kleiner kommunikation
max@kleiner.com

DelphiWebStart (DWS)

System Manual v1.8

1.1 Abstract

Loading different apps or several files without a browser over a TCP/IP net and on Win as Linux as well needs a decision once. With a loader on the client side, no further installation is in charge, we call that DelphiWebStart.

We had the requirement starting different applications from a Linux or windows server, wherever you are. We call it Delphi Web Start (DWS). In short the DWS-client gets a list and after clicking on it, the app or the file is loading from a socket server over the net to the client with just a TCP stream.

So a user can download data (exes, maps, files etc.) from a easy list and start it. DWS 1.8 supports OpenSSL.

1.2 Indy Sockets on CLX

DWS is building on the Relationship between Indy Sockets, CLX and the Qt library. CLX is a cross platform environment for migrating or develop Delphi apps.

In V1.8 we use Delphi 7.1, CLX 2.1 with Indy 9.0

As you may know 3 porting techniques exists:

- Platform specific port: targets an OS and underlying APIs (Win32)
- Cross platform port: targets a cross platform API (like CLX)
- Emulation: leaves the code alone and port parts of the API it uses

Kylix uses CLX in place of the VCL for Cross platform. CLX provides access to Qt widgets (from window + gadget) in the Qt shared libraries. This Qt-library is available on Linux and Win as well! Functions are then wrapped in a shared object in Linux and DLL in Win.

```
on win: qtinf.dll
on linux: qtinf.so
```

Because Delphi can't have a direct access to the C++ Lib of Qt (Multiple Inheritance, RTTL and so on) all functions are wrapped in the interface with simple C functions like the Win API (uses qt.pas).

In most cases, we don't need to change occurrences of `TWinControl` to `TWidgetControl`. The following type declaration appears in the `QControls.pas` to simplify sharing of source code:

```
TWinControl = TWidgetControl;
```

Because CLX is a wrapper around Qt, and most of the controls on the Kylix Component palette are simply Pascal wrappers around Qt classes, so most of the units start with a Q in their names.

DWS is building on Indy Components (see Abb. 1). First was winshoes, a set of “blocking sockets” components for Delphi. I have worked with winshoes and I have created many programs. When Delphi 6 was released, the winshoes project was integrated in Delphi, but with a new name (Indy) and with a new sponsor (Nevrona

design). In order to understand what you can do with Indy we think DWS with the TCP/IP socket is the best example.

Characteristic for a TCP/IP protocol and DWS are the following elements:

- IP address;
- Host Name;
- Port Number.

In DWS client (see Abb. 6) you can set the IP address or the host name.

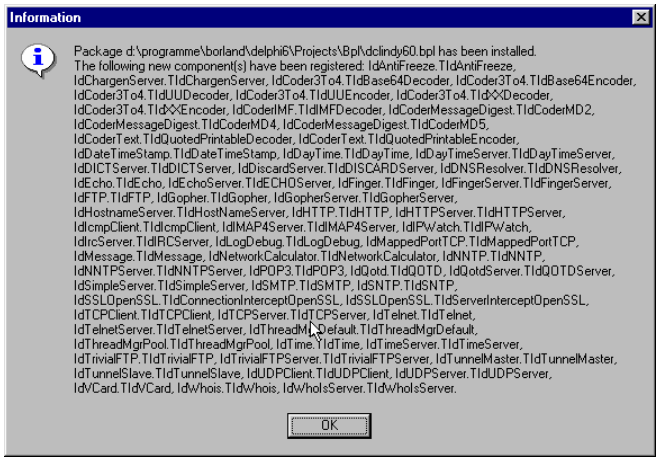

Abb. 1: The Indy Components

Kylix with CLX runs only on Linux, and Delphi 6/7 supports also CLX on Windows with Indy Components. However, Qt alone has a wider range of deployment e.g.:

AIX 4.1, FreeBSD 2.1, HP-UX 10.20, Solaris 2.5.1, Tru64 (Digital UNIX) 4.0, Windows NT and so on.

1.3 DWS Socket Server

We will concentrate now on creating TCP/IP applications with Delphi using Indy components in DWS. But, first of all I'll explain you what "blocking" and "non-blocking" sockets are. In fact there are 2 programming models used in TCP/IP applications.

Non Blocking means that the application will not be blocked when the application socket read/write data. This is efficient, because your application don't have to wait for a connection. Unfortunately, using this technique is little complicated to develop a protocol. If your protocol has many commands, your code will be very unintelligible.

The most important elements that I will present here are: `TIdTcpClient` and `TIdTcpServer` (see Abb. 1) components. And I will do that because these 2 components are the base stones of the Indy architecture. All client components are inherited from `TIdTcpClient` and all server components are inherited from `TIdTcpServer` (there are some exception). DWS use the non blocking technique.

First you have to define the file list with the apps to load. The standard file name is: "**binaries3.txt**". **Don't edit this file directly**, cause of the platform independence you have to do this in the DWS Server form on the right side in the String Grid at the Loader List (Abb. 2).

You can set an absolute path or a relative one in the field "Application Name".

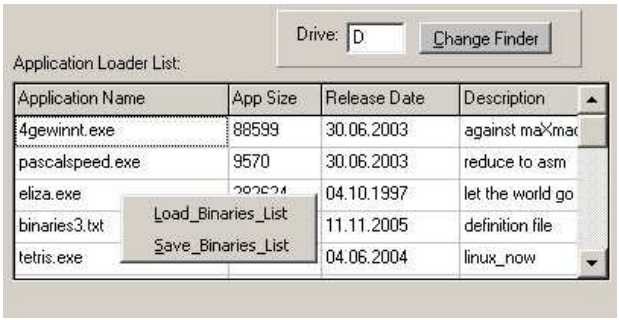

Abb. 2: The Loader List

With a right mouse click on the pop up menu you can store and load this file or use different definition files to load and store from.

Next, if a data file with the standard name already exists, it is opened and the previously stored data is read from it. In our case the data is written to the string grid (which is serving both as a memory structure for holding the data and as a visual control for navigating and editing the data).

We need internally a constructor to pass the grid and the filename, so the class `TBuildAppGrid` is independent from a form namespace, no uses like `frmUnit` is needed, just uses QGrids as a dependency.

This CLX example of direct access presented so far assumes that only a single user can access the files. If two or more users (or applications) can be permitted to access the data simultaneously, your code must also be build to resolve competition for records. Next, we have to store the data after editing:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  myDatFile:= 'binaries3.txt';
  myGridC:= TBuildAppGrid.initGrid(strGrd, myDatFile);
  myGridC.fillGrid;
end;
```

The data structure is a direct file access. In this case, rather than populating a stand-alone memory structure, the data is written to the String Grid (which is serving both as a memory structure for holding the data and as a visual control for navigating and editing the data).

You can always use a String Grid to display a Query Result, but if you don’t need the overhead of a data aware component or you don't want a database in an embedded system for ex., direct file access is the choice.

But as a developer you are responsible for every aspect of the data access.
In short, before any data access can occur, data must be read from one or more files and stored in memory.

```
type TAppData = record
   Name: string[50];
   Size: longint;
   Release: string[30];
   descript: string[80];
end;

  TbuildAppGrid = class (Tobject)
  private
    aGrid: TstringGrid;
    app: TappData;
    f: file of TappData;
    FaDatfile: ShortString;
    Fmodified: Boolean;
  protected
    function GetaDatfile: ShortString;
    procedure SetaDatfile(const Value: ShortString);
  public
    constructor initGrid(vGrid: TstringGrid; vFile: shortString);
    procedure fillGrid;
    procedure storeGrid;
    property aDatfile: ShortString read GetaDatfile write SetaDatfile;
    property modified: Boolean read Fmodified write Fmodified;
  end;
```

On the left side of the DWS server form you can see the app monitor. This is the view the client gets when the "Get Files" button is pressed. It shows the files prepared to load. In the Abb. 3 below or Abb. 5 you can see server and client together. Just for testing or broadcasting you can use the monitor to write directly some information in it so the client gets the input (content) like a messaging system after press the button.
The logger below the app monitor in the form traces these activities.

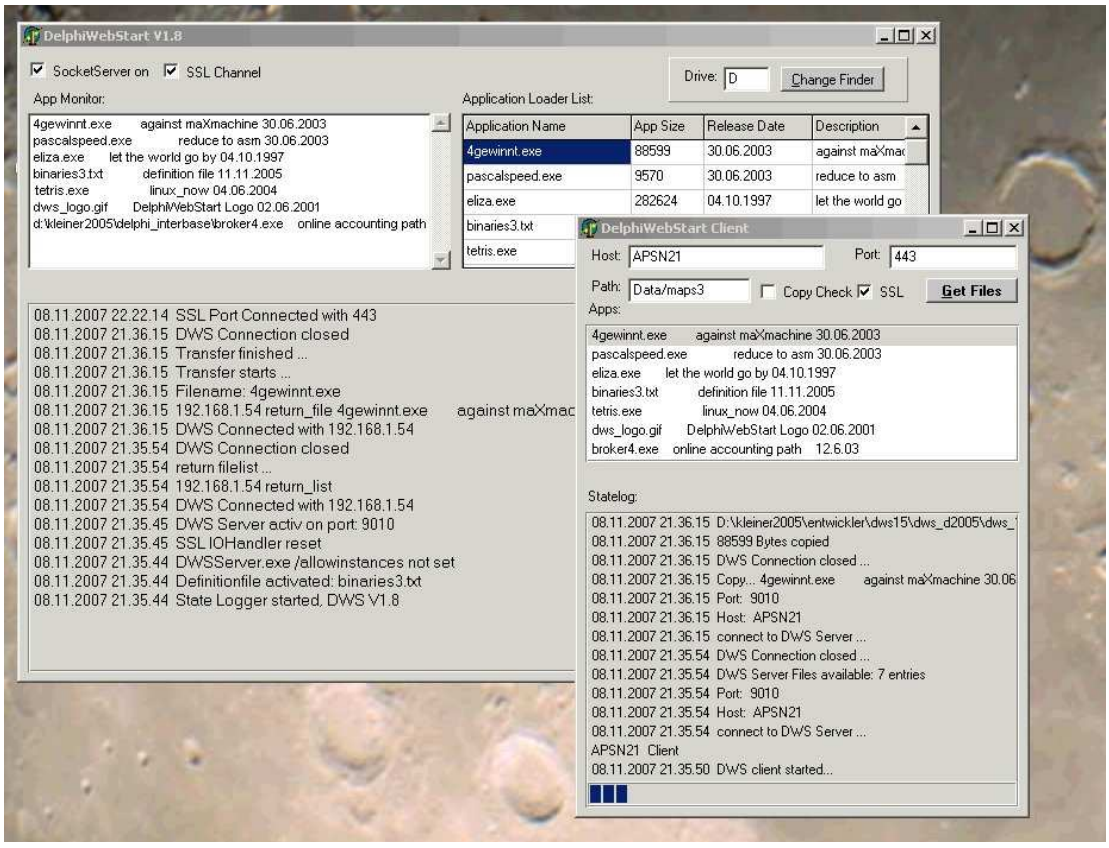

Abb. 3: DWS Server and Client

First we had to choose between an ftp and or a TCP solution. The advantage of TCP is the freedom to define a separate port, which is "**services, port 9010 – DelphiWebStart**".
Internally as you know you will need Indy in the source code. (Because Indy is simple to use and very fast). The TCP-server comes from Indy which has one great advantage:
"CommandHandlers" are a collection of text commands that will be processed by the server. This property greatly simplifies the process of building servers based on text protocols.

First we start with DWS_Server, so we define two command handlers:

```
CTR_LIST = 'return_list';
CTR_FILE = 'return_file';
```

by starting the TCP-server it returns with the first command

```
handler "CTR_LIST" a list of the apps:
```

```
procedure TForm1.IdTCPServer1Execute(AThread: TIdPeerThread);
...
// comes with writeline from client
if sRequest = CTR_LIST then begin
  for idx:= 0 to meData.Lines.Count - 1 do
  athread.Connection.WriteLn(ExtractFileName(meData.Lines[idx]));
```

```
  aThread.Connection.WriteLn('::END::');
  aThread.Connection.Disconnect;
```

One word concerning the thread: In the internal architecture there are 2 threads categories.

First is a listener thread that “listens” and waits for a connection. So we don't have to worry about threads, the built in thread will be served by Indy though parameter:

```
IdTCPServer1Execute(AThread: TIdPeerThread)
```

When our DWS-client is connected, this thread transfer all the communication operations to another thread. This technique is very efficient because your client application will be able to connect any time, even if there are many different connections to the server.
The second command "CTR_FILE" transfers the app to the client:

```
if Pos(CTR_FILE, sRequest) > 0 then begin
    iPos:= Pos(CTR_FILE, sRequest);
    FileName:= GetFullPath(FileName);

    if FileExists(FileName) then begin
      lbStatus.Items.Insert(0, Format('%-20s %s',
        [DateTimeToStr(now), 'Transfer starts ...']));
     FileStream:= TFileStream.Create(FileName, fmOpenRead +fmShareDenyNone);
     aThread.Connection.OpenWriteBuffer;
     aThread.Connection.WriteStream(FileStream);
     aThread.Connection.CloseWriteBuffer;
     FreeAndNil(FileStream);
     aThread.Connection.Disconnect;
```

Test your server with the telnet program. After connecting with host and the port address 9010, type "return_list" and you'll see a first result of the list. I know that we haven't implemented an error handling procedure, but for our scope this example is almost sufficient. The DWS-source holds version 1.8 and has the following references:

- Video On Demand Loader
- CBT Multiple Choice Trainer
- Maps or GIS Server
- Application Loader of executables
- Push or Pull Services to exchange messages
- Real time FTP, monitoring and so on

Abb. 4: the DWS Logo is share like Software ;)

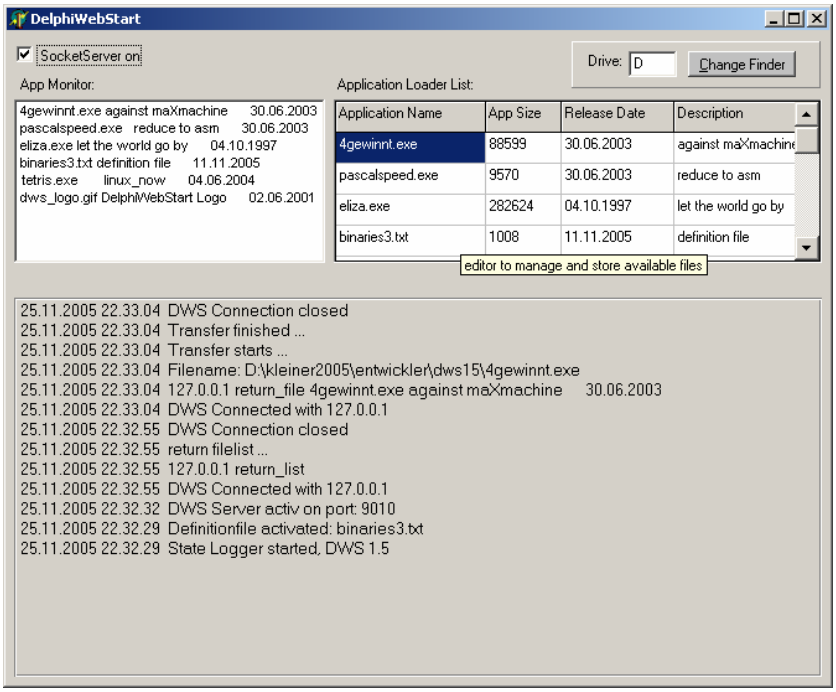

Abb. 5: DWS Server is running

1.3.1 Change Finder

Since V 1.5 you can find the button “Change Finder”. Where or when a modification has changed an existing file or a new one, the Change Finder can show the files in a **listview** by searching the now system date on your drives.
The changed files are shown with attributes, size and path names rather than numbers making it easier to see what's going on your storage system or hard drive.
This should make it easier to determine which files had changed and diminish the chance that a change is causing an undiscovered impact (spy ware, caching, integrity etc.).
First we need a class:

```
Type
  TChangeFinder = class
  private
    nYear,
    nMonth,
    nDay: word;
    dflistView: TListView;
  protected
    procedure ShowFiles (Showpath: string; sr: TSearchRec);
  public
    constructor Create_prepList_and_Date(alistView: TListView);
    procedure SearchDirectories(path: string; const fname: string);
  end;
```

The call from the client needs an instance of a 3 column TListView or **ListBox** and the chosen drive letter:

```
procedure TmainForm1.cfinderClick(Sender: TObject);
//tshChangefind: TTabSheet, pageindex=5, event= onShow
var drive: string[20];
```

```
    mycf: TChangeFinder;
begin
  screen.cursor:= crHourglass;
    drive:= dcbHD.Drive;
    drive:= drive + ':';
    mycf:= TChangeFinder.prepList_and_Date(mainForm1.view);
    mycf.SearchDirectories(drive + '\','*.*');
    mycf.Free;
  screen.cursor:=crDefault;
end;
```

The form independent unit itself scans for all files with the system date of today or now so you get a list in a listview or listbox for a file change detection system for Web Servers, protocol machines, loggers, databases or something else useful ;).
We use Change Finder once per day so the footprint is as small as possible (ca. 5 sec.). Call backs (asynchronous) are smart but in this way a disadvantage, because we expand the loop to find other information event driven by a script. So it's impossible to generate a report called by a script at once or on your time with call backs. Change Finder provides an overview of files changed in a day.
File versions can be then sliced, summarized, and restricted by date, file type an so on. One report lists all files changed in the past week with who changed them or list files changed recently, but which hadn't been touched in a long time before.
So the more call-backs you had to register the more complicated the maintenance.

1.3.2 OpenSSL

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

Latest compile of the OpenSSL libraries with the Indy modifications built into.

These files have been built from the OpenSSL version 0.9.6m source code from http://www.openssl.org

Since intellicom published their recommendation regarding how to compile OpenSSL with the Indy modification, a couple of functions have been added to the OpenSSL exports, requiring the Indy functions to be renumbered.

Finally, you can retrieve the libeay32.dll and ssleay32.dll files from the zip folder and place them in your application's folder or system32 folder.

Furthermore you have to set the certificate path in the `ip_a.ini` file:

[CERT]

ROOTCERT=cert\CA_indy_cert.pem

SCERT=cert\client_indy_cacert.pem

RSAKEY=cert\client_indy.pem

```
IdTCPServer1.IOHandler:= IdServerIOHandlerSSL1;
      with IdServerIOHandlerSSL1.SSLOptions do begin
        Method:= sslvSSLv3;
        Mode:= sslmServer;
        AppDir:= ExtractFilePath(Application.ExeName);
        RootCertFile:= AppDir +_IniFile.ReadString('CERT', 'ROOTCERT', '');
        CertFile:= AppDir +_IniFile.ReadString('CERT', 'SCERT', '');
        KeyFile:= AppDir +_IniFile.ReadString('CERT', 'RSAKEY', '');
        VerifyMode:= [sslvrfPeer];
      end;
```

OpenSSL includes a command line utility that can be used to perform a variety of cryptographic functions like generating your machine certificate in [CERT]. It is described in the openssl(1) manpage. Documentation for developers is currently being written. A few manual pages already are available; overviews over libcrypto and libssl are given in the crypto(3) and ssl(3) manpages. [i]

There is some documentation about certificate extensions and PKCS#12 in doc/openssl.txt

The original SSLeay documentation is included in OpenSSL as doc/ssleay.txt. It may be useful when none of the other resources help, but please note that it reflects the obsolete version SSLeay 0.6.6.

1.4 DWS Client

Now let's have a look at the client side. The client connects to the server, using the connect method of `TIdTcpClient`. In this moment, the client sends any command to the server, in our case (you remember DelphiWebStart) he gets the list of available apps:

```
    with IdTCPClient1 do begin
      if Connected then DisConnect;
      showStatus;
      Host:= edHost.Text;
      Port:= StrToInt(edPort.Text);
      Connect;
      WriteLn(CTR_LIST);
```

After double clicking on his choice, the app will be served:

```
with IdTCPClient1 do begin
  ExtractFileName(lbres.Items[lbres.ItemIndex])]));
  WriteLn(CTR_FILE + lbres.Items[lbres.ItemIndex]);
  FileName:= ExpandFileName(edPath.Text + '/' +
             ExtractFileName(lbres.Items[lbres.ItemIndex]));
...
  FileStream:= TFileStream.Create(FileName, fmCreate);
   while connected do begin
     ReadStream(FileStream, -1, true);
   ....
   execv(pchar(filename),NIL);
```

Better is source code with a compiler directive to load the delivered files:

```
     {$IFDEF LINUX}
        execv(pchar(filename),NIL);
      //libc.system(pchar(filename));
     {$ENDIF}
     {$IFDEF MSWINDOWS}
     // shellapi.WinExec('c:\testcua.bat', SW_SHOW);
     with lbstatus.items do begin
       case  shellapi.shellExecute(0,'open', pchar(filename), '',NIL,
                   SW_SHOWNORMAL) of
         0: insert(0, 'out of memory or resources');
         ERROR_BAD_FORMAT: insert(0, 'file is invalid in image');
         ERROR_FILE_NOT_FOUND: insert(0,'file was not found');
         ERROR_PATH_NOT_FOUND: insert(0,'path was not found');
       end;
       Insert(0, Format('%-20s %s',
              [DateTimeToStr(now), filename + ' Loaded...']));
     end
     {$ENDIF}
```

One note about execution on Linux with libc-commands; there will be better solutions (execute and wait and so

on) and we still work on it, so I'm curious about comments, therefore my aim is to publish improvements in a basic framework on sourceforge.net depends on your feedback ;)

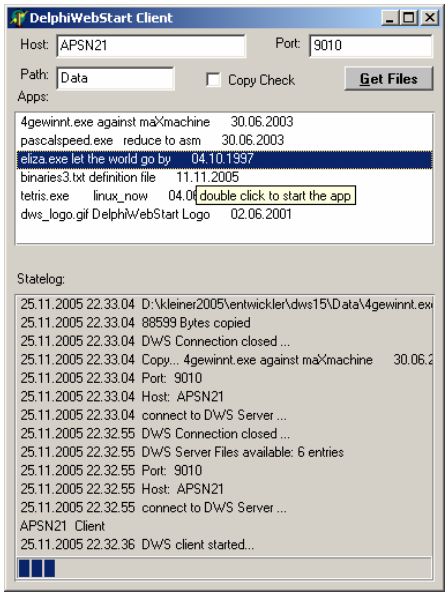

Abb. 6: DWS client with the loadable files

1.4.1 Copy Check

Copy Check is a small Version Checking so if the check box is activated once a file is copied in the Path directory and the name is the same no further transport is done over the net. Means you will use a name convention to differentiate the file versions (e.g. Belplan_1_6.exe) or without Copy Check[ii] each time the file will be delivered over the net. The same goes with OpenSSL (activated or not)!

But you can't change dynamically between encrypted or not encrypted at runtime on the Server!

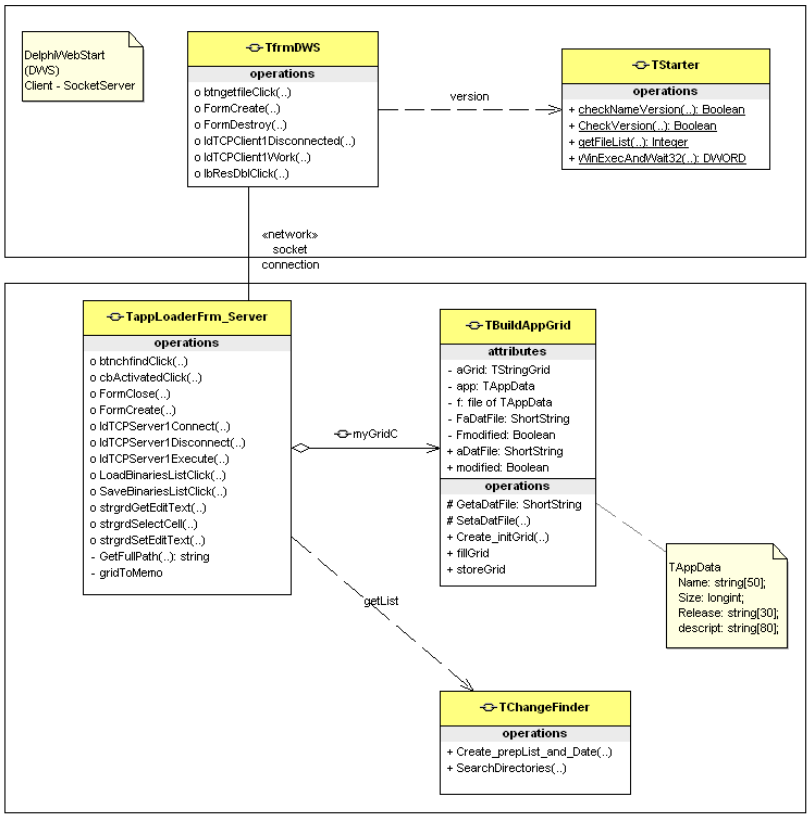

Abb. 7: UML Class Model of DWS

In the end let's have a look at the class diagram in Abb. 6 separated in client and server classes. The Server holds an association to `TChangeFinder` and the client an association to `TStarter` for Version Checking.

21.11.2005 DWS 1.5 and deployment now available[iii]:
- new package with executables (win32) and **qtintf70.dll**
- recompile without change finder also on CLX and Linux
- source improvements, UML diagrams, tooltip descriptions
- load&store definition files
- change finder and copy checking
- better client disconnecting, monitor as shortmessages

20.02.2007 V1.6
- load files from any path on the server and some small corrections (getFullPath2),
- exit message to server, letTCPConnect, antifreeze component,

10.11.2007 V1.8 integration of OpenSSL with Server Authentication

Literature:

Kleiner et al., Patterns konkret, 2003, Software & Support

Links of DWS and OpenSSL:

http://max.kleiner.com/download/dws.zip

http://www.softwareschule.ch/

http://sourceforge.net/projects/delphiwebstart

http://www.openssl.org/support/faq.html

http://www.indyproject.org/index.en.aspx

---

[i] You can't change (the checkbox) with or without SSL at runtime

[ii] Not activated on checkbox

[iii] Max Kleiner, November 2005