Free Spider Web development for Free Pascal/Lazarus

user's manual

FreeSpider Version 1.2.2
Modified: 14.Aug.2010
Author: Motaz Abdel Azeem
Home Page : http://code.sd
License: LGPL

Introduction:

Free Spider is a web development tool for Lazarus. You can create any web application in Linux, Windows, Mac or any other platform that already supported by FreePascal/Lazarus. Lazarus produces independent executable file. You can copy this produced CGI application and put it in your **cgi-bin** directory and then access it from your network as a web application.
Current supported web technology and protocol is **CGI** (Common Gateway Interface). In the future we may support more web technologies.

How to create Free Spider web applications

to create Free Spider web application follow these steps:

1. In Lazarus IDE main menu select File/New.. select FreeSpider CGI Web Application
2. Put **TSpiderCGI** component in the data module, select it from FreeSpider page
3. Double click on Data Module or select **OnCreate** event and write this code on it:

```
  SpiderCGI1.Execute;
```

4. Double click on SpiderCGI1 or select **OnRequest** event and write this code on it:

```
  Response.Add('Hello world');
```

5. Change application **output directory** your web application cgi-bin directory, such as /usr/lib/cgi-bin in Linux. You can change this settings by clicking Project menu/Project Options/Application Tab/Output settings group/Target file name, then you write your application name after cgi-bin path.
6. Suppose that your project name is **first**, then you can call it from your browser like this:

```
http://localhost/cgi-bin/first
```

Request object:

Request Object of TSpiderCGI's **OnRequest** event contains request information, such as query fields that has been called with web application's URL like this:

```
http://localhost/cgi-bin/first?name=Mohammed&address=Sudan
```

Then you can access this query information like this:

```
  UserName:= Request.Query('name');
  Address:= Request.Query('address');
```

Also you can access it from Query's String List:

```
  UserName:= Request.QueryFields.values['name'];
```

Query reads the data that has been sent in URL or from HTML form using GET method.

If you are using POST method of an HTML form, you should read it using Form method of Request:

```
  Login:= Request.Form('login');
  Password:= Request.Form('password');
```

Also you can use Content fields string list to access all posted data:

```
  UserName:= Request.ContentFields.values['login'];
```

Request object also contains some user's browser information like **UserAgent**, and **RemoteAddress** of the client.

Response object:

Response object of TspiderCGI's **OnRequest** event represents the response HTML that should be shown in user's browser after requesting a URL.
You can add HTML text using **Add** method. You can use **Add** many times to build the whole HTML page. **Add** method accumulates HTML text in a string list. At the end of **OnRequest** event all these HTML that has been added will be displayed in the browser at once.

Example:

```
  Response.Add('Hello world<br>');
  Response.Add('This has been written in Object Pascal');
```

TSpiderAction component

TspiderCGI is needed for any CGI application in Free Spider web application. It can handle only one request/response (**action**) which has been requested from browser using this method like:

```
http://localhost/cgi-bin/first
http://localhost/cgi-bin/first?name=Motaz
```

TSpiderActions components can handle more additional request/response actions, for example suppose that you have a web application which has many HTML forms, each form represent different request, for example an e-mail system needs these request/response actions: register user, login, logout, view Inbox, send e-mail, etc. You can call these requests like this:

```
http://localhost/cgi-bin/mail/login
http://localhost/cgi-bin/mail/logout
http://localhost/cgi-bin/mail/reg
http://localhost/cgi-bin/mail/inbox
```

/login, /logout, /reg, and /inbox parts are called **Paths**. Each TSpiderAction component can handle one Path. In this case you will need 4 TSpiderAction components.
TSpiderAction's **Request** and **Response** parameters are the same like TspiderCGI Request/Response parameters.
OnRequest event of TSpiderAction is the same like TSpiderCGI's. The main difference is that TSpiderCGI's OnRequest will be called when no path information is send like cgi-bin/mail. But TSpiderAction's OnRequest will be called when the user click a URL that contains this path, for example: cgi-bin/mail/inbox

TSpiderTable component

TSpiderTable component generates HTML table, either manually or from dataset.

You can use it manually like this:

```
  SpiderTable1.ColumnCount:= 3;
  SpiderTable1.SetHeader(['ID', 'Name', 'Telephone Number']);
  SpiderTable1.AddRow('', ['1', 'Mohammed', '01223311']);
  SpiderTable1.AddRow('#FFEEDD', ['2', 'Ahmed', '01754341']);
  SpiderTable1.AddRow('#FFDDEE', ['3', 'Omer', '045667890']);
  Response.Add(SpiderTable1.Contents);
```

This will generate a table in browser like this:

ID	Name	Telephone Number
1	Mohammed	01223311
2	Ahmed	01754341
3	Omer	045667890

Other mode for TSpiderTable is using **DataSet**. When you set a DataSet property either at design time or at run time you will get an HTML table from DataSet:

```
SpiderTable1.DataSet:= ATable;
ATable.Open;
Response.Add(SpiderTable1.Contents);
```

TSpiderTable contains two events:
OnDrawHeader: will be triggered when drawing Table's header
OnDrawDataCell: will be triggered when drawing other data cells

Both events contains **CellData** and **BgColor** for being drawn cell. Web Developer can change them according to specific state.

TSpiderForm

TSpiderForm generates HTML form to be used in entering data and to be submitted to other or the same action.
TSpiderForm contains many important properties which are:

1. Method: by default it should be POST which can contains more data and will not display it in

URL. GET method will send form field values in the URL like previous examples.

2. Action: represents which server/alias/web application/path should receive and handle this form submittion, for example it can be like this: /cgi-bin/mail/login
3. ExtraParam: contains additional form properties such as Java Script code, or encode type like file upload multipart type which will take about later in uploading files.
4. PutInTable: by default it is True. If it is true, it puts table fields and labels (AddText) in a table of two columns: first column contains the text that has been added by AddText, and second column is the field that has been added by AddInput. If you call AddInput without AddText; then this input will be displayed in the first column, the same like the button on the example below.

Example of generating form:

```
SpiderForm1.AddText('Enter Subscriber ID: ');
SpiderForm1.AddInput(itText, 'id', '');

SpiderForm1.AddText('Enter Subscriber Name: ');
SpiderForm1.AddInput(itText, 'subname', '');

SpiderForm1.AddText('Address: ');
SpiderForm1.AddInput(itText, 'address', '');

SpiderForm1.AddText('Telephone Number: ');
SpiderForm1.AddInput(itText, 'telephone', '');

SpiderForm1.AddInput(itSubmit, 'add', 'Add');


Response.Add(SpiderForm1.Contents);
```

This will generate form in user's browser like this:

Enter Subscriber ID:

Enter Subscriber Name:

Address:

Telephone Number:

Add

TSpiderPage

TSpiderPage component enables web developer/designer to design HTML pages using any HTML editor such as Office Wirter. This the designer should embed tags in which he want dynamic contents to be shown. Tags should be like this `@tag;` inside HTML.
You can replace these tags later in TSpiderPage's **OnTag** event.
TSpiderPage enables the developer to separate web application user interface design from compiled code. If any change in design later is needed, the developer may not need to recompile the web application, instead he/she needs to change only external HTML pages.
HTML pages can be put in the same directory with CGI application like cgi-bin directory.

CGI life cycle

CGI applications has a very short life cycle. The application will be uploaded into server memory and executed starting from user's request (click/refresh, enter URL, etc) until being served (until user gets result on browser). This time normally can be less than 1 seconds according to simplicity of action's event handler (OnRequest). For that reason Free Pascal/Lazarus CGI web applications is easy to develop, no memory leaks will occur, the developer shouldn't be worry about freeing resources, because these resources will be reserved for a short time and freed automatically when this CGI application goes out of memory. These resources can be memory allocation, file, socket, etc. This behavior of CGI application grantees more reliability and more uptime for the web application.

Concurrent requests

One of the most important issue of any web application is concurrency control. CGI application upload a different instance of the executable on each request. For example if there are a concurrent 3 requests, a 3 instance of CGI exe will be uploaded into memory to serve each request separately. This make CGI application not suitable for huge Internet applications. The maximum number of concurrent requests that the web server could handle depends on these variables:

1. Server memory
2. CGI application memory usage
3. Average response time

whenever average response time is very small, the most users our web server can handle.
Concurrent requests does not represents total web application users. For example if we developed a web application for a company and it can handle 50 concurrent requests, that doesn't mean only 50 employees can access this web application, because users may request a very fast-fraction of second request and stay minutes reading the response. In this case we could say roughly our web server can serve 1000 employees and we expected those 1000 users to have maximumly 50 requests concurrently. If we felt that the server response is starting to degrade then we can deploy another instance of web server containing the same web application and split users between these web servers.

User Session

User session in Free Spider CGI and in CGI as general is handle by using cookies. Cookies are a hidden data that web application can store in browser's page to let the browser to send it in the next request. The behavior of stateless life cycle makes it difficult to web application to link between first/second/thid … requests. For that reason we need to put some data in each web browser to keep session tracking. This methodology is handled by Response's SetCookie method, which can be called like this:

```
  Response.SetCookie('sessionid', '2', '/');
```

Cookies path parameter represents the scope of this cookies. If it is '/' that means all web applications in this server can access this value. Some times you need to have a large web application system which falls into many projects and then executables, these executables can serve the same user using only one login by using cookies and global path '/'.
Previous cookies will be erased when the user closes the browser.
If you want to set more/less expiration time for cookies (In GMT) you shoul provide Expiration parameter like:

```
  Response.SetCookie('sessionid', '2', '/', Now + 1);
```

In this case, cookes will last for one day. You can set it to hours, or minutes, but at first you should know your GMT difference. For example if your country time is GMT + 3, you should deduct 3 hours from Now time, after that you can set expiration time:

```
    Response.SetCookie('sessionid', '2', '/', Now -
      EncodeTime(3, 0, 0, 0) + EncodeTime(0, 5, 0, 0));
```

This will make the sessionid cookie field last for 5 minutes

Cookies can read using Request's GetCookie method like this:

```
  Response.Add(Request.GetCookie('sessionid'));
```

File upload/Download

To upload file into web application, you need to change form encode type to multipart/form-data using TSpiderForm's **ExtraParam** property. You can put this value at design time or at run time:

```
  SpiderForm1.ExtraParam:= 'enctype="multipart/form-data"';
```

Example:

```
    Response.Add('<h2>File Upload sample</h2>');
```

```
    SpiderForm1.AddInput(itFile, 'upload');
    SpiderForm1.AddInput(itSubmit, 'upload', 'Upload');
    Response.Add(sfUpload.Contents);
```

Then you can receive this file using Request **ContentFiles** property, and download it to user:

```
    Response.ContentType:= Request.ContentFiles[0].ContentType;
    Response.CustomHeader.Add('Content-Disposition: filename="' +
      Request.ContentFiles[0].FileName + '"');
    Response.Content.Add(Request.ContentFiles[0].FileContent);
```

Smart Module Loading Design

As a new feature of Free Spider, you can create a web application that contains many Data Modules. Each data Module can contain Spider Actions, Pages, Tables, and Forms components, and their required datasets and any other objects.
The idea of Smart module loading, is to have a large web application that contains a lot of data modules, but only the required modules will be loaded per request.
Main module that contains SpiderCGI will be created regardless of requested path, plus additional data module that contains this SpiderAction/path will be created too. That means if you have a web application that has 100 different paths, you can put them in 10 modules for example, and only two modules will be loaded into memory to serve this user's request.
This method will reduce memory consumption because only required objects will be created, and this will make the response more faster because creation objects in memory takes time.
To make a smart module loading design follow these steps:

1. Create Free Spider web application the same as above examples
2. Add New Data Module
3. Call RegisterClass procedure to register this new data module class. Suppose that this new data module class is called TdmMod2, then you should write this code in Data Module 2 Initialization section:

```
initialization
  {$I mod2.lrs}
  RegisterClass(TdmMod2);
```

4. Add this unit name in main Data Module uses clause
5. Put Free Spider TspiderAction components and define your new paths
6. in main Data Module OnCreate event write this code:

```
  SpiderCGI1.AddDataModule('TdmMod2', ['/path2', '/path3']);

  SpiderCGI1.Execute;
```

Paths parameter should contain all the paths that exist in this new Data Module, if you forget to add any path, then you can not call it from user's browser, because Free Spider is using this method to link Actions with their proper Data Modules.

SpiderSample web application is using this design for additional Data Modules.

Note:
You can put global objects that every module requires in main Data Module, like database connection objects.

Performance

You can refer to FreeSpider web page to see the detailed performance test that has been done to FreeSpider applications.
As a summary of these tests, on a dual core 2.16 Ghz and 2 GB RAM Linux server with Apache you can get **2500** request per minute for a FreeSpider web application that connected to database without concurrent connections (no busy hour). With concurrent connections (busy hour) this will reduce the performance which caused by database server. Result should be **1000** requests per minute. This would be varying depending on busy hour/normal hour conditions, and depends on the complexity of database query/update operations.
Suppose that you have deployed a FreeSpider web application in a company, and this application is doing a database operations, and every employee is requesting a page every minute as an average, then your web application could handle **1000** employees working at the same time . At the busy hour, they may feel some delay, but in normal hours they will get a faster response and more employees could can use the same web application. In normal non-busy hours (or if you have no busy hour) your web application could handle **2500** employees/clients.