Using the Epanet Toolkit v2.00.12 with Different Programming Environments

Oscar Tomas Vegas Niño
Fernando Martínez Alzamora
Joan Carles Alonso Campos
Velitchko G. Tzatchkov

```
izeComponent();

void BtnOpen_Click(object sender, 

penFileDialog OpenFD = new OpenFileDialo
OpenFD.ShowReadOnly = true;
OpenFD.InitialDirectory = System.Environmen
OpenFD.Filter = "Inp files (*.inp)|*.inp";
OpenFD.RestoreDirectory = true;
OpenFD.Title = "Select Inp file";

if (OpenFD.ShowDialog() == DialogResult.OK)
{
    TxtINP.Text = OpenFD.FileName;

INP.SelectionStart = TxtINP.TextLeng
inks.Text = string.Empty;
s.Text = string.Empty;
t.Text = string.Empty;
```

USING THE EPANET TOOLKIT v2.00.12 WITH DIFFERENT PROGRAMMING ENVIRONMENTS

Oscar Tomas Vegas Niño
Fernando Martínez Alzamora
Joan Carles Alonso Campos
Velitchko G. Tzatchkov

2018

627.0113 Vegas Niño, Oscar Tomas
V15 *Using the Epanet toolkit v2.00.12 with different programming environments* / Oscar Tomas Vegas Niño et al.. -- Jiutepec, Mor. : Mexican Institute of Water Technology, 2018.
119 p.

ISBN: 978-607-9368-97-5 (digital work)

1. Computer Programs 2. Hydraulic Simulation

Authors:
Oscar Tomas Vegas Niño
Fernando Martínez Alzamora
Joan Carles Alonso Campos
Velitchko G. Tzatchkov

Editorial coordination:
José Manuel Rodríguez Varela
Gema Alin Martínez Ocampo
Urban Hydraulics Department
Mexican Institute of Water Technology

Design and layout:
Gema Alín Martínez Ocampo

English translation:
Dante Cuevas Navarrete

First edition in English: 2018

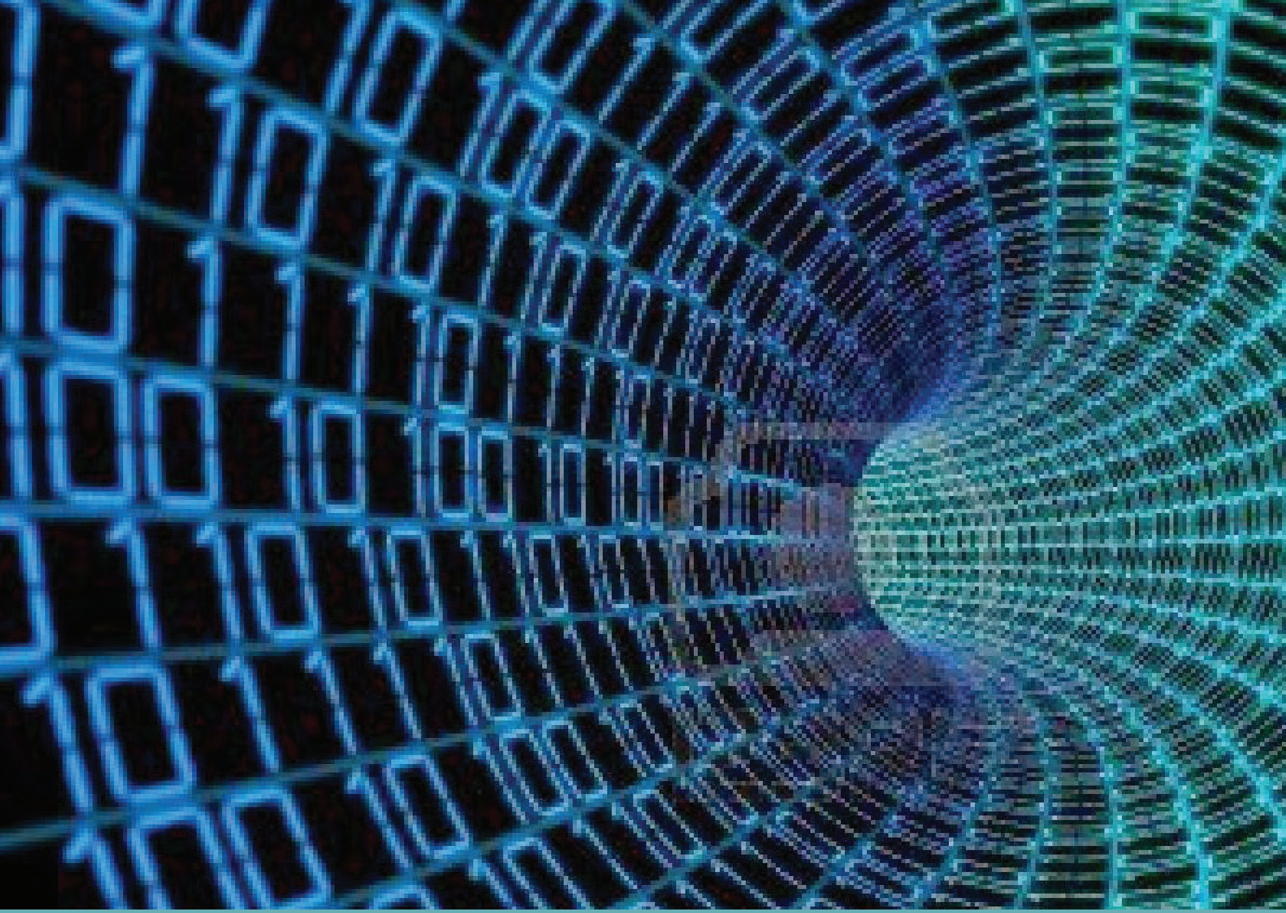
Mexican Institute of Water Technology
Paseo Cuauhnáhuac 8532
62550 Progreso, Jiutepec, Morelos
Mexico
www.imta.gob.mx

ISBN: 978-607-9368-97-5 (digital work)

C o n t e n i d o

T a b l e s

F i g u r e s



PRESENTATION

Water utilities are betting heavily on the use of computer applications such as Computer-Aided Drafting (CAD), and Geographic Information Systems (GIS), along with other computing tools and specialized software for modeling water-supply systems in both freeware or licensed software forms.

Many of the computer applications associated with the use of models need to have a hydraulic simulator that will provide a network response to any given scenario according to the defined hydraulic and water-quality variables. Epanet, on top of covering this need as an independent application that has its own graphical interface for constructing a network model and analyzing its results, permits the connection of its Toolkit to any other graphical interface, thus opening a broad field of possibilities for any technician or researcher that wants to go beyond what the application currently offers.

The Epanet Toolkit can be integrated into any native programming environment or licensed software programs such Excel, AutoCad, ArcGIS, etc. or their freeware versions such as OpenOffice, gvSIG, QSIG etc. As a matter of fact, this is what some commercial programs developed by companies or universities have done with the aim of designing and analyzing water distribution networks.

The Epanet toolkit library has many functions that help us to retrieve and modify certain parameters of a network model before and after carrying out a hydraulic simulation. This permits us step-by-step control of its simulation process. Little information exists in the technical literature, however, concerning the how to use of all these functions. More information is needed on how to use the Toolkit in the most common programming environments and how to resolve the most frequent problems.

The current book aims to help postgraduate students, technical personnel and researchers to begin programming with the Epanet library using different languages and environments. In so doing, use of the majority of the functions contained in the library can be learned by carrying out practical exercises using the Visual Studio 2017 programming environment, thus, creating an understanding in the use that can be used afterwards to create his/her own tools.

PROLOGUE

The Epanet software has become the most widely used tool worldwide for modeling water distribution networks. Its success is due to the fact that its distribution is free, and also because we can use the source code of the graphic interface, as well as its hydraulic calculation engine for our own purposes.

The Epanet Toolkit can be downloaded from the official website of the United States Environmental Protection Agency (USEPA US). It is a set of files (mainly headers), that enable us you to use the Epanet library (epanet2.dll) functions. These files were created in programming environments that are no longer used today.

The Epanet Toolkit can be integrated into any native programming environment or in many proprietary software packages, such as Excel, AutoCAD, ArcGIS; or in free software packages as Open Office, gvSIG, QGIS, etc.

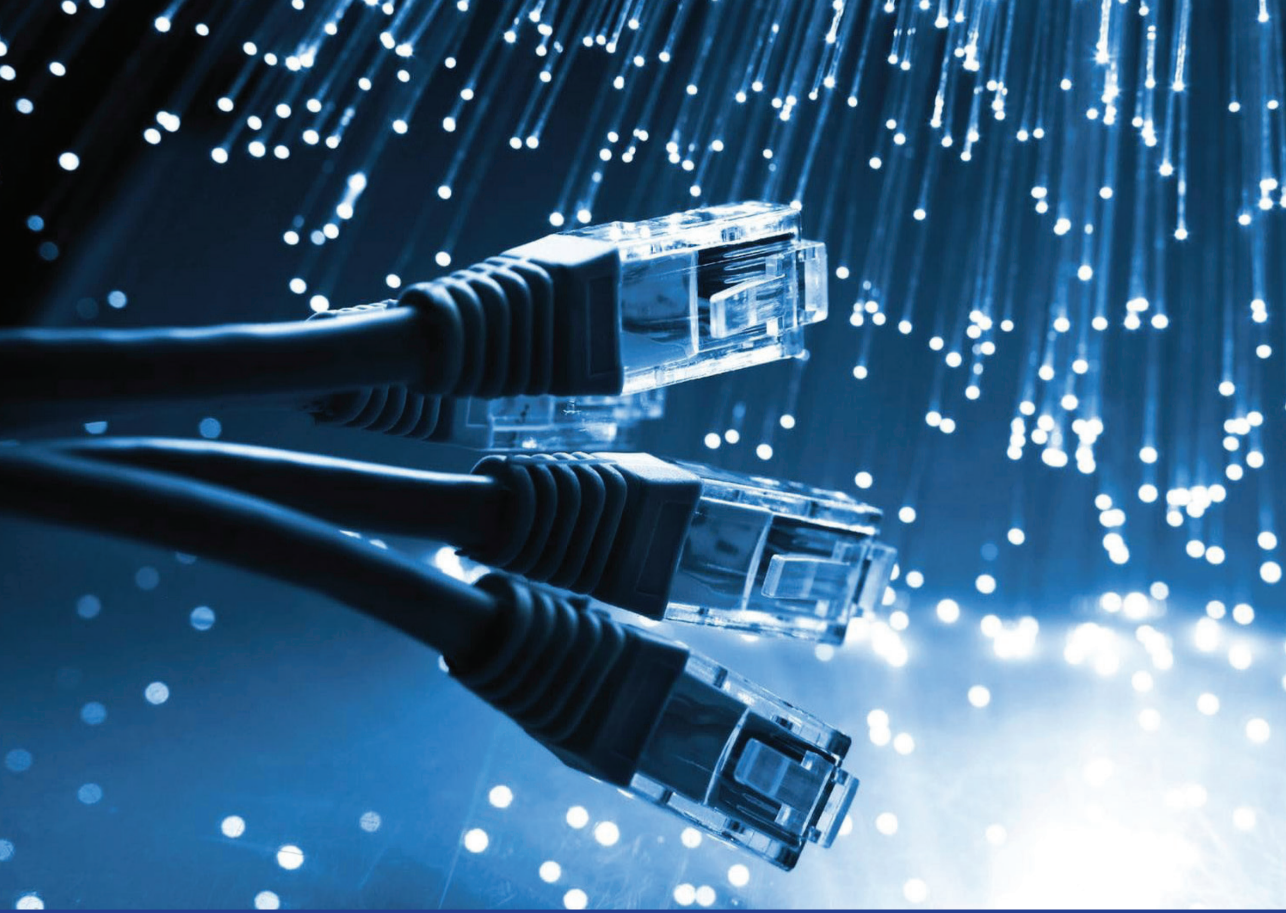
In the previous Spanish edition of this book, a great effort was made to collect all the existing information related to how to connect the Epanet library to different programming environments. No other manual offered step by step explanation on how to make such a connection.

The main changes made with respect to the Spanish edition are:

- Connection of the Epanet library (epanet2.dll) to the Visual Basic and C ++ products of the new version of Microsoft Visual Studio Community 2017.
- Connection of the Epanet library (epanet2.dll) to the new version of Matlab (R2017b).
- Addition of a new environment from to which you can connect the Epanet library: Microsoft Excel 2016, through its programming module (VBA).

The objective pursued in this new edition remains the same as in the Spanish edition, to help understand how to connect the Epanet library, in its two versions 2.00.12 for 32 bits, and 2.01.00 for 64 bits, to different programming environments, and thus to be able to analyze and solve problems related to the design and operation of drinking water distribution networks.

The authors



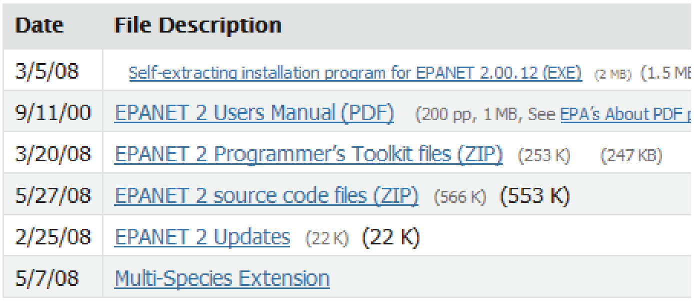
1 Epanet Software

1.1. What is Epanet?

Epanet is a computer program that carries out extended-period simulations (one or several days) of the hydraulic behavior and the evolution of water quality in pressurized-pipe networks. A network is made up of pipes, nodes (pipe junctions) pumps, valves, storage tanks, and reservoirs. Epanet tracks the evolution of flows in pipes, the pressure at every node, water height in tanks, and the concentration of chemical species throughout the network during a simulation period made up of multiple time steps. Additionally, it can simulate water age and perform source tracing.

Epanet has been designed and developed by the United States Environmental Protection Agency, USEPA. As such, it is in the public domain and is distributed freely on its official website: https://www.epa.gov/water-research/epanet. The original version was developed in English and has since been translated to other languages such as Spanish , French, Italian, Czech, German, Portuguese, etc.

On the official EPA webpage, aside from being able to download the program (v.2.00.12), the additional files, shown in Figure 1.1. can be downloaded.



Date	File Description
3/5/08	Self-extracting installation program for EPANET 2.00.12 (EXE) (2 MB) (1.5 M
9/11/00	EPANET 2 Users Manual (PDF) (200 pp, 1 MB, See EPA's About PDF
3/20/08	EPANET 2 Programmer's Toolkit files (ZIP) (253 K) (247 KB)
5/27/08	EPANET 2 source code files (ZIP) (566 K) (553 K)
2/25/08	EPANET 2 Updates (22 K) (22 K)
5/7/08	Multi-Species Extension

Figure 1.1. List of Epanet files on the EPA website

The following files are available on the EPA website:

- User's manual
- Programmer's toolkit
- The Epanet source code files
- A text file that lists program updates
- Multi-Species Extension

To learn how to use this tool you can consult the User's Manual that is available on the EPA website.

1.2. Physical and non-physical components

Epanet models a water distribution network as a group of links connected to nodes. The links represent pipes, pumps or control valves. The nodes represent tanks, reservoirs, or pipe junctions with or without water demand. Epanet employs these six objects to carry out a simulation in a water distribution network.

Aside from these aforementioned elements, there are other complementary components that help to describe the behavioral and operational aspects of the system. These non-physical components are: curves, patterns and controls. For more information, consult the Epanet User's Manual.

1.3. What can be done on Epanet?

Epanet was developed to be a research tool for improving our knowledge of the movement and fate of drinking water constituents within water distribution networks. Sampling program design, hydraulic model calibration, chlorine residual analysis, and total chlorine doses supplied to a water user are among its different uses. Epanet can also be of help when evaluating different management strategies aimed at improving water quality throughout the system, such as:

- Altering source utilization within multiple source systems
- Altering pumping and tank filling/emptying schedules
- Secondary water treatment implantation, such as re-chlorination at storage tanks
- Targeted pipe cleaning and replacement.

Epanet can also be used to improve the hydraulic network characteristics helping in the design of new elements, the reduction of operational costs, vulnerability studies, bottleneck detection, water quality and residence time analysis in the pipes, pressure control, to regulate the use of tanks in order to reduce water residence time, predicting the network response in supply point closures or the incorporation of new urbanizations, contingency planning, as well as the introduction of uncontrolled contaminants, network sectorization for controlling leaks and better network management, etc. In the future, models should be used as a continuous support tool for the technical decision-making in all water utilities.

1.4. Application limitations

By being a quasi-static hydraulic model (non-inertial model, that is to say, it doesn't consider abrupt flow changes in the network), cannot model rapid flow phenomena such as: a rupture in a pipe, water hammer caused by a sudden valve closure or the start of stop of a pump, sudden return valve closure, etc.

In regards to water-quality modeling, Epanet does not consider the dispersion term in the differential equation for transport of the modelled substance. This term, along with the influence of dispersion is important in low velocity network pipes. Tzatchkov el al. (2002)[1] published and applied an algorithm that allows for the consideration of this term using the results file from the Epanet hydraulic calculations.

1.5. Installing Epanet on 32/64-bit Windows

The latest version of Epanet, as well as its dynamic library (dll) have been made to run in a Windows 32-bit Operating System (OS). When newer computers appeared with a Windows 64-bit OS architecture, some Epanet users wondered if the application would cease to work on their updated models. In the end, these worries were without merit, no compatibility problems arose. Currently Epanet can be used on Windows 10 with total functionality[2]. The different versions of Epanet from 1.0 to 2.0 published by the USEPA have been designed and developed to be installed and run on a Windows® environment. Some researchers have rewritten the original source code in other programming languages for use in other environments such as Linux and MAC.

A small problem surfaced, however, when the Online Help file couldn't be opened on the Windows Vista OS or its subsequent versions. This was due to Microsoft's exclusion of the help program WinHlp32.exe on those operating systems. Downloading and installing the WinHlp32.exe program from the Microsoft's official support page https://support.microsoft.com/en-nz/help/917607/error-opening-help-in-windows-based-programs-feature-not-included-or-h resolves this issue.

On that note, Elad Salomons describes the required steps for using the Epanet's online help file web: http://www.water-simulation.com/wsp/2015/10/01/how-to-open-epanets-help-file-in-windows-10/.

1 Velitchko G. Tzatchkov, Alvaro A. Aldama, Felipe I. Arreguin, "Advection-dispersion-reaction modeling in water distribution networks", Journal of Water Resources Planning and Management, American Society of Civil Engineers, vol. 128(5), p. 334-342.

2 http://www.water-simulation.com/wsp/

1.6. User list

Epanet has created a turning point in the world of hydraulic network simulation software. Its free availability, the EPA's backing and the quality work of its author, L. Rossman, have made this possible.

Since Epanet's first appearance, Prof. William James from the University of Guelph (Canada) compiled a user list with a subscription option that, for now, is free of charge. It permits Epanet users to find out the latest news about the program, share experiences, and raise questions that can be answered by more experienced users.

As is the norm in users lists, there is an address listserv@listserv.uoguelph.ca reserved for communication with the server through commands, and another EPANET-USERS@LISTSERV.UOGUELPH.CA for access to messaging. Subscription to the email list is quite simple, all that is required is to send an email to listserv@listserv.uoguelph.ca with the following message: SUBSCRIBE EPANET-USERS and your user name. Example: SUBSCRIBE EPANET-USERS Oscar Vegas Niño.

It is important to leave the subject field blank, and to make sure that the message contains no other text or characters (like a period, etc.). In a question of minutes the system will respond with another email confirming the user's successful subscription to the mailing list or informing of some error in the original message.

To remove yourself from the subscription list, it is necessary to send an email to listserv@listserv.uoguelph.ca with the following message: UNSUBSCRIBE EPANET-USERS user name. To ask questions, make consultations, or give replies it is necessary to send an email to the following address: EPANET-USERS@LISTSERV.UOGUELPH.CA. This message will be sent to all people on the User List.

There is also a developer (researcher) community that works to add features to Epanet and resolve issues surrounding other user's queries. Use the following link: http://community.wateranalytics.org/ to sign-up to this community.

2 Epanet data structure

2.1. Sections that make up the input file

An input file that Epanet can read must have the following extensions *.inp (ASCII text file) or *.net (binary file) along with a defined data structure that stores all the necessary information for viewing and creating a valid simulation of the network model being worked upon.

The Input File data is clustered in distinct sections, each section begins with a keyword enclosed in brackets, which allows for clear identification. The various key words, which total 29, are listed on Table 2.1.

Table 2.1. Keywords for the different sections of the input file

Network Componets	System Operation	Water Quality	Options and Reporting	Network Map/Tags
[TITLE]	[CURVES]	[QUALITY]	[OPTIONS]	[COORDINATES]
[JUNCTIONS]	[PATTERNS]	[REACTIONS]	[TIMES]	[VERTICES]
[RESERVOIRS]	[ENERGY]	[SOURCES]	[REPORT]	[LABELS]
[TANKS]	[STATUS]	[MIXING]	[REACTIONS]	[TAGS]
[PIPES]	[CONTROLS]		[BACKDROP]	
[PUMPS]	[RULES]		[END]	
[VALVES]	[DEMANDS]			
[EMITTERS]				

The order of these distinct sections is unimportant, just assure that when a section refers to a node or link, that it be established previously in any of the following sections: [JUNCTIONS], [RESERVOIRS], [TANKS], [PIPES], [PUMPS], or [VALVES]. For that reason, it is recommended to place these sections first just after the [TITLE] section. The sections corresponding to *"Network Map and Tags"* are not used in the Epanet simulations so they may or may not be in the file.

Each section may contain one or more lines of data, according to the number of elements that are added to our network model. Blank lines can appear anywhere in the file, and the semicolon (;) can be used to indicate that what follows is a comment and should not be interpreted as data. The maximum number of characters that a text line can have is 255. The tags "ID" are used to identify visible components, curves and patterns, they can consist of any combination of numbers and letters up to 31 characters.

2.2. Main sections for drawing a network

When we draw a water distribution network using the Epanet's graphical interface and then proceed to save it, the application automatically generates the labels (keywords found in brackets) that make up the input file which can have up to 29 sections.

If the supply network was originally drawn on AutoCAD, GIS or any other application that is not Epanet, another tool must be used to transfer the information stored in those formats, particularly the drawing of the network. Another choice is to program our own options that help to draw the pipes and nodes, the most numerous elements in a network, which is why it is enough for the input file to have just four main sections: [JUNCTIONS], [PIPES], [COORDINATES], [VERTICES]. If the pipes do not have vertices, the section [VERTICES] can be dispensed with. The input file would be as follows:

```
[JUNCTIONS]
;ID     Elev.    Demand        Pattern

[PIPES]
;ID    Node1    Node2    Length    Diameter    Roughness    MinorLoss    Status

[COORDINATES]
;Node    X-Coord    Y-Coord

[VERTICES]
;Link    X-Coord    Y-Coord
```

The rest of the remaining elements such as pumps, valves, tanks, and reservoirs, should they exist, can be added manually using the Epanet interface to complete the network model.

3 The Epanet Toolkit

3.1. Epanet's API

The Epanet Toolkit, which can be downloaded off of the official EPA website, is a set of files made up of an online-help file, a text file, a dynamic library and four header files that allow the functions in the dynamic library to be linked to the programming environment we are using. Using these functions, we can access simulation results and the information contained in the Epanet input file which is available in an inp format.

Epanet's API is a dynamic library, a .dll extension file which comes with the rest of the files that make up the Toolkit. This library is made up of a series of functions that permit programmers to personalize the Epanet engine module according to their own particular needs.

A bit of support can be found in the online-help file, included in the Toolbox, where a sequence of steps is given for using the functions, but for a user with little computer programming knowledge, using the dynamic library will be complicated given that he will not having all the lines of code necessary for its proper operation.

The Epanet Toolkit in the latest version (v2.00.12), is made up of seven files that are shown in Figure 3.1 and are described in Table 3.1.

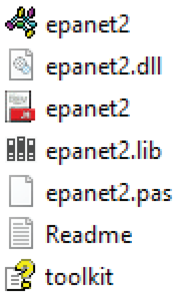


Figure 3.1. Files that make up the Epanet Toolkit

Table 3.1. Description of the files that make up the Epanet Toolkit

epanet2.bas	declarations module for using the Toolkit with Visual Basic 6.0
epanet2.dll	the Toolkit function library
epanet2.h	header file for using the Toolkit with C/C++
epanet2.lib	LIB file for using the Toolkit with Borland C/C++ 5.0 or Microsoft Visual C/C++ 6.0
epanet.pas	import unit for using the Toolkit with Delphi (Pascal)
Readme	describes the content of the toolkit
Toolkit.hlp	Windows Help file that describes how to use the Toolkit

The variables and functions used in each of these headers are for use in programming environments that run on a 32-bit Windows OS. If we want to use the Toolbox in more updated programming environments, we have to modify the data used for certain variables and functions as well as the Epanet library location that we are using.

The headers that contain variables and functions can be incorporated into programming environments that run on a 32-bit Windows OS such as C/C++, Delphi Pascal, Visual Basic, or any other language that allows for the use of functions included in a Windows DLL. The .dll file included in the Toolbox is called epanet2.dll and is distributed jointly with the Epanet application. It contains 55 functions that allow for: the opening of a data file, the reading and modifying distinct design and network operation parameters, running extended period simulations, retrieving results, saving these to file, or writing results to a text file in a given format.

In the help file, also included in the Toolkit, the proper manner of using these functions in different programming environments is briefly described using some basic examples.

The toolkit is useful for developing customized applications such as optimization or automatic calibration of models in which multiple analyses are required according to the values that adopt certain input parameters within an iterative process. Moreover, the dynamic library (epanet2.dll) permits its functions to be added to other integrated environments based on CAD, GIS or databases.

3.2. API functions

The Epanet (v2.00.12) dynamic library, also called API (Aplication Programming Interface), is written in the ANSI C language and has separate code modules for input data processing, hydraulic analysis, water-quality analysis, solving systems of linear equations with sparse matrices, and report generation. It has 55 functions and 104 variables that are used with functions such as input parameters. The functions arranged according to their task are shown on Table 3.2.

Table 3.2. Epanet Toolkit functions by task

Task	Functions	
Running a complete "command line style" simulation	ENepanet	
Opening and closing the EPANET Toolkit system	ENopen	ENclose
Retrieving information about network nodes	ENgetnodeindex ENgetnodeid	ENgetnodetype ENgetnodevalue
Retrieving information about network links	ENgetlinkindex ENgetlinkvalue ENgetlinktype	ENgetlinknodes ENgetlinkid
Retrieving information about time patterns	ENgetpatternid ENgetpatternindex	ENgetpatternlen ENgetpatternvalue
Retrieving other network information	ENgetcontrol ENgetqualtype ENgetoption ENgetversion	ENgetcount ENgetflowunits ENgettimeparam
Setting new values for network parameters	ENsetcontrol ENsetnodevalue ENsetlinkvalue ENaddpattern ENsetpattern	ENserpatternvalue ENsetqualtype ENsettimeparam ENsetoption
Saving and using hydraulic analysis results files	ENsavehydfile	ENusehyfile
Running a hydraulic analysis	ENsolveH ENopenH ENinitH	ENrunH ENnextH ENcloseH

Table 3.2 Epanet Toolkit functions by task (Continuation)

Task	Functions	
Running a water quality analysis	ENsolveQ ENopenQ ENinitQ ENrunQ	ENnextQ ENstepQ ENcloseQ
Generating an output report	ENsaveH ENsaveinpfile ENreport ENresetreport	ENsetreport ENsetstatusreport ENgeterror ENwriteline

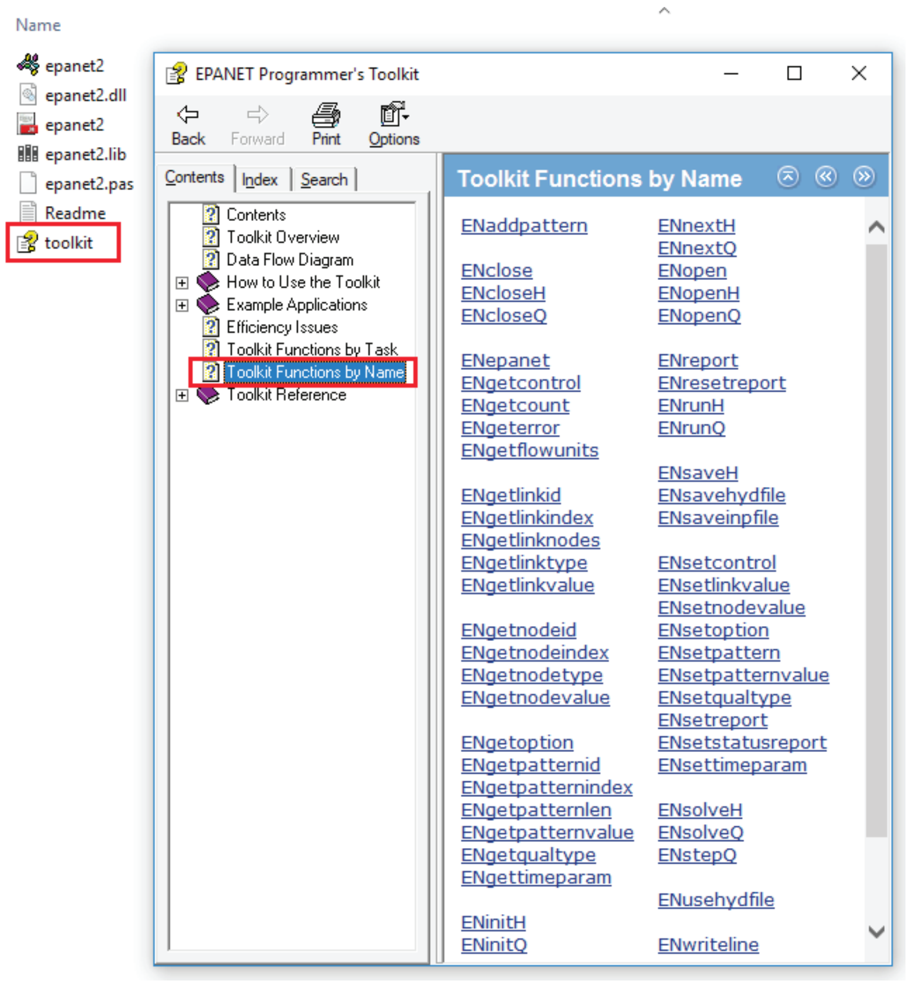
The procedure for working with these functions is as follows:

1. Use the **ENopen** function to open the Epanet library, which will enable the rest of the functions, except the ENepanet function.
2. Use the **ENsetxxx** functions to change the properties of elements or characteristics of the network or use the **ENgetxxx** functions to retrieve data from the network saved in the input file (file with inp extension).
3. For a complete simulation use the **ENsolveH** function (this function automatically saves the results in a hydraulic-results file), or if a step-by-step simulation is desired use the following sequence of functions: **ENopenH**, **ENinitH**, **ENrunH**, **ENnextH**, **ENcloseH**, and access the results through the ENgetxxx function series.
4. The **ENsolveQ** function runs a complete simulation of a water quality model (this function automatically saves the hydraulic and water quality results in an output file), or if running the simulation step-by-step, use the following functions sequence: **ENopenQ**, **ENinitQ**, **ENrunQ**, **ENnextQ**, **ENcloseQ**, and access the results through the ENgetxxx function series.
5. If running new analyses is desired, return to step 2 or use the **ENreport** function to save a results report formatted in the results report file.
6. Call the **ENclose** function to close the Epanet file that was opened initially, this frees up the memory that was occupied on the computer.

To sum up, with the Epanet API we can do the following tasks:

1. Open and close the Toolkit
2. Retrieve and set network parameters
3. Run a hydraulic simulation
4. Run a water quality simulation
5. Retrieve results
6. Generate a results report

To know what each of the functions does, it is recommended to open the help file that also comes included in the Toolkit and click on the section "Toolkit Functions by Name", where all functions in the Epanet library can be seen in a drop-down menu. Figure 3.2 shows the help file with the Epanet functions.



Toolkit Functions by Name

ENaddpattern
ENclose
ENcloseH
ENcloseQ
ENepanet
ENgetcontrol
ENgetcount
ENgeterror
ENgetflowunits
ENgetlinkid
ENgetlinkindex
ENgetlinknodes
ENgetlinktype
ENgetlinkvalue
ENgetnodeid
ENgetnodeindex
ENgetnodetype
ENgetnodevalue
ENgetoption
ENgetpatternid
ENgetpatternindex
ENgetpatternlen
ENgetpatternvalue
ENgetqualtype
ENgettimeparam
ENinitH
ENinitQ
ENnextH
ENnextQ
ENopen
ENopenH
ENopenQ
ENreport
ENresetreport
ENrunH
ENrunQ
ENsaveH
ENsavehydfile
ENsaveinpfile
ENsetcontrol
ENsetlinkvalue
ENsetnodevalue
ENsetoption
ENsetpattern
ENsetpatternvalue
ENsetqualtype
ENsetreport
ENsetstatusreport
ENsettimeparam
ENsolveH
ENsolveQ
ENstepQ
ENusehydfile
ENwriteline

Figure 3.2. Online help with the Epanet Toolkit for programmers

4 Connecting Epanet's API to programming environments

The dynamic link library (epanet2.dll), in its latest version 2.00.12, launched in March 2008, was written in the C programming language. Moreover, it was compiled for use in Windows 32-bit OS programming environments such as Borland C/C++ 5.0, Delphi Pascal 5.0, Visual Basic 6.0, or any other environment that allows functions to be incorporated into a Windows DLL.

The Toolbox includes header files where variables and functions are declared that will allow us to interact with the Epanet library in order to obtain or modify information of the network model being studied.

In order to use the Epanet dynamic library (epanet2.dll) in the new 64-bits programming environments, it will be necessary to generate a new header file with modifications to the types of data used by the variables, the functions and their input parameters, according to the programming language being used.

The purpose of this chapter is to learn how to use the Epanet dynamic library with the most prevalent Toolbox items in a variety of languages so as to allow the user to choose the language that best suits him. The most-used programming languages for working with the Epanet Toolkit are Basic 6.0, Basic .NET, the technical computing language (Matlab), C#, Python, and C++. The programming environments that support the aforementioned languages will be installed and run on a Windows 10 64-bit OS.

There is no problem installing and running 32-bit applications on a Windows 10 64-bit OS. The steps to follow for each programming environment will be given in order to connect the Epanet's API, along with a few simple exercises to make sure that we can access the stored information in an Epanet inp file through the dynamic library (epanet2.dll). Before using the Epanet library, it is recommended to have Epanet 2.00.12 installed, which can be downloaded from the following link: https://www.epa.gov/water-research/epanet.

The network model that will be worked on is the Net3.inp water distribution network which comes with the Epanet application installer. The exercise consists of retrieving the number of nodes (the sum of the number of tanks, reservoirs and demand nodes), number of links (the sum of pipes, valves, and pumps), and the flow unit. More examples are given in Chapter 5, where the majority of functions available in the Epanet's library are used.

4.1. Visual Basic 6.0 (Basic 6.0)

Visual Basic 6.0 is a programming language and an Integrated Development Environment (IDE). It is derived from the oldest version of the BASIC language and as such, is considered a useful language in that it is relatively easy to use for beginners. Visual Basic 6.0 is the latest edition of Visual Basic, it is a programming language driven by events. The latest version published in 1998 stopped being supported by Microsoft in 2008, but in spite of this, the 32-bit versions that it generates are compatible with the most modern platforms such as Windows Vista, Windows Server 2008, Windows 7, Windows 8 and Windows 10. Visual Basic contains an IDE that has a text editor for editing the source code, a debugger, a compiler, and a graphical-interface editor.

Although Basic 6.0, as a native language has ceased to be supported by Microsoft, its syntax remains in use for the VBA language (Visual Basic for Applications) and Microsoft remains the owner of other popular applications such as Excel on up to the latest version of Office 2016.

Whenever installing Visual Basic 6.0 in any of its versions on a 64-bit Windows OS, an installation-error message will appear, blocking the process. This is due to the Visual Basic 6.0 installer setup.exe being compiled in 16 bits. On a 64-bit Windows OS, 32-bit applications run without problems but cannot execute 16-bit ones. On the Internet we can find many aids such as blogs and video tutorials which tell us how we can continue using this application on a 64-bit Windows OS.

Once our application is installed on a 32/64-bit Windows OS, we have to follow a series of steps in order to start using the functions in the Epanet library (epanet2.dll). These steps are as follows:

A. Create a project folder.

It is recommended to create this on a disc unit that is not a root directory (C:), this will prevent future reading and writing problems. In this folder, the epanet2.dll and epanet2.bas files will be found. Both files are found in the compressed file that can be downloaded from the following link: www.epa.gov/water-research/epanet. One possible route could be: D:\ConnectAPIEpanet\VB6.0\, where the aforementioned files will be stored in the VB6.0 folder.

B. Adding the epanet2.bas file to a new project

The following step is to open the Visual Basic 6.0 application and load epanet2.bas. For that we need to go to the project menu and select the "Add module" option. We can see a form where we will click on the "existing" tab and proceed to navigate through the directories until finding the epanet2.bas and saving it in the VB6.0 folder. We change the name of the module "Module1" to "Epanet2.bas" as shown in Figure 4.1.

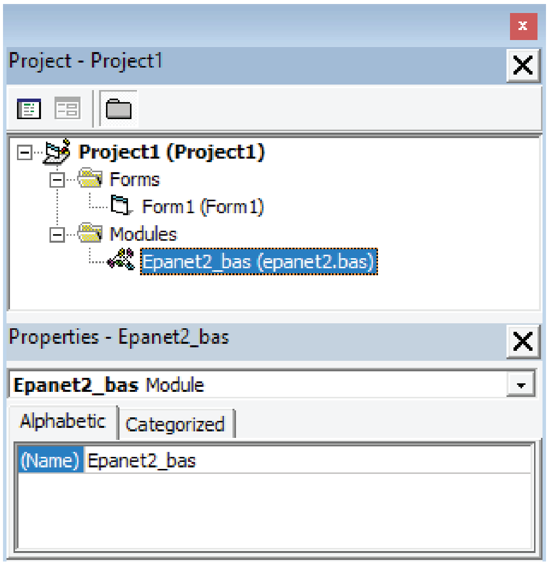
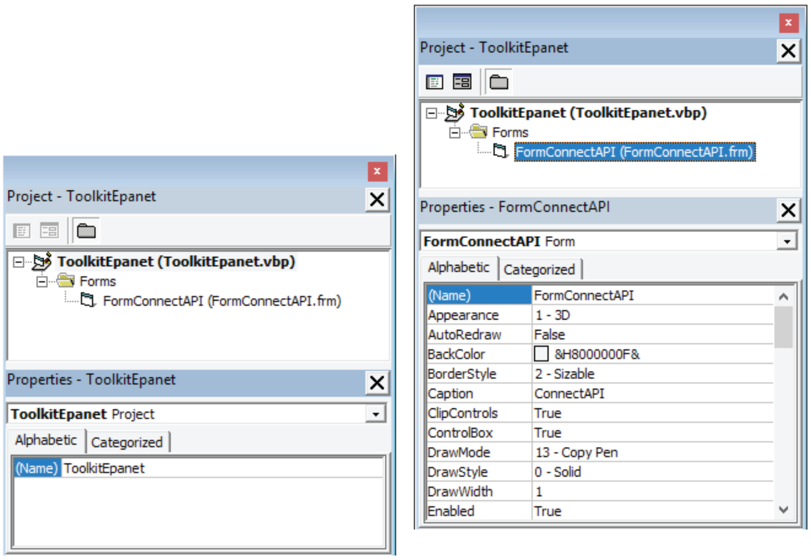


Figure 4.1. Loading the epanet2.bas file into a project

Afterwards, the names of our project and form will be changed. By default, the name of the project is "Project1" and the form "Form1". The new name of the project will be "EpanetToolkit" and the form "FormConnectAPI". Each form that is created must be given a name and a title (Caption), which in our case will be "ConnectAPI". The new changes can be seen in Figure 4.2.



Project - ToolkitEpanet

```
ToolkitEpanet (ToolkitEpanet.vbp)
  Forms
    FormConnectAPI (FormConnectAPI.frm)
```

Properties - ToolkitEpanet

ToolkitEpanet Project

Alphabetic | Categorized

(Name)	ToolkitEpanet

Project - ToolkitEpanet

```
ToolkitEpanet (ToolkitEpanet.vbp)
  Forms
    FormConnectAPI (FormConnectAPI.frm)
```

Properties - FormConnectAPI

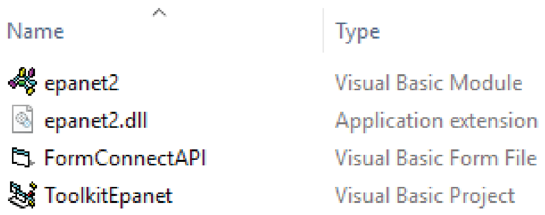
FormConnectAPI Form

Alphabetic | Categorized

(Name)	FormConnectAPI
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F&
BorderStyle	2 - Sizable
Caption	ConnectAPI
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True

Figure 4.2. Name changes to the project name and form

With these changes, we can save our project and form in the "VB6.0" project folder. There should be four files in this folder, see Figure 4.3.



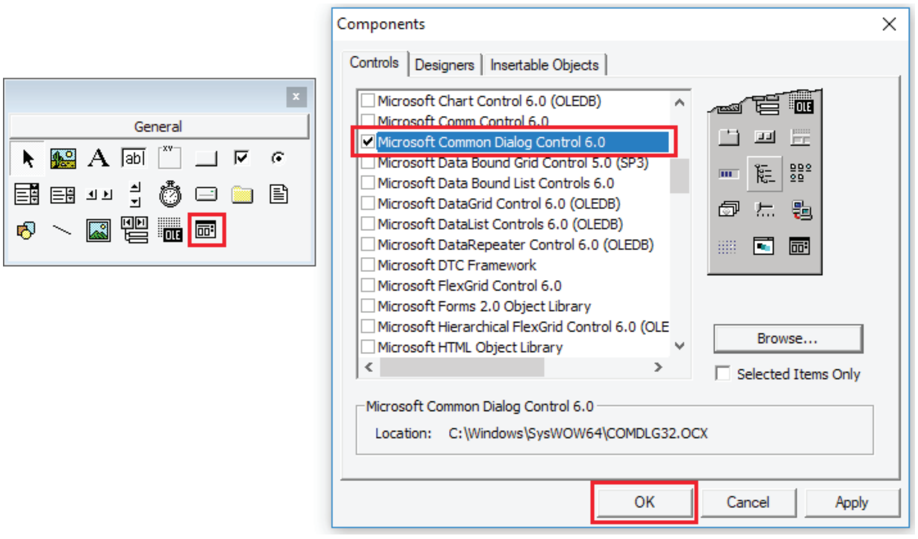
Name	Type
epanet2	Visual Basic Module
epanet2.dll	Application extension
FormConnectAPI	Visual Basic Form File
ToolkitEpanet	Visual Basic Project

Figure 4.3. Files saved in the projects folder VB6.0

Now we are ready to begin using the functions found in the Epanet dynamic library. The following is to prepare a form where our inp Epanet file path can be viewed, as well as the number of links, nodes and the flow unit.

C. Preparing the form

Our form will have the following controls: 01 Frame, 04 Label, 02 CommandButton, 04 Textbox, 01 CommonDialog. This last control does not appear in the toolkit when beginning a new project. To add this to the toolbox it is necessary to go to Project Menu, then to select the Components option and afterwards, click on the Controls tab and look for the Microsoft Common Dialog Control 6.0, and mark it before finally clicking on Accept. This control is not visible in run time mode, only in design mode. The CommonDialog control added can be viewed on Figure 4.4.



- [ ] Microsoft Chart Control 6.0 (OLEDB)
- [ ] Microsoft Comm Control 6.0
- [✓] Microsoft Common Dialog Control 6.0
- [ ] Microsoft Data Bound Grid Control 5.0 (SP3)
- [ ] Microsoft Data Bound List Controls 6.0
- [ ] Microsoft DataGrid Control 6.0 (OLEDB)
- [ ] Microsoft DataList Controls 6.0 (OLEDB)
- [ ] Microsoft DataRepeater Control 6.0 (OLEDB)
- [ ] Microsoft DTC Framework
- [ ] Microsoft FlexGrid Control 6.0
- [ ] Microsoft Forms 2.0 Object Library
- [ ] Microsoft Hierarchical FlexGrid Control 6.0 (OLE
- [ ] Microsoft HTML Object Library

- [ ] Selected Items Only

Microsoft Common Dialog Control 6.0
Location: C:\Windows\SysWOW64\COMDLG32.OCX

Figure 4.4. CommonDialog Control added to the VB6.0 Toolkit

In Table 4.1, we can see the original names of the controls added to the form and the changed names along with the content that these will have in the Caption and Text properties.

Table 4.1. Controls added to the main form (Visual Basic)

Property: Name [Original]	Property: Name [Changed]	Property
Frame1	FrameContainer	Caption: empty
Label1	LblINP	Caption: Select Epanet Inp file
Label2	LblNumL	Caption: # of links
Label3	LblNumN	Caption: # of nodes
Label4	LblFlowUnt	Caption: Flow Unit
TextBox1	TxtINP	Caption: empty
TextBox2	TxtLinks	Caption: empty
TextBox3	TxtNodes	Caption: empty
TextBox4	TxtFlowUnt	Caption: empty
CommandButton1	CmdOpen	Caption: ...
CommandButton2	CmdAccept	Caption: Accept
CommonDialog	CDialog	----

We can see the final design that our form will have in Figure 4.5, in design mode as well as in run mode.

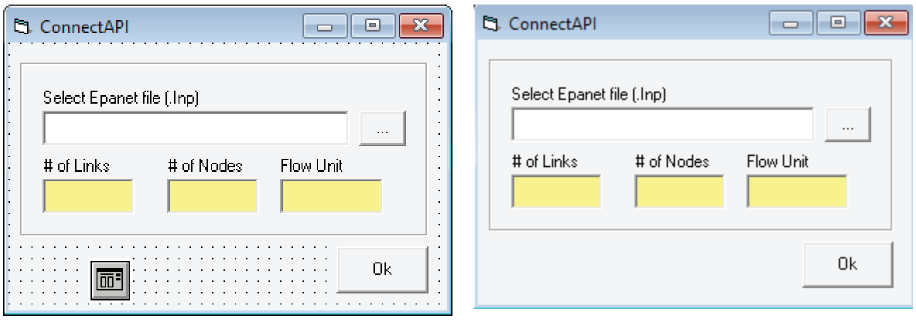
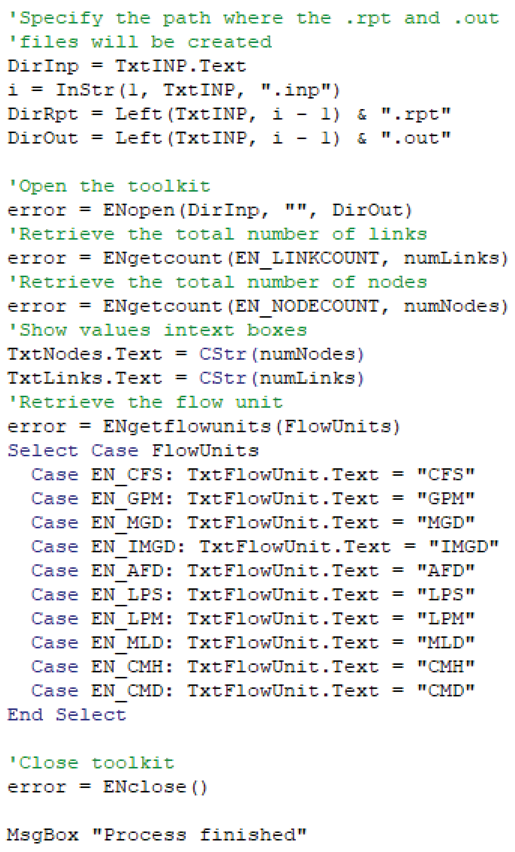


Figure 4.5. Form in design and run time modes

D. Working with the Epanet Toolkit

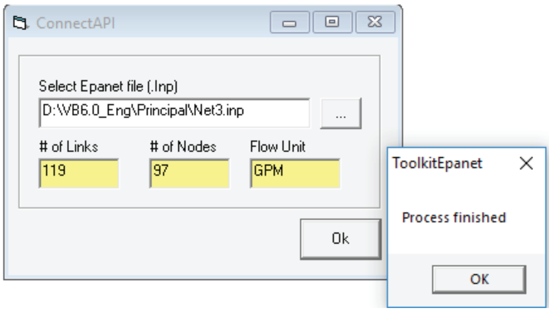
To run a sequence of lines of code, it must be associated to some event corresponding to objects inserted in the form. In our case, we have to associate code with the CmdOpen and CmdAccept objects, and for that, we have but to double click on each object and the text editor will appear wherein we will write our lines of code. In Figure 4.6, part of the code sentences associated with the CmdAccept button is shown. To download the complete code click on the following link: https://www.imta.gob.mx/biblioteca/download/?key=244.



```
'Specify the path where the .rpt and .out
'files will be created
DirInp = TxtINP.Text
i = InStr(1, TxtINP, ".inp")
DirRpt = Left(TxtINP, i - 1) & ".rpt"
DirOut = Left(TxtINP, i - 1) & ".out"

'Open the toolkit
error = ENopen(DirInp, "", DirOut)
'Retrieve the total number of links
error = ENgetcount(EN_LINKCOUNT, numLinks)
'Retrieve the total number of nodes
error = ENgetcount(EN_NODECOUNT, numNodes)
'Show values intext boxes
TxtNodes.Text = CStr(numNodes)
TxtLinks.Text = CStr(numLinks)
'Retrieve the flow unit
error = ENgetflowunits(FlowUnits)
Select Case FlowUnits
  Case EN_CFS: TxtFlowUnit.Text = "CFS"
  Case EN_GPM: TxtFlowUnit.Text = "GPM"
  Case EN_MGD: TxtFlowUnit.Text = "MGD"
  Case EN_IMGD: TxtFlowUnit.Text = "IMGD"
  Case EN_AFD: TxtFlowUnit.Text = "AFD"
  Case EN_LPS: TxtFlowUnit.Text = "LPS"
  Case EN_LPM: TxtFlowUnit.Text = "LPM"
  Case EN_MLD: TxtFlowUnit.Text = "MLD"
  Case EN_CMH: TxtFlowUnit.Text = "CMH"
  Case EN_CMD: TxtFlowUnit.Text = "CMD"
End Select

'Close toolkit
error = ENclose()

MsgBox "Process finished"
```

Figure 4.6. Source code associated to the CmdAccept button

Two events are coded in the application. In the first, it is necessary to click on the CmdOpen button to select the Epanet inp file, Net3.inp, and show its complete path in the text box next to the button. The second event is to click on the CmdAccept button so that it can show us the total number of nodes (the sum of demand nodes, reservoirs and tanks), the total number of links (the sum of pipes, valves and pumps), and the flow unit. Figure 4.7 shows the result of said query and Figure 4.8 the summary of the numbers of elements on Epanet.

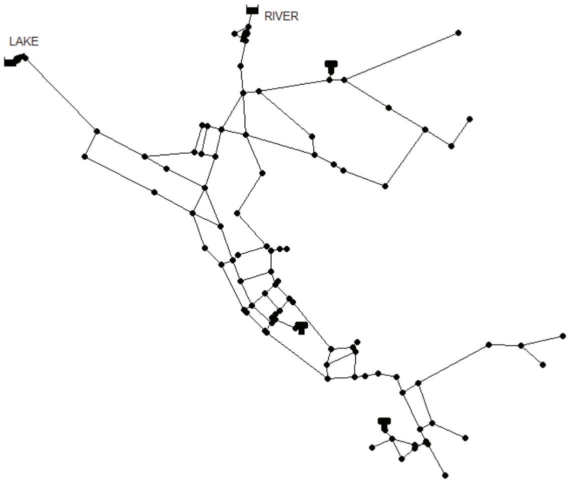


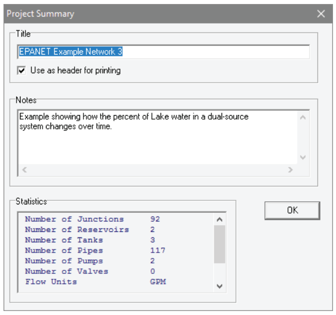
Select Epanet file (.Inp)
D:\VB6.0_Eng\Principal\Net3.inp

# of Links 119
# of Nodes 97
Flow Unit GPM

Process finished

Figure 4.7. Result from consulting number of links and nodes in the Net3 network





Title
EPANET Example Network 3
☑ Use as header for printing

Notes
Example showing how the percent of Lake water in a dual-source system changes over time.

Statistics
```
Number of Junctions     92
Number of Reservoirs    2
Number of Tanks         3
Number of Pipes         117
Number of Pumps         2
Number of Valves        0
Flow Units              GPM
```

Figure 4.8. Number of nodes and links in the Net3 network using Epanet

The functions that have been used to consult the information required by the Epanet Toolkit are: ENopen, ENgetcount, ENgetflowunits, y ENclose. If we want to consult the headloss formula we will not be able use the Epanet Toolkit because there is no programmed function for that task in the library. To obtain it we have to use the Visual Basic functions and scan through the inp file until finding said parameter in the [OPTIONS] section.

4.2. Visual Studio 2017 (Visual Basic .NET)

In 2001 Microsoft proposed leaving the API Win32 based development to migrate to a common library framework, known as .NET Framework, irrespective of the OS version, to give support to different programming languages such as Basic.NET, C#, etc, aiding the translation of code between them; this was the Visual Basic 6's successor.

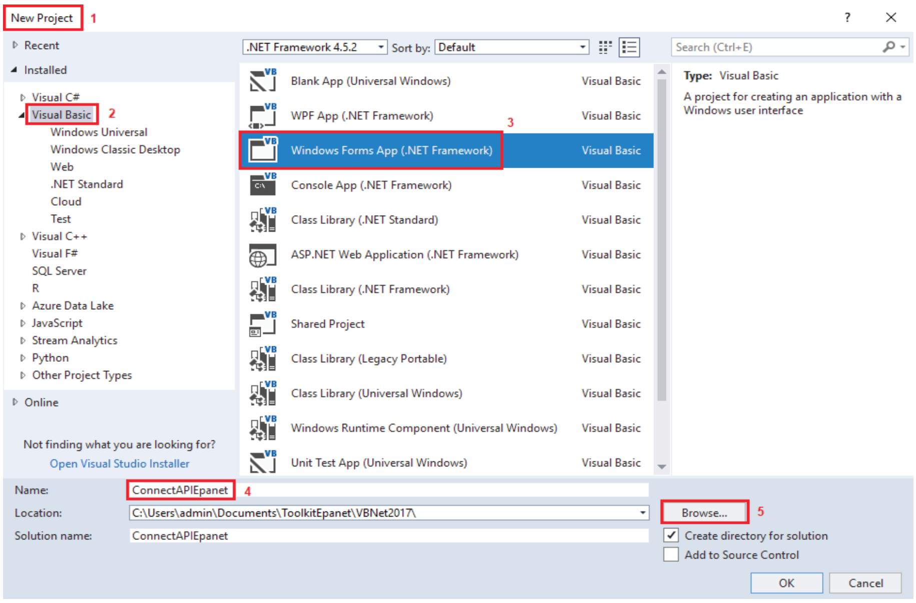
The Basic language evolved to integrate the .NET platform; there it lost its identity as a unique language, and became part of a product package called Microsoft .NET; within this package or framework, this language can be found now renamed Visual Basic .NET, which runs on Microsoft's Visual Studio environment. This new version of the language possesses deep differences in programming methods compared to Visual Basic 6, but maintains strong similarities in its basic syntax.

The following is an explanation of how to connect the Epanet API (32 bit version 2.00.12) to the new Visual Studio 2017 programming environment, along with a how-to guide for transforming the epanet2.bas file into a new module file (Epanet2.vb) compatible with the new programming environment (declaration of variable data types, functions and input parameters).

Once Visual Studio 2017 is installed, we prepare our work environment in order to use the epanet2.dll file functions. The same network model (Net3.inp) will be used and the total number of nodes, links and flow unit will be retrieved. The sequence of steps is as follows:

A. Create a new project

We open Visual Studio 2017 and click on New Project, a window will appear in which we will choose a programming language to use (Visual Basic) and then we select the option of creating an application with a Windows user interface (Windows Forms Application). Next, we write a name for our project, put (ConnectEpanetAPI) in the solution, place it in the desired directory and finisih by clicking on the Accept button. Figure 4.9 summarizes the previous steps.


New Project 1
Recent
Installed
Visual C#
Visual Basic 2
Windows Universal
Windows Classic Desktop
Web
.NET Standard
Cloud
Test
Visual C++
Visual F#
SQL Server
R
Azure Data Lake
JavaScript
Stream Analytics
Python
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.5.2
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual Basic
WPF App (.NET Framework) Visual Basic
3
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Framework) Visual Basic
Class Library (.NET Standard) Visual Basic
ASP.NET Web Application (.NET Framework) Visual Basic
Class Library (.NET Framework) Visual Basic
Shared Project Visual Basic
Class Library (Legacy Portable) Visual Basic
Class Library (Universal Windows) Visual Basic
Windows Runtime Component (Universal Windows) Visual Basic
Unit Test App (Universal Windows) Visual Basic
Browse... 5
OK
Cancel

Type: Visual Basic
A project for creating an application with a Windows user interface

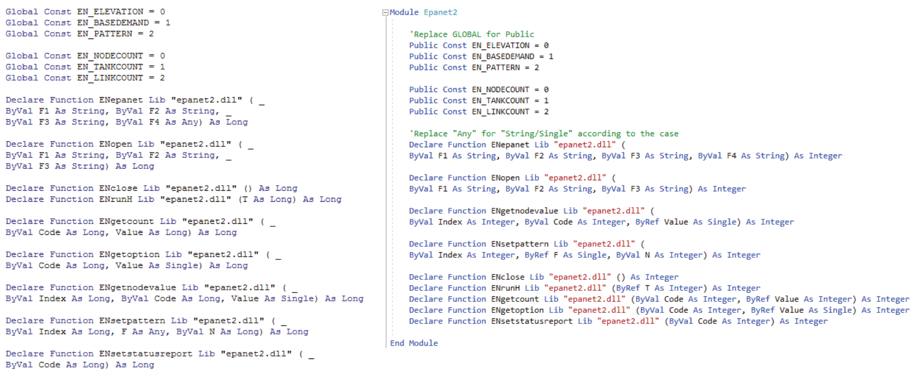
Name: ConnectAPIEpanet 4
Location: C:\Users\admin\Documents\ToolkitEpanet\VBNet2017\
Solution name: ConnectAPIEpanet
[✓] Create directory for solution
[ ] Add to Source Control

Figure 4.9. Creating a new project in Visual Studio 2017

After creating the project, an empty form will appear in which we can add the objects (Controls) needed to design our own tools. We will not yet jump ahead to that, however, first we have to prepare our new base module (Epanet2.vb) where global variables and functions that interact with the Epanet library (epanet2.dll) are declared.

B. Adding the Epanet2.vb to a new project

The content of the epanet2.bas file, that is used in Visual Basic 6.0, should be copied to a new Visual Studio 2017 base module file (Epanet2.vb) and the declarations of data types, global variables, functions and their parameters modified. Part of the changes carried out on the epanet2.bas module that is converted into Epanet2.vb are shown in Figure 4.10 for comparison. In the following link: https://www.imta.gob.mx/biblioteca/download/?key=266 the Epanet2.vb can be downloaded for direct use in our project just like in Figure 4.11.



```
Global Const EN_ELEVATION = 0
Global Const EN_BASEDEMAND = 1
Global Const EN_PATTERN = 2

Global Const EN_NODECOUNT = 0
Global Const EN_TANKCOUNT = 1
Global Const EN_LINKCOUNT = 2

Declare Function ENepanet Lib "epanet2.dll" ( _
ByVal F1 As String, ByVal F2 As String, _
ByVal F3 As String, ByVal F4 As Any) As Long

Declare Function ENopen Lib "epanet2.dll" ( _
ByVal F1 As String, ByVal F2 As String, _
ByVal F3 As String) As Long

Declare Function ENclose Lib "epanet2.dll" () As Long
Declare Function ENrunH Lib "epanet2.dll" (T As Long) As Long

Declare Function ENgetcount Lib "epanet2.dll" ( _
ByVal Code As Long, Value As Long) As Long

Declare Function ENgetoption Lib "epanet2.dll" ( _
ByVal Code As Long, Value As Single) As Long

Declare Function ENgetnodevalue Lib "epanet2.dll" ( _
ByVal Index As Long, ByVal Code As Long, Value As Single) As Long

Declare Function ENsetpattern Lib "epanet2.dll" ( _
ByVal Index As Long, F As Any, ByVal N As Long) As Long

Declare Function ENsetstatusreport Lib "epanet2.dll" ( _
ByVal Code As Long) As Long
```

```
Module Epanet2

    'Replace GLOBAL for Public
    Public Const EN_ELEVATION = 0
    Public Const EN_BASEDEMAND = 1
    Public Const EN_PATTERN = 2

    Public Const EN_NODECOUNT = 0
    Public Const EN_TANKCOUNT = 1
    Public Const EN_LINKCOUNT = 2

    'Replace "Any" for "String/Single" according to the case
    Declare Function ENepanet Lib "epanet2.dll" (
    ByVal F1 As String, ByVal F2 As String, ByVal F3 As String, ByVal F4 As String) As Integer

    Declare Function ENopen Lib "epanet2.dll" (
    ByVal F1 As String, ByVal F2 As String, ByVal F3 As String) As Integer

    Declare Function ENgetnodevalue Lib "epanet2.dll" (
    ByVal Index As Integer, ByVal Code As Integer, ByRef Value As Single) As Integer

    Declare Function ENsetpattern Lib "epanet2.dll" (
    ByVal Index As Integer, ByRef F As Single, ByVal N As Integer) As Integer

    Declare Function ENclose Lib "epanet2.dll" () As Integer
    Declare Function ENrunH Lib "epanet2.dll" (ByRef T As Integer) As Integer
    Declare Function ENgetcount Lib "epanet2.dll" (ByVal Code As Integer, ByRef Value As Integer) As Integer
    Declare Function ENgetoption Lib "epanet2.dll" (ByVal Code As Integer, ByRef Value As Single) As Integer
    Declare Function ENsetstatusreport Lib "epanet2.dll" (ByVal Code As Integer) As Integer

End Module
```

Figure 4.10. Declaration of the data types, variables and functions on VB6.0 and VB.NET

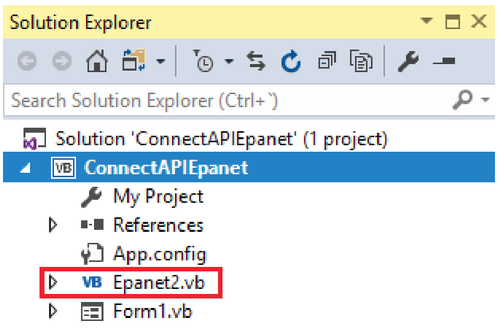


Figure 4.11. Epanet2.vb module added to project

C. Copying the epanet2.dll file in the Debug folder

After adding the base module, we need to copy the Epanet library (epanet2.dll) inside the "Debug" folder that is found inside the "bin" folder. For that we have to click on the "Show All" button to show all the files from the solution explorer and view the "bin" and "obj" folders as shown in Figure 4.12.

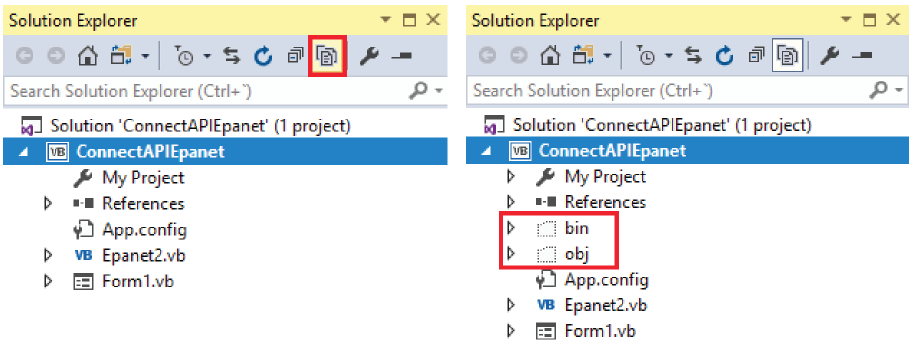


Figure 4.12. Viewing the bin and obj folders on the VB.NET solution explorer

Upon opening the "Debug" folder it is possible to verify that the library has been copied correctly (Figure 4.13). Finally we proceed to save all the changes that were made.

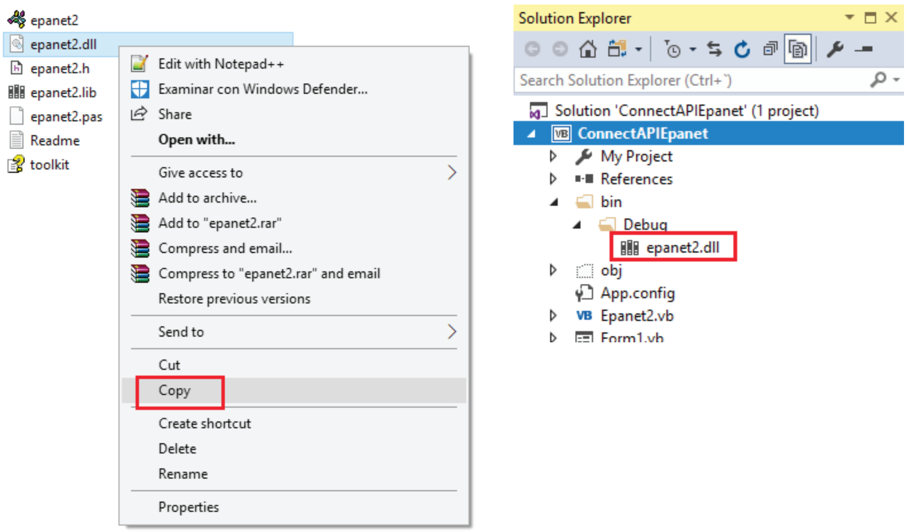


Figure 4.13. Copying the epanet2.dll file to the Debug folder

After these steps we can add the controls to our form, the same ones that were used for Visual Basic 6.0 will be inserted here.

D. Preparing the form

The controls that will be added to the form are as follows: 01 GroupBox, 04 Label, 02 Button, 04 Textbox, 01 OpenFileDialog. This last control is visible in design mode but not in run time mode. All these controls can be found in the toolbox as can be seen in Figure 4.14. If the toolbox is not visible, we can make it appear by pressing Ctrl+Alt+X or clicking on the View menu and in the dropdown clicking on the Toolbox.

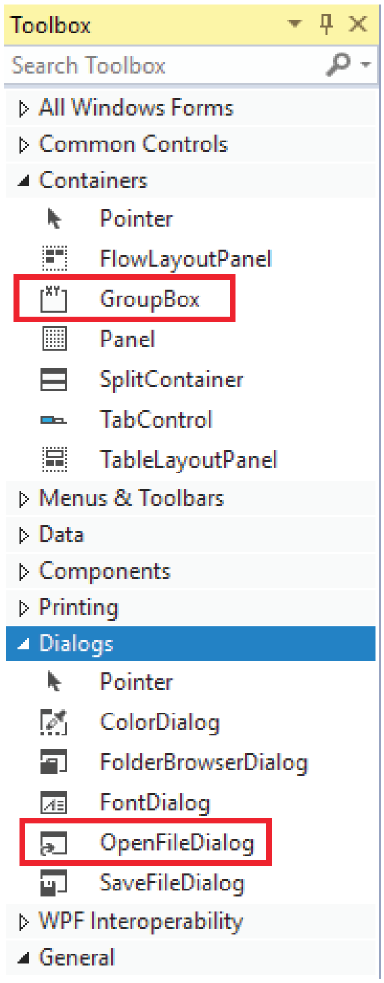


Figure 4.14. Visual Basic 2017 toolkit

Table 4.2 shows the original control names added to the form, the changed names as well as the content that they'll have in the Caption and Text properties.

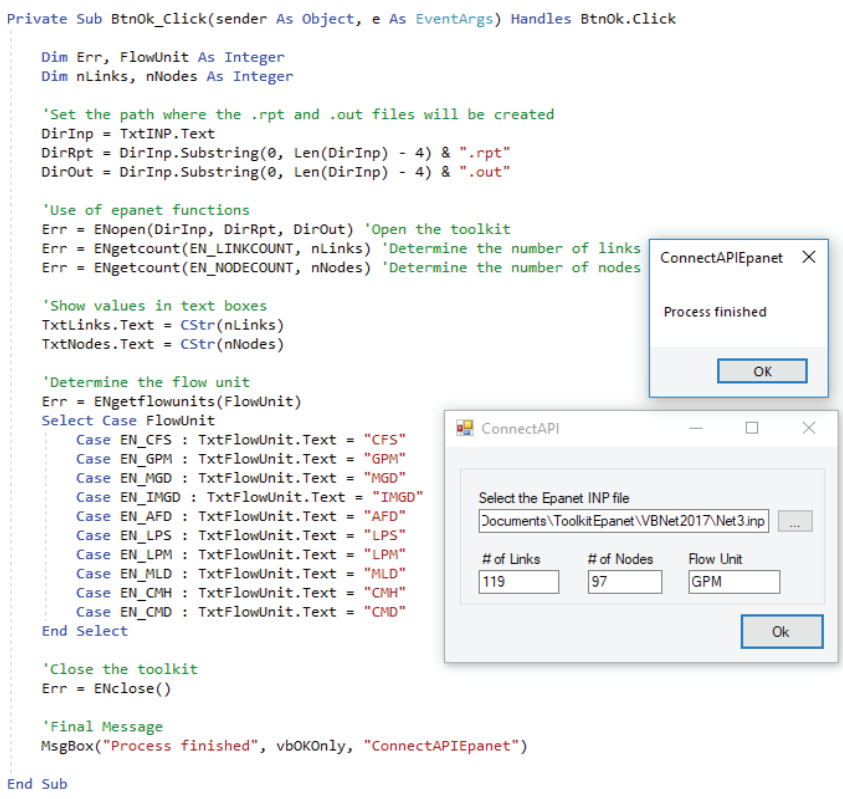
Table 4.2. Controls added to the main form (Visual Basic.NET)

Property: Name [Original]	Property: Name [Changed]	Property
GroupBox1	GBContainer	Text: empty
Label1	LblINP	Text: Select the Epanet Inp file
Label2	LblNumLinks	Text: # of Links
Label3	LblNumNodes	Text: # de Nodes
Label4	LblFlowUnt	Text: Flow Unit
TextBox1	TxtINP	Text: empty
TextBox2	TxtLinks	Text: empty
TextBox3	TxtNodes	Text: empty
TextBox4	TxtFlowUnt	Text: empty
Button1	BtnOpen	Text: ...
Button2	BtnOk	Text: Accept
OpenFileDialog	OpenFD	FileName: empty

E. Working with the Epanet Toolkit

Just like in Visual Basic 6.0, it is necessary to associate a group of sentences to some controls. In this case they are the same buttons, OpenBtn and AcceptBtn, to which lines of code are associated. In Figure 4.15 we can see part of the source code used to fulfill our objective and the final form in run time mode. You can go to the following link to download and view all of the source code: https://www.imta.gob.mx/biblioteca/download/?key=245.

The functions used are: ENopen, ENgetcount, ENgetflowunits, and ENclose. If we want to consult the headloss formula used, it won't be possible with Epanet's Toolkit because there is no such function in that library. To obtain it, it is necessary to use the Visual Basic.Net's own functions to browse through the inp file until finding that parameter in the [OPTIONS] section.



```
Private Sub BtnOk_Click(sender As Object, e As EventArgs) Handles BtnOk.Click

    Dim Err, FlowUnit As Integer
    Dim nLinks, nNodes As Integer

    'Set the path where the .rpt and .out files will be created
    DirInp = TxtINP.Text
    DirRpt = DirInp.Substring(0, Len(DirInp) - 4) & ".rpt"
    DirOut = DirInp.Substring(0, Len(DirInp) - 4) & ".out"

    'Use of epanet functions
    Err = ENopen(DirInp, DirRpt, DirOut) 'Open the toolkit
    Err = ENgetcount(EN_LINKCOUNT, nLinks) 'Determine the number of links
    Err = ENgetcount(EN_NODECOUNT, nNodes) 'Determine the number of nodes

    'Show values in text boxes
    TxtLinks.Text = CStr(nLinks)
    TxtNodes.Text = CStr(nNodes)

    'Determine the flow unit
    Err = ENgetflowunits(FlowUnit)
    Select Case FlowUnit
        Case EN_CFS : TxtFlowUnit.Text = "CFS"
        Case EN_GPM : TxtFlowUnit.Text = "GPM"
        Case EN_MGD : TxtFlowUnit.Text = "MGD"
        Case EN_IMGD : TxtFlowUnit.Text = "IMGD"
        Case EN_AFD : TxtFlowUnit.Text = "AFD"
        Case EN_LPS : TxtFlowUnit.Text = "LPS"
        Case EN_LPM : TxtFlowUnit.Text = "LPM"
        Case EN_MLD : TxtFlowUnit.Text = "MLD"
        Case EN_CMH : TxtFlowUnit.Text = "CMH"
        Case EN_CMD : TxtFlowUnit.Text = "CMD"
    End Select

    'Close the toolkit
    Err = ENclose()

    'Final Message
    MsgBox("Process finished", vbOKOnly, "ConnectAPIEpanet")

End Sub
```

ConnectAPIEpanet

Process finished

OK

ConnectAPI

Select the Epanet INP file: Documents\ToolkitEpanet\VBNet2017\Net3.inp ...

# of Links	# of Nodes	Flow Unit
119	97	GPM

Ok

Figure 4.15. Part of the source code and main form in run time mode

4.3. Matlab (technical computing language)

The first version of MATLAB goes back to the 70s, it was designed as a support tool for the *Theory of Matrices, Linear Algebra and Numerical Analysis* courses by the mathematician Cleve Moler. The name is an acronym: "MATrix LABoratory". It is mathematical software that offers an Integrated Development Environment with its own programming language (M language) that allows for the performing of operations with vectors, matrices and functions, lambda calculus and programming oriented towards objects, as well as communication with programs written in other languages and with other hardware devices. Moreover, it has "Simulink" and "Guide", two additional tools that further increase the number of features found in MATLAB[3]. Originally, it was written in Fortran over the course of several years, but currently it is written in C by "The Mathworks".

MATLAB is a programming and computing system based on the manipulation of matrices. In this way, matrix algebra and other time-saving properties can be used. It integrates numerical analysis, matrix calculus, signal processing, graphs, etc., within an environment that is easy to use, and problems and solutions are expressed just as they are when written mathematically, without the need to resort to using traditional, lower-level programming.

MATLAB is very quick for certain operations when it can run its functions in native code in sizes adequate for taking advantage of its vectorization capacities. In other applications it is significantly slower than the equivalent code developed in C/C++ or Visual .NET. All the same, it is a very useful and high level tool for developing technical applications that are easy to use and can help to greatly increase a programmers' productivity compared to other development environments.

The programming in MATLAB is done through a language that is very similar to high-level languages such as BASIC or C. This allows the user to group sentences that are frequently used in the program and can be invoked later. Time and effort are saved this way in successive sessions, since it is no longer necessary to write all the sentences over again.

MATLAB has a basic code and several specialized libraries (toolboxes). Moreover, it includes a great amount of predefined functions that help to perform typical calculations, as well as to view data and results. Code lines written in an ASCII file (with an *.m extension) can be run, if said file is in another subdirectory indicated in the PATH or the current work directory.

[3] https://www.mathworks.com/products/matlab.html

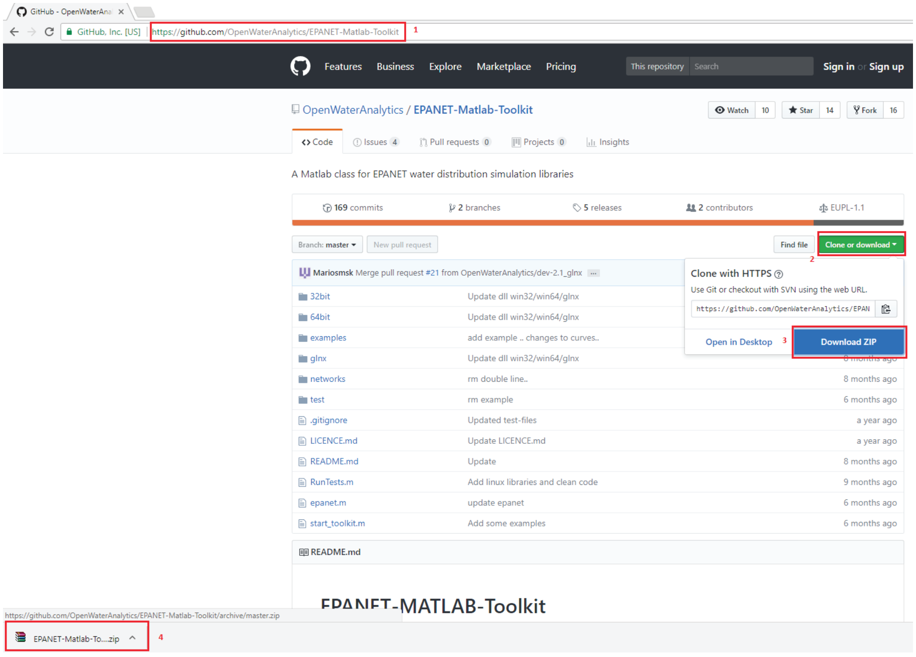
The MATLAB user tends to be a person that needs something more than a calculator but doesn't want to involve himself with a programming language, that's why the work environment is easy to use, almost as easy as a calculator.

MATLAB is a very powerful program with a pleasing development environment that includes development tools for scientific and technical calculation, graphic visualization, as well as a high-level programming language. The last updated version of MATLAB is R2018a and can be downloaded from the official webpage www.mathworks.com, in 32 and 64 bits according to the user's Windows Operating System. The downloaded version for the development in this section is the 64-bit R2017b and will still work with the Net3.inp network.

Marios Kyriakou and Demetrios Eliades have created a Toolkit for Matlab to connect to Epanet, which is in the public domain and is accessible in a digital repository managed by the OpenWaterAnalytics association on the freeware portal github, which we will use in the current book.

A. Create a Project Folder

Just like we've been explaining for other programming environments, it is recommended that the project folder be created on a storage unit that will not have problems upon being read or written. For example, if we have two shared units: C: for installing programs and D: for saving our work, a possible directory could be D:\Epanet_Matlab\Project1. In the folder 'Project1' a compressed file will be saved from the following link, https://github.com/OpenWaterAnalytics/EPANET-Matlab-Toolkit, in which the steps to follow are given, as shown in 4.16.


https://github.com/OpenWaterAnalytics/EPANET-Matlab-Toolkit 1

OpenWaterAnalytics / EPANET-Matlab-Toolkit

A Matlab class for EPANET water distribution simulation libraries

169 commits · 2 branches · 5 releases · 2 contributors · EUPL-1.1

Branch: master · Clone or download 2

Mariosmsk Merge pull request #21 from OpenWaterAnalytics/dev-2.1_glnx

Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/OpenWaterAnalytics/EPAN

Open in Desktop 3 Download ZIP

32bit	Update dll win32/win64/glnx	
64bit	Update dll win32/win64/glnx	
examples	add example .. changes to curves..	
glnx	Update dll win32/win64/glnx	
networks	rm double line..	8 months ago
test	rm example	6 months ago
.gitignore	Updated test-files	a year ago
LICENCE.md	Update LICENCE.md	a year ago
README.md	Update	8 months ago
RunTests.m	Add linux libraries and clean code	9 months ago
epanet.m	update epanet	6 months ago
start_toolkit.m	Add some examples	6 months ago

README.md

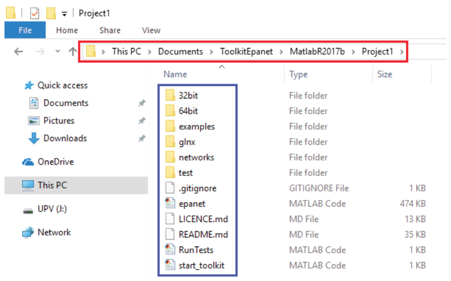
EPANET-MATLAB-Toolkit

https://github.com/OpenWaterAnalytics/EPANET-Matlab-Toolkit/archive/master.zip

EPANET-Matlab-To....zip 4

Figure 4.16. Downloading the Epanet Toolkit for Matlab

Afterward, we proceed to decompress the file and its content will be copied in the 'Project1' folder as shown in Figure 4.17. Once done, we can eliminate the file that was initially downloaded.


This PC › Documents › ToolkitEpanet › MatlabR2017b › Project1

Name	Type	Size
32bit	File folder	
64bit	File folder	
examples	File folder	
glnx	File folder	
networks	File folder	
test	File folder	
.gitignore	GITIGNORE File	1 KB
epanet	MATLAB Code	474 KB
LICENCE.md	MD File	13 KB
README.md	MD File	35 KB
RunTests	MATLAB Code	1 KB
start_toolkit	MATLAB Code	1 KB

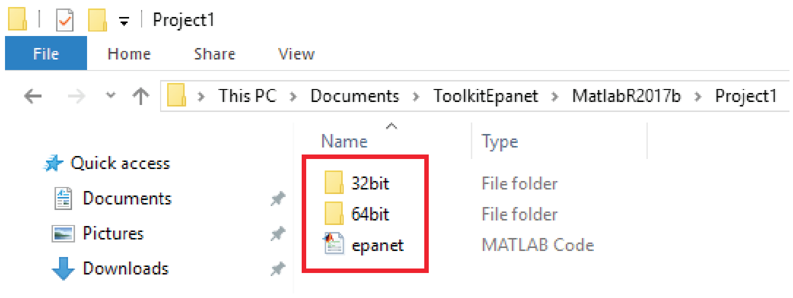
Figure 4.17. Folders and files our project will have

Each one of the files and the content that exists in each folder can be viewed in Table 4.3.

Table 4.3. Files and folders contained in the zipped file EPANET-Matlab-Toolkit-master.zip

Folder/File	Description
32bit	Folder that contains the following files for Windows 32bit OS: epanet2.dll, epanet2.h, epanet2d.exe, epanetmsx.dll, epanetmsx. exe, epanetmsx.h
64bit	Folder that contains the following files for Windows 64bit OS: epanet2.dll, epanet2.h, epanet2d.exe, epanetmsx.dll, epanetmsx. exe, epanetmsx.h
networks	Folder with five Epanet *.inp files and two *.msx files
Test	Folder that has six Matlab files with an *.m extension in which tests can be done for the Epanet functions used in retrieving or modifying the data for the network elements
Epanet	File with an *.m extension in which a class is created that serves as an interface between MATLAB and Epanet
LICENSE	File with an *.md extension in which a description is given for the tool license
README	File with an *.md extension that contains information regarding software requirements, use of the toolbox and a list of the functions that will be used with MATLAB as well as new functions that support the new version of the Epanet 2.1 library
Run Tests	File with *.m extension that has the names of the functions to be called after clicking on the 'Run' button

Of the folders and files in our project folder 'Project1', only the 32-bit and 64-bit folders are necessary as well as the epanet.m file, see Figure 4.18. The rest of the folders can be saved in another directory.

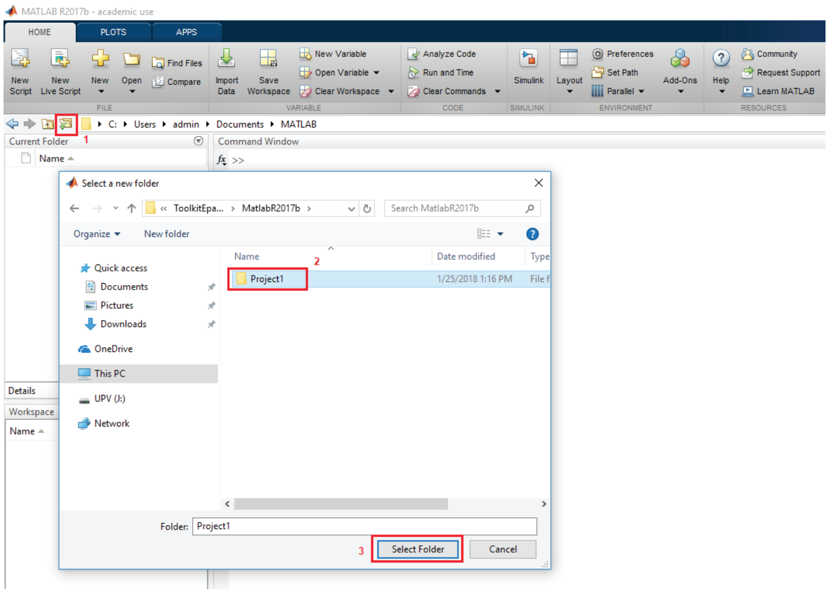


Name	Type
32bit	File folder
64bit	File folder
epanet	MATLAB Code

Figure 4.18. Required files for using the Epanet Toolkit with Matlab

B. Preparing the current project folder with Matlab

After preparing our project folder where our future Matlab files will be saved, we can proceed to open the application and first we will have to choose our work directory, that is to say the folder 'Project1'. For that we follow the steps in Figure 4.19.



Folder: Project1

Figure 4.19. Steps for selecting the current work folder in Matlab

First we will click on the button 'Browse for folder' and it will show us a dialogue box. Second, we will browse through the disc units until finding our project folder 'Project1' and we select it. Third, we will click on the 'Select folder' button, then the 32bit and 64bit folders as well as the epanet.m file will appear in the 'Current Folder' window as shown in Figure 4.20.

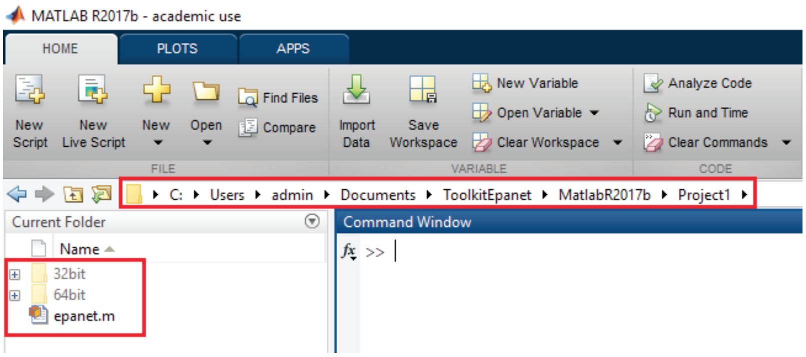


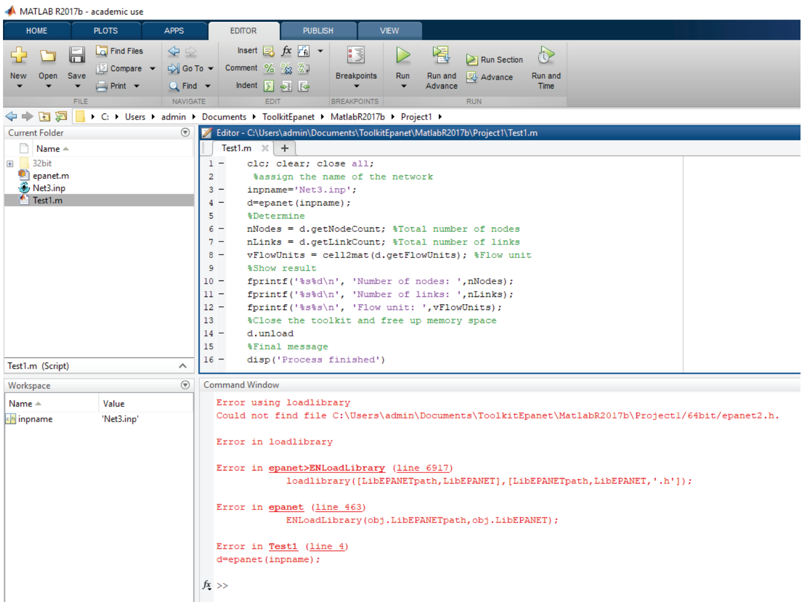
Figure 4.20. Project folder ready for working with the Epanet Toolkit

At this point it is worth mentioning how to go about working with the 32bit and 64bit folders, as well as with the wrapper created (epanet2.m) to access the functions written in the epanet2.dll 32 and 64 bit libraries in an easy manner.

These libraries have been modified and compiled with the new functions that have been added thanks to the work of a group of researchers that contributed their know-how to improving the Epanet Toolkit. Currently the latest available version is 2.1, created on January 22th, 2018.

1. 32bit and 64bit folders

These folders can be saved in any project folder but only one will be called from the epanet2.m. file. This is due to the fact that the code written in the epanet2.m file first analyzes if our computer is 32 or 64 bits. The computer OS on which these exercises are being developed is a 64-bits Windows 10. In this case, if we eliminate the folder named 64bit and run the script Test1.m, it will show an error message, see Figure 4.21, because it won't find said folder. To avoid any future problem, it is recommended that these two folders go together in any new project.



```
clc; clear; close all;
 %assign the name of the network
inpname='Net3.inp';
d=epanet(inpname);
%Determine
nNodes = d.getNodeCount; %Total number of nodes
nLinks = d.getLinkCount; %Total number of links
vFlowUnits = cell2mat(d.getFlowUnits); %Flow unit
%Show result
fprintf('%s%d\n', 'Number of nodes: ',nNodes);
fprintf('%s%d\n', 'Number of links: ',nLinks);
fprintf('%s%s\n', 'Flow unit: ',vFlowUnits);
%Close the toolkit and free up memory space
d.unload
%Final message
disp('Process finished')
```

```
Error using loadlibrary
Could not find file C:\Users\admin\Documents\ToolkitEpanet\MatlabR2017b\Project1/64bit/epanet2.h.

Error in loadlibrary

Error in epanet>ENLoadLibrary (line 6917)
            loadlibrary([LibEPANETpath,LibEPANET],[LibEPANETpath,LibEPANET,'.h']);

Error in epanet (line 463)
            ENLoadLibrary(obj.LibEPANETpath,obj.LibEPANET);

Error in Test1 (line 4)
d=epanet(inpname);

>>
```

Figure 4.21. Error message for not finding the 64bit folder in the project folder

2. The epanet2.m file

In this file we are going to find a class named Epanet, within which a series of properties and methods are set that will enable the use of the Epanet library functions.

All the developed functions up to the latest Epanet version (2.00.12) created by the EPA have been kept. In the most recent version (2.1), created by a group of researchers, several functions have been added, like how to get the node and vertex coordinates for each link, as well as functions for obtaining the property values of an element, for example the diameter of pipes or their length.

C. Creating our first script

To create our script we will click on the New Script button from the HOME menu bar and it will create a new tab with the name Untitled in the editor window as seen in Figure 4.22. We will also copy the network Net3.inp in our project directory.

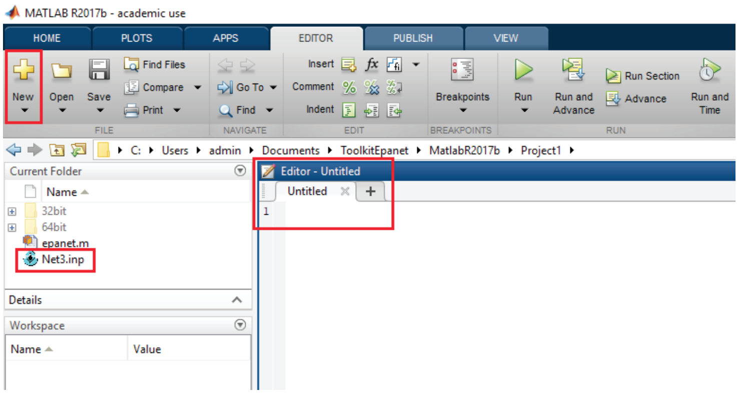
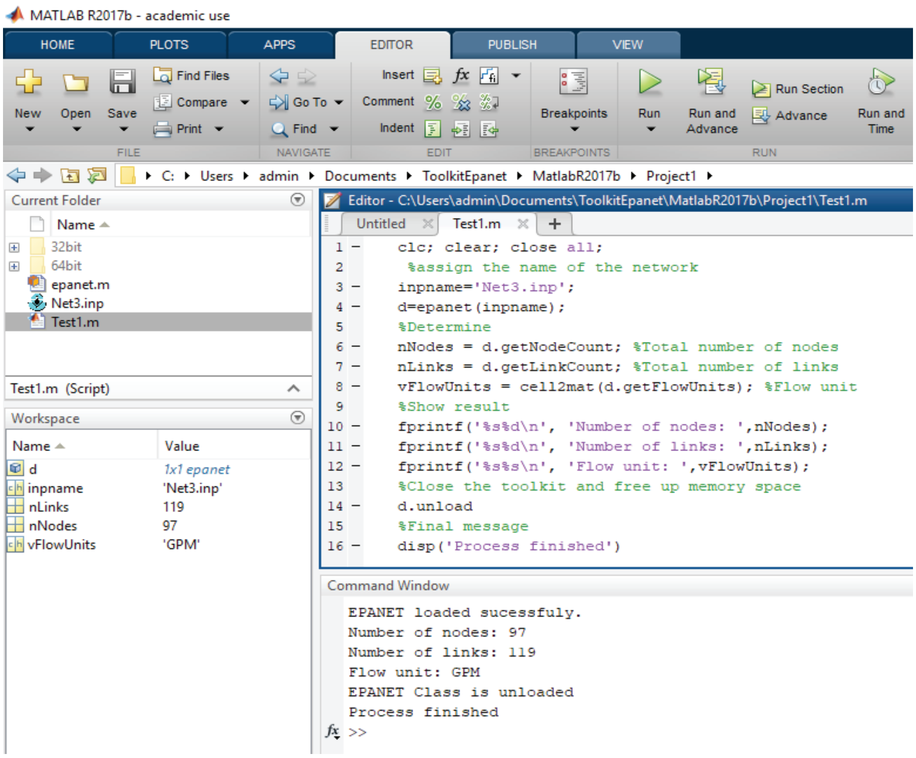


Figure 4.22. Creating a new script in Matlab

Then we write some lines of code like shown in Figure 4.23 and we save the file under the name Test1. This script will allow the retrieval of the total number of nodes and links as well as the flow unit used. In that code, the Epanet class starts up with an input parameter, the name of the network to be analyzed. Afterwards, it goes on to use new methods (functions) declared in the Epanet class, and the results are printed out in the command window. Lastly, we disconnect the Epanet library from the input file, which frees up the memory used. To view the results in the command window we press F5 on our keyboard or we click on the Run button from the EDITOR menu. The file Test1.m can be downloaded from the following link: https://www.imta.gob.mx/biblioteca/download/?key=243.



Name	Value
d	1x1 epanet
inpname	'Net3.inp'
nLinks	119
nNodes	97
vFlowUnits	'GPM'

```
clc; clear; close all;
 %assign the name of the network
inpname='Net3.inp';
d=epanet(inpname);
%Determine
nNodes = d.getNodeCount; %Total number of nodes
nLinks = d.getLinkCount; %Total number of links
vFlowUnits = cell2mat(d.getFlowUnits); %Flow unit
%Show result
fprintf('%s%d\n', 'Number of nodes: ',nNodes);
fprintf('%s%d\n', 'Number of links: ',nLinks);
fprintf('%s%s\n', 'Flow unit: ',vFlowUnits);
%Close the toolkit and free up memory space
d.unload
%Final message
disp('Process finished')
```

```
EPANET loaded sucessfuly.
Number of nodes: 97
Number of links: 119
Flow unit: GPM
EPANET Class is unloaded
Process finished
>>
```

Figure 4.23. Results shown after running the script

4.4. Visual Studio 2017 (C#)

C# (pronounced “C sharp”) is a programming language oriented towards objects, it was developed and standardized by Microsoft® as part of its .NET platform, which afterwards was approved as a standard by the European Computer Manufactures Association (ECMA) and the International Organization for Standardization (ISO). Its basic syntax is derived from C/C++ and it uses the objects model from the .NET platform. It is similar to Java, although it includes improvements derived from other languages. The creation of the language C#, comes from the overlap of drawing of two positive signs over the two symbols of “C++” in seeking to invoke an image of an evolutionary leap as was the case when going from C to C++.

C# or C Sharp is a modern language, simple and entirely oriented towards objects. It simplifies and modernizes C++ in the classes, namespaces, method overloading and exception handling. The complexity of C++ was reduced, making it easier to use and less error-prone. C# can be used to create Windows client applications, Web XML services, distributed components, client-server applications, database applications and user interface designs. This includes an integrated debugger and many other tools. The C# language allows NULL values, enumerations, delegates, lambda expressions and direct memory access which is not available in Java. It also allows generic methods and types that provide greater performance and type security.

As a language oriented towards objects, C# allows for encapsulation concepts, inheritance and polymorphism. All the variables and methods including the Main method, which is an entry point for the application, are enclosed within class definitions. The C# compilation process is simpler compared to C/C++, and it is more flexible than Java. There are no independent header files, nor are the methods and types that need to be declared in a determined order. A C# source code file can define any number of classes, structures, interfaces and events. It saves time in programming since it has a classes library that is very complete and well designed.

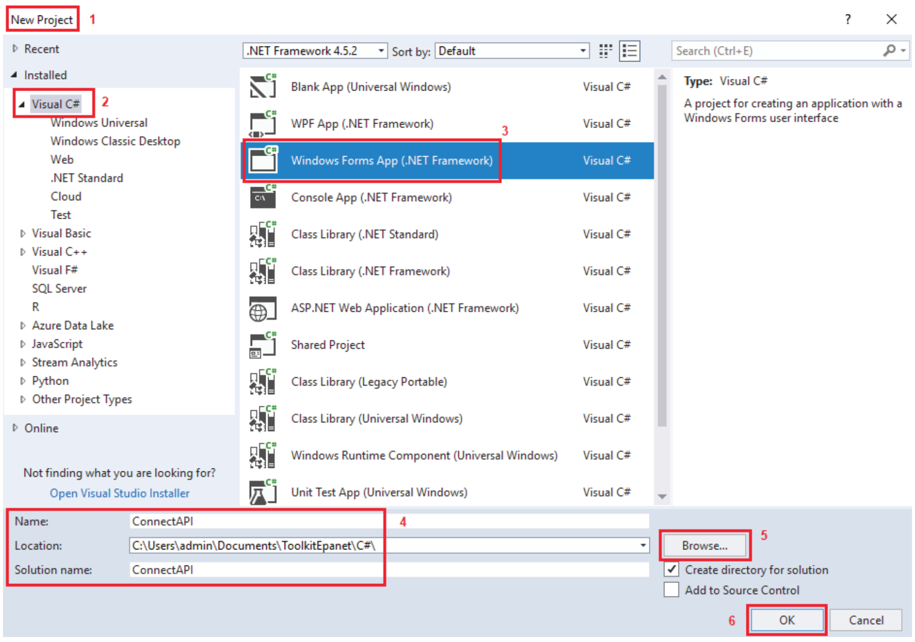
Microsoft Visual Studio 2017 offers compatibility with Visual C# and comes with a complete code editor, a compiler, project template, designers, code assistants, an efficient and easy to use debugger, and many other tools. The .NET Framework class library offers access to numerous OS services and other useful and well-designed classes that accelerate the development cycle significantly.

Through the use of this programming language, we can demonstrate how to connect the Epanet API (32-bit version 2.00.12) within a Visual Studio 2017 environment and have access to its functions. We also have to create a class that contains all the properties and methods that will call the functions contained in the Epanet library, what in computing terms is called a wrapper.

Once Visual Studio 2017 is installed, including C# language, we prepare our work environment to begin using the Epanet library. The same network model (Net3.inp) will be used and the total number of nodes, links and the flow unit used will be retrieved. The following is a description of steps to follow.

A. Create a new project

We open Visual Studio 2017 and we click on New Project…, then a window appears in which we choose the programming language to be used (C#) and we select the option of creating an application with a Windows user interface (Windows Forms Application). Next, we write a name for our project and our solution (ConnectAPI) then we save it to the directory of our choosing and, finally, we click on the OK button. Figure 4.24 summarizes these steps.


New Project
1
?
×
Recent
Installed
Visual C#
2
Windows Universal
Windows Classic Desktop
Web
.NET Standard
Cloud
Test
Visual Basic
Visual C++
Visual F#
SQL Server
R
Azure Data Lake
JavaScript
Stream Analytics
Python
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.5.2
Sort by: Default
Search (Ctrl+E)
Blank App (Universal Windows) Visual C#
WPF App (.NET Framework) Visual C#
3
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Standard) Visual C#
Class Library (.NET Framework) Visual C#
ASP.NET Web Application (.NET Framework) Visual C#
Shared Project Visual C#
Class Library (Legacy Portable) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
4
Browse...
5
Add to Source Control
6
OK
Cancel

Name: ConnectAPI
Location: C:\Users\admin\Documents\ToolkitEpanet\C#\
Solution name: ConnectAPI
☑ Create directory for solution
☐ Add to Source Control

Figure 4.24. Creating a new project and solution with Visual Studio 2017

After creating a new project, an empty form will appear for us to add our objects (Controls) that are necessary to design our own tools. This will be done, further on down, however, since first we need to prepare a class folder (EpanetCSharpLibrary.cs) where a group of global variables and functions will be declared that will permit us to interact with the Epanet (epanet2.dll) library.

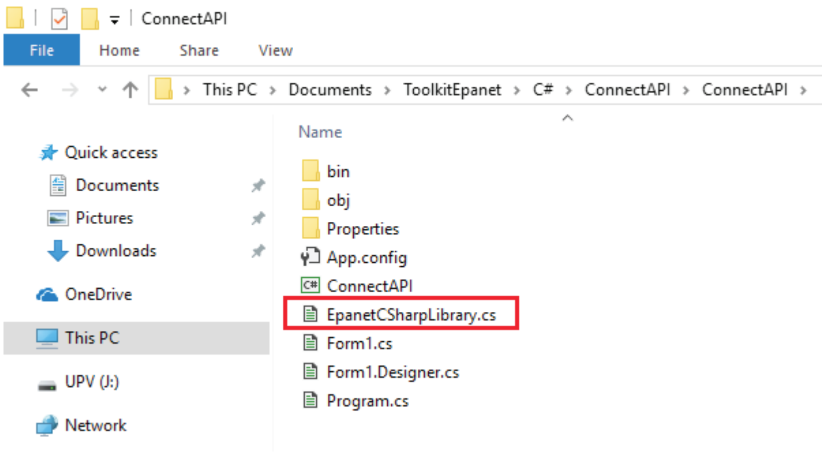
B. Adding an EpanetCSharpLibrary.cs object to a new project

To use all the functions in the Epanet library, a class that has all the global variables will have to be created to allow interaction with the Epanet library.

Elad Salomons, through his webpage http://www.water-simulation.com/wsp/ published an article dated April 21st 2013, titled 'Using EPANet Toolkit in C#', available on the following link: http://www.water-simulation.com/wsp/2013/04/21/using-epanet-Toolkit-in-csharp/, in which the content for a new Epanet class can be viewed, created within a EpanetCSSharpLibrary workspace that permits interaction with the Epanet (epanet2.dll) library functions. This class can be downloaded from the following link: https://www.imta.gob.mx/biblioteca/download/?key=962785, although it is recommended to consult the article periodically in case of possible future changes. The last update to the class file, created to connect to epanet2.dll with C#, is the 24th of February, 2014.

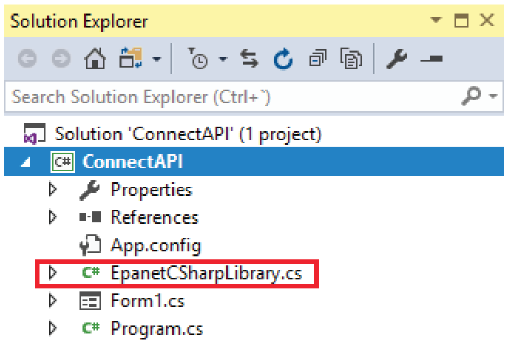
With the following link: http://www.water-simulation.com/wsp/2014/02/25/epanet-class-for-c-sharp/, Elad published another article related to Epanet and C#, this time developed by Vyacheslav Shevelyov.

To add the EpanetCSSharpLibrary class file, first we download it from the following link: https://www.imta.gob.mx/biblioteca/download/?key=962785 or from Elad Salomons' website, and we copy it to the ConnectAPI folder, as can be seen on Figure 4.25. Then from Visual Studio 2017, we load the class file, like shown on Figure 4.26. It is recommended to save all changes as they are made to the project.



bin
obj
Properties
App.config
ConnectAPI
EpanetCSharpLibrary.cs
Form1.cs
Form1.Designer.cs
Program.cs

Figure 4.25. Copying the EpanetCSSharpLibrary class to the project folder



Solution 'ConnectAPI' (1 project)
ConnectAPI
Properties
References
App.config
EpanetCSharpLibrary.cs
Form1.cs
Program.cs

Figure 4.26. Adding the EpanetCSSharpLibrary class using Visual Studio 2017

C. Copying the epanet2.dll file in the Debug folder

After adding the EpanetCSSharpLibrary to the project, we need to copy the Epanet (epanet2.dll version 2.00.12) library within the "Debug" folder which is in the "bin" folder. For this we have to click on the "Show all folders" button from the Solution explorer to view the "bin" and "obj" folders, as can be seen in Figure 4.27.

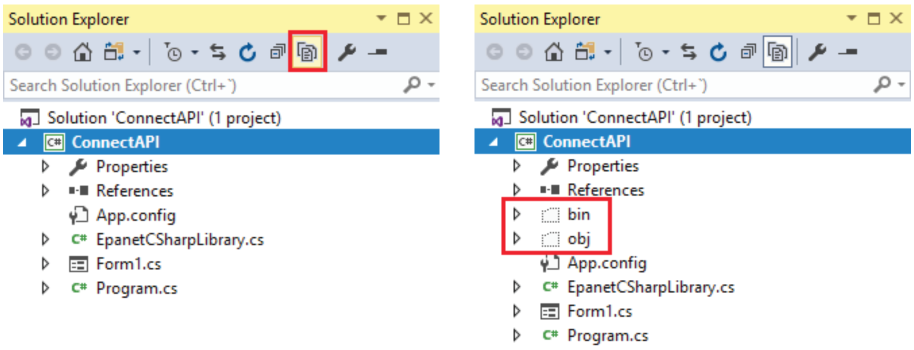


Figure 4.27. Viewing the bin y obj folders with the solution explorer

For users that don't have the Epanet library (v. 2.00.12), it can be downloaded from the official EPA website https://www.epa.gov/sites/production/files/2014-06/en2toolkit.zip.

Upon opening the "Debug" folder (Figure 4.28), we can verify if the Epanet library has been copied correctly.

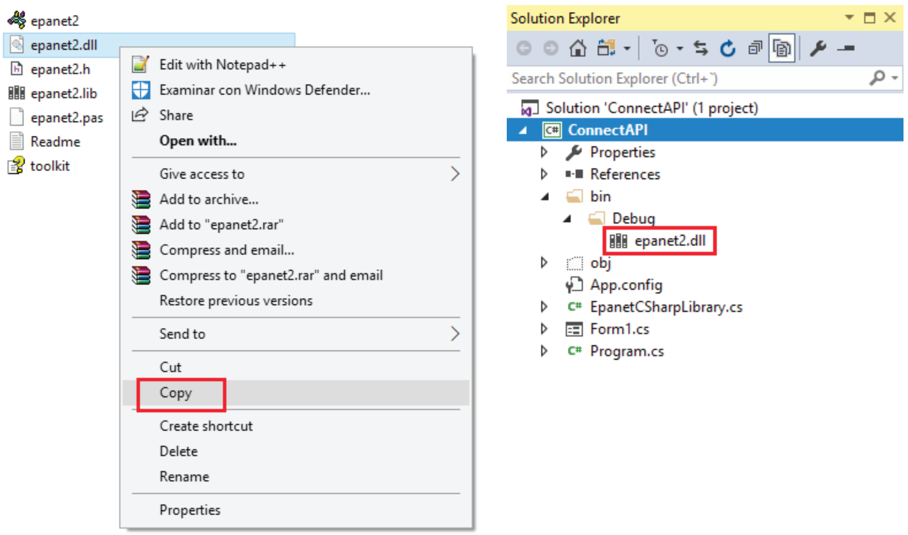


Figure 4.28. Copying the epanet2.dll file in the Debug folder

After the previous steps we can add controls to our form. They are the same ones that were added to Visual Basic 6.0 and Visual Basic 2017.

D. Preparing our form

The controls that will be added to the form are as follows: 01 GroupBox, 04 Label, 02 Button, 04 Textbox, 01 OpenFileDialog. The last control is only visible in design mode not in run time mode. All these controls are found in the toolbox window (Figure 4.29), should it not be visible, we can activate it by pressing Ctrl+Alt+X or clicking the toolbox in the View menu.

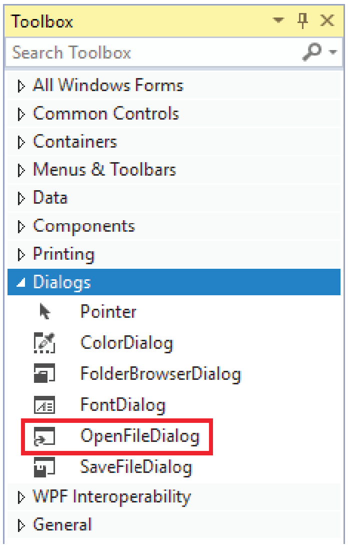


Figure 4.29. Visual Studio 2017 (C#) Toolkit

On Table 4.4, we can see the original names of the controls as well as the new ones added to the form, additionally, you can see the content they will have in the Caption and Text properties.

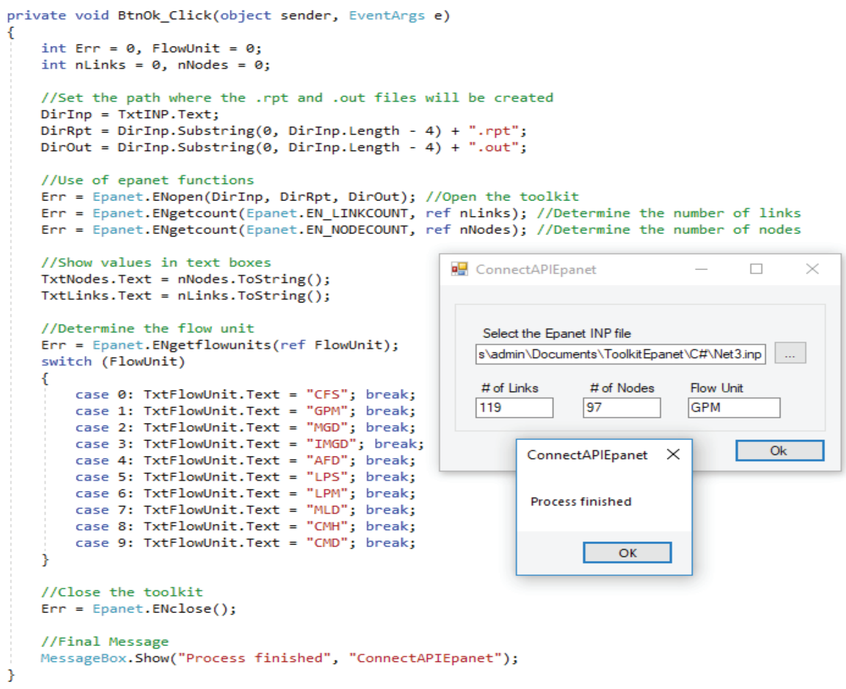
Table 4.4. Controls added to the main form

Property: Name [Original]	Property: Name [Changed]	Property
GroupBox1	GBContainer	Text: empty
Label1	LblINP	Text: Select the Epanet Inp file
Label2	LblNumLinks	Text: # of Links
Label3	LblNumNodes	Text: # de Nodes
Label4	LblFlowUnt	Text: Flow Unit
TextBox1	TxtINP	Text: empty
TextBox2	TxtLinks	Text: empty
TextBox3	TxtNodes	Text: empty
TextBox4	TxtFlowUnt	Text: empty
Button1	BtnOpen	Text: …
Button2	BtnOk	Text: Ok
OpenFileDialog	OpenFileDialog	FileName: empty

E. Working with the Epanet Toolkit

The Buttons OpenBtn and AcceptBtn will be associated with a cluster of sentences that will permit the selection of the Epanet inp file and the retrieval of the total number of junctions and links, and the unit flow used. Before using the ENopen, ENgetcount, ENgetflowunits, and ENclose methods or functions it is necessary to import the namespaces EpanetCSSharpLibrary, that will contain the 'Epanet' class, which in turn will hold the aforementioned methods.

If we wish to consult the headloss formula, it will not be possible through the Epanet Toolkit since such function does not exist in that library. To get this, it will be necessary to use the native Visual C# functions and to search through the inp file until finding said parameter within the [OPTIONS] section. In Figure 4.30, part of the source code used along with the form in run time mode can be seen as well as the results. To download the complete source code, click on the following link: https://www.imta.gob.mx/biblioteca/download/?key=231.



```
private void BtnOk_Click(object sender, EventArgs e)
{
    int Err = 0, FlowUnit = 0;
    int nLinks = 0, nNodes = 0;

    //Set the path where the .rpt and .out files will be created
    DirInp = TxtINP.Text;
    DirRpt = DirInp.Substring(0, DirInp.Length - 4) + ".rpt";
    DirOut = DirInp.Substring(0, DirInp.Length - 4) + ".out";

    //Use of epanet functions
    Err = Epanet.ENopen(DirInp, DirRpt, DirOut); //Open the toolkit
    Err = Epanet.ENgetcount(Epanet.EN_LINKCOUNT, ref nLinks); //Determine the number of links
    Err = Epanet.ENgetcount(Epanet.EN_NODECOUNT, ref nNodes); //Determine the number of nodes

    //Show values in text boxes
    TxtNodes.Text = nNodes.ToString();
    TxtLinks.Text = nLinks.ToString();

    //Determine the flow unit
    Err = Epanet.ENgetflowunits(ref FlowUnit);
    switch (FlowUnit)
    {
        case 0: TxtFlowUnit.Text = "CFS"; break;
        case 1: TxtFlowUnit.Text = "GPM"; break;
        case 2: TxtFlowUnit.Text = "MGD"; break;
        case 3: TxtFlowUnit.Text = "IMGD"; break;
        case 4: TxtFlowUnit.Text = "AFD"; break;
        case 5: TxtFlowUnit.Text = "LPS"; break;
        case 6: TxtFlowUnit.Text = "LPM"; break;
        case 7: TxtFlowUnit.Text = "MLD"; break;
        case 8: TxtFlowUnit.Text = "CMH"; break;
        case 9: TxtFlowUnit.Text = "CMD"; break;
    }

    //Close the toolkit
    Err = Epanet.ENclose();

    //Final Message
    MessageBox.Show("Process finished", "ConnectAPIEpanet");
}
```

ConnectAPIEpanet

Select the Epanet INP file: s\admin\Documents\ToolkitEpanet\C#\Net3.inp

# of Links	# of Nodes	Flow Unit
119	97	GPM

ConnectAPIEpanet

Process finished

Figure 4.30. Running Visual Studio 2017 (C#) and results

4.5. Python Shell (Python)

Python is a general-purpose programming language that is free even for business use. It bets on simplicity, versatility and developmental quickness. It is a scripting language that is platform-independent (Windows, Mac, Linux, etc.) and object oriented. It is made for programming any type of application, from Windows applications to network servers and even web pages.

It is a mutiparadigmatic programming language because it supports object orientation, imperative programming and to a lesser degree, functional programming[4].

[4] https://en.wikipedia.org/wiki/Python_(programming_language)

It is an interpreted language, which means that it is not necessary to compile source code to run it, which offers advantages like developmental quickness and disadvantages like a lower speed.

Python has become popular thanks to the number of libraries it contains and the functions incorporated into the language itself, which help to carry out many routine tasks without the need to program them from scratch. It is worth noting that Python has a very visual syntax thanks to its indented notation (with margins) of required compliance.

The Python interpreter and the extensive standard library are freely available in binary form and in source code for the main platforms off of the Python website, http://www.python.org/, and can be distributed freely. The same website has distributions and links to many free third-party Python modules, programs, tools and additional documentation.

Python is Open Source, anyone can contribute to its development and dissemination. Moreover, it is not necessary to pay for a license to distribute software developed with this language, even its interpreter can be distributed freely for different platforms. The latest version of Python goes by several names among them, Python 3000 or Py3K, even though it is habitually referred to as Python 3.

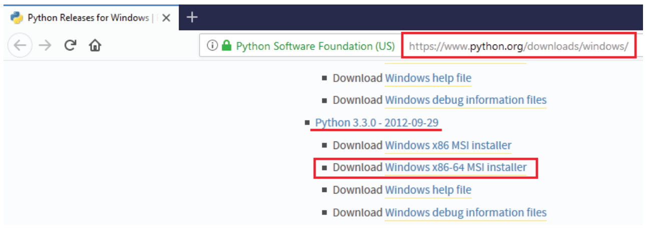
Using this programming language, we will import an Epanet module (a type of wrapper) to call functions found in the Epanet (version 2.00.12) API from a Python 3.3 integrated development environment (since this wrapper only works with Python 3). To be able to use the Epanet library functions we have to already have the Epanet 2.00.12 application installed from the official EPA page, since its wrapper (epanet2.py) searches for the (epanet2.dll) library within our computer: https://www.epa.gov/sites/production/files/2014-06/en2setup_0.exe.

Once Python 3.3 is installed, we open up the Python Shell application and it will show the Epanet functions step-by-step. We will work with the same network model

(Net3.inp) and the total number of network junctions and links, and the flow unit used will be retrieved.

A. Installing Python 3.3

In case it is not installed we can go to the following link https://www.python.org/downloads/windows/, look for the section Python 3.3.0-2012-09-29 and download the Windows x86-64 MSI.exe installer, as shown in Figure 4.31.


Python Releases for Windows
Python Software Foundation (US)
https://www.python.org/downloads/windows/

- Download Windows help file
- Download Windows debug information files
- Python 3.3.0 - 2012-09-29
  - Download Windows x86 MSI installer
  - Download Windows x86-64 MSI installer
  - Download Windows help file
  - Download Windows debug information files

Figure 4.31. Download link for Python 3.3 from the official webpage

After following the default installation process, we will have Python installed on our computer (Figure 4.32). Among the installed files there is one called IDLE (Python GUI). IDLE means Integrated DeveLopment Environment. This is a programming tool that permits for the writing and editing of Python code (Figure 4.33). It is a multi-window text editor that has an autofill function, and an integrated debugger along with the possibility of step-by-step execution tracking with interruption points and a stack viewer. For more on IDLE consult the following link: https://docs.python.org/3/library/idle.html.

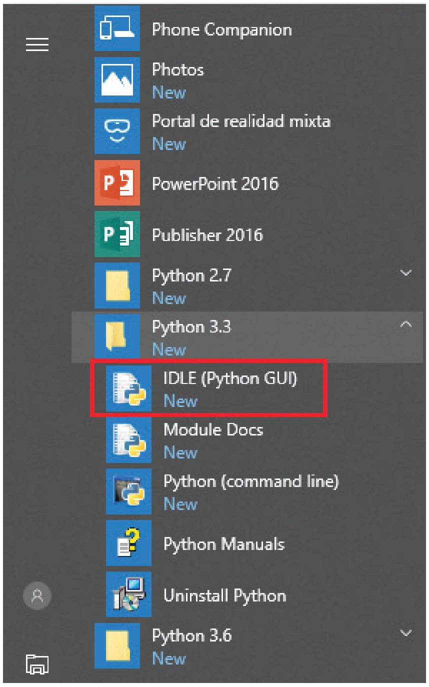
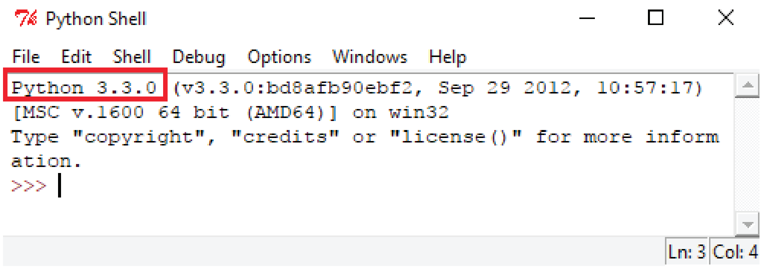


Figure 4.32. Installing Python 3.3



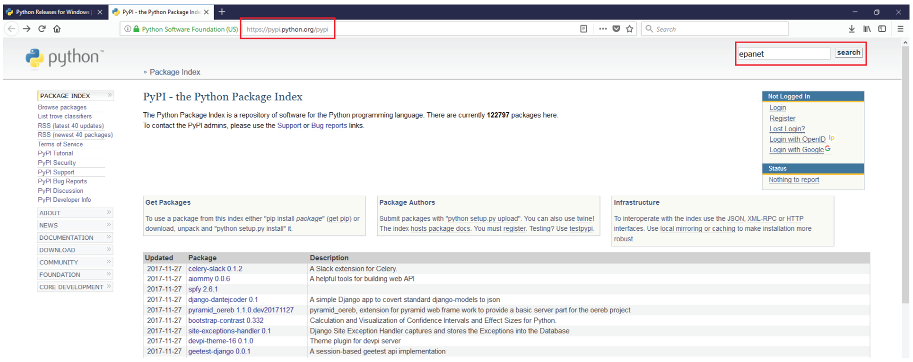
```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> 
```

Figure 4.33. Python Shell tool in Python 3.3

B. Download and installation of the Epanet2 0.4.0.1dev file

To connect to the Epanet library and use its functions using the Python Shell tool, it is necessary to install a package from inside the Python directory, specifically, the file: "*/Python33/Lib/site-packages/*". For that we go to the following link https://pypi.python.org/pypi, which is a repository of packages and/or Python modules that the community develops and can be downloaded free of charge under the user's responsibility.

If we want to search the package or module and we don't remember the link, we can write "Epanet" in the text box on the right side of the page (Figure 4.34), and at that moment we get a list of what we might be searching for as shown in Figure 4.35.



PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **122797** packages here.
To contact the PyPI admins, please use the Support or Bug reports links.

Get Packages

To use a package from this index either "pip install *package*" (get pip) or download, unpack and "python setup.py install" it.

Package Authors

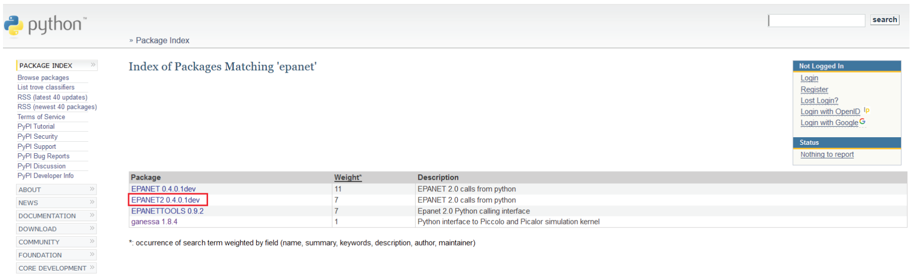
Submit packages with "python setup.py upload". You can also use twine! The index hosts package docs. You must register. Testing? Use testpypi.

Infrastructure

To interoperate with the index use the JSON, XML-RPC or HTTP interfaces. Use local mirroring or caching to make installation more robust.

Updated	Package	Description
2017-11-27	celery-slack 0.1.2	A Slack extension for Celery.
2017-11-27	aiommy 0.0.6	A helpful tools for building web API
2017-11-27	spfy 2.6.1	
2017-11-27	django-dantejcoder 0.1	A simple Django app to covert standard django-models to json
2017-11-27	pyramid_oereb 1.1.0.dev20171127	pyramid_oereb, extension for pyramid web frame work to provide a basic server part for the oereb project
2017-11-27	bootstrap-contrast 0.332	Calculation and Visualization of Confidence Intervals and Effect Sizes for Python.
2017-11-27	site-exceptions-handler 0.1	Django Site Exception Handler captures and stores the Exceptions into the Database
2017-11-27	devpi-theme-16 0.1.0	Theme plugin for devpi server
2017-11-27	geetest-django 0.0.1	A session-based geetest api implementation

Figure 4.34. Searching for the Epanet2 0.4.0.1dev installer



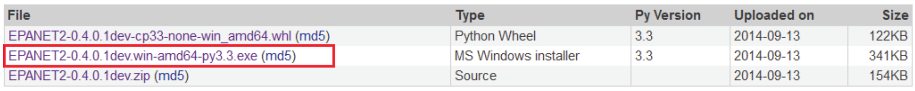
Index of Packages Matching 'epanet'

Package	Weight*	Description
EPANET 0.4.0.1dev	11	EPANET 2.0 calls from python
EPANET2 0.4.0.1dev	7	EPANET 2.0 calls from python
EPANETTOOLS 0.9.2	7	Epanet 2.0 Python calling interface
ganessa 1.8.4	1	Python interface to Piccolo and Picalor simulation kernel

*: occurrence of search term weighted by field (name, summary, keywords, description, author, maintainer)

Figure 4.35. Result of the Epanet2 0.4.0.1dev installer search

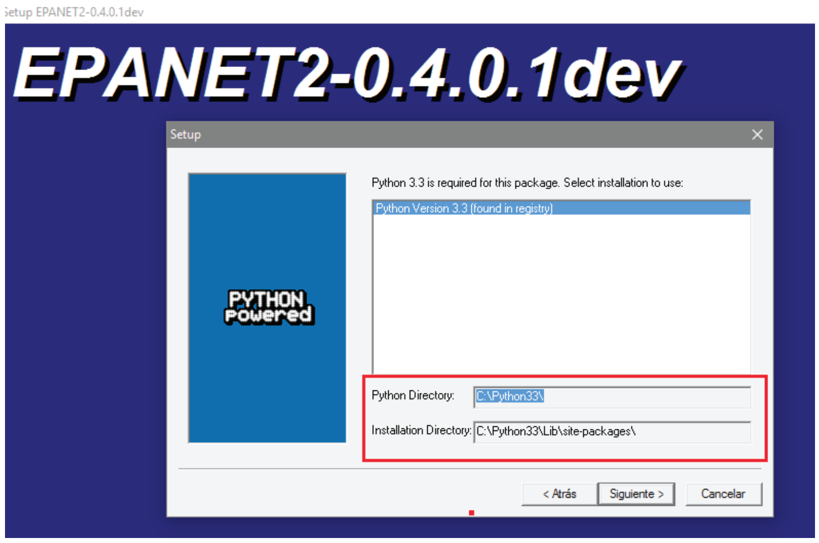
Upon clicking on the name of the “Epanet2 0.4.0.1dev” installer, it redirects us to the main page of the file we are seeking to install. Once there, we click on the green button that is on the right side of the page and it takes us to the file section for downloading. We will click on “Epanet2-0.4.0.1dev.win-amd64-py3.3.exe (md5)”, as shown in Figure 4.36.



File	Type	Py Version	Uploaded on	Size
EPANET2-0.4.0.1dev-cp33-none-win_amd64.whl (md5)	Python Wheel	3.3	2014-09-13	122KB
EPANET2-0.4.0.1dev.win-amd64-py3.3.exe (md5)	MS Windows installer	3.3	2014-09-13	341KB
EPANET2-0.4.0.1dev.zip (md5)	Source		2014-09-13	154KB

Figure 4.36. Downloading the “Epanet2 0.4.0.1 dev” installer

Once downloaded we begin the installation process. Inside said process, previous installation of the Python 3.3 will be verified as can be seen in Figure 4.37.


Setup EPANET2-0.4.0.1dev

EPANET2-0.4.0.1dev

Setup

Python 3.3 is required for this package. Select installation to use:

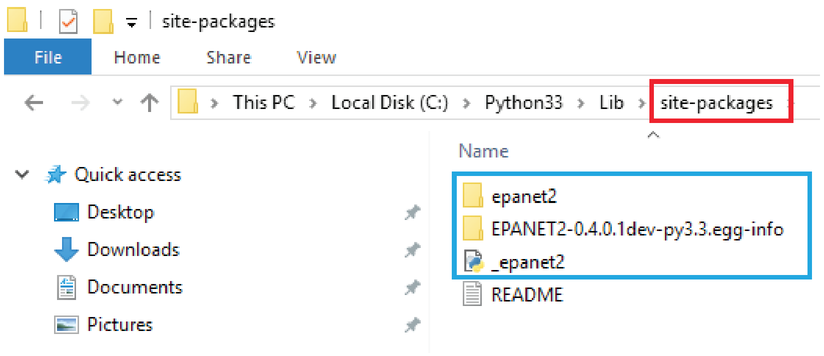
Python Version 3.3 (found in registry)

Python Directory: C:\Python33\

Installation Directory: C:\Python33\Lib\site-packages\

Figure 4.37. Verifying the Python 3.3 directory in the installation process

Now the folder "site-packages" should contain two additional folders and a Python file, as seen in Figure 4.38. In that folder there might be other packages or modules from other installations.



epanet2
EPANET2-0.4.0.1dev-py3.3.egg-info
_epanet2
README

Figure 4.38. Installing the epanet2 package, EPANET2-0.4.0.1dev and _epanet2

C. Working with the Epanet Toolkit

Now we are almost ready to begin using the Epanet functions, but first it will be important to explain a few things. Our package and module of interest are called epanet2, as shown in Figure 4.39. The epanet2.py module is the wrapper in which the declared global variables and functions are found that are used to modify the values of our network model.

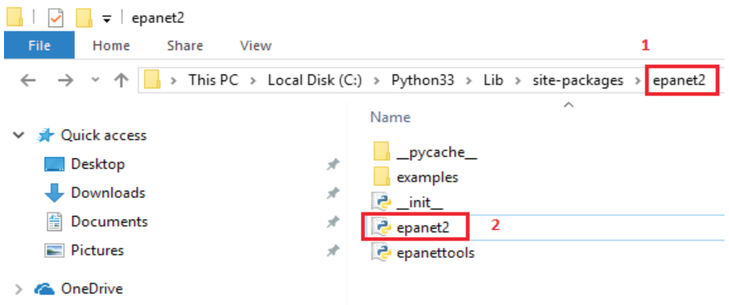
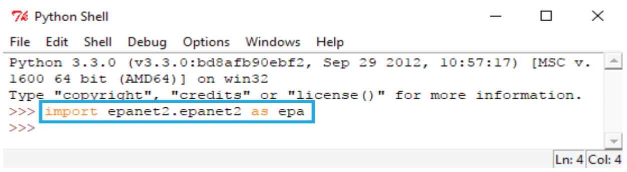


Figure 4.39. "epanet2" package and "epanet2" module

The next step is to import the "epanet2" module which is found inside the "epanet2" package. For that we open the Python Shell tool and we write the line of code that appears in Figure 4.40. To access the Epanet library functions written in the epanet2.py file, we have to import the namespace (epanet2.epanet2). Afterwards, in the following line of code we go back to writing said namespace followed by a period and the name of the function to be used.

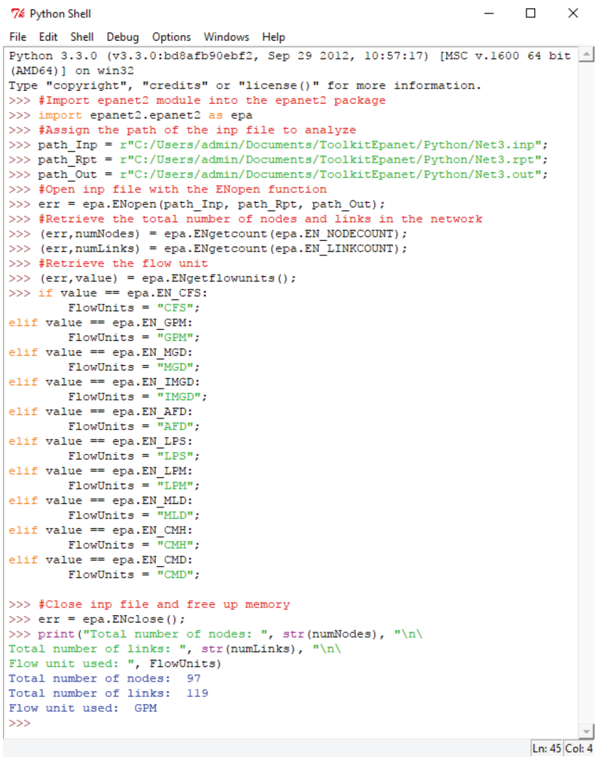


```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.
1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import epanet2.epanet2 as epa
>>>
```

Figure 4.40. Importing the epanet2.py module

It is also possible to abbreviate the namespaces using an alias. For that, during the import phase, the "as" key is added followed by an alias which we will use to refer to that imported namespace in the future. In our case, writing "epa" will be the equivalent to writing the namespace "epanet2.epanet2".

A large amount of that source code can be seen in Figure 4.41, where we will go on to retrieve the total number of nodes, links as well as the flow unit in the Net3.inp network. The ConnectEpanetAPI.py file can be downloaded off of the following link https://www.imta.gob.mx/biblioteca/download/?key=232. In this file, all that appears in the Python Shell tool text editor (code and results) is saved using the line-by-line mode (do not copy the file in the command window). It is supposed that the Net3 file is to be found in the E:/ directory, should that not be the case then update the path. The path names do not allow for blanks or special characters.

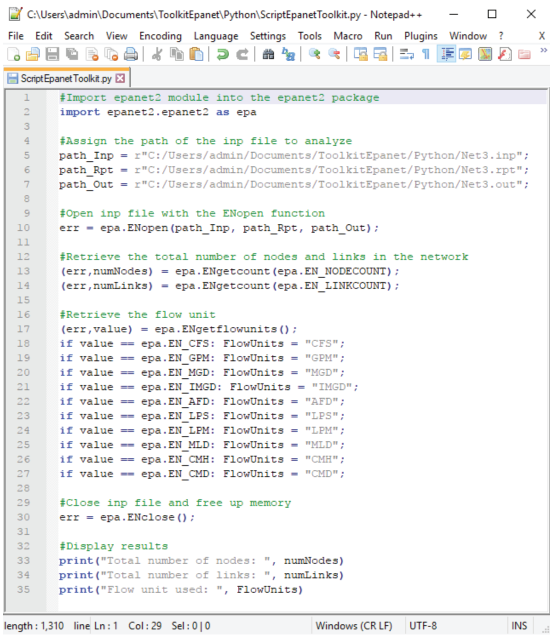


```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> #Import epanet2 module into the epanet2 package
>>> import epanet2.epanet2 as epa
>>> #Assign the path of the inp file to analyze
>>> path_Inp = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.inp";
>>> path_Rpt = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.rpt";
>>> path_Out = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.out";
>>> #Open inp file with the ENopen function
>>> err = epa.ENopen(path_Inp, path_Rpt, path_Out);
>>> #Retrieve the total number of nodes and links in the network
>>> (err,numNodes) = epa.ENgetcount(epa.EN_NODECOUNT);
>>> (err,numLinks) = epa.ENgetcount(epa.EN_LINKCOUNT);
>>> #Retrieve the flow unit
>>> (err,value) = epa.ENgetflowunits();
>>> if value == epa.EN_CFS:
        FlowUnits = "CFS";
elif value == epa.EN_GPM:
        FlowUnits = "GPM";
elif value == epa.EN_MGD:
        FlowUnits = "MGD";
elif value == epa.EN_IMGD:
        FlowUnits = "IMGD";
elif value == epa.EN_AFD:
        FlowUnits = "AFD";
elif value == epa.EN_LPS:
        FlowUnits = "LPS";
elif value == epa.EN_LPM:
        FlowUnits = "LPM";
elif value == epa.EN_MLD:
        FlowUnits = "MLD";
elif value == epa.EN_CMH:
        FlowUnits = "CMH";
elif value == epa.EN_CMD:
        FlowUnits = "CMD";

>>> #Close inp file and free up memory
>>> err = epa.ENclose();
>>> print("Total number of nodes: ", str(numNodes), "\n\
Total number of links: ", str(numLinks), "\n\
Flow unit used: ", FlowUnits)
Total number of nodes:  97
Total number of links:  119
Flow unit used:  GPM
>>>
```

Figure 4.41. Result of consulting the Net3.inp file using Python Shell

As shown in Figure 4.41, the result appears immediately after pressing the Enter key on the last command line. If we wish to modify some input data from the Net3.inp network file, whether it's eliminating or adding elements (junctions/links), we have to write everything all over. A very common practice by programmers is to write the source code using the NotePad++ application and to run the module using Python Shell.

Now we will learn to write source code using NotePad++. If we don´t have said tool installed, we can download it from the following link: https://notepad-plus-plus.org/. The source code is almost the same as the one used in the Python Shell. We open the NotePad++ application, write the code that appears in Figure 4.42 and we save the file with the name ScriptEpanetToolkit.py. If we prefer to download the code file, click on the following link: https://www.imta.gob.mx/biblioteca/download/?key=242.


C:\Users\admin\Documents\ToolkitEpanet\Python\ScriptEpanetToolkit.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ScriptEpanetToolkit.py
length : 1,310 lines : 35 Ln : 1 Col : 29 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

```python
#Import epanet2 module into the epanet2 package
import epanet2.epanet2 as epa

#Assign the path of the inp file to analyze
path_Inp = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.inp";
path_Rpt = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.rpt";
path_Out = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.out";

#Open inp file with the ENopen function
err = epa.ENopen(path_Inp, path_Rpt, path_Out);

#Retrieve the total number of nodes and links in the network
(err,numNodes) = epa.ENgetcount(epa.EN_NODECOUNT);
(err,numLinks) = epa.ENgetcount(epa.EN_LINKCOUNT);

#Retrieve the flow unit
(err,value) = epa.ENgetflowunits();
if value == epa.EN_CFS: FlowUnits = "CFS";
if value == epa.EN_GPM: FlowUnits = "GPM";
if value == epa.EN_MGD: FlowUnits = "MGD";
if value == epa.EN_IMGD: FlowUnits = "IMGD";
if value == epa.EN_AFD: FlowUnits = "AFD";
if value == epa.EN_LPS: FlowUnits = "LPS";
if value == epa.EN_LPM: FlowUnits = "LPM";
if value == epa.EN_MLD: FlowUnits = "MLD";
if value == epa.EN_CMH: FlowUnits = "CMH";
if value == epa.EN_CMD: FlowUnits = "CMD";

#Close inp file and free up memory
err = epa.ENclose();

#Display results
print("Total number of nodes: ", numNodes)
print("Total number of links: ", numLinks)
print("Flow unit used: ", FlowUnits)
```

Figure 4.42. Source code (Python Language) written using NotePad++

To run Python files, there are two ways to do it: Importing the script or running the script using Python Shell.

1. Importing script using Python Shell

   To run a script directly using Python Shell, it must be saved in the following direction /Python33/Lib/site-packages, like shown in Figure 4.43.

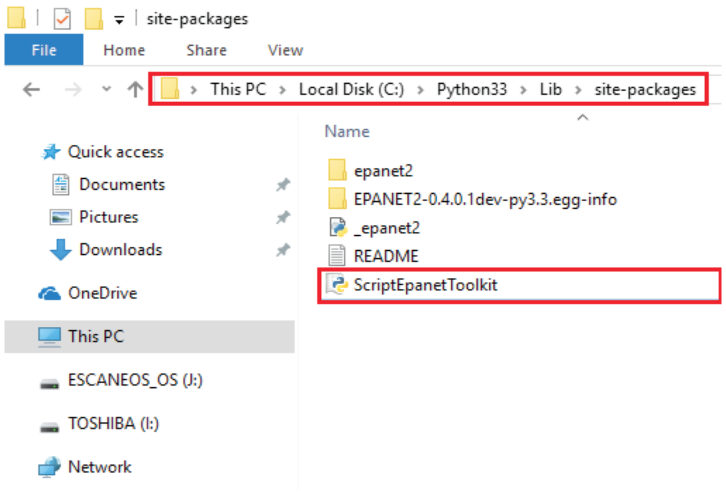
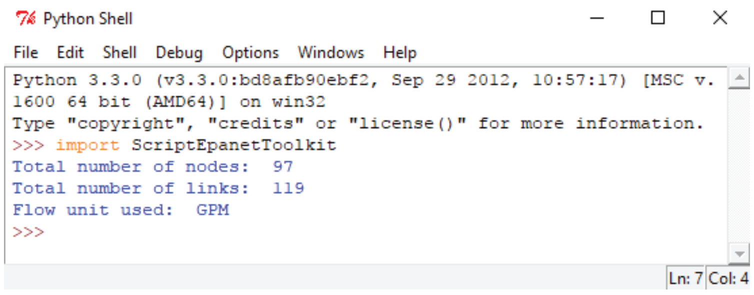


Figure 4.43. Copying the ScriptEpanetToolkit.py file to the site-packages folder

After writing "import ScriptEpanetToolkit" in the Python Shell editor and pressing the Enter key, the result will be shown (Figure 4.44).

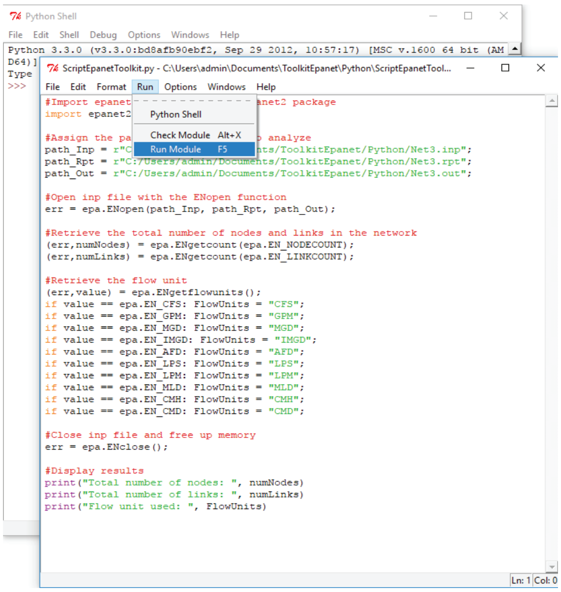


```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.
1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import ScriptEpanetToolkit
Total number of nodes:  97
Total number of links:  119
Flow unit used:  GPM
>>>
```

Figure 4.44. Importing the ScriptEpanetToolkit file and showing results

2. Running script using Python Shell

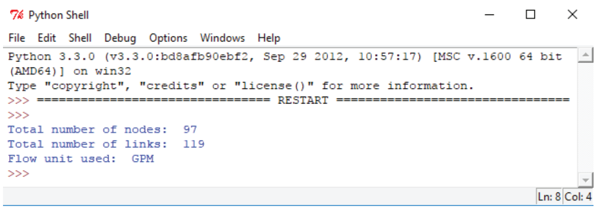
In this case it is not necessary to save our script in the "site-packages" folder, it can be in any removable or disc unit. First we open Python Shell and using the "open…" option from the File menu, we will search for our script that will appear in another window from which we will run the Script. For that we will click on the "Run" menu and then we will click on the "Run Module" command, as can be seen in Figure 4.45. The result will appear in the main Python Shell, see Figure 4.46.


Python Shell
File Edit Shell Debug Options Windows Help
ScriptEpanetToolkit.py - C:\Users\admin\Documents\ToolkitEpanet\Python\ScriptEpanetTool...
File Edit Format Run Options Windows Help
Python Shell
Check Module Alt+X
Run Module F5
Ln: 1 Col: 0

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AM
D64)]
Type
>>>
```

```
#Import epanet                anet2 package
import epanet2

#Assign the pa                 analyze
path_Inp = r"C                nts/ToolkitEpanet/Python/Net3.inp";
path_Rpt = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.rpt";
path_Out = r"C:/Users/admin/Documents/ToolkitEpanet/Python/Net3.out";

#Open inp file with the ENopen function
err = epa.ENopen(path_Inp, path_Rpt, path_Out);

#Retrieve the total number of nodes and links in the network
(err,numNodes) = epa.ENgetcount(epa.EN_NODECOUNT);
(err,numLinks) = epa.ENgetcount(epa.EN_LINKCOUNT);

#Retrieve the flow unit
(err,value) = epa.ENgetflowunits();
if value == epa.EN_CFS: FlowUnits = "CFS";
if value == epa.EN_GPM: FlowUnits = "GPM";
if value == epa.EN_MGD: FlowUnits = "MGD";
if value == epa.EN_IMGD: FlowUnits = "IMGD";
if value == epa.EN_AFD: FlowUnits = "AFD";
if value == epa.EN_LPS: FlowUnits = "LPS";
if value == epa.EN_LPM: FlowUnits = "LPM";
if value == epa.EN_MLD: FlowUnits = "MLD";
if value == epa.EN_CMH: FlowUnits = "CMH";
if value == epa.EN_CMD: FlowUnits = "CMD";

#Close inp file and free up memory
err = epa.ENclose();

#Display results
print("Total number of nodes: ", numNodes)
print("Total number of links: ", numLinks)
print("Flow unit used: ", FlowUnits)
```

Figure 4.45. Running script from Python Shell



```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Total number of nodes:  97
Total number of links:  119
Flow unit used:  GPM
>>>
```

Figure 4.46. Result of running script

4.6. Dev-C++ (C++)

C++ is a programming language created by Bjarne Stroustrup in the AT&T labs in 1983. Its creator took the most popular programming language at that time, C, as its basis. The intention of its creator was to extend the C language using mechanisms that would permit the manipulation of objects. In that sense, from the perspective of object-oriented languages, C++ is a hybrid language[5].

The name C++ was proposed by Rick Mascitti in 1983, when the language was used for the first time in a scientific laboratory. Before that, it bore the name “C with classes”. In C++, the expression “C++” means “increments of C” and refers to C++ being an extension of C.

Stroustrup saw the need to make the programming in C language easier. For that he redesigned C, broadening its possibilities while conserving its main quality, allowing the programmer to have full control at all times, enabling faster speeds that were not possible in other languages.

C++ took C to a new paradigm of classes and objects with which he sought a more human understanding based on the construction of objects with their own, exclusive characteristics grouped in classes.

In this section we will learn how to import an Epanet (epanet2.h) header file, and work with the functions in the Epanet dynamic library (epanet2.dll) using a Dev-C++ programming environment (freeware). The header file to be used is not the same as the file distributed on the EPA (Toolkit) website, since said file is prepared for use with 32-bit Microsoft Visual C 6.0.

As we have done with the previous languages, we will be using the Net3.inp network model to retrieve the total number of junctions and links, as well as the flow unit being used.

[5] https://en.wikipedia.org/wiki/C%2B%2B

A. Installing the Dev-C++ application

Dev-C++ is an integrated development environment (IDE), at the same time it is a free compiler for programming in the C/C++ language. The environment is developed in Borland's Delphi language. It has a page of optional packages that can be installed, with different open-source libraries.

Among its most notable characteristics are: integrated debugger, source program editor compatible with a C/C++ syntax, multiwindow editor with multiple editing options, independent files and multifile projects can be worked on, it possesses an installation package generator for Windows environment programs, it can generate DOS programs (console mode) - Windows applications - DLLs, independent windows for the projects manager - editor - compilation results, compilation results - linker - resource generator, it permits the integration of external tools via a "tool manager", etc.

To download the latest version of Dev-C++ (Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe), go to the official blog: http://orwelldevcpp.blogspot.com.es/. The installation process is quite simple, all that is necessary is to follow the installation guide.

B. Create a directory for the project

Before beginning to use the Dev-C++ application we must create a project folder to save the epanet2.h and epanet2.dll files as well as the files that are created for the project. In Figure 4.47, a folder was created named Project1, in the following directory C++\, and in the project folder we will save the header file (epanet2.h) and the Epanet dynamic library (epanet2.dll). These two files can be downloaded from the following link: https://www.imta.gob.mx/biblioteca/download/?key=962789.

Each user can create his project folder in any storage unit so long as it isn't write protected.

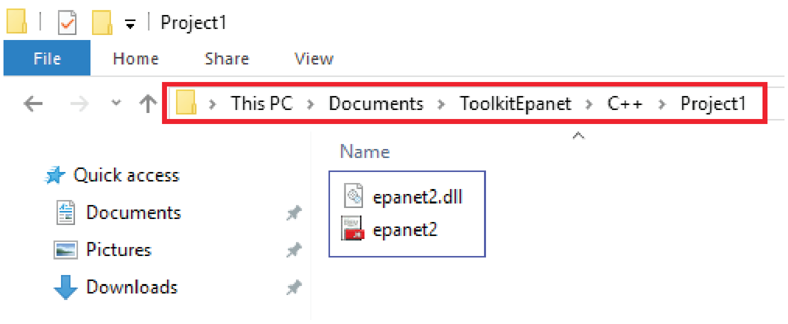


Figure 4.47. Copying the epanet2.h and epanet2.dll files to the Project1 folder

C. Configuring project options using Dev-C++

The first thing we will do is open our Dev-C++ 5.11 application and create a new project. For that we need to use the following steps: File >> New >> Project. A new window will appear in which we will select "Empty Project" from the Basic tab, and we will give our project a name. We will keep the suggested name "Project1" and we will click on the Ok button (Figure 4.48). After that we save our file in the "Project1" folder. Next, we change the name of our first ".cpp" file to ConnectAPIEpanet, by clicking on the Save button, (Figure 4.49). The final look that our project will have using the Dev-C++ application, and the other files created in the folder Project1 can be seen on Figure 4.50.

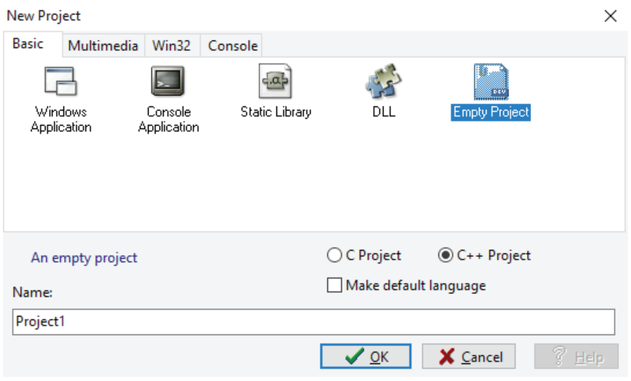
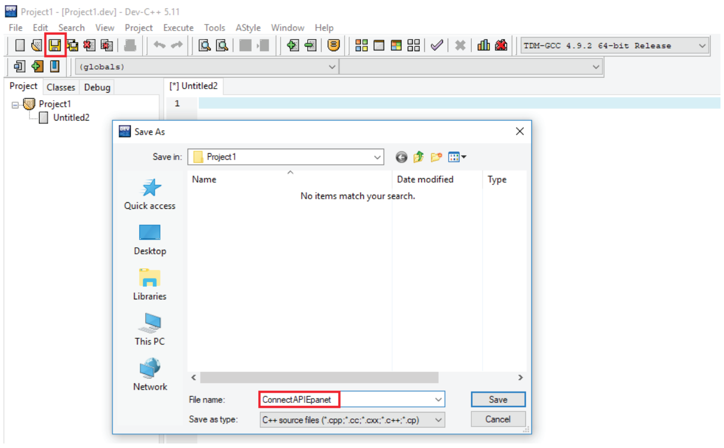


Figure 4.48. Creating a new project using Dev-C++



No items match your search.

File name: ConnectAPIEpanet

Save as type: C++ source files (*.cpp;*.cc;*.cxx;*.c++;*.cp)

Figure 4.49. Saving the ConnectEpanetAPI.cpp file inside the Project1 folder

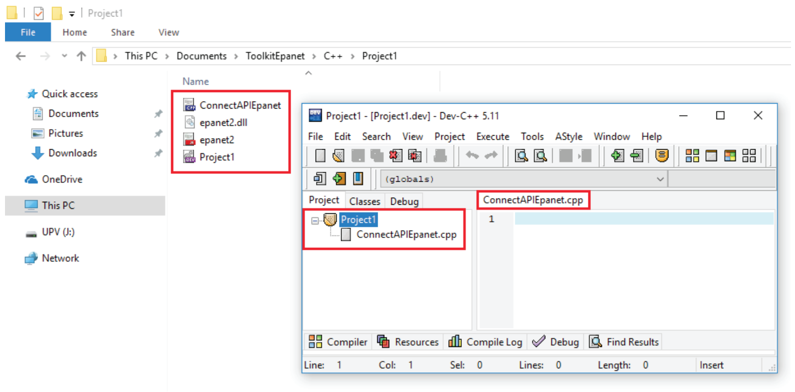
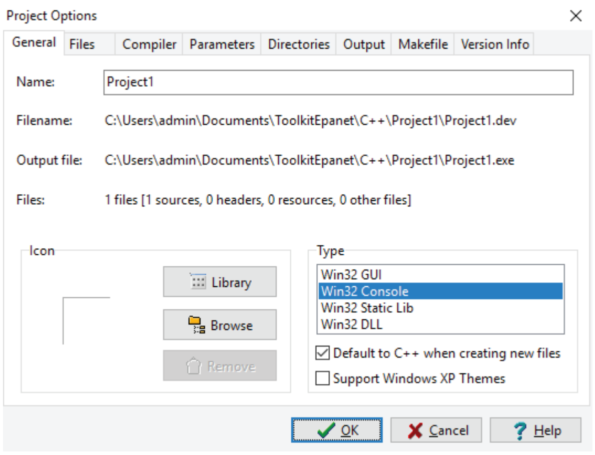

Project1
File Home Share View
This PC › Documents › ToolkitEpanet › C++ › Project1
Name
Quick access
Documents
Pictures
Downloads
OneDrive
This PC
UPV (J:)
Network
ConnectAPIEpanet
epanet2.dll
epanet2
Project1
Project1 - [Project1.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug
ConnectAPIEpanet.cpp
Project1
ConnectAPIEpanet.cpp
Compiler Resources Compile Log Debug Find Results
Line: 1 Col: 1 Sel: 0 Lines: 0 Length: 0 Insert

Figure 4.50. Files generated in the Project1 folder

The next step will be to configure some project options to be able to work with our Epanet library, for that we will go to the Project menu and select Project Options and a dialogue box will appear as can be seen on Figure 4.51. This box has eight tabs in which we'll be modifying some of the default values.



Name: Project1

Filename: C:\Users\admin\Documents\ToolkitEpanet\C++\Project1\Project1.dev

Output file: C:\Users\admin\Documents\ToolkitEpanet\C++\Project1\Project1.exe

Files: 1 files [1 sources, 0 headers, 0 resources, 0 other files]

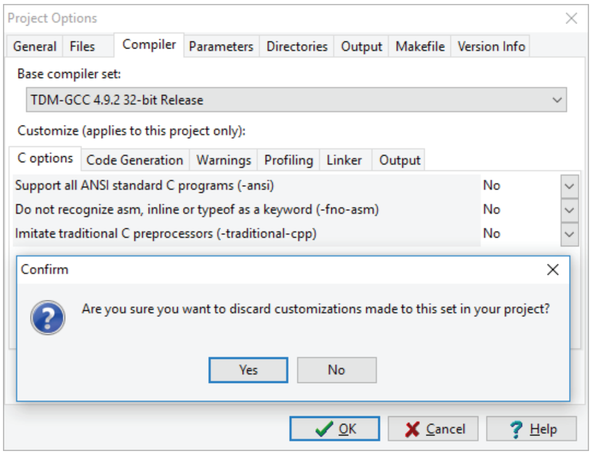
Type: Win32 GUI, [✓ Win32 Console], Win32 Static Lib, Win32 DLL

☑ Default to C++ when creating new files

☐ Support Windows XP Themes

Figure 4.51. "Project Options" window using Dev-C++

In the "Main" and "Files" tabs there will be no adjustment to the default values. In the "Compiler" tab we will select the TDM-GCC 4.9.2 32-bit option Release and confirm using the "Yes" button to the changes made (Figure 4.52). In the "Compiler" tab there are six more tabs where we will only modify the Generate Information parameter in Debug from "No" to "Yes" on the "Linker" tab.



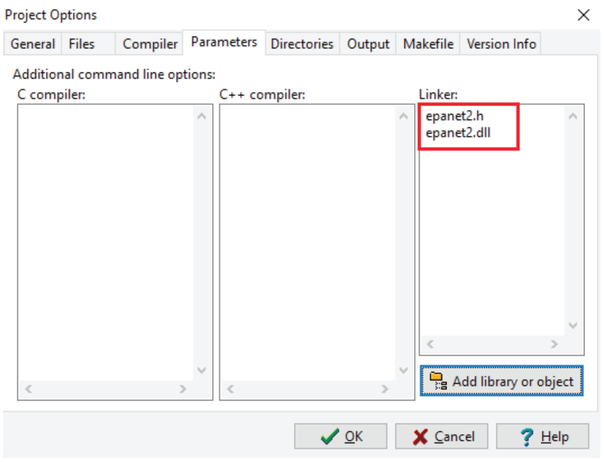
Base compiler set:
TDM-GCC 4.9.2 32-bit Release
Customize (applies to this project only):

Support all ANSI standard C programs (-ansi)	No
Do not recognize asm, inline or typeof as a keyword (-fno-asm)	No
Imitate traditional C preprocessors (-traditional-cpp)	No

Are you sure you want to discard customizations made to this set in your project?

Figure 4.52. Selecting a compiler to use

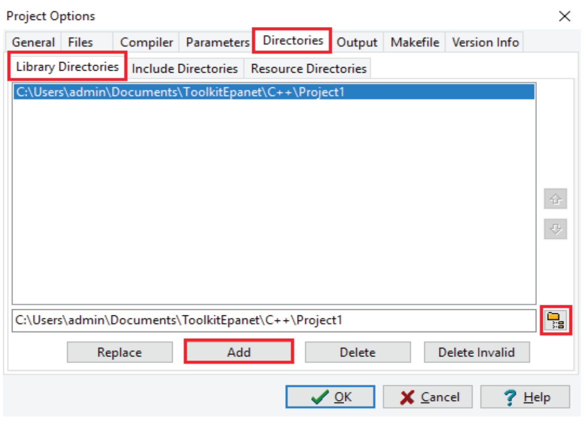
To link the file paths for the epanet2.h and epanet2.dll files, we must go to the "Parameters" tab, place the cursor over the Linker section and look for both files in the project folder, then click on the Add Library button or File and add them to the list, see Figure 4.53.



Additional command line options:
C compiler:
C++ compiler:
Linker:
epanet2.h
epanet2.dll

Figure 4.53. Search and selection of epanet2.dll and epanet2.h files

In the Library and Include directories found in the "Directories" tab, we are going to add the directory where the epanet2.dll and epanet2.h files are saved. Both files are available in the following directory: C:\Users\admin\Documents\ToolkitEpanet\C++\Project1, as displayed in Figure 4.54.

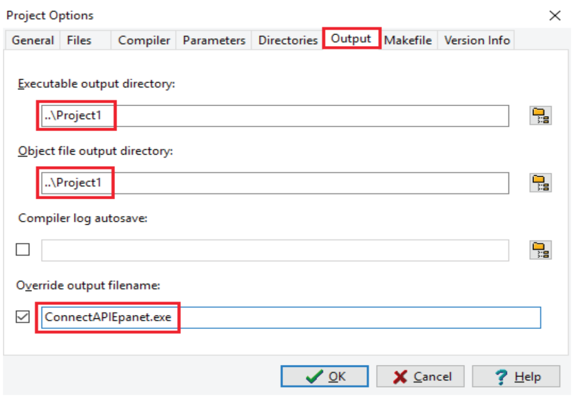


C:\Users\admin\Documents\ToolkitEpanet\C++\Project1

C:\Users\admin\Documents\ToolkitEpanet\C++\Project1

Figure 4.54. Adding a directory to save the epanet2.h and epanet2.dll files

Next, in the "Output" tab, we will look for and select the Project1 folder to save the .exe file and an object file after compiling the code as seen in Figure 4.55. Lastly, we will force the name change of the .exe file to ConnectEpanetAPI.exe, and we will close the Options window to Accept all the changes.



Executable output directory: ..\Project1

Object file output directory: ..\Project1

Compiler log autosave: ☐

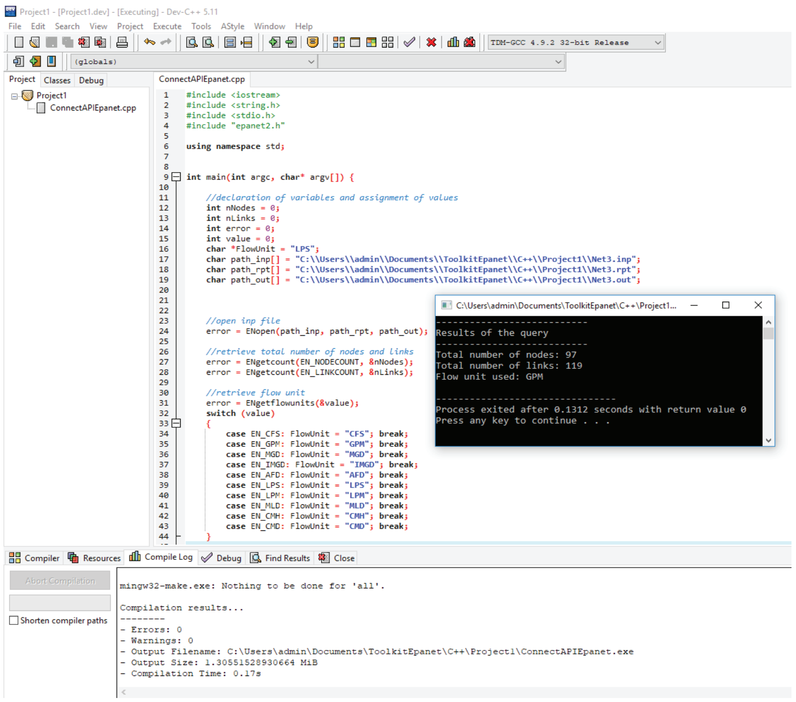
Override output filename: ☑ ConnectAPIEpanet.exe

Figure 4.55. Output tab settings

D. Working with the Epanet Toolkit

After configuring the project options, what follows is to write the lines of code in the C++ language that will permit us to retrieve the total number of junctions and links in the network, as well as the flow unit being used. To compile and run our code we click on the Compile menu (twice), and then we click on Run (only once). In Figure 4.56 we can see the results obtained from the system console.

The project can be downloaded from the following link: https://www.imta.gob.mx/biblioteca/download/?key=241. If it is necessary to run the source code from the previous link directly, keep in mind the required path change of the Project1 folder and modify the file path of the (Net3.inp) network being studied, as well as modify project options using Dev-C++.


Project1 - [Project1.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 32-bit Release
(globals)
Project Classes Debug
Project1
ConnectAPIEpanet.cpp
ConnectAPIEpanet.cpp
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths

```cpp
#include <iostream>
#include <string.h>
#include <stdio.h>
#include "epanet2.h"

using namespace std;


int main(int argc, char* argv[]) {

    //declaration of variables and assignment of values
    int nNodes = 0;
    int nLinks = 0;
    int error = 0;
    int value = 0;
    char *FlowUnit = "LPS";
    char path_inp[] = "C:\\Users\\admin\\Documents\\ToolkitEpanet\\C++\\Project1\\Net3.inp";
    char path_rpt[] = "C:\\Users\\admin\\Documents\\ToolkitEpanet\\C++\\Project1\\Net3.rpt";
    char path_out[] = "C:\\Users\\admin\\Documents\\ToolkitEpanet\\C++\\Project1\\Net3.out";



    //open inp file
    error = ENopen(path_inp, path_rpt, path_out);

    //retrieve total number of nodes and links
    error = ENgetcount(EN_NODECOUNT, &nNodes);
    error = ENgetcount(EN_LINKCOUNT, &nLinks);

    //retrieve flow unit
    error = ENgetflowunits(&value);
    switch (value)
    {
        case EN_CFS: FlowUnit = "CFS"; break;
        case EN_GPM: FlowUnit = "GPM"; break;
        case EN_MGD: FlowUnit = "MGD"; break;
        case EN_IMGD: FlowUnit = "IMGD"; break;
        case EN_AFD: FlowUnit = "AFD"; break;
        case EN_LPS: FlowUnit = "LPS"; break;
        case EN_LPM: FlowUnit = "LPM"; break;
        case EN_MLD: FlowUnit = "MLD"; break;
        case EN_CMH: FlowUnit = "CMH"; break;
        case EN_CMD: FlowUnit = "CMD"; break;
    }
```

C:\Users\admin\Documents\ToolkitEpanet\C++\Project1...

```
---------------------------
Results of the query
---------------------------
Total number of nodes: 97
Total number of links: 119
Flow unit used: GPM

--------------------------------
Process exited after 0.1312 seconds with return value 0
Press any key to continue . . .
```

```
mingw32-make.exe: Nothing to be done for 'all'.

Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\admin\Documents\ToolkitEpanet\C++\Project1\ConnectAPIEpanet.exe
- Output Size: 1.30551528930664 MiB
- Compilation Time: 0.17s
```

Figure 4.56. Source-code compiling result

4.7. Microsoft Office Excel 2016 (VBA)

Microsoft Excel is a spreadsheet application that is part of the Microsoft Office suite. Organized in rows and columns, it is capable of making extensive calculations in seconds through formulas of all kinds. It is very useful for solving financial and accounting operations, with formulas, graphs and an embedded programming language. Since 1993, Excel has included Visual Basic for Applications (VBA), a programming language based on Visual Basic, which adds the ability to automate tasks in Excel and to provide user-defined functions for use in worksheets. VBA is a powerful addition to the application, which includes a complete integrated development environment (IDE) also known as the VBA Editor. Macro recording allows to play code repeatedly, doing the user's tasks in an automated way. Likewise, you can create forms and controls in the worksheet for communication with the user. VBA also supports the use (but not the creation) of ActiveX DLLs (COM).

Many people use it in various fields. Its easy handling, design, and speed make it very competitive when using it at work. Its creator, Charles Simonyi, of Hungarian origin, was the one who supervised the creation of Microsoft Office, along with Richard Brodie.

Excel made its appearance in 1982 when Microsoft released a spreadsheet called Multiplan, which was popular in large systems, but could not surpass its competitor Lotus 1-2-3, which worked very well on personal computers, reason why Microsoft developed a new spreadsheet to compete with Lotus 1-2-3. In 1985, the first version of Excel was released for the Macintosh. The first version for Windows was called Microsoft Excel 2.0 and was launched in 1987. In 1988 Excel exceeded the popularity of Lotus 1-2-3. The first time Excel was introduced in Office was in 1993. Before Excel and Multiplan there was VisiCalc, the first spreadsheet created by Dan Bricklin.

A. Display the "Developer" tab

The *Developer* tab is used for macro and form control creation. This tab is not shown by default in Excel. In order to display it, we click on the menu "File" and select *Options.* The *Excel Options* will be shown where we click on *Customize*

Ribbon. Next we check the *Developer* selection box in the *Main Tabs* section, and accept the change made by clicking on OK. Figure 4.57 shows these steps.

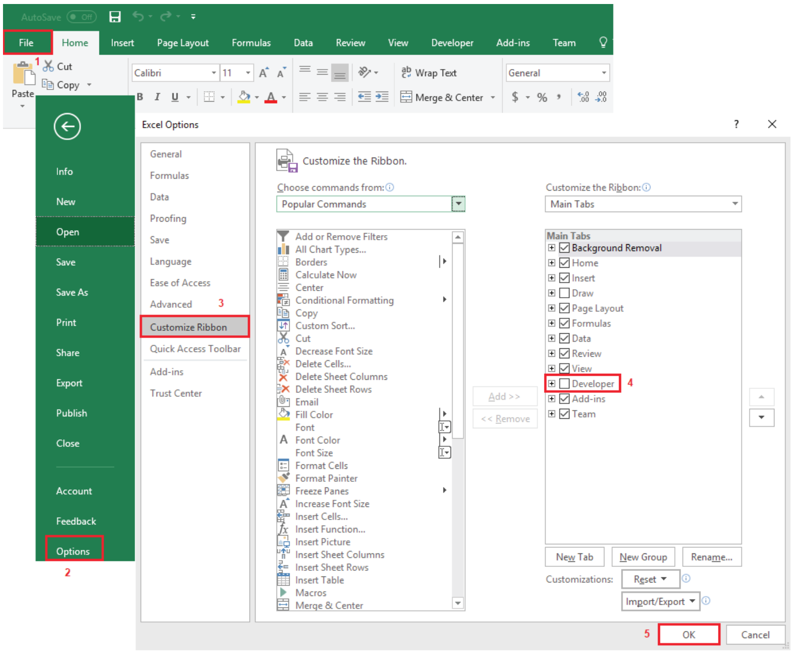


Figure 4.57. Displaying the Developer tab

Another way to reach quickly the *Excel Options* dialog box is to right click *options ribbon* and select *Customize the Ribbon*.

The Developer tab (Figure 4.58) contains the necessary commands for creating macros and also for running previously saved macros. From this tab we can open the Visual Basic Editor where the VBA code can be written.

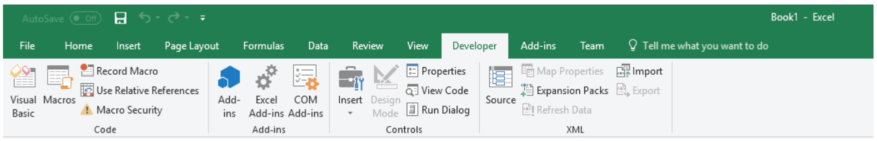


Figure 4.58. Menu Developer and its commands

B. Prepare the epanet2 module

In order to make use of the Epanet library functions, we must declare first the global constants and the functions themselves in a Visual Basic module. In the *Developer* tab we click on *Visual Basic*. The Microsoft VBA editor is thus opened, where we go to the *Insert* menu and click on *Module*. This first module will be classified in the Modules folder and have the name Module1 (Figure 4.59).

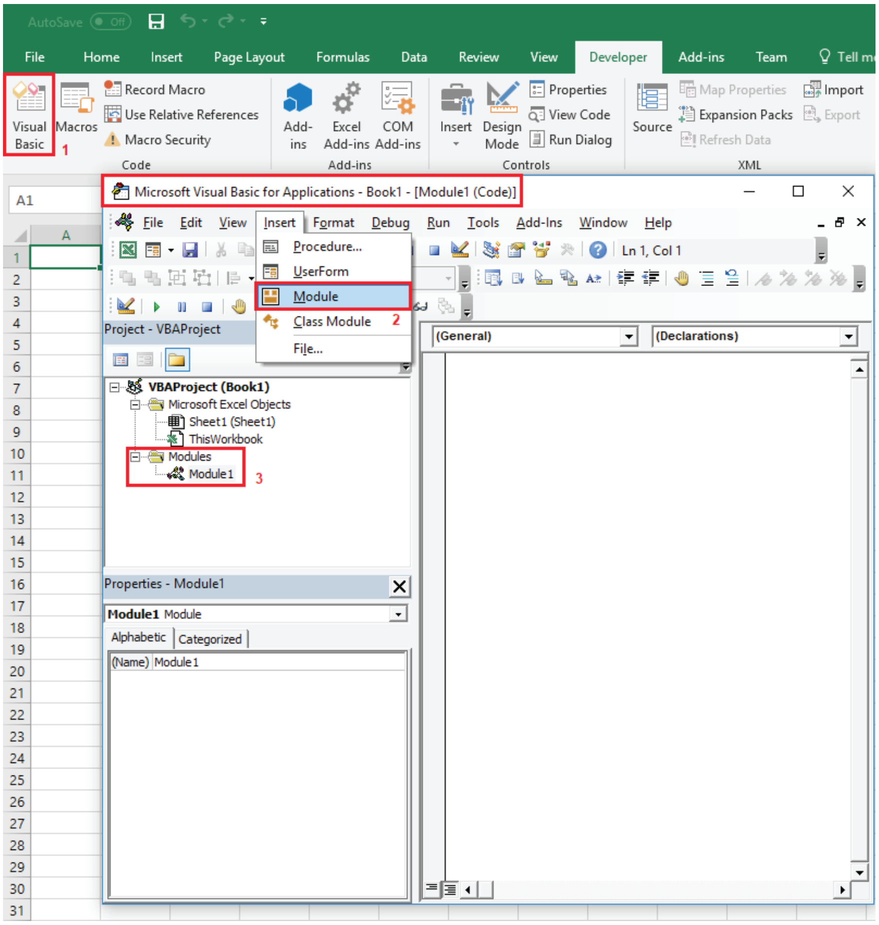


Figure 4.59. Inserting Module1 into Project

Next, we change the Project and module default names. The project will be named ConnectAPIEpanet and the module epanet2 (Figure 4.60).

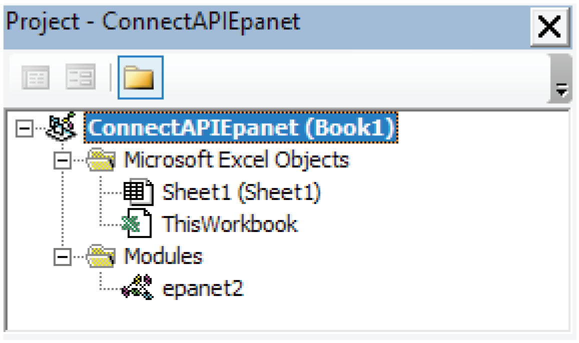
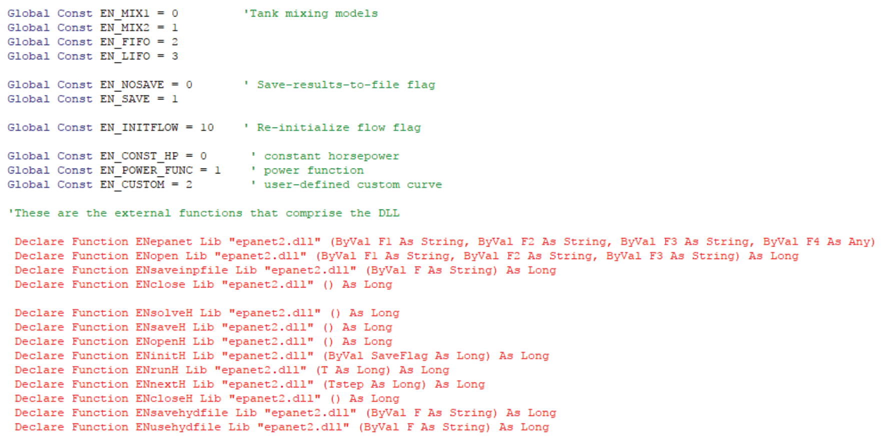


Figure 4.60. Changing the project and module names

The header file epanet2.bas can be downloaded from:
https://github.com/OpenWaterAnalytics/EPANET/blob/master/include/epanet2.bas.

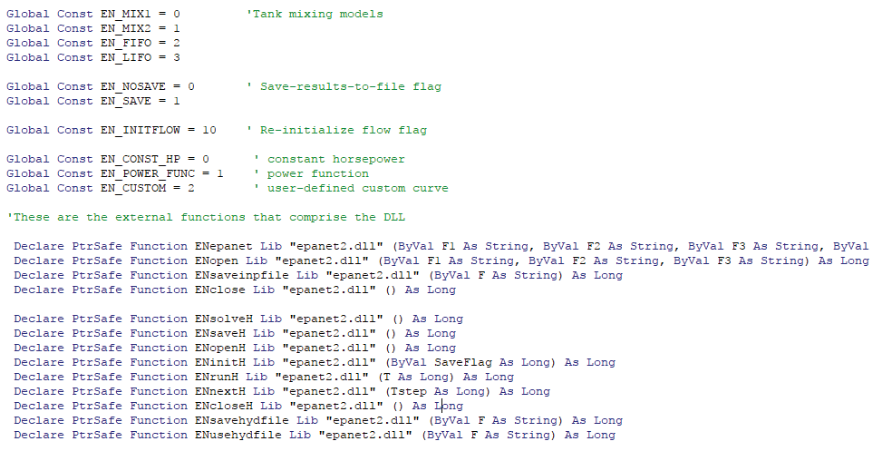
Elad Salomons made some ajustments to that file in order to use the new Epanet library (versión 2.1) from Visual Basic 6.0. In our module epanet2 we copy its content from line 3 to line 233.

The 74 function declarations in the epanet2 module turn red (Figure 4.61). This is because the functions are declared to work on 32-bit Excel. The keyword PtrSafe must be added after the instruction Declare in order to fix the error (Figure 4.62).



```
Global Const EN_MIX1 = 0         'Tank mixing models
Global Const EN_MIX2 = 1
Global Const EN_FIFO = 2
Global Const EN_LIFO = 3

Global Const EN_NOSAVE = 0       ' Save-results-to-file flag
Global Const EN_SAVE = 1

Global Const EN_INITFLOW = 10    ' Re-initialize flow flag

Global Const EN_CONST_HP = 0      ' constant horsepower
Global Const EN_POWER_FUNC = 1    ' power function
Global Const EN_CUSTOM = 2        ' user-defined custom curve

'These are the external functions that comprise the DLL

 Declare Function ENepanet Lib "epanet2.dll" (ByVal F1 As String, ByVal F2 As String, ByVal F3 As String, ByVal F4 As Any)
 Declare Function ENopen Lib "epanet2.dll" (ByVal F1 As String, ByVal F2 As String, ByVal F3 As String) As Long
 Declare Function ENsaveinpfile Lib "epanet2.dll" (ByVal F As String) As Long
 Declare Function ENclose Lib "epanet2.dll" () As Long

 Declare Function ENsolveH Lib "epanet2.dll" () As Long
 Declare Function ENsaveH Lib "epanet2.dll" () As Long
 Declare Function ENopenH Lib "epanet2.dll" () As Long
 Declare Function ENinitH Lib "epanet2.dll" (ByVal SaveFlag As Long) As Long
 Declare Function ENrunH Lib "epanet2.dll" (T As Long) As Long
 Declare Function ENnextH Lib "epanet2.dll" (Tstep As Long) As Long
 Declare Function ENcloseH Lib "epanet2.dll" () As Long
 Declare Function ENsavehydfile Lib "epanet2.dll" (ByVal F As String) As Long
 Declare Function ENusehydfile Lib "epanet2.dll" (ByVal F As String) As Long
```

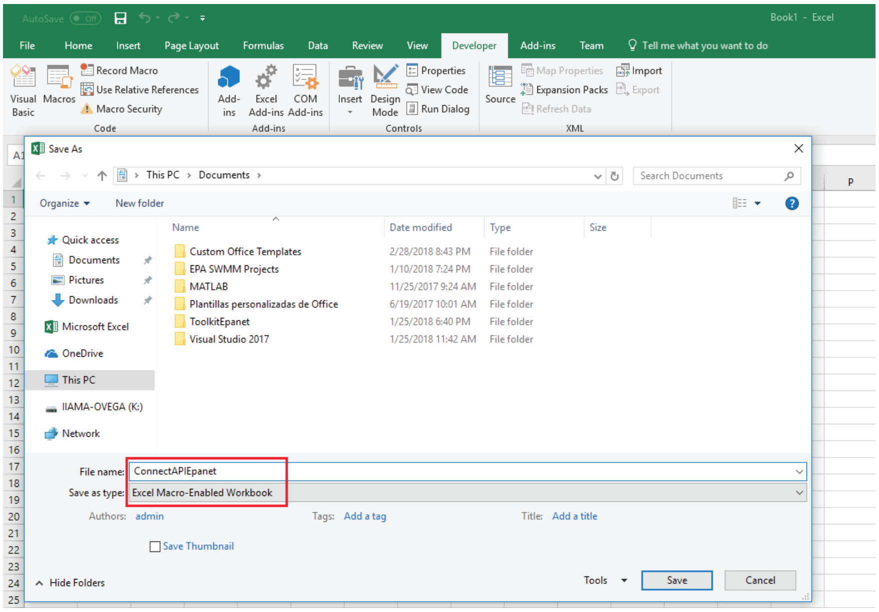
Figure 4.61. Error at declaring the Epanet functions



```
Global Const EN_MIX1 = 0         'Tank mixing models
Global Const EN_MIX2 = 1
Global Const EN_FIFO = 2
Global Const EN_LIFO = 3

Global Const EN_NOSAVE = 0       ' Save-results-to-file flag
Global Const EN_SAVE = 1

Global Const EN_INITFLOW = 10    ' Re-initialize flow flag

Global Const EN_CONST_HP = 0      ' constant horsepower
Global Const EN_POWER_FUNC = 1    ' power function
Global Const EN_CUSTOM = 2        ' user-defined custom curve

'These are the external functions that comprise the DLL

 Declare PtrSafe Function ENepanet Lib "epanet2.dll" (ByVal F1 As String, ByVal F2 As String, ByVal F3 As String, ByVal
 Declare PtrSafe Function ENopen Lib "epanet2.dll" (ByVal F1 As String, ByVal F2 As String, ByVal F3 As String) As Long
 Declare PtrSafe Function ENsaveinpfile Lib "epanet2.dll" (ByVal F As String) As Long
 Declare PtrSafe Function ENclose Lib "epanet2.dll" () As Long

 Declare PtrSafe Function ENsolveH Lib "epanet2.dll" () As Long
 Declare PtrSafe Function ENsaveH Lib "epanet2.dll" () As Long
 Declare PtrSafe Function ENopenH Lib "epanet2.dll" () As Long
 Declare PtrSafe Function ENinitH Lib "epanet2.dll" (ByVal SaveFlag As Long) As Long
 Declare PtrSafe Function ENrunH Lib "epanet2.dll" (T As Long) As Long
 Declare PtrSafe Function ENnextH Lib "epanet2.dll" (Tstep As Long) As Long
 Declare PtrSafe Function ENcloseH Lib "epanet2.dll" () As Long
 Declare PtrSafe Function ENsavehydfile Lib "epanet2.dll" (ByVal F As String) As Long
 Declare PtrSafe Function ENusehydfile Lib "epanet2.dll" (ByVal F As String) As Long
```

Figure 4.62. Correct declaration of the Epanet functions

What follows is to save our Excel file. We write a name, for example *ConnectAPIEpanet*, and select *Excel Macro-Enabled Workbook* as file type (Figure 4.63).



Name	Date modified	Type	Size
Custom Office Templates	2/28/2018 8:43 PM	File folder	
EPA SWMM Projects	1/10/2018 7:24 PM	File folder	
MATLAB	11/25/2017 9:24 AM	File folder	
Plantillas personalizadas de Office	6/19/2017 10:01 AM	File folder	
ToolkitEpanet	1/25/2018 6:40 PM	File folder	
Visual Studio 2017	1/25/2018 11:42 AM	File folder	

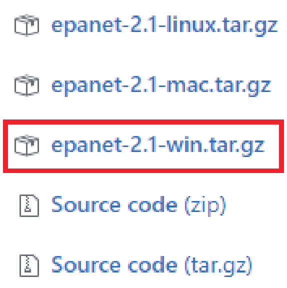
File name: ConnectAPIEpanet
Save as type: Excel Macro-Enabled Workbook

Figure 4.63. Saving the ConnectAPIEpanet file

C. Copying epanet2.dll to the folder System

Errors have been fixed and new capabilities added in the new Epanet library version 2.1. All the documentation about that new version can be consulted from the WaterAnalytics web site: http://wateranalytics.org/EPANET.

The Epanet libraries, for 32 bit and 64 bit computers, can be downloaded from the following link: https://github.com/OpenWaterAnalytics/EPANET/releases (Figure 4.64).


epanet-2.1-linux.tar.gz

epanet-2.1-mac.tar.gz

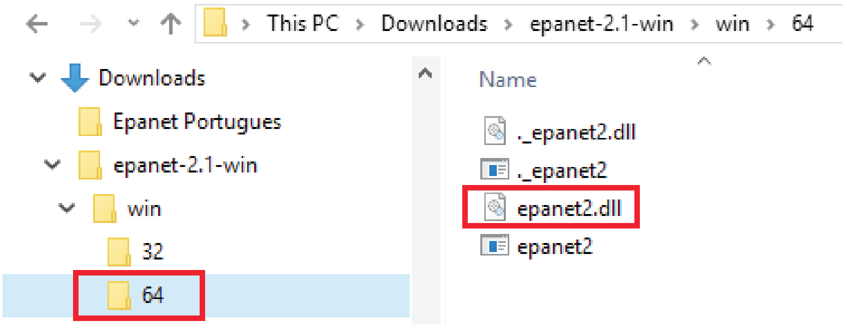
epanet-2.1-win.tar.gz

Source code (zip)

Source code (tar.gz)

Figure 4.64. Downloading the Epanet-2.1-win.tar.gz file

After decompressing the file, we search for the folder named 64, and copy the epanet2.dll file there to the following folder: C:\Windows\System (Figure 4.65).



._epanet2.dll
._epanet2
epanet2.dll
epanet2

Figure 4.65. "epanet2.dll" 64-bit file

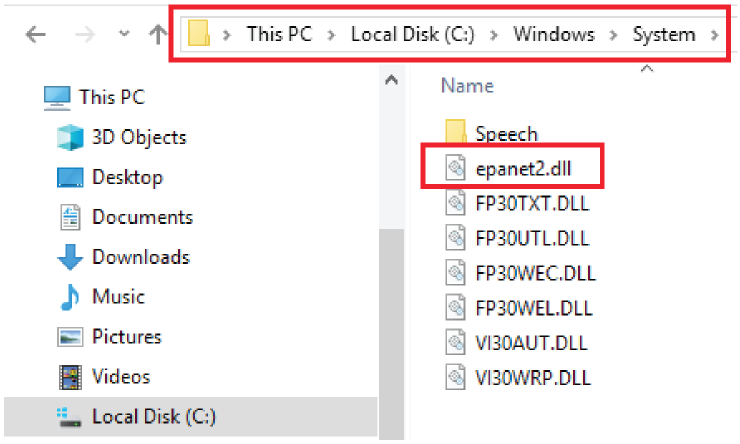
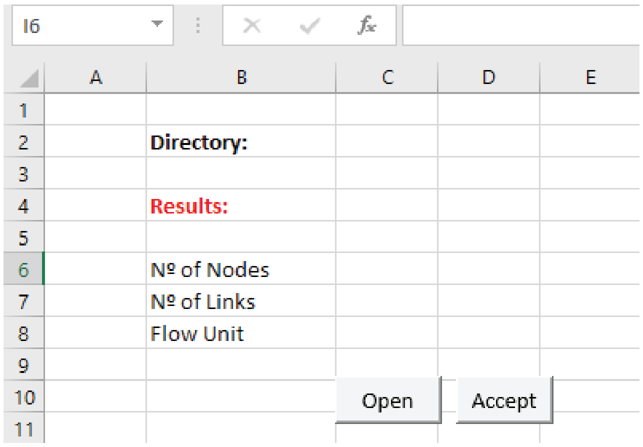


Figure 4.66. Copying epanet2.dll to the folder System

D. Worksheet preparation

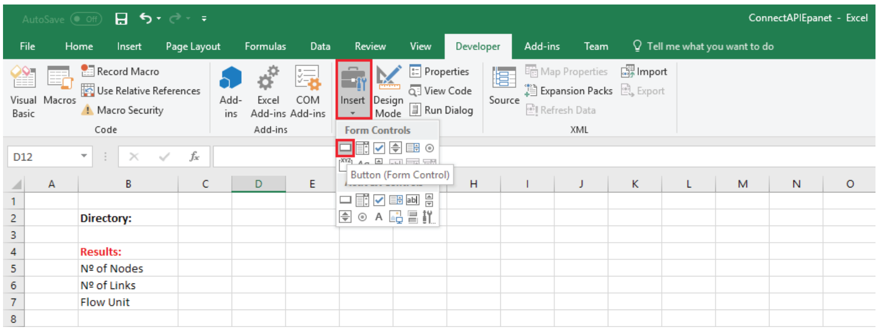
We reopen our Excel Macro-Enabled ConnectAPIEpanet Workbook (in case we closed it) and add some tags and two buttons, as shown in Figure 4.67.



	A	B	C	D	E
1					
2		**Directory:**			
3					
4		**Results:**			
5					
6		Nº of Nodes			
7		Nº of Links			
8		Flow Unit			
9					
10					
11					

Figure 4.67. Excel worksheet

In order to add the two buttons shown in Figure 4.67, we open the *Developer* tab and click on *Insert*. A dialog box is displayed (Figure 4.68) where we select the control *Button (Form Control)*. At left clicking on the cell "C10" to draw the control, a floating dialog box appears where we are asked to assign a macro, but by now no macro will be assigned. The same process is applied for the second button, or the Copy and Paste option can be used.



	A	B	C	D	E
1					
2		**Directory:**			
3					
4		**Results:**			
5		Nº of Nodes			
6		Nº of Links			
7		Flow Unit			
8					

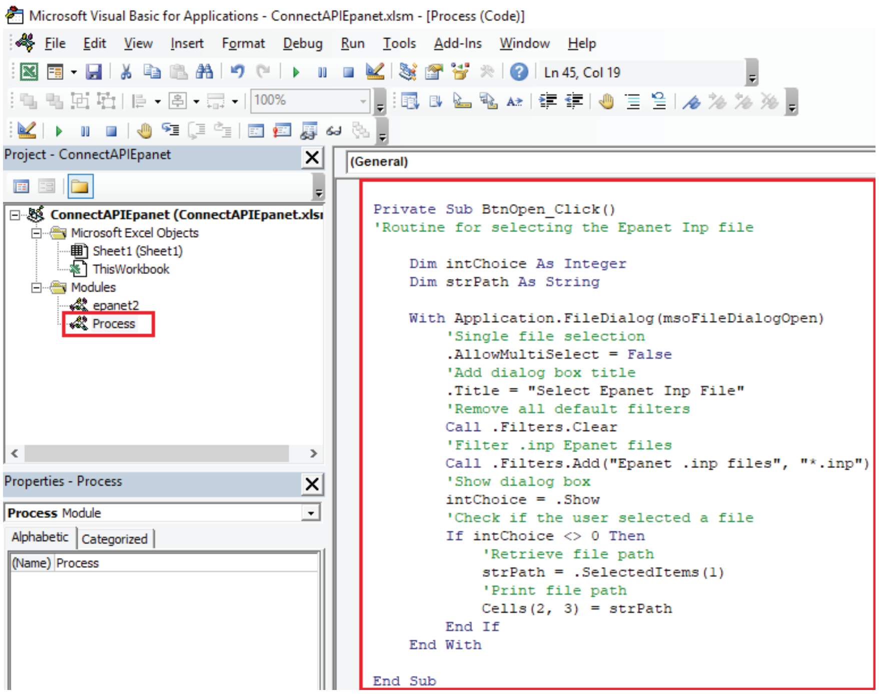
Figure 4.68. Inserting the button controls

The buttons added exhibit the name *Button* by default, that can be changed by right clicking on each of them and selecting Edit Text.

E. Coding inside Microsoft VBA

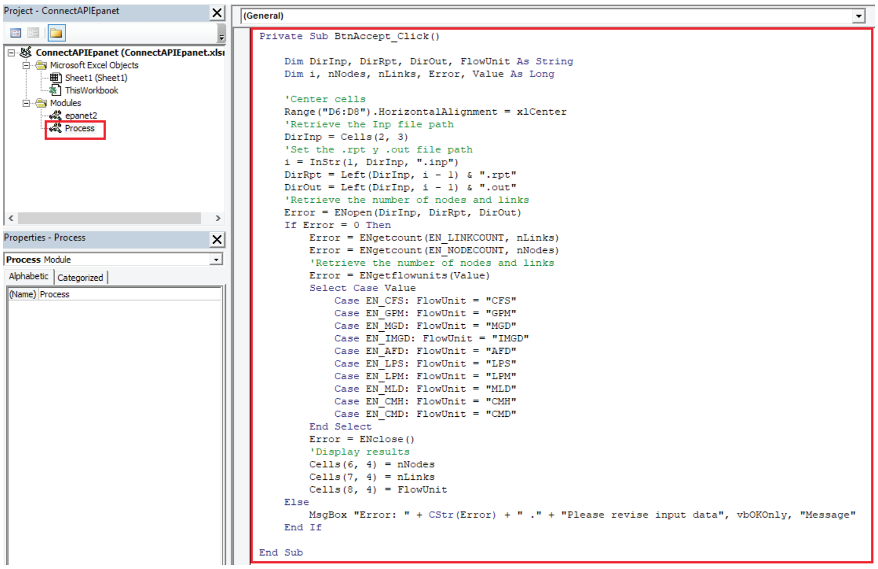
In order to write lines of code, that will be associated with the *Open* and *Accept* buttons, we must add first a new module. We go to the *Visual Basic* option from the *Developer* menu. The *Visual Basic Editor* is opened, where we click on *Insert* and add the new module and change its default name to *Processes*.

In the module, the routines *BtnOpen_Click* and *BtnAccept_Click* are written, that will be assigned to buttons *Open* and *Accept* respectively. The code for each of them can be seen in Figure 4.69 and Figure 4.70. In order to assign these routines to the corresponding buttons, we right click on each of them and select *Assign Macro...*. Next, the name of each routine to be associated is written in the dialog box (Figure 4.71).


Microsoft Visual Basic for Applications - ConnectAPIEpanet.xlsm - [Process (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 45, Col 19
100%
Project - ConnectAPIEpanet
ConnectAPIEpanet (ConnectAPIEpanet.xlsm)
Microsoft Excel Objects
Sheet1 (Sheet1)
ThisWorkbook
Modules
epanet2
Process
Properties - Process
Process Module
Alphabetic
Categorized
(Name) Process
(General)

```
Private Sub BtnOpen_Click()
'Routine for selecting the Epanet Inp file

    Dim intChoice As Integer
    Dim strPath As String

    With Application.FileDialog(msoFileDialogOpen)
        'Single file selection
        .AllowMultiSelect = False
        'Add dialog box title
        .Title = "Select Epanet Inp File"
        'Remove all default filters
        Call .Filters.Clear
        'Filter .inp Epanet files
        Call .Filters.Add("Epanet .inp files", "*.inp")
        'Show dialog box
        intChoice = .Show
        'Check if the user selected a file
        If intChoice <> 0 Then
            'Retrieve file path
            strPath = .SelectedItems(1)
            'Print file path
            Cells(2, 3) = strPath
        End If
    End With

End Sub
```

Figure 4.69. Routine BtnOpen_Click



```
Private Sub BtnAccept_Click()

    Dim DirInp, DirRpt, DirOut, FlowUnit As String
    Dim i, nNodes, nLinks, Error, Value As Long

    'Center cells
    Range("D6:D8").HorizontalAlignment = xlCenter
    'Retrieve the Inp file path
    DirInp = Cells(2, 3)
    'Set the .rpt y .out file path
    i = InStr(1, DirInp, ".inp")
    DirRpt = Left(DirInp, i - 1) & ".rpt"
    DirOut = Left(DirInp, i - 1) & ".out"
    'Retrieve the number of nodes and links
    Error = ENopen(DirInp, DirRpt, DirOut)
    If Error = 0 Then
        Error = ENgetcount(EN_LINKCOUNT, nLinks)
        Error = ENgetcount(EN_NODECOUNT, nNodes)
        'Retrieve the number of nodes and links
        Error = ENgetflowunits(Value)
        Select Case Value
            Case EN_CFS: FlowUnit = "CFS"
            Case EN_GPM: FlowUnit = "GPM"
            Case EN_MGD: FlowUnit = "MGD"
            Case EN_IMGD: FlowUnit = "IMGD"
            Case EN_AFD: FlowUnit = "AFD"
            Case EN_LPS: FlowUnit = "LPS"
            Case EN_LPM: FlowUnit = "LPM"
            Case EN_MLD: FlowUnit = "MLD"
            Case EN_CMH: FlowUnit = "CMH"
            Case EN_CMD: FlowUnit = "CMD"
        End Select
        Error = ENclose()
        'Display results
        Cells(6, 4) = nNodes
        Cells(7, 4) = nLinks
        Cells(8, 4) = FlowUnit
    Else
        MsgBox "Error: " + CStr(Error) + " ." + "Please revise input data", vbOKOnly, "Message"
    End If

End Sub
```

Figure 4.70. Routine BtnAccept_Click

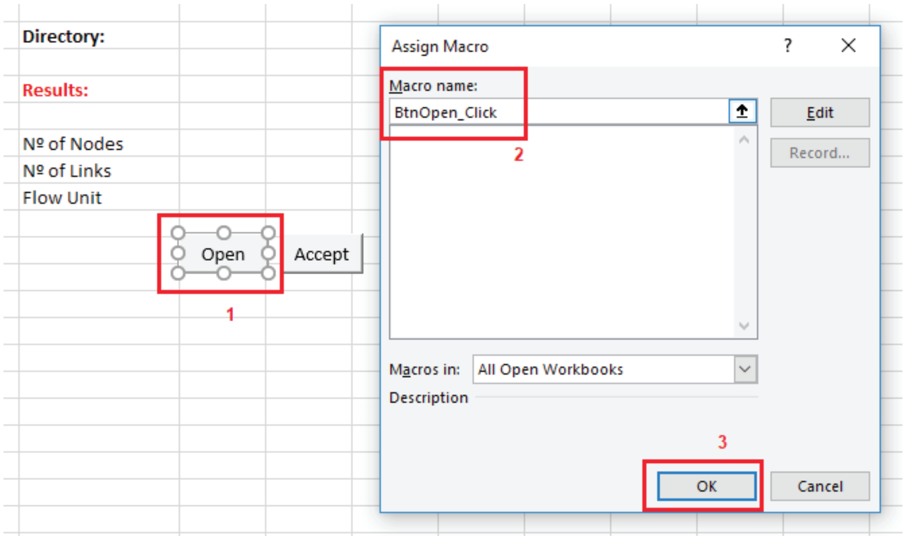
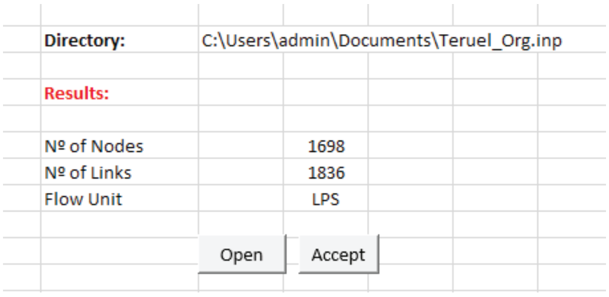


Figure 4.71. Assign the macro BtnOpen_Click to the Open button

F. Running the macros

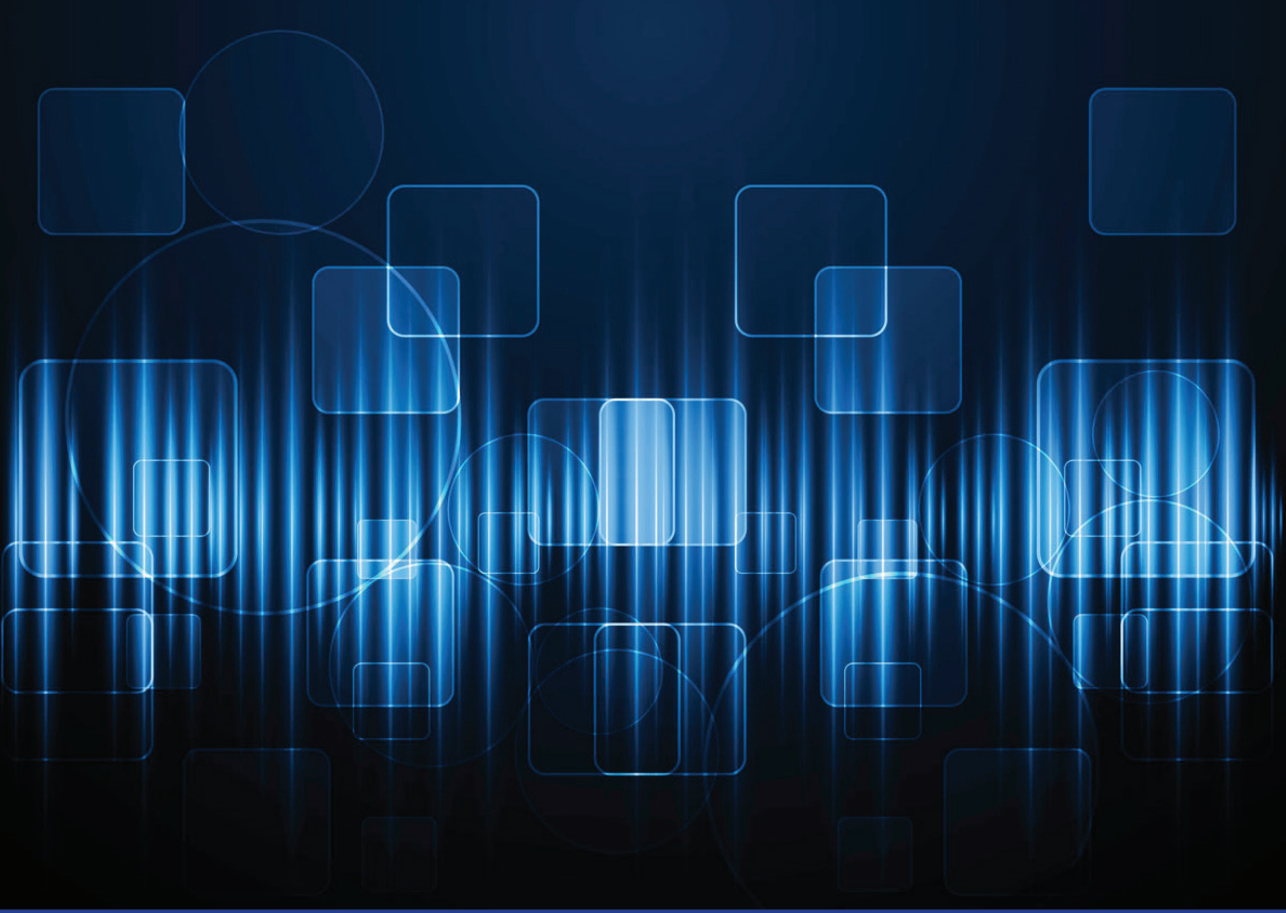
The macros are executed in the following order. First the button *Open* is selected. Its execution displays a dialog box to navigate through the computer and external storage devices, and select an Epanet Inp file. The path of the selected file will be written on the worksheet. The Accept button runs the Epanet library functions for determining the total number of nodes (demand nodes, reservoirs and tanks) and links (pipes, pumps and valves), as well as the flow unit (Figure 4.72).

To download the file ConnectAPIEpanet, that contains the macros, click on the following link: https://goo.gl/AzBzvA.



Directory:	C:\Users\admin\Documents\Teruel_Org.inp
Results:	
Nº of Nodes	1698
Nº of Links	1836
Flow Unit	LPS

Figure 4.72. Result of applying the macros



5 Practical Exercises

The following is a presentation of five practical cases in which the majority of the Epanet library v2.00.12 functions will be used using Microsoft Visual Studio 2017 (Visual Basic .NET). These exercises do not pretend to resolve operational improvement problems for water supply systems like sectorization, simplification, system reliability, optimal design, optimization etc., rather, they seek only to teach the proper use of the functions in such a way that combining them with the native functions that correspond to each of the programming languages, will allow for more potent tools that can be designed in favor of urban hydraulics.

Source code will not be written for each one of the practical cases, since that could take up a large volume of pages, thus, only a step-by-step sequence will be used, and the functions will be mentioned as they appear in this book. The network model that will be used in all the proposed tasks will be the Net3.inp network. All the source code generated can be downloaded using links that will be left at the end of every task and will be gone further-into for better understanding.

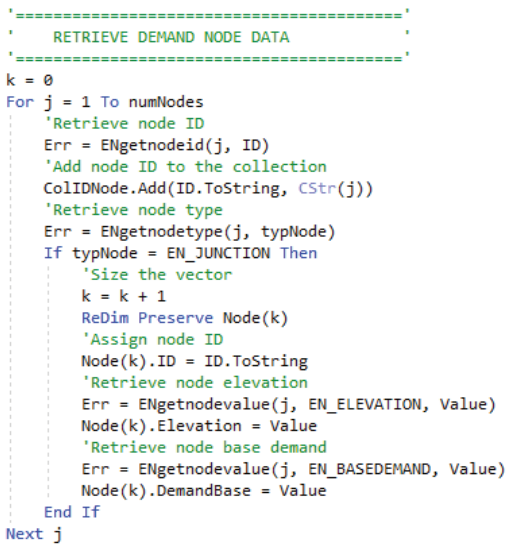
The tasks that will be carried out in this chapter are: retrieving and modifying network parameters, running a hydraulic and water quality simulation, retrieving results and generating a report, calculating pressure at a specific junction for an extended period of time, and changing the link direction.

A. Retrieving and modifying network parameters

If we want to retrieve, for example, the ID of a demand node and its base demand, ENgetnodetype, ENgetnodeid, and ENgetnodevalue will be needed. In the case of a pipe, for retrieving its ID, start and end node, length, and its status, ENgetlinkid, ENgetlinknodes, y ENgetlinkvalue will be used. Before using these functions, however, it is necessary to call the ENopen function to open the Epanet inp file and after using it, ENclose must be called to close the input file and free up the memory used on the computer.

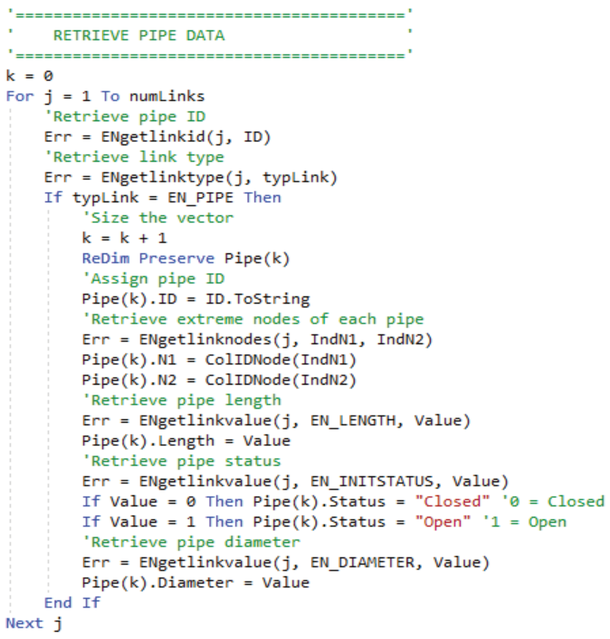
To save the drawn-up information, a user-defined vector data type will be created in which the Epanet inp file data will be stored. In Figure 5.1 and Figure 5.2 part of the source code used to retrieve node and pipe data can be seen, which has been added to the code from the ConnectEpanetAPI example given in the previous section.

It is important to verify that the information being saved in our vector is correct. For that, an interruption point will be inserted near the end of the event upon clicking the Accept button, specifically on the line of code error = ENclose(). We can see the results of the data stored in the DemandNode and Pipes vectors added to the inspection window in Figure 5.3.



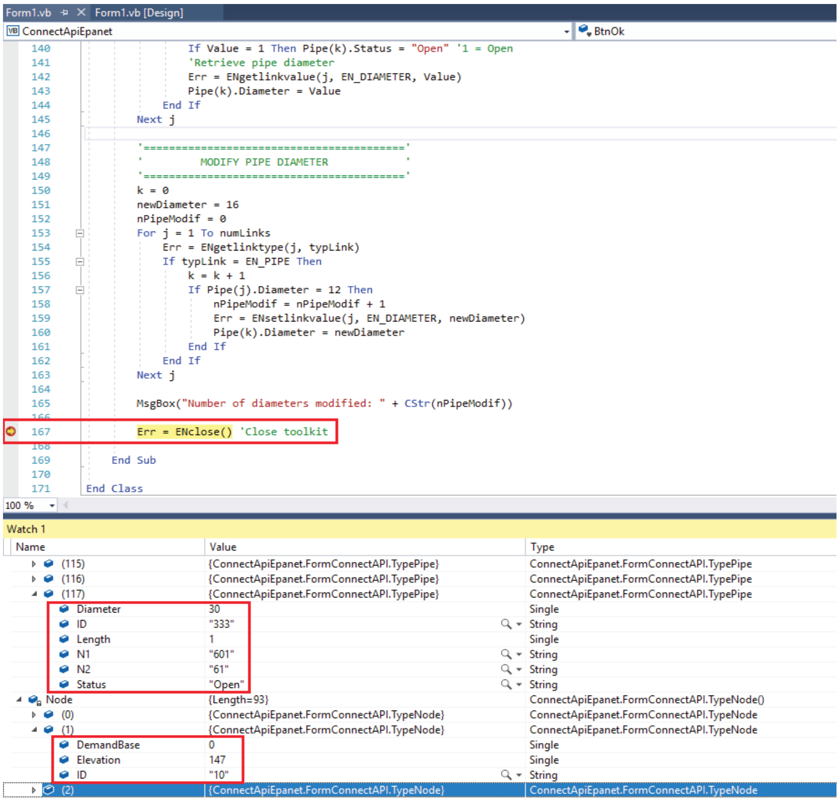
```
'=========================================='
'    RETRIEVE DEMAND NODE DATA             '
'=========================================='
k = 0
For j = 1 To numNodes
    'Retrieve node ID
    Err = ENgetnodeid(j, ID)
    'Add node ID to the collection
    ColIDNode.Add(ID.ToString, CStr(j))
    'Retrieve node type
    Err = ENgetnodetype(j, typNode)
    If typNode = EN_JUNCTION Then
        'Size the vector
        k = k + 1
        ReDim Preserve Node(k)
        'Assign node ID
        Node(k).ID = ID.ToString
        'Retrieve node elevation
        Err = ENgetnodevalue(j, EN_ELEVATION, Value)
        Node(k).Elevation = Value
        'Retrieve node base demand
        Err = ENgetnodevalue(j, EN_BASEDEMAND, Value)
        Node(k).DemandBase = Value
    End If
Next j
```

Figure 5.1. Functions for retrieving data from demand nodes



```
'=========================================='
'    RETRIEVE PIPE DATA                    '
'=========================================='
k = 0
For j = 1 To numLinks
    'Retrieve pipe ID
    Err = ENgetlinkid(j, ID)
    'Retrieve link type
    Err = ENgetlinktype(j, typLink)
    If typLink = EN_PIPE Then
        'Size the vector
        k = k + 1
        ReDim Preserve Pipe(k)
        'Assign pipe ID
        Pipe(k).ID = ID.ToString
        'Retrieve extreme nodes of each pipe
        Err = ENgetlinknodes(j, IndN1, IndN2)
        Pipe(k).N1 = ColIDNode(IndN1)
        Pipe(k).N2 = ColIDNode(IndN2)
        'Retrieve pipe length
        Err = ENgetlinkvalue(j, EN_LENGTH, Value)
        Pipe(k).Length = Value
        'Retrieve pipe status
        Err = ENgetlinkvalue(j, EN_INITSTATUS, Value)
        If Value = 0 Then Pipe(k).Status = "Closed" '0 = Closed
        If Value = 1 Then Pipe(k).Status = "Open" '1 = Open
        'Retrieve pipe diameter
        Err = ENgetlinkvalue(j, EN_DIAMETER, Value)
        Pipe(k).Diameter = Value
    End If
Next j
```

Figure 5.2. Functions used to retrieve pipe data



```
                If Value = 1 Then Pipe(k).Status = "Open" '1 = Open
                'Retrieve pipe diameter
                Err = ENgetlinkvalue(j, EN_DIAMETER, Value)
                Pipe(k).Diameter = Value
            End If
        Next j

        '=========================================='
        '          MODIFY PIPE DIAMETER            '
        '=========================================='
        k = 0
        newDiameter = 16
        nPipeModif = 0
        For j = 1 To numLinks
            Err = ENgetlinktype(j, typLink)
            If typLink = EN_PIPE Then
                k = k + 1
                If Pipe(j).Diameter = 12 Then
                    nPipeModif = nPipeModif + 1
                    Err = ENsetlinkvalue(j, EN_DIAMETER, newDiameter)
                    Pipe(k).Diameter = newDiameter
                End If
            End If
        Next j

        MsgBox("Number of diameters modified: " + CStr(nPipeModif))

        Err = ENclose() 'Close toolkit

    End Sub

End Class
```

Name	Value	Type
(115)	{ConnectApiEpanet.FormConnectAPI.TypePipe}	ConnectApiEpanet.FormConnectAPI.TypePipe
(116)	{ConnectApiEpanet.FormConnectAPI.TypePipe}	ConnectApiEpanet.FormConnectAPI.TypePipe
(117)	{ConnectApiEpanet.FormConnectAPI.TypePipe}	ConnectApiEpanet.FormConnectAPI.TypePipe
Diameter	30	Single
ID	"333"	String
Length	1	Single
N1	"601"	String
N2	"61"	String
Status	"Open"	String
Node	{Length=93}	ConnectApiEpanet.FormConnectAPI.TypeNode()
(0)	{ConnectApiEpanet.FormConnectAPI.TypeNode}	ConnectApiEpanet.FormConnectAPI.TypeNode
(1)	{ConnectApiEpanet.FormConnectAPI.TypeNode}	ConnectApiEpanet.FormConnectAPI.TypeNode
DemandBase	0	Single
Elevation	147	Single
ID	"10"	String
(2)	{ConnectApiEpanet.FormConnectAPI.TypeNode}	ConnectApiEpanet.FormConnectAPI.TypeNode

Figure 5.3. Reviewing data stored in the DemandNode and Pipes vectors

To carry out any modification of the parameters that define the characteristics or the mode of operation of our network model, the following functions should be used with the ENset prefix (ex. ENsetnodevalue, ENsetlinkvalue, ENsetpattern, etc.). As an example, we are going to change the diameters of the pipes on the Net3 network model, from 12" to 16" (inches). Moreover, the changes made to the Pipes().Diameter will be saved if a list of IDs is desired for the pipes with the new diameters. From the Epanet application we make a query that will permit us to visualize and know the total number of pipes to be modified. We can do this through the following sequence of commands: View menu >> Query (Figure 5.4).

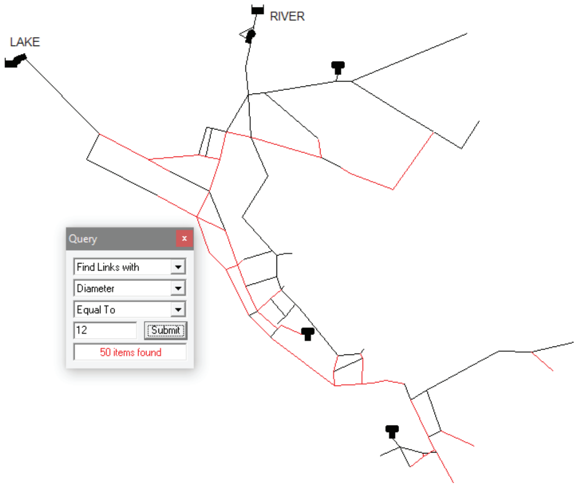


Figure 5.4. Consulting the number of pipes with a 12" diameter using Epanet

If we wish to modify the diameters, it will be necessary to select first the entire network from the Edit menu: Edit >> Select Region and the Edit >> Group Edit and configure the options in the dialogue box to make the changes (Figure 5.5). Using the Epanet library functions we also get the same results, see Figure 5.6.

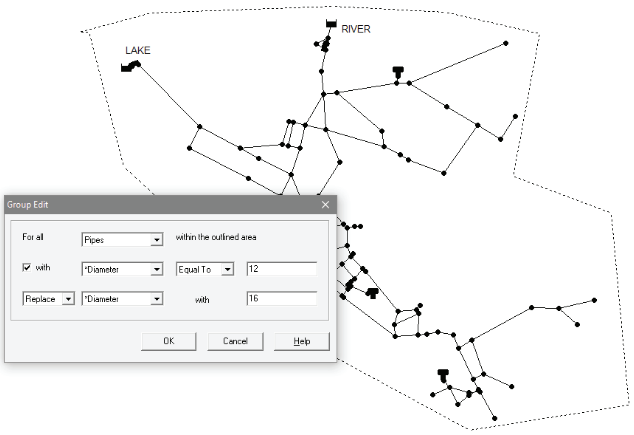
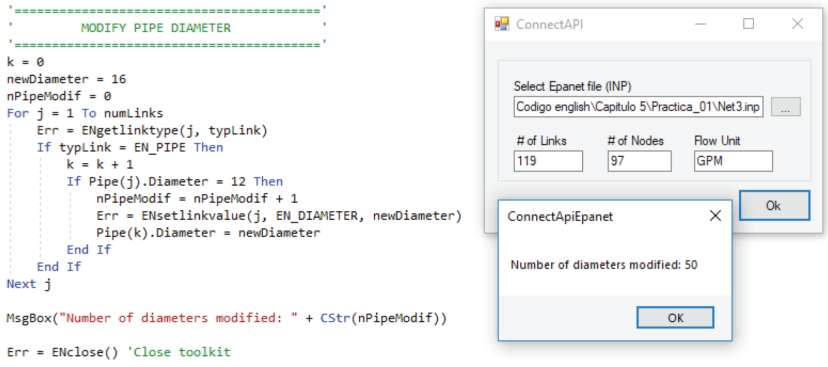


Figure 5.5. Modifying pipe diameters with the Epanet Group Edit tool



```
'=========================================='
'              MODIFY PIPE DIAMETER               '
'=========================================='
k = 0
newDiameter = 16
nPipeModif = 0
For j = 1 To numLinks
    Err = ENgetlinktype(j, typLink)
    If typLink = EN_PIPE Then
        k = k + 1
        If Pipe(j).Diameter = 12 Then
            nPipeModif = nPipeModif + 1
            Err = ENsetlinkvalue(j, EN_DIAMETER, newDiameter)
            Pipe(k).Diameter = newDiameter
        End If
    End If
Next j

MsgBox("Number of diameters modified: " + CStr(nPipeModif))

Err = ENclose() 'Close toolkit
```

ConnectAPI

Select Epanet file (INP): Codigo english\Capitulo 5\Practica_01\Net3.inp

# of Links: 119
# of Nodes: 97
Flow Unit: GPM

ConnectApiEpanet

Number of diameters modified: 50

Figure 5.6. Modifying pipe diameters with the Epanet library function

The source code used for this first task can be downloaded off of the following link: https://www.imta.gob.mx/biblioteca/download/?key=237. The module file, Epanet library (v2.00.12) and the Net3.inp can be found in the compressed folder.

B. Running a hydraulic and water-quality simulation model

The most important parts of the Epanet library are the hydraulic simulation model and the water-quality simulation model. We can get the results of a hydraulic simulation in two ways. One is using the ENsolveH function that enables us to run a complete extended period analysis, without having access to the intermediate results, the second way is by using the sequence of ENopenH, ENinitH, ENrunH, ENnextH, ENcloseH, functions to run the simulation step-by-step.

The first method is recommended if a water-quality analysis will also be carried out. With this method, the results of the hydraulic computations for each one of the time steps are always saved to a hydraulic results file (binary file). The second method is more appropriate when accessing the intermediate results is desired throughout the all computational time steps or if it is intended to carry out many runs in an efficient manner. In this case, only one function call to

ENopenH will be used to start the process, the following is a succession of calls to ENinitH-ENrunH-ENnextH to run each stage of the analysis, and finally the ENcloseH function will be called to close the hydraulic module. If a steady-state analysis is to be run, it is not necessary to use the ENnextH function.

In the case of water-quality analysis, the hydraulics results need to have already been generated through running a hydraulic simulation or importing a previously-saved results file. As is the case of hydraulic analysis, there are two forms to carry out a water-quality simulation. The first is by using the ENSolveQ function to run a complete extended-period water-quality analysis, without having access to the intermediate result files, and the second is using the sequence of ENopenQ-ENinitQ-ENrunQ-ENnextQ-ENcloseQ functions to carry out a step-by-step simulation. (Swapping ENnextQ for ENstepQ, a simulation can be made advancing one time step at a time).

Having briefly described how to use the Epanet library functions for the hydraulic and water-quality calculation, we will go on to write lines of code with the aforementioned methods.

Hydraulic simulation

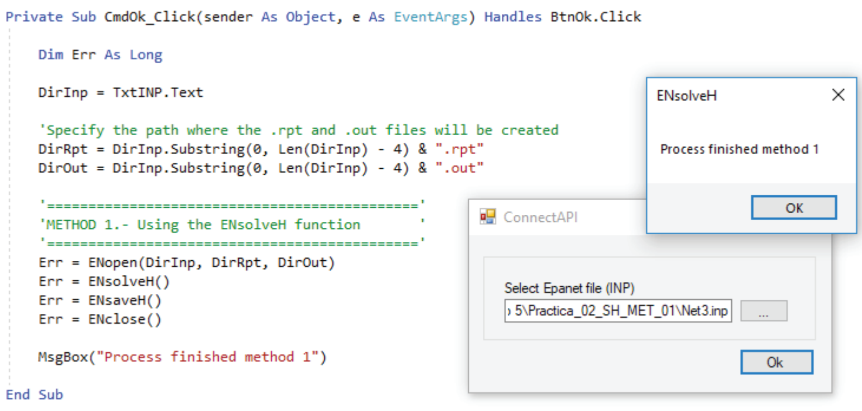
1) Method 1. Using the ENsolveH function

To run a complete hydraulic simulation it is necessary to use the following sequence of functions: ENopen-ENsolveH-ENsaveH-ENclose. The ENopen function receives three input parameters. If you want to have the hydraulic results, it will be necessary to specify that in one of its parameters and use the ENsaveH function to save them, if you neglect to do so, said file with the *.out extension will be erased upon calling the ENclose function.

The Hydraulic Results File (*.out) is a binary file, used for saving the results of a hydraulic analysis. In this file, the results for all the regular time steps set in the clause REPORT TIMESTEP in the [TIMES] section (default 1 hour) are saved.

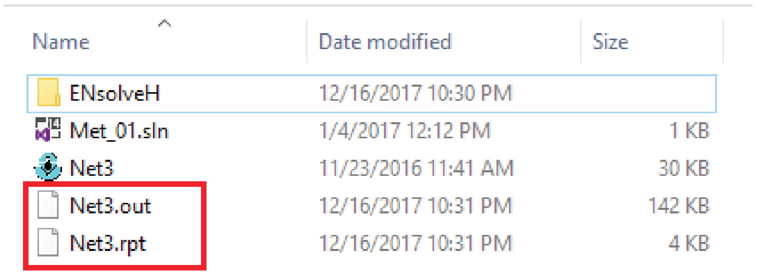
All the same, the intermediate results in which hydraulic changes take place (e. g. opening or closing of pumps or valves due to some rule-based control) are not saved.

In Figure 5.7 we can see the functions employed to carry out a complete simulation. After running the code, two files will have been created with the network file name along with the *.rpt and *.out file extensions in the same folder where the Epanet inp file is (Figure 5.8).



```
Private Sub CmdOk_Click(sender As Object, e As EventArgs) Handles BtnOk.Click

    Dim Err As Long

    DirInp = TxtINP.Text

    'Specify the path where the .rpt and .out files will be created
    DirRpt = DirInp.Substring(0, Len(DirInp) - 4) & ".rpt"
    DirOut = DirInp.Substring(0, Len(DirInp) - 4) & ".out"

    '============================================='
    'METHOD 1.- Using the ENsolveH function      '
    '============================================='
    Err = ENopen(DirInp, DirRpt, DirOut)
    Err = ENsolveH()
    Err = ENsaveH()
    Err = ENclose()

    MsgBox("Process finished method 1")

End Sub
```

Figure 5.7. Use of the ENsolveH and ENsaveH Epanet library functions



Name	Date modified	Size
ENsolveH	12/16/2017 10:30 PM	
Met_01.sln	1/4/2017 12:12 PM	1 KB
Net3	11/23/2016 11:41 AM	30 KB
Net3.out	12/16/2017 10:31 PM	142 KB
Net3.rpt	12/16/2017 10:31 PM	4 KB

Figure 5.8. Net3.rpt and Net3.out files created upon finishing the process

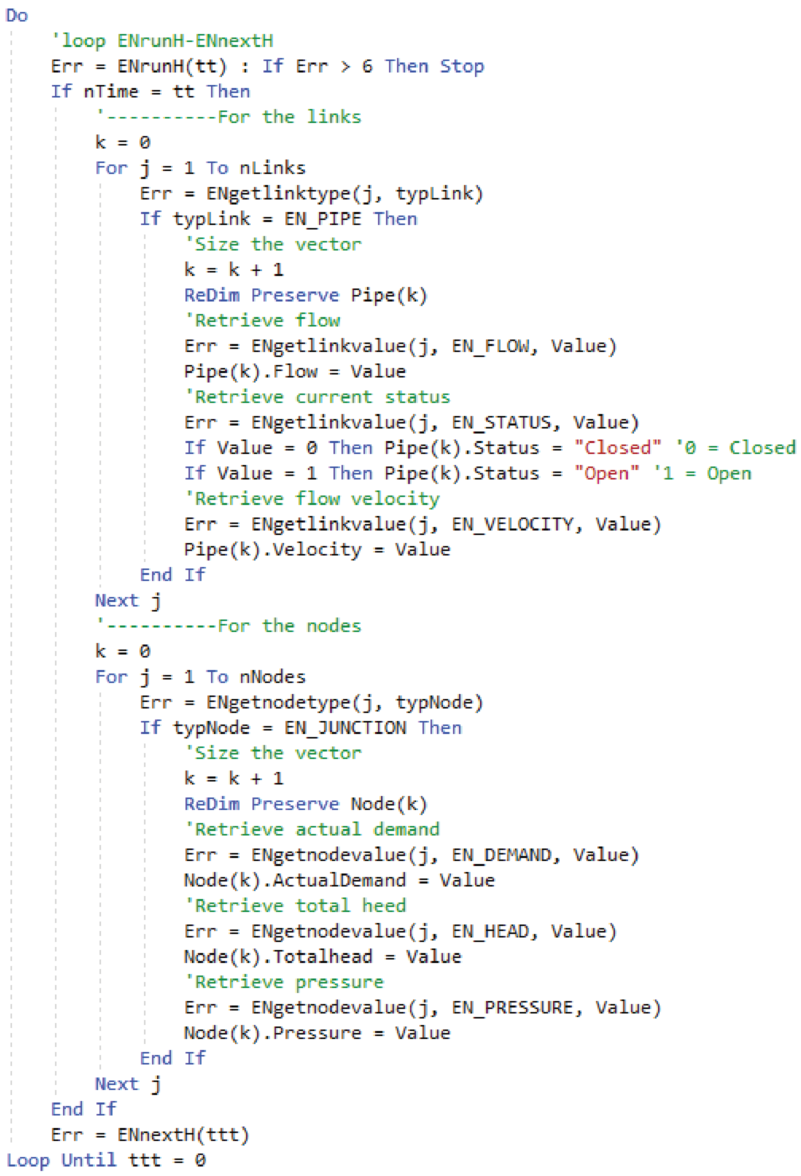
If we wish to save the flow results for all the computational time steps, including those for the intermediate time steps, with the purpose of using them in a water quality analysis, we must indicate it, by using the HYDRAULICS USE filename in the [OPTIONS] section of the Input file, or making a ENusehydfile call function. The source code can be downloaded off of the following link: https://www.imta.gob.mx/biblioteca/download/?key=235.

2) Method 2. Using the ENopenH-ENinitH-ENrunH-ENnextH-ENcloseH functions.

For a better understanding of the use of the group of functions destined to control the simulation process, the following exercise is suggested. Calculate the flow, velocity and status of pipes, as well as the actual demand, the total hydraulic head and the pressure at all demand nodes at 14:00 h.

The Net3 network model has the following configuration in the [TIMES] section: duration 24 hours, 1 hour hydraulic computational time step, 1 hour report interval, Report Start Time at 0 hours.

The previously mentioned demand node and pipe properties at 14:00 h must be obtained. The variable Time = 14*3600.0# determines the instant of time to be captured. In Figure 5.9 an important part of the source code can be seen where the ENrunH-ENnextH functions are used to carry out a step-by-step simulation and capture the desired results at 14:00 h. To download the complete source code, click on the following link: https://www.imta.gob.mx/biblioteca/download/?key=236.



```
Do
    'loop ENrunH-ENnextH
    Err = ENrunH(tt) : If Err > 6 Then Stop
    If nTime = tt Then
        '----------For the links
        k = 0
        For j = 1 To nLinks
            Err = ENgetlinktype(j, typLink)
            If typLink = EN_PIPE Then
                'Size the vector
                k = k + 1
                ReDim Preserve Pipe(k)
                'Retrieve flow
                Err = ENgetlinkvalue(j, EN_FLOW, Value)
                Pipe(k).Flow = Value
                'Retrieve current status
                Err = ENgetlinkvalue(j, EN_STATUS, Value)
                If Value = 0 Then Pipe(k).Status = "Closed" '0 = Closed
                If Value = 1 Then Pipe(k).Status = "Open" '1 = Open
                'Retrieve flow velocity
                Err = ENgetlinkvalue(j, EN_VELOCITY, Value)
                Pipe(k).Velocity = Value
            End If
        Next j
        '----------For the nodes
        k = 0
        For j = 1 To nNodes
            Err = ENgetnodetype(j, typNode)
            If typNode = EN_JUNCTION Then
                'Size the vector
                k = k + 1
                ReDim Preserve Node(k)
                'Retrieve actual demand
                Err = ENgetnodevalue(j, EN_DEMAND, Value)
                Node(k).ActualDemand = Value
                'Retrieve total heed
                Err = ENgetnodevalue(j, EN_HEAD, Value)
                Node(k).Totalhead = Value
                'Retrieve pressure
                Err = ENgetnodevalue(j, EN_PRESSURE, Value)
                Node(k).Pressure = Value
            End If
        Next j
    End If
    Err = ENnextH(ttt)
Loop Until ttt = 0
```

Figure 5.9. Step-by-step hydraulic simulation with the Epanet library

Water-quality simulation

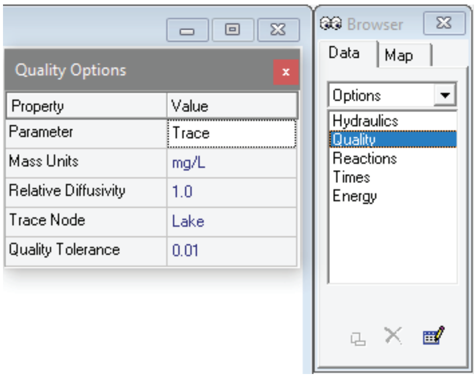
Epanet was born out of the intention to predict the fate of chemical substances transported by the flow in a water distribution network.

Aside from analyzing chemical substance transport, Epanet makes it possible to study other phenomena related to water quality, such as mixing of water from different sources, water age and source tracing, chlorine residual decay, growth of chlorination by-products, the fate of a contaminant introduced to the network, etc.

Before running a water-quality simulation, it is necessary to already have generated the hydraulic results via running a hydraulic simulation or importing previously-saved hydraulic results file.

Just as was the case for hydraulic analysis, there are two ways to carry out a water-quality analysis. We can use the ENsolveQ function or the group of ENopenQ-ENinitQ-ENrunQ-ENnextQ-ENcloseQ functions.

To know how to use both ways, the following exercise is proposed. In this task we are going to determine the fraction of flow that arrives to each one of our junctions in the Net3 network, coming from the Lake reservoir (source tracing analysis). For that we have to verify that the input values in the water quality options section are correct. In Figure 5.10 we can see the information that needs to be entered to run a source tracing analysis.



Quality Options

Property	Value
Parameter	Trace
Mass Units	mg/L
Relative Diffusivity	1.0
Trace Node	Lake
Quality Tolerance	0.01

Figure 5.10. Introducing water quality parameters in Epanet

1) Method 1. Using the ENsolveQ function

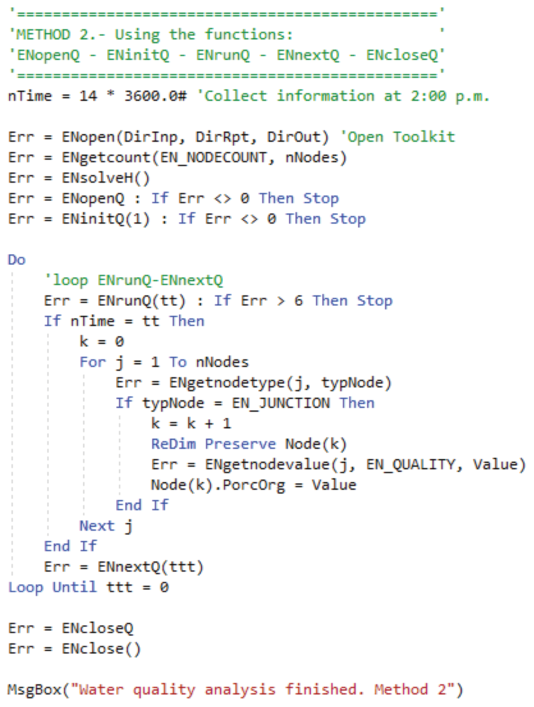
The ENsolveQ function runs a complete water-quality simulation, saving the results at the regular time intervals in the Epanet binary results file (*.out), without having access to intermediate results. In Figure 5.11 the sequence of functions to be used jointly with the ENsolveQ function is shown. To download the complete source code, click on the following link: https://www.imta.gob.mx/biblioteca/download/?key=233.

```
Private Sub BtnOk_Click(sender As Object, e As EventArgs) Handles BtnOk.Click

    Dim Err As Long

    DirInp = TxtINP.Text

    'Specify the path where the .rpt and .out files will be created
    DirRpt = DirInp.Substring(0, Len(DirInp) - 4) & ".rpt"
    DirOut = DirInp.Substring(0, Len(DirInp) - 4) & ".out"

    '=============================================='
    'METHOD 1.- Using the ENsolveQ function        '
    '=============================================='
    Err = ENopen(DirInp, DirRpt, DirOut)
    Err = ENsolveH()
    Err = ENsaveH()
    Err = ENsolveQ()
    Err = ENclose()

    MsgBox("Water quality analysis finished. Use of ENsolveQ")

End Sub
```

Figure 5.11. Use of the ENsolveQ Epanet library function

2) Method 2. Using the ENopenQ-ENinitQ-ENrunQ-ENnextQ-ENcloseQ functions.

To apply the group of functions meant for water-quality analysis, the following exercise is proposed. Determine the fraction of flow coming from the "Lake" reservoir to all of the network nodes at 14:00 hours. In Figure 5.12, the core part of the corresponding code can be seen. To download the complete code, click on the following link https://www.imta.gob.mx/biblioteca/download/?key=234.



```
'=============================================='
'METHOD 2.- Using the functions:              '
'ENopenQ - ENinitQ - ENrunQ - ENnextQ - ENcloseQ'
'=============================================='
nTime = 14 * 3600.0# 'Collect information at 2:00 p.m.

Err = ENopen(DirInp, DirRpt, DirOut) 'Open Toolkit
Err = ENgetcount(EN_NODECOUNT, nNodes)
Err = ENsolveH()
Err = ENopenQ : If Err <> 0 Then Stop
Err = ENinitQ(1) : If Err <> 0 Then Stop

Do
    'loop ENrunQ-ENnextQ
    Err = ENrunQ(tt) : If Err > 6 Then Stop
    If nTime = tt Then
        k = 0
        For j = 1 To nNodes
            Err = ENgetnodetype(j, typNode)
            If typNode = EN_JUNCTION Then
                k = k + 1
                ReDim Preserve Node(k)
                Err = ENgetnodevalue(j, EN_QUALITY, Value)
                Node(k).PorcOrg = Value
            End If
        Next j
    End If
    Err = ENnextQ(ttt)
Loop Until ttt = 0

Err = ENcloseQ
Err = ENclose()

MsgBox("Water quality analysis finished. Method 2")
```

Figure 5.12. Step-by-step water quality simulation using the Epanet library

C. Retrieving computed results and generating a report

The ENgetnodevalue and ENgetlinkvalue functions, which we have used in previous exercises, can be used to retrieve the calculated hydraulic and water-quality results. In Table 5.1 the keywords used for retrieving element values in a water distribution network using Epanet are given.

Table 5.1. Keywords to retrieve values after a simulation

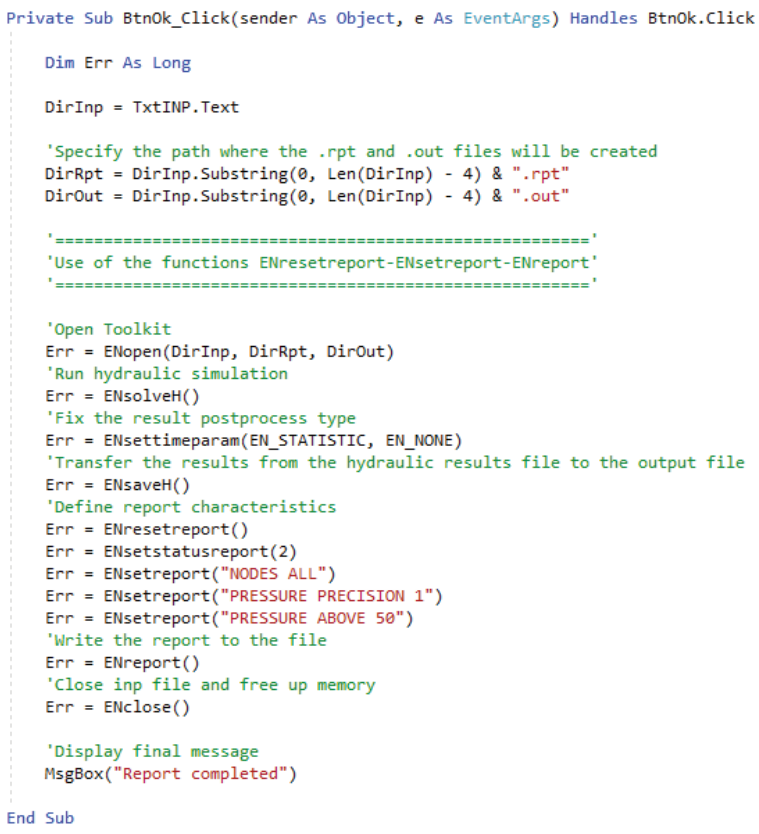
For Nodes	For Links
EN_DEMAND (Actual demand) EN_HEAD (Hydraulic head) EN_PRESSURE (Pressure) EN_QUALITY (Actual quality) EN_SOURCEMASS (Mass flow rate per minute of a chemical source)	EN_FLOW (Flow rate) EN_VELOCITY (Flow velocity) EN_HEADLOSS (Headloss) EN_STATUS (Actual link status) EN_SETTING (Roughness for pipes, actual speed for pumps, actual setting for valves)

The syntax for retrieving the values for an element in the network after running a simulation is as follows: error = ENgetnodevalue (i, keyword, j), where error is a numerical value and in the case that everything goes well, will return zero. The "i" parameter is an index, that is to say, a position that corresponds to the node in the Epanet inp file (which can be obtained using the ENgetnodeindex or ENgetlinkindex functions), the "keyword" parameter is a code that is used to specify the parameter to be recovered according to the kind of element (see Table 5.1), and finally, the "j" parameter, is the variable where the calculated value is to be stored.

The Epanet library has some functions that help to generate a results report with a proprietary format. Although the possibility of writing our own reports with the help of the previous functions in conjunction with other functions particular to the programming language itself is always there, we can simplify the writing of a personalized report, down to just the desired variables through the use of some of the Toolkit functions. The ENsetreport function is used to define the format of a report, while the ENreport function generates said report. This last function should be called only after having done a hydraulic or water quality analysis.

In Figure 5.13 the core part of the code is shown, in which a report is created with a list of all the network junctions whose pressure exceeds 50 psi. The complete code is available for download on the following link: https://www.imta.gob.mx/biblioteca/download/?key=238.

The results will be saved in the results file with the *.rpt extension.



```
Private Sub BtnOk_Click(sender As Object, e As EventArgs) Handles BtnOk.Click

    Dim Err As Long

    DirInp = TxtINP.Text

    'Specify the path where the .rpt and .out files will be created
    DirRpt = DirInp.Substring(0, Len(DirInp) - 4) & ".rpt"
    DirOut = DirInp.Substring(0, Len(DirInp) - 4) & ".out"

    '======================================================='
    'Use of the functions ENresetreport-ENsetreport-ENreport'
    '======================================================='

    'Open Toolkit
    Err = ENopen(DirInp, DirRpt, DirOut)
    'Run hydraulic simulation
    Err = ENsolveH()
    'Fix the result postprocess type
    Err = ENsettimeparam(EN_STATISTIC, EN_NONE)
    'Transfer the results from the hydraulic results file to the output file
    Err = ENsaveH()
    'Define report characteristics
    Err = ENresetreport()
    Err = ENsetstatusreport(2)
    Err = ENsetreport("NODES ALL")
    Err = ENsetreport("PRESSURE PRECISION 1")
    Err = ENsetreport("PRESSURE ABOVE 50")
    'Write the report to the file
    Err = ENreport()
    'Close inp file and free up memory
    Err = ENclose()

    'Display final message
    MsgBox("Report completed")

End Sub
```

Figure 5.13. Functions that help to generate a results report

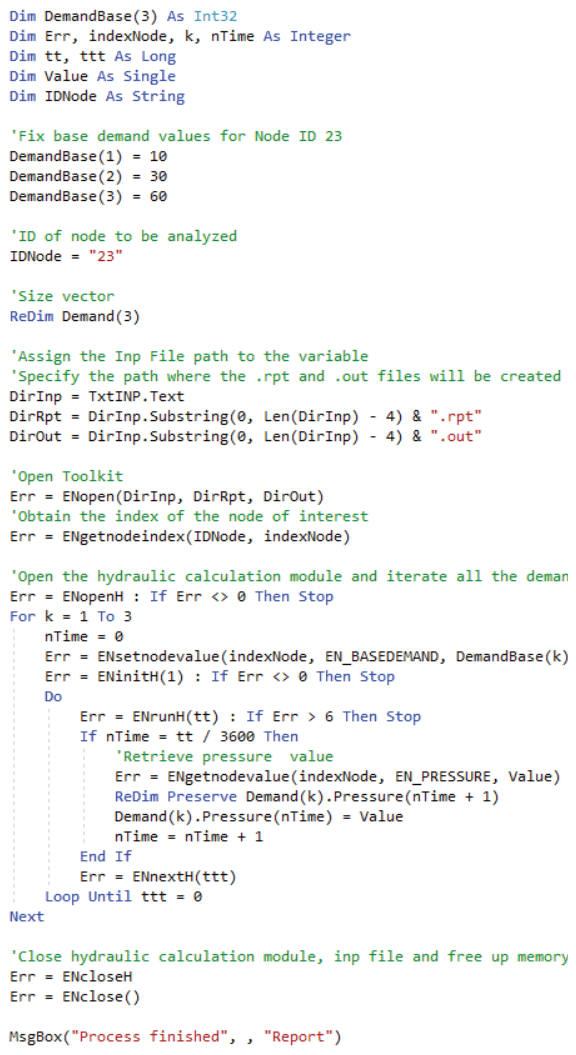
The ENsettimeparam function allows us to set a time parameter value. In this case, all the results for the calculated instants have been specified to take place at regular intervals. The ENresetreport function eliminates any format command that comes before the [REPORT] section of the Epanet input file, or that has been set through the ENsetreport function. The ENsetreport function sets the formatting commands for customizing the results report. The formatting commands are the same as the ones used in the [REPORT] section from the Epanet input file. The ENreport makes a report in text format with the results of the simulation and saves it in the results report file (*rpt). There are other functions in the Epanet library such as ENsaveinpfile, ENgeterror, ENwriteline, ENsavehydfile, ENusehyfile, which can be used according to each simulation case. A description for each of them along with other functions that have been used here can be found in the help file from the Epanet Toolkit.

D. Calculate the effect of demand variation on node pressure

To find a solution we will use a case study with the Net1_SI.inp network model. In this exercise it will be necessary to find the resulting pressures at junction "23", for an entire simulation period (24 hours), as well as for three different values of the base demand for same node.

One solution is to use the following sequence of functions, ENopenH-ENinitH-ENrunH-ENnextH-ENcloseH, and to save the computed pressure values in the vector as they appear. The resulting pressures can be gotten using the ENgetnodevalue function. Another option is to get the pressure using a result report file in which only the node to be analyzed is set. In this case, the first solution has been opted for.

The core part of the code for the exercise's solution can be seen in Figure 5.14. This code, as well as the network model, are available for complete download on the following link: https://www.imta.gob.mx/biblioteca/download/?key=239.



```
Dim DemandBase(3) As Int32
Dim Err, indexNode, k, nTime As Integer
Dim tt, ttt As Long
Dim Value As Single
Dim IDNode As String

'Fix base demand values for Node ID 23
DemandBase(1) = 10
DemandBase(2) = 30
DemandBase(3) = 60

'ID of node to be analyzed
IDNode = "23"

'Size vector
ReDim Demand(3)

'Assign the Inp File path to the variable
'Specify the path where the .rpt and .out files will be created
DirInp = TxtINP.Text
DirRpt = DirInp.Substring(0, Len(DirInp) - 4) & ".rpt"
DirOut = DirInp.Substring(0, Len(DirInp) - 4) & ".out"

'Open Toolkit
Err = ENopen(DirInp, DirRpt, DirOut)
'Obtain the index of the node of interest
Err = ENgetnodeindex(IDNode, indexNode)

'Open the hydraulic calculation module and iterate all the deman
Err = ENopenH : If Err <> 0 Then Stop
For k = 1 To 3
    nTime = 0
    Err = ENsetnodevalue(indexNode, EN_BASEDEMAND, DemandBase(k)
    Err = ENinitH(1) : If Err <> 0 Then Stop
    Do
        Err = ENrunH(tt) : If Err > 6 Then Stop
        If nTime = tt / 3600 Then
            'Retrieve pressure  value
            Err = ENgetnodevalue(indexNode, EN_PRESSURE, Value)
            ReDim Preserve Demand(k).Pressure(nTime + 1)
            Demand(k).Pressure(nTime) = Value
            nTime = nTime + 1
        End If
        Err = ENnextH(ttt)
    Loop Until ttt = 0
Next

'Close hydraulic calculation module, inp file and free up memory
Err = ENcloseH
Err = ENclose()

MsgBox("Process finished", , "Report")
```

Figure 5.14. Functions used to calculate pressure at node 23

E. Changing pipe orientation

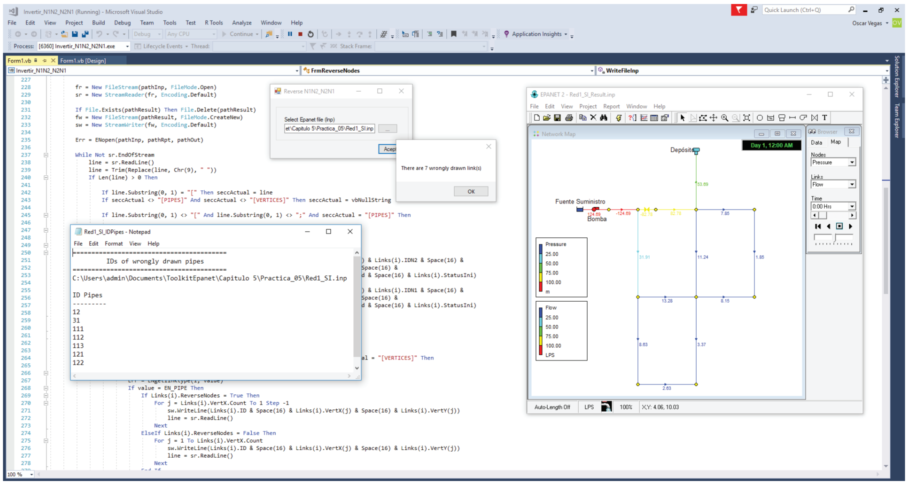
Many times network models appear with a negative pipe flow upon running a hydraulic simulation. This is not an error and is due to having imported the network using AutoCAD, GIS, a Database, or any other storage means where the link orientation does not match the flow direction of the current time step.

If the flow sign is predominantly negative, we can resolve this issue by using the Epanet "Reverse" option. For that, we have to put the cursor over the link, right click and select "Reverse". This does not require much effort if there are few links that need reversal. What if, however, there are hundreds or thousands of oppositely oriented links? With the aid of certain functions contained in the Epanet library, we can resolve this problem.

This issue is resolved by running a step-by-step hydraulic simulation and getting the pipe flows throughout the simulation period. If the flow is negative in all the simulation period time steps, that means that the pipes have not been digitized in the final flow direction.

Since no such function exists in the Epanet library v2.00.12, we will use Visual Studio 2017's native functions (Visual Basic .NET) to recover the information from the [PIPES] section, and the coordinates from the [VERTICES] section. We will work with the Net1_SI.inp network model, whose link orientation has been modified to verify that the algorithm works correctly.

The tool will ask that we select an Epanet inp file and that will give us another (corrected) inp file in the same directory as the original file, along with a text file with pipe IDs whose orientation has been modified. To download the complete code, go to the following link: https://www.imta.gob.mx/biblioteca/download/?key=240. In Figure 5.15 we can see the operation of this tool.



```
            fr = New FileStream(pathInp, FileMode.Open)
            sr = New StreamReader(fr, Encoding.Default)

            If File.Exists(pathResult) Then File.Delete(pathResult)
            fw = New FileStream(pathResult, FileMode.CreateNew)
            sw = New StreamWriter(fw, Encoding.Default)

            Err = ENopen(pathInp, pathRpt, pathOut)

            While Not sr.EndOfStream
                line = sr.ReadLine()
                line = Trim(Replace(line, Chr(9), " "))
                If Len(line) > 0 Then

                    If line.Substring(0, 1) = "[" Then seccActual = line
                    If seccActual <> "[PIPES]" And seccActual <> "[VERTICES]" Then seccActual = vbNullString

                    If line.Substring(0, 1) <> "[" And line.Substring(0, 1) <> ";" And seccActual = "[PIPES]" Then
```

```
) & Links(i).IDN2 & Space(16) &
Space(16) &
d & Space(16) & Links(i).StatusIni)

) & Links(i).IDN1 & Space(16) &
Space(16) &
d & Space(16) & Links(i).StatusIni)

al = "[VERTICES]" Then
```

```
                            If value = EN_PIPE Then
                                If Links(i).ReverseNodes = True Then
                                    For j = Links(i).VertX.Count To 1 Step -1
                                        sw.WriteLine(Links(i).ID & Space(16) & Links(i).VertX(j) & Space(16) & Links(i).VertY(j))
                                        line = sr.ReadLine()
                                    Next
                                ElseIf Links(i).ReverseNodes = False Then
                                    For j = 1 To Links(i).VertX.Count
                                        sw.WriteLine(Links(i).ID & Space(16) & Links(i).VertX(j) & Space(16) & Links(i).VertY(j))
                                        line = sr.ReadLine()
                                    Next
```

Select Epanet file (Inp): et\Capitulo 5\Practica_05\Red1_SI.inp

There are 7 wrongly drawn link(s)

Red1_SI_IDPipes - Notepad

```
========================================
        IDs of wrongly drawn pipes
========================================
C:\Users\admin\Documents\ToolkitEpanet\Capitulo 5\Practica_05\Red1_SI.inp

ID Pipes
---------
12
31
111
112
113
121
122
```

Figure 5.15. Tool to reverse the oppositely-oriented links with help of the Epanet library

ABOUT THE AUTHORS

Oscar Tomas Vegas Niño

Agricultural engineer from the Trujillo National University (Perú), he has a Master degree in Hydraulic and Environmental Engineering from the Universitat Politècnica de València (Spain) and two majors in the urban hydraulic and water resources field. He participates actively in national and international congresses, is scientific papers reviewer and collaborates with projects financed by FYNCyT (Perú), all the while developing programming tools using the Epanet library and Geographic Information Systems (GIS). At the date of this book publishing, he finds himself developing his doctoral thesis in the area of urban hydraulics financed by the Peruvian government (PRONABEC).

Fernando Martínez Alzamora

Industrial Engineering doctor and Hydraulic engineering professor at the Universitat Politècnica de València (UPV, Spain). Author of the Spanish version of Epanet 2.0, over the last 35 years, he has published numerous papers in journals and congresses covering the use and improvement of this tool. He also is responsible for the GISRed and SCARed applications aimed at connecting Epanet to GIS environments to aid the construction of models in SCADA systems for achieving real-time control of water networks. He leads the REDHISP research group from the Environment and Water Engineering Institute (IIAMA) of the UPV.

ABOUT THE AUTHORS

Joan Carles Alonso Campos

Industrial Engineer from the Universitat Politècnica de València (Spain). At the date of this publication, he is studying his doctorate in the Water and Environmental Engineering at the same university. His thesis project, financed through the Val I+D program from the Generalitat Valenciana, has focused on the development and implementation of algorithms for the energy optimization of pressurized hydraulic networks.

Velitchko G. Tzatchkov

Civil Engineer from the Sofía Superior Civil Engineering Institute (Bulgaria) and doctor in Hydraulics from the Hydrotechnique, Irrigation and Drainage Institute from the same city. Since 1991 he has been researcher in Hydraulics at the Mexican Institute of Water Technology. He has developed algorithms and computer programming tools for the analysis, design and operation of water distribution systems as well as pressure irrigation, including an algorithm that broadens the Epanet modelling of water quality with a dispersion term, which was published and accoladed in the U.S.A. He has published numerous papers in journals and congresses, and several books.

The book Using the Epanet toolkit v2.00.12 with different programming environments was published in April 2018 in digital format.
Mexican Institute of Water Technology.

Avances del Conocimiento

IMTA
Instituto Mexicano de Tecnología del Agua