Een eenvoudige webserver maken in Lazarus

Michaël Van Canneyt

2 december 2011

Samenvatting

Free Pascal - en dus ook Lazarus - kan meerdere TCP/IP componenten compileren: Synapse, Indy en lnet. Free Pascal wordt echter ook verdeeld met enkele eenvoudige netwerk componenten. Een van deze componenten is een eenvoudige webserver. Dit artikel beschrijft hoe deze component gebruikt kan worden.

1 Inleiding

Soms is het nodig een kleine webserver aan een toepassing toe te voegen. Bijvoorbeeld, als een web-interface voor configuratie of onderhoud van de toepassing nodig is. Dat komt bijvoorbeeld vaak voor bij een Windows service-applicatie (of daemon), omdat die geen directe interactie met de desktop kan hebben.

Er zijn verschillende TCP/IP netwerk componenten beschikbaar: Indy en Synapse zijn het meest bekend, zeker voor mensen die ervaring hebben met Delphi. LNet is speciaal voor Free Pascal ontwikkeld, en gebruikt een event-driven (gebeurtenis gestuurde), niet blokkerende aanpak. Sinds lange tijd beschikt Free Pascal over een aantal eenvoudige TCP/IP klassen, die recent gebruikt werden om een paar componenten te maken die de client en server kant van het HTTP protocol implementeren.

Gecombineerd met de bestaande FCL-Web componenten, kunnen deze nieuwe componenten gebruikt worden om een complete HTTP server te maken. Dit kan dan met een standaard installatie van Free Pascal en Lazarus gebeuren, zonder dat er extra componenten geinstalleerd moeten worden. Deze HTTP server kan dan gebruikt worden voor een web applicatie of om een webservice zoals WST te hosten. WST staat voor Web Services Toolkit, en is uitvoerig beschreven in het Lazarus boek.

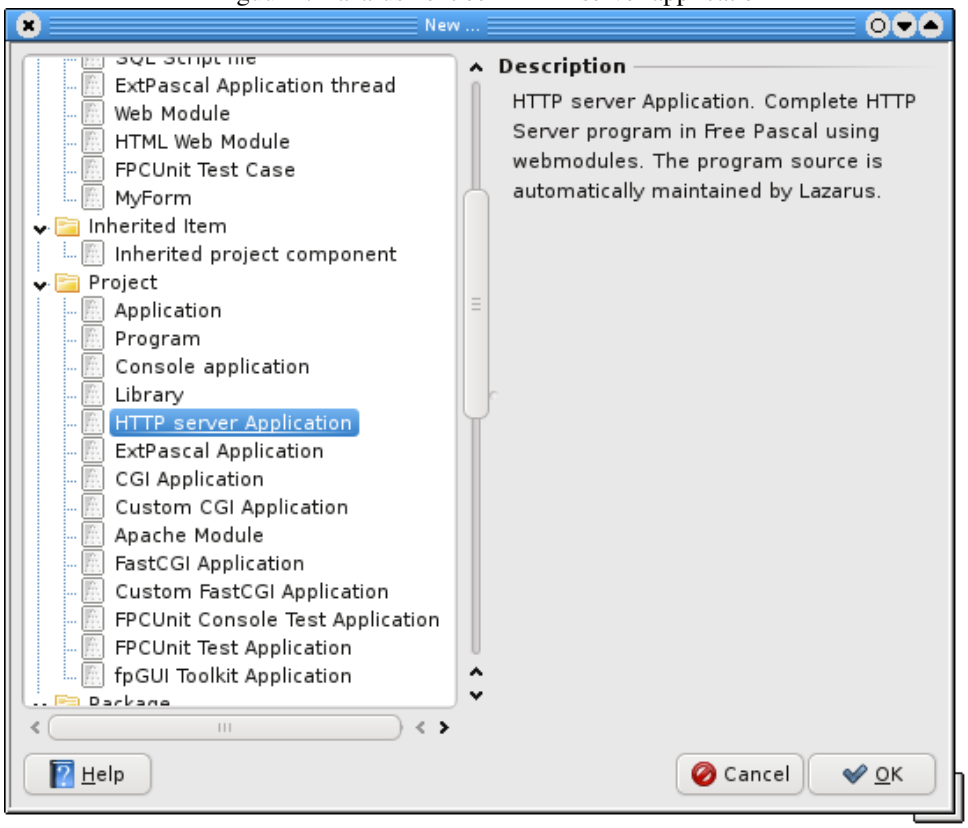
De componenten zijn eenvoudig van architectuur, en daarom niet geschikt om een webserver te maken die schaalbaar is en zware netwerk-belasting moet kunnen afhandelen. Voor eenvoudige webservices of als ingebedde webserver is de component echter zeer bruikbaar.

Het gaat om 2 componenten:

TFPHTTPServer Dit is de component die het eigenlijke werk doet. Er zijn wat eigenschappen (properties) beschikbaar die toestaan de server te configureren. De belangrijkste, 'Port', is de TCP/IP poort (port) waarop de server moet luisteren naar connecties. De tweede belangrijke property is de `Active` property, die de server start. Er is ook een event beschikbaar, die de eigenlijke HTTP requests afhandelt. De event heet - hoe kan het ook anders - `OnRequest`. Wanneer men een HTTP server in een bestaande applicatie wil inbedden, is dit de component die gebruikt moet worden.

THTTPApplication Dit is een `TCustomApplication` descendant: deze component gebruikt een `THTTPServer` component om naar requests te luisteren, en speelt de

Figuur 1: Lazarus kent een HTTP server applicatie



requests dan door aan het FCL-Web request dispatcher mechanisme. Dit is de component die gebruikt mag worden als een web-server of web service hosting applicatie gebouwd wordt. De server applicatie kan geconfigureerd worden d.m.v. dezelfde eigenschappen (properties) als bij de `THTTPServer` component.

Als in Lazarus het package LazWebExtra geïnstalleerd is, dan is deze laatste component beschikbaar d.m.v. de ¨File - New¨ dialoog. De optie 'HTTP Server Application' (zichtbaar in figure 1 on page 2).

2 Een eenvoudige webserver

De eenvoudigste webserver kan met enkele lijnen code geschreven worden:

```
program myserver;

uses
  SysUtils, fphttpapp, fpwebfile;

Const
  MyPort = 8080;
  MyDocumentRoot = '/home/michael/public_html';
  MyMiMeFile = '/etc/mime.types';

begin
```

```
  RegisterFileLocation('files',MyDocumentRoot);
  MimeTypesFile:=MyMimeFile;
  Application.Initialize;
  Application.Port:=MyPort;
  Application.Title:='My HTTP Server';
  Application.Run;
end.
```

Deze broncode verschilt niet zo heel veel van wat de Lazarus IDE automatisch aanmaakt als een 'HTTP Server application' project gestart wordt.

De unit fpwebfile in de uses clause verdient enige aandacht: Deze unit bevat de implementatie van een complete en gebruiksklare FCL-Web module (`TFPCustomFileModule`) die bestanden doorstuurt via HTTP. Het is niet nodig een instance van deze module te maken, FCL-Web doet dit helemaal automatisch als een HTTP request voor een bestand binnenkomt. Het enige dat wel dient te gebeuren is FCL-Web duidelijk te maken welke directory op de harde schijf overeenstemt met een HTTP locatie. Dit is exact wat gebeurt in de eerste lijn van het programma:

```
  RegisterFileLocation('files',MyDocumentRoot);
```

Deze lijn maakt aan FCL-Web duidelijk dat een URL als

```
http://localhost/files/myfile.html
```

vertaald moet worden naar een bestand op schijf met de naam

```
/home/michael/public_html/myfile.html
```

Deze syntax is - niet toevallig - erg gelijkend op het effect dat het 'Alias' directive heeft in een Apache webserver configuratie bestand. Het spreekt voor zich dat de directory moet bestaan op de machine waar het programma zal lopen. Meerdere locaties kunnen door de `RegisterFileLocation` call geregistreerd worden.

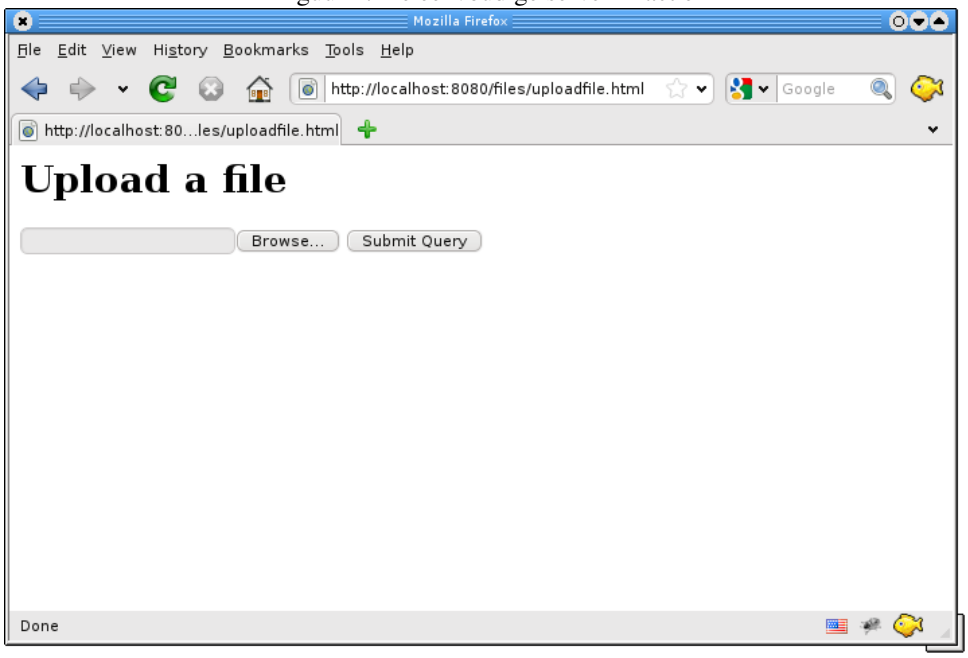
De tweede lijn code vertelt de module die bestanden doorstuurt waar het `mime.types` bestand staat. Dit bestand (standaard geinstalleerd op alle unix system) wordt gebruikt om bestands extensies om te zetten naar mime-types. Het mime-type moet doorgegeven worden in de `Content-Type` HTTP header. Deze header vertelt de web-browser welk soort bestand het toegespeeld krijgt van de server, zodat de browser weet wat te doen met het bestand. Op windows is zo'n bestand niet standaard beschikbaar, en moet het dus op een of andere manier naast het programmabestand gezet worden. Een typische Apache installatie bevat een 'mime.types' bestand, en dit bestand kan gebruikt worden voor de `TFPCustomFileModule` component.

Als het bovenstaande programma gestart wordt, kan de browser gebruikt worden om bestanden te tonen uit de directories die geregistreerd werden. Een voorbeeldje is zichtbaar in figure 2 on page 4.

3 Een webserver ingebed in een GUI applicatie

Het vorige voorbeeld is zeer eenvoudig, maar ook beperkt. Het programma kan niets doen dan het beantwoorden van web-requests. Gebruik makend van de `THTTPServer` component, is het echter mogelijk de webserver functionaliteit in te bedden in een bestaande applicatie. Dit hoeft niet moeilijker te zijn als het vorige voorbeeld. Om dit aan te tonen, kan een kleine grafische applicatie gebouwd worden, met een TMemo component en 2

Figuur 2: De eenvoudige server in actie



knoppen: 1 knop om de server te starten - met de toepasselijke naam `Bstart` - en een knop om de server te stoppen (`BStop`). De memo component (met de naam `MLog`) dient om de web requests te loggen.

De toepassing zal bestanden doorsturen, en een `TFPCustomFileModule` instance kan gebruikt worden om dit te doen. Deze instance kan aangemaakt en geconfigureerd worden in de `OnCreate` event van form:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  RegisterFileLocation('files','/home/michael/public_html');
  MimeTypesFile:='/etc/mime.types';
  FHandler:=TFPCustomFileModule.CreateNew(Self);
  FHandler.BaseURL:='files/';
end;
```

De eerste lijnen zijn gelijkaardig aan die in het eerste voorbeeld. De laatste 2 lijnen cree¨eren een instance van de module en vertellen die instance van welke locatie bestanden moeten doorgestuurd worden. Merk op dat de instance aangemaakt wordt met de `CreateNew` constructor: dit zorgt ervoor dat de Run-Time Library niet op zoek gaat naar een form bestand voor de module. Aangezien er zo geen bestand is, zou de constructor `Create` een exception geven.

Als alles eenvoudig was, zou de start knop een instance van `TFPHTTPServer` maken en de `Active` eigenschap op `True` zetten:

```
procedure TMainForm.BStartClick(Sender: TObject);
begin
  MLog.Lines.Add('Starting server');
  FServer:=TFPHTTPServer.Create(Self);
  FServer.Port:=8080;
  FServer.OnRequest:=@DoHandleRequest;
```

```
  FServer.Active:=True;
end;
```

Helaas zou dit de toepassing laten 'bevriezen' zodra op de knop 'start' geklikt werd. De lijn

```
  FServer.Active:=True;
```

keert niet terug totdat de server stopt, waardoor de grafische laag (het zichtbare scherm) niet meer zou reageren op clicks, toetsaanslagen of verplaatsen van het scherm. Dit is natuurlijk niet erg handig.

Daarom moet de server in een aparte thread gestart worden. De klik op de start knop moet deze thread aanmaken. De volgende thread implementatie start de HTTP server:

```
THTTPServerThread = Class(TThread)
Private
  FServer : TFPHTTPServer;
Public
  Constructor Create(APort : Word;
         Const OnRequest : THTTPServerRequestHandler);
  Procedure Execute; override;
  Procedure DoTerminate; override;
  Property Server : TFPHTTPServer Read FServer;
end;
```

De constructor creëert de `TFPHTTPServer` instance en stelt de `Port` eigenschap en `onRequest` handler in:

```
constructor THTTPServerThread.Create(APort: Word;
   const OnRequest: THTTPServerRequestHandler);
begin
  FServer:=TFPHTTPServer.Create(Nil);
  FServer.Port:=APort;
  FServer.OnRequest:=OnRequest;
  Inherited Create(False);
end;
```

De Execute methode zet eenvoudigweg `Active` op `True` en wacht dan tot de call terugkeert. Daarna wordt de server instance vrijgegeven:

```
procedure THTTPServerThread.Execute;
begin
  try
    FServer.Active:=True;
  Finally
    FreeAndNil(FServer);
  end;
end;
```

Als de thread gestopt wordt - door een andere thread, bv. de GUI thread - moet de server gestopt worden:

```
procedure THTTPServerThread.DoTerminate;
begin
```

```
  inherited DoTerminate;
  FServer.Active:=False;
end;
```

Met deze thread klasse, wordt de `OnClick` handler van de start knop zeer eenvoudig:

```
procedure TMainForm.BStartClick(Sender: TObject);
begin
  MLog.Lines.Add('Starting server');
  FServer:=THTTPServerThread.Create(8080,@DoHandleRequest);
end;
```

Merk op dat een form methode `DoHandleRequest` doorgegeven wordt om de requests af te handelen. Deze methode wordt uitgevoerd in de context van de server thread, en enige synchronisatie van threads dus is vereist.

Op gelijkaardige manier kan de `OnClick` handler van de 'stop' knop de thread stoppen:

```
procedure TMainForm.BStopClick(Sender: TObject);
begin
  MLog.Lines.Add('Stopping server');
  FServer.Terminate;
end;
```

Het enige wat nog te doen is, is het opvullen van de `DoHandleRequest` methode:

```
procedure TMainForm.DoHandleRequest(Sender: TObject;
  var ARequest: TFPHTTPConnectionRequest;
  var AResponse: TFPHTTPConnectionResponse);
begin
  FURL:=Arequest.URL;
  FServer.Synchronize(@ShowURL);
  FHandler.HandleRequest(ARequest,AResponse);
end;
```

De methode bewaart de URL en roept dan de `ShowURL` methode op. Omdat de `DoHandleRequest` event routine opgeroepen wordt vanuit de server thread, moet het eigenlijke tonen van de URL in de main GUI thread gebeuren: Dit gebeurt door de methode met `Synchronize` in de GUI thread te laten uitvoeren. De `ShowURL` methode is ook extreem eenvoudig:

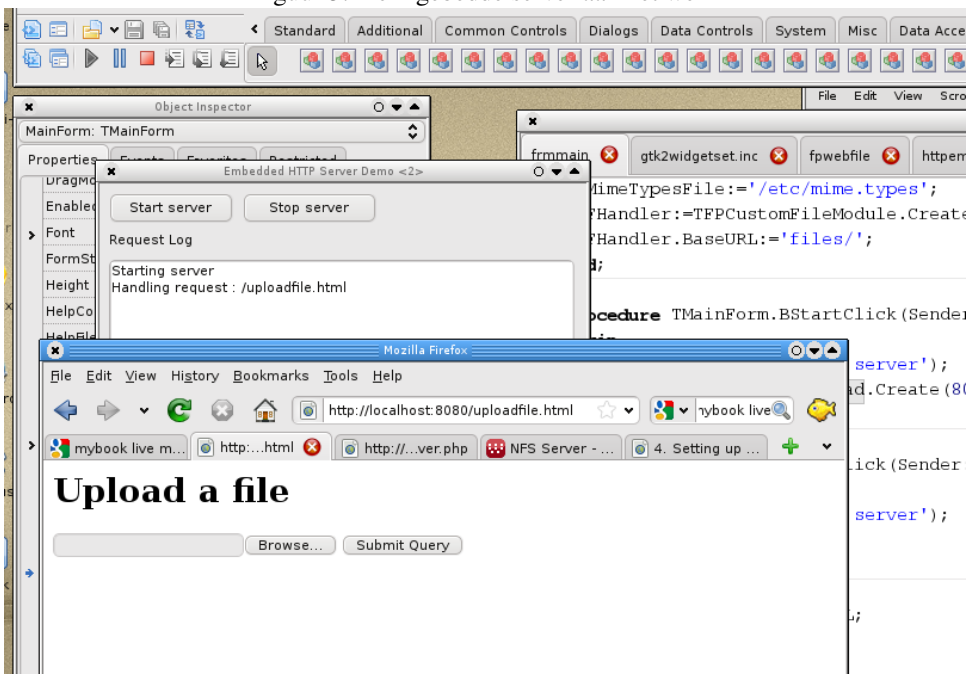
```
procedure TMainForm.ShowURL;

begin
  MLog.Lines.Add('Handling request : '+FURL);
end;
```

Hiermee is de demo applicatie klaar. Na het starten van de applicatie kan de 'Start' gebruikt worden om de HTTP server te starten. Wat er dan gebeurt als een URL voor de webserver wordt ingegeven in een browser, wordt getoond in figure 3 on page 7: de pagina wordt getoond in de browser, en een log bericht wordt getoond in het scherm van de applicatie.

Merk op dat de URL die ingegeven wer, niet de locatie 'files' bevat, zoals in het eerste voorbeeld. De reden hiervoor is dat de applicatie niet moet beslissen welke FCL-Web module nodig is om het HTTP request af te handelen: alle requests worden doorgegeven aan de file server module. Deze module werd in de `OnCreate` event handler van het scherm gecreÃńerd en de lijn

Figuur 3: De ingebedde server aan het werk




```
FHandler.BaseURL:='files/';
```

maakt de module duidelijk dat het in de geregistreerde locatie 'files' naar de gevraagde bestanden moet zoeken.

4 conclusie

In dit artikel werd het gebruik van de standaard FPC componenten voor get HTTP protocol gedemonstreerd: met enkele lijnen code is het mogelijk een programma te schrijven dat zich als webserver gedraagt. Niet alle beschikbare functionaliteit werd besproken: de client-side component bijvoorbeeld werden niet behandeld, evenals ondersteuning van Web Services.