A Better Build Process

by Primoz Gabrijelcic

The command line is my tool, my companion and my passion. Yes, I'm an old-timer! I'd rather have my fingers on a keyboard, if possible. This extends to compiling and building, too. In the good old times, you just started a compile or build from the command line and took a long nap. Or a short one, if there was a compile error somewhere at the beginning of a project (that's why my build batch file always included some form of beep as a final command). Now one hardly gets a chance to nap. Press `F9` and all is done.

I had to admit, when I moved to 32-bit Windows, I stopped running the compiler from the command line. It was so nice just to press `F9` and enjoy. Then more complicated projects came and sometimes I went back to basics. But I still compiled mostly in the IDE.

Then in October 1998 I found a nice article in *The Delphi Magazine: Creating A Delphi Build Process* by Dave Collie. A helpful command line introduction for younger members of the Delphi community and a reminder for me to stop compiling in the IDE and return to basics. So I studied a little, built some tools and searched the net. Finally, I prepared a basic build script, which I now just modify a little for each new project. So here it is, my build.bat batch file, some tools that I found or created, plus some tips and tricks. Thanks a lot, Dave, for pushing me back to CMD.EXE!

Release Build

The main motivation for command line compiling is to make a distinction between debug and release builds.

When I work on a project, I don't leave the IDE very much. I have various debugging `DEFINE`s set up in `Project|Options` and I compile or build strictly in the IDE. When everything is working correctly (I hope) I leave the IDE and create a release build on the command line. This enables me to set a completely different set of compiler directives and `DEFINE`s. For example, I always enable optimisation for a release build, but never for debugging purposes. My build batch file also includes commands to create distribution files in a special directory, increment the build number, compile my custom resources, make a backup to the network server, etc.

Up to Delphi 3, creating a release build was a very simple process. You just had to run `DCC32 -B <project name>` and set the appropriate parameters in dcc32.cfg (I keep a copy of this file in each project directory).

Delphi 4 messed things up a little. The IDE silently manages a file called <*project*>.cfg, which includes all the settings specified in `Project|Options`. This file is also used by the DCC32 command line compiler. Therefore, it doesn't help much if you specify `-$O+` in dcc32.cfg but have `-$O-` set in <*project*>.cfg. DCC32 will use the project-specific configuration file and ignore any global settings. The solution is simple: delete the project configuration file before compiling. This creates no problems for the Delphi IDE as the settings are also in the project .dof file.

Version Information

The aspect that was keeping me from creating a completely self-contained build script was the version information resource. I'm using this (see `Project|Options`, `Version Info`) in all my projects. While working in the IDE everything is very simple. You just set version information, check `Include version information in project` and recompile. Version information is written into a project resource file (<*project*>.res) and linked into executable. A piece of cake.

There is no problem even when working from the command line. DCC32 will happily reuse the .res file created by the Delphi IDE. But what if the project resource is deleted somehow? Then you have to fire up Delphi, open up the project options, close the dialog and save all to create a new .res file. A real pain. It would be nice if we could create the project resource from the command line, but Delphi offers no solution.

I must admit that I did not see a simple solution to this problem. I played with it a little and then forgot about it. I had to live with the problem. Until the day when it occurred to me that I could very probably find the solution to my problem on the internet. So I connected to Deja News and searched. To cut a long story short, I found a solution. To be exact, I found *the* solution. Only one on the whole internet. Now that's something!

The creator of that simple but effective solution is Hector Santos (Hector, if you're reading this, thanks a lot!). He noticed that all the required information is stored in the project settings file (.dof), so he created a console application that opened a settings file, read the appropriate keys and created a resource source file (.rc). After that, you just have to run the resource compiler (BRCC32, included with Delphi), which compiles this source file to give us a compiled resource file (.res). now we have our resource.

Originally, though, Hector's solution had some problems. Besides some bugs, it did not include the project icon into the resource, but that was really easy to fix.

The resulting console program, MakePrjRes, is shown in Listing 1. It takes up to three parameters: the name of settings file, the name of project icon and the name of the resource file to create. It makes a resource file containing the version information resource and the icon resource, like the one shown

in Listing 2. It is also included on this month's disk of course.

Build Batch File

Let's take a look at a real-life build script now. It prepares a release build for my freeware Delphi profiler, GpProfile. For your convenience it uses only plain COMMAND.COM functionality. It is of course possible to create much smarter scripts in the 4DOS or 4NT command language, VBScript, Perl, or whatever, but I found that COMMAND.COM works for me. The script is displayed in its entirety in Listing 3. I will use label names (eg :start) to refer to various parts of this script.

➤ *Listing 1*

```
{$APPTYPE CONSOLE}
{$H+,O+}
{Based on the work of Hector Santos}
program MakePrjRes;
uses
  Windows, SysUtils, IniFiles;
const
  HexNibble: array [0..15] of char = '0123456789ABCDEF';
function HexByte(b: byte): string;
  begin
    HexByte := HexNibble[b shr 4]+HexNibble[b AND $F];
  end; { HexByte }
function HexWord(w: word): string;
  begin
    Result := HexByte(w shr 8)+HexByte(w AND $FF);
  end; { HexWord }
function OptAddExtension(fName, ext: string): string;
  begin
    if Pos('.',fName) = 0 then
      Result := fName+'.'+ext
    else
      Result := fName;
  end; { OptAddExtension }
procedure MakeResource(dofFile, icoFile, rcFile: string);
var
  rcf: text;
  procedure WriteVersionResource;
    var
      ini     : TIniFile;
      sect    : string;
      sBuild  : string;
      locale  : word;
      codepage: word;
      flags   : string;
    procedure AddFlag(fl: string);
      begin
        if flags <> '' then flags := flags + ' | ';
        flags := flags + fl;
      end;
    begin
      // TIniFile opens file in windows dir by default!
      if Pos('\',dofFile) = 0 then
        dofFile := '.\'+dofFile;
      if not FileExists(dofFile) then begin
        Writeln('MakeProjectResource 1.01');
        Writeln('file ',dofFile,' does not exist!');
        Halt(1);
      end;
      ini := TIniFile.Create(dofFile);
      try
        Writeln(rcf, 'VS_VERSION_INFO VERSIONINFO');
        sect:='Version Info';
        sBuild:=ini.ReadString(sect, 'MajorVer', '1');
        sBuild:=
          sBuild+','+ini.ReadString(sect,'MinorVer','0');
        sBuild:=
          sBuild+','+ini.ReadString(sect,'Release','0');
        sBuild:=
          sBuild+','+ini.ReadString(sect,'Build','0');
        Writeln(rcf, 'FILEVERSION      ', sBuild);
        Writeln(rcf, 'PRODUCTVERSION   ', sBuild);
        flags := '';
        if ini.ReadInteger(sect,'Debug',0) = 1 then
          AddFlag('VS_FF_DEBUG');
        if ini.ReadInteger(sect,'PreRelease',0) = 1 then
          AddFlag('VS_FF_PRERELEASE');
        if ini.ReadInteger(sect,'Special',0) = 1 then
          AddFlag('VS_FF_SPECIALBUILD');
        if ini.ReadInteger(sect,'Private',0) = 1 then
          AddFlag('VS_FF_PRIVATEBUILD');
        if flags = '' then
          flags := '0';
        sect:='Version Info Keys';
        Writeln(rcf, 'FILEFLAGSMASK  VS_FFI_FILEFLAGSMASK');
        Writeln(rcf, 'FILEFLAGS  ', flags);
        Writeln(rcf, 'FILEOS   VFT_APP');
        Writeln(rcf, 'FILESUBTYPE  VFT2_UNKNOWN');
        Writeln(rcf, 'BEGIN');
        Writeln(rcf, '  BLOCK "VerFileInfo"');
        Writeln(rcf, '  BEGIN');
        locale := Ini.ReadInteger(sect,'Locale',$409);
        codepage := Ini.ReadInteger(sect,'CodePage',1252);
        Writeln(rcf, '    Value "TRANSLATION",
          0x',HexWord(locale),', ',codepage);
        Writeln(rcf, '  END');
        Writeln(rcf, '  BLOCK "STRINGFILEINFO"');
        Writeln(rcf, '  BEGIN');
        Writeln(rcf, '    BLOCK
          "',HexWord(locale),HexWord(codepage),'"');
        Writeln(rcf, '    BEGIN');
        Writeln(rcf, '      VALUE "CompanyName",
          "',Ini.ReadString(Sect,'CompanyName',''),' \0"');
        Writeln(rcf, '      VALUE "FileDescription",
          "',Ini.ReadString(Sect,'FileDescription',''),
          '\0"');
        Writeln(rcf, '      VALUE "FileVersion",
          "',Ini.ReadString(Sect,'FileVersion',sBuild),
          ' \0"');
        Writeln(rcf, '      VALUE "InternalName",
          "',Ini.ReadString(Sect,'InternalName',''),' \0"');
        Writeln(rcf, '      VALUE "LegalCopyright",
          "',Ini.ReadString(Sect,'LegalCopyRight',''),
          ' \0"');
        Writeln(rcf, '      VALUE "LegalTrademarks",
          "',Ini.ReadString(Sect,'LegalTrademarks',''),
          ' \0"');
        Writeln(rcf, '      VALUE "OriginalFileName",
          "',Ini.ReadString(Sect,'OriginalFileName',''),
          ' \0"');
        Writeln(rcf, '      VALUE "ProductName",
          "',Ini.ReadString(Sect,'ProductName',''),' \0"');
        Writeln(rcf, '      VALUE "ProductVersion",
          "',Ini.ReadString(Sect,'ProductVersion',''),
          ' \0"');
        Writeln(rcf, '      VALUE "Comments",
          "',Ini.ReadString(Sect,'Comments',''),' \0"');
        Writeln(rcf, '    END');
        Writeln(rcf, '  END');
        Writeln(rcf, 'END');
      finally
        ini.free; end;
    end; { WriteVersionResource }
  procedure WriteIconResource;
    begin
      if not FileExists(icoFile) then begin
        Writeln('MakeProjectResource 1.01');
        Writeln('file ',icoFile,' does not exist!');
        Halt(1);
      end;
      Writeln(rcf,'MAINICON ICON "',icoFile,'"');
    end; { WriteIconResource }
  begin
    AssignFile(rcf, rcFile);
    Rewrite(rcf);
    if icoFile <> '-' then
      WriteIconResource;
    if dofFile <> '-' then begin
      if icoFile <> '-' then Writeln(rcf);
      WriteVersionResource;
    end;
    CloseFile(rcf);
  end; { MakeResource }
procedure Usage;
  begin
    Writeln('MakeProjectResource 1.01');
    Writeln(
      'Usage: makeprjres dof_file ico_file [rc_file]');
    Writeln(
      '       makeprjres - ico_file [rc_file]');
    Writeln(
      '       makeprjres dof_file - [rc_file]');
    Halt(1);
  end; { Usage }
var
  dofFile: string;
  icoFile: string;
  rcFile : string;
begin
  if (ParamCount < 2) or (ParamCount > 3) then
    Usage;
  dofFile := ParamStr(1);
  icoFile := ParamStr(2);
  if icoFile <> '-' then begin
    OptAddExtension(icoFile,'ICO');
    rcFile := ChangeFileExt(icoFile,'.RC');
  end;
  if dofFile <> '-' then begin
    OptAddExtension(dofFile,'DOF');
    rcFile := ChangeFileExt(dofFile,'.RC');
  end;
  if ParamCount = 3 then
    rcFile := OptAddExtension(ParamStr(3),'RC');
  MakeResource(dofFile, icoFile, rcFile);
end.
```

The first thing to point out is that all my build scripts can create more than one executable. Even more, I can select with a command line parameter which programs should be built. The logic to implement this behaviour is programmed in the blocks `:start` and `:loop`. If we converted build.bat into pseudocode, it would look like Listing 4.

This approach offers great flexibility. The code is modularised (each compilation block only deals with one part of the project) and parts can be compiled at will, which is very useful when you have to troubleshoot problems which do not occur in the debug build.

All the compilation blocks follow the same structure. First, I delete the project configuration file (for the reasons I already mentioned) and the old executable (so I can see at a glance if the compilation was successful). Then I prepare the files needed for the project, compile it and do any required post processing. But let's take a look at these steps one by one.

As the GpProfile distribution contains a help file, it is prepared first. I copy the .hlp and .cnt files from the folder with the last help version. Then I run hpj2inc, a simple program that extracts help topic constants from a help project (.hpj) and stores them into a simple Pascal include file (.inc). That way, I can use symbolic names to access the help file from the program. You can use hpj2inc with your projects, too, as it is included on this month's disk.

This is probably a good moment to examine the error checking in build.bat. When calling any external command I make sure that the output is redirected into a file (build.log) and that the error code is checked immediately after control is returned to the batch file. If an error is found, control is transferred to the `:error` block, where this log file is displayed on the screen (using `type build.log | more`).

Next, I prepare all resource files used in the program (see `:gpprof_brcc_baggage`). First, I compile baggage.rc, a very simple

➤ *Listing 2*

```
MAINICON ICON "gpprofile.ico"
VS_VERSION_INFO VERSIONINFO
FILEVERSION      1, 3, 0, 5
PRODUCTVERSION   1, 3, 0, 5
FILEFLAGSMASK  VS_FFI_FILEFLAGSMASK
FILEFLAGS  0
FILEOS   VFT_APP
FILESUBTYPE  VFT2_UNKNOWN
BEGIN
  BLOCK "VerFileInfo"
  BEGIN
   Value "TRANSLATION", 0x0409, 1252
  END
  BLOCK "STRINGFILEINFO"
  BEGIN
    BLOCK "040904E4"
    BEGIN
     VALUE "CompanyName", " \0"
     VALUE "FileDescription", "Profiler for Delphi 2, 3, and 4. \0"
     VALUE "FileVersion", "1.3.0.5 \0"
     VALUE "InternalName", "GpProfile \0"
     VALUE "LegalCopyright", "(c) 1998, 1999 Primoz Gabrijelcic \0"
     VALUE "LegalTrademarks", " \0"
     VALUE "OriginalFileName", " \0"
     VALUE "ProductName", "GpProfile \0"
     VALUE "ProductVersion", "1.3 \0"
     VALUE "Comments", " \0"
    END
  END
END
```

➤ *Listing 3*

```
@echo off
:start
  set build_justone=1
  if %1.==gpprof. goto gpprof
  if %1.==gppunreg. goto gppunreg
  set build_justone=0
:gpprof
  echo Building GpProf
  echo.
    if exist gpprof.exe del gpprof.exe >nul
    if exist gpprof.cfg del gpprof.cfg >nul
  :gpprof_hpj2inc
    copy help\95nt\gpprof.hlp . >nul
    copy help\95nt\gpprof.cnt . >nul
    hpj2inc help\95nt\gpprof.hpj help.inc >build.log
    if not errorlevel 1 goto gpprof_brcc_baggage
    goto error
  :gpprof_brcc_baggage
    brcc32 -r baggage.rc >build.log
    if not errorlevel 1 goto gpprof_makeproj
    goto error
  :gpprof_makeproj
    makeprjres gpprof.dof gpprofile.ico gpprof.rc >build.log
    if not errorlevel 1 goto gpprof_brcc
    goto error
  :gpprof_brcc
    brcc32 gpprof.rc >build.log
    if not errorlevel 1 goto gpprof_dcc
    goto error
  :gpprof_dcc
    dcc32 -b -h- -w- -$r-,q-,c-,o+ gpprof.dpr >build.log
    if not errorlevel 1 goto gpprof_dccok
    head build.log 1
    tail build.log 10
    goto error_nolog
  :gpprof_dccok
    tail build.log 1
    incver gpprof.dof
    echo.
  :gpprof_ok
    aspack gpprof.exe /r+ /b+ /d+ /e+
    if not exist g:\programs\gpprofile\gpprof.exe md
      g:\programs\gpprofile >nul
    copy gpprof.exe g:\programs\gpprofile >nul
    copy gpprof.hlp g:\programs\gpprofile >nul
    copy gpprof.cnt g:\programs\gpprofile >nul
    copy gpprof.pas x:\mstpl\gp >nul
    copy gpprofh.pas x:\mstpl\gp >nul
    if %build_justone=1 goto loop
:gppunreg
  echo Building gppUnreg
  echo.
    if exist gppunreg.exe del gppunreg.exe >nul
    if exist gppunreg.cfg del gppunreg.cfg >nul
  :gppunreg_dcc
    dcc32 -b -h- -w- -$r-,q-,c-,o+ %1 %2 %3 %4 %5 %6 %7
      gppunreg.dpr >build.log
    if not errorlevel 1 goto gppunreg_dccok
    head build.log 1
    tail build.log 10
    goto error_nolog
  :gppunreg_dccok
    tail build.log 1
    echo.
  :gppunreg_ok
    if %build_justone=1 goto loop
:loop
  shift
  if %1.=. goto OK
  goto start
:error
  type build.log | more
:error_nolog
  echo.
  echo Error!
  goto exit
:OK
  if exist build.log erase build.log >nul
  set build_justone=
  echo Program(s) built successfully!
:exit
```

file that just includes two rich text format files:

```
#include "baggage.inc"
IDD_OPENSOURCE RCDATA
  "opensource.rtf"
IDD_WHATSNEW RCDATA
  "whatsnew.rtf"
```

The file baggage.inc stores symbolic constants used to access this file from the program:

```
const
  IDD_WHATSNEW = 1;
  IDD_OPENSOURCE = 2;
```

I display those files in the About box. This approach enables me to create a simple program history as a formatted RTF file and then just load it into a `RichEdit` component: see Listing 5.

The next two sections (`:gpprof_makeproj` and `:gpprof_brcc`) are easy to understand. I create the source for the project resource from the .dof and .ico files, then I compile it.

Next comes the compilation part (`:gpprof_dcc`). When making a release build I always recompile all sources (including third party components). Error processing is slightly different in this case. If compilation fails, I display the first line and last ten lines of the resulting log. The last ten lines are usually enough to identify the problem (if not, I can open the log file) and the first line identifies the compiler so I can be sure I'm using the right version of my tools. If compilation succeeds, I display only the last line of the log file, the one that contains the compilation statistics:

```
29926 lines, 7.39 seconds,
  697272 bytes code, 10705
  bytes data.
```

and then call the program IncVer, which increments the build number by one. How? Very simple, it opens the project settings file (.dof), locates the appropriate line and increments the last number in it. For this trick to work, the project must not be open in the Delphi IDE at that time, as the IDE silently overwrites the .dof file when the project is closed. For that reason, I always do `Close All` in the IDE and only then run build.bat.

Head, Tail and IncVer are three simple console applications, all included on this month's disk. Head and Tail (shown in Listing 6) are good examples of what can be done in a few lines with some VCL classes. They are also a good example of how not to program, so kids, don't do this in real world applications, Ok?

The post-processing part is usually different for each application. For GpProfile I compress the .exe with a great shareware packer: ASPack (www.entechtaiwan.com/aspack.htm). The main reason for this compression is that I found that the distribution package is smaller if I compress the .exe first, even though the installer does its own compression. Every byte counts if your users have to download the app from the internet.

At the end, I copy all the distribution files into a special folder where they are waiting to be packed into an installation package. I do most of installing with the excellent INF-Tool, which has the smallest footprint of all installation packages (remember, each byte counts). For more details check http://inner-smile.com.

That is about it. We have walked the build process together from start to finish, and now you're ready to do it yourself. So go forth and use the command line!

Primoz Gabrijelcic is an avid command line user, who hopes that the good old DOS box will not be removed from future Windows incarnations. He likes to create small command line utilities, which can be freely used, reused, and abused (that goes for all the code published in this article, too).

➤ *Above: Listing 4*

```
:start
  Fetch first command line parameter.
  If parameter is empty, set compile all flag.
:jump
  If parameter is not empty, jump to appropriate compilation block.
:program_1
  Compile first program.
  If not compile all jump to :loop.
...
:program_n
  Compile nth program.
  If not compile all jump to :loop.
:loop
  Fetch next parameter.
  If parameter is empty, exit.
  Else jump to :jump.
```

➤ *Below: Listing 5*

```
{$R BAGGAGE.RES}
{$I BAGGAGE.INC}
var
  verInfo: TGpVersionInfo;
  stream : TResourceStream;
begin
  stream := TResourceStream.CreateFromID(HInstance, IDD_WHATSNEW, RT_RCDATA);
  try RichEdit1.Lines.LoadFromStream(stream);
  finally stream.Free; end;
end;
```

➤ *Listing 6*

```
{$APPTYPE Console}
program tail;
uses
  SysUtils, Classes;
var
  str  : TStringList;
  i    : integer;
  first: integer;
begin
  str := TStringList.Create;
  try
    str.LoadFromFile(ParamStr(1));
    first := str.Count-StrToIntDef(ParamStr(2),20);
    if first < 0 then first := 0;
    for i := first to str.Count-1 do Writeln(str[i]);
  finally
    str.Free;
  end;
end.
```