EPANET Programmer's Toolkit

EPANET is a program that analyzes the hydraulic and water quality behavior of water distribution systems. The EPANET Programmer's Toolkit is a dynamic link library (DLL) of functions that allows developers to customize EPANET's computational engine for their own specific needs. The functions can be incorporated into 32-bit Windows applications written in C/C++, Delphi Pascal, Visual Basic, or any other language that can call functions within a Windows DLL. The Toolkit DLL file is named EPANET2.DLL and is distributed with EPANET. The Toolkit comes with several different header files, function definition files, and .lib files that simplify the task of interfacing it with C/C++, Delphi, and Visual Basic code.

EPANET and its Programmer's Toolkit were developed by the Water Supply and Water Resources Division of the U.S. Environmental Protection Agency's National Risk Management Research Laboratory.

1. Toolkit Overview

The Programmer's Toolkit is an extension of the EPANET simulation package. EPANET performs extended period simulation of hydraulic and water quality behavior within pressurized pipe networks. A network can consist of pipes, nodes (pipe junctions), pumps, valves and storage tanks or reservoirs. EPANET tracks the flow of water in each pipe, the pressure at each node, the height of water in each tank, and the concentration of a chemical species throughout the network during a multi-time period simulation. In addition to chemical species, water age and source tracing can also be simulated.

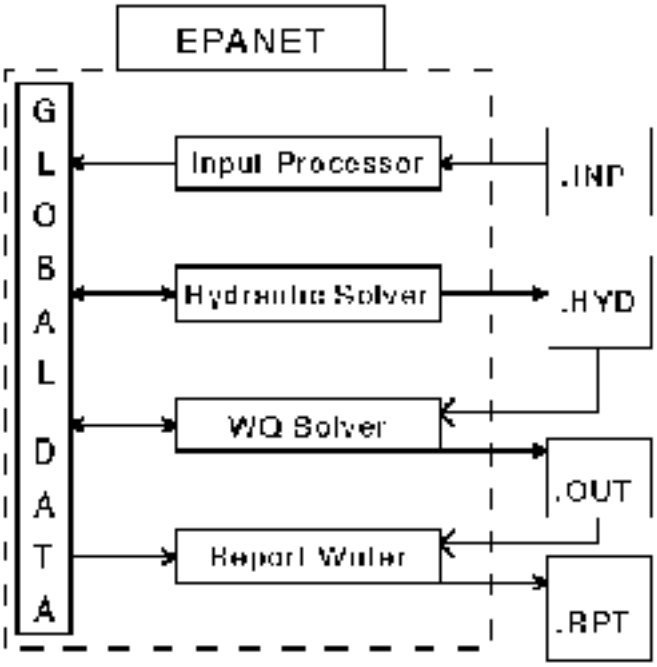
The Toolkit provides a series of functions that allow programmers to customize the use of EPANET's hydraulic and water quality solution engine to their own applications. Before using the Toolkit one should become familiar with the way that EPANET represents a pipe network and the design and operating information it requires to perform a simulation. This information can be obtained from reading EPANET's on-line Help file or from the EPANET Users Manual.

A typical usage of the Toolkit functions to analyze a distribution system might look as follows:

1. Use the `ENopen` function to open the Toolkit system, along with an EPANET Input file.
2. Use the `ENsetxxx` series of functions to change selected system characteristics.
3. Run a full hydraulic simulation using the `ENsolveH` function (which automatically saves results to a Hydraulics file) or use the `ENopenH` - `ENinitH` - `ENrunH` - `ENnextH` - `ENcloseH` series of functions to step through a hydraulic simulation, accessing results along the way with the `ENgetxxx` series of functions.
4. Run a full water quality simulation using `ENsolveQ` (which automatically saves hydraulic and water quality results to an Output file) or use the `ENopenQ` - `ENinitQ` - `ENrunQ` - `ENnextQ` (or `ENstepQ`) - `ENcloseQ` series of functions to step through a water quality simulation, accessing results along the way with the `ENgetxxx` series of functions.
5. Return to Step 2 to run additional analyses or use the `ENreport` function to write a formatted report to the Report file.
6. Call the `ENclose` function to close all files and release system memory.

More specific examples of using the functions can be found in the Example Applications topic.

2. Data Flow Diagram



The EPANET Toolkit is written in ANSI standard C with separate code modules for input processing, hydraulic analysis, water quality analysis, sparse matrix/linear equation analysis, and report generation. The data flow diagram for analyzing a pipe network is shown below. The processing steps depicted in this diagram can be summarized as follows:

- The input processor module receives a description of the network being simulated from an external input file (.INP). The file's contents are parsed, interpreted, and stored in a shared memory area.
- The hydraulics solver module carries out an extended period hydraulic simulation. The results obtained at every time step can be written to an external, unformatted (binary) hydraulics file (.HYD). Some of these time steps might represent intermediate points in time where system conditions change because of tanks becoming full or empty or pumps turning on or off due to level controls or timed operation.
- If a water quality simulation is requested, the water quality module accesses the flow data from the hydraulics file as it computes substance transport and reaction throughout the network over each hydraulic time step. During this process it can write both the formerly computed hydraulic results as well as its water quality results for each preset reporting interval to an unformatted (binary) output file (.OUT). If no water quality analysis was called for, then the hydraulic results stored in the .HYD file can simply be written out to the binary output file at uniform reporting intervals.
- If requested, a report writer module reads back the computed simulation results from the binary output file (.OUT) for each reporting period and writes out selected values to a formatted report file (.RPT). Any error or warning messages generated during the run are also written to this file.

Toolkit functions exist to carry out all of these steps under the programmer's control, including the ability to read or modify most of the system's global data.

3. How to Use the Toolkit

The following topics briefly describe how to accomplish some basic tasks for which the Toolkit would be used. See the Example Applications topic for code listings of complete applications of the Toolkit.

3.1. Opening and Closing the Toolkit

The Toolkit must open an EPANET Input File to obtain a description of the pipe network to be analyzed before any of its other functions can be called. (The exception to this is the `ENepanet` function, which performs a complete hydraulic/water quality simulation similar to a command line execution of EPANET). Once all analysis is completed, it must close itself down to free all allocated memory. The functions for doing this are `ENopen` and `ENclose`, respectively. An example of using these functions is shown below.

```
char *f1, *f2, *f3;
int errcode;
errcode = ENopen(f1, f2, f3);
if (errcode > 0)
{
   ENclose();
   return;
}
/*
   Call functions that perform desired analysis
*/
ENclose();
```

3.2. Retrieving and Setting Network Parameters

The Toolkit has various functions available for retrieving and setting the parameters that define the design and operation of the pipe network being analyzed. The names of retrieval functions all begin with *`ENget`* (e.g., `ENgetnodevalue`, `ENgetoption`, etc.) while the functions used for setting parameter values begin with *`ENset`* (e.g., `ENsetnodevalue`, `ENsetoption`, etc.).

Most of these functions use an index number to reference a specific network component (such as a node, link, or time pattern). This number is simply the position of the component in the list of all components of similar type (e.g., node 10 is the tenth node, starting from 1, in the network) and is not the same as the ID label assigned to the component in the Input File being processed. A series of functions exist to determine a component's index number given its ID label (see `ENgetlinkindex`, `ENgetnodeindex`, and `ENgetpatternindex`). Likewise, functions exist to retrieve a component's ID label given its index number (see `ENgetlinkid`, `ENgetnodeid`, and `ENgetpatternid`). The `ENgetcount` function can be used to determine the number of different components in the network.

The code below is an example of using the parameter retrieval and setting functions. It changes all pipes with diameter of 10 inches to 12 inches.

```
int  i, Nlinks;
float D;
ENgetcount(EN_LINKCOUNT, &Nlinks);
for (i = 1; i <= Nlinks; i++)
{
  ENgetlinkvalue(i, EN_DIAMETER, &D);
  if (D == 10)
    ENsetlinkvalue(i, EN_DIAMETER, 12);
}
```

3.3. Running a Hydraulic Analysis

There are two ways to use the Toolkit to run a hydraulic analysis:

1. Use the `ENsolveH` function to run a complete extended period analysis, without having access to intermediate results
2. Use the `ENopenH` - `ENinitH` - `ENrunH`- `ENnextH` - `ENcloseH` series of functions to step through the simulation one hydraulic time step at a time.

Method 1 is useful if you only want to run a single hydraulic analysis, perhaps to provide input to a water quality analysis. With this method hydraulic results are always saved to the hydraulics file at every time step.

Method 2 must be used if you need to access results between time steps or if you wish to run many analyses efficiently. To accomplish the latter, you would make only one call to `ENopenH` to begin the process, then make successive calls to `ENinitH` - `ENrunH` - `ENnextH` to perform each analysis, and finally call `ENcloseH` to close down the hydraulics system. An example of this is shown below (calls to `ENnextH` are not needed because we are only making a single period analysis in this example).

```
int i, Nruns;
long t;
ENopenH()
for (i = 1; i <= Nruns; i++)
{
/* Set parameters for current run */
  setparams(i);
/* Initialize hydraulics */
  ENinitH(0);
/* Make a single period run */
  ENrunH(&t);
/* Retrieve results */
  getresults(i);
}
ENcloseH();
```

3.4. Running a Water Quality Analysis

Before you can run a water quality analysis, hydraulic results must have been generated either from running a hydraulic analysis or from importing a saved hydraulics file from a previous run. As with a hydraulic analysis, there are two ways to carry out a water quality analysis:

1. Use the `ENsolveQ` function to run a complete extended period analysis, without having access to intermediate results
2. Use the `ENopenQ` - `ENinitQ` - `ENrunQ`- `ENnextQ` - `ENcloseQ` series of functions to step through the simulation one hydraulic time step at a time. (Replacing `ENnextQ` with `ENstepQ` will step through one water quality time step at a time.)

An example of using method 2 is shown below.

```
int err;
long t, tstep;
err = ENsolveH();
if (err > 100) return(err);
ENopenQ();
ENinitQ(1);
do  {
  ENrunQ(&t);
  ENnextQ(&tstep);
} while (tstep > 0);
ENcloseQ();
ENreport();
```

3.5. Retrieving Computed Results

The ENgetnodevalue and ENgetlinkvalue functions are used to retrieve the results of hydraulic and water quality simulations. The computed parameters (and their Toolkit codes) that can be retrieved are as follows:

For Nodes:	**For Links:**
EN_DEMAND (demand)	EN_FLOW (flow rate)
EN_HEAD (hydraulic head)	EN_VELOCITY (flow velocity)
EN_PRESSURE (pressure)	EN_HEADLOSS (headloss)
EN_QUALITY (water quality)	EN_STATUS (link status)
EN_SOURCEMASS (water quality source mass inflow)	EN_SETTING (pump speed or valve setting)

The following code shows how to retrieve the pressure at each node of the network after each time step of a hydraulic analysis (writetofile is a user-defined function that will write a record to a file):

```
int  i, NumNodes;
long  t, tstep;
float p;
char  id[16];
ENgetcount(EN_NODECOUNT, &NumNodes);
ENopenH();
ENinitH(0);
do  {
  ENrunH(&t);
  for (i = 1; i <= NumNodes; i++)
  {
     ENgetnodevalue(i, EN_PRESSURE, &p);
     ENgetnodeid(i, id);
     writetofile(t, id, p);
  }
  ENnextH(&tstep);
} while (tstep > 0);
ENcloseH();
```

3.6. Writing a Report

The Toolkit has some built-in capabilities to produce formatted output results saved to a file. More specialized reporting needs can always be handled by writing specialized code.

The `ENsetreport` function is used to define the format of a report while the `ENreport` function actually writes the report. The latter should be called only after a hydraulic or water quality analysis has been made. An example of creating a report that lists all nodes where the pressure variation over the duration of the simulation exceeds 20 psi is shown below:

```
/* Compute ranges (max - min) */
  ENsettimeparam(EN_STATISTIC, EN_RANGE);
/* Solve hydraulics */
  ENsolveH();
/* Transfer results from hydraulics file to output file */
  ENsaveH();
/* Define contents of the report */
  ENresetreport();
  ENsetreport("FILE myfile.rpt");
  ENsetreport("NODES ALL");
  ENsetreport("PRESSURE PRECISION 1");
  ENsetreport("PRESSURE ABOVE 20");
/* Write the report to file */
  ENreport();
```

4. Example Applications

Example 1 - Providing an embedded engine for other applications

This example shows how simple it is for the Toolkit to provide a network analysis engine for other applications. There are three steps that the application would need to take:

1. Write distribution system data to an EPANET-formatted Input file (see Input File Format).

2. Call the ENepanet function, supplying the name of the EPANET input file, the name of a Report file where status and error messages are written, and the name of a binary Output file which will contain analysis results.

3. Access the output file to display desired analysis results (see Output File Format) in the application.

Example 2 - Developing a hydrant rating curve for fire flow studies

This example illustrates how the Toolkit could be used to develop a hydrant rating curve used in fire flow studies. This curve shows the amount of flow available at a node in the system as a function of pressure. The curve is generated by running a number of steady state hydraulic analyses with the node of interest subjected to a different demand in each analysis. For this example we assume that the ID label of the node of interest is MyNode and that N different demand levels stored in the array D need to be examined. The corresponding pressures will be stored in P. To keep the code more readable, no error checking is made on the results returned from the Toolkit function calls.

Example 2 – C

```
#include "epanet2.h"

void HydrantRating(char *MyNode, int N, float D[], float P[])
{
  int  i, nodeindex;
  long  t;
  float pressure;

  /* Open the EPANET toolkit & hydraulics solver */
  ENopen("example2.inp", "example2.rpt", "");
  ENopenH();

  /* Get the index of the node of interest */
  ENgetnodeindex(MyNode, &nodeindex);

  /* Iterate over all demands */
  for (i=1; i<N; i++)
  {
  /* Set nodal demand, initialize hydraulics, make a
     single period run, and retrieve pressure */
     ENsetnodevalue(nodeindex, EN_BASEDEMAND, D[i]);
     ENinitH(0);
     ENrunH(&t);
     ENgetnodevalue(nodeindex, EN_PRESSURE, &pressure);
     P[i] = pressure;
  }
  /* Close hydraulics solver & toolkit */
  ENcloseH();
  ENclose();
}
```

Example 2 – Pascal

```
uses epanet2;  { Import unit supplied with Toolkit }

procedure HydrantRating(MyNode: PChar; N: Integer;
   D: array of Single; var P: array of Single);

var
  i, nodeindex: Integer;
  t: LongInt;
  pressure: Single;

begin
{ Open the EPANET toolkit & hydraulics solver }
  ENopen('example2.inp', 'example2.rpt', '');
  ENopenH();
{ Get the index of the node of interest }
  ENgetnodeindex(MyNode, nodeindex);
{ Iterate over all demands }
```

```
  for i := 1 to N do
  begin
  { Set nodal demand, initialize hydraulics, make a }
  { single period run, and retrieve pressure }
     ENsetnodevalue(nodeindex, EN_BASEDEMAND, D[i]);
     ENinitH(0);
     ENrunH(t);
     ENgetnodevalue(nodeindex, EN_PRESSURE, pressure);
     P[i] := pressure;
  end;
{ Close hydraulics solver & toolkit }
  ENcloseH();
  ENclose();
end;
```

Example 2 - Visual Basic

```
'Add EPANET2.BAS as a code module to your project

Sub HydrantRating(ByVal MyNode as String, N as Long,_
  D() as Single, P() as Single)

Dim i as Long
Dim nodeindex as Long
Dim t as Long
Dim pressure as Single

'Open the EPANET toolkit and hydraulics solver
ENopen "example2.inp", "example2.rpt", ""
ENopenH

'Get the index of the node of interest
ENgetnodeindex MyNode, nodeindex

'Iterate over all demands
For i = 1 to N
  'Set nodal demand, initialize hydraulics, make a
  'single period run, and retrieve pressure
  ENsetnodevalue nodeindex, EN_BASEDEMAND, D(i)
  ENinitH 0
  ENrunH t
  ENgetnodevalue nodeindex, EN_PRESSURE, pressure
  P(i) = pressure
Next i
'Close hydraulics solver & toolkit
ENcloseH
ENclose
End Sub
```

Example 3 - Meeting a minimum chlorine residual target

This example illustrates how the Toolkit could be used to determine the lowest dose of chlorine applied at the entrance to a distribution system needed to ensure that a minimum residual is met throughout the system. We assume that the EPANET input file contains the proper set of kinetic coefficients that describe the rate at which chlorine will decay in the system being studied. In the example code, the ID label of the source node is contained in SourceID, the minimum residual target is given by Ctarget, and the target is only checked after a start-up duration of 5 days (432,000 seconds). To keep the code more readable, no error checking is made on the results returned from the Toolkit function calls.

Example 3 – C

```
#include "epanet.h"

float cl2dose(char *SourceID, float Ctarget)
{
  int i, nnodes, sourceindex, violation;
  float c, csource;
  long t, tstep;

  /* Open the toolkit & obtain a hydraulic solution */
  ENopen("example3.inp", "example3.rpt", "");
  ENsolveH();

  /* Get the number of nodes & the source node's index   */
  ENgetcount(EN_NODES, &nnodes);
  ENgetnodeindex(SourceID, &sourceindex);

  /* Setup system to analyze for chlorine
     (in case it was not done in the input file.) */
  ENsetqualtype(EN_CHEM, "Chlorine", "mg/L", "");

  /* Open the water quality solver */
  ENopenQ();

  /* Begin the search for the source concentration */
  csource = 0.0;
  do  {
  /* Update source concentration to next level */
    csource = csource + 0.1;
    ENsetnodevalue(sourceindex, EN_SOURCEQUAL, csource);
  /* Run WQ simulation checking for target violations */
    violation = 0;
    ENinitQ(0);
    do    {
      ENrunQ(&t);
      if (t > 432000)
      {
         for (i=1; i<=nnodes; i++)
         {
            ENgetnodevalue(i, EN_QUALITY, &c);
            if (c < Ctarget)
            {
```

```
                violation = 1;
                break;
             }
          }
       }
       ENnextQ(&tstep);
     /* End WQ run if violation found */
     } while (!violation && tstep > 0);
  /* Continue search if violation found */
  } while (violation && csource <= 4.0);

  /* Close up the WQ solver and toolkit */
  ENcloseQ();
  ENclose();
  return csource;
}
```

Example 3 – Pascal

```
uses epanet2; { Import unit supplied with the Toolkit }

function cl2dose(SourceID: PChar; Ctarget: Single): Single;
var
  i, nlinks, nnodes, sourceindex, violation: Integer;
  c, csource: Single;
  t, tstep: LongInt;
begin
  { Open the toolkit & obtain a hydraulic solution }
  ENopen('example3.inp', 'example3.rpt', '');
  ENsolveH();

  { Get the number of nodes & }
  { the source node's index   }
  ENgetcount(EN_NODES, nnodes);
  ENgetnodeindex(SourceID, sourceindex);

  { Setup system to analyze for chlorine }
  { (in case it was not done in the input file.) }
  ENsetqualtype(EN_CHEM, 'Chlorine', 'mg/L', '');

  { Open the water quality solver }
  ENopenQ();

  { Begin the search for the source concentration }
  csource := 0;
  repeat

  { Update source concentration to next level }
    csource := csource + 0.1;
    ENsetnodevalue(sourceindex, EN_SOURCEQUAL, csource);

  { Run WQ simulation checking for target violations }
```

```
    violation := 0;
    ENinitQ(0);
    repeat
      ENrunQ(t);
      if (t > 432000) then
      begin
         for i := 1 to nnodes do
         begin
            ENgetnodevalue(i, EN_QUALITY, c);
            if (c < Ctarget) then
            begin
               violation := 1;
               break;
            end;
         end;
      end;
      ENnextQ(tstep);
 { End WQ run if violation found }
    until (violation = 1) or (tstep = 0);

  { Continue search if violation found }
  until (violation = 0) or (csource >= 4.0);

  { Close up the WQ solver and toolkit }
  ENcloseQ();
  ENclose();
  result := csource;
end;
```

Example 3 - Visual Basic

```
'Add EPANET.BAS as a code module to your project

Function cl2dose(ByVal SourceID as String,_
 ByVal Ctarget as Single)as Single

Dim i as Long
Dim nlinks as Long
Dim nnodes as Long
Dim sourceindex as Long
Dim violation as Integer
Dim c as Single
Dim csource as Single
Dim t as Long
Dim tstep as Long

'Open the toolkit & obtain a hydraulic solution
ENopen "example3.inp", "example3.rpt", ""
ENsolveH

'Get the number of nodes & the source node's index
```

```
ENgetcount EN_NODES, nnodes
ENgetnodeindex SourceID, sourceindex

'Setup system to analyze for chlorine
'(in case it was not done in the input file.)
ENsetqualtype EN_CHEM, "Chlorine", "mg/L", ""

'Open the water quality solver
ENopenQ

'Begin the search for the source concentration
csource = 0
Do

  'Update source concentration to next level
  csource = csource + 0.1
  ENsetnodevalue sourceindex, EN_SOURCEQUAL, csource

  'Run WQ simulation checking for target violations
  violation = 0
  ENinitQ 0
  Do
    ENrunQ t
    If t > 432000 Then
      For i = 1 to nnodes
         ENgetnodevalue i, EN_QUALITY, c
         If c < Ctarget Then
            violation = 1
            Exit For
          End If
       Next i
    End If
    ENnextQ tstep

  'End WQ run if violation found
Loop Until (violation = 1) Or (tstep = 0)

'Continue search if violation found
Loop Until (violation = 0) Or (csource >= 4.0)

'Close up the WQ solver and toolkit
ENcloseQ
ENclose
cl2dose = csource
End Function
```

5. Efficiency Issues

When making multiple hydraulic analyses on the same network (as would be done in an optimization procedure), do not use repeated calls to `ENsolveH`. Instead, use an iterated `ENrunH` - `ENnextH` loop as shown below:

```
int stop;
  long t, tstep;
  ENopenH();
  stop = 0;
  do  {
    setparams();
    ENinitH(0);
    do
    {
      ENrunH(&t);
      evalresults(t, &stop);
      ENnextH(&tstep);
    } while (tstep > 0 && !stop);
  } while (!stop);
  ENcloseH();
```

In the code above, `setparams()` would be a user-defined function which modifies the network in some manner from one iteration to the next. Another user-defined function, `evalresults()`, would evaluate the results at time `t` and set the value of a stop flag that signals the end of the iterations. Note that the argument passed to `ENinitH()` is 0, indicating that there is no need to save hydraulic results to file since they are being used directly as they are generated. This will also speed up the computations.

When there is a need to make repeated water quality runs using the same hydraulics, then call `ENsolveH` once to generate and save the hydraulic solution and use code similar to that above for the water quality runs (using the `ENopenQ`, `ENinitQ`, `ENrunQ`, `ENnextQ`, and `ENcloseQ` functions instead).

6. Toolkit Reference

6.1. Error Codes

Code	Description
0	No error
101	Insufficient memory
102	No network data to process
103	Hydraulics solver not initialized
104	No hydraulic results available
105	Water quality solver not initialized
106	No results to report on

110 Cannot solve hydraulic equations
120 Cannot solve WQ transport equations

200 One or more errors in input file
202 Illegal numeric value in function call
203 Undefined node in function call
204 Undefined link in function call
205 Undefined time pattern in function call
207 Attempt made to control a check valve
223 Not enough nodes in network
224 No tanks or reservoirs in network
240 Undefined source in function call
241 Undefined control statement in function call
250 Function argument has invalid format
251 Illegal parameter code in function call

301 Identical file names
302 Cannot open input file
303 Cannot open report file
304 Cannot open binary output file
305 Cannot open hydraulics file
306 Invalid hydraulics file
307 Cannot read hydraulics file
308 Cannot save results to file
309 Cannot write report to file

6.2. Warning Codes

Code	Description
1	System hydraulically unbalanced - convergence to a hydraulic solution was not achieved in the allowed number of trials
2	System may be hydraulically unstable - hydraulic convergence was only achieved after the status of all links was held fixed
3	System disconnected - one or more nodes with positive demands were disconnected from all supply sources
4	Pumps cannot deliver enough flow or head - one or more pumps were forced to either shut down (due to insufficient head) or operate beyond the maximum rated flow
5	Valves cannot deliver enough flow - one or more flow control valves could not deliver the required flow even when fully open
6	System has negative pressures - negative pressures occurred at one or more junctions with positive demand

6.3. File Descriptions

Support Files

The EPANET Programmer's Toolkit comes with several files that support its use with different programming languages.

File Name	Purpose
`epanet2.h`	Header file for C/C++
`epanet2bc.lib`	Library file for Borland C/C++
`epanet2vc.lib`	Library file for Microsoft Visual C++
`epanet2.pas`	Import unit for Delphi (Pascal)
`epanet2.bas`	Declarations module for Visual Basic

Input File

The **Input file** is a standard EPANET input data file that describes the system being analyzed (see Input File Format). It can either be created external to the application being developed with the Toolkit or by the application itself. It is the first file name supplied to the ENopen function. None of the other Toolkit functions (except `ENepanet`) can be used until an Input file has been opened with `ENopen`. The data associated with the Input file remains accessible until the Toolkit system is closed down with the `ENclose` function.

Hydraulics File

The **Hydraulics file** is an unformatted binary file used to store the results of a hydraulic analysis. Results for all time periods are stored, including those at intermediate times when special hydraulic events occur (e.g., pumps and tanks opening or closing because control conditions have been satisfied).

Normally it is a temporary file that is deleted after the `ENclose` function is called. However, it will be saved if the `ENsavehydfile` function is called.

Likewise, a previously saved Hydraulics file can be used if the command `HYDRAULICS USE` *`filename`* appears in the `[OPTIONS]` section of the input file, or if the `ENusehydfile` function is called.

When the Toolkit function `ENsolveH` is used to make a hydraulic analysis, results are automatically saved to the Hydraulics file. When the `ENinitH` - `ENrunH` - `ENnextH` set of functions is used, the *`saveflag`* argument to `ENinitH` determines whether results are saved or not. The need to save hydraulic results is application-dependent. They must always be saved to the Hydraulics file if a water quality analysis will follow.

Report File

The **Report file** is the second file name supplied to the `ENopen` (or `ENepanet`) function. It is used to log any error messages that occur when the Input file is being processed and to

record all status messages that are generated during a hydraulic simulation. In addition, if the ENreport function is called the resulting report can also be written to this file. The format of the report is controlled by statements placed in the [REPORT] section of the Input file and by similar statements included in calls to the ENsetreport function. Only results at a specified uniform reporting time interval are written to this file.

To suppress the writing of all error and warning messages to the Report file either include the command MESSAGES NO in the [REPORT] section of the Input file or call the Toolkit function ENsetreport("MESSAGES NO").

To route a formatted report to a different file than the Report file either include the command FILE *filename* in the [REPORT] section of the Input file or call the Toolkit function ENsetreport("FILE *filename*"), where *filename* is the name of the file to use.

Output File

The **Output file** is an unformatted binary file used to store both hydraulic and water quality results at uniform reporting intervals (see Output File Format). It is the third file name supplied to the ENopen function. If an empty string ("") is used as its name then a scratch temporary file will be used. Otherwise the Output file will be saved after the ENclose function is called. Saving this file is useful if further post-processing of the output results are needed. The function ENsaveH will transfer hydraulic results to the Output file if no water quality analysis will be made. Using ENsolveQ to run a water quality analysis automatically saves both hydraulic and water quality results to this file. If the ENinitQ - ENrunQ - ENnextQ set of functions is used to perform a water quality analysis, then results will be saved only if the *saveflag* argument of ENinitQ is set to 1. Again, the need to save results to the Output file is application-dependent. If a formatted output report is to be generated using ENreport, then results must first be saved to the Output file.

6.3.1. Input File Format

The EPANET Toolkit works with an input text file that describes the pipe network being analyzed. The file is organized by sections where each section begins with a keyword enclosed in brackets. The various keywords are listed below. Click on a section to see the format of the data it contains.

Network Components	System Operation	Water Quality	Options & Reporting
[TITLE]	[CURVES]	[QUALITY]	[OPTIONS]
[JUNCTIONS]	[PATTERNS]	[REACTIONS]	[TIMES]
[RESERVOIRS]	[ENERGY]	[SOURCES]	[REPORT]
[TANKS]	[STATUS]	[MIXING]	
[PIPES]	[CONTROLS]		
[PUMPS]	[RULES]		
[VALVES]	[DEMANDS]		
[EMITTERS]			

The order of sections is not important. However, whenever a node or link is referred to in a section it must have already been defined in the [JUNCTIONS], [RESERVOIRS], [TANKS], [PIPES], [PUMPS], or [VALVES] sections. Thus it is recommended that these sections be placed first.

Each section can contain one or more lines of data. Blank lines can appear anywhere in the file and the semicolon (;) can be used to indicate that what follows on the line is a comment, not data. A maximum of 255 characters can appear on a line.

The ID labels used to identify nodes, links, curves and patterns can be any combination of up to 15 characters and numbers.

[TITLE]

Purpose:
Attaches a descriptive title to the network being analyzed.

Format:
Any number of lines of text.

Remarks:
The [TITLE] section is optional.

[JUNCTIONS]

Purpose:
Defines junction nodes contained in the network.

Format:
One line for each junction containing:
- ID label
- Elevation, ft (m)
- Base demand flow (flow units) (optional)
- Demand pattern ID (optional)

Remarks:
1. A [JUNCTIONS] section with at least one junction is required.
2. If no demand pattern is supplied then the junction demand follows the Default Demand Pattern provided in the [OPTIONS] section, or Pattern 1 if no Default Pattern is specified. If the Default Pattern (or Pattern 1) does not exist, then the demand remains constant.
3. Demands can also be entered in the [DEMANDS] section and include multiple demand categories per junction.

Example:

```
[JUNCTIONS]
;ID    Elev.   Demand   Pattern
;------------------------------
J1    100      50       Pat1
J2    120      10               ;Uses default demand pattern
J3    115                       ;No demand at this junction
```

[RESERVOIRS]

Purpose: Defines all reservoir nodes contained in the network.

Format: One line for each reservoir containing:
- ID label
- Head, ft (m)
- Head pattern ID (optional)

Remarks:
1. Head is the hydraulic head (elevation + pressure head) of water in the reservoir.
2. A head pattern can be used to make the reservoir head vary with time.
3. At least one reservoir or tank must be contained in the network.

Example:

```
[RESERVOIRS]
;ID    Head    Pattern
;---------------------
 R1    512             ;Head stays constant
 R2    120     Pat1    ;Head varies with time
```

[TANKS]

Purpose: Defines all tank nodes contained in the network.

Format: One line for each tank containing:
- ID label
- Bottom elevation, ft (m)
- Initial water level, ft (m)
- Minimum water level, ft (m)
- Maximum water level, ft (m)
- Nominal diameter, ft (m)
- Minimum volume, cubic ft (cubic meters)
- Volume curve ID (optional)

Remarks:
1. Water surface elevation equals bottom elevation plus water level.
2. Non-cylindrical tanks can be modeled by specifying a curve of volume versus water depth in the `[CURVES]` section.
3. If a volume curve is supplied the diameter value can be any non-zero number
4. Minimum volume (tank volume at minimum water level) can be zero for a cylindrical tank or if a volume curve is supplied.
5. A network must contain at least one tank or reservoir.

Example:

```
[TANKS]
;ID   Elev.  InitLvl  MinLvl  MaxLvl  Diam  MinVol  VolCurve
;-----------------------------------------------------------
;Cylindrical tank
T1    100     15       5       25     120   0
;Non-cylindrical tank with arbitrary diameter
 T2   100     15       5       25     1     0        VC1
```

[PIPES]

Purpose: Defines all pipe links contained in the network.

Format:
One line for each pipe containing:
- ID label
- ID of start node
- ID of end node
- Length, ft (m)
- Diameter, inches (mm)
- Roughness coefficient
- Minor loss coefficient
- Status (**OPEN, CLOSED,** or **CV**)

Remarks:
1. Roughness coefficient is unitless for Hazen-Williams and Chezy-Manning head loss formulas and has units of millifeet (mm) for the Darcy-Weisbach formula. Choice of head loss formula is supplied in the `[OPTIONS]` section.
2. Setting status to **CV** means that the pipe contains a check valve restricting flow to one direction.
3. If minor loss coefficient is 0 and pipe is **OPEN** then these two items can be dropped from the input line.

Example:
```
[PIPES]
;ID   Node1  Node2   Length   Diam.  Roughness  Mloss   Status
;-------------------------------------------------------------
 P1    J1     J2     1200      12      120       0.2    OPEN
 P2    J3     J2      600       6      110       0      CV
 P3    J1     J10    1000      12      120
```

[PUMPS]

Purpose:
Defines all pump links contained in the network.

Format:
One line for each pump containing:
- ID label
- ID of start node
- ID of end node
- Keyword and Value (can be repeated)

Remarks:
1. Keywords consists of:
   - **POWER** - power for constant energy pump, hp (kw)
   - **HEAD** - ID of curve that describes head versus flow for the pump
   - **SPEED** - relative speed setting (normal speed is 1.0, 0 means pump is off)
   - **PATTERN** - ID of time pattern that describes how speed setting varies with time
2. Either **POWER** or **HEAD** must be supplied for each pump. The other keywords are optional.

Example:

```
[PUMPS]
;ID    Node1    Node2   Properties
;-------------------------------------------
Pump1   N12      N32     HEAD Curve1
Pump2   N121     N55     HEAD Curve1  SPEED 1.2
Pump3   N22      N23     POWER 100
```

[VALVES]

Purpose: Defines all control valve links contained in the network.

Format: One line for each valve containing:
- ID label
- ID of start node
- ID of end node
- Diameter, inches (mm)
- Valve type
- Valve setting
- Minor loss coefficient

Remarks:
1. Valve types and settings include:

Valve Type	Setting
PRV (pressure reducing valve)	Pressure, psi (m)
PSV (pressure sustaining valve)	Pressure, psi (m)
PBV (pressure breaker valve)	Pressure, psi (m)
FCV (flow control valve)	Flow (flow units)
TCV (throttle control valve)	Loss Coefficient
GPV (general purpose valve)	ID of head loss curve

2. Shutoff valves and check valves are considered to be part of a pipe, not a separate control valve component (see `[PIPES]`)

[EMITTERS]

Purpose: Defines junctions modeled as emitters (sprinklers or orifices).

Format: One line for each emitter containing:
- Junction ID label
- Flow coefficient, flow units at 1 psi (1 meter) pressure drop

Remarks:
1. Emitters are used to model flow through sprinkler heads or pipe leaks.
2. Flow out of the emitter equals the product of the flow coefficient and the junction pressure raised to a power.

3. The power can be specified using the `EMITTER EXPONENT` option in the `[OPTIONS]` section. The default power is 0.5, which normally applies to sprinklers and nozzles.
4. Actual demand reported in the program's results includes both the normal demand at the junction plus flow through the emitter.
5. An `[EMITTERS]` section is optional.

[CURVES]

Purpose: Defines data curves and their X,Y points.

Format: One line for each X,Y point on each curve containing:
- Curve ID label
- X value
- Y value

Remarks:
1. Curves can be used to represent the following relations:
   - Head v. Flow for pumps
   - Efficiency v. Flow for pumps
   - Volume v. Depth for tanks
   - Head Loss v. Flow for General Purpose Valves
2. The points of a curve must be entered in order of increasing X-values (lower to higher).
3. If the input file will be used with the Windows version of EPANET, then adding a comment which contains the curve type and description, separated by a colon, directly above the first entry for a curve will ensure that these items appear correctly in EPANET's Curve Editor. Curve types include PUMP, EFFICIENCY, VOLUME, and HEADLOSS. See the examples below.

Example:

```
[CURVES]
;ID   Flow    Head
;PUMP: Curve for Pump 1
C1    0       200
C1    1000    100
C1    3000    0

;ID   Flow    Effic.
;EFFICIENCY:
E1    200     50
E1    1000    85
E1    2000    75
E1    3000    65
```

[PATTERNS]

Purpose: Defines time patterns.

Format: One or more lines for each pattern containing:
- Pattern ID label
- One or more multipliers

Remarks:
1. Multipliers define how some base quantity (e.g., demand) is adjusted for each time period.
2. All patterns share the same time period interval as defined in the `[TIMES]` section.
3. Each pattern can have a different number of time periods.
4. When the simulation time exceeds the pattern length the pattern wraps around to its first period.
5. Use as many lines as it takes to include all multipliers for each pattern.

Example:

```
[PATTERNS]
;Pattern P1
P1     1.1    1.4    0.9    0.7
P1     0.6    0.5    0.8    1.0
;Pattern P2
P2     1      1      1      1
P2     0      0      1
```

[ENERGY]

Purpose: Defines parameters used to compute pumping energy and cost.

Formats:

GLOBAL **PRICE/PATTERN/EFFIC** *value* **PUMP** *PumpID*
PRICE/PATTERN/EFFIC *value*
DEMAND CHARGE *value*

Remarks:
1. First format is used to set global default values of energy price, price pattern, and pumping efficiency for all pumps.
2. Second format is used to override global defaults for specific pumps.
3. Parameters are defined as follows:
   PRICE = average cost per kW-hour,
   PATTERN = ID label of time pattern describing how energy price varies with time,
   EFFIC = either a single percent efficiency for global setting or the ID label of an efficiency curve for a specific pump,
   DEMAND CHARGE = added cost per maximum kW usage during the simulation period.
4. The default global pump efficiency is 75% and the default global energy price is 0.
5. All entries in this section are optional. Items offset by slashes (/) indicate allowable choices.

Example:

```
[ENERGY]
GLOBAL PRICE      0.05   ;Sets global energy price
GLOBAL PATTERN    PAT1   ;and time-of-day pattern
PUMP   23  PRICE  0.10   ;Overrides price for Pump 23
PUMP   23  EFFIC  E23    ;Assigns effic. curve to Pump 23
```

[STATUS]

Purpose: Defines initial status of selected links at the start of a simulation.

Format: One line per link being controlled containing:
- Link ID label
- Status or setting

Remarks:
1. Links not listed in this section have a default status of **OPEN** (for pipes and pumps) or **ACTIVE** (for valves).
2. The Status value assigned in this section can be **OPEN** or **CLOSED.** For control valves (e.g., PRVs, FCVs, etc.) this means that the valve is either fully opened or closed, not active at its control setting.
3. The Setting value can be a speed setting for pumps or valve setting for valves.
4. The initial status of pipes can also be set in the [PIPES] section.
5. Check valves cannot have their status be preset.
6. Use [CONTROLS] or [RULES] to change status or setting at some future point in the simulation.
7. If a **CLOSED** or **OPEN** control valve is to become ACTIVE again, then its pressure or flow setting must be specified in the control or rule that reactivates it.

Example:

```
[STATUS]
; Link   Status/Setting
;----------------------
   L22    CLOSED       ;Link L22 is closed
   P14    1.5          ;Speed for pump P14
   PRV1   OPEN         ;PRV1 forced open
                       ;(overrides normal operation)
```

[DEMANDS]

Purpose:
Supplement to [JUNCTIONS] section for defining multiple water demands at junction nodes.

Format: One line for each category of demand at a junction containing:
- Junction ID label
- Base demand (flow units)
- Demand pattern ID (optional)
- Name of demand category preceded by a semicolon (optional)

Remarks:
1. Only use for junctions whose demands need to be changed or supplemented from entries in [JUNCTIONS] section.
2. Data in this section replaces any demand entered in [JUNCTIONS] section for the same junction.
3. Unlimited number of demand categories can be entered per junction.
4. If no demand pattern is supplied then the junction demand follows the Default Demand Pattern provided in the [OPTIONS] section, or Pattern 1 if no Default Pattern is supplied. If the Default Pattern (or Pattern 1) does not exist, then the demand remains constant.

Example:

```
[DEMANDS]
;ID     Demand    Pattern    Category
;-------------------------------
J1      100        101      ;Domestic
J1      25         102      ;School
J256    50         101      ;Domestic
```

[CONTROLS]

Purpose: Defines simple controls that modify links based on a single condition.

Format: One line for each control which can be of the form:

```
LINK linkID status IF NODE nodeID ABOVE/BELOW value
LINK linkID status AT TIME time
LINK linkID status AT CLOCKTIME clocktime AM/PM
```

where:

`linkID`	=	a link ID label
`status`	=	**OPEN** or **CLOSED,** a pump speed setting, or a control valve setting
`nodeID`	=	a node ID label
`value`	=	a pressure for a junction or a water level for a tank
`time`	=	a time since the start of the simulation in hours
`clocktime`	=	a 24-hour clock time (hrs:min)

Remarks:

1. Simple controls are used to change link status or settings based on tank water level, junction pressure, time into the simulation or time of day.
2. See the notes for the `[STATUS]` section for conventions used in specifying link status and setting, particularly for control valves.

Examples:

```
[CONTROLS]
;Close Link 12 if the level in Tank 23 exceeds 20 ft.
LINK 12 CLOSED IF NODE 23 ABOVE 20

;Open Link 12 if the pressure at Node 130 is under 30 psi
LINK 12 OPEN IF NODE 130 BELOW 30

;Pump PUMP02's speed is set to 1.5 at 16 hours into
;the simulation
LINK PUMP02 1.5 AT TIME 16

;Link 12 is closed at 10 am and opened at 8 pm
;throughout the simulation
LINK 12 CLOSED AT CLOCKTIME 10 AM
LINK 12 OPEN AT CLOCKTIME 8 PM
```

[RULES]

Purpose:
Defines rule-based controls which modify links based on a combination of conditions.

Format: Each rule is a series of statements of the form:

```
RULE ruleID
IF  condition_1
AND condition_2
OR  condition_3
AND condition_4
etc.
THEN action_1
AND  action_2
etc.
ELSE action_3
AND  action_4
etc.
PRIORITY value
```

where:

`ruleID`	=	an ID label assigned to the rule
`conditon_n`	=	a condition clause
`action_n`	=	an action clause
`priority`	=	a priority value (e.g., a number from 1 to 5)

Remarks:
1. Only the **RULE**, **IF** and **THEN** portions of a rule are required; the other portions are optional.
2. When mixing **AND** and **OR** clauses, the **OR** operator has higher precedence than **AND**, i.e.,
   ```
   IF A or B and C
   ```
   is equivalent to
   ```
   IF (A or B) and C.
   ```
   If the interpretation was meant to be
   ```
   IF A or (B and C)
   ```
   then this can be expressed using two rules as in
   ```
   IF A THEN ...
   IF B and C THEN ...
   ```
3. The **PRIORITY** value is used to determine which rule applies when two or more rules require that conflicting actions be taken on a link. A rule without a priority value always has a lower priority than one with a value. For two rules with the same priority value, the rule that appears first is given the higher priority.

Example:
```
[RULES]

RULE 1
IF   TANK   1 LEVEL ABOVE 19.1
THEN PUMP 335 STATUS IS CLOSED
AND  PIPE 330 STATUS IS OPEN

RULE 2
```

```
IF   SYSTEM CLOCKTIME >= 8 AM
AND  SYSTEM CLOCKTIME < 6 PM
AND  TANK 1 LEVEL BELOW 12
THEN PUMP 335 STATUS IS OPEN

RULE 3
IF   SYSTEM CLOCKTIME >= 6 PM
OR   SYSTEM CLOCKTIME < 8 AM
AND  TANK 1 LEVEL BELOW 14
THEN PUMP 335 STATUS IS OPEN
```

Rule Condition Clauses

A condition clause in a Rule-Based Control takes the form of:

```
object id attribute relation value
```

where

`object`	=	a category of network object
`id`	=	the object's ID label
`attribute`	=	an attribute or property of the object
`relation`	=	a relational operator
`value`	=	an attribute value

Some example conditional clauses are:

```
JUNCTION 23 PRESSURE > 20
TANK T200 FILLTIME BELOW 3.5
LINK 44 STATUS IS OPEN
SYSTEM DEMAND >= 1500
SYSTEM CLOCKTIME = 7:30 AM
```

Objects can be any of the following keywords:

```
NODE       LINK       SYSTEM
JUNCTION        PIPE
RESERVOIR  PUMP
TANK       VALVE
```

When `SYSTEM` is used in a condition no ID is supplied.

The following attributes can be used with Node-type objects:

```
DEMAND
HEAD
PRESSURE
```

The following attributes can be used with Tanks:

`LEVEL`
`FILLTIME` (hours needed to fill a tank)
`DRAINTIME` (hours needed to empty a tank)

These attributes can be used with Link-Type objects:

`FLOW`
`STATUS (OPEN, CLOSED, or ACTIVE)`
`SETTING` (pump speed or valve setting)

The SYSTEM object can use the following attributes:
DEMAND (total system demand)
TIME (hours from the start of the simulation expressed either as a decimal number or in hours:minutes format))
CLOCKTIME (24-hour clock time with AM or PM appended)

Relation operators consist of the following:

=	IS
<>	NOT
<	BELOW
>	ABOVE
<=	
>=	

Rule Action Clauses

An action clause in a Rule-Based Control takes the form of:

```
object id STATUS/SETTING IS value
```

where

object	=	LINK, PIPE, PUMP, or VALVE keyword
id	=	the object's ID label
value	=	a status condition (OPEN or CLOSED), pump speed setting, or valve setting

Some example action clauses are:

```
LINK 23 STATUS IS CLOSED
PUMP P100 SETTING IS 1.5
VALVE 123 SETTING IS 90
```

See the notes for the [STATUS] section for conventions used in specifying link status and setting, particularly for control valves.

[QUALITY]

Purpose: Defines initial water quality at nodes.

Format: One line per node containing:
- Node ID label
- Initial quality

Remarks:
1. Quality is assumed to be zero for nodes not listed.
2. Quality represents concentration for chemicals, hours for water age, or percent for source tracing.
3. The [QUALITY] section is optional.

[REACTIONS]

Purpose: Defines parameters related to chemical reactions occurring in the network.

Formats:

```
ORDER BULK/WALL/TANK  value
GLOBAL BULK/WALL      value
BULK/WALL/TANK        pipeID  value
LIMITING POTENTIAL    value
ROUGHNESS CORRELATION value
```

Remarks:
1. Remember to use positive numbers for growth reaction coefficients and negative numbers for decay coefficients.
2. The time units for all reaction coefficients are 1/days.
3. All entries in this section are optional. Items offset by slashes (/) indicate allowable choices.

ORDER is used to set the order of reactions occurring in the bulk fluid, at the pipe wall, or in tanks, respectively. Values for wall reactions must be either `0` or `1`. If not supplied the default reaction order is `1.0`.

GLOBAL is used to set a global value for all bulk reaction coefficients (pipes and tanks) or for all pipe wall coefficients. The default value is `0`.

BULK, WALL, and **TANK** are used to override the global reaction coefficients for specific pipes and tanks.

LIMITING POTENTIAL specifies that reaction rates are proportional to the difference between the current concentration and some limiting potential value.

ROUGHNESS CORRELATION will make all default pipe wall reaction coefficients be related to pipe roughness in the following manner:

Head Loss Equation	Roughness Correlation
Hazen-Williams	F / C
Darcy-Weisbach	F / log(e/D)
Chezy-Manning	F*n

where F = roughness correlation, C = Hazen-Williams C-factor, e = Darcy-Weisbach roughness, D = pipe diameter, and n = Chezy-Manning roughness coefficient. The default value computed this way can be overridden for any pipe by using the **WALL** format to supply a specific value for the pipe.

Example:

```
[REACTIONS]
ORDER WALL    0    ;Wall reactions are zero-order
GLOBAL BULK  -0.5  ;Global bulk decay coeff.
GLOBAL WALL  -1.0  ;Global wall decay coeff.
WALL   P220  -0.5  ;Pipe-specific wall coeffs.
WALL   P244  -0.7
```

[SOURCES]

Purpose: Defines locations of water quality sources.

Format: One line for each water quality source containing:
- Node ID label
- Source type (**CONCEN**, **MASS**, **FLOWPACED**, or **SETPOINT**)
- Baseline source strength
- Time pattern ID (optional)

Remarks:
1. For **MASS** type sources, strength is measured in mass flow per minute. All other types measure source strength in concentration units.
2. Source strength can be made to vary over time by specifying a time pattern.
3. A **CONCEN** source:
   - represents the concentration of any external source inflow to the node
   - applies only when the node has a net negative demand (water enters the network at the node)
   - if the node is a junction, reported concentration is the result of mixing the source flow and inflow from the rest of the network
   - if the node is a reservoir, the reported concentration is the source concentration
   - if the node is a tank, the reported concentration is the internal concentration of the tank
   - is best used for nodes that represent source water supplies or treatment works (e.g., reservoirs or nodes assigned a negative demand)
   - do not use at storage tanks with simultaneous inflow/outflow.
4. A **MASS, FLOWPACED**, or **SETPOINT** source:
   - represents a booster source, where the substance is injected directly into the network regardless of what the demand at the node is
   - affects water leaving the node to the rest of the network in the following way:
     - a **MASS** booster adds a fixed mass flow to that resulting from inflow to the node
     - a **FLOWPACED** booster adds a fixed concentration to the resultant inflow concentration at the node
     - a **SETPOINT** booster fixes the concentration of any flow leaving the node (as long as the concentration resulting from the inflows is below the setpoint)
   - the reported concentration at a junction or reservoir booster source is the concentration that results after the boosting is applied; the reported concentration for a tank with a booster source is the internal concentration of the tank
   - is best used to model direct injection of a tracer or disinfectant into the network or to model a contaminant intrusion.
5. A [SOURCES] section is not needed for simulating water age or source tracing.

Example:
```
[SOURCES]
;Node  Type     Strength  Pattern
;--------------------------------
  N1   CONCEN     1.2       Pat1    ;Concentration varies with time
  N44  MASS       12                ;Constant mass injection
```

[MIXING]

Purpose: Identifies the model that governs mixing within storage tanks.

Format: One line per tank containing:
- Tank ID label
- Mixing model (**MIXED, 2COMP, FIFO**, or **LIFO**)
- Compartment volume (fraction)

Remarks:
1. Mixing models include:
   - Completely Mixed (**MIXED**)
   - Two-Compartment Mixing (**2COMP**)
   - Plug Flow (**FIFO**)
   - Stacked Plug Flow (**LIFO**)
2. The compartment volume parameter only applies to the two-compartment model and represents the fraction of the total tank volume devoted to the inlet/outlet compartment.
3. The [MIXING] section is optional. Tanks not described in this section are assumed to be completely mixed.

Example:
```
[MIXING]
;Tank       Model
;----------------------
T12         LIFO
T23         2COMP      0.2
```

[OPTIONS]

Purpose: Defines various simulation options.

Formats:

```
UNITS                CFS/GPM/MGD/IMGD/AFD/
                     LPS/LPM/MLD/CMH/CMD
HEADLOSS             H-W/D-W/C-M
HYDRAULICS           USE/SAVE   filename
QUALITY              NONE/CHEMICAL/AGE/TRACE id
VISCOSITY            value
DIFFUSIVITY          value
SPECIFIC GRAVITY     value
TRIALS               value
ACCURACY             value
UNBALANCED           STOP/CONTINUE/CONTINUE n
PATTERN              id
DEMAND MULTIPLIER    value
EMITTER EXPONENT     value
TOLERANCE            value
MAP                  filename
```

UNITS sets the units in which flow rates are expressed where:

CFS	= cubic feet per second
GPM	= gallons per minute
MGD	= million gallons per day
IMGD	= Imperial MGD
AFD	= acre-feet per day
LPS	= liters per second
LPM	= liters per minute
MLD	= million liters per day
CMH	= cubic meters per hour
CMD	= cubic meters per day

For CFS, GPM, MGD, IMGD, and AFD other input quantities are expressed in US Customary Units. If flow units are in liters or cubic meters then Metric Units must be used for all other input quantities as well. (See Units of Measurement). The default flow units are GPM.

HEADLOSS selects a formula to use for computing head loss for flow through a pipe. The choices are the Hazen-Williams (**H-W**), Darcy-Weisbach (**D-W**), or Chezy-Manning (**C-M**) formulas. The default is **H-W**.

The **HYDRAULICS** option allows you to either **SAVE** the current hydraulics solution to a file or **USE** a previously saved hydraulics solution. This is useful when studying factors that only affect water quality behavior. If the file name supplied contains any spaces then the name must be placed between double quotes.

QUALITY selects the type of water quality analysis to perform. The choices are **NONE, CHEMICAL, AGE**, and **TRACE**. In place of **CHEMICAL** the actual name of the chemical can be used followed by its concentration units (e.g., **CHLORINE mg/L**). If **TRACE** is selected it must be followed by the ID label of the node being traced. The default selection is **NONE** (no water quality analysis).

VISCOSITY is the kinematic viscosity of the fluid being modeled relative to that of water at 20 deg. C (1.0 centistoke). The default value is 1.0.

DIFFUSIVITY is the molecular diffusivity of the chemical being analyzed relative to that of chlorine in water. The default value is 1.0. Diffusivity is only used when mass transfer limitations are considered in pipe wall reactions. A value of 0 will cause EPANET to ignore mass transfer limitations.

SPECIFIC GRAVITY is the ratio of the density of the fluid being modeled to that of water at 4 deg. C (unitless).

TRIALS are the maximum number of trials used to solve network hydraulics at each hydraulic time step of a simulation. The default is 40.

ACCURACY prescribes the convergence criterion that determines when a hydraulic solution has been reached. The trials end when the sum of all flow changes from the previous solution divided by the total flow in all links is less than this number. The default is 0.001.

UNBALANCED determines what happens if a hydraulic solution cannot be reached within the prescribed number of **TRIALS** at some hydraulic time step into the simulation. **"STOP"** will halt the entire analysis at that point. **"CONTINUE"** will continue the analysis with a warning message issued. **"CONTINUE n"** will continue the search for a solution for another "n" trials

with the status of all links held fixed at their current settings. The simulation will be continued at this point with a message issued about whether convergence was achieved or not. The default choice is **"STOP"**.

PATTERN provides the ID label of a default demand pattern to be applied to all junctions where no demand pattern was specified. If no such pattern exists in the [PATTERNS] section then by default the pattern consists of a single multiplier equal to 1.0. If this option is not used, then the global default demand pattern has a label of "1".

The **DEMAND MULTIPLIER** is used to adjust the values of baseline demands for all junctions and all demand categories. For example, a value of 2 doubles all baseline demands, while a value of 0.5 would halve them. The default value is 1.0.
EMITTER EXPONENT specifies the power to which the pressure at a junction is raised when computing the flow issuing from an emitter. The default is 0.5.

TOLERANCE is the difference in water quality level below which we can say that one parcel of water is essentially the same as another. The default is 0.01 for all types of quality analyses (chemical, age (measured in hours), or source tracing (measured in percent)).

MAP is used to supply the name of a file containing coordinates of the network's nodes so that a map of the network can be drawn. It is not used for any hydraulic or water quality computations.

Remarks:
1. All options assume their default values if not explicitly specified in this section.
2. Items offset by slashes (/) indicate allowable choices.

Example:
```
[OPTIONS]
UNITS        CFS
HEADLOSS     D-W
QUALITY      TRACE  Tank23
UNBALANCED   CONTINUE  10
```

[TIMES]

Purpose: Defines various time step parameters used in the simulation.

Formats:

```
DURATION              value      (units)
HYDRAULIC TIMESTEP    value      (units)
QUALITY TIMESTEP      value      (units)
RULE TIMESTEP         value      (units)
PATTERN TIMESTEP      value      (units)
PATTERN START         value      (units)
REPORT TIMESTEP       value      (units)
REPORT START          value      (units)
START CLOCKTIME       value      (AM/PM)
STATISTIC             NONE/AVERAGED/
                      MINIMUM/MAXIMUM/
                      RANGE
```

Remarks:
1. Units can be **SECONDS (SEC)**, **MINUTES (MIN), HOURS,** or **DAYS.** The default is hours.
2. If no units are supplied, then time values can be expressed in either decimal hours or in hours:minutes notation.
3. All entries in the [TIMES] section are optional. Items offset by slashes (/) indicate allowable choices.

DURATION is the duration of the simulation. Use 0 to run a single period snapshot analysis. The default is 0.

HYDRAULIC TIMESTEP determines how often a new hydraulic state of the network is computed. If greater than either the PATTERN or REPORT time step it will be automatically reduced. The default is 1 hour.

QUALITY TIMESTEP is the time step used to track changes in water quality throughout the network. The default is 1/10 of the hydraulic time step.

RULE TIMESTEP is the time step used to evaluate Rule-Based controls. If supplied, it should be some fraction of the Hydraulic Timestep. If not supplied, the default value is 1/10 of the Hydraulic Timestep.

PATTERN TIMESTEP is the interval between time periods in all time patterns. The default is 1 hour.

PATTERN START is the time offset at which all patterns will start. For example, a value of 6 hours would start the simulation with each pattern in the time period that corresponds to hour 6. The default is 0.

REPORT TIMESTEP sets the time interval between which output results are reported. The default is 1 hour.

REPORT START is the length of time into the simulation at which output results begin to be reported. The default is 0.

START CLOCKTIME is the time of day (e.g., 3:00 PM) at which the simulation begins. The default is 12:00 AM midnight.

STATISTIC determines the type statistical post-processing to apply to the time series of analysis results before they are reported. The choices are:

NONE	no post-processing (the default)
AVERAGED	report time-averaged values
MINIMUM	report minimum values
MAXIMUM	report maximum values
RANGE	report the range (maximum - minimum) of values

Example:

```
[TIMES]
DURATION            240 HOURS
QUALITY TIMESTEP    3 MIN
QUALITY TIMESTEP    0:03
REPORT START        120
START CLOCKTIME     6:00 AM
```

[REPORT]

Purpose: Describes the contents of the output report produced from a simulation.

Formats:

```
PAGESIZE        value
FILE            filename
STATUS          YES/NO/FULL
SUMMARY         YES/NO
MESSAGES        YES/NO
ENERGY          YES/NO
NODES           NONE/ALL/node1 node2 ...
LINKS           NONE/ALL/link1 link2 ...
variable        YES/NO
variable        BELOW/ABOVE/PRECISION value
```

Remarks:
1. All options assume their default values if not explicitly specified in this section.
2. Items offset by slashes (/) indicate allowable choices.
3. The default is to not report on any nodes or links, so a **NODES** or **LINKS** option must be supplied if you wish to report results for these items.

PAGESIZE sets the number of lines written per page of the output report. The default is `0`, meaning that no line limit per page is in effect.

FILE supplies the name of a file to which the output report will be written. If the file name contains spaces then it must be surrounded by double quotes. If not supplied then the Report file, as specified in the second parameter of the `ENopen` (or `ENepanet`) function will be used.

STATUS determines whether hydraulic status messages are written to the Report file. If **YES** is selected the messages will identify those network components that change status during each time step of the simulation. If **FULL** is selected, then convergence information will also be included from each trial of each hydraulic analysis. This level of detail is only useful for de-bugging networks that become hydraulically unbalanced. The default is **NO**.

SUMMARY determines whether a summary table of number of network components and key analysis options is generated. The default is **YES**.

MESSAGES determines whether error and warning messages generated during a hydraulic/water quality analysis are written to the Report file. The default is **YES.**

ENERGY determines if a table reporting average energy usage and cost for each pump is provided. The default is **NO**.

NODES identifies which nodes will be reported on. You can either list individual node ID labels or use the keywords **NONE** or **ALL**. Additional **NODES** lines can be used to continue the list. The default is **NONE**.

LINKS identifies which links will be reported on. You can either list individual link ID labels or use the keywords **NONE** or **ALL**. Additional **LINKS** lines can be used to continue the list. The default is **NONE**.

This reporting option is used to identify which variables are reported on, how many decimal places are displayed, and what kind of filtering should be used to limit output reporting. Node variables that can be reported on include:

- **Elevation**
- **Demand**
- **Head**
- **Pressure**
- **Quality**

Link variables include:

- **Length**
- **Diameter**
- **Flow**
- **Velocity**
- **Headloss**
- **LinkQuality**
- **LinkStatus**
- **Setting** (Roughness for pipes, speed for pumps, pressure/flow setting for valves)
- **Reaction** (reaction rate)
- **F-Factor** (friction factor)

The default reporting variables are Demand, Head, Pressure, and Quality for nodes and Flow, Velocity, and Headloss for links. The default precision is two decimal places.

Example:
The following example reports on nodes N1, N2, N3, and N17 and all links with velocity above 3.0. The standard node variables (Demand, Head, Pressure, and Quality) are reported on while only Flow, Velocity, and F-Factor (friction factor) are displayed for links.

```
[REPORT]
NODES N1 N2 N3 N17
LINKS ALL
FLOW YES
VELOCITY PRECISION 4
F-FACTOR PRECISION 4
VELOCITY ABOVE 3.0
```

6.3.2. Output File Format

The Toolkit uses an unformatted binary output file to store both hydraulic and water quality results at uniform reporting intervals. Data written to the file is either 4-byte integers, 4-byte floats, or fixed-size strings whose size is a multiple of 4 bytes. This allows the file to be divided conveniently into 4-byte records. The file consists of four sections of the following sizes in bytes:

Section	Size in Bytes
Prolog	852 + 20*Nnodes + 36*Nlinks + 8*Ntanks
Energy Use	28*Npumps + 4
Dynamic Results	(16*Nnodes + 32*Nlinks)*Nperiods
Epilog	28

where

Nnodes	=	number of nodes (junctions + reservoirs + tanks)
Nlinks	=	number of links (pipes + pumps + valves)
Ntanks	=	number of tanks and reservoirs
Npumps	=	number of pumps
Nperiods	=	number of reporting periods

and all of these counts are themselves written to the file's prolog or epilog sections.

Output File - Prolog

The prolog section of the binary Output File contains the following data:

Item	Type	Number of Bytes
Magic Number (= 516114521)	Integer	4
Version (= 200)	Integer	4
Number of Nodes (Junctions + Reservoirs + Tanks)	Integer	4
Number of Reservoirs & Tanks	Integer	4
Number of Links (Pipes + Pumps + Valves)	Integer	4
Number of Pumps	Integer	4
Number of Valves	Integer	4
Water Quality Option 0 = none 1 = chemical 2 = age 3 = source trace	Integer	4
Index of Node for Source Tracing	Integer	4

Flow Units Option 0 = cfs 1 = gpm 2 = mgd 3 = Imperial mgd 4 = acre-ft/day 5 = liters/second 6 = liters/minute 7 = megaliters/day 8 = cubic meters/hour 9 = cubic meters/day	Integer	4
Pressure Units Option 0 = psi 1 = meters 2 = kPa	Integer	4
Time Statistics Flag 0 = none (report time series) 1 = report time-averaged values 2 = report minimum values 3 = report maximum values 4 = report ranges	Integer	4
Reporting Start Time (seconds)	Integer	4
Reporting Time Step (seconds)	Integer	4
Simulation Duration (seconds)	Integer	4
Problem Title (1st line)	Char	80
Problem Title (2nd line)	Char	80
Problem Title (3rd line)	Char	80
Name of Input File	Char	260
Name of Report File	Char	260
Name of Chemical	Char	16
Chemical Concentration Units	Char	16
ID String of Each Node	Char	16*Nnodes
ID String of Each Link	Char	16*Nlinks
Index of Head Node of Each Link	Integer	4*Nlinks
Index of Tail Node of Each Link	Integer	4*Nlinks
Type Code of Each Link 0 = Pipe with CV 1 = Pipe 2 = Pump 3 = PRV 4 = PSV 5 = PBV 6 = FCV 7 = TCV 8 = GPV	Integer	4*Nlinks
Node Index of Each Tank	Integer	4*Ntanks

Cross-Sectional Area of Each Tank (value of 0 denotes a Reservoir)	Float	4*Ntanks
Elevation of Each Node	Float	4*Nnodes
Length of Each Link	Float	4*Nlinks
Diameter of Each Link	Float	4*Nlinks

Output File - Energy Use

The Energy Use section of the Output File contains the following data:

Item	Type	# Bytes
Repeated for Each Pump:		
▪ Pump Index in list of links	Integer	4
▪ Pump Utilization (%)	Float	4
▪ Average Efficiency (%)	Float	4
▪ Average kwatts/MGal (or kwatts/cu m)	Float	4
▪ Average kwatts	Float	4
▪ Peak kwatts	Float	4
▪ Average Cost per Day	Float	4
Peak Demand Cost	Float	4

Output File - Dynamic Results

The Dynamic Results section of the binary Output File contains the following set of data **for each reporting period** (the reporting time step is written to the Output File's prolog section and the number of such steps is written to the epilog section):

Item	Type	Number of Bytes
Demand at Each Node	Float	4*Nnodes
Head (Grade) at Each Node	Float	4*Nnodes
Pressure at Each Node	Float	4*Nnodes
Water Quality at Each Node	Float	4*Nnodes
Flow in Each Link (negative for reverse flow)	Float	4*Nlinks
Velocity in Each Link	Float	4*Nlinks
Headloss per 1000 Units of Length for Each Link (Total head for pumps and head loss for valves)	Float	4*Nlinks
Avg. Water Quality in Each Link	Float	4*Nlinks

Status Code for Each Link ▪ 0 = closed (max. head exceeded) ▪ 1 = temporarily closed ▪ 2 = closed ▪ 3 = open ▪ 4 = active (partially open) ▪ 5 = open (max. flow exceeded) ▪ 6 = open (flow setting not met) ▪ 7 = open (pressure setting not met)	Float	4*Nlinks
Setting for Each Link ▪ Roughness coeff. for Pipes, ▪ Speed for Pumps ▪ Setting for Valves	Float	4*Nlinks
Reaction Rate for Each Link (mass/L/day)	Float	4*Nlinks
Friction Factor for Each Link	Float	4*Nlinks

Output File - Epilog

The Epilog section of the binary Output File contains the following data:

Item	Type	# Bytes
Average bulk reaction rate (mass/hr)	Float	4
Average wall reaction rate (mass/hr)	Float	4
Average tank reaction rate (mass/hr)	Float	4
Average source inflow rate (mass/hr)	Float	4
Number of Reporting Periods	Integer	4
Warning Flag ▪ 0 = no warnings ▪ 1 = warnings were generated	Integer	4
Magic Number (= 516114521)	Integer	4

6.3.3. Example Input File

The following contains excerpts from an example EPANET Input file, using the formats described in Input File Format:

```
[TITLE]
Example EPANET Input File

[JUNCTIONS]
;ID     Elev.
;-----------
J1      100
J2      120
< etc. >

[RESERVOIRS]
;ID     Head
;-----------
R1      55

[TANKS]
;             Init.  Min.   Max.
;ID   Elev.   Level  Level  Level  Diam.
;--------------------------------------
T1    200      12     2     20      120

[PIPES]
;                                          Minor
;ID   Node1  Node2  Length  Diam.  Rough.  Loss   Status
;-------------------------------------------------------
P1    J1     J2     1200    12     100     0      Open
P2    J2     J3     2400    16     100     0      Open
P3    J3     J20     400     8     100     2.4    CV
< etc. >

[PUMPS]
;ID   Node1  Node2  Characteristics
;---------------------------------
PMP1  R1     J1     HEAD CURVE1

[DEMANDS]
;          Base   Demand
;Junction  Demand Pattern  Category
;----------------------------------
J1         50      PAT1    ;Domestic
J1         100     PAT2    ;Hospital
J2         55      PAT1    ;Domestic
< etc. >

[PATTERNS]
;ID   Multipliers
;-----------------------------------
PAT1  1.1  1.2  0.95  0.87  0.65  0.77
PAT1  0.83  1.0  1.1  1.4  1.2  1.1
```

```
PAT2  1.0
< etc. >

[CURVES]
;ID    X-value   Y-value
;----------------------
CURVE1   0        120
CURVE1   150       60
CURVE1   500        0

[REACTIONS]
GLOBAL BULK -0.5

[SOURCES]
;Node  Type  Strength
;-------------------
R1     CONCEN  1.0

[TIMES]
DURATION  24 HRS
PATTERN TIMESTEP 2 HRS

[OPTIONS]
QUALITY  Chlorine mg/L

[END]
```

6.4. Units of Measurement

NOTE: US Customary units apply when CFS, GPM, MGD, IMGD, or AFD are chosen as flow units. SI Metric units apply when flow units are expressed using either liters or cubic meters. See [OPTIONS] for how to select flow units.

Parameter	US Customary	SI Metric
Concentration	mg/L or ug/L	mg/L or ug/L
Demand	(see Flow units)	(see Flow units)
Diameter (Pipes)	inches	millimeters
Diameter (Tanks)	feet	meters
Efficiency	percent	percent
Elevation	feet	meters
Emitter Coeff.	flow units @ 1 psi drop	flow units @ 1 meter drop
Energy	kwatt - hours	kwatt - hours
Flow	CFS (cubic feet / sec)	LPS (liters / sec)
	GPM (gallons / min)	LPM (liters / min)
	MGD (million gal / day)	MLD (megaliters / day)
	IMGD (Imperial MGD)	CMH (cubic meters / hr)
	AFD (acre-feet / day)	CMD (cubic meters / day)
Friction Factor	unitless	unitless
Head	feet	meters
Length	feet	meters
Minor Loss Coeff.	unitless	unitless
Power	horsepower	kwatts
Pressure	psi	meters
Reaction Coeff. (Bulk)	1/day (1st-order)	1/day (1st-order)
Reaction Coeff. (Wall)	mass/sq-ft/day (0-order)	mass/sq-m/day (0-order)
	ft/day (1st-order)	meters/day (1st-order)
Roughness Coeff.	millifeet (Darcy-Weisbach)	mm (Darcy-Weisbach)
	unitless otherwise	unitless otherwise
Source Mass Injection	mass/minute	mass/minute
Velocity	ft/sec	meters/sec
Volume	cubic feet	cubic meters
Water Age	hours	hours

ENepanet

```
int ENepanet( char* f1, char* f2, char* f3, void (*) (vfunc) )
```

Description:

Runs a complete EPANET simulation.

Arguments:

f1:	name of the input file
f2:	name of an output report file
f3:	name of an optional binary output file
vfunc:	pointer to a user-supplied function which accepts a character string as its argument.

Returns:

Returns an error code.

Notes:

ENepanet is a stand-alone function and does not interact with any of the other functions in the toolkit.

If there is no need to save EPANET's binary output file then *f3* can be an empty string ("").

The *vfunc* function pointer allows the calling program to display a progress message generated by EPANET during its computations. A typical function for a console application might look as follows:

```
void  writecon(char *s)
{
   puts(s);
}
```

and somewhere in the calling program the following declarations would appear:

```
void (* vfunc) (char *);
vfunc = writecon;
ENepanet(f1,f2,f3,vfunc);
```

If such a function is not desired then this argument should be *NULL* (*NIL* for Delphi/Pascal, *VBNULLSTRING* for Visual Basic).

ENepanet is used mainly to link the EPANET engine to third-party user interfaces that build network input files and display the results of a network analysis.

ENopen

```
int ENopen( char* f1, char* f2, char* f3)
```

Description: Opens the Toolkit to analyze a particular distribution system.

Arguments:

f1: name of an EPANET Input file
f2: name of an output Report file
f3: name of an optional binary Output file.

Returns: Returns an error code.

Notes:

If there is no need to save EPANET's binary Output file then *f3* can be an empty string (`""`).

If *f2* is an empty string, then reporting will be made to the operating system's *stdout* device (which is usually the console).

`ENopen` must be called before any of the other toolkit functions (except `ENepanet`) are used.

See Also: `ENclose`

ENclose

```
int ENclose( void )
```

Description: Closes down the Toolkit system (including all files being processed).

Returns: Returns an error code.

Notes:

ENclose must be called when all processing has been completed, even if an error condition was encountered.

See Also: `ENopen`

ENgetnodeindex

```
int ENgetnodeindex( char* id, int* index )
```

Description: Retrieves the index of a node with a specified ID.

Arguments:

id: node ID label
index: node index

Returns: Returns an error code.

Notes: Node indexes are consecutive integers starting from `1`.

See Also: `ENgetnodeid`

ENgetnodeid

`int` ENgetnodeid(**`int`** index, **`char*`** id)

Description: Retrieves the ID label of a node with a specified index.

Arguments:

index: node index
id: ID label of node

Returns: Returns an error code.

Notes:

The ID label string should be sized to hold at least `15` characters.
Node indexes are consecutive integers starting from `1`.

See Also: `ENgetnodeindex`

ENgetnodetype

```
int ENgetnodetype( int index, int* typecode )
```

Description: Retrieves the node-type code for a specific node.

Arguments:

index: node index
typecode: node-type code (see below)

Returns: Returns an error code.

Notes: Node indexes are consecutive integers starting from `1`. Node type codes consist of the following constants:

`EN_JUNCTION`	`0`	Junction node
`EN_RESERVOIR`	`1`	Reservoir node
`EN_TANK`	`2`	Tank node

ENgetnodevalue

```
int ENgetnodevalue( int index, int paramcode, float* value )
```

Description: Retrieves the value of a specific link parameter.

Arguments:

index: node index
paramcode: parameter code (see below)
value: parameter value

Returns: Returns an error code.

Notes: Node indexes are consecutive integers starting from 1.

Node parameter codes consist of the following constants:

EN_ELEVATION	0	Elevation
EN_BASEDEMAND	1	Base demand
EN_PATTERN	2	Demand pattern index
EN_EMITTER	3	Emitter coeff.
EN_INITQUAL	4	Initial quality
EN_SOURCEQUAL	5	Source quality
EN_SOURCEPAT	6	Source pattern index
EN_SOURCETYPE	7	Source type (See note below)
EN_TANKLEVEL	8	Initial water level in tank
EN_DEMAND	9	Actual demand
EN_HEAD	10	Hydraulic head
EN_PRESSURE	11	Pressure
EN_QUALITY	12	Actual quality
EN_SOURCEMASS	13	Mass flow rate per minute of a chemical source

Parameters 9 - 13 (EN_DEMAND through EN_SOURCEMASS) are computed values. The others are input design parameters.

Source types are identified with the following constants:

EN_CONCEN	0
EN_MASS	1
EN_SETPOINT	2
EN_FLOWPACED	3

See [SOURCES] for a description of these source types.

Values are returned in units which depend on the units used for flow rate in the EPANET input file (see Units of Measurement).

ENgetlinkindex

```
int ENgetlinkindex( char* id, int* index )
```

Description: Retrieves the index of a link with a specified ID.

Arguments:

id: link ID label
index: link index

Returns: Returns an error code.

Notes: Link indexes are consecutive integers starting from 1.

See Also: `ENgetlinkid`

ENgetlinkid

```
int ENgetlinkid( int index, char* id )
```

Description: Retrieves the ID label of a link with a specified index.

Arguments:

index: link index
id: ID label of link

Returns: Returns an error code.

Notes:

The ID label string should be sized to hold at least `15` characters.

Link indexes are consecutive integers starting from `1`.

See Also: `ENgetlinkindex`

ENgetlinktype

```
int ENgetlinktype( int index, int* typecode )
```

Description: Retrieves the link-type code for a specific link.

Arguments:

index: link index
typecode: link-type code (see below)

Returns: Returns an error code.

Notes:

Link indexes are consecutive integers starting from 1.

Link type codes consist of the following constants:

EN_CVPIPE	0	Pipe with Check Valve
EN_PIPE	1	Pipe
EN_PUMP	2	Pump
EN_PRV	3	Pressure Reducing Valve
EN_PSV	4	Pressure Sustaining Valve
EN_PBV	5	Pressure Breaker Valve
EN_FCV	6	Flow Control Valve
EN_TCV	7	Throttle Control Valve
EN_GPV	8	General Purpose Valve

See Also: ENgetlinkindex

ENgetlinknodes

```
int ENgetlinknodes( int index, int* fromnode, int* tonode )
```

Description: Retrieves the indexes of the end nodes of a specified link.

Arguments:

index:	link index
fromnode:	index of node at start of link
tonode:	index of node at end of link

Returns: Returns an error code.

Notes:

Node and link indexes are consecutive integers starting from 1.
The From and To nodes are as defined for the link in the EPANET input file. The actual direction of flow in the link is not considered.

See Also: ENgetlinkindex

ENgetlinkvalue

```
int ENgetlinkvalue( int index, int paramcode, float* value )
```

Description: Retrieves the value of a specific link parameter.

Arguments:

index: link index
paramcode: parameter code (see below)
value: parameter value

Returns: Returns an error code.

Notes:

Link indexes are consecutive integers starting from 1.

Link parameter codes consist of the following constants:

EN_DIAMETER	0	Diameter
EN_LENGTH	1	Length
EN_ROUGHNESS	2	Roughness coeff.
EN_MINORLOSS	3	Minor loss coeff.
EN_INITSTATUS	4	Initial link status (0 = closed, 1 = open)
EN_INITSETTING	5	Initial pipe roughness Initial pump speed Initial valve setting
EN_KBULK	6	Bulk reaction coeff.
EN_KWALL	7	Wall reaction coeff.
EN_FLOW	8	Flow rate
EN_VELOCITY	9	Flow velocity
EN_HEADLOSS	10	Head loss
EN_STATUS	11	Actual link status (0 = closed, 1 = open)
EN_SETTING	12	Pipe roughness Actual pump speed Actual valve setting
EN_ENERGY	13	Energy expended in kwatts

Parameters 8 – 13 (EN_FLOW through EN_ENERGY) are computed values. The others are design parameters.

Flow rate is positive if the direction of flow is from the designated start node of the link to its designated end node, and negative otherwise.

Values are returned in units which depend on the units used for flow rate in the EPANET input file (see Units of Measurement).

See Also: ENgetlinkindex

ENgetpatternid

```
int ENgetpatternid( int index, char* id )
```

Description: Retrieves the ID label of a particular time pattern.

Arguments:

index: pattern index
id: ID label of pattern

Returns: Returns an error code.

Notes:

The ID label string should be sized to hold at least 15 characters.

Pattern indexes are consecutive integers starting from 1.

ENgetpatternindex

```
int ENgetpatternindex( char* id, int* index )
```

Description: Retrieves the index of a particular time pattern.

Arguments:

id: pattern ID label
index: pattern index

Returns: Returns an error code.

Notes: Pattern indexes are consecutive integers starting from 1.

ENgetpatternlen

```
int ENgetpatternlen( int index, int* len )
```

Description: Retrieves the number of time periods in a specific time pattern.

Arguments:

index: pattern index
len: number of time periods in the pattern

Returns: Returns an error code.

Notes: Pattern indexes are consecutive integers starting from 1.

ENgetpatternvalue

```
int ENgetpatternvalue( int index, int period, float* value )
```

Description: Retrieves the multiplier factor for a specific time period in a time pattern.

Arguments:

index:	time pattern index
period:	period within time pattern
value:	multiplier factor for the period

Returns: Returns an error code.

Notes: Pattern indexes and periods are consecutive integers starting from 1.

See Also: ENgetpatternindex, ENgetpatternlen, ENsetpatternvalue

ENgetcontrol

```
int ENgetcontrol( int cindex, int* ctype, int* lindex,
                  float* setting, int* nindex, float* level )
```

Description:

Retrieves the parameters of a simple control statement. The index of the control is specified in cindex and the remaining arguments return the control's parameters.

Arguments:

cindex:	control statement index
ctype:	control type code
lindex:	index of link being controlled
setting:	value of the control setting
nindex:	index of controlling node
level:	value of controlling water level or pressure for level controls or of time of control action (in seconds) for time-based controls

Returns: Returns an error code.

Notes:

Controls are indexed starting from 1 in the order in which they were entered into the [CONTROLS] section of the EPANET input file.

Control type codes consist of the following:

0 (Low Level Control)	applies when tank level or node pressure drops below specified level
1 (High Level Control)	applies when tank level or node pressure rises above specified level
2 (Timer Control)	applies at specific time into simulation
3 (Time-of-Day Control)	applies at specific time of day

For pipes, a *setting* of 0 means the pipe is closed and 1 means it is open. For a pump, the *setting* contains the pump's speed, with 0 meaning the pump is closed and 1 meaning it is open at its normal speed. For a valve, the *setting* refers to the valve's pressure, flow, or loss coefficient value, depending on valve type

For Timer or Time-of-Day controls the *nindex* parameter equals 0.

See ENsetcontrol for an example of using this function.

ENgetcount

```
int ENgetcount( int countcode,
int*count )
```

Description:

Retrieves the number of network components of a specified type.

Arguments:

countcode:	component code (see below)
count:	number of *countcode* components in the network

Returns: Returns an error code.

Notes:

Component codes consist of the following:

EN_NODECOUNT	0	Nodes
EN_TANKCOUNT	1	Reservoirs and tank nodes
EN_LINKCOUNT	2	Links
EN_PATCOUNT	3	Time patterns
EN_CURVECOUNT	4	Curves
EN_CONTROLCOUNT	5	Simple controls

The number of junctions in a network equals the number of nodes minus the number of tanks and reservoirs.

There is no facility within the Toolkit to add to or delete from the components described in the Input file.

ENgetflowunits

```
int ENgetflowunits( int* unitscode )
```

Description:

Retrieves a code number indicating the units used to express all flow rates.

Arguments:

unitscode: value of a flow units code number (see below).

Returns: Returns an error code.

Notes: Flow units codes are as follows:

0	= EN_CFS	cubic feet per second
1	= EN_GPM	gallons per minute
2	= EN_MGD	million gallons per day
3	= EN_IMGD	Imperial mgd
4	= EN_AFD	acre-feet per day
5	= EN_LPS	liters per second
6	= EN_LPM	liters per minute
7	= EN_MLD	million liters per day
8	= EN_CMH	cubic meters per hour
9	= EN_CMD	cubic meters per day

Flow units are specified in the [OPTIONS] section of the EPANET Input file.

Flow units in liters or cubic meters implies that metric units are used for all other quantities in addition to flow. Otherwise US units are employed. (See Units of Measurement).

ENgettimeparam

```
int ENgettimeparam( int paramcode, long* timevalue )
```

Description: Retrieves the value of a specific analysis time parameter.

Arguments:

paramcode: time parameter code (see below)
timevalue: value of time parameter in seconds

Returns: Returns an error code.

Notes:

Time parameter codes consist of the following constants:

EN_DURATION	0	Simulation duration
EN_HYDSTEP	1	Hydraulic time step
EN_QUALSTEP	2	Water quality time step
EN_PATTERNSTEP	3	Time pattern time step
EN_PATTERNSTART	4	Time pattern start time
EN_REPORTSTEP	5	Reporting time step
EN_REPORTSTART	6	Report starting time
EN_RULESTEP	7	Time step for evaluating rule-based controls
EN_STATISTIC	8	Type of time series post-processing used: 0 = none 1 = averaged 2 = minimums 3 = maximums 4 = ranges
EN_PERIODS	9	Number of reporting periods saved to binary output file

ENgetqualtype

```
int ENgetqualtype( int* qualcode, int* tracenode )
```

Description: Retrieves the type of water quality analysis called for.

Arguments:

qualcode: water quality analysis code (see below)
tracenode: index of node traced in a source tracing analysis

Returns: Returns an error code.

Notes: Water quality analysis codes are as follows:

EN_NONE	0	No quality analysis
EN_CHEM	1	Chemical analysis
EN_AGE	2	Water age analysis
EN_TRACE	3	Source tracing

The *tracenode* value will be 0 when *qualcode* is not EN_TRACE.

See Also: ENsetqualtype

ENgetoption

```
int ENgetoption( int optioncode, float* value )
```

Description:

Retrieves the value of a particular analysis option.

Arguments:

optioncode: an option code (see below)
value: an option value

Returns: Returns an error code.

Notes: Option codes consist of the following constants:

EN_TRIALS	0
EN_ACCURACY	1
EN_TOLERANCE	2
EN_EMITEXPON	3
EN_DEMANDMULT	4

ENgetversion

```
int ENgetversion( int* v )
```

Description: Retrieves the current version number of the Toolkit.

Arguments:

v: version number

Returns: Returns an error code (should always be 0).

Notes: The version number is a 5-digit integer that increases sequentially from 20001 with each new update of the Toolkit.

ENsetcontrol

```
int ENsetcontrol( int cindex, int ctype, int lindex,
                  float setting, int nindex, float level )
```

Description: Sets the parameters of a particular simple control statement.

Arguments:

cindex: control statement index
ctype: control type code
lindex: index of link being controlled
setting: value of the control setting
nindex: index of controlling node
level: value of controlling water level or pressure for level controls or of time of control action (in seconds) for time-based controls

Returns: Returns an error code.

Notes:

Controls are indexed starting from 1 in the order in which they were entered into the [CONTROLS] section of the EPANET input file.

Control type codes consist of the following:

EN_LOWLEVEL	0	Control applied when tank level or node pressure drops below specified level
EN_HILEVEL	1	Control applied when tank level or node pressure rises above specified level
EN_TIMER	2	Control applied at specific time into simulation
EN_TIMEOFDAY	3	Control applied at specific time of day

For pipes, a *setting* of 0 means the pipe is closed and 1 means it is open. For a pump, the *setting* contains the pump's speed, with 0 meaning the pump is closed and 1 meaning it is open at its normal speed. For a valve, the *setting* refers to the valve's

pressure, flow, or loss coefficient, depending on valve type.

For Timer or Time-of-Day controls set the *nindex* parameter to 0.

For level controls, if the controlling node *nindex* is a tank then the `level` parameter should be a water level above the tank bottom (not an elevation). Otherwise *level* should be a junction pressure.

To remove a control on a particular link, set the *lindex* parameter to 0. Values for the other parameters in the function will be ignored.

Example:

This example uses `ENgetcontrol` and `ENsetcontrol` to change the low level setting on the node that controls a link with index *thelink* to a new value *newlevel*.

```
ENgetcount(EN_CONTROLS, &numctrls);
for (i=1; i<=numctrls; i++)
{
  ENgetcontrol(i, &ctype, &lindex, &setting,
    &nindex, &level);
  if (ctype == EN_LOWLEVEL && lindex == thelink)
  {
    ENsetcontrol(i, ctype, lindex, setting,
      nindex, newlevel);
    break;
  }
}
```

See Also: `ENgetcontrol`

ENsetnodevalue

```
int ENsetnodevalue( int index, int paramcode, float value )
```

Description: Sets the value of a parameter for a specific node.

Arguments:

index:	node index
paramcode:	parameter code (see below)
value:	parameter value

Returns: Returns an error code.

Notes: Node indexes are consecutive integers starting from 1.

Node parameter codes consist of the following constants:

EN_ELEVATION	0	Elevation
EN_BASEDEMAND	1	Baseline demand
EN_PATTERN	2	Time pattern index
EN_EMITTER	3	Emitter coefficient
EN_INITQUAL	4	Initial quality
EN_SOURCEQUAL	5	Source quality
EN_SOURCEPAT	6	Source pattern
EN_SOURCETYPE	7	Source type (See note below)
EN_TANKLEVEL	8	Initial water level in tank

Source types are identified with the following constants:

EN_CONCEN	0
EN_MASS	1
EN_SETPOINT	2
EN_FLOWPACED	3

See [SOURCES] for a description of these source types.

Values are supplied in units which depend on the units used for flow rate in the EPANET input file (see Units of Measurement).

ENsetlinkvalue

```
int ENsetlinkvalue( int index, int paramcode, float value )
```

Description: Sets the value of a parameter for a specific link.

Arguments:

index:	link index
paramcode:	parameter code (see below)
value:	parameter value

Returns: Returns an error code.

Notes:

Link indexes are consecutive integers starting from 1.

Link parameter codes consist of the following constants:

EN_DIAMETER	0	Diameter
EN_LENGTH	1	Length
EN_ROUGHNESS	2	Roughness coeff.
EN_MINORLOSS	3	Minor loss coeff.
EN_INITSTATUS	4	Initial link status (0 = closed, 1 = open)
EN_INITSETTING	5	Pipe roughness Initial pump speed Initial valve setting
EN_KBULK	6	Bulk reaction coeff.

EN_KWALL	7	Wall reaction coeff.
EN_STATUS	11	Current pump or valve status (0 = closed, 1 = open)
EN_SETTING	12	Current pump speed or valve setting

Values are supplied in units which depend on the units used for flow rate in the EPANET input file (see Units of Measurement).

Use EN_INITSTATUS and EN_INITSETTING to set the design value for a link's status or setting that exists prior to the start of a simulation. Use EN_STATUS and EN_SETTING to change these values while a simulation is being run (within the ENrunH - ENnextH loop).

If a control valve has its status explicitly set to OPEN or CLOSED, then to make it active again during a simulation you must provide a new valve setting value using the EN_SETTING parameter.

For pipes, either EN_ROUGHNESS or EN_INITSETTING can be used to change roughness.

ENsetpattern

```
int ENsetpattern( int index, float* factors, int nfactors )
```

Description:

Sets all of the multiplier factors for a specific time pattern.

Arguments:

index:	time pattern index
factors:	multiplier factors for the entire pattern
nfactors:	number of factors in the pattern

Returns: Returns an error code.

Notes:

Pattern indexes are consecutive integers starting from 1.

factors points to a zero-based array that contains *nfactors* elements.

Use this function to redefine (and resize) a time pattern all at once; use ENsetpatternvalue to revise pattern factors in specific time periods of a pattern.

See Also: ENgetpatternindex, ENgetpatternlen, ENgetpatternvalue, ENsetpatternvalue

ENsetpatternvalue

```
int ENsetpatternvalue( int index, int period, float value )
```

Description: Sets the multiplier factor for a specific period within a time pattern.

Arguments:

index:	time pattern index
period:	period within time pattern
value:	multiplier factor for the period

Returns: Returns an error code.

Notes:

Pattern indexes are consecutive integers starting from 1.

Use `ENsetpattern` to reset all of the factors in a time pattern.

See Also: `ENgetpatternindex, ENgetpatternlen, ENgetpatternvalue, ENsetpattern`

ENsetqualtype

```
int ENsetqualtype( int qualcode, char* chemname, char* chemunits,
                                                 char* tracenode )
```

Description: Sets the type of water quality analysis called for.

Arguments:

qualcode:	water quality analysis code (see below)
chemname:	name of the chemical being analyzed
chemunits:	units that the chemical is measured in
tracenode:	ID of node traced in a source tracing analysis

Returns: Returns an error code.

Notes: Water quality analysis codes are as follows:

EN_NONE	0	No quality analysis
EN_CHEM	1	Chemical analysis
EN_AGE	2	Water age analysis
EN_TRACE	3	Source tracing

Chemical name and units can be an empty string if the analysis is not for a chemical.
The same holds for the trace node if the analysis is not for source tracing.
Note that the trace node is specified by ID and not by index.

See Also: `ENgetqualtype`

ENsettimeparam

int ENsettimeparam(**int** paramcode, long timevalue)

Description: Sets the value of a time parameter.

Arguments:

paramcode:	time parameter code (see below)
timevalue:	value of time parameter in seconds

Returns: Returns an error code.

Notes: Time parameter codes consist of the following constants:

EN_DURATION	0	Simulation duration
EN_HYDSTEP	1	Hydraulic time step
EN_QUALSTEP	2	Water quality time step
EN_PATTERNSTEP	3	Time pattern time step
EN_PATTERNSTART	4	Time pattern start time
EN_REPORTSTEP	5	Reporting time step
EN_REPORTSTART	6	Report starting time
EN_RULESTEP	7	Time step for evaluating rule-based controls
EN_STATISTIC	8	Type of time series post-processing to use: EN_NONE (0) = none EN_AVERAGE (1) = averaged EN_MINIMUM (2) = minimums EN_MAXIMUM (3) = maximums EN_RANGE (4) = ranges

Do not change time parameters after calling ENinitH in a hydraulic analysis or ENinitQ in a water quality analysis.

ENsetoption

int ENsetoption(**int** optioncode, **float** value)

Description: Sets the value of a particular analysis option.

Arguments:

optioncode:	an option code (see below)
value:	an option value

Returns: Returns an error code.

Notes: Option codes consist of the following constants:

EN_TRIALS	0
EN_ACCURACY	1
EN_TOLERANCE	2
EN_EMITEXPON	3
EN_DEMANDMULT	4

ENsavehydfile

```
int ENsavehydfile( char* fname )
```

Description: Saves the current contents of the binary hydraulics file to a file.

Arguments:

fname: name of the file where the hydraulics results should be saved.

Returns: Returns an error code.

Notes:

Use this function to save the current set of hydraulics results to a file, either for post-processing or to be used at a later time by calling the ENusehydfile function.

The hydraulics file contains nodal demands and heads and link flows, status, and settings for all hydraulic time steps, even intermediate ones.

Before calling this function hydraulic results must have been generated and saved by having called ENsolveH or the ENinitH - ENrunH - ENnextH sequence with the saveflag parameter of ENinitH set to 1.

See Also: ENusehydfile, ENsolveH, ENinitH

ENusehydfile

```
int ENusehydfile( char* fname )
```

Description: Uses the contents of the specified file as the current binary hydraulics file.

Arguments:

fname: name of the file containing hydraulic analysis results for the current network.

Returns: Returns an error code.

Notes:

Call this function to refuse a set of hydraulic analysis results saved previously. These results are checked to see if they match the following the parameters associated with the current network being analyzed: number of nodes, number of tanks and reservoirs, number of links, number of pumps, number of valves, and simulation duration.

Do not call this function when the hydraulics analysis system is still opened (i.e., ENopenH has been called but ENcloseH has not).

See Also: ENsavehydfile

ENsolveH

```
int ENsolveH( void )
```

Description:

Runs a complete hydraulic simulation with results for all time periods written to the binary Hydraulics file.

Returns: Returns an error code.

Notes:

Use `ENsolveH` to generate a complete hydraulic solution which can stand alone or be used as input to a water quality analysis. It can also be followed by calls to `ENsaveH` and `ENreport` to write a report on hydraulic results to the report file. Do not use `ENopenH`, `ENinitH`, `ENrunH`, `ENnextH`, and `ENcloseH` in conjunction with `ENsolveH`.

Example:

```
ENopen("net1.inp", "net1.rpt", "");
ENsolveH();
ENsolveQ();
ENreport();
ENclose();
```

ENopenH

```
int ENopenH( void )
```

Description: Opens the hydraulics analysis system.

Returns: Returns an error code.

Notes:

Call `ENopenH` prior to running the first hydraulic analysis using the `ENinitH` - `ENrunH` - `ENnextH` sequence. Multiple analyses can be made before calling `ENcloseH` to close the hydraulic analysis system.

Do not call this function if `ENsolveH` is being used to run a complete hydraulic analysis.

See Also: `ENinitH, ENrunH, ENnextH, ENcloseH`

ENinitH

```
int ENinitH( int flag )
```

Description:

Initializes storage tank levels, link status and settings, and the simulation clock time prior to running a hydraulic analysis.

Arguments:

flag: Two-digit flag indicating if hydraulic results will be saved to the hydraulics file (rightmost digit) and if link flows should be re-initialized.

Returns: Returns an error code.

Notes:

Call ENinitH prior to running a hydraulic analysis using ENrunH and ENnextH. ENopenH must have been called prior to calling ENinitH.
Do not call ENinitH if a complete hydraulic analysis is being made with a call to ENsolveH.
Values of *flag* have the following meanings:

00	do not re-initialize flows, do not save results to file
01	do not re-initialize flows, save results to file
10	re-initialize flows, do not save results to file
11	re-initialize flows, save results to file

Set *flag* to 1 (or 11) if you will be making a subsequent water quality run, using ENreport to generate a report, or using ENsavehydfile to save the binary hydraulics file.

See Also: ENopenH, ENrunH, ENnextH, ENcloseH

ENrunH

```
int ENrunH( long* t )
```

Description:

Runs a single period hydraulic analysis, retrieving the current simulation clock time *t*.

Arguments:

t: current simulation clock time in seconds.

Returns: Returns an error code.

Notes:

Use ENrunH along with ENnextH in a do...while loop to analyze hydraulics in each period of an extended period simulation. This process automatically updates the

simulation clock time so treat *t* as a read-only variable.

ENinitH must have been called prior to running the ENrunH - ENnextH loop.

See ENnextH for an example of using this function.

See Also: ENopenH, ENinitH, ENnextH, ENcloseH

ENnextH

```
int ENnextH( long* tstep )
```

Description:

Determines the length of time until the next hydraulic event occurs in an extended period simulation.

Arguments:

tstep: time (in seconds) until next hydraulic event occurs or 0 if at the end of the simulation period.

Returns: Returns an error code.

Notes:

This function is used in conjunction with ENrunH to perform an extended period hydraulic analysis (see example below).

The value of *tstep* should be treated as a read-only variable. It is automatically computed as the smaller of:

- the time interval until the next hydraulic time step begins
- the time interval until the next reporting time step begins
- the time interval until the next change in demands occurs
- the time interval until a tank becomes full or empty
- the time interval until a control or rule fires

Example:

```
long t, tstep;
ENopenH();
ENinitH(0);
do {
   ENrunH(&t);
   /* Retrieve hydraulic results for time t  */
   ENnextH(&tstep);
}  while (tstep > 0);
ENcloseH();
```

See Also: ENopenH, ENinitH, ENrunH, ENcloseH, ENsettimeparam

ENcloseH

```
int ENcloseH( void )
```

Description: Closes the hydraulic analysis system, freeing all allocated memory.

Returns: Returns an error code.

Notes: Call ENcloseH after all hydraulics analyses have been made using ENinitH - ENrunH - ENnextH. Do not call this function if ENsolveH is being used.

See Also: ENopenH, ENinitH, ENrunH, ENnextH

ENsolveQ

```
int ENsolveQ( void )
```

Description: Runs a complete water quality simulation with results at uniform reporting intervals written to EPANET's binary Output file.

Returns: Returns an error code.

Notes:

A hydraulic analysis must have been run and saved to the binary hydraulics file before calling ENsolveQ. It can be followed by a call to ENreport to write a report on hydraulic and water quality results to the report file. Do not use ENopenQ, ENinitQ, ENrunQ, ENnextQ, and ENcloseQ in conjunction with ENsolveQ.

Example:

```
ENopen("net1.inp", "net1.rpt", "");
ENsolveH();
ENsolveQ();
ENreport();
ENclose();
```

ENopenQ

```
int ENopenQ( void )
```

Description: Opens the water quality analysis system.

Returns: Returns an error code.

Notes:

Call ENopenQ prior to running the first water quality analysis using an ENinitQ - ENrunQ - ENnextQ (or ENstepQ) sequence. Multiple water quality analyses can be made before calling ENcloseQ to close the water quality analysis system.

Do not call this function if a complete water quality analysis is being made using ENsolveQ.

See Also: ENintQ, ENrunQ, ENnextQ, ENstepQ, ENcloseQ

ENinitQ

```
int ENinitQ( int saveflag )
```

Description: Initializes water quality and the simulation clock time prior to running a water quality analysis.

Arguments:

saveflag: 0-1 flag indicating if analysis results should be saved to EPANET's binary output file at uniform reporting periods.

Returns: Returns an error code.

Notes:

Call ENinitQ prior to running a water quality analysis using ENrunQ in conjunction with either ENnextQ or ENstepQ.

ENopenQ must have been called prior to calling ENinitQ.

Do not call ENinitQ if a complete water quality analysis is being made with a call to ENsolveQ.

Set *saveflag* to 1 if you intend to use ENreport to generate a report or wish to save computed results to the binary output file.

See Also: ENopenQ, ENrunQ, ENnextQ, ENstepQ, ENcloseQ

ENrunQ

```
int ENrunQ( long* t )
```

Description:

Makes available the hydraulic and water quality results that occur at the start of the next time period of a water quality analysis, where the start of the period is returned in *t*.

Arguments:

t: current simulation clock time in seconds.

Returns: Returns an error code.

Notes:

Use ENrunQ along with ENnextQ in a do...while loop to access water quality results at the start of each hydraulic period in an extended period simulation. Or use it with ENstepQ in a do...while loop to access results at the start of each water quality time step. See each of these functions for examples of how to code such loops.

ENinitQ must have been called prior to running an ENrunQ - ENnextQ (or ENstepQ) loop.

The current time *t* of the simulation is determined from information saved with the hydraulic analysis that preceded the water quality analysis. Treat it as a read-only variable.

See Also: ENopenQ, ENinitQ, ENnextQ, ENstepQ, ENcloseQ

ENnextQ

```
int ENnextQ( long* tstep )
```

Description: Advances the water quality simulation to the start of the next hydraulic time period.

Arguments:

tstep: time (in seconds) until next hydraulic event occurs or 0 if at the end of the simulation period.

Returns: Returns an error code.

Notes:

This function is used in a do-while loop with ENrunQ to perform an extended period water quality analysis. It allows you to access water quality results at each hydraulic period of the simulation. The water quality routing and reactions are carried out internally at a much smaller time step. Use ENstepQ instead of this function if you need to access results after each water quality time step.

The value of *tstep* is determined from information saved with the hydraulic analysis that preceded the water quality analysis. Treat it as a read-only variable.

Example:

```
long t, tstep;
ENsolveH(); /* Generate & save hydraulics */
ENopenQ();
ENinitQ(0);
do {
   ENrunQ(&t);
   /* Monitor results at time t, which
      begins a new hydraulic time period
   */
   ENnextQ(&tstep);
}  while (tstep > 0)
ENcloseQ();
```

See Also: ENopenQ, ENinitQ, ENrunQ, ENcloseQ

ENstepQ

```
int ENstepQ( long* tleft )
```

Description:

Advances the water quality simulation one water quality time step. The time remaining in the overall simulation is returned in *tleft.*

Arguments:

tleft: seconds remaining in the overall simulation duration.

Returns: Returns an error code.

Notes:

This function is used in a do-while loop with ENrunQ to perform an extended period water quality simulation. It allows you to access water quality results at each water quality time step of the simulation, rather than at the start of each hydraulic period as with ENnextQ.

Use the argument *tleft* to determine when no more calls to ENrunQ are needed because the end of the simulation period has been reached (i.e., when *tleft* = 0).

Treat *tleft* as a read-only variable (do not assign it a value).

Example:

```
long t, tleft;
ENsolveH(); /* Generate & save hydraulics */
ENopenQ();
ENinitQ(0);
do {
   ENrunQ(&t);
   /* Monitor results at time t */
   ENstepQ(&tleft);
}  while (tleft > 0)
ENcloseQ();
```

See Also: ENopenQ, ENinitQ, ENrunQ, ENcloseQ

ENcloseQ

```
int ENcloseQ( void )
```

Description: Closes the water quality analysis system, freeing all allocated memory.

Returns: Returns an error code.

Notes:

Call ENcloseQ after all water quality analyses have been made using the ENinitQ - ENrunQ - ENnextQ (or ENstepQ) sequence of function calls. Do not call this function if ENsolveQ is being used.

See Also: ENopenQ, ENinitQ, ENrunQ, ENstepQ, ENnextQ

ENsaveH

```
int ENsaveH( void )
```

Description:

Transfers results of a hydraulic simulation from the binary Hydraulics file to the binary Output file, where results are only reported at uniform reporting intervals.

Returns: Returns an error code.

Notes:

ENsaveH is used when only a hydraulic analysis is run and results at uniform reporting intervals need to be transferred to EPANET's binary output file. Such would be the case when an output report to EPANET's report file will be written using ENreport.

The reporting times can be set either in the EPANET input file (in its [TIMES] section) or by using the ENsettimeparam function.

See Also: ENreport, ENsettimeparam

ENsaveinpfile

```
int ENsaveinpfile( char* fname )
```

Description:

Writes all current network input data to a file using the format of an EPANET input file.

Arguments:

fname: name of the file where data is saved.

Returns: Returns an error code.

Notes:

The data saved reflect any changes made by calls to the `ENsetxxx` family of functions since EPANET data was first loaded using `ENopen`.

ENreport

```
int ENreport( void )
```

Description: Writes a formatted text report on simulation results to the Report file.

Returns: Returns an error code.

Notes:

Either a full hydraulic analysis or full hydraulic and water quality analysis must have been run, with results saved to file, before `ENreport` is called. In the former case, `ENsaveH` must also be called first to transfer results from the Hydraulics file to the Output file.

The format of the report is controlled by commands placed in the `[REPORT]` section of the EPANET input file or by similar commands issued with the `ENsetreport` function.

ENresetreport

```
int ENresetreport( void )
```

Description:

Clears any report formatting commands that either appeared in the `[REPORT]` section of the EPANET Input file or were issued with the `ENsetreport` function.

Returns: Returns an error code.

Notes:

After calling this function the default reporting options are in effect. These are:

- No status report
- No energy report
- No nodes reported on
- No links reported on
- Node variables reported to 2 decimal places
- Link variables reported to 2 decimal places (3 for friction factor)
- Node variables reported are elevation, head, pressure, and quality
- Link variables reported are flow, velocity, and head loss

See Also: `ENreport, ENsetreport, ENsetstatusreport`

ENsetreport

```
int ENsetreport( char* command )
```

Description:

Issues a report formatting command. Formatting commands are the same as used in the `[REPORT]` section of the EPANET Input file.

Arguments:

`command:` text of a report formatting command.

Returns: Returns an error code.

Notes:

Call `ENresetreport` to clear any previous report formatting commands that either appeared in the Input file or were issued with calls to `ENsetreport` or `ENsetstatusreport`. Formatted results of a simulation can be written to the Report file using the `ENreport` function.

See Also: `ENreport, ENresetreport, ENsetstatusreport`

ENsetstatusreport

```
int ENsetstatusreport( int statuslevel )
```

Description: Sets the level of hydraulic status reporting.

Arguments:

`statuslevel:` level of status reporting (see below).

Returns: Returns an error code.

Notes:

Status reporting writes changes in the hydraulics status of network elements to the Report file as a hydraulic simulation unfolds. There are three levels of reporting:

0 - no status reporting
1 - normal reporting
2 - full status reporting

The full status report contains information on the convergence of each trial of the solution to the system hydraulic equations at each time step of a simulation. It is useful mainly for debugging purposes.

If many hydraulic analyses will be run in the application it is recommended that status reporting be turned off (*`statuslevel`* = 0).

ENgeterror

```
int ENgeterror( int errcode, char* errmsg, int nchar )
```

Description:

Retrieves the text of the message associated with a particular error or warning code.

Arguments:

errcode:	error or warning code
errmsg:	text of the error or warning message for *errcode*
nchar:	maximum number of characters that *errmsg* can hold

Returns: Returns an error code.

Notes: Error message strings should be at least 80 characters in length.

Toolkit Functions by Task