나씩 검색할 수 있습니다. 샘플 응용프로그램에 대해서는 CMBROOT\dk\samples 디렉토리의 TxdoAsyncRetDL을 참조하십시오.

Java 힙 크기 설정(Java 전용)

Java는 기본 초기 및 최대 힙 크기가 제한됩니다. 기본 초기 힙 크기는 1 048 576바이트이고 기본 최대 힙 크기는 16 777 216바이트입니다. Java 응용프로그램이 힙 크기보다 큰 오브젝트를 사용할 경우, 프로그램 실행 중 오류가 발생합니다. 응용프로그램의 최대 힙 크기를 늘리려면 Java 응용프로그램 실행 시 -mx 옵션을 사용하십시오. 예를 들면, 다음과 같습니다.

Java

```
java -mx40000000 yourApplication
```

DDO를 사용하여 이전 Content Manager 콘텐츠 표현

DKDatastoreDL과 연관된 DDO는 문서, 폴더, 부분, 항목, 항목 ID, 등급 등의 Enterprise Information Portal 문서 모델을 나타낼 수 있는 일부 특정 정보를 갖고 있습니다. 다음 절에서는 이 정보에 액세스하는 방법에 대해 설명합니다.

DDO 등록 정보

문서 또는 폴더인지에 상관없이 항목 유형은 DK_CM_PROPERTY_ITEM_TYPE 아래에 있는 등록 정보입니다. DDO의 항목 유형을 가져오려면 다음을 호출하십시오.

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 = cdd->getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (!any.isNull()) {
    unsigned short item_type = (unsigned short) any;
} ... // do something
```

등록 정보가 호출된 후 item_type은 문서인 경우 DK_CM_DOCUMENT, 또는 폴더인 경우 DK_CM_FOLDER입니다. if문은 등록 정보의 존재 여부를 확인합니다. 자세한 정보는 51 페이지의 『DDO에 등록 정보 추가』 및 55 페이지의 『DKDDO 및 속성 등록 정보 가져오기』를 참조하십시오.

지속 식별자(PID)

PID는 Enterprise Information Portal에 고유한 중요 정보를 포함합니다. 오브젝트 유형은 DDO가 속하는 색인 클래스를 나타내고 PID는 콘텐츠 서버에 있는 연관된 항목의 항목 ID를 포함합니다. 52 페이지의 『지속 식별자(PID) 작성』을 참조하십시오.

문서 표현

문서를 나타내는 DDO는 DK_CM_DOCUMENT로 설정된 등록 정보 DK_CM_PROPERTY_ITEM_TYPE을 가지고 있습니다. 해당 PID는 색인 클래스 이름을 오브젝트 유형으로 포함합니다. PID ID는 항목 ID와 동일합니다.

문서 내부의 부분은 DKPartsDL 오브젝트로 나타나는데, 이 오브젝트는 각각 DKBlobDL 오브젝트로 나타나는 BLOB(2진 대형 오브젝트)의 컬렉션입니다.

문서 DDO는 이름이 DKPARTS이고 값이 DKParts 오브젝트인 특정 속성을 가지고 있습니다.

문서의 각 부분을 얻으려면 먼저 DKParts를 검색한 다음 반복자를 작성하여 해당 부분에 대해 반복을 수행하십시오. 문서에 부분이 없는 경우, DKParts는 널(null)이거나 DKParts의 기본 행수는 0입니다.

결합된 조회(매개변수식 조회 및 텍스트 조회의 결합)와 연관된 문서는 DKRANK라는 일시적인 속성을 가질 수 있으며, 이 값은 텍스트 검색 엔진에서 계산한 정수 등급이 포함된 오브젝트입니다.

DKParts 오브젝트 작성 및 처리에 대한 자세한 정보는 280 페이지의 『문서 또는 폴더 작성, 갱신 및 삭제』, 289 페이지의 『문서 또는 폴더 검색』 및 98 페이지의 『문서 작성 및 DKPARTS 속성 사용』을 참조하십시오.

폴더 표현

폴더를 나타내는 DDO는 DK_CM_FOLDER와 동일한 등록 정보 DK_CM_PROPERTY_ITEM_TYPE을 가지고 있습니다. 문서 DDO와 마찬가지로 PID는 오브젝트 유형으로 색인 클래스 이름을 포함하며 PID의 ID에 항목 ID를 포함합니다.

DKFolder 오브젝트는 폴더 내의 목차를 나타냅니다. DKFolder 오브젝트는 DDO의 컬렉션입니다. 각 DDO는 폴더 항목인 문서 또는 폴더를 나타냅니다. DDO 폴더 DKFOLDER라는 속성을 가지고 있으며, 이 값은 DKFolder 오브젝트입니다.

폴더의 각 DDO 구성원을 얻으려면 먼저 DKFolder 오브젝트를 검색한 다음 반복자를 작성하여 각 항목 구성원에 액세스하십시오. 폴더에 구성원이 없는 경우, DKFolder는 널(null)이지만 DKFOLDER 속성은 항상 콘텐츠 서버에서 작성한 DDO 폴더에 존재합니다.

DKFolder 오브젝트 작성 및 처리에 대한 자세한 정보는 『문서 또는 폴더 작성, 갱신 및 삭제』, 289 페이지의 『문서 또는 폴더 검색』 및 101 페이지의 『폴더 작성 및 DKFOLDER 속성 사용』을 참조하십시오.

문서 또는 폴더 작성, 갱신 및 삭제

이 절에서는 문서 및 폴더 작성, 갱신 및 삭제와 관련된 프로세스에 대해 설명합니다.

문서 작성

문서를 작성하고 이 문서의 지속적 데이터를 콘텐츠 서버에 저장하려면 DDO를 작성하여 항목 ID를 제외한 모든 속성 및 기타 정보를 설정하십시오. 항목 ID는 콘텐츠 서버에 의해 할당되고 리턴됩니다. 다음 예제에 이전의 몇몇 예제가 결합되어 있습니다.

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

...
...
// continued...
// create and add some more blobs
// to the collection as necessary
```

C++(계속)

```
// 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();
```

이전 예제의 마지막 단계에서 콘텐츠 서버(정보 포함)에 문서를 작성했습니다. 문서 DDO를 콘텐츠 서버에 추가할 때마다 DKParts 컬렉션 내의 모든 부분을 포함한 해당 속성이 모두 추가됩니다.

DDO 폴더 추가에 대해서도 동일한 프로세스를 사용합니다. DKFOLDER 컬렉션 구성원은 폴더의 구성요소로서 콘텐츠 서버에 추가됩니다. 폴더에는 기존의 문서 및 폴더인 해당 구성원의 목차가 들어 있습니다. 따라서 DDO 폴더를 추가하기 전에 콘텐츠 서버에 모든 폴더 구성원을 작성하십시오.

Java

같은 문서를 같은 유형의 다른 콘텐츠 서버에 추가할 수 있습니다. 이 문서를 LIBSV와 동일한 구조를 가진 색인 클래스 LIBSV2가 들어 있는 Content Manager 서버 LIBSRVRN에 추가하려면 다음 예제를 사용하십시오.

```
// ----- 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++

같은 문서를 같은 유형의 다른 콘텐츠 서버에 추가할 수 있습니다. 예를 들어, 문서를 서버 LIBSRVRN에 추가할 수 있습니다. 이 서버에는 GRANDPA와 동일한 구조를 가진 색인 클래스 GRANDPA2가 들어 있습니다.

```
// 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
```

문서 또는 폴더 갱신

문서 또는 폴더를 갱신하려면 다음을 수행하십시오.

1. 항목 ID 및 오브젝트 유형을 설정하십시오.
2. 해당 속성을 갱신하거나 DKParts 컬렉션에 추가하십시오.
3. 갱신 메소드를 호출하여 변경사항을 저장하십시오.

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();
```

갱신 메소드를 호출한 후에는 콘텐츠 서버의 Title 속성값이 갱신됩니다. 해당 콘텐츠가 변경되지 않으면 이 문서의 부분은 갱신되지 않습니다. 갱신 메소드 호출 시 서버에 대한 연결이 유효해야 합니다.

위와 유사한 단계를 사용하여 DDO 폴더를 갱신하십시오. 속성값을 갱신하거나 DKFolder에 요소를 추가 또는 갱신한 다음 갱신 메소드를 호출하십시오.

부분 갱신

DKParts 오브젝트를 사용하여 문서의 부분 컬렉션을 나타내십시오.

DKParts는 DKSequentialCollection의 서브클래스입니다. DKParts는 순차 컬렉션 기능을 상속하는 것 외에 컬렉션에 부분을 추가 및 제거하는 두 가지 메소드를 더 갖고 있습니다. 이 메소드도 변경사항을 즉시 콘텐츠 서버에 저장합니다.

문서는 이미 콘텐츠 서버에 존재해야 합니다.

구성원 추가 및 제거: 다음 예제에서는 문서에 일부를 추가합니다.

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);
```

컬렉션 및 콘텐츠 서버에서 newPart를 제거하려면 다음을 사용해야 합니다.

Java

```
parts.removeMember(addo, newPart);
```

C++

```
parts->removeMember(ddo, newPart);
```

DKParts의 removeMember 메소드는 실제로 콘텐츠 서버에서 부분의 지속 사본을 삭제합니다.

문서 DDO에서 갱신, 추가 및 제거의 차이점: DKParts의 addMember 및 removeMember 메소드는 컬렉션과 콘텐츠 서버에서 부분을 추가 및 제거하는 데 편리함을 제공합니다. 이 메소드는 문서 DDO의 갱신 메소드보다 빠릅니다. DDO의 갱신 메소드는 변경된 DKParts와 모든 구성원을 비롯하여 DDO의 모든 속성을 갱신합니다. 단계는 다음과 같습니다.

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();  
...
```

폴더 갱신

DKFolder 오브젝트를 사용하여 폴더 내의 문서 및 폴더 컬렉션을 나타냅니다. 콘텐츠 서버에서 폴더는 모든 실제 오브젝트를 유지하는 대신 해당 오브젝트를 참조하는 목록을 보유합니다.

DKFolder는 DKSequentialCollection의 서브클래스입니다. DKFolder는 순차 컬렉션 메소드를 상속하는 것 외에 컬렉션에 구성원(문서 또는 폴더)을 추가 또는 제거하는 두 가지 기능을 더 갖고 있으며 이러한 변경사항을 즉시 저장합니다.

추가 또는 제거할 문서나 폴더는 콘텐츠 서버에 이미 존재해야 합니다.

구성원 추가 및 제거: 다음 예제에서는 다른 문서 또는 DDO 폴더를 DDO 폴더에 추가하는 방법을 보여줍니다.

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);
```

추가할 다른 문서 또는 폴더에 대해 newMember 및 folderDDO가 모두 콘텐츠 서버에 존재해야 합니다.

마찬가지로 컬렉션 및 콘텐츠 서버에서 newMember를 제거하려면 다음 예제를 사용하십시오.

Java

```
folder.removeMember(folderDDO, newMember);
```

C++

```
folder->removeMember(folderDDO, newMember);
```

중요사항: 폴더에서 구성원을 제거하면 폴더 목차에서 해당 구성원만 제거됩니다. `removeElementAt` 함수를 사용하면 메모리나 콘텐츠 서버에서 구성원이 삭제되지 않습니다.

폴더 DDO에서 갱신, 추가 및 제거의 차이점: DKFolder의 `addMember` 및 `removeMember` 메소드는 컬렉션과 콘텐츠 서버에서 문서 또는 폴더를 추가 및 제거하는 데 편리함을 제공합니다. 이 메소드는 폴더 DDO의 갱신 메소드보다 빠릅니다.

DDO의 갱신 메소드는 DKFolder와 모든 구성원을 비롯하여 DDO의 모든 속성을 갱신하는 반면, `addMember` 및 `removeMember` 메소드는 폴더 목차에만 구성원을 추가 또는 제거합니다.

문서 또는 폴더 삭제

콘텐츠 서버에서 문서 또는 폴더를 삭제하려면 DDO의 `del` 메소드를 사용하십시오.

Java

```
ddo.del();
```

C++

```
ddo->del();
```

DDO에는 항목 ID와 오브젝트 유형 세트가 있어야 하고 콘텐츠 서버에 올바르게 연결되어 있어야 합니다.

폴더도 삭제하려면 위의 명령문을 사용하십시오. 지속 데이터만 삭제되고 DDO의 인메모리 사본은 변경되지 않습니다. 따라서 이 DDO를 나중에 응용프로그램의 동일한 또는 다른 콘텐츠 서버에 다시 추가할 수 있습니다. 자세한 정보는 280 페이지의 『문서 작성』을 참조하십시오.

문서 또는 폴더 검색

이전 Content Manager 콘텐츠 서버를 나타내는 DKDatastoreDL에서 문서를 검색하려면 문서의 색인 클래스 이름 및 항목 ID를 알아야 합니다. 또한 DDO를 콘텐츠 서버와 연관시킨 다음 연결을 설정해야 합니다.

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();
```

검색을 호출한 후 모든 DDO의 속성값은 콘텐츠 서버에 저장된 지속 데이터의 값으로 설정됩니다. 문서에 부분이 포함되어 있으면 DKPARTS 속성은 DKParts 오브젝트로 설정됩니다. 그러나 이 컬렉션에 있는 각 부분의 콘텐츠는 검색되지 않습니다. 부분이 대 형일 수 있으므로 모든 부분을 한번에 메모리에서 검색해서는 안됩니다. 원하는 부분만 명시적으로 검색하는 것이 더 좋습니다.

DDO가 조회 옵션 CONTENT=NO를 사용하여 실행된 매개변수식 조회 결과인 경우, DDO 는 비어 있습니다(속성값이 없음). 그러나 검색하는 데 필요한 모든 정보는 이미 설정 되어 있습니다.

부분 검색

DDO 검색 후 다음과 같이 DKPARTS 속성에 저장된 해당 부분을 검색할 수 있습니다.

Java

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

C++

```
DKAny any = ddo->getDataByName(DKPARTS);  
DKParts* parts = (DKParts*) any.value();
```

이 예제에서는 DKPARTS 속성이 있다고 가정합니다. 그 속성이 존재하지 않는 경우, 예외가 생성됩니다. 먼저 데이터 ID를 확보한 다음 0인지를 테스트하여 속성값을 추출하는 예제에 대해서는 292 페이지의 『폴더 검색』을 참조하십시오.

각 부분을 검색하려면 반복자를 작성하여 컬렉션을 모두 처리한 후 각 부분을 검색해야 합니다. 98 페이지의 『문서 작성 및 DKPARTS 속성 사용』을 참조하십시오.

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;
}
```

매개변수식 조회의 DDO 결과와 마찬가지로 DKParts 컬렉션 내부의 각 부분 XDO는 비어 있습니다(컨텐츠 세트가 포함되어 있지 않음). 그러나 검색에 필요한 정보는 모두 들어 있습니다. 해당 컨텐츠 및 관련 정보를 메모리로 가져오려면 검색 메소드를 호출 하십시오.

Java

```
blob.retrieve();
```

C++

```
blob->retrieve();
```

폴더 검색

문서 DDO를 검색했던 방법과 동일한 방법으로 폴더 DDO를 검색하십시오. 검색 후 폴더 DDO에는 속성 DKFOLDER를 비롯하여 모든 속성 세트가 포함되어 있습니다. 이 속성값은 이 폴더의 DDO 구성원 컬렉션인 DKFolder 오브젝트로 설정됩니다. DKParts 오브젝트의 부분과 같이 이러한 구성원 DDO에는 검색에 필요한 정보만 들어 있습니다. 다음과 같이 폴더 DDO를 검색할 수 있습니다.

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;
}
```

또한 101 페이지의 『폴더 작성 및 DKFOLDER 속성 사용』을 참조하십시오.

텍스트 검색의 이해(텍스트 검색 엔진)

텍스트 검색 엔진 제품은 다양한 조회 유형을 지원합니다.

- 『논리 조회』
- 295 페이지의 『텍스트 조회』
- 295 페이지의 『혼용 조회』
- 295 페이지의 『근접 조회』
- 296 페이지의 『GTR(Global Text Retrieval) 조회』

조회에서 리턴된 텍스트 검색 항목 ID, 부분 번호 및 등급 정보를 사용하여 이전 Content Manager 서버에서 문서를 검색하는 XDO를 작성할 수 있습니다.

텍스트 검색 엔진을 나타내려면 DKDatastoreTS 오브젝트를 사용하십시오. 텍스트 검색 엔진은 실제로 데이터는 저장하지 않고 이전 Content Manager에 저장된 데이터만 색인화하여 텍스트 검색을 지원합니다. 텍스트 검색 결과는 Content Manager에서 문서의 위치를 설명하는 항목 ID입니다. 문서를 검색하려면 이 ID를 사용하십시오.

DKDatastoreTS 오브젝트는 add, update, retrieve 및 delete 함수를 지원하지 않습니다. 그러나 이 콘텐츠 서버를 조회할 수 있습니다. 텍스트 검색 엔진에 의해 색인화된 Content Manager에 데이터를 추가하는 방법에 대한 정보는 307 페이지의 『텍스트 검색 엔진에 의해 색인화될 데이터 로드』를 참조하십시오.

논리 조회

논리 조회는 논리 연산자로 분리된 단어와 구문으로 구성됩니다. 구문은 작은따옴표(')로 묶어야 합니다. 구문은 리터럴 문자열로 취급됩니다.

다음 예제에서는 TMINDEX 텍스트 검색 색인에서 www라는 단어나 web site라는 구문이 들어 있는 모든 텍스트 문서를 검색하는 조회 문자열을 작성합니다.

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)";
```

텍스트 조회

텍스트 조회는 중괄호({ })로 묶은 단어, 구문으로 구성됩니다. 단어끼리 서로 인접할 필요는 없습니다. 다음 예제에서는 TMINDEX 텍스트 검색 색인에서 web site라는 텍스트가 들어 있는 모든 텍스트 문서를 검색하는 조회 문자열을 작성합니다.

Java

```
String cmd = "SEARCH=(COND=({web site}));" +  
            "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=({Web site}));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

혼용 조회

혼용 조회는 논리 조회 다음에 텍스트 조회가 수행되도록 구성됩니다. 다음 예제에서는 TMINDEX 텍스트 검색 색인에서 IBM 및 UNIX라는 단어와 web site라는 텍스트가 들어 있는 모든 텍스트 문서를 검색하는 조회 문자열을 작성합니다.

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)";
```

근접 조회

근접 조회는 동일한 문서, 단락 또는 구문에 있는 검색 인수의 순서를 찾습니다. 다음 예제에서는 TMINDEX 텍스트 검색 색인을 사용하여 동일한 단락에서 rational numbers라는 구문과 math라는 단어가 들어 있는 모든 텍스트 문서를 검색하는 조회 문자열을 작성합니다.

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)";
```

이 유형의 조회에는 최소한 두 개의 검색 인수가 필요합니다.

GTR(Global Text Retrieval) 조회

GTR 조회는 일본어나 중국어와 같은 2바이트 문자 세트(DBCS) 언어에 대해 최적화되었습니다. GTR은 1바이트 문자 세트(SBCS) 언어도 지원합니다. 2바이트 문자는 모두 작은따옴표(')로 묶어야 합니다. 검색할 구문이 지정된 문자 코드 세트 및 언어에 있는지 확인하십시오.

다음 예제에서는 IBM marketing이라는 구문이 포함된 모든 텍스트 문서를 검색하는 GTR을 보여줍니다. 구문에 대한 유사성 정도를 나타내기 위해 MATCH 키워드가 설정됩니다.

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)";
```

텍스트 검색 콘텐츠 서버 옵션 DK_OPT_TS_CCSID(코드화된 문자 세트 식별자(ID)) 및 DK_OPT_TS_LANG(언어 식별자(ID))가 적절하게 설정되었는지 확인하십시오. DK_OPT_TS_CCSID의 기본값은 DK_CCSID_00850입니다. DK_OPT_TS_LANG의 기본값은

DK_LANG_ENU입니다. 이러한 값은 텍스트 조회의 글로벌 기본값으로 사용됩니다. 자세한 정보는 온라인 API 참조서를 참조하십시오.

또한 다음 예제에 표시된 것처럼 특정 CCSID 및 LANG 정보를 입력할 수 있습니다. CCSID와 LANG 모두 지정해야 합니다. 하나의 값을 다른 값 없이 지정할 수는 없습니다.

DDO를 사용하여 텍스트 검색 엔진 정보 표현

DKDatastoreTS 오브젝트와 연관된 DDO를 사용하여 텍스트 검색 결과를 나타냅니다.

DKDatastoreTS는 DKDatastoreDL 오브젝트처럼 등록 정보 항목 유형을 포함하지 않습니다. ID의 형식도 다릅니다. 텍스트 조회로부터 발생한 DDO는 항목 내부의 텍스트 부분에 해당합니다. DDO에는 다음과 같은 표준 속성이 포함됩니다.

DKDLITEMID

이 텍스트가 포함되는 항목 ID. 이 항목 ID를 사용하면 콘텐츠 서버에서 전체 항목을 검색할 수 있습니다.

DKPARTNO

이 텍스트 부분에 대한 정수 부분 번호. 항목 ID로 부분 번호를 사용하면 콘텐츠 서버에서 텍스트 부분을 검색할 수 있습니다.

DKREPTYPE

이 텍스트 부분의 RepType. 항목 ID 및 부분 번호와 함께 이 속성을 사용하면 콘텐츠 서버에서 텍스트 부분을 검색할 수 있습니다.

DKRANK

텍스트 조회의 결과에 대한 이 부분의 관련도를 나타내는 정수 등급. 등급이 높을수록 좀더 정확하게 일치함을 의미합니다. 자세한 정보는 온라인 API 참조서를 참조하십시오.

DKDSIZE

논리 조회 결과에서 단어가 발생한 횟수를 나타내는 정수. 자세한 정보는 온라인 API 참조서를 참조하십시오.

DKRCNT

논리 검색이 일치함을 나타내는 정수. 자세한 정보는 온라인 API 참조서를 참조하십시오.

텍스트 검색 DDO의 PID는 다음 정보를 포함합니다.

콘텐츠 서버 유형

TS.

콘텐츠 서버 이름

이 이름은 콘텐츠 서버 연결에 사용됩니다.

오브젝트 유형

텍스트 검색 색인.

ID 텍스트 검색 엔진 문서 ID.

연결 설정

DKDatastoreTS 오브젝트는 두 개의 연결 함수와 하나의 연결 해제 함수를 제공합니다. 일반적으로 DKDatastoreTS 오브젝트를 작성한 다음 이 오브젝트에 연결하고, 조회를 실행한 후 작업이 완료되면 연결 해제합니다. 다음 예제에서는 텍스트 검색 서버 TM을 사용하는 첫 번째 연결 함수를 보여줍니다.

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM", "", "", "");
.... // run a query
dsTS.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TConnectTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","", "", "");
... // do some work
dsTS.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TConnectTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

다음 예제에서는 호스트 이름이 apollo, 포트 번호가 7502, TCP/IP 통신 유형이 DK_CTYP_TCPIP인 텍스트 검색 서버를 사용하는 두 번째 연결 함수를 보여줍니다.

```
dsTS.connect("apollo", "7502", DK_CTYP_TCPIP);
```

다음 예제에서는 텍스트 검색 서버 호스트 이름 apollo, 포트 번호 7502, 연결 유형 T(TCP/IP)를 사용하는 첫 번째 연결 함수를 보여줍니다.

```
dsTS.connect("apollo", "", "", "PORT=7502; COMMTYPE=T");
```

다음 예제에서는 이름이 TM인 텍스트 검색 서버와 사용자 ID가 FRNADMIN, 암호가 PASSWORD인 라이브러리 서버 LIBSRVR2를 사용하는 첫 번째 연결 메소드를 보여줍니다.

다음 예제에서는 텍스트 검색 서버 이름 TM, 라이브러리 이름 LIBSRVRN, 사용자 ID FRNADMIN 및 암호 PASSWORD를 사용하는 첫 번째 연결 함수를 보여줍니다.

```
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
```

연결 문자열이라고도 하는 마지막 매개변수인 LIBACCESS를 사용하여 일련의 매개변수를 전달할 수 있습니다.

팁: 텍스트 검색 엔진에 연결 시 시간이 종료되는 것을 방지하려면 텍스트 검색 엔진에 연결하여 조회를 실행한 다음 즉시 연결 해제하십시오. 연결을 열어 놓지 마십시오.

텍스트 검색 옵션 가져오기 및 설정

텍스트 검색은 해당 함수를 사용하여 설정하거나 가져올 수 있는 몇 가지 옵션을 제공합니다. 옵션 목록 및 해당 설명에 대해서는 온라인 API 참조서를 참조하십시오. 다음 예제에서는 텍스트 검색 문자 코드 세트에 대해 옵션을 설정하고 가져오는 방법을 보여줍니다.

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option = new Integer(DK_TS_CCSSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSSID);
```

C++

```
DKAny input_option = DK_CCSSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSSID, input_option);
dsTS.getOption(DK_OPT_TS_CCSSID, output_option);
```

output_option은 오브젝트이지만 일반적으로 Integer로 캐스트됩니다

팁: 검색 옵션 CCSID 및 LANG은 함께 이동합니다. 하나의 옵션이 지정되면 다른 옵션도 지정되어야 합니다. 기본 CCSID 및 LANG은 DKDatastoreTS 옵션인 DK_OPT_TS_CCSSID 및 DK_OPT_TS_LANG에 지정됩니다. 콘텐츠 서버 옵션 목록과 해당 설명에 대해서는 온라인 API 참조서를 참조하십시오.

하나의 조회 용어에 대해 두 개 이상의 검색 옵션을 지정할 수 있습니다. 검색 옵션은 쉼표로 구분합니다. 다중 검색 용어에 대한 예제는 296 페이지의 『GTR(Global Text Retrieval) 조회』에 나와 있습니다.

SC(단일 요구 문자) 및 MC(선택적 문자의 순서)가 모두 검색 옵션인 경우, SC 검색 옵션을 먼저 지정해야 합니다. 예를 들면, \$SC=?,MC=*\$ U?I*와 같습니다.

서버 나열

DKDatastoreTS 오브젝트는 연결할 수 있는 텍스트 검색 서버를 나열하는 함수를 제공합니다. 다음 예제에서는 서버의 목록을 검색하는 방법을 보여줍니다.

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);
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

서버 목록은 DKServerInfoTS 오브젝트의 DKSequentialCollection에 리턴됩니다. DKServerInfoTS 오브젝트를 가져온 후에는 서버 이름과 위치를 검색할 수 있습니다. 그런 다음 서버 이름을 사용하여 서버에 대한 연결을 설정할 수 있습니다.

스키마 나열

DKDatastoreTS 오브젝트는 스키마를 나열하는 함수를 제공합니다. 텍스트 검색의 경우, 이것은 텍스트 검색 색인입니다. 다음 예제에서는 색인 목록을 검색하는 방법을 보여줍니다.

색인의 목록은 DKIndexTS 오브젝트의 DKSequentialCollection 오브젝트에 리턴됩니다. DKIndexTS 오브젝트를 가져온 후에는 이름 및 라이브러리 ID와 같은 색인 정보를 검색할 수 있으며, 이 색인은 조회를 형성하는 데 사용할 수 있습니다.

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
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogTS.cpp)은 Cmbroot/Samples/cpp/dl 디렉토리에 있습니다. 또한 컬렉션 삭제에 대한 정보는 112 페이지의 『컬렉션의 메모리 관리(C++ 전용)』를 참조하십시오.

검색 엔진에 의해 XDO 색인화

텍스트 검색 엔진 및 이미지 검색으로 데이터 항목을 검색하기 전에 데이터를 먼저 색인화해야 합니다. 색인화를 수행하려면 SearchEngine, SearchIndex 및 SearchInfo의 세 가지 값이 필요합니다.

SearchIndex 등록 정보의 값은 검색 서비스 이름 및 검색 색인 이름을 결합한 것입니다. 예를 들어, 시스템 관리 클라이언트에서 TM이라는 텍스트 검색 서버 및 이와 연관된 TMINDEX라는 검색 색인을 정의한 경우, SearchIndex의 값은 TM-TMINDEX가 됩니다.

텍스트 검색 엔진에 의해 색인화될 오브젝트의 경우 SearchEngine의 값은 SM이어야 하고, 이미지 검색을 사용한 조회에 의해 색인화될 데이터 항목의 경우 SearchEngine의 값은 QBIC여야 합니다.(이미지 검색에 대한 자세한 정보는 319 페이지의 『이미지 검색 용어 및 개념 이해』를 참조하십시오.)

QBIC의 SearchIndex는 세 개의 이름(QBIC 데이터베이스 이름, QBIC 카탈로그 이름 및 QBIC 서버 이름)을 결합한 것입니다. 예를 들어, QBIC 데이터베이스 이름은 SAMPLEDB, QBIC 카탈로그 이름은 SAMPLECAT, QBIC 서버 이름은 QBICSRV이면 SearchIndex의 올바른 값은 SAMPLEDB-SAMPLECAT-QBICSRV가 됩니다.

데이터 로드 또는 폴더 작성 및 데이터 로드 방법에 대한 예제는 CMBROOT\Samples\java\d1 디렉토리의 LoadSampleTSQBICDL.java 및 LoadFolderTSQBICDL.java를 참조하십시오.

중요사항: 검색 엔진에 의해 색인화될 부분 오브젝트를 추가할 때 RepType은 설정하지 마십시오. 현재 텍스트 검색 엔진은 기본 RepType FRN\$NULL에서만 작동합니다.

텍스트 검색 엔진에 의해 색인화될 XDO 추가: 다음 예제에서는 색인화할 XDO를 추가하는 방법을 보여줍니다.

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++(계속)

```
//--- 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;
}
```

중요사항: 검색 엔진에 의해 색인화될 부분 오브젝트를 추가할 때 RepType(표현 유형)을 설정하지 마십시오. 텍스트 검색 엔진은 기본 RepType FRN\$NULL에서만 작동합니다.

텍스트 검색 엔진에 의해 색인화될 데이터 로드: 텍스트 검색 엔진에 의해 색인화될 데이터를 Content Manager로 로드하려면 색인 및 텍스트 검색 색인을 모두 작성해야 합니다.

텍스트 검색 색인을 작성하려면 먼저 텍스트 검색 서버가 실행 중이어야 합니다. 환경이 제대로 설정되어 있는지 확인하십시오. 이렇게 하려면 샘플 CMBROOT\Samples 디렉토리의 TListCatalogDL 및 TListCatalogTS를 사용자 조정하고 실행할 수 있어야 합니다.

Content Manager에 텍스트 검색 엔진에 의해 색인화되는 부분을 작성하려면 59 페이지의 『확장 데이터 오브젝트(XDO)에 대한 작업』을 참조하십시오.

데이터를 Content Manager로 로드한 후 DKDatastoreDL 클래스에서 wakeUpService 함수를 사용하여 문서 큐에 문서를 넣으십시오. 이 함수는 검색 엔진 이름을 매개변수로 취합니다.

그런 다음 Content Manager 텍스트 검색 관리 창에서 다음 단계를 완료하십시오.

1. 텍스트 검색 서버를 두 번 누르십시오.
2. 텍스트 검색 색인을 두 번 누르십시오.
3. 색인을 누르십시오.

이렇게 하면 문서 조회에 있는 문서를 색인화합니다. 색인화가 완료된 후에는 텍스트 검색 조회를 수행할 수 있습니다.

텍스트 검색 관리에 대한 자세한 정보는 시스템 관리 안내서를 참조하십시오.

텍스트 구조 문서 지원 사용

텍스트 구조 문서는 태그가 있는 텍스트(예: HTML 파일)로 구성됩니다. 문서 모델은 문서 구조를 정의하고 텍스트 검색 엔진은 태그 사이의 단어 또는 구문을 검색할 수 있습니다.

다음과 같은 방법으로 구조 문서에서 텍스트 조회를 수행할 수 있습니다.

1. 문서 모델을 작성하십시오. 문서 모델은 섹션을 포함하고, 각 섹션은 사용된 섹션 이름을 포함합니다. 예를 들어, 다음과 같습니다.

```
<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 문서 모델을 사용하는 텍스트 검색 색인을 작성하십시오.
3. 텍스트 검색 색인에 대해 색인화 규칙을 설정하고 기본 문서 형식(예: HTML 파일의 경우 DK_TS_DOCFMT_HTML)을 지정하십시오.

4. 부분 오브젝트를 Content Manager 서버에 추가하십시오.
5. 텍스트 검색 색인에 대해 색인화 프로세스를 시작하십시오.

이 예제에서는 시스템에 정의된 문서 모델을 나열하는 방법을 보여줍니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListDocModelsTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListDocModelsTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

다음 예제에서는 문서 모델 작성 방법을 보여줍니다.

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("SAMPORPBODY");
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();
```

추가 예제에 대해서는 CMBROOT\Samples\java\d1 디렉토리의
TCreateDocModelTS.java 및 TCreateStructDocIndexTS.java를 참조하십시오.
오.

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 CORPTITLE");
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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TCreateDocModelTS.cpp) 및 (TCreateStructDocIndexTS.cpp)가 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

다음 예제에서는 문서 모델을 사용하는 텍스트 검색 색인의 색인화 규칙을 작성 및 설정하는 방법을 보여줍니다.

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(계속)

```
indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);

dsAdmin.setIndexingRules(indexRules);

dsTS.disconnect();
dsTS.destroy();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TCreateStructDocIndexTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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++(계속)

```
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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TCreateStructDocIndexTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

다음 예제에서는 비동기인 색인화 프로세스의 시작 방법을 보여줍니다. 시스템 관리 프로그램을 사용하여 색인화 프로세스를 시작하고 그 상태를 점검할 수 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TIndexingTS.java)은
CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TIndexingTS.cpp)은
Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

상태 점검 예제에 대해서는 CMBROOT\Samples 디렉토리의 TCheckStatusTS를 참조하십시오. 예제에서는 대기열에 있는 요청이 스케줄된 문서 큐에서 기본 또는 보조 대기열로 이동했는지 여부를 점검합니다. 색인화 오류가 발생하는 경우, 텍스트 검색 인스턴스 디렉토리의 imldiag.log 파일을 점검할 수 있습니다.

다음 예제에서는 위에서 정의한 텍스트 검색 색인 및 문서 모델을 기준으로 구조 문서 텍스트 조회를 실행하는 방법을 보여줍니다.

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair 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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteStructDocTS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteStructDocTS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

컨텐츠별 이미지 검색

이미지 검색 서버는 이미지 유형을 지정하거나 이미지 예제를 제공한 경우 저장된 이미지를 검색할 수 있습니다.

그림 16에서는 이미지 검색 서버에 연결하는 샘플 응용프로그램을 보여줍니다. 이미지 검색 서버는 QBIC(Query by Image Content) 기술을 사용하여 유사한 색, 배치 및 패턴을 검색합니다.

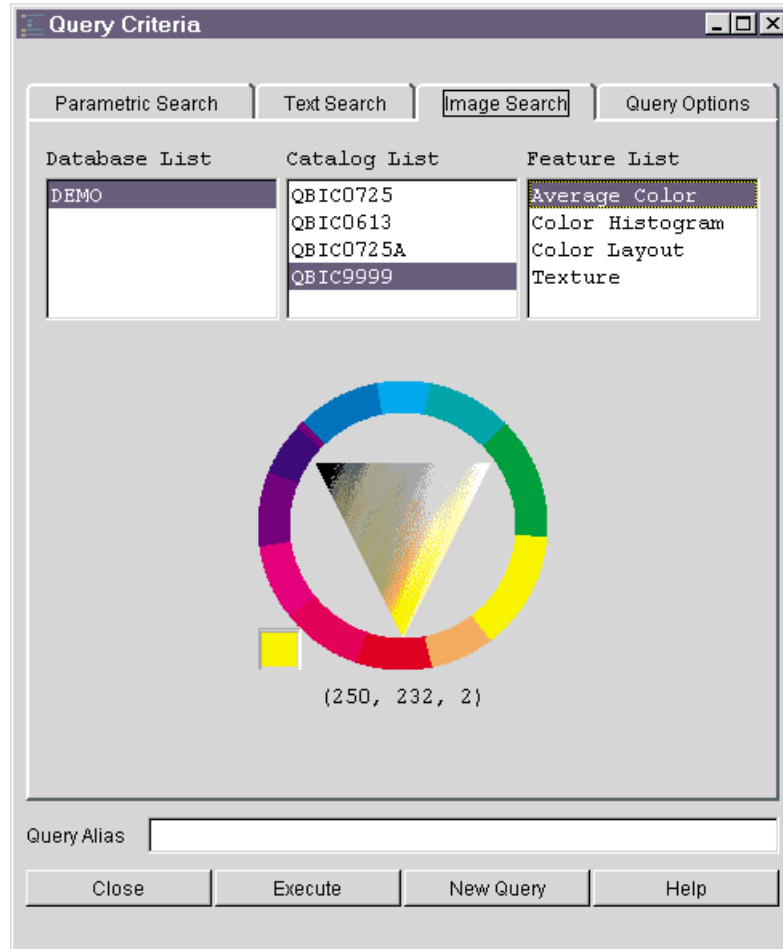


그림 16. 이미지 검색 샘플 클라이언트

이미지 검색 용어 및 개념 이해

이 절에서는 이미지 검색 구성요소(서버, 데이터베이스, 카탈로그 및 이전 Content Manager에 대한 이미지 검색 서버의 관계)에 대해 설명합니다. 또한 이미지의 검색 가능한 비주얼 특성인 기능에 대해서도 설명합니다.

이미지 검색 서버, 데이터베이스 및 카탈로그 이해: 이전 Content Manager는 이미지 검색 서버를 사용하여 이미지를 검색합니다. Content Manager 응용프로그램은 오

브젝트 서버에 이미지 오브젝트를 저장합니다. 이미지 검색 서버는 이미지 정보를 분석하고 색인화합니다. 이미지 검색 서버는 이미지 자체는 저장하지 않습니다.

DKDatastoreQBIC 오브젝트에서 정의한 콘텐츠 서버는 이미지 검색 서버를 나타냅니다. 이미지 검색 결과에는 Content Manager 서버에서 이미지 위치를 설명하는 식별자(항목 ID)가 포함됩니다. 이러한 식별자(ID)를 부분 번호 및 RepType과 같은 다른 결과와 함께 사용하여 이미지를 검색할 수 있습니다.

콘텐츠 서버에서 조회를 수행할 수 있습니다. 그러나 이미지 검색을 위한 콘텐츠 서버는 추가, 갱신, 검색 및 삭제 조작을 지원하지 않습니다. 그림 17에서는 이미지 검색 서버의 예제를 보여줍니다.

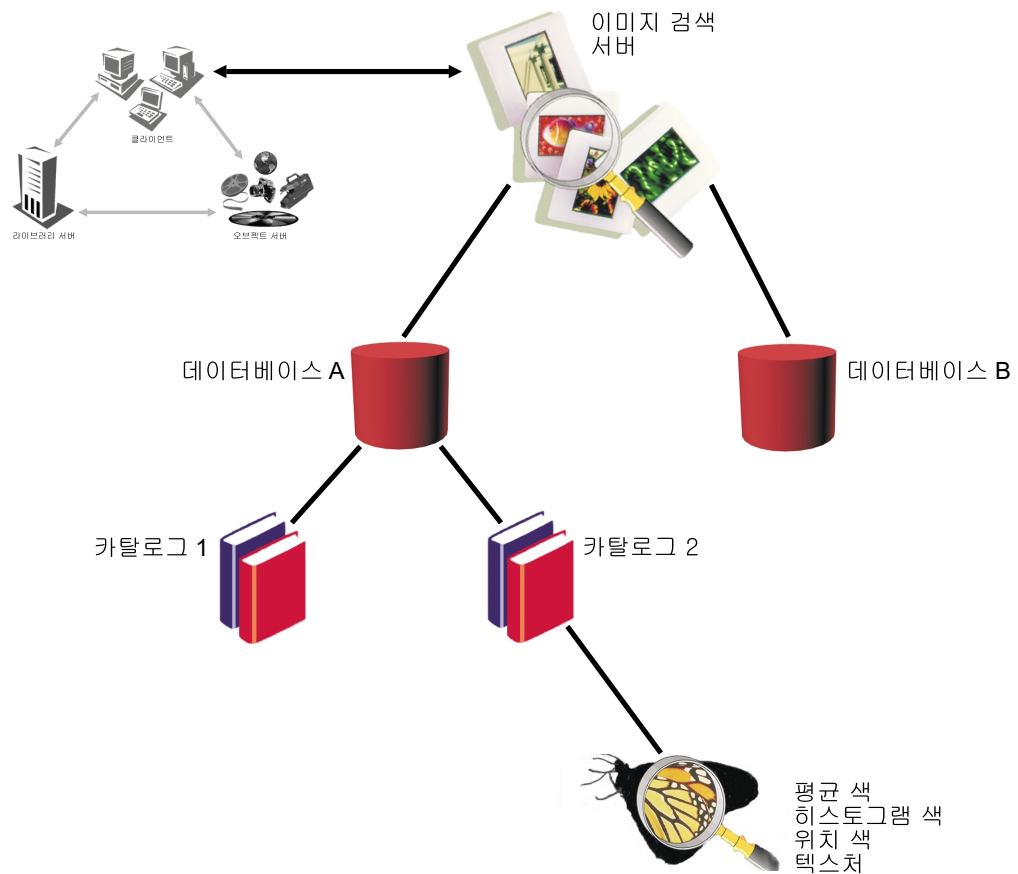


그림 17. 이전 Content Manager 시스템의 이미지 검색 서버

이미지 검색 서버는 하나 이상의 데이터베이스를 포함할 수 있습니다. 각 데이터베이스는 이미지의 비주얼 기능에 대한 정보가 들어 있는 하나 이상의 카탈로그를 포함할 수 있습니다. 이 네 가지 이미지 검색 기능은 다음과 같습니다.

- 평균 색
- 히스토그램 색

- 위치 색(색 배치)
- 텍스처.

이미지 검색 기능 이해: 이 절에서는 네 가지 이미지 검색 기능 및 목적을 정의합니다.

평균 색 평균 색을 사용하여 주요 색을 가진 이미지를 검색할 수 있습니다. 비슷한 주요 색을 가진 이미지는 평균 색과 비슷합니다. 예를 들어, 적색과 노란색이 동일하게 들어 있는 이미지의 평균 색은 오렌지색입니다.

QbColorFeatureClass는 평균 색상의 기능 이름입니다.

히스토그램 색 이미지 색 분포의 백분율을 측정합니다. 히스토그램 분석은 개별적으로 이미지에 있는 여러 가지 색을 측정합니다. 예를 들어, 시골 풍경 이미지는 청색, 녹색 및 회색이 많이 들어 있는 히스토그램 색을 포함합니다.

히스토그램 색을 사용하면 여러 가지 유사한 색을 포함하는 이미지를 검색할 수 있습니다. QbColorHistogramFeatureClass는 히스토그램 색의 기능 이름입니다.

위치 색(색 배치)

위치 색은 이미지의 지정된 영역에 있는 평균 색 값을 픽셀로 측정합니다. 예를 들어, 가운데 밝은 적색 오브젝트가 있는 이미지의 위치 색은 밝은 적색입니다.

QbDrawFeatureClass는 위치 색의 기능 이름입니다.

텍스처

텍스처를 사용하면 특정 패턴을 가진 이미지를 검색할 수 있습니다. 텍스처는 이미지의 조밀도, 명암 및 방향을 측정합니다. 조밀도는 이미지 내의 반복 항목의 크기를 나타냅니다. 명암은 이미지에서 밝기의 변화를 식별합니다. 방향은 이미지에 방향이 두드러지는지 여부를 나타냅니다. 예를 들어, 나무결 이미지는 나무결이 들어 있는 다른 이미지와 유사한 텍스처를 가집니다.

QbTextureFeatureClass는 텍스처의 기능 이름입니다.

이미지 검색 응용프로그램 사용

이미지 검색 클라이언트 응용프로그램은 이미지 조회를 작성하고, 실행한 후 이미지 검색 서버에서 리턴한 정보를 평가합니다. 응용프로그램이 콘텐츠별로 이미지를 검색하려면 먼저 이미지를 색인화하고 콘텐츠 정보를 이미지 검색 데이터베이스에 저장해야 합니다.

제한사항: 오브젝트 서버의 기존 이미지는 색인화할 수 없습니다. 이미지 검색 서버와 클라이언트를 설치한 후 오브젝트 서버에서 작성한 이미지만을 색인화할 수 있습니다. 그림 18에서는 클라이언트 및 이미지 검색의 예제를 보여줍니다.

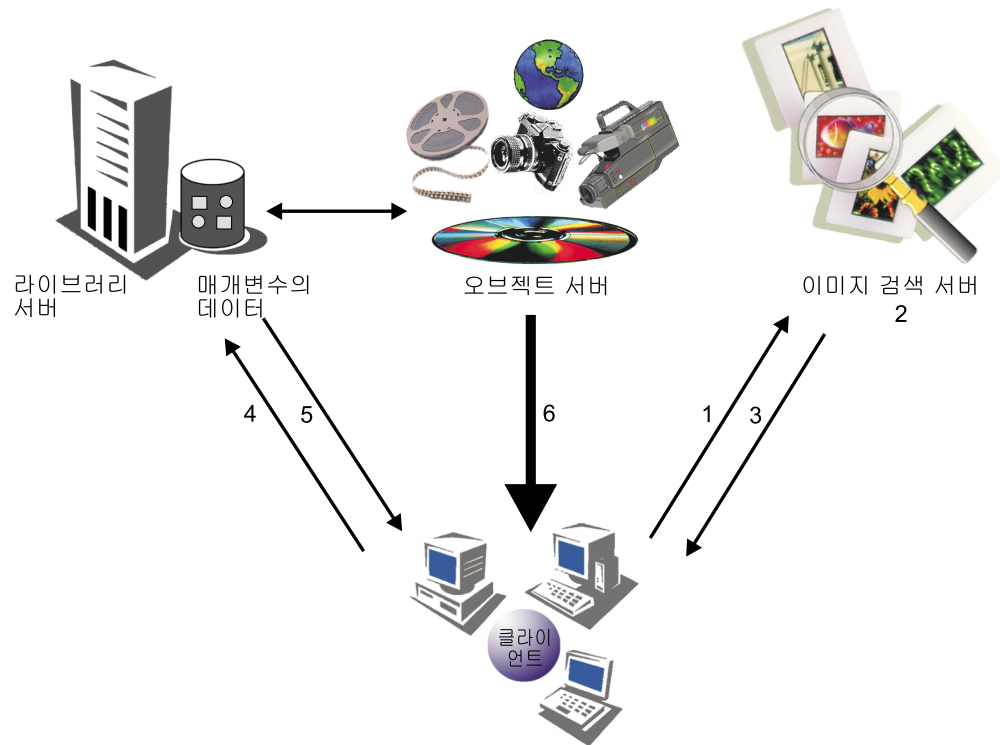


그림 18. 이미지 검색 클라이언트 및 이미지 검색 방법

이미지 검색을 수행하려면 다음을 수행하십시오.

1. 클라이언트는 QBIC 조회 문자열을 빌드하여 이미지 검색 서버로 전송합니다.
2. 이미지 검색 서버는 조회 문자열을 수신하고 카탈로그된 이미지에서 일치하는 항목을 검색합니다.
3. 클라이언트는 일치하는 이미지를 식별자(ID) 목록으로 수신합니다. 일치하는 각 이미지의 식별자(ID)는 다음으로 구성됩니다.
 - 항목 ID
 - 부분 번호
 - RepType
 - 등급
4. 클라이언트는 라이브러리 서버에게 이미지 부분 및 색인 정보를 요청합니다.
5. 라이브러리 서버는 텍스트 설명과 같은 색인 정보를 클라이언트로 리턴합니다. 또한 라이브러리 서버는 오브젝트 서버에게 지정된 이미지 부분을 클라이언트로 전송하도록 요청합니다.

6. 오브젝트 서버는 이미지 부분을 전송하고 클라이언트는 이 이미지 부분을 받았음을 확인하는 메시지를 수신합니다.

조회 작성

조회를 작성할 때 응용프로그램이 이미지 검색 서버로 전달하는 조회 문자열을 구성합니다. 이미지 조회는 검색 기준을 지정하는 문자열입니다. 검색 기준은 다음으로 구성되어 있습니다.

이미지 조회는 검색 기준을 지정하는 문자열입니다. 검색 기준은 다음으로 구성되어 있습니다.

기능 이름	검색에 사용된 기능
기능 값	이러한 기능의 값. 324 페이지의 표 19에 조회 문자열에서 전달될 수 있는 이미지 검색 기능 이름 및 값이 나와 있습니다.
기능 가중치	각 기능에 부여된 상대적 가중치 또는 강조. 기능의 가중치는 이미지 검색 서버가 유사도 점수를 계산하고 조회 결과를 리턴할 때 해당 기능에 부여한 강조입니다. 지정된 가중치가 높을수록 강조도 강해집니다.
최대 결과	조회에서 찾을 이미지의 유형을 정의할 뿐만 아니라 조회에서 리턴할 수 있는 최대 일치 항목 수를 지정할 수 있습니다.

조회 문자열의 양식은 `feature_name value`입니다. 여기서 `feature_name`은 이미지 검색 기능 이름이고 `value`는 기능과 연관된 값입니다. 조회에서 둘 이상의 기능을 사용하면 각 기능에 대해 기능 이름 값 쌍을 지정해야 합니다. "and" 문자열로 각 쌍을 분리합니다.

이미지 검색 조회는 다음과 같은 구문을 갖습니다.

```
feature_name value
feature_name value weight
```

단일 조회 내에서 기능을 반복할 수 없습니다. 한 조회에 여러 기능을 지정할 수 있습니다. 조회에서 여러 기능을 지정할 때 가중치를 하나 이상의 기능에 할당할 수 있습니다. 각 기능에 대한 강조를 포함하는 조회의 양식은 `feature_name value weight`입니다. 여기서 `feature_name`은 이미지 검색 기능 이름이고 `value`는 해당 기능에 연관된 값이며 `weight`는 기능과 할당된 가중치입니다. `weight`는 키워드 `weight`, 등호(=) 및 0.0보다 큰 실제 수가 결합된 것입니다.

또한 조회에서 리턴할 최대 일치 항목 수를 지정할 수 있습니다. 최대 결과 수를 지정하려면 `and max_results`를 해당 조회에 추가하십시오. `max_results`는 키워드 `max`, 등호(=) 및 0보다 큰 정수로 구성되어 있습니다. 324 페이지의 표 19에 기능 이름 및 값에 대한 설명이 나와 있습니다.

표 19. 이미지 검색 조회: 올바른 기능 값

기능 이름	값
QbColorFeatureClass 또는 QbColor	<p>color = < rgbValue , rgbValue , rgbValue ></p> <p>여기서 rgbValue는 0부터 255까지의 정수입니다.</p> <p>file = < fileLocation , " fileName " ></p> <p>여기서 fileLocation은 server 또는 client이고 fileName은 파일이 있는 시스템에 대해 적절한 형식으로 지정된 전체 파일 경로입니다. 예를 들어, 평균 색 및 텍스처 값을 사용하여 검색할 수 있습니다. 텍스처 값은 클라이언트 파일의 이미지에서 제공합니다. 텍스처의 가중치는 평균 색의 두 배입니다.</p> <p>QbColorFeatureClass color=<50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif"> weight=2.0</p>
QbColorHistogramFeatureClass 또는 QbHistogram	<p>histogram = < histList ></p> <p>여기서 histList는 쉼표(,)로 구분된 하나 이상의 histClause로 구성되어 있습니다.</p> <p>histClause는 (histValue, rgbValue, rgbValue, rgbValue)로 지정되고, 여기서 histValue는 1부터 100까지의(백분율 값) 정수이고 rgbValue는 0부터 255까지의 정수입니다.</p> <p>file = < fileLocation , " fileName " ></p> <p>여기서 fileLocation은 server 또는 client이고 fileName은 파일이 있는 시스템에 대해 적절한 형식으로 지정된 전체 파일 경로입니다.</p>

표 19. 이미지 검색 조회: 올바른 기능 값 (계속)

기능 이름	값
QbDrawFeatureClass 또는 QbDraw	<p>description = < " descString " ></p> <p>여기서 descString은 피커 파일에 대해 설명하는 특수 인 코드 문자열입니다. 설명 문자열의 형식은 다음과 같습니다.</p> <ol style="list-style-type: none"> 1. Dw, h는 너비가 w이고 높이가 h인 이미지의 외부 크기를 지정합니다. 2. Rx, y, w, h, r, g, b는 이미지 직사각형의 왼쪽 위 구석에 원점이 있는 경우 좌표(x,y)의 왼쪽 위 구석에 너비 w 및 높이 h의 직사각형이 배치되도록 지정하고, 이 직사각형에는 색 값 r(적색), g(녹색) 및 b(청색)가 있어야 합니다. 3. 콜론 문자(:)가 분리 문자로 사용됩니다. <p>예를 들어, 설명 문자열에서 설명된 색 배치 (QbDrawFeatureClass)를 검색할 수 있습니다.</p> <p>QbDrawFeatureClass description= <"D100,50:R0,0,50,50,255,0,0"</p> <p>file = < fileLocation , " fileName " ></p> <p>여기서 fileLocation은 server 또는 client이고 fileName은 파일이 있는 시스템에 대해 적절한 형식으로 지정된 전체 파일 경로입니다.</p>
QbTextureFeatureClass 또는 QbTexture	<p>file = < fileLocation , " fileName " ></p> <p>여기서 fileLocation은 server 또는 client이고 fileName은 파일이 있는 시스템에 대해 적절한 형식으로 지정된 전체 파일 경로입니다.</p>

조회 예제:

1. 평균 색인 적색을 검색하십시오.

QbColorFeatureClass color=<255,0,0>

2. 세 가지 히스토그램 색(적색 10%, 청색 50% 및 녹색 40%)을 사용하여 검색하십시오.

QbColorHistogramFeatureClass histogram=
<(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>

3. 평균 색 및 텍스처 값을 사용하여 검색하십시오. 텍스처 값은 클라이언트의 파일에 있는 이미지에서 제공합니다. 텍스처의 가중치는 평균 색의 두 배입니다.

QbColorFeatureClass color=
<50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif">
weight=2.0

4. 설명이 되어 있는 색 배치를 검색하십시오.

QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">

5. 평균 색인 적색을 검색하고 리턴되는 일치 항목 수를 5개로 제한하십시오.

```
QbColorFeatureClass color=<255,0,0> and max=5
```

조회 실행 및 검색 결과 평가

응용프로그램은 이미지 검색 API를 사용하여 조회를 발행하고 검색 결과를 평가합니다. 이미지 검색 데이터베이스의 정보가 이미지 검색 기준과 일치하면 일치하는 이미지 식별자(ID) 또는 이미지가 리턴됩니다. 이 식별자(ID)는 Content Manager 오브젝트 내의 이미지 부분에 해당하는 동적 데이터 오브젝트(DDO)입니다.

QBIC에서 연결 설정

이미지 검색은 콘텐츠 서버에 연결 및 연결 해제하는 함수를 제공합니다. 다음 예제에서는 사용자 ID QBICUSER 및 암호 PASSWORD를 사용하여 QBICSRV라는 이미지 검색 서버에 연결하는 방법을 보여줍니다.

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // Process as appropriate
dsQBIC.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TConnectQBIC.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // do some work
dsQBIC.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TConnectQBIC.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

이미지 검색 연결을 사용하면 응용프로그램이 이미지 검색 서버에 연결할 수 있습니다.

연결되면 listDatabases와 같은 이미지 검색 카탈로그와 관계없는 함수를 제외한 이미지 검색 서버에 액세스하는 함수를 사용할 수 있습니다. openCatalog 함수는 처리 시 카탈로그를 열어야 합니다. closeCatalog 함수는 처리가 완료된 후 호출됩니다. 다음 예제에서는 카탈로그에 연결하여 카탈로그를 연 다음 카탈로그를 닫고, 연결 해제하는 방법을 보여줍니다.

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();
```

이미지 검색 서버 나열

이미지 검색 서버는 연결할 수 있는 이미지 검색 서버를 나열하는 함수를 제공합니다. 다음 예제에서는 `DKSequentialCollection` 오브젝트에서 `DKServerInfoQBIC` 오브젝트를 포함하는 서버 목록을 검색하는 방법을 보여줍니다. `DKServerInfoQBIC` 오브젝트를 가져온 후 서버 이름, 호스트 이름 및 포트 번호를 검색할 수 있습니다.

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
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogQBIC.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogQBIC.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

이미지 검색 데이터베이스, 카탈로그 및 기능 나열

DKDatastoreQBIC는 이미지 검색 서버의 모든 이미지 검색 데이터베이스, 카탈로그 및 기능을 나열하는 함수를 제공합니다. 이 목록은 DKIndexQBIC 오브젝트를 포함하는 DKSequentialCollection 오브젝트에 리턴됩니다. DKIndexQBIC 오브젝트를 가져온 후 데이터베이스, 카탈로그 및 기능 이름을 검색할 수 있습니다. 다음 예제에서는 데이터베이스, 카탈로그 및 기능의 목록을 검색하는 방법을 보여줍니다.

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();
.....
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogQBIC.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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++(계속)

```
        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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogQBIC.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

DDO를 사용하여 이미지 검색 정보 표시

DKDatastoreQBIC와 연관된 DDO는 이미지 검색 결과를 나타내는 데 필요한 특정 정보를 포함합니다. 이미지 조회로 생성된 DDO는 항목 내부의 이미지 부분에 해당하며 다음의 표준 속성 세트를 포함합니다.

DKDLITEMID

이 이미지 부분이 속하는 항목의 항목 ID입니다. 콘텐츠 서버에서 전체 항목을 검색하려면 항목 ID를 사용하십시오.

DKPARTNO

이 이미지 부분의 정수 부분 번호입니다. 콘텐츠 서버에서 이 부분을 검색하려면 항목 ID와 함께 이 번호를 사용하십시오.

DKREPTYPE

표현 유형(RepType)에 대한 문자열입니다. 기본값은 FRN\$NULL입니다. 이 속성은 예약되어 있습니다.

DKRANK

이미지 조회의 결과 세트와 이 부분의 관계를 나타내는 정수 등급입니다. 등급의 범위는 0부터 100까지입니다. 등급이 높을수록 좀더 정확하게 일치함을 의미합니다.

이미지 검색 DDO의 PID는 다음 정보를 포함합니다.

콘텐츠 서버 유형
QBIC.

컨텐츠 서버 이름

이 이름은 컨텐츠 서버 연결에 사용됩니다.

ID 결과 세트에서 DDO의 0 기준 일련 번호.

일반적으로 속성값은 항상 오브젝트입니다.

이미지 조회에 대한 작업

이 절에서는 이미지 조회의 실행 및 평가 방법에 대해 설명합니다.

이미지 조회 실행

DKDatastoreQBIC에서 dkQuery의 인스턴스를 사용하면 조회 오브젝트를 작성하여 조회를 실행한 후 결과를 얻을 수 있습니다. 다음 예제에서는 이미지 조회 오브젝트를 작성하여 이를 실행하는 방법을 보여줍니다. 조회를 실행한 후 결과는 DKResults 컬렉션에 리턴됩니다.

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();
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(SampleIQryQBIC.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSampleIQryQBIC.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

컨텐츠 서버에서 이미지 조회 실행

다른 방법으로 DKDatastoreQBIC의 실행 함수를 사용하여 조회를 실행할 수 있습니다. 결과는 dkResultSetCursor 오브젝트에 리턴됩니다. 다음 예제에서는 컨텐츠 서버에서 이미지 조회를 실행하는 방법을 보여줍니다. 결과는 dkResultSetCursor 오브젝트에 리턴됩니다.

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");
// ----- 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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteQBIC.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteQBIC.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

컨텐츠 서버에서 이미지 조회 평가

DKDatastoreQBIC는 조회를 평가하는 함수도 제공합니다. 다음 예제에서는 컨텐츠 서버에서 이미지 조회를 평가하는 방법을 보여줍니다. 결과는 DKResults 콜렉션에 리턴됩니다.

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();
```

이미지 검색 엔진 사용

이미지 검색 서버를 사용하여 다음 기능(평균 색, 색 백분율, 색 배치 및 텍스트) 중 하나를 기본으로 하는 조회를 지정할 수 있습니다. 또한 하나의 조회에 여러 기능을 지정할 수 있습니다. 조회 결과에는 항목 ID, 부분 번호, 표현 유형 및 등급 정보가 포함됩니다. 이 정보를 사용하여 이미지 콘텐츠를 검색하는 XDO를 작성할 수 있습니다.

이미지 검색 시 색인화될 데이터 로드

이미지 검색 서버에 의해 색인화될 데이터를 Content Manager 서버에 로드하려면 Content Manager 색인 클래스, 이미지 검색 데이터베이스 및 이미지 검색 카탈로그를 작성해야 합니다. 데이터베이스는 이미지 검색 카탈로그의 컬렉션입니다. 카탈로그는 이미지의 비주얼 기능에 대한 데이터를 보유합니다.

색인화하려면 이미지 검색 기능을 카탈로그에 추가해야 합니다. 지원되는 모든 기능을 카탈로그에 추가해야 합니다.

이미지 검색 데이터베이스와 카탈로그를 작성할 때 이미지 검색 서버가 실행 중이어야 합니다. 사용자 환경이 제대로 설정되었는지 확인하십시오.

데이터가 Content Manager에 로드된 후 이미지 대기열에 이미지를 넣을 수 있습니다. 시스템 관리 프로그램에서 이미지 대기열 처리를 선택하십시오. 색인화가 완료된 후 이미지 검색을 실행할 수 있습니다.

검색 엔진을 사용하여 기존 XDO 색인화

지정된 검색 엔진을 사용하여 기존 XDO를 색인화할 수 있습니다. 다음 예제에서는 DKBlobDL 클래스의 setToBeIndexed 함수를 호출합니다.

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++(계속)

```
//--- 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;
}
```

결합된 조회 사용

결합된 조회를 사용하면 범위 존재 여부에 관계없이 매개변수식 조회 및 텍스트 조회로 구성된 결합된 조회를 실행할 수 있습니다. 범위는 이전의 매개변수식 조회 또는 텍스트 조회에서 형성된 DKResults 오브젝트입니다. 이 결과는 각 조회의 범위와 결과 사이의 교집합입니다. 따라서 조회를 구성하고 범위를 포함시킬 때 주의하지 않으면 개별 조회 결과가 서로 교차하지 않고 결합된 조회의 결과가 비어 있게 됩니다.

최소한 하나의 매개변수식 조회 및 하나의 텍스트 조회가 있으면 결과 DDO는 문서에 속하는 일치하는 부분 중 가장 높은 등급을 나타내는 속성 DKRANK를 포함하게 됩니다.

제한사항: 결합된 조회의 각 조회에 대해 검색 엔진에 다른 연결을 사용해야 합니다. 같은 연결을 통해 여러 조회를 경로지정할 수 없습니다.

결합된 매개변수식 조회 및 텍스트 조회

범위가 없는 하나의 매개변수식 조회 및 하나의 텍스트 조회로 구성된 결합된 조회를 실행하려면 결합된 조회 오브젝트를 작성하고 결합된 조회에서 실행할 입력 매개변수로 두 개의 조회를 전달해야 합니다. 예를 들면, 다음과 같습니다.

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
    ...
}
```

마지막 if문은 DKResults 오브젝트가 널(null)이 아닌지를 확인하는 데 필요합니다.

범위 사용

범위로 사용하려는 DKResults 오브젝트가 있는 경우 이를 추가 조회 매개변수 중 하나로 전달하십시오. 다음 예제에서는 DKResults 오브젝트를 사용하여 결합된 조회의 범위로 작동시키는 방법을 보여줍니다.

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 if DKNVPair 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);
...
```

또한 결합된 조회의 결과는 다른 결합된 조회의 범위로 사용할 수도 있으며, 때때로 결과를 조회할 수 있습니다.

등급

결합된 조회에 최소한 하나의 텍스트 조회가 포함될 경우, 결과 DDO는 DKRANK 속성을 포함합니다. 이 속성은 저장되지 않지만 매번 텍스트 검색 엔진에 의해 계산됩니다. 등급값은 텍스트 조회 조건에 부합되는 문서에 있는 부분 중 가장 높은 등급에 해당합니다.

팁

여러 개의 매개변수식 조회와 범위를 포함하는 경우, 하나의 전체적인 조회를 실행하는 것이 더 효율적입니다. 텍스트 조회인 경우에도 마찬가지입니다.

조회 옵션 "MAX_RESULTS=nn"은 리턴되는 결과의 수를 제한합니다. 대개 결과는 등급에 따라 내림차순으로 저장되므로 이 옵션은 텍스트 조회에 적합합니다. 예를 들어, 이 옵션을 10으로 설정할 경우, 호출자가 일치하는 결과 중 가장 높은 10개만 원한다는 것을 의미합니다.

매개변수식 조회에서는 조회 옵션 "MAX_RESULTS=nn"의 의미가 다릅니다. 등급에 대한 개념이 없으므로 호출자는 처음 10개의 결과를 얻습니다. 이 결과는 텍스트 조회 결과에서 발생한 부분과 교차됩니다. 따라서 매개변수식 조회와 텍스트 조회를 결합할 경우, 매개변수식 조회에서는 조회 옵션 "MAX_RESULTS=nn"을 지정하지 않는 것이 좋습니다.

이전 Content Manager 워크플로우 및 작업함 기능 이해

이 절에서는 이전 Content Manager 워크플로우 및 작업함 기능에 대해 설명합니다.

이전 Content Manager 워크플로우 서비스 이해

작업함은 처리할 문서와 폴더를 보유하는 컨테이너입니다. 워크플로우는 특정 비즈니스 프로세스를 나타내는 순서화된 작업함 세트입니다. 폴더와 문서는 워크플로우 내의 작업함 사이를 이동하고, 완료할 때까지 응용프로그램이 간단한 비즈니스 모델을 작성하고 프로세스를 통해 작업을 경로지정할 수 있게 합니다.

Content Manager에서 워크플로우 모델은 다음 규칙을 따릅니다.

- 작업함이 워크플로우에 있을 필요는 없습니다.
- 작업함은 하나 이상의 워크플로우에 있을 수 있습니다.
- 작업함은 두 번 이상 같은 워크플로우에 있을 수 있습니다.
- 문서 또는 폴더를 한번에 하나의 워크플로우에만 저장할 수 있습니다.
- 문서나 폴더를 워크플로우에 있지 않은 작업함에 저장할 수 있습니다.

Enterprise Information Portal API는 Content Manager 워크플로우에 대해 작업함 클래스를 제공합니다.

DKWorkflowServiceDL 클래스는 Content Manager의 워크플로우 서비스를 나타냅니다. 이 클래스는 워크플로우의 문서나 폴더를 시작, 변경, 제거, 경로지정 및 완료하는 기능을 제공합니다. 또한 DKWorkflowServiceDL 클래스를 사용하여 작업함 및 워크플로우에 대한 정보를 검색할 수 있습니다.

DKWorkflowDL 및 DKWorkBasketDL 클래스는 각각 워크플로우 항목 및 작업함 항목의 객체 지향 표현입니다.

연결 설정

워크플로우 서비스를 사용하려면 먼저 Content Manager 서버에 대한 연결을 설정해야 합니다. 콘텐츠 서버는 connection 및 disconnection 함수를 제공합니다.

다음 예제에서는 사용자 ID FRNADMIN 및 암호 PASSWORD를 사용하여 LIBSRVR이라는 Content Manager 서버에 연결하는 방법을 보여줍니다.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...           // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkFlowWFS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...           // do some work
dsDL.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkFlowWFS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

워크플로우 작성

워크플로우를 작성하려면 DKWorkflowServiceDL을 사용하십시오. 이를 수행하려면 일반적으로 다음의 6단계를 완료해야 합니다.

1. DKWorkflowDL의 인스턴스를 작성하십시오.
2. 워크플로우 이름("GOLF")을 설정하십시오.

3. 작업함 순서("NULL")를 설정하여 이 워크플로우가 작업함을 포함하지 않음을 표시하십시오.
4. 사용 권한("All Privileges")을 설정하십시오.
5. 배치(DK_WF_SAVE_HISTORY)를 설정하십시오.
6. add 함수인 add ()를 호출하십시오.

다음 예제에서는 위의 6단계를 따라 워크플로우를 작성합니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TCreateDelWorkflow.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TCreateDelWorkflowWFS.cpp)은 Cmbroot/Samples/cpp/dl 디렉토리에 있습니다.

중요사항: 일반 사용자(DK_SS_NORMAL)로 콘텐츠 서버에 연결할 경우, 연결 후 정의된 워크플로우를 가져올 수 없습니다. 따라서 이 샘플은 DK_SS_CONFIG를 사용합니다.

워크플로우 나열

DKWorkflowServiceDL은 다음 예제에서 나와 있는 것처럼 시스템의 워크플로우를 나열하는 함수를 제공합니다. 이 목록은 DKWorkflowDL 오브젝트의 순차 컬렉션에 리턴됩니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkFlowWFS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkFlowWFS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

Content Manager 작업함 작성

작업함을 작성하려면 DKWorkflowServiceDL을 사용하십시오. 이를 수행하려면 일반적으로 다음 단계를 완료해야 합니다.

1. DKWorkBasketDL의 인스턴스를 작성하십시오.

2. 작업함 이름(Hot Items)을 설정하십시오.
3. 사용 권한(All Privileges)을 설정하십시오.
4. add 함수를 호출합니다.

다음 예제에서는 위의 단계에 따라 워크플로우를 작성합니다. 일반 사용자(DK_SS_NORMAL)로 콘텐츠 서버에 연결할 경우, 연결 후 정의된 작업함을 가져올 수 없습니다. 따라서 이 샘플은 DK_SS_CONFIG를 사용합니다.

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();
```

이 예제가 수행된 완전함 샘플 응용프로그램(TCreateDelWorkBasket.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전함 샘플 응용프로그램 (TCreateDelWorkBasket.java)은 CMBROOT\Samples\cpp\d1 디렉토리에 있습니다.

작업함 나열

DKWorkflowServiceDL은 다음 예제에 나와 있는 것처럼 시스템의 작업함을 나열하는 함수를 제공합니다. 이 목록은 DKWorkBasketDL 오브젝트의 순차 컬렉션에 리턴됩니다.

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 = (DKWorkBasketDL)pIter.next();
        pwb1->retrieve();
        ... // do some work
    }
}
dsDL.disconnect();
dsDL.destroy();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkBasketWFS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1=(DKSequentialCollection *)wfDL.listWorkBaskets();
if (wfList1 != NULL)
{
    dkIterator * pIter1 = wfList1->createIterator();
    DKWorkBasketDL * pwb1;
    while (pIter1->more())
    {
        pwb1 = (DKWorkBasketDL *)((void*)(*pIter1->next()));
        pwb1->retrieve();
        ... // do some work
        delete pwb1;
    }
}
dsDL.disconnect();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListWorkBasketWFS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

이전 Content Manager 워크플로우의 항목 나열

DKWorkflowServiceDL은 다음 예제에 나와 있는 것처럼 워크플로우에서 해당 항목의 항목 ID를 나열하는 함수를 제공합니다. 이 목록은 DKString 오브젝트의 순차 컬렉션에 리턴됩니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListItemsWFS.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListItemsWFS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

이전 Content Manager 워크플로우 실행

DKWorkflowServiceDL은 워크플로우를 실행하는 함수를 제공합니다. 다음 예제에서는 워크플로우에서 항목을 시작하는 방법, 작업함으로 항목을 경로지정하는 방법 및 워크플로우에서 항목을 완료하는 방법에 대해 설명합니다. 이 샘플을 사용하려면 다음과 같이 샘플을 수정해야 합니다.

- EP8L80R9MHH##QES 대신 올바른 항목 ID를 사용하십시오.
- HI7MOPALUPFQ1U47 대신 올바른 워크플로우 ID를 사용하십시오.
- E3PP1UZ0ZUFQ1U3M 대신 올바른 작업함 ID를 사용하십시오.

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("E3PP1UZ0ZUFQ1U3M");
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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TProcessWFS.javav)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemID = DKString("EP8L80R9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZ0ZUFQ1U3M");
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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TProcessWFS.cpp)은 Cmbroot/Samples/cpp/d1 디렉토리에 있습니다.

OnDemand에 대한 작업

Enterprise Information Portal에서는 Content Manager OnDemand 서버의 콘텐츠에 액세스하는 데 필요한 커넥터 및 관련 클래스를 제공합니다. OnDemand 클래스 및 API를 사용하여 다음 작업을 수행할 수 있습니다.

- OnDemand Server에 연결 및 연결 해제
- 응용프로그램 그룹 및 응용프로그램 그룹 필드 나열
- OnDemand 폴더 나열
- 응용프로그램 그룹 조회
- 폴더 또는 응용프로그램 그룹 인터페이스를 사용하여 문서 검색
- 연합 검색에서 OnDemand 폴더를 원래의 엔티티로 처리
- 응용프로그램 그룹 또는 폴더 모드에서 동기 및 비동기 검색
- 전체 OnDemand 문서 또는 문서 세그먼트 검색
- 지정된 OnDemand 문서의 논리 보기 데이터 검색
- 지정된 OnDemand 문서의 자원 그룹 검색
- 지정된 OnDemand 문서의 주석 데이터 검색
- 주석 작성 및 수정

제한사항: OnDemand는 텍스트 검색 엔진 및 QBIC 검색 또는 결합된 조회를 지원하지 않습니다.

OnDemand Server 및 문서 표현

Enterprise Information Portal 응용프로그램에서는 DKDatastoreOD를 사용하여 Content Manager OnDemand 콘텐츠 서버를 나타내고 DKDDO를 사용하여 OnDemand 문서를 DDO로 나타냅니다. OnDemand DDO는 다음 정보를 포함합니다.

- 문서 속성 이름 및 값
- 문서 데이터 및 주식(DKParts로 표현)
- 문서의 논리 보기 컬렉션
- 자원 그룹 데이터

OnDemand 문서의 속성은 DKDDO에 등록 정보로 저장되고 OnDemand 문서의 세그먼트 및 메모는 DKParts로 저장됩니다.

다른 모든 문서 데이터(자원 그룹 및 보기, 모두 고정 및 논리)는 OnDemand DDO에 특수 등록 정보로 저장되며 다음 등록 정보 이름은 OnDemand에 대해 예약되어 있습니다.

DKViews

논리 보기 컬렉션

DKFixedView

고정 보기 정보 포함

DKResource

자원 그룹 데이터 포함

주:

1. Enterprise Information Portal 시스템 관리자는 초기화 매개변수 페이지의 추가 매개변수 필드에 문자열을 지정하여 OnDemand 커넥터를 제대로 정의해야 합니다. 문자열은 ENTITY_TYPE=TEMPLATES;; 형식을 갖습니다. 두 개의 세미콜론이 포함되어야 합니다.
2. Enterprise Information Portal 버전 8.2에서 DKViews, DKFixedView 및 DKResource는 각각 DKViewDataOD, DKFixedViewDataOD 및 DKResourceGrpOD를 대체합니다.

OnDemand Server에 연결 및 연결 해제

OnDemand 콘텐츠 서버로 로그인하려면 서버 이름(예: ODServer.mycompany.com), 사용자 ID 및 암호를 connect 메소드를 통해 전달하십시오.

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 Server에서 로그아웃하려면 disconnect 메소드를 사용하십시오.

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 정보 나열

OnDemand Server의 응용프로그램 그룹 및 폴더를 나열할 수 있습니다.

응용프로그램 그룹 나열

DKDatastoreOD의 listEntities() 메소드를 사용하여 OnDemand의 응용프로그램 그룹을 나열할 수 있습니다. 다음 예제에서는 이 메소드의 사용 방법을 보여줍니다.

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;
}
...
```

다음 예제에서는 각 응용프로그램 그룹의 속성 정보를 얻는 방법을 보여줍니다.

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 폴더 나열

OnDemand 컨텐츠 서버에서 폴더 목록을 가져오려면 listSearchTemplates() 함수를 사용하십시오.

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 문서 검색

OnDemand Server에서 문서를 검색할 수 있습니다. 또한 문서를 해당 부분 및 속성과 함께 표시할 수도 있습니다.

특정 문서 검색

다음 예제에서는 OnDemand 콘텐츠 서버의 응용프로그램 그룹(OnDemand Publications)을 검색합니다.

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;
```

다음 예제에서는 응용프로그램 그룹(OnDemand Publications)을 검색한 후 리턴된 문서를 검색합니다.

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;
}
```

문서와 해당 부분 및 속성 표시

다음 예제에서는 조회에서 찾은 문서를 해당 부분 및 속성과 함께 표시합니다.

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(계속)

```
        if (pD0.protocol() == DK_XD0)
        {
            System.out.println("      dkXD0 object " + i + " in collection");
            pXD0 = (dkXD0)pD0;
            DKPidXD0 pid2 = pXD0.getPidObject();
            System.out.println("          XD0 pid string: " +
                               pid2.pidString());
            //----- Retrieve and open instance handler for an XD0
            pXD0.retrieve();
            // pXD0.open();
        }
    }
}
else if (a != null)
{
    System.out.println("      Attribute Value: " + a.toString());
    if (strDataName.equals("DKResource") ||
        strDataName.equals("DKFixedView") ||
        strDataName.equals("DKLargeObject"))
    {
        pD0 = (dkDataObjectBase)a;

        if (pD0.protocol() == DK_XD0)
        {
            System.out.println("          dkXD0 object ");
            pXD0 = (dkXD0)pD0;
            DKPidXD0 pid2 = pXD0.getPidObject();
            System.out.println("          XD0 pid string: " +
                               pid2.pidString());
            // Retrieve and open instance handler for an XD0
            pXD0.retrieve();
            // pXD0.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;
    }
}
```

전체 응용프로그램에 대해서는 CMBROOT\samples\cpp\od 디렉토리의 TRetrieveOD.cpp를 참조하십시오.

OnDemand 폴더 모드 사용

OnDemand 폴더 모드를 사용하려면 다음 문자열을 연결 문자열 또는 구성 문자열의 일부로 OnDemand 커넥터에 전달해야 합니다.

ENTITY_TYPE=TEMPLATES

샘플 구성 문자열은 다음과 같습니다.

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

샘플 연결 문자열은 다음과 같습니다.

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");
```

비동기 검색

OnDemand 커넥터는 연합 및 직접 비동기 검색을 모두 지원합니다. 비동기 검색은 기본 스레드에 영향을 미치지 않으므로 언제든지 검색을 취소할 수 있습니다. 검색을 종료하려면 검색 템플릿 보기 프로그램에서 검색 정지 단추를 누르십시오. 검색 템플릿 보기 프로그램의 최대 히트 등록 정보는 리턴되는 최대 결과 수를 제한합니다.

OnDemand 커넥터는 AIX 클라이언트에서 응용프로그램 그룹 모드의 동기 및 비동기 검색도 지원합니다.

WHERE userid LIKE '%'와 같은 검색 기준을 사용하는 경우, 클라이언트로 리턴되는 결과 문서 수가 많아 클라이언트 기계에서 사용 가능한 모든 메모리를 소비할 수 있습니다. executeWithCallback() 메소드를 사용하여 비동기 검색을 발행하면 리턴되는 최대 문서 수를 설정하고 언제든지 검색을 취소할 수 있습니다.

결과 세트가 너무 큰 경우에는 기본 JVM(Java Virtual Machine) 스택 크기도 늘려야 합니다. 각 Java 스레드의 기본 스택 크기는 400k이며, 이는 스택이 오버플로우되기 전에 3920개의 항목 리턴을 허용합니다. JVM 스택 크기를 800k로 늘리면 용량이 7840 항목으로 두 배가 늘어납니다. 필요한 경우 JVM 스택 크기를 더 늘릴 수 있습니다.

JVM 스택 크기를 늘리려면 Java 명령행 옵션 -oss 다음에 nnnK 또는 nnM을 사용하십시오. 여기서 K는 킬로바이트, M은 메가바이트를 나타냅니다.

비동기 검색 사용의 예제에 대해서는 TRetrieveWithCallbackOD, TRetrieveFolderWithCallbackOD 및 TCallbackOD 샘플 프로그램을 참조하십시오.

검색 템플릿으로서 OnDemand 폴더

세 개의 EIP 비주얼 JavaBeans(CMBSearchTemplateList, CMBSearchTemplateViewer 및 CMBSearchResultsView)는 EIP 검색 템플릿을 사용합니다. CMBConnection dsType을 Fed로 설정하여 연합 검색에 이러한 Bean을 사용할 수 있습니다.

OnDemand Server에서의 직접 검색에도 이러한 Bean을 사용할 수 있습니다. 로그인 하기 전에 다음 등록 정보를 설정하십시오.

```
| connection.setDsType("OD");  
| connection.setServerName(<odserver>);  
| connection.setConnectString("ENTITY_TYPE=TEMPLATES");
```

원래의 엔티티로서 OnDemand 폴더

OnDemand 커넥터는 연결 문자열 "ODENTITY=TEMPLATES"를 지정하여 OnDemand 폴더를 원래의 엔티티로 맵핑할 수도 있습니다. OnDemand 폴더를 엔티티로 사용하면 연합 검색에 유용합니다. 이 경우 폴더 정의 작업이 OnDemand의 원래의 기본 엔티티인 OnDemand 응용프로그램 그룹에 대한 작업보다 훨씬 쉽습니다.

연합 검색의 경우 Enterprise Information Portal 관리에 서버 정의를 지정하십시오. 연합 엔티티를 OnDemand Server의 폴더에 정의 및 맵핑할 수 있습니다.

주석 작성 및 수정

CMBDocumentViewer Bean에서 시작한 OnDemand 보기 프로그램 사용 시 CMBDataManagement Bean 및 연관된 CMBAnnotation 클래스를 사용하여 OnDemand 문서에 대해 주석을 작성, 수정 및 삭제할 수 있습니다.

추적

OnDemand 커넥터와 함께 이벤트를 추적할 수 있습니다. 커넥터 java API 추적을 사용하려면 추적 INI 파일(Java용 `cmbodtrace.ini` 또는 C++용 `cmbodCtrace.ini`)을 C 드라이브의 루트(C:\) 또는 CMBROOT 변수에 지정된 디렉토리에 두십시오. AIX의 경우에는 이 파일을 `/usr/lpp/cmb/cmgmt`에 두십시오. Solaris의 경우, 이 파일을 `/opt/IBMcmb/cmgmt`에 두십시오.

추적 파일에 대한 기본 출력 디렉토리는 C:\Ctrace입니다. 다른 곳에서 추적 정보를 작성하려면 추적 INI 파일을 편집하십시오. AIX의 경우에는 파일 이름이 모두 소문자여야 한다는 점에 유의하십시오.

추적 파일에 지정된 경로 이름이 올바르며 CMBODTRACEDIR을 포함하는 행이 # 부호로 시작되지 않는지를 확인하십시오. 다음은 추적 INI 파일의 샘플입니다.

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/cmgmt/ctrace
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

OS/390용 Content Manager ImagePlus에 대한 작업

Enterprise Information Portal API는 OS/390용 Content Manager ImagePlus 컨텐츠 서버에 대한 작업 시 다음 기능을 지원합니다.

- 하나 이상의 ImagePlus 서버에 연결 및 연결 해제
- 카테고리 검색
- 속성 필드 검색
- 폴더 검색
- 문서 검색

응용프로그램에서 DKDatastoreIP를 사용하여 OS/390용 ImagePlus 컨텐츠 서버를 나 타냅니다.

제한사항: OS/390용 ImagePlus는 다음을 지원하지 않습니다.

- 텍스트 검색 엔진 및 QBIC 검색
- 결합된 조회

- 작업함 및 워크플로우

엔티티 및 속성 나열

OS/390용 ImagePlus 콘텐츠 서버에 대해 콘텐츠를 서버를 DKDatastoreIP 오브젝트로 작성하고 연결한 후 콘텐츠 서버의 엔티티 및 속성을 점검할 수 있습니다. 다음 예제에서는 OS/390용 ImagePlus 콘텐츠 서버에 대해 모든 엔티티를 나열합니다.

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
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogIP.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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
{
    ...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogIP.cpp)은 Cmbroot/Samples/cpp/ip 디렉토리에 있습니다.

다음 예제에서는 DKEntityDefIP의 getAttr 및 listAttrNames 함수를 사용하여 각 엔티티와 연관된 모든 속성을 나열합니다.

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<tmpcoun; 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;
```

다음 예제에서는 DKDatastoreIP의 listEntityAttrs 메소드를 사용하여 각 엔티티와 연관된 속성을 나열하는 다른 방법을 보여줍니다.

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;
```

OS/390용 ImagePlus 조회 구문규칙

다음 예제에서는 OS/390용 ImagePlus의 조회 구문규칙을 보여줍니다.

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};])]
```

조회는 다음 매개변수를 사용합니다.

search_expression

각 검색 표현식은 하나 이상의 검색 기준으로 구성되어 있습니다. 검색 기준 사이에 논리 연산자 AND만을 사용할 수 있습니다.

검색 기준의 양식은 다음과 같습니다.

{attr_name | mapped_attr_name} operator literal

여기서

attr_name

검색 기준이 될 엔티티 속성의 이름.

mapped_attr_name

검색 기준이 될 속성을 사용하여 맵핑된 속성 이름.

operator

모든 속성은 등식을 지원합니다(==). DATE 유형 속성에는 다음 추가 연산자를 사용할 수 있습니다.

> 초과

< 미만

>= 이상

<= 이하

literal

리터럴입니다. 숫자 속성에는 다음과 같이 따옴표(")를 사용하지 마십시오.

FolderType == 9

날짜, 시간 및 시간 소인 속성에는 따옴표 또는 어포스트로피(')를 사용할 필요는 없지만 다음과 같이 사용할 수도 있습니다.

```
ReceiveDate == 1999-03-08  
ReceiveDate == '1999-03-08'
```

문자열 속성에는 따옴표 또는 어포스트로피(')를 사용할 필요는 없지만 사용할 수도 있습니다. 문자열에 어포스트로피(')가 있으면 예제 Folder'1의 값처럼 두 개의 어포스트로피를 사용하여 문자열을 지정해야 합니다.

```
FolderId == 'Folder''1'
```

entity_name

검색할 엔티티의 이름.

mapped_entity_name

검색할 엔티티에 매핑된 엔티티 이름.

maximum_results

리턴할 최대 결과 수.

옵션 키워드는 다음과 같습니다.

CONTENT 결과에 리턴될 정보의 수를 제어합니다.

YES(기본값)

문서 또는 폴더의 PID, 속성 및 값을 설정합니다. 문서에 부분이 있는 경우, XDO PID가 설정됩니다. 폴더에 문서가 있는 경우, 문서 PID가 설정됩니다.

NO 문서 또는 폴더 PID만 설정합니다.

ATTRONLY

문서 또는 폴더의 PID, 속성 및 값만 설정합니다.

PENDING 부분이 없는 보류 중 문서의 포함 여부를 제어합니다. 이 옵션은 ENTITY가 DOCUMENT로 설정되거나 DOCUMENT에 매핑된 엔티티로 설정될 경우에만 적용됩니다.

YES 결과에 보류 중인 문서가 포함됩니다.

NO(기본값)

결과에 보류 중인 문서가 포함되지 않습니다.

AS/400용 Content Manager에 대한 작업

AS/400용 Content Manager에 제공된 API 클래스(AS/400용 VisualInfo)는 Content Manager에 제공된 API 클래스와 유사합니다.

제한사항: AS/400용 Content Manager는 다음을 지원하지 않습니다.

- 텍스트 검색 엔진 및 QBIC 검색
- 결합된 조회
- 작업함 및 워크플로우

엔티티(색인 클래스) 및 속성 나열

AS/400용 Content Manager 컨텐츠 서버를 DKDatastoreV4로 나타냅니다. 컨텐츠 서버를 작성하여 연결한 후 AS/400용 Content Manager 서버에 대한 엔티티(색인 클래스) 및 속성을 나열할 수 있습니다(예제 참조).

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogV4.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogV4.cpp)은
Cmbroot/Samples/cpp/v4 디렉토리에 있습니다.

조회 실행

다음 예제에서는 AS/400용 Content Manager의 조회를 실행한 후 결과를 처리합니다.

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(계속)

```
        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
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteV4.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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...
```

```

        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++(계속)

```
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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteV4.cpp)은
Cmbroot/Samples/cpp/v4 디렉토리에 있습니다.

매개변수식 조회 실행

다음 예제에서는 매개변수식 조회를 실행합니다.

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);
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSamplePQryV4.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TSamplePQryV4.cpp)은 Cmbroot/Samples/cpp/v4 디렉토리에 있습니다.

Domino.Doc에 대한 작업

Domino.Doc은 비즈니스 문서를 구성, 관리 및 저장하고 비즈니스 내부나 외부에서 이 문서에 액세스할 수 있게 만드는 Lotus Domino 솔루션입니다.

Domino.Doc은 ODMA(Open Document Management API)를 지원하므로 ODMA 사용 응용프로그램으로 문서를 작성, 저장 및 검색할 수 있습니다. ODMA는 HTTP 또는 Lotus Notes[™] 프로토콜을 사용하여 Domino.Doc 서버에 연결합니다.

Domino.Doc는 다음 기능을 포함합니다.

- 하나 이상의 Domino.Doc 서버에 연결 및 연결 해제
- 바인더 검색 기능
- 문서 검색 기능
- 사용자가 익숙한 응용프로그램에서 작업할 수 있도록 ODMA 준수

제한사항: Domino.Doc는 다음을 지원하지 않습니다.

- 문서 메소드 추가, 갱신 및 삭제
- 텍스트 검색 엔진 및 QBIC 검색
- 결합된 조회
- 작업함 및 워크플로우

API를 사용하여 Domino.Doc 오브젝트에 대한 작업을 수행하는 경우 오브젝트를 리턴하기 위한 표현식을 빌드해야 합니다. 이 절에서는 Domino.Doc API의 설계, 오브젝트가 계층 구조에 일치하는 방법 및 표현식을 빌드하는 방법에 대해 설명합니다. 387 페이지의 그림 19에서는 Domino.Doc 오브젝트 모델 및 해당 구성요소 간 관계를 보여줍니다.

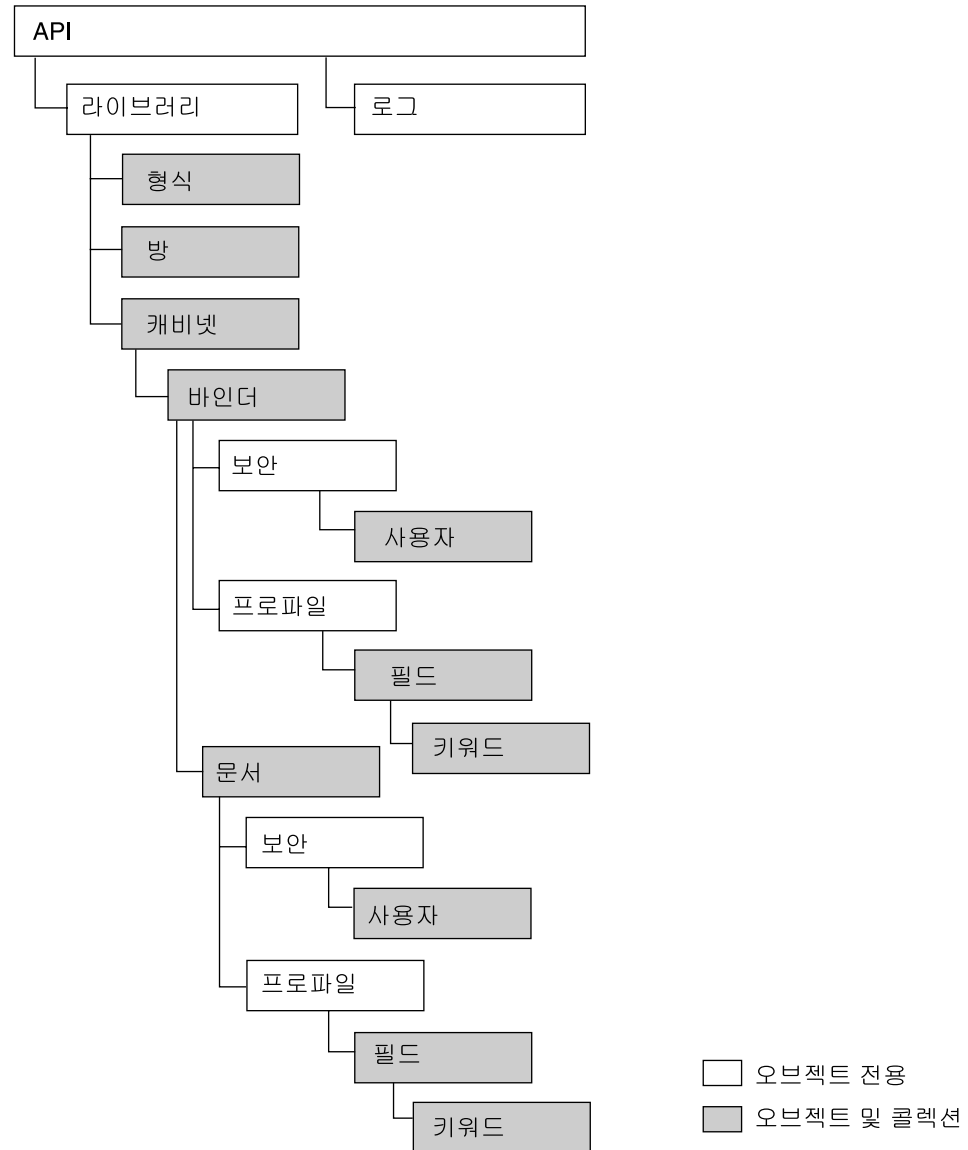


그림 19. Domino.Doc 오브젝트 모델

Domino.Doc(및 이러한 요소를 나타내기 위한 API)에 포함된 요소는 다음과 같이 배열됩니다.

- 라이브러리에 방(DKRoomDefDD 오브젝트) 및 캐비닛(DKCabinetDefDD 오브젝트)이 포함됩니다.
- 각 캐비닛에 바인더(DKBinderDefDD 오브젝트)가 포함됩니다.
- 각 바인더에 프로파일(DKAttrProfileDefDD 오브젝트) 및 보안이 포함됩니다.
- 각 바인더에 문서(DKDocumentDefDD 오브젝트)가 포함됩니다.
- 각 문서에 프로파일(DKAttrProfileDefDD 오브젝트) 및 보안이 포함됩니다.
- 각 프로파일에 필드(DKAttrFieldDefDD 오브젝트)가 포함됩니다.
- 각 필드에 키워드(DKAttrKeywordDefDD 오브젝트)가 포함될 수 있습니다.

엔티티 및 서브엔티티 나열

다음 예제에서는 Domino.Doc의 엔티티(이 예제에서 방) 및 서브엔티티(이 예제에서 캐비닛, 바인더 및 문서)를 나열합니다.

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
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListSubEntitiesDD.java)은 CMBROOT\Samples\java\dd 디렉토리에 있습니다.

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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListEntitiesDD.cpp)은 Cmbroot/Samples/cpp/dd 디렉토리에 있습니다.

다음 예제에서는 문서 속성 및 키워드를 나열합니다.

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
                }
            }
        }
    }
}
...
```

다음 예제에서는 엔티티(이 경우 방)와 연관된 서브엔티티(캐비닛, 바인더 및 문서)를 나열합니다.

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;
}
```

캐비넷 속성 나열

캐비넷은 유용한 속성을 포함하는 유일한 항목입니다. 방에 대한 엔티티 속성을 나열하려고 할 경우, 컬렉션에 아무것도 나타나지 않습니다. 따라서 DKDatastoreDD가 검색 가능한 엔티티를 나열하면 캐비넷만을 나열합니다.

다음 예제에서는 캐비넷 및 해당 속성을 나열합니다.

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();
    ...
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListAttributes.java)은 CMBROOT\Samples\java\dd 디렉토리에 있습니다.

Domino.Doc에 조회 빌드

조회를 하나의 캐비닛으로 제한하려고 할 경우, ENTITY=는 조회 문자열의 첫 번째 단어가 되어야 합니다. ENTITY 매개변수 및 해당 값이 누락되면 엔티티 라이브러리가 검색됩니다. 또한 이 값은 따옴표 (")로 묶어야 합니다.(예: "Diane Cabinet").

QUERY=는 필수 매개변수입니다.

Domino.Doc에서 조회 문자열은 다음과 같습니다.

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

Domino.Doc 콘텐츠 서버를 조회하려면 FTSearch 함수를 사용하십시오. 이 함수가 효율적으로 작동하려면 Domino.Doc 콘텐츠 서버의 텍스트가 완전히 색인화되어야 합니다. 색인을 테스트하려면 IsFTIndexed 등록 정보를 사용하십시오. 색인을 작성하려면 UpdateFTIndex 함수를 사용하십시오.

FTSearch 함수는 콘텐츠 서버의 모든 문서를 검색합니다. 특정 보기 안에서 문서를 검색하려면 NotesView의 FTSearch 함수를 사용하십시오. 특정 문서 컬렉션 내의 문서를 검색하려면, NotesDocumentCollection의 FTSearch 함수를 사용하십시오.

정렬 옵션을 지정하지 않으면 문서가 관련도에 따라 정렬됩니다. 날짜별로 정렬하고자 할 경우, 정렬된 결과에 관련도 점수를 가져오지 않아도 됩니다. 결과

DocumentCollection을 NotesNewsletter 인스턴스로 전달하는 경우, 사용하는 정렬 옵션에 따라 문서 작성 날짜 또는 관련도 점수별로 결과가 정렬됩니다.

조회 구문규칙 사용

조회에 대한 구문규칙이 다음 목록에 나와 있습니다. 우선순위를 대체하고 연산을 그룹화하려면 괄호를 사용하십시오.

일반 텍스트

단어 또는 구문을 있는 그대로 검색하려면 일반 텍스트를 사용하십시오. 검색 키워드 및 기호를 어포스트로피(")로 묶으십시오. LotusScript 리터럴 내부에 있으면 따옴표(")를 사용해야 합니다.

와일드 카드

단어 내의 위치에 상관없이 한 문자를 일치시키려면 물음표(?)를 사용하십시오. 단어 내의 위치에 상관없이 0- n (n 은 숫자임) 문자를 일치시키려면 별표(*)를 사용하십시오.

논리 연산자

검색을 확장하거나 제한하려면 논리 연산자를 사용하십시오. 연산자 및 해당 우선순위는 다음과 같습니다.

1. ! (not)
2. & (and)
3. , (accrue)
4. | (or)

키워드 또는 기호를 사용할 수 있습니다.

근접 연산자

서로 비슷한 단어를 검색하려면 근접 연산자를 사용하십시오. 이 연산자를 사용하려면 전체 텍스트 색인에 단어, 구문 및 단락 구분 기호가 필요합니다. 연산자는 다음과 같습니다.

- near
- sentence
- paragraph

필드 연산자

지정된 필드로 검색을 제한하려면 필드 연산자를 사용하십시오. 구문규칙은 `FIELDfield-name operator`입니다. 여기서 *operator*는 텍스트 및 Rich Text 필드의 경우에는 CONTAINS이며 숫자 및 날짜의 경우에는 네 가지 기호(=, >, >=, <, <=) 중 하나입니다.

exactcase 연산자

다음 표현식에 대한 검색을 지정된 문자로 제한하려면 exactcase 연산자를 사용하십시오.

termweight 연산자

뒤에 오는 표현식의 관련 등급을 조정하려면 termweight n 연산자를 사용하십시오. 여기서 n 은 0-100입니다.

ES(Extended Search)에 대한 작업

Enterprise Information Portal은 IBM Extended Search 7.1을 지원하지만 ES 3은 지원하지 않습니다. ES를 사용하면 다음으로부터 문서를 조회하고 검색할 수 있습니다.

- Lotus Notes 데이터베이스
- NotesPump 데이터베이스
- 파일 시스템
- 웹 검색 엔진

Enterprise Information Portal 클래스 및 API는 다음 ES 기능을 지원합니다.

- 하나 이상의 ES 서버에 연결 및 연결 해제
- ES 서버 나열
- 데이터베이스 및 필드 나열
- GQL(Generalized Query Language)을 사용하여 검색 수행
- 문서 검색
- AIX에서 ES C++ 클래스 및 API 사용
- 연합 계층 및 직접 ES 연결을 통해 매개변수식 텍스트 검색 수행
- 연합 계층에서 CONTAINS_TEXT 및 CONTAINS_TEXT_IN_CONTENT 텍스트 검색 연산자 사용
- Enterprise Information Portal JavaBeans 사용

제한사항: ES는 다음을 지원하지 않습니다.

- 문서 추가, 갱신 및 삭제
- 텍스트 검색 엔진 및 QBIC 검색
- 결합된 조회
- 작업함 및 워크플로우

모든 ES 기능은 ES 구성 데이터베이스에 의해 액세스 및 제어됩니다. 구성 데이터베이스를 사용하면 검색될 데이터 원본의 데이터베이스 정의, 네트워크 주소, 액세스 제어 정보 및 다른 관련 정보를 할당할 수 있습니다.

ES 서버 나열

여러 ES 서버에 대한 액세스를 제공하기 위해 서버 정보가 포함된 cmbdes.ini 파일을 작성할 수 있습니다. 이 파일을 C:\CMBROOT에 저장합니다(여기서 C는 드라이브 이름임). cmbdes.ini 파일은 각 서버에 대해 다음 형식의 한 행을 포함해야 합니다.

`DATASOURCE=TCP/IP address;PORT=port number`

여기서 *TCP/IP*는 ES 서버의 TCP/IP 주소이고 *port number*는 서버에 액세스하기 위해 정의된 포트 번호입니다(예: PORT=80).

엔티티(데이터베이스) 및 속성(필드) 나열

조회를 빌드하여 ES 서버를 검색할 경우 사용 가능한 데이터베이스와 필드 이름을 알아야 합니다. DKDatastoreDES 오브젝트는 `listEntities` 함수를 제공하여 데이터베이스를 나열하고, `listEntityAttrs` 함수를 제공하여 각 데이터베이스의 필드를 나열합니다. 다음 예제에서는 데이터베이스와 해당 필드 목록을 검색하는 방법을 보여줍니다.

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(계속)

```
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();
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogDES.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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 << "NumVals: " << 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;
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TListCatalogDES.cpp)은 Cmbroot/Samples/cpp/ES 디렉토리에 있습니다.

GQL(Generalized Query Language) 사용

ES(Extended Search)는 GQL(Generalized Query Language)을 사용하여 검색을 수행합니다. 표 20에 올바른 GQL 표현식에 대한 예제가 나와 있습니다.

표 20. GQL 표현식

GQL 표현식	설명
"software"	software라는 단어를 포함하는 문서를 검색합니다.
(TOKEN:WILD "exec*")	exec로 시작하는 단어를 포함하는 문서를 검색합니다.
(AND "software" "IBM")	software 및 IBM이라는 단어를 모두 포함하는 문서를 검색합니다.
(START "View" "How")	보기 필드가 How라는 단어로 시작하는 문서를 검색합니다.
(EQ "View" "How Do I?")	보기 필드가 정확히 How Do I?라는 문자열을 포함하는 문서를 검색합니다.
(GT "BIRTHDATE" "19330804")	BIRTHDATE 필드가 1933년 8월 4일보다 큰 문서를 검색합니다.

ES는 조회 유형 DK_DES_GQL_QL_TYPE을 사용합니다. 이 조회 유형은 다음과 같은 구문규칙을 가지고 있습니다.

```
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])]
```

여기서 db_name_list는 쉼표로 구분된 데이터베이스 이름(db_name) 목록이며 ALL은 사용 가능한 모든 데이터베이스를 검색함을 의미합니다. 기본 검색 제한 시간은 30초입니다.

이 예제에서는 조회 문자열을 사용하여 Notes Help 데이터베이스의 문서를 검색합니다. 여기서 보기 필드는 How Do I?이며 예상되는 최대 결과는 5입니다.

```
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
             "COND=(EQ \"View\" \"How Do I?\");" +
             "OPTION=(MAX_RESULTS=5)";
```

이 예제에서는 ES에 대해 GQL 조회를 실행합니다. 조회 실행 후 결과는 dkResultSetCursor 오브젝트에 리턴됩니다.

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dkResultSetCursor pCur = null;
DKNameValuePair 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");
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDES.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDES.cpp)은 Cmbroot/Samples/cpp/ES 디렉토리에 있습니다.

ES에서 DDO 항목 유형 식별

ES에서 DDO는 항상 DK_CM_DOCUMENT 유형을 포함합니다. DDO의 항목 유형을 가져오려면 다음을 호출하십시오.

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
}
```

ES에 PID 작성

지속 식별자(PID)는 문서에 대한 특정 정보를 포함합니다. 오브젝트 유형은 문서를 찾을 수 있는 데이터베이스를 식별합니다. PID는 데이터베이스 이름, | 문자 및 문서 ID를 차례로 사용하여 작성합니다. 예를 들면, 다음과 같습니다.

database name|documentId ()

PID에 대한 자세한 정보는 16 페이지의 『지속 식별자(PID) 이해』 및 52 페이지의 『지속 식별자(PID) 작성』을 참조하십시오.

ES 문서의 콘텐츠 처리

DDO의 각 항목은 필드, 컬렉션 또는 DKParts 오브젝트를 나타냅니다.

필드 단일 필드에 대한 필드 이름은 항목 이름 내에 있습니다. 또한 필드값은 항목 값 내에 있습니다. 필드 등록 정보는 다음과 같을 수 있습니다.

- DK_CM_VSTRING
- DK_CM_FLOAT
- DK_CM_XDOOBJECT
- DK_CM_DATE
- DK_CM_SHORT

컬렉션 필드가 여러 값을 포함할 때 필드 이름은 항목 이름 내에 있습니다. 항목값은 DKSequentialCollection 오브젝트입니다. 필드가 BLOB인 경우, 등록 정보는 DK_CM_COLLECTION 또는 DK_CM_COLLECTION_XDO일 수 있습니다.

DKParts

문서 DDO에는 예약 이름이 DKPARTS인 특정 속성이 있습니다. 그 값은 DKParts 오브젝트입니다. DKPARTS 오브젝트는 문서에 대한 웹 주소(URL) 정보를 저장할 수 있습니다. 또한 DKPARTS는 문서의 URL을 나타내는 문자열로 해당 콘텐츠에 XDO를 포함할 수 있습니다.

이 예제에서는 DDO의 콘텐츠를 처리합니다.

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);
        }
    }
}
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDES.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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;
DKPidXDOES *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++(계속)

```
        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;
                    }
                }
            }

            break;
        }
        // continued...
```


C++(계속)

```
        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;
```

이 예제가 수행된 완전한 샘플 응용프로그램(TExecuteDES.cpp)은
Cmbroot/Samples/cpp/ES 디렉토리에 있습니다.

문서 검색

DKDatastoreDES 오브젝트에서 문서를 검색하려면 문서와 문서 ID를 포함하는 데이터베이스의 이름을 알아야 합니다. 또한 DDO를 콘텐츠 서버와 연관시킨 다음 연결을 설정해야 합니다. 이 예제에서는 문서를 검색합니다.

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();
```

이 예제가 수행된 완전한 샘플 응용프로그램(TRetrieveDDODES.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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();
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TRetrieveDDODES.cpp)은 Cmbroot/Samples/cpp/ES 디렉토리에 있습니다.

BLOB(2진 대형 오브젝트) 검색

DKDatastoreDES 오브젝트에서 BLOB를 검색하려면 문서가 있는 데이터베이스 이름, BLOB가 포함된 문서 ID 및 BLOB를 포함하는 필드 이름을 알아야 합니다. 또한 DDO를 콘텐츠 서버와 연관시킨 다음 연결을 설정해야 합니다.

다음 예제에는 ES files라는 파일 시스템 데이터베이스에 D:\desdoc\README.html이라는 HTML 파일이 들어 있습니다. HTML 파일을 포함하는 필드는 Doc\$Content로 이름 지정됩니다. 다음 예제에서는 HTML 파일을 검색하고 이 파일을 D:\DESReadme.html로 저장합니다.

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKBlobDES xdo = new DKBlobDES(dsDES);
DKPidXDOES pid = new DKPidXDOES();
pid.setDocumentId("D:\\desdoc\\README.html");
pid.setDatabaseName("DES files");
pid.setFieldName("Doc$Content");
xdo.setPidObject(pid);
xdo.retrieve("c:\\DESReadme.html");
```

이 예제가 수행된 완전한 샘플 응용프로그램(TRetrieveXDOES.java)은 CMBROOT\Samples\java\d1 디렉토리에 있습니다.

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);
DKPidXDOES 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((DKPidXDO*)&pid);

p->retrieve("c:\\temp\\DESReadme.html");

cout << "retrieve executed" << endl;
...
```

이 예제가 수행된 완전한 샘플 응용프로그램(TRetrieveXDOES.cpp)은 Cmbroot/Samples/cpp/ES 디렉토리에 있습니다.

문서와 MIME 유형 연관

ES는 MIME(Multipurpose Internet Mail Extension) 유형의 ID를 직접 지원하지 않습니다. 그러나 웹 브라우저 내에 표시할 XDO의 MIME 유형을 알아야 합니다.

CMBCC2MIME.INI 파일은 문서의 MIME 유형 결정에 사용됩니다. NotesPump 또는 FileSystem 데이터베이스에서 ES 조치가 BLOB를 리턴할 때 MIME 유형을 BLOB에 할당할 수 있는지 여부를 결정하기 위해 CMBCC2MIME.INI 파일을 검색합니다. 기본 MIME 유형은 text/html입니다. cmbcc2mime.ini.samp라는 샘플 파일은 Samples 디렉토리에 있습니다.

ES에서 연합 검색 사용

연합 조화를 작성할 때 ES에 사용되는 구문규칙은 SQL 구문규칙과 유사합니다. 연합 조회 표현식은 ES에 제출되기 전에 GQL 구문규칙으로 변환됩니다. 그러나 SQL과 GQL 문법에 차이점이 있으므로 SQL 문법의 서브세트만이 Enterprise Information Portal에서 지원됩니다.

표 21에는 지원되는 SQL 비교 및 논리 연산자를 GQL로 변환한 내용이 요약되어 있습니다.

표 21. SQL 및 GQL 연산자

SQL 연산자	GQL 연산자
AND	AND
OR	OR
NOT	지원되지 않음
IN	지원되지 않음
BETWEEN	BETWEEN
EQ	EQ
NEQ	지원되지 않음
GT	GT
LT	LT
LIKE	지원되지 않음
GEQ	GTE
LEQ	LTE
NOTLIKE	지원되지 않음
NOTIN	지원되지 않음
NOTBETWEEN	지원되지 않음

Panagon Image Services에 대한 작업(Java 전용)

Panagon Image Services(FileNET)에 제공된 Enterprise Information Portal API 클래스를 사용하여 Panagon Image Services 서버의 콘텐츠에 액세스할 수 있습니다. Panagon Image Services에 대한 지원이 EIP의 특수 기능으로만 사용 가능합니다.

제한사항: Panagon Image Services는 텍스트 검색 엔진 및 QBIC 검색 또는 결합된 조회를 지원하지 않습니다.

데이터 모델링

Enterprise Information Portal에는 커넥터가 지원하는 콘텐츠 서버의 해당 오브젝트에 맵핑되어야 하는 일부 데이터 모델링 개념이 있습니다. 아래의 표에서는 FileNET 환경에서의 EIP 데이터 모델을 보여줍니다.

EIP 개념	FileNET
콘텐츠 서버	Panagon Image Services 서버
서버	서버
엔티티	문서 클래스
속성	색인
DDO	문서(폴더) FileNET의 폴더는 일시적으로 문서를 구성하는 데 사용됩니다. 현재 이들 폴더는 커넥터에서 모델링되어 있지 않습니다.
XDO	페이지 콘텐츠, 주식
DDO에 대한 PID	document_id, 기본 ID로서
페이지 콘텐츠 XDO에 대한 PID	System_serial_num + document_id + page#
주식 XDO에 대한 PID	System_serial_num + document_id + page# + annotation_id

FileNET 커넥터는 FileNET에서 사용자 정의 색인 또는 속성만을 리턴하며, 이는 연합 엔티티, 속성 맵핑 및 검색 결과 속성 표시 목록에서 사용될 수 있습니다. 다음 표에서는 FileNET 커넥터에서 해당 EIP 속성 데이터 유형에 대한 FileNET 지원 색인 또는 속성 데이터 유형의 맵핑을 보여줍니다.

FileNET 색인 데이터 유형	EIP 속성 데이터 유형
숫자	double
날짜	FileNET 색인에 대해 정의된 최대 길이로 설정된 max_length를 가진 varchar
메뉴	max_length = 14(메뉴 이름에 대해 최대)인 varchar

EIP의 모든 XDO에는 MIME 유형이 있습니다. EIP 클라이언트는 MIME 유형을 사용하여 문서 표시 방법을 결정합니다. FileNET 커넥터는 동일 문서의 모든 XDO가 동일한 MIME 유형이라고 가정합니다. XDO의 MIME 유형은 속해 있는 문서의 F_DOCFORMAT FileNET 시스템 속성값에 따라서 설정됩니다. 해당 값이 널(null)인 경우, F_DOCTYPE FileNET 시스템 속성이 이미지 유형을 나타내면 MIME 유형은 image\tiff로 설정됩니다. 그렇지 않은 경우에는 기본 MIME 유형 application/octet-stream으로 설정됩니다.

Panagon Image Services의 문서 및 페이지

DDO를 사용하여 Panagon Image Services의 문서를 나타내려면 DDO PID의 필드 네 개가 적절히 설정되어야 합니다.

1. DatastoreType - DKConstantFN에 정의된 상수 DK_FN_DSTYPE입니다. DKDatastoreFN의 datastoreType() 메소드로부터 이 값을 가져올 수 있습니다.
2. DatastoreName - FileNET 도메인 조직 이름입니다. DKDatastoreFN의 datastoreName() 메소드로부터 이 값을 가져올 수 있습니다.
3. ObjectType - 문서가 속하는 FileNET 문서 클래스(EIP의 원래의 엔티티 이름) 이름입니다.
4. PrimaryID - FileNET이 할당한 FileNET 문서 번호입니다.

일단 PID가 설정되면 응용프로그램이 DKDatastoreFN 또는 DKDDO 클래스에서 retrieve() 메소드를 호출하여 DDO의 속성 또는 콘텐츠를 가져올 수 있습니다.

EIP의 문서 DDO는 예약 이름이 DKPARTS인 특정 속성을 포함하고 해당 값은 DKParts 오브젝트입니다. DKParts 오브젝트는 BLOB(2진 대형 오브젝트)의 컬렉션입니다. 문서 내의 FileNET 부분(페이지/주석)은 DKPasts 컬렉션의 요소인 DKBlobFN 오브젝트(DKXDO의 서브클래스)로서 나타납니다.

FileNET 페이지는 일반적인 페이지가 아닙니다. FileNET의 단일 페이지(또는 단일 부분) 문서는 여러 실제 페이지를 포함할 수 있으나 이들은 FileNET 문서 내의 단일 페이지에 저장됩니다. FileNet의 여러 페이지 문서는 Panagon Capture Software 또는 일괄처리 입력 시스템을 사용하여 작성된 여러 부분 또는 여러 파일을 포함하는 문서를 참조합니다.

엔티티 및 속성 나열

Panagon Image Services 서버를 DKDatastoreFN으로 나타냅니다. 콘텐츠 서버를 작성하여 연결한 후 Panagon Image Services 서버에 대한 속성 및 엔티티(문서 x 클래스)를 나열할 수 있습니다. 다음 예제에서는 이 작업을 수행하는 방법을 보여줍니다.

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)
```

샘플 응용프로그램에 대해서는 CMBROOT\Samples\java\fn 디렉토리를 참조하십시오.

조회

EIP는 Panagon Image Services에 대한 세 가지 기본 조회 양식을 지원합니다.

- 조회 오브젝트
- 명령 문자열
- DKCQExpr

조회 오브젝트 및 조회의 명령 문자열 양식은 모두 Panagon Image Services 서버에 직접 액세스하는 응용프로그램에서 일반적으로 사용됩니다. EIP 연합 콘텐츠 서버 및 조회는 조회의 DKCQExpr 양식만을 사용하여 개별 콘텐츠 서버와 인터페이스합니다.

Panagon Image Services 서버 조회에 대한 샘플 응용프로그램은 CMBROOT\Samples\java\fn 디렉토리에 있습니다.

조회 문자열 구문 및 매개변수

FileNET 커넥터는 조회를 제출하고 실행하기 위해 DKParametricQuery 사용을 지원합니다. DKParametricQuery는 명령 문자열로 구성될 수 있습니다. 명령 문자열은 FileNET 필터 조건에 대한 조회 스펙을 나타냅니다. 기타 조회 옵션(ENTITY_NAME, MAX_RESULTS, CONTENT) 및 FileNET 키 조건(KEY)은 (이름, 값) 쌍으로 evaluate() 또는 execute() 메소드에 대한 매개변수로서 승인됩니다(예: DKNVPair).

문자열 유형 및 매개변수 이름의 값은 항상 DKConstant 및 DKConstantFN에 정의됨에 유의하십시오. 리턴 결과는 evaluate()에 대한 문서 DDO의 컬렉션 및 execute()에 대한 결과를 가리키는 결과 세트 커서입니다.

명령에 대한 조건식 구문규칙 및 KEY 매개변수는 FileNET WAL PRS_Parse() 함수에 의해 지원된 FileNET 조회 구문규칙을 따릅니다. 다음의 연산자를 승인합니다.

연산자	사용
AND	논리 and
OR	논리 or
NOT	논리 not
<	미만
<=	이하
=	같음
>	초과
>=	이상
!=	같지 않음
+	더하기
-	빼기
*	곱하기

연산자	사용
/	나누기
IN RANGE	범위는 구문규칙 <code>IN RANGE op constant op constant</code> 로 지정됩니다. 여기서 <i>op</i> 는 <, <=, > 또는 >=이고 <i>constant</i> 는 숫자 또는 문자열 상수입니다.
LIKE	유사. LIKE 연산자의 오른쪽 패턴은 와일드 문자(임의 문자에 일치하는 것을 표시)로서 ? 및 와일드 카드로서의 *를 포함할 수 있는 단순 인용 문자열이어야 합니다.
DEFINED(x)	정의된 함수는 제공된 열 이름 <i>x</i> 가 널(null)이 아닌 경우 0이 아닌 값을, 값이 널(null)인 경우 0을 리턴합니다.

피연산자는 숫자 상수, 문자열 상수 및 속성 이름입니다. 명령 문자열은 직접 FileNET 콘텐츠 서버를 검색하는 데 사용되므로 속성 이름은 모두 로컬 FileNET 속성/색인 이름입니다. 연합 속성 이름 및 스키마 매핑은 지원되지 않으며 필요하지도 않습니다.

숫자의 형식은 `+ddd.dddE+eee`입니다. 여기서 *ddd*는 0 이상의 자릿수를 나타내고 +는 + 또는 -(+는 선택적), *eee*는 지수의 0에서 세 자리까지의 자릿수를 나타냅니다. .은 소수점(선택적), 그리고 E는 지수의 시작(선택적) 부분입니다.

문자열 상수는 작은따옴표로 묶이며 문자열 내의 삽입된 작은따옴표는 두 개의 연속 작은따옴표로 나타냅니다.

명령 문자열에 대한 조건식은 많은 피연산자와 연산자를 포함할 수 있습니다. 필터에서 괄호를 사용하여 우선순위를 대체할 수도 있습니다. 필터 조건의 예제는 다음과 같습니다.

```
PROD_PRICE > 100 AND (PROD_DATE > '10/15/92' OR PROD_NAME LIKE 'comp*')
PROD_PRICE >= 100 AND (NOT (PROD_ID < 30))
```

KEY 매개변수에 대한 조건식은 FileNET에 검색 키로서 정의된 하나의 속성 이름과 하나의 조건만을 포함해야 합니다. FileNET에 정의된 키 이름이 사용되어야 하며 비전환 열 이름은 사용될 수 없습니다. 키 문자열의 예제는 다음과 같습니다.

```
PROD_PRICE = 100000
PROD_PRICE IN RANGE >= 100000 <= 200000
```

추가 검색 옵션

evaluate() 및 execute()에 대해 지원되는 기타 선택적 매개변수는 다음과 같습니다.

(ENTITY_NAME, entity_Name):

조회 범위로 엔티티 이름(FileNET 문서 클래스 이름)을 지정합니다. 샘플:

```
parms[1] = new DKNVPair (DK_FN_PARM_ENTITY_NAME, entity_Name);
```

Panagon Image Services 서버에 제출된 실제 조회 문자열은

AND(F_DOCCLASSNUMBER = doc_cls#)로 추가되며, 여기서 = doc_cls#은 지

정된 엔티티 이름의 일치하는 문서 클래스 번호입니다. 지정되지 않은 경우에는 모든 FileNET 문서 클래스의 문서가 검색됩니다.

(MAX_RESULTS, Max_results)

지정된 최대 결과 수만이 결과 컬렉션에 리턴됩니다. 값이 0이거나 지정되지 않은 경우, FileNET이 허용한 최대 결과 수가 리턴됩니다. 샘플:

```
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, Max_results);
```

(CONTENT, YES / NO / ATTRONLY)

이 매개변수는 콘텐츠 서버 레벨에서 DK_CM_OPT_CONTENT 옵션 세트를 겹쳐줍니다.

- YES - 결과 문서는 검색된 해당 콘텐츠를 가집니다(기본값).
- NO - 문서 ID만이 결과 DDO에서 설정됩니다.
- ATTRONLY - 문서 ID, 데이터 속성 및 해당 값이 결과 DDO에 설정됩니다.

샘플:

```
dsFN.setOption(DK_CM_OPT_CONTENT, DK_CM_CONTENT_YES);
```

예를 들어, 콘텐츠 값 YES는 결과 문서 DDO가 해당 PID, 오브젝트 유형, 등록 정보 및 모든 속성 세트를 갖도록 합니다. 또한 이 옵션은 DKPARTS 속성이 실제 콘텐츠가 아닌 PID 정보 세트를 가진 콘텐츠 부분(부분 및 주석에 대한 XDO)의 컬렉션에 설정되도록 합니다. CONTENT 값이 ATTRONLY인 경우, 결과 문서 DDO PID, 오브젝트 유형, 등록 정보 및 모든 데이터 속성이 설정됩니다. CONTENT 값이 NO인 경우, 결과 문서 DDO PID, 오브젝트 유형 및 등록 정보가 설정됩니다. 문서 부분 또는 XDO 콘텐츠는 DKBlobFN 클래스의 retrieve() 메소드를 사용하여 필요할 경우 명시적으로 검색할 수 있습니다.

예외 처리 및 메시지

Panagon Image Services는 EIP에 정의된 동일한 Java 예외 클래스 세트를 사용합니다. 또한 예외에 포함된 텍스트 메시지는 가능할 때마다 DKMessageID에서 공통적으로 정의된 메시지 중 하나를 사용합니다. 예를 들어, DKUsageError 예외는 (DKMessage.getMessage(DK_CM_MSG_NOTIMP) + this.getClass().getName() + XXXX , DK_CM_MSG_NOTIMP)로 메시지를 포함합니다. 여기서 XXX는 프로그램이 호출하려는 메소드 이름입니다.

DKDatastoreAccessError 예외는 다음 형식으로 FileNET 특정 메시지 텍스트를 포함합니다.

```
WAL_function_name [IMS_SESSION= sss, ERR_CAT=ccc, ERR_FUN=fff, ERR_NUM=nnn]
```

FileNET 명령 msg(c:\fnsw\client\bin 디렉토리를 통해 액세스할 수 있음)를 사용하여 세 자릿수 *ccc*, *fff* 및 *nnn*(예: msg *ccc,fff,nnn*)을 사용하는 원래의 FileNET 오류 메시지 텍스트를 더 자세하게 표시할 수 있습니다.

Panagon Image Services 서버에 대해 고유하게 사용된 EIP 메시지 ID는 3401-3600입니다.

관계형 데이터베이스에 대한 작업

Enterprise Information Portal API 클래스는 IBM DB2 Universal Database 및 C++용 JDBC(Java Database Connectivity), ODBC(Open Database Connectivity) 또는 IBM DB2 DataJoiner를 사용하는 기타 관계형 데이터베이스를 지원합니다.

관계형 데이터베이스에 연결

관계형 데이터베이스를 나타내려면 DKDatastorexx 오브젝트를 작성하십시오. 여기서 xx는 DB2 데이터베이스 DB2, Java Database Connectivity의 경우에는 JDBC, Open Database Connectivity의 경우에는 ODBC입니다. 다음 예제 샘플에서는 DB2 샘플 데이터베이스에 연결합니다.

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). . .
```

연결 시 데이터베이스 이름을 사용하십시오.

연결 문자열

관계형 데이터베이스에 연결 시 연결 문자열을 지정하여 이를 매개변수로 전달할 수 있습니다. 여러 연결 문자열을 지정하는 경우 세미콜론(;)으로 구분하십시오. 연결 문자열은 다음 형식을 취할 수 있습니다.

DB2 DataJoiner 및 ODBC용 연결 문자열:

NATIVECONNECTSTRING=(*native connect string*)

연결 시 데이터베이스에 전달할 원래의 연결 문자열을 지정합니다. 콘텐츠 서버의 정보를 점검하여 올바른 원래의 연결 문자열을 결정하십시오.

SCHEMA=*schema name*

listEntities, listEntityAttrs, listPrimaryKeyNames, listForeignKeyNames 메소드 실행 시 사용할 데이터베이스 스키마 이름을 지정합니다.

JDBC용 연결 문자열:

SCHEMA=*schema name*

listEntities, listEntityAttrs, listPrimaryKeyNames, listForeignKeyNames 메소드 실행 시 사용할 데이터베이스 스키마 이름을 지정합니다.

구성 문자열

구성 문자열을 지정하여 구성 메소드에 매개변수로 전달할 수 있습니다. 여러 구성 문자열을 지정하는 경우 세미콜론(;)으로 구분하십시오. 구성 문자열은 다음 형식을 가집니다.

DB2 DataJoiner 및 ODBC용 구성 문자열:

CC2MIMEURL=(*URL*)

cmbcc2mime.ini 파일을 고유한 자원 위치 지정자 주소로 지정합니다. 파일 위치에 따라 구성 문자열 또는 CC2MIMEFILE 양식을 사용하십시오.

CC2MIMEFILE=(*filename*)

cmbcc2mime.ini 파일을 이름으로 지정합니다.

DSNAME=(*content server name*)

콘텐츠 서버의 이름을 지정합니다. 연합 조회 및 기타 연합 함수의 경우, Enterprise Information Portal은 이 작업을 자동으로 설정합니다.

AUTOCOMMIT=ON | OFF

자동 확약을 설정 또는 해제합니다. 기본값은 해제입니다. 이 콘텐츠를 서버를 연합 조회 및 기타 연합 함수에 사용하면 기본적으로 자동 확약이 설정됩니다.

JDBC용 구성 문자열:

CC2MIMEURL=(*URL*)

cmbcc2mime.ini 파일을 고유한 자원 위치 지정자 주소로 지정합니다. 파일 위치에 따라 구성 문자열 또는 CC2MIMEFILE 양식을 사용하십시오.

cmbcc2mime.ini 파일을 이름으로 지정합니다.

JDBC_SERVER_URL=(URL)

cmbjdcsrvs.ini 파일을 고유한 자원 위치 지정자 주소로 지정합니다. 이 파일은 JDBC 서버의 목록을 포함합니다.

JDBC_SERVER_FILE=(filename)

JDBC 서버 목록이 포함된 cmbjdcsrvs.ini 파일을 파일 이름으로 지정합니다.

JDBC_DRIVER=(JDBC driver)

사용할 JDBC 드라이버를 지정합니다. 이 작업은 시스템 관리 클라이언트 클라이언트 프로그램 사용 시 자동으로 설정됩니다.

DSNAME=(content server name)

컨텐츠 서버의 이름을 지정합니다. 연합 조회 및 기타 연합 함수의 경우, Enterprise Information Portal은 이 작업을 자동으로 설정합니다.

AUTO_COMMIT=ON | OFF

자동 확약을 설정 또는 해제합니다. 기본값은 해제입니다. 이 컨텐츠 서버를 연합 조회 및 기타 연합 함수에 사용하면 기본적으로 자동 확약이 설정됩니다.

엔티티 및 엔티티 속성 나열

관계형 데이터베이스에 대해 컨텐츠 서버를 작성하여 연결한 후 엔티티 및 엔티티 속성을 나열할 수 있습니다. 다음 예제에서는 목록을 검색한 후 단계를 밟는 방법을 보여줍니다.

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();
```

완전한 예제에 대해서는 CMBROOT\Samples\java\d1 디렉토리의 TListCatalogDB2.java, TListCatalogJDBC.java 및 TListCatalogDJ.java 를 참조하십시오.

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++(계속)

```
        delete pIter2;
        delete pCol2;
        delete pEnt;
    }
    delete pIter;
    delete pCol;
    dsDB2.disconnect();
}
    catch(DKException &exc)
    . . .
```

완전한 예제에 대해서는 CMBROOT\Samples\cpp\db2, odbc TListCatalogDB2.cpp 및 dj 디렉토리의 TListCatalogODBC.cpp를 참조하십시오.

조회 실행

조회를 실행하려면 먼저 조회 문자열을 작성한 다음 조회를 실행하십시오. 다음 예제에서는 조회를 실행한 후 결과를 처리합니다.

Java

```
// ----- After creating a datastore and connecting, build the
//           query and parameters and execute it
sDB2 = new DKDatastoreDB2();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = new DKNameValuePair[2];
String strMax = "5";
parms[0] = new DKNameValuePair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[1] = new DKNameValuePair(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(계속)

```
// ----- 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();
```

완전한 예제에 대해서는 CMBROOT\Samples\java\d1 디렉토리의 TExecuteDB2.java, TExecuteJDBC.java 및 TExecuteDJ.java를 참조하십시오.

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 qstrng = "SELECT * FROM EMPLOYEE";
    // ----- Execute the query
    pCur = dsDB2.execute(qstrng, 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++(계속)

```
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)
. . .
```

완전한 예제에 대해서는 TExecuteDB2.cpp, TExecuteODBC.cpp 및 CMBROOT\Samples\cpp 디렉토리의 TExecuteDJ.cpp를 참조하십시오.

사용자 조정 콘텐츠 서버 커넥터 작성

현재 Enterprise Information Portal에 포함되지 않은 사용자 조정 콘텐츠 서버에 대한 사용자 고유의 서버 정의를 작성할 수 있습니다. 사용자 조정 서버를 EIP로 통합하는 경우, 자체 Java 또는 C++ 클래스를 제공하여 정의를 지원해야 합니다.

사용자 조정 콘텐츠 서버 커넥터 개발

객체 지향 API 프레임워크는 다음 목적으로 개발되었습니다.

- 추가 데이터 기억영역 시스템 또는 콘텐츠 서버를 프레임워크에 추가할 수 있음
- 모든 복합 콘텐츠 서버 데이터 유형에 맵핑할 수 있는 기능
- 모든 콘텐츠 서버 데이터 액세스에 대한 공통 오브젝트 모델
- 텍스트 검색 엔진 및 이미지 검색(QBIC)과 같이 다른 유형의 검색 엔진이 결합된 것에 사용할 수 있는 융통성 있는 메커니즘
- Java 응용프로그램 사용자를 위한 클라이언트/서버 구현

특정 객체 지향 API에 대한 정보는 온라인 API 참조서를 참조하십시오.

사용자 조정 콘텐츠 서버를 Enterprise Information Portal에 통합할 경우 다음 작업을 수행해야 합니다.

- `com.ibm.mm.sdk.common` 패키지를 가져오십시오.
- **Java 전용:** 공통 프레임워크에 액세스하려면 `cmbcm81.jar`(Java) 파일에 링크하십시오.
- **C++ 전용:** 공통 프레임워크에 액세스하려면 디버그되지 않은 버전인 `cmbcm817.dll` 및 디버그된 버전인 `cmbcm8167.dll`에 링크하십시오.

Enterprise Information Portal 데이터베이스 하부 구조

`dkDatastore` 클래스는 Enterprise Information Portal과 콘텐츠 서버 간 기본 인터페이스입니다. 각 콘텐츠 서버는 `dkDatastore` 클래스가 특정 콘텐츠 서버에 대한 구현 정보를 제공하기 위해 구현하는 개별 클래스를 포함합니다. 각 콘텐츠 서버 유형은 `DKDatastorexx`라는 클래스로 나타납니다. 여기서 `xx`는 특정 콘텐츠 서버의 이름이나 유형을 나타냅니다. 42 페이지의 표 7에는 Enterprise Information Portal에 제공된 현재 콘텐츠 서버가 나열되어 있습니다.

서버 정의를 작성할 때 EIP 시스템 관리 클라이언트에 콘텐츠 서버용으로 작성하는 `DKDatastore` 클래스를 지정해야 합니다.

Enterprise Information Portal의 공통 클래스

dkDDO

`dkDDO` 클래스는 오브젝트의 유형에 관계없이 오브젝트의 데이터를 정의하고 액세스하는 표현 및 프로토콜을 제공합니다. DDO 프로토콜은 오브젝트의 각 데이터 항목을 정의, 추가 및 액세스하는 함수 세트로 구현됩니다. 이 프로토콜을 사용하여 콘텐츠 서버의 유형에 관계없이 동적으로 오브젝트를 작성하고 콘텐츠 서버에서 이 오브젝트를 가져올 수 있습니다.

콘텐츠 서버를 구현할 때 콘텐츠 서버 클래스에 등록된 스키마 맵핑 정보를 활용할 수 있습니다. 스키마는 개별 지속적 데이터 항목을 콘텐츠 서버의 기본 표현에 맵핑합니다.

DDO는 속성 세트를 포함하고, 각 속성은 속성과 연관된 등록 정보, 값 및 유형을 포함합니다. DDO 자체는 전체적으로 DDO에 속하는 등록 정보를 포함할 수 있습니다. 예를 들어, 임의의 클래스를 Content Manager 콘텐츠 서버의 항목 또는 OnDemand의 문서에 맵핑할 수 있습니다.

Java

이 도표는 dkDDO 클래스의 계층 구조를 나타냅니다.

```
java.lang.Object
|
+----com.ibm.mm.sdk.common.dkDataObjectBase
|
+----com.ibm.mm.sdk.common.dkDataObject
|
+----com.ibm.mm.sdk.server.DKDDOBase
|
+----com.ibm.mm.sdk.server.DKDDO
```

dkXDO

dkXDO 클래스는 독립적으로 존재하거나 DDO의 일부로 존재할 수 있는 복잡한 사용자 정의 유형 또는 LOB(대형 오브젝트)를 나타냅니다. 따라서 지속 식별자(PID)를 포함하며 함수를 작성, 검색, 갱신 및 삭제합니다.

dkXDO 클래스는 독립적인 콘텐츠 서버 액세스, 작성, 검색, 갱신 및 삭제 함수를 정의하여 dkXDODBase의 공용 인터페이스를 확장합니다. 이 함수는 응용 프로그램이 관련 DDO 클래스 오브젝트나 독립형 XDO 없이 오브젝트의 데이터를 콘텐츠 서버에 저장하고 검색할 수 있게 합니다.

콘텐츠 서버에서 해당 위치를 찾으려면 독립형 XDO에 대한 PID를 설정해야 합니다. DDO와 함께 XDO를 사용하는 경우 PID가 자동으로 설정됩니다. 예를 들어, 클래스를 Content Manager 콘텐츠 서버의 항목에 매핑할 수 있고 OnDemand 콘텐츠 서버의 메모에 매핑될 수 있습니다.

Java

다음은 dkXDO 클래스의 클래스 계층 구조입니다.

```
java.lang.Object
|
+----com.ibm.mm.sdk.server.dkDataObjectBase
|
+----com.ibm.mm.sdk.server.dkXDODBase
|
+----com.ibm.mm.sdk.server.dkXDO
```

dkCollection

dkCollection 클래스는 오브젝트의 컬렉션입니다. dkCollection은 조회를 평가할 수 없습니다. 컬렉션은 이름을 가질 수 있습니다(기본 이름은 빈 문자열임).

예를 들어, DKParts는 DKSequentialCollection의 서브클래스이고, 이는 차례로 dkCollection의 서브클래스가 됩니다.

DKResults

DKResults는 dkQueryableCollection의 서브클래스이므로 정렬 및 양방향 반복자를 지원하며 조회가 가능합니다. DKResults 클래스의 요소 구성원은 오브젝트이고, 조회 결과를 나타내는 dkDDO 클래스의 인스턴스입니다. 이 클래스에서 작성한 반복자는 dkSequentialIterator입니다.

Java

다음은 DKResults 클래스의 클래스 계층 구조입니다.

```
java.lang.Object
|
+----com.ibm.mm.sdk.server.DKSequentialCollection
|
+----com.ibm.mm.sdk.server.dkQueryableCollection
|
+----com.ibm.mm.sdk.server.DKResults
```

dkQuery

dkQuery는 특정 콘텐츠 서버와 연관된 조회 오브젝트의 인터페이스입니다. 이 인터페이스를 구현하는 오브젝트는 콘텐츠 서버 클래스에서 작성됩니다. 조회 결과는 일반적으로 DKResults 오브젝트입니다. dkQuery 인터페이스의 구체적인 구현에 대한 예제는 DKParametricQuery, DKTextQuery 및 DKImageQuery이며, 이는 연관된 콘텐츠 서버에서 작성됩니다.

dkCQExpr

dkCQExpr 클래스는 복합 및 결합된 조회 표현식을 나타냅니다. 이 클래스는 dkQExpr 조회 표현식 트리를 포함할 수 있으며 이 트리는 매개변수식, 텍스트 및 이미지 조회의 조합을 포함할 수 있습니다. 각 콘텐츠 서버의 연합 검색을 허용할 수 있게 하려면 콘텐츠 서버가 이 dkCQExpr 오브젝트를 처리할 수 있어야 합니다.

dkSchemaMapping

dkSchemaMapping은 연합 엔티티와 콘텐츠 서버의 원래의 엔티티 간에 연관된 맵핑을 정의하는 인터페이스입니다. 연합 엔티티와 속성을 조회 및 리턴 결과에 대한 원래의 엔티티와 속성에 맵핑 해제했다가 다시 맵핑하려면 콘텐츠 서버가 이 맵핑 클래스를 이해해야 합니다.

dkDatastore 및 관련 클래스

해당 콘텐츠 서버에 대한 다음의 각 클래스나 인터페이스에 대해 하나의 구체적인 클래스를 구현해야 합니다. 예를 들어, OnDemand 서버에서 dkDatastore 인터페이스가 구현하는 구체적인 클래스는 DKDatastoreOD입니다.

dkDatastore

dkDatastore는 콘텐츠 서버에 대한 연결, 해당 트랜잭션 및 명령을 나타내고 관리합니다. evaluate 함수를 지원하므로 조회 관리자의 서브클래스로 간주할 수 있습니다.

dkDatastore 인터페이스에서 기본 메소드는 다음과 같습니다.

connect()

콘텐츠 서버에 연결합니다.

disconnect()

콘텐츠 서버에서 연결 해제합니다.

evaluate(), execute(), executeWithCallback()

콘텐츠 서버를 조회합니다.

commit(), rollback()

콘텐츠 서버에서 트랜잭션을 수행합니다.

제한사항: 일부 콘텐츠 서버는 이러한 함수를 지원하지 않습니다.

registerServices(), unregisterServices()

검색 엔진을 등록합니다.

changePassword(userid, oldPasswd, newPasswd)

콘텐츠 서버에서 현재 로그인 사용자 ID의 로그인 암호를 변경합니다.

listDataSources()

로그인에 사용할 콘텐츠 서버 사용자 ID 오브젝트의 컬렉션을 리턴합니다. 이 함수를 사용하기 위해 콘텐츠 서버에 연결할 필요는 없습니다.

listDataSourceNames()

콘텐츠 서버 이름의 배열을 리턴합니다.

getExtension(String)

콘텐츠 서버에서 dkExtension 오브젝트를 가져옵니다. 지정된 확장자가 아직 없지만 콘텐츠 서버에서 지원되는 경우, 새로 작성된 오브젝트가 리턴되고, 그렇지 않으면 널(null) 값이 리턴됩니다.

addExtension(String, dkExtension)

새 확장자 오브젝트(XDO)를 이 콘텐츠 서버에 추가합니다.

createDDO(String,int)

지정된 오브젝트 유형과 플래그에 따라 데이터 오브젝트를 작성합니다. CreateDDO는 모든 등록 정보와 속성 세트를 사용하여 새 DKDDO 오브젝트를 리턴합니다. 호출 프로그램은 이 데이터 오브젝트에 대한 속성값을 제공해야 합니다.

dkDatastore 인터페이스의 데이터 오브젝트 조작 메소드는 다음과 같습니다.

addObject(dkDataObject)

컨텐츠 서버에 새 문서 또는 폴더를 추가합니다.

retrieveObject(dkDataObject)

컨텐츠 서버에서 문서나 폴더를 검색합니다.

deleteObject(dkDataObject)

컨텐츠 서버에서 문서 또는 폴더를 삭제합니다.

updateObject(dkDataObject)

컨텐츠 서버에서 문서나 폴더를 갱신합니다.

moveObject(dkDataObject, String)

하나의 엔티티에서 다른 엔티티로 폴더나 문서를 이동합니다.

dkDatastore 인터페이스의 스키마 맵핑 관련 메소드는 다음과 같습니다.

registerMapping(DKNVPair)

이 컨텐츠 서버에 맵핑 정보를 등록합니다.

unRegisterMapping(String)

이 컨텐츠 서버에서 맵핑 정보를 제거합니다.

listMappingNames()

이 컨텐츠 서버에서 맵핑 이름의 배열을 리턴합니다.

getMapping(String)

dkSchemaMapping 오브젝트를 리턴합니다.

dkDatastoreDef

dkDatastoreDef 인터페이스는 컨텐츠 서버 정보에 액세스하여 해당 엔티티를 작성, 표시 및 삭제하는 함수를 정의합니다. 이는 dkEntityDef 오브젝트 콜렉션도 관리합니다.

표 22에 dkDatastoreDef 인터페이스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 22. dkDatastoreDef에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKDatastoreDefICM
OnDemand	DKDatastoreDefOD

표 22. *dkDatastoreDef*에 대한 구체적인 클래스 (계속)

서버 유형	클래스 이름
AS/400용 Content Manager	DKDatastoreDefV4
OS/390용 ImagePlus	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Extended Search	DKDatastoreDefDES
IBM DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
이전 Content Manager	DKDatastoreDefDL

dkDatastoreDef 인터페이스의 기본 메소드는 다음과 같습니다.

listEntities()

엔티티를 나열합니다.

listEntityAttrs()

엔티티 속성을 나열합니다.

addEntity()

엔티티를 추가합니다.

getEntity(name)

엔티티를 가져옵니다.

또한 각각의 구체적인 클래스는 자체적으로 해당 콘텐츠 서버에 익숙한 이름을 가진 콘텐츠 서버 특정 함수를 포함할 수 있습니다. 예를 들어, *DKDatastoreDefDL* 클래스는 다음과 같은 특정 함수를 포함합니다.

- *listIndexClassNames()*
- *listIndexClasses()*

DKDatastoreDefOD 클래스는 다음과 같은 특정 함수를 포함합니다.

- *listAppGrps()*
- *listAppGrpNames()*

dkEntityDef

dkEntityDef 클래스는 함수를 정의하여 다음과 같은 작업을 수행합니다.

- 엔티티 정보에 액세스
- 속성 작성 및 삭제
- 엔티티 작성 및 삭제

dkEntityDef 클래스에서 서브엔티티와 관련되는 모든 함수는 기본 콘텐츠 서버가 서브엔티티를 지원하지 않음을 나타내는 *DKUsageError*를 생성합니다. 그

리나 Domino.Doc에서처럼 콘텐츠 서버가 이러한 종류의 다중 레벨 엔티티를 지원하는 경우, 이 콘텐츠 서버의 서브클래스는 예외를 겹쳐쓰도록 적절한 함수를 구현해야 합니다.

표 23에 dkEntityDef 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 23. dkEntityDef에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKItemTypeDefDL
OnDemand	DKAppGrpDefOD
AS/400용 Content Manager	DKIndexClassDefV4
OS/390용 ImagePlus	DKEntityDefIP
Domino.Doc	DKCabinetDefDD
Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
이전 Content Manager	DKIndexClassDefDL

dkEntityDef 클래스의 기본 함수는 다음과 같습니다.

listAttrs()

엔티티 속성을 나열합니다.

getAttr(String attrName)

지정된 엔티티 속성을 가져옵니다.

addAttr(DKAttrDef)

엔티티에 속성을 추가합니다.

getName()

엔티티의 이름을 가져옵니다.

setName(String)

엔티티의 이름을 설정합니다.

hasSubEntities()

엔티티가 서브엔티티를 포함하는지 여부를 판별합니다.

getSubEntity(String)

서브엔티티를 가져옵니다.

addSubEntity(dkEntityDef)

엔티티에 서브엔티티를 추가합니다.

listSubEntities()

엔티티의 서브엔티티를 나열합니다.

removeAttr(*String*)

엔티티에서 서브엔티티를 제거합니다.

add() 콘텐츠 서버에 엔티티를 추가합니다.

update()

콘텐츠 서버에서 엔티티를 갱신합니다.

retrieve()

콘텐츠 서버에서 엔티티 값을 검색합니다.

del() 콘텐츠 서버에서 엔티티를 삭제합니다.

dkAttrDef

dkAttrDef 클래스는 속성 정보에 액세스하여 속성을 작성 및 삭제하는 함수를 정의합니다. 표 24에 dkAttrDef 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 24. dkAttrDef에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKAttrDefDICM
OnDemand	DKFieldDefOD
AS/400용 Content Manager	DKAttrDefV4
OS/390용 ImagePlus	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Extended Search	DKFieldDefDES
DB2 Universal Database	DKColumnDefDB2
JDBC	DKColumnDefJDBC
ODBC	DKColumnDefODBC
이전 Content Manager	DKAttrDefDL

dkAttrDef 클래스의 기본 메소드는 다음과 같습니다.

listAttrs()

속성을 나열합니다.

getAttr(*String attrName*)

지정된 속성을 가져옵니다.

getName()

속성의 이름을 가져옵니다.

getDescription()

속성에 대한 설명을 가져옵니다.

add() 콘텐츠 서버에 엔티티를 추가합니다.

dkServerDef

dkServerDef 클래스는 각 콘텐츠 서버에 대한 서버 정의 정보를 제공합니다.
표 25에 dkServerDef 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 25. dkServerDef에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKServerDefICM
OnDemand	DKServerDefOD
AS/400용 Content Manager	DKServerDefV4
Domino.Doc	DKServerDefDD
Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
JDBC	DKServerDefJDBC
ODBC	DKServerDefODBC
이전 Content Manager	DKServerDefDL

dkServerDef 클래스의 기본 함수는 다음과 같습니다.

setDatastore(dkDatastore ds)

콘텐츠 서버 오브젝트에 대한 참조를 설정합니다.

getDatastore()

콘텐츠 서버 오브젝트에 대한 참조를 가져옵니다.

getName()

콘텐츠 서버의 이름을 가져옵니다.

setName(String name)

콘텐츠 서버의 이름을 설정합니다.

datastoreType()

콘텐츠 서버 유형을 가져옵니다.

dkResultSetCursor

dkResultSetCursor는 DDO 오브젝트의 가상 컬렉션을 관리하는 데 사용할 수 있는 조회 결과 세트의 콘텐츠 서버 커서입니다. 이 컬렉션은 조회 결과 세트입니다. 컬렉션의 각 요소는 콘텐츠 서버가 요소를 검색할 때까지 작성되지 않습니다.

dkResultSetCursor 클래스의 기본 함수는 다음과 같습니다.

isScrollable()

커서가 앞뒤로 화면이동할 수 있는 경우, TRUE를 리턴합니다.

isUpdatable()

커서를 갱신할 수 있는 경우, TRUE를 리턴합니다.

isValid()

커서가 올바른 경우, TRUE를 리턴합니다.

isBegin()

커서가 결과 세트의 시작 부분에 위치하는 경우, TRUE를 리턴합니다.

isEnd()

커서가 결과 세트의 끝 부분에 위치하는 경우, TRUE를 리턴합니다.

isInBetween()

커서가 결과 세트의 데이터 요소 사이에 위치하는 경우, TRUE를 리턴합니다.

getPosition()

커서의 현재 위치를 가져옵니다.

setPosition(int position, Object value)

지정된 위치로 커서를 설정합니다.

setToNext()

결과 세트의 다음 요소를 가리키도록 커서를 설정합니다.

fetchObject()

결과 세트에서 현재 요소를 검색한 후 이를 DDO로서 리턴합니다.

fetchNext()

결과 세트에서 다음 요소를 검색한 후 이를 DDO로서 리턴합니다.

findObject(int position, String predicate)

지정된 술어에 부합되는 데이터 오브젝트를 찾아 커서를 해당 위치로 이동한 다음 오브젝트를 검색합니다.

addObject(DKDDO ddo)

컨텐츠 서버에 특정 DDO가 표시한 것과 동일한 유형의 새 요소를 추가합니다.

deleteObject()

컨텐츠 서버에서 현재 요소를 삭제합니다.

updateObject(DKDDO ddo)

지정된 DDO를 사용하여 컨텐츠 서버의 현재 위치에 있는 현재 요소를 갱신합니다.

newObject()

같은 유형의 요소를 작성한 후 이를 DDO로서 리턴합니다.

open() 커서를 열고 필요한 경우 조회를 실행하여 결과 세트를 작성합니다.

close() 커서 및 결과 세트를 닫습니다.

isOpen()

커서가 열려 있는 경우, TRUE를 리턴합니다.

destroy()

커서를 삭제합니다. 그러면 커서가 가비지로 수집되기 전에 커서를 제거할 수 있습니다.

datastoreName()

커서가 속하는 콘텐츠 서버의 이름을 가져옵니다.

datastoreType()

커서가 속하는 콘텐츠 서버 유형을 가져옵니다.

handle(int type)

유형별로 결과 세트 커서와 연관된 결과 세트 핸들을 가져옵니다.

요구사항: addObject, deleteObject 및 updateObject 함수를 사용하려면 콘텐츠 서버 옵션 DK_DL_OPT_ACCESS_MODE를 DK_READWRITE를 설정해야 합니다.

dkBlob

dkBlob는 기본 BLOB(2진 대형 오브젝트) 데이터 유형에 대해 공통적인 공용 인터페이스를 선언하는 추상 클래스입니다. dkBlob 클래스에서 파생된 구체적인 클래스는 이기종 콘텐츠 서버에서 생성된 BLOB 컬렉션을 다형적으로 처리할 수 있는 이러한 공통적인 공용 인터페이스를 공유합니다. 또한 구체적인 클래스를 가질 수 있는 dkClob 및 dkDBClob 클래스도 있습니다.

표 26에 dkBlob 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 26. dkBlob에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKLobICM
OnDemand	DKBlobOD
AS/400용 Content Manager	DKBlobV4
OS/390용 ImagePlus	DKBlobIP
Domino.Doc	DKBlobDD
Extended Search	DKBlobDES
DB2 Universal Database	DBBlobDB2, DKBlobDB2
JDBC	DKBlobJDBC, DKBlobJDBC
ODBC	DKBlobODBC, DKBlobODBC
이전 Content Manager	DKBlobDL

dkBlob 클래스의 기본 메소드는 다음과 같습니다.

getContent()

오브젝트의 BLOB 데이터를 포함하는 바이트 배열을 리턴합니다.

getContentToClientFile(String afileName, int fileOption)

오브젝트에서 지정된 필드로 BLOB 데이터를 복사합니다.

setContent(byte[] aByteArr)

오브젝트의 LOB 데이터를 바이트 배열의 콘텐츠로 설정합니다.

setContentFromClientFile(String afileName)

afileName 파일의 콘텐츠로 오브젝트의 LOB 데이터를 대체합니다.

add(String afileName)

컨텐츠 서버에 지정된 파일의 콘텐츠를 추가합니다.

retrieve(String afileName)

지정된 파일에서 컨텐츠 서버의 콘텐츠를 검색합니다.

update(String afileName)

지정된 파일의 콘텐츠로 오브젝트와 컨텐츠 서버를 갱신합니다.

del(boolean flush)

*flush*가 TRUE인 경우에는 컨텐츠 서버에서 오브젝트의 데이터를 삭제하고, 그렇지 않으면 현재 콘텐츠를 보존합니다.

concatReplace(dkBlob aBlob), concatReplace(byte[] aByteArr)

이 오브젝트와 다른 dkBlob 오브젝트 또는 바이트 배열을 연결합니다.

length()

오브젝트의 LOB 콘텐츠 길이를 리턴합니다.

indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)

오프셋 시작 위치에서 검색을 시작하면 이 오브젝트에서 첫 번째 발생하는 검색 인수의 바이트 오프셋을 리턴합니다.

substring(int startPos, int length)

이 오브젝트의 LOB 데이터에 대한 부속 문자열을 포함하는 문자열 오브젝트를 리턴합니다.

remove(int startPos, int aLength)

aLength 바이트의 *startPos*에서 시작하면 이 오브젝트 LOB 데이터 일부를 삭제합니다.

insert(String aString, int startPos), insert(dkBlob aBlob, int startPos)

오브젝트의 LOB 데이터에 있는 *startPos* 위치 다음에 인수 데이터를 삽입합니다.

open(String afileName)

afileName 파일로 오브젝트 콘텐츠를 로드 해제한 후 기본 파일 핸들러를 실행합니다.

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)

실행 가능 프로그램 이름으로 전체 클래스의 파일 핸들러를 표시합니다. 이 함수는 파일 오브젝트에 대해 핸들러를 동기적으로 실행할지 또는 비동기적으로 실행할지를 식별합니다.

setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)

실행 가능 프로그램 이름으로 파일 핸들러를 식별하고 이 오브젝트에 대해 이 파일 핸들러를 동기적으로 또는 비동기적으로 실행할지를 표시합니다.

getOpenHandler()

전체 클래스에 대해 파일 핸들러의 실행 가능 프로그램 이름을 가져옵니다.

isOpenSynchronous()

파일 핸들러의 현재 동기화 설정을 리턴합니다.

dkClob

dkClob는 문서와 같은 CLOB(문자 대형 오브젝트) 데이터 유형을 저장하기 위해 공용 인터페이스를 선언하는 추상 클래스입니다.

표 27에 dkClob 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 27. dkClob에 대한 구체적인 클래스

서버 유형	클래스 이름
DB2 Universal Database	DKClobDB2
ODBC	DKClobODBC

dkClob 클래스의 기본 함수는 다음과 같습니다.

open() Open()는 dkXDOBase에서 상속된 구성원입니다. Open()는 dkClob의 구체적인 서브클래스에서 구현되거나 대체됩니다.

dkXDO Members: dkXDO& add(), dkXDO retrieve(), dkXDO update(), dkXDO del()

dkXDO로부터 보호된 구성원으로 상속됩니다. 필요한 경우, 이러한 보호된 구성원은 dkClob의 구체적인 서브클래스에서 구현되거나 대체됩니다.

다음 목록은 dkClob에서 정의한 구성원을 포함합니다.

add(String afileName)

컨텐츠 서버에 지정된 파일의 컨텐츠를 추가합니다.

retrieve(String afileName)

지정된 파일에서 컨텐츠 서버의 컨텐츠를 검색합니다.

update(String afileName)

지정된 파일의 콘텐츠로 오브젝트와 콘텐츠 서버를 갱신합니다.

del(DKBoolean flush)

*flush*가 TRUE인 경우에는 콘텐츠 서버에서 오브젝트의 데이터를 삭제하고, 그렇지 않으면 현재 콘텐츠를 보존합니다.

getContentToClientFile(String afileName, int fileOption)

오브젝트에서 지정된 파일로 CLOB 데이터를 복사합니다.

setContentFromClientFile(String afileName)

afileName 파일의 콘텐츠로 오브젝트의 LOB 데이터를 대체합니다.

indexOf(String& aString, long startPos=1), indexOf(dkClob& adkClob, long startpos=1)

오프셋 시작 위치에서 검색을 시작하면 이 오브젝트에서 첫 번째 발생하는 검색 인수의 바이트 오프셋을 리턴합니다.

substring(long startpos, long length)

이 오브젝트의 LOB 데이터에 대한 부속 문자열을 포함하는 문자열 오브젝트를 리턴합니다.

remove(long startpos, long aLength)

aLength 바이트의 *startPos*에서 시작하면 이 오브젝트 LOB 데이터 일부를 삭제합니다.

insert(DKString aString, long startpos), insert(dkClob& adkClob, long startpos)

오브젝트의 CLOB 데이터에 있는 *startPos* 위치 다음에 인수 데이터를 삽입합니다.

open(String afileName)

afileName 파일로 오브젝트 콘텐츠를 로드 해제한 후 기본 파일 핸들러를 실행합니다.

setInstanceOpenHandler(String ahandler, DKBoolean newSynchronousFlag)

실행 가능 프로그램 이름으로 파일 핸들러를 식별하고 이 오브젝트에 대해 이 파일 핸들러를 동기적으로 또는 비동기적으로 실행할지를 표시합니다.

setClassOpenHandler(String ahandler, DKBoolean newSynchronousFlag)

실행 가능 프로그램 이름으로 전체 클래스의 파일 핸들러를 식별합니다. 이 함수는 파일 오브젝트에 대해 핸들러를 동기적으로 실행할지 또는 비동기적으로 실행할지를 표시합니다.

getOpenHandler()

전체 클래스에 대해 파일 핸들러의 실행 가능 프로그램 이름을 가져옵니다.

isOpenSynchronous()

파일 핸들러의 현재 동기화 설정을 리턴합니다.

dkAnnotationExt

dkAnnotationExt는 모든 주석 오브젝트의 인터페이스 클래스입니다. 콘텐츠 서버가 주석 데이터를 지원할 경우, 이 인터페이스를 구현해야 합니다. 이 주석 오브젝트는 DKBlobxx 클래스의 확장자입니다. 여기서 dkBlob 오브젝트는 2진 주석 데이터 및 DKParts 컬렉션을 나타냅니다.

dkDatastoreExt

dkDatastoreExt 클래스는 표준 콘텐츠 서버 확장자 클래스를 정의합니다.

표 28에 dkDatastoreExt 클래스에 대한 구체적인 클래스의 예제가 나와 있습니다.

표 28. dkDatastoreExt에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKDatastoreExtICM
OnDemand	DKDatastoreExtOD
AS/400용 Content Manager	DKDatastoreExtV4
OS/390용 ImagePlus	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Extended Search	DKDatastoreExtDES
DB2 Universal Database	DKDatastoreExtDB2
JDBC	DKDatastoreExtJDBC
이전 Content Manager	DKDatastoreExtDL

dkDatastoreExt 클래스의 기본 함수는 다음과 같습니다.

getDatastore()

소유하는 콘텐츠 서버 오브젝트에 대한 참조를 가져옵니다.

setDatastore(dkDatastore ds)

소유하는 콘텐츠 서버 오브젝트에 대한 참조를 설정합니다.

isSupported(String functionName)

이 확장자가 지정된 함수 이름을 지원하는지 여부를 판별합니다.

listFunctions()

이 확장자에 대해 지원되는 모든 함수 이름을 나열합니다.

addToFolder(dkDataObject folder, dkDataObject member)

이 폴더 및 콘텐츠 서버에 구성원을 추가합니다.

removeFromFolder(dkDataObject folder, dkDataObject member)

이 폴더 및 콘텐츠 서버에서 구성원을 제거합니다.

checkOut(dkDataObject item)

콘텐츠 서버에서 문서 또는 폴더를 체크아웃합니다. 항목이 체크아웃되는 동안 사용자는 항목에 대한 독점적인 갱신 권한을 갖고 있으므로 다른 사용자는 읽기 액세스만 갖습니다.

checkIn(dkDataObject item)

이전에 콘텐츠 서버에서 체크아웃된 문서나 폴더 항목을 체크인합니다. 파일을 체크인하면 이 함수가 갖고 있던 모든 쓰기 권한을 해제합니다.

getCommonPrivilege()

특정 콘텐츠 서버의 공통 사용 권한을 가져옵니다.

isCheckedOut(dkDataObject item)

콘텐츠 서버에서 문서나 폴더 항목이 체크아웃되는지 여부를 판별합니다.

checkedOutUserId(dkDataObject item)

콘텐츠 서버에서 항목을 체크아웃한 사용자 ID를 가져옵니다.

unlockCheckedOut(dkDataObject item)

콘텐츠 서버에서 항목을 잠금 해제합니다.

changePassword (String userId, String oldPwd, String newPwd)

콘텐츠 서버에서 지정된 사용자 ID에 대한 암호를 변경합니다.

moveObject (dkDataObject dataObj, String entityName)

하나의 엔티티에서 다른 엔티티로 *entityName* 오브젝트를 이동합니다.

retrieveFormOverlay(String id)

양식 오버레이 오브젝트를 검색합니다.

DKPidXDO

DKPidXDO 클래스는 콘텐츠 서버에 있는 BLOB 데이터의 지속 식별자(PID)를 나타냅니다.

표 29에 DKPidXDO 클래스에 대해 구체화된 클래스의 예제가 나와 있습니다.

표 29. DKPidXDO에 대한 구체적인 클래스

서버 유형	클래스 이름
이전 Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
AS/400용 Content Manager	DKPidXD0V4
OS/390용 ImagePlus	DKPidXDOIP
Domino.Doc	DKPidXDODD

표 29. DKPidXDO에 대한 구체적인 클래스 (계속)

서버 유형	클래스 이름
Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
JDBC	DKPidXDOJDBC
ODBC	DKPidXDOODBC

dkUserManagement

dkUserManagement 클래스는 모든 콘텐츠 서버의 사용자 관리 함수를 나타내고 처리합니다.

표 30에 dkUserManagement 클래스에 대해 구체화된 클래스의 예제가 나와 있습니다.

표 30. dkUserManagement에 대한 구체적인 클래스

서버 유형	클래스 이름
Content Manager	DKUserMgmtICM
AS/400용 Content Manager	DKUserMgmtV4
OS/390용 ImagePlus	DKUserMgmtIP
이전 Content Manager	DKUserMgmtDL

DKConstant

모든 공통 상수는 DKConstant 클래스에 정의됩니다. 각 콘텐츠 서버에는 해당 콘텐츠 서버에 고유한 상수를 정의할 수 있는 자체 DKConstantxx 클래스가 있습니다.

권장사항: 모든 콘텐츠 서버는 가능하면 공통 메시지를 사용합니다.

DKMessageId

모든 공통 메시지 ID는 이 클래스에 정의됩니다. 각 콘텐츠 서버에는 소유하는 메시지 ID를 정의할 수 있는 자체 DKMessageIdxx 클래스가 있습니다.

권장사항: 모든 콘텐츠 서버는 가능하면 공통 메시지를 사용해야 합니다.

이러한 등록 정보 파일에는 공통 경고 및 오류 메시지가 포함됩니다.

Java의 경우

- DKMessage_en.properties
- DKMessage_es.properties

C++의 경우

- DKMessage_en_US.properties
- DKMessage_es_ES.properties

각 콘텐츠 서버에는 해당 경고와 오류 메시지에 대한 자체 DKMessagexx_yy_zz.properties 파일이 있습니다.

FeServerDefBase 클래스 사용(Java 전용)

FeServerDefBase 클래스는 사용자 조정 서버 정의를 작성하기 위해 확장해야 하는 추상 클래스입니다. 이 기본 클래스를 확장하는 Java 클래스에는 다음 매개변수를 승인하여 상위 클래스로 전달하는 구성자가 있어야 합니다.

String connectString

서버용 연결 문자열.

String[] serverList

정의된 서버 목록.

String[] associatedServerList

이 서버와 연관된 서버 목록(없는 경우 널(null)).

String[] serverTypes

정의된 서버 유형 ID 목록.

String[] serverTypeDescriptions

정의된 서버 유형에 대한 설명 목록.

FeServerDefBase 클래스를 확장하는 Java 클래스를 작성하는 경우, 새 서버 대화 상자의 데이터 처리 방법을 결정해야 합니다. 동일한 클래스 또는 별도의 모델 클래스를 사용할 수 있습니다. 연결 문자열을 위해 사용자 조정 콘텐츠 서버에 더 많은 필드가 필요한 경우, 추가 기능을 제대로 수행하려면 Enterprise Information Portal 데이터베이스 및 Java API를 모델로 사용해야 합니다.

Enterprise Information Portal 관리 프로그램에서 콘텐츠 서버를 선택하면 신규 메뉴에는 Enterprise Information Portal 데이터베이스의 FASERVERTYPES 테이블에 저장된 서버 유형 목록이 포함됩니다. 이 테이블에는 메뉴 항목 선택 시 인스턴스화될 Java 클래스 이름이 들어 있습니다.

암호 확인을 지원하는 경우에는 Java 클래스를 Enterprise Information Portal 관리 .jar 파일과 동일한 디렉토리에 두어야 하며, 사용자 입력 암호를 매개변수로 사용하여 이 Java 클래스를 동적으로 인스턴스화하고 verify 메소드를 호출할 수 있습니다. verify 메소드는 암호가 올바른 경우에는 널(null)을 리턴하며 암호가 올바르지 않은 경우에는 정보와 함께 문자열 배열을 리턴합니다.

EIP 워크플로우 응용프로그램 빌드

EIP 클래스 및 API를 사용하여 EIP 워크플로우 지원을 사용할 수 있도록 응용프로그램을 작성 또는 확장할 수 있습니다. 일반적으로 연합 검색을 수행하고 검색 결과(다중 콘텐츠 항목의 항목 또는 폴더)를 가진 워크플로우를 시작합니다. API를 사용하여 작업 목록에 액세스한 다음 작업 목록 콘텐츠를 표시합니다. 각 활동이 완료되면 워크플로우는 워크플로우의 다음 활동으로 이동합니다.

워크플로우 서비스에 연결

응용프로그램에서 Enterprise Information Portal 워크플로우를 사용하려면 DKWorkflowServicesFed의 인스턴스를 작성하여 시작하십시오. 그런 다음 여기에 연결하십시오. 다음 예제에서는 워크플로우 서비스를 시작합니다.

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, "");
```

워크플로우 서비스 사용을 완료했으면 `disconnect()` 및 `delete()` 함수를 호출하여 연결을 해제해야 합니다.

Java

```
svWF.disconnect();
dsFed.disconnect();
svWF.destroy();
dsFed.destroy();
```

C++

```
svWF->disconnect();
dsFed->disconnect();
delete svWF;
delete dsFed;
```

워크플로우 시작

워크플로우를 작성한 후에는 시작해야 합니다. 워크플로우를 시작하려면 다음 단계를 완료하십시오.

1. DKWorkFlowFed 오브젝트를 작성하고 워크플로우 이름을 설정하십시오.
2. Enterprise Information Portal 워크플로우 빌더에 정의되어 있는 워크플로우 정의인 올바른 워크플로우 템플리트를 사용하여 워크플로우 인스턴스를 작성하십시오.

3. 컨테이너에서 PID 및 우선순위를 설정하십시오.

4. 워크플로우를 시작하십시오.

다음 예제에서는 다음 단계를 사용하여 워크플로우를 시작합니다.

Java

```
// ----- 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 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;
```

워크플로우 종료

다음 예제에서와 같이 `terminate()` 또는 `del()` 함수를 호출하여 워크플로우를 종료할 수 있습니다.

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;
```

모든 워크플로우 나열

listWorkFlows() 함수를 사용하여 워크플로우 서비스에 모든 워크플로우를 나열할 수 있습니다. 다음 예제에서는 DKWorkFlowServiceFed 오브젝트 svWF가 참조하는 워크플로우 서비스의 모든 워크플로우에 대한 설명 및 이름을 나열합니다.

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;
```

워크플로우 일시중단

특정 시간까지 또는 무기한으로 실행 중인 워크플로우를 일시중단할 수 있습니다. 다음 예제에서는 특정 시간까지 워크플로우를 일시중단하는 방법을 보여줍니다. 널(null) DKTimestamp를 제공하는 경우, EIP는 워크플로우를 무기한 일시중단합니다.

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;
```

워크플로우 재개

resume() 함수를 호출하여 일시중단된 워크플로우를 재개할 수 있습니다. 다음 예제에서는 일시중단된 워크플로우를 재개합니다.

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;
```

모든 작업 목록 나열

워크플로우 서비스에서 listWorkLists() 함수를 호출하여 워크플로우 서비스에 모든 작업 목록을 나열할 수 있습니다. 다음 예제에서는 DKWorkflowServiceFed 인스턴스 svWF가 참조하는 워크플로우 서비스의 모든 작업 목록에 대한 설명 및 이름을 나열합니다.

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;
```

작업 목록에 액세스

시스템 관리 클라이언트를 사용하여 작성한 작업 목록을 참조하는 DKWorkListFed 인스턴스를 작성하여 작업 목록에 액세스할 수 있습니다. 다음 예제에서는 WL0712라는 작업 목록에 액세스하여 이 작업 목록에 포함된 정보를 표시합니다.

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;
```

작업 항목에 액세스

DKWorkListFed를 작성한 후 작업 항목을 컬렉션으로서 검색할 수 있습니다. 다음 예제에서는 작업 항목을 검색합니다.

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;
    }
```

워크플로우의 항목 이동

워크플로우가 진행됨에 따라 checkOut() 및 checkIn() 함수를 사용하여 작업 항목을 한 활동에서 다음 활동으로 이동합니다. 다음 예제에서는 작업 항목 이동 방법을 보여줍니다. 작업 항목을 수행하도록 현재 할당된 워크플로우 사용자만이 작업 항목을 체크아웃 및 체크인할 수 있음에 유의하십시오.

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;
```

모든 워크플로우 템플릿 나열

listWorkflowTemplates() 함수를 호출하여 워크플로우 서비스에 모든 워크플로우 템플릿을 나열할 수 있습니다. 다음 예제에서는 DKWorkflowServiceFed 오브젝트 svWF가 참조하는 워크플로우 서비스의 워크플로우 템플릿에 대한 설명 및 이름을 나열합니다.

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;
```

조치 작성(Java 전용)

워크플로우에서 사용할 수 있는 사용자의 조치를 작성할 수 있습니다. 조치를 정의하여 이를 Enterprise Information Portal 관리의 조치 목록에 추가합니다. 조치 오브젝트(DKWorkFlowActionFed 오브젝트)를 사용하는 조치를 작성합니다. 조치 오브젝트는 특정 작업이 클라이언트 노드에서 실행되는 방법에 대해 상세한 지시사항을 기록하는

메타데이터 컨테이너입니다. 조치 오브젝트(메타데이터 컨테이너)만이 지시사항을 기록합니다. 조치 오브젝트는 조치 메타데이터에 설명한 작업을 호출하지 않습니다.

조치는 조치 목록으로 그룹화될 수 있습니다(DKWorkflowActionListFed). 워크플로우 컨테이너는 작업 항목 관련 조치 세트와 연관된 조치 목록의 이름(조치 목록의 콘텐츠가 아님)을 수행합니다. 클라이언트는 조치 목록을 검색한 후 목록의 항목(조치)을 통해 반복하고 그에 따라 반응해야 합니다. 아래의 샘플에서는 조치 및 조치 목록에 대해 작업하는 방법을 보여줍니다. 샘플에서는 다음 작업을 완료합니다.

1. 작업 항목 검색
2. 작업 항목과 함께 경로지정된 컨테이너 검색
3. 컨테이너에서 조치 목록 검색
4. 조치 목록에서 조치 목록 가져오기
5. 조치 시작

```
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");
```

비주얼 및 비주얼이 아닌 JavaBeans를 사용하여 응용프로그램 빌드

이 장에서는 Enterprise Information Portal에 제공되는 비주얼 및 비주얼이 아닌 JavaBeans에 대해 설명합니다.

EIP JavaBeans는 다음 카테고리로 분할될 수 있습니다.

비주얼이 아닌 Bean 비주얼이 아닌 Bean을 사용하여 사용자 조정 사용자 인터페이스를 필요로 하는 Java 및 웹 클라이언트 응용프로그램을 빌드할 수 있습니다. 비주얼이 아닌 Bean은 기본 구성자, 등록 정보, 이벤트 및 직렬화 가능한 인터페이스를 제공하여 표준 Bean 프로그래밍 모델을 지원합니다. 내성을 지원하는 빌더 도구에서 비주얼이 아닌 Bean을 사용할 수 있습니다.

비주얼 Bean 비주얼 Bean은 사용자 조정이 가능하고 스윙 기반인 그래픽 사용자 인터페이스 구성요소입니다. Windows용 Java 응용프로그램을 빌드하려면 비주얼 Bean을 사용하십시오. Java 기반 응용프로그램의 창과 대화 상자 내에 위치시킬 수 있습니다. 비주얼 Bean은 비주얼이 아닌 Bean(데이터 모델로서)으로 빌드되므로 응용프로그램 빌드 시 비주얼이 아닌 Bean과 함께 비주얼 Bean을 사용해야 합니다.

Java Viewer 관련 Bean Java Viewer Bean은 비주얼 및 비주얼이 아닌 구성요소 세트입니다. 그 세트를 사용하여 문서 보기 프로그램을 빌드하고 문서를 변환할 수 있습니다. Java Viewer Bean은 eClient 보기 프로그램 애플릿 및 eClient의 중간 단계에서 사용됩니다.

기본 Bean 개념 이해

JavaBeans(이하 Bean으로 사용)는 Java 프로그래밍 언어로 쓰여진 재사용 가능한 소프트웨어 구성요소로, Bean을 아는 빌더 도구를 사용하여 조작할 수 있습니다. Bean은 다시 사용할 수 있으므로 Bean을 사용하여 더 많은 복합 구성요소를 구성하고 새 응용프로그램을 빌드하거나 기존 응용프로그램에 기능을 추가할 수 있습니다. 빌더를 사용하여 위의 기능을 시각적으로 수행하거나 프로그램에서 Bean 메소드를 호출하여 수동으로 기능을 모두 수행할 수 있습니다.

Bean은 필수 Java 클래스입니다. 기존 프로그래밍 구성요소 및 Java 클래스를 Bean으로 전환할 수 있습니다. 일반적으로 등록 정보 및 이벤트 인터페이스 정의에 관한 특정 규칙을 준수하는 Java 클래스는 Bean으로 간주될 수 있습니다.

Bean은 응용프로그램 설계자 도구 또는 빌더 도구를 사용하여 디자인 시 인터페이스를 정의함으로써 구성요소가 정의하는 등록 정보 종류 및 구성요소가 생성 또는 응답하는

이벤트 종류를 판별하도록 구성요소를 조회합니다. Bean에 대한 작업 시 특정 내성 및 구성 도구를 사용해서는 안됩니다. 패턴 서명은 제대로 정의되어 있고 비주얼 점점으로 인식 및 이해하기 쉽습니다.

Bean에는 다음 특성이 있습니다.

내성 내성은 빌더 도구가 설계 및 런타임 시 Bean의 작업 방법을 결정하고 분석하도록 하는 프로세스입니다. Bean이 메소드 서명 및 클래스 정의에 대해 미리 정의된 패턴으로 코드화되므로, 이 패턴을 인식하는 도구는 Bean "내부를 살펴보고" 그 등록 정보 및 작동을 결정할 수 있습니다. 각 Bean에는 관련된 Bean 정보 클래스가 있습니다. 이 클래스는 등록 정보, 메소드 및 Bean 자체에 대한 이벤트 정보를 제공합니다. 각 Bean 정보 클래스는 BeanInfo 인터페이스를 구현합니다. 이 인터페이스는 응용프로그램 빌더 도구에 노출될 Bean 기능을 명시적으로 나열합니다.

등록 정보

등록 정보는 Bean의 모양 및 작동을 제어합니다. 빌더 도구는 Bean에서 내성하여 해당 등록 정보를 발견하고 조작에 대해 등록 정보를 노출합니다. 이렇게 하면 설계 시 Bean의 등록 정보를 변경할 수 있습니다.

사용자 조정

Bean의 노출된 등록 정보는 설계 시 사용자 조정될 수 있습니다. 사용자 조정으로 Bean의 모양 및/또는 작동을 대체할 수 있습니다. Bean은 등록 정보 편집기 또는 특수하고 세련된 Bean 사용자 조정 프로그램을 사용하여 사용자 조정을 지원합니다.

이벤트

Bean은 이벤트를 사용하여 다른 Bean과 통신합니다. Bean은 이벤트를 다른 Bean으로 전송하는 것을 의미하는 이벤트를 발행할 수 있습니다. Bean이 이벤트를 발행하는 경우, 원본 Bean으로 간주됩니다. Bean도 이벤트를 수신할 수 있습니다. 이 경우에는 리스너 Bean으로 간주됩니다. 리스너 Bean은 원본 Bean으로 이벤트에 관심을 등록합니다. 빌더 도구는 내성을 사용하여 Bean이 전송하고 수신하는 이벤트를 결정합니다.

지속성 Bean은 Java 오브젝트 직렬화를 사용하여 java.io.Serializable 인터페이스를 구현함으로써 사용자 조정의 결과로 변경될 수도 있는 상태를 저장한 후 재저장합니다. 예를 들어, 상태는 응용프로그램이 응용프로그램 빌더에 Bean을 사용자 조정한 경우 저장되므로 변경된 등록 정보가 이후 재저장될 수 있습니다.

메소드 모든 Bean 메소드는 다른 Java 클래스의 메소드와 동일합니다. Bean 메소드는 다른 Bean 또는 언어를 스크립트하여 호출될 수 있습니다. 기본적으로 모든 Bean의 공용 메소드는 내보내집니다.

빌더에서 JavaBeans 사용

이 절에서는 IBM Websphere Studio Application Developer 및 기타 빌더에서 JavaBeans를 사용하는 방법에 대해 설명합니다.

IBM Websphere Studio Application Developer 이외의 빌더를 사용하려면 빌더가 Java 2를 지원하는지를 확인하십시오. 새 jar 파일을 추가하기 위한 빌더의 지시사항을 따라 아래 지시사항에 지정된 jar을 추가하십시오. 그런 다음 jar의 Bean을 추가하기 위한 빌더의 지시사항에 따라 cmb81.jar에 있는 EIP Bean을 추가하십시오.

중요사항: Bean을 사용하려면 최소한 Java JDK 1.3.1 이상이 있어야 합니다.

CMBROOT\Samples\java 디렉토리에는 비주얼이 아닌 Bean의 코드 샘플이 들어 있습니다.

IBM Websphere Studio Application Developer 사용

다음 단계를 완료하면 Webphere Studio Application Developer에서 servlet 및 JSP 페이지를 빌드하는 데 비주얼이 아닌 Bean을 사용할 수 있습니다.

1. 웹 응용프로그램에 대한 웹 프로젝트를 작성합니다.
2. Java 빌드 경로 | 라이브러리의 프로젝트 등록 정보에서 다음 JAR 파일을 지정합니다.

\CMBROOT\lib\cmb81.jar

\CMBROOT\lib\cmbview81.jar

\CMBROOT\lib\cmbsdk81.jar

\CMBROOT\lib\esclisrv.jar

\SQLLIB\java\db2java.zip

3. EIP servlet 및 JSP taglib를 사용할 계획이면 다음 파일도 지정해야 합니다.

\CMBROOT\lib\cmbervlet81.jar

\CMBROOT\lib\cmbtag81.jar

태그 라이브러리의 경우, 웹 응용프로그램에 EIP의 JSP taglib용 taglib.tld 파일도 가져와야 합니다.

- \CMBROOT\lib>taglib.tld를 웹 응용프로그램의 webApplication\WEB-INF 디렉토리에 복사합니다.
- 다음을 추가하여 웹 응용프로그램의 webApplication\WEB-INF\web.xml 파일에서 taglib를 구성합니다.

```
<taglib>
    <taglib-uri>cmb</taglib-uri>
    <taglib-location>/WEB-INF/taglib.tld</taglib-location>
</taglib>
```

4. 위에 나열된 JAR은 J2EE 클래스를 포함하기 때문에 다음 위치에 있는 J2EE JAR을 포함시킬 필요가 있습니다. \Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\j2ee.jar

EIP Java Bean 호출

EIP 계층의 Bean은 다음 두 가지 방법 중 하나로 호출될 수 있습니다. 공용 인터페이스(공용 메소드)를 사용하여 직접 호출할 수 있습니다. 이 경우 명시적 Java 예외가 발생되어 오류 이벤트를 표시합니다.

세션 Bean에서 기능을 호출하는 또다른 방법은 요청 및 응답 이벤트를 사용하여 다른 EIP Bean에 이 유형의 Bean에 대한 인스턴스를 연결하는 것입니다. 이 방법을 사용할 때 다음 사항에 유의하십시오.

- CMBCConnection Bean은 연결 요청 이벤트를 청취하고 연결 응답 이벤트를 발행하여 응답합니다.
- CMBDataManagement Bean은 데이터 요청 이벤트를 청취하고 답으로 데이터 응답 이벤트를 발행합니다.
- CMBSchemaManagement Bean은 스키마 요청 이벤트를 청취하고 답으로 스키마 응답 이벤트를 발행합니다.
- CMBQueryService Bean은 검색 요청 이벤트를 청취하고 답으로 검색 응답 이벤트를 발행합니다.
- CMBWorkflowDataManagement Bean은 워크플로우 데이터 요청 이벤트를 청취하고 답으로 워크플로우 데이터 응답 이벤트를 발행합니다.
- CMBWorkflowQueryService Bean은 작업 목록 요청 이벤트를 청취하고 답으로 작업 목록 응답 이벤트를 발행합니다.

비주얼이 아닌 Bean

EIP는 Java 응용프로그램을 빌드하는 데 사용할 수 있는 비주얼이 아닌 JavaBeans 세트를 제공합니다. 비주얼이 아닌 Bean은 Bean 규칙을 따르는 Java 클래스 세트입니다. 이는 EIP 및 CM Java 커넥터 클래스를 사용하여 빌드됩니다. 일반적인 사용 방법은 Servlet 또는 JSP(Java Server Page)를 빌드하는 것이지만 명령행 또는 Windows 응용프로그램에서 사용될 수도 있습니다.

비주얼이 아닌 Bean 사용의 이점은 다음과 같습니다.

- EIP에서 제공하는 연합 액세스 메커니즘 및 여러 다른 커넥터에 대해 공통 프로그래밍 모델 제공
- 추상 상위 레벨에서 프로그래밍 허용
- 개별 커넥터의 복잡도 및 세부사항 숨기기

- 대부분의 상업 개발 환경에 빌드된 Bean 지원을 레버리지하도록 허용

Bean을 사용하면 기본 응용프로그램을 쉽게 빌드합니다. 그러나 사용하기 전에 알아야 할 제한사항이 몇 가지 있습니다. Bean에 대한 제한사항은 다음과 같습니다.

- Java API 계층에서 사용 가능한 모든 기능(예: 관리 또는 구성 기능)을 제공하지는 않습니다.
- 일괄처리 가져오기를 지원하지 않습니다. Bean은 일상적 단일 항목 유형 가져오기 기능만을 지원합니다. 대량의 데이터 가져오기 및 내보내기에 대한 일괄처리 프로세스는 커넥터 인터페이스를 사용하여 기록되어야 합니다.
- 전체 서버가 지원하는 기능을 모두 제공하지는 않습니다. 어떤 기능은 서버 고유 기능입니다. 예를 들어, 항목 내의 하위 항목과 관련된 기능은 Content Manager 버전 8 커넥터에서만 사용 가능한 기능과 연결됩니다.

위에 나열된 제한사항은 어떤 경우 기본 Java API의 메소드에 액세스할 수 있는 액세스 서 메소드를 사용하여 해결할 수 있습니다.

중요사항: EIP Bean은 EJB(Enterprise Java Bean)가 아닙니다. 그러므로 IBM Websphere와 같은 컨테이너가 제공한 관리 환경 내에 직접 호스트될 수 없습니다. 그러나 기본 연결 메커니즘으로서 EJB 내부에서 구조화되지 않은 데이터 저장소로 사용되었을 수 있습니다.

비주일이 아닌 Bean 구성

비주일이 아닌 Bean은 로컬, 원격 및 동적 구성을 가지고 있습니다.

로컬 컨텐츠 서버에 직접 연결합니다.

원격 RMI 서버를 사용하여 컨텐츠 서버에 연결합니다.

동적 cmbcs.ini 파일을 기준으로 로컬 및 원격 사이를 동적으로 전환하는 응용프로그램을 사용할 수 있습니다. cmbcs.ini 파일은 컨텐츠 서버의 로컬 또는 원격 여부를 지정합니다.

비주일이 아닌 Bean 기능 이해

등록 정보는 일반적으로 문자열 및 배열과 같은 단순 유형이므로 JSP의 EIP Bean을 사용할 수 있습니다. 근본적으로 MVC(Model View Controller) 설계 패턴을 사용하여 모델링되기 때문에 웹 응용프로그램에 대한 모델 구성요소로 작동합니다. 보기 구성요소는 일반적으로 JSP 및 servlet의 제어기 구성요소로 구성됩니다(EJB servlet 킷의 경우와 동일함). 다음은 EIP 비주일이 아닌 Bean 기능의 목록입니다.

- 라이브러리 서버에서 스키마 정의에 대한 액세스를 제공합니다.
- EIP가 지원하는 모든 저장소의 문서, 단순(비자원) 항목 및 자원 항목에 대해 작성, 검색, 갱신, 삭제(CRUD) 메소드를 제공합니다.

- EIP가 지원하는 모든 저장소의 문서, 단순 항목 및 자원 항목을 검색하는 기능을 제공합니다.
- 보기 가능한 형식으로 데이터 유형 변환을 지원합니다.
- EIP가 지원하는 많은 content management 저장소를 통해 의미의 일관성있는 세트를 시행하는 연합 계층으로 작동합니다.
- EIP 워크플로우 및 Information Mining 서비스에 제공되는 기능을 통합 및 노출합니다.
- 문서 주식 관리 지원뿐만 아니라 문서 추출 및 변환 서비스를 제공합니다.
- 정렬 및 변환 기능을 제공합니다.
- 연결 및 연결 해제 이벤트, 검색 결과 공고 이벤트 및 콘텐츠 변경 공고 이벤트와 같이 구성 Bean에서 발생하는 키 조치에 대해 발행되는 이벤트를 제공합니다.

비주일이 아닌 Bean 카테고리

비주일이 아닌 Bean은 다음 카테고리로 분할될 수 있습니다.

데이터스토어 Bean

이 Bean은 일반 사용자 세션에서 존재하며 사용자에게 특수한 서비스를 제공합니다. 세션 Bean은 다음을 포함합니다.

- **CMBConnection** 이 Bean은 원래의 콘텐츠 서버 또는 연합 서버일 수 있는 백엔드 서버에 대한 연결을 유지보수합니다. 이 Bean은 JavaBeans 사용에 필수입니다.
- **CMBSchemaManagement** 저장소 메타데이터로 작업하는 데 사용됩니다.
- **CMBDataManagement** 저장소 데이터로 작업하는 데 사용됩니다.
- **CMBQueryService** 및 **CMBSearchResults** 조회를 실행하고 조회 결과로 작업하는 데 사용됩니다.
- **CMBWorkflowDataManagement** 및 **CMBWorkflowQueryService** EIP 고급 워크플로우 프로세스로 작업하는 데 사용됩니다.
- **CMBDocRoutingDataManagement** 및 **CMBDocRoutingQueryService** CM v8 문서 경로지정 프로세스로 작업하는 데 사용됩니다.
- **CMBDocumentServices** 문서 스트리밍 및 주식 서비스를 제공하는 데 사용됩니다.

헬퍼 Bean

헬퍼 Bean은 하나 이상의 세션 Bean 구문에 존재하고 기본적으로 데이터 값의 캡슐화 및 세션 Bean에 서비스 제공용으로 사용됩니다. 사용 가능한 헬퍼 Bean에는 다음이 포함됩니다.

- **CMBEntity** content management 저장소에서 사용 가능한 데이터 항목 정의를 나타냅니다. 예를 들어, CM v8 저장소의 경우, CMBEntity는 항목 유

형 및 하위 구성요소 정의를 나타내지만 CM v7 저장소의 경우, **CMBEntity**는 색인 클래스를 나타냅니다. **CMBEntity**는 **CMBSchemaManagement**용 헬퍼 클래스입니다.

- **CMBAttribute** 저장소에서 속성 정의를 나타냅니다. **CMBAttribute**는 **CMBSchemaManagement**용 헬퍼 클래스입니다.
- **CMBSearchTemplate** 연합 검색 템플리트를 나타냅니다. **CMBSearchTemplate**는 **CMBSchemaManagement**용 헬퍼 클래스입니다.
- **CMBSTCriterion** 연합 검색 템플리트의 부분인 검색 기준을 나타냅니다. **CMBSTCriterion**은 **CMBSchemaManagement**용 헬퍼 클래스입니다.
- **CMBItem** 문서 인스턴스, 자원 항목 및 비자원 항목을 나타냅니다. **CMBItem**은 **CMBDataManagement**용 헬퍼 클래스입니다.
- **CMBObject** 자원 항목 인스턴스, 기본 부분 및 메모 로그 부분을 나타냅니다. **CMBObject**는 **BLOB** 속성 표시에도 사용됩니다. **CMBObject**는 **CMBDataManagement**용 헬퍼 클래스입니다.
- **CMBAnnotation** CM v8 저장소용 주석 부분 인스턴스 및 **OnDemand** 저장소용 메모를 나타냅니다.
- **CMBPrivilege** CM 또는 EIP 지원 저장소에서 사용 권한 관련 정보를 검색하는 데 필요한 기능을 제공합니다.

보조 Bean

보조 Bean은 응용프로그램에서 필수는 아니지만 기능 향상에 유용할 수 있습니다. 다음은 보조 Bean의 목록입니다.

- **CMBConnectionPool** **CMBConnection** Bean으로 풀링 서비스를 제공하는 데 사용됩니다. Java API 클래스 **DKDatastorePool**은 **CMBConnection** 인스턴스에서 사용하는 **DKDatastore** 인스턴스 유지보수에 사용됩니다. 콘텐츠 서버는 서버 유형에 기반하여 더 지능적으로 풀링될 수 있으므로 Java API 레벨로 풀링을 이동하면 콘텐츠 서버가 제대로 관리될 수 있습니다.
- **CMBUserManagement** 연합 저장소와 연결하여 원래의 서버 사용자로 연합 사용자 매핑을 관리하는 데 사용됩니다.
- **CMBExceptionSupport** 공통 예외 이벤트 처리에 대한 프레임워크를 제공합니다.
- **CMBTraceLog** 공통 추적 이벤트 처리 프레임워크 및 다른 Bean이 발행한 추적 이벤트에 대한 리스너 기능을 제공합니다.

워크플로우 Bean

워크플로우 Bean은 워크플로우 서비스를 제공합니다. EIP Bean 계층에서 제공한 워크플로우 서비스는 고급 워크플로우 및 문서 경로지정이라는 두 가지 워크플로우 시스템 유형을 지원합니다. 고급 워크플로우 기능은 **MQSeries Workflow**를 사용하여 빌드되지만 문서 경로지정은 CM V8 제품 및 API 세

트로 통합된 워크플로우 시스템입니다. Bean 계층은 워크플로우 정의를 작성하는 데 필요한 전체 오브젝트 세트를 제공하고, 작성된 정의에 기초한 워크플로우 인스턴스를 실행하며 워크플로우를 실행하는 인스턴스를 관리합니다. 다음 Bean은 고급 워크플로우 지원의 기본 구성요소입니다.

- **CMBWorkflowDataManagement** 이 Bean은 고급 워크플로우 인스턴스를 작성하고 작업하는 데 사용됩니다. 이 Bean의 인스턴스는 CMBCConnection 오브젝트에서 검색할 수 있습니다. 이 Bean은 다음에 대한 지원을 제공합니다.
 - 워크플로우 인스턴스 시작, 종료, 일시중단 및 재개
 - 한 사용자에서 다른 사용자로 작업 항목 및 작업 공고 전송
 - 작업 공고 취소
- **CMBWorkflowQueryService** CMBWorkflowQueryService는 고급 워크플로우 관련 정보 조회에 대한 인터페이스를 제공합니다. 이 Bean의 인스턴스는 CMBCConnection 오브젝트에서 검색할 수 있습니다. 이 Bean은 다음 정보 검색에 대한 지원을 제공합니다.
 - 시스템에서 워크플로우에 대한 정보.
 - 활성 워크플로우의 부분으로 시스템을 통해 이동하는 작업 항목에 대한 정보.
 - 작업 목록 관련 정보.
 - 등록된 모든 작업 공고에 대한 정보.

다음 Bean은 문서 경로지정 지원의 기본 구성요소입니다.

- **CMBDocRoutingManagementICM** 이 Bean은 문서 경로지정 프로세스를 작성하고 관리하는 데 사용됩니다. 이 Bean의 인스턴스는 CMBCConnection 오브젝트의 인스턴스에서 얻을 수 있습니다. 이 Bean은 다음 기능에 대한 지원을 제공합니다.
 - 문서 경로지정 인스턴스의 시작, 종료, 일시중단 및 재개
 - 작업 패키지 내에 포함된 CM 항목 체크아웃
 - 문서 경로지정 인스턴스에서 경로지정된 작업 패키지에 대한 등록 정보 설정
- **CMBDocRoutingQueryServiceICM** 이 Bean은 문서 경로지정 프로세스와 관련된 정보 조회에 대한 인터페이스를 제공합니다. 이 Bean의 인스턴스는 CMBCConnection 오브젝트의 인스턴스에서 얻을 수 있습니다. 이 Bean은 다음 정보 검색에 대한 지원을 제공합니다.
 - 문서 경로지정 시스템을 통해 경로지정된 작업 패키지 관련 정보
 - 시스템에서 현재 활성화된 프로세스 관련 정보
 - 시스템에 있는 모든 작업 목록에 대한 정보

- 시스템의 부분인 모든 작업 노드

Information Mining Bean

EIP Information Mining Bean은 응용프로그램을 사용하여 텍스트 분석 및 마이닝 기술을 통합할 수 있습니다. Information Mining Bean은 다음 기능을 제공합니다.

- 텍스트 요약 및 카테고리, 문서 요약 작성
- 카테고리, 문서에 카테고리 할당
- 문서에서 관련 정보 추출 지원
- 문서를 기록한 언어 식별 지원
- 문서 컬렉션에 유사한 문서 클러스터링 지원
- 카탈로그 또는 특정 카테고리의 문서로 제한된 문서의 텍스트 검색
- IBM Web Crawler를 사용하여 웹에서 지속적으로 검색된 문서에 대한 액세스 지원

Information Mining Bean은 다음 기능을 포함합니다.

- **CMBCatalogService** 카탈로그 관련 정보 검색에 대한 인터페이스를 제공합니다. Information Mining 조작에서 작성된 항목을 콘텐츠 서버로 가져오기에 대한 지원도 제공합니다. 모든 Information Mining 조작은 카탈로그로 바운드됩니다. 각 카탈로그는 차례대로 카테고리의 계층 구조로 구성된 분류와 연관됩니다.
- **CMBAdvancedSearchService** 카탈로그의 정보에서 텍스트 검색 완료에 대한 지원을 제공합니다. 이 Bean의 메소드를 사용하여 텍스트 검색에서 생성된 결과의 콘텐츠 및 크기뿐 아니라 검색이 실행되는 카탈로그를 제어할 수도 있습니다.
- **CMBCategorizationService** 이 Bean은 특정 카탈로그에 기초한 문서 카테고리 결정에 사용됩니다.
- **CMBSummarizationService** 문서 요약 생성에 필요한 기능을 제공합니다.
- **CMBClusteringService** 콘텐츠의 유사성에 기초한 클러스터로 문서를 그룹화하는 데 사용됩니다.
- **CMBWebCrawlerService** 웹 크롤러 조치의 결과를 관리하는 인터페이스를 제공합니다. 사용자가 특정 웹 공간의 웹 크롤링 요청을 시작하고 결과를 관리할 수 있습니다. 그런 다음 작성된 결과는 백엔드 콘텐츠 서버로 구분하고 요약하여 가져올 수 있습니다.

문서 서비스 Bean

문서 서비스 Bean은 문서 스트리밍 및 문서 주석 서비스를 제공합니다. 다음 Bean은 문서 서비스 하위 섹션의 부분입니다.

- **CMBDocumentServices** - **CMBDocumentServices** Bean은 하나 이상의 문서 페이지 렌더, 변환 및 재구성에 필요한 기능을 포함하여 문서 작업 시 필요한 서비스를 제공합니다.
- **CMBDocument** - 이 Bean은 **CMBDocumentServices**를 사용하여 문서 로드 시 작성된 엔티티를 나타냅니다. 본질적으로 **CMBDocument**는 문서의 페이지에 대한 컨테이너입니다. 또한 **CMBDocument**를 사용하여 문서의 특성을 제어하는 등록 정보를 조회하고 설정할 수 있습니다.
- **CMBPage** - 이 Bean은 문서의 특정 페이지에 대한 표현을 제공합니다. 이 클래스의 기능으로 페이지에 생성될 수 있는 렌더 가능한 이미지 세트의 등록 정보를 지정하고 제어할 수 있습니다.
- **CMBPageAnnotation** 이 Bean은 문서의 페이지와 연관될 수 있는 주석을 모델링합니다. 지원된 모든 주석은 이 Bean의 서브클래스로 모델링됩니다. 또한 **CMBPageAnnotation** 자체에는 주석이 있는 페이지 및 주석 유형 등의 등록 정보가 포함됩니다. 서브클래스에는 다음이 포함됩니다.
 - **CMBArrowAnnotation**
 - **CMBCircleAnnotation**
 - **CMBHighlightAnnotation**
 - **CMBLineAnnotation**
 - **CMBNoteAnnotation**
 - **CMBPenAnnotation**
 - **CMBRectAnnotation**
 - **CMBStampAnnotation**
 - **CMBTextAnnotation**

다른 Bean 클래스

이 절에 나열된 Bean 클래스는 아래 설명된 다양한 기능을 제공합니다.

- **BeanInfo** 클래스 **BeanInfo** 인터페이스를 구현하는 별도의 연관된 클래스의 EIP Bean 기능에 대한 명시적 노출을 허용합니다.
- **예외 클래스** Bean 계층에서 예외를 캡슐화하는 데 사용됩니다. 모든 예외 클래스는 기본 클래스 **CMBException**에서 계승됩니다. **CMBException**의 각 서브클래스는 특정 오류 상태를 나타냅니다. 각각의 경우에 예외 오브젝트의 등록 정보는 예외가 발생한 오류 상태의 상세한 오류 정보를 얻는 데 사용될 수 있습니다. Bean이 이벤트 구동 방식으로 사용될 경우, 예외는 이벤트 내에 발생합니다.
- **이벤트 및 리스너 클래스** 표준 Bean 이벤트 리스너 모델을 구현합니다. 클래스는 이벤트와 연관된 리스너 인터페이스를 구현하고 이벤트를 생성하는 오브젝트로 리스너 인터페이스를 등록하여 명시적으로 이벤트를 요청해야 합니다.

다. EIP Bean 계층은 스키마 액세스 조작, 데이터 액세스 조작, 워크플로우 조작 및 검색 조작에 대한 이벤트 및 리스너 쌍을 제공합니다.

- 세션 리스너 이는 세션 레벨에 존재하는 리스너 클래스입니다. EIP Bean에 현재 연결 요청 및 연결 응답 이벤트를 추적하는 세션 리스너가 있습니다.

비주일이 아닌 Bean 사용 시 고려사항

비주일이 아닌 Bean을 사용하면 EIP에서 지원하는 content management 저장소 액세스에 필요한 기능이 있는 범용 응용프로그램을 사용할 수 있습니다. 이 절에는 Bean의 특정 사용법 패턴에 대한 몇 가지 팁이 있습니다.

Bean의 Singleton CMBConnection에는 다른 세션 EIP Bean에 대한 인스턴스 액세스를 얻는 메소드가 있습니다. CMBSchemaManagement 및 CMBDataManagement와 같은 세션 Bean을 이 방법으로 얻을 경우, 연결 또는 연결 해제된 것으로 알려지고 추적 및 예외 이벤트 핸들러를 공유하는 CMBConnection Bean(얻은 위치에서)에 이미 연결되어 있습니다. 각 다른 세션 Bean의 단일 인스턴스만이 작성됩니다. 이 메소드를 반복해서 호출할 경우, 동일한 인스턴스가 리턴됩니다(singleton 설계 패턴). 세션 Bean이 응용프로그램에 작성되고 CMBConnection Bean으로는 작성되지 않은 경우, 사용할 CMBConnection Bean에 연결되어야 합니다.

Bean에서 스레딩 고려사항

CMBConnection Bean의 단일 인스턴스는 언제든지 단일 스레드에만 사용될 수 있습니다. 이 제한사항은 CMBConnection Bean(연관 Bean의 연결 등록 정보를 통해서)에 연관된 다른 모든 Bean을 확장합니다. 이는 각 스레드에 대한 개별 연결을 작성해야 함을 의미합니다. 대신, 여러 스레드는 CMBConnectionPool Bean을 사용하여 연결을 얻고 해제할 수 있습니다. 따라서 각 스레드는 연결을 얻고 사용한 후 해제해야 합니다.

모든 세션 Bean은 검색되거나 작성 후 연관된 CMBConnection의 인스턴스에 유사성이 있습니다. 이는 세션 Bean의 인스턴스(예: CMBSchemaManagement)가 정해진 시기에 단일 스레드에서만 사용될 수 있음을 의미합니다. 세션 Bean 인스턴스가 여러 스레드에서 사용되는 경우, 응용프로그램에서 명시적 동기화를 수행하여 단일 스레드만이 정해진 시기에 세션 Bean 인스턴스를 사용하고 있음을 확인해야 합니다.

또한 모든 세션 Bean은 CMBConnection Bean이 생성한 연결 응답 이벤트를 청취합니다. 그러면 CMBConnection Bean 인스턴스와 연관된 기본 콘텐츠 저장소가 변경되었음을 인식하므로 그 Bean이 적절한 조치를 수행할 수 있습니다.

CMBConnection과는 달리 CMBConnectionPool Bean은 여러 스레드 용도로 설계됩니다. 여러 스레드는 연결 오브젝트를 얻고 해제하는 것과 관련된 메소드를 동시에 호출할 수 있습니다. 풀에서 얻은 연결은 CMBConnection의 인스턴스이고 단

일 스레드 액세스로 제한됩니다. 연결 풀 Bean에서 얻은 연결은 사용 후 가능한 빨리 풀로 리턴되어야 하므로 풀에서 연결을 요청할 수도 있는 다른 스레드에 사용할 수 있습니다.

Bean에서 추적 및 로그 기록

EIP Bean 계층에서 모든 세션 Bean에 대한 추적이 사용 가능합니다. CMBConnection Bean의 인스턴스에 대한 추적을 사용 가능하게 하면 스키마 관리, 데이터 관리, 조회 서비스 및 워크플로우 Bean을 포함하여 이 연결 Bean에서 얻은 EIP Bean에 대한 추적도 사용 가능합니다.

추적이 사용 가능한 경우, 추적 이벤트가 발행됩니다. 유틸리티 Bean인 CMBTraceLog는 추적 이벤트를 청취할 수 있고 추적 레코드를 정의된 로드, stdout, stderr 또는 창(비주얼 Bean으로 사용 시)에 기록할 수 있습니다.

모든 세션 Bean도 추적 이벤트를 청취합니다. Bean의 추적 기능은 log4j가 로그 기록에 사용되는 경우 Java API와 동일한 로그 파일에 로그 기록 정보를 기록합니다.

비주얼이 아닌 Bean에 대한 등록 정보 및 이벤트 이해

각 비주얼이 아닌 Bean은 다음 사항을 제공합니다.

- 가져오기 등록 정보, 거부 가능 또는 거부 불가
등록 정보 값은 런타임 시 PropertyChange 또는 VetoableChange 이벤트에 따라 다른 Bean에 의해 결정됩니다. 가져오기 등록 정보가 있는 Bean은 PropertyChange 또는 VetoableChange 이벤트를 청취해야 합니다.
- 내보내기 등록 정보, 거부 가능 또는 거부 불가
비주얼이 아닌 Bean은 제한된 등록 정보를 가질 수 있으며 일부 다른 Bean은 해당 값과 관련되어 있습니다. 값을 변경할 때마다 이 Bean은 PropertyChange 또는 VetoableChange 이벤트를 생성해야 합니다.
- 독립형 등록 정보
다른 Bean은 이 등록 정보 값과 관련이 없습니다.
- 이 Bean에서 생성한 이벤트
- 이 Bean이 관련된 이벤트

비주얼이 아닌 Bean을 사용하여 응용프로그램 빌드

샘플 GUI(Graphical User Interface)가 아닌 응용프로그램

이 절의 예제에서는 비주얼이 아닌 Bean을 사용하여 샘플 GUI가 아닌 응용프로그램을 작성합니다. 샘플 응용프로그램은 CMBUserManagement Bean을 제외한 모든 Bean

을 포함합니다. 이 예제가 수행된 완전한 샘플 응용프로그램(DemoSimpleApp1.java)은 Cmbroot/Samples/java/beans 디렉토리에 있습니다. 샘플 응용프로그램에서는 다음과 같은 작업의 수행 방법을 보여줍니다.

1. Enterprise Information Portal (연합) 서버에 연결
2. 검색 템플리트 이름 목록 얻기
3. 검색 템플리트 이름을 사용하여 검색 기준 이름 얻기
4. 검색 템플리트를 선택하여 검색 기준 얻기
5. 검색 값 완료 및 조회 제출
6. 검색 결과 Bean을 사용하여 결과 인쇄
7. 결과 행 선택 및 표시
8. 서버 연결 해제

비주얼 Bean에 대한 작업

비주얼 Bean을 사용하여 Enterprise Information Portal 기능 또는 다른 콘텐츠 서버를 스윙 기반 Java 응용 프로그램에 통합할 수 있습니다. 비주얼 Bean은 로그인, 검색, 결과 표시 및 보기, 문서 갱신, 버전 정보 보기 등 많은 응용 프로그램에 공통적인 기본 작업을 수행합니다.

각 비주얼 Bean에는 연결 등록 정보가 있습니다. 이 등록 정보는 콘텐츠 서버에 대한 연결을 유지보수하는 비주얼이 아닌 Bean인 CMBConnection의 인스턴스를 참조해야 합니다. EIP 비주얼 Bean을 사용하여 빌드된 모든 응용프로그램도 CMBConnection 비주얼이 아닌 Bean의 인스턴스를 포함해야 합니다.

CMBLogonPanel

이 Bean은 Enterprise Information Portal 또는 Content Manager 버전 8.2(CM 8.2)와 같은 콘텐츠 서버에 로그인하는 패널을 표시합니다. 또한 연합 사용자가 콘텐츠 서버의 사용자 ID 및 암호를 수정할 수 있는 창을 제공합니다.

CMBSearchResultsViewer

이 Bean은 검색 결과를 표시합니다. 검색 결과가 폴더를 리턴하면 CMBSearchResultsViewer Bean을 사용하여 폴더로 "드릴다운"한 다음 해당 콘텐츠를 보십시오. Windows 탐색기 양식 창에서 검색 결과 또는 폴더의 항목을 선택한 다음 열어 내용을 볼 수 있습니다.

CMBSearchTemplateList

검색 템플리트를 지원하는 서버에서 이 Bean은 사용 가능한 검색 템플리트 목록을 표시하고 템플리트를 선택할 수 있도록 합니다.

CMBSearchTemplateViewer

검색 템플리트를 지원하는 서버에서 이 Bean은 검색 템플리트를 표시하고 사용자가 검색 기준을 입력할 수 있도록 필드를 제공합니다. 이 기준을 기반으로 검색을 수행합니다.

CMBSearchPanel

모든 서버에서 검색 패널은 사용 가능한 엔티티 목록을 표시하고 사용자가 검색 기준을 입력할 수 있도록 필드를 제공합니다. 이 기준을 기반으로 검색을 수행합니다. CMBSearchPanel은 검색 템플리트를 지원하지 않는 콘텐츠 서버에서 검색을 수행하는 데 유용합니다.

CMBFolderViewer

Windows 탐색기 양식 창에서 하나 이상의 폴더 콘텐츠를 표시합니다.

CMBItemAttributesEditor

사용자가 항목의 색인화 속성 및 색인 클래스를 갱신할 수 있는 창을 표시합니다.

CMBDocumentViewer

해당 보기 프로그램을 시작하여 하나 이상의 문서를 표시합니다.

CMBVersionsViewer

버전화가 사용되는 경우 문서의 버전 정보를 표시합니다.

CMBLogonPanel Bean

CMBLogonPanel Bean(그림 20 참조)은 사용자가 콘텐츠 서버에 로그인하고, 사용자 맵핑을 갱신하고, 암호를 변경할 수 있는 창을 표시합니다.



그림 20. CMBLogonPanel Bean 창

CMBLogonPanel Bean에서 사용자가 **변경**을 누르면 **암호 변경** 창이 나타납니다(471 페이지의 그림 21 참조). 사용자는 이전 암호를 입력하고 새 암호를 두 번 입력합니다.

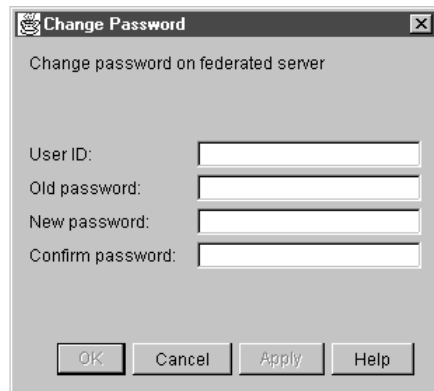


그림 21. 암호 변경 창

CMBLogonPanel Bean에서 사용자가 로그인 창의 맵핑 갱신을 누르면 사용자 ID 맵핑 갱신 창이 표시됩니다(그림 22 참조). 맵핑을 갱신하면 서버에 지정된 사용자 ID 및 암호를 갱신합니다. 이 기능은 Enterprise Information Portal 연합 데이터베이스에 로그인할 때만 사용 가능합니다.

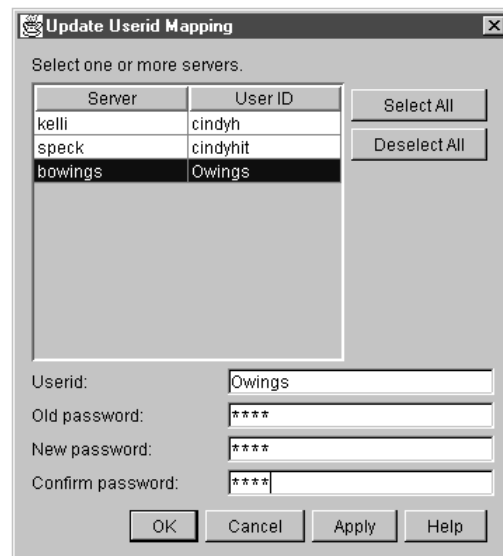


그림 22. 사용자 ID 맵핑 갱신 창

창의 맨 위에는 모든 서버 및 해당 사용자 ID 목록이 있습니다. 사용자는 이 목록에서 하나 이상의 서버를 선택할 수 있습니다. 모든 서버를 선택하려면 전체 선택을 누르십시오. 사용자는 하나 이상의 서버를 선택한 후 새 사용자 ID 및 (선택적으로) 암호를 지정할 수 있습니다. 하나의 서버를 선택하는 경우, 사용자 ID는 사용자 ID 필드에 나타납니다. 둘 이상의 사용자 ID를 선택하는 경우, 사용자 ID 필드는 공백입니다.

전체 선택 취소

모든 서버 선택을 제거합니다.

- 적용 창을 닫지 않고 맵핑 및 암호 변경을 적용할 경우에 누르십시오.
- 확인 변경사항을 승인하고 창을 닫을 경우에 누르십시오.
- 취소 변경하지 않고 창을 닫을 경우에 누르십시오.

CMBSearchTemplateList Bean

CMBSearchTemplateList Bean은 세 가지 양식을 가지고 있습니다. 그림 23에서와 같이 이미지 양식은 선택한 항목의 배경에 대해 하나의 이미지를 사용하고 선택하지 않은 항목에 대해 다른 이미지를 사용합니다. 그림 24에서는 간단한 템플릿 목록 양식을 보여줍니다. 그림 25에서는 드롭다운 템플릿 목록 양식을 보여줍니다.

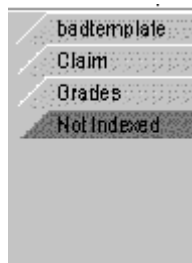


그림 23. 이미지 템플릿 목록 양식



그림 24. 간단한 템플릿 목록 양식

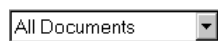


그림 25. 드롭다운 템플릿 목록 양식

CMBSearchTemplateViewer Bean

CMBSearchTemplateViewer Bean(473 페이지의 그림 26 참조)은 시스템 관리자가 정의한 검색 템플릿에 따라 검색 기준을 지정할 수 있는 창을 표시합니다. CMBSearchTemplateViewer Bean은 검색을 시작하고 CMBSearchResults 이벤트를 생성하여 검색 결과를 리턴합니다.

그림 26. *CMBSearchTemplateViewer Bean*

CMBSearchTemplateViewer Bean은 원본 또는 사용자 ID와 같은 검색 기준을 나열합니다. 각 검색 기준에는 레이블, 연산자 드롭다운 상자 및 텍스트 필드가 있습니다. BETWEEN 또는 NOTBETWEEN 연산자 화면에는 두 개의 텍스트 필드가 있습니다. IN 또는 NOTIN 연산자 화면에는 여러 행으로 구성된 텍스트 영역이 있습니다. 각 값은 서로 다른 행에 입력해야 합니다.

텍스트 검색 영역

CMBSearchTemplateViewer Bean에는 사용자가 전체 텍스트 또는 색인 속성에서 검색을 수행할 수 있도록 하는 영역이 포함될 수 있습니다. 템플릿의 전체 텍스트 검색 영역은 레이블이 있는 텍스트 필드만큼 단순할 수 있습니다.

텍스트 필드에 조회 문자열을 입력할 때 사용자는 텍스트 검색 또는 논리 텍스트 검색에 대해 조회 구문규칙을 일치시켜야 합니다(DKDatastoreTS 클래스 참조). 세부사항은 온라인 API 참조서를 참조하십시오.

CMBSearchTemplateViewer의 필드 유효성 확인 또는 편집

CMBSearchTemplateViewer Bean에 대해 유효성 확인 논리를 제공하여 사용자가 입력한 검색 기준을 수정할 수 있습니다. CMBTemplateFieldChangedEvent에 핸들러를 제공하여 이 작업을 수행하십시오. 검색 기준의 현재 값은 이 이벤트를 호출하기 전에 getTemplate 메소드에서 리턴한 CMBTemplate에 저장됩니다. 검색 기준을 검토 또는 변경할 수 있습니다. 이벤트 처리가 완료되면 새 값이 표시됩니다.

CMBSearchPanel Bean

CMBSearchPanel Bean은 현재 콘텐츠 서버에서 사용 가능한 엔티티에 따라 검색 기준을 지정할 수 있는 창을 표시합니다. CMBSearchPanel Bean은 검색을 시작하고 검색 결과를 리턴할 CMBSearchResultsEvent를 생성합니다. CMBSearchPanel은 창의 맨 위에 있는 드롭다운 목록에 사용 가능한 모든 엔티티를 나열합니다. 엔티티를 선택하면 CMBSearchPanel에 엔티티 속성이 표시됩니다. 각 속성에는 레이블, 연산자 드롭다운 상자 및 텍스트 필드가 있습니다. BETWEEN 또는 NOTBETWEEN와 같이 범위 연산자 표시에는 두 개의 텍스트 필드가 있습니다. IN 또는 NOTIN 연산자와 같

이 여러 값이 있는 연산자에는 여러 행으로 구성된 텍스트 영역이 있습니다. 각 값은 여러 행으로 구성된 텍스트 영역에서 서로 다른 행에 입력해야 합니다.

CMBSearchResultsViewer Bean

CMBSearchResultsViewer Bean은 트리 분할 영역 및 자세히 분할 영역이 있는 창에 검색 결과를 표시합니다. 사용자는 분할 영역을 구분하는 선을 누른 다음 끌어 창의 크기를 조정할 수 있습니다.

그림 27에 검색 결과 폴더가 선택되어 있는 CMBSearchResultsViewer Bean이 나와 있습니다.

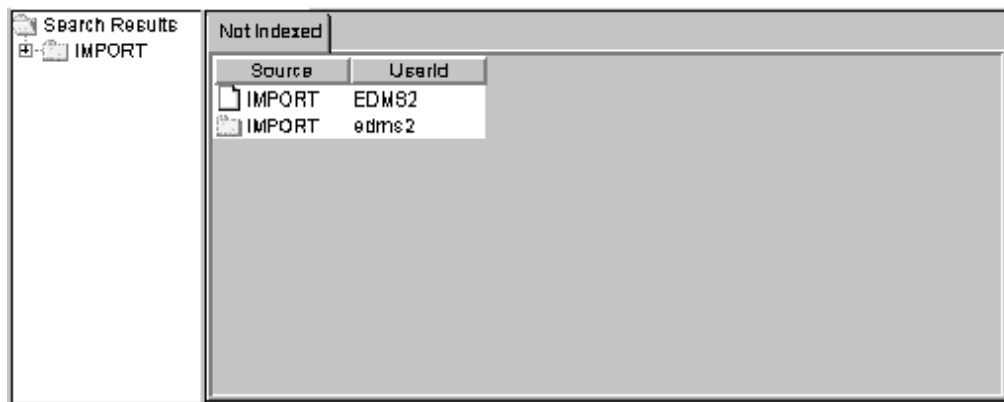


그림 27. CMBSearchResultsViewer Bean

CMBSearchResultsViewer 트리 분할 영역

왼쪽의 트리 분할 영역에는 검색 결과라는 기본 폴더가 들어 있습니다. 이 폴더 아래에 검색에서 찾은 각 폴더가 있습니다. 트리 분할 영역은 선택적입니다. TreePaneVisible 등록 정보 `setTreePaneVisible(false)`을 설정하여 제거하십시오.

CMBSearchResultsViewer 자세히 분할 영역

자세히 분할 영역에는 트리 분할 영역에서 선택한 폴더의 콘텐츠가 표시됩니다. 사용자가 검색 결과 폴더를 선택하면 검색 템플릿 이름이 포함된 노트북에 탭이 나타납니다. 사용자가 검색 결과 내에서 다른 폴더를 선택하면 폴더의 각 색인 클래스에 해당하는 하나 이상의 탭이 표시됩니다. 탭 이름은 다음 양식을 가지고 있습니다.

index class @ server

여기서 *index class*는 색인 클래스 또는 항목 유형 이름이고 *server*는 콘텐츠 서버 이름입니다. 테이블 열은 색인 클래스 또는 항목 유형에 따라 속성을 표시하도록 변경됩니다. 자세히 분할 영역에서는 다중 선택이 지원됩니다.

MultiSelectEnabled 등록 정보 `setMultiSelectEnabled(false)`를 설정하여 다중 선택을 해제하십시오. 항목 유형이 계층 구조인 경우, 하위 속성 값은 하

위 구성요소 이름/속성 이름과 같은 양식의 열 헤더가 있는 테이블에 표시됩니다. 여기서 하위 구성요소 이름은 해당 하위 구성요소의 이름이고 속성 이름은 하위 구성요소의 속성 이름입니다. 예를 들어, Journal 항목 유형이 Author 하위 구성요소를 가지고 있고, Author 하위 구성요소가 Last Name 속성을 가지고 있으면 열 헤더는 Author/Last Name과 같이 표시됩니다.

팝업 메뉴

테이블 열 표제를 마우스 오른쪽 단추로 누르면 정렬 옵션을 제공하는 팝업 메뉴가 나타납니다. 오름차순 정렬을 눌러 테이블의 항목을 오름차순으로 정렬합니다. 내림차순 정렬을 눌러 항목을 내림차순으로 정렬합니다. 트리 분할 영역에서 검색 결과 폴더 외의 폴더를 마우스 오른쪽 단추로 누르거나 자세히 분할 영역에서 문서나 폴더를 마우스 오른쪽 단추로 누르면 다른 팝업 메뉴가 나타납니다. 이 팝업 메뉴를 통해 트리 분할 영역의 폴더 세부사항을 보거나 폴더 속성을 편집할 수 있습니다.

선택적: CMBSearchResultsViewer Bean 내에서는 폴더의 세부사항을 표시하는 대신 CMBViewFolderEvent를 사용하십시오. CMBFolderViewer Bean에 선택한 폴더의 콘텐츠를 표시하려면 이벤트를 사용하십시오.

두 번 누르기 조치

트리 분할 영역에서 폴더를 두 번 누르거나 자세히 분할 영역에서 항목을 두 번 누르면 보기 팝업 메뉴 항목에서 누르는 것과 동일한 조치를 수행합니다. 기본 항목 팝업 메뉴를 표시하지 않으면 CMBItemActionEvent가 발생합니다.

팝업 메뉴 대체

CMBSearchResultsViewer 및 CMBFolderViewer의 팝업 메뉴를 다른 팝업 메뉴로 대체하거나 팝업 메뉴를 표시하지 않도록 할 수 있습니다. 기본 메뉴를 표시하지 않으려면 setDefaultPopupMenu(false)를 사용하십시오.

트리 분할 영역에서 폴더를 마우스 오른쪽 단추로 누르면 CMBFolderPopupEvent가 생성됩니다. 자세히 분할 영역에서 항목을 마우스 오른쪽 단추로 누르면 CMBItemPopupEvent가 생성됩니다. 핸들러를 사용하여 다른 팝업 메뉴를 제공할 수 있습니다.

CMBFolderViewer Bean

CMBFolderViewer Bean은 CMBSearchResultsViewer Bean과 유사한 트리 분할 영역을 표시합니다. 이 영역에는 기본 검색 결과 폴더가 없습니다. 476 페이지의 그림 28에 CMBFolderViewer Bean의 트리 및 자세히 분할 영역이 나와 있습니다.

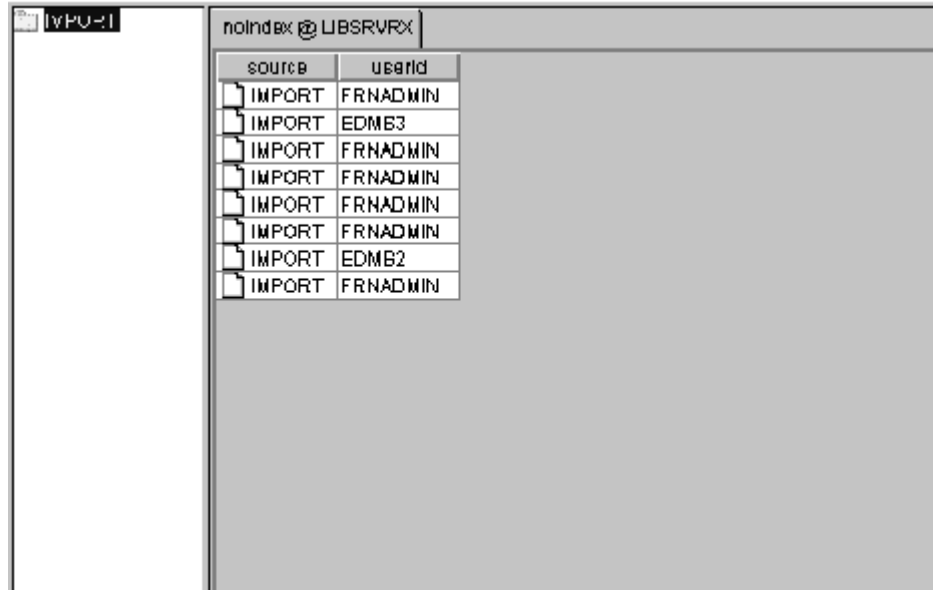


그림 28. CMBFolderViewer Bean

CMBFolderViewer Bean은 왼쪽 분할 영역에 폴더 트리를 표시합니다. 오른쪽 분할 영역에는 트리 분할 영역에서 선택한 폴더에 포함된 문서에 있는 노트북 테이블이 표시됩니다. 크기를 조정할 수 있는 분할기가 트리 및 노트북 분할 영역을 구분합니다.

CMBFolderViewer 트리 분할 영역

트리 분할 영역에는 폴더가 포함됩니다. 중첩된 폴더는 각 폴더 아래에 나타납니다.

CMBFolderViewer 자세히 분할 영역

자세히 분할 영역에는 트리 분할 영역에서 선택한 폴더의 콘텐츠가 들어 있습니다. 이 콘텐츠는 각 엔티티(색인 클래스, 항목 유형 또는 기타) 및 서버에 대한 탭과 함께 테이블의 항목이 아래에 색인화된 노트북에 표시됩니다. 탭 이름에는 색인 클래스 이름이 *index class*이고 서버 이름이 *server*인 *index class @ server* 양식이 있습니다. 각 노트북 페이지 안에는 선택한 폴더 내의 문서 및 폴더를 표시하는 테이블이 있습니다. 테이블 열은 색인 클래스에 따라 속성을 표시하도록 변경됩니다.

팝업 메뉴

폴더 보기 프로그램에 대한 팝업 메뉴의 작동은 검색 결과 보기 프로그램에서의 작동과 동일합니다.

두 번 누르기 조치

폴더 보기 프로그램에서의 두 번 누르기 조치는 검색 결과 보기 프로그램에서의 두 번 누르기 조치와 동일합니다.

CMBDocumentViewer Bean

CMBDocumentViewer Bean은 콘텐츠 유형별 문서 보기 프로그램을 시작하거나 병합하여 문서를 볼 수 있는 기능을 제공합니다. 지원되는 보기 프로그램에는 두 가지 유형이 있습니다.

1. Java 기반 보기 프로그램. 이러한 보기 프로그램은 CMBJavaDocumentViewer 클래스를 확장해야 합니다.
2. 비Java 보기 프로그램. 특정 콘텐츠 유형의 경우 모든 실행 파일을 보기 프로그램으로 시작할 수 있습니다.

비주얼 등록 정보가 false로 설정되어 있는 경우, 보기 프로그램은 항상 별도의 창에 표시됩니다. 비주얼 등록 정보가 true인 경우, 보기 프로그램은 가능하면

CMBDocumentViewer Bean의 표시 영역 내에 표시됩니다.(현재는 Java 기반 보기 프로그램의 경우에만 가능합니다.)

CMBJavaDocumentViewer는 CMBDocumentViewer Bean에 플러그인 Java 기반 문서 보기 프로그램 제공자가 확장한 추상 클래스입니다. 이러한 보기 프로그램은 문서를 CMBDocumentViewer Bean의 비주얼 공간 또는 화면의 별도의 창에 표시할 수 있습니다.

모든 문서 닫기 이벤트가 처리될 때까지 CMBDocumentViewer terminate()에 대한 호출이 대기합니다. 문서 닫기 이벤트 핸들러 내부로부터 terminate()를 호출하면 교착 상태가 발생하고 프로그램이 정지될 수 있습니다. 이 문제점을 피하려면 onDocumentClosed(CMBDocumentClosedEvent) 이벤트 핸들러 내부로부터 terminate()를 호출할 때 SwingUtilities.invokeLater(Runnable)를 사용하여 CMBDocumentViewer.terminate() 메소드를 호출하십시오. 그러면 terminate() 호출이 이벤트 대기열 끝에 추가되고, 종료 메소드를 호출하기 전에 대기열의 다른 이벤트(예: 기타 문서 닫기 이벤트 처리)를 계속합니다.

보기 프로그램 스펙

두 가지 방법으로 보기 프로그램을 지정할 수 있습니다.

1. EIP 관리에서 응용프로그램 연관 편집기에 대한 MIME 유형을 사용하여 보기 프로그램을 지정합니다. 이는 도구 메뉴에서 **MIME to Appl. Editor**를 선택하면 선택됩니다. Java 기반 보기 프로그램의 경우, 응용프로그램 이름은 **.class** 접미부가 포함된 Java 클래스 이름이어야 합니다. 실행 파일의 경우, 응용프로그램 이름은 실행 파일의 이름이어야 합니다.
2. CMBDocumentViewer에서 Mime2App 등록 정보를 사용하여 보기 프로그램을 지정합니다. 이 등록 정보는 MIME 유형을 응용프로그램 이름으로 맵핑하는 등록 정보 오브젝트의 인스턴스로 설정될 수 있습니다.

EIP 관리와 Mime2App 등록 정보를 사용할 때 모두 MIME 유형에 대해 보기 프로그램을 지정한 경우, Mime2App를 사용한 스펙이 우선합니다.

기본 보기 프로그램

특정 콘텐츠 유형에 대해 보기 프로그램을 지정하지 않으면 기본 보기 프로그램이 시작됩니다. OnDemand의 문서의 경우에는 OnDemand 클라이언트(보기 전용 모드)가 시작됩니다. 다른 모든 콘텐츠 서버의 문서는 Content Manager 보기 프로그램을 사용하여 볼 수 있습니다. 주석을 편집하려면 보기 프로그램의 "파일" 메뉴에서 "문서 편집"을 선택하십시오.

외부 보기 프로그램 시작

특정 MIME 유형의 문서에 대해 문서 보기 프로그램으로 시작할 응용 프로그램을 지정하려면 CMBDocumentViewer의 Mime2App 등록 정보를 사용하십시오. setMime2App를 등록 정보 오브젝트와 함께 값(실행 파일 이름임)에 매핑하는 MIME 유형 이름을 갖는 인수로 사용하십시오.

CMBItemAttributesEditor Bean

CMBItemAttributesEditor Bean(그림 29 참조)은 문서 또는 폴더의 색인화 속성 및 색인 클래스를 보고 변경할 수 있는 창을 표시합니다.

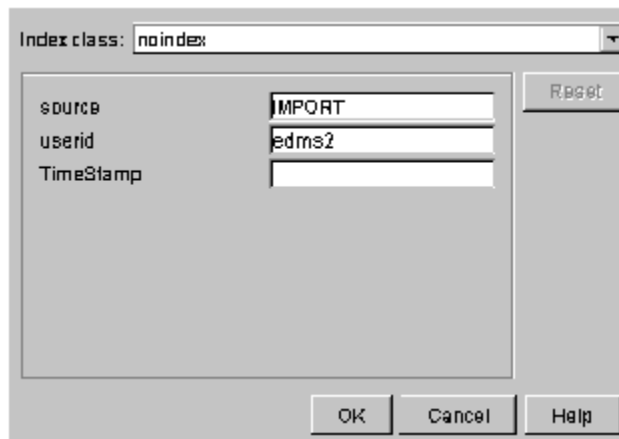


그림 29. CMBItemAttributesEditor Bean

사용 가능한 모든 엔티티를 포함하는 목록이 창의 맨 위에 나타납니다. 현재 엔티티는 기본적으로 선택됩니다. 해당 엔티티의 속성 목록은 엔티티 아래에 나타납니다. 텍스트 필드(성 및 이름 등)는 초기에 항목의 현재 값을 포함합니다.

사용자가 새 엔티티를 선택하는 경우, 이전에 선택한 엔티티와 이름이 동일한 속성은 새 엔티티에서 이름이 동일한 속성으로 값을 전달합니다.

재설정을 누르면 엔티티 및 속성을 원래 값으로 리턴합니다.

확인을 누르면 엔티티 및 속성을 갱신하며 갱신 전후에 이벤트를 트리거합니다. 갱신 전에 이벤트를 사용하여 필드의 유효성을 확인하거나 갱신을 수행하기 전에 누락된 필드를 완료할 수 있습니다. 이 이벤트는 지정된 갱신을 거부할 수 있습니다.

CMBItemAttributesEditor의 변경 거부

값이 올바르지 않은 경우 CMBItemAttributesEditor에 대해 사용자가 입력한 속성값을 확인하고 수정하거나 갱신을 거부하는 추가 유효성 확인 논리를 제공할 수 있습니다. CMBEditRequestedEvent에 핸들러를 제공하여 이 작업을 수행하십시오.

CMBVersionsViewer Bean

CMBVersionsViewer Bean은 단일 문서 또는 항목에 대한 버전화 속성 테이블을 표시합니다. 표시된 버전화 속성에는 버전화 번호, 작성자의 사용자 ID, 버전이 작성된 시간 소인, 마지막 갱신자의 사용자 ID 및 마지막 갱신 시간 소인 등이 있습니다. 버전 보기 프로그램에서 항목의 다른 버전을 보거나 항목의 속성을 갱신할 수 있습니다.

비주얼 Bean을 위한 일반 작동

다음 절에서는 비주얼 Bean 사이에 공통인 등록 정보 및 작동에 대해 설명합니다.

등록 정보

이 절에서는 비주얼 Bean에서 공유하는 세 가지 등록 정보에 대해 설명합니다.

연결 각 Bean은 CMBCConnection 비주얼이 아닌 Bean의 인스턴스를 참조하는 연결 등록 정보를 가지고 있습니다. 비주얼 Bean이 올바르게 작동하려면 연결 등록 정보를 설정해야 합니다.

CollationStrength

정렬을 수행하는 모든 Bean은 CollationStrength 등록 정보를 가지고 있습니다. CollationStrength 등록 정보에 정의된 값은 Java의 java.text.Collator 클래스에 정의된 값과 동일합니다.

단추 숨기기/표시

모든 비주얼 Bean에 나타나는 단추를 숨기거나 표시할 수 있습니다. *setnameButtonVisible* 등록 정보를 사용하십시오. 여기서 *Name*은 누름 단추의 이름입니다.

구성 저장/복원

CMBSearchTemplateViewer, CMBSearchResultsViewer 및 CMBFolderViewer는 응용프로그램 세션 간에 필드값 및 열 크기를 저장 및 복원하는 데 사용될 수 있는 두 가지 메소드(loadConfiguration 및 saveConfiguration)를 가지고 있습니다. 등록 정보 오브젝트는 이러한 모든 메소드의 인수입니다. 세 가지 Bean 모두에 대해 동일한 등록 정보 오브젝트를 사용할 수 있습니다. 저장된 등록 정보의 이름은 Bean 전체에서 고유합니다.

도움말 이벤트

사용자가 도움말을 요청하면 각 비주얼 Bean은 도움말 단추 또는 F1을 눌러 CMBHelp 이벤트를 생성합니다. 사용자가 2차 창에서 F1 또는 도움말을 누르면 일부 Bean은 다음과 같은 도움말 관련 이벤트를 생성합니다.

CMBChangePasswordHelpEvent

암호 변경 창에서 도움말을 누르는 경우

CMBUpdateMappingHelpEvent

맵핑 갱신 창에서 도움말을 누르는 경우

CMBLoginFailedHelpEvent

서버 로그인 실패 창에서 도움말을 누르는 경우

CMBServerUnavailableHelpEvent

서버 사용 불가능 창에서 도움말을 누르는 경우

참: 이러한 모든 소스에서 도움말을 처리하는 방법 중 하나는 이러한 모든 이벤트에 대해 리스너를 구현하는 단일 클래스를 작성하는 것입니다. onHelp 메소드 내에서 이벤트 원본인 Bean을 결정하고 이 Bean에 적합한 도움말 텍스트를 표시하기 위해서는 추가 논리가 필요합니다.

비주얼 Bean 대체

비주얼 Bean 중 하나를 다른 Bean 또는 스윙 구성요소와 대체할 수 있습니다. 이렇게 하려면 새 Bean이 대체할 비주얼 Bean의 이벤트에 대한 핸들러를 구현해야 합니다. 최소한 대체할 Bean의 주요 이벤트도 생성해야 합니다. 주요 이벤트가 표 31에 설명되어 있습니다.

표 31. 비주얼 Bean 및 주요 이벤트

비주얼 Bean	주요 이벤트
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent
CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Event
CMBItemAttributesEditor	없음

Bean 기능 구현에 필요한 모든 데이터는 Bean이 처리 중인 이벤트 또는 CMBConnection 비주얼이 아닌 Bean에서 사용 가능합니다.

비주얼 Bean을 사용하여 응용프로그램 빌드

비주얼 Bean을 사용하여 기록된 샘플 클라이언트 응용프로그램이 제공됩니다. 샘플의 원본 파일은 <cmbrout>/samples/java/beans/gui에 있습니다. 샘플 클라이언트 및 설정 요구사항에 대한 세부사항을 보려면 이 디렉토리의 readme.html 파일을 읽어야 합니다.

다음 절에서는 응용프로그램 빌드 시 비주얼 Bean이 함께 작동하는 방식을 보여줍니다.

비주얼 Bean 연결

이 절에서는 간단한 응용프로그램을 작성하기 위해 비주얼 Bean을 연결하는 시나리오에 대해 설명합니다. 검색 단추를 제외한 모든 Bean은 목표 Bean을 표시된 원본 Bean 이벤트의 리스너로 추가하여 연결됩니다. 예를 들어, SearchTemplateList를 SearchTemplateViewer에 연결하려면 한 행의 코드가 필요합니다. 검색을 시작하는 단추를 추가하려면 표준 JButton을 사용하십시오. 단추의 조치 이벤트가 해당 메소드를 호출하도록 내부 클래스를 작성하십시오.

그림 30에서 각 Bean에서 연결 Bean으로의 선은 Bean이 연결 Bean에 대한 참조를 포함하고 있음을 나타냅니다. 이는 각 Bean에 대해 연결 등록 정보를 설정하여 작성됩니다. 예를 들어, 로그인 패널 Bean에서 연결 Bean으로의 참조를 작성하려면 한 행의 코드가 필요합니다.

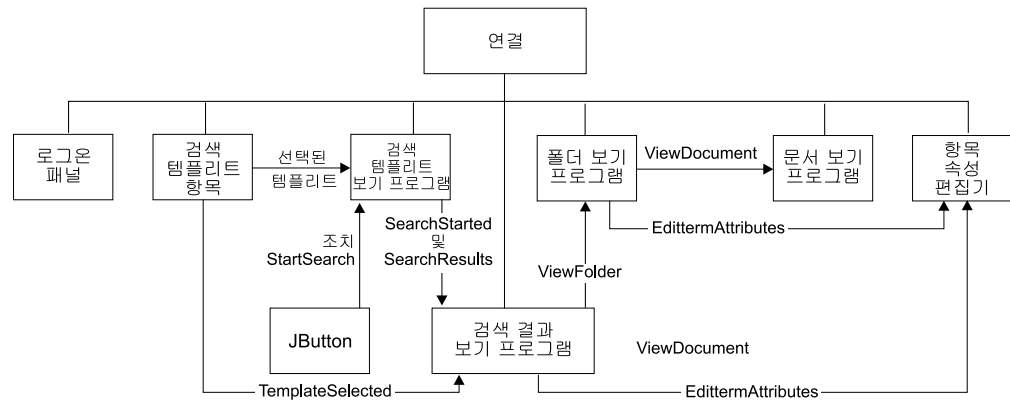


그림 30. 비주얼 Bean 연결

그림 30에 9개의 Bean이 나와 있습니다. JFrame 또는 기타 컨테이너 Bean은 이러한 모든 Bean의 상위이 될 수 있습니다. 런타임 중 가능한 이벤트 순서는 다음과 같습니다.

1. 사용자가 로그인 창에 사용자 ID 및 암호를 입력한 다음 확인을 누릅니다. CMBLogonPanel Bean이 CMBCConnection Bean의 connect 메소드를 호출하여 서버에 대한 연결을 설정합니다.

2. 연결 Bean이 연결을 설정합니다. CMBSearchTemplateList Bean은 해당 사용자 ID에 대한 검색 템플릿 목록을 검색 및 표시합니다.(이 작업을 수행하기 위해 메소드를 호출할 필요는 없습니다. CMBSearchTemplateList Bean이 CMBConnection Bean의 해당 이벤트를 청취합니다. CMBSearchTemplateList는 연관된 CMBConnection Bean이 setConnection 메소드를 사용하는 경우 리스너를 설정합니다.)
3. 사용자는 목록에서 검색 템플릿을 선택합니다. CMBSearchTemplateList Bean은 CMBTemplateSelectedEvent를 생성합니다. CMBSearchTemplateViewer 및 CMBSearchResultsViewer는 모두 이벤트를 청취합니다.
CMBSearchTemplateViewer는 해당 템플릿을 표시합니다.
CMBSearchResultsViewer는 템플릿에서 정의된 대로 자세히 분할 영역의 열을 지운 다음 표시합니다.
4. 사용자는 템플릿을 완료하고 Enter 또는 검색을 누릅니다. 사용자가 검색을 누르면 조치 이벤트 핸들러가 startSearch 메소드를 호출합니다. 사용자가 Enter를 누르면 startSearch 메소드가 암시적으로 호출됩니다.
5. CMBSearchTemplateViewer Bean은 템플릿 필드의 유효성을 확인하여 검색 시작 여부를 결정합니다. 검색을 시작할 수 있으면 CMBSearchStartedEvent가 생성됩니다. CMBSearchResultsViewer는 CMBSearchStartedEvent를 청취하고 새 검색 결과에 대비하여 결과를 지웁니다.
6. 검색이 진행됨에 따라 CMBSearchResultsEvents가 생성되고 CMBSearchResultsViewer에 부분 검색 결과를 제공합니다.(검색을 완료하면 CMBSearchCompleted 이벤트가 생성됩니다. 이 이벤트는 검색 단추가 검색 시작 시 사용 불가능인 경우 다시 사용 가능하도록 하는 데 사용될 수 있습니다.)
7. 사용자는 검색 결과 창에서 폴더를 확장한 다음 보고자 하는 문서 또는 폴더를 선택할 수 있습니다. 이 작업을 완료하면 CMBViewFolderEvent 또는 CMBViewDocumentEvent가 생성됩니다.CMBFolderViewer 및 CMBDocumentViewer Bean은 해당 이벤트를 각각 청취하고 폴더 또는 문서를 표시합니다.
8. 사용자는 CMBFolderViewer에서 볼 문서를 선택할 수 있습니다. 볼 문서를 선택하면 CMBViewDocumentEvent가 생성됩니다. CMBDocumentViewer는 이 이벤트를 청취하고 해당 보기 프로그램에 문서를 표시합니다.
9. 사용자는 CMBSearchResultsViewer 또는 CMBFolderViewer에서 갱신할 문서 또는 폴더의 속성을 선택할 수 있습니다. 문서를 선택하면 CMBEditItemAttributesEvent가 생성됩니다.
10. CMBItemAttributesEditor Bean은 CMBEditItemAttributesEvent를 청취합니다. 이 Bean은 항목의 엔티티 및 속성을 표시합니다. 사용자는 엔티티 및 속성을 변경하고 확인을 눌러 변경사항을 적용할 수 있습니다.

둘 이상의 창 또는 대화 상자에서 Bean 사용

한 창의 Bean에서 다른 창의 Bean으로 이벤트를 전달하려면 추가 코드를 제공해야 합니다. 일반적으로 이벤트를 전송하면 창이 표시됩니다. `EditAttributesDialog` 창은 `ItemAttributesEditor`를 포함합니다. `CMBEditItemAttributesEvent`가 시작되면 `SearchFrame`은 창을 작성합니다.

```
// 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);
    }
});
```

대개 `CMBItemAttributesEditor` Bean으로 전달되는 정보는 창의 구성자에 인수로 전달됩니다. 구성자 내에서 다음 등록 정보를 설정하여 정보를 `CMBItemAttributesEditor` Bean으로 전달합니다.

```
itemAttributesEditor.setConnection(connection);
```

```
itemAttributesEditor.setItem(item);
```

Java 문서 보기 프로그램 툴킷에 대한 작업

문서 보기 프로그램을 사용하면 콘텐츠 서버에 포함된 문서에 액세스하고 주석을 처리할 수 있습니다. EIP Java 보기 프로그램 툴킷을 사용하면 사용자 조정 문서 보기 프로그램을 작성할 수 있습니다. 또한 사용자 조정 보기 프로그램 애플릿 및 응용프로그램을 작성하여 EIP 또는 독립형 응용프로그램에 통합할 수 있습니다.

중요사항: *viewdata* 옵션은 OnDemand 백엔드를 지원하지 않습니다.

Java 보기 프로그램 툴킷 클래스에는 다음과 같은 기능을 제공하는 조치 오브젝트가 포함됩니다.

- 페이지 보기 옵션
 - 문서 회전: 시계방향으로 90도, 시계 반대 방향으로 90도, 180도
 - 확대/축소
 - 스케일: 25%, 50%, 100%, 150%, 200% 및 400%
- 역순
- 확장
- 인쇄
- 현재 문서 닫기
- 모든 문서 닫기
- 주석 작성 및 편집
 - 쓰기
 - 강조표시
 - 상자 그리기
 - 원 그리기
 - 선 그리기
 - 화살표 그리기
 - 텍스트 추가
 - 소인
 - 메모 추가
 - 지우기
 - 숨기기 또는 표시
 - 기본 커서 모드
 - 주석 앞으로

- 주식 뒤로
- 주식 등록 정보 변경
- 주식 조작 실행 취소 및 재실행
- 주식 자르기, 복사, 붙이기 및 삭제
- 문서 저장(주식만 저장)
- 문서 또는 문서 내 페이지 탐색
 - 페이지 탐색: 첫 번째 페이지, 이전 페이지, 페이지 찾아가기, 다음 페이지, 마지막 페이지
 - 문서 탐색: 첫 번째 문서, 이전 문서, 문서 찾아가기, 다음 문서, 마지막 문서
- 썸네일
 - 썸네일 숨기기 또는 표시
 - 썸네일을 사용하여 페이지 탐색
 - 회전 및 확대/축소

Java 보기 프로그램 툴킷은 스윙 기반 응용프로그램 빌드에 사용할 수 있는 많은 GUI 클래스를 제공합니다. 또한 스윙 기반이 아닌 문서 보기 응용프로그램에 사용할 수 있는 비 GUI 클래스도 제공합니다.

보기 프로그램 아키텍처

Java 보기 프로그램 툴킷에는 문서 보기 프로그램 및 문서 서비스 Bean이 들어 있습니다. Bean은 보기 프로그램을 EIP 기반 응용프로그램에 통합하는 메커니즘을 제공합니다.

툴킷에는 일반 문서 보기 프로그램, 스트리밍 문서 서비스, 주식 서비스 클래스도 들어 있습니다. 콘텐츠 스토어에 대한 연결이 로컬이 아닌 분산 프로세스 상황이나 독립형 보기와 같이 Bean을 사용할 수 없는 응용프로그램에서 클래스를 통해 보기 프로그램을 사용할 수 있습니다.

스트리밍 문서 서비스는 문서를 구문 분석하고, 페이지를 렌더하며, 문서 페이지를 조작하는 기능을 제공하는 문서 처리 엔진 세트를 관리합니다. 엔진은 텍스트, Rich Text 문서 및 오피스 형식뿐 아니라 TIFF 및 IOCA와 같은 다양한 형식의 구문 분석 문서 이미지를 제공합니다. 또한 추가 문서 엔진을 작성하고 이를 보기 프로그램 툴킷 아키텍처로 플러그인하여 추가 형식 및 문서 형식의 대체 렌더링을 지원할 수 있습니다.

주식 서비스 클래스를 사용하면 주석을 조작할 수 있습니다. 주식 엔진은 Content Manager 고유 주식 형식을 구문 분석합니다. 추가 주식 엔진을 작성할 수 있습니다.

문서 엔진

EIP에 제공되는 문서 엔진에는 다음 네 가지가 있습니다.

- MS-Tech 문서 엔진: 이 엔진은 IBM Content Manager에서 일반적으로 발견되는 콘텐츠 유형을 처리하여 페이지를 이미지로 렌더합니다. 지원되는 문서 유형에는 TIFF, MO:DCA가 있습니다.이 엔진은 또한 GIF, JPEG 및 일반 텍스트를 지원합니다.
- INSO 문서 엔진: 이 엔진은 Microsoft Office, Lotus SmartSuite 및 기타 사무용 문서 형식을 지원합니다.
- AFP2Web 문서 엔진: 이 엔진은 AFP를 이해하고 AFP 문서를 HTML 또는 PDF로 변환합니다.
- Java 문서 엔진: 이 엔진은 URL 문서를, URL로 링크를 전달하는 HTML로 변환합니다. 또한 이 엔진은 XSLT를 호출하여 XML을 HTML로 변환합니다.

엔진은 공용 인터페이스이지만, 엔진에 직접 프로그래밍해서는 안됩니다. 대신, Document Services Bean이나 Streaming Doc Services 클래스가 제공하는 인터페이스를 사용하십시오.

일부 엔진은 순수 Java가 아니며 이식성 제한사항을 가지고 있습니다. 따라서 일부 플랫폼에서 툴킷 사용에 제한이 있을 수 있습니다. MS-Tech 및 Java 엔진은 순수 Java입니다. 다른 엔진에는 Windows 플랫폼에만 사용할 수 있도록 제한된 플랫폼 특정 논리가 들어 있습니다.

주석 엔진

EIP는 Content Manager 주석을 처리하도록 MS-Tech 주석 엔진을 제공합니다. MS-Tech 엔진은 Content Manager 버전 8.2, Content Manager 버전 8.1, Content Manager 버전 7 및 VI/400 주석을 지원합니다.

일반 문서 보기 프로그램 작성

일반 문서 보기 프로그램을 작성하려면 CMBStreamingDocServices 인터페이스를 사용하는 CMBGenericDocViewer 클래스와 주로 작업합니다. CMBStreamingDocServices는 문서를 로드 및 렌더합니다. CMBStreamingDocServices는 문서 엔진 세트를 사용하여 TIFF 및 MO:DCA와 같은 서로 다른 문서 형식을 변환합니다. 문서의 주석을 로드, 편집 및 저장하려면 CMBAnnotationServices 인터페이스를 사용하십시오.

일반 문서 보기 프로그램 사용자 조정

cmbview81.jar 파일에 위치한 기본 구성 파일 CMBViewerConfiguration.properties를 사용자 조정하거나 새 구성 파일을 작성할 수 있습니다. 구성 파일을 작성하면 CMBViewerConfiguration.properties를 사용자 조정하면, 파일 이름을 동일하게 유지하고 클래스 경로의 cmbview81.jar 앞에 놓아야 합니다.

구성 파일을 작성하려면 다음 단계를 완료하십시오.

1. 도구 모음 이름으로 지정된 도구 모음마다 다음 항목을 지정해야 합니다. 이 단계는 기본 프레임에서 도구 모음 위치를 지정합니다. 기본 위치는 NORTH입니다. 나열된 각 도구 모음의 위치를 지정하십시오.

```
Toolbars=[<toolbar_name>[,<toolbar2_name>][,<toolbar3_name>]...]
<toolbar_name>.position={NORTH|SOUTH|EAST|WEST}
```

2. 지정된 도구 모음에 추가할 조치를 지정하십시오. 도구 모음의 조치 사이에 분리 문자를 삽입하려면 단어 'separator'를 사용하십시오.

```
<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. 새 팝업 메뉴를 추가할 수 없습니다. 그러나 미리 정의된 세 가지 팝업 메뉴에 서브메뉴 및 메뉴 항목을 추가할 수 있습니다.

```
<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>
```

해당 구성 파일을 지정한 경우, 독립형 보기 응용프로그램 또는 애플릿을 빌드할 준비가 된 것입니다. 온라인 API 참조서를 참조하여 다음 단계를 완료하십시오.

1. CMBStreamingDocServicesCallbacks를 구현하는 개인용 클래스를 작성하십시오.
2. 첫 번째 단계에서 구현되는 콜백으로 CMBStreamingDocServices를 작성하십시오.
3. CMBAnnotationServicesCallbacks를 구현하는 개인용 클래스를 작성하십시오.
4. 구현되는 콜백으로 CMBAnnotationServices를 작성하십시오.
5. CMBGenericDocViewer의 인스턴스를 작성하고 CMBStreamingDocServices, CMBAnnotationServices 및 구성 등록 정보 파일을 사용하여 초기화하십시오. 기본 구성 파일을 원하면 등록 정보 파일에 대한 널(null)을 전달하십시오.
6. CMBGenericDocViewer에서 loadDocument()를 호출하여 문서를 로드하십시오. CMBDocument 인스턴스가 리턴됩니다.
7. CMBGenericDocViewer에서 loadAnnotationSet()을 호출하여 주석을 로드하십시오. CMBAnnotationSet 오브젝트가 리턴됩니다. CMBAnnotationServices에서 setItemHandle(CMBAnnotationSet, CMBItem) 메소드를 사용하십시오.
8. 위치, 썸네일 크기, MDI 또는 SDI 보기 등을 위한 보기 프로그램이 제공하는 여러 가지 메소드를 사용하여 보기 프로그램의 모양 및 느낌을 사용자 정의하십시오.
9. 보기 프로그램을 응용프로그램 또는 애플릿의 기본 프레임에 추가하십시오.

10. 일반 문서 보기 프로그램으로부터 조치를 검색하고 메뉴를 응용프로그램 기본 프레임에 추가하여 메뉴 막대를 준비하십시오.
11. 일반 문서 보기 프로그램에서 showDocument()를 호출하여 문서를 표시하십시오.
12. 문서에 주석을 저장하려면 CMBGenericDocViewer에서 saveAnnotations(CMBDocument)를 호출하십시오.
13. 한 문서를 닫으려면 closeDocument(CMBDocument)를 호출하고 모든 문서를 닫으려면 closeAllDocuments()를 호출하십시오.

그림 31은 모든 기본 설정을 사용하는 일반 문서 보기 프로그램의 예제입니다.



그림 31. 일반 문서 보기 프로그램

응용프로그램 예제

Java 문서 보기 프로그램 툴킷에 대한 이해를 돕기 위해 이 절에서는 작성할 수 있는 다섯 가지 응용프로그램 예제를 제공합니다. 아래 예제에 나와 있는 방식 외에도 여러 가지 방식으로 툴킷을 사용할 수 있습니다.

독립형 보기 프로그램

일반 문서 보기 프로그램을 사용하여 독립형 보기 프로그램을 구현할 수 있습니다. 독립형 보기 프로그램을 사용하여 URL에서 검색한 파일이나 문서를 볼 수 있습니다. 예를 들어, 웹 페이지의 애플릿으로 독립형 보기 프로그램을 사용하거나 전자 우편으로 받은 문서를 보는 데 독립형 보기 프로그램을 사용할 수 있습니다. 그림 32에서는 독립형 보기 프로그램의 아키텍처에 대해 설명합니다.

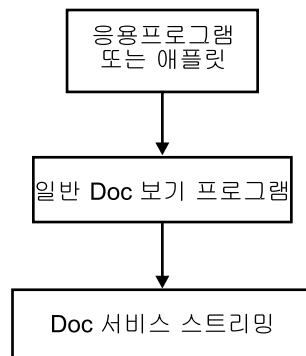


그림 32. 독립형 보기 프로그램

Java 응용프로그램

프로덕션 Java 응용프로그램을 작성하려면 CMBDocumentViewer 비주얼 Bean을 사용하는 EIP 비주얼 Bean을 사용하십시오. 이 Bean은 내부적으로 일반 문서 보기 프로그램을 사용하여 문서를 표시합니다. 또한 CMBDocumentViewer Bean은 다른 보기 프로그램을 시작하여 문서를 볼 수도 있습니다. 그러나 이러한 보기 프로그램은 플랫폼에 종속된다는 점을 기억하십시오. 491 페이지의 그림 33에서는 Java 응용프로그램을 빌드하는 데 사용할 수 있는 아키텍처에 대해 설명합니다.

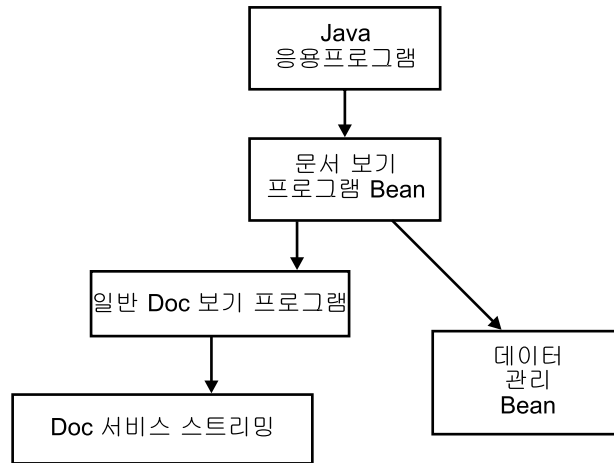


그림 33. Java 응용프로그램

Thin 클라이언트

CMBDocumentServices Bean을 사용하여 웹 기반 응용프로그램에서 서버측 문서 변환을 수행할 수 있습니다. 브라우저에서 처리하지 않는 콘텐츠 유형(플러그인이나 원래의 응용프로그램 시작이 필요한 문서)에서 브라우저가 기본적으로 처리할 수 있는 콘텐츠 유형(예: HTML, GIF, JPEG 또는 플러그인을 즉시 사용할 수 있는 PDF)으로 문서를 변환할 수 있습니다. 그림 34에서는 thin 클라이언트 아키텍처에 대해 설명합니다.

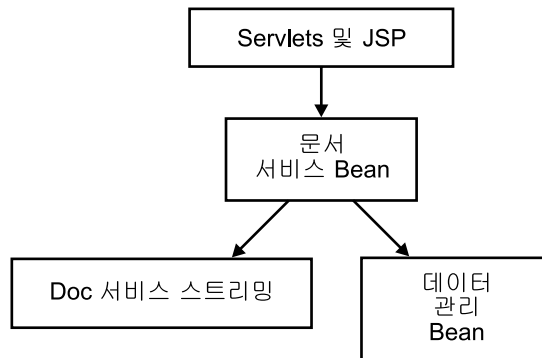


그림 34. Thin 클라이언트

애플릿 또는 servlet

또한 웹 기반 응용프로그램은 애플릿 또는 servlet 접근 방법을 사용한 문서 보기 및 주석 편집 기능을 제공합니다. eClient 보기 프로그램 애플릿은 이 아키텍처를 사용합니다. 애플릿은 일반 문서 보기 프로그램을 사용하여 문서를 볼 수 있습니다. 일부 문서 유형은 콘텐츠 서버의 여러 부분에 저장되어 백그라운드 양식과 같은 공통 정보를

보다 효율적으로 공유할 수 있도록 합니다. 필요한 경우 일반 문서 보기 프로그램은 추가 부분을 요청합니다. 보기 프로그램이 들어 있는 애플릿은 servlet으로 HTTP 요청을 전송하여 요청을 충족시킵니다. servlet측에서 볼 때, CMBDataManagement Bean을 사용하는 콘텐츠 서버가 요청 정보를 얻습니다. 그림 35에서는 애플릿 또는 servlet 아키텍처에 대해 설명합니다.

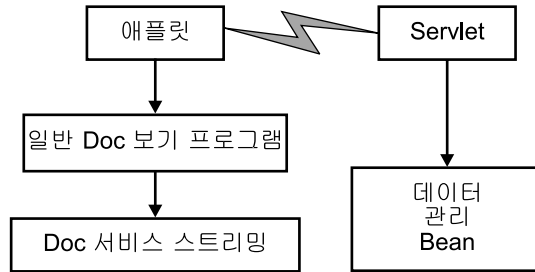


그림 35. 애플릿 또는 servlet

이중 모드 및 애플릿 또는 servlet

제공된 예제를 변형하여 웹 기반 보기 응용프로그램에 적용할 수 있습니다. 서버 및 애플릿에서 CMBDocumentServices를 사용하십시오. 문서가 애플릿으로 렌더되지 않고 서버에서 변환된 경우에는 이 접근 방법이 유용합니다. 애플릿과 서버에서 기본 문서 엔진이 서로 다른 기능을 가지는 경우에 해당합니다.

예를 들어, 서버가 Windows NT 또는 2000이고 애플릿이 OS/2에서 실행되는 경우, 애플릿은 모든 문서 유형을 렌더하지 못할 수도 있습니다. 애플릿은 TIFF와 같이, 지원되는 문서 형식을 렌더합니다. Office 형식과 같이 애플릿이 렌더할 수 없는 유형의 경우, servlet으로부터 변환을 요청합니다. 응용프로그램 사용자의 관점에서 볼 때, 동일한 기능을 가진 동일한 인터페이스가 제공됩니다. 그러나 서버에서 변환된 문서의 서버 성능은 로컬로 렌더된 문서에 비해 느릴 수 있습니다.

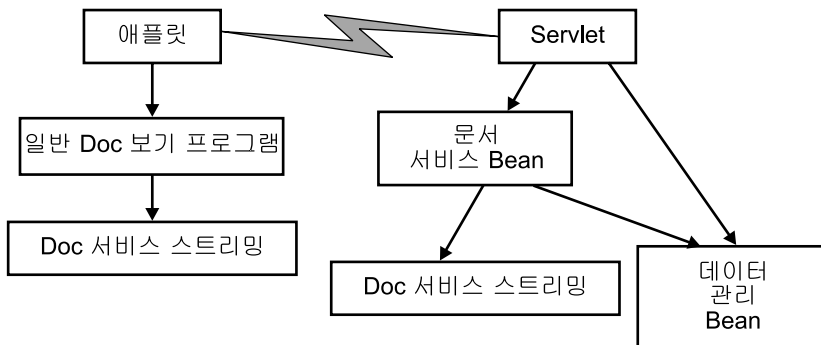


그림 36. 이중 모드 보기 프로그램

그림 38에서는 주석 서비스 클래스 도표를 보여줍니다

주석 엔진을 구현하려면 추상 클래스 CMBAnnotationEngine을 확장해야 합니다. 주석 엔진은 CMBAnnotationServicesCallbacks 및 CMBAnnotationEngineCallbacks 인터페이스를 사용하여 응용프로그램 및 주석 서비스와 통신합니다. EIP 8.1의 주석 엔진은 Content Manager 주석 형식만 이해합니다. 이 주석 형식은 Content Manager 버전 7, Content Manager 버전 8. 1 및 VI/400 백엔드에서 사용됩니다.

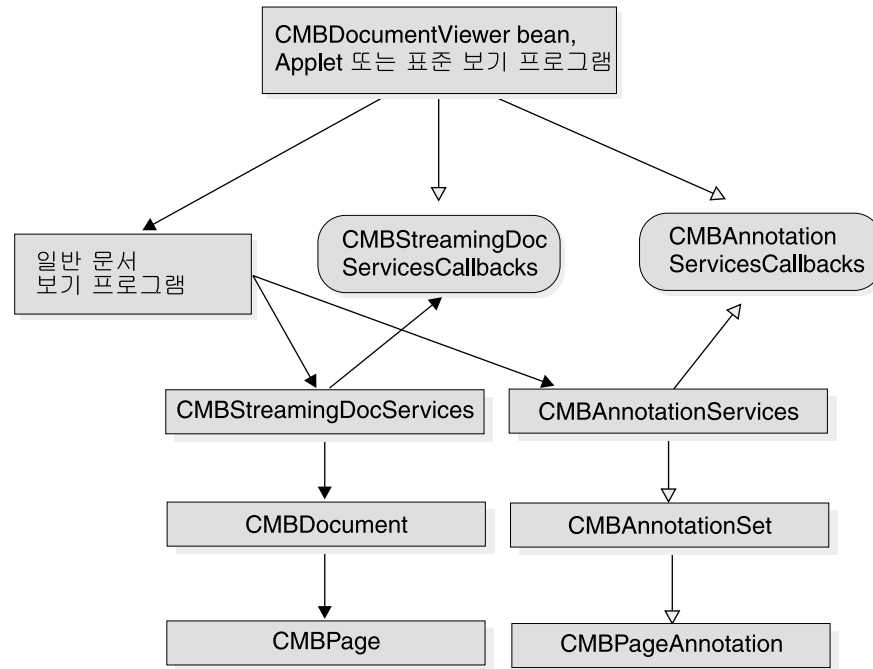


그림 38. 주석 서비스 클래스 도표

주석 편집 자원의 이해

Model View Controller(MVC) 설계 패턴은 주석 편집 기능을 구현하는 데 사용됩니다. MVC는 주석 데이터를 나타내는 모델처럼 작동합니다. CMBAnnotationSet에는 데이터를 조작하는 메소드가 있지만 사용자 인터페이스는 없습니다. CMBAnnotationSet 클래스는 CMBPageAnnotation 오브젝트 목록을 유지보수합니다. 각 문서는 해당 주석을 나타내는 CMBAnnotationSet 오브젝트와 연관되어 있습니다. CMBAnnotationView는 모델에서 사용자에게 데이터를 제공하는 보기로 작동합니다. CMBAnnotation은 보기 구성요소(JComponent)에서 모든 주석 그리기를 처리합니다. CMBAnnotationComponent는 주석이 그려진 보기 구성요소로 사용할 수 있는 헬퍼 클래스입니다. 제어기는 보기 프로그램 툴킷의 내부에 있으며 주석 작성 및 편집에 대한 마우스 및 키보드 이벤트를 처리합니다.

CMBPageAnnotation은 문서 페이지의 단일 주석을 설명하는 기본 클래스입니다. 그래픽 주석 유형을 추가로 정의해야 하는 경우에는 CMBPageAnnotation 클래스를 확장해야 합니다. Content Manager 주석 유형은 CMBArrowAnnotation, CMBCircleAnnotation, CMBHighlightAnnotation, CMBLineAnnotation, CMBNoteAnnotation, CMBPenAnnotation, CMBRectAnnotation, CMBStampAnnotation 및 CMBTextAnnotation과 같이 9가지 유형으로 구분됩니다.

주: 비 GUI 응용프로그램에서 주석 서비스를 사용할 수 있습니다.

주석 서비스를 사용하여 응용프로그램 빌드

이 절은 주석 서비스 응용프로그램을 빌드하는 데 수행할 단계 및 빌드하는 데 사용되는 API를 포함합니다. API 사용에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

1. 주석 콜백을 처리하는 추상 매소드를 구현하려면

CMBAnnotationServicesCallbacks 서브클래스를 작성하십시오.

2. CMBAnnotationServices 인스턴스를 작성하십시오.

```
CMBAnnotationServices annoServices = new
CMBAnnotationServices(annoServicesCallbacks);
```

3. 주석 스트림을 로드하여 CMBAnnotationSet 인스턴스를 가져오십시오.

```
CMBAnnotationSet annotationSet = annoServices.loadAnnotationSet(annoStream,
format, documentResolution, annotationPartNumber );
```

4. 주석 보기를 준비하십시오.

```
CMBAnnotationComponent annoComponent = new CMBAnnotationComponent();
CMBAnnotationViewer annoView = annoServices.prepareAnnotationView(
annoComponent, annotationSet );
annoView.refreshEntireDrawingArea();
```

5. 응용프로그램의 JFrame이나 JPanel과 같은 스윙 컨테이너에 주석 구성요소를 추가하십시오.

6. 이제 주석 서비스 APT를 사용하여 새 주석을 추가, 편집 또는 기존 주석을 삭제할 수 있습니다.

```
annoServices.prepareToAddAnnotation();
annoServices.addAnnotation()
annoServices.removeAnnotation();
annoServices.reorderAnnotation();
...
```

7. 수정된 주석을 저장하십시오.

```
annoServices.saveAnnotationset( annotationSet );
```

주석 서비스 샘플은 <CMBROOT>\Samples\java\viewer directory (TAnnotationEditor.java) 디렉토리에 제공됩니다.

응용프로그램에 사용자 조정 주석 유형 추가

주석 서비스에 사용자 조정 주석 유형을 추가하려면 아래 단계를 완료하십시오. API 사용에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

1. CMBPageAnnotation의 서브클래스를 작성하십시오. `public class TImageAnnotation extends CMBPageAnnotation`
2. 100보다 큰 값으로 사용자 조정 주석에 대한 상수를 정의하십시오. 1에서 99 사이의 범위가 예약됩니다.

`public static final int ANN_IMAGE = 101;`
3. 다음 CMBPageAnnotation 메소드를 대체하십시오.

`public void draw(Graphics2D g2);
public void drawOutline(Graphics2D g2);
public CMBPropertiesPanel getAnnotationPropertiesPanel();`
4. CMBAnnotationPropertiesInterface 인터페이스를 구현하여 등록 정보 패널을 작성하십시오. 등록 정보 패널은 사용자 조정 주석 등록 정보를 편집하려고 선택하면 표시됩니다.
5. 사용자 조정 주석 등록 정보에 고유한 set 및 get 메소드를 제공하십시오.
6. 사용자 조정 주석 유형을 추가하십시오.

`annoServices.prepareToAddAnnotation(TImageAnnotation.ANN_IMAGE,
"TImageAnnotation",1);`

주석 유형 샘플 TImageAnnotation이 <CMBROOT>\Samples\java\viewer 디렉토리에 포함되어 있습니다. 샘플에는 사용자 조정 주석 유형을 주석 서비스에 추가하는 방법이 설명되어 있습니다.

Enterprise Information Portal 태그 라이브러리 및 제어기 servlet에 대한 작업

Enterprise Information Portal에는 웹 응용프로그램용 JSP 또는 servlet을 작성할 때 사용할 수 있는 JSP(Java Server Pages) 태그 라이브러리 및 servlet이 들어 있습니다. 태그 라이브러리를 사용하면 EIP JavaBeans에 작성된 JSP의 Java scriptlets에 대한 필요성이 줄어듭니다.

이 태그 라이브러리는 모델 보기 제어기 설계 웹 응용프로그램의 제어기로 작동할 수 있는 servlet과 함께 사용되며 Bean 초기화 및 기타 조치를 수행합니다.

태그 라이브러리 및 servlet 설정

IBM WebSphere® Application Server가 있는 웹 서버에 태그 라이브러리 및 servlet을 설치하고 웹 서버가 이들을 사용하도록 구성해야 합니다. 태그 라이브러리 및 servlet을 설치하고 구성하는 방법에 대한 정보 및 WAR/EAR 파일 빌드에 대한 정보는 *Enterprise Information Portal* 계획 및 설치를 참조하십시오.

태그 라이브러리 사용

다음 JSP 샘플에 검색 템플릿 태그를 사용하여 검색 템플릿 목록을 가져오는 방법이 나와 있습니다.

```
<%@ 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>
      <cmb:searchtemplates>
        <tr>
          <td><%= searchtemplate.getName() %></td>
        </tr>
      </tr>
    </table>
  </body>
</html>
```

```

        </cmb:searchtemplates>
    </table>
</body>
</html>

```

taglib 지시문은 페이지가 EIP 태그 라이브러리를 사용하고 cmb 접두부와 연관되어 있음을 선언합니다. 그런 다음 searchtemplates 태그가 호출되고 getName() 메소드는 각 검색 템플릿의 이름을 리턴합니다.

태그 라이브러리에 사용되는 규칙

JSP 태그는 자신이 사용하거나 생성하는 Bean을 지정하는 속성을 가지고 있습니다. 이 속성을 지정하지 않으면 기본값이 사용됩니다. 이 매개변수는 선택적입니다. 이 값은 요청 및 세션 범위의 로컬 변수 및 속성에서 가져옵니다. servlet 툴킷과 함께 사용하는 경우, 로컬 변수, 요청 속성 또는 세션 속성에 해당 기본값이 포함됩니다. 표 32에는 사용될 것으로 간주되거나 배치된 기본 Bean 및 범위가 나와 있습니다. servlet에서도 이 규칙을 따릅니다. 태그 라이브러리에 대해 작업하기 위해 작성한 다른 servlet에서는 이 규칙을 따르십시오.

표 32. 태그 라이브러리 규칙

범위	이름	유형	설명
응용프로그램	connectionPool	CMBCConnectionPool	세션에서 공유되는 연결 풀 Bean
세션	connection	CMBCConnection	세션에 대한 CMBCConnection 인스턴스
세션	schema	CMBSchemaManagement	스키마 관리 Bean
세션	data	CMBDataManagement	데이터 관리 Bean
세션	user	CMBUserManagement	사용자 관리 Bean
세션	query	CMBQuesryService	조회 서비스 Bean
세션	traceLog	CMBTraceLog	추적 로그 Bean. 다른 모든 Bean은 추적을 이 Bean으로 전송합니다.
세션	docservices	CMBDocumentServices	문서 서비스 Bean
요청	item	CMBItem	조작한 마지막 항목
요청	items	CMBItem[]	마지막으로 조작한 항목 컬렉션
요청	searchTemplate	CMBSearchTemplate	선택된 검색 템플릿
요청	searchResutls	CMBSearchResults	마지막 검색 결과

태그 요약

다음 절에서는 태그 라이브러리를 요약합니다.

연결 관련 태그

<cmb:datasources connection="connection">datasource ... </cmb:datasources>

이 태그는 사용 가능한 데이터 원본 전체를 반복합니다.

connection

연결을 포함하는 CMBConnection 유형의 변수 이름을 지정합니다.

datasource

데이터 원본 이름을 문자열로 포함할 문자열 변수.

스키마 관련 태그

<cmb:searchtemplates searchTemplates="searchTemplate"> ...

</cmb:searchTemplates>

이 태그는 사용 가능한 검색 템플릿 전체를 반복합니다.

searchTemplates

검색 템플릿을 포함할 CMBSearchTemplate[] 유형의 배열 이름을 지정합니다.

searchTemplate

검색 템플릿을 포함할 CMBSearchTemplate 유형의 변수.

<cmb:searchcriteria searchTemplate="searchTemplate">criteria ...

</cmb:searchcriteria>

이 태그는 검색 템플릿의 검색 기준 전체를 반복합니다.

searchTemplate

검색 템플릿의 이름을 지정합니다.

criteria

검색 기준을 포함할 CMBSTCriterion 유형의 변수.

<cmb:displaycriteria searchTemplate="searchTemplate">criteria ...

</cmb:displaycriteria>

이 태그는 검색 템플릿에 표시될 수 있는 검색 기준 전체를 반복합니다.

searchTemplate

검색 템플릿의 이름을 지정합니다.

criteria

검색 기준을 포함할 CMBSTCriterion 유형의 변수.

<cmb:allowedoperators criterion="criterion">operator ... </cmb:allowedoperators>

이 태그는 검색 기준에 허용되는 연산자 전체를 반복합니다.

criterion

검색 기준을 포함할 CMBSTCriterion 유형의 변수 이름을 지정합니다.

operator

연산자의 값을 포함할 문자열.

<cmb:predefinedvalues criterion="criterion"> value... </cmb:predefinedvalues>

이 태그는 검색 기준의 미리 정의된 값 전체를 반복합니다.

criterion

검색 기준을 포함할 CMBSTCriterion 유형의 검색 이름을 지정합니다.

value 검색 기준의 미리 정의된 값을 포함할 문자열.

<cmb:entities schema="schema">entity ... </cmb:entities>

이 태그는 사용 가능한 연합 엔티티 전체를 반복합니다.

schema

스키마를 포함하는 CMBSchemaManagement 유형의 변수 이름을 지정합니다.

entity 엔티티의 이름을 포함할 문자열.

<cmb:attributes entity="entity" schema="schema">attribute ... </cmb:attributes>

이 태그는 연합 엔티티의 연합 속성 전체를 반복합니다.

schema

스키마를 포함할 CMBSchemaManagement 유형의 변수 이름을 지정합니다.

entity 엔티티의 이름을 지정합니다.

attribute

속성을 포함하는 CMBAttribute 유형의 변수 이름을 보유하는 문자열.

검색 관련 태그

< cmb:searchresults searchresults="searchResults">item ... </cmb:searchresults>

이 태그는 검색 결과 전체를 반복합니다.

searchResults

검색 결과를 포함하는 CMBSearchResults 유형의 변수 이름을 지정합니다.

item 검색에서 발생한 항목이 들어 있는 CMBItem 유형의 변수 이름을 포함할 문자열.

항목 관련 태그

<cmb:itemattributes item="item"> attrname ... attrtype... attrvalue... </cmb:itemattributes>

이 태그는 항목의 속성 전체를 반복합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

attrname

속성 이름을 포함할 문자열 변수.

attrtype

속성 유형을 포함할 문자열 변수.

attrvalue

속성값을 포함할 문자열 변수.

<cmb:itemcontents " data="data" item="item"> content... </cmb:itemcontents>

이 태그는 항목의 콘텐츠 전체를 반복합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

content

항목의 콘텐츠에 대한 CMBOject 유형의 변수.

<cmb:itemnotelogs " data="data" item="item">notelog ... </cmb:itemnotelogs>

이 태그는 항목의 메모 로그 전체를 반복합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

notelog

항목의 메모 로그를 포함할 CMBOject 유형의 변수.

<cmb:itemprivileges data="data" item="item">privilege ... </cmb:itemprivileges>

이 태그는 항목의 사용 권한 전체를 반복합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

privilege

항목의 사용 권한을 포함할 CMBPrivilege 유형의 변수.

<cmb:itemresources data="datamanagement" item="item"> resource... </cmb:itemresources>

이 태그는 항목의 자원 전체를 반복합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

resource

항목의 자원을 포함할 CMBResources 유형의 변수.

<cmb:unmappeditem data="data" item="item">unmappeditem...</cmb:unmappeditem>

이 태그는 주어진 맵핑 항목으로부터 맵핑되지 않은 항목을 리턴합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 맵핑된 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

unmappeditem

맵핑되지 않은 항목을 포함할 CMBItem 유형의 변수.

<cmb:viewdata data="data" item="item">viewdata... </cmb:viewdata>

이 태그는 항목의 보기를 리턴합니다.

data CMBDataManagement 유형의 변수 이름을 지정합니다.

item 항목을 포함하는 CMBItem 유형의 변수 이름을 지정합니다.

viewdata

볼 수 있는 데이터를 포함할 CMBViewData 유형의 변수.

폴더 관련 태그

<cmb:folderitems folder="folder"> item... </cmb:folderitems>

이 태그는 폴더의 콘텐츠 전체를 반복합니다.

folder 폴더 콘텐츠를 포함할 CMBItem 유형의 변수 이름을 지정합니다.

item 폴더를 나타내는 CMBItem 유형의 변수.

문서 관련 태그

<cmb:viewdocuments docservices="docservices">document ...

</cmb:viewdocuments>

이 태그는 현재 로드된 문서 전체를 반복합니다.

docservices

CMBDocumentServices 유형의 변수 이름을 지정합니다.

document

문서를 포함할 CMBDocument 유형의 변수.

<cmb:documentpages document="document"> docpage... </cmb:documentpages>

이 태그는 문서의 페이지 전체를 반복합니다.

document

문서를 포함할 CMBDocument 유형의 변수 이름을 지정합니다.

docPage

페이지를 포함할 CMBPage 유형의 변수.

EIP 제어기 servlet

EIP는 웹 응용프로그램 빌드 시 사용할 수 있는 플러그 가능한 조치를 servlet에 제공합니다. 이 servlet은 조치를 수행하고 JSP 태그를 사용하여 JSP(보기)에서 직접 또는 간접적으로 액세스되는 Bean(모델)을 초기화하는 모델 보기 제어기 설계 웹 응용프로그램의 제어기로 작동합니다.

다음과 같은 일반적인 응용프로그램 작업을 위한 조치가 제공됩니다.

- 로그인 및 로그오프.
- 검색.
- 문서 작성, 검색, 수정 및 삭제.
- 폴더 작성, 폴더에 문서 추가 또는 폴더에서 문서 제거.
- 보기위해 문서 및 문서 페이지 시작.

또한 servlet은 조치의 전후에 콘텐츠 서버에 대한 연결 관리와 같은 공통 작업을 수행합니다. 모든 조치 다음에 결과를 형식화하여 이를 브라우저로 다시 전송하기 위해 JSP가 호출됩니다.

새 조치를 추가하고 JSP를 조치와 연관시키도록 servlet을 사용자 조정할 수 있습니다.

servlet이 수행할 수 있는 작업

다음에는 사용할 수 있는 servlet의 몇몇 종류가 있습니다.

연결 풀링

제어기 servlet은 EIP 연결 풀링을 사용하여 고성능 연결 관리를 제공합니다. 세션에 할당된 연결 시간은 세션이 로그인된 시간 또는 요청을 위한 것일 수 있습니다. 현재 연결 풀링은 응용프로그램 범위에 있습니다.

시간 종료 세션의 로그인

세션이 시간종료되고 servlet으로 요청이 들어가면 사용자가 다시 로그인할 수 있도록 하는 로그인 JSP가 표시됩니다. 로그인 후 원래 요청이 수행됩니다.

세션 종료 제거

로그오프 또는 시간종료에 의해 세션이 종료되면 servlet은 적절히 세션을 제거합니다. 이는 연결이 제거되거나 풀로 리턴된다는 것을 의미합니다. servlet이 작성한 다른 모든 EIP Bean이 종료되고 해당 자원은 가비지 콜렉션 주기 발생을 기다리지 않고 해제됩니다.

로케일 servlet은 로케일이 기반 Bean에 대해 올바르게 설정되도록 하므로 메시지 및 문자열은 로케일을 구분합니다.

다른 JSP 세트 사용

cmbservletjsp.properties라는 등록 정보 파일은 기본적으로 servlet 조치

에 대한 응답에 사용할 JSP를 설명합니다. 등록 정보 파일의 위치는 응용프로그램 매개변수입니다. 따라서 몇몇 다른 웹 응용프로그램은 다른 JSP 세트를 사용하여 작성될 수 있습니다.

servlet 확장

servlet에 알려진 모든 조치는 cmbervlet.properties(기본값)라는 등록 정보 파일에 정의되어 있습니다. 이 파일을 변경하여 servlet 조치를 추가, 수정 또는 삭제할 수 있습니다. 새 조치를 추가하려면 다음 단계를 수행하십시오.

1. 조치를 수행하려면 클래스를 구현하십시오. 클래스는 `com.ibm.mm.servlets.CMBServletAction`을 확장해야 합니다.
2. 클래스 이름 및 조치 이름을 cmbervlet.properties 파일에 추가하십시오. 이는 다음과 같은 구문규칙을 가지고 있습니다.

```
actions = list of actionsaction.<action_name>.class = class_name
```

actions는 servlet이 이해한 조치를 나열합니다. 각 조치에 대해 등록 정보 파일 내의 행은 조치에 대한 클래스를 정의합니다. ReplayAction이라는 클래스에 replay라는 조치를 추가하는 예제는 다음과 같습니다.

```
actions = ... replay
action.replay.class = ReplayAction
```

조치를 대체하거나 또는 미리 정의된 조치를 진행하거나 따르기 위해 사용자 조치를 제공할 수도 있습니다. 예를 들어, 사용자 조치로 로그온을 진행하려면 추가 유효성 확인을 수행하십시오.

```
action.logon.class = MyLogonAction com.ibm.mm.servlets.CMBLogonAction
```

미리 정의된 모든 조치에 사용된 이름 지정 규칙은

`com.ibm.mm.servlets.CMBactionAction`이고, 여기서 *action*은 첫 번째 문자가 대문자인 조치의 이름입니다.

servlet 참조

응용프로그램 매개변수, 요청 매개변수 세트 및 등록 정보 파일을 사용하여 응용프로그램에서 제어기 servlet을 사용합니다.

규칙

servlet은 다른 JSP 또는 servlet에서 사용될 수 있는 다음 세션 및 요청값을 정의합니다. 이러한 규칙은 JSP 태그 라이브러리에서 수행됩니다. 이러한 규칙은 EIP 태그 라이브러리에 대한 규칙과 동일합니다.

응용프로그램 매개변수

servlet은 다음 응용프로그램 매개변수를 이해합니다.(대안은 이러한 매개변수를 cmbervlet.properties 파일에 놓는 것입니다.)

응용프로그램 매개변수	값	설명
servletPropertiesURL	URL	cmbServlet.properties 파일의 위치
defaultServerType	Fed, ICM, OD, DL, DES, V4, IP, DD, ...	기본 로그인 정보. defaultServer, defaultUserid 및 defaultPassword와 함께 이는 공유 사용자 ID에서 사용될 수 있습니다. 로그인 페이지를 사용하여 프롬프트를 표시하기보다는 기본 로그인 정보가 로그인을 수행하는 데 사용됩니다.
defaultServer		기본 로그인 정보
defaultUserid		기본 로그인 정보
defaultPassword		기본 로그인 정보
connectionpool	Boolean: true false	연결 풀링 사용 가능화
maxfreeconnection	정수	연결 풀에서 사용 가능한 최대 연결 수
minfreeconnection	정수	연결 풀에서 사용 가능한 최소 연결 수
timeout	정수	무료 연결이 해제 후 지속 시간(단위: 밀리초)
noSessionPage	URL	이것은 세션 또는 연결이 설정되지 않고 servlet 이 호출된 경우 로그인을 표시하는 페이지입니다. 사용자가 로그인해야 할 경우 작업할 EIP 로 북마크된 링크를 허용하여 로그인을 위한 프롬프트를 표시하고 원래 조치로 되돌아가도록 하는 데 사용될 수 있습니다.
timedOutPage	URL	이것은 비활성으로 인해 세션이 시간 종료될 경우 표시되는 페이지입니다.
serverErrorPage	URL	이것은 서버에 액세스 시 오류가 발생할 경우 표시되는 페이지입니다.
connectFailedPage	URL	이것은 서버에 연결 시 오류가 발생할 경우 표시되는 페이지입니다. 서버에 대한 올바른 사용자 ID/암호를 입력하기 위한 프롬프트가 표시되어 재시도될 수 있습니다.
tracelevel	0, 1 또는 2	추적의 레벨을 표시하려면 다음과 같이 하십시오. <ul style="list-style-type: none"> • 0 - 아무것도 기록 안함 • 1 - 예외만 기록(기본값) • 2 - 예외, 경고 메시지, WebSphere Application Server 헤더 및 속성, EIP ini 파일, JVM 시스템 등록 정보, EIP 내부 추적 정보 기록
connectiontype	0, 1 또는 2	EIP 데이터베이스 및 콘텐츠 서버 런타임의 위치 <ul style="list-style-type: none"> • 0 - 로컬(기본값) • 1 - 원격 • 2 - 동적
cmbclient	URL	cmbclient.ini의 위치
cmbcs	URL	cmbcs.ini의 위치

응용프로그램 매개변수	값	설명
serviceconnectiontype	0, 1 또는 2	서비스 런타임의 위치 <ul style="list-style-type: none"> • 0 - 로컬(기본값) • 1 - 원격 • 2 - 동적
cmbsvclient	URL	cmbsvclient.ini의 위치
cmbsvcs	URL	cmbsvcs.ini의 위치
cmbcc2mime	URL	cmbcc2mime.ini의 위치
cachedir	디렉토리 이름	문서 변환 중 문서를 캐시하기 위한 디렉토리
jnitrace	파일 이름	문서 변환에 사용된 JNI 논리를 위해 JNI 추적 정보를 기록하는 파일(IBM 진단용)
conversion	Boolean: true 또는 false	true인 경우, 가능하다면 문서는 중간 단계의 브라우저에서 표시될 수 있는 형식으로 변환됩니다. false인 경우, 변환되지 않은 원래 문서가 브라우저로 전송됩니다.
maxresults	정수	리턴된 최대 히트 수. -1(기본값)은 모든 히트 수를 의미합니다.
valuedelimiter	문자	검색 기준에서 값을 구분하는 문자를 정의합니다. 기본값은 로케일에 따라 다르며, 미국 영어의 경우 쉼표(,)입니다.
conversion.<mimetype>	<none document page >	특정 MIME 유형의 문서를 보기 위한 변환 옵션. 이 옵션은 viewDocument servlet 작동에 영향을 미칩니다. 여기서 Page는 문서에 페이지 수를 지정하는 것을 의미합니다. Document는 브라우저에서 읽을 수 있는 양식으로 문서를 변환하는 것을 의미합니다. None은 변환을 수행하지 않고 문서를 원래의 양식으로 리턴하는 것을 의미합니다.
nameseparator	문자	규정된 이름의 상위 구성요소 속성에서 하위 구성요소 속성을 분리하는 문자를 정의합니다. 기본값은 로케일에 따라 다르며, 미국 영어의 경우 슬래시(/)입니다.

등록 정보 파일

servlet은 등록 정보 파일 cmbservlet.properties를 찾습니다. 이 파일은 여기에 정의된 조치를 포함하여 servlet이 사용할 수 있는 조치를 정의합니다. 사용되는 JSP 파일의 이름도 정의합니다.

웹 응용프로그램 서버(servlet 엔진)에 대해 servlet 등록 정보도 정의할 수 있습니다. 구문규칙은 파일에 사용된 것과 동일합니다.

제어 servlet에서 cmbservlet.properties 콘텐츠를 등록 정보 오브젝트에 저장합니다. 다음 예제처럼 응용프로그램 속성 "cmbServletProperties"를 통해 액세스할 수 있습니다.

```
// 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
```

요청 매개변수

servlet은 다음 요청 매개변수를 이해합니다. 응답 JSP에 사용하기 위해 추가 매개변수가 지정될 수 있습니다.

일반

action=action

수행될 조치. 허용된 추가 매개변수는 조치에 기준하며 아래에 설명됩니다.

이 매개변수는 선택적입니다. 이 매개변수가 지정되지 않은 경우, 응답 페이지는 시간종료 및 로그인에 대한 연결 점검과 같은 표준 설정을 수행한 후 servlet에서 실행됩니다.

reply=<URL>

(선택적). cmbervlet.properties 파일 조치에 대한 응답으로 정의된 JSP가 아닌 해당 매개변수에 지정된 JSP로 전달합니다. 조치 매개변수가 지정되지 않았는데 응답이 지정된 경우, 응답 페이지는 시간종료 및 로그인에 대한 연결 점검과 같은 표준 설정을 수행한 후 제어기 servlet에서 실행됩니다.

관련 연결

action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]

서버에 로그인합니다. 서버 유형, 서버 이름, 사용자 ID 및 암호를 지정해야 합니다. 연결 문자열 및 초기 문자열은 선택적이며 서버 유형에 따라서 다릅니다.

action=logoff [endSession=<true|false>]

서버에서 로그아웃합니다. 또한 이 세션은 기본적으로 종료됩니다.

관련 검색

action=searchTemplate template=<> {<criterionname>.op=<> <criterionname>=<>}

지정된 검색 템플릿 및 기준값을 사용하여 검색을 수행합니다™.

action=searchEntity entity=<> {attribute.<attrname>.op=<> attribute.<attrname>=<>} [conjunction=<andlor>]

엔티티를 사용하여 검색을 수행합니다. 속성값 및 연산자도 지정될 수 있습니다. 속성값에서 여러 값은 응용프로그램 매개변수에서 지정된 대로 값 분리문자로 구분됩니다. 결합 매개변수에서 지정된 대로 속성은 and(기본값) 또는 or를 사용하여 조회를 형성하도록 함께 결합됩니다.

**action=searchQuery queryString=<> {queryParameter.
<parametername>=<>}**

지정된 조회 문자열을 사용하여 검색을 수행합니다. 조회 구문규칙은 검색되는 서버에 따라서 다릅니다.

다수의 추가 조회 매개변수가 지정될 수 있습니다. 이들은 또한 서버에 따라 다릅니다.

관련 항목

action=lock itemId=<>

갱신하는 동안 독점 액세스를 위해 보통 항목을 잠급니다.

action=unlock itemId=<>

잠긴 항목을 잠금 해제 합니다.

**action=createItem type=<document|folder> entity=<>
{attribute.<attrname>=<attrvalue>}**

항목을 작성합니다. 전송된 경우, 콘텐츠가 제공될 수 있습니다.

action=retrieveItem itemId=<>

항목의 속성과 콘텐츠를 검색합니다. 이는 서버에 저장된 최신 콘텐츠가 사용되도록 할 때 유용합니다.

**action=updateItem itemId=<> [entity=<>] {attribute.
<attrname>=<attrvalue>}**

항목의 속성을 갱신합니다. 엔티티가 지정된 경우, 해당 항목은 다시 색인화됩니다. servlet이 전송을 통해 호출된 경우, 콘텐츠도 갱신됩니다.

action=deleteItem itemId=<>

항목을 삭제합니다.

action=addContent itemId=<>

항목에 콘텐츠 부분을 추가합니다. 콘텐츠 데이터가 전송됩니다.

action=getContent itemId=<> contentIndex=<>

콘텐츠 부분을 가져와서 이를 브라우저로 리턴합니다.

action=updateContent itemId=<> contentIndex=<>

항목에 대한 콘텐츠 부분을 갱신합니다. 콘텐츠 데이터가 전송됩니다. 콘텐츠가 존재하지 않는 경우, 콘텐츠 부분이 추가됩니다.

action=deleteContent itemId=<> contentIndex=<>

지정된 항목에 대한 콘텐츠 부분을 삭제합니다.

action=addNoteLog itemId=<>

항목의 메모 로그를 수정합니다. 메모 로그 텍스트가 전송됩니다.

action=updateNoteLog itemId=<> notelogIndex=<>

항목의 메모 로그를 수정합니다. 메모 로그 텍스트가 전송됩니다. 메모 로그가 존재하지 않는 경우에는 추가됩니다.

action=deleteNoteLog itemId=<> notelogIndex=<>

항목의 메모 로그 텍스트를 삭제합니다. 메모 로그의 텍스트가 전송됩니다.

관련 폴더

action=addItemToFolder itemId=<> folderId=<>

지정된 폴더에 지정된 항목을 추가합니다.

action=removeItemFromFolder itemId=<> folderId=<>

지정된 폴더에서 지정된 항목을 제거합니다.

관련 문서

action=viewDocument itemId=<>

문서를 검색한 후 봅니다. 문서에 페이지 수가 지정되면 이 조치는 보기 프로그램 프레임 세트를 생성하는 JSP로 전달됩니다. 문서 페이지 수가 지정되지 않으면 이 조치는 문서의 실제 콘텐츠를 리턴합니다.

action=viewPage itemId=<> page=<> scale=<> rotation=<>

annotations=<yes|no>

문서 페이지를 검색합니다.

Servlet 툴킷 기능 매트릭스

표 33. Servlet 기능 매트릭스

조치	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
로그온	Y	Y	Y	Y	Y	Y	Y
로그오프	Y	Y	Y	Y	Y	Y	Y
searchTemplate	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	Y	적용되지 않음
searchEntity	Y	Y	Y	Y	Y	Y	적용되지 않음
searchQuery	Y	Y	Y	Y	Y	Y	Y
잠금	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
잠금 해제	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
createItem	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
retrieveItem	Y	Y	Y	Y	Y	Y	Y
updateItem	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
deleteItem	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
addContent	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
getContent	Y	Y	Y	Y	Y	Y	Y
updateContent	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음

표 33. Servlet 기능 매트릭스 (계속)

조치	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
deleteContent	Y	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
addNoteLog	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
updateNoteLog	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
deleteNoteLog	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
addItemToFolder	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
removeItemFromFolder	Y	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
viewDocument	Y	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y	Y

Y -- 기능 지원 N/A -- 기능을 지원하지 않음 주: 로그인, 로그오프, getContent, retrieveitem, viewDocument 및 viewPage 조치는 모든 콘텐츠 서버에서 지원됩니다.

표 34. Servlet 기능 매트릭스 계속

조치	DES	DB2	JDBC	IC	Fed	FileNet
로그온	Y	Y	Y	Y	Y	Y
로그오프	Y	Y	Y	Y	Y	Y
searchTemplate	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
searchEntity	적용되지 않음	Y	Y	적용되지 않음	Y	적용되지 않음
searchQuery	Y	Y	Y	Y	Y	Y
잠금	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
잠금 해제	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
createItem	적용되지 않음	Y	Y	적용되지 않음	적용되지 않음	적용되지 않음
retrieveItem	Y	Y	Y	Y	Y	Y
updateItem	적용되지 않음	Y	Y	적용되지 않음	Y	적용되지 않음
deleteItem	적용되지 않음	Y	Y	적용되지 않음	Y	적용되지 않음
addContent	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
getContent	Y	Y	Y	Y	Y	Y
updateContent	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
deleteContent	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
addNoteLog	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
updateNoteLog	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
deleteNoteLog	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	Y	적용되지 않음
addItemToFolder	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
removeItemFromFolder	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음	적용되지 않음
viewDocument	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y

Y -- 기능 지원 N/A -- 기능이 지원되지 않음 주: 로그인, 로그오프, getContent, retrieveitem, viewDocument 및 viewPage 조치는 모든 콘텐츠 서버에서 지원됩니다.

Information Mining에 대한 작업

이 절은 Information Mining Bean을 사용하여 응용프로그램을 빌드하는 방법, 사용자의 콘텐츠 제공자 빌드, Information Mining 서비스 API 사용 및 JSP 샘플에 대한 작업으로 구성되어 있습니다.

Bean을 사용하여 Information Mining 응용프로그램 빌드

Information Mining은 첨단 텍스트 분석 및 마이닝 기술을 기반으로 강력한 응용프로그램을 작성할 수 있도록 합니다. Information Mining Bean은 다음을 제공합니다.

- 텍스트 요약 및 카테고리, 즉 문서에 요약 할당
- 카테고리, 즉 문서에 카테고리 할당
- 정보 추출, 즉 문서로부터 관련 정보를 찾기 및 추출
- 언어 식별, 즉 문서가 작성된 언어 판별
- 클러스터링, 즉 문서 컬렉션에 있는 유사 문서의 그룹화
- 특정 카테고리의 문서로 제한될 수 있는 텍스트 검색
- IBM Web Crawler를 사용하여 웹에서 지속적으로 검색된 문서에 액세스

주

Information Mining Bean을 사용하기 전에 서비스 API와 JavaBeans 사이의 차이점을 이해하는 것이 중요합니다.

- Information Mining JavaBeans는 신속한 응용프로그램 개발을 위한 소프트웨어 구성요소로서 JavaBeans을 따릅니다. 코드의 일부 요소는 ‘함께 묶여’ 있으므로 사용자가 변경할 수 없습니다.
- Java 서비스 API에는 응용프로그램 작성을 위한 개별 빌딩 블록으로서 전체 Information Mining 기능이 포함됩니다. 자세한 정보는 557 페이지의 『서비스 API 사용』을 참조하십시오.

Information Mining Bean을 사용하려면 먼저 Bean을 실행할 기계에 Enterprise Information Portal Information Mining 구성요소를 설치 및 구성해야 합니다.

Information Mining Bean은 다음과 같습니다.

- **CMBSummarizationService**

이것은 문서에 대한 요약을 작성하는 데 사용할 수 있는 비주일이 아닌 Bean입니다. 이 Bean은 세 가지 다른 모드로 실행될 수 있고 요약의 길이를 결정할 수 있습니다.

- **CMBCategorizationService**

이것은 문서 카테고리를 결정하는 데 사용할 수 있는 비주일이 아닌 Bean입니다. 이 Bean은 Information Structuring Tool을 사용하여 얻은 결과(즉, 다른 카테고리 및 각 카테고리에 대해 설명하는 규칙으로서 식별되는 용어를 구성하는 작성된 분류 및 메타데이터)에서 실행됩니다. 문서는 하나 이상의 카테고리에 할당될 수 있습니다. 카테고리 결과는 하나 이상의 카테고리, 문서가 각 카테고리에 얼마나 잘 맞는지를 보여주는 신뢰값으로 구성되어 있습니다. 리턴될 최대 결과 수뿐만 아니라 이 신뢰값도 설정할 수 있습니다.

- **CMBInformationExtractionService**

이것은 문서로부터 이름, 용어 및 표현식을 추출하는 데 사용할 수 있는 비주일이 아닌 Bean입니다.

- **CMBLanguageIdentificationService**

이것은 문서가 작성된 언어를 판별하는 데 사용할 수 있는 비주일이 아닌 Bean입니다.

- **CMBClusteringService**

이것은 문서 또는 클러스터의 유사 세트로 문서 컬렉션을 파티션하는 데 사용할 수 있는 비주일이 아닌 Bean입니다.

- **CMBAdvancedSearchService**

CMBAdvancedSearchService는 검색을 수행하는 비주일이 아닌 Bean입니다. 고급 검색은 Information Mining 데이터스토어에 저장된 항목만 찾을 수 있습니다.

- **CMBCatalogService**

CMBCatalogService는 지속적 Information Mining 데이터스토어를 유지보수합니다. 이 서비스를 사용하려면 Information Structuring Tool을 사용하여 카탈로그를 작성해야 합니다. Bean을 사용하여 Information Mining 서비스에서 작성된 결과를 저장하고, 이 정보를 검색 및 삭제할 수 있습니다.

- **CMBWebCrawlerService**

CMBWebCrawlerService는 웹 공간, 즉 비동기 실행 중인 웹 크롤러가 검색된 모든 HTML 문서를 저장하는 디렉토리를 모니터링합니다. CMBItems(문서를 나타내는 EIP 헬퍼 클래스)는 Information Mining 서비스와 함께 사용될 수 있는 이들 파일로부터 작성되거나, CMBCatalogService를 사용하여 Information Mining 데이터스토어로 가져오기만 하면 됩니다.

- **CMBInfoMiningAdapter**

CMBInfoMiningAdapter는 다른 Information Mining 이벤트 간의 교환입니다. 또한 이는 CMBResultEvent를 지원하여 Information Mining Bean이 다른 Enterprise

Information Portal Bean과 함께 작동할 수 있게 합니다. CMBInfoMiningAdapter는 다음 도표에 표시된 것처럼 Bean을 결합하여 다양한 시나리오를 빌드할 수 있게 합니다.

그림 39에서는 Information Mining bean 사용에 대해 설명합니다.

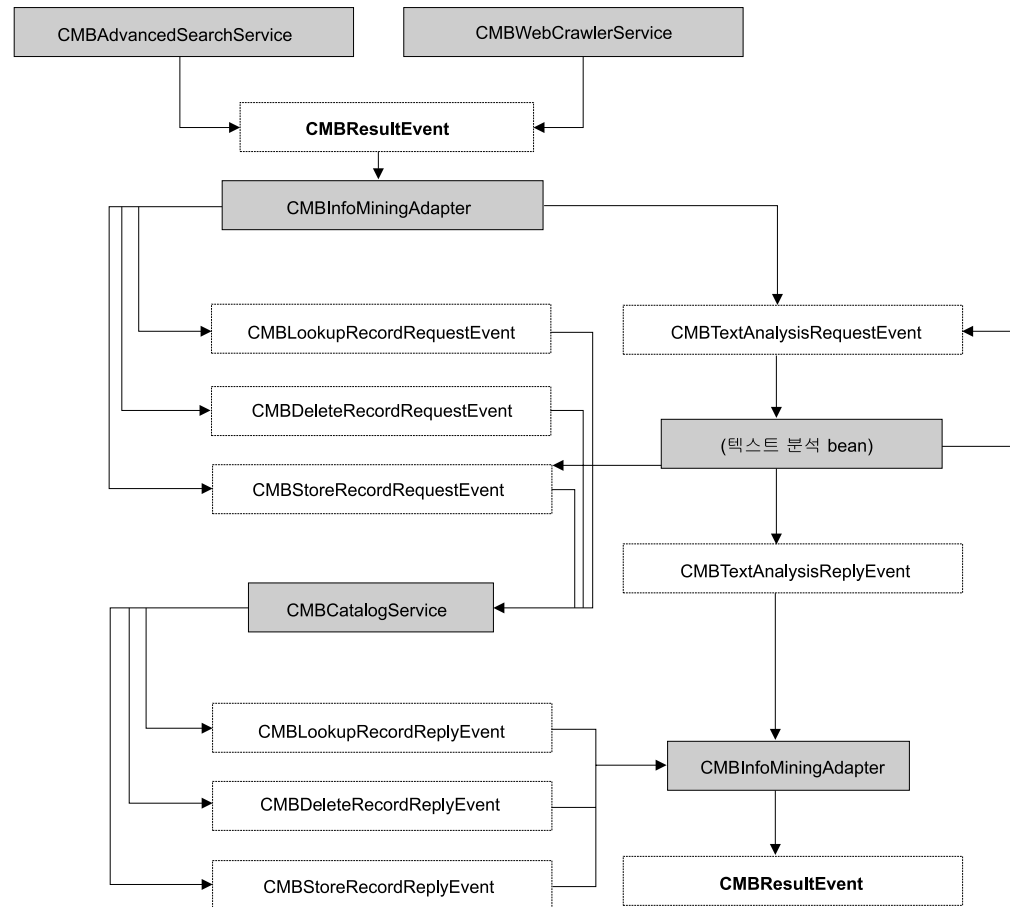


그림 39. Information Mining Bean

514 페이지의 그림 40에는 텍스트 분석 Bean이 나열됩니다.

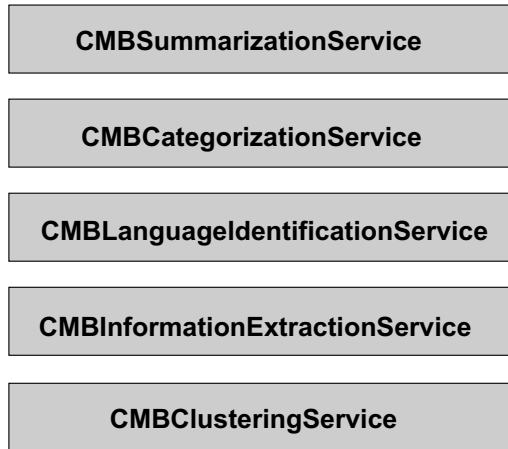


그림 40. 텍스트 분석 Bean

다음 예제에서는 Information Mining Bean을 사용하여 응용프로그램을 빌드할 수 있는 방법에 대해 설명합니다.

- **카테고리 샘플:** Information Mining 준비에 있어서 정보 수집은 라이브러리 관리자에 대한 일반적인 단계입니다. 516 페이지의 그림 41을 참조하십시오. 이 샘플에서 사용자는 표준 EIP(Enterprise Information Portal) 검색을 수행하여 EIP 콘텐츠 서버에서 문서를 찾습니다.(웹 크롤러 샘플에 웹으로부터 문서 수집이 나타납니다.) 그런 다음 문서의 언어가 판별되고 영어 문서가 구분됩니다. 카테고리 정보가 메타데이터 저장소에 저장됩니다.
- **요약 샘플:** 라이브러리 관리자에 대한 또다른 일반적인 단계입니다. 카테고리 샘플에서처럼 표준 EIP 검색을 수행하여 EIP 콘텐츠 서버에서 문서를 찾은 후 영어 문서를 식별하고, 문서를 요약하여 메타데이터 저장소에 요약을 저장합니다. 524 페이지의 그림 43을 참조하십시오.
- **정보 추출 샘플:** Information Mining 단계입니다. 이 샘플에서 사용자는 표준 EIP 검색을 수행하여 EIP 콘텐츠 서버에서 문서를 찾습니다. 그런 다음 영어 문서에서 이름 및 용어가 추출됩니다. 추출된 정보가 메타데이터 저장소에 저장됩니다.
이 내용은 530 페이지의 그림 45에 나와 있습니다.
- **클러스터링 샘플:** Information Mining 단계입니다. 사용자는 표준 EIP 검색을 수행하여 EIP 콘텐츠 서버에서 문서를 찾습니다. 그런 다음 영어 문서가 판별되고 클러스터링 서비스가 수동으로 트리거됩니다. 화면에 결과가 표시됩니다. 클러스터링 정보가 메타데이터 저장소에 저장되지 않습니다.
이 내용은 536 페이지의 『클러스터링』에 나와 있습니다.
- **웹 크롤러 샘플:** 웹 크롤러를 사용하여 문서를 가져온 다음 정보를 카테고리 내에서 검색하는 데 사용할 수 있도록 합니다(고급 검색 샘플). 또한 이 샘플은 EIP 콘텐츠 서버가 아닌 인터넷이나 인트라넷에서 문서를 수집한다는 것을 제외하고는 카테고리

및 요약 샘플과 유사한 라이브러리 관리자 단계입니다. 카테고리화 요약 정보를 카탈로그에서 검색 결과로 복원하려면 카탈로그 서비스를 사용하여 이 정보를 검색하십시오.

541 페이지의 그림 49를 참조하십시오.

- 고급 검색 샘플: Information Mining 단계입니다. 고급 검색을 수행하며 특정 카테고리화 제한된 융통성 있는 조화를 사용하여 문서를 검색할 수 있습니다. 카테고리화 요약 정보를 카탈로그에서 검색 결과로 복원하려면 카탈로그 서비스를 사용하여 이 정보를 검색하십시오.

샘플 파일의 위치

이 장에서 설명한 샘플은 다음 디렉토리에 제공됩니다.

샘플 위치

카테고리 응용프로그램

<CMBROOT>\samples\java\beans\infomining\categorization

요약 응용프로그램

<CMBROOT>\samples\java\beans\infomining\summarization

정보 추출 응용프로그램

<CMBROOT>\samples\java\beans\infomining\
informationExtraction

클러스터링 응용프로그램

<CMBROOT>\samples\java\beans\infomining\clustering

고급 검색 응용프로그램

<CMBROOT>\samples\java\beans\infomining\advancedSearch

웹 크롤러 응용프로그램

<CMBROOT>\samples\java\beans\infomining\webcrawler

각 디렉토리에는 소스 코드를 컴파일할 수 있게 하는 compile.bat 파일이 들어 있습니다. 또한 응용프로그램 샘플 디렉토리에는 샘플 응용프로그램을 실행할 수 있게 하는 run.bat 파일도 들어 있습니다.

문서 구분

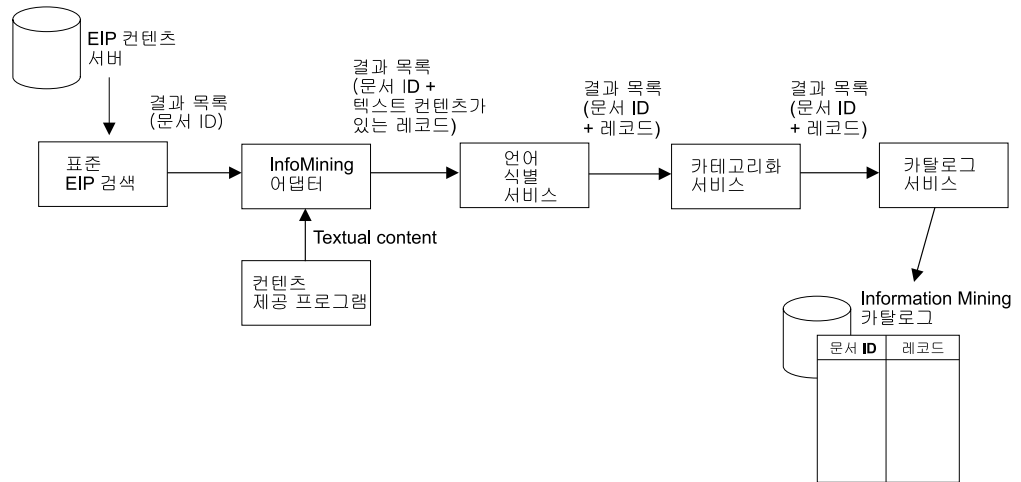


그림 41. 카테고리 샘플

이 샘플에서는 표준 EIP 검색에서 검색된 문서를 구분할 수 있는 방법에 대해 설명합니다. 분석 결과 저장되며 해당 문서는 카테고리 내의 검색에 사용할 수 있게 됩니다.

그림 41에서는 EIP 콘텐츠 서버에서 보유하는 문서에 대해 표준 EIP 검색이 수행됩니다. 언어 식별 서비스 Bean은 문서가 작성되는 언어를 판별합니다. Bean은 문서 ID를 가져가서 문서 콘텐츠를 검색한 후 콘텐츠를 분석하여 언어를 판별합니다. 카테고리 서비스 Bean은 문서를 구분합니다. 그런 다음 이 정보(문서 ID 및 카테고리 정보)는 후속 처리에 사용할 수 있습니다.

이 샘플에서는 다음 Bean이 사용됩니다.

- CMBCConnection
- CMBQueryService(표준 EIP 검색 수행)
- CMBSearchResults(표준 EIP 검색 수행)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBCategorizationService
- CMBCatalogService

이 샘플에 대해 해당 응용프로그램은 다음을 수행합니다.

1. Bean을 작성합니다.
2. Bean을 사용자 조정합니다.
3. 카테고리 서비스는 영어 문서를 분석합니다. 결과가 카탈로그에 저장됩니다.
4. 표준 EIP 조회를 실행합니다.

5. 확인할 검색 결과(문서의 목록)와 카테고리를 표시합니다.

Java 원본은 다음과 같습니다.

Categorization.java의 완전한 원본

```
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      = "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 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
sampleConnection.setServerName(sampDbName);
sampleConnection.setUserid(sampUserName);
sampleConnection.setPassword(sampPasswd);
sampleConnection.setConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);
sampleConnection.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_SYNCH,
                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 작성

검색을 수행하려면 콘텐츠 서버에 연결해야 합니다. CMBConnection Bean을 사용하여 연결을 설정할 수 있습니다. 표준 EIP 검색을 수행하려면 Bean CMBQueryService 및 CMBSearchResults가 필요합니다. CMBLanguageIdentificationService를 사용하여 문서의 언어가 판별되고 해당 레코드의 IKF_LANGUAGE 속성에 저장됩니다. 문서는 CMBCategorizationService를 사용하여 하나 이상의 카테고리에 할당되는데, 이는 카테고리 정보를 해당 레코드에 저장합니다. CMBCatalogService는 카탈로그에 항목의 카테고리 정보를 저장하고 해당 문서를 고급 검색에 사용할 수 있게 하는 데 사용됩니다. 검색 결과 이벤트를 텍스트 분석 요청 이벤트로 변환한 후 항목 응답 이벤트를 다시 검색 결과 이벤트로 저장하려면 두 개의 어댑터가 필요합니다.

필수 Bean을 작성하는 코드는 다음과 같습니다.

```

CMBConnection sampleConnection = 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 사용자 조정

사용자 조정 절에 표시된 코드는 설치에 따라 수정해야 합니다. 연결 Bean에서 연결할 콘텐츠 서버의 이름, 사용자 ID 및 적절한 암호를 지정해야 합니다.

카테고리 서비스 Bean과 카탈로그 서비스 Bean을 실행하려면 먼저 해당 Bean을 기존의 카탈로그와 연관시켜야 합니다. 또한 카탈로그 서비스 Bean에는 카테고리 정보가 포함되지 않는 항목을 할당할 기본 카테고리가 필요합니다.

샘플에 대해 사용자 조정되는 코드는 다음과 같습니다.

```

samlConnection.setServerName(sampDbName);
samlConnection.setUserid(sampUserName);
samlConnection.setPassword(sampPasswd);
samlConnection.setConnectionType
    (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samlConnection.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 연결

522 페이지의 그림 42에 Bean 간의 이벤트 플로우가 나와 있습니다.

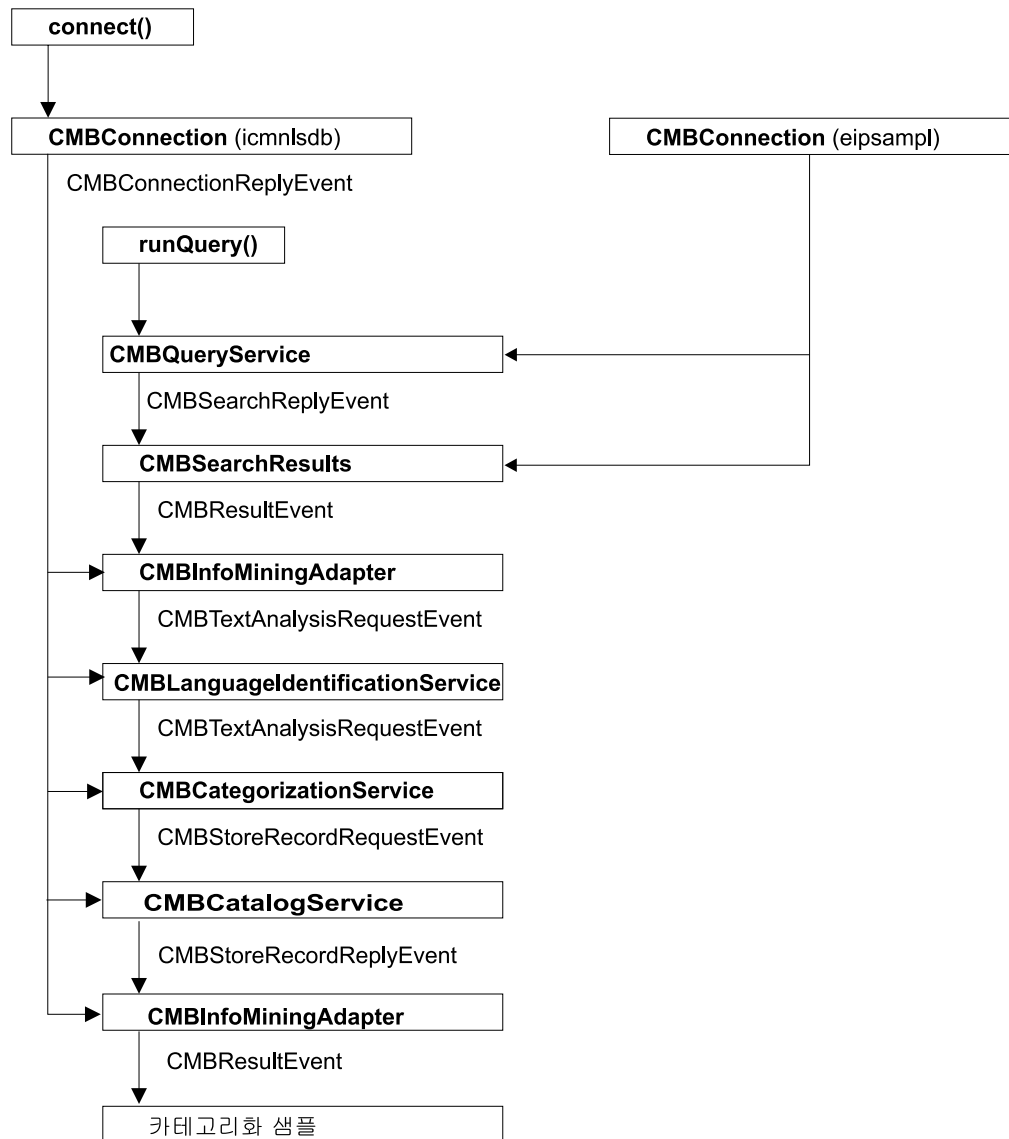


그림 42. 카테고리 샘플: 이벤트 플로우

이 샘플에 사용된 다섯 개의 Bean은 연결 핸들을 가져오기 위해 **CMBConnectionReplyEvent**를 청취합니다. **CMBQueryService** Bean은 이벤트를 생성하는 검색을 시작하고 다른 Bean을 통해 이벤트 플로우를 시작합니다.

Bean을 연결하는 코드는 다음과 같습니다.

```

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);

samlConnection.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);

```

또한 예외가 이벤트로서 전송되므로 카테고리 클래스는 CMBExceptionListener 인터페이스를 구현하여 적절한 이벤트를 처리해야 하고 예외 발생을 알려줄 Bean에 연결됩니다.

팁

카테고리 서비스 Bean과 요약 서비스 Bean을 차례로 결합하여 문서를 분석하는데 사용할 수 있습니다. 이를 수행하는 방법에 대한 자세한 정보는 541 페이지의 『웹 공간에서 문서 가져오기』를 참조하십시오.

조회 실행

조회를 실행하려면 먼저 CMBConnection Bean에서 연결 메소드를 호출하여 콘텐츠 서버에 대한 연결을 설정해야 합니다.

```

samlConnection.connect();
eipConnection.connect();

```

표준 EIP 검색을 시작하려면 검색 템플릿, 템플릿의 기준, 비교 연산자 및 검색값을 지정해야 합니다.

구성에 따라 다음 코드를 수정해야 합니다.

```

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,
    CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);
```

현재 연결을 닫으려면 연결 해제 메소드를 호출하십시오.

```
samplConnection.disconnect();
eipConnection.disconnect();
```

텍스트 분석 결과 표시

카테고리 클래스는 검색 중에 찾은 문서를 나열하고 각 문서에 작성된 카테고리 정보를 표시할 수 있도록 **CMBResultListener** 인터페이스를 구현합니다. **CMBResult** 메소드에 대한 인수로 수신된 **CMBResultEvent**에는 **CMBItem** 오브젝트 벡터가 들어 있으며, 각 **CMBItem** 오브젝트는 문서를 나타냅니다.

```
Vector cmbItemVector = (Vector)e.getData();
```

CMBItem 오브젝트는 문서의 PID를 캡슐화합니다.

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
CMBRecord currentRecord = currentItem.getInfoMiningRecord();
System.out.println("\n\nPID          : " + currentRecord.getPID());
```

또한 텍스트 분석 결과도 캡슐화합니다.

```
System.out.println("Categories : " + currentRecord.getValue("IKF_CATEGORIES"));
```

문서 요약

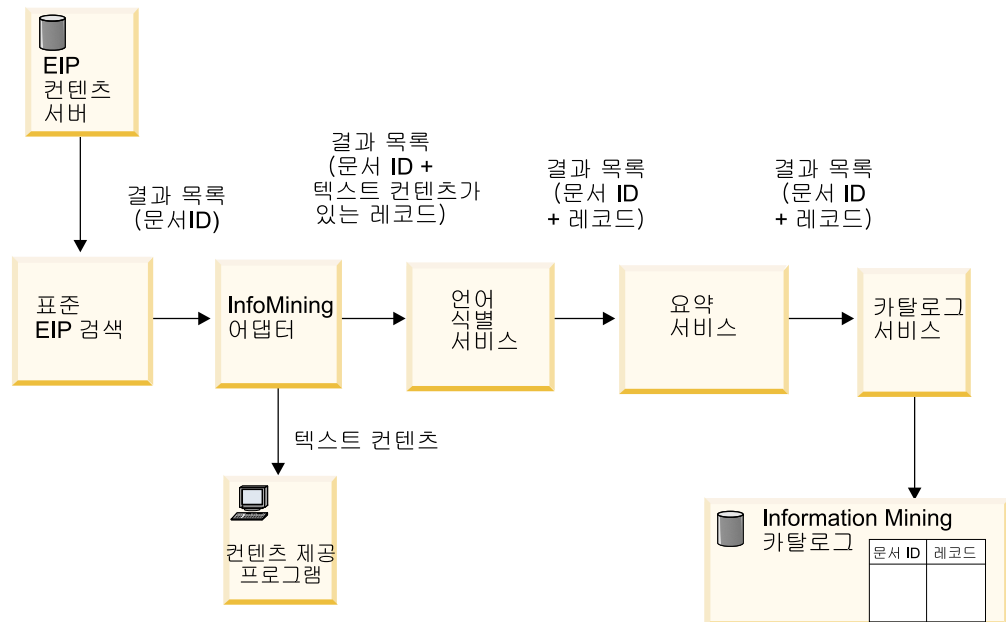


그림 43. 요약 샘플

이 샘플에서는 표준 EIP 검색에서 검색된 문서를 요약할 수 있는 방법에 대해 설명합니다. 분석 결과는 저장되며 해당 문서는 고급 검색에 사용할 수 있게 됩니다.

524 페이지의 그림 43에서는 EIP 콘텐츠 서버에서 보유하는 문서에 대해 표준 EIP 검색이 수행됩니다. 언어 식별 서비스 Bean은 문서가 작성되는 언어를 판별합니다. Bean은 문서 ID를 가져가서 문서 콘텐츠를 검색한 후 콘텐츠를 분석하여 언어를 판별합니다. 요약 서비스 Bean은 영어 문서 ID를 가져가서 콘텐츠 제공자를 사용하여 찾은 문서의 콘텐츠를 검색하고 이 문서에 대한 요약을 작성합니다. 그런 다음 카탈로그 서비스 Bean은 메타데이터 저장소에 요약 정보를 저장합니다. 그런 다음 이 정보(문서 ID 및 요약 정보)는 후속 처리에 사용할 수 있습니다. 이 샘플에서는 다음 Bean이 사용됩니다.

- CMBCConnection
- CMBQueryService(표준 EIP 검색 수행)
- CMBSearchResults(표준 EIP 검색 수행)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBSummarizationService
- CMBCatalogService

Summarization.java의 완전한 원본

다음은 요약 샘플의 완전한 원본입니다. 요약 및 카테고리 Bean은 유사한 방식으로 작동하므로 자세한 정보는 516 페이지의 『문서 구분』을 참조하십시오.

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBCConnection;
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      = "icmnlsdb";
    String DEFAULT_CMDBUSER      = "icmadmin";
    String DEFAULT_CMDBPASSWORD = "password";
    String DEFAULT_SAMPDBNAME    = "eipsamp1";
    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
    CMBCConnection sampConnection = new CMBCConnection();
    CMBCConnection eipConnection = new CMBCConnection();
    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
samlConnection.setServerName(sampDbName);
samlConnection.setUserid(sampUserName);
samlConnection.setPassword(sampPasswd);
samlConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samlConnection.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
samlConnection.addCMBConnectionReplyListener(queryService);
samlConnection.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);

samlConnection.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
samlConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
    getRootCategory().getPathAsString());

CMBSchemaManagement schema = samlConnection.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);

samlConnection.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();
    }
}
}

```

다음에 Bean 간의 이벤트 플로우가 나와 있습니다.

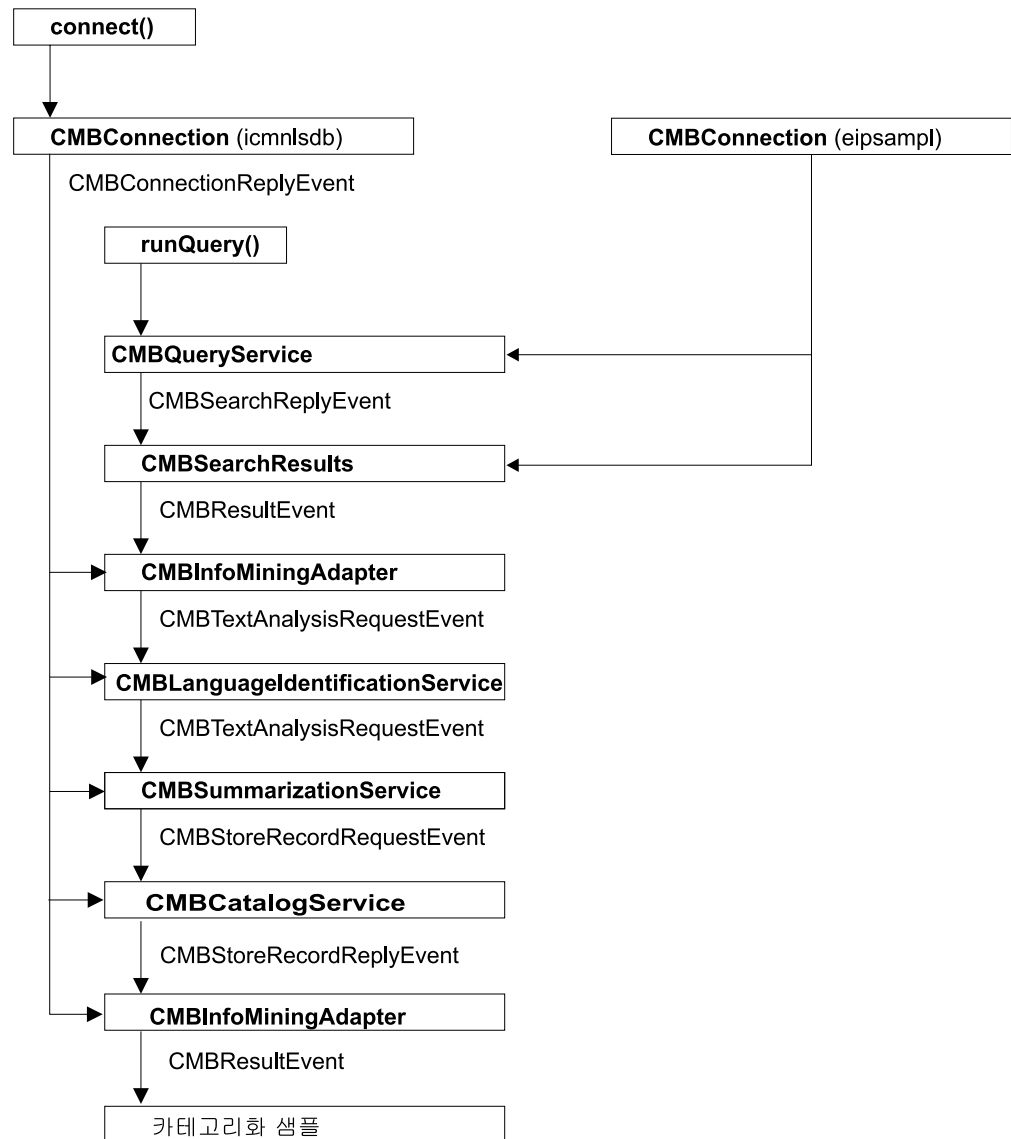


그림 44. 요약 샘플: 이벤트 플로우

정보 추출

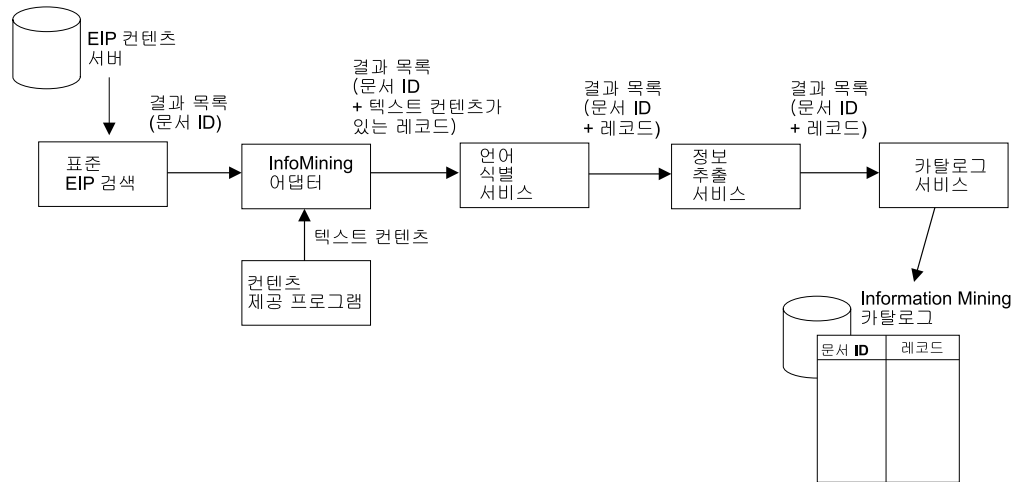


그림 45. 정보 추출 샘플

이 샘플에서는 표준 EIP 검색에서 검색된 문서로부터 이름, 용어 및 표현식과 같은 정보를 추출할 수 있는 방법에 대해 설명합니다. 분석 결과는 저장되며, 관련 문서를 검색하거나 개별 문서로부터 중요 정보를 추출하는 데 사용됩니다.

그림 45에서는 EIP 콘텐츠 서버에서 보유하는 문서에 대해 표준 EIP 검색이 수행됩니다. 언어 식별 서비스 Bean은 문서가 작성되는 언어를 판별합니다. Bean은 문서 ID를 가져가서 문서 콘텐츠를 검색한 후 콘텐츠를 분석하여 언어를 판별합니다. 정보 추출 서비스 Bean은 영어 문서를 가져와서 이러한 문서에서 정보를 추출하고 그 결과를 메타데이터 저장소에 저장합니다. 그런 다음 이 정보(문서 ID 및 추출된 정보)는 후속 처리에 사용할 수 있습니다.

이 샘플에서는 다음 Bean이 사용됩니다.

- CMBCConnection
- CMBQueryService(표준 EIP 검색 수행)
- CMBSearchResults(표준 EIP 검색 수행)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBInformationExtractionService
- CMBCatalogService

이 샘플에 대해 해당 응용프로그램은 다음을 수행합니다.

1. Bean을 작성합니다.
2. Bean을 사용자 조정합니다.

3. 언어 식별 서비스가 문서를 분석할 수 있도록 Bean을 연결합니다. 결과가 카탈로그에 저장됩니다.
4. 정보 추출 서비스는 문서를 분석합니다. 결과가 카탈로그에 저장됩니다.
5. 확인할 검색 결과(문서 목록) 및 추출된 정보를 표시합니다.

Java 원본은 다음과 같습니다. 정보 추출 및 카테고리 Bean은 유사한 방식으로 작동하므로 자세한 정보는 516 페이지의 『문서 구분』을 참조하십시오.

InformationExtraction.java의 완전한 원본

```
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    = "eipsamp1";
    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
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
samlConnection.addCMBConnectionReplyListener(queryService);
samlConnection.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);

samlConnection.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
samlConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
    getRootCategory().getPathAsString());

CMBSchemaManagement schema = samlConnection.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);

samlConnection.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();
    }
}
}

```

535 페이지의 그림 46에 Bean 간의 이벤트 플로우가 나와 있습니다.

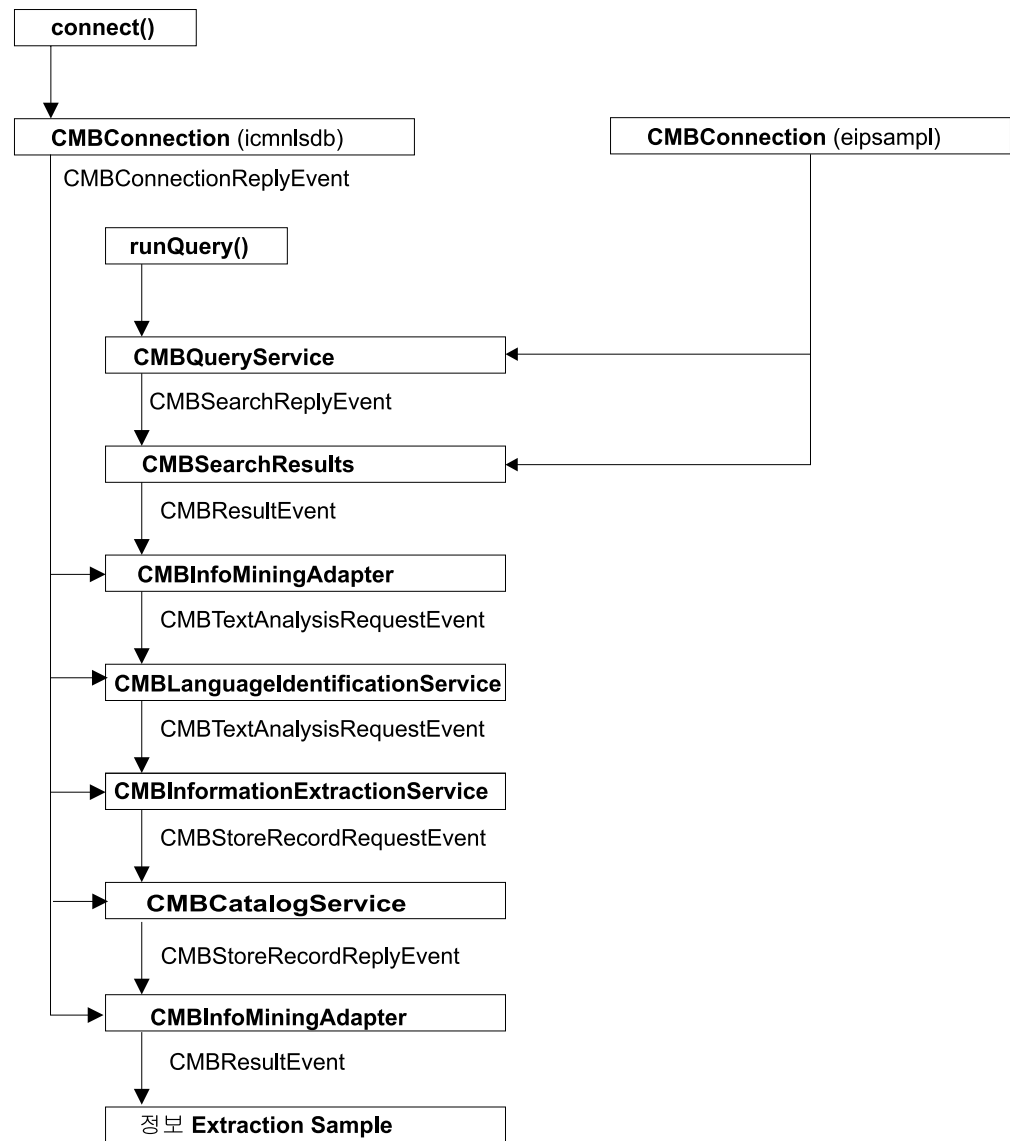


그림 46. 정보 추출 샘플: 이벤트 플로우

클러스터링

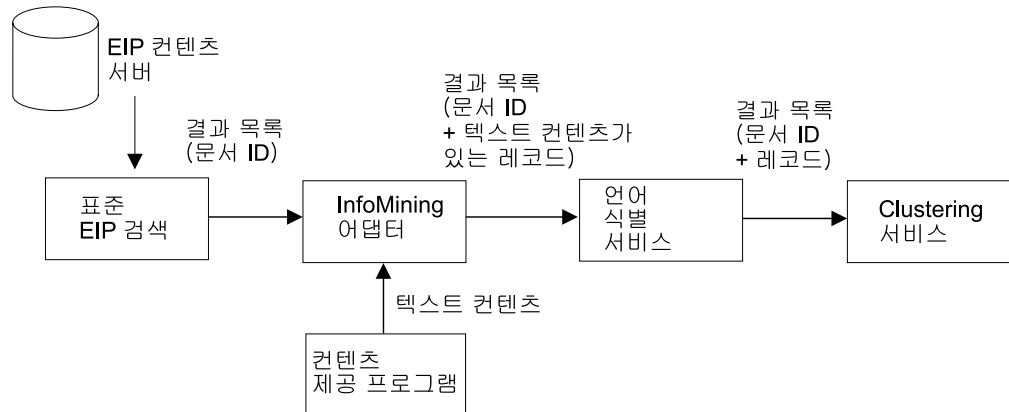


그림 47. 클러스터링 샘플

이 샘플에서는 표준 EIP 검색에서 검색된 문서를 클러스터링하는 방법에 대해 설명합니다. 클러스터링하기 전에 문서 언어를 식별해야 합니다. 클러스터링 서비스는 EIP 검색으로부터 리턴된 문서만 자동으로 수집합니다. 실제 클러스터링 프로세스는 cluster() 메소드를 호출하여 수동으로 트리거되어야 합니다.

이 샘플에서는 다음 Bean이 사용됩니다.

- CMBCConnection
- CMBQueryService(표준 EIP 검색 수행)
- CMBSearchResults(표준 EIP 검색 수행)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBClusteringService

이 샘플에 대해 해당 응용프로그램은 다음을 수행합니다.

1. Bean을 작성합니다.
2. Bean을 사용자 조정합니다.
3. 언어 식별 서비스가 문서를 분석하고 문서 언어를 식별할 수 있도록 Bean을 연결합니다. 그러면 clustering 서비스가 후속 clustering 준비가 된 문서를 수집할 수 있습니다.
4. cluster() 메소드를 호출하여 clustering을 트리거합니다. 그러면 (CMBClusterResult 클래스의) 결과가 읽기 가능한 양식으로 화면에 인쇄됩니다.

클러스터링 서비스용 Bean은 다음을 제외하고 기타 Information Mining Bean과 유사합니다.

- 클러스터링 서비스는 EIP 검색으로부터 리턴된 문서만 수집합니다. 실제 clustering 프로세스는 수동으로 트리거되어야 합니다.
- 카탈로그는 clustering 결과를 저장하도록 지정되지 않았습니다. 이는 카탈로그 서비스가 이 시나리오에 필요하지 않음을 의미합니다.

Bean 조작 방법에 대한 자세한 정보는 516 페이지의 『문서 구분』을 참조하십시오.

Java 원본은 다음과 같습니다.

Clustering.java의 완전한 원본

```
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      = "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 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("\n\nRunning 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
sampleConnection.setServerName(sampDbName);
sampleConnection.setUserid(sampUserName);
sampleConnection.setPassword(sampPasswd);
sampleConnection.setConnectionType(CMBCConnection.CMB_CONNTYPE_DYNAMIC);
sampleConnection.setServiceConnectionType(CMBCConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBCConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBCConnection.CMB_CONNTYPE_DYNAMIC);

```

```

adapter.setContentProvider(new CMBDefaultContentProvider());
    adapter.setCatalogName(catalog);
    clusteringService.setMinClusterCount(2);
    clusteringService.setMaxClusterCount(6);
    clusteringService.setClusterFeatureCount(5);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(clusteringService);

    queryService.addCMBSearchReplyListener(searchResults);
    searchResults.addCMBResultListener(adapter);
    adapter.addCMBTextAnalysisRequestListener(languageIdentificationService);
    languageIdentificationService.addCMBTextAnalysisRequestListener(clusteringService);

samplConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
    queryService.addCMBExceptionListener(this);
    searchResults.addCMBExceptionListener(this);
    languageIdentificationService.addCMBExceptionListener(this);
    clusteringService.addCMBExceptionListener(this);
    adapter.addCMBExceptionListener(this);

// running query
samplConnection.connect();
eipConnection.connect();

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_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();
        }
    }
}

```

그림 48에 Bean 간의 이벤트 플로우가 나와 있습니다.

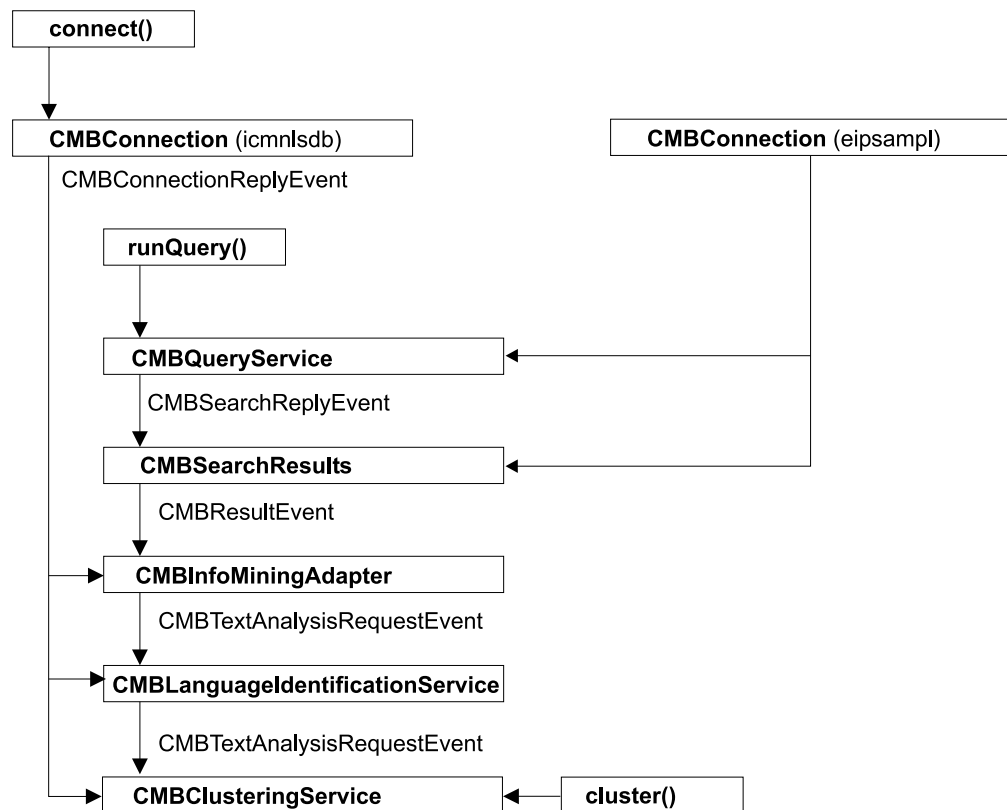


그림 48. 클러스터링 샘플: 이벤트 플로우

웹 공간에서 문서 가져오기

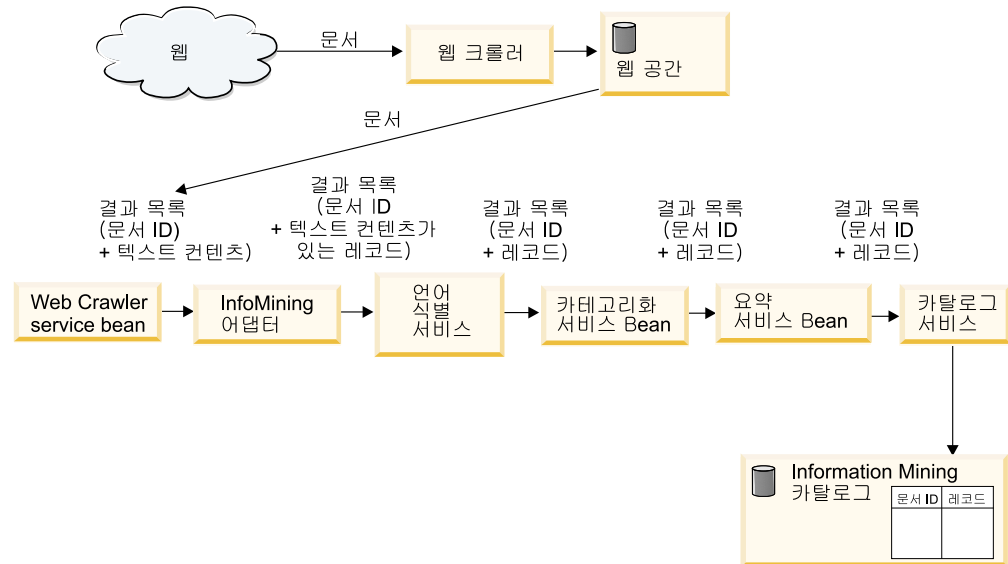


그림 49. 웹 크롤러 샘플

이 샘플에서는 검색된 문서를 Information Mining 구성요소로 가져오고 가져온 문서에서 텍스트 분석(언어 식별, 카테고리 및 요약)을 수행하는 방법에 대해 설명합니다. 이 방법은 웹 크롤러가 실행되었거나 실행 중인 경우, 그리고 새 오브젝트 또는 변경된 오브젝트가 웹 크롤러 구성의 summaries-dir 속성에 정의된 지정된 웹 공간 디렉토리 내에 저장된 경우에만 수행될 수 있습니다.

Information Mining에 대해 웹 크롤러를 사용하는 경우, *Enterprise Information Portal*의 Information Mining 관리 절에 설명된 Information Mining 구성 설정으로 실행되는지 확인하십시오. 이 설정은 다음 디렉토리의 Information Mining 구성요소 im-crawler-config-sample.xml에 대해 샘플 구성 파일의 부분입니다.

- Windows의 경우

```
<CMBROOT>\samples\java\beans\infomining\webcrawler\
```

- UNIX(AIX 및 Solaris)의 경우

```
<CMBROOT>/samples/java/beans/infomining/webcrawler/
```

웹 크롤러가 모니터한 웹 페이지 문서의 사본을 지정된 웹 공간 디렉토리에 배치하면 CMBWebCrawlerService Bean을 사용하여 검색된 문서를 Enterprise Information Portal Information Mining으로 가져올 수 있습니다.

웹 크롤러 및 CMBWebCrawlerService Bean을 지속적 프로세스로 실행하여 문서 변경 시 이 문서를 모니터하고 가져올 수 있습니다. 가져온 문서의 수가 특정 값을 초과하거나 모니터된 모든 파일을 가져올 때 CMBResultEvent를 트리거할 수 있습니다.

CMBWebCrawlerService Bean은 첨부된 모든 리스너에게 이를 알립니다. 이벤트에는 CMBItems의 벡터로 가져온 파일이 포함됩니다.

AIX 및 Solaris에서 CMBWebCrawlerService에 대한 액세스 권한.

CMBWebCrawlerService를 사용하여 문서를 올 때 해당 파일은 파일 시스템에서 삭제되거나, 아카이브 옵션이 **archiveEnabled** 사용 중 켜져 있는 경우 파일이 디스크 디렉토리에서 아카이브 디렉토리로 이동합니다. 아카이브 디렉토리는 디스크 디렉토리 (.../webspaces/ikf에서)와 동일한 레벨에 있으며 디스크 디렉토리와 동일한 서브디렉토리 구조를 가집니다. CMBWebCrawlerService를 사용하려면 해당 사용자 ID에 다음 권한이 있는지를 확인하십시오.

- 디스크 디렉토리, 이 디렉토리 아래의 모든 서브디렉토리 및 파일에 대한 쓰기 권한. 디스크 디렉토리의 위치는 웹 크롤러 구성 파일의 summaries-dir에 정의된 .../webspaces/ikf/disks입니다. 이 권한은 검색된 문서 및 파일을 삭제 또는 이동하는 데 필요합니다.
- 아카이브가 .../webspaces/ikf 디렉토리에 쓰기 권한을 설정한 경우, 처음 아카이브가 사용될 때 아카이브 디렉토리를 작성할 수 있습니다.
- 아카이브가 아카이브 디렉토리와 해당 서브디렉토리 및 모든 파일에 대한 쓰기 권한을 설정한 경우, 디스크 디렉토리에서 이동한 파일이 아카이브 디렉토리에서 작성될 수 있습니다.

가져오기 연산 수정. CMBWebCrawlerService Bean에서 일부 가져오기 연산을 변경할 수 있습니다. 처음에 이 프로그램은 30분마다 검색된 문서를 찾지만 이 값을 변경할 수 있습니다. 또한 100개의 문서를 가져온 후 모든 리스너에게 CMBResultEvent 공고를 발행합니다. 100을 다른 문서 수로 변경하거나, 검색된 모든 파일을 Enterprise Information Portal 세트에 가져올 때 공고를 발행할 수 있습니다.

CMBWebCrawlerService Java Bean의 등록 정보:

pollCycles

폴링 횟수.

pollMinutes

다음 폴 이전에 대기하는 시간(분 단위). 기본값은 30분입니다.

rootDir

로컬 호스트에서 %IMY_WEBSPACE%를 검색합니다.

archiveEnabled

.../webspaces/ikf/archives에 검색된 파일을 보관합니다. 기본값은 false로, 다른 곳에서 백업하지 않고 처리 중 CMBWebCrawlerService가 .../webspaces/ikf/disks에서 검색된 파일을 삭제하는 것을 의미합니다.

pageSize

CMBResultListeners로 CMBResultEvent를 발행할 때까지 가져온 항목 수

webSpace

웹 크롤러가 모니터하는 웹 공간

기타 텍스트 분석 Bean을 사용하면 웹 크롤러 검색 Bean 결과를 후속 고급 검색에 사용할 수 있습니다.

이 샘플에서는 다음 Bean이 사용됩니다.

- CMBCConnection
- CMBWebCrawlerService
- CMBInfoMiningAdapter
- CMBLanguageIdentification
- CMBCategorizationService
- CMBSummarizationService
- CMBCatalogService

이 샘플에 대해 해당 응용프로그램은 다음을 수행합니다.

1. Bean을 작성합니다.
2. Bean을 사용자 조정합니다.
3. Bean을 연결합니다.
4. 웹 크롤러 서비스를 시작합니다.
5. 크롤러 결과 및 텍스트 분석 결과를 표시합니다.

위의 각 단계에 대한 설명은 WebCrawler.java의 원본 다음에 설명됩니다.

WebCrawler.java의 완전한 원본

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBCConnection;
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("WebSpace directory = " + webSpaceDir);
    System.out.println("WebSpace name    = " + webSpaceName);
    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(webSpaceDir);
    crawlerService.setWebSpace(webSpaceName);
}

```

```

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// connecting beans
connection.addCMBCConnectionReplyListener(crawlerService);
connection.addCMBCConnectionReplyListener(adapter1);
connection.addCMBCConnectionReplyListener(languageIdentificationService);
connection.addCMBCConnectionReplyListener(categorizationService);
connection.addCMBCConnectionReplyListener(summarizationService);
connection.addCMBCConnectionReplyListener(catalogService);
connection.addCMBCConnectionReplyListener(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 작성

보안상의 이유로 콘텐츠 서버에 대한 연결을 빌드합니다. Information Mining은 시스템에서 누가 작업하는지 알아야 합니다. 이는 시스템에 등록된 사용자만이 시스템을 사용하도록 합니다. CMBConnection Bean을 사용하여 연결을 설정할 수 있습니다. Bean CMBWebCrawlerService는 이전에 검색된 문서를 Information Mining 구성요소로 가져오고 CMBResultEvent 공고를 발행하는 데 사용합니다. 언어 식별, 요약 및 카테고리 정보는 Bean CMBLanguageIdentificaton, CMBSummarizationService 및 CMBCategorizationService를 사용하여 작성됩니다. 두 개의 어댑터는 결과 이벤트를 텍스트 분석 요청 이벤트로 변환한 후 레코드 응답 이벤트를 검색 결과 이벤트에 다시 저장합니다.

Bean을 작성하는 코드는 다음과 같습니다.

```

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 사용자 조정

사용자 조정 절에 표시된 코드는 설치에 따라 수정해야 합니다. 연결할 콘텐츠 서버의 이름, 사용자 ID 및 적절한 암호를 지정해야 합니다.

웹 크롤러 서비스 Bean에서는 웹 크롤러가 웹 공간 내에서 검색된 오브젝트를 저장하거나 변경한 로컬 호스트의 루트 디렉토리를, 이전에 정의된 웹 공간 이름과 함께 지정해야 합니다. 텍스트 분석 서비스 Bean과 카탈로그 서비스 Bean을 실행하려면 먼저 이 Bean을 기존의 카탈로그와 연관시켜야 합니다.

샘플에 대해 사용자 조정되는 코드는 다음과 같습니다.

```

        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 연결

548 페이지의 그림 50에 Bean 간의 이벤트 플로우가 나와 있습니다.

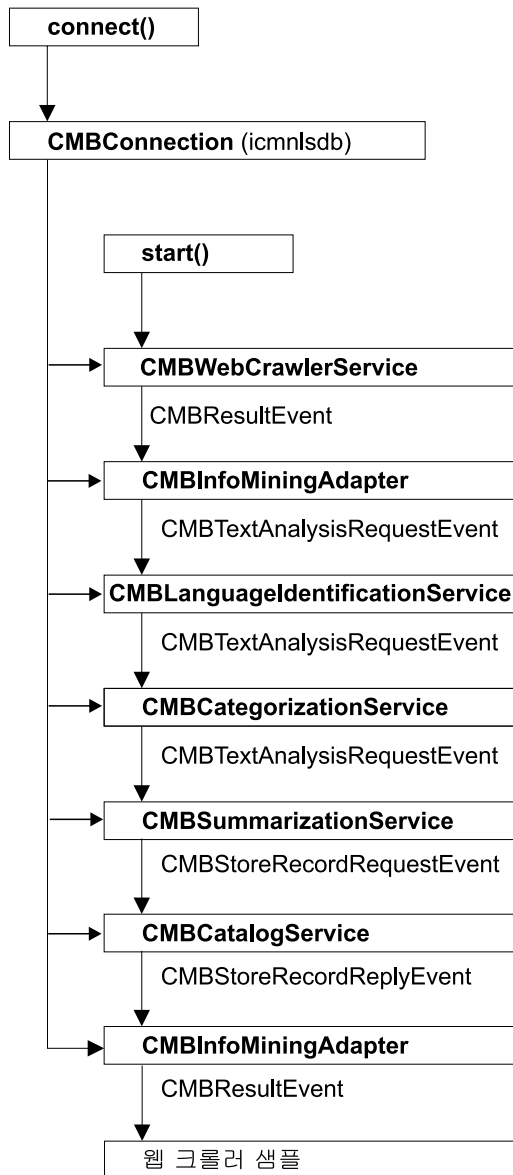


그림 50. 웹 크롤러 샘플: 이벤트 플로우

이 샘플에 사용된 다섯 개의 Bean은 연결 핸들을 가져오기 위해 `CMBConnectionReplyEvent`를 청취합니다. `CMBWebCrawlerService`는 이벤트를 생성하는 검색 서비스를 시작하고 다른 Bean을 통해 이벤트 플로우를 시작합니다.

Bean을 연결하는 코드는 다음과 같습니다.

```

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);

```

또한 예외는 이벤트로서 전송되므로 WebCrawler 클래스는 CMBExceptionListener 인터페이스를 구현하여 적절한 이벤트를 처리해야 하고 예외 발생을 알려줄 Bean에 연결됩니다.

웹 크롤러 서비스 시작

웹 크롤러 서비스를 시작하려면 먼저 CMBCConnection Bean에서 연결 메소드를 호출하여 콘텐츠 서버에 대한 연결을 설정해야 합니다.

```
connection.connect();
```

웹 크롤러 서비스를 시작하는 코드는 다음과 같습니다.

```

catalogService.setDefaultCategoryPath
    (catalogService.getTaxonomy().getRootCategory().getPathAsString());
crawlerService.start();

```

현재 연결을 닫으려면 disconnect() 메소드를 호출하십시오.

```
connection.disconnect();
```

텍스트 분석 결과 표시

WebCrawler 클래스는 검색 중에 찾은 문서를 나열하고 각 문서에 작성된 카테고리 및 요약 정보를 표시할 수 있도록 CMBResultListener 인터페이스를 구현합니다. onCMBResult 메소드에 인수로 수신된 CMBResultEvent에는 CMBItem 오브젝트의 벡터가 들어 있으며, 각 CMBItem 오브젝트는 문서를 나타냅니다.

```
Vector cmbItemVector = (Vector)e.getData();
```

CMBItem 오브젝트는 문서의 PID를 캡슐화합니다.

```

CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

CMBRecord currentRecord = currentItem.getInfoMiningRecord();

System.out.println("\n\nPID          : " + currentRecord.getPID());

```

또한 텍스트 분석 결과도 캡슐화합니다.

그런 다음 요약 및 카테고리 정보를 가져옵니다.

```

        System.out.println("Categories    : " +
                           currentRecord.getValue("IKF_CATEGORIES"));
        System.out.println("Summary      : " +
                           currentRecord.getValue("IKF_SUMMARY"));

```

카테고리별 문서 검색

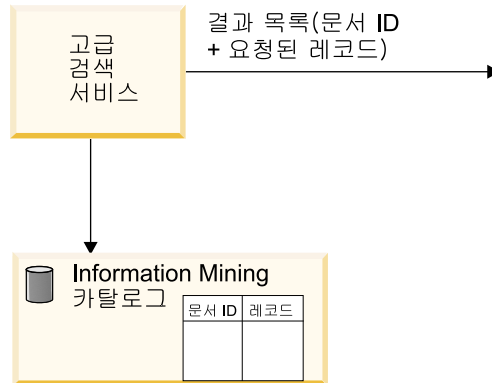


그림 51. 고급 검색 샘플

이 샘플에서는 고급 검색을 실행하고 찾은 문서에서 카테고리 정보를 찾는 방법에 대해 설명합니다. 고급 검색은 카테고리 샘플에서처럼 해당 유형의 검색에 사용할 수 있는 문서만 찾을 수 있습니다. 515 페이지의 『샘플 파일의 위치』를 참조하십시오. 표준 EIP 검색이 전체 EIP 콘텐츠 서버에서 수행되던 이전 샘플과 달리, 이 샘플에서는 데이터스토어에 있는 메타데이터를 검색합니다.

복합 조회 제출 시 성능 문제점이 발생하는 경우, 자세한 정보는 568 페이지의 『성능 조정』을 참조하십시오.

고급 검색 결과 목록이 작성될 때, 찾은 문서의 ID는 이전에 데이터스토어에 저장되었던 메타데이터를 검색하기 위해 카탈로그 서비스 Bean에서 사용할 수 있습니다.

이 샘플에서는 다음 Bean이 사용됩니다.

- CMBConnection
- CMBAdvancedSearchService
- CMBInfoMiningAdapter
- CMBCatalogService

이 샘플에 대해 해당 응용프로그램은 다음을 수행합니다.

1. Bean을 작성합니다.
2. Bean을 사용자 조정합니다.
3. Bean을 연결합니다.
4. 고급 검색을 실행합니다.

5. 확인할 검색 및 텍스트 분석 결과를 표시합니다.

위의 각 단계에 대한 설명은 AdvancedSearch.java의 원본 다음에 설명됩니다.

AdvancedSearch.java의 완전한 원본

```
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      = "icmn1sdb";
    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 작성

검색을 수행하려면 콘텐츠 서버에 연결해야 합니다. **CMBConnection** Bean을 사용하여 연결을 설정할 수 있습니다. 고급 검색을 수행하려면 **CMBAdvancedSearchService** Bean이 필요합니다. **CMBCatalogService** Bean을 사용하여 카테고리 정보를 검색할 수 있습니다.

Bean을 작성하는 코드는 다음과 같습니다.

```

CMBConnection connection = new CMBConnection();
CMBAdvancedSearchService advancedSearchService = new CMBAdvancedSearchService();
catalogService = new CMBCatalogService();

```

Bean 사용자 조정

사용자 조정 절에 표시된 코드는 설치에 따라 수정해야 합니다. 연결할 콘텐츠 서버의 이름, 사용자 ID 및 적절한 암호를 지정해야 합니다.

고급 검색 서비스와 카탈로그 서비스 Bean을 실행하려면 먼저 이 Bean을 기존의 카탈로그와 연관시켜야 합니다.

샘플에 대해 사용자 조정되는 코드는 다음과 같습니다.

```
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 + '\"')");
```

Bean 연결

그림 52에 Bean 간의 이벤트 플로우가 나와 있습니다.

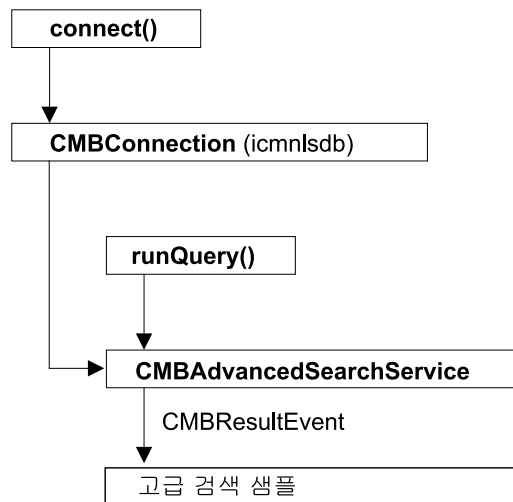


그림 52. 고급 검색 샘플: 이벤트 플로우

이 샘플에 사용된 두 개의 Bean은 연결 핸들을 가져오기 위해 CMBConnectionReplyEvent를 청취합니다. CMBAdvancedSearchService Bean은 이벤트를 생성하는 검색을 시작하고 다른 Bean을 통해 이벤트 플로우를 시작합니다.

Bean을 연결하는 코드는 다음과 같습니다.

```
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
```

또한 예외는 이벤트로서 전송되므로 AdvancedSearch 클래스는 CMBExceptionListener 인터페이스를 구현하여 적절한 이벤트를 처리해야 하고 예외 발생을 알려줄 Bean에 연결됩니다.

조회 실행

조회를 실행하려면 먼저 CMBCConnection Bean에서 연결 메소드를 호출하여 콘텐츠 서버에 대한 연결을 설정해야 합니다.

```
connection.connect();
```

고급 검색을 시작하려면 검색할 카탈로그, 검색 조회 문자열 및 메타데이터를 지정해야 합니다.

고급 검색을 시작합니다.

```
advancedSearchService.runQuery();
```

현재 연결을 닫으려면 disconnect() 메소드를 호출하십시오.

```
connection.disconnect();
```

텍스트 분석 결과 표시

AdvancedSearch 클래스는 검색 중에 찾은 문서를 나열하고 각 문서에 대해 검색된 카테고리 정보를 표시할 수 있도록 CMBResultListener 인터페이스를 구현합니다. onCMBResult 메소드에 인수로 수신된 CMBResultEvent에는 CMBItem 오브젝트의 벡터가 들어 있으며, 각 CMBItem 오브젝트는 문서를 나타냅니다.

```
Vector cmbItemVector = (Vector)e.getData();
```

CMBItem 오브젝트는 문서의 PID를 캡슐화합니다.

```
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());
```

이벤트 플로우에 텍스트 분석 결과가 있으면 이 결과도 캡슐화합니다.

그런 다음 카테고리 정보를 가져옵니다.

```
System.out.println("Categories      :  
                  " + currentRecord.getValue("IKF_CATEGORIES"));
```

CMBItem 클래스의 `getCategories()` 메소드에서 리턴된 벡터는 현재 카테고리에 대한 절대 경로를 결정하는 데 사용할 수 있는 `CMBCategory` 클래스의 오브젝트를 포함합니다.

사용자의 콘텐츠 제공자 빌드

사용자가 독점 필터를 적용해야만 독점 형식의 2진 부분에서 텍스트를 검색할 수 있을 경우, 사용자의 콘텐츠 제공자를 빌드하려고 할 수 있습니다.

이 절에서는 사용자 정의 오브젝트 모델이나 `CMBItem`의 부분 내부의 독점 형식으로 처리할 수 있는 사용자의 콘텐츠 제공자를 작성하는 방법에 대해 설명합니다. 이러한 작업에서 사용자를 지원하기 위해 `Information Mining`은 다음을 제공합니다.

- 텍스트 분석에 사용될 텍스트를 결정하는 방법을 알고 있는 클래스에 대한 인터페이스를 정의하는 인터페이스 **`CMBCContentProvider`**를 제공합니다.
- `ContentProvider`를 설정하는 `CMBInfoMiningUtilities`에서 **`setContentProvider(CMBCContentProvider)`** 메소드를 제공합니다.

`CMBCContentProvider` 인터페이스는 텍스트 분석에 사용할 지정된 항목의 텍스트를 리턴하는 하나의 `getContent()` 메소드를 정의합니다. 예를 들면, 다음과 같습니다.

```
public CMBTextAnalysisDocument getContent(CMBConnection connection, CMBItem item)
throws CMBCContentProviderException;
```

- 매개변수 `connection`: 서버에 대한 개방형 연결.
- 매개변수 `item`: 처리될 현재 항목.
- 예외 `CMBCContentProviderException`: 현재 항목 처리 중 오류가 발생하는 경우.
- 클래스 `CMBTextAnalysisDocument`의 오브젝트로 텍스트 리턴

시스템에 사용할 `ContentProvider`를 알려려면 `CMBInfoMiningUtilities` 클래스의 `setContentProvider(CMBCContentProvider)` 메소드를 사용하여 이 인터페이스가 있는 오브젝트를 지정하십시오. 예제는 다음과 같습니다.

```
CMBInfoMiningUtilities.setContentProvider
    (new MyCompaniesLatestGreatestContentProvider());
```

사용자의 콘텐츠 제공자를 개발하기 위해 시작점으로 샘플 콘텐츠 제공자 (`SimpleContentProvider`)를 사용할 수 있습니다. 콘텐츠 제공자 샘플은 `<CMBROOT>\samples\java\beans\infomining\contentprovider`에 있습니다.

기본 콘텐츠 제공자는 `Information Mining`과 함께 제공됩니다. `CMBCContentProvider` 인터페이스를 확장하여 처리할 개별 부분을 선택하고 비디오 스트림과 같이 인메모리로 처리하기에는 너무 큰 오브젝트를 처리하지 못하게 하는 메커니즘을 제공합니다.

기본 콘텐츠 제공자를 등록하려면 다음을 사용하십시오.

```
CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
```


서비스 API 사용

Information Mining 서비스 API는 Information Mining 기능을 EIP 서비스로 통합하는 Java API입니다. Information Mining 서비스 API를 사용하여 다음을 수행할 수 있는 응용프로그램을 작성할 수 있습니다.

- 다른 문서 형식(예: pdf, Microsoft Word 및 HTML)에서 텍스트 콘텐츠 필터링
- 텍스트 문서에 대한 텍스트 분석을 수행하여 요약 및 카테고리 같은 메타데이터 작성
- 지속적 레코드에 문서의 메타데이터 저장
- 레코드 검색

주

Information Mining Bean을 사용하기 전에 서비스 API와 JavaBeans 사이의 차이점을 이해하는 것이 중요합니다.

- JavaBeans는 신속한 응용프로그램 개발을 위한 기능을 제공합니다. 코드의 일부 요소는 ‘함께 묶여’ 있으므로 사용자가 변경할 수 없습니다. 자세한 정보는 511 페이지의 『Bean을 사용하여 Information Mining 응용프로그램 빌드』를 참조하십시오.
- Information Mining 서비스 API는 Information Mining 응용프로그램 빌드를 위한 추가 융통성을 제공합니다. 코드는 함께 결합되어 사용자별 요구사항을 충족시킬 수 있는 개별 ‘빌딩 블록’으로 볼 수 있습니다.

Information Mining 서비스 API에 연결

Information Mining 서비스 API를 사용하려면 먼저 DKIKFServiceFed 클래스를 사용하여 서비스 오브젝트를 작성해야 합니다. 원하는 응용프로그램 유형에 따라 다음 세 가지 패키지 중 하나로부터 클래스를 가져와야 합니다.

- 서버 호스트에서 응용프로그램을 작성하려면 `import com.ibm.mm.sdk.server.DKIKFServiceFed;`와 같은 import문을 사용하십시오.
- 클라이언트 호스트에서 응용프로그램을 작성하려면 `import com.ibm.mm.sdk.client.DKIKFServiceFed;`와 같은 import문을 사용하십시오.
- 융통성 있게 클라이언트뿐만 아니라 서버에서도 실행되는 응용프로그램을 작성하려면 `import com.ibm.mm.sdk.cs.DKIKFServiceFed;`와 같은 import문을 사용하십시오.

적절한 import문을 사용한 후에 다음을 사용하여 `ikfservice` 오브젝트를 작성하십시오.

```
DKIKFService ikfService = DKIKFServiceFed.create();
```

서비스 오브젝트를 작성한 후에 이름, 사용자 ID 및 암호를 사용하여 데이터베이스에 연결할 수 있습니다.

```
ikfService.connect("databaseName", "userID", "password", null);
```

이제 Information Mining 서비스 API에 연결되었으므로 다음 두 패키지로부터 클래스를 사용할 수 있습니다.

- **com.ibm.mm.sdk.common.infomining** - 라이브러리, 카탈로그 및 레코드 작업 관리
 - **com.ibm.mm.sdk.common.infomining.analysis** - 다른 텍스트 분석 도구 사용
- 이 주요 기능에 대한 설명은 다음 절을 참조하십시오.

연결을 해제하려면 다음을 사용하십시오.

```
ikfService.disconnect();
```

라이브러리, 분류 및 카탈로그 관리

라이브러리, 카탈로그 및 분류 작업 관리를 위한 클래스는 **com.ibm.mm.sdk.common.infomining** 패키지에 있습니다.

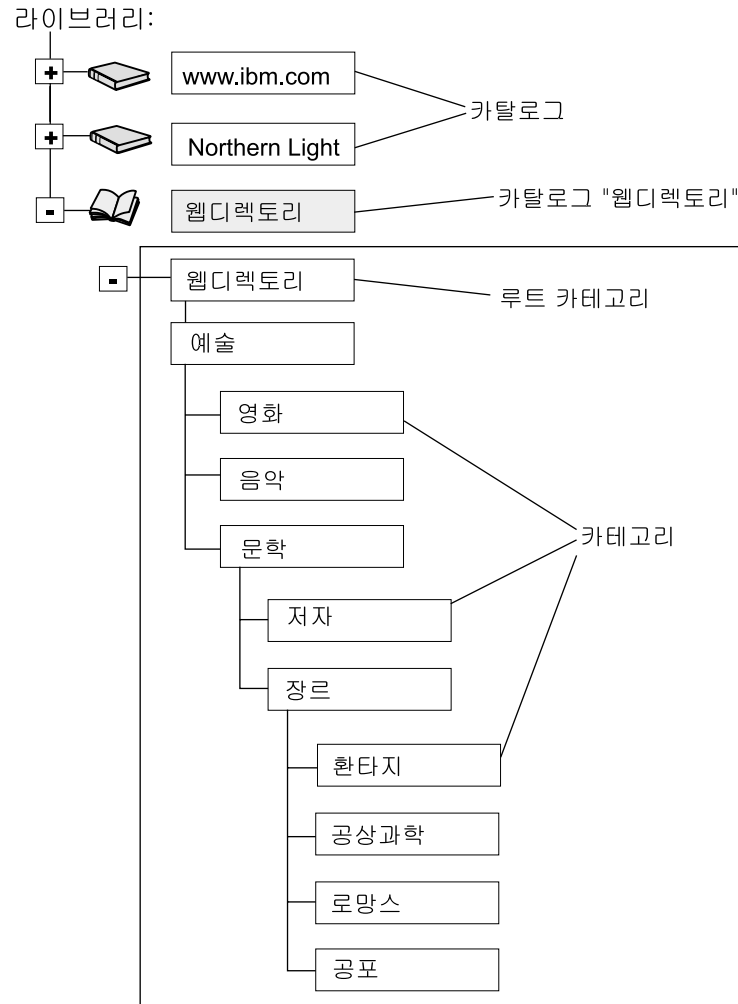


그림 53. 카탈로그 예제

카탈로그 예제에서 라이브러리에는 세 가지 카탈로그, 즉 `www.ibm.com`, `Northern Light` 및 `Webdirectory`가 포함되어 있습니다. `Webdirectory`에서는 도표에 카테고리 트리 구조가 표시됩니다.

카탈로그는 Information Structuring Tool(IST)에 의해서만 작성할 수 있습니다.

라이브러리 리턴

라이브러리 오브젝트를 얻으려면 다음을 사용하십시오.

```
DKIKFLibrary library = ikfService.getLibrary();
```

라이브러리에 모든 카탈로그 나열

라이브러리에 있는 모든 카탈로그 이름을 가져오려면 다음을 사용하십시오.

```
String[] catalogNames = library.getCatalogNames();
```

모든 카탈로그 이름을 나열하는 배열을 리턴합니다. 예제는 순서에 관계없이 `Webdirectory`, `www.ibm.com` 및 `Northern Light`와 같은 카탈로그 이름을 리턴합니다.

특정 카탈로그 리턴

특정 카탈로그를 얻으려면 다음을 사용하십시오.

```
DKIKFCatalog catalog = library.getCatalog(catalogNames[0]);
```

이 샘플에서는 이전 단계에서 리턴된 목록의 첫 번째 이름으로 카탈로그를 리턴합니다. 이 경우 Webdirectory 카탈로그가 리턴됩니다.

카탈로그 분류 리턴

카탈로그 분류를 가져오려면 다음을 사용하십시오.

```
DKIKFTaxonomy taxonomy = catalog.getTaxonomy();
```

분류 오브젝트에는 카테고리에 대해 작업할 수 있는 여러 가지 방법이 있습니다.

주

분류 오브젝트는 데이터베이스로부터 읽어들이며, 이 사본에 대해 이후의 모든 추가 조치가 수행됩니다.

분류에 모든 카테고리 나열

분류에 있는 모든 카테고리를 가져오려면 다음을 사용하십시오.

```
DKIKFCategory[] categories = taxonomy.getCategories();
```

루트 카테고리를 포함하여 모든 카테고리를 나열하는 배열을 리턴합니다. 예제는 순서에 관계없이 Horror, Author, Music, Literature, Webdirectory, Genres, Science Fiction, Fantasy, Movies, Romance 및 Arts와 같은 카테고리를 리턴합니다.

주

카테고리 오브젝트는 데이터베이스로부터 읽어들이며, 사본에 대해 이후의 모든 추가 조치가 수행됩니다.

루트 카테고리 리턴

루트 카테고리를 가져오려면 다음을 사용하십시오.

```
DKIKFCategory rootCategory = taxonomy.getRootCategory();
```

이 샘플에서는 이전 단계에서 리턴된 목록으로부터 루트 카탈로그를 리턴합니다. 이 경우 Webdirectory 루트 카테고리가 리턴됩니다.

특정 카테고리 리턴

특정 카테고리를 리턴하려면 다음을 사용하십시오.

```
DKIKFCategory category = taxonomy.getCategory("Webdirectory/Literature/Genres");
```

예제를 사용하면 카테고리 Genres가 리턴됩니다.

현재 분류 오브젝트에 설정된 동일한 분리 문자 "/"를 사용하십시오.

주

세 가지 getCategory 메소드가 분류 오브젝트에서 사용 가능하며, 각각 다른 매개변수가 필요합니다.

- 경로는 지정된 문자를 사용하여 구분되는 카테고리 이름이 있는 문자열입니다.
- 경로는 카테고리 이름의 배열입니다.
- 또다른 카테고리 오브젝트

자세한 정보는 온라인 API 참조서를 참조하십시오.

현재 분류가 최신인지 확인

현재 분류가 최신인지 확인하려면 다음을 사용하십시오.

```
if(taxonomy.getTimestamp().before(catalog.getTaxonomyLastModified()))
{
    taxonomy = catalog.getTaxonomy();
}
```

주

분류 오브젝트는 데이터베이스로부터 읽어들이며, 이 사본에 대해 이후의 모든 추가 조치가 수행됩니다. 따라서 분류 오브젝트가 최신인지 확인하십시오.

카테고리 하위 리턴

카테고리 하위를 리턴하려면 다음을 사용하십시오.

```
DKIKFCategory[] children = category.getChildren();
```

카테고리의 모든 하위를 나열하는 배열을 리턴합니다. Genres 카테고리를 기초로 한 예제는 순서에 관계없이 Fantasy, Science Fiction, Romance 및 Horror와 같은 하위를 리턴합니다.

주

카테고리에 대해 여러 하위가 있을 수 있지만 이러한 하위로부터 추가 서브카테고리를 리턴하지 않습니다.

카테고리 상위 리턴

카테고리 상위를 리턴하려면 다음을 사용하십시오.

```
DKIKFCategory parent = category.getParent();
```

Genres 카테고리를 기초로 한 예제에서는 Literature 카테고리만 리턴합니다.

특정 카탈로그 스키마를 리턴한 후 스키마에 모든 속성 나열

특정 카탈로그 스키마를 가져오려면 다음을 사용하십시오.

```
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());
}
```

예제에서는 Webdirectory 카탈로그의 스키마를 리턴합니다.

그런 다음 스키마로부터 키(속성 이름)를 리턴할 수 있습니다. 자세한 정보는 온라인 API 참조서를 참조하십시오.

주

스키마 오브젝트는 데이터베이스로부터 읽어들이며, 이 사본에 대해 이후의 모든 추가 조치가 수행됩니다.

Information Mining 도구 사용

Information Mining 도구를 사용할 클래스는 **com.ibm.mm.sdk.common.infomining.analysis** 패키지에 있습니다. 도구를 사용하여 다음을 수행하십시오.

- 문서 요약 생성
- 문서 카테고리 결정
- 문서 언어 결정
- 문서로부터 이름, 용어 및 표현식과 같은 정보 추출
- 문서의 세트 클러스터링

도구는 텍스트 문서를 처리하며, 이 문서는 DKIKFTextDocument 클래스의 오브젝트입니다. 텍스트 문서 오브젝트 작성 메소드에는 다음과 같이 두 가지가 있습니다.

- 문서 콘텐츠가 이미 Java 문자열로 존재하는 경우 DKIKFTextDocument 클래스의 메소드 작성
- 문서 콘텐츠가 형식화된 양식인 경우 DKIKFDocumentFilter 클래스

두 클래스 모두 **com.ibm.mm.sdk.common.infomining** 패키지에 있습니다.

텍스트 문서

문서 콘텐츠가 Java 문자열로 존재하는 경우 텍스트 문서 오브젝트를 작성하려면 다음을 사용하십시오.

```
DKIKFTextDocument document = DKIKFTextDocument.create("the document content");
```

주

세 가지 다른 create 메소드를 사용하여 다음을 수행할 수 있습니다.

- 지정된 콘텐츠로 텍스트 문서 작성
- 지정된 콘텐츠 및 이름으로 텍스트 문서 작성
- 지정된 콘텐츠, 이름 및 언어로 텍스트 문서 작성

자세한 정보는 온라인 API 참조서를 참조하십시오.

문서 필터링

형식화된 문서로부터 텍스트 문서 오브젝트를 작성하려면 다음을 사용하십시오.

```
DKIKFDocumentFilter documentFilter = new DKIKFDocumentFilter(ikfService);
byte[] documentBytes = ... //set the bytes of the document from a data source
DKIKFTextDocument document2 = documentFilter.getTextDocument(documentBytes);
```

원래 텍스트 문서는 지원되는 형식(예: pdf 또는 Microsoft Word 문서) 중 하나일 수 있다는 점에 유의하십시오.

필터 인코딩을 설정하려면 문서 필터 오브젝트 메소드를 사용하십시오. 자세한 정보는 온라인 API 참조서를 참조하십시오.

문서 언어 판별

문서의 언어를 판별하려면 다음을 사용하십시오.

```
DKIKFLanguageIdentifier languageIdentifier =
    new DKIKFLanguageIdentifier(ikfService);
DKIKFLanguageIdentificationResult[] languageResults =
    languageIdentifier.analyze(document);
document.setLanguage(languageResults[0].getLanguage());
```

언어 및 신뢰값 결과가 포함된 모든 오브젝트를 나열하는 최고 신뢰값이 먼저 나열됩니다.

주

요약 및 카테고리 서비스와 같은 다른 Information Mining 도구를 사용하려면 언어가 텍스트 문서에 설정되어야 합니다.

신뢰값 및 언어를 리턴하려면 언어 결과 오브젝트의 메소드를 사용하십시오.

```
string language = languageResults[0].getLanguage();  
float confidence = languageResults[0].getConfidence();
```

언어 식별 서비스 및 기본 설정에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

문서 요약 생성

문서 요약을 작성하려면 다음을 사용하십시오.

```
DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);  
DKIKFSummarizationResult summarizationResult = summarizer.analyze(document);
```

요약을 리턴하려면 다음을 사용하십시오.

```
string summary = summarizationResult.getSummary();
```

요약 서비스 및 기본 설정에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

문서 정보 추출

문서에서 정보를 추출하려면 다음을 사용하십시오.

```
DKIKFInformationExtractor extractor = new DKIKFInformationExtractor(ikfService);  
DKIKFInformationExtractionResult information = extractor.analyze(document);
```

추가 분석 기능을 표시하려면 정보 추출 결과 오브젝트 메소드를 사용하십시오.

```
DKIKFFeature[] features = information.getFeatures();
```

정보 추출 서비스 및 기본 설정에 대한 자세한 정보는 온라인 API 참조서를 참조하십시오.

클러스터링

유사한 문서가 다음을 사용하여 함께 그룹화되도록 문서를 클러스터링할 수 있습니다.

```
DKIKFClusterer clusterer = new DKIKFClusterer();  
clusterer.analyze(doc1);  
clusterer.analyze(doc2);  
clusterer.analyze(doc3);  
DKIKFClusterResult clusterResult = clusterer.cluster();
```

클러스터 노드, 클러스터 수, 총 문서 수를 리턴하려면 클러스터 결과 오브젝트 메소드를 사용하십시오.


```
int clusterCount = clusterResult.getClusterCount();
DKIKFClusterNode[] nodes = clusterResult.getClusterNodes();
int documentCount = clusterResult.getDocumentCount();
```

클러스터링 서비스 및 기본 설정에 대한 자세한 정보는 온라인 *API* 참조서를 참조하십시오.

구분

카테고리를 문서에 할당하려면 다음을 사용하십시오.

```
DKIKFCategorizer categorizer = new DKIKFCategorizer(catalog);
DKIKFCategorizationResult[] categorizationResults = categorizer.analyze(document);
```

카테고리 및 신뢰값이 포함된 모든 카테고리 결과 오브젝트를 나열하는 배열을 리턴합니다. 최고 신뢰값이 먼저 나열됩니다.

신뢰값 및 카테고리를 리턴하려면 카테고리 결과 오브젝트 메소드를 사용하십시오.

```
DKIKFCategory bestCategory = categorizationResults[0].getCategory();
```

주

카테고리 서비스를 사용하려면 먼저 Information Structuring Tool(IST)을 사용하여 카탈로그를 작성 및 연계해야 합니다. 카테고리 서비스는 하나의 카탈로그를 사용합니다.

리턴된 카테고리 오브젝트가 분류 오브젝트에 속하지 않습니다. 분류 메소드를 사용하여 상위 및 하위 카테고리를 탐색하십시오.

카테고리 서비스 및 기본 설정에 대한 자세한 정보는 온라인 *API* 참조서를 참조하십시오.

레코드 작성 및 카탈로그에 메타데이터 저장

레코드를 작성하고 메타데이터를 저장할 클래스는 **com.ibm.mm.sdk.common.infomining** 패키지에 있습니다. 이를 사용하여 다음을 수행하십시오.

- 카탈로그에 새 레코드 작성
- 카탈로그에서 레코드 검색
- 레코드의 카테고리 리턴
- 레코드 값 갱신
- 레코드 카테고리 할당 갱신
- 레코드 삭제

Information Mining 서비스 API는 Information Structuring Tool(IST)에 의해 작성된 카탈로그만 관리할 수 있습니다. 일단 작성되면 레코드 및 메타데이터를 추가할 수 있습니다.

카탈로그에 새 레코드 작성

PID 및 카탈로그 스키마를 사용하여 레코드가 작성됩니다.

```
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);
```

이 예제에서 레코드의 제목, 요약 및 신뢰값이 설정됩니다. 위에 설정된 값을 사용하여 레코드가 작성되고 하나 이상의 카테고리에 할당할 수 있습니다.

카탈로그에서 레코드 검색

PID를 사용하여 카탈로그에서 레코드를 검색하려면 다음을 입력하십시오.

```
DKIKFRecord retrievedRecord = catalog.getRecord("PID");
```

레코드의 카테고리 리턴

레코드의 카테고리를 리턴하려면 다음을 사용하십시오.

```
DKIKFCategory[] recordCategories = catalog.getCategoriesForRecord("PID");
```

카탈로그의 레코드 값 갱신

레코드 값(예: 레코드 제목)을 갱신하려면 다음을 사용하십시오.

```
record.setValue("IKF_TITLE", "This is the new title");
catalog.updateRecord(record);
```

주

세 가지 다른 update 메소드를 사용하여 다음을 수행할 수 있습니다.

- 레코드 값만으로 레코드 갱신(위의 제목 예제 참조)
- 레코드 값 및 할당된 카테고리로 레코드 갱신
- 할당된 카테고리로만 레코드 갱신(한 카테고리의 레코드가 다른 카테고리에도 추가되는 아래 예제를 참조하십시오.)

자세한 정보는 온라인 API 참조서를 참조하십시오.

다른 카테고리에 카테고리의 레코드 추가

레코드를 다른 카테고리에 추가하려면 다음을 사용하십시오.

```

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()]));

```

이 예제에서 레코드가 새 Movies 카테고리에 추가됩니다. 원래의 카테고리 할당을 보 관하려면 위 예제에서처럼 갱신 레코드 호출에 원래의 카테고리를 포함시켜야 합니다. 새 카테고리에 대한 갱신 레코드 호출만이 원래의 모든 카테고리 할당을 제거합니다.

레코드를 갱신하는 방법에는 세 가지가 있습니다. 자세한 정보는 566 페이지의 『카탈 로그의 레코드 값 갱신』을 참조하십시오.

레코드는 루트 카테고리 및 둘 이상의 카테고리에도 추가될 수 있습니다. 세부사항은 온 라인 API 참조서를 참조하십시오.

저장 레코드 삭제

레코드를 삭제하려면 다음을 사용하십시오.

```
catalog.deleteRecord("PID");
```

문서 검색

레코드를 검색할 클래스는 **com.ibm.mm.sdk.common.infomining** 패키지에 있습니다.

특정 단어가 포함된 레코드 찾기

예를 들어, "Music" 카테고리에서 "Bach" 단어가 포함된 레코드를 찾으려면 다음을 사 용하십시오.

```

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();

```

카탈로그 오브젝트에서 searchRecords 메소드를 실행하려면 다음이 필요합니다.

- 조회 문자열
- 검색 구성 오브젝트

검색 구성 오브젝트는 검색 등록 정보(예: 최대 검색 결과 수)를 지정합니다. 조회 문자 열에서 카테고리를 사용 중인 경우, 분류 오브젝트도 필요합니다.

검색 결과는 PID 문자열 또는 레코드일 수 있으며, DKIKFSearchConfiguration 호출 에 지정할 수 있습니다.

성능 조정

복합 조회 제출 시, 특히 조회 문자열이 여러 OR 연산자를 포함하는 경우 성능 문제점이 발생할 수 있습니다. 성능을 증가시키려면 deMorgan의 규칙을 적용하여 OR 연산자를 포함하는 복합 조회를 피하여 OR 표현식을 변환하십시오. 단항 + 및 - 연산자를 사용하는 복합 텍스트 검색 조회를 빌드하십시오. 이 표현식을 Information Mining 조회 언어로 변환할 클래스는 서비스 API를 사용하는 응용프로그램에 대한 com.ibm.mm.sdk.common.infomining.DKIKFWebQueryConverter에 있습니다. Bean을 사용하는 응용프로그램의 경우 CMBAdvancedSearchService.convertWebQuery 메소드를 사용하십시오.

서버 작업 실행

클라이언트 응용프로그램으로부터 정의된 순서로 작업을 실행하는 경우, 서버 작업에서 이들 호출을 번들하고, 이를 서버로 한 번 전송한 후, 클라이언트에서 필요한 만큼 자주 이 서버 작업을 호출할 수 있습니다. 그러면 네트워크 통신량 레벨이 가능한 한 낮게 유지되며 성능이 크게 향상됩니다.

서버 작업은 com.ibm.mm.sdk.common.infomining.DKIKFServerTask 인터페이스를 구현하는 오브젝트입니다. 오브젝트는 클라이언트 응용프로그램에서 인스턴스화되고 setServerTask 메소드를 사용하는 서비스 오브젝트에 설정된 후 runServerTask 메소드를 사용하여 서버에서 실행됩니다.

아래 예제에서 서버 작업 인터페이스를 구현하는 AnalysisTask 클래스의 오브젝트가 서비스에 설정된 후 실행됩니다.

```
...
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"));
}
...

```

문서의 맵핑이 처리를 위해 서버 작업으로 전달됩니다. 서버 작업은 작성된 메타데이터 (이 경우에는 문서 언어 및 요약)를 포함하는 지정된 문서에 대한 레코드 맵핑을 리턴합니다.

샘플 서버 작업의 원본은 다음과 같습니다.

```

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);
        }
    }
}

```

서버 작업은 레코드 작성에 필요한 스키마가 포함된 카탈로그 이름으로 인스턴스화됩니다. `runServerTask` 메소드는 스키마를 한 번만 검색합니다. 각 문서의 경우 레코드를 작성하고, 도구를 실행하여 문서를 분석하고, 작성된 메타데이터를 레코드에 저장합니다. 마지막으로, 작성된 모든 레코드가 호출자(클라이언트 응용프로그램)로 리턴됩니다.

주

위의 예제 코드에서 메타데이터만 리턴되며, 카탈로그에서는 레코드가 작성되지 않습니다.

JSP를 기준으로 한 Information Mining 응용프로그램 예제

Information Mining JSP(Java Server Page) 응용프로그램은 카테고리 내의 문서에 대해 Information Mining 구성요소를 검색합니다. 찾은 문서는 카테고리 구조로 표시됩니다.

JSP 샘플 응용프로그램은 다음 디렉토리에서 찾을 수 있습니다.

Windows	<CMBROOT>\samples\jsp\infomining\
AIX	/usr/lpp/cmb/samples/jsp/infomining/
Solaris	/opt/IBMcmb/samples/jsp/infomining/

이 디렉토리에는 다음 파일이 포함됩니다.

advSearch.jsp

샘플의 최상위 레벨 파일.

이 부분은 검색 고유의 고급 양식 처리 코드 및 양식 형식화 명령어(HTML)를 제공합니다. 또한 데이터의 보기를 선택하기 위한 제어기로 작동합니다. 이 파일은 고급 검색에만 해당하는 초기화 명령어와, 특히 검색할 수 있는 카테고리의 사용가능성을 포함합니다.

catView.jsp

이 부분은 보기 고유의 코드와 형식화 명령어(HTML)를 리턴된 결과의 카테고리 보기에 제공합니다. 리턴된 결과에서 반복할 루프가 포함되기도 하지만 대개 형식화 명령어를 포함합니다.

classes.jsp

이 부분은 논리 및 Bean 연결 코드를 제공합니다. 이 부분에는 Java 코드만 포함됩니다. 리턴된 결과를 보는 데 사용되는 간단한 데이터 구조 클래스를 구현합니다. 또한 리턴된 결과를 검색하고 조작하는 이벤트 핸들러도 구현합니다. 이것은 고급 검색에서 결과 목록이 리턴된 후 핵심 작업이 수행된 위치입니다.

logon.html

샘플을 실행하는 데 필요한 계정 및 카탈로그 입력.

계정 및 카탈로그 입력은 서버 이름, 사용자 이름, 암호 및 카탈로그 이름으로 구성되어 있습니다.

소스 코드는 Information Mining Bean을 사용하는 샘플입니다. 여기에는 코드 작동 방식, 즉 Bean 인스턴스화, Bean 연결, 이벤트 핸들러를 사용한 문서의 리턴 처리 등에 대한 설명이 들어 있습니다.

JSP가 실행하려면 Enterprise Information Portal 및 Information Mining 구성요소가 설치되어 있어야 합니다. 또한 JSP를 실행할 수 있는 웹 서버도 있어야 합니다.

JSP 응용프로그램에 대한 세부사항은 <http://java.sun.com/products/jsp/index.html> 웹 사이트를 참조하십시오.

JSP 설치

JSP를 전개하기 전에 IBM WebSphere Application Server(WAS)가 설치되어 실행 중인지를 확인하십시오.

JSP의 전개에 필요한 사용자 액세스 권한은 다음과 같습니다.

- Windows의 경우: 시스템 관리자 권한
- AIX의 경우: 루트 사용자 사용 권한
- Solaris의 경우: 루트 사용자 사용 권한

JSP는 <WAS_Home>\installedApps\JSP.ear\JSP.war 디렉토리의 웹 응용프로그램으로 전개됩니다. 샘플 JSP로 변경하는 경우, 작성한 JSP로 언급된 디렉토리에서 JSP를 대체하십시오. WAS는 자동으로 재컴파일합니다.

JSP용 WebSphere Application Server 구성 방법에 대한 정보는 *Enterprise Information Portal* 계획 및 설치를 참조하십시오.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서는 이 자료에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급하는 것이 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산을 침해하지 않는 한, 기능상 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

135-270

서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩

한국 아이.비.엠 주식회사

고객만족센터

전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이선스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

IBM World Trade Asia Corporation Licensing

2-31 Roppongi 3-chome, Minato-ku

Tokyo 106, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증없이 이 책을 『현상 태대로』 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책 사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및(또는) 프로그램을 사전 통지없이 언제든지 개선 및(또는) 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(1) 독립적으로 작성된 프로그램 및 기타 프로그램(본 프로그램 포함)간의 정보 교환 및
(2) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이선스 사용자는 다음 주소로 문의하십시오.

135-270

서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩

한국 아이.비.엠 주식회사

고객만족센터

이러한 정보는 해당 조항 및 조건(예를 들어, 사용료 지불 등)에 따라 사용할 수 있습니다.

이 정보에 기술된 라이선스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이선스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 레벨 상태의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고 보증할 수 없습니다. 또한, 일부 성능은 추정치일 수도 있습니다. 실제 결과는 다를 수 있습니다. 이 문서의 사용자는 해당 데이터를 사용자의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 비IBM 제품을 테스트하지 않았으므로, 이들 제품과 관련된 성능의 정확성, 호환성 또는 배상 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM의 향후 방향 또는 의도에 관한 모든 언급은 별도의 통지없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이 예제에는 개념을 완벽하게 설명하기 위해 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이선스:

이 정보에는 여러 가지 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원시 언어로 된 샘플 응용프로그램이 포함되어 있습니다. 귀하는 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스에 부합하는 응용프로그램을 개발, 사용, 마케팅 및 배포하기 위한 목적으로 이러한 샘플 프로그램을 IBM에 추가 비용없이 어떤 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이러한 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 암시할 수 없습니다. 귀하는 IBM의 응용프로그램 프로그래밍 인터페이스에 부합하는 응용프로그램을 개발, 사용, 마케팅 및 배포하기 위한 목적으로 이러한 샘플 프로그램을 IBM에 추가 비용없이 어떤 형태로든 복사, 수정 및 배포할 수 있습니다.

상표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

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	

Approach, Domino, Lotus, Lotus 1-2-3, Lotus Notes 및 SmartSuite는 미국 또는 기타 국가에서 사용되는 Lotus Development Corporation의 상표 또는 등록상표입니다.

Intel 및 Pentium은 미국 또는 기타 국가에서 사용되는 Intel Corporation의 상표 또는 등록상표입니다.

Microsoft, Windows 및 Windows NT는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 등록상표입니다.

Java 및 모든 Java 기반 상표와 로고는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표 또는 등록상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 Open Group의 등록상표입니다.

기타 회사, 제품 및 서비스 이름은 해당 회사의 상표 및 서비스표입니다.

용어집

이 용어집은 이 시스템에 고유한 용어 및 약어를 정의합니다. 기울임꼴로 표시된 용어는 이 용어집의 다른 곳에 정의되어 있습니다.

가

검색 기준(search criteria). Content Manager에서 저장된 항목을 검색하는 데 사용되는 속성값. Enterprise Information Portal에서 관리자가 사용자에게 사용 가능한 선택사항을 제한하거나 자세히 정의하는 검색 템플릿에 대해 정의하는 특정 필드.

검색 템플릿(search template). 관리자가 설계한 검색 기준으로 구성된 특정 유형의 연합 검색을 위한 양식. 시스템 관리자는 각 검색 템플릿에 액세스할 수 있는 사용자 및 사용자 그룹을 식별합니다.

게이트웨이(gateway). 다른 네트워크 아키텍처를 가진 두 개의 컴퓨터 네트워크를 상호 연결하는 기능적 단위. 게이트웨이는 다른 아키텍처를 가진 네트워크 또는 시스템을 연결합니다. 브릿지는 동일하거나 유사한 아키텍처를 가진 네트워크 또는 시스템을 상호 연결합니다.

결합 검색(combined search). 매개변수식, 텍스트 또는 이미지와 같은 하나 이상의 검색 유형이 결합된 조회.

관리 클래스(management class). API에서 이주 방함에 사용되는 용어.

교환(interchange). CIF(Common Interchange File)와 CIU(Common Interchange Unit)를 사용하여 하나의 OS/390용 Content Manager ImagePlus 시스템에서 다른 ImagePlus 시스템으로 색인과 함께 이미지를 가져오거나 내보내는 기능.

구성요소(component). 루트 구성요소 또는 하위 구성요소의 일반 용어.

근거리 통신망(LAN: Local Area Network). 장치 세트가 통신을 위해 다른 장치 세트에 연결되고 보다 큰 네트워크에 연결할 수 있는 네트워크.

기가바이트(gigabyte: GB). (1) 프로세서 기억영역, 실제 및 가상 기억영역, 채널 볼륨의 경우, 2^{30} 또는 1 073 741 824바이트입니다. (2) 디스크 기억영역 용량 및 통신 볼륨의 경우, 1 000 000 000바이트입니다.

기능(feature). 이미지 검색 서버에 저장된 비주얼 콘텐츠 정보. 또한 이미지 검색 응용프로그램이 일치 여부를 결정할 때 사용하는 비주얼 특성. QBIC 기능에는 평균 색, 히스토그램 색, 위치 색 및 텍스처의 네 가지가 있습니다.

기본 속성(base attributes). 각 오브젝트에 할당된 색인 세트. 모든 Content Manager 오브젝트에는 기본 속성이 있습니다.

기본 행수(cardinality). 데이터베이스 테이블의 행 수.

기억영역 그룹(storage group). 기억영역 시스템을 기억영역 클래스에 연관시킵니다.

기억영역 시스템(storage system). Content Manager 시스템의 기억영역에 대한 일반 용어. TSM 볼륨, MediaArchiver 및 볼륨을 참조하십시오.

기억영역 클래스(storage class). 오브젝트가 저장되는 미디어의 유형을 식별합니다. 실제 위치와 직접 연관되지는 않지만 장치 관리되는 직접 연관됩니다. 기억영역 클래스 유형은 다음과 같습니다.

DASD

고정 디스크

광학

스트림

테이프

TSM

나

네트워크 테이블 파일(network table file). Content Manager 시스템의 각 노드에 대한 시스템 고유의 구성 정보가 들어 있는 텍스트 파일. 시스템의 각 노드에는 노드를 식별하고 연결에 필요한 노드를 나열하는 네트워크 테이블 파일이 있어야 합니다.

네트워크 테이블의 이름은 FRNOLINT.TBL입니다.

다

데이터 형식(data format). MIME 유형을 참조하십시오.

데이터스토어(datastore). (1) 데이터가 저장되는 위치(예: 데이터베이스 시스템, 파일 또는 디렉토리)에 대한 일반 용어. (2) 응용프로그램에서 콘텐츠 서버의 가상 표현.

독립형 시스템(stand-alone system). 한 대의 개인용 컴퓨터에 Content Manager 시스템의 모든 구성요소를 설치하는 미리 구성된 Content Manager 시스템.

등급(rank). 조회 결과에 대한 주어진 부분의 관련도를 나타내는 정수값. 등급이 높을수록 일치도가 높습니다.

등록 정보(property). 오브젝트에 대해 설명하는 오브젝트의 특성. 등록 정보를 변경하거나 수정할 수 있습니다. 등록 정보의 예로는 유형 양식이 있습니다.

디스테이저(destager). 오브젝트를 스테이징 영역에서 오브젝트의 이주 방침의 First steps로 이동하는 Content Manager 자원 관리자의 기능.

라

라이브러리 서버(library server). 항목에 대한 조회를 저장, 관리 및 처리하는 Content Manager 시스템의 구성요소.

라이브러리 오브젝트(library object). 항목을 참조하십시오.

라이브러리 클라이언트(library client). 라이브러리 시스템에 하위 레벨의 프로그래밍 인터페이스를 제공하는 Content Manager 시스템의 구성요소. 라이브러리 클라이언트에는 소프트웨어 개발자 키트의 일부분인 API가 포함됩니다.

렌더(render). 일반적으로 이미지 지향이 아닌 데이터를 가져와 이미지로 표현하거나 표시하는 작업. Content Manager에서 워드프로세싱 문서는 표시 목적으로 이미지로 렌더될 수 있습니다.

루트 구성요소(root component). 시스템에서 정의하거나 사용자가 정의한 속성으로 구성된 계층 구조 항목 유형의 첫 번째 레벨 또는 유일한 레벨.

링크(link). 두 항목(원본 및 대상)간의 방향 관계. 링크 세트를 사용하여 일 대 다 연관을 모델링할 수 있습니다. 참조와 다릅니다.

마

마운트됨(mounted). Content Manager에서 온라인 상태이며 드라이브 내에 있고 마운트가 활성화된 오브젝트. 인라인과 다릅니다.

마운트(mount). 데이터 미디어를 조작할 위치에 갖다 놓는 것.

매개변수식 검색(parametric search). 오브젝트의 등록 정보를 기본으로 한 오브젝트에 대한 조회.

멀티미디어 파일 시스템(multimedia file system). 비디오 및 오디오의 저장과 전달에 알맞게 최적화된 파일 시스템.

멀티미디어(multimedia). 컴퓨터에서 표시 및 제어를 위해 여러 가지 미디어 요소(텍스트, 그래픽, 오디오, 스틸 이미지, 비디오, 애니메이션)를 결합한 것.

메소드(method). Java 설계 또는 프로그래밍에서 조작에 지정된 작업을 구현하는 소프트웨어. C++의 구성원 함수와 동일한 의미입니다.

목차(TOC: Table Of Contents). 폴더 또는 작업함에 들어 있는 문서 및 폴더의 목록. 검색 결과는 폴더 목차로서 표시됩니다.

문서 경로지정 프로세서(document routing process). Content Manager에서 문서 또는 폴더가 처리되는 동안 거치는 일련의 작업 단계 및 이들 단계를 제어하는 규칙.

문서(document). Content Manager 시스템과 사용자 사이에서 별도의 단위로 저장, 검색 및 교환할 수 있는 항목. 문서 의미 유형이 있는 항목은 문서를 이루는 정보를 포함하도록 예상되나, Content Manager 문서 모델의 구현을 의미하지는 않습니다.

항목 유형이 분류된 문서에서 작성된 항목(Content Manager 문서 모델의 특정 구현)은 문서 부분을 포함해야 합니다. 분류된 항목 유형 문서를 사용하여 문서 또는 폴더 의미 유형을 갖는 항목을 작성할 수 있습니다.

문서 부분은 텍스트, 이미지 및 스프레드시트를 비롯하여 여러 가지 유형의 콘텐츠를 포함할 수 있습니다.

미디어 서버(media server). 비디오 파일의 저장 및 액세스에 사용되는 Content Manager 시스템의 AIX 기본 구성요소.

바

반복자(iterator). 한 번에 오브젝트 콜렉션 전체를 처리하는 데 사용하는 클래스 또는 구조.

볼륨(volume). 시스템의 오브젝트가 저장된 실제 물리적 기억 영역 장치 또는 장치를 나타냅니다.

부분(part). 오브젝트를 참조하십시오.

사

사용 권한세트(privilege set). 시스템 구성요소 및 기능에 대한 작업에 필요한 사용 권한 컬렉션. 시스템 관리자는 사용자(사용자 ID) 및 사용자 그룹에 사용 권한 세트를 할당합니다.

사용 권한(privilege). 특정 방법으로 특정 오브젝트에 액세스하는 권한. 사용 권한에는 시스템에 저장된 오브젝트의 작성, 삭제 및 선택과 같은 권한이 포함됩니다. 사용 권한은 시스템 관리자에 의해 할당됩니다.

사용 기록(history log). 워크플로우의 활동 레코드를 보관하는 파일.

사용자 그룹(user group). 한 명 이상의 정의된 개별 사용자로 구성된 그룹으로서, 단일 그룹 이름으로 식별됩니다.

사용자 맵핑(user mapping). Enterprise Information Portal 사용자 ID 및 암호를 하나 이상의 콘텐츠 서버에 있는 해당 사용자 ID 및 암호에 연관시키는 것. 사용자 맵핑은 Enterprise Information Portal 및 다중 콘텐츠 서버에 대한 단일 로그인을 가능하게 합니다.

사용자 종료 루틴(user exit routine). 미리 정의된 사용자 종료에서 제어를 수신하는 사용자가 작성한 루틴.

사용자 종료(user exit). 사용자 종료 루틴에 제어를 제공할 수 있는 IBM 제공 프로그램의 지점.

사용자(user). Content Manager의 서비스를 요구한 사람. 이 용어는 일반적으로 Content Manager API를 사용하는 응용프로그램 개발자가 아닌 클라이언트 응용프로그램의 사용자를 지칭합니다. Enterprise Information Portal에서는 Enterprise Information Portal 관리 프로그램에서 식별되는 모든 사람을 지칭합니다.

색인 클래스 보기(index class view). 이전 Content Manager에서 색인 클래스 서브세트에 대해 API에서 사용되는 용어.

색인 클래스 서브세트(index class subset). 이전의 Content Manager에서 응용프로그램이 폴더 및 오브젝트를 저장, 검색 및 표시하는 데 사용하는 색인 클래스의 보기.

색인 클래스(index class). 항목 유형을 참조하십시오.

색인(index). 나중에 검색할 수 있도록 특정 항목 또는 오브젝트를 식별하는 속성값을 추가하거나 편집하는 것.

생성자(constructor). 프로그래밍 언어에서 클래스와 동일한 이름을 가지며 해당 클래스에 대한 오브젝트를 작성 및 초기화하는 데 사용되는 메소드.

서버 목록(server inventory). 지정된 콘텐츠 서버에 있는 원래의 엔티티 및 원래의 속성의 포괄적인 목록.

서버 유형 정의(server type definition). Enterprise Information Portal에 대해 특정 유형의 사용자 조정 서버를 고유하게 식별하는 데 필요한 특성 목록으로 시스템 관리자에 의해 식별됩니다.

서버 정의(server definition). Enterprise Information Portal에 대해 서버를 고유하게 식별하는 콘텐츠 서버의 특성.

서브클래스(subclass). 다른 클래스에서 파생된 클래스. 클래스와 서브클래스 사이에 하나 이상의 클래스가 있을 수 있습니다.

속성 그룹(attribute group). 편의상 하나 이상의 속성에 대한 그룹화. 예를 들어, 주소는 동, 구/군/시, 시/도 및 우편번호를 포함합니다.

속성(attribute). 항목의 특정 특성 또는 등록 정보(예: 이름, 주소, 연령 등)를 기술하는 데이터 단위로, 해당 항목을 찾는 데도 사용될 수 있습니다. 속성은 해당 속성별로 저장되는 정보 범위를 나타내는 유형과, 해당 범위 내에 있는 값을 가집니다. 예를 들어, 제목, 실행 시간 또는 인코딩 유형(MPEG1, H.263 등)과 같은 멀티미디어 파일 시스템 내의 파일에 대한 정보가 있습니다. Enterprise Information Portal의 경우, 연합 속성 및 원래의 속성도 참조하십시오.

수퍼 클래스(superclass). 클래스가 파생된 클래스. 클래스와 수퍼 클래스 사이에 하나 이상의 클래스가 있을 수 있습니다.

스테이징 영역(staging area). 자원 관리자의 작업 기억영역. 자원 관리자 캐시라고도 합니다.

스테이징(staging). 일반적으로 시스템 요구 또는 사용자 요청이 있을 때 저장된 오브젝트를 오프라인 또는 우선순위가 낮은 장치에서 다시 온라인 또는 우선순위가 높은 장치로 이동하는 프로세스. 사용자가 영구 기억영역에 저장된 오브젝트를 요청하면, 작업 사본이 스테이징 영역에 쓰여집니다.

스트림 데이터(streamed data). 지정된 전송률로 네트워크 연결을 통해 전송되는 모든 데이터. 스트림은 하나의 데이터 유형 또는 조합된 유형일 수 있습니다. 데이터 비율(초당 비트 수로 표시됨)은 스트림 및 네트워크의 여러 유형에 따라 다양합니다.

시스템 관리 기억영역(SMS: System-Managed Storage). 기억영역 관리에 대한 Content Manager의 접근 방식. 시스템은 오브젝트 위치를 판별한 다음, 자동으로 오브젝트 백업, 이동, 공간 및 보안을 관리합니다.

아

아카이브(archive). 장기간 정보 보관에 사용되는 영구 기억영역. 일반적으로 저장 단위당 가격이 매우 저렴하고 액세스가 느리며, 종종 장비 고장이나 자연 재해로부터 데이터를 보호하기 위해 다른 장소에 보관합니다.

액세스 제어 목록(access control list). 하나 이상의 사용자 ID 또는 사용자 그룹 및 연관된 사용 권한으로 구성된 목록. Content Manager 시스템에서는 액세스 제어 목록을 사용하여 항목 및 오브젝트에 대한 사용자 액세스를 제어합니다. Enterprise Information Portal 시스템에서는 액세스 제어 목록을 사용하여 검색 템플릿에 대한 사용자 액세스를 제어합니다.

액세스 제어(access control). 특정 함수 및 저장된 오브젝트에 권한이 부여된 사용자만이 권한이 부여된 방법으로 액세스할 수 있게 하는 프로세스.

연결 관리자(connection manager). 각 조회를 위해 연결을 새로 시작하지 않고 라이브러리 서버에 대한 연결의 유지보수를 도와 주는 Content Manager 구성요소. 연결 관리자에는 API(Application Programming Interface)가 있습니다.

연합 검색(federated search). 이기종일 수 있는 하나 이상의 콘텐츠 서버에서 동시에 데이터를 검색하는 Enterprise Information Portal에서 발행된 조회.

연합 데이터스토어(federated datastore). Content Manager와 같은 여러 개의 특정 콘텐츠 서버를 가상적으로 표현한 것.

연합 속성(federated attribute). 하나 이상의 콘텐츠 서버에서 원래의 속성에 매핑된 Enterprise Information Portal 메타데이터 카테고리. 예를 들어, 연합 속성인 보험 증권 번호는 Content Manager에서는 보험 증권 번호 속성에 매핑될 수 있고 OS/390용 Content Manager ImagePlus에서는 보험 증권 ID 속성에 매핑될 수 있습니다.

연합 엔티티(federated entity). 연합 속성으로 구성되었으며 선택적으로 하나 이상의 연합 텍스트 색인에 연관된 Enterprise Information Portal 메타데이터 오브젝트.

연합 컬렉션(federated collection). 연합 검색에서 발생한 오브젝트 그룹.

연합 텍스트 색인(federated text index). 하나 이상의 콘텐츠 서버에서 하나 이상의 원래 텍스트 색인에 매핑된 Enterprise Information Portal 메타데이터 오브젝트.

오버레이(overlay). 선, 음영, 텍스트, 상자 또는 로고와 같이 인쇄 중에 페이지에서 변수 데이터에 병합될 수 있는 미리 정의된 컬렉션.

오브젝트 서버 캐시(object server cache). 자원 관리자 캐시를 참조하십시오.

오브젝트 서버(object server). 자원 관리자를 참조하십시오.

오브젝트(object). 사용자가 하나의 단위로 저장, 검색 및 조작할 수 있는 모든 디지털 콘텐츠로서 JPEG 이미지, MP3 오디오, AVI 비디오 및 책의 텍스트 블록을 예로 들 수 있습니다.

와일드 카드 문자(wildcard character). 하나 이상의 문자를 나타내는 데 사용될 수 있는 별표(*) 또는 물음표(?)와 같은 특수 문자. 모든 문자 또는 문자 세트를 와일드 카드 문자로 대체할 수 있습니다.

요소(element). 목록 관리자가 응용프로그램에 할당하는 오브젝트

워크플로우 상태(Workflow State). 전체 워크플로우 상태.

워크플로우 조정자(workflow coordinator). 이전의 Content Manager 워크플로우에서 워크플로우의 작업 항목이 지정된 시간 내에 처리되지 못했을 경우에 공고를 받을 사용자. 사용자는 특정 사용자 그룹에 대해 선택되거나 워크플로우 작성시 선택됩니다.

워크플로우(workflow). 이전 Content Manager에서 문서 또는 폴더가 처리되는 동안 거치는 일련의 작업함. Enterprise Information Portal에서 작업 패킷, 문서 또는 폴더가 처리되는 동안 거치는 일련의 작업 단계 및 이러한 단계를 제어하는 규칙. 예를 들어, 지불 요구 승인은 개별 보험 지불 요구가 승인을 받기 위해 따라야 하는 프로세스에 대해 설명합니다.

원래의 색인 텍스트(native text index). 특정 콘텐츠 서버에서 관리되는 텍스트 항목의 색인. 예를 들어, Content Manager 콘텐츠 서버의 단일 텍스트 검색 색인이 있습니다.

원래의 속성(native attribute). 특정 콘텐츠 서버에서 관리되고 해당 콘텐츠 서버에 고유한 오브젝트의 특성. 예를 들어, 키 필드인 보험 증권 번호는 Content Manager 콘텐츠 서버의 원래의 속성인 반면, 보험 증권 ID 필드는 Content Manager OnDemand 콘텐츠 서버의 원래의 속성일 수 있습니다.

원래의 엔티티(native entity). 특정 콘텐츠 서버에서 관리되고 원래의 속성으로 구성된 오브젝트. 예를 들어, Content Manager 색인 클래스는 Content Manager 키 필드로 구성된 원래의 엔티티입니다.

유틸리티 서버(utility server). 스케줄하기 위해 데이터베이스 유틸리티에 의해 사용되는 Content Manager 구성요소. 자원 관리자 또는 라이브러리 서버를 구성할 때 유틸리티 서버를 구성합니다. 자원 관리자 및 라이브러리 서버마다 하나의 유틸리티 서버가 있습니다.

의미 유형(semantic type). 항목에 대한 사용법 또는 규칙. 기본, 주석, 메모는 Content Manager에서 제공하는 의미 유형입니다. 사용자도 자신의 고유한 의미 유형을 정의할 수 있습니다.

이주 방침(migration policy). 오브젝트를 하나의 기억영역 클래스에서 다음 기억영역 클래스로 이동하도록 사용자가 정의한 스케줄. 여기서는 기억영역 계층 구조의 오브젝트 그룹에 대한 보유 및 클래스 변환 특성에 대해 설명합니다.

이주 프로그램(migrator). 오브젝트 이동이 스케줄되면 이주 방침을 점검하고 오브젝트를 다음 기억영역 클래스로 이동하는 자원 관리자의 기능.

이주(migration). (1) 새 운영 환경으로 이동할 때와 같이 데이터를 변환하지 않고 한 컴퓨터 시스템에서 다른 컴퓨터 시스템으로 데이터 및 원본을 이동하는 프로세스. (2) 이전 버전 또는 릴리스를 대체하기 위해 프로그램의 새 버전 또는 릴리스를 설치하는 작업.

인라인(inline). Content Manager에서 온라인 상태이며 드라이브 내에 있으나 마운트는 활성화되지 않은 오브젝트. 마운트됨과 다릅니다.

일시중단(suspend). 워크플로우에서 오브젝트를 제거하고 이를 활성화하는 데 필요한 일시중단 기준을 정의하는 것. 나중에 오브젝트를 활성화하면 처리를 계속할 수 있습니다.

자

자원 관리자. 오브젝트를 관리하는 Content Manager 시스템의 구성요소. 이러한 오브젝트는 라이브러리 서버에 저장된 항목에 의해 참조됩니다.

자원 관리자 캐시(resource manager cache). 자원 관리자의 작업 기억영역. 스테이징 영역이라고도 합니다.

작업 단계(work step). 개별 작업 항목, 문서 또는 폴더가 통과해야 하는 워크플로우 또는 문서 경로지정 프로세스의 분리된 지점.

작업 목록(worklist). 사용자에게 할당된 작업 항목, 문서 또는 폴더의 컬렉션.

작업 상태(work state). 개별 작업 항목, 문서 또는 폴더 상태.

작업 패킷(work packet). Enterprise Information Portal 버전 7.1의 한 위치에서 다른 위치로 경로지정된 문서의 컬렉션. 사용자는 작업 목록을 통해 작업 패킷에 액세스하여 작업합니다.

작업 항목(work item). 이전의 Content Manager 워크플로우 및 Enterprise Information Portal 고급 워크플로우에서, 워크플로우 내에서 활성화된 모든 작업 활동.

작업함(workbasket). 처리 중이거나 처리 대기 중인 문서 또는 폴더의 컬렉션. 작업함 정의에는 프리젠테이션, 상태 및 해당 콘텐츠의 보안을 관리하는 규칙이 포함됩니다.

장치 관리자(device manager). Content Manager 시스템에서 자원 관리자 및 하나 이상의 실제 장치 사이의 인터페이스.

조치 목록(action list). 사용자가 워크플로우 또는 문서 경로지정 프로세스에서 수행할 수 있는 승인된 조치 목록으로, 시스템 관리자 또는 다른 워크플로우 조정자에 의해 정의됩니다.

조회 문자열(query string). 조회의 등록 정보 및 등록 정보 값을 지정하는 문자열. 응용프로그램에 조회 문자열을 작성하여 조회로 전달할 수 있습니다.

차

참조(reference). 루트 또는 하위 구성요소와 다른 루트 구성요소 간의 단방향성을 가진 일 대 일 연관. 링크와 다릅니다.

추상 클래스(abstract class). 개념을 나타내는 객체 지향 프로그래밍 클래스. 여기에서 파생된 클래스는 개념의 구현을 나타냅니다. 추상 클래스 오브젝트는 구성할 수 없습니다. 즉, 인스턴스로 작성될 수 없습니다.

카

카테고리(category). 항목 유형을 참조하십시오.

캐시(cache). 기본 기억영역보다 작고 빠른 특수한 목적의 버퍼로, 자주 액세스될 수 있는 데이터의 사본을 보유하는 데 사용

됩니다. 캐시를 사용하면 액세스 시간은 단축되나 메모리 요구사항이 증가될 수도 있습니다. 자원 관리자 캐시 및 LAN 캐시도 참조하십시오.

커넥터 클래스(connector class). 특정 콘텐츠 서버에 고유한 API에 대한 표준 액세스를 제공하는 객체 지향 프로그래밍 클래스.

커서(cursor). 응용프로그램에서 일부 순서화된 행 세트 내의 특정 행을 가리키는 데 사용하는 명명된 제어 구조. 커서는 세트에서 행을 검색하는 데 사용됩니다.

컨테이너(container). 오브젝트를 보유하는 사용자 인터페이스의 요소. 폴더 관리자에서 다른 폴더 또는 문서를 포함할 수 있는 오브젝트.

콘텐츠 서버(content server). 멀티미디어 및 비즈니스 데이터와 사용자가 이 데이터에 대해 작업하는 데 필요한 관련 메타데이터를 저장하는 소프트웨어 시스템. 콘텐츠 서버의 예로는 OS/390용 Content Manager ImagePlus 및 Content Manager가 있습니다.

콘텐츠 클래스(content class). MIME 유형을 참조하십시오.

컬렉션(collection). 관리 규칙 세트가 유사한 오브젝트의 그룹.

클라이언트 응용프로그램(client application). 사용자 인터페이스를 사용자 조정하기 위해 Content Manager API를 사용하여 작성된 응용프로그램. Enterprise Information Portal에서 콘텐츠 서버에 액세스하기 위해 객체 지향 또는 인터넷 API를 사용하여 작성된 응용프로그램.

클라이언트/서버(client/server). 통신에서 한 사이트에 있는 프로그램이 다른 사이트에 있는 프로그램으로 요청을 보낸 다음, 응답을 기다리는 분산 데이터 처리에서의 상호작용 모델. 요청 프로그램을 클라이언트라고 하고 응답 프로그램을 서버라고 합니다.

클래스(class). 객체 지향 설계 또는 프로그래밍에서 공통 정의, 공통 등록 정보, 조작 및 작동을 사용하는 오브젝트를 작성하기 위해 인스턴스로 작성될 수 있는 모델 또는 템플릿. 오브젝트는 클래스의 인스턴스입니다.

키 필드(key field). 속성을 참조하십시오.

과

파일 시스템(file system). AIX에서 기억영역에 대해 하드 드라이브를 파티션하는 메소드.

패키지(package). 액세스 보호 및 이름 공간 관리를 제공하는 관련 클래스 및 인터페이스의 컬렉션.

퍼저(purger). 시스템에서 오브젝트를 제거하는 자원 관리자의 기능.

폴더 관리자(folder manager). 데이터를 온라인 문서 및 폴더로 관리하기 위한 Content Manager 모델. 응용프로그램과 Content Manager 콘텐츠 서버 사이의 기본 인터페이스로 폴더 관리자 API를 사용할 수 있습니다.

폴더(folder). 분류에 관계없이 폴더 의미 유형이 있는 항목 유형의 항목. 폴더 의미 유형이 있는 항목은 Content Manager에서 제공하는 특정 폴더 기능을 포함합니다. 또한 문서 또는 자원 항목과 같은 항목 유형 분류에서 사용 가능한 추가 기능 및 모든 비자원 항목 성능을 포함합니다. 폴더는 문서 및 서브폴더를 포함하는 많은 항목 유형을 포함할 수 있습니다. 폴더는 속성에 의해 색인화될 수 있습니다.

하

하위 구성요소(child component). 계층 구조 항목 유형의 두 번째 이하 레벨로서, 이는 선택적입니다. 각 하위 구성요소는 직접 상위 레벨에 연관됩니다.

항목 유형 분류(item type classification). 해당 항목 유형의 항목을 보다 자세히 식별하는 항목 유형 내의 구분. 동일한 항목 유형을 가진 모든 항목은 동일한 항목 유형 분류를 갖습니다.

Content Manager는 폴더, 문서, 오브젝트, 비디오, 이미지 및 텍스트와 같은 항목 유형 분류를 제공하며, 사용자가 고유한 항목 유형 분류를 정의할 수도 있습니다.

항목 유형(item type). 항목과 같이 정의한 후 나중에 찾을 수 있는 템플릿로서, 루트 구성요소, 0개 이상의 하위 구성요소 및 분류로 구성됩니다.

항목(item). Content Manager에서 항목 유형의 인스턴스에 대한 일반 용어. 예를 들어, 항목은 폴더, 문서, 비디오 또는 이미지일 수 있습니다. Enterprise Information Portal에서 관리하는 최소 정보 단위에 대한 일반 용어. 각 항목에는 ID가 있습니다. 예를 들어, 항목은 폴더 또는 문서일 수 있습니다.

해제(release). 항목에서 일시중단 기준을 제거하는 것. 일시중단 항목은 기준에 해당되거나 적절한 권한을 가진 사용자가 기준을 대체하고 수동으로 이를 해제할 경우에 해제됩니다.

핸들(handle). 오브젝트를 나타내고 오브젝트를 검색하는 데 사용되는 문자열.

후원자(**patron**). Content Manager API에서 사용자에게 대해 사용되는 용어.

A

ADSM. *Tivoli® Storage Manager*를 참조하십시오.

API. *API(Application Programming Interface)*를 참조하십시오.

API(Application Programming Interface). 응용프로그램이 다른 응용프로그램과 통신할 수 있도록 하는 소프트웨어 인터페이스. API는 기본 라이선스 프로그램에서 제공하는 특정 기능 및 서비스를 확보하기 위해 응용프로그램에서 코드화될 수 있는 프로그래밍 언어 구성 또는 명령문 세트입니다.

AVI. *AVI(Audio/Video Interleaved)*를 참조하십시오.

AVI(오디오/비디오 인터리브: Audio/Video Interleaved). 파일에 오디오 및 비디오 데이터를 삽입할 수 있도록 하는 RIFF(*Resource Interchange File Format*) 파일 스펙. 파일 장치에서의 순차 액세스를 유지보수하는 동안 재생 또는 레코딩을 할 때 대체 청크에서 개별 트랙에 액세스할 수 있습니다.

B

BLOB. *BLOB(Binary Large Object)*를 참조하십시오.

BLOB(Binary Large Object). 크기의 범위가 0에서 2기가바이트인 일련의 바이트. 이 문자열에는 연관된 코드 페이지 및 문자 세트가 없습니다. 이미지, 오디오 및 비디오 오브젝트는 BLOB로 저장됩니다.

C

CGI. *CGI(Common Gateway Interface)*를 참조하십시오.

CGI 스크립트(CGI script). 웹 서버에서 실행되며, *CGI(Common Gateway Interface)*를 사용하여 일반적으로 웹 서버에서 수행되지 않는 작업(예: 데이터베이스 액세스 및 양식 처리)을 수행하는 컴퓨터 프로그램. CGI 스크립트는 Perl과 같은 스크립트 언어로 작성된 CGI 프로그램입니다.

CGI(Common Gateway Interface). 웹 서버와, 웹 서버 외부 프로그램 간의 정보 교환을 위한 표준. 외부 프로그램은 웹 서버가 실행되는 운영 체제에서 지원하는 프로그램 언어로 작성될 수 있습니다. *CGI 스크립트*를 참조하십시오.

CIF. *CIF(Common Interchange File)*를 참조하십시오.

CIF(Common Interchange File). 하나의 IPIA(*ImagePlus Interchange Architecture*) 데이터 스트림이 들어 있는 파일.

CIU. *CIU(Common Interchange Unit)*를 참조하십시오.

CIU(Common Interchange Unit). CIF(*Common Interchange File*)의 독립적인 전송 단위. 이는 수신 데이터베이스와의 관계를 식별하는 CIF의 일부분입니다. CIF는 여러 개의 CIU를 포함할 수 있습니다.

D

DCA. *DCA(Document Content Architecture)*를 참조하십시오.

DCA(Document Content Architecture). 사무실 시스템 네트워크에서 교환되는 문서에 대한 정보의 무결성을 보장하는 아키텍처. DCA는 문서의 양식 및 의미를 지정하는 규칙을 제공합니다. 이는 개정 가능한 양식 텍스트(변경 가능)와 최종 양식 텍스트(변경 불가능)를 정의합니다.

DDO. *DDO(Dynamic Data Object)*를 참조하십시오.

DDO(Dynamic Data Object). 응용프로그램에서 해당 오브젝트를 기억영역 내부 또는 외부로 이동하는 데 사용되는 저장 오브젝트의 일반적인 표현 방법.

DTD. *DTD(Document Type Definition)*를 참조하십시오.

DTD(Document Type Definition). XML 문서의 특정 클래스에 대한 구조를 지정하는 규칙. DTD는 요소, 속성 또는 표기법을 사용하여 구조를 정의하며 문서의 특정 클래스에서 각 요소, 속성 또는 표기법을 사용할 수 있는 방법에 대한 제한조건을 설정합니다. DTD는 특정 마크업 언어의 구조에 대해 완전히 설명할 수 있다는 점에서 데이터베이스 스키마와 유사합니다.

G

GB. *기가바이트*를 참조하십시오.

H

HTML. *HTML(Hypertext Markup Language)*을 참조하십시오.

HTML(Hypertext Markup Language). SGML 표준을 따르고 기본적으로 하이퍼텍스트 링크를 포함하는 온라인 텍스트 및 그래픽 정보를 지원하도록 설계된 마크업 언어.

I

Information Mining. 텍스트에서 핵심 정보를 추출하는 자동화된 프로세스로서(요약) 문서 컬렉션에서 핵심 주제를 찾고(카테고리) 강력하고 유연한 조화를 사용하여 관련 문서를 검색합니다.

IOCA. *IOCA(Image Object Content Architecture)*를 참조하십시오.

IOCA(Image Object Content Architecture). 이미지를 상호 교환하고 표시하는 데 사용되는 구조의 컬렉션.

J

JavaBeans. 『Bean』이라고 하는 재사용 가능한 Java 구성요소를 빌드하기 위한 플랫폼 독립 소프트웨어 구성요소 기술. 빌드한 후, 이 bean을 다른 소프트웨어 엔지니어가 사용하거나 Java 응용프로그램에서 사용할 수 있습니다. 소프트웨어 엔지니어는 JavaBeans를 사용하여 그래픽 끌어서 놓기 개발 환경에서 bean을 조작 및 어셈블할 수 있습니다

JPEG. *JPEG(Joint Photographic Experts Group)*를 참조하십시오.

JPEG(Joint Photographic Experts Group). (1) 디지털화된 연속 톤 이미지 압축의 표준을 설정하기 위해 작업하는 그룹. (2) 이 그룹에서 개발한 스틸 그림에 대한 표준.

L

LAN. 근거리 통신망을 참조하십시오.

LAN 캐시(LAN cache). 원격 자원 관리자에 저장된 오브젝트의 사본을 포함하는 로컬 자원 관리자의 임시 기억영역.

M

MediaArchiver. 오디오 및 비디오 스트림 데이터를 저장하는 데 사용되는 실제 장치. VideoCharger는 MediaArchiver의 한 유형입니다.

MGDS. *MGDS(Machine-Generated Data Structure)*를 참조하십시오.

MGFS(Machine-Generated Data Structure). (1) 여러 OS/390용 Content Manager ImagePlus 프로그램 간에 문자 데

이터를 전달하기 위한 IBM 구조화 데이터 형식 프로토콜. (2) 이 미지에서 추출하여 GDS(General Data Stream) 형식으로 기록한 데이터.

MIME 유형(MIME type). 인터넷에서 전송되고 있는 오브젝트의 유형을 식별하기 위한 인터넷 표준. MIME 유형에는 여러 가지 변형된 오디오, 이미지 및 비디오가 포함되어 있습니다. 각 오브젝트에는 MIME 유형이 있습니다.

MIME(Multipurpose Internet Mail Extensions) . *MIME* 유형을 참조하십시오.

MO:DCA. *Mixed Object Document Content Architecture*

MO:DCA(Mixed Object Document Content Architecture). 교환 환경 내의 응용프로그램 사이 및 환경 사이에서 오브젝트 데이터의 교환이 허용되도록 개발된 IBM 아키텍처.

MO:DCA-P. *Mixed Object Document Content Architecture--Presentation*

MO:DCA-P(Mixed Object Document Content Architecture-Presentation). 화면에 표시하거나 인쇄를 위해 OS/390용 Content Manager ImagePlus 워크스테이션으로 전송되는 문서를 포함할 봉투로 사용되는 MO:DCA의 서브세트 아키텍처.

O

OLE. *OLE(Object Linking and Embedding)*를 참조하십시오.

OLE(Object Linking and Embedding). 다른 응용프로그램 내에서 활성화될 수 있도록 응용프로그램 링크 및 포함 모두에 필요한 Microsoft® 스펙.

P

PID. *PID(지속 식별자)*를 참조하십시오.

PID(지속 식별자). 저장된 위치에 관계없이 오브젝트를 고유하게 식별하는 ID. PID는 항목 ID와 위치로 구성됩니다.

Q

QBIC. *QBIC(Query By Image Content)*를 참조하십시오.

QBIC(Query By Image Content). 일반 텍스트가 아닌 기능이라고 하는 비주얼 콘텐츠를 기본으로 검색할 수 있게 하는

조회 기법. QBIC를 사용하여 색 및 텍스트와 같은 비주얼 특성을 기본으로 오브젝트를 검색할 수 있습니다.

R

README 파일(README file). 파일과 연관된 프로그램을 설치하거나 실행하기 전에 보아야 하는 파일. README 파일에는 일반적으로 최신 제품 정보, 설치 정보 또는 제품 사용에 대한 팁이 들어 있습니다.

RIFF. *RIFF(Resource Interchange File Format)*를 참조하십시오.

RIFF(Resource Interchange File Format) . 다른 유형의 컴퓨터 장비에서 재생할 소리 또는 그래픽을 저장하는 데 사용됩니다.

RMI 서버(RMI server). Java *RMI(Remote Method Invocation)* 분산 오브젝트 모델을 구현하는 서버

RMI(Remote Method Invocation). 분산 프로그래밍을 가능하게 하는 API 세트. 한 JVM(Java Virtual Machine)에 있는 오브젝트는 다른 JVM의 오브젝트에 있는 메소드를 호출할 수 있습니다

S

SMS. 시스템 관리 기억영역을 참조하십시오.

T

thin 클라이언트(thin client). 설치된 소프트웨어는 거의 없거나 접속된 네트워크 서버에 의해 관리 및 전달되는 소프트웨어에 대한 액세스 권한은 갖고 있는 클라이언트. thin 클라이언트는 워크스테이션과 같은 전기능 클라이언트에 대한 대안입니다.

Tivoli StorageManager(TSM). 이기종 환경에서 기억영역 관리 및 데이터 액세스 서비스를 제공하는 클라이언트/서버 제품. TSM은 여러 가지 통신 메소드를 지원하고 파일의 백업 및 기억영역을 관리하기 위한 관리 기능을 제공하며 백업 조작을 스케줄하기 위한 기능을 제공합니다.

TOC. 목차를 참조하십시오.

TSM. *Tivoli Storage Manager*를 참조하십시오.

TSM 볼륨(TSM volume). *Tivoli Storage Manager*에서 관리하는 논리 영역

U

URL(Uniform Resource Locator). 컴퓨터 또는 인터넷과 같은 네트워크의 정보 자원을 나타내는 문자 순서. 이 문자 순서에는 정보 자원에 액세스하는 데 사용되는 프로토콜의 약어 이름과, 정보 자원을 찾기 위해 프로토콜에서 사용하는 정보가 포함됩니다. 예를 들어, 인터넷 구문에서 http, ftp, gopher, telnet 및 news는 여러 가지 정보 자원에 액세스하는 데 사용되는 프로토콜의 약어 이름 중 일부입니다.

W

Windows용 클라이언트 응용프로그램(Client Application for Windows). Content Manager에서 제공되며, Content Manager API를 사용하여 작성된 완전한 오브젝트 관리 시스템. 문서 및 폴더 작성, 기억영역, 프리젠테이션, 처리 및 액세스 제어를 지원합니다. 사용자 종료 루틴을 사용하여 사용자 조정하고 API를 사용하여 부분적으로 호출할 수 있습니다.

X

XDO. *XDO(Extended Data Object)*를 참조하십시오.

XDO(Extended Data Object). 응용프로그램에서 오브젝트를 기억영역 내부 또는 외부로 이동하는 데 사용되는 저장된 복합 멀티미디어 오브젝트의 일반적인 표현 방법. XDO는 주로 DDO에 포함되어 있습니다.

XML. *XML(Extensible Markup Language)*을 참조하십시오.

XML(Extensible Markup Language). 마크업 언어를 정의하는 표준 다중 언어로 SGML에서 파생되었으며 SGML의 서브세트입니다. XML은 좀더 복잡하고 많이 사용되지 않는 SGML의 부분을 없애며, 좀더 쉽게 문서 유형 및 작성자를 처리하고 구조화 정보를 관리하며 다양한 컴퓨터 시스템에서 구조화 정보를 전송 및 공유하는 응용프로그램을 작성할 수 있게 합니다. XML을 사용하는 데는 SGML에서는 필수였던 강력한 응용프로그램 및 처리는 필요하지 않습니다. XML은 W3C(World Wide Web Consortium)의 후원 아래 개발 중입니다.

색인

[가]

가비지 콜렉터, Java 29
강조표시, 텍스트 검색 129
 각 결과에 대해 정보 가져오기 129
 특정 결과에 대해 정보 가져오기 134
갱신, BLOB 435
검색
 C++
 부분 291
 Java 289
 부분 290
 폴더 292
검색 실행 38
검색 엔진, 등록 427
검색 중
 조회 언어 이해 211
 API 모듈 160
 Bean 463, 498, 500
 DKResults 클래스 426
검색 템플릿, OnDemand
 다음과 같이 폴더 사용 367
 보기 프로그램 366
검색 평가 38
검색, BLOB 435
결과 세트 커서
 함수 432
 Java 138
 설정 및 얻기 138
 열기 및 닫기 138
 콜렉션 작성 141
결과 세트 커서 화면이동 432
결합된 조회
 정의 38
 C++
 순위 343
 프로그래밍 팁 343
 Java 217, 339
 매개변수식 텍스트 340
 범위 사용 341
경로지정 464
 문서 경로지정 프로세스 나열 270
 사용 권한 부여 272
 상수 273

경로지정 (계속)
 새 일반 작업 노드 정의 255
 새 콜렉션 지점 정의 258
 새 프로세스 및 연관된 경로 정의 262
 서비스 오브젝트 작성 254
 설정 253
 소개 252
 시작 265
 액세스 제어 목록(ACL) 273
 예제 271
 임시 270
 작업 노드 252
 작업 노드 나열 256
 작업 목록 254
 작업 목록 나열 261
 작업 목록 정의 260
 작업 목록에서 패키지 PID 나열 268
 작업 패키지 254
 작업 패키지 정보 검색 269
 재개 목록 254
 콜렉션 지점 252
 프로세스 계속 266
 프로세스 일시중단 267
 프로세스 재개 267
 프로세스 종료 266
 Bean 462, 464
공통 사용 권한, dkDatastoreExt 439
공통 EIP 클래스 424
관계형 데이터베이스
 구성 문자열 416
 소개 415
 속성 나열 417
 엔티티 나열 417
 연결 대상 415
 연결 문자열 415
 연합에서 맵핑 20
 조회 420
관리
 사용 권한 153
 오브젝트
 검색 242
관리 데이터베이스
 설명 4
 소개 4

관리 데이터베이스 (계속)
 스키마 맵핑 14
관리 클라이언트
 설명 4
 소개 4
 추가 워크스테이션에 설치 4
구성 문자열, 관계형 데이터베이스 61
구성원
 제거 285
 추가 285
권한,
 참조: 사용 권한
가능
 가중치, 이미지 검색 323
 값, 이미지 검색 323
 이름, 이미지 검색 323
 이미지 검색 319
 최대 결과, 이미지 검색 323
기억영역 콜렉션
 다음에 XDO 추가 89
 XDO 변경 91
기준, 검색, Bean 499
길이, BLOB 435

[나]

널(NULL), 조회 236

[다]

다중 문자(MC) 300
단계, 조회 237
단일 요구 문자(SC) 300
단기, 결과 세트 커서 433
대형 오브젝트 (LOB) 160
데이터 모델, Content Manager 버전 8
 조회 언어 적용 212
데이터 모델, 조회
 XML 표현 221
데이터 정의 클래스 277
데이터 항목, DDO 53

데이터스토어,

참조 : 콘텐츠 서버

도구

Content Manager 버전 8 158

동적 구성, Bean 461

동적 데이터 오브젝트(DDO)

다음에 포함하는 모든 폴더 획득 196

등록 정보 278

등록 정보 추가 51

멀티미디어 콘텐츠 표현 16

소개 14

속성 등록 정보 검색 56

속성 비교 16

속성에 액세스 54

연합 19

이미지 검색 결과 326

이미지 검색 지속 ID 331

이미지 검색에서 표현 331

지속 식별자 16

참조 150

콘텐츠 서버와의 관계 16

컬렉션 정렬 114

폴더 콘텐츠에 액세스 103

CM 8의 속성 그룹 168

COBRA 스펙 14

C++

삭제 59

dkDDO 424

ES에서 식별 399

FileNET에 표시 410

Java 48

데이터 항목값 53

등록 정보 55

속성, DKPARTS 98, 101

작성 49

정보, Digital Library 160, 278

정보, 텍스트 검색 엔진 297

추가 53

표시 57

PID 52

retrieveObject 184

setData로 채우기 173

XML 가져오기 96

XML 내보내기 97

[라]

라이브러리

C++ 나열 34

Java 31, 33

Microsoft Visual Studio .NET 36

라이브러리 서버, Content Manager 버전

8 145

나열 162

버전화 151

로그 기록,

참조 : 추적

로케일

servlet 503

롤백 427

루트 구성요소

데이터 모델의 표현 212

링크 149

버전화 151

참조 150

CM 8에 작성 172

Content Manager 버전 8 148

리터럴

OS/390용 ImagePlus 376

리터럴, 조회 228, 238

링크

데이터 모델의 표현 213

링크된 항목 검색 199

목표 197

설명 항목 197

속성 유형 160

원본 197

유형 이름 199

유형 이름 상수 199

인바운드 및 아웃바운드 198

조회를 통한 탐색 224

항목 간 정의 197

Content Manager 버전 8 149

[마]

매개변수식 검색

여러 기준 구성 118

여러 기준 조회 118

연산자 214

연합 23

이해 214

정의 38

매개변수식 검색 (계속)

조회 문자열 구성 117

조회 실행 119

추적 39

콘텐츠 서버에서 조회 120

콘텐츠 서버에서 조회 평가 122

AS/400용 CM에서 조회 384

Panagon Image Services 412

멀티미디어 콘텐츠 16

멀티스트리밍 29

멀티플랫폼용 IBM Enterprise Information

Portal

구성요소 4

관리 데이터베이스 4

관리 클라이언트 4

샘플 클라이언트 응용프로그램 5

커넥터 5

메모

상수 174

의미 유형 174

메모, Bean 로그 501

메시지

등록 정보 파일 440

DKException 정보 157

메타데이터

information mining 5

명암 321

문서 174

갱신 428

검색 428

경로지정 프로세스 시작 265

관리 데이터 모델의 버전화 부분 248

다음에 대해 텍스트 검색 215

데이터 모델의 표시 214

문서 항목 유형 작성 241

보기,

참조 : 문서 보기 프로그램 툴킷

삭제 428

의미 유형 174

이동 428

이전 CM 관리의 차이점 286

이전 CM 워크플로우 343

제어기 servlet에서의 캐싱 506

주석 처리 485

체크아웃 및 체크인 439

추가 428

프로세스를 통한 경로지정 252

Bean 465, 498, 502

문서 (계속)

Bean에 표시 470, 477
Bean으로 클러스터링 465
CM 8 관리 240
CM 8에 관리 데이터 모델 작성 241
CM 8에 작성 243
CM 8에서 갱신 245
CM 8에서 검색 247
CM 8에서 삭제
SDocModelItemICM 247
Content Manager 버전 8 147, 150
Domino.Doc 387

문서 경로지정,

참조: 경로지정

문서 모델,

참조: 문서 부분

문서 보기 프로그램 툴킷

기능 485
독립형 보기 프로그램 490
소개 485
아키텍처 486
애플릿 또는 servlet 491
엔진
AFP2Web 문서 487
INSO 문서 487
Java 문서 487
MS-Tech 문서 486
예제 응용프로그램 489
이중 모드 및 애플릿 또는 servlet 492
일반 문서 보기 프로그램 사용자 조정 487
일반 보기 프로그램 작성 487
주석 서비스 493
팝업 메뉴 488
Java 응용프로그램 490
thin 클라이언트 491

문서 부분

상수 163
항목 유형 163
Content Manager 버전 8 147

문서 서비스 스트리밍 486

문제점 해결

오류 메시지 등록 정보 파일 440
추적 38

미디어 오브젝트

검색 84
삭제 80
이전 CM에서 추가 76

미디어 오브젝트 (계속)

코드 샘플 이름 90

[바]

바인더, Domino.Doc 387
바인딩, DataJoiner 7
반복자
연합 114
DKResults에 위치 지정 115
방침 예제 160
방향 321
방, Domino.Doc 387
버전 8.2 API의 새로운 기능 6
버전 8.2의 새로운 기능 7
버전 8.2, API의 새로운 기능 6
버전 8.2, 새로운 기능 7
버전화

문서 관리 데이터 모델의 부분 248

버전 제어 방침 187
속성 184
응용프로그램 제어 151
정책 151
지속 식별자 (PID) 151
항목 184
항목 유형 188
항목에 대해 설정 186
Bean 470, 479
Content Manager 버전 8 151

버전, 조회 227

버퍼, XDO 추가 65

범위

태그 라이브러리 498
트랜잭션 249

보기 프로그램, Java 문서,

참조: 문서 보기 프로그램 툴킷

부분

버전화 248
이전 CM 279
이전 CM의 갱신 285
Extended Search 400

비교 237

비자원 항목

Content Manager 버전 8 147

[사]

사용 권한

경로지정 부여 272
데이터 액세스 153
미리 구성된 ACL 156
미리 정의된 Content Manager 버전 8 세트 153

세트 153
세트 유형 154
세트 작성 202
세트의 등록 정보 표시 204

시스템 정의 153
액세스 제어 목록 (ACL) 155

코드 154

Bean 463, 501
CM 8에 작성 200
CM 8의 사용자 및 사용자 그룹 155
Content Manager 버전 8 152, 153

사용 내용

상수 174
의미 유형 174

사용자

CM 8에 액세스 제어 155

사용자 사용 권한

Content Manager 버전 8 152, 153
사용자 정의 속성 213
텍스트를 검색 가능하게 함 216

사용자 조정 커넥터

개발 423
시스템 관리 424
FeServerDefBase 클래스 확장 441

사용자 종료

시스템 관리 27
워크플로우 455

사용자 ID

연합에서 맵핑 21

삭제 435

삭제, BLOB 435

산술 조회

구문규칙 237
연산 224

삽입, BLOB 435

상대적 위치 지정, 결과 세트 커서 140

상수 41

경로지정 273

색 검색

평균 321

색상 검색

- 위치 321, 325
- 텍스처 321, 325
- 평균 324, 325
- 히스토그램 321, 324, 325

색인

- 다음에 대해 텍스트 검색 124

색인화

- 이미지 검색 336

샘플 클라이언트 319

- 응용프로그램 5

샘플 파일

- Content Manager 버전 8 158

샘플 파일, 위치

- Information Mining 515

서브엔티티

- Domino.Doc에 나열 388

서비스 API

- 검색 중 567
- 라이브러리 관리 558
- 레코드 작성 565
- 메타데이터 저장 565
- 문서 언어 판별 563
- 문서 정보 추출 564
- 문서 필터링 563
- 서버 작업 실행 568
- 연결 557
- 요약 생성 564
- 용어 관리 558
- 카탈로그 관리 558
- 카테고리 할당 565
- 클러스터링 564
- 텍스트 문서 작성 563
- Information Mining 557
- Information Mining 도구 사용 562

설정

- 샘플 Java 애플릿 및 servlet 사용 497
 - 로컬 액세스 498
 - 원격 액세스 498
 - 클라이언트의 Java 응용프로그램 497
- servlet 검색 497
- C++
 - NT에 빌드 36
- C++ 환경 33
- Java
 - 클라이언트 연결 및 연결 해제 43
 - 클라이언트/서버 30
 - 프로그래밍 팁 31

설정 (계속)

Java (계속)

- NT 31, 32, 35

Java 환경 31

세션

- servlet 503

세션 리스너, Bean 466

속성

- 관계형 데이터베이스에 나열 417
- 그룹 147
- 다음에서 등록 정보 검색 56
- 문서 항목 유형 검색 242
- 버전화 184
- 사용자 정의 213
 - 텍스트를 검색 가능하게 함 216

여러 값 지정 147

이미지 검색 331

정의 147, 276

정의에 대해 특정 DK 클래스 431

제어기 servlet에서 관리 508

조회 224

참조 213

항목 유형에 대해 나열 170

항목에 대해 수정 176

AS/400용 CM에 나열 378

Bean 463, 500, 501

Bean에 표시 470, 478

CM 8에 설정 및 검색 175

CM 8에 작성 166

CM 8의 그룹 갱신 169

CM8의 그룹 관리 168

Content Manager 버전 8 147

DDO 및 XDO 비교 16

DDO를 통해 표현 161

DDO에서 액세스 54

DKAny에 대해 삭제 113

Extended Search 394

FileNET에 나열 410

IP OS/390에 나열 371

OnDemand에 대해 나열 356

순서 이탈, 조회 예제 230

스코어 기본, 텍스트 검색

텍스트 검색

- contains-text 217

스키마 맵핑

관계형 데이터베이스 415

사용자 조정 커넥터 424

소개 14

스키마 맵핑 (계속)

연합에서 연관 21

dkDatastore 메소드 428

dkSchemaMapping 클래스 426

스키마, Bean 498

스택 추적 157

시간종료된 세션, servlet 503

시나리오, Content Manager 버전 8에 대한 보

험 샘플 159

시스템 관리 클라이언트

사용자 조정 27

사용자 종료 27

작업 목록 작성 450

식별자(ID)

- 이미지 검색 320, 322

[아]

아웃바운드 링크 195, 198

암호

변경 427

연합에서 맵핑 21

Bean에서 변경 480

Content Manager 버전 8에 작성 162

엑세스 제어

도표 152

목록 (ACL) 155

목록에 대한 규칙 (ACLs) 155

목록(ACL) 152

목록(ACL) 검색 및 표시 207

목록(ACL) 정의 205

문서 경로지정에 대한 목록으로 작업 273

작업 200

항목에 목록 할당 209

Content Manager 버전 8 152

NoAccessACL 156

PublicReadACL 156

SuperUserACL 156

양식 오버레이 오브젝트 439

엔티티

관계형 데이터베이스에 나열 417

다음에 연합 폴더 저장 26

사용 권한 153

스키마 맵핑 426

식별 16

연합에서 맵핑 20

오브젝트 이동 439

원래 367, 426

엔티티 (계속)

원래의 OnDemand 폴더 367
 정의에 대해 DK 클래스 이름 430
 제어된, Content Manager 버전 8 152
 클래스 계층 구조 275
 AS/400용 CM에 나열 378
 Bean 462, 500
 CM 8에 나열 165
 Domino.Doc에 나열 388
 Domino.Doc에서 검색 390
 Extended Search 394
 FileNET에 나열 410
 FileNET에서 조회 413
 IP OS/390에 나열 371
 OS/390용 ImagePlus 377

연결 풀링

servlet 503
 servlet 값 504

연결 Bean 499

연산자

매개변수식 검색 214
 Domino.Doc 392
 FileNET 412
 GQL(Generalized Query Language) 408
 OS/390용 ImagePlus 조회 376

연합 26

검색 중
 관계형 설명 23
 설명 20
 소개 3, 19
 표현식 클래스 426
 프로세스 22
 Extended Search 408

맵핑 20

문서 모델 19
 반복자 114
 사용자 ID 맵핑 21
 서버 정의 기본 클래스 441
 속성 20
 스키마 맵핑 21
 시스템 관리 기능 27
 엔티티 20
 이미지 검색 23
 조회

결과 22
 구문 23
 설명 처리 중 23
 예제 24

연합 (계속)

조회 (계속)
 작성 22
 콘텐츠 서버 등록 21
 콜렉션 22, 114
 폴더 26
 항목 19
 Bean 500
 Bean에 대한 계층 462
 연합 문서 모델 19
 열기, BLOB 435
 열기, CLOB 436
 열기, 결과 세트 커서 433
 예외, 처리 39
 오류 메시지
 등록 정보 파일 440
 오버레이 오브젝트 439
 오브젝트
 속성값 저장 175
 콘텐츠에 대한 텍스트 검색 215
 Bean 501
 CM 8에서 갱신 177
 Content Manager 버전 8 149
 Domino.Doc 관리 387
 오브젝트 관리
 Java 280
 갱신 176, 284
 삭제 185, 289
 작성 171, 280
 오브젝트 관리 그룹(OMG) 14
 와일드 카드
 Domino.Doc 392
 와일드카드
 조회 231
 텍스트 검색 216
 워크플로우
 나열 447
 다음에서 연결 해제 중 444
 모델 343
 모든 작업 목록 나열 449
 모든 템플릿 나열 454
 삭제 446
 소개 6, 443
 시작 444
 연결 대상 443
 연합 폴더에서 결과 전송 26
 이전 CM에 나열 346
 이전 CM에 작성 344

워크플로우 (계속)

이전 CM에 항목 ID 나열 350
 이전 CM에서 실행 351
 이전 CM에서 정의 343
 이전 Content Manager 서비스 343
 일시중단 448
 작업 목록에 액세스 450
 작업 항목에 액세스 451
 종료 446
 항목 이동 453
 Bean 460
 Bean 지원 462
 Java 조치 작성 455
 워크플로우 일시중단 448
 워크플로우 재개 448
 웹 크롤러
 소개 6
 Bean 465
 위치 색
 정의 321
 위치 색상
 올바른 값 325
 조회 예제 325
 응용프로그램 빌드
 계획 156
 비주얼 Bean 481
 비주얼이 아닌 Bean 468
 주식 서비스 사용 495
 Content Manager 버전 8에 작성 159
 의미 유형
 미리 정의된 유형 174
 사용자 조정 174
 정의 174
 이미지 검색 319
 검색 기준 323
 기능 319, 323
 기능 나열 329
 다음에서 연결 해제 중 326
 데이터베이스 319, 320
 데이터베이스 나열 329
 도표 320
 색인화할 데이터 로딩 336
 서버 나열 327
 소개 6
 속성 331
 식별자(ID) 322
 연결 대상 326
 연합 23

이미지 검색 (계속)

- 위치 색 321
- 위치 색상 325
- 응용프로그램 321
- 정의 38
- 조회 332
- 조회 구문규칙 323
- 조회 작성 323
- 최대 결과 323
- 카탈로그 319, 320
- 카탈로그 나열 329
- 컨텐츠 서버에서 조회 실행 333
- 컨텐츠 서버에서 조회 평가 335
- 텍스처 321, 325
- 평균 색 321, 324
- 평균 색상 325
- 히스토그램 색 321, 324
- 히스토그램 색상 325
- closeCatalog 326
- DDO를 사용하여 정보 표시 331
- listDatabases 326
- openCatalog 326
- XDO 색인화 336

이미지 대기열 처리 336

이벤트

- Bean 466

인바운드 링크 198

일괄처리 항목 시스템, FileNET 410

일반 문서 보기 프로그램,

- 참조: 문서 보기 프로그램 툴킷

일치 강조표시,

- 참조: 강조표시, 텍스트 검색

임시 경로지정 270

[자]

자원 관리자, Content Manager 버전 8 145

- 나열 162
- 버전화 151
- 오브젝트에 대한 작업 239
- 작업 238

자원 컨텐츠,

- 참조: XDO

자원 항목

- 다음에 대해 항목 유형 정의 178
- 상수 163
- 항목 유형 163

Content Manager 버전 8 147

작성 174

작업 노트

- 나열 256

작업 메모

- 경로지정 252
- 정의 255

작업 목록 254

- 나열 261, 449
- 액세스 450
- 작업 항목에 액세스 451
- 정의 260
- 패키지 PID 나열 268
- 항목 이동 453
- Bean 460

작업 패키지 254, 269

- 작업 목록에서 PID 문자열 나열 268

작업 패킷

- 소개 443

작업 항목

- 액세스 451

작업함

- 규칙 343
- 이전 CM에 나열 349
- 이전 CM에 작성 347
- 이전 CM에서 식별 351
- 이전 CM에서 정의 343

재개 목록 254

정보 검색 269

제거, BLOB

- BLOB (Binary Large Object)
- 부분 제거 435

제거, CLOB 437

제거, 결과 세트 커서 433

제어기 서버

- 기본값 504

제어기 servlet

- 규칙 504
- 등록 정보 파일 504, 506
- 로케일 503
- 매개변수 504, 507
- 변환 매개변수 506
- 사용 503
- 서비스 런타임 506
- 세션 관리 503
- 연결 풀링 503
- 연결 풀링 값 504
- 오류 메시지를 표시하는 페이지 505

제어기 servlet (계속)

- 요청 매개변수
- 검색 507
- 관련 문서 509
- 관련 연결 507
- 응답 507
- 일반 507
- 조치 508
- 컨텐츠 관리 508
- 폴더 509
- 항목 508

이름 분리 문자 매개변수 506

재생 504

제거 503

조치 503, 504

참조 504

추적 505

툴킷 기능 매트릭스 509

확장 504

JSP 세트 503

제어된 엔티티 152

조건식, FileNET 413

조치

- 목록 액세스 455
- 작성 455

조회

- 결과 나열 228
- 결합 표현식 426
- 경로지정 예제 271
- 관계형 데이터베이스 420
- 구문규칙 223
- 구조화된 문서 검색 308
- 데이터 모델 221
- 버전 227
- 복합 표현식 426
- 산술 연산 224
- 순서 이탈 230
- 실행 116
- 언어 230
- 언어 문법 235
- 언어 이해 211
- 연합
- 처리 22
- 연합 컬렉션 115
- 예제 220, 223
- 와일드 카드 예제 227
- 이미지 검색 323, 332
- 이미지 검색 조회 구문규칙 323

조회 (계속)
 텍스트 검색 216
 평가 116
 평균 색 검색 325
 AS/400용 CM 379
 Content Manager 버전 8 211
 dkQuery 클래스 426
 Domino.Doc 391
 Domino.Doc 구문규칙 391
 Extended Search 398
 GQL(Generalized Query Language) 398
 ImagePlus 구문규칙 375
 Java 116
 매개변수식 유형 117
 조회 오브젝트 유형 116
 텍스트 유형 124
 dkResultSetCursor 및 DKResults 117
 OnDemand 검색 359
 Panagon Image Services 412
 조회 가능 콜렉션
 Java 142
 결과 얻기 142
 조회 가능 및 세분화 144
 평가 142
 프로그래밍 팁 144
 조회 실행 116
 조회 언어 230
 조회 평가 116
 주식
 문서 485
 사용자 조정 496
 상수 174
 서비스 493
 서비스 클래스 486
 엔진 487
 의미 유형 174
 편집 지원 494
 Bean 463, 466
 dkAnnotationExt 클래스 438
 FileNET 410
 OnDemand 367
 XDO에 오브젝트 추가 67
 지속 데이터 서비스(PDS), COBRA 14
 지속 식별자 (PID)
 버전화 151
 지속 식별자(PID)
 경로지정 268

지속 식별자(PID) (계속)
 동적 데이터 오브젝트(DDO) 14
 소개 16
 이미지 검색 331
 확장 데이터 오브젝트(XDO) 15
 지속 오브젝트 서비스, COBRA 14
 지수, 조회 236
 진단 정보,
 참조 : 추적
 [차]
 참조
 속성 유형 160
 조화를 통한 탐색 226
 Content Manager 버전 8 150
 참조 속성 213
 초기화 매개변수 필드, OnDemand 354
 추가 매개변수 필드, OnDemand 354
 추가, BLOB 435
 추적
 리턴 코드 컨테이너 157
 매개변수식 검색 39
 샘플 도구 158
 스택 157
 오류 상태 157
 제어기 servlet 505
 텍스트 검색 38
 Bean 468, 498
 DKException이 있는 오류 처리 157
 FileNET 414
 JNI, servlet 506
 OnDemand 368

[카]

카탈로그, 이미지 검색 326
 캐비닛, Domino.Doc 387
 속성 나열 390
 캐싱, 제어기 servlet 506
 커넥터
 로컬 대 원격 5
 사용자 조정 423
 설명 5
 제어기 servlet 값 504
 커서, 결과 세트
 갱신 432
 닫는 중 433

커서, 결과 세트 (계속)
 열기 433
 요소 갱신
 updateObject 433
 요소 검색 433
 요소 삭제
 deleteObject 433
 요소 지정 433
 요소 추가 433
 위치 지정 433
 유효성 점검 432
 컨텐츠 서버 정보 가져오기 434
 파기 433
 핸들 가져오기 434
 화면이동 432
 컨테이너
 상수 174
 의미 유형 174
 Content Manager 버전 8 149
 컨텐츠 갱신 436
 XDO 69
 컨텐츠 서버
 결과 세트 커서 276
 결과 세트 커서가 속해 있는 위치 식별 434
 공통 사용 권한 가져오기 439
 관계형 데이터베이스 415
 관리 오브젝트 검색 242
 다음 정보에 액세스 276
 다음에 DDO 작성 427
 다음에서 매개변수식 조회 실행 120
 다음에서 매개변수식 조회 평가 122
 다음에서 이미지 조회 실행 333
 다음에서 이미지 조회 평가 335
 다음에서 텍스트 조회 실행 126
 다음에서 텍스트 조회 평가 128
 데이터 관리 13
 데이터 정의 계층 구조 277
 동적 데이터 오브젝트(DDO) 14
 맵핑 정보 등록 428
 문서 또는 폴더 갱신 428
 문서 또는 폴더 검색 428
 문서 또는 폴더 삭제 428
 문서 또는 폴더 이동 428
 문서 또는 폴더 체크아웃 439
 문서 또는 폴더 추가 428
 사용자 관리 클래스 440
 사용자 조정 423

컨텐츠 서버 (계속)

사용자 조정 커넥터에서 feServerDefBase
확장 441
사용자 ID 오브젝트 리턴 427
소개 1
암호 변경 427
양식 오버레이 오브젝트 검색 439
연결 427
연결 해제 중 427
연합 19
연합하여 등록 21
오브젝트 확장 427
이름 리턴 427
이미지 검색 319
작성 275
정의 275
정의에 대해 특정 DK 클래스 432
제어기 servlet 값 504
조회 427
지속 식별자(PID) 16
지원된 커넥터 5
참조 438
클래스 이름 275
트랜잭션 수행 427
폴더에 멤버 추가 438
폴더에서 멤버 제거 438
함수 이름 나열 438
확장 데이터 오브젝트(XDO) 15
확장 클래스 438
2진 데이터 관리 435
AS/400용 Content Manager 378
Bean 462
CLOB 컨텐츠 관리 436
Content Manager 버전 8 145
Content Manager, 이전 버전 277
DDO에 대한 관계 16
Domino.Doc 385
EIP가 지원하는 서버 13
Extended Search 393
OnDemand 353
OS/390용 ImagePlus 370
Panagon Image Services 408
RMI 사용 33
XDO 클래스 439
코드 샘플, 갱신 6
코드 페이지 변환
Windows에 콘솔 서브시스템 설정 37
콘솔 서브시스템, Windows에 설정 37

컬렉션

삭제 113
연합 114
정렬 114
dkCollection 클래스 425
Extended Search 400
컬렉션 및 반복자
C++
메모리 관리 112
Java 109
순차 반복자 109
순차 컬렉션 109
컬렉션 지점
설명 252
정의 258
컬백으로 검색 실행 38, 212
컬백, 실행 38, 212
클라이언트
Information Mining에 대해 작업하기 위해
빌드 5
클라이언트 구성 보조 프로그램 4
클래스 이름 용어 275
키워드, Domino.Doc 387

[타]

태그 라이브러리

관련 검색 500
관련 문서 502
관련 스키마 499
관련 연결 499
관련 폴더 502
관련 항목 500

태그가 있는 문서

텍스트 검색 308

텍스처

명암 321
방향 321
올바른 값 325
정의 321
조밀도 321
조회 예제 325

텍스트 검색

강조표시 129
고급 검색 217
관리 기능 294
구조화된 문서 검색 308, 317
기본 조회 216

텍스트 검색 (계속)

문서 검색 215
문서 모델 나열 309
문서 모델 작성 310
사용자 정의 속성을 검색 가능하게 함
216
색인화 규칙 설정 312
색인화 프로세스 시작 315
여러 색인 조회 124
연산자 216
오브젝트 컨텐츠 검색 215
와일드카드 216
이해 215
정의 38
조회 가능 컬렉션 143
조회 구문규칙 216
조회 문자열 구성 124
조회 실행 125
조회 예제 225
지속 식별자(PID) 297
추적 38
컨텐츠 서버에서 조회 126
컨텐츠 서버에서 조회 평가 128
Bean 473
OnDemand 354
score-basic 217
텍스트 검색 엔진
Java
근접 조회 295
논리 조회 294
데이터 로드 및 색인화 307
텍스트 조회 295
프로그래밍 팁 299
혼용 조회 295
힙 크기 설정 278
GTR 조회 296

템플릿

워크플로우 454
트랜잭션, Content Manager 버전 8 249
고려사항 250
명시적 트랜잭션에 대한 주의사항 250
목록 251
처리 251
체크인 및 체크아웃 250

[파]

파일, 다음에서 XDO 추가 66

패키지

- 클라이언트 및 서버(cs) 30
- Java 서버 30
- Java 클라이언트 30
- Java의 계층 구조 30

패턴 321

페이지

- Bean 502
- FileNET 410

평균 색

- 정의 321

평균 색상

- 올바른 값 324
- 조회 예제 325

폴더

- 갱신 428
- 검색 428
- 다음에 구성원 추가 438
- 다음에 콘텐츠 추가 191
- 다음에서 구성원 제거 438
- 다음에서 콘텐츠 제거 193
- 삭제 428
- 액세스 103
- 원래의 엔티티로서 OnDemand 폴더 367
- 의미 유형 174
- 이동 428
- 이전 CM 워크플로우 343
- 이전 CM의 갱신 287
- 이전 CM의 표시 279
- 제어기 servlet에서 관리 509
- 체크아웃 및 체크인 439
- 추가 428
- 콘텐츠 검색 195
- 특정 DDO가 있는 모든 폴더 획득 196
- Bean 502
- Bean에 표시 475
- Bean에 표현 470
- CM 8에 작성 190
- Content Manager 버전 8 150
- Content Manager 작동 103
- OnDemand에 나열 358
- OnDemand에서 모드 사용 가능 366

표현식 클래스 426

표현식, 조회 228, 237

프로세스

- 경로지정 253
- 계속 266
- 나열 270

프로세스 (계속)

- 연관된 경로에 대한 정의 262
- 일시중단 267
- 재개 267
- 종료 266

프로세스 일시중단 267

프로파일, Domino.Doc 387

필드, Domino.Doc 387

필드, Extended Search 400

필터 조건, FileNET 413

[하]

하위 구성요소

- 데이터 모델의 표현 213
- 도표 148
- 버전화 151
- 참조 150
- CM 8에 작성 173
- Content Manager 버전 8 148
- DDO 속성으로 표시 161

항목

- 데이터 모델의 표현 212
- 링크 197
- 링크된 항목 검색 199
- 문서 항목 유형 작성 241
- 버전화 151, 186
- 연합 19
- 제어기 servlet으로 관리 508
- 참조 161
- 체크인 및 체크아웃 185
- 트리 상수 195
- 폴더를 통해 링크 150
- 폴더에 추가 191
- 폴더에서 콘텐츠 제거 193
- 항목 수정 176
- Bean 463, 498, 500
- CM 8에 속성 설정 및 검색 175
- CM 8에서 검색 182
- Content Manager 버전 8 147

항목 부분

- DDO 및 XDO 비교 16

항목 유형

- 구성요소, 루트 및 하위 148
- 나열 165
- 다음에 대해 속성 나열 170
- 다중 조회 224
- 버전화 151, 188

항목 유형 (계속)

- 분류 163
- 정의 작성 178
- Content Manager 버전 8 147
- Content Manger 버전 8에 작성 163

항목 클래스,

- 참조 : 항목 유형

핸들, 결과 세트 커서 434

헬퍼 Bean 462

확장 가능한 마크업 언어,

- 참조 : XML

확장 데이터 오브젝트,

- 참조 : XDO

확장자 15

환경, 설정

- C++ 33
- Java 31

히스토그램 색

- 정의 321

히스토그램 색상

- 올바른 값 324
- 조회 예제 325

[숫자]

2진 데이터,

- 참조 : XDO

A

addObject 428, 433

addToFolder 438

AIX

- C++용 공유 오브젝트 34
- RMI 서버 시작 33

AllPrivSet 153, 156

anyA, DKAny 108

API(Application Programming Interface)

- 개념 13
- 성능 29
- 소프트웨어 구성요소 159
- 온라인 참조 158
- Java 29
- 복수 검색 37
- 아키텍처 30
- 패키지화 30
- C++와의 차이점 29

APR(Application Programming
Reference) 158
ASCENDING 235
AS/400용 Content Manager
색인 클래스 378
소개 378
엔터티 및 속성 나열 378
연합에서 맵핑 20
조회 실행 379
ATTRONLY, OS/390용 ImagePlus 377
AUTOCOMMIT, 관계형 데이터베이스 416

B

BasicExpression 237
Bean 464
검색 결과 498
검색 기준 499
검색 요청 460
검색 템플릿 498, 499
관련된 Java 보기 프로그램
정의 457
기본 개념 이해 457
기타 빌더 459
데이터 관리 460, 498
데이터 원본
datasource Bean 499
문서 502
문서 서비스 465, 498
버전화 470, 479
보조 463
비주얼
검색 결과 보기 프로그램 474
검색 템플릿 목록 472
검색 템플릿 보기 프로그램 472
검색 패널 473
구성 저장/재저장 479
기본 보기 프로그램 478
단추 숨기기 및 표시 479
대체 480
도움말 이벤트 480
로그온 패널 470
문서 보기 프로그램 477
버전 보기 프로그램 479
보기 프로그램 스펙 477
소개 469
속성 편집기 478
연결 479, 481

Bean (계속)
비주얼 (계속)
외부 보기 프로그램 478
응용프로그램 빌드 481
이름 469
일반 작동 479
정의 457
주요 이벤트 480
창에서 사용 483
텍스트 검색 영역 473
트리 분할 영역 474
특별 작동 480
팝업 메뉴 475
팝업 메뉴 대체 475
폴더 보기 프로그램 475
collation strength 479
비주얼이 아닌
고려사항 467
구성 461
기능 461
등록 정보 468
소개 460
응용프로그램 빌드 468
이벤트 468
정의 457
카테고리 462
빌더에서 사용 459
사용자 관리 498
세션 리스너 466
소개 457
스레딩 467
스키마 460
스키마 관리 498
연결 460, 498, 499
연결 풀
CMBCConnectionPool 498
예외 클래스 466
요구사항 459
워크플로우 460, 463
이벤트 및 리스너 클래스 466
일괄처리 지원 461
작업 목록 460
조회 서비스 498
주석 463, 466
추적 468
추적 로그 498
특성
내성 458

Bean (계속)
특성 (계속)
등록 정보 458
메소드 458
사용자 정의 458
이벤트 458
지속성 458
폴더 502
항목 498
헬퍼 462
호출 460
BeanInfo 클래스 466
CMBSchemaManagement 460
Information Mining 465
JAR 파일 459
MVC (Model View Controller) 461
OnDemand 검색 367
singletons 467
WebSphere Studio Application
Developer에서 빌드 459
web.xml 459
BeanInfo 466
BETWEEN 230, 235
BLOB (Binary Large Object) 435
갱신 435
검색 435
검색 중 435
길이 리턴 435
데이터 복사 434
메소드 434
비동기 열기 435
연결 435
인수 데이터 삽입 435
추가 435
컨텐츠 관리 434
특정 DK 클래스 434
특정 DKPidXDO 클래스 439
파일 핸들러 식별 435
BLOB(Binary Large Object)
클래스 276
ES에서 검색 406
XDO 작성 60
XDO 함수로 테스트 71

BLOB,

참조 : BLOB(Binary Large Object)

C

CC2MIMEFILE, 관계형 데이터베이스 416

cc2mime.ini 416

CCSID 299

changePassword 162, 427

checkedOutUserid 439

checkIn, dkDatastoreExt 439

checkOut, dkDatastoreExt 439

CLOB (Character Large Object)

부분 삭제 437

파일 핸들러 437

CLOB(Character Large Object) 436

컨텐츠 검색 436

컨텐츠 삭제 436

컨텐츠 추가 436

클래스 436

CLOB,

참조 : CLOB(Character Large Object)

cmb81.jar 459

CMBAnnotationPropertiesInterface 496

CMBAnnotationServices 488

CMBAnnotationServicesCallbacks 488

CMBAnnotationSet 488

CMBCC2MIME.INI 408

cmbcc2mime.ini 506

cmbcc2mime.ini.samp 408

cmbclient.ini 4, 505

cmbcm817d.dll 424

cmbcm817.dll 424

cmbcm81d.lib 33

cmbcm81.jar 424

cmbcm81.lib 33

CMBCConnection 367, 460, 498

cmbcs.ini 461, 505

CMBDataManagement 460, 498

cmbdes.ini 393

CMBDocumentServices 491, 498

CMBDocumentViewer 470, 477

스펙 477

종료 477

CMBFolderViewer 470, 475

CMBGenericDocViewre 488

CMBItem 498

CMBItemAttributesEditor 478

CMBLogonPanel 469, 470

CMBObject 501

CMBPage 502

CMBPageAnnotation 494

CMBQueryService 460, 498

cmbregist81.bat 33

cmbregist81.sh 33

CMBSchemaManagement 498

CMBSearchPanel 473

CMBSearchResults 498

CMBSearchResultsView 367

CMBSearchResultsViewer 469, 474

CMBSearchTemplate 498

CMBSearchTemplateList 367, 469, 472

CMBSearchTemplateViewer 367, 470, 472

CMBServletAction 504

cmbservletjsp.properties 503

cmbservlet.properties 504

CMBSTCriterion 499

CMBStreamingDocServices 488

CMBStreamingDocServicesCallbacks 488

cmbsvclient.ini 506

cmbsvcs.ini 506

CMBTraceLog 498

CMBUserManagement 498

cmbview81.jar 487

CMBViewerConfiguration.properties 487

CMCOMMON 4

CMCOMMON_URL 4

cmvcmenv.properties 4

COBRA

지속 데이터 서비스(PDS) 14

지속 오브젝트 서비스 14

CompareOperator 238

com.ibm.mm.sdk.client 30

com.ibm.mm.sdk.common package 424

com.ibm.mm.sdk.server 30

concatReplace 435

contains-text, 텍스트 검색 217

contains-text-basic, 텍스트 검색 216

CONTENT

FileNET 414

OS/390용 ImagePlus 377

Content Manager 버전 8

검색 조회 이해 211

구성요소 설명 146

기본 개념 146

라이브러리 서버 145

Content Manager 버전 8 (계속)

라이브러리 서버 및 자원 관리자 일관성
249

루트 구성요소 148

링크 149

문서 147, 150

문서 갱신 245

문서 검색 247

문서 경로지정 252

문서 관리 240

문서 부분 147

문서 삭제 247

문서 작성 243

버전화 151

보험 시나리오 샘플 159

비자원 항목 147

사용 권한 세트 153

샘플 158

소개 145

속성 147

속성 그룹 관리 168

속성 작성 166

암호 작성 162

액세스 사용 권한 제어 153

액세스 제어 152

사용자 그룹 155

액세스 목록 155

액세스 제어에 대한 작업 200

연결 대상 161

연합에서 맵핑 20

오브젝트 149

응용프로그램 계획 156

응용프로그램 작성 159

자원 관리자 145

자원 관리자에 대한 작업 238

자원 항목 147

자원 항목 유형 정의 178

조회 211

조회 언어 이해 211

조회 예제 220

참조 149, 150

컨텐츠 서버 작성 161

트랜잭션 249

폴더 150

폴더에 대한 작업 189

하위 구성요소 148

항목 147

항목 검색 182

Content Manager 버전 8 (계속)

항목 링크 197
항목 속성 설정 및 검색 175
항목 속성 수정 176
항목 유형 147
항목 유형 나열 165
항목 유형 작성 163
항목 유형에 대해 속성 나열 170
항목 체크인 및 체크아웃 185
eClient 5
SMS 서버 145
XML 91

Content Manager, 이전 버전

대형 오브젝트 처리 278
문서 표시 279
부분 갱신 285
부분 표시 279
소개 277
연합에서 맵핑 20
워크플로우 343
이미지 검색 6, 319
지속 식별자(PID) 279
폴더 갱신 287
폴더 표시 279
eClient 5
XML 저장 95

createChildDDO 172

createDDO 50

cs 패키지 30

C++

관계형 데이터베이스 구성 문자열 61
라이브러리 34
변경된 클래스 10
상수 41
새 클래스 10
순서 이탈 234
오류 메시지 등록 정보 파일 440
커넥터 5
환경 설정 33
AIX용 공유 오브젝트 34
DKAny 104
DKConstant.h 29
DLL 파일 34
XML 지원 29

D

databaseNameStr 162

DataJoiner,

참조 : DB2 DataJoiner

DATASOURCE 393

data_id 56

DB2

구성 문자열 61
클라이언트 구성 보조 프로그램 4, 33
클라이언트 지원 4, 33

DB2 Data Warehouse Manager Information

Catalog,

참조 : Information Catalog

DB2 DataJoiner 415

구성 문자열 61
버전 8.2 제한사항 6

DB2 NSE(Net Search Engine) 38

DB2 확장자 15

db2cliws_dj.bnd 7

db2clprj_dj.bnd 7

ddoDocument 67

ddo.dtd 93

DEFINED, FileNET 412

deleteDKAttributeDef 113

deleteObject 428

DemoSimpleAppl.java 468

DESCENDING 235

DfltACLCode 156

DIGIT 236

DIV 235

dkAbstractWorkFlowUserExit 455

dkAnnotationExt 438

DKAny

메모리 관리 105
버전 8.2 변경사항 11
유형 구성자 사용 105
유형 점검 107
유형 코드 105
유형 코드, 인기 106
컬렉션 내부 사용 109
컬렉션 삭제 113
파기 108
표시 108
프로그래밍 팁 108
할당 106

dkAttrDef 276

클래스 431

DKAttrFieldDefDD 387

DKAttrGroupDefICM 169

DKAttrKeywordDefDD 387

DKAttrProfileDefDD 387

DKBinderDefDD 387

dkBlob 276

메소드 434

클래스 434

DKBlobDL

setToBeIndexed 336

DKBlobFN 410

DKCabinetDefDD 387

dkClob

클래스 436

dkCollection 425

DKCollectionResumeListEntryICM 254

DKConstant

상수

공통 440

DKConstantFN 412

DKConstants 41

DKConstant.h 29

DKCQExpr 412

dkCQExpr 426

DKDatastore 275

사용자 조정 캐릭터 424

dkDatastore

롤백 427

소개 427

실행 427

연결 427

연결 해제 427

평가 427

확약 427

addObject 428

changePassword 427

deleteObject 428

executeWithCallback 427

listDataSourceNames 427

listDataSources 427

listMappingNames 428

moveObject 428

registerMapping 428

registerServices 427

retrieveObject 428

unRegisterMapping 428

unregisterServices 427

updateObject 428

DKDatastoreAccessError 414

DKDatastoreDef

메소드 429

- dkDatastoreDef 275
 - 클래스 428
- DKDatastoreDefDL
 - 추가 함수 429
- DKDatastoreDefOD
 - 추가 함수 429
- DKDatastoreDES
 - 조회 398
 - listEntities 394
 - listEntityAttrs 394
- DKDatastoreDL
 - Java 42
 - 서버 나열 44
 - 스키마 및 스키마 속성 나열 45
 - 연결 42
 - DKDatastoreDL 옵션 43
- dkDatastoreExt
 - 클래스 438
 - 함수 438
- DKDatastoreFN 408
- DKDatastoreICM 159
- DKDatastoreIP 370
- DKDatastoreOD 354
 - listEntities 355
- DKDatastoreQBIC 320
- DKDatastoreTS
 - 속성 297
 - Java 294
 - 서버 나열 300
 - 스키마 나열 301
 - 연결 298
 - DKDatastoreTS 옵션 299
- DKDatastoreV4 378
- DKDatastorexx 275
- DKDatastoreIP
 - listEntityAttrs 373
- DKDatastoreOD
 - listSearchTemplates 358
- dkDDO 424
- DKDDO,
 - 참조: 동적 데이터 오브젝트(DDO)
- DKDLITEMID 297
- DKDocRoutingServiceICM 253
- DKDocRoutingServiceMgmtICM 253
- DKDocumentDefDD 387
- DKDSIZE 297
- dkEntityDef 275
 - 클래스 429

- dkEntityDef (계속)
 - 함수 430
- DKEntityDefIP
 - listAttrNames 372
- DkEntityDefIP
 - getAttr 372
- DKException 39, 157
- DKFederatedCollection 114
- dkFederatedIterator 114
- DKFederatedQuery 22
- dkFederatedQuery 114
- DKFixedView 354
- DKFixedViewDataOD 354
- DKFolder 279
- dkIterator 114
 - 버전 8.2 변경사항 9
- DKLink 197
- DKLinkCollection 199
- DKLITEMID 331
- DKMessageId 440
- DKMessageID, FileNET 414
- DKMessage_en.properties 440
- DKMessage_en_US.properties 440
- DKMessage_es.properties 440
- DKMessage_es_ES.propertie 440
- DKParametricQuery 412
- DKPARTNO 297, 331
- DKParts
 - 이전 CM 279
 - Extended Search 400
- DKPidXDO 439
- DKPrivilegeSetICM 202
- DKProcessICM 253
- dkQuery 426
- dkQueryableCollection 426
- DKRANK 279, 297, 331, 339
- DKRCNT 297
- DKREPTYPE 297, 331
- DKResource 354
- DKResourceGrpOD 354
- DKResults 426
 - 반복자 위치 지정 115
- dkResultSetCursor 276
 - 함수 432
- DKResumeListEntryICM 254
- DKRMConfiguration 162
- DKRoomDefDD 387
- DKRouteListEntryICM 254

- dkSchemaMapping 428
- DKSequentialCollection 285, 287, 425
- dkSequentialIterator 426
- dkServerDef 276
 - 클래스 432
 - 함수 432
- dkSort 114
- DKStorageManageInfo 89
- DKString
 - 버전 8.2 변경사항 11
- DKTimestamp
 - 워크플로우 일시중단 448
- dkUserManagement 440
- DkViewOD 354
- DKViews 354
- DKWorkBasketDL 343
- DKWorkflowFed
 - 일시중단 448
 - 재개 448
- DKWorkflowServiceDL 343
- DKWorkflowServiceFed
 - svWf 447
- DKWorkflowServicesFed 443
- dkWorkflowUserExit 455
- DKWorkItemFed
 - checkIn 453
 - checkOut 453
- DKWorkListFed 450
- DKWorkListICM 254, 261
- DKWorkNodeICM 254, 256
- DKWorkPackageICM 254, 269
- dkXDO 436
 - 버전 8.2 변경사항 9
- dkXDOBase 425
 - 열기 436
- DK_CM_DOCUMENT 279
- DK_CM_FOLDER 279
- DK_CM_OPT_ACCESS_MODE 276
- DK_CM_OPT_CONTENT 414
- DK_CM_PROPERTY_ITEM_TYPE 279
- DK_CM_READWRITE 276
- DK_CM_VERSION_LATEST 184
- DK_DES_GQL_QL_TYPE 398
- DK_DL_OPT_ACCESS_MODE 434
- DK_OPT_TS_CCSSID 299
- DK_OPT_TS_LANG 299
- DK_READWRITE 434
- DK_SS_CONFIG 346, 347

DK_SS_NORMAL 346, 347
 DK_TS_DOCFMT_HTML 308
 DLL 파일
 C++ 34
 DLSEARCH_DocType 117
 doAction 455
 Domino.Doc
 소개 385
 엔티티 및 서브엔티티 나열 388
 연합에서 맵핑 20
 오브젝트 모델 386
 조회 391
 조회 구문규칙 391
 캐비닛 속성 나열 390
 DKResults 오브젝트 조회 117
 ODMA (Open Document Management API) 385
 DSNAME, 관계형 데이터베이스 416
 dsType 367
 DTD(Document Type Definition), XML 가
 저오기 93

E

eClient
 소개 5
 JavaBeans로 작성 457
 EIP 워크플로우 서비스,
 참조 : 워크플로우
 EJB(Enterprise Java Bean) 461
 Enterprise Information Portal
 개념 13
 공통 클래스 424
 구성요소 4
 다중 아키텍처 2
 데이터베이스 14
 데이터베이스 하부 구조 424
 동적 데이터 오브젝트(DDO) 14
 문서 엔진 486
 보험 시나리오 1
 새로운 기능 7
 소개 1
 스키마 맵핑 14
 워크플로우 기능 6
 이주 4
 조직 도표 13
 지속 식별자(PID) 16
 지원된 콘텐츠 서버 13

Enterprise Information Portal (계속)
 커넥터 클래스 5
 확장 데이터 오브젝트(XDO) 15
 ENTITY_TYPE 354, 366
 ErrorCode 157
 ErrorState 157
 ES
 연합 검색 중 408
 ESCAPE,KEYWORD, 조회 236
 ESCAPE_LITERAL 236
 EXCEPT 229, 235
 executeWithCallback 427
 ExpressionList 238
 ExpressionWithOptionalSortBy 237
 Extended Search
 문서 검색 406
 서버 나열 393
 소개 393
 엔티티 및 속성 나열 394
 연합에서 맵핑 20
 컨텐츠 처리 중 400
 필드 처리 중 400
 BLOB 검색 406
 DDO 식별 399
 GQL을 사용하여 조회 398
 MIME 유형과 연관 408
 PID 작성 400

F

FASERVERTYPES 테이블 441
 FeServerDefBase 441
 fetchNext 433
 fetchObject 433
 FileNET,
 참조 : Panagon Image Services
 findObject 433
 FLoadSampleTSQBICDL 304
 FLOAT_LITERAL 236
 fromXML 92
 FTSearch 391
 FunctionName 238
 F_DOCFORMAT 409
 F_DOCTYPE 409

G

getCommonPrivilege 439
 getContent 434
 getContentToClientFile 434, 437
 getDatastore 438
 getOpenHandler 436, 437
 getPosition 433
 GQL(Generalized Query Language)
 표현식 398
 SQL과의 차이점 408
 GUI(Graphical User Interface),
 참조 : Bean, 비주얼

H

HSM(Hierarchical Storage
 Management) 238

I

ICM 커넥터,
 참조 : Content Manager 버전 8
 ICMLogon 154
 IDENTIFIER 237
 imldiag.log 317
 IN RANGE 412
 INBOUNDLINK 213
 indexOf 435, 437
 information center 158
 Information Mining
 고급 검색 샘플 550
 문서 샘플 가져오기 541
 사용자의 콘텐츠 제공자 556
 샘플 511
 샘플 파일의 위치 515
 서비스 API 사용 557
 설명 5
 요약 샘플 524
 웹 크롤러 샘플 541
 응용프로그램 빌드 511
 정보 추출 샘플 529
 카테고리 샘플 515
 카테고리별 검색 550
 클러스터링 샘플 535
 Bean 465, 511
 Bean 지원 462
 JSP 응용프로그램 570

Information Mining의 고급 검색 샘플 550
 Information Mining의 웹 크롤러 샘플 541
 Information Mining의 정보 추출 샘플 529
 Information Mining의 카테고리 샘플 515, 524
 Information Mining의 콘텐츠 제공자 556
 Information Mining의 클러스터링 샘플 535
 Information Mining의 JSP 응용프로그램
 샘플 570
 WAS 전개 570
 Information 카탈로그
 버전 8.2 제한사항 7
 연합에서 맵핑 20
 INTERGER_LITERAL 236
 INTERSECT 229, 235
 IS 235
 isBegin 433
 isCheckedOut 439
 isEnd 433
 isFTIndexed 391
 isInBetween 433
 isOpen 433
 isOpenSynchronous 436, 438
 isScrollable 432
 isSupported 438
 isUpdatable 432
 isValid 432
 ItemAdd 154
 ItemAdminPrivSet 154
 ItemQuery 154
 ItemReadPrivSet 156
 ItemSetSysAttr 154
 ItemSetUserAttr 154
 ItemSQLSelect 154
 ItemTypeQuery 154

J

j2ee.jar 460
 Java
 가비지 콜렉터 29
 문서 보기 프로그램 툴킷 485
 변경된 클래스 9
 상수 41
 새 클래스 8
 순서 이탈 234
 오류 메시지 등록 정보 파일 440
 워크플로우 조치 작성 455

Java (계속)
 커넥터 5
 패키지 30
 환경 설정 31
 DKConstant.h 29
 FeServerDefBase 441
 JVM 스택 크기 증가 367
 XML 함수 29
 XML에 대한 작업 91
 Java Database Connectivity (JDBC) 415
 JavaBeans,
 참조 : Bean
 java.lang.exception 158
 java.lang.objects 175
 JDBC DRIVER, 관계형 데이터베이스 416
 JDBC SERVERS FILE, 관계형 데이터베이스 416
 JDBC SERVERS URL, 관계형 데이터베이스 416
 JNI 추적, servlet 506
 JSP(Java Server Pages)
 servlet 503
 JSP,
 참조 : JSP(Java Server Pages)

K

KEY 412

L

LANG 299
 LETTER 236
 LIKE 230, 235
 LIKE, FileNET 412
 LinkTypeName 197
 ListConstructor 238
 ListContent 238
 listDataSourceNames 427
 listDataSources 162, 427
 listEntityNames 165
 listFunctions
 dkDatastoreExt 438
 listMappingNames 428
 listResourceMgrs 162
 listWorkflowTemplates 454
 listWorkLists 449
 LoadFolderTSQBICDL 304

LOB(handling large objects)
 이전 CM의 처리 278
 LocalPart 238
 LogicalOrSetExpression 237
 LogicalOrSetPrimitive 237
 LogicalOrSetTerm 237

M

MATCH_DICT 129
 MATCH_INFO 129
 MAX_RESULTS, FileNET 414
 Microsoft Visual C++ 컴파일러,
 참조 : Visual C++ 컴파일러
 Microsoft Visual Studio .NET 36
 MIME 유형
 자원 항목 설정 181
 ES에 연관 408
 Mime2App 등록 정보 477
 MOD 235
 moveObject 428
 moveObject, dkDatastoreExt 439
 msg 명령, FileNET 414
 MVC(Model View Controller) 461, 494

N

nameOfAttrStr 176
 NameText 238
 NATIVECONNECTSTRING, 관계형 데이터베이스 415
 NoAccessACL 156
 NodeGenerator 237
 NONZERO 236
 NoPrivSet 153
 NOT 235

O

ODBC(Open Database Connectivity) 415
 구성 문자열 61
 ODMA(Open Document Management API) 385
 OnDemand
 검색 템플릿으로 폴더 사용 367
 다음에서 연결 해제 중 354
 등록 정보 이름 354
 문서 검색 359

OnDemand (계속)
문서 표시 362
비동기 검색 중 366
서버 및 문서 표시 354
서버 정보 나열 355
소개 353
속성 표시 362
연결 대상 354
연합에서 맵핑 20
응용프로그램 그룹 나열 355
응용프로그램 그룹 조회 360
주석 367
추적 368
폴더 나열 358
폴더 모드 사용 가능 366
OptionalExpressionList 238
OptionalPredicateList 237
OR 235
OS/390용 ImagePlus
소개 370
속성 나열 371
엔티티 나열 371
연합에서 맵핑 20
조회 구문규칙 375
조회 매개변수 376
eClient 5
output_option 44

P

Panagon Capture Software 410
Panagon Image Services
문서 및 페이지 표시 410
문서 클래스 410
문제점 해결 414
샘플 412
선택적 검색 매개변수 413
소개 408
엔티티 및 속성 나열 410
연산자 412
조회 412
조회 구문규칙 412
지원된 데이터 유형 409
클래스 409
PENDING, OS/390용 ImagePlus 377
pEnt, C++ 48
Predicae 237
PrivSetCode 153

PublicReadACL 156

Q

QbColor 324
QbColorFeatureClass 321, 324
QbColorHistogramFeatureClass 321, 324, 325
QbDraw 325
QbDrawFeatureClass 321, 325
QbHistogram 324
QBIC(Query by Image Content),
참조 : 이미지 검색
QbTexture 325
QbTextureFeatureClass 321, 325
QName 238

R

REFERENCEDBY 213
REFERENCER 213
registerMapping 428
registerServices 427
removeAllElement 113
removeFromFolder 438
removeMember 286
replaceElementAt 113
ReplayAction, servlet 504
RepType 304
retrieveFromOverlay 439
retrieveObject 428
RMI (Remote Method Invocation)
구성 4
컨텐츠 서버 커넥트 5
RMI 서버,
참조 : RMI(Remote Method Invocation)
RMI(Remote Method Invocation)
Bean과 연결 461
Java API 사용 33
Java 시작 33

S

SAttributeDefinitionCreationICM 147
SConnectDisconnectICM 158
score-basic, 텍스트 검색 216
SDocModelItemICM 150

SDocRoutingDefinitionCreationICM 255, 256, 260, 261, 265
SDocRoutingListingICM 258, 262, 268, 270
SDocRoutingProcessingICM 265, 266, 267, 268, 269
searchTemplate Bean 499
SequencedValue 237
servlet,
참조 : 제어기 servlet
setClassOpenHandler 435, 437
setContent 435
setContentFromClientFile 435, 437
setData 173
setDatastore 438
setInstanceOpenHandler 436, 437
setPosition 433
setToBeIndexed 336
setToFirstCollection 115
setToLastCollection 115
setToNext 433
setToNextCollection 115
setToPreviousCollection 115
SFolderICM 150, 189, 196
SItemCreationICM 150
SItemDeletionICM 185
SItemRetrievalICM 185
SItemTypeCreationICM 148
SItemTypeRetrievalICM 188
SItemUpdateICM 182, 186
SLinksICM 149, 198, 199
SLinkTypeDefinitionCreationICM 199
SMS(System Managed Storage), Content
Manager 버전 8 145
SORTBY 235, 237
SortSpec 237
SortSpecList 237
SOURCEITEMREF 213
SReferenceAttrDefCreationICM 150, 214
SResourceItemMimeTypesICM 181
SSearchICM 182
STRING_LITERAL 236
subString 435, 437
SuperUserACL 156
svWF 447
SYSREFERENCEATTRS 213
SystemAdmin 154
SystemAdminPrivSet 153

SystemDefineItemType 154

T

taglib.tld 459
TARGETITEMREF 224
TCallbackOD 367
TCheckStatusTS 317
TCP/IP
 텍스트 검색 298
TExportICM 92
TExportPackageICM 92
thin 클라이언트 491
TIE(Text Information Extender)
 NSE(Net Search Engine) 38
TImageAnnotation 496
TImportICM 92
toXML 92
TRetrieveFolderWithCallbackOD 367
TRetrieveWithCallbackOD 367
TxdoAdd 샘플 90
TxdoAsyncRetDL 278

U

UNICODE_CHARACTER 236
UNION 229, 235
unlockCheckedOut 439
unRegisterMapping 428
unregisterServices 427
UpdateFTIndex 391
updateObject 177, 428
URI(Uniform Resource Identifier) 239

V

valueObj 176
VideoCharger 145
Visual C++ 컴파일러 33
Visual Studio.NET 36

W

W3C XML 조회 211
wakeUpService 308
WAL PRS_Parse 412
WebSphere Studio Application Developer
 Bean 빌드 459

web.xml 459

Windows

 콘솔 서브시스템 설정 37
 C++ DLL 파일 34
 RMI 서버 시작 33

X

XDO

 갱신 69
 검색 69
 구성원 436
 기억영역 콜렉션 변경 91
 기억영역 콜렉션에 추가 89
 다음에 주식 오브젝트 추가 67
 대형 오브젝트 160
 멀티미디어 콘텐츠 표시 16
 미디어 오브젝트 검색 84
 미디어 오브젝트 추가 76
 버퍼에서 추가 65
 삭제 69, 80
 소개 15
 속성 비교 16
 예제 69
 오브젝트 149
 이미지 검색에서 색인화 336
 이전 CM에서 색인화 61
 자원 항목 작성 180
 클래스 425
 클래스 유형 계층 구조 180
 텍스트 검색 226
 특정 DK 클래스 439
 파일에서 추가 66
 함수 호출 71
 dkBlob 434
 dkXDO 425
 FileNET 410
 Java 59
 데이터 등록 정보 60
 독립형 65
 색인화 303
 프로그래밍 팁 62
 DDO, 부분 63
 PID 60
 MIME 유형 설정 181
 XML 내보내기 97

XML

 가져오기 92

XML (계속)

 가져오기에 대해 DTD(Document Type
 Definition) 93
 내보내기 97
 발췌
 버퍼 96
 웹 주소 96
 샘플 도구 92
 소개 91
 예제 95
 조회 예제 데이터 모델의 표시 221
 추출
 파일 96
 Bean 태그 라이브러리 459
 CM에 저장 94
 CM으로 가져오기 96
 Java 지원 29
 W3C 조회 211
 XML 가져오기 92
 XQuery Path Expressions(XQPE) 211

IBM 한글 지원에 관한 설문



FAX : (02) 3787-0123

보내 주시는 의견은 더 나은 고객 지원 체제를 위한 귀중한 자료가 됩니다.
독자 여러분의 좋은 의견을 기다립니다.

책 제목: 멀티플랫폼용 IBM Content Manager/
컨텐츠용 IBM Information Integrator
워크스태이션 응용프로그램
프로그래밍 안내서 버전 8 릴리스 2

책 번호: SA30-1551-01

성명			직위/담당업무	
회사명			부서명	
주소				
전화번호			팩스번호	
전자우편 주소				
사용중인 시스템	<input type="checkbox"/> 중대형 서버 <input type="checkbox"/> UNIX 서버 <input type="checkbox"/> PC 및 PC 서버			

1. IBM에서 제공하는 한글 책자와 영문 책자 중 어느 것을 더 좋아하십니까? 그 이유는 무엇입니까?
☐ 한글 책자 ☐ 영문 책자
 (이유: _____)
2. 본 책자와 해당 소프트웨어에서 사용된 한글 용어에 대한 귀하의 평가 점수는?
☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가
3. 본 책자와 해당 소프트웨어에서 번역 품질에 대한 귀하의 평가 점수는?
☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가
4. 본 책자의 인쇄 상태에 대한 귀하의 평가 점수는?
☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가
5. 한글 소프트웨어 및 책자가 지원되는 분야에 대해 귀하는 어떻게 생각하십니까?
☐ 한글 책자를 늘려야 함 ☐ 현재 수준으로 만족
☐ 그다지 필요성을 느끼지 않음
6. IBM은 인쇄물 형식(hardcopy)과 화면 형식(softcopy)의 두 종류로 책자를 제공합니다. 어느 형식을 더 좋아하십니까?
☐ 인쇄물 형식(hardcopy) ☐ 화면 형식(softcopy) ☐ 둘 다

☎ IBM 한글 지원 서비스에 대해 기타 제안사항이 있으시면 적어주십시오.

☺ 설문에 답해 주셔서 감사합니다.

귀하의 의견은 저희에게 매우 소중한 것이며, 고객 여러분들께 보다 좋은 제품을 제공해 드리기 위해 최선을 다하겠습니다.



프로그램 번호: 5724-B43

Printed in U.S.A

SA30-1551-01



Spine information:



멀티플랫폼용 IBM Content
Manager/
컨텐츠용 IBM Information
Integrator

워크스테이션 응용프로그램 프로그래밍 안내서

버전 8 릴리스 2