The Promex Heat Exchanger Design Toolkit

Version 0.12

Jack Devanney

Sisyphus Beach

Tavernier, Florida

2012

2012-12-02
This is an incomplete draft for discussion.
Please send any comments to djw1@c4tx.org.

Be aware that Versions 0.10 and earlier had a bug in the optimization logic which could lead to badly misleading results. Version 0.10 users should upgrade to 0.12 ASAP.

Contents

1 Introduction

Promex is a shell and tube heat exchanger (HX) design toolkit. Promex's focus is on HX's employing molten salts as the heat transfer fluids. It was inspired by the ORNL program Primex and uses the core Primex logic.[1] However, while Primex is a fiendishly clever program, the coding violates every rule of modern programming. Primex is written in 1960's style Fortran, with short, always cryptic and often meaningless variable names. Except for the tube stress analysis, there are no subroutines, just a single 500 line main program, with almost no comments. All variables are global. The loops are implemented with intertwining GOTO's. One result is that Primex had some serious bugs which apparently went unnoticed until this rewrite.[2] Finally, the whole thing is in cursed English units.

Our goal was to totally restructure the code to make it far more flexible, and far more easily maintained. We also took advantage of this re-write to convert the program to strict SI units.

However, the result is a toolkit, not a single program. To make proper use of Promex, the user must be prepared to do some scripting. The goal is to be engineer friendly, where an engineer is presumed to have some programming capability. Promex is written in Perl, and assumes some knowledge of Perl. But anyone who has done any scripting in any language can quickly learn the little Perl he needs.

Promex is released under the BSD License which reads

[1] Bettis, C. et al, Computer Programs for MSBR Heat Exchangers, ORNL-TM-2815, June, 1971. Promex comes after Primex in the vowelphabet.

[2] Devanney, J., Resurrecting Primex. http://www.c4tx.org/ctx/pub/primex_bugs.pdf.

Caveat receptor. In particular, this is an early-alpha version which undoubtedly has serious bugs. It has been very lightly tested. It is being released in the hope that people who have a need for an Open Source heat exchanger program will correct and extend the package. Or to put it more correctly, Promex is being released now so the real testing can begin.

To our knowledge, this version of Promex has only been tested on the Linux distribution Ubuntu. CTX does not support proprietary operating systems. But Promex should run on any platform that has a reasonably up-to-date Perl interpreter.

Throught this manual we use the convention that text in fixed-width, upright font such as `promex.pl` is to be entered verbatim. Material in fixed-width italic such as *`filename`* is variable and under the control of the user.

2 Logic

At its core, Promex uses the same overall logic as Primex. This logic requires that the user specify a required heat transfer rate (duty), and the hot end and cold end temperatures on both the tubes-side and the the shell side. This immediately determines the mass flow rates on both sides by a simple energy balance. Unless the specified temperatures and required duty are reasonably consistent, you will almost certainly get a "No feasible solution. See logile `promex_logfile_`*`nnnnn`*" message. Promex will transfer heat from the tube side fluid to the shell side or vice versa depending on whether the tube side hot end temperature is higher or lower than the shell side hot end.

Once all the heat exchanger parameters other than length are established, Promex starts at the hot end of the HX and keeps adding length increments until it reaches the user specified cold end temperatures or decides that the combination is infeasible. Each length increment must be short enough so that Promex/Primex assumption that the material properties at the average temperature in the length increment can be used throughout the increment is acceptably accurate. It is the user's responsibility to look at the results and decide if this is the case.[3]

Where the Promex logic completely differs from Primex is in the outer loops. In a Primex run everything other that HX diameter baffle spacing and HX length is fixed by the user. Primex searched over HX diameters and baffle spacing looking for a feasible HX that had pressure drops that were close to but not more than user set targets. Primex used a baroque algorithm to try and do this.

Promex allows the user to search over the entire HX design space with as many search loops as the user desires and his hardware will accommodate. Promex requires that the user supply a cost model which can be as simple as a linear function of HX weight and pressure drops. Promex finds the minimum cost HX that meets the heat load and the user specified hot end and cold end temperatures, according to that cost model.

Usually the goal of Promex's optimization is not *optimum* in some absolute sense, but reasonable, or, as the economists would say, efficient. For

[3] A rule of thumb for molten salts is that as long as the temperature drop in an increment is less than 10C, you are probably OK. Early in the design process, you may want to bend this rule to enable a wide-ranging search over the design space. Later as the design process focuses in on a particular sub-region in the design space, you should probably go to smaller length increments to get a bit more accuracy, and more importantly see if the choice of length increment significantly affects your results. Promex is designed to this facilitate this kind of changing of the rules.

example, by varying the ratio of material cost to pump power cost in the simple cost model outlined above, we can pull out the set of efficient HX's, those for which we can't have a lower material cost without a higher pump cost. This subset of sub-optimum alternatives then can be passed to a more comprehensive plant design program which can figure out what a change in HX diameter or length implies for the plant as a whole.

In some cases, minimizing a weighted combination of material and power will result in a very long/tall heat exchanger which will have its own costs in terms of plant arrangment. By adding a length penalty, a form of Lagrange multiplier, Promex can be induced to come up with a shorter heat exchanger if that is feasible.

Promex does allow the user to specify maximum allowable pressure drops on either or both of the shellside and the tube side. This capability simply throws out candidates who fail to meet the required pressure drop(s). We recommend that this feature be used judiciously if at all. Usually you will get a more intelligent design if you increase the ratio of pump power cost to material cost until you get down to the pressure drop you think you need.

3 Limitations

This version of Promex has at least the following major limitations.

Only Countercurrent Heat Exchangers It is not clear — at least to me — to what degree it is possible to extend the core Primex logic to non-countercurrent heat exchangers.[4]

Single phase, incompressible, near-constant specific heat The current version only works on incompressible fluids in which the specific heat does not change drastically over the HX temperature range. Primex assumed the enthalpy change as the fluids move through the heat exchanger is a function only of the inlet and outlet temperature, not pressure.

This limitation rules out super-critical and two phase fluids. We have plans to relax this via an outer loop which will guess the enthalpy change for such a fluid which will allow us to do the Primex calculations, after which the actual enthalpy change will be compared with the guess, and the guess adjusted. Repeat until guess and actual converge.

Models only the middle section Most shell and tube heat exchangers can be divided into three sections: a section at either end where the fluid inlets and outlets are, and a big section in the middle. The current version of Promex does not model the section at either end, implicitly assuming that all the heat transfer and pressure drops take place in the middle section. Nor does it model the specifics of the U-bend in a U-tube heat exchanger. These restrictions are largely unnecessary. Future versions will need to rectify this. The basic framework has been set up to accommodate a special hot end section and to a lesser degree a cold end section.

Only cylindrical and segment-of-a-ring geometries Currently Promex implements only two geometries;

1. a cylindrical shell and tube bundle with or without a central downcomer,
2. a segment of a ring. This geomentry is popular among reactor designs which put a heat exchanger just above the core.

[4] Practically, this is not as bad a restriction as it sounds. The delta T's a molten salt reactor has to work with are such that anything other than a nearly counter-current HX is quite unlikely to be economically infeasible.

The cylindrical geometry can be used to model a U-tube, but in this case either the heat transfer/pressure drop in the U must either be ignored or the U assumed to transfer heat/momentum in the same manner as the rest of the heat exchanger. This restriction is largely unnecessary and other geometries are certainly possible.

No stress analysis. The current version does no thermal expansion nor stress analysis. All material thicknesses must be provided by the user, and there are no strength checks. This restriction in largely unnecessary. Hooks have been provided to allow this limitation to be relaxed in the future.

Brute force search In all the sample problems in this version, the main program does an exhaustive search over the design space. This can be very computer intensive. If the user is willing to make some assumptions, then massive improvements in search efficiency are possible. If the assumption is made that there is a single global optimum, then hill climbing techniques can be used. Less restrictive assumptions such as assuming that, as soon as the cost starts going the wrong way in the innermost loop(s), there is no point searching further along this axis can cut the design space drastically.

No GUI Promex has only a command line interface. CTX regards this as more of a feature than a limitation. Inter alia, it allows Promex calculations to be incorporated into larger overall system design studies. The input and output is in XML to facilitate this. In any event, we have no plans to give Promex a GUI.

4 Installing Promex

To install Promex on Linux or a Linux-like Unix platform,

1. Make a folder where you want to install Promex. You might want to name this folder `promex`.
2. Point your browser at www.c4tx.org/ctx/job/promex/. Select the version you want from the index and download the file to this folder. The download file will be a tar archive with a name like `promex_`*`n.mm`*`.tgz` where *`n.mm`* is the Promex version number.
3. Issue
   `tar xvzf promex_`*`n.mm`*`.tar.gz`
   to expand the tarball. You will find the version you downloaded in a sub-folder called *`n.mm`*. This setup allows you to keep multiple versions of Promex around.
4. Set an environment variable called `PROMEX_PATH` to the full path to this folder including the *`n.mm`*. For example, if you expanded the tarball in `/usr/local/promex`, `PROMEX_PATH` must be set to `/usr/local/promex/`*`n.mm`*.
5. Set an environment variable called `PROMEX_VERSION` to the version number of the Promex you have installed.
6. Make sure all the Promex sub-folders are executable by the relevent user group(s) and writable by users whom you want to be able to edit the code. Include the Promex Design folders (see next section) in your users' path.
7. If your Perl interpreter is not at `/usr/bin/perl`, make a symbolic link from wherever it is to `/usr/bin/perl`.[5]
8. Go to the ornl4541 sub-folder in your Promex folder and issue `promex.pl`. You should end up seeing something like Table 1. Be patient. There's quite a bit going on and the run could take 20 seconds or more depending on your machine. You will also get some intermediate output. including one line every time Promex finds a feasible candidate HX that has a lower cost than the best so far.
9. If you have a problem on a Linux platform, send as email to `djw1@c4tx.org` describing the situation. We may be able to help, but no promises.

[5] If your site policy does not allow you to do this, then you will have to either

(a) edit the first line in each executable file changing `/usr/bin/perl` to the path to your Perl interpreter, or

(b) prefix each command in this manual with "`perl `". That is, instead of issuing "`promex.pl`", you will have to issue "`perl promex.pl`".

If you are not on a Linux-like platform, then you will have to find a local guru who knows how to modify these instructions for your platform.

5 Promex Structure and Philosphy

We can begin to understand the structure and philosophy of Promex by looking at the directory tree. When you untar the download file, you will find a number of folders in the directory where you have placed Promex. These folders fall into two categories:

1. *Common folders* containing programming or libraries which are independent of a specific HX geometry.
2. *Design folders* which are specific to a particular HX concept.

To help make this distinction clear, the Common folders have capitalized names; the Design folders have lower case names.

The Common folders are

MANUAL Contains this manual.

MAT A library of material property functions.

SHELL_PD A library of shell-side pressure drop routines.

SHELL_HT A library of shell-side heat transfer routines.

TUBES_PD A library of tube-side pressure drop routines.

TUBES_HT A library of tube-side heat transfer routines.

WALL_HT A library of tube wall heat transfer routines.

STRESS A library of tube stress routines.

COST This folder contains a library of cost models.

IN_OUT Input and output routines.

COMMON This important folder contains all the library and utility routines which do not fit into one of the above categories. In particular, it contains the core Primex-based logic. the core Primex logic.

The input to Promex dictates which of the various heat transfer and pressure drop routines the user wants to use in a particular run.

In this version, the geometry specific design programs that are included are:

ornl4541 This folder is based on the geometry of the ORNL4541 Primary Heat Exchanger.[6] One of its purposes is to facilitate a reasonably close comparison of Promex's results with Primex's for this heat exchanger.

twist0 This design concept assumes the same basic geometry of the ORNL4541 PHX but replaces the circular tubes and disk and donut baffles with baffle-less, elliptical, twisted tubes.

All design folders contain at least two Perl files:

promex_layout_hx.pl Given a candidate HX, this subroutine must calculate all the physical parameters of the HX such as number of tubes, cross-section flow areas, etc that do not depend on HX length, for the user's specific HX geometry.

promex.pl This usually very simple main program implements the search over the design space. It consists of nothing more than a set of loops over whatever HX parameters the users wants to regard as variable, repeatedly calling the various Common routines to evaluate each possible candidate.

Looking at this directory tree, the implications should be fairly obvious:

1. The user has the responsibility of choosing which material properties routines, which heat transfer and pressure drop correlations, and which stress and cost models will be used. If you do not find the material or correlation or model you want or need, you are expected to provide it. In doing so, you will have to follow the Promex AP which is described later in this manual. This is usually done by a copying an existing routine and changing it as required.
2. Unless your heat exchanger geometry just happens to match one of the geometries already implemented, the user is expected to provide his own. That is, he will have to create a new design folder and write the necessary code. In doing so, he can use one of the existing Design folders as a template. Often the changes required are close to trivial.

[6] Robertson, R, et al, Conceptual Design Study of a Singel-Fluid Moltes Salt Breeder Reactor, ORNL-4541, June, 1971.

6 Data Input Files

6.1 Introduction

To use Promex, you will need to prepare an input data file. Unless you are testing new Promex code, CTX strongly recommends that you do NOT do this from the Promex folder. Rather for each HX design study, set up a totally separate folder and run Promex from that folder. This will require the Promex design folders to be on your command path. Or you can set an environment variable say `PX` to the design folder you want to run and issue `$PX/promex.pl` with the proper command line variables. See next section.

6.2 Cylindrical Heat Exchangers

Figure 1 shows a typical input file for a cylindrical heat exchanger. Promex input is in XML format, and the XML rules must be followed or you will get an input parse error. The comments to the right of the colon, including the colon, in Figure 1 are ***not*** part of the real file. The file would be illegal XML if they were.

Each input variable is an attribute in the `<user>` element. Within this element, order and white space have no meaning. The order and grouping shown in Figures 1 and Figure 2 is for convenience in explanation. Promex will accept any order. Each variable is specified by a line of the form `name = "value"` ***with the value always in quotes***. And the attribute name must match the corresponding variable name within the code.[7]

Most of the attribute names in Figure 1 are almost self-explanatory, and the illegal text on the right should fill in the gaps. But a few additional comments are in order. The three lines at the top set output labeling and file names. They do not afffect the actual caclulations.

The next seven lines in this file select the heat transfer, pressure drop, etc models which the user wants to use in this run. For example, the line

```
shell_ht_func    = "ornl_2815"
```

instructs Promex to use the shell side heat transfer correlation which is in the file called `ornl_2815.pl` in the `SHELL_HT` folder.

In searching for the cold end temperature in each increment Primex used a temperature tolerance, hardwired at 3F. That is, if the trial temperature and the calculated temperature were within 3F, then Primex accepted that

[7] Internally, the input data is stored in a hash called $hx which is keyed by the attribute name. To be more precise, the attribute name must match the hash key for each variable.

temperature. Promex uses percent of the power transferred in the increment for this purpose. If the increment heat rate calculated from the enthalpy change in the fluids differs from the heat rate calculated from the heat transfer coefficients by less than this percent than Promex accepts the results and moves on to the next increment. The use can control this tolerance by setting `tol_W_pct`. The default is 0.1. At these levels, Promex's tolerance is much tighter than Primex. Typically, the temperature error is about 0.1C.

However, this accuracy is phony. Not only are the heat transfer correlations rarely accurate to +/-10 percent; but also, like Primex, Promex employs an extremely important short-cut. In computing the heat transfer in any length increment, Promex uses the difference in the bulk fluid temperature and the fluid temperature at the tube wall from the last increment. The assumption is:

a These two temperature differences are of secondary importance in determining the heat transfer coefficient.

b They are not going to change much from one increment to the next.

(a) is true for most heat transfer models but certainly not to the 0.1% level. (b) is true only after second or third increment.

`relax_factor` sets the relaxation factor for the search, that is how aggressively Promex changes the trial cold end temperature based on the current error. 1.0, the default, is fairly quick, but could lead to convergence errors. 0.5 will take about twice as long, but may converge in situations where a more aggressive relaxation factor will not.

`duty_W` is the required heat rate. `shell` should be read as shell-side. `tubes` should be read as tube-side. `tube` refers to an individual tube. To do a Promex run, the user must specify not only the required duty (heat load), but also the cold end temperatures, `tubes_cold_C` and `shell_cold_C`, and the hot end temperatures `tubes_hot_C` and `shell_hot_C`. Fortunately, these are the natural variables which arise in an overall molten salt plant design study. As soon as we have a candidate reactor and primary salt, we know the thermal power that the reactor is generating and we have a pretty good idea of the hot end temperature we want from materials considerations, and the low end temperature from melting point plus margin.[8]

[8] The required heat rate plus the hot and cold end temperatures determine the mass flow rates on both sides via the required enthalpy change in the fluids. In some cases, it may be more natural to specify one temperature and the mass flow rate which will then set the other temperature. Promex could easily be modified to work with input in this form. But so far there appears to be no pressing need for this capability.

When heat exchangers are cascaded as is often the case in molten salt reactors, the mass flows and the temperatures must match for both heat exchangers in the loop. If and

The variables which end with `fluid` or `mat` are material names which must match the name used to reference the material properties functions for that material in the `MAT` folder.. See `MAT/promex_mat_functions.pl` for a list.

The next set of variables set the unit costs used in the optimization.

The final set of variable deal with the physical particulars of the HX. Disk cut and donut cut are the fraction of the shell side cross-section area which the baffle leaves open, that is, the ratio of the area of the window outside/inside the baffle to the overall area.

All Promex input must be in strict, ***prefix-less*** SI. Linear dimensions must be in meters, not millimeters, not kilometers. Energy in joules, not kJ or MJ. Power in watts, not kW, or MW or GW. You can fake prefixes with exponents. For example, in the line setting `gap`, we used 9.525e-3 instead of 0.009525. 9.525e-3 meters can be read as 9.525 millimeters. the required heat rate, $duty\_W$ is set to 556.78e6 watts which can be easily read as 557 MW.

XML is self-identifying and allows flexible formatting. ***But far more importantly, XML allows you to add new variables to Promex without changing any code that does not depend on the new variable.***

For example, twisted tube HX's have parameters such as `tube_spiral_pitch` which are specific to this kind of HX. Often such parameters are required only in the heat transfer and pressure drop correlations. If so, all a designer who wants to add say `tube_spiral_pitch` to the list of input variables has to do is add a line in his input file of the form

```
tube_spiral_pitch = "200e-3"
```

and the new variable will be passed to all the Promex code as `$hx->{tube_spiral_pitch}`. "All" he has to do is write heat transfer and pressure drop correlations that respond to this variable, and select these correlations in his input file. ***He does not have to touch the rest of the code at all.***[9]

6.3 U-tube Heat exchangers

In this version of Promex, a U-tube HX is simply a cylindrical heat exchanger. Promex does not know that the cylinder has been bent into a U.

only if the specific heat is a very weak function of temperature, this will be automatic and you can work from one end of the cascade to the other.

[9] This is a slight over-statement. Often such new variables affect the HX layout and will also show up in `promex_layout_hx.pl` for the concept.

The specifics of the 180-degree bend are not modelled. The user has two choices:

Ignore the bend Treat the bend as if it were not there. No bend heat tranfer and no bend pressure drop. The result will be conservative for heat transfer but not for pressure drop. In this case, Promex's heat exchanger length is really twice the length of the straight legs of the U-tube. Adjustments for heat transfer and pressure drop in the bend will have to be made off-line.

Treat the bend like the legs. If the bend heat transfer and pressure drops are very similiar to that in the legs, then assume they are the same. In this case, Promex's heat exchanger length is really twice the average length of the U-tubes. This assumption ignores the additional pressure drop associated with the bend.

6.4 Segment-of-Ring Heat Exchangers

Figure 2 displays an input file for a ring segment, twisted tube HX. Only the new variables are commented. Segment-of-ring heat exchangers are described by the radius of the ring at the middle of the HX, `$hx->{ring_pr}`, the tube bundle arc length on this radius, `$hx->{bundle_arcl}` the bundle width, and the bundle tube length. In this case, the user wanted to keep the HX length/height as short as possible, so he set a very high penalty on length.

The final set of variables in Figure 2 demonstrate an important Promex feature. Variables whose name ends in `_array` must contain a comma separated list. This list will be split into an array on input. The array's name will be the attribute name with the `_array` removed. A common use of this feature is to set a list of possible candidates to be searched over. For example, the array called `tube_od_maxes_array` is a list of tube outside major axes which will be examined in the search over the design space. The array name must be plural to avoid a name clash. If the list had been named `tube_od_max_array`, then the array name would have clashed with the scalar `tube_od_max`. Single element arrays are allowed and common. One way of debugging is to set all the search lists to a single number. But the plural name rule for arrays must still be followed.

The final set of variables in Figure 2 tells Promex to examine all the combinations of tube bundle widths, tube major axes, tube aspect ratios (major axis/minor axis), and spiral pitch ratios (spiral pitch/tube major axis) in the corresponding four lines.

As noted earlier and as Figures 1 and 2 make clear, using XML allows Promex to accept a different set of input variables for different design concepts. But this flexibility does means that the attribute name in an input file is at least as important as the value. A mistake or typo in the left column is a bigger mistake than a mistake in the right column. Hopefully, you will get an "uninitialized variable" warning, but don't count on it.

Figure 1: Sample Input for a Cylindrical, Baffled HX

```
<!-- close approximation of the ORNL 4541 PHX as described in ORNL TM-2815 -->
<promex_input>
<user
    run_title         = "emulates ORNL-TM-2815 wo bugs except optimizes at kg/kW=4" : displayed on output
    hx_label          = "ornl_2815_phx" : short label to be displayed on in ouput for this run
    run_label         = "ornl_4541"     : base file name for the output for this run

    tubes_pd_func     = "ornl_2815"     : tube side pressure drop model, must match name in TUBES_PD
    tubes_ht_func     = "ornl_2815"     : tube side heat transfer model, must match name in TUBES_HT
    tubes_heat_method = "constant_c_p"  : tube side enthalpy change model, match name in COMMON
    shell_pd_func     = "ornl_2815"     : shell side pressure drop model, must match name in SHELL_PD
    shell_ht_func     = "ornl_2815"     : shell side heat trasnfer model, must match name in SHELL_HT
    shell_heat_method = "constant_c_p"  : shell side enthalpy change model, match name in COMMON
    wall_ht_func      = "circular"      : tube wall heat transfer model, match name in WALL_HT
    cost_model        = "simplest"      : HX cost model, must match name in COST
    stress_model      = "stub"          : HX stress model, must match name in STRESS
    tol_W_pct         = "0.1"           : increment allowable power error as a fraction
    relax_factor      = "1.0"           : increment relaxation factor, 0.5 is safer, 1.0 is faster
    wants_stress      = "N"             : Y if stress analysis required (NOT IMPLEMENTED)

    duty_W            = "556.78e6"      : required thermal power, watts
    tubes_fluid       = "msbr_fuel"     : tubeside fluid name, must match name in MAT
    shell_fluid       = "msbr_cool"     : shell side fluid name, must match name in MAT
    tubes_hot_end_Pa  = "1.241e6"       : tube side hot end pressure, pascals
    shell_hot_end_Pa  = "0.2344e6"      : shell side hot end pressure, pascals
    shell_hot_C       = "621.11"        : shell side hot end temperature, C
    tubes_hot_C       = "704.44"        : tube side hot end temperature, C
    shell_cold_C      = "454.44"        : shell side cold end, target temperature C
    tubes_cold_C      = "565.55"        : tube side cold end, target temperature C

    hx_mat_usd_p_kg   = "50.0"          : HX material cost, $/kg
    pump_usd_p_kW     = "200.0"         : HX pump cost, $/kW

    geometry       = "cylinder"         : currently either "cylinder" (Default) or "ring"
    has_baffles       = "Y"             : Y if HX is baffled
    tube_wall_mat     = "alloy_n"       : tube wall material name, must match name in MAT
    hx_mat            = "alloy_n"       : rest of HX material name, must match name in MAT
    ht_leak_factor    = "0.80"          : leak factor for heat transfer calcs, fraction
    pd_leak_factor    = "0.52"          : leak factor for pressure drop calcs, frction
    dncmr_or          = "0.254"         : Outside radius of central downcomer, m
    gap               = "9.525e-3"      : Gap between tube bunker and shell/downcomer, m
    hx_ir_max         = "3.0"           : Largest HX radius in search
    has_grooves       = "Y"             : Y if tubes have helical indentations
    tube_od           = "9.525e-3"      : tube outside diameter, m
    tube_wall_thk     = "0.890e-3"      : tube wall thickness,m
    tube_arrange      = "R"             : tube pitch arrangement R/T, radial/triangular
    radial_pitch      = "19.050e-3"     : tube radial pitch, m
    circum_pitch      = "19.050e-3"     : tube circumferential pitch, m
    disk_cut          = "0.400"         : fraction open area in way of disk
    donut_cut         = "0.400"         : fraction open area in way of donut
    shell_thk         = "31.25e-3"      : HX shell thickness, m
    ntube_sheets      = "2"             : number of tube sheets, 1 if U-tube
    tube_sheet_thk    = "121.0e-3"      : tube sheet thickness, m
    baffle_thk        = "12.5e-3"       : baffle thickness, m
/>
</promex_input>
```

Figure 2: Sample Input for a Ring Segment, Twisted Tube Heat Exchanger

```
<promex_input>
<user
    run_title           = "ring, twisted, nabe fuel shell, nabe tubes, kW/kg = 4, enormouse penalty on height"
    hx_label            = "twisted_top"
    run_label           = "top_nabe"

    tubes_pd_func       = "twisted_2001"
    tubes_ht_func       = "twisted_2011a"
    tubes_heat_method   = "constant_c_p"
    shell_pd_func       = "twisted_2001"
    shell_ht_func       = "twisted_2001"
    shell_heat_method   = "constant_c_p"
    wall_ht_func        = "circular"
    cost_model          = "simplest"
    stress_model        = "stub"
    tol_W_pct           = "0.1"
    relax_factor        = "1.0"
    wants_stress        = "N"
    hx_len_max          = "15.0"          : infeasible if HX length exceeds this value
    hx_len_del          = "0.2"           : length increment in search

    duty_W              = "556.78e6"
    tubes_fluid         = "nabe_cool"
    shell_fluid         = "nabe_fuel"     : fuel salt on shell side
    tubes_hot_end_Pa    = "0.2344e6"
    tubes_hot_C         = "621.11"
    tubes_cold_C        = "454.44"
    shell_hot_end_Pa    = "1.241e6"
    shell_hot_C         = "704.44"        : shell hot > tubes hot, so heat flow is reversed
    shell_cold_C        = "565.55"

    hx_mat_usd_p_kg     = "50.0"
    pump_usd_p_kW       = "200.0"
    hx_len_usd_p_m      = "1.0e7"         : very large length penalty to find shortest feasible HX

    geometry            = "ring"          : segment of ring
    has_twisted         = 'Y'             : twisted tubes
    has_baffles         = 'N'             : no baffles
    tube_wall_mat       = "alloy_n"
    hx_mat              = "alloy_n"
    ring_pr             = "1.860"         : radius of ring at middle of HX
    bundle_arcl         = "3.251"         : arc length of bundle at mid radius
    gap                 = "9.525e-3"
    tube_wall_thk_ratio = "0.1"
    shell_thk           = "31.25e-3"
    ntube_sheets        = "2"
    tube_sheet_thk      = "121.0e-3"

    bundle_wids_array   = "0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.060, 1.1, 1.2, 1.4, 1.6"
    tube_od_maxes_array = "9.525e-3, 12.25e-3, 15.875e-3, 19.05e-3, 22.225e-3, 25.4e-3"
    aspect_ratios_array = "3.333, 2.500, 2.000, 1.429, 1.111"
    pitch_ratios_array  = "5, 10, 20"
/>
</promex_input>
```

7 Running Promex

The standard way of running Promex is to issue the `promex.pl` command in the Design folder for the concept you are working on. `promex.pl` does a search over the design space and, if all goes well, comes up with the minimum cost heat exchanger that meets the requirements of the user's input file. For example, currently the `ornl4541` search is over a range of HX diameters and baffle spacings. The `twist0` search is over a range of HX diameters, tube major axes, tube minor axes, and spiral pitch. For each such combination, Promex starts at the hot end of the HX and keeps adding length increments until it reaches the user specified cold end temperatures or finds that the combination is infeasible.

The hard way to do a Promex design run is to issue

`$PROMEX_PATH/`*`folder`*`/promex.pl -u`*`inputfile`*

where *`folder`* is the Design folder for the concept you are working on, say `ornl4541` and *`inputfile`* is the name of your data input file, for examples `myrun_15.xml`. You can shorten the typing considerably by assigning `$PROMEX_PATH/`*`folder`* to an environment variable, say `MYPX` and then issuing

`$MYPX/promex.pl -u`*`inputfile`*

or you can put the command you want in little shell script with a short name like `run_px` and then issue

`run_px`

CTX uses a scheme in which the Design folder of the project is copied to a folder in the project's directory tree, and this copy of the Design folder code is then run from the project directory. This facilitates ad hoc changes to the Design folder code but keeps the standard Promex folders off-limits. This may or may not make sense depending on the rules of your shop.

By default, Promex simply displays its result to your terminal. Often it is a good idea to pipe the Promex command to `tee` *`xxxxxxx`* which will both display the results on your terminal and save them in *`xxxxxxx`*.

In addition to the `-u` user file argument, Promex takes the following command line options

`-k` Promex creates a log file for every run. By default, it will automatically delete this file at the end of the run, unless there are problems. If you include `-k` on the command line, the log file will not be deleted. This can be useful both in debugging and in seeing how the parameter search can be speeded up.

`-d`*`n`* This sets the debug level to *`n`*. *`n`* must be an integer between 0 and 9.

The higher the *n*, the more output you will get. The default is zero, no debugging output.

`-b` This options replicates the Primex bugs. Currently, only the `ornl4541` Design folder and the heat transfer and pressure drop correlations lifted from ORNL-TM-2815 respond to this option. It will probably go away in future releases.

`-x` This option will result in the summary output being written to an XML file. This is an ideal format for downstream processing. Or you can place the `promex.pl` command within a script that has a larger design horizon. That script can extract whatever it needs from each Promex run from the run's XML output file, which file will be called `out_`*run_label*`.xml`, where *run_label* is a string specified in the user's input file. See Figure 1.

`-L` By default, Promex displays its output on the screen as a simple pipe-delimited table. Ideally, we would like to have the capability of output in a variety of markup languages. Currently, the only such markup language, other than XML, is Latex. If you set the [-L] option, Promex will output the results as two Latex tables, which can be easily incorporated in Latex formatted reports and papers. The two files will called `tbl_promex_`*run_label*`.tex` and `tbl_promex_`*run_label*`_steps.tex`. The first file contains a summary; the second file shows the result length increment by length increment.

8 Output

Tables 1 and 2 show typical Promex output. Table 1 shows the overall stuff. Table 2 show the step-wise results, length increment by length increment. This HX is baffled so the length increment is the baffle spacing.

Table 1: Summary results for the ornl4541 test case

Promex version/variant:	0.08/ornl4541		
ornl_2815_phx		Input file=ornl2815.xml	
Tube-side pressure drop correlation:		ornl_2815	
Tube-side heat transfer correlation:		ornl_2815	
Shellside pressure drop correlation:		ornl_2815	
Shellside heat transfer correlation:		ornl_2815	
Tube wall heat transfer correlation:		circular	
Cost model:	simplest	Stress model:	stub
Tube_side heat method:	constant_c_p	Shellside heat method:	constant_c_p
Heat load required(MW)	556.8		
Tube side inlet pressure(MPa)	1.241	Shell outlet pressure (MPa)	0.234
High temp tube side C	704.4	High temp shell side(C)	621.1
Low temp tube side C	565.5	Low temp shell side C	454.4
Heat transfer leakage factor	0.800	Pressure leakage factor	0.520
Tube wall material	alloy_n	HX material	alloy_n
Tubeside fluid	msbr_fuel	Shellside fluid	msbr_cool
HX material cost (USD/kg)	50.00	Pump cost (USD/kW)	200.00
Central downcomer OD (mm)	254.00	Shell to tube gap (mm)	9.525
Max allowed HX radius(m)	3.000		
Use enhanced tubes	Y	Use stress analysis	N
Tube Outside diameter(mm)	9.525	Tube wall thickness(mm)	0.89000
Radial pitch(mm)	19.050	Circumferential pitch(mm)	19.050
Shell thickness(mm)	31.250	Tube sheet thickness(mm)	121.000
Inner baffle cut (fraction)	0.400	Outer baffle cut (fraction)	0.400
Number of tube sheets	2	Baffle thickness(mm)	12.500
Total heat transferred(MW)	546.7	Percent of target	98.20
Tube-side mass flow rate(kg/s)	2955.2	Shellside mass flow rate(kg/s)	2216.7
Tubes-side pressure drop(MPa)	1.23848	Shellside pressure drop(MPa)	0.61892
Exchanger internal diameter(m)	1.6510	Exchanger length(m)	7.32
Number of baffles	22	Baffle spacing(m)	0.3329
Fluid volume in tubes(m3)	1.838	Heat transfer area(m2)	1167.22
Total number of tubes	5326	Total tube_length(m)	7.324
Tube wall average temp(C)	594.92		
Tube-side average temp(C)	636.61	Shellside average temp(C)	539.71
Exchanger weight(kg)	28009	Exchanger material cost(USD)	1586095
Tube-side pump kW (HX only)	1100.2	Shellside pump kW (HX only)	734.4
Pump cost (HX only) (USD)	366904		
Total cost(USD)	1952999		

If the "optimum" HX according to Promex ends up on a biundary of the search space, then there is a very good chance that a better optimium lies outside the search space, and the designer needs to consider expanding the search space.

Table 3 shows a portion of the XML output file for the same input. This file was created by issuing

```
$PROMEX_PATH/ornl4541/promex.pl -uornl2815.xml -x
```

The XML output is ugly, but computers are not into aethetics. If you are using Promex inside a broader analysis simply slurp the XML into your code with whatever XML parser you have available and pick out the output variables you need. For example, a plant design progrm may only be interested in the heat exchanger's length, diameter, weight, and pressure drops. If you are using Perl, after slurping, these would show up in the `$promex_output->{hx_len}` etc variables.

Table 2: Step-wise results for the ornl4541 test case

I	shell_C	tube_od	tube_id	tubes_C	shell	shell pd	tubes	tubes pd	tubes	shell	total	Heat	shell	tubes
		C	C		Re	Pa	Re	Pa	W/m2-K	W/m2-K	W/m2-K	MW	m/s	m/s
0	621.1	0.0	0.0	704.4	0	0	0	0	0	0	0	0.000	0.0000	0.0000
1	613.9	658.2	680.2	698.4	30603	28981	12582	54440	21135	11420	5436	24.207	1.9230	3.5869
2	606.5	651.0	673.1	692.3	29974	28899	12255	54747	20590	11529	5424	24.501	1.9175	3.5825
3	599.3	643.2	665.0	686.3	29349	28817	11933	55060	18611	11465	5262	24.111	1.9120	3.5781
4	592.0	636.5	658.5	680.2	28730	28735	11618	55380	18314	11398	5224	24.272	1.9066	3.5737
5	584.7	629.7	651.8	674.1	28110	28654	11304	55709	18031	11335	5188	24.437	1.9013	3.5693
6	577.4	622.9	645.2	668.0	27489	28573	10993	56048	17745	11270	5150	24.595	1.8959	3.5649
7	570.0	616.0	638.4	661.8	26867	28492	10685	56398	17454	11205	5112	24.745	1.8905	3.5605
8	562.5	609.1	631.6	655.6	26245	28410	10380	56758	17159	11138	5073	24.888	1.8851	3.5560
9	555.0	602.2	624.8	649.4	25624	28329	10078	57129	16860	11070	5032	25.022	1.8797	3.5516
10	547.5	595.1	617.8	643.1	25003	28248	9780	57158	16465	11001	4982	25.104	1.8743	3.5471
11	540.0	587.9	610.7	636.8	24385	28167	9485	57065	16024	10930	4927	25.153	1.8689	3.5426
12	532.4	580.8	603.5	630.5	23770	28086	9196	56976	15584	10859	4870	25.187	1.8635	3.5382
13	524.9	573.5	596.3	624.3	23159	28006	8911	56889	15146	10786	4812	25.209	1.8582	3.5337
14	517.3	566.3	589.1	618.0	22552	27926	8630	56807	14711	10713	4753	25.215	1.8529	3.5293
15	509.8	559.1	581.9	611.7	21950	27846	8355	56727	14278	10638	4692	25.207	1.8477	3.5248
16	502.3	551.8	574.6	605.4	21353	27767	8085	56652	13848	10563	4631	25.184	1.8424	3.5204
17	494.7	544.5	567.3	599.1	20762	27689	7821	56580	13421	10486	4567	25.145	1.8372	3.5160
18	487.2	537.2	559.9	592.9	20177	27611	7562	56512	12998	10409	4503	25.090	1.8321	3.5116
19	479.7	529.9	552.5	586.6	19599	27534	7308	56448	12579	10331	4437	25.018	1.8269	3.5073
20	472.3	522.6	545.2	580.4	19028	27458	7060	56388	12163	10253	4370	24.929	1.8219	3.5029
21	464.8	515.3	537.8	574.2	18465	27382	6819	56333	11752	10173	4302	24.822	1.8168	3.4986
22	457.4	508.0	530.4	568.1	17911	27307	6583	56282	11346	10093	4232	24.697	1.8119	3.4943

target_shell_cold_C= 454.4 target_tubes_cold_C= 565.5

Table 3: Portion of the XML output for the ornl4541 test case

```
<promex_output baf_ctc="0.33289875"
               baffle_thk="12.5e-3"
               baffs_kg="2824.40737169093"
               baffs_usd="141220.368584547"
               cflow_circum_i="1.75844394965195"
               cflow_circum_o="2.06164235016081"
               circum_pitch="19.050e-3"
               cost_model="simplest"
               disk_cut="0.400"
               disk_or="0.65405"
               dncmr_or="0.254"
               donut_cut="0.400"
               donut_ir="0.5588"
               duty_W="556.78e6"
               g_i="3661.8057802264"
               g_m="3486.24252405528"
               g_o="3342.62742078302"

               ....

               tube_wall_thk="0.890e-3"
               tubes_ave_C="636.648330052825"
               tubes_cold_C="565.55"
               tubes_farea="0.25091919621178"
               tubes_fluid="msbr_fuel"
               tubes_heat_method="constant_c_p"
               tubes_hot="1"
               tubes_hot_C="704.44"
               tubes_hot_end_Pa="1.241e6"
               tubes_ht_func="ornl_2815"
               tubes_kg="8406.78888356391"
               tubes_kg_p_s="2955.24064115633"
               tubes_pd_func="ornl_2815"
               tubes_pump_W="1100177.18292426"
               tubes_pump_usd="220035.436584853"
               tubes_total_Pa="1238484.54175607"
               tubes_usd="420339.444178195"
               user_file="ornl2815.xml"
               wall_ave_C="594.437195683434"
               wall_ht_func="circular"
               wants_stress="N">
</promex_output>
\normalsize
```

9 Hacking Promex

Promex is intended to be hacked. The command line options, `-k` amd `-d` can be a big help in this process. In addition, almost all the Promex subroutine files include test code. To run these tests, go to the routine's folder and issue *xxxxxxxx.pl* where *xxxxxxxx.pl* is the subroutine's file name. For example, the test code for the material properties files, produces a table of material proerties for the file's material. In some cases, a separate test has not been implemented in which case you should get a message, telling you how to test the file's code. Your own code should include such tests.

Other than that you need only follow the programming interfaces described in the remainder of this manual.

10 The Materials Folder, *MAT*

The remainder of the manual documents the programming interfaces between the various library and utility routines.

We will start with the material properties functions in the `MAT` subfolder. For each heat exchanger fluid, Promex requires density, conductivity, viscosity, and specfic heat as a function of temperature and pressure. If you need a fluid that is not already in the library, or are unhappy with the routines for a fluid that are already there, you must do two things:

1. Add a file containing four subroutines for your fluid. each of these subroutines must take two arguments:
   (a) The fluid temperature in Kelvin.
   (b) The fluid absolute pressure in Pascals.

   Your subroutines may ignore either or both of these arguments but the arguments must be there. The subroutines should be named:
   (a) `promex_k_t_`***xxxxxx*** which returns conductivity in W/m2-K
   (b) `promex_c_p_`***xxxxxx*** which returns specific heat in J/kg-K
   (c) `promex_rho_`***xxxxxx*** which returns density in kg/m3
   (d) `promex_mu_`***xxxxxx*** which returns viscosity in Pa-s.

   xxxxxx is any name you want as long as it is not already in the library. Each subroutine must return the indicated property at the given temperature and pressure. Once again everything must be in strict, prefix-less SI.

   The easiest way to prepare such a file is to copy over an existing fluid properties file and make the necessary changes. Make sure you change ***ALL*** the old function names. Your fluid property may depend on temperature and pressure in any way you like, but table look ups can slow Promex down a lot. Usually fitting a curve to your property data is the better way to go.

   It is very bad form to wipe out routines that are already in the library. Even if you are "replacing" routines for an existing material with better ones, you should use a new material ID, eg *flibe_ucb* rather than overwriting the old *flibe* routines.
2. Register your four routines in `promex_mat_funcs.pl`. This file contains a two dimensional table of subroutine names. You will need to add four lines to this table of the form

```
$mat_funcs->{yyyyyy}{c_p}  = 'promex_c_p_xxxxxx';
$mat_funcs->{yyyyyy}{k_t}  = 'promex_k_t_xxxxxx';
$mat_funcs->{yyyyyy}{rho}  = 'promex_rho_xxxxxx';
$mat_funcs->{yyyyyy}{mu}   = 'promex_mu_xxxxxx';
```

Usually, but not necessarily, *yyyyyy* and *xxxxxx* are the same.
All Promex input files have two lines of the form
`shell_fluid = "aaaaaa"`
`tubes_fluid = "bbbbbb"`
Whenever the user sets *aaaaaa* or *bbbbbb* to *yyyyyy*, she will end up using your new material properties functions. Solid material properties work in exactly the same way, but currently Promex only needs conductivity and density.

Promex often searches widely if sometimes blindly over the design space. If a candidate HX is infeasible, temperatures may not converge. You must assume your routines will be called with out of range input in which case your routines must place an error message in `PROMEX_LOGFILE` and return a negative number. It is the caller's responsibility to open and close `PROMEX_LOGFILE`. In particular for fluids, your routine should check that the material temperature is between the melting point and the boiling point. Currently, we have adopted the short-cut that only the viscosity routine does this test, blithely assuming that if any of the material routines are called, the viscosity routine will be called, and the out-of-range temperature caught.

Any new materials file should include test code. This test code should be triggered by issuing the file name from the `MAT` folder. Typically, this test code produces a materials property table which can be compared with your data and other sources. See one of the existing materials files for how this is done.

11 The Heat Transfer Folders, *SHELL_HT* and *TUBES_HT*

The shell side heat transfer functions are filed in the `SHELL_HT` folder; the tube-side in `TUBES_HT`. These functions compute the average heat trasnfer rate in a given length increment.

To add a new shellside heat transfer function, simply put the code in `SHELL_HT` giving it a unique name within the folder. To use this function, the user must set the attribute `shell_ht_func` to this name (without the `.pl`) in the user input file.

The arguments to this routine must be in order

1. a reference to the `$hx` hash
2. the length of this increment (m)
3. the average shellside fluid temperature in this increment (K)
4. the average shellside pressure in this increment (Pa)
5. the average shellside tube wall temperature in this increment (K)
6. debug level.

Promex puts all the heat exchanger parameters, both user input and internally calculated values in a big hash referenced by `$hx`. For example, the tube OD is in `\$hx->{tube_od}`. Thus your routine has access to any variable in this hash. You can inspect this hash by placing `promex_print_hx($hx)` at the top of your code. `$hx` contains not only all the heat exchanger physical parameters but all the user input including material choices.

Debug level is an integer between 0 and 9. The higher the debug level, the more debugging information the caller wants. Debugging messages should be written to STDERR. You can respond to this parameter anyway you want except zero means no debugging messages.

Your function is bound to be called with crazy input. You must check the value returned from each material property function, and return immediately with `(-1, 0, 0, 0, 0, 0, 0, 0)` if it is non-positive. The material property function will have placed an error message in `PROMEX_LOGFILE`. If the error occurs in your routine, — for example, an out-of-range Reynolds number — you must write an error message to `PROMEX_LOGFILE` and then return with `(-1, 0, 0, 0, 0, 0, 0, 0)`. It is the caller's responsibility to open and close `PROMEX_LOGFILE`.

If there are no errors, your routine must return in order:

1. the average heat transfer rate, W/m2-K, in the increment
2. the specific heat used, J/kg-K
3. the Nusselt number,

4. a characteristic Reynolds number,
5. the Prandtl number used,
6. the corresponding fluid velocity (m/s),
7. the average viscosity (Pa-s),
8. the average density (kg/m3).[10]

Your file should include test code which is triggered when the user issues the filename from this folder. See one of the existing routines for how this is done.

The situation is exactly the same for tube-side heat transfer fucntions except the code goes in the `TUBES_HT`. File and function names need be unique only within their respective folders. A shell side heat transfer function can have the same name as a tube side. This is typically the case when both come from the same source.

[10] If your routine does not compute one of these values — other than the heat transfer rate — return a -1 in the corresponding slot.

12 The Pressure Drop Folders, *SHELL_PD* and *TUBES_PD*

The shell side pressure drop functions are filed in the `SHELL_PD` folder; the tube-side in `TUBES_PD`. These functions compute the pressure drop in a given length increment.

To add a new shellside pressure drop function, simply put the code in `SHELL_PD` giving it a unique name within the folder. To use this function, the user must set the attribute `shell_pd_func` to this name (without the `.pl`) in the user input file.

The arguments to this routine are exactly the same as for the heat trasnfer functions. See Section 11.

Your function is bound to be called with crazy input. You must check the value returned from each material property function, and return immediately with `(-1, 0, 0, 0, 0)` if it is non-positive. The material property function will have placed an error message in `PROMEX_LOGFILE`. If the error occurs in your routine, — for example, an out-of-range Reynolds number — you must write an error message to `PROMEX_LOGFILE` and then return with `(-1, 0, 0, 0, 0)`. It is the caller's responsibility to open and close `PROMEX_LOGFILE`.

If there are no errors, your routine must return in order:

1. the pressure drop inthe increment, Pa
2. a characteristic Reynolds number,
3. the corresponding fluid velocity (m/s),
4. the average viscosity (Pa-s),
5. the average density (kg/m3).[11]

Your file should include test code which is triggered when the user issues the filename from this folder. See one of the existing routines for how this is done.

The situation is exactly the same for tube-side pressure drop fucntions except the code goes in the `TUBES_PD`. File and function names need be unique only within their respective folders. A shell side pressure drop function can have the same name as a tube side.

[11] If your routine does not compute one of these values — other than the pressure drop — return a -1 in the corresponding slot.

13 The STRESS Folder

Currently, no stress analysis is implemented, but thanks to the flexibility of Perl hashes we do have an outline of an API. The stress routine will be called only after the candidate HX has proven itself feasible from a heat transfer point of view and pressure drop point of view. The stress routine will be called with

`$hx` This hash contains all the HX parameters which have been calculated up to this point.

`$steps` This array of hashes contains all the step-wise results for the HX including the temperatures and pressures in each increment.

`$debug_level`

Thus the stress and strength routine will have access to everything that it needs to compute the stresses. The routine will be free to vary shell and tube sheet thickness but not tube wall thickness.

If the stress analysis is successful, the routine should add the results to `$hx` and `$steps` and return a 'Y'. Otherwise it will need to write a message to `PROMEX_LOGFILE` explaining why the candidate HX was infeasible and return a non-Y.

To implement stress analysis, we will need to expand the material properties library to include allowable stress as a function of temperature. These routines will use the same API as the existing material properties functions.

14 The COST Folder

The user selects the costing model she wants by the `cost_model` attribute in her input file. There must be a file in the COST folder whose name is the same as the value of this attribute with a `.pl` appended and that file must contain a costing routine with this name. The cost model should be called only after the candidate HX has passed all the feasibility tests.

The cost routine is called with `$hx` and if needed `$steps` and the ubiquitous `$debug_level`. The cost routine should add at least the following variables to the `$hx` hash.

`$hx->{total_usd}` The estimated cost of the heat exchanger in dollars.

`$hx->{hx_kg}` The total mass of the heat exchanger in kilograms.

`$hx->{tubes_pump_W}` The tube-side pump power required in watts. This should be the net pump power required to handle only the pressure drop in the tubes. Do not apply any pump efficiencies or worry about pressure drops elsewhere in the loop.

`$hx->{tubes_pump_W}` The tube-side pump power required in watts. This should be the net pump power required to handle only the pressure drop on the shell side. Do not apply any pump efficiencies or worry about pressure drops elsewhere in the loop.

Other output such as a more detailed breakdown of the weight or cost can also be added to the `$hx` hash.

Utility routines are provided to aid in these calculations. See the code.

The cost routine must return `$hx->{total_usd}`.

Be aware that in future versions of Promex we will probably move costing into the project folder. It was a bad decision to try and provide non-project specific costing.

15 The COMMON Folder

The COMMON is the most important Promex folder. It is the core of Promex. You probably should not mess with the code in this folder unless you are a maintainer. This folder contains the following routines in files with the same name as the routine with a .pl appended.

promex_setup This routine should be called just after the user input is read in. It sets all the various functions to the user's choices and computes the mass flow in kg/s on both sides using the user specified shell_heat_method and tubes_heat_method. Currently the only heat method that is implemented is constant_c_p which bases the mass flow on the material specific heat at the average between the hot end and cold end temperatures. constant_c_p is also filed in this folder.

promex_find_temp This is the heart of the beast. Given the hot end temperatures in any increment it computes the cold end temperatures in that increment by trial and error. If successful it adds the results for this increment to $steps, and returns 1 plus some numbers which are used to initiate the calculation for the next increment. If unsuccessful, it returns a negative number. The temperature convergence logic is lifted from Primex.

promex_length Starting at the hot end of the HX, this routine keeps adding length increments until it reaches the user's specified cold end temperature or an error occurs. It also compute the pressure drops in each increment. If successful, it adds all its results, most importantly the HX length, to $hx and $steps and returns Y indicating that this candidate HX is feasible as far as heat transfer is concerned. Otherwise it returns N.

All the real work is done by promex_length and promex_find_temp. The user's main program need only set the problem up, call promex_length for each candidate HX that it wants examined, cost the results if the candidate is feasible, and pick out the most interesting candidates.

promex_pd_check This little routine checks the tubeside and shell side pressure drops against the user-supplied attribute tubes_max_Pa and shell_max_Pa. If you do not want to impose pressure drop constraints, simply don't include these attributes in your input file.

promex_save_hx This routine is called with $hx and $steps. It does a deep copy of these two data structures, and returns references to these

copies. This is an easy way to save a particular HX and its performance. One obvious use is the save the minimum cost HX found so far. The current implementation of `promex_save_hx` requires the Perl Storable module which is distributed with most recent Perl distributions. If Storable is not installed at your site, you can download it from CPAN.

16 The IN_OUT Folder

The IN_OUT contain non-design specific input and output routines. This folder contains the following routines in files with the same name as the routine with a .pl appended.

promex_input This routine reads in the user's input file. Currently, it uses the Perl XML::Simple function XMLin which does all the dirty work of parsing the XML and putting the variables into the $hx hash keyed by the attribute name. promex_input also splits the array strings into arrays. And it adds some meta-data such as the current PROMEX_PATH and date and time to the $hx hash. The file includes some test code which can be executed via issuing promex_input.pl for the IN_OUT folder. See the code for the options.

promex_print_hx This routine is called with a reference to a hash. It prints out all the scalars in that hash in semi-readable form. It can be used for inspecting a $hx at any point. Remember just after the user's input file is read in, $hx will contain only the user's input but as the calculation proceed lots of stuff gets added. Also $hx is really the hash representing the current candidate HX under analysis and keeps changing. If you want to save a particular HX for whatever reason, you need a line like

```
($my_hx, $my_steps) = promex_save_hx($hx, $steps, $debug_level);
```

and then you can use promex_print_hx($my_hx); to see what you saved.

promex_hx2xml This routine is called with a reference to a hash. It turns that hash back into XML and prints the results to out_*xxxxxx*.xml where *xxxxxx* is the value of the run_label attribute in the user's input file. Output in this form is difficult to read, but it is ideal for post-processing by other programs. A program that needs the results of a Promex run, need only slurp in the contents of out_*xxxxxx*.xml using for example XMLin (see the code in promex_input.pl), and pick out the numbers it needs from the resulting hash.

promex_output Right now readable Promex output is bit of a mess. The problem is that a non-design specific output routine cant know what variables the program is using nor which of these are important enough

to include in the output. `promex_ouput` is a compromise that tries to print out the most important variables for baffled and twisted tube HX. This area needs a lot of work. One possibility is to change the user input file so it tells Promex which variables to show on the output.

You will see some Latex generators in this folder. You can ignore them for now.

17 The Design Folders

All design folders must contain at least two Perl files:

promex_layout_hx.pl Given a candidate HX, the `promex_layout_hx` subroutine in this file must calculate all the physical parameters of the HX such as number of tubes, cross-section flow areas, etc that do not depend on HX length, for the user's specific HX geometry. This routine should be called as soon as all the HX parameters other than length are fixed. Normally this will be at the top of the innermost loop in the main program. The program takes `$hx` and `$debug_level` and adds the result of its calculations to the `$hx` hash. Best way to see how this works is study the code in `ornl4541` or `twist0`.

promex.pl The Promex main program, `promex` in `promex.pl` is quite simple. It consists of nothing more than a set of loops over whatever HX parameters the users wants to regard as variable, repeatedly calling the various Common routines to evaluate each possible candidate. Moreover all Promex main programs do nearly the same thing. In most cases, you can create the main program you need by copying over an existing main program and then modifying the loops slightly to fit your needs. Best way to see how this works is study the code in `ornl4541` or `twist0`.