개발자 안내서

Supplement(보충판)

Borland®
Kylix™ 2
Delphi™ for Linux®

볼랜드 코리아 주식회사
서울특별시 강남구 삼성동 159-1 ASEM 타워 30층
연락처:(02)6001-3162
www.borlandkorea.co.kr

Kylix2 개발자 안내서 Supplement(보충판) 소개

Kylix2 Supplement(보충판)는 기존의 Kylix1 개발자 안내서에서 수정되거나 추가된 내용으로 구성된 문서입니다. 따라서 Kylix1의 개발자 안내서와 본 문서를 함께 사용하시기 바랍니다.

Kylix1의 개발자 안내서에 같은 장 제목으로 들어 있는 내용보다 본 문서의 내용이 보다 새로운 내용이므로, 같은 제목의 장에 대해서는 본 문서의 내용을 참조하십시오. 또한 함께 제공되는 영문 개발자 안내서(Developer's Guide)의 PDF 파일도 함께 참조하실 수 있습니다.

Kylix1에 익숙하신 사용자께서는 본 문서를 통해 Kylix 2에서 수정되거나 추가된 내용을 쉽게 보실 수 있습니다. Kylix를 처음 접하시는 사용자께서는 반드시 Kylix1의 개발자 안내서와 본 문서를 함께 사용하시기 바랍니다.

감사합니다.

Printed in the U.S.A.

HDE7020WW21001 2E1R1001
0102030405-9 8 7 6 5 4 3 2 1
D3

목 차

1 부
Kylix 프로그래밍

1 장

애플리케이션과 공유 객체 구축 1-1
애플리케이션 생성 1-1
GUI 애플리케이션 1-1
사용자 인터페이스 모델 1-2
IDE, 프로젝트 및 컴파일 옵션 설정 . . 1-2
기본 프로젝트 옵션 설정 1-2
콘솔 애플리케이션 1-2
패키지와 공유 객체 파일 생성 1-3
공유 객체 라이브러리 사용 1-3
패키지와 공유 객체를 사용하는 경우 . . . 1-4
데이터베이스 애플리케이션 개발 1-4
분산 데이터베이스 애플리케이션 1-5
인터넷용 애플리케이션 개발 1-5
웹 서버 애플리케이션 생성 1-5
Web Broker 사용 1-6
WebSnap 애플리케이션 생성 1-6
Web Services 애플리케이션 생성 1-7
저수준(low-level) 인터넷 애플리케이션 개발 1-7
데이터 모듈 사용 1-7
데이터 모듈 생성 1-8
데이터 모듈에서 비즈니스 룰 만들기 . 1-8
폼에서 데이터 모듈 액세스 1-8
프로그래밍 템플릿 1-9
코드 공유: Object Repository 사용 1-9
프로젝트 내에서 항목 공유 1-9
항목 추가: Object Repository 1-10
팀 환경에서 객체 공유 1-10
프로젝트에서 Object Repository 항목 사용 1-11
항목 복사 1-11
항목 상속 1-11
항목 사용 1-11
프로젝트 템플릿 사용 1-11
공유 항목 수정 1-12
기본 프로젝트, 새 폼 및 메인 폼 지정 . . 1-12
컴포넌트와 컴포넌트 그룹의 재사용 1-12
컴포넌트 템플릿 생성 및 사용 1-13
CLX 애플리케이션에서 도움말 지원 1-13
도움말 시스템 인터페이스 1-14
ICustomHelpViewer 구현 1-15
Help Manager와 통신 1-15
Help Manager에 정보 요청 1-15
키워드 방식 도움말 표시 1-16
목차 표시 1-17
IExtendedHelpViewer 구현 1-17
IHelpSelector 구현 1-18
도움말 시스템 객체 등록 1-19
도움말 뷰어 등록 1-19
도움말 선택기 등록 1-19
CLX 애플리케이션에서 도움말 사용 1-19
TApplication의 도움말 처리 방법 1-19
컨트롤의 도움말 처리 방법 1-20
도움말 시스템 직접 호출 1-20
IHelpSystem 사용 1-20
IDE 도움말 시스템 사용자 지정 1-21

2 장

크로스 플랫폼 애플리케이션 개발 2-1
Windows 애플리케이션을 Linux로 포팅 . . . 2-1
포팅 기법 2-2
플랫폼별 포팅 2-2
크로스 플랫폼 포팅 2-2
Windows 에뮬레이션 포팅 2-2
애플리케이션 포팅 2-2
CLX와 VCL 비교 2-4
VCL과 CLX의 차이점 2-5
룩앤필(Look and feel) 2-5
스타일 2-5
Variants 2-5
레지스트리 사용 안 함 2-6
메소드 공유 2-6
그 밖의 차이점 2-6
CLX 또는 Kylix에 없는 기능 2-7
포팅되지 않는 기능 2-8
CLX와 VCL 유닛 2-8
CLX 객체 생성자의 차이점 2-10
Windows와 Linux 간의 소스 파일 공유 . 2-10
Windows와 Linux 간의 환경적 차이점 . . 2-11
Linux의 디렉토리 구조 2-13
이식 가능한 코드 작성 2-14
조건 지시어 사용 2-15
조건 지시어 종료 2-16
메시지 나타내기 2-17
인라인 어셈블러 코드 포함 2-17
메시지와 시스템 이벤트 2-18
Linux에서의 프로그래밍 차이점 2-19

크로스 플랫폼 데이터베이스 애플리케이션 2-20
dbExpress 차이점 2-20
컴포넌트 수준 차이점 2-21
사용자 인터페이스 수준 차이점 2-22
데이터베이스 애플리케이션을 Linux로 포팅 2-23
dbExpress 애플리케이션에서 데이터 업데이트. 2-25
크로스 플랫폼 인터넷 애플리케이션 2-26
인터넷 애플리케이션을 Linux로 포팅. . . 2-27

3 장

패키지와 컴포넌트 사용 3-1
패키지의 이점 3-2
패키지와 표준 공유 객체 파일 3-2
런타임 패키지 3-3
애플리케이션에서 패키지 사용 3-3
패키지 동적 로딩 3-4
사용할 런타임 패키지 결정 3-4
사용자 지정 패키지 3-6
디자인 타임 패키지 3-6
컴포넌트 패키지 설치 3-6
패키지 생성 및 편집 3-7
패키지 생성 3-7
기존 패키지 편집 3-8
수동으로 패키지 소스 파일 편집 3-9
패키지 구조 이해 3-9
패키지 이름 지정. 3-9
Requires 절 3-10
Contains 절 3-10
패키지 컴파일 3-11
패키지 특정 컴파일러 지시어 3-11
약한 패키징 3-12
명령줄 컴파일러와 링커 사용 3-12
컴파일이 성공했을 때 생성되는 패키지 파일 3-13
패키지 배포 3-13
패키지를 사용하는 애플리케이션 배포 . . 3-13
다른 개발자에게 패키지 배포 3-14

2 부 데이터베이스 애플리케이션 개발

4 장

웹 서비스를 사용하여 다계층 데이터베이스 애플리케이션 생성 4-1
다계층 데이터베이스 모델의 이점 4-2
다계층 데이터베이스 애플리케이션 이해 . . 4-2
3계층 애플리케이션 개요 4-3
클라이언트 애플리케이션의 구조 4-4
애플리케이션 서버의 구조. 4-4
원격 데이터 모듈의 컨텐트 4-5
SOAP 연결 사용. 4-5
다계층 애플리케이션 생성 4-6
애플리케이션 서버 생성 4-6
원격 데이터 모듈 설정 4-7
TSoapDataModule 구성 4-7
애플리케이션 서버의 인터페이스 확장. . . 4-8
다계층 애플리케이션의 트랜잭션 관리. . . 4-8
마스터/디테일 관계 지원 4-8
원격 데이터 모듈의 상태 정보 지원 4-9
애플리케이션 서버 등록 4-11
클라이언트 애플리케이션 생성 4-11
애플리케이션 서버에 연결. 4-12
SOAP를 사용하여 연결 지정 4-12
서버 연결 관리 4-13
서버에 연결 4-13
서버 연결 끊기 및 바꾸기. 4-13
서버 인터페이스 호출. 4-14

5 장

데이터베이스 애플리케이션에서 XML 사용 5-1
변환 정의 5-1
XML 노드와 데이터 패킷 필드 간 매핑. . . 5-2
XMLMapper 사용 5-4
XML 스키마 또는 데이터 패킷 로딩. . . 5-4
매핑 정의 5-4
변환 파일 생성 5-5
XML 문서를 데이터 패킷으로 변환 5-6
소스 XML 문서 지정 5-6
변환 지정 5-6
결과 데이터 패킷 가져오기 5-7
사용자 정의 노드 변환 5-7
프로바이더의 소스로 XML 문서 사용. . . . 5-8
프로바이더의 클라이언트로 XML 문서 사용. 5-9
프로바이더에서 XML 문서 페치(fetch). . . 5-9

XML 문서에서 프로바이더로
업데이트 적용 5-10

3 부
인터넷 애플리케이션 개발

6 장

인터넷 서버 애플리케이션 생성 6-1
Web Broker 및 WebSnap 정보 6-1
용어 및 표준 6-3
URL(Uniform Resource Locator)의
각 부분 6-3
URI와 URL 비교 6-4
HTTP 요청 헤더 정보 6-4
HTTP 서버 활동. 6-5
클라이언트 요청 구성 6-5
CGI 요청 서비스 6-5
동적 공유 객체 요청 서비스 6-6
웹 서버 애플리케이션의 유형 6-6
CGI 독립형 6-6
Apache DSO 모듈 6-6

7 장

Web Broker 사용 7-1
Web Broker로
웹 서버 애플리케이션 생성 7-1
웹 모듈 7-2
웹 애플리케이션 객체 7-2
Web Broker 애플리케이션 구조 7-3
웹 디스패처 7-4
디스패처에 액션 추가 7-4
요청 메시지 디스패칭 7-4
액션 항목 7-5
액션 항목 실행 시기 결정 7-5
대상 URL 7-5
요청 메소드 유형. 7-6
액션 항목 활성화 및 비활성화 7-6
기본 액션 항목 선택 7-6
액션 항목으로 요청 메시지에 응답 . . . 7-7
응답 보내기 7-7
여러 가지 액션 항목 사용 7-7
클라이언트 요청 정보 액세스 7-8
요청 헤더 정보를 포함하는 속성 7-8
대상을 식별하는 속성. 7-8
웹 클라이언트를 설명하는 속성 . . . 7-8
요청 목적을 식별하는 속성 7-9
예상 응답을 설명하는 속성 7-9
컨텐트를 설명하는 속성. 7-9
HTTP 요청 메시지의 내용 7-10
HTTP 응답 메시지 만들기 7-10
응답 헤더 채우기 7-10
응답 상태 표시 7-10
클라이언트 액션에 대한 요구 표시 . . 7-11
서버 애플리케이션 설명 7-11
컨텐트 설명 7-11
응답 컨텐트 설정 7-11
응답 보내기. 7-12
응답 메시지 컨텐트 생성. 7-12
페이지 프로듀서 컴포넌트 사용 7-13
HTML 템플릿 7-13
HTML 템플릿 지정. 7-14
HTML 투명 태그 변환 7-14
액션 항목에서 페이지 프로듀서 사용 . 7-14
페이지 프로듀서 간의 연결 7-15
응답에서 데이터베이스 정보 사용. 7-16
웹 모듈에 세션 추가 7-16
HTML로 데이터베이스 정보 표시 7-16
데이터셋 페이지 프로듀서 사용 . . . 7-17
테이블 프로듀서 사용 7-17
테이블 속성 지정 7-17
행 속성 지정 7-18
열 지정 7-18
HTML 문서에 테이블 포함(embed). . 7-18
데이터셋 테이블 프로듀서 설정 . . . 7-18
쿼리 테이블 프로듀서 설정 7-19

8 장

WebSnap 사용 8-1
기본적인 WebSnap 컴포넌트. 8-2
웹 모듈. 8-2
웹 애플리케이션 모듈 유형 8-3
웹 페이지 모듈 8-3
웹 데이터 모듈 8-4
어댑터 8-5
필드. 8-5
액션. 8-5
오류. 8-6
레코드. 8-6
페이지 프로듀서 8-6
WebSnap을 사용한
웹 서버 애플리케이션 생성 8-7
서버 유형 8-8
애플리케이션 모듈 컴포넌트. 8-9
웹 애플리케이션 모듈 옵션 8-10
WebSnap 자습서. 8-11
새 애플리케이션 생성. 8-11

1단계. WebSnap 애플리케이션 마법사 시작 8-11
2단계. 생성된 파일과 프로젝트 저장 8-11
3단계. 애플리케이션 제목 지정 8-12
CountryTable 페이지 생성 8-12
1단계. 새 웹 페이지 모듈 추가 8-12
2단계. 새 웹 페이지 모듈 저장 8-13
CountryTable 모듈에 데이터 컴포넌트 추가 8-13
1단계. data-aware 컴포넌트 추가 . . 8-14
2단계. 키 필드 지정 8-14
3단계. 어댑터 컴포넌트 추가 8-14
데이터를 표시할 그리드 생성 8-15
1단계. 그리드 추가 8-15
2단계. 그리드에 편집 명령 추가 . . . 8-17
Edit 폼 추가 8-18
1단계. 새 웹 페이지 모듈 추가 8-18
2단계. 새 모듈 저장 8-19
3단계. CountryTableU 유닛 사용 . . . 8-19
4단계. 입력 필드 추가 8-19
5단계. 버튼 추가 8-20
6단계. 그리드 페이지에 폼 액션 연결 8-21
7단계. 폼 페이지에 그리드 액션 연결 8-21
완성된 애플리케이션 실행 8-22
오류 보고 추가 8-22
1단계. 그리드에 오류 지원 추가 . . . 8-22
2단계. 폼에 오류 지원 추가 8-22
3단계. 오류 보고 메커니즘 테스트 . . 8-23
고급 HTML 디자인 8-23
HTML 파일에서 서버측 스크립트 수정 . . 8-24
로그인 지원 8-25
로그인 지원 추가 8-25
세션 서비스 사용 8-26
로그인 페이지 8-27
로그인 요구 페이지 설정 8-28
사용자 액세스 권한 8-29
편집 상자 및 텍스트 상자로 필드를 동적으로 표시 8-29
필드와 컨텐트 숨기기 8-30
페이지 액세스 차단 8-30
WebSnap에서의 서버측 스크립팅 8-31
스크립트 엔진 8-31
스크립트 블록 8-32
스크립트 작성 8-32
템플릿 마법사 8-32
TAdapterPageProducer 8-32
스크립트 편집과 보기 8-33
페이지에 스크립트 포함 8-33
스크립트 객체 8-33
WebSnap 서버측 스크립팅 참조 8-35
전역 객체 8-35
Application 8-35
EndUser 8-36
Modules 8-37
Page 8-37
Pages 8-37
Producer 8-38
Request 8-38
Response 8-39
Session 8-39
객체 타입 8-39
AdapterType 8-40
AdapterActionType 8-42
AdapterErrorsType 8-44
AdapterFieldType 8-44
AdapterFieldValuesType 8-47
AdapterFieldValuesListType 8-48
AdapterHiddenFieldsType 8-49
AdapterImageType 8-49
ModuleType 8-50
PageType 8-50
예제 8-51
예제 1 8-52
예제 2 8-53
예제 3 8-53
예제 4 8-53
예제 5 8-53
예제 6 8-54
예제 7 8-54
예제 8 8-55
예제 9 8-55
예제 10 8-56
예제 11 8-58
예제 12 8-59
예제 13 8-60
예제 14 8-61
예제 15 8-62
예제 16 8-64
예제 17 8-65
예제 18 8-66
예제 19 8-67
예제 20 8-67
예제 21 8-68
예제 22 8-69
요청 디스패치 8-71

웹 컨텍스트 8-71
디스패처 컴포넌트 8-71
어댑터 디스패처 작업 8-71
어댑터 컴포넌트를 사용하여 컨텐트 생성 8-71
어댑터 요청 및 응답 8-73
액션 요청 8-73
액션 응답 8-74
이미지 요청 8-75
이미지 응답 8-75
액션 항목 디스패칭 8-76
페이지 디스패처 작업 8-77

9 장

XML 문서 사용 9-1
DOM 사용 9-2
XML 컴포넌트 사용 9-3
TXMLDocument 사용 9-3
XML 노드 사용 9-4
노드 값 사용 9-4
노드 속성 사용. 9-5
자식 노드 추가 및 삭제 9-5
데이터 바인딩 마법사로 XML 문서 요약. . . 9-5
XML 데이터 바인딩 마법사 사용. 9-7
XML 데이터 바인딩 마법사가 생성하는 코드 사용 9-8

10 장

웹 서비스 사용 10-1
웹 서비스를 지원하는 서버 작성 10-2
웹 서비스 서버 구축. 10-2
호출 가능한 인터페이스 정의 10-3
호출 가능한 인터페이스에서 복잡한 타입 사용 10-5
구현 및 등록 10-6
웹 서비스에 대한 사용자 지정 예외 클래스 생성 10-7
웹 서비스 애플리케이션에 대한 WSDL 문서 생성 10-7
웹 서비스용 클라이언트 작성 10-8
WSDL 문서 import하기 10-9
호출 가능 인터페이스 호출 10-9

11 장

소켓 사용 11-1
서비스 구현 11-1
서비스 프로토콜 이해. 11-2
애플리케이션과 통신. 11-2
서비스와 포트 11-2
소켓 연결 유형 11-2
클라이언트 연결 11-3
리스닝(listening) 연결 11-3
서버 연결 11-3
소켓 설명 11-3
호스트 설명 11-4
호스트 이름 또는 IP 주소 선택. . . . 11-4
포트 사용 11-5
소켓 컴포넌트 사용 11-5
연결 정보 얻기 11-5
클라이언트 소켓 사용. 11-6
원하는 서버 지정 11-6
연결 구성 11-6
연결 정보 얻기 11-6
연결 닫기 11-6
서버 소켓 사용 11-7
포트 지정 11-7
클라이언트 요청 리스닝(listening) . . 11-7
클라이언트에 연결. 11-7
서버 연결 닫기 11-7
소켓 이벤트에 응답 11-7
오류 이벤트 11-8
클라이언트 이벤트 11-8
서버 이벤트 11-8
리스닝 시의 이벤트 11-8
클라이언트 연결 시의 이벤트 11-9
소켓 연결을 통한 읽기 및 쓰기 11-9
비차단 연결 (Non-blocking connections) 11-9
읽기 및 쓰기 이벤트 11-9
차단 연결(Blocking connections) . . 11-10

색인 I-1

I

Kylix 프로그래밍

"Kylix 프로그래밍"의 각 장은 Kylix 애플리케이션 생성에 필요한 개념과 기술에 대해 소개합니다. 또한 *개발자 안내서*의 이후 단원에서 설명될 개념에 대해서도 소개합니다.

1

애플리케이션과 공유 객체 구축

* 이 장은 영문 Kylix2 개발자 안내서의 5장입니다.

이 장에서는 Kylix를 사용하여 애플리케이션과 공유 객체를 만드는 방법에 대한 개요를 다룹니다.

애플리케이션 생성

Kylix의 주된 용도는 다음과 같은 애플리케이션을 디자인하고 구축하는 것입니다.

- GUI 애플리케이션
- 콘솔 애플리케이션

GUI 애플리케이션

GUI(그래픽 사용자 인터페이스) 애플리케이션은 창, 메뉴, 대화 상자와 같은 그래픽 기능과 애플리케이션을 사용하기 쉽게 만드는 기능을 사용하여 디자인된 애플리케이션입니다. GUI 애플리케이션을 컴파일할 때 시동 코드와 함께 실행 파일이 생성됩니다. 실행 파일은 보통 사용자 프로그램의 기본적인 기능을 제공하며, 단순한 프로그램인 경우 대개 실행 파일 하나로만 구성됩니다. 공유 객체 파일, 패키지 및 실행 파일의 기타 지원 파일을 호출하여 애플리케이션을 확장할 수 있습니다.

Kylix는 다음 두 가지 애플리케이션 UI 모델을 제공합니다.

- 단일 문서 인터페이스(SDI)
- 다중 문서 인터페이스(MDI)

애플리케이션의 구현 모델을 비롯한 프로젝트의 디자인 타임 동작(behavior)과 애플리케이션의 런타임 동작은 IDE에서 프로젝트 옵션을 설정하여 처리할 수 있습니다.

사용자 인터페이스 모델

폼을 다중 문서 인터페이스(MDI) 또는 단일 문서 인터페이스(SDI) 폼으로 구현할 수 있습니다. MDI 애플리케이션에서는 단일 부모 창에서 문서나 자식 창을 여러 개 열 수 있습니다. MDI는 스프레드시트나 워드 프로세서와 같은 애플리케이션에서 일반적인 인터페이스입니다. 반면 SDI 애플리케이션은 단일 문서 뷰를 가집니다. 폼을 SDI 애플리케이션으로 만들려면 *Form* 객체의 *FormStyle* 속성을 *fsNormal*로 설정합니다.

애플리케이션의 UI 개발에 대한 자세한 내용은 *Kylix1 개발자 안내서* 6장 "애플리케이션 사용자 인터페이스 개발"을 참조하십시오.

SDI 애플리케이션

새 SDI 애플리케이션을 만들려면 다음과 같이 합니다.

1 File|New를 선택하여 New Items 대화 상자를 나타냅니다.
2 Projects 페이지를 클릭하고 SDI Application을 선택합니다.
3 OK를 클릭합니다.

기본적으로 *Form* 객체의 *FormStyle* 속성은 *fsNormal*로 설정되므로 Kylix에서는 모든 새 애플리케이션을 SDI 애플리케이션으로 간주합니다.

MDI 애플리케이션

새 MDI 애플리케이션을 생성하려면 다음과 같이 합니다.

1 File|New를 선택하여 New Items 대화 상자를 나타냅니다.
2 Projects 페이지를 클릭하고 MDI Application을 선택합니다.
3 OK를 클릭합니다.

MDI 애플리케이션은 SDI 애플리케이션보다 디자인하기가 다소 복잡하므로 계획을 잘 세워야 합니다. MDI 애플리케이션은 클라이언트 창 내에 상주하는 자식 창을 생성하고 메인 폼은 자식 폼을 포함합니다. *TForm* 객체의 *FormStyle* 속성을 설정하여 자식 폼(*fsMDIChild*)인지 또는 메인 폼(*fsMDIForm*)인지 여부를 지정합니다. 자식 폼의 속성 재설정을 피하려면 자식 폼의 기본 클래스(base class)를 정의하고 이 클래스에서 각각의 자식 폼을 파생시키는 것이 좋습니다.

IDE, 프로젝트 및 컴파일 옵션 설정

Project|Options를 선택해서 프로젝트에 다양한 옵션을 지정합니다. 자세한 내용은 온라인 도움말을 참조하십시오.

기본 프로젝트 옵션 설정

차후의 모든 프로젝트에 적용될 기본 옵션을 변경하려면 Project Options 대화 상자에서 옵션을 설정하고 창의 오른쪽 하단에 있는 Default 상자를 선택 표시합니다. 모든 새 프로젝트에서 현재 옵션을 사용합니다.

콘솔 애플리케이션

콘솔 애플리케이션은 창 관리 레이어 없이 콘솔 창에서 직접 실행시킬 수 있습니다. 콘솔 애플리케이션은 그래픽 인터페이스 없이 실행되는 32비트 프로그램이며 대체로 사용자 입력이 많지 않고 실행되는 함수들도 제한되어 있습니다.

새 콘솔 애플리케이션을 생성하려면 다음과 같이 합니다.

1 File|New를 선택한 다음 New Items 대화 상자에서 Console Application을 선택합니다.

Kylix는 콘솔형 소스 파일에 대한 프로젝트 파일을 생성하고 코드 에디터를 표시합니다.

참고 새 콘솔 애플리케이션을 생성하면 IDE는 새 폼을 만들지 않고 코드 에디터만 표시합니다.

패키지와 공유 객체 파일 생성

공유 객체 파일은 실행 파일과 함께 작동되며 애플리케이션에 추가 기능을 제공하는 컴파일된 코드 모듈입니다. 공유 객체 파일은 애플리케이션이 실행되는 동안 오류가 나타나지 않게 모든 예외를 처리하도록 디자인해야 됩니다.

패키지는 Kylix 애플리케이션, IDE 또는 두 가지에서 모두 사용되는 특별한 공유 객체 파일입니다. 패키지의 종류에는 런타임 패키지와 디자인 타임 패키지가 있습니다. 런타임 패키지는 프로그램 실행 중에 프로그램에 기능을 제공합니다. 디자인 타임 패키지는 IDE의 기능을 확장합니다.

패키지에 대한 자세한 내용은 3장 "패키지와 컴포넌트 사용"을 참조하십시오.

공유 객체 라이브러리 사용

Linux의 공유 객체 라이브러리는 Windows DLL과 유사합니다. Windows에서 DLL 함수를 사용하는 것처럼 외부 함수 선언을 사용하여 협력 업체의 공유 객체와 연결할 수 있습니다.

Linux 프로그램 로더는 외부 함수 참조를 해석할 때 모듈 이름 바인딩을 무시합니다. 애플리케이션이 Foo라는 함수를 export하는 .so 라이브러리를 두 개 사용하면 로더는 로더가 찾는 첫 번째 Foo 함수에 모든 Foo 참조를 바인딩합니다(검색 순서는 ELF 표준의 "Shared Object Dependencies" 단원에 설명되어 있습니다). 이름 충돌을 피할 수 없을 때 dlopen()을 사용하여 객체를 동적으로 로드함으로써 의도하지 않은 동작(behavior)을 방지할 수 있습니다.

라이브러리 프로젝트를 구축할 때 컴파일러는 일반적인 실행 파일 대신에 공유 객체(.so 파일)를 생성합니다. 생성된 파일 이름은 기본적으로 "lib"(표준 라이브러리인 경우) 또는 "bpl"(패키지인 경우)로 시작합니다. 예를 들어 프로젝트 파일이 something.pas이면 컴파일러는 libsomething.so 또는 bplsomething.so라는 공유 객체를 생성합니다.

다음과 같은 컴파일러 지시어를 라이브러리 프로젝트 파일에 사용할 수 있습니다.

표 1.1 라이브러리용 컴파일러 지시어

컴파일러 지시어	설명
{$SOPREFIX 'string'}	출력 파일 이름에서 기본 접두어 'lib' 또는 'bpl'을 오버라이드합니다. 예를 들어 디자인 타임 패키지에 {$SOPREFIX 'dcl'}을 지정하거나 {$SOPREFIX ' '}를 사용하여 접두어를 없앨 수 있습니다.
{$SOSUFFIX 'string'}	출력 파일 이름에서 .so 확장자 앞에 지정된 접미어를 추가합니다. 예를 들어 something.pas에서 {$SOSUFFIX '-2.1.3'}을 사용하여 libsomething-2.1.3.so를 만듭니다.
{$SOVERSION 'string'}	출력 파일 이름에서 .so 확장자 뒤에 두 번째 확장자를 추가합니다. 예를 들어 something.pas에서 {$SOVERSION '2.1.3'}을 사용하여 libsomething.so.2.1.3을 만듭니다.

패키지와 공유 객체를 사용하는 경우

Kylix로 개발된 대부분의 애플리케이션에서 패키지는 공유 객체보다 훨씬 유연하며 더 쉽게 생성됩니다. 그러나 프로젝트에 패키지보다 공유 객체가 더 적합한 경우도 있습니다.

- 코드 모듈이 Kylix 이외의 애플리케이션에서 호출되는 경우
- 웹 서버의 기능을 확장하는 경우
- 협력 업체의 개발자를 위해 코드 모듈을 생성하는 경우

런타임 타입 정보(RTTI)를 공유 객체 간에 또는 공유 객체에서 실행 파일로 전달할 수는 없습니다. 왜냐하면 모든 공유 객체는 자기 자신의 심볼 정보를 갖고 있기 때문입니다. 공유 객체로부터 *TStrings* 객체를 전달해야 하는 경우에는 **is** 또는 **as** 연산자를 사용하므로 공유 객체보다는 패키지로 만들어야 합니다. 패키지들은 심볼 정보를 공유합니다.

데이터베이스 애플리케이션 개발

참고 Kylix의 모든 버전이 데이터베이스 지원을 포함하지는 않습니다.

Kylix의 장점 중 하나는 고급 데이터베이스 애플리케이션 생성에 대한 지원입니다. Kylix에는 애플리케이션 간에 투명한 데이터 공유를 제공하면서 InterBase, MySQL 또는 기타 서버에 연결할 수 있도록 하는 기본 제공 툴이 있습니다.

데이터베이스 애플리케이션은 사용자 인터페이스 요소, 데이터베이스 정보(데이터셋)를 나타내는 컴포넌트 및 데이터베이스 정보 자체에 연결하는 컴포넌트로 구축됩니다. Kylix는 다음 두 종류의 데이터셋을 지원합니다.

- dbExpress
- 클라이언트

다양한 종류의 데이터셋은 원본으로 사용하는 데이터베이스 정보에 다양한 방식으로 연결됩니다. *dbExpress*는 데이터베이스 정보에 대한 빠른 액세스를 제공하고 크로스 플랫폼 개발을 지원하지만 데이터 처리 함수가 많이 들어 있지는 않습니다. 클라이언트 데이터셋은 데이터를 메모리에 버퍼링할 수 있지만 클라이언트 데이터셋을 데이터베이스 서버에 직접 연결할 수 없습니다. 왜냐하면 클라이언트 데이터셋에는 기본 제공 데이

터베이스 액세스 메커니즘이 없기 때문입니다. 대신 데이터 액세스를 처리할 수 있는 다른 데이터셋에 클라이언트 데이터셋을 연결해야 합니다. *Kylix1 개발자 안내서* 14-4페이지의 "데이터베이스 아키텍처"를 참조하십시오.

*Kylix 개발자 안내서*의 2부에서는 Kylix를 사용하여 데이터베이스 애플리케이션을 디자인하는 방법에 대한 자세한 내용을 제공합니다.

분산 데이터베이스 애플리케이션

Kylix에서는 컴포넌트들을 조합하여 분산 데이터베이스 애플리케이션을 만들 수 있습니다. 분산 데이터베이스 애플리케이션에서 CORBA, TCP/IP 및 SOAP 등을 비롯한 다양한 통신 프로토콜을 사용할 수 있습니다.

분산 데이터베이스 애플리케이션을 만드는 자세한 내용은 4 장 "웹 서비스를 사용하여 다계층 데이터베이스 애플리케이션 생성"을 참조하십시오.

인터넷용 애플리케이션 개발

인터넷에서 사용할 수 있는 다양한 유형의 애플리케이션을 Delphi를 사용하여 개발할 수 있습니다. 저수준(low-level) 인터넷 프로그래밍에 TCP/IP와 소켓을 사용할 수 있습니다. 대부분의 개발자는 웹 컨텐트를 전달하는 웹 서버 애플리케이션을 개발하는 데 Web Broker 및 WebSnap과 같은 기술을 사용할 것입니다. 또 다른 인터넷 기술로는 랭귀지 중립 방식으로 인터넷 상에서 다른 프로그램이 호출할 수 있는 애플리케이션인 Web Services가 있습니다. 인터넷 상에서 통신하는 API를 개발하는 경우에는 Web Services를 사용하는 것이 좋습니다.

Web Broker는 Web Snap과 Web Services에서 모두 사용할 수 있는 기본 아키텍처를 제공합니다. Web Broker의 프레임워크는 HTTP 메시지에 응답하는 데 기초가 됩니다.

다음 단원에서는 최신 Web Services 기술에 대한 개요를 제공합니다. TCP/IP와 소켓에 대한 소개는 1-7페이지의 "저수준(low-level) 인터넷 애플리케이션 개발"을 참조하십시오.

웹 서버 애플리케이션 생성

웹 서버 애플리케이션은 인터넷 상에 HTML 웹 페이지나 XML 문서와 같은 웹 컨텐트를 전달하는 서버에서 실행되는 애플리케이션입니다. 웹 서버 애플리케이션의 예로는 웹 사이트에 대한 액세스를 제어하고 주문서를 작성하거나 정보 요청에 응답하는 애플리케이션이 있습니다.

웹 서버 애플리케이션은 웹 서버로부터 HTTP 요청 메시지를 수신하여 메시지에서 요청한 모든 액션을 수행한 다음 웹 서버로 응답을 다시 전달합니다. Kylix 애플리케이션으로 수행할 수 있는 여러 작업을 하나의 웹 서버 애플리케이션으로 통합할 수 있습니다.

다음과 같은 Kylix 기술을 사용하여 여러 가지 다른 유형의 인터넷 애플리케이션을 만들 수 있습니다.

- Web Broker
- WebSnap
- Web Services

이 기술을 사용하여 애플리케이션을 설치하는 웹 서버를 다음과 같은 유형으로 만들 수 있습니다.

표 1.2 웹 서버 애플리케이션

웹 서버 애플리케이션 유형	설명
CGI 독립형 실행 파일	CGI 웹 서버 애플리케이션은 표준 입력에서 클라이언트의 요청을 수신하여 처리한 다음 클라이언트에 보낼 결과를 표준 출력으로 서버에 돌려 보내는 콘솔 애플리케이션입니다. 이 유형의 애플리케이션을 선택하면 필요한 항목이 프로젝트 파일의 **uses** 절에 추가되고 적절한 $APPTYPE 지시어가 소스에 추가됩니다.
Apache 공유 모듈	이 유형의 애플리케이션을 선택하면 프로젝트는 공유 객체로 설정됩니다. Apache 웹 서버 애플리케이션은 웹 서버에 의해 로드되는 공유 객체입니다. 정보가 공유 객체에 전달되고 처리되며 웹 서버에 의해 클라이언트에 반환됩니다.

Web Broker 사용

Web Broker(NetCLX 아키텍처라고도 함)는 CGI 애플리케이션이나 공유 객체와 같은 웹 서버 애플리케이션을 만드는 데 사용할 수 있습니다. 이러한 웹 서버 애플리케이션에는 모든 넌비주얼 컴포넌트가 포함될 수 있습니다. 컴포넌트 팔레트의 Internet 페이지에 있는 컴포넌트는 이벤트 핸들러를 생성하고 프로그램에서 HTML 또는 XML 문서를 생성하여 생성한 문서를 클라이언트에게 전송합니다.

새 웹 서버 애플리케이션을 만들려면 File|New를 선택한 다음 New Items 대화 상자에서 Web Server Application을 선택합니다. 그런 다음 웹 서버 애플리케이션 유형을 선택합니다.

웹 서버 애플리케이션 구축에 대한 자세한 내용은 6장 "인터넷 서버 애플리케이션 생성"을 참조하십시오.

WebSnap 애플리케이션 생성

웹 브라우저와 상호 작용하는 고급 웹 서버를 구축할 때 WebSnap을 사용할 수 있습니다. WebSnap 컴포넌트는 웹 페이지의 HTML을 생성하거나 다른 MIME 컨텐트를 생성합니다. WebSnap은 서버측 개발을 위한 기술이며 프로그래밍을 간단하게 해주는 기능이 많이 있습니다. 예를 들면 WebSnap은 여러 가지 모듈과 서버측 스크립팅을 지원하며 세션 관리와 같은 일반 작업을 수행하는 많은 객체들을 지원합니다.

새 WebSnap 애플리케이션을 만들려면 File|New를 선택한 후 New Items 대화 상자에서 WebSnap 탭을 선택합니다. WebSnap Application을 선택합니다. 그런 다음 웹 서버 애플리케이션 유형(CGI, Apache)을 선택합니다. 자세한 내용은 표 1.2 "웹 서버 애플리케이션"을 참조하십시오.

WebSnap에 대한 자세한 내용은 8장 "WebSnap 사용"을 참조하십시오.

Web Services 애플리케이션 생성

Web Services는 WWW와 같은 네트워크 상에서 게시 및 호출할 수 있는 모듈이 자체적으로 포함되어 있는 애플리케이션입니다. Web Services는 제공된 서비스를 설명하는 잘 정의된(well-defined) 인터페이스를 제공합니다. Web Services, XML, XML Schema, SOAP(Simple Object Access Protocol) 및 WSDL(Web Service Definition Language)과 같은 새로운 표준을 사용하여 인터넷 상에 프로그래밍할 수 있는 서비스를 만드는 데 사용됩니다.

Web Services는 분산 환경에서 정보를 교환하는 표준 경량급(lightweight) 프로토콜인 SOAP를 사용합니다. 통신 프로토콜은 HTTP를 사용하고 원격 프로시저 호출을 인코딩하는 데 XML을 사용합니다.

Kylix를 사용하여 Web Services를 구현하는 서버 및 서비스에서 호출하는 클라이언트를 구축할 수 있습니다. SOAP 메시지에 응답하는 Web Services를 구현하는 임의의 서버 및 임의의 클라이언트에 사용할 목적으로 Web Services를 게시하는 Kylix 서버에 대한 클라이언트를 작성할 수 있습니다.

Web Services에 대한 자세한 내용은 10장 "웹 서비스 사용"을 참조하십시오.

저수준(low-level) 인터넷 애플리케이션 개발

TCP/IP는 네트워크 상에서 통신하는 애플리케이션을 개발할 수 있는 통신 프로토콜입니다. 가상으로 애플리케이션에 모든 디자인을 구현할 수 있습니다. TCP/IP는 전송 레이어(transport layer)를 제공하지만 분산 애플리케이션 생성을 위한 특별한 아키텍처가 필요하지는 않습니다.

인터넷이 확산됨에 따라 대부분의 컴퓨터가 이미 TCP/IP 액세스가 가능한 환경을 갖추고 있으므로 애플리케이션의 배포와 설치가 단순화되었습니다.

TCP/IP를 사용하는 애플리케이션은 HTTP 요청 메시지를 서비스하는 웹 서버 애플리케이션과 같은 메시지 방식의 분산 애플리케이션 또는 소켓을 사용하여 통신하는 분산 데이터베이스 애플리케이션과 같은 분산 객체 애플리케이션입니다.

TCP/IP 기능을 애플리케이션에 추가하는 가장 기본적인 방법은 클라이언트 또는 서버 소켓을 사용하는 것입니다. 가장 낮은 수준에서는 HTTP 메시지를 소켓을 사용하여 보냅니다. *TTCPClient* 클래스와 *TTCPServer* 클래스를 사용하여 TCP/IP 소켓 연결을 만들어 다른 원격 애플리케이션과 통신할 수 있습니다. 소켓에 대한 자세한 내용은 11장 "소켓 사용"을 참조하십시오.

데이터 모듈 사용

*데이터 모듈*은 넌비주얼(nonvisual) 컴포넌트를 포함하는 특수한 폼과 유사합니다. 데이터 모듈의 모든 컴포넌트는 비주얼 컨트롤과 함께 일반적인 폼에 *둘 수 있습니다*. 그러나 데이터베이스와 시스템 객체 그룹을 재사용하거나 비즈니스 룰을 처리하는 애플리케이션의 일부를 데이터베이스 연결과 분리하려는 경우에 데이터 모듈은 간편한 구성 툴을 제공합니다.

데이터 모듈의 종류에는 표준 데이터 모듈과 웹 모듈의 두 가지가 있습니다.

- 표준 데이터 모듈을 사용하여 애플리케이션을 보다 쉽게 구성하고 작성할 수 있습니다. 표준 데이터 모듈은 단일 계층 및 2계층(2-tier) 데이터베이스 애플리케이션에 특히 유용하지만 모든 애플리케이션에서 넌비주얼(nonvisual) 컴포넌트를 구성하는 데 사용할 수 있습니다. 자세한 내용은 1-8페이지의 "데이터 모듈 생성"을 참조하십시오.
- 웹 모듈은 웹 서버 애플리케이션의 기본을 형성합니다. 웹 모듈은 HTTP 응답 메시지의 내용을 만드는 컴포넌트를 유지하는 것 외에 클라이언트 애플리케이션에서 HTTP 메시지의 디스패칭을 처리합니다. 웹 모듈 사용에 대한 자세한 내용은 6장 "인터넷 서버 애플리케이션 생성"을 참조하십시오.

데이터 모듈 생성

데이터 모듈을 만들려면 File|New를 선택하고 Data Module을 더블 클릭합니다. Kylix는 빈 데이터 모듈을 열고 코드 에디터에 새 모듈에 대한 유닛 파일을 나타낸 다음 현재 프로젝트에 모듈을 추가합니다. 기존 데이터 모듈을 다시 열면 Kylix의 데이터 모듈 창에 컴포넌트가 표시됩니다.

디자인 타임에 컴포넌트 팔레트에서 넌비주얼 컴포넌트를 선택한 다음 데이터 모듈 창에서 클릭하여 넌비주얼 컴포넌트를 데이터 모듈에 추가할 수 있습니다. 데이터 모듈 창에서 컴포넌트를 선택하면 폼에서 컴포넌트를 선택하여 편집하듯이 Object Inspector에서 컴포넌트의 속성을 편집할 수 있습니다. 데이터 모듈에서 컴포넌트에 대해 설정한 속성은 모듈을 사용하는 애플리케이션의 모든 폼에 일관적으로 적용됩니다.

데이터 모듈의 이름은 데이터 모듈 창의 제목 표시줄에 나타납니다. 기본 이름은 DataModule*n*이며 이때 *n*은 프로젝트에서 사용되지 않은 유닛의 최소 번호를 나타냅니다. 예를 들어 새 프로젝트를 시작하면 다른 애플리케이션을 구축하기 전에 모듈을 프로젝트에 추가하며 이 경우 데이터 모듈의 이름은 기본적으로 DataModule2가 됩니다. DataModule2에 대한 해당 유닛 파일은 Unit2입니다(Unit1은 폼입니다). 데이터 모듈 창을 선택하고 Object Inspector에서 *Name* 속성을 편집하여 데이터 모듈의 이름을 변경할 수 있습니다.

데이터 모듈에서 비즈니스 룰 만들기

데이터 모듈의 유닛 파일에서 비즈니스 룰을 캡슐화하는 전역 루틴뿐만 아니라 모듈의 컴포넌트에 대한 이벤트 핸들러를 포함하는 여러 메소드를 작성할 수 있습니다. 예를 들어 월별, 분기별 또는 년도별 부기 작업을 수행하는 프로시저를 작성할 수 있고 모듈의 컴포넌트에 대한 이벤트 핸들러나 모듈을 사용하는 유닛에서 해당 프로시저를 호출할 수 있습니다.

폼에서 데이터 모듈 액세스

폼에 있는 비주얼 컨트롤을 데이터 모듈에 연결하려면 먼저 데이터 모듈을 폼의 **uses** 절에 추가해야 합니다. 이 작업을 다음과 같은 방법으로 수행할 수 있습니다.

- 코드 에디터에서 폼의 유닛 파일을 열고 데이터 모듈의 이름을 **interface** 섹션에 있는 **uses** 절에 추가합니다.

- File | Use Unit을 선택한 다음 모듈 이름을 입력하거나 Use Unit 대화 상자에 있는 리스트 박스에서 모듈 이름을 선택합니다.
- 데이터 모듈에서 테이블 또는 쿼리 컴포넌트를 더블 클릭하여 Fields 에디터를 엽니다. Fields 에디터에서 모든 필드를 폼으로 드래그합니다. Kylix는 모듈을 폼의 **uses** 절에 추가할지 사용자에게 물어 본 다음 편집 상자와 같은 필드용 컨트롤을 만듭니다.

프로그래밍 템플릿

프로그래밍 템플릿은 공통적으로 사용되는 "뼈대" 구조에 해당하므로 템플릿을 소스 코드에 추가한 다음 필요한 코드를 삽입할 수 있습니다. 예를 들어 코드에서 **for** 순환문을 사용하기 위해 다음과 같은 템플릿을 삽입할 수 있습니다.

```
for :=  to  do
begin

end;
```

코드 에디터에 코드 템플릿을 삽입하려면 *Ctrl-j*를 누르고 사용하려는 템플릿을 선택합니다. 사용자가 직접 만든 템플릿도 다음과 같은 방법으로 이 컬렉션에 추가할 수 있습니다.

1 Tools | Editor Options를 선택합니다.
2 Code Insight 탭을 클릭합니다.
3 템플릿 섹션에서 Add를 클릭합니다.
4 바로 가기 이름을 선택하고 새 템플릿에 대한 간단한 설명을 입력합니다.
5 템플릿 코드를 코드 텍스트 상자에 추가합니다.
6 OK를 클릭합니다.

코드 공유: Object Repository 사용

Object Repository(Tools | Repository)를 사용하면 폼, 대화 상자, 프레임 및 데이터 모듈을 쉽게 공유할 수 있습니다. Object Repository는 폼과 프로젝트를 만들 때 사용자를 안내하는 마법사 및 새 프로젝트에 대한 템플릿도 제공합니다. Object Repository는 Repository 및 New Items 대화 상자에 나타나는 항목에 대한 참조를 포함하는 텍스트 파일인 delphi65dro 파일(기본적으로 .borland 디렉토리에 있음)에서 유지 관리합니다.

프로젝트 내에서 항목 공유

항목을 Object Repository에 추가하지 않고 프로젝트 *내에서* 공유할 수 있습니다. New Items 대화 상자(File | New)를 열면 현재 프로젝트의 이름이 있는 페이지 탭을 볼 수 있습니다. 이 페이지에는 프로젝트에 있는 모든 폼, 대화 상자 및 데이터 모듈을 나열합니다. 기존 항목에서 새 항목을 파생시키고 필요하면 항목을 사용자 지정할 수 있습니다.

항목 추가: Object Repository

사용자의 프로젝트, 폼, 프레임 및 데이터 모듈을 Object Repository에 추가할 수 있습니다. 다음과 같은 방법으로 항목을 Object Repository에 추가합니다.

1. 항목이 프로젝트이거나 프로젝트에 있는 경우에 해당 프로젝트를 엽니다.
2. 프로젝트에서 Project | Add To Repository를 선택합니다. 폼 또는 데이터 모듈에서 항목을 마우스 오른쪽 버튼으로 클릭하고 Add To Repository를 선택합니다.
3. 설명, 제목 및 작성자를 입력합니다.
4. New Items 대화 상자에서 항목을 나타내려는 페이지를 결정한 다음 페이지의 이름을 입력하거나 Page 콤보 박스에서 페이지의 이름을 선택합니다. 존재하지 않는 페이지 이름을 입력한 경우 Kylix는 새 페이지를 생성합니다.
5. Object Repository에 있는 객체를 나타내는 아이콘을 선택하려면 Browse를 선택합니다.
6. OK를 선택합니다.

팀 환경에서 객체 공유

네트워크 상에서 레포지토리를 사용 가능하게 하여 작업 그룹 또는 개발 팀과 객체를 공유할 수 있습니다. 공유 레포지토리를 사용하려면 팀원 모두 Environment Options 대화 상자에서 동일한 Shared Repository 디렉토리를 선택해야 합니다.

1. Tools | Environment Options를 선택합니다.
2. Preferences 페이지에서 Shared Repository 패널을 찾습니다. Directory 편집 상자에서 공유 레포지토리를 검색하려는 디렉토리를 입력합니다. 반드시 모든 팀원이 액세스할 수 있는 디렉토리를 지정합니다.

항목이 레포지토리에 처음으로 추가되면 delphi65dro 파일이 없는 경우 Kylix는 Shared Repository 디렉토리에 이 파일을 만듭니다.

참고 Object Repository 디렉토리(objrepos)에 대한 액세스 권한을 올바로 설정하는 것이 중요합니다. 왜냐하면 사용자가 디렉토리에 대한 쓰기 권한을 가지고 있지 않다면 디렉토리에 항목을 추가할 수 없기 때문입니다. 따라서 공통 Object Repository에 대한 액세스를 여러 사용자에게 허용하려면 그룹을 만들고 그룹의 멤버에게 objrepos 디렉토리에 대한 읽기 및 쓰기 액세스 권한을 주어야 합니다. 예를 들어 "dev" 그룹의 이름을 지정한다면 명령줄에서 다음 명령을 입력하여 사용 권한을 설정합니다.

```
cd <install directory>
chmod -R 775 objrepos
chgrp -R dev objrepos
```

자세한 내용은 *group(5)* man 페이지를 참조하십시오.

프로젝트에서 Object Repository 항목 사용

Object Repository의 항목에 액세스하려면 File | New를 선택합니다. New Items 대화 상자가 나타나 모든 사용 가능한 항목을 보여 줍니다. 사용하려는 항목 유형에 따라 프로젝트에 항목을 추가하는 다음 세 가지 옵션이 있습니다.

- Copy
- Inherit
- Use

항목 복사

Copy를 선택하여 선택한 항목의 정확한 복사본을 만들고 복사본을 프로젝트에 추가합니다. Object Repository에 있는 항목의 차후 변경 내용은 복사본에 반영되지 않으며 복사본의 변경 내용은 원래의 Object Repository 항목에 영향을 미치지 않습니다.

Copy는 프로젝트 템플릿에서 사용할 수 있는 유일한 옵션입니다.

항목 상속

Inherit를 선택하여 Object Repository의 선택한 항목에서 새 클래스를 파생시키고 이를 프로젝트에 추가합니다. 프로젝트를 다시 컴파일할 때 Object Repository에 있는 항목의 변경된 내용은 모두 프로젝트의 항목 변경 내용과 더불어 파생된 클래스에 반영됩니다. 파생된 클래스의 변경 내용은 Object Repository에 있는 공유 항목에 영향을 주지 않습니다.

Inherit는 폼, 대화 상자 및 데이터 모듈에 사용할 수 있지만 프로젝트 템플릿에는 사용할 수 없으며 동일한 프로젝트 내에서 항목을 재사용할 때 사용할 수 있는 *유일한* 옵션입니다.

항목 사용

선택한 항목 자체를 프로젝트의 일부가 되게 하려면 Use를 선택합니다. 프로젝트에서 항목의 변경된 내용은 Inherit 또는 Use 옵션으로 항목을 추가했던 다른 모든 프로젝트에 나타납니다. 이 옵션은 신중하게 선택하십시오.

Use 옵션은 폼, 대화 상자 및 데이터 모듈에 사용할 수 있습니다.

프로젝트 템플릿 사용

템플릿은 애플리케이션 개발의 첫 단계에서 사용할 수 있도록 미리 디자인된 프로젝트입니다. 다음과 같은 방법으로 템플릿을 사용하여 새 프로젝트를 만듭니다.

1 File | New를 선택하여 New Items 대화 상자를 표시합니다.
2 Projects 탭을 선택합니다.
3 원하는 프로젝트 템플릿을 선택하고 OK를 클릭합니다.
4 Select Directory 대화 상자에서 새 프로젝트 파일의 디렉토리를 지정합니다.

Kylix는 템플릿 파일을 지정된 디렉토리에 복사하며 복사한 후 수정할 수 있습니다. 프로젝트 템플릿 원본은 변경된 내용에 영향을 받지 않습니다.

공유 항목 수정

Object Repository에서 항목을 수정하면 변경된 내용은 Use 또는 Inherit 옵션으로 항목을 추가했던 기존 프로젝트뿐만 아니라 수정한 항목을 사용하는 향후의 모든 프로젝트에 영향을 미칩니다. 변경된 내용이 다른 프로젝트에 전파되는 것을 막으려면 다음 중 하나를 수행합니다.

- 항목을 복사하고 현재 프로젝트 내에서만 수정합니다.
- 항목을 현재 프로젝트로 복사하고 수정한 다음 다른 이름으로 바꾸어 Object Repository에 추가합니다.
- 항목으로부터 컴포넌트, 공유 객체, 컴포넌트 템플릿 또는 프레임을 만듭니다. 컴포넌트 또는 공유 객체를 만드는 경우에 다른 개발자와 공유할 수 있습니다.

기본 프로젝트, 새 폼 및 메인 폼 지정

기본적으로 File | New Application 또는 File | New Form을 선택하면 Kylix는 빈 폼을 표시합니다. 다음과 같이 레포지토리를 재구성하여 빈 폼을 다른 폼으로 변경할 수 있습니다.

1. Tools | Repository를 선택합니다.
2. 기본 프로젝트를 지정하려면 Projects 페이지를 선택하고 Objects 아래에 있는 항목을 선택합니다. 그런 다음 New Project 체크 박스를 선택합니다.
3. 기본 폼을 지정하려면 Forms와 같은 Repository 페이지를 선택한 다음 Objects에 있는 폼을 선택합니다. 새로운 기본 폼(File | New Form)을 지정하려면 New Form 체크 박스를 선택합니다. 새 프로젝트에 사용할 기본 메인 폼을 지정하려면 Main Form 체크 박스를 선택합니다.
4. OK를 클릭합니다.

컴포넌트와 컴포넌트 그룹의 재사용

Kylix에서는 CLX 컴포넌트로 수행했던 작업을 저장하고 재사용하는 여러 가지 방법을 제공합니다.

- 컴포넌트 템플릿은 컴포넌트 그룹을 구성하고 저장하기 위한 간단하고 빠른 방법을 제공합니다. 1-13페이지의 "컴포넌트 템플릿 생성 및 사용"을 참조하십시오.
- 폼, 데이터 모듈 및 프로젝트를 레포지토리에 저장할 수 있습니다. 레포지토리는 재사용 가능한 요소들의 집중식 데이터베이스를 제공하고 폼을 상속하여 변경 내용을 전파할 수 있습니다. 1-9페이지의 "코드 공유: Object Repository 사용"을 참조하십시오.
- 프레임을 컴포넌트 팔레트 또는 레포지토리에 저장할 수 있습니다. 프레임은 폼 상속을 사용하고 폼이나 다른 프레임에 포함(embedded)될 수 있습니다. *Kylix1 개발자 안내서* 6-11페이지의 "프레임 사용"을 참조하십시오.
- 사용자 지정 컴포넌트를 생성하는 것은 코드 재사용을 위한 가장 복잡한 방법이지만 가장 높은 유연성을 제공합니다. *Kylix1 개발자 안내서*의 24장 "컴포넌트 생성 개요"를 참조하십시오.

컴포넌트 템플릿 생성 및 사용

하나 이상의 컴포넌트로 구성된 템플릿을 생성할 수 있습니다. 컴포넌트를 폼에 정렬한 후 폼의 속성을 설정하고 컴포넌트 코드를 작성한 다음 *컴포넌트 템플릿*으로 저장합니다. 나중에 컴포넌트 팔레트에서 템플릿을 선택하여 미리 구성된 컴포넌트를 폼 위에 두면 모든 관련 속성과 이벤트 처리 코드가 프로젝트에 동시에 추가됩니다.

일단 템플릿을 폼에 두면 다른 작업에 각각의 컴포넌트를 둔 것처럼 컴포넌트의 위치를 재지정할 수 있고 속성을 재설정할 수 있으며 컴포넌트를 위한 이벤트 핸들러를 만들거나 수정할 수 있습니다.

다음과 같은 방법으로 컴포넌트 템플릿을 만듭니다

1. 폼에 컴포넌트를 두고 정렬합니다. Object Inspector에서 컴포넌트의 속성 및 이벤트를 설정합니다.
2. 컴포넌트를 선택합니다. 여러 컴포넌트를 선택하는 가장 쉬운 방법은 모든 컴포넌트 위로 마우스를 드래그하는 것입니다. 회색 핸들이 선택한 각 컴포넌트의 모서리에 나타납니다.
3. Component|Create Component Template을 선택합니다.
4. Component Name 편집 상자에서 컴포넌트 템플릿의 이름을 지정합니다. 기본적으로 2단계에서 선택한 첫 번째 컴포넌트의 컴포넌트 타입 다음에 "Template"이라는 단어가 오는 것이 좋습니다. 예를 들어 레이블을 선택한 후 편집 상자를 선택하면 이름은 "TLabelTemplate"이 됩니다. 이름을 변경할 수 있지만 기존 컴포넌트 이름과 중복되지 않도록 주의하십시오.
5. Palette Page 편집 상자에서 템플릿을 넣으려는 컴포넌트 팔레트 페이지를 지정합니다. 존재하지 않는 페이지를 지정하면 템플릿을 저장할 때 새 페이지가 생성됩니다.
6. Palette 아이콘 아래에서 팔레트의 템플릿을 나타내는 비트맵을 선택합니다. 기본적으로 2단계에서 선택한 첫 번째 컴포넌트의 컴포넌트 타입에서 사용하는 비트맵을 사용합니다. 다른 비트맵을 찾으려면 Change를 클릭합니다. 선택한 비트맵은 24 x 24 픽셀 이하여야 합니다.
7. OK를 클릭합니다.

컴포넌트 팔레트에서 템플릿을 제거하려면 Component|Configure Palette를 선택합니다.

CLX 애플리케이션에서 도움말 지원

CLX는 도움말을 직접 제공하지 않지만 구성 가능한 키를 눌러서 트리거된 도움말 요청이 여러 외부 도움말 뷰어(예: Man, Info, HyperHelp) 중 하나에 전달되도록 하는 수단을 제공합니다. 도움말을 지원하려면 애플리케이션 개발자는 *ICustomHelpViewer* 인터페이스(및 옵션으로 자손 인터페이스 중 하나)를 구현하는 클래스를 생성한 다음 이 클래스의 인스턴스를 전역 Help Manager에 등록해야 합니다.

Help Manager는 등록된 뷰어의 목록을 유지 관리하며 다음과 같이 두 단계로 뷰어에 요청을 전달합니다. 먼저 특정 도움말 키워드 또는 컨텍스트를 지원하는지 각각의 뷰어에게 물어 본 다음 지원이 가능한 해당 뷰어에 도움말 요청을 전달합니다. (Man과 Info 모두에 대해 등록된 뷰어가 있는 경우처럼 키워드를 지원하는 뷰어가 둘 이상이라면 애플리케이션에 Help Manager는 호출할 도움말 뷰어를 애플리케이션 사용자가 선택할 수 있도록 선택 상자를 나타냅니다. 그렇지 않으면 처음으로 응답한 도움말 시스템을 나타냅니다.)

도움말 시스템 인터페이스

CLX 도움말 시스템은 일련의 인터페이스를 통해 애플리케이션과 도움말 뷰어 간의 통신을 가능하도록 합니다. 이러한 인터페이스는 모두 HelpIntfs.pas에 정의되어 있으며 이를 사용하여 Help Manager를 구현합니다.

*ICustomHelpViewer*는 키워드 방식의 도움말을 지원하며 특정 뷰어에서 사용 가능한 모든 도움말에 대한 목차를 표시하는 기능을 지원합니다.

*IExtendedHelpViewer*는 숫자 도움말 컨텍스트를 기반으로 하는 도움말과 항목을 표시하도록 지원합니다. 대부분의 도움말 시스템에서 항목은 상위 수준의 키워드로 사용됩니다(예를 들어 "IntToStr"은 Kylix 도움말 시스템에서 키워드일 수 있지만 "String manipulation routines"는 항목 이름이 될 수 있습니다).

*ISpecialWinHelpViewer*는 Windows에서 실행하는 애플리케이션이 받을 수 있지만 쉽게 일반화할 수 없는 특화된 WinHelp 메시지에 응답하도록 지원합니다. 일반적으로 Windows 환경에서 실행되는 애플리케이션만이 이러한 인터페이스를 구현하게 되며 때로는 비표준 WinHelp 메시지를 광범위하게 사용하는 애플리케이션에만 필요한 인터페이스입니다.

*IHelpManager*는 애플리케이션의 Help Manager와 통신하고 추가 정보를 요청하는 도움말 뷰어용 메커니즘을 제공합니다. *IHelpManager*는 도움말 뷰어를 등록하면 생성됩니다.

*IHelpSystem*은 *TApplication*이 도움말 요청을 도움말 시스템으로 전달하는 메커니즘을 제공합니다. *TApplication*은 애플리케이션을 로드할 때 *IHelpSystem*과 *IHelpManager*를 모두 구현하는 객체의 인스턴스를 획득한 다음, 획득한 인스턴스를 속성으로 export합니다. 이렇게 하면 애플리케이션 내의 다른 코드에서 도움말을 직접 요청할 수 있습니다.

*IHelpSelector*는 둘 이상의 뷰어가 도움말 요청을 처리할 수 있을 경우에 도움말 시스템이 어떤 도움말 뷰어를 사용할지 묻고 목차를 표시할 수 있도록 사용자 인터페이스를 호출할 수 있는 메커니즘을 제공합니다. 이러한 표시 기능은 사용 중인 widget 집합이나 클래스 라이브러리와 상관없이 동일한 Help Manager 코드를 사용하기 위해 Help Manager로 직접 만들지 않습니다.

ICustomHelpViewer 구현

ICustomHelpViewer 인터페이스에는 세 가지 종류의 메소드가 있으며, 이 세 가지 메소드는 시스템 수준 정보(예를 들어 특정한 도움말 요청과 관계없는 정보)를 Help Manager와 통신하는 데 사용하는 메소드, Help Manager가 제공하는 키워드 방식의 도움말을 표시하는 메소드, 목차를 표시하는 메소드입니다.

Help Manager와 통신

*ICustomHelpViewer*는 시스템 정보를 Help Manager와 통신하는 데 사용할 수 있는 네 가지 함수를 제공합니다.

- *GetViewerName*
- *NotifyID*
- *ShutDown*
- *SoftShutDown*

Help Manager는 다음 환경에서 이 함수들을 사용하여 호출합니다.

- *ICustomHelpViewer.GetViewerName: String*은 Help Manager가 뷰어의 이름을 알려고 할 때(예를 들어 애플리케이션이 모든 등록된 뷰어의 목록을 표시하도록 요청받았을 경우) 호출됩니다. 이러한 정보는 문자열로 반환되고 논리적으로 정적 데이터여야 합니다(애플리케이션 실행 중에 변경할 수 없습니다). 멀티바이트 문자 집합은 지원되지 않습니다.
- *ICustomHelpViewer.NotifyID(const ViewerID: Integer)*는 뷰어를 식별하는 고유한 쿠키를 뷰어에게 제공하기 위하여 등록 **즉시** 호출됩니다. 이러한 정보는 나중에 사용하기 위해 저장되어야 하며 Help Manager의 공지에 대한 응답과 달리 뷰어가 스스로 종료하는 경우 Help Manager가 뷰어에 대한 모든 참조를 해제할 수 있도록 Help Manager에 쿠키 식별 기능을 제공해야 합니다. 쿠키를 제공하는 데 실패하거나 잘못된 쿠키를 제공하면 Help Manager는 잘못된 뷰어에 대한 참조를 잠재적으로 해제하게 됩니다.
- *ICustomHelpViewer.ShutDown*은 Help Manager에 의해 호출되어 도움말 뷰어에게 Manager가 종료 중이고 도움말 뷰어가 할당한 리소스가 해제되어야 한다는 것을 알립니다. 리소스 해제 시에는 이 메소드를 사용하는 것이 좋습니다.
- *ICustomHelpViewer.SoftShutDown*은 Help Manager에 의해 호출되어 뷰어를 언로드하지 않고 도움말 시스템의 외부에서 볼 수 있는 창(예를 들어 도움말 정보를 표시하는 창)을 닫도록 도움말 뷰어에게 요청합니다.

Help Manager에 정보 요청

도움말 뷰어는 *IHelpManager* 인터페이스를 통해 Help Manager와 통신하며 도움말 뷰어가 Help Manager에 등록될 때 인터페이스의 인스턴스는 도움말 뷰어에 반환됩니다. 도움말 뷰어는 *IHelpManager*를 통해 네 가지 사항에 대해 통신할 수 있습니다. 이러한 네 가지 사항에는 현재 활성 컨트롤의 창 핸들에 대한 요청, 현재 활성 컨트롤에 대한 도움말을 포함해야 한다고 Help Manager가 알고 있는 도움말 파일 이름에 대한 요

청, 도움말 파일의 경로에 대한 요청과 Help Manager의 요청이 아닌 다른 것에 대한 응답으로 도움말 뷰어가 자체 종료 중이라는 공지가 있습니다.

*IHelpManager.GetHandle: LongInt*는 현재 활성 컨트롤의 핸들을 알아야 하는 경우에 도움말 뷰어에 의해 호출되며 창 핸들이 반환됩니다.

*IHelpManager.GetHelpFile: String*은 현재 활성 컨트롤이 도움말을 포함하는 것으로 알고 있는 도움말 파일의 이름을 알고자 할 때 도움말 뷰어에 의해 호출됩니다.

*IHelpManager.Release*는 도움말 뷰어가 연결 해제 중이라는 것을 Help Manager에게 알리기 위해 호출됩니다. 이것은 *ICustomHelpViewer.ShutDown*을 통한 요청에 대한 응답으로는 *절대 호출되지 않고* 예상하지 못한 연결 해제를 Help Manager에게 알리는 데에만 사용됩니다.

키워드 방식 도움말 표시

도움말 요청은 *키워드 방식* 도움말로서 도움말 뷰어에 요청될 때 뷰어가 특정한 문자열 방식의 도움말을 제공하도록 요청되거나 *컨텍스트 방식* 도움말로서 요청될 때 뷰어가 특정한 숫자 식별자 방식의 도움말을 제공하도록 요청됩니다. 숫자 도움말 컨텍스트는 WinHelp 시스템을 사용하는 Windows에서 실행 중인 애플리케이션에서 도움말 요청의 기본 형태입니다. 대부분의 Linux 도움말 시스템에서는 숫자 도움말 컨텍스트를 인식하지 못하기 때문에 CLX 애플리케이션에서 사용하지 않는 것이 좋습니다. *ICustomHelpViewer* 구현은 키워드 방식의 도움말 요청을 지원하는 데 필요하고 *IExtendedHelpViewer* 구현은 컨텍스트 방식의 도움말 요청을 지원하는 데 필요합니다.

*ICustomHelpViewer*는 키워드 방식의 도움말을 처리하는 세 가지 메소드를 제공합니다.

- *UnderstandsKeyword*
- *GetHelpStrings*
- *ShowHelp*

```
ICustomHelpViewer.UnderstandsKeyword(const HelpString: String): Integer
```

이 구문은 Help Manager에 의해 호출되는 세 가지 메소드 중에서 첫 번째로서 뷰어가 해당 문자열에 대한 도움말을 제공하는지 묻기 위해 동일한 문자열을 사용하여 *각각의* 등록된 도움말 뷰어를 호출하는데 여기서 뷰어는 도움말 요청에 응답하여 표시할 수 있는 도움말 페이지의 수를 나타내는 정수로 응답할 것으로 예상됩니다. 뷰어는 원하는 메소드를 사용하여 이를 결정할 수 있습니다. Kylix IDE에서 HyperHelp 뷰어는 자체 인덱스를 유지 관리하고 인덱스를 찾는 반면 Man 페이지 뷰어는 man 프로그램을 호출하여 인덱스를 찾습니다. 뷰어가 이 키워드에 대한 도움말을 지원하지 않으면 뷰어는 0을 반환합니다. 현재 음수는 0을 의미하는 것으로 해석되지만 차후 릴리스에서는 변경될지도 모릅니다.

```
ICustomHelpViewer.GetHelpStrings(const HelpString: String): TStringList
```

이 구문은 둘 이상의 뷰어가 항목에 대한 도움말을 제공할 수 있는 경우에 Help Manager에 의해 호출됩니다. 뷰어는 *TStringList*를 반환할 것입니다. 반환된 목록에 있는 문자열은 그 키워드에 사용할 수 있는 페이지에 매핑되어야 하지만 매핑의 특성은 뷰어로 결정될 수 있습니다. HyperHelp 뷰어의 경우에 문자열 목록은 항상 정확히 하

나의 항목(entry)을 포함하지만(HyperHelp는 자체 인덱싱을 제공하며 그 외에는 pointless 복제를 제공함) Man 페이지 뷰어의 경우에 문자열 목록은 여러 개의 문자열들로 구성되며, 하나의 문자열은 해당 키워드에 대한 페이지가 포함된 설명서의 한 섹션에 해당됩니다.

```
ICustomHelpViewer.ShowHelp(const HelpString: String)
```

이 구문은 도움말 뷰어가 특정한 키워드에 대한 도움말을 표시해야 하는 경우에 Help Manager에 의해 호출됩니다. 이것은 이 작업의 마지막 메소드 호출인데 언제나 *UnderstandsKeyword*가 먼저 호출된 다음에 호출됩니다.

목차 표시

*ICustomHelpViewer*는 목차 표시와 관련된 두 가지 메소드를 제공합니다.

- *CanShowTableOfContents*
- *ShowTableOfContents*

이 작업의 이론은 다음 키워드 도움말 요청 함수의 작업과 유사합니다. 즉 Help Manager는 먼저 *ICustomHelpViewer.CanShowTableOfContents: Boolean*을 호출하여 모든 도움말 뷰어를 쿼리한 다음 *ICustomHelpViewer.ShowTableOfContents*를 호출하여 특정 도움말 뷰어를 호출합니다.

특정 뷰어는 목차 지원 요청을 거부할 수도 있습니다. 예를 들어 목차의 개념이 Man 페이지의 작동 방식에 제대로 매핑되지 않기 때문에 이런 현상이 일어날 수도 있습니다. 이와 반대로 HyperHelp 뷰어는 목차를 표시하는 요청을 직접 HyperHelp에 전달함으로써 목차를 지원합니다. 그러나 *ICustomHelpViewer*의 구현이 *CanShowTableOfContents*를 통한 쿼리에 *true*로 응답하고 *ShowTableOfContents*를 통한 요청을 무시하는 것은 타당하지 *않습니다*.

IExtendedHelpViewer 구현

*ICustomHelpViewer*는 키워드 방식 도움말에 대한 직접적인 지원만 제공합니다. 일부 도움말 시스템(특히 WinHelp)은 애플리케이션에 보이지 않는 내부적인 방식으로 *컨텍스트 ID*로 알려진 숫자를 키워드에 연결시켜서 작동합니다. 이 도움말 시스템에서는 애플리케이션이 문자열이 아닌 컨텍스트를 사용하여 도움말 시스템을 호출하는 컨텍스트 방식 도움말을 지원해야 하고 도움말 시스템은 번호 자체를 번역합니다.

CLX로 작성된 애플리케이션은 *ICustomHelpViewer*를 구현하는 객체를 *IExtendedHelpViewer*를 구현하는 객체로 확장함으로써 컨텍스트 방식 도움말을 요구하는 시스템과 통신할 수 있습니다. 또한 *IExtendedHelpViewer*는 도움말 시스템과의 통신을 지원하여 키워드 검색을 사용하는 대신 상위 수준의 항목으로 직접 이동할 수 있도록 합니다.

*IExtendedHelpViewer*는 네 가지 함수를 사용합니다. 네 가지 함수 중에서 *UnderstandsContext*와 *DisplayHelpByContext*는 컨텍스트 방식 도움말을 지원하는 데 사용되고 나머지 *UnderstandsTopic*과 *DisplayTopic*은 항목을 지원하는 데 사용됩니다.

애플리케이션 사용자가 F1 키를 누르면 Help Manager는 다음을 호출합니다.

```
IExtendedHelpViewer.UnderstandsContext(const ContextID: Integer;
const HelpFileName: String): Boolean
```

그리고 현재 활성화된 컨트롤은 키워드 방식 도움말이 아닌 컨텍스트 방식 도움말을 지원합니다. *ICustomHelpViewer.UnderstandsKeyword*에서와 같이 Help Manager는 등록된 모든 도움말 뷰어를 반복하여 쿼리합니다. 그러나 *ICustomHelpViewer.UnderstandsKeyword*의 경우와 달리 지정된 컨텍스트를 둘 이상의 뷰어가 지원하면 주어진 컨텍스트를 지원하는 *첫 번째* 등록된 뷰어가 호출됩니다.

다음 구문은

```
IExtendedHelpViewer.DisplayHelpByContext(const ContextID: Integer;
const HelpFileName: String)
```

Help Manager는 등록된 도움말 뷰어를 폴링(polling)한 후에 호출합니다.

항목 지원 함수는 동일한 방식으로 작동합니다.

```
IExtendedHelpViewer.UnderstandsTopic(const Topic: String): Boolean
```

이 구문은 항목을 지원하는지 묻는 도움말 뷰어를 폴링(polling)하는 데 사용되고

```
IExtendedHelpViewer.DisplayTopic(const Topic: String)
```

이 구문은 항목에 대한 도움말을 제공할 수 있다고 보고하는 첫 번째 등록된 뷰어를 호출하는 데 사용됩니다.

IHelpSelector 구현

*IHelpSelector*는 *ICustomHelpViewer*와 짝을 이룹니다. 둘 이상의 등록된 뷰어가 입력된 키워드, 컨텍스트 또는 항목에 대한 지원을 제공하거나 목차를 제공하겠다고 요청하면 Help Manager는 이들 중에서 선택해야 합니다. 컨텍스트 또는 항목의 경우에 *언제나* Help Manager는 지원을 제공하겠다고 요청하는 첫 번째 도움말 뷰어를 선택합니다. 키워드나 목차의 경우에 Help Manager는 기본적으로 첫 번째 도움말 뷰어를 선택합니다. 뷰어 선택은 애플리케이션에서 조정할 수 있습니다.

Help Manager의 선택을 무시하고 애플리케이션에서 뷰어를 선택하려면 애플리케이션은 *IHelpSelector* 인터페이스의 구현을 제공하는 클래스를 등록해야 합니다. *IHelpSelector*는 *SelectKeyword*와 *TableOfContents*의 두 가지 함수를 export합니다. 두 함수는 가능한 키워드 일치 사항이나 목차 제공을 요청하는 뷰어의 이름을 포함하는 *TStrings*를 하나씩 인수로 사용합니다. 구현 프로그램은 선택한 문자열을 나타내는 *TStrings*에서 인덱스를 반환해야 합니다.

참고 문자열이 재정렬되면 Help Manager에서 혼란스러울 수 있으므로 *IHelpSelector*의 구현 프로그램에서 문자열을 재정렬하지 않는 것이 좋습니다. 도움말 시스템은 오직 *하나의* HelpSelector만 지원하므로 새 선택기(selector)가 등록되면 이전에 사용한 기존 선택기는 연결이 끊어집니다.

도움말 시스템 객체 등록

Help Manager가 객체들과 통신하려면 *ICustomHelpViewer*, *IExtendedHelpViewer*, *ISpecialWinHelpViewer* 및 *IHelpSelector*를 구현하는 객체들을 Help Manager에 등록해야 합니다.

도움말 시스템 객체들을 Help Manager에 등록하려면 다음 작업을 수행해야 합니다.

- 도움말 뷰어 등록
- 도움말 선택기 등록

도움말 뷰어 등록

객체 구현을 포함하는 유닛은 HelpIntfs를 사용해야 합니다. 객체의 인스턴스는 구현 유닛의 **var** 섹션에서 선언해야 합니다.

구현 유닛의 초기화 섹션은 인스턴스 변수를 할당하여 *RegisterViewer* 함수로 전달해야 합니다. *RegisterViewer*는 *ICustomHelpViewer*를 인수로 사용하고 *IHelpManager*를 반환하는 HelpIntfs.pas로 export한 평면 함수(flat function)입니다. *IHelpManager*는 차후 사용을 위해 저장해야 합니다.

도움말 선택기 등록

객체 구현을 포함하는 유닛은 HelpIntfs와 QForms를 사용해야 합니다. 객체의 인스턴스는 구현 유닛의 **var** 섹션에서 선언해야 합니다.

구현 유닛의 초기화 섹션은 다음과 같은 전역 Application 객체의 *HelpSystem* 속성을 통해 도움말 선택기를 등록해야 합니다.

```
Application.HelpSystem.AssignHelpSelector(myHelpSelectorInstance)
```

이 프로시저는 값을 반환하지 않습니다.

CLX 애플리케이션에서 도움말 사용

다음 단원에서는 CLX 애플리케이션 내에서 도움말을 사용하는 방법에 대해 설명합니다.

- TApplication의 도움말 처리 방법
- 컨트롤의 도움말 처리 방법
- 도움말 시스템 직접 호출
- IHelpSystem 사용

TApplication의 도움말 처리 방법

*TApplication*은 애플리케이션 코드로부터 액세스할 수 있는 다음과 같은 두 가지 메소드를 제공합니다.

- *ContextHelp*는 컨텍스트 방식 도움말 요청을 사용하여 도움말 시스템을 호출합니다.
- *KeywordHelp*는 키워드 방식 도움말 요청을 사용하여 도움말 시스템을 호출합니다.

이 두 함수는 전달(pass)되는 컨텍스트 또는 키워드를 인수로 사용하고 도움말 시스템을 나타내는 *TApplication*의 데이터 멤버를 통해 요청을 전달(forward)합니다. 이 데이터 멤버는 읽기 전용 속성인 *HelpSystem*을 통해 직접 액세스할 수 있습니다.

컨트롤의 도움말 처리 방법

*TControl*에서 파생되는 모든 컨트롤은 도움말 시스템이 사용하는 *HelpType*, *HelpFile*, *HelpContext* 및 *HelpKeyword*라는 네 가지 속성을 나타냅니다. *HelpFile*은 컨트롤의 도움말이 있는 파일의 이름을 포함하게 되며 도움말이 파일 이름과 상관없는 외부 도움말 시스템(예를 들어 Man 페이지 시스템)에 있는 경우 속성은 비어 있어야 합니다.

HelpType 속성은 컨트롤의 디자이너가 키워드 방식 도움말 또는 컨텍스트 방식 도움말 중에서 어떤 것을 예상하는지 결정하는 열거 타입의 인스턴스를 포함합니다. 나머지 두 가지 속성은 여기에 연결됩니다. *HelpType*이 *htKeyword*로 설정될 경우 도움말 시스템은 컨트롤이 키워드 방식 도움말을 사용하는 것으로 예상하고 *HelpKeyword* 속성의 내용만 봅니다. 이와 반대로 *HelpType*이 *htContext*로 설정될 경우 도움말 시스템은 컨트롤이 컨텍스트 방식 도움말을 사용하는 것으로 예상하고 *HelpContext* 속성의 내용만 봅니다.

속성 이외에 컨트롤은 단일 메소드인 *InvokeHelp*를 보여 주는데 이 메소드는 요청을 도움말 시스템으로 전달하기 위해 호출될 수 있습니다. 이 메소드는 매개변수를 사용하지 않고 전역 Application 객체에서 컨트롤이 지원하는 도움말의 종류에 해당하는 메소드를 호출합니다.

*TWidgetControl*의 *KeyDown* 메소드가 *InvokeHelp*를 호출하기 때문에 F1 키를 누르면 도움말 메시지가 자동으로 호출됩니다.

도움말 시스템 직접 호출

이 메커니즘에 의해 나타나지 않은 추가적인 도움말 시스템 기능의 경우 *TApplication*은 도움말 시스템에 직접 액세스할 수 있는 읽기 전용 속성을 제공합니다. 이 속성은 인터페이스 *IHelpSystem* 구현의 인스턴스입니다. *IHelpSystem*과 *IHelpManager*는 동일한 객체에 의해 구현되지만 어떤 인터페이스는 애플리케이션이 Help Manager와 통신하는 데 사용되고 어떤 인터페이스는 도움말 뷰어가 Help Manager와 통신하는 데 사용됩니다.

IHelpSystem 사용

*IHelpSystem*을 통해 애플리케이션은 다음 세 가지를 수행할 수 있습니다.

- Help Manager에 경로 정보 제공
- 새 도움말 선택기 제공
- Help Manager가 도움말을 표시하도록 요청

Help Manager는 플랫폼 독립형 및 도움말 시스템 독립형이므로 도움말 파일의 위치를 확인할 수 없기 때문에 경로 정보를 제공하는 것은 중요합니다. 애플리케이션에서 파일 위치 자체를 확인할 수 없는 외부 도움말 시스템이 도움말을 제공한다고 예상하는 경우

*IHelpSystem.ProvideHelpPath*를 통해 이러한 정보를 제공하여 *IHelpManager.GetHelpPath*를 통해 정보를 사용 가능하도록 만듭니다. 이러한 정보는 도움말 뷰어가 요청하는 경우에만 외부로 전파(propagate)됩니다.

여러 외부 도움말 시스템에서 동일한 키워드에 대한 도움말을 제공할 수 있는 경우에는 도움말 선택기 할당을 통해 Help Manager는 의사 결정을 위임(delegate)할 수 있습니다. 자세한 내용은 1-18페이지의 "IHelpSelector 구현"의 단원을 참조하십시오.

*IHelpSystem*은 Help Manager에게 도움말을 표시하도록 요청하기 위해 다음 네 가지 프로시저와 한 가지 함수를 export합니다.

- *ShowHelp*
- *ShowContextHelp*
- *ShowTopicHelp*
- *ShowTableOfContents*
- *Hook*

*Hook*은 전적으로 WinHelp 호환성을 위해 고안되었으므로 CLX 애플리케이션에서 사용해서는 안 됩니다. Hook을 통해 키워드 방식, 컨텍스트 방식 또는 항목 방식 도움말에 대한 요청에 직접적으로 매핑될 수 없는 WM_HELP 메시지를 처리할 수 있습니다. 나머지 각 메소드는 키워드, 컨텍스트 ID를 인수로 사용하거나 도움말에 요청하는 항목과 도움말을 찾게 되리라고 예상되는 도움말 파일을 사용합니다.

일반적으로 항목 방식 도움말을 요청하지 않는 경우 컨트롤의 *InvokeHelp* 메소드를 통해 도움말 요청을 Help Manager에 전달하는 것이 효과적이고 더욱 확실합니다.

IDE 도움말 시스템 사용자 지정

Kylix IDE는 CLX 애플리케이션과 동일한 방식으로 다양한 도움말 뷰어를 지원하고, 도움말 요청을 Help Manager에 위임하여 등록된 도움말 뷰어에 전달(forward)합니다. IDE는 두 가지 도움말 뷰어를 설치하여 출시됩니다. 그 중 하나는 HyperHelp 뷰어로서 이를 통해 Kylix 도움말 파일을 보는 외부 WinHelp 에뮬레이터인 HyperHelp로 도움말 요청을 전달하고, 다른 하나는 Man 페이지 뷰어로서 이를 통해 대부분의 Unix 시스템에 설치된 Man 시스템에 액세스할 수 있습니다. HyperHelp 뷰어는 Kylix 도움말이 작동하는 데 필요하기 때문에 제거할 수 없습니다. Man 페이지 뷰어는 뷰어의 소스를 예제 디렉토리에서 사용할 수 있는 별도의 패키지로 출시됩니다.

IDE에서의 새 도움말 뷰어 설치 방법은 한 가지를 제외하고 CLX 애플리케이션에서의 설치 방법과 동일합니다. *ICustomHelpViewer*(원하는 경우 *IExtendedHelpViewer*)를 구현하는 객체를 작성하여 도움말 요청을 자신이 선택한 외부 뷰어에 전달하고 *ICustomHelpViewer*를 IDE에 등록합니다.

사용자 지정 도움말 뷰어를 IDE에 등록하려면 다음과 같이 합니다.

1. 도움말 뷰어를 구현하는 유닛이 HelpIntfs.pas를 포함하는지 확인합니다.
2. 유닛을 IDE에 등록된 디자인 타임 패키지로 만들고 런타임 패키지를 선택하여 패키지를 만듭니다. 이것은 유닛이 사용하는 Help Manager 인스턴스가 IDE가 사용하는 Help Manager 인스턴스와 동일하다는 것을 확인하는 데 필요합니다.
3. 도움말 뷰어가 유닛 내에서 전역 인스턴스로 존재하는지 확인합니다.
4. 유닛의 초기화 섹션에서 인스턴스가 *RegisterHelpViewer* 함수로 전달되는지 확인합니다.

2

크로스 플랫폼 애플리케이션 개발

* 이 장은 영문 Kylix2 개발자 안내서의 10장입니다.

Linux 및 Windows 운영 체제에서 모두 실행되는 32비트 크로스 플랫폼 애플리케이션을 개발할 수 있습니다. 기존 Windows 애플리케이션을 크로스 플랫폼 애플리케이션으로 수정하거나 플랫폼 독립적인 코드 작성 권장안대로 새 애플리케이션을 만들 수 있습니다.

이 장에서는 Delphi 또는 CLX Windows 애플리케이션을 Linux로 포팅하는 방법을 설명하고 Windows와 Linux에서 애플리케이션을 개발할 때의 차이점에 관한 정보도 제공합니다. 두 운영 체제 간에 이식 가능한 코드를 작성하는 지침도 제공합니다.

참고 Linux용으로 개발된 대부분의 애플리케이션(Linux 특정 API 호출이 없는 애플리케이션)은 Linux에서 실행되며 다시 컴파일하면 Windows에서(CLX를 사용할 수 있으면) 실행됩니다.

Windows 애플리케이션을 Linux로 포팅

Windows 환경용으로 개발된 Delphi 또는 CLX 애플리케이션이 있는 경우 이 애플리케이션을 Linux 환경으로 포팅할 수 있습니다. 포팅의 난이도는 애플리케이션의 특성 및 복잡성, Windows 종속 정도에 따라 다릅니다.

다음 단원에서는 Windows와 Linux 환경 간의 주요한 차이점을 설명하고 애플리케이션 포팅 입문에 대한 지침을 제공합니다.

포팅 기법

다음은 애플리케이션을 한 플랫폼에서 다른 플랫폼으로 포팅하기 위해 이용할 수 있는 여러 가지 방법입니다.

표 2.1 포팅 기법

기법	설명
플랫폼별 포팅	운영 체제와 기본으로 사용한 API를 대상으로 함
크로스 플랫폼 포팅	크로스 플랫폼 API를 대상으로 함
Windows 에뮬레이션	코드는 그대로 두고 코드에서 사용하는 API를 포팅함

플랫폼별 포팅

플랫폼별 포팅은 시간이 걸리고 비용이 많이 들며 주로 해당 플랫폼에서만 애플리케이션을 사용할 수 있습니다. 플랫폼별 포팅은 코드 베이스가 달라서 유지 관리가 어렵습니다. 하지만 플랫폼별 포팅은 특정 운영 체제를 위해 설계된 것으로 플랫폼 특정 기능을 십분 활용할 수 있습니다. 따라서 애플리케이션이 더 빨리 실행됩니다.

크로스 플랫폼 포팅

크로스 플랫폼 포팅은 일반적으로 가장 신속한 기법을 제공하며 포팅된 애플리케이션은 다중 플랫폼을 대상으로 합니다. 실제로 크로스 플랫폼 애플리케이션 개발에 관련된 작업의 양은 기존 코드에 따라 결정됩니다. 코드가 플랫폼 독립성을 염두에 두지 않고 개발되었다면 플랫폼 독립적인 "로직"과 플랫폼 독립적인 "구현"이 같이 섞여 있을 수 있습니다.

크로스 플랫폼 접근법은 비즈니스 로직을 플랫폼 독립적인 관점에서 개발하므로 더 많이 사용됩니다. 일부 서비스는 플랫폼마다 고유하게 구현되어 있지만 겉으로는 유사해 보이는 내부 인터페이스를 통해 추출됩니다. 한 가지 예가 Kylix의 런타임 라이브러리입니다. 인터페이스가 두 플랫폼에서 매우 유사하지만 구현 내용은 매우 다릅니다. 크로스 플랫폼 부분을 분리한 후 그 위에 특정 서비스를 구현해야 합니다. 결국 이 접근법은 소스를 공유하고 애플리케이션 아키텍처를 개선하여 유지 비용을 줄이는 방식이며 가장 비용이 적게 드는 해결책입니다.

Windows 에뮬레이션 포팅

Windows 에뮬레이션은 가장 복잡한 방법이며 비용이 많이 들 수 있지만 최종의 Linux 애플리케이션이 기존 Windows 애플리케이션과 가장 유사하게 포팅됩니다. 이 접근법은 Linux에 Windows 기능을 구현하는 것을 포함합니다. 엔지니어링의 관점에서 볼 때 이 방법은 유지 관리가 가장 어렵습니다.

애플리케이션 포팅

Linux에서만 실행시키기 위해 애플리케이션을 Linux로 포팅하는 경우 Windows 특정 기능 전체를 제거할 수 있습니다. 하지만 Windows와 Linux에서 모두 실행하기 위해 애플리케이션을 포팅하는 경우 코드를 수정하거나 Windows 또는 Linux에만 적용되는 코드의 섹션을 **$IFDEF**로 표시합니다.

Windows 애플리케이션을 Linux로 포팅하려면 다음과 같은 일반적인 절차를 따릅니다.

1 Delphi 또는 CLX Windows 애플리케이션 소스 파일과 기타 프로젝트 관련 파일을 Linux 컴퓨터로 옮깁니다. 프로그램을 Linux와 Windows 플랫폼에서 모두 실행할 경우 Linux와 Windows 간에 소스 파일을 공유할 수 있습니다. ASCII 모드를 사용하는 ftp와 같은 툴을 사용하여 파일을 전송해도 됩니다.

   소스 파일에는 유닛 파일(.pas 파일), 프로젝트 파일(.dpr 파일) 및 패키지 파일(.dpk 파일)이 포함됩니다. 프로젝트 관련 파일에는 폼 파일(.dfm 파일), 리소스 파일(.res 파일) 및 프로젝트 옵션 파일(.dof 파일)이 포함됩니다. IDE를 사용하지 않고 명령줄에서만 애플리케이션을 컴파일하려면 구성 파일(.cfg 파일)이 필요합니다.

2 단일 소스 애플리케이션을 Windows와 Linux에서 모두 사용하려는 경우 .dfm 파일을 동일한 이름의 .xfm 파일로 복사합니다(예를 들어 unit1.dfm을 unit1.xfm으로 이름 변경). 유닛 파일에서 .dfm 파일에 대한 참조를 {$R *.dfm}에서 {$R *.xfm}으로 이름 변경(또는 **$IFDEF**)합니다. (.dfm 파일은 Kylix에서 작동하지만 Delphi에서는 작동하지 않도록 변경됩니다.)

3 모든 **uses** 절을 변경(또는 **$IFDEF**)하여 Kylix의 정확한 유닛을 참조하게 합니다. (자세한 내용은 2-8페이지의 "CLX와 VCL 유닛" 참조)

4 Windows 종속성이 필요 없는 모든 코드를 런타임 라이브러리 루틴과 상수를 사용하여 다시 작성함으로써 더욱 플랫폼 독립적인 코드로 만듭니다. (자세한 내용은 2-14페이지의 "이식 가능한 코드 작성" 참조)

5 Linux에서 다르게 작동하는 기능에 대해 동등한 기능을 찾습니다. **$IFDEF**를 가능한 적게 사용하여 Windows 특정 정보를 구분합니다. (자세한 내용은 2-15페이지의 "조건 지시어 사용" 참조)

   예를 들어 소스 파일에서 다음과 같이 플랫폼 특정 코드에 **$IFDEF**를 사용할 수 있습니다.

```
{$IFDEF MSWINDOWS}
IniFile.LoadfromFile('c:\x.txt');
{$ENDIF}

{$IFDEF LINUX}
IniFile.LoadfromFile('/home/name/x.txt');
{$ENDIF}
```

6 모든 프로젝트 파일에서 경로에 대한 참조를 검색합니다.

   - Linux의 경로명은 슬래시(/)를 구분자로 사용하며(예를 들어 /usr/lib) Linux 시스템에서는 파일들이 서로 다른 디렉토리에 위치할 수 있습니다. SysUtils에서 PathDelim 상수를 사용하여 시스템에 적절한 경로 구분자를 지정합니다. Linux 상의 파일에 대한 정확한 위치를 지정합니다.
   - 드라이브 문자에 대한 참조(예를 들어 C:\)와 문자열에서 두 번째 위치에 있는 콜론을 찾아서 드라이브 문자를 검색하는 코드를 변경합니다. SysUtils에서 DriveDelim 상수를 사용하여 시스템에 적절한 위치를 지정합니다.
   - 다중 경로를 지정하는 곳에서는 경로 분리자를 세미콜론(;)에서 콜론(:)으로 변경합니다. SysUtils에서 PathSep 상수를 사용하여 시스템에 적절한 경로 분리자를 지정합니다.

- Linux에서는 파일 이름의 대소문자를 구별하므로 애플리케이션에서 파일 이름의 대소문자를 변경하거나 특정 대소문자를 추측하여 사용하지 않도록 합니다.

7 Linux에서 프로젝트를 컴파일합니다. 추가로 변경해야 할 곳이 없는지 알아보기 위해서 모든 오류 메시지를 검토합니다.

CLX와 VCL 비교

Kylix는 비주얼 컴포넌트 라이브러리(VCL) 대신 Borland 크로스 플랫폼용 컴포넌트 라이브러리(CLX)를 사용합니다. VCL의 여러 컨트롤에서 Windows 컨트롤에 액세스하기 위한 간단한 방법을 제공합니다. 이와 마찬가지로 CLX는 Qt 공유 라이브러리에서 Qt widget("widget"은 window와 gadget의 합성어)에 대한 액세스를 제공합니다. Kylix에서 컴포넌트 라이브러리는 그래픽이나 GUI 요소로 Win32 대신 Qt를 호출합니다.

CLX는 Delphi의 VCL과 매우 유사합니다. 컴포넌트 이름이 대부분 동일하며 속성 이름 역시 대부분 동일합니다. VCL과 CLX 모두 Delphi에서 사용할 수 있습니다.

CLX 컴포넌트는 다음과 같은 부분으로 나뉩니다.

표 2.2 CLX 부분

부분	설명
VisualCLX	원시 크로스 플랫폼 GUI 컴포넌트와 그래픽. 이 부분의 컴포넌트는 Linux와 Windows에서 서로 다를 수 있습니다. Qt는 라이브러리의 VisualCLX 서브셋이 호출하는 widget 집합입니다.
DataCLX	클라이언트 데이터 액세스 컴포넌트. 이 부분의 컴포넌트는 클라이언트 데이터셋에 기반한 로컬, 클라이언트/서버 및 n계층의 서브셋입니다. Linux와 Windows에서 동일합니다. (이 부분의 컴포넌트는 일부 Kylix 에디션에서 사용할 수 없습니다.)
NetCLX	Apache DSO와 CGI Web Broker를 비롯한 인터넷 컴포넌트. Linux와 Windows에서 동일합니다. (이 부분의 컴포넌트는 일부 Kylix 에디션에서 사용할 수 없습니다.)
BaseCLX	Classes.pas를 구현하고 포함하는 런타임 라이브러리. 코드는 Linux와 Windows에서 동일합니다.

VisualCLX의 widget은 Windows 컨트롤을 대체합니다. CLX에서 *TWidgetControl*은 VCL의 *TWinControl*을 대체합니다. *TScrollingWidget*과 같은 나머지 다른 컴포넌트는 대응하는 이름을 가지고 있습니다. *TWinControl*을 *TWidgetControl*로 변경할 필요는 없습니다. 예를 들어 다음과 같은 타입 선언의 경우

```
TWinControl = TWidgetControl;
```

QControls.pas 소스 파일에 이 구문을 사용하면 간단하게 소스 코드를 공유할 수 있습니다. *TWidgetControl*과 자손들은 모두 Qt 객체의 참조인 *Handle* 속성과 이벤트 메커니즘을 처리하는 Hook 객체의 참조인 *Hooks* 속성을 갖습니다.

CLX에서는 일부 클래스의 유닛 이름과 위치가 다릅니다. Kylix에는 존재하지 않는 유닛에 대한 참조를 제거하고 이름을 Kylix 유닛으로 변경하려면 **uses** 절을 수정해야 합니다. 프로젝트 파일과 유닛의 인터페이스 섹션에는 대부분 **uses** 절이 있습니다. 유닛의 구현 섹션에도 자체 **uses** 절이 있습니다.

VCL과 CLX의 차이점

대체로 CLX와 VCL은 일관적으로 구현되지만 일부 기능은 다르게 구현됩니다. 이 단원에서는 CLX와 VCL 구현 간의 일부 차이점에 대한 개요를 제공합니다.

룩앤필(Look and feel)

Linux의 비주얼 환경은 외관이 Windows와 약간 다릅니다. 어떤 윈도우 관리자(예를 들어 KDE나 Gnome)를 사용하느냐에 따라 대화 상자의 모양이 다를 수 있습니다.

스타일

애플리케이션 전체의 "스타일"을 *OwnerDraw* 속성에 추가하여 사용할 수 있습니다. *TApplication.Style* 속성을 사용하여 애플리케이션 그래픽 요소의 룩앤필을 지정할 수 있습니다. 스타일을 사용하여 widget이나 애플리케이션의 외관을 완전히 바꿀 수 있습니다. Linux에서도 owner-draw를 사용할 수 있지만 스타일을 사용하는 것이 좋습니다.

Variants

시스템에 있던 모든 가변 타입 배열/SafeArray 코드는 다음 두 개의 유닛에 있습니다.

- Variants.pas
- VarUtils.pas

운영 체제 의존 코드는 이제 VarUtils.pas 파일에 분리되어 있고 Variants.pas에서 필요한 모든 일반 버전도 포함합니다. Windows에서 Windows 호출이 포함된 코드를 포팅하는 경우 Windows 호출을 VarUtils.pas 호출로 대체해야 합니다.

가변 타입을 사용하려면 **uses** 절에서 Variants 유닛을 포함해야 합니다.

*VarIsEmpty*는 *varEmpty*에 대한 간단한 테스트로 가변 타입이 지워졌는지 확인하고 Linux에서는 *VarIsClear* 함수를 사용하여 가변 값이 정의되지 않았는지 확인합니다.

사용자 지정 가변 타입 데이터 핸들러

가변 타입에 대해 사용자 지정 데이터 타입을 정의하여 타입이 가변으로 할당될 때 연산자를 오버로드할 수 있습니다. 새 가변 타입을 만들려면 *TCustomVariantType* 클래스의 자손으로 새로운 가변 타입을 인스턴스화합니다.

예를 들어 VarCmplx.pas를 보면 이 유닛은 사용자 지정 가변 타입을 통해 복잡한 수학적 지원을 구현합니다. 이 유닛은 더하기, 빼기, 곱하기, 나누기(정수 나누기 제외), 부정과 같은 가변 연산을 지원합니다. 또한 SmallInt, Integer, Single, Double, Currency, Date, Boolean, Byte, OleStr 및 String의 변환을 처리합니다. 모든 실수 타입/순서 타입 변환에서는 복잡한 값의 허수 부분을 버립니다.

레지스트리 사용 안 함

Linux에서는 구성 정보를 저장하는 레지스트리는 사용하지 않습니다. 레지스트리 대신 텍스트 구성 파일과 환경 변수를 사용합니다. Linux의 시스템 구성 파일은 /etc/hosts 처럼 /etc 디렉토리에 위치하기도 합니다. 기타 사용자 프로파일은 .bashrc(bash 셸 설정) 또는 .XDefaults(X 프로그램의 기본값을 설정)와 같이 점(.)이 앞에 오는 숨김 파일에 위치합니다.

애플리케이션과 동일한 디렉토리에 저장하는 방법 대신 로컬 구성 텍스트 파일을 사용하도록 레지스트리 의존 코드를 변경할 수 있습니다. 모든 레지스트리 함수를 사용해서 유닛을 작성하고 모든 결과를 로컬 구성 파일에 저장되도록 하여 레지스트리 의존성을 관리할 수 있습니다.

전역 위치에 정보를 입력하기 위해 루트 디렉토리에 전역 구성 파일을 저장할 수 있습니다. 이렇게 하면 사용자의 모든 애플리케이션이 동일한 구성 파일에 액세스할 수 있습니다. 그 대신 파일 권한과 액세스 권한이 올바르게 설정되었는지 확인해야 합니다.

Windows 와 마찬가지로 ini 파일도 사용할 수 있습니다. 그러나 Kylix 에서는 *TRegIniFile* 대신 *TMemIniFile*을 사용해야 합니다.

메소드 공유

랭귀지 간에 메소드를 공유하려면 공통 애플리케이션 바이너리 인터페이스(ABI)를 사용해야 합니다. Kylix 가상 메소드 테이블(VMT)은 현재 제안된 UNIX ABI 사양을 따릅니다.

C++ 객체에 대해 VMT 호환 gcc를 생성하려면 gcc 버전 2.95 이상을 사용하여 C++ 클래스 선언에 COM_INTERFACE 속성을 지정해야 합니다. 이 속성이 없는 경우 gcc 가 VMT 오프셋을 변경하는 정보를 입력하여 Kylix에서 C++ 메소드 호출이 실패합니다.

Kylix 인터페이스 또는 클래스 선언은 C++ 클래스 선언과 동일한 호출 규칙을 사용해야 합니다. Kylix에서 cdecl도 지원하고 있기는 하지만 호출 규칙으로 stdcall을 사용하는 것이 좋습니다.

그 밖의 차이점

Kylix 구현에서는 애플리케이션이 동작하는 방법에 영향을 미치는 몇 가지 다른 차이점이 있습니다. 이 단원에서는 이러한 차이점에 대해 설명합니다.

*ToggleButton*은 Enter 키로 토글되지 않습니다. Kylix 에서는 Enter 키를 눌러도 Delphi에서처럼 클릭 이벤트가 발생하지 않습니다.

*TColorDialog*는 설정할 *TColorDialog.Options* 속성이 없습니다. 따라서 색상 선택 대화 상자의 모양 및 기능을 사용자 지정할 수 없습니다. 또한 *TColordialog*가 항상 모달은 아닙니다. Kylix에서 애플리케이션의 제목 표시줄을 모달 대화 상자로 조작할 수 있습니다.(즉 색상 대화 상자의 부모 폼을 선택하고 색상 대화 상자가 열려 있는 동안 최대화하는 등의 작업을 할 수 있습니다.)

런타임 시 콤보 박스는 Kylix에서 Delphi에서와 다르게 동작합니다. Kylix(Delphi는 아님)에서는 콤보 박스의 편집 필드에 텍스트를 입력하고 Enter 키를 눌러서 드롭다운에 항목을 추가할 수 있습니다. *InsertMode*를 ciNone으로 설정하여 이 기능을 끌 수

있습니다. 콤보 박스의 목록에 문자열이 없는 빈 항목을 추가할 수도 있습니다. 또한 아래쪽 화살표를 계속 누르면 콤보 박스 목록의 마지막 항목에서 중단되지 않고 위에서부터 다시 스크롤됩니다.

*TCustomEdit*는 *Undo*, *ClearUndo*, *CanUndo*를 구현하지 않기 때문에 편집 취소를 프로그램으로 작성할 수 없습니다. 그러나 런타임 시 편집 상자를 마우스 오른쪽 단추로 클릭하고 Undo 명령을 선택하면 편집 상자(*TEdit*)의 편집 내용을 취소할 수 있습니다.

Windows에서 Enter 키에 대한 *OnKeyDown* 이벤트나 *KeyUp* 이벤트의 키 값은 13입니다. Linux에서 이 값은 4100입니다. Enter 키에 대한 키 값 13을 확인하듯이 키에 대해 하드 코딩된 숫자 값을 확인하려면 Delphi 애플리케이션을 Kylix로 포팅할 때 이 값을 변경해야 합니다.

CLX 또는 Kylix에 없는 기능

CLX에는 다음과 같은 일반적인 기능이 없습니다.

- 오른쪽에서 왼쪽 텍스트 입력이나 출력을 위한 양방향 속성(*BidiMode*)
- 공용 컨트롤의 일반 베벨 속성(일부 객체에는 아직도 베벨 속성이 있음)
- 도킹 속성 및 메소드
- Win3.1 탭 및 *Ctl3D*와 같은 이전 버전과의 호환성을 위한 기능
- *DragCursor*와 *DragKind*(드래그 앤 드롭 포함)
- 메뉴 항목에 대한 힌트

Kylix는 다음과 같은 기능을 지원하지 않습니다.

- .a 파일 생성 및 연결
- C++ .o 파일 생성 및 연결
- 변수 선언에서 절대 주소 지정. 다음과 같이 absolute 지시어를 사용하면 다른 변수 이름을 참조할 수 있습니다.

  ```
  var Var2: Byte absolute Var1;
  ```

- TASM. Kylix에서 무료로 지원하는 이식 가능한 x86 어셈블러의 일종인 Netwide Assembler(NASM)가 지원하는 구문을 사용하지 않으면 외부 어셈블러 루틴을 import할 수 없습니다.
- Borland의 make 유틸리티. GNU make 유틸리티를 대신 사용합니다.
- 리소스 introspection. 컴파일 시 애플리케이션은 자신이 사용할 모든 리소스의 이름을 알아야 합니다. 리소스가 동적으로 검색되지 않습니다.

포팅되지 않는 기능

Windows 버전의 Kylix에서 지원하는 일부 Windows 특정 기능은 Linux 환경으로 직접 포팅되지 않습니다. COM, ActiveX, OLE, BDE 및 ADO와 같은 기능은 Windows 기술에 의존하므로 Kylix에서는 사용할 수 없습니다. 다음 표는 두 플랫폼 간에 다르게 나타나는 기능들과 Kylix에 해당 기능이 있는 경우 그 기능을 나열한 것입니다.

표 2.3 변경되었거나 다른 기능

Delphi/Windows 기능	Kylix/Linux 기능
Windows API 호출	CLX 메소드, Qt 호출, libc 호출 또는 다른 시스템 라이브러리 호출
COM 컴포넌트(ActiveX 포함)	사용 불가
ADO 컴포넌트	일반 데이터베이스 컴포넌트 사용 가능
Windows 메시징	Qt 이벤트
Winsock	BSD 소켓
MAPI(Messaging Application Programming Interface)에는 Windows 메시징 기능의 표준 라이브러리가 있음	SMTP/POP3로 전자 우편 메시지를 보내고 받고 저장할 수 있음
레거시 컴포넌트(예: Win 3.1 컴포넌트 팔레트 탭의 항목)	사용 불가

Kylix에서 Windows DLL에 해당하는 것은 공유 객체 라이브러리(.so 파일)로서 위치 독립 코드(PIC)를 포함합니다. 다음과 같은 결과가 나타납니다.

- 메모리의 절대 주소를 참조하는 변수(**absolute** 지시어 사용)는 허용되지 않습니다.
- 외부 함수에 대한 전역 메모리 참조와 호출은 호출 간에 보존되어야 하는 EBX 레지스터에 따라 이루어집니다.

Kylix는 정확한 코드를 생성하므로 어셈블러를 사용할 경우 외부 함수에 대한 전역 메모리 참조와 호출만 신경 쓰면 됩니다. 자세한 내용은 2-17페이지의 "인라인 어셈블러 코드 포함"을 참조하십시오.

Kylix 라이브러리 모듈과 패키지는 .so 파일을 사용하여 구현합니다.

CLX와 VCL 유닛

VCL과 CLX의 모든 객체는 유닛 파일인 .pas 소스 파일에 정의되어 있습니다. 예를 들어 시스템 유닛에는 *TObject*의 구현이 있으며 클래시스 유닛은 기본(base) *TComponent* 클래스를 정의합니다. 폼에 객체를 갖다 놓거나 애플리케이션 내에서 객체를 사용하면 해당 유닛의 이름이 **uses** 절에 추가되어 컴파일러에게 어떤 유닛을 프로젝트에 연결해야 하는지 알려 줍니다.

Kylix는 버전마다 포함된 컴포넌트가 다르므로 포함된 유닛의 수도 다릅니다. 포함된 유닛의 종류는 Kylix 버전에 따라 다릅니다.

Delphi의 일부 에디션은 VCL뿐 아니라 CLX도 포함하므로 일반적으로 Kylix에 있는 모든 유닛은 Delphi에도 있습니다. 애플리케이션을 VCL에서 CLX로 이전할 계획이면

CLX에서 유닛 이름이 대부분 다르다는 사실에 유념하십시오. 유닛의 이름은 애플리케이션의 **uses** 절에서 변경해야 합니다. 가장 일반적인 이름 변경 방법은 VCL 이름 앞에 Q를 추가하는 것입니다. 표 2.4는 CLX와 다른 이름을 사용하는 VCL 유닛의 이름을 나열한 것입니다.

표 2.4 CLX 와 다른 이름을 사용하는 VCL 유닛

VCL 유닛	CLX 유닛
ActnList	QActnList
Buttons	QButtons
CheckLst	QCheckLst
Clipbrd	QClipbrd
ComCtrls	QComCtrls
Consts	Consts, Qconsts 및 RTLConsts
Controls	QControls
DBActns	QDBActns
DBConsts	QDBConsts
DBCtrls	QDBCtrls
DBGrids	QDBGrids
DBLogDlg	QDBLogDlg
DBPWDlg	QDBPWDlg
Dialogs	QDialogs
ExtCtrls	QExtCtrls
Forms	QForms
Graphics	QGraphics
Grids	QGrids
ImgList	QImgList
IMask	QMask
Menus	QMenus
Printers	QPrinters
Search	QSearch
StdActns	QStdActns
StdCtrls	QStdCtrls
Types	Types 및 QTypes

uses 절이 위에 나열된 VCL 유닛을 포함하는 경우 VCL 유닛 이름을 해당하는 CLX 유닛 이름으로 변경해야 합니다. 기타 유닛 이름은 Kylix 와 동일합니다. 예를 들어 Classes, Contnrs, ConvUtils, DateUtils, DB, IniFiles, System, SysInit, SysUtils 및 런타임 라이브러리(RTL)에 있는 유닛 등 기타 많은 유닛의 이름은 Kylix와 Delphi 에서 동일합니다.

일부 Windows 유닛은 ADO, COM 및 Borland Database Engine과 같이 Linux에 없는 Windows 특정 기능이므로 Kylix에 포함되어 있지 않습니다. Kylix는 ADO* 유닛, BDE 특정 유닛(BdeConst, DBCGrids, DBExcept, DBInpReq, DBPWDlg, DBTables,

DRTable), COM 유닛(AxCtrls, ComStrs, DbOleCtl, MConnect, Mtsobj, MtsRdm, Mtx, mxConsts, Ole* 유닛, VCLCom), Windows 특정 유닛(CtlPanel, Messages, MPlayer, Registry, SvcMgr, Windows)을 포함하지 않습니다. 이러한 유닛을 참조하거나 유닛 내의 객체를 사용하려면 Linux에서 실행하려는 애플리케이션을 제거해야 합니다. Kylix에 없는 유닛으로 프로그램 컴파일을 시도하면 참조를 포함하는 유닛의 이름 앞에 다음과 같은 치명적인 오류 메시지가 나타납니다.

```
File not found: 'unitname.dcu'
```

uses 절에서 해당 유닛을 삭제하고 컴파일을 다시 시도합니다.

CLX 객체 생성자의 차이점

폼 디자이너에서 암시적으로 해당 객체를 폼에 넣거나 코드에서 객체의 *Create* 메소드를 명시적으로 사용하여 CLX 객체를 만들면 기본으로 사용하는 관련된 widget의 인스턴스도 생성됩니다. widget의 인스턴스는 이 CLX 객체가 소유합니다. *Free* 메소드를 호출하여 CLX 객체를 삭제하거나 CLX 객체의 부모 컨테이너에 의해 CLX가 자동으로 삭제되는 경우에는 기본으로 사용하는 widget도 삭제됩니다. 이 기능은 Windows 애플리케이션의 VCL에서 볼 수 있는 기능과 동일한 유형입니다.

객체의 *Create(AHandle)* 메소드를 사용하여 코드에서 명시적으로 CLX 객체를 만드는 경우에는 생성 중에 사용하기 위해 기존 Qt widget의 인스턴스가 CLX 객체에 전달됩니다. 이 CLX 객체는 자신에게 전달된 Qt widget을 소유하지 않습니다. 따라서 이러한 방법으로 객체를 만든 후 *Free* 메소드를 호출하면 CLX 객체만 소멸되고 기본으로 사용하는 Qt widget 인스턴스는 소멸되지 않습니다. 이것이 VCL과 다른 점입니다.

일부 CLX 객체에서는 *OwnHandle* 메소드를 사용하여 기본으로 사용하는 widget의 소유권을 추정해 볼 수 있습니다. *OwnHandle*을 호출한 후 CLX 객체를 삭제하면 기본으로 사용하는 widget도 소멸됩니다.

Windows와 Linux 간의 소스 파일 공유

애플리케이션을 Windows와 Linux에서 모두 실행하려면 두 운영 체제에서 액세스할 수 있도록 소스 파일을 공유하면 됩니다. 두 컴퓨터에서 액세스할 수 있는 서버에 소스 파일을 저장하거나 Linux 컴퓨터의 Samba를 사용하여 Linux와 Windows용 Microsoft 네트워크 파일을 공유하여 파일에 대한 액세스를 제공하는 등 소스 파일을 공유하는 방법에는 여러 가지가 있습니다. Linux에 소스를 보관하고 Linux에 공유 드라이브를 만들 수 있습니다. 또는 Windows에 소스를 보관하고 Windows에 공유를 만들어 Linux 컴퓨터가 액세스하도록 할 수 있습니다.

VCL과 CLX에서 모두 지원되는 객체를 사용하여 Kylix에서 파일을 개발하고 컴파일할 수 있습니다. 개발이 끝난 파일은 Linux와 Windows에서 모두 컴파일할 수 있습니다.

Kylix의 폼 파일은 .xfm 파일입니다(Delphi는 .dfm 파일). 코드를 단일 소스로 만들려면 Windows의 .dfm 파일을 Linux의 .xfm으로 복사하여 둘 다 관리합니다. 그렇게 하지 않으면 .dfm 파일은 Linux에서 수정되어 Windows에서 사용할 수 없게 됩니다. 크로스 플랫폼 애플리케이션을 작성하려는 경우에는 CLX를 지원하는 Delphi 버전에서 .xfm을 사용하면 됩니다.

Windows와 Linux 간의 환경적 차이점

다음 표는 Windows 환경에서 작업하는 것에 익숙한 사용자가 Linux에 대해 알아야 할 몇 가지 차이점을 나열한 것입니다.

표 2.5 Linux 운영 체제에서의 차이점

차이점	설명
파일 이름 대소문자 구별	Linux에서는 대문자와 소문자를 *구별합니다*. Test.txt 파일은 test.txt 파일과 *다른 파일로 인식됩니다*. Linux에서는 파일 이름의 대소문자에 신경 써야 합니다.
줄 끝 문자	Windows에서 텍스트 행은 CR/LF(즉 ASCII 13 + ASCII 10)로 종료되지만 Linux에서는 LF로 종료됩니다. Kylix의 코드 에디터가 이를 처리하기는 하지만 Windows에서 코드를 import할 때 이 점을 인식하고 있어야 합니다.
파일 끝 문자	DOS와 Windows에서 #26(Ctrl-Z) 문자 값은 해당 문자 뒤에 데이터가 있더라도 텍스트 파일의 끝으로 처리됩니다. Linux에서는 특별한 파일 끝 문자가 없으며 파일의 끝에서 텍스트 데이터가 끝납니다.
배치 파일/셸 스크립트	Linux에서 .bat 파일에 해당하는 것은 셸 스크립트입니다. 스크립트는 명령어가 텍스트 파일로 저장된 파일로서 `chmod +x <scriptfile>` 명령으로 모드를 변경하면 스크립트 파일을 실행할 수 있습니다. 스크립트를 실행하려면 스크립트 파일의 이름을 입력합니다. 스크립트 언어는 Linux에서 사용하는 셸에 따라 다릅니다. 일반적으로 Bash를 사용합니다.
명령 확인	DOS나 Windows에서는 파일이나 폴더를 삭제하려는 경우 "파일을 삭제하시겠습니까?"라고 물어 봅니다. Linux는 일반적으로 묻지 않고 바로 실행하므로 실수로 파일이나 전체 파일 시스템을 삭제해 버릴 수 있습니다. 파일을 다른 매체에 백업하지 않는 한 Linux에서는 삭제를 취소할 방법이 없습니다.
명령 피드백	Linux에서는 명령이 성공하면 상태 메시지 없이 명령 프롬프트를 다시 표시합니다.
명령 스위치	DOS에서 슬래시(/)나 대시(-)를 사용하는 대신 Linux에서는 대시(-)를 사용하여 명령 스위치를 표시하거나 이중 대시(--)를 사용하여 여러 문자 옵션을 표시합니다.
구성 파일	Windows에서 구성은 레지스트리나 autoexec.bat와 같은 파일에서 수행됩니다. Linux에서 구성 파일은 점(.)으로 시작하는 숨김 파일로 만듭니다. 구성 파일들은 /etc 디렉토리와 사용자의 홈 디렉토리에 많이 있습니다. Linux도 LD_LIBRARY_PATH(라이브러리의 검색 경로)와 같은 환경 변수를 사용합니다. 중요한 기타 환경 변수는 다음과 같습니다. HOME 사용자의 홈 디렉토리(/home/sam) TERM 터미널 유형(xterm, vt100, console) SHELL 사용자 셸의 경로(/bin/bash) USER 사용자의 로그인 이름(sfuller) PATH 프로그램을 검색하는 경로 이 환경 변수들은 셸이나 .bashrc와 같은 rc 파일에 지정되어 있습니다.

표 2.5 Linux 운영 체제에서의 차이점(계속)

차이점	설명
DLL	Linux에서는 공유 객체 파일(.so)을 사용합니다. Windows에서는 동적 연결 라이브러리(DLL)를 사용합니다.
드라이브 문자	Linux에는 드라이브 문자가 없습니다. Linux의 경로명을 예로 들면 /lib/security입니다. 런타임 라이브러리에서 DriveDelim을 참조하십시오.
예외	운영 체제 예외는 Linux에서 시그널이라고 합니다.
실행 파일	Linux의 실행 파일에는 확장자가 없습니다. Windows의 실행 파일에는 exe 확장자가 있습니다.
파일 이름 확장자	Linux는 파일 형식을 식별하거나 파일을 응용 프로그램과 연결하는 데 파일 이름 확장자를 사용하지 않습니다.
파일 권한	Linux에서는 파일 소유자, 그룹 등에 대한 읽고 쓰고 실행할 권한이 파일 및 디렉토리에 할당됩니다. 예를 들면 다음과 같습니다. -rwxr-xr-x는 왼쪽에서 오른쪽으로 다음과 같은 의미입니다. 첫 번째 -는 파일 형식(- = 일반 파일, d = 디렉토리, l = 링크)이고 rwx는 파일 소유자의 권한(읽기, 쓰기, 실행)이며 r-x는 파일 소유자 그룹의 권한(읽기, 실행)이고 r-x는 다른 모든 사용자의 권한(읽기, 실행)을 의미합니다. 루트 사용자(수퍼유저)는 이 권한을 무시할 수 있습니다. 올바른 사용자가 애플리케이션을 실행하는지, 애플리케이션의 요청 파일에 대한 적절한 액세스 권한이 있는지 확인해야 합니다.
make 유틸리티	Borland의 make 유틸리티는 Linux 플랫폼에서 사용할 수 없습니다. 대신 Linux에 있는 GNU make 유틸리티를 사용하면 됩니다.
멀티태스킹	Linux는 멀티태스킹을 완벽하게 지원합니다. 동시에 여러 프로그램(Linux에서는 프로세스라고 함)을 실행할 수 있습니다. 명령 뒤에 &를 사용하면 프로세스를 백그라운드로 실행한 다음 작업을 계속할 수 있습니다. Linux에서는 세션을 여러 개 실행할 수도 있습니다.
경로명	DOS에서 역슬래시(\)를 사용하는 곳에 Linux는 슬래시(/)를 사용합니다. PathDelim 상수를 사용하여 플랫폼에 적절한 문자를 지정할 수 있습니다. 런타임 라이브러리에서 PathDelim을 참조하십시오.
검색 경로	프로그램 실행 시 Windows는 항상 현재 디렉토리를 먼저 검색한 다음 PATH 환경 변수를 찾습니다. Linux는 현재 디렉토리를 먼저 검색하지 않고 PATH에 나열된 디렉토리만 검색합니다. 현재 디렉토리에서 프로그램을 실행하려면 대개 앞에 ./를 입력해야 합니다. 검색할 첫 번째 경로에 ./가 포함되도록 PATH를 수정해도 됩니다.
검색 경로 구분자	Windows에서는 검색 경로 구분자로 세미콜론을 사용합니다. Linux에서는 콜론을 사용합니다. 런타임 라이브러리에서 PathDelim을 참조하십시오.
심볼릭 링크	Linux에서 심볼릭 링크는 디스크의 다른 파일을 가리키는 특별한 파일입니다. 애플리케이션의 주요 파일을 가리키는 전역 bin 디렉토리에 심볼릭 링크를 넣으면 시스템 검색 경로를 수정할 필요가 없습니다. 심볼릭 링크는 ln(link) 명령으로 만듭니다. Windows에는 GUI 데스크탑의 단축키가 있습니다. 명령줄에서 프로그램을 사용하기 위해 Windows 설치 프로그램은 일반적으로 시스템 검색 경로를 수정합니다.

Linux의 디렉토리 구조

Linux에서는 디렉토리가 다릅니다. 파일 시스템 상의 어느 곳에서든지 모든 파일 또는 장치를 마운트할 수 있습니다.

참고 Windows 경로명에 역슬래시를 사용하는 것과 달리 Linux 경로명에는 슬래시를 사용합니다. 첫 번째 슬래시는 루트 디렉토리를 의미합니다.

다음은 Linux에서 일반적으로 사용하는 디렉토리입니다.

표 2.6 일반적인 Linux 디렉토리

디렉토리	내용
/	전체 Linux 파일 시스템의 루트 또는 최상위 디렉토리
/root	루트 파일 시스템인 수퍼유저의 홈 디렉토리
/bin	명령, 유틸리티
/sbin	시스템 유틸리티
/dev	파일처럼 보이는 장치
/lib	라이브러리
/home/*username*	*username*이 사용자의 로그인 이름인 곳에서 사용자가 소유한 파일
/opt	옵션
/boot	시스템 시동 시 호출되는 커널
/etc	구성 파일
/usr	애플리케이션, 프로그램. 보통 /usr/spool, /usr/man, /usr/include, /usr/local과 같은 디렉토리를 포함
/mnt	CD나 플로피 디스크 드라이브와 같이 시스템에 마운트되는 기타 매체
/var	로그, 메시지, 스풀 파일
/proc	가상 파일 시스템 및 시스템 통계 보고
/tmp	임시 파일

참고 Linux는 배포판에 따라 파일 위치가 다릅니다. 유틸리티 프로그램이 Red Hat 배포판에는 /bin에 있지만 Debian 배포판에는 /usr/local/bin에 있습니다.

계층적 파일 시스템의 구성에 대한 자세한 내용은 www.pathname.com을 참조하거나 *Filesystem Hierarchy Standard*를 참조하십시오.

이식 가능한 코드 작성

Windows 및 Linux에서 모두 실행할 수 있는 크로스 플랫폼 애플리케이션을 작성하는 경우 다른 조건으로 컴파일할 코드를 작성할 수 있습니다. 조건부 컴파일을 사용하면 Windows 코드를 유지하면서도 Linux 운영 체제와의 차이점을 수용할 수 있습니다.

Windows와 Linux 간에 쉽게 이식 가능한 애플리케이션을 만들려면 다음 사항을 염두에 두어야 합니다.

- 플랫폼 특정(Win32 또는 Linux) API에 대한 호출을 줄이거나 분리시키고 대신 CLX 메소드를 사용합니다.
- 애플리케이션 내에서 구성되는 Windows 메시징(PostMessage, SendMessage)을 제거합니다.
- *TRegIniFile* 대신 *TIniFile* 또는 *TMemIniFile*을 사용합니다. (Linux에서 *TIniFile*과 *TMemIniFile*은 동일합니다.)
- 파일과 디렉토리 이름의 대소문자에 유의합니다.
- 모든 외부 어셈블러 TASM 코드를 포팅합니다. GNU 어셈블러 "as"는 TASM 구문을 지원하지 않습니다. (2-17페이지의 "인라인 어셈블러 코드 포함" 참조)

플랫폼 독립적인 런타임 라이브러리 루틴을 사용하고 시스템, SysUtils 및 기타 런타임 라이브러리 유닛의 상수를 사용하는 코드를 작성하십시오. 예를 들어 PathDelim 상수를 사용하여 '/'와 '\' 플랫폼 차이로 코드를 구분합니다.

두 플랫폼에서 멀티바이트 문자를 사용하는 것이 또 다른 방법입니다. Windows 코드는 오래 전부터 멀티바이트 문자당 2바이트를 사용해 왔습니다. Linux에서 멀티바이트 문자 인코딩은 문자당 더 많은 바이트(UTF-8에 대해 최대 6바이트까지)를 가질 수 있습니다. 두 플랫폼 모두 SysUtils의 StrNextChar 함수를 사용하여 조정할 수 있습니다. 다음과 같은 기존의 Windows 코드가 있다고 가정합니다.

```
while p^ <> #0 do
begin
   if p^ in LeadBytes then
      inc(p);
   inc(p);
end;
```

위의 코드를 다음과 같이 플랫폼 독립적인 코드로 대체할 수 있습니다.

```
while p^ <> #0 do
begin
  if p^ in LeadBytes then
    p := StrNextChar(p)
  else
    inc(p);
end;
```

이 예는 플랫폼을 포팅할 수 있고 2바이트보다 긴 멀티바이트 문자를 지원하지만 멀티바이트가 아닌 로케일에 대해서는 프로시저 호출의 성능 비용을 고려하지 않습니다.

런타임 라이브러리 함수가 적절한 해결책이 아니라면 플랫폼 특정 코드를 루틴에 하나로 넣어 분리하거나 서브루틴으로 넣어 봅니다. 소스 코드의 가독성과 이식성을 유지하려면 **$IFDEF** 블록의 수를 제한하도록 합니다. 조건 기호 WIN32는 Linux에서 정의되지 않습니다. 조건 기호 LINUX는 소스 코드를 Linux 플랫폼용으로 컴파일함을 나타냅니다.

조건 지시어 사용

$IFDEF 컴파일러 지시어를 사용하는 것은 Windows 및 Linux 플랫폼용으로 코드를 조건부 컴파일하는 데 적합한 방법입니다. 하지만 **$IFDEF**가 있는 소스 코드는 코드를 이해하거나 관리하기가 어렵기 때문에 **$IFDEF**를 언제 사용하는 것이 적당한지를 알고 있어야 합니다. **$IFDEF**를 사용할 때 염두에 두어야 할 가장 중요한 질문은 "이 코드에 왜 **$IFDEF**가 필요한가?"와 "**$IFDEF**를 쓰지 않고도 이 코드를 작성할 수 있는가?"입니다.

크로스 플랫폼 애플리케이션에서 $IFDEF를 사용하려면 다음 지침을 따릅니다.

- **$IFDEF**는 반드시 필요한 경우가 아니면 사용하지 마십시오. 소스 파일에 있는 **$IFDEF**는 소스 코드가 컴파일될 때 평가됩니다. C/C++와 달리 Kylix는 프로젝트를 컴파일 시 유닛 소스(헤더 파일)를 요구하지 않습니다. 모든 소스 코드를 완전히 다시 빌드하는 것은 Kylix 프로젝트에서 매우 드문 경우입니다.
- 패키지 파일(.dpk)에서는 **$IFDEF**를 사용하지 마십시오. 소스 파일에만 사용해야 합니다. 컴포넌트 작성자는 크로스 플랫폼 개발 시 **$IFDEF**를 사용하여 하나의 패키지가 아닌 두 개의 디자인 타임 패키지를 만들어야 합니다.
- WIN32를 비롯한 Windows 플랫폼에 대해 테스트할 때에는 일반적으로 **$IFDEF** MSWINDOWS를 사용하십시오. **$IFDEF** WIN32는 32비트와 64비트 Windows와 같은 특정 Windows 플랫폼 간을 구분하기 위해 사용합니다. WIN64에서 사용하지 않는 것이 확실한 경우에만 코드를 WIN32로 제한하십시오.
- **$IFNDEF**와 같은 부정 테스트는 반드시 필요한 경우에만 하십시오. **$IFNDEF** LINUX는 **$IFDEF** MSWINDOWS와는 *다릅니다*.
- **$IFNDEF/$ELSE** 조합을 피하십시오. 그 대신 가독성을 높이려면 긍정 테스트(**$IFDEF**)를 사용합니다.
- 플랫폼에 따라 다른 **$IFDEF**에는 **$ELSE** 절을 사용하지 마십시오. **$IFDEF** LINUX/**$ELSE** 또는 **$IFDEF** MSWINDOWS/**$ELSE** 대신 LINUX와 MSWINDOWS 특정 코드에 따라 별도의 **$IFDEF** 블록을 사용합니다.

예를 들어 기존 코드에는 다음과 같은 내용이 있을 수 있습니다.

```
{$IFDEF WIN32}
   (32-bit Windows code)
{$ELSE}
   (16-bit Windows code)   //!! By mistake, Linux could fall into this code.
{$ENDIF}
```

$IFDEF 블록의 이식 불가능한 코드의 경우 플랫폼이 $ELSE 절에 의해 런타임 시 알 수 없는 오류를 일으키는 것보다 소스 코드가 컴파일되지 않는 것이 낫습니다. 컴파일 오류를 찾는 것이 런타임 오류를 찾는 것보다 쉽습니다.

- 복잡한 테스트에는 $IF 문을 사용합니다. 중첩된 $IFDEF를 $IF 지시어의 부울 표현식으로 바꿉니다. $IF 지시어는 $ENDIF가 아닌 $IFEND를 사용해서 종료해야 합니다. 이전 컴파일러로부터 새로운 $IF 구문을 숨기려면 $IF 표현식을 $IFDEF 안에 둡니다.

온라인 도움말에는 조건 지시어가 모두 나와 있습니다. 자세한 내용은 도움말의 "Conditional Compilation" 항목을 참조하십시오.

조건 지시어 종료

$IFEND 지시어를 사용하여 조건 지시어 $IF와 $ELSEIF를 종료합니다. $IFDEF/$ENDIF를 사용하는 대신 이렇게 하면 $IF/$IFEND 블록이 이전 컴파일러로부터 숨겨집니다. 그러면 이전 컴파일러가 $IFEND 지시어를 인식하지 못합니다. $IF는 $IFEND로만 종료할 수 있습니다. 이전의 지시어($IFDEF, $IFNDEF, $IFOPT)는 $ENDIF로만 종료할 수 있습니다.

참고 $IFDEF/$ENDIF 내에 $IF를 중첩할 경우 $ELSE를 $IF와 함께 사용하지 마십시오. 이전 컴파일러가 $ELSE를 발견하면 $IFDEF의 일부로 생각하여 행 아래에 컴파일러 오류를 일으킵니다. $IF를 먼저 받아들이면 $ELSEIF를 받아들이지 못하므로 이전 컴파일러는 $ELSEIF를 인식하지 못합니다. 이러한 상황에서는 {$ELSE} 대신 {$ELSEIF True}를 사용할 수 있습니다. 여러 다른 버전에서 코드를 실행하려는 협력업체와 애플리케이션 개발자들에게는 역호환성을 위해 $IF를 숨기는 것이 가장 중요합니다.

$ELSEIF는 $ELSE와 $IF의 조합입니다. $ELSEIF 지시어를 통해 여러 조건부 블록 중 하나만 받아들이게 작성할 수 있습니다. 예를 들면 다음과 같습니다.

```
{$IFDEF doit}
   do_doit
{$ELSEIF  RTLVersion >= 14}
    goforit
{$ELSEIF  somestring = 'yes'}
    beep
{$ELSE}
    last chance
{$IFEND}
```

위의 네 가지 경우 중 하나만 받아들입니다. 처음 세 개의 조건이 true가 아닌 경우 $ELSE 절을 받아들입니다. $ELSEIF는 $IFEND로 종료해야 합니다. $ELSEIF는 $ELSE 뒤에 나타날 수 없습니다. 조건은 일반적인 $IF...$ELSE처럼 위에서 아래로 계산됩니다. 예제에서 doit이 정의되지 않고 RTLVersion은 15이며 somestring = 'yes'

인 경우 조건이 둘 다 True일지라도 "beep" 블록이 아닌 "goforit" 블록만 받아들입니다.

$ENDIF**를 사용하여 **$IFDEF를 종료해야 한다는 것을 잊어버린 경우에는 컴파일러가 소스 파일의 끝에 다음 오류 메시지를 보고합니다.

```
Missing ENDIF
```

소스 파일에 **$IF/$IFDEF** 지시어가 여러 개 있는 경우에는 어느 것이 문제를 일으키는지 알아내기 어려울 수도 있습니다. Kylix는 일치하지 않는 **$ENDIF/$IFEND**가 있는 마지막 **$IF/$IFDEF** 컴파일러 지시어의 소스 행에 다음 오류 메시지를 보고합니다.

```
Unterminated conditional directive
```

이 위치에서 문제점을 찾기 시작하면 됩니다.

메시지 나타내기

$MESSAGE 컴파일러 지시어는 컴파일러가 하는 것처럼 소스 코드에 힌트, 경고 및 오류를 나타낼 수 있습니다.

{**$MESSAGE** HINT|WARN|ERROR|FATAL 'text string' }

메시지 유형은 옵션입니다. 메시지 유형이 지정되지 않은 경우 기본값은 HINT입니다. 텍스트 문자열이 필요하며 작은 따옴표로 묶어야 합니다.

예제:

{**$MESSAGE** 'Boo!'} 힌트를 표시합니다.

{**$Message** Hint 'Feed the cats'} 힌트를 표시합니다.

{**$Message** Warn 'Looks like rain.'} 경고를 표시합니다.

{**$Message** Error 'Not implemented'} 오류를 표시하고 컴파일을 계속합니다.

{**$Message** Fatal 'Bang. Yer dead.'} 오류를 표시하고 컴파일러를 종료합니다.

인라인 어셈블러 코드 포함

Windows 애플리케이션에 인라인 어셈블러 코드를 포함하면 Linux의 위치 독립 코드(PIC) 요구 조건 때문에 Linux에서 같은 코드를 사용하지 못할 수도 있습니다. Linux 공유 객체 라이브러리(Windows의 DLL에 해당됨)는 코드를 수정하지 않고 모두 메모리에 재배치할 수 있어야 합니다. 이는 전역 변수 또는 기타 절대 주소를 사용하거나 외부 함수를 호출하는 인라인 어셈블러 루틴에 1차적인 영향을 줍니다.

오브젝트 파스칼 코드만 포함하는 유닛의 경우 컴파일러가 필요할 때 자동으로 PIC를 생성합니다. PIC 유닛의 확장자는 .dcu가 아닌 .dpu입니다. 모든 파스칼 유닛 소스 파일은 PIC 형식과 PIC가 아닌 형식 두 가지로 컴파일하는 것이 좋습니다. -p 컴파일러 스위치를 사용하여 PIC를 생성합니다. 미리 컴파일되어 있는 유닛은 두 가지 형식으로 사용할 수 있습니다.

실행 파일이나 공유 라이브러리 중 어느 것으로 컴파일하는지에 따라 어셈블러 루틴을 다르게 코딩하려는 경우 {**$IFDEF** PIC} 를 사용하여 어셈블러 코드를 두 개의 버전으로 나눕니다. 다른 방법으로는 오브젝트 파스칼에서 루틴을 다시 작성해도 됩니다.

다음은 인라인 어셈블러 코드에 대한 PIC 규칙입니다.

- PIC에서는 모든 메모리 참조가 현재 모듈의 기본 주소 포인터(Linux에서 전역 오프셋 테이블 또는 GOT라고 함)를 포함하는 EBX 레지스터에 상대적이어야 합니다. 따라서 다음을 사용하지 않습니다.

  ```
  MOV EAX,GlobalVar
  ```

  대신 다음을 사용합니다

  ```
  MOV EAX,[EBX].GlobalVar
  ```

- PIC에서는 Win32에서와 같이 호출 간에 EBX 레지스터를 어셈블리 코드에 남겨 두어야 하고 또한 Win32에서와 달리 외부 함수를 호출하기 *전에* EBX 레지스터를 복원해야 합니다.
- PIC 코드는 기본 실행 파일에서 작동되기는 하지만 성능이 떨어지고 코드가 더 많이 생성됩니다. 공유 객체에서는 선택의 여지가 없지만 실행 파일에서는 가능한 한 가장 높은 수준의 성능을 유지하십시오.

메시지와 시스템 이벤트

메시지 루프와 이벤트는 Linux에서 다르게 수행되지만 컴포넌트 작성에 1차적인 영향을 줍니다. 대부분의 컴포넌트와 속성 편집기는 쉽게 포팅됩니다. *TObject.Dispatch*와 클래스의 메시지 메소드 구문이 Linux에서 제대로 수행되는 반면 Linux 환경에서 운영 체제 공지는 메시지가 아닌 시스템 이벤트를 사용하여 처리됩니다.

이벤트 핸들러를 작성하려면 Windows 메시지에 응답하는 대신 표 2.7에 설명된 메소드 중 하나를 오버라이드하여 사용자 지정 메시지를 작성할 수 있습니다. 오버라이드 시 상속된 메소드를 호출하면 모든 기본 프로세스(default process)를 계속 발생시킬 수 있습니다.

표 2.7 시스템 이벤트에 응답하는 TWidgetControl protected 메소드

메소드	설명
ChangeBounds	*TWidgetControl*의 크기가 변경될 때 사용됩니다. Windows의 WM_SIZE 또는 WM_MOVE와 대체로 유사합니다. Qt는 클라이언트 영역에 기반하여 widget의 "geometry"를 설정하며 VCL은 Qt가 프레임으로 참조하는 것을 포함하는 전체 컨트롤 크기를 사용합니다.
ChangeScale	컨트롤 크기 변경 시 자동으로 호출됩니다. 폼의 크기와 화면 해상도나 글꼴 크기를 조절하는 모든 컨트롤의 크기를 변경하는 데 사용됩니다. ChangeScale은 컨트롤의 Top, Left, Width 및 Height 속성을 수정하므로 컨트롤과 컨트롤의 자식의 크기는 물론 위치를 변경합니다.
ColorChanged	컨트롤의 색상이 변경되었을 때 호출됩니다.
CursorChanged	커서 모양이 바뀔 때 호출됩니다. 마우스 커서가 이 widget 위에 있을 때 모양이 바뀝니다.
EnabledChanged	애플리케이션이 창이나 컨트롤의 활성화 상태를 변경할 때 호출됩니다.
FontChanged	글꼴 리소스의 컬렉션이 변경될 때 호출됩니다. widget의 글꼴을 설정하고 모든 자식에게 변경 내용을 알려 줍니다. WM_FONTCHANGE 메시지와 대체로 유사합니다.

표 2.7 시스템 이벤트에 응답하는 TWidgetControl protected 메소드(계속)

메소드	설명
PaletteChanged	시스템 팔레트가 변경될 때 호출됩니다. WM_PALETTECHANGED 메시지와 대체로 유사합니다.
ShowHintChanged	도움말 힌트가 컨트롤에 표시되거나 숨겨졌을 때 호출됩니다.
StyleChanged	창이나 컨트롤의 GUI 스타일이 변경되었을 때 호출됩니다. WM_STYLECHANGED 메시지와 대체로 유사합니다.
TabStopChanged	폼 상의 탭 순서가 변경되었을 때 호출됩니다.
VisibleChanged	컨트롤이 숨겨지거나 보여질 때 호출됩니다.
WidgetDestroyed	컨트롤을 기본으로 사용하는 widget이 소멸될 때 호출됩니다.

Qt는 C++ 툴킷이므로 모든 Qt widget은 C++ 객체입니다. CLX는 오브젝트 파스칼로 작성되는데 오브젝트 파스칼은 C++ 객체와 직접적으로 상호 작용하지 않습니다. 또한 Qt는 여러 곳에서 다중 상속을 사용합니다. 그러므로 Kylix에는 인터페이스 레이어가 들어 있어 모든 Qt 클래스를 일련의 직접적인 C 함수로 변환합니다. 그런 다음 레이어들이 Linux에서는 공유 객체로, Windows에서는 DLL로 랩됩니다.

모든 *TWidgetControl*에는 *CreateWidget*, *InitWidget*과 거의 항상 오버라이드해야 하는 *HookEvents* 가상 메소드가 있습니다. *CreateWidget*은 Qt widget을 만들고 Handle을 FHandle private field 변수로 할당합니다. *InitWidget*은 widget이 생성된 후 호출되고 Handle이 유효하게 됩니다.

Linux에서 일부 속성은 이제 Create 생성자에서 할당되지 않고 *InitWidget*에서 할당됩니다. 이로 인해 Qt 객체가 실제로 필요할 때까지 Qt 객체 생성이 지연됩니다. 예를 들어 *Color*라는 이름의 속성이 있다고 가정합니다. SetColor에서 Qt 핸들이 있는지 보려면 HandleAllocated에서 확인할 수 있습니다. Handle이 할당되어 있으면 색상을 설정하기 위해 Qt를 호출할 수 있습니다. Handle이 할당되지 않았으면 private field 변수에 값을 저장하고 *InitWidget*에서 속성을 설정합니다.

Linux는 Widget과 System이라는 두 가지 유형의 이벤트를 지원합니다. *HookEvents*는 CLX 컨트롤 이벤트 메소드를 Qt 객체와 통신하는 특별한 훅 객체에 훅(hook)하는 가상 메소드입니다. 훅 객체는 메소드 포인터의 집합입니다. Linux에서 시스템 이벤트는 기본적으로 *WndProc*을 대체하는 *EventHandler*에서 실행됩니다.

Linux에서의 프로그래밍 차이점

Linux wchar_t widechar는 문자당 32비트입니다. 오브젝트 파스칼 widechar가 지원하는 16비트 유니코드 표준은 Linux와 GNU 라이브러리가 지원하는 32비트 UCS 표준의 서브셋입니다. 파스칼 widechar 데이터는 wchar_t로 OS 함수에 전달되기 전에 문자당 32비트로 확장되어야 합니다.

Linux에서 와이드 문자열은 긴 문자열처럼 간주되는 참조입니다(Windows에서는 아님).

Linux에서의 멀티바이트 처리는 Windows와 다릅니다. Windows에서 멀티바이트 문자(MBCS)는 1바이트와 2바이트 char 코드로 표시됩니다. Linux에서는 1에서 6바이트로 표시됩니다.

AnsiStrings는 멀티바이트 문자 구조를 가질 수 있으며 사용자의 로케일 설정에 따라 달라집니다. 일본어, 중국어, 히브리어 및 아라비아어와 같은 멀티바이트 문자의 Linux 인코딩은 동일한 로케일에 대한 Windows 인코딩과 호환되지 않습니다. 멀티바이트는 포팅할 수 없지만 유니코드는 포팅할 수 있습니다.

Linux에서는 절대 주소에 변수를 사용할 수 없습니다. `var X: Integer absolute $1234;` 구문은 PIC에서는 지원하지 않으며 CLX를 포함하고 있는 모든 Delphi 버전에서 허용되지 않습니다.

크로스 플랫폼 데이터베이스 애플리케이션

Windows의 Delphi에서는 데이터베이스 정보에 액세스하는 방법이 여러 가지 있습니다. 여러 가지 방법 중에 Windows Delphi에서는 ADO, Borland Database Engine(BDE) 및 InterBase Express를 사용할 수 있는 반면 Kylix에서는 사용할 수 없습니다. 대신 Delphi 버전 6과 함께 출시되어 Windows에서도 사용할 수 있는 새로운 크로스 플랫폼 데이터 액세스 기술인 *dbExpress*를 사용할 수 있습니다.

*dbExpress*가 Linux에서도 실행되도록 데이터베이스 애플리케이션을 포팅하기 전에 *dbExpress*와 이전에 사용했던 데이터 액세스 메커니즘 간의 차이점을 이해해야 합니다. 차이점은 수준에 따라 다릅니다.

- 가장 낮은 수준에는 애플리케이션과 데이터베이스 서버 간에 통신하는 레이어가 있습니다. 이 레이어는 ADO, BDE 또는 InterBase 클라이언트 소프트웨어가 될 수 있습니다. 이 레이어는 동적 SQL 프로세싱을 위한 경량급(lightweight) 드라이버 집합인 *dbExpress*로 대체됩니다.
- 낮은 수준의 데이터 액세스는 데이터 모듈이나 폼에 추가하는 컴포넌트 집합에 랩됩니다. 이 컴포넌트에는 데이터베이스 서버로의 연결을 나타내는 데이터베이스 연결 컴포넌트와 서버로부터 fetch된 데이터를 나타내는 데이터셋이 있습니다. 중요한 차이점이 있기는 하지만 *dbExpress* 커서의 단방향성 때문에 이 수준에서는 그 차이가 적습니다. 그 이유는 데이터베이스 연결 컴포넌트처럼 데이터셋이 공통 조상을 공유하기 때문입니다.
- 사용자 인터페이스 수준에는 차이가 거의 없습니다. CLX data-aware 컨트롤은 해당 Windows 컨트롤과 최대한 유사하게 디자인되어 있습니다. 사용자 인터페이스 수준에서의 주요 차이점은 캐시된 업데이트의 사용을 적용하기 위해 변경이 필요할 때 나타납니다.

기존의 데이터베이스 애플리케이션을 *dbExpress*로 포팅하는 방법에 대한 자세한 내용은 2-23페이지의 "데이터베이스 애플리케이션을 Linux로 포팅"을 참조하십시오. 새 *dbExpress* 애플리케이션을 디자인하는 방법에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 14장 "데이터베이스 애플리케이션 디자인"을 참조하십시오.

dbExpress 차이점

Linux에서 *dbExpress*는 데이터베이스 서버와의 통신을 관리합니다. *dbExpress*는 공통 인터페이스를 구현하는 경량급(lightweight) 드라이버의 집합으로 구성되어 있습니다. 드라이버는 모두 사용자 애플리케이션에 연결되어야 하는 공유 객체(.so 파일)입니다.

*dbExpress*는 크로스 플랫폼용으로 디자인되었으므로 Windows에서도 동적 연결 라이브러리(.dll) 집합으로 사용할 수 있습니다.

모든 데이터 액세스 레이어와 마찬가지로 *dbExpress*에도 데이터베이스 협력 업체가 제공하는 클라이언트측 소프트웨어가 필요합니다. 또한 데이터베이스 특정 드라이버와 두 개의 구성 파일 dbxconnection과 dbxdriver를 사용합니다. 주요 Borland Database Engine 라이브러리(Idapi32.dll)와 데이터베이스 특정 드라이버 및 기타 많은 지원 라이브러리를 요구하는 BDE 등의 경우보다는 훨씬 적은 것입니다.

다음은 *dbExpress*와 애플리케이션을 포팅하기 위해 필요한 기타 데이터 액세스 레이어의 차이점입니다.

- *dbExpress*는 원격 데이터베이스에 대해 보다 간단하고 빠른 경로를 제공합니다. 그 결과 간단하고 빠른 데이터 액세스를 통해 눈에 띄게 향상된 성능을 기대할 수 있습니다.
- *dbExpress*는 쿼리와 내장 프로시저를 처리할 수 있지만 개방형 테이블의 개념은 지원하지 않습니다.
- *dbExpress*는 단방향 커서만 반환합니다.
- *dbExpress*는 INSERT, DELETE 또는 UPDATE 쿼리를 실행하는 기능 이외에 기본 제공된 업데이트 지원이 없습니다.
- *dbExpress*는 메타데이터를 캐시하지 않으며 디자인 타임 메타데이터 액세스 인터페이스는 코어 데이터 액세스 인터페이스를 사용하여 구현됩니다.
- *dbExpress*는 사용자가 요청한 쿼리만 실행하므로 다른 쿼리를 사용하지 않고 데이터베이스 액세스를 최적화합니다.
- *dbExpress*는 레코드 버퍼나 레코드 버퍼 블록을 내부적으로 관리합니다. 이로 인해 BDE와는 달리 클라이언트가 레코드 버퍼에 사용되는 메모리를 할당해야 합니다.
- *dbExpress*는 Paradox, dBase 또는 FoxPro와 같은 SQL 방식이 아닌 로컬 테이블은 지원하지 않습니다.
- *dbExpress* 드라이버는 InterBase, Oracle, Informix, DB2 및 MySQL을 지원합니다. 다른 데이터베이스 서버를 사용하고 있을 경우에는 이 데이터베이스 중 하나로 데이터를 포팅하여 사용 중인 데이터베이스 서버용 *dbExpress* 드라이버를 작성하거나 사용자의 데이터베이스 서버용으로 사용할 외부 *dbExpress* 드라이버를 구해야 합니다.

컴포넌트 수준 차이점

dbExpress 애플리케이션을 작성할 때 기존의 데이터베이스 애플리케이션에 사용되는 것과 다른 데이터 액세스 컴포넌트 집합이 필요합니다. *dbExpress* 컴포넌트는 다른 데이터 액세스 컴포넌트(*TDataSet* 및 *TCustomConnection*)와 동일한 기본 클래스를 공유하며 이는 속성, 메소드 및 이벤트가 기존 애플리케이션에 사용되는 컴포넌트와 동일하다는 것을 의미합니다.

표 2.8은 Windows 환경의 InterBase Express, BDE 및 ADO에서 사용되는 중요한 데이터베이스 컴포넌트를 나열하며 Linux와 크로스 플랫폼 애플리케이션에서 사용되는 유사한 *dbExpress* 컴포넌트를 보여 줍니다.

표 2.8 유사한 데이터 액세스 컴포넌트

InterBase Express 컴포넌트	BDE 컴포넌트	ADO 컴포넌트	dbExpress 컴포넌트
TIBDatabase	*TDatabase*	*TADOConnection*	***TSQLConnection***
TIBTable	*TTable*	*TADOTable*	***TSQLTable***
TIBQuery	*TQuery*	*TADOQuery*	***TSQLQuery***
TIBStoredProc	*TStoredProc*	*TADOStoredProc*	***TSQLStoredProc***
TIBDataSet		*TADODataSet*	***TSQLDataSet***

하지만 *dbExpress* 데이터셋(*TSQLTable*, *TSQLQuery*, *TSQLStoredProc* 및 *TSQLDataSet*)은 편집을 지원하지 않고 포워드(forward) 탐색만 허용하기 때문에 다른 컴포넌트의 유사한 데이터셋에 비해 더 제한적입니다. *dbExpress* 데이터셋과 Windows에서 사용 가능한 기타 데이터셋의 차이점에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 18장 "단방향 데이터셋 사용"을 참조하십시오.

편집과 탐색 지원이 부족하므로 대부분의 *dbExpress* 애플리케이션은 *dbExpress* 데이터셋과 직접 연동하지 않습니다. 그보다는 메모리에 레코드를 버퍼하고 편집과 탐색 지원을 제공하는 클라이언트 데이터셋에 *dbExpress* 데이터셋을 연결합니다. 이 아키텍처에 대한 자세한 내용은 *Kylix1 개발자 안내서* 14-4페이지의 "데이터베이스 아키텍처"를 참조하십시오.

참고 아주 간단한 애플리케이션의 경우 클라이언트 데이터셋에 연결된 *dbExpress* 데이터셋 대신 *TSQLClientDataSet*을 사용할 수 있습니다. 이것은 포팅하려는 애플리케이션의 데이터셋과 포팅된 애플리케이션의 데이터셋이 일대일로 대응하기 때문에 간단하다는 이점이 있지만 *dbExpress* 데이터셋을 클라이언트 데이터셋에 명시적으로 연결하는 것보다는 유연하지 않습니다. 대부분의 애플리케이션에서는 *TClientDataSet* 컴포넌트에 연결된 *dbExpress* 데이터셋을 사용하는 것이 좋습니다.

사용자 인터페이스 수준 차이점

CLX data-aware 컨트롤은 해당 Windows 컨트롤과 가능한 유사하게 디자인되어 있습니다. 그 결과 데이터베이스 애플리케이션의 사용자 인터페이스 부분을 포팅하면 Windows 애플리케이션을 CLX로 포팅하는 것 외에 추가로 고려할 사항이 거의 없습니다.

사용자 인터페이스 수준의 주요 차이점은 *dbExpress* 데이터셋이나 클라이언트 데이터셋이 데이터를 제공하는 방법에 있습니다.

dbExpress 데이터셋만 사용하는 경우 편집을 지원하지 않고 포워드 탐색만 지원한다는 사실에 맞추어 사용자 인터페이스를 조정해야 합니다. 따라서 사용자가 이전의 레코드로 이동하도록 허용하는 컨트롤을 제거해야 할 수도 있습니다. *dbExpress* 데이터셋은 데이터를 버퍼하지 않으므로 data-aware 그리드에 데이터를 표시할 수 없습니다. 즉 한 번에 하나의 레코드만 표시할 수 있습니다.

dbExpress 데이터셋을 클라이언트 데이터셋에 연결했을 경우 편집 및 탐색과 관련된 사용자 인터페이스 요소는 계속 작동합니다. 클라이언트 데이터셋에 다시 연결만 하면 됩니다. 이 경우에 제일 먼저 고려해야 할 것은 데이터베이스에 업데이트 내용을 기록하는 방법을 처리하는 것입니다. 기본적으로 Windows의 대부분의 데이터셋은 포스트될 때(예를 들어 사용자가 새 레코드로 이동할 때) 자동으로 데이터베이스 서버에 업데이트를 기록합니다. 반면 클라이언트 데이터셋은 항상 메모리에 업데이트를 캐시합니다. 이러한 차이점을 조정하는 방법에 대한 자세한 내용은 2-25페이지의 "dbExpress 애플리케이션에서 데이터 업데이트"를 참조하십시오.

데이터베이스 애플리케이션을 Linux로 포팅

데이터베이스 애플리케이션을 *dbExpress*로 포팅하면 Windows와 Linux에서 모두 실행되는 크로스 플랫폼 애플리케이션을 만들 수 있습니다. 포팅 기법이 다르기 때문에 포팅 과정에는 애플리케이션을 변경하는 것이 포함됩니다. 포팅의 난이도는 애플리케이션의 유형, 복잡한 정도 및 필요 조건 등에 따라 다릅니다. ADO와 같은 Windows 특정 기술을 많이 사용하는 애플리케이션을 포팅하는 것이 Delphi 데이터베이스 기술을 사용하는 애플리케이션을 포팅하는 것보다 더 어렵습니다.

다음과 같은 일반적인 절차를 따라 Windows 데이터베이스 애플리케이션을 Linux로 포팅합니다.

1 데이터베이스 데이터를 저장할 곳을 고려합니다. *dbExpress*는 Oracle, Interbase, Informix, DB2 및 MySQL용 드라이버를 제공합니다. 데이터는 이들 SQL 서버 중 하나에 두어야 합니다.

   Delphi 5 Enterprise 버전이 있으면 Data Pump 유틸리티를 사용하여 Paradox, dBase 및 FoxPro와 같은 플랫폼에서 지원되는 플랫폼 중 하나로 로컬 데이터베이스 데이터를 옮길 수 있습니다. (Data Pump 유틸리티 사용에 대한 자세한 내용은 Program Files\Common Files\Borland\Shared\BDE에 있는 datapump.hlp 파일을 참조하십시오.)

2 사용자 인터페이스 폼을 데이터셋 및 연결 컴포넌트를 가진 데이터 모듈과 분리하지 않았을 경우 포팅을 시작하기 전에 분리해도 됩니다. 이렇게 하면 완전히 새로운 컴포넌트 집합을 요구하는 애플리케이션 부분을 데이터 모듈과 분리할 수 있습니다. 그런 다음 사용자 인터페이스를 나타내는 폼을 다른 애플리케이션과 마찬가지로 포팅할 수 있습니다. 자세한 내용은 2-2페이지의 "애플리케이션 포팅"을 참조하십시오.

   나머지 단계에서는 데이터셋과 연결 컴포넌트가 자신의 데이터 모듈에서 분리되어 있는 것으로 가정합니다.

3 데이터셋의 CLX 버전과 연결 컴포넌트를 보유할 새로운 데이터 모듈을 만듭니다.

4 원본 애플리케이션의 각 데이터셋에 *dbExpress* 데이터셋, *TDataSetProvider* 컴포넌트 및 *TClientDataSet* 컴포넌트를 추가합니다. 표 2.8에서 해당되는 것을 사용하여 사용할 *dbExpress* 데이터셋을 결정합니다. 컴포넌트에 이름을 부여합니다.

   - *TClientDataSet* 컴포넌트의 *ProviderName* 속성을 *TDataSetProvider* 컴포넌트의 이름으로 설정합니다.

- *TDataSetProvider* 컴포넌트의 *DataSet* 속성을 *dbExpress* 데이터셋으로 설정합니다.
- 원본 데이터셋을 참조하는 데이터 소스 컴포넌트의 *DataSet* 속성을 변경하여 클라이언트 데이터셋을 바로 참조할 수 있게 합니다.

5 다음과 같이 새 데이터셋이 원본 데이터셋과 일치하도록 속성을 설정합니다.

- 원본 데이터셋이 *TTable*, *TADOTable* 또는 *TIBTable* 컴포넌트였을 경우 새 *TSQLTable*의 *TableName* 속성을 원본 데이터셋의 *TableName*으로 설정합니다. 또한 마스터/디테일 관계를 설정하거나 인덱스를 지정하는 데 사용한 모든 속성을 복사합니다. 범위와 필터를 지정하는 속성은 새로운 *TSQLTable* 컴포넌트가 아닌 클라이언트 데이터셋에 설정해야 합니다.
- 원본 데이터셋이 *TQuery*, *TADOQuery* 또는 *TIBQuery* 컴포넌트였을 경우 새 *TSQLQuery* 컴포넌트의 *SQL* 속성을 원본 데이터셋의 *SQL* 속성으로 설정합니다. 새 *TSQLQuery*의 *Params* 속성을 원본 데이터셋의 *Params*나 *Parameters* 속성 값과 일치하도록 설정합니다. 마스터/디테일 관계 성립을 위해 *DataSource* 속성을 설정했으면 이 속성도 복사합니다.
- 원본 데이터셋이 *TStoredProc*, *TADOStoredProc* 또는 *TIBStoredProc* 컴포넌트였으면 새 *TSQLStoredProc* 컴포넌트의 *StoredProcName*을 원본 데이터셋의 *StoredProcName*이나 *ProcedureName* 속성으로 설정합니다.
  새 *TSQLStoredProc*의 *Params* 속성을 원본 데이터셋의 *Params*나 *Parameters* 속성 값과 일치하도록 설정합니다.

6 원본 애플리케이션에 있는 모든 데이터베이스 연결 컴포넌트 (*TDatabase*, *TIBDatabase* 또는 *TADOConnection*)의 경우 새 데이터 모듈에 *TSQLConnection* 컴포넌트를 추가합니다. 또한 ADO 데이터셋의 *ConnectionString* 속성을 사용하거나 BDE 데이터셋의 *DatabaseName* 속성을 BDE 앨리어스로 설정하는 등 연결 컴포넌트 없이 연결한 모든 데이터베이스 서버에 *TSQLConnection* 컴포넌트를 추가해야 합니다.

7 4단계의 각 *dbExpress* 데이터셋마다 해당 *SQLConnection* 속성을 적절한 데이터베이스 연결에 맞는 *TSQLConnection* 컴포넌트로 설정합니다.

8 각 *TSQLConnection* 컴포넌트마다 데이터베이스 연결에 필요한 정보를 지정합니다. 그렇게 하려면 *TSQLConnection* 컴포넌트를 더블 클릭하여 Connection Editor를 표시하고 매개변수 값을 설정하여 적절한 설정을 지정합니다. 1단계에서 데이터를 새로운 데이터베이스 서버로 전송했으면 새 서버에 맞는 설정을 지정합니다. 이전과 동일한 서버를 사용하는 경우 다음과 같이 원본 연결 컴포넌트에서 일부 정보를 찾을 수 있습니다.

- 원본 애플리케이션이 *TDatabase*를 사용했을 경우에는 *Params*와 *TransIsolation* 속성에 나타나는 정보를 전송해야 합니다.
- 원본 애플리케이션이 *TADOConnection*을 사용했을 경우에는 *ConnectionString*과 *IsolationLevel* 속성에 나타나는 정보를 전송해야 합니다.
- 원본 애플리케이션이 *TIBDatabase*를 사용했을 경우에는 *DatabaseName*과 *Params* 속성에 나타나는 정보를 전송해야 합니다.

- 원본 연결 컴포넌트가 없을 경우 BDE 앨리어스와 관련되었거나 데이터셋의 *ConnectionString* 속성에 나타난 정보를 전송해야 합니다.

새로운 연결 이름에서 이 매개변수 집합을 저장할 수 있습니다. 이 과정에 대한 자세한 내용은 *kylix1 개발자 안내서*의 19-2페이지의 "서버 연결 설명"을 참조하십시오.

dbExpress 애플리케이션에서 데이터 업데이트

dbExpress 애플리케이션은 클라이언트 데이터셋을 사용하여 편집을 지원합니다. 편집 내용을 클라이언트 데이터셋에 포스트하면 변경 사항이 클라이언트 데이터셋의 데이터 메모리 내(in-memory) 스냅샷에는 기록되지만 데이터베이스 서버에는 자동으로 기록되지 않습니다. 원본 애플리케이션이 업데이트 캐시를 위해 클라이언트 데이터셋을 사용했으면 Linux에서의 편집 지원을 위해 어떤 것도 변경할 필요가 없습니다. 그러나 Windows에서 레코드를 포스트할 때 대부분 데이터셋의 기본 동작에 의존하여 데이터베이스 서버에 편집 내용을 기록한 경우 클라이언트 데이터셋을 사용하기 위해 변경해야 합니다.

이전에 업데이트를 캐시하지 않은 애플리케이션을 변환하는 방법에는 다음 두 가지가 있습니다.

- 업데이트된 레코드가 포스트되자마자 코드를 작성하여 데이터베이스 서버에 적용함으로써 Windows에서 데이터셋의 동작을 흉내낼 수 있습니다. 이렇게 하려면 다음과 같이 업데이트를 데이터베이스 서버에 적용하는 *AfterPost* 이벤트 핸들러와 함께 클라이언트 데이터셋을 사용합니다.

```
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  with DataSet as TClientDataSet do
    ApplyUpdates(1);
end;
```

- 사용자 인터페이스를 조정하여 캐시된 업데이트 내용을 처리할 수 있습니다. 이 접근법은 네트워크 트래픽의 양을 줄이고 트랜잭션 시간을 최소화하는 등의 이점이 있습니다. 하지만 캐시된 업데이트 사용으로 전환하면 해당 업데이트를 데이터베이스 서버에 다시 적용할 시기를 정해야 하고, 사용자 인터페이스를 변경하여 사용자가 업데이트된 애플리케이션을 초기화하도록 하거나 사용자에게 편집 내용이 데이터베이스에 기록되었는지 여부에 대한 피드백을 제공해야 합니다. 또한 사용자가 레코드를 포스트할 때 업데이트 오류가 감지되지 않기 때문에 사용자에게 오류를 보고하는 방법을 변경하여 발생한 문제의 유형뿐 아니라 문제를 일으킨 업데이트의 종류를 사용자가 알 수 있게 합니다.

원본 애플리케이션이 업데이트를 캐시하기 위해 BDE나 ADO가 제공하는 지원을 사용했을 경우 코드를 수정하여 클라이언트 데이터셋 사용으로 전환해야 합니다. 다음 표는

BDE와 ADO 데이터셋에서 캐시된 업데이트를 지원하는 속성, 이벤트 및 메소드와 *TClientDataSet*의 해당 속성, 메소드 및 이벤트를 나열한 것입니다.

표 2.9 캐시된 업데이트를 위한 속성, 메소드 및 이벤트

BDE 데이터셋 (또는 TDatabase)	ADO 데이터셋	TclientDataSet	용도
CachedUpdates	*LockType*	필요 없음. 클라이언트 데이터셋은 항상 업데이트를 캐시함	캐시된 업데이트가 효력이 있는지 결정합니다.
지원하지 않음	*CursorType*	지원하지 않음	서버의 변경으로 인해 데이터셋이 분리된 방법을 지정합니다.
UpdatesPending	지원하지 않음	*ChangeCount*	로컬 캐시가 데이터베이스에 적용되어야 하는 업데이트된 레코드를 포함하는지 나타냅니다.
UpdateRecordTypes	*FilterGroup*	*StatusFilter*	캐시된 업데이트를 적용할 때 볼 수 있도록 업데이트된 레코드의 종류를 나타냅니다.
UpdateStatus	*RecordStatus*	*UpdateStatus*	레코드가 변경되지 않았는지, 수정되었는지, 삽입되었는지 또는 삭제되었는지 나타냅니다.
OnUpdateError	지원하지 않음	*OnReconcileError*	레코드별로 업데이트 오류를 처리하기 위한 이벤트입니다.
ApplyUpdates (데이터셋이나 데이터베이스에서)	UpdateBatch	*ApplyUpdates*	로컬 캐시의 레코드를 데이터베이스에 적용합니다.
CancelUpdates	CancelUpdates 또는 CancelBatch	*CancelUpdates*	대기 중인 업데이트를 적용하지 않고 로컬 캐시에서 제거합니다.
CommitUpdates	자동 처리됨	*Reconcile*	애플리케이션 업데이트가 성공한 다음 업데이트 캐시를 지웁니다.
FetchAll	지원하지 않음	*GetNextPacket* (및 *PacketRecords*)	데이터베이스 레코드를 편집 및 업데이트할 목적으로 로컬 캐시에 복사합니다.
RevertRecord	CancelBatch	*RevertRecord*	업데이트가 아직 적용되지 않았으면 현재 레코드에 대한 업데이트를 취소합니다.

크로스 플랫폼 인터넷 애플리케이션

인터넷 애플리케이션은 클라이언트를 서버에 연결하기 위해 표준 인터넷 프로토콜을 사용하는 클라이언트/서버 애플리케이션입니다. 클라이언트/서버 통신용 표준 인터넷 프로토콜을 사용하므로 애플리케이션을 크로스 플랫폼으로 만들 수 있습니다. 예를 들어 인터넷 애플리케이션의 서버측 프로그램은 컴퓨터의 웹 서버 소프트웨어를 통해 클라이언트와 통신합니다. 서버 애플리케이션은 일반적으로 Linux 또는 Windows용으로 작성되지만 크로스 플랫폼으로 바꿀 수 있습니다. 클라이언트는 두 플랫폼 중 어느 하나에 둘 수 있습니다.

Kylix를 사용하면 Linux에 배치하기 위한 CGI나 Apache 애플리케이션과 같은 웹 서버 애플리케이션을 작성할 수 있습니다. Windows에서는 Microsoft Server DLL(ISAPI), Netscape Server DLL(NSAPI) 및 Windows CGI 애플리케이션과 같은 다른 유형의 웹 서버를 만들 수 있습니다. 직접적인 CGI 애플리케이션과 WebBroker를 사용하는 일부 애플리케이션만 Windows와 Linux에서 모두 실행됩니다.

인터넷 애플리케이션을 Linux로 포팅

기존의 인터넷 애플리케이션을 Linux로 옮기려면 웹 서버 애플리케이션을 포팅할 것인지 아니면 Linux에서 새 애플리케이션을 작성할 것인지 고려해야 합니다. 웹 서버 작성에 대한 자세한 내용은 6장 "인터넷 서버 애플리케이션 생성"을 참조하십시오. 애플리케이션이 WebBroker를 사용하고 WebBroker 인터페이스에 쓰지만 원시 API 호출을 사용하지 않는 경우에는 애플리케이션을 Linux로 포팅하는 것이 그다지 어렵지 않습니다.

애플리케이션이 ISAPI, NSAPI, Windows CGI 또는 기타 웹 API를 사용하는 경우에는 포팅하기가 더 어렵습니다. 소스 파일을 통해 검색하고 이 API 호출을 Apache(Apache API용 함수 프로토타입에 대해서는 인터넷 디렉토리에서 httpd.pas 참조)나 CGI 호출로 변환해야 합니다. 2-1페이지의 "Windows 애플리케이션을 Linux로 포팅"에 설명되어 있는 다른 모든 내용들도 변경해야 합니다.

3

패키지와 컴포넌트 사용

* 이 장은 영문 Kylix2 개발자 안내서의 11장입니다.

*패키지*는 Kylix 애플리케이션과 IDE에서 사용하는 특수 공유 객체 파일입니다. *런타임 패키지*는 사용자가 애플리케이션을 실행할 때 사용하는 기능을 제공합니다. *디자인 타임 패키지*는 IDE에 컴포넌트를 설치하고 사용자 지정 컴포넌트에 대한 특수 속성 편집기를 생성하는 데 사용됩니다. 단일 패키지는 디자인 타임과 런타임 모두 사용할 수 있고 디자인 타임 패키지는 종종 런타임 패키지를 호출하여 작동합니다.

패키지는 공유 객체 파일이 들어 있는 bin 디렉토리에 bplpackage.so.*version#*와 같이 bpl 접두사가 붙은 상태로 저장됩니다. 여기서 *version#*는 패키지의 주요 버전 번호입니다. (실제 패키지 이름은 버전 번호만 있는 이름을 가리키는 심볼릭 링크뿐만 아니라 개정 번호도 포함할 수 있습니다. 예를 들어 bplindy.so.6이 bplindy.so.6.2.를 가리킬 수도 있습니다.) 패키지 설명서에는 패키지 이름이 bpl 접두사나 버전 번호, 개정 번호 없이 표시됩니다.

다른 런타임 라이브러리처럼 패키지는 애플리케이션 사이에 공유할 수 있는 코드를 포함할 수 있습니다. 예를 들어 가장 많이 사용하는 Kylix 비주얼 컴포넌트는 visualclx라는 패키지에 있습니다. 비주얼 컨트롤이 포함된 GUI 애플리케이션을 개발하기 위해 visualclx를 사용한 경우 애플리케이션을 컴파일하면 공통 코드는 visualclx에 있고 애플리케이션의 실행 이미지에는 이미지에 고유한 코드와 데이터만 포함됩니다. 패키지를 사용하는 여러 개의 애플리케이션이 컴퓨터 한 대에 설치된 경우 이 컴퓨터에는 visualclx 복사본 하나만 필요합니다. 이는 모든 애플리케이션과 IDE가 visualclx 복사본을 공유하기 때문입니다.

Kylix에는 CLX 컴포넌트를 캡슐화하는 몇 개의 미리 컴파일된 런타임 패키지가 함께 포함되어 있습니다. Kylix는 IDE에서 컴포넌트를 처리하는 디자인 타임 패키지를 사용합니다.

애플리케이션을 패키지와 함께 구축하거나 패키지 없이 구축할 수 있습니다. 그러나 IDE에 사용자 지정 컴포넌트를 추가하려면 패키지를 디자인 타임 패키지로 설치해야 합니다.

애플리케이션 사이에 공유하는 런타임 패키지를 사용자가 직접 만들 수 있습니다. Kylix 컴포넌트를 작성하는 경우 컴포넌트를 설치하기 전에 디자인 타임 패키지로 컴파일할 수 있습니다.

패키지의 이점

디자인 타임 패키지는 사용자 지정 컴포넌트를 배포하고 설치하는 작업을 단순화합니다. 옵션으로 사용하는 런타임 패키지는 몇 가지 측면에서 전통적인 프로그래밍보다 유용합니다. 재사용된 코드를 런타임 라이브러리로 컴파일하여 애플리케이션 간에 공유할 수 있습니다. 예를 들어 Kylix를 비롯한 모든 애플리케이션은 패키지를 통해 표준 컴포넌트에 액세스할 수 있습니다. 애플리케이션에는 실행 파일에 바운드된 별도의 컴포넌트 라이브러리 복사본이 없기 때문에 실행 파일은 더욱 작아져 시스템 자원과 하드 디스크 저장소가 절약됩니다. 또한 애플리케이션에 고유한 코드만 빌드 시 컴파일되기 때문에 패키지를 사용하면 컴파일이 빨라집니다.

패키지와 표준 공유 객체 파일

IDE에서 사용할 수 있는 사용자 지정 컴포넌트를 만들려고 하는 경우 패키지를 만듭니다. 애플리케이션 구축 시 사용한 개발 툴에 상관없이 애플리케이션에서 호출할 수 있는 라이브러리를 구축하려는 경우 표준 공유 객체 파일을 만듭니다.

다음 표는 패키지에 연결된 파일 형식을 나열합니다.

표 3.1 컴파일된 패키지 파일

파일 확장자	내용
dpk	패키지에 포함된 유닛을 나열하는 소스 파일.
dcp	컴파일러가 필요로 하는 모든 기호 정보를 비롯한 패키지 헤더와 패키지의 모든 dpu 파일의 연결을 포함하는 바이너리 이미지. 각 패키지에 대해 dcp 파일이 하나 생성됩니다. dcp에 대한 기본 이름은 dpk 소스 파일의 기본(base) 이름입니다. 애플리케이션을 패키지와 함께 구축하려면 .dcp 파일이 있어야 합니다.
dpu	패키지에 포함된 유닛 파일에 대한 바이너리 이미지. 각 유닛 파일에 대해 필요한 경우 하나의 dpu가 생성됩니다
so	런타임 패키지. 이 파일은 특수한 Kylix 특정 기능이 있는 공유 객체 파일입니다. 패키지 이름은 bpl*package*.so이며 여기에서 *package*는 dpk 파일의 기본(base) 이름입니다. 옵션으로서 bpl과 같은 접두사, 접미사, 개정 번호를 패키지의 프로젝트 옵션에 지정할 수 있습니다.

참고 패키지는 애플리케이션에 있는 다른 모듈과 전역 데이터를 공유합니다.

공유 객체 파일과 패키지에 대한 자세한 내용은 *오브젝트 파스칼 랭귀지 안내서*를 참조하십시오.

런타임 패키지

런타임 패키지는 Kylix 애플리케이션과 함께 배포됩니다. 런타임 패키지는 사용자가 애플리케이션을 실행할 때 기능을 제공합니다.

패키지를 사용하는 애플리케이션을 실행하려면 컴퓨터에 애플리케이션의 실행 파일과 애플리케이션이 사용하는 모든 패키지가 있어야 합니다. 패키지 파일은 애플리케이션이 사용하는 시스템 경로에 있어야 합니다. 애플리케이션을 배포하는 경우 사용자가 필요한 패키지의 적합한 버전을 가지고 있는지 확인해야 합니다.

애플리케이션에서 패키지 사용

다음과 같은 방법으로 애플리케이션에서 패키지를 사용합니다.

1. IDE에서 프로젝트를 로드하거나 만듭니다.
2. Project|Options를 선택합니다.
3. Packages 탭을 선택합니다.
4. "Build with runtime packages" 체크 박스를 선택하고 아래에 있는 편집 상자에 패키지 이름을 하나 이상 입력합니다. 설치된 디자인 타임 패키지에 연결된 런타임 패키지를 편집 상자에서 선택해도 됩니다.
5. 기존 목록에 패키지를 추가하려면 Add 버튼을 클릭하고 Add Runtime Package 대화 상자에 새 패키지 이름을 입력합니다.
6. 사용 가능한 패키지 목록에서 검색하려면 Add 버튼을 클릭한 다음 Add Runtime Package 대화 상자의 Package Name 편집 상자 옆에 있는 Browse 버튼을 클릭합니다.

   Add Runtime Package 대화 상자의 Search Path 편집 상자를 편집하면 Kylix의 전역 Library Path가 변경됩니다.

패키지 이름에 파일 확장자를 쓸 필요는 없습니다. 파일 이름 여러 개를 Runtime Packages 편집 상자에 직접 입력하는 경우 이름을 콜론으로 분리해야 합니다.

Runtime Packages 편집 상자에 나열된 패키지는 컴파일할 때 애플리케이션에 자동으로 연결됩니다. 중복되는 패키지 이름은 무시되며 편집 상자가 비어 있으면 애플리케이션은 패키지 없이 컴파일됩니다.

런타임 패키지는 현재 프로젝트에 대해서만 선택됩니다. 현재 선택 값을 새 프로젝트에 대한 자동 기본값으로 만들려면 대화 상자 아래쪽에 있는 "Defaults" 체크 박스를 선택합니다.

참고 패키지가 포함된 애플리케이션을 생성하는 경우에도 원래의 Kylix 유닛 이름을 소스 파일의 **uses** 절에 포함시켜야 합니다. 예를 들어 메인 폼의 소스 파일은 다음과 같이 시작합니다.

```
unit MainForm;

interface

uses
SysUtils, Types, Classes, QGraphics, QControls, QForms, QDialogs;
```

이 예제에서 참조하는 유닛은 visualclx 및 baseclx 패키지에 포함되어 있지만 여기에서도 **uses** 절에 이러한 참조가 포함되어야 합니다. 참조가 포함되어 있지 않으면 컴파일러 오류가 발생합니다. 생성된 소스 파일에서 Kylix는 이 유닛을 자동으로 **uses** 절에 추가합니다.

패키지 동적 로딩

패키지를 실행 파일에 포함시키는 대신에 동적으로 로드하는 것이 좋은 경우도 있습니다. 패키지를 동적으로 로드하는 경우 패키지가 실행 파일과 분리되므로 더 작은 실행 파일을 만들 수 있습니다. 패키지를 실행 파일에 포함시키는 것은 동일한 패키지를 여러 개의 애플리케이션에서 사용하는 시스템이나 하나의 패키지 집합에 액세스하는 실행 파일을 여러 개 배포할 경우에 유용합니다. 패키지를 사용하면 패키지를 제외하고 실행 파일을 전송할 수 있기 때문에 업데이트를 전송하는 작업도 간단해집니다.

다음 두 가지 방법으로 패키지를 동적으로 로드할 수 있습니다.

- Project|Options를 선택하고 Packages 페이지에 있는 "Build with runtime packages" 체크 박스를 선택합니다.
- *LoadPackage* 함수를 사용하여 프로그램을 작성합니다.

"Build with runtime packages" 체크 박스를 사용하는 경우 패키지는 필요할 때만 암시적으로 로드됩니다. 다시 말해서 해당 패키지에 있는 유닛 중의 하나에 정의되어 있는 오브젝트를 지정했을 경우에만 암시적으로 로드됩니다. 3-3페이지의 "애플리케이션에서 패키지 사용"에서 설명한 것처럼 체크 박스 아래의 편집 상자에 사용하는 애플리케이션에 필요한 패키지를 지정합니다. 어떤 런타임 패키지를 사용해야 할지 보려면 다음 단원을 참조하십시오.

애플리케이션에서 *LoadPackage* 함수를 호출해서 런타임에 패키지를 로드할 수도 있습니다. *LoadPackage*는 이름 매개변수에 지정된 패키지를 로드하고 중복 유닛을 확인하고 패키지에 포함되어 있는 모든 유닛의 초기화 블록을 호출합니다. 예를 들어 다음 코드는 파일 선택 대화 상자에서 파일을 선택할 때 실행될 수 있습니다.

```
with OpenDialog1 do
  if Execute then
    with PackageList.Items do
      AddObject(FileName, Pointer(LoadPackage(FileName)));
```

패키지를 동적으로 언로드하려면 *UnloadPackage*를 호출합니다. 패키지에 정의된 클래스 인스턴스를 소멸하고 패키지로 등록된 클래스의 등록을 해제할 때 주의하십시오.

사용할 런타임 패키지 결정

Kylix에는 baseclx 및 visualclx를 비롯한 기본 랭귀지와 컴포넌트 지원을 제공하는 몇 개의 미리 컴파일된 런타임 패키지가 함께 포함되어 있습니다. 사용자가 보유하고 있는 Kylix의 버전에 따라 추가적인 패키지가 포함되어 있습니다. 패키지를 사용하여 컴포넌트를 제품과 함께 배포할 수 있고 패키지에 기능이 서로 연관된 유닛을 포함시킬 수 있습니다.

예를 들어 visualclx 패키지에는 가장 일반적으로 사용되는 컴포넌트 및 버튼과 리스트 박스 같은 사용자 인터페이스 요소가 있습니다. baseclx 패키지에는 시스템 지원을 위해 필요한 모든 넌비주얼 컴포넌트가 있습니다.

패키지를 사용하는 데이터베이스 애플리케이션을 만들려면 최소한 두 개의 런타임 패키지(baseclx와 dataclx)가 있어야 합니다. 비주얼 컴포넌트를 사용하려는 경우에도 visualclx가 필요하고 data-aware 컨트롤을 사용하려는 경우에는 visualdbclx도 있어야 합니다. 웹 서버 애플리케이션을 작성하려는 경우에는 baseclx와 visualclx는 물론이고 netclx도 필요합니다.

이 패키지를 사용하려면 Project|Options를 선택하고 Packages 탭을 선택하여 Runtime Packages 편집 상자에 사용하려는 패키지를 지정합니다. 패키지 이름이 여러 개인 경우 콜론으로 구분합니다.

다음 표는 Kylix와 함께 제공되는 런타임 패키지를 모두 나열하고 설명한 것입니다. 사용하고 있는 제품의 버전에 따라 패키지가 다르고 애플리케이션의 유형에 따라 사용자의 애플리케이션에 필요한 패키지가 다릅니다.

표 3.2 런타임 패키지

패키지	내용
baseclx	런타임 라이브러리 및 시스템 지원 유닛. 모든 애플리케이션에 필수.
dataclx	데이터베이스 컴포넌트. 모든 데이터베이스 애플리케이션에 필수.
indy	서버와 클라이언트 환경을 위한 크로스 플랫폼 인터넷 컴포넌트. Indy 컴포넌트를 사용하는 인터넷 애플리케이션에 필수인 오픈 소스 컴포넌트.
netclx	인터넷 컴포넌트. 모든 인터넷 애플리케이션에 필수.
netdataclx	Web Broker 애플리케이션용 컨텐트 프로듀서. 데이터베이스 정보를 사용하는 Web Broker 애플리케이션에 필수.
visualclx	GUI 애플리케이션을 지원하기 위한 비주얼 컴포넌트. 모든 GUI 애플리케이션에 필수.
visualdbclx	컴포넌트 팔레트의 Data Controls 탭에 나타나는 Data-aware 컨트롤(예: DBImage, DBGrid, DBEdit). 데이터베이스로부터 실제 데이터를 표시하는 애플리케이션에 필수.
webdsnapclx	DataSnap 애플리케이션 작성을 위한 컴포넌트. 원격 데이터베이스 서버나 XML 문서에서 데이터를 fetch하는 클라이언트 애플리케이션과 애플리케이션 서버를 포함하는 다계층 데이터베이스 애플리케이션에 필수.
websnapclx	스크립트 가능한 HTML 페이지 탬플릿 제작을 위한 디스패처, 어댑터, 페이지 프로듀서, 세션 및 사용자 목록 컴포넌트. WebSnap 컴포넌트를 사용하는 웹 서버 애플리케이션 구축에 필수.
xmlrtl	XML 문서, XML 스키마, DOM(Document Object Model) 표준 인터페이스를 지원하는 컴포넌트. XML 데이터로 작업하는 애플리케이션에 필수.

이 표를 참조하면 사용자의 애플리케이션에 필요한 패키지가 어떤 것인지 구분할 수 있습니다. 사용자의 애플리케이션에 필요한 패키지를 판단하는 다른 방법은 패키지를 직접 실행한 다음 이벤트 로그를 검토하는 것입니다. View|Debug Windows|Event Log를 선택합니다. 이벤트 로그에는 모든 패키지를 포함해서 로드된 모든 모듈이 표시됩니다. 전체 패키지 이름이 나열됩니다. 예를 들어 visualclx의 경우라면 다음과 같은 줄을 볼 수 있을 것입니다.

```
Module Load: bplvisualclx.so.6. Has Debug Info. Base Address $40017000. Process Proj1 (2906)
```

사용자 지정 패키지

사용자 지정 패키지는 사용자가 직접 코딩하고 컴파일한 패키지 또는 공급 업체에서 제공하는 미리 컴파일된 패키지입니다. 사용자 지정 런타임 패키지를 애플리케이션과 함께 사용하려면 Project|Options를 선택하고 Packages 페이지에 있는 Runtime Packages 편집 상자에 패키지 이름을 추가합니다. 예를 들어, bplstats.so라는 통계 패키지가 있다고 가정합니다. 애플리케이션에서 사용하려면 Runtime Packages 편집 상자에 이 패키지를 포함합니다.

사용자가 패키지를 직접 만드는 경우 필요에 따라 목록에 직접 만든 패키지를 추가할 수 있습니다.

디자인 타임 패키지

디자인 타임 패키지는 IDE의 컴포넌트 팔레트에 컴포넌트를 설치하고 사용자 지정 컴포넌트에 대한 특수 속성 편집기를 생성하는 데 사용합니다.

Kylix에는 IDE에 미리 설치되어 있는 여러 개의 디자인 타임 컴포넌트 패키지가 함께 포함되어 있습니다. 설치된 패키지는 사용하는 Kylix 버전 및 사용자 지정 여부에 따라 다릅니다. Component|Install Packages를 선택하면 시스템에 설치되어 있는 패키지 목록을 볼 수 있습니다.

디자인 타임 패키지는 Requires 절에서 참조하는 런타임 패키지를 호출함으로써 실행됩니다. 3-10페이지의 "Requires 절"을 참조하십시오. 예를 들어, dclstd는 bplvcl을 참조합니다. dclstd 패키지에는 많은 표준 컴포넌트를 컴포넌트 팔레트에 사용할 수 있는 추가 기능이 있습니다.

이미 설치된 패키지 외에 IDE에 사용자의 컴포넌트 패키지나 협력 업체에서 제공하는 컴포넌트 패키지를 설치할 수 있습니다. dclusr 디자인 타임 패키지는 새 컴포넌트에 대한 기본 컨테이너로 제공됩니다.

컴포넌트 패키지 설치

모든 컴포넌트는 IDE에 패키지로 설치됩니다. 컴포넌트를 직접 만든 경우 컴포넌트를 포함하는 패키지를 만들고 컴파일합니다(3-7페이지의 "패키지 생성 및 편집" 참조). 컴포넌트 소스 코드를 직접 작성하려면 *Kylix1 개발자 안내서*의 4부 "사용자 지정 컴포넌트 생성"에서 설명한 방법을 따라야 합니다.

사용자가 만든 컴포넌트 또는 협력 업체에서 제공하는 컴포넌트를 설치하거나 설치를 제거하려면 다음 단계를 따릅니다.

1 새 패키지를 설치하는 경우 패키지 파일을 로컬 디렉토리로 복사하거나 이동합니다. 패키지에 추가 파일이 포함되어 있는 경우 추가 파일을 모두 복사했는지 확인합니다. 추가 파일에 대한 내용은 3-13페이지의 "컴파일이 성공했을 때 생성되는 패키지 파일"을 참조하십시오.

   .dcp 파일과 .dpu 파일이 배포본에 포함된 경우 이 파일을 저장하는 디렉토리가 Kylix Library Path에 있어야 합니다.

2 Component|Install Packages를 IDE 메뉴에서 선택하거나 Project|Options를 선택하고 Packages 탭을 클릭합니다.

3 사용 가능한 패키지 목록이 "Design packages" 아래에 나타납니다.
   - IDE에 패키지를 설치하려면 옆에 있는 체크 박스를 선택합니다.
   - 패키지를 설치 제거하려면 체크 박스의 선택을 해제합니다.
   - 설치된 패키지에 포함된 컴포넌트 목록을 보려면 패키지를 선택하고 Components를 클릭합니다.
   - 패키지를 목록에 추가하려면 Add를 클릭하고 Open Package 대화 상자에서 패키지 파일이 있는 디렉토리를 검색합니다(1 단계 참조). 패키지 파일을 선택하고 Open을 클릭합니다.
   - 목록에서 패키지를 제거하려면 패키지를 선택하고 Remove를 클릭합니다.

4 OK를 클릭합니다.

컴포넌트의 *RegisterComponents* 프로시저에서 지정한 컴포넌트 팔레트 페이지에 패키지의 컴포넌트를 동일한 프로시저에서 할당된 이름으로 설치합니다.

기본 설정 값을 변경하지 않는 경우 모든 사용 가능한 패키지를 설치한 새 프로젝트가 생성됩니다. 현재 설치 선택 사항을 새 프로젝트의 자동 기본값으로 만들려면 대화 상자 아래쪽에 있는 Default 체크 박스를 선택합니다.

패키지를 설치 제거하지 않고 컴포넌트 팔레트에서 컴포넌트를 제거하려면 Component|Configure Palette를 선택하거나 Tools|Environment Options를 선택하고 Palette 탭을 클릭합니다. Palette 옵션 탭은 컴포넌트가 나타나는 컴포넌트 팔레트 페이지의 이름과 함께 설치된 각 컴포넌트를 나열합니다. 컴포넌트를 선택하고 Hide를 클릭하면 팔레트에서 컴포넌트가 제거됩니다.

패키지 생성 및 편집

패키지를 만들려면 다음을 지정해야 합니다.

- 패키지 *이름*
- 새 패키지에 *필요*하거나 새 패키지에 연결되는 다른 패키지의 목록
- 컴파일될 때 패키지가 *포함*되거나 패키지에 바운드되는 유닛 파일의 목록. 패키지는 본질적으로 컴파일된 패키지의 기능을 포함하는 소스 코드 유닛에 대한 랩퍼(wrapper)입니다. Contains 절에 패키지에 컴파일하려는 사용자 지정 컴포넌트의 소스 코드 유닛을 지정합니다.

.dpk 확장자로 끝나는 패키지 소스 파일은 Package 에디터에서 생성합니다.

패키지 생성

패키지를 만들려면 다음 절차를 따릅니다. 여기에서 대략적으로 설명한 단계에 대한 자세한 내용은 3-9페이지의 "패키지 구조 이해"를 참조하십시오.

참고 크로스 플랫폼 애플리케이션을 개발하는 경우라면 패키지 파일(.dpk)에서 IFDEF를 사용하지 마십시오. 그러나 소스 코드에는 IFDEF를 사용할 수 있습니다.

1 File|New를 선택하고 Package 아이콘을 선택한 다음 OK를 클릭합니다.

2 Package 에디터에 생성된 패키지가 나타납니다.

3 Package 에디터는 새 패키지의 *Requires* 노드와 *Contains* 노드를 보여 줍니다.

4 Add to package 스피드 버튼을 클릭하여 유닛을 **contains** 절에 추가합니다. Add unit 페이지에서 Unit file name 편집 상자에 .pas 파일명을 입력하거나 Browse를 클릭하여 파일을 검색한 다음 OK를 클릭합니다. 선택한 유닛은 Package 에디터의 Contains 노드 아래에 나타납니다. 이 단계를 반복하여 유닛을 더 추가할 수 있습니다.

5 Add to package 스피드 버튼을 클릭하여 패키지를 **requires** 절에 추가합니다. Requires 페이지에서 Package name 편집 상자에 .dcp 파일명을 입력하거나 Browse를 클릭하여 파일을 검색한 다음 OK를 클릭합니다. 선택한 패키지는 Package 에디터의 Requires 노드 아래에 나타납니다. 이 단계를 반복하여 패키지를 더 추가할 수 있습니다.

6 Options 스피드 버튼을 클릭하고 빌드하려는 패키지 종류를 결정합니다.

- 디자인 타임 전용 패키지(런타임에 사용할 수 없는 패키지)를 만들려면 Designtime only 라디오 버튼을 선택합니다. 또는 {$DESIGNONLY} 컴파일러 지시어를 dpk 파일에 추가합니다.
- 런타임 전용 패키지(설치될 수 없는 패키지)를 만들려면 Runtime only 라디오 버튼을 선택합니다. 또는 {$RUNONLY} 컴파일러 지시어를 dpk 파일에 추가합니다.
- 디자인 타임과 런타임 모두에 사용할 수 있는 패키지를 만들려면 Designtime and runtime 라디오 버튼을 선택합니다.

7 Package 에디터에서 Compile package 스피드 버튼을 클릭하여 패키지를 컴파일합니다.

기존 패키지 편집

기존 패키지를 열어서 편집하려면 다음 방법 중 하나를 사용합니다.

- File|Open(또는 File|Reopen)을 선택하고 dpk 파일을 선택합니다.
- Component|Install Packages를 선택하고 Design Packages 목록에서 패키지를 선택한 다음 Edit 버튼을 클릭합니다.
- Package 에디터가 열려 있으면 Requires 노드의 패키지 중 하나를 선택하여 마우스 오른쪽 버튼을 클릭한 다음 Open을 선택합니다.

패키지의 설명을 편집하거나 사용 옵션을 설정하려면 Package 에디터에서 Options 스피드 버튼을 클릭하고 Description 탭을 선택합니다.

Project Options 대화 상자에는 하단 왼쪽 모서리에 Default 체크 박스가 있습니다. 이 체크 박스를 선택하고 OK를 클릭하면 선택한 옵션이 새 프로젝트에 대한 기본 설정 값으

로 저장됩니다. 원래의 기본값을 복원하려면 defjproj.dof 파일을 삭제하거나 이름을 변경하십시오.

수동으로 패키지 소스 파일 편집

프로젝트 파일처럼 패키지 소스 파일은 사용자가 제공하는 정보로부터 Kylix에 의해 생성됩니다. 프로젝트 파일처럼 패키지 소스 파일도 수동으로 편집할 수 있습니다. 패키지 소스 파일은 오브젝트 파스칼 소스 코드를 포함하는 다른 파일과의 혼동을 피하기 위해 .dpk(Kylix 패키지) 확장자로 저장해야 합니다.

다음과 같은 방법으로 코드 에디터에서 패키지 소스 파일을 엽니다.

1 Package 에디터에서 패키지를 엽니다.

2 Package 에디터를 마우스 오른쪽 버튼으로 클릭하고 View Source를 선택합니다.

- **package** 헤더는 패키지 이름을 지정합니다.
- **requires** 절은 현재 패키지가 사용하는 다른 외부 패키지를 나열합니다. 다른 패키지에 있는 유닛을 사용하는 유닛이 패키지에 포함되어 있지 않으면 **requires** 절은 필요 없습니다.
- **contains** 절은 컴파일되고 패키지에 바인드된 유닛 파일을 식별합니다. 포함된 유닛에서 사용하지만 필요한 패키지에 존재하지 않는 모든 유닛은 contains 절에 나열되지 않더라도(컴파일러가 경고를 보냄) 패키지에 바인드됩니다.

예를 들어 다음 코드는 dataclx 패키지를 선언합니다.

```
package dataclx;
  requires visualclx;
  contains Db, Dbcgrids, Dbctrls, Dbgrids, Dbinpreq, Dblogdlg, Dbpwdlg, Dbtables,
mycomponent in 'usr/components/mycomponent.pas';
end.
```

패키지 구조 이해

패키지는 다음을 포함합니다.

- 패키지 이름
- Requires 절
- Contains 절

패키지 이름 지정

패키지 이름은 프로젝트 내에서 고유해야 합니다. 패키지 이름을 **stats**라고 지정하면 Package 에디터는 stats.dpk라는 소스 파일을 생성합니다. 컴파일러는 bplstats.so와 stats.dcp라는 실행 파일과 바이너리 이미지를 생성합니다. 다른 패키지의 **requires** 절에서 패키지를 참조하거나 애플리케이션에서 패키지를 사용할 때 stats를 사용합니다.

Requires 절

requires 절에는 현재 패키지가 사용하는 다른 외부 패키지를 지정합니다. **requires** 절에 포함된 외부 패키지는 외부 패키지에 포함된 유닛 중 하나와 현재 패키지를 모두 사용하는 애플리케이션에 컴파일 시 자동으로 연결됩니다.

패키지에 포함된 유닛 파일이 다른 패키지화된 유닛을 참조하면 다른 패키지는 사용자 패키지의 **requires** 절에 있어야 하고 만약 없으면 직접 추가해야 합니다. 다른 패키지가 **requires** 절에 생략되어 있으면 컴파일러는 다른 패키지를 "암시적으로 포함된 유닛"인 사용자의 패키지에 import합니다.

참고 사용자가 만든 대부분의 패키지는 bplclx가 필요합니다. SysUtils 유닛을 비롯해서 CLX 유닛에 의존하는 패키지는 **requires** 절에 bplclx를 나열하거나 bplclx가 필요한 다른 패키지를 나열합니다.

순환 패키지 참조 피하기

패키지는 **requires** 절에 순환 참조를 포함할 수 없습니다. 이는 다음을 의미합니다.

- 패키지는 자신의 **requires** 절에서 자신을 참조할 수 없습니다.
- 참조 체인은 체인에서 패키지를 참조하지 않고 종료해야 합니다. 패키지 A가 패키지 B를 요구하면 패키지 B는 패키지 A를 요구할 수 없습니다. 패키지 A가 패키지 B를 요구하고 패키지 B가 패키지 C를 요구하면 패키지 C는 패키지 A를 요구할 수 없습니다.

중복 패키지 참조 처리

패키지의 **requires** 절이나 Runtime Packages 편집 상자의 중복 참조는 컴파일러에 의해 무시됩니다. 그러나 프로그래밍의 명확성과 가독성을 위해 중복 패키지 참조를 찾아서 제거해야 합니다.

Contains 절

contains 절은 패키지로 바인드되는 유닛 파일을 식별합니다. 패키지를 직접 작성하는 경우 소스 코드를 pas 파일에 저장하고 **contains** 절에 이 파일을 포함합니다.

소스 코드 중복 사용 피하기

패키지는 다른 패키지의 **contains** 절에 나타날 수 없습니다.

패키지의 **contains** 절에 직접적으로 포함되거나 유닛에 간접적으로 포함된 모든 유닛은 컴파일 시 패키지에 바인드됩니다.

Kylix IDE를 비롯하여 동일한 애플리케이션에서 사용하는 둘 이상의 패키지에 하나의 유닛이 직접적으로나 간접적으로 포함될 수 없습니다. 이는 bplclx에 있는 유닛 중 하나를 포함하는 패키지를 만들면 IDE에 패키지를 설치할 수 없다는 것을 의미합니다. 다른 패키지에 이미 패키지로 만든 유닛 파일을 사용하려면 첫 번째 패키지를 두 번째 패키지의 **requires** 절에 포함하십시오.

패키지 컴파일

IDE나 명령줄에서 패키지를 컴파일할 수 있습니다. IDE에서 패키지를 다시 컴파일하려면 다음과 같이 합니다.

1 File|Open을 선택합니다.

2 Kylix Package (*.dpk)를 Files of Type 드롭다운 목록에서 선택합니다.

3 대화 상자에서 .dpk 파일을 선택합니다.

4 Package 에디터를 열 때 Compile 스피드 버튼을 클릭합니다.

컴파일러 지시어를 패키지 소스 코드에 삽입할 수 있습니다. 자세한 내용은 아래의 "패키지 특정 컴파일러 지시어"를 참조하십시오.

명령줄에서 컴파일하면 몇 가지 패키지 특정 스위치를 사용할 수 있습니다. 자세한 내용은 3-12페이지의 "명령줄 컴파일러와 링커 사용"을 참조하십시오.

패키지 특정 컴파일러 지시어

다음 표는 소스 코드에 삽입할 수 있는 패키지 특정 컴파일러 지시어를 나열한 것입니다.

표 3.3 패키지 특정 컴파일러 지시어

지시어	용도
{$IMPLICITBUILD OFF}	나중에 패키지가 암시적으로 다시 컴파일되지 않게 합니다. 저수준(low-level) 기능을 제공하고 빌드 사이에서 드물게 변경되거나 소스 코드가 배포되지 않을 패키지를 컴파일할 때 .dpk 파일에서 사용합니다.
{$G-} 또는 {IMPORTEDDATA OFF}	import한 데이터 참조를 생성할 수 없습니다. 이 지시어는 메모리 액세스 효율을 높이지만 지시어가 있는 유닛은 다른 패키지에서 변수를 참조할 수 없습니다.
{$WEAKPACKAGEUNIT ON}	유닛을 "약하게" 패키지화합니다. 3-12페이지의 "약한 패키징"을 참조하십시오.
{$DENYPACKAGEUNIT ON}	유닛을 패키지에 포함할 수 없습니다.
{$DESIGNONLY ON}	IDE에서 설치를 위해 패키지를 컴파일합니다. .dpk 파일에 삽입합니다.
{$RUNONLY ON}	패키지를 런타임 전용으로 컴파일합니다. .dpk 파일에 삽입합니다.

참고 소스 코드에 **{$DENYPACKAGEUNIT ON}**을 포함하면 유닛 파일을 패키지로 만들 수 없습니다. **{$G-}** 또는 **{$IMPORTEDDATA OFF}**를 포함하면 패키지는 동일한 애플리케이션에서 다른 패키지와 함께 사용할 수 없습니다. **{$DESIGNONLY ON}** 지시어로 컴파일된 패키지는 IDE에 필요한 추가 코드가 포함되므로 일반적인 애플리케이션에서는 사용하지 마십시오. 적절한 경우 다른 컴파일러 지시어는 패키지 소스 코드에 포함시킬 수 있습니다. 여기에서 다루지 않은 컴파일러 지시어에 대한 내용은 온라인 도움말의 컴파일러 지시어를 참조하십시오.

약한 패키징

$WEAKPACKAGEUNIT** 지시어는 .dpu 파일을 패키지의 파일에 저장하는 방식에 영향을 줍니다. 컴파일러가 생성하는 파일에 대한 내용은 3-13페이지의 "컴파일이 성공했을 때 생성되는 패키지 파일"을 참조하십시오. **{$WEAKPACKAGEUNIT ON}이 유닛 파일에 나타나면 컴파일러는 가능한 경우 유닛을 패키지에서 생략하고 다른 애플리케이션이나 패키지가 요구하면 유닛의 패키지화되지 않은 로컬 복사본을 생성합니다. 이 지시어로 컴파일된 유닛을 "약하게 패키지화"되었다고 합니다.

예를 들어, unit1이라는 하나의 유닛만을 포함하는 **pack**이라는 패키지를 만들었다고 가정합니다. unit1은 다른 유닛을 사용하지 않지만 rare.so를 호출한다고 가정합니다. 패키지를 컴파일할 때 **{$WEAKPACKAGEUNIT ON}**을 unit1.pas에 삽입하면 unit1은 bplpack.so에 포함(include)되지 않습니다. rare.so의 복사본을 pack과 함께 배포할 필요는 없습니다. 그러나 unit1은 여전히 pack.dcp에 포함됩니다. pack을 사용하는 다른 패키지나 애플리케이션이 unit1을 참조하면 pack.dcp에서 unit1을 복사하고 프로젝트에 직접 컴파일합니다.

이제 두 번째 유닛인 unit2를 pack에 추가한다고 가정합니다. unit2가 unit1을 사용한다고 가정합니다. 이번에는 unit1.pas에 **{$WEAKPACKAGEUNIT ON}**으로 pack을 컴파일하면 컴파일러는 unit1을 bplpack.so에 포함(include)합니다. 그러나 unit1을 참조하는 다른 패키지나 애플리케이션은 pack.dcp에서 가져온 패키지화되지 않은 복사본을 사용합니다.

참고 **{$WEAKPACKAGEUNIT ON}** 지시어가 포함된 유닛 파일에는 전역 변수, 초기화 섹션 또는 완료 섹션이 없어야 합니다.

$WEAKPACKAGEUNIT 지시어는 다른 Kylix 프로그래머에게 패키지를 배포하는 개발자를 위한 고급 기능입니다. 이 지시어를 사용하면 드물게 사용되는 공유 객체 파일의 배포를 막을 수 있고, 동일한 외부 라이브러리에 의존하는 패키지 간의 충돌을 없앨 수 있습니다.

명령줄 컴파일러와 링커 사용

명령줄에서 컴파일하는 경우 다음 표에서 나열하는 패키지 특정 스위치를 사용할 수 있습니다.

표 3.4 패키지 특정 명령줄 컴파일러 스위치

스위치	용도
-$G-	import한 데이터 참조를 생성할 수 없습니다. 이 스위치를 사용하면 메모리 액세스 효율을 증가시키지만 이 스위치를 사용하여 컴파일된 패키지는 다른 패키지의 변수를 참조할 수 없습니다.
-LE*path*	패키지(bcp*package*.so) 파일을 저장할 디렉토리를 지정합니다.
-LN*path*	패키지(*package*.dcp) 파일을 저장할 디렉토리를 지정합니다.
-LU*package*	패키지를 사용합니다.
-Z	나중에 패키지가 암시적으로 다시 컴파일되지 않게 합니다. 저수준(low-level)의 기능을 제공하고 빌드 사이에서 드물게 변경되거나 소스 코드가 배포되지 않을 패키지를 컴파일할 때 사용합니다.

참고 **-$G-** 스위치를 사용하면 패키지를 동일한 애플리케이션에서 다른 패키지와 함께 사용할 수 없습니다. 적절한 경우 다른 명령줄 옵션을 패키지를 컴파일할 때 사용할 수도

있습니다. 여기에서 설명하지 않은 명령줄 옵션에 대한 내용은 온라인 도움말의 "Command-line compiler"를 참조하십시오.

컴파일이 성공했을 때 생성되는 패키지 파일

패키지를 생성하려면 확장자가 .dpk인 소스 파일을 컴파일합니다. .dpk 파일의 기본(base) 이름은 컴파일러가 생성한 파일의 기본 이름이 됩니다. 예를 들어, traypak.dpk라는 패키지 소스 파일을 컴파일하면 컴파일러는 bpltraypak.so라는 패키지를 생성합니다.

다음 테이블은 패키지 컴파일이 성공했을 때 생성되는 파일을 보여 줍니다.

표 3.5 컴파일된 패키지 파일

파일명 또는 확장자	내용
dcp	패키지 헤더를 포함하는 바이너리 이미지와 패키지의 모든 dpu 파일의 연결. 패키지마다 하나의 dcp 파일이 생성됩니다. dcp에 대한 기본 이름은 dpk 소스 파일의 기본(base) 이름입니다.
dpu	패키지에 포함된 유닛 파일에 대한 바이너리 이미지. 각 유닛 파일에 대해 필요한 경우 dpu 하나가 생성됩니다
bpl*package*.so	런타임 패키지. 이 파일은 특수한 Kylix 특정 기능을 가진 공유 객체 파일입니다. 이름에서 *package*는 dpk 소스 파일의 기본(base) 이름입니다. 접두사(예: bpl), 접미사, 개정 번호는 패키지의 프로젝트 옵션에서 수정할 수 있습니다.

컴파일할 때 패키지와 라이브러리 파일은 기본적으로 Tools|Environment Options 대화 상자의 Library 페이지에서 지정한 디렉토리에 생성됩니다. Package 에디터의 Options 스피드 버튼을 클릭하여 Project Options 대화 상자가 표시되면 Directories/Conditionals 페이지를 변경하여 기본 설정을 오버라이드할 수 있습니다.

패키지 배포

다른 애플리케이션을 배포하는 것과 같은 방식으로 패키지를 배포합니다. 일반적인 배포 정보에 대한 내용은 *Kylix1 개발자 안내서*의 13장 "애플리케이션 배포"를 참조하십시오.

패키지를 사용하는 애플리케이션 배포

런타임 패키지를 사용하는 애플리케이션을 배포하는 경우 사용자가 애플리케이션이 호출하는 모든 라이브러리 파일과 더불어 애플리케이션의 실행 파일도 있는지 확인합니다. 라이브러리 파일이 실행 파일과 다른 디렉토리에 있으면 사용자의 경로를 통해 액세스할 수 있어야 합니다.

다른 개발자에게 패키지 배포

런타임 패키지나 디자인 타임 패키지를 다른 Kylix 개발자에게 배포하는 경우 .dcp와 .so 파일을 제공해야 합니다. .dpu 파일도 포함시킬 수 있습니다.

참고 관리자(root)가 새 패키지를 Kylix에 추가한 경우 사용자는 .borland 디렉토리에서 delphi65rc 파일을 제거해야만 패키지를 볼 수 있습니다.

II

데이터베이스 애플리케이션 개발

"데이터베이스 애플리케이션 개발"에 포함된 장에서는 Kylix 데이터베이스 애플리케이션을 만드는 데 필요한 개념 및 기술을 설명합니다.

참고 Kylix의 일부 버전에서는 데이터베이스 컴포넌트를 사용할 수 없습니다.

4

웹 서비스를 사용하여 다계층 데이터베이스 애플리케이션 생성

* 이 장은 영문 Kylix2 개발자 안내서의 22장입니다.

이 장에서는 다계층 클라이언트/서버 데이터베이스 애플리케이션을 만드는 방법에 대해 설명합니다. 다계층 클라이언트/서버 애플리케이션은 계층이라는 논리 단위로 분할되어 개별 시스템 상에서 결합하여 실행됩니다. 다계층 애플리케이션은 LAN 또는 인터넷 상에서 서로 데이터를 공유하고 통신합니다. 다계층 애플리케이션은 중앙 집중화된 비즈니스 로직 및 씬(thin) 클라이언트 애플리케이션과 같은 여러 가지 이점을 제공합니다.

간단하게 "3계층 모델(three-tiered model)"이라는 다계층 애플리케이션은 다음 세 가지 계층으로 나뉩니다.

- **클라이언트 애플리케이션**: 사용자의 컴퓨터에서 사용자 인터페이스를 제공합니다.
- **애플리케이션 서버**: 모든 클라이언트에 액세스할 수 있는 중앙 네트워킹 위치에 상주하며 일반적인 데이터 서비스를 제공합니다.
- **원격 데이터베이스 서버**: 관계형 데이터베이스 관리 시스템(RDBMS)을 제공합니다.

이러한 3계층 모델에서 애플리케이션 서버는 클라이언트와 원격 데이터베이스 서버 간의 데이터의 흐름을 관리하므로 "데이터 브로커"라고도 합니다. 사용자 소유의 데이터베이스 백엔드(back end)를 생성할 수도 있지만 Kylix에서는 일반적으로 애플리케이션 서버와 그 클라이언트만을 생성합니다.

복잡한 다계층 애플리케이션에서는 추가 서비스가 클라이언트와 원격 데이터베이스 서버 간에 상주합니다. 예를 들어 안전한 인터넷 트랜잭션을 처리하는 보안 서비스 브로커나 다른 플랫폼의 데이터베이스와의 데이터 공유를 처리하는 브리지(bridge) 서비스가 있을 수도 있습니다.

다계층 애플리케이션 개발을 지원하기 위해 Kylix는 클라이언트 데이터셋이 전송 가능한 데이터 패킷을 사용하여 프로바이더 컴포넌트와 통신하는 방법을 확장해 줍니다. 이 장에서는 3계층 데이터베이스 애플리케이션 생성에 초점을 맞춥니다. 3계층 애플리케이션을 생성하고 관리 방법을 이해하면 필요에 따라 서비스 레이어를 만들거나 추가할 수 있습니다.

다계층 데이터베이스 모델의 이점

다계층 데이터베이스 모델은 데이터베이스 애플리케이션을 논리 단위로 분할합니다. 클라이언트 애플리케이션은 데이터 표시와 사용자 상호 작용에 초점을 맞출 수 있습니다. 이론적으로 클라이언트 애플리케이션에서는 데이터가 저장되거나 유지되는 방법을 몰라도 됩니다. 애플리케이션 서버(중간 계층)는 여러 클라이언트로부터의 요청과 업데이트를 조정하고 처리합니다. 또한 데이터셋 정의와 데이터베이스 서버와의 상호 작용에 대한 모든 세부 사항을 처리합니다.

이러한 다계층 모델의 이점은 다음과 같습니다.

- **공유된 중간 계층의 비즈니스 로직 캡슐화**. 서로 다른 클라이언트 애플리케이션들이 모두 동일한 중간 계층에 액세스합니다. 이를 통해 별도의 각 클라이언트 애플리케이션에 대해 비즈니스 룰을 복제해야 하는 중복성(및 유지 비용)을 피할 수 있습니다.
- **씬 클라이언트 애플리케이션**. 처리 작업을 중간 계층에 대해 더 많이 위임함으로써 가벼운 클라이언트 애플리케이션을 작성할 수 있습니다. 클라이언트 애플리케이션은 크기가 작을 뿐만 아니라 데이터베이스 연결 소프트웨어(예를 들어, 데이터베이스 서버의 클라이언트측 소프트웨어)를 설치, 구성 및 유지 관리할 필요가 없으므로 쉽게 배포할 수 있습니다. 씬 클라이언트 애플리케이션은 유연성을 갖추고 있어 인터넷을 통해 배포할 수 있습니다.
- **분산 데이터 처리**. 여러 컴퓨터에 애플리케이션의 작업을 분산시키면 로드 밸런싱으로 인해 성능이 향상되고 서버 중단 시 여분의 시스템에서 작업을 대신할 수 있습니다.
- **보안 강화**. 민감하고 중요한 기능을 다른 액세스 제한을 갖는 계층으로 분리할 수 있습니다. 이는 유연하고 구성 가능한 보안 수준을 제공합니다. 중간 계층은 민감한 자원에 대한 엔트리 포인트를 제한할 수 있으므로 액세스를 쉽게 제어할 수 있습니다.

다계층 데이터베이스 애플리케이션 이해

Kylix는 컴포넌트 팔레트의 WebServices 페이지 및 Data Access 페이지에 있는 컴포넌트를 사용하는 다계층 애플리케이션을 지원하고, New Items 대화 상자의 Multitier 페이지에서 마법사에 의해 생성된 원격 데이터 모듈을 지원합니다. 다계층 애플리케이션에서 사용된 컴포넌트는 데이터를 전송 가능한 데이터 패킷으로 패키지화하고 전송 가능한 델타 패킷으로 수신한 업데이트를 처리하는 프로바이더 컴포넌트의 기능에 기반을 둡니다.

다계층 애플리케이션에 필요한 컴포넌트는 표 4.1에서 설명합니다.

표 4.1 다계층 애플리케이션에 사용되는 컴포넌트

컴포넌트	설명
원격 데이터 모듈	클라이언트 애플리케이션에 들어 있는 모든 프로바이더에 액세스를 제공하기 위해 SOAP 서버로서 작동할 수 있는 특수한 데이터 모듈입니다. 애플리케이션 서버에서 사용됩니다.
프로바이더 컴포넌트	데이터 패킷을 생성하여 데이터를 제공하고 클라이언트 업데이트를 해결하는 데이터 브로커입니다. 애플리케이션 서버에서 사용됩니다.

표 4.1 다계층 애플리케이션에 사용되는 컴포넌트(계속)

컴포넌트	설명
클라이언트 데이터셋 컴포넌트	midas.so 또는 midaslib.dcu를 사용하여 데이터 패킷에 저장된 데이터를 관리하는 특수한 데이터셋입니다. 클라이언트 데이터셋은 클라이언트 애플리케이션에 사용됩니다. 클라이언트 데이터셋은 로컬로 업데이트를 캐시하며 델타 패킷의 업데이트를 애플리케이션 서버에 적용합니다.
연결 컴포넌트	서버를 찾고, 연결을 만들고, *IAppServer* 인터페이스를 클라이언트 데이터셋에 사용 가능하게 하는 컴포넌트 패밀리입니다. 각 연결 컴포넌트는 특정 통신 프로토콜을 사용하도록 특수화되었습니다.

프로바이더와 클라이언트 데이터셋 컴포넌트에는 데이터 패킷으로 저장된 데이터셋을 관리하는 midas.so 또는 midaslib.dcu가 필요합니다. (프로바이더는 애플리케이션 서버에 사용되고 클라이언트 데이터셋은 클라이언트 애플리케이션에 사용되므로 midas.so를 사용하는 경우 애플리케이션 서버와 클라이언트 애플리케이션에 모두 배포해야 합니다.)

참고 애플리케이션 서버를 배포하기 위해서는 서버 라이센스를 구입해야 합니다.

이러한 컴포넌트가 적용될 아키텍처에 대한 개요는 *Kylix1 개발자 안내서* 14-12페이지의 "다계층 아키텍처 사용"에서 설명합니다.

3계층 애플리케이션 개요

다음 단계들은 프로바이더 방식의 3계층 애플리케이션에 대한 이벤트의 일반적인 순서를 나타낸 것입니다.

1 사용자는 클라이언트 애플리케이션을 시작합니다. 클라이언트는 디자인 타임이나 런타임 시 지정할 수 있는 애플리케이션 서버에 연결합니다. 아직 실행 중이 아니면 애플리케이션 서버를 시작합니다. 클라이언트는 애플리케이션 서버에서 *IAppServer* 인터페이스를 수신합니다.

2 클라이언트는 애플리케이션 서버의 데이터를 요청합니다. 클라이언트는 한 번에 모든 데이터를 요청하거나 세션에 걸쳐 데이터 일부를 요청(요구 즉시 fetch: fetch on demand)할 수도 있습니다.

3 애플리케이션 서버는(필요한 경우 먼저 데이터베이스에 연결하여) 데이터를 가져오고, 클라이언트를 위해 데이터를 패키지화하고, 데이터 패킷을 클라이언트에 반환합니다. 필드 표시 특징과 같은 추가 정보는 데이터 패킷의 메타데이터에 포함시킬 수 있습니다. 데이터 패킷에 데이터를 패키지화하는 프로세스를 "공급 프로세스"라고 합니다.

4 클라이언트는 데이터 패킷을 디코딩하고 데이터를 사용자에게 표시합니다.

5 사용자가 클라이언트 애플리케이션과 상호 작용하면서 데이터가 업데이트(레코드 추가, 삭제, 수정)됩니다. 이러한 수정은 클라이언트의 변경 로그에 저장됩니다.

6 그런 다음 클라이언트는 일반적으로 사용자 동작에 응답하여 해당 업데이트를 애플리케이션 서버에 적용합니다. 업데이트를 적용하기 위해서 클라이언트는 변경 로그를 패키지화하고 그 변경 로그를 서버에 데이터 패킷으로 보냅니다.

7 애플리케이션 서버는 패키지를 디코딩하고 적절한 경우 트랜잭션에 업데이트 내용을 포스트합니다. 예를 들어 클라이언트가 레코드를 요청하고 나서 해당 업데이트를 적용하기 전에 다른 애플리케이션에서 그 레코드를 변경했기 때문에 레코드를 포스트할 수 없는 경우, 애플리케이션 서버는 클라이언트의 변경 내용을 현재 데이터로 조정을 시도하거나 또는 포스트할 수 없는 레코드를 저장합니다. 레코드를 포스트하고 문제가 되는 레코드를 캐시하는 프로세스를 "해결 프로세스"라고 합니다.

8 애플리케이션 서버가 해결 프로세스를 완료하면 추가 해결을 위해 포스트되지 않은 레코드를 클라이언트에 반환합니다.

9 클라이언트는 해결되지 않은 레코드를 조정합니다. 클라이언트가 해결되지 않은 레코드를 조정할 수 있는 여러 가지 방법이 있습니다. 대개 클라이언트는 레코드 포스트를 막거나 변경 사항을 무시하는 상황을 고치려고 합니다. 오류 상황이 정정되면 클라이언트는 업데이트를 다시 적용합니다.

10 클라이언트는 서버의 데이터를 새로 고칩니다.

클라이언트 애플리케이션의 구조

다계층 애플리케이션의 클라이언트 애플리케이션이 캐시된 업데이트를 사용하는 기존의 2계층 애플리케이션과 비교해 보면 최종 사용자에게는 외관 및 동작에 그다지 차이가 없는 것처럼 보입니다. 사용자 상호 작용은 *TClientDataSet* 컴포넌트의 데이터를 표시하는 표준 data-aware 컨트롤을 통해 발생합니다. 클라이언트 데이터셋의 속성, 이벤트 및 메소드 사용에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 20장 "클라이언트 데이터셋 사용"을 참조하십시오.

*TClientDataSet*은 외부 프로바이더가 있는 클라이언트 데이터셋을 사용하는 2계층 애플리케이션에서처럼 프로바이더 컴포넌트에서 데이터를 fetch하고 업데이트를 프로바이더 컴포넌트에 적용합니다. 프로바이더에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 21장 "프로바이더 컴포넌트 사용"을 참조하십시오. 프로바이더와 쉽게 통신할 수 있도록 해주는 클라이언트 데이터셋 기능에 대한 자세한 내용은 *Kylix1 개발자 안내서* 20-25페이지의 "프로바이더와 함께 클라이언트 데이터셋 사용"을 참조하십시오.

클라이언트 데이터셋은 *IAppServer* 인터페이스를 통해 프로바이더와 통신합니다. 클라이언트 데이터셋은 연결 컴포넌트에서 이 인터페이스를 얻습니다. *TSoapConnection* 연결 컴포넌트는 애플리케이션 서버에 연결을 만듭니다.

참고 Kylix는 애플리케이션 서버에 연결되지 않는 연결 컴포넌트를 포함하지만 클라이언트 데이터셋이 동일한 애플리케이션에서 프로바이더와 통신할 때 사용할 수 있는 *IAppServer* 인터페이스를 제공합니다. 이 *TLocalConnection* 컴포넌트는 꼭 필요한 것은 아니지만 나중에 다계층 애플리케이션으로 쉽게 확장할 수 있게 해줍니다.

연결 컴포넌트 사용에 대한 자세한 내용은 4-12페이지의 "애플리케이션 서버에 연결"을 참조하십시오.

애플리케이션 서버의 구조

애플리케이션 서버를 설정하고 실행할 때는 클라이언트 애플리케이션과 연결되지 않습니다. 그 대신 클라이언트 애플리케이션에 의해 연결이 초기화되고 유지됩니다. 클라이

언트 애플리케이션은 자신의 연결 컴포넌트를 사용하여 선택한 프로바이더와 통신하는 데 사용할 애플리케이션 서버에 연결합니다. 이 모든 작업은 수신 요청을 관리하고 인터페이스를 제공하는 코드를 작성할 필요 없이 자동으로 수행됩니다.

애플리케이션 서버에서는 *IAppServer* 인터페이스를 지원하는 특수한 데이터 모듈인 원격 데이터 모듈이 기반이 됩니다. 클라이언트 애플리케이션은 *IAppServer* 인터페이스를 사용하여 애플리케이션 서버에서 프로바이더와 통신합니다.

TSoapDataModule 클래스는 웹 서비스 애플리케이션의 호출 가능한 인터페이스로서 *IAppServer* 자손을 구현하는 원격 데이터 모듈입니다.

원격 데이터 모듈의 컨텐트

다른 데이터 모듈과 마찬가지로 원격 데이터 모듈에 넌비주얼 컴포넌트를 포함시킬 수 있습니다. 반드시 포함해야 하는 컴포넌트는 다음과 같습니다.

- 원격 데이터 모듈이 데이터베이스 서버의 정보를 노출하는 경우, 해당 데이터베이스 서버의 레코드를 나타내는 데이터셋 컴포넌트를 포함해야 합니다. 데이터셋이 데이터베이스 서버와 상호 작용할 수 있도록 데이터베이스 연결 컴포넌트 같은 일부 타입의 다른 컴포넌트가 필요할 수도 있습니다. 데이터셋에 대한 내용은 *Kylix1 개발자 안내서*의 16장 "데이터셋 이해"를 참조하십시오. 데이터베이스 연결 컴포넌트에 대한 내용은 *Kylix1 개발자 안내서*의 19장 "데이터베이스에 연결"을 참조하십시오.

  원격 데이터 모듈이 클라이언트에 노출하는 모든 데이터셋은 데이터셋 프로바이더를 포함해야 합니다. 데이터셋 프로바이더는 클라이언트 데이터셋에 보내는 데이터 패킷으로 데이터를 패키지화하고 클라이언트 데이터셋에서 수신한 업데이트를 소스 데이터셋 또는 데이터베이스 서버에 다시 적용합니다. 데이터셋 프로바이더에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 21장 "프로바이더 컴포넌트 사용"을 참조하십시오.
- 원격 데이터 모듈이 클라이언트에 노출시키는 모든 XML 문서는 XML 프로바이더를 포함해야 합니다. XML 프로바이더는 데이터를 fetch하고 업데이트를 데이터베이스 서버가 아닌 XML 문서에 적용한다는 점을 제외하고는 데이터셋 프로바이더처럼 동작합니다.

참고 데이터셋을 데이터베이스 서버에 연결하는 데이터베이스 연결 컴포넌트와 다계층 애플리케이션의 클라이언트 애플리케이션에서 사용하는 연결 컴포넌트를 혼동하지 마십시오. 다계층 애플리케이션에서 사용되는 *TSoapConnection* 컴포넌트는 컴포넌트 팔레트의 WebServices 페이지에 있습니다.

SOAP 연결 사용

SOAP는 웹 서비스 애플리케이션에 대해 Kylix에서 지원하는 프로토콜입니다. SOAP는 XML 인코딩을 사용하여 메소드 호출을 마샬링합니다. SOAP 연결은 전송 프로토콜로 HTTP를 사용합니다.

SOAP 연결은 Windows와 Linux에서 모두 지원되므로 크로스 플랫폼 애플리케이션에서 작동한다는 장점이 있습니다. HTTP는 모든 클라이언트가 사용할 수 있는 최저의 공통 분모를 제공하며 클라이언트는 "방화벽"으로 보호되는 애플리케이션 서버와도 통신할 수 있습니다. SOAP를 통한 Kylix에서의 애플리케이션 배포에 대한 자세한 내용은 10장 "웹 서비스 사용"을 참조하십시오.

SOAP를 통해 연결이 구성된 경우 콜백을 사용할 수 없습니다. SOAP 연결은 또한 애플리케이션 서버의 단일 원격 데이터 모듈로 제한됩니다.

다계층 애플리케이션 생성

다계층 데이터베이스 애플리케이션을 생성하는 일반적인 단계는 다음과 같습니다.

1 애플리케이션 서버를 생성합니다.

2 애플리케이션 서버를 등록하거나 설치합니다.

3 클라이언트 애플리케이션을 생성합니다.

생성 순서가 중요합니다. 클라이언트를 생성하기 전에 애플리케이션 서버를 생성하여 실행해야 합니다. 그런 다음 디자인 타임에 애플리케이션 서버에 연결하여 클라이언트를 테스트할 수 있습니다. 물론 디자인 타임에 애플리케이션 서버를 지정하지 않고 클라이언트를 생성한 후 런타임 시 서버 이름을 입력해도 됩니다. 하지만 그렇게 하면 디자인 타임에 코드를 작성할 때 애플리케이션이 예상대로 작동하는지 알 수 없고 Object Inspector에서 서버 및 프로바이더를 선택할 수 없게 됩니다.

참고 *TSoapConnection* 컴포넌트는 서버 시스템으로부터 등록된 서버의 이름을 fetch합니다.

애플리케이션 서버 생성

대부분의 데이터베이스 애플리케이션을 생성하는 것과 거의 동일한 방식으로 애플리케이션 서버를 생성합니다. 중요한 차이점은 애플리케이션 서버는 원격 데이터 모듈을 사용한다는 점입니다.

다음의 단계에 따라 애플리케이션 서버를 생성합니다.

1 새 프로젝트를 시작합니다.

새 프로젝트는 웹 서비스 애플리케이션이어야 합니다. File|New를 선택하고 New Items 대화 상자의 WebServices 페이지에서 SOAP 서버 애플리케이션을 선택합니다.

새 프로젝트를 저장합니다.

2 새 원격 데이터 모듈을 프로젝트에 추가합니다. 메인 메뉴에서 File|New를 선택한 다음 New Items 대화 상자의 MultiTier 또는 WebServices 페이지에서 SOAP Data Module을 선택합니다.

원격 데이터 모듈 설정에 대한 자세한 내용은 4-7페이지의 "원격 데이터 모듈 설정"을 참조하십시오.

참고 원격 데이터 모듈은 간단한 데이터 모듈 이상의 기능을 갖습니다. 예를 들어, SOAP 데이터 모듈은 웹 서비스 애플리케이션에서 호출 가능한 인터페이스(invokable interface)를 구현합니다. 클라이언트 애플리케이션에 기능을 추가하면 이 인터페이스를 확장할 수 있습니다. 4-8페이지의 "애플리케이션 서버의 인터페이스 확장"을 참조하십시오.

3 데이터 모듈에 적절한 데이터셋 컴포넌트를 두고 데이터베이스 서버에 액세스하도록 설정합니다.

4 각 데이터셋의 데이터 모듈에 *TDataSetProvider* 컴포넌트를 놓습니다. 이 프로바이더는 클라이언트 요청을 중개하고 데이터를 패키지화하는 데 필요합니다. 각 프로바이더 컴포넌트의 *DataSet* 속성을 액세스할 데이터셋의 이름으로 설정합니다. 프로바이더에 대한 추가 속성을 설정할 수 있습니다. 프로바이더 설정에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 21장 "프로바이더 컴포넌트 사용"을 참조하십시오.

   XML 문서의 데이터를 사용하는 경우 데이터셋과 *TDataSetProvider* 컴포넌트를 사용하는 대신 *TXMLTransformProvider* 컴포넌트를 사용할 수 있습니다. *TXMLTransformProvider*를 사용하는 경우, 데이터를 제공하고 업데이트를 적용하는 XML 문서를 *XMLDataFile* 속성에 지정합니다.

5 이벤트, 공유 비즈니스 룰, 공유 데이터 검증 및 공유 보안을 구현하도록 애플리케이션 서버 코드를 작성합니다. 코드를 작성할 때 다음과 같은 작업을 할 수 있습니다.

   - 애플리케이션 서버의 인터페이스를 확장하여 클라이언트 애플리케이션이 서버를 호출하는 방법을 추가합니다. 4-8페이지의 "애플리케이션 서버의 인터페이스 확장"을 참조하십시오.
   - 업데이트 적용 시 자동으로 생성되는 트랜잭션의 범위를 넘는 트랜잭션 지원을 제공합니다. 다계층 데이터베이스 애플리케이션의 트랜잭션 지원에 대해서는 4-8페이지의 "다계층 애플리케이션의 트랜잭션 관리"에서 설명합니다.
   - 애플리케이션 서버의 데이터셋 간에 마스터/디테일 관계를 생성합니다. 마스터/디테일 관계에 대해서는 4-8페이지의 "마스터/디테일 관계 지원"에서 설명합니다.
   - 애플리케이션 서버가 stateless인지 확인합니다. 상태 정보 처리에 대해서는 4-9페이지의 "원격 데이터 모듈의 상태 정보 지원"에서 설명합니다.

6 애플리케이션 서버를 저장, 컴파일, 등록 또는 설치합니다. 애플리케이션 서버 등록에 대해서는 4-11페이지의 "애플리케이션 서버 등록"에서 설명합니다.

원격 데이터 모듈 설정

원격 데이터 모듈을 생성할 때 클라이언트 요청에 응답하는 방법을 나타내는 특정 정보를 제공해야 합니다. 필요한 원격 데이터 모듈 타입에 대한 내용은 4-4페이지의 "애플리케이션 서버의 구조"를 참조하십시오.

TSoapDataModule 구성

TSoapDataModule 컴포넌트를 애플리케이션에 추가하려면 File|New를 선택하고 New Items 대화 상자의 WebServices 페이지에서 SOAP Server Data Module을 선택합니다. SOAP 데이터 모듈 마법사가 나타납니다.

SOAP 데이터 모듈에 클래스 이름을 입력해야 합니다. 클래스 이름은 애플리케이션에서 생성하는 *TSoapDataModule* 자손의 기본(base) 이름입니다. 이 이름은 해당 클래스에 대한 인터페이스의 기본 이름이기도 합니다. 예를 들어 클래스 이름 *MyDataServer*를 지

정하면 마법사는 *TMyDataServer*를 선언하는 새 유닛, *IMyDataServer*를 구현하는 *TSoapDataModule*의 자손, *IAppServer* 자손을 생성합니다.

사용자 고유의 속성과 메소드를 추가하여 생성된 인터페이스와 *TSoapDataModule* 자손의 정의를 편집할 수 있습니다. 이러한 속성과 메소드는 자동으로 호출되지는 않지만 이름으로 새 인터페이스를 요청하는 클라이언트 애플리케이션에서 추가하는 속성과 메소드를 사용할 수 있습니다.

참고 새 *TSoapDataModule* 객체가 웹 서비스 애플리케이션에 추가되어야 합니다. *IAppServer* 인터페이스는 새 유닛의 초기화 섹션에 등록되는 호출 가능 인터페이스입니다. 이를 통해 메인 웹 모듈의 호출자 컴포넌트는 들어오는 모든 호출을 데이터 모듈에 전송할 수 있습니다.

애플리케이션 서버의 인터페이스 확장

클라이언트 애플리케이션은 원격 데이터 모듈의 인스턴스를 생성하거나 연결하여 애플리케이션 서버와 상호 작용합니다. 클라이언트 애플리케이션은 언제나 인터페이스를 기반으로 하여 애플리케이션 서버와 통신합니다.

원격 데이터 모듈의 인터페이스에 추가하여 클라이언트 애플리케이션에 대한 추가 지원을 제공할 수 있습니다. 이 인터페이스는 *IAppServer*의 자손이며 원격 데이터 모듈을 생성할 때 마법사에 의해 자동으로 생성됩니다.

TSoapDataModule 자손의 경우 서버 인터페이스를 직접 편집해야 합니다.

원격 데이터 모듈의 인터페이스에 추가한 후 원격 데이터 모듈의 구현에 추가된 속성과 메소드를 찾습니다. 새 메소드의 몸체를 작성하여 구현이 완료된 코드를 추가합니다.

클라이언트 애플리케이션은 연결 컴포넌트의 *AppServer* 속성을 사용하여 인터페이스 확장을 호출합니다. 자세한 내용은 4-14페이지의 "서버 인터페이스 호출"을 참조하십시오.

참고 *TSoapDataModule* 컴포넌트는 인터페이스 확장에서 콜백을 사용할 수 없습니다.

다계층 애플리케이션의 트랜잭션 관리

클라이언트 애플리케이션에서 애플리케이션 서버에 업데이트를 적용하면 프로바이더 컴포넌트는 업데이트 적용 및 트랜잭션의 오류 해결 프로세스를 자동으로 랩합니다. 이 트랜잭션은 문제 레코드의 수가 *ApplyUpdates* 메소드의 인수로 지정된 *MaxErrors* 값을 초과하지 않는 경우에 커밋됩니다. 그렇지 않을 경우에는 롤백됩니다.

마스터/디테일 관계 지원

클라이언트 애플리케이션의 클라이언트 데이터셋 간에 마스터/디테일 관계를 테이블 형식의 데이터셋을 사용하여 설정하는 것과 동일한 방식으로 생성할 수 있습니다. 이러한 마스터/디테일 관계 설정에 대한 자세한 내용은 *Kylix1 개발자 안내서* 20-10페이지의 "마스터/디테일 관계 표시"를 참조하십시오.

하지만 이 방법은 다음과 같은 두 가지 큰 단점이 있습니다.

- 디테일 테이블은 한 번에 하나의 디테일 셋만 사용하더라도 애플리케이션 서버로부터 모든 레코드를 fetch하고 저장해야 합니다. (이 문제는 매개변수를 사용하면 해결할 수 있습니다. 자세한 내용은 *Kylix1 개발자 안내서* 20-28페이지의 "매개변수로 레코드 제한"을 참조하십시오.)
- 클라이언트 데이터셋은 데이터셋 수준에서 업데이트를 적용하는데 마스터/디테일 업데이트는 다중 데이터셋으로 확장되므로 업데이트를 적용하기가 매우 어렵습니다. 데이터베이스 연결 컴포넌트를 사용하여 단일 트랜잭션의 다중 테이블에 대한 업데이트를 적용하는 2계층 환경에서도 마스터/디테일 폼에 업데이트를 적용하는 것은 까다롭습니다.

다계층 애플리케이션에서 중첩 테이블을 사용하여 마스터/디테일 관계를 나타내면 이러한 문제를 피할 수 있습니다. 데이터셋에서 제공하는 경우 이 작업을 수행하려면 애플리케이션 서버의 데이터셋 간에 마스터/디테일 관계를 설정합니다. 그런 다음 프로바이더 컴포넌트의 *DataSet* 속성을 마스터 테이블로 설정합니다. XML 문서를 사용하는 경우 중첩 테이블을 사용하여 마스터/디테일 관계를 나타내려면 중첩 디테일 셋을 정의하는 변환 파일을 사용합니다.

클라이언트가 프로바이더의 *GetRecords* 메소드를 호출하면 자동으로 데이터 패킷의 레코드에 디테일 데이터셋을 DataSet 필드로 포함시킵니다. 클라이언트가 프로바이더의 *ApplyUpdates* 메소드를 호출하면 자동으로 업데이트 적용이 적절한 순서로 처리됩니다.

원격 데이터 모듈의 상태 정보 지원

애플리케이션 서버에서 클라이언트 데이터셋과 프로바이더 간에 모든 통신을 제어하는 *IAppServer* 인터페이스는 대부분 stateless입니다. 애플리케이션이 stateless이면 클라이언트의 이전 호출에서 발생한 것을 "기억"하지 않습니다. 이러한 stateless 특성은 애플리케이션 서버에서 레코드 Currency와 같은 영구적 정보에 대한 데이터베이스 연결 간에 구별할 필요가 없으므로 트랜잭션 데이터 모듈에서 데이터베이스 연결을 풀링할 때 유용합니다. stateless 특성은 just-in-time 활성화, 객체 풀링에서 발생하므로 여러 클라이언트 간에 원격 데이터 모듈 인스턴스를 공유할 때에도 중요합니다. SOAP 데이터 모듈은 stateless여야 합니다.

하지만 애플리케이션 서버를 여러 번 호출하는 동안의 상태 정보를 유지해야 할 경우가 있습니다. 예를 들어, 점진적 fetch(incremental fetching)를 사용하는 데이터 요청 시 애플리케이션 서버의 프로바이더는 이전 호출의 정보(현재 레코드)를 "기억"해야 합니다.

클라이언트 데이터셋이 만드는 *IAppServer* 인터페이스(AS_*ApplyUpdates*, AS_*Execute*, AS_*GetParams*, AS_*GetRecords* 또는 AS_*RowRequest*)를 호출하기 전후에 고객 상태 정보를 가져오거나 보낼 수 있는 이벤트를 수신합니다. 마찬가지로 프로바이더가 이러한 클라이언트가 생성한 호출에 응답하기 전후에 고객 상태 정보를 가져오거나 보낼 수 있는 이벤트를 수신합니다. 이 메커니즘을 사용하면 애플리케이션 서버가 stateless인 경우에도 클라이언트 애플리케이션과 애플리케이션 서버 간에 영구적인 상태 정보를 주고받을 수 있습니다.

예를 들어 다음과 같은 매개변수화된 쿼리를 나타내는 데이터셋을 가정합니다.

```
SELECT * from CUSTOMER WHERE CUST_NO > :MinVal ORDER BY CUST_NO
```

stateless 애플리케이션 서버에서 점진적 fetch(incremental fetching)를 활성화하려면 다음을 수행합니다.

- 프로바이더가 데이터 패킷의 레코드 집합을 패키지화하면 패킷의 최종 레코드에 CUST_NO의 값이 기록됩니다.

```
TRemoteDataModule1.DataSetProvider1GetData(Sender: TObject; DataSet: TCustomClientDataSet);
begin
  DataSet.Last; { move to the last record }
  with Sender as TDataSetProvider do
    Tag := DataSet.FieldValues['CUST_NO']; {save the value of CUST_NO }
end;
```

- 프로바이더는 데이터 패킷을 보낸 후 CUST_NO 최종 값을 클라이언트에게 보냅니다.

```
TRemoteDataModule1.DataSetProvider1AfterGetRecords(Sender: TObject;
                  var OwnerData: OleVariant);
begin
  with Sender as TDataSetProvider do
    OwnerData := Tag; {send the last value of CUST_NO }
end;
```

- 클라이언트에서 클라이언트 데이터셋은 CUST_NO 최종 값을 저장합니다.

```
TDataModule1.ClientDataSet1AfterGetRecords(Sender: TObject; var OwnerData: OleVariant);
begin
  with Sender as TClientDataSet do
    Tag := OwnerData; {save the last value of CUST_NO }
end;
```

- 데이터 패킷을 fetch하기 전에 클라이언트는 수신한 CUST_NO의 최종 값을 보냅니다.

```
TDataModule1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
begin
  with Sender as TClientDataSet do
  begin
    if not Active then Exit;
    OwnerData := Tag; { Send last value of CUST_NO to application server }
  end;
end;
```

- 마지막으로 서버에서 프로바이더는 쿼리에서 최소 값으로 보낸 CUST_NO 최종 값을 사용합니다.

```
TRemoteDataModule1.DataSetProvider1BeforeGetRecords(Sender: TObject;
                   var OwnerData: OleVariant);
begin
  if not VarIsEmpty(OwnerData) then
    with Sender as TDataSetProvider do
      with DataSet as TSQLDataSet do
      begin
        Params.ParamValues['MinVal'] := OwnerData;
        Refresh; { force the query to reexecute }
    end;
end;
```

애플리케이션 서버 등록

클라이언트 애플리케이션에서 애플리케이션 서버를 찾거나 사용하려면 먼저 등록 또는 설치해야 합니다.

애플리케이션 서버가 SOAP를 사용하기 때문에 웹 서비스 애플리케이션이어야 합니다. 이 경우 수신 HTTP 메시지를 받을 수 있도록 웹 서버에 등록해야 합니다. 또한 Kylix를 사용하여 작성되지 않은 클라이언트가 애플리케이션의 모든 인터페이스에 액세스할 수 있도록 하기 위해서 애플리케이션의 호출 가능한 인터페이스(invokable interface)를 나타내는 WSDL 문서를 게시할 수 있습니다. 웹 서비스 애플리케이션의 WSDL 문서 export에 대한 내용은 10-7페이지의 "웹 서비스 애플리케이션에 대한 WSDL 문서 생성"을 참조하십시오.

클라이언트 애플리케이션 생성

대부분의 경우 다계층 클라이언트 애플리케이션 생성은 업데이트를 캐시하는 데 클라이언트 데이터셋을 사용하는 2계층 클라이언트 생성과 유사합니다. 주요 차이점은 다계층 클라이언트는 연결 컴포넌트를 사용하여 애플리케이션 서버에 연결한다는 것입니다.

다계층 클라이언트 애플리케이션을 생성하려면 새 프로젝트를 시작하고 다음 단계를 수행합니다.

1. *TSoapDataModule* 객체를 프로젝트에 추가합니다.
2. 데이터 모듈에 *TSoapConnection* 컴포넌트를 둡니다. 자세한 내용은 4-4페이지의 "클라이언트 애플리케이션의 구조"를 참조하십시오.
3. 연결 컴포넌트에 대한 속성을 설정하여 연결을 설정할 애플리케이션 서버를 지정합니다. 연결 컴포넌트 설정에 대한 자세한 내용은 4-12페이지의 "애플리케이션 서버에 연결"을 참조하십시오.
4. 애플리케이션에 필요한 경우 다른 연결 컴포넌트 속성을 설정합니다. 예를 들어 연결 컴포넌트가 여러 서버로부터 동적으로 선택하도록 *ObjectBroker* 속성을 설정할 수도 있습니다. 연결 컴포넌트 사용에 대한 자세한 내용은 4-13페이지의 "서버 연결 관리"를 참조하십시오.
5. 데이터 모듈에 필요한 만큼 *TClientDataSet* 컴포넌트를 두고 각 컴포넌트의 *RemoteServer* 속성을 2단계에서 가져다 놓은 연결 컴포넌트의 이름으로 설정합니다. 클라이언트 데이터셋에 대한 전체적인 소개는 *Kylix1 개발자 안내서*의 20장 "클라이언트 데이터셋 사용"을 참조하십시오.
6. 각 *TClientDataSet* 컴포넌트의 *ProviderName* 속성을 설정합니다. 디자인 타임에 연결 컴포넌트가 애플리케이션 서버에 연결된 경우 *ProviderName* 속성의 드롭다운 목록에서 사용 가능한 애플리케이션 서버 프로바이더를 선택할 수 있습니다.

7 다른 데이터베이스 애플리케이션을 생성하는 것과 동일한 방식으로 계속합니다. 다계층 애플리케이션의 클라이언트가 사용할 수 있는 몇 가지 추가 기능은 다음과 같습니다.

- 작성한 애플리케이션에서 애플리케이션 서버를 직접 호출할 수 있습니다. 이 내용은 4-14페이지의 "서버 인터페이스 호출"에서 설명합니다.
- 프로바이더 컴포넌트와의 상호 작용을 지원하는 클라이언트 데이터셋의 특별한 기능을 사용할 수 있습니다. 이 내용은 *Kylix1 개발자 안내서* 20-25페이지의 "프로바이더와 함께 클라이언트 데이터셋 사용"에서 설명합니다.

애플리케이션 서버에 연결

애플리케이션 서버에 대한 연결을 설정 및 유지하기 위해 클라이언트 애플리케이션은 하나 이상의 *TSoapConnection* 컴포넌트를 사용합니다. 컴포넌트 팔레트의 WebServices 페이지에 이러한 컴포넌트가 있습니다.

연결 컴포넌트를 사용하여 다음을 수행합니다.

- 서버 컴퓨터를 찾는 방법을 표시합니다.
- *TSoapConnection* 컴포넌트의 *URL* 속성을 사용하여 서버를 식별합니다.
- 서버 연결을 관리합니다. 연결 컴포넌트를 사용하여 연결을 생성 또는 삭제하고 애플리케이션 서버 인터페이스를 호출할 수 있습니다.

일반적으로 애플리케이션 서버는 클라이언트 애플리케이션과 다른 컴퓨터에 있지만 서버가 클라이언트 애플리케이션과 동일한 컴퓨터에 상주하는 경우에도(예: 다계층 애플리케이션 전체를 빌드 및 테스트) 연결 컴포넌트를 사용하여 애플리케이션을 이름으로 식별하고, 서버 컴퓨터를 지정하며, 애플리케이션 서버 인터페이스를 사용할 수 있습니다.

SOAP를 사용하여 연결 지정

TSoapConnection 컴포넌트를 사용하여 SOAP 애플리케이션 서버에 연결할 수 있습니다. *TSoapConnection*은 전송 프로토콜로 HTTP를 사용합니다. 그러므로 TCP/IP 주소가 있는 컴퓨터에서 *TSoapConnection*을 사용할 수 있으며 SSL 보안을 이용하여 방화벽으로 보호되는 서버와 통신할 수 있습니다.

SOAP 연결 컴포넌트는 *IAppServer* 인터페이스를 구현하는 웹 서버 애플리케이션에 웹 서비스로 연결합니다. *TSoapConnection*은 URL(Uniform Resource Locator)을 사용하여 웹 서버 애플리케이션을 찾습니다. URL은 프로토콜(http, SSL 보안을 사용하는 경우에는 https), 웹 서버를 실행하는 컴퓨터의 호스트 이름, 웹 서비스 애플리케이션의 이름, 애플리케이션 서버에서 *THTTPSoapDispatcher*의 경로명과 일치하는 경로를 지정합니다. *URL* 속성을 사용하여 이 값을 지정합니다.

웹 서버에 인증이 필요한 경우 또는 인증이 필요한 프록시 서버를 사용하고 있을 경우, 연결 컴포넌트가 로그온할 수 있도록 *UserName* 속성 값과 *Password* 속성 값을 설정해야 합니다.

서버 연결 관리

애플리케이션 서버를 찾아 연결하는 것이 연결 컴포넌트의 주요한 용도입니다. 연결 컴포넌트는 서버 연결을 관리하므로 애플리케이션 서버 인터페이스의 메소드를 호출하는 데에도 연결 컴포넌트를 사용할 수 있습니다.

서버에 연결

애플리케이션 서버를 찾고 연결하려면 먼저 연결 컴포넌트의 속성을 설정하여 애플리케이션 서버를 식별해야 합니다. 이 프로세스는 4-12페이지의 "애플리케이션 서버에 연결"에서 다룹니다. 연결을 열기 전에 애플리케이션 서버와 통신하는 데 연결 컴포넌트를 사용하는 클라이언트 데이터셋은 연결 컴포넌트를 지정하는 *RemoteServer* 속성을 설정하여 나타내야 합니다.

클라이언트 데이터셋이 애플리케이션 서버에 대한 액세스를 시도하면 연결은 자동으로 열립니다. 예를 들어, 클라이언트 데이터셋의 *Active* 속성을 *True*로 설정하면 *RemoteServer* 속성이 설정되어 있는 한 연결이 열립니다.

연결 컴포넌트에 클라이언트 데이터셋을 연결하지 않은 경우 연결 컴포넌트의 *Connected* 속성을 *True*로 설정하여 연결을 열 수 있습니다.

연결 컴포넌트에서 애플리케이션 서버에 연결하기 전에 *BeforeConnect* 이벤트를 생성합니다. 코딩하는 *BeforeConnect* 핸들러로 연결하기 전에 사용자가 특별한 액션을 수행할 수 있습니다. 연결 컴포넌트는 연결 후 특별한 액션에 대한 *AfterConnect* 이벤트를 생성합니다.

서버 연결 끊기 및 바꾸기

연결 컴포넌트는 다음 작업을 수행할 때 애플리케이션 서버에 대한 연결을 끊습니다.

- *Connected* 속성을 *False*로 설정합니다.
- 연결 컴포넌트를 해제합니다. 연결 객체는 사용자가 클라이언트 애플리케이션을 종료하면 자동으로 해제됩니다.
- *TSoapConnection* 컴포넌트의 *URL* 속성을 변경합니다. 이러한 속성을 변경하면 런타임 시 사용 가능한 애플리케이션 서버 간에 전환할 수 있습니다. 연결 컴포넌트는 현재 연결을 끊고 새 연결을 설정합니다.

참고 단일 연결 컴포넌트를 사용하여 사용 가능한 애플리케이션 서버 간에 전환하는 대신 클라이언트 애플리케이션은 연결 컴포넌트를 두 개 이상 가질 수 있으며 각 연결 컴포넌트는 다른 애플리케이션 서버에 연결됩니다.

연결 컴포넌트는 연결을 끊기 전에 *BeforeDisconnect* 이벤트 핸들러를 자동으로 호출합니다. 연결을 끊기 전에 특정 액션을 수행하려면 *BeforeDisconnect* 핸들러를 작성합니다. 마찬가지로 연결을 끊은 후 *AfterDisconnect* 이벤트 핸들러가 호출됩니다. 연결을 끊은 후 특정 액션을 수행하려면 *AfterDisconnect* 핸들러를 작성합니다.

서버 인터페이스 호출

클라이언트 데이터셋의 속성과 메소드 사용 시 적절한 호출이 자동으로 만들어지므로 애플리케이션에서 *IAppServer* 인터페이스를 직접 호출할 필요가 없습니다. *IAppServer* 인터페이스를 직접 사용할 필요는 없지만 사용자 고유의 확장을 원격 데이터 모듈 인터페이스에 추가할 수 있습니다. 애플리케이션 서버의 인터페이스를 확장하는 경우 연결 컴포넌트에 의해 생성된 연결을 사용하여 해당 확장자를 호출하기 위한 방법이 필요합니다. 애플리케이션 서버의 인터페이스 확장에 대한 내용은 4-8페이지의 "애플리케이션 서버의 인터페이스 확장"을 참조하십시오.

확장을 호출하려면 원격 인터페이스 객체(*THTTPRio*)를 사용하여 우선 바운드 호출(early-bound call)을 만들 수 있습니다. 모든 우선 바운드 호출과 마찬가지로 클라이언트 애플리케이션에서 컴파일 타임 시 애플리케이션 서버의 인터페이스 선언을 알고 있어야 합니다. 서버에서 인터페이스를 선언 및 등록하는 데 사용하는 것과 동일한 유닛을 클라이언트의 uses 절에 추가하여 클라이언트 애플리케이션에 선언을 추가하거나 인터페이스를 설명하는 WSDL 문서를 참조할 수 있습니다. 서버 인터페이스를 설명하는 WSDL 문서 import에 대한 내용은 10-9페이지의 "WSDL 문서 import하기"를 참조하십시오.

참고 서버 인터페이스를 선언하는 유닛은 호출(invocation) 레지스트리에도 등록해야 합니다. 호출 가능한 인터페이스를 등록하는 방법에 대한 자세한 내용은 10-3페이지의 "호출 가능한 인터페이스 정의"를 참조하십시오.

일단 애플리케이션에서 인터페이스를 선언 및 등록하는 서버 유닛을 사용하거나 WSDL 문서를 import 해서 해당 유닛을 생성했다면 필요한 인터페이스에 대한 *THTTPRio*의 인스턴스를 생성합니다.

```
X := THTTPRio.Create(nil);
```

그런 다음 연결 컴포넌트에서 사용하는 URL을 원격 인터페이스 객체에 할당합니다.

```
X.URL := SoapConnection1.URL;
```

그러면 **as** 연산자를 사용하여 *THTTPRio*의 인스턴스를 애플리케이션 서버의 인터페이스에 타입 변환할 수 있습니다.

```
InterfaceVariable := X as IMyAppServer;
InterfaceVariable.SpecialMethod(x,y);
```

5

데이터베이스 애플리케이션에서 XML 사용

* 이 장은 영문 Kylix2 개발자 안내서의 23장입니다.

Kylix를 사용하면 데이터베이스 서버에 대한 연결이 지원될 뿐만 아니라 데이터베이스 서버를 사용하는 것처럼 XML 문서를 사용할 수 있습니다. XML(Extensible Markup Language)은 구조화된 데이터를 표현하는 마크업 랭귀지입니다. XML 문서는 웹 애플리케이션, B2B 통신 등에서 사용되는 데이터에 대해 전송 가능한 표준 형식을 제공합니다. XML 문서를 직접 사용하기 위해 Kylix에서 지원하는 내용은 9장 "XML 문서 사용"을 참조하십시오.

데이터베이스 애플리케이션에서 XML 문서 사용에 대한 Kylix의 지원은 데이터 패킷(클라이언트 데이터셋의 *Data* 속성)을 XML 문서로 변환하고 XML 문서를 데이터로 변환할 수 있는 컴포넌트 집합을 기반으로 합니다. 이러한 컴포넌트를 사용하려면 먼저 XML 문서와 데이터 패킷 사이의 변환을 정의해야 합니다. 일단 변환을 정의하면 특수 컴포넌트를 사용하여 다음 작업을 수행할 수 있습니다.

- XML 문서를 데이터 패킷으로 변환합니다.
- XML 문서로부터 데이터를 제공하고 업데이트를 수행합니다.
- XML 문서를 프로바이더의 클라이언트로 사용합니다.

변환 정의

데이터 패킷과 XML 문서 간을 변환하려면 데이터 패킷의 메타데이터와 해당 XML 문서 노드 사이의 관계를 정의해야 합니다. 이 관계에 대한 설명은 변환 파일이라는 특수한 XML 문서에 저장됩니다.

각 변환 파일에는 XML 스키마의 노드와 데이터 패킷 필드 사이의 매핑 및 변환 결과에 대한 구조를 나타내는 뼈대(skeletal) XML 문서라는 두 가지 항목이 포함됩니다. 변환은 XML 스키마나 문서에서 데이터 패킷으로 또는 데이터 패킷의 메타데이터에서 XML 스키마로 수행되는 단방향 매핑입니다. 변환 파일은 XML에서 데이터 패킷으로 매핑하

는 경우와 데이터 패킷에서 XML로 매핑하는 경우처럼 대개 변환 파일을 쌍으로 만듭니다.

매핑을 하기 위한 변환 파일을 만들려면 bin 디렉토리에 있는 XMLMapper 유틸리티를 사용하십시오.

XML 노드와 데이터 패킷 필드 간 매핑

XML은 구조화된 데이터를 저장하고 표현하기 위해 텍스트 방식을 사용합니다. 데이터셋은 구조화된 데이터를 저장하고 표현하기 위해 텍스트 방식이 아닌 다른 방식을 사용합니다. 그러므로 XML 문서를 데이터셋으로 변환하려면 XML 문서의 노드와 데이터셋의 필드 사이의 대응 관계를 식별해야 합니다.

예를 들어 전자 메일 메시지 모음을 나타내는 XML 문서를 가정해 보십시오. 메시지를 하나만 포함하고 있다면 다음과 유사한 형식일 것입니다.

```
<?xml version="1.0" standalone="yes" ?>
<email>
   <head>
      <from>
         <name>Dave Boss</name>
         <address>dboss@MyCo.com</address>
      </from>
      <to>
         <name>Joe Engineer</name>
         <address>jengineer@MyCo.com</address>
      <to>
      <cc>
         <name>Robin Smith/name>
         <address>rsmith@MyCo.com</address>
      </cc>
      <cc>
         <name>Leonard Devon</name>
         <address>ldevon@MyCo.com</address>
      </cc>
   </head>
   <body>
      <subject>XML components</subject>
      <content>
        Joe,
        Attached is the specification for the new XML component support in Kylix.
        This looks like a good solution to our business-to-business application!
        Also attached, please find the project schedule. Do you think its reasonable?
           Dave.
      </content>
      <attachment attachfile="XMLSpec.txt"/>
      <attachment attachfile="Schedule.txt"/>
   </body>
</email>
```

이 문서와 데이터셋 간의 자연스러운 매핑 방법 중 하나는 전자 메일 메시지마다 레코드를 하나씩 매핑하는 것입니다. 레코드에는 보낸 사람의 이름과 주소에 대한 필드가 있을 것입니다. 전자 메일 메시지는 받는 사람이 여럿이 될 수 있기 때문에 받는 사람(<to>)은 중첩

된 데이터셋에 매핑합니다. cc 목록 역시 받는 사람과 마찬가지로 중첩된 데이터셋에 매핑합니다. 메시지는 (<content>) 메모 필드에 매핑하고 제목줄은 문자열 필드에 매핑합니다. 메시지의 첨부 파일이 여러 개일 수 있기 때문에 첨부 파일의 이름은 중첩된 데이터셋에 매핑합니다. 따라서 이 전자 메일을 다음과 같은 데이터셋에 매핑할 수 있습니다.

SenderName	SenderAddress	To	CC	Subject	Content	Attach
Dave Boss	dboss@MyCo.Com	(DataSet)	(DataSet)	XML components	(MEMO)	(DataSet)

"To" 필드의 중첩된 데이터셋은 다음과 같습니다.

Name	Address
Joe Engineer	jengineer@MyCo.Com

"CC" 필드의 중첩된 데이터셋은 다음과 같습니다.

Name	Address
Robin Smith	rsmith@MyCo.Com
Leonard Devon	ldevon@MyCo.Com

"Attach" 필드의 중첩된 데이터셋은 다음과 같습니다.

Attachfile
XMLSpec.txt
Schedule.txt

매핑을 정의하는 것은 XML 문서의 반복 가능한 노드를 식별하고 이 노드들을 중첩된 데이터셋에 매핑하는 작업입니다. 값이 있고 <content>...</content>처럼 한 번만 나타나는 태그 요소(element)는 값으로 표시할 수 있는 데이터 타입을 나타내는 필드로 매핑됩니다. attachment 태그의 AttachFile 속성처럼 태그의 속성도 필드에 매핑됩니다.

XML 문서의 모든 태그를 해당 데이터셋으로 매핑하는 것은 아닙니다. 예를 들어 <head>...<head/> 요소에 해당하는 데이터셋은 없습니다. 일반적으로 값이 있는 요소, 반복될 수 있는 요소, 태그의 속성만 중첩된 데이터셋 필드를 비롯한 데이터셋의 필드에 매핑됩니다. XML 문서에서 자식 노드의 값을 조합하여 필드 값이 구성되는 경우 이 부모 노드는 예외적으로 필드에 매핑됩니다. 예를 들어 XML 문서에는 다음과 같은 태그 집합이 포함될 수 있습니다.

```
<FullName>
   <Title> Mr. </Title>
   <FirstName> John </FirstName>
   <LastName> Smith </LastName>
</FullName>
```

이 문서는 다음과 같은 값이 있는 단일 데이터셋 필드로 매핑됩니다.

```
Mr. John Smith
```

XMLMapper 사용

XML 매퍼 유틸리티인 xmlmapper를 사용하여 다음과 같은 세 가지 방법으로 매핑을 정의할 수 있습니다.

- 기존 XML 스키마 또는 문서에서 사용자가 정의하는 클라이언트 데이터셋으로 매핑합니다. 이것은 이미 XML 스키마가 있는 데이터를 사용하는 데이터베이스 애플리케이션을 만들려는 경우에 유용합니다.
- 기존 데이터 패킷에서 사용자가 정의하는 새로운 XML 스키마로 매핑합니다. 예를 들어 새로운 B2B 통신 시스템을 만드는 경우처럼 기존 데이터베이스 정보를 XML로 만드는 경우에 유용합니다.
- 기존 XML 스키마와 기존 데이터 패킷 간에 매핑합니다. 이것은 정보가 동일한 XML 스키마와 데이터베이스가 있고 두 가지를 함께 사용하는 경우에 유용합니다.

일단 매핑을 정의하면 XML 문서를 데이터 패킷으로 변환하고 데이터 패킷을 XML 문서로 변환하는 데 사용되는 변환 파일을 생성할 수 있습니다. 변환 파일에서만 매핑 방향을 지정할 수 있습니다. XML에서 데이터 패킷으로 변환하는 변환만 만들 수도 있고 데이터 패킷에서 XML로 변환하는 변환만 만들 수도 있습니다.

참고 XML 매퍼가 제대로 작동하려면 midas.so 공유 라이브러리가 있어야 합니다. xmlmapper를 사용하기 전에 midas.so 공유 라이브러리가 설치되어 있는지 확인하십시오.

XML 스키마 또는 데이터 패킷 로딩

매핑을 정의하고 변환 파일을 생성할 수 있으려면 먼저 매핑하려는 XML 문서와 데이터 패킷에 대한 설명을 로드해야 합니다.

File|Open을 선택하면 나타나는 대화 상자에서 문서나 스키마를 선택하여 XML 문서나 스키마를 로드할 수 있습니다.

File|Open을 선택하면 나타나는 대화 상자에서 데이터 패킷 파일을 선택하면 데이터 패킷을 로드할 수 있습니다. (이 데이터 패킷은 클라이언트 데이터셋의 *SaveToFile* 메소드를 호출하면 생성되는 파일입니다.)

XML 문서나 스키마만 로드하거나 또는 데이터 패킷만 로드하거나 두 가지를 모두 로드할 수 있습니다. 매핑의 한쪽만 로드하면 XML 매퍼는 다른 쪽의 자연 매핑을 생성할 수 있습니다.

매핑 정의

XML 문서와 데이터 패킷 사이의 매핑은 데이터 패킷의 모든 필드나 XML 문서의 태그 요소를 모두 포함하지 않아도 됩니다. 따라서 먼저 매핑할 요소를 지정해야 합니다. 매핑할 요소를 지정하려면 먼저 대화 상자의 가운데 창에서 매핑 페이지를 선택합니다.

데이터 패킷의 필드로 매핑할 XML 문서나 스키마의 요소를 지정하려면 XML 문서 창의 Sample 또는 Structure 탭을 선택하고 데이터 패킷 필드로 매핑할 요소의 노드를 더블 클릭합니다.

XML 문서의 속성 또는 태그 요소로 매핑할 데이터 패킷의 필드를 지정하려면 Datapacket 창에서 해당 필드의 노드를 더블 클릭합니다.

매핑 대상(XML 문서 또는 데이터 패킷)을 하나만 로드한 경우 매핑할 노드를 선택한 후 나머지 하나를 생성할 수 있습니다.

- XML 문서에서 데이터 패킷을 생성하는 경우 먼저 데이터 패킷에서 해당하는 필드의 타입을 결정하기 위해 선택한 노드의 속성을 정의합니다. 가운데 창에서 Node Repository 페이지를 선택합니다. 매핑할 각 노드를 선택하고 해당 필드의 속성을 지정합니다. 매핑이 직접적이지 않은 경우(예를 들어 필드에 매핑되는 노드가 있고 이 노드의 자식 노드의 값을 조합하여 필드 값이 구성되는 경우)에는 User Defined Translation 체크 박스를 선택합니다. 나중에 사용자 정의 노드에서 변환을 수행하려면 이벤트 핸들러를 작성해야 합니다.

  일단 노드 매핑 방식을 지정했으면 Create|Datapacket from XML을 선택합니다. 해당 데이터 패킷이 자동으로 생성되어 Datapacket 창에 표시됩니다.

- 데이터 패킷에서 XML 문서를 생성하는 경우 Create|XML from Datapacket을 선택합니다. 데이터 패킷의 필드, 레코드, 데이터셋에 해당하는 XML 문서의 속성 및 태그 이름을 지정할 수 있는 대화 상자가 나타납니다. 필드 값을 값이 있는 태그 요소로 매핑할지, 이름을 지정하는 방식으로 속성으로 매핑할지를 지정합니다. @ 기호로 시작하는 이름은 레코드에 해당하는 태그의 속성으로 매핑되지만 @ 기호로 시작하지 않는 이름은 값이 있고 레코드의 요소 내에서 중첩되는 태그 요소로 매핑됩니다.

- XML 문서와 데이터 패킷(클라이언트 데이터셋 파일)을 모두 로드한 경우에는 해당 노드를 같은 순서로 선택했는지 확인해야 합니다. 해당 노드들은 Mapping 페이지의 맨 위에 있는 테이블에서 서로의 옆에 나타나야 합니다.

XML 문서와 데이터 패킷을 모두 로드하거나 생성한 다음 매핑에 나타나는 노드를 선택하고 나면 Mapping 페이지의 맨 위에 있는 테이블에 정의한 매핑이 나타나야 합니다.

변환 파일 생성

변환 파일을 만들려면 다음 단계를 따릅니다.

1 먼저 만들려는 변환에 따라 적절한 라디오 버튼을 선택합니다.

   - 데이터 패킷에서 XML 문서로 매핑을 진행하려는 경우에는 Datapacket to XML 버튼을 선택합니다.
   - XML 문서에서 데이터 패킷으로 매핑을 진행하려는 경우에는 XML to Datapacket 버튼을 선택합니다.

2 데이터 패킷을 생성하려는 경우 Create Datapacket As의 라디오 버튼도 사용해야 합니다. 이 버튼을 사용하면 데이터 패킷이 사용되는 방법을 데이터셋으로, 업데이트를 적용하는 델타 패킷으로, 데이터를 fetch하기 전에 프로바이더에 제공하는 매개변수로 지정할 수 있습니다.

3 변환의 메모리 내(in-memory) 버전을 만들려면 Create and Test Transformation을 클릭합니다. XML 매퍼는 데이터 패킷에 대해 생성되는 XML 문서를 Datapacket 창에 표시하거나 XML 문서에 대해 생성되는 데이터 패킷을 XML Document 창에 표시합니다.

4 마지막으로 File|Save|Transformation을 선택하여 변환 파일을 저장합니다. 변환 파일은 사용자가 정의한 변환을 표현하는 파일이며 확장자가 .xtr인 특수한 XML 파일입니다.

XML 문서를 데이터 패킷으로 변환

XML 문서를 데이터 패킷으로 변환하는 방법을 나타내는 변환 파일을 작성하면 변환에 사용된 스키마에 호환되는 XML 문서에 대한 데이터 패킷을 만들 수 있습니다. 그런 다음 이러한 데이터 패킷을 클라이언트 데이터셋에 할당하고 파일에 저장하여 파일 기반 데이터베이스 애플리케이션의 기초를 형성합니다.

TXMLTransform 컴포넌트는 변환 파일의 매핑에 따라 XML 문서를 데이터 패킷으로 변환합니다.

참고 *TXMLTransform*을 사용하면 XML 형식의 데이터 패킷을 임의의 XML 문서로 변환할 수 있습니다.

소스 XML 문서 지정

소스 XML 문서를 지정하는 방법에는 다음 세 가지가 있습니다.

- 소스 문서가 디스크에 있는 .xml 파일이면 *SourceXmlFile* 속성을 사용할 수 있습니다.
- 소스 문서가 XML의 메모리 내(in-memory) 문자열이면 *SourceXml* 속성을 사용할 수 있습니다.
- 소스 문서에 대한 IDOMDocument 인터페이스가 있으면 *SourceXmlDocument* 속성을 사용할 수 있습니다.

*TXMLTransform*은 위에 나열된 순서로 이 속성들을 검사합니다. 즉 *SourceXmlFile* 속성에서 파일 이름을 먼저 검사합니다. *SourceXmlFile*이 빈 문자열인 경우에만 *SourceXml* 속성을 검사합니다. *SourceXml*이 빈 문자열인 경우에만 *SourceXmlDocument* 속성을 검사합니다.

변환 지정

XML 문서를 데이터 패킷으로 바꾸는 변환을 지정하는 방법에는 다음과 같은 두 가지 방법이 있습니다.

- xmlmapper.exe를 사용하여 만든 변환 파일을 나타내도록 *TransformationFile* 속성을 설정합니다.
- 변환에 대한 *IDOMDocument* 인터페이스가 있는 경우에는 *TransformationDocument* 속성을 설정합니다.

*TXMLTransform*은 위에 나열된 순서대로 이 속성들을 검사합니다. 즉 *TransformationFile* 속성에서 파일 이름을 먼저 검사합니다. *TransformationFile*이 빈 문자열인 경우에만 *TransformationDocument* 속성을 검사합니다.

결과 데이터 패킷 가져오기

*TXMLTranform*이 변환을 수행하고 데이터 패킷을 생성하도록 하려면 *Data* 속성을 읽기만 하면 됩니다. 예를 들어 다음 코드는 XML 문서와 변환 파일을 사용하여 데이터 패킷을 생성한 다음 클라이언트 데이터셋에 할당합니다.

```
XMLTransform1.SourceXMLFile := 'CustomerDocument.xml';
XMLTransform1.TransformationFile := 'CustXMLToCustTable.xtr';
ClientDataSet1.XMLData := XMLTransform1.Data;
```

사용자 정의 노드 변환

xmlmapper를 사용하여 변환을 정의할 때 XML 문서의 일부 노드를 "사용자 정의"할 수 있습니다. 사용자 정의 노드는 직접적인 '노드 값에서 필드 값으로의 변환'에 의존하지 않고 코드에서 변환을 제공하는 노드입니다.

OnTranslate 이벤트를 사용하는 사용자 정의 노드를 번역하도록 코드를 제공할 수 있습니다. *OnTranslate*는 *TXMLTransform* 컴포넌트가 XML 문서에서 사용자 정의 노드를 만날 때마다 호출됩니다. *OnTranslate* 이벤트 핸들러에서는 소스 문서를 읽고 데이터 패킷의 필드에 대한 결과 값을 지정할 수 있습니다.

예를 들어 *OnTranslate* 이벤트 핸들러는 다음과 같은 형식의 XML 문서의 노드를

```
<FullName>
   <Title> </Title>
   <FirstName> </FirstName>
   <LastName> </LastName>
</FullName>
```

다음과 같은 단일 필드 값으로 변환합니다.

```
procedure TForm1.XMLTransform1Translate(Sender: TObject; Id: String; SrcNode: IDOMNode;
  var Value: String; DestNode: IDOMNode;
var
  CurNode: IDOMNode;
begin
  if Id = 'FullName' then
  begin
    Value = '';
    if SrcNode.hasChildNodes then
    begin
      CurNode := SrcNode.firstChild;
      Value := Value + CurNode.nodeValue;
      while CurNode <> SrcNode.lastChild do
      begin
        CurNode := CurNode.nextSibling;
        Value := Value + ' ';
        Value := Value + CurNode.nodeValue;
      end;
```

```
    end;
  end;
end;
```

프로바이더의 소스로 XML 문서 사용

TXMLTransformProvider 컴포넌트를 사용하면 XML 문서를 데이터베이스 테이블처럼 사용할 수 있습니다. *TXMLTransformProvider*는 XML 문서의 데이터를 패키지로 만들고 클라이언트의 업데이트를 해당 XML 문서에 다시 적용합니다.
TXMLTransformProvider는 다른 프로바이더 컴포넌트처럼 클라이언트 데이터셋이나 XML 브로커 같은 클라이언트에 나타납니다. 프로바이더 컴포넌트에 대한 내용은 *Kylix1 개발자 안내서*의 21장 "프로바이더 컴포넌트 사용"을 참조하십시오. 프로바이더 컴포넌트를 클라이언트 데이터셋과 함께 사용하는 내용은 *Kylix1 개발자 안내서* 20-25페이지의 "프로바이더와 함께 클라이언트 데이터셋 사용"을 참조하십시오.

XML 프로바이더가 데이터를 제공하고 *XMLDataFile* 속성을 사용하여 업데이트를 적용하는 XML 문서를 지정할 수 있습니다.

TXMLTransformProvider 컴포넌트는 내부 *TXMLTransform* 컴포넌트를 사용하여 데이터 패킷과 소스 XML 문서 간을 번역합니다. 즉 XML 문서를 데이터 패킷으로 번역하는 경우와 업데이트를 적용한 후 데이터 패킷을 다시 XML 형식의 소스 문서로 번역하는 경우가 있습니다. 이 두 가지 *TXMLTransform* 컴포넌트는 각각 *TransformRead* 속성과 *TransformWrite* 속성을 사용하여 액세스할 수 있습니다.

*TXMLTransformProvider*를 사용할 때는 이 두 가지 *TXMLTransform* 컴포넌트가 데이터 패킷과 소스 XML 문서 간에 번역할 때 사용하는 변환을 지정해야 합니다. 독립형 *TXMLTransform* 컴포넌트를 사용할 때와 같이 *TXMLTransform* 컴포넌트의 *TransformationFile* 속성이나 *TransformationDocument* 속성을 설정하면 됩니다.

또한 변환에 사용자 정의 노드가 포함되면 *OnTranslate* 이벤트 핸들러를 내부 *TXMLTransform* 컴포넌트에 제공해야 합니다.

TransformRead 및 *TransformWrite*의 값인 *TXMLTransform* 컴포넌트에서 소스 문서를 지정할 필요는 없습니다. *TransformRead*의 경우 소스는 프로바이더의 *XMLDataFile* 속성에서 지정한 파일이지만 *XMLDataFile*을 빈 문자열로 설정한 경우에는 *TransformRead*, *XmlSource* 또는 *TransformRead*, *XmlSourceDocument*를 사용하여 소스 문서를 제공할 수 있습니다. *TransformWrite*의 경우 소스는 업데이트를 적용할 때 프로바이더에서 내부적으로 생성됩니다.

프로바이더의 클라이언트로 XML 문서 사용

TXMLTransformClient 컴포넌트는 어댑터로 작동하여 사용자가 XML 문서나 문서 집합을 애플리케이션 서버의 클라이언트로서 또는 단순히 *TDataSetProvider* 컴포넌트를 통해 연결되는 데이터셋의 클라이언트로서 사용할 수 있게 합니다. 즉 *TXMLTransform* 클라이언트를 사용하면 데이터베이스 데이터를 XML 문서로 게시할 수 있으며 XML 문서의 형태로 제공하는 외부 애플리케이션에서의 업데이트 요청(삽입 또는 삭제)을 사용할 수 있습니다.

TXMLTransformClient 객체가 데이터를 fetch하고 업데이트를 적용하는 프로바이더를 지정하려면 *ProviderName* 속성을 설정합니다. 클라이언트 데이터셋의 *ProviderName* 속성을 사용하는 경우처럼 *ProviderName*은 원격 애플리케이션 서버의 프로바이더 이름이 되거나 *TXMLTransformClient* 객체와 같이 같은 형태나 데이터 모듈에서 로컬 프로바이더가 될 수 있습니다. 프로바이더에 대한 자세한 내용은 *Kylix1 개발자 안내서*의 21장 "프로바이더 컴포넌트 사용"을 참조하십시오.

프로바이더가 원격 애플리케이션 서버에 있는 경우 컴포넌트 팔레트의 WebServices 페이지에 있는 *TSoapConnection* 컴포넌트를 사용해야 합니다. *RemoteServer* 속성을 사용하여 연결 컴포넌트를 지정합니다. *TSoapConnection* 컴포넌트에 대한 내용은 4-12페이지의 "애플리케이션 서버에 연결"을 참조하십시오.

프로바이더에서 XML 문서 fetch

*TXMLTransformClient*는 내부 *TXMLTransform* 컴포넌트를 사용하여 프로바이더에서 XML 문서로 데이터 패킷을 번역합니다. *TransformGetData* 속성의 값으로 TXMLTransform 컴포넌트에 액세스할 수 있습니다.

프로바이더에서 데이터를 나타내는 XML 문서를 생성하려면 *TransformGetData*를 사용하여 데이터 패킷을 적절한 XML 형식으로 번역하는 변환 파일을 지정해야 합니다. 독립형 *TXMLTransform* 컴포넌트를 사용할 때와 같이 *TXMLTransform* 컴포넌트의 *TransformationFile* 속성이나 *TransformationDocument* 속성을 설정함으로써 변환 파일을 지정할 수 있습니다. 사용자 정의 노드가 변환에 포함되면 *OnTranslate* 이벤트 핸들러가 있는 *TransformGetData*를 제공할 수 있습니다.

*TransformGetData*의 소스 문서를 지정할 필요가 없으며 *TXMLTransformClient*가 프로바이더에서 소스 문서를 fetch합니다. 그러나 프로바이더가 입력 매개변수를 예상하면 데이터를 fetch하기 전에 입력 매개변수를 설정할 수 있습니다. *SetParams* 메소드를 사용하여 프로바이더에서 데이터를 fetch하기 전에 이러한 입력 매개변수를 제공합니다. *SetParams*는 2개의 인수가 있으며 이 두 인수는 매개변수 값을 추출할 XML 문자열과 이 XML을 데이터 패킷으로 번역하는 변환 파일의 이름입니다. *SetParams*는 변환 파일을 사용하여 XML 문자열을 데이터 패킷으로 변환한 다음 데이터 패킷에서 매개변수 값을 추출합니다.

참고 매개변수 문서나 변환을 다른 방법으로 지정하려면 2개의 인수 중 하나를 오버라이드할 수 있습니다. *TransformSetParams* 속성의 여러 속성 중 하나를 설정하여 매개변수 또는 매개변수를 변환할 때 사용하는 변환이 포함된 문서를 지정한 다음 *SetParams*를 호출할 때 빈 문자열로 오버라이드할 인수를 설정합니다. 사용할 수 있는 속성에 대한 자세한 내용은 5-6페이지의 "XML 문서를 데이터 패킷으로 변환"을 참조하십시오.

*TransformGetData*를 구성하고 입력 매개변수를 제공하면 *GetDataAsXml* 메소드를 호출하여 XML을 fetch할 수 있습니다. *GetDataAsXml*은 현재의 매개변수 값을 프로바이더로 보내고, 데이터 패킷을 fetch하여 XML 문서로 변환하고, 해당 문서를 문자열로 반환합니다. 이 문자열을 다음과 같이 파일에 저장할 수 있습니다.

```
var
  XMLDoc: TFileStream;
  XML string;
begin
  XMLTransformClient1.ProviderName := 'Provider1';
  XMLTransformClient1.TransformGetData.TransformationFile := 'CustTableToCustXML.xtr';
  XMLTransformClient1.TransFormSetParams.SourceXmlFile := 'InputParams.xml';
  XMLTransformClient1.SetParams('', 'InputParamsToDP.xtr');
  XML := XMLTransformClient1.GetDataAsXml;
  XMLDoc := TFileStream.Create('Customers.xml', fmCreate or fmOpenWrite);
  try
    XMLDoc.Write(XML, Length(XML));
  finally
    XMLDoc.Free;
  end;
end;
```

XML 문서에서 프로바이더로 업데이트 적용

*TXMLTransformClient*를 사용하여 XML 문서의 모든 데이터를 프로바이더의 데이터셋으로 삽입하거나 XML 문서에 있는 모든 레코드를 프로바이더의 데이터셋에서 삭제할 수 있습니다. 이러한 업데이트를 수행하려면 *ApplyUpdates* 메소드를 호출하여 다음으로 전달합니다.

- 삽입하거나 삭제할 데이터가 있는 XML 문서의 컨텐트인 문자열.
- XML 데이터를 삽입 또는 삭제 델타 패킷으로 변환할 수 있는 변환 파일의 이름. (XML 매퍼 유틸리티를 사용하여 변환 파일을 지정할 때는 이 변환이 삽입 델타 패킷에 대한 것인지, 삭제 델타 패킷에 대한 것이지 지정합니다.)
- 업데이트 작업이 중지되기 전에 허용할 수 있는 업데이트 오류의 수. 지정한 수보다 적은 레코드를 삽입 또는 삭제할 수 없으면 *ApplyUpdates*는 실제 오류 수를 반환합니다. 지정한 수보다 많은 레코드를 삽입할 수 없거나 삭제할 수 없는 경우 전체 업데이트 작업이 롤백되고 업데이트가 수행되지 않습니다.

다음과 같은 구문은 XML 문서 Customers.xml을 델타 패킷으로 변환하고 오류 수에 관계없이 모든 업데이트를 적용합니다.

```
StringList1.LoadFromFile('Customers.xml');
nErrors := ApplyUpdates(StringList1.Text, 'CustXMLToInsert.xtr', -1);
```

III

인터넷 애플리케이션 개발

"인터넷 애플리케이션 개발"의 각 장은 LAN이나 인터넷 상에서 분산된 애플리케이션을 구축하는 데 필요한 개념과 기술을 설명합니다. Kylix 일부 에디션에서는 여기서 다룬 컴포넌트를 사용하지 못할 수도 있습니다.

6

인터넷 서버 애플리케이션 생성

* 이 장은 영문 Kylix2 개발자 안내서의 24장입니다.

웹 서버 애플리케이션은 기존 웹 서버의 기능을 확장합니다. 웹 서버 애플리케이션은 웹 서버로부터 HTTP 요청 메시지를 수신하여 이 메시지에서 요청된 모든 작업을 수행하고 웹 서버로 다시 보낼 응답을 작성합니다. Kylix 애플리케이션을 사용하여 수행할 수 있는 모든 작업을 웹 서버 애플리케이션에 포함시킬 수 있습니다.

Kylix는 웹 서버 애플리케이션 개발을 위해 두 개의 다른 아키텍처인 Web Broker와 WebSnap을 제공합니다. WebSnap과 Web broker가 다르기는 하지만 공통 요소가 많습니다. WebSnap 아키텍처는 Web Broker의 수퍼셋으로 작용합니다. WebSnap 아키텍처는 개발자가 애플리케이션을 실행하지 않고도 페이지의 컨텐트를 표시할 수 있게 하는 WebSnap Surface Designer와 같은 새로운 기능 및 추가적인 컴포넌트를 제공합니다. WebSnap을 사용하여 개발한 애플리케이션은 Web Broker 컴포넌트를 포함할 수 있는 반면 Web Broker를 사용하여 개발한 애플리케이션은 WebSnap 컴포넌트를 포함할 수 없습니다.

이 장에서는 Web Broker와 WebSnap 기술의 특징에 대해 설명하고 인터넷 기반 클라이언트/서버 애플리케이션에 대한 일반적인 정보를 제공합니다.

Web Broker 및 WebSnap 정보

애플리케이션의 기능 중 하나는 사용자가 데이터에 액세스할 수 있도록 하는 것입니다. 표준 Kylix에서 사용자는 대화 상자 및 스크롤 윈도우와 같은 종래의 프런트 엔드 요소를 생성하여 데이터에 액세스할 수 있습니다. 개발자는 친숙한 Kylix 폼 디자인 툴을 사용하여 이러한 객체의 레이아웃을 지정할 수 있습니다. 하지만 웹 서버 애플리케이션은 다르게 디자인해야 합니다. 사용자에게 전달되는 모든 정보가 HTTP를 통해 전송되는 HTML 페이지의 형태여야 합니다. 일반적으로 페이지는 사용자의 특정 시스템에 맞게 페이지를 표시하는 웹 브라우저 애플리케이션에 의해 클라이언트 컴퓨터에서 해석됩니다.

웹 서버 애플리케이션을 구축하는 첫 번째 단계는 Web Broker와 WebSnap 중에서 사용할 아키텍처를 선택하는 것입니다. Web Broker와 WebSnap은 다음과 같은 동일한 기능을 갖고 있습니다.

- CGI와 Apache 웹 서버 애플리케이션 지원. 이 내용은 6-6페이지의 "웹 서버 애플리케이션의 유형"에서 설명합니다.
- 수신된 클라이언트 요청을 별도의 스레드에서 처리하는 다중 스레드 지원
- 신속한 응답을 위해 웹 모듈을 캐시에 저장
- 크로스 플랫폼 개발. 웹 서버 애플리케이션을 Windows와 Linux 운영 체제 간에 쉽게 포팅할 수 있습니다. 두 가지 플랫폼 중 하나에서 소스 코드를 컴파일합니다.

모든 Web Broker와 WebSnap 컴포넌트는 페이지 전송 메커니즘을 처리합니다. WebSnap은 Web Broker를 기초로 사용하여 Web Broker 아키텍처의 모든 기능을 통합합니다. WebSnap은 페이지 생성을 위한 더 강력한 툴을 제공합니다. 또한 WebSnap 애플리케이션을 통해 런타임에 페이지를 생성할 수 있도록 서버측 스크립트를 사용할 수 있습니다. Web Broker에는 이러한 스크립팅 기능이 없습니다. Web Broker에서 제공하는 툴은 WebSnap에 비해 완벽하지 않고 그다지 직관적이지도 않습니다. 웹 서버 애플리케이션을 새로 개발하려는 경우 대개는 Web Broker보다 WebSnap 아키텍처를 선택하는 것이 좋습니다.

이 두 가지 방식의 주요 차이점은 다음 표에서 개괄적으로 설명합니다.

표 6.1 Web Broker와 WebSnap 비교

Web Broker	WebSnap
이전 버전과 호환됩니다.	WebSnap 애플리케이션은 컨텐트를 만드는 모든 Web Broker 컴포넌트를 사용할 수 있지만 이를 포함하는 웹 모듈과 디스패처는 새로운 개념입니다.
애플리케이션에서 하나의 웹 모듈만 사용할 수 있습니다.	다중 웹 모듈은 애플리케이션을 유닛으로 분할할 수 있으며 여러 개발자가 충돌 없이 프로젝트를 함께 개발할 수 있습니다.
애플리케이션에서 하나의 웹 디스패처만 사용할 수 있습니다.	용도에 따라 여러 디스패처가 다양한 종류의 요청을 처리합니다.
컨텐트를 만드는 특수한 컴포넌트에는 페이지 프로듀서, InternetExpress 컴포넌트 및 Web Services 컴포넌트가 있습니다.	Web Broker 애플리케이션에 있는 모든 컨텐트 프로듀서뿐 아니라 복잡한 데이터 방식 웹 페이지를 신속하게 구축할 수 있도록 디자인된 기타 많은 컴포넌트를 지원합니다.
스크립트를 지원하지 않습니다.	JavaScript 서버측 스크립트를 지원하여 HTML 생성 로직이 비즈니스 로직으로부터 분리됩니다.
명명된(named) 페이지를 기본 제공하지 않습니다.	명명된 페이지는 페이지 디스패처가 자동으로 검색하므로 서버측 스크립트에서 사용할 수 있습니다.
세션을 지원하지 않습니다.	단기간 동안 최종 사용자에 대해 필요한 정보를 세션에서 저장합니다. 이는 로그인/로그아웃 지원과 같은 작업에 사용될 수 있습니다.

표 6.1 Web Broker와 WebSnap 비교(계속)

Web Broker	WebSnap
모든 요청은 액션 항목이나 자동 디스패칭 컴포넌트 중 하나를 사용하여 명시적으로 처리되어야 합니다.	디스패치 컴포넌트는 다양한 요청에 자동으로 응답합니다.
일부 특수한 컴포넌트에서만 컨텐트의 미리 보기를 제공합니다. 대부분의 개발 작업에서는 미리 볼 수 없습니다.	WebSnap surface designer는 사용자가 웹 페이지를 비주얼하게 구축하고 디자인 타임에 결과를 볼 수 있도록 합니다. 모든 컴포넌트에서 미리 보기를 사용할 수 있습니다.

Web Broker에 대한 자세한 내용은 7장 "Web Broker 사용"을 참조하십시오. WebSnap에 대한 자세한 내용은 8장 "WebSnap 사용"을 참조하십시오.

용어 및 표준

인터넷 상의 활동을 제어하는 대부분의 프로토콜은 RFC(Request for Comment) 문서에 정의되며 이 문서는 인터넷 프로토콜 엔지니어링 및 개발 조직인 IETF(Internet Engineering Task Force)에 의해 작성, 업데이트 및 유지 관리됩니다. 인터넷 애플리케이션을 작성할 때 다음과 같은 몇 가지 중요한 RFC가 유용하게 사용됩니다.

- RFC822. "ARPA 인터넷 텍스트 메시지의 포맷에 관한 표준"에서 메시지 헤더의 구조 및 내용을 설명합니다.
- RFC1521. "MIME(Multipurpose Internet Mail Extensions) 1부: 인터넷 메시지 본문(body)의 포맷을 지정하고 설명하기 위한 메커니즘"에서 multipart와 multiformat 메시지를 캡슐화하고 전송하는 데 사용되는 방법을 설명합니다.
- RFC1945. "Hypertext Transfer Protocol – HTTP/1.0"에서 협력적 하이퍼미디어 문서를 배포하는 데 사용되는 전송 메커니즘을 설명합니다.

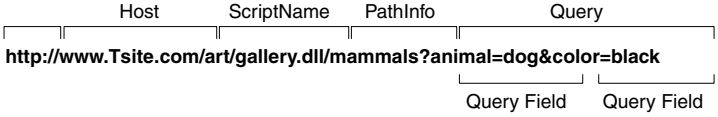
IETF는 다음 웹 사이트에서 RFC의 라이브러리를 유지 관리합니다.

```
www.ietf.cnri.reston.va.us
```

URL(Uniform Resource Locator)의 각 부분

URL(Uniform Resource Locator)은 네트워크에서 사용할 수 있는 리소스 위치에 대한 완전한 설명으로 애플리케이션에서 액세스할 수 있는 여러 부분들로 구성되어 있습니다. 그림 6.1은 URL을 구성하는 부분들을 보여 줍니다.

그림 6.1 URL(Uniform Resource Locator)의 각 부분



첫 번째 부분은 기술적으로 URL의 부분은 아니지만 프로토콜(http)을 식별합니다. 이 부분에 https(보안 http), ftp 등 다른 프로토콜을 지정할 수 있습니다.

Host 부분은 웹 서버와 웹 서버 애플리케이션을 실행하는 시스템을 식별합니다. 위의 그림에는 나와 있지 않지만 이 부분은 메시지를 받는 포트를 오버라이드할 수 있습니다. 일반적으로 포트 번호가 프로토콜에 함축되어 있으므로 포트를 지정할 필요가 없습니다.

ScriptName 부분은 웹 서버 애플리케이션의 이름을 지정합니다. 웹 서버는 이 애플리케이션에 메시지를 전달합니다.

스크립트 이름 다음에는 PathInfo가 옵니다. 이 부분은 웹 서버 애플리케이션 내의 메시지의 대상을 식별합니다. PathInfo 값은 호스트 시스템의 디렉토리, 특정 메시지에 응답하는 컴포넌트의 이름 또는 웹 서버 애플리케이션이 수신 메시지의 처리를 분할하는 데 사용하는 기타 메커니즘을 참조할 수 있습니다.

Query 부분에는 명명된(named) 값 집합이 포함됩니다. 이러한 값과 이름은 웹 서버 애플리케이션에 의해 정의됩니다.

URI와 URL 비교

URL은 HTTP 표준인 RFC1945에 정의된 URI(Uniform Resource Identifier)의 하위 집합입니다. 웹 서버 애플리케이션은 대개 최종 결과가 특정 위치에 있지 않고 필요에 따라 만들어지는 다양한 소스로부터 컨텐트를 만듭니다. URI는 특정 위치와 관련되지 않은 리소스를 정의할 수 있습니다.

HTTP 요청 헤더 정보

HTTP 요청 메시지에는 클라이언트, 요청 대상, 요청 처리 방법, 요청과 함께 전달된 컨텐트 등에 대한 정보를 설명하는 여러 개의 헤더가 들어 있습니다. 각 헤더는 문자열 값이 따라오는 "Host"와 같은 이름으로 식별합니다. 예를 들어 다음과 같은 HTTP 요청이 있다고 가정합니다.

```
GET /art/gallery/animals?animal=dog&color=black HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0b4Gold (WinNT; I)
Host: www.TSite.com:1024
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

첫 번째 줄은 GET으로 요청을 식별합니다. GET 요청 메시지는 GET 다음의 URI(/art/gallery.cgi/animals?animal=doc&color=black)와 연결된 컨텐트를 반환하도록 웹 서버 애플리케이션에 요청합니다. 첫 번째 줄의 마지막 부분은 클라이언트가 HTTP 1.0 표준을 사용하고 있음을 나타냅니다.

두 번째 줄은 Connection 헤더이며 요청이 서비스되었더라도 연결이 계속 유지되어야 한다는 것을 나타냅니다. 세 번째 줄은 User-Agent 헤더이며 요청을 생성한 프로그램에 대한 정보를 제공합니다. 네 번째 줄은 Host 헤더이며 Host 이름과 연결을 설정하기 위해 서버의 포트를 제공합니다. 마지막 줄은 Accept 헤더로 클라이언트가 유효한 응답으로 승인하는 매체 유형을 나열합니다.

HTTP 서버 활동

웹 브라우저의 클라이언트/서버 특성은 언뜻 보기에는 간단합니다. 링크만 클릭하면 정보가 화면에 표시되기 때문에 웹에서의 정보 검색은 대부분의 사용자에게 매우 간단한 일입니다. 좀더 경험 있는 사용자들은 HTML 구문의 특성과 프로토콜의 클라이언트/서버 특성을 어느 정도는 이해하고 있으며 이 정도면 일반적으로 간단한 페이지 방식 웹 사이트 컨텐트를 충분히 만들 수 있습니다. 그러나 더 복잡한 웹 페이지를 만드는 사람이라면 다양한 옵션을 사용하여 HTML을 통해 정보 수집 및 표시를 자동화할 수 있습니다.

웹 서버 애플리케이션을 구축하기 전에 클라이언트가 요청을 하는 방법과 서버가 클라이언트 요청에 응답하는 방법을 이해하는 것이 좋습니다.

클라이언트 요청 구성

HTML 하이퍼텍스트 링크가 선택되거나 사용자가 URL을 지정하면 브라우저는 프로토콜, 지정된 도메인, 정보 경로, 날짜 및 시간, 운영 환경, 브라우저 등에 대한 정보와 기타 컨텐트 정보를 수집한 다음 요청을 구성합니다.

예를 들어 폼에서 버튼을 클릭하여 선택된 기준에 따라 이미지 페이지를 표시하려면 클라이언트는 다음과 같은 URL을 만듭니다.

```
http://www.TSite.com/art/gallery/animals?animal=dog&color=black
```

이 URL은 www.TSite.com 도메인에 있는 HTTP 서버를 지정합니다. 클라이언트는 www.TSite.com에 접속하고 HTTP 서버에 연결하여 요청을 전달합니다. 이 요청은 다음과 같습니다.

```
GET /art/gallery/animals?animal=dog&color=black HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0b4Gold (WinNT; I)
Host: www.TSite.com:1024
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

CGI 요청 서비스

웹 서버는 클라이언트 요청을 수신하여 서버 구성에 따라 여러 작업을 수행할 수 있습니다. 요청의 /gallery.cgi 부분을 프로그램으로 인식하도록 구성된 경우 웹 서버는 요청에 포함된 정보를 직접 CGI 프로그램에 전달합니다. 웹 서버는 CGI 프로그램이 실행되는 동안 대기합니다. CGI 프로그램이 종료하면 stdout 파이프를 통해 HTML 컨텐트가 웹 서버에 다시 전달됩니다.

CGI 프로그램은 프로그래머의 요청에 따라 데이터베이스 액세스, 간단한 테이블 조회나 계산, HTML 문서 작성이나 선택 등 프로그래머가 지정한 작업을 수행합니다.

동적 공유 객체 요청 서비스

웹 서버 애플리케이션은 클라이언트 요청을 모두 처리했을 때 HTML 코드 페이지 또는 다른 MIME 컨텐트를 생성하고 클라이언트가 이를 표시할 수 있도록 서버를 통해 클라이언트로 다시 전달합니다. 동적 공유 객체(DSO)가 처리되면 HTML 페이지와 응답 정보는 직접 서버로 전달되고 서버에서 다시 클라이언트로 전달됩니다.
웹 서버 애플리케이션을 공유 객체로 만들면 개별 요청을 서비스하는 데 필요한 프로세스 및 디스크 액세스 수가 감소하여 시스템 로드 및 리소스 사용이 줄어듭니다. DSO에 대한 자세한 내용은 다음 사이트를 참조하십시오.

http://httpd.apache.org/docs/dso.html

웹 서버 애플리케이션의 유형

Web Broker와 WebSnap 중 어떤 것을 사용해도 CGI와 Apache 웹 서버 애플리케이션을 모두 만들 수 있습니다. 이 두 가지 유형은 *TWebApplication*, *TWebRequest* 및 *TWebResponse*와 같은 유형별 자손을 사용합니다.

표 6.2 웹 서버 애플리케이션 컴포넌트

애플리케이션 유형	애플리케이션 객체	요청 객체	응답 객체
콘솔 CGI 애플리케이션	*TCGIApplication*	*TCGIRequest*	*TCGIResponse*
Apache DSO 모듈	*TApacheApplication*	*TApacheRequest*	*TApacheResponse*

CGI 독립형

CGI 독립형 웹 서버 애플리케이션은 표준 입력으로 클라이언트 요청 정보를 수신하여 표준 출력으로 결과를 서버에 전달하는 콘솔 애플리케이션입니다. 이 데이터는 *TCGIRequest* 및 *TCGIResponse* 객체를 만드는 *TCGIApplication*에 의해 처리됩니다. 각각의 요청 메시지는 개별 애플리케이션 인스턴스에 의해 처리됩니다.

Apache DSO 모듈

참고 DSO를 지원하려면 소스 코드를 다운로드하여 Apache를 재설치해야 합니다.

Apache DSO 모듈은 라이브러리 애플리케이션입니다. Apache 프로그램은 공유 객체를 메모리에 로드한 다음 Apache 프로그램과 모듈 간에 데이터를 전달하는 데 사용되는 메모리 블록을 예약합니다. 이 데이터는 *TApacheRequest* 및 *TApacheResponse* 객체를 만드는 *TApacheApplication*에 의해 처리됩니다.

7

Web Broker 사용

* 이 장은 영문 Kylix2 개발자 안내서의 25장입니다.

Web Broker 아키텍처는 Kylix의 모든 웹 서버 개발에 있어서 기초가 됩니다. 컴포넌트 팔레트의 Internet 탭에 있는 Web Broker 컴포넌트를 통해 특정 Uniform Resource Identifier(URI)와 연결된 이벤트 핸들러를 작성할 수 있습니다. 이 이벤트 핸들러는 웹 서버가 해당 URI의 요청을 수신하면 활성화됩니다. 처리가 완료되면 웹 브로커를 사용하여 프로그램에서 HTML이나 XML 문서를 만들어 클라이언트에게 전송할 수 있습니다.

웹 페이지의 컨텐트는 대개 데이터베이스에서 가져옵니다. 인터넷 컴포넌트를 사용하면 하나의 공유 객체가 스레드에 대해 안전하게 동시에 여러 개의 데이터베이스를 관리할 수 있도록 데이터베이스와의 연결을 자동으로 관리할 수 있습니다.

이 장의 다음 단원은 Web Broker 컴포넌트를 사용하여 웹 서버 애플리케이션을 만드는 방법을 설명합니다.

Web Broker로 웹 서버 애플리케이션 생성

다음과 같은 Web Broker 아키텍처를 사용하여 새로운 웹 서버 애플리케이션을 생성할 수 있습니다.

1. File|New를 선택합니다.
2. New Items 대화 상자에서 New 탭을 선택하여 웹 서버 애플리케이션을 선택합니다.
3. 대화 상자에서 웹 서버 애플리케이션의 유형 중 하나를 선택합니다.
   - CGI stand-alone executable: 이 애플리케이션 유형을 선택하면 프로젝트가 콘솔 애플리케이션으로 설정되고 프로젝트 파일의 uses 절에 필수 항목이 추가됩니다.
   - Apache Shared Module(DSO): 이 애플리케이션 유형을 선택하면 프로젝트는 Apache 웹 서버에서 요구하는 export된 메소드를 가진 동적 공유 객체(DSO)로 설정됩니다. 프로젝트 파일에 라이브러리 헤더를 추가하고 사용 목록과 프로젝트 파일의 exports 절에 필수 항목을 추가합니다.

애플리케이션이 사용하게 될 웹 서버 유형과 통신하는 웹 서버 애플리케이션의 유형을 선택합니다. 그러면 인터넷 컴포넌트를 사용하도록 구성되고 빈 웹 모듈을 포함하는 새 프로젝트가 만들어집니다.

웹 모듈

웹 모듈(*TWebModule*)은 *TDataModule*의 자손이며 웹 애플리케이션의 비즈니스 룰과 넌비주얼 컴포넌트에 대해서 중앙에서 일괄적으로 제어하기 위해 조상과 동일한 방법으로 사용됩니다.

애플리케이션이 응답 메시지를 생성하는 데 사용하는 컨텐트 프로듀서를 추가합니다. 이러한 프로듀서에는 *TPageProducer*와 같은 기본 제공 컨텐트 프로듀서나 사용자가 직접 작성한 *TCustomContentProducer*의 자손이 있습니다. 사용자의 애플리케이션이 데이터베이스에서 가져온 자료를 포함한 응답 메시지를 생성하면 사용자는 데이터 액세스 컴포넌트 또는 다계층 데이터베이스 애플리케이션에서 클라이언트로서 작동하는 웹 서버를 작성하는 특수한 컴포넌트를 추가할 수 있습니다.

웹 모듈은 넌비주얼 컴포넌트와 비즈니스 룰을 저장하는 작업뿐만 아니라 들어오는 HTTP 요청 메시지에 응답할 액션 항목을 찾아 주는 디스패처 역할도 수행합니다.

웹 애플리케이션에서 사용할 많은 넌비주얼 컴포넌트와 비즈니스 룰과 함께 설정된 데이터 모듈을 이미 가지고 있을 수도 있습니다. 이럴 경우 웹 모듈을 기존 데이터 모듈로 바꿀 수 있습니다. 자동으로 생성된 웹 모듈을 그냥 삭제하고 사용자의 데이터 모듈로 바꿉니다. 그런 다음 웹 모듈의 경우처럼 데이터 모듈에 *TWebDispatcher* 컴포넌트를 추가하여 요청 메시지를 액션 항목에 디스패치할 수 있도록 합니다. 들어오는 HTTP 요청 메시지에 응답할 액션 항목이 선택되는 방법을 변경하려면 *TCustomWebDispatcher*에서 새 디스패처 컴포넌트를 파생시켜 데이터 모듈에 추가합니다.

프로젝트는 디스패처를 단 하나만 포함할 수 있습니다. 디스패처는 프로젝트를 만들 때 자동으로 생성된 웹 모듈일 수도 있고, 웹 모듈을 대체하는 데이터 모듈에 추가된 *TWebDispatcher* 컴포넌트일 수도 있습니다. 실행 중에 디스패처가 포함된 두 번째 데이터 모듈이 생성되면 웹 서버 애플리케이션은 런타임 오류를 발생시킵니다.

참고 디자인 타임에 설정한 웹 모듈은 실제로 템플릿입니다.

웹 애플리케이션 객체

웹 애플리케이션용으로 설정된 프로젝트에는 *Application*이라는 전역 변수가 포함되어 있습니다. *Application*은 사용자가 만들려는 애플리케이션 유형에 적당한 *TWebApplication*(*TApacheApplication* 또는 *TCGIApplication*) 의 자손이며 웹 서버가 수신한 HTTP 요청 메시지에 응답하여 실행됩니다.

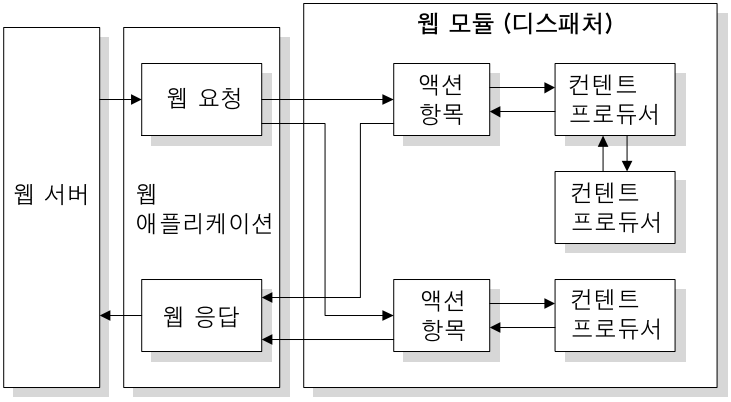
경고 프로젝트의 **uses** 절에서 CGIApp 또는 ApacheApp 유닛 뒤에 폼 유닛을 포함하면(include) 안 됩니다. 폼도 *Application*이라는 전역 변수를 선언하기 때문에 CGIApp 또는 ApacheApp 유닛 뒤에 올 경우 *Application*이 잘못된 유형의 객체로 초기화됩니다.

Web Broker 애플리케이션 구조

웹 애플리케이션은 HTTP 요청 메시지를 수신하면 *TWebRequest* 객체를 만들어 HTTP 요청 메시지를 나타내고 *TWebResponse* 객체를 만들어 반환해야 할 응답을 나타냅니다. 그런 다음 웹 애플리케이션은 이 객체들을 웹 디스패처(웹 모듈 또는 *TWebDispatcher* 컴포넌트)에 전달합니다.

웹 디스패처는 웹 서버 애플리케이션의 흐름을 제어합니다. 디스패처는 특정 유형의 HTTP 요청 메시지에 대한 처리 방법을 알고 있는 액션 항목(*TWebActionItem*)의 컬렉션을 유지 관리합니다. 디스패처는 적절한 액션 항목 또는 자동 디스패칭 컴포넌트를 식별하여 HTTP 요청 메시지를 처리하며 요청된 모든 액션을 수행하거나 응답 메시지를 생성할 수 있도록 요청 및 응답 객체를 식별된 핸들러에 전달합니다. 자세한 내용은 7-4페이지의 "웹 디스패처"에 설명되어 있습니다.

그림 7.1 서버 애플리케이션의 구조



액션 항목은 요청을 읽고 응답 메시지를 결합합니다. 특수한 컨텐트 프로듀서 컴포넌트는 액션 항목이 사용자 지정 HTML 코드 또는 다른 MIME 컨텐트를 포함할 수 있는 응답 메시지의 컨텐트를 동적으로 생성하는 데 도움을 줍니다. 컨텐트 프로듀서는 다른 컨텐트 프로듀서나 *THTMLTagAttributes*의 자손을 사용하여 액션 항목이 응답 메시지의 컨텐트를 만들 수 있게 합니다. 컨텐트 프로듀서에 대한 자세한 내용은 7-12페이지의 "응답 메시지 컨텐트 생성"을 참조하십시오.

모든 액션 항목 또는 자동 디스패칭 컴포넌트가 *TWebResponse* 객체에 데이터를 채워 응답 생성을 끝내면 디스패처는 결과를 웹 애플리케이션에 전달합니다. 그러면 웹 애플리케이션은 웹 서버를 통해 클라이언트에게 응답을 보냅니다.

웹 디스패처

웹 모듈을 사용하고 있으면 웹 모듈이 웹 디스패처 역할도 합니다. 이미 존재하는 데이터 모듈을 사용하는 경우 *TWebDispatcher* 디스패처 컴포넌트 하나를 이 데이터 모듈에 추가해야 합니다. 디스패처는 특정 유형의 HTTP 요청 메시지에 대한 처리 방법을 알고 있는 액션 항목의 컬렉션을 유지 관리합니다. 웹 애플리케이션이 요청 객체와 응답 객체를 디스패처에 전달하면 디스패처는 요청에 응답하기 위해 하나 이상의 액션 항목을 선택합니다.

디스패처에 액션 추가

Object Inspector에서 디스패처의 *Actions* 속성의 생략 버튼을 클릭하여 액션 에디터를 엽니다. 이 액션 에디터에서 Add 버튼을 클릭하면 디스패처에 액션 항목을 추가할 수 있습니다.

다른 요청 메소드나 대상 URI에 응답하려면 디스패처에 액션을 추가합니다. 다양한 방법으로 액션 항목을 설정할 수 있습니다. 요청을 미리 처리하는 액션 항목으로 시작한 후, 응답 전송 또는 오류 코드 반환의 응답 완료 여부를 검사하는 기본 액션으로 끝낼 수 있습니다. 또는 각 액션 항목이 요청을 완벽하게 처리할 수 있도록 모든 유형의 요청에 대해 별도의 액션 항목을 추가할 수 있습니다.

액션 항목은 7-5페이지의 "액션 항목"에 자세히 설명되어 있습니다.

요청 메시지 디스패칭

디스패처는 클라이언트 요청을 수신하면 *BeforeDispatch* 이벤트를 생성합니다. 이 이벤트가 생성됨으로써 액션 항목 전에 사용자가 애플리케이션에서 요청 메시지를 미리 처리할 수 있습니다.

다음으로 디스패처는 요청 메시지의 대상 URL의 PathInfo 부분과 일치하는 항목과 요청 메시지의 메소드로 지정된 서비스를 제공하는 항목을 찾기 위해서 항목을 액션 항목 목록에서 계속 반복해서 찾습니다. 디스패처는 이 작업을 *TWebRequest* 객체의 *PathInfo* 및 *MethodType* 속성과 액션 항목에 있는 같은 이름의 속성과 비교해서 수행합니다.

적절한 액션 항목을 찾으면 디스패처는 해당 액션 항목을 실행시키며 이 때 액션 항목은 다음 중 하나를 실행합니다.

- 응답 컨텐트를 완성하고 요청이 완전히 처리된 응답 또는 시그널을 보냅니다.
- 응답을 추가하고 다른 액션 항목이 작업을 완료할 수 있게 합니다.
- 요청을 다른 액션 항목에 전달합니다.

모든 액션 항목과 특별히 등록된 자동 디스패칭 컴포넌트를 검사한 후에도 여전히 요청 메시지가 모두 처리되지 않으면 디스패처는 기본 액션 항목을 호출합니다. 기본 액션 항목은 대상 URL이나 요청 메소드와 반드시 일치할 필요는 없습니다.

기본 액션이 있어서 디스패처가 기본 액션이 포함된 액션 리스트의 끝에 도달했고 트리거된 액션이 없을 경우 서버에는 아무것도 전달되지 않습니다. 이 경우 서버는 클라이언트와의 연결을 끊습니다.

요청이 액션 항목에 의해 처리될 경우 디스패처는 *AfterDispatch* 이벤트를 생성합니다. 이 이벤트가 생성되면 애플리케이션은 생성된 응답을 검사하여 최종적으로 필요한 내용을 변경할 수 있습니다.

액션 항목

각 액션 항목(*TWebActionItem*)은 주어진 유형의 요청 메시지에 응답하여 특정 작업을 수행합니다.

액션 항목은 요청에 대한 완전한 응답을 전송하거나 응답의 일부를 수행하고 다른 액션 항목이 작업을 완료하도록 할 수 있습니다. 액션 항목은 요청에 대해 HTTP 응답 메시지를 전송하거나 다른 액션 항목이 완료할 수 있도록 응답의 일부만 설정할 수 있습니다. 액션 항목이 응답을 완료했지만 전송하지 않았을 경우 웹 서버 애플리케이션이 응답 메시지를 보냅니다.

액션 항목 실행 시기 결정

대부분의 액션 항목 속성은 HTTP 요청 메시지를 처리하기 위해 디스패처가 액션 항목을 선택하는 시기를 결정합니다. 액션 항목의 속성을 설정하려면 Object Inspector에서 디스패처의 *Actions* 속성을 선택하고 생략 버튼을 클릭하여 액션 에디터를 열어야 합니다. 액션 에디터에서 액션을 선택하면 Object Inspector에서 해당 액션의 속성을 수정할 수 있습니다.

대상 URL

디스패처는 액션 항목의 *PathInfo* 속성과 요청 메시지의 *PathInfo*를 비교합니다. 이 속성 값은 액션 항목이 처리할 준비가 된 모든 요청의 URL에 있는 경로 정보 부분이어야 합니다. 예를 들어, 다음과 같은 URL이 있다고 가정합니다.

```
http://www.TSite.com/art/gallery/mammals?animal=dog&color=black
```

여기서 /gallery 부분이 웹 서버 애플리케이션을 나타낸다고 가정하면 다음 부분이 경로 정보입니다.

```
/mammals
```

요청을 서비스할 때 웹 애플리케이션이 정보를 검색해야 할 장소를 나타내거나 웹 애플리케이션을 논리적 하위 서비스로 나누기 위해 경로 정보를 사용합니다.

요청 메소드 유형

액션 항목의 *MethodType* 속성은 처리할 수 있는 요청 메시지 유형을 나타냅니다. 디스패처는 액션 항목의 *MethodType* 속성과 요청 메시지의 *MethodType*을 비교합니다. *MethodType* 값은 다음 중 하나입니다.

표 7.1 MethodType 값

값	의미
mtGet	요청에서 응답 메시지로 반환될 대상 URI와 연결된 정보를 요구합니다.
mtHead	요청에서 *mtGet* 요청에 서비스하지만 응답 컨텐트를 생략하는 경우처럼 응답의 헤더 속성을 요구합니다.
mtPost	요청에서 웹 애플리케이션에 포스트한 정보를 제공합니다.
mtPut	요청에서 대상 URI와 연결된 리소스를 요청 메시지의 컨텐트로 바꾸도록 요구합니다.
mtAny	*mtGet*, *mtHead*, *mtPut* 및 *mtPost*를 비롯한 모든 요청 메소드 유형을 일치시킵니다.

액션 항목 활성화 및 비활성화

각 액션 항목에는 해당 액션 항목을 활성화 또는 비활성화할 수 있는 *Enabled* 속성이 있습니다. *Enabled*를 *False*로 설정하면 액션 항목이 비활성화되므로 디스패처가 요청을 처리하기 위해 액션 항목을 검색할 때 False로 설정된 액션 항목은 고려하지 않습니다.

BeforeDispatch 이벤트 핸들러는 디스패처가 액션 항목을 요청 메시지와 일치시키기 전에 액션 항목의 *Enabled* 속성을 변경하여 액션 항목이 처리해야 할 요청을 제어할 수 있습니다.

주의 실행 도중에 액션의 *Enabled* 속성을 변경하면 후속 요청에 대해 예기치 않은 결과가 발생할 수 있습니다. 모든 액션 항목을 해당 시작 상태로 올바르게 초기화하려면 *BeforeDispatch* 이벤트를 사용합니다.

기본 액션 항목 선택

액션 항목 중 하나만 기본 액션 항목이 될 수 있습니다. 기본 액션 항목을 선택하려면 해당 액션 항목의 *Default* 속성을 *True*로 설정합니다. 특정 액션 항목의 *Default* 속성이 *True*로 설정되면 기존의 기본 액션 항목이 있을 경우에는 그 *Default* 속성이 *False*로 설정이 변경됩니다.

디스패처는 요청 처리를 위해 액션 항목 목록을 검색하여 특정 액션 항목을 선택할 때 기본 액션 항목의 이름을 저장합니다. 액션 항목 목록의 끝에 도달했지만 요청이 완전히 처리되지 않은 경우 디스패처는 기본 액션 항목을 실행합니다.

디스패처는 기본 액션 항목의 *PathInfo* 또는 *MethodType*을 검사하지 않습니다. 또한 디스패처는 기본 액션 항목의 *Enabled* 속성도 검사하지 않습니다. 따라서 기본 액션 항목의 *Enabled* 속성을 *False*로 설정하면 기본 액션 항목이 끝 부분에서만 호출되게 할 수 있습니다.

잘못된 URI 또는 *MethodType*을 나타내는 오류 코드만 반환될 경우에도 기본 액션 항목은 발생하는 모든 요청을 처리하도록 준비되어야 합니다. 기본 액션 항목이 요청을 처리하지 않을 경우 웹 클라이언트에 응답이 전송되지 않습니다.

주의 실행 도중에 액션의 *Default* 속성을 변경하면 현재 요청에 대해 예기치 않은 결과가 발생할 수 있습니다. 이미 트리거된 액션의 *Default* 속성을 *True*로 설정하면 해당 액션은 다시 평가(evaluate)되지 않고 액션 리스트의 끝에 도달했을 때 디스패처는 이 액션을 실행하지 않습니다.

액션 항목으로 요청 메시지에 응답

액션 항목은 실행 시 웹 서버 애플리케이션의 실제 작업을 수행합니다. 웹 디스패처가 액션 항목을 실행시키면 해당 액션 항목은 다음 두 가지 방법으로 현재 요청 메시지에 응답할 수 있습니다.

- 액션 항목이 *Producer* 속성 값으로 연결된 프로듀서 컴포넌트를 가질 경우 해당 프로듀서는 *Content* 메소드를 사용하여 응답 메시지의 *Content*를 자동으로 지정합니다. 컴포넌트 팔레트의 인터넷 페이지에는 응답 메시지 컨텐트의 HTML 페이지를 만드는 데 도움이 되는 여러 컨텐트 프로듀서 컴포넌트가 포함되어 있습니다.
- 연결된 프로듀서가 있고 이 프로듀서가 응답 내용을 지정한 후에 액션 항목은 *OnAction* 이벤트를 수신합니다. *OnAction* 이벤트 핸들러에는 HTTP 요청 메시지를 나타내는 *TWebRequest* 객체와 모든 응답 정보를 채울 *TWebResponse* 객체가 전달됩니다.

하나의 컨텐트 프로듀서가 액션 항목의 내용을 생성할 수 있을 경우 가장 간단한 방법은 컨텐트 프로듀서를 액션 항목의 *Producer* 속성으로 지정하는 것입니다. 그러나 *OnAction* 이벤트 핸들러에서 언제든지 모든 컨텐트 프로듀서에 액세스할 수 있습니다. *OnAction* 이벤트 핸들러는 여러 컨텐트 프로듀서를 사용하고 응답 메시지 속성을 지정하는 등의 작업을 수행할 수 있도록 더 많은 유연성을 부여합니다.

컨텐트 프로듀서 컴포넌트와 *OnAction* 이벤트 핸들러는 모두 객체 또는 런타임 라이브러리 메소드를 사용하여 요청 메시지에 응답합니다. 또한 데이터베이스 액세스, 계산 수행, HTML 문서 만들기 또는 선택 등의 작업을 할 수 있습니다. 컨텐트 프로듀서 컴포넌트를 사용한 응답 컨텐트 생성에 대한 자세한 내용은 7-12페이지의 "응답 메시지 컨텐트 생성"을 참조하십시오.

응답 보내기

OnAction 이벤트 핸들러는 *TWebResponse* 객체의 메소드를 사용하여 웹 클라이언트에게 응답을 돌려 보낼 수 있습니다. 그러나 클라이언트에게 응답을 보내는 액션 항목이 없으면 마지막 액션 항목이 요청이 처리되었다는 것을 알 때까지 웹 서버 애플리케이션에서 계속하여 요청을 전송합니다.

여러 가지 액션 항목 사용

단일 액션 항목에서 요청에 응답하거나 여러 가지 액션 항목 간에 작업을 분할할 수 있습니다. 액션 항목이 응답 메시지 설정을 완전히 끝내지 않았을 경우 액션 항목은 *Handled* 매개변수를 *False*로 설정하여 *OnAction* 이벤트 핸들러에서 이러한 상태를 시그널로 보내야 합니다.

여러 액션 항목 간에 요청 메시지에 응답하는 작업이 분할되면 다른 액션 항목이 작업을 계속할 수 있도록 각 *Handled* 매개변수를 *False*로 설정합니다. 이 때 기본 액션 항목의 *Handled* 매개변수는 *True*로 설정된 상태로 두어야 합니다. 그렇지 않으면 웹 클라이언트에게 응답이 전송되지 않습니다.

여러 액션 항목 간에 작업을 분할할 때 기본 액션 항목의 *OnAction* 이벤트 핸들러 또는 디스패처의 *AfterDispatch* 이벤트 핸들러는 모든 작업이 수행되었는지 검사하고 그렇지 않을 경우 적절한 오류 코드를 설정해야 합니다.

클라이언트 요청 정보 액세스

웹 서버 애플리케이션이 HTTP 요청 메시지를 수신하면 클라이언트 요청의 헤더가 *TWebRequest* 객체의 속성으로 로드됩니다. 콘솔 CGI 애플리케이션은 *TCGIRequest* 객체를 사용하고 Apache 애플리케이션은 *TApacheRequest* 객체를 사용합니다.

요청 객체의 속성은 읽기 전용입니다. 이러한 속성을 사용하여 클라이언트 요청에서 사용할 수 있는 모든 정보를 수집할 수 있습니다.

요청 헤더 정보를 포함하는 속성

요청 객체에 있는 대부분의 속성은 HTTP 요청 헤더에서 가져온 요청에 대한 정보를 포함합니다. 그러나 모든 요청이 이러한 속성에 값을 제공하는 것은 아닙니다. 특히 HTTP 표준이 계속 향상됨에 따라 일부 요청은 요청 객체의 속성으로 표시되지 않는 헤더 필드를 포함할 수도 있습니다. 요청 객체 속성으로 표시되지 않는 요청 헤더 필드의 값을 얻으려면 *GetFieldByName* 메소드를 사용하십시오.

대상을 식별하는 속성

요청 메시지의 전체 대상은 *URL* 속성에서 제공합니다. 보통 대상은 프로토콜(HTTP), *Host*(서버 시스템), *ScriptName*(서버 애플리케이션), *PathInfo*(호스트 상의 위치), *Query*로 분리할 수 있는 URL입니다.

이러한 각 부분은 고유한 속성으로 표시됩니다. 프로토콜은 항상 HTTP이고 *Host* 및 *ScriptName*은 웹 서버 애플리케이션을 식별합니다. 디스패처는 액션 항목을 요청 메시지에 일치시킬 때 *PathInfo* 부분을 사용합니다. 일부 요청에서는 요청된 정보의 세부 사항을 지정하기 위해 *Query*가 사용되며 이 부분의 값도 *QueryFields* 속성으로 구문 분석됩니다.

웹 클라이언트를 설명하는 속성

요청에는 요청이 시작된 장소에 대한 정보를 제공하는 몇 가지 속성도 포함되어 있습니다. 이러한 속성에는 보낸 사람의 전자 메일 주소(*From* 속성)에서 메시지가 시작된 URI(*Referer* 또는 *RemoteHost* 속성)에 이르기까지 모든 정보가 포함됩니다. 요청에 컨텐트가 포함되어 있고 이 컨텐트가 요청과 동일한 URI에서 나온 것이 아닌 경우 *DerivedFrom* 속성에서 컨텐트의 소스를 찾을 수 있습니다. 이외에도 클라이언트의 IP 주소(*RemoteAddr* 속성) 및 요청을 보낸 애플리케이션의 이름과 버전(*UserAgent* 속성)을 확인할 수 있습니다.

요청 목적을 식별하는 속성

Method 속성은 요청 메시지가 서버 애플리케이션에 요구하는 작업을 설명하는 문자열입니다. HTTP 1.1 표준은 다음 메소드를 정의합니다.

값	메시지가 요청하는 내용
OPTIONS	사용할 수 있는 통신 옵션에 대한 정보
GET	*URL* 속성에 의해 식별된 정보
HEAD	응답 컨텐트 없이 해당되는 GET 메시지에서 가져온 헤더 정보
POST	*Content* 속성에 포함된 데이터를 적절하게 포스트하도록 서버 애플리케이션에 요청
PUT	*URL* 속성이 나타낸 리소스를 *Content* 속성에 포함된 데이터로 바꾸도록 서버 애플리케이션에 요청
DELETE	*URL* 속성에 의해 식별되는 리소스를 삭제하거나 숨기도록 서버 애플리케이션에 요청
TRACE	요청 수신을 확인하는 루프백을 보내도록 서버 애플리케이션에 요청

Method 속성은 웹 클라이언트가 요청하는 서버의 다른 메소드를 나타낼 수 있습니다.

웹 서버 애플리케이션은 *Method*의 모든 가능한 값에 대해 응답을 제공할 필요가 없습니다. 단, HTTP 표준에서는 GET 및 HEAD 요청 모두에 대해 서비스를 요구합니다.

MethodType 속성은 *Method* 값이 GET(mtGet), HEAD(mtHead), POST(mtPost), PUT(mtPut) 또는 다른 문자열(mtAny) 중에서 무엇인지 나타냅니다. 디스패처는 *MethodType* 속성의 값을 각 액션 항목의 *MethodType*과 일치시킵니다.

예상 응답을 설명하는 속성

Accept 속성은 웹 클라이언트가 응답 메시지의 컨텐트로 승인하는 미디어 유형을 나타냅니다. *IfModifiedSince* 속성은 클라이언트가 최근에 변경된 정보만 원하는지 여부를 지정합니다. *Cookie* 속성은 응답을 수정할 수 있는 상태 정보를 포함하고 있는데 이 정보는 이전에 애플리케이션이 추가한 것입니다.

컨텐트를 설명하는 속성

대부분의 요청은 정보에 대한 요청이기 때문에 컨텐트를 포함하지 않습니다. 그러나 POST 요청과 같은 일부 요청은 웹 서버 애플리케이션이 사용할 컨텐트를 제공합니다. 컨텐트의 미디어 유형은 *ContentType* 속성에서 제공되고 그 길이는 *ContentLength* 속성에서 제공됩니다. 데이터 압축을 위해 메시지 컨텐트가 인코딩된 경우에는 정보가 *ContentEncoding* 속성에 들어 있습니다. 컨텐트를 만든 애플리케이션의 이름과 버전 번호는 *ContentVersion* 속성에 의해 지정됩니다. 또한 *Title* 속성이 컨텐트에 대한 정보를 제공할 수도 있습니다.

HTTP 요청 메시지의 내용

헤더 필드 이외에 일부 요청 메시지는 웹 서버 애플리케이션이 몇몇 방법으로 처리해야 할 컨텐트 부분을 포함합니다. 예를 들어, POST 요청은 웹 서버 애플리케이션이 유지 관리하는 데이터베이스에 추가해야 할 정보를 포함할 수 있습니다.

컨텐트의 처리되지 않은 값은 *Content* 속성에서 제공됩니다. 컨텐트를 앰퍼샌드(&)로 분리된 필드로 구문 분석할 수 있는 경우 구문 분석된 버전은 *ContentFields* 속성에 있습니다.

HTTP 응답 메시지 만들기

들어오는 HTTP 요청 메시지에 대해 *TWebRequest* 객체를 만들 경우 웹 서버 애플리케이션은 반환 시 전송될 응답 메시지를 나타내는 해당 *TWebResponse* 객체도 함께 만듭니다. Apache DSO 애플리케이션에서 응답 메시지는 *TApacheResponse* 객체로 캡슐화합니다. 콘솔 CGI 애플리케이션은 *TCGIResponse* 객체를 사용합니다.

웹 클라이언트 요청에 대한 응답을 생성하는 액션 항목은 응답 객체의 속성을 채웁니다. 어떤 경우는 오류 코드를 반환하거나 요청을 다른 URI에 리디렉션하는 것만큼 작업이 간단할 수 있지만 또 다른 경우 다른 소스에서 정보를 fetch하여 완성된 폼으로 결합하도록 액션 항목에 요청하는 복잡한 계산을 포함할 수 있습니다. 요청 메시지가 단지 요청된 액션이 수행되었다는 사실을 알리는 것에 불과할지라도 대부분의 요청 메시지는 응답을 필요로 합니다.

응답 헤더 채우기

TWebResponse 객체의 속성 대부분은 웹 클라이언트로 돌려 보내는 HTTP 응답 메시지의 헤더 정보를 나타냅니다. 액션 항목은 자신의 *OnAction* 이벤트 핸들러에서 이러한 속성을 설정합니다.

모든 응답 메시지가 헤더 속성 모두에 대해 값을 지정할 필요는 없습니다. 설정해야 할 속성은 요청의 특성과 응답 상태에 따라 달라집니다.

응답 상태 표시

모든 응답 메시지는 응답 상태를 나타내는 상태 코드를 포함해야 합니다. 상태 코드는 *StatusCode* 속성을 설정하여 지정할 수 있습니다. HTTP 표준은 의미가 이미 정의된 여러 개의 표준 상태 코드를 정의합니다. 이외에도 사용자는 사용되지 않은 가능한 값을 사용하여 고유한 상태 코드를 정의할 수 있습니다.

각 상태 코드는 다음과 같이 세 자리 숫자이며 여기서 최상위 숫자가 응답의 종류를 나타냅니다.

- 1xx: 정보(요청을 수신했지만 완전히 처리하지 않았습니다.)
- 2xx: 성공(요청을 수신한 다음, 이해하고 승인했습니다.)
- 3xx: 리디렉션(요청을 완료하기 위해 추가 액션이 필요합니다.)
- 4xx: 클라이언트 오류(요청을 이해하거나 서비스할 수 없습니다.)
- 5xx: 서버 오류(요청은 유효하지만 서버가 요청을 처리할 수 없습니다.)

각 상태 코드에 연결된 문자열은 상태 코드의 의미를 설명합니다. 이 문자열은 *ReasonString* 속성에서 제공됩니다. 이미 정의된 상태 코드의 경우 *ReasonString* 속성을 설정할 필요가 없습니다. 고유한 상태 코드를 정의할 경우에는 *ReasonString* 속성도 설정해야 합니다.

클라이언트 액션에 대한 요구 표시

상태 코드의 범위가 300–399 사이인 경우 웹 서버 애플리케이션이 요청을 완료할 수 있으려면 클라이언트가 추가 액션을 수행해야 합니다. 클라이언트를 다른 URI로 리디렉션해야 하거나 요청 처리를 위해 새 URI를 만들었다는 것을 나타내야 한다면 *Location* 속성을 설정합니다. 클라이언트가 암호를 제공해야 사용자가 작업을 계속 진행할 수 있는 경우에는 *WWWAuthenticate* 속성을 설정합니다.

서버 애플리케이션 설명

일부 응답 헤더 속성은 웹 서버 애플리케이션의 기능을 설명합니다. *Allow* 속성은 애플리케이션이 응답할 수 있는 메소드를 나타냅니다. *Server* 속성은 응답을 생성하는 데 사용되는 애플리케이션의 이름과 버전 번호를 제공합니다. *Cookies* 속성은 후속 요청 메시지에 포함되는 클라이언트의 서버 애플리케이션 사용에 대한 상태 정보를 유지할 수 있습니다.

컨텐트 설명

몇 가지 속성들은 응답 컨텐트를 설명합니다. *ContentType*은 응답의 미디어 유형을 제공하고 *ContentVersion*은 해당 미디어 유형에 대한 버전 번호이며 *ContentLength*는 응답의 길이를 제공합니다. 데이터 압축의 경우와 같이 컨텐트가 인코딩된 경우 *ContentEncoding* 속성을 사용하여 나타냅니다. 컨텐트를 다른 URI에서 가져온 경우 *DerivedFrom* 속성에서 이를 나타내야 합니다. 컨텐트 값이 시간과 밀접한 관계가 있는 경우 *LastModified* 및 해당 값이 여전히 유효한지 여부를 나타내는 *Expires* 속성이 있습니다. *Title* 속성은 컨텐트에 대한 설명 정보를 제공할 수 있습니다.

응답 컨텐트 설정

일부 요청의 경우 요청 메시지에 대한 응답이 응답의 헤더 속성에 전부 포함됩니다. 그러나 대부분의 경우 액션 항목은 응답 메시지에 일부 컨텐트를 지정합니다. 파일에 저장된 정적 정보나 액션 항목 또는 컨텐트 프로듀서에 의해 동적으로 만들어진 정보도 컨텐트가 될 수 있습니다.

Content 속성이나 *ContentStream* 속성을 사용하여 응답 메시지의 컨텐트를 설정할 수 있습니다.

Content 속성은 문자열입니다. Kylix 문자열은 텍스트 값에만 국한되지 않으므로 *Content* 속성 값은 HTML 명령의 문자열 또는 비트 스트림과 같은 그래픽 컨텐트나 다른 MIME 컨텐트 형식이 될 수 있습니다.

응답 메시지의 컨텐트를 스트림에서 읽을 수 있는 경우 *ContentStream* 속성을 사용합니다. 예를 들어 응답 메시지가 파일 컨텐트를 전송해야 할 경우에는 *ContentStream* 속성에 대해 *TFileStream* 객체를 사용합니다. *Content* 속성과 마찬가지로 *ContentStream*은 HTML 명령의 문자열이나 다른 MIME 컨텐트 형식을 제공할 수 있습니다.
ContentStream 속성을 사용할 경우에는 웹 응답 객체가 자동으로 스트림을 해제하므로 직접 해제하지 마십시오.

참고 *ContentStream* 속성 값이 **nil**이 아닌 경우 *Content* 속성은 무시됩니다.

응답 보내기

요청 메시지에 대한 응답에서 더 이상 수행할 작업이 없다는 것이 확실하면 *OnAction* 이벤트 핸들러에서 응답을 직접 보낼 수 있습니다. 응답 객체는 응답을 보내기 위한 두 가지 메소드인 *SendResponse* 및 *SendRedirect*를 제공합니다. *TWebResponse* 객체의 모든 헤더 속성과 지정된 컨텐트를 사용하는 응답을 전송하려면 *SendResponse*를 호출합니다. 단지 웹 클라이언트를 다른 URI로 리디렉션해야 할 경우에는 *SendRedirect* 메소드가 더 효율적입니다.

응답을 보내는 이벤트 핸들러가 없을 경우 웹 애플리케이션 객체는 디스패처가 종료된 후 응답을 전송합니다. 그러나 응답을 처리했다는 사실을 나타내는 액션 항목이 없으면 웹 애플리케이션은 응답을 보내지 않고 웹 클라이언트와의 연결을 끊습니다.

응답 메시지 컨텐트 생성

Kylix에서는 액션 항목이 HTTP 응답 메시지의 컨텐트를 만들 수 있도록 하는 여러 가지 객체를 제공합니다. 이 객체들을 사용하면 파일에 저장되거나 웹 클라이언트로 직접 돌려 보내는 HTML 명령의 문자열을 생성할 수 있습니다. *TCustomContentProducer* 또는 자손 중 하나에서 컨텐트 프로듀서를 파생시켜 자신만의 고유한 컨텐트 프로듀서를 작성할 수 있습니다.

*TCustomContentProducer*는 HTTP 응답 메시지로 MIME 형식을 생성하는 일반 인터페이스를 제공하며 이 프로듀서의 자손에는 페이지 프로듀서와 테이블 프로듀서가 포함됩니다.

- 페이지 프로듀서는 사용자 지정 HTML 코드로 교체할 특수 태그를 HTML 문서에서 스캔합니다. 페이지 프로듀서에 대한 자세한 내용은 다음 단원에서 설명합니다.
- 테이블 프로듀서는 데이터셋의 정보에 기초하여 HTML 명령을 만듭니다. 자세한 내용은 7-16페이지의 "응답에서 데이터베이스 정보 사용"에 설명되어 있습니다.

페이지 프로듀서 컴포넌트 사용

페이지 프로듀서(*TPageProducer*와 자손 컴포넌트)는 HTML 템플릿을 가져온 다음 특수 HTML 투명 태그를 사용자 지정 HTML 코드로 교체하여 템플릿을 변환합니다. 사용자는 HTTP 요청 메시지에 대한 응답을 생성해야 할 경우 페이지 프로듀서가 채우는 표준 응답 템플릿 집합을 저장할 수 있습니다. 또한 페이지 프로듀서를 서로 연결하면 HTML 투명 태그를 연속적으로 구체화하여 HTML 문서를 반복적으로 작성할 수 있습니다.

HTML 템플릿

HTML 템플릿은 일련의 HTML 명령과 HTML 투명 태그들로 구성됩니다. HTML 투명 태그는 다음과 같은 형태를 갖습니다.

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

꺽쇠 괄호(< 및 >)는 태그의 전체 유효 범위(scope)를 정의합니다. 열린 꺽쇠 괄호(<) 바로 뒤에는 공백 없이 파운드 기호(#)가 옵니다. 파운드 기호는 페이지 프로듀서에 대한 문자열을 HTML 투명 태그로 식별합니다. 파운드 기호 바로 뒤에는 공백 없이 태그 이름이 옵니다. 태그 이름은 유효한 식별자가 될 수 있고 태그가 나타내는 변환 유형을 식별합니다.

태그 이름 다음에 오는 HTML 투명 태그는 수행할 변환에 대한 세부 정보를 지정하는 매개변수를 포함할 수 있습니다. 각 매개변수는 *ParamName=Value*의 형태를 가지며 등호 기호(=) 양쪽에 공백이 있으면 안 됩니다. 각 매개변수들은 공백으로 구분합니다.

꺽쇠 괄호(< 및 >)는 #TagName 구조를 인식하지 않는 HTML 브라우저에서 태그를 인식하도록 합니다.

자신의 고유한 HTML 투명 태그를 만들어 페이지 프로듀서가 처리하는 모든 종류의 정보를 나타낼 수도 있지만 *TTag* 데이터 형식 값에 연결된 여러 가지 태그 이름들이 이미 정의되어 있습니다. 이와 같이 이미 정의된 태그 이름은 응답 메시지에 따라 달라질 수 있는 HTML 명령에 해당합니다. 다음 표는 이러한 태그 이름을 나열한 것입니다.

태그 이름	TTag 값	태그가 변환되어야 하는 대상
Link	*tgLink*	하이퍼텍스트 링크. 결과는 <A> 태그로 시작하여 </A> 태그로 끝나는 HTML입니다.
Image	*tgImage*	그래픽 이미지. 결과는 HTML <IMG> 태그입니다.
Table	*tgTable*	HTML 테이블. 결과는 <TABLE> 태그로 시작하여 </TABLE> 태그로 끝나는 HTML입니다.
ImageMap	*tgImageMap*	연결된 핫 존(hot zone)이 있는 그래픽 이미지. 결과는 <MAP> 태그로 시작하여 </MAP> 태그로 끝나는 HTML입니다.

다른 모든 태그 이름은 *tgCustom*에 연결됩니다. 페이지 프로듀서는 이미 정의된 태그 이름의 기본 제공 처리를 제공하지 않습니다. 이미 정의된 태그 이름은 애플리케이션이 변환 프로세스를 더 일반적인 작업으로 구성할 수 있도록 하기 위해 제공됩니다.

참고 이미 정의된 태그 이름은 대소문자를 구분하지 않습니다.

HTML 템플릿 지정

페이지 프로듀서는 HTML 템플릿을 지정하는 방법에 대해 여러 가지 선택 사항을 제공합니다. *HTMLFile* 속성을 HTML 템플릿을 포함하는 파일의 이름으로 설정할 수 있고, *HTMLDoc* 속성을 HTML 템플릿을 포함하는 *TStrings* 객체로 설정할 수 있습니다. *HTMLFile* 속성 또는 *HTMLDoc* 속성을 사용하여 템플릿을 지정하는 경우 *Content* 메소드를 호출하여 변환된 HTML 명령을 생성할 수 있습니다.

이외에도 *ContentFromString* 메소드를 호출하여 매개변수로 전달되는 단일 문자열인 HTML 템플릿을 직접 변환할 수 있습니다. 또한 *ContentFromStream* 메소드를 호출하여 스트림에서 HTML 템플릿을 읽을 수 있습니다. 따라서 예를 들면 모든 HTML 템플릿을 데이터베이스의 메모 필드에 저장하고 *ContentFromStream* 메소드를 사용하여 변환된 HTML 명령을 얻을 수 있으며 이 때 *TBlobStream* 객체에서 직접 템플릿을 읽을 수 있습니다.

HTML 투명 태그 변환

페이지 프로듀서는 사용자가 *Content* 메소드 중 하나를 호출할 때 HTML 템플릿을 변환합니다. *Content* 메소드는 HTML 투명 태그를 만나면 *OnHTMLTag* 이벤트를 트리거합니다. 확인되는 태그 유형을 결정하고 이를 사용자 지정 컨텐트로 바꾸기 위해 이벤트 핸들러를 작성해야 합니다.

페이지 프로듀서에 *OnHTMLTag* 이벤트 핸들러를 만들지 않은 경우 HTML 투명 태그는 빈 문자열로 바뀝니다.

액션 항목에서 페이지 프로듀서 사용

페이지 프로듀서 컴포넌트는 일반적으로 HTML 템플릿이 포함된 파일을 지정하기 위해 *HTMLFile* 속성을 사용합니다. *OnAction* 이벤트 핸들러는 다음과 같이 *Content* 메소드를 호출하여 템플릿을 최종적인 HTML로 변환합니다.

```
procedure WebModule1.MyActionEventHandler(Sender: TObject; Request: TWebRequest;
   Response: TWebResponse; var Handled: Boolean);
begin
  PageProducer1.HTMLFile := 'Greeting.html';
  Response.Content := PageProducer1.Content;
end;
```

Greeting.html은 다음과 같은 HTML 템플릿이 포함된 파일입니다.

```
<HTML>
<HEAD><TITLE>Our Brand New Web Site</TITLE></HEAD>
<BODY>
Hello <#UserName>!  Welcome to our Web site.
</BODY>
</HTML>
```

OnHTMLTag 이벤트 핸들러는 다음과 같은 방법으로 실행 도중 HTML에 있는 사용자 지정 태그(<#UserName>)를 바꿉니다.

```
procedure WebModule1.PageProducer1HTMLTag(Sender : TObject;Tag: TTag;
   const TagString: string; TagParams: TStrings; var ReplaceText: string);
begin
  if CompareText(TagString,'UserName') = 0 then
    ReplaceText := TPageProducer(Sender).Dispatcher.Request.Content;
```

```
end;
```

요청 메시지의 컨텐트가 문자열 *Mr. Ed*일 경우 *Response.Content*의 값은 다음과 같습니다.

```
<HTML>
<HEAD><TITLE>Our Brand New Web Site</TITLE></HEAD>
<BODY>
Hello Mr. Ed!  Welcome to our Web site.
</BODY>
</HTML>
```

참고 이 예제는 *OnAction* 이벤트 핸들러를 사용하여 컨텐트 프로듀서를 호출하고 응답 메시지의 컨텐트를 할당합니다. 디자인 타임에 페이지 프로듀서의 *HTMLFile* 속성을 지정하는 경우 *OnAction* 이벤트 핸들러를 작성할 필요가 없습니다. 이 경우에는 간단히 *PageProducer1*을 액션 항목의 *Producer* 속성으로 지정하면 위의 *OnAction* 이벤트 핸들러와 동일한 효과를 거둘 수 있습니다.

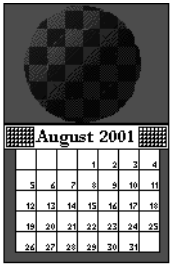
페이지 프로듀서 간의 연결

OnHTMLTag 이벤트 핸들러에서 가져온 대체 텍스트는 HTTP 응답 메시지에서 사용할 최종적인 HTML이 아니어도 됩니다. 특정 페이지 프로듀서의 출력이 다음 페이지의 입력이 되는 페이지 프로듀서를 여러 개 사용할 수도 있습니다.

페이지 프로듀서를 서로 연결하는 가장 간단한 방법은 각 페이지 프로듀서를 별도의 액션 항목에 연결하는 것이며, 이 때 모든 액션 항목은 동일한 *PathInfo*와 *MethodType*을 가집니다. 첫 번째 액션 항목은 컨텐트 프로듀서에 가져온 웹 응답 메시지를 설정하지만 이 액션 항목의 *OnAction* 이벤트 핸들러는 메시지를 처리되지 않은 것으로 간주합니다. 다음 액션 항목은 연결된 프로듀서의 *ContentFromString* 메소드를 사용하여 웹 응답 메시지 컨텐트를 조작하는 등의 작업을 수행합니다. 첫 번째 이후의 액션 항목은 다음과 같이 *OnAction* 이벤트 핸들러를 사용합니다.

```
procedure WebModule1.Action2Action(Sender: TObject; Request: TWebRequest;
   Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := PageProducer2.ContentFromString(Response.Content);
end;
```

예를 들어, 원하는 페이지의 월과 년도를 지정하는 요청 메시지에 대한 응답으로 달력 페이지를 반환하는 애플리케이션을 가정해 보십시오. 각 달력 페이지에는 우선 그림이 있고 전 달과 다음 달의 이미지 사이에 월 이름과 년도가 있으며 마지막으로 실제 달력이 나옵니다. 결과 이미지는 다음과 같이 나타납니다.



이 달력의 일반 폼은 템플릿 파일에 저장되며 다음과 같습니다.

```
<HTML>
<Head></HEAD>
<BODY>
<#MonthlyImage> <#TitleLine><#MainBody>
</BODY>
</HTML>
```

첫 번째 페이지 프로듀서의 *OnHTMLTag* 이벤트 핸들러는 요청 메시지에서 월과 년도를 조회합니다. 조회한 정보와 템플릿 파일을 사용하여 이벤트 핸들러가 다음과 같이 수행됩니다.

- <#MonthlyImage>를 <#Image Month=August Year=2001>로 바꿉니다.
- <#TitleLine>을 <#Calendar Month=July Year=2001 Size=Small> August 2001 <#Calendar Month=September Year=2001 Size=Small>로 바꿉니다.
- <#MainBody>를 <#Calendar Month=August Year=2001 Size=Large>로 바꿉니다.

다음 페이지 프로듀서의 *OnHTMLTag* 이벤트 핸들러는 첫 번째 페이지 프로듀서가 만든 컨텐트를 사용하고 <#Image Month=January Year=2000> 태그를 해당 HTML <IMG> 태그로 바꿉니다. 다른 페이지 프로듀서는 적절한 HTML 테이블을 사용하여 #Calendar 태그를 해석(resolve)합니다.

응답에서 데이터베이스 정보 사용

HTTP 요청 메시지에 대한 응답에는 데이터베이스에서 가져온 정보를 포함할 수 있습니다. Internet 팔레트 페이지에 있는 특수한 컨텐트 프로듀서는 HTML을 생성하여 데이터베이스의 레코드를 HTML 테이블에 나타낼 수 있습니다.

웹 모듈에 세션 추가

콘솔 CGI 애플리케이션 및 Apache DSO 애플리케이션은 HTTP 요청 메시지에 응답하여 시작됩니다. 이러한 유형의 애플리케이션에서 데이터베이스 작업을 수행할 경우 각각의 요청 메시지가 고유한 애플리케이션 인스턴스를 갖기 때문에 사용자는 기본 세션을 사용하여 데이터베이스 연결을 관리할 수 있습니다.

HTML로 데이터베이스 정보 표시

인터넷 팔레트 페이지의 특수한 컨텐트 프로듀서 컴포넌트는 데이터셋의 레코드에 기초하여 HTML 명령을 제공합니다. 다음과 같은 두 가지 유형의 data-aware 컨텐트 프로듀서가 있습니다.

- 데이터셋 필드 서식을 HTML 문서의 텍스트로 설정하는 데이터셋 페이지 프로듀서
- 데이터셋 레코드 서식을 HTML 테이블로 설정하는 테이블 프로듀서

데이터셋 페이지 프로듀서 사용

데이터셋 페이지 프로듀서는 다른 페이지 프로듀서 컴포넌트와 같은 방식으로 작동합니다. 즉 HTML 투명 태그를 포함하는 템플릿을 최종 HTML 형태로 변환합니다. 그러나 데이터셋 페이지 프로듀서는 데이터셋에 있는 필드 이름과 일치하는 태그 이름을 가진 태그를 해당 필드의 현재 값으로 변환하는 특수 기능을 포함합니다. 페이지 프로듀서의 일반적인 사용에 관한 자세한 내용은 7-13페이지의 "페이지 프로듀서 컴포넌트 사용"을 참조하십시오.

데이터셋 페이지 프로듀서를 사용하려면 *TDataSetPageProducer* 컴포넌트를 웹 모듈에 추가하고 *DataSet* 속성을 필드 값이 HTML 컨텐트에 표시되어야 하는 데이터셋으로 설정합니다. 데이터셋 페이지 프로듀서의 출력을 설명하는 HTML 템플릿을 만듭니다. 그리고 표시하려는 모든 필드 값에 대해 다음과 같은 형태의 태그를 HTML 템플릿에 포함시킵니다.

```
<#FieldName>
```

여기서 *FieldName*은 값을 표시해야 하는 데이터셋의 필드 이름을 지정합니다.

애플리케이션이 *Content*, *ContentFromString* 또는 *ContentFromStream* 메소드를 호출하면 데이터셋 페이지 프로듀서는 필드를 나타내는 태그를 현재 필드 값으로 대체합니다.

테이블 프로듀서 사용

인터넷 팔레트 페이지에는 HTML 테이블을 만들어 데이터셋 레코드를 나타내는 다음과 같은 두 가지 컴포넌트가 들어 있습니다.

- 데이터셋 필드 서식을 HTML 문서의 텍스트로 설정하는 데이터셋 테이블 프로듀서
- 요청 메시지가 제공한 매개변수를 설정한 후 쿼리를 실행하고 결과 데이터셋의 서식을 HTML 테이블로 설정하는 쿼리 테이블 프로듀서

두 가지 테이블 프로듀서 중 하나를 사용하면 테이블 색, 테두리, 구분자 두께 등의 속성을 지정하여 결과 HTML 테이블의 모양을 사용자 지정할 수 있습니다. 디자인 타임에 테이블 프로듀서의 속성을 설정하려면 테이블 프로듀서 컴포넌트를 더블 클릭하여 Response Editor 대화 상자를 표시합니다.

테이블 속성 지정

테이블 프로듀서는 *THTMLTableAttributes* 객체를 사용하여 데이터셋의 레코드를 표시하는 HTML 테이블의 시각적 외관을 기술합니다. *THTMLTableAttributes* 객체에는 테이블 너비, HTML 문서에서의 간격 및 배경 색, 테두리 두께, 셀 안쪽 여백, 셀 간격 등에 대한 속성이 포함됩니다. 이러한 모든 속성은 테이블 프로듀서가 만든 HTML <TABLE> 태그에서 옵션으로 전환됩니다.

디자인 타임에 Object Inspector로 이러한 속성을 지정합니다. Object Inspector에서 테이블 프로듀서 객체를 선택하고 *TableAttributes* 속성을 확장하여 *THTMLTableAttributes* 객체의 표시 속성에 액세스합니다.

또한 이러한 속성을 런타임에 프로그램에서 지정할 수도 있습니다.

행 속성 지정

데이터를 표시하는 테이블의 행에서 테이블 속성처럼 셀의 배경색 및 정렬을 지정할 수 있습니다. 이 때 사용되는 *RowAttributes* 속성은 *THTMLTableRowAttributes* 객체입니다.

디자인 타임에 Object Inspector에서 *RowAttributes* 속성을 확장하여 이러한 속성을 지정합니다. 또한 이러한 속성을 런타임에 프로그램에서 지정할 수도 있습니다.

이외에도 *MaxRows* 속성을 설정하여 HTML 테이블에 표시되는 행 수를 조정할 수 있습니다.

열 지정

디자인 타임에 테이블의 데이터셋을 알고 있다면 Columns 에디터를 사용하여 열의 필드 바인딩을 사용자 지정하고 속성을 표시할 수 있습니다. 이렇게 하려면 테이블 프로듀서 컴포넌트를 선택하고 마우스 오른쪽 버튼을 클릭한 다음 컨텍스트 메뉴에서 Columns 에디터를 선택합니다. 이 에디터로 테이블 열을 추가, 삭제 또는 재정렬할 수 있습니다. Columns 에디터에서 열을 선택한 후 Object Inspector에서 열마다 필드 바인딩을 설정하고 속성을 표시할 수 있습니다.

HTTP 요청 메시지에서 데이터셋의 이름을 가져오는 경우라면 디자인 타임에 Columns 에디터에서 필드를 바인딩할 수 없습니다. 그러나 적당한 *THTMLTableColumn* 객체를 설정하고 *Columns* 속성의 메소드를 사용하여 테이블에 열을 추가해서 런타임에 프로그램에서 열을 사용자 지정할 수 있습니다. *Columns* 속성을 설정하지 않을 경우, 테이블 프로듀서는 데이터셋 필드와 일치하는 열의 기본 집합을 만들고 특별한 표시 특성들을 지정하지 않습니다.

HTML 문서에 테이블 포함(embed)

테이블 프로듀서의 *Header* 및 *Footer* 속성을 사용하면 더 큰 문서의 데이터셋을 나타내는 HTML 테이블을 포함(embed)할 수 있습니다. *Header*는 테이블 앞에 오는 모든 내용을 지정하기 위해 *Footer*는 테이블 뒤에 오는 모든 내용을 지정하기 위해 사용합니다.

페이지 프로듀서와 같은 다른 컨텐트 프로듀서를 사용하여 *Header* 및 *Footer* 속성에 대한 값을 만들 수도 있습니다.

더 큰 문서의 테이블을 포함시킬 경우 테이블에 캡션을 추가할 수 있습니다. 테이블에 캡션을 추가하려면 *Caption* 및 *CaptionAlignment* 속성을 사용합니다.

데이터셋 테이블 프로듀서 설정

*TDataSetTableProducer*는 데이터셋의 HTML 테이블을 만드는 테이블 프로듀서입니다. 표시할 레코드를 포함하는 데이터셋을 지정하려면 *TDataSetTableProducer*의 *DataSet* 속성을 설정합니다. 기본 데이터베이스 애플리케이션에서의 대부분의 data-aware 객체와 마찬가지로 *DataSource* 속성은 설정하지 않습니다. 이 속성을 설정하지 않는 이유는 *TDataSetTableProducer*가 고유한 데이터 소스를 내부적으로 생성하기 때문입니다.

웹 애플리케이션이 항상 같은 데이터셋에서 가져온 레코드를 표시하는 경우 디자인 타임에 *DataSet*의 값을 설정할 수 있습니다. 데이터셋이 HTTP 요청 메시지에 있는 정보를 사용한다면 런타임에 *DataSet* 속성을 설정해야 합니다.

쿼리 테이블 프로듀서 설정

HTML 테이블을 만들어 HTTP 요청 메시지에서 매개변수를 가져온 쿼리의 결과를 표시할 수 있습니다. 이렇게 하려면 쿼리 매개변수를 *TSQLQueryTableProducer* 컴포넌트의 *Query* 속성으로 사용하는 *TSQLQuery* 객체를 지정합니다.

요청 메시지가 GET 요청인 경우 HTTP 요청 메시지의 대상으로 제공된 URL의 *Query* 필드에서 쿼리의 매개변수를 가져옵니다. 요청 메시지가 POST 요청인 경우에는 요청 메시지의 컨텐트에서 쿼리 매개변수를 가져옵니다.

*TSQLQueryTableProducer*의 *Content* 메소드를 호출하면 이 메소드는 요청 객체에서 찾은 매개변수를 사용하여 쿼리를 실행한 다음 HTML 테이블 서식을 설정하여 결과 데이터셋의 레코드를 표시합니다.

다른 테이블 프로듀서와 마찬가지로 사용자는 HTML 테이블의 표시 속성이나 열 바인딩을 사용자 지정하거나 더 큰 HTML 문서에 테이블을 포함시킬 수 있습니다.

8

WebSnap 사용

* 이 장은 영문 Kylix2 개발자 안내서의 26장입니다.

새 컴포넌트, 마법사 및 뷰가 추가되어 Web Broker의 기능을 확장하는 WebSnap을 사용하면 복잡한 데이터 방식의 웹 페이지가 포함된 웹 애플리케이션을 쉽게 만들 수 있습니다. WebSnap에서는 여러 가지 모듈과 서버측 스크립팅을 지원하며 Kylix 개발자 및 웹 디자이너 팀은 WebSnap으로 웹 애플리케이션을 쉽게 개발하거나 유지 관리할 수 있습니다.

WebSnap을 사용하여 팀의 HTML 디자인 담당자는 효과적으로 웹 서버를 개발하고 유지 관리할 수 있습니다. WebSnap으로 개발된 최종 제품에는 스크립트를 실행할 수 있는 일련의 HTML 페이지 템플릿이 포함됩니다. HTML 페이지는 Microsoft FrontPage나 일반 텍스트 에디터와 같이 포함된(embeded) 스크립트 태그를 지원하는 HTML 에디터를 사용하여 변경할 수 있습니다. 애플리케이션을 배포한 후에도 필요에 따라 템플릿을 변경할 수 있습니다. 프로젝트 소스 코드를 수정할 필요가 없으므로 귀중한 개발 시간을 절약할 수 있고 WebSnap의 여러 가지 모듈 지원 기능을 사용하면 프로젝트 코딩 단계에서 애플리케이션을 작은 부분으로 나눌 수 있으므로 Kylix 개발자는 보다 자유롭고 독립적으로 작업할 수 있습니다.

디스패처 컴포넌트는 페이지 컨텐트 요청, HTML 폼 전송, 동적 이미지 요청을 자동으로 처리합니다. 어댑터라는 새로운 컴포넌트는 애플리케이션의 비즈니스 룰에 대해 스크립트를 실행할 수 있는 인터페이스를 정의하는 수단을 제공합니다. 예를 들어 *TDataSetAdapter* 객체를 사용하면 데이터셋 컴포넌트를 스크립트를 실행할 수 있는 컴포넌트로 만듭니다. 새로운 프로듀서 컴포넌트를 사용하면 복잡한 데이터 방식 폼과 테이블을 쉽게 만들 수 있고 XSL을 사용하여 페이지를 생성할 수 있습니다. 세션 컴포넌트를 사용하면 최종 사용자를 추적할 수 있습니다. 사용자 목록 컴포넌트를 사용하면 사용자 이름, 암호 및 액세스 권한에 사용할 수 있습니다.

웹 애플리케이션 마법사를 사용하면 필요한 컴포넌트가 포함된 사용자 지정된 애플리케이션을 신속하게 구축할 수 있습니다. 웹 페이지 모듈 마법사를 사용하면 새 페이지를 정의하는 모듈을 애플리케이션에 만들 수 있습니다. 웹 데이터 모듈 마법사를 사용하면 웹 애플리케이션에서 공유하는 컴포넌트의 컨테이너를 만들 수 있습니다.

페이지 모듈 뷰를 사용하면 애플리케이션을 실행하지 않아도 서버측 스크립트의 결과를 볼 수 있습니다. Preview 탭을 클릭하면 내장 브라우저에 페이지가 표시됩니다. HTML Result 탭은 생성된 HTML을 보여 줍니다. HTML Script 탭은 페이지에 HTML을 생성하는 서버측 스크립팅이 포함된 페이지를 보여 줍니다.

이 장의 다음 단원에서는 WebSnap 컴포넌트를 사용하여 웹 서버 애플리케이션을 만드는 방법에 대해서 설명합니다.

기본적인 WebSnap 컴포넌트

WebSnap 웹 서버 애플리케이션을 만들려면 우선 WebSnap 개발에 사용하는 기본적인 컴포넌트에 대해 이해하고 있어야 합니다. WebSnap 개발 시 사용하는 컴포넌트는 다음과 같습니다.

- 애플리케이션을 구성하고 페이지를 정의하는 컴포넌트가 포함된 웹 모듈
- HTML 페이지와 웹 서버 애플리케이션 사이에 인터페이스를 제공하는 어댑터
- 최종 사용자에게 제공할 HTML 페이지를 만드는 루틴이 포함된 페이지 프로듀서

이제 각 컴포넌트 유형을 보다 자세히 살펴보겠습니다.

웹 모듈

웹 모듈은 WebSnap 애플리케이션에서 기본적으로 구축되는 블록입니다. 모든 WebSnap 서버 애플리케이션에는 하나 이상의 웹 모듈이 있어야 합니다. 필요하면 웹 모듈을 추가할 수 있습니다. 웹 모듈 유형에는 다음 네 가지가 있습니다.

- *TWebAppPageModule*
- *TWebAppDataModule*
- *TWebPageModule*
- *TWebDataModule*

웹 페이지 모듈(*TWebPageModule*)은 페이지에 컨텐트를 제공합니다. 웹 데이터 모듈(*TWebDataModule*)은 애플리케이션이 공유하는 컴포넌트의 컨테이너 역할을 하며 일반적인 Kylix 애플리케이션에서 데이터 모듈이 수행하는 것과 동일한 기능을 웹 데이터 모듈이 WebSnap 애플리케이션에서 수행합니다. 서버 애플리케이션에 여러 개의 웹 페이지나 데이터 모듈을 포함시킬 수 있습니다.

애플리케이션에 몇 개의 모듈이 있어야 하는지 궁금하실 것입니다. 모든 WebSnap 애플리케이션에는 웹 애플리케이션 모듈이 하나만 있어야 합니다. 웹 페이지 또는 데이터 모듈은 필요한 만큼 추가할 수 있습니다.

웹 페이지 모듈의 경우 일반적으로 페이지 스타일마다 모듈을 하나만 사용하는 것이 좋습니다. 기존의 페이지 포맷을 사용할 수 있는 페이지를 구현하는 경우 새 웹 페이지 모듈은 필요하지 않습니다. 기존 페이지 모듈을 수정하는 것만으로 충분합니다. 페이지가 기존 모듈과 상이할 경우 새 모듈을 만들어야 할 것입니다. 예를 들어 온라인 카탈로그 판매를 처리하는 서버를 한 대 구축한다고 가정합니다. 상품을 설명하는 페이지에 레이아웃이 동일한 기본 정보가 일괄적으로 포함되어 있는 경우에 같은 웹 페이지 모듈을 공

유할 수도 있습니다. 그러나 주문 폼의 포맷과 기능은 상품을 설명하는 페이지의 포맷이나 기능과는 다르기 때문에 주문 폼에는 다른 웹 페이지 모듈이 필요할 것입니다.

웹 데이터 모듈의 경우 다른 룰이 적용됩니다. 웹 모듈 간에 공유할 수 있는 컴포넌트는 공유 액세스를 간소화할 수 있도록 웹 데이터 모듈에 두어야 합니다. 여러 가지 웹 애플리케이션에서 사용하는 컴포넌트를 웹 애플리케이션 각각의 웹 데이터 모듈에 둘 수도 있습니다. 이렇게 하면 애플리케이션 간에 항목을 쉽게 공유할 수 있습니다. 물론 이 두 가지에 해당하지 않는 경우에는 웹 데이터 모듈을 사용하지 않을 수도 있습니다. 경험에 비추어 웹 데이터 모듈이 필요하다고 판단될 경우 일반 데이터 모듈과 동일한 방법으로 사용하십시오.

웹 애플리케이션 모듈 유형

웹 애플리케이션 모듈은 웹 애플리케이션의 비즈니스 룰과 넌비주얼 컴포넌트를 중앙에서 제어하도록 해줍니다. 웹 애플리케이션 모듈에는 다음 두 가지 유형이 있습니다.

- 페이지 모듈: 이 모듈 유형을 선택하면 컨텐트 페이지가 만들어집니다. 페이지 모듈에는 페이지의 컨텐트를 생성하는 페이지 프로듀서가 있습니다. 페이지 프로듀서는 HTTP 요청 경로 정보가 페이지 이름과 같을 때 연결된 페이지를 표시합니다. 연결된 페이지는 경로 정보가 비어 있을 때 기본 페이지로 사용할 수 있습니다.
- 데이터 모듈: 이 모듈 유형을 선택하면 컨텐트 페이지가 만들어지지 않습니다. 이 모듈은 두 개의 웹 페이지 모듈에서 사용하는 데이터베이스 컴포넌트와 같이 다른 모듈들이 공유하는 컴포넌트의 컨테이너로 사용됩니다.

웹 애플리케이션 모듈은 요청 디스패치, 세션 관리, 사용자 목록 유지 관리와 같은 애플리케이션에 대한 전반적인 기능을 수행하는 컴포넌트의 컨테이너로 동작합니다. 웹 브로커 아키텍처를 잘 알고 있으면 웹 애플리케이션 모듈은 *TWebApplication* 객체와 유사하다고 생각해도 됩니다. 웹 애플리케이션 모듈에는 웹 애플리케이션 모듈 유형에 따라 일반적인 웹 페이지 모듈이나 웹 데이터 모듈 기능이 있는데 프로젝트는 한 가지 웹 애플리케이션만 포함할 수 있습니다. 일반 웹 모듈을 서버에 추가하면 원하는 모든 기능이 제공되므로 웹 애플리케이션이 하나만 있으면 됩니다.

웹 애플리케이션 모듈을 사용하여 가장 기본적인 서버 애플리케이션 기능을 포함합니다. 서버에서 특정 종류의 홈 페이지를 유지 관리할 예정이면 웹 애플리케이션 모듈에 *TWebAppDataModule* 대신 *TWebAppPageModule*을 사용하여 페이지에 추가 웹 페이지 모듈을 만들 필요가 없도록 할 수 있습니다.

웹 페이지 모듈

각각의 웹 페이지 모듈마다 페이지 프로듀서가 연결되어 있습니다. 요청이 수신되면 페이지 디스패처는 요청을 분석하고 해당 페이지 모듈을 호출하여 요청을 처리하고 페이지의 컨텐트를 반환합니다.

웹 데이터 모듈과 마찬가지로 웹 페이지 모듈은 컴포넌트의 컨테이너입니다. 웹 데이터 모듈과 웹 페이지 모듈 간의 차이점은 웹 페이지 모듈은 웹 페이지를 만드는 데 사용한다는 점입니다.

모든 웹 페이지 모듈에는 Preview라고 하는 에디터 뷰가 있으며 이 에디터 뷰로 구축 중인 페이지를 미리 볼 수 있습니다. Kylix가 제공하는 비주얼 애플리케이션 개발 환경을 최대한 활용하십시오.

페이지 프로듀서 컴포넌트

웹 페이지 모듈은 페이지의 컨텐트를 생성하는 데 사용되는 페이지 프로듀서 컴포넌트를 식별하는 속성을 갖고 있습니다 (페이지 프로듀서에 대한 자세한 내용은 8-6페이지의 "페이지 프로듀서"를 참조하십시오). WebSnap 마법사는 웹 페이지 모듈을 만들 때 자동으로 프로듀서를 추가합니다. 나중에 WebSnap 팔레트에서 다른 프로듀서를 가져다 놓음으로써 페이지 프로듀서 컴포넌트를 바꿀 수 있습니다. 하지만 페이지 모듈이 템플릿 파일을 가지고 있는 경우 이 파일의 컨텐트가 프로듀서 컴포넌트에 적합한지 확인해야 합니다.

페이지 이름

웹 페이지 모듈에는 HTTP 요청 또는 애플리케이션의 로직 내에서 페이지 참조 시 사용하는 페이지 이름이 있습니다. 웹 페이지 모듈의 유닛에 있는 팩토리가 웹 페이지 모듈의 페이지 이름을 지정합니다.

프로듀서 템플릿

대부분의 페이지 프로듀서는 템플릿을 사용합니다. HTML 템플릿에는 대개 투명 태그 (transparent tag) 또는 서버측 스크립트와 혼합된 정적인 HTML이 일부 들어 있습니다. 페이지 프로듀서는 컨텐트를 만들 때 투명 태그를 해당 값과 바꾸고 서버측 스크립트를 실행하여 클라이언트 브라우저에 의해 표시되는 HTML을 만들어냅니다. XSLPageProducer는 이와 다르게 HTML이 아닌 XSL이 포함된 XSL 템플릿을 사용합니다. XSL 템플릿은 투명 태그나 서버측 스크립트를 지원하지 않습니다.

웹 페이지 모듈에는 유닛의 일부로 관리되는 연결된 템플릿 파일이 있을 수도 있습니다. 관리되는 템플릿 파일은 프로젝트 관리자에 나타나고 유닛 서비스 파일과 동일한 기본 (base) 파일 이름 및 위치를 사용합니다. 웹 페이지 모듈에 연결된 템플릿 파일이 없는 경우 페이지 프로듀서 컴포넌트의 속성에서 템플릿을 지정합니다.

웹 데이터 모듈

표준 데이터 모듈처럼 웹 데이터 모듈은 팔레트에 있는 컴포넌트들의 컨테이너입니다. 데이터 모듈은 컴포넌트 추가, 제거, 선택에 대한 디자인 외관을 제공합니다. 웹 데이터 모듈은 유닛의 구조 및 웹 데이터 모듈이 구현하는 인터페이스에 있는 표준 데이터 모듈과 다릅니다.

웹 데이터 모듈을 애플리케이션에서 공유되는 컴포넌트의 컨테이너로 사용합니다. 예를 들면 데이터셋 컴포넌트를 데이터 모듈에 넣고 다음과 같은 모듈에서 데이터셋에 액세스할 수 있습니다.

- 그리드를 표시하는 페이지 모듈
- 입력 폼을 표시하는 페이지 모듈

웹 데이터 모듈을 사용하여 여러 가지 상이한 웹 서버 애플리케이션에서 사용할 수 있는 컴포넌트 집합을 포함할 수도 있습니다.

웹 데이터 모듈 유닛의 구조

표준 데이터 모듈에는 데이터 모듈 객체에 액세스할 때 사용하는 폼 변수라는 변수가 있습니다. 웹 데이터 모듈은 이 변수를 함수와 대체합니다. 용도는 동일합니다. 하지만

WebSnap 애플리케이션은 멀티스레드로 실행될 수 있고 여러 요청을 동시에 서비스하는 특정 모듈의 여러 인스턴스를 가질 수 있기 때문에 이 함수는 정확한 인스턴스를 반환하도록 구현됩니다.

이 유닛은 또한 팩토리를 등록합니다. 팩토리에서는 WebSnap 애플리케이션이 모듈을 관리하는 방법을 지정합니다. 예를 들면 플래그는 모듈을 캐시할 것인지, 다른 요청에 모듈을 재사용할 것인지, 요청이 서비스된 후 모듈을 소멸시킬 것인지를 나타냅니다.

어댑터

어댑터는 서버 애플리케이션에 대한 스크립트 인터페이스를 정의합니다. 이 인터페이스로 페이지에 스크립트 랭귀지를 삽입하고 스크립트 코드에서 어댑터를 호출하여 정보를 가져올 수 있습니다. 예를 들어 HTML 페이지에 표시할 데이터 필드를 정의하는 데 어댑터를 사용할 수 있습니다. 스크립트 HTML 페이지에는 해당 데이터 필드의 값을 가져오는 HTML 컨텐트와 스크립트 문이 포함되기도 합니다. Web Broker 애플리케이션에 사용되는 투명 태그도 이와 유사합니다. 어댑터는 명령을 실행하는 액션을 지원하기도 합니다. 예를 들어 하이퍼링크를 클릭하거나 HTML 폼을 전송하여 어댑터 액션을 사용할 수 있습니다.

어댑터는 HTML 페이지를 동적으로 생성하는 작업을 간소화한다는 점에서 유용합니다. 애플리케이션에 어댑터를 사용할 경우 조건부 논리문 및 순환문을 지원하는 객체 지향 스크립트를 이용할 수 있습니다. 어댑터나 서버측 스크립트가 없는 경우에는 Pascal 이벤트 핸들러에서 훨씬 많은 HTML 생성 로직을 작성해야 합니다. 어댑터를 사용하면 개발 시간을 현저하게 줄일 수 있습니다.

스크립팅에 대한 자세한 내용은 8-31페이지의 "WebSnap에서의 서버측 스크립팅"과 8-35페이지의 "WebSnap 서버측 스크립팅 참조"를 참조하십시오.

페이지 컨텐트 생성에 사용할 수 있는 어댑터 컴포넌트에는 필드, 액션, 오류, 레코드의 네 가지가 있습니다.

필드

필드는 페이지 프로듀서가 애플리케이션에서 데이터를 가져오고 웹 페이지에 그 컨텐트를 표시하기 위해 사용하는 컴포넌트입니다. 필드는 이미지를 가져오는 데에도 사용할 수 있습니다. 이 경우에 필드는 웹 페이지에 작성된 이미지의 주소를 반환합니다. 페이지가 컨텐트를 표시할 때 웹 애플리케이션에 전달된 요청이 어댑터 디스패처를 호출하여 필드 컴포넌트로부터 실제 이미지를 가져옵니다.

액션

액션은 어댑터를 대신하여 명령을 실행하는 컴포넌트입니다. 스크립트 언어는 페이지 프로듀서가 페이지를 생성할 때 어댑터 액션 컴포넌트를 호출하여 명령을 실행하는 데 필요한 매개변수와 함께 액션의 이름을 반환합니다. 예를 들어 HTML 폼 상의 버튼을 클릭하면 테이블의 행이 삭제되는 것을 생각해 봅시다. 여기서는 HTTP 요청에서 버튼과 연결된 액션 이름과 행 번호를 나타내는 매개변수가 반환됩니다. 어댑터 디스패처는 명명된 액션 컴포넌트를 찾고 매개변수로 행 번호를 액션에 전달합니다.

오류

어댑터는 액션 실행 중 발생한 오류의 목록을 가지고 있습니다. 페이지 프로듀서는 이 오류 목록에 액세스하고 애플리케이션이 최종 사용자에게 반환하는 웹 페이지에 오류를 표시합니다.

레코드

*TDataSetAdapter*와 같은 일부 어댑터 컴포넌트에는 레코드가 여러 개 나타납니다. 어댑터는 여러 레코드를 반복할 수 있는 스크립트 인터페이스를 제공합니다. 일부 어댑터는 페이징을 지원하고 현재 페이지의 행들만 반복합니다.

페이지 프로듀서

웹 페이지 모듈 대신 페이지 프로듀서를 사용하여 컨텐트를 생성할 수 있습니다. 프로듀서는 다음과 같은 기능을 제공합니다.

- HTML 컨텐트를 생성합니다.
- HTMLFile 속성을 사용하여 외부 파일을 참조하거나 HTMLDoc 속성을 사용하여 내부 문자열을 참조할 수 있습니다.
- 프로듀서가 웹 페이지 모듈과 함께 사용되는 경우 유닛에 연결된 파일이 템플릿이 될 수 있습니다.
- 프로듀서는 투명 태그나 스크립트를 사용하여 템플릿에 삽입할 수 있는 HTML을 동적으로 생성합니다. 투명 태그는 WebBroker 애플리케이션과 동일한 방식으로 사용할 수 있습니다. 투명 태그 사용에 대한 자세한 내용은 7-14페이지의 "HTML 투명 태그 변환"을 참조하십시오. 스크립팅 지원 기능을 사용하면 페이지 컨텐트를 채우기 위해 HTML 페이지에 서버측 JavaScript를 포함할 수 있습니다.

먼저 페이지 프로듀서를 사용하기 위한 표준 WebSnap 메소드에 대해 설명합니다. 웹 페이지 모듈을 만들 때 Web Page Module 마법사에서 페이지 프로듀서 유형을 선택해야 합니다. 선택할 수 있는 방법은 여러 가지 있으나 대부분의 WebSnap 개발자는 어댑터 페이지 모듈인 *TAdapterPageProducer*를 사용하여 페이지를 프로토타입으로 만들 것입니다. 어댑터 페이지 프로듀서로 표준 컴포넌트 모델과 유사한 프로세스를 사용하여 프로토타입 웹 페이지를 만들 수 있습니다. 어댑터 폼을 어댑터 페이지 프로듀서에 추가합니다. 필요에 따라 어댑터 그리드와 같은 컴포넌트를 어댑터 폼에 추가할 수 있습니다. 어댑터 페이지 프로듀서를 사용하여 사용자 인터페이스를 만드는 데 사용하는 표준 Kylix 방법과 유사한 방식으로 웹 페이지를 생성할 수 있습니다.

어댑터 페이지 프로듀서에서 일반적인 페이지 프로듀서로의 전환이 보장되는 환경이 있습니다. 예를 들어 어댑터 페이지 프로듀서의 일부 함수는 런타임 시 페이지 템플릿에 스크립트를 동적으로 생성합니다. 서버를 최적화하기 위해 정적인 스크립트를 삽입하려고 할 수도 있고 스크립트를 잘 아는 사용자라면 스크립트를 직접 변경하려고 할 수 있습니다. 이러한 경우 동적 스크립트와 정적 스크립트 간의 충돌을 피하기 위해 일반 페이지 프로듀서를 사용해야 합니다. 일반 페이지 프로듀서 변경 방법에 대한 내용은 8-23페이지의 "고급 HTML 디자인"을 참조하십시오.

프로듀서를 웹 디스패처 액션 항목에 연결하면 WebBroker 애플리케이션에서 사용하는 것과 동일한 방법으로 페이지 프로듀서를 사용할 수도 있습니다. 웹 페이지 모듈을 사용함으로써 얻는 이점은 다음과 같습니다.

- 애플리케이션을 실행하지 않고 페이지의 레이아웃을 미리 볼 수 있습니다.
- 페이지 이름을 모듈과 연결할 수 있으므로 페이지 디스패처가 자동으로 페이지 프로듀서를 호출할 수 있습니다.

WebSnap을 사용한 웹 서버 애플리케이션 생성

이제 WebSnap 아키텍처에 대해 잘 이해하셨을 것입니다. WebSnap용 Kylix 소스 코드를 보면 WebSnap에 수백 개의 객체가 있음을 알 수 있습니다. 실제로 WebSnap에는 객체와 기능이 많이 있으므로 WebSnap을 완전히 이해하려면 오랫동안 자세히 아키텍처를 공부해야 하며 이렇게 많은 WebSnap 기능을 모두 아는 것은 무리일 것입니다. 서버 애플리케이션을 개발하기에 전에 WebSnap 시스템에 대해 완벽하게 알고 있어야 할까요? 반드시 그럴 필요는 없습니다.

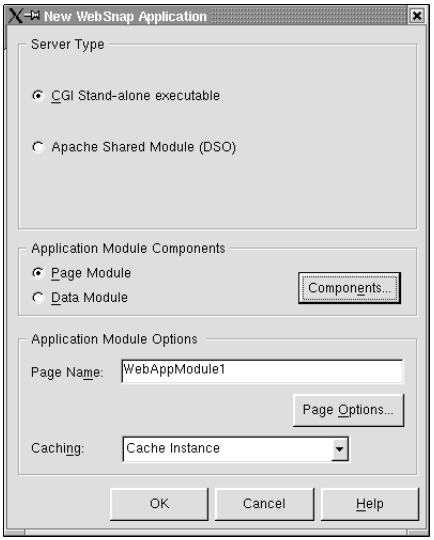
이전에도 이와 비슷한 경험을 하셨을 것입니다. 전반적으로 Kylix는 WebSnap보다 훨씬 복잡합니다. 일반 개발자 수준이라면 Kylix의 자세한 기능까지 이해할 수 없으며 이해할 필요성 또한 느끼지 못할 것입니다. 일반적으로 Kylix의 사용 방법을 배울 때는 현 프로젝트를 완수하는 데 필요한 부분을 배우는 데 중점을 두게 되고 필요한 부분은 나중에 배우게 됩니다. GUI가 없는 서버 애플리케이션을 개발하는 경우 사용자 지정할 수 있는 그래픽 사용자 인터페이스를 만드는 데 사용하는 ActionBands 툴을 배울 필요는 없습니다. 필요한 툴에 대해 개발 중에 배우면서 서버 애플리케이션을 구축하면 됩니다. 마찬가지로 GUI를 실제로 만들어야 하는 시기에 GUI에 대해 생각하면 됩니다.

일반 개발자 수준이라면 지금 필요한 것은 WebSnap 사용 방법에 대해 아는 것입니다. 다행히 WebSnap으로 작업할 준비가 되었으므로 실행하면서 배울 수 있습니다. 여기에서는 새 웹 서버 애플리케이션을 만들어 봄으로써 기본 WebSnap 아키텍처에 대해 배워 보겠습니다.

WebSnap 아키텍처를 사용하여 새로운 웹 서버 애플리케이션을 만드는 방법은 다음과 같습니다.

1. File|New를 선택합니다.
2. New Items 대화 상자에서 WebSnap 탭을 선택한 후 WebSnap Application을 선택합니다.
3. 그림 8.1처럼 다음과 같은 정보가 포함된 대화 상자가 나타납니다.
   - 서버 유형
   - 애플리케이션 모듈 컴포넌트
   - 애플리케이션 모듈 옵션

그림 8.1 새로운 WebSnap Application 대화 상자



서버 유형

다음 웹 서버 애플리케이션 유형 중 한 가지를 선택합니다.

- Apache: 이 애플리케이션 유형을 선택하면 프로젝트는 export된 메소드가 있는 DSO로 설정됩니다. export된 메소드는 Apache 웹 서버에 필요합니다. 프로젝트 파일에 라이브러리 헤더가 추가되고 사용 목록과 프로젝트 파일의 exports 절에 필수 항목이 추가됩니다.
- CGI 독립형: 이 애플리케이션 유형을 선택하면 프로젝트가 콘솔 애플리케이션으로 설정되고 프로젝트 파일의 uses 절에 필수 항목이 추가됩니다.

애플리케이션이 사용할 웹 서버 유형과 통신할 웹 서버 애플리케이션의 유형을 선택합니다.

애플리케이션이 생성된 후에 애플리케이션의 서버 유형을 변경할 수 있습니다. 다음 단계에 따라 서버 유형을 변경합니다.

1 IDE에서 프로젝트를 엽니다.

2 View|Project Manager를 선택하여 Project Manager를 엽니다. 프로젝트를 확장하면 유닛이 모두 나타납니다.

3 Project Manager에서 New 버튼을 클릭하여 새 웹 서버 애플리케이션 프로젝트를 만듭니다. WebSnap 탭에서 WebSnap Application 항목을 더블 클릭합니다. 사용하고자 하는 서버 유형을 포함하여 프로젝트에 적합한 옵션을 선택한 다음 OK를 클릭합니다.

4 Project Manager에서 새 프로젝트를 확장합니다. 프로젝트의 파일을 선택한 다음 삭제합니다.

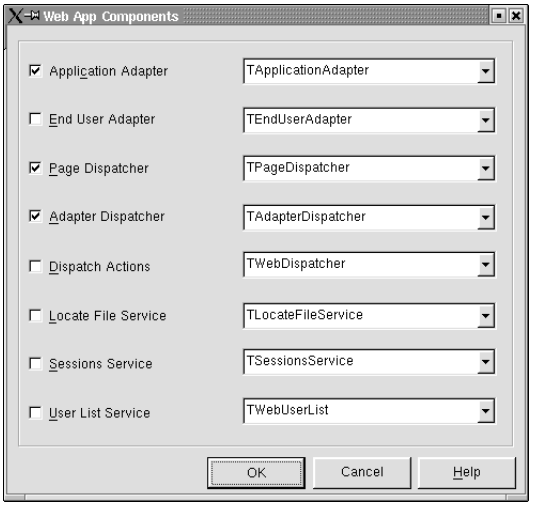
5 한 번에 하나씩 Web Application Debugger 프로젝트의 파일을 선택하여 새 프로젝트에 끌어다 놓습니다. 새 프로젝트에 파일을 추가할지 여부를 묻는 대화 상자가 나타나면 Yes를 클릭합니다.

애플리케이션 모듈 컴포넌트

애플리케이션 컴포넌트는 웹 애플리케이션의 기능을 제공합니다. 예를 들어 어댑터 디스패처 컴포넌트를 포함하면 HTML 폼 전송과 동적으로 생성된 이미지의 반환이 자동으로 처리됩니다. 페이지 디스패처를 포함하면 HTTP 요청 경로 정보에 페이지의 이름이 포함될 때 페이지의 컨텐트가 자동으로 표시됩니다.

새 WebSnap 애플리케이션 대화 상자에서 Components 버튼을 선택하면(그림 8.1 참조) 하나 이상의 웹 애플리케이션 모듈 컴포넌트를 선택할 수 있는 또 다른 대화 상자가 나타납니다. 그림 8.2와 같이 Web App Components라는 대화 상자가 표시됩니다.

그림 8.2 Web App Components 대화 상자



다음은 사용 가능한 컴포넌트를 간략히 설명한 것입니다.

- Application Adapter: 제목 등의 애플리케이션 정보를 포함하고 있습니다. 기본 타입은 *TApplicationAdapter*입니다.
- End User Adapter: 사용자 이름, 액세스 권한 및 로그인 여부 등의 사용자 정보를 포함하고 있습니다. 기본 타입은 *TEndUserAdapter*입니다.
  *TEndUserSessionAdapter*를 선택할 수도 있습니다.
- Page Dispatcher: HTTP 요청의 경로 정보를 검사하고 페이지의 컨텐트를 반환하는 해당 페이지 모듈을 호출합니다. 기본 타입은 *TPageDispatcher*입니다.
- Adapter Dispatcher: 어댑터 액션과 필드 컴포넌트를 호출하여 HTML 폼 전송 및 동적 이미지 요청을 자동으로 처리합니다. 기본 타입은 *TAdapterDispatcher*입니다.

- Dispatch Actions: 경로 정보와 메소드 타입에 따라 요청을 처리하는 액션 항목의 컬렉션을 정의할 수 있게 합니다. 액션 항목은 사용자가 정의한 이벤트를 호출하거나 페이지 프로듀서 컴포넌트의 컨텐트를 요청합니다. 기본 타입은 *TWebDispatcher* 입니다.
- Locate File Service: 웹 애플리케이션이 실행 중일 때 템플릿 파일 및 스크립트 포함 파일(include file)의 로드를 제어할 수 있습니다. 기본 타입은 *TLocateFileService* 입니다.
- Sessions Service: 단기간 동안 필요한 최종 사용자 정보를 저장하기 위해 사용됩니다. 예를 들면 세션을 사용하여 로그인한 사용자를 추적하고 비활성(inactivity) 기간이 지나면 자동으로 사용자를 로그아웃할 수 있습니다. 기본 타입은 *TSessionsService*입니다.
- User List Service: 승인된 사용자 및 사용자의 암호와 액세스 권한을 추적합니다. 기본 타입은 *TWebUserList*입니다.

위의 각 컴포넌트와 관련하여 대화 상자에 나열되어 있는 컴포넌트 타입은 Kylix에 포함되어 있는 기본 타입입니다. 사용자는 사용자 고유의 컴포넌트 타입을 만들거나 협력업체 컴포넌트 타입을 사용할 수 있습니다.

웹 애플리케이션 모듈 옵션

선택한 애플리케이션 모듈 유형이 페이지 모듈이면 대화 상자의 Page Name 필드에 이름을 입력하여 이름과 페이지를 연결할 수 있습니다. 런타임 시 이 모듈의 인스턴스는 캐시에 보관되거나 요청이 서비스될 때 메모리에서 제거될 수 있습니다. Caching 필드에 있는 옵션 중 하나를 선택합니다. Page Options 버튼을 통해 더 많은 페이지 모듈 옵션을 선택할 수 있습니다. 다음 범주를 설정할 수 있습니다.

- Producer: 페이지의 프로듀서 타입은 *AdapterPageProducer*, *DataSetPageProducer*, *InetXPageProducer*, *PageProducer*, *XSLPageProducer* 중 하나로 설정할 수 있습니다. 선택한 페이지 프로듀서가 스크립팅을 지원하면 Script Engine 드롭다운 목록을 사용하여 페이지를 스크립트할 때 사용할 언어를 선택하십시오.

참고 *AdapterPageProducer*는 JavaScript만 지원합니다.

- HTML: 선택한 프로듀서가 HTML 템플릿을 사용할 경우 이 그룹이 보입니다.
- XSL: 선택한 프로듀서가 *TXSLPageProducer*와 같은 XSL 템플릿을 사용하면 이 그룹이 보입니다.
- New File: 유닛의 일부로 작성되고 관리되는 템플릿 파일이 필요하면 New File을 선택합니다. 관리되는 템플릿 파일은 프로젝트 관리자에 나타나고 유닛 소스 파일과 파일 이름 및 위치가 동일합니다. 프로듀서 컴포넌트의 속성(일반적으로 *HTMLDoc* 또는 *HTMLFile* 속성)을 사용하려면 New File을 선택 해제합니다.
- Template: New File을 선택했으면 Template 드롭다운 목록에서 템플릿 파일의 기본 컨텐트를 선택합니다. "Default" 템플릿은 애플리케이션 제목, 페이지 제목, 게시된 페이지에 대한 하이퍼링크를 표시합니다.

- Page: 페이지 모듈의 페이지 이름과 제목을 입력합니다. 페이지 이름은 HTTP 요청 또는 애플리케이션의 로직 내에 있는 페이지를 참조할 때 사용되는 반면 제목은 페이지가 브라우저에 표시될 때 최종 사용자가 보게 되는 이름입니다.
- Published: 페이지 이름이 요청 메시지의 경로 정보와 일치하는 HTTP 요청에 페이지가 자동으로 응답하게 하려면 Published를 선택합니다.
- Login Required: 사용자가 로그온한 뒤에 페이지에 액세스할 수 있게 하려면 Login Required를 선택합니다.

이제 WebSnap 서버 애플리케이션의 기본 생성 방법에 대해 배웠습니다. 다음 단원에 나오는 WebSnap 자습서에서는 좀더 복잡한 애플리케이션 개발 과정에 대해 설명합니다.

WebSnap 자습서

다음 단원에서는 WebSnap 애플리케이션 구축 방법을 설명합니다. 자습서를 통해 새 디스패처와 어댑터 컴포넌트를 웹 페이지 모듈에 통합해 봄으로써 WebSnap 아키텍처와 새로운 개념에 친숙해질 것입니다. WebSnap 애플리케이션은 WebSnap HTML 컴포넌트를 사용하여 테이블의 컨텐트를 편집하는 애플리케이션을 만드는 방법을 보여줍니다.

새 애플리케이션 생성

이 자습서에서는 CountryTable 애플리케이션이라는 새 WebSnap 애플리케이션을 생성하는 방법을 설명합니다. CountryTable은 웹 상에 있는 사용자들의 다양한 국적에 관한 정보 테이블을 표시합니다. 사용자는 국가명을 추가/삭제할 수 있고 기존 국가에 관한 정보를 편집할 수 있습니다. 다음의 간단한 예제를 통해 WebSnap 애플리케이션 개발의 기초를 배울 수 있습니다.

1단계. WebSnap 애플리케이션 마법사 시작

1 Kylix 애플리케이션을 실행하고 File|New를 선택합니다.

2 New Items 대화 상자에서 WebSnap 탭을 선택한 후 WebSnap Application을 선택합니다.

3 New WebSnap Application 대화 상자에서 다음과 같이 합니다.

- CGI Stand-alone executable을 선택합니다.
- 컴포넌트 타입으로서 Page Module을 선택합니다.
- Page Name 필드에 **Home**을 입력합니다.

4 OK를 클릭합니다.

2단계. 생성된 파일과 프로젝트 저장

다음과 같은 방법으로 파스칼 유닛 파일과 프로젝트를 저장합니다.

1 File|Save All을 선택합니다.

2 File name 필드에 **HomeU.pas**를 입력하고 Save를 클릭합니다.

3 File name 필드에 **CountryTutorial.dpr**을 입력하고 Save를 클릭합니다.

참고 애플리케이션은 HomeU.html 파일을 실행 파일과 동일한 디렉토리에서 찾으므로 유닛과 프로젝트를 실행 파일이 있는 디렉토리에 저장하십시오.

3단계. 애플리케이션 제목 지정

애플리케이션 제목은 최종 사용자에게 표시되는 이름입니다. 다음과 같은 방법으로 애플리케이션 제목을 지정합니다.

1 View|Project Manager를 선택합니다.

2 Project Manager 창에서 CountryTutorial을 확장하고 HomeU 항목을 더블 클릭합니다.

3 Object Inspector의 풀다운 목록에서 ApplicationAdapter를 선택합니다.

4 Properties 탭에서 ApplicationTitle 필드에 **Country Tutorial**을 입력합니다.

5 에디터 창에서 Preview 탭을 클릭합니다. 애플리케이션 제목이 페이지 이름인 Home과 함께 페이지 상단에 표시됩니다.

표시된 페이지는 매우 단순한 페이지입니다. HomeU.html 에디터 탭이나 외부 에디터를 사용하여 HTML을 편집할 수 있습니다. 페이지 템플릿 편집 방법에 대한 자세한 내용은 8-23페이지의 "고급 HTML 디자인" 단원을 참조하십시오.

CountryTable 페이지 생성

웹 페이지 모듈은 게시된 페이지를 정의하는 데 사용되며 데이터 컴포넌트의 컨테이너로도 사용됩니다. 웹 페이지를 최종 사용자에게 반환할 때마다 웹 페이지 모듈은 페이지 생성에 필요한 정보를 포함하고 사용하는 데이터 컴포넌트에서 필요한 정보를 추출합니다. 다음 단계에서는 WebSnap 애플리케이션에 새 모듈을 추가하는 방법을 설명합니다.

1단계. 새 웹 페이지 모듈 추가

다음과 같은 방법으로 새로운 모듈을 추가합니다.

1 File|New를 선택합니다.

2 New Items 대화 상자에서 WebSnap 탭을 선택한 후 WebSnap Page Module을 선택합니다.

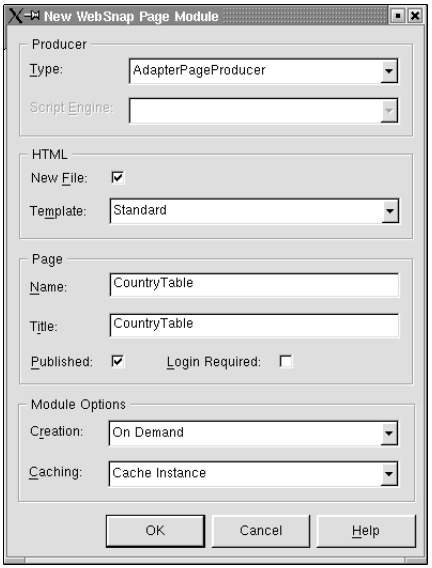
3 대화 상자에서 Producer Type을 목록에 있는 AdapterPageProducer로 설정합니다.

4 Page Name 필드에 **CountryTable**을 입력합니다.

5 나머지 필드와 선택 사항은 기본값으로 남겨 둡니다.

6 대화 상자가 그림 8.3과 같이 나타납니다. OK를 클릭합니다.

그림 8.3 CountryTable 페이지의 New WebSnap Page Module 대화 상자



CountryTable 모듈이 모양과 기능 면에서 폼과 유사한 창의 형태로 IDE에 나타납니다. 모듈을 저장한 다음 새 컴포넌트를 CountryTable 모듈에 추가합니다.

2단계. 새 웹 페이지 모듈 저장

유닛을 프로젝트 파일의 디렉토리에 저장합니다. 애플리케이션은 실행될 때 실행 파일이 있는 디렉토리에서 CountryTableU.html 파일을 찾습니다.

1 File|Save를 선택합니다.

2 File name 필드에 **CountryTableU.pas**를 입력하고 OK를 클릭합니다.

CountryTable 모듈에 데이터 컴포넌트 추가

TClientDataSet 컴포넌트는 HTML 테이블에 데이터를 제공합니다. *TDataSetAdapter* 컴포넌트를 통해 서버측 스크립트는 *TClientDataSet* 컴포넌트에 액세스할 수 있습니다. 여기에서는 data-aware 컴포넌트를 애플리케이션에 추가할 것입니다.

Kylix 데이터베이스 프로그래밍이 생소할 경우에는 아래의 1단계와 2단계를 이해할 수 없을 것입니다. 이 단계가 어려우면 자습서를 마치기 위해 단계를 계속할 필요는 없습니다. WebSnap에는 추가 데이터베이스 기능이 포함되어 있지 않습니다. WebSnap은 (어댑터 컴포넌트를 통해) 데이터베이스 컴포넌트의 인터페이스 역할만 수행합니다. 데이터베이스 프로그래밍에 대한 자세한 내용은 Developing Database Applications 도움말 파일이나 *Kylix 개발자 안내서*의 2부에서 해당 내용을 다룬 장을 참조하십시오. 그러나 우선은 자습서를 마치고 데이터베이스의 작동 방법에 대해서는 나중에 생각합니다.

1단계. data-aware 컴포넌트 추가

1 View|Project Manager를 선택합니다.

2 Project Manager 창에서 CountryTutorial을 확장하고 CountryTableU 항목을 더블 클릭합니다.

3 컴포넌트 팔레트에서 Data Access 탭을 선택합니다.

4 ClientDataSet 컴포넌트를 선택하고 CountryTable 웹 모듈에 추가합니다.

5 웹 페이지 모듈 창에서 ClientDataSet 컴포넌트를 선택합니다. 이 컴포넌트는 Object Inspector에서 ClientDataSet 컴포넌트 값을 표시합니다.

6 Object Inspector에서 다음 속성을 변경합니다.

- *Name* 속성에 **Country**를 입력합니다.
- *FileName* 속성을 Kylix가 설치된 demos/db/data 디렉토리에서 country.xml로 설정합니다.
- *Active* 속성을 *True*로 설정합니다.

2단계. 키 필드 지정

키 필드는 클라이언트 데이터셋의 레코드를 식별하는 데 사용됩니다. 이 필드는 애플리케이션에 edit 페이지를 추가할 때 중요합니다. 다음과 같은 방법으로 키 필드를 지정합니다.

1 CountryTable 모듈의 Country 컴포넌트를 더블 클릭하여 Fields Editor를 표시합니다.

2 에디터 창을 마우스 오른쪽 버튼으로 클릭하고 Add All Fields 명령을 선택합니다.

3 추가된 필드 목록에서 Name 필드를 선택합니다.

4 Object Inspector에서 *ProviderFlags* 속성을 확장합니다.

5 pfInKey 속성 값을 *True*로 설정합니다.

3단계. 어댑터 컴포넌트 추가

이제 필요한 데이터베이스 컴포넌트를 추가했으므로 WebSnap 프로그래밍으로 되돌아갑니다. 서버측 스크립팅을 통해 *TClientDataSet*의 데이터를 노출하려면 데이터셋 어댑터(*TDataSetAdapter*) 컴포넌트를 포함하고 있어야 합니다. 다음과 같은 방법으로 해당 컴포넌트를 추가합니다.

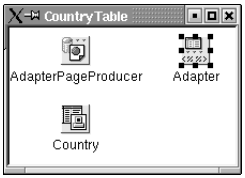
1 컴포넌트 팔레트에서 WebSnap 탭을 선택합니다.

2 DataSetAdapter 컴포넌트를 선택하고 CountryTable 웹 모듈에 추가합니다.

3 새로 만든 DataSetAdapter 컴포넌트를 클릭합니다. Object Inspector에서 다음 속성을 변경합니다.

- DataSet 필드를 Country로 설정합니다.
- Name 필드 유형에 **Adapter**를 입력합니다.

그림 8.4 CountryTable 웹 페이지 모듈



위의 과정을 마치면 CountryTable 웹 페이지 모듈이 그림 8.4처럼 보일 것입니다. 모듈의 요소는 보이지 않으므로 이 요소가 어디에 나타나는지는 중요하지 않습니다. 중요한 것은 모듈이 그림에 표시된 것과 같은 컴포넌트를 모두 포함해야 한다는 것입니다.

데이터를 표시할 그리드 생성

1단계. 그리드 추가

다음과 같은 방법으로 country 테이블 데이터베이스에서 데이터를 표시하는 그리드를 추가합니다.

1 View|Project Manager를 선택합니다.

2 Project Manager 창에서 CountryTutorial을 확장하고 CountryTableU 항목을 더블 클릭합니다.

3 CountryTable 모듈의 *AdapterPageProducer* 컴포넌트를 더블 클릭합니다. 이 컴포넌트는 HTML 테이블을 신속하게 만드는 데 사용되는 서버측 스크립트를 생성합니다. AdapterPageProducer 컴포넌트를 더블 클릭하면 웹 페이지 에디터가 나타납니다.

4 웹 페이지 에디터에서 AdapterPageProducer 항목을 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다.

5 Add Web Component 창에서 AdapterForm을 선택한 다음 OK를 클릭합니다. AdapterForm1 컴포넌트가 웹 페이지 에디터에 나타납니다.

6 AdapterForm1을 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다.

7 Add Web Component 대화 상자에서 AdapterGrid를 선택한 다음 OK를 클릭합니다. AdapterGrid1 컴포넌트가 웹 페이지 에디터에 나타납니다.

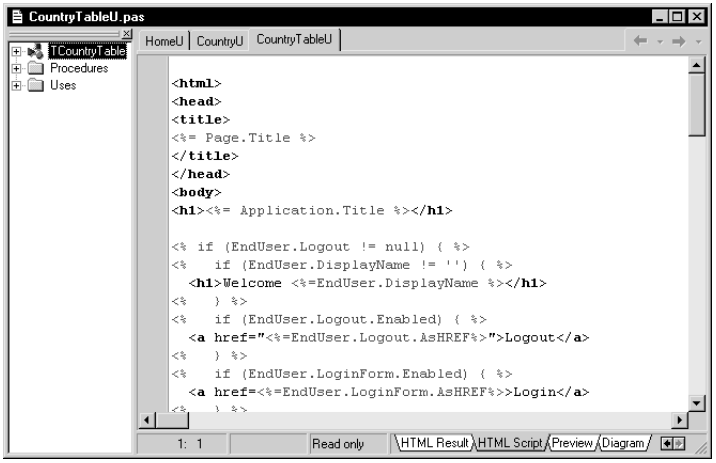
8 Object Inspector에서 Adapter 속성을 Adapter로 설정합니다.

그림 8.5 CountryTable Preview 탭

페이지를 미리 보려면 코드 에디터 창에서 CountryTableU.pas 탭을 선택하고 아래쪽에 있는 Preview 탭을 선택합니다. Preview 탭이 보이지 않으면 아래쪽에 있는 오른쪽 화살표를 사용하여 탭을 스크롤합니다. 미리 보기는 그림 8.5와 유사합니다.

Preview 탭은 웹 브라우저에서 정적인 HTML 페이지가 최종적으로 어떻게 나타나는지 보여 줍니다. 이 페이지는 스크립트를 포함하는 동적 HTML 페이지에서 생성됩니다. 스크립트 명령을 모두 표시한 상태에서 동적 페이지의 텍스트 표현을 보는 것이 도움이 될 때가 있습니다. 이것은 그림 8.6에 표시된 대로 에디터 창 아래쪽의 HTML Script 탭을 선택하여 실행할 수 있습니다.

그림 8.6 CountryTable HTML Script 탭



HTML Script 탭은 HTML과 스크립트를 조합하여 보여 줍니다. HTML과 스크립트는 에디터에서 글꼴 색상과 속성으로 구분됩니다. 기본적으로 HTML 태그는 검은색으로 굵게 쓰여진 텍스트로 나타나고 HTML 속성 이름은 검은색으로, HTML 속성 값은 파

란색으로 나타납니다. 스크립트 괄호 <% %>사이의 스크립트는 녹색으로 나타납니다. 에디터를 마우스 오른쪽 버튼으로 클릭하거나 Properties를 선택하면 Editor Properties 대화 상자가 나타납니다. 이 대화 상자의 Color 탭에 있는 항목의 기본 글꼴 색상과 속성은 변경할 수 있습니다.

이외에도 HTML과 관련된 에디터 탭이 두 개 더 있습니다. HTML Result 탭은 원래의 (raw) HTML 미리 보기 컨텐트를 보여 줍니다. HTML Result, HTML Script, Preview는 모두 읽기 전용임에 유의하십시오. 마지막의 HTML 관련 에디터 탭인 CountryTable.html은 편집 시 사용할 수 있습니다.

페이지의 모습을 보기 좋게 만들려면 CountryTable.html 탭이나 외부 에디터를 사용하여 언제든지 HTML을 추가할 수 있습니다. 페이지 템플릿 편집 방법에 대한 자세한 내용은 8-23페이지의 "고급 HTML 디자인" 단원을 참조하십시오.

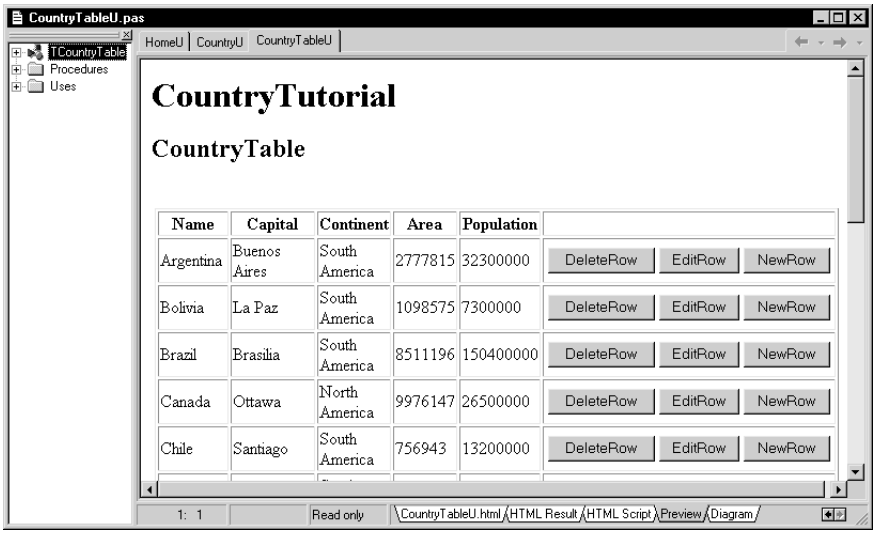
2단계. 그리드에 편집 명령 추가

사용자가 행을 삭제, 삽입, 편집하여 테이블의 컨텐트를 업데이트할 수도 있습니다. 사용자가 업데이트를 할 수 있도록 하려면 명령 컴포넌트를 추가합니다.

다음과 같은 방법으로 명령 컴포넌트를 추가합니다.

1. AdapterPageProducer 컴포넌트의 웹 페이지 에디터에서 *AdapterPageProducer* 컴포넌트와 모든 분기를 확장합니다.
2. AdapterGrid1 컴포넌트를 클릭합니다. AdapterGrid1 컴포넌트를 마우스 오른쪽 버튼으로 클릭하고 Add All Columns를 선택합니다.
3. AdapterGrid1 컴포넌트를 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다.
4. AdapterCommandColumn을 선택하고 OK를 클릭합니다.
   AdapterCommandColumn1 항목이 AdapterGrid1 컴포넌트에 추가됩니다.
5. AdapterCommandColumn1을 마우스 오른쪽 버튼으로 클릭하고 Add Commands를 선택합니다.
6. DeleteRow, EditRow, NewRow 명령 중에서 하나 이상 선택한 다음 OK를 클릭합니다.
7. 페이지를 미리 보려면 코드 에디터의 아래쪽에서 Preview 탭을 클릭합니다.
   그림 8.7과 같이 테이블의 각 행 끝에 DeleteRow, EditRow, NewRow 버튼이 보입니다. 애플리케이션이 실행 중일 때 이 버튼을 누르면 연결된 액션이 수행됩니다.

그림 8.7 편집 명령을 추가한 후의 CountryTable 미리 보기



Edit 폼 추가

이제 country 테이블의 Edit 폼이 되는 웹 페이지 모듈을 만듭니다. 사용자는 Edit 폼을 통해 CountryTable 애플리케이션의 데이터를 변경할 수 있습니다. 좀더 구체적으로 설명하면 사용자가 EditRow나 NewRow 버튼을 누르면 Edit 폼이 나타납니다. 사용자가 Edit 폼을 완료하면 수정된 정보가 테이블에 나타납니다.

1단계. 새 웹 페이지 모듈 추가

다음과 같은 방법으로 새로운 웹 페이지 모듈을 추가합니다.

1 File|New를 선택합니다.

2 New Items 대화 상자에서 WebSnap 탭을 선택한 후 WebSnap Page Module을 선택합니다.

3 대화 상자에서 Producer Type을 목록에 있는 AdapterPageProducer로 설정합니다.

4 Page Name 필드에 **CountryForm**을 입력합니다.

5 Published 상자를 선택 해제하여 사이트에서 사용 가능한 페이지의 목록에 이 페이지가 나타나지 않도록 합니다. Edit 폼은 Edit 버튼을 사용하여 액세스하고 해당 컨텐트는 country 테이블의 어느 행이 수정되었는지에 따라 달라집니다.

6 나머지 필드와 선택 사항은 기본값으로 둡니다.

7 OK를 클릭합니다.

2단계. 새 모듈 저장

유닛을 프로젝트 파일로서 디렉토리에 저장합니다. 애플리케이션이 실행되면 실행 파일이 있는 디렉토리에서 CountryFormU.html 파일을 찾습니다.

1 File|Save를 선택합니다.

2 File name 필드에 **CountryFormU.pas**를 입력하고 OK를 클릭합니다.

3단계. CountryTableU 유닛 사용

CountryTableU 유닛을 **uses** 절에 추가하여 모듈이 Adapter 컴포넌트에 액세스할 수 있게 합니다.

1 File|Use Unit을 선택합니다.

2 목록에서 CountryTableU를 선택한 다음 OK를 클릭합니다.

3 File|Save를 선택합니다.

4단계. 입력 필드 추가

컴포넌트를 *AdapterPageProducer* 컴포넌트에 추가하여 데이터 입력 필드를 HTML 형태로 생성합니다.

다음과 같은 방법으로 입력 필드를 추가합니다.

1 View|Project Manager를 선택합니다.

2 Project Manager 창에서 CountryTutorial을 확장하고 CountryFormU 항목을 더블 클릭합니다.

3 CountryForm 모듈에서 *AdapterPageProducer* 컴포넌트를 더블 클릭하여 웹 페이지 에디터를 표시합니다.

4 웹 페이지 에디터에서 AdapterPageProducer를 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다.

5 AdapterForm을 선택한 다음 OK를 클릭합니다. AdapterForm1 항목이 웹 페이지 에디터에 나타납니다.

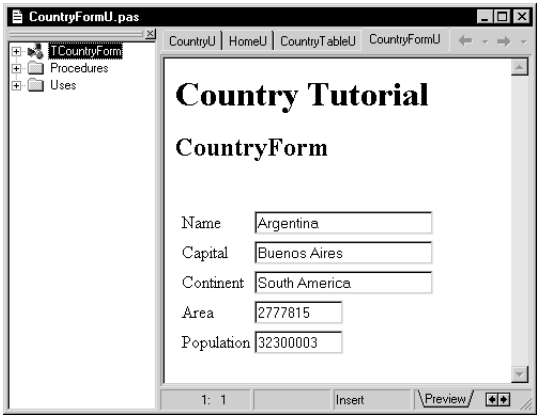
6 AdapterForm1을 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다.

7 AdapterFieldGroup을 선택한 다음 OK를 클릭합니다. AdapterFieldGroup1 항목이 웹 페이지 에디터에 나타납니다.

8 Object Inspector에서 Adapter 속성을 CountryTable.Adapter로 설정하고 AdapterMode 속성을 Edit로 설정합니다.

9 페이지를 미리 보려면 코드 에디터의 아래쪽에 있는 Preview 탭을 클릭합니다. 미리 보기가 그림 8.8과 유사하게 보일 것입니다.

그림 8.8 CountryForm 의 Preview 탭



5단계. 버튼 추가

컴포넌트를 *AdapterPageProducer* 컴포넌트에 추가하고 HTML 폼에 전송 버튼을 생성합니다. 다음과 같은 방법으로 컴포넌트를 추가합니다.

1 웹 페이지 에디터의 AdapterForm1을 클릭한 다음 마우스 오른쪽 버튼을 클릭하고 New Component를 선택합니다.

2 AdapterCommandGroup을 선택한 다음 OK를 클릭합니다.
AdapterCommandGroup1 항목이 웹 페이지 에디터에 나타납니다.

3 Object Inspector 창에서 DisplayComponent 속성을 AdapterFieldGroup1로 설정합니다.

4 AdapterCommandGroup1을 클릭한 다음 마우스 오른쪽 버튼을 클릭하여 Add Commands를 선택합니다.

5 Cancel, Apply, Refresh Row 명령 중에서 하나 이상 선택한 다음 OK를 클릭합니다.

6 페이지를 미리 보려면 코드 에디터 창의 아래쪽에 있는 Preview 탭을 클릭합니다. 미리 보기에 country 폼이 보이지 않으면 Code 탭을 클릭한 다음 Preview 탭을 다시 클릭합니다. 미리 보기가 그림 8.9와 유사하게 보일 것입니다.

그림 8.9 전송 폼이 있는 CountryForm

6단계. 그리드 페이지에 폼 액션 연결

사용자가 버튼을 클릭하면 해당 액션을 수행하는 어댑터 액션이 실행됩니다. 다음과 같은 방법으로 어댑터 액션이 실행된 후에 표시할 페이지를 지정합니다.

1 AdapterCommandGroup1을 클릭하여 웹 페이지 에디터 창의 오른쪽 상단 창에 CmdCancel, CmdApply, CmdRefreshRow 항목을 표시합니다.

2 CmdCancel을 선택합니다. Object Inspector에서 PageName 속성에 **CountryTable**을 입력합니다.

3 CmdApply를 선택합니다. Object Inspector에서 PageName 속성에 **CountryTable**을 입력합니다.

7단계. 폼 페이지에 그리드 액션 연결

다음과 같은 방법으로 그리드에서 버튼을 눌러 어댑터 액션이 실행된 후에 표시할 페이지를 지정합니다.

1 View|Project Manager를 선택합니다.

2 Project Manager 창에서 CountryTutorial을 확장하고 CountryTableU 항목을 더블 클릭합니다.

3 CountryTable 모듈에서 *AdapterPageProducer* 컴포넌트를 더블 클릭하여 웹 페이지 에디터를 표시합니다. AdapterCommandColumn1 항목을 클릭하여 웹 페이지 에디터의 오른쪽 상단 창에 CmdNewRow, CmdEditRow, CmdDeleteRow 항목을 표시합니다.

4 CmdNewRow를 선택합니다. Object Inspector에서 PageName 속성에 **CountryForm**을 입력합니다.

5 CmdEditRow를 선택합니다. Object Inspector에서 PageName 속성에 **CountryForm**을 입력합니다.

6 애플리케이션이 작동되는지, 모든 버튼의 액션이 수행되는지 확인하려면 애플리케이션을 실행하십시오. 애플리케이션을 실행하면 서버가 실행됩니다. 애플리케이션이 실행되는지 확인하려면 웹 브라우저에서 실행해 보아야 합니다. 이러한 방법에 대한 내용은 8-22페이지의 "완성된 애플리케이션 실행"을 참조하십시오.

참고 잘못된 타입과 같은 데이터베이스 오류는 표시되지 않습니다. 예를 들어 Area 필드에 국가명을 잘못된 값(예: 'abc')으로 추가해 보십시오.

완성된 애플리케이션 실행

다음 지침은 사용할 Apache 웹 서버, Kylix, 웹 브라우저가 모두 동일한 시스템에 있다는 것을 전제로 합니다. 다음과 같은 방법으로 완성된 애플리케이션을 실행합니다.

1 Project|Build CountryTutorial을 선택하여 CGI 애플리케이션을 구축합니다.

2 CountryTutorial, HomeU.html, CountryTableU.html, CountryFormU.html 파일을 Apache 설치 디렉토리의 cgi-bin 디렉토리에 복사합니다. 필요한 경우 HTML 파일에 대한 권한을 변경하여 모든 사용자가 이 파일들을 읽을 수 있도록 합니다. 실행 파일(CountryTable)에 대한 실행 권한을 모든 사용자에게 부여합니다.

3 Apache 서버가 실행 중이 아니면 다음 명령을 사용하여 서버를 시작합니다.

```
apachectl start
```

4 웹 브라우저로 다음 URL을 가리킵니다.

```
http://localhost/cgi-bin/CountryTutorial
```

오류 보고 추가

최종 사용자에게 오류를 보고하려면 AdapterErrorList 컴포넌트를 사용하여 country 테이블을 편집하는 어댑터 액션을 실행할 때 발생하는 오류를 표시합니다.

1단계. 그리드에 오류 지원 추가

1 CountryTable 모듈에서 *AdapterPageProducer* 컴포넌트를 더블 클릭하여 웹 페이지 에디터를 표시합니다.

2 AdapterForm1을 클릭한 후 마우스 오른쪽 버튼을 클릭하여 New Component를 선택합니다.

3 목록에서 AdapterErrorList를 선택한 다음 OK를 클릭합니다. AdapterErrorList1 항목이 웹 페이지 에디터의 오른쪽 상단 창에 나타납니다.

4 AdapterErrorList1을 클릭하여 AdapterGrid1 위로 옮긴 후 웹 페이지 에디터 툴바의 위쪽 화살표를 클릭합니다.

5 Object Inspector에서 Adapter 속성을 Adapter로 설정합니다.

2단계. 폼에 오류 지원 추가

1 CountryForm 모듈에서 *AdapterPageProducer* 컴포넌트를 더블 클릭하여 웹 페이지 에디터를 표시합니다.

2 AdapterForm1을 클릭한 후 마우스 오른쪽 버튼을 클릭하여 New Component를 선택합니다.

3 목록에서 AdapterErrorList를 선택한 다음 OK를 클릭합니다. AdapterErrorList1 항목이 웹 페이지 에디터의 오른쪽 상단 창에 나타납니다.

4 AdapterErrorList1을 클릭하여 AdapterFieldGroup1과 AdapterCommandGroup1 위로 이동시킨 다음 웹 페이지 에디터 툴바의 위쪽 화살표를 두 번 클릭합니다.

5 Object Inspector에서 Adapter 속성을 CountryTable.Adapter로 설정합니다.

3단계. 오류 보고 메커니즘 테스트

여기에 나오는 오류 보고 메커니즘을 설명하기 위해서는 먼저 Kylix IDE를 약간 변경해야 합니다. Tools|Debugger Options를 선택합니다. Language Exceptions 탭에서 Stop on Kylix Exceptions 체크 박스를 선택 해제함으로써 예외가 발생해도 애플리케이션이 계속 실행되도록 합니다. 또한 CGI 및 연결된 모든 HTML 파일을 재구축 및 재설치하십시오. 재구축 및 재설치 방법에 대한 내용은 8-22페이지의 "완성된 애플리케이션 실행"을 참조하십시오.

이제 다음과 같은 방법으로 그리드의 오류를 테스트합니다.

1 브라우저를 사용하여 CountryTable 페이지로 이동합니다.

2 브라우저의 다른 인스턴스를 시작하여 CountryTable 페이지로 이동합니다.

3 그리드의 첫 행에서 DeleteRow 버튼을 클릭합니다.

4 두 번째 브라우저를 새로 고치지 않고 그리드의 첫 행에서 DeleteRow 버튼을 클릭합니다.

Row not found in Country라는 오류 메시지가 그리드 위에 나타납니다.

다음과 같은 방법으로 폼의 오류를 테스트합니다.

1 CountryTable 페이지로 이동합니다.

2 여러 행 중 한 행의 EditRow 버튼을 클릭합니다. CountryForm 페이지가 표시됩니다.

3 Area 필드를 'abc'로 변경하고 Apply 버튼을 클릭합니다.

4 Invalid value for field 'Area'라는 오류 메시지가 첫 번째 필드 위에 표시됩니다.

이제 WebSnap 자습서를 마쳤습니다. 계속하기 전에 Stop on Kylix Exceptions 체크 박스를 다시 선택해도 됩니다.

고급 HTML 디자인

WebSnap을 사용하면 어댑터와 어댑터 페이지 프로듀서를 통해 웹 서버 애플리케이션에 스크립트된 HTML 페이지를 쉽게 만들 수 있습니다. WebSnap 툴을 사용하여 사용자에게 적합한 애플리케이션 데이터의 웹 프런트 엔드를 만들 수 있습니다. 다른 소스의 웹 디자인 전문 기술을 애플리케이션에 통합하는 기능은 WebSnap의 강력한 기능 중

하나입니다. 이 단원에서는 다른 툴과 프로그래머 이외의 팀원을 포함하도록 웹 서버 디자인 및 유지 관리 프로세스를 확장하는 방법을 다룹니다.

WebSnap 개발 단계의 최종 제품은 서버 애플리케이션 및 서버가 생성하는 페이지에 대한 HTML 템플릿입니다. 템플릿에는 스크립트와 HTML이 조합되어 있습니다. 일단 템플릿을 생성했으면 HTML 툴을 사용하여 언제라도 편집할 수 있습니다(에디터 때문에 스크립트가 변경되는 실수를 범하지 않으려면 포함된(embeded) 스크립트 태그를 지원하는 툴을 사용하는 것이 좋습니다). IDE 외부의 템플릿 페이지 편집 기능은 매우 유용한 기능입니다.

제품이 배포된 후에 최종 HTML 페이지 모습을 변경할 수도 있습니다. 소프트웨어 배포 팀은 최종 페이지의 레이아웃과는 상관이 없을 수도 있습니다. 페이지 레이아웃은 조직 내의 전용 웹 페이지 디자이너의 업무라고 할 수 있습니다. 페이지 디자이너가 Kylix를 개발한 경험이 없을 수도 있으며 실제로 개발 경험이 없더라도 상관없습니다. 디자이너들은 소스 코드를 수정하지 않아도 제품 개발과 유지 관리 주기 중에 페이지 템플릿을 편집할 수 있습니다. 다음은 편집 방법의 일례입니다.

개발 과정에서 Kylix 개발 팀은 어댑터 페이지 프로듀서나 페이지 프로듀서로 만든 프로토타입 페이지 템플릿을 사용하여 웹 서버 애플리케이션을 생성할 수 있습니다. 소프트웨어 개발 팀이 프로토타입 템플릿 페이지를 만든 후 이를 HTML 전문가에게 전달하면 MTML 전문가는 해당 템플릿 페이지를 최종 포맷에 삽입합니다. 이들은 JavaScript와 같은 랭귀지를 사용하여 컨텐트나 클라이언트측 스크립트를 페이지에 추가하거나 기타 필요한 편집 작업을 수행할 수 있습니다. 최종 HTML 편집이 끝나면 서버 애플리케이션을 호스트하는 웹 서버에 템플릿을 배포할 수 있습니다. 템플릿이 배포된 후 HTML 디자이너는 프로젝트 소스 코드를 수정하지 않아도 필요에 따라 페이지를 변경할 수 있습니다.

분명한 것은 이와 같은 방법이 유일한 서버 개발 프로세스는 아니지만 WebSnap HTML 템플릿으로 보다 효율적으로 서버를 개발하고 유지 관리하는 방법이라는 것입니다.

HTML 파일에서 서버측 스크립트 수정

개발 사이클 중에 페이지 템플릿의 HTML을 수정할 수 있습니다. 그러나 서버측 스크립팅은 별개의 문제입니다. Kylix 외부 템플릿의 서버측 스크립트는 언제라도 수정할 수 있지만 어댑터 페이지 프로듀서로 생성된 페이지는 수정하지 않는 것이 좋습니다. 어댑터 페이지 프로듀서는 런타임 시 페이지 템플릿의 서버측 스크립팅을 변경할 수 있다는 점에서 일반적인 페이지 프로듀서와 다릅니다. 다른 스크립트가 동적으로 추가될 때 스크립트가 어떻게 작동할지를 예측하기란 쉽지 않습니다. 스크립트를 직접 편집하려면 웹 페이지 모듈에 어댑터 페이지 프로듀서 대신 페이지 프로듀서가 있어야 합니다.

어댑터 페이지 프로듀서를 사용하는 웹 페이지 모듈을 가지고 있는 경우 다음 단계를 수행하여 어댑터 페이지 프로듀서를 변환하여 일반 페이지 프로듀서를 대신 사용할 수 있습니다.

1 변환하려는 모듈(이하 ModuleName)에서 HTML Script 탭의 모든 정보를 ModuleName.html 탭으로 복사하여 이전에 포함했던 모든 정보를 교체합니다.

2 컴포넌트 팔레트의 Internet 탭에 있는 페이지 프로듀서를 웹 페이지 모듈에 가져다 놓습니다.

3 교체할 어댑터 페이지 프로듀서의 해당 속성과 일치하도록 페이지 프로듀서의 ScriptEngine 속성을 설정합니다.

4 웹 페이지 모듈의 페이지 프로듀서를 어댑터 페이지 프로듀서에서 새 페이지 프로듀서로 변경합니다. Preview 탭을 클릭하여 페이지 컨텐트가 이전과 같은지 확인합니다.

5 어댑터 페이지 프로듀서 대신 새 페이지 프로듀서로 교체되었습니다. 이제 웹 페이지 모듈에서 어댑터 페이지 프로듀서를 삭제해도 좋습니다.

로그인 지원

웹 서버에는 여러 가지 이유로 로그인 지원이 필요합니다. 예를 들어 서버 애플리케이션은 일부 또는 전체 웹 사이트에 대한 사용자 액세스 권한을 부여하기 전에 사용자에게 로그인할 것을 요구할 수 있습니다. 사용자마다 표시되는 페이지의 모양이 다를 수 있으므로 웹 서버가 올바른 페이지를 전달하기 위해 로그인이 필요할 수도 있습니다. 또한 서버에는 메모리 및 프로세서 사이클에 물리적인 한계가 있으므로 주어진 시간 동안 사용자 수를 제한하는 기능이 때로는 필요합니다.

WebSnap을 사용하면 웹 서버 애플리케이션에 로그인 지원을 매우 간단하고 쉽게 통합할 수 있습니다. 이 단원에서는 초기 개발 단계에서 로그인 지원을 디자인하거나 기존 애플리케이션에 적절하게 수정하여 로그인 지원을 추가하는 방법에 대해 배우게 됩니다.

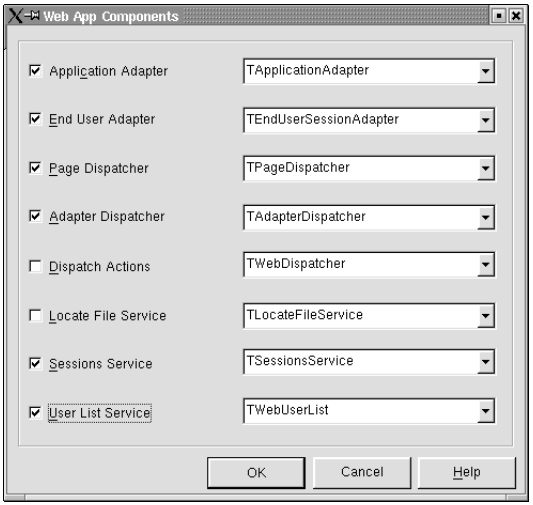
로그인 지원 추가

로그인 지원을 구현하려면 웹 애플리케이션 모듈에 다음 컴포넌트가 포함되어야 합니다.

- 사용자 이름, 암호, 서버 사용자 권한을 포함하는 사용자 목록 서비스(TWebUserList 타입의 객체)
- 서버에 현재 로그인한 사용자 관련 정보를 저장하는 세션 서비스(TSessionsService)
- 로그인과 연결된 액션을 처리하는 최종 사용자 어댑터(TEndUserSessionAdapter)

웹 서버 애플리케이션을 처음 만들 때 New WebSnap Application 대화 상자를 사용하여 이러한 컴포넌트를 추가할 수 있습니다. 대화 상자의 Components 버튼을 눌러 New Web App Components 대화 상자를 표시합니다. End User Adapter, Sessions Service, Web User List 상자를 선택합니다. End User Adapter 상자 옆의 드롭다운 메뉴에서 TEndUserSessionAdapter를 선택하여 최종 사용자 어댑터 유형을 선택합니다(기본값 TEndUserAdapter는 현재 로그인한 사용자를 추적할 수 없으므로 로그인 지원에 적합하지 않습니다). 이 과정을 마치면 대화 상자는 아래와 유사한 모습일 것입니다. OK를 두 번 클릭하여 대화 상자를 닫습니다. 이제 로그인 지원에 필요한 컴포넌트가 웹 애플리케이션 모듈에 생겼습니다.

그림 8.10 로그인 지원 옵션이 선택되어 있는 Web App Components 대화 상자



로그인 지원을 기존 웹 애플리케이션 모듈에 추가할 경우 컴포넌트 팔레트의 WebSnap 탭에서 모듈로 이러한 컴포넌트를 직접 가져다 놓을 수 있습니다. 웹 애플리케이션 모듈이 자동으로 구성됩니다.

디자인 단계에서 세션 서비스와 최종 사용자 어댑터에 신경 쓸 필요는 없지만 웹 사용자 목록(Web User List)에는 주의를 기울여야 합니다. WebUserList 컴포넌트 에디터를 통해 기본 사용자를 추가하고 이들의 읽기/수정 권한을 설정할 수 있습니다. 해당 컴포넌트를 더블 클릭하여 사용자 이름, 암호, 액세스 권한을 설정하는 에디터를 표시합니다. 액세스 권한 설정 방법에 대한 자세한 내용은 8-29페이지의 "사용자 액세스 권한"을 참조하십시오.

세션 서비스 사용

TSessionsService 타입의 객체인 세션 서비스는 웹 서버 애플리케이션에 로그인한 사용자를 추적합니다. 세션 서비스는 각 사용자에게 다른 세션을 할당하고 이름/값 쌍(예: 사용자 이름)을 사용자와 연결해 줍니다.

세션 서비스에 들어 있는 정보는 애플리케이션의 메모리에 저장되므로 웹 서버 애플리케이션은 세션 서비스가 작동하도록 요청 중간에도 계속 실행되어야 합니다. CGI와 같은 일부 서버 애플리케이션 유형은 요청 중간에 종료됩니다.

참고 애플리케이션이 로그인을 지원하도록 하려면 요청 중간에도 종료되지 않는 Apache DSO로 애플리케이션을 구축해야 합니다.

기본 서버 행동을 변경하는 데 사용할 수 있는 세션 서비스에는 두 가지 중요한 속성이 있습니다. MaxSessions 속성은 주어진 시간에 시스템에 로그인할 수 있는 최대 사용자 수를 지정합니다. MaxSessions의 기본값은 -1이며, 이 값으로 설정되면 소프트웨어에서 허용된 사용자 수에 제한을 두지 않습니다. 이로 인해 서버 하드웨어의 메모리나 프로세서 사이클이 계속 줄어들어 시스템 성능에 안 좋은 영향을 줄 수도 있습니다. 과

도한 사용자 수로 서버가 다운될 우려가 있으면 MaxSessions에 적절한 값을 설정하십시오.

DefaultTimeout 속성은 분 단위로 기본 시간 초과 기간을 지정합니다. 어떠한 사용자 작업도 없이 DefaultTimeout 시간이 경과하면 세션은 자동으로 종료됩니다. 사용자가 로그인했었다면 로그인 정보는 모두 없어집니다. 기본값은 20입니다. TimeoutMinutes 속성을 변경하여 주어진 세션 동안 기본값을 무시할 수 있습니다.

로그인 페이지

애플리케이션에는 로그인 페이지가 있어야 합니다. 사용자는 제한된 페이지에 액세스할 때나 액세스하기 전에 인증을 받기 위해 사용자 이름과 암호를 입력합니다. 사용자는 인증 과정을 거친 후 표시할 페이지를 지정할 수 있습니다. 사용자 이름과 암호가 웹 사용자 목록의 이름과 일치하면 사용자는 적절한 액세스 권한을 얻고 로그인 페이지에 지정된 페이지로 이동하게 됩니다. 사용자 인증이 확인되지 않은 경우 기본 액션으로 로그인 페이지를 다시 표시하거나 일부 다른 액션이 실행될 수도 있습니다.

WebSnap을 통해 웹 페이지 모듈이나 어댑터 페이지 프로듀서를 사용하여 간단한 로그인 페이지를 손쉽게 만들 수 있습니다. 로그인 페이지를 만들기 위해 새 웹 페이지 모듈을 생성합니다. File|New|Other를 선택하여 New Items 대화 상자를 표시한 다음 WebSnap 탭에서 WebSnap Page Module을 선택합니다. 페이지 프로듀서 유형으로서 AdapterPageProducer를 선택합니다. 원하는 옵션을 설정합니다. 로그인 페이지에는 Login이라는 이름을 주로 사용합니다.

이제 가장 기본적인 로그인 페이지 필드인 사용자 이름 필드, 암호 필드, 로그인 후 처음 표시되는 페이지를 선택할 수 있는 선택 상자, 페이지 전송 및 사용자 인증을 수행하는 Login 버튼을 추가합니다. 다음과 같은 방법으로 필드를 추가합니다.

1. (컴포넌트 팔레트의 WebSnap 탭에 있는) LoginFormAdapter 컴포넌트를 방금 만든 웹 페이지 모듈에 추가합니다.
2. AdapterPageProducer 컴포넌트를 더블 클릭하여 웹 페이지 에디터 창을 표시합니다.
3. 왼쪽 상단 창의 AdapterPageProducer를 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다. Add Web Component 대화 상자에서 AdapterForm을 선택하고 OK를 클릭합니다.
4. AdapterFieldGroup을 AdapterForm에 추가합니다.
   (왼쪽 상단 창의 AdapterForm을 마우스 오른쪽 버튼으로 클릭하고 New Component를 선택합니다. Add Web Component 대화 상자에서 AdapterFieldGroup을 선택하고 OK를 클릭합니다.)
5. 이제 Object Inspector로 가서 AdapterFieldGroup의 Adapter 속성을 LoginFormAdapter로 설정합니다. UserName, Password, NextPage 필드가 웹 페이지 에디터의 Browser 탭에 자동으로 나타납니다.

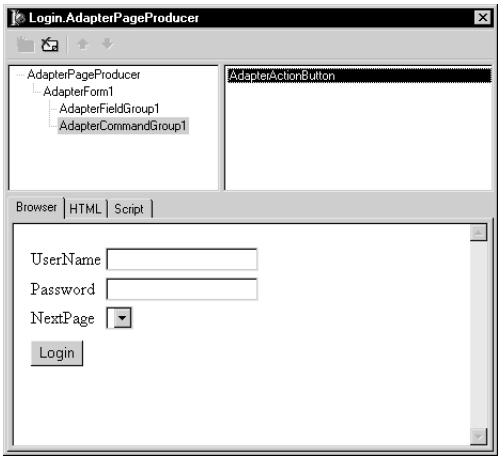
이를 통해 WebSnap은 일부 간단한 단계의 작업을 대부분 처리합니다. 아직까지 로그인 페이지에 인증을 위해 폼에 정보를 전송하는 Login 버튼이 없습니다.

다음과 같은 방법으로 Login 버튼을 추가합니다.

1. AdapterCommandGroup을 AdapterForm에 추가합니다.
2. AdapterActionButton을 AdapterCommandGroup에 추가합니다.
3. 웹 페이지 에디터의 오른쪽 상단 창에 나열되어 있는 AdapterActionButton을 클릭하고 Object Inspector를 사용하여 ActionName 속성을 Login으로 변경합니다. 웹 페이지 에디터의 Browser 탭에서 로그인 페이지의 미리 보기를 볼 수 있습니다.

웹 페이지 에디터가 아래의 그림과 같이 보입니다.

그림 8.11 웹 페이지 에디터에 표시된 로그인 페이지의 예



버튼이 AdapterFieldGroup 아래에 나타나지 않는 경우 AdapterCommandGroup이 웹 페이지 에디터의 AdapterFieldGroup 아래에 나열되도록 합니다. 버튼이 위에 나타나는 경우 AdapterCommandGroup을 선택하고 웹 페이지 에디터의 아래쪽 화살표를 클릭합니다(일반적으로 웹 페이지 요소는 웹 페이지 에디터에 나열된 순서에 따라 세로로 나타납니다).

로그인 페이지가 작동하려면 한 단계가 더 필요합니다. 최종 사용자 세션 어댑터에서 로그인 페이지가 될 페이지를 지정해야 합니다. 페이지를 지정하려면 웹 애플리케이션 모듈에서 EndUserSessionAdapter 컴포넌트를 선택합니다. Object Inspector에서 LoginPage 속성을 로그인 페이지의 이름으로 변경합니다. 로그인 페이지를 웹 서버 애플리케이션의 모든 페이지에 대해 사용할 수 있습니다.

로그인 요구 페이지 설정

일단 로그인 페이지가 사용되고 있는 경우 액세스 제어가 필요한 페이지에는 로그인을 해야 합니다. 로그인해야만 페이지에 액세스할 수 있도록 하는 가장 쉬운 방법은 페이지에 로그인 요구 구문을 삽입하여 디자인하는 것입니다. 처음 웹 페이지 모듈을 만들 때 New WebSnap Page Module 대화 상자의 Page 섹션에 있는 Login Required 상자를 선택합니다.

로그인을 요구하지 않는 페이지를 만든 경우 나중에 이를 변경할 수 있습니다. 다음과 같은 방법으로 웹 페이지 모듈이 생성된 후에 로그인 요구를 설정할 수 있습니다.

1 에디터에서 웹 페이지 모듈과 연결된 .pas 파일을 엽니다.

2 구현 섹션으로 스크롤다운합니다. WebRequestHandler.AddWebModuleFactory 명령의 매개변수에서 TWebPageInfo 객체의 생성자를 찾습니다. 생성자는 다음과 같습니다.

```
TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html')
```

3 중괄호를 제거하여 매개변수 목록의 wpLoginRequired 부분에 주석을 달지 않습니다. TWebPageInfo 생성자는 이제 다음과 같습니다.

```
TWebPageInfo.Create([wpPublished , wpLoginRequired], '.html')
```

페이지에서 로그인 요구를 제거하려면 위의 과정을 반대로 시행하고 생성자의 ", wpLoginRequired" 부분에 다시 주석을 답니다. 동일한 과정을 사용하여 페이지를 게시하거나 게시하지 않을 수 있습니다. "wpPublished ," 부분의 양 옆에 중괄호를 추가하거나 삭제하면 됩니다.

사용자 액세스 권한

사용자 액세스 권한은 웹 서버 애플리케이션에서 중요한 부분입니다. 서버가 제공하는 정보를 보고 수정하는 사용자를 통제할 수 있어야 합니다. 예를 들어 온라인 판매를 처리하는 서버 애플리케이션을 구축한다고 가정합니다. 사용자는 카탈로그의 상품은 볼 수 있으나 가격은 변경할 수 없도록 해야 합니다! 따라서 액세스 권한은 매우 중요하다고 볼 수 있습니다.

WebSnap은 페이지 및 서버 컨텐트에 대한 액세스를 제어할 수 있는 다양한 방법을 제공합니다. 앞의 단원에서 로그인을 요구함으로써 페이지 액세스를 제어하는 방법에 대해 배웠습니다. 그 밖에도 다른 옵션이 있습니다. 예를 들면 다음과 같습니다.

- 적당한 수정 액세스 권한을 갖고 있는 사용자에게는 편집 상자의 데이터 필드를 보여줄 수 있습니다. 그러나 다른 사용자는 필드 컨텐트를 볼 수는 있으나 편집은 할 수 없습니다.
- 보기 액세스 권한이 없는 사용자에게는 특정 필드를 숨길 수 있습니다.
- 인증되지 않은 사용자에게는 특정 페이지가 표시되지 않도록 할 수 있습니다.

이 단원에서는 이러한 행동을 구현하는 방법에 대해 설명합니다.

편집 상자 및 텍스트 상자로 필드를 동적으로 표시

어댑터 페이지 프로듀서를 사용할 경우 각각 다른 액세스 권한을 갖는 사용자에게 표시되는 페이지 요소의 외관을 변경할 수 있습니다. 예를 들어 Biolife 데모(Demos 디렉토리의 WebSnap 하위 디렉토리에 있음)에는 주어진 종(species)에 대한 모든 정보를 보여 주는 폼 페이지가 들어 있습니다. 이 폼은 사용자가 그리드의 Details 버튼을 클릭할 때 나타납니다. Will이라는 이름으로 로그인한 사용자는 일반 텍스트로 표시된 폼 데이터를 볼 수 있습니다. Will은 데이터를 수정할 수 있는 권한이 없으므로 폼은 데이터를 수정하는 메커니즘이 부여되지 않습니다. 사용자 Ellen은 수정 권한을 가지고 있으므로 폼 페이지를 볼 때 필드 컨텐트를 수정할 수 있는 일련의 편집 상자를 볼 수 있습니다. 이런 방식으로 액세스 권한을 사용하면 추가 페이지를 생성하는 번거로움을 덜 수 있습니다.

TAdapterDisplayField 및 TAdapterDisplayColumn과 같은 일부 페이지 요소의 외관은 ViewMode 속성에 의해 결정됩니다. ViewMode가 vmToggleOnAccess로 설정되면 수정 액세스 권한이 있는 사용자에게는 페이지 요소가 편집 상자로 나타납니다. 수정 액세스 권한이 없는 사용자는 일반 텍스트를 보게 됩니다. ViewMode 속성을 vmToggleOnAccess로 설정하여 페이지 요소의 모습과 기능이 동적으로 결정되도록 할 수 있습니다.

웹 사용자 목록(Web User List)은 시스템에 로그인할 수 있는 각 사용자가 하나씩 가지고 있는 TWebUserListItem 객체의 목록입니다. 사용자 권한은 웹 사용자 목록 항목의 AccessRights 속성에 저장됩니다. AccessRights는 텍스트 문자열이므로 원하는 방식으로 권한을 지정할 수 있습니다. 서버 애플리케이션에서 원하는 액세스 권한에 대한 이름을 만듭니다. 사용자가 액세스 권한을 여러 개 갖도록 하려면 공백이나 세미콜론, 쉼표로 목록의 항목을 분리합니다.

필드에 대한 액세스 권한은 ViewAccess 및 ModifyAccess 속성에 의해 제어됩니다. ViewAccess는 주어진 필드를 보는 데 필요한 액세스 권한의 이름을 저장합니다. ModifyAccess는 필드 데이터를 수정하는 데 필요한 액세스 권한을 지정합니다. 이 두 가지 속성은 각각의 필드와 필드를 포함하고 있는 어댑터 객체 두 곳에 나타납니다.

액세스 권한 검사 프로세스는 두 단계로 구성됩니다. 페이지의 필드 모양을 결정할 때 애플리케이션은 먼저 필드 자체의 액세스 권한을 검사합니다. 값이 빈 문자열이라면 애플리케이션은 필드를 포함하고 있는 어댑터의 액세스 권한을 검사합니다. 어댑터 속성도 비어 있는 경우에는 애플리케이션은 기본 행동(default behavior)을 따릅니다. 수정 액세스의 경우 기본 행동은 웹 사용자 목록에 있는 비어 있지 않은 AccessRights 속성을 갖는 사용자에 의한 수정을 허용합니다. 보기 액세스의 경우 보기 액세스 권한이 지정되어 있지 않을 경우 자동으로 권한이 부여됩니다.

필드와 컨텐트 숨기기

적절한 보기 권한을 가지고 있지 않은 사용자에게는 필드의 컨텐트를 숨길 수 있습니다. 먼저 필드의 ViewAccess 속성을 사용자가 소유해야 하는 권한과 일치하도록 설정합니다. 그런 다음 필드 페이지 요소의 ViewAccess를 vmToggleOnAccess로 설정합니다. 필드 캡션은 나타나지만 필드 값은 나타나지 않습니다.

물론 사용자에게 보기 권한이 없는 경우 필드에 대한 참조를 모두 숨기는 것이 때로는 가장 좋습니다. 이렇게 하려면 필드 페이지 요소의 HideOptions를 설정하여 hoHideOnNoDisplayAccess를 포함합니다. 그러면 필드의 컨텐트와 캡션이 모두 표시되지 않습니다.

페이지 액세스 차단

애플리케이션에서 인증되지 않은 사용자는 특정 페이지에 액세스하지 못하도록 할 수 있습니다. 페이지를 표시하기 전에 액세스 권한 검사를 부여하려면 웹 요청 핸들러의 AddWebModuleFactory 명령에서 TWebPageInfo 생성자로 호출을 변경합니다. 이 명령은 모듈 소스 코드의 초기화 섹션에 나타납니다.

TWebPageInfo의 생성자는 최대 6개의 인수를 사용합니다. WebSnap은 4개의 인수를 기본값(빈 문자열)으로 설정하므로 호출 구문은 다음과 유사할 것입니다.

```
TWebPageInfo.Create([wpPublished , wpLoginRequired], '.html')
```

액세스를 허가하기 전에 권한을 검사하려면 여섯 번째 매개변수에 필요한 권한에 대한 문자열을 제공해야 합니다. 예를 들어 권한을 "Access"로 가정합시다. 다음은 생성자를 수정하는 방법입니다.

```
TWebPageInfo.Create([wpPublished, wpLoginRequired], '.html', '', '', '', 'Access')
```

이제 "Access" 권한이 없는 사람의 페이지 액세스는 거부됩니다.

WebSnap에서의 서버측 스크립팅

페이지 프로듀서 템플릿에는 JavaScript와 같은 스크립트 언어가 포함될 수 있습니다. 페이지 프로듀서는 프로듀서의 컨텐트 요청에 응답하여 스크립트를 실행합니다. 웹 서버 애플리케이션이 스크립트를 평가하기 때문에 브라우저가 평가하는 클라이언트측 스크립트와 반대로 서버측 스크립트라고 합니다.

이 단원에서는 서버측 스크립팅의 개념적 개요 및 WebSnap 애플리케이션에서 서버측 스크립팅을 사용하는 방법을 제공합니다. 다음 단원의 "WebSnap 서버측 스크립팅 참조"에서는 스크립트의 객체, 속성, 메소드에 대한 보다 상세한 내용을 다룹니다. 서버측 스크립팅은 서버측 스크립트를 작성하는 API 참조로 생각할 수 있으며, 도움말 파일에서 볼 수 있는 Kylix 객체에 대한 설명과 유사합니다. 다음 단원에는 상세한 스크립트 예제가 포함되어 있으며 이 예제를 통해 HTML 페이지 생성에 스크립트가 사용되는 방법을 정확하게 알 수 있습니다.

서버측 스크립팅은 WebSnap에서 중요한 부분이지만 WebSnap 애플리케이션에서 반드시 사용되는 것은 아닙니다. 스크립팅은 HTML 생성에만 사용되며 스크립팅으로 애플리케이션 데이터를 HTML 페이지에 삽입할 수 있습니다. 실제로 어댑터와 기타 스크립트를 실행할 수 있는 객체에 의해 노출되는 속성은 거의 모두 읽기 전용 속성입니다. 서버측 스크립트는 파스칼로 작성된 컴포넌트와 이벤트 핸들러에 의해 관리되는 애플리케이션 데이터를 변경하는 데에는 사용되지 않습니다 .

이와 다른 방법으로 애플리케이션 데이터를 HTML 페이지에 삽입할 수 있습니다. 원할 경우 웹 브로커의 투명 태그나 다른 태그 방식의 솔루션을 사용할 수 있습니다. 예를 들면 WebSnap Demos 폴더에 설치되어 있는 일부 프로젝트는 스크립트 대신 XML과 XSL을 사용합니다. 그러나 스크립트를 사용하지 않으면 Pascal로 대부분의 HTML 생성 로직을 작성해야 하기 때문에 개발 시간이 많이 소요됩니다.

WebSnap에 사용되는 스크립팅은 객체 지향 개념을 적용하는 동시에 조건부 논리문, 순환문을 지원함으로써 페이지 생성 업무를 극도로 간소화합니다. 예를 들어 페이지는 일부 사용자만 편집할 수 있는 데이터 필드를 포함할 수도 있습니다. 스크립팅을 사용하면 조건부 논리문을 템플릿 페이지에 둘 수 있으므로 인증된 사용자에게는 편집 상자가, 그 밖의 사용자에게는 단순한 텍스트가 표시됩니다. 태그 방식의 접근법으로는 이와 같은 의사 결정을 HTML 생성 소스 코드로 프로그램해야 합니다.

스크립트 엔진

페이지 프로듀서의 *ScriptEngine* 속성은 템플릿 내의 스크립트를 평가하는 서버측 스크립팅 엔진을 식별합니다. Kylix는 JavaScript 엔진으로 SpiderMonkey(Mozilla 오픈 소스 브라우저 프로젝트의 일부)를 사용합니다. 다른 스크립트 엔진은 일반 페이지

프로듀서와 함께 사용되기도 하나 어댑터 페이지 프로듀서 객체는 JavaScript를 만들도록 디자인되었습니다.

Borland의 RAD 툴은 스크립트 엔진으로서 Microsoft의 JScript를 사용하고 있습니다. JScript와 JavaScript는 비슷한 랭귀지로 WebSnap 컴포넌트에 의해 만들어진 스크립트는 두 플랫폼 상에서 동일한 것으로 취급되므로 WebSnap 애플리케이션을 쉽게 포팅할 수 있습니다.

스크립트 블록

스크립트 블록은 <%와 %>로 구분됩니다. 스크립트 엔진은 스크립트 블록 내에 있는 텍스트를 평가합니다. 그 결과는 페이지 프로듀서 컨텐트의 일부가 됩니다. 페이지 프로듀서는 포함된(embeded) 투명 태그를 변환한 후 스크립트 블록 외부에 텍스트를 작성합니다. 스크립트 블록은 또한 조건부 논리 및 루프에서 텍스트의 출력을 지시할 수 있도록 텍스트를 괄호로 묶을 수 있습니다. 예를 들어 다음과 같은 JavaScript 블록은 5줄로 된 목록을 생성합니다.

```
<ul>
<% for (i=0;i<5;i++) { %>
     <li>Item <% Response.Write(i) %></li>
<% } %>
</ul>
```

다음의 스크립트 블록과 동일합니다.

```
<ul>
<% for (i=0;i<5;i++) { %>
     <li>Item <%=i %></li>
<% } %>
</ul>
```

<%= 구분자는 *Response.Write*의 단축형입니다.

스크립트 작성

개발자는 WebSnap 기능을 활용하여 자동으로 스크립트를 생성할 수 있습니다.

템플릿 마법사

새로운 WebSnap 애플리케이션이나 페이지 모듈을 작성할 때 WebSnap 마법사는 페이지 모듈 템플릿의 초기 컨텐트를 선택하는 데 사용되는 템플릿 필드를 제공합니다. 예를 들어 "Default"라는 템플릿은 애플리케이션 제목, 페이지 이름, 게시된 페이지에 대한 링크를 표시하는 JavaScript를 생성합니다.

TAdapterPageProducer

*TAdapterPageProducer*는 HTML과 JavaScript를 생성하여 폼과 테이블을 만듭니다. 생성된 JavaScript는 어댑터 객체를 호출하여 필드 값, 필드 이미지 매개변수, 액션 매개변수를 가져옵니다.

스크립트 편집과 보기

WebSnap 외관 디자이너는 스크립트된 페이지를 미리 볼 수 있도록 웹 페이지 모듈 뷰를 제공합니다. HTML Result 탭을 사용하면 실행된 스크립트에서 HTML 결과를 볼 수 있습니다. Preview 탭을 사용하면 브라우저에서 결과를 볼 수 있습니다. HTML Script 탭은 웹 페이지 모듈이 *TAdapterPageProducer*를 사용하는 경우에 사용할 수 있습니다. HTML Script 탭은 *TAdapterPageProducer* 객체에 의해 생성된 HTML과 JavaScript를 표시합니다. 어댑터 필드를 표시하고 어댑터 액션을 실행하는 HTML 폼을 만드는 스크립트를 작성하는 방법은 이 뷰를 참조하십시오.

페이지에 스크립트 포함

템플릿은 파일이나 다른 페이지의 스크립트를 포함할 수 있습니다. 파일에서 스크립트를 포함하려면 다음과 같은 코드를 사용합니다.

```
<!-- #include file="filename.html" -->
```

템플릿이 다른 페이지의 스크립트를 포함하고 스크립트가 포함하고 있는 페이지에 의해 평가되는 경우 다음과 같은 코드를 사용하여 page1의 평가되지 않은 컨텐트를 포함시킵니다.

```
<!-- #include page="page1" -- >
```

스크립트 객체

스크립트 객체는 스크립트로 참조할 수 있는 객체입니다. 활성 스크립트 엔진으로 객체에 대해 *IDispatch* 인터페이스를 등록하여 CLX 객체를 스크립트할 수 있게 만듭니다. 다음 객체들을 스크립트에 사용할 수 있습니다.

- **Application**–애플리케이션 객체(Null일 수도 있음)로 웹 애플리케이션 모듈의 애플리케이션 어댑터에 액세스할 수 있습니다. 다음의 JavaScript 블록은 애플리케이션 제목을 씁니다.

  ```
  <%= Application.Title %>
  ```

- **EndUser**–EndUser 객체를 통해 웹 애플리케이션 모듈의 최종 사용자 어댑터에 액세스할 수 있습니다. 다음의 JavaScript 블록은 최종 사용자 이름을 씁니다.

  ```
  <%= EndUser.DisplayName %>
  ```

- **Session**– 세션 객체를 통해 웹 애플리케이션 모듈의 세션 객체에 액세스할 수 있습니다. 다음의 JavaScript 블록은 세션 ID를 씁니다.

  ```
  <%= Session.SessionID %>
  ```

- **Pages**–페이지 객체(*Pages*)로 애플리케이션 페이지에 액세스할 수 있습니다. 다음의 JavaScript 블록은 게시된 모든 페이지에 대한 링크를 씁니다.

  ```
  <%  e = new Enumerator(Pages)
      for (; !e.atEnd(); e.moveNext())
      {
        if (e.item().Published)
        {
  ```

```
            Response.Write('<A HREF="' + e.item().HREF +'">' + e.item().Title + '</A>')
        }
    }
%>
```

에디터의 Preview 탭은 스크립트 블록의 적절한 결과를 표시하지는 않습니다. 페이지 객체는 디자인 타임 시에는 항상 비어 있는데 이는 웹 페이지 모듈 팩토리가 등록되지 않았기 때문입니다.

- **Modules**–모듈 객체는 애플리케이션 모듈에 대한 액세스를 제공합니다. 다음의 JavaScript 블록은 DM이라는 모듈에 어댑터 필드의 컨텐트를 씁니다.

  ```
  <%= Modules.DM.Adapter1.Field1.DisplayText %>
  ```

- **Page**–Page 객체로 현재 페이지에 액세스할 수 있습니다. 다음의 JavaScript 블록은 현재 페이지의 제목을 씁니다.

  ```
  <%= Page.Title %>
  ```

- **Producer**–Producer 객체로 웹 페이지 모듈의 페이지 프로듀서에 액세스할 수 있습니다. 다음의 JavaScript 블록은 컨텐트를 작성하기 전에 투명 태그를 평가합니다.

  ```
  <% Producer.Write('Here is a tag <#TAG>') %>
  ```

  에디터의 Preview 탭은 이 스크립트 블록의 정확한 결과를 표시하지는 않습니다. 대개 투명 태그를 바꾸는 이벤트 핸들러는 애플리케이션이 실행 중이 아닌 경우에는 실행되지 않습니다.

- **Response**–Response 객체로 WebResponse에 액세스할 수 있습니다. 태그 교체를 원하지 않을 때는 이 객체를 사용하십시오.

  ```
  <% Response.Write('Hello World!') %>
  ```

- **Request**–Request 객체로 WebRequest에 액세스할 수 있습니다. 다음의 JavaScript 블록은 경로 정보를 표시합니다.

  ```
  <%= Request.PathInfo %>
  ```

- **Adapter Objects**–현재 페이지에 있는 모든 어댑터의 컴포넌트는 제한 없이 참조할 수 있습니다. 다른 모듈에 있는 어댑터 객체들은 Modules 객체를 사용하여 제한되어야 합니다. 다음의 JavaScript 블록은 Adapter1의 모든 행에서 *FirstName* 필드의 텍스트 값을 표시합니다.

  ```
  <% e = new Enumerator(Adapter1.Records) %>
    <% for (; !e.atEnd(); e.moveNext()) %>
    <% { %>
        <p><%= Adapter1.FirstName.DisplayText %>
    <% } %>
  ```

이러한 객체에 대한 자세한 내용은 다음 단원의 "WebSnap 서버측 스크립팅 참조"를 참조하십시오.

WebSnap 서버측 스크립팅 참조

이제 WebSnap 서버 애플리케이션에서의 스크립팅 역할에 대해 잘 이해하셨을 것입니다. 이 단원은 HTML 템플릿 페이지에서 스크립트가 사용되는 방법에 대해 보다 자세히 알고 싶은 개발자들을 위한 것입니다. 여기에서는 WebSnap이 동적 HTML 페이지를 생성하기 위해 스크립트를 사용하는 방법에 대해 살펴봅니다.

이 단원은 웹 페이지 모듈의 어댑터 페이지 프로듀서 대신 페이지 프로듀서를 사용하는 개발자들에게 특히 흥미로울 것이며 페이지 템플릿에 스크립트를 프로그래밍하는 데 도움이 될 것입니다. 이 단원은 또한 단순히 어댑터 페이지 프로듀서의 결과물을 자세히 이해하고자 하는 어댑터 페이지 프로듀서 사용자에게도 유익할 것입니다.

전역 객체

다음의 전역 객체는 서버측 스크립트에 의해 참조될 수 있습니다.

표 4.1 WebSnap 전역 객체

객체	설명
Application	Application 객체를 사용하여 Title 필드와 같은 애플리케이션 어댑터의 액션과 필드에 액세스합니다.
EndUser	EndUser 객체를 사용하여 로그인 액션, 로그아웃 액션, 최종 사용자의 DisplayName 등 최종 사용자 어댑터의 액션과 필드에 액세스합니다.
Modules	Modules 객체를 사용하여 데이터 모듈이나 페이지 모듈을 이름으로 참조합니다. Modules 변수는 애플리케이션의 모듈을 열거하는 데 사용되기도 합니다.
Page	Page 객체를 사용하여 Title 페이지와 같은 생성되고 있는 페이지 속성에 액세스합니다.
Pages	Pages 객체를 사용하여 등록된 페이지를 이름으로 참조합니다. Pages 변수는 애플리케이션의 등록된 페이지를 열거하는 데 사용되기도 합니다.
Producer	Producer 객체를 사용하여 투명 태그를 포함하는 HTML 컨텐트를 작성합니다.
Request	Request 객체를 사용하여 HTTP 요청의 속성과 메소드에 액세스합니다.
Response	Response 객체를 사용하여 HTTP 응답에 HTML 컨텐트를 작성합니다.
Session	Session 객체를 사용하여 최종 사용자의 세션 속성에 액세스합니다.

Application

AdapterType *참조*

Application 객체는 애플리케이션과 관련된 정보에 대한 액세스를 제공합니다.

Application 객체를 사용하여 Title 필드와 같은 애플리케이션 어댑터의 액션과 필드에 액세스합니다. Application 객체는 어댑터이므로 추가 필드와 액션을 사용하여 사용자 지정할 수 있습니다. 애플리케이션 어댑터에 추가된 필드와 액션은 Application 객체의 속성으로서 이름으로 액세스할 수 있습니다.

속성

Designing: Boolean, read

예제 1 *참조*

웹 애플리케이션이 IDE에서 디자인 모드인지를 나타냅니다.

Designing 플래그를 사용하여 웹 애플리케이션이 실행 중일 때와 디자인 모드에 있을 때 각각 다른 HTML을 상황에 따라 생성합니다.

ModulePath: text, read

QualifyFileName *참조*

웹 애플리케이션 실행 파일의 위치를 식별합니다.

ModulePath를 사용하여 실행 파일과 동일한 디렉토리에 있는 파일의 이름을 생성합니다.

ModuleFileName: text, read

QualifyFileName *참조*

실행 파일의 전체 파일 이름을 식별합니다.

Title: text, read

예제 18 *참조*

애플리케이션 제목을 제공합니다.

Title 속성은 TApplicationAdapter.Title 컴포넌트 속성의 값입니다. 일반적으로 이 값은 HTML 페이지 위쪽에 표시됩니다.

메소드

QualifyFileName(FileName): text

예제 1 *참조*

상대 파일 이름이나 디렉토리 참조를 절대 참조로 만듭니다.

QualifyFileName은 한정되지 않은 파일 이름을 한정하기 위해 웹 애플리케이션 실행 파일이 있는 디렉토리를 사용합니다. 전체 파일 이름이 반환됩니다. 파일 이름 매개변수가 한정되어 있는 경우 파일 이름은 변경되지 않고 반환됩니다. 디자인 모드인 경우 파일 이름 매개변수는 프로젝트 파일이 있는 디렉토리에 한정됩니다.

EndUser

AdapterType *참조*

EndUser 객체로 현재 최종 사용자에 대한 정보에 액세스합니다.

EndUser 객체를 사용하여 로그인 액션, 로그아웃 액션, 최종 사용자의 DisplayName 등 최종 사용자 어댑터의 액션과 필드에 액세스합니다. 최종 사용자 어댑터에 추가된 필드와 액션은 EndUser 객체의 속성으로서 이름으로 액세스할 수 있습니다.

속성

DisplayName: text, read

예제 19 *참조*

최종 사용자 이름을 제공합니다.

LoggedIn: Boolean, read

예제 19 *참조*

최종 사용자의 로그인 여부를 나타냅니다.

LogInFormAction: AdapterAction, read

예제 19, AdapterActionType *참조*

사용자를 로그인하는 데 사용되는 어댑터 액션을 제공합니다.

LogoutAction: AdapterAction, read

예제 19, AdapterActionType *참조*

사용자를 로그아웃하는 데 사용되는 어댑터 액션을 제공합니다.

Modules

예제 2, 예제 20 *참조*

Modules는 현재 HTTP 요청을 서비스하기 위해 인스턴스화되거나 활성화된 모든 모듈에 대한 액세스를 제공합니다.

특정 모듈을 참조하려면 Modules 변수의 속성으로서 모듈 이름을 사용합니다. 애플리케이션 내의 모든 모듈을 열거하려면 Modules 객체를 사용하여 enumerator를 생성합니다.

Page

예제 5, PageType *참조*

Page는 생성되는 페이지의 속성에 대한 액세스를 제공합니다.

Page 객체의 속성과 메소드에 대한 내용은 PageType을 참조하십시오.

Pages

예제 5 *참조*

Pages는 애플리케이션에 등록된 모든 페이지에 대한 액세스를 제공합니다.

특정 페이지를 참조하려면 Pages 변수 속성으로서 페이지 이름을 사용합니다. 애플리케이션 내의 모든 페이지를 열거하려면 Pages 객체를 사용하여 enumerator를 생성합니다.

Producer

Response *참조*

Producer 객체를 사용하여 투명 태그가 포함된 텍스트를 작성합니다. 태그는 페이지 프로듀서에 의해 번역된 후 HTTP 응답에 태그를 씁니다. 텍스트가 투명 태그를 포함하지 않을 경우 성능을 향상시키려면 Response 객체를 사용하십시오.

속성

Content: text, read/write

HTTP 응답의 컨텐트 부분에 대한 액세스를 제공합니다.

Content를 사용하여 HTTP 응답의 전체 컨텐트 부분을 읽거나 씁니다. Content를 설정하여 투명 태그를 번역합니다. 투명 태그를 사용하지 않을 경우 성능을 향상시키려면 Response.Content를 설정하십시오.

메소드

Write(Value)

투명 태그 지원을 사용하여 HTTP 요청의 컨텐트 부분에 추가합니다.

Write 메소드를 사용하여 HTTP 요청 컨텐트의 컨텐트 부분에 추가합니다. 번역 투명 태그를 씁니다(예: Write('Translate this: <#MyTag>')). 투명 태그를 사용하지 않을 경우 성능을 향상시키려면 Response.Write를 사용하십시오.

Request

HTTP 요청에 대한 액세스를 제공합니다.

Response 객체의 속성을 사용하여 HTTP 요청에 대한 정보에 액세스합니다.

속성

Host: text, read

HTTP 요청의 Host 헤더 값을 보고합니다.

Host는 TWebRequest.Host와 동일합니다.

PathInfo: text, read

URL의 PathInfo 부분을 포함합니다.

PathInfo는 TWebRequest.InternalPathInfo 속성과 동일합니다.

ScriptName: text, read

웹 서버 애플리케이션의 이름을 지정하는 URL의 스크립트 이름 부분을 포함합니다.

ScriptName은 TWebRequest.InternalScriptName 속성과 동일합니다.

Response

Producer *참조*

HTTP 응답에 대한 액세스를 제공합니다. Response 객체를 사용하여 HTTP 응답의 컨텐트 부분에 씁니다. 투명 태그를 쓸 경우 Response 객체 대신 Producer 객체를 사용합니다.

속성

Content: text, read/write

HTTP 응답의 컨텐트 부분에 대한 액세스를 제공합니다.

Content를 사용하여 HTTP 응답의 전체 컨텐트 부분을 읽거나 씁니다.

메소드

Write(Value)

예제 5 *참조*

HTTP 요청의 컨텐트 부분에 추가합니다.

Write 메소드를 사용하여 HTTP 요청 컨텐트의 컨텐트에 추가합니다. Write 메소드는 투명 태그를 번역하지 않습니다.

Producer.Write를 사용하여 하나 이상의 투명 태그를 포함하는 문자열을 씁니다.

Session

Session 객체는 세션 ID와 값에 대한 액세스를 제공합니다.

세션은 짧은 기간 동안 최종 사용자에 대한 정보를 추적하는 데 사용됩니다.

속성

SessionID.Value: text, read/write

현재 최종 사용자 세션 ID에 대한 액세스를 제공합니다.

Values(Name): variant, read

현재 최종 사용자의 세션에 저장된 값에 대한 액세스를 제공합니다.

객체 타입

일부 객체의 속성은 객체입니다. 다음 표는 객체를 속성으로 갖는 객체 유형을 나열한 것입니다. 이러한 유형 이름은 분류상의 편의를 위한 것입니다. WebSnap 애플리케이

션의 특정 인스턴스를 지칭하는 것이 아니라 단순히 객체 유형을 뜻하는 것이므로 서버측 스크립트는 이러한 유형 이름을 인식하지 못합니다.

표 4.2 WebSnap 객체 유형

유형 이름	설명
AdapterType	AdapterType은 어댑터의 속성과 메소드를 정의합니다. 어댑터는 Module 속성으로서 이름으로 액세스할 수 있습니다.
AdapterActionType	AdapterActionType은 어댑터 액션의 속성과 메소드를 정의합니다. 액션은 어댑터 속성으로서 이름으로 참조할 수 있습니다.
AdapterErrorsType	AdapterErrorsType은 어댑터의 Errors 속성을 정의합니다. Errors 속성은 액션을 실행하거나 페이지를 생성할 때 발생하는 오류를 나열하는 데 사용됩니다.
AdapterFieldType	AdapterFieldType은 어댑터 필드의 속성과 메소드를 정의합니다. 필드는 어댑터 속성으로서 이름으로 참조할 수 있습니다.
AdapterFieldValuesType	AdapterFieldValuesType은 어댑터 필드의 Values 속성의 메소드와 속성을 정의합니다.
AdapterFieldValuesListType	AdapterFieldValuesListType은 어댑터 필드의 ValuesList 속성의 메소드와 속성을 정의합니다.
AdapterHiddenFieldsType	AdapterHiddenFieldsType은 어댑터의 HiddenFields와 HiddenRecordFields 속성을 정의합니다.
AdapterImageType	AdapterImageType은 어댑터 필드와 어댑터 액션의 Image 속성을 정의합니다.
ModuleType	ModuleType은 Module 속성을 정의합니다. 모듈은 Modules 변수의 속성으로서 이름으로 액세스할 수 있습니다.
PageType	PageType은 페이지의 속성을 정의합니다. 페이지는 Pages 객체의 속성으로서 이름으로 액세스할 수 있습니다. 생성되고 있는 페이지에는 Page 객체를 사용하여 액세스할 수 있습니다.

AdapterType

어댑터의 속성과 메소드를 정의합니다. 어댑터는 Module 속성으로서 이름으로 액세스할 수 있습니다.

어댑터는 각각 데이터 항목, 명령을 나타내는 필드 컴포넌트와 액션 컴포넌트를 포함합니다. 서버측 스크립트 문은 HTML 폼과 테이블을 만들기 위해 어댑터 필드의 값과 어댑터 액션의 매개변수에 액세스합니다.

속성

Actions: Enumerator

필드, 예제 8 *참조*

액션 객체를 열거합니다. Actions 속성을 사용하여 어댑터의 액션을 루프(순환)합니다.

CanModify: Boolean, read

CanView, AdapterFieldType.CanView *참조*

최종 사용자가 이 어댑터의 필드 수정 권한을 가지고 있는지 여부를 나타냅니다. CanModify 속성을 사용하여 최종 사용자 권한에 중요한 HTML을 동적으로 생성합니다. 예를 들어 CanModify가 True일 경우 <input> 요소를 사용합니다. CanModify가 False일 경우 <p>를 사용합니다.

CanView: Boolean, read

CanModify, AdapterFieldType.CanModify *참조*

최종 사용자가 이 어댑터의 필드 보기 권한을 가지고 있는지 여부를 나타냅니다. CanModify 속성을 사용하여 최종 사용자 권한에 중요한 HTML을 동적으로 생성합니다.

ClassName_: text, read

Name_ *참조*

어댑터 컴포넌트의 클래스 이름을 식별합니다.

Errors: AdapterErrors, read

AdapterErrorsType, 예제 7 *참조*

HTTP 요청을 처리하는 동안 발견된 오류를 열거합니다. 어댑터는 HTML 페이지 생성 시 또는 어댑터 액션 실행 시 발생하는 오류를 캡처합니다. Errors 객체는 오류를 열거하고 HTML 페이지에 오류 메시지를 표시하는 데 사용됩니다.

Fields: Enumerator

Actions *참조*

필드 객체를 열거합니다. Fields 속성을 사용하여 어댑터 필드를 루프합니다.

HiddenFields: AdapterHiddenFields

HiddenRecordFields, AdapterHiddenFieldsType, 예제 10, 예제 22 *참조*

HiddenFields는 어댑터 상태 정보를 전달하는 숨겨진 입력 필드를 정의합니다. 상태 정보의 일례로 TDataSetAdapter의 모드를 들 수 있습니다. "Edit"과 "Insert"는 모두 사용 가능한 모드 값입니다. TDataSetAdapter를 사용하여 HTML 폼을 생성할 때 HiddenFields 객체는 모드의 숨겨진 필드를 정의합니다. HTML 폼이 전송되면 HTTP 요청은 이 숨겨진 필드 값을 포함합니다. 액션을 실행할 때 HTTP 요청에서 모드 값을 추출합니다. "Insert" 모드인 경우 새 행이 데이터셋에 삽입됩니다. "Edit" 모드인 경우 데이터셋 행은 업데이트됩니다.

HiddenRecordFields: AdapterHiddenFields

HiddenFields, AdapterHiddenFieldsType, 예제 10, 예제 22 *참조*

HiddenRecordFields는 각 행에서 필요로 하는 상태 정보나 HTML 폼의 레코드를 전달하는 숨겨진 입력 필드를 정의합니다. 예를 들어 TDataSetAdapter가 HTML 폼 생성에 사용될 경우 HiddenRecordFields 객체는 HTML 테이블의 각 행에 대한 키 값을 식별하는 숨겨진 필드를 정의합니다. HTML 폼이 전송될 경우 HTTP 요청은 이러한 숨겨진 필드 값을 포함합니다. 데이터셋의 여러 행을 업데이트하는 액션을 실행할 때 TDataSetAdapter는 키 값을 사용하여 업데이트할 행을 찾습니다.

Mode: text, read/write

예제 10 *참조*

어댑터 모드를 설정하거나 얻습니다.

일부 어댑터는 모드를 지원합니다. 예를 들어 TDataSetAdapter는 "Edit", "Insert", "Browse", "Query" 모드를 지원합니다. 모드는 어댑터 행동에 영향을 줍니다. TDataSetAdapter가 "Edit" 모드인 경우 전송된 폼은 테이블의 행을 업데이트합니다. TDataSetAdapter가 "Insert" 모드인 경우 전송된 폼은 테이블에 행을 삽입합니다.

Name: text, read

어댑터의 변수 이름을 식별합니다.

Records: Enumerator, read

예제 9 *참조*

어댑터의 레코드를 열거합니다. Records 속성을 사용하여 어댑터 레코드를 루프하고 HTML 테이블을 생성합니다.

AdapterActionType

AdapterType, AdapterFieldType *참조*

AdapterAction은 어댑터 액션의 속성과 메소드를 정의합니다.

속성

Array: Enumerator

예제 11 참조

어댑터 액션의 명령을 열거합니다. Array 속성을 사용하여 명령을 루프합니다. Array 속성은 액션이 여러 명령을 지원하지 않을 경우에는 Null입니다.

여러 명령을 갖는 액션으로 TAdapterGotoPageAction이 있습니다. 이 액션은 부모 어댑터가 정의한 페이지에 대한 명령을 가지고 있습니다. Array 속성은 일련의 하이퍼링크를 생성하는 데 사용되며 최종 사용자는 하이퍼링크를 클릭하여 페이지로 이동할 수 있습니다.

AsFieldValue: text, read

AsHREF, 예제 10, 예제 21 *참조*

숨겨진 필드에서 전송할 수 있는 텍스트 값을 제공합니다.

AsFieldValue는 액션의 이름과 액션의 매개변수를 식별합니다. 이 값을 "__act"라는 숨겨진 필드에 입력합니다. HTML 폼이 전송되면 어댑터 디스패처는 HTTP 요청에서 값을 추출하고 그 값을 사용하여 어댑터 액션을 찾아서 호출합니다.

AsHREF: text, read

AsFieldValue, 예제 11 *참조*

<a> 태그에서 href 속성 값으로 사용할 수 있는 텍스트 값을 제공합니다.

AsHREF는 액션의 이름과 액션의 매개변수를 식별합니다. 이 값을 anchor 태그 안에 넣어 액션 실행 요청을 전송합니다. HTML 폼의 anchor 태그는 폼을 전송하지 않습니다. 액션이 전송된 폼 값을 사용할 경우 숨겨진 폼 필드와 AsFieldValue를 사용하여 액션을 식별합니다.

CanExecute: Boolean, read

최종 사용자가 이 액션을 실행할 권한을 갖는지 여부를 나타냅니다.

DisplayLabel: text, read

예제 21 *참조*

어댑터 액션의 DisplayLabel을 설명합니다.

DisplayStyle: string, read

예제 21 *참조*

어댑터 액션의 HTML 표시 스타일을 설명합니다.

서버측 스크립트에서 DisplayStyle을 사용하여 HTML 생성 방법을 결정할 수 있습니다. 내장되어 있는 어댑터는 다음의 표시 스타일 중 한 가지를 반환할 수 있습니다.

값	의미
''	표시 스타일을 정의 안 함
'Button'	<input type="submit">으로 표시
'Anchor'	<a> 사용

Enabled: Boolean, read

예제 21 *참조*

액션이 HTML 페이지에서 활성화되었는지 여부를 나타냅니다.

Name: string, read

어댑터 액션의 변수 이름을 제공합니다.

Visible: Boolean, read

해당 어댑터 필드가 HTML 페이지에서 보이는지 여부를 나타냅니다.

메소드

LinkToPage(PageSuccess, PageFail): AdapterAction, read

예제 10, 예제 11, 예제 21, Page, AdapterActionType *참조*

LinkToPage를 사용하여 페이지를 지정하고 액션을 실행한 후 표시합니다. 첫 번째 매개변수는 액션이 성공적으로 실행되었는지 여부를 표시하는 페이지 이름입니다. 두 번째 매개변수는 실행 중 오류가 발생하였는지 여부를 표시하는 페이지 이름입니다.

AdapterErrorsType

AdapterType.Errors *참조*

AdapterErrors는 어댑터 오류 속성의 속성을 정의합니다.

속성

Field: AdapterField, read

AdapterFieldType *참조*

오류를 일으키는 어댑터 필드를 식별합니다.

이 속성은 오류가 특정 어댑터 필드와 연결되어 있지 않을 경우 Null입니다.

ID: integer, read

오류에 대한 숫자 식별자를 제공합니다.

이 속성은 ID가 정의되어 있지 않은 경우 0입니다.

Message: text, read

예제 7 *참조*

오류에 대한 텍스트 형식의 설명을 제공합니다.

AdapterFieldType

AdapterType, AdapterActionType *참조*

AdapterField는 어댑터 필드의 속성과 메소드를 정의합니다.

속성

CanModify: Boolean, read

CanView, AdapterType.CanView *참조*

최종 사용자가 필드 값의 수정 권한을 가지고 있는지 여부를 나타냅니다.

CanView: Boolean, read

CanModify, AdapterType.CanModify *참조*

최종 사용자가 필드 값의 보기 권한을 가지고 있는지 여부를 나타냅니다.

DisplayLabel: text, read

어댑터 필드의 DisplayLabel을 나타냅니다.

DisplayStyle: text, read

InputStyle, ViewMode, 예제 17 *참조*

DisplayStyle은 필드 값의 읽기 전용 표현을 표시하는 방법을 나타냅니다.

서버측 스크립트에서 DisplayStyle을 사용하여 HTML 생성 방법을 결정할 수 있습니다. 어댑터 필드는 다음의 표시 스타일 중 한 가지를 반환할 수 있습니다.

값	의미
''	표시 스타일을 정의 안 함
'Text'	<p> 사용
'Image'	<img> 사용. 필드의 Image 속성은 src 속성을 정의합니다.
'List'	<UL> 사용. Values 속성을 열거하여 각 <LI> 항목을 생성합니다.

ViewMode 속성은 HTML을 생성할 때 InputStyle 또는 DisplayStyle 중 어떤 스타일을 사용할지 여부를 나타냅니다.

DisplayText: text, read

EditText, 예제 9 *참조*

읽기 전용의 어댑터 필드 값을 표시할 때 사용할 텍스트를 제공합니다. DisplayText 값에는 숫자 서식을이 포함시킬 수 있습니다.

DisplayWidth: integer, read

MaxLength *참조*

어댑터 필드 값은 표시 너비를 문자 수로 나타냅니다.

표시 너비가 정의되어 있지 않은 경우 −1이 반환됩니다.

EditText: text, read

DisplayText, 예제 10 *참조*

이 어댑터 필드는 HTML 입력을 정의할 때 사용할 텍스트를 제공합니다. EditText 값에는 일반적으로 서식이 지정되어 있지 않습니다.

Image: AdapterImage, read

AdapterImageType, 예제 12 *참조*

이 어댑터 필드는 이미지를 정의하는 객체를 제공합니다.

어댑터 필드가 이미지를 제공하지 않는 경우 Null이 반환됩니다.

InputStyle: text, read

DisplayStyle, ViewMode, 예제 17 *참조*

이 필드는 HTML 입력 스타일을 나타냅니다.

서버측 스크립트에서 InputStyle을 사용하여 HTML 요소를 생성하는 방법을 결정할 수 있습니다. 어댑터 필드는 다음 입력 스타일 중 한 가지를 반환할 수 있습니다.

값	의미
''	입력 스타일을 정의 안 함
'TextInput'	<input type="text"> 사용
'PasswordInput'	<input type="password"> 사용

값	의미
'Select'	<select> 사용. ValuesList 속성을 열거하여 각 <option> 요소를 생성합니다.
'SelectMultiple'	<select multiple> 사용. ValuesList 속성을 열거하여 각 <option> 요소를 생성합니다.
'Radio'	ValuesList 속성을 열거하여 하나 이상의 <input type="radio">를 생성합니다.
'CheckBox'	ValuesList 속성을 열거하여 하나 이상의 <input type="checkbox">를 생성합니다.
'TextArea'	<textarea> 사용
'File'	<input type="file"> 사용

ViewMode 속성은 HTML을 생성할 때 InputStyle 또는 DisplayStyle 중 어떤 스타일을 사용할지를 나타냅니다.

InputName: text, read

예제 10 *참조*

HTML 입력 요소의 이름을 제공하여 어댑터 필드를 편집합니다.

HTML <input>, <select> 또는 <textarea> 요소를 생성할 때 InputName을 사용하여 어댑터 컴포넌트가 어댑터 필드로 HTTP 요청의 이름/값 쌍을 연결할 수 있도록 합니다.

MaxLength: integer, read

DisplayWidth *참조*

필드에 입력할 수 있는 최대 길이를 문자 수로 나타냅니다.

최대 길이가 정의되어 있지 않은 경우 −1이 반환됩니다.

Name: text, read

어댑터 필드의 변수 이름을 반환합니다.

Required: Boolean, read

폼 전송 시 어댑터 필드 값이 필요한지 여부를 나타냅니다.

Value: variant, read

Values, DisplayText, EditText *참조*

계산에 사용할 수 있는 값을 반환합니다. 예를 들어 두 개의 어댑터 필드 값을 함께 추가할 때 Value를 사용합니다.

Values: AdapterFieldValues, read

ValuesList, AdapterFieldValuesType, Value, 예제 13 *참조*

필드 값의 목록을 반환합니다. Values 속성은 이 어댑터 필드가 여러 개의 값을 지원하지 않으면 Null입니다. 예를 들어 값 필드를 여러 개 사용하여 최종 사용자가 선택 목록에서 값을 여러 개 선택하도록 할 수 있습니다.

ValuesList: AdapterFieldValuesList, read

Values, AdapterFieldValuesListType, 예제 13 *참조*

이 어댑터 필드는 선택할 수 있는 목록을 제공합니다. HTML 선택 목록, 체크 박스 그룹, 라디오 버튼 그룹 생성 시 ValuesList를 사용합니다. ValuesList의 각 항목은 값을 가지며 이름도 가질 수 있습니다.

Visible: Boolean, read

이 어댑터 필드가 HTML 페이지에서 보이는지 여부를 나타냅니다.

ViewMode: text, read

DisplayStyle, InputStyle, 예제 17 *참조*

HTML 페이지에 어댑터 필드 값을 표시하는 방법을 나타냅니다.

어댑터 필드는 다음 뷰 모드 중 하나를 반환할 수 있습니다.

값	의미
''	뷰 모드를 정의 안 함
'Input'	<input>, <textarea> 또는 <select>를 사용하여 편집할 수 있는 HTML 폼 요소를 생성합니다.
'Display'	<p>, 또는 <img>를 사용하여 읽기 전용 HTML을 생성합니다.

ViewMode 속성은 HTML을 생성할 때 InputStyle 또는 DisplayStyle 중 어떤 스타일을 사용할지 여부를 나타냅니다.

메소드

IsEqual(Value): Boolean

예제 16 *참조*

이 함수를 호출하여 어댑터 필드의 값과 변수를 비교합니다.

AdapterFieldValuesType

AdapterFieldType.Values *참조*

필드 값의 목록을 제공합니다. 여러 개의 값 어댑터 필드가 이 속성을 지원합니다. 예를 들어 값 필드를 여러 개 사용하여 최종 사용자가 선택 목록에서 값을 여러 개 선택하도록 할 수 있습니다.

속성

Value: variant, read

ValueField *참조*

현재 열거 항목의 Value를 반환합니다.

Records: Enumerator, read

예제 15 *참조*

Value 목록의 레코드를 열거합니다.

ValueField: AdapterField, read

AdapterFieldType, 예제 15 *참조*

현재 열거 항목에 대한 어댑터 필드를 반환합니다. 예를 들어 ValueField를 사용하여 현재 열거 항목에 대한 DisplayText를 얻습니다.

메소드

HasValue(Value): Boolean

예제 14 *참조*

주어진 값이 필드 값의 목록에 있는지 여부를 나타냅니다. 이 메소드를 사용하여 HTML 선택 목록의 항목을 선택할지 또는 체크 박스 그룹의 항목을 검사할지 여부를 결정합니다.

AdapterFieldValuesListType

AdapterType *참조*

어댑터 필드에 사용 가능한 값의 목록을 제공합니다.

HTML 선택 목록, 체크 박스 그룹, 라디오 버튼 그룹 생성 시 ValuesList를 사용합니다. ValuesList의 각 항목은 값을 포함하며 이름을 포함할 수도 있습니다.

속성

Image: AdapterImage, read

현재 열거 항목의 이미지를 반환합니다.

항목에 이미지가 없는 경우 Null이 반환됩니다.

Records: Enumerator, read

Value 목록의 레코드를 열거합니다.

Value: variant, read

현재 열거 항목의 Value를 반환합니다.

ValueField: AdapterField, read

AdapterFieldType, 예제 15 *참조*

현재 열거 항목에 대한 어댑터 필드를 반환합니다. 예를 들어 ValueField를 사용하여 현재 열거 항목에 대한 DisplayText를 얻습니다.

ValueName: text, read

현재 항목의 텍스트 이름을 반환합니다.

값에 이름이 없는 경우 빈 칸이 반환됩니다.

메소드

ImageOfValue(Value): AdapterImage

해당 값과 연결된 이미지를 찾습니다.

이미지가 없는 경우 Null이 반환됩니다.

NameOfValue(Value): text

해당 값과 연결된 이름을 찾습니다.

값을 찾을 수 없거나 값에 이름이 없는 경우 빈 칸이 반환됩니다.

AdapterHiddenFieldsType

AdapterType.HiddenFields, AdapterType.HiddenRecordFields *참조*

어댑터가 변경 사항을 전송하는 데 사용하는 HTML 폼에서 필요로 하는 값과 숨겨진 필드 이름에 대한 액세스를 제공합니다.

속성

Name: text, read

열거할 숨겨진 필드의 이름을 반환합니다.

Value: text, read

열거할 숨겨진 필드의 문자열 값을 반환합니다.

메소드

WriteFields(Response)

예제 10, 예제 22 *참조*

<input type="hidden">을 사용하여 숨겨진 필드 이름과 값을 씁니다.

이 메소드를 호출하여 숨겨진 HTML 필드를 모두 HTML 폼에 씁니다.

AdapterImageType

AdapterFieldType, AdapterActionType *참조*

액션이나 필드와 연결된 이미지를 나타냅니다.

속성

AsHREF: text, read

예제 11, 예제 12 *참조*

HTML <img> 요소를 정의하는 데 사용할 수 있는 URL을 제공합니다.

ModuleType

Modules *참조*

어댑터 컴포넌트는 모듈 속성으로서 이름으로 참조할 수 있습니다. 또한 모듈을 사용하여 모듈의 스크립트를 실행할 수 있는 객체(일반적으로 어댑터)를 열거합니다.

속성

Name_: text, read

예제 20 *참조*

모듈의 변수 이름을 식별합니다. 이 이름은 Modules 변수 속성으로서 모듈에 액세스할 때 사용됩니다.

ClassName_: text, read

예제 20 *참조*

모듈의 VCL 클래스 이름을 식별합니다.

Objects: Enumerator

예제 20 *참조*

Objects를 사용하여 모듈 내에 있는 스크립트를 실행할 수 있는 객체(일반적으로 어댑터)를 열거합니다.

PageType

Page, 예제 20 *참조*

페이지의 속성과 메소드를 정의합니다.

속성

CanView: Boolean, read

최종 사용자가 페이지 보기 권한을 가지고 있는지 여부를 나타냅니다.

페이지는 액세스 권한을 등록합니다. CanView는 페이지가 등록한 권한과 최종 사용자에게 부여된 권한을 비교합니다.

DefaultAction: AdapterAction, read

예제 6 *참조*

해당 페이지와 연결된 기본 어댑터 액션을 식별합니다.

기본 액션은 일반적으로 매개변수를 페이지에 전달해야 하는 경우에 사용됩니다. DefaultAction이 Null일 수도 있습니다.

HREF: text, read

예제 5 *참조*

<a> 태그를 사용하여 해당 페이지와 연결하는 하이퍼링크를 생성하는 데 사용할 수 있는 URL을 제공합니다.

LoginRequired: Boolean, read

최종 사용자가 해당 페이지에 액세스하기 전에 로그인해야 하는지 여부를 나타냅니다.

페이지는 LoginRequired 플래그를 등록합니다. True일 경우 최종 사용자는 로그인하지 않고 해당 페이지에 액세스할 수 없습니다.

Name: text, read

예제 5 *참조*

등록된 페이지의 이름을 제공합니다.

페이지가 게시되고 페이지 이름이 HTTP 요청 경로 정보의 접미사일 때 PageDispatcher는 페이지를 생성합니다.

Published: Boolean, read

예제 5 *참조*

최종 사용자가 URL의 접미사로서 페이지 이름을 지정하여 해당 페이지에 액세스할 수 있는지 여부를 나타냅니다.

페이지는 게시된 플래그를 등록합니다. PageDispatcher는 게시된 페이지를 자동으로 디스패치합니다. 일반적으로 게시된 플래그는 페이지와 하이퍼링크된 메뉴를 생성하는 동안 사용됩니다. 게시된 플래그를 False로 설정하는 Pages는 메뉴에 나열되지 않습니다.

Title: text, read

예제 5, 예제 18 *참조*

해당 페이지의 제목을 제공합니다.

일반적으로 사용자에게는 제목이 표시됩니다.

예제

다음은 서버측 스크립팅 속성과 메소드의 사용 방법을 다양하게 보여 주는 JavaScript 예제입니다.

표 4.3 서버측 스크립팅 속성과 메소드의 예제

예제	설명
예제 1	Application.QualifyFilename 메소드를 사용하여 이미지에 대한 상대 경로 참조를 생성합니다.
예제 2	모듈을 참조하는 변수를 선언합니다.
예제 3	웹 애플리케이션의 모듈을 열거하고 HTML 테이블에 있는 모듈 이름을 표시합니다.
예제 4	등록된 페이지를 참조하는 변수를 선언합니다.

표 4.3 서버측 스크립팅 속성과 메소드의 예제(계속)

예제	설명
예제 5	등록된 페이지를 열거하여 게시된 페이지와 연결된 하이퍼링크의 메뉴를 생성합니다.
예제 6	등록된 페이지를 열거하여 게시된 페이지의 기본 액션과 연결된 하이퍼링크의 메뉴를 생성합니다.
예제 7	어댑터가 발견한 오류의 목록을 씁니다.
예제 8	어댑터의 액션 객체를 모두 열거하여 HTML 테이블에 있는 액션 객체 속성 값을 표시합니다.
예제 9	어댑터의 레코드를 열거하여 HTML 테이블에 있는 어댑터 필드 값을 표시합니다.
예제 10	HTML 폼을 생성하여 어댑터 필드를 편집하고 어댑터 액션을 전송합니다.
예제 11	GotoPage 액션에는 명령이 배열로 되어 있습니다. 열거된 명령은 각 페이지로 이동하는 하이퍼링크를 생성합니다.
예제 12	<img> 태그를 사용하여 어댑터 필드의 이미지를 표시합니다.
예제 13	<select>와 <option> 태그를 사용하여 어댑터 필드를 표시합니다.
예제 14	체크 박스의 그룹으로서 어댑터 필드를 표시합니다.
예제 15	과 태그를 사용하여 어댑터 필드의 값을 표시합니다.
예제 16	라디오 버튼의 그룹으로서 어댑터 필드를 표시합니다.
예제 17	어댑터 필드의 DisplayStyle, InputStyle, ViewMode 속성을 사용하여 HTML을 생성합니다.
예제 18	Application 객체와 Page 객체의 속성을 사용하여 페이지 헤더를 생성합니다.
예제 19	EndUser 객체의 속성을 사용하여 최종 사용자의 이름, 로그인 명령, 로그아웃 명령을 표시합니다.
예제 20	모듈에서 스크립트를 실행할 수 있는 객체를 나열합니다.
예제 21	어댑터 액션의 DisplayStyle 속성을 사용하여 HTML을 생성합니다.
예제 22	HTML 테이블을 생성하여 여러 개의 디테일 레코드를 업데이트합니다.

예제 1

Application.Designing, Application.QualifyFileName, Request.PathInfo *참조*

이 예제는 이미지에 대한 상대 경로 참조를 생성합니다. 스크립트가 디자인 모드인 경우 실제 디렉토리를 참조하고 디자인 모드가 아니면 가상의 디렉토리를 참조합니다.

```
<%
     function PathInfoToRelativePath(S)
     {
       var R = '';
       var L = S.length
       I := 0;
       while (I < L)
       {
         if (S.charAt(I) == '/')
           R = R + '../'
         I++
       }
       return R
     }

     function QualifyImage(S)
     {
```

```
        if (Application.Designing)
          return Application.QualifyFileName("..\\images\\" + S);   // relative directory
        else
          return PathInfoToRelativePath(Request.PathInfo) + '../images/' + S;  // virtual
directory
      }
  %>
```

예제 2

Modules *참조*

이 예제에서는 WebModule1을 참조하는 변수를 선언합니다.

```
<% var M  = Modules.WebModule1 %>
```

예제 3

Modules *참조*

예제 3에서는 다음과 같이 인스턴스화된 모듈을 열거하고 테이블에 있는 이 모듈들의 변수 이름과 클래스 이름을 표시합니다.

```
<table border=1>
<tr><th>Name</th><th>ClassName</th></tr>
<%
   var e = new Enumerator(Modules)
   for (; !e.atEnd(); e.moveNext())
   {
%>
  <tr><td><%=e.item().Name_%></td><td><%=e.item()ClassName._%></td></tr>
<%
   }
%>
</table>
```

예제 4

Pages, Page.Title *참조*

예제 4에서는 Home이라는 페이지를 참조하는 변수를 선언하고 Home의 제목을 표시합니다.

```
<% var P = Pages.Home %>
<p><%= P.Title %></p>
```

예제 5

Pages, Page.Published, Page.HREF, Response.Write *참조*

예제 5에서는 등록된 페이지를 열거하고 게시된 모든 페이지에 연결된 하이퍼링크를 보여 주는 메뉴를 생성합니다.

```
<table>
<td>
<%  e = new Enumerator(Pages)
```

```
      s = ''
      c = 0
      for (; !e.atEnd(); e.moveNext())
      {
        if (e.item().Published)
        {
          if (c>0) s += ' | '
          if (Page.Name != e.item().Name)
            s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
          else
            s += e.item().Title
          c++
        }
      }
      if (c>1) Response.Write(s)
%>
</td>
</table>
```

예제 6

PageType.DefaultAction *참조*

예제 6에서는 등록된 페이지를 열거하고 DefaultActions에 연결된 하이퍼링크를 보여주는 메뉴를 생성합니다.

```
<table>
<td>
<%  e = new Enumerator(Pages)
      s = ''
      c = 0
      for (; !e.atEnd(); e.moveNext())
      {
        if (e.item().Published)
        {
          if (c>0) s += ' | '
          if (Page.Name != e.item().Name)
            if (e.item().DefaultAction != null)
              s += '<a href="'+e.item().DefaultAction.AsHREF+'"> +e.item().Title+'</a>'
            else
              s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
          else
            s += e.item().Title
          c++
        }
      }
      if (c>1) Response.Write(s)
%>
</td>
</table>
```

예제 7

AdapterType.Errors, AdapterErrorsType, Modules, Response.Write *참조*

예제 7에서는 어댑터가 발견한 오류의 목록을 씁니다.

```
<% {
   var e = new Enumerator(Modules.CountryTable.Adapter.Errors)
   for (; !e.atEnd(); e.moveNext())
   {
     Response.Write("<li>" + e.item().Message)
   }
   e.moveFirst()
   } %>
```

예제 8

AdapterType.Actions, AdapterActionType *참조*

이 예제에서는 어댑터 액션을 모두 열거하고 테이블에 있는 액션 속성 값을 표시합니다.

```
<% // Display some properties of an action in a table
   function DumpAction(A)
   {
%>
      <table border="1">
         <tr><th COLSPAN=2><%=A.Name%></th>
         <tr><th>AsFieldValue:</th><td><%= A.AsFieldValue %></td>
         <tr><th>AsHREF:</th><td><%= A.AsHREF %></span>
         <tr><th>DisplayLabel:</th><td><%= A.DisplayLabel %></td>
         <tr><th>Enabled:</th><td><%= A.Enabled %></td>
         <tr><th>CanExecute:</th><td><span class="value"><%= A.CanExecute %></td>
     </table>
<%
  }
%>

<% // Call the DumpAction function for every action in an adapter
   function DumpActions(A)
   {
     var e = new Enumerator(A)
     for (; !e.atEnd(); e.moveNext())
     {
       DumpAction(e.item())
     }
   }
%>


<%
// Display properties of actions in the adapter named Adapter1
DumpActions(Adapter1.Actions) %>
```

예제 9

AdapterType.Records, AdapterFieldType.DisplayText *참조*

예제 9에서는 어댑터의 레코드를 열거하여 HTML 테이블을 생성합니다.

```
<%
// Define variables to reference the adapter and fields that will be used.

vAdapter=Modules.CountryTable.Adapter
vAdapter_Name=vAdapter.Name
vAdapter_Capital=vAdapter.Capital
vAdapter_Continent=vAdapter.Continent
%>

<%
// Function to write column text so that all cells have borders
function WriteColText(t)
{
   Response.Write((t!="")?t:" ")
}
%>

<table border="1">
  <tr>
  <th>Name</th>
  <th>Capital</th>
  <th>Continent</th>
<%
   // Enumerate all the records in the adapter and write the field values.

   var e = new Enumerator(vAdapter.Records)
   for (; !e.atEnd(); e.moveNext())
   { %>
      <tr>
      <td><div><% WriteColText(vAdapter_Name.DisplayText) %></div></td>
      <td><div><% WriteColText(vAdapter_Capital.DisplayText) %></div></td>
      <td><div><% WriteColText(vAdapter_Continent.DisplayText) %></div></td>
      </tr>
<%
   }
%>
</table>
```

예제 10

AdapterActionType.LinkToPage, AdapterActionType.AsFieldValue, AdapterFieldType.InputName, AdapterFieldType.DisplayText, AdapterType.HiddenFields , AdapterType.HiddenRecordFields *참조*

예제 10에서는 HTML 폼을 생성하여 어댑터 필드를 편집하고 어댑터 액션을 전송합니다.

```
<%
// Define variables to reference the adapter, fields, and actions that will be used.

vAdapter=Modules.CountryTable.Adapter
vAdapter_Name=vAdapter.Name
vAdapter_Capital=vAdapter.Capital
vAdapter_Continent=vAdapter.Continent
```

```
vAdapter_Apply=vAdapter.Apply
vAdapter_RefreshRow=vAdapter.RefreshRow

// Put the adapter in "Edit" mode unless the mode is already set. If the mode is already
// set then this is probably because an adapter action set the mode. For example, an insert
// row action would put the adapter in "Insert" mode.

if (vAdapter.Mode=="")
  vAdapter.Mode="Edit"
%>
<form name="AdapterForm1" method="post">

  <!-- This hidden field is used to define the action that is executed when the form is
submitted -->

  <input type="hidden" name="__act">

<%
   // Write hidden fields defined by the adapter.

   if (vAdapter.HiddenFields != null)
   {
     vAdapter.HiddenFields.WriteFields(Response)
   } %>
<% if (vAdapter.HiddenRecordFields != null)
   {
     vAdapter.HiddenRecordFields.WriteFields(Response)
   } %>
  <table>
    <tr>
      <td>
        <table>
          <tr>
          <!-- Write input fields to edit the fields of the adapter -->

          <td>Name</td>
          <td ><input type="text" size="24" name="<%=vAdapter_Name.InputName%>" value="
             <%= vAdapter_Name.EditText %>" ></td>
          </tr>
          <tr>
          <td>Capital</td>
          <td ><input type="text" size="24" name="<%=vAdapter_Capital.InputName%>"
             value="<%= vAdapter_Capital.EditText %>" ></td>
          </tr>
          <tr>
          <td>Continent</td>
          <td ><input type="text" size="24" name="<%=vAdapter_Continent.InputName%>"
             value="<%= vAdapter_Continent.EditText %>" ></td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td>
```

```
        <table>
          <!-- Write submit buttons to execute actions.  LinkToPage is used so that this
               page is regenerated after executing an action. -->

          <tr>
          <td><input type="submit" value="Apply"
            onclick = "AdapterForm1.__act.value='
              <%=vAdapter_Apply.LinkToPage(Page.Name).AsFieldValue%>'"></td>
          <td><input type="submit" value="Refresh"
            onclick = "AdapterForm1.__act.value='
              <%=vAdapter_RefreshRow.LinkToPage(Page.Name).AsFieldValue%>'"> </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</form>
```

예제 11

AdapterActionType.Array, AdapterActionType.AsHREF *참조*

페이징을 지원하는 어댑터 액션을 표시합니다. PrevPage, GotoPage, NextPage 액션이 하이퍼링크로 표시됩니다. GotoPage 액션에는 명령이 배열로 되어 있습니다. 열거된 명령은 각 페이지로 이동하는 하이퍼링크를 생성합니다.

```
<%
   // Define some variables for adapter and actions

   vAdapter = Modules.WebDataModule1.QueryAdapter
   vPrevPage = vAdapter.PrevPage
   vGotoPage = vAdapter.GotoPage
   vNextPage = vAdapter.NextPage
%>

<!-- Generate a table that will display hyperlinks to move between pages of the adapter -->

<table cellpadding="5">
<tr>
<td>
<%
   // Prevpage displays "<<".  Only use an anchor tag if the action is enabled

   if (vPrevPage.Enabled)
   { %>
     <a href="<%=vPrevPage.LinkToPage(Page.Name).AsHREF%>"><<<</a>
<%
   }
   else
   {%>
     <a><<<</a>
 <%} %>
<%
```

```
    // GotoPage has a list of commands.  Loop through the list.  Only use an anchor tag if the command
    //  is enabled

    if (vGotoPage.Array != null)
    {
      var e = new Enumerator(vGotoPage.Array)
      for (; !e.atEnd(); e.moveNext())
      {
%>
        <td>
<%      if (vGotoPage.Enabled)
        { %>
           <a href="<%=vGotoPage.LinkToPage(Page.Name).AsHREF%>">
             <%=vGotoPage.DisplayLabel%></a>
<%      }
        else
        { %>
           <a><%=vGotoPage.DisplayLabel%></a>
<%      }
%>
        </td>
<%
      }
    }
%>
<td>
<%
    // NextPage displays ">>".  Only use an anchor tag if the action is enabled

    if (vNextPage.Enabled)
    { %>
      <a href="<%=vNextPage.LinkToPage(Page.Name).AsHREF%>">>></a>
<%
    }
    else
    {%>
      <a>>></a>
 <%} %>
</td>
</table>
```

예제 12

AdapterFieldType.Image *참조*

예제 12에서는 어댑터 필드의 이미지를 표시합니다.

```
<%
// Declare variables for adapter and field

vAdapter=Modules.WebDataModule3.DataSetAdapter1
vGraphic=vAdapter.Graphic
%>

<!-- Display the adapter field as an image. -->
```

```
<img src="<%=(vGraphic.Image!=null) ? vGraphic.Image.AsHREF : ''%>"
alt="<%=vGraphic.DisplayText%>">
```

예제 13

AdapterFieldType.Values, AdapterFieldType.ValuesList *참조*

예제 13에서는 HTML <select>와 <option> 요소를 사용하여 어댑터 필드를 씁니다.

```
<%
// Return an object that defines HTML select options for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <option> elements.
// count - the number of <option> elements.
// multiple - string containing the either 'multiple' or ''.  Use this value as an attribute
of the
//            <select> element.
//
// Use as follows:
//   obj=SelOptions(f)
//   Response.Write('<select size="' + obj.count + '" name="' + f.InputName + '" ' +
obj.multiple + '>' +
//     obj.text + '</select>')

function SelOptions(f)
{
   var s=''
   var v=''
   var n=''
   var c=0
   if (f.ValuesList != null)
   {
     var e = new Enumerator(f.ValuesList.Records)
     for (; !e.atEnd(); e.moveNext())
     {
       s+= '<option'
       v = f.ValuesList.Value;
       var selected
       if (f.Values == null)
          selected = f.IsEqual(v)
       else
          selected = f.Values.HasValue(v)
       if (selected)
          s += ' selected'
       n = f.ValuesList.ValueName;
       if (n=='')
       {
         n = v
         v = ''
       }
       if (v!='') s += ' value="' + v + '"'
       s += '>' + n + '</option>\r\n'
       c++
     }
```

```
    e.moveFirst()
  }
  r = new Object;
  r.text = s
  r.count = c
  r.multiple = (f.Values == null) ? '' : 'multiple'
  return r;
 }
%>

<%
  // Generate HTML select options for an adapter field
  function WriteSelectOptions(f)
  {
    obj=SelOptions(f)
%>
    <select size="<%=obj.count%>" name="<%=f.InputName%>" <%=obj.multiple%> >
      <%=obj.text%>
    </select>
<%
  }
%>
```

예제 14

AdapterFieldType.Values, AdapterFieldType.ValuesList *참조*

예제 14에서는 <input type="checkbox"> 요소의 그룹으로서 어댑터 필드를 씁니다.

```
<%
// Return an object that defines HTML check boxes for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <input type=checkbox> elements.
// count - the number of <option> elements.
//
// Use as follows to define a check box group with three columns and no additional attributes:
//   obj=CheckBoxGroup(f, 3, '')
//   Response.Write(obj.text)
//
function CheckBoxGroup(f,cols,attr)
{
  var s=''
  var v=''
  var n=''
  var c=0;
  var nm=f.InputName
  if (f.ValuesList == null)
  {
    s+= '<input type="checkbox"'
    if (f.IsEqual(true)) s+= ' checked'
    s += ' value="true"' + ' name="' + nm + '"'
    if (attr!='') s+= ' ' + attr
    s += '></input>\r\n'
    c = 1
```

```
    }
    else
    {
      s += '<table><tr>'
      var e = new Enumerator(f.ValuesList.Records)
      for (; !e.atEnd(); e.moveNext())
      {
        if (c % cols == 0 && c != 0) s += '</tr><tr>'
        s+= '<td><input type="checkbox"'
        v = f.ValuesList.Value;
        var checked
        if (f.Values == null)
          checked = (f.IsEqual(v))
        else
          checked = f.Values.HasValue(v)
        if (checked)
          s+= ' checked'
        n = f.ValuesList.ValueName;
        if (n=='')
          n = v
        s += ' value="' + v + '"' + ' name="' + nm + '"'
        if (attr!='') s+= ' ' + attr
        s += '>' + n + '</input></td>\r\n'
        c++
      }
      e.moveFirst()
      s += '</tr></table>'
    }
    r = new Object;
    r.text = s
    r.count = c
    return r;
  }
%>


<%
    // Write an adapter field as a check box group
    function WriteCheckBoxGroup(f, cols, attr)
    {
       obj=CheckBoxGroup(f, cols, attr)
       Response.Write(obj.text);
    }
%>
```

예제 15

AdapterFieldType.Values, AdapterFieldValuesType, AdapterFieldType.ValuesList *참조*

이 예제에서는 과 <li> 요소를 사용하여 읽기 전용 값의 목록으로서 어댑터 필드를 씁니다.

```
<%
// Return an object that defines HTML list values for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <li> elements.
// count - the number of elements.
//
// text will be blank and count will be zero if the adapter field does not
// support multiple values.
//
// Use as follows to define a displays a read only list of this an adapter
// fields values.
//   obj=ListValues(f)
//   Response.Write('<ul>' + obj.text + '</ul>')
//
function ListValues(f)
{
   var s=''
   var v=''
   var n=''
   var c=0;
   r = new Object;
   if (f.Values != null)
   {
     var e = new Enumerator(f.Values.Records)
     for (; !e.atEnd(); e.moveNext())
     {
       s+= '<li>'
       s += f.Values.ValueField.DisplayText;
       s += '</li>'
       c++
     }
     e.moveFirst()
   }
   r.text = s
   r.count = c
   return r;
}
%>

<%
// Write an adapter field as a list of read-only values
function WriteListValues(f)
{
   obj=ListValues(f)
%>
   <ul><%=obj.text%></ul>
<%
}
%>
```

예제 16

AdapterFieldValuesListType, AdapterFieldType.Values, AdapterFieldType.ValuesList *참조*

예제 16에서는 <input type="radio"> 요소의 그룹으로서 어댑터 필드를 씁니다.

```
<%
// Return an object that defines HTML radio buttons for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <input type=radio> elements.
// count - the number of elements.
//
// Use as follows to define a radio button group with three columns and no additional attributes:
//   obj=RadioGroup(f, 3, '')
//   Response.Write(obj.text)
//

function RadioGroup(f,cols,attr)
{
   var s=''
   var v=''
   var n=''
   var c=0;
   var nm=f.InputName
   if (f.ValuesList == null)
   {
     s+= '<input type="radio"'
     if (f.IsEqual(true)) s+= ' checked'
     s += ' value="true"' + ' name="' + nm + '"'
     if (attr!='') s+= ' ' + attr
     s += '></input>\r\n'
     c = 1
   }
   else
   {
     s += '<table><tr>'
     var e = new Enumerator(f.ValuesList.Records)
     for (; !e.atEnd(); e.moveNext())
     {
       if (c % cols == 0 && c != 0) s += '</tr><tr>'
       s+= '<td><input type="radio"'
       v = f.ValuesList.Value;
       var checked
       if (f.Values == null)
         checked = (f.IsEqual(v))
       else
         checked = f.Values.HasValue(v)
       if (checked)
         s+= ' checked'
       n = f.ValuesList.ValueName;
       if (n=='')
       {
       n = v
```

```
        }
        s += ' value="' + v + '"' + ' name="' + nm + '"'
        if (attr!='') s+= ' ' + attr
        s += '>' + n + '</input></td>\r\n'
        c++
      }
      e.moveFirst()
      s += '</tr></table>'
    }
    r = new Object;
    r.text = s
    r.count = c
    return r;
  }
%>

<%
    // Write an adapter field as a radio button group
    function WriteRadioGroup(f, cols, attr)
    {
       obj=RadioGroup(f, cols, attr)
       Response.Write(obj.text);
    }
%>
```

예제 17

AdapterFieldType.DisplayStyle, AdapterFieldType.ViewMode, AdapterFieldType.InputStyle *참조*

예제 17에서는 필드의 InputStyle, DisplayStyle, ViewMode 속성 값에 기반하여 어댑터 필드의 HTML을 생성합니다.

```
<%
    // Write HTML for an adapter field using the InputStyle, DisplayStyle, and
    // ViewMode properties.
    function WriteField(f)
    {
      Mode = f.ViewMode
      if (Mode == 'Input')
      {
        Style = f.InputStyle
        if (Style == 'SelectMultiple' || Style == 'Select')
           WriteSelectOptions(f)
        else if (Style == 'CheckBox')
          WriteCheckBoxGroup(f, 2, '')
        else if (Style == 'Radio')
          WriteRadioGroup(f, 2, '')
        else if (Style == 'TextArea')
        {
%>
          <textarea wrap=OFF name="<%=f.InputName%>"><%= f.EditText %></textarea>
<%
        }
```

```
      else if (Style == 'PasswordInput')
      {
%>
        <input type="password" name="<%=f.InputName%>"/>
<%
      }
      else if (Style == 'File')
      {
%>
         <input type="file" name="<%=f.InputName%>"/>
<%
      }
      else
      {
%>
        <input type="input" name="<%=f.InputName%>" value="<%= f.EditText %>"/>
<%
      }
    }
    else
    {
      Style = f.DisplayStyle
      if (Style == 'List')
        WriteListValues(f)
      else if (Style == 'Image')
      {
%>
        <img src="<%=(f.Image!=null) ? f.Image.AsHREF : ''%>" alt="<%=f.DisplayText%>">
<%    }
      else
        Response.Write('<p>' + f.DisplayText + '</p>')
    }
  }
%>
```

예제 18

Page, Application.Title *참조*

이 예제에서는 Application 객체와 Page 객체의 속성을 사용하여 페이지 헤더를 생성합니다.

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>

<h2><%= Page.Title %></h2>
```

예제 19

EndUser *참조*

예제 19에서는 EndUser 객체의 속성을 사용하여 최종 사용자의 이름, 로그인 명령, 로그아웃 명령을 표시합니다.

```
<% if (EndUser.Logout != null)
   {
     if (EndUser.DisplayName != '')
     {
%>
       <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   }
     if (EndUser.Logout.Enabled) {
%>
       <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   }
     if (EndUser.LoginForm.Enabled) {
%>
       <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   }
   }
%>
```

예제 20

ModuleType *참조*

이 예제에서는 모듈에서 스크립트를 실행할 수 있는 객체를 나열합니다.

```
<%
   // Write an HTML table list the name and classname of all scriptable objects in a module
   function ListModuleObjects(m)
   {
%>
     <p></p>
     <table border="1">
     <tr>
     <th colspan="2"><%=m.Name_ + ': ' + m.ClassName_%></th>
     </tr>
<%
     var e = new Enumerator(m.Objects)
     for (; !e.atEnd(); e.moveNext())
     {
%>
     <tr>
     <td>
        <%= e.item().Name_ + ': ' + e.item().ClassName_ %>
     </td>
     </tr>
<%
     }
%>
     </table>
```

```
<%
  }
%>
```

예제 21

AdapterActionType.DisplayStyle, AdapterActionType.Enabled *참조*

예제 21에서는 액션의 DisplayStyle 속성에 따라 어댑터 액션의 HTML을 생성합니다.

```
<%
   // Write HTML for an adapter action using the DisplayStyle property.
   //
   // a - action
   // cap - caption.  If blank the action's display label is used.
   // fm - name of the HTML form
   // p - page name to goto after action execution.  If blank, the current page is used.
   //
   // Note that this function does not use the action's Array property.   Is is assumed that
   // the action has a single command.
   //
   function WriteAction(a, cap, fm, p)
   {
       if (cap == '')
         cap = a.DisplayLabel
       if (p == '')
         p = Page.Name
       Style = a.DisplayStyle
       if (Style == 'Anchor')
       {

          if (a.Enabled)
          {
            // Do not use the href property.  Instead, submit the form so that  HTML form
            // fields are part of the HTTP request.
%>
            <a href=""onclick="<%=fm%>.__act.value='
               <%=a.LinkToPage(p).AsFieldValue%>';<%=fm%>.submit();return false;">
               <%=cap%></a>
<%
          }
          else
          {
%>
            <a><%=cap%></a>
<%
          }
       }
       else
       {
%>
          <input type="submit" value="<%= cap%>"
onclick="<%=fm%>.__act.value='<%=a.LinkToPage(p).AsFieldValue%>'">
<%
       }
```

```
  }
%>
```

예제 22

AdapterType.HiddenFields, AdapterType.HiddenRecordFields, AdapterType.Mode *참조*

이 예제에서는 HTML 테이블을 생성하여 여러 개의 디테일 레코드를 업데이트합니다.

```
<%
vItemsAdapter=Modules.DM.ItemsAdapter
vOrdersAdapter=Modules.DM.OrdersAdapter
vOrderNo=vOrdersAdapter.OrderNo
vCustNo=vOrdersAdapter.CustNo
vPrevRow=vOrdersAdapter.PrevRow
vNextRow=vOrdersAdapter.NextRow
vRefreshRow=vOrdersAdapter.RefreshRow
vApply=vOrdersAdapter.Apply
vItemNo=vItemsAdapter.ItemNo
vPartNo=vItemsAdapter.PartNo
vDiscount=vItemsAdapter.Discount
vQty=vItemsAdapter.Qty
%>

<!-- Use two adapters to update multiple detail records.
     The orders adapter is associated with the master dataset.
     The items adapter is associated with the detail dataset.
     Each row in a grid displays values from the items adapter.  One
     column display an <input> element for editing Qty.  The apply button
     updates the Qty value in each detail record.

<!-- Display the order number and customer number values -->
<h2>OrderNo: <%= vOrderNo.DisplayText %></h2>
<h2>CustNo: <% vCustNo.DisplayText %></h2>

<%
   // Put the items adapter in edit mode because this form updates
   // the Qty field.
   vItemsAdapter.Mode = 'Edit'
%>

<form name="AdapterForm1" method="post">

  <!-- Define a hidden field for submitted the action name and parameters -->
  <input type="hidden" name="__act">

<%
   // Write hidden fields containing state information about the
   // orders adapter and items adapter.
   if (vOrdersAdapter.HiddenFields != null)
     vOrdersAdapter.HiddenFields.WriteFields(Response)
   if (vItemsAdapter.HiddenFields != null)
     vItemsAdapter.HiddenFields.WriteFields(Response)
```

```
    // Write hidden fields containing state information about the current
    // record of the orders adapter.
    if (vOrdersAdapter.HiddenRecordFields != null)
      vOrdersAdapter.HiddenRecordFields.WriteFields(Response)%>

  <table border="1">
    <tr>
    <th>ItemNo</th>
    <th>PartNo</th>
    <th>Discount</th>
    <th>Qty</th>
  </tr>
<%
   var e = new Enumerator(vItemsAdapter.Records)
   for (; !e.atEnd(); e.moveNext())
   { %>
     <tr>
     <td><%=vItemNo.DisplayText%></td>
     <td><%=vPartNo.DisplayText%></td>
     <td><%=vDiscount.DisplayText%></td>
     <td><input type="text" name="<%=vQty.InputName%>" value="<%= vQty.EditText %>" ></td>
     </tr>
<%
     // Write hidden fields containing state information about each record of the
     // items adapter.   This is needed by the items adapter when updating the Qty field.

     if (vItemsAdapter.HiddenRecordFields != null)
       vItemsAdapter.HiddenRecordFields.WriteFields(Response)
   }
%>
  </table>
  <p></p>
  <table>
    <td><input type="submit" value="Prev Order"

onclick="AdapterForm1.__act.value='<%=vPrevRow.LinkToPage(Page.Name).AsFieldValue%>'"></td>
    <td><input type="submit" value="Next Order"

onclick="AdapterForm1.__act.value='<%=vNextRow.LinkToPage(Page.Name).AsFieldValue%>'"></td>
    <td><input type="submit" value="Refresh"

onclick="AdapterForm1.__act.value='<%=vRefreshRow.LinkToPage(Page.Name).AsFieldValue%>'"></
td>
    <td><input type="submit" value="Apply"
      onclick="AdapterForm1.__act.value='<%=vApply.LinkToPage(Page.Name).AsFieldValue%>'"></
td>
  </table>
</form>
```

요청 디스패치

WebSnap 애플리케이션은 HTTP 요청 메시지를 수신하면 HTTP 요청 메시지를 나타내는 *TWebRequest* 객체를 만들고, 반환해야 할 응답을 나타내는 *TWebResponse* 객체를 만듭니다.

웹 컨텍스트

요청을 처리하기 전에 웹 애플리케이션 모듈은 *TWebContext* 타입의 웹 컨텍스트 객체를 초기화합니다. 웹 컨텍스트 객체는 전역 WebContext 함수를 통해 액세스할 수 있으며 스레드 변수로서 요청을 서비스하는 컴포넌트에 의해 사용되는 변수에 대한 전역 액세스를 제공합니다. 예를 들어 웹 컨텍스트는 *TWebResponse*와 *TWebRequest* 객체뿐 아니라 이 단원의 뒷부분에서 다루게 될 어댑터 요청과 어댑터 응답 객체를 포함합니다.

디스패처 컴포넌트

디스패처 컴포넌트는 웹 애플리케이션 모듈 내에서 애플리케이션의 흐름을 제어합니다. 디스패처는 HTTP 요청을 검사하여 특정한 유형의 HTTP 요청 메시지를 처리하는 방법을 결정합니다.

어댑터 디스패처 컴포넌트(*TAdapterDispatcher*)는 어댑터 액션 컴포넌트 또는 어댑터 이미지 필드 컴포넌트를 식별하는 컨텐트 필드나 쿼리 필드를 찾습니다. 어댑터 디스패처가 컴포넌트를 찾으면 제어를 컴포넌트에 넘깁니다.

웹 디스패처 컴포넌트(*TWebDispatcher*)는 특정 타입의 HTTP 요청 메시지를 처리하는 방법을 알고 있는 액션 항목(*TWebActionItem*) 컬렉션을 유지 관리합니다. 웹 디스패처는 요청과 일치하는 액션 항목을 찾습니다. 일치하는 액션 항목을 찾은 경우 액션 항목에 제어를 넘깁니다. 웹 디스패처는 요청을 처리할 수 있는 자동 디스패칭 컴포넌트도 찾습니다.

페이지 디스패처 컴포넌트(*TPageDispatcher*)는 *TWebRequest* 객체의 pathInfo 속성을 검사하여 등록된 웹 페이지 모듈의 이름을 찾습니다. 디스패처가 웹 페이지 모듈 이름을 찾은 경우 웹 페이지 모듈에 제어를 넘깁니다.

어댑터 디스패처 작업

어댑터 디스패처 컴포넌트는 어댑터 액션과 필드 컴포넌트를 호출하여 HTML 폼 전송, 동적 이미지 요청을 자동으로 처리합니다.

어댑터 컴포넌트를 사용하여 컨텐트 생성

WebSnap 애플리케이션이 자동으로 어댑터 액션을 실행하고 어댑터 필드에서 동적 이미지를 가져오려면 HTML 컨텐트가 제대로 구성되어 있어야 합니다. HTML 컨텐트가 제대로 구성되지 않은 경우 결과 HTTP 요청은 어댑터 디스패처가 어댑터 액션과 필드 컴포넌트를 호출하기 위해 필요한 정보를 포함하지 않습니다.

HTML 페이지를 구성할 때 오류를 줄이기 위해서 어댑터 컴포넌트는 HTML 요소의 이름과 값을 나타냅니다. 어댑터 컴포넌트에는 어댑터 필드 업데이트 시 사용하는 HTML 폼 상에 표시되어야 하는 숨겨진 필드의 이름과 값을 가져오는 메소드가 있습니다. 대개 페이지 프로듀서는 서버측 스크립트를 사용하여 어댑터 컴포넌트에서 이름과 값을 가져오고 가져온 이름과 값으로 HTML을 생성합니다. 예를 들어 다음의 스크립트는 Adapter1에서 Graphic 필드를 참조하는 <IMG> 요소를 구성합니다.

```
<img src="<%=Adapter1.Graphic.Image.AsHREF%>" alt="<%=Adapter1.Graphic.DisplayText%>">
```

웹 애플리케이션이 스크립트를 평가하면 HTML src 속성은 필드 컴포넌트가 이미지를 가져오기 위해 필드와 매개변수를 식별하는 데 필요한 정보를 포함하게 됩니다. 결과 HTML은 다음과 같습니다.

```
<img src="?_lSpecies No=90090&__id=DM.Adapter1.Graphic" alt="(GRAPHIC)">
```

브라우저가 이 이미지를 웹 애플리케이션에 가져오기 위해 HTTP 요청을 보내면 어댑터 디스패처는 모듈 DM에서 Adapter1의 Graphic 필드가 매개변수 "Species No=90090"으로 호출되도록 지정할 수 있습니다. 어댑터 디스패처는 Graphic 필드를 호출하여 적절한 HTTP 응답을 씁니다.

다음 스크립트는 Adapter1의 EditRow 액션과 페이지 "Details"를 참조하는 <A> 요소를 구성합니다.

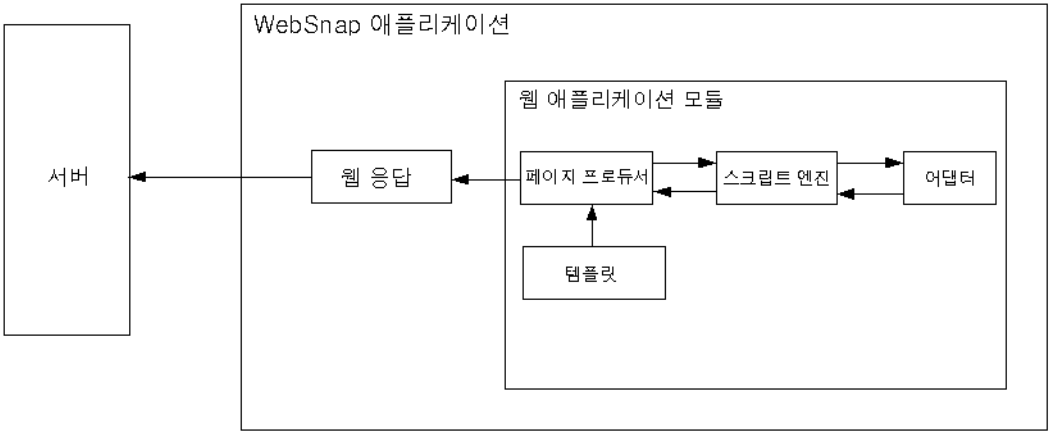
```
<a href="<%=Adapter1.EditRow.LinkToPage("Details", Page.Name).AsHREF%>">Edit...</a>
```

결과 HTML은 다음과 같습니다.

```
<a href="?&_lSpecies No=90310&__sp=Edit&__fp=Grid&__id=DM.Adapter1.EditRow">Edit...</a>
```

최종 사용자가 하이퍼링크를 클릭하면 브라우저가 HTTP 요청을 보내고 어댑터 디스패처는 모듈 DM에서 Adapter1의 EditRow 액션이 매개변수 "Species No=903010"으로 호출되도록 결정할 수 있습니다. 또한 어댑터 디스패처는 액션이 성공적으로 실행될 경우에는 Edit 페이지를 표시하고 액션 실행이 실패할 경우 Grid 페이지를 표시하도록 지시합니다. 그런 다음 EditRow 액션을 호출하여 편집할 행을 찾고 Edit라는 이름의 페이지가 호출되어 HTTP 응답을 생성합니다. 그림 8.12는 어댑터 컴포넌트를 사용하여 컨텐트를 생성하는 방법을 보여 줍니다.

그림 8.12 컨텐트 흐름 생성



어댑터 요청 및 응답

어댑터 디스패처는 클라이언트 요청을 받으면 어댑터 요청과 어댑터 응답 객체를 만들어 HTTP 요청에 관한 정보를 보유합니다. 어댑터 요청과 어댑터 응답 객체는 웹 컨텍스트에 저장되어 요청을 처리하는 동안 액세스할 수 있습니다.

어댑터 디스패처는 두 가지 유형의 어댑터 요청 객체, 즉 액션과 이미지를 생성합니다. 어댑터 디스패처는 어댑터 액션을 실행할 때 액션 요청 객체를 만들고 어댑터 필드에서 이미지를 가져올 때 이미지 요청 객체를 만듭니다.

어댑터 응답 객체는 어댑터 컴포넌트가 어댑터 액션 또는 어댑터 이미지 요청에 대한 응답을 나타내는 데 사용됩니다. 어댑터 응답 객체에는 액션과 이미지의 두 가지 유형이 있습니다.

액션 요청

액션 요청 객체는 HTTP 요청을 어댑터 액션을 실행하는 데 필요한 정보로 나누는 일을 합니다. 어댑터 액션을 실행하는 데 필요한 정보 타입은 다음과 같습니다.

- **Component Name**–어댑터 액션 컴포넌트를 식별합니다.
- **Adapter Mode**–어댑터는 모드를 정의할 수 있습니다. 예를 들어 *TDataSetAdapter*는 편집, 삽입, 찾기 모드를 지원합니다. 어댑터 액션은 모드에 따라 다르게 실행할 수도 있습니다. 예를 들어 *TDataSetAdapter* Apply 액션은 삽입 모드에서 새 레코드를 추가하고 편집 모드에서는 레코드를 업데이트합니다.
- **Success Page**–성공 페이지는 액션을 성공적으로 실행한 후에 표시되는 페이지를 식별합니다.
- **Failure Page**–실패 페이지는 액션을 실행하는 동안 오류가 발생할 경우 표시되는 페이지를 식별합니다.
- **Action Request Parameters**–어댑터 액션에서 필요로 하는 매개변수를 나타냅니다. 예를 들어 *TDataSetAdapter* Apply 액션은 업데이트할 레코드를 나타내는 키 값을 포함합니다.

- **Adapter Field Values**–HTML 폼이 전송될 때 HTTP 요청에 전달되는 어댑터 필드의 값입니다. 필드 값은 최종 사용자가 입력한 새로운 값, 어댑터 필드의 원래 값, 업로드된 파일을 포함할 수 있습니다.
- **Record Keys**–HTML 폼이 여러 레코드의 변경 사항을 전송할 경우 어댑터 액션 컴포넌트에 의해 사용되는 키는 각 레코드를 고유하게 나타내어 어댑터 액션이 각 레코드에서 수행될 수 있도록 하기 위해 필요합니다. 예를 들어 *TDataSetAdapter* Apply 액션이 여러 레코드에서 수행되면 데이터셋 필드를 업데이트하기 전에 레코드 키를 사용하여 데이터셋의 각 레코드를 찾을 수 있습니다.

액션 응답

액션 응답 객체는 어댑터 액션 컴포넌트를 대신하여 HTTP 응답을 생성합니다. 어댑터 액션은 객체 내에서 속성을 설정하거나 액션 응답 객체에서 메소드를 호출하여 응답 유형을 나타냅니다. 속성은 다음과 같습니다.

- *Redirect Options*–리디렉션 옵션은 HTML 컨텐트를 반환하는 대신 HTTP 리디렉션을 수행할지 여부를 나타냅니다.
- *Execution Status*–상태를 success로 설정하면 기본 액션 응답은 액션 요청에 확인된 성공 페이지의 컨텐트가 됩니다.

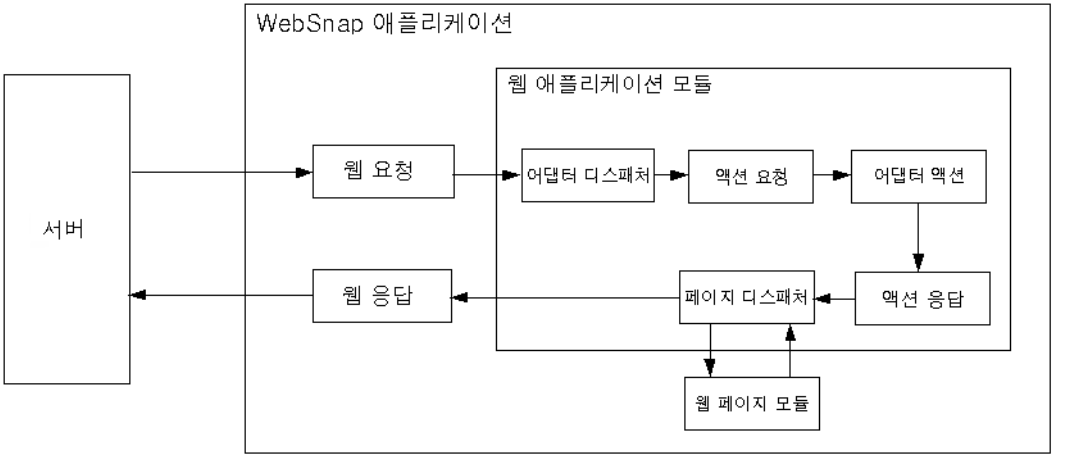
액션 응답 메소드는 다음과 같습니다.

- *RespondWithPag* –어댑터 액션은 특정 웹 페이지 모듈이 응답을 생성해야 할 때 이 메소드를 호출합니다.
- *RespondWithComponent*–어댑터 액션은 이 컴포넌트가 들어 있는 웹 페이지 모듈에서 응답이 나와야 할 때 이 메소드를 호출합니다.
- *RespondWithURL*–어댑터 액션은 지정된 URL에 대한 리디렉션이 응답일 때 이 메소드를 호출합니다.

페이지로 응답할 때 액션 응답 객체는 페이지 디스패처를 사용하여 페이지 컨텐트를 생성하려고 합니다. 페이지 디스패처를 찾지 못하면 직접 웹 페이지 모듈을 호출합니다.

그림 8.15는 액션 요청과 액션 응답 객체가 요청을 처리하는 방법을 보여 줍니다.

그림 8.13 액션 요청과 응답



이미지 요청

이미지 요청 객체는 HTTP 요청을 어댑터 이미지 필드가 필요로 하는 정보로 나누어 이미지를 생성하는 일을 합니다. 이미지 요청에 의해 나타나는 정보의 유형은 다음과 같습니다.

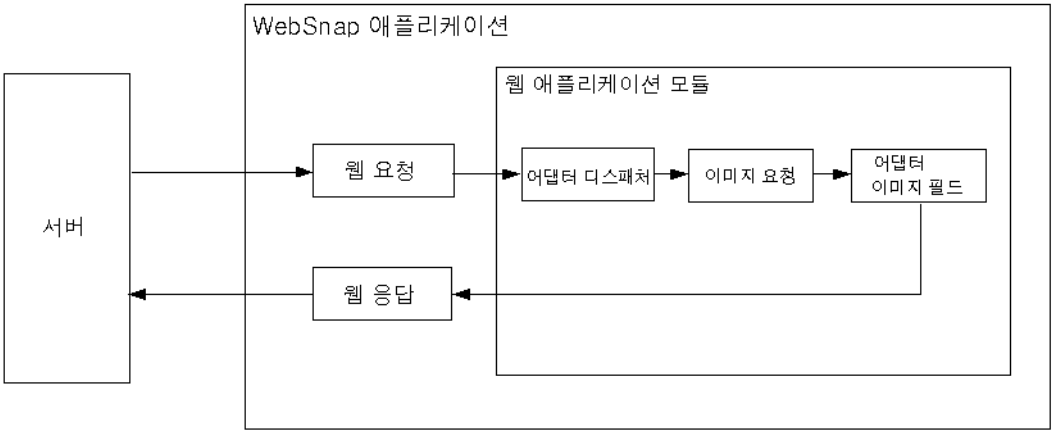
- Component Name – 어댑터 필드 컴포넌트를 식별합니다.
- Image Request Parameters – 어댑터 이미지가 필요로 하는 매개변수를 식별합니다. 예를 들어 *TDataSetAdapterImageField* 객체에는 이미지가 포함된 레코드를 식별하는 키 값이 필요합니다.

이미지 응답

이미지 응답 객체는 *TWebResponse* 객체를 포함합니다. 어댑터 필드는 웹 응답 객체에 이미지를 작성함으로써 어댑터 요청에 응답합니다.

그림 8.14는 어댑터 이미지 필드가 요청에 어떻게 응답하는지 보여 줍니다.

그림 8.14 요청에 대한 이미지 응답



액션 항목 디스패칭

웹 디스패처(*TWebDispatcher*)는 다음과 같은 액션 항목의 목록을 검색합니다.

- 대상 URL 요청 메시지의 Pathinfo 부분과 일치하는 액션
- 요청 메시지의 메소드로 지정된 서비스를 제공할 수 있는 액션

TWebRequest 객체의 *PathInfo* 및 *MethodType* 속성과 액션 항목에 있는 PathInfo 및 MethodType 속성을 비교하여 액션 항목을 검색합니다.

디스패처는 적절한 액션 항목을 찾으면 해당 액션 항목을 실행하며 이 때 액션 항목은 다음 중 하나를 실행합니다.

- 응답 컨텐트를 완성하고 요청이 완전히 처리되었다는 응답 또는 신호를 보냅니다.
- 응답에 추가한 후 다른 액션 항목이 작업을 완료할 수 있게 해줍니다.
- 요청을 다른 액션 항목에 전달합니다.

디스패처가 액션 항목을 모두 확인한 뒤에도 메시지가 정확히 처리되지 않았으면 디스패처는 액션 항목을 사용하지 않는 특별히 등록된 자동 디스패칭 컴포넌트를 확인합니다. 이것은 다계층 데이터베이스 애플리케이션에만 있는 고유 컴포넌트입니다. 아직 요청 메시지가 완전히 처리되지 않았다면 디스패처는 기본 액션 항목을 호출합니다. 기본 액션 항목은 대상 URL 또는 요청의 메소드와 일치하지 않아도 됩니다.

페이지 디스패처 작업

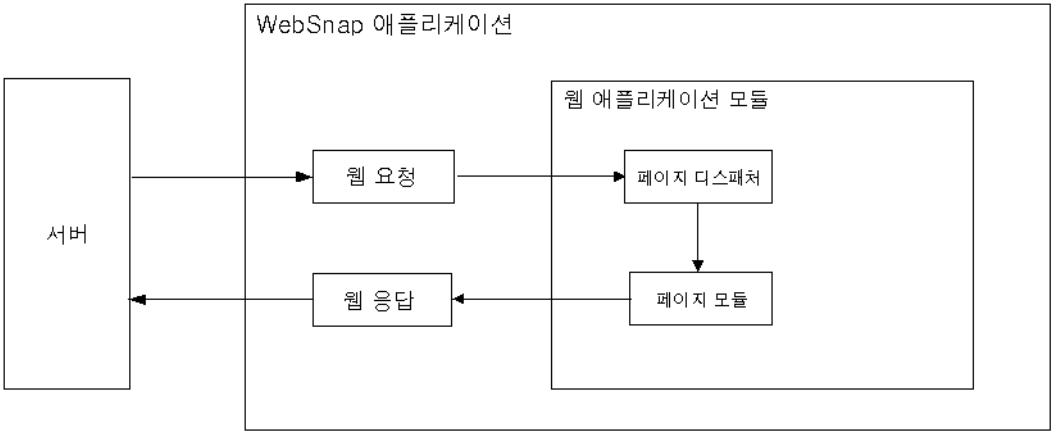
페이지 디스패처가 클라이언트 요청을 받으면 대상 URL의 요청 메시지의 Pathinfo 부분을 확인하여 페이지 이름을 알아냅니다. 경로 정보 부분이 비어 있지 않으면 페이지 디스패처는 경로 정보의 끝 단어를 페이지 이름으로 사용합니다. 경로 정보 부분이 비어 있으면 페이지 디스패처는 기본 페이지 이름을 지정하려고 시도합니다.

페이지 디스패처의 *DefaultPage* 속성에 페이지 이름이 있으면 페이지 디스패처는 이 이름을 기본 페이지 이름으로 사용합니다. *DefaultPage* 속성이 비어 있고 웹 애플리케이션 모듈이 페이지 모듈인 경우 페이지 디스패처는 웹 애플리케이션 모듈의 이름을 기본 페이지 이름으로 사용합니다.

페이지 이름이 비어 있지 않으면 페이지 디스패처는 일치하는 이름을 갖는 웹 페이지 모듈을 찾고 웹 페이지 모듈을 찾으면 그 모듈을 호출하여 응답을 생성합니다. 페이지 이름이 비어 있거나 페이지 디스패처가 웹 페이지 모듈을 찾지 못하면 페이지 디스패처는 예외를 생성합니다.

그림 8.15는 페이지 디스패처가 요청에 어떻게 응답하는지 보여 줍니다.

그림 8.15 페이지 디스패칭



9

XML 문서 사용

* 이 장은 예를 Xylix2 개발자 안내서의 2장입니다.

XML(Extensible Markup Language)은 구조화된 데이터를 나타내는 마크업 랭귀지입니다. 태그가 표시 특성이 아닌 정보의 구조를 나타낸다는 것을 제외하고는 HTML과 유사합니다. XML 문서는 컨텐트뿐만 아니라 컨텐트의 구조를 나타내는 메타 정보도 포함하고 있습니다. XML은 애플리케이션뿐만 아니라 운영 체제 간에 데이터를 전송하는 표준화된 방식이므로 문서 클래스의 전체 구조를 나타내는 데 사용할 수 있습니다. XML 문서는 검색과 편집이 용이하도록 정보를 저장하는 간단한 텍스트 방식을 제공합니다. XML은 웹 애플리케이션, B2B 통신 등에서 데이터에 대한 표준 전송 포맷으로 사용되기도 합니다.

XML 문서는 데이터 몸체의 계층적인 뷰를 제공합니다. 다음 문서에서 보여 주는 XML 문서의 태그는 데이터 요소 각각의 역할과 의미를 나타내며 이 문서의 내용은 주식에 관한 것입니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE StockHoldings SYSTEM "sth.dtd">
<StockHoldings>
   <Stock exchange="NASDAQ">
      <name>Inprise (Borland)</name>
      <price>15.375</price>
      <symbol>INPR</symbol>
      <shares>100</shares>
   </Stock>
   <Stock exchange="NYSE">
      <name>Pfizer</name>
      <price>42.75</price>
      <symbol>PFE</symbol>
      <shares type="preferred">25</shares>
   </Stock>
</StockHoldings>
```

이 예제는 XML 문서의 여러 가지 일반적인 요소를 보여 줍니다. 첫 번째 줄은 XML 선언이라는 처리 명령입니다. XML 선언은 옵션이지만 문서에 관한 유용한 정보를 제공하

기 때문에 포함시켜야 합니다. 예제의 XML 선언을 보면 이 문서가 XML 사양의 버전 1.0을 따르며 UTF-8 문자 인코딩을 사용하고 DTD(Document Type Declaration)는 외부 파일을 사용한다는 것을 알 수 있습니다.

<!DOCType> 태그로 시작하는 두 번째 줄은 DTD입니다. DTD란 XML 문서의 구조를 정의하는 방식입니다. DTD는 문서에 포함된 요소(태그)에 구문 규칙을 적용합니다. 이 예제의 DTD는 sth.dtd를 참조합니다. 이 예제에서는 XML 문서 자체가 아닌 외부 파일로 구조를 정의합니다. 외부 파일을 사용하면 XML은 문서 클래스 전체의 구조를 정의할 수 있습니다. XML 문서의 DTD로 클래스의 구조를 완벽하게 정의하는 외부 파일을 참조함으로써 XML 문서가 특정 클래스에 속한다는 것을 알 수 있습니다. XML 문서의 구조를 나타내는 다른 파일 타입으로는 Reduced XML Data(XDR) 와 XML 스키마(XSD)가 있습니다.

나머지 줄은 루트 노드(<StockHoldings> 태그)가 하나인 계층 구조로 구성되어 있습니다. 이 계층 구조의 노드마다 자식 노드 집합이나 텍스트 값이 포함되어 있습니다. 일부 태그(<Stock> 및 <shares> 태그)는 태그 해석 방법에 대한 세부 사항을 제공하는 속성으로 Name=Value 쌍을 포함합니다.

XML 문서의 텍스트를 직접 수정할 수도 있지만 일반적으로 애플리케이션에서는 데이터를 구문 분석하거나 편집하기 위해 몇 가지 툴을 추가적으로 사용합니다. W3C는 DOM(Document Object Model)이라는 구문 분석된 XML 문서를 표현하는 표준 인터페이스 집합을 정의합니다. 대부분의 공급 업체들은 XML 문서를 쉽게 해석하고 편집할 수 있도록 DOM 인터페이스를 구현한 XML 파서를 사용자에게 제공합니다.

Kylix는 XML 문서에 사용하는 여러 가지 추가 툴을 제공합니다. 이 툴들은 다른 공급 업체가 제공하는 DOM 파서를 사용하며 XML 문서를 훨씬 더 쉽게 사용할 수 있도록 합니다. 이 장에서는 이 툴에 대해 설명합니다.

DOM 사용

DOM(Document Object Model)은 구문 분석된 XML 문서를 표현하는 표준 인터페이스 집합입니다. Kylix에는 IBM에서 제공하는 DOM 구현 및 Open XML DOM 구현(www.philo.de/xml)이 포함되어 있습니다. 다른 공급 업체의 추가 DOM 구현을 Kylix XML 프레임워크에 통합할 수 있게 하는 등록 메커니즘도 포함되어 있습니다.

xmldom 유닛에는 W3C XML DOM 레벨 2 사양에 정의된 모든 DOM 인터페이스에 대한 선언이 포함되어 있습니다. DOM 공급 업체마다 DOM 인터페이스에 대한 구현을 제공합니다.

- IBM 구현을 사용하려면 사용자의 uses 절에 ibmxmldom 유닛을 포함시킵니다.
- Open XML 구현을 사용하려면 uses 절에 oxmldom 유닛을 포함시킵니다.

- 다른 DOM 구현을 사용하려면 (최소한) *TDOMVendor* 클래스의 자손을 생성하는 유닛을 만들어야 합니다. 자손 클래스에서 공급 업체를 식별하는 문자열을 반환하는 추상 *Description* 함수를 구현해야 합니다. 또한 상위 레벨 인터페이스 (*IDOMImplementation*)를 반환하는 *DOMImplementation* 함수를 구현해야 합니다. 유닛의 초기화 섹션에서 전역 *RegisterDOMVendor* 프로시저를 호출하여 공급 업체를 등록합니다. 유닛의 완료 섹션에서 전역 *UnRegisterDOMVendor* 프로시저를 호출하여 공급 업체를 제거합니다.

DOM 인터페이스를 직접 사용하여 XML 문서를 구문 분석하고 편집할 수 있습니다. *GetDOM* 함수를 호출하여 시작점으로 사용할 수 있는 *IDOMImplementation* 인터페이스를 얻습니다.

참고 DOM 인터페이스에 대한 자세한 설명은 DOM 공급 업체가 제공하는 문서인 XMLDOM 유닛의 선언을 참조하거나 W3C 웹 사이트(www.w3.org)에서 제공하는 DOM 사양을 참조하십시오.

DOM 인터페이스를 직접 사용하는 것보다 Kylix의 XML 클래스를 사용하는 것이 더 편리합니다. 이 내용은 다음 단원에서 설명합니다.

XML 컴포넌트 사용

Kylix는 XML 문서를 처리하기 위해 많은 클래스와 인터페이스를 정의합니다. Kylix에서 제공하는 클래스와 인터페이스를 사용하면 XML 문서의 로드, 편집 및 저장 프로세스가 간편해집니다.

TXMLDocument 사용

XML 문서를 사용하는 시작점은 *TXMLDocument* 컴포넌트입니다. *TXMLDocument*를 사용하여 XML 문서를 직접 사용하려면 다음 단계를 따릅니다.

1. *TXMLDocument* 컴포넌트를 폼이나 데이터 모듈에 추가합니다. *TXMLDocument*는 컴포넌트 팔레트의 Internet 페이지에 있습니다.
2. *DOMVendor* 속성을 설정하여 XML 문서의 구문 분석과 편집에 사용하려는 DOM 구현을 지정합니다. Object Inspector에 현재 등록된 모든 DOM 공급 업체가 나열됩니다. DOM 구현에 대한 내용은 9-2페이지의 "DOM 사용"을 참조하십시오.
3. 기존 XML 문서를 사용하려면 다음과 같이 문서를 지정합니다.
   - XML 문서를 파일에 저장하려면 *FileName* 속성을 해당 파일의 이름으로 설정합니다.
   - *XML* 속성을 사용하는 대신 XML 문서를 문자열로 지정할 수 있습니다.
4. *Active* 속성을 *True*로 설정합니다.

일단 *TXMLDocument* 객체가 활성화되면 노드의 계층 구조의 값을 읽거나 설정할 수 있습니다. 계층 구조의 루트 노드는 *DocumentElement* 속성으로 사용할 수 있습니다.

XML 노드 사용

일단 XML 문서가 DOM 구현에 의해 구문 분석되었으면 문서에 표현된 데이터는 노드의 계층 구조로 사용할 수 있습니다. 각 노드는 문서의 태그 요소에 해당합니다. 예를 들어 다음과 같은 XML이 있다고 가정합니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE StockHoldings SYSTEM "sth.dtd">
<StockHoldings>
   <Stock exchange="NASDAQ">
      <name>Inprise (Borland)</name>
      <price>15.375</price>
      <symbol>INPR</symbol>
      <shares>100</shares>
   </Stock>
   <Stock exchange="NYSE">
      <name>Pfizer</name>
      <price>42.75</price>
      <symbol>PFE</symbol>
      <shares type="preferred">25</shares>
   </Stock>
</StockHoldings>
```

*TXMLDocument*는 다음과 같이 노드의 계층 구조를 생성합니다. 계층 구조의 루트는 *StockHoldings* 노드입니다. *StockHoldings*에는 자식 노드로 *Stock* 태그가 두 개 있습니다. 각각의 자식 노드마다 네 개의 자식 노드(*name*, *price*, *symbol*, *shares*)가 있습니다. 이 네 개의 자식 노드는 리프(leaf) 노드로 작동합니다. 네 개의 자식 노드의 텍스트는 각 리프 노드의 값으로 나타납니다.

참고 이러한 노드 분할은 DOM 구현이 XML 문서에 대한 노드를 생성하는 방식과는 약간 다릅니다. 특히 DOM 파서는 모든 태그 요소를 내부 노드로 처리합니다. 텍스트 타입 노드의 추가 노드가 *name*, *price*, *symbol*, *shares* 노드 값에 대해 생성됩니다. 그런 다음 이 텍스트 노드는 *name*, *price*, *symbol*, *shares* 노드의 자식으로 나타납니다.

각 노드는 XML 문서 컴포넌트의 *DocumentElement* 속성 값인 루트 노드에서 시작하여 *IXMLNode* 인터페이스를 통해 액세스됩니다.

노드 값 사용

IXMLNode 인터페이스가 있으면 *IsTextElement* 속성을 확인하여 노드가 내부 노드인지 리프 노드인지 확인할 수 있습니다.

- 리프 노드이면 *Text* 속성을 사용하여 노드 값을 읽거나 설정할 수 있습니다.
- 내부 노드이면 *ChildNodes* 속성을 사용하여 자식 노드에 액세스할 수 있습니다.

예를 들면 위의 XML 문서를 사용하여 다음과 같이 Inprise의 주가를 읽을 수 있습니다.

```
InpriseStock := XMLDocument1.DocumentElement.ChildNodes[0];
Price := InpriseStock.ChildNodes['price'].Text;
```

노드 속성 사용

노드에 속성이 있으면 *Attributes* 속성을 사용할 수 있습니다. 기존 속성 이름을 지정하여 속성 값을 읽거나 변경할 수 있습니다. 다음과 같이 *Attributes* 속성을 설정할 때 새로운 속성을 지정하여 추가할 수 있습니다.

```
InpriseStock := XMLDocument1.DocumentElement.ChildNodes[0];
InpriseStock.ChildNodes['shares'].Attributes['type'] := 'common';
```

자식 노드 추가 및 삭제

AddChild 메소드를 사용하여 자식 노드를 추가할 수 있습니다. *AddChild*는 XML 문서에서 태그 요소에 해당하는 새 노드를 생성합니다. 이러한 노드를 요소 노드라고 합니다.

새 요소 노드를 생성하려면 새 태그에 나타나는 이름을 지정하고 옵션으로 새 노드를 나타낼 위치를 지정합니다. 예를 들어 다음 코드는 앞의 문서에 새로운 주식 목록을 추가합니다.

```
var
  NewStock: IXMLNode;
  ValueNode: IXMLNode;
begin
  NewStock := XMLDocument1.DocumentElement.AddChild('stock');
  NewStock.Attributes['exchange'] := 'NASDAQ';
  ValueNode := NewStock.AddChild('name');
  ValueNode.Text := 'Cisco Systems'
  ValueNode := NewStock.AddChild('price');
  ValueNode.Text := '62.375';
  ValueNode := NewStock.AddChild('symbol');
  ValueNode.Text := 'CSCO';
  ValueNode := NewStock.AddChild('shares');
  ValueNode.Text := '25';
end;
```

*AddChild*의 오버로드된 버전을 사용하여 태그 이름이 정의된 네임스페이스(namespace) URI를 지정할 수 있습니다.

ChildNodes 속성의 메소드를 사용하여 자식 노드를 삭제할 수 있습니다. *ChildNodes*는 자식 노드를 관리하는 *IXMLNodeList* 인터페이스입니다. *Delete* 메소드를 사용하여 위치나 이름으로 식별되는 자식 노드를 하나 삭제할 수 있습니다. 예를 들어 다음 코드는 앞의 문서에 나열된 마지막 주식을 삭제합니다.

```
StockHoldings := XMLDocument1.DocumentElement;
StockHoldings.ChildNodes.Delete(StockHoldings.ChildNodes.Count - 1);
```

데이터 바인딩 마법사로 XML 문서 요약

해당 문서의 노드를 알 수 있는 *IXMLNode* 인터페이스와 *TXMLDocument* 컴포넌트만으로 XML 문서를 사용할 수도 있고, *TXMLDocument*를 사용하지 않고 DOM 인터페이스만 사용하여 XML 문서를 처리할 수도 있습니다. XML 데이터 바인딩 마법사를 사용하면 훨씬 간단하고 읽기 쉬운 코드를 작성할 수 있습니다.

데이터 바인딩 마법사는 XML 스키마 또는 데이터 파일을 가져온 다음 매핑하는 인터페이스 집합을 생성합니다. 예를 들어 XML 데이터의 경우 다음과 같습니다.

```
<customer id=1>
  <name>Mark</name>
  <phone>(831) 431-1000</phone>
</customer>
```

데이터 바인딩 마법사는 구현할 클래스와 함께 다음과 같은 인터페이스를 생성합니다.

```
ICustomer = interface(IXMLNode)
  property id: Integer read Getid write Setid;
  property name: DOMString read Getname write Setname;
  property phone: DOMString read Getphone write Setphone;
  function Getid: Integer;
  function Getname: DOMString;
  function Getphone: DOMString;
  procedure Setid(Value: Integer);
  procedure Setname(Value: DOMString);
  procedure Setphone(Value: DOMString);
end;
```

모든 자식 노드는 속성 이름이 자식 노드의 태그 이름과 일치하고 속성 값이 자식 노드의 인터페이스(자식이 내부 노드인 경우)이거나 (리프 노드에 대한) 자식 노드의 값인 속성에 매핑됩니다. 또한 모든 노드 속성(attribute)은 속성(property) 이름이 속성(attribute) 이름이고 속성(property) 값이 속성(attribute) 값인 속성에 매핑됩니다.

마법사는 XML 문서의 태그 요소에 대한 인터페이스와 구현 클래스를 생성하는 것 이외에 루트 노드에 대한 인터페이스를 얻기 위한 전역 함수를 생성합니다. 예를 들어 앞의 XML이 루트 노드에 <Customers> 태그가 있는 문서에서 유래된 경우 데이터 바인딩 마법사는 다음과 같은 전역 루틴을 생성합니다.

```
function GetCustomers(XMLDoc: TXMLDocument): ICustomers;
```

생성된 인터페이스를 사용하면 XML 문서의 구조를 더 직접적으로 반영하기 때문에 사용자의 코드가 단순해집니다. 예를 들어 다음과 같은 코드를 작성하는 대신

```
CustName := XMLDocument1.DocumentElement.ChildNodes[0].ChildNodes['name'].Value;
```

사용자의 코드는 다음과 같을 것입니다.

```
CustName := GetCustomers(XMLDocument1)[0].Name;
```

데이터 바인딩 마법사에서 생성된 인터페이스는 모두 *IXMLNode*에서 파생된다는 점에 유의하십시오. 이는 데이터 바인딩 마법사를 사용하지 않을 때와 동일한 방식으로 자식 노드를 추가하고 삭제할 수 있다는 것을 의미합니다(9-5페이지의 "자식 노드 추가 및 삭제" 참조). 또한 자식 노드가 반복적인 요소이면(노드의 모든 자식이 동일한 타입인 경우) 부모 노드에는 차후의 반복을 추가하기 위한 *Add* 및 *Insert* 메소드가 제공됩니다. 생성할 노드의 타입을 지정할 필요가 없기 때문에 *AddChild*를 사용하는 것보다 이 메소드를 사용하는 것이 더 편리합니다.

XML 데이터 바인딩 마법사 사용

다음과 같은 방법으로 데이터 바인딩 마법사를 사용합니다.

1 File|New를 선택하고 New 페이지에서 XML Data Binding 아이콘을 선택합니다.

2 XML 데이터 바인딩 마법사가 나타납니다.

3 마법사의 첫 페이지에서 XML 문서를 지정하거나 인터페이스를 생성하려는 스키마를 지정합니다. 예제 XML 문서, DTD 파일(.dtd), Reduced XML Data 파일(.xdr) 또는 XML 스키마 파일(.xsd)을 지정할 수 있습니다.

4 Options 버튼을 클릭하여 마법사에서 인터페이스와 구현 클래스 생성 시 사용하려는 이름 지정 방식 및 파스칼 데이터 타입에 대해 스키마에서 정의된 타입의 기본 매핑을 지정합니다.

5 마법사의 두 번째 페이지로 이동합니다. 이 페이지에서 문서나 스키마의 모든 노드 타입에 대한 상세한 정보를 제공할 수 있습니다. 문서의 노드 타입을 모두 보여 주는 트리 뷰가 왼쪽에 있습니다. 복잡한 노드(자식을 갖는 노드)의 경우 트리 뷰를 확장하여 자식 요소 역시 표시할 수 있습니다. 트리 뷰에서 노드를 선택하면 대화 상자의 오른쪽에 노드에 대한 정보가 나타나고 사용자는 마법사에서 노드를 처리하는 방법을 지정할 수 있습니다.

   - Source Name 컨트롤은 XML 스키마에 있는 노드 타입의 이름을 표시합니다.
   - Source Type 컨트롤은 XML 스키마에 지정된 노드 값의 타입을 표시합니다.
   - Documentation 컨트롤은 스키마에 주석을 첨가하여 사용자가 노드의 사용법이나 용도를 설명할 수 있게 합니다.
   - 마법사에서 선택한 노드에 대한 코드를 생성할 경우(즉 마법사에서 인터페이스 및 구현 클래스를 생성하는 복잡한 타입일 경우 또는 마법사에서 복잡한 타입의 인터페이스에 속성을 생성하는 복잡한 타입의 자식 요소 중 하나인 경우) Generate Binding 체크 박스를 사용하여 마법사에서 노드에 대한 코드를 생성할지 여부를 지정할 수 있습니다. Generate Binding을 선택 해제한 경우 마법사에서는 복잡한 타입에 대한 인터페이스나 구현 클래스를 생성하지 않거나 자식 요소 또는 속성에 대해서 부모 인터페이스에 속성을 만들지 않습니다.
   - Binding Options 섹션을 통해 선택한 요소에 대해 마법사에서 생성한 코드에 영향을 줄 수 있습니다. 어떤 노드에서는 Identifier Name(생성된 인터페이스나 속성의 이름)을 지정할 수 있습니다. 또한 인터페이스의 경우 문서의 루트 노드를 나타내도록 지시해야 합니다. 속성을 나타내는 노드의 경우 속성의 타입을 지정할 수 있고 속성이 인터페이스가 아니면 읽기 전용 속성 여부를 지정할 수 있습니다.

6 일단 마법사에서 각 노드에 대해 생성하려는 코드를 지정했으면 세 번째 페이지로 이동합니다. 이 페이지에서 마법사가 코드를 생성하는 방식을 제어하는 일부 전역 옵션을 선택할 수 있고 생성될 코드를 미리 볼 수 있습니다. 마지막으로 나중에 사용하기 위해 선택 사항을 저장하는 방식을 마법사에게 지시할 수 있습니다.

   - 마법사에서 생성하는 코드를 미리 보려면 Binding Summary 목록에 있는 인터페이스를 선택하고 Code Preview 컨트롤에서 결과 인터페이스 정의를 봅니다.
   - Data Binding Settings를 사용하여 마법사에서 선택 사항을 저장하는 방법을 지시합니다. 설정을 문서에 연결된 스키마 파일(대화 상자의 첫 페이지에서 지정한 스키마 파일)에 주석으로 저장하거나 마법사에서만 사용되는 독립적인 스키마 파일의 이름을 지정할 수 있습니다.

7 Finish를 클릭하면 데이터 바인딩 마법사는 XML 문서의 모든 노드 타입에 대한 인터페이스 및 구현 클래스를 정의하는 새로운 유닛을 생성합니다. 또한 *TXMLDocument* 객체를 가져와서 데이터 계층 구조의 루트 노드에 대한 인터페이스를 반환하는 전역 함수를 만듭니다.

XML 데이터 바인딩 마법사가 생성하는 코드 사용

일단 마법사가 인터페이스 및 구현 클래스 집합을 생성했으면 이를 사용하여 문서의 구조 또는 사용자가 마법사에 제공한 스키마와 일치하는 XML 문서를 처리할 수 있습니다. Kylix에 들어 있는 기본 제공 XML 컴포넌트만 사용할 때와 마찬가지로 시작점은 컴포넌트 팔레트의 Internet 페이지에 있는 *TXMLDocument* 컴포넌트입니다.

XML 문서를 사용하려면 다음 단계를 따릅니다.

1 *TXMLDocument* 컴포넌트를 폼이나 데이터 모듈에 놓습니다.

2 *FileName* 속성을 설정하여 *TXMLDocument*를 XML 문서에 바인딩합니다. (다른 접근 방법으로는 런타임 시 *XML* 속성을 설정하여 XML의 문자열을 사용할 수 있습니다.)

3 코드에서 XML 문서의 루트 노드에 대한 인터페이스를 얻기 위해 마법사가 생성한 전역 함수를 호출합니다. 예를 들어 XML 문서의 루트 요소가 <StockList> 태그였다면 기본적으로 마법사는 다음과 같은 *IStockListType* 인터페이스를 반환하는 *GetStockListType* 함수를 생성합니다.

```
var
  StockHoldings: IStockHoldingsType;
begin
  StockHoldings := GetStockHoldingsType(XMLDocument1);
```

4 이 인터페이스는 루트 요소의 속성(attribute)에 해당하는 속성(property)뿐만 아니라 문서 루트 요소의 하위 노드에 해당하는 속성(property)을 갖습니다. 이를 사용하여 XML 문서의 계층 구조에서 이동하거나 문서의 데이터를 수정할 수 있습니다.

5 마법사에 의해 생성된 인터페이스를 사용하여 변경 내용을 저장하려면 *TXMLDocument* 컴포넌트의 *SaveToFile* 메소드를 호출하거나 *XML* 속성을 읽습니다.

10

웹 서비스 사용

* 이 장은 영문 Kylix2 개발자 안내서의 28장입니다.

웹 서비스는 WWW와 같은 네트워크 상에서 게시 및 호출할 수 있는 모듈이 자체적으로 들어 있는 애플리케이션입니다. 웹 서비스는 제공하는 서비스에 대한 인터페이스가 잘 정의되어 있습니다.

웹 서비스는 클라이언트와 서버 간의 느슨한 결합(loose coupling)을 허용하도록 디자인됩니다. 즉 서버 구현에서 클라이언트가 특정 플랫폼이나 특정 프로그램 언어만 사용해야 하는 것은 아닙니다. 랭귀지 중립 방식으로 인터페이스를 정의하는 것 외에 여러 통신 메커니즘도 사용할 수 있도록 디자인됩니다.

Kylix의 웹 서비스 지원은 SOAP(Simple Object Access Protocol)를 사용하여 작동하도록 디자인됩니다. SOAP는 분산 환경에서 정보를 교환하기 위한 표준 경량급(lightweight) 프로토콜입니다. XML을 사용하여 원격 프로시저 호출을 인코딩하고 통신 프로토콜로써 HTTP를 주로 사용합니다. SOAP에 대한 자세한 내용은 다음 주소에서 SOAP 사양을 참조하십시오.

```
http://www.w3.org/TR/SOAP/
```

참고 Kylix의 웹 서비스 지원은 SOAP 및 HTTP를 기반으로 하지만 매우 일반적인 프레임워크를 사용하므로 확장하여 다른 인코딩 방식 및 통신 프로토콜을 사용할 수 있습니다.

Kylix의 SOAP 기반 기술은 크로스 플랫폼 분산 애플리케이션의 기초를 형성합니다. CORBA를 사용하는 애플리케이션을 배포할 때처럼 특별한 클라이언트 런타임 소프트웨어를 설치할 필요는 없습니다. 이 기술은 HTTP 메시지를 기반으로 하기 때문에 다양한 시스템에서 광범위하게 사용할 수 있다는 장점이 있습니다. 웹 서비스 지원은 Kylix의 크로스 플랫폼 웹 서버 애플리케이션 아키텍처의 맨 위에서 만들어집니다.

Kylix를 사용하여 웹 서비스를 구현하는 서버와 그러한 서비스를 호출하는 클라이언트를 모두 만들 수 있습니다. Kylix로 서버 및 클라이언트 애플리케이션을 만들면 웹 서비스에 대한 인터페이스를 정의하는 단일 유닛을 공유할 수 있습니다. 또한 SOAP 메시지에 응답하는 웹 서비스를 구현하는 임의의 서버에 대한 Kylix 클라이언트와 임의의 클라이언트가 사용할 수 있는 웹 서비스를 게시하는 Kylix 서버를 작성할 수 있습니다.

클라이언트나 서버를 Kylix를 사용하여 작성하지 않으면 사용 가능한 인터페이스에 관한 정보와 WSDL(Web Service Definition Language) 문서를 사용하여 인터페이스를 호출하는 방법에 관한 정보를 게시하거나 import할 수 있습니다. 서버측에서 애플리케이션은 웹 서비스를 설명하는 WSDL 문서를 게시할 수 있습니다. 클라이언트측에서 마법사는 게시된 WSDL 문서를 import하여 필요한 인터페이스 정의 및 연결 정보를 제공할 수 있습니다.

웹 서비스를 지원하는 서버 작성

Kylix에서 웹 서비스를 지원하는 서버는 호출 가능한 인터페이스(invokable interfaces)를 사용하여 만듭니다. 호출 가능한 인터페이스는 런타임 타입 정보(RTTI)를 포함하도록 컴파일된 인터페이스입니다. 이 RTTI는 클라이언트로부터 들어오는 메소드 호출을 제대로 마샬링(marshaling)하기 위해 호출을 해석할 때 사용합니다.

서버에는 호출 가능한 인터페이스와 이를 구현하는 클래스 외에도 디스패처 및 호출자(invoker)라는 두 개의 컴포넌트가 필요합니다. 디스패처(*THTTPSoapDispatcher*)는 들어오는 SOAP 메시지를 수신하여 호출자(invoker)에게 전달하는 자동 디스패칭 컴포넌트입니다. 호출자(*THTTPSoapPascalInvoker*)는 SOAP 메시지를 해석하고, 호출 가능한 인터페이스를 식별하고, 해당 호출을 실행하고, 응답 메시지를 어셈블합니다.

참고 *THTTPSoapDispatcher* 및 *THTTPSoapPascalInvoker*는 SOAP 요청을 포함하는 HTTP 메시지에 응답하도록 디자인되었습니다. 하지만 기본 아키텍처는 매우 보편적이어서 다른 디스패처 및 호출자 컴포넌트의 대체 컴포넌트를 사용하여 다른 프로토콜도 지원할 수 있습니다.

일단 호출 가능한 인터페이스 및 구현 클래스를 등록하면 디스패처와 호출자는 HTTP 요청 메시지의 SOAP Action 헤더에 있는 인터페이스들을 식별하는 모든 메시지를 자동으로 처리합니다.

웹 서비스 서버 구축

다음 단계에 따라 웹 서비스를 구현하는 서버 애플리케이션을 만듭니다.

1. 웹 서비스를 구성하는 인터페이스를 정의합니다. 인터페이스 정의는 반드시 호출 가능한 인터페이스(invokable interfaces)여야 합니다. 인터페이스 정의는 구현 클래스가 포함된 유닛이 아닌 별도의 유닛에 만드는 것이 좋습니다. 인터페이스를 정의하는 유닛을 별도의 유닛으로 분리하면 서버 및 클라이언트 애플리케이션 모두 인터페이스를 포함할 수 있습니다. 이 유닛의 초기화 섹션에 인터페이스를 등록하는 코드를 추가합니다. 호출 가능한 인터페이스 작성 및 등록에 대한 자세한 내용은 10-3페이지의 "호출 가능한 인터페이스 정의"를 참조하십시오.
2. 인터페이스가 스칼라가 아닌 복잡한 타입을 사용하는 경우 이 타입이 제대로 마샬링될 수 있는지 확인해야 합니다. 복잡한 타입을 사용한다면 웹 서비스 애플리케이션에서는 구조를 설명하는 런타임 타입 정보(RTTI)가 포함된 특별한 객체를 사용해야 이 타입을 처리할 수 있습니다. 복잡한 타입을 표현하는 객체를 생성하거나 등록에 대한 자세한 내용은 10-5페이지의 "호출 가능한 인터페이스에서 복잡한 타입 사용"을 참조하십시오.

3 1단계에서 정의한 호출 가능한 인터페이스를 구현하는 클래스를 정의하고 구현합니다. 또한 각 구현 클래스에 대해 클래스를 인스턴스화하는 팩토리 프로시저를 만들어야 할 경우도 있습니다. 이 유닛의 초기화 섹션에 구현 클래스를 등록하는 코드를 추가합니다. 이 프로세스는 10-6페이지의 "구현 및 등록"에서 다룹니다.

4 SOAP 요청을 실행하려고 할 때 애플리케이션에서 예외가 발생하면 예외는 메소드 호출 결과 대신 SOAP 폴트 패킷(fault packet)으로 자동으로 인코딩되어 반환됩니다. 간단한 오류 메시지 외에 정보를 추가로 전달하려는 경우 인코딩되어 클라이언트에게 전달되는 사용자 고유의 예외 클래스를 만들 수 있습니다. 이 내용은 10-7페이지의 "웹 서비스에 대한 사용자 지정 예외 클래스 생성"에 설명되어 있습니다.

5 File|New를 선택하고 WebServices 페이지에서 Soap Server 애플리케이션 아이콘을 더블 클릭합니다. 웹 서비스를 구현하려는 웹 서버 애플리케이션의 유형을 선택합니다. 다른 유형의 웹 서비스 애플리케이션에 대한 내용은 6-6페이지의 "웹 서버 애플리케이션의 유형"을 참조하십시오.

6 마법사는 다음과 같은 세 가지 컴포넌트가 포함된 새 웹 서비스 애플리케이션을 생성합니다.

- 호출자 컴포넌트(*THTTPSOAPPascalInvoker*). 호출자는 SOAP 메시지와 1단계에서 등록한 인터페이스의 메소드 사이를 변환합니다.
- 디스패처 컴포넌트(*THTTPSoapDispatcher*). 디스패처는 들어오는 SOAP 메시지에 자동으로 응답하고 이를 호출자에게 전달합니다. *WebDispatch* 속성을 사용하여 애플리케이션이 응답하는 HTTP 요청 메시지를 식별할 수 있습니다. 이렇게 하려면 애플리케이션을 가리키는 URL의 경로 부분을 나타내는 *PathInfo* 속성 및 요청 메시지에 대한 메소드 헤더를 나타내는 *MethodType* 속성을 설정해야 합니다.
- WSDL 게시자(*TWSDLHTMLPublish*). WSDL 게시자는 인터페이스와 호출 방법을 설명하는 WSDL 문서를 게시합니다. 이렇게 하면 Kylix를 사용하여 작성되지 않은 클라이언트가 웹 서비스 애플리케이션을 호출할 수 있습니다. WSDL 게시자에 대한 자세한 내용은 10-7페이지의 "웹 서비스 애플리케이션에 대한 WSDL 문서 생성"을 참조하십시오.

7 Project|Add To Project를 선택하고 1~4단계에서 만든 유닛을 웹 서버 애플리케이션에 추가합니다.

호출 가능한 인터페이스 정의

호출 가능한 인터페이스를 만들려면 {$M+} 컴파일러 옵션으로 인터페이스를 컴파일하면 됩니다. 호출 가능한 인터페이스의 자손도 호출할 수 있습니다. 하지만 호출 가능한 인터페이스가 호출 불가능한 다른 인터페이스의 자손이라면 웹 서비스 서버의 클라이언트는 호출 가능한 인터페이스와 그 자손에서 정의한 메소드만 호출할 수 있습니다. 호출 불가능한 조상에서 상속된 메소드는 타입 정보와 함께 컴파일되지 않으므로 클라이언트에서 호출할 수 없습니다.

Kylix는 호출 가능한 기본 인터페이스인 *IInvokable*을 정의하며 Iinvokable 인터페이스는 웹 서비스 서버에 의해 클라이언트에게 공개되는 인터페이스의 기초로 사용됩니다. *IInvokable*은 {$M+} 컴파일러 옵션으로 컴파일하면 그 자손과 RTTI를 포함하도록 컴파일된다는 것을 제외하면 기본 인터페이스(*IInterface*)와 동일합니다.

예를 들어 다음 코드는 숫자 값을 인코딩 및 디코딩하는 두 개의 메소드가 포함된 호출 가능한 인터페이스를 정의합니다.

```
IEncodeDecode = interface(IInvokable)
  ['{C527B88F-3F8E-1134-80e0-01A04F57B270}']
    function EncodeValue(Value: Integer): Double; stdcall;
    function DecodeValue(Value: Double): Integer; stdcall;
end;
```

웹 서비스 애플리케이션이 호출 가능한 인터페이스를 사용하기 전에 인터페이스를 호출 레지스트리에 등록해야 합니다. 서버에서 호출 레지스트리 항목은 호출자 컴포넌트(*THTTPSOAPPascalInvoker*)가 인터페이스 호출을 실행하기 위해 사용하는 구현 클래스를 식별할 수 있도록 해줍니다. 클라이언트 애플리케이션에서 호출 레지스트리 항목은 컴포넌트가 호출 가능한 인터페이스를 식별하고 이를 호출하는 방법에 대한 정보를 찾을 수 있도록 해줍니다.

인터페이스를 정의하는 유닛의 초기화 섹션에 인터페이스를 호출 레지스트리에 등록하는 코드를 추가합니다. 호출 레지스트리에 액세스하려면 유닛의 uses 절에 InvokeRegistry 유닛을 추가합니다. InvokeRegistry 유닛은 등록된 호출 가능한 모든 인터페이스의 카탈로그, 구현 클래스, 구현 클래스의 인스턴스를 만드는 팩토리를 메모리에 유지하는 전역 함수인 *InvRegistry*를 선언합니다.

작성이 끝났으면 인터페이스를 정의하는 유닛은 다음과 유사할 것입니다.

```
unit EncodeDecode;

interface
type

IEncodeDecode = interface(IInvokable)
  ['{C527B88F-3F8E-1134-80e0-01A04F57B270}']
    function EncodeValue(Value: Integer): Double; stdcall;
    function DecodeValue(Value: Double): Integer; stdcall;
end;

implementation
uses  InvokeRegistry;

initialization
InvRegistry.RegisterInterface(TypeInfo(IEncodeDecode));
end.
```

웹 서비스의 인터페이스에는 가능한 모든 웹 서비스에 있는 모든 인터페이스 중에 이를 식별하는 네임스페이스가 있어야 하기 때문에 인터페이스를 등록할 때 호출 레지스트리는 인터페이스에 대한 네임스페이스를 자동으로 생성합니다. 기본 네임스페이스는 애플리케이션(*AppNamespacePrefix* 변수), 인터페이스 이름, 이를 정의한 유닛의 이름을 고유하게 식별하는 문자열로부터 만들어집니다.

팁 인터페이스 정의는 구현 클래스가 포함된 유닛이 아닌 별도의 유닛에 만드는 것이 좋습니다. 인터페이스를 정의하는 유닛을 별도의 유닛으로 분리하면 서버 및 클라이언트 애플리케이션 모두 인터페이스를 포함할 수 있습니다. 생성된 네임스페이스에는 인터페이스를 정의한 유닛의 이름이 포함되기 때문에 클라이언트 및 서버 애플리케이션이 모두 동일한 유닛을 공유하면 자동으로 동일한 네임스페이스를 사용할 수 있습니다.

호출 가능한 인터페이스에서 복잡한 타입 사용

호출자 컴포넌트(*THTTPSOAPPascalInvoker*)는 호출 가능한 인터페이스의 스칼라 타입을 마샬링하는 방법을 자동으로 인식합니다. 복잡한 타입이 원격 가능 타입 레지스트리(아래 참조)에 등록되어 있으면 동적 배열도 처리할 수 있습니다. 그러나 정적 배열, 인터페이스, 레코드, 셋(sets), 클래스와 같이 더 복잡한 타입의 데이터는 전송할 수 없습니다. 그 대신 이러한 복잡한 타입은 호출자가 SOAP 스트림의 데이터와 타입 값 사이의 변환에 사용할 수 있는 런타임 타입 정보(RTTI)를 포함하는 클래스로 매핑해야 합니다. 그런 다음 호출 가능한 인터페이스에서 정적 배열, 레코드, 셋 또는 원격 불가능 클래스보다는 RTTI를 포함하는 클래스를 사용합니다.

호출 가능한 인터페이스에 복잡한 데이터 타입을 나타내기 위해 클래스를 정의하는 경우 기본(base) 클래스로 *TRemotable*을 사용합니다. 예를 들어 레코드를 매개변수로 전달할 경우에는 레코드의 모든 멤버가 새 클래스에서 published 속성인 *TRemotable* 자손을 대신 정의합니다.

새로운 *TRemotable* 자손의 값이 오브젝트 파스칼 스칼라 타입과 일치하지 않는 WSDL 문서의 스칼라 타입에 해당하면 기본(base) 클래스로 *TRemotableXS*를 대신 사용합니다. *TRemotableXS*는 새로운 클래스와 문자열 표현 간을 변환하는 메소드가 2개 있는 *TRemotable* 자손입니다. *XSToNative* 및 *NativeToXS* 메소드를 오버라이드하여 이러한 메소드를 제공합니다.

TRemotable 자손을 정의하는 유닛의 초기화 섹션에서 이 클래스를 원격 가능 타입 레지스트리(remotable type registry)에 등록해야 합니다. InvokeRegistry 유닛을 uses 절에 추가하여 원격 가능 타입 레지스트리에 액세스합니다. InvokeRegistry 유닛은 원격 가능 타입 레지스트리를 반환하는 두 가지 전역 함수인 *RemClassRegistry* 및 *RemTypeRegistry*를 선언합니다. 두 개의 함수 중 하나를 사용하여 원격 가능 클래스를 등록하는 객체를 얻을 수 있습니다(두 함수 모두 동일한 레지스트리를 참조합니다). 예를 들어 XSBuiltIns 유닛의 다음과 같은 줄에서 *TDateTime* 값을 나타내는 *TRemotable* 자손인 *TXSDateTime*을 등록합니다.

```
RemClassRegistry.RegisterXSClass(TXSDateTime, XMLSchemaNameSpace, 'dateTime', '',True);
```

첫 번째 매개변수는 *TRemotable* 자손의 이름입니다. 두 번째 매개변수는 새 클래스의 네임스페이스를 고유하게 식별하는 URI(Uniform Resource Identifier)입니다. 이 매개변수가 빈 문자열이면 레지스트리에서 URI를 생성할 수 있습니다. 세 번째와 네 번째 매개변수는 클래스가 나타내는 데이터 타입의 이름입니다. 이 매개변수가 문자열이면 레지스트리는 간단하게 클래스 이름을 사용합니다. 마지막 매개변수는 클래스 인스턴스의 값이 문자열로 전송될 수 있는지 여부(*XSToNative* 및 *NativeToXS* 메소드 구현 여부)를 나타냅니다.

팁 *TRemotable* 자손은 호출 가능한 인터페이스를 선언하고 등록하는 유닛을 비롯한 나머지 서버 애플리케이션이 아닌 별도의 유닛에 구현하고 등록하는 것이 좋습니다. 자손을 구현하고 등록하는 유닛을 별도의 유닛으로 분리하면 서버 및 클라이언트 모두 사용자의 타입을 정의하는 유닛을 사용할 수 있고 둘 이상의 인터페이스에 대한 타입을 사용할 수 있습니다.

매개변수에 동적 배열, 열거 타입 또는 부울 타입을 사용한 경우 이를 나타내는 클래스를 생성할 필요는 없지만 원격 가능 타입 레지스트리에 등록해야 합니다. 예를 들어 인터페이스가 다음과 같은 타입을 사용하는 경우

```
type
  TDateTimeArray = array of TXSDateTime;
```

동적 배열을 선언한 유닛의 초기화 섹션에 다음과 같이 등록을 추가해야 합니다.

```
RemTypeRegistry.RegisterXSInfo(TypeInfo(TDateTimeArray), MyNameSpace, 'DTarray', False);
```

클래스 참조보다는 동적 배열의 타입 정보에 대한 포인터를 사용하는 첫 번째 매개변수를 제외하면 이 매개변수들은 *RegisterXSClass*에서 사용하는 것과 동일합니다.

구현 및 등록

호출 가능 인터페이스를 구현하는 가장 간단한 방법은 *TInvokableClass*의 자손 클래스를 만드는 것입니다. 사용자가 지원하려는 호출 가능한 인터페이스(및 해당 메소드)를 포함하는 클래스 선언을 추가한 다음 *Ctrl+Shift+C*를 입력하여 class completion을 호출합니다. 빈 메소드가 유닛의 구현 섹션에 나타납니다.

예를 들어 "호출 가능한 인터페이스 정의" 위에 선언된 인터페이스를 구현하는 구현 클래스에 대한 선언은 다음과 같습니다.

```
TEncodeDecode = class(TInvokableClass, IEncodeDecode)
protected
  function EncodeValue(Value: Integer): Double; stdcall;
  function DecodeValue(Value: Double): Integer; stdcall;
end;
```

이 클래스를 선언하는 유닛의 구현 섹션에서 *EncodeValue* 및 *DecodeValue* 메소드를 작성하십시오.

일단 구현 클래스를 만들었으면 이 클래스를 호출 레지스트리(invocation registry)에 등록해야 합니다. 호출 레지스트리는 등록된 인터페이스를 구현하는 클래스를 식별하고 호출자가 인터페이스를 호출해야 할 때 호출자 컴포넌트에서 인터페이스를 사용할 수 있도록 합니다. 구현 클래스를 등록하려면 전역 *InvRegistry* 함수가 반환한 객체의 *RegisterInvokableClass* 메소드 호출을 구현 유닛의 초기화 섹션에 추가합니다.

```
InvRegistry.RegisterInvokableClass(TEncodeDecode);
```

*TInvokableClass*의 자손이 아닌 구현 클래스를 만들 수도 있습니다. 하지만 이 경우에는 호출 레지스트리가 클래스의 인스턴스를 만들기 위해 호출하는 팩토리 프로시저를 제공해야 합니다.

팩토리 프로시저는 *TCreateInstanceProc* 타입이어야 합니다. 이는 구현 클래스의 인스턴스를 반환합니다. 팩토리 프로시저가 새로운 인스턴스를 만들면 호출 레지스트리에서 객체 인스턴스를 명시적으로 해제하지 않는 것처럼 인터페이스의 참조 카운트가 0으로 내려갈 때 객체가 저절로 해제되어야 합니다. 대안으로서 팩토리 프로시저는 모든 호출자가 공유하는 전역 인스턴스에 대한 참조를 반환할 수 있습니다. 다음 코드는 후자의 방법을 설명합니다.

```
procedure CreateEncodeDecode(out obj: TObject);
begin
  if FEncodeDecode = nil then
  begin
    FEncodeDecode := TEncodeDecode.Create;
    {save a reference to the interface so that the global instance doesn't free itself }
    FEncodeDecodeInterface := FEncodeDecode as IEncodeDecode;
  end;
  obj := FEncodeDecode; { return global instance }
end;
```

팩토리 프로시저를 사용하는 경우 *RegisterInvokableClass* 메소드에 대한 두 번째 매개변수로 팩토리 프로시저를 제공합니다.

```
InvRegistry.RegisterInvokableClass(TEncodeDecode, CreateEncodeDecode);
```

웹 서비스에 대한 사용자 지정 예외 클래스 생성

SOAP 요청을 실행하려는 동안 웹 서비스 애플리케이션에서 예외가 발생하면 메소드 호출의 결과 대신 예외 정보가 SOAP 폴트 패킷으로 자동으로 인코딩되어 반환됩니다. 그러면 클라이언트 애플리케이션이 예외를 발생시킵니다.

기본적으로 클라이언트 애플리케이션은 SOAP 폴트 패킷의 오류 메시지와 함께 일반 예외(*Exception*)를 발생시킵니다. 하지만 *ERemotableException*의 자손인 예외 클래스를 사용하여 추가적인 예외 정보를 전송할 수 있습니다. 예외 클래스에 추가한 published 속성 값은 클라이언트가 동등한 예외를 발생시킬 수 있도록 SOAP 폴트 패킷에 포함됩니다.

ERemotableException 자손을 사용하려면 원격 가능 타입 레지스트리에 등록해야 합니다. 즉 *ERemotableException* 자손을 정의하는 유닛에서 uses 절에 InvokeRegistry 유닛을 추가하고 전역 *RemTypeRegistry* 함수가 반환하는 객체의 *RegisterXSClass* 메소드 호출을 추가해야 합니다.

클라이언트가 사용자의 *ERemotableException* 자손을 정의하고 등록하는 동일한 유닛을 사용한다면 SOAP 폴트 패킷을 수신할 때 클라이언트는 SOAP 폴트 패킷의 값에 설정된 모든 속성과 함께 적절한 예외 클래스의 인스턴스를 자동으로 발생시킵니다.

웹 서비스 애플리케이션에 대한 WSDL 문서 생성

호출 가능한 인터페이스, 복잡한 타입 정보를 나타내는 클래스, 사용자의 원격 가능 예외를 정의하고 등록하는 동일한 유닛을 Kylix 클라이언트 애플리케이션에 포함시키면 웹 서비스를 호출할 수 있습니다. 이렇게 하려면 웹 서비스 애플리케이션을 설치한 URL을 제공하면 됩니다.

여러 종류의 클라이언트들이 웹 서비스를 호출하도록 만들 수 있습니다. 예를 들어 클라이언트 중에는 Kylix로 작성되지 않은 클라이언트도 있을 수 있습니다. 여러 버전의 서버 애플리케이션을 배포하는 경우 서버에 대해 하드 코딩된 단일 URL을 사용하기보다는 클라이언트가 서버 위치를 동적으로 찾도록 할 수도 있습니다. 이 경우 웹 서비스의 타입과 인터페이스를 설명하는 WSDL 문서를 이를 호출하는 방법에 관한 정보와 함께 게시할 수도 있습니다.

웹 서비스를 설명하는 WSDL 문서를 게시하기 위해서는 단지 웹 모듈에 *TWSDLHTMLPublish* 컴포넌트를 추가하면 됩니다. *TWSDLHTMLPublish*는 웹 서비스에 대한 WSDL 문서의 목록을 요청하는 수신 메시지에 자동으로 응답하는 자동 디스패칭 컴포넌트입니다. WSDL 문서의 목록을 액세스하는 데 URL 클라이언트의 경로 정보를 반드시 사용하도록 지정하려면 *WebDispatch* 속성을 사용합니다. 그러면 웹 브라우저는 WebDispatch 속성의 경로 앞에 있는 서버 애플리케이션의 위치를 구성하는 URL을 지정하여 WSDL 문서의 목록을 요청할 수 있습니다. 이 URL은 다음과 유사한 형태일 것입니다.

```
http://www.myco.com/MyService.dll/WSDL
```

팁 물리적 WSDL 파일이 필요하면 웹 브라우저에 WSDL 문서를 표시한 다음 이를 저장하여 WSDL 문서 파일을 생성할 수 있습니다.

웹 서비스를 구현하는 동일한 애플리케이션의 WSDL 문서를 반드시 게시할 필요는 없습니다. WSDL 문서를 게시하는 간단한 애플리케이션을 만들려면 구현 객체를 포함하는 유닛을 생략하고 호출 가능한 인터페이스, 복잡한 타입을 나타내는 원격 가능 클래스, 모든 원격 가능 예외를 정의하고 등록하는 유닛만 포함시킵니다.

기본적으로 WSDL 문서를 게시할 경우 경로가 달라도 WSDL 문서를 게시했던 곳과 동일한 URL에서 서비스가 사용 가능함을 나타냅니다. 여러 버전의 웹 서비스 애플리케이션을 배포하는 경우 또는 웹 서비스를 구현하는 것 이외에 다른 애플리케이션의 WSDL 문서를 게시하는 경우 웹 서비스의 위치에 관한 업데이트된 정보를 포함하는 WSDL 문서를 변경해야 합니다.

URL을 변경하기 위해 WSDL 관리자를 사용합니다. 첫 번째 단계는 WSDL 관리자를 활성화하는 것입니다. *TWSDLHTMLPublish* 컴포넌트의 *AdminEnabled* 속성을 *True*로 설정하면 WSDL 관리자가 활성화됩니다. 그런 다음 브라우저를 사용하여 WSDL 문서의 목록을 표시하면 문서를 관리할 버튼도 함께 포함됩니다. WSDL 관리자를 사용하여 웹 서비스 애플리케이션을 배포했던 위치(URL)를 지정하십시오.

웹 서비스용 클라이언트 작성

Kylix는 SOAP 기반 바인딩을 사용하는 웹 서비스 호출에 대한 클라이언트측 지원을 제공합니다. Kylix로 작성된 서버 또는 WSDL 문서에 있는 웹 서비스를 정의하는 다른 서버에서 이러한 웹 서비스를 제공할 수 있습니다.

서버가 Kylix로 작성되지 않은 경우 먼저 서버를 설명하는 WSDL 문서를 import할 수 있습니다. 이 과정은 다음 단원에서 설명합니다. 서버가 Kylix를 사용하여 작성되었으면 WSDL 문서를 사용하지 않아도 됩니다. 복잡한 타입을 나타내는 원격 가능 클래스를 정의하는 유닛과 웹 서비스 애플리케이션에서 발생할 수 있는 원격 가능 예외를 정의하는 모든 유닛 외에도 프로젝트에 사용할 호출 가능한 인터페이스를 정의하는 모든 유닛을 추가할 수 있습니다.

참고 Kylix 웹 서비스 서버에서 호출 가능한 인터페이스를 정의하는 유닛 작성에 대한 내용은 10-3페이지의 "호출 가능한 인터페이스 정의"를 참조하십시오. 복잡한 타입에 대한 원격 가능 클래스를 정의하는 유닛 작성에 대한 내용은 10-5페이지의 "호출 가능한 인터페이스에서 복잡한 타입 사용"을 참조하십시오. 원격 가능 예외를 정의하는 유닛 작성에 대한 내용은 10-7페이지의 "웹 서비스에 대한 사용자 지정 예외 클래스 생성"을 참조하십시오.

WSDL 문서 import하기

Kylix로 작성되지 않은 웹 서비스를 사용하기 전에 먼저 서비스를 정의하는 WSDL 문서 또는 XML 스키마 파일을 import해야 합니다. 웹 서비스 importer는 사용해야 할 인터페이스와 타입을 정의하고 등록하는 유닛을 만듭니다.

웹 서비스 importer를 사용하려면 File|New를 선택하고 Web Services 페이지에서 Web Services importer 아이콘을 더블 클릭합니다. 나타나는 대화 상자에서 WSDL 문서 또는 XML 스키마 파일의 이름을 지정하거나 문서가 게시된 URL을 입력합니다. Generate를 클릭하면 importer는 문서에 정의된 작업에 대해 호출 가능한 인터페이스를 정의하고 등록하는 새로운 유닛과 문서가 정의하는 타입에 대해 원격 가능 클래스를 정의하고 등록하는 새로운 유닛을 만듭니다.

WSDL 문서 또는 XML 스키마 파일이 오브젝트 파스칼 키워드이기도 한 식별자를 사용하면 importer는 생성 코드가 컴파일할 수 있도록 해당 이름을 자동으로 조정합니다. 복잡한 타입이 인라인(inline)으로 선언되면 importer는 호출 가능한 인터페이스와 동일한 유닛에서 해당 원격 가능 클래스를 정의하고 등록하는 코드를 추가합니다. 인라인으로 선언되지 않으면 타입이 별도의 유닛에서 정의되고 등록됩니다.

호출 가능 인터페이스 호출

호출 가능 인터페이스를 호출하려면 클라이언트 애플리케이션은 복잡한 타입을 구현하는 모든 원격 가능 클래스 및 호출 가능 인터페이스를 정의하는 유닛을 포함해야 합니다. 서버가 Kylix에서 작성된 경우 서버 애플리케이션이 이러한 인터페이스와 클래스를 정의하고 등록하기 위해 사용하는 유닛과 동일한 유닛이어야 합니다. 호출 가능한 인터페이스 또는 원격 가능 클래스를 등록할 때 이를 고유하게 식별하는 URI(Uniform Resource Identifier)를 알 수 있기 때문에 동일한 유닛을 사용하는 것이 가장 좋습니다. 해당 URI는 인터페이스(또는 클래스)의 이름 및 URI가 정의된 유닛의 이름에서 파생됩니다. 클라이언트 및 서버가 동일한 URI를 사용하여 인터페이스(또는 클래스)를 등록하지 않으면 통신할 수 없습니다. 동일한 유닛을 사용하지 않는 경우 인터페이스 및 구현 클래스를 등록하는 코드는 클라이언트와 서버가 동일한 네임스페이스를 사용하도록 네임스페이스 URI를 명시적으로 지정해야 합니다.

서버가 Kylix에서 작성되지 않았거나 서버에서 사용한 동일한 유닛을 클라이언트에서 사용하지 않는 경우 웹 서비스 importer로 유닛을 만들 수 있습니다.

일단 클라이언트 애플리케이션에 호출 가능한 인터페이스의 선언이 있으면 원하는 인터페이스에 대해 *THTTPRio*의 인스턴스를 만듭니다.

```
X := THTTPRio.Create(nil);
```

그런 다음 서버 인터페이스를 식별하고 서버의 위치를 찾는 데 필요한 정보와 함께 *THTTPRio* 객체를 제공합니다. 이 정보를 제공하는 데는 두 가지 방법이 있습니다.

- 서버가 Kylix에서 작성된 경우 서버의 인터페이스는 인터페이스가 등록될 때 생성된 URI를 기반으로 자동으로 식별됩니다. 서버의 위치를 나타내려면 *URL* 속성을 설정하기만 하면 됩니다. URL의 경로 부분은 서버의 웹 모듈에 있는 디스패처 컴포넌트의 경로와 일치해야 합니다.

```
X.URL := 'http://www.myco.com/MyService.dll/SOAP/';
```

- 서버가 Kylix에서 작성되지 않은 경우 *THTTPRio*는 인터페이스에 대한 URI, Soap Action 헤더에 포함시켜야 하는 정보, WSDL 문서로부터의 서버 위치를 조회해야 합니다. *WSDLLocation*, *Service*, *Port* 속성을 사용하여 찾을 방법을 지정합니다.

```
X.WSDLLocation := 'Cryptography.wsdl';
X.Service := 'Cryptography';
X.Port := 'SoapEncodeDecode';
```

그런 다음 **as** 연산자를 사용하여 *THTTPRio*의 인스턴스를 호출 가능한 인터페이스로 변환할 수 있습니다. 이렇게 함으로써 연결된 인터페이스에 대한 vtable을 메모리에 동적으로 생성하여 인터페이스를 호출할 수 있습니다.

```
InterfaceVariable := X as IEncodeDecode;
Code := InterfaceVariable.EncodeValue(5);
```

*THTTPRio*는 호출 레지스트리를 사용하여 호출 가능한 인터페이스에 대한 정보를 얻습니다. 클라이언트 애플리케이션에 호출 레지스트리가 없거나 또는 호출 가능한 인터페이스가 등록되어 있지 않으면 *THTTPRio*는 메모리 내(in-memory) vtable을 만들 수 없습니다.

11

소켓 사용

* 이 장은 영문 Kylix2 개발자 안내서의 29장입니다.

이 장에서는 TCP/IP 및 관련 프로토콜을 사용하여 다른 시스템과 통신하는 애플리케이션을 만들 수 있는 소켓 컴포넌트에 대해 설명합니다. 소켓을 사용하면 필요한 네트워크 소프트웨어에 대해 자세히 모르더라도 다른 컴퓨터에 연결해서 읽고 쓸 수 있습니다. 소켓은 TCP/IP 프로토콜 방식의 연결을 제공하며 UDP(User Datagram Protocol), XNS(Xerox Network System), Digital의 DECnet 또는 Novell의 IPX/SPX 제품군과 같은 관련 프로토콜과도 잘 작동됩니다.

소켓을 사용하면 다른 시스템에 읽기와 쓰기를 수행하는 네트워크 서버 또는 클라이언트 애플리케이션을 작성할 수 있습니다. 서버 애플리케이션이나 클라이언트 애플리케이션은 일반적으로 HTTP(Hypertext Transfer Protocol) 또는 FTP(File Transfer Protocol)와 같은 한 가지 서비스만 제공하는 애플리케이션입니다. 서버 소켓을 사용하면 서비스를 제공하는 애플리케이션을 서비스를 사용하려는 클라이언트 애플리케이션에 연결할 수 있습니다. 클라이언트 소켓을 사용하면 서비스를 사용하는 애플리케이션을 서비스를 제공하는 서버 애플리케이션에 연결할 수 있습니다.

서비스 구현

소켓은 네트워크 서버 또는 클라이언트 애플리케이션을 작성하는 데 필요한 서비스를 제공합니다. 외부 서버에서도 HTTP 또는 FTP와 같은 대부분의 서비스를 쉽게 사용할 수 있습니다. 일부 서비스는 운영 체제와 함께 제공되는 경우도 있기 때문에 사용자가 직접 작성할 필요가 없습니다. 그러나 서비스가 구현되는 방법을 더 자세히 제어해야 하거나 애플리케이션과 네트워크 통신 간의 보다 긴밀한 통합이 필요한 경우 또는 특정 서비스를 제공하는 서버가 없는 경우에는 사용자가 직접 서버 애플리케이션이나 클라이언트 애플리케이션을 만들 수 있습니다. 예를 들어 분산 데이터셋을 사용하는 경우 다른 시스템의 데이터베이스와 통신하기 위해 레이어를 작성할 수 있습니다.

서비스 프로토콜 이해

네트워크 서버 또는 클라이언트를 작성하기 전에 애플리케이션에서 제공하거나 사용하는 서비스를 이해해야 합니다. 대부분 서비스는 네트워크 애플리케이션에서 지원해야 하는 표준 프로토콜을 사용합니다. HTTP, FTP, finger 또는 time과 같은 표준 서비스 네트워크 애플리케이션을 작성한다면 먼저 다른 시스템과 통신하는 데 사용할 프로토콜을 알고 있어야 합니다. 프로토콜은 제공하거나 사용하려는 특정 서비스에 대한 설명서를 참조하는 것이 좋습니다.

다른 시스템과 통신하는 애플리케이션에 새로운 서비스를 제공하려면 우선 해당 서비스의 서버와 클라이언트의 통신 프로토콜을 어떤 메시지가 전송되는가, 이 메시지들이 어떻게 조정되는가, 정보 인코딩 방식은 무엇인가 등의 관점에서 디자인해야 합니다.

애플리케이션과 통신

대개 네트워크 서버 애플리케이션이나 클라이언트 애플리케이션은 네트워크 소프트웨어와 서비스를 사용하는 애플리케이션 중간에 레이어를 제공합니다. 예를 들어 HTTP 서버는 컨텐트를 제공하고 HTTP 요청 메시지에 응답하는 웹 서버 애플리케이션과 인터넷의 중간에 있습니다.

소켓은 네트워크 서버 또는 클라이언트 애플리케이션과 네트워크 소프트웨어 간에 인터페이스를 제공합니다. 사용자는 직접 개발한 애플리케이션과 이를 사용하는 다른 애플리케이션 간에 인터페이스를 제공해야 합니다. 사용자는 표준 협력 업체 서버(예: Apache)의 API를 복사하거나 사용자 고유의 API를 디자인하고 게시할 수 있습니다.

서비스와 포트

일반적으로 대부분의 표준 서비스는 특정 포트 번호와 연결됩니다. 포트 번호에 대한 자세한 내용은 나중에 설명이 되며 여기에서는 포트 번호를 서비스에 대한 숫자 코드로 간주합니다.

표준 서비스를 구현하려는 경우 *TTcpClient* 소켓 컴포넌트에서 해당 서비스에 대한 포트 번호를 조회할 수 있는 메소드를 제공합니다. 새로운 서비스를 제공하려는 경우 연결된 포트 번호를 /etc/services 파일에 지정할 수 있습니다. 이 서비스 파일은 이름, 포트 번호 및 프로토콜 종류를 비롯한 인터넷 네트워크 서비스를 나열하는 ASCII 파일입니다. 서비스 파일 설정에 대한 자세한 내용은 서비스 man 페이지를 참조하십시오.

소켓 연결 유형

소켓의 연결이 시작되는 방법과 로컬 소켓이 연결되는 대상에 따라 소켓 연결을 다음과 같이 세 가지 기본 유형으로 구분할 수 있습니다.

- 클라이언트 연결
- 리스닝(listening) 연결
- 서버 연결

일단 클라이언트 소켓에 연결되면 서버 연결이 클라이언트 연결을 구별할 수 없습니다. 연결된 두 끝점은 기능이 동일해지며 같은 타입의 이벤트를 수신합니다. 하지만 리스닝 연결에는 끝점이 하나만 있기 때문에 근본적으로 다릅니다.

클라이언트 연결

클라이언트 연결은 로컬 시스템의 클라이언트 소켓을 원격 시스템의 서버 소켓에 연결하여 클라이언트 소켓에 의해 클라이언트 연결이 시작됩니다. 우선 클라이언트 소켓은 연결하려는 서버 소켓을 기술합니다. 그런 다음 서버 소켓을 조회하고 서버를 찾으면 연결을 요청합니다. 서버 소켓은 즉시 연결되지 않을 수도 있습니다. 서버 소켓은 클라이언트 요청의 대기열을 유지 관리하다가 적당한 시간에 연결을 완료합니다. 서버 소켓이 클라이언트 연결을 승인하면 연결하려는 서버 소켓에 대한 모든 정보를 클라이언트 소켓으로 전달하고 클라이언트는 연결을 완료합니다.

리스닝(listening) 연결

서버 소켓은 클라이언트를 찾는 대신 클라이언트 요청을 리스닝하는 수동적인 "반 연결(half connection)"을 구성합니다. 서버 소켓은 대기열을 리스닝 연결에 연결하며 대기열이 클라이언트 연결 요청을 수신하면 요청을 기록합니다. 서버 소켓이 클라이언트 연결 요청을 승인하면 리스닝 연결이 열려 있는 상태에서 다른 클라이언트 요청을 수신할 수 있도록 새 소켓을 구성하여 클라이언트에 연결합니다.

서버 연결

서버 연결은 리스닝 소켓이 클라이언트 요청을 승인할 때 서버 소켓에 의해 구성됩니다. 서버가 연결을 승인하면 클라이언트에 대한 연결을 완료하는 서버 소켓의 정보가 클라이언트에게 전달됩니다. 클라이언트 소켓이 이 정보를 받고 연결을 완료하면 연결이 설정됩니다.

소켓 설명

네트워크 애플리케이션은 소켓을 사용하여 네트워크 상에서 다른 시스템과 통신할 수 있습니다. 각각의 소켓은 네트워크 연결의 끝점이라고 할 수 있으며 끝점에는 다음을 지정하는 주소가 있습니다.

- 소켓이 실행되는 시스템
- 인식하는 인터페이스 타입
- 연결 시 사용하는 포트

소켓 연결에 대한 전체 설명에는 연결의 두 끝점에 있는 소켓 주소가 포함됩니다. IP 주소나 호스트 및 포트 번호를 통해 소켓 양 끝점의 주소를 설명할 수 있습니다.

소켓 연결을 하기 전에 먼저 끝점을 구성하는 소켓을 충분히 설명해야 합니다. 일부 정보는 애플리케이션이 실행 중인 시스템에서 얻을 수 있습니다. 예를 들어 클라이언트 소켓의 로컬 IP 주소는 운영 체제에서 얻을 수 있으므로 설명할 필요가 없습니다.

사용자가 제공해야 할 정보는 사용 중인 소켓 타입에 따라 달라집니다. 클라이언트 소켓은 연결하고자 하는 서버 정보를 제공해야 합니다. 리스닝 서버 소켓은 소켓이 제공하는 서비스를 나타내는 포트 정보를 제공해야 합니다.

호스트 설명

호스트는 소켓이 포함된 애플리케이션을 실행하는 시스템입니다. 다음과 같이 표준 인터넷 도트 표시법에 따라 4개의 숫자(바이트) 값으로 구성된 문자열인 IP 주소를 제공하여 소켓의 호스트 정보를 제공할 수 있습니다.

```
123.197.1.2
```

하나의 시스템이 둘 이상의 IP 주소를 지원할 수도 있습니다.

IP 주소는 기억하기가 어려워서 잘못 입력하는 경우가 많습니다. 이런 경우 주소 대신 호스트 이름을 사용할 수 있습니다. 호스트 이름은 URL(Uniform Resource Locator)에서 자주 볼 수 있는 IP 주소의 별칭이며 다음과 같이 도메인 이름 및 서비스가 포함되어 있는 문자열입니다.

```
http://www.wSite.Com
```

대부분의 인트라넷은 인터넷에 있는 시스템의 IP 주소에 대한 호스트 이름을 제공합니다. 사용자는 명령 프롬프트에서 다음의 명령을 실행하여 IP 주소(이미 존재한다면)와 연결된 호스트 이름을 알 수 있습니다.

```
nslookup IPADDRESS
```

*IPADDRESS*는 사용자가 찾고자 하는 IP 주소입니다. 사용자의 로컬 IP 주소에 호스트 이름이 없어서 이를 갖고자 한다면 네트워크 관리자에게 문의하십시오. 일반적으로 컴퓨터는 이름 *localhost*와 IP 번호 127.0.0.1을 사용하여 참조합니다.

서버 소켓은 호스트를 지정할 필요가 없으며 로컬 IP 주소는 시스템에서 읽을 수 있습니다. 로컬 시스템이 둘 이상의 IP 주소를 지원할 경우 서버 소켓은 모든 IP 주소에서 동시에 클라이언트 요청을 리스닝합니다. 서버 소켓이 연결을 승인하면 클라이언트 소켓은 원격 IP 주소를 제공합니다.

클라이언트 소켓은 호스트 이름이나 IP 주소를 제공함으로써 원격 호스트를 지정해야 합니다.

호스트 이름 또는 IP 주소 선택

대부분의 애플리케이션은 호스트 이름을 사용하여 시스템을 지정합니다. 호스트 이름은 기억하기가 쉽고 입력 오류를 쉽게 확인할 수 있습니다. 또한 서버는 특정 호스트 이름과 연결된 시스템 또는 IP 주소를 변경할 수 있습니다. 호스트 이름을 사용하면 클라이언트 소켓은 호스트 이름이 나타내는 추상적 사이트가 새로운 IP 주소로 이동한 경우에도 해당 사이트를 찾을 수 있습니다.

호스트 이름이 없는 경우 클라이언트 소켓은 IP 주소를 사용하여 서버 시스템을 지정해야 합니다. IP 주소를 제공하면 서버 시스템을 더 빠르게 지정할 수 있습니다. 호스트 이름이 제공되는 경우에는 소켓이 호스트 이름과 연결된 IP 주소를 검색한 후에만 서버 시스템을 찾을 수 있습니다.

포트 사용

IP 주소가 소켓 연결의 다른 끝에 있는 시스템을 찾는 데 필요한 정보를 제공하더라도 해당 시스템의 포트 번호가 필요합니다. 포트 번호가 없으면 시스템은 한 번에 하나의 연결만 구성할 수 있습니다. 포트 번호는 단일 시스템이 각 연결에 별도의 포트 번호를 제공하여 다중 연결을 동시에 호스트할 수 있도록 하는 고유 식별자입니다.

앞에서 이미 포트 번호는 네트워크 애플리케이션에 의해 구현된 서비스에 대한 숫자 코드임을 설명한 바 있습니다. 이는 리스닝 서버 연결을 고정된 포트 번호에서 사용할 수 있도록 하여 클라이언트 소켓이 리스닝 서버 연결을 찾을 수 있도록 하는 규칙입니다. 서버 소켓은 자신이 제공하는 서비스와 연결된 포트 번호를 리스닝합니다. 서버 소켓이 클라이언트 소켓에 대한 연결을 승인하면 다른 임의의 포트 번호를 사용하는 별도의 소켓 연결을 만듭니다. 이와 같은 방법으로 리스닝 연결은 서비스와 연결된 포트 번호를 계속 리스닝할 수 있습니다.

클라이언트 소켓은 다른 소켓이 알고 있을 필요가 없으므로 임의의 로컬 포트 번호를 사용합니다. 클라이언트 소켓은 서버 애플리케이션을 찾을 수 있도록 연결하려는 서버 소켓의 포트 번호를 지정합니다. 일반적으로 이 포트 번호는 원하는 서비스 이름을 지정하여 간접적으로 지정됩니다.

소켓 컴포넌트 사용

Internet 팔레트 페이지에는 사용자의 네트워크 애플리케이션에서 다른 시스템과 연결을 구성할 수 있게 하고 연결을 통해 정보를 읽고 작성할 수 있게 하는 세 가지 소켓 컴포넌트가 있습니다. 세 가지 소켓 컴포넌트는 다음과 같습니다.

- *TcpServer*
- *TcpClient*
- *UdpSocket*

각 소켓 컴포넌트에는 실제 소켓 연결의 끝점을 나타내는 소켓 객체가 연결됩니다. 소켓 컴포넌트가 소켓 객체를 사용하여 소켓 서버 호출을 캡슐화하므로 애플리케이션에서는 연결을 구성하거나 소켓 메시지를 관리하는 세부 사항에 대해 신경 쓸 필요가 없습니다.

사용자 대신 소켓 컴포넌트가 만드는 연결의 세부 사항을 사용자 지정하려는 경우 소켓 객체의 속성, 이벤트 및 메소드를 사용할 수 있습니다.

연결 정보 얻기

클라이언트 또는 서버 소켓에 대한 연결을 완료한 후 연결에 대한 정보를 얻기 위해 사용자의 소켓 컴포넌트와 연결된 클라이언트 또는 서버 소켓 객체를 사용할 수 있습니다. 로컬 클라이언트 또는 서버 소켓에 사용되는 주소와 포트 번호를 결정하는 데 *LocalHost* 및 *LocalPort* 속성을 사용하거나 원격 클라이언트 또는 서버 소켓에 의해 사용되는 주소와 포트 번호를 결정하는 데 *RemoteHost* 및 *RemotePort* 속성을 사용합니다. 호스트 이름과 포트 번호로 유효한 소켓 주소를 만들려면 *GetSocketAddr* 메소드를 사용하십시오. *LookupPort* 메소드를 사용하면 포트 번호를 찾을 수 있습니다. *LookupProtocol* 메

소드를 사용하면 프로토콜 번호를 찾을 수 있습니다. *LookupHostName* 메소드를 사용하면 호스트 시스템의 IP 주소를 기반으로 하는 호스트 이름을 찾을 수 있습니다.

소켓을 오고 가는 네트워크 트래픽을 보려면 *BytesSent* 및 *BytesReceived* 속성을 사용합니다.

클라이언트 소켓 사용

TcpClient 또는 *UdpSocket* 컴포넌트를 폼이나 데이터 모듈에 추가하여 사용자의 애플리케이션을 TCP/IP 또는 UDP 클라이언트로 바꿉니다. 클라이언트 소켓을 사용하면 연결하고자 하는 서버 소켓과 서버가 제공하는 서비스를 지정할 수 있습니다. 원하는 연결 정보를 제공했다면 클라이언트 소켓 컴포넌트를 사용하여 서버에 대한 연결을 완료할 수 있습니다.

각 클라이언트 소켓 컴포넌트는 연결에서의 클라이언트 끝점을 나타내기 위해 하나의 클라이언트 소켓 객체를 사용합니다.

원하는 서버 지정

클라이언트 소켓 컴포넌트에는 사용자가 연결하고자 하는 서버 시스템과 포트를 지정할 수 있는 여러 가지 속성들이 있습니다. *RemoteHost* 속성을 사용하여 호스트 이름 또는 IP 주소로 원격 호스트 서버를 지정합니다.

서버 시스템뿐만 아니라 사용자의 클라이언트 소켓이 연결할 서버 시스템의 포트를 지정해야 합니다. *RemotePort* 속성을 사용하여 대상 서비스의 이름을 지정함으로써 서버 포트 번호를 직접 또는 간접적으로 지정할 수 있습니다.

연결 구성

일단 클라이언트 소켓 컴포넌트의 속성을 설정하여 연결하려는 서버 정보를 제공했으면 *Open* 메소드를 호출하여 런타임 시 연결을 구성할 수 있습니다. 애플리케이션이 시작될 때 자동으로 연결되도록 구성하려면 디자인 타임에 Object Inspector를 사용하여 *Active* 속성을 *True*로 설정합니다.

연결 정보 얻기

서버 소켓에 대한 연결을 완료한 후 클라이언트 소켓 컴포넌트와 연결된 클라이언트 소켓 객체를 사용하여 연결에 대한 정보를 얻을 수 있습니다. *LocalHost* 및 *LocalPort* 속성을 사용하면 클라이언트 및 서버 소켓이 연결의 끝점을 구성하기 위해 사용한 주소와 포트 번호를 알 수 있습니다. *Handle* 속성을 사용하면 소켓을 호출할 때 사용하는 소켓 연결에 대한 핸들을 얻을 수 있습니다.

연결 닫기

소켓 연결을 통한 서버 애플리케이션과의 통신이 끝나면 *Close* 메소드를 호출하여 연결을 종료할 수 있습니다. 서버측에서 연결이 끊어질 수도 있습니다. 이 경우에 사용자는 *OnDisconnect* 이벤트를 통해 공지받습니다.

서버 소켓 사용

서버 소켓 컴포넌트(*TcpServer* 또는 *UdpSocket*)를 폼이나 데이터 모듈에 추가하여 사용자의 애플리케이션을 IP 서버로 바꿉니다. 서버 소켓을 사용하면 제공하려는 서비스를 지정하거나 클라이언트 요청을 리스닝하는 데 사용할 포트를 지정할 수 있습니다. 서버 소켓 컴포넌트를 사용하면 클라이언트 연결 요청을 리스닝하고 승인할 수 있습니다.

각각의 서버 소켓 컴포넌트는 단일 서버 소켓 객체를 사용하여 리스닝 연결의 서버 끝점을 나타냅니다. 또한 각각의 컴포넌트는 서버가 승인하는 클라이언트 소켓에 대한 각 활성 연결의 서버 끝점에 대한 서버 클라이언트 소켓 객체를 사용합니다.

포트 지정

서버 소켓이 클라이언트 요청을 리스닝하려면 먼저 서버가 리스닝할 포트를 지정해야 합니다. *LocalPort* 속성을 사용하여 이 포트를 지정할 수 있습니다. 서버 애플리케이션이 일반적으로 특정 포트 번호와 연결된 표준 서비스를 제공하는 경우에는 *LocalPort* 속성을 사용하여 서비스 이름을 지정할 수도 있습니다. 포트 번호를 지정할 때에는 입력 오류가 발생하기 쉬우므로 포트 번호 대신 서비스 이름을 사용하는 것이 좋습니다.

클라이언트 요청 리스닝(listening)

서버 소켓 컴포넌트의 포트 번호를 일단 설정했으면 *Open* 메소드를 호출하여 런타임 시 리스닝 연결을 구성할 수 있습니다. 애플리케이션이 시작될 때 자동으로 리스닝 연결을 구성하도록 하려면 디자인 타임에 Object Inspector를 사용하여 *Active* 속성을 *True*로 설정합니다.

클라이언트에 연결

AutoAccept 속성이 *True*로 설정된 경우 리스닝 서버 소켓 컴포넌트는 클라이언트 연결 요청을 받았을 때 이를 자동으로 승인합니다. 사용자는 자동 승인이 *OnAccept* 이벤트에서 발생할 때마다 공지받습니다.

서버 연결 닫기

리스닝 연결을 종료하려면 *Close* 메소드를 호출하거나 *Active* 속성을 *False*로 설정합니다. 그렇게 하면 클라이언트 애플리케이션에 대해 열려 있는 모든 연결은 종료되며 아직 승인되지 않고 보류 중인 모든 연결이 취소된 후 리스닝 연결이 종료됩니다. 그런 다음 서버 소켓 컴포넌트는 더 이상 새로운 연결을 승인하지 않습니다.

TCP 클라이언트가 사용자의 서버 소켓에 대한 개별적인 연결을 종료하면 *OnDisconnect* 이벤트에서 사용자에게 이러한 사실을 알립니다.

소켓 이벤트에 응답

소켓을 사용하는 애플리케이션을 작성할 때 프로그램의 어디에서든지 소켓을 읽거나 쓸 수 있습니다. 소켓이 열린 후 사용자의 프로그램에서 *SendBuf*, *SendStream* 또는 *SendIn* 메소드를 사용하여 소켓에 작성할 수 있습니다. 유사한 이름의 메소드 *ReceiveBuf* 및 *ReceiveIn*을 이용하여 소켓을 읽을 수 있습니다. *OnSend* 및 *OnReceive* 이벤트는 소켓에

무언가를 쓰고 읽을 때마다 트리거됩니다. 필터링을 위해 이러한 이벤트를 사용할 수 있습니다. 사용자가 읽기 또는 쓰기 작업을 할 때마다 읽기 또는 쓰기 이벤트가 트리거됩니다.

연결로부터 오류 메시지를 받으면 클라이언트 소켓과 서버 소켓은 모두 오류 이벤트를 생성합니다.

또한 소켓 컴포넌트는 연결을 열고 완료하는 과정에서 두 개의 이벤트를 수신합니다. 애플리케이션이 소켓이 열리는 방법에 영향을 줄 필요가 있을 경우 *SendBuf* 및 *ReceiveBuf* 메소드를 사용하여 이러한 클라이언트 이벤트 또는 서버 이벤트에 응답합니다.

오류 이벤트

연결 중에 오류 메시지를 수신하면 클라이언트와 서버 소켓은 *OnError* 이벤트를 생성합니다. *OnError* 이벤트 핸들러를 작성하면 이러한 오류 메시지에 응답할 수 있습니다. 이러한 이벤트 핸들러에는 다음에 관한 정보가 전달됩니다.

- 오류 공지를 수신한 소켓 객체
- 오류 발생 시 소켓이 시도한 작업
- 오류 메시지가 제공한 오류 코드

이벤트 핸들러에서 오류에 응답하고 오류 코드를 0으로 변경하여 소켓에서의 예외 발생을 막을 수 있습니다.

클라이언트 이벤트

클라이언트 소켓이 연결되면 다음과 같은 이벤트가 발생합니다.

- 이벤트 공지를 위해 소켓이 설정되고 초기화됩니다.
- 서버와 서버 소켓이 생성된 후 *OnCreateHandle* 이벤트가 발생합니다. *Handle* 속성을 통해 사용할 수 있는 소켓 객체는 연결의 다른 끝을 구성할 서버나 클라이언트 소켓에 대한 정보를 제공할 수 있습니다. 이 때 연결에 사용되는 실제 포트를 처음으로 획득할 수 있으며 실제 포트는 연결을 승인한 리스닝 소켓의 포트와 다를 수도 있습니다.
- 연결 요청이 서버에 의해 승인되고 클라이언트 소켓에 의해 완료됩니다.
- 연결이 수립되면 *OnConnect* 공지 이벤트가 발생합니다.

서버 이벤트

서버 소켓 컴포넌트는 리스닝 연결과 클라이언트 애플리케이션에 대한 연결을 구성합니다. 각 연결이 구성되는 동안 서버 소켓은 이벤트를 수신합니다.

리스닝 시의 이벤트

리스닝 연결이 구성되기 바로 전에 *OnListening* 이벤트가 발생합니다. *Handle* 속성을 사용하면 리스닝에 대해 소켓이 열리기 전에 소켓을 변경할 수 있습니다. 예를 들어 리

스닝을 위해 서버가 사용하는 IP 주소를 제한하려면 *OnListening* 이벤트 핸들러에서 이 작업을 처리합니다.

클라이언트 연결 시의 이벤트

서버 소켓이 클라이언트 연결 요청을 승인하면 다음과 같은 이벤트가 발생합니다.

- *OnAccept* 이벤트가 발생하고 새 *TTcpClient* 객체가 이벤트 핸들러에 전달됩니다. 이 때 바로 *TTcpClient*의 속성을 사용하여 클라이언트에 대한 연결의 서버 끝점에 관한 정보를 획득할 수 있습니다.
- *BlockMode*가 *bmThreadBlocking*일 경우 *OnGetThread* 이벤트가 발생합니다. *TServerSocketThread*의 고유한 사용자 지정 자손을 제공하려는 경우 *OnGetThread* 이벤트 핸들러에서 사용자 지정 자손을 생성하여 *TServerSocketThread* 대신 사용할 수 있습니다. 스레드를 초기화하거나 연결을 통해 스레드가 읽거나 쓰기를 시작하기 전에 소켓 API를 호출하려면 *OnGetThread* 이벤트 핸들러도 사용해야 합니다.
- 클라이언트가 연결을 완료한 후 *OnAccept* 이벤트가 발생합니다. 비차단(non-blocking) 서버의 경우에는 이 때 소켓 연결을 통해 읽기 또는 쓰기를 시작할 수 있습니다.

소켓 연결을 통한 읽기 및 쓰기

소켓 연결을 통해 정보를 읽거나 쓰기 위하여 다른 시스템에 대한 소켓 연결을 구성합니다. 읽거나 쓸 대상이 되는 정보 또는 정보를 읽거나 쓰는 시기는 소켓 연결과 관련된 서비스에 따라 달라집니다.

소켓을 통한 읽기 및 쓰기는 비동기적으로 발생할 수 있기 때문에 네트워크 애플리케이션에서 다른 코드의 실행이 차단되지 않는데 이를 비차단 연결(non-blocking connection)이라고 합니다. 또한 차단 연결을 구성할 수도 있는데 그 다음 코드 줄을 실행하기 전에 애플리케이션은 읽기 또는 쓰기가 완료되기를 기다립니다.

비차단 연결(Non-blocking connections)

비차단 연결은 비동기적으로 읽고 쓰기 때문에 네트워크 애플리케이션에서 데이터 전송으로 인해 다른 코드의 실행을 차단하지 않습니다. 클라이언트 또는 서버 소켓에 대해 비차단 연결을 만들려면 *BlockMode* 속성을 *bmNonBlocking*으로 설정합니다.

비차단 연결인 경우 소켓은 읽기 및 쓰기 이벤트를 통해 연결의 다른 끝에 있는 소켓이 정보를 읽거나 쓰는 시기를 알 수 있습니다.

읽기 및 쓰기 이벤트

비차단 소켓은 연결을 통해 읽거나 써야 할 경우 읽기 및 쓰기 이벤트를 생성합니다. 사용자는 *OnReceive* 또는 *OnSend* 이벤트 핸들러에서 이러한 공지에 응답할 수 있습니다.

소켓 연결과 연결된 소켓 객체는 이벤트 핸들러를 읽거나 쓰기 위한 매개변수로써 제공됩니다. 소켓 객체는 사용자가 연결을 통해 정보를 읽거나 쓸 수 있도록 하는 여러 가지 메소드를 제공합니다.

소켓 연결에서 정보를 읽으려면 *ReceiveBuf* 또는 *ReceiveIn* 메소드를 사용합니다. 소켓 연결에 쓰려면 *SendBuf*, *SendStream* 또는 *SendIn* 메소드를 사용합니다.

차단 연결(Blocking connections)

연결이 차단된 경우 소켓은 연결을 통해 읽거나 쓰기를 시작해야 하며 소켓 연결로부터의 공지를 수동적으로 기다릴 수 없습니다. 읽기 및 쓰기가 발생하는 시기가 연결의 끝에서 결정될 경우 차단 소켓을 사용합니다.

클라이언트 또는 서버 소켓의 경우 *BlockMode* 속성을 *bmBlocking*으로 설정하여 차단 연결을 구성합니다. 사용자는 애플리케이션이 연결을 통한 읽기 또는 쓰기가 완료되기를 기다리는 동안 다른 스레드에서 코드를 계속 실행할 수 있도록 클라이언트 애플리케이션이 수행하는 작업에 따라 읽기 또는 쓰기를 위한 새 실행 스레드를 만들 수 있습니다.

서버 소켓의 경우에는 *BlockMode* 속성을 *bmBlocking* 또는 *bmThreadBlocking*으로 설정하여 차단 연결을 구성합니다. 소켓이 연결을 통한 정보 읽기 또는 쓰기가 완료되기를 기다리는 동안 차단 연결이 다른 모든 코드의 실행을 보류하기 때문에 서버 소켓 컴포넌트는 *BlockMode*가 *bmThreadBlocking*일 경우 항상 모든 클라이언트 연결에 대해 새 실행 스레드를 생성합니다. *BlockMode*가 *bmBlocking*이면 새 연결이 이루어질 때까지 프로그램 실행이 차단됩니다.

색인

숫자

3계층 애플리케이션, 다계층 애플리케이션 *참조*

가

객체 생성자 2-10
검색 경로 2-12
검색 경로 구분자 2-12
경로(URL) 6-3
경로명 2-12
계층 4-1
공급 프로세스 4-3
공유 객체
  동적(HTTP 서버) 6-6
  생성 1-3
  패키지 3-1
공유 객체 파일 2-8, 2-12
  패키지 3-2
구성 파일 2-11
그래픽
  HTML에 추가 7-13
기능
  이식 불가능 Windows 2-7
기본
  프로젝트 옵션 1-2
끝점
  소켓 연결 11-5

나

네임스페이스
  호출 가능한 인터페이스 10-4

다

다계층 아키텍처 4-4, 4-5
다계층 애플리케이션 4-1
  개요 4-3 ~ 4-4
  마스터/디테일 관계 4-8
  빌드 4-6 ~ 4-14
  서버 라이센스 4-3
  아키텍처 4-4, 4-5
  이점 4-2
  컴포넌트 4-2 ~ 4-3
  콜백 4-8
다중 문서 인터페이스 1-2
단일 문서 인터페이스 1-2
대소문자 구별 2-11
데이터 모듈 1-7 ~ 1-9
  비즈니스 룰 1-8
  생성 1-8
  원격과 표준 비교 1-8
  웹 8-2, 8-3, 8-4 ~ 8-5
  웹 애플리케이션 7-2
  이름 재지정 1-8
  폼에서 액세스 1-8
  Web Broker 애플리케이션 7-4
데이터 브로커 4-1
데이터 패킷 5-4
  XML 문서로 매핑 5-2
  XML 문서로 변환 5-1, 5-6 ~ 5-8
데이터베이스 1-4
  웹 애플리케이션 7-16
  HTML 응답 생성 7-16 ~ 7-19
데이터베이스 관리 시스템 4-1
데이터베이스 애플리케이션 1-4
  다계층 4-3 ~ 4-4
  분산 1-5
  포팅 2-23
  XML 5-1 ~ 5-10
데이터베이스 서버 1-4
데이터셋
  HTML 문서 7-18
데이터셋 페이지 프로듀서 7-17
  필드 값 변환 7-17
도움말 객체 등록 1-19
도움말 뷰어 1-13
도움말 시스템 1-13
  객체 등록 1-19
  인터페이스 1-14
도움말 선택기 1-19, 1-21
동적 공유 객체(DSO) 6-6
드라이브 문자 2-12
디렉토리, Linux 2-13
디스패처
  액션 항목 8-76
디스패처 컴포넌트 8-71
  어댑터 8-71
디스패처, 웹 7-2, 7-4 ~ 7-5
디스패치 액션 8-10
디자인 타임 패키지 3-1, 3-6 ~ 3-7

라

런타임 패키지 3-1, 3-3 ~ 3-6
레지스트리 2-11
레코드
  업데이트
    XML 문서 5-10
  웹 어댑터 8-6
레포지토리 1-9 ~ 1-12
  항목 사용 1-11
  항목 추가 1-10
로그인
  SOAP 연결 4-12
로그인 지원
  WebSnap 8-25 ~ 8-31
로그인, 요구 8-28 ~ 8-29
로그인 페이지
  WebSnap 8-27 ~ 8-28
루트 디렉토리 2-13
리스닝 연결 11-2, 11-3, 11-7, 11-8
  닫기 11-7
  포트 번호 11-5

마

마법사 1-9
  웹 서비스 10-3
  Console Wizard 1-3
  SOAP Data Module 4-7 ~ 4-8
  XML 데이터 바인딩 9-5 ~ 9-8
마샬링
  웹 서비스 10-5
마스터/디테일 관계
  다계층 애플리케이션 4-8
  중첩 테이블 4-9
매개변수
  HTML 태그 7-13
  TXMLTransformClient 5-9
매핑
  XML 5-2 ~ 5-3
    정의 5-4
멀티바이트 문자(MBCS) 2-14, 2-19
멀티태스킹 2-12
메시지 방식 서버
  웹 서버 애플리케이션 *참조*
메시지 헤더(HTTP) 6-3, 6-4
메시징 2-18
명령 스위치 2-11

바

배치 파일 2-11
변환 파일 5-1 ~ 5-6
  사용자 정의 노드 5-5, 5-7 ~ 5-8
  TXMLTransform 5-6
  TXMLTransformClient 5-9
  TXMLTransformProvider 5-8
보안
  다계층 애플리케이션 4-2
  SOAP 연결 4-12
분산 데이터 처리 4-2
분산 애플리케이션
  데이터베이스 1-5
비즈니스 룰 4-2, 4-7
  데이터 모듈 1-8
비차단 연결 11-9 ~ 11-10
  차단 연결과 비교 11-9

사

사용자 목록 서비스 8-10, 8-25
상태 정보
  통신 4-9 ~ 4-11
서버
  인터넷 6-1 ~ 6-6
서버 소켓 11-7
  오류 메시지 11-8
  이벤트 처리 11-8
  지정 11-6
  클라이언트 요청 승인 11-7
  Windows 소켓 객체 11-7
서버 애플리케이션
  다계층 4-4, 4-6 ~ 4-11
  등록 4-6, 4-11
  서비스 11-1
  아키텍처 4-5
  웹 서비스 10-2 ~ 10-8
  인터페이스 11-2
서버 연결 11-2, 11-3
  포트 번호 11-5
서버 유형 8-8
서버측 스크립트 8-31
  참조 8-35
서버측 스크립팅 8-7, 8-70
  참조 8-70
서비스
  구현 11-1 ~ 11-2, 11-7
  네트워크 서버 11-1
  요청 11-6
  포트 11-2
선택기
  도움말 1-19
세션
  웹 애플리케이션 7-16
세션 서비스 8-10, 8-25, 8-26 ~ 8-27
셸 스크립트 2-11
소스 코드
  재사용 1-12
소스 파일
  패키지 3-2, 3-7, 3-9, 3-13
소스 파일, 공유 2-10
소켓 11-1 ~ 11-10
  네트워크 주소 11-3
  서비스 구현 11-1 ~ 11-2, 11-7
  설명 11-3
  쓰기 11-10
  오류 처리 11-8
  이벤트 처리 11-7 ~ 11-9
  읽기 11-10
  읽기/쓰기 11-9 ~ 11-10
  정보 제공 11-4
  클라이언트 요청 승인 11-3
  호스트 지정 11-4
소켓 객체
  클라이언트 11-6
  클라이언트 소켓 11-6
소켓 연결 11-2 ~ 11-3
  끝점 11-3, 11-5
  다중 11-5
  닫기 11-6, 11-7
  열기 11-6, 11-7
  유형 11-2
  정보 전달/수신 11-9
소켓 컴포넌트 11-5 ~ 11-7
속성
  HTML 테이블 7-17
스크립트 8-7
  서버측 8-31
  편집과 보기 8-33
  활성 8-31
  WebSnap에서 생성 8-32
스크립트 객체 8-33
스크립트 보기 8-33
스크립트 편집 8-33
스크립트(URL) 6-3
스크립팅
  서버측 8-70
스타일 2-5
시그널 2-12
시스템 공지 2-18
시스템 이벤트 2-18
씬 클라이언트 애플리케이션 4-2
실행 파일 2-12
심볼릭 링크 2-12

아

아키텍처
  다계층 4-4, 4-5
  데이터베이스 애플리케이션
    서버 4-5
    클라이언트 4-4
  Web Broker 서버/애플리케이션 7-3
애플리케이션
  다계층 4-1
    개요 4-3 ~ 4-4
  데이터베이스 1-4
  웹 서버 1-5, 1-6, 8-7
  이식 가능 2-1 ~ 2-27
  크로스 플랫폼 2-1 ~ 2-27
  클라이언트/서버 4-1
  포팅 2-14
  Apache 7-1, 8-8
  Apache 웹 서버 1-6
  CGI 독립형 1-6, 8-8
  MDI 1-2
  NSAPI 7-1
  SDI 1-2
  Web Broker 7-1 ~ 7-19
애플리케이션 변수 7-2
애플리케이션 서버 4-1, 4-6 ~ 4-11
  등록 4-6, 4-11
  식별 4-12
  연결 끊기 4-13
  연결 열기 4-13
  인터페이스 4-8, 4-14
  작성 4-7
  콜백 4-8
애플리케이션 어댑터 8-9
애플리케이션 포팅 2-1 ~ 2-27
액세스 권한
  WebSnap 8-29 ~ 8-31
액션
  웹 어댑터 8-5
액션 응답 8-74
액션 에디터
  액션 변경 7-5
  액션 추가 7-4
액션 요청
  HTML 8-73
액션 항목 7-3, 7-4, 7-5 ~ 7-8
  기본 7-4, 7-6
  선택 7-5, 7-6
  연결 7-7
  요청에 응답 7-7
  웹 디스패처 8-76
  이벤트 핸들러 7-3

추가 7-4
페이지 프로듀서 7-14
활성화 및 비활성화 7-6
양방향 속성 2-7
어댑터 8-2, 8-5 ~ 8-6
레코드 8-6
액션 8-5
오류 8-6
필드 8-5
어댑터 디스패처 8-9, 8-71
요청 8-73
어댑터 컴포넌트
사용 8-71
어댑터 페이지 프로듀서 8-35
어셈블러 코드 2-17
연결
끊기 4-13
열기 4-13, 11-6
종료 11-7
클라이언트 11-3
프로토콜 4-12
SOAP 4-5, 4-12
TCP/IP 11-2 ~ 11-3
연결 컴포넌트
데이터베이스
원격 데이터 모듈 4-5
DataSnap 4-3, 4-4, 4-11, 4-12 ~ 4-14
연결 관리 4-13
연결 끊기 4-13
연결 열기 4-13
프로토콜 4-12
열
HTML 테이블에 포함 7-18
예외 2-12
오류
소켓 11-8
웹 어댑터 8-6
요청
디스패치 8-71
액션과 HTML 8-73
어댑터 8-73
이미지 8-75
요청 객체
헤더 정보 7-3
요청 디스패치
WebSnap 8-71
요청 메시지 7-3
디스패치 7-4
유형 7-9
응답 7-7 ~ 7-8, 7-11
액션 항목 7-5
컨텐트 7-10
처리 7-4
헤더 정보 7-8 ~ 7-10
HTTP 개요 6-5 ~ 6-6
요청 목록(패키지) 3-7, 3-8, 3-10
요청 헤더 7-8
원격 가능 클래스
등록 10-5
예외 10-7
원격 가능 타입 레지스트리 10-5, 10-7
원격 데이터 모듈 4-2, 4-6, 4-7 ~ 4-8
stateless 4-9 ~ 4-11
원격 애플리케이션
웹 데이터 모듈 8-2, 8-3, 8-4 ~ 8-5
구조 8-4
웹 디스패처 7-2, 7-4 ~ 7-5, 8-71
액션 항목 8-76
액션 항목 선택 7-5, 7-6
요청 처리 7-3, 7-7
웹 모듈 7-2, 7-4, 8-2, 8-2 ~ 8-5
데이터베이스 세션 추가 7-16
유형 8-2
웹 브라우저 6-5
URL 6-5
웹 서버 6-1 ~ 6-6
유형 8-8
클라이언트 요청 6-5
웹 서버 애플리케이션 1-5, 1-6, 6-1 ~ 6-6
개요 6-6
리소스 위치 6-3
유형 6-6
테이블 쿼리 7-19
표준 6-3
웹 서버 유형 8-8
웹 서비스 10-1 ~ 10-10
구현 클래스 10-6 ~ 10-7
등록 10-6
네임스페이스 10-4
마법사 10-3
복잡한 타입 10-5 ~ 10-6
서버 10-2 ~ 10-8
작성 10-2 ~ 10-3
예외 10-7
클라이언트 10-8 ~ 10-10
웹 서비스 importer 10-9
웹 스크립팅 8-7
웹 애플리케이션
객체 7-2
웹 애플리케이션 모듈 8-2, 8-3
웹 어댑터
레코드 8-6
액션 8-5
오류 8-6
필드 8-5
웹 컨텍스트 8-71
웹 페이지 6-5
웹 페이지 모듈 8-2, 8-3, 8-3 ~ 8-4
웹 페이지 모듈 작성 8-18
위치 독립 코드(PIC) 2-8, 2-17, 2-18
유닛
패키지 포함 3-3
응답
액션 8-74
어댑터 8-73
이미지 8-75
응답 메시지 7-3
데이터베이스 정보 7-16 ~ 7-19
보내기 7-7, 7-12
생성 7-10 ~ 7-12, 7-12 ~ 7-19
상태 정보 7-10
컨텐트 7-11, 7-12 ~ 7-19
헤더 정보 7-10 ~ 7-11
응답 템플릿 7-13
응답 헤더 7-11
이미지 요청 8-75
이벤트 2-18
이벤트 핸들러 2-18
이식 가능 코드 2-14
인터넷 서버 6-1 ~ 6-6
인터넷 표준 및 프로토콜 6-3
인터페이스
도움말 시스템 1-14
애플리케이션 서버 4-8, 4-14
웹 서비스 10-1
호출 가능 10-2
DOM 9-2
XML 노드 9-4
인트라넷
호스트 이름 11-4

자

자습서
  WebSnap 8-11 ~ 8-23
재배치 가능 코드 2-17
전역 오프셋 테이블(GOT) 2-18
절대 주소 2-8
점진적 페치(incremental fetching) 4-9
조건부 컴파일 2-15, 2-16
주소
  소켓 연결 11-3
줄 끝 문자 2-11
중첩 디테일 4-9
중첩 테이블 4-9
지시어 2-16
  조건부 컴파일 2-15
  패키지 관련 3-11
  $DENYPACKAGEUNIT 컴파일러 3-11
  $DESIGNONLY 컴파일러 3-11
  $ELSEIF 2-16
  $ENDIF 2-16
  $G 컴파일러 3-11
  $IF 2-16
  $IFDEF 2-15
  $IFEND 2-16
  $IFNDEF 2-15
  $IMPLICITBUILD 컴파일러 3-11
  $IMPORTEDDATA 컴파일러 3-11
  LEpath 컴파일러 3-12
  LNpath 컴파일러 3-12
  Lupackage 컴파일러 3-12
  $MESSAGE 컴파일러 2-17
  $RUNONLY 컴파일러 3-11
  $SOPREFIX 컴파일러 1-4
  $SOSUFFIX 컴파일러 1-4
  $SOVERSION 컴파일러 1-4
  $WEAKPACKAGEUNIT 컴파일러 3-11
  Z 컴파일러 3-12

차

차단 연결 11-10
  비차단 연결과 비교 11-9
  이벤트 처리 11-9
참조
  패키지 3-3
최종 사용자 어댑터 8-9, 8-25

카

컨텍스트 ID 1-17
컨텐트 속성
  웹 응답 객체 7-12
컨텐트 프로듀서 7-3, 7-12
  이벤트 처리 7-14, 7-15, 7-16
컴포넌트
  디스패처 8-71
  사용자 지정 1-12
  설치 3-6 ~ 3-7
  페이지 프로듀서 8-4
컴포넌트 라이브러리 2-4
컴포넌트 템플릿 1-12, 1-13
컴포넌트 팔레트
  컴포넌트 추가 3-7
  Data Access 페이지 4-2
  DataSnap 페이지 4-5
  WebServices 페이지 4-2
컴파일러 옵션 1-2
컴파일러 지시어 2-16
  패키지 특정 3-11
코드
  이식 가능 2-12, 2-14
코드 에디터
  패키지 열기 3-9
콘솔 애플리케이션 1-2
  CGI 6-6
콜백
  다계층 애플리케이션 4-8
    제한 4-6
콤보 박스 2-6
쿼리
  웹 애플리케이션 7-19
  HTML 테이블 7-19
쿼리 부분(URL) 6-3
크로스 플랫폼 애플리케이션 2-1 ~ 2-27
클라이언트 데이터셋 4-3
클라이언트 애플리케이션
  다계층 4-2, 4-4
  사용자 인터페이스 4-1
  생성 4-11 ~ 4-14
  소켓 11-1
  씬 4-2
  아키텍처 4-4
  웹 서비스 10-8 ~ 10-10
  인터페이스 11-2
클라이언트 연결 11-2, 11-3
  열기 11-6
  요청 승인 11-7
  포트 번호 11-5
클라이언트 소켓 11-3, 11-6
  서버 식별 11-6
  서버에 연결 11-8
  서비스 요청 11-6
  속성 11-6
  오류 메시지 11-8
  이벤트 처리 11-8
  호스트 지정 11-4
클라이언트 소켓 객체 11-6
클라이언트 요청 6-5 ~ 6-6, 7-8
클라이언트, 클라이언트 애플리케이션 *참조*
클라이언트/서버 애플리케이션 1-4
키워드 방식 도움말 1-17

타

타입
  웹 서비스 10-5 ~ 10-6
터미널 유형 2-11
테이블 프로듀서 7-17 ~ 7-19
  속성 설정 7-17
템플릿 1-9, 1-11
  컴포넌트 1-12, 1-13
  페이지 프로듀서 8-4
  프로그래밍 1-9
  HTML 7-13 ~ 7-16
  Web Broker 애플리케이션 7-2
통신 11-1
  표준 6-3
  프로토콜 6-3, 11-2
트랜잭션
  다계층 애플리케이션 4-8
  업데이트 적용 4-8
  트랜잭션 데이터 모듈 4-8

파

파일
  웹으로 보내기 7-12
파일 권한 2-12
파일 끝 문자 2-11
패키지 3-1 ~ 3-14
  공유 객체 파일 3-2
  기본 설정 3-8
  디자인 전용 옵션 3-8
  디자인 타임 3-1, 3-6 ~ 3-7
  런타임 3-1, 3-3 ~ 3-6, 3-8
  사용 1-4
  사용자 지정 3-6
  생성 1-3, 3-7 ~ 3-13

설치 3-6 ~ 3-7
소스 파일 3-2, 3-13
애플리케이션 배포 3-3, 3-13
애플리케이션에서 사용 3-3 ~ 3-6
옵션 3-8
요청 목록 3-7, 3-8, 3-10
중복 참조 3-10
참조 3-3
컴파일 3-11 ~ 3-13
옵션 3-11
컴파일러 지시어 3-11
파일명 확장자 3-1
편집 3-8 ~ 3-9
포함 목록 3-7, 3-8, 3-10
팩토리 8-5
페이지 디스패처 8-9, 8-71
디스패처
페이지 8-77
페이지 모듈 8-2, 8-3, 8-3 ~ 8-4
페이지 프로듀서 7-13 ~ 7-16, 8-2, 8-4, 8-6 ~ 8-7, 8-35
연결 7-15
유형 8-10
이벤트 처리 7-14, 7-15, 7-16
컴포넌트 8-4
템플릿 8-4
템플릿 변환 7-14
Content 메소드 7-14
ContentFromStream 7-14
ContentFromString 7-14
data-aware 7-17
포트 11-5
다중 연결 11-5
서비스 11-2
클라이언트 소켓 11-6
포함 목록(패키지) 3-7, 3-8, 3-10
폼과 대화 상자 공유 1-9 ~ 1-12
프레임 1-12
프로바이더
원격 4-5
XML 5-8
XML 문서에 데이터 제공 5-9 ~ 5-10
XML 문서와 연결 5-8
프로세스 2-12
프로젝트 옵션 1-2
기본 1-2
프로젝트 템플릿 1-11
프로토콜
연결 컴포넌트 4-12
인터넷 6-3, 11-1
필드
웹 어댑터 8-5

하

하이퍼텍스트 링크
HTML에 추가 7-13
해결 프로세스 4-4
핸들
소켓 연결 11-6
핸들 속성 11-6
헤더
HTTP 요청 6-4
호스트 11-4
주소 11-4
URL 6-3
호스트 이름 11-4
IP 주소 11-4
호출 가능한 인터페이스 10-2, 10-3 ~ 10-4
구현 10-6 ~ 10-7
네임스페이스 10-4
등록 10-4
정의 10-2
호출 10-9 ~ 10-10
URI 10-9
호출 레지스트리 10-4, 10-6
홈 디렉토리 2-11
활성 스크립팅 8-31

A

Actions 속성 7-4
Active 속성
서버 소켓 11-7
클라이언트 소켓 11-6
AdapterPageProducer 8-10
Add To Repository 명령 1-10
AfterConnect 이벤트 4-13
AfterDisconnect 이벤트 4-13
AfterDispatch 이벤트 7-5, 7-8
Apache 서버 공유 객체
생성 8-8
Apache 서버 DSO
생성 7-1
Apache 애플리케이션
생성 7-1, 8-8
Apache 웹 서버 애플리케이션 1-6
ApplyUpdates 메소드 2-26
TXMLTransformClient 5-10
AppServer 속성 4-8
AutoAccept 속성 11-7

B

.bashrc 2-11
B2B 통신
XML 9-1
BaseCLX 2-4
baseclx 3-5
BeforeConnect 이벤트 4-13
BeforeDisconnect 이벤트 4-13
BeforeDispatch 이벤트 7-4, 7-6
bin 디렉토리 2-13
BlockMode 속성 11-9, 11-10
bmBlocking 11-10
bmThreadBlocking 11-9, 11-10

C

C++ 객체 2-6
CachedUpdates 속성 2-26
CancelBatch 메소드 2-26
CancelUpdates 메소드 2-26
CGI 애플리케이션
생성 8-8
CGI 프로그램 1-6, 6-6
생성 7-1
ChangeBounds 속성 2-18
ChangeCount 속성 2-26
ChangeScale 속성 2-18
CLX
객체 생성자 2-10
VCL과 비교 2-4
Code Insight
템플릿 1-9
ColorChanged 속성 2-18
CommitUpdates 메소드 2-26
Console Wizard 1-3
Content 메소드
페이지 프로듀서 7-14
ContentFromStream 메소드
페이지 프로듀서 7-14
ContentFromString 메소드
페이지 프로듀서 7-14
ContentStream 속성
웹 응답 객체 7-12
ContextHelp 1-19
Copy(Object Repository) 1-11
CursorChanged 속성 2-18

D

.dcp 파일 3-2, 3-13
.dpk 파일 3-2, 3-7
.dpu 파일 3-2, 3-13
$DENYPACKAGEUNIT/컴파일러 지시어 3-11
$DESIGNONLY 컴파일러/지시어 3-11
Data Controls 페이지(컴포넌트 팔레트) 4-2
DataCLX 2-4
dataclx 3-5
DataSnap 페이지(컴포넌트 팔레트) 4-5
dbExpress 2-20 ~ 2-25
DBMS 4-1
DECnet 프로토콜 (Digital) 11-1
Default 속성
  액션 항목 7-6
DefaultPage 속성 8-77
DLL 2-8, 2-12
DocumentElement 속성 9-3
DOM 9-2, 9-2 ~ 9-3
  구현 9-2
    사용 9-3
DSO 6-6

E

$ELSEIF 지시어 2-16
EBX 레지스터 2-8, 2-18
Enabled 속성
  액션 항목 7-6
EnabledChanged 속성 2-18
ERemotableException 10-7

F

FetchAll 메소드 2-26
Fields 에디터 1-9
FontChanged 속성 2-18
Footer 속성 7-18
Free 메소드 2-10

G

$G 컴파일러 지시어 3-11, 3-12
gcc 버전 2-6
GetFieldByName 메소드 7-8
GetHandle 1-16
GetHelpFile 1-16
GetHelpStrings 1-16
GetNextPacket 메소드 2-26
GetViewerName 1-15
GetXML 메소드 5-10
GNU 어셈블러 2-14
GNU make 유틸리티 2-12

H

Header 속성 7-18
Help Manager 1-13, 1-14 ~ 1-22
HelpContext 1-20
HelpFile 1-20
HelpIntfs.pas 1-14
HelpKeyword 1-20
HelpSystem 1-20
HelpType 1-20
HTML 명령 7-13
  데이터베이스 정보 7-16
  생성 7-14
HTML 문서 6-5
  데이터베이스 7-16
  데이터셋 7-18
  데이터셋 페이지 프로듀서 7-17
  테이블 포함 7-18
  테이블 프로듀서 7-17 ~ 7-19
  템플릿 7-13 ~ 7-14
  페이지 프로듀서 7-13 ~ 7-16
  HTTP 응답 메시지 6-6
HTML 테이블 7-13, 7-18
  생성 7-17 ~ 7-19
  속성 설정 7-17
  캡션 7-18
HTML 템플릿 7-13 ~ 7-16, 8-4
HTML 투명 태그
  구문 7-13
  매개변수 7-13
  변환 7-13, 7-14
  이미 정의된 태그 7-13
HTML Result 탭 8-2
HTML Script 탭 8-2
HTMLDoc 속성 7-14
HTMLFile 속성 7-14
HTTP 6-3
  개요 6-5 ~ 6-6
  메시지 헤더 6-3
  상태 코드 7-10
  요청 헤더 6-4, 7-8
  응답 헤더 7-11
  SOAP 10-1
HTTP 요청
  이미지 8-75
HTTP 요청 메시지 8-71
HTTP 응답
  액션 8-74
  이미지 8-75
HyperHelp 1-13
HyperHelp 뷰어 1-21

I

$IFDEF 지시어 2-15
$IFEND 지시어 2-16
$IFNDEF 지시어 2-15
$IMPLICITBUILD 컴파일러 지시어 3-11
$IMPORTEDDATA 컴파일러 지시어 3-11
IAppServer 인터페이스 4-5
  상태 정보 4-9
  호출 4-14
  확장 4-8
ICustomHelpViewer 1-13, 1-14, 1-15, 1-16
  구현 1-15
IDOMImplementation 9-3
IETF 프로토콜 및 표준 6-3
IExtendedHelpViewer 1-14, 1-17
IHelpManager 1-14, 1-20
IHelpSelector 1-14, 1-18
IHelpSystem 1-14, 1-20
IInvokable 10-3
Image HTML 태그 (<IMG>) 7-13
ImageMap HTML 태그 (<MAP>) 7-13
indy 3-5
Inherit(Object Repository) 1-11
ini 파일 2-6
InterBaseExpress 2-22
interface
  호출 가능 10-3 ~ 10-4
Internet Engineering Task Force 6-3
IP 주소 11-3, 11-4, 11-6
  호스트 11-4
  호스트 이름 11-4
IPX/SPX 프로토콜 11-1
ISpecialWinHelpViewer 1-14
IXMLNode 9-4 ~ 9-5, 9-6

K

KeywordHelp 1-19

L

-LEpath 컴파일러 지시어 3-12
-LNpath 컴파일러 지시어 3-12

-LUpackage 컴파일러 지시어 3-12
Link HTML 태그(<A>) 7-13
Linux
  디렉토리 2-13
  환경 2-11
LocalHost 속성
  클라이언트 소켓 11-6
LocalPort 속성
  클라이언트 소켓 11-6

M

$MESSAGE 지시어 2-17
make 유틸리티 2-12
Man 페이지 1-13
MaxRows 속성 7-18
MDI 애플리케이션 1-2
  생성 1-2
Method 속성 7-9
MethodType 속성 7-6, 7-9
midas.so 4-3
midaslib.dcu 4-3
MIME 메시지 6-6
Multitier 페이지(New Items 대화 상자) 4-2

N

NetCLX 1-6, 2-4
netclx 3-5
netdataclx 3-5
Netscape Server DLL
  생성 7-1
New Items 대화 상자 1-9, 1-10, 1-11
NotifyID 1-15
NSAPI 애플리케이션
  생성 7-1

O

Object Repository 1-9 ~ 1-12
  공유 디렉토리 지정 1-10
  항목 사용 1-11
  항목 추가 1-10
objrepos 디렉토리 1-10
OnAccept 이벤트 11-7
  서버 소켓 11-9
OnAction 이벤트 7-7
OnClientDisconnect 이벤트 11-7
OnConnect 이벤트
  클라이언트 소켓 11-8
OnConnecting 이벤트
  서버 소켓 11-8
OnDisconnect 이벤트
  클라이언트 소켓 11-6
OnError 이벤트
  소켓 11-8
OnGetThread 이벤트 11-9
OnHandleActive 이벤트
  클라이언트 소켓 11-8
OnHTMLTag 이벤트 7-14, 7-15, 7-16
OnReceive 11-9
OnReceive 이벤트 11-7
OnReconcileError 이벤트 2-26
OnSend 11-9
OnSend 이벤트 11-7
OnTranslate 이벤트 5-7
OnUpdateError 이벤트 2-26
Open 메소드
  서버 소켓 11-7

P

PacketRecords 속성 2-26
PaletteChanged 속성 2-19
PathInfo 속성 7-5
Port 속성
  서버 소켓 11-7
Project Options 대화 상자 1-2
ProviderName 속성 4-11, 5-9

Q

Qt widget 2-10

R

$RUNONLY 컴파일러 지시어 3-11
RDBMS 4-1
ReasonString 속성 7-11
ReceiveBuf 메소드 11-7, 11-8
Receiveln 메소드 11-7
Reconcile 메소드 2-26
RegisterComponents 프로시저 3-7
RegisterHelpViewer 1-22
RegisterViewer 함수 1-19
RemoteHost 속성
  클라이언트 소켓 11-6
RemotePort 속성
  클라이언트 소켓 11-6
RemoteServer 속성 4-11, 4-13, 5-9
Repository 대화 상자 1-9
RevertRecord 메소드 2-26
RFC(Request for Comment) 문서 6-3
RowAttributes 속성 7-18
RTTI
  호출 가능한 인터페이스 10-2

S

$SOPREFIX 지시어 1-4
$SOSUFFIX 지시어 1-4
$SOVERSION 지시어 1-4
SDI 애플리케이션 1-2
SelectKeyword 1-18
SendBuf 메소드 11-7, 11-8
Sendln 메소드 11-7
SendStream 메소드 11-7
ShowHintChanged 속성 2-19
ShutDown 1-15
SOAP 10-1
  다계층 애플리케이션 4-5
  애플리케이션 서버에 연결 4-12
SOAP 폴트 패킷 10-7
SOAP Data Module 마법사 4-7 ~ 4-8
SOAP 연결 4-5, 4-12
SOAP(Simple Object Access Protocol) *참조*
SoftShutDown 1-15
SourceXml 속성 5-6
SourceXmlDocument 속성 5-6
SourceXmlFile 속성 5-6
StatusCode 속성 7-10
StatusFilter 속성 2-26
StrNextChar 함수 2-14
StyleChanged 속성 2-19

T

Table HTML 태그 (<TABLE>) 7-13
TableAttributes 속성 7-17
TableOfContents 1-18
TabStopChanged 속성 2-19
TAdapterDispatcher 8-71
TAdapterPageProducer 8-32
TApplication 1-14, 1-20, 2-5
TASM 코드 2-14
TCGIApplication 6-6
TCGIRequest 6-6
TCGIResponse 6-6
TClientSocket 11-6

TCP/IP 11-1
  서버 11-7
  클라이언트 11-6
TCustomContentProducer 7-12
TCustomEdit 2-7
TDataSetTableProducer 7-18
THTMLTableAttributes 7-17
THTMLTableColumn 7-18
THTTPRio 10-9
THTTPSoapDispatcher 10-2, 10-3
THTTPSOAPPascalInvoker 10-3
THTTPSoapPascalInvoker 10-2
TInvokableClass 10-6
TMemIniFile 2-6
ToggleButton 2-6
TPageDispatcher 8-71
TPageProducer 7-13
TQuery 1-9
TransformGetData 속성 5-9
TransformRead 속성 5-8
TransformSetParams 속성 5-9
TransformWrite 속성 5-8
TRegIniFile 2-6
TRemotable 10-5
TServerSocket 11-7
TSQLQuery 7-19
TSQLQueryTableProducer 7-19
TStringList 1-16
TTable 1-9
TWebActionItem 7-3
TWebAppDataModule 8-2
TWebApplication 6-6
TWebAppPageModule 8-2
TWebContext 8-71
TWebDataModule 8-2
TWebDispatcher 8-71, 8-76
TWebPageModule 8-2
TWebRequest 6-6
TWebResponse 6-6, 7-3
TWidgetControl 2-4
TWinControl 2-4
TWSDLHTMLPublish 10-7
TWSDLHTMLPublisher 10-3
TXMLDocument 9-3, 9-8
TXMLTransform 5-6 ~ 5-8
  소스 문서 5-6
TXMLTransformClient 5-9 ~ 5-10
  매개변수 5-9
TXMLTransformProvider 5-8

U

UDP 프로토콜 11-1
UpdateBatch 메소드 2-26
UpdateRecordTypes 속성 2-26
UpdatesPending 속성 2-26
UpdateStatus 속성 2-26
URI
  URL과 비교 6-4
URL 6-3
  웹 브라우저 6-5
  호스트 이름 11-4
  IP 주소 11-4
  SOAP 연결 4-12
  URI와 비교 6-4
URL 속성 4-12, 7-8, 10-10
Use Unit 명령 1-9
USEPACKAGE 매크로 3-8
uses 절 2-4
  데이터 모듈 추가 1-8
  패키지 포함 3-3
UTF-8 문자 집합 2-14

V

VisibleChanged 속성 2-19
VisualCLX 2-4
visualclx 3-5
visualdbclx 3-5

W

$WEAKPACKAGEUNIT/컴파일러 지시어 3-11
W3C 9-2
wchar_t widechar 2-19
Web Broker 1-6, 6-1 ~ 6-3
Web Broker 서버 애플리케이션 7-1 ~ 7-19
  개요 7-1 ~ 7-3
  데이터 포스트 대상 7-9
  데이터베이스 액세스 7-16
  생성 7-1 ~ 7-2
  아키텍처 7-3
  웹 디스패처 7-4
  응답 만들기 7-7
  응답 템플릿 7-13
  이벤트 처리 7-4, 7-6, 7-7
  템플릿 7-2
  파일 보내기 7-12
  프로젝트에 추가 7-2
Web Service Definition Language, WSDL *참조*
webdsnapclx 3-5
WebServices 페이지(컴포넌트 팔레트) 4-2
WebSnap 6-1 ~ 6-3
  로그인 지원 8-25 ~ 8-31
  로그인 페이지 8-27 ~ 8-28
  로그인 요구 8-28 ~ 8-29
  서버측 스크립팅 8-70
    참조 8-35 ~ 8-70
    예제 8-51
  서버측 스크립트 8-31
  스크립팅 객체 유형 8-39
  액세스 권한 8-29 ~ 8-31
  전역 스크립트 객체 8-35
WebSnap 자습서 8-11 ~ 8-23
websnapclx 3-5
widestrings 2-19
widget 2-4
WidgetDestroyed 속성 2-19
WIN32 2-15
WIN64 2-15
Windows 메시징 2-14
Windows 애플리케이션 2-1
  포팅 2-1 ~ 2-12
Windows 소켓 객체 11-5
  서버 소켓 11-7
  클라이언트 11-5
WSDL 10-2
  게시 10-7 ~ 10-8
  파일 10-8
  import하기 10-9

X

XDR 파일 9-2
XML 5-1, 9-1
  데이터베이스 애플리케이션 5-1 ~ 5-10
  매핑 5-2 ~ 5-3
    정의 5-4
  처리 명령 9-1
  파서 9-2
  DTD 9-2
  SOAP 10-1
XML 데이터 바인딩 마법사 9-5 ~ 9-8
XML 문서 5-1, 9-1 ~ 9-8
  노드 9-2, 9-4 ~ 9-5
    값 9-4

변환 파일 5-1
속성 5-5, 9-5, 9-6
자식 9-5
필드로 매핑 5-2
데이터 패킷으로 변환 5-1, 5-6 ~ 5-8
데이터베이스 정보 게시 5-9
루트 노드 9-3, 9-6, 9-8
인터페이스 생성 9-6
컴포넌트 9-3, 9-8
XML 스키마 9-2
XMLDataFile 속성 5-8
XMLMapper 5-2, 5-4 ~ 5-6
xmlrtl 3-5
XNS(Xerox Network System) 11-1
XSD 파일 9-2
XSLPageProducer 8-4

Z

-Z 컴파일러 지시어 3-12