**EZCodeGenerator**

**ABSTRACT**

*In the daily life, all the people around the world always go to the internet to find the information that they need. But in order to get that information they may need to spend many time for go to check one by one. Sometime they also cannot get all what they need, because of the limitation of time or other duty that they need to complete. Therefore we got an idea and decided to establish one desktop application call “EZCodeGenerator”.**EZCodeGenerator**is a Web Scraping software that interacted with websites in the same way as your web browser. But instead of displaying the data served by the website on screen, the Web Scraping software saves the required data from the web page to a local file or database. However we also generate a module (project) to run in our desktop application as well.*

*As the result, we got the smart application that is, useful, attractive, have the high performance, reducible code and reduce spending time that easy for us to scrap what we want in other websites.*

# **I. Introduction**

# **1.1. Overview**

Today, technology is moving faster from day to day. It plays a very important role for people’s daily life for almost every sector that make us more easily. Meanwhile, new technology also increased and have different features with the very nice and colorful interface.

That why, we overlook some feature such as Web Scraping that it is the technique of automating displayed data from webpages, so it instead of manually copying the data from websites, the Web Scraping software will perform the same task within a fraction of the time. Before we get that data with a short time, we need to spend much time to build it such as: Module, it should be a flexible, easy to use and not easily to crush when we want to update it. How can the developer handle it? So that why we need long time to find all the problems and solve it to get the best solution and best idea for develop it.

# **EZCodeGenerator’s Objective**

After we find out these problem, we had try our best to establish a desktop application for creating, compiling and running our module, and we named it to ‘EZCodeGenerator’. EZCodeGenerator’s vision are:

* Developer no need to type code manually, by just click to choose what they want to generate in their project automatically.
* Reducing development time with redundancy thing.
* Making it easy to implement and customize scrapping project.

# **Motivation**

Web Scraping is the technique of automating displayed data from webpages, so it instead of manually copying the data from websites, the Web Scraping software will perform the same task within a fraction of the time. The reason that push our team to develop EZCodeGenerator’s desktop application because scrapping technology is still has a limitation to the developers, especially in our country (Cambodia). As well, we strongly believed that our application will be useful for all the developer that interested in this field. Meanwhile, scrapping technology is very popular in European country such as Korea, Japan…etc.

# **1.2. Background**

So far, to scrap data or make an application to handle process for a webpage, developers need to write different project, different code to create scrapping modules but what we have seen is mostly the same format. Developers usually copy their code from their old project and then customize their code by typing manually with the code. They need to look seriously for what should be changed, what should be deleted or whatever with the code which easily causes errors occur. For developers with fully experience with scrapping technology and project structure look it doesn’t matter, but we have a vision that it shouldn’t waste the time to do the same task again and again. We have thought if we all do almost the same things, why it shouldn’t be automatic? Furthermore, to train new developers to develop the scrapping project may take more time, what if we have an automatic project generating? As we know, using interface is easier than coding, so if the result and performance would be the same, we may prefer interface rather than coding.

Thus, we have an innovation to create a project with ability to generate scrapping project in order to solve the problem as we have mentioned above.

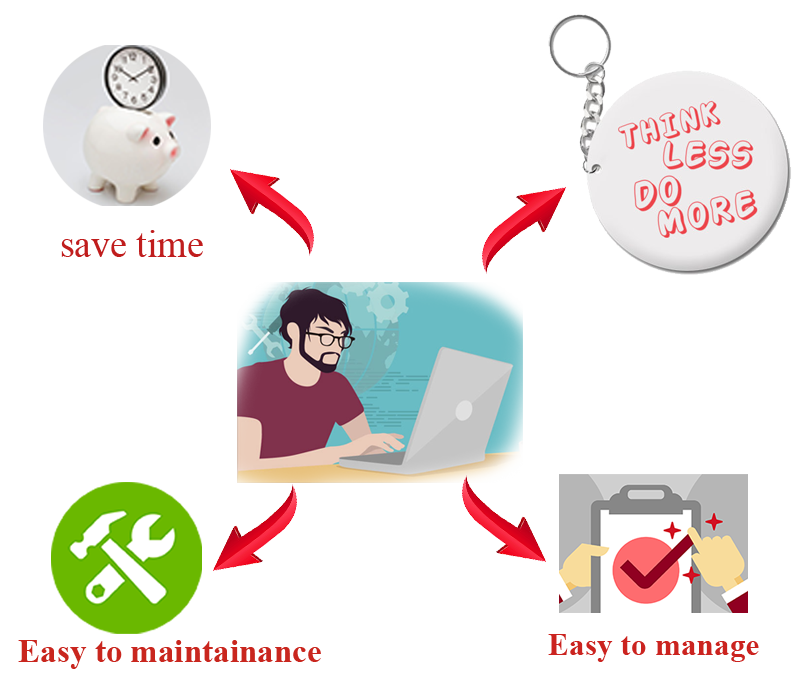
# **II. Requirement**

In the step, we start to study about the process flow in code generating, defining the module structure, and collecting important information data for designing our project software concept which we including two aspects such as:

# **2.1 Business Modeling**

According to what module scrapping structures, webpage structure, Business Modeling is the conceptual structure supporting the viability of a business, including its purpose, its goals and its ongoing plans for achieving them.

EZCodeGenerator, business model, generates scrapping project to build modules which is used to scrap or do the processing for each of its webpage such as in type of general information, e-pay, services, local bank, and international bank. The purpose of scrapping these sites is to reduce time on checking user’s information or make it easily to the user while some sites are really complicated and difficult to login to their account when they want to know their balance, transfer, or transaction. This is the reason that developers also hard to develop their scrapping project and sometimes they waste time to write the almost the same code. So they may need a new way to do these easier.



**Figure 1: How developer get benefit from EZCodeGenerator**

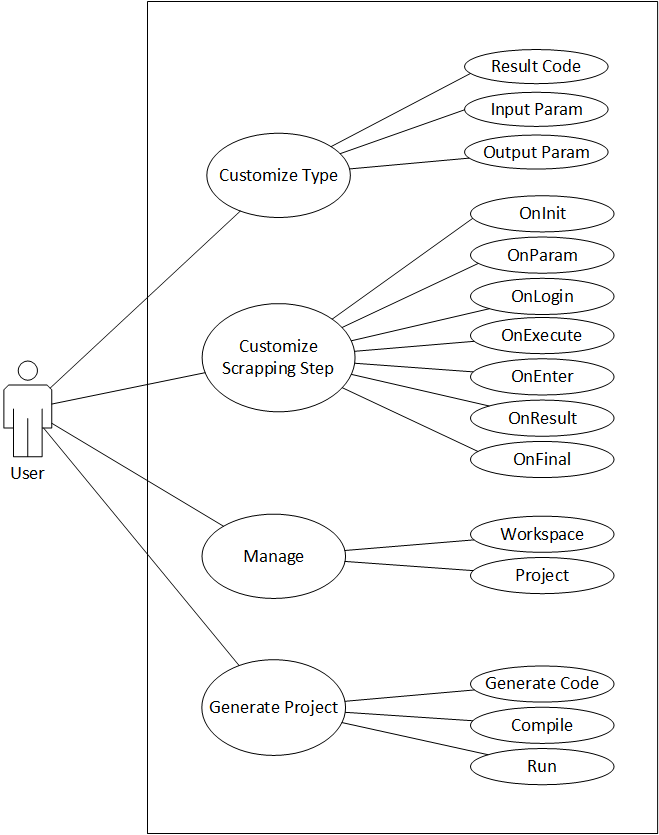
# **2.2 System Requirement**

# **2.2.1 Problem of Statement**

As we already described in business modeling phase, most scraping modules are come from the e-paid service or bank websites. But, for developing these modules, there are some points which have been considered by developers:

* Most of scrapping module is almost the same format and structures
* They hard to look back their code again when the error occurs in the modules
* The developed modules are not in the standard structure, individual developer will have their own coding style even the structure have set.
* The developers still need to take more time to check their code again and again for errors.

# **2.2.2 Use case**



**Figure 2: EZCodeGenerator Use Case**

The use case diagram in Figure 2 can be detailed into understandable terms as given below:

In EZCodeGenerator application, user can have 4 actions include customize type, customize scrapping step, manage project and generate project. For customize type, user can define, delete and update the type in project such as output parameters, input parameters and result code. Second, for customizing scrapping step, user can customize the 7 step as seen in the pictures which user don’t need to type code manually, they only need to click what they need, then they can get the code they want. Furthermore, they can manage their project easily in a specific workspace or they can change or move workspace as they want. Last, they can generate the project in standard structure, compile and run their code easily.

# **III. Analysis**

# **3.1 Scope**

Based on a **Guide to the Project Management Body of Knowledge**, scope is the work that needs to be accomplished to deliver a product, service, or result with the specified features and functions.

To accomplish our project on time without problem, our scopes are:

* User friendly UI and codeless development
* Project organization and maintenance
* Choosing scrapping services to use
* Choosing parameters and error checking
* Providing full scrapping step code generation
* Standard code organization with comments
* Real time source code preview with syntax highlighter
* User can add more code for advance scrapping project
* Project compilation and running

# **3.2 Risk Management**

Every project always has its own risk. Also, our project has it too. During our development EZCodeGenerator, risk can be included as following.

|  |  |  |
| --- | --- | --- |
| Type of Risk | Description | Solution |
| Unexpected accident risk | Unexpected weather or something emergency like traffic accident | Standby or extra hour work |
| Staff risk | Staff knowledge | Research more and ask others to help |
| Scope risk | Wrong time estimation | Use work breakdown structure |
| Technical risk | Unexpected scope project expansion | priority on scope |
| Difficult with file structure manipulation | define specific file structure |

**Figure 3: Risk Management**

# **IV. Planning**

After we have finished on Analysis phase, our next phase will be planning. In planning the major activities is mostly focus on schedule for keeping tracks on the processes and the estimation related to the project are done. Hence, in the phase, our team will raise the most popular tool that most of project manager uses.

# **4.1 Work Breakdown Structure (WBS)**

A work breakdown structure is a key project deliverable that organizes the team’s work into manageable sections. Also, the Work Breakdown Structure (WBS) provides a structural view into the project. For this reason, our team will depict the WBS that we think it requires for completed project as below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Name | Start Date | End Date | Assigned To | Duration |
| 1. Requirement | 07-Dec-16 | 13-Dec-16 | All | 3 |
| 1.1. Business Modeling | 07-Dec-16 | 08-Dec-16 | All Members | 2 |
| 1.2.System Requirement Modeling | 08-Dec-16 | 09-Dec-16 | All Members | **3** |
| 1.2.1. Problem of Statement | 09-Dec-16 | 09-Dec-16 | All Members | **2** |
| 1.2.1. System Requirement | 09-Dec-16 | 13-Dec-16 | All Members | **2** |
| 2. Analysis | 09-Dec-16 | 16-Dec-16 |  | **2** |
| 2.1. Define Scope and Purpose | 09-Dec-16 | 14-Dec-16 | All Members | **2** |
| 2.2. Define Risk Management | 15-Dec-16 | 16-Dec-16 | All Members | **2** |
| 3. Planning | 19-Dec-16 | 20-Dec-16 |  | **2** |
| 3.1. Work Breakdown Structure | 19-Dec-16 | 19-Dec-16 | All Members | **1** |
| 3.2. Time Management | 19-Dec-16 | 19-Dec-16 | All Members | **1** |
| 3.3. Risk Management | 19-Dec-16 | 20-Dec-16 | All Members | **2** |
| 4. Design | 19-Dec-16 | 23-Dec-16 |  | **5** |
| 4.1. Class Diagram | 19-Dec-16 | 19-Dec-16 | All Members | **1** |
| 4.2. Data Dictionary | 19-Dec-16 | 19-Dec-16 | All Members | **1** |
| 4.3. Database Design | 19-Dec-16 | 20-Dec-16 | All Members | **2** |
| 4.4. ERD | 20-Dec-16 | 21-Dec-16 | All Members | **2** |
| 4.5. Software Architecture | 20-Dec-16 | 21-Dec-16 | All Members | **2** |
| 4.6. Interface Design | 20-Dec-16 | 04-Dec-16 | All Members | **4** |
| 5. Implementation | 05-Jan-16 | 24-Jan-17 |  | **14** |
| 5.1. Workspace management | 05-Jan-16 | 10-Jan-16 | Rany | **4** |
| 5.2. Project, module creation and deletion | 05-Jan-16 | 10-Jan-16 | Somphors | **4** |
| 5.3. Project Explorer management and           And search function | 05-Jan-16 | 10-Jan-16 | Rany, SiekHai | **4** |
| 5.4. Project file tab open, close | 05-Jan-16 | 10-Jan-16 | Chhunly | **4** |
| 5.5. Hotkey and shortcut key | 05-Jan-16 | 10-Jan-16 | Somphors | **4** |
| 5.6. OnInit, OnParam, OnResult function | 11-Jan-16 | 24-Jan-16 | Rany | **10** |
| 5.7. OnLogin, OnEnter, OnExcecute,           OnFinal function | 11-Jan-16 | 24-Jan-16 | SiekHai | **10** |
| 5.8. Result code, Input param, and           Result Output management | 11-Jan-16 | 24-Jan-16 | Chhunly | **10** |
| 5.9. Module compiling and running | 11-Jan-16 | 24-Jan-16 | Somphors | **10** |
| 5.10. Open Project and close project | 11-Jan-16 | 24-Jan-16 | Somphors | **10** |
| 6. Testing and deploy | 25-Jan-17 | 07-Feb-17 | All Members | **10** |
| 7. Submit | 08-Feb-17 | 08-Feb-17 | HRD Center | **1** |
|  |  |  |  | Totally: 47 days |

**Figure 4: Work Breakdown Structure**

# **V. Design**

The requirement specification from all the three phases are studied, then in this phase, system, is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. Undoubtedly, in the phase, we will present our class diagram, database design, software architecture, and interface design

# **5.1 Customizing types**

In scrapping process module, we need three constants variables type such as parameter input, result output and result code. Parameter input is the constants value to store index value represent to data to post to the webpage. Parameter output is the constants value to store index value represent to data to be scrapped. Last one is result code which is used to tell whether the scrapping process is success or fail.

Input Parameter



Webpages

request

Response

Result Output

Result Code

Response

**Figure 5: How we use type in scrapping process**

Because different websites have different context and data, so developers usually need to add or delete type in the code. Day by day, the types may increase more and more which makes developers need to scroll their code and find the type they want to delete or update which feels waste the time and boring. As the fact, we have an innovation to control them with interface which would be an easier way.

# **5.2 Customizing scrapping steps**

There are 7 step of each scrapping module to scrap data from webpages in our scrapping project technology. The main point of scrapping process uses http request with socket. Thus, the 7 step in our scrapping project technology are the functions OnInit, OnParam, OnLogin, OnEnter, OnExecute, OnResult and finally OnFinal. With this feature, developers can customize everything easily without trying to look their code, but only click on interface.

Initialize request object

Check parameters before request

Send or Post data via http request using Indy Agent or other libraries.

Process to do after receive response

Do redirection or re-request to web page

Get result and send to a specific destination as needed

Release and free memory

**Figure 6: 7 steps of scrapping module**

# **5.3 Project management**

Project management is the feature with ability to manage many projects in one platform. It performs like an IDE such as eclipse. You can switch workspace, open projects in the workspace easily without using windows explorer to find them.

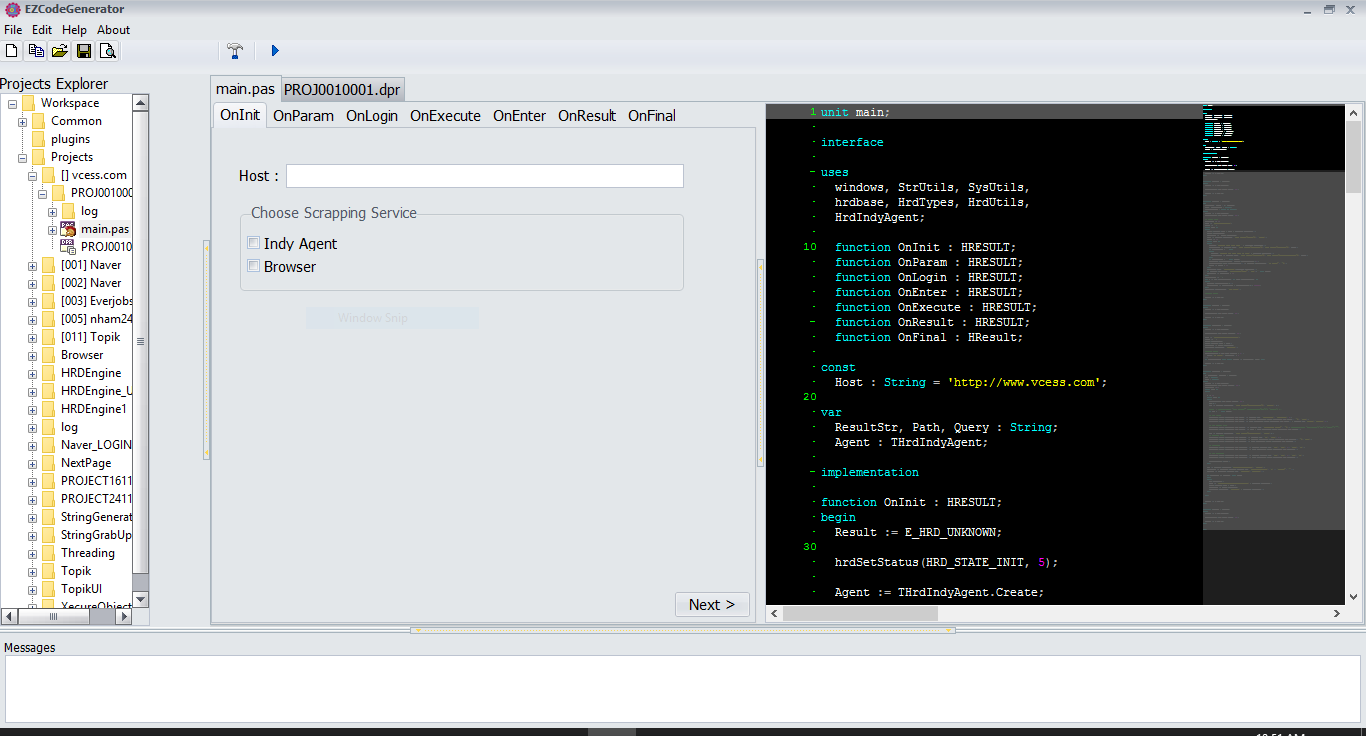
# **5.4 Project generation**

As we know that EZCodeGenerator provide customization to scrapping project, so it will simply able to generate code. But the scope is not only code generation because developer will need to get the code and reopen in different ide to compile and test their code. So, we provide the compiling tool and run to make sure the codes are working properly.

# **5.5 Interface Design**

After we have completed both planning and designing phase, let start with pre-drawing our application as your information, to develop our application we’ve used RAD Studio XE8 as our Integrated Development Environment (IDE).

Now, let’s take a look our application using RAD Studio XE8 as below:



The 7 steps customization

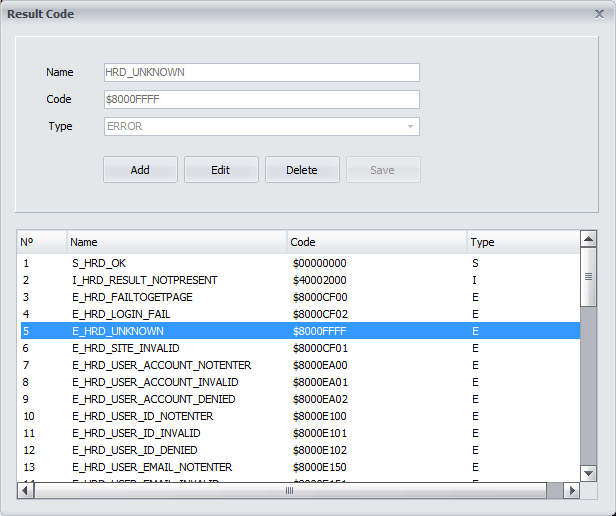
Project management

Code preview

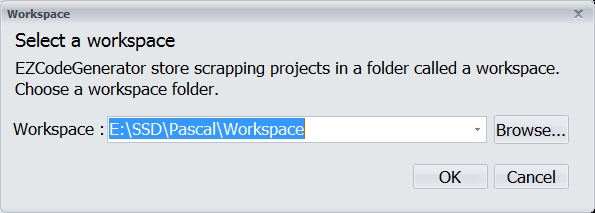
Compilation messages

Compiling and Running tool

**Figure 7: Main form in EZCodeGenerator**



**Figure 8: Customizing types (Result code management)**



**Figure 9: Workspace configuration**

# **VI. Implementation**

For project implementation, we use Object Pascal language using RAD Studio XE8., because of some features of the IDE has some limitation design we use other libraries to make the implementation easier such as TBCEditor, RAIZE components, ShellBrowser, JVCL, DevExpress VCL, WebBrowser, Indy Agent, and other useful components. We implemented interfaces such as UnitCodeGenerator (main form), UnitCompile (compiling process), UnitCreateModule, UnitCreateProject, UnitFindText, UnitInputParam, UnitResultCode, UnitResultOutput and finally UnitWorkspace. We also use INIFile and text file in order to store project configuration.

Let’s see some of our implementation as the codes below:

Workspace Configuration:

var

text : string;

IsDup : Boolean;

begin

//check duplicate before save to file

AssignFile(myFile, 'workspace.rany');

ReSet(myFile);

IsDup := False;

while not eof(myFile) do

begin

ReadLn(myFile, text);

if text = ComboBoxWorkspace.Text then

begin

IsDup := True;

break;

end;

end;

if IsDup <> true then

begin

Append(myFile);

Writeln(myFile, ComboBoxWorkspace.Text);

end;

//read the last selected workspace

ProjectSetting.WriteString('Workspace', 'WorkspacePath', ComboBoxWorkspace.Text);

EZCodeGenerator.TreeViewProjectExplorer.RootedAtFileSystemFolder := ProjectSetting.ReadString('Workspace', 'WorkspacePath', '');

CloseFile(myFile);

//clear BCeditor and hide tab

EZCodeGenerator.BCEditorCodePreview.Clear;

EZCodeGenerator.BCEditorProjCode.Clear;

EZCodeGenerator.RzPageControlMain.Visible := False;

Self.Close;

end;

UnitCompilation:

procedure TFormCompile.FormShow(Sender: TObject);

begin

//save all code in editor

EZCodeGenerator.ToolButtonSaveClick(Self);

//compiling

with TMyThead.Create do

FreeOnTerminate := true;

//clear and add new compile message

EZCodeGenerator.ListBoxMessage.Clear;

EZCodeGenerator.ListBoxMessage.Items.Add('Compiling project...');

//compile info

EditProjectPath.Text := EZCodeGenerator.projectfiletocompile.Caption;

EditLinkingPath.Text := ChangeFileExt(EditProjectPath.Text, '.exe');

EditTotalLines.Text := 'Total Lines: ' + IntToStr(EZCodeGenerator.BCEditorProjCode.Lines.Count);

end;

function TFormCompile.GetDosOutput(CommandLine: string; Work: string = 'C:\'): string;

var

SA: TSecurityAttributes;

SI: TStartupInfo;

PI: TProcessInformation;

StdOutPipeRead, StdOutPipeWrite: THandle;

WasOK: Boolean;

Buffer: array[0..255] of AnsiChar;

BytesRead: Cardinal;

WorkDir: string;

Handles: Boolean;

last3Line : String;

begin

Result := '';

with SA do begin

nLength := SizeOf(SA);

bInheritHandle := True;

lpSecurityDescriptor := nil;

end;

CreatePipe(StdOutPipeRead, StdOutPipeWrite, @SA, 0);

try

with SI do

begin

FillChar(SI, SizeOf(SI), 0);

cb := SizeOf(SI);

dwFlags := STARTF\_USESHOWWINDOW or STARTF\_USESTDHANDLES;

wShowWindow := SW\_HIDE;

hStdInput := GetStdHandle(STD\_INPUT\_HANDLE);

hStdOutput := StdOutPipeWrite;

hStdError := StdOutPipeWrite;

end;

WorkDir := Work;

Handles := CreateProcess(nil, PChar(WideString('cmd.exe /C rsvars.bat&&' + CommandLine)),

nil, nil, True, 0, nil,

PChar(WideString(WorkDir)), SI, PI);

CloseHandle(StdOutPipeWrite);

if Handles then

try

repeat

WasOK := ReadFile(StdOutPipeRead, Buffer, 255, BytesRead, nil);

if BytesRead > 0 then

begin

Buffer[BytesRead] := #0;

// Result := Result + Buffer;

EZCodeGenerator.ListBoxMessage.Items.Add(Buffer);

end;

until not WasOK or (BytesRead = 0);

WaitForSingleObject(PI.hProcess, INFINITE);

finally

CloseHandle(PI.hThread);

CloseHandle(PI.hProcess);

end;

finally

CloseHandle(StdOutPipeRead);

EditStatus.Text := 'Status: Done';

//compile info

last3Line := EZCodeGenerator.ListBoxMessage.Items[EZCodeGenerator.ListBoxMessage.Count-1] +

EZCodeGenerator.ListBoxMessage.Items[EZCodeGenerator.ListBoxMessage.Count-2] +

EZCodeGenerator.ListBoxMessage.Items[EZCodeGenerator.ListBoxMessage.Count-3];

if Pos('Warning(s)', last3Line)> 0 then

begin

EditWarning.Text := 'Warning: ' + Trim(MidStr(last3Line, Pos('Warning(s)', last3Line)-2, 2));

EditError.Text := 'Error: ' + Trim(MidStr(last3Line, Pos('Error(s)', last3Line)-2, 2));

end;

//running

if (EZCodeGenerator.runorcompile.Caption = 'Running') and ((EditError.Text = 'Error: 0') or (EditError.Text = 'Error:')) then

ShellExecute(Handle, 'open', Pchar(EditLinkingPath.Text), '', '', SW\_SHOWNORMAL);

//close form compile

if EditError.Text = 'Error: 0' then

self.Close;

end;

end;

Project Creation:

procedure TFormCreateProject.ButtonFinishClick(Sender: TObject);

begin

if Not DirectoryExists(Edit1.Text) then

begin

if CreateDir(Edit1.Text) then

begin

EZCodeGenerator.TreeViewProjectExplorer.SelectedFolder := Edit1.Text;

Self.Close;

FormCreateModule.ShowModal;

end

else

MessageDlg('Project create failed!', mtError, [mbOK], 0);

end

else

MessageDlg('This project already exited!', mtError, [mbOK], 0);

end;

Because some part of codes has the same process, so we also create our own libraries to use such as FileManip.pas, TypeConfigurer.pas. The library FileManip.pas is created to work with file, created TFileSearchReplace type in order to replace string in file because we’re working with files which need to replace codes.

unit FileManip;

interface

uses

System.Classes,

System.SysUtils;//, Vcl.Dialogs;

type

TFileSearchReplace = class(TObject)

private

FSourceFile: TFileStream;

FtmpFile: TFileStream;

FEncoding: TEncoding;

public

constructor Create(const AFileName: string);

destructor Destroy; override;

procedure Replace(const AFrom, ATo: string; ReplaceFlags: TReplaceFlags);

end;

implementation

uses

System.IOUtils,

System.StrUtils;

function Max(const A, B: Integer): Integer;

begin

if A > B then

Result := A

else

Result := B;

end;

{ TFileSearchReplace }

constructor TFileSearchReplace.Create(const AFileName: string);

begin

inherited Create;

FSourceFile := TFileStream.Create(AFileName, fmOpenReadWrite);

FtmpFile := TFileStream.Create(ChangeFileExt(AFileName, '.tmp'), fmCreate);

end;

destructor TFileSearchReplace.Destroy;

var

tmpFileName: string;

begin

if Assigned(FtmpFile) then

tmpFileName := FtmpFile.FileName;

FreeAndNil(FtmpFile);

FreeAndNil(FSourceFile);

TFile.Delete(tmpFileName);

inherited;

end;

procedure TFileSearchReplace.Replace(const AFrom, ATo: string;

ReplaceFlags: TReplaceFlags);

procedure CopyPreamble;

var

PreambleSize: Integer;

PreambleBuf: TBytes;

begin

// Copy Encoding preamble

SetLength(PreambleBuf, 100);

FSourceFile.Read(PreambleBuf, Length(PreambleBuf));

FSourceFile.Seek(0, soBeginning);

PreambleSize := TEncoding.GetBufferEncoding(PreambleBuf, FEncoding);

if PreambleSize <> 0 then

FtmpFile.CopyFrom(FSourceFile, PreambleSize);

end;

function GetLastIndex(const Str, SubStr: string): Integer;

var

i: Integer;

tmpSubStr, tmpStr: string;

begin

if not(rfIgnoreCase in ReplaceFlags) then

begin

i := Pos(SubStr, Str);

Result := i;

while i > 0 do

begin

i := PosEx(SubStr, Str, i + 1);

if i > 0 then

Result := i;

end;

if Result > 0 then

Inc(Result, Length(SubStr) - 1);

end

else

begin

tmpStr := UpperCase(Str);

tmpSubStr := UpperCase(SubStr);

i := Pos(tmpSubStr, tmpStr);

Result := i;

while i > 0 do

begin

i := PosEx(tmpSubStr, tmpStr, i + 1);

if i > 0 then

Result := i;

end;

if Result > 0 then

Inc(Result, Length(tmpSubStr) - 1);

end;

end;

var

SourceSize: int64;

procedure ParseBuffer(Buf: TBytes; var IsReplaced: Boolean);

var

i: Integer;

ReadedBufLen: Integer;

BufStr: string;

DestBytes: TBytes;

LastIndex: Integer;

begin

if IsReplaced and (not(rfReplaceAll in ReplaceFlags)) then

begin

FtmpFile.Write(Buf, Length(Buf));

Exit;

end;

// 1. Get chars from buffer

ReadedBufLen := 0;

for i := Length(Buf) downto 0 do

if FEncoding.GetCharCount(Buf, 0, i) <> 0 then

begin

ReadedBufLen := i;

Break;

end;

if ReadedBufLen = 0 then

raise EEncodingError.Create('Cant convert bytes to str');

FSourceFile.Seek(ReadedBufLen - Length(Buf), soCurrent);

BufStr := FEncoding.GetString(Buf, 0, ReadedBufLen);

if rfIgnoreCase in ReplaceFlags then

IsReplaced := ContainsText(BufStr, AFrom)

else

IsReplaced := ContainsStr(BufStr, AFrom);

if IsReplaced then

begin

LastIndex := GetLastIndex(BufStr, AFrom);

LastIndex := Max(LastIndex, Length(BufStr) - Length(AFrom) + 1);

end

else

LastIndex := Length(BufStr);

SetLength(BufStr, LastIndex);

FSourceFile.Seek(FEncoding.GetByteCount(BufStr) - ReadedBufLen, soCurrent);

BufStr := StringReplace(BufStr, AFrom, ATo, ReplaceFlags);

DestBytes := FEncoding.GetBytes(BufStr);

FtmpFile.Write(DestBytes, Length(DestBytes));

end;

var

Buf: TBytes;

BufLen: Integer;

bReplaced: Boolean;

begin

FSourceFile.Seek(0, soBeginning);

FtmpFile.Size := 0;

CopyPreamble;

SourceSize := FSourceFile.Size;

BufLen := Max(FEncoding.GetByteCount(AFrom) \* 5, 2048);

BufLen := Max(FEncoding.GetByteCount(ATo) \* 5, BufLen);

SetLength(Buf, BufLen);

bReplaced := False;

while FSourceFile.Position < SourceSize do

begin

BufLen := FSourceFile.Read(Buf, Length(Buf));

SetLength(Buf, BufLen);

ParseBuffer(Buf, bReplaced);

end;

FSourceFile.Size := 0;

FSourceFile.CopyFrom(FtmpFile, 0);

end;

end.

For TypeConfigurer.pas is used to manipulate with constants types in project such as input parameter, result output, and result code which is used frequently.

unit TypeConfigurer;

interface

uses

System.SysUtils, hrdutils, Strutils, Types, Filemanip, Dateutils, Vcl.Dialogs;

function getInputParam : TStringDynArray;

function getOutputParam : TStringDynArray;

function getResultCode : TStringDynArray;

function saveInputParam (params : TStringDynArray) : boolean;

function saveOutputParam (params : TStringDynArray) : boolean;

function saveResultCode (resultcode, code : string; resulttype : integer) : boolean;

function updateInputParam(oldone, newone : TStringDynArray) : boolean;

function updateOutputParam(oldone, newone : TStringDynArray) : boolean;

function updateResultCode(oldone, newone : string) : boolean;

function deleteType(typearray : TStringDynArray) : boolean;

function isExistType(typename : string) : boolean;

function isValidType(typename:string):boolean;

function isExistCode(code: string) : boolean;

function isExistResultPair(resultpair : string) : boolean;

function getTypeLocation : string;

function getAllStringFromType : string;

function getTypeWholeLine(typename:string) : string;

File storage configuration:

[Workspace]

WorkspacePath=E:\SSD\Pascal\Workspace

# **VII. Conclusion**

# **7.1 Summary**

In this EZCodeGenerator project, there are so many benefits that provides to the developer of scrapping project which will surely improve their working progress with less coding and redundant jobs. However, this project will either have its own weakness and strength.

# **7.1.1 Project Strength**

The key strength of EZCodeGenerator project is the goal of the project. EZCodeGenerator’s goal is to help scrapping developers to speed up their working process by redesign a codeless working environment while keeping anything customizable in developer’s choice. EZCodeGenerator provide the ability to create scrapping project as well as generating the module as well. Furthermore, creating the scrapping module in EZCodeGenerator works like it has never been before; EZCodeGenerator provides intuitive UI for developer to generate their scrapping module, where developer need only some clicks and fill in the information, like the scrapping URL path, selecting the result they want, fill in some error checking, and finally with a button of click to generate the module out. Or even some customizable option, developer can also manually edit their code in the code preview pane as well. And even more conveniently, EZCodeGenerator also provide the ability to compile and test run the module instantly.

# **7.1.2 Project Weakness**

One of the lacking feature of EZCodeGenerator is the application is not yet to be able to provide developer to import custom library (like ActiveX Library) yet, so once there is a chance that developer need to import those libraries into their project, the only way they could is to manually import the library in. Second weakness of the application is that the application is limited to generate the 7 base module functions (OnInit, OnParam, OnLogin, OnEnter, OnExecute, OnResult, OnFinal) for this early built. So, in the case that developer want extra function than the base function in their module, they need to manually added it by their own. The last point to noted out about the weakness of the project is the ability to import the project which is not generated by this application. Currently, EZCodeGenerator can only maintenance on the projects which was generated by EZCodeGenerator only, the older project which manually created by developer could not be import into EZCodeGenerator just yet.

# **7.2 Conclusion**

EZCodeGenerator is a big change to the scrapping work project and it’s the nice helper for scrapping developers. Since EZCodeGenerator provide such convenient and full option for the developer to work with their scrapping project, we truly see developer’s work and mistake has been cut down during their project implementation phase. However, we also admitted that the application is not in its final built yet, so there would be bugs and point to improve here and there. Finally, we hope that EZCodeGenerator will receive good comment from our developer and so we could improve the project to make it works even more magically.