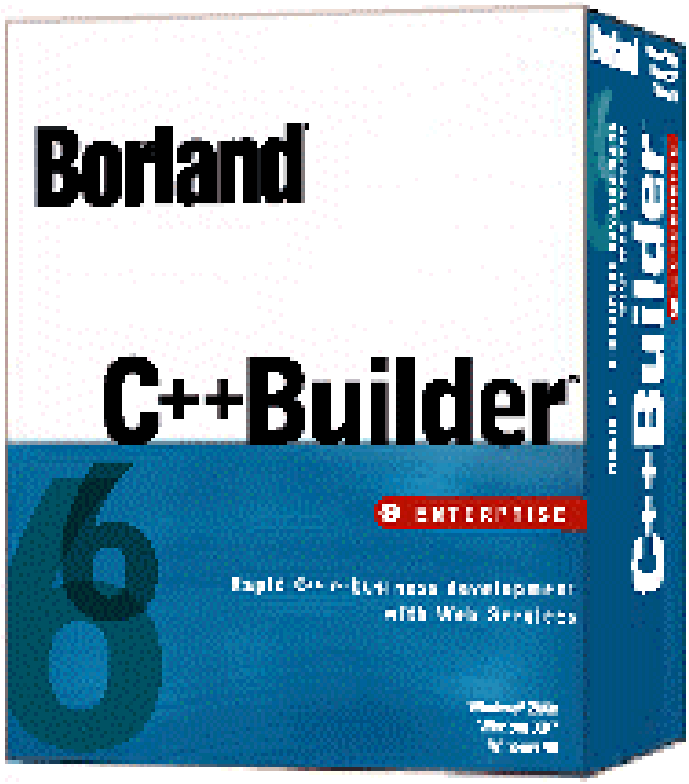
C++ Strikes Back - C++빌더 6



박지훈 (볼랜드포럼)

선정 이유

웹서비스의 열풍은 마치 마이크로소프트의 닷넷과 썬 마이크로시스템즈의 자바 플랫폼에만 해당되는 것처럼 양쪽 진영의 불꽃튀기는 경쟁이 거세다. 하지만 웹서비스가 개발 언어와는 무관하게 제안된 표준에 의한 것이므로 C++ 개발자들이 웹서비스 개발에서 소외되어야 할 이유는 전혀 없다. 볼랜드에서 2월 5일자로 발표한 C++Builder 6는 이러한 요구를 충분히 소화할 뿐 아니라, 뒤이어 출시될 리눅스 플랫폼용 C++Builder 호환 개발툴과 100% 호환되는 크로스플랫폼 개발까지 약속하고 있다.

특징

- Two-Way Tool : 생산성을 위한 RAD 개발과 최적화를 위한 코드기반 개발이 동시에 가능
- CLX 크로스플랫폼 라이브러리를 통한 윈도우/리눅스 크로스플랫폼 개발 지원
- BizSnap을 통한 웹서비스 개발 지원
- WebSnap / WebBroker를 이용한 고성능 웹 애플리케이션 개발 지원
- DataSnap / dbExpress를 통한 강력하고 빠른 데이터베이스 애플리케이션 개발
- MFC / STL / OWL / VCL / CLX 등 자주 사용되는 거의 모든 클래스라이브러리 지원
- 더욱 강화된 코딩 편의 기능

제품 구성

- CD 1 : C++Builder 6 / InterBase 6.5 (개발자 라이선스) / 비지브로커 4.5 (개발자 라이선스) / TeamSource

- CD 2 : 컴패니언 시디 (다양한 공개 컴퍼넌트들을 포함)
- 매뉴얼 셋 : Developer's Guide / Quick Start

사용 환경

- CPU : 인텔 펜티엄II 400MHz 혹은 호환 CPU
- 메모리 : 128MB (256MB 권장)
- 하드디스크(전체설치) : 엔터프라이즈 750MB / 프로페셔널 650MB / 퍼스널 550MB
- OS : 윈도우 98 / ME / 2000 / XP

문의처

- 업체명 : 볼랜드코리아
- TEL : 02-6001-3195
- URL : www.borlandkorea.co.kr <http://www.borlandkorea.co.kr>
- 가격 : (문의)

최근의 IT 업계의 동향을 지켜보면, 마치 자바나 C#과 같은 비교적 새롭게 나타난 언어들이 C++의 위치를 대체하게 될 것 같은 느낌이 들게 된다. 하지만 그런 분위기에도 불구하고, 전세계에서 가장 많은 개발자와 가장 많은 소스를 가지고 있는 언어는 역시 두말할 필요도 없이 C++인 것이다. C++은 그 역사를 같이하는 C언어와 함께 20년에 가까운 시간동안 IT 업계를 지배해왔고, 그 시간의 무게만큼 호락호락하게 자리를 내어줄리가 없다. 더욱이 새롭게 등장하는 언어들이 C/C++의 강력함을 모두 커버하지 못하고 있음을 생각해보면 더욱더 그러하다.

볼랜드는 프로그래밍 언어 컴파일러에서의 높은 기술로 인기를 누려온 전통의 개발툴 전문 벤더다. Turbo C와 Borland C++에서 현재의 C++Builder로 이어지는 C++ 컴파일러들과, Turbo Pascal과 Borland Pascal을 거쳐 현재의 Delphi, Kylix에 이르는 파스칼 컴파일러들, 그리고 자바의 등장 이후로 자바 개발툴들 중 가장 높은 점유율을 보이며 볼랜드 개발툴 제품군의 새로운 축을 형성한 JBuilder가 그 중심이라고 할 수 있다. Turbo C와 Borland C++은 도스 시절의 초기부터 윈도우가 전성기를 이룰 때까지 C/C++ 열풍의 중심에서 거의 100%에 가까운 점유율로 IT 업계를 이끌어온 제품들로서, 5년 이상의 개발 경력을 가진 개발자라면 잊을 수 없는 이름들이기도 하다. 그리고 C++Builder는 Turbo C와 Borland C++의 명성에 결코 뒤지지 않는 뛰어난 성능과 편의성, 그리고 최근의 IT 업계의 최신 기술들에 대한 완벽한 지원 기능들로 무장하고 다시 나타났다.

1. Borland C++Builder 소개

Borland C++Builder에 익숙하지 않은 분들을 위해, 조금 무리를 해서라도 C++Builder를 한마디로 소개하자면, Borland C++의 탁월한 C++ 컴파일러에 Delphi의 RAD 엔진을 얹은 것이라고 할 수 있다. C++Builder의 개발환경(IDE)는 Delphi와 거의 다른 점이 없으며, 눈에 보이는 대로만 생각하자면 오히려 Delphi에 C++을 접목한 것이 아닌가 하고 의심할 정도다. 볼랜드의 개발툴에 전혀 경험이 없는 분이라면, 마이크로소프트의 Visual C++에 Visual Basic을 더했다고 보면 어느정도 상상이 될 것이다. 1997년에 C++Builder의 첫 버전이 출시되기 전까지는 C++ 언어를 이용하는 RAD 개발툴을 구현한다는 것은 거의 불가능하다고 알려져왔으나, 볼랜드는 ANSI/ISO C++ 표준에 컴퍼넌트 프로그래밍 인터페이스를 위한 단 6개의 키워드를 추가함으로써 RAD 환경을 완벽하게 구현해냈다. 수십년 동안 컴파일러 기술에서는 최고 수준으로 꼽혀왔던 볼랜드이기에 가능한 것이었다.

여기에서는 C++Builder에 아직 익숙하지 않은 분들을 위해 C++Builder의 대표적인 특징 몇가지를 간략하게 설명해고자 한다.

1.1 컴파일러로서의 C++Builder

볼랜드의 C/C++ 컴파일러들은 ANSI/ISO C++ 표준을 잘 준수하는 것으로 잘 알려져 있다. C++Builder의 컴파일러인 Borland C++ 5.5는 ANSI/ISO 표준 C++의 최신 버전을 지원하며, 마이크로소프트의 Win32 SDK를 지원하므로 Win32 SDK 레벨에서 Visual C++과 완전히 동등하다. 또한 C++Builder는 MFC / STL / OWL / VCL / ATL / CLX 등 자주 사용되는 거의 모든 클래스 라이브러리들을 모두 포함하고 있다. (물론 이전 버전인 Turbo-C와 Borland C++의 런타임 라이브러리(RTL)도 여전히 포함하고 있다.) 다른 어떤 C++ 컴파일러 제품에서도 볼 수 없는 광범위한 지원라고 할 수 있다.
더욱이 C++Builder는 최신 버전의 Delphi 컴파일러도 내장하고 있으므로 Delphi로 작성된 소스파일을 C++ 소스파일과 함께 단일 프로젝트에서 포함시켜 사용할 수 있다. 뿐만 아니라 Delphi용으로 개발된 다양한 컴퍼넌트들도 C++Builder에서 거의 모두 사용가능하다. 내장된 VC++ Project Conversion Wizard는 그 이름처럼 Visual C++로 작성된 프로젝트를 불러와서 C++Builder 프로젝트로 변환해준다. C++Builder에는 Visual C++의 MFC 라이브러리가 라이선스되어 포함되어 있으므로, 이렇게 불러온 Visual C++ 프로젝트를 컴파일하는데 아무런 문제가 없다. (불러온 프로젝트가 ANSI/ISO 표준을 따른다고 가정할 경우 말이다.)

1.2 RAD 개발툴로서의 C++Builder

C++Builder가 정말 뛰어난 것은, 위에서도 간단히 설명했듯이 표준 C++ 언어의 바탕 위에 RAD(Rapid Application Development) 개발 방식을 적용했다는 점이다. C++Builder의 RAD 개발 방법은 높은 생산성을 보장하고 컴퍼넌트화를 통해 코드를 재사용율을 극대화할 수 있게 하여 UI를 구현하는데 있어 개발자의 반복적인 잔손질을 줄여준다. 이렇게 코드 기반과 RAD 기반 양쪽을 동시에 지원하는 특징을 볼랜드에서는 Two-Way tool이라는 말로 표현하는데, 상황과 필요에 따라 생산성과 성능을 고려하여 개발자가 선택할 수 있게 해주므로 프로젝트 수행시 유연성이 높다. 이런 특징은 비단 C++Builder 뿐 아니라 Delphi와 JBuilder 등 볼랜드의 모든 개발툴에서 볼 수 있는 특징이기도 하다.

이러한 강력한 RAD 개발 기능을 바탕으로, C++Builder는 데이터베이스 애플리케이션 개발에도 높은 생산성을 보여준다. Delphi에 포함된 것과 같은 다양한 데이터베이스 컴퍼넌트들을 포함하고 있으며, Oracvle, DB2, SQL Server, Infomix, Sybase, InterBase 등 많이 사용되는 RDBMS들을 위한 BDE(Borland Database Engine) 외에도 OBDC, ADO와 같은 마이크로소프트의 데이터베이스 기술들도 지원한다.

1.3 웹 및 분산 개발 지원

C++Builder에는 Web Broker라고 하는 강력한 웹 개발 방법이 지원되고 있다. 이것은 CGI, WinCGI, ISAPI/NSAPI를 컴퍼넌트화한 단일 프로그래밍 인터페이스로서, ASP나 JSP, 자바 서블릿 등에서 사용되는 객체 모델과 대단히 유사한 구조를 가지고 있으면서도 오버헤드를 최소화시킨 강력한 웹 애플리케이션을 개발할 수 있게 해준다.

C++Builder는 분산 개발에도 대단히 강력한 지원을 제공하고 있는데, COM/DCOM 개발은 물론, CORBA 개발을 위한 많은 기능들을 포함하고 있다. 게다가 CORBA ORB 미들웨어의 표준이라고 할 수 있는 볼랜드

VisiBroker 4.5를 개발자 라이선스로 포함하고 있어 C++ 환경에서라면 CORBA 개발을 위한 최상의 개발툴이라고 할 수 있다. 또한 마이다스(MIDAS)라고 불리는 C++Builder/Delphi 특유의 고성능 미들티어 기술을 포함하고 있다. (C++Builder 6에서는 DataSnap 이라는 이름으로 바뀌었다.)

1.4 강력한 디버깅 도구 - Code Guard

C++Builder 5 이상에는 일반적으로 흔히 사용되는 기본적인 디버깅 도구들 외에, Code Guard라고 하는 강력한 툴이 기본으로 탑재되어 있다. 이것은 런타임 디버깅으로 유명한 바운즈 체커와 유사한 도구로서, 메모리와 리소스 리크, C++Builder의 라이브러리와 Win32 SDK 호출 등을 실시간으로 감시한다. Code Guard는 런타임 디버깅이 진행중인 애플리케이션에서 에러들을 별도의 로그파일(.cgl)에 기록하며, Configuration 유틸리티를 이용하여 세세하게 설정을 할 수 있다.

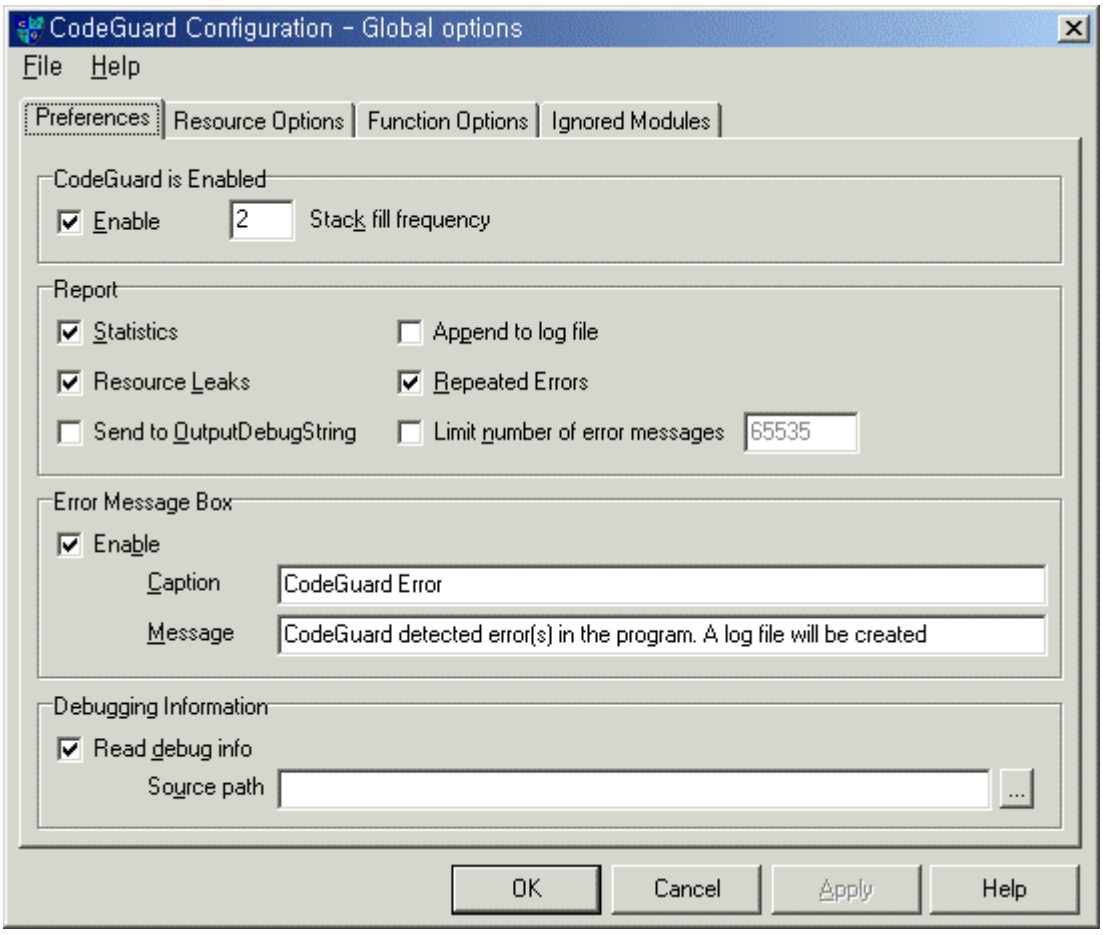


그림1. C++Builder에 포함된 Code Guard 런타임 디버깅 도구

2. BizSnap을 통한 웹서비스 개발 지원

BizSnap은 C++Builder 6에 포함되어 있는 웹서비스/SOAP 개발을 위한 플랫폼이다. BizSnap에서 지원하는 웹서비스 기능은 웹서비스 자체를 생성하는 서버 측과 존재하는 웹서비스 서버에 연결하여 임포트해오는 클라이언

트 측으로 나누어진다. 웹 서비스 서버는 기술적으로 C++Builder 기존 버전에도 존재하던 Web Broker를 기반으로 하여 그 위에 웹 서비스 서버를 위한 세가지 전용 컴퍼넌트를 얹은 것이지만, 이러한 구현에 대해서는 프로그래머는 거의 알 필요가 없으며 웹서비스의 실질적인 로직에만 신경을 쓰면 된다. 그럼, 서버와 클라이언트로 나누어 각각 아주 간단한 예제 하나씩을 들어 C++Builder 6에서 웹 서비스를 생성하거나 이용하는 것이 얼마나 쉽고 간단한지 알아보자.

2.1 웹서비스 클라이언트 만들기

이제, 간단한 웹서비스 클라이언트를 만들어보면서 C++Builder 6에서 지원하는 웹서비스 클라이언트 프로그래밍이 얼마나 간단한지 알아보자. 우리는 여기서 비주얼스투디오 닷넷으로 만들어진 메일 전송 웹서비스를 이용하여 메일을 전송하는 클라이언트 예제를 만들어볼 것이다. C++Builder 6로 만든 서버를 임포트해도 되겠지만, 웹서비스의 중요한 특징 중의 하나가 호환성(interoperatability)이므로 닷넷 기반의 웹서비스를 임포트해보는 것이다. 이 웹서비스의 WSDL URL은 다음과 같다.

http://www.xml-webservices.net/services/messaging/smtp_mail/mailsender.asmx?WSDL

먼저, 메뉴에서 File|New|Application을 차례로 선택하여 빈 프로젝트를 하나 만든다. 그런 후, File|New|Other…를 선택하여 New Items 다이얼로그를 띄운다. 여기서 WebService 탭을 선택한 후, 화면에 나타난 리스트에서 Web Service Importer를 선택하고 OK 버튼을 클릭한다.

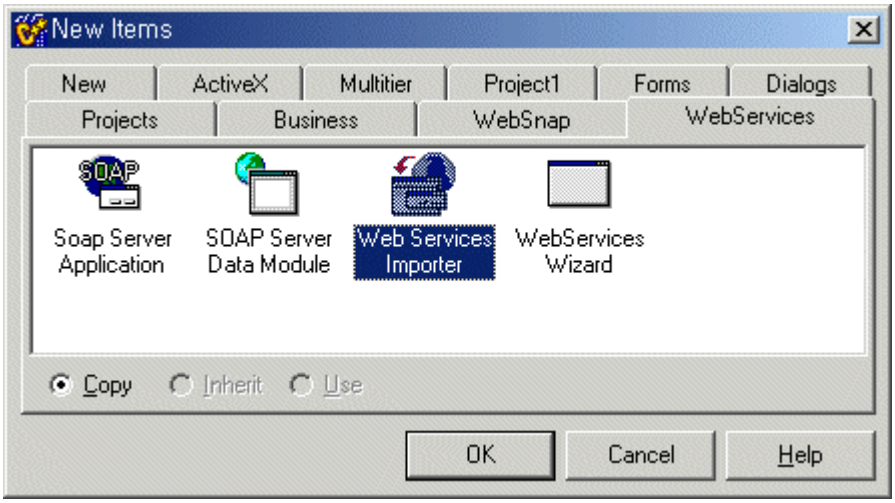


그림2. 웹서비스 클라이언트를 만들기 위해 Web Service Importer 선택

그러면 다음과 같은 다이얼로그가 나타날 것이다. 여기서 위의 WSDL URL을 입력하고 Next를 클릭한다.

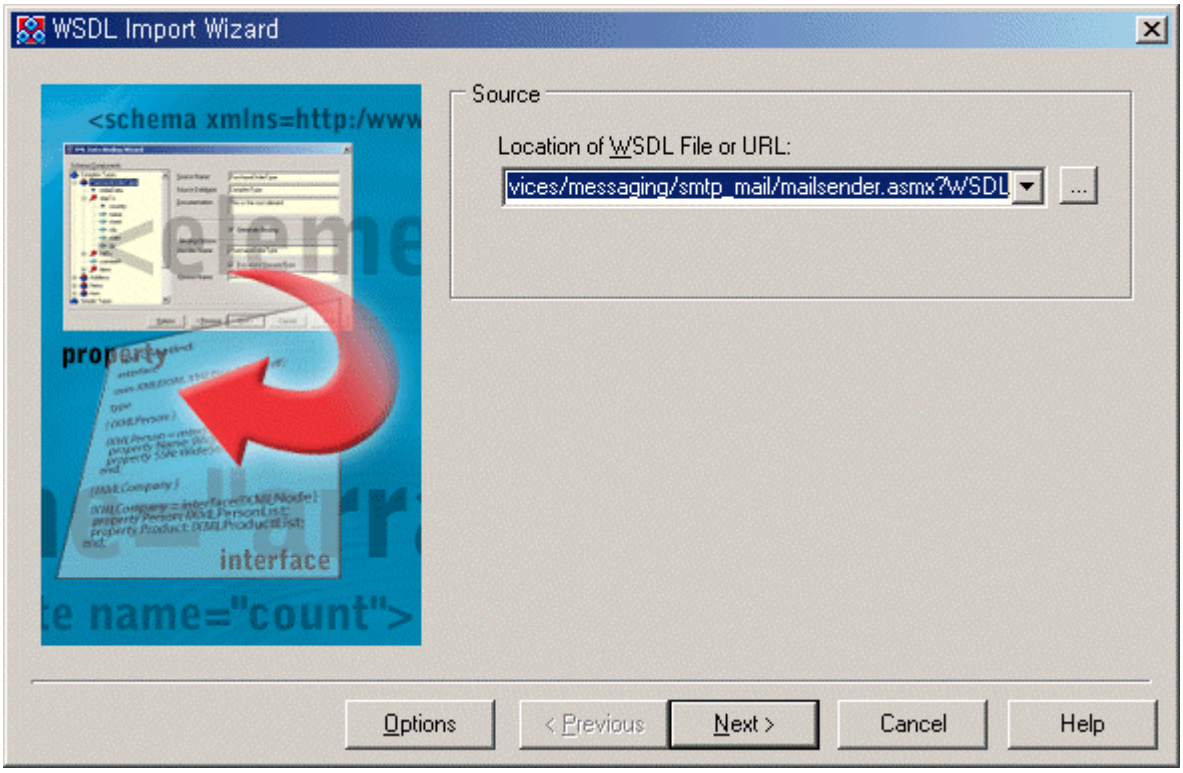


그림3. WSDL Import Wizard

그러면 마법사의 마지막 단계로 생성될 코드를 보여주면서 Finish 버튼이 활성화된다. Finish 버튼을 누르면 마법사가 종료되고 생성된 코드가 C++Builder의 코드에디터에 나타난다. 생성되는 코드는 다음과 같다. (주석은 생략)

리스트1. mailsender.h

```
#ifndef  __mailsender_h__
#define  __mailsender_h__
#include <System.hpp>
#include <InvokeRegistry.hpp>
#include <XSBuiltIns.hpp>
#include <SoapHTTPClient.hpp>

__interface  INTERFACE_UUID("{4D52A43C-3295-4904-AA2C-2590F66C9243}")  MessageSoap  :
public IInvokable
{
public:
   virtual  AnsiString  SendSimpleMail(const  AnsiString  ToAddress,  const  AnsiString
FromAddress, const AnsiString Subject, const AnsiString Message) = 0;
   virtual  AnsiString  SendHTML_Mail(const  AnsiString  ToAddress,  const  AnsiString
FromAddress, const AnsiString Subject, const AnsiString HTML_Message) = 0;
};
```

```
typedef DelphiInterface MessageSoapIntf;
#endif // __mailsender_h__
```

리스트2. mailsender.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "mailsender.h"
static void RegTypes()
{
  /* MessageSoap */
      InvRegistry()->RegisterInterface(__delphirtti(MessageSoap),      "http://www.xml-
webservices.net/services/messaging/smtp_mail/", "utf-8");
   InvRegistry()->RegisterDefaultSOAPAction(__delphirtti(MessageSoap),  "http://www.xml-
webservices.net/services/messaging/smtp_mail/%operationName%");
  InvRegistry()->RegisterInvokeOptions(__delphirtti(MessageSoap), ioDocument);
}
#pragma startup RegTypes 32
```

다음으로, 처음 프로젝트를 생성했을 때 함께 생성된 빈 폼을 들여다보자. 이 빈 폼의 유닛(.cpp 파일)의 상단에 위의 mailsender.h 파일을 인클루드시킨다. 그리고, 폼으로 돌아가서 컴퍼넌트 팔레트의 WebService 탭의 HTTPRIO 컴퍼넌트를 하나 놓는다. 그리고 에디트와 레이블 몇 개를 다음과 같이 배치하여 메일 전송을 위한 폼을 만든다.

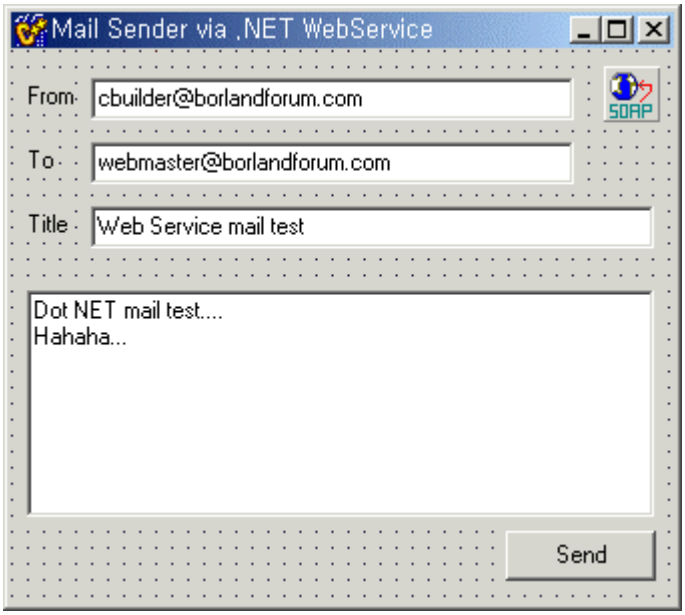


그림4. 간단한 메일전송 폼의 UI

HTTPRIO1 컴퍼넌트를 클릭하고, WSDLLocation 프로퍼티에 아까 입력했던 WSDL URL을 그대로 입력하자. 그

러면 Service와 Port 프로퍼티가 각각 “Message”와 “MessageSoap”로 자동으로 세팅될 것이다. 그대로 놔두고, Send 버튼을 더블클릭해서 클릭이벤트 핸들러를 만들고 다음과 같이 코딩한다.

리스트 3. Send 버튼의 클릭 핸들러

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    MessageSoapIntf InterfaceVariable;
    HTTPRIO1->QueryInterface(InterfaceVariable);
    if(InterfaceVariable)
        InterfaceVariable->SendSimpleMail(Edit1->Text, Edit2->Text, Edit3->Text, Memo1->Lines->Text);
}
```

이것으로 모든 절차가 끝났다. F9 키를 눌러 컴파일하고 실행해보자. 화면에 나타난 프로그램의 각 에디트를 생각나는대로 적당히 채워넣고(물론 To 메일주소는 받아볼 수 있는 주소로) Send를 클릭해본다. 의도한 대로 메일이 올 것이다. (본문에 메일을 보낸 웹서비스의 홍보문이 추가되는 것을 제외하면.)

위의 웹서비스 외에도, <http://www.xmethods.net> 사이트에 가보면 닷넷 외에도, 아파치, 델파이, 카일릭스, PearlEx 등 수많은 웹서비스 개발방법으로 구현한 다양한 웹서비스들이 테스트를 위해 준비되어 있다.

2.2 웹서비스 서버 만들기

웹서비스 서버를 만드는 것도 그리 어렵지 않다. 사실 웹 서비스 서버측에서 어떤 기능을 구현하려고 하느냐에 따라 소스가 엄청나게 복잡해질 수 있으므로, 여기서는 단순히 이름을 입력받고 거기에 대해 “Hello!” 라고 간단한 인사를 하는 고전적인 예제를 웹서비스로 구현해보자.

먼저, 메뉴에서 File|New|Other…를 선택하여 New Items 다이얼로그를 띄운다. 여기서 WebService 탭을 선택한 후, 화면에 나타난 리스트에서 SOAP Server Application을 선택하고 OK 버튼을 클릭한다.

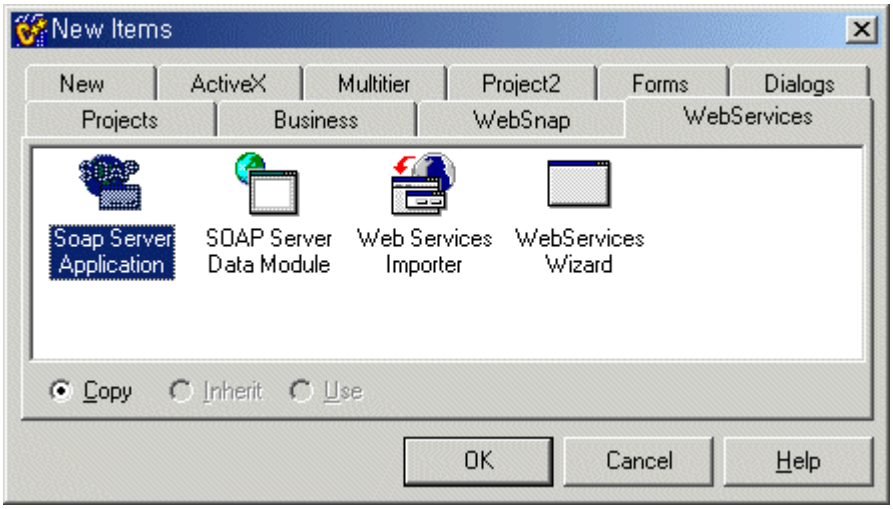


그림5. 웹서비스 서버를 생성하기 위해 SOAP Server Application 선택

그러면 생성할 서버 애플리케이션의 종류를 물어보는 다이얼로그가 나타날 것이다. 여기서는 CGI Stand Alone executable을 선택하기로 하자.

여기서 C++Builder는 새 SOAP 서버 프로젝트를 자동으로 생성해준다. 프로젝트에는 다음과 같은 웹모듈 하나만 포함되어 있을 것이다.

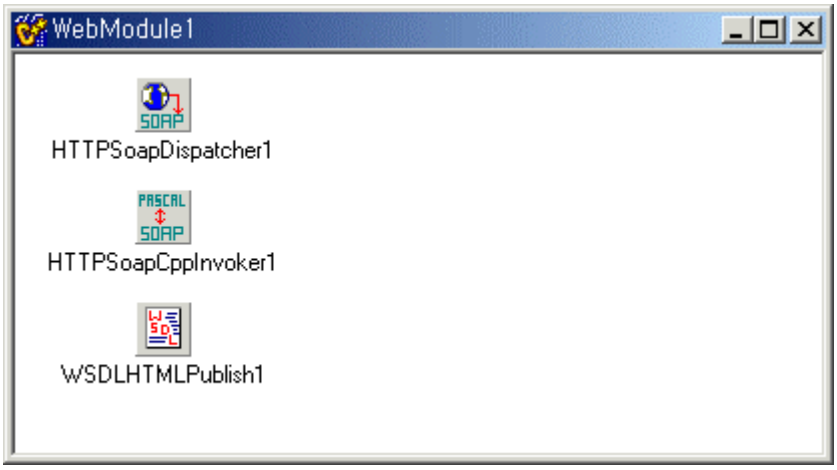


그림6. 자동으로 생성된 웹서비스의 웹모듈

사실, 이 웹모듈과 프로젝트는 이전 버전의 C++Builder에서도 있었던 Web Broker에서 생성해주는 프로젝트 템플릿과 완전히 동일하며, 단지 거기에 웹 서비스를 위한 세개의 컴퍼넌트를 자동으로 올려준 것에 불과하다. 이 세개의 컴퍼넌트가 SOAP 관련 기능의 거의 대부분을 하게 되며, 사실상 아주 특수한 경우가 아니라면 웹서비스를 개발하는 프로그래머가 이 웹모듈 유닛을 손댈 필요는 전혀 없다. 프로그래머가 손을 댈 부분은, 다음의 인터페이스 유닛이다.

위와 같은 프로젝트를 생성한 직후에, C++Builder는 “Create Interface for SOAP module?” 라고 물어올 것이다. 여기서 Yes 버튼을 클릭하고 넘어간다. 그러면 다시 다음과 같은 다이얼로그가 나타난다. 프로젝트 이름과 동일하게 HelloWorld라고 입력하고 OK를 클릭한다. (만약 이 단계에서 No를 클릭하고 넘어갔다면 다음에 File|New|Other…메뉴를 통해 New Items 다이얼로그에서 WebService Wizard를 선택하면 아래 다이얼로그가 나타난다.)

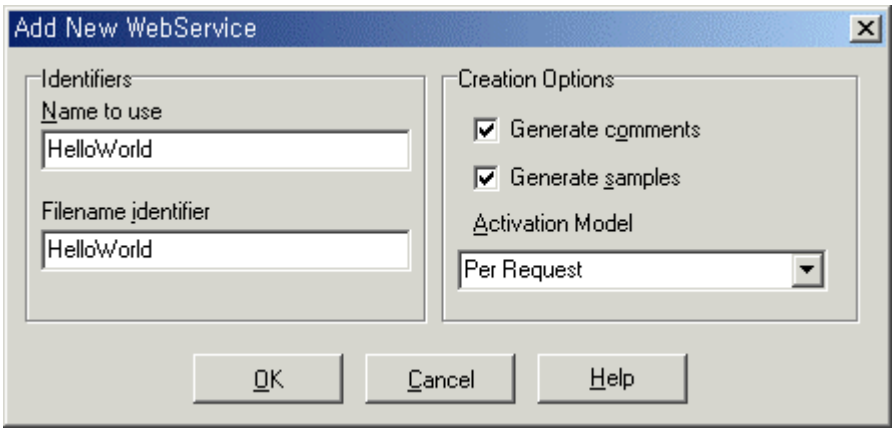


그림7. 수동으로 웹서비스 인터페이스를 추가하기 위한 New WebService 윈도우

그러면 C++Builder는 다시 Project1Impl.cpp라는 새로운 소스 하나를 더 생성해준다. 이제 이 소스에 코드를 추가하기 시작한다.

리스트 4. HelloworldImpl.h

```
#ifndef  __HelloworldImpl_h__
#define  __HelloworldImpl_h__

#include <System.hpp>
#include <InvokeRegistry.hpp>
#include <XSBuiltIns.hpp>
#include <Types.hpp>

__interface  INTERFACE_UUID("{E7BA0B33-9919-4274-B911-4C7A3ED7C3AF}")  Helloworld  :
public IInvokable
{
public:
  virtual AnsiString Hello(const AnsiString UserName)= 0; // 실제로 코딩해야 할 부분
};
typedef DelphiInterface HelloworldIntf;

#endif // __Helloworld_h__
```

리스트 5. HelloworldImpl.cpp

```
#include <vcl.h>
#pragma hdrstop

#if !defined(__HelloworldImpl_h__)
#include "HelloworldImpl.h"
#endif

class THelloworldImpl : public TInvokableClass, public Helloworld
{
public:
  AnsiString Hello(const AnsiString UserName); // 실제로 코딩해야 할 부분

  /* IUnknown */
  HRESULT STDMETHODCALLTYPE QueryInterface(const GUID& IID, void **Obj)
                    { return GetInterface(IID, Obj) ? S_OK : E_NOINTERFACE; }
  ULONG STDMETHODCALLTYPE AddRef() { return TInterfacedObject::_AddRef();  }
  ULONG STDMETHODCALLTYPE Release(){ return TInterfacedObject::_Release(); }
```

```
};


AnsiString TProject1Impl::Hello(const AnsiString UserName) // 실제로 코딩해야 할 부분
{
    return "Hello " + UserName + "!";
}

static void RegTypes()
{
  InvRegistry()->RegisterInterface(__delphirtti(Helloworld), "", "");
  InvRegistry()->RegisterInvokableClass(__classid(THelloworldImpl));
}
#pragma startup RegTypes 32
```

이제 웹서비스 서버가 완성되었다. 컴파일해서 얻은 것은 앞에서 선택한 대로 CGI 애플리케이션이므로, 마이크로소프트 IIS나 아파치 웹서버에 올려서 테스트해봐야 한다. 클라이언트를 만드는 절차는 앞서 구현했던 메일 전송 클라이언트 예제와 동일하며, WSDL URL은 다음과 같이 해야 한다.

http://localhost/cgi-bin/Project2.exe/wsdl/Helloworld

박스기사 : C++Builder 6로 만든 체스 웹 서비스 서버

그림8. 체스 웹서비스 서버

http://www.danmarinescu.com/에 가면 100% C++Builder 6로 만든 체스 서버가 운영중이다. 이것은 서버측에서 C++Builder 6로 컴퓨터에 의한 체스 로직을 웹 서비스로 구현하고, 클라이언트에서는 Delphi 6를 이용하여 이 웹서비스를 임포트하는 체스 클라이언트를 구현한 것이다. (클라이언트 측의 소스가 Delphi로 되어있긴 해도 맘만 먹으면 C++Builder에서 컴파일시키는 데는 큰 무리는 없을 것이다.) 참고하기 바란다.

3. WebSnap과 기타 웹 개발 기능의 강화

3.1 WebSnap

6 버전이 나오기 전까지, C++Builder에서는 Web Broker라는 웹 개발 방법을 지원하고 있었다. 그럼, Web Broker라는 꽤 괜찮은 웹 개발 방법이 있는데도 WebSnap이라는 새로운 방법을 추가한 이유는 뭘까? 사실 WebSnap은 그 기반의 많은 부분을 Web Broker와 공유하고 있다. 하지만 그 개발 방식은 정반대라고 할 정도로 완전히 다르다.

Web Broker 방식에서는 요청을 받아와서 응답을 하기까지 거의 절차적으로 처리한다. 또한 하나의 웹 애플리케이션 내에서는 단 하나의 웹모듈이 모든 액션을 처리하는 방식으로 되어있다. 생성하는 웹 페이지는 코드에서 취급하는 데이터일 뿐, 하나의 로직으로 분리하여 처리하기는 거의 불가능하다. 간단히 말해, 구조는 아주 간단하지만, 하나의 소스 모듈에서 거의 모든 처리가 이루어지고 페이지 디자인이 코드의 일부로서 취급됨으로써 개발자들간, 그리고 개발자와 디자이너 사이의 팀 프로젝트가 쉽지 않은 구조로 되어있다.

WebSnap의 구조에서는 이와 달리 비즈니스 로직과 프리젠테이션 로직이 분리되어 있다. 모든 페이지 생성 루틴은 각각의 소스 모듈로 나누어지며, 각각의 소스모듈마다 자동으로 생성되는 HTML 페이지는 소스 모듈과 동등한 프리젠테이션 로직으로서 존재하게 된다. 이러한 HTML 페이지를 위한 기능의 추가로서 C++Builder 자체에 신택스 하이라이팅이 되는 html 에디트 기능까지 포함되어 있다. 또한 WenSnap에서는 JScript와 VBScript 등 서버 사이드 스크립트를 이용하여 유연하게 기능을 확장할 수 있다.

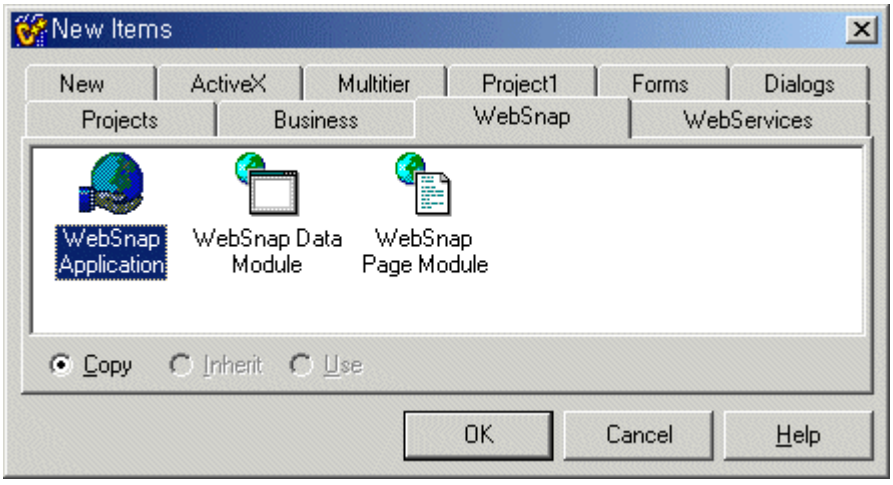


그림9. WebSnap 애플리케이션을 만들기 위한 마법사 선택

3.2 Web Application Debugger

웹 애플리케이션 디버거는 개발단계에서의 디버깅 편의를 위해 C++Builder 6에 새로이 추가된 유틸리티이다. Web App Debugger를 이용하여 디버깅을 하려면 먼저 웹 애플리케이션을 생성할 때 Web App Debugger executable이라는 새로운 형식을 선택해야 하는데, 이 형식의 웹 애플리케이션은 실제 서비스에는 사용할 수는 없지만 매우 강력한 웹 애플리케이션 디버깅 방법을 제공한다. 일단 개발 중에는 이 Debugger executable 형식을 이용하여 디버깅을 모두 마친 후, 실제 서비스를 위해서는 기존의 CGI나 ISAPI/NSAPI 등의 형식으로 변경이 가능하다.

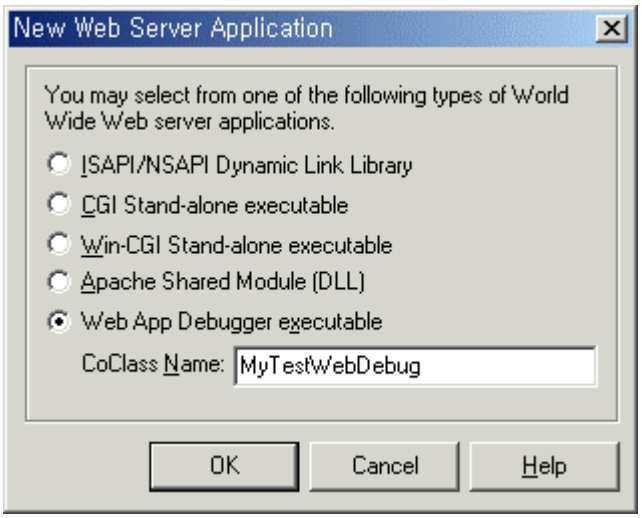


그림10. 런타임 디버깅을 위한 웹서버 애플리케이션의 타입 설정

Web App Debugger executable 디버거 형식으로 생성된 프로젝트에는 웹모듈 외에 빈 폼이 하나 포함되어 있다. 이 Debugger executable 형식으로 컴파일된 애플리케이션을 실행시키면 빈 폼만 나타나며, 첫번째 실행때 그 자신을 COM 서버로서 등록시킨다.

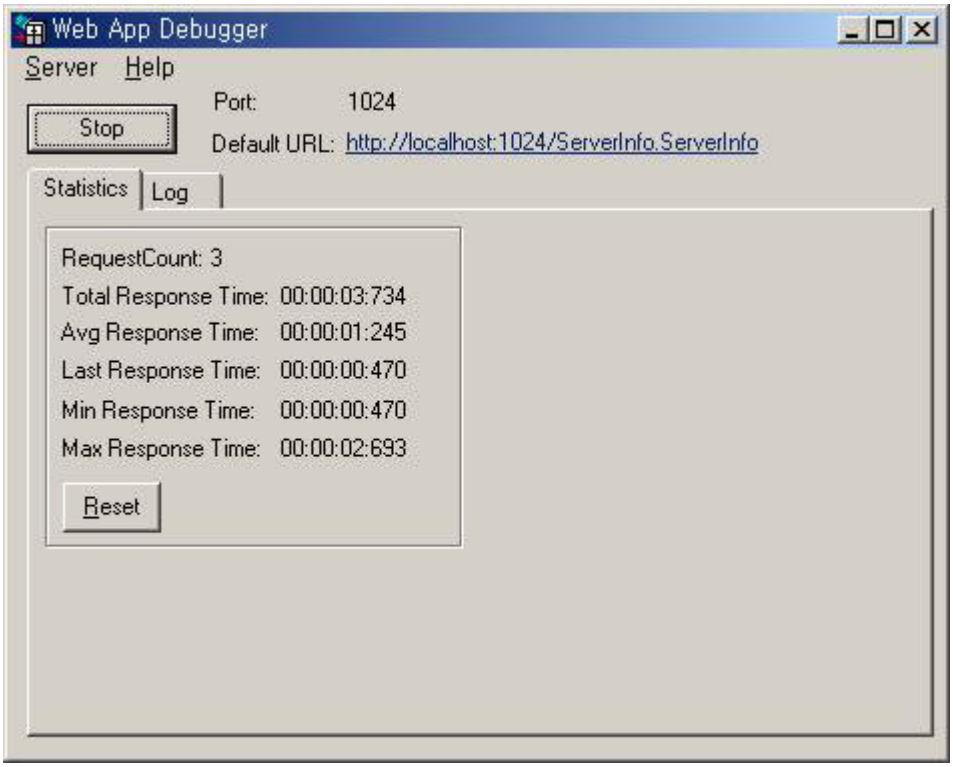


그림11. Web App Debugger

이 디버깅 기능을 이용하려면 웹 서버 애플리케이션 소스에서 브레이크 포인트를 걸어놓고 디버거를 실행시킨 후 윈도우에 나타난 URL을 클릭하면 되는데, 이 때 디폴트 브라우저가 실행되면서 웹 서버 애플리케이션이 호출되고 미리 걸어놓은 브레이크 포인트에서 실행이 멈추게 된다.

4. 그밖의 새롭거나 강화된 기능들

4.1 윈도우/리눅스 크로스플랫폼 개발 가능

기존의 VCL 대신 C++Builder 6에 새롭게 포함된 CLX(Component Library for Cross-platform) 라이브러리를 이용하여 개발하면 곧 출시될 리눅스용 C++ 개발툴(아직 정식 제품 이름이 확정되지 않았다)과 소스 수준에서 100% 호환된다. (물론 Win32 API 등 시스템에 의존하는 코드를 사용할 경우는 제외된다) CLX는 이미 윈도우 플랫폼의 Delphi 6와 리눅스 플랫폼의 Kylix 1, Kylix 2를 거쳐 안정화된 크로스플랫폼을 위한 클래스 라이브러리로서, 윈도우와 리눅스 각 플랫폼에 의존적인 부분을 클래스로 래핑함으로써 동일한 프로그래밍 인터페이스로 양쪽 플랫폼 모두에서 사용가능한 소스를 개발할 수 있게 해준다. CLX는 오픈소스 크로스플랫폼 라이브러리인 Qt를 기반으로 한 것으로, VCL과 아주 유사한 구조로 설계되었으므로 기존의 VCL 기반 프로젝트를 수정하여 크로스플랫폼 CLX 프로젝트로 전환하는 것도 가능하다. 현재 볼랜드의 리눅스용 C++ 개발툴은 공개 필드테스트를 진행하고 있는 중이며, 올해 2/4분기에 출시될 예정이므로 이 제품의 발표 전후에 다시 리뷰할 기회를 기다려보기로 하자.

4.2 DataSnap, dbExpress

DataSnap은 기존 버전의 MIDAS가 강화되면서 이름이 바뀐 것이다. 기존 MIDAS의 기능들은 거의 그대로이지만, 몇 개의 새로운 컴퍼넌트(TConnectionBroker, TSharedConnection, TLocalConnection)가 생겼으며 TClientDataSet이 DataAccess 탭으로 옮기면서 일반화되는 듯한 모습을 보이고 있다.

C++Builder 6의 데이터베이스 프로그래밍에서 확실히 변화된 모습을 보여주고 있는 것은 dbExpress이다. dbExpress는 기존의 BDE를 대체하기 위한 가볍고 빠른 크로스플랫폼 데이터베이스 기술로서, 단방향 커서만을 지원함으로써 기존의 BDE보다 속도면에서 획기적인 향상을 가져왔다. 또한 배포를 위한 파일들의 크기와 갯수를 대폭 줄임으로써 기존의 개발자들의 불평을 들어왔던 BDE의 대체 역할을 확실히 할 것으로 보인다. 현재 dbExpress에서 지원하는 RDBMS는 Oracle, DB2, Informix, InterBase, MySQL의 다섯 가지이며, 앞으로도 계속 지원 DB가 늘어날 것으로 보인다. (실제로 Informix는 이전의 Delphi6와 Kylix2에서는 지원되지 않았던 것이다) 이러한 변화와 함께, 오랫동안 기본 DB 기술로 DataAccess 탭을 차지하고 있었던 BDE 컴퍼넌트들은 BDE라는 새로운 탭으로 내보내져 C++Builder 6에서 지원하는 다른 데이터베이스 기술들(dbExpress, ADOExpress, IBExpress, DataSnap) 들과 동등한 레벨의 선택가능한 기술로 내려앉게 되었다.

그림12. dbExpress와 기타 C++Builder DB 컴퍼넌트 군

4.3 Code Insight

위의 모든 새롭거나 강화된 기능들에 무관심한 C++Builder 개발자라고 해도 환호할 수밖에 없는 획기적인 개선점도 하나 있다. 오랫동안 C++Builder 개발자들을 괴롭혀왔던 Code Insight 기능이 놀랍도록 개선된 것이다. 새로운 Code Insight 기능은 이전 버전과는 그야말로 비교도 되지 않을 정도로 빨라졌으며, 델파이와 비교될 정도의 수준이다. 또한 Code Insight 팝업 윈도우가 리사이즈 가능하며 리스팅된 멤버의 종류에 따라 다양한 컬러로 표시해준다.

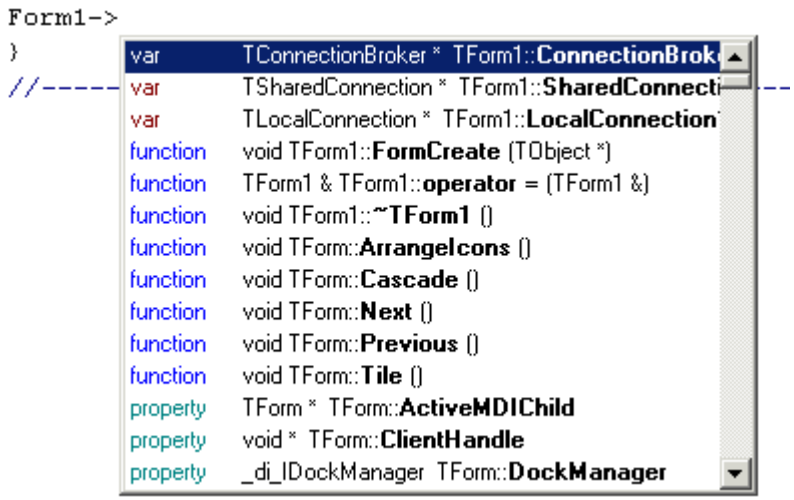


그림13. 놀랍도록 개선된 Code Insight

4.4 Object TreeView와 Data Diagram

오브젝트 트리뷰와 데이터 다이어그램은 C++Builder 5 버전에서 데이터모듈(에서 상속받는 객체)에만 나타났던 것인데 6 버전으로 업그레이드되면서 일반적인 폼에도 나타나게 된 것이다. 오브젝트 트리뷰는 현재 활성화된 모듈의 컴퍼넌트들의 논리적인 연결 관계를 보여주는 트리 형식의 윈도우다. 이전 버전에서는 데이터베이스 컴퍼넌트들과 같이 복잡한 관계로 얽혀있는 컴퍼넌트들의 관계를 한눈에 보여준다. 데이터 다이어그램은 코드 에디터 아래쪽 StatusBar에서 Diagram 탭을 선택하면 나타난다. 이것은 오브젝트 트리뷰에 나타난 객체들을 드래그&드롭해서 개발자가 스스로 시각적으로 논리적인 연결 다이어그램을 그릴 수 있게 해주는 것이다.

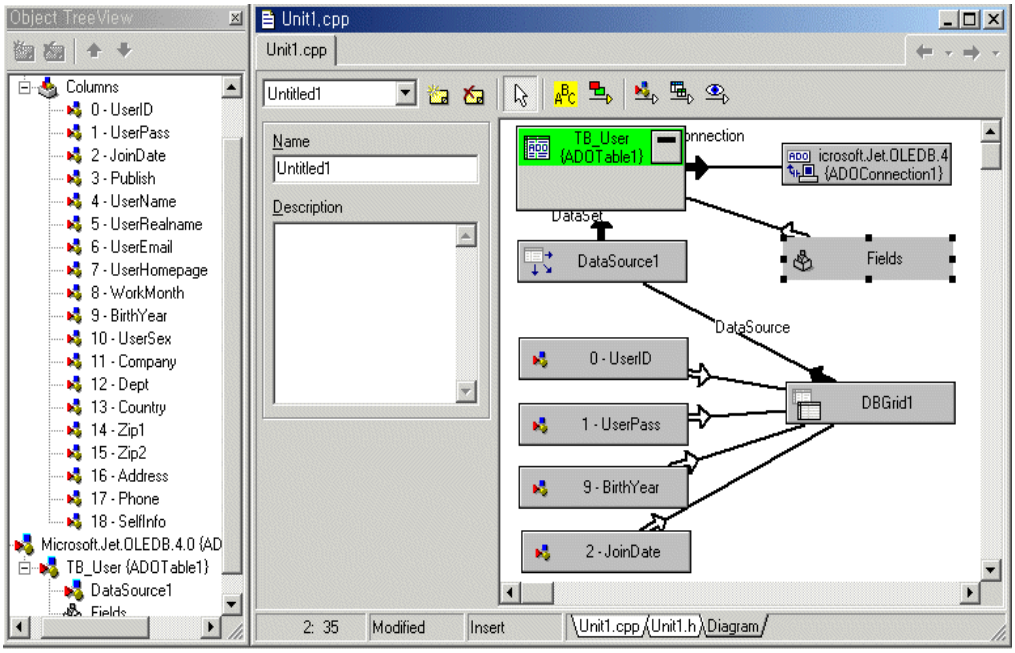


그림14. Object TreeView와 Data Diagram

4.5 그외의 IDE 개선 사항

Build Tools는 외부 컴파일러 등의 툴을 등록시켜서 사용하는 것으로, 특정 파일 확장자들(예를 들어 펄을 위한 .pl)을 여기에 등록해두면 프로젝트 컴파일 때 실행이 되고, 또 프로젝트 매니저의 팝업메뉴에 등록이 되어 수동으로 호출할 수도 있게 된다. 또 프로젝트 매니저 자체도 여러가지로 개선이 되었는데, 팝업메뉴의 Edit Local Options를 선택하면 프로젝트 옵션 중 컴파일러, 코드 가드 등의 설정을 해당 소스파일에만 한정적으로 지정할 수 있다. 또한 To-Do List가 추가되어 프로젝트에 대한 여러가지 개발자의 코멘트를 추가할 수도 있으며, 메인 메뉴에 Window 메뉴가 추가되어 MDI에서처럼 현재 열려있는 모든 창들을 편하게 관리할 수도 있다.

4.6 VCL의 변경 및 추가된 컴퍼넌트

언뜻 보기에는 C++Builder 6의 VCL 구조에는 큰 변화는 없어보이지만, 실제로는 많은 모듈이 새로 생기거나 기존의 모듈로부터 독립해 나오는 등 만많찮은 변화가 있었다. 이러한 구조적인 변화와 함께 전체 VCL 라이브러리의 크기가 증가함에 따라, 패키지를 포함하여 애플리케이션을 생성했을 때 상당한 정도의 크기가 증가한다.

컴퍼넌트 팔레트의 Additional 탭에 추가된 액션툴바 관련 컴퍼넌트들은 일반적인 메뉴-툴바 기반의 애플리케

이션을 만들 때 개발속도를 높여줄 수 있다. 콤보박스 형태의 컬러 선택 컴퍼넌트인 ColorBox, 오브젝트인스펙터와 유사한 컴퍼넌트인 ValueListEditor, 트리형태의 리스트를 보여주는 ComboBoxEx 등은 서드파티 컴퍼넌트에 대한 의존성을 상당히 줄여줄 수 있다. 이미 잘 알려져 있겠지만, 최근 몇 년 사이에 많은 C++Builder, Delphi 개발자들에게 각광받고 있는 인터넷 컴퍼넌트 라이브러리인 Indy가 기본으로 탑재된 것도 멋진 소식이다.

4.7 STL 지원 변경

C++Builder 5까지 기본 STL은 Rogue Wave였으나 6 버전에서는 STLPort로 기본 STL 구현이 바뀌었다. 볼랜드에 의하면 기본 STL 지원을 바꾼 것은 ANSI C++ 호환성을 높이기 위해서였다고 한다. 이전의 Rogue Wave 구현을 계속 쓰고 싶은 경우에는 프로젝트 옵션에서 설정하면 여전히 문제없이 동작한다.

C++Builder의 미래

자바와 C#의 불꽃튀기는 경쟁속에 C++은 역사속으로 사라지게 될까? 전혀 그럴 것 같지 않다. 자바와 C#이 대형 소프트웨어 벤더들의 필요에 의해 전략적으로 만들어진 도구인데 비해, C++은 20년의 역사를 거치면서 수많은 개발자들이 스스로 선택하고 진화시켜온 언어이다.

C++Builder를 위한 볼랜드의 2002년 로드맵은 숨가쁘기 짝이 없다. 이번 리뷰에서 다루고 있는 2월 5일자로 발표된 윈도우용 C++Builder 6를 필두로 해서, 2/4분기에는 리눅스용 C++Builder 호환 C++ 개발툴(제품명 미정), 그리고 모바일 환경을 위한 C++Builder MobileSet이 출시를 기다리고 있다. 또한, 2002년의 하반기에는 닷넷 플랫폼을 위한 C++Builder가 진행될 예정이다. 물론 각각의 플랫폼용으로 출시된 개발툴들은 모두 C++Builder와 소스 레벨에서 호환될 전망이다. 윈도우/리눅스/닷넷/모바일의 네가지 플랫폼을 아우르는 서로 호환되는 하나의 프로그래밍 인터페이스, 이것이 오랫동안 C++ 개발자들의 지지를 받아온 볼랜드가 진정한 개발툴 전문 벤더로서 C++ 개발자들을 위해 제시하는 명쾌한 해답이라고 할 수 있다.