Dave's Development Blog

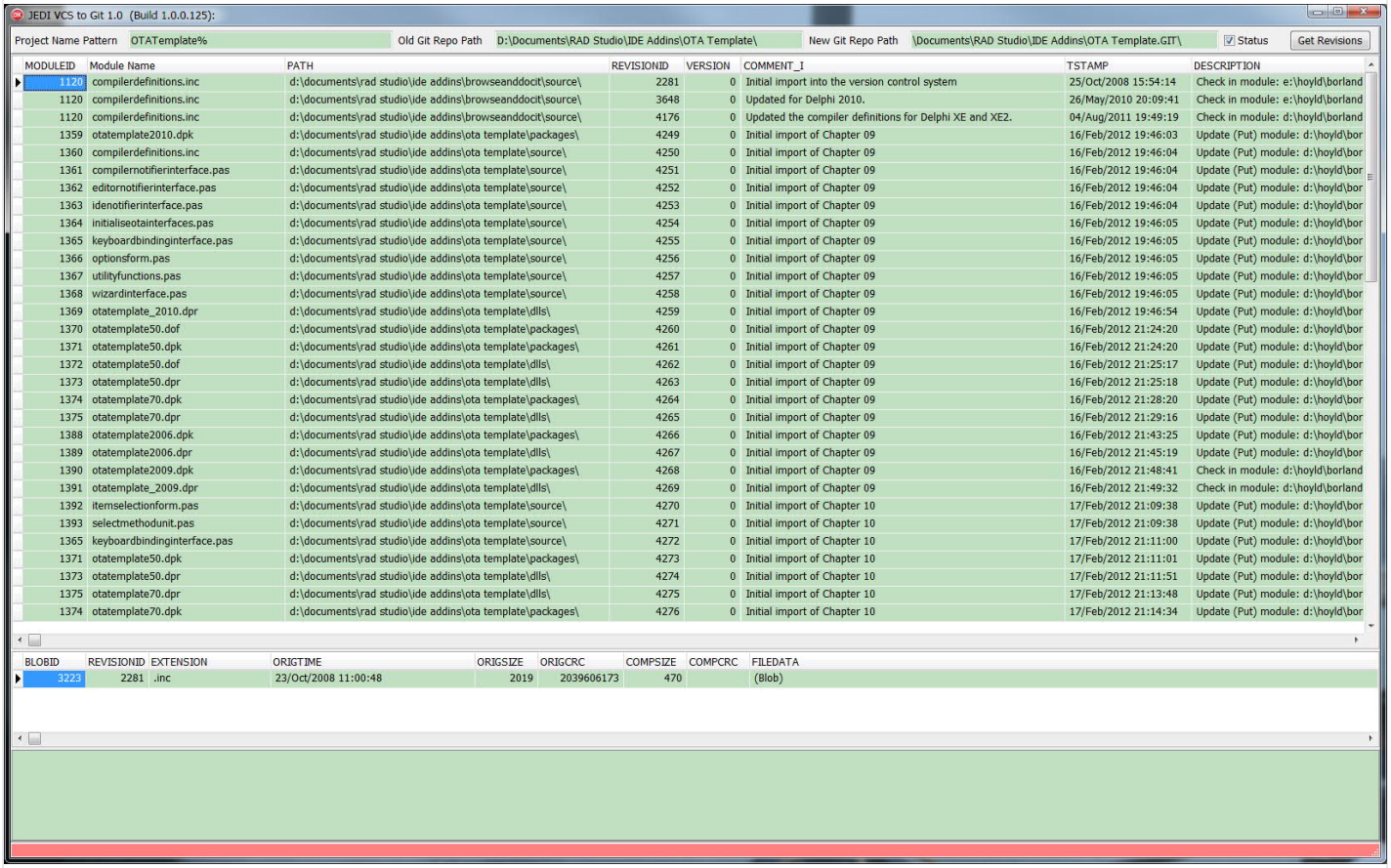
Software Development using Borland / Codegear / Embarcadero RAD Studio

Migrating from JEDI VCS to Git

By David | June 17, 2018 0 Comment

Overview

The purpose of this article is to describe how I migrated my RAD Studio projects from JEDI VCS to Git using a custom Delphi application so that other can either use the code as is if it satisfies their needs or provide enough information about how the JEDI VCS database is structured so that others can use my application as a starting point.



MODULEID	Module Name	PATH	REVISIONID	VERSION	COMMENT_I	TSTAMP	DESCRIPTION
1120	compilerdefinitions.inc	d:\documents\rad studio\ide addins\browseanddocit\source\	2281	0	Initial import into the version control system	25/Oct/2008 15:54:14	Check in module: e:\hoyld\borland
1120	compilerdefinitions.inc	d:\documents\rad studio\ide addins\browseanddocit\source\	3648	0	Updated for Delphi 2010.	26/May/2010 20:09:41	Check in module: e:\hoyld\borland
1120	compilerdefinitions.inc	d:\documents\rad studio\ide addins\browseanddocit\source\	4176	0	Updated the compiler definitions for Delphi XE and XE2.	04/Aug/2011 19:49:19	Check in module: d:\hoyld\borland
1359	otatemplate2010.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4249	0	Initial import of Chapter 09	16/Feb/2012 19:46:03	Update (Put) module: d:\hoyld\bor
1360	compilerdefinitions.inc	d:\documents\rad studio\ide addins\ota template\source\	4250	0	Initial import of Chapter 09	16/Feb/2012 19:46:04	Update (Put) module: d:\hoyld\bor
1361	compilernotifierinterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4251	0	Initial import of Chapter 09	16/Feb/2012 19:46:04	Update (Put) module: d:\hoyld\bor
1362	editornotifierinterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4252	0	Initial import of Chapter 09	16/Feb/2012 19:46:04	Update (Put) module: d:\hoyld\bor
1363	idenotifierinterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4253	0	Initial import of Chapter 09	16/Feb/2012 19:46:04	Update (Put) module: d:\hoyld\bor
1364	initialiseotainterfaces.pas	d:\documents\rad studio\ide addins\ota template\source\	4254	0	Initial import of Chapter 09	16/Feb/2012 19:46:05	Update (Put) module: d:\hoyld\bor
1365	keyboardbindinginterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4255	0	Initial import of Chapter 09	16/Feb/2012 19:46:05	Update (Put) module: d:\hoyld\bor
1366	optionsform.pas	d:\documents\rad studio\ide addins\ota template\source\	4256	0	Initial import of Chapter 09	16/Feb/2012 19:46:05	Update (Put) module: d:\hoyld\bor
1367	utilityfunctions.pas	d:\documents\rad studio\ide addins\ota template\source\	4257	0	Initial import of Chapter 09	16/Feb/2012 19:46:05	Update (Put) module: d:\hoyld\bor
1368	wizardinterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4258	0	Initial import of Chapter 09	16/Feb/2012 19:46:05	Update (Put) module: d:\hoyld\bor
1369	otatemplate_2010.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4259	0	Initial import of Chapter 09	16/Feb/2012 19:46:54	Update (Put) module: d:\hoyld\bor
1370	otatemplate50.dof	d:\documents\rad studio\ide addins\ota template\packages\	4260	0	Initial import of Chapter 09	16/Feb/2012 21:24:20	Update (Put) module: d:\hoyld\bor
1371	otatemplate50.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4261	0	Initial import of Chapter 09	16/Feb/2012 21:24:20	Update (Put) module: d:\hoyld\bor
1372	otatemplate50.dof	d:\documents\rad studio\ide addins\ota template\dlls\	4262	0	Initial import of Chapter 09	16/Feb/2012 21:25:17	Update (Put) module: d:\hoyld\bor
1373	otatemplate50.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4263	0	Initial import of Chapter 09	16/Feb/2012 21:25:18	Update (Put) module: d:\hoyld\bor
1374	otatemplate70.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4264	0	Initial import of Chapter 09	16/Feb/2012 21:28:20	Update (Put) module: d:\hoyld\bor
1375	otatemplate70.dof	d:\documents\rad studio\ide addins\ota template\dlls\	4265	0	Initial import of Chapter 09	16/Feb/2012 21:29:16	Update (Put) module: d:\hoyld\bor
1388	otatemplate2006.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4266	0	Initial import of Chapter 09	16/Feb/2012 21:43:25	Update (Put) module: d:\hoyld\bor
1389	otatemplate2006.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4267	0	Initial import of Chapter 09	16/Feb/2012 21:45:19	Update (Put) module: d:\hoyld\bor
1390	otatemplate2009.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4268	0	Initial import of Chapter 09	16/Feb/2012 21:48:41	Check in module: d:\hoyld\borland
1391	otatemplate_2009.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4269	0	Initial import of Chapter 09	16/Feb/2012 21:49:32	Check in module: d:\hoyld\borland
1392	itemselectionform.pas	d:\documents\rad studio\ide addins\ota template\source\	4270	0	Initial import of Chapter 10	17/Feb/2012 21:09:38	Update (Put) module: d:\hoyld\bor
1393	selectmethodunit.pas	d:\documents\rad studio\ide addins\ota template\source\	4271	0	Initial import of Chapter 10	17/Feb/2012 21:09:38	Update (Put) module: d:\hoyld\bor
1365	keyboardbindinginterface.pas	d:\documents\rad studio\ide addins\ota template\source\	4272	0	Initial import of Chapter 10	17/Feb/2012 21:11:00	Update (Put) module: d:\hoyld\bor
1371	otatemplate50.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4273	0	Initial import of Chapter 10	17/Feb/2012 21:11:01	Update (Put) module: d:\hoyld\bor
1373	otatemplate50.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4274	0	Initial import of Chapter 10	17/Feb/2012 21:11:51	Update (Put) module: d:\hoyld\bor
1375	otatemplate70.dpr	d:\documents\rad studio\ide addins\ota template\dlls\	4275	0	Initial import of Chapter 10	17/Feb/2012 21:13:48	Update (Put) module: d:\hoyld\bor
1374	otatemplate70.dpk	d:\documents\rad studio\ide addins\ota template\packages\	4276	0	Initial import of Chapter 10	17/Feb/2012 21:14:34	Update (Put) module: d:\hoyld\bor

BLOBID	REVISIONID	EXTENSION	ORIGTIME	ORIGSIZE	ORIGCRC	COMPSIZE	COMPCRC	FILEDATA
3223	2281	.inc	23/Oct/2008 11:00:48	2019	2039606173	470		(Blob)

The files for the project can be downloaded from https://github.com/DGH2112/JEDIVCSToGit.

Use

There are a number of steps to configure in order to use this application as described in the following sections.

FireDAC

The application uses FireDAC to communicate with the JEDI VCS database so in order for you to be able to use this project you need a version of RAD Studio which has FireDAC and also an appropriate driver for the database you are linking to.

For me, I had migrated the original JEDI VCS database which is implemented using DBISAM to Microsoft SQL Server a long long time ago for better performance and more importantly a better backup capability. The tools that came with JEDI VCS allowed me to do this once I had installed a SQL Server on my machine. I don't think that FireDAC has a DBISAM driver so if your repositories are still in that type of database you might want to migrate them to something else FireDAC supports first using the tools provided by JEDI VCS.

So the first thing you need to do in create a FireDAC INI file which you will pass to the application as the first parameter of the command line. This INI file defines the connection to use to connect the the database. My INI file looks like this:

```
SERVER=SEASONSFALL0001\SQLEXPRESS2008
OSAuthent=Yes
ApplicationName=JEDIVCSToGit
Workstation=SEASONSFALL0001
Database=JEDIVCS24
DriverID=MSSQL
```

The above tells FireDAC that the driver type is `MSSQL` (i.e. Microsoft SQL Server), the server is `SEASONSFALL0001\SQLEXPRESS2008`, the database is named `JEDIVCS24` and to use trusted authentication. The workstation and application name attributes I don't think are required however they will show up in MS SQL Server when examining the connections.

Once you have created the appropriate INI file, pass this as the first parameter to the application when launching the application, there will be a pause before the application appears. If there are any error your connection information needs to be address.

Project Filter

The first edit box on the top line of the application allows you to provide a filter pattern (in SQL language) to filter all the repositories in the database for just the one or two you want. Once you have provided a filter, press *Tab* to move to the next edit control and the top data grid should populate with a list of the commits in the JEDI VCS system for the filter pattern provided.

Existing Repository Location

You should provide here the original path to the JEDI VCS project you are migrating. It does not have to exist however it is used to understand the relative paths for the files in the project.

New Repository Location

You should provide a valid new directory for the application to write the files to which should be empty BUT also pre-initialised as a git repository.

Status

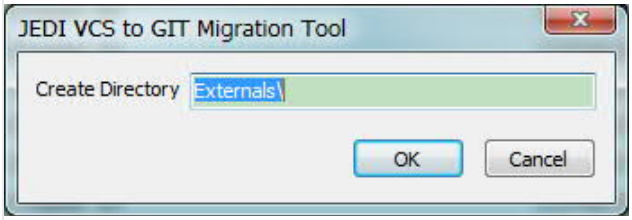
This is an optional item. If checked a `GIT STATUS` command will be run between the adds and commits. This is useful if you are having issues migrating a repository and are receiving error messages.

Migrating (Get Revisions)

To start the migration, press the *Get Revisions* button in the top right hand corner of the application.

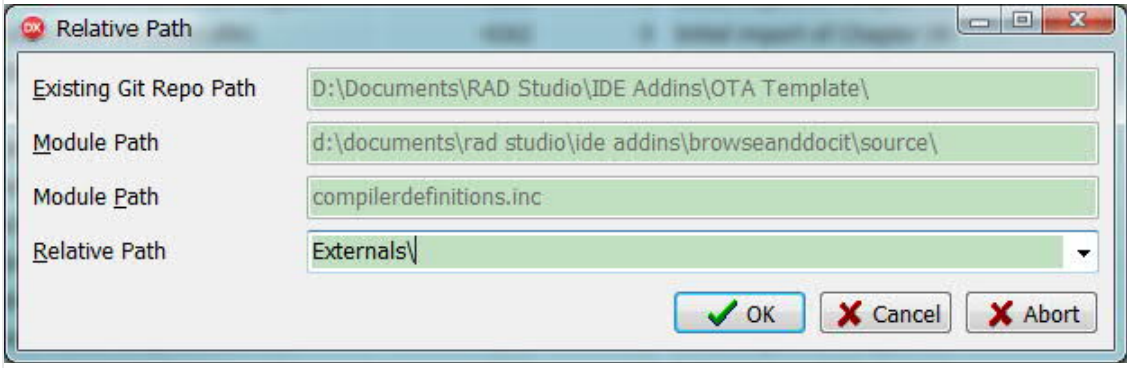
The application then walks through all the revisions / commits in the repository in chronological order extracting the file(s) for that revision / commit to the new repository location and commits them to *GIT* with the check-in comment from the JEDI VCS commit and the original date and time of the commit.

If the file being saved are from within the original repository location the file will be saved to the same relative path as it was in the original repository. If the directory does not exist you will be asked to confirm the directory name as below:



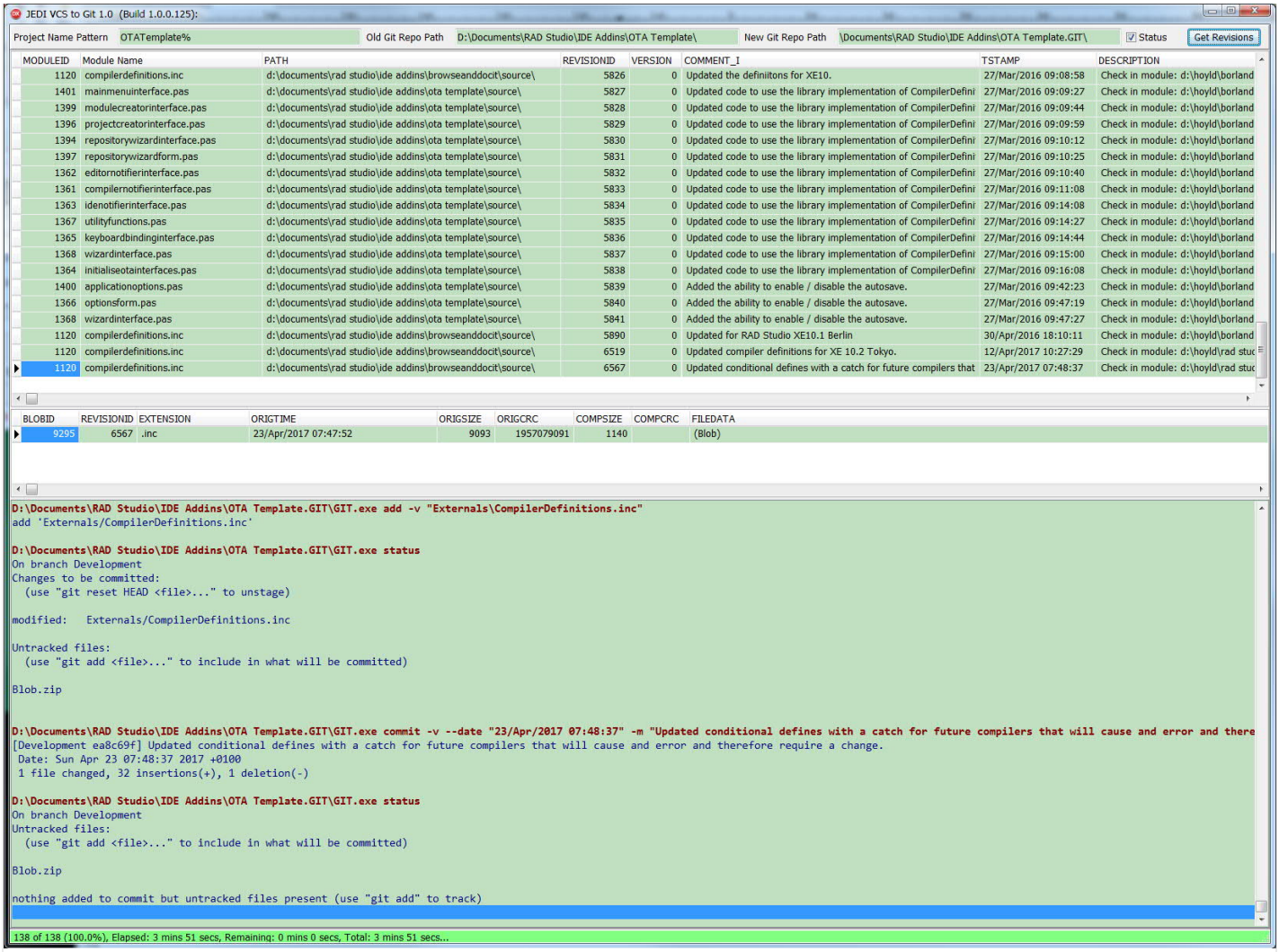
The reason I've done this is that JEDI VCS stores all the module names and path in lower-case letters and I would prefer them to be in a mixed case most of the time and this allows me to do this.

If files in the existing repository were not within the original repository path, i.e. an external reference from somewhere else you will be prompted for a new relative path location for the file(s) as shown below.



The files are written to the relative location using the case of the original file stored in the JEDI VCS blobs (which are actually zip files).

During the progress the log file at the bottom of the application is populated with the results of the GIT commands which are actually from a hidden console so you must ensure that `GIT` is on your path either in the system `PATH` variable or your profiles `PATH` variable.



A progress bar in the status bar plots the progress of the migration. Once the migration has finished you will have 2 extra files in the new Git repository: `Blog.zip` and `Git.log`. The blog file can be deleted as it was a temporary file used to extract the files from the JEDI VCS system. This log file is a record of the command line outputs during the migration.

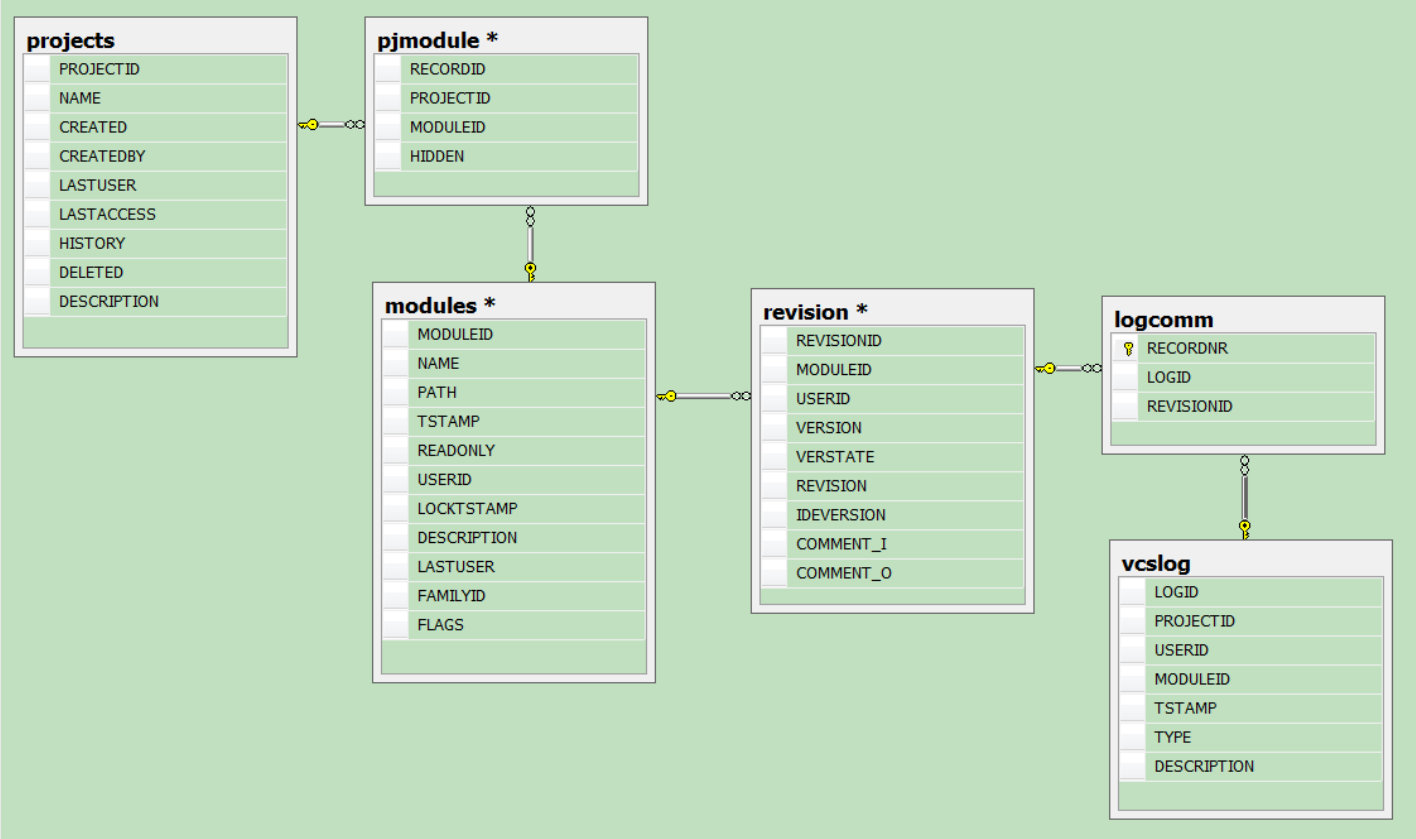
Implementation

How does it work? Well is starts with a complex query as follows:

```
SELECT DISTINCT
  M.MODULEID, M.NAME AS [Module Name], M.PATH,
  R.REVISIONID, R.VERSION, R.REVISION, R.COMMENT_I,
  VL.TSTAMP, VL.DESCRIPTION
FROM projects P
  INNER JOIN pjmodule PM ON P.PROJECTID = PM.PROJECTID
  INNER JOIN modules M ON PM.MODULEID = M.MODULEID
  INNER JOIN revision R ON M.MODULEID = R.MODULEID
  INNER JOIN logcomm L ON L.REVISIONID = R.REVISIONID
  INNER JOIN vcslog VL ON L.LOGID = VL.LOGID
WHERE'
```

```
  P.NAME LIKE !ProjectNamePattern
ORDER BY
  TSTAMP, MODULEID, VERSION, REVISION
```

The entity relationship diagram for this is as follows:



This isn't the end, there is still the `blog` table to consider (the detailed data grid) which is related to the `revision` table through the `revisionid`.

For me, I've been the only person working with my JEDI VCS repository and therefore I've not had to consider thinking about the user who committed the information. However, for any one wanting to enhance the project to include multiple users then the place to do that is with the `vcslog` table where the `userid` committing the information is recorded.

The application runs a command prompt in the background so the commits to the Git repository will be based on the default user profile (i.e the information stored either in your global git configuration or specific to the project). For multiple users, each commit would have to specify the user name / email address of the user committing the information and therefore you would have to write a mechanism for mapping this information in JEDI VCS system to Git.

Below are some of the methods from the main file with notes on what they do and how they works.

Function DGHFindOnPath(var strEXEName : String; Const strDirs : String) : Boolean

This method searches the given paths (a semi-colon delimited list if directories) and the environment `PATH` for the executable file name. If found the result is `true` and the full path to the executable file is returned in `strEXEName` parameter else the function returns false.

The method places all the directories in a string list as separate lines, checks these directories and then searches the paths for the executable.

```
Function DGHFindOnPath(var strEXEName : String; Const strDirs : String) : Boolean;

Const
  strPathEnVar = 'path';

Var
  slPaths : TStringList;

Begin
  slPaths := TStringList.Create;
  Try
    slPaths.Text := GetEnvironmentVariable(strPathEnVar);
    If strDirs <> '' Then
      slPaths.Text := strDirs + ';' + slPaths.Text;
    slPaths.Text := StringReplace(slPaths.Text, ';', #13#10, [rfReplaceAll]);
    CheckPaths(slPaths);
    Result := SearchPaths(slPaths);
  Finally
    slPaths.Free;
  End;
End;
```

Function SearchPaths(Const slPaths : TStringList) : Boolean

This function searches the paths in the given string list for the executable file and returns `true` if found and strEXEName is updated. This is a local method to `DGHFindOnPath` and therefore has access to the `strEXEName` `var` parameter of that method.

```
Function SearchPaths(Const slPaths : TStringList) : Boolean;

Var
  iPath: Integer;
  i : Integer;
  recSearch: TSearchRec;
  strPath, strExPath : String;
  iSize: Integer;

Begin
  Result := False;
  strEXEName := ExtractFileName(strEXEName);
  For iPath := 0 To slPaths.Count - 1 Do
    Begin
      strPath := slPaths[iPath];
      SetLength(strExPath, MAX_PATH);
      iSize := ExpandEnvironmentStrings(PChar(strPath), PChar(strExPath), MAX_PATH);
```

```
      SetLength(strExPath, Pred(iSize));
      i := FindFirst(strExPath + strEXEName, faAnyFile, recSearch);
      Try
        If i = 0 Then
          Begin
            strEXEName := strExPath + strEXEName;
            Result := True;
            Break;
          End;
      Finally
        FindClose(recSearch);
      End;
    End;
End;
```

Procedure CheckPaths(Const slPaths : TStringList)

This procedure checks the paths in the given string list and deletes empty paths and ensures the rest have a trailing backslash.

```
Procedure CheckPaths(Const slPaths : TStringList);

Var
  iPath: Integer;
  iLength: Integer;

Begin
  For iPath := slPaths.Count - 1 DownTo 0 Do
    Begin
      iLength := Length(slPaths[iPath]);
      If iLength = 0 Then
        slPaths.Delete(iPath)
      Else
        If slPaths[iPath][iLength] <> '\' Then
          slPaths[iPath] := slPaths[iPath] + '\';
    End;
End;
```

Procedure TfrmJEDIVCSToGit.btnGetRevisionsClick(Sender: TObject)

This is an on click event handler for the Get Revisions button. This method starts the process of extracting files from JEDI VCS to put into the Git repository.

```
Procedure TfrmJEDIVCSToGit.btnGetRevisionsClick(Sender: TObject);

Const
```

```
  strGitStatus = 'status';

Begin
  CheckGitRepoPath;
  CheckThereIsAnExistingGitRepo;
  DBGrid.ReadOnly := True;
  BlobsGrid.ReadOnly := True;
  Try
    FStartTime := GetTickCount64;
    RevisionsDataSource.DataSet.Last;
    FItemCount := RevisionsDataSource.DataSet.RecordCount;
    RevisionsDataSource.DataSet.First;
    ProcessRevisions;
  Finally
    DBGrid.ReadOnly := False;
    BlobsGrid.ReadOnly := False;
  End;
End;
```

Procedure CheckFileNamesForRename(Const strSubDir, strFileToExtract : String)

This method checks the file to be extracted and whether it needs to rename an existing file. Renames the file if its name has changed and updates the output filename. Note: this is a local method to `ProcessBlogs()`.

```
Procedure CheckFileNamesForRename(Const strSubDir, strFileToExtract : String);

ResourceString
  strFileNeedsRenaming = 'The file "%s" needs renaming to "%s"!';

Const
  strModuleName = 'Module Name';
  strExtension = 'Extension';
  strMoveParams = 'mv -v "%s" "%s%s"';

Var
  strOldFileName: String;
  strRepoFileName: String;
  strActualPathAndFile: String;

Begin
  strRepoFileName := RevisionsDataSource.DataSet.FieldByName(strModuleName).AsString +
'.' + BlobsDataSource.DataSet.FieldByName(strExtension).AsString;
  strOldFileName := FFileNames.Values[strRepoFileName];
  If strOldFileName <> '' Then
```

```
    If CompareText(strOldFileName, strFileToExtract) <> 0 Then
      Begin
        strActualPathAndFile := strSubDir + strOldFileName;
        strActualPathAndFile := GetActualPathAndFileCase(strActualPathAndFile);
        ExecuteGit(Format(strMoveParams, [strActualPathAndFile,
ExtractFilePath(strActualPathAndFile), strFileToExtract]));
        If chkStatus.Checked Then
          ExecuteGit(strGitStatus);
      End;
  FFileNames.Values[strRepoFilename] := strFileToExtract;
End;
```

Function ProcessBlobs(Const strZipFileName : String) : Integer

This method processes the blobs associated with a revision and adds them to the Git repository. Note: this is a local method to the `btnGetRevisionsClick()` event handler.

```
Function ProcessBlobs(Const strZipFileName : String) : Integer;

ResourceString
  strExtracting = 'Extracting: %s';

Const
  strFileData = 'FileData';
  strAddParams = 'add -v "%s"';
  strPath = 'path';
  strModuleName = 'Module Name';

Var
  Z: TZipFile;
  iFile: Integer;
  strSubDir: String;
  boolAbort: Boolean;
  RepoData : TJVTGRepoData;
  strActualFileCase: String;

Begin
  Result := 0;
  BlobsDataSource.DataSet.First;
  While Not BlobsDataSource.DataSet.Eof Do
    Begin
      (BlobsDataSource.DataSet.FieldByName(strFileData) As
TBlobField).SaveToFile(strZipFileName);
      Z := TZipFile.Create;
      Try
```

```
        Z.Open(strZipFileName, zmRead);
        For iFile := 0 To Z.FileCount - 1 Do
          Begin
            RepoData.Create(FOldGitRepoPath, FNewGitRepoPath,
RevisionsDataSource.Dataset.FieldByName(strPath).AsString,
RevisionsDataSource.Dataset.FieldByName(strModuleName).AsString);
            If TfrmExtractRelPath.Execute(FRelativePaths, RepoData, strSubDir) Then
              Begin
                CheckFileNamesForRename(strSubDir, Z.FileName[iFile]);
                Z.Extract(Z.FileName[iFile], FNewGitRepoPath + strSubDir);
                ProcessMsgevent(Format(strExtracting, [FNewGitRepoPath + strSubDir +
Z.FileName[iFile]]), boolAbort);
                strActualFileCase := strSubDir + Z.FileName[iFile];
                strActualFileCase := GetActualPathAndFileCase(strActualFileCase);
                ExecuteGit(Format(strAddParams, [strActualFileCase]));
                Inc(Result);
                If chkStatus.Checked Then
                ExecuteGit(strGitStatus);
              End;
          End;
        Z.Close;
      Finally
        Z.Free;
      End;
    BlobsDataSource.DataSet.Next;
  End;
End;
```

Procedure ProcessRevisions

This method iterates through the revision records processing the blobs associated with each revision extracting the files, adding them and committing them. Note: this is a local method to the `btnGetRevisionsClick()` event handler.

```
Procedure ProcessRevisions;

Const
  strBlobZip = 'Blob.zip';
  strComment_i = 'comment_i';
  strTSTAMP = 'TSTAMP';

Var
  strZipFileName: String;

Begin
```

```
  strZipFileName := FNewGitRepoPath + strBlobZip;
  While Not RevisionsDataSource.DataSet.Eof Do
    Begin
      ProcessBlobs(strZipFileName);
      CommitToGit(RevisionsDataSource.DataSet.FieldByName(strComment_i).AsString,
RevisionsDataSource.DataSet.FieldByName(strTSTAMP).AsDateTime);
      If chkStatus.Checked Then
        ExecuteGit(strGitStatus);
      Inc(FItem);
      UpdateStatus;
      RevisionsDataSource.DataSet.Next;
    End;
End;
```

Procedure TfrmJEDIVCSToGit.CheckGitRepoPath

This method checks the Git Repository Path to ensure its a valid directory. An exception is raised if the path does not exist or is empty.

```
Procedure TfrmJEDIVCSToGit.CheckGitRepoPath;

ResourceString
  strGitRepositoryPathDoesNotExist = 'The Git Repository path "%s" does not exist!';

Begin
  If (Length(edtNewGitRepoPath.Text) = 0) Or (Not
DirectoryExists(edtNewGitRepoPath.Text)) Then
    Raise Exception.CreateFmt(strGitRepositoryPathDoesNotExist,
[edtNewGitRepoPath.Text]);
  FNewGitRepoPath := edtNewGitRepoPath.Text;
  If FNewGitRepoPath[Length(FNewGitRepoPath)] <> '\' Then
    FNewGitRepoPath := FNewGitRepoPath + '\';
  FOldGitRepoPath := edtOldGitRepoPath.Text;
  If FOldGitRepoPath[Length(FOldGitRepoPath)] <> '\' Then
    FOldGitRepoPath := FOldGitRepoPath + '\';
End;
```

Procedure TfrmJEDIVCSToGit.CheckThereIsAnExistingGitRepo

This method checks that there is an existing Git repository in the repository path. An exception is raised is there is not Git repository.

```
Procedure TfrmJEDIVCSToGit.CheckThereIsAnExistingGitRepo;

ResourceString
  strGitRepositoryDoesNotExists = 'A GIT repository does NOT exists in "%s"!';
```

```
Const
  strGitDir = '.git';

Begin
  If Not DirectoryExists(FNewGitRepoPath + strGitDir) Then
    Raise Exception.CreateFmt(strGitRepositoryDoesNotExists, [FNewGitRepoPath]);
End;
```

Procedure TfrmJEDIVCSToGit.CommitToGit(Const strComment: String; Const dtCommitDateTime: TDateTime)

This method commits the current staged files to Git using the given comment and time and date stamp.

```
Procedure TfrmJEDIVCSToGit.CommitToGit(Const strComment: String; Const
dtCommitDateTime: TDateTime);

Const
  strCommitDate = 'commit -v --date "%s" -m "%s"';
  strDateFmt = 'dd/mmm/yyyy HH:nn:ss';

Var
  strCleanComment : String;

Begin
  strCleanComment := StringReplace(strComment, '"', '''', [rfReplaceAll]);
  strCleanComment := StringReplace(strCleanComment, #13, '', [rfReplaceAll]);
  strCleanComment := StringReplace(strCleanComment, #10, '\n', [rfReplaceAll]);
  ExecuteGit(Format(strCommitDate, [FormatDateTime(strDateFmt, dtCommitDateTime),
strCleanComment]));
End;
```

Function TfrmJEDIVCSToGit.DGHCreateProcess(Var Process : TProcessInfo; Const ProcessMsgHandler : TProcessMsgHandler; Const IdleHandler : TIdleHandler) : Integer

This function creates a process with message handlers which must be implemented by the passed interface in order for the calling process to get messages from the process console and handle idle and abort. This method is used to run all the Git console commands.

```
Function TfrmJEDIVCSToGit.DGHCreateProcess(Var Process : TProcessInfo; Const
ProcessMsgHandler : TProcessMsgHandler; Const IdleHandler : TIdleHandler) : Integer;

Type
  EDGHCreateProcessException = Exception;

Const
```

```
  iPipeSize = 4096;

Var
  boolAbort: Boolean;

Var
  hRead, hWrite : THandle;
  SecurityAttrib : TSecurityAttributes;
  StartupInfo : TStartupInfo;

Begin
  Result := 0;
  boolAbort := False;
  ConfigSecurityAttrib(SecurityAttrib);
  Win32Check(CreatePipe(hRead, hWrite, @SecurityAttrib, iPipeSize));
  Try
    If Process.boolEnabled Then
      Try
        CheckProcess;
        ConfigStartUp(StartupInfo, hWrite);
        RunProcess(SecurityAttrib, StartupInfo, hRead);
      Except
        On E : EDGHCreateProcessException Do
          If Assigned(ProcessMsgHandler) Then
            Begin
              ProcessMsgHandler(E.Message, boolAbort);
              Inc(Result);
            End;
      End;
  Finally
    Win32Check(CloseHandle(hWrite));
    Win32Check(CloseHandle(hRead));
  End;
End;
```

Procedure CheckProcess

This method checks that the process directory and executable exists. An exception is raises if either the process directory or executable are not valid. Note: this is a local method to `DGHCreateProcess()`.

```
Procedure CheckProcess;

Begin
  If Not DirectoryExists(Process.strDir) Then
    Raise EDGHCreateProcessException.CreateFmt(strDirectoryNotFound, [Process.strDir]);
```

```
  If Not FileExists(Process.strEXE) Then
    If Not DGHFindOnPath(Process.strEXE, '') Then
      Raise EDGHCreateProcessException.CreateFmt(strEXENotFound, [Process.strEXE]);
End;
```

Procedure ConfigSecurityAttrib(Var SecurityAttrib : TSecurityAttributes)

This procedure configures the security attributes for the progress to be created. Note: this is a local method to `DGHCreateProcess()`.

```
Procedure ConfigSecurityAttrib(Var SecurityAttrib : TSecurityAttributes);

Begin
  FillChar(SecurityAttrib, SizeOf(SecurityAttrib), 0);
  SecurityAttrib.nLength := SizeOf(SecurityAttrib);
  SecurityAttrib.bInheritHandle := True;
  SecurityAttrib.lpSecurityDescriptor := Nil;
End;
```

Procedure ConfigStartUp(Var StartupInfo : TStartupInfo; Const hWrite : THandle)

This procedure configures the startup information for the new process to be created. Note: this is a local method to `DGHCreateProcess()`.

```
Procedure ConfigStartUp(Var StartupInfo : TStartupInfo; Const hWrite : THandle);

Begin
  FillChar(StartupInfo, SizeOf(TStartupInfo), 0);
  StartupInfo.cb := SizeOf(TStartupInfo);
  StartupInfo.cb := SizeOf(StartupInfo);
  StartupInfo.dwFlags := STARTF_USESHOWWINDOW or STARTF_USESTDHANDLES;
  StartupInfo.wShowWindow := SW_HIDE;
  StartupInfo.hStdOutput := hWrite;
  StartupInfo.hStdError := hWrite;
End;
```

Procedure ProcessOutput(Const slLines : TStringList; Const hRead : THandle; Const Purge : Boolean = False)

This prcoedure is called periodically by the process handler in order to retreive console output from the running process. The method outputs everything from the console (via an anonymous pipe) but the last line as this may not be a complete line of information from the console (except if `boolPurge` is `true`). Note: this is a local method to `DGHCreateProcess()`.

```
Procedure ProcessOutput(Const slLines : TStringList; Const hRead : THandle; Const Purge
: Boolean = False);

Var
```

```
  iTotalBytesInPipe : Cardinal;
  iBytesRead : Cardinal;
  strOutput : AnsiString;

Begin
  If Assigned(Idlehandler) Then
    IdleHandler;
  If boolAbort Then
    Begin
      If Assigned(ProcessMsgHandler) Then
        ProcessMsgHandler(strUserAbort, boolAbort);
      Exit;
    End;
  Win32Check(PeekNamedPipe(hRead, Nil, 0, Nil, @iTotalBytesInPipe, Nil));
  If iTotalBytesInPipe > 0 Then
    Begin
      SetLength(strOutput, iTotalBytesInPipe);
      ReadFile(hRead, strOutput[1], iTotalBytesInPipe, iBytesRead, Nil);
      SetLength(strOutput, iBytesRead);
      slLines.Append(StringReplace(UTF8ToString(strOutput), #10, #13#10,
[rfReplaceAll]));
    End;
  // Use a string list to output each line except the last as it may not
  // be complete yet.
  If Assigned(ProcessMsgHandler) Then
    While slLines.Count > 1 - Integer(Purge) Do
      Begin
        ProcessMsgHandler(slLines[0], boolAbort);
        slLines.Delete(0);
      End;
End;
```

Procedure RunProcess(Const SecurityAttrib : TSecurityAttributes; Const StartupInfo : TStartupInfo; Const hRead : THandle)

This procedure runs the process, collecting information from the console output and feeding it back into the output memo. Note: this is a local method to `DGHCreateProcess()`.

```
Procedure RunProcess(Const SecurityAttrib : TSecurityAttributes; Const StartupInfo :
TStartupInfo; Const hRead : THandle);

Const
  iWaitIntervalInMS = 50;

Var
```

```
  ProcessInfo : TProcessInformation;
  slLines : TStringList;
  iExitCode : Cardinal;

Begin
  Win32Check(CreateProcess(PChar(Process.strEXE),
  PChar('"' + Process.strEXE + '" ' + Process.strParams), @SecurityAttrib,
  Nil, True, CREATE_NEW_CONSOLE, Nil, PChar(Process.strDir), StartupInfo,
ProcessInfo));
  Try
    slLines := TStringList.Create;
    Try
      While WaitforSingleObject(ProcessInfo.hProcess, iWaitIntervalInMS) = WAIT_TIMEOUT
Do
        Begin
          ProcessOutput(slLines, hRead);
          If boolAbort Then
            Begin
              TerminateProcess(ProcessInfo.hProcess, 0);
              Break;
            End;
        End;
      ProcessOutput(slLines, hRead, True);
    Finally
      slLines.Free;
    End;
    If GetExitCodeProcess(ProcessInfo.hProcess, iExitCode) Then
    Inc(Result, iExitCode)
  Finally
    Win32Check(CloseHandle(ProcessInfo.hThread));
    Win32Check(CloseHandle(ProcessInfo.hProcess));
  End;
End;
```

Procedure TfrmJEDIVCSToGit.edtProjectNamePatternExit(Sender: TObject)

This method updates the `ProjectNamePattern` macro in the revision query while maintaining the DBGrids column widths.

```
Procedure TfrmJEDIVCSToGit.edtProjectNamePatternExit(Sender: TObject);

Const
  strProjectNamePatternMacro = 'ProjectNamePattern';

Var
```

```
  M: TFDMacro;
  iColumn: Integer;
  aiColumnWidths : TArray&lt;Integer&gt;;;

Begin
  If FDConnection.Connected Then
    Begin
      SetLength(aiColumnWidths, DBGrid.Columns.Count);
      For iColumn := 0 To DBGrid.Columns.Count - 1 Do
        aiColumnWidths[iColumn] := DBGrid.Columns[iColumn].Width;
      M := RevisionsFDQuery.MacroByName(strProjectNamePatternMacro);
      M.Value := edtProjectNamePattern.Text;
      RevisionsFDQuery.Active := True;
      For iColumn := 0 To DBGrid.Columns.Count - 1 Do
        DBGrid.Columns[iColumn].Width := aiColumnWidths[iColumn];
    End;
End;
```

Procedure TfrmJEDIVCSToGit.ExecuteGit(Const strCmdParams: String)

This method executes GIT and captures any errors and prompts for an action.

```
Procedure TfrmJEDIVCSToGit.ExecuteGit(Const strCmdParams: String);

ResourceString
  strMsg = 'The last GIT command (%s) failed:'#13#10'%s';

Var
  boolAbort : Boolean;
  iResult: Integer;

Begin
  FGitPI.strDir := FNewGitRepoPath;
  FGitPI.strParams := strCmdParams;
  ProcessMsgevent(Format('%s%s %s', [FGitPI.strDir, ExtractFileName(FGitPI.strEXE),
FGitPI.strParams]), boolAbort, mtTitle);
  FLastMessage := '';
  iResult := DGHCreateProcess(FGitPI, ProcessMsgEvent, IdleEvent);
  If iResult <> 0 Then
    Case TfrmGITError.Execute(Format(strMsg, [strCmdParams, FLastMessage])) Of
      mrAbort: Abort;
    End;
  ProcessMsgevent(#13#10, boolAbort);
End;
```

Procedure TfrmJEDIVCSToGit.FormCreate(Sender: TObject)

This is an `OnFormCreate` Event Handler for the `TfrmJEDVICSToGit` class. This event handler loads the applications settings and creates a string list for filenames. Note: the `FFileNames` string list is used to store a relative path for each filename so that you are not prompted for each directory for the same file committed repeatedly.

```
Procedure TfrmJEDIVCSToGit.FormCreate(Sender: TObject);

ResourceString
  strPleaseSpecifyFireDACINIFileAsFirstParameter = 'Please specify a FireDAC INI file
as the first ' +
    'parameter!';
  strCouldNotLoadINIFile = 'Could not load the INI file "%s"';
  strJEDIVCSToGitBuild = 'JEDI VCS to Git %d.%d%s (Build %d.%d.%d.%d): ';

Const
  strBugFix = ' abcedfghijklmnopqrstuvwxyz';
  strGITExe = 'GIT.exe';

Var
  BuildInfo: TJVTGBuildInfo;

Begin
  GetBuildInfo(BuildInfo);
  Caption := Format(strJEDIVCSToGitBuild, [
    BuildInfo.FMajor,
    BuildInfo.FMinor,
    strBugFix[BuildInfo.FRelease + 1],
    BuildInfo.FMajor,
    BuildInfo.FMinor,
    BuildInfo.FRelease,
    BuildInfo.FBuild
  ]);
  FFilenames := TStringList.Create;
  FFileNames.Duplicates := dupIgnore;
  FItemCount := 0;
  FItem := 0;
  FGitPI.boolEnabled := True;
  FGitPI.strEXE := strGITExe;
  If (ParamCount > 0) And FileExists(ParamStr(1)) Then
    Begin
      FDConnection.Params.LoadFromFile(ParamStr(1));
      FDConnection.Connected := True;
      RevisionsFDQuery.Active := True;
```

```
      BlobsFDQuery.Active := True;
    End Else
      If ParamCount = 0 Then
        ShowMessage(strPleaseSpecifyFireDACINIFileAsFirstParameter)
      Else
        ShowMessage(Format(strCouldNotLoadINIFile, [ParamStr(1)]));
  FRelativePaths := TStringList.Create;
  FRelativePaths.Duplicates := dupIgnore;
  LoadSettings;
End;
```

Function TfrmJEDIVCSToGit.GetActualPathAndFileCase(Const strRelPathFile: String) : String

This method searches for the actual case for the file path and filename so that GIT ADD does not fail to add the file. **Note: even if you have set Git to be case-insentitive, this is a situation where it does not appear to work**.

```
Function TfrmJEDIVCSToGit.GetActualPathAndFileCase(Const strRelPathFile: String) :
String;

Var
  sl : TStringList;
  i: Integer;
  strCurrentPath : String;
  recSearch: TSearchRec;
  iResult: Integer;

Begin
  Result := '';
  sl := TStringList.Create;
  Try
    sl.Text := StringReplace(strRelPathFile, '\', #13#10, [rfReplaceAll]);
    strCurrentPath := FNewGitRepoPath;
    For i := 0 To sl.Count - 1 Do
      Begin
        If (Result <> '') And (Result[Length(Result)] <> '\') Then
          Result := Result + '\';
        If (strCurrentPath <> '') And (strCurrentPath[Length(strCurrentPath)] <> '\')
Then
          strCurrentPath := strCurrentPath + '\';
        iResult := FindFirst(strCurrentPath + sl[i], faAnyFile, recSearch);
        Try
          If iResult = 0 Then
            Begin
              Result := Result + recSearch.Name;
```

```
              strCurrentPath := strCurrentPath + recSearch.Name;
            End;
        Finally
          FindClose(recSearch);
        End;
      End;
  Finally
    sl.Free;
  End;
End;
```

Procedure TfrmJEDIVCSToGit.IdleEvent

This is an on idle event handler for the command line processes to ensure the application updates its interface.

```
Procedure TfrmJEDIVCSToGit.IdleEvent;

Begin
  Application.ProcessMessages;
End;
```

Procedure TfrmJEDIVCSToGit.ProcessMsgevent(Const strMsg: String; Var boolAbort: Boolean

This method processes a message from a command line and outputs the information to the output log.

```
Procedure TfrmJEDIVCSToGit.ProcessMsgevent(Const strMsg: String; Var boolAbort:
Boolean;
  Const MsgType : TMsgType = mtInformation);

Var
  sl : TStringList;
  iLine: Integer;

Begin
  sl := TStringList.Create;
  Try
    sl.Text := strMsg;
    For iLine := 0 To sl.Count - 1 Do
      Begin
        lbxGitOutput.Items.AddObject(sl[iLine], TObject(MsgType));
        lbxGitOutput.ItemIndex := Pred(lbxGitOutput.Items.Count);
        If sl[iLine] <> '' Then
          Begin
            If FLastMessage <> '' Then
              FLastMessage := FLastMessage + #13#10;
```

```
            FLastMessage := FLastMessage + sl[iLine];
          End;
      End;
  Finally
    sl.Free;
  End;
End;
```

I hope this help those of you who want to migrated your JEDI VCS repositories to Git.

regards

Dave.

Category: Delphi Git RAD Studio