EPANET Results Database (ERD)

User’s Guide

Version 1.00.00

|  |  |  |
| --- | --- | --- |
| Chris Geib | Tom Taxon | Sam Hatchett |
| Christopher.Geib@uc.edu | tntaxon@anl.gov | samhatchett@gmail.com |

10/12/11

Contents

1 About 3

2 Files and Formats 3

2.1 Prologue/Header 3

2.2 Index 5

2.3 Hydraulics Results Data 5

2.4 Water Quality Results Data 6

3 Library Distribution Package 6

4 ERD Library Functions 7

4.1 ERD Database Related Functions 7

4.1.1 ERD\_create 7

4.1.2 ERD\_open 7

4.1.3 ERD\_isERD 8

4.1.4 ERD\_close 8

4.1.5 ERD\_setHydStorage 8

4.1.6 ERD\_GetCompressionDesc 8

4.1.7 ERD\_getCompressionLevel 9

4.2 ERD Data Retrieval Functions 9

4.2.1 ERD\_getNetworkData 9

4.2.2 ERD\_getResults 9

4.2.3 ERD\_getRawSimulationResults 9

4.2.4 ERD\_getHydSimCount 9

4.2.5 ERD\_getQualSimCount 10

4.2.6 ERD\_getQualSimCountFor 10

4.2.7 ERD\_getApplicationData 10

4.2.8 ERD\_getERDcontrolLinkIndex 10

4.2.9 ERD\_getSpeciesIndex 10

4.3 ERD Data Storage Functions 10

4.3.1 ERD\_newHydFile 10

4.3.2 ERD\_newQualFile 10

4.3.3 ERD\_writeHydResults 11

4.3.4 ERD\_writeQualResults 11

4.3.5 ERD\_clearQualityData 11

4.4 ERD Error Handling Functions 11

4.4.1 ERD\_getError 11

4.4.2 ERD\_getErrorMessage 11

4.5 ERD Utility Functions 11

4.5.1 ERD\_UTIL\_positionFile 12

4.5.2 ERD\_UTIL\_getFilePosition 12

4.5.3 ERD\_UTIL\_initQual 12

4.5.4 ERD\_UTIL\_initHyd 12

4.6 newDPXIndexData 12

4.7 newTEVAIndexData 12

5 ENL Library Functions 12

5.1 ENL\_getNetworkData 12

5.2 ENL\_getHydResults 13

5.3 ENL\_getQualResults 13

5.4 ENL\_createDemandProfiles 13

5.5 ENL\_saveInitQual 13

5.6 ENL\_restoreInitQual 13

5.7 ENL\_releaseInitQual 14

5.8 ENL\_setSource 14

5.9 ENL\_writeTSI 14

5.10 epanetmsxError 14

5.11 epanetError 14

5.12 ENL\_getQualResults 14

6 Library Data Structures 14

6.1 ERD 15

6.2 NetInfo 15

6.3 SpeciesData 15

6.4 NodeInfo 16

6.5 LinkInfo 16

6.6 HydData 17

6.7 QualData 17

6.8 Source 17

6.9 SourceData 17

7 Error Codes 18

8 Acknowledgements 18

List of Tables

Table 1: Prologue File Header 3

Table 2: Prologue File 4

Table 3: Index File 5

Table 4: Hydraulics Results File Entry 6

Table 5: Water Quality Results File Entry 6

Table 6: Valid values for the flag parameter to the ERD\_open function 8

# About

The EPANET Results Database is a method of storing multiple EPANET hydraulic and water quality simulation results for one network. It has the following key features.

• Water quality results data are grouped into Hydraulics Groups - results with common hydraulics data.

• Elimination of code allowing backward-compatibility older versions of TSO.

• Allows storage of muti-species simulations.

• Database and API meet the requirements of EPANET-DPX.

This document describes a version of the database that will typically be used to store results from EPANET-DPX applications. In its present form, the database can store data for one network. The ERD application programming interface (API) allows data from EPANET-DPX simulations to be retrieved for analysis.

# Files and Formats

This database consists of four different types of files in a single directory – the header, index, hydraulics and water quality. Each file ends with the extension .erd. The remainder of the file name is discussed in each of the sections below

## Prologue/Header

(basename)/output.erd

The Header holds data about the database and the network used to produce it. The first 32 bytes of the Prologue file are the database header. They contain flags and version information. Following the header are network data, like coordinates, reporting times, and node and link counts. Contents are listed in the following table.

|  |  |  |
| --- | --- | --- |
| byte # | quantity | value(s) |
| 0 | header flag | 0xff (255) |
| 1 | file version | 12 |
| 2 | compression level | enum CompressionLevel  3 – RLE  4 – LZMA |
| 3 | bytes for node/link/species ID | 64 |
| 4 | EPANET application flag | enum OutputFrom  0 – TEVA  1 – EpanetDPX  2 – Undefined |
| 5 | hydraulics storage type | binary 00000 – 01111  00000 = no hydraulic data stored  00001 = node demands  00010 = link flows  00100 = link velocities  01000 = node pressure  10000 = demand profiles |
| 6 - 31 | currently unused |  |

Table 1: Prologue File Header

|  |  |
| --- | --- |
| **quantity** | **type** |
| header | char array (32 bytes) |
| Hydraulic Simulation Count | int |
| Quality Simulation Count | int |
| node count | int |
| link count | int |
| tank and reservoir count | int |
| junction count | int |
| species count | int |
| number of reporting steps | int |
| size of report time step (seconds) | float |
| Value that represent the FLT\_MAX | float |
| EPANET quality code | int  0 - ???  1 - ???  2 - ???  3 - ???  4 – Multi-species |
| simulation duration (seconds) | int |
| report start time (seconds) | int |
| report time step x 3600 (seconds) | int |
| Simulation start time | int ??? |
| Number of control links | int |
| Control link indices | int array (Number of control links) |
| tank node indices | int array (tank and reservoir count) |
| node IDs | char array (node ID length, node count) |
| link IDs | char array (link ID length, link count) |
| For each specie: |  |
| species index | int |
| species type | enum SpecieTypes  0 – bulk  1 - wall |
| species ID | char \* (species ID length) |
| node X coordinates | float array (node count) |
| node Y coordinates | float array (node count) |
| link start node indices | int array (link count) |
| link end node indices | int array (link count) |
| link lengths | int array (link count) |
| for each link: |  |
| number of vertices | int |
| if number of vertices > 0 |  |
| vertices’ X coordinates | float array (number of vertices) |
| vertices’ Y coordinates | float array (number of vertices) |
| Number of hydraulic and quality files | int |
| For each file: |  |
| length of filename | int |
| Filename | char \* (length of filename+1) |

Table 2: Prologue File

## Index

(basename)/output.index.erd

The Index contains data that coordinate the results files. It holds results ID numbers, results data file indices, hydraulics group indices, and results data offsets and lengths.

|  |  |
| --- | --- |
| Quantity | type |
| for each set of quality results: |  |
| index type | unsigned char |
| water quality file index | int |
| hydraulic sim index | int |
| bytes data length | int |
| file offset | int |
| Application specific data |  |
| TEVA |  |
| Number of sources | int |
| For each source |  |
| Source node index | int |
| Specie index | int |
| Source type | int  Types??? |
| Source strength | float |
| Source start (units???) | int |
| Source stop (units???) | int |
| EPANET-DPX |  |
| INP filename length | int |
| EPANET inp file | char \* (length) |
| MSX filename length | int |
| MSX filename | char \* (length) |

Table 3: Index File

## Hydraulics Results Data

(basename)/output-(group-number).hyd.erd

The Hydraulics Results files contain node demands and link flows and velocities from the simulation that produced the database. One file can be created for each unique set of hydraulic data from the network.

The inclusion of the individual components (demands, pressures, etc) are controlled by the flags passed to the ERD\_setHydStorage function. Additionally, if the comression method is lzma, then this data is also compressed.

|  |  |
| --- | --- |
| Quantity | type |
| Link status | int array (number of reporting steps, number of control links) |
| Node demands | float array (number of reporting steps, number of nodes) |
| Node pressures | float array (number of reporting steps, number of nodes) |
| Link flows | float array (number of reporting steps, number of nodes) |
| Link velocities | float array (number of reporting steps, number of nodes) |
| Demand patterns |  |
| for each node: |  |
| demand pattern length | int |
| demand pattern | float array (node count) |

Table 4: Hydraulics Results File Entry

## Water Quality Results Data

\*.qual.erd

The Water Quality Results files contain node concentration data. Data are indexed by species, node, then reporting time step. The data are compressed to one record for each node with non-zero value, and then either run-length encoded or compressed using LZMA compression.

|  |  |
| --- | --- |
| Quantity | type |
| Result index | int |
| Water quality | float (number of species, number of nodes, number of reporting times) |

Table 5: Water Quality Results File Entry

# Library Distribution Package

REVIEW THIS!!!

The EPANET Results Database Library is distributed in a compressed archive file. Its contents are listed here.

**Readme.txt** Description of contents in the archive

**License.txt** License agreement for ERD library

**lib/** Compiled library files for the API

**liberd.so** ERD Linux shared-object library

**liberd.dylib** ERD Darwin dynamic library

**erd.dll** ERD Windows dynamic link library

**libenl.so** ENL Linux shared-object library

**libenl.dylib** ENL Darwin dynamic library

**enl.dll** ENL Windows dynamic link library

**doc/** Library documentation

**ERD-Guide.pdf** This file

**erd/include/** C header files

**erd.h** ERD Library header file, containing all public function prototypes and data structures

**erd/src/** C source files

**erd.c** All ERD public library functions

**erdinternal.c** Internal functions used by library functions in erd.c

**erdinternal.h** Header file for erdinternal.c

**teva.c** Functions specific to ERD’s TEVA extension

**teva.h** Header file for teva.c

**dpx.c** Functions specific to ERD’s EPANET-DPX extension

**dpx.h** Header file for dpx.h

**erd/make/** GNU Make files

**linux** GNU/Linux makefiles and script templates

**darwin** Mac OS X makefiles and script templates

**cygwin** Cygwin makefiles and script templates

**enl/include/** C header files

**enl.h** ENL Library header file, containing all public function prototypes

**enl/src/** C source files

**enl.h** All ENL public library functions

**enl/make/** GNU Make files

**linux** GNU/Linux makefiles and script templates

**darwin** Mac OS X makefiles and script templates

**cygwin** Cygwin makefiles and script templates

# ERD Library Functions

The ERD Library Functions interact directly with the database files. The main purpose of the ERD functions is reading results, but there are write functions, that should be used in conjunction with the ENL functions (described in the next section).

## ERD Database Related Functions

### ERD\_create

int ERD\_create(PERD \*database, const char \*erdName, enum OutputFrom application, enum CompressionLevel compLevel);

Creates a new database named erdName, allocating data and initializing internal data structures. If the erdName also contains a path, the database will be stored in that path. If the location does not exist or is not writeable, an error code is returned.

The following would create a database, using the EPANET-DPX extensions, in a subdirectory named "epanet-output"; all database files having the prefix "example".

PERD d;

if(ERD\_create(d, "~/epanet-output/example", epanetdpx, lzma))

fprintf(stderr, "Database could not be created.\n");

### ERD\_open

int ERD\_open(PERD database, char \*erdName, int flags);

Opens an existing database, specified in erdName, for reading. If a database is not found in the directory, it returns an error code. The flags parameter controls what information your application needs from the database. This is available to reduce the memory footprint of the application if the various data aren’t needed by the application. The available flags are listed in Table 6. If data is requested (via the flags parameter), but wasn’t stored an error message is displayed and an error code is returned.

|  |  |  |
| --- | --- | --- |
| Flag | Value | Description |
| READ\_QUALITY | 0x00000001 | Load quality data |
| READ\_DEMANDS | 0x00000002 | Load demand data |
| READ\_LINKFLOW | 0x00000004 | Load link flow data |
| READ\_LINKVEL | 0x00000008 | Load link velocity data |
| READ\_PRESSURE | 0x00000010 | Load node pressure data |
| READ\_DEMANDPROFILES | 0x00000020 | Load demand profile data |
| READ\_ALL | 0x0000003F | Load everything |

Table 6: Valid values for the flag parameter to the ERD\_open function

The following would open the database created in above section and only load the demand profiles and water quality.

PERD d;

if(ERD\_open(d, "~/epanet-output/example",READ\_QUALITY | READ\_DEMANDPROFILES))

fprintf(stderr, "Database could not be opened.\n");

### ERD\_isERD

int ERD\_isERD(const char \*erdDBName);

Determines if the erdDBName is actually an ERD database. This function aids in allowing applications to be easily backwards compatible with TSO files.

PERD d;

if(ERD\_isERD("~/epanet-output/example"))

fprintf(stderr, "Database is an ERD database.\n");

### ERD\_close

int ERD\_close(PERD database);

Closes an open database and deallocate memory used by the ERD API in managing the database.

The following would close a database created or opened by a function in one of the previous sections.

if(ERD\_close(d))

fprintf(stderr, "Database could not be closed. Was it open?\n");

### ERD\_setHydStorage

int ERD\_setHydStorage(PERD db, int velocity, int flow, int demand, int pressure, int profile);

Sets what hydraulic data from a simulation will be stored. If this function is not called, all are stored.

### ERD\_GetCompressionDesc

LIBEXPORT(char \*) ERD\_GetCompressionDesc(PERD db);

Returns the description of the database’s compression method.

### ERD\_getCompressionLevel

LIBEXPORT(int) ERD\_getCompressionLevel(PERD db);

Returns the database’s compression level.

## ERD Data Retrieval Functions

### ERD\_getNetworkData

PNetInfo ERD\_getNetworkData(PERD database);

Returns the network data stored in a database.

### ERD\_getResults

int ERD\_getResults(PQualitySim qualSim, PERD database);

Populates the qualResults (type PQualData) and hydResults (type PHydData) elements of the NetInfo structure for the specified water quality simulation.

int count, resultIndex;

count = database->qualSimCount;

for(resultIndex = 0; resultIndex <= count; resultIndex++) {

ERD\_getResults(resultIndex, database);

PNetInfo net = database->network;

PHydData hyd = net->hydResults;

PQualData qual = net->qualResults;

/\* process results \*/

/\* qual->nodeC[specie][node\_idx][time\_idx] \*/

/\* qual->linkC[specie][link\_idx][time\_idx] \*/

/\* hyd->flow[time\_idx][link\_idx] \*/

/\* hyd->velocity[time\_idx][link\_idx] \*/

/\* hyd->demand[time\_idx][link\_idx] \*/

/\* hyd->pressure[time\_idx][link\_idx] \*/

/\* hyd->linkStatus[time\_idx][link\_idx] \*/

}

### ERD\_getRawSimulationResults

int ERD\_getRawSimulationResults(int id, PERD database, PSource sources, char \*buffer, int length);

Retrieves raw, compressed results data.

### ERD\_getHydSimCount

int ERD\_getHydSimCount(PERD database);

Returns the number of hydraulic simulations stored in the database.

### ERD\_getQualSimCount

LIBEXPORT(int) ERD\_getQualSimCount(PERD database);

Returns the number of quality simulations stored in the database.

### ERD\_getQualSimCountFor

int ERD\_getQualSimCountFor(int hydSimIndex, PERD database);

Returns the number of water quality results associated with a hydraulics group.

### ERD\_getApplicationData

LIBEXPORT(void\*) ERD\_getApplicationData(PERD database, int index);

Returns the application-specific data stored in the index that identifies the source of the water quality results.

### ERD\_getERDcontrolLinkIndex

LIBEXPORT(int) ERD\_getERDcontrolLinkIndex(PERD database, int epanetIndex);

Returns the internal ERD controlLink index for given epanet index. If it is not found, it returns -1

### ERD\_getSpeciesIndex

LIBEXPORT(int) ERD\_getSpeciesIndex(PERD database, const char \*speciesName);

Returns the internal index of the specified specie.

## ERD Data Storage Functions

### ERD\_newHydFile

int ERD\_newHydFile(PERD database);

Creates a new hydraulics group, and new hydraulics results file, for output.

### ERD\_newQualFile

int ERD\_newQualFile(PERD database);

Creates a new water quality output file for output. Analogous to a server group in EPANET-DPX and TEVA.

### ERD\_writeHydResults

int LIBEXPORT ERD\_writeHydResults(PERD database);

Writes hydraulics results from a simulation. This should be called after the hydraulics results stored in PERD->network->hydResults are updated. See the ENL library functions ENL\_getNetworkData and ENL\_getHydResults.

### ERD\_writeQualResults

int LIBEXPORT ERD\_writeQualResults(PERD \*database, void \*appdata);

Writes water quality results from a simulation. This should be called after the water quality results stored in PERD->network->qualResults are updated (see ENL\_getQualResults in the ENL Library Functions section below). The second parameter is the index data structure appropriate to the application that is generating the results. For TEVA, it should be a fully populated TEVAIndexData structure. For EPANET-DPX, it should be a fully-populated DPXIndexData structure.

### ERD\_clearQualityData

LIBEXPORT(int) ERD\_clearQualityData(PERD database);

Clears the water quality data stored in PERD->network->qualResults to prepare the data structure for the next set of simulation results.

## ERD Error Handling Functions

### ERD\_getError

LIBEXPORT(void) ERD\_Error(int errorCode);

Prints the ERD error message string and exits the program.

### ERD\_getErrorMessage

char \*ERD\_getErrorMessage(int errorCode);

Returns the character string associated with an error code number.

PQualitySim qsr = erd->qualSim[id];

Int errorCode = ERD\_getResults(qsr, erd);

if(errorCode) {

fprintf(stderr, "Error %i returned from ERD\_getResults: %s\n",errorCode, ERD\_getErrorMessage(errorCode));

}

## ERD Utility Functions

This section describes the utility functions defined. These functions are primarily exposed for the use by other APIs and libraries, most notably by the ENL library. In general a user of the ERD API will never have to use these.

### ERD\_UTIL\_positionFile

LIBEXPORT(int) ERD\_UTIL\_positionFile(FILE \*fp,\_\_file\_pos\_t offs);

Utility function to properly position file in a cross-platofrm manner

### ERD\_UTIL\_getFilePosition

LIBEXPORT(\_\_file\_pos\_t) ERD\_UTIL\_getFilePosition(FILE \*fp);

Utility function to properly retrieve the file position in a cross-platofrm manner

### ERD\_UTIL\_initQual

LIBEXPORT(PQualData) ERD\_UTIL\_initQual(PNetInfo network, int flags);

Utility function to initialize the quality data structure.

### ERD\_UTIL\_initHyd

LIBEXPORT(PHydData) ERD\_UTIL\_initHyd(PNetInfo network, int flags);

Utility function to initialize the hydraulic data structure.

## newDPXIndexData

LIBEXPORT(PDPXIndexData) newDPXIndexData(char \*inputFilename, char \*msxInputFilename);

Create and populate a new index data structure for EPANET-DPX.

## newTEVAIndexData

LIBEXPORT(PTEVAIndexData) newTEVAIndexData(int numSources, PSource source);

Create and populate a new index data structure for TEVA.

# ENL Library Functions

The ENL (EpaNet Link) Library provides a connection between the EPANET / EPANET-MSX toolkits and the ERD library for writing simulation results. It is built as a separate library so applications written to read results need only to link to the ERD library.

## ENL\_getNetworkData

LIBEXPORT(int) ENL\_getNetworkData(PERD database, const char \*inputFile, const char \*msxInputFile,char \*msx\_species);

## ENL\_getHydResults

LIBEXPORT(int) ENL\_getHydResults(int time, long timeStep, PERD database);

Read the current hydraulic state from EPANET and update the PHydData structure using a weighting based on the reporting interval. The timeStep parameter is the amount of time since the last hydraulic event. This method uses HydVal \* (timeStep/database->network->reportStep) to compute the incremental addition to the hydraulic results for time time.

## ENL\_getQualResults

LIBEXPORT(int) ENL\_getQualResults(int time, long timeStep, PERD database);

Read the current water quality for each specie from EPANET and update the PQualData structure using a weighting based on the reporting interval. The timeStep parameter is the amount of time since the last water quality reading. This method uses QualityVal \* (timeStep/database->network->reportStep) to compute the incremental addition to the hydraulic results for time time. For multi-species runs, this will put the bulk specie data in the nodeC element and wall specie data in the linkC element in the PQualData structure.

## ENL\_createDemandProfiles

LIBEXPORT(void) ENL\_createDemandPatterns(PERD db);

Create the demand patterns for each node in the network. The result is one demand profile for each junction in the network model. Each profile is the application of all demands & demand patterns defined for each junction with a length is the least common multiple of all componenet demand patterns. These demand profiles are created so that there is a consistent set of demands to use for purposes such as estimating demand.

## ENL\_saveInitQual

LIBEXPORT(int) ENL\_saveInitQual(PNetInfo net, double \*\*\*initQual);

This function saves the initial quality values from an MSX-based simulation so they can be restored for the next attack scenario.

## ENL\_restoreInitQual

LIBEXPORT(int) ENL\_restoreInitQual(PNetInfo net, double \*\*initQual);

Restore previously saved initial quality values to initialize a new attack scenario in an MSX-based simulation.

## ENL\_releaseInitQual

LIBEXPORT(int) ENL\_releaseInitQual(PNetInfo net, double \*\*\*initQual);

Release the menory allocated to store the initial quality data from an MSX-based simualtion.

## ENL\_setSource

LIBEXPORT(int) ENL\_setSource(PSourceData source, PNetInfo net, FILE \*simin,int isMSX);

This function reads a line from the simin file and populates the source array.

## ENL\_writeTSI

LIBEXPORT(void) ENL\_writeTSI(PNetInfo net, PNodeInfo nodes, PSource sources, FILE \*simgen, FILE \*simin);

This function reads the simgen file contents and expands the sources using the specified network information and writes a TSI file to simin.

## epanetmsxError

LIBEXPORT(int) epanetmsxError(int errorCode);

This function outputs the appropriate error message based on the errorCode and exits. This function handles both EPANET and EPANET-MSX errors and should only be used with MSX-based simulations.

## epanetError

LIBEXPORT(int) epanetError(int errorCode);

This function outputs the appropriate error message based on the errorCode and exits. This function handles only EPANET and should only be used with EPANET-based simulations.

## ENL\_getQualResults

int ENL\_getQualResults(int time, long timeStep, PERD database);

Retrieves water quality data (node concentrations) from a simulation at a given time step. Automatically detects multi-species data. Data are stored in the ERD data structure’s NodeInfo field for retrieval by ERD\_writeQualResults.

# Library Data Structures

The following sections contain the portions of the data structures that might be useful while using this API. Consult the source code for an exhaustive list of the data structure fields.

## ERD

ERD is the main data structure for the database. Access to this structure is provided by the ERD\_ functions, so minimal knowledge of this structure is required. All of the data structures are initialized and released through the library’s functions. Below are only the fields that might be useful to access.

typedef struct

{

PNetInfo network; /\* Network data - only one network \*/

PNodeInfo nodes; /\* Network node data \*/

PLinkInfo links; /\* Network link data \*/

} ERD, \*PERD;

## NetInfo

NetInfo contains data about the network stored in the database.

typedef struct

{

int simDuration; /\* Simulation duration \*/

long simStart; /\* Start time of the simulation \*/

int reportStart; /\* Report start time \*/

int reportStep; /\* Report time step \*/

int numNodes; /\* Number of nodes \*/

int numLinks; /\* Number of links \*/

int numTanks; /\* Number of tanks \*/

int numJunctions; /\* Number of junctions \*/

int numSpecies; /\* Number of species \*/

PSpeciesData \*species; /\* Species data \*/

int numSteps; /\* Number of time steps \*/

int qualCode; /\* EPANET/MSX quality code \*/

float stepSize; /\* Size of reporting step \*/

float fltMax; /\* Maximum value of float \*/

int numControlLinks; /\* Number of controlled links \*/

int \*controlLinks; /\* Vector of controlled erd link indices -- base 0 \*/

PHydData hydResults; /\* Hydraulics results \*/

PQualData qualResults; /\* Water quality results \*/

} NetInfo, \*PNetInfo;

## SpeciesData

There is one SpeciesData data structure in the NetInfo data structure for each species species stored. This structure just holds the name and index of each species. If the simulation was not EPANET-MSX, there is only one species.

typedef struct

{

char id[MAX\_ID\_LENGTH]; /\* Species name \*/

int index; /\* Species index; -1 is species is not stored \*/

enum SpecieTypes type; /\* Specie type - 'bulk' or 'wall' \*/

} SpeciesData, \*PSpeciesData;

## NodeInfo

There are multiple NodeInfo data structures in a database. When reading from a database, results can be stored in an array of NodeInfo data structures for processing.

typedef struct

{

enum NodeTypes type; /\* Junction or Tank \*/

float x; /\* X position \*/

float y; /\* Y position \*/

char id[MAX\_ID\_LENGTH + 1]; /\* Node ID \*/

/\* Demand profile represents the repeating demands that are specified in the EPANET input file. The length is calculated by the LCM of all demand patterns for the node. The actual values are the sum (at each time step) of all the base demands multiplied by the demand multiplier. This was added to ensure consistent average demands regardless of the simulation length \*/

float \*demandProfile; /\* Demand profile \*/

int demandProfileLength; /\* Length of demand profile \*/

int \*nz; /\* Specie indices that contain non-zero data \*/

} NodeInfo, \*PNodeInfo;

## LinkInfo

There are multiple LinkInfo data structures in a database. When reading from a database, results can be stored in an array of LinkInfo data structures for processing.

typedef struct

{

int from; /\* Link from node \*/

int to; /\* Link to node \*/

float length; /\* Link length \*/

char id[MAX\_ID\_LENGTH]; /\* Link ID \*/

int nv; /\* Number of vertices \*/

float \*vx; /\* X coordinates of vertices \*/

float \*vy; /\* Y coordinates of vertices \*/

int \*nz; /\* Specie indices that contain non-zero data \*/

} LinkInfo, \*PLinkInfo;

## HydData

HydData structures contain link velocity, link flow, and node demand data. They are indexed by time and then node or link.

typedef struct

{

float \*\*flow; /\* Link flows - time step, link \*/

float \*\*velocity; /\* Link velocities - time step, link \*/

float \*\*demand; /\* Node demands - time step, node \*/

float \*\*pressure; /\* Node head - time, node \*/

int \*\*linkStatus; /\* Link Status - time, link (link indexed by NetInfo controlLink index -- base 0) \*/

} HydData, \*PHydData;

## QualData

QualData structures contain node concentration data. They are indexed by species, time, and then node.

typedef struct

{

float \*\*\*nodeC; /\* Node concentrations - species, node, time \*/

float \*\*\*linkC; /\* Link concentrations - species, link, time \*/

} QualData, \*PQualData;

## Source

Source structures contain source data from TEVA simulations.

typedef struct

{

char sourceNodeID[MAX\_ID\_LENGTH]; /\* Node ID \*/

int sourceNodeIndex; /\* Node index \*/\* Source type \*/

int speciesIndex; /\* Species index \*/

long sourceStart; /\* Source start time \*/

long sourceStop; /\* Source stop time \*/

float sourceStrength; /\* Source strength \*/

} Source, \*PSource;

## SourceData

SourceData encapsulates an array of Source elements

typedef struct

{

int nsources; /\* Number of sources \*/

PSource source; /\* Source data \*/

} SourceData, \*PSourceData;

# Error Codes

Each ERD function returns an integer value that corresponds to one of the following error codes. The function ERD\_getErrorMessage can be used to change the integer value to a character array.

|  |  |
| --- | --- |
| Integer Value | String Value |
| 701 | Database not found in directory |
| 702 | Incorrect database format |
| 703 | Bad directory name |
| 704 | Database already closed |
| 705 | Results not found for ID |
| 706 | Temp file error |
| 707 | No results found in database |
| 711 | Error opening index file |
| 712 | Error opening prologue file |
| 713 | Error opening hydraulics results file |
| 714 | Error opening water quality results file |
| 721 | Error closing index file |
| 722 | Error closing prologue file |
| 723 | Error closing hydraulics results file |
| 724 | Error closing water quality results file |
| 731 | Error reading qual index data |
| 732 | Error reading prologue data |
| 733 | Error reading hydraulics data |
| 734 | Error reading water quality data |
| 741 | Error writing qual index data |
| 742 | Error writing prologue data |
| 743 | Error writing hydraulics data |
| 744 | Error writing water quality data |
| 745 | Invalid application type |
| 746 | Invalid compression method |
| 747 | Data requested that is not stored |
| 748 | No hydraulic data stored |

Table 5: Error Codes

# Acknowledgements

This software was developed with funding from the National Homeland Security Research Center of the U.S. Environmental Protection Agency, Cincinnati, OH. Robert Janke was the work assignment manager, and the authors wish to thank him as well as Regan Murray for their input and encouragement for developing general modeling tools that can help the water industry assess and mitigate risks to water quality.