

웹서버 어플리케이션의 제작

텔파이를 이용하여 실제 인터넷에서 사용할 수 있는 어플리케이션을 개발하는 방법에는 전통적인 CGI 형식의 프로그램을 개발하는 방법과 액티브 X 기술을 기반으로 하는 방법으로 크게 나누어 볼 수 있다. 여기에 CORBA 와 MTS 기반 기술을 미들웨어로 하여 보다 효과적인 CGI 또는 액티브 X 컨트롤을 제작할 수 있을 것이다.

이번 장에서는 이 중에서도 CGI 를 기반으로 한 웹 서버 어플리케이션을 작성하는 방법에 대해서 알아볼 것이다.

CGI 어플리케이션의 종류

텔파이에서 만들 수 있는 단순한 CGI 어플리케이션은 하드 코딩에 가깝다고 볼 수 있다. 실제로 텔파이의 GUI 디자인을 이용하여 그 폼을 그대로 표현할 수 있다면 좋겠지만, 이는 액티브 폼을 이용하지 않고서는 구현이 불가능하다.

텔파이에서 만들 수 있는 CGI 의 형식에는 다음과 같은 것들이 있다.

어플리케이션 종류	어플리케이션 객체	Request 객체	Response 객체
MS Server DLL (ISAPI)	TISAPIApplication	TISAPIRequest	TISAPIResponse
Netscape Server DLL (NSAPI)	TISAPIApplication	TISAPIRequest	TISAPIResponse
Console CGI Application	TCGIApplication	TCGIRequest	TCGIResponse
Windows CGI Application	TCGIApplication	TWinCGIRequest	TWinCGIResponse

표에서 보듯이 ISAPI 와 NSAPI 를 지원함으로써 양대 웹 서버의 API 를 모두 지원하며, 동시에 Console CGI 나 Windows CGI 도 지원하기 때문에, 텔파이를 CGI 개발 도구로 사용할 수 있다.

텔파이로 CGI 어플리케이션을 작성하는 이유

현재 웹 서버 어플리케이션을 제작하는 방법은 다음과 같이 크게 3 가지로 나눌 수 있다.

1. 웹 서버에서 지원되는 기본적인 스크립트를 사용하는 방법이다. 대표적으로 ASP, VB 스크립트와 자바 스크립트, Perl 등이 있다.
2. ISAPI 나 NSAPI 를 통한 CGI 어플리케이션을 제작하는 방법
3. 자바를 사용하여 서블릿이나 애플릿 형태로 개발하거나, 액티브 X 컨트롤의 형태로 클라이언트에서 동작하는 작은 프로그램을 제작하는 방법

물론, 한가지 방법으로 웹 서버의 모든 기능을 만족시킬 수도 있다. 그러나, 프로그램의 성능과 보안문제등 다양한 문제를 해결하기 위해서는 이러한 방법들을 혼용하여야 한다.

여기에서 CGI 로 제작되는 방식은 이미 구세대적인 방식인지는 모르나 현재 시점에서 DB 에 접근하여 작업하는 경우에는 이 방법이 가장 효율적이다.

텔파이는 이러한 CGI 를 제작하는 경우에 매우 효율적인 방법을 제공하는데, 그것은 CGI 의 4 가지 방식을 따로 구분할 필요 없이 한가지 방식으로 개발한 다음 필요한 내용을 링크하여 CGI 를 빌드하기만 하면 된다는 것이다.

기본적인 Console CGI 어플리케이션의 제작

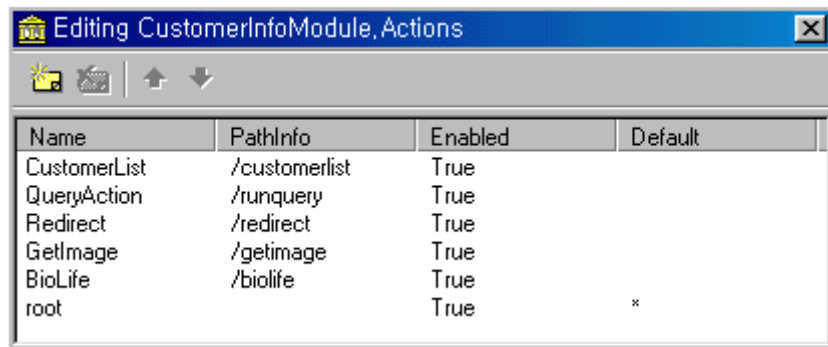
텔파이의 Demos 디렉토리의 WebServ 서브 디렉토리에 있는, Iservcgi 프로젝트를 기준으로 설명하도록 한다. 해당 소스를 컴파일한 다음 해당 프로그램을 IIS 의 CGI-Bin 디렉토리에 복사한 다음 해당 내용을 브라우저하여 보면, 텔파이에서 기본적인 데이터를 보여주는 것 뿐만 아니라 테이블의 표를 HTML 형식으로 표시할 수 있도록 지원할 수 있다는 것을 알 수 있을 것이다. 실제 CGI 어플리케이션을 적용한 이 예제를 브라우저를 이용하여 접근할 때, CGI 가 동작할 링크 부분에 커서를 가지고 가면 다음과 같은 형태의 코드가 나타나는 것을 볼 수 있을 것이다.

```
실행파일.exe/runquery?custono=1645
```

이 코드는 CGI 에 파라미터로 runquery 와 custono=1645 라는 코드를 전달한다는 의미이다. CGI 어플리케이션을 구동시키기 위해서는 실행 파일을 단독으로 수행하는 방법과 이렇게 파라미터를 이용하여 전달하는 방법이 있다.

이렇게 파라미터를 처리하는 방법을 Get 메소드라고 하는데, 여기에 대해서 먼저 알아보도록 하자.

실제 해당 소스의 웹 모듈을 살펴보면 여러 테이블과 쿼리가 있다는 것을 알 수 있다. 웹 모듈을 선택하고, Actions 프로퍼티의 ‘...’ 버튼을 클릭하면 다음과 같은 화면을 볼 수 있을 것이다.



Name	PathInfo	Enabled	Default
CustomerList	/customerlist	True	
QueryAction	/runquery	True	
Redirect	/redirect	True	
GetImage	/getimage	True	
BioLife	/biolife	True	
root		True	*

웹 모듈의 파라미터와 해당 Action 의 연결 상태를 알 수 있다. 개발자는 이 Action 에서 필요한 파라미터를 설정할 수 있고, 해당 파라미터와 연결되는 코드를 이벤트 추가하듯이 추가하는 것이 가능하다.

이제 해당 Action 부분의 코드가 어떻게 만들어졌는지 살펴 보자.

기본적으로 수행되는 이벤트는 root 라는 디폴트로 설정된 Action 으로, 이 Action 은 해당 CGI 가 동작할 때에 동작하는 이벤트이고, 여기에서 파라미터에 /runquery 가 전달되면 동작하는 코드는 QueryAction 이벤트이다. 그 밖에 다른 Action 도 많이 있지만, 이 중에서 CustomerList 와 QueryAction 에 대해서 알아보도록 하자.

먼저 WebModule1CustomerListAction 의 코드를 살펴보자.

```
procedure TCustomerInfoModule.WebModule1CustomerListAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := CustomerList.Content;
end;
```

해당 코드는 Response 객체의 Content 부분에 CustomerList.Content 를 전달하는 단순한 코드이다. 그렇다면 CustomerList 는 무엇일까 ? CustomerList 는 다음과 같이 선언되어 있다.

```
CustomerList: TPageProducer;
```

이 PageProducer 의 Content 를 Response 의 Content 에 넣는 단순한 코드로 일견 복잡해 보이는 HTML 화면을 만들어 낸 것이다.

CustomerList 페이지 프로듀서 컴포넌트는 필요한 HTML 파일을 가지고 있다가 해당 내용을 전달한다. 프로퍼티를 살펴보면 Strings 객체의 HTMLDoc 와 HTML 파일의 위치를 가리키는 프로퍼티인 HTMLFile 프로퍼티가 있다. 둘 중 하나의 값으로 내용을 유지한다.

개발자는 해당 내용을 HTMLDoc 프로퍼티에 직접 넣을 수 있으며, 해당 파일을 호출할 수

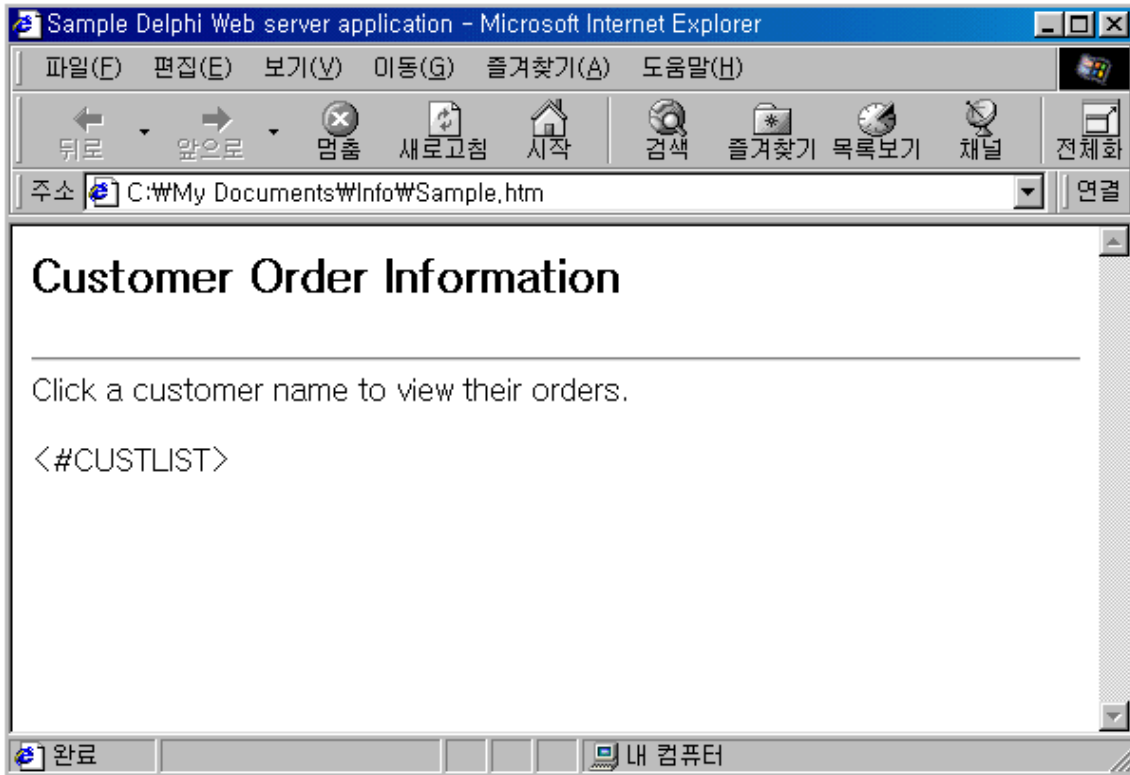
도 있다.

참고로, HTML 파일의 위치를 HTMLFile 프로퍼티에 지정하는 것이 나중에 프로그램을 수정할 때에 좀더 효율적으로 사용할 수 있다. 그러나, 이 방식은 수행 속도를 저하시키는 단점이 있다. 그러므로, 높은 수행 속도를 요구한다면 해당 내용을 직접 HTMLDoc 프로퍼티에 넣어서 사용하는 것이 좋다.

그렇다면, 해당 HTMLDoc 프로퍼티에 들어 있는 내용을 살펴보자.

```
<HTML>
<!------->
<!-- Copyright Inprise Corporation 1998 -->
<!------->
<HEAD>
<TITLE>Sample Delphi Web server application</TITLE>
</HEAD>
<BODY>
<H2>Customer Order Information</H2>
<HR>
Click a customer name to view their orders.<P>
<#CUSTLIST><P>
</BODY>
</HTML>
```

HTML 태그에 대해 조금만 알고 있다면 이 화면이 다음과 같이 보일 것이라는 것을 알 수 있을 것이다. 여기서 주의해서 보아야 하는 것은 중간에 있는 <#CUSTLIST> 부분이다. 이 부분은 일종의 스크립트로서 해당 부분을 페이지 프로듀서가 채우는 것이다.



해당 내용을 채우는 것은 OnHTMLTag 이벤트에서이다. 여기에 해당되는 코드는 다음과 같다.

```
procedure TCustomerInfoModule.CustomerListHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
var
  Customers: String;
begin
  if CompareText(TagString, 'CUSTLIST') = 0 then
  begin
    Customers := '';
    Customer.First;
    while not Customer.Eof do
    begin
      Customers :=
        Customers + Format('<A HREF="/scripts/%s/runquery?CustNo=%d">%s</A><BR>',
          [ScriptName, CustomerCustNo.AsInteger, CustomerCompany.AsString]);
      Customer.Next;
    end;
  end;
```

```
end;
ReplaceText := Customers;
end;
```

이 코드의 의미는 전달되는 태그의 값에서 해당 부분을 교체하는 것이다. TagString 부분으로 해당 태그를 읽어서 ReplaceText 영역에 해당 내용을 넣으면 해당 부분을 교체하게 된다. 그럼 변환한 내용은 무엇인가 ?

```
Customers :=
    Customers + Format('<A HREF="/scripts/%s/runquery?CustNo=%d">%s</A><BR>',
        [ScriptName, CustomerCustNo.AsInteger, CustomerCompany.AsString]);
```

이 부분은 필요한 부분의 링크를 만드는 코드인데, HTML 의 링크를 만드는 문장을 통하여 해당 파라미터와 코드 값을 주어 문장을 만들어 낸다.

이중에 ScriptName 은 수행시켜야 할 프로그램을 지정한 부분이다. 일반적으로 CGI 어플리케이션이 어느 곳에 위치할 지 모르기 때문에, 이를 고정시키면 매번 프로그램을 수정해야 한다. 그렇기 때문에, 이렇게 웹 모듈을 생성하는 시점에서 해당 프로그램의 위치를 읽어 내는 코드를 만드는 것이 좋다.

소스 코드를 잘 살펴보면 public 섹션에 ‘ScriptName: String;’이라고 선언된 부분을 찾을 수 있을 것이다. 그리고, 이 내용은 웹 모듈의 OnCreate 이벤트에서 해당 위치를 지정하도록 다음과 같이 작성되어 있다.

```
procedure TCustomerInfoModule.DataModule1Create(Sender: TObject);
var
    FN: array[0..MAX_PATH- 1] of char;
begin
    Customer.Open;
    BioLife.Open;
    SetString(ScriptName, FN, GetModuleFileName(hInstance, FN, SizeOf(FN)));
    ScriptName := ExtractFileName(ScriptName);
end;
```

코드는 이와 같다. 먼저 Customer 테이블 파일을 연 뒤에, GetModuleFileName 함수를 이용하여 현재 이 프로그램의 이름을 읽어 낸다. GetModuleFileName API 함수는 현재 프로그램의 인스턴스에 해당되는 프로그램의 시작 패스를 읽어서 FN 문자 배열에 저장한다. 그리고, ‘SetString(var s: string; buffer: PChar; len: Integer);’ 코드는 PChar 로 되어 있는

내용을 문자열 형태로 변환하여 저장하는 함수이다. 그 다음에 ExtractFileName 함수를 이용하여 원하는 패스와 프로그램 명을 분리한다.

이 코드를 이용하면 실제 수행 시의 프로그램의 위치를 저장하고, 링크를 생성하는 코드에서 해당 ScriptName 을 사용하게 된다. 여기서 생성되는 링크를 웹 브라우저에서 클릭하면 QueryAction 이 동작하게 된다.

앞서 CustomerList 페이지 프로듀서에 의해 생성되는 하이퍼 링크 태그는 다음과 같은 형태를 가진다.

```
<A HREF="/scripts/실행파일 패스/runquery?CustNo=고객 번호"> 고객의 회사 </A><BR>',
```

이를 클릭하면 파라미터로 전달 되는 것이 CustNo 필드에 대한 값이다. 그러므로, CustNo 뒤의 번호를 통하여 데이터에 대한 정보를 찾아다가 이를 보여주는 역할을 하는 것이 QueryAction 이다. 그러므로, PathInfo 의 내용이 파라미터로 날아오는 코드이다. 소스 코드는 다음과 같이 작성되어 있다.

```
procedure TCustomerInfoModule.WebModule1.QueryActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  if Customer.Locate('CustNo', Request.QueryFields.Values['CustNo'], []) then
  begin
    CustomerOrders.Header.Clear;
    CustomerOrders.Header.Add('The following table was produced using
      a TDataSetTableProducer.<P>');
    CustomerOrders.Header.Add('Orders for: ' + CustomerCompany.AsString);
    Response.Content := CustomerOrders.Content;
  end
  else
    Response.Content := Format('<html><body><b>Customer: %s not found</b></body></html>',
      [Request.QueryFields.Values['CustNo']]);
end;
```

약간 복잡하지만 차근차근 살펴 보자. Customer.Locate 메소드를 이용하여 넘어온 파라미터 번호에 해당되는 레코드를 찾는다. Locate 문의 선언부는 다음과 같다.

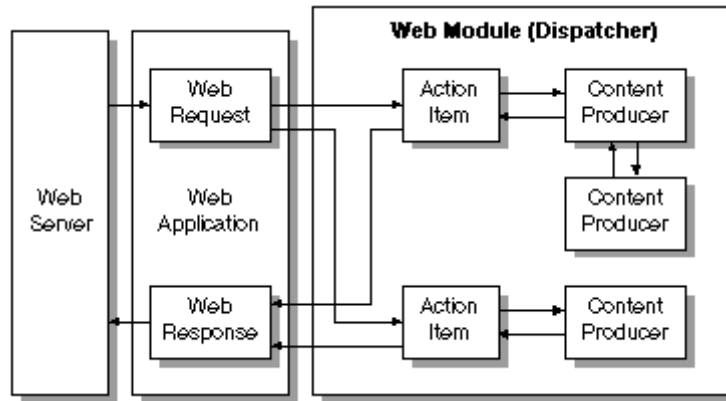
```
function Locate(const KeyFields: string; const KeyValues: Variant; Options:
  TLocateOptions): Boolean;
```

여기에서 KeyFields 파라미터에 원하는 키 값과 찾는 옵션이 들어간다.

그러므로, 다음의 코드는 'CustoNo'라는 필드의 값 중에서 Request.QueryFields 의 값 중에서 해당되는 레코드를 기본적인 옵션으로 찾는다는 의미이다.

```
Customer.Locate('CustNo', Request.QueryFields.Values['CustNo'], [])
```

여기에서 Request 라는 것은 TWebRequest 클래스 객체이다. TWebRequest 에 대해서 이해하기 위해서는 웹 서버 어플리케이션이 어떤 방식으로 동작하는지 알아야 한다. 이를 가장 잘 표현한 그림이 바로 다음 그림이다.



Action 아이템 이벤트 핸들러의 코드는 Request 정보에 대해 TWebRequest 객체의 프로퍼티를 통해 접근할 수 있다. 그리고, 콘텐츠 프로듀서 컴포넌트 (TPageProducer, TDataSetTableProducer 등)를 통해 Action 아이템들이 동적으로 커스텀 HTML 코드나 다른 MIME 콘텐츠를 만들어 낼 수 있다. 마지막으로, 콘텐츠 프로듀서는 다른 콘텐츠 프로듀서를 이용하거나 HTMLTagAttributes 의 자손을 이용하여 Response 메시지의 콘텐츠를 생성하는 것을 도와준다.

모든 Action 아이템들이 TWebResponse 객체에 기록을 마치고 나면, Dispatcher 가 결과를 웹 어플리케이션에 돌려준다. 어플리케이션은 이런 Response 를 서버를 통해 웹 브라우저로 보낸다.

이 TWebRequest 객체에는 여러가지 프로퍼티와 이벤트가 있으나, 먼저 중요한 TWebRequest.QueryFields 프로퍼티에 대해서 알아보도록 하자. 이 프로퍼티는 'Name=Value' 의 형태로 값을 전달한다.

예제에서 살펴 보면, 다음과 같은 코드가 있다.

```
Request.QueryFields.Values['CustNo']
```


이 코드의 의미는 'CustoNo'라는 Name 으로 들어온 파라미터의 값을 전달하라는 의미이다. 그러므로, 다음의 코드는 Request 로 들어온 파라미터 중에서 'CustNo'의 값을 받아서 Customer 테이블의 Locate 문장을 이용하여 'CustNo'에 해당되는 레코드를 찾게 된다.

```
Customer.Locate('CustNo', Request.QueryFields.Values['CustNo'], [])
```

여기에서 해당되는 레코드가 있을 경우에는 다음 코드가 수행되어 TQueryTableProducer 클래스 객체인 CustomerOrders 객체의 헤더를 적당하게 채우고, 해당 테이블의 내용을 보여주게 된다.

```
CustomerOrders.Header.Clear;  
CustomerOrders.Header.Add('The following table was produced using  
  a TDatasetTableProducer.<P>');  
CustomerOrders.Header.Add('Orders for: ' + CustomerCompany.AsString);  
Response.Content := CustomerOrders.Content;
```

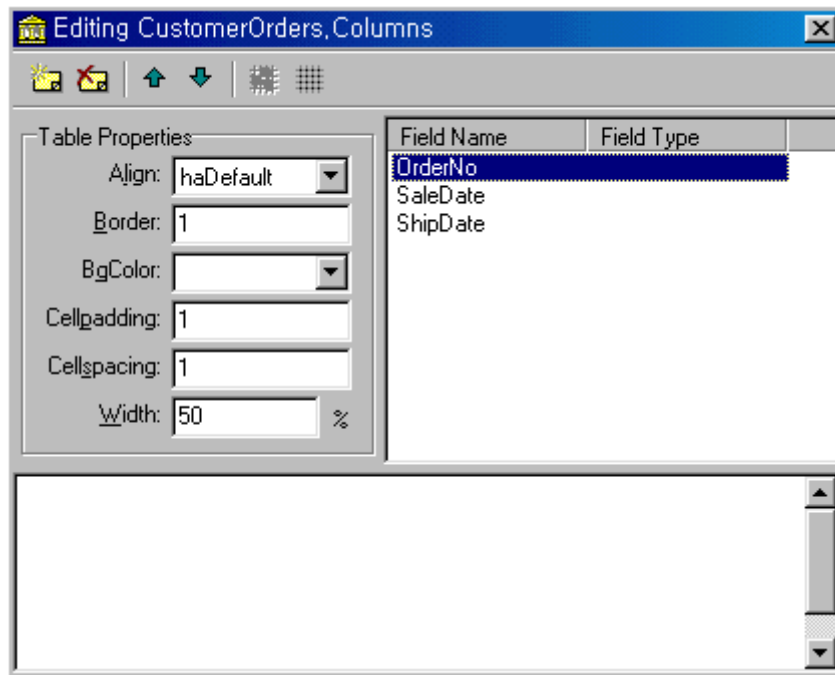
그리고, 레코드가 없는 경우에는 다음의 문장이 수행된다.

```
Response.Content := Format('<html><body><b>Customer: %s not found</b></body></html>',  
  [Request.QueryFields.Values['CustNo']]);
```

이 코드의 의미는 Response 의 Content 에 Format 의 함수를 사용하여 'CustNo' 파라미터에 해당되는 레코드를 찾을 수 없다는 메시지를 내보내는 것이다.

앞에서 테이블의 내용을 보여주기 위해서 CustomerOrders 라는 TQueryTableProducer 컴포넌트를 사용하였다. 이 컴포넌트는 무척 다양한 프로퍼티와 이벤트를 가지고 있다. 먼저 Dispatcher 프로퍼티에는 해당 Dispatcher 를 설정해야 하는데, 앞의 예제에서는 웹 모듈의 이름인 CustomerInfoModule 이 설정된다. 그리고, Footer 와 Header 프로퍼티에는 테이블 형태의 HTML 태그를 생성할 때에 처음과 마지막 부분을 장식할 데이터를 지정하게 된다. 또한, RowAttributes 와 TableAttributes 프로퍼티에서는 어떤 쿼리를 통해 데이터를 가져오고, 출력할 내용의 속성을 정의할 수 있다.

그리고, Columns 프로퍼티를 이용하면 출력될 그리드의 형태를 다음 그림과 같이 설정할 수 있다.



이 내용을 살펴보면 OrderNo, SaleDate, ShipDate 의 3 가지 필드영역을 가지는 그리드를 출력함을 알 수 있다. 그리고 Align 을 비롯한 출력에 필요한 다양한 속성을 원하는 형태로 설정할 수 있다.

문장에서 사용한 Header 와 Footer 의 내용은 기본적으로 비어있는데, 이 비어있는 내용을 채우기 위해서 다음과 같은 내용을 사용하였다.

```
CustomerOrders.Header.Clear;
```

```
CustomerOrders.Header.Add('Orders for: ' + CustomerCompany.AsString);
```

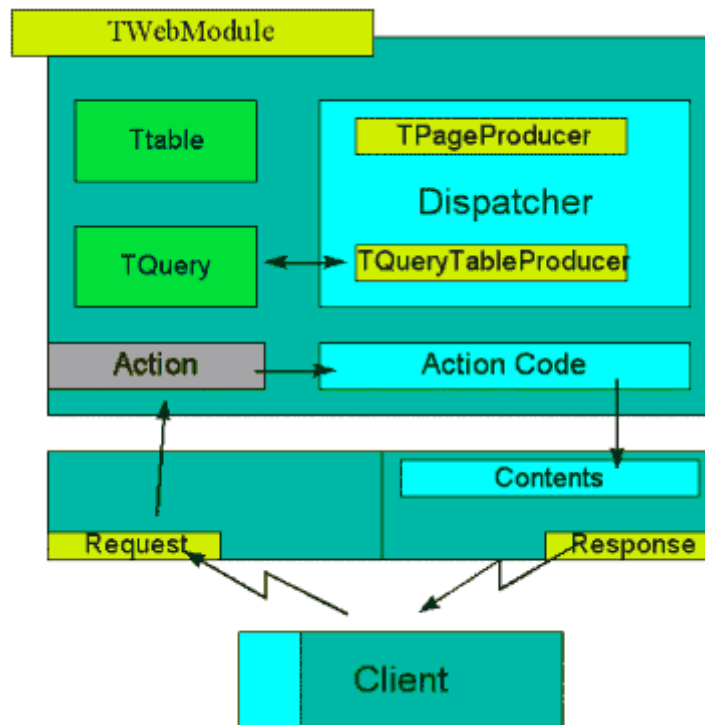
마지막으로, 다음과 같은 코드를 이용하여 Response 객체의 Content 를 CustomerOrders 객체의 HTML 출력 내용으로 채워서 출력을 하도록 한다.

```
Response.Content := CustomerOrders.Content;
```

Response 객체는 Request 객체에 대응되는 것으로 TWebResponse 클래스의 객체이다. 그러면, TWebResponse 클래스에 대해 간단히 알아보자.

TWebResponse 객체는 Response 를 전송하는 두가지 메소드를 제공한다. 이들은 각각 SendResponse 와 SendRedirect 이다. TWebResponse 객체의 모든 헤더 프로퍼티와 지정된 콘텐츠를 사용하는 Response 를 전송할 때는 SendResponse 를 호출한다. 이에 비해 SendRedirect 메소드는 웹 클라이언트에서 다른 URI 로 redirect 하기만 하면 될 때 효과적으로 쓰인다.

만약 Response 를 전송하는 이벤트 핸들러가 없다면, 웹 어플리케이션 객체는 Dispatcher 가 종료된 후 Response 를 전송하게 된다. 어쨌든 Response 를 다루는 Action 아이템이 없으면, 어플리케이션은 Response 를 전송하지 않고 웹 클라이언트와의 접속을 끊는다. 결국, Response 객체는 실제 웹 서버로 전송할 Content 에 해당되는 문장을 담게 된다. 이상의 구조를 단순하게 정리하여 그림으로 표현하면 다음과 같다.



CGI 어플리케이션 개발자는 TWebModule 위에서 원하는 데이터 세트로 사용할 테이블이나 쿼리를 얻는다. 그리고, TPageProducer 나 TQueryTableProducer 컴포넌트를 이용하여 원하는 작업을 Response 의 Contents 프로퍼티에 넣을 수 있고, 이러한 작업들이 바로 TWebModule 의 Action 에 의하여 구동되는 것이다.

이것이 텔파일을 이용한 CGI 어플리케이션을 작성하는 방법과 그 구조이다. CGI 프로그래밍은 이런 이벤트의 동작에만 주의하고 Contents 를 통하여 이루어진다는 것만 주의하면 그렇게 어려운 내용은 아니다.

업로드 CGI 의 제작

그러면, 자료를 업로드할 수 있는 CGI 를 하나 만들어 보자. 먼저 업로드를 호출하기 위한 HTML 을 작성하도록 한다.

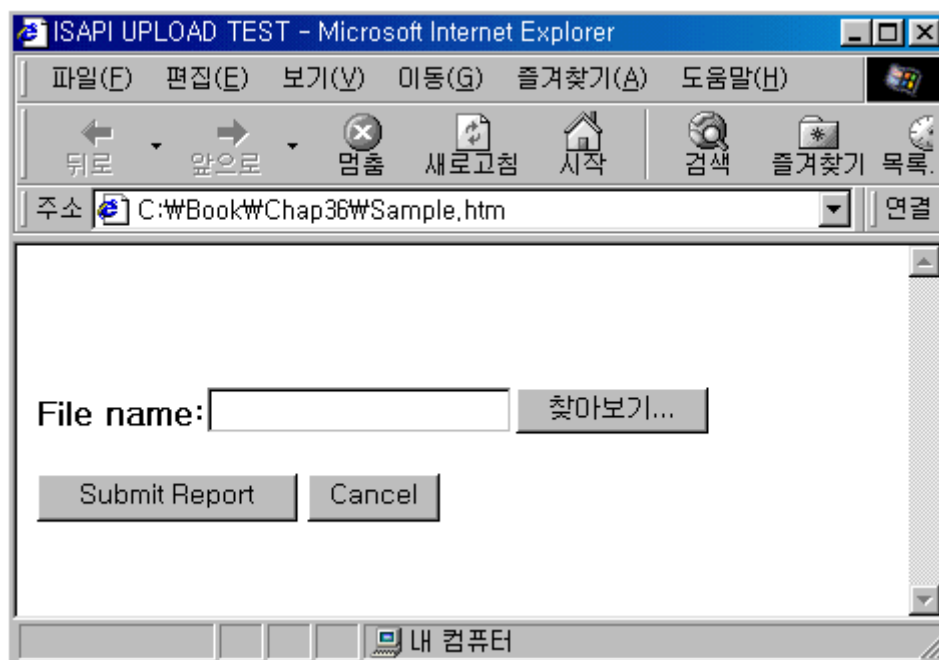
우선 파일 업로드 양식을 만들기 위해서는 fileupload 객체를 이용해서 작성을 해야 하며, CGI 로 보내는 데이터 형도 설정해야 된다. 다음은 웹 페이지의 샘플 소스이다.

```

<html>
<head>
<title>ISAPI UPLOAD TEST</title>
</head>
<body><p>&nbsp;</p>
<p> </p>
<FORM ENCTYPE="multipart/form-data" ACTION="Exam1.dll/upload" METHOD="POST">
<BR><B>File name:</B><INPUT TYPE="file" NAME="File">
<P><INPUT TYPE="submit" VALUE="Submit Report">
<INPUT TYPE="button" VALUE="Cancel" onClick="window.close()">
</FORM>
</body>
</html>

```

이 HTML 파일을 브라우저로 읽어 보면 다음과 같이 나타날 것이다.



이때에 CGI 를 호출하는 부분이 다음과 같다.

```

<FORM ENCTYPE="multipart/form-data" ACTION="Exam1.dll/upload" METHOD="POST">

```

이 내용 중에서 ‘ENCTYPE=”multipart/form-data”’ 부분은 해당 페이지에 있는 데이터 들을 cgi 로 송신할 종류를 나타내며 ”multipart/form-data”는 데이터를 송신하는 내용들 중에서 한가지이다.

METHOD=”POST”는 데이터를 어떠한 프로토콜로 보낼 것인지를 결정하며, “POST”는 표준 입력 방식으로 보낸다는 것을 의미한다.

참고로, 이렇게 파일을 업로드할 때에는 반드시 “POST”로 해야 한다.

그리고, 다음 내용은 fileupload 객체에 해당하는 부분으로, 실제 업로드할 파일을 선택하는 객체이다.

```
<INPUT TYPE="submit" VALUE="Submit Report">
```

cgi 에서 업로드를 실제 수행하는 작업은 의외로 간단하다. 웹 페이지에서 표준 입력으로 보내준 데이터 중에서 파일에 관련된 부분만 골라서 저장하면 된다. 업로드 웹 페이지에서 송신한 데이터는 인터넷 표준 문서인 RFC 문서(rfc1867.txt)에 정의된 데이터 포맷으로 cgi 에서 수신할 수 있다.

수신된 데이터 중에서 파일 정보에 해당하는 부분은 다음과 같은 구조로 되어 있다.

```
Content-disposition: attachment; filename="file1.txt"
```

```
Content-Type: text/plain
```

```
Contents
```

```
--BbC04y
```

수신된 파일의 파일 이름은 filename 이라는 부분에 정의되어 있고, 파일의 내용은 “contents”부분에 정의되어 있으므로, “contents” 부분을 따로 잘라내고 잘라낸 내용을 수신된 파일명으로 저장하면 파일 업로드를 구현할 수 있다.

다음 내용은 표준 입력으로 들어온 데이터를 출력한 파일이다.

```
-----7ce28a3a32c
```

```
Content-Disposition: form-data; name="uploadfilename"; filename="C:WtmpWWIN95WPcmcard.inf"
```

```
Content-Type: text/plain
```

```
; Ethernet PCMCIA Adapter INF file for Windows 95
```

```
;
```

[version]

signature="\$CHICAGO\$"

Class=Net

provider=%V_MS%

[Manufacturer]

%V_KINGMAX%=KINGMAX

[KINGMAX]

%KINGMAX.DeviceDesc%=EN10T2.ndi,PCMCIAWKINGMAX-EN10T2T-1BAB

[EN10T2.ndi]

AddReg=EN10T2.ndi.reg,PCM95.ndi.reg,PCM95.params.reg

[EN10T2.ndi.reg]

HKR,,Ndi,DeviceID,,"PCMCIAWKINGMAX-EN10T2T"

[PCM95.ndi.reg]

: key,subkey,valname,type,value

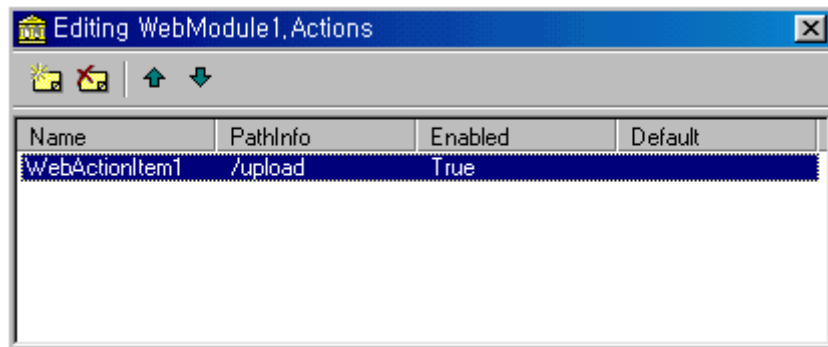
HKR,,DevLoader,,*ndis

HKR,,DeviceVxDs,,pcm95.sys

HKR,,EnumPropPages,,,"netdi.dll,EnumPropPages"

-----7ce28a3a32c-----

먼저 File|New 메뉴를 선택하고 여기에서 Web Server Application 아이템을 더블 클릭하고, 나타나는 CGI 형태로는 ISAPI/NSAPI 를 선택하도록 하자. 이렇게 하면, 웹 모듈이 생성되는데 오브젝트 인스펙터에서 Actions 프로퍼티를 선택하고, ‘...’ 버튼을 클릭하면 Actions 프로퍼티 에디터가 나타날 것이다. 여기에서 ‘Add New (Ins)’ 버튼을 클릭하고 이 아이템을 선택한 뒤, PathInfo 프로퍼티를 ‘/upload’로 설정하면 다음과 같이 나타날 것이다.



이제 업로드 CGI 를 구현하기 위해서는 다음과 같이 Action 아이템의 이벤트 핸들러를 구현해야 한다. 참고로, 이렇게 원시적인 코딩을 위해서는 isapi2.pas 와 isapiapp.pas 유닛을 uses 절에 추가해 주어야 한다.

```

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  i: Integer;
  L: LongInt;
  Buffer, Prueba: ^Byte;
  MemoryStream: TMemoryStream;
  Nowecb: PEXTENSION_CONTROL_BLOCK;
  TotalByte: Integer;          //수신될 데이터 크기
begin
  MemoryStream := TMemoryStream.Create;
  GetMem(Prueba, 11856);
  try
    Nowecb := TISAPIRequest(Request).ECB;
    TotalByte := Nowecb.cbTotalBytes;
    Buffer := Nowecb.lpbdata;
    L := Nowecb.cbAvailable;
    MemoryStream.Write(Buffer^, L);
    Dec(TotalByte, L);
    i := L;
    while((TotalByte > 0) and (i <> -1)) do
    begin
      i := Request.ReadClient(Prueba^, 11586);
      MemoryStream.Write(Prueba^, i);
    end;
  end;
end;

```

```

        Dec(TotalByte, i);
    end;
    MemoryStream.SaveToFile('c:\Windows\Temp\Wtmps.txt');
finally
    FreeMem(Prueba);
    MemoryStream.Free;
end;
end;

```

먼저 사용되는 변수 중에 PEXTENSION_CONTROL_BLOCK 데이터 형의 Nowecb 라는 변수가 있다. 이 데이터 형은 TISAPIRequest 클래스의 ECB 프로퍼티의 포인터 형으로, 서버 어플리케이션 DLL 이 웹 서버와 통신을 하기 위해 사용하는 확장 컨트롤 블록(extension control block)이다. 그러므로, ECB 프로퍼티는 서버 어플리케이션 DLL 이 클라이언트 요구 메시지를 대표할 때 사용된다. 이와 같이 ECB 프로퍼티를 직접 읽어올 수도 있지만, GetFieldByName 메소드를 이용하여 헤더의 필드 값을 읽어올 수 있다.

GetFieldByName 메소드는 HTTP 헤더 변수의 값을 읽어올 때 사용하는데, 이를 읽어와서 특정 작업을 하기 위해서는 인터넷 표준 문서인 각종 RFC 문서를 참고하면 된다. 텔넷이나 FTP 등의 대표적인 프로토콜 들도 RFC 로 규정되어 있으며, 이를 기초로 하여 원시적인 내용을 모두 구현할 수 있는 것이다. 그리고, 데이터를 읽어와서 저장하기 쉽도록 델파이에서 제공하는 TMemoryStream 클래스를 사용한다.

TotalByte, Buffer, L 이라는 변수는 각각 확장 컨트롤 블록(ECB)의 멤버를 저장하기 위해서 사용된다. TotalByte 변수는 클라이언트에 의해 지정되는 총 바이트 수를 저장하며, Buffer 변수는 해당 데이터에 대한 포인터 변수, 그리고 L 변수는 가능한 데이터의 바이트 수를 저장한다.

그러므로, 일단은 TISAPIRequest 클래스의 ECB 프로퍼티를 이용하여 Nowecb 변수에 Request 객체의 값을 읽어온 후 TotalByte, Buffer, L 변수에 해당 멤버의 값을 저장한다. 그리고, 메모리 스트림 객체에 Buffer 변수에 들어 있는 데이터를 L 변수에 크기 만큼 읽어와서 저장하는 것이 다음의 코드 들이다.

```

Nowecb := TISAPIRequest(Request).ECB;
TotalByte := Nowecb.cbTotalBytes;
Buffer := Nowecb.lpbdata;
L := Nowecb.cbAvailable;
MemoryStream.Write(Buffer^, L);

```

그리고, TotalByte 변수에서 읽은 수 만큼을 계속 감소시키고 이 값이 0 이 되거나 더 이상

읽어올 데이터가 없을 때까지 읽는다.

```
Dec(TotalByte, L):  
i := L;  
while((TotalByte > 0) and (i <> -1)) do  
begin  
  i := Request.ReadClient(Prueba^, 11586);  
  MemoryStream.Write(Prueba^, i);  
  Dec(TotalByte, i);  
end;
```

여기에서 Request 객체의 ReadClient 메소드는 HTTP request 의 콘텐츠 내용을 첫번째 파라미터로 지정된 변수에 두번째 파라미터의 크기 만큼 읽어오는 메소드이다. 즉, Prueba 변수에 11586 크기의 데이터를 읽게 된다. ReadClient 메소드는 서버 어플리케이션이 한번에 읽어오기에 너무 큰 request 데이터를 읽을 때, 이를 쪼개서 읽어올 수 있도록 해준다. 더 이상 읽어올 데이터가 없을 경우에는 -1 을 반환한다.

이렇게 루프를 돌게 되면, 결국에는 MemoryStream 변수에는 HTTP request 에 의해 넘어온 데이터가 그대로 저장된다. 이를 SaveToFile 메소드를 이용하여 저장하도록 할 수 있다.

정 리 (Summary)

이번 장에서는 텔파이를 이용하여 CGI 어플리케이션을 제작하는 방법에 대해서 알아보았다. 현재의 인터넷 환경은 앞서 이야기 했듯이 CGI 를 비롯한 여러가지 방법으로 웹 페이지의 내용과 형태를 풍부하게 만들어볼 수 있다. 하지만, CGI 는 아직도 가장 많이 쓰이는 방법 중의 하나이기 때문에 이를 텔파이를 이용하여 쉽게 작성할 수 있다는 점은 커다란 매력이 아닐 수 없다.

다음 장에서는 텔파이에서 제공하는 소켓 컴포넌트를 이용하여 프로그래밍하는 방법에 대해서 알아볼 것이다.