IBM WebSphere Business Integration Adapters
IBM WebSphere InterChange Server

Data Handler 안내서

IBM WebSphere Business Integration Adapters
IBM WebSphere InterChange Server

Data Handler 안내서

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 257 페이지의 『주의사항』의 정보를 읽으십시오.

2003년 12월 19일

이 개정판은 새 개정판에 별도로 명시하지 않는 한 IBM WebSphere InterChange Server 버전 4.2.2, IBM WebSphere Business Integration Adapter Framewotk 버전 2.4 및 모든 후속 릴리스와 수정판에 적용됩니다.

이 문서에 대한 의견을 보내려면 ibmkspoe@kr.ibm.com으로 전자 우편을 보내십시오. 고객의 의견을 기대합니다.

IBM에 정보를 보내는 경우, IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

목차

이 책의 정보 vii
이 책의 사용자 vii
관련 서적 vii
인쇄 규칙. viii

이 릴리스의 새로운 기능 ix
WebSphere InterChange Server v4.2.2 및 WebSphere Business Integration Adapter Framework v2.4.0의 새로운 기능. ix
WebSphere InterChange Server v4.2.1 및 WebSphere Business Integration Adapter v2.3.1의 새로운 기능 . . ix
WebSphere InterChange Server v4.2 및 WebSphere Business Integration Adapter v2.2.0의 새로운 기능 . . . xi
WebSphere Business Integration Adapter Framework v2.1의 새로운 기능 xii
WebSphere Business Integration Adapter Framework v2.0.1의 새로운 기능. xii
WebSphere Business Integration Adapter Framework v2.0의 새로운 기능 xiii
릴리스 4.1.0의 새로운 기능. xiii
릴리스 4.0.1의 새로운 기능. xiii
릴리스 4.0.0의 새로운 기능. xiv

제 1 부 시작하기 1

제 1 장 Data Handler 개요 3
Data Handler 개념 3
Data Handler 인스턴스화 15
Data Handler 호출. 19
메타 데이터 구동 Data Handler 설계 23

제 2 장 Data Handler 설치 및 구성 25
Data Handler 설치. 25
Data Handler 구성. 28
Data Handler를 사용할 커넥터 구성 33

제 3 장 XML Data Handler. 37
개요. 37
Business Object 정의의 요구사항. 41
XML Data Handler 구성 47
DTD를 사용하는 XML 문서 51
스키마 문서를 사용하는 XML 문서 66
Business Object 정의 작성 95
Business Object를 XML 문서로 변환 98
XML 문서에서 Business Object로 변환 100
XML Data Handler 사용자 정의 102

제 4 장 EDI Data Handler. 105
개요 105

EDI Data Handler 구성 107
EDI 문서의 Business Object 정의 111
Business Object를 EDI 문서로 변환 119
Business Object로 EDI 문서 변환 123
EDI Data Handler 사용자 정의 132

제 5 장 Request-Response Data Handler 133
개요 133
Business Object 정의의 요구사항 141
Request-Response Data Handler 구성 146
Requrest Data Handler로 Business Object 변환 149
Response Data Handler로 Business Object 변환 150
오류 처리 151
Request-Response Data Handler 사용자 정의 151

제 6 장 FixedWidth Data Handler 153
개요 153
FixedWidth Data Handler 구성 155
Business Object를 FixedWidth 문서로 변환 158
FixedWidth 문서를 Business Object로 변환 159

제 7 장 Delimited Data Handler 163
개요 163
Delimited Data Handler 구성 164
Business Object를 Delimited 데이터로 변환 168
Delimited 데이터를 Business Object로 변환 169

제 8 장 NameValue Data Handler 173
개요 173
NameValue Data Handler 구성 174
Business Object를 NameValue 데이터로 변환 178
NameValue 데이터를 Business Object로 변환 179

제 9 장 Binary Host Data Handler 183
개요 183
Binary Host Data Handler 구성 187
COBOL 레코드용 Business Object 정의 189
Business Object를 COBOL 레코드로 변환 191
Business Object로 COBOL 레코드 변환 191

제 2 부 Custom Data Handler 193

제 10 장 Custom Data Handler 작성 195
Data Handler 개발 프로세스의 개요 195
Data Handler 개발 도구 198
Data Handler 설계 200
Data Handler 기본 클래스 확장 201
메소드 구현 201

사용자 정의 네임 핸들러 빌드. 215
jar 파일에 Data Handler 추가 217
Data Handler Meta Objects 작성 218
기타 Business Object 설정 221
커넥터 구성 221
국제화된 Data Handler 221

제 11 장 Data Handler 기본 클래스 메소드 225
createHandler() 225
getBO() - 추상 227
getBO() - 공용 228
getBOName() 230
getBooleanOption() 231
getByteArrayFromBO(). 232
getEncoding() 233
getLocale() 233
getOption() 234
getStreamFromBO() 235
getStringFromBO() 236
setConfigMOName() 237
setEncoding() 237
setLocale() 238
setOption() 239
traceWrite(). 240

부록. XML ODA 사용. 241
설치 및 사용법. 241
Business Object Designer에서 XML ODA 사용 245
생성된 Business Object 정의의 내용 254
Business Object 정의에서 정보 수정 255

주의사항 257
프로그래밍 인터페이스 정보 258
상표 및 서비스표 259

색인 261

이 책의 정보

IBM(R) WebSphere(R) Business Integration Adapter 포트폴리오는 앞선 e-business 기술 및 엔터프라이즈 응용프로그램에 대한 통합 연결을 제공합니다. 시스템에는 비즈니스 프로세스 통합 구성요소의 사용자 정의, 작성 및 관리를 위한 도구 및 템플리트가 포함됩니다.

이 책에서는 제공된 Data Handler와 Custom Data Handler 성능에 대해 설명합니다.

이 책의 사용자

이 책은 컨설턴트 및 고객을 위한 것입니다. Business Object와 메타 데이터 오브젝트에 익숙해야 합니다. XML Data Handler를 사용하려면, XML 문서, 현재 XML 표준 및 SAX(Simple API for XML)에 익숙해야 합니다. EDI Data Handler를 사용하려면 EDI 문서와 현재 EDI 표준에 익숙해 있어야 합니다. Data Handler 라이브러리를 확장하려면 Java 프로그래밍 언어에 익숙해야 합니다.

관련 서적

이 제품과 함께 사용할 수 있는 전체 문서 세트에는 모든 WebSphere Business Integration Adapters 설치에 공통되는 사양 및 구성요소에 대한 설명과 특정 구성요소에 관한 참고 자료가 수록되어 있습니다.

다음 사이트로부터 관련된 문서를 설치할 수 있습니다.

- 일반 어댑터 정보, WebSphere 메시지 브로커(WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker)와 함께 어댑터를 사용하는 경우 및 WebSphere Application Server와 함께 어댑터를 사용하는 경우에는 다음 웹 사이트를 참조하십시오.

  http://www.ibm.com/websphere/integration/wbiadapters/infocenter

- InterChange Server와 함께 어댑터를 사용하는 경우에는 다음 웹 사이트를 참조하십시오.

  http://www.ibm.com/websphere/integration/wicserver/infocenter

  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter

- 메시지 브로커(WebSphere MQ Integrator Broker, WebSphere MQ Integrator 및 WebSphere Business Integration Message Broker)에 관한 자세한 정보:

  http://www.ibm.com/software/integration/mqfamily/library/manualsa/

- WebSphere Application Server에 관한 자세한 정보

  http://www.ibm.com/software/webservers/appserv/library.html

이 사이트에서는 문서를 다운로드 및 설치하고 보는 데 필요한 간단한 지시사항을 제공합니다.

인쇄 규칙

이 문서에서는 다음과 같은 규칙을 사용합니다.

courier 글꼴	명령어, 파일 이름, 사용자가 입력한 정보, 시스템이 화면에 인쇄한 정보와 같은 리터럴 값을 나타냅니다.
새로운 용어	처음 사용하는 새로운 용어를 표시합니다.
기울임	변수 이름이나 상호 참조를 표시합니다.
파란색 윤곽선	온라인 매뉴얼을 보는 경우에만 볼 수 있는 파란색 윤곽선은 상호 참조 하이퍼링크를 표시합니다. 참조 오브젝트로 바로 가려면 윤곽선 내부를 누르십시오.
{ }	구문 행에서 중괄호는 그 중에서 단 하나만 선택해야 하는 옵션 세트를 묶습니다.
[ ]	구문 행에서 대괄호는 선택적 매개변수를 묶습니다.
...	구문 행에서 말줄임표는 이전 매개변수의 반복을 표시합니다. 예를 들어 option[,...]은 여러 개의 쉼표로 구분되는 옵션을 입력할 수 있습니다.
< >	이름 지정 규칙에서 꺽쇠괄호는 <server_name><*connector_name*> tmp.log에서처럼 이름의 각 요소를 서로 구별하기 위해 각 요소를 묶습니다.
/, \	이 책에서 백슬래시(\)는 디렉토리 경로의 규칙으로 사용됩니다. UNIX 설치의 경우, 백슬래시를 슬래시(/)로 대체합니다. 모든 IBM WebSphere 제품 경로 이름은 IBM WebSphere 제품이 시스템에 설치되는 디렉토리와 관련되어 있습니다.
ProductDir	제품이 설치된 디렉토리를 나타냅니다. IBM WebSphere InterChange Server 환경의 경우 기본 제품 디렉토리는 ″IBM\WebSphereICS″입니다. IBM WebSphere Business Integration Adapters 환경의 경우 기본 제품 디렉토리는 ″WebSphereAdapters″입니다.
%text% 및 *$text*	퍼센트(%) 내의 텍스트는 Windows text 시스템 변수나 사용자 변수의 값을 표시합니다. UNIX 환경에서 이에 해당하는 표기는 *$text*로, 이는 *text* UNIX 환경 변수의 값을 표시합니다.

이 릴리스의 새로운 기능

이 책의 최근 개정 내역은 다음과 같습니다.

WebSphere InterChange Server v4.2.2 및 WebSphere Business Integration Adapter Framework v2.4.0의 새로운 기능

IBM WebSphere InterChange Server 4.2.2 릴리스와 WebSphere Business Integration Adapter Framework 2.4 릴리스는 Data Handler에 대해 다음과 같은 변경사항을 제공합니다.

- 이들 두 제품은 모두 이제 기본 Data Handler 세트의 파트로 Binary Host Data Handler를 포함합니다. 해당 Data Handler에 관한 자세한 정보는 183 페이지의 제 9 장『Binary Host Data Handler』의 내용을 참조하십시오.
- 이들 두 제품 모두 Request-Response Data Handler에 대한 다음과 같은 변경사항을 지원합니다.
  - Request-Response Data Handler는 텍스트 및 2진 데이터 둘 다에 대한 변환을 지원합니다. 이러한 지원은 2진 호스트 Data Handler를 Request-Response Data Handler와 결합해서 사용할 수 있으므로 가능합니다. Request-Response Data Handler에 관한 자세한 정보는 133 페이지의 제 5 장『Request-Response Data Handler』의 내용을 참조하십시오.
  - Request-Response Data Handler는 기본적으로 MO_DataHandler_DefaultRequestResponseConfig가 기본 저장소에 Data Handler와 함께 제공하는 하위 Meta Object 정의를 사용합니다. 이 Meta Object 정의를 사용하여 Request-Response Data Handler를 구성할 수 있습니다.
- 이들 두 제품 모두 XML Data Handler에 대한 다음과 같은 변경사항을 지원합니다.
  - XML Data Handler는 XML 스키마로부터 작성한 Business Object에서 유형 대체를 지원합니다. 자세한 정보는 74 페이지의『스키마 문서를 기반으로 한 Business Object 정의의 유형 대체』를 참조하십시오.

WebSphere InterChange Server v4.2.1 및 WebSphere Business Integration Adapter v2.3.1의 새로운 기능

IBM WebSphere InterChange Server 4.2.1 릴리스 및 WebSphere Business Integration Adapter Framework 2.3.1 릴리스는 이제 새 Data Handler, Request-Response Data Handler를 제공합니다. 자세한 정보는 133 페이지의 제 5 장『Request-Response Data Handler』를 참조하십시오.

이 두 제품 모두 이제 XML Data Handler의 다음 변경사항을 지원합니다.

- XML 요소 및 속성의 이름을 지정하는 새 방법
  - 이제 XML 요소가 응용프로그램 특정 정보에 elem_name 태그를 포함해야 함을 나타내는 business-object 속성. 자세한 정보는 59 페이지의 『XML 요소의 경우』(DTD에 정의되어 있는 XML 요소의 경우) 또는 85 페이지의 『XML 요소의 경우』(스키마 문서에 정의되어 있는 XML 요소의 경우)를 참조하십시오.
  - 이제 XML 속성이 응용프로그램 특정 정보에 type=attribute 태그 외에 attr_name 태그를 포함해야 함을 나타내는 business-object 속성. 자세한 정보는 60 페이지의 『XML 속성의 경우』(DTD에 정의되어 있는 XML 속성의 경우) 또는 89 페이지의 『XML 속성의 경우』(스키마 문서에 정의되어 있는 XML 속성의 경우)를 참조하십시오.

  주: elem_name 및 attr_name 태그는 business-object 속성의 응용프로그램 특정 정보에 있는 XML 요소 또는 속성의 이름만을 필요로 했던 이전 구문을 대체합니다. XML Data Handler는 계속 기존 Business Object 정의와의 역방향 호환성에 대한 이전 구문을 지원합니다. 그러나 XML ODA는 Business Object 정의 생성 시 새 구문을 사용합니다.
- 스키마 문서의 이름 공간을 지정하는 새 방법

  이제 Business Object 정의에는 응용프로그램 특정 정보에 대상 이름 공간을 식별하기 위한 target_ns 태그가 포함되어야 합니다. 자세한 정보는 77 페이지의 『스키마 이름 공간』을 참조하십시오.

  주: target_ns 태그는 Business Object 정의에 각 이름 공간의 접두부를 정의한 속성이 포함되어야 했던 이전 구문을 대체합니다. 이러한 속성은 응용프로그램 특정 정보에 각각 type=defaultNS 또는 type=xmlns 태그로 기본 이름 공간 또는 접두부 이름 공간을 나타냈는지 여부를 표시했습니다. XML Data Handler는 계속 기존 Business Object 정의와의 역방향 호환성에 대한 이전 구문을 지원합니다. 그러나 XML ODA는 Business Object 정의 생성 시 새 구문을 사용합니다.
- 규정된 XML 요소 및 속성 이름을 지정하는 새 방법

  XML Data Handler는 이제 다음 XML 구조를 인식합니다.
  - 스키마 요소의 elementFormDefault 및 attributeFormDefault 속성

    이들 스키마 속성 값을 기반으로 하여, Business Object 정의의 응용프로그램 특정 정보에 있는 elem_fd 및 attr_fd 태그는 XML 요소와 속성 이름이 규정되어 있는지 여부를 표시합니다. 자세한 정보는 82 페이지의 『규정된 구성요소 이름』 응용프로그램 특정 정보를 참조하십시오.
  - XML 요소 또는 속성의 form 속성

Business Object 정의의 응용프로그램 특정 정보에 있는 elem_fd 및 attr_fd 태그는 이 속성 값을 기반으로 특정 XML 요소 또는 속성 이름이 규정되어 있는지 여부를 표시합니다.

주: elem_fd 및 attr_fd 태그는 일부 Business Object 정의가 XML 요소 또는 속성을 나타낸 속성의 이름에 이름 공간 접두부를 포함해야 했던 이전 구문을 대체합니다. Business Object 정의는 더 이상 이름 공간 접두부 정보를 저장하지 않습니다. XML Data Handler는 계속 기존 Business Object 정의와의 역방향 호환성에 대한 이전 구문을 지원합니다. 그러나 XML ODA는 Business Object 정의 생성 시 새 구문을 사용합니다.

- XML Data Handler는 이제 스키마 문서의 다음 XML 구조를 지원합니다.
  - 각각 XML 요소 및 특성을 나타내는 Business Object 속성의 응용프로그램 특정 정보에 elem_ns 및 attr_ns 태그가 제공된 복수 스키마 이름 공간(import 요소). 자세한 정보는 77 페이지의 『스키마 이름 공간』을 참조하십시오.
  - simple-content 및 complex-content complex 유형에 대한 몇 가지 제한사항(restriction 요소). 자세한 정보는 93 페이지의 『지원되는 스키마 문서 구조』를 참조하십시오.

WebSphere InterChange Server v4.2 및 WebSphere Business Integration Adapter v2.2.0의 새로운 기능

IBM WebSphere InterChange Server 4.2 릴리스와 WebSphere Business Integration Adapter Framework 2.2.0 릴리스는 Data Handler에 대해 다음과 같은 변경사항을 제공합니다.

- 릴리스는 전체 시스템을 설명하거나 구성요소의 이름 또는 도구를 수정하는 데 더 이상 "CrossWorlds" 이름을 사용하지 않습니다. 예를 들면, "IBM CrossWorlds XML Data Handler"는 이제 "IBM WebSphere Business Integration Data Handler for XML"이고 "CrossWorlds InterChange Server"는 이제 "WebSphere InterChange Server"입니다.
- 이제 이러한 IBM WebSphere 제품 중 하나의 일부로서 Data Handler를 설치할 수 있습니다.
  - IBM WebSphere InterChange Server: IBM에서 제공하는 모든 Data Handler는 이 제품의 일부입니다.
  - IBM WebSphere Business Integration Adapters: XML Data Handler만 이 제품의 일부입니다.

  자세한 정보는 25 페이지의 『Data Handler 설치』를 참조하십시오.
- WebSphere InterChange Server 4.2 릴리스는 MO_Server_DataHandler Meta Object의 기본 구성을 변경합니다. 이 Meta Object는 이제 더미 속성만 포함합니다. 따라

서 기본적으로 Server Access Interface 프로세스는(InterChange Server 내의) Data Handler를 전혀 지원하지 않습니다. 이 Meta Object가 Server Access Interface 프로세스에 Data Handler 지원을 제공하도록 구성하는 방법은, 29 페이지의 『MO_Server_DataHandler Meta Object』를 참조하십시오.

WebSphere Business Integration Adapter Framework v2.1의 새로운 기능

WebSphere Business Integration Adapter Framework 2.1.0 릴리스에서는 다음 XML 데이터 모델에서 Business Object 정의를 작성할 수 있는 새로운 XML ODA를 제공합니다.

- XML DTD(문서 유형 정의)

  XML Data Handler의 이전 버전은 XML DTD와 Business Object 정의 사이에서 변환을 수행했으며, DTD에서 Business Object 정의를 작성하는 데 두 가지 외부 도구를 사용했습니다. 이 기능성이 이제 XML ODA에 의해 제공됩니다. 자세한 정보는 51 페이지의 『DTD를 사용하는 XML 문서』를 참조하십시오.

- XML 스키마 문서

  스키마 문서와 Business Object 정의 사이의 변환을 위한 지원은 XML ODA의 새로운 기능입니다. 자세한 정보는 66 페이지의 『스키마 문서를 사용하는 XML 문서』를 참조하십시오.

XML ODA의 사용에 대한 정보는 95 페이지의 『XML ODA를 사용하여 Business Object 정의 작성』을 참조하십시오.

WebSphere Business Integration Adapter Framework v2.0.1의 새로운 기능

WebSphere Business Integration Adapter Framework 2.0.1 릴리스에서는 다수의 어댑터 및 Data Handler의 국제화 버전을 제공합니다. 이 국제화 제품은 영어 및 일본어 로케일용으로 지역화되었습니다(로케일은 문화 고유의 규약과 문자 코드 세트를 포함합니다). 세부사항은 다음 섹션을 참조하십시오.

- Data Handler API는 현재 Custom Data Handler가 Data Handler 환경의 로케일 및 문자 인코딩에 액세스할 수 있게 해주는 메소드를 제공합니다.
  - 로케일에 액세스하려면: getLocale() 및 setLocale()
  - 문자 인코딩에 액세스하려면: getEncoding() 및 setEncoding()
- Data Handler를 국제화하는 방법에 대한 개요는 221 페이지의 『국제화된 Data Handler』에 나와 있습니다.

WebSphere Business Integration Adapter Framework v2.0의 새로운 기능

Data Handler는 이제 WebSphere MQ Integrator Integration Broker에서 지원됩니다. 이 통합 브로커에 대한 자세한 정보는 WebSphere Business Integration Adapters 문서 세트의 *WebSphere MQ Integrator Broker 구현 안내서*를 참조하십시오. Data Handler는 InterChange Server를 통합 브로커로 계속 지원합니다.

릴리스 4.1.0의 새로운 기능

- Business Object 정의에 있는 요소에 해당하는 속성이 있을 경우, XML Data Handler는 이제 DOCTYPE 및 XML 선언과 같은 프롤로그 정보를 채웁니다.
- Delimited Data Handler는 이제 분리문자로 복수 문자들이 있는 문자열을 사용할 수 있습니다.

릴리스 4.0.1의 새로운 기능

- Chem eStandards의 TPI 지원을 위해 `tpi_rnif` Data Handler가 `CwDataHandler.jar`에 추가되었습니다.

XML Data Handler에 대해 중요한 성능 개선이 이루어졌습니다. 대부분의 성능이 뚜렷하게 개선되었으나 일부는 그렇지 않습니다.

- 이스케이프 처리를 필요로 하는 내용이 있는 XML 속성이나 XML 요소를 표시하는 Business Object 속성은 이제 `escape=true` 응용프로그램 특정 정보를 사용해야 합니다. Data Handler의 이전 버전에서는, 모든 속성이 기본적으로 이스케이프 처리되었습니다. 이 최신 버전의 XML Data Handler에서는 속성에 `escape=true` 응용프로그램 특정 정보가 없을 경우 이스케이프 처리되지 않습니다. 값에 작은 따옴표(')나 큰 따옴표, &, < 또는 > 문자가 있는 XML 요소를 속성이 표시할 경우, 속성은 이스케이프 처리를 필요로 합니다. 이러한 새 응용프로그램 특정 정보(`escape=true`)는 기존 텍스트 끝에 위치해야 합니다. 예를 들면 다음과 같습니다.

```
[Attribute]
Name=Data
Type=String
AppSpecificInfo=Price;type=pcdata;escape=true
[End]
```

- Business Object를 XML로 변환할 때 사용하는 버퍼의 *초기 크기*를 정의하기 위해 새 XML Data Handler 하위 Meta Object 속성인 `InitialBufferSize`를 만들었습니다. 이 값은 XML Business Object *크기*(바이트)로 설정하십시오. 이 값을 큰 수로 설정하면 직렬화된 XML로의 Business Object 변환 속도가 빨라집니다. 기본값은 2MB입니다.

- 새 XML Data Handler 하위 Meta Object 속성인 UseNewLine을 만들었습니다. 출력 XML의 각 태그가 새 행에 놓이도록 하려면 이 속성을 true로 설정하십시오. (XML Data Handler는 줄 바꾸기 및 캐리지 리턴 양식에서 여분의 내용을 XML 문서에 추가합니다.) XML 출력을 변경하지 않으려면 false로 설정하십시오.

릴리스 4.0.0의 새로운 기능

- Apache의 Xerces Parser는 이제 XML Data Handler에 대한 기본 구문 분석기입니다. 기본값 대신 IBM의 SAX Parser 사용에 대한 지시사항이 제공됩니다.
- Edifecs SpecBuilder 유틸리티는 이제 Business Object 정의를 작성용으로 많이 사용하는 XML Data Handler 도구입니다.
- 속성 OmitObjectEventId가 FixedWidth 및 Delimited Data Handler에 추가되었습니다.

제 1 부 시작하기

제 1 장 Data Handler 개요

이 장에서는 WebSphere Business Integration System Data Handler를 소개합니다. Data Handler는 Business Object를 직렬화된 데이터로 변환하고 직렬화된 데이터를 Business Object로 변환합니다. 이러한 직렬화된 데이터는 응용프로그램이 읽을 수 있는 형식(문자열 또는 입력 스트림)으로 되어 있습니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『Data Handler 개념』
- 15 페이지의 『Data Handler 인스턴스화』
- 19 페이지의 『Data Handler 호출』
- 23 페이지의 『메타 데이터 구동 Data Handler 설계』

Data Handler 개념

*Data Handler*는 고유의 직렬화된 형식과 Business Object 사이에 변환하는 Java 클래스입니다. WebSphere Business Integration Broker와 일부 외부 프로세스 사이에 정보를 전송하는 비즈니스 통합 시스템의 구성요소가 Data Handler를 사용합니다. 표 1에서는 WebSphere Business Integration Broker와 외부 프로세스 사이에 정보를 전송하는 것을 처리하는 구성요소를 보여줍니다.

표 1. WebSphere Business Integration System에서 정보를 전송하는 구성요소

구성요소	목적	자세한 정보
어댑터	WebSphere Business Integration Broker와 외부 프로세스(예: 응용프로그램 또는 기술) 사이에 정보를 전송하는 것을 처리합니다. **주:** 어댑터는 *connector*라는 런타임 구성요소를 사용하여 통합 브로커와 응용프로그램(또는 기술) 간의 정보 전송을 실제로 처리합니다. 이 외부 프로세스는 이벤트 저장소로 이벤트 레코드를 전송하여 이들 내에서 발생하는 이벤트를 식별합니다. 어댑터가 이 이벤트 저장소에서 이벤트를 발견합니다. 어댑터가 트리거링 이벤트를 찾으면, 이벤트를 나타내는 Business Object를 작성하고 이 이벤트를 *비동기적*으로 비즈니스 통합 브로커에 전송합니다. 이 Business Object에는 데이터와 이벤트의 유형을 나타내는 verb(예: Create 또는 Update)가 들어 있습니다.	IBM 제공 어댑터의 경우, 개별 어댑터 안내서를 참조하십시오. 사용자 정의 어댑터의 경우, 사용자 정의 커넥터에 대한 정보는 커넥터가 구현되는 언어에 따라 *Connector Development Guide for Java* 또는 *Connector Development Guide for C++*를 참조하십시오.

표 1. WebSphere Business Integration System에서 정보를 전송하는 구성요소 (계속)

구성요소	목적	자세한 정보
액세스 클라이언트	InterChange Server 통합 브로커와 일부 외부 프로세스(예: 웹 서버 내의 Servlet) 사이에 정보가 전송되는 것을 처리합니다. 액세스 클라이언트는 서버 액세스 인터페이스를 사용하여 InterChange Server와 직접 통신하는 외부 프로세스입니다. 이 구성요소가 전송되어야 하는 일부 정보를 받으면, 이벤트를 나타내는 Business Object를 작성하고 이 이벤트를 *비동기적*으로 InterChange Server 내에 있는 협업으로 전송합니다. 어댑터와 같이, Business Object에는 데이터와 이벤트의 유형을 나타내는 verb(예: Create 또는 Update)가 들어 있습니다.	Server Access Interface Development Guide

표 1에서처럼 이 구성요소 모두(커넥터 및 액세스 클라이언트)의 타스크는 다음과 같이 브로커와 외부 프로세스 사이에 정보를 전송하는 것입니다.

- 통합 브로커에 정보를 전송하기 위해 이들 구성요소는 *그것*을 Business Object로 형식화해야 합니다.
- 정보를 외부 프로세스로 전송하기 위해 이들 구성요소는 *그것*을 기본 직렬화된 형식으로 변환해야 합니다.

흔히 외부 프로세스는 기본 직렬화된 데이터를 위해 공통된 형식(예: XML)을 사용합니다. 모든 어댑터(또는 액세스 클라이언트)가 이 공통된 형식과 Business Object 사이의 전송을 처리하게 하기 보다는 WebSphere Business Integration System이 여러 개의 IBM 제공 Data Handler를 제공합니다. 또한, 사용자의 기본 형식 사이의 변환을 처리하기 위해 Custom Data Handler를 작성할 수도 있습니다. 그런 다음, 어댑터(또는 액세스 클라이언트)가 직렬화된 데이터의 MIME(Multipurpose Internet Mail Extensions) 유형을 기본으로 하여 데이터 변환을 수행하기 위해 해당 Data Handler를 호출할 수 있습니다.

주: Data Handler는 `DataHandler` Java 클래스에서 구현됩니다. 이 클래스는 Data Handler 인스턴스를 구현하기 위해 Data Handler 개발자가 확장하는 추상 클래스입니다. 자세한 정보는 201 페이지의 『Data Handler 기본 클래스 확장』을 참조하십시오.

이 섹션에서는 Data Handler에 대한 다음 정보를 제공합니다.

- 5 페이지의 『IBM 제공 Data Handler』
- 6 페이지의 『Data Handler Meta Object』
- 7 페이지의 『Data Handler 호출에 필요한 문맥』

IBM 제공 Data Handler

IBM은 표 2에서처럼 Java 아카이브(jar) 파일로 Data Handler를 제공합니다. 이 jar 파일은 제품 디렉토리의 DataHandlers 서브디렉토리에 있습니다.

표 2. IBM 제공 Data Handler jar 파일

내용	설명	Data Handler jar 파일
기본 Data Handler	텍스트를 기본으로 하는 Data Handler와 일부 IBM 제공 어댑터에 특정한 Data Handler	CwDataHandler.jar
특수 Data Handler	XML Data Handler	CwXMLDataHandler.jar
	EDI Data Handler	CwEDIDataHandler.jar
Custom Data Handler	사용자가 구현하는 Data Handler	CustDataHandler.jar

기본 Data Handler

기본 Data Handler 파일인 CwDataHandler.jar은 대부분의 IBM 제공 Data Handler를 포함합니다. 이 파일은 제품 디렉토리의 DataHandlers 서브디렉토리에 상주합니다. 표 3에서는 이러한 기본 Data Handler 파일이 포함하는 기본 Data Handler를 보여줍니다.

표 3. 기본 Data Handler 파일의 기본 Data Handler

Data Handler	MIME 유형	자세한 정보
Request-Response Data Handler	text/requestresponse	133 페이지의 제 5 장『Request-Response Data Handler』
FixedWidth Data Handler	text/fixedwidth	153 페이지의 제 6 장『FixedWidth Data Handler』
Delimited Data Handler	text/delimited	163 페이지의 제 7 장『Delimited Data Handler』
NameValue Data Handler	text/namevalue	173 페이지의 제 8 장『NameValue Data Handler』
Binary Host Data Handler	N/A	183 페이지의 제 9 장『Binary Host Data Handler』

주: 이 매뉴얼에서는 표 3에 나열된 텍스트 Data Handler에 대해 설명합니다. 기본 Data Handler 파일은 특정 IBM 제공 어댑터에 고유한 몇 개의 Data Handler도 포함합니다. IBM 어댑터가 특수 Data Handler를 사용하면, 어댑터 안내서가 Data Handler의 설치, 구성 및 사용에 대해 설명합니다.

특수 Data Handler

IBM은 몇몇 Data Handler에 사용 가능한 별도의 설치 프로그램을 작성합니다. 이러한 특수 Data Handler를 설치하려면, *WebSphere Business Integration Adapters용 설치 안내서*에 제공된 단계를 수행해야 합니다.

Data Handler를 기본 Data Handler 파일과 분리하면, 많은 어댑터가 기본 Data Handler 파일에 있는 다른 Data Handler를 저장하는 오버헤드를 일으키지 않고 Data Handler를 사용할 수 있습니다. 표 4에서는 IBM이 별도의 설치 프로그램 및 별도의 jar 파일을 제공하는 Data Handler를 보여줍니다.

표 4. 별도의 jar 파일과 IBM 제공 Data Handler

Data Handler	Data Handler jar 파일	MIME 유형	자세한 정보
XML Data Handler	CwXMLDataHandler.jar	text/xml	37 페이지의 제 3 장『XML Data Handler』
EDI Data Handler	CwEDIDataHandler.jar	edi	105 페이지의 제 4 장『EDI Data Handler』

Custom Data Handler

IBM 제공 Data Handler가 직렬화된 데이터를 Business Object로 변환하는 것을 처리하지 *못하면*, 사용자의 Custom Data Handler를 작성할 수 있습니다. CustDataHandler.jar 파일은 개발할 수 있는 Custom Data Handler를 보유하기 위한 것입니다. 이 파일은 제품 디렉토리의 DataHandlers 서브디렉토리에 상주합니다. Custom Data Handler 작성 방법에 대한 정보는 195 페이지의 제 10 장『Custom Data Handler 작성』을 참조하십시오.

주: Custom Data Handler를 개발하는 것을 돕기 위해 FixedWidth, Delimited 및 NameValue Data Handler의 소스 코드도 샘플 코드로 제공됩니다. 자세한 정보는 198 페이지의『샘플 Data Handler』를 참조하십시오.

Data Handler Meta Object

커넥터 또는 서버 액세스 인터페이스 프로세스(InterChange Server가 통합 브로커일 경우)가 입력 파일의 MIME 유형이나 Business Object 요청에 지정한 MIME 유형을 기초로 하여 Data Handler의 인스턴스를 작성합니다.

Data Handler Meta Object는 임의의 개수의 하위 오브젝트를 포함할 수 있는 계층 구조의 Business Object입니다. Data Handler 구성 정보는 다음의 계층 구조로 배열됩니다.

- *최상위 레벨* Meta Object에는 다른 Data Handler가 지원할 수 있는 MIME 유형에 대한 정보를 포함합니다. 각각의 최상위 레벨 속성은 Data Handler 인스턴스에 대해 하위 Meta Object를 참조하는 카디널리티 1 속성입니다. 각각의 속성은 하나의 MIME 유형과, 조작할 수 있는 Data Handler를 표시합니다.
- *하위* Meta Object는 특정 Data Handler에 대한 실제 구성 정보를 포함합니다. 각 속성은 구성 등록 정보를 나타내고 기본값, 유형과 같은 정보를 제공합니다.

주: Data Handler는 Meta Object를 사용하여 구성 정보를 보유할 필요가 없습니다. 그러나 IBM이 제공하는 모든 Data Handler는 해당 구성 정보를 위해 Meta Object를 사용하도록 설계되었습니다.

Data Handler Meta Object는 커넥터 또는 서버 액세스 인터페이스 프로세스(InterChange Server가 통합 브로커일 경우)가 입력 파일이나 Business Object 요청에 지정한 MIME 유형 또는 입력 파일의 MIME 유형을 기초로 Data Handler의 인스턴스를 생성할 수 있게 합니다. Data Handler를 구성하려면, Meta Object를 올바르게 초기화하여 호출자(커넥터 또는 액세스 클라이언트)가 사용할 수 있도록 해야 합니다.

주: 각 IBM 제공 Data Handler는 Data Handler Meta Object에 정의된 구성 등록 정보를 사용합니다. 그러나 Custom Data Handler는 구성 등록 정보에 대해 Meta Object를 사용할 수도, 사용하지 않을 수도 있습니다. 자세한 정보는 200 페이지의 『Data Handler Meta Object 사용』을 참조하십시오.

Data Handler 호출에 필요한 문맥

3 페이지의 표 1에서 설명한 것처럼 WebSphere Business Integration System에서 데이터를 전송해야 하는 구성요소가 Data Handler를 호출할 수 있습니다. 표 5에서는 Data Handler를 호출할 수 있는 구성요소에 대한 추가 정보를 제공합니다.

표 5. Data Handler 호출에 필요한 문맥

구성요소	이벤트 통신 유형	플로우 유형	**Data Handler**를 호출하는 소프트웨어
어댑터	비동기	이벤트 트리거 플로우	커넥터
액세스 클라이언트 (InterChange Server 통합 브로커만)	동기	호출 트리거 플로우	서버 액세스 인터페이스 (InterChange Server 내의)

표 5에서처럼 이벤트 트리거 플로우에서는 어댑터가 직접 Data Handler를 호출합니다. 호출 트리거 플로우에서, 서버 액세스 인터페이스(액세스 클라이언트라고 함)를 사용하는 외부 프로세스는 Data Handler에 대한 호출을 초기화합니다. Data Handler는 어댑터에 의해 직접 호출하거나 액세스 클라이언트에 의해 간접적으로 호출해도 동일하게 작동합니다. 이러한 문맥은 다음 섹션에서 설명합니다.

커넥터 문맥의 Data Handler

이벤트 트리거 플로우에서, *커넥터*라는 어댑터의 런타임 구성요소가 직접 Data Handler와 상호작용하여 데이터를 변환합니다

주: IBM 제공 어댑터의 경우, 각각의 어댑터 안내서를 참조하십시오. 사용자 정의 어댑터의 경우, 어댑터를 구현한 언어에 따라 *Connector Development Guide for Java* 또는 *Connector Development Guide for C++*를 참조하십시오. 이러한 안내서는 WebSphere Business Integration Adapters 문서 세트의 일부입니다.

커넥터가 Data Handler를 호출할 경우, Data Handler는 커넥터 프로세스의 일부로 실행됩니다. 그림 1에서는 커넥터 문맥에서의 Data Handler를 보여줍니다.

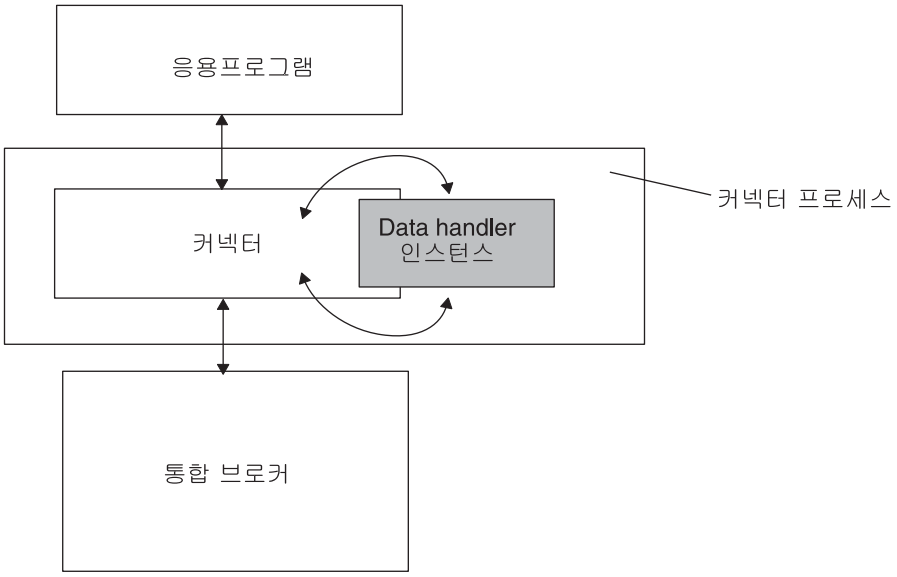

그림 1. 커넥터 문맥의 Data Handler

데이터 변환은 Business Object 요구사항 및 플로우 방향을 반영합니다.

- 커넥터가 Business Object 요청 처리를 처리할 때 Business Object에서 문자열로의 변환을 위해 Data Handler를 호출합니다.
- 커넥터가 이벤트 공고를 처리할 때 문자열에서 Business Object로의 변환을 위해 Data Handler를 호출합니다.

커넥터의 Business Object에서 문자열로의 변환: Business Object에서 문자열로의 변환의 경우, 커넥터가 Data Handler를 호출하고 Business Object를 전달합니다. Data Handler는 Business Object에 있는 정보와 오브젝트 정의를 사용하여 데이터 스트림이나 문자열을 작성합니다. 이러한 데이터 스트림 또는 문자열은 Data Handler(보통 특정 MIME 유형의)와 연관된 형식으로 되어 있습니다. Business Object에서 문자열로 변환하는 것은 커넥터가 통합 브로커에서 Business Object 양식으로 정보를 받을 때 유용합니다. 그러면, 커넥터는 Business Object의 정보를 직렬화된 데이터로서 응용프로그램(또는 기술)에 전송해야 합니다.

그림 2에서는 Data Handler가 Business Object에서 문자열로의 변환을 처리할 때 커넥터 문맥에서의 Data Handler를 보여줍니다.

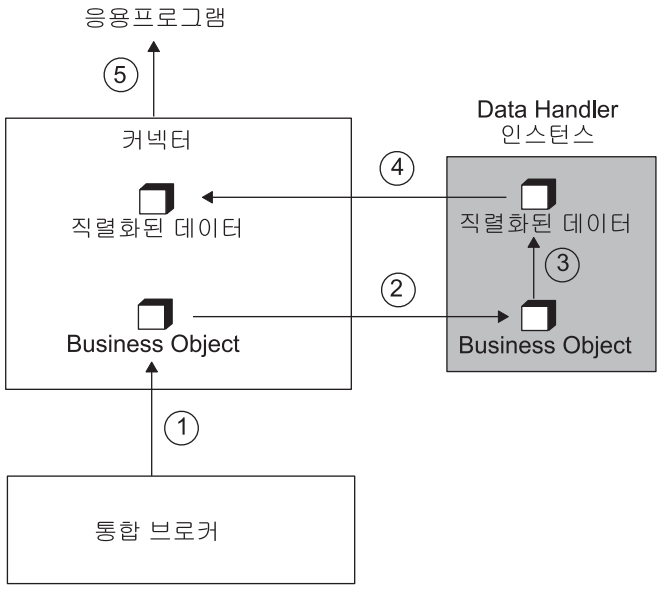

그림 2. 커넥터 문맥의 Business Object에서 문자열로의 변환

1. 커넥터는 통합 브로커로부터 Business Object를 수신합니다.
2. 커넥터는 Data Handler의 인스턴스를 작성하여 Business Object를 처리합니다 (`DataHandler` 기본 클래스의 `createHandler()` 정적 메소드 사용).

   커넥터가 Data Handler를 인스턴스화하는 방법에 대한 자세한 정보는 20 페이지의 『커넥터 문맥에서의 인스턴스화』를 참조하십시오.
3. 커넥터는 다음 data-handler 메소드 중 하나를 호출하여 Business Object에서 문자열로의 변환을 요청합니다.
   - `getStreamFromBO()`
   - `getStringFromBO()`
   - `getByteArrayFromBO()`

   커넥터는 Business Object를 인수로서 이 메소드에 전송합니다. Data Handler는 오브젝트를 요청한 데이터 형식으로 직렬화합니다.
4. Data Handler는 직렬화된 데이터를 커넥터에 리턴합니다.
5. 커넥터는 직렬화된 데이터를 목적지(전자 우편, 파일 또는 HTTP 연결)에 기록합니다.

커넥터의 문자열에서 Business Object로의 변환: 문자열에서 Business Object로의 변환의 경우, 커넥터가 Data Handler를 호출하고 직렬화된 데이터와 연관된 MIME 유형 오브젝트를 전달합니다. Data Handler는 데이터의 스트림 또는 문자열을 수신합니다. Data Handler는 데이터 스트림에 있는 정보를 사용하여 지정한 유형의 Business Object를 작성하고, 이름을 지정한 후 정보로 채웁니다. 문자열에서 Business Object

로의 변환은 커넥터가 이벤트를 통합 브로커에 전송해야 할 경우에 유용합니다. 응용프로그램은 이 이벤트를 직렬화된 데이터(특정 MIME 유형을 가지는)로서 커넥터에 전송합니다.

그림 3에서는 Data Handler가 문자열에서 Business Object로의 변환을 처리할 때 커넥터 문맥에서의 Data Handler를 보여줍니다.

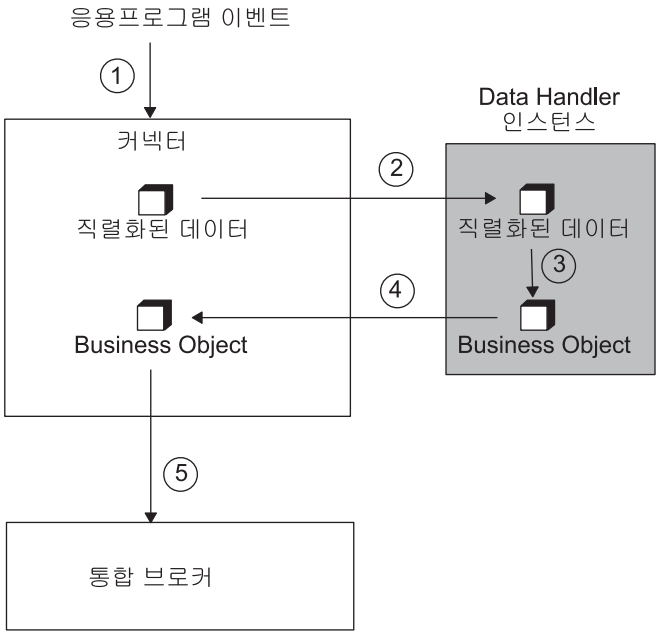

그림 3. 커넥터 문맥의 문자열에서 Business Object로의 변환

1. 커넥터는 응용프로그램 이벤트를 발견합니다. 이벤트는 전자 우편, 텍스트 파일, XML 문서 또는 Data Handler가 존재하는 다른 공통 형식일 수 있습니다.
2. 커넥터는 Data Handler의 인스턴스를 작성하여 이벤트를 처리합니다(`DataHandler` 기본 클래스의 `createHandler()` 정적 메소드 사용).

   커넥터가 Data Handler를 인스턴스화하는 방법에 대한 자세한 정보는 20 페이지의 『커넥터 문맥에서의 인스턴스화』를 참조하십시오.
3. 커넥터는 직렬화된 데이터에서 인수로서 data-handler 인스턴스의 `getBO()` 메소드로 전송합니다. Data Handler는 Business Object 인스턴스를 빌드합니다.

   커넥터는 `getBO()` 메소드가 데이터를 변환하는 Business Object를 지정할 수도 있습니다. 일부 커넥터는 Business Object 유형을 지정하고, 다른 커넥터는 Data Handler가 직렬화된 텍스트로부터 Business Object 텍스트를 추출할 수 있는 것으로 가정합니다. Data Handler는 데이터를 구문 분석하고 직렬화된 데이터를 기반으로 Business Object에 대한 속성 값을 채웁니다.
4. Data Handler는 Business Object를 커넥터에 리턴합니다.
5. 커넥터는 Business Object를 통합 브로커에 전송합니다.

서버 액세스 인터페이스 문맥에서의 Data Handler

호출 트리거 플로우에서, 커넥터는 Data Handler와 상호작용하여 데이터를 변환합니다. *액세스 클라이언트*는 InterChange Server와 사용작용하기 위해 서버 액세스 인스턴스를 사용하는 외부 프로세스입니다. 액세스 클라이언트가 `ItoExternalForm()` 또는 `IcreateBusinessObjectFrom()`이라는 서버 액세스 인터페이스 API의 메소드를 호출할 경우, Data Handler로의 호출을 시작합니다. InterChange Server 프로세스의 일부로 실행되는 서버 액세스 인터페이스는 실제로 Data Handler를 호출합니다.

InterChange Server

서버 액세스 인터페이스는 액세스 클라이언트가 InterChange Server 내에서 협업을 실행할 수 있도록 하는 API입니다. 이 인터페이스는 InterChange Server가 통합 브로커인 *경우에만* 사용 가능합니다. 이 인터페이스 및 액세스 클라이언트에 대한 자세한 정보는 IBM WebSphere InterChange Server 문서 세트에 있는 *Access Development Guide*를 참조하십시오.

액세스 클라이언트는 클라이언트 브라우저로부터 요청을 처리하는 Servlet이 될 수도 있습니다. 요청은 데이터 요청, 주문 요청 또는 비즈니스 간 트랜잭션의 또다른 유형이 될 수 있습니다. 또다른 예로, 액세스 클라이언트는 서버 액세스 인터페이스를 사용하여 InterChange Server에 액세스하고 다른 응용프로그램과 데이터를 교환하는 C++ 또는 Java 프로그램이 될 수 있습니다.

액세스 클라이언트가 Data Handler를 필요로 하는 호출을 초기화할 경우, Data Handler는 InterChange Server의 일부로 프로세스를 실행합니다. 그림 4에서는 서버 액세스 인터페이스 문맥에서의 Data Handler를 보여줍니다. 이 예에서, 액세스 클라이언트는 웹 서버 및 Servlet입니다.

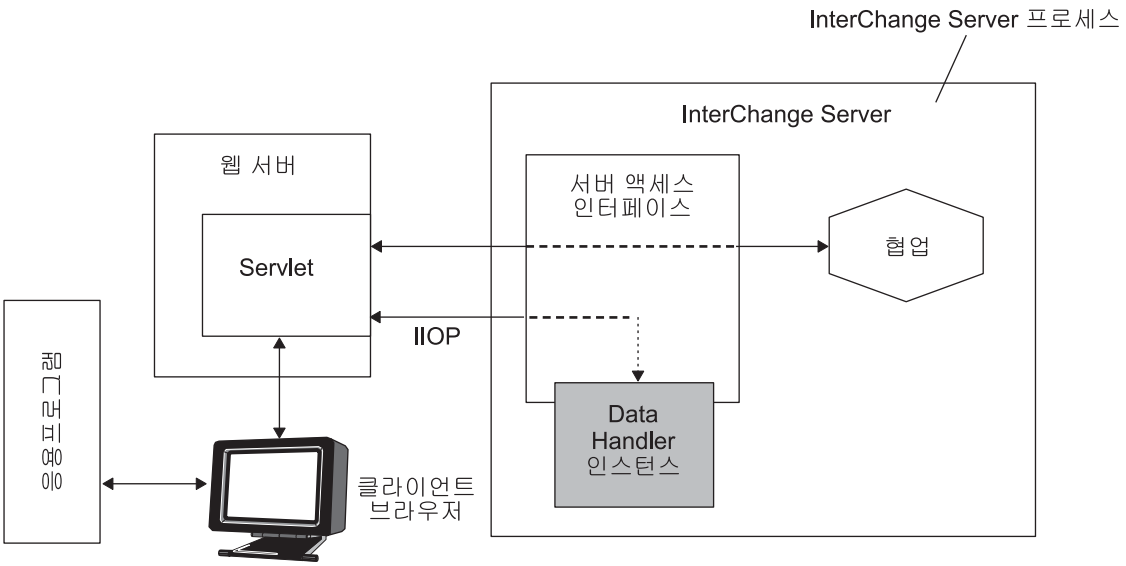

그림 4. 서버 액세스 인터페이스 문맥에서의 Data Handler

데이터 변환은 Business Object 요구사항 및 플로우 방향을 반영합니다.

- InterChange Server로부터 Business Object를 수신할 경우, 액세스 클라이언트는 Data Handler가 Business Object에서 문자열로의 변환을 수행하도록 요청합니다.
- InterChange Server로 데이터를 전송할 경우, 액세스 클라이언트는 Data Handler가 문자열에서 Business Object로의 변환을 수행하도록 요청합니다.

서버 액세스 인터페이스의 Business Object에서 문자열로의 변환: Business Object에서 문자열로의 변환의 경우, Data Handler는 Business Object를 협업 실행 결과로 수신합니다. Data Handler는 Business Object에 있는 정보를 사용하여 데이터 스트림 또는 문자열을 작성합니다. 이 데이터는 Data Handler(보통 특정의 MIME 유형)와 연관된 형식으로 되어 있습니다. 액세스 클라이언트는 종종 결과로 생성되는 Business Object를 직렬화된 데이터로 응용프로그램에 전송합니다.

그림 5에서는 Data Handler가 액세스 클라이언트에 대해 Business Object에서 문자열로의 변환을 수행할 때 서버 액세스 인터페이스 문맥에서의 Data Handler를 보여줍니다.

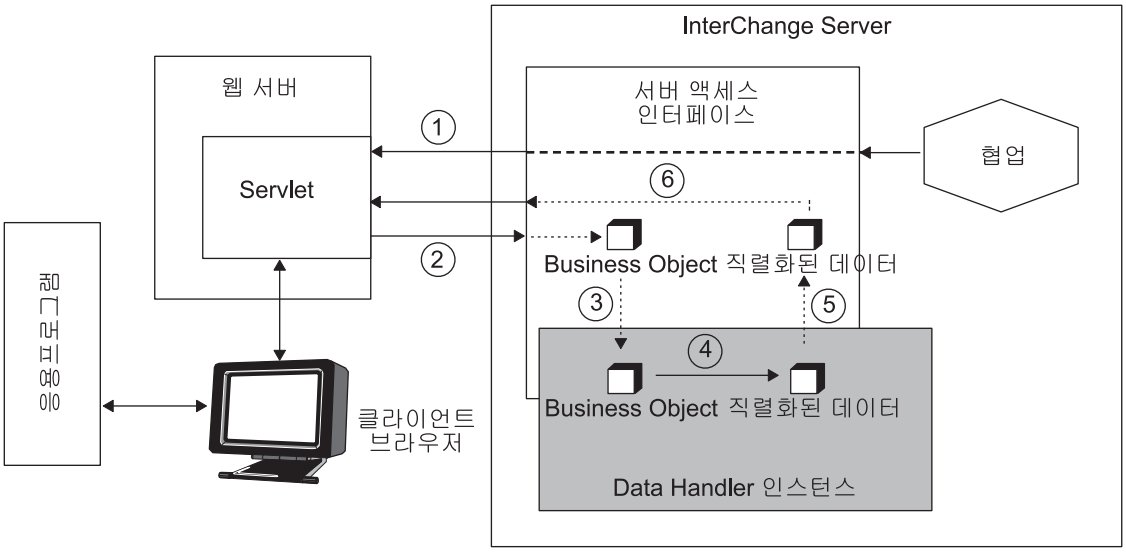

그림 5. 서버 액세스 인터페이스 문맥에서의 Business Object에서 문자열로의 변환

1. 협업은 요청한 데이터와 요청한 조치의 결과를 액세스 클라이언트에 리턴합니다.
2. Business Object를 필요한 형식으로 변환하기 위해, 액세스 클라이언트는 Business Object를 인수로 서버 액세스 인터페이스의 `ItoExternalForm()` 메소드에 전송합니다.
3. 서버 액세스 인터페이스는 다음 조치를 취합니다.
   - Data Handler의 인스턴스를 작성하여 변환을 처리합니다(`DataHandler` 기본 클래스의 `createHandler()` 정적 메소드 사용).
   - 직렬화된 데이터에서 인수로서 다음 data-handler 메소드 중 하나로 전송합니다.
     - `getStreamFromBO()`
     - `getStringFromBO()`
     - `getByteArrayFromBO()`
4. Data Handler는 Business Object를 구문 분석하여 직렬화된 데이터를 작성합니다.
5. Data Handler는 직렬화된 데이터를 서버 액세스 인터페이스에 리턴합니다.
6. 서버 액세스 인터페이스는 직렬화된 데이터를 액세스 클라이언트에 리턴합니다.

서버 액세스 인터페이스의 문자열에서 Business Object로의 변환: 문자열에서 Business Object로의 변환의 경우, Data Handler는 데이터의 스트림 또는 문자열을 수신합니다. Data Handler는 데이터 스트림에 있는 정보를 사용하여 지정한 유형의 Business Object를 작성하고, 이름을 지정한 후 정보로 채웁니다. 문자열에서 Business Object로의 변환은 액세스 클라이언트가 Business Object를 InterChange Server의 협업으로 전송해야 할 경우에 유용합니다. 액세스 클라이언트는 직렬화된 데이터(보통 특정의 MIME 유형 보유)를 Data Handler에 전송합니다.

그림 6에서는 Data Handler가 액세스 클라이언트에 대해 문자열에서 Business Object로의 변환을 수행할 때 서버 액세스 인터페이스 문맥에서의 Data Handler를 보여줍니다.

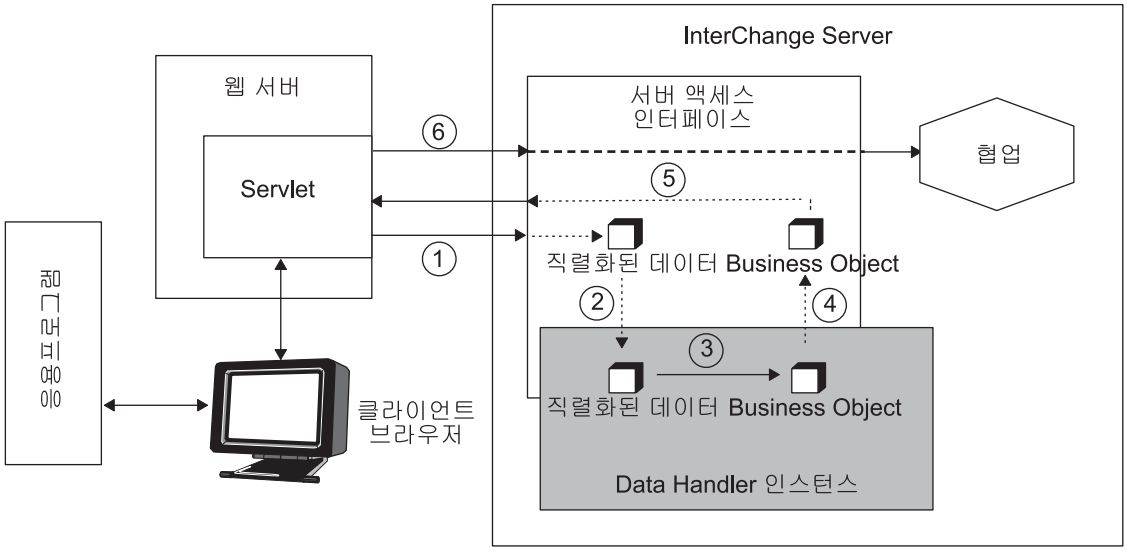

그림 6. 서버 액세스 인터페이스 문맥에서의 문자열에서 Business Object로의 변환

1. 직렬화된 데이터를 Business Object로 변환하기 위해, 액세스 클라이언트는 직렬화된 데이터를 인수로 서버 액세스 인터페이스의 `IcreateBusinessObjectFrom()` 메소드에 전송합니다.
2. 서버 액세스 인터페이스는 다음 조치를 취합니다.
   - Data Handler의 인스턴스를 작성하여 변환을 처리합니다(`DataHandler` 기본 클래스의 `createHandler()` 정적 메소드 사용).
   - 직렬화된 데이터를 Data Handler 인스턴스의 `getBO()` 메소드에 인수로 전송합니다.
3. Data Handler는 Business Object 인스턴스를 빌드합니다.

   Data Handler는 데이터를 구문 분석하고 직렬화된 데이터를 기반으로 Business Object에 대한 속성 값을 채웁니다.
4. Data Handler는 Business Object를 서버 액세스 인터페이스에 리턴합니다.
5. 서버 액세스 인터페이스는 Business Object를 액세스 클라이언트에 리턴합니다.
6. 액세스 클라이언트는 비즈니스 프로세스에서 Business Object 데이터를 사용하여 협업을 호출합니다.

Data Handler 인스턴스화

Data Handler는 커넥터 또는 서버 액세스 인터페이스(액세스 클라이언트가 InterChange Server 통합 브로커와 통신하도록 하기 위해)가 사용하는 클래스 라이브러리로 구현됩니다. `DataHandler` 기본 클래스는 추상 클래스입니다. 그러므로 Data Handler를 인스턴스화하려면 `DataHandler` 서브클래스 중 하나를 인스턴스화해야 합니다. 각 Data Handler(IBM 제공 Data Handler 또는 Custom Data Handler)는 `DataHandler` 기본 클래스의 서브클래스입니다. Data Handler를 인스턴스화하는 메소드는 `createHandler()`입니다.

`createHandler()` 메소드는 *Data Handler Meta Object*에 있는 정보를 사용하여 인스턴스를 작성할 Data Handler와 이 Data Handler를 초기화하는 방법을 결정합니다. Data Handler Meta Object는 임의의 개수의 하위 오브젝트를 포함할 수 있는 계층 구조의 Business Object입니다. Data Handler 구성 정보는 다음의 계층 구조로 배열됩니다.

- *최상위 레벨* Meta Object에는 다른 Data Handler가 지원할 수 있는 MIME 유형에 대한 정보를 포함합니다. 각각의 최상위 레벨 속성은 Data Handler 인스턴스에 대해 하위 Meta Object를 참조하는 카디널리티 1 속성입니다. 각 속성은 하나의 MIME 유형을 나타내고, 속성 유형은 이 MIME 유형을 조작할 수 있는 Data Handler의 하위 Meta Object를 나타냅니다.
- *하위* Meta Object는 특정 Data Handler에 대한 실제 구성 정보를 포함합니다. 각 속성은 구성 등록 정보를 나타내고 기본값, 유형과 같은 정보를 제공합니다.

주: Data Handler는 Meta Object를 사용하여 구성 정보를 보유할 필요가 없습니다. 그러나 IBM이 제공하는 모든 Data Handler는 해당 구성 정보를 위해 Meta Object를 사용하도록 설계되었습니다.

`createHandler()` 메소드는 다음 단계를 수행하여 Data Handler를 인스턴스화합니다.

- 『data-handler 클래스 식별』
- 18 페이지의 『Data Handler 구성 등록 정보 설정』
- 19 페이지의 『Business Object 접두부 설정』

data-handler 클래스 식별

Data Handler를 작성하려면, `DataHandler` 기본 클래스 구현에 대해 인스턴스화해야 합니다. Data Handler 인스턴스화 방법은 인수로서 `createHandler()` 메소드로 전달되는 두 개의 값 중 하나에서 이 data-handler 클래스의 이름을 유추합니다.

- 인스턴스화할 Data Handler의 클래스 이름
- 변환할 데이터의 MIME 유형

클래스 이름 사용

Data Handler 호출자가 클래스 이름을 인수로 전달하면, createHandler() 메소드는 해당되는 클래스 이름의 Data Handler에 대해 인스턴스화합니다. 다음 위치에서 지정된 클래스를 찾습니다.

1. CwDataHandler.jar 파일
2. CwXMLDataHandler.jar 파일
3. CwEDIDataHandler.jar 파일
4. CustDataHandler.jar 파일
5. CLASSPATH에 있는 다른 위치

호출자가 Data Handler의 클래스 이름만 제공하면, createHandler()는 Data Handler Meta Object도 찾지 못하고 오브젝트로부터 구성 등록 정보를 설정하지도 못합니다. 그러므로 이런 방법으로 인스턴스가 작성된 Data Handler는 Meta Object를 필요로 하지 *않습니다*. Custom Data Handler가 Meta Object를 사용해야 하는지 여부에 대한 자세한 정보는 200 페이지의 『Data Handler Meta Object 사용』을 참조하십시오.

MIME 유형 사용

Data Handler 호출자가 클래스 이름을 인수로 전달하지 않으면, createHandler() 메소드는 MIME 유형의 값을 요구합니다. 호출하는 구성요소(커넥터 또는 액세스 클라이언트)가 MIME 유형을 전달할 때, createHandler()가 해당 MIME 유형과 연관된 하위 Data Handler Meta Object를 사용하여 클래스 이름과 Data Handler 인스턴스에 대한 다른 구성 정보를 알아냅니다.

주: Meta Object에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오. Custom Data Handler가 Meta Object를 사용해야 하는지 여부에 대한 자세한 정보는 200 페이지의 『Data Handler Meta Object 사용』을 참조하십시오.

지정된 MIME 유형으로부터 클래스 이름을 유도하기 위해, createHandler() 메소드는 다음 단계를 수행합니다.

1. MIME 유형을 MIME 유형 문자열로 변환합니다.

   createHandler() 메소드는 최상위 레벨 Data Handler Meta Object를 검색할 때 등 영문자가 아닌 모든 문자(예: 하이픈(-), 마침표(.) 또는 슬래시(/))를 밑줄(_)로 변환합니다. 예를 들어, MIME 유형이 text/html일 경우, createHandler()는 유형을 문자열 text_html로 분석합니다.

   createHandler() 메소드가 이러한 영문자가 아닌 문자 변환을 여러 단계에서 수행하므로, 마침표를 포함하는 MIME 유형 이름과 일치하는 이름이 발생할 수 있습니다. 그러나 Business Object Designer는 속성 이름에 마침표를 허용하지 않습니다. 그러므로 IBM은 MIME 유형 이름에 이를 포함시키지 않을 것을 권장합니다.

고유한 MIME 유형/부속 유형 조합을 작성하여 특정 MIME 유형에서의 변화를 표시할 수 있습니다. MIME 유형 이름에서, MIME 유형 및 부속 유형은 영숫자가 아닌 문자(예: 하이픈 또는 밑줄)로 구분합니다. 그러나 createHandler()가 영문자가 아닌 문자를 밑줄로 바꾸기 때문에 IBM은 MIME 유형과 부속 유형을 구분하는 데 밑줄만 사용할 것을 권장합니다. MIME 유형이 text/xml-sgml일 경우, 메소드는 유형을 문자열 text_xml_sgml로 변환합니다.

2. DataHandler 기본 클래스의 정적 등록 정보에서 최상위 레벨 Data Handler Meta Object의 이름을 확보합니다. 이 최상위 레벨 Meta Object에서, createHandler()는 변환할 데이터의 MIME 유형 문자열과 일치하는 속성을 찾습니다.
3. createHandler()가 최상위 레벨 Meta Object에서 일치하는 MIME 유형을 찾으면, createHandler()는 다음 단계를 수행합니다.
   a. 호출자가 Business Object 접두부(선택적인 세 번째 인수) 값을 createHandler()에 제공했으면, 이 값을 MIME 부속 유형으로 해석하고 이를 MIME 유형에 추가하여 다음 양식의 MIME 유형 문자열을 작성합니다.

      MIMETypeString_BOPrefix

      이 이름의 속성이 존재하지 않을 경우, createHandler()는 *오직* MIME 유형과만 일치하는 속성을 찾습니다. 예를 들어, 호출자가 MIME 유형 edi와 Business Object 접두부 x12를 전달할 경우, createHandler()는 최상위 레벨 Meta Object "edi_x12"를 찾습니다. 이 이름의 속성이 존재하지 않을 경우, createHandler()는 "edi" 이름의 속성을 찾습니다.
   b. 사용할 Data Handler 인스턴스에 대한 구성 등록 정보를 포함하는 연관된 하위 Data Handler Meta Object를 확보합니다. (예를 들면, 가능한 구성 등록 정보는 Data Handler 분리문자에 대해 사용할 문자를 식별할 수 있습니다.)
   c. 하위 Meta Object에서 ClassName 속성 값을 검색합니다.
      - ClassName 속성에 값이 있으면, createHandler()는 해당되는 클래스 이름의 Data Handler를 인스턴스화합니다. 연관된 하위 Data Handler Meta Object를 작성하는 것은 Data Handler 구현 프로세스의 일부입니다. 이때, Data Handler를 구현하는 개발자는 클래스 이름을 하위 Meta Object의 ClassName 속성에 추가할 수 있습니다. 이 ClassName 값은 4a단계에 설명된 것처럼 클래스 이름이 기본 클래스 이름과 다른 경우에 필요합니다.
      - 이 ClassName 속성에 값이 *없으면*, createHandler()는 4a단계에 설명된 것처럼 인스턴스화하는 Data Handler의 클래스 이름을 빌드합니다.
4. createHandler() 메소드가 최상위 레벨 Meta Object에서 일치하는 MIME 유형 문자열을 찾지 *못하면*, 다음과 같이 인스턴스화하는 Data Handler의 클래스 이름을 빌드합니다.
   a. MIME 유형 문자열을 기본 Data Handler 패키지에 추가합니다.

      com.crossworlds.DataHandlers

예를 들어, MIME 유형 문자열이 text_html이면 결과 문자열은 다음과 같습니다.

```
com.crossworlds.DataHandlers.text.html
```

b. 메소드가 CLASSPATH에서 생성된 클래스 이름을 찾을 수 있으면, 이 클래스의 인스턴스를 작성합니다. 메소드는 다음 위치에서 이 클래스를 찾습니다.

- CwDataHandler.jar 파일
- CustDataHandler.jar 파일

- CLASSPATH에 있는 다른 위치

클래스 이름 및 MIME 유형 사용

호출자가 클래스 이름과 MIME 유형 모두에 대해 제공될 경우, createHandler()는 다음 조치를 수행합니다.

- 지정한 클래스의 Data Handler를 작성합니다. 이 클래스를 찾기 위해, Data Handler는 16 페이지의 『클래스 이름 사용』에 나열된 디렉토리를 살펴봅니다.
- MIME 유형과 연관된 하위 Meta Object를 사용하여 Data Handler의 구성 옵션을 초기화합니다. createHandler()가 MIME 유형에서 하위 Meta Object를 판별하는 방법에 대한 정보는 16 페이지의 『MIME 유형 사용』을 참조하십시오.

즉, 호출자가 클래스 이름을 제공할 때, 이 클래스 이름은 하위 Meta Object의 ClassName 속성에 지정된 클래스 이름을 *대체합니다*.

Data Handler 구성 등록 정보 설정

IBM에서 제공하는 모든 Data Handler(표 3 및 표 4 참조)는 해당 구성 정보를 위해 Data Handler Meta Object를 사용하도록 설계되었습니다. 다음과 같이 Data Handler Meta Object는 계층 구조 Business Object입니다.

- 최상위 레벨 Data Handler Meta Object에서 각 속성이 MIME 유형에 의해 식별되고 하위 Data Handler Meta Object를 나타냅니다.
- 하위 Data Handler Meta Object에는 MIME 유형과 연관된 Data Handler에 대한 구성 정보가 있습니다.

IBM 제공 Data Handler는 등록 정보를 초기화하기 위해 연관된 하위 Data Handler Meta Object의 구성 정보를 사용합니다. 그러므로 IBM은 제공된 Data Handler 각각에 대해 하위 Meta Object를 제공합니다(표 10 참조).

주: Data Handler Meta Object에 대한 설명은 28 페이지의 『Data Handler 구성』을 참조하십시오.

createHandler() 메소드가 Data Handler를 인스턴스화하면, 특수 보호 메소드인 setupOptions()를 호출하여 적절한 하위 Data Handler Meta Object의 값으로 Data Handler 구성을 초기화합니다.

주: Custom Data Handler에서는 Meta Object를 사용하여 구성을 초기화하지 않아도 됩니다. Data Handler가 연관된 하위 Meta Object를 가지고 있는 경우, `createHandler()` 메소드는 Data Handler에 대해 `setupOptions()`를 호출하지 않습니다. 자세한 정보는 221 페이지의 『기타 Business Object 설정』을 참조하십시오.

Meta Object에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

Business Object 접두부 설정

`createHandler()` 메소드는 세 번째 인수로 선택적 Business Object 접두부를 승인할 수 있습니다. 이 인수를 사용하여 MIME 유형 이름을 판별합니다(16 페이지의 『MIME 유형 사용』 참조). `createHandler()`가 Data Handler를 초기화하고 구성 등록 정보를 설정한 후, 최종 타스크는 Data Handler에서 `BOPrefix` 구성 옵션(존재하는 경우)의 값을 세 번째 인수 값으로 설정하는 것입니다.

주: `BOPrefix` 구성 옵션을 사용하는 Data Handler(예: XML Data Handler)는 이 접두부를 Business Object 이름 앞에 추가합니다.

Data Handler는 작성하는 Business Object의 이름 앞에 이 접두부를 추가할 수 있습니다(문자열에서 Business Object로의 변환을 수행할 때). 접두부와 Business Object 이름 사이에는 밑줄(_)을 넣습니다. 예를 들어, 커넥터는 다음의 `createHandler()` 호출을 사용하여 XML Data Handler를 호출할 수 있습니다.

```
createHandler(null, "text/xml", "UserApp");
```

`createHandler()` 메소드는 XML Data Handler를 인스턴스화하고 `BOPrefix` 속성을 "`UserApp`"로 설정합니다. XML Data Handler가 고객 Business Object를 작성하면, 다음을 수행합니다.

- 직렬화된 데이터로부터 Business Object 이름을 확보합니다.
- "`UserApp`" 접두부를 앞에 추가합니다.

결과로 생성되는 Business Object 이름은 "`UserApp_Customer`"입니다.

Data Handler 호출

Data Handler를 호출하는 문맥에 관계없이, Data Handler는 `createHandler()` 메소드에 의해 인스턴스화됩니다. 각 문맥에서 메소드를 호출하는 방법은 다음과 같습니다.

- 커넥터는 *명시적으로* `createHandler()`를 호출하여 Data Handler의 인스턴스를 작성합니다.

- 액세스 클라이언트는 *내재적*으로 `createHandler()`를 호출하고, 서버 액세스 인터페이스는 액세스 클라이언트가 서버 액세스 인터페이스 메소드인 `ItoExternalForm()` 또는 `IcreateBusinessObjectFrom()` 중 하나를 통해 메소드 호출을 초기화할 때 실제로 `createHandler()`를 호출합니다.

커넥터 문맥에서의 인스턴스화

그림 7에서는 Data Handler를 커넥터 문맥으로 호출할 때 Data Handler 인스턴스화의 예를 보여줍니다.

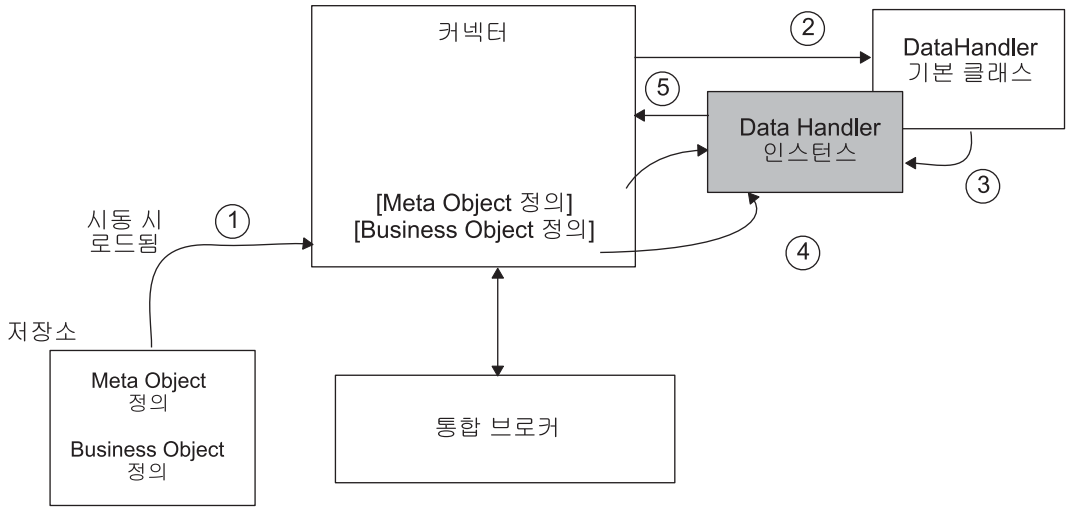

그림 7. 커넥터 문맥에서의 Data Handler 인스턴스화

커넥터 문맥에서 호출되는 Data Handler를 인스턴스화하기 위해, 커넥터는 다음 단계를 수행합니다.

1. Meta Object가 커넥터에 대해 지원되는 오브젝트 목록에 포함되어 있으면, Meta Object는 커넥터가 시작할 때 메모리 내로 읽혀집니다.

   Meta Object는 Business Object와 함께 저장소에 저장됩니다. Business Object 정의와 같이 Meta Object 정의에 액세스하려면 Data Handler용 메모리에 있어야 합니다. 이러한 Meta Object가 커넥터 프로세스의 일부로 메모리에 있으면, Data Handler는 커넥터 문맥으로 호출하여 Meta Object에 액세스할 수 있습니다.

2. 커넥터는 `DataHandler` 기본 클래스에서 `setConfigMOName()` 정적 메소드를 호출하여 Data Handler 기본 클래스의 정적 등록 정보를 Data Handler에 대한 최상위 레벨 Meta Object의 이름으로 설정합니다.

   이러한 최상위 레벨 Meta Object가 커넥터 Meta Object(기본적으로 `MO_DataHandler_Default`)입니다. 최상위 레벨 Meta Object는 커넥터에서 지원하는 오브젝트 목록의 일부여야 합니다.

주: 커넥터가 Data Handler 최상위 레벨 Meta Object의 이름을 확보하는 방법은 커넥터 설계의 일부로 결정됩니다. 자세한 정보는 221 페이지의 『커넥터 구성』을 참조하십시오.

3. 커넥터는 DataHandler 기본 클래스에서 createHandler() 정적 메소드를 호출하여 필요한 데이터 변환을 수행하는 DataHandler 기본 클래스의 인스턴스를 작성합니다. 작성할 클래스의 이름은 다음의 두 방법 중 하나로 판별합니다.
   - 커넥터가 클래스 이름을 인수로 전달하면, createHandler() 메소드는 해당 클래스 이름의 Data Handler를 인스턴스화합니다. 커넥터는 createHandler()를 호출할 때 명시적으로 클래스 이름을 지정할 수 있습니다.
   - 클래스 이름 대신 MIME 유형을 전달하면, createHandler()는 MIME 유형에서 클래스 이름을 유도합니다.

     createHandler() 메소드는 MIME 유형을 MIME 유형 문자열로 변환하고 DataHandler 기본 클래스의 정적 등록 정보에서 Data Handler의 최상위 레벨 Meta Object 이름을 확보합니다. 이 최상위 레벨 Meta Object에서, createHandler()는 Data Handler에 대한 하위 Meta Object 이름을 확보합니다. 이 하위 Meta Object에는 인스턴스화할 클래스의 이름를 포함하여, 구성 정보가 있습니다. 이 유도 방법에 대한 정보는 15 페이지의 『data-handler 클래스 식별』을 참조하십시오.
4. Data Handler는 필요한 데이터 변환을 수행합니다. 커넥터 에이전트는 적절한 DataHandler 메소드를 호출하여 필요한 변환을 수행합니다.
   - 문자열에서 Business Object로의 변환을 위한 getBO() 메소드
   - Business Object에서 문자열로의 변환을 위한 getStringFromBO() 메소드나, Business Object에서 스트림으로의 변환을 위한 getStreamFromBO() 메소드
5. Data Handler는 적절한 형식을 커넥터에 리턴합니다.

서버 액세스 인터페이스 문맥에서의 인스턴스화

그림 8에서는 Data Handler를 서버 액세스 인터페이스 문맥으로 호출할 때 Data Handler 인스턴스화의 예를 보여줍니다.

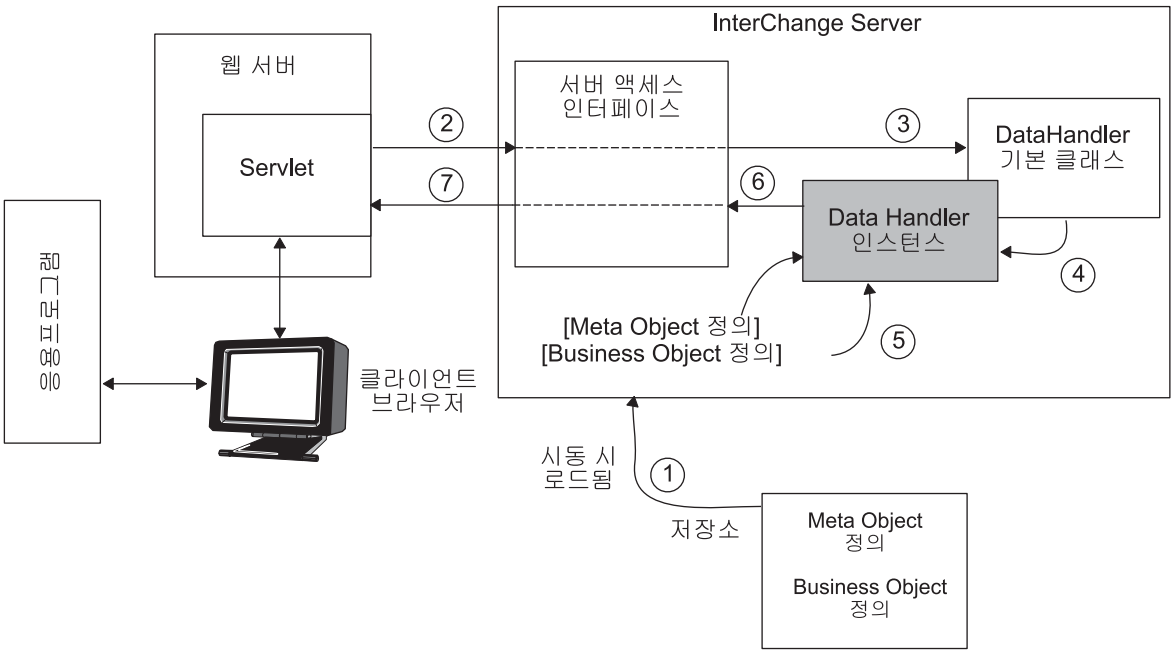

그림 8. 서버 액세스 인터페이스 문맥에서의 Data Handler 인스턴스화

서버 액세스 인터페이스 문맥에서 호출되는 Data Handler를 인스턴스화하기 위해, 서버 액세스 인터페이스는 다음 단계를 수행합니다.

1. 저장소에 있는 다른 모든 Business Object 정의와 함께 서버가 시작될 때, Meta Object는 메모리 내로 읽혀집니다.

   Meta Object는 Business Object와 함께 저장소에 저장됩니다. Business Object 정의와 같이 Meta Object 정의에 액세스하려면 Data Handler용 메모리에 있어야 합니다. 이러한 Meta Object가 InterChange Server(및 서버 액세스 인터페이스) 프로세스의 일부로 메모리에 있으면, Data Handler는 서버 액세스 인터페이스 문맥으로 호출하여 Meta Object에 액세스할 수 있습니다.

2. 액세스 클라이언트는 Data Handler 인스턴스의 작성을 서버 액세스 인터페이스 메소드인 `IcreateBusinessObjectFrom()` 또는 `ItoExternalForm()` 중 하나로 초기화합니다.

   이러한 메소드는 변환할 데이터의 MIME 유형을 전달합니다.

   주: 액세스 클라이언트는 서버 액세스 인터페이스 메소드를 사용하여 Data Handler를 호출해야 합니다. 이러한 메소드는 간접적으로 Data Handler 인터페이스 메소드를 호출하지만, Data Handler 인터페이스 서브세트만 제공합니다. 서버 액세스 인터페이스 메소드에 대한 정보는 *Access Development Guide*를 참조하십시오. Data Handler 인터페이스에 제공되는 메소드에 대한 정보는 225 페이지의 제 11 장 『Data Handler 기본 클래스 메소드』를 참조하십시오.

3. 서버 액세스 인터페이스는 Data Handler의 최상위 레벨 Meta Object 이름을 MO_Server_DataHandler로 설정합니다.
4. 서버 액세스 인터페이스는 DataHandler 서브클래스의 인스턴스를 작성하여 필요한 데이터 변환(DataHandler 기본 클래스의 createHandler() 메소드를 사용하여)을 수행합니다.

   서버 액세스 인터페이스 문맥으로 호출한 경우, createHandler() 메소드는 클래스 이름을 지정하지 *않습니다*. 대신, createHandler() 메소드는 MIME 유형을 MIME 유형 문자열로 변환하고 Data Handler의 최상위 레벨 Meta Object 이름을 확보합니다. 이 최상위 레벨 Meta Object에서, createHandler()는 Data Handler에 대한 하위 Meta Object 이름을 확보합니다. 이 하위 Meta Object에는 인스턴스화할 클래스의 이름를 포함하여, 구성 정보가 있습니다. 이 유도 방법에 대한 정보는 15 페이지의 『data-handler 클래스 식별』을 참조하십시오.
5. Data Handler는 필요한 데이터 변환을 수행합니다. 서버 액세스 인터페이스는 적절한 DataHandler 메소드를 호출하여 필요한 변환을 수행합니다.
   - 문자열에서 Business Object로의 변환을 위한 getBO() 메소드
   - Business Object에서 문자열로의 변환을 위한 getStringFromBO() 메소드나, Business Object에서 스트림으로의 변환을 위한 getStreamFromBO() 메소드
6. Data Handler는 필요한 형식을 서버 액세스 인터페이스에 리턴합니다.
7. 서버 액세스 인터페이스는 요청한 형식을 액세스 클라이언트에 리턴합니다.

메타 데이터 구동 Data Handler 설계

IBM 제공 Data Handler는 메타 데이터에 의해 구동됩니다. 메타 데이터란 Business Object 정의에 저장되는 Business Object에 관한 데이터입니다. Business Object 정의의 메타 데이터는 Business Object 인스턴스에서 데이터를 설명하는 정보를 제공합니다. 일반적으로, Business Object 메타 데이터에는 Business Object의 구조, 속성 특성 설정 및 응용프로그램 특정 정보의 내용이 포함됩니다. 데이터 처리 방법에 대한 지시사항도 제공합니다.

커넥터는 일반적으로 Business Object를 처리할 때 Business Object 메타 데이터를 사용하도록 설계됩니다. 마찬가지로, Data Handler 또한 Business Object 메타 데이터를 사용하도록 설계됩니다. 예를 들면, 다음과 같습니다.

- XML Data Handler에는 각각의 Business Object 정의에 각 속성을 설명하는 응용프로그램 특정 정보가 있어야 합니다. 이 텍스트는 Data Handler가 XML 요소, XML 속성, 처리 명령어 및 기타 유형의 XML 마크업을 식별할 수 있게 합니다.
- FixedWidth Data Handler는 Business Object 속성 등록 정보 MaxLength의 값을 사용하여 고정 폭 문자열을 구문 분석합니다.
- NameValue Data Handler는 Business Object의 이름 및 값을 사용합니다.

메타 데이터 구동 Data Handler는 Data Handler에 하드 코딩된 정보가 아니라 Business Object 정의에 인코딩된 메타 데이터를 기반으로 지원하는 각 Business Object를 처리합니다. 그러므로 Data Handler는 Data Handler 코드에 대한 수정 없이 수정된 Business Object 또는 새 Business Object를 처리할 수 있습니다.

제 2 장 Data Handler 설치 및 구성

이 장에서는 Data Handler를 설치 및 구성하는 방법에 대해 설명합니다. 또한 커넥터가 Data Handler를 지원하도록 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『Data Handler 설치』
- 28 페이지의 『Data Handler 구성』
- 33 페이지의 『Data Handler를 사용할 커넥터 구성』

Data Handler 설치

IBM WebSphere InterChange Server 또는 WebSphere Business Integration Adapters 제품의 파트로 Data Handler를 설치할 수 있습니다. 다음 섹션은 Data Handler를 설치하기 위한 설치 환경 및 단계를 설명합니다.

- 『IBM WebSphere InterChange Server의 Data Handler』
- 26 페이지의 『IBM WebSphere Business Integration Adapters의 Data Handler』
- 26 페이지의 『그래픽 설치 프로그램을 사용한 Data Handler 설치』
- 27 페이지의 『Data Handler 자동 설치』

IBM WebSphere InterChange Server의 Data Handler

IBM WebSphere InterChange Server 제품은 기본 Data Handler 파일, `CwDataHandlers.jar`을 포함합니다. 따라서 이 제품은 5 페이지의 표 3에 나열된 Data Handler를 포함합니다. Server Access Interface 프로세스(InterChange Server 내에서)는 `CwDataHandlers.jar` 파일의 모든 Data Handler에 액세스할 수 있습니다. InterChange Server 설치 프로그램은 자동으로 이 Data Handler 파일을 설치합니다. InterChange Server 설치 프로그램의 사용에 대해서는 *UNIX용 시스템 설치 안내서* 또는 *Windows용 시스템 설치 안내서*의 지시사항을 따르십시오.

주: InterChange Server에 IBM에서 제공하는 추가 Data Handler를 사용하려면 WebSphere Business Integration Adapters 제품을 구입해야 합니다. 이 제품에는 모든 IBM 제공 Data Handler와 Data Handler 샘플 코드가 들어 있어 Custom Data Handler 개발을 도울 수 있습니다.

설치가 완료되면, 표 6에 있는 파일이 시스템의 제품 디렉토리에 설치됩니다.

표 6. WebSphere InterChange Server의 Data Handler에 설치된 파일 구조

디렉토리	설명
DataHandler	IBM 제공 Data Handler의 컴파일된 버전을 위한 CwDataHandler.jar 파일을 포함합니다.
repository\edk	서버 액세스 인터페이스(InterChange Server 통합 브로커와 함께 사용되는) 문맥으로 호출되는 Data Handler에 대한 Meta Object 정의가 있는 텍스트 파일을 포함합니다.

주: 이 장에서 백슬래시(\)는 디렉토리 경로의 규칙으로 사용됩니다. UNIX 설치의 경우, 백슬래시를 슬래시(/)로 대체하십시오. 모든 경로 이름은 WebSphere Business Integration System이 설치된 제품 디렉토리와 관련되어 있습니다.

IBM WebSphere Business Integration Adapters의 Data Handler

IBM WebSphere Business Integration Adapters 제품은 IBM에서 제공하는 모든 Data Handler를 포함합니다(5 페이지의 『IBM 제공 Data Handler』 참조). Data Handler를 설치하기 위해 WebSphere Business Integration Adapters 설치 프로그램을 사용하면 자동으로 설치됩니다.

별도의 XML 또는 EDI Data Handler를 설치하려면, 『그래픽 설치 프로그램을 사용한 Data Handler 설치』 또는 27 페이지의 『Data Handler 자동 설치』의 지시사항을 수행해야 합니다.

설치가 완료되면, 표 7에 있는 파일이 시스템의 제품 디렉토리에 설치됩니다.

표 7. WebSphere Business Integration Adapters의 Data Handler에 설치된 파일 구조

디렉토리	설명
DataHandler	IBM 제공 기본 Data Handler의 컴파일된 버전에 대한 CwDataHandler.jar 파일을 포함합니다. 또한 비어 있는 jar 파일인 CustDataHandler.jar도 포함하며, 이는 개발하는 Custom Data Handler를 보유하기 위한 용도입니다.
DevelopmentKits\edk\DataHandler	Custom Data Handler를 위한 템플리트 파일(StubDataHandler.java)과 Custom Data Handler를 컴파일하기 위한 일괄처리 파일(make_datahandler.bat)을 포함합니다.
DevelopmentKits\edk\DataHandler\Samples	샘플 FixedWidth, NameValue 및 Delimited Data Handler에 대한 소스 코드를 포함합니다.
repository\DataHandlers	커넥터 문맥으로 호출되는 Data Handler에 대한 Meta Object 정의가 있는 텍스트 파일을 포함합니다.

주: 이 장에서 백슬래시(\)는 디렉토리 경로의 규칙으로 사용됩니다. UNIX 설치의 경우, 백슬래시를 슬래시(/)로 대체하십시오. 모든 경로 이름은 WebSphere Business Integration System이 설치된 제품 디렉토리와 관련되어 있습니다.

그래픽 설치 프로그램을 사용한 Data Handler 설치

그래픽 설치 프로그램을 사용하여 별도의 XML 및 EDI Data Handler를 설치하려면 다음을 수행해야 합니다.

1. 사용자의 Data Handler에 대해 설명된 대로 설치 프로그램을 호출하십시오.
2. 언어 선택 프롬프트에서 드롭 다운 메뉴로부터 원하는 언어를 선택한 후 **확인**을 누르십시오.
3. 시작 화면에서 **다음**을 누르십시오.
4. IBM 라이센스 승인 화면에서 **라이센스 계약 조건에 동의합니다**를 누른 다음 **확인**을 누르십시오.
5. 제품 디렉토리 화면을 사용하여 Data Handler를 설치할 위치를 지정할 수 있습니다. 제품 디렉토리에는 호환 가능한 어댑터 프레임워크 버전이 설치되어 있어야 합니다. 또다른 디렉토리를 지정할 수도 있지만, 호환 가능한 어댑터 프레임워크 버전이 설치되어 있어야 합니다.

   Windows 플랫폼에서 필드 값은 기본적으로 CROSSWORLDS 환경 변수에 포함된 값이며, 어댑터 프레임워크의 WebSphere InterChange Server 설치 프로그램 또는 WebSphere Business Integration Adapters 설치 프로그램에 의해 설정됩니다.

   UNIX 플랫폼에서 설치 프로그램은 어댑터 프레임워크 설치 프로그램 또는 WebSphere InterChange Server 설치 프로그램에 의해 작성된 파일의 항목을 조회합니다.

   WebSphere InterChange Server 설치에 관한 자세한 정보는 *Windows용 시스템 설치 안내서*를 참조하십시오.

   제품 디렉토리 화면에서 다음 타스크 중 하나를 수행하십시오.
   - **디렉토리 이름** 필드에 어댑터 프레임워크를 설치할 디렉토리의 전체 경로를 입력한 후 **다음**을 누르십시오.
   - **찾아보기**를 눌러서 디렉토리를 선택한 후 **다음**을 누르십시오.
   - 기본 경로를 승인한 후 **다음**을 누르십시오.
6. 요약 화면은 설치하도록 선택한 기능, 지정된 제품 디렉토리 및 필요한 디스크 공간의 양을 나열합니다. 확인할 정보를 읽은 후 **다음**을 누르십시오.
7. Windows 컴퓨터에서 설치 중인 경우, 설치 프로그램은 일부 Data Handler에 대한 프로그램 폴더 선택 화면을 표시합니다. **프로그램 그룹** 필드에서 어댑터에 대한 바로 가기를 작성하려는 프로그램 그룹의 이름을 입력하거나 기본 프로그램 그룹을 승인한 후 **다음**을 누르십시오.
8. 설치 프로그램이 성공적으로 종료되면, **완료**를 누르십시오.

Data Handler 자동 설치

별도의 XML 및 EDI Data Handler를 자동 설치하려면 다음을 수행해야 합니다.

1. 28 페이지의 표 8에 나열된 원하는 옵션을 사용하여 Data Handler를 설치할 응답 파일을 준비하십시오.

표 8. Data Handler에 대한 자동 설치 옵션

옵션 이름	옵션 값
-W installLocation.active	
-W installLocation.value	InterChange Server가 설치된 디렉토리 경로로 설정하십시오. 이 옵션을 주석 표시된 상태로 둘 경우, 제품이 나열된 기본 디렉토리에 설치됩니다. 이 옵션은 Windows에 설치된 WICS 브로커에 대해서만 해당합니다. WMQI 또는 WAS 브로커에 대해 설치할 경우 또는 UNIX 컴퓨터에 설치할 경우 주석으로 표시하십시오.
-G replaceExistingResponse	설치 프로그램이 복사 중인 파일과 동일한 이름을 갖는 시스템에서 발견된 모든 파일을 바꾸려면 yesToAll 또는 yes로 설정하십시오. 설치 프로그램이 복사 중인 파일과 동일한 이름을 갖는 시스템에서 발견된 모든 파일을 바꾸지 않으려면 noToAll 또는 no로 설정하십시오.
-G replaceNewerResponses	설치 프로그램이 복사 중인 파일보다 새로운 시스템에서 발견된 모든 파일을 바꾸려면 yesToAll 또는 yes로 설정하십시오. 설치 프로그램이 복사 중인 파일보다 새로운 시스템에서 발견된 모든 파일을 바꾸기 않으려면 noToAll 또는 no로 설정하십시오.
-G createDirectoryResponse	아직 존재하지 않는 경우 옵션에서 지정한 제품 디렉토리를 작성하려면 yes로 설정하십시오. 아직 존재하지 않는 경우 제품 디렉토리를 작성하지 않으려면 no로 설정하십시오. 지정한 디렉토리가 존재하지 않을 경우 설치가 성공하려면 이 옵션을 yes로 설정해야 합니다.
-G removeExistingResponse	이 옵션은 어떤 플랫폼의 어떤 브로커에도 관련이 없습니다. 이 옵션을 주석으로 표시하십시오.
-G removeModifiedResponse	이 옵션은 어떤 플랫폼의 어떤 브로커에도 관련이 없습니다. 이 옵션을 주석으로 표시하십시오.

2. 27 페이지의 1단계에서 준비한 응답 파일을 사용하여 설명된 대로 자동 설치를 수행하십시오.

Data Handler 구성

Data Handler Meta Object는 커넥터 또는 서버 액세스 인터페이스 프로세스(InterChange Server가 통합 브로커일 경우)가 입력 파일이나 Business Object 요청에 지정한 MIME 유형 또는 입력 파일의 MIME 유형을 기초로 Data Handler의 인스턴스를 생성할 수 있게 합니다. Data Handler를 구성하려면, Meta Object를 올바르게 초기화하여 호출하는 구성요소(커넥터 또는 액세스 클라이언트)가 사용할 수 있도록 해야 합니다.

주: IBM에서 제공하는 각 Data Handler는 Data Handler Meta Object에 정의된 구성 등록 정보를 사용합니다. 그러나 Custom Data Handler는 구성 등록 정보에 대

해 Meta Object를 사용할 수도, 사용하지 않을 수도 있습니다. 자세한 정보는 200 페이지의 『Data Handler Meta Object 사용』을 참조하십시오.

IBM 제공 Data Handler 지원에서, IBM은 표 9에 나열된 Data Handler Meta Object를 전달합니다.

표 9. IBM 제공 Data Handler Meta Object

Meta Object 레벨	수량	위치
최상위 레벨		
InterChange Server의 경우	하나	repository\edk
커넥터의 경우	하나	repository\DataHandlers
하위	Data Handler마다 하나	repository\DataHandlers

호출자가 사용하도록 하나 이상의 Data Handler 사용을 구성하려면, 다음을 수행해야 합니다.

- 최상위 레벨 Meta Object에서, 호출자에게 지원되는 MIME 유형과 해당되는 연관된 Data Handler를 제공하십시오.
- 하위 Meta Object에서, 호출자에게 원하는 Data Handler 작동에 적절한 구성 정보를 제공하십시오.

최상위 레벨 Meta Object

최상위 레벨 Data Handler Meta Object는 MIME 유형을 하위 Data Handler Meta Object와 연관됩니다. 하위 Meta Object는 항상 인스턴스를 생성할 Data Handler 클래스의 이름을 포함하는 구성 정보를 제공합니다. 그러므로 최상위 레벨 Meta Object는 MIME 유형을 Data Handler와 연관됩니다. 특정의 최상위 레벨 Meta Object에 대한 액세스를 가지는 모든 호출 구성요소가 Meta Object에 표시되는 MIME 유형의 모든 Data Handler를 호출할 수 있습니다.

특정의 최상위 레벨 Meta Object에서 적절한 MIME 유형 속성을 그룹화하고 호출하는 구성요소가 사용해야 하는 Data Handler를 포함하는 Meta Object의 이름을 제공하도록 하여, 호출하는 구성요소가 지원할 수 있는 Data Handler를 제어할 수 있습니다. IBM에서는 다음과 같은 최상위 레벨 Data Handler Meta Object를 제공합니다.

- MO_Server_DataHandler: Data Handler를 호출하는 액세스 클라이언트가 사용할 수 있는 Data Handler를 식별합니다.
- MO_DataHandler_Default: Data Handler를 호출하는 커넥터가 사용할 수 있는 Data Handler를 식별합니다.

MO_Server_DataHandler Meta Object

서버 액세스 인터페이스 프로세스는 MO_Server_DataHandler Meta Object를 사용하여, 프로세스에서 사용할 수 있는 Data Handler를 식별합니다. 제공되는 MO_Server_DataHandler 버전은 MIME 유형을 지원하도록 구성되어 있지 않습니다.

하나의 더미 속성만 포함하고 있습니다. 이 Meta Object를 사용자 정의하면, 현재 InterChange Server에 설치된 모든 Data Handler를 지원할 수 있습니다. 액세스 클라이언트가 MIME 유형을 지원하게 하려면, 최상위 레벨의 MO_Server_DataHandler Meta Object의 더미 속성을 지원되는 MIME 유형의 이름으로 다시 지정하고 해당 MIME 유형에 대해 연관된 하위 Meta Object를 제공하십시오.

예를 들어, text_xml MIME 유형에 대한 지원을 액세스 클라이언트에 제공하려면 dummy 속성의 이름을 text_xml로 바꾸고, 속성 유형으로서 MIME 유형의 연관된 하위 Meta Object의 이름을 제공하십시오. 이 하위 Meta Object는 XML Data Handler를 구성합니다. 그림 9에서는 한 속성인 text_xml이 있는 MO_Server_DataHandler Meta-Object를 보여주며, 이 속성은 MO_DataHandler_DefaultXMLConfig 하위 Meta Object를 나타냅니다.

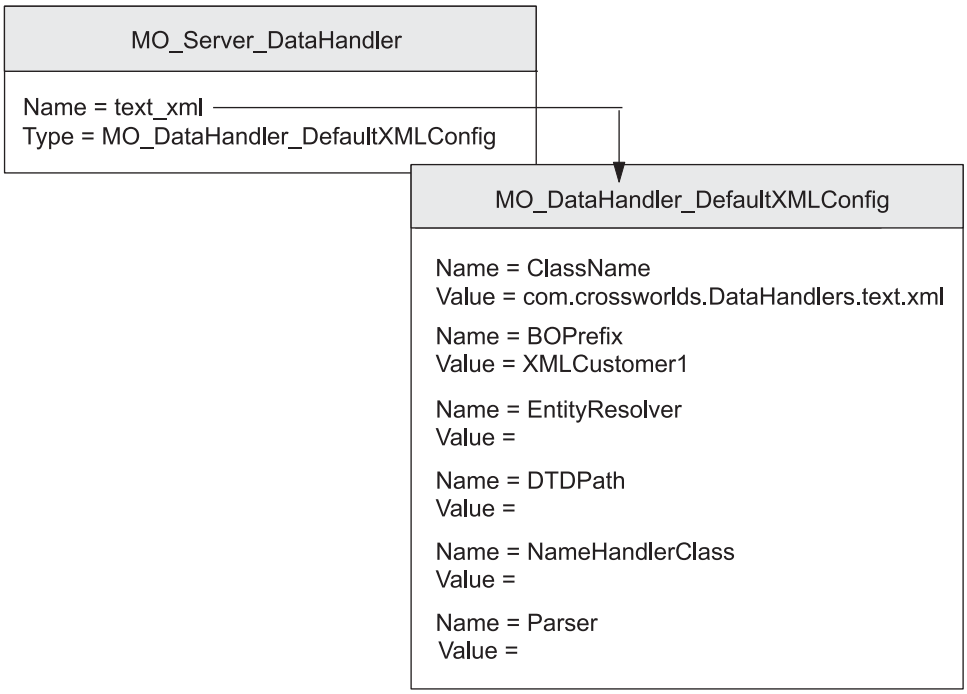

그림 9. MO_Server_DataHandler Meta Object

액세스 클라이언트가 추가 MIME 유형을 지원하도록 하려면, MO_Server_DataHandler Meta Object에서 각 MIME 유형마다 새 속성을 정의하고 해당 MIME 유형에 대해 연관된 하위 Meta Object를 제공하십시오. 둘 이상의 Data Handler를 호출할 경우, Data Handler 인스턴스마다 하위 Meta Object를 정의해야 합니다. 추가 MIME 유형을 위한 지원을 제공하기 위해 다음을 수행할 수 있습니다.

- IBM 제공 Data Handler에 의해 지원되는 MIME 유형 중 아무 것이나 최상위 레벨 Data Handler Meta Object로 추가합니다(5 페이지의 표 3 및 6 페이지의 표 4 참조).

- 사용자가 직접 정의한 MIME 유형과 하위 Meta Object를 지원하는 Data Handler가 있으면, 이를 정의합니다. 자세한 정보는 219 페이지의 『최상위 레벨 Meta Object 수정』을 참조하십시오.

주: Data Handler의 최상위 레벨 서버 Meta Object 이름은 *반드시* `MO_Server_DataHandler`의 기본 이름이어야 하지만, 얼마든지 하위 Meta Object를 포함하도록 최상위 레벨 Meta Object를 구성할 수 있습니다.

MO_DataHandler_Default Meta Object

기본적으로, 커넥터는 `MO_DataHandler_Default` Meta Object를 통해 사용할 수 있는 Data Handler를 식별합니다. `MO_DataHandler_Default`의 제공된 버전은 IBM에서 제공하는 모든 Data Handler(이 문서에서 다루지 않는 일부 어댑터 특정 Data Handler를 포함하여)의 MIME 유형을 지원하도록 구성되어 있습니다.

사용하는 커넥터가 다른 MIME 유형을 지원하도록 구성하려면, 커넥터가 지원하도록 할 각 MIME 유형마다 `MO_DataHandler_Default` Meta Object에 속성이 존재하도록 해야 합니다. 이 속성은 적절한 MIME 유형을 지정하고 해당되는 MIME 유형에 대해 연관된 하위 Meta Object를 표시해야 합니다. 추가 MIME 유형을 위한 지원을 제공하기 위해 사용자가 직접 정의한 MIME 유형과 하위 Meta Object를 지원하는 Data Handler가 있으면, 이를 정의할 수 있습니다. 자세한 정보는 219 페이지의 『최상위 레벨 Meta Object 수정』을 참조하십시오.

주: 특정 커넥터, 또는 커넥터가 처리하기 위해 필요로 하는 특정 유형의 파일이나 특정 Business Object에 해당하도록 커넥터의 최상위 레벨 Meta Object 이름을 변경할 수 있습니다. 그러나 사용하는 모든 오브젝트는 커넥터 정의에 의해 지원되어야 하므로, 다른 최상위 레벨 오브젝트를 사용할 경우 그 오브젝트를 지원하도록 커넥터 정의를 구성해야 합니다. 자세한 정보는 33 페이지의 『Data Handler를 사용할 커넥터 구성』을 참조하십시오.

그림 10에서는 두 개의 Data Handler(XML 및 NameValue)를 정의하는 커넥터의 최상위 레벨 Data Handler Meta Object를 보여줍니다.

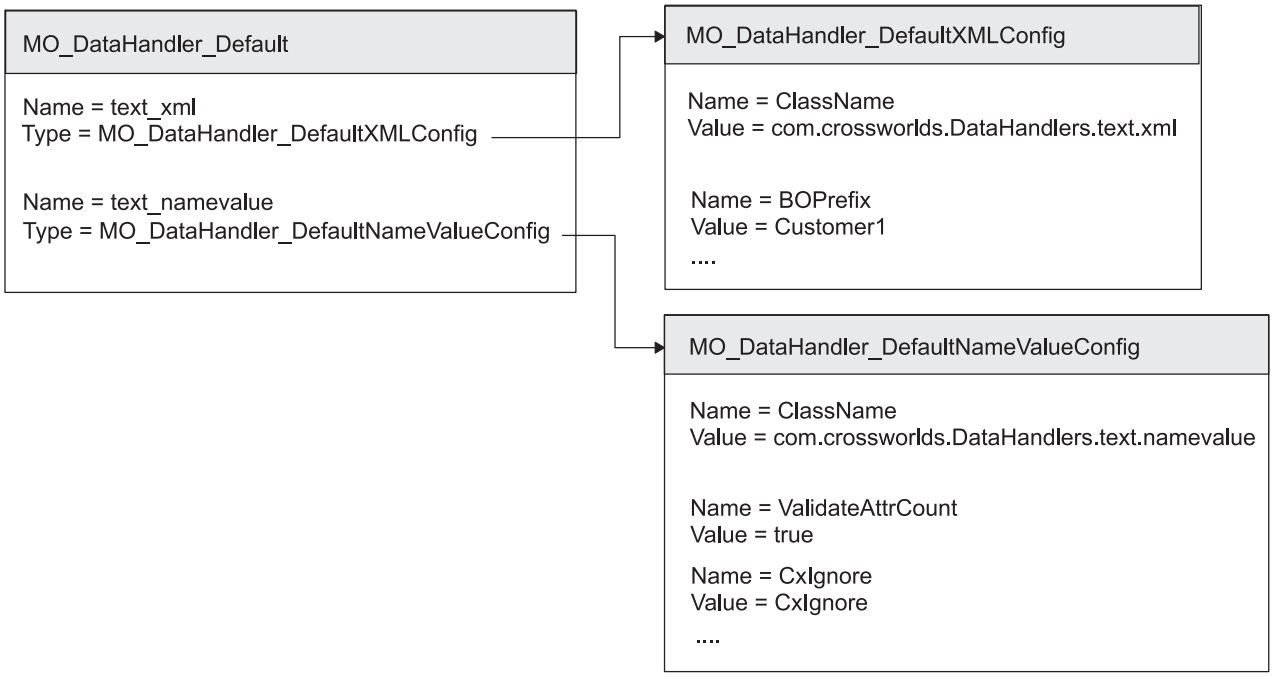

그림 10. 두 개의 다른 Data Handler의 Meta Object 예

주: 커넥터가 Data Handler에 액세스할 수 있으려면, 최상위 레벨 Data Handler Meta Object가 커넥터에 대해 지원되는 오브젝트 목록에 있어야 합니다. 그렇지 않으면, 커넥터는 시작 시 Meta Object를 로드할 수 없습니다.

하위 Meta Object

하위 Data Handler Meta Object는 Data Handler를 초기화하기 위한 구성 정보를 포함하는 플랫 Business Object입니다. 여러 가지의 Data Handler 유형마다 여러 가지의 구성 요구사항이 있으므로, 하위 Meta Object에는 다른 여러 속성이 있습니다. 이러한 구성 정보는 Data Handler 인스턴스의 작동을 사용자에 맞게 정의합니다. 그러므로 하위 Meta Object의 속성 값 세트는 다시 특정의 Data Handler 작동과 연관된 특정 구성을 정의합니다. 특정의 하위 Meta Object에 액세스하는 모든 호출자는 구성 정보가 정의하는 연관된 Data Handler의 작동을 호출합니다.

- 제공된 최상위 레벨 Meta Object에 액세스하는 모든 호출자는 *단 하나의* 특정 Data Handler 작동을 필요로 하며, 하위 Meta Object에서 적절한 구성 정보를 제공하고, 이 하위 Meta Object를 최상위 레벨 Meta Object의 Data Handler MIME 유형과 연관됩니다.

  예를 들면 다음과 같습니다.

  – 모든 커넥터가 특정의 Data Handler 작동에 액세스하도록 하려면, 하위 Meta Object가 커넥터의 최상위 레벨 Meta Object에서 Data Handler의 MIME 유형과 연관되는지 확인하십시오(기본값은 `MO_DataHandler_Default`).

– InterChange Server 통합 브로커에만 해당 -- 특정의 Data Handler 작동에 액세스하도록 하려면, 모든 액세스 클라이언트의 경우 하위 Meta Object가 서버의 최상위 레벨 Meta Object에서 Data Handler의 MIME 유형 (MO_Server_DataHandler)과 연관되어 있어야 합니다.

- 제공된 최상위 레벨 Meta Object에 액세스하는 모든 호출자가 특정 Data Handler에 대해 *두 가지 이상*의 작동을 요구할 경우, 각 Data Handler 작동에 적절한 구성 정보로 하위 Meta Object를 작성하여 각 하위 Meta Object를 고유한 MIME 유형 이름과 연관지으십시오(최상위 레벨 Data Handler Meta Object에서).

  IBM은 고유한 MIME 유형/부속 유형 조합을 사용하여 하위 Meta Object의 이름을 지정할 것을 권장합니다.

  text_*MIMEtype_subtype*

  설명:

  – MIME 유형(*MIMEtype*)은 Data Handler가 지원하는 MIME 유형을 나타냅니다

  – MIME 부속 유형(*subtype*)은 Data Handler의 특정 작동을 나타냅니다.

  예를 들어, 모든 커넥터가 기본 XML Data Handler와 SGML 버전을 모두 지원할 수 있으면, MIME 유형으로 text_xml 및 text_xml_sgml을 작성할 수 있습니다. MIME 유형 이름은 영숫자와 특수 문자인 마침표(.)와 밑줄(_)만 포함할 수 있습니다.

IBM에서는 표 10에서 처럼, 제공하는 Data Handler마다 하위 Data Handler Meta Object를 제공합니다.

표 10. 하위 Data Handler Meta Object

하위 **Meta Object**	자세한 정보
MO_DataHandler_DefaultXMLConfig	47 페이지의『XML Data Handler 구성』
MO_DataHandler_DefaultEDIConfig	107 페이지의『EDI Data Handler 구성』
MO_DataHandler_DefaultFixedWidthConfig	155 페이지의『FixedWidth Data Handler 구성』
MO_DataHandler_DefaultDelimitedConfig	164 페이지의『Delimited Data Handler 구성』
MO_DataHandler_DefaultNameValueConfig	174 페이지의『NameValue Data Handler 구성』
MO_DataHandler_DefaultRequestResponseConfig	146 페이지의『Request-Response Data Handler 구성』

Data Handler를 사용할 커넥터 구성

Data Handler가 커넥터의 문맥으로 실행될 경우, 커넥터가 Data Handler를 사용하도록 구성해야 합니다.

- 커넥터가 Data Handler의 인스턴스를 생성하려면 Data Handler Meta Object에 대한 액세스 권한을 가지고 있어야 합니다.

  Data Handler를 처음 호출하기 전에, 커넥터는 Data Handler 기본 클래스에서 정적 등록 정보를 최상위 레벨 Data Handler Meta Object의 이름으로 설정합니다. 이

최상위 레벨 Meta Object에서, Data Handler는 해당되는 구성 정보를 확보합니다. 계속해서 Data Handler가 인스턴스화될 때마다, Data Handler 인스턴스의 구성 등록 정보가 확보됩니다. Data Handler가 이 구성 정보에 대한 액세스 권한을 가지고 있어야 해당되는 작업을 수행할 수 있습니다.

- Data Handler를 인스턴스화하려면, 커넥터는 Data Handler 클래스의 이름이나 데이터의 MIME 유형을 알고 있어야 합니다.

  커넥터가 `createHandler()`를 호출하여 Data Handler를 호출할 경우, 커넥터는 클래스 이름이나 데이터의 MIME 유형을 전달합니다.

  – 커넥터가 MIME 유형을 전달하면, `createHandler()` 메소드는 이름이 MIME 유형과 일치하는 속성에 대한 최상위 레벨 Data Handler Meta Object를 확인합니다. 일치하는 속성이 발견되면, `createHandler()` 메소드는 MIME 유형과 연관된 하위 Meta Object에서 `ClassName` 속성의 값을 확인합니다.

  – 커넥터가 클래스 이름을 전달하면, `createHandler()` 메소드는 해당 클래스 이름의 Data Handler에 대해 인스턴스화합니다.

  커넥터가 올바른 클래스 이름 또는 MIME 유형을 전달하지 않을 경우, 인스턴스화 프로세스는 실패합니다. 자세한 정보는 15 페이지의 『data-handler 클래스 식별』을 참조하십시오.

커넥터는 다른 방식으로 구성 정보를 확보하도록 구성됩니다. 예를 들면, 다음과 같습니다.

- WebSphere Business Integration Adapter for XML은 최상위 레벨 Meta Object의 이름을 지정하는 `DataHandlerConfigMO` 구성 등록 정보가 있습니다. 이 등록 정보를 채우지 못하면, 커넥터는 Meta Object를 찾을 수 없습니다. 또한 XML 커넥터의 최상위 레벨 Business Object는 Business Object에 있는 데이터의 MIME 유형을 지정하는 `MimeType` 속성을 가지고 있어야 합니다. 커넥터는 `MimeType` 속성 값을 사용하여 적절한 Data Handler를 호출합니다.
- WebSphere Business Integration Adapter for JText 커넥터에는 파일에 대해 각각 클래스의 이름, Data Handler Meta Object 및 MIME 유형을 지정하기 위한 `ClassName`, `DataHandlerConfigMO`, `MimeType` 속성이 있는 고유한 구성 Meta Object가 있습니다.

기타 커넥터는 다른 방식으로 Data Handler 사용을 구성할 수 있습니다. 자세한 정보는 커넥터의 어댑터 안내서를 참조하십시오.

커넥터가 Data Handler 최상위 레벨 Meta Object를 찾을 수 없을 경우, 클래스 이름 또는 MIME 유형을 판별할 수 없어서 Data Handler를 작성할 수 없습니다. 그러므로 Data Handler를 사용하도록 커넥터를 구성할 경우, 다음을 수행해야 합니다.

1. 커넥터의 최상위 레벨 Data Handler Meta Object 이름 구성 방법을 판별합니다. Meta Object 이름의 철자가 올바른지 확인하십시오.

2. MIME 유형 구성 방법을 판별합니다. MIME 유형의 철자가 올바른지 확인하십시오.
3. 최상위 레벨 Data Handler Meta Object가 커넥터에 대해 지원되는 오브젝트 목록에 있는지 확인합니다.
4. Data Handler의 하위 Meta Object에서 Data Handler 클래스 이름 값(`ClassName` 속성에서)을 올바르게 지정했는지 확인합니다.

제 3 장 XML Data Handler

*XML Data Handler*라고 하는 IBM WebSphere Business Integration Data Handler for XML은 Business Object를 XML 문서로 변환하고 XML 문서를 Business Object로 변환합니다. XML Data Handler 설치에 관한 지시사항은 25 페이지의 『Data Handler 설치』의 내용을 참조하십시오.

주: XML Data Handler는 XML 버전 1.0을 지원합니다.

이 장에서는 XML Data Handler가 XML 문서를 처리하는 방법과 XML Data Handler에서 처리할 Business Object를 정의하는 방법에 대해 설명합니다. 또한 XML Data Handler를 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 41 페이지의 『Business Object 정의의 요구사항』
- 47 페이지의 『XML Data Handler 구성』
- 51 페이지의 『DTD를 사용하는 XML 문서』
- 66 페이지의 『스키마 문서를 사용하는 XML 문서』
- 95 페이지의 『Business Object 정의 작성』
- 98 페이지의 『Business Object를 XML 문서로 변환』
- 100 페이지의 『XML 문서에서 Business Object로 변환』
- 102 페이지의 『XML Data Handler 사용자 정의』

개요

XML Data Handler는 1차 역할로서 Business Object를 XML 문서로 변환(반대의 경우 포함)하는 데이터 변환 모듈입니다. XML 문서는 `text/xml` MIME 유형의 직렬화된 데이터입니다. XML Data Handler는 커넥터 및 액세스 클라이언트가 사용할 수 있습니다.

이 개요에서는 XML Data Handler에 대한 다음 정보를 제공합니다.

- 『XML 문서 및 Business Object 처리』
- 38 페이지의 『XML Data Handler 구성요소』

XML 문서 및 Business Object 처리

XML 문서는 구조를 정의하는 데 스키마라는 템플리트를 사용합니다. 38 페이지의 표 11에서는 이 스키마를 정의하기 위한 가장 공통적인 데이터 모델을 보여줍니다.

표 11. XML 데이터 모델

XML 데이터 모델	자세한 정보
DTD(문서 유형 정의) 스키마 문서	51 페이지의 『DTD를 사용하는 XML 문서』 66 페이지의 『스키마 문서를 사용하는 XML 문서』

DTD 또는 스키마 문서가 XML의 구조에 대해 설명하듯이 Business Object 정의는 Business Object의 구조에 대해 설명합니다. XML Data Handler는 Business Object와 XML 문서 사이에서 변환할 때 Business Object 정의를 사용합니다. Business Object 정의의 구조와 해당되는 응용프로그램 특정 정보를 사용하여 변환을 수행하는 방법을 결정합니다. Business Object 정의를 적절하게 구성하면 Data Handler가 Business Object를 XML 문서로, XML 문서를 Business Object로 올바르게 변환할 수 있습니다. XML Data Handler가 XML 문서와 Business Object 사이의 변환을 수행할 수 있으려면, 연관된 Business Object 정의를 찾을 수 있어야 합니다.

XML 문서를 Business Object로 변환하거나 Business Object를 XML 문서로 변환하는 데 XML Data Handler를 사용하려면 다음 단계가 발생해야 합니다.

표 12. XML Data Handler 사용

단계	자세한 정보
1. XML 및 Business Object 구조를 설명하는 Business Object 정의가 존재해야 하고 XML Data Handler가 실행될 때 이것이 사용 가능해야 합니다.	41 페이지의 『Business Object 정의의 요구사항』 64 페이지의 『DTD에서 Business Object 정의 작성』
2. XML Data Handler가 환경을 위해 구성되어야 합니다.	47 페이지의 『XML Data Handler 구성』
3. 다음과 같은 해당 데이터 조작을 수행하기 위해 커넥터(또는 액세스 클라이언트)로부터 XML Data Handler가 호출되어야 합니다.	
a) 데이터 조작: 호출자로부터 Business Object를 수신하고, Business Object를 XML 문서로 변환하며, XML 문서를 호출자에게 전달	98 페이지의 『Business Object를 XML 문서로 변환』
b) 데이터 조작: 호출자로부터 XML 문서를 수신하고, 네임 핸들러, SAX 구문 분석기를 사용하여 Business Object를 빌드. 그런 다음, Business Object 호출자에게 리턴	100 페이지의 『XML 문서에서 Business Object로 변환』

XML Data Handler 구성요소

XML Data Handler는 다음 구성요소를 사용하여 XML 데이터를 Business Object로 변환합니다.

- 네임 핸들러
- XML(SAX) 구문 분석기용 단순 API
- (선택적) XML 문서에 DTD가 있고 엔티티 참조가 포함된 경우, XML Data Handler는 추가 구성요소(Entity Resolver)를 사용하여 참조를 분석합니다.

그림 11에서는 XML Data Handler 구성요소 및 해당 관계를 보여줍니다. 이러한 구성요소에 대해서는 다음 섹션에 설명되어 있습니다.

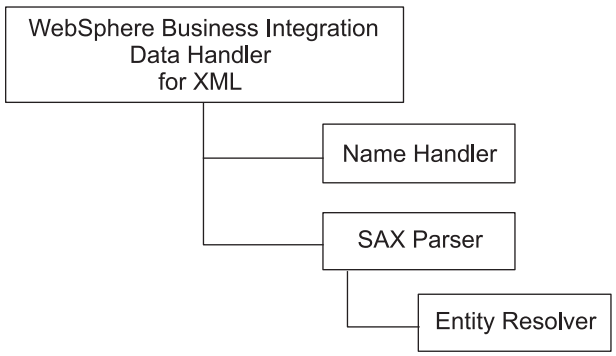

그림 11. XML Data Handler 구성요소

네임 핸들러

XML Data Handler는 네임 핸들러를 사용하여 Business Object의 이름을 XML 메시지로부터 추출합니다. Data Handler는 XML Data Handler 하위 Meta Object의 `NameHandlerClass` 속성 값에 기반으로 하는 네임 핸들러의 인스턴스를 호출합니다.

- 클래스 이름이 `NameHandlerClass` 속성에 제공되면, XML Data Handler는 네임 핸들러를 사용하여 Business Object 이름을 판별합니다.
- 클래스 이름을 제공하지 않으면, Data Handler는 기본 네임 핸들러를 사용하여 Business Object 이름을 판별합니다. 기본 네임 핸들러는 다음과 같이 XML 문서와 `BOPrefix` 등록 정보에 있는 루트 요소의 이름을 사용하여 Business Object 이름을 형성합니다.

  `BOPrefix_rootElement`

사용자 정의 네임 핸들러 작성 방법에 대한 정보는 102 페이지의 『사용자 정의 XML 네임 핸들러 빌드』를 참조하십시오.

SAX Parser

구문 분석기가 XML 하위 Meta Object의 `Parser` 속성의 기본값 등록 정보에 지정되어 있지 않으면, Data Handler는 기본 SAX Parser를 사용합니다.

```
org.apache.xerces.parsers.SAXParser
```

유효성 검증 구문 분석기를 사용하기 위해 다음 단계 중 아무 것이나 수행할 수 있습니다.

- XML 하위 Meta Object에 있는 `Validation` 속성의 기본값 등록 정보를 참으로 설정하십시오. IBM이 이 속성에 제공하는 기본값은 false입니다.

- IBM에서 유효성 검증 SAX Parser를 사용하려면, Parser 속성의 Default Value 등록 정보에서 클래스 이름을 다음으로 변경합니다.

  com.ibm.xml.parsers.ValidatingSaxParser

  Business Object 정의가 DTD를 근거로 할 경우, 로컬 Entity Resolver를 사용하여 DTDPath 속성에 DTD(문서 유형 정의) 또는 스키마 경로에 대한 기본값을 제공하십시오. 반드시 모든 DTD 또는 스키마 파일을 DTDPath 속성에 지정된 위치에 배치하도록 하십시오.

주: 유효성 검증 구문 분석기를 사용할 경우, 올바른 EntityResolver를 사용하고 있는지, 그리고 DTDPath가 올바르게 설정되어 있는지 확인하십시오. 이를 수행하는 방법에 대한 지시사항은 47 페이지의 『XML Data Handler 구성』을 참조하십시오.

대안으로서, IBM의 비유효성 검증 SAX Parser를 사용할 수도 있습니다. 이 구문 분석기를 사용하려면, XML 하위 Meta Object의 Parser 속성의 Default Value 등록 정보를 값 com.ibm.xml.parsers.SAXParser로 설정하십시오.

Entity Resolver

Entity Resolver는 SAX Parser가 XML 데이터의 외부 참조(예: DTD 및 스키마 문서에 대한 참조)의 분석 방법을 지정합니다. XML 문서에 엔티티 참조가 있으면, SAX Parser는 XML Data Handler 구성 Meta Object에 있는 EntityResolver 속성을 사용하여 Entity Resolver의 인스턴스를 호출합니다.

외부 참조는 EntityResolver가 지정하는 Entity Resolver에 따라 다르게 처리됩니다. 표 13에서는 XML Data Handler가 제공하는 Entity Resolver 클래스를 보여줍니다.

표 13. XML Data Handler에 대한 entity-resolver 클래스

entity-resolver 클래스	**설명**
DefaultEntityResolver	이 클래스는 기본 Entity Resolver입니다. 이 Entity Resolver를 호출하면, 모든 외부 참조가 무시됩니다.
LocalEntityResolver	로컬 Entity Resolver는 외부 참조를 로컬 파일 이름으로 처리합니다. 이것의 작동은 유효성 검증에 사용되는 데이터 모델에 따라 다릅니다. • 유효성 검증에 DTD를 사용하는 경우, systemID가 file:// 또는 http://로 시작하고 DTDPath 속성이 설정되어 있으면, 로컬 Entity Resolver가 systemID의 경로를 DTDPath Meta Object 속성 값으로 대체합니다. 외부 참조는 systemID가 경로 이름이 아니거나 DTDPath 속성을 설정하지 않은 경우 무시됩니다. • 유효성 검증에 스키마 문서를 사용하는 경우, 경로가 file:// 또는 http://로 시작되거나 DOS 파일 이름(예: "D:\xmlschemas\test")을 포함하면, 로컬 Entity Resolver가 schemaLocation 또는 noNamespaceSchemaLocation 속성이 지정하는 경로를 DTDPath Meta Object 속성 값으로 대체합니다.

표 13. XML Data Handler에 대한 entity-resolver 클래스 (계속)

entity-resolver 클래스	**설명**
URIEntityResolver	이 Entity Resolver는 외부 참조를 로컬 파일 이름 또는 다운로드 가능한 URL로 처리합니다. 동적으로 외부 참조를 다음 중 하나의 경우로 해석합니다. • DTD를 유효성 검증에 사용하는 경우: DOCTYPE이 http:// 또는 file://로 시작하는 SYSTEM 값을 포함하는 경우 • 유효성 검증에 스키마 문서를 사용하는 경우: schemaLocation 또는 noNamespaceSchemaLocation 속성이 http:// 또는 file://로 시작하는 경우 그러면, Entity Resolver가 HTTP 연결을 열고 지정한 웹 사이트에서 DTD 또는 스키마 문서를 다운로드합니다. **경고:** XML Data Handler는 DTD 또는 스키마 문서를 캐시하지 *않습니다*. Data Handler가 URIEntityResolver 클래스를 Entity Resolver로 사용할 경우, XML 문서를 구문 분석할 *때마다* HTTP 연결을 엽니다. 그러므로 네트워크 트래픽은 XML Data Handler의 성능에 영향을 줄 수 있습니다.

주: 표 13의 모든 Entity Resolver는 다음의 클래스 접두부를 가져야 합니다.

```
com.crossworlds.DataHandlers.xml
```

사용자의 XML 문서가 스키마 문서를 사용하는 경우, 스키마 문서가 포함하는 외부 스키마도 외부 엔티티로 취급됩니다. 따라서, SAX Parser는 Entity Resolver를 호출하여 포함된 스키마 문서를 해석합니다. XML 문서가 schemaLocation 또는 noNamespaceSchemaLocation을 사용하여 스키마 위치를 지정하면, 외부 스키마 문서(포함되거나 가져온)의 유효성 검증을 위해 EntityResolver 속성을 LocalEntityResolver 또는 URIEntityResolver로 설정할 수 있습니다.

외부 엔티티를 찾기 위한 또다른 방법을 지정해야 할 경우, 사용자 정의 Entity Resolver를 작성해야 합니다. 사용자 정의 Entity Resolver 작성에 대한 정보는 104 페이지의 『사용자 정의 Entity Resolver 빌드』를 참조하십시오.

Business Object 정의의 요구사항

Business Object 정의가 XML Data Handler의 요구사항을 따르도록 하려면, 다음을 포함하는 이 섹션의 지침을 사용하십시오.

- 42 페이지의 『Business Object 구조』
- 43 페이지의 『Business Object 속성 등록 정보』
- 46 페이지의 『응용프로그램 특정 정보』
- 47 페이지의 『Business Object verb』

Business Object 정의를 적절하게 구성하면 Data Handler가 Business Object를 XML 문서로, XML 문서를 Business Object로 올바르게 변환할 수 있습니다. XML Data Handler의 Business Object 작성 방법에 대한 정보는 64 페이지의 『DTD에서 Business Object 정의 작성』을 참조하십시오.

Business Object 구조

DTD 또는 스키마 문서를 나타내려면 다음과 같이 최소한 두 개 이상의 Business Object 정의가 필요합니다.

- *최상위 레벨* Business Object는 DTD 또는 스키마 문서를 정의하는 정보를 나타내며 다음과 같은 내용을 포함해야 합니다.
  - XML 버전을 나타내는 `XMLDeclaration` 속성

    이 속성은 응용프로그램 특정 정보에 `type=pi` 태그가 있어야 합니다.
  - DTD 또는 스키마 문서에 루트 요소를 나타내는 속성

    이 속성의 유형은 단일 카디널리티 Business Object여야 합니다. 이 유형은 DTD 또는 스키마 문서의 루트 요소에 대한 Business Object 정의입니다. XML ODA는 이 루트 요소의 이름을 `Root` ODA 구성 등록 정보에서 가져옵니다. 응용프로그램 특정 정보는 `elem_name` 태그로 이 요소의 이름을 나열해야 합니다.

    주: `elem_name` 태그는 Business Object의 응용프로그램 특정 정보에서 XML 요소의 이름만을 필요로하는 이전 구문을 바꿉니다. XML Data Handler는 계속 기존 Business Object 정의와의 역방향 호환성에 대한 이전 구문을 지원합니다. 그러나 XML ODA는 Business Object 정의 생성 시 새 구문을 사용합니다.
- *root-element* Business Object 정의는 XML 정의 문서의 루트 요소를 나타냅니다. `Root` 구성 등록 정보를 통해 루트 요소로 고려해야 할 요소를 XML ODA에 알려줄 수 있습니다. 이 등록 정보에는 루트 요소에 각 XML 구성요소의 속성이 들어 있습니다.

DTD 또는 스키마 문서의 Business Object 정의를 사용하여 XML Data Handler가 처리하는 Business Object는 다음 추가 규칙을 따라야 합니다.

- XML 문서의 모든 태그에는 Business Object에 연관된 속성을 가지고 있어야 합니다. Business Object 정의는 Business Object 속성의 유형을 제공하고 해당 속성에 대한 응용프로그램 특정 정보가 저장됩니다. 이 정보는 XML 요소의 구조 및 내용에 의해 판별됩니다.
- XML 요소의 Business Object 정의에서 XML 속성을 나타내는 모든 속성은 다른 속성보다 *앞에* 일어나야 합니다. XML Data Handler는 제공된 XML 요소의 속성이 Business Object 정의의 *첫 번째* 속성인 것으로 가정합니다.

주: Business Object에는 XML Data Handler가 올바른 XML 문서를 작성할 수 있도록 충분한 데이터가 있어야 합니다. 데이터 없이 Data Handler Business Object를 전송하지 않도록 하십시오.

이 문서는 DTD 및 스키마 문서에 대한 Business Object 정의의 구조에 대한 다음 정보를 제공합니다.

데이터 모델	자세한 정보
문서 유형 정의(DTD) 스키마 문서	51 페이지의 『DTD의 Business Object 구조』 66 페이지의 『스키마 문서의 Business Object 구조』

Business Object 속성 등록 정보

Business Object 정의는 속성을 정의합니다. 각 속성은 속성에 대한 정보를 제공하는 다양한 등록 정보를 가집니다. 이 섹션에서는 여러 등록 정보에 대한 XML Data Handler의 해석 방법 및 Business Object 정의 수정 시 해당 등록 정보의 설정 방법에 대해 각각 설명합니다.

Name 속성 등록 정보

각 Business Object 속성에는 고유한 이름이 있어야 합니다. XML 요소 또는 XML 속성 이름은 항상 elem_name 또는 attr_name 태그에 지정됩니다. 이 경우, 속성 응용프로그램 특정 정보의 elem_name(또는 attr_name) 태그에 지정된 XML 요소(또는 속성)의 이름이 특수 문자를 포함합니다. 그러나 Business Object 속성(이러한 특수 문자를 허용하지 않음)에서는 특수 문자를 생략합니다.

Type 속성 등록 정보

각 Business Object 속성은 다음과 같은 유형(예: Integer, String 또는 포함된 하위 Business Object의 유형)을 가지고 있어야 합니다.

- DTD의 경우: 하위 요소 또는 하나 이상의 FIXED가 아닌 속성을 포함하는 XML 요소는 Business Object로 처리됩니다. 단일 카디널리티를 통해 상위 오브젝트에 XML 요소가 포함된 경우, PCDATA 값만 있는 XML 요소는 속성으로 처리됩니다. 다중 카디널리티를 통해 포함되는 경우, XML 요소는 Business Object 정의가 다중 카디널리티 스칼라 값(예: 문자열 값의 배열)을 지원하지 않으므로 Business Object로 표시됩니다.
- 스키마 문서의 경우: 각 Business Object 속성은 문자열 유형 또는 포함된 하위 Business Object의 유형을 가지고 있어야 합니다. 하위 요소 또는 복합 유형을 가지는 XML 요소는 Business Object로 처리됩니다. 단일 카디널리티를 통해 상위 오브젝트에 XML 요소가 포함된 경우 단순 유형의 값만 있는 XML 요소는 Business Object 오브젝트 속성으로 처리됩니다. 다중 카디널리티를 통해 포함되는 경우, XML

요소는 Business Object 정의가 다중 카디널리티 스칼라 값(예: 문자열 값의 배열)을 지원하지 않으므로 Business Object로 표시됩니다.

주: 모든 단순 속성의 유형은 `String`이어야 합니다.

Key 및 Foreign Key 속성 등록 정보

각 Business Object는 Key 등록 정보를 속성에 대해 true로 설정하여 지정하는 최소한 하나 이상의 1차 키 속성을 가지고 있어야 합니다. Foreign Key 등록 정보는 선택적이고 XML 문서의 구조에 따라 다릅니다. 이 섹션에서는 Key 및 Foreign Key 속성 등록 정보에 대해 다음 정보를 제공합니다.

- 『Business Object 정의에서 키 지정』
- 『키 및 "필수" 핸들링』

Business Object 정의에서 키 지정: XML Business Object 정의 생성 도구(예: XMLBorgen, Edifecs SpecBuilder 및 XML ODA)의 이전 버전에서 생성 도구는 상위 XML Business Object의 키로 ObjectEventId 속성을 지정했습니다. 그러나 이 릴리스에서 Business Object Designer는 키로 지정된 ObjectEventId 속성을 갖는 Business Object 정의를 더 이상 저장하지 않습니다.

이러한 제한 때문에 XML ODA의 현재 버전은 이제 다음 조치를 수행합니다.

- 각 하위 Business Object에서 첫 번째 속성을 키로 설정합니다.
- 상위 Business Object에서 키 속성을 설정하지 *않습니다*.

XML ODA가 생성하는 상위 Business Object 정의에 키를 제공하려면 Business Object Designer에서 Business Object 정의를 제공하고 Business Object 정의를 분석하여 키로 지정할 해당 속성을 판별해야 합니다. Business Object Designer에서 Business Object 정의를 저장하기 *전에* Business Object 정의의 키 속성을 변경해야 합니다.

주: XML ODA는 이전 XML Business Object 정의 생성 도구(예: XMLBorgen 및 Edifecs SpecBuilder)를 바꿉니다. 그러므로 XML ODA만이 이러한 특수 단계를 수행하여 상위 Business Object의 키 속성으로 ObjectEventId의 지정을 피합니다. 이전 Business Object 정의 생성 도구에서 생성한 기존 XML Business Object 정의가 있는 경우(XML ODA의 이전 버전 포함), 이 Business Object 정의는 여전히 ObjectEventId를 키로 사용할 수 있습니다. Business Object를 현재 릴리스로 이주 중인 경우 이 Business Object 정의를 분석해야 합니다. Business Object 정의에서 적절한 키 속성을 설정하는 데 실패하면 이벤트 순차 기능의 성능에 부정적인 영향을 미칠 수 있습니다.

키 및 "필수" 핸들링: 이 문서는 키와 "필수" 사이의 관계에 대해 다음 정보를 제공합니다.

데이터 모델	자세한 정보
문서 유형 정의(DTD)	53 페이지의 『DTD의 Business Object 속성 등록 정보』
스키마 문서	75 페이지의 『스키마 문서의 Business Object 속성 등록 정보』

필수 속성 등록 정보

단일 카디낼리티 하위 Business Object를 포함하는 속성에 대해 등록 정보를 지정할 경우, XML Data Handler는 상위 Business Object가 이 속성에 대한 하위 Business Object를 포함하도록 요구합니다. 카디낼리티, 키 및 외부 키 속성 등록 정보를 설정하면, 속성의 Required 등록 정보를 설정하는 것에 영향을 줄 수 있습니다.

이 문서는 ″필수″에 대해 다음 정보를 제공합니다.

데이터 모델	자세한 정보
DTD(문서 유형 정의)	53 페이지의 『DTD의 Business Object 속성 등록 정보』
스키마 문서	75 페이지의 『스키마 문서의 Business Object 속성 등록 정보』

Cardinality 속성 등록 정보

Cardinality 등록 정보는 그 유형으로 Business Object 정의를 가지는 속성에서 허용되는 하위 Business Object의 수를 나타냅니다. 이 등록 정보의 설정은 XML 문서 및 요소의 구조에 따라 다릅니다. 설정은 속성이 필수여야 하는지에도 영향을 줍니다 (Required 등록 정보가 true로 설정).

이 문서는 카디낼리티와 ″필수″ 사이의 관계에 대해 다음 정보를 제공합니다.

데이터 모델	자세한 정보
DTD(문서 유형 정의)	53 페이지의 『DTD의 Business Object 속성 등록 정보』
스키마 문서	

특수 속성 값

Business Object 속성은 유형이 속성의 Type 등록 정보와 일치하는 값을 갖습니다. 또한, 속성은 다음 두 가지 특수 값 중 아무 것이나 가질 수 있습니다.

- CxIgnore

  XML Data Handler가 통합 브로커에서 Business Object를 수신할 경우, CxIgnore 값을 갖는 모든 속성을 무시합니다. 이는 해당되는 속성이 Data Handler에 표시되지 않는 것처럼 보입니다. 그러므로 Data Handler는 해당 XML 요소를 생성하지 않습니다. 즉, 이 속성에 대해, XML 태그(비어있는 태그 조차도)를 생성하지 않습니

다. XML Data Handler가 Business Object 속성에 해당하는 XML 태그가 없는 XML 입력을 수신하면,Data Handler는 속성에 값 CxIgnore를 지정합니다.

- CxBlank

  XML Data Handler가 통합 브로커에서 Business Object를 수신하면, 속성의 Type 등록 정보에 따라 CxBlank 속성 값을 처리합니다.

  – 복합 속성인 경우(속성의 Type 등록 정보를 다른 Business Object 정의의 이름으로 설정한 경우), Data Handler는 복합 속성에 CxBlank 값이 없다고 가정합니다.

  – 단순 속성인 경우(속성의 Type 등록 정보를 `String` 데이터 유형으로 설정한 경우), Data Handler는 XML 문서에 빈 태그를 작성합니다. DTD를 기본으로 하는 XML 문서의 경우, 빈 큰 따옴표(″ ″)는 CxBlank에 해당하는 PCDATA 값으로 사용됩니다.

XML Data Handler가 빈 태그가 있는 XML 입력을 수신하면, Data Handler는 값 CxBlank를 해당하는 Business Object 속성에 지정합니다.

응용프로그램 특정 정보

Business Object 정의에서 응용프로그램 특정 정보는 Data Handler에 Business Object를 XML 문서로 변환하는 방법에 대한 명령어를 제공합니다. 응용프로그램 특정 정보는 Data Handler가 Business Object를 올바르게 처리할 수 있게 합니다. 그러므로 XML Data Handler에 대해 새 Business Object를 작성하거나 기존의 Business Object를 수정할 경우, Business Object 정의에 있는 응용프로그램 특정 정보가 Data Handler에서 예상하는 구문과 일치하는지 확인하십시오. XML Data Handler는 아래와 같은 종류의 응용프로그램 특정 정보를 사용할 수 있습니다.

- Business Object 레벨 응용프로그램 특정 정보는 Business Object 정의에 대한 정보를 전체적으로 제공합니다.
- 속성 레벨 응용프로그램 특정 정보는 특정 속성에 대한 정보를 제공합니다.

주: XML Data Handler는 XML 문서의 요소와 Business Object의 속성을 일치시키기 위해 응용프로그램 특정 정보를 사용합니다. 응용프로그램 특정 정보의 최대 길이는 255자입니다. 응용프로그램 특정 정보의 값이 255자를 초과하면, DTD 또는 스키마 문서를 재구성한 후 Business Object를 다시 생성해야 합니다.

이 문서는 응용프로그램 특정 정보에 대해 다음 정보를 제공합니다.

데이터 모델	자세한 정보
DTD(문서 유형 정의)	54 페이지의 『DTD의 XML 구성요소에 대한 응용프로그램 특정 정보』
스키마 문서	76 페이지의 『스키마 문서의 XML 구성요소에 대한 응용프로그램 특정 정보』

Business Object verb

Business Object를 XML 문서로 변환할 경우, XML Data Handler는 verb에 대한 XML을 생성하지 *않으며*, XML 문서를 Business Object로 변환할 때 verb를 설정하지도 않습니다. 그러나 verb 정보는 다음 방법 중 하나로 보존할 수 있습니다.

- DTD 또는 스키마 문서에서 verb에 대한 요소를 작성하고 verb에 대한 Business Object 속성을 작성할 수 있습니다. verb를 Business Object 속성으로 복사하도록 비즈니스 통합 시스템의 내용을 설계할 수 있습니다. 그러면 Data Handler가 verb를 XML 요소로 변환하여, XML 문서에서 verb를 보존합니다. XML 문서가 리턴될 때, 비즈니스 통합 시스템은 Business Object 속성의 값에 따라 verb를 설정할 수 있습니다.
- 특정 Business Object 및 verb 조합에 대해 DTD 또는 스키마 문서를 작성하고 각 Business Object 요청을 해당 Business Object 및 verb에 대한 DTD 또는 스키마 문서와 연관될 수 있습니다. XML 문서가 Business Object로 변환되어 호출자에게 리턴된 경우, 커넥터는 DTD 또는 스키마 문서에 해당하는 verb를 설정할 수 있습니다.
- verb에 대한 정보와 함께 호출 커넥터를 제공할 수 있으면, 통합 브로커에 전송하기 전에 Business Object에서 verb를 설정할 수 있습니다.

XML Data Handler 구성

XML Data Handler를 구성하려면, XML Data Handler의 하위 Meta Object에서 구성 정보가 제공되는지 확인해야 합니다.

주: XML Data Handler를 구성하려면, Business Object 정의가 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정해야 합니다. 자세한 정보는 41 페이지의 『Business Object 정의의 요구사항』을 참조하십시오.

XML Data Handler의 경우, IBM은 기본 하위 Meta Object `MO_DataHandler_DefaultXMLConfig`를 제공합니다. 이 Meta Object에 있는 각각의 속성이 XML Data Handler의 구성 등록 정보를 정의합니다. 표 14에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 14. XML Data Handler에 대한 하위 Meta Object 속성

속성 이름	설명	제공되는 기본값
`BOPrefix`	Business Object 이름을 빌드하기 위해 기본 `NameHandler` 클래스에서 사용하는 접두부. 연관된 Business Object 정의의 이름과 일치하도록 기본값을 변경해야 합니다. 속성 값은 대소문자가 구분됩니다.	`XMLTEST`
`ClassName`	지정된 MIME 유형으로 사용하기 위해 로드할 data-handler 클래스의 이름. 최상위 레벨 Data Handler Meta Object에는 이름이 지정된 MIME 유형과 일치하고 유형이 XML 하위 Meta Object인 속성이 있습니다(표 14 참조).	`com.crossworlds.DataHandlers.text.xml`

표 14. XML Data Handler에 대한 하위 Meta Object 속성 (계속)

속성 이름	설명	제공되는 기본값
DefaultEscapeBehavior	속성 값이 특수 문자를 가지면, XML Data Handler가 이스케이프 처리를 수행해야 합니다. 속성의 응용프로그램 특정 정보가 이스케이프 태그를 포함하지 않으면, XML Data Handler가 이스케이프 처리를 수행할지를 판별하기 위해 DefaultEscapeBehavior 등록 정보의 값을 확인합니다. 자세한 정보는 62 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』를 참조하십시오.	true
DTDPath	DTD(문서 유형 정의) 또는 XSD(스키마)에 대한 경로를 구성하기 위해 Data Handler에서 사용됩니다.	없음
DummyKey	키 속성. Data Handler에서 사용되지는 않지만 비즈니스 통합 시스템에 필요합니다.	1
EntityResolver	외부 엔티티(예: DTD 또는 스키마)에 대한 참조를 처리하기 위해 사용할 클래스의 이름. 이 속성의 값에 대한 자세한 정보는 40 페이지의 『Entity Resolver』를 참조하십시오.	없음
IgnoreUndefinedAttributes	속성이 false로 설정된 경우, XML Data Handler는 Business Object 정의에 대해 모든 XML 속성의 유효성을 검증하고 정의되지 않은 속성을 발견하면 예외를 발행합니다. 이 속성이 true로 설정되면, XML Data Handler가 정의되지 않은 모든 XML 속성을 무시하고 경고를 생성합니다.	true
IgnoreUndefinedElements	이 속성이 false로 설정된 경우, XML Data Handler는 Business Object 정의에 대해 모든 XML 요소의 유효성을 검증하고 응용프로그램 특정 정보에서 정의되지 않은 요소를 발견하면 예외를 발행합니다. 이 속성이 true로 설정되면, XML Data Handler가 정의되지 않은 모든 XML 요소(및 이 정의되지 않은 요소 내의 모든 속성)를 무시하고 경고를 생성합니다.	false
InitialBufferSize	Business Object를 XML로 변환할 때 사용하는 버퍼의 초기 크기를 정의합니다. 이 값을 XML Business Object 크기(바이트)로 설정하십시오. 이 값을 큰 수로 설정하면 직렬화된 XML로의 Business Object 변환 속도가 빨라집니다.	2MB(2,097,152KB)
NameHandlerClass	XML 문서 내용에서 Business Object 이름을 판별하기 위해 사용할 클래스의 이름. 고유의 사용자 정의 네임 핸들러를 작성할 경우, 이 속성의 기본값을 변경하십시오. 자세한 정보는 102 페이지의 『사용자 정의 XML 네임 핸들러 빌드』를 참조하십시오.	com.crossworlds.DataHandlers.xml.TopElementNameHandler
Parser	XML 문서에 대한 SAX 준수 구문 분석기의 패키지 이름	없음
UseNewLine	출력 XML의 각 태그의 줄을 바꾸려면 true로 설정하십시오. (XML Data Handler는 줄 바꾸기 및 캐리지 리턴 양식에서 여분의 내용을 XML 문서에 추가합니다.) XML 출력을 변경하지 않으려면 false로 설정하십시오.	false

표 14. XML Data Handler에 대한 하위 Meta Object 속성 (계속)

속성 이름	설명	제공되는 기본값
Validation	이 값은 유효성 검증 구문 분석기가 사용됨을 지정하기 위해 Data Handler에서 사용합니다. Data Handler는 xerces 구문 분석기의 기능 http://xml.org/sax/features/validation 을 true로 설정하여 이를 수행합니다. 유효성 검증 구문 분석기를 사용하려면, 기본값을 true로 변경해야 합니다. Validation이 true로 설정되면, XML 구문 분석기가 다음에 대해 XML 문서의 유효성을 검증합니다. • DTD가 있으면, DTD • 스키마 문서가 지정되어 있으면, 스키마 문서. 이런 경우, XML 구문 분석기가 전체 스키마 확인을 수행합니다. • DTD와 스키마 문서가 둘다 지정되어 있으면, 둘다	false
ObjectEventId	Data Handler에서는 사용되지 않지만 비즈니스 통합 시스템에서 필요로 하는 위치 표시기	없음

표 14에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 해당 속성의 Default Value 등록 정보를 적절한 값으로 설정해야 합니다. 최소한 ClassName 및 BOPrefix 속성이 기본값을 가지고 있는지 확인해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

각 조합에서 동일한 XML Data Handler 구성을 사용할 경우, 상이한 MIME 유형/부속 유형 조합에서 단일 하위 Meta Object를 사용할 수 있습니다. 커넥터에서 MIME 유형마다 다른 XML Data Handler 구성을 요구할 경우, 각 Data Handler 인스턴스마다 별도의 하위 Meta Object를 작성해야 합니다. XML Data Handler의 여러 구성을 준비하려면, 다음 단계를 수행하십시오.

- 기본 XML 하위 Meta Object를 복사하고 이름을 바꾸십시오. 새 하위 Meta Object의 이름을 지정하기 위한 권장 방법은 MIME 유형에 부속 유형을 제공하는 것입니다. 예를 들어, 기본 XML Data Handler와 SGML 버전 모두에 대해 지원하기 위해, 기본 XML 하위 Meta Object를 복사하고 이 사본에 이름 MO_DataHandler_DefaultSGMLConfig를 지정할 수 있습니다.
- 각 XML 하위 Meta Object에서 속성의 기본값을 설정하여 Data Handler 인스턴스를 구성하십시오.

  각 MIME 유형/부속 유형 조합에 대한 최상위 레벨 Data Handler Meta Object에서 속성을 작성하십시오. 예를 들어, XML 및 SGML 지원에서 MIME 유형 text_xml 및 text_xml_sgml을 작성할 수 있습니다. 이러한 속성 각각은 연관된 하위 Meta Object를 나타냅니다.

동일한 Data Handler의 복수 인스턴스를 지원하도록 XML Data Handler를 구성할 수도 있습니다. 이러한 경우, 이름이 text_xml_*subtype*으로 지정된 또다른 최상위 레벨 속성을 작성할 수 있습니다. 여기서 *subtype*은 text_xml_AppA에서처럼 응용프로그램 이름, 응용프로그램 엔티티 이름 또는 또다른 적절한 이름이 될 수 있습니다.

Data Handler 구성 방법에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

그림 12에서는 이름이 최상위 레벨 Meta Object와 해당되는 하위 Meta Object의 예를 보여줍니다. 최상위 레벨 Meta Object MO_DataHandler_XMLSample에는 네 개의 속성이 있지만, 하위 Meta Object는 세 개만 있습니다. 이는 속성 Application_xml_AppC가 적절한 Data Handler를 호출하기 위해 속성 text_xml_AppB와 동일한 하위 Meta Object를 사용하기 때문입니다.

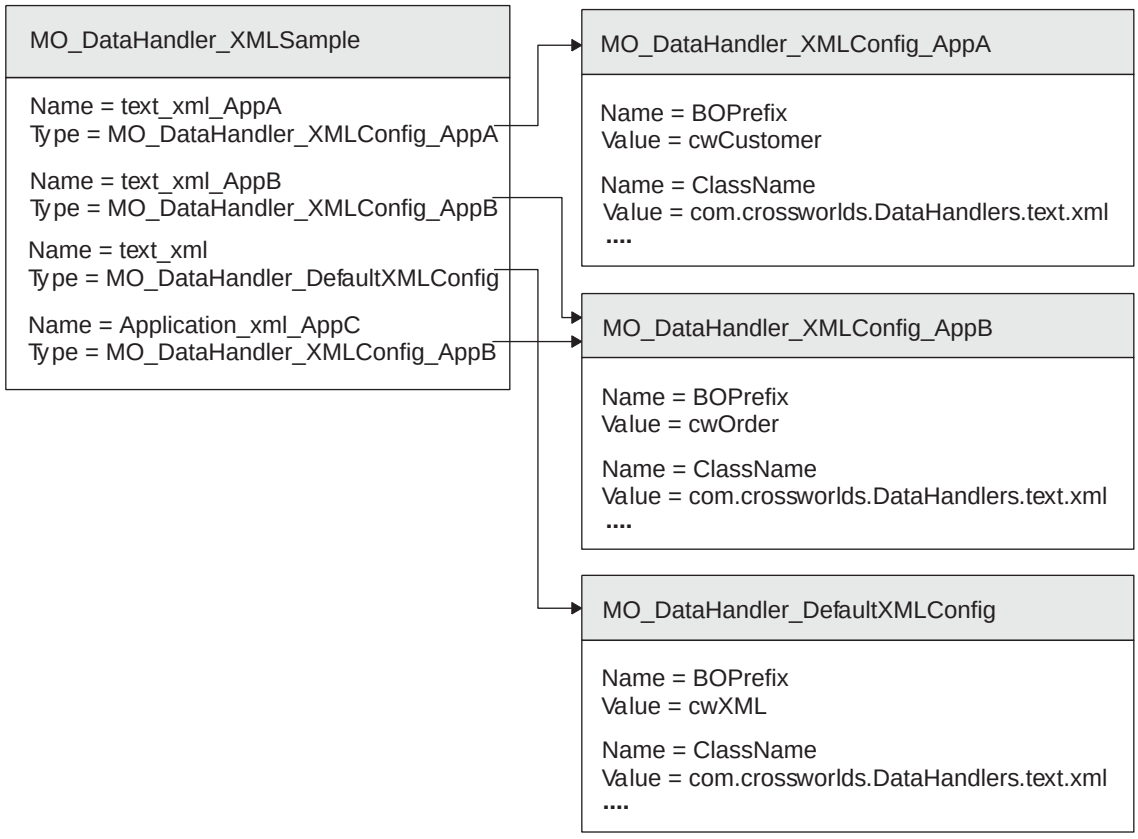

그림 12. 복수 XML Data Handler에 대한 Meta Object 예

DTD를 사용하는 XML 문서

DTD(*문서 유형 데이터*)는 스키마라는 XML 문서의 템플리트를 설명하기 위해 특수 구문을 제공하는 XML 문서용 데이터 모델입니다. 이 DTD는 `.dtd` 확장자를 가지는 파일입니다. XML 문서의 스키마를 나타내는 Business Object 정의는 문서의 구조를 보존하고 기록하는 데 DTD의 정보를 사용합니다. 이 섹션에서는 DTD에서 Business Object 정의의 구조 정보를 알아내는 것에 대해 다음 정보를 제공합니다.

- 『DTD를 기본으로 하는 Business Object 정의 요구사항』
- 64 페이지의 『DTD에서 Business Object 정의 작성』

DTD를 기본으로 하는 Business Object 정의 요구사항

DTD에 상응하는 Business Object 정의가 XML Data Handler의 요구사항을 따르도록 하려면, 다음을 포함하는 이 섹션의 지침을 따르십시오.

- 『DTD의 Business Object 구조』
- 53 페이지의 『DTD의 Business Object 속성 등록 정보』
- 54 페이지의 『DTD의 XML 구성요소에 대한 응용프로그램 특정 정보』

Business Object 정의를 적절하게 구성하면 Data Handler가 Business Object를 XML 문서로, XML 문서를 Business Object로 올바르게 변환할 수 있습니다. XML Data Handler의 Business Object 작성 방법에 대한 정보는 66 페이지의 『스키마 문서를 사용하는 XML 문서』를 참조하십시오.

DTD의 Business Object 구조

DTD를 나타내려면 42 페이지의 『Business Object 구조』에 설명되어 있는 최소한 두 개 이상의 Business Object 정의가 필요합니다. DTD의 경우, 이 Business Object 정의에는 다음과 같은 추가 요구사항이 있습니다.

- *최상위 레벨* Business Object는 XML DTD를 나타내며 다음을 포함할 수 있습니다.
  - `DOCTYPE` 선언을 나타내는 `DocType` 속성

    XML ODA가 최상위 레벨 Business Object 정의에 `DocType` 속성을 생성하는지 여부는 해당 `DocTypeOrSchemaLocation` 구성 등록 정보의 설정에 따라 다릅니다. 자세한 정보는 63 페이지의 『XML DOCTYPE 선언의 경우』 및 65 페이지의 『지원되는 DTD 구조』를 참조하십시오.
  - XML 버전을 나타내는 `XMLDeclaration` 속성

    이 속성은 응용프로그램 특정 정보에 `type=xmlns` 태그가 있어야 합니다. 자세한 정보는 91 페이지의 『XML 처리 명령어의 경우』를 참조하십시오.
  - DTD에 루트 요소를 나타내는 속성

42 페이지의 『Business Object 구조』에 설명되어 있는 대로, 이 속성의 유형은 단일 카디널리티 Business Object여야 합니다. 이 유형은 루트 요소에 대한 Business Object 정의입니다. 응용프로그램 특정 정보는 elem_name 태그로 이 요소의 이름을 나열해야 합니다. 자세한 정보는 85 페이지의 『XML 요소의 경우』를 참조하십시오.

- *root-element* Business Object 정의는 DTD의 루트 요소를 나타냅니다.

DTD를 기반으로 하는 Business Object 정의를 사용하여 XML Data Handler가 처리하는 Business Object는 다음 규칙을 따라야 합니다.

- XML 문서의 모든 태그에는 Business Object 정의에 연관된 속성이 있습니다. 이 규칙의 예외는 FIXED 속성의 경우입니다. 기본적으로 FIXED 속성은 정적인 데이터를 가지므로 Business Object 정의에 포함되지 *않습니다*. 그러나 FIXED 속성을 Business Object 정의에 포함시키려면, 수동으로 Business Object 정의에 속성을 추가할 수 있습니다.

주: 일반적인 Business Object 요구사항 목록은 41 페이지의 『Business Object 정의의 요구사항』을 참조하십시오.

다음은 XML 문서의 DTD 예입니다. DTD의 이름은 Order이며, 응용프로그램 Order 엔티티에 해당하는 요소가 있습니다.

```
<!--Order -->
<!-- Element Declarations -->
<!ELEMENT Order (Unit+)>
<!ELEMENT Unit (PartNumber?, Quantity, Price, Accessory*)>
<!ELEMENT PartNumber (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessory (Quantity, Type)>
<!ATTLIST Accessory
     Name CDATA >
<!ELEMENT Type (#PCDATA)>
```

그림 13에서는 Order DTD와 연관된 XML 문서에 해당하도록 작성할 수 있는 Business Object의 구조를 보여줍니다. Order DTD의 각 XML 요소 및 요소 속성에는 해당 Business Object 속성이 있습니다. 최상위 레벨 Business Object에는 XML 선언 DOCTYPE과 최상위 레벨 Order 요소에 대한 속성이 있습니다. 또한 요소 속성 이름은 Accessory Business Object에서 첫 번째 속성입니다.

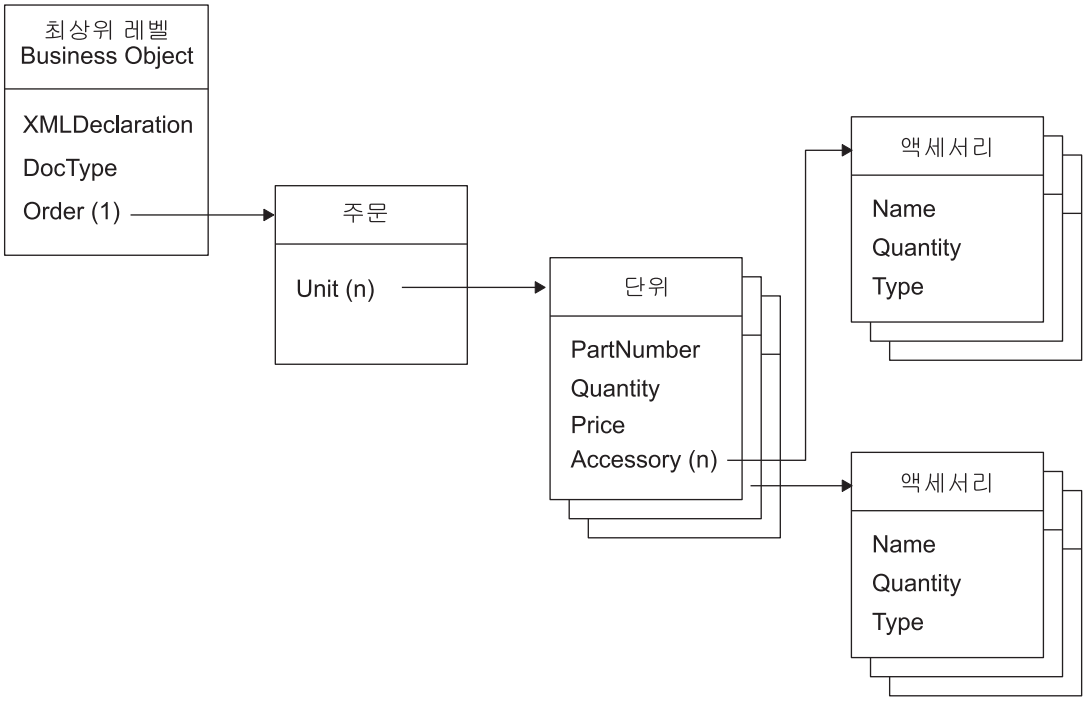

그림 13. Order DTD를 사용하는 XML 문서의 Business Object 예

DTD의 Business Object 속성 등록 정보

XML 문서의 Business Object 정의가 DTD를 기본으로 하면, Business Object 속성이 43 페이지의 『Business Object 속성 등록 정보』에서 논의된 제한사항을 가집니다. 또한, DTD 구문이 Business Object 속성의 "필수"를 판별할 수 있습니다. "필수"는 XML Data Handler가 속성을 필요로 하는지를 판별하는, 카디널리티와 속성이 키인지 여부를 포함하는 요소의 조합입니다. 속성이 필수이면, 필수 속성 등록 정보가 true로 설정되어야 합니다.

필수 속성 등록 정보의 설정은 다음과 같이 XML 요소 및 속성 스펙뿐만 아니라 카디널리티, 키 및 외부 키 속성 등록 정보의 설정에 따라 다릅니다.

- Business Object 속성의 카디널리티는 DTD의 ELEMENT 단편에 의해 판별됩니다. 이 카디널리티는 속성이 필수인지에 영향을 줍니다. 표 15에서는 카디널리티와 DTD에 있는 요소 선언의 가능한 조합에 대해 대략적으로 요약합니다.

표 15. DTD의 카디널리티 및 "필수"

DTD ELEMENT 단편	카디널리티	필수
지정 안됨	1	예
?	1	아니오
+	N	예
*	N	아니오

- Business Object 속성이 1차 키인지 foreign key인지는 DTD의 ATTLIST 단편에 의해 판별됩니다. 키가 있는지가 속성이 필수인지에 영향을 줍니다. 표 16에서는 ATTLIST 단편의 구문이 DTD에 있는 속성 선언의 가능한 조합에 대한 Business Object 속성의 "필수"에 영향을 주는 방법을 대략적으로 요약합니다.

표 16. DTD의 키 및 "필수"

DTD ATTLIST 단편	Key	필수	주석
#IMPLIED	아니오	아니오	
#REQUIRED	아니오	예	
ID {#IMPLIED | #REQUIRED}	예	아니오	#IMPLIED, #REQUIRED 무시
IDREF {#IMPLIED, #REQUIRED}	Foreign key	#IMPLIED 또는 #REQUIRED 지정 여부에 따라 결정됩니다.	
NMTOKEN {#IMPLIED | #REQUIRED}	예	아니오	#IMPLIED, #REQUIRED 무시

DTD의 XML 구성요소에 대한 응용프로그램 특정 정보

이 섹션에서는 DTD를 기본으로 하는 Business Object 정의의 응용프로그램 특정 정보 형식에 대한 다음 정보를 제공합니다.

- 『Business-object 레벨 응용프로그램 특정 정보』
- 58 페이지의 『배열 속성 응용프로그램 특정 정보』
- 58 페이지의 『속성 응용프로그램 특정 정보』

Business-object 레벨 응용프로그램 특정 정보: XML Data Handler는 다음 유형의 Business Object를 사용하여 DTD에서 생성된 여러 종류의 XML 요소를 나타냅니다.

- 『DTD를 기반으로 하는 일반 Business Object 정의』
- 55 페이지의 『DTD를 기반으로 하는 혼합 Business Object 정의』
- 55 페이지의 『DTD를 기반으로 하는 랩퍼 Business Object 정의』

이러한 유형의 Business Object는 Business Object 레벨에서 응용프로그램 특정 정보에 의해 구별됩니다.

DTD를 기반으로 하는 일반 Business Object 정의: *일반* Business Object는 XML 요소를 나타냅니다. 이 유형의 Business Object에서, Business Object 레벨에 있는 응용프로그램 특정 정보는 Business Object가 참조하는 XML 요소 이름을 식별합니다. 예를 들어, XML 요소를 다음과 같이 정의되어 있다고 가정하십시오.

```
<!ELEMENT Unit(...)>
```

연관된 Business Object 정의에 대한 Business Object 레벨에서의 응용프로그램 특정 정보는 다음과 같습니다.

```
[BusinessObjectDefinition]
Name = MyApp_Unit
AppSpecificInfo = elem_name=Unit
[Attribute]
...
```

DTD를 기반으로 하는 혼합 Business Object 정의: *혼합* Business Object는 문자 데이터(#PCDATA) 및 기타 하위 요소의 혼합 내용을 포함하는, 혼합 XML 요소를 나타냅니다. 혼합 유형 XML 요소의 DTD 표시는 다음과 같습니다.

```
<!ELEMENT (#PCDATA | CONTAINED_ELEMENT1 | CONTAINED_ELEMENTN)*>
```

예를 들어, 다음과 같이 DTD에 Cust XML 요소가 정의되어 있다고 가정하십시오.

```
<!ELEMENT Cust(#PCDATA | Address | Phone)*>
```

혼합 유형 XML 요소를 나타내려면 혼합 유형 Business Object 정의를 사용하십시오. 혼합 Business Object 정의에서는 그 Business Object 레벨 응용프로그램 특정 정보가 다음으로 되어 있습니다.

- 혼합 XML 요소의 이름
- type=MIXED 태그

Business Object 정의에서는 Business Object 레벨에서의 응용프로그램 특정 정보인 Cust 요소를 나타내는 MyApp_Cust가 다음과 같습니다.

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo = Cust;type=MIXED;
```

DTD를 기반으로 하는 랩퍼 Business Object 정의: 랩퍼 Business Object는 반복 선택사항 목록을 나타냅니다. 이 유형의 Business Object 정의는 XML 요소에 임의의 순서 및 임의의 카디널리티로 나타날 수 있는 하위가 있는 경우 필요합니다. 랩퍼 Business Object는 XML 문서에서 하위 요소의 순서 및 카디널리티를 보존합니다.

선택사항 목록 XML 요소에서는 DTD 정의가 다음과 같습니다.

```
(CONTAINEDELEMENT1 | ... | CONTAINEDELEMENTN)*
```

예를 들어, DTD의 선택사항 목록 XML 요소 정의는 다음과 같을 수 있습니다.

```
<!ELEMENT CUST( U | I | B )* )>
```

이 요소에는 아무 순서로나 표시될 수 있는 선택적인 세 가지의 부속 요소가 있습니다. 각 부속 요소는 단순 요소입니다. 56 페이지의 그림 14에서는 이 유형의 XML 문서를 보여줍니다.

```
<CUST>
   <U>.....
   </U>
   <B>.....
\   </B>
   <I>.....
   </I>
   <B>.....
\   </B>
   <U>.....
   </U>
...
```

그림 14. 반복 선택사항 목록의 XML 문서 내용

DTD에 정의된 선택사항 목록 XML 요소를 나타내기 위해 Business Object 정의는 계층 구조입니다. 다음 Business Object 정의를 포함합니다.

- 상위 Business Object 정의

  이 상위 Business Object 정의에는 다중 카디널리티 Business Object 배열을 나타내는 하나의 속성이 있습니다. 이 속성은 유형으로서 연관된 랩퍼 Business Object의 Business Object 정의를 가집니다. 상위 Business Object 정의에는 다음 응용프로그램 특정 정보가 있습니다.

  – Business Object 레벨에서 상위 Business Object 정의는 응용프로그램 특정 정보에 선택사항 목록 요소의 이름을 포함합니다.

  – 속성 레벨에서 다중 카디널리티 속성(상위 Business Object 정의에 있는)은 선택적인 선택사항 요소를 다음 형식으로 지정합니다.

    (*choiceElement1*|...|*choiceElementN*)

    여기서 *choiceElement1...choiceElementN*은 정의된 선택사항 요소 각각에 해당됩니다. 파이프(|) 문자로 선택사항 요소 각각을 구분해야 하고 전체 태그는 괄호로 묶어야 합니다.

- 랩퍼 Business Object 정의

  랩퍼 Business Object 정의에는 선택사항 목록 요소에 정의된 선택사항 요소 각각에 대한 하나의 속성이 있습니다. Business Object 레벨에서 응용프로그램 특정 정보를 필요로 하지 *않습니다.*

57 페이지의 그림 15에서는 선택사항 목록 XML 요소에 대한 Business Object 정의의 계층 구조 그림을 보여줍니다. 런타임에서 각각의 하위 Business Object는 랩퍼 Business Object의 인스턴스이고 데이터로 채워진 단 하나의 속성을 가집니다. 예로서, 그림 14에 있는 XML 내용의 Business Object는 5개의 하위를 가지며, 각각은 해당 속성으로 채워져 있습니다.

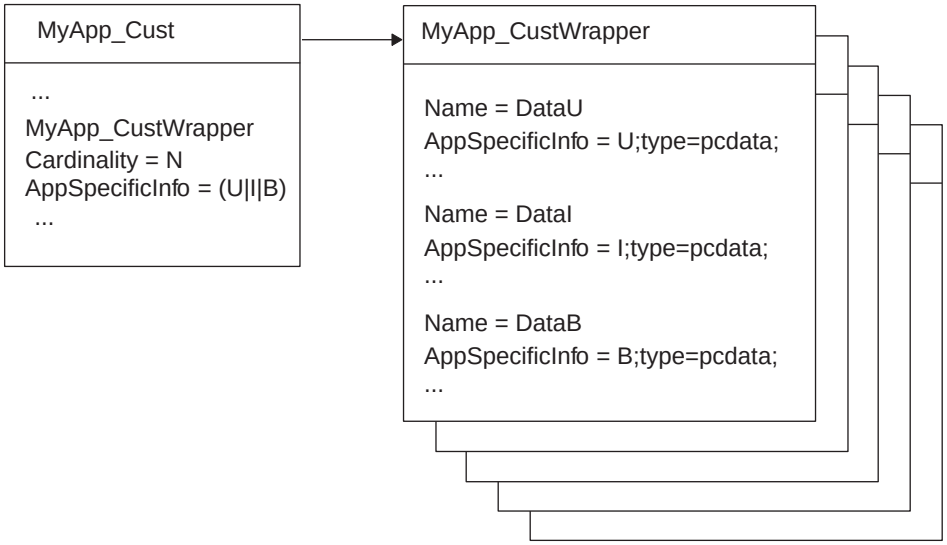

그림 15. 선택사항 목록 XML 요소의 계층 구조 Business Object 정의

그림 16에서는 상위 Business Object 정의인 MyApp_Cust를 응용프로그램 특정 정보와 함께 보여줍니다.

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo =

[Attribute]
Name = CustWrapper
Type = MyApp_CustWrapper
Cardinality = N
   AppSpecificInfo = attr_name=CustWrapper;(U|I|B)
[End]
```

그림 16. 선택사항 목록 요소의 상위 Business Object 정의

랩퍼 Business Object 정의인 MyApp_CustWrapper는 각 선택사항 요소마다 하나씩, 세 개의 속성을 가집니다. 각 선택사항 요소에 문자 데이터가 있기 때문에 각 속성의 응용프로그램 특정 정보가 다음을 지정합니다.

- 요소의 이름
- type=pcdata 태그

주: 문자 데이터의 속성에 대한 자세한 정보는 60 페이지의 『PCDATA만 있는 XML 요소의 경우』를 참조하십시오.

그림 17에서는 이 XML 문서의 랩퍼 Business Object 정의를 보여줍니다.

```
[BusinessObjectDefinition]
Name = MyApp_CustWrapper
AppSpecificInfo =

[Attribute]
Name = DataU
Type = String
   AppSpecificInfo = attr_name=U;type=pcdata;
[End]

[Attribute]
Name = DataI
Type = String
   AppSpecificInfo = attr_name=I;type=pcdata;
[End]

[Attribute]
Name = DataB
Type = String
   AppSpecificInfo = attr_name=B;type=pcdata;
[End]
```

그림 17. 선택사항 목록 요소의 랩퍼 Business Object 정의

배열 속성 응용프로그램 특정 정보: Business Object 속성이 다른 요소를 포함하는 XML 요소를 표시할 경우, 응용프로그램 특정 정보에는 요소의 이름이 포함되어야 합니다. 예를 들어, DeliveryDate 속성이 Business Object 유형을 가지며 DATETIME 요소를 표시할 경우, 응용프로그램 특정 정보에는 요소의 이름이 포함됩니다.

```
Name = DeliveryDate
Relationship = Containment
Cardinality = n
AppSpecificInfo = DATETIME
```

속성 응용프로그램 특정 정보: Business Object 정의의 속성은 다음 XML 구성요소를 나타낼 수 있습니다.

- 59 페이지의 『XML 요소의 경우』
- 60 페이지의 『PCDATA만 있는 XML 요소의 경우』
- 60 페이지의 『XML 속성의 경우』
- 61 페이지의 『문자 데이터 및 속성이 있는 XML 요소의 경우』
- 62 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』
- 63 페이지의 『XML DOCTYPE 선언의 경우』
- 63 페이지의 『CDATA 섹션의 경우』
- 63 페이지의 『XML 주석의 경우』
- 64 페이지의 『XML 처리 명령어의 경우』

표 27에서는 이들 다른 XML 구성요소의 속성 레벨 응용프로그램 특정 정보에 대한 태그 및 이 태그에 대해 자세히 설명하는 이 매뉴얼의 절을 함께 보여줍니다.

표 17. 속성 응용프로그램 특정 정보에 대한 태그

Business Object 속성의 표시	**응용프로그램 특정 정보**	**자세한 정보**
XML 요소	elem_name=*XML 요소 이름*	『XML 요소의 경우』
PCDATA만 있는 XML 요소	elem_name=*name of XML element*;type=pcdata	60 페이지의 『PCDATA만 있는 XML 요소의 경우』
XML 요소의 속성	attr_name=*XML 속성의 이름* type=attribute	60 페이지의 『XML 속성의 경우』
문자 데이터 및 속성이 있는 XML 요소	type=pcdata;notag	61 페이지의 『문자 데이터 및 속성이 있는 XML 요소의 경우』
내용에 특수 문자가 포함되어 있는 XML 요소 또는 속성	escape=true	90 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』
DOCTYPE 선언의 경우	type=doctype	63 페이지의 『XML DOCTYPE 선언의 경우』
CDATA 섹션의 경우	type=cdata	63 페이지의 『CDATA 섹션의 경우』
XML 문서에 추가할 주석	type=comment	91 페이지의 『XML 주석의 경우』
처리 명령어	type=pi	91 페이지의 『XML 처리 명령어의 경우』

주: 속성 응용프로그램 특정 정보는 (a | b | c) 태그 양식의 포함하여 반복 선택사항을 나타내는 다중 카디낼리티 속성을 지정할자세한 정보는 55 페이지의 『DTD를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

XML 요소의 경우: XML 요소를 나타내는 모든 단순(String) Business Object 속성에는 연관된 요소를 식별하기 위한 elem_name 태그가 해당 응용 프로그램 특정 정보에 있어야 합니다.

elem_name=*XML 요소 이름*

예를 들어, Business Object 속성인 CustLName이 단순 XML 속성을 나타내는 경우, 해당 응용프로그램 특정 정보는 다음과 같습니다.

```
Name = CustLName
AppSpecificInfo = elem_name=CustLName;
```

XML 요소 이름은 특수 문자(예: 마침표 및 하이픈(-))를 포함할 수 있습니다. 그러나 Business Object 속성의 이름은 이러한 특수 문자를 포함할 수 없습니다. 따라서 XML 요소의 이름을 elem_name 태그에 지정해야 합니다. Business Object 속성의 이름을 지정하려면, XML ODA는 XML 요소의 이름에서 특수 문자를 제거하고 이를 밑줄(_) 문자로 바꿉니다.

다음 예제에서, XML 요소에 대한 응용프로그램 특정 정보는 속성이 특수 문자를 포함하고 있으므로 실제 XML 요소 이름과의 차이점을 지정합니다.

```
Name = Phone_Tag
AppSpecificInfo = elem_name=Phone#Tag;
```

XML 요소의 실제 이름은 Business Object 속성 이름에는 올바르지 않은 파운드 기호(#)를 포함합니다. 그러므로, 응용프로그램 특정 정보의 elem_name 태그는 실제 XML 요소 이름을 지정합니다. 연관된 Business Object 속성의 이름에는 파운드 기호는 밑줄로 교체됩니다.

PCDATA만 있는 XML 요소의 경우: 문자 데이터만 있는 XML 요소는 PCDATA 요소 내용 지정자만 있는 혼합 요소입니다. PCDATA만 있는 XML 요소를 나타내는 Business Object 속성은 응용프로그램 특정 정보에 다음과 같은 type 태그가 있어야 합니다.

```
type=pcdata
```

이 경우, 요소 이름은 응용프로그램 특정 정보에서 첫 번째 필드이고, type 매개변수는 두 번째 필드입니다.

예를 들어, PCDATA만 포함하는 PartNumber라는 요소는 DTD에 다음 정의를 가집니다.

```
<!ELEMENT PartNumber (#PCDATA)>
```

Business Object 정의에 있는 해당 속성은 다음 응용프로그램 특정 정보를 가질 것입니다.

```
Name = PartNumber
AppSpecificInfo = elem_name=PartNumber;type=pcdata;
```

응용프로그램 특정 정보에 notag 텍스트가 포함된 경우에도 XML Data Handler가 XML 마크업을 생성하지 않습니다. 속성 값만 XML 문서에 추가합니다. 자세한 정보는 61 페이지의 『문자 데이터 및 속성이 있는 XML 요소의 경우』를 참조하십시오.

XML 속성의 경우: Business Object 속성이 XML 요소의 속성을 나타내는 경우, 해당 응용프로그램 특정 정보에 다음 태그가 포함되어야 합니다.

- attr_name 태그:

  ```
  attr_name=attrName
  ```

  XML 속성 이름은 특수 문자(예: 마침표 및 하이픈(-))를 포함할 수 있습니다. 그러나 Business Object 속성의 이름은 이러한 특수 문자를 포함할 수 없습니다. 따라서 XML 요소의 이름을 attr_name 태그에 지정해야 합니다. XML ODA는 Business Object 속성의 이름을 지정하기 위해 XML 속성 이름에서 모든 특수 문자를 제거합니다.

- type 태그:

  ```
  type=attribute
  ```

  이 type 태그는 연관된 Business Object 속성의 용도를 XML 속성으로 식별합니다.

주: 42 페이지의 『Business Object 구조』에 설명되어 있는 대로, XML 속성을 나타내는 모든 Business Object 속성은 XML 요소를 나타내는 모든 Business Object 속성 앞의 Business Object 정의에 나타나야 합니다.

예를 들어 Business Object 속성, ID가 XML 속성, ID를 나타내는 경우, 해당 응용프로그램 특정 정보는 다음과 같습니다.

```
Name = ID
AppSpecificInfo = attr_name=ID;type=attribute;
```

type=attribute 태그를 사용하는 다른 예를 보려면 『문자 데이터 및 속성이 있는 XML 요소의 경우』를 참조하십시오.

문자 데이터 및 속성이 있는 XML 요소의 경우: XML 요소가 PCDATA 또는 CDATA만 포함하고 하나 이상의 XML 속성을 갖는 경우, Business Object 정의는 다음 속성을 포함해야 합니다.

- 각 XML 속성의 Business Object 속성

  속성 이름은 XML 속성의 이름과 일치해야 합니다. 해당 속성 레벨 응용프로그램 특정 정보는 attr_name 및 type=attribute 태그를 포함해야 합니다. 자세한 정보는 60 페이지의 『XML 속성의 경우』를 참조하십시오.
- PCDATA 또는 CDATA 요소 내용 지정자와 연관된 문자 데이터의 business-object 속성

  이 속성에는 상위 XML 요소와 연관된 데이터가 있습니다. 응용프로그램 특정 정보는 다음을 포함해야 합니다.
  - 올바른 type=*typename* 태그(여기서 *typename*은 pcdata 또는 cdata)로 다음에 세미콜론(;)이 옵니다.
  - notag 키워드로, XML Data Handler가 중복되는 시작 태그(한 개는 Business Object용이며, 다른 하나는 속성용)를 생성하지 못하도록 합니다. XML Data Handler는 해당 속성에 대한 응용프로그램 특정 정보에 notag가 표시되지 않으면 모든 Business Object 속성에 대해 시작 및 끝 태그를 작성합니다.

예를 들어, Price라는 XML 요소에 Currency라는 속성이 있고 Price의 데이터를 필요로 한다고 가정하십시오.

```
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Price Currency NMTOKEN #IMPLIED>
```

Price 요소에 XML 속성이 있기 때문에, Business Object 정의에서 Currency에 대한 Business Object 속성도 작성되어야 합니다. 또한, Price 데이터를 가지는 다른 속성이 존재해야 합니다. Price 데이터의 속성은 응용프로그램 특정 정보에서 Data Handler가 속성에 대해 시작 및 끝 태그를 작성하지 않도록 notag를 지정해야 합니다.

Price 하위 Business Object는 다음과 같을 수 있습니다.

```
[BusinessObjectDefinition]
Name = Price
AppSpecificInfo = Price

   [Attribute]
Name = Currency
Type = String
AppSpecificInfo = attr_name=Currency;type=attribute;
   ...
[End]

   [Attribute]
Name = Price
Type = String
AppSpecificInfo = Price;type=pcdata;notag
   ...
[End]
```

이러한 경우, Data Handler는 Price 데이터에 대해 새 XML 요소를 생성하지 않고 상위 요소에만 데이터를 추가합니다.

특수 문자를 포함하는 XML 요소 또는 속성의 경우: XML 요소를 표시하는 Business Object 속성이나 이스케이프 처리를 필요로 하는 내용이 있는 XML 속성에는 해당 응용프로그램 특정 정보에 다음 태그가 있어야 합니다.

```
escape=true
```

속성이 값에 다음 특수 문자가 있는 XML 요소를 나타내면, 속성이 이스케이프 처리를 필요로 합니다.

- 작은 따옴표(')
- 큰 따옴표(")
- 앰퍼샌드 (&)
- 미만 부호(<)
- 초과 부호(>)

속성이 응용프로그램 특정 정보에 escape=true가 없으면, 속성은 이스케이프 처리되지 *않습니다*. 이 태그는 기존의 어떠한 응용프로그램 특정 정보든지 끝에 있어야 합니다. 예를 들면, 다음과 같습니다.

```
[Attribute]
Name=Data
Type=String
AppSpecificInfo=Price;type=pcdata;escape=true
[End]
```

속성의 응용프로그램 특정 정보가 이스케이프 태그를 포함하지 않으면, XML Data Handler는 DefaultEscapeBehavior 등록 정보의 값을 확인하여 이스케이프 처리 수행 여부를 결정합니다.

- DefaultEscapeBehavior가 true이면, XML Data Handler가 모든 속성 값에서 이스케이프 처리를 수행합니다.
- DefaultEscapeBehavior가 false이면, XML Data Handler가 *오직* 응용프로그램 특정 정보에 escape 태그가 있는 속성에서만 이스케이프 처리를 수행합니다.

XML DOCTYPE 선언의 경우: Business Object 속성이 프롤로그에 문서 유형 선언을 나타내는 경우, 응용프로그램 특정 정보에 다음 type 태그가 포함되어야 합니다.

```
type=doctype
```

예를 들어 Business Object 속성인 DocType이 DOCTYPE 요소를 나타내는 경우, 해당 응용프로그램 특정 정보는 다음과 같습니다.

```
Name = DocType
AppSpecificInfo = type=doctype;
```

DocType 속성의 값이 다음과 같을 경우,

```
DOCTYPE CUSTOMER "customer.dtd"
```

Data Handler는 다음 XML을 생성합니다.

```
<!DOCTYPE CUSTOMER "customer.dtd">
```

또한 이 응용프로그램 특정 정보는 XML 문서에 일반 엔티티 선언을 포함하기 위해 사용될 수 있습니다. 그러나 내부 DTD 또는 매개변수 엔티티 선언을 포함하기 위한 명시적 지원은 없습니다. 이는 응용프로그램 특정 정보에서 type=doctype의 속성 값에 전체 텍스트를 넣어 문서에 포함될 수 있습니다.

CDATA 섹션의 경우: Business Object 속성이 XML 문서 내에서 CDATA를 표시할 경우, 응용프로그램 특정 정보에는 다음 type 태그가 포함되어 있어야 합니다.

```
type=cdata
```

예를 들어, Business Object 속성인 UserArea가 CDATA 속성을 나타내는 경우, 응용프로그램 특정 정보는 다음과 같습니다.

```
Name = UserArea
AppSpecificInfo = type=cdata;
```

XML 주석의 경우: XML Data Handler가 Business Object를 XML 문서로 변환할 때, XML 문서에 주석을 추가한다고 지정할 수 있습니다. Data Handler가 주석을 추가할 수 있게 하려면, 다음 단계를 수행하십시오.

- Business Object 정의에 주석을 나타내는 Business Object 속성(단수 또는 복수)을 작성하십시오.

  주: XML ODA는 XML 주석에 대한 Business Object 속성을 자동으로 생성하지 *않습니다*. 96 페이지의 『수동으로 Business Object 정의 작성』에 설명된 것처럼 이 속성을 수동으로 추가해야 합니다.

- 속성 레벨 응용프로그램 특정 정보에 다음 type 태그를 포함시키십시오.

  type=comment

- 실제 Business Object에서 주석 텍스트를 이 속성의 값으로서 지정하십시오.

예를 들면, Comment라는 Business Object 속성이 Data Handler가 XML 문서에 추가해야 하는 XML 주석을 나타내는 경우, Comment 속성은 다음과 같을 것입니다.

```
Name = Comment
AppSpecificInfo = type=comment;
```

이 속성이 Customer information update from application A"라는 값을 가질 경우, 다음 XML이 생성됩니다.

```
<!--Customer information update from application A-->
```

XML 처리 명령어의 경우: Business Object 속성이 처리 명령어를 나타내는 경우, 응용프로그램 특정 정보에는 다음 type 태그가 포함되어 있어야 합니다.

```
type=pi
```

예를 들어, Business Object 속성인 XMLDeclaration이 프롤로그에 XML 선언을 나타내는 경우, 응용프로그램 특정 정보는 다음과 같습니다.

```
Name = XMLDeclaration
AppSpecificInfo = type=pi;
```

속성 값이 다음과 같을 경우

```
xml version = "1.0"
```

XML Data Handler는 다음과 같은 XML을 생성합니다.

```
<?xml version="1.0"?>
```

DTD에서 Business Object 정의 작성

DTD는 XML 문서의 형식에 대해 설명합니다. 그러므로 DTD는 Business Object 정의에 필요한 정보를 얻는 데 매우 유용합니다. DTD에 있는 구조 정보를 Business Object 정의로 변환하기 위해 XML Object Discovery Agent(ODA)를 사용할 수 있습니다. XML ODA에 대한 정보는 95 페이지의 『XML ODA를 사용하여 Business Object 정의 작성』을 참조하십시오.

주: XML Data Handler의 이전 버전은 DTD에서 Business Object 정의를 작성하는 두 가지 도구를 포함합니다. Edifecs SpecBuilder와 현재는 사용하지 않는 XMLBorgen 유틸리티입니다. XML ODA이 두 도구를 대신하고 이들 도구는 기능면에서 바뀌었습니다.

지원되는 DTD 구조

XML ODA는 다음과 같은 DTD 구조를 지원합니다.

- 엔티티 -- 다음 양식의 사용자 정의 엔티티는 참조할 때마다 인식되어 대체됩니다.

  ```
  <!ENTITY % name value>
  ```

  주: ENTITY 태그와 내용 사이에, 또는 요소나 속성 태그와 태그 내용 사이에 문자가 있을 경우, XML ODA가 줄 바꾸기 문자(\n), 캐리지 리턴(\r) 또는 탭 문자(\t)를 제거합니다.

- 외부 DTD -- XML ODA는 외부 DTD에 대한 참조를 지원합니다. (여기서 하나의 DTD는 하나 이상의 다른 DTD를 말합니다.) 외부 DTD가 로컬 파일 시스템에 있을 *경우에만* 이를 해석할 수 있습니다. 이들은 DTD를 찾기 위해 URL에 액세스할 수 없습니다. 두 도구 모두 항상 로컬 파일 시스템에서 외부 DTD에 대한 참조를 찾으려고 합니다. 도구는 이를 무시하지 *않습니다*.

- ANY 지시문 -- XML ODA는 내용으로 ANY가 있는 요소의 String 속성을 작성합니다. 예를 들어, DTD 구성이 다음과 같을 경우

  ```
  <!ELEMENT SCENE (ANY) >
  ```

  해당되는 Business Object 정의는 다음과 같습니다.

  ```
  [Attribute]
  Name = SCENE
  Type = String
  Cardinality = 1
  MaxLength = 255
  IsKey = false
  IsForeignKey = false
  IsRequired = true
  AppSpecificInfo = SCENE;type=pcdata;
  [End]
  ```

- 프롤로그 -- XML Data Handler는 Business Object 정의에 해당 요소에 해당하는 속성이 있는 경우 프롤로그 정보(예: DOCTYPE 및 XML 선언)를 채웁니다. 그러나 Data Handler는 DOCTYPE이라는 이름만 채웁니다. 가능한 기타 메타 정보는 채우지 *않습니다*.

지원되지 않는 DTD 구조

XML ODA는 대부분의 DTD를 처리할 수 있습니다. 그러나 다음 DTD 구조는 지원하지 *않습니다*.

- 조건부 섹션 -- 이러한 섹션의 구조는 다음과 같습니다.

  ```
  <![INCLUDE[ information to be included ]]>
  <![IGNORE[ information to be ignored ]]>
  ```

- 이름 공간 -- xxx:yyy 양식의 태그는 단지 xxx:yyy 태그로 처리되고, 이름 공간 xxx에 속하는 yyy 태그로 처리되지 않습니다.

스키마 문서를 사용하는 XML 문서

*XML 스키마*는 XML 문서의 템플리트(스키마)를 정의하는 *스키마 문서*(.xsd 확장자)를 사용하는 XML 문서의 데이터 모델입니다. DTD와 달리 스키마 문서는 스키마를 설명하는 데 XML 문서와 동일한 구문을 사용합니다. XML 문서의 스키마를 나타내는 Business Object 정의는 문서의 구조를 보존하고 기록하는 데 스키마 문서의 정보를 사용합니다. 이 섹션에서는 스키마 문서에서 Business Object 정의의 구조 정보를 알아보도록 다음 정보를 제공합니다.

- 『스키마 문서를 기본으로 하는 Business Object 정의를 위한 요구사항』
- 93 페이지의 『스키마 문서에서 Business Object 정의 작성』

스키마 문서를 기본으로 하는 Business Object 정의를 위한 요구사항

스키마 문서를 나타내는 Business Object 정의가 XML Data Handler의 요구사항을 준수하도록 하려면, 다음을 포함하는 이 섹션의 지침을 사용하십시오.

- 『스키마 문서의 Business Object 구조』
- 75 페이지의 『스키마 문서의 Business Object 속성 등록 정보』
- 76 페이지의 『스키마 문서의 XML 구성요소에 대한 응용프로그램 특정 정보』

스키마 문서의 Business Object 구조

스키마 문서를 기본으로 하는 Business Object 정의를 사용하여 XML Data Handler에 의해 처리되는 Business Object는 다음 규칙을 따라야 합니다.

- 스키마 문서에는 다음과 같은 Business Object 정의가 필요합니다.
  - schema 요소를 나타내는 최상위 레벨 Business Object 정의
  - 스키마 문서의 루트 요소를 나타내는 root-element Business Object 정의. 일반, 혼합 또는 랩퍼 Business Object 정의가 스키마 문서의 루트 요소를 나타냅니다.

  자세한 정보는 67 페이지의 『스키마 문서의 필수 Business Object 정의』를 참조하십시오.
- 일반, 혼합 또는 랩퍼 Business Object 정의는 포함되어 있는 XML 구성요소 또한 나타낼 수 있습니다.

XML 문서의 스키마 문서 예는 그림 18에 표시됩니다. 스키마 문서의 이름은 Order이며, 응용프로그램 Order 엔티티에 해당하는 요소가 있습니다. 이 샘플 스키마 문서는 샘플 DTD 문서가 설명한 것과 동일한 Business Object 구조에 대해 설명합니다. 53 페이지의 그림 13에서는 Order 스키마 문서와 연관된 XML 문서에 해당하도록 작성할 수 있는 Business Object 정의의 구조를 보여줍니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

        <xs:complexType name="AccessoryType">
                <xs:sequence>
            <xs:element ref="Quantity"/>
                        <xs:element ref="Type"/>
                </xs:sequence>
                <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:complexType>
        <xs:element name="Order">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="Unit" type="UnitType"
                   maxOccurs="unbounded"/>
                        </xs:sequence>
    </xs:complexType>
        </xs:element>
        <xs:element name="PartNumber" type="xs:string"/>
        <xs:element name="Price" type="xs:string"/>
        <xs:element name="Quantity" type="xs:string"/>
        <xs:element name="Type" type="xs:string"/>
        <xs:complexType name="UnitType">
                <xs:sequence>
                        <xs:element ref="PartNumber" minOccurs="0"/>
            <xs:element ref="Quantity"/>
            <xs:element ref="Price"/>
            <xs:element name="Accessory" type="AccessoryType" minOccurs="0"
                   maxOccurs="unbounded"/>
                </xs:sequence>
    </xs:complexType>
</xs:schema>
```

그림 18. 샘플 XML 스키마 문서

스키마 문서의 필수 Business Object 정의: 스키마 문서를 나타내려면 42 페이지의 『Business Object 구조』에 설명되어 있는 최소한 두 개 이상의 Business Object 정의가 필요합니다. 스키마 문서의 경우, 이 Business Object 정의에는 다음과 같은 추가 요구사항이 있습니다.

- *최상위 레벨* Business Object는 `schema` 요소를 나타내며 다음을 포함해야 합니다.
  - XML 버전을 나타내는 `XMLDeclaration` 속성

    이 속성은 응용프로그램 특정 정보에 `type=xmlns` 태그가 있어야 합니다. 자세한 정보는 91 페이지의 『XML 처리 명령어의 경우』를 참조하십시오.
  - 스키마 문서에 루트 요소를 나타내는 속성

    42 페이지의 『Business Object 구조』에 설명되어 있는 대로, 이 속성의 유형은 단일 카디널리티 Business Object여야 합니다. 이 유형은 루트 요소에 대한

Business Object 정의입니다. 응용프로그램 특정 정보는 elem_name 태그로 이 요소의 이름을 나열해야 합니다. 자세한 정보는 85 페이지의 『XML 요소의 경우』를 참조하십시오.

- *root-element* Business Object 정의는 스키마 문서의 루트 요소를 나타냅니다. 스키마 문서는 글로벌 레벨에 여러 XML 구성요소를 정의할 수 있으므로 Root 구성 등록 정보를 통해 루트 요소로 고려해야 할 요소를 XML ODA에 알려줄 수 있습니다. 루트 요소 Business Object 정의에는 다음과 같은 속성이 포함될 수 있습니다.
  - 스키마 문서(단수 또는 복수)가 상주하는 위치를 나타내는 schemaLocation 속성(선택적)

    XML ODA가 최상위 레벨 Business Object 정의에 schemaLocation 속성을 생성하는지 여부는 해당 DocTypeOrSchemaLocation 구성 등록 정보의 설정에 따라 다릅니다. 자세한 정보는 91 페이지의 『XML 스키마 위치의 경우』를 참조하십시오.
  - 루트 요소에 있는 각 XML 구성요소의 속성

주: 일반적인 Business Object 요구사항 목록은 41 페이지의 『Business Object 정의의 요구사항』을 참조하십시오.

이 두 가지 필수 Business Object 정의 모두 대상 이름 공간 및 임의의 구성요소 이름 자격을 정의하는 Business Object 레벨 응용프로그램 특정 정보가 필요합니다. 자세한 정보는 76 페이지의 『Business-object 레벨 응용프로그램 특정 정보』를 참조하십시오.

67 페이지의 그림 18에 있는 XML schema 요소는 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">
```

그림 19에서는 이 스키마 요소를 나타내는 최상위 레벨 Business Object 정의인 TopLevel을 보여줍니다.

```
[BusinessObjectDefinition]
Name=TopLevel
AppSpecificInfo=elem_fd=qualified;attr_fd=unqualified
...
   [Attribute]
   Name=XMLDeclaration
Type=String
   AppSpecificInfo=type=pi;
   ...
[End]

   [Attribute]
   Name=Order
   Type=TopLevel_Order
   AppSpecificInfo=elem_name=Order;
   ...
[End]
...
[End]
```

그림 19. 샘플 최상위 레벨 Business Object 정의

XML ODA가 그림 19에 최상위 레벨 Business Object 정의를 생성한 경우, 다음 ODA 구성 등록 정보를 설정한 상태여야 합니다.

ODA 구성 등록 정보	**등록 정보 값**
BOPrefix	설정되지 않음
TopLevel	"TopLevel"
Root	"Order"
DoctypeorSchemaLocation	true

루트 요소 Business Object 정의의 예는 82 페이지의 그림 26을 참조하십시오.

스키마 문서를 기반으로 하는 Business Object 정의: *일반* Business Object 정의는 다음 중 임의의 XML 구성을 나타냅니다.

- 하위 요소의 sequence 그룹을 포함하는 복합 유형(이름 지정 또는 익명)을 포함하는 XML 요소

  sequence 그룹의 각 하위 요소는 Business Object 정의의 속성으로 나타납니다. 자세한 정보는 87 페이지의 『복합 유형의 XML 요소의 경우』를 참조하십시오.

  주: 복합 유형에 choice 또는 all 그룹이 있는 경우, 랩퍼 Business Object 정의로 나타내야 합니다. 자세한 정보는 72 페이지의 『스키마 문서를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.
- 속성이 있는 XML 요소

이러한 유형의 Business Object 정의의 경우, Business Object 레벨 응용프로그램 특정 정보에는 특수 정보가 필요하지 *않습니다*. 일반 Business Object 정의의 속성은 XML 복합 유형에 정의되어 있는 요소를 나타냅니다.

주: 하위 요소의 순서 그룹을 포함하는 복잡한 유형의 경우, 순서 그룹에 있는 각각의 하위 요소는 Business Object 정의의 속성으로 표현됩니다.

예를 들어, Unit라는 XML 요소가 다음과 같이 정의되는 것으로 가정하십시오.

```
<xsd:element name="Unit">
<xsd:complexType>
    ...
    </xsd:complexType>
</element>
```

다음 Business Object 정의가 Unit 요소를 나타냅니다.

```
[BusinessObjectDefinition]
Name = MyApp_Unit
AppSpecificInfo =
[Attribute]
   ...
```

스키마 문서를 기반으로 하는 혼합 Business Object 정의: 혼합 Business Object 정의는 문자 데이터 및 기타 하위 요소의 혼합 내용을 포함하는, 혼합 XML 요소를 나타냅니다. 스키마 문서는 다음과 같이 mixed 속성이 true로 설정되어 있는 복합 유형으로서 혼합 유형 XML 요소를 설명합니다.

```
<xsd:complexType mixed="true">
        <xsd:sequence>
        <xsd:element name="subElement1" type="subElementType"/>
        ...
        </xsd:sequence>
    </xsd:complexType>
```

주: 이 섹션에 설명된 대로 혼합 Business Object는 요소 및 문자 데이터의 반복 목록을 위해 사용됩니다. 선택사항 목록이 어떠한 문자 데이터도 가지지 않으면, 82 페이지의 『규정된 구성요소 이름』에 설명된 것처럼 랩퍼 Business Object를 사용하십시오.

이 복합 유형은 mixed 속성을 true로 설정하기 때문에, 정의하는 하나 이상의 부속 요소뿐만 아니라 문자 데이터도 가질 수 있습니다. mixed 속성이 false로 설정되면, 문자 데이터가 복합 유형에 허용됩니다.

예를 들어, 그림 20에서는 스키마 문서에 정의된 혼합 유형 XML 요소를 보여줍니다.

```
<xsd:complexType name="Cust" mixed="true">
        <xsd:sequence>
        <xsd:element name="Name"/>
        <xsd:element name="Address"/>
        <xsd:element name="Phone"/>
        </xsd:sequence>
    </xsd:complexType>
```

그림 20. 혼합 유형 XML 요소의 샘플 스키마 문서

혼합 유형 XML 요소를 나타내려면 다음 두 가지 Business Object 정의가 필요합니다.

- 상위 Business Object 정의

  이 상위 Business Object 정의에는 다중 카디널리티 Business Object 배열을 나타내는 하나의 속성이 있습니다. 이 속성은 유형으로서 연관된 랩퍼 Business Object의 Business Object 정의를 가집니다. 상위 Business Object 정의에는 다음 응용프로그램 특정 정보가 있습니다.

  – Business Object 레벨 응용프로그램 특정 정보는 다음 구성요소로 되어 있습니다.

    - 연관된 혼합 유형 XML 요소의 이름과 세미콜론(;)
    - `type=MIXED` 태그

  – 다중 카디널리티 속성에는 해당 응용프로그램 특정 정보에 다음 태그가 있습니다.

    (*mixedTypeElement*|*subElement1*|...|*subElementN*)

    설명:

    - *mixedTypeElement*는 연관된 혼합 유형 XML 요소의 이름입니다.
    - *subelement1...subElementN*은 복합 유형으로 정의된 부속 요소 각각에 해당합니다.

    파이프(|) 문자로 부속 요소 각각을 구분해야 하고 전체 태그는 괄호로 묶여야 합니다.

- 랩퍼 Business Object 정의

  랩퍼 Business Object 정의는 혼합 데이터의 속성을 가집니다.

  – `type=pcdata` 태그뿐만 아니라(문자 데이터를 나타내는) `notag` 태그(데이터가 별도의 요소가 아닌 현재 데이터와 연관되어 있다는 것을 나타내는)를 필요로 하는, 문자 데이터를 위한 하나의 속성.

  – 혼합 유형 요소에 정의된 부속 요소 각각을 위한 하나의 속성 각 속성은 `type=pcdata` 태그(단순 유형을 나타낸다는 것을 표시하는)를 필요로 합니다.

  `type=pcdata` 태그에 대한 자세한 정보는 87 페이지의 『복합 유형의 XML 요소의 경우』를 참조하십시오.

주: 이 랩퍼 Business Object 정의는 Business Object 레벨에서 응용프로그램 특정 정보를 필요로 하지 *않습니다*.

71 페이지의 그림 20에서 Business Object 정의인 MyApp_Cust가 Cust 혼합 유형 요소를 나타내는 경우, 해당 응용프로그램 특정 정보는 다음과 같습니다.

```
[BusinessObjectDefinition]
Name = MyApp_Cust
AppSpecificInfo = type=MIXED;

[Attribute]
   Name=Cust_wrapper1
   Type=MyApp_CustWrapper1
Cardinality=n
AppSpecificInfo=(Cust|Address|Phone)
   ...
[End]
```

스키마 문서를 기반으로 하는 랩퍼 Business Object 정의: *랩퍼* Business Object 정의는 반복 선택사항 목록을 나타냅니다. 이 유형의 Business Object 정의는 XML 요소에 임의의 순서 및 임의의 카디널리티로 나타날 수 있는 하위가 있는 경우 필요합니다. 랩퍼 Business Object는 특정 XML 문서에서 하위 요소의 순서 및 카디널리티를 보존합니다.

주: 이 섹션에 설명된 대로 랩퍼 Business Object는 문자 데이터가 *아닌* 요소가 있는 반복 선택사항 목록을 위해 사용됩니다. 선택사항 목록에 문자 데이터가 있으면, 혼합 Business Object를 사용하십시오. 자세한 정보는 70 페이지의 『스키마 문서를 기반으로 하는 혼합 Business Object 정의』를 참조하십시오.

스키마 문서는 다음 모델 그룹 중 하나가 있는 복합 유형으로서 선택사항 목록 XML 요소를 설명할 수 있습니다.

- 선택사항 그룹은 한 개의 요소만 표시되어야 하는 요소의 비순차 목록을 정의합니다.

  ```
  <xsd:complexType>
      <xsd:choice>
      ...
      </choice>
  </complexType>
  ```

  선택사항 그룹이 있는 복합 유형을 가지는 XML 요소를 나타내기 위해 XML Data Handler는 다음과 같이 DTD에 정의된 선택사항 목록 XML 요소를 위해 수행하는 것과 동일한 계층 구조 Business Object 정의를 예상합니다.

  – 유형이 랩퍼 Business Object인 *다중 카디널리티*를 가지는 단일 속성을 포함하는 상위 Business Object
  – 선택사항 그룹에서 각 부속 요소를 위한 속성을 가지는 랩퍼 Business Object 정의

이 Business Object 정의에는 DTD용 선택사항 목록 XML 요소와 동일한 형식으로 되어 있는 Business Object 레벨 응용프로그램 특정 정보가 있습니다. 자세한 정보는 55 페이지의 『DTD를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

- 모든 그룹은 요소들이 각각 1회 이상 나타날 수 없는 요소의 비순차 목록을 정의합니다.

  ```
  <xsd:complexType>
      <xsd:all>
      ...
      </xsd:all>
      </xsd:complexType>
  ```

  모든 그룹이 있는 복합 유형을 가지는 XML 요소를 나타내기 위해 XML Data Handler는 다음과 같이 Business Object 정의가 있는 계층 구조 Business Object 정의를 예상합니다.

  – 유형이 랩퍼 Business Object인 *다중 카디널리티*를 가지는 단일 속성을 포함하는 상위 Business Object
  – 모든 그룹에서 각 부속 요소의 속성이 있는 랩퍼 Business Object 정의

    이 Business Object 정의에는 DTD용 선택사항 목록 XML 요소와 동일한 형식으로 되어 있는 Business Object 레벨 응용프로그램 특정 정보가 있습니다. 자세한 정보는 55 페이지의 『DTD를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

카디널리티를 표시하기 위해 이 모델 그룹은 `minOccurs`와 `maxOccurs` 속성으로 발생 제한을 지원합니다. 자세한 정보는 76 페이지의 표 20을 참조하십시오.

예를 들어, 스키마 문서의 XML 요소 정의는 다음과 같을 수 있습니다.

```
<xsd:element name="CUST">
<xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="U"/>
            <xsd:element ref="I"/>
            <xsd:element ref="B"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
```

이 요소에는 선택적이며(목록 내 하나의 하위 요소는 *반드시* 나타나야 함), 임의 순서대로 나타날 수 있는 세 가지 하위 요소가 있습니다. 56 페이지의 그림 14에서는 이 유형의 XML 문서를 보여줍니다. 57 페이지의 그림 16에서는 이 XML 문서의 랩퍼 Business Object 정의를 보여줍니다. 58 페이지의 그림 17에서는 랩퍼 Business Object 정의를 보여줍니다.

스키마 문서를 기반으로 한 Business Object 정의의 유형 대체: *유형 대체*를 사용하여 파생된 유형을 개별 XML 문서 인스턴스에 기본 유형 대신 표시할 수 있습니다. 유형 대체가 발생할 경우, 하나의 데이터 유형이 있는 선언을 따르는 요소에 이를 확장하거나 제한하는 데이터 유형이 있을 수 있습니다. 다음 스키마 정의에서 ShirtType 및 HatType은 기본 ProductType의 파생된 유형입니다.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="items" type="ItemsType"/>
<xsd:complexType name="ItemsType">
        <xsd:sequence>
            <xsd:element ref="product" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
<xsd:element name="product" type="ProductType"/>
<xsd:complexType name="ProductType">
        <xsd:sequence>
          <xsd:element name="number" type="xsd:string"/>
            <xsd:element name="name" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
<xsd:complexType name="ShirtType">
    <xsd:complexContent>
    <xsd:extension base="ProductType">
        <xsd:sequence>
                <xsd:element name="size" type="xsd:string"/>
                <xsd:element name="color" type="xsd:string"/>
        </xsd:sequence>
            </xsd:extension>
    </xsd:complexContent>
    </xsd:complexType>
<xsd:complexType name="HatType">
    <xsd:complexContent>
    <xsd:extension base="ProductType">
        <xsd:sequence>
                <xsd:element name="size" type="xsd:string"/>
        </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

그림 21. 유형 대체를 사용하는 샘플 스키마

위의 스키마를 기반으로 하는, 다음과 같은 XML 문서가 유효합니다.

```
<items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<product>
      <number>999</number>
      <name>Special Seasonal</name>
</product>
<product xsi:type="ShirtType">
      <number>557</number>
      <name>Short-Sleeved Linen Blouse</name>
      <size>M</size>
      <color>blue</color>
</product>
<product xsi:type="HatType">
      <number>443</number>
      <name>Four-Gallon Hat</name>
      <size>L</size>
</product>
</items>
```

그림 22. XML 문서의 파생된 유형

ProductType이 발생하는 곳마다, xsi:type 속성에서 표시하는 파생된 유형 ShirtType 및 HatType을 대신 표시할 수 있습니다. 유형 대체가 발생한 XML 문서를 표시하기 위해, XML Data Handler는 랩퍼 Business Object를 XML 문서의 하위 속성으로 작성합니다. 이 랩퍼 Business Object에는 복합 유형(74 페이지의 그림 21의 ProductType) 및 파생된 유형(ShirtType 및 HatType)에 해당하는 하위 속성이 있습니다. 표 18 및 표 19에서는 위의 XML 스키마에서 생성되는 Business Object 정의를 표시합니다.

표 18. ItemsType에 대한 Business Object 정의

속성 이름	유형	카디널리티	**ASI**
Product	ProductTypeWrapper	N	(product);typeSub=true

표 19. ProductTypeWrapper에 대한 Business Object 정의

속성	유형	카디널리티	**ASI**
ProductType	ProductType	1	Elem_name=product; xsiType=ProductType
ShirtType	ShirtType	1	Elem_name=product; xsiType=ShirtType;
HatType	HatType	1	Elem_name=product; xsiType=HatType;

스키마 문서의 Business Object 속성 등록 정보

XML 문서의 Business Object 정의가 스키마 문서를 기본으로 하면, Business Object 속성들이 43 페이지의 『Business Object 속성 등록 정보』에서 논의된 제한사항을 가집니다. 또한, 스키마 문서 구문이 Business Object 속성의 "필수"를 판별할 수 있습니다. "필수"는 XML Data Handler가 속성을 필요로 하는지를 판별하는, 카디널리티와 속성이 키인지 여부를 포함하는 요소의 조합입니다. 속성이 필수이면, 필수 속성 등록 정보가 true로 설정되어야 합니다.

필수 속성 등록 정보의 설정은 다음과 같이 XML 요소 및 속성 스펙뿐만 아니라 카디널리티, 키 및 외부 키 속성 등록 정보의 설정에 따라 다릅니다.

- Business Object 속성의 카디널리티는 스키마 문서의 Occurrence Indicator 부분에 의해 판별됩니다. 이 카디널리티는 속성이 필수인지에 영향을 줍니다. 표 20에서는 카디널리티와 스키마 문서에 있는 요소 선언의 가능한 조합에 대해 대략적으로 요약합니다.

표 20. 스키마 문서의 카디널리티 및 "필수"

스키마 단편 발생	카디널리티	필수
지정 안됨	1	예
maxOccurs > 1	N	예
maxOccurs = ″unbounded″	N	예
minOccurs=0	영향 없음	아니오
minOccurs>1	N	예

- Business Object 속성이 필수인지는 스키마 문서에 있는 use 속성에 의해서도 판별됩니다. 표 21에서는 use 속성의 가능한 값이 ″필수″인지에 대해 대략적으로 설명합니다.

표 21. 스키마 문서의 "필수"

스키마 단편 발생 속성: use	카디널리티	필수
지정 안됨	영향 없음	아니오
use=required	영향 없음	예

- Business Object 속성이 1차 키인지 foreign key인지는 스키마 문서의 id 속성에 의해 판별됩니다. 키가 있는지가 속성이 필수인지에 영향을 줍니다. 표 22에서는 id 속성의 값이 어떻게 Business Object 속성의 ″필수″에 영향을 주는지에 대해 대략적으로 설명합니다.

표 22. 스키마 문서의 키 및 ″필수″

스키마 단편 속성: id	키	필수	주석
id=ID	예	아니오	

스키마 문서의 XML 구성요소에 대한 응용프로그램 특정 정보

이 섹션에서는 스키마 문서를 기본으로 하는 Business Object 정의의 응용프로그램 특정 정보 형식에 대한 다음 정보를 제공합니다.

- 『Business-object 레벨 응용프로그램 특정 정보』
- 84 페이지의 『속성 응용프로그램 특정 정보』

Business-object 레벨 응용프로그램 특정 정보: XML Data Handler는 다음 유형의 Business Object 정의를 사용하여 스키마 문서에 정의되어 있는 다른 종류의 루트 XML 요소를 나타냅니다. 이러한 유형의 Business Object 정의는 Business Object 레벨에서 응용프로그램 특정 정보로 구별됩니다.

표 23. Business Object 레벨 응용프로그램 특정 정보에 대한 태그

태그 응용프로그램 특정 정보	설명	자세한 정보
target_ns	스키마 문서의 대상 이름 공간을 지정합니다.	『스키마 이름 공간』
attr_fd	속성 이름의 규정 여부를 지정합니다.	82 페이지의 『규정된 구성요소 이름』
elem_fd	로컬로 선언된 요소 이름의 규정 여부를 지정합니다.	82 페이지의 『규정된 구성요소 이름』

주: 비즈니스 레벨 응용프로그램 특정 정보 또한 type=MIXED 태그를 포함합니다. 자세한 정보는 70 페이지의 『스키마 문서를 기반으로 하는 혼합 Business Object 정의』를 참조하십시오.

스키마 이름 공간: DTD와 달리 스키마 문서는 최소한 하나 이상의 이름 공간을 정의해야 합니다. *이름 공간*은 XML 문서 내에 있는 요소의 이름, 요소 유형 및 속성을 위한 문맥을 제공합니다. 이름 공간은 HTTP, FTP 및 다른 종류의 경로를 포함하는 URI(Uniform Resource Identifier)입니다. 표 24에서는 스키마 문서가 스키마 구성요소 사이의 참조를 분석하도록 선언할 수 있는 이름 공간을 보여줍니다.

표 24. XML 이름 공간

이름 공간	설명	이름	공통 접두부
XML 스키마 이름 공간	XSDL(XML Schema Definition Language)에 사용되는 모든 구성요소(예: element, schema 및 simpleType)를 정의합니다.	http://www.w3.org/2001/XMLSchema	xsd, xs
XML 스키마 인스턴스 이름 공간	스키마 인스턴스와 연관된 네 가지 속성(type, nil, schemaLocation 및 noNamespaceSchemaLocation)을 정의합니다.	http://www.w3.org/2001/XMLSchema-instance	xsi
사용자 정의 이름 공간	선언되거나 글로벌 선언(예: element, attribute, type 또는 group)에 의해 정의된 모든 구성요소를 정의합니다. **주:** 로컬로 선언된 요소는 대상 이름 공간을 사용할 수도 있고 사용하지 않을 수도 있습니다. 자세한 정보는 82 페이지의 『규정된 구성요소 이름』을 참조하십시오.	사용자 정의	사용자 정의

주: XML ODA는 대상 이름 공간이 여러 개인 스키마 문서를 지원합니다.

모든 스키마 문서는 targetNamespace 태그와 함께 글로벌 구성요소(요소, 속성, 유형 또는 그룹)가 속하는 이름 공간을 식별하는 하나의 *대상 이름 공간*을 선언할 수 있습니다. 스키마 요소에 targetNamespace 태그가 포함되는 경우, XML 문서에 대해 선언된 대상 이름 공간을 지정하려면 해당 스키마 문서에 대해 생성된 각 Business Object 정의의 해당 응용프로그램 특정 정보에 target_ns 태그가 있어야 합니다.

target_ns=*대상 이름 공간의 URI 주소*

다음과 같이 모든 Business Object 정의에 대해 Business Object 레벨 응용프로그램 특정 정보에 target_ns 태그가 존재해야 합니다.

- 최상위 레벨 Business Object 정의의 경우, target_ns 태그의 값은 schema 요소의 targetNamespace 속성이 지정하는 값을 지정합니다.
- XML 요소를 나타내는 각 Business Object 정의 또한 해당 응용프로그램 특정 정보에 target_ns 태그를 포함해야 합니다. 모든 글로벌 구성요소(예: 요소, 속성 또는 유형)는 해당 스키마 문서의 대상 이름 공간에 속하므로, 이러한 구성요소를 나타내는 Business Object 정의 또한 스키마 문서의 대상 이름 공간을 지정해야 합니다.

주: 이전 XML Data Handler 버전에서는 최상위 레벨 Business Object 정의가 이름 공간 접두부에 대한 속성을 갖고, 특성 레벨 응용프로그램 특정 정보에 해당 type=defaultNS 및 type=xmlns 태그를 사용하는 것으로 예상했습니다. 이름 공간 접두부를 정의하기 위한 이 메커니즘은 바뀌었으나, 기존 Business Object 정의와의 역호환성을 위해 XML Data Handler는 계속 이 메커니즘을 지원합니다. 새 Business Object 정의는 이 섹션에 설명된 대로 target_ns 태그를 사용해야 합니다. XML ODA는 target_ns 태그를 사용하도록 수정되었습니다.

예를 들어, 그림 23의 스키마 문서는 XML 스키마 이름 공간(xsd 접두부 사용) 및 대상 이름 공간(기본 이름 공간)을 정의합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ibm.com/ns1" xmlns="http://www.ibm.com/ns1">

   <xsd:complexType name="TaxInfoType">
        <xsd:sequence>
         <xsd:element name="SSN" type="string">
</xsd:element>

         <xsd:element name="State" type="string">
</xsd:element>
        </xsd:sequence>
    </xsd:complexType>

   <xsd:element name="Customer">
<xsd:complexType>
        <xsd:sequence>
            <xsd:element name="TaxInfo" type="TaxInfoType">
</xsd:element>

            <xsd:element name="BillTo" type="xsd:string">
</xsd:element>
        </xsd:sequence>

         <xsd:attribute name="Name" type="xsd:string">
         </xsd:attribute>

         <xsd:attribute name="ID" type="xsd:string">
         </xsd:attribute>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

그림 23. Schema1.xsd 샘플 스키마 문서

XML ODA는 이 스키마 문서에 대한 세 가지 Business Object 정의 (BOPrefix_TopLevel, BOPrefix_TopLevel_Customer 및 BOPrefix_TopLevel_TaxInfoType)를 생성합니다. 여기서 BOPrefix 및 TopLevel은 이들 ODA 구성 등록 정보의 값입니다. 이들 세 가지 Business Object 정의는 모두 해당 Business Object 레벨 응용프로그램 특정 정보에 다음 내용이 들어 있습니다.

```
target_ns=http://www.ibm.com/ns1;elem_fd=unqualified;attr_fd=unqualified
```

주: 그림 23의 schema 요소에는 elementFormDefault 또는 attributeFormDefault 속성이 포함되지 않으므로, 이 응용프로그램 특정 정보에는 unqualified로 설정된 elem_fd 및 attr_fd 태그가 포함됩니다. 자세한 정보는 82 페이지의 『규정된 구성요소 이름』을 참조하십시오.

하나의 스키마 문서는 하나의 대상 이름 공간만 정의할 수 있습니다. 그러나 import 요소를 사용하여 다른 스키마 문서의 대상 이름 공간에 정의되어 있는 요소 및 속성을 포함할 수 있습니다.

그림 24에서는 79 페이지의 그림 23에 정의되어 있는 Schema1.xsd 문서를 기반으로 하는 스키마 문서를 보여줍니다. 이 스키마 문서는 ns2 이름 공간을 가져오고, 이 이름 공간은 TaxInfoType 복합 유형, BillTo 요소 및 Name 속성을 선언합니다.

```
<?xml version "1.0" encoding="UTF-8"?>
<xsd:schema smlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.com/ns1" xmlns="http://www.example.com/ns1"
   attributeFormDefault="qualified" elementFormDefault="qualified"
   xmlns:ns2="http://www.example.com/ns2">

   <xsd:import schemaLocation="Schema2.xsd"
      namespace="http://www.example.com/ns2"/>

   <xsd:element name="Customer2">
<xsd:complexType>
        <xsd:sequence>
            <xsd:element name="TaxInfo" type="ns2:TaxInfoType">
</xsd:element>

            <xsd:element ref="ns2:BillTo">
</xsd:element>
        </xsd:sequence>

         <xsd:attribute ref="ns2:Name">
         </xsd:attribute>

         <xsd:attribute name="ID" type="xsd:string">
         </xsd:attribute>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

그림 24. 대상 이름 공간 가져오기

루트 요소인 Customer2에 대한 Business Object 정의는 다음과 같이 TaxInfo, BillTo 및 Name XML 구성요소를 나타내는 해당 속성에 대한 이 대체 이름 공간을 지정해야 합니다.

- TaxInfo 속성의 유형은 ns2(http://www.example.com/ns2) 이름 공간에 정의되어 있는 TaxInfoType을 나타내는 Business Object 정의입니다(81 페이지의 그림 25 참조).
- BillTo 속성에는 해당 응용프로그램 특정 정보에 ns2 이름 공간을 연관된 BillTo 요소의 소스로 지정하기 위한 elem_ns 태그가 있습니다.
- Name 속성에는 해당 응용프로그램 특정 정보에 ns2 이름 공간을 해당 연관된 XML 속성인 Name의 소스로 지정하기 위한 attr_ns 태그가 있습니다.

그림 25에서는 ns2 이름 공간에 TaxInfoType, BillTo 및 Name XML 구성요소를 정의하는 스키마 문서를 보여줍니다.

```
<?xml version "1.0" encoding="UTF-8"?>
<schema smlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.com/ns2"
   attributeFormDefault="qualified" elementFormDefault="qualified"
   xmlns:ns2="http://www.example.com/ns2">

   <complexType name="TaxInfoType">
      <sequence>
         <element name="SSN" type="string">
</element>
         <element name="State" type="string">
</element>
      </sequence>
</complexType>

   <attribute name="Name" type="string"
   </attribute>

   <complexType name="AddressType">
      <sequence>
         <element name="Zip" type="string">
</element>
         <element name="Street" type="string">
</element>
      </sequence>
</complexType>

   <element name="BillTo" type="ns2:AddressType">
</element>
</schema>
```

그림 25. 두 번째 이름 공간 정의

그림 25에서는 Customer2 루트 요소에 대한 Business Object 정의를 보여줍니다.

```
[BusinessObjectDefinition]
Name=TopLevel_Customer2
AppSpecificInfo=target_ns=http://www.example.com/ns1;elem_fd=qualified;
   attr_fd=qualified
...
   [Attribute]
Name=Name
Type=String
   AppSpecificInfo=attr_name=Name;type=attribute;attr_ns=http://www.example.com/ns2
   ...
[End]

   [Attribute]
   Name=ID
Type=String
   AppSpecificInfo=attr_name=ID;type=attribute
   ...
[End]

   [Attribute]
Name=schemaLocation
Type=String
   AppSpecificInfo=attr_name=schemaLocation;type=xsischemalocation
   ...
[End]

   [Attribute]
   Name=TaxInfo
   Type=TopLevel_TaxInfoType
   AppSpecificInfo=elem_name=TaxInfo
   ...
[End]

   [Attribute]
Name=BillTo
   Type=TopLevel_AddressType
   AppSpecificInfo=elem_name=BillTo;elem_ns=http://www.example.com/ns2
   ...
[End]
...
[End]
```

그림 26. 샘플 루트 요소 Business Object 정의

규정된 구성요소 이름: XML 문서에서, 이름 공간과 연관된 구성요소의 이름은 다음과 같이 규정 또는 규정되지 않습니다.

- 규정되지 않은 이름에는 접두부가 *없으며* 이름 공간의 일부가 *아닙니다*.
- 규정된 이름은 다음 중 하나입니다.
  - 구성요소 이름에는 이름 공간과 연관된 접두부가 있습니다.

    하나 이상의 이름 공간에 접두부를 지정할 수 있습니다. xmlns:*prefix* 태그로 이름 공간의 접두부를 선언하십시오. 여기서 *prefix*는 선언된 접두부입니다. 이 구성요소 이름은 스키마 문서의 구성요소 정의에 *규정됩니다*. 구성요소 이름 앞에 접두부가 추가되기 때문입니다(*prefix*:*componentName*).

– 이름에는 접두부가 없지만, 기본 이름 공간의 일부입니다(요소에만 해당).

*기본 이름 공간*은 해당 구성요소 이름에 접두부를 포함하지 않는 구성요소와 연관시킬 이름 공간을 지정합니다. xmlns 태그로 기본 이름 공간을 선언하십시오.

XML Data Handler가 XML에서 Business Object로의 변환을 제대로 처리하려면, XML 문서 및 스키마 문서의 이름 공간이 다음과 같이 일치해야 합니다.

- 스키마 문서가 기본 이름 공간을 지정하면, XML 문서가 기본 이름 공간도 지정해야 합니다.
- 스키마 문서가 기본 이름 공간을 가지지 않으면, XML 문서가 기본 이름 공간을 가질 수 없습니다.

schema 요소의 elementFormDefault 속성은 로컬로 선언된 요소 이름의 규정 여부를 지정합니다. 기본적으로, 로컬로 선언된 요소는 규정되지 않으며 기본 이름 공간에 속합니다. elementFormDefault 속성 값은 표 25에서처럼 Business Object 레벨 응용프로그램 특정 정보에서 elem_fd 태그 값을 판별합니다.

표 25. elem_fd 태그 설정

elementFormDefault 값	**elem_fd 태그 값**
"unqualified"(또는 속성이 지정되지 않음) "qualified"	elem_fd=unqualified elem_fd=qualified

예를 들어, 79 페이지의 그림 23에 있는 스키마 문서의 schema 요소에는 elementFormDefault 속성이 *없습니다*. 따라서 이 스키마 문서의 모든 Business Object 정의에 있는 Business Object 레벨 응용프로그램 특정 정보(BOPrefix_TopLevel, BOPrefix_TopLevel_Customer 및 BOPrefix_TopLevel_TaxInfoType. 여기서 BOPrefix 및 TopLevel은 이 ODA 구성 등록 정보 값임)에는 다음 태그가 있습니다.

```
elem_fd=unqualified
```

주: 스키마 문서의 schema 요소에 다음 속성이 포함된 경우, 이 세 가지 Business Object 정의에 대한 Business Object 레벨 응용프로그램 특정 정보에 이와 동일한 태그가 포함됩니다.

```
elementFormDefault="unqualified"
```

개별 XML 요소에 form 속성이 포함되어 있는 경우, 이 form 속성 값은 elementFormDefault 속성의 설정을 대체합니다.

schema 요소의 elementFormDefault 속성은 요소 이름의 규정 여부를 지정합니다. 기본적으로, 속성 이름은 규정되지 않고 임의의 이름 공간에 속하지 않습니다. attributeFormDefault 속성 값은 표 26에서처럼, elementFormDefault 속성 값이 elem_fd 태그 값을 판별하는 것과 동일한 방법으로 Business Object 레벨 응용프로그램 특정 정보의 attr_fd 태그 값을 판별합니다.

표 26. attr_fd 태그 설정

attributeFormDefault 값	**elem_fd 태그 값**
"unqualified"(또는 속성이 지정되지 않음) "qualified"	attr_fd=unqualified attr_fd=qualified

예를 들어, 79 페이지의 그림 23에 있는 스키마 문서의 schema 요소에는 attributeFormDefault 속성이 *없습니다*. 따라서 이 스키마 문서의 모든 Business Object 정의에 있는 Business Object 레벨 응용프로그램 특정 정보(BOPrefix_TopLevel, BOPrefix_TopLevel_Customer 및 BOPrefix_TopLevel_TaxInfoType. 여기서 BOPrefix 및 TopLevel은 이 ODA 구성 등록 정보 값임)에는 다음 태그가 있습니다.

```
attr_fd=unqualified
```

주: 스키마 문서의 schema 요소에 다음 속성이 포함된 경우, 이 세 가지 Business Object 정의에 대한 Business Object 레벨 응용프로그램 특정 정보에 이와 동일한 태그가 포함됩니다.

```
attributeFormDefault="unqualified"
```

속성 응용프로그램 특정 정보: Business Object 정의의 속성은 다음 XML 구성요소를 나타낼 수 있습니다.

- 85 페이지의 『XML 요소의 경우』
- 87 페이지의 『복합 유형의 XML 요소의 경우』
- 89 페이지의 『XML 속성의 경우』
- 91 페이지의 『XML 처리 명령어의 경우』
- 91 페이지의 『XML 주석의 경우』
- 90 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』
- 91 페이지의 『XML 스키마 위치의 경우』

표 27에서는 이들 다른 XML 구성요소의 속성 레벨 응용프로그램 특정 정보에 대한 태그 및 이 태그에 대해 자세히 설명하는 이 매뉴얼의 절을 함께 보여줍니다.

표 27. 속성 응용프로그램 특정 정보에 대한 태그

Business Object 속성의 표시	**응용프로그램 특정 정보**	**자세한 정보**
XML 요소	elem_name=*XML 요소 이름* elem_ns=*요소 정의에 대한 이름 공간* elem_fd=*속성 양식 값*	『XML 요소의 경우』
복합 유형의 XML 요소	type=pcdata	87 페이지의 『복합 유형의 XML 요소의 경우』
XML 요소의 속성	attr_name=*XML 속성의 이름* type=attribute attr_ns=*속성 정의에 대한 이름 공간* attr_fd=*속성 양식 값*	89 페이지의 『XML 속성의 경우』
내용에 특수 문자가 포함되어 있는 XML 요소 또는 속성	escape=true	90 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』
XML 문서에 추가할 주석	type=comment	91 페이지의 『XML 주석의 경우』
처리 명령어	type=pi	91 페이지의 『XML 처리 명령어의 경우』
XML 인스턴스의 schemaLocation 또는 noNamespaceSchemaLocation 속성	type=xsischemalocation type=xsinoNSlocation	91 페이지의 『XML 스키마 위치의 경우』

주: 속성 응용프로그램 특정 정보는 (a | b | c) 태그 양식의 포함하여 반복 선택사항을 나타내는 다중 카디널리티 속성을 지정할자세한 정보는 72 페이지의 『스키마 문서를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

XML 요소의 경우: Business Object 속성이 XML 요소를 나타내는 경우, 해당 응용프로그램 특정 정보에 연관된 요소를 식별하기 위한 다음 elem_name 태그가 포함되어야 합니다.

elem_name=*XML 요소 이름*

XML 요소 이름은 특수 문자(예: 마침표 및 하이픈(-))를 포함할 수 있습니다. 그러나 Business Object 속성의 이름은 이러한 특수 문자를 포함할 수 없습니다. 따라서 XML 요소의 이름을 elem_name 태그에 지정해야 합니다. XML ODA는 Business Object 속성의 이름을 지정하기 위해 XML 요소 이름에서 모든 특수 문자를 제거합니다.

다음과 같은 경우 Business Object 속성이 XML 요소를 나타낼 수 있습니다.

- XML 요소가 XML 복합 유형이거나 이를 포함하는 경우

  이런 경우, Business Object 속성은 데이터 유형이 XML 복합 유형을 나타내는 Business Object 정의인 *복합* 속성입니다. elem_name 태그(속성의 응용프로그램 특정 정보에 있음)에는 XML 요소(또는 복합 유형)의 이름이 있습니다.

- XML 요소가 XML 복합 유형의 일부인 경우

이런 경우, Business Object 속성은 String 유형의 단순 속성입니다. 해당 응용프로그램 특정 정보에는 elem_tag(복합 유형에 XML 요소 이름 포함) 및 type=pcdata 태그가 포함됩니다. 자세한 정보는 87 페이지의 『복합 유형의 XML 요소의 경우』를 참조하십시오.

XML 요소를 나타내는 Business Object 속성 또한 해당 응용프로그램 특정 정보에 다음 태그를 포함해야 합니다.

- elem_fd 태그는 로컬로 선언된 요소 이름의 규정 여부를 표시하는 XML 요소의 form 속성에 대한 설정을 지정합니다. XML 요소에서 form="qualified" 속성을 지정한 경우, 값 elem_fd가 form 속성의 값으로 설정됩니다.

  예를 들어, 로컬로 선언된 XML 요소가 다음과 같은 정의를 갖는 것으로 가정하십시오.

  ```
  <xsd:element ref="Name" form="qualified"></xsd:element>
  ```

  연관된 해당 Business Object 속성은 다음과 같은 형식을 갖게 됩니다.

  ```
  [Attribute]
  Name=ns2:Name
  Type=String
  AppSpecificInfo=elem_name=Name;elem_fd=qualified;
  ...
  ```

  XML 요소가 form 속성을 지정하지 않는 경우, elementFormDefault 속성(schema 요소에 있음) 값이 요소 이름의 규정 여부를 판별합니다. 자세한 정보는 82 페이지의 『규정된 구성요소 이름』을 참조하십시오.

- elem_ns 태그는 XML 요소의 대상 이름 공간을 지정합니다(이 이름 공간이 스키마 문서의 대상 이름 공간과 다른 경우). 이 태그는 스키마 문서가 여러 이름 공간을 사용할 때 필요합니다. 한 스키마 문서에서 참조되는 XML 요소가 다른 스키마 문서의 대상 이름 공간에 정의되어 있는 경우, 이 태그는 해당 이름 공간의 이름을 나열합니다.

  예를 들어, 복합 유형의 XML 요소가 다음과 같은 정의를 갖는 것으로 가정하십시오.

  ```
  <xsd:element ref="ns2:BillTo"></xsd:element>
  ```

  연관된 해당 Business Object 속성은 다음과 같은 형식을 갖게 됩니다.

  ```
  [Attribute]
  Name=BillTo
  Type=String
  AppSpecificInfo=elem_name=BillTo;elem_ns=http:/www.imb.com/ns2;
  ...
  ```

  두 번째 이름 공간에 정의되어 있는 XML 요소를 포함하는 전체 스키마 문서 예는 77 페이지의 『스키마 이름 공간』을 참조하십시오.

복합 유형의 XML 요소의 경우: Business Object 정의가 XML 복합 유형(complexType)을 나타내는 경우, 이 복합 유형의 내용은 이 Business Object 정의에서 단순(String) 속성으로 표시됩니다. 이 Business Object 속성의 응용프로그램 특정 정보가 요소 이름을 식별하려면 *반드시* elem_name 태그를 포함해야 합니다.

주: 이 태그에 대한 자세한 정보는 85 페이지의 『XML 요소의 경우』를 참조하십시오.

XML 복합 유형의 경우, Business Object 속성은 표 28에서처럼 복합 유형 내용을 나타낼 수 있습니다.

표 28. XML 복합 유형의 내용

복합 유형 요소의 유형	설명	속성 응용프로그램 특정 정보
단순 XML 요소	문자 내용만 있는 요소. 다른 어떠한 요소 또는 XML 속성도 포함할 수 없습니다. 복합 유형에만 나타날 수 있습니다.	type=pcdata
단순 내용	문자 데이터만(요소가 아님)	type=pcdata;notag
속성이 있는 XML 요소	문자 데이터 및 하위 요소와 속성의 조합이 모두 있는 요소	없음 이러한 유형의 XML 요소는 단일 Business Object 속성으로서가 아닌 Business Object 정의로 표시되어야 합니다. 자세한 정보는 69 페이지의 『스키마 문서를 기반으로 하는 Business Object 정의』를 참조하십시오.

type=pcdata 태그를 사용하는 한 예로, 단순 XML 요소만 있는 TaxInfoType 복합 유형에 대한 정의가 79 페이지의 그림 23의 스키마 문서에 들어 있습니다. TaxInfoType XML 복합 유형에 대한 Business Object 정의(BOPrefix_TopLevel_TaxInfoType. 여기서 BOPrefix 및 TopLevel은 각각 이름을 지정한 ODA 구성 등록 정보 값임)에는 두 가지 속성이 있습니다. 각 속성은 그림 27에 표시된 대로 해당 응용프로그램 특정 정보의 연관된 XML 부속 요소 이름을 갖게 됩니다. 이 복합 유형에는 단순 XML 요소만 있으므로(하위 요소 또는 속성이 없음), 해당 Business Object 속성에도 응용프로그램 특정 정보에 type=pcdata 태그가 있어야 합니다.

```
[BusinessObjectDefinition]
Name=BOPrefix_TopLevel_TaxInfoType
AppSpecificInfo=target_ns=;elem_fd=unqualified;attr_fd=unqualified
...
[End]

[Attribute]
Name=SSN
Type=String
AppSpecificInfo=elem_name=SSN;type=pcdata
...
[End]

[Attribute]
Name=State
Type=String
AppSpecificInfo=elem_name=SSN;type=pcdata
...
[End]
```

그림 27. 단순 요소가 있는 XML 요소 유형에 대한 샘플 Business Object 정의

type=pcdata 태그와 결합해서 notag 키워드를 사용하는 한 예로, Price라는 XML 요소가 단순 내용만을 갖는 즉, 문자 데이터만을 포함하는 PriceType 복합 유형이라고 가정하십시오. 이 경우, simpleContent 요소가 Currency 속성을 정의했고, Price에 대한 데이터가 필요합니다.

```
<xsd:element name="Price" type="PriceType">
    <xsd:complexType name="PriceType">
        <xsd:simpleContent>
            <xsd:extension base="xsd:decimal">
                <xsd:attribute name="Currency" type="xsd:NMTOKEN"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</element>
```

PriceType 복합 유형에 대한 Business Object 정의에는 단순 내용과 연관된 문자 데이터를 보유하는 속성이 있습니다. 이 속성에 대한 응용프로그램 특정 정보에는 다음이 포함되어야 합니다.

```
type=pcdata;notag
```

notag 키워드는 XML Data Handler가 중복 시작 태그(하나는 Business Object 정의용, 다른 하나는 속성용)를 생성하지 못하도록 합니다. XML Data Handler는 해당 속성에 대한 응용프로그램 특정 정보에 notag가 표시되지 않으면 모든 Business Object 속성에 대해 시작 및 끝 태그를 작성합니다.

Price 하위 Business Object 정의는 다음과 같이 표시됩니다.

```
[BusinessObjectDefinition]
Name = Price
AppSpecificInfo =
```

```
[Attribute]
Name = Currency
Type = String
AppSpecificInfo = attr_name=Currency;type=attribute;
...
[End]

[Attribute]
Name = Price
Type = String
AppSpecificInfo = elem_name=Price;type=pcdata;notag
...
[End]
```

Price 데이터를 보유하기 위한 Business Object 속성이 존재해야 합니다. Price 데이터의 속성은 응용프로그램 특정 정보에서 Data Handler가 속성에 대해 시작 및 끝 태그를 작성하지 않도록 notag를 지정해야 합니다. 이러한 경우, Data Handler는 Price 데이터에 대해 새 XML 요소를 생성하지 않고 상위 요소에만 데이터를 추가합니다.

또한 Currency 속성 값을 보유하기 위해 존재해야 하는 다른 속성이 Business Object 정의에 있어야 합니다. 단순 내용이 속성을 포함하면, Business Object 정의도 각 XML 속성을 위한 속성을 포함해야 합니다. 속성의 응용프로그램 특정 정보에는 type=attribute 태그가 포함되어야 합니다. 자세한 정보는 『XML 속성의 경우』를 참조하십시오.

XML 속성의 경우: Business Object 정의가 XML 유형 또는 복합 유형을 나타내는 경우, 스키마 문서가 이 유형에 대해 선언하는 속성은 Business Object 정의에서 속성으로 나타납니다. Business Object 속성이 XML 요소의 속성을 나타내는 경우, 해당 응용프로그램 특정 정보는 *반드시* 다음 태그를 포함해야 합니다.

- attr_name 태그:

  attr_name=*XML 속성의 이름*

- type 태그:

  type=attribute

주: 이들 태그에 대한 자세한 정보는 60 페이지의 『XML 속성의 경우』를 참조하십시오. type=attribute 태그 사용에 대한 예는 87 페이지의 『복합 유형의 XML 요소의 경우』를 참조하십시오.

XML 속성을 나타내는 Business Object 속성 또한 해당 응용프로그램 특정 정보에 다음 태그를 포함해야 합니다.

- attr_fd 태그는 속성 이름의 규정 여부를 표시하는 XML 속성의 form 속성에 대한 설정을 지정합니다. XML 속성에서 form="unqualified"에 대해 form="qualified" 속성을 지정한 경우, attr_fd가 "form" 속성에 지정된 값을 갖습니다.

예를 들어, XML 속성이 다음과 같은 정의를 갖는 것으로 가정하십시오.

```
<xsd:attribute ref="Name" form="qualified"></xsd:attribute>
```

연관된 해당 Business Object 속성은 다음과 같은 형식을 갖게 됩니다.

```
[Attribute]
Name=ns2:Name
Type=String
AppSpecificInfo=attr_name=Name;type=attribute;attr_fd=qualified
...
```

XML 속성이 form 속성을 지정하지 않는 경우, attributeFormDefault 속성(schema 요소에 있음) 값이 속성 이름의 규정 여부를 판별합니다. 자세한 정보는 82 페이지의 『규정된 구성요소 이름』을 참조하십시오.

- attr_ns 태그는 XML 속성의 대상 이름 공간을 지정합니다(이 이름 공간이 *스키마 문서의 대상 이름 공간과 다른 경우*). 이 태그는 *스키마 문서*가 여러 이름 공간을 사용할 때 필요합니다. 한 *스키마 문서*에서 참조되는 XML 속성이 *다른 스키마 문서*의 대상 이름 공간에 정의되어 있는 경우, 이 태그는 해당 이름 공간의 이름을 나열합니다.

  예를 들어, XML 속성이 다음과 같은 정의를 갖는 것으로 가정하십시오.

  ```
  <xsd:attribute ref="ns2:Name"></xsd:attribute>
  ```

  연관된 해당 Business Object 속성은 다음과 같은 형식을 갖게 됩니다.

  ```
  [Attribute]
  Name=Name
  Type=String
  AppSpecificInfo=attr_name=Name;attr_ns=http://www.example.com/ns2;
  type=attribute
  ...
  ```

  두 번째 이름 공간에 정의되어 있는 XML 속성을 포함하는 전체 *스키마* 문서 예는 77 페이지의 『*스키마* 이름 공간』을 참조하십시오.

특수 문자를 포함하는 XML 요소 또는 속성의 경우: 내용에 특수 문자가 포함되어 있는 XML 요소 또는 XML 속성을 나타내는 Business Object 속성에는 XML Data Handler에 의한 이스케이프 처리가 필요합니다. Data Handler에 이스케이프 처리를 수행해야 함을 알리려면, Business Object 속성의 응용프로그램 특정 정보에 다음 태그가 포함되어야 합니다.

```
escape=true
```

스키마 문서를 사용하는 XML 문서를 위해 이스케이프 처리를 지정하는 단계는 DTD를 사용하여 *스키마*를 설명하는 XML 문서를 위해 이스케이프 처리를 지정하는 것과 동일합니다. 자세한 정보는 62 페이지의 『특수 문자를 포함하는 XML 요소 또는 속성의 경우』를 참조하십시오.

XML 주석의 경우: XML Data Handler가 Business Object를 XML 문서로 변환하는 경우, 속성의 응용프로그램 특정 정보에 다음 태그를 포함시켜 XML 문서에 XML 주석을 추가하도록 지정할 수 있습니다.

```
type=comment
```

XML ODA는 XML 주석에 대한 Business Object 속성을 자동으로 생성하지 *않습니다*. 이 속성은 수동으로 추가해야 합니다. 스키마 문서에서 설명하는 대로 XML 문서에 주석을 추가하는 단계는 DTD가 설명하는 XML 문서에 대한 주석을 정의하는 단계와 동일합니다. 자세한 정보는 63 페이지의 『XML 주석의 경우』를 참조하십시오.

XML 처리 명령어의 경우: XML 문서에 XML 처리 명령어가 있는 경우, 스키마 문서와 연관된 일부 Business Object 정의에 처리 값을 보유하는 속성이 있어야 합니다. 이 속성의 응용프로그램 특정 정보에는 다음 `type` 태그가 있어야 합니다.

```
type=pi
```

예를 들어, XML Data Handler가 XML 문서를 Business Object로 변환하는 경우, XML 버전에 `XMLDeclaration`이라는 최상위 레벨 Business Object 정의의 특수 속성을 부여합니다. 69 페이지의 그림 19에 있는의 최상위 레벨 Business Object에서는 TopLevel_Customer Business Object 정의의 XMLDeclaration 속성을 보여줍니다. `type=pi` 태그에 대한 자세한 정보는 64 페이지의 『XML 처리 명령어의 경우』를 참조하십시오.

XML 스키마 위치의 경우: 스키마 문서를 위해 스키마 위치를 참조하는 XML 문서는 다음과 같은 정보를 포함해야 합니다.

- XML 스키마 인스턴스 이름 공간의 선언 및 이 이름 공간으로의 `xsi` 접두부 맵핑
- 다음 XML 스키마 인스턴스 속성 중 하나를 포함
  - `xsi:schemaLocation`

    이 속성은 이름 공간의 이름을 스키마 위치에 연관시킵니다. 스키마 문서가 대상 이름 공간을 사용하면, 이 대상 이름 공간의 이름이 `xsi:schemaLocation` 속성(XML 문서에서)이 지정하는 이름 공간과 일치해야 합니다.
  - `xsi:noNamespaceSchemaLocation`

    이 속성은 스키마 위치를 나타냅니다. 스키마 문서가 대상 이름 공간을 사용하지 않으면, XML 문서가 `noNamespaceSchemaLocation` 속성을 포함하여 하나의 스키마 위치를 나타냅니다.

예를 들면, XML 문서가 92 페이지의 그림 28에 있는 이름 공간 선언을 가진다고 가정하십시오.

```
<order xmlns="http://sampleDoc.org.ord"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.com/order order.xsd">
   ...
/<order>
```

그림 28. XML 문서의 샘플 스키마 위치 정의

스키마 문서는 대상 이름 공간을 정의하는 다음 이름 공간 선언을 가질 수 있습니다.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://www.example.com/order"
     targetNamespace="http://www.example.com/order">
```

이 XML 문서를 나타내는 Business Object가 schemaLocation 속성에 정보를 보유하려면, 해당 Business Object 정의가 이 스키마 위치 정보에 대한 속성을 제공해야 합니다. 루트 요소 Business Object 정의에서, 이 스키마 위치 정보는 다음과 같이 표시됩니다.

- schemaLocation 속성을 사용하는 XML 문서에서는 최상위 레벨 Business Object 정의에 다음 등록 정보를 가진 속성이 있어야 합니다.
  - 속성 이름은 schemaLocation이고 이 속성의 유형은 String입니다.
  - 속성은 응용프로그램 특정 정보에 다음 type 태그가 있어야 합니다.

    ```
    type=xsischemalocation
    ```

  - 이 Business Object 속성의 값은 schemaLocation 속성의 값입니다.
- noNamespaceSchemaLocation 속성을 사용하는 XML 문서의 경우, 루트 요소 Business Object 정의에 다음 등록 정보를 가진 속성이 있어야 합니다.
  - 속성 이름은 noNamespaceSchemaLocation이고 이 속성의 유형은 String입니다.
  - 속성은 응용프로그램 특정 정보에 다음 type 태그가 있어야 합니다.

    ```
    type=xsinoNSlocation
    ```

이 XML 문서를 나타내는 스키마 문서의 루트 요소 Business Object 정의에는 스키마 위치에 대한 다음 속성이 포함되어야 합니다.

```
[Attribute]
Name=schemaLocation
Type=String
AppSpecificInfo=type=xsischemalocation;
```

주: 속성이 schemaLocation 또는 noNamespaceSchemaLocation XML 속성을 나타내는 경우, 해당 응용프로그램 특정 정보에 type=attribute 태그가 필요하지 않습니다.

XML ODA는 해당 DoctypeorSchemaLocation ODA 구성 등록 정보의 값에 따라 최상위 레벨 Business Object에 schemaLocation 또는 noNamespaceSchemaLocation 속성을 생성할지 여부를 결정합니다.

스키마 문서에서 Business Object 정의 작성

스키마 문서는 XML 문서의 내용을 설명하고 제한합니다. 그러므로 스키마 문서는 Business Object 정의에 필요한 정보를 얻는 데 매우 유용합니다. 스키마 문서에 있는 구조 정보를 Business Object 정의로 변환하기 위해 XML Object Discovery Agent(ODA)를 사용할 수 있습니다. XML ODA에 대한 정보는 95 페이지의 『XML ODA를 사용하여 Business Object 정의 작성』을 참조하십시오.

지원되는 스키마 문서 구조

XML ODA는 스키마의 요소 선언을 Business Object 정의의 속성으로 맵핑합니다. Business Object 속성 유형은 XML 요소 선언에 지정된 유형에 따라 다릅니다. 단순 유형은 String Java 유형으로 맵핑됩니다. 복합 유형은 Business Object 정의로 맵핑됩니다.

XML ODA는 다음과 같은 스키마 구조를 지원합니다.

- 단순 유형 -- 단순 유형은 최소단위(제한을 사용하여 구축되거나 파생됨), 목록 또는 통합 유형 중 하나일 수 있습니다. XML ODA는 이러한 유형을 모두 문자열의 Business Object 속성 유형으로 맵핑합니다. 자세한 정보는 87 페이지의 『복합 유형의 XML 요소의 경우』를 참조하십시오.
- 요소 와일드 카드 any -- XML ODA는 임의의 와일드 카드를 Business Object 정의의 단순 속성으로 맵핑합니다. 런타임에서 사용자는 데이터의 지식을 기반으로 해당 응용프로그램 특정 정보를 추가하여 이 속성을 구성해야 합니다.
- anyAttribute 요소 -- 사용자가 이 요소를 사용하여 스키마에 지정되어 있지 않은 속성이 있는 XML 문서를 확장할 수 있습니다. XML ODA가 사용자에게 이러한 속성의 이름을 넣으라고 프롬프트를 표시한 후, Business Object 정의에서 속성을 단순 속성에 맵핑합니다(253 페이지의 그림 51 참조). ODA는 anyAttribute 와일드 카드의 모든 namespace 및 processContents 속성을 무시합니다.
- 순서 그룹 -- 이 모델 그룹은 하위 요소가 특정 순서대로 나타나야 합니다. XML ODA는 순서 그룹에 있는 하위 요소를 Business Object 정의 내의 속성에 맵핑합니다. 이것이 XML type 속성으로부터 이 속성의 유형을 판별합니다.
- choice 그룹 -- 이 모델 그룹은 해당하는 준수 요소 선언 중 하나만이 인스턴스에 나타나야 함을 가리킵니다. XML ODA는 선택사항 그룹에 있는 하위 요소를 랩퍼 Business Object 정의 내의 속성에 맵핑합니다. 상위 Business Object 정의에는 유형이 랩퍼 Business Object인 다중 카디널리티를 가지는 단일 속성이 있습니다. 자세한 정보는 72 페이지의 『스키마 문서를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

- `all` 그룹 -- XML ODA는 `all` 그룹의 하위 요소를 랩퍼 Business Object 정의 내의 속성으로 맵핑합니다. 상위 Business Object 정의에는 유형이 랩퍼 Business Object인 다중 카디낼리티를 가지는 단일 속성이 있습니다. 자세한 정보는 72 페이지의 『스키마 문서를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.
- `include` 요소 - `include`를 사용하면, 스키마 문서에 포함된 스키마 문서에 있는 모든 정의 및 선언에 영향을 미칩니다. XML ODA는 포함될 스키마 문서로의 전체 경로를 제공할 것을 필요로 합니다. 이 전체 경로는 파일 시스템에서 포함된 스키마 문서의 실제 위치이며, URI가 *아닙니다*. 스키마 문서 위치를 XML ODA 등록 정보인 `FileName`으로 지정하면, 포함된 모든 스키마 문서가 동일한 위치에 있어야 합니다.

  주: 스키마 문서를 포함할 때 포함된 스키마 문서가 포함을 수행하는 스키마 문서와 동일한 이름을 가지는 글로벌 요소를 선언하지 *않는지* 확인하십시오.
- `import` 요소 -- XML 문서는 이 요소를 사용하여 다른 이름 공간의 구성요소를 참조할 수 있습니다. Business Object 정의에서 속성 응용프로그램 특정 정보에 `elem_ns` 또는 `attr_ns` 태그를 지정하여, 가져온 이름 공간을 식별할 수 있습니다. 자세한 정보는 77 페이지의 『스키마 이름 공간』을 참조하십시오.
- 복합 유형 파생 -- XML ODA는 제한 및 확장을 사용하여 파생된 복합 유형을 지원합니다. 이러한 구성의 경우, 파생된 내용 모델을 기반으로 파생된 복합 유형에 대해 Business Object를 작성합니다.
- 유형 대체 -- XML ODA 등록 정보 `TypeSubstitution`은 스키마에서 xsi:type 속성을 기반으로 하는 유형 대체를 지원합니다. 사용자가 `TypeSubstitution`을 ODA의 드롭 다운에서 True로 설정하면, XML ODA는 ASI ″typeSub=true″의 추가 랩퍼 오브젝트와 기본 유형 및 파생된 유형을 랩퍼 오브젝트의 하위 오브젝트로 표시하는 Business Object를 생성합니다. 자세한 정보는 74 페이지의 『스키마 문서를 기반으로 한 Business Object 정의의 유형 대체』를 참조하십시오.

지원되지 않는 스키마 문서 구조

XML ODA는 대부분의 스키마 문서를 처리할 수 있습니다. 그러나 다음 스키마 문서 구조는 지원하지 *않습니다*.

- 요소 대체 -- XML Data Handler는 런타임 시 요소 대체를 지원하지 *않습니다*. 스키마에서 요소 대체를 글로벌 요소 선언에 있는 `substitutionGroup` 속성으로 지정해야 합니다.
- 유형 및 그룹의 재정의 -- `redefine` 요소를 사용하면, 특정 작동만 지원되도록 유형(단순 또는 복합) 또는 그룹(요소 또는 속성)을 재정의할 수 있습니다.
- 기본 및 수정된 속성 -- XML ODA는 요소 및 요소 선언에서 기본 및 수정 속성을 무시합니다. 요소나 속성 선언은 이러한 속성 중 하나를 지정하고 인스턴스는 요소나 속성을 포함하지 *않으며*, XML ODA는 해당 Business Object 속성을 채우지 않습니다.

Business Object 정의 작성

XML 문서는 구조를 정의하는 DTD 또는 스키마 문서를 가질 수 있습니다. XML 문서에서 요소를 나타내는 Business Object 정의는 문서의 구조에 대한 정보를 포함해야 합니다. XML Data Handler에서 처리할 Business Object를 작성하려면, XML Data Handler가 처리할 각 XML 문서에 대한 구조 정보를 포함하는 Business Object 정의를 찾을 수 있어야 합니다. 다음 방법 중 하나로 XML 문서의 Business Object 정의를 생성할 수 있습니다.

- 『XML ODA를 사용하여 Business Object 정의 작성』
- 96 페이지의 『수동으로 Business Object 정의 작성』

이 기술 모두 Business Object Designer 도구를 사용하는 것과 관계가 있습니다. 이 섹션에서는 XML 문서를 위한 Business Object 정의를 생성하는 데 Business Object Designer를 사용하는 방법에 대한 개요를 제공합니다. Business Object Designer에 대한 전체적인 설명은 *Business Object Development Guide*를 참조하십시오.

주: XML Data Handler를 사용하는 일부 커넥터는 하위 오브젝트로 내용 Business Object를 포함하는 최상위 레벨 랩퍼 Business Object를 필요로 합니다. 사용 중인 커넥터에서 필요로 하는 Business Object 구조에 따라, 하위 오브젝트로 생성된 Business Object를 다른 Business Object에 추가해야 합니다. Business Object 구조에 대한 정보는 각 커넥터에 대한 문서를 참조하십시오.

XML ODA를 사용하여 Business Object 정의 작성

XML Object Discovery Agent(ODA)는 DTD 또는 스키마 문서를 기본으로 하여 XML 문서용 Business Object 정의를 작성합니다. ODA가 XML 문서 구조에 대한 정보를 얻기 위해 DTD 또는 스키마 문서를 조사합니다. 그런 다음, Business Object 정의를 비즈니스 통합 시스템에 로드할 수 있는 파일에 기록합니다.

주: XML 문서가 DTD 또는 스키마 문서를 가지지 않으면, 문서의 Business Object 정의를 수동으로 작성할 수 있습니다. 자세한 정보는 96 페이지의 『수동으로 Business Object 정의 작성』을 참조하십시오.

XML ODA는 XML Data Handler의 요구사항을 따르는 Business Object 정의를 빌드합니다. ODA는 모든 Business Object 정의에 필수 `ObjectEventId` 속성을 추가합니다. 또한 저장소 버전 번호를 지정할 경우, 이를 Business Object 파일의 맨 위에 추가합니다. 이 버전 번호는 Business Object 정의를 InterChange Server 비즈니스 통합 시스템에 가져오는 데 필요합니다. 이 Business Object 정의는 보통 추가로 편집하지 *않아*도 됩니다. 그러나 편집해야 하면, 255 페이지의 『Business Object 정의에서 정보 수정』을 참조하십시오.

XML ODA를 사용하는 방법에 대한 정보는 241 페이지의 『XML ODA 사용』을 참조하십시오. 이 부록에서는 XML ODA를 설치 및 구성하는 방법에 대해 설명합니다.

Business Object Designer에서 XML ODA를 사용하여 Business Object 정의를 생성하는 방법도 설명합니다. Business Object Designer 실행에 대한 정보는 *Business Object Development Guide*를 참조하십시오.

수동으로 Business Object 정의 작성

이 섹션에서는 수동으로 Business Object 정의를 작성하여 XML 문서를 표시하는 방법에 대해 설명합니다. 해당 속성 및 응용프로그램 특정 정보를 포함하여 Business Object 정의를 정확하게 정의했는지 확인해야 합니다.

주: XML 문서가 DTD 또는 스키마 문서를 가지지 않으면, 문서의 Business Object 정의를 수동으로 작성해야 합니다. DTD나 스키마 문서가 있으면, IBM에서는 Business Object 정의를 작성하는 데 XML ODA를 사용할 것을 권장합니다.

DTD 또는 스키마 문서용 XML 문서 형식에 대한 설명은 XML ODA가 빌드하는 Business Object 정의에 대해 설명합니다. 38 페이지의 표 11에서는 스키마를 설명하기 위해 해당 데이터 모델을 가지는 XML 문서의 형식에 대해 설명하는 이 매뉴얼의 절을 보여줍니다. 이 섹션에 설명된 대로 Business Object 정의는 XML Data Handler의 요구사항을 따릅니다. 그러므로 Business Object 정의를 수동으로 작성해야 할 때 이 설명을 따를 수 있습니다.

다음 단계에서, *ElementTypeName*은 Business Object 구성(속성 또는 Business Object)에 의해 표시되는 XML 요소 유형입니다. XML 문서를 기초로 Business Object를 정의하려면 다음을 수행하십시오.

1. 최상위 레벨 Business Object 정의를 작성하십시오. 이 Business Object 정의의 이름은 XML 문서의 최상위 레벨 요소(DTD 또는 스키마 문서의 이름)여야 하며 *BOPrefix_TopLevelName*의 형식을 갖추어야 합니다.

   주: 일부 커넥터에서는 하위 오브젝트로 내용 Business Object를 포함하는 랩퍼 Business Object를 필요로 합니다. 자세한 정보는 55 페이지의 『DTD를 기반으로 하는 랩퍼 Business Object 정의』를 참조하십시오.

2. 최상위 레벨 Business Object 정의에서 XML 요소의 속성을 작성하십시오.

   DTD 또는 스키마 문서를 기반으로 하는 최상위 레벨 Business Object 정의의 경우, 다음 속성이 필요합니다.

   - `XMLDeclaration` - 이 속성의 응용프로그램 특정 정보는 `type=pi`입니다.
   - DTD 또는 스키마 문서에서 루트 요소를 나타내는 속성 - 이 속성에는 단일 카디널리티 하위 Business Object가 있고 해당 응용프로그램 특정 정보는 `elem_name` 태그로 요소의 이름을 지정합니다.

   이 필수 속성에 대한 일반 정보는 42 페이지의 『Business Object 구조』를 참조하십시오. 이 문서는 DTD 및 스키마 문서를 기반으로 하는 최상위 레벨 Business

Object 정의의 구조에 대한 다음 정보도 제공합니다.

데이터 모델	자세한 정보
문서 유형 정의(DTD) 스키마 문서	51 페이지의 『DTD의 Business Object 구조』 67 페이지의 『스키마 문서의 필수 Business Object 정의』

3. 최상위 레벨 Business Object 정의의 하위 오브젝트인 루트 요소 Business Object 정의를 작성하십시오. 루트 XML 요소의 속성이 들어 있습니다. 이 Business Object 정의의 이름은 XML 문서의 루트 요소여야 하며 *BOPrefix_TopLevelName_RootElementName*의 형식을 갖추어야 합니다.
4. 루트 요소 Business Object 정의에서, 각 구성 요소에 대한 Business Object 속성을 작성하십시오. 다음에 유의하십시오.
   - Business Object 속성 이름은 XML 요소(또는 속성) 이름과 동일하지 않아도 됩니다. 응용프로그램 특정 정보는 요소(또는 속성) 이름을 지정하는 데 사용됩니다.
   - XML 속성은 Business Object 정의에 있는 첫 번째 속성이어야 합니다.
   - 유형 판별:
     - 문자열 유형은 요소 내용 또는 연관된 속성 목록 선언이 없는 카디널리티 1 요소입니다.
     - Business Object 유형은 카디널리티 n의 포함 요소이거나, 요소 내용 또는 연관된 속성 스펙(단수 또는 복수)이 있는 포함된 요소입니다.
   - 응용프로그램 특정 정보는 String 유형 속성, 혼합 유형 요소 또는 카디널리티가 n인 선택사항 목록 요소에 필요합니다.

     이 응용프로그램 특정 정보에 대한 일반 정보는 46 페이지의 『응용프로그램 특정 정보』의 내용을 참조하십시오. 이 문서는 DTD 및 스키마 문서를 기반으로 하는 Business Object 속성의 응용프로그램 특정 정보에 대한 다음 정보도 제공합니다.

데이터 모델	자세한 정보
문서 유형 정의(DTD)	54 페이지의 『DTD의 XML 구성요소에 대한 응용프로그램 특정 정보』
스키마 문서	76 페이지의 『스키마 문서의 XML 구성요소에 대한 응용프로그램 특정 정보』

주: 스키마 정의가 있는 XML 문서의 경우, 루트 요소 Business Object 정의는 스키마 위치에 대한 속성 또한 필요할 수 있습니다. 자세한 정보는 67 페이지의 『스키마 문서의 필수 Business Object 정의』를 참조하십시오.

5. 포함된 모든 요소에 대해 하위 Business Object 정의를 작성하십시오. 위에 나열된 규칙을 따르십시오.

Business Object를 XML 문서로 변환

Business Object를 XML 문서로 변환하려면, XML Data Handler는 순차적으로 Business Object 정의의 속성을 통해 루핑합니다. Business Object와 해당되는 하위 오브젝트에 속성이 표시되는 순서를 기초로 순환하여 XML을 생성합니다.

XML Data Handler는 다음과 같이 Business Object를 XML 문서로 처리합니다.

1. Data Handler는 XML 데이터를 포함할 문서를 작성합니다.
2. Data Handler는 최상위 레벨 Business Object 정의에서 응용프로그램 특정 정보를 조사하여 하위 Meta Object(이름이 Business Object 레벨 응용프로그램 특정 정보의 `cw_mo_label` 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 이러한 속성을 XML 문서에 포함하지 *않습니다*.
3. Data Handler는 Business Object 정의의 나머지 속성을 통해 루핑합니다.

   Data Handler는 다음 규칙을 사용하여 각 속성마다 XML을 생성합니다.

   - Data Handler는 해당 속성에 대한 응용프로그램 특정 정보에 `notag`가 표시되지 않으면, 단순 `String` 유형 속성마다 시작 및 끝 태그를 작성합니다. 시작 태그가 열려 있고 처리 중인 Business Object 속성이 XML 속성을 표시하지 않을 경우, XML Data Handler는 시작 태그를 닫습니다. (즉, 문자 ">"를 XML 문서에 추가합니다.)
   - 속성이 Business Object를 표시할 경우, XML Data Handler는 시작 태그를 열고, 하위 Business Object에서 속성을 검색하기 위해 순환 호출을 작성한 후 하위 Business Object의 속성에 대해 XML을 생성한 다음 요소의 끝 태그를 생성합니다. XML Data Handler는 *속성 레벨* 응용프로그램 특정 정보 `elem_name`을 요소 이름으로 사용합니다. 다중 카디널리티 속성의 경우, 배열의 각 Business Object 인스턴스에 대해 이 프로세스가 반복됩니다.
   - 속성이 응용프로그램 특정 정보 `xsiType`을 포함할 경우, XML Data Handler는 `xsi:type=ValueofXsiType` 형식으로 현재 요소에 대한 속성을 작성합니다. 예를 들어, Business Object 속성이 `xsiType=ShirtType`을 지정할 경우 해당하는 XML 속성은 `xsi:type="ShirtType"`입니다. 유형 대체에 관한 자세한 정보는 74 페이지의 『스키마 문서를 기반으로 한 Business Object 정의의 유형 대체』를 참조하십시오.
   - XML 마크업을 표시하는 속성의 경우, Data Handler는 속성 응용프로그램 특정 정보를 사용하며 99 페이지의 표 29에 표시된 대로 XML을 생성합니다.

표 29. XML 마크업을 표시하는 속성의 XML 출력

XML 엔티티 Business Object 속성이 표시하는 사항	XML 출력	예	응용프로그램 특정 정보
처리 명령어	<?*AttrValue*?>	<?xml version="1.0"?>	type=pi
DTD	<!*AttrValue*>	<!DOCTYPE CUSTOMER "customer.dtd">	type=doctype
요소	<*ElementName*>... </*ElementName*>	DTD를 기반으로 하는 XML 문서의 경우: <CUSTOMER>... </CUSTOMER> 스키마 문서를 기반으로 하는 XML 문서의 경우: <element name=CUSTOMER...> </element>	type=pcdata
XML 속성	AttrName= "*AttrValue*"	DTD를 기반으로 하는 XML 문서의 경우: Seqno="1" 스키마 문서를 기반으로 하는 XML 문서의 경우: <element name=CUSTOMER... Seqno="1"...> </element>	type=attribute
CDATA 절	<![CDATA[*AttrValue*]]>	<![CDATA [<HTML>Text</HTML>]]>	type=cdata
주석	<!--*CommentText* -->	<!--Customer information from source application A-->	type=comment
스키마 위치(대상 이름 공간이 있는)	<*elementName* xmlns="*URI_path*" xmlns:xsi= "http://www.w3.org/ 2001/XMLSchema- instance" xsi:schemaLocation= "*URI_for_schema schema_location*" ...	92 페이지의 그림 28 참조	type= xsischemalocation
스키마 위치(대상 이름 공간이 없는)	<*elementName* xmlns="*URI_path*" xmlns:xsi= "http://www.w3.org/ 2001/XMLSchema- instance" xsi:noNamespace SchemaLocation= "*schema_location*" ...	<order xmlns=″http://sampleDoc.org.ord″ xmlns:xsi=″http://www.w3.org/2001/XMLSchema-instance″ xsi:noNamespaceSchemaLocation= ″order.xsd″> ... /<order>	type=xsinoNSlocation

- 주어진 요소와 연관된 문자 데이터(DTD의 PCDATA)를 포함하는 속성의 경우, 마크업이 생성되지 않으며 속성 값만 문서에 추가됩니다(응용프로그램 특정 정보에 notag 태그가 존재하는 것으로 가정).
- 요소 또는 속성에 대한 문자 데이터가 있는 속성의 경우, Data Handler가 표 30에서처럼 모든 특수 문자를 적절한 이스케이프 문자열로 바꿉니다(응용프로그램 고유 정보에 escape=true 태그가 존재하는 것으로 가정).

표 30. 특수 문자 및 XML 표시

특수 문자	XML 이스케이프 문자열
ampersand (&)	&amp
less than (<)	&lt
greater than (>)	&gt

표 30. 특수 문자 및 XML 표시 (계속)

특수 문자	XML 이스케이프 문자열
single quote (‘)	&apos
double quote (“)	&quot

주: 값에 작은 따옴표(’), 큰 따옴표, &, < 또는 > 문자가 있는 XML 요소를 속성이 표시할 경우, 속성은 이스케이프 처리를 필요로 합니다. 해당 응용프로그램 특정 정보에서 속성에 escape=true 값이 없는 경우, 속성은 이스케이프 처리되지 않습니다. 이 응용프로그램 특정 정보는 다른 텍스트의 끝에 있어야 합니다.

- 다음 조건 중 하나라도 존재하면 속성을 건너뜁니다.
  - 속성에 CxIgnore 값이 있습니다.
  - 속성 이름은 Business Object 정의의 응용프로그램 고유 정보에 있는 cw_mo_ 태그에 나열됩니다.

  이러한 속성에 대해 XML이 생성되지 않습니다.
- CxBlank라는 값을 가진 단순 String 속성이 XML 문서에 빈 태그로 포함되어 있습니다. DTD를 기본으로 하는 XML 문서의 경우, CxBlank에 해당하는 PCDATA로 큰 따옴표(″ ″)가 사용됩니다. 그러나 Data Handler는 복잡한 속성(속성 유형이 Business Object)의 값이 CxBlank가 아니라고 가정합니다.

4. Data Handler가 변환을 완료하면, XML 문서를 호출자에게 리턴합니다.

주: Business Object의 verb 정보는 XML 문서로의 변환에서 유실됩니다. verb 보존에 대한 정보는 47 페이지의 『Business Object verb』를 참조하십시오

XML 문서에서 Business Object로 변환

이 섹션에서는 XML Data Handler가 XML 문서를 Business Object로 변환하는 방법에 대한 다음과 같은 정보를 제공합니다.

- 『XML 문서 요구사항』
- 101 페이지의 『직렬화된 데이터 처리』

XML 문서 요구사항

XML Data Handler는 XML 문서에 대해 다음 사항을 가정합니다.

- XML 문서는 제대로 양식을 갖추고 있습니다.
- XML 문서는 SAX Parser 요구사항을 준수합니다. XML Data Handler에서 사용하는 기본 구문 분석기는 XML 문서에 XML 선언이 포함되도록 요구합니다.
- XML 문서 구조는 해당 Business Object 구조와 일치해야 합니다. 또한 문서에서 요소 순서는 Business Object에서의 속성 순서와 일치해야 합니다.

직렬화된 데이터 처리

XML 문서를 Business Object로 변환하는 경우, XML Data Handler는 Business Object가 XML 문서 구조를 따르고 41 페이지의 『Business Object 정의의 요구사항』에서 설명된 대로 Business Object 정의 요구사항을 준수하는 것으로 간주합니다. Business Object 정의에 해당 요소 이름에 대한 속성이 없는 경우, XML Data Handler가 오류를 리턴합니다.

XML 문서를 Business Object로 변환하려면, XML Data Handler는 다음을 수행합니다.

1. 호출하는 커넥터가 Business Object를 변환 메소드에 전달하면, Data Handler는 이 Business Object를 사용하여 계속합니다. 호출자가 Business Object를 전달하지 않을 경우, Data Handler는 Business Object 이름을 판별하고 XML 문서의 데이터를 포함할 Business Object를 작성합니다.

   Business Object 이름을 판별하기 위해 Data Handler는 네임 핸들러를 호출합니다. 기본 네임 핸들러는 `BOPrefix` Meta Object 속성, 밑줄 및 루트 요소의 값을 결합하여 최상위 레벨 Business Object 이름을 형성합니다. 예를 들면, XML 문서에 `<!DOCTYPE Customer>`가 있고 `BOPrefix` 속성이 `MyApp`인 경우, 결과 이름은 `MyApp_Customer`입니다. 사용자 정의 네임 핸들러를 제공하여 다른 작동을 구성할 수 있습니다.

2. Data Handler는 `Parser` Meta Object의 값을 검색하여 XML 문서의 구문 분석에 사용할 SAX Parser를 결정합니다. 어떤 SAX Parser가 XML Data Handler를 사용하는지에 대한 정보는 39 페이지의 『SAX Parser』를 참조하십시오. Data Handler가 구문 분석기 이름을 판별할 때, 구문 분석기를 인스턴스화합니다.

3. Data Handler가 이벤트 핸들러(DTD를 기본으로 하는 XML 문서에서는 Entity Resolver도 등록)를 구문 분석기에 등록합니다. 이벤트 핸들러는 각각의 XML 요소 및 속성을 처리하는 콜백 메소드입니다.

   주: Entity Resolver는 DTD 문서에 있는 외부 엔티티 참조를 처리합니다. Data Handler가 올바른 클래스 이름을 가진 `EntityResolver` 옵션을 찾을 수 없는 경우, Data Handler는 `com.crossworlds.DataHandlers.xml.DefaultEntityResolver`를 사용합니다. 이 Entity Resolver는 모든 외부 참조를 무시합니다.

4. Data Handler는 구문 분석기를 호출하여 XML 문서를 구문 분석합니다.
   - Data Handler는 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 `cw_mo_`*`label`* 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 Business Object의 이러한 속성을 채우기 위해 처리를 수행하지 않습니다.

- 요소 유형에 따라, 구문 분석기의 이벤트 핸들러는 속성 등록 정보에 대한 Business Object 정의를 조회하고 요소 데이터를 처리합니다. 실행은 구문 분석기에서 심각한 오류가 발생하거나 XML 데이터의 요소가 Business Object 정의에 없을 경우에만 중지됩니다.

5. Business Object가 완료되면, Data Handler는 이 Business Object를 호출자에게 리턴합니다.

주: XML 문서에서 발견되는 모든 요소 및 속성의 경우, Data Handler는 Business Object에서 속성을 찾을 것을 예상합니다. Business Object 정의에 제공된 요소 또는 속성 이름에 대한 속성이 없을 경우, XML Data Handler는 오류를 리턴합니다. 이 규칙의 예외는 Business Object 정의에 필요하지 않은 FIXED 유형의 속성입니다. FIXED 속성이 Business Object 정의에 표시되지 않으면, XML 문서에서 FIXED 속성이 발견될 경우 실행이 중지되지 않습니다.

XML Data Handler 사용자 정의

다음을 수행하여 XML Data Handler를 사용자 정의할 수 있습니다.

- 『사용자 정의 XML 네임 핸들러 빌드』
- 104 페이지의『사용자 정의 Entity Resolver 빌드』

사용자 정의 XML 네임 핸들러 빌드

XML Data Handler는 네임 핸들러를 호출하여 XML 메시지로부터 Business Object의 이름을 추출합니다. XML Data Handler와 함께 포함되는 기본 네임 핸들러는 다음 태그를 찾습니다.

```
<!DOCTYPE Name>
```

Data Handler는 이 태그 및 `BOPrefix` Meta Object 속성에서 Business Object 이름을 생성합니다. EDI Data Handler는 Data Handler Meta Object에 저장된 `NameHandlerClass` 속성 값을 사용하여 호출할 네임 핸들러를 결정합니다. 네임 핸들러를 다른 방식으로 기능하도록 하려면, 다음을 수행해야 합니다.

1. `NameHandler` 클래스를 확장하여 사용자 정의 네임 핸들러를 작성합니다.
2. XML Data Handler용 Meta Object에 있는 `NameHandlerClass` 속성의 기본값을 갱신하여 사용자 정의 클래스를 사용하도록 XML Data Handler를 구성합니다.

다음 샘플 코드는 XML Data Handler에 대한 Custom Data Handler인 `CustomDataHandler`를 작성하도록 `DataHandler` 클래스를 확장합니다.

```
package com.crossworlds.DataHandlers.xml;

// DataHandler Dependencies
import com.crossworlds.DataHandlers.Exceptions.MalformedDataException;
import com.crossworlds.DataHandlers.NameHandler;
import com.crossworlds.DataHandlers.DataHandler;
```

```
// Java classes
import java.io.*;
import java.lang.Exception;

/*********************************************************
 * CustomNameHandler class. This class extends the Namehandler
 * class and implements method:
 *      getBOName( Reader serializedData, String subType )
 * The method getBOName contains the logic to extract the BOName
*********************************************************/
public class CustomNameHandler extends NameHandler
{

         /**
          * This method generates the business object name from
          * the data extracted from the 'serializedData' arg.
          * In this case, it is up to the caller to create
          * the BOName.
          */

          public String  getBOName( Reader serializedData, String subType )
            throws MalformedDataException
         {
           // The NameHandler uses DataHandler tracing. If the
           // DataHandler is not set, the NameHandler won't run.
            if (dh == null)
             return null;
            // Log a message
            dh.traceWrite(
            "Entering CustomNameHandler.getBOName for subtype '"
              + subType + "'.", 4);

            // This method parses the XML document and extracts the
           // business object name from the following tag in
           // the XML doc:
            //        <cml title=
            // For example, in:
            //        <cml title="cholestrol" id="cml_cholesterol">
            // the Business Object name is 'cholestrol'.

            // Log a message
            dh.traceWrite(
             "Name resolution will be done using <cml title= ",4);
            String name = null;

            try
           {
              // Read line of data from the Reader object
              LineNumberReader lineReader =
                new LineNumberReader( serializedData );
              serializedData.mark( 1000 );
              String line = lineReader.readLine();
              while ( line != null )
             {
                // search for <cml title= in the line
                int start = line.indexOf("<cml title=");
                if ( start != -1 )
               {
                  start += 12;
```

```
                    // search for the ending quotes for the tile tag
                    int end = line.indexOf('\"', start);

                    // extract name from line
                    name = line.substring(start, end);
                    break;
                  }
                   line = lineReader.readLine();
                 }

                  if ( name == null || name.length() == 0 )
                    throw new MalformedDataException(
                    "Error: can't determine the BusinessObject Name.");

               }

                catch(Exception e)
               {
                  throw new MalformedDataException( e.getMessage() );
               }
                serialized Data.reset();
                return name;
             }
}
```

사용자 정의 Entity Resolver 빌드

XML Data Handler에서 사용하는 SAX Parser는 Entity Resolver를 호출하여 XML 문서 내에서 외부 엔티티(참조된 DTD)를 찾습니다. XML Data Handler와 함께 포함된 Entity Resolver는 로컬 파일 시스템에서 외부 참조를 무시하거나 검색할 수 있습니다. 외부 엔티티를 찾기 위한 또다른 방법을 지정해야 할 경우, 사용자 정의 Entity Resolver 클래스를 작성해야 합니다.

XML Data Handler는 XML Data Handler Meta Object에 저장된 `EntityResolver` 속성 값을 사용하여 Entity Resolver를 결정합니다.

제 4 장 EDI Data Handler

EDI Data Handler라고 하는 IBM WebSphere Business Integration Data Handler for EDI(Electronic Data Interchange)는 Business Object를 EDI 문서로 변환하고 EDI 문서를 Business Object로 변환합니다. EDI Data Handler 설치에 관한 지시사항은 25 페이지의 『Data Handler 설치』를 참조하십시오.

이 장에서는 EDI Data Handler가 EDI 문서를 처리하는 방법과 Data Handler에서 처리할 Business Object를 정의하는 방법에 대해 설명합니다. 이 정보를 안내서로 사용하여 EDI Data Handler 요구사항을 따르는 Business Object를 구현할 수 있습니다. 또한 EDI Data Handler를 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 107 페이지의 『EDI Data Handler 구성』
- 111 페이지의 『EDI 문서의 Business Object 정의』
- 119 페이지의 『Business Object를 EDI 문서로 변환』
- 123 페이지의 『Business Object로 EDI 문서 변환』
- 132 페이지의 『EDI Data Handler 사용자 정의』

개요

EDI Data Handler는 1차 역할로서 Business Object를 EDI 문서로, 그리고 EDI 문서를 Business Object로 변환하는 데이터 변환 모듈입니다. EDI 문서는 비즈니스 정보 전달을 위한 표준화된 형식입니다. EDI Data Handler는 두 가지의 메시지 표준(X.12 및 EDIFACT)을 지원합니다.

EDI 문서는 `edi` MIME 유형의 직렬화된 데이터입니다. 기본 최상위 레벨 커넥터 Meta Object(`MO_DataHandler_Default`)는 `edi` MIME 유형을 지원합니다. 그러므로 `MO_DataHandler_Default` Data Handler Meta Object를 사용하도록 구성된 커넥터는 EDI Data Handler를 호출할 수 있습니다.

주의

EDI Data Handler는 단일 트랜잭션 유형(문서는 동일한 유형의 여러 트랜잭션을 포함할 수 있음)만 포함하는 단일 그룹이 있는 EDI 문서만 처리할 수 있습니다. 대부분의 EDI 환경은 여러 그룹 및 트랜잭션 유형을 가지고 있는 문서를 처리합니다. TPI용 IBM WebSphere Business Integration Adapter에는 여러 그룹 및 트랜잭션 유형을 가지고 있는 EDI 문서를 처리할 수 있도록 하는 분할 논리가 있습니다. 다른 어댑터에는 분할 논리가 없으므로, 기술적으로 EDI Data Handler를 사용할 수 있음에도 불구하고, TPI용 IBM WebSphere Business Integration Adapter만 EDI 문서에 복수 그룹이 있는 상황을 처리할 수 있습니다.

Data Handler는 EDI 문서에서 식별하는 문서 분리문자를 사용하여 문서 데이터의 구문을 분석합니다. Data Handler가 문서에서 분리문자를 식별할 수 없으면, EDI Data Handler와 연관된 하위 Meta Object에서 속성으로 지정한 분리문자 값을 사용합니다. EDI 하위 Meta Object에 대한 자세한 정보는 109 페이지의 『EDI Data Handler 하위 Meta Object 구성』을 참조하십시오.

EDI data-handler 구성요소

EDI Data Handler는 네임 핸들러를 사용하여 EDI 메시지에서 Business Object의 이름을 추출합니다. 그림 29에서는 EDI Data Handler 구성요소 및 해당 관계를 보여줍니다.

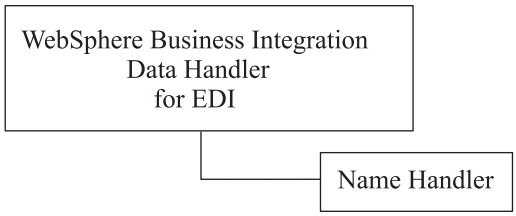

그림 29. EDI data-handler 구성요소

Data Handler는 EDI Data Handler 하위 Meta Object의 `NameHandlerClass` 속성 값에 기반으로 하는 네임 핸들러의 인스턴스를 호출합니다.

- 클래스 이름이 `NameHandlerClass` 속성에 제공되면, EDI Data Handler는 네임 핸들러를 사용하여 Business Object 이름을 판별합니다.

  제품 참조서에서 제공되는 Meta Object 버전에서 `NameHandlerClass` 속성은 `NameHandlerFile` 속성이 지정하는 네임 핸들러 찾아보기 파일에서 작성할 Business Object의 이름을 확보하는 기본 EDI 네임 핸들러를 말합니다.

- 클래스 이름을 제공하지 않을 경우, Data Handler는 오류를 로그하고 예외를 생성합니다.

사용자 정의 네임 핸들러 작성 방법에 대한 정보는 132 페이지의 『EDI Data Handler 사용자 정의』를 참조하십시오.

Business Object 및 EDI 문서 처리

EDI Data Handler는 표 31에 나열된 조작을 수행합니다.

표 31. EDI Data Handler의 데이터 조작

Data Handler 조작	**자세한 정보**
호출자로부터 Business Object를 수신하고 Business Object를 EDI 문서로 변환한 후 EDI 문서를 호출자에게 전달합니다.	119 페이지의 『Business Object를 EDI 문서로 변환』
호출자로부터 EDI 문서를 수신하고, Business Object를 빌드한 후 Business Object를 호출자에게 리턴합니다.	123 페이지의 『Business Object로 EDI 문서 변환』

EDI Data Handler 구성

커넥터와 함께 사용하기 위해 EDI Data Handler를 구성하려면 다음 단계를 수행하십시오.

- 트랜잭션 ID, DUNS 번호 및 Business Object 이름에 대한 EDI 네임 핸들러 찾아보기 파일을 작성하십시오.
- EDI 하위 Meta Object 속성에 대한 적절한 값을 입력하십시오.

이러한 단계는 각각 다음 섹션에 자세히 설명되어 있습니다.

주: EDI Data Handler를 사용하려면, Business Object 정의가 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정해야 합니다. 자세한 정보는 111 페이지의 『EDI 문서의 Business Object 정의』를 참조하십시오.

네임 핸들러 찾아보기 파일 작성

EDI Data Handler는 EDI 네임 핸들러 찾아보기 파일을 근거로 EDI 메시지에 따라 작성할 Business Object를 판별합니다. 이 네임 핸들러 찾아보기 파일에는 다음과 같이 탭으로 구분된 열이 있습니다.

- 트랜잭션 ID: 이 값은 EDI 문서 유형(예: 850)을 식별합니다. EDI 문서의 내용을 고유하게 식별하지는 않지만, 문서에 포함된 정보의 종류를 표시합니다.
- (선택적) 버전 번호: 이는 EDI Data Handler가 여러 버전의 EDI 문서를 관리하기 위해 사용하는 버전/릴리스/업계 ID 코드입니다.
- DUNS 번호: EDI Data Handler는 이 번호를 사용하여 거래업체를 고유하게 식별합니다.

- 연관된 Business Object의 이름: EDI Data Handler는 Business Object 이름을 사용하여 작성해야 최상위 레벨 EDI Business Object를 식별합니다.

주: 탭 문자를 사용하여 네임 핸들러 찾아보기 파일의 필드를 구분하십시오.

선택적 버전 번호가 없는 네임 핸들러 찾아보기 파일의 샘플은 다음과 같습니다.

```
850             123465      X12_850A_Order
850             122227      X12_850B_Order

855             122227      X12_855A_Response
855             123465      X12_855A_Response
```

버전 번호가 있는 네임 핸들러 찾아보기 파일 샘플은 다음과 같습니다. (버전 번호는 두 번째 열에 표시됩니다. 이 예에서 버전 번호는 004010입니다.)

```
850             004010    111111      X12_850A_Order
855             004010    122227      X12_855A_Response
```

주: 이 예에서, "X12_"는 최상위 레벨 Business Object 이름에 대한 공통 접두부입니다. 이 접두부는 반드시 지정하지 *않아*도 됩니다. 최상위 레벨 비즈니스를 작성할 때 식별하는 접두부를 선택합니다. 자세한 정보는 112 페이지의 『최상위 레벨 EDI Business Object』를 참조하십시오.

EDI Data Handler에 작성하는 Business Object에 대한 정보를 제공하려면, 다음을 수행해야 합니다.

- Data Handler가 Business Object를 작성할 DUNS 번호 및 트랜잭션 ID(및 선택적으로 버전 번호) 조합마다 네임 핸들러 찾아보기 파일에 항목이 있는지 확인해야 합니다. 열 값이 탭 문자로 구분되어 있는지 확인해야 합니다.
- NameHandlerFile Meta Object 속성을 네임 핸들러 찾아보기 파일의 완전한 경로 이름으로 설정해야 합니다.

주: 하위 Meta Object의 제공된 버전에서 NameHandlerFile 속성의 Default Value 등록 정보에는 값이 있습니다. 그러나 NameHandlerFile 속성에 있는 경로 이름은 EDI 네임 핸들러 찾아보기 파일의 이름을 지정하는지 확인해야 합니다. Windows 시스템에서 경로를 지정할 경우, 두 번째 백슬래시(\)를 포함하여 모든 백슬래시 문자를 이스케이프해야 합니다. 예를 들면, 다음과 같습니다.

```
c:\\home\\DataHandlers\\edi\\edi_xref
```

UNIX 경로 이름은 백슬래시를 사용하지 않으므로 이스케이프하지 않아도 됩니다.

```
/home/DataHandlers/edi/edi_xref
```

EDI Data Handler는 파일을 갱신할 때마다 이 파일에서 정보를 새로 고칩니다. 그러므로 즉시 새 값이나 변경된 값을 선택하면 구성요소를 다시 시작하지 않아도 됩니다.

EDI Data Handler 하위 Meta Object 구성

EDI Data Handler를 구성하려면, EDI Data Handler의 하위 Meta Object에서 구성 정보가 제공되는지 확인해야 합니다. EDI Data Handler의 경우, IBM은 기본 하위 Meta Object MO_DataHandler_DefaultEDIConfig를 제공합니다. 이 Meta Object에 있는 각각의 속성이 EDI Data Handler의 구성 등록 정보를 정의합니다. 표 32에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 32. EDI Data Handler의 하위 Meta Object 속성

속성 이름	설명	제공되는 기본값
ClassName	지정된 MIME 유형으로 사용하기 위해 로드할 Data Handler 클래스의 이름. 최상위 레벨 Data Handler Meta Object에는 이름이 지정된 MIME 유형과 일치하고 유형이 EDI 하위 Meta Object인 속성이 있습니다(표 32 참조).	com.crossworlds.DataHandlers.edi.edi
DefaultVerb	EDI 문서에서 Business Object로 변환할 때 Business Object에서 설정할 verb의 이름. 이 속성에 대해 값이 존재하지 않을 경우, EDI Data Handler는 Business Object에 verb를 포함하지 않습니다.	Create
DummyKey	키 속성. Data Handler에서 사용되지는 않지만 비즈니스 통합 시스템에 필요합니다.	1
ISA (X.12 표준) UNA 및 UNB (EDIFACT 표준)	EDI Data Handler가 EDI 문서 자체로부터 분리문자의 값을 확보할 수 있도록 분리문자에 대한 위치 정보를 제공합니다. 이 속성의 이름은 다음과 같이 EDI 문서의 첫 번째 세그먼트 이름에 해당해야 합니다. • EDI 메시지가 X.12 표준을 따를 경우, EDI 문서는 세그먼트 ISA에서 시작합니다. 이 위치 정보 속성의 이름은 ISA입니다. • EDI 메시지가 EDIFACT 표준을 따를 경우, EDI 문서는 UNA 서비스 문자열 어드바이스(선택적)와 초기 세그먼트 UNB에서 시작합니다. 그러므로 이 Meta Object에서 두 가지의 위치 정보 속성인 UNA 및 UNB를 작성해야 합니다. 이 Meta Object 속성의 값에 대한 정보는 125 페이지를 참조하십시오.	없음
NameHandlerClass	EDI 문서 내용에서 Business Object 이름을 판별하기 위해 사용할 클래스의 이름. 고유의 사용자 정의 네임 핸들러를 작성할 경우, 이 속성의 Default Value 등록 정보를 변경하십시오. 자세한 정보는 132 페이지의 『EDI Data Handler 사용자 정의』를 참조하십시오.	com.crossworlds.DataHandlers.edi.EdiNameHandler
NameHandlerFile	트랜잭션 ID, 선택적 버전 번호, DUNS 번호 및 Business Object 이름에 대한 네임 핸들러 찾아보기 파일을 포함하는 EDI 네임 핸들러 찾아보기 파일의 완전한 이름. 자세한 정보는 107 페이지의 『네임 핸들러 찾아보기 파일 작성』을 참조하십시오.	Windows systems: C:\\crossworlds\ edi\dbfile.txtUNIX systems: /home/crossworlds/ edi/dbfile.txt
RELEASE_CHAR	속성 값에서 이스케이프 문자로 사용할 문자. 이 이스케이프 문자는 EDI 문서 분리문자가 속성의 실제 값의 일부인 경우에 필요합니다. 실제 값에서 문자 앞에 이 이스케이프 문자를 붙여야 합니다. 예를 들어, 속성 값이 "*dog?"이고 요소 분리문자가 별표일 경우, "?*dog??".	? (물음표)
SEPARATOR_ELEMENT	EDI 문서에서 요소 분리문자로 사용하는 문자	*(별표)

표 32. EDI Data Handler의 하위 Meta Object 속성 (계속)

속성 이름	설명	제공되는 기본값
SEPARATOR_COMPOSIT	EDI 문서에서 복합 분리문자로 사용하는 문자	,(쉼표)
SEPARATOR_REPEAT	EDI 문서에서 반복 분리문자로 사용하는 문자. 반복하는 복합 요소를 구분하는 데 사용됩니다.	^(탈자 부호)
SEPARATOR_SEGMENT	EDI 문서에서 세그먼트 분리문자로 사용하는 문자. 세그먼트 분리문자를 줄 바꾸기 문자로 설정하려면, 다음과 같이 문자를 이스케이프해야 합니다. • Windows 시스템 – X12 문서: \r\n – EDIFACT 문서: \n • UNIX 시스템: \n	~(틸데)
ObjectEventId	키 속성. Data Handler에서 사용되지는 않지만 비즈니스 통합 시스템에 필요합니다.	없음

표 32에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 모든 속성의 Default Value 등록 정보를 적절한 값으로 설정해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

EDI Data Handler의 여러 구성을 호출하려면 다음 단계를 수행하십시오.

- 기본 EDI 하위 Meta Object(X.12 표준에서 EDI 문서에 대한 EDI Data Handler 구성)를 복사하고 이름을 바꾸십시오. 새 하위 Meta Object의 이름을 지정하기 위한 권장 방법은 MIME 유형에 부속 유형을 제공하는 것입니다. 예를 들어, 기본 EDI 하위 Meta Object의 이름을 MO_DataHandler_DefaultEDI_X12Config로 바꾸고 사본의 이름으로 MO_DataHandler_DefaultEDI_EDIFACTConfig를 지정할 수 있습니다.
- 각 EDI 하위 Meta Object에서 속성의 기본값을 설정하여 Data Handler 인스턴스를 구성하십시오.
- 최상위 레벨 Data Handler Meta Object edi_*subtype*에서 속성을 작성하십시오. 여기서 *subtype*은 EDI 표준 중 하나가 될 수 있습니다. X.12 또는 EDIFACT 표준에서 EDI 문서를 처리하려면, 최상위 레벨 Meta Object에서 두 개의 속성(edi_x12 및 edi_edifact)을 작성할 수 있습니다. 이러한 속성 각각은 연관된 하위 Meta Object를 나타냅니다.

Data Handler 구성 방법에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

EDI 문서의 Business Object 정의

EDI Data Handler를 사용하려면, Data Handler에서 필요로 하는 메타 데이터를 포함하고 EDI 메시지의 해당하는 필드를 포함하도록 Business Object 정의를 작성하거나 수정해야 합니다. 이 섹션에서는 EDI Data Handler에 대해 작동하도록 Business Object 정의를 작성하기 위해 필요한 정보를 제공합니다. 특히, 다음 정보를 제공합니다.

- 『EDI Business Object 구조 이해』
- 118 페이지의 『EDI 문서의 Business Object 정의 작성』

EDI Business Object 구조 이해

EDI Data Handler는 Business Object 또는 EDI 문서를 변환할 때 Business Object 정의를 사용합니다. Business Object 구조 및 해당되는 응용프로그램 특정 정보를 사용하여 변환을 수행합니다. 그림 30에서는 EDI 메시지를 표시하는 Business Object 구조를 보여줍니다.

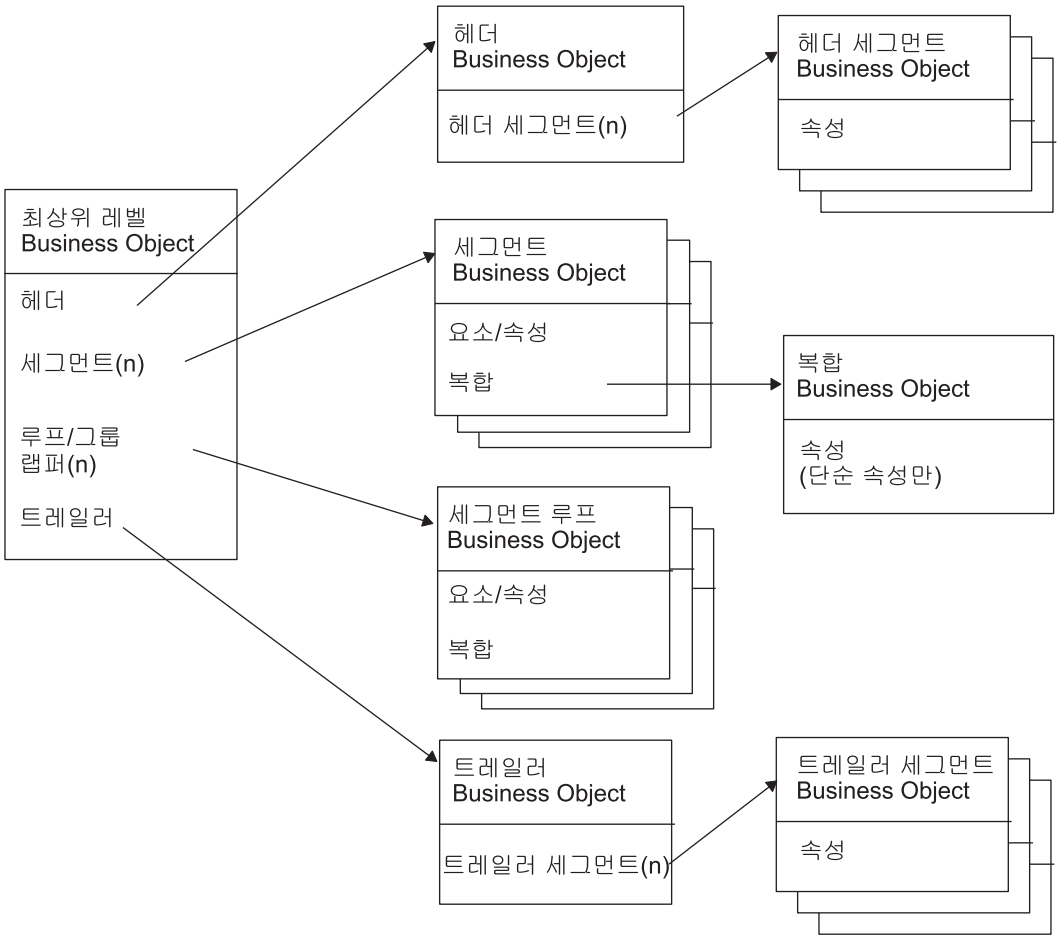

그림 30. EDI 메시지의 Business Object 구조

Business Object 정의가 EDI Data Handler의 요구사항을 따르도록 하려면, 다음 Business Object 각각에 대한 지침을 사용하십시오.

- 『최상위 레벨 EDI Business Object』
- 115 페이지의 『머리글 Business Object』
- 115 페이지의 『세그먼트 Business Object』
- 117 페이지의 『복합 Business Object』
- 117 페이지의 『세그먼트 루프 Business Object』
- 118 페이지의 『트레일러 Business Object』

최상위 레벨 EDI Business Object

EDI Data Handler는 최상위 레벨 Business Object가 EDI 메시지에 대한 정보를 보유할 것으로 예상합니다. 표 33에서는 EDI Data Handler가 Business Object의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다.

표 33. EDI 최상위 레벨 Business Object 정의 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 이 Business Object 정의는 표준 접두부로 시작하는 것이 좋습니다. 최상위 레벨 Business Object의 이름은 다음과 같이 메시지 표준에 따라 결정됩니다. • EDIFACT 표준을 따르는 EDI 메시지의 경우, Business Object 이름의 양식은 다음과 같습니다. *BusObj 접두부 + 메시지 유형* • X.12 표준을 따르는 EDI 메시지의 경우, Business Object 이름 양식은 다음과 같습니다. *BusObj Prefix + Transaction Set Identifier Code*
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

주: Data Handler가 EDI 문서를 Business Object로 변환할 때, 네임 핸들러 찾아보기 테이블을 통해 EDI 문서에 대한 최상위 레벨 Business Object를 식별합니다. 자세한 정보는 107 페이지의 『네임 핸들러 찾아보기 파일 작성』을 참조하십시오.

그림 30에 표시된 대로, 최상위 레벨 Business Object 정의에는 다음과 같은 속성이 있습니다.

- EDI 문서 머리글을 표시하는 속성
- 세그먼트를 표시하기 위해 필요한 만큼 다수의 속성
- 세그먼트 루프를 표시하기 위해 필요한 만큼 다수의 속성

• EDI 문서 트레일러를 표시하는 속성

머리글 속성: 최상위 레벨 EDI Business Object의 머리글 속성은 머리글 정보를 포함하는 단일 카디널리티 배열을 표시합니다. 이 속성에 대한 응용프로그램 특정 정보에는 다음 태그가 포함되어야 합니다.

```
type=header
```

이 속성의 Type 등록 정보에는 머리글 Business Object의 이름이 있습니다. 머리글 Business Object의 속성에 대한 정보는 115 페이지의 『머리글 Business Object』를 참조하십시오.

세그먼트 속성: 최상위 레벨 EDI Business Object의 각 세그먼트 속성은 세그먼트 정보를 포함하는 단일 카디널리티 배열을 표시합니다. 표 34에서는 최상위 레벨 Business Object 정의에서 세그먼트 속성에 대한 속성 등록 정보를 보여줍니다.

표 34. EDI 최상위 레벨 Business Object 정의에서 세그먼트 속성에 대한 속성 등록 정보

등록 정보 이름	설명
Name	세그먼트 속성 이름의 양식은 다음과 같습니다. *태그 + 위치* (중복된 경우)
Type	세그먼트 속성 유형의 이름 양식은 다음과 같습니다. *TopLevelBusObj + 태그* 이 속성 등록 정보에는 적절한 세그먼트 Business Object의 이름이 포함됩니다.
ContainedObjectVersion	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Relationship	"containment"로 설정합니다.
Cardinality	Max Use 또는 Repetition을 EDI 문서 스펙에서 "1"로 설정할 경우, 이 속성 등록 정보의 값을 "1로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "N"으로 설정하십시오.
MaxLength	항상 "1"로 설정합니다.
Key	항상 "false"로 설정합니다.
Foreign key	항상 "false"로 설정합니다.
Required	Status 또는 Option을 EDI 문서 스펙에서 "M"으로 설정할 경우, 이 속성 등록 정보를 "true"로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "false"로 설정하십시오.
Default value	EDI Data Handler에서는 사용하지 않습니다.
Application-specific information	name= *name of segment* 로 설정합니다.

주: Max Use, Repetition, Status, Option 필드는 EDI 문서 스펙의 일부입니다. 자세한 정보는 EDI 문서를 참조하십시오.

세그먼트 Business Object의 속성에 대한 정보는 115 페이지의 『세그먼트 Business Object』를 참조하십시오.

세그먼트 루프 속성: 각 세그먼트 루프 속성은 세그먼트 정보를 포함하는 다중 카디널리티 배열을 나타냅니다. 표 35에서는 최상위 레벨 Business Object 정의의 세그먼트 루프 속성에 대한 속성 등록 정보를 보여줍니다.

표 35. EDI 최상위 레벨 Business Object 정의의 세그먼트 루프 속성에 대한 속성 등록 정보

등록 정보 이름	설명
Name	세그먼트 루프 속성의 이름 양식은 다음과 같습니다. *태그 + 위치* (중복된 경우)
Type	세그먼트 루프 속성 유형의 이름 양식은 다음과 같습니다. *TopLevelBusObj + 루프/그룹 키워드 + 태그* 이 속성 등록 정보에는 적절한 세그먼트 루프 Business Object의 이름이 포함됩니다. 이 이름은 세그먼트의 목적을 표시하기 위한 키워드(예: Loop)를 포함할 수 있습니다.
ContainedObjectVersion	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Relationship	"containment"로 설정합니다.
Cardinality	항상 "N"으로 설정합니다.
MaxLength	항상 "1"로 설정합니다.
Key	항상 "false"로 설정합니다.
Foreign key	항상 "false"로 설정합니다.
Required	Status 또는 Option을 EDI 문서 스펙에서 "M"으로 설정할 경우, 이 속성 등록 정보를 "true"로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "false"로 설정하십시오. 이 속성 등록 정보를 루프/그룹의 첫 번째 세그먼트에서 "true"로만 설정하십시오.
Default value	EDI Data Handler에서는 사용하지 않습니다.
Application-specific information	다음을 포함합니다. • name=*name of first segment in loop* • type=loop 예를 들어, 다음의 응용프로그램 특정 정보는 첫 번째 세그먼트 이름이 AMT: AppSpecificInfo=name=AMT;type=loop 인 세그먼트 루프를 식별합니다.

주: Status 및 Option 필드는 EDI 문서 스펙의 일부입니다. 자세한 정보는 EDI 문서를 참조하십시오.

세그먼트 루프 Business Object의 속성에 대한 정보는 117 페이지의 『세그먼트 루프 Business Object』를 참조하십시오.

트레일러 속성: 최상위 레벨 EDI Business Object의 트레일러 속성은 트레일러 정보를 포함하는 단일 카디널리티 배열을 나타냅니다. 이 속성에 대한 응용프로그램 특정 정보에는 다음 태그가 포함되어야 합니다.

```
type=trailer
```

이 속성의 Type 등록 정보에는 트레일러 Business Object의 이름이 있습니다. 트레일러 Business Object의 속성에 대한 정보는 118 페이지의『트레일러 Business Object』를 참조하십시오.

머리글 Business Object

EDI 메시지에 대한 머리글 정보를 보유할 경우, EDI Data Handler는 최상위 레벨 Business Object의 첫 번째 속성으로 머리글 Business Object를 예상합니다. 표 36에서는 EDI Data Handler가 Business Object 정의의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다.

표 36. EDI 머리글 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 머리글 Business Object의 이름에는 Business Object 접두부가 포함되도록 하는 것이 좋습니다. 식별 정보(예: 키워드 "header")를 포함할 수도 있습니다.
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

Business Object 정의에는 머리글의 각 머리글 세그먼트를 표시하기 위한 속성이 있습니다. 각 속성에 대한 응용프로그램 특정 정보는 머리글 세그먼트의 이름을 식별합니다. 각 머리글 세그먼트는 단순, 단일 카디널리티 또는 다중 카디널리티 속성을 포함할 수 있습니다.

세그먼트 Business Object

EDI 메시지에 대한 세그먼트 정보를 갖고 있는 경우, EDI Data Handler는 세그먼트 Business Object를 예상합니다. 표 37에서는 EDI Data Handler가 Business Object 정의의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다

표 37. EDI 세그먼트 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 세그먼트 Business Object 정의 이름은 다음과 같은 양식을 사용하는 것이 좋습니다. *BusObj 접두부* + *태그*
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

그림 30에 표시된 대로, 세그먼트 Business Object에는 다음 속성이 포함될 수 있습니다.

- EDI 요소를 표시하기 위한 단순(String) 속성

- 복합 요소를 표시하기 위한 배열 속성

단순 속성: 세그먼트 Business Object의 각 단순 속성은 표 38에 표시된 속성 등록 정보를 가지고 있어야 합니다.

표 38. 단순 속성의 속성 등록 정보

등록 정보 이름	설명
Name	각 Business Object 속성에는 고유한 이름이 있어야 합니다.
Type	각 단순 Business Object 속성 유형은 String이어야 합니다.
Cardinality	항상 "1"로 설정합니다.
Key	단순 속성에만 사용되며, Business Object의 *첫 번째* 문자열 속성에 대해 설정해야 합니다.
MaxLength	이 String 속성의 최대 *크기*를 설정합니다. EDI 문서 내에서, 실제 데이터의 일부로 분리문자를 임베드할 경우
Foreign key	항상 "false"로 설정합니다.
Required	Status 또는 Option을 EDI 문서 스펙에서 "M"으로 설정할 경우, 이 속성 등록 정보를 "true"로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "false"로 설정하십시오.
Default value	EDI Data Handler에서는 사용하지 않습니다.

주: Repetition, Status, Option 필드는 EDI 문서 스펙의 일부입니다. 자세한 정보는 EDI 문서를 참조하십시오.

복합 속성: 각 복합 Business Object는 EDI 복합 요소를 포함하는 배열입니다. 표 39에서는 복합 속성에 대한 속성 등록 정보를 보여줍니다.

표 39. EDI 세그먼트 Business Object 정의의 복합 속성에 대한 속성 등록 정보

등록 정보 이름	설명
Name	복합 속성의 이름 양식은 *Tag + Position*입니다(중복될 경우).
Type	세그먼트 속성 유형의 이름 양식은 다음과 같습니다. *BusObj 접두부 + 태그* 이 속성 등록 정보에는 적절한 복합 Business Object의 이름이 포함됩니다.
ContainedObjectVersion	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Relationship	"containment"로 설정합니다.
Cardinality	Repetition이 1로 설정되는 경우, 이 속성 등록 정보의 값을 "1"로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "N"으로 설정하십시오.
MaxLength	항상 "1"로 설정합니다.
Key	항상 "false"로 설정합니다.
Foreign key	항상 "false"로 설정합니다.
Required	Status 또는 Option을 "M"으로 설정할 경우, 이 속성 등록 정보를 "true"로 설정하십시오. 그렇지 않으면, 이 속성 등록 정보를 "false"로 설정하십시오.
Default value	EDI Data Handler에서는 사용하지 않습니다.
Application-specific information	*없음*
Required Server Bound	항상 "false"로 설정합니다.

주: Repetition, Status, Option 필드는 EDI 문서 스펙의 일부입니다. 자세한 정보는 EDI 문서를 참조하십시오.

자세한 정보는 『복합 Business Object』를 참조하십시오.

복합 Business Object

EDI 메시지에 요소에 대한 복합 정보를 보유할 경우, EDI Data Handler는 복합 Business Object를 예상합니다.

주: 복합은 보통 EDIFACT 표준을 따르는 EDI 문서에서 발견됩니다. 그러나 X.12 표준을 따르는 문서에서도 존재할 수 있습니다.

표 40에서는 EDI Data Handler가 Business Object 정의의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다.

표 40. EDI 복합 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 복합 Business Object 정의 이름은 다음과 같은 양식을 사용하는 것이 좋습니다. *BusObj 접두부 + 태그*
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

복합 Business Object는 단순(String) 속성 또는 배열을 포함할 수 있습니다.

세그먼트 루프 Business Object

EDI 메시지에 세그먼트 루프 또는 그룹에 대한 정보를 보유할 경우, EDI Data Handler는 세그먼트 루프 Business Object를 예상합니다. 표 41에서는 EDI Data Handler가 Business Object 정의의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다.

표 41. EDI 세그먼트 루프 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 세그먼트 루프 Business Object 정의 이름은 다음과 같은 양식을 사용하는 것이 좋습니다. *BusObj 접두부 + 태그*
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

트레일러 Business Object

EDI 메시지에 대한 트레일러 정보를 보유할 경우, EDI Data Handler는 트레일러 Business Object를 예상합니다. 표 42에서는 EDI Data Handler가 Business Object 정의의 등록 정보를 해석하는 방법과 EDI Data Handler에 대해 사용하기 위해 Business Object를 수정할 때 등록 정보를 설정하는 방법에 대해 설명합니다.

표 42. EDI 트레일러 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 트레일러 Business Object의 이름에는 Business Object 접두부가 포함되도록 하는 것이 좋습니다. 식별 정보(예: 키워드 "trailer")를 포함할 수도 있습니다.
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	type 태그를 포함하는 메타 데이터는 없습니다. 변환 프로세스 중 무시할 속성을 표시하는 cw_mo 태그를 포함하는 메타 데이터가 있을 수도 있습니다.

Business Object 정의에는 트레일러의 각 트레일러 세그먼트를 표시하기 위한 속성이 있습니다. 각 속성에 대한 응용프로그램 특정 정보는 트레일러 세그먼트의 이름을 식별합니다. 각 트레일러 세그먼트는 단순, 단일 카디널리티 또는 다중 카디널리티 속성을 포함할 수 있습니다.

EDI 문서의 Business Object 정의 작성

EDI 문서의 Business Object 정의를 작성하는 방법은 두 가지입니다.

- Edifec SpecBuilder를 사용하여 정의 EDI 문서를 Business Object 정의로 내보낼 수 있습니다.
- 문서에 대해 Business Object 정의를 수동으로 작성할 수 있습니다.

SpecBuilder를 사용하여 Business Object 정의 작성

SpecBuilder는 오브젝트 발견 유틸리티로 기능할 수 있으며, EDI 문서를 기반으로 Business Object 정의를 작성합니다. 그런 다음, 비즈니스 통합 시스템에 로드할 수 있는 Business Object 정의 파일에 이를 기록합니다. SpecBuilder는 Edifecs Inc.에서 발표하고 지원하는 제 3 자 응용프로그램입니다. 지원에 대해서는 SpecBuilder 문서 또는 Edifecs 웹 사이트를 참조하십시오.

주: IBM은 SpecBuilder 도구를 릴리스의 일부로 포함하지 않습니다. 그러나 Edifecs CD에서 사용할 수는 있습니다. Edifecs CD 사본을 구하려면, IBM 담당자 또는 기술 지원에 문의하십시오.

수동으로 Business Object 정의 작성

이 섹션에서는 수동으로 Business Object 정의를 작성하여 EDI 문서를 표시하는 방법에 대해 설명합니다. Business Object Designer를 사용하여 필요에 따라 Business Object 정의에서 속성을 추가 또는 삭제하십시오.

주: EDI 문서의 구조는 꽤 복잡할 수 있습니다. 가능하면 SpecBuilder를 사용하여 많은 Business Object 정의를 빌드하는 것이 좋습니다.

EDI 문서를 기본으로 Business Object를 정의하려면 다음을 수행하십시오.

1. 최상위 레벨 Business Object 정의를 작성하십시오.

   최상위 레벨 Business Object의 구조에 대한 정보는 112 페이지의 『최상위 레벨 EDI Business Object』를 참조하십시오.

2. 최상위 레벨 Business Object에 대한 하위 Business Object를 작성하십시오. 최상위 레벨 Business Object에서, 표 43에 표시된 Business Object에 대한 하위 오브젝트 속성을 작성하십시오.

표 43. EDI Data Handler의 Business Object

EDI 문서의 부분	참고	Business Object
머리글	이 머리글에는 머리글 세그먼트가 포함될 수도 있습니다.	115 페이지의 『머리글 Business Object』
세그먼트	세그먼트에는 복합 요소가 포함될 수도 있습니다.	115 페이지의 『세그먼트 Business Object』, 117 페이지의 『복합 Business Object』
세그먼트 루프 및 그룹	세그먼트 루프는 반복되는 세그먼트로 구성됩니다.	117 페이지의 『세그먼트 루프 Business Object』
트레일러	이 트레일러에는 트레일러 세그먼트가 포함될 수도 있습니다.	118 페이지의 『트레일러 Business Object』

다음에 유의하십시오.

- Business Object 속성 이름은 EDI 요소 이름과 동일할 필요는 없습니다. 응용프로그램 고유 정보는 요소 이름을 지정하기 위해 사용됩니다.
- 유형 판별: `String`은 요소 내용이 없고 연관된 속성 목록 선언이 없는 카디널리티 1의 포함된 요소입니다. `BusinessObject`는 카디널리티 `n`의 포함된 요소이거나, 요소 내용이 있거나 연관된 속성 스펙이 있는 포함된 요소입니다.
- 응용프로그램 고유 정보는 많은 속성에 대해 필수입니다. 자세한 정보는 112 페이지의 『최상위 레벨 EDI Business Object』를 참조하십시오.

3. 각 단순 요소마다 Business Object 속성을 작성하십시오. 자세한 정보는 116 페이지의 『단순 속성』을 참조하십시오.
4. 중첩된 Business Object(예: 머리글 세그먼트, 트레일러 세그먼트 및 복합 요소)에 대해 하위 Business Object를 작성하십시오. 위에 나열된 규칙을 따르십시오.

Business Object를 EDI 문서로 변환

Business Object를 EDI 문서로 변환하기 위해, EDI Data Handler는 최상위 레벨 Business Object 정의의 속성을 루핑합니다. 최상위 레벨 Business Object에 표시되는 순서대로, 순환하여 속성을 처리하고 속성 값을 EDI 문서의 요소로 기록합니다.

EDI Data Handler는 다음과 같이 Business Object를 EDI 문서로 처리합니다.

1. Data Handler는 하위 Meta Object의 구성 정보를 기초로 복합, 요소, 세그먼트 및 반복 분리문자를 설정하여 자체를 초기화합니다. 구성 옵션 중 하나에 대해 값을 제공하지 않으면, Data Handler는 표 44 및 표 47에서처럼 하드 코딩된 기본값을 사용합니다.

표 44. 요소 및 세그먼트 분리문자의 기본값

우선순위	단계	요소 분리문자	세그먼트 분리문자
1	해당 Meta Object 속성의 값을 확보합니다.	SEPARATOR_ELEMENT	SEPARATOR_SEGMENT
2	연관된 Meta Object 속성을 설정하지 않을 경우, 하드 코딩된 기본값을 사용합니다.	더하기 부호(+)	작은 따옴표(')

2. Data Handler는 최상위 레벨 Business Object 정의에서 응용프로그램 특정 정보를 조사하여 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 cw_mo_ 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 이러한 속성을 XML 문서에 포함하지 *않습니다*. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
3. Data Handler는 최상위 레벨 Business Object 정의에서 나머지 속성을 루핑합니다. 각 속성의 카디낼리티에 따라, Data Handler는 속성이 표시하는 EDI 문서 부분을 판별합니다. 자세한 정보는 121 페이지의 『속성과 연관된 EDI 데이터 판별』을 참조하십시오.
4. Data Handler가 연관된 EDI 데이터를 식별하고 나면, 적절한 단계를 수행하여 속성 데이터를 EDI 문서에 기록합니다.
   - 속성이 세그먼트를 표시할 경우, Data Handler는 세그먼트가 널인지 확인합니다. Business Object가 널인 경우 Data Handler는 속성을 건너뛰고, Business Object가 널이 *아니면* Data Handler는 세그먼트 처리 단계를 수행합니다. 자세한 정보는 122 페이지의 『세그먼트 처리』를 참조하십시오.
   - 세그먼트에서 복합 요소를 표시하는 하위 Business Object 각각의 경우, Data Handler는 속성(String이어야 하는 모든 속성)을 루핑하고 복합 처리 단계를 수행합니다. 자세한 정보는 122 페이지의 『복합 처리』를 참조하십시오.
5. Data Handler는 문서에 기록한 총 세그먼트 수를 문서의 "세그먼트 수" 필드에 기록합니다. Data Handler는 하위 Meta Object의 ISA 속성에 있는 seg_count 태그로부터 이 필드의 위치를 판별합니다. 이 필드는 종종 SE 세그먼트에 위치합니다. seg_count 태그 설정에 대한 정보는 124 페이지의 『위치 정보 획득』을 참조하십시오.
6. Data Handler가 변환을 완료하면, 직렬화된 데이터를 호출자에게 리턴합니다. Data Handler는 데이터를 EDI 문서를 포함하는 문자열로 리턴합니다.

삽입할 문서 분리문자 판별

Business Object를 EDI 문서로 변환하려면, EDI Data Handler는 분리문자를 EDI 문서에 올바르게 삽입해야 합니다. Data Handler는 하위 Meta Object의 속성을 사용하

여 이러한 분리문자를 지정할 값을 판별합니다. 속성 중 하나에 대해 값을 제공하지 않으면, Data Handler는 분리문자에 대해 하드 코딩된 기본값을 사용합니다.

표 45에서는 EDI 분리문자와, 해당되는 Meta Object 속성 및 하드 코딩된 기본값을 보여줍니다.

표 45. EDI 분리문자의 기본값

EDI 분리문자	하위 Meta Object의 분리문자 속성	하드 코딩된 기본값
요소 분리문자	SEPARATOR_ELEMENT	더하기 부호(+)
세그먼트 분리문자	SEPARATOR_SEGMENT	작은 따옴표(')
복합 분리문자	SEPARATOR_COMPOSIT	콜론(:)
반복 분리문자(EDIFACT 문서만)	SEPARATOR_REPEAT	탈자 부호(^)

경고:

X.12 표준을 따르는 EDI 문서: Data Handler가 Business Object를 EDI 문서로 적절하게 변환하게 하려면, 하위 Meta Object에 있는 분리문자 속성 값이 EDI 문서의 ISA 세그먼트에 있는 값과 *반드시* 일치해야 합니다. EDI Data Handler는 문서 분리문자를 판별하기 위해 ISA 세그먼트에서 데이터를 읽지 *않습니다*.

EDIFACT 표준을 따르는 EDI 문서: Data Handler가 Business Object를 EDI 문서로 적절하게 변환하게 하려면, 하위 Meta Object에 있는 분리문자 속성 값을 표 45에 정의된 대로 기본값으로 설정해야 합니다. 기본 하위 Meta Object (MO_DataHandler_DefaultEDIConfig)의 분리문자 속성은 X.12 표준에 대해 올바른 값을 포함합니다. 이 속성 기본값을 표 45에 정의된 값으로 재설정하도록 하십시오.

속성과 연관된 EDI 데이터 판별

EDI를 보유하는 Business Object의 구조는 EDI 문서 스펙에 의해 판별됩니다. (이 Business Object 구조 작성 방법에 대한 정보는 118 페이지의 『EDI 문서의 Business Object 정의 작성』을 참조하십시오.) EDI Data Handler는 속성의 카디널리티를 사용하여 속성이 표시하는 EDI 문서의 부분을 판별합니다. 이 카디널리티를 기반으로, Data Handler는 다음 조치를 취합니다.

- 속성이 *단일 카디널리티 배열*을 표시할 경우, Data Handler는 다음과 같이 속성의 응용프로그램 특정 정보를 조사하여 속성과 연관된 EDI 문서의 부분을 판별합니다.

  Data Handler는 EDI 구조에서 자체 목적을 식별하는 type 태그(예: type=header 또는 type=trailer)에 대해 속성의 응용프로그램 특정 정보를 확인합니다.

  - Data Handler가 type 태그를 찾으면, 순환하여 하위 Business Object를 처리합니다.
  - Data Handler가 이러한 태그를 찾지 *못할* 경우, Data Handler는 하위 Business Object가 EDI 문서의 세그먼트를 표시하고 122 페이지의 『세그먼트 처리』에 설명된 것처럼 처리되는 것으로 가정합니다.

- 속성이 *다중 카디널리티 배열*을 표시할 경우, 이는 세그먼트 루프를 나타냅니다. Data Handler는 단일 카디널리티 배열인 것처럼 각각의 하위 Business Object를 순환하여 처리합니다. 배열에 있는 각 Business Object 인스턴스마다 EDI 문서에 새 세그먼트를 작성합니다.
- 속성 유형이 `String`일 경우, EDI 구조는 최상위 레벨 Business Object의 모든 속성이 단일 카디널리티 또는 다중 카디널리티 배열임을 지시하므로 Data Handler는 예외를 생성합니다.

세그먼트 처리

속성이 세그먼트를 표시할 경우, Data Handler가 취하는 조치는 속성이 널인지 여부에 따라 결정됩니다.

- 속성 값이 널일 경우, Data Handler는 속성을 건너뛰고, EDI 문서에 이를 포함하지 *않습니다*.
- 속성 값이 널이 아니면, Data Handler는 다음의 처리 단계를 수행합니다.
  - 세그먼트 이름(다음 양식의 태그)에 대한 Business Object 응용프로그램 특정 정보를 구문 분석합니다.

    `name=`*`segment_name`*
  - 세그먼트 분리문자를 EDI 문서에 추가합니다.
  - 세그먼트 이름을 EDI 문서에 추가하고, 필요한 이스케이프 문자를 삽입합니다.
  - 각 하위 Business Object(단일 카디널리티 또는 다중 카디널리티)마다, Data Handler는 요소 분리문자를 EDI 문서에 추가하고 하위 Business Object를 복합 분리문자로 처리합니다(『복합 처리』 참조). 다중 카디널리티 속성의 경우, Data Handler는 발생하는 순서대로 각 하위 Business Object를 처리합니다.
  - 널이 *아닌* 각 `String` 속성마다, Data Handler는 요소 분리문자와 속성 값을 추가하며, 필요에 따라 이스케이프 문자를 삽입합니다.

복합 처리

복합을 표시하는 하위 Business Object 각각의 경우, Data Handler는 속성(`String`이어야 하는 모든 속성)을 루핑하고 다음과 같은 처리 단계를 수행합니다.

- 속성 데이터를 구문 분석하여, 필요한 이스케이프 문자를 추가합니다.
- 이스케이프 문자가 있는 속성 값을 EDI 문서에 추가합니다.
- 복합 분리문자를 문서에 추가합니다.

Business Object로 EDI 문서 변환

EDI 문서를 Business Object로 변환하려면, EDI Data Handler는 최상위 레벨 Business Object 정의의 속성을 루핑합니다. 작성할 Business Object의 이름을 확보한 후, 최상위 레벨 Business Object 및 하위 오브젝트에 속성이 표시되는 순서로, 순환하여 속성을 처리합니다. 이때 EDI 문서의 요소 값을 Business Object에 지정합니다.

EDI Data Handler는 다음과 같이 EDI 문서를 Business Object로 처리합니다.

1. Data Handler는 선택적 구성 오브젝트를 통해 전달된 등록 정보를 설정합니다. 이 정보는 `getBO()` 메소드의 `config` 인수를 통해 전달됩니다.
2. Data Handler는 자체를 초기화하여 EDI 문서를 읽을 준비를 합니다. 자세한 정보는 『Data Handler 초기화』를 참조하십시오.
3. Data Handler가 호출자로부터 Business Object를 수신하지 않을 경우, Data Handler는 네임 핸들러 찾아보기 파일에서 찾은 Business Object 이름을 기초로 Business Object를 작성해야 합니다. 자세한 정보는 129 페이지의 『Business Object의 이름 판별』을 참조하십시오.
4. Data Handler가 최상위 레벨 Business Object의 인스턴스에 대한 액세스 권한을 가지고 있으면, Data Handler는 Business Object와 해당되는 하위 오브젝트를 EDI 문서의 데이터로 채웁니다. 자세한 정보는 130 페이지의 『Business Object 채우기』를 참조하십시오.
5. Data Handler가 변환을 완료하면, 최상위 레벨 Business Object를 호출자에게 리턴합니다. Data Handler는 전체 계층 구조, 최상위 레벨 Business Object 및 해당되는 모든 하위 오브젝트를 리턴합니다.

Data Handler 초기화

EDI 문서를 Business Object로 변환하기 위해 자체를 초기화할 경우, EDI Data Handler는 다음 단계를 수행합니다.

1. 직렬화된 데이터를 포함하는 `Reader` 오브젝트가 `mark()` 조작을 지원하는지 확인합니다.
2. EDI 문서의 구문 분석을 시작하여 첫 번째 세그먼트 이름, 분리문자, 트랜잭션 ID 및 DUNS 번호를 확보합니다.

이러한 단계는 각각 다음 부속 절에 자세히 설명되어 있습니다.

Reader 오브젝트 확인

EDI Data Handler는 EDI 문서 내에서 특정 위치를 표시한 후 계속해서 해당 위치로 리턴할 수 있어야 합니다. EDI 문서는 `Reader` 오브젝트로 EDI Data Handler에 전달되므로, 이 `Reader` 오브젝트는 `mark()` 조작을 지원할 수 있어야 합니다.

그러므로 첫 번째 초기화 단계처럼, EDI Data Handler는 수신하는 Reader Object가 mark() 조작을 지원하는지 확인합니다. 그렇지 않으면, Data Handler는 오류를 로그하고 예외를 생성합니다. 직렬화된 모든 데이터는 StringReader 오브젝트에서 EDI Data Handler로 전달되는 것이 좋습니다.

주: Reader 오브젝트 및 mark() 조작에 대한 자세한 정보는 228 페이지의 『getBO() - 공용』의 설명에서 참고 절을 참조하십시오.

읽을 문서 분리문자 판별

EDI 문서를 Business Object로 변환하려면, EDI Data Handler는 EDI 문서에서 분리문자를 올바르게 읽어야 합니다. Data Handler는 문서를 구문 분석하여 이러한 분리문자를 확보합니다. EDI 문서의 처음 세 문자는 알려져 있으므로, Data Handler는 이러한 문자를 먼저 구문 분석합니다. 처음 세 문자를 읽어서 다음을 표시하는지 판별합니다.

- UNA 서비스 문자열 어드바이스(EDIFACT 문서만)

  UNA 서비스 문자열 어드바이스는 사용할 문서 분리문자를 포함합니다.

- 초기 세그먼트 이름

  Data Handler는 문서를 구문 분석하여 문서 분리문자 및 기타 위치 정보를 확보합니다.

UNA 서비스 문자열 어드바이스 확인: UNA 서비스 문자열 어드바이스는 EDIFACT 표준을 따르는 EDI 문서에서 선택적 첫 요소입니다. 이 서비스 문자열은 6개의 영숫자로 구성되는데, 순서는 다음과 같습니다.

구성요소 분리문자
요소 분리문자
10진 표시
릴리스 문자
반복 분리문자(구문 버전 4만)
세그먼트 분리문자

EDI 문서의 처음 세 자가 "UNA"일 경우, Data Handler는 EDI 문서를 해석하기 위해 UNA 서비스 문자열이 지정하는 값을 사용합니다. 이러한 분리문자 값은 Meta Object의 UNA 또는 UNB 위치 정보 속성에 있는 것을 포함하여, EDI 문서에 있는 다른 모든 분리문자 속성보다 우선합니다.

주: UNA 서비스 문자열 어드바이스가 있는 EDI 문서의 경우, Data Handler는 하위 Meta Object의 UNA 위치 정보 속성에서 트랜잭션 ID 및 DUNS 번호를 확보합니다. 자세한 정보는 다음 섹션을 참조하십시오.

위치 정보 획득: EDI 문서의 처음 세 자가 "UNA"가 *아닐* 경우, Data Handler는 해당 문자가 초기 세그먼트의 이름을 표시하는 것으로 간주합니다. Data Handler는 초기

세그먼트가 머리글의 일부이고 정확히 세 자인 이름을 가지고 있다고 가정합니다. UNA 서비스 문자열 어드바이스가 존재하지 않을 경우, Data Handler는 EDI 문서 자체로부터 문서 분리문자를 확보해야 합니다. Data Handler는 계속해서 EDI 문서를 구문 분석하여 다음 타스크를 수행합니다.

- EDI 문서의 네 번째 문자를 읽고 요소 분리문자를 판별합니다.

  Data Handler가 요소 분리문자를 판별할 수 없으면, 하위 Meta Object의 SEPARATOR_ELEMENT 속성 값을 사용합니다. 제공된 SEPARATOR_ELEMENT 값은 별표(*)입니다. 어떤 이유로 Data Handler가 하위 Meta Object로부터 요소를 확보할 수 없으면, 하드 코딩된 기본값인 더하기 부호(+)를 사용합니다.

- 하위 Meta Object에서 속성으로부터 위치 정보를 확보합니다.

  위치 정보에는 분리문자(세그먼트, 복합 및 반복 분리문자), 트랜잭션 ID 및 DUNS 번호가 포함됩니다. Data Handler는 보통 EDI 문서를 스캔하여 위치 정보를 판별할 수 있습니다.

  Data Handler가 EDI 문서 내에서 위치 정보를 찾을 수 있도록 하기 위해, EDI Data Handler와 연관된 하위 Meta Object(기본값은 MO_DataHandler_DefaultEDIConfig)에는 위치 정보를 보유하는 속성이 있습니다. 이 위치 정보 속성의 이름은 다음과 같이 EDI 문서의 첫 요소 이름에 해당됩니다.

  – X.12 표준의 경우, EDI 문서는 세그먼트 "ISA"로 시작합니다. 따라서, Data Handler는 하위 Meta Object에서 ISA 속성을 찾습니다.

  – EDIFACT 표준의 경우, 대부분의 EDI 문서는 세그먼트 "UNB"로 시작합니다. 그러나 선택적 UNA 서비스 문자열 어드바이스가 먼저 발생할 수도 있습니다. 따라서, Data Handler는 하위 Meta Object에서 초기 요소(UNA, UNB 또는 첫 번째 요소 이름)와 일치하는 속성을 찾습니다. Data Handler는 EDI 문서의 구문 분석을 시작할 때까지 UNA 서비스 문자열 어드바이스가 존재하는지 알 수 없으므로, UNA 속성과 UNB 속성 모두 하위 Meta Object에 존재해야 합니다.

    주: 기본적으로, MO_DataHandler_DefaultEDIConfig Meta Object는 X.12 표준에 대해 EDI Data Handler를 구성합니다. 그러므로 ISA 속성을 제공합니다. EDI 메시지가 EDIFACT 표준을 따를 경우, UNA 및 UNB 속성을 기본값인 하위 Meta Object에 추가하거나, EDIFACT 표준에 맞는 별도의 하위 Meta Object를 작성해야 합니다. 이 표준은 ISA 대신 UNA 및 UNB 속성을 포함하고 있습니다.

위치 정보 속성은 표 46에 표시된 일련의 태그를 사용하여 정보를 지정합니다.

표 46. 위치 정보 속성의 EDI 문서 정보

EDI 문서 정보	속성 태그	설명	필수 여부
세그먼트 분리문자	length	세그먼트 분리문자를 제외하고, 세그먼트 이름을 포함하여 첫 번째 세그먼트의 길이를 숫자로 지정합니다.	예

표 46. 위치 정보 속성의 EDI 문서 정보 (계속)

EDI 문서 정보	속성 태그	설명	필수 여부
세그먼트 계수	seg_count	EDI 문서에 기록된(business-object에서 EDI로의 변환 중) 세그먼트 수를 포함하는 필드의 상대 위치를 지정합니다. 태그 사용에 대한 정보는 119 페이지의 『Business Object를 EDI 문서로 변환』을 참조하십시오.	예
복합 분리문자	cs	복합 분리문자의 상대 위치를 제공합니다.	아니오
반복 분리문자	rs	반복 분리문자의 상대 위치를 제공합니다.	아니오
트랜잭션 ID	tid	트랜잭션 ID의 상대 위치를 제공합니다.	예
DUNS 번호	duns	DUNS 번호의 상대 위치를 제공합니다.	예
버전/릴리스 번호	version	기능 그룹/메시지 버전 번호의 상대 위치를 제공합니다.	아니오

표 46에 표시된 것처럼, 위치 정보 속성은 seg_count, length, tid, duns 태그의 값과 선택적으로 version 태그의 값을 제공해야 합니다. cs 및 rs 태그의 값은 반드시 지정하지 않아도 됩니다. 그러나 이러한 태그 중 하나를 생략하고 Data Handler가 복합 분리문자를 포함하는 EDI 문서를 구문 분석할 경우, Data Handler는 표 47에 나와 있는 우선순위를 사용하여 값을 확보합니다.

표 47. 복합 및 반복 분리문자의 기본값

우선순위	단계	복합 분리문자	반복 분리문자
1	해당 Meta Object 속성의 값을 확보합니다.	SEPARATOR_COMPOSIT	SEPARATOR_REPEAT
2	연관된 Meta Object 속성을 설정하지 않을 경우, 하드 코딩된 기본값을 사용합니다.	콜론(:)	탈자 부호(^)

cs, rs, tid, duns 태그는 다음 형식을 사용하여 EDI 문서 내에서 상대 위치를 표시합니다.

tagname=seg_name+elem_pos+compos_pos

설명:

- seg_name은 정보가 발견되는 세그먼트의 이름입니다.
- elem_pos는 *seg_name* 세그먼트 내에서 요소의 위치입니다. 요소 위치에 대해 번호를 매기는 것은 "1"(0이 아님)에서 시작합니다.
- compos_pos는 선택적이며, *elem_pos* 복합 내에서 요소 위치를 지정합니다. (*elem_pos*는 복합을 가리키는 것으로 가정합니다.) 복합 위치의 번호를 매기는 것은 "1"(0이 아님)에서 시작합니다.

seg_count 태그는 다음 형식을 사용하여 EDI 문서 내에서 상대 위치를 표시합니다.

seg_count=*seg_name+elem_pos*

여기서 seg_name 및 *elem_pos*는 위에 설명되어 있는 대로입니다. 즉, seg_count 스펙에는 i*compos_pos* 값이 *전혀* 포함되지 않습니다.

주: seg_name, *elem_pos*, *compos_pos* 값과 더하기 부호 사이에 공백을 두지 마십시오.

그림 31은 X.12 표준을 사용하는 샘플 EDI 문서를 나열한 것입니다. 이 EDI 문서를 쉽게 읽을 수 있도록 하기 위해, 예에서는 각 세그먼트 끝에 줄 바꾸기 문자가 삽입되어 있습니다.

```
ISA*00*0000000000*02*XXXX*cw*ldtp3*cw*ld*970106*1525*U*00200*0000000100*0*P*<
GS*AA*ldtp3*ld*20010424*1525*142*X*004010
ST*846*001420001
SE*2*001420001
GE*1*142
IEA*1*0000000100
```

그림 31. X.12 표준에서의 샘플 EDI 문서

X.12 표준을 따르는 EDI 문서에서 위치 정보를 확보하기 위해, EDI Data Handler는 다음 단계를 수행합니다.

1. Data Handler의 하위 Meta Object에서 첫 번째 세그먼트와 일치하는 속성을 찾습니다.

   그림 31에 있는 샘플 EDI 문서의 경우, 하위 Meta Object는 `ISA` 속성을 포함해야 합니다(첫 번째 세그먼트 이름이 `ISA`이므로).

2. 이 Meta Object 속성에서 위치 정보를 확보합니다.

   현재 예에서, ISA 속성은 다음과 같은 위치 정보를 포함합니다.

   ```
   length=77;tid=ST+1;duns=ISA+6;seg_count=SE+1
   ```

   또는 (버전이 dbfile.txt에 포함된 경우)

   ```
   length=77;tid=ST+1;version=GS+8;duns=ISA+6;seg_count=SE+1
   ```

3. 문서의 첫 번째 세그먼트 구문 분석을 계속하여 세그먼트 분리문자를 판별합니다. Data Handler는 세그먼트 분리문자가 첫 번째 세그먼트의 끝에 있는 것으로 간주하므로, 다음 단계를 수행합니다.
   - 하위 Meta Object의 위치 정보 속성에서 length 태그에 제공된 첫 번째 세그먼트 길이에 따라 세그먼트의 끝으로 이동합니다.
   - 이 위치에 있는 문자를 세그먼트 분리문자로 확보합니다.

   주: Data Handler가 세그먼트 분리문자를 찾기 위해 사용하는 알고리즘으로 인해, 이 분리문자는 영숫자로 설정될 수 없습니다. 2에서, length 태그는 그림 31 문서의 세그먼트 분리문자가 줄 바꾸기(캐리지 리턴) 문자임을 표시하는 77자의 세그먼트 길이를 지정합니다. 그러므로 Data Handler는 줄 바꾸기 문자를 세그먼트 분리문자로 해석합니다.

4. 첫 번째 세그먼트의 구문 분석을 계속하여, 하위 Meta Object의 위치 정보 속성에 있는 `tid=` 태그를 기초로 트랜잭션 ID를 판별합니다.

그림 31은 X.12 표준을 따릅니다. 이 예에는 복합 분리문자가 포함되어 있지 *않습니다*. 그러므로 ISA 속성(2에 있는)은 복합(cs) 또는 반복(rs) 분리문자를 포함하지 않습니다. 이 속성은 tid 태그를 포함하고 있습니다. 이 태그는 트랜잭션 ID가 세그먼트 ST에서 첫 번째 요소로 발생하도록 지정하므로, 그림 31의 트랜잭션 ID는 846입니다.

5. 첫 번째 세그먼트의 구문 분석을 계속하여 version을 찾습니다(선택적 version 태그를 dbfile.txt에 지정한 경우).

   그림 31에서 version은 GS 세그먼트의 8번째 요소(004010)입니다.

6. 문서를 구문 분석하여 DUNS 번호를 찾습니다. Data Handler가 DUNS 번호를 찾을 수 없으면, 오류를 로그하고 예외를 생성합니다.

   2에서 duns 태그는 DUNS 번호가 세그먼트 ISA에서 6번째 요소로 발생하도록 지정하므로, 그림 31 문서의 DUNS 번호는 ldtp3입니다.

EDIFACT 표준을 따르는 EDI 문서에서 위치 정보를 확보하기 위해, Data Handler는 X.12 표준을 따르는 EDI 문서의 구문 분석에 설명된 것과 동일한 단계를 수행합니다. 유일한 차이점은 다음과 같습니다.

- Data Handler는 Data Handler의 하위 Meta Object에서 UNB 속성을 찾아야 합니다(첫 번째 세그먼트의 이름은 ISA가 아니라 UNB이므로).
- Data Handler는 EDIFACT EDI 문서의 구문 분석에서 복합 분리문자를 좀더 찾으려고 합니다. (복합 분리문자는 X.12 문서에서 거의 발생하지 않습니다.) 복합 분리문자가 발생하면, Data Handler는 복합 및 반복 분리문자에 대해 문서를 구문 분석해야 합니다.
  - EDI 문서에 복합 분리문자가 있을 경우, 하위 Meta Object의 위치 정보 속성은 최소한 cs 태그를 포함하고 있어야 합니다. 문서가 기본값이 아닌 반복 분리문자를 사용할 경우 rs 태그를 포함할 수도 있습니다. Data Handler는 cs 및 rs 태그에 의해 제공된 위치 정보를 기초로 복합 및 반복 분리문자를 판별합니다. Data Handler가 분리문자를 판별할 수 없으면, 표 47에 나열된 기본값 중 하나를 사용합니다.
  - EDI 문서가 복합 분리문자를 포함하지 않을 경우, 위치 정보 속성은 cs 또는 rs 태그를 포함하지 않아도 됩니다.

다음 행은 EDIFACT 표준을 따르는 EDI 문서의 단편입니다.

```
ST*st_child_value_1*,*st_grand_child_val_11,st_grand_child_val_12^
st_grand_child_val_13,st_grand_child_val_14*st_child_value_4*
st_grand_child_val_21,st_grand_child_val_22
```

그림 32. 복합 분리문자가 있는 샘플 EDI 문서 단편

첫 번째 세그먼트의 이름이 "UNB"인 경우, 하위 Meta Object는 다음의 cs 태그를 포함하는 UNB 속성을 포함합니다.

```
cs=ST+2;
```

이 cs 태그는 복합 분리문자가 세그먼트 ST에서 두 번째 요소로 발생함을 지정하므로, Data Handler는 복합 분리문자로 쉼표(,)를 해석합니다. 이 단편이 발생하는 EDI 문서는 반복 분리문자를 지정하지 않으므로, 기본값인 탈자 부호(^)를 사용합니다. 그러므로 이 문서가 사용하는 하위 Meta Object의 UNB 속성은 반복 분리문자를 지정하기 위한 rs 태그를 포함할 필요가 *없습니다*. rs 태그가 없으면, Data Handler는 반복 분리문자가 자체 기본값을 가지고 있다고 가정합니다. Data Handler가 탈자 부호(^)를 발견하면 이 문자를 반복 분리문자로 해석합니다.

기본값이 아닌 반복 분리문자를 정의하려면, EDI 문서는 기본값이 아닌 문자를 필드(보통 머리글)에 포함해야 하고 연관된 하위 Meta Object의 위치 정보 속성은 이 필드의 위치를 표시하기 위해 rs 태그를 포함해야 합니다.

Business Object의 이름 판별

Data Handler는 다음의 두 방법 중 하나로 직렬화된 데이터를 수신할 수 있습니다.

- 직렬화된 데이터 *및* 빈 Business Object를 수신합니다. Data Handler는 이 Business Object를 직렬화된 데이터로 채웁니다.
- 직렬화된 데이터*만* 수신합니다. Data Handler는 Business Object를 작성한 후 직렬화된 데이터로 이를 채울 수 있습니다.

주: Data Handler가 Business Object를 수신한 경우, 130 페이지의 『Business Object 채우기』에 설명된 단계로 건너뜁니다

Data Handler가 Business Object를 수신하지 *않을* 경우, Data Handler는 작성할 Business Object 유형을 판별해야 합니다. Data Handler는 다음 단계를 통해 네임 핸들러를 호출합니다.

1. 하위 Meta Object의 NameHandlerFile 속성에 제공된 이름을 기초로 EDI 네임 핸들러 찾아보기 파일을 엽니다. 이 네임 핸들러 찾아보기 파일은 이미 존재하고 있어야 합니다. 이 파일의 열기에 실패할 경우, 네임 핸들러는 예외를 생성합니다. 자세한 정보는 107 페이지의 『네임 핸들러 찾아보기 파일 작성』을 참조하십시오.
2. 네임 핸들러 찾아보기 파일을 마지막으로 읽은 후 수정했는지 확인합니다. 수정한 경우, 메모리 내 네임 핸들러 찾아보기 테이블로 다시 내용을 읽어들입니다.
3. 트랜잭션 ID 및 DUNS 번호(초기화 단계에서 판별됨)를 기반으로, 네임 핸들러 찾아보기 테이블에서 EDI 문서와 연관된 최상위 레벨 EDI Business Object의 이름을 찾습니다.

Business Object 이름의 찾아보기에 실패한 경우, Data Handler는 오류를 로그하고 예외를 생성합니다. 이름 찾아보기에 성공한 경우, Data Handler는 데이터를 포함하는 지정된 유형의 Business Object를 작성합니다.

주: 다음 단계에서는 EDI Data Handler와 함께 제공되는 기본 네임 핸들러의 작동에 대해 설명합니다. 사용자 정의 네임 핸들러 작성 방법에 대한 정보는 132 페이지의 『EDI Data Handler 사용자 정의』를 참조하십시오.

Business Object 채우기

EDI 분리문자를 판별하고 최상위 레벨 Business Object를 작성했다면, Data Handler는 다음 단계를 통해 직렬화된 데이터로 Business Object를 채웁니다.

1. `DefaultVerb` Meta Object 속성이 설정되어 있으면, Data Handler는 Business Object의 verb를 `DefaultVerb`가 지정하는 값으로 설정합니다. `DefaultVerb`에 대해 제공되는 값은 `Create`입니다. 그렇지 않으면, Data Handler는 verb를 설정할 필요가 없는 것으로 가정합니다.
2. Data Handler는 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 `cw_mo_` 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 Business Object의 이러한 속성을 채우기 위해 처리를 수행하지 *않습니다*. `cw_mo_` 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
3. Data Handler는 최상위 레벨 Business Object 정의에서 나머지 속성을 루핑합니다. 각 속성의 카디널리티에 따라, Data Handler는 속성이 표시하는 EDI 문서 부분을 판별합니다. 자세한 정보는 『EDI 데이터와 연관된 속성 판별』을 참조하십시오.
4. Data Handler가 현재 EDI 데이터와 연관된 속성을 식별하고 나면, 적절한 단계를 수행하여 EDI 데이터를 이 속성에 쓸 수 있습니다. Data Handler는 분리문자(초기화 단계에서 판별)를 기초로 EDI 데이터를 구문 분석합니다. 자세한 정보는 131 페이지의 『EDI 문서 구문 분석』을 참조하십시오.

Data Handler가 최상위 레벨 Business Object의 모든 속성을 채우고 나면, 선택적으로 모든 EDI 데이터의 구문이 분석되었는지 확인할 수 있습니다.

EDI 데이터와 연관된 속성 판별

EDI를 보유하는 Business Object의 구조는 EDI 문서 스펙에 의해 판별됩니다. (이 Business Object 구조 작성 방법에 대한 정보는 118 페이지의 『EDI 문서의 Business Object 정의 작성』을 참조하십시오.) EDI Data Handler는 속성의 카디널리티를 사용하여 속성이 EDI 문서의 현재 EDI 부분을 표시하는지 판별합니다. 이 카디널리티를 기반으로, Data Handler는 다음 조치를 취합니다.

- 속성이 *단일 카디널리티 배열*을 표시할 경우, Data Handler는 다음과 같이 속성의 응용프로그램 특정 정보를 조사하여 속성과 연관된 EDI 문서 부분을 판별합니다.

– 속성의 응용프로그램 특정 정보에서 메타 데이터를 조사하여 작성할 Business Object 유형을 판별합니다.
– 표 48에 표시된 대로, 속성의 응용프로그램 특정 정보로 표시되는 Business Object를 작성합니다.

표 48. 응용프로그램 특정 정보 및 연관된 EDI Business Object

응용프로그램 특정 정보	하위 **Business Object**
`type=header`	머리글 Business Object
`name=segment name`	세그먼트 Business Object
`name=name of first segment in loop;type=loop`	세그먼트 루프 Business Object
`type=trailer`	트레일러 Business Object

– Business Object 정의에서 발생하는 순서대로 속성을 루핑하여 새로운 하위 Business Object를 순환하여 처리합니다.

- 속성이 *다중 카디널리티 배열*을 표시할 경우, 이는 세그먼트 루프를 나타냅니다. Data Handler는 단일 카디널리티 배열인 것처럼 각각의 하위 Business Object를 순환하여 처리합니다. EDI 문서에 발생하는 루프의 각 인스턴스에 대해 배열에서 새 Business Object를 작성합니다.
- 속성 유형이 `String`일 경우, EDI 구조는 최상위 레벨 Business Object의 모든 속성이 단일 카디널리티 또는 다중 카디널리티 배열임을 지시하므로 Data Handler는 예외를 생성합니다.

EDI 문서 구문 분석

EDI Data Handler는 초기화 단계에서 식별한 분리문자를 기초로 EDI 문서의 정보를 구문 분석합니다. 이러한 분리문자는 데이터의 여러 조각을 각각 판별하고, 이후 Data Handler는 적절한 속성에 이를 대응시킵니다. 표 49에서는 Data Handler가 여러 가지의 EDI Business Object에 대해 수행하는 구문 분석 타스크를 보여줍니다.

표 49. EDI Business Object에 대한 구문 분석 타스크

응용프로그램 특정 정보	구문 분석 타스크
`type=header, type=trailer`	Data Handler는 문서에 나타나는 다음 세그먼트에 해당하는 Business Object에서 위치를 찾아 해당 세그먼트를 구문 분석하여 하위 Business Object를 채웁니다.
`name=`*`segment_name`*(type 태그 아님) `type=loop`	Data Handler는 Business Object가 세그먼트를 표시하고 현재 세그먼트를 구문 분석하여 Business Object를 채울 것으로 가정합니다. 루프에 포함된 첫 번째 세그먼트의 이름은 응용프로그램 특정 정보에 지정해야 합니다. Data Handler는 이러한 루프 세그먼트에 대해 EDI 문서를 구문 분석하고 데이터를 Business Object에 추가합니다.

EDI Data Handler 사용자 정의

특수한 네임 핸들러를 작성하여 EDI Data Handler를 사용자 정의할 수 있습니다. EDI Data Handler는 네임 핸들러를 호출하여 작성할 Business Object의 이름을 확보합니다. Data Handler는 data-handler Meta Object에 저장된 NameHandlerClass 속성 값을 사용하여 호출할 네임 핸들러를 결정합니다. EDI Data Handler와 함께 포함된 기본 네임 핸들러는 네임 핸들러 찾아보기 파일의 Business Object 이름(NameHandlerFile Meta Object 속성으로 표시된)을 찾습니다. 네임 핸들러를 다른 방식으로 기능하도록 하려면 다음을 수행하십시오.

1. NameHandler 클래스를 확장하여 사용자 정의 네임 핸들러를 작성하십시오.
2. EDI Data Handler용 Meta Object에 있는 NameHandlerClass 속성의 기본값을 갱신하여 EDI Data Handler가 사용자 정의 name-handler 클래스를 사용하도록 구성하십시오.

Custom Data Handler 작성 방법에 대한 정보는 215 페이지의 『사용자 정의 네임 핸들러 빌드』를 참조하십시오.

제 5 장 Request-Response Data Handler

Request-Response Data Handler는 요청 및 응답 Business Object에 대해 다른 데이터 형식이 필요한 시나리오를 다룹니다. Request-Response Data Handler에서는 이들 두 형식이 다를 수 있습니다. 이 장에서는 Request-Response Data Handler의 정보 처리 방법 및 Request-Response Data Handler가 처리할 Business Object 정의 방법에 대해 설명합니다. Request-Response Data Handler의 구성 방법에 대해서도 설명합니다.

이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 141 페이지의 『Business Object 정의의 요구사항』
- 146 페이지의 『Request-Response Data Handler 구성』
- 149 페이지의 『Requrest Data Handler로 Business Object 변환』
- 150 페이지의 『Response Data Handler로 Business Object 변환』
- 151 페이지의 『오류 처리』
- 151 페이지의 『Request-Response Data Handler 사용자 정의』

주: Request-Response Data Handler는 `CwDataHandler.jar` 파일에 들어 있는 기본 Data Handler 중 하나입니다. 이 Data Handler 설치 방법 및 해당 소스 코드의 저장 위치에 대한 정보는 25 페이지의 제 2 장 『Data Handler 설치 및 구성』을 참조하십시오.

개요

Request-Response Data Handler는 기본 역할이 형식이 다른 요청 및 응답 데이터에 대한 지원을 제공하는 데이터 변환 모듈입니다. 즉, 이 Data Handler를 사용하면 호출 문맥(예: 어댑터 또는 서버 액세스 인터페이스)이 다음과 같이 두 개의 다른 Data Handler를 호출할 수 있습니다.

- *Rrequest Data Handler*는 요청을 *시작*하는 WebSphere Business Integration System 구성요소에 대한 데이터 변환을 처리합니다.
- *Response Data Handler*는 요청에 *응답*하는 WebSphere Business Integration System 구성요소에 대한 데이터 변환을 처리합니다.

다른 Data Handler를 사용하는 경우 호출 문맥은 응용프로그램 또는 액세스 클라이언트에서 동일한 형식의 데이터를 전송 및 수신하는 것으로 가정합니다. 따라서 호출 문맥은 요청 및 응답 Business Object의 변환을 처리하기 위해 단일 Data Handler를 호출하도록 구성됩니다.

그러나 XML을 입력으로 승인하고 일부 사용자 정의 형식의 문서를 출력으로 리턴하는 레거시 응용프로그램이 있을 수 있습니다. Data Handler를 종료하는 것만으로는 이 상황을 쉽게 처리할 수 없습니다. 그러나 이 레거시 응용프로그램과 통신하는 어댑터가 Request-Response Data Handler를 호출하도록 구성할 수 있습니다. 이 Data Handler는 다음과 같이 입력 및 출력에 대해 개별 Data Handler를 호출하도록 구성될 수 있습니다.

- 요청 Business Object를 처리하기 위한 IBM WebSphere Business Integration Data Handler for XML(XML Data Handler)
- 응답 Business Object를 처리하기 위한 Custom Data Handler

통합 브로커가 이 어댑터로 요청을 전송하는 요청 처리 시, 어댑터는 Request-Response Data Handler를 호출하여 요청 Business Object로 전송합니다. Request-Response Data Handler는 해당 구성을 확인하여 이 Business Object를 XML 문서로 변환하기 위해 XML Data Handler를 호출해야 하는지 여부를 판별합니다. 일단 이 Business Object가 변환되면 Request-Response Data Handler가 어댑터로 XML 문서를 리턴하고 레거시 응용프로그램으로 라우트합니다.

그림 33에서는 Request-Response Data Handler가 수행하는 Business Object에서 문자열로의 변환 예를 보여줍니다.

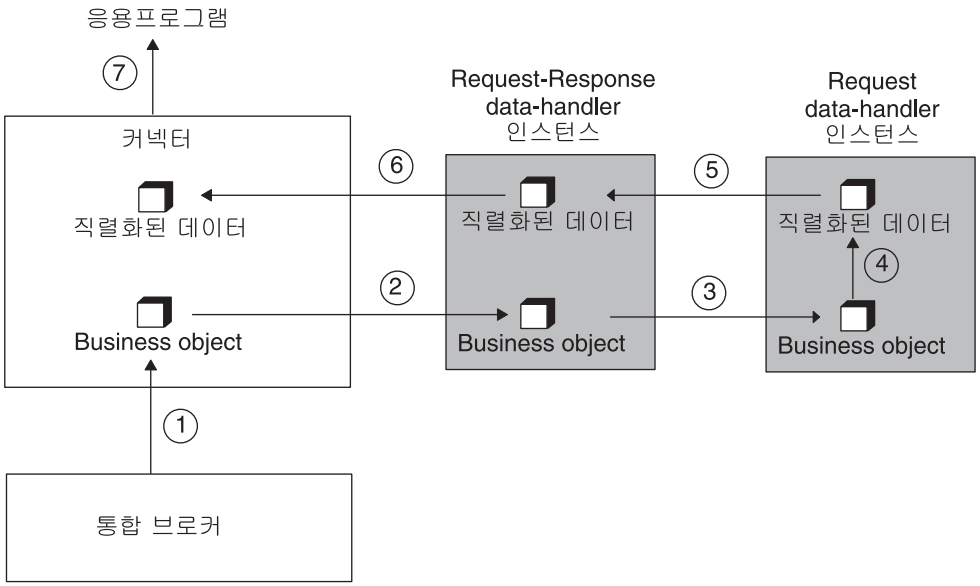

그림 33. Request-Response Data Handler를 사용한 Business Object에서 문자열로의 변환

어댑터는 나중에 레거시 응용프로그램에서 응답을 수신할 수 있습니다. 이 응답은 레거시 응용프로그램의 사용자 정의 형식을 갖게 됩니다. 어댑터는 다시 Request-Response Data Handler를 호출하여 응답 데이터를 전송합니다. Request-Response Data Handler는 해당 구성을 확인하여 이 응답 데이터를 Business Object로 변환하기 위해 Custom Data Handler를 호출해야 하는지 여부를 판별합니다. 일단 이 데이터가 변환되면 Request-Response Data Handler가 어댑터로 Business Object를 리턴하고 이어 통합 브로커로 라우트합니다.

Request-Response Data Handler는 ICS 통합 브로커가 하나의 Business Object 유형을 협업 포트로 지정하고 하나 이상의 다른 Business Object를 이어 수신하도록 할 수도 있습니다. 예를 들어 액세스 클라이언트가 고객 오브젝트를 협업으로 전송하고 이어 해당 고객에 대한 보류 주문 오브젝트 배열을 수신할 수 있습니다.

Request-Response Data Handler는 `text/requestresponse` MIME 유형의 직렬화된 데이터를 지원합니다. 직렬화된 데이터는 텍스트 또는 2진 데이터일 수 있습니다. 그러나 기본 최상위 레벨 Meta Object(`MO_DataHandler_Default` 또는 `MO_Server_DataHandler`)는 `text/requestresponse` MIME 유형을 지원하지 *않습니다*. 따라서 액세스 클라이언트 또는 커넥터가 Request-Response Data Handler를 호출하려면, 해당 최상위 레벨 Meta Object가 `text/requestresponse` MIME 유형을 지원하도록 수정해야 합니다. 자세한 정보는 146 페이지의 『최상위 레벨 Meta Object 구성』을 참조하십시오.

이 개요에서는 Request-Response Data Handler에 대한 다음 정보를 제공합니다.

- 『Request-Response data-handler 구성요소』
- 136 페이지의 『Request-Response Data Handler의 기능』
- 141 페이지의 『요청 및 응답 Business Object 처리』

Request-Response data-handler 구성요소

Data Handler는 다음의 두 방법 중 하나로 직렬화된 데이터를 수신할 수 있습니다.

- 직렬화된 데이터 및 빈 Business Object를 수신합니다. Data Handler는 이 Business Object를 직렬화된 데이터로 채웁니다.
- 직렬화된 데이터*만* 수신합니다. Data Handler는 Business Object를 작성한 후 직렬화된 데이터로 이를 채울 수 있습니다.

Request-Response Data Handler는 네임 핸들러를 사용하여, 자신이 작성하는 최상위 레벨 Business Object의 이름을 작성합니다. 그림 34에서는 Request-Response Data Handler 구성요소 및 구성요소 간의 관계를 보여줍니다.

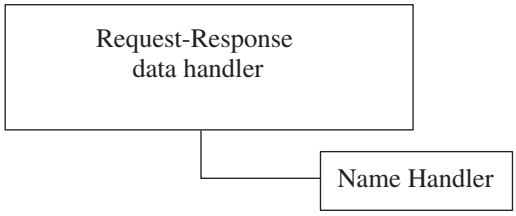

그림 34. Request-Response data-handler 구성요소

Data Handler는 Request-Response Data Handler 하위 Meta Object의 NameHandlerClass 속성 값을 기반으로 하는 네임 핸들러의 인스턴스를 호출합니다.

- 클래스 이름이 제공되지 않은 경우(NameHandlerClass 속성이 비어 있는 경우), Data Handler는 요청에 대해 구성된 Data Handler가 생성하는 최상위 레벨 Business Object의 이름 앞에 BOPrefix 속성 값을 추가하는 기본 네임 핸들러를 사용합니다. 오류 및 예외를 생성합니다.

  예를 들어, 사용자가 REQUESTTEST의 기본 BOPrefix를 지정하고 Requrest Data Handler가 Customer Business Object를 생성하는 경우, Request-Response Data Handler가 최상위 레벨 오브젝트인 REQUESTTEST_Customer를 작성하고 Customer 오브젝트로 하위 오브젝트 중 하나를 채웁니다.
- 클래스 이름이 NameHandlerClass 속성에 제공되면, Request-Response Data Handler가 이 네임 핸들러를 사용하여 Business Object 이름을 판별합니다.

  사용자 정의 네임 핸들러를 지정하려면 사용자 정의 네임 핸들러 클래스의 이름에 NameHandlerClass를 설정하십시오. 사용자 정의 네임 핸들러 작성 방법에 대한 정보는 151 페이지의 『Request-Response Data Handler 사용자 정의』를 참조하십시오.

제품과 함께 제공된 Meta Object 버전으로 된 NameHandlerClass 속성은 공백입니다. 그러므로 Request-Response 네임 핸들러는 기본 네임 핸들러를 사용합니다.

Request-Response Data Handler의 기능

Request-Response Data Handler는 다음 두 가지 경우에 모두 유용합니다.

- 『이벤트 처리 지원』
- 139 페이지의 『요청 처리 지원』

이벤트 처리 지원

이벤트 처리에는 응용프로그램 비즈니스 엔티티의 변경을 나타내는 일부 이벤트가 발생했다는 통합 브로커의 공고가 포함됩니다. 이벤트 공고에서, Data Handler의 호출 문맥은 직렬화된 데이터를 Business Object로 변환시킬 Data Handler를 호출합니다. 이후 이 Business Object는 통합 브로커로 라우트됩니다. 이러한 문자열에서 Business

Object로의 변환은 Requrest Data Handler에 의해 수행됩니다. 이 Data Handler가 요청(입력) 형식에서 Business Object로의 변환을 처리하기 때문입니다.

이벤트 처리는 동기 또는 비동기일 수 있습니다. 그러나, 비동기 이벤트 처리에서 어댑터, 특히 어댑터의 커넥터 구성요소는 다음과 같이 통합 브로커로부터 응답을 기다리지 않습니다.

- 동기 이벤트 처리는 액세스 클라이언트가 IBM WebSphere InterChange Server(특히 이 서버 내의 일부 협업)에 일부 이벤트가 발생했다고 알리는 경우 발생합니다. 액세스 클라이언트는 ICS의 응답을 대기하지 않으며 응답도 대기하지 않습니다. 이런 경우, ICS의 서버 액세스 인터페이스가 필요한 Data Handler를 호출합니다.
- 비동기 이벤트 처리는 어댑터(특히 어댑터의 커넥터 구성요소)가 응용프로그램 또는 기술에서 이벤트를 수신하고 이 이벤트를 Business Object의 양식으로 통합 브로커에 전송하여 무엇인가 발생했음을 알리는 경우 이루어집니다. 그러나 커넥터는 통합 브로커의 응답을 기다리지 않습니다. 그러므로 Request-Response Data Handler는 비동기 이벤트 처리의 경우에는 유용하지 않습니다.

주: Data Handler의 호출 문맥에 대한 자세한 정보는 7 페이지의 『Data Handler 호출에 필요한 문맥』을 참조하십시오.

예를 들어, 다음 단계는 동기 이벤트 처리에 대해 설명합니다. 이 단계는 Request-Response Data Handler가 ICS의 서버 액세스 인터페이스가 한 가지 형식(해당 요청 형식)으로 ICS에 데이터를 전송하고 이어 다른 양식의 데이터를 수신하도록 허용하는 방법에 대해 설명합니다. 이를 통해 액세스 클라이언트가 고객 XML 문서를 전송하고 이어 해당 고객에 대한 보류 주문이 들어 있는 XML 문서를 수신하는 시나리오를 완료할 수 있습니다.

1. 액세스 클라이언트가 ICS의 협업으로 실행될 이벤트를 전송합니다(요청 형식의 데이터로).
2. ICS는 Request-Response Data Handler의 새 인스턴스를 작성하고 요청 형식 데이터를 전달합니다.
3. Request-Response Data Handler는 구성된 Requrest Data Handler를 호출하며, 요청 형식 데이터를 Business Object로 변환합니다.

   Request-Response Data Handler는 `getBO()` 메소드를 사용하여, 수신하는 직렬화된 데이터를 처리합니다. 또한 다음 조건 중 하나가 true인 경우 이 직렬화된 데이터에 Requrest Data Handler를 호출합니다.

   - `getBO()` 메소드가 직렬화된 데이터만 인수로 수신하는 경우, 이 데이터를 새 요청으로 간주합니다.
   - `getBO()` 메소드가 최상위 레벨 Business Object 및 직렬화된 데이터를 인수로 수신하는 경우, 최상위 레벨 Business Object의 하위 Business Object를 확인하여 다음 조치를 판별합니다.

모든 하위 Business Object에 CxIgnore(*비어 있음*을 표시) 값이 있는 경우, Data Handler는 직렬화된 데이터가 새 요청을 나타내는 것으로 가정합니다.

주: 최상위 레벨 Business Object의 하위 Business Object를 채우는 경우, Data Handler는 이 하위 오브젝트가 원래 요청을 나타내고 결과적으로 직렬화된 데이터가 응답을 나타내는 것으로 가정합니다. 따라서 데이터를 처리할 Response Data Handler의 인스턴스를 작성합니다. 자세한 정보는 139 페이지의 『요청 처리 지원』을 참조하십시오.

두 경우 모두 Request-Response Data Handler는 Requrest Data Handler의 인스턴스를 작성하고 이 Data Handler를 사용하여 요청 형식 데이터를 요청 Business Object로 변환합니다. Request-Response Data Handler는 Requrest Data Handler로 요청 형식 데이터를 전달합니다. Requrest Data Handler는 해당 요청 Business Object를 리턴합니다.

4. Request-Response Data Handler는 다음과 같이 최상위 레벨 Business Object의 하위로서 요청 Business Object를 추가합니다.
   - Data Handler의 `getBO()` 메소드가 최상위 레벨 Business Object를 수신하지 않은 경우, 새 Business Object를 작성한 후 요청 Business Object를 추가해야 합니다. Data Handler는 이 새 Business Object의 이름을 지정하기 위해 네임 핸들러를 호출합니다. 자세한 정보는 135 페이지의 『Request-Response data-handler 구성요소』를 참조하십시오.
   - Data Handler의 `getBO()` 메소드가 최상위 레벨 Business Object를 *수신한* 경우, 요청 Business Object를 추가합니다.
5. Request-Response Data Handler가 동기적으로 최상위 레벨 Business Object를 협업(액세스 클라이언트가 지정)으로 리턴합니다.
6. 협업이 최상위 레벨 Business Object를 수신하고 하위 오브젝트(요청 Business Object 포함)를 추출하며 일부 비즈니스 프로세스를 실행합니다.
7. 협업이 새 응답 Business Object를 작성하여 최상위 레벨 Business Object에 추가한 후 리턴합니다.

   이 최상위 레벨 Business Object는 Request-Response Data Handler에서 협업이 수신한 Business Object입니다. 협업이 이 Business Object를 갱신하면, Business Object에 원래 요청 Business Object 및 새로 작성된 응답 Business Object가 모두 포함됩니다.
8. ICS가 수정된 최상위 레벨 Business Object를 Request-Response Data Handler로 전달합니다.
9. Request-Response Data Handler가 구성된 Response Data Handler를 호출하여 응답 Business Object를 응답 형식 데이터로 변환합니다.

Request-Response Data Handler는 getStringFromBO() 메소드를 사용하여, 수신하는 최상위 레벨 Business Object를 처리합니다. 최상위 레벨 Business Object의 둘 이상의 하위 Business Object를 채우는 경우, Data Handler는 최상위 레벨 Business Object가 원래 요청 Business Object와 응답 Business Object를 모두 포함하여 결과적으로 응답 Business Object를 변환해야 하는 것으로 가정합니다. 따라서 최상위 레벨 Business Object에서 마지막으로 정의된 하위 Business Object를 응답 Business Object로 처리할 Response Data Handler의 인스턴스를 작성합니다.

주: getStringFromBO()가 수신한 최상위 레벨 Business Object의 한 하위 Business Object만 채우는 경우, Data Handler는 하위 오브젝트가 새 요청을 나타내는 것으로 가정합니다. 따라서 요청 Business Object를 직렬화된 데이터로 변환할(요청 형식으로) Requrest Data Handler의 인스턴스를 작성합니다. 자세한 정보는 『요청 처리 지원』을 참조하십시오.

10. Request-Response Data Handler는 응답 형식 데이터를 호출자(ICS의 서버 액세스 프레임워크)에 리턴합니다.
11. ICS는 액세스 클라이언트에 응답 형식 데이터(응답 Business Object를 기반으로 함)를 리턴합니다.

커넥터는 executeCollaboration() 메소드로 동기 이벤트 처리를 수행할 수도 있습니다. 그러나 일반적으로 이벤트 감지 메커니즘(예: 폴링)을 사용하여 비동기 이벤트 처리를 수행합니다.

요청 처리 지원

요청 처리에는 통합 브로커의 요청 수신 및 응용프로그램 비즈니스 엔티티에서의 적절한 변경 시작이 포함됩니다. 액세스 클라이언트(동기식) 또는 커넥터(비동기식)로 시작될 수 있는 이벤트 처리와 달리 요청 처리는 통합 브로커에 의해 시작되며 커넥터(액세스 클라이언트가 아님)와의 통신만 포함됩니다.

요청 처리에서, Data Handler의 호출 문맥은 직렬화된 데이터를 Business Object로 변환시킬 Data Handler를 호출합니다. 이후 이 Business Object는 통합 브로커로 라우트됩니다. 이것은 문자열에서 Business Object로의 변환입니다.

예를 들어, 다음 단계는 IBM WebSphere InterChange Server 통합 브로커 및 기술 어댑터를 포함하는 요청 처리에 대해 설명합니다. 이 기술 어댑터는 요청 및 응답 데이터를 처리하기 위해 Request-Response Data Handler를 사용하도록 구성됩니다. 이 경우, 요청 데이터의 형식과 응답 데이터의 형식은 서로 다릅니다.

1. 협업이 새 최상위 레벨 Business Object의 인스턴스를 작성하여 요청 Business Object를 하위로서 추가합니다.

2. 협업이 기술 커넥터로 요청을 전송하고(최상위 레벨 Business Object의 형식으로), 이어 최상위 레벨 Business Object를 Request-Response Data Handler로 전달합니다.
3. Request-Response Data Handler가 구성된 Requrest Data Handler를 호출하여 요청 Business Object를 요청 형식 데이터로 변환합니다.

   Request-Response Data Handler는 `getStringFromBO()` 메소드를 사용하여, 수신하는 최상위 레벨 Business Object를 처리합니다. 최상위 레벨 Business Object의 한 하위 Business Object만 채우는 경우, Request-Response Data Handler는 하위 오브젝트가 새 요청을 나타내는 것으로 가정합니다. 따라서 요청 Business Object를 직렬화된 데이터로 변환할(요청 형식으로) Requrest Data Handler의 인스턴스를 작성합니다.

   주: `getStringFromBO()`가 수신한 최상위 레벨 Business Object의 둘 이상의 하위 Business Object를 채우는 경우, Data Handler는 최상위 레벨 Business Object가 원래 요청 Business Object와 응답 Business Object를 모두 포함하여 결과적으로 응답 Business Object를 변환해야 하는 것으로 가정합니다. 따라서 최상위 레벨 Business Object에서 마지막으로 정의된 하위 Business Object를 응답 Business Object로 처리할 Response Data Handler의 인스턴스를 작성합니다. 자세한 정보는 136 페이지의 『이벤트 처리 지원』을 참조하십시오.
4. 기술 커넥터가 요청 형식 데이터를 응용프로그램으로 전송합니다.
5. 응용프로그램은 몇 가지 타스크를 수행하고 응답 형식 데이터를 기술 커넥터로 리턴합니다.
6. 기술 커넥터는 원래 최상위 레벨 Business Object와 응답 형식 데이터를 모두 Request-Response Data Handler로 전달합니다.
7. Request-Response Data Handler는 응답 형식 데이터를 Business Object로 변환하는 구성된 Response Data Handler를 호출합니다.

   Request-Response Data Handler는 `getBO()` 메소드를 사용하여, 수신하는 최상위 레벨 Business Object 및 직렬화된 데이터를 처리합니다. 최상위 레벨 Business Object의 하위 Business Object를 채우는 경우, Request-Response Data Handler는 하위 오브젝트가 원래 요청을 나타내고 결과적으로 수신한 직렬화된 데이터가 응답 형식을 나타내는 것으로 가정합니다. 따라서 직렬화된 데이터를 응답 Business Object로 변환할 Response Data Handler의 인스턴스를 작성합니다.

   주: `getBO()`가 수신하는 최상위 레벨 Business Object의 모든 하위 Business Object에 CxIgnore(*비어 있음*을 표시) 값이 있는 경우, Data Handler는 직렬화된 데이터가 새 요청을 나타내고 데이터를 처리할 Requrest Data

Handler의 인스턴스를 작성하는 것으로 가정합니다. 자세한 정보는 136 페이지의 『이벤트 처리 지원』을 참조하십시오.

8. Request-Response Data Handler는 최상위 레벨 Business Object의 하위로서 이 응답 Business Object를 추가한 후 호출자(기술 커넥터)에게 이 최상위 레벨 Business Object를 리턴합니다.
9. 기술 커넥터는 갱신된 최상위 레벨 Business Object를 협업으로 리턴합니다.
10. 협업이 최상위 레벨 Business Object를 수신하고 응답 Business Object의 내용을 비즈니스 프로세스로 통합합니다.

요청 및 응답 Business Object 처리

Request-Response Data Handler를 사용하여 요청 Business Object를 해당 요청 형식으로 변환하거나 응답 형식의 데이터를 응답 Business Object로 변환하려면 표 50에서 설명하는 단계를 수행해야 합니다.

표 50. Request-Response Data Handler 사용

단계	자세한 정보
1. Business Object 구조에 대해 설명하는 Business Object 정의가 존재해야 하고 Request-Response Data Handler(및 해당 구성요소 Data Handler) 실행 시 사용할 수 있어야 합니다.	『Business Object 정의의 요구사항』
2. Request-Response Data Handler를 사용자 환경에 맞게 구성해야 합니다.	146 페이지의 『Request-Response Data Handler 구성』
3. 다음과 같은 해당 데이터 조작을 수행하기 위해 호출 문맥(커넥터 또는 액세스 클라이언트)에서 Request-Response Data Handler를 호출해야 합니다.	
a) 데이터 조작: 이 요청을 시작하는 구성요소에서 요청을 수신하여 적절한 형식으로 변환합니다.	149 페이지의 『Requrest Data Handler로 Business Object 변환』
b) 데이터 조작: 이 요청에 대응하는 구성요소에서 응답을 수신하여 적절한 형식으로 변환합니다.	150 페이지의 『Response Data Handler로 Business Object 변환』

Business Object 정의의 요구사항

Request-Response Data Handler를 사용하려면, Business Object 정의가 Data Handler에서 필요한 구조를 제공하도록 Business Object 정의를 작성하거나 수정해야 합니다. 그러나 다른 Data Handler와 달리, 메타 데이터를 포함하도록 Business Object 정의를 수정할 필요는 없습니다. 이 섹션에서는 Request-Response data handler에서 사용할 수 있도록 Business Object 정의를 작성하기 위해 필요한 정보를 제공합니다. 특히 다음 정보를 제공합니다.

- 142 페이지의 『요청-응답 Business Object 구조 이해』
- 145 페이지의 『Request-Response Data Handler의 Business Object 정의 작성』

요청-응답 Business Object 구조 이해

Request-Response Data Handler는 해당 요청 또는 Response Data Handler와 Business Object를 송수신할 때 Business Object 정의를 사용합니다. Request-Response Data Handler는 처리할 수 있는 Business Object의 구조에 특정 요구사항을 적용합니다. Data Handler로 전달된 Business Object에는 하나의 요청 하위 오브젝트 및 하나 이상의 응답 하위 오브젝트가 있어야 합니다. 이들 하위 오브젝트는 이들을 처리할 Data Handler의 요구사항을 준수해야 합니다.

주: Request-Response Data Handler는 Business Object 정의 또는 해당 속성에서 응용프로그램 특정 정보가 필요하지 *않습니다.*

그림 35에서는 요청-응답 Business Object를 나타내는 Business Object 구조를 보여줍니다.

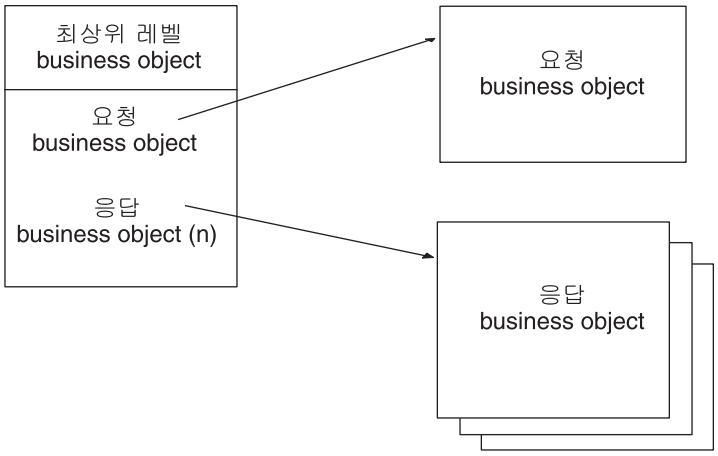

그림 35. 요청-응답 Business Object의 Business Object 구조

Business Object 정의가 Request-Response Data Handler의 요구사항을 준수하도록 하려면 다음 Business Object 각각에 대한 지침을 사용하십시오.

- 『최상위 레벨 Business Object』
- 143 페이지의 『요청 Business Object』
- 144 페이지의 『응답 Business Object』

최상위 레벨 Business Object

Request-Response Data Handler는 최상위 레벨 Business Object가 호출 문맥 송수신 정보를 보유할 것으로 예상합니다. 표 51에서는 Request-Response Data Handler의 Business Object 등록 정보 해석 방법 및 Request-Response Data Handler에서 사용할 수 있도록 Business Object 수정 시 등록 정보 설정 방법에 대해 설명합니다.

표 51. 최상위 레벨 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 이 Business Object 정의는 표준 접두부로 시작하는 것이 좋습니다. 최상위 레벨 Business Object 의 이름은 다음과 같이 메시지 표준에 따라 결정됩니다. *BusObj 접두부 + 응답 Business Object*
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific Information	응용프로그램 특정 정보가 필요하지 않습니다.

이 최상위 레벨 Business Object에는 다음과 같은 속성이 있어야 합니다.

- 단일 요청 Business Object를 보유하는 단일 카디널리티 속성

  이 요청 Business Object는 Requrest Data Handler가 필요로 하는 모든 Business Object 요구사항을 준수해야 합니다.

- 응답 Business Object를 보유하기 위한 단일 카디널리티 속성.

  이 응답 Business Object는 Response Data Handler가 필요로 하는 모든 Business Object 요구사항을 준수해야 합니다.

  참고:

  1. 대상 응용프로그램이 둘 *이상의* 응답 Business Object 종류를 리턴하는 경우, 최상위 레벨 Business Object는 응답 Business Object의 각 유형마다 하나의 하위 Business Object를 포함해야 합니다.

     예를 들어, 대상 응용프로그램이 Customer XML 문서 또는 OrderUpdate XML 문서를 리턴하는 경우, 최상위 레벨 Business Object 정의는 두 개의 속성을 포함해야 합니다. 하나는 Customer XML 문서를 표현하는 Business Object 정의를 보유하기 위한 것이고 두 번째는 OrderUpdate XML 문서를 보유하는 Business Object 정의를 보유하기 위한 것입니다. 응답 Business Object를 받으면 Data Handler는 해당 속성을 채워야 합니다.

  2. 기대를 충족시키지 않는 최상위 레벨 Business Object를 수신하거나 다른 관련 Data Handler가 Business Object 또는 Business Object로 전달된 문서를 변환하지 못하는 경우 Data Handler는 실패합니다.

요청 Business Object

Requrest Data Handler에 대한 요청 정보를 보유하기 위해, Request-Response Data Handler는 요청 Business Object를 최상위 레벨 Business Object의 첫 번째 속성으로 예상합니다. 이 속성은 단일 카디널리티의 속성이어야 합니다. 표 52에서는 Request-Response Data Handler가 이 Business Object 정의의 등록 정보를 해석하는 방법 및

Request-Response Data Handler에서 사용할 수 있도록 Business Object를 수정할 때

이 등록 정보를 설정하는 방법에 대해 설명합니다.

표 52. 요청 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 이 이름은 Requrest Data Handler가 처리하는 Business Object 정의의 이름과 일치해야 합니다.
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	사용 중인 특정 Requrest Data Handler에 따라 다릅니다.

주: 요청 Business Object의 형식에 대한 정보는 Requrest Data Handler의 기능을 수행하는 Data Handler에 대한 문서를 참조하십시오.

예를 들어, `RequestDataHandlerMimeType`을 `text/xml`로 지정한 경우, 요청 Business Object로 정의하는 하위 오브젝트는 XML Data Handler와 호환할 수 있어야 합니다.

응답 Business Object

Request-Response Data Handler는 Response Data Handler에 대한 응답 정보를 보유하기 위해 응답 Business Object를 최상위 레벨 Business Object의 두 번째 속성으로 시작하는 속성에 있는 것으로 예상합니다. 이 속성은 단일 카디널리티의 속성이어야 합니다. Response Data Handler가 둘 이상의 Business Object를 리턴하는 경우, 최상위 Business Object는 각 가능한 유형에 대한 속성을 가지고 있습니다. 표 53에서는 Request-Response Data Handler가 이 Business Object 정의의 등록 정보를 해석하는 방법 및 Request-Response Data Handler에서 사용할 수 있도록 Business Object를 수정할 때 이 등록 정보를 설정하는 방법에 대해 설명합니다.

표 53. 응답 Business Object 정의에 대한 등록 정보

등록 정보 이름	설명
Name	각 Business Object 정의에는 고유한 이름이 있어야 합니다. 이 이름은 Response Data Handler가 처리하는 Business Object 정의의 이름과 일치해야 합니다.
Version	Business Object 정의의 현재 버전을 표시하는 상수. 현재 값은 1.0.0입니다.
Application-specific information	사용 중인 특정 Requrest Data Handler에 따라 다릅니다.

주: 요청 Business Object의 형식에 대한 정보는 Requrest Data Handler의 기능을 수행하는 Data Handler에 대한 문서를 참조하십시오.

예를 들어, `ResponseDataHandlerMimeType`을 `text/abc`로 지정한 경우, 요청 Business Object로 정의하는 하위 오브젝트는 `abc` MIME 유형을 처리할 수 있는 Custom Data Handler와 호환할 수 있어야 합니다.

Request-Response Data Handler의 Business Object 정의 작성

이 섹션에서는 Request-Response Data Handler가 예상하는 구조를 나타내도록 Business Object 정의를 작성하는 방법에 대해 설명합니다. Business Object Designer를 사용하여 필요에 따라 Business Object 정의에서 속성을 추가 또는 삭제하십시오.

142 페이지의 『요청-응답 Business Object 구조 이해』에서 설명한 대로, 사용자가 작성하는 Request-Response Data Handler에서는 다음 Business Object 정의를 작성해야 합니다.

- 『최상위 레벨 Business Object 정의 작성』
- 『기타 Business Object 정의 작성』

최상위 레벨 Business Object 정의 작성

Request-Response Data Handler에 대한 최상위 레벨 Business Object 정의를 작성하려면, 다음과 같이 Business Object Designer를 사용하여 Business Object 정의를 수동으로 작성해야 합니다.

1. 최상위 레벨 Business Object 정의를 작성하십시오.

   최상위 레벨 Business Object의 구조에 대한 정보는 142 페이지의 『최상위 레벨 Business Object』를 참조하십시오.

2. 최상위 레벨 Business Object에 대한 하위 Business Object를 작성하십시오. 최상위 레벨 Business Object에서, 표 54에 표시된 Business Object에 대한 하위 오브젝트 속성을 작성하십시오.

표 54. Request-Response Data Handler의 Business Object

속성	참고	Business Object
요청 Business Object	요청에 대한 정보가 들어 있습니다.	143 페이지의 『요청 Business Object』
응답 Business Object	요청의 응답에 대한 정보가 들어 있습니다.	144 페이지의 『응답 Business Object』

기타 Business Object 정의 작성

요청 및 응답 Business Object 정의를 작성하려면, 다음 중 한 가지 방법을 사용할 수 있습니다.

- ODA(Object Discovery Agent)를 사용하여 직렬화된 데이터를 Business Object 정의로 내보낼 수 있습니다. ODA는 XML 문서를 포함한 여러 데이터 형식에 사용할 수 있습니다.

  XML ODA에 대한 정보는 241 페이지의 『XML ODA 사용』을 참조하십시오. 기타 ODA에 대한 정보는 해당 어댑터 안내서를 참조하십시오.

- Business Object Designer를 사용하여 데이터에 대한 Business Object 정의를 수동으로 작성할 수 있습니다.

  자세한 정보는 『최상위 레벨 Business Object 정의 작성』을 참조하십시오.

Request-Response Data Handler 구성

Request-Response Data Handler는 다음과 같이 Meta Object의 계층 구조에서 구성 등록 정보를 검색합니다.

- 상위 Meta Object를 사용하면 커넥터 또는 서버 액세스 인터페이스가 MIME 유형의 문서를 기반으로 Data Handler를 인스턴스화할 수 있습니다.
- 하위 Meta Object에는 Data Handler의 클래스 이름, 작성할 Business Object의 접두부 등을 포함한 문서 데이터를 처리하는 데 필요한 모든 정보가 들어 있습니다.

최상위 레벨 Meta Object 구성

상위 Meta Object에 있는 MIME 유형은 지원되는 MIME 유형 및 이러한 지원을 제공하는 Data Handler를 나타냅니다. 제공된 최상위 레벨 Meta Object 모두 Request-Response Data Handler에 대한 항목은 포함하고 있지 않습니다. 커넥터 또는 액세스 클라이언트가 Request-Response Data Handler를 사용할 수 있도록 하려면, `text/requestresponse` MIME 유형에 대한 속성을 `MO_Server_DataHandler` 또는 `MO_DataHandler_Default` 최상위 Meta Object에 추가해야 합니다. 이 속성의 유형은 `MO_DataHandler_DefaultRequestResponseConfig`여야 합니다.

다음 Business Object 정의의 단편은 `text/requestresponse` 속성에 대한 정의를 보여줍니다.

```
[Attribute]
   Name = text.requestresponse
   Type = MO_DataHandler_DefaultRequestResponseConfig
   ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
   MaxLength = 1
IsKey = false
IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
[End]
```

주: 최상위 레벨 Meta Object 및 수정 방법에 대한 자세한 정보는 29 페이지의 『최상위 레벨 Meta Object』를 참조하십시오.

하위 Meta Object 구성

Request-Response Data Handler를 구성하려면, Request-Response의 하위 Meta Object에 구성 정보가 제공되는지 확인해야 합니다.

주: Request-Response Data Handler를 구성하려면, 또한 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정해야 합니다. 자세한 정보는 141 페이지의 『Business Object 정의의 요구사항』을 참조하십시오.

Request-Response Data Handler의 경우, IBM은 기본 하위 Meta Object MO_DataHandler_DefaultRequestResponseConfig를 제공합니다. 이 Meta Object의 각 속성은 Request-Response Data Handler의 구성 등록 정보를 정의합니다. 표 55에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 55. Request-Response Data Handler의 하위 Meta Object 속성

속성 이름	설명	제공되는 기본값
BOPrefix	최상위 레벨 Business Object의 이름을 빌드하기 위해 기본 NameHandler 클래스가 사용하는 접두부. 연관된 Business Object 정의의 이름과 일치하도록 기본값을 변경해야 합니다. 속성 값은 대소문자가 구분됩니다.	REQUESTTEST
ClassName	지정된 MIME 유형으로 사용하기 위해 로드할 data-handler 클래스의 이름. 최상위 레벨 Data Handler Meta Object에는 이름이 지정된 MIME 유형과 일치하고 유형이 요청-응답 하위 Meta Object(이 테이블에서 설명)인 속성이 있어야 합니다.	com.crossworlds.DataHandlers.text.requestresponse
NameHandlerClass	요청 문서의 내용에서 최상위 레벨 Business Object의 이름을 판별하기 위해 사용하는 name-handler 클래스의 이름. 고유의 사용자 정의 네임 핸들러를 작성할 경우, 이 속성의 기본값을 변경하십시오. 자세한 정보는 102 페이지의 『사용자 정의 XML 네임 핸들러 빌드』를 참조하십시오.	com.crossworlds.DataHandlers.xml.TopElementNameHandler
RequestDataHandlerMimeType	이 Data Handler가 처리하는 요청의 MIME 유형. Request-Response Data Handler는 이 MIME 유형을 사용하여 요청 Business Object 또는 문서 처리를 위해 인스턴스화할 Data Handler를 판별합니다.	text/xml
ResponseDataHandlerMimeType	이 Data Handler가 처리하는 응답의 MIME 유형. Request-Response Data Handler는 이 MIME 유형을 사용하여 응답 Business Object 또는 문서 처리를 위해 인스턴스화할 Data Handler를 판별합니다.	text/xml
ObjectEventId	Data Handler에서는 사용되지 않지만 비즈니스 통합 시스템에서 필요로 하는 위치 표시기	없음

표 55에서 "제공되는 기본값" 열은 제공되는 Business Object에서의 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 해당 속성의 Default Value 등록 정보를 적절한 값으로 설정해야 합니다. 최소한 ClassName 및 BOPrefix 속성이 기본값을 가지고 있는지 확인해야 합니다.

MO_DataHandler_DefaultRequestResponseConfig 하위 Meta Object를 작성하려면, Business Object Designer를 사용하여 다음과 같은 형식의 Business Object 정의를 작성하십시오.

```
[BusinessObjectDefinition]
Name = MO_DataHandler_DefaultRequestResponseConfig
Version = 1.0.0

   [Attribute]
   Name = ClassName
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
```

```
IsForeignKey = false
   IsRequired = false
   DefaultValue = com.crossworlds.DataHandlers.text.requestresponse
   IsRequiredServerBound = false
[End]

   [Attribute]
   Name = NameHandlerClass
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
[End]

   [Attribute]
   Name = RequestDataHandlerMimeType
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
   IsRequired = false
   DefaultValue = text/xml
   IsRequiredServerBound = false
[End]

   [Attribute]
   Name = ResponseDataHandlerMimeType
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
   IsRequired = false
   DefaultValue = text/xml
   IsRequiredServerBound = false
[End]

   [Attribute]
   Name = BOPrefix
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
   IsRequired = false
   DefaultValue = Wrapper
   IsRequiredServerBound = false
[End]

   [Attribute]
   Name = DummyKey
Type = String
   MaxLength = 1
   IsKey = true
IsForeignKey = false
   IsRequired = false
   DefaultValue = 1
   IsRequiredServerBound = false
[End]
```

```
   [Attribute]
   Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
[End]

   [Verb]
   Name = Create
[End]

   [Verb]
   Name = Delete
[End]

   [Verb]
   Name = Retrieve
[End]

   [Verb]
   Name = Update
[End]
[End]
```

이 하위 Meta Object 파일을 둘 위치에 대한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

Requrest Data Handler로 Business Object 변환

*요청 Data Handler*는 요청을 시작하는 WebSphere Business Integration System 구성요소에 대한 데이터 변환을 처리합니다.

- 요청 처리 시, 통합 브로커는 커넥터로 전송된 요청 Business Object의 양식으로 요청을 시작합니다.

> **WebSphere InterChange Server**
>
> 통합 브로커가 ICS(InterChange Server)인 경우, 협업은 요청 Business Object를 작성하여 요청 처리를 위한 적절한 커넥터로 전송합니다.

따라서 Requrest Data Handler는 요청에 대해 Business Object에서 문자열로의 변환을 수행하여 요청 Business Object를 직렬화된 데이터로 변환합니다. 이 직렬화된 데이터의 형식은 *요청 형식*입니다. 이 형식은 커넥터의 응용프로그램(또는 액세스 클라이언트)이 입력으로 승인하는 형식입니다.

- 이벤트 처리 시, 호출 문맥(액세스 클라이언트 또는 커넥터)은 통합 브로커로 전송된 직렬화된 데이터의 양식으로 요청을 시작합니다.

WebSphere InterChange Server

통합 브로커가 ICS인 경우, 액세스 클라이언트를 사용하여 동기 이벤트 처리를 시작할 수 있습니다. 자세한 정보는 136 페이지의 『이벤트 처리 지원』을 참조하십시오.

따라서 Requrest Data Handler는 요청에 대해 문자열에서 Business Object로의 변환을 수행하여 직렬화된 데이터를 요청 Business Object로 변환합니다. 이 직렬화된 데이터의 형식은 요청 형식입니다. 이 형식은 액세스 클라이언트 또는 커넥터의 응용프로그램이 출력으로 생성하는 형식입니다.

Request-Response Data Handler는 하위 Meta Object의 `RequestDataHandlerMimeType` 등록 정보를 기반으로 Requrest Data Handler로 호출할 Data Handler를 판별합니다. `RequestDataHandlerMimeType` 등록 정보에는 Requrest Data Handler가 지원하는 MIME 유형이 있습니다. `RequestDataHandlerMimeType`이 *초기화되지 않은 경우*, Request-Response Data Handler는 오류를 로그하고 예외를 생성합니다. 그러므로 `RequestDataHandlerMimeType` 등록 정보를 *초기화해야* 합니다.

Response Data Handler로 Business Object 변환

*Response Data Handler*는 다음과 같이 요청에 응답하는 WebSphere Business Integration System 구성요소에 대한 데이터 변환을 처리합니다.

- 요청 처리 시, 커넥터의 응용프로그램은 통합 브로커로 라우트될 커넥터로 전송된 직렬화된 데이터의 양식으로 요청에 응답합니다.

WebSphere InterChange Server

통합 브로커가 ICS(InterChange Server)인 경우, 협업은 커넥터에서 응답 Business Object를 수신합니다.

따라서 Response Data Handler는 응답에 대해 문자열에서 Business Object로의 변환을 수행하여 직렬화된 데이터를 응답 Business Object로 변환합니다. 이 직렬화된 데이터의 형식은 *응답 형식*입니다. 이 형식은 커넥터의 응용프로그램(또는 액세스 클라이언트)이 출력으로 생성하는 형식입니다.

- 이벤트 처리 시, 통합 브로커는 호출 문맥(액세스 클라이언트 또는 커넥터)으로 전송된 응답 Business Object의 양식으로 요청에 응답합니다.

WebSphere InterChange Server

통합 브로커가 ICS인 경우, CIS의 협업이 응답 Business Object를 생성합니다. 그런 다음, ICS가 이 응답 Business Object를 다시 호출 문맥으로 라우트합니다. 자세한 정보는 136 페이지의 『이벤트 처리 지원』을 참조하십시오.

따라서 Response Data Handler는 응답에 대해 Business Object에서 문자열로의 변환을 수행하여 응답 Business Object를 직렬화된 데이터로 변환합니다. 이 직렬화된 데이터의 형식은 응답 형식입니다. 이 형식은 액세스 클라이언트 또는 커넥터의 응용프로그램이 입력으로 승인하는 형식입니다.

Request-Response Data Handler는 하위 Meta Object의 `ResponseDataHandlerMimeType` 등록 정보를 기반으로 Response Data Handler로 호출할 Data Handler를 판별합니다. `ResponseDataHandlerMimeType` 등록 정보에는 Response Data Handler가 지원하는 MIME 유형이 있습니다. `ResponseDataHandlerMimeType`이 *초기화되지 않은 경우*, Request-Response Data Handler는 오류를 로그하고 예외를 생성합니다. 그러므로 `ResponseDataHandlerMimeType` 등록 정보를 *초기화해야* 합니다.

오류 처리

Request-Response Data Handler가 요청을 처리할 수 없는 경우 예외가 발생합니다. Request-Response Data Handler는 오류 발생 시 Data Handler의 조치 내용 및 트랜잭션과 관련된 구성요소(예: 다른 Data Handler 또는 JCDK)가 리턴한 메시지에 대해 모두 설명하는 예외 메시지를 제공합니다. 이러한 예외는 Data Handler를 처음 호출한 구성요소로 전파됩니다(예외가 아닌 다른 데이터 구조로서).

Request-Response Data Handler는 Data Handler 프레임워크가 제공하는 서비스를 사용하여 오류를 로그하고 메시지를 추적합니다. 오류 메시지 및 경고는 ICS 로그에 기록됩니다. Data Handler는 Java 콘솔에 메시지를 로그하지 않습니다.

Request-Response Data Handler 사용자 정의

특수 네임 핸들러를 작성하여 Request-Response Data Handler를 사용자 정의할 수 있습니다. Request-Response Data Handler는 네임 핸들러를 호출하여 작성할 Business Object의 이름을 확보합니다. Data Handler는 data-handler Meta Object에 저장된 `NameHandlerClass` 속성 값을 사용하여 호출할 네임 핸들러를 결정합니다. Request-Response Data Handler에 포함되어 있는 기본 네임 핸들러는 `BOPrefix` 속

성 값을 Response Data Handler가 리턴하는 값을 Business Object 이름에 추가합니다. 네임 핸들러를 다른 방식으로 기능하도록 하려면 다음을 수행하십시오.

1. `NameHandler` 클래스를 확장하여 사용자 정의 네임 핸들러를 작성하십시오.
2. Request-Response Data Handler용 Meta Object에 있는 `NameHandlerClass` 속성의 기본값을 갱신하여 Request-Response Data Handler가 사용자 정의 Name-Handler 클래스를 사용하도록 구성하십시오.

Custom Data Handler 작성 방법에 대한 정보는 215 페이지의 『사용자 정의 네임 핸들러 빌드』를 참조하십시오.

제 6 장 FixedWidth Data Handler

FixedWidth Data Handler는 Business Object를 고정 폭 문자열 및 스트림으로 변환하고, 고정 폭 문자열 및 스트림은 Business Object로 변환합니다. 이 장에서는 FixedWidth Data Handler가 고정 폭 문서를 처리하는 방법과 Data Handler에서 처리할 Business Object를 정의하는 방법에 대해 설명합니다. 이 정보를 안내서로 사용하여 FixedWidth Data Handler 요구사항을 따르는 Business Object를 구현할 수 있습니다. 또한 FixedWidth Data Handler를 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 155 페이지의 『FixedWidth Data Handler 구성』
- 158 페이지의 『Business Object를 FixedWidth 문서로 변환』
- 159 페이지의 『FixedWidth 문서를 Business Object로 변환』

주: FixedWidth Data Handler는 `CwDataHandler.jar` 파일에 들어 있는 기본 Data Handler 중 하나입니다. 이 Data Handler 설치 방법 및 해당 소스 코드의 저장 위치에 대한 정보는 25 페이지의 제 2 장 『Data Handler 설치 및 구성』을 참조하십시오.

개요

FixedWidth Data Handler는 1차 역할로서 Business Object를 고정 폭 필드 형식의 문자열 또는 스트림으로 변환하는 데이터 변환 모듈입니다. 고정 폭 문자열 또는 스트림은 `text/fixedwidth` MIME 유형으로 직렬화된 데이터입니다. Data Handler는 고정 폭 필드를 사용하여 텍스트 데이터의 구문을 분석합니다. 필드 길이는 각 Business Object 속성의 MaxLength 등록 정보에 지정합니다. 이 등록 정보의 값은 Business Object 정의에 저장됩니다.

기본 최상위 레벨 커넥터 Meta Object(`MO_DataHandler_Default`)는 `text/fixedwidth` MIME 유형을 지원합니다. 그러므로 `MO_DataHandler_Default` Data Handler Meta Object를 지원하도록 구성된 커넥터는 FixedWidth Data Handler를 호출할 수 있습니다. 액세스 클라이언트가 InterChange Server 통합 브로커를 사용할 때 이 Data Handler를 호출할 수 있도록 하려면, `text/fixedwidth` MIME 유형을 지원하도록 `MO_Server_DataHandler` Meta Object를 수정해야 합니다. 자세한 정보는 219 페이지의 『최상위 레벨 Meta Object 수정』을 참조하십시오.

FixedWidth Data Handler의 기능

FixedWidth Data Handler는 Business Object 정의에서 속성의 MaxLength 등록 정보 값을 사용하여 데이터를 읽고 기록하는 방법을 판별합니다. MaxLength는 오른쪽 맞추기 또는 왼쪽 맞추기 텍스트에 허용하는 채우기를 포함하여, 속성 값의 최대 문자 수를 정의합니다.

줄 맞춤을 데이터에서 제거하거나 추가할 공간을 표시하는 채움 문자를 구성할 수 있습니다. 채움 문자는 Business Object를 문자열로 변환할 때 사용되고 문자열을 Business Object로 변환할 때 제거됩니다. 왼쪽 또는 오른쪽에 맞추도록 줄 맞춤 Business Object 속성을 구성할 수도 있습니다. 이렇게 하면 속성 값 데이터를 의미있는 문자로 보유할 수 있게 됩니다. 표 56에서는 Alignment Meta Object 속성의 값에 대해 설명합니다.

표 56. 줄 맞춤 Meta Object 속성의 값

값	설명
LEFT	왼쪽을 자르고 오른쪽을 채웁니다.
RIGHT	오른쪽을 자르고 왼쪽을 채웁니다.
BOTH	문자열을 Business Object로 변환할 때, 오른쪽과 왼쪽을 모두 자릅니다. Business Object를 문자열로 변환할 경우에는 오른쪽을 채움 문자로 채웁니다.

또한 Truncation Meta Object 속성을 사용하여 Business Object 값을 MaxLength로 자르거나, 속성 값이 MaxLength보다 긴 문자열일 경우, 오류를 생성하도록 FixedWidth Data Handler를 구성할 수 있습니다. Business Object 이름, verb, 그리고 카디널리티 1 또는 n 하위 오브젝트에 대한 Business Object 계수에 사용할 길이 크기도 설정합니다.

String 속성의 MaxLength 등록 정보 값을 설정하려면 Business Object Designer를 사용하십시오. 다른 유형(예: 정수, Double 등)의 MaxLength 등록 정보 값을 변경하려면, Business Object 정의를 파일로 저장하고, 수동으로 파일을 편집한 후 수정한 정의를 비즈니스 통합 시스템에 가져와야 합니다.

Business Object 및 FixedWidth 문자열 처리

FixedWidth Data Handler는 표 57에 나열된 조작을 수행합니다.

표 57. FixedWidth Data Handler의 데이터 조작

Data Handler 조작	자세한 정보
호출자로부터 Business Object를 수신하고 Business Object를 FixedWidth 문자열 또는 스트림으로 변환한 후 FixedWidth 데이터를 호출자에게 전달합니다.	158 페이지의 『Business Object를 FixedWidth 문서로 변환』
호출자로부터 문자열 또는 스트림을 수신하고 Business Object를 빌드한 후 Business Object를 호출자에게 리턴합니다.	159 페이지의 『FixedWidth 문서를 Business Object로 변환』

FixedWidth Data Handler 구성

FixedWidth Data Handler를 구성하려면 다음 단계를 수행하십시오.

- FixedWidth 하위 Meta Object 속성에 대한 적절한 값을 입력하십시오.
- Business Object 정의가 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정하십시오.

이러한 단계는 각각 다음 섹션에 자세히 설명되어 있습니다.

FixedWidth 하위 Meta Object 구성

FixedWidth Data Handler를 구성하려면, FixedWidth 하위 Meta Object에서 구성 정보가 제공되는지 확인해야 합니다. FixedWidth Data Handler의 경우, IBM은 MO_DataHandler_DefaultFixedWidthConfig 하위 Meta Object를 제공합니다. 이 Meta Object에 있는 각각의 속성이 FixedWidth Data Handler의 구성 등록 정보를 정의합니다. 표 58에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 58. FixedWidth Data Handler의 하위 Meta Object 속성

Meta Object 속성 이름	**의미**	**제공되는 기본값**
ClassName	최상위 레벨 Data Handler Meta Object의 속성 이름과 일치하는, MIME 유형으로 사용하기 위해 로드할 Data Handler 클래스의 이름. 이 속성은 유형으로 FixedWidth 하위 Meta Object를 수반합니다.	com.crossworlds.DataHandlers.text.fixedwidth
Alignment	PadCharacter 속성을 추가 또는 제거합니다. 이벤트 처리의 경우에는 채움 문자는 잘리고, 요청 처리의 경우 채움 문자가 추가됩니다. 가능한 값은 BOTH, LEFT 및 RIGHT입니다. 예를 들어, "LEFT" 맞추기는 Business Object의 값이 해당되는 속성 값 공간의 왼쪽 끝으로 이동한다는 것 의미합니다. 이벤트 공고에 사용하는 "BOTH" 맞추기는 왼쪽 및 오른쪽 모두에서 채움 문자가 잘린다는 것을 의미합니다. 요청 처리에 사용하는 "RIGHT" 맞추기는 오른쪽이 채움 문자로 채워진다는 것을 의미합니다.	BOTH
BOCountSize	처리하는 총 Business Object 수에 대해 할당되는 공간을 지정합니다.	3
BONameSize	Business Object 이름에 대해 할당되는 공간을 지정합니다.	50
BOVerbSize	verb에 대해 할당되는 공간을 지정합니다.	20
CxBlank	Business Object에서 변환할 때, FixedWidth Data Handler는 속성 값이 CxBlank인 Business Object 속성을 발견할 때마다 CxBlank Meta Object 속성의 Default Value 등록 정보에 구성된 값을 고정 폭 문서에 기록합니다. Business Object로 변환할 때, FixedWidth Data Handler는 고정 폭 문서에서 이 CxBlank Meta Object 속성의 값을 발견할 때마다 CxBlank Meta Object 속성의 Default Value 등록 정보에 구성된 값을 Business Object 속성 값에 지정합니다. Business Object에는 실행할 때 CxBlank값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.	CxBlank 값

표 58. FixedWidth Data Handler의 하위 Meta Object 속성 (계속)

Meta Object 속성 이름	의미	제공되는 기본값
CxIgnore	Business Object로부터 변환할 때, FixedWidth Data Handler는 속성 값이 CxIgnore인 Business Object 속성을 발견할 때마다 CxIgnore Meta Object 속성의 Default Value 등록 정보에 구성된 값을 고정 폭 문서에 기록합니다. Business Object로 변환할 때, FixedWidth Data Handler는 고정 폭 문서에서 이 CxIgnore Meta Object 속성의 값을 발견할 때마다 CxIgnore Meta Object 속성의 Default Value 등록 정보에 구성된 값을 Business Object 속성 값에 지정합니다. Business Object에는 실행할 때 CxIgnore 값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.	CxIgnore 값
DummyKey	비즈니스 통합 시스템에서 필요로 하는 키 속성	1
OmitObjectEventId	Business Object에서 문자열로, 그리고 문자열에서 Business Object로의 변환에 ObjectEventId 데이터의 포함 여부를 판별하기 위한 부울 값	false
PadCharacter	출 맞춤을 위해 추가 또는 제거할 공간을 표시합니다. 채움 문자로 모든 문자를 지정할 수 있습니다.	#
Truncation	문자 제거를 설정합니다. true일 경우, Business Object에서 MaxLength보다 큰 속성 값은 요청 처리 중 MaxLength로 잘립니다. false이면 오류가 로그에 기록되고 형식화가 중지됩니다.	false
ObjectEventId	Data Handler에서는 사용되지 않지만 비즈니스 통합 시스템에서 필요로 하는 위치 표시기	없음

표 58에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 해당 속성의 Default Value 등록 정보를 시스템과 FixedWidth 문서에 적절한 값으로 설정해야 합니다. 최소한 ClassName 속성이 기본값을 가지고 있는지 확인해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

Business Object 요구사항

FixedWidth Data Handler는 처리하는 Business Object 구조에 대한 사항을 가정합니다. 그러므로 FixedWidth Data Handler를 사용한 변환을 위해 Business Object를 작성할 경우 다음 규칙을 따라야 합니다.

- Business Object 정의에 있는 모든 속성이 적절한 MaxLength 등록 정보 값을 가지고 있는지 확인하십시오. 그러면 FixedWidth Data Handler는 Business Object에서 FixedWidth 형식으로, 그리고 FixedWidth 형식에서 Business Object로의 데이터 변환을 적절하게 처리할 수 있습니다.
- ObjectEventId 속성이 Business Object 계층 구조의 모든 레벨에서 모든 Business Object에 포함되어 있는지 확인하십시오. Business Object Designer는 Business Object 정의를 저장할 때 자동으로 이를 수행하지만, 요구사항을 만족하는지 확인해야 합니다.

Business Object 구조

FixedWidth Data Handler의 Business Object 구조에 대한 요구사항은 없습니다. Data Handler는 MaxLength 속성 등록 정보에 값이 있는 경우 모든 Business Object를 처리할 수 있습니다.

Data Handler가 처리하는 Business Object는 비즈니스 통합 시스템에서 허용하는 임의의 이름을 가질 수 있습니다.

Business Object 속성 등록 정보

Business Object 아키텍처에는 속성에 적용하는 다양한 등록 정보가 있습니다. 표 59에서는 FixedWidth Data Handler가 등록 정보를 해석하는 방법과 Business Object를 수정할 때 이 등록 정보를 설정하는 방법에 대해 설명합니다.

표 59. FixedWidth Data Handler를 사용하여 변환되는 Business Object의 속성 등록 정보

등록 정보 이름	설명
Name	각 Business Object 속성에는 고유한 이름이 있어야 합니다.
Type	각 Business Object 속성은 유형(예: 정수, 문자열 또는 포함된 하위 Business Object의 유형)을 가지고 있어야 합니다.
Key	FixedWidth Data Handler에서는 사용하지 않습니다.
MaxLength	속성 값이 포함되는 필드의 폭을 판별합니다.
Foreign Key	FixedWidth Data Handler에서는 사용하지 않습니다.
Required	FixedWidth Data Handler에서는 사용하지 않습니다.
Default Value	FixedWidth Data Handler에서는 사용하지 않습니다.
Cardinality	카디널리티 1 및 카디널리티 n 오브젝트를 지원합니다.

Business Object 응용프로그램 특정 정보

FixedWidth Data Handler는 Business Object 또는 해당 속성에서 응용프로그램 특정 정보를 요구하지 않습니다. 그러나 Data Handler는 cw_mo_ 태그의 존재를 확인합니다. 이 태그는 커넥터가 사용하는 하위 Meta Object를 표시하기 위해 Business Object가 사용할 수 있습니다. Data Handler는 Business Object의 응용프로그램 특정 정보에서 cw_mo_ 태그에 의해 식별되는 속성을 무시합니다. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.

기존 Business Object 정의 사용

FixedWidth Data Handler는 Business Object가 Data Handler 요구사항을 따르는 양식으로 데이터를 제공할 경우, Business Object를 FixedWidth 문자열로 변환할 수 있습니다. FixedWidth Data Handler의 한 가지 요구사항은 각 Business Object 속성에 올바른 MaxLength 값을 지정하는 것입니다. 기존의 Business Object를 수정하여 MaxLength에 적절한 값을 지정해야 할 수도 있습니다.

이 요구사항을 만족하는 기존 Business Object를 FixedWidth Data Handler로 변환할 수는 있지만, 각 유형의 데이터가 처리되도록 사용자의 Business Object를 작성하는 것이 좋습니다. 샘플 Business Object를 사용하거나 다른 구현에서 샘플 응용프로

그램을 지원하기 위해 개발된 Business Object를 사용할 경우, 개발 중인 구현에 필요한 속성만 포함하도록 필요에 따라 정의를 수정하도록 하십시오.

그러므로 기존의 Business Object를 거의 데이터에 해당하는 양식으로 변환하려면, 응용프로그램에서 요구하는 데이터와 Data Handler에서 요구하는 정보만 제공하도록 Business Object를 수정하십시오. FixedWidth Data Handler와 함께 기존 Business Object를 사용하려면 다음을 수행하십시오.

1. 대상 응용프로그램의 기능 분석을 수행하고, 결과를 기존 Business Object와 비교하여 Business Object 정의의 필수 필드를 판별하십시오.
2. Business Object Designer를 사용하여 필요에 따라 Business Object 정의에서 속성을 추가 또는 삭제하십시오.

Business Object를 FixedWidth 문서로 변환

Business Object를 FixedWidth 문서로 변환하려면, FixedWidth Data Handler는 순차적으로 Business Object의 속성을 루핑합니다. Business Object와 자체 하위 오브젝트에 속성이 표시되는 순서대로 순환하여 고정 폭 문자열에 필드를 생성합니다.

FixedWidth Data Handler는 다음과 같이 Business Object를 FixedWidth 문서로 처리합니다.

1. Data Handler는 Business Object의 데이터를 포함하는 고정 폭 문자열을 작성합니다.
2. Data Handler는 Business Object 이름을 추가하고, 고정 폭 문자열에 verb를 추가합니다. Business Object의 이름을 인수로 변환 메소드에 지정할 수 있습니다.
3. Data Handler는 Business Object 정의에서 응용프로그램 고유 정보를 조사하여 하위 Meta Object(이름이 Business Object 응용프로그램 고유 정보의 `cw_mo_` 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 이러한 속성을 FixedWidth 문서에 포함하지 않습니다. `cw_mo_` 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
4. Data Handler는 Meta Object 속성인 `OmitObjectEventId`를 찾습니다. 이 속성이 `true`이면, Data Handler는 Business Object의 `ObjectEventId` 데이터를 `FixedWidth` 문서에 포함하지 않습니다.
5. Data Handler는 순서대로 나머지 Business Object 속성을 루핑하여, 각 단순 속성의 값을 문자열에 추가합니다. 배열 속성의 경우, Data Handler는 다음을 수행합니다.
   - 속성이 단일 카디널리티 속성을 표시할 경우, Data Handler는 문자열에 속성 이름 및 계수 1을 추가하고, 순환 방식으로 하위 Business Object를 처리하여 각 속성 값을 문자열에 추가합니다.

- 속성이 단일 카디널리티 배열을 표시할 경우, Data Handler는 문자열에 속성 이름 및 하위 오브젝트 계수를 추가하고, 순환 방식으로 각각의 하위 Business Object를 처리하여 각 속성 값을 문자열에 추가합니다.

6. Data Handler가 변환을 완료하면, 직렬화된 데이터를 호출자에게 리턴합니다. Data Handler는 호출자가 요청하는 양식(`String` 또는 `InputStream`)으로 데이터를 리턴합니다.

주: `Truncation` Meta Object 속성의 Default Value 등록 정보 값이 `true`일 경우, Business Object에서 길이가 `MaxLength`보다 긴 속성 값은 요청 처리 중 `MaxLength`로 잘립니다. `Truncation`을 `false`로 설정하고 속성 값 길이가 `MaxLength`보다 길면, 형식화는 중지되고 오류가 로그에 기록됩니다.

FixedWidth 문서를 Business Object로 변환

이 섹션에서는 FixedWidth Data Handler가 FixedWidth 문서를 Business Object로 변환하는 방법에 대한 다음과 같은 정보를 제공합니다.

- 『FixedWidth 문자열 요구사항』
- 160 페이지의 『직렬화된 데이터 처리』

FixedWidth 문자열 요구사항

문자열을 Business Object로 변환할 때, FixedWidth Data Handler는 다음 사항을 가정합니다.

- Business Object 이름은 데이터에서 첫 번째 필드로 표시됩니다.
- verb는 데이터에서 두 번째 필드로 표시됩니다.
- 모든 속성은 Business Object 정의에 표시되는 순서대로 제공됩니다.
- `ObjectEventId` 속성은 각 Business Object에 표시됩니다. `ObjectEventId` 항목은 값이 `CxIgnore`인 경우에도 항상 Business Object에 대해 필요합니다. Data Handler가 런타임 시 이 항목을 사용하여 Business Object 인스턴스를 구별하기 때문입니다.

고정 폭 문자열의 형식은 다음과 같습니다.

```
[Bus_Obj_Name<blank_space_padding_for_size>]
[Verb<blank_space_padding_for_size>]
[Attr1<blank_space_padding_for_size>]
[Attr2...<blank_space_padding_for_size>]
[Number-of-child-object_instances<blank_space_padding_for_size>]
[Child_Object_Name<blank_space_padding_for_size>]
[Child_Object_Verb<blank_space_padding_for_size>]
[Child_Object_Attr1<blank_space_padding_for_size>]
[Child_Object_Attr2...<blank_space_padding_for_size>]
<EndBO:Bus_Obj_Name>
```

이 형식에서, 처음 두 필드(Bus_Obj_Name 및 Verb)는 FixedWidth 하위 Meta Object에서 BONameSize 및 BOVerbSize 속성에 지정된 길이의 필드를 작성하도록 채움 문자로 채워집니다. 그 다음의 속성은 각 Business Object 속성마다 MaxLength 등록 정보에 지정된 길이의 필드를 작성하도록 채움 문자가 채워집니다.

FixedWidth Data Handler에서 사용되는 필드는 해당 속성에 대해 CxIgnore가 가능한 값일 경우, 최소한 8자 이상이어야 합니다. 속성이 CxIgnore 값을 가지지 않는 것이 확실하면, MaxLength는 8자 미만이 될 수 있습니다.

커넥터가 고정 폭 형식의 파일을 읽는 경우, CxIgnore 및 CxBlank Meta Object 속성은 원하는 값을 생성하도록 구성되어야 합니다. 두 문자열은 최소 MaxLength 속성에 영향을 줍니다. MaxLength의 최소값은 두 속성에 맞게 조정해야 합니다.

직렬화된 데이터 처리

FixedWidth Data Handler는 다음과 같이 FixedWidth 문서를 Business Object로 처리합니다.

1. Data Handler는 데이터를 포함하는 Business Object를 작성합니다. Business Object 유형은 커넥터에 의해 변환 메소드로 전달되거나, Data Handler가 문자열의 첫 번째 필드에서 Business Object 이름을 추출합니다. 유형 매개변수와 데이터의 내용이 일치하지 않을 경우, Data Handler는 오류를 생성합니다.
2. Data Handler는 Business Object에서 verb를 설정합니다. Data Handler는 최상위 레벨 Business Object의 verb가 데이터에서 두 번째 필드에 있는 것으로 간주합니다.
3. Data Handler는 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 cw_mo_ 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 Business Object의 이러한 속성을 채우기 위해 처리를 수행하지 *않습니다*. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
4. Data Handler는 Meta Object 속성인 OmitObjectEventId를 찾습니다. 이 속성이 true이면, Data Handler는 ObjectEventId 속성을 채우기 위한 처리를 수행하지 않습니다.
5. 나머지 Business Object 속성을 설정하려면, Data Handler는 Business Object 정의에 지정된 대로 각 속성의 MaxLength를 기초로 데이터의 구문을 분석합니다. Data Handler는 데이터에서 속성 값을 추출하고 Business Object에 있는 단순 속성의 값을 채웁니다.

   Data Handler는 다음과 같이 배열 속성을 처리합니다.

   - 속성이 단일 카디널리티일 경우, Data Handler는 순환 방식으로 속성 목록의 구문을 분석하고 속성 값을 설정한 후 하위 Business Object를 상위 Business Object에 추가합니다.

- 속성이 다중 카디널리티 배열일 경우, Data Handler는 각 하위 속성 목록에서 순환 방식으로 속성의 구문을 분석하고 하위 Business Object를 상위 Business Object에 추가합니다.

제 7 장 Delimited Data Handler

Delimited Data Handler는 Business Object를 구분된 형식의 문자열 및 스트림으로 변환하고, 구분된 형식의 문자열 및 스트림은 Business Object로 변환합니다. 이 장에서는 Delimited Data Handler가 구분된 데이터를 처리하는 방법과 Data Handler에서 처리할 Business Object를 정의하는 방법에 대해 설명합니다. 이 정보를 안내서로 사용하여 기존 Business Object를 수정하거나 Data Handler 요구사항을 따르는 새 Business Object를 구현할 수 있습니다. 또한 Delimited Data Handler를 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 164 페이지의 『Delimited Data Handler 구성』
- 168 페이지의 『Business Object를 Delimited 데이터로 변환』
- 169 페이지의 『Delimited 데이터를 Business Object로 변환』

주: Delimited Data Handler는 `CwDataHandler.jar` 파일에 들어 있는 기본 Data Handler 중 하나입니다. 이 Data Handler 설치 방법 및 해당 소스 코드의 저장 위치에 대한 정보는 25 페이지의 제 2 장 『Data Handler 설치 및 구성』을 참조하십시오.

개요

Delimited Data Handler는 Business Object를 구분된 형식의 문자열 또는 스트림으로 변환하는 1차 역할인 데이터 변환 모듈입니다. 구분된 형식의 문자열 또는 스트림은 `text/delimited` MIME 유형으로 직렬화된 데이터입니다. Data Handler는 Business Object 데이터의 개별 필드를 구분하는 지정된 분리문자를 기초로 텍스트 데이터를 구문 분석합니다. 이러한 유형의 데이터 변환은 기본적으로 시스템 읽기 효율성이 중요한 경우에 사용됩니다.

기본 최상위 레벨 커넥터 Meta Object(`MO_DataHandler_Default`)는 `text/delimited` MIME 유형을 지원합니다. 그러므로 `MO_DataHandler_Default` Meta Object를 사용하도록 구성된 커넥터는 Delimited Data Handler를 호출할 수 있습니다. InterChange Server가 통합 브로커이고 액세스 클라이언트가 이 Data Handler를 호출할 수 있어야 할 경우, `text/delimited` MIME 유형을 지원하도록 최상위 레벨 서버 Meta Object(`MO_Server_DataHandler`)를 수정해야 합니다. 자세한 정보는 219 페이지의 『최상위 레벨 Meta Object 수정』을 참조하십시오.

Delimited Data Handler의 기능

Delimited Data Handler는 다음 문자열을 설정할 수 있게 합니다.

- 분리문자 -- Data Handler는 분리문자를 사용하여 구분된 데이터의 여러 필드를 구분합니다. Delimiter Meta-Object 속성을 사용자 데이터에 대해 원하는 분리분자로 설정할 수 있습니다. 기본적으로, Data Handler는 분리문자로 틸데(~)를 사용합니다. 이는 데이터 읽기 및 쓰기 방법을 판별하기 위한 속성 등록 정보의 크기 MaxLength입니다. MaxLength는 오른쪽 맞추기 또는 왼쪽 맞추기 텍스트에 허용할 채움 문자를 포함하여, 속성 값의 최대 문자 수를 정의하는 Business Object 속성 등록 정보입니다. 저장소에 있는 Business Object의 정의에서 MaxLength를 읽습니다. 이에 따라, Business Object의 기본 요구사항은 각 문자열 속성의 MaxLength를 적절하게 설정하는 것입니다.
- 이스케이프 문자열 -- Data Handler는 이스케이프 문자열을 사용하여 분리문자 및 이스케이프 문자열을 이스케이프하기 위한 문자열을 구성하는 것입니다. 이스케이프 문자열을 사용하면 속성 값 데이터에 분리문자 및 이스케이프와 유사한 문자열을 포함할 수 있습니다. Escape meta-object 속성을 이스케이프 문자열 구성을 위해 설정할 수 있습니다. 기본적으로, 기본 Data Handler는 이스케이프 문자열로 백슬래시(\)를 사용합니다.

Business Object 및 문자열 처리

Delimited Data Handler는 표 60에 나열된 조작을 수행합니다.

표 60. Delimited Data Handler의 데이터 조작

Data Handler 조작	자세한 정보
호출자로부터 Business Object를 수신하고 Business Object를 Delimited 문자열 또는 스트림으로 변환한 후 직렬화된 데이터를 호출자에게 전달합니다.	168 페이지의 『Business Object를 Delimited 데이터로 변환』
호출자로부터 문자열 또는 스트림을 수신하고 Business Object를 빌드한 후 Business Object를 호출자에게 리턴합니다. 호출자로부터 Delimited 문자열 또는 스트림을 수신하고 Business Object를 빌드한 후 Business Object를 호출자에게 리턴합니다.	169 페이지의 『Delimited 데이터를 Business Object로 변환』

Delimited Data Handler 구성

Delimited Data Handler를 구성하려면 다음 단계를 수행하십시오.

- Delimited 하위 Meta Object 속성에 대한 적절한 값을 입력하십시오.
- Business Object 정의가 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정하십시오.

이러한 단계는 각각 다음 섹션에 자세히 설명되어 있습니다.

Delimited 하위 Meta Object 구성

Delimited Data Handler를 구성하려면, Delimited 하위 Meta Object에서 구성 정보가 제공되는지 확인해야 합니다. Delimited Data Handler의 경우, IBM은 MO_DataHandler_DefaultDelimitedConfig 하위 Meta-Object를 제공합니다. 이 Meta

Object에 있는 각각의 속성이 Delimited Data Handler의 구성 등록 정보를 정의합니다. 표 61에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 61. Delimited Data Handler의 하위 Meta Object 속성

Meta Object 속성 이름	**의미**	**제공되는 기본값**
ClassName	지정된 MIME 유형으로 사용하기 위해 로드할 Data Handler 클래스의 이름. 최상위 레벨 Data Handler Meta Object에는 이름이 지정된 MIME 유형과 일치하고 유형이 Delimited 하위 Meta Object(표 61 참조)인 속성이 있습니다.	com.crossworlds.DataHandlers.text.delimited
CxBlank	특수 Business Object 속성 값인 Blank(CxBlank 상수)에 맞도록 Delimited 데이터에 동일한 값을 설정하십시오. 자세한 정보는 167 페이지의 『CxBlank』를 참조하십시오.	CxBlank 상수 (공백)
CxIgnore	특수 Business Object 속성 값인 Ignore(CxIgnore 상수)에 맞도록 Delimited 데이터에 동일한 값을 설정하십시오. 자세한 정보는 166 페이지의 『CxIgnore』를 참조하십시오.	CxIgnore 상수 (공백 문자열)
Delimiter	Business Object 데이터를 파일에 기록할 때 Business Object 속성에서 값을 구분하기 위해 사용되거나, 파일을 Business Object로 변환할 때 속성에 해당하는 데이터의 필드를 구분하는 것으로 간주하는 문자열. 이 값에는 여러 문자가 있을 수 있습니다.	~(틸데)
DummyKey	비즈니스 통합 시스템에서 필요로 하는 키 속성	1
Escape	Business Object 속성 값에서 발생할 경우 분리문자 또는 이스케이프 문자를 이스케이프하기 위해 사용하는 문자열. 이 값의 길이는 1자입니다.	\(백슬래시)
OmitObjectEventId	Business Object에서 문자열로, 그리고 문자열에서 Business Object로의 변환에 ObjectEventId 데이터의 포함 여부를 판별하기 위한 부울 값	false
ObjectEventId	비즈니스 통합 시스템에서 요구하는 Placeholder 속성	없음

표 61에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 해당 속성의 Default Value 등록 정보를 시스템과 Delimited 문서에 적절한 값으로 설정해야 합니다. 최소한 ClassName 속성이 기본값을 가지고 있는지 확인해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

Business Object 요구사항

Delimited Data Handler는 처리하는 Business Object에 대한 사항을 가정합니다. 그러므로 Delimited Data Handler를 사용한 변환을 위해 Business Object를 작성할 경우 다음 규칙을 따라야 합니다.

- Delimited Data Handler가 적절하게 데이터를 변환할 수 있도록 모든 Business Object 속성이 이름을 가지고 있는지 확인하십시오.
- ObjectEventId 속성이 Business Object 계층 구조의 모든 레벨에서 모든 Business Object에 포함되어 있는지 확인하십시오. Business Object Designer는 Business Object 정의를 저장할 때 자동으로 이를 수행하지만, 요구사항을 만족하는지 확인해야 합니다.

하위 Meta Object에 있는 Delimiter 속성은 속성 필드를 구분하기 위해 사용하는 분리문자를 구성합니다. 기본값은 틸데(~)입니다.

분리문자 및 이스케이프 문자열을 이스케이프하기 위한 문자열을 구성하기 위해 하위 Meta Objext 속성 Escape를 설정할 수 있습니다. 이스케이프 문자열을 사용하면 속성 값 데이터에 분리문자 및 이스케이프와 유사한 문자열을 포함할 수 있습니다.

Business Object 구조

Delimited Data Handler의 Business Object 구조에 대한 요구사항은 없습니다. Data Handler는 모든 Business Object를 처리할 수 있습니다.

Data Handler가 처리하는 Business Object는 통합 브로커에서 허용하는 모든 이름을 가질 수 있습니다.

Business Object 속성 등록 정보

Business Object 구조에는 속성에 적용하는 다양한 등록 정보가 있습니다. 표 62에서는 Delimited Data Handler가 몇 개의 등록 정보를 해석하는 방법과 Business Object를 수정할 때 이 등록 정보를 설정하는 방법에 대해 설명합니다.

표 62. Delimited Data Handler를 사용하여 변환되는 Business Object의 속성 등록 정보

등록 정보 이름	설명
Name	모든 Business Object 속성에는 고유한 이름이 있어야 합니다.
Type	각 Business Object 속성은 유형(예: 정수, 문자열 또는 포함된 하위 Business Object의 유형)을 가지고 있어야 합니다.
Key	Delimited Data Handler에서는 사용하지 않습니다.
MaxLength	Delimited Data Handler에서는 사용하지 않습니다.
Foreign Key	Delimited Data Handler에서는 사용하지 않습니다.
Required	Delimited Data Handler에서는 사용하지 않습니다.
Default Value	Delimited Data Handler에서는 사용하지 않습니다.
Cardinality	카디널리티 1 및 카디널리티 n 오브젝트를 지원합니다.

Business Object의 속성에는 특수 값 CxIgnore 또는 CxBlank이 있을 수 있습니다. Delimited Data Handler는 속성에 이런 값이 있을 때, 다음 섹션에 설명된 대로 특수 처리 단계를 수행합니다.

CxIgnore: CxIgnore Meta Object 속성은 Delimited 데이터에서 Ignore 속성 값(CxIgnore 상수)에 해당하는 값을 설정합니다. 기본적으로, CxIgnore Meta Object 속성은 CxIgnore 상수의 값으로 설정됩니다. Data Handler는 다음과 같이 CxIgnore Meta Object 속성을 사용합니다.

- Business Object*로부터* 변환할 때, Delimited Data Handler는 속성 값이 CxIgnore 상수인 Business Object 속성을 발견할 때마다 CxIgnore Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 Delimited 데이터에 기록합니다.

- Business Object로 변환할 때, Delimited Data Handler는 Delimited 데이터에서 CxIgnore Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 발견할 때마다 CxIgnore 상수를 Business Object 속성 값에 지정합니다.

주: Business Object에는 실행할 때 CxIgnore 값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.

CxBlank: CxBlank Meta Object 속성은 Delimited 데이터에서 Blank 속성 값(CxBlank 상수)에 해당하는 값을 설정합니다. 기본적으로, CxBlank Meta Object 속성은 CxBlank 상수의 값으로 설정됩니다. Data Handler는 다음과 같이 CxBlank Meta Object 속성을 사용합니다.

- Business Object*로부터* 변환할 때, Delimited Data Handler는 속성 값이 CxBlank 상수인 Business Object 속성을 발견할 때마다 CxBlank Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 Delimited 데이터에 기록합니다.
- Business Object로 변환할 때, Delimited Data Handler는 Delimited 데이터에서 이 CxBlank Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 발견할 때마다 CxBlank 상수를 Business Object 속성 값에 지정합니다.

주: Business Object에는 실행할 때 CxBlank값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.

Business Object 응용프로그램 특정 정보

Delimited Data Handler는 Business Object 또는 해당 속성에서 응용프로그램 특정 정보를 요구하지 않습니다. 그러나 Data Handler는 cw_mo_ 태그의 존재를 확인합니다. 이 태그는 커넥터가 사용하는 하위 Meta Object를 표시하기 위해 Business Object가 사용할 수 있습니다. Data Handler는 Business Object의 응용프로그램 특정 정보에서 cw_mo_ 태그에 의해 식별되는 속성을 무시합니다. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.

기존 Business Object 정의 사용

Delimited Data Handler는 Business Object가 Data Handler 요구사항을 따르는 양식으로 데이터를 제공할 경우, Business Object를 Delimited 문자열로 변환할 수 있습니다. Delimited Data Handler의 한 가지 요구사항은 Data Handler가 구분된 파일에서 읽어야 할 경우, 각각의 개별 필드는 구성된 분리문자로 구분하는 것입니다.

이 요구사항을 만족하는 기존 Business Object를 Delimited Data Handler로 변환할 수는 있지만, 각 유형의 데이터가 처리되도록 사용자의 Business Object를 작성하는 것이 좋습니다. 샘플 Business Object를 사용하거나 다른 구현에서 샘플 응용프로그램을 지원하기 위해 개발된 Business Object를 사용할 경우, 개발 중인 구현에 필요한 속성만 포함하도록 필요에 따라 정의를 수정하도록 하십시오.

그러므로 기존의 Business Object를 거의 데이터에 해당하는 양식으로 변환하려면, 응용프로그램에서 요구하는 데이터와 Data Handler에서 요구하는 정보만 제공하도록 Business Object를 수정하십시오. Delimited Data Handler와 함께 기존 Business Object를 사용하려면 다음을 수행하십시오.

1. 대상 응용프로그램의 기능 분석을 수행하고, 결과를 기존 Business Object와 비교하여 Business Object 정의의 필수 필드를 판별하십시오.
2. Business Object Designer를 사용하여 필요에 따라 Business Object 정의에서 속성을 추가 또는 삭제하십시오.

Business Object를 Delimited 데이터로 변환

Business Object를 문자열로 변환하려면, Delimited Data Handler는 순차적으로 Business Object의 속성을 루핑합니다. Business Object 및 해당되는 하위 오브젝트에 속성이 표시되는 순서대로 Delimited 형식을 반복적으로 생성합니다. Business Object의 이름은 인수로 변환 메소드에 전달됩니다.

Delimited Data Handler는 다음과 같이 Business Object를 구분된 데이터로 처리합니다.

1. Data Handler는 Business Object의 데이터를 포함하는 문자열을 작성합니다.
2. Data Handler는 문자열에서 첫 번째 토큰으로 Business Object 이름을 추가하고, 문자열에서 두 번째 토큰으로 verb를 추가합니다.
3. Data Handler는 Business Object 정의에서 응용프로그램 특정 정보를 조사하여 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 `cw_mo_` 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 이러한 속성을 구분된 데이터에 포함하지 *않습니다*. `cw_mo_` 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
4. Data Handler는 Meta Object 속성인 `OmitObjectEventId`를 찾습니다. 이 속성이 `true`이면, Data Handler는 Business Object의 `ObjectEventId` 데이터를 구분된 데이터에 포함하지 않습니다.
5. Data Handler는 순서대로 나머지 Business Object 속성을 루핑하여, 각 단순 속성이 값을 문자열에 추가하고 구성된 분리문자를 각 속성 다음에 추가합니다. 컨테이너 속성의 경우, Data Handler는 다음을 수행합니다.
   - 속성이 카디널리티 1 컨테이너일 경우, Data Handler는 문자열에 속성 계수를 추가하고, 순환 방식으로 하위 Business Object를 처리하여 각 속성의 값을 추가합니다.
   - 속성이 카디널리티 n 컨테이너일 경우, Data Handler는 컨테이너에 있는 하위 오브젝트 수를 문자열에 추가하고, 순환 방식으로 하위 Business Object를 처리하여 각 속성의 값을 문자열에 추가합니다.

6. Data Handler가 변환을 완료하면, 직렬화된 데이터를 호출자에게 리턴합니다. Data Handler는 호출자가 요청하는 양식(String 또는 InputStream)으로 데이터를 리턴합니다.

Data Handler가 생성하는 형식은 다음 패턴을 따릅니다.

```
Bus_Obj_Name<delimiter>Verb<delimiter>Attr1<delimiter>Attr2<delimiter>
Number_of_child_object_instances<delimiter>Child_Object_Name<delimiter>Verb
<delimiter>Attr1<delimiter>Attr2<EndBO:Bus_Obj_Name>
```

이스케이프 문자열은 속성 값에서 분리문자와 유사한 문자열의 앞에 추가됩니다. 이스케이프 문자열은 하위 Meta Object의 `Escape` 속성을 사용하여 구성합니다.

Delimited 데이터를 Business Object로 변환

이 섹션에서는 Delimited Data Handler가 Delimited 데이터를 Business Object로 변환하는 방법에 대한 다음과 같은 정보를 제공합니다.

- 『Delimited 문자열 요구사항』
- 170 페이지의 『직렬화된 데이터 처리』

Delimited 문자열 요구사항

문자열 또는 스트림을 변환할 때, Delimited Data Handler는 다음 사항을 가정합니다.

- 데이터에는 `Delimiter` Meta Object 속성에 지정된 분리문자가 포함되어 있습니다.
- Business Object 이름은 데이터의 첫 번째 필드에 표시됩니다.
- verb는 데이터에서 두 번째 필드로 표시됩니다.
- 속성은 Business Object 정의에 표시되는 순서로 되어 있습니다.
- 하위 컨테이너의 모든 오브젝트 유형은 동일합니다.
- 데이터에는 각 카디널리티 `n` 컨테이너에 있는 하위 오브젝트 수를 표시하는 토큰이 있습니다.
- `ObjectEventId` 속성은 각 Business Object에 표시됩니다. `ObjectEventId` 항목은 값이 `CxIgnore`인 경우에도 항상 Business Object에 대해 필요합니다. Data Handler가 실행할 때 이 항목을 사용하여 Business Object 인스턴스를 구별하기 때문입니다.

데이터에 여러 개의 Business Object가 있을 경우, Business Object 사이에 새로운 문자(예: 공백, 탭, 줄 바꾸기 또는 캐리지 리턴)를 넣지 마십시오.

Delimited Data Handler가 Delimited 형식으로 파일을 읽을 때는, 다음 특수 처리 단계를 수행하여 Business Object 속성에 `CxIgnore` 또는 `CxBlank` 속성 값을 지정합니다.

- CxIgnore 상수(null)를 Delimited 데이터에 다음 조건 중 어느 것이든 나타날 때마다 해당 속성 값으로 지정합니다.
  - CxIgnore Meta Object 속성 값(Default Value 등록 정보에 구성된 값)
  - 공백 문자열 값(" ")
- CxBlank 상수는 CxBlank Meta Object 속성을 구성하고 이 구성된 값이 해당 Delimited 데이터에 있을 때에만 속성 값으로 지정합니다.

주: 이스케이프 문자열 및 분리문자가 Delimiter Data Handler 하위 Meta Object에서 Escape 및 Delimiter Meta Object 속성에 의해 구성된 대로 서로 다른 값을 갖고 있는지 확인하십시오.

다음 행에서는 Delimited 형식의 문자열 예를 보여줍니다. 구문은 다음과 같습니다.

```
Bus_Obj_Name<delimiter>Verb<delimiter>Attr1<delimiter>Attr2<delimiter>
Number_of_child_object_instances<delimiter>Child_Object_Name<delimiter>
Verb<delimiter>Attr1<delimiter>Attr2<EndBO:Bus_Obj_Name>
```

다음 샘플은 틸데(~) 분리문자를 사용합니다.

```
Customer~Create~p1~p2~p3~1~CustomerAddress~Create~q1~q2~q3~q4~q5~q6~q7~q8~q9~q10~3~
PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7~PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7~
PhoneInfo~Create~r1~r2~r3~r4~r5~r6~r7
```

직렬화된 데이터 처리

Delimited Data Handler는 다음과 같이 구분된 데이터를 Business Object로 처리합니다.

1. Data Handler는 데이터의 첫 번째 토큰에서 Business Object 이름을 확보하고 데이터를 포함할 Business Object를 작성합니다.
2. Data Handler는 Business Object에서 verb를 설정합니다. Data Handler는 최상위 레벨 Business Object의 verb가 구분된 데이터에서 두 번째 토큰에 있는 것으로 간주합니다. 하위 Business Object에는 verb가 설정되지 않을 수도 있습니다.
3. Data Handler는 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 cw_mo_ 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 Business Object의 이러한 속성을 채우기 위해 처리를 수행하지 *않습니다*. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
4. Data Handler는 Meta Object 속성인 OmitObjectEventId를 찾습니다. 이 속성이 true이면, Data Handler는 ObjectEventId 속성을 채우기 위한 처리를 수행하지 않습니다.
5. Data Handler는 데이터의 구문을 분석하고 Business Object에 있는 나머지 단순 속성의 값을 데이터의 토큰 값으로 채웁니다. Data Handler는 다음과 같이 컨테이너 속성을 처리합니다.

- 속성이 단일 카디낼리티일 경우, Data Handler는 문자열에서 순환 방식으로 속성 토큰의 구문을 분석하고 Business Object에서 속성 값을 설정한 후 하위 Business Object 컨테이너를 상위 Business Object에 추가합니다.
- 속성이 다중 카디낼리티일 경우, Data Handler는 각 하위 오브젝트마다 순환 방식으로 속성 토큰의 구문을 분석하고 하위 Business Object에서 속성 값을 설정한 후 하위 Business Object 컨테이너를 상위 Business Object에 추가합니다.

제 8 장 NameValue Data Handler

NameValue Data Handler는 Business Object를 이름-값 쌍으로 형식화된 문자열 또는 스트림으로 변환하고, 이름-값 쌍으로 형식화된 문자열 또는 스트림을 Business Object로 변환합니다. 이 장에서는 NameValue Data Handler가 NameValue 데이터를 처리하는 방법과 NameValue Data Handler에서 처리할 Business Object를 정의하는 방법에 대해 설명합니다. 이 정보를 안내서로 사용하여 NameValue Data Handler 요구사항을 따르는 Business Object를 구현할 수 있습니다. 또한 NameValue Data Handler를 구성하는 방법에 대해서도 설명합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 174 페이지의 『NameValue Data Handler 구성』
- 178 페이지의 『Business Object를 NameValue 데이터로 변환』
- 179 페이지의 『NameValue 데이터를 Business Object로 변환』

주: NameValue Data Handler는 `CwDataHandler.jar` 파일에 들어 있는 기본 Data Handler 중 하나입니다. 이 Data Handler 설치 방법 및 해당 소스 코드의 저장 위치에 대한 정보는 25 페이지의 제 2 장 『Data Handler 설치 및 구성』의 내용을 참조하십시오.

개요

NameValue Data Handler는 Business Object를 이름-값 쌍으로 형식화된 문자열 또는 스트림으로, 그리고 이름-값 쌍으로 형식화된 문자열 또는 스트림을 Business Object로 변환하는 1차 역할인 데이터 변환 모듈입니다. NameValue 형식의 문자열 또는 스트림은 `text/namevalue` MIME 유형으로 직렬화된 데이터입니다. NameValue Data Handler는 테스트 목적의 Business Object 파일을 생성하는 데 사용할 수 있습니다.

Data Handler는 이름 지정된 필드를 기초로 직렬화된 데이터의 구문을 분석합니다. 예를 들어, 이 Data Handler의 텍스트 파일에 Business Object 유형 (BusinessObject=BOname), verb(Verb=verbName), 속성 수 (AttributeCount=numericValue) 및 속성 값(`AttributeName=`*`Value`*)을 식별하는 필드가 있을 경우, Data Handler는 이름-값 정보를 사용하여 데이터의 구문을 분석합니다.

기본 최상위 레벨 커넥터 Meta Object(`MO_DataHandler_Default`)는 `text/namevalue` MIME 유형을 지원합니다. 그러므로 `MO_DataHandler_Default` Meta Object를 사용하도록 구성된 커넥터는 NameValue Data Handler를 호출할 수 있습니다. 액세스 클라이언트가 InterChange Server 통합 브로커를 사용할 때 이 Data Handler를 호출할 수 있도록 하려면, `text/namevalue` MIME 유형을 지원하도록

MO_Server_DataHandler Meta Object를 수정해야 합니다. 자세한 정보는 219 페이지의 『최상위 레벨 Meta Object 수정』을 참조하십시오.

NameValue Data Handler는 표 63에 나열된 조작을 수행합니다.

표 63. NameValue Data Handler의 데이터 조작

Data Handler 조작	**자세한 정보**
호출자로부터 Business Object를 수신하고 Business Object를 NameValue 문자열 또는 스트림으로 변환한 후 직렬화된 데이터를 호출자에게 전달합니다.	178 페이지의 『Business Object를 NameValue 데이터로 변환』
호출자로부터 문자열이나 스트림을 수신하고 Business Object를 빌드한 후 Business Object를 호출자에게 리턴합니다.	179 페이지의 『NameValue 데이터를 Business Object로 변환』

NameValue Data Handler 구성

NameValue Data Handler를 구성하려면 다음 단계를 수행하십시오.

- NameValue 하위 Meta Object 속성에 대한 적절한 값을 입력하십시오.
- Business Object 정의가 Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정하십시오.

이러한 단계는 각각 다음 섹션에 자세히 설명되어 있습니다.

NameValue 하위 Meta Object 구성

NameValue Data Handler를 구성하려면, NameValue 하위 Meta Object에서 구성 정보가 제공되는지 확인해야 합니다. NameValue Data Handler의 경우, IBM은 MO_DataHandler_DefaultNameValueConfig 하위 Meta Object를 제공합니다. 이 Meta Object에 있는 각각의 속성이 NameValue Data Handler의 구성 등록 정보를 정의합니다. 표 64에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 64. NameValue Data Handler의 하위 Meta Object 속성

Meta Object 속성 이름	**설명**	**제공되는 기본값**
ClassName	지정된 MIME 유형으로 사용하기 위해 로드할 Data Handler 클래스의 이름. 최상위 레벨 Data Handler Meta Object에는 이름이 지정된 MIME 유형과 일치하고 유형이 NameValue 하위 Meta Object(표 64 참조)인 속성이 있습니다.	com.crossworlds.DataHandlers.text.namevalue
CxBlank	특수 Business Object 속성 값인 Blank(CxBlank 상수)에 해당하는 값을 NameValue 데이터에 설정하십시오. 자세한 정보는 177 페이지의 『CxBlank』를 참조하십시오.	CxBlank 상수
CxBlankValue	*이 속성은 더 이상 사용하지 않습니다.* Data Handler에 NameValue 데이터가 CxBlank 속성 값을 표시하는 방법을 알리려면 CxBlank Meta Object 속성(위)을 사용하십시오.	공백 공간
CxIgnore	특수 Business Object 속성 값인 Ignore(CxIgnore 상수)에 해당하는 값을 NameValue에 설정하십시오. 자세한 정보는 176 페이지의 『CxIgnore』를 참조하십시오.	CxIgnore 상수
DefaultVerb	Business Object verb	Create
DummyKey	비즈니스 통합 시스템에서 필요로 하는 키 속성	1

표 64. NameValue Data Handler의 하위 Meta Object 속성 (계속)

Meta Object 속성 이름	설명	제공되는 기본값
SkipCxIgnore	요청 처리 시, 특수 속성 값 CxIgnore는 Meta Object 속성 SkipCxIgnore에 따라 달라집니다. 자세한 정보는 176 페이지의 『CxIgnore』를 참조하십시오.	false
ValidateAttrCount	Data Handler가 Business Object 데이터에 있는 속성의 계수를 포함하는 토큰을 찾는지(또는 출력 문자열에 추가하는지) 판별합니다.	true
ObjectEventId	Data Handler에서는 사용되지 않지만 비즈니스 통합 시스템에서 필요로 하는 위치 표시기	없음

표 64에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 해당 속성의 Default Value 등록 정보를 시스템과 이름-값 쌍 형식의 문서에 적절한 값으로 설정해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

Business Object 요구사항

NameValue Data Handler는 처리하는 Business Object에 대한 사항을 가정합니다. 그러므로 NameValue Data Handler를 사용한 변환을 위해 Business Object를 전달할 경우 다음 규칙을 따라야 합니다.

- 모든 Business Object 속성에 Name 등록 정보가 있는지 확인하십시오. 그러면 NameValue Data Handler는 Business Object에서 NameValue 형식으로, 그리고 NameValue 형식에서 Business Object로의 데이터 변환을 적절하게 처리할 수 있습니다.
- ObjectEventId 속성이 Business Object 계층 구조의 모든 레벨에서 모든 Business Object에 포함되어 있는지 확인하십시오. Business Object Designer는 Business Object 정의를 저장할 때 자동으로 이를 수행하지만, 요구사항을 만족하는지 확인해야 합니다.

Business Object 구조

NameValue Data Handler의 Business Object 구조에 대한 요구사항은 없습니다. Data Handler는 모든 Business Object를 처리할 수 있습니다.

Data Handler가 처리하는 Business Object는 비즈니스 통합 시스템에서 허용하는 임의의 이름을 가질 수 있습니다.

Business Object 속성 등록 정보

Business Object 구조에는 속성에 적용하는 다양한 등록 정보가 있습니다. 표 65에서는 여러 등록 정보에 대한 NameValue Data Handler의 분석 방법 및 Business Object 수정 시 해당 등록 정보의 설정 방법에 대해 각각 설명합니다.

표 65. *NameValue Data Handler를 사용하여 변환되는 Business Object의 속성 등록 정보*

등록 정보 이름	설명
Name	각 Business Object 속성에는 고유한 이름이 있어야 합니다.
Type	각 Business Object 속성은 유형(예: 정수, 문자열 또는 포함된 하위 Business Object의 유형)을 가지고 있어야 합니다. 모든 단순 속성의 유형은 String이어야 합니다.
Key	NameValue Data Handler에서는 사용하지 않습니다.
MaxLength	NameValue Data Handler에서는 사용하지 않습니다.
Foreign Key	NameValue Data Handler에서는 사용하지 않습니다.
Required	NameValue Data Handler에서는 사용하지 않습니다.
Default Value	NameValue Data Handler에서는 사용하지 않습니다.
Cardinality	카디널리티 1 및 카디널리티 n 오브젝트를 지원합니다.

Business Object의 속성은 특수 값 `CxIgnore` 또는 `CxBlank`를 가질 수 있습니다. NameValue Data Handler는 속성에 이러한 값이 있을 때, 다음 섹션에 설명된 대로 특수 처리 단계를 수행합니다.

CxIgnore: `CxIgnore` Meta Object 속성은 NameValue 데이터에서 `Ignore` 속성 값(`CxIgnore` 상수)에 해당하는 값을 설정합니다. 기본적으로, `CxIgnore` Meta Object 속성은 `CxIgnore` 상수 값으로 설정됩니다. Data Handler는 다음과 같이 `CxIgnore` Meta Object 속성을 사용합니다.

- Business Object*로부터* 변환할 때, NameValue Data Handler는 속성 값이 `CxIgnore` 상수인 Business Object 속성을 발견할 때마다 `CxIgnore` Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 NameValue 데이터에 기록합니다.
- Business Object로 변환하는 경우, NameValue Data Handler는 NameValue 데이터에서 다음 조건 중 하나라도 발견할 때마다 `CxIgnore` 상수를 Business Object 속성 값에 지정합니다.
  - `CxIgnore` Meta Object 속성(Default Value 등록 정보에 구성된 값)의 값
  - 공백 문자열 값
  - 해당 값이 없음

주: Business Object에는 실행할 때 `CxIgnore`값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.

NameValue Data Handler가 `CxIgnore` 속성 값으로 설정된 속성을 처리하는 방법을 구성할 수 있습니다. 예를 들면, 다음과 같습니다.

- Data Handler가 요청 처리 중 `CxIgnore` 값을 갖는 속성을 처리할 것인지, 아니면 무시할 것인지 구성할 수 있습니다.
- 커넥터가 이벤트 공고 중 Business Object 데이터를 올바르게 처리할 수 있도록 `CxIgnore` 값을 갖는 속성에 대해 항목을 작성하지 않을 것을 결정할 수 있습니다. 이는 오브젝트 유형에 대해 더미 파일을 작성할 때 도움이 됩니다.

요청 처리 중, Data Handler는 Business Object의 직렬화된 버전을 작성합니다. 이 경우, 특수 속성 값 CxIgnore는 다음과 같이 하위 Meta Object 속성 SkipCxIgnore을 기본으로 합니다.

- SkipCxIgnore를 false로 설정할 때, Data Handler는 속성 값이 CxIgnore 상수인 Business Object 속성을 발견할 때마다 CxIgnore Meta Object 속성의 값을 NameValue 데이터에 기록합니다.
- SkipCxIgnore를 true로 설정하면, Data Handler는 값이 CxIgnore인 모든 속성을 무시하고 이 속성에 대한 NameValue 데이터를 생성하지 *않습니다*.

주: SkipCxIgnore가 true로 설정되는 경우, NameValue Data Handler는 양방향이 *아닙니다*. 즉, Business Object에서 문자열로의 변환 중 생성한 문자열에 대해 문자열에서 Business Object 변환을 수행할 수 없습니다.

CxBlank: CxBlank Meta Object 속성은 NameValue 데이터에서 Blank 속성 값(CxBlank 상수)에 해당하는 값을 설정합니다. 기본적으로, CxBlank Meta Object 속성은 CxBlank 상수 값으로 설정됩니다. Data Handler는 다음과 같이 CxBlank Meta Object 속성을 사용합니다.

- Business Object로*부터* 변환할 때, NameValue Data Handler는 속성 값이 CxBlank 상수인 Business Object 속성을 발견할 때마다 CxBlank Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 NameValue 데이터에 기록합니다.
- Business Object로 변환할 때, NameValue Data Handler는 NameValue 데이터에서 이 CxBlank Meta Object 속성 값(Default Value 등록 정보에 구성된 값)을 발견할 때마다 CxBlank 상수를 Business Object 속성 값에 지정합니다.

주: Business Object에는 실행할 때 CxBlank 값을 포함하지 않는 최소한 하나 이상의 1차 키가 있어야 합니다.

Business Object 응용프로그램 특정 정보

NameValue Data Handler는 Business Object 또는 해당 속성에서 응용프로그램 특정 정보를 요구하지 않습니다. 그러나 Data Handler는 cw_mo_ 태그의 존재를 확인합니다. 이 태그는 커넥터가 사용하는 하위 Meta Object를 표시하기 위해 Business Object가 사용할 수 있습니다. Data Handler는 Business Object의 응용프로그램 특정 정보에서 cw_mo_ 태그에 의해 식별되는 속성을 무시합니다. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.

기존 Business Object 정의 사용

NameValue Data Handler는 Business Object가 Data Handler 요구사항을 따르는 양식으로 데이터를 제공할 경우, Business Object를 NameValue 직렬화된 데이터로 변환할 수 있습니다. NameValue Data Handler에서는 각각의 데이터에 식별하는 이름

(예: BusinessObject=Customer, Verb=Create 및 CustomerName=JDoe)이 있어야 합니다. 이처럼 속성은 이름을 가지고 있어야 하므로, 속성을 NameValue Data Handler와 함께 사용할 수 있습니다.

이 요구사항을 만족하는 기존 Business Object를 NameValue Data Handler로 변환할 수는 있지만, 각 유형의 데이터가 처리되도록 사용자의 Business Object를 작성하는 것이 좋습니다. 샘플 Business Object를 사용하거나 다른 구현에서 샘플 응용프로그램을 지원하기 위해 개발된 Business Object를 사용할 경우, 개발 중인 구현에 필요한 속성만 포함하도록 필요에 따라 정의를 수정하도록 하십시오.

그러므로 기존의 Business Object를 거의 데이터에 해당하는 양식으로 변환하려면, 응용프로그램에서 요구하는 데이터와 Data Handler에서 요구하는 정보만 제공하도록 Business Object를 수정하십시오. NameValue Data Handler에 사용하기 위해 기존 Business Object를 조정하려면 다음을 수행하십시오.

1. 대상 응용프로그램의 기능적 분석을 수행하고, 결과를 기존 Business Object와 비교하여 Business Object 정의의 필수 필드를 판별하십시오.
2. Business Object Designer를 사용하여 필요에 따라 Business Object 정의에서 속성을 추가 또는 삭제하십시오.

Business Object를 NameValue 데이터로 변환

Business Object를 문자열 또는 스트림으로 변환하려면, NameValue Data Handler는 순차적으로 Business Object의 속성을 루핑합니다. Business Object 및 해당되는 하위 오브젝트에 속성이 표시되는 순서대로 이름-값 쌍을 순환하여 생성합니다. Business Object의 이름은 인수로 변환 메소드에 전달됩니다.

NameValue Data Handler는 다음과 같이 Business Object를 NameValue 데이터로 처리합니다.

1. Data Handler는 Business Object의 데이터를 포함하는 문자열을 작성합니다.
2. Business Object 이름을 지정하기 위해, Data Handler는 BusinessObject=*Name*을 문자열에 추가합니다.
3. verb를 지정하기 위해, Data Handler는 Verb=*Verb*를 문자열에 추가합니다.
4. Meta Object 속성 ValidateAttrCount가 true로 설정되어 있으면, Data Handler는 문자열에 AttributeCount=*Count*를 추가합니다. 이 이름-값 쌍은 Business Object 데이터에 있는 속성 수를 지정합니다.
5. Data Handler는 Business Object 정의에서 응용프로그램 특정 정보를 조사하여 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 cw_mo_ 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 이러한 속성을 NameValue 데이터에 포함하지 *않습니다*. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.

6. Data Handler는 순서대로 나머지 Business Object 속성을 루핑하여, 각 단순 속성의 이름-값 쌍을 문자열에 추가합니다. 컨테이너 속성의 경우, Data Handler는 다음을 수행합니다.
   - 속성이 카디널리티 1 컨테이너일 경우, Data Handler는 문자열에 속성 이름 및 계수 1을 추가하고, 순환 방식으로 하위 Business Object를 처리하여 각 속성의 이름-값 쌍을 문자열에 추가합니다.
   - 속성이 카디널리티 n 컨테이너일 경우, Data Handler는 속성 이름과 컨테이너에 있는 하위 오브젝트 수를 문자열에 추가하고, 순환 방식으로 각 하위 Business Object를 처리하여 각 속성의 이름-값 쌍을 문자열에 추가합니다.
7. Data Handler가 변환을 완료하면, 직렬화된 데이터를 호출자에게 리턴합니다. Data Handler는 호출자가 요청하는 양식(String 또는 InputStream)으로 데이터를 리턴합니다.

하위 Meta Object 속성 `ValidateAttrCount`가 `true`로 설정되어 있으면, Data Handler는 Business Object에 속성 계수를 포함하는 토큰을 출력 데이터에 추가합니다. Data Handler는 캐리지 리턴을 출력 데이터에 추가합니다. 결과는 180에 있는 그림 36과 유사합니다.

NameValue 데이터를 Business Object로 변환

이 섹션에서는 NameValue Data Handler가 이름-값 쌍 형식의 문자열 또는 스트림을 Business Object로 변환하는 방법에 대한 다음과 같은 정보를 제공합니다.

- 『NameValue 문자열 요구사항』
- 181 페이지의 『직렬화된 데이터 처리』

NameValue 문자열 요구사항

NameValue Data Handler는 직렬화된 데이터에 대해 다음 사항을 가정합니다.

- Business Object 이름은 첫 번째 이름-값 쌍에 표시됩니다.
- Verb는 두 번째 이름-값 쌍에 표시됩니다.
- 데이터에는 Business Object에 포함된 각 하위 오브젝트에 대한 하위 오브젝트 인스턴스 수를 표시하는 토큰이 있습니다.
- `ObjectEventId` 속성은 각 Business Object에 표시됩니다.

속성 계수를 표시하는 토큰은 선택적입니다. 하위 Meta Object 속성 `ValidateAttrCount`가 `true`로 설정되어 있으면, Data Handler는 Business Object에서 속성 계수를 포함하는 토큰을 찾습니다. 속성 계수를 지정할 경우, 속성은 Business Object 정의에 있는 속성 수를 정확하게 반영해야 합니다.

NameValue Data Handler가 이름-값 형식으로 파일을 읽을 때, 다음 특수 처리 단계를 수행하여 Business Object 속성에 CxIgnore 또는 CxBlank 속성 값을 지정하게 됩니다.

- Data Handler는 NameValue 데이터에서 다음 조건 중 무엇이든지 발견할 때마다 CxIgnore 상수((null))를 해당 속성 값으로 지정합니다.
  - CxIgnore Meta Object 속성 값(Default Value 등록 정보에 구성된 값)
  - 공백 문자열 값(" ")
  - Business Object 속성에 대한 NameValue 데이터에 해당 하는 값이 없음
- Data Handler는 CxBlank Meta Object 속성이 구성되어 이 구성된 값이 해당되는 NameValue 데이터에 있을 때에만 CxBlank 상수를 속성 값으로 지정합니다.

그림 36에서는 NameValue 형식의 직렬화된 데이터 예를 보여줍니다.

```
BusinessObject=Customer
              Verb=Update
           AttributeCount=7
           CustomerID=103
           CustomerName=Thai Inc.
           Cust_Phone_Number=CxIgnore
           ProductName=GoodProduct
           Address=2
             BusinessObject=Address
               Verb=Update
               AttributeCount=3
               AddressID=105
               AddressLine=CxIgnore
               ObjectEventID=12345
             BusinessObject=Address
               Verb=Delete
               AttributeCount=3
               AddressID=106
               AddressLine=2758 Forest Avenue
               ObjectEventID=CxIgnore
           Item=1
             BusinessObject=Item
               Verb=Update
               ItemID=107
               ItemName=CxIgnore
               ObjectEventID=Obj_201
           ObjectEventID=SampleConnector_894927711_2
```

그림 36. NameValue 데이터 예

이 예에서, 항목은 다음을 표시합니다.

- BusinessObject는 처리할 상위 또는 하위 Business Object의 이름입니다.
- Verb는 상위 또는 하위 Business Object가 전송되는 요청의 유형(예: Create 또는 Update)입니다.

- AttributeCount는 해당 레벨에서 상위 또는 하위 Business Object의 총 속성 수입니다.
- CustomerID, CustomerName, Cust_Phone_Number 및 ProductName은 상위 Business Object의 속성 이름입니다. 속성 이름 다음에는 각각의 상위 Business Object 값이 옵니다.
- Address = 2는 두 개의 Address 하위 Business Object 인스턴스가 있음을 나타냅니다. Address는 상위 오브젝트에서 Address 하위 Business Object를 참조하는 속성 이름입니다.
- Item = 1은 Item 속성이 Item Business Object의 단일 인스턴스를 포함함을 나타냅니다.
- AddressID 및 AddressLine은 Address 하위 Business Object에 대한 속성 이름입니다. 속성 이름 다음에는 각각의 하위 Business Object 값이 옵니다.
- ObjectEventID=Obj_201은 하위 Business Object인 Item에 대해 시스템에서 생성하는 ID입니다.
- ObjectEventID=SampleConnector_894927711_2는 상위 Business Object Customer에 대해 시스템에서 생성하는 ID입니다.

직렬화된 데이터 처리

NameValue Data Handler는 다음과 같이 이름-값 쌍 형식의 문자열 또는 스트림을 Business Object로 변환합니다.

1. Data Handler는 문자열 또는 스트림 데이터를 포함하는 Business Object를 작성합니다.
2. Data Handler는 Business Object에서 verb를 설정합니다. Data Handler는 최상위 레벨 Business Object의 verb가 데이터에서 두 번째 이름-값 쌍에 있는 것으로 간주합니다. 하위 Business Object에는 verb가 설정되지 않을 수도 있습니다.
3. ValidateAttrCount 하위 Meta Object 속성이 true로 설정되어 있는 경우, Data Handler는 파일의 속성 수가 Business Object의 속성 수와 일치하는지 확인합니다.
4. Data Handler는 직렬화된 데이터의 구문을 분석합니다.
   - Data Handler는 먼저 하위 Meta Object(이름이 Business Object 응용프로그램 특정 정보의 cw_mo_ 태그에 나열된 오브젝트)가 있는지 판별합니다. Data Handler는 Business Object의 이러한 속성을 채우기 위해 처리를 수행하지 *않습니다*. cw_mo_ 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.
   - Data Handler는 Business Object에 있는 나머지의 단순 속성 값을 채웁니다. Data Handler는 다음과 같이 컨테이너 속성을 처리합니다.

속성이 단일 카디널리티일 경우, Data Handler는 순환 방식으로 속성 목록에 있는 속성의 구문을 분석하고 하위 Business Object 컨테이너를 상위 Business Object에 추가합니다.

속성이 다중 카디널리티 배열일 경우, Data Handler는 각 하위 속성 목록에서 순환 방식으로 속성을 구문 분석하고 하위 Business Object 컨테이너를 상위 Business Object에 추가합니다.

Data Handler는 이름 및 값을 연관시키므로, 문자열에서 Business Object로의 변환에 대해 직렬화된 데이터에서 임의의 순서로 속성을 지정할 수 있습니다.

제 9 장 Binary Host Data Handler

Binary Host Data Handler라고 하는 IBM WebSphere Business Integration Data Handler for COBOL 레코드는 비즈니스 오브젝트를 COBOL 레코드로 변환하고 COBOL 레코드를 Business Object로 변환합니다. 이 장에서는 Binary Host Data Handler가 COBOL 레코드를 처리하는 방법과 Data Handler에서 처리할 Business Object를 정의하는 방법을 설명합니다. 이 정보를 안내서로 사용하여 Binary Host Data Handler의 요구사항을 따르는 Data Handler를 구현할 수 있습니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『개요』
- 187 페이지의 『Binary Host Data Handler 구성』
- 189 페이지의 『COBOL 레코드용 Business Object 정의』
- 191 페이지의 『Business Object를 COBOL 레코드로 변환』
- 191 페이지의 『Business Object로 COBOL 레코드 변환』

개요

Binary Host Data Handler는 1차 역할로서 Business Object를 호스트 시스템 COBOL 레코드로/로부터 변환하는 데이터 변환 모듈입니다.

호스트 레코드를 Business Object로 변환할 뿐만 아니라(반대의 변환도 포함), Binary Host Data Handler는 바이트 양식으로 사용 가능한 PD(팩 10진수) 및 DBCS(2바이트 문자 세트) 데이터와 일반 ASCII/텍스트 데이터를 포함하는 COBOL 레코드를 구문 분석합니다.

Binary Host Data Handler는 표 66에 설명된 COBOL 데이터 유형을 처리합니다.

표 66. Binary Host Data Handler에서 지원하는 데이터 유형

데이터 유형	PIC 절 형식
영문자	A(n), AAA... n회
숫자	9(n), 999... n회
영숫자	X(n), XXX... n회
팩 10진수	9(n), COMP-3 절 사용
DBCS	G(n), GGG... n회

이들 데이터 유형에 추가하여, Binary Host Data Handler는 또한 많은 그밖의 PICTURE 절(예를 들어, V, P, S 등)을 지원합니다.

JText Connector 에이전트는 Data Handler에 플러그인하여 이를 사용할 수 있습니다. 다음 섹션은 Binary Host Data Handler를 보다 자세히 설명합니다. JText Connector 에이전트 구성 및 사용에 관한 자세한 정보는 *Adapter for Jtext 사용자 안내서*를 참조하십시오.

COBOL 레코드 및 Business Object 처리

COBOL 레코드는 copybook이라고 하는 템플리트를 사용하여 해당 구조를 정의합니다. 그림 37에서는 COBOL 프로그램의 데이터 부분에 지정되는, COBOL 레코드의 기본 구조를 표시합니다.

```
Data Division.
          File Section.
                FD Customer-File
                Record Contains 50 Characters.
                01 Customer-Record.
                   05 Customer-Name.
          10 Last-Name Pic x(17).
          10 Filler Pic x.
          10 Initials Pic xx.
       05 Part-Order.
          10 Part-Name Pic x(15).
          10 Part-Color Pic x(15).
  Working-Storage Section.
    01 Orig-Customer-Name.
       05 Surname Pic x(17).
       05 Initials Pic x(3).
    01 Inventory-Part-Name Pic x(15).
    01 Inventory-Part-Color Pic x(15).
```

그림 37. COBOL 레코드 구조

데이터 부분의 파일 섹션은 파일에 지정된 대로 레코드의 구조를 명시하는 반면, Working-Storage 섹션은 COBOL 프로그램 내에서 데이터의 핸들링을 명시합니다.

COBOL copybook은 데이터 부분의 파일 섹션에 해당합니다. 위의 구조를 갖는 레코드에 해당하는 copybook이 그림 38에 표시됩니다.

```
01  CUSTOMER-RECORD.
  05  CUSTOMER-NAME
        10 LAST-NAME          PIC X(17)      USAGE DISPLAY-1
        10 FILLER                PIC X
        10 INITIALS              PIC XX
  05  PART-ORDER
        10 PART-NAME          PIC X (15)      USAGE DISPLAY-1
        10 PART-COLOR         PIC X (15)
```

그림 38. 샘플 copybook

copybook이 COBOL 레코드 구조를 설명하듯이, Business Object 정의는 Business Object의 구조를 설명합니다. Binary Host Data Handler는 Business Object 및 COBOL 레코드 간에 변환을 수행할 때 Business Object 정의를 사용합니다. Business Object 정의의 구조와 해당되는 응용프로그램 특정 정보를 사용하여 변환을 수행하는 방법을 결정합니다.

Business Object 정의를 적절하게 구성하면 Data Handler가 Business Object를 COBOL 레코드로, COBOL 레코드를 Business Object로 올바르게 변환할 수 있습니다. 2진 호스트 Data Handler가 COBOL 레코드 및 Business Object 간의 변환을 수행하려면, 연관된 Business Object 정의를 찾을 수 있어야 합니다.

Binary Host Data Handler를 사용하여 COBOL 레코드를 Business Object로 또는 Business Object를 COBOL 레코드로 변환하려면 다음과 같은 단계가 발생해야 합니다.

표 67. Binary Host Data Handler 사용

단계	자세한 정보
1. COBOL 레코드 및 Business Object 구조를 설명하는 Business Object 정의가 존재해야 하고 Binary Host Data Handler 실행 시 사용 가능해야 합니다.	189 페이지의『COBOL 레코드용 Business Object 정의』 190 페이지의『COBOL 레코드에 대한 Business Object 정의 작성』
2. Binary Host Data Handler를 사용자 환경에 맞게 구성해야 합니다.	187 페이지의『Binary Host Data Handler 구성』
3. 다음과 같은 해당 데이터 조작을 수행하려면 Jtext 커넥터에서 Binary Host Data Handler를 호출해야 합니다.	
a) 데이터 조작: 호출자로부터 Business Object를 수신하고, Business Object를 COBOL 레코드로 변환하며, COBOL 레코드를 호출자에게 전달.	191 페이지의『Business Object를 COBOL 레코드로 변환』
b) 데이터 작업: 호출자로부터 COBOL 레코드를 수신하고, 2진 호스트 구성요소를 사용하여 제공된 2진 데이터 유형에 따라 Business Object를 빌드. 그런 다음, Business Object 호출자에게 리턴	191 페이지의『Business Object로 COBOL 레코드 변환』

Binary Host Data Handler 한계

Binary Host Data Handler는 특정 2진 데이터 형식, COBOL 레코드 및 특히 Jtext 커넥터와 함께 사용하도록 특별히 작성되었습니다. 비즈니스 통합 시스템에서 해당 기능의 한계는 다음과 같습니다.

- Data Handler는 한 번에 하나의 레코드 스키마만을 지원하도록 설계되었습니다. 복수의 스키마를 지원하지 않습니다. 그러나 일반적으로 2진 호스트 응용프로그램은 한 번에 단일 스키마를 처리합니다.
- 현재 COBOL copybook을 대응하는 Business Object 오브젝트로 변환하기 위한 ODA는 없습니다. 사용자는 런타임 시 Binary Host Data Handler를 사용하기 전에 COBOL 레코드 자체에 대한 Business Object 정의를 작성해야 합니다.

- Binary Host Data Handler는 2진 데이터 처리에 대해 작동하도록 구성된 어댑터에서만 사용할 수 있습니다.
- Data Handler는 MVS, z/OS, AS/400 및 기타 변환에서만 사용하도록 최적화되었습니다.

2진 호스트 구성요소

내부적으로, Binary Host Data Handler는 서로 다른 구성요소를 사용하여 COBOL 레코드의 다양한 데이터 유형을 처리합니다. 그림 39에서는 이들 구성요소를 표시합니다.

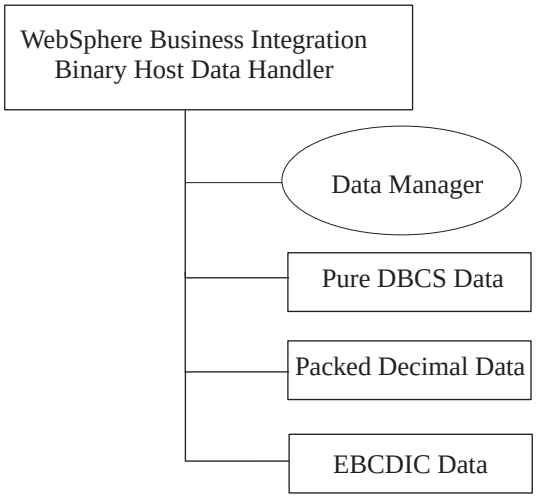

그림 39. Binary Host Data Handler 구성요소

이들 구성요소에 대하여 다음 섹션에서 자세히 설명됩니다.

순수 DBCS 데이터

G(x)의 COBOL PICTURE 절 및 DBCS로 지정된 데이터 유형을 포함하는 ASI를 사용하는 모든 속성은 Binary Host Data Handler에 의해 DBCS 데이터로 취급됩니다. Data Handler의 순수 DBCS 데이터 구성요소는 메인프레임에서 검색한 DBCS 데이터를 다음과 같이 처리합니다.

- shift-in, shift-out 문자를 추가하여 문자열 확보
- 사용자가 지정한 인코딩 형식(기본적으로, Cp930)으로 해당 문자열을 디코딩하여 Unicode Java String 오브젝트 확보

처리할 수 있는 DBCS 데이터의 최소 길이는 1개 문자(즉, 2바이트)입니다. COBOL 컴파일러는 최대 길이를 제한합니다. DBCS 데이터는 항상 오른쪽이 2바이트 공백 문자로 채워집니다.

EBCDIC 데이터

Binary Host Data Handler는 영문자 COBOL PICTURE 절(A 또는 X) 및 EBCDIC으로 지정된 데이터 유형을 포함하는 BO 정의에 있는 응용프로그램 특정 정보를 사용하여 EBCDIC 데이터를 인식합니다. Data Handler의 EBCDIC 구성요소는 MO_DataHandler_DefaultBinaryHostConfig의 'BinaryEncoding' 특성에서 사용자가 지정한 인코딩 형식(기본적으로, Cp930)을 사용하여 EBCDIC 데이터를 디코딩하여 Unicode Java String 오브젝트를 확보합니다.

EBCDIC 데이터의 최소 길이 제한은 1개 문자이며, 1바이트에 해당합니다. COBOL 컴파일러는 최대 길이를 제한합니다. 이 데이터는 항상 오른쪽이 2바이트 공백 문자로 채워집니다.

묶음 10진수 데이터

Binary Host Data Handler는 COMP-3 절을 사용하는, 숫자 PIC 절(9 또는 X) 및 팩 10진수로 지정된 데이터 유형을 포함하는 Business Object 정의에 있는 응용프로그램 특정 정보를 통해 묶음 10진수 데이터를 인식합니다.

묶음 10진수 데이터의 기본 한계는 1개 숫자입니다. 숫자는 부호가 있는 정수여야 하며 부동 소수 값을 표시할 수 없습니다. 묶음 10진수 데이터 구성요소는 COBOL의 팩 10진수 필드가 보유할 수 있는 최대값(S/390 구조에서 15개 숫자로 지정됨)을 초과하지 않습니다. 묶음 10진수 데이터는 항상 왼쪽이 0으로 채워집니다.

주: 예제 입력 999(10진수)는 16진수 값 03D7에 해당합니다. 따라서 팩 10진수로 지정하지 않을 경우, 000003D7로 저장됩니다. 팩 10진수로 지정할 경우, 각 숫자는 니블로 저장되므로 값은 999C(16진수)가 됩니다.

가장 오른쪽에 있는 니블은 팩 10진수의 부호를 나타냅니다. 가장 오른쪽에 있는 A, C, E 또는 F 니블은 양수를, B 또는 D는 음수를 의미합니다.

Binary Host Data Handler 구성

커넥터에서 사용할 Binary Host Data Handler를 구성하려면, 다음 단계를 수행하십시오.

- 커넥터 구성 오브젝트의 속성에 대한 적절한 값을 입력하십시오.
- 2진 호스트 하위 Meta Object 속성에 대한 적절한 값을 입력하십시오.

이러한 단계는 각각 다음 섹션에 자세히 설명되어 있습니다.

주: Binary Host Data Handler를 사용하려면, Data Handler를 지원하도록 Business Object 정의를 작성하거나 수정해야 합니다. 자세한 정보는 111 페이지의 『EDI 문서의 Business Object 정의』를 참조하십시오.

주: Binary Host Data Handler는 2진 처리에 대해 작동하도록 구성된 어댑터에서만 사용할 수 있습니다. JText 어댑터가 해당하는 어댑터입니다.

JText Connector 구성 오브젝트 구성

커넥터 구성 오브젝트인, MO_JTextConnector_Default 파일에서 다음과 같은 속성을 설정해야 합니다.

표 68. MO_JTextConnector_Default의 속성

속성 이름	설명
EventDataHandler	이벤트 처리 시나리오에 대해 DataHandler를 설정합니다. 오브젝트 유형을 포함하는 드롭 상자에서 MO_DataHandler_DefaultBinaryHostConfig를 선택해야 합니다.
OutputDataHandler	서비스 호출 시나리오에 대해 DataHandler를 설정합니다. 오브젝트 유형을 포함하는 드롭 상자에서 MO_DataHandler_DefaultBinaryHostConfig를 선택해야 합니다.
DataProcessingMode	Binary Host Data Handler에 연결하려면 항상 ″binary″여야 합니다. 지정하지 않거나 ‘text’로 지정할 경우 Binary Host Data Handler에서 오류가 발생합니다.
FTPTransferType	항상 ″record″여야 합니다. 지정하지 않거나 ‘file’로 지정할 경우 Binary Host Data Handler에서 오류가 발생합니다.
FTPDataStructure	항상 ″record″여야 합니다. 기본값은 ″file″입니다.

JText Connector의 특성 구성 방법에 관한 자세한 정보는 *Adapter for Jtext 사용자 안내서*를 참조하십시오.

Binary Host Data Handler 하위 Meta Object 구성

MO_DataHandler_DefaultBinaryHostConfig 오브젝트는 2진 데이터 변형을 판별하는 Binary Host Data Handler의 특성을 정의하는 Meta Object입니다.

Binary Host Data Handler를 사용 가능하게 하려면 MO_JTextConnector_Default의 EventDataHandler 및 OutputDataHandler 속성을 MO_DataHandler_DefaultBinaryHostConfig로 설정해야 합니다.

이들 속성을 설정한 후, 이벤트에서 Binary Host Data Handler의 작동을 사용자 정의하거나 시나리오 처리를 요청하려면 MO_DataHandler_DefaultBinaryHostConfig의 하위 속성을 설정해야 합니다. 표 69에서는 이 하위 Meta Object의 속성에 대해 설명합니다.

표 69. Binary Host Data Handler의 하위 Meta Object

속성 이름	설명	제공되는 기본값	필수
ClassName	2진 데이터를 처리하기 위해 로드할 Data Handler 클래스의 이름. 완전한 패키지 구조를 지정해야 합니다.	com.crossworlds.DataHandlers.binary	예
BODefinitionName	COBOL copybook을 지정하기 위해 사용자가 작성한 BO 템플리트의 이름.		예
CxBlank	공백을 처리하기 위해 Data Handler에서 사용하는 필드. 사용자는 여기에 아무 것도 지정할 필요가 없습니다.	CxBlank	아니오

표 69. Binary Host Data Handler의 하위 Meta Object (계속)

속성 이름	설명	제공되는 기본값	필수
CxIgnore	널값을 처리하기 위해 Data Handler가 사용하는 필드. 사용자는 여기에 아무 것도 지정할 필요가 없습니다.	CxIgnore	아니오
DefaultVerb	ASBO(응용프로그램 특정 Business Object)에 관하여 작업하기 위해 Data Handler에서 사용하는 기본 Verb를 지정합니다.	Create	아니오
BinaryEncoding	Binary Host Data Handler가 MVS에서 얻은 데이터를 디코드하거나 MVS로 전송된 데이터를 인코드하는 데 사용하는 인코딩을 지정합니다.		예
DummyKey	키 필드(모든 인스턴스에 대하여 고유한 값이 있음).	1	예

표 32에서 "제공되는 기본값" 열은 제공되는 Business Object에서 해당 속성의 Default Value 등록 정보에 있는 값을 나열합니다. 환경을 조사하고 모든 속성의 Default Value 등록 정보를 적절한 값으로 설정해야 합니다.

주: Business Object 정의를 수정하려면 Business Object Designer를 사용하십시오.

Data Handler 구성 방법에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

COBOL 레코드용 Business Object 정의

2진 Data Handler를 사용하려면, Data Handler에서 필요로 하는 메타 데이터를 포함하고 COBOL 레코드에 있는 해당하는 필드가 포함되도록 Business Object 정의를 작성하거나 수정해야 합니다. 이 섹션에서는 Binary Host Data Handler에 대해 작동하도록 Business Object 정의를 작성하는 데 필요한 정보를 제공합니다. 특히 다음 정보를 제공합니다.

- 『2진 호스트 Business Object 구조 이해』
- 190 페이지의 『COBOL 레코드에 대한 Business Object 정의 작성』

2진 호스트 Business Object 구조 이해

Binary Host Data Handler는 몇 개의 구성요소를 관리하는 관리자로 구성되며, 각각은 특정 데이터 유형을 해석할 수 있습니다. 해당 데이터 유형, 순수 DBCS, 팩 10진수 또는 EBCDIC을 186 페이지의 『2진 호스트 구성요소』에서 설명합니다.

사용자는 이벤트 파일 레코드 구조에 해당하는 copybook에 따라 BO 정의를 개발합니다. COBOL copybook 형식에 관한 자세한 정보는 184 페이지의 『COBOL 레코드 및 Business Object 처리』의 내용을 참조하십시오.

속성의 ASI(응용프로그램 특정 정보)의 파트로, 사용자는 190 페이지의 표 70에서 확보할 수 있는 특정 매개변수, 세부사항 및 적용 가능한 값을 지정해야 합니다.

표 70. COBOL 레코드에 대한 ASBO 정의

속성 이름	설명	올바른 값
데이터 유형	데이터의 데이터 유형 카테고리를 지정합니다. PIC 절만으로는 데이터 유형을 결정할 수 없으므로 포함되었습니다. 이 ASI는 필수이며 모든 속성에 대해 지정해야 합니다.	DBCS ; PackedDecimal ; EBCDIC
PICClause	COBOL copybook에 지정된 대로 정확히 PIC 절과 길이를 지정합니다. 이 ASI는 필수이며 모든 속성에 대해 지정해야 합니다.	X(5); G(8);
COMPClause	존재하는 경우, 필드의 COMP 절을 지정합니다. 팩 10진수 필드에는 일반적으로 연관된 COMP-3이 있습니다. 이 ASI는 선택적이며 비숫자 데이터의 경우 필요하지 않습니다.	COMP-1, COMP-3

Binary Host Data Handler는 응용프로그램 특정 정보에 있는 해당 매개변수 값에 따라 2진 데이터를 해석합니다.

COBOL 레코드에 대한 Business Object 정의 작성

이 섹션에서는 수동으로 Business Object 정의를 작성하여 COBOL 레코드를 표시하는 방법에 대해 설명합니다. Business Object 정의는 COBOL 레코드에 대한 데이터 정의를 제공하는, COBOL copybook을 근거로 합니다. COBOL copybook을 Business Object 정의로 변환하는 특정 ODA(Object Discovery Agent)가 존재하지 않으므로, Business Object Designer를 사용하여 Business Object 정의를 작성해야 합니다. Business Object Designer를 사용하여 Business Object 정의로부터 속성을 추가하거나 삭제하며 필요에 따라 속성 특성을 편집할 수 있습니다.

COBOL copybook을 근거로 Business Object를 정의하려면, 다음을 수행하십시오.

1. Business Object Designer를 열고 '새 Business Object'를 지정하십시오.

   또한 파일 -> 새로 작성... -> Business Object 이름을 선택한 후 ASI를 공백으로 둘 수도 있습니다.
2. 레코드 이름을 Business Object 이름으로 지정하십시오. 예를 들어, CustomerRecord 또는 Customer를 지정하십시오.
3. BO Designer의 일반 탭에서, Delete, Retrieve 및 Update Verb를 선택한 후 Delete 키를 눌러서 삭제하십시오. Create Verb만을 보존하십시오.
4. 속성 탭으로 이동한 후 copybook 필드에 따라, BO 정의에 대한 속성 작성을 시작하십시오. 오브젝트 이벤트 ID 위에 새 속성을 작성하십시오.

Business Object를 COBOL 레코드로 변환

Business Object를 COBOL 레코드로 변환하기 위해, 2진 호스트 Data Handler는 최상위 레벨 Business Object 정의 속성을 루핑합니다. 최상위 레벨 Business Object에 표시되는 순서대로, 순환하여 속성을 처리하고 속성 값을 COBOL 레코드의 요소로 기록합니다.

Binary Host Data Handler는 Business Object를 다음과 같이 COBOL 레코드로 처리합니다.

1. Binary Host Data Handler는 레코드 데이터를 보유할 2진 바이트 배열 인스턴스를 작성합니다.
2. 요청 BO에 해당하는 Business Object 정의를 검색합니다.
3. Business Object 정의에 저장된 메타 데이터(이름, 데이터 유형, 바이트 단위 길이)를 구문 분석합니다.

   데이터 처리에는 다음과 같은 추가 변형이 포함됩니다.

데이터 유형	처리
ASCII/EDBDIC	그대로
숫자(일반)	그대로
팩 10진수	바이트 언팩
DBCS	Cp930을 사용하여 디코드

4. Data Handler는 처리된 바이트로 Business Object 인스턴스를 입력합니다.
5. 마지막으로, Data Handler는 입력된 Business Object를 어댑터로 다시 리턴합니다.

Business Object로 COBOL 레코드 변환

COBOL 레코드를 Business Object로 변환하기 위해, 2진 호스트 Data Handler는 최상위 레벨 Business Object 정의의 속성을 루핑합니다. 작성할 Business Object의 이름을 확보한 후, 최상위 레벨 Business Object 및 하위 오브젝트에 속성이 표시되는 순서대로 순환하여 속성을 처리합니다. 이 때 값을 2진 레코드에서 Business Object에 지정합니다.

2진 데이터를 Business Object로 처리하기 위해 Data Handler에서 취하는 단계는 다음과 같습니다.

1. Data Handler는 메타 데이터 라이브러리에서 Business Object 정의를 선택하여, 지정된 Business Object 인스턴스를 작성합니다.
2. Business Object 정의에 저장된 메타 데이터(이름, 데이터 유형, 바이트 단위 길이)를 구문 분석합니다.
3. Data Handler는 길이 및 데이터 유형 정보에 따라 바이트 스트림을 추출합니다.

데이터 처리에는 다음과 같은 추가 변형이 포함됩니다.

데이터 유형	처리
ASCII/EDBDIC	그대로
숫자(일반)	그대로
팩 10진수	바이트 언팩
DBCS	Cp930을 사용하여 디코드

4. 처리된 바이트로 데이터의 2진 스트림을 생성합니다.
5. 마지막으로, Data Handler는 2진 스트림을 다시 어댑터로 리턴합니다.

제 2 부 Custom Data Handler

제 10 장 Custom Data Handler 작성

이 장에서는 WebSphere Business Integration Adapter와 함께 사용하기 위해 Custom Data Handler를 구현하거나, 통합 브로커가 InterChange Server일 경우, 액세스 클라이언트를 구현하는 방법에 대해 설명합니다. IBM 제공 Data Handler처럼, Custom Data Handler는 Business Object를 특정의 직렬화된 데이터 형식으로 변환하고, 특정 형식의 직렬화된 데이터를 Business Object로 변환합니다. 이 장은 다음 섹션으로 구성되어 있습니다.

- 『Data Handler 개발 프로세스의 개요』
- 198 페이지의 『Data Handler 개발 도구』
- 200 페이지의 『Data Handler 설계』
- 201 페이지의 『Data Handler 기본 클래스 확장』
- 201 페이지의 『메소드 구현』
- 215 페이지의 『사용자 정의 네임 핸들러 빌드』
- 217 페이지의 『jar 파일에 Data Handler 추가』
- 218 페이지의 『Data Handler Meta Objects 작성』
- 221 페이지의 『기타 Business Object 설정』
- 221 페이지의 『커넥터 구성』
- 221 페이지의 『국제화된 Data Handler』

Data Handler 개발 프로세스의 개요

Custom Data Handler를 개발하려면, Data Handler 소스 파일을 코딩하고 다른 타스크(예: Data Handler에 대한 Meta Object 개발)를 완료하면 됩니다. Custom Data Handler를 작성하는 타스크에는 다음과 같은 일반 단계가 포함됩니다.

1. 변환하는 Business Object의 구조와 직렬화된 데이터 형식을 기초로 Data Handler를 설계합니다.
2. 클래스를 확장하는 클래스를 작성합니다.

   `com.crossworlds.DataHandlers.DataHandler`
3. 특정 데이터 형식과 Business Object 사이에 데이터를 변환하는 추상 메소드를 구현합니다.
4. 클래스를 컴파일하여 `CustDataHandler.jar` 파일에 추가합니다.
5. Data Handler Meta Object를 작성합니다.
6. Data Handler의 요구사항과 호출자 요구사항을 따르는 Business Object 정의를 개발합니다.

그림 40은 Data Handler 개발 프로세스의 시각적 개요를 제공하고 특정 주제에 대한 정보를 찾을 수 있는 장에 대한 신속한 참조를 제공합니다. Data Handler 개발을 위해 한 팀을 사용할 수 있으면, Data Handler 개발의 주요 타스크는 개발 팀의 여러 구성원에 의해 병렬로 수행될 수 있습니다.

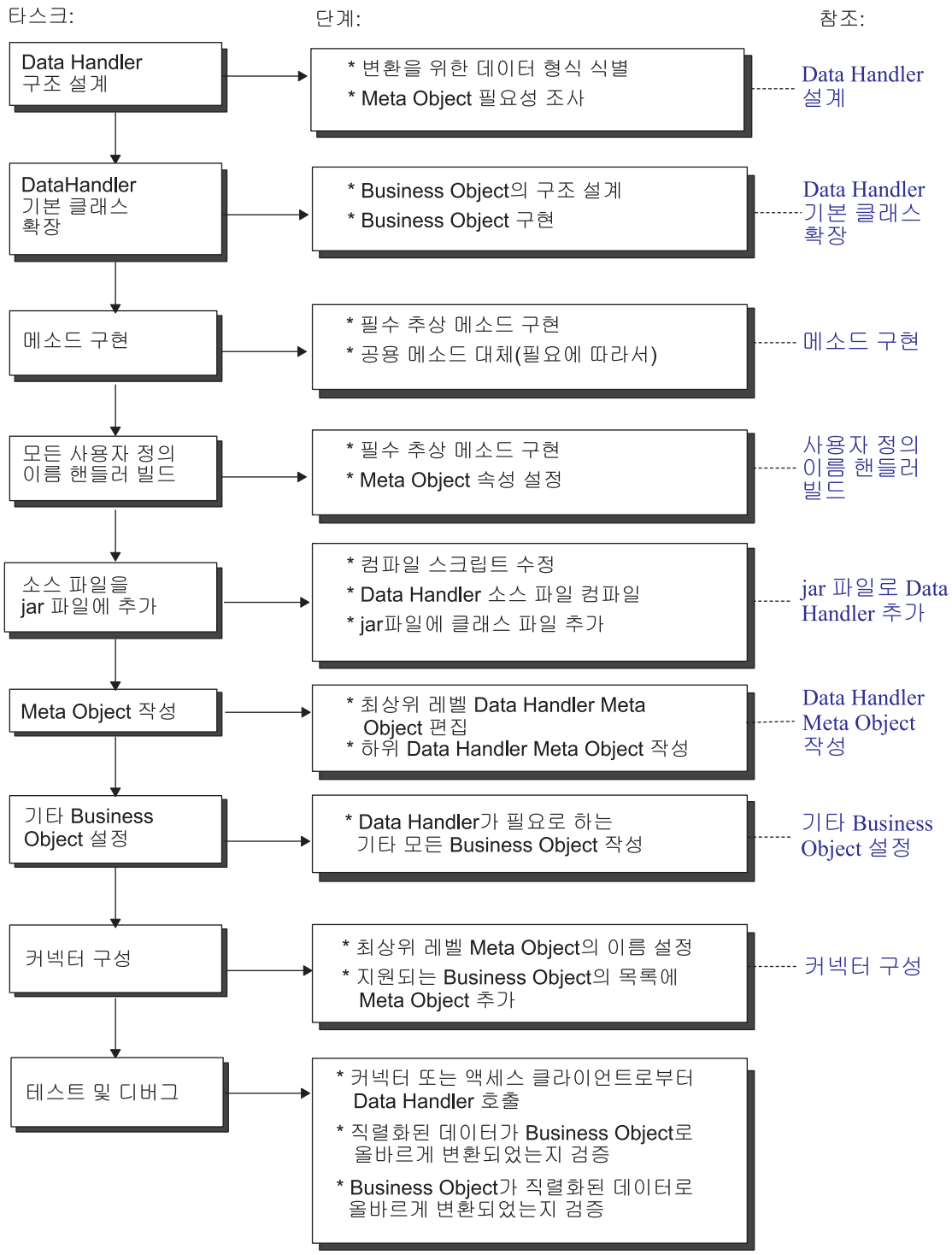

그림 40. Data Handler 개발 프로세스의 개요

Data Handler 개발 도구

Data Handler는 Java로 작성되므로, Data Handler는 Windows 또는 UNIX 시스템에서 개발할 수 있습니다. 표 71에는 IBM에서 Data Handler 개발을 위해 제공하는 도구가 나열되어 있습니다.

표 71. Data Handler 개발 도구

개발 도구	설명
ADK(Adapter Development Kit)	다음을 포함합니다. • 샘플 Data Handler • DataHandler 클래스를 확장하기 위한 스텁 파일
Data Handler API	Custom Data Handler를 작성하기 위해 확장하는 단일 클래스 DataHandler
JCDK(Java Connector Development Kit)	Business Object에 대해 작동할 Java 클래스를 포함합니다.

ADK(Adapter Development Kit)

ADK(Adapter Development Kit)는 어댑터 개발을 도와주는 파일 및 샘플을 제공합니다. ODA(Object Discovery Agent), 커넥터 및 Data Handler를 포함한 많은 어댑터 구성요소에 대한 샘플을 제공합니다. 이들 샘플은 제품 디렉토리의 DevelopmentKits 서브디렉토리에 제공됩니다.

주: ADK는 WebSphere Business Integration Adapters 제품의 파트이며 자체적인 별도의 설치 프로그램이 필요합니다. 따라서 ADK의 개발 샘플을 액세스하려면, WebSphere Business Integration Adapters 제품 및 ADK를 설치해야 합니다. ADK는 Windows 시스템에 대해서만 사용 가능함을 주의하십시오. 표 72에서는 Data Handler의 개발을 위해 ADK에서 제공하는 샘플과, 샘플이 위치하는 DevelopmentKits 디렉토리의 서브디렉토리를 나열합니다.

표 72. Data Handler 개발용 ADK 샘플

Adapter Development Kit 구성요소	**DevelopmentKits** 서브디렉토리
Data Handler 샘플	edk\DataHandler

샘플 Data Handler

Data Handler 개발을 지원하기 위해, ADK는 다음 제품 디렉토리에 몇 가지 샘플 Data Handler의 코드를 포함합니다.

```
DevelopmentKits\edk\DataHandler\Samples
```

표 73에서는 ADK가 제공하는 샘플 Data Handler를 나열합니다.

표 73. EDK와 함께 포함된 샘플 Data Handler

이름	설명
delimited.java	Business Object를 Delimited 문자열로, 그리고 Delimited 문자열을 Business Object로 변환합니다.
fixedwidth.java	Business Object를 FixedWidth 문자열로, 그리고 FixedWidth 문자열을 Business Object로 변환합니다.
namevalue.java	Business Object를 NameValue 문자열로, 그리고 NameValue 문자열을 Business Object로 변환합니다.

주: 이러한 샘플은 연습용으로 유용하지만, DataHandler 클래스에서 지원하는 모든 기능의 예를 제공하지는 *않습니다*.

개발 파일

DevelopmentKits\edk\DataHandler 디렉토리는 표 74에 나열된 것을 포함하여, Custom Data Handler의 개발을 지원하는 여러 개의 파일을 제공합니다.

표 74. Data Handler 개발 파일

Data Handler 개발 파일	자세한 정보
StubDataHandler.java	201 페이지의 『Data Handler 기본 클래스 확장』
makeDataHandler.bat (Windows systems)	217 페이지의 『jar 파일에 Data Handler 추가』

Data Handler API

Data Handler API는 DataHandler라고 하는 단일 Java 클래스를 제공합니다. 추상 DataHandler 기본 클래스는 Custom Data Handler 개발을 쉽게 합니다. 이 클래스에는 Business Object를 입력 데이터에서 추출한 값으로 채우는 메소드 및 Business Object를 문자열 또는 스트림으로 직렬화하는 메소드가 포함됩니다. 또한 Custom Data Handler가 사용할 수 있는 유틸리티 메소드도 포함됩니다. DataHandler 클래스에서 Custom Data Handler를 유도합니다.

DataHandler 클래스의 메소드에 대한 정보는 225 페이지의 제 11 장 『Data Handler 기본 클래스 메소드』를 참조하십시오.

JCDK(Java Connector Development Kit)

Business Object에 대해 작동하려면, Data Handler가 JCDK(Java Connector Development Kit)에 있는 클래스의 메소드를 사용해야 합니다. Data Handler를 개발할 때, 추가 JCDK 클래스(예: CxCommon.CxObjectContainerInterface 또는 CxCommon.CxObjectAttr)를 가져와야 합니다. JCDK 메소드에 대한 참조 정보는 WebSphere Business Integration Adapters 문서 세트에 있는 *Connector Development Guide for Java*를 참조하십시오.

주: JCDK는 하위 레벨 Java 커넥터 라이브러리입니다. 메소드에 대한 문서는 *Connector Development Guide for Java* 중 별첨에 들어 있습니다.

Data Handler 설계

Custom Data Handler 작성을 시작하려면, 다음에 대해 확실히 이해해야 합니다.

- Data Handler가 변환할 파일의 데이터 형식
- Business Object 모델

  특히, 다음을 수행하는 방법에 대해 알고 있어야 합니다.

  – Business Object 인스턴스에서 값을 추출하는 방법

  – Business Object 인스턴스를 빌드하고 이를 파일의 값으로 채우는 방법

메타 데이터 구동 Data Handler 작성

Custom Data Handler가 메타 데이터에 의해 구동되게 하려면, 사용할 Data Handler를 식별하는 정보를 동적으로 지정해야 합니다. 메타 데이터 구동 Data Handler에 관한 자세한 정보는 23 페이지의 『메타 데이터 구동 Data Handler 설계』의 내용을 참조하십시오.

Data Handler Meta Object 사용

결정해야 하는 설계사항 중 하나는 Data Handler가 Meta Object를 사용하여 구성을 초기화하는지 여부입니다.

주: Meta Object에 대한 자세한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

Meta Object 사용 여부를 결정할 때 다음을 고려하십시오.

- Meta Object는 Data Handler가 동적으로 구성되도록 합니다. 이러한 설계 계획은 더 융통성 있는 Data Handler(호출할 때의 문맥을 기초로 구성할 수 있는 Data Handler)를 작성합니다.

  Meta Object를 사용하는 Data Handler를 호출하기 위해, 호출자는 Data Handler의 연관된 MIME 유형을 `createHandler()` 메소드에 전달합니다. MIME 유형으로 호출한 경우, `createHandler()`는 하위 Meta Object에 있는 구성 정보를 사용하여 새로 인스턴스를 생성한 Data Handler를 초기화합니다.

- Meta Object 액세스 및 검색과 연관된 오버헤드가 있습니다. Data Handler는 구성 정보를 변경하지 않거나(하드 코딩될 수 있음) 연관된 MIME 유형을 가지고 있지 않은 데이터를 변환할 경우 Meta Object를 피할 수도 있습니다.

  Meta Object를 사용하지 않는 Data Handler를 호출하기 위해, 호출자는 Data Handler의 클래스 이름만 `createHandler()` 메소드에 전달합니다. 클래스 이름으

로 호출한 경우, createHandler()는 해당 클래스의 Data Handler에 대해 인스턴스를 생성하고, 연관된 Meta Object는 검색하지 않습니다.

Data Handler를 사용하도록 Custom Data Handler를 설계하면, Data Handler 구현의 일부로 이러한 Meta Object를 작성해야 합니다. 자세한 정보는 218 페이지의 『Data Handler Meta Objects 작성』을 참조하십시오.

Data Handler 기본 클래스 확장

Custom Data Handler를 작성하려면, data-handler 기본 클래스(DataHandler)를 확장하여 사용자 고유 *data-handler 클래스*를 작성하십시오. DataHandler 클래스에는 변환(문자열에서 Business Object로 및 Business Object에서 문자열로) 및 개발을 지원하기 위한 유틸리티 메소드가 들어 있습니다. EDK에는 Custom Data Handler에 대한 스텁 코드 및 makefile이 있습니다. 스텁 파일에는 구현해야 하는 모든 메소드가 나열되어 있으며 빈 클래스를 정의하는 Java 코드가 있습니다. Custom Data Handler를 생성하기 위한 템플리트로 스텁 파일을 사용할 수 있습니다.

스텁 파일을 사용하여 Data Handler 소스 파일을 작성하려면 다음을 수행하십시오.

1. StubDataHandler.java 파일을 복사하고 파일이 정의하는 data-handler 클래스 이름과 일치하도록 이름을 바꾸십시오.

   스텁 파일은 제품 디렉토리의 DevelopmentKits\edk\DataHandler 서브디렉토리에 상주합니다. 여기에는 Data Handler 패키지인 com.crossworlds.DataHandlers를 가져오는 import문이 포함됩니다. 이 명령문은 JCDK(Java Connector Development Kit)에서 일부 클래스도 가져옵니다.
2. StubDataHandler 키워드를 Custom Data Handler를 구현하는 클래스의 이름으로 변경하십시오.

   예를 들어, 다음 행은 DataHandler 클래스를 확장하여 HtmlDataHandler라고 하는 Custom Data Handler 클래스를 작성합니다.

   ```
   public class HtmlDataHandler extends DataHandler
   ```

메소드 구현

Data Handler를 개발하기 위해, DataHandler 클래스의 다음 메소드를 구현합니다.

- 추상 DataHandler 메소드(필수)
- 공용 DataHandler 메소드(선택적)

Custom Data Handler의 메소드는 『Data Handler 기본 클래스 확장』에서 작성한 DataHandler 클래스의 Java 소스 파일로 진행합니다.

주: 호출자가 다중 스레드를 거쳐 DataHandler 클래스의 캐시된 인스턴스를 다시 사용할 경우, 클래스를 스레드 안전 상태로 만들어야 합니다. 이를 반드시 수행해야 하는지 여부를 판별하려면, *Connector Development Guide for Java*에서 스레드 모델에 대한 세부사항을 참조하십시오.

추상 메소드 구현

data-handler 기본 클래스인 DataHandler는 Custom Data Handler의 DataHandler 클래스에 *반드시* 구현해야 하는 표 75의 추상 메소드를 제공합니다.

표 75. DataHandler 클래스의 추상 메소드

데이터 변환	직렬화된 데이터의 형식	DataHandler 메소드
문자열에서 Business Object로 변환	Reader 오브젝트를 통해 액세스되는 직렬화된 데이터를 Business Object로 변환합니다.	getBO() - 추상
Business Object에서 문자열로 변환	Business Object를 InputStream 오브젝트로 변환합니다.	getStreamFromBO()
	Business Object를 String 오브젝트로 변환합니다.	getStringFromBO()
	Business Object를 바이트 배열로 변환합니다.	getByteArrayFromBO()

주: DataHandler 클래스를 사용자 Data Handler로 확장하는 StubDataHandler.java 파일의 사본에는 구현해야 하는 추상 메소드에 대한 선언이 포함되어 있습니다.

다음 섹션에서는 추상 DataHandler 메소드 각각에 대한 구현 정보를 제공합니다.

Business Object로 변환 구현

추상 getBO() 메소드는 문자열에서 Business Object로의 변환을 수행합니다. 즉, Business Object를 Java Reader 오브젝트에서 추출한 데이터로 채웁니다. getBO() 메소드 버전은 두 가지입니다.

- getBO(Reader *serializedData*, BusinessObjectInterface *theObj*, Object *config*)

  입력 인수에는 직렬화된 데이터 및 Business Object에 대한 참조가 들어 있는 Reader 오브젝트가 포함됩니다. 메소드는 *theBusObj* Business Object를 *serializedData* 데이터로 채웁니다.

- getBO(Reader *serializedData*, Object *config*)

  입력 인수에는 직렬화된 데이터가 들어 있는 Reader 오브젝트가 포함됩니다. 메소드는 데이터에서 Business Object 유형의 이름(Business Object 정의)을 판별한 후 해당 유형의 Business Object 인스턴스를 작성하여 채웁니다.

주: 커넥터의 문맥에서 호출하는 경우, Data Handler를 지원하려면, data-handler 클래스(DataHandler 확장)가 getBO() 메소드의 두 버전을 구현해야 합니다. 액세스 클라이언트에서 호출하는 *경우에만* Data Handler를 지원하려면(IBM WebSphere

InterChange Server 통합 브로커에만 해당), 메소드의 두 번째 양식만 구현해야 합니다. getBO() 서버 액세스 인터페이스는 이 두 번째 양식의 getBO()만 사용합니다.

getBO() 메소드는 호출자가 구성 정보(*config* 매개변수)를 포함하는 선택적 오브젝트를 전달합니다. 이 정보는 Data Handler Meta Object에서 지정된 정보 외에 추가되는 정보입니다. 예를 들어, 구성 오브젝트는 템플리트 파일이나, Data Handler가 사용하는 URL을 지시할 수 있습니다.

주: Business Object로 변환할 때, getBO()는 상위 Business Object의 응용프로그램 특정 정보에서 cw_mo_*label* 태그로 식별된 속성이 값을 얻지 *못하*도록 해야 합니다. cw_mo_*label* 태그에 대한 자세한 정보는 207 페이지의 『Business Object에서 변환 구현』을 참조하십시오.

추상 getBO() 메소드의 목적은 Reader 오브젝트에 들어 있는 직렬화된 데이터로 Business Object를 채우는 것입니다. 그러면 getBO()의 공용 버전이 지원되는 여러 양식 중 하나로 직렬화된 데이터를 수신하고 데이터를 Reader 오브젝트로 변환한 후, 추상 getBO() 메소드를 호출하여, 실제 문자열에서 Business Object로의 변환을 수행할 수 있습니다. 공용 getBO() 메소드에 대한 자세한 정보는 228 페이지의 『getBO() - 공용』을 참조하십시오.

그림 41에서는 두 번째 양식의 getBO() 메소드에 대한 기본 구현을 보여줍니다. 예에서는 고정 폭 데이터를 포함하는 Reader 오브젝트의 데이터가 Business Object로 변환할 때 사용되는 단계를 보여줍니다.

1. getBO() 메소드는 Reader 오브젝트의 데이터를 String 오브젝트로 변환합니다. 그런 다음, 사용자 정의 getBOFromString() 메소드를 호출하여 Business Object 인스턴스를 작성합니다.
2. getBOFromString() 메소드는 String의 첫 번째 고정 폭 토큰을 기초로 작성할 Business Object 유형을 판별한 후 해당 유형의 Business Object 인스턴스를 작성합니다. 이 메소드는 String의 두 번째 고정 폭 토큰에서 verb를 확보하고 Business Object에서 verb를 설정합니다. 그런 다음, 사용자 정의 parseAttributeList() 메소드를 호출하여 String의 나머지를 구문 분석하고 Business Object를 값으로 채웁니다.
3. parseAttributeList() 메소드는 String을 구문 분석하고 순환 방식으로 Business Object를 채웁니다. 메소드가 오브젝트 유형의 속성을 찾으면, 메소드는 오브젝트의 단일 또는 다중 카디낼리티 여부를 판별합니다. 메소드는 getMultipleCard()를 호출하여 배열에 있는 Business Object를 반복적으로 처리하고 getSingleCard()를 호출하여 단일 카디낼리티 하위 Business Object를 처리합니다.

팁: Data Handler는 Business Object에서 데이터를 추출하고 커넥터와 동일한 방식으로 Business Object를 데이터로 채웁니다. 예를 들어, 다음 코드 샘플에서

getBOFromString() 메소드는 JavaConnectorUtil.createBusinessObject()를 호출하여 Business Object 인스턴스를 작성하고 BusinessObjectInterface.setVerb()를 호출하여 verb를 설정합니다. Business Object 처리 방법에 대한 정보는 *Connector Development Guide for Java*를 참조하십시오.

```
public BusinessObjectInterface getBO(Reader serializedData,
          Object config)
          throws Exception
{
          clear(config);
          BusinessObjectInterface resultObj = null;

          // Create a String object from the Reader, then use the string
          // method
          int conversionCheck;
    char[] temp = new char[2000];
          StringBuffer tempStringBuffer = new StringBuffer(1000);

          while ( (conversionCheck = serializedData.read(temp)) != -1 )
            tempStringBuffer.append(new String (temp, 0, conversionCheck));

          mBOString = new String(tempStringBuffer);
          mBOStringLength = mBOString.length();

          resultObj = getBOFromString(null);
          return resultObj;
}


// Gets Business Object name and verb and creates a bus obj instance
private BusinessObjectInterface getBOFromString(String pvBOType)
          throws Exception
{
          BusinessObjectInterface returnObj = null;
          String lvBOName = null;
          String lvVerb = null;

          lvBOName = this.getNextToken(mBONameSize, true);
          lvVerb = this.getNextToken(mBOVerbSize, true);

          if( (pvBOType != null) && (lvBOName.compareTo(pvBOType) != 0) ) {
            throw new Exception(...);
    }
               else
    {
            returnObj = JavaConnectorUtil.createBusinessObject(lvBOName);
    }

          returnObj.setVerb(lvVerb);

          parseAttributeList(returnObj);

          return returnObj;
}
```

그림 41. getBO() 메소드 예 (1/4)

```
// Parse String to populate the attributes in the Business Object
protected void parseAttributeList(BusinessObjectInterface pvBO)
          throws Exception
{
          if ( mEndOfBOString )
            throw new Exception(...);
          else if( pvBO == null )
            throw new Exception(...);

          int lvAttrNum = pvBO.getAttrCount();

          String lvAttrName = null;
          String lvAttrValue = null;
          int lvAttrMaxLength = 0;

    try {
            for (int lvAttrIndex = 0; lvAttrIndex < lvAttrNum;
                lvAttrIndex++)
        {
              CxObjectAttr lvAttrSpec = pvBO.getAttrDesc(lvAttrIndex);
              lvAttrName = lvAttrSpec.getName();

            // Determine if the attribute is a simple attribute or a
            // Business Object container.
              if (lvAttrSpec.isObjectType())
            {
              // Get the next token based on the BOCountSize
              lvAttrMaxLength = mBOCountSize;
              lvAttrValue =  this.getNextToken(mBOCountSize, true);
              String lvBOType = lvAttrSpec.getTypeName();
              Object lvAttrBOValue = null;

                if (lvAttrSpec.isMultipleCard())
                {
                this.getMultipleCard(pvBO,lvAttrIndex,lvBOType,
                  lvAttrValue);
                }
            else {
                this.getSingleCard(pvBO,lvAttrIndex,lvBOType,
                  lvAttrValue);
                }
            }
                else
            {
              // Get the next token based on the MaxLength of the attribute
                lvAttrMaxLength = lvAttrSpec.getMaxLength();
                if (lvAttrMaxLength > 0)
                lvAttrValue =  this.getNextToken(lvAttrMaxLength, false);
                else
                lvAttrValue = null;
```

그림 41. getBO() 메소드 예 (2/4)

```
            // For simple String attribute, set to null, set to
            // configured CxIgnore or CxBlank values, or set to the
            // attribute value
            if (lvAttrValue == null )
              pvBO.setAttrValue(lvAttrIndex, null);
            else if (lvAttrValue.equals(mCxIgnore)||
                  lvAttrValue.equals(""))
              pvBO.setAttrValue(lvAttrIndex, null);
            else if (lvAttrValue.equals(mCxBlank)||
                  lvAttrValue.equals(" "))
              pvBO.setAttrValue(lvAttrIndex, "");
              else
              pvBO.setAttrValue(lvAttrIndex, lvAttrValue);
          }
        }
    }
}


// Populates a child container with values in the String
protected void getMultipleCard(BusinessObjectInterface pvParentBO,
    int pvParentAttrIndex, String pvBOType, String pvObjectCountString)
          throws CW_BOFormatException, Exception
{
    try {
            if ( pvObjectCountString.equals(mCxIgnore) )
        {
              // trace message
        }
            else {
              int lvObjectCount = Integer.parseInt(pvObjectCountString);
              if ( lvObjectCount == 0)
            {
                // trace message with the number of objects in container
            }
              else if (lvObjectCount > 0)
            {
           // There is at least one instance of the object in the string
              BusinessObjectInterface lvChildBO = null;

                // For each instance of the child object, parse the attribute
                // list, and then add the object container to the parent.
                for (int lvObjectIndex = 0; lvObjectIndex < lvObjectCount;
                      lvObjectIndex++)
                {
              lvChildBO = getBOFromString(pvBOType);
                  pvParentBO.setAttrValue(pvParentAttrIndex,lvChildBO);
                }
            }
        }
    }
}
```

그림 41. getBO() 메소드 예 (3/4)

```
// Populates a single cardinality child with values in the String
protected void  getSingleCard(BusinessObjectInterface pvParentBO,
    int pvParentAttrIndex, String pvBOType, String pvObjectCountString)
          throws CW_BOFormatException, Exception
{
    try {
            BusinessObjectInterface lvChildBO = null;

            // Check the object count token
            // If it does not match "1", assume that the child object should
            //  be null
            if (pvObjectCountString.equals("1"))
        {
              // The string contains a single instance of the child
              lvChildBO = getBOFromString(pvBOType);
              pvParentBO.setAttrValue(pvParentAttrIndex, lvChildBO);
        }
            else if ( pvObjectCountString.equals(mCxIgnore) ||
                pvObjectCountString.equals("0"))
        {
              // Validate that the object count token is valid
        }
                else
              throw new CW_BOFormatException(...);
    }
}
```

그림 41. getBO() 메소드 예 (4/4)

Business Object에서 변환 구현

표 76의 추상 메소드는 Business Object에서 문자열로의 변환을 수행합니다. 즉, 각각 Business Object의 특정 양식으로 직렬화된 데이터를 작성합니다.

표 76. Business Object에서 문자열로의 변환을 구현하는 추상 메소드

추상 메소드	설명
getStringFromBO()	Business Object의 데이터를 String 오브젝트로 변환합니다.
getStreamFromBO()	Business Object의 데이터를 InputStream 오브젝트로 변환합니다.
getByteArrayFromBO()	Business Object의 데이터를 바이트 배열로 변환합니다.

Business Object로부터의 변환 목적은 Business Object에 있는 모든 데이터의 직렬화된 양식을 작성하는 것입니다. 그러나 때때로 일부 Business Object 데이터는 직렬화된 데이터에 포함되지 *말아야* 합니다. 예를 들어, Business Object는 하위 Meta Object를 사용하여 커넥터에 대한 동적 구성 정보를 보유할 수 있습니다.

IBM은 구성 및/또는 동적 메타 데이터를 위해 cw_mo_*label* 접두부로 시작하는 모든 응용프로그램 특정 정보를 예약합니다. Data Handler가 Business Object로부터의 변

환 중에 무시해야 하는 속성을 표시하기 위해, 상위 Business Object에 대한 Business Object 정의는 응용프로그램 특정 정보에 다음 태그를 지정합니다.

```
cw_mo_label=child_meta-object_attribute_name
```

여기서 *label*은 추가로 cw_mo_ 태그의 목적을 더 식별하기 위해 정의하는 문자열이고 *child_meta-object_attribute_name*은 무시할 속성의 이름을 식별합니다. 이 속성에는 보통 하위 Meta Object가 포함됩니다. 여러 cw_mo_*label* 태그는 세미콜론(;)으로 구분됩니다.

Custom Data Handler에 대해 getStringFromBO(), getStreamFromBO() 및 getByteArrayFromBO() 메소드를 구현하는 경우, 이 메소드는 다음과 같이 Data Handler가 커넥터 특정 속성을 건너뛰도록 해야 합니다.

1. 변환되고 있는 Business Object를 위한 Business Object 정의의 응용프로그램 특정 정보에 있는 cw_mo_*label* 태그(여기서 *label*은 무시할 속성을 식별하기 위해 제공하는 문자열)가 존재하는지 확인하십시오.
2. cw_mo_*label* 태그가 있으면, 이 태그가 제공하는 문자열(*child_meta-object_attribute_name*)을 찾으십시오. 등호(=) 앞뒤의 모든 공백은 무시하십시오.
3. Business Object 속성을 통해 루핑하는 동안, 각 속성 이름을 *child_meta-object_attribute_name* 문자열과 비교하십시오. 이 이름의 속성을 건너뛰십시오.

다음 코드 단편에서는 속성을 건너뛰는 방법을 보여줍니다.

```
List configObjList =
     com.crossworlds.DataHandlers.text.namevalue.listMOAttrNames(BusObj);

//this list contains attribute names, which are configuration objects
for ( attributes in BusObj )
{
          String attrName = BusObj.getAttrDisc(i).getName();
          if ( configObjList.contains(attrName) )
          {
            //skip
            continue;
          }
}
```

예를 들어, MyCustomer라고 하는 Business Object가 하위 Meta Object를 사용하여 커넥터 특정 라우팅 정보를 보유한다고 가정합니다. 이 Meta Object를 속성 CustConfig로 표시할 경우, MyCustomer는 응용프로그램 특정 정보에 다음 태그를 가질 수 있습니다.

```
cw_mo_cfg=CustConfig
```

Business Object로부터 변환 중, Custom Data Handler는 MyCustomer와 연관된 Business Object 정의를 위해 응용프로그램 특정 정보를 확인하고 cw_mo_cfg 태그를

찾은 후, CustConfig 속성을 건너뛰어야 하는지를 판별합니다. 이에 따라 Data Handler에서 생성되는 직렬화된 데이터에는 CustConfig 하위 Meta Object가 포함되지 *않습니다*.

주: Business Object로부터 변환할 때, IBM 제공 Data Handler는 cw_mo_*label* 태그가 식별하는 모든 속성을 건너뜁니다.

Data Handler가 하위 Meta Object나 다른 동적 오브젝트를 사용하는 커넥터와 함께 작동할 *경우에만* cw_mo_*label* 태그를 처리하기 위한 Custom Data Handler를 개발해야 합니다.

getStringFromBO() 메소드 구현: getStringFromBO() 메소드는 Business Object에서 문자열로의 변환을 수행합니다. 즉, Business Object의 데이터를 String 오브젝트로 변환합니다. 이 메소드의 경우, 호출자는 변환할 Business Object를 전달합니다. 그림 42에서는 FixedWidth Data Handler에 의해 구현되는 getStringFromBO() 메소드를 보여줍니다. 메소드는 고정 폭 필드의 String을 작성합니다.

다음 예에서는 Business Object에서 Reader 오브젝트로 데이터를 변환할 때 사용되는 단계를 보여줍니다.

1. getStringFromBO() 메소드는 setAttrList()를 호출하여 Business Object의 속성을 통해 반복적으로 루핑합니다. setAttrList() 메소드가 단순 속성을 찾으면, setSimpleToken() 메소드를 호출하여 값을 설정합니다.
2. setSimpleToken() 메소드는 속성 값을 StringBuffer 오브젝트에 추가하고 StringBuffer를 String 오브젝트로 변환시킨 후 전체 Business Object를 표시하는 StringBuffer에 문자열을 추가합니다. setAttrList()가 Business Object를 처리하면, getStringFromBO() 메소드는 StringBuffer를 String 오브젝트로 변환시켜 String을 호출자에게 리턴합니다.

```
public String getStringFromBO(BusinessObjectInterface theObj,
          Object config)
          throws Exception
{
                    traceWrite(
            "Entering getStringFromBO(BusinessObjectInterface, Object) "
            + " for object type " + theObj.getName(),
            JavaConnectorUtil.LEVEL4);

          clear(config);
          String lvBOString = null;
          setAttrList(theObj);
          lvBOString = mBOStringBuffer.toString();

                    traceWrite(
            "Exiting getStringFromBO(BusinessObjectInterface, Object) "
            + " for object type " + theObj.getName(),
            JavaConnectorUtil.LEVEL4);

          return lvBOString;
}


protected void setAttrList(BusinessObjectInterface pvBO) throws Exception
{
                    traceWrite(
            "Entering setAttrList(BusinessObjectInterface) for object "
             + pvBO.getName(), JavaConnectorUtil.LEVEL4);

          int lvAttrNum = pvBO.getAttrCount();

          String lvAttrName = null;
          String lvAttrValue = null;
          int lvAttrMaxLength = 0;

          // Add the Business Object name and verb to the fixed width format
          //  String
          this.setSimpleToken( mBONameSize, pvBO.getName());
          this.setSimpleToken( mBOVerbSize, pvBO.getVerb());

    try {
            List moAttrNames = listMOAttrNames( pvBO );
            int lvAttrCount = pvBO.getAttrCount();

            ATTRIBUTE_WALK: for (int lvAttrIndex = 0;
              lvAttrIndex < lvAttrCount; ++lvAttrIndex)
        {
              CxObjectAttr lvAttrSpec = pvBO.getAttrDesc(lvAttrIndex);
              lvAttrName = lvAttrSpec.getName();

              // Check if the current attribute is a simple (String) type
              //  or a contained object.
              if (lvAttrSpec.isObjectType())
            {
                //skip child objects designated as meta objects
                if( moAttrNames.contains( lvAttrName ) )
                {
                   continue ATTRIBUTE_WALK;
                }
```

그림 42. getStringFromBO() 메소드 예 (1/5)

```
    if (lvAttrSpec.isMultipleCard())
{
      CxObjectContainerInterface lvAttrMultiCardBOValue =
  (CxObjectContainerInterface) pvBO.getAttrValue(lvAttrIndex);

      if (lvAttrMultiCardBOValue == null)
    {
        traceWrite(
    "setAttrList found empty multiple cardinality container "
    + lvAttrSpec.getTypeName(), JavaConnectorUtil.LEVEL5);

        // Add the count to the fixed width String
        this.setSimpleToken( mBOCountSize, "0");
    }
    else
    {
        int lvObjectCount =
          lvAttrMultiCardBOValue.getObjectCount();
        traceWrite(
      "setAttrList found multiple cardinality container "
      + lvAttrSpec.getTypeName() + " with "
      + lvObjectCount + " instances",
      JavaConnectorUtil.LEVEL5);

        // Add the count to the fixed width String
        this.setSimpleToken( mBOCountSize,
          Integer.toString(lvObjectCount));

        // Add each object in the container to the fixed
        //  width String.
        for (int lvContObjectIndex = 0;
            lvContObjectIndex < lvObjectCount;
            ++lvContObjectIndex)
          setAttrList(
lvAttrMultiCardBOValue.getBusinessObject(lvContObjectIndex));
    }
}
    else
{
      BusinessObjectInterface lvAttrSingleCardBOValue =
  (BusinessObjectInterface) pvBO.getAttrValue(lvAttrIndex);

      if (lvAttrSingleCardBOValue == null)
    {
        traceWrite(
    "setAttrList found empty single cardinality container "
    + lvAttrSpec.getTypeName(), JavaConnectorUtil.LEVEL5);

        // Add the count to the fixed width String
        this.setSimpleToken( mBOCountSize, "0");
    }
```

그림 42. getStringFromBO() 메소드 예 (2/5)

```
                else
                {
                    traceWrite(
                  "setAttrList found single cardinality container "
                  + lvAttrSpec.getTypeName(),
                  JavaConnectorUtil.LEVEL5);

                    // Add the count to the fixed width String
                    this.setSimpleToken( mBOCountSize, "1");
                    setAttrList(lvAttrSingleCardBOValue);
                }
            }
        }
                else
        {
                lvAttrValue = (String) pvBO.getAttrValue(lvAttrIndex);
                lvAttrMaxLength = lvAttrSpec.getMaxLength();

                if (lvAttrMaxLength > 0)
                  this.setSimpleToken(lvAttrMaxLength, lvAttrValue);
           }
        }
    }
          catch (CxObjectNoSuchAttributeException e)
    {
            throw new Exception(e.getMessage());
    }

                    traceWrite(
"Exiting setAttrList(BusinessObjectInterface) for object "
             + pvBO.getName(), JavaConnectorUtil.LEVEL4);
}


protected void setSimpleToken( int pvCellSize, String pvTokenValue)
          throws Exception
{
          traceWrite( "Entering setSimpleToken(int, String)",
            JavaConnectorUtil.LEVEL4);

          StringBuffer lvNewBuffer = new StringBuffer(pvCellSize);
          int lvTokenLength = 0;
          int lvCxIgnoreLength = mCxIgnore.length();
          int lvCxBlankLength = mCxBlank.length();

          int lvPadNumber = 0;


          // Check the token value to see if it is null
          if (pvTokenValue == null)
    {
            // For this case, we add the configured CxIgnore value to the
            //  fixed width String if it fits in the cell.
            if (!mTruncation && lvCxIgnoreLength > pvCellSize)
              throw new Exception(
                " Null attribute value encountered where cell size is "
                + pvCellSize + ", size of CxIgnore value is "
                + lvCxIgnoreLength + "and trucation is not allowed. "
            + "Please check your MO format configuration.");
```

그림 42. getStringFromBO() 메소드 예 (3/5)

```
            else
        {
              lvPadNumber = pvCellSize - lvCxIgnoreLength;
              lvNewBuffer.append(mCxIgnore);
        }

    }
          else if (pvTokenValue.equals(""))
    {
            // For this case, the configured CxBlank value is added to the
            //  fixed width String if it fits in the cell.
            if (! mTruncation && lvCxBlankLength > pvCellSize)
              throw new Exception(
            " Blank attribute value encountered where cell size is "
            + pvCellSize + ", size of CxBlank value is "
            + lvCxBlankLength + "and trucation is not allowed. "
            + "Please check your MO format configuration.");
                else
        {
              lvPadNumber = pvCellSize - lvCxBlankLength;
               lvNewBuffer.append(mCxBlank);
        }
    }
                else
    {
            // For this case, actually add the token value to the fixed
            // width String, unless the data is too long for the cell.
            lvTokenLength = pvTokenValue.length();
            if (!mTruncation && lvTokenLength > pvCellSize )
              throw new Exception(
                " Attribute value encountered where cell size is "
                + pvCellSize + ", size of token value is "
                + lvTokenLength + "and trucation is not allowed. "
            + "Please check your MO format configuration.");
                else
        {
              lvNewBuffer.append(pvTokenValue);
              lvPadNumber = pvCellSize - lvTokenLength;
        }
    }

          if (lvPadNumber <= 0 && mTruncation)
            // Token is longer than the cell and truncation is allowed,
            //  so the characters up to pvCellSize are added
            lvNewBuffer.setLength(pvCellSize);
          else if (lvPadNumber > 0)
    {
            // Pad the cell based on the configuration option chosen
            if ( mAlignment.equals(fixedwidth.AlignmentLeft) ||
                mAlignment.equals(fixedwidth.AlignmentBoth))
              this.padRight(lvNewBuffer, lvPadNumber);
            else if (mAlignment.equals(fixedwidth.AlignmentRight))
              this.padLeft(lvNewBuffer, lvPadNumber);
    }
```

그림 42. getStringFromBO() 메소드 예 (4/5)

```
            String lvNewBuffString =  lvNewBuffer.toString();

            // Note that this may cause a performance issue when the tracing
            //  level is low, but in most cases it should not as any one token
            //  is *usually* not very long.
                      traceWrite(
              "Adding the following token to the fixed width String: "
              + lvNewBuffString, JavaConnectorUtil.LEVEL5);

            // After the cell has been fully formatted, append to fixed width
            //  String being built
            mBOStringBuffer.append(lvNewBuffString);

            traceWrite( "Exiting setSimpleToken(int, String)",
              JavaConnectorUtil.LEVEL4);
}
```

그림 42. getStringFromBO() 메소드 예 (5/5)

getStreamFromBO() 메소드의 구현: getStreamFromBO() 메소드는 Business Object의 데이터를 InputStream 오브젝트로 변환시킵니다. 그림 43에서는 getStreamFromBO() 메소드를 구현한 예를 보여줍니다. 이 구현에서, getStreamFromBO()는 getStringFromBO()를 호출하여 Business Object 데이터를 포함하는 String 오브젝트를 빌드한 후 String을 InputStream으로 변환합니다. 메소드는 Business Object의 데이터를 표시하는 InputStream 오브젝트를 리턴합니다.

```
public InputStream getStreamFromBO(BusinessObjectInterface theObj,
          Object config)
          throws Exception
{
          clear(config);

String BOstring;
BOstring = getStringFromBO(theObj, config);

return new ByteArrayInputStream( BOstring.getBytes() );
}
```

그림 43. getStreamFromBO() 메소드 예

공용 메소드 대체

추상 DataHandler 메소드(*구현해야 하는* 메소드) 외에도, Custom Data Handler에 대해 최적으로 작동하도록 DataHandler 클래스의 공용 메소드를 대체해야 하는 경우도 있습니다(표 77 참조).

표 77. DataHandler 클래스의 공용 메소드

공용 DataHandler 메소드	설명
getBO() - 공용	직렬화된 데이터(몇 가지 형식 중 하나)를 Business Object로 변환합니다.
getBOName()	직렬화된 데이터에서 Business Object의 이름을 확보합니다.
getBooleanOption()	Data Handler에서 부울 구성 옵션의 값을 확보합니다.
getOption()	Data Handler에서 구성 옵션의 값을 확보합니다.
setOption()	Data Handler에서 구성 옵션을 설정합니다.
traceWrite()	Data Handler(커넥터 또는 통합 브로커가 InterChange Server인 경우 서버 액세스 인터페이스)의 해당 문맥에 대해 trace-write 메소드를 호출합니다.

사용자 정의 네임 핸들러 빌드

Data Handler는 *네임 핸들러*를 호출하여 직렬화된 데이터에서 Business Object 정의의 이름을 추출할 수 있습니다. 이 타스크는 Data Handler 호출자가 직렬화된 데이터로 채워질 Business Object를 전달하지 않는 경우, 문자열에서 Business Object로 변환하는 과정에서 필요합니다. 이러한 경우, Data Handler는 Business Object를 채우기 전에 먼저 Business Object를 작성해야 합니다. Business Object를 작성하려면, Data Handler가 연관된 Business Object 정의의 이름을 알아야 합니다. 이는 Business Object 이름을 확보하는 네임 핸들러입니다.

주: 현재, XML 및 EDI Data Handler는 네임 핸들러를 사용하여 작성할 Business Object의 이름을 확보합니다.

사용자 정의 네임 핸들러를 작성 및 구현하는 타스크에는 다음과 같은 일반 단계가 포함됩니다.

1. `NameHandler` 클래스를 확장하는 클래스를 작성합니다.
2. 직렬화된 데이터를 읽고 Business Object 이름을 리턴하는 추상 `getBOName()` 메소드를 구현합니다.
3. 클래스를 컴파일하여 `DataHandlers\CustDataHandler.jar` 파일에 추가합니다. 자세한 정보는 217 페이지의 『jar 파일에 Data Handler 추가』를 참조하십시오.
4. Data Handler의 Meta Object에서 `NameHandlerClass` 속성의 기본값을 설정합니다.

NameHandler 클래스 확장

사용자 정의 네임 핸들러를 작성하려면, name-handler 기본 클래스(`NameHandler`)를 확장하여 사용자 고유 *name-handler 클래스*를 작성하십시오. `NameHandler` 클래스에는 직렬화된 데이터에서 Business Object의 이름을 추출하기 위한 메소드가 들어 있습니다. 이 name-handler 기본 클래스의 패키지는 `com.crossworlds.DataHandlers.NameHandler`입니다.

name-handler 클래스를 가져오려면 다음 단계를 수행하십시오.

1. NameHandler 클래스를 확장하는 name-handler 클래스를 작성하십시오.
2. 다음과 같이 name-handler 클래스 파일이 NameHandler 패키지의 클래스를 가져오는지 확인하십시오.

   ```
   import
   ```

3. NameHandler 클래스의 추상 메소드인 getBOName() 메소드를 구현하십시오. 자세한 정보는 『getBOName() 메소드 구현』을 참조하십시오.

NameHandler 클래스의 정의는 다음과 같습니다.

```
// Imports
import java.lang.String;
import java.io.Reader;
import com.crossworlds.DataHandlers.Exceptions.MalformedDataException;

public abstract class NameHandler {

// Constructors
public NameHandler() { }

// Methods
public abstract String getBOName(Reader serializedData,
String boPrefix)
throws MalformedDataException;
}

/* This method was introduced so that the NameHandler would have
 * a reference to the DataHandler. The DataHandler base calls this
 * method after it instantiated the NameHandler:
 * eg. nh = (NameHandler)Class.forName(className).newInstance();
 *     nh.setDataHandler(this);
*/
public final void setDataHandler( DataHandler dataHandler )
{
dh = dataHandler;
}
```

사용자의 네임 핸들러를 작성하려면, NameHandler 추상 기본 클래스를 확장하십시오.

getBOName() 메소드 구현

NameHandler 클래스를 확장하려면 직렬화된 데이터를 읽어 해당 데이터와 연관된 Business Object의 이름을 리턴하는 getBOName() 메소드를 구현해야 합니다. 이 메소드의 구문은 다음과 같습니다.

```
public abstract String getBOName(Reader serializedData, String BOPrefix)
throws MalformedDataException
```

설명:

serializedData

메시지를 포함하는 오브젝트에 대한 참조.

BOPrefix Business Object 정의의 이름에 대한 Business Object 접두부가 들어 있는 String 값. 이 인수는 Meta Object(존재하는 경우)의 BOPrefix 속성으로 설정할 수 있습니다.

Meta Object 속성 설정

Data Handler가 사용자 정의 네임 핸들러를 사용하도록 하려면, Meta Object 속성의 Default Value 등록 정보를 완전한 클래스 이름으로 설정해야 합니다. 그러면 Data Handler는 런타임 시 구성 옵션 중 하나에서 클래스 이름으로 확보할 수 있습니다. 기본적으로, 이 Meta Object 속성을 NameHandlerClass라고 합니다. XML 및 EDI Data Handler 모두와 연관된 하위 Meta Object에는 이 속성이 포함됩니다. 이 속성에 대해 IBM 제공 기본값은 기본 네임 핸들러 클래스의 이름을 지정합니다. Data Handler가 사용자 정의 네임 핸들러를 사용하도록 하려면, 확장하는 Data Handler와 연관된 하위 Meta Object에서 NameHandlerClass 속성의 Default Value 등록 정보를 갱신하도록 하십시오.

jar 파일에 Data Handler 추가

새 Data Handler에 대한 코딩을 완료하면, 클래스를 컴파일하고 이를 Java 아카이브(jar) 파일에 추가해야 합니다. Custom Data Handler를 포함하도록 파일 CustDataHandler.jar이 제공됩니다. 이 jar 파일은 제품 디렉토리의 DataHandlers 서브디렉토리에 있습니다. data-handler 클래스를 찾기 위해 createHandler() 메소드는 전달된 Data Handler가 들어 있는 CwDataHandler.jar 파일을 검색한 후 이 jar 파일을 검색합니다.

주: Java 코드를 컴파일하려면 시스템에 JDK(Java Development Kit)가 설치되어 있어야 합니다. 필수 JDK 버전 및 설치 방법은 제품 설치 정보를 참조하십시오.

Custom Data Handler를 CustDataHandler.jar에 추가하려면 다음을 수행하십시오.

1. data-handler 컴파일 스크립트를 편집하여 Java 소스 파일의 이름을 추가하십시오.

   이 data-handler 컴파일 스크립트는 제품 디렉토리의 다음 서브디렉토리에 있습니다.

   ```
   DevelopmentKits\edk\DataHandler
   ```

   Windows

   Windows 시스템의 경우, data-handler 컴파일 스크립트는 make_datahandler.bat입니다. Java 소스 파일의 이름을 다음 행에 추가하십시오.

   ```
   set SOURCE_FILES_DH=
   ```

UNIX

UNIX 시스템의 경우, data-handler 컴파일 스크립트는 `make_datahandler` 입니다. Java 소스 파일의 이름을 다음 행에 추가하십시오.

```
SOURCE_FILES_DH=
```

2. Data Handler 컴파일 스크립트를 실행하여 Java 소스 파일을 `.class` 파일로 컴파일하십시오.
3. 다음 명령을 사용하여 새 클래스를 `CustDataHandler.jar` 파일에 추가하십시오.

   ```
   jar -vf CustDataHandler.jar input_files
   ```

   여기서 *input_files*는 추가할 클래스 파일 목록입니다.

Data Handler Meta Objects 작성

Data Handler Meta Object를 사용하는 Custom Data Handler를 작성할 경우, 다음을 수행해야 합니다.

- Custom Data Handler의 구성 정보에 대한 속성을 포함하는 하위 Data Handler Meta Object를 작성하십시오.
- Data Handler의 인스턴스가 작성될 때 Data Handler가 구성될 수 있도록 최상위 레벨 Data Handler Meta Object를 수정하여 지원되는 MIME 유형을 포함하게 하십시오.
- 비즈니스 통합 시스템에서 Meta Object를 설정하십시오.

주: Data Handler 설계에서 Meta Object의 사용 여부를 판별하는 방법에 대한 정보는 200 페이지의 『Data Handler Meta Object 사용』을 참조하십시오.

하위 Meta Object 작성

하위 Meta Object에는 Data Handler에 대한 구성 정보가 있습니다. `createHandler()` 메소드는 새로 인스턴스화되는 Data Handler를 초기화하기 위해 이 정보를 사용합니다. 이 프로세스에 대한 자세한 정보는 16 페이지의 『MIME 유형 사용』을 참조하십시오

Custom Data Handler에 대한 하위 Meta Object를 작성하려면 다음을 수행하십시오.

1. Data Handler의 인스턴스를 표시할 하위 Meta Object를 작성하십시오.

이 하위 Meta Object를 작성하는 데 Business Object Designer 도구를 사용할 수 있습니다. 하위 Meta Object에는 Data Handler가 요구하는 구성 정보를 정의하는 속성이 있어야 합니다. 최소한, 하위 Meta Object는 ClassName 속성을 *반드시* 가지고 있어야 합니다.

2. ClassName 속성에서 Data Handler 클래스의 이름을 지정하기 위해 필요한지 판별하십시오.
   - 기본 형식으로 된 Data Handler의 경우, ClassName 값을 지정하지 않아도 됩니다. 기본 형식은 다음과 같습니다.

     com.crossworlds.DataHandlers.*MimeTypeString*

     그러나 ClassName 속성의 기본값으로 Data Handler의 클래스 이름을 지정할 수 있습니다.
   - 기본 형식이 아닌 Data Handler 클래스의 경우, Data Handler 인스턴스에 대해 완전한 클래스 이름을 지정해야 합니다. 그렇지 않으면, createHandler() 메소드는 Data Handler 인스턴스를 생성하려고 할 때 Data Handler를 찾을 수 없습니다.
3. 하위 Meta Object의 해당 속성의 기본값을 설정하여 Data Handler 인스턴스가 데이터를 처리하는 방법을 구성하십시오.

최상위 레벨 Meta Object 수정

호출자가 MIME 유형을 createHandler() 메소드에 제공할 때, createHandler()는 다음 단계를 사용하여 인스턴스를 생성할 Data Handler를 판별합니다.

1. Data Handler와 연관된 최상위 레벨 Meta Object의 이름을 찾습니다.
2. 변환할 데이터와 일치하는 MIME 유형을 찾기 위해 이 최상위 레벨 Meta Object를 살펴봅니다.
3. 이 MIME 유형이 존재할 경우, 구성 정보를 포함하는 연관된 하위 Meta Object 이름을 찾습니다.

이 프로세스의 자세한 설명은 16 페이지의 『MIME 유형 사용』을 참조하십시오. 이 프로세스가 성공하려면, createHandler()가 데이터와 연관된 MIME 유형을 찾을 수 있어야 합니다. 그러므로 Data Handler가 변환할 데이터의 MIME 유형에 대한 속성을 포함하도록 최상위 레벨 Data Handler Meta Object를 편집해야 합니다. 이 속성은 다음 사항을 포함해야 합니다.

- Data Handler의 연관된 MIME 유형에 대해 MIME 유형 문자열과 일치하는 이름

  MIME 유형의 이름은 영숫자 및 밑줄(_)만 포함할 수 있습니다. 다른 특수 문자는 유효하지 않습니다. MIME 유형이 마침표(.)를 가지면, 밑줄로 바꾸십시오.
- 단일 카디널리티 Business Object
- Data Handler를 표시하는 하위 Meta Object의 이름인 유형

예를 들어, 그림 44에서는 사용자 정의 HTML Data Handler에 대해 구성 설정하는 최상위 레벨 커넥터 Meta Object를 보여줍니다.

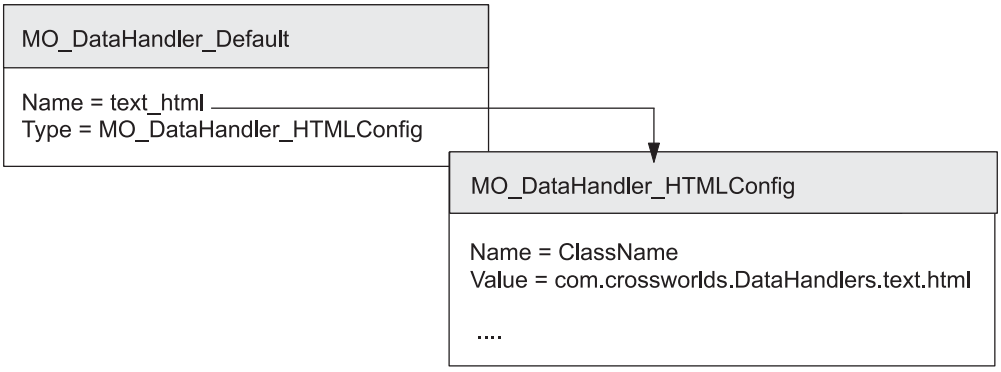

그림 44. Custom Data Handler의 최상위 레벨 커넥터 Meta Object 예

그림 44에서, 커넥터에 대한 기본 최상위 레벨 Meta Object(MO_DataHandler_Default)는 새 MIME 유형인 HTML을 지원하도록 수정했습니다. 이 MIME 유형의 지원에서, MO_DataHandler_Default Meta Object에는 다음과 같은 속성 등록 정보가 있습니다.

```
Attribute Name = text_html
Attribute Type = MO_DataHandler_HTMLConfig
```

중요: MIME 유형의 이름은 영숫자 및 밑줄(_)에만 제한됩니다. MIME 유형에 대해 다른 특수 문자는 사용될 수 없습니다.

비즈니스 통합 시스템에 Meta Object 설정

Data Handler Meta Object를 작성한 후, 다음과 같이 WebSphere Business Integration System의 Meta Object를 설정해야 합니다.

1. 새 Meta Object를 저장소로 로드하십시오.
2. Data Handler를 호출할 문맥에 따라 적절한 Meta Object를 수정하십시오.
   - Data Handler가 커넥터 문맥으로 실행할 경우, Data Handler Meta Object를 하위 오브젝트로 최상위 레벨 Data Handler Meta Object에 추가하십시오. 그런 다음, 최상위 레벨 Data Handler Meta Object에 대한 지원을 커넥터 정의에 추가하십시오.
   - 통합 브로커가 InterChange Server이고 Data Handler가 서버 액세스 인터페이스의 문맥으로 실행할 경우, Data Handler Meta Object를 하위 오브젝트로 최상위 레벨 서버 Meta Object MO_Server_Datahandler에 추가하십시오.

기타 Business Object 설정

Data Handler 코딩 외에도, Data Handler에 맞게 Business Object를 설정해야 합니다. Data Handler의 요구사항과 호출자 요구사항을 따르는 Business Object 정의를 작성하십시오.

팁: Data Handler에서 필요로 하는 Business Object가 Data Handler를 사용할 커넥터에 대해 지원되는 오브젝트 목록에 있는지 확인하십시오.

커넥터 구성

Custom Data Handler를 커넥터 문맥에서 사용할 경우, 각 커넥터를 구성하여 최상위 레벨 커넥터 Meta Object의 이름을 확보해야 합니다. 커넥터는 Data Handler Meta Object 이름 및 클래스 이름에 대한 정보를 여러 방법으로 가져옵니다. 예를 들면, 다음과 같습니다.

- Data Handler를 사용하도록 WebSphere Business Integration Adapter for XML을 구성하기 위해, `DataHandlerConfigMO` 커넥터 구성 등록 정보를 설정하고 XML 커넥터의 Business Object에서 `MimeType` 속성을 설정합니다.
- Data Handler를 사용하도록 WebSphere Business Integration Adapter for JText 커넥터를 구성하기 위해, JText 구성 Meta Object에서 `ClassName`이나, `DataHandlerConfigMO` 및 `MimeType` 속성을 설정합니다.

자세한 정보는 33 페이지의 『Data Handler를 사용할 커넥터 구성』을 참조하십시오.

팁: Data Handler를 사용하도록 커넥터를 구성할 경우, Meta Object 이름의 철자가 올바른지 확인하고 MIME 유형 철자가 올바른지 확인하십시오.

국제화된 Data Handler

*국제화된 Data Handler*는 특정 로케일에 대해 사용자 정의할 수 있는 방법으로 작성된 Data Handler입니다. *로케일*은 일반 사용자의 특정 국가, 언어 또는 지역 고유 데이터를 처리하는 방법에 대한 정보를 가져오는 사용자 환경의 일부입니다. 로케일은 일반적으로 운영 체제의 일부로서 설치됩니다. 로케일 감지 데이터를 처리하는 Data Handler 작성을 Data Handler의 *국제화*(I18N)라고 합니다. 특정 로케일에 대해 국제화된 Data Handler 준비를 Data Handler의 *로컬화*(L10N)라고 합니다.

이 섹션에서는 국제화된 Data Handler에 대한 다음 정보를 제공합니다.

- 로케일의 개념
- 국제화된 Data Handler에 대한 설계 고려사항

로케일의 개념

*로케일*은 사용자 환경에 대한 다음 정보를 제공합니다.

- 언어 및 국가(또는 지역)에 따른 문화적 규약
  - 데이터 형식:
    - 날짜: 날짜 구조(날짜 분리문자 포함)는 물론 요일과 월에 대한 전체 및 축약 이름을 정의하십시오.
    - 숫자: 천 단위 분리문자와 소수점에 대한 기호 및 숫자 내 기호의 위치를 정의하십시오.
    - 시간: 시간 구조는 물론 12시간제 표시기(AM 및 PM 표시기)를 정의하십시오.
    - 통화 값: 숫자와 통화 기호 및 통화 값 내 기호의 위치를 정의하십시오.
  - 대조(Collation) 순서: 특정 문자 코드 세트와 언어에 대한 데이터 정렬 방법
  - 문자열 처리에는 "대소문자"(대문자 및 소문자) 비교, 하위 문자열 및 연결과 같은 *타스크*가 포함됩니다.
- *문자 인코딩* -- 문자(영문자)에서 문자 코드 세트의 숫자 값으로 맵핑 예를 들어, ASCII 문자 코드 세트는 문자 "A"를 65로 인코드하는 반면, EBCIDIC 문자 세트는 이 문자를 43으로 인코드합니다. *문자 코드 세트*에는 한 개 이상 언어 영문자의 모든 문자에 대한 인코딩이 포함됩니다.

로케일 이름은 다음 형식을 지닙니다.

ll_TT.codeset

여기서 *ll*은 2문자 언어 코드(보통 소문자)이고 *TT*는 2문자 국가 및 지역 코드(보통 대문자)이며 *codeset*는 연관된 문자 코드 세트의 이름입니다. 때때로 이름의 *codeset* 위치는 선택적입니다. 로케일은 일반적으로 운영 체제 설치의 일부로서 설치됩니다.

국제화된 Data Handler에 대한 설계 고려사항

이 섹션에서는 Data Handler의 국제화를 위한 설계 고려사항 카테고리를 제공합니다.

- 로케일별 설계 원칙
- 문자 인코딩 설계 원칙

로케일별 설계 원칙

국제화하기 위해서는 Data Handler를 로케일 감지 Data Handler로 코딩해야 합니다. 즉, 동작은 로케일 설정을 고려하여 해당 로케일에 적절한 *타스크*를 수행해야 합니다. 예를 들어, 영어를 사용하는 로케일의 경우 Data Handler는 영어 메시지 파일에서 오류 메시지를 가져와야 합니다. 제품과 함께 설치한 Data Handler 프레임워크는 국제화되어 있습니다. 개발하는 Data Handler의 국제화(I18N)를 완료하려면, Data Handler 구현이 국제화되었는지 확인해야 합니다.

국제화된 Data Handler는 로케일 감지 설계 원칙 세트를 따라야 합니다.

- 모든 오류, 상태 및 추적 메시지의 텍스트는 메시지 파일의 Data Handler 구현에서 분리하여 로케일 언어로 변환해야 합니다.
- 데이터의 정렬 또는 대조는 로케일의 언어 및 국가에 적절한 대조(collation) 순서를 사용합니다.
- 문자열 처리(예: 비교, 하위 문자열 및 대소문자)는 로케일 언어의 문자에 적절합니다.
- 날짜, 숫자 및 시간의 형식은 로케일에 적절합니다.

Data Handler는 직렬화된 데이터 응용프로그램과 Business Object 간을 변환할 때, 로케일 감지 처리(예: 데이터 형식 변환)를 수행해야 하는 경우도 있습니다. Data Handler 환경과 연관된 로케일을 추적하기 위해 `DataHandler` 클래스는 Data Handler가 실행되는 운영 체제의 로케일로 *초기화되는* 개인용 로케일 변수를 갖습니다. 표 78의 액세서 메소드를 통해 런타임 시 Data Handler 환경의 로케일(이 개인용 `locale` 변수의 값)에 액세스할 수 있습니다.

표 78. Data Handler 환경의 로케일 액세스 메소드

Data Handler 클래스	**메소드**
`DataHandler`	getLocale(),setLocale()

Business Object는 작성될 때 데이터와 연관된 로케일을 갖습니다. 이 로케일은 Business Object 정의 또는 속성 이름이 *아닌* Business Object의 데이터에 적용됩니다(미국 영어 로케일 `en_US`와 연관된 코드 세트의 문자여야 함). Business Object를 작성하기 위해, Data Handler는 표 79에 나와 있는 메소드를 사용할 수 있습니다. 이 메소드는 `DataHandler` 클래스의 개인용 로케일 변수에 대한 액세스 권한을 갖습니다. 이 메소드 중 하나가 Business Object를 작성하는 경우, 해당 메소드는 개인용 `DataHandler` 로케일 변수가 지정하는 로케일을 이 Business Object와 연관시킵니다.

주: 표 79의 메소드는 최상위 레벨 Business Object에서*만* 로케일을 설정합니다. Business Object에 하위 Business Object가 포함되는 경우, 하위는 시스템 기본값으로 설정된 로케일을 가지므로 올바른 로케일 값을 가지지 못할 수 있습니다.

표 79의 메소드를 사용하여 Business Object를 작성하고 데이터에 대한 로케일을 설정하십시오. 개인용 로케일 변수가 Business Object의 데이터에 대한 올바른 로케일을 지정하도록 하기 위해, 표 79의 메소드 중 하나를 호출하기 *전*에 `setLocale()` 메소드를 사용할 수 있습니다.

표 79. Business Object에 로케일을 지정하는 메소드

Data Handler 클래스	**메소드**
`DataHandler`	getBO() - 공용,getBOName()

문자 인코딩 설계 원칙

하나의 코드 세트를 사용하는 위치에서 다른 코드 세트를 사용하는 위치로 데이터를 전송하는 경우, 의미를 보존하기 위해 데이터에 대해 몇몇 양식의 문자 변환을 수행해야 합니다. Java(Java Virtual Machine) 내의 Java 런타임 환경은 데이터를 Unicode 문자 세트로 표시합니다. Unicode 문자 세트는 잘 알려져 있는 문자 코드 세트(1바이트 및 복수 바이트 모두)의 문자에 대한 인코딩을 포함하는 범용 문자 세트입니다. 여러 인코딩 형식의 Unicode가 있습니다. 다음 인코딩은 통합 비즈니스 시스템에서 가장 자주 사용됩니다.

- 범용 복수 8진수 코드화 문자 세트: UCS-2

  UCS-2 인코딩은 2바이트(8진수)로 인코드된 Unicode 문자 세트입니다.

- UCS Transformation Format, 8비트 양식: UTF-8

  UTF-8 인코딩은 UNIX 환경에서 Unicode 문자 데이터 사용을 설명하기 위해 설계되었습니다. true ASCII 코드를 제외한 다른 것으로 해석되는 일이 없도록 모든 ASCII 코드 값(0...127)을 지원합니다. 각 코드 값은 보통 1, 2 또는 3바이트 값으로 표시됩니다.

IBM WebSphere Business Integration System 구성요소의 대부분은 Java로 작성되었습니다. 따라서 데이터를 대부분의 시스템 구성요소 사이에서 전송하는 경우, Unicode 코드 세트로 인코딩되므로 문자 변환은 필요없습니다.

Data Handler는 Java로 작성된 구성요소이므로, 직렬화된 데이터를 Unicode 코드 세트로 처리합니다. 보통 데이터 입력 스트림의 소스도 Unicode로 처리됩니다. 따라서 Data Handler는 일반적으로 직렬화된 데이터의 문자 변환을 수행하지 않아도 됩니다. 그러나 입력 또는 출력 데이터에 문자 인코딩이 시스템 기본값과 동일하지 않은 바이트 배열이 포함된 경우, Data Handler는 문자 인코딩을 제공해야 합니다.

Data Handler 환경과 연관된 문자 인코딩을 추적하기 위해, `DataHandler` 클래스는 Data Handler가 실행되는 운영 체제의 로케일과 연관된 문자 인코딩으로 초기화되는 개인용 문자 인코딩 변수를 갖습니다. 표 80의 액세서 메소드를 통해 런타임 시 Data Handler 환경의 문자 인코딩(이 개인용 문자 인코딩 변수의 값)에 액세스할 수 있습니다.

표 80. Data Handler의 문자 인코딩 검색 메소드

Data Handler 클래스	**메소드**
`DataHandler`	getEncoding(),setEncoding()

제 11 장 Data Handler 기본 클래스 메소드

DataHandler 클래스는 Data Handler의 기본 클래스입니다. 이것은 com.crossworlds.DataHandlers 패키지에 포함되어 있습니다. Custom Data Handler를 포함하여, 모든 Data Handler는 이 추상 클래스를 확장해야 합니다.

이 장에서 설명하는 메소드는 세 가지의 카테고리에 속합니다.

- 구현해야 하는 추상 메소드
- 제공된 구현이 있으며 필요할 경우 대체할 수 있는 공용 메소드
- 커넥터 또는 액세스 클라이언트가 호출하는 정적 메소드

표 81에 DataHandler 클래스의 메소드가 나열되어 있습니다.

표 81. DataHandler 클래스의 구성원 메소드

구성원 메소드	유형	설명	페이지
createHandler()	Public Static	Data Handler의 인스턴스를 작성합니다.	225
getBO() - 추상	추상	Business Object를 직렬화된 입력 데이터에서 추출한 값으로 채웁니다.	227
getBO() - 공용	공용	직렬화된 데이터를 Business Object로 변환합니다.	228
getBOName()	공용	직렬화된 데이터 내용을 기초로 Business Object 이름을 가져옵니다.	230
getBooleanOption()	공용	부울 데이터를 포함할 경우, 지정된 Data Handler 구성 옵션 값을 가져옵니다.	231
getByteArrayFromBO()	공용	Business Object를 바이트 배열로 직렬화합니다.	232
getEncoding()	공용	Data Handler가 사용 중인 문자 인코딩을 검색합니다.	233
getLocale()	공용	Data Handler의 로케일을 검색합니다.	233
getOption()	공용	지정된 Data Handler 구성 옵션(설정한 경우) 값을 가져옵니다.	234
getStreamFromBO()	추상	Business Object를 InputStream 오브젝트로 직렬화합니다.	235
getStringFromBO()	추상	Business Object를 String 오브젝트로 직렬화합니다.	236
setConfigMOName()	정적	DataHandler 기본 클래스의 정적 등록 정보에서 최상위 레벨 Data Handler Meta Object의 이름을 설정합니다.	237
setEncoding()	공용	Data Handler가 사용 중인 문자 인코딩을 설정합니다.	237
setLocale()	공용	Data Handler의 로케일을 설정합니다.	238
setOption()	공용	지정된 Data Handler 구성 옵션 값을 설정합니다.	239
traceWrite()	공용	적절한 trace-write 함수를 호출하여 Data Handler의 추적 메시지를 작성합니다.	240

createHandler()

Data Handler의 인스턴스를 작성합니다.

구문

```
public static DataHandler createHandler(String className,
    String mimeType, String BOPrefix);
```

매개변수

className 작성할 Data Handler 인스턴스의 클래스 이름. 지정하지 않는 경우, 메소드가 *mimeType* 인수를 사용하여 인스턴스화할 data-handler 클래스를 판별합니다.

mimeType 작성할 Data Handler 인스턴스의 MIME 유형을 지정합니다. 제공되지 않을 경우, 메소드는 제공할 *className* 값을 예상합니다. Meta Object에 대한 키. *BOPrefix*를 제공하면, *mimeType*은 키의 일부가 됩니다.

BOPrefix 선택적 매개변수. 존재할 경우, *mimeType*과 결합하여 Meta Object에 대한 키를 형성합니다. 이 인수를 사용하여 MIME 부속 유형을 지정할 수 있습니다. 또한 Data Handler 등록 정보 BOPrefix를 설정하는 데 사용할 수도 있습니다.

리턴값

Data Handler의 인스턴스

예외

Exception

메소드가 Data Handler의 인스턴스를 작성할 수 없는 경우 발행됩니다.

참고

이 메소드는 *className*, *mimeType* 및 *BOPrefix* 매개변수 값을 기초로 Data Handler의 인스턴스를 작성합니다.

- 커넥터에서 createHandler() 메소드를 호출할 경우, 커넥터는 *className* 값을 지정할 수 있습니다. *className*을 지정하면, createHandler()는 해당되는 클래스 이름의 Data Handler를 인스턴스화합니다.
- *mimeType*을 지정하면, createHandler()는 지정한 MIME 유형을 기초로 Data Handler를 작성합니다.

  메소드는 이름이 *mimeType* 매개변수나 *mimeType* 및 *BOPrefix* 조합의 내용 유형과 일치하는 속성에 대해 최상위 레벨 Data Handler Meta Object를 확인합니다. 일치하는 속성이 발견되면, 하위 Meta Object에서 ClassName 속성 값이 클래스 이름으로 사용됩니다.

메소드가 Data Handler에 대해 클래스의 인스턴스를 생성하는 데 성공하면, setupOptions()를 호출하여 Data Handler 인스턴스에서 사용할 구성 등록 정보를 설정합니다. createHandler()가 Data Handler의 인스턴스화 하는 방법에 대한 자세한 설명은 15 페이지의 『Data Handler 인스턴스화』를 참조하십시오.

예를 들어, MIME = "text/xml-application-xxx"의 경우 메소드는 com.crossworlds.DataHandlers.text.xml_application_xxx 클래스를 로드합니다.

getBO() - 추상

Business Object를 직렬화된 입력 데이터에서 추출한 값으로 채웁니다.

구문

```
public abstract void getBO(Reader serializedData,
        BusinessObjectInterface theBusObj, Object config);

public abstract BusinessObjectInterface getBO(Reader serializedData,
        Object config);
```

매개변수

serializedData 직렬화된 데이터에 액세스하는 Java Reader 오브젝트

theBusObj 데이터로 채울 Business Object

config Data Handler에 대한 추가 구성 정보가 들어 있는 선택적 오브젝트

리턴값

이 메소드의 첫 번째 양식에는 리턴값이 없습니다. 두 번째 양식은 Business Object를 리턴합니다.

참고

이 getBO() 메소드는 Data Handler에 대해 문자열에서 Business Object로의 변환을 수행하는 추상 메소드입니다. 즉, 직렬화된 일반 데이터(Reader 오브젝트로 액세스)를 Business Object로 변환하는 방법을 정의합니다. 이 메소드의 양식은 두 가지입니다.

- 첫 번째 양식은 호출자가 전달한 비어 있는 Business Object인 *theBusObj*입니다.
- 두 번째 양식은 Business Object 인스턴스를 작성하여 이 인스턴스를 채웁니다.

주: 서버 액세스 인터페이스의 문맥에서 호출되는 Data Handler는 getBO() 메소드의 두 번째 양식에 *대해서만* 기능을 제공해야 합니다.

중요: getBO() 메소드는 기본 구현이 없는 추상 메소드이므로, Data Handler 클래스가 이 메소드를 *구현해야 합니다*.

직렬화된 데이터는 Java Reader 오브젝트로 getBO()에 전달합니다. 그러나 Reader는 기본 클래스이므로, 실제로 Reader 클래스의 여러 서브클래스 중 하나의 인스턴스를 전달합니다. 일부 Reader 서브클래스는 mark() 조작에 대한 구현을 제공하지만, 일부는 그렇지 않습니다. mark() 조작은 호출자가 스트림 내에서 특정 위치를 표시한 후 순차적으로 해당 지점에 리턴하도록 합니다.

주: Reader 옵션을 XML 또는 EDI Data Handler의 getBO() 메소드에 전달하려면, Reader 서브클래스가 mark() 메소드를 구현하도록 해야 합니다. Reader 클래스의 isMarkSupported() 메소드를 호출하여 이 메소드가 사용 중인 Reader 오브젝트를 지원하는지 여부를 판별할 수 있습니다. 직렬화된 데이터는 StringReader 오브젝트로 전달하는 것이 좋습니다.

Meta Object에 포함된 것보다 더 많은 구성 정보를 Data Handler에 제공해야 할 경우, *config* 옵션을 사용하여 이러한 정보를 포함하는 오브젝트를 전달할 예를 들어, *config*는 Business Object로부터 XML 문서를 빌드하기 위해 사용하는 스키마에 대한 URL 문자열 또는 템플리트 파일이 될 수 있습니다.

*config*가 Business Object 유형이면, getBO() 메소드를 구현하여 setupOptions(*config*)를 호출할 수 있습니다. setupOptions() 메소드는 DataHandler 기본 클래스에서 정의합니다. 이 메소드는 Business Object에서 등록 정보 이름으로 속성 이름을 사용하고 등록 정보에 대한 값으로 기본값을 사용합니다. Data Handler에서 사용하기 위해 오브젝트에 있는 구성 등록 정보의 값을 설정합니다.

추상 getBO() 메소드를 구현하면, Data Handler를 호출하는 구성요소가 표 82에서처럼 공용 문자열에서 Business Object로의 변환 메소드 중 하나를 호출할 수 있습니다.

표 82. 공용 문자열에서 Business Object로의 변환 메소드

공용 문자열에서 **Business Object**로의 변환 메소드	설명
getBO(Object *serializedData*, Object *config*)	일반 Object의 직렬화된 데이터를 Business Object로 변환
getBO(String *serializedData*, Object *config*)	Object의 직렬화된 데이터를 Business Object로 변환
getBO(InputStream *serializedData*, Object *config*)	InputStream의 직렬화된 데이터를 Business Object로 변환
getBO(byte[] *serializedData*, Object *config*)	바이트 배열의 직렬화된 데이터를 Business Object로 변환

참조

getBO() - 공용

---

getBO() - 공용

직렬화된 데이터를 Business Object로 변환합니다.

구문

```
public BusinessObjectInterface getBO(Object serializedData,
    Object config);

public BusinessObjectInterface getBO(String serializedData,
      Object config);
```

```
public BusinessObjectInterface getBO(InputStream serializedData,
      Object config);

public BusinessObjectInterface getBO(byte[] serializedData,
      Object config);

public void getBO(Object serializedData,
    BusinessObjectInterface theBusObj, Object config);
```

매개변수

serializedData 직렬화된 데이터에 대한 참조

theBusObj 데이터로 채울 Business Object

config Data Handler에 대한 추가 구성 정보가 들어 있는 선택적 오브젝트

리턴값

처음 네 가지 양식은 입력 Object, String, InputStream 또는 바이트 배열 오브젝트의 데이터로 채워진 Business Object를 리턴합니다. 5번째 양식은 직렬화된 데이터로 지정된 Business Object를 채웁니다.

예외

Exception

메소드가 직렬화된 데이터를 Business Object로 변환할 수 없는 경우 발행됩니다.

NotImplementedException

getBO() 메소드의 공용 버전이 구현되지 않은 경우 발행됩니다.

참고

이 getBO() 메소드는 문자열에서 Business Object로의 변환을 수행하는 공용 메소드입니다. DataHandler 기본 클래스에는 getBO()의 추상 양식이 있으며(227 페이지의 설명 참조), 이는 data-handler 클래스의 일부로 구현해야 합니다. 이 getBO() 공용 버전은 구성요소(예: 커넥터 또는 액세스 클라이언트)가 *serializedData*를 Object, String, InputStream 오브젝트 또는 바이트 배열로 지정하도록 허용하는 한 세트의 유틸리티 메소드를 정의합니다. 메소드는 지정된 직렬화된 데이터를 Reader 오브젝트로 변환한 후 추상 getBO() 메소드 중 하나를 호출하여 Reader 오브젝트를 Business Object로 변환합니다.

공용 getBO() 메소드의 양식은 다음과 같습니다.

- 첫 번째 양식은 *serializedData* 오브젝트 유형을 기반으로 적절한 getBO() 메소드를 호출합니다. 예를 들어, 데이터 유형이 String일 경우, 메소드는 getBO(String*serializedData*, Object*config*)를 호출합니다.

- 두 번째, 세 번째 및 네 번째 양식은 String, InputStream 또는 바이트 배열 오브젝트에서 Reader 오브젝트를 작성하고 추상 getBO() 메소드를 호출하여 Reader 오브젝트를 Business Object로 변환합니다.
- 다섯번째 양식은 호출자가 Business Object에서 전달할 때 getBO() 호출을 처리하는 또다른 유틸리티입니다. 이는 다음과 같은 작업을 수행합니다.
  - 전달된 serializedData 오브젝트 유형을 판별합니다.
  - 적절할 경우, 오브젝트를 String 또는 InputStream 오브젝트로 변환합니다.
  - 추상 버전을 호출하여 Business Object와 데이터를 전달합니다.

config 인수에 대한 정보는 getBO()의 추상 양식(227 페이지의 설명 참조) 아래에 있는 설명을 살펴보십시오.

참조

getBO() - 추상

---

getBOName()

직렬화된 데이터 내용을 기초로 Business Object 이름을 가져옵니다.

구문

```
public String getBOName(Reader serializedData);
public String getBOName(String serializedData);
public String getBOName(InputStream serializedData);
```

매개변수

serializedData 메시지를 포함하는 Reader 오브젝트에 대한 참조

리턴값

Business Object 이름을 포함하는 String 오브젝트를 리턴합니다. NameHandlerClass 속성에 대해 값이 존재하지 않을 경우, 이 메소드는 null을 리턴합니다.

예외

getBOName()의 두 번째 및 세 번째 양식은 다음 예외를 발행할 수 있습니다.

MalformedDataException

직렬화된 데이터(*serializedData*)의 형식이 올바르지 않은 경우 발행됩니다.

NotImplementedException

이름 핸들러가 구현되지 않은 경우 발행됩니다.

참고

getBOName() 메소드는 네임 핸들러의 인스턴스를 작성하여 직렬화된 데이터에서 Business Object 정의의 이름을 추출합니다. NameHandlerClass Meta Object 속성 값을 기반으로 이 name-handler 오브젝트를 인스턴스화합니다. 네임 핸들러는 메시지 내용을 기초로 Business Object 이름을 빌드합니다.

getBOName() 메소드의 양식은 다음과 같습니다.

- 첫 번째 양식은 BOPrefix Meta Object 속성(존재할 경우)과 NameHandler 클래스의 리턴값을 기반으로 Business Object 이름을 리턴합니다.
- 두 번째 양식은 String에서 Reader 오브젝트를 작성하고 첫 번째 양식을 호출합니다.
- 세 번째 양식은 InputStream에서 Reader 오브젝트를 작성하고 첫 번째 양식을 호출합니다.

현재, 다음과 같은 IBM 제공 다음 Data Handler만 이 메소드를 사용합니다.

- XML Data Handler

  XML Data Handler의 기본 네임 핸들러는 기본 클래스인 getBOName (Reader*data*)을 호출합니다. Data Handler가 요청을 처리할 수 없으면, <!DOCTYPE Name을 사용하여 Business Object의 기본 이름을 추출합니다. 최종 이름 양식은 다음과 같습니다.

  ```
  BOPrefix + "_" + Name.getStreamFromBO()
  ```

- EDI Data Handler

  EDI Data Handler에 대한 기본 네임 핸들러는 EDI 네임 핸들러 찾아보기 테이블에서 Business Object 이름을 확보합니다.

사용자의 네임 핸들러를 작성하려면, NameHandler 추상 기본 클래스를 확장하여 getBOName() 메소드를 대체하십시오.

자세한 정보는 102 페이지의 『사용자 정의 XML 네임 핸들러 빌드』를 참조하십시오.

getBooleanOption()

부울 데이터를 포함할 경우, 지정된 Data Handler 구성 옵션 값을 가져옵니다.

구문

```
public boolean getBooleanOption(String name);
```

매개변수

name 구성 옵션의 이름

리턴값

Boolean 유형 옵션의 값을 리턴합니다.

참고

createHandler() 메소드는 구성 옵션을 초기화하기 위해 Data Handler와 연관된 하위 Meta Object를 사용합니다. 옵션에 부울 값이 있으면, getBooleanOption() 메소드를 사용하여 옵션 중 하나의 값을 확보할 수 있습니다.

---

getByteArrayFromBO()

Business Object를 바이트 배열로 직렬화합니다.

구문

```
abstract byte[] getByteArrayFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

매개변수

theBusObj 바이트 배열로 변환할 Business Object

config Data Handler의 추가 구성 정보를 포함하는 선택적 오브젝트

리턴값

지정된 Business Object를 표시하는 직렬화된 데이터를 포함하는 byte 배열

예외

Exception

메소드가 Business Object를 직렬화된 데이터의 바이트 배열로 변환할 수 없는 경우 발행됩니다.

참고

getByteArrayFromBO() 메소드는 Data Handler에 대한 Business Object에서 바이트 배열로의 변환을 수행합니다. *theBusObj* Business Object의 데이터를 바이트 배열(Java byte[] 오브젝트)로 변환합니다.

중요: getByteArrayFromBO() 메소드는 기본 구현이 없는 추상 메소드이므로 Data Handler 클래스가 *반드시* 이 메소드를 구현해야 합니다.

Meta Object에 포함된 것보다 더 많은 구성 정보를 Data Handler에 제공해야 할 경우, *config* 옵션을 사용하여 이러한 정보를 포함하는 오브젝트를 전달할 예를 들어, *config*는 Business Object로부터 XML 문서를 빌드하기 위해 사용하는 스키마에 대한 URL 문자열 또는 템플리트 파일이 될 수 있습니다.

*config*가 Business Object 유형인 경우, getByteArrayFromBO() 메소드를 구현하여 setupOptions(*config*)를 호출할 수 있습니다. setupOptions() 메소드는 DataHandler 기본 클래스에서 정의합니다. 이 메소드는 Business Object에서 등록 정보 이름으로 속성 이름을 사용하고 등록 정보에 대한 값으로 기본값을 사용합니다. Data Handler에서 사용하기 위해 오브젝트에 있는 구성 등록 정보의 값을 설정합니다.

참조

getBO() - 공용, getStreamFromBO(), getStringFromBO()

---

getEncoding()

Data Handler가 사용 중인 문자 인코딩을 검색합니다.

구문

```
public final String getEncoding();
```

매개변수

없음

리턴값

Data Handler의 문자 인코딩을 포함하는 String

참고

getEncoding() 메소드는 Data Handler의 문자 인코딩을 검색합니다. 문자 인코딩은 언어, 국가(또는 지역)에 따라 데이터에 대한 문화적 규약을 정의하는 로케일의 일부입니다. 이 메소드는 DataHandler 클래스의 개인용 문자 인코딩 변수 값을 검색하는 액세서 메소드입니다. 이 문자 인코딩은 Data Handler가 처리 중인 직렬화된 데이터의 문자 인코딩을 나타내야 합니다.

이 메소드는 Data Handler가 문자 인코딩 처리(예: 문자 변환)를 수행해야 할 경우 유용합니다.

참조

setEncoding()

---

getLocale()

Data Handler의 로케일을 검색합니다.

구문

```
public final Locale getLocale();
```

매개변수

없음

리턴값

Data Handler의 환경에 대한 로케일을 설명하는 Java `Locale` 오브젝트

참고

`getLocale()` 메소드는 언어, 국가(또는 지역) 및 문자 인코딩에 따라 데이터에 대한 문화적 규약을 정의하는 Data Handler 로케일을 검색합니다. 이 메소드는 `DataHandler` 클래스의 개인용 문자 인코딩 변수 값을 검색하는 액세서 메소드입니다. 기본적으로, Data Handler의 로케일은 Data Handler가 실행되는 운영 체제의 로케일입니다.

이 메소드는 Data Handler가 로케일 감지 처리를 수행해야 할 경우 유용합니다.

참조

setLocale()

---

getOption()

지정된 Data Handler 구성 옵션(설정한 경우) 값을 가져옵니다.

구문

```
public String getOption(String name);
```

매개변수

name 구성 옵션의 이름

리턴값

옵션 값을 포함하는 `String` 오브젝트

참고

`getOption()` 메소드는 구성 옵션의 값을 확보합니다. Data Handler가 연관된 하위 Meta Object를 가지고 있는 경우, `createHandler()` 메소드는 이러한 Meta Object 속성의 기본값을 사용하여 구성 옵션을 초기화합니다. `getOption()` 메소드를 사용하여 옵션 중 하나의 값을 확보할 수 있습니다.

참조

setOption()

---

getStreamFromBO()

Business Object를 InputStream 오브젝트로 직렬화합니다.

구문

```
abstract InputStream getStreamFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

매개변수

theBusObj 스트림으로 변환할 Business Object

config Data Handler의 추가 구성 정보를 포함하는 선택적 오브젝트

리턴값

Business Object를 표시하는 직렬화된 데이터를 포함하는 InputStream 오브젝트

예외

Exception

메소드가 Business Object를 직렬화된 데이터의 스트림으로 변환할 수 없는 경우 발행됩니다.

참고

getStreamFromBO() 메소드는 Data Handler에 대해 Business Object에서 스트림으로의 변환을 수행합니다. *theBusObj* Business Object의 데이터를 스트림(Java InputStream 오브젝트)으로 변환합니다.

중요: getStreamFromBO() 메소드는 기본 구현이 없는 추상 메소드이므로, Data Handler 클래스가 이 메소드를 구현*해야 합니다*.

Meta Object에 포함된 것보다 더 많은 구성 정보를 Data Handler에 제공해야 할 경우, config 옵션을 사용하여 이러한 정보를 포함하는 오브젝트를 전달할 수 있습니다. 예를 들어, *config*는 Business Object로부터 XML 문서를 빌드하기 위해 사용하는 스키마에 대한 URL 문자열 또는 템플리트 파일이 될 수 있습니다.

config가 Business Object 유형일 경우, getStreamFromBO() 메소드를 구현하여 setupOptions(*config*)를 호출할 수 있습니다. setupOptions() 메소드는 DataHandler 기본 클래스에서 정의합니다. 이 메소드는 Business Object에서 등록 정

보 이름으로 속성 이름을 사용하고 등록 정보에 대한 값으로 기본값을 사용합니다. Data Handler에서 사용하기 위해 오브젝트에 있는 구성 등록 정보의 값을 설정합니다.

참조

getBO() - 공용, getByteArrayFromBO(), getStringFromBO()

---

getStringFromBO()

Business Object를 String 오브젝트로 직렬화합니다.

구문

```
abstract String getStringFromBO(BusinessObjectInterface theBusObj,
      Object config);
```

매개변수

theBusObj String으로 변환할 Business Object

config Data Handler의 추가 구성 정보를 포함하는 선택적 오브젝트

리턴값

Business Object에 있는 데이터를 표시하는 직렬화된 데이터를 포함하는 String 오브젝트

예외

Exception

메소드가 Business Object를 직렬화된 데이터의 문자열로 변환할 수 없는 경우 발행됩니다.

참고

getStringFromBO() 메소드는 Data Handler에 대해 Business Object에서 문자열로의 변환을 수행합니다. *theBusObj* Business Object의 데이터를 Java String 오브젝트로 변환합니다.

중요: getStringFromBO() 메소드는 기본 구현이 없는 추상 메소드이므로, Data Handler 클래스가 이 메소드를 구현*해야 합니다*.

Meta Object에 포함된 것보다 더 많은 구성 정보를 Data Handler에 제공해야 할 경우, config 옵션을 사용하여 이러한 정보를 포함하는 오브젝트를 전달할 수 있습니다. 예를 들어, *config*는 Business Object로부터 XML 문서를 빌드하기 위해 사용하는 스키마에 대한 URL 문자열 또는 템플리트 파일이 될 수 있습니다.

config가 Business Object 유형일 경우, getStreamFromBO() 메소드를 구현하여 setupOptions(*config*)를 호출할 수 있습니다. setupOptions() 메소드는 DataHandler 기본 클래스에서 정의합니다. 이 메소드는 Business Object에서 등록 정보 이름으로 속성 이름을 사용하고 등록 정보에 대한 값으로 기본값을 사용합니다. Data Handler에서 사용하기 위해 오브젝트에 있는 구성 등록 정보의 값을 설정합니다.

참조

getBO() - 공용, getByteArrayFromBO(), getStreamFromBO()

---

setConfigMOName()

DataHandler 기본 클래스의 정적 등록 정보에서 최상위 레벨 Data Handler Meta Object의 이름을 설정합니다.

구문

```
public static void setConfigMOName(String name);
```

매개변수

name Data Handler Meta Object의 이름을 포함하는 문자열

리턴값

없음

예외

Exception

메소드가 지정된 최상위 레벨 Data Handler Meta Object를 설정하는 경우 발행됩니다.

참고

최상위 레벨 Data Handler Meta Object는 지원되는 MIME 유형과 연관된 하위 Meta Object의 이름을 보유합니다. Data Handler Meta Object에 대한 정보는 28 페이지의 『Data Handler 구성』을 참조하십시오.

---

setEncoding()

Data Handler가 사용 중인 문자 인코딩을 설정합니다.

구문

```
public final void setEncoding(String encodingName);
```

매개변수

encodingName

Data Handler의 문자 인코딩으로 지정할 새 값을 포함하는 `String` 오브젝트

리턴값

없음

참고

`setEncoding()` 메소드는 Data Handler의 문자 인코딩을 설정합니다. 문자 인코딩은 언어, 국가(또는 지역)에 따라 데이터에 대한 문화적 규약을 정의하는 로케일의 일부입니다. 이 메소드는 `DataHandler` 클래스의 개인용 문자 인코딩 변수를 설정하는 액세서 메소드입니다. 이 문자 인코딩은 Data Handler가 처리 중인 직렬화된 데이터의 문자 인코딩을 나타내야 합니다.

이 메소드는 Data Handler가 문자 인코딩 처리(예: 문자 변환)를 수행해야 할 경우 유용합니다.

참조

getEncoding()

---

setLocale()

Data Handler의 로케일을 설정합니다.

구문

```
public final void setLocale(Locale localeObject);
```

매개변수

localeObject

Data Handler의 환경에 지정할 새 로케일을 포함하는 Java `Locale` 오브젝트.

리턴값

없음

참고

`setLocale()` 메소드는 언어, 국가(또는 지역) 및 문자 인코딩에 따라 데이터에 대한 문화적 규약을 정의하는 Data Handler 로케일을 검색합니다. 이 메소드는 `DataHandler` 클래스의 개인용 로케일 변수를 설정하는 액세서 메소드입니다. 이 로케일은 Data Handler가 받아 작성하는 직렬화된 데이터의 로케일을 나타내야 합니다.

이 메소드는 Data Handler가 Data Handler의 로케일을 변경해야 할 경우 유용합니다. Business Object로 변환할 직렬화된 데이터에 대해 다른 로케일을 지정할 수 있습니다. `getBO()` 호출 이전에 `setLocale()`을 호출하면 `getBO()`가 작성하는 Business Object를 로케일과 연관시킬 때 사용하는 로케일이 변경됩니다.

참조

getLocale()

---

setOption()

지정된 Data Handler 구성 옵션 값을 설정합니다.

구문

```
public void setOption(String name, String value);
```

매개변수

name 구성 옵션의 이름

value 구성 옵션의 값

리턴값

없음

참고

`setOption()` 메소드는 새 값을 구성 옵션에 지정합니다. Data Handler가 연관된 하위 Meta Object를 가지고 있는 경우, `createHandler()` 메소드는 이러한 Meta Object 속성의 기본값을 사용하여 구성 옵션을 초기화합니다. Data Handler를 커넥터 문맥으로 호출할 경우, 이 하위 Meta Object는 커넥터 프로세스 메모리에 있습니다. 통합 브로커가 InterChange Server이고 Data Handler가 액세스 클라이언트 문맥으로 호출될 경우, Meta Object는 InterChange Server 프로세스의 메모리에 있습니다. `setOption()` 메소드를 사용하여 옵션 중 하나의 값을 대체할 수 있습니다.

주: `setOption()`으로 구성 옵션을 변경하면 메모리에서 Meta Object 속성 값이 설정됩니다. 저장소에 있는 속성 값에는 영향을 주지 않습니다.

참조

getOption()

traceWrite()

적절한 trace-write 함수를 호출하여 Data Handler의 추적 메시지를 작성합니다.

구문

```
public void traceWrite(String message, int level);
```

매개변수

message 추적 메시지에 사용할 메시지 텍스트

level 메시지의 추적 레벨을 지정하는 정수. 추적 레벨은 0 - 5입니다. 여기서 0은 추적하지 않음을 지정하고 5는 전체 추적을 지정합니다.

리턴값

없음

참고

`traceWrite()` 메소드는 Data Handler가 실행하는 문맥에 따라 적절한 trace write 함수를 호출하는 랩퍼 메소드입니다. 기본 추적은 커넥터 추적입니다. 통합 브로커가 InterChange Server이고 Data Handler가 서버 액세스 인터페이스 문맥으로 실행될 경우, `traceWrite()` 메소드는 trace-write 메소드를 호출하기 전에 추적 서브시스템을 설정합니다.

부록. XML ODA 사용

이 장에서는 XML 문서의 Business Object 정의를 생성하는 XML ODA(Object Discovery Agent)에 대해 설명합니다. XML 문서는 DTD(문서 유형 정의) 또는 스키마 문서가 정의하는 스키마를 가지기 때문에 XMLODA는 XML 문서에 특정한 Business Object 요구사항를 발견하기 위해 이들 데이터 모델 중 하나를 사용할 수 있습니다.

이 장은 다음 섹션으로 구성되어 있습니다.

- 『설치 및 사용법』
- 245 페이지의 『Business Object Designer에서 XML ODA 사용』
- 254 페이지의 『생성된 Business Object 정의의 내용』
- 255 페이지의 『Business Object 정의에서 정보 수정』

설치 및 사용법

이 섹션에서는 다음을 논의합니다.

- 『XML ODA 설치』
- 242 페이지의 『XML ODA를 사용하기 전에』
- 243 페이지의 『XML ODA 실행』
- 243 페이지의 『XML ODA의 여러 인스턴스 실행』
- 244 페이지의 『오류 및 추적 메시지 파일에 대한 작업』

XML ODA 설치

XML ODA를 설치하려면 IBM WebSphere 설치 프로그램을 사용하십시오. IBM WebSphere Business Integration Adapters에 대한 설치 프로그램을 사용하여 이 ODA를 설치하는 방법에 대한 지시사항은 *Implementing Adapters with WebSphere MQ Integrator Broker* 또는 *Implementing Adapters with WebSphere Application Server*를 참조하십시오. 이 ODA를 InterChange Server 설치 프로그램으로 설치하는 방법에 대한 지시사항은 *Windows용 시스템 설치 안내서* 또는 *UNIX용 시스템 설치 안내서*를 참조하십시오.

설치가 완료되면, 다음 파일이 시스템의 제품 디렉토리에 설치됩니다.

- `ODA\XML\XMLODA.jar`
- `ODA\messages\XMLODAAgent.txt`

- ODA\messages\XMLODAAgent_*ll_TT*.txt files(언어(*ll*) 및 국가나 지역(*TT*) 특정 메시지 파일)
- ODA\XML\start_XMLODA.bat(Windows만)
- ODA/XML/start_XMLODA.sh(UNIX만)

주: 달리 표시한 경우를 제외하고 이 책에서는 백슬래시(\)가 디렉토리 경로의 규칙으로 사용됩니다. UNIX 설치의 경우, 백슬래시를 슬래시(/)로 대체합니다. 모든 제품 경로 이름은 제품이 시스템에 설치된 디렉토리에서 상대적입니다.

XML ODA를 사용하기 전에

XML ODA를 실행하기 전에 시스템에 XML ODA용 필수 파일이 있는지 확인하십시오. 특히, ODA 환경 파일이 제품 디렉토리의 bin 서브디렉토리에 설치되었는지 확인하십시오.

UNIX

ODA 환경 파일 CWODAEnv.sh가 *ProductDir*/bin 디렉토리에 설치되어 있는지 확인하십시오.

Windows

ODA 환경 파일 CWODAEnv.bat가 *ProductDir*\bin 디렉토리에 설치되어 있는지 확인하십시오.

또한 ODA를 실행하는 스크립트 또는 일괄처리 파일에 변수가 제대로 설정되어 있는지 확인하십시오. 편집하기 위해 쉘(start_XMLODA.sh) 또는 일괄처리(start_XMLODA.bat) 파일을 열고 표 83에 설명된 값을 확인하십시오.

표 83. 쉘 및 일괄처리 파일 구성 변수

변수	설명	예
set AGENTNAME	ODA의 이름	set AGENTNAME=XMLODA
set AGENT	ODA jar 파일의 이름	**UNIX:** set AGENT = ${*ProductDir*}/ODA/XML/XMLODA.jar **WINDOWS:** set AGENT = *%ProductDir%*\ODA\XML\XMLODA.jar
set AGENTCLASS	ODA Java 클래스의 이름	set AGENTCLASS=com.crossworlds.oda.xml.XMLAgent

XML ODA를 설치하고 쉘 또는 일괄처리 파일에 구성 값을 설정한 후(표 83 참조), 다음을 수행하여 Business Object를 생성해야 합니다.

1. XML ODA를 실행합니다.
2. Business Object Designer를 실행합니다.
3. Business Object Designer에서 제공하는 GUI 인터페이스, Business Object 마법사의 6단계 프로세스에 따라 ODA를 구성 및 실행하십시오.

다음 섹션은 이들 단계에 대해 자세히 설명합니다.

XML ODA 실행

사용 중인 운영 체제에 적합한 시작 스크립트로 XML ODA를 실행할 수 있습니다.

UNIX

```
start_XMLODA.sh
```

Windows

```
start_XMLODA.bat
```

주: Windows 설치 프로그램은 설치하는 ODA를 시작하는 바로 가기를 제공합니다. 이 설치 프로그램을 사용하여 XML ODA를 설치한 경우, 프로그램 > IBM WebSphere Business Integration Adapters > 어댑터 > Object Discovery Agent 메뉴에서 시작 바로 가기를 찾을 수 있습니다.

Business Object Designer를 사용하여 XML ODA를 구성 및 실행합니다. Business Object Designer에서 시작하는 Business Object 마법사는 각 스크립트 또는 일괄처리 파일의 AGENTNAME 변수에 지정되어 있는 이름으로 각 ODA를 찾습니다. 이 커넥터의 기본 ODA 이름은 XMLODA입니다.

XML ODA의 여러 인스턴스 실행

XML ODA의 여러 인스턴스를 로컬 호스트 시스템 또는 원격 호스트 시스템에서 실행할 수 있습니다. 각 인스턴스는 고유 포트에서 실행됩니다. 이 포트 번호를 Business Object Designer 내에서 ODA를 실행할 때 함께 지정할 수 있습니다. 246 페이지의 그림 45에서는 실행할 ODA를 선택하는 Business Object Designer의 창을 보여줍니다.

오류 및 추적 메시지 파일에 대한 작업

오류 및 추적 메시지 파일(기본값은 XMLODAAgent.txt)은 제품 디렉토리 아래의 \ODA\messages\ 서브디렉토리에 있습니다. 이들 파일은 다음 이름 지정 규칙을 사용합니다.

*AgentName*Agent.txt

ODA 스크립트 또는 일괄처리 파일의 복수 인스턴스를 작성하고 표시된 각 ODA에 고유 이름을 제공하는 경우, 각 ODA 인스턴스에 대한 메시지 파일을 가질 수 있습니다. 또는 서로 다르게 이름 지정된 파일이 동일한 메시지 파일을 사용하도록 할 수 있습니다. 유효한 메시지 파일을 지정하는 두 가지 방법이 있습니다.

- ODA의 이름을 변경하고 이에 대한 메시지 파일을 작성하지 않는 경우, Business Object Designer에서 ODA 구성의 일부로 메시지 파일의 이름을 변경해야 합니다. Business Object Designer가 메시지 파일에 대한 이름을 제공하지만 실제로 파일을 작성하지는 않습니다. ODA 구성의 일부로 표시된 파일이 존재하지 않는 경우, 기존 파일을 가리키도록 값을 변경하십시오.
- 특정 ODA에 대한 기존 메시지 파일을 복사해서 필요한 대로 이를 수정할 수 있습니다. Business Object Designer는 각 파일의 이름을 이름 지정 규칙에 따라 지정한다고 가정합니다. 예를 들어, AGENTNAME 변수가 XMLODA1을 지정하는 경우, 도구는 연관된 메시지 파일의 이름이 XMLODA1Agent.txt라고 가정합니다. 따라서 Business Object Designer가 ODA 구성의 일부로 검증을 위해 파일 이름을 제공할 때, 파일 이름은 ODA 이름에 따라 달라집니다. 기본 메시지 파일의 이름이 올바르게 지정되었는지 확인하고 필요한 대로 이를 수정하십시오.

중요: ODA를 구성할 때 메시지 파일의 이름을 올바르게 지정하는 데 실패하면 메시지 없이 실행됩니다. 메시지 파일 이름 지정에 대한 자세한 정보는 247 페이지의 표 85를 참조하십시오.

구성 프로세스 중 다음을 지정합니다.

- XML ODA가 오류 및 추적 정보를 기록하는 파일의 이름
- 0 ~ 5 범위의 추적 레벨

표 84에서는 이들 값에 대해 설명합니다.

표 84. 추적 레벨

추적 레벨	설명
0	모든 오류 로그
1	메소드에 대한 모든 입력 및 종료 메시지 추적
2	ODA의 등록 정보 및 해당 값 추적
3	모든 Business Object의 이름 추적
4	모든 하위 스레드의 세부사항 추적

표 84. 추적 레벨 (계속)

5	• 모든 ODA 등록 정보의 초기화 값 표시 • XML ODA의 각 하위 스레드의 세부 상태 추적 • Business Object 정의 덤프 추적

이 값의 구성 위치에 대한 정보는 247 페이지의 표 85를 참조하십시오.

Business Object Designer에서 XML ODA 사용

이 섹션에서는 XML ODA를 사용하여 Business Object 정의를 생성하기 위해 Business Object Designer를 사용하는 방법에 대해 설명합니다. Business Object Designer 실행에 대한 정보는 *Business Object Development Guide*를 참조하십시오. Business Object Designer는 이들 각 단계를 안내하는 Business Object 마법사를 제공합니다. ODA를 실행한 후 Business Object Designer를 실행하여 Business Object 마법사로 ODA를 구성하고 실행해야 합니다. Business Object Designer에서 ODA를 사용하여 Business Object 정의를 생성하는 과정은 6단계로 구성됩니다.

ODA를 시작한 후, 다음을 수행하여 마법사를 시작하십시오.

1. Business Object Designer를 엽니다.
2. 파일 메뉴에서, ODA를 사용하여 새로 작성... 서브메뉴를 선택합니다.

   Business Object 마법사가 에이전트 선택이라는 이름의 첫 번째 창을 표시합니다. 246 페이지의 그림 45에서는 이 창을 보여줍니다.

ODA를 선택, 구성 및 실행하려면 다음 단계를 수행하십시오.

1. 『ODA 선택』
2. 246 페이지의 『구성 등록 정보 지정』
3. 249 페이지의 『노드 펼치기 및 XML 요소 선택』
4. 250 페이지의 『오브젝트 선택 확인』
5. 250 페이지의 『Business Object 정의 생성』 및 선택적으로 251 페이지의 『추가 정보 제공』
6. 253 페이지의 『Business Object 정의 저장』

ODA 선택

246 페이지의 그림 45에서는 6단계 Business Object 마법사의 첫 번째 대화 상자를 보여줍니다. 이 창에서, 실행할 ODA를 선택하십시오.

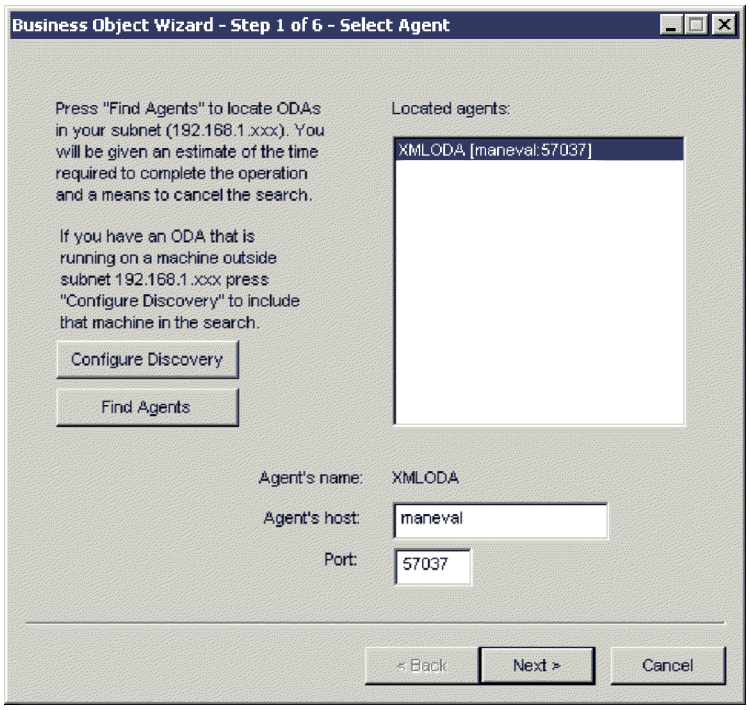

그림 45. ODA 선택

ODA를 선택하려면 다음을 수행하십시오.

1. 에이전트 찾기 단추를 눌러 찾은 에이전트 필드에서 등록된 모든 ODA 또는 현재 실행 중인 ODA를 표시하십시오. 그렇지 않으면, 해당 호스트 이름 및 포트 번호를 사용하여 ODA를 찾을 수 있습니다.

   주: Business Object 마법사가 원하는 ODA를 찾지 못하는 경우, ODA의 설정을 확인하십시오.

2. 표시된 목록에서 원하는 ODA를 선택하십시오.

   Business Object 마법사가 에이전트의 이름 필드에 선택사항을 표시합니다.

구성 등록 정보 지정

Business Object 마법사가 XML ODA와 처음 통신하면, 247 페이지의 그림 46에 표시된 대로 한 세트의 ODA 구성 등록 정보를 입력하도록 프롬프트를 표시합니다.

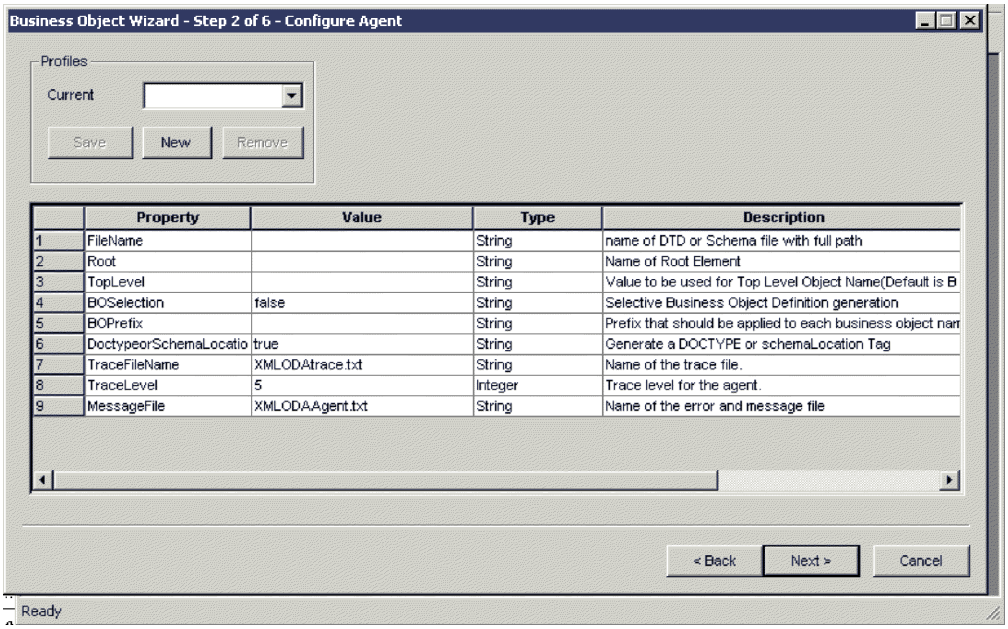

그림 46. ODA 구성 등록 정보 지정

표 85에 설명된 XML ODA 등록 정보를 구성하십시오.

표 85. XML ODA 구성 등록 정보

행 번호	등록 정보 이름	등록 정보 유형	설명
1	`FileName`	String	DTD 또는 스키마 문서의 전체 경로 이름. DTD 파일은 `.dtd` 확장자를 가져야 합니다. 스키마 문서 파일은 `.xsd` 확장자를 가져야 합니다.
2	`Root`	String	루트 요소로 처리될 XML 요소의 이름. 루트 요소가 지정되지 않으면, XML ODA가 다음을 가정합니다. • ODA의 DTD의 구문을 분석하면, 첫 번째 XML 요소를 루트로 취급합니다. • ODA의 스키마 문서의 구문을 분석하면, 첫 번째 글로벌 요소를 루트로 취급합니다.
3	`TopLevel`	String	ODA가 생성하는 최상위 레벨 Business Object에 사용될 이름. ODA는 최상위 레벨 Business Object에 business-object 접두부(`BOPrefix` 등록 정보로 지정)를 추가합니다. 이 접두부는 밑줄(_)로 구분됩니다. 최상위 레벨 이름을 지정하지 않으면, ODA가 최상위 레벨 Business Object의 이름으로서 *`BOPrefix_Root`*(여기서 *BOPrefix* 및 *`Root`*는 `BOPrefix` 및 Root 등록 정보의 값임)를 지정합니다.

표 85. XML ODA 구성 등록 정보 (계속)

행 번호	등록 정보 이름	등록 정보 유형	설명
4	BOSelection	String	XML ODA가 Business Object 정의가 사용될 요소의 이름을 선택할 수 있게 할지를 표시하는 부울 값(true 또는 false) • 이 등록 정보가 false로 설정되면, XML ODA는 사용자가 루트만 요소로서 선택할 수 있게 하고 루트와 하위 요소 모두를 위한 Business Object 정의를 생성합니다. • 이 등록 정보가 true로 설정되면, XML ODA는 사용자가 아무 요소나 선택할 수 있게 하고 선택한 요소에 대해서만 Business Object 정의를 생성합니다. 기본값은 false입니다.
5	BOPrefix	String	ODA가 XML 문서의 각 Business Object 정의의 이름에 적용하는 접두부. Business Object 접두부를 지정하지 않으면, ODA가 Business Object 정의의 이름 앞에 아무런 문자열도 붙이지 *않습니다*
6	DoctypeorSchemaLocation	String	XML ODA가 다음에 대한 속성을 생성해야 하는지를 표시하는 부울 값(true 또는 false). • DTD를 처리할 때: DOCTYPE 태그 • 스키마 문서를 처리할 때: schemaLocation 및 xsi 속성(XML 스키마 인스턴스 이름 공간의) 기본값은 true입니다.
7	TraceFileName	String	XML ODA가 추적 정보를 쓰는 파일의 전체 경로 이름. 파일이 존재하지 않는 경우, XML ODA가 지정된 디렉토리에 이를 작성합니다. 파일이 이미 존재하는 경우, XML ODA가 추가합니다. 기본적으로, XML ODA는 제품 디렉토리의 ODA\XML 서브디렉토리에 XMLODAtrace.txt라는 추적 파일을 작성합니다. 이 등록 정보를 사용하여 추적 파일에 다른 이름을 지정하십시오.
8	TraceLevel	Integer	XML ODA에 사용 가능한 추적 레벨. 유효한 값은 0 - 5입니다. 등록 정보의 기본값은 5입니다(전체 추적 사용 가능). 자세한 정보는 244 페이지의 『오류 및 추적 메시지 파일에 대한 작업』을 참조하십시오.
9	MessageFile	String	오류 및 메시지 파일의 전체 경로 이름. 기본적으로 XML ODA는 XMLODAAgent.txt라는 메시지 및 오류 파일을 작성합니다. **중요**: 오류 및 메시지 파일은 *반드시* 제품 디렉토리의 ODA\messages 서브디렉토리에 있어야 합니다. 이 등록 정보를 사용하여 기존 파일을 검증하거나 지정하십시오.

중요: Business Object Designer에 표시된 기본값이 존재하지 않는 파일을 나타내는 경우, 메시지 파일의 이름을 정정하십시오. 이 대화 상자에서 이동할 때 이름이 올바르지 않은 경우, Business Object Designer는 ODA가 실행될 창에 오류 메시지를 표시합니다. 이 메시지는 Business Object Designer에서 팝업되지 않습니다. 유효한 메시지 파일을 지정하는 데 실패하면 ODA가 메시지 없이 실행됩니다.

XML ODA를 사용할 때마다 다시 입력하지 않아도 되게 이름 지정된 프로파일에 이들 등록 정보를 저장할 수 있습니다. ODA 프로파일 지정에 대한 정보는 *Business Object Development Guide*를 참조하십시오.

노드 펼치기 및 XML 요소 선택

Business Object Designer는 이전 단계에서 구성한 등록 정보를 사용하여 도구를 지정된 XML 스키마(DTD 또는 스키마 문서)에 연결합니다. 연결한 후, Business Object Designer는 노드가 XML 스키마에 정의된 모든 XML 요소를 나타내는 트리를 표시합니다.

전체 계층 구조를 표시하기 위해 최상위 레벨 XML 요소를 펼칠 수 있습니다. XML 요소마다 XML ODA가 하위 Business Object 정의를 작성합니다.

그림 47에서는 일부 XML 요소가 펼쳐진 이 대화 상자를 보여줍니다.

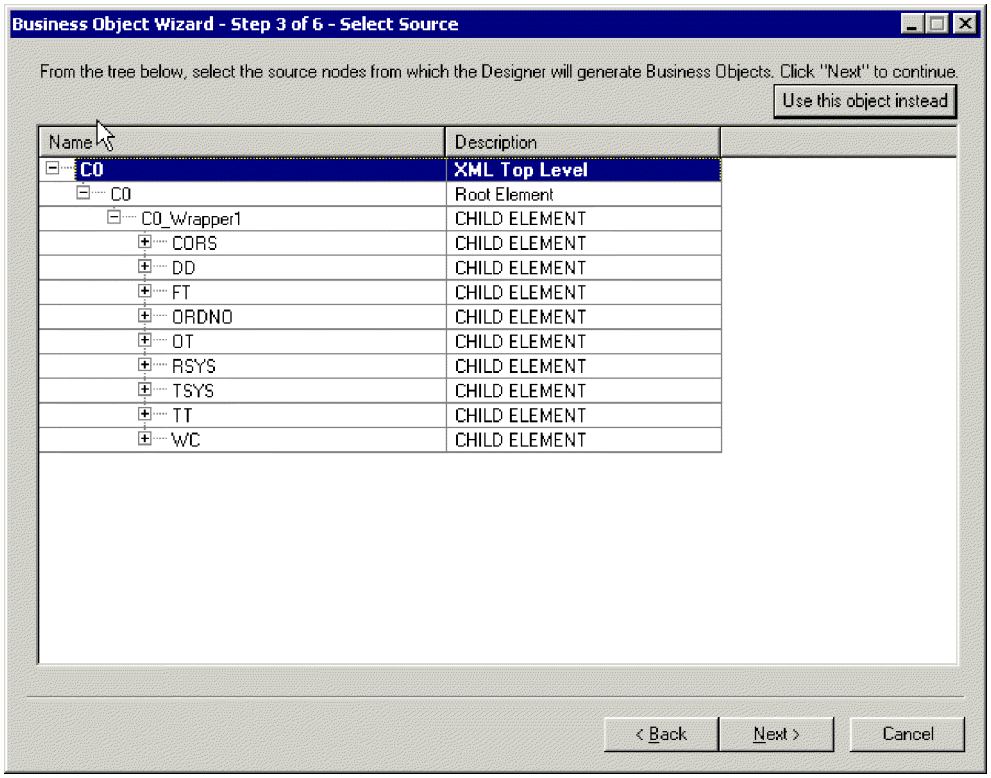

그림 47. 노드가 펼쳐진 XML 요소의 트리

필요한 모든 XML 요소를 선택하고 다음을 누르십시오.

오브젝트 선택 확인

생성된 Business Object 정의와 연관시킬 모든 XML 요소를 식별한 후, Business Object Designer는 선택된 오브젝트만 있는 대화 상자를 표시합니다. 그림 48에서는 이 대화 상자를 보여줍니다.

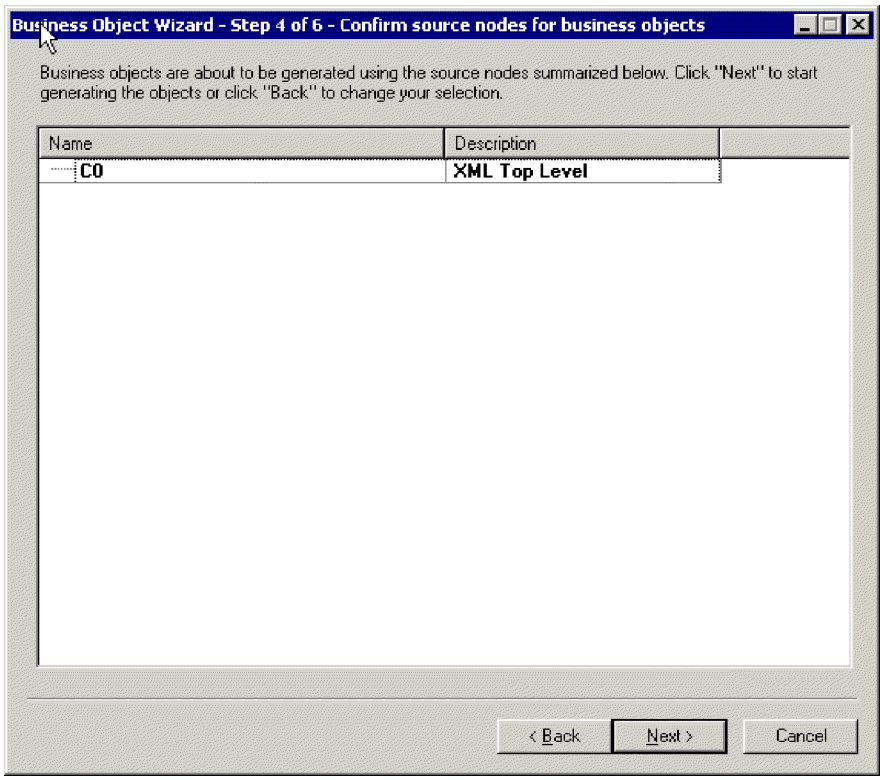

그림 48. 오브젝트 선택 확인

이 창은 다음 옵션을 제공합니다.

- 선택사항을 확인하려면 다음을 누르십시오.
- 선택사항이 올바르지 않은 경우, 이전을 눌러 이전 창으로 리턴하고 필요한 변경사항을 수행하십시오. 선택사항이 맞으면, 다음을 누르십시오.

Business Object 정의 생성

XML 요소를 확인하고 나면, 다음 대화 상자에서는 Business Object Designer가 Business Object 정의를 생성하고 있다는 것을 알려줍니다. 많은 양의 구성요소 인터페이스가 선택되면, 이 생성 단계가 시간이 걸릴 수 있습니다.

251 페이지의 그림 49에서는 이 대화 상자를 보여줍니다.

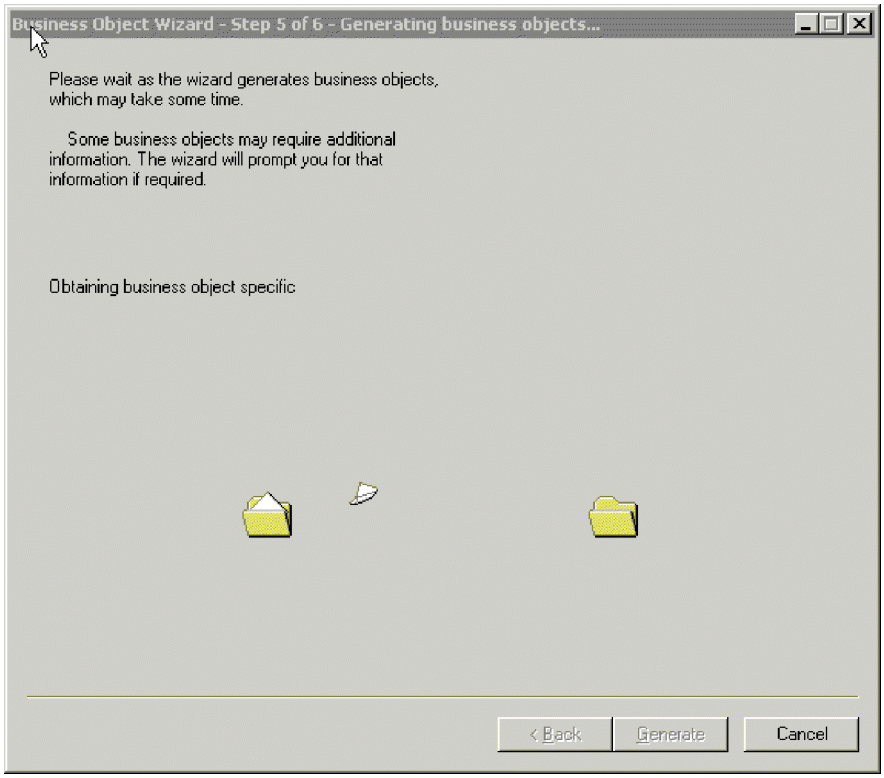

그림 49. Business Object 정의 생성

XML ODA는 다음 정보에서 Business Object 정의의 이름을 생성합니다.

- `BOPrefix` ODA 구성 등록 정보 값
- `TopLevel` ODA 구성 등록 정보 값
- Business Object 정의가 표시하는 XML 요소의 이름

밑줄(_) 문자로 이들 값 각각을 구분합니다. 따라서 생성되는 이름은 다음과 같은 형식을 갖게 됩니다.

`BOPrefix_TopLevel_XMLelement`

추가 정보 제공

XML ODA가 verb에 대한 추가 정보를 필요로 하기 때문에 Business Object Designer가 정보를 넣도록 프롬프트하는 BO 등록 정보 창을 표시합니다. 252 페이지의 그림 50에서는 이 대화 상자를 보여줍니다.

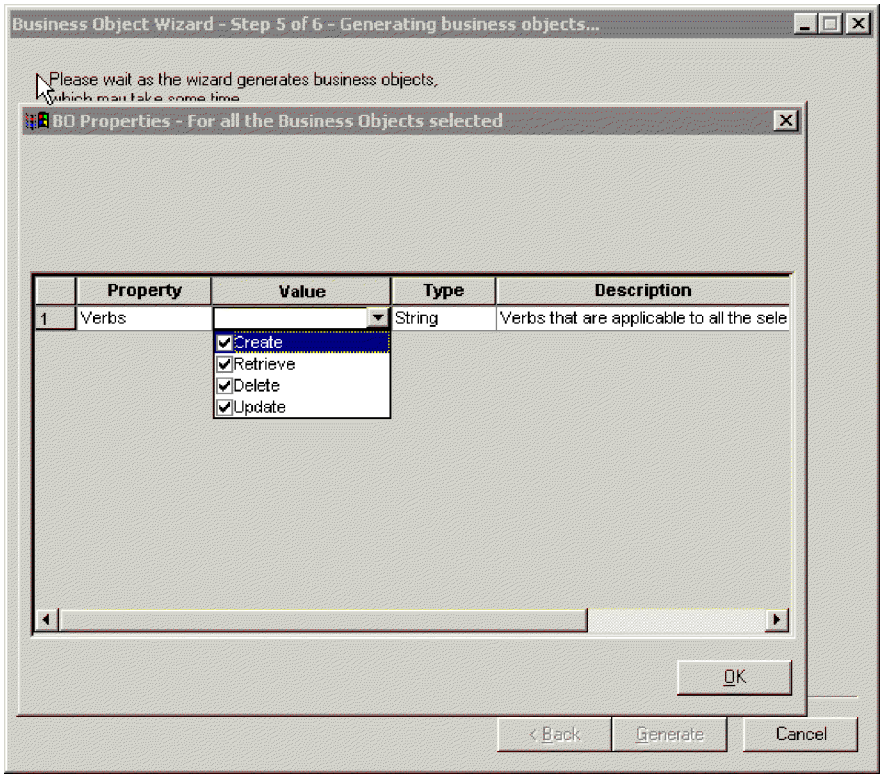

그림 50. 추가 정보 제공 - verb

BO 등록 정보 창에서 verb 정보를 입력 또는 변경하십시오. 값 필드를 누르고 팝업 메뉴에서 하나 이상의 verb를 선택하십시오. 이들은 Business Object가 지원하는 verb입니다.

주: BO 등록 정보 대화 상자의 필드에 복수 값이 있는 경우, 대화 상자가 처음 표시될 때 필드가 비어 있는 것처럼 보입니다. 필드에서 눌러 값의 드롭 다운 목록을 표시하십시오.

XML 문서에 `anyAttribute` 요소를 포함하는 스키마가 있으면 그림 51에서처럼 XML ODA는 추가 BO 등록 정보 창을 표시합니다.

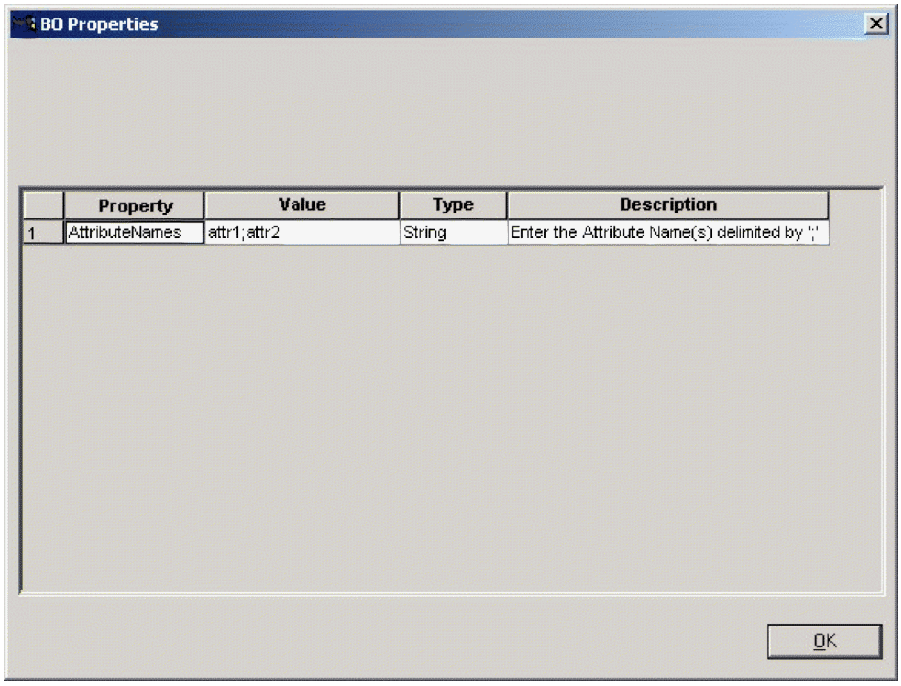

그림 51. 추가 정보 제공 - 속성 이름

이 BO BO 등록 정보 창에서 XML ODA가 작성하기를 원하는 Business Object 속성의 이름을 입력하십시오. 각 속성을 세미콜론(;)으로 구분하십시오. `anyAttribute`에 대한 자세한 정보는 93 페이지의 『지원되는 스키마 문서 구조』를 참조하십시오.

Business Object 정의 저장

BO 등록 정보 대화 상자에 모든 필수 정보를 제공하고 확인을 누르면, Business Object Designer가 마법사의 마지막 대화 상자를 표시합니다. 이 대화 상자에서 다음 조치를 수행할 수 있습니다.

- Business Object를 서버에 저장(InterChange Server가 통합 브로커인 경우)
- Business Object 정의를 파일에 저장(모든 통합 브로커에 대해)
- Business Object Designer에서 편집하기 위해 Business Object 정의 열기

자세한 정보 및 추가 수정사항은 *Business Object Development Guide*를 참조하십시오.

254 페이지의 그림 52에서는 이 대화 상자를 보여줍니다.

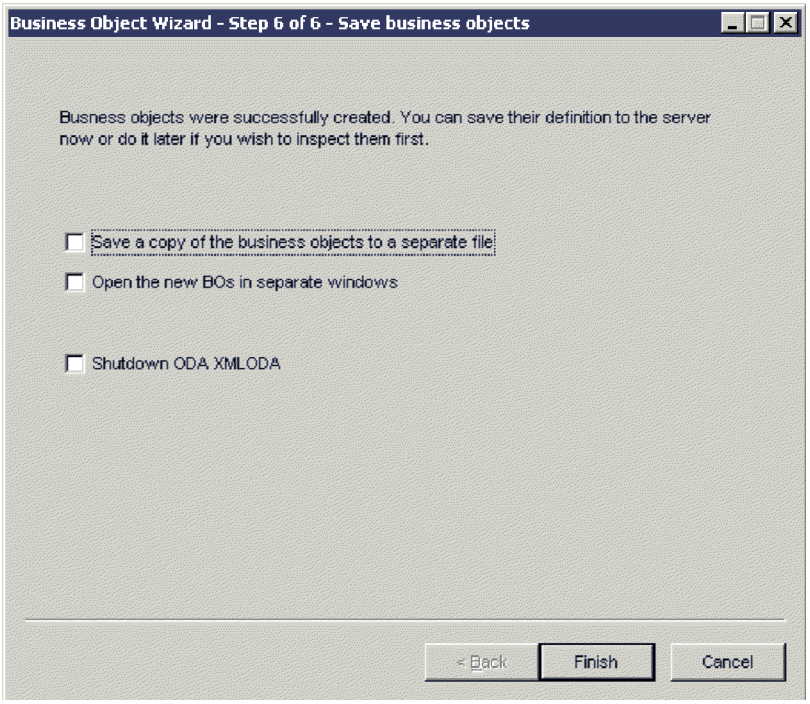

그림 52. Business Object 정의 저장

생성된 Business Object 정의의 내용

다음 섹션에 설명된 것처럼 XML ODA가 속성, verb, 응용프로그램 특정 정보를 생성하는 Business Object를 정의합니다.

Business Object 정의 구성요소	**자세한 정보**
속성	42 페이지의 『Business Object 구조』 43 페이지의 『Business Object 속성 등록 정보』
응용프로그램 특정 정보	46 페이지의 『응용프로그램 특정 정보』
Verb	47 페이지의 『Business Object verb』

주: XML ODA의 이전 버전은 ObjectEventId 속성을 키로 지정하는 상위 Business Object 정의를 생성합니다. Business Object Designer는 더 이상 Business Object 정의에서 ObjectEventId를 키 속성으로 지정하게 하지 않습니다. 그러므로 XML ODA는 이제 이러한 작동을 피하는 특수 단계를 수행합니다. 이 새로운 작동에서는 키 속성을 지정하기 위해 생성된 Business Object 정의를 수정하는 것이 필요합니다. 자세한 정보는 44 페이지의 『Key 및 Foreign Key 속성 등록 정보』를 참조하십시오.

Business Object 정의에서 정보 수정

XML ODA가 작성하는 Business Object 정의에서 정보를 수정하는 것이 필요할 수 있습니다. 예를 들면, 수동으로 원하지 않는 속성을 제거하고 속성 응용프로그램 특정 정보의 필수 태그를 추가해야 합니다. Business Object 정의를 검토하거나 수정하기 위해 Business Object Designer 또는 문서 편집기를 사용할 수 있습니다. 수정된 정의를 저장소로 재로드하기 위해 Business Object Designer를 사용할 수 있습니다. 그렇지 않으면, InterChange Server(ICS)가 통합 브로커인 경우, `repos_copy` 명령을 사용하여 정의를 저장소로 로드할 수 있습니다. WebSphere MQ Integrator Broker가 통합 브로커인 경우, 시스템 명령을 사용하여 파일을 저장소 디렉토리에 복사할 수 있습니다.

주의사항

IBM은 다른 국가에서는 이 책에 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 재산권을 침해하지 않는 한, 기능상 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 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 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

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 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 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 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (2) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이센스 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 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 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 것이라고는 보증할 수 없습니다. 또한, 일부 성능은 추정치일 수도 있으므로 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 기타 주장에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 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.

IBM의 향후 방향 또는 의도에 관한 모든 언급은 별도의 통지없이 변경될 수 있습니다.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보(제공될 경우)는 이 프로그램을 사용하여 응용프로그램 소프트웨어를 작성하는 것을 돕기 위한 것입니다.

범용 프로그래밍 인터페이스를 사용하면 이 프로그램 도구 서비스를 확보하는 응용프로그램 소프트웨어를 작성할 수 있습니다.

그러나 이 정보는 도한 진단, 수정 및 성능 조정에 대한 정보를 포함할 수 있습니다. 진단, 수정 및 성능 조정에 대한 정보는 사용자의 응용프로그램 소프트웨어를 디버그하는 데 도움을 주기 위해 제공됩니다.

경고: 진단, 수정 및 성능 조정에 대한 정보는 변경될 수 있으므로 프로그래밍 인터페이스로 사용해서는 안됩니다.

상표 및 서비스표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표 또는 등록상표입니다.

IBM
IBM 로고
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

MMX, Pentium 및 ProShare는 미국 또는 기타 국가에서 사용되는 Intel Corporation의 상표 또는 등록상표입니다.

Java 및 모든 Java 기반의 상표는 Sun Microsystems, Inc.의 상표 또는 등록상표입니다.

기타 회사, 제품 또는 서비스 이름은 해당 회사의 상표 또는 서비스표입니다. IBM WebSphere InterChange Server V4.2.2, IBM WebSphere Business Integration Toolset V4.2.2, IBM WebSphere Business Integration Collaborations V4.2, IBM WebSphere Business Integration Adapter Framework V2.4.0

색인

[가]

개발 프로세스 195, 198

[나]

네임 핸들러 215, 231
  EDI Data Handler 및 106, 129, 132
  Request-Response Data Handler 및 135, 151
  XML Data Handler 및 39, 101, 102

[다]

데이터 변환
  문자열로 236
  바이트 배열로 232
  스트림으로 235
  Business Object로 9, 13, 202, 227, 229
  Business Object에서 8, 12, 207, 232, 235, 236

[라]

로케일 222, 233, 238

[마]

문서 유형 정의(DTD) 51, 66
  경로 48
  랩퍼 Business Object 55
  루트 요소 39, 42, 51, 96, 97, 247
  샘플 52
  속성 60, 62, 99
  속성 등록 정보 및 43, 53
  외부 65
  요소 59, 62, 99
  위치 247
  이름 공간 65
  일반 Business Object 54
  조건부 섹션 65
  주석 63, 99
  지원되는 구조 65, 93
  처리 명령어 64, 99
  필수 Business Object 정의 42, 51
  혼합 Business Object 55
문서 유형 정의(DTD) *(계속)*
  ANY 지시문 65
  ATTLIST 단편 54, 76
  Business Object 정의 요구사항 51
  Business Object 정의 작성 64, 95, 245, 254
  Business Object 정의로 변환 64
  Business Object의 구조 42, 51
  CDATA 절 61, 63, 99
  DOCTYPE 선언 51, 63, 65, 99, 102
  ELEMENT 단편 53
  Entity Resolver 40, 48
  FIXED 속성 52, 102
  PCDATA 요소 55, 60, 61, 99
  verb 보존 47
문자 인코딩 222, 233, 237

[사]

서버 액세스 인터페이스
  최상위 레벨 Meta Object 29
  Data Handler 사용 11
  Data Handler 인스턴스화 21
  getBO() 및 227
  IcreateBusinessObjectFrom() 14, 22
  ItoExternalForm() 13, 22
속성
  단순 86, 87
  무시 208
  복합 85
스키마 문서 66, 95
  기본 이름 공간 83
  단순 요소 87
  단순 유형 87
  대상 이름 공간 77
  랩퍼 Business Object 72, 82
  루트 요소 39, 42, 66, 67, 96, 97, 247, 248
  모든 그룹 73, 94
  복합 유형 69, 70, 85, 87, 94
  샘플 66
  선택사항 그룹 72, 93
  속성 89, 90, 99
  속성 등록 정보 및 75
  순서 그룹 69, 70, 93
  스키마 요소 67

스키마 문서 *(계속)*
스키마 위치 40, 68, 91, 247, 248
양식 속성 86, 89
요소 54, 69, 85, 87, 90, 99
이름 공간 77
일반 Business Object 69
주석 91, 99
처리 명령어 91, 99
필수 Business Object 정의 42, 67
혼합 Business Object 70
any 요소 93
anyAttribute 요소 93, 252
attributeFormDefault 83, 90
Business Object 정의 요구사항 66
Business Object 정의 작성 93, 95, 245, 254
Business Object의 구조 66
elementFormDefault 83, 86
Entity Resolver 40
import 요소 79, 94
include 요소 94
occurrence indicator 76
targetNamespace 77
use 속성 76
xmlns 속성 82

[아]

액세스 클라이언트 4, 11
최상위 레벨 Meta Object 29
응용프로그램 특정 정보
속성 58, 84, 98
크기 한계 46
Business Object에 대한 54, 76
Delimited Data Handler 및 167
FixedWidth Data Handler 및 157
NameValue Data Handler 및 177
XML Data Handler 및 54
이스케이프 문자 165, 170

[차]

최상위 레벨 Meta Object 6, 15, 20, 219, 237
추적 메시지 240

[카]

커넥터 3
구성 33, 221
커넥터 *(계속)*
최상위 레벨 Meta Object 29, 31
Data Handler 사용 7
Data Handler 인스턴스화 20

[파]

필수 속성 등록 정보
XML Data Handler 및 53

[하]

하위 Meta Object 6, 15, 32, 218
Delimited Data Handler 164
EDI Data Handler 109
FixedWidth Data Handler 155
NameValue Data Handler 174
Request-Response Data Handler 146
XML Data Handler 47

A

ADK(Adapter Development Kit) 198
Alignment Data Handler 구성 등록 정보 155
Application-specific information
EDI Data Handler 및 112
Request-Response Data Handler 및 143
Attribute 등록 정보
이름 43, 143, 144
카디널리티 53, 75, 76, 116, 157, 166, 176
필수 53, 116
Delimited Data Handler 및 166
FixedWidth Data Handler 및 157
Foreign Key 44, 45, 54, 76, 116
Key 44, 45, 54, 76, 116
MaxLength 116, 157
Name 112, 113, 114, 115, 116, 117, 118, 157, 166, 176
NameValue Data Handler 및 175
Required 45, 76
Type 43, 116, 157, 166, 176
XML Data Handler 및 43, 53, 75

B

Binary Host Data Handler 183
BOCountSize Data Handler 구성 등록 정보 155
BONameSize Data Handler 구성 등록 정보 155, 160
BOPrefix Data Handler 구성 등록 정보 47, 147

BOPrefix Data Handler 구성 등록 정보 *(계속)*
createHandler() 및 19, 226
getBOName() 및 217, 231
Request-Response 네임 핸들러 및 136
XML 네임 핸들러 및 39, 101, 102
BOPrefix XML ODA 등록 정보 248, 251
BOSelection XML ODA 등록 정보 248
BOVerbSize Data Handler 구성 등록 정보 155, 160
Business Object
랩퍼 55, 71, 72, 82
로케일 223
리턴 227, 229
변환 8, 9, 12, 13, 202, 207, 228, 232, 235, 236
고정 폭 문자열 158, 159
구분된 데이터 168, 169
이름-값 쌍 178, 179
입력 형식 149
EDI 문서 119, 123, 150
XML 문서 98, 100
속성 무시 203, 208
요구사항
Delimited Data Handler 165
EDI Data Handler 111
FixedWidth Data Handler 156
NameValue Data Handler 175
Request-Response Data Handler 142
XML Data Handler 51, 66
이름 102
이름 가져오기 39, 106, 135, 230
일반 54, 69
작성 227
접두부 17, 226
채우기 10, 14, 227
혼합 55, 70
Data Handler 221
Business Object 정의 118
랩퍼 55, 71, 72, 82
루트 요소 42, 52, 68, 92, 97
상위 56, 71, 72
스키마 문서 및 66, 95
스키마 문서에서 93
스키마 문서의 요구사항 51, 66
이름 251
일반 54, 69
작성 95, 145, 245, 254
최상위 레벨
EDI Data Handler 112
Request-Response Data Handler 135, 142, 145
Business Object 정의 *(계속)*
최상위 레벨 *(계속)*
XML Data Handler 42, 51, 66, 67, 96, 98
혼합 55, 70
DTD 및 51, 66
DTD에서 64

C

Cardinality 속성 등록 정보
Delimited Data Handler 및 166
EDI Data Handler 및 116
FixedWidth Data Handler 및 157
NameValue Data Handler 및 176
XML Data Handler 및 53, 75, 76
ClassName Data Handler 구성 등록 정보 17, 18, 34, 219
Delimited Data Handler 165
EDI Data Handler 109
FixedWidth Data Handler 155
NameValue Data Handler 174
Request-Response Data Handler 147
XML Data Handler 47
COBOL
COBOL 레코드 183
copybook 184
COBOL copybook 190
copybook 184
Copybook, COBOL 190
createHandler() 메소드 15, 21, 23, 34, 225
서버 액세스 인터페이스에 의해 호출 13, 14
커넥터에서 호출 9, 10
클래스 이름 사용 16, 18, 201
클래스 찾기 16, 217
MIME 유형 16, 18, 200
CustDataHandler.jar 파일 6, 16, 18, 26, 217
Custom Data Handler 102, 195, 221
개발 프로세스 195
네임 핸들러 132, 151, 215
메소드 구현 201
설계 200
스텁 파일 사용 201
위치 6
필수 메소드 202
Business Object 설정 221
getBO() 예 202
getStreamFromBO() 예 214
getStringFromBO() 예 209
Jar 파일에 추가 217

Custom Data Handler *(계속)*
Meta Object 200, 218
CwDataHandler.jar 파일 5, 16, 18, 26, 217
CwEDIDataHandler.jar 파일 6, 16
CwXMLDataHandler.jar 파일 6, 16
CxBlank Data Handler 구성 등록 정보
Delimited Data Handler 165, 167, 170, 180
FixedWidth Data Handler 155, 160
NameValue Data Handler 174, 177
CxBlank 속성 값
Delimited Data Handler 165, 167, 170
FixedWidth Data Handler 155
NameValue Data Handler 174, 177, 180
XML Data Handler 46, 100
CxBlankValue Data Handler 구성 등록 정보(사용하지 않음) 174
CxIgnore Data Handler 구성 등록 정보
Delimited Data Handler 165, 166, 170
FixedWidth Data Handler 156, 160
NameValue Data Handler 174, 176, 180
CxIgnore 속성 값
Delimited Data Handler 166, 170
FixedWidth Data Handler 156, 160
NameValue Data Handler 176, 180
XML Data Handler 45, 100

D

Data Handler 3
개발 프로세스 195
개요 24
구성 18, 28, 35, 228, 232, 235, 236
구성 옵션 231, 234, 239
국제화된 221, 225
기본 5
기본 클래스 201, 225
다음에 대한 클래스 15, 201
로케일 223, 233, 238
문맥 7
문자 인코딩 233, 237
사용자 정의 6, 102, 132, 151, 195, 221
샘플 6, 198
서버 액세스 인터페이스 11
설치 25
속성 무시 203, 208
위치 5
인스턴스화 9, 10, 13, 14, 15, 23, 225
추적 240
커넥터 7
Data Handler *(계속)*
컴파일 스크립트 217
클래스 식별 15
특수 5
패키지 17, 201
호출 트리거 플로우에서 7
IBM 제공 5, 18
Meta Data 구동 23
Data Handler API 199
DataHandler 클래스 199, 225, 240
추상 메소드 202
패키지 225
확장 201
createHandler() 225
getBOName() 230
getBooleanOption() 231
getBO() (abstract) 227
getBO() (public) 228
getByteArrayFromBO() 232
getEncoding() 233
getLocale() 233
getOption() 234
getStreamFromBO() 235
getStringFromBO() 236
setConfigMOName() 237
setEncoding() 237
setLocale() 238
setOption() 239
traceWrite() 240
DataHandler 패키지 225
DefaultEscapeBehavior Data Handler 구성 등록 정보 48, 62
DefaultVerb Data Handler 구성 등록 정보
EDI Data Handler 109, 130
NameValue Data Handler 174
Delimited Data Handler 163, 173
개요 163
구성 164
기능 163
기존 Business Object 사용 167
문자열 요구사항 169
문자열에서 Business Object로 변환 169
분리문자 166
샘플 파일 199
예 문자열 170
응용프로그램 특정 정보 167
이스케이프 문자열 166
처리 164
하위 Meta Object 164

Delimited Data Handler *(계속)*
Business Object 구조 166
Business Object 속성 등록 정보 166
Business Object 요구사항 165
Business Object에서 문자열로 변환 168
ClassName 165
CxBlank 165
CxIgnore 165
Delimiter 165
DummyKey 165
Escape 165
OmitObjectEventId 165
Delimiter Data Handler 구성 등록 정보 164, 165, 166, 169, 170
DoctypeorSchemaLocation XML ODA 등록 정보 51, 68, 93, 248
DTDPath Data Handler 구성 등록 정보 40, 48
DummyKey Data Handler 구성 등록 정보
Delimited Data Handler 165
EDI Data Handler 109
FixedWidth Data Handler 156
NameValue Data Handler 174
XML Data Handler 48

E

EDI Business Object
머리글 115
복합 117, 122
세그먼트 115, 122
세그먼트 루프 117
최상위 레벨 108, 112, 130
트레일러 118
EDI Data Handler 105, 132
개요 105
구성 107
네임 핸들러 106, 129, 132
네임 핸들러 찾아보기 파일 106, 107, 129
문자열에서 Business Object로 변환 123, 150
사용자 정의 132
위치 6
처리 107
하위 Meta Object 109
Business Object 구조 111
Business Object 요구사항 111
Business Object에서 EDI 문서로 변환 119
ClassName 109
DefaultVerb 109
DummyKey 109
ISA 109
EDI Data Handler *(계속)*
NameHandlerClass 109
NameHandlerFile 109
Reader 오브젝트 제한사항 123, 228
RELEASE_CHAR 109
SEPARATOR_COMPOSIT 110
SEPARATOR_ELEMENT 109
SEPARATOR_REPEAT 110
SEPARATOR_SEGMENT 110
UNA 109
UNB 109
EDI 네임 핸들러 찾아보기 파일 106, 107, 129
EDI 문서
구문 분석 131
반복 분리문자 110, 120, 124, 125, 126
복합 분리문자 110, 120, 124, 125, 126, 128
세그먼트 분리문자 110, 120, 124, 125
요소 분리문자 109, 120, 124, 125
이스케이프 문자 109, 122
트랜잭션 ID 107, 126, 127, 129
Business Object로 변환 118
DUNS 번호 107, 126, 128, 129
EDK(E-Business Development Kit) 201
Entity Resolver 40, 101, 104
EntityResolver Data Handler 구성 등록 정보 40, 48, 101, 104
Escape Data Handler 구성 등록 정보 164, 165, 170
Escape 문자열 166

F

FileName XML ODA 등록 정보 94, 247
FixedWidth Data Handler 153, 163
개요 153
구성 155
기능 154
기존 Business Object 사용 157
맞추기 값 154
문자열 요구사항 159
문자열에서 Business Object로 변환 159
샘플 파일 199
응용프로그램 특정 정보 157
자르기 159
채움 문자 154
처리 154
하위 Meta Object 155
Alignment 155
BOCountSize 155
BONameSize 155

FixedWidth Data Handler *(계속)*
BOVerbSize 155
Business Object 구조 157
Business Object 속성 등록 정보 157
Business Object 요구사항 156
Business Object에서 문자열로 변환 158
ClassName 155
CxBlank 155
CxIgnore 156
DummyKey 156
Max Length 속성 등록 정보 154, 164
OmitObjectEventId 156
PadCharacter 156
Truncation 156
Foreign Key 속성 등록 정보
EDI Data Handler 및 116
XML Data Handler 및 44, 45, 54, 76

G

getBOName() 메소드 215, 216, 223, 230
getBooleanOption() 메소드 215, 231
getBO() 메소드 10, 14, 123, 202
abstract 202, 227
public 215, 223, 228
getByteArrayFromBO() 메소드 202, 207, 232
getEncoding() 메소드 224, 233
getLocale() 메소드 223, 233
getOption() 메소드 215, 234
getStreamFromBO() 메소드 202, 207, 214, 235
getStringFromBO() 메소드 202, 207, 209, 236

I

IgnoreUndefinedAttributes Data Handler 구성 등록 정보 48
IgnoreUndefinedElements Data Handler 구성 등록 정보 48
InitialBufferSize Data Handler 구성 등록 정보 48
ISA Data Handler 구성 등록 정보 109, 125

J

JCDK(Java Connector Development Kit) 199, 201

K

Key 속성 등록 정보
EDI Data Handler 및 116
XML Data Handler 및 44, 45, 54, 76

M

makeDataHandler.bat 컴파일 스크립트 199, 217
make_datahandler 컴파일 스크립트 218
MaxLength 속성 등록 정보
Delimited Data Handler 및 164
EDI Data Handler 및 116
FixedWidth Data Handler 및 153, 154, 157, 160
MessageFile XML ODA 등록 정보 248
Meta Object 15, 28, 33
구조 6, 15
로드 220
사용 여부 200
사용자 정의 Data Handler 200, 218
서버 액세스 인터페이스에서 사용 29
설정 220
액세스 클라이언트에서 사용 29
이름 설정 20, 23, 237
작성 218
최상위 레벨 6, 15, 17, 18, 20, 29, 219, 237
커넥터 221
커넥터에서 사용 31
하위 6, 15, 17, 18, 32
MIME 유형 속성 6, 7, 28
MIME 유형 15, 21, 29, 226
부속 유형 17, 33, 49, 110
이름 제한사항 33, 219
edi 6, 105
text/delimited 5, 163
text/fixedwidth 5, 153
text/namevalue 5, 173
text/requestresponse 5, 135
text/xml 6, 37
MO_DataHandler_Default Meta Object 20, 31
MO_DataHandler_DefaultDelimitedConfig Meta Object 33, 164
MO_DataHandler_DefaultEDIConfig Meta Object 33, 109, 121
MO_DataHandler_DefaultFixedWidthConfig Meta Object 33, 155
MO_DataHandler_DefaultNameValueConfig Meta Object 33, 174
MO_DataHandler_DefaultRequestResponseConfig Meta Object 33, 147
MO_DataHandler_DefaultXMLConfig Meta Object 33, 47
MO_Server_DataHandler Meta Object 23, 29

N

Name 속성 등록 정보
Delimited Data Handler 및 166
EDI Data Handler 및 112, 113, 114, 115, 116, 117, 118

Name 속성 등록 정보 *(계속)*
FixedWidth Data Handler 및 157
NameValue Data Handler 및 176
Request-Response Data Handler 및 143, 144
XML Data Handler 및 43
NameHandler 클래스 215
NameHandlerClass Data Handler 구성 등록 정보 217, 231
EDI Data Handler 106, 109, 132
Request-Response Data Handler 136, 147, 151
XML Data Handler 39, 48, 102
NameHandlerFile Data Handler 구성 등록 정보 106, 108, 109, 129
NameValue Data Handler 173, 182
개요 173
구성 174
기존 Business Object 사용 177
문자열 요구사항 179
문자열에서 Business Object로 변환 179
샘플 파일 199
예 파일 180
응용프로그램 특정 정보 177
처리 174
하위 Meta Object 174
Business Object 구조 175
Business Object 속성 등록 정보 175
Business Object 요구사항 175
Business Object에서 문자열로 변환 178
ClassName 174
CxBlank 174
CxIgnore 174
DefaultVerb 174
DummyKey 174
SkipCxIgnore 175
ValidateAttrCount 175

O

ObjectEventId 속성 95, 156, 165, 175
OmitObjectEventId Data Handler 구성 등록 정보
Delimited Data Handler 165
FixedWidth Data Handler 156

P

PadCharacter Data Handler 구성 등록 정보 155, 156
Parser Data Handler 구성 등록 정보 39, 48, 101

R

Reader 오브젝트 123, 227
RELEASE_CHAR Data Handler 구성 등록 정보 109
Request Data Handler 133
RequestDataHandlerMimeType data-handler 구성 등록 정보 147, 150
Request-Response Data Handler 133, 152
개요 133
구성 146
구성요소 135
네임 핸들러 135, 151
사용자 정의 151
요청 Business Object 143, 145
응답 Business Object 144, 145
입력 형식으로 Business Object 변환 149
처리 141
최상위 레벨 Business Object 135, 142, 145
하위 Meta Object 146
BOPrefix 147
Business Object 구조 142
Business Object 요구사항 142
Business Object 정의 작성 145
ClassName 147
NameHandlerClass 147
RequestDataHandlerMimeType 147
ResponseDataHandlerMimeType 147
Required 속성 등록 정보
EDI Data Handler 및 116
XML Data Handler 및 45, 76
Requrest Data Handler 149
Response Data Handler 133, 150
ResponseDataHandlerMimeType data-handler 구성 등록 정보 147, 151
Root XML ODA 등록 정보 42, 247

S

SAX Parser 39, 48, 101
SEPARATOR_COMPOSIT Data Handler 구성 등록 정보 110, 121, 126
SEPARATOR_ELEMENT Data Handler 구성 등록 정보 109, 120, 121, 125
SEPARATOR_REPEAT Data Handler 구성 등록 정보 110, 121, 126
SEPARATOR_SEGMENT Data Handler 구성 등록 정보 110, 120, 121
setConfigMOName() 메소드 20, 237

setEncoding() 메소드 224, 237
setLocale() 메소드 223, 238
setOption() 메소드 215, 239
setupOptions() 메소드 18, 226, 228, 233, 235, 237
SkipCxIgnore Data Handler 구성 등록 정보 175, 177
StubDataHandler.java 파일 199, 201, 202

T

TopLevel XML ODA 등록 정보 247, 251
TraceFileName XML ODA 등록 정보 248
TraceLevel XML ODA 등록 정보 248
traceWrite() 메소드 215, 240
Truncation Data Handler 구성 등록 정보 154, 156, 159
Type 속성 등록 정보
  Delimited Data Handler 및 166
  EDI Data Handler 및 116
  FixedWidth Data Handler 및 157
  NameValue Data Handler 및 176
  XML Data Handler 및 43
TypeSubstitution XML ODA 등록 정보 94

U

UNA Data Handler 구성 등록 정보 109, 125
UNB Data Handler 구성 등록 정보 109, 125
UseNewLine Data Handler 구성 등록 정보 48

V

ValidateAttrCount Data Handler 구성 등록 정보 175, 178, 179, 181
Validation Data Handler 구성 등록 정보 39, 49
Verb(XML에 보존) 47, 100

X

XML Data Handler 37, 104
  개요 37
  구성 47
  구성요소 38, 106
  네임 핸들러 39, 101, 102, 231
  사용자 정의 102
  속성 레벨 응용프로그램 특정 정보 58, 84
  위치 6
  응용프로그램 특정 정보 54
    이스케이프 48, 62
    attr_fd 85, 89, 90
XML Data Handler *(계속)*
  응용프로그램 특정 정보 *(계속)*
    attr_name 59, 60, 85
    attr_ns 85
    cw_mo_label 98, 100, 101, 207
    elem_fd 85, 86
    elem_name 59, 85, 87
    elem_ns 85
    escape=true 59, 62, 85, 99
    notag 59, 61, 87, 88, 98, 99
    type=attribute 59, 60, 61, 85, 89, 99
    type=attr_name 89
    type=cdata 59, 61, 63, 99
    type=comment 59, 64, 85, 91, 99
    type=defaultNS 78
    type=doctype 59, 63, 99
    type=MIXED 55, 71
    type=pcdata 57, 59, 60, 61, 85, 87, 99
    type=pi 59, 64, 85, 91
    type=xmlns 78
    type=xsinoNSlocation 85, 92
    type=xsischemalocation 85, 92
    xsinoNSlocation 99
    xsinoschemalocation 99
  이스케이프 처리 48, 62, 99
  처리 37, 184
  하위 Meta Object 47
  BOPrefix 47
  Business Object 구조 42, 51, 66
  Business Object 변환 98
  Business Object 속성 등록 정보 43, 53, 75
  business-object 레벨 응용프로그램 특정 정보 54, 76
  ClassName 47
  DefaultEscapeBehavior 48
  DTDPath 48
  DummyKey 48
  Entity Resolver 40, 104
  EntityResolver 48
  IgnoreUndefinedAttributes 48
  IgnoreUndefinedElements 48
  InitialBufferSize 48
  NameHandlerClass 48
  Parser 48
  Reader 오브젝트 제한사항 228
  SAX Parser 39
  UseNewLine 48
  Validation 49
  verb 변환 47, 100

XML Data Handler *(계속)*
XML 문서 변환 100
XML Object Discovery Agent(ODA) 241, 255
구성 등록 정보 246
등록 정보 247
설치 241
실행 243
여러 인스턴스 243
파일 이름 94
BOPrefix 248
BOSelection 248
DoctypeorSchemaLocation 93, 248
FileName 247
MessageFile 248
Root 247
TopLevel 247
TraceFileName 248
TraceLevel 248
XML ODA(Object Discovery Agent)
TypeSubstitution 94
XML 문서 37
구문 분석 101
루트 요소 247
속성 42, 60, 89
외부 참조 40, 101
요구사항 100
이스케이프 처리 62, 90
주석 63, 91, 99
처리 명령어 64
프롤로그 63, 64, 65
Business Object 정의 작성 118
Business Object로 변환 100
CDATA 절 61, 63, 99
noNamespaceSchemaLocation 91, 99
schemaLocation 68, 91, 99
XML 스키마 이름 공간 77
XML 스키마 인스턴스 이름 공간 77, 91, 248

IBM 한글 지원에 관한 설문

IBM WebSphere Business Integration Adapters
IBM WebSphere InterChange Server
Data Handler 안내서

FAX : (02) 3787-0123

보내 주시는 의견은 더 나은 고객 지원 체제를 위한 귀중한 자료가 됩니다. 독자 여러분의 좋은 의견을 기다립니다.

성 명		직위/담당업무	
회 사 명		부 서 명	
주 소			
전화번호		팩스번호	
전자우편 주소			
사용중인 시스템	o 중대형 서버 o UNIX 서버 o 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.