*I.J. Information Engineering and Electronic Business*, **2016, 6, 28-36**
Published Online November 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2016.06.04

# Methodology of Compiling Web-Applications into Executables, Obtaining Seamless Server Installations and GUI Navigations through Qt and C++ Process Communications

**Emmanuel C. Paul**
Department of Mathematics, University of Ilorin, Ilorin, Nigeria
Email: ewebstech@gmail.com

***Abstract*—**Server-side scripts like Hyper-Text Preprocessor, Active Server Pages, and their interaction with databases, has been one of the most popularly used packages for Large Database Intensive Enterprise Software today. In this paper, an approach based on a detailed and efficient method for compiling Web-Applications into executable formats to increase ease of software distribution, where limited internet access exists, is proposed. By this approach, first, one of the server-side scripts and a very popular web-application language in the world, Hyper-Text Preprocessor, is employed as a case study. Second, the methodology of using C++ for writing server installation scripts and creating Graphics User Interface Applications with Qt is shown, with tested applications. Third, Inno Setup Compiler scripts are written and used for compiling into installation and uninstallation setup files. Finally, the relevance of offline Web-Applications for solving scientific problems, the enhancement of C++ codes powered by Graphics User Interface for scientific computation, through inter-channels communication using Qt, and the steps required to easily conquer the challenges faced during the installation of Web-Applications' Servers and Databases like MySQL, are discussed. This approach is efficiently manifested by indicating and confirming this computational potential in the installation and usage of offline web-applications.

***Index Terms*—**Compiling Web-Applications, Qt, C++, GUI, Executables, PHP, Seamless Server Installations, Web-Applications' Servers and Databases.

## I. Introduction

The importance and use of Server Side driven Web-Applications have undoubtedly affected the working structure of our world today. According to Lee Babin (2007), Internet scripting technology has come along at a very brisk pace. While its roots are lodged in text-based displays (due to very limited amounts of storage space and memory), over the years it has rapidly evolved into a visual and highly functional medium [2].

It is generally believed that Web Programming/Scripting languages like Hyper Text Markup Language (HTML), Hyper-Text Preprocessor (PHP) and JAVASCRIPT are mostly used for website development projects while other languages like C, C++, Visual Basic or even Python are used to create desktop applications. However, it is also true that Web-Applications can be built, distributed as exes without any existing knowledge of Web Server configurations by the software users. Hence, the software would be navigated and managed by the user just as with any other desktop application.

The flexibility and ease in the development and use of web applications for large network-dependent software projects like Computer Based Tests (CBT) Software and several Enterprise Management Software has made the relevance of web programming important to both the native web developer and also the scientific researcher especially in developing countries, and hence, its relevance cannot be overemphasized. Javascript and its frameworks like jQuery has improved the world of web development today, hereby creating avenues for comfortable, fast and efficient client-side programming. PHP is known for its robust nature, viable documentations, stable version releases and frequent updates. PHP's integration with cURL, Perl and its GD library has placed it amongst the finest programming tool in the programming field.

In these treatise, we disclose practical examples of how Web Applications could be run locally on computer systems, introduce the usage of Qt with C++ to write commands for automatic installation of necessary servers on the client's computer and finally write codes in Inno Compiler setup in order to compile the scripts into .exes which makes the web application install and uninstall like a system application.

The remainder of this paper is organized as follows: Section II gives an overview of the languages used in this paper, Section III contains the C++ methodologies and needed configuration procedures, Section IV entails the GUI application techniques, Section V contains the web application compilation procedures, and the discussion of results is done in Section VI. The Conclusion is given in the final section.

## II. Key LANGUAGES/Technologies Used

### *A. Hyper-Text Preprocessor (PHP)*

PHP is one of the best free open-source server side scripting languages used by many web-application developers. Hence, according to Steve Suehring, Tim Converse and Joyce Park (2009), PHP is a server-side scripting language, usually used to create web applications in combination with a web server, such as Apache. PHP can also be used to create command-line scripts akin to Perl or shell scripts, but such use is much less common than PHPs use as a web language [10].

Matt Zandstra (2000), also revealed that PHP's support for Apache and MySQL further secured its popularity. Apache is now the most-used Web server in the world, and PHP can be compiled as an Apache module. MySQL is a powerful free SQL database, and PHP provides a comprehensive set of functions for working with it. The combination of Apache, MySQL, and PHP is all but unbeatable [9].

Unfortunately, Tutorials on the working principles of PHP is not in the scope of this work. Therefore, no PHP Scripts would be provided here. Alternatively, web sites like http://www.hotscripts.com can be visited and browsed through to get working PHP scripts for test purposes in this work.

### *B. Qt*

Qt is a complete C++ application development framework. It includes a comprehensive

C++ class library, RAD GUI development tool (Qt Designer), Internationalization tool (Qt Linguist), Help browser (Qt Assistant) and comprehensive documentation. QT is very comprehensive in the sense that it possesses

- 400+ fully documented classes,
- Core libs such as GUI, Utility, Events, File, Print, Network, Plugins, Threads, Date and
- Time, Image processing, Styles and Standard dialogs.
- Modules like Canvas, Iconview, Network, OpenGL, SQL, Table, Workspace, XML
- Tools such as Designer, Assistant, Linguist and finally,
- Extensions like ActiveQt, Motif migration and MFC migration.
- The version of Qt IDE used as at the time of this publication is Qt 5.5.1 (MSVC 2013, 32 bit).

### *C. C++*

C++ is a general purpose programming language with a bias towards systems programming. According to the Inventor of C++, Bjarne Stroustrup (1997), C++:

- is a better C,
- supports data abstraction,
- supports object oriented programming, and
- Supports generic programming.

However, this section gives an insight into what this means without going into the finer details of the language definition. Its purpose is to give a general overview of C++ and the key techniques for using it, not to provide you with the detailed information necessary to start programming in C++.

### *D. Inno Setup Compiler*

Inno Setup Compiler is a free installer for Windows programs. First introduced in 1997,

Inno Setup today rivals and even surpasses many commercial installers in feature set and stability.

Key features:

- Support for every Windows release since 2000, including: Windows 10, Windows 8.1, Windows 8, Windows Server 2012, Windows 7, Windows Server 2008 R2, Windows Vista, Windows Server 2008, Windows XP, Windows Server 2003, and Windows 2000. (No service packs are required.)
- Extensive support for installation of 64-bit applications on the 64-bit editions of Windows. Both the x64 and Itanium architectures are supported. (On the Itanium architecture, Service Pack 1 or later is required on Windows Server 2003 to install in 64-bit mode.)
- Supports creation of a single EXE to install your program for easy online distribution. Disk spanning is also supported.
- Standard Windows wizard interface.
- Customizable setup types, e.g. Full, Minimal, Custom.
- Complete uninstall capabilities etc.

## III. Let's Begin

### *A. Setting up the Web-Application running environment*

For the purpose of the subsequent discussions, we would take the following assumptions:

- That we have created a Web-Application written in PHP to computationally find the exact solution to some Differential Algebraic Equations (DAE) problems in which some members of the system are differential equations and the others are purely algebraic, having no derivatives in them.
- There exists some interested users who need to run this program on their computers, and they know nothing about PHP or any complex information about servers and their installations. Hereby, bringing in the question of distributing this application for use in a very convenient manner whereby the user easily installs the application and opens it conveniently without any help.
- That the web-application scripts are written and kept in a folder in the system path *C://*

We would now go over the installation procedures for installing Apache 2.4, PHP 5.4 and MySQL (If needed) on your local machine.

However, installing a Web server (and its related programs) is not as simple as installing commercial applications. There are a lot of variables involved and many things that can go wrong. However, with patience, it can be done without errors occurring. Several components are needed to build a standalone PHP development system. PHP development is often done with either a system called LAMP (Linux, Apache, MySQL, and PHP) or WAMP (Windows, Apache, MySQL, and PHP).

*B. Needed Files*

For the purpose of this discussion, needed configured versions of Apache 2.4, PHP 5.4 and MySQL 5.5 can be downloaded at - https://drive.google.com/open?id=0B2ORI93vyUo8RGlmeVVadmVLdjg

Please Note that all necessary initial/basic configurations have been done on the Apache/conf file and the php.ini settings have also been set appropriately for use on any machine it is installed on. If a different Apache, PHP or MySQL server, downloading a copy and doing some crosschecking whenever problems are encountered with installation.

We would configure the downloaded files when we are dealing with the compiling procedures.

*C. Methodology*

We simply create a GUI Application using the Qt Framework to help the user navigate to the address where the web-application is stored on the server. This means that the GUI Application will act as a bridge between the web-app and the user. We would discuss and show practically how we could write the configuration script in C++ and make the GUI application access and run the process asynchronously, communicate with the process and give the user the results generated from the C++ server installation process. Therefore, after compiling, the user installs the setup file, and gets access to an (exe) application that can either give him options to install servers either by clicking a button, or do it automatically when the (exe) application is opened, and also be able to perform some other functions that might be useful for the web-application like providing navigations to the web application.

Although the whole process is a bit tricky and for non-Qt programmers can be very burdensome and time consuming to understand, but with the examples and explanations here, it would be surprisingly easy to develop and implement.

*D. Writing the Server Installation Program with C++*

We would now write a simple C++ program to access the server configuration files and install them as a service. The purpose of installation as a service is to enable an auto start-up of the servers during system start-up.

We now create the C++ file named installation.cpp

```
1  #include <iostream>
2  #include <stdlib.h>
3  #include <stdio.h>
4  using namespace std;
5  void mysqlserver();
6  void startapache();
7  void startmysql();
6  void main()
7  {
8system("C:\\MySoftwareFolder\\Apache24\\bin\\httpd
- k install");
9   mysqlserver();
10 startapache();
11 startmysql();
12 }
13 void startapache()
14 {
15system("C:\\MySoftwareFolder\\Apache24\\bin\\http
d - k start");
16 }
17 void startmysql()
18 {
19
system("C:\\MySoftwareFolder\\mysql\\bin\\mysqld");
20 cout<< "Program Complete" <<endl; exit(0);
21 }
21 void mysqlserver()
22 {
23  system("C:\\MySoftwareFolder\\mysql\\bin\\mysqld
- install")) ;
24 }
```

Line 1 to 3 includes the necessary standard header files needed for the task we want to accomplish. Line 5, 6 & 7 performs prototyping of the functions - *mysqlserver(), startapache()* and *startmysql()*. This is done because these functions when called inside the main function on Line 9, 10 and 11, will produce a compilation error because it has not been declared prior to that time. Line 8 inside the *main()* function, performs a system command operation that installs the apache server if and only if the path included as its argument is correct/found. Line 9 installs the MySQL server, while Line 10 and 11 starts Apache's Server and MySQL Server respectively by performing the system commands on Line 15 and 19. Once installed as service and started successfully at first, this servers requires no future additional effort in starting up again, because, it starts automatically during system start-up.

For this discussion we would take *MySoftwareFolder* as the folder where the intended software is to be compiled into, i.e. *C:/MySoftwareFolder/.../.*

This is where the earlier downloaded servers would be extracted into. Otherwise, the files can also be extracted to any other folder, as long as the path passed as the argument in the *system()* function tallies with it.

The *System()* function then performs an automated command-line shell/batch operation by looking for the path specified as its argument, and if found, tries running the code. If successful, a success message shows up, else, an error message would be shown. The Apache 2.4 server does not install itself as a service twice, once installed at first, it would not install again when next it is told to except it no longer exists on the local disk or has been stopped manually for reasons best known. Same goes for the MySQL server too.

Now compile installation.cpp file in a C++ compiler to get an exe file called installation.exe

Move this installation.exe program into a folder called bin inside the MySoftwareFolder to produce the path *C:/MySoftwareFolder/bin/installation.exe*. When this is done we are set to move into the next process.

## IV. Developing the GUI Interface Using Qt

### A. Getting and setting up Qt IDE for C++

*Qt* for C++ IDE can be downloaded from its website at https://www.qt.io/download-open-source/section-2.

Problems with installation might be encountered if the wrong bit version for the development system is downloaded. The version being used for the purpose of this work is 64-bit Qt 5.5.1.

Most functions covered during this discussion would be duly explained to induce understanding to the Novice Qt User and increase the knowledge of Qt Developers. The example given here serves as a possible tool for further development in similar regards.

Graphics User Interface Applications can be developed in Qt either by using the built in Qt Visual Designer or by totally hand coding to produce the designs you need. For large projects, many Qt Developers prefer the Qt Visual Designer because they find it more natural and faster than hand-coding, and they want to be able to experiment with and change designs more quickly and easily than is possible with hand-coded forms.

Using the Qt Designer, here is how we create a Dialog with two buttons in Horizontal layout and a label to display *'Welcome to My App'*.

If Qt IDE is configured properly, create a new Project - Choose Application - Qt Widgets Application - Enter Project Name - set class name to *Myguiapp* and change base class to *QDialog* - Save.

Once project has been created, on the toolbar, then click on the "design tab", drag two Push Buttons from the section between the Dialog's UI form and the Toolbar and rename them to Start Application and Quit Application respectively. Highlight them both and press *Ctrl+H* on your keyboard. Drag another label to the form above the buttons and rename it "Welcome to My App". Highlight them all and press *Ctrl+V* on your keyboard.

If all goes well, the result should be identical to the image below

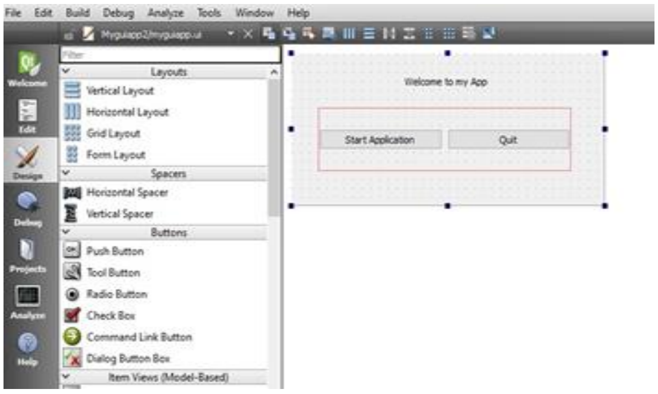


Fig.1. Qt Designer's Window

Here is the source code.

First we create the dialog's header file *myguiapp.h*

```
1 #ifndef MYGUIAPP_H
2 #define MYGUIAPP_H
3 #include<QProcess>
4 #include <QDialog>
5 namespace Ui {
6 class Myguiapp;
7 }
```

Lines 1 and 2 protects the header file against multiple inclusions. Line 3 includes The *QProcess* class which is used to start external programs and to communicate with them. Line 4 includes the definition of *QDialog*, the base class for dialogs in *Qt*. *QDialog* inherits QWidget.

Next, we define Myguiapp as a subclass of *QDialog*

```
8 class Myguiapp : public QDialog
9 {
10 Q_OBJECT
11 public:
12 explicit Myguiapp(QWidget *parent = 0);
13 ~Myguiapp();
```

The Q OBJECT macro at the beginning of the class definition on Line 10 is necessary for all classes that define signals or slots. The *Myguiapp* constructor is typical of Qt widget classes. The parent parameter specifies the parent widget. The default is a null pointer, meaning that the dialog has no parent. Line 13 is a deconstructor which handles memory management of the dialog.

```
14 private slots:
15 void begin_installations();
16 void on_action_started();
17 private:
18 Ui::Myguiapp *ui;
19 QProcess myProcess;
20 };
21 #endif // MYGUIAPP_H
```

In the class's private section, we declare two slots. The first slot is called whenever the start button emits a clicked signal, while the second slot is called whenever the process emits the *readyRead()* signal. The slots keyword is, like signals, a macro that expands into a construct that the C++ compiler can digest.

Line 19 declares a *QProcess* object which will be used in Myguiapp.cpp implementation file.

We would now look at the *Myguiapp.cpp* file which is an implementation of the *Myguiapp* dialog class.

```
22 #include "myguiapp.h"
23 #include "ui_myguiapp.h"
24 #include<QDesktopServices>
25 #include<QUrl>
26 #include<QString>
27 #include<QMessageBox>
28 #include<QProcess>
29 #include<QFile>
30 #include<QDir>
31 Myguiapp::Myguiapp(QWidget *parent) :
32 QDialog(parent),
33 ui(new Ui::Myguiapp)
34 {
35 ui->setupUi(this);
36 connect(ui->pushButton, SIGNAL(clicked(bool)),
this, SLOT(begin_installations()));
37 connect(ui->pushButton_2, SIGNAL(clicked(bool)),
this, SLOT(close()));
38 connect(&myProcess, SIGNAL(readyRead()), this,
SLOT(on_action_started()));
39 }
```

Line 24 to 30, contain all the necessary header files. In line 32, we pass on the parent parameter to the base class constructor.

In line 36, we connect the first *pushbutton* to a slot when a *clicked()* signal is emitted by the button. This takes the action to the function which executes and returns required result. Line 37 connects the second pushbutton to a slot that closes the GUI App when it emits the *clicked()* signal. Line 38 connects the *readyRead()* signal to a slot that performs a specific action. i.e. When *QProcess* executes an external process, it emits *readyRead()* whenever data is available to be read from that process.

```
40 Myguiapp::~Myguiapp()
41 {
42 delete ui;
43 }
44 void Myguiapp::begin_installations()
45 {
46 QString program;
47 program = "C:/Easytrades/bin/syscommands.exe";
48
myProcess.setProcessChannelMode(QProcess::Merge
dChannels);
49 myProcess.start(program);
50 }
```

The slot on Line 44 is called whenever the first pushbutton's *clicked()* signal is emitted. The function declares a *QString* variable on line 46, sets the path to the C++ installation program created earlier. Line 48 sets the Process channel mode to *MergedChannels* using the *QProcess* object *'myProcess'*. Line 49 starts the program.

```
51 void Myguiapp::on_action_started()
52 {
53 QFile file;
54 QDir::setCurrent("C:/Easytrades");
55 file.setFileName("donotdelete.xml");
56 if(file.exists())
57 {
58 if(!myProcess.waitForFinished()){
59 QMessageBox::warning(this, tr("Error"),
   tr("<p>Error has occurred. App wont start"));
60 }
```

The slot in line 51 is called for execution whenever the *QProcess* Object emits a *readyRead()* signal. To disable multiple installation attempts, we resort into creating a file called *'donotdelete.xml*' place it into the installation folder of the application. This file can be of any type. The idea is just to find the file in the directory stipulated and install the server if the file is available. Once the install program has been executed, the object *'file'* deletes that file from its directory and keeps the installation from happening multiple times. This method is only one of the many ways to eradicate multiple installations.

Line 58 checks the *waitforFinished()* function to see if the process has started and ended, if it returns false, a message box is displayed with an error message.

```
61 else{
62 if(myProcess.Running)
63 {
64 QByteArray newData = myProcess.readAll();
65 int exitstatus = myProcess.exitCode();
66 if(exitstatus == 0)
67 {
68 QMessageBox::information(this, tr("Server
Installation"),
69 tr("<h3>Installation Status</h3>" + newData +
"\n"));
70 QString link="http://localhost/appurl/";
71 QDesktopServices::openUrl(QUrl(link));
72 else{
73 QApplication::quit();
74 }
75 }
76 }
77 file.remove();
78 }
79 else{
80 QString link="http://localhost/appurl/";
81 QDesktopServices::openUrl(QUrl(link));
82 }
83 }
```

If the Process has emitted the *waitforFinished()* signal, Line 61 begins execution.

The *readAll()* functions outputs results which take the form of *QByteArray*, hence, declaring *newData* to be *QByteArray*, we can retrieve the value of *myProcess.readAll()*. Line 65 gets the value of the process' *exitcode* by retrieving the data into *exitstatus*. If the process *exitcode* is 0, this indicates that the program completed properly and returned the correct exit code.

If the *exitcode* returned is 0, a message is displayed to the user on the status of the installation.

Line 71 then triggers the link set in Line 70, if found, it opens the link which is the url pointing to where the web application is stored in the server's htdocs folder. Otherwise, in the case of not finishing the process properly and inability to return *exitstatus* of 0, the application quits.

The file *(donotdelete.xml)* is then deleted from its directory on Line 78. Line 80 to 83 gets executed whenever the file *(donotdelete.xml)* is not found.

We then create the main.cpp file.

```
85 #include "myguiapp.h"
86 #include <QApplication>
87 int main(int argc, char *argv[])
88 {
89 QApplication a(argc, argv);
90 Myguiapp w;
91 w.show();
92 return a.exec();
93 }
```

Line 85 includes the *myguiapp* header file while the Line 86 include the definitions of the *QApplication* class. For every Qt class, there is a header file with the same name (and capitalization) as the class that contains the class's definition.

Line 89 creates a *QApplication* object to manage application-wide resources. The QApplication constructor requires *argc* and *argv* because *Qt* supports a few command-line arguments of its own. Line 91 makes the object created from the *Myguiapp* widget visible and Line 92 passes control of the application on to Qt. At this point, the program enters the event loop. This is a kind of stand-by mode where the program waits for user actions such as mouse clicks and key presses.

Running the application should produce the result similar to the diagram below:

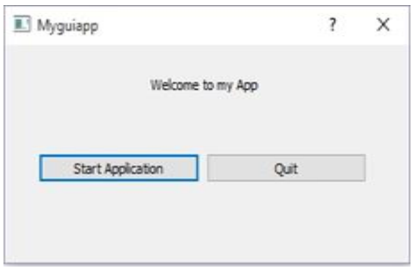


Fig.2. Simple GUI Dialog for navigating Web-Application with Qt

*B. Qt Applications Running Environment*

In order to run a *Qt* Application successfully

on a computer different from the one in which the application was compiled or in a path that is not accessible by the *Qt* Application, we would need to transfer some dynamic-link library (DLL) files to share code and other resources necessary to perform particular tasks in the course of running the Qt application. There are some fundamental Qt DLLs that must be compiled alongside your application in order to enable the application run on almost any computer system it is installed.

- Open this path *c:/your-path/Qt5.5.1/5.5/msvc2013 64/*
- All contents in the folder 'bin' and 'plugins' would need to be copied as they are into the root folder of the application. i.e. MySoftwareFolder. This is to ensure all needed files for the application you are creating are all included without any omissions.
- Next, we run myguiapp.exe, and while it is running, we highlight all files we copied from bin and platform folder which are now in the Application's root folder, and we delete them.
- Once the delete button is pressed, all DLL files and folders selected which are not in use by the application will be deleted and the files that are in use would be left behind. This way, we have exactly all the DLL files we need for our application.

## V. Compiling with Inno Setup Compiler

For the purpose of this work, the version of Inno Setup Compiler used is 5.5.8.

Once downloaded and installed, a new script can be created by *Ctrl+N*. On the Inno Setup Script

Wizard dialog, Check the 'create a new empty script' box and click finish.

Please Note that the wizard can also be used, but it does not provide all available options.

*A. Setting & Configuring Server Files*

Before we dive into compilation, we need to make sure all files are where they should be and the paths in the Apache Configuration file are correctly linked. Some configurations have to be changed in the Apache configuration file *C:/yourpath/Apache24/conf/httpd.conf/*

It is also necessary that the folder which contains the web application is stored inside the htdocs folder which can be found in the path *C:/your-path/Apache24/*

First, the *ServerRoot* will have to be changed. For the sake of this demonstration, we want the compiled setup to extract its contents into a specified folder during setup called MySoftwareFolder. Since the path to the Apache server files have changed, we need to change some values in the configuration file –

```
ServerRoot "c:/MySoftwareFolder/Apache24".
```

Next, we need to change the directory address to some php modules to match the directory path on the user's computer.

```
LoadModule php5_module "c:/MySoftwareFolder/php/php5apache2_4.dll"
```

Then we do the same to the DocumentRoot, i.e. the directory out of which you will serve your documents.

```
DocumentRoot "c:/MySoftwareFolder/Apache24/htdocs"
<Directory "c:/MySoftwareFolder/Apache24/htdocs">
```

Then the CGI directory

```
<Directory "c:/MySoftwareFolder/Apache24/cgi-bin">
```

And Lastly, the PHP ini Directory

```
PHPIniDir C:/MySoftwareFolder/php
```

Next, we need to correct some paths in the php.ini file to match the installation directory.

```
extension_dir = "C:\MySoftwareFolder\php\ext"
session.save_path = "C:\MySoftwareFolder\php\tmp"
session.save_path = "C:\MySoftwareFolder\php\tmp"
```

*B. Compiling*

We then create a file with the name *ompilemyapp.iss* Here is the source code

```
1 #define MyAppName "Your Software Name"
2 #define MyAppVersion "1.0"
3 #define MyAppPublisher "Publisher's Name"
4 #define MyAppURL "http://www.example.com"
5 #define MyAppExeName "Myguiapp.exe"
```

Line 1 to 5 defines the identity of the to-be-compiled software.

```
6 ; NOTE: The value of AppId uniquely identifies this application.
7 ; Do not use the same AppId value in installers for other applications.
8 [Setup]
9 AppId={{2EDA0D0A-43B1-4B39-9A6C-47EA23F4D2E5}
10 AppName={#MyAppName}
11 AppVersion={#MyAppVersion}
12 ;AppVerName={#MyAppName} {#MyAppVersion}
13 AppPublisher={#MyAppPublisher}
14 AppPublisherURL={#MyAppURL}
15 AppSupportURL={#MyAppURL}
16 AppUpdatesURL={#MyAppURL}
17 DefaultDirName=C:/MySoftwareFolder
18 DisableProgramGroupPage=yes
19 LicenseFile=C:\your-path\license.txt ;
20 OutputDir=C:\your-path\Desktop
21 OutputBaseFilename=myguiapp_setup_3.01
22 SetupIconFile=C:\your-path\icon.ico
23 Compression=lzma
24 SolidCompression=yes
25 ;Additional Options
26 AppContact=ewebstech@gmail.com
27 AppCopyright=Copyright (C) 2014-2016 company, Inc.
28 AppSupportPhone=+234 000 000 000
29 ChangesEnvironment=yes
30 CloseApplications=Force
31 VersionInfoVersion=3.0
```

Line 21 defines the name of the setup file after compilation. Compression lzma is the method of compression employed by the 7-Zip LZMA compressor. It typically compresses significantly better than the zip and bzip methods. Line 29, When set to yes, at the end of the installation Setup will notify other running applications (notably Windows Explorer) that they should reload their environment variables from the registry. Line 30, If set to yes or force and Setup is not running silently, Setup will pause on the Preparing to Install wizard page if it detects applications using files that need to be updated by the [Files] or [InstallDelete] sections, showing the applications and asking the user if Setup should automatically close the applications and restart them after the installation has been completed. If set to yes or force and Setup is running silently, Setup will always close and restart such applications, unless told not to via the command line. If set to force Setup will force close when closing applications, unless told not to via the command line. Use with care since this may cause the user to lose unsaved works. If your installation creates or changes an environment variable but doesn't have *ChangesEnvironment* set to yes, the new/changed environment variable will not be seen by applications launched from Explorer until the user logs off or restarts the computer.

```
32 [Languages]
33 Name: "english"; MessagesFile: "compiler:Default.isl"
34 [Tasks]
35 Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
36 GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
37 Name: "quicklaunchicon"; Description: "{cm:CreateQuickLaunchIcon}";
38 GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked; OnlyBelowVersion: 0,6.1
```

Line 35 gives the user the option of creating desktop shortcut icon and Line 37 a quick launch icon in the case of older operating systems.

```
[Files]
Source: "C:\path-where-Myguiapp.exe-app-is-stored\Myguiapp.exe";
DestDir: "{app}"; Flags: ignoreversion
Source: "C:\path-to-downloaded-apacheserver-folder\Apache24\*";
DestDir: "{app}\Apache24"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\EWEBS\Documents\Easytrades\imageformats\*";
DestDir: "{app}\imageformats";
Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\EWEBS\Documents\Easytrades\mysql\*";
DestDir: "{app}\mysql"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Users\EWEBS\Documents\Easytrades\php\*";
DestDir: "{app}\php";
Flags: ignoreversion recursesubdirs createallsubdirs
```

The Source is the path to where the files to be compiled reside in, while the DestDir is the destination for the compiled application. If there exists some files to be extracted into its own separate folder inside the application folder MySoftwareFolder, then a '*' is placed after the folder's name as shown above. The ***ags*** *recursesubdirs createallsubdirs* needs to be placed in front of the Flag for that file. Otherwise, it would put all files and sub-folders into the same directory with every other file in *MySoftwareFolder*. More files can be added following the procedure for the Source, DestDir & Flags.

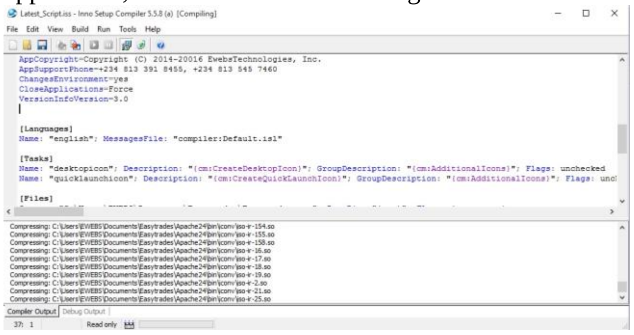
Note: Don't use "Flags: ignoreversion" on any shared system files.

```
[Icons]
Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}";
Tasks: desktopicon
Name: "{userappdata}\Microsoft\Internet Explorer\Quick Launch\{#MyAppName}";
Filename: "{app}\{#MyAppExeName}"; Tasks: quicklaunchicon

[Run]
Filename: "{app}\{#MyAppExeName}";
Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}";
Flags: nowait postinstall skipifsilent
```

Running this script would compile the files to produce our setup file which contains our assumed

Web-application which solves DAE problems which also can now be distributed easily at a compressed size. The user also gets an opportunity to uninstall the application, without the risk of leaving the servers behind.



```
Latest_Script.iss - Inno Setup Compiler 5.5.8 (a) [Compiling]
File Edit View Build Run Tools Help

AppCopyright=Copyright (C) 2014-20016 EwebsTechnologies, Inc.
AppSupportPhone=+234 813 391 8455, +234 813 545 7460
ChangesEnvironment=yes
CloseApplications=Force
VersionInfoVersion=3.0


[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
Name: "quicklaunchicon"; Description: "{cm:CreateQuickLaunchIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: uncl

[Files]

Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-154.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-155.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-158.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-16.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-17.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-18.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-19.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-2.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-21.so
Compressing: C:\Users\EWEBS\Documents\Easytrades\Apache24\bin\iconv\iso-ir-25.so

Compiler Output | Debug Output
37: 1    Read only
```

Fig.3. Compilation

However, for the purpose of installing the servers, the GUI application would have to be opened in administrator's mode to give appropriate access for installation. This is typical for Windows Operating Systems.

## VI. Discussion of Result

With the proposed outlined steps and algorithms above, we can successfully create a system through which web applications could be compiled, distributed and installed by any one easily. These also gives an opportunity for navigating to the software through the GUI application that acts as a bridge between the Web application, Server, and the user. *Fig. 2* shows a simple typical example of a GUI dialog created using Qt that can function as the bridge between the Web application, server, and the user. Any type of GUI application can be created for these purpose, i.e. any type of style could be adopted. Therefore, if the outlined procedures are followed, Web Applications can now be transferred as compiled (exe) applications, its servers can be installed silently by Qt and C++ communications and can also be navigated by Qt GUI applications.

## VII. Conclusion

In this paper, the methodology of compiling web-applications into executable file formats, obtaining seamless server installations and GUI navigations, through Qt and C++ process communications is proposed and its potential manifests good performance in the indication and confirmation of offline web-applications' installation and usage. Undoubtedly, an expanded use of web-applications has been illustrated in easy-to-implement procedures of describing significantly, the methods for the silent and automated installation of a web server using native C++ for its distribution and usage offline; the steps to be taken when creating basic GUI applications that run the installation and navigation of the web-application without complexities on the user's end; and finally, the method for compilation into exes at a well compressed size. This ideology would aid the distribution of, and enhance opportunities in developing countries, for the usage of offline web-applications, especially in countries with little or no internet access. Most Lines in the displayed source codes were also explained for clarity on the author's style of implementation.

## ACKNOWLEDGMENT

The author wishes to thank the Almighty God for bringing him this far. Also, many thanks to Professor Christopher Thron, University of Edinburgh, U.S.A, Dr. O.T. Olotu, Dept. of Mathematics, University of Ilorin and J.B. Okeowo, University of Ilorin, for their support and constructive reviews.

## REFERENCES

[1] Gabe Rudy, *Cross-platform C++ Development Using Qt*, 2005.
[2] Lee Babin, *Beginning Ajax with PHP: From Novice to Professional*, 2007.
[3] Wolfram Mathematica, *Differential Equation Solving With Dsolve.*
[4] Jasmin Blanchette and Mark Summer_eld. *C++ GUI Programming with Qt*, 2006.
[5] Juan Souli, *The C++ Language Tutorial*.
[6] Harry Fuecks, *The Php Anthology, Volume 1: Foundations.*
[7] Quentin Zervaas, *Practical Web 2.0 Applications with PHP*, 2008.
[8] Bjarne Stroustrup, *The C++ Programming Language Third Edition, AT&T Labs*, Murray Hill, New Jersey, 1997.
[9] Brian Schaffner, Matt Zandstra, *Teach Yourself PHP4 in 24 Hours*, 2000.
[10] Steve Suehring, Tim Converse and Joyce Park, *PHP6 and MySQL*, 2009.
[11] Koch, N., Wirsing, M.: Software Engineering for Adpatative Hypermedia Applications. In: 3rd Workshop on Adaptative Hypertext and Hypermedia (2001).
[12] Muruguesan, S., Desphande, Y.: Web Engineering. Software Engineering and Web Application Development. Springer LNCS – Hot Topics (2001).
[13] Chen, J.Q, & Heath, R. D. (2005). Web application development methodologies. In W. Suh (Ed.), Web Engínerring: Principles and techniques. Hershey, PA: Idea Group Publishing
[14] Coda, F., Ghezzi, C,. Vigna, G., & Garzotto, F. (1998, April 16-18). *Towards a software engineering approach to Web site development.* Paper presented at the Ninth International Workshop on Software Specification and Design (IWSSD-9), Ise-shima, Japan.
[15] Deborah Kurata, *Doing Web Development: Client-Side Techniques*, 2008.
[16] Make a windows installer file (.exe file) using Inno Setup Compiler, published on 31st, July 2015, https://www.scirra.com/tutorials/4779/make-a-windows-installer-file-exe-file-using-inno-setup-compiler/page-3.

## Authors' Profiles

**Emmanuel C. Paul,** born in Lagos, Nigeria on the 28th April, 1994, is an undergraduate student for bachelor's degree for Mathematics in University of Ilorin, Ilorin, Nigeria. Emmanuel C. Paul majors in computational and applied mathematics.

His current research is based on the mathematical modelling of bacteria resistance to multiple antibiotics and immune system response.

**How to cite this paper:** Emmanuel C. Paul,"Methodology of Compiling Web-Applications into Executables, Obtaining Seamless Server Installations and GUI Navigations through Qt and C++ Process Communications", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.8, No.6, pp.28-36, 2016. DOI: 10.5815/ijieeb.2016.06.04