CSE User's Manual

February 23, 2016

Contents

1 Introduction

1.1 Greetings

The purpose of this manual is to document the California Simulation Engine computer program, CSE. CSE is an hourly building and HVAC simulation program which calculates annual energy requirements for building space conditioning and lighting. CSE is specifically tailored for use as internal calculation machinery for compliance with the California building standards.

CSE is a batch driven program which reads its input from a text file. It is not intended for direct use by people seeking to demonstrate compliance. Instead, it will be used within a shell program or by technically oriented users. As a result, this manual is aimed at several audiences:

1. People testing CSE during its development.
2. Developers of the CSE shell program.
3. Researchers and standards developers who will use the program to explore possible conservation opportunities.

Each of these groups is highly sophisticated. Therefore this manual generally uses an exhaustive, one-pass approach: while a given topic is being treated, *everything* about that topic is presented with the emphasis on completeness and accuracy over ease of learning.

Please note that CSE is under development and will be for many more months. Things will change and from time to time this manual may be inconsistent with the program.

1.2 Manual Organization

This Introduction covers general matters, including program installation.

Next, Section 2 (About CSE) will describe the program and the calculation techniques used in it.

Section 3 (Operation) documents the operational aspects of CSE, such as command line switches, file naming conventions, and how CSE finds files it needs.

Section 4 (Input Structure) documents the CSE input language in general.

Section 5 (Input Data) describes all of the specific input language statements.

Section 6 (Output) will describe the output reports.

Finally, Appendix A gives an example CSE input file and the output it produces.

1.3 Installation

1.3.1 Hardware and Software Requirements

CSE is a 32-bit Microsoft Windows console application. That is, it runs at the command prompt on Windows XP, Windows Vista, and Windows 7. Memory and disk space requirements depend on the size of projects being modeled, but are generally modest.

To prepare input files, a text editor is required. Notepad will suffice, although a text editor intended for programming is generally more capable. Alternatively, some word processors can be used in "ASCII" or "text" or "non-document" mode.

1.3.2 Installation Files

(To be written.)

1.3.3 Installation Procedure

Create a directory on your hard disk with the name \CSE or some other name of your choice. Copy the files into that directory. Add the name of the directory to the PATH environment setting unless you intend to use CSE only from the CSE directory.

1.3.4 Simple Test Run

Page break field here (2)

2 About CSE

2.1 About CSE

2.1.1 About CSE

To be written

3 Operation

3.1 Command Line

CSE is invoked from the command prompt or from a batch file using the following command:

```
CSE *inputfile* {*switches*}
```

where:

inputfile specifies the name of the text input file for the run(s). If the filename has an extension other than ".cse" (the default), it must be included. The name of the file with weather data for the simulation(s) is given in this file (wfName= statement, Section 5).

{switches} indicates zero or more of the following:

- -D*name* defines the preprocessor symbol *name* with the value "". Useful for testing with #ifdef name, to invoke variations in the simulation without changing the input file. The CSE preprocessor is described in Section 4.4.
- -D*name=value* defines the preprocessor symbol *name* with the specified *value*. *Name* can then be used in the input file to allow varying the simulation without changing the input file – see Section 4.4 for more information. The entire switch should be enclosed in quotes if it contains any spaces – otherwise the command processor will divide it into two arguments and CSE will not understand it.
- -b batch mode: CSE will never stop for a response from the user when an error occurs. Error messages may thus scroll off the screen, but will all be in the error message file.
- -p display all the class and member names that can be "probed" or accessed in CSE expressions. "Probes" are described in Section 4. Use with command processor redirection operator ">" to obtain a report in a file. *Inputfile* may be given or omitted when -p is given.
- -q similar to -p, but displays additional member names that cannot be probed or would not make sense to probe in an input file (development aid).
- -x specifies report test prefix; see TOP repTestPfx. The -x command line setting takes precedence over the input file value, if any.

3.2 Locating Files

As with any program, in order to invoke CSE, the directory containing CSE.EXE must be the current directory, or that directory must be on the operating system path, or you must type the directory path before CSE.

A CSE simulation requires a weather file. The name of the weather file is given in the CSE input file (wfName= statement, Section 5). The weather file must be in one of the same three places: current directory, directory containing CSE.EXE, or a directory on the operating system path; or, the directory path to the file must be given in the wfName= statement in the usual pathName syntax. ?? Appears that file must be in current directory due to file locating bugs 2011-07

The CSE input file, named on the CSE command line, must be in the current directory or the directory path to it must be included in the command line.

Included input files, named in #include preprocessor directives (Section 4.4) in the input file, must be in the current directory or the path to them must be given in the #include directive. In particular, CSE will NOT automatically look for included files in the directory containing the input file. The default extension for included files is ".CSE".

Output files created by default by CSE (error message file, primary report and export files) will be in the same directory as the input file; output files created by explicit command in the input file (additional report and/or export files) will be in the current directory unless another path is explicitly specified in the command creating the file.

3.3 Output File Names

If any error or warning messages are generated, CSE puts them in a file with the same name and path as the input file and extension .ERR, as well as on the screen and, usually, in the primary (default) report file. The exception is errors in the command line: these appear only on the screen. If there are no error or warning messages, any prior file will this name is deleted.

By default, CSE generates an output report file with the same name and path as the input file, and extension ".REP". This file may be examined with a text editor and/or copied to an ASCII printer. If any exports are specified, they go by default into a file with the same name and path as the input file and extension ".EXP".

In response to specifications in the input file, CSE can also generate additional report and export files with user-specified names. The default extensions for these are .REP and .CSV respectively and the default directory is the current directory; other paths and extensions may be specified. For more information on report and export files, see REPORTFILE and EXPORTFILE in Section 5.

3.4 Errorlevel

CSE sets the command processor ERRORLEVEL to 2 if any error occurs in the session. This should be tested in batch files that invoke CSE, to prevent use of the output reports if the run was not satisfactory. The ERRORLEVEL is NOT set if only warning messages that do not suppress or abort the run occur, but such messages DO create the .ERR file.

3.4.1 Results Files

3.4.2 Error Files

3.4.3 External File Interface

3.4.4 Hourly files

3.5 Helpful Hints

3.5.1 Memory

4 Input Structure

DRAFT: In the following, any text annotated with ?? indicates areas of uncertainty or probable change. As the program and input language develop, these matters will be resolved.

4.1 Introduction

The CSE Internal/Development Language (CIDL) is the fundamental interface to the CSE program. The language has been designed with three objectives in mind:

1. Providing direct access to all program features (including ones included for self-testing), to assist in program development.
2. Providing a set of parametric and expression evaluation capabilities useful for standards development and program testing.
3. Providing a means for other programs, such as an interactive user interface, to transmit input data and control data to the program.

Thus, the language is not intended to be used by the average compliance or simulation user. Instead, it will be used during program development for testing purposes and subsequently for highly technical parametric studies, such as those conducted for research and standards development. In all of these situations, power, reproducibility, and thorough input documentation take precedence over user-friendliness.

CSE reads its CIDL input from a file. The file may be prepared by the user with a text editor, or generated by some other program.

4.2 Form of the CSE Data

The data used by CSE consists of *objects*. Each object is of a *class*, which determines what the object represents. For example, objects of class ZONE represent thermally distinct regions of the building; each thermally distinct region has its own ZONE object. An object's class determines what data items or *members* it contains. For instance, a ZONE object contains the zone's area and volume. In addition, each object can have a *name*.

The objects are organized in a hierarchy, or tree-like structure. For example, under each ZONE object, there can be SURFACE objects to represent the walls, floors, and ceilings of the ZONE Under SURFACE there can be WINDOW objects to represent glazings in the particular wall or roof. SURFACE is said to be a *subclass* of the class ZONE and WINDOW a subclass of SURFACE each individual SURFACE is said to be a *subobject* of its particular ZONE object. Conversely, each individual SURFACE is said to be *owned by* its zone, and the SURFACE class is said to be owned by the ZONE class.

The hierarchy is rooted in the one *top-level object* (or just *Top*). The top level object contains information global to the entire simulation, such as the start and end dates, as well as all of the objects that describe the building to be simulated and the reports to be printed.

Objects and their required data must be specified by the user, except that Top is predefined. This is done with CIDL *statements*. Each statement begins an object (specifying its class and object name) or gives a value for a data member of the object being created. Each object is specified with a group of statements that are usually given together, and the objects must be organized according to the hierarchy. For example, SURFACE must be specified within ZONE and WINDOW within SURFACE Each SURFACE belongs to (is a subobject of) the ZONE within which it is specified, and each WINDOW is a subobject of its SURFACE

The entire hierarchy of CSE classes can be represented as follows, using indentation to indicate subclasses:

TODO: review hierarchy

```
TOP (Top-level class; object of this class supplied automatically by CSE)
    HOLIDAY
    MATERIAL
    CONSTRUCTION
        LAYER
    METER
    ZONE
        GAIN
        SURFACE
            WINDOW
                SHADE
                SGDIST
            DOOR
        PERIMETER
        TERMINAL
    IZXFER
    AIRHANDLER
    HEATPLANT
        BOILER
    COOLPLANT
    TOWERPLANT
    HPLOOP
    REPORTFILE
    REPORT
    REPORTCOL
    EXPORTFILE
    EXPORT
```

```
EXPORTCOL
```

4.3 Overview of CIDL

CIDL consists of *commands*, each beginning with a particular word and, preferably, ending with a semicolon. Each command is either an *action-command*, which specifies some action such as starting a simulation run, or a *statement*, which creates or modifies an *object* or specifies a value for a *member* of an object.

4.3.1 Statements – Overview

A statement that creates an object consists basically of the *class name* followed by your name for the object to be created. (The name can be omitted for most classes; optional modifying clauses will be described later.) For example,

```
ZONE "north";
```

begins an object of class ZONE the particular zone will be named "north". This zone name will appear in reports and error messages, and will be used in other statements that operate on the zone. As well as creating the ZONE this statement sets CSE to expect statements specifying ZONE data members or ZONE subobjects to follow.

A statement specifying a data member consists of the data member's name, an = sign, an *expression* specifying the value, and a terminating semicolon. An expression is a valid combination of operands and operators as detailed later; commonly it is just a number, name, or text enclosed in quotes. For example,

```
znVol = 100000;
```

specifies that the zone has a volume of 100000 cubic feet. (If the statement occurs outside of the description of a ZONE an error message occurs.) All of the member names for each class are described in Section 5; most of them begin with an abbreviation of the class name for clarity.

The description of a zone or any object except Top can be terminated with the word "END"; but this is not essential; CSE will assume the ZONE ends when you start another ZONE or any object not a subobject of ZONE or when you specify a member of a higher level class (Top for ZONE or give an action-command such as RUN.

Statements are free-form; several can be put on a line, or a single statement can occupy several lines. Indentation according to class hierarchy will help make your input file readable. Spaces may be used freely except in the middle of a word or number. Tab characters may be used. Each statement should end with a semicolon. If the semicolon is omitted and the statement and the following statement are both correctly formed, CSE will figure out your intent anyway. But when there is an error, CSE gives clearer error messages when the statements are delimited with semicolons.

Capitalization generally does not matter in CIDL statements; we like to capitalize class names to make them stand out. Words that differ only in capitalization are NOT distinct to CSE.

Comments (remarks) may be interspersed with commands. Comments are used to make the input file clearer to humans; they are ignored by CSE. A comment introduced with "//" ends at the end of the line; a comment introduced with "/*" continues past the next "*/", whether on the same line, next line, or many lines down. Additional CIDL may follow the */ on the same line.

4.3.2 Nested Objects

The following is a brief CSE input file, annotated with comments intended to exemplify how the CIDL processor follows the object hierarchy when decoding input describing objects and their subobjects.

```
// short example file
                          // initially, the current object is Top.
wfName = "CZ12RV2.CEC"; // give weather file name, a Top member
begDay = Jan 1;           // start and ...
endDay = Dec 31;          // ...end run dates: Top members.

MATERIAL carpet;          // create object of class MATERIAL
matThk = .296;            // specify 'matThk' member of MATERIAL 'carpet'
matCond = 1./24;          // give value of 'matCond' for 'carpet'

CONSTRUCTION slab140C;    /* create object of class CONSTRUCTION, named
                             slab140C. Terminates MATERIAL, because
                             CONSTRUCTION is not a subclass of material
                             in the hierarchy shown in section 4.2. */
  LAYER                   /* start an unnamed object of class LAYER.
                             Since LAYER is a subclass of CONSTRUCTION,
                             this will be a subobject of slab140C. */
    lrMat = carpet;       /* member of the LAYER. Note use of name of
                             MATERIAL object. */
  // (additional layers would be here)

METER Elec;               /* create METER named Elec;
                             since METER is a subobject of Top,
                             this ends slab140C and its LAYER. */

ZONE North;               // start a ZONE named North. Ends METER.
  znArea = 1000;          // specify data members of ZONE North.
  znVol = 10;             // (you don't have to capitalize these as shown.)
  GAIN NorthLights        /* create GAIN object named NorthLights.
                             Creates a subobject of ZONE North. */
    gnPower = 0.01;       // member of NorthLights -- numeric value
    gnMeter = Elec;       // member of NorthLights -- object name value

  znCAir = 3.5;           /* CIDL knows that znCAir is a member of ZONE;
                             thus this statement terminates the GAIN
                             subobject & continues ZONE 'North'. */

  /*lrMat = ...             would be an error here, because the current
                             object is not a LAYER nor a subobject of LAYER */

RUN;                      /* initiate simulation run with data given.
                             Terminates ZONE North, since action-commands
                             terminate all objects being constructed. */
```

4.3.3 Expressions – Overview

Expressions are the parts of statements that specify values – numeric values, string values, object name values, and choice values. Expressions are composed of operators and operands, in a manner similar to many programming languages. The available operators and operands will be described in Section .

Unlike most programming languages, CSE expressions have Variation. *Variation* is how often a value changes during the simulation run – hourly, daily, monthly, yearly (i.e. does not change during run), etc. For instance, the operand $hour represents the hour of the day and has "hourly" variation. An expression has the variation of its fastest-varying component.

Each data member of each object (and every context in which an expression may be used) has its allowed *variability*, which is the fastest variation it will accept. Many members allow no variability. For example, begDay, the date on which the run starts, cannot meaningfully change during the run. On the other hand, a thermostat setting can change hourly. Thermostat settings and other scheduled values are specified in CSE with expressions that often make use of variability; there is no explicit SCHEDULE class.

For example, a heating setpoint that was 68 during business hours and 55 at night might be expressed as

```
select( $hour > 8 && $hour < 18, 68, default 55)
```

An example of a complete statement containing the above expression is:

```
tuTH = select( $hour > 8 && $hour < 18, 68, default 55);
```

The preceding is valid a statement if used in a TERMINAL description. The following:

```
begDay = select( $hour > 8 && $hour < 18, 68, default 55);
```

would always get an error message, because begDay (the starting day of the run) will not accept hourly variation, and the expression varies hourly, since it contains $hour. The expression's variation is considered "hourly" even though it changes only twice a day, since CSE has no variation category between hourly and daily.

CSE's expression capability may be used freely to make input clearer. For example,

```
znVol = 15 * 25 * 8;
```

meaning that the zone volume is 15 times 25 times 8 is the same to CSE as

```
znVol = 3000;
```

but might be useful to you to tersely indicate that the volume resulted from a width of 15, a length of 25, and a height of 8. Further, if you wished to change the ceiling height to 9 feet, the edit would be very simple and CSE would perform the volume calculation for you.

CSE computes expressions only as often as necessary, for maximum simulation speed. For example,

```
tuTH = 68;
```

causes 68 to be stored in the heating setpoint once at the start of the run only, even though tuTH will accept expressions with variability up to hourly. Furthermore, constant inner portions of variable expressions are pre-evaluated before the run begins.

CSE statements and expressions do not (yet) have user-settable variables in the usual programming language sense. They do, however, have user-defined functions to facilitate using the same computation several places, and preprocessor macros, to facilitate using the same text several places, specifying parametric values in a separate file, etc.

4.3.4 The Preprocessor – Overview

The preprocessor scans and processes input file text before the CIDL language processor sees the text. The preprocessor can include (embed) additional files in the input, include sections of input conditionally, and define and expand macros.

Macros are a mechanism to substitute a specified text for each occurrence of a word (the macro name). For example,

```
#define ZNWID 20
#define ZNLEN 30
. . .
```

```
znArea = ZNWID * ZNLEN;
znVol  = ZNWID * ZNLEN * 8;
```

The first line above says that all following occurrences of "ZNWID" are to be replaced with "20" (or whatever follows ZNWID on the same line). The effect of the above is that the zone width and length are specified only one place; if the single numbers are editing, both the zone area and zone volume change to match.

Macros can be especially powerful when combined with the file inclusion feature; the generic building description could be in one file, and the specific values for multiple runs supplied by another file. By also using conditional compilation, the values-specifying file can select from a range of features available in the building description file.

The preprocessor is similar to that of the C programming language, and thus will be familiar to C programmers.

After the next section, which is an example of a complete CSE input file using many features, Section 4.4 describes the preprocessor in detail. The preprocessor description is followed by sections detailing statements, then expressions.

4.3.5 Example

A CSE input file for a building with two zones, four gains, and two air handlers is given in Appendix A. You may wish to look at this example of CIDL now.

4.4 The Preprocessor

Note: The organization and wording of this section is based on section A12 of Kernigan and Richie [1988]. The reader is referred to that source for a somewhat more rigorous presentation but with the caution that the CIDL preprocessor does not ***completely*** *comply to ANSI C specifications.*

The preprocessor performs macro definition and expansion, file inclusion, and conditional inclusion/exclusion of text. Lines whose first non-whitespace character is # communicate with the preprocessor and are designated *preprocessor directives*. Line boundaries are significant to the preprocessor (in contrast to the rest of CIDL in which a newline is simply whitespace), although adjacent lines can be spliced with \, as discussed below. The syntax of preprocessor directives is separate from that of the rest of the language. Preprocessor directives can appear anywhere in an input file and their effects last until the end of the input file. The directives that are supported by the CIDL preprocessor are the following:

```
#if
#else
#elif
#endif
#ifndef

#define
#redefine
#undef

#include
```

4.4.1 Line splicing

If the last character on a line is the backslash \, then the next line is spliced to that line by elimination of the backslash and the following newline. Line splicing occurs *before* the line is divided into tokens.

Line splicing finds its main use in defining long macros:

```
// hourly light gain values:
#define LIGHT_GAIN      .024, .022, .021, .021, .021, .026, \
                        .038, .059, .056, .060, .059, .046, \
                        .045, .5  , .5  , .05 , .057, .064, \
                        .064, .052, .050, .055, .044, .027
```

4.4.2 Macro definition and expansion

A directive of the form

```
#define _identifier_ _token-sequence_
```

is a macro definition and causes the preprocessor to replace subsequent instances of the identifier with the given token sequence. Note that the token string can be empty (e.g. #define FLAG).

A line of the form

```
#define _identifier_( _identifier-list_) _token-sequence_
```

where there is no space between the identifier and the (, is a macro with parameters given by the identifier list. The expansion of macros with parameters is discussed below.

Macros may also be defined *on the CSE command line*, making it possible to vary a run without changing the input files at all. As described in Section 3, macros are defined on the CSE command line using the −D switch in the forms

```
-D_identifier_

-D_identifier_=_token-sequence_
```

The first form simply defines the name with no token-sequence; this is convenient for testing with #ifdef, #ifndef, or defined(), as described in Section 4.4.4. The second form allows an argument list and token sequence. The entire command line argument must be enclosed in quotes if it contains any spaces.

A macro definition is forgotten when an #undef directive is encountered:

```
#undef _identifier_
```

It is not an error to #undef an undefined identifier.

A macro may be re-#defined without a prior #undef unless the second definition is identical to the first. A combined #undef/#define directive is available to handle this common case:

```
#redefine _identifier_ _token-sequence_

#redefine _identifier_( _identifier-list_) _token-sequence_
```

When a macro is #redefined, it need not agree in form with the prior definition (that is, one can have parameters even if the other does not). It is not an error to #redefine an undefined identifier.

Macros defined in the second form (with parameters) are expanded whenever the preprocessor encounters the macro identifier followed by optional whitespace and a comma-separated parameter list enclosed in parentheses. First the comma separated token sequences are collected; any commas within quotes or nested parentheses do not separate parameters. Then each unquoted instance of the each parameter identifier in the macro definition is replaced by the collected tokens. The resulting string is then repeatedly re-scanned for more defined identifiers. The macro definition and reference must have the same number of arguments.

It is often important to include parentheses within macro definitions to make sure they evaluate properly in all situations. Suppose we define a handy area macro as follows:

```
#define AREA(w, h) w*h          // WRONG
```

Consider what happens when this macro is expanded with arguments 2+3 and 4+1. The preprocessor substitutes the arguments for the parameters, then CIDL processes the statement containing the macro expansion without regard to the beginning and end of the arguments. The expected result is 25, but as defined, the macro will produce a result of 15. Parentheses fix it:

```
#define AREA(w, h) ((w)*(h))   // RIGHT
```

The outer enclosing set of parentheses are not strictly needed in our example, but are good practice to avoid evaluation errors when the macro expands within a larger expression.

Note 1: The CSE preprocessor does not support the ANSI C stringizing (#) or concatenation (##) operators.

Note 2: Identifiers are case *insensitive* (unlike ANSI C). For example, the text "myHeight" will be replaced by the #defined value of MYHEIGHT (if there is one).

The preprocessor examples at the end of this section illustrate macro definition and expansion.

4.4.3 File inclusion

Directives of the form

#include "filename" and

#include <filename>

cause the replacement of the directive line with the entire contents of the referenced file. If the filename does not include an extension, a default extension of .INP is assumed. The filename may include path information; if it does not, the file must be in the current directory.

#includes may be nested to a depth of 5.

For an example of the use #includes, please see the preprocessor examples at the end of this section.

4.4.4 Conditional inclusion of text

Conditional text inclusion provides a facility for selectively including or excluding groups of input file lines. The lines so included or excluded may be either CIDL text *or other preprocessor directives*. The latter capability is very powerful.

Several conditional inclusion directive involve integer constant expressions. Constant integer expressions are formed according the rules discussed in Section 4.6 with the following changes:

1. Only constant integer operands are allowed.
2. All values (including intermediate values computed during expression evaluation) must remain in the 16 bit range (-32768 - 32767). The expression processor treats all integers as signed values and requires signed decimal constants – however, it requires unsigned octal and hexadecimal constants. Thus decimal constants must be in the range -32768 - 32767, octal must be in the range 0 - 0o177777, and hexadecimal in the range 0 - 0xffff. Since all arithmetic comparisons are done assuming signed values, 0xffff < 1 is true (unhappily). Care is required when using the arithmetic comparison operators (<, <=, >=, >).
3. The logical relational operators && and || are not available. Nearly equivalent function can be obtained with & and |.
4. A special operand defined( ) is provided; it is described below.

Macro expansion *is* performed on constant expression text, so symbolic expressions can be used (see examples below).

The basic conditional format uses the directive

```
#if _constant-expression_
```

If the constant expression has the value 0, all lines following the #if are dropped from the input stream (the preprocessor discards them) until a matching #else, #elif, or #endif directive is encountered.

The defined(*identifier*) operand returns 1 if the identifier is the name of a defined macro, otherwise 0. Thus

```
#if defined( _identifier_ )
```

can be used to control text inclusion based on macro flags. Two #if variants that test whether a macro is defined are also available. #ifdef *identifier* is equivalent to #if defined(*identifier*) and #ifndef *identifier* is equivalent to #if !defined(*identifier*).

Defined(), #ifdef, and #ifndef consider a macro name "defined" even if the body of its definition contains no characters; thus a macro to be tested with one of these can be defined with just

```
#define _identifier_
```

or with just "-D*identifier*" on the CSE command line.

Conditional blocks are most simply terminated with #endif, but #else and #elif *constant-expression* are also available for selecting one of two or more alternative text blocks.

The simplest use of #if is to "turn off" sections of an input file without editing them out:

```
#if 0This text is deleted from the input stream.#endif
```

Or, portions of the input file can be conditionally selected:

```
#define FLRAREA 1000   // other values used in other runs
#if FLRAREA <= 800
    CIDL language for small zones
#elif FLRAREA <= 1500
    CIDL language for medium zones
#else
    CIDL language for large zones
#endif
```

Note that if a set of #if … #elif … #elif conditionals does not contain an #else, it is possible for all lines to be excluded.

Finally, it is once again important to note that conditional directives *nest*, as shown in the following example (indentation is included for clarity only):

```
#if 0
    This text is NOT included.
    #if 1
        This text is NOT included.
    #endif
#else
    This text IS included.
#endif
```

4.4.5 Preprocessor examples

This section shows a few combined examples that demonstrate the preprocessor's capabilities.

The simplest use of macros is for run parameterization. For example, a base file is constructed that derives values from a macro named FLRAREA. Then multiple runs can be performed using #include:

```
// Base file
... various CIDL input ...

ZONE main
    znArea = FLRAREA
    znVol  = 8*FLRAREA
    znCAir = 2*FLRAREA ...
    ... various other CIDL input ...

RUN

CLEAR
```

The actual input file would look like this:

```
// Run with zone area = 500, 1000, and 2000 ft2
#define FLRAREA 500
#include "base."
#redefine FLRAREA 1000
#include "base."
#redefine FLRAREA 2000
#include "base."
```

Macros are also useful for encapsulating standard calculations. For example, most U-values must be entered *without* surface conductances, yet many tabulated U-values include the effects of the standard ASHRAE winter surface conductance of 6.00 Btuh/ft^2-^{o}F. A simple macro is very helpful:

```
#define UWinter(u)  ( 1/(1/(u)-1/6.00) )
```

This macro can be used whenever a U-value is required (e.g. SURFACE … sfU=UWinter(.11) …).

4.5 CIDL Statements

This section describes the general form of CIDL statements that define objects, assign values to the data members of objects, and initiate actions. The concepts of objects and the class hierarchy were introduced in Section 4.2. Information on statements for specific CIDL classes and their members is the subject of Section 5.

4.5.1 Object Statements

As we described in Section 4.3.1, the description of an object is introduced by a statement containing at least the class name, and usually your chosen name for the particular object. In addition, this section will describe several optional qualifiers and modifying clauses that permit defining similar objects without repeating all of the member details, and reopening a previously given object description to change or add to it.

Examples of the basic object-beginning statement:

```
ZONE "North";

METER "Electric - Cooling";

LAYER;
```

As described in Section 4.3.2, such a statement is followed by statements giving the object's member values or describing subobjects of the object. The object description ends when you begin another object that is

not of a subclass of the object, or when a member of an embedding (higher level) object previously begun is given, or when END is given.

4.5.1.1 Object Names

An object name consists of up to 63 characters. If you always enclose the name in quotation marks, punctuation and spaces may be used freely; if the name starts with a letter or dollar sign and consists only of letters, digits, underscore, and dollar sign, and is different from all of the words already defined in CIDL (as listed below in this section), you may omit the quotes. Capitalization, and Leading and trailing spaces and tabs, are always disregarded by CIDL. Names of 0 length, and names containing control characters (ASCII codes 0-31) are not allowed.

Examples of valid names that do not require quotes:

```
North
gas_meter
slab140E
```

The following object names are acceptable if always enclosed in quotes:

```
"Front Door"
"M L King Day"
"123"
"3.5-inch wall"
```

We suggest always quoting object names so you won't have to worry about disallowed words and characters.

Duplicate names result in error messages. Object names must be distinct between objects of the same class which are subobjects of the same object. For example, all ZONE names must be distinct, since all ZONE are subobjects of Top. It is permissible to have SURFACE with the same name in different ZONE – but it is a good idea to keep all of your object names distinct to minimize the chance of an accidental mismatch or a confusing message regarding some other error.

For some classes, such as ZONE a name is required for each object. This is because several other statements refer to specific ZONE and because a name is needed to identify ZONE in reports. For other classes, the name is optional. The specific statement descriptions in Section 5 say which names are required. We suggest always using object names even where not required; one reason is because they allow CSE to issue clearer error messages.

The following *reserved words will not work as object names unless enclosed in quotes*:

(this list needs to be assembled and typed in)

4.5.1.2 ALTER

ALTER is used to reopen a previously defined object when it is not possible or desired to give the entire description contiguously.

ALTER could be used if you wish to order the input in a special way. For example, SURFACE objects are subobjects of ZONE and are normally described with the ZONE they are part of. However, if you wanted to put all roofs together, you could use input of the general form:

```
ZONE "1";  . . .  (zone 1 description)
ZONE "2";  . . .
. . .
ALTER ZONE "1";                  // revert to specifying zone 1
    SURFACE "Roof1";  . . .      (describe roof of zone 1)
ALTER ZONE "2";
    SURFACE "Roof2";  . . .
```

ALTER can be used to facilitate making similar runs. For example, to evaluate the effect of a change in the size of a window, you might use:

```
ZONE "South";
    SURFACE "SouthWall";
    ...
        WINDOW "BigWindow";
            wnHeight = 6;  wnWidth = 20;
. . .
RUN;            // perform simulation and generate reports
// data from simulation is still present unless CLEAR given
ALTER ZONE "South";
    ALTER SURFACE "SouthWall";
        ALTER WINDOW "BigWindow";
            wnHeight = 4;  wnWidth = 12;  // make window smaller
RUN;            // perform simulation and print reports again
```

ALTER also lets you access the predefined "Primary" REPORTFILE and EXPORTFILE objects which will be described in Section 5:

```
ALTER REPORTFILE "Primary";     /* open description of object automatically
                                   supplied by CSE -- no other way to access */
    rfPageFmt = NO;             /* Turn off page headers and footers --
                                   not desired when reports are to be
                                   reviewed on screen. */
```

4.5.1.3 DELETE

DELETE followed by a class name and an object name removes the specified object, and any subobjects it has. You might do this after RUN when changing the data for a similar run (but to remove all data, CLEAR is handier), or you might use DELETE after COPYing (below) an object if the intent is to copy all but certain subobjects.

4.5.1.4 LIKE clause

LIKE lets you specify that an object being defined starts with the same member values as another object already defined. You then need give only those members that are different. For Example:

```
MATERIAL "SheetRock";           // half inch gypsum board
    matCond = .0925;            // conductivity per foot
    matSpHt = .26;              // specific heat
    matDens = 50;               // density
    matThk = 0'0.5;             // thickness 1/2 inch
MATERIAL "5/8 SheetRock" LIKE "SheetRock"; // 5/8" gypsum board
    matThk = 0'0.625;           // thickness 5/8 inch
    // other members same as "SheetRock", need not be repeated
```

The object named after LIKE must be already defined and must be of the same class as the new object.

LIKE copies only the member values; it does not copy any subobjects of the prototype object. For example, LIKEing a ZONE to a previously defined ZONE does not cause the new zone to contain the surfaces of the prototype ZONE If you want to duplicate the surfaces, use COPY instead of LIKE.

4.5.1.5 COPY clause

COPY lets you specify that the object being defined is the same as a previously defined object including all of the subobjects of that object. For example,

```
. . .
ZONE "West" COPY "North";
    DELETE WALL "East";
    ALTER WALL "South";
        sfExCnd = ambient;
```

Specifies a ZONE named "West" which is the same as ZONE North except that it does not contain a copy of West's East wall, and the South wall has ambient exposure.

4.5.1.6 USETYPE clause

USETYPE followed by the type name is used in creating an object of a type previously defined with DEFTYPE (next section). Example:

```
SURFACE "EastWall" USETYPE "IntWall";      // use interior wall TYPE (below)
    sfAzm = 90;                            // this wall faces to the East
    sfArea = 8 * 30;                       // area of each wall is different
    sfAdjZn = "East";                      // zone on other side of wall
```

Any differences from the type, and any required information not given in the type, must then be specified. Any member specified in the type may be respecified in the object unless FROZEN (Section 4.5.2.3) in the type (normally, a duplicate specification for a member results in an error message).

4.5.1.7 DEFTYPE

DEFTYPE is used to begin defining a TYPE for a class. When a TYPE is created, no object is created; rather, a partial or complete object description is stored for later use with DEFTYPE. TYPES facilitate creating multiple similar objects, as well as storing commonly used descriptions in a file to be #included in several different files, or to be altered for multiple runs in comparative studies without changing the including files. Example (boldface for emphasis only):

```
DEFTYPE SURFACE "BaseWall"                 // common characteristics of all walls
    sfType = WALL;                         // walls are walls, so say it once
    sfTilt = 90;                           // all our walls are vertical;
                                           //  but sfAzm varies, so it is not in TYPE.
    sfU = .83;                             // surf conductance; override if different
    sfModel = QUICK;

DEFTYPE SURFACE "ExtWall" USETYPE "BaseWall";
    sfExCnd = AMBIENT;                     // other side of wall is outdoors
    sfExAbs = 0.5;                         // member only needed for exterior walls

DEFTYPE SURFACE "IntWall" USETYPE "BaseWall";   // interior wall
    sfExCnd = ADJZN;                       // user must give sfAdjZn.
```

In a TYPE as much or as little of the description as desired may be given. Omitting normally-required members does not result in an error message in the type definition, though of course an error will occur at use if the member is not given there.

At use, member values specified in the TYPE can normally be re specified freely; to prevent this, "freeze" the desired member(s) in the type definition with

```
FREEZE *memberName*;
```

Alternately, if you wish to be sure the user of the TYPE enters a particular member even if it is normally optional, use

```
REQUIRE *memberName*
```

Sometimes in the TYPE definition, member(s) that you do not want defined are defined – for example, if the TYPE definition were itself initiated with a statement containing LIKE, COPY, or USETYPE. In such cases the member specification can be removed with

```
UNSET *memberName*;
```

4.5.1.8 END and ENDxxxx

END, optionally followed by an object name, can be used to unequivocally terminate an object. Further, as of July 1992 there is still available a specific word to terminate each type of object, such as ENDZONE to terminate a ZONE object. If the object name is given after END or ENDxxxx, an additional check is performed: if the name is not that of an object which has been begun and not terminated, an error message occurs. Generally, we have found it is not important to use END or ENDxxxx, especially since the member names in different classes are distinct.

4.5.2 Member Statements

As introduced in Section 4.3.1, statements which assign values to members are of the general form:

```
*memberName* = *expression*;
```

The specific member names for each class of objects are given in Section 5; many have already been shown in examples.

Depending on the member, the appropriate type for the expression giving the member value may be numeric (integer or floating point), string, object name, or multiple-choice. Expressions of all types will be described in detail in Section 4.6.

Each member also has its *variability* (also given in Section 5), or maximum acceptable *variation*. This is how often the expression for the value can change during the simulation – hourly, daily, monthly, no change (constant), etc. The "variations" were introduced in Section 4.3.3 and will be further detailed in Section 4.6.8.

Three special statements, UNSET, REQUIRE, and FREEZE, add flexibility in working with members.

4.5.2.1 UNSET

UNSET followed by a member name is used when it is desired to delete a member value previously given. UNSETing a member resets the object to the same internal state it was in before the member was originally given. This makes it legal to specify a new value for the member (normally, a duplicate specification results in an error message); if the member is required (as specified in Section 5), then an error message will occur if RUN is given without re specifying the member.

Situations where you really might want to specify a member, then later remove it, include:

- After a RUN command has completed one simulation run, if you wish to specify another simulation run without CLEARing and giving all the data again, you may need to UNSET some members of some objects in order to re specify them or because they need to be omitted from the new run. In this case, use ALTER(s) to reopen the object(s) before UNSETing.
- In defining a TYPE (Section 4.5.1.7), you may wish to make sure certain members are not specified so that the user must give them or omit them if desired. If the origin of the type (possibly a sequence of DEFTYPEs, LIKEs, and/or COPYs) has defined unwanted members, get rid of them with UNSET.

Note that UNSET is only for deleting *members* (names that would be followed with an = and a a value when being defined). To delete an entire *object*, use DELETE (Section 4.5.1.3).

4.5.2.2 REQUIRE

REQUIRE followed by a member name makes entry of that member mandatory if it was otherwise optional; it is useful in defining a TYPE (Section 4.5.1.7) when you desire to make sure the user enters a particular member, for example to be sure the TYPE is applied in the intended manner. REQUIRE by itself does not delete any previously entered value, so if the member already has a value, you will need to UNSET it. ?? *verify*

4.5.2.3 FREEZE

FREEZE followed by a member name makes it illegal to UNSET or redefine that member of the object. Note that FREEZE is unnecessary most of the time since CSE issues an error message for duplicate definitions without an intervening UNSET, unless the original definition came from a TYPE (Section 4.5.1.7). Situations where you might want to FREEZE one or more members include:

- When defining a TYPE (Section 4.5.1.7). Normally, the member values in a type are like defaults; they can be freely overridden by member specifications at each use. If you wish to insure a TYPE is used as intended, you may wish to FREEZE members to prevent accidental misuse.
- When your are defining objects for later use or for somebody else to use (perhaps in a file to be included) and you wish to guard against misuse, you may wish to FREEZE members. Of course, this is not foolproof, since there is at present no way to allow use of predefined objects or types without allowing access to the statements defining them.

4.5.3 Action Commands

CSE has two action commands, RUN and CLEAR.

4.5.3.1 RUN

RUN tells CSE to do an hourly simulation with the data now in memory, that is, the data given in the preceding part of the input file.

Note that CSE does NOT automatically run the simulator; an input file containing no RUN results in no simulation (you might nevertheless wish to submit an incomplete file to CSE to check for errors in the data already entered). The explicit RUN command also makes it possible to do multiple simulation runs in one session using a single input file.

When RUN is encountered in the input file, CSE checks the data. Many error messages involving inconsistencies between member values or missing required members occur at this time. If the data is good, CSE starts the simulation. When the simulation is complete and the reports have been output, CSE continues reading the input file. Statements after the first run can add to or change the data in preparation for another RUN. Note that the data for the first run is NOT automatically removed; if you wish to start over with complete specifications, use CLEAR after RUN.

4.5.3.2 CLEAR

CLEAR removes all input data (objects and all their members) from CSE memory. CLEAR is normally used after RUN, when you wish to perform another simulation run and wish to start clean. If CLEAR is not used, the objects from the prior run's input remain in memory and may be changed or added to produce the input data for the next simulation run.

4.6 Expressions

Probably CIDL's most powerful characteristic is its ability to accept expressions anywhere a single number, string, object name, or other value would be accepted. Preceding examples have shown the inputting

zone areas and volumes as numbers (some defined via preprocessor macros) with *'s between them to signify multiplication, to facilitate changes and avoid errors that might occur in manual arithmetic. Such expressions, where all operands are constants, are acceptable *anywhere* a constant of the same type would be allowed.

But for many object members, CSE accepts *live expressions* that *vary* according to time of day, weather, zone temperatures, etc. (etc., etc., etc.!). Live expressions permit simulation of many relationships without special-purpose features in the language. Live expressions support controlling setpoints, scheduling HVAC system operation, resetting air handler supply temperature according to outdoor temperature, and other necessary and foreseen functions without dedicated language features; they will also support many unforeseen user-generated functionalities that would otherwise be unavailable.

Additional expression flexibility is provided by the ability to access all of the input data and much of the internal data as operands in expressions (*probes*, Section 4.6.7).

As in a programming language, CSE expressions are constructed from operators and operands; unlike most programming languages, CSE determines how often an expression's operands change and automatically compute and store the value as often as necessary.

Expressions in which all operands are known when the statement is being decoded (for example, if all values are constants) are *always* allowed, because CIDL immediately evaluates them and presents the value to the rest of the program in the same manner as if a single number had been entered. *Most* members also accept expressions that can be evaluated as soon as the run's input is complete, for example expressions involving a reference to another member that has not been given yet. Expressions that vary during the run, say at hourly or daily intervals, are accepted by *many* members. The *variability* or maximum acceptable variation for each member is given in the descriptions in Section 5, and the *variation* of each non-constant expression component is given in its description in this section.

Interaction of expressions and the preprocessor: Generally, they don't interact. The preprocessor is a text processor which completes its work by including specified files, deleting sections under false #if's, remembering define definitions, replacing macro calls with the text of the definition, removing preprocessor directives from the text after interpreting them, etc., *then* the resulting character stream is analyzed by the CIDL statement compiler. However, the if statement takes an integer numeric expression argument. This expression is similar to those described here except that it can only use constant operands, since the preprocessor must evaluate it before deciding what text to feed to the CIDL statement compiler.

4.6.1 Expression Types

The type of value to which an expression must evaluate is specified in each member description (Section 5) or other context in which an expression can be used. Each expression may be a single constant or may be made up of operators and operands described in the rest of this section, so long as the result is the required type or can be converted to that type by CSE, and its variation is not too great for the context. The possible types are:

float

A real number (3.0, 5.34, -2., etc.). Approximately 7 digits are carried internally. If an int is given where a real is required, it is automatically converted.

int

An integer or whole number (-1, 0, 1, 2 etc.). If a real is given, an error may result, but we should change it to convert it (discarding any fractional part).

Boolean

Same as int; indicates that a 0 value will be interpreted as "false" and any non-0 value will be interpreted as "true".

string

A string of characters; for example, some text enclosed in quotes.

object name

Name of an object of a specified class. Differs from string in that the name need not be enclosed in quotes if it consists only of letters, digits, _, and $, begins with a non-digit, and is different from all reserved words now in or later added to the language (see Object Names, Section 4.5.1.1).

The object may be defined after it is referred to. An expression using conditional operators, functions, etc. may be used provided its value is known when the RUN action command is reached.; no members requiring object names accept values that vary during the simulation.

choice

One of several choices; a list of the acceptable values is given wherever a choice is required. The choices are usually listed in CAPITALS but may be entered in upper or lower case as desired. As with object names, quotes are allowed but not required.

Expressions may be used for choices, subject to the variability of the context.

date

May be entered as a 3-letter month abbreviation followed by an int for the day of the month, or an int for the Julian day of the year (February is assumed to have 28 days). Expressions may be used subject to variability limitations. Examples:

Jan 23 // January 23

23 // January 23

32 // February 1

These words are used in following descriptions of contexts that can accept more than one basic type:

numeric

float or int. When floats and ints are intermixed with the same operator or function, the result is float.

anyType

Any type; the result is the same type as the argument. If floats and ints are intermixed, the result is float. If strings and valid choice names are intermixed, the result is choice. Other mixtures of types are generally illegal, except in expressions for a few members that will accept either one of several choices or a numeric value.

The next section describes the syntax of constants of the various data types; then, we will describe the available operators, then other operand types such as system variables and built-in functions.

4.6.2 Constants

This section reviews how to enter ordinary non-varying numbers and other values.

int

optional - sign followed by digits. Don't use a decimal point if your intent is to give an int quantity – the decimal point indicates a float to CSE. Hexadecimal and Octal values may be given by prefixing the value with 0x and 0O respectively (yes, that really is a zero followed by an 'O').

float

optional - sign, digits and decimal point. Very large or small values can be entered by following the number with an "e" and a power of ten. Examples;

1.0 1. .1 -5534.6 123.e25 4.56e-23

The decimal point indicates a float as opposed to an int. Generally it doesn't matter as CSE converts ints to floats as required, but be careful when dividing: CSE interprets "2/3" as integer two divided by integer

3, which will produce an integer 0 before CSE notices any need to convert to float. If you mean .6666667, say 2./3, 2/3., or .6666667.

feet and inches

Feet and inches may be entered where a float number of feet is required by typing the feet (or a 0 if none), a single quote ', then the inches. (Actually this is an operator meaning "divide the following value by 12 and add it to the preceding value", so expressions can work with it.) Examples:

3'6 0'.5 (10+20)'(2+3)

string

"Text" – desired characters enclosed in double quotes. Maximum length 80 characters (make 132??). To put a " within the "'s, precede it with a backslash. Certain control codes can be represented with letters preceded with a backslash as follows:

\e escape

\t tab

\f form feed

\r carriage return

\n newline or line feed

object name

Same as string, or without quotes if name consists only of letters, digits, _, and $, begins with a non-digit, and is different from all reserved words now in or later added to the language (see Object Names, Section 4.5.1.1). Control character codes (ASCII 0-31) are not allowed.

choice

Same as string; quotes optional on choice words valid for the member. Capitalization does not matter.

date

Julian day of year (as int constant), or month abbreviation

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

followed by the int day of month. (Actually, the month names are operators implemented to add the starting day of the month to the following int quantity).

4.6.3 Operators

For *floats* and *ints*, CIDL recognizes a set of operators based closely on those found in the C programming language. The following table describes the available numeric operators. The operators are shown in the order of execution (precedence) when no ()'s are used to control the order of evaluation; thin lines separate operators of equal precedence.

Operator

Name

Notes and Examples

'

Feet-Inches Separator

a ' b yields a + b/12; thus 4'6 = 4.5.

+

Unary plus

The familiar "positive", as in +3. Does nothing; rarely used.

-

Unary minus

The familiar "minus", as in -3. -(-3) = +3 etc.

!

Logical NOT

Changes 0 to 1 and any non-0 value to 0. !0 = 1, !17 = 0.

~

One's complement

Complements each bit in an int value.

*

Multiplication

Multiplication, e.g. 3*4 = 12; 3.24*18.54 = 60.07

/

Division

Division, e.g. 4/2 = 2, 3.24/1.42 = 2.28. Integer division truncates toward 0 (e.g. 3/2 = 1, 3/-2 = -1, -3/2 = -1, 2/3 = 0) CAUTION!

%

Modulus

Yields the remainder after division, e.g. 7%2 = 1. The result has the same sign as the left operand (e.g.(-7)%2 = -1). % is defined for both integer and floating point operands (unlike ANSI C).

+

Addition

Yields the sum of the operands, e.g. 5+3 = 8

-

Subtraction

Yields the difference of the operands, e.g. 5-3 = 2

>>

Right shift

a >> b yields a shifted right b bit positions, e.g. 8>>2 = 2

<<

Left shift

a << b yields a shifted left b bit positions, e.g. 8<<2 = 32

<

Less than

a < b yields 1 if a is less than b, otherwise 0

<=

Less than or equal

a <= b yields 1 if a is less than or equal to b, otherwise 0

>=

Greater than or equal

a >= b yields 1 if a is greater than or equal to b, otherwise 0

>

Greater than

a > b yields 1 if a is greater than b, otherwise 0

==

Equal

a == b yields 1 if a is exactly (bit wise) equal to b, otherwise 0

!=

Not equal

a != b yields 1 if a is not equal to b, otherwise 0

&

Bitwise and

a & b yields the bitwise AND of the operands, e.g. 6 & 2 = 2.

^

Bitwise exclusive or

a ^ b yields the bitwise XOR of the operands, e.g. 6 ^ 2 = 4.

|

Bitwise inclusive or

a | b yields the bitwise IOR of the operands, e.g. 6 | 2 = 6.

&&

Logical AND

a && b yields 1 if both a and b are non-zero, otherwise 0. && guarantees left to right evaluation: if the first operand evaluates to 0, the second operand is not evaluated and the result is 0.

||

Logical OR

a || b yields 1 if either a or b is true (non-0), otherwise 0. || guarantees left to right evaluation: if the first operand evaluates to non-zero, the second operand in not evaluated and the result is 1.

? :

Conditional

a ? b : c yields b if a is true (non-0), otherwise c.

Dates are stored as *ints* (the value being the Julian day of the year), so all numeric operators could be used. The month abbreviations are implemented as operators that add the first day of the month to the following *int* value; CSE does not disallow their use in other numeric contexts.

For *strings*, *object names*, and *choices*, CIDL currently has no operators except the ?: conditional operator. A concatenation operator is being considered. Note, though, that the choose, choose1, select, and hourval functions described below work with strings, object names, and choice values as well as numbers.

4.6.4 System Variables

System Variables are built-in operands with useful values. To avoid confusion with other words, they begin with a $. Descriptions of the CSE system variables follow. Capitalization shown need not be matched. Most system variables change during a simulation run, resulting in the *variations* shown; they cannot be used where the context will not accept variation at least this fast. (Section 5 gives the *variability*, or maximum acceptable variation, for each object member.)

$dayOfYear < /td >< tdalign = left > Dayofyearofsimulation, 1 - 365; 1correspondstoJan - 1.(NotethatthisisnotthedayofthesimulationunlessbegDayisJan - 1.) < strong > Variation :< /strong > daily. < /td >< /tr >< trclass = even >< tdalign = left >$month

Month of year, 1 - 12. Variation: monthly.

$dayOfMonth < /td >< tdalign = left > Dayofmonth, 1 - 31. < strong > Variation < /strong >: daily. < /td >< /tr >< trclass = even >< tdalign = left >$hour

Hour of day, 1 - 24; 1 corresponds to midnight - 1 AM. Variation: hourly.

$dayOfWeek < /td >< tdalign = left > Dayofweek, 1 - 7; 1correspondstoSunday, 2toMonday, etc. < strong > Variation :< /strong > daily. < /td >< /tr >< trclass = even >< tdalign = left >$DOWH

Day of week 1-7 except 8 on every observed holiday. Variation: daily.

$isHoliday < /td >< tdalign = left > 1ondaysthataholidayisobserved(regardlessofthetruedateoftheholiday); 0onotherday strong > Variation < /strong >: daily. < /td >< /tr >< trclass = even >< tdalign = left >$isHoliTrue

1 on days that are the true date of a holiday, otherwise 0. Variation: daily.

$isWeHol < /td >< tdalign = left > 1onweekenddaysordaysthatareobservedasholidays. < strong > Variation :< /strong > daily. < /td >< /tr >< trclass = even >< tdalign = left >$isWeekend

1 on Saturday and Sunday, 0 on any day from Monday to Friday. Variation: daily.

$isWeekday < /td >< tdalign = left > 1onMondaythroughFriday, 0onSaturdayandSunday. < strong > Variation :< /strong > daily. < /td >< /tr >< trclass = even >< tdalign = left >$isBegWeek

1 for any day immediately following a weekend day or observed holiday that is neither a weekend day or an observed holiday. Variation: daily.

Weather variables: the following allow access to the current hour's weather conditions in you CSE expressions. Units of measure are shown in parentheses. All have **Variation**: hourly.

$radBeamSolar < /td >< tdalign = left > beamirradiance(onasun−trackingsurface)thishour(Btu/ft2) < /td >< /tr >< trclass = even >< tdalign = left >$radDiffSolar

diffuse irradiance (on horizontal surface) this hour (Btu/ft2)

$tDbOOutdoor < /td >< tdalign = left > drybulbtemperaturethishour(degreesF) < /td >< /tr >< trclass = even >< tdalign = left >$tWbOOutdoor

wetbulb temperature this hour (degrees F)

$wOOutdoor < /td >< tdalign = left > humidityratiothishour(lbH2O/lbdryair) < /td >< /tr >< trclass = even >< tdalign = left >$windDirDegWind

direction (compass degrees)

$windSpeedWind

speed (mph)

4.6.5 Built-in Functions

Built-in functions perform a number of useful scheduling and conditional operations in expressions. Built-in functions have the combined variation of their arguments; for *hourval*, the minimum result variation is hourly. For definitions of *numeric* and *anyType*, see Section 4.6.1.

brkt

Function

limits a value to be in a given range

Syntax

numeric brkt(numeric min, numeric val, numeric max)

Remark

If val is less than min, returns min; if val is greater than max, returns max; if val is in between, returns val.

Example

In an AIRHANDLER object, the following statement would specify a supply temperature equal to 130 minus the outdoor air temperature, but not less than 55 nor greater than 80:

```
ahTsSp = brkt( 55, 130 - $tDbO, 80);
```

This would produce a 55-degree setpoint in hot weather, an 80-degree setpoint in cold weather, and a transition from 55 to 70 as the outdoor temperature moved from 75 to 50.

fix

Function

converts float to int

Syntax

int fix(float val)

Remark

val is converted to int by truncation – fix(1.3) and fix(1.99) both return 1. fix(-4.4) returns -4.

toFloat

Function	converts *int* to *float*
Syntax	*float* **toFloat**(*int val*)

min

Function

returns the lowest quantity from a list of values.

Syntax

numeric min(numeric value1, numeric value2, … numeric valuen)

Remark

there can be any number of arguments separated by commas; if floats and ints are intermixed, the result is float.

max

Function

returns the highest quantity from a list of values.

Syntax

numeric max (numeric value1, numeric value2, … numeric valuen)

choose

Function

returns the nth value from a list. If arg0 is 0, value0 is returned; for 1, value1 is returned, etc.

Syntax

anyType choose (int arg0, anyType value0, anyType value1, … anyType valuen) or anyType choose (int arg0, anyType value0, … anyType valuen, default valueDef)

Remarks

Any number of value arguments may be given. If default and another value is given, this value will be used if arg0 is less than 0 or too large; otherwise, an error will occur.

choose1

Function

same as choose except arg0 is 1-based. Choose1 returns the second argument value1 for arg0 = 1, the third argument value2 when arg0 = 2, etc.

Syntax

anyType choose1 (int arg0, anyType value1, anyType value2, … anyType valuen) or anyType choose1 (int arg0, anyType value1, … anyType valuen, default valueDef)

Remarks

choose1 is a function that is well suited for use with daily system variables. For example, if a user wanted to denote different values for different days of the week, the following use of choose1 could be implemented:

tuTC = choose1($dayOfWeek, MonTemp, TueTemp, …)

Note that for hourly data, the hourval function would be a better choice, because it doesn't require the explicit declaration of the $hour system variable.

select

Function

contains Boolean-value pairs; returns the value associated with the first Boolean that evaluates to true (non-0).

Syntax

anyType (Boolean arg1, anyType value1, Boolean arg2, anyType value2, … default anyType) (the default part is optional)

Remark

select is a function that simulates if-then logic during simulation (for people familiar with C, it works much like a series of imbedded conditionals: (a?b:(a?b:c))).

Examples

Select can be used to simulate a dynamic (run-time) if-else statement:

gnPower = select($isHoliday, HD_GAIN, //if$(isHoliday)

default WD_GAIN) // else

This technique can be combined with other functions to schedule items on a hourly and daily basis. For example, an internal gain that has different schedules for holidays, weekdays, and weekends could be defined as follows:

// 24-hour lighting power schedules for weekend, weekday, holiday:

#define WE_LIGHT hourval(.024, .022, .021, .021, .021, . 026, </code>

.038, .059, .056, .060, .059, .046, </code>

.045, .005, .005, .005, .057, .064, </code>

.064, .052, .050, .055, .044, .027)

#define WD_LIGHT hourval(.024, .022, .021, .021, .021, . 026, </code>

.038, .059, .056, .060, .059, .046, </code>

.045, .005, .005, .005, .057, .064, </code>

.064, .052, .050, .055, .044, .027)

#define HD_LIGHT hourval(.024, .022, .021, .021, .021, . 026, </code>

.038, .059, .056, .060, .059, .046, </code>

.045, .005, .500, .005, .057, .064, </code>

.064, .052, .050, .055, .044, .027)

// set power member of zone's GAIN object for lighting

gnPower = BTU_Elec(ZAREA*0.1)* // .1 kW/ft2 ti mes…

select($isHoliday, HD_LIGHT, //Holidays < /code >< /td >< /tr >< trclass = even >< td >< /td >< tdalign = left > ăăăăăăă < code >isWeekend, WE_LIGHT, // Saturday & Sunday

default WD_LIGHT); // Week Days

In the above, three subexpressions using hourval (next) are first defined as macros, for ease of reading and later change. Then, gnPower (the power member of a GAIN object) is set, using select to choose the appropriate one of the three hourval calls for the type of day. The expression for gnPower is a live expression with hourly variation, that is, CSE will evaluate it an set gnPower to the latest value each hour of the simulation. The variation comes from hourval, which varies hourly (also, $isHoliday and $isWeekend vary daily, but the faster variation determines the variation of the result).

hourval

Function

from a list of 24 values, returns the value corresponding to the hour of day.

Syntax

anyType hourval (anyType value1, anyType value2, … anyType value24)

anyType hourval (anyType value1, anyType value2, … default anyType)

Remark

hourval is evaluated at runtime and uses the hour of the day being simulated to choose the corresponding value from the 24 suppplied values.

If less than 24 value arguments are given, default and another value (or expression) should be supplied to be used for hours not explicitly specified.

Example

see select, just above.

abs

Function	converts numeric to its absolute value
Syntax	numeric **abs**(numeric val)

sqrt

Function

Calculates and returns the positive square root of val (val must be 0).

Syntax

float sqrt (float val)

exp

Function

Calculates and returns the exponential of val (= eval)

Syntax

float exp(float val)

logE

Function

Calculates and returns the base e logarithm of val (val must be 0).

Syntax

float logE(float val)

log10

Function

Calculates and returns the base 10 logarithm of val (val must be 0).

Syntax

float log10(float val)

sin

Function	Calculates and returns the sine of *val* (val in radians)
Syntax	*float* **sin**(*float val*)

sind

Function	Calculates and returns the sine of *val* (val in degrees)
Syntax	*float* **sind**(*float val*)

asin

Function	Calculates and returns (in radians) the arcsine of *val*
Syntax	*float* **asin**(*float val*)

asind

Function	Calculates and returns (in degrees) the arcsine of *val*
Syntax	*float* **asind**(*float val*)

cos

Function	Calculates and returns the cosine of *val* (val in radians)
Syntax	*float* **cos**(*float val*)

cosd

Function	Calculates and returns the cosine of *val* (val in degrees)
Syntax	*float* **cosd**(*float val*)

acos

Function	Calculates and returns (in radians) the arccosine of *val*
Syntax	*float* **acos**(*float val*)

acosd

Function	Calculates and returns (in degrees) the arccosine of *val*
Syntax	*float* **acosd**(*float val*)

tan

Function

Calculates and returns the tangent of val (val in radians)

Syntax

float tan(float val)

tand

Function

Calculates and returns the tangent of val (val in degrees)

Syntax

float tand(float val)

atan

Function	Calculates and returns (in radians) the arctangent of *val*
Syntax	*float* **atan**(*float val*)

atand

Function	Calculates and returns (in degrees) the arctangent of *val*
Syntax	*float* **atand**(*float val*)

atan2

Function

Calculates and returns (in radians) the arctangent of y/x (handling x = 0)

Syntax

float atan2(float y, float x)

atan2d

Function

Calculates and returns (in degrees) the arctangent of y/x (handling x = 0)

Syntax

float atan2d(float y, float x)

pow

Function

Calculates and returns val raised to the xth power (= valx). val and x cannot both be 0. If val < 0, x must be integral.

Syntax

float pow(float val, numeric x)

enthalpy

Function

Returns enthalpy of moist air (Btu/lb) for dry bulb temperature (F) and humidity ratio (lb/lb)

Syntax

float enthalpy(float tDb, float w)

wFromDbWb

Function

Returns humidity ratio (lb/lb) of moist air from dry bulb and wet bulb temperatures (F)

Syntax

float wFromDbWb(float tDb, float tWb)

wFromDbRh

Function

Returns humidity ratio (lb/lb) of moist air from dry bulb temperature (F) and relative humidity (0 – 1)

Syntax

float wFromDbWb(float tDb, float rh)

import

Function	Returns *float* read from import file.
Syntax	*float* **import**(*?? arguments to be documented*)

importStr

Function	Returns *string* read from import file.
Syntax	*string* **importStr**(*?? arguments to be documented*)

contin

Function

Returns continuous control value, e.g. for lighting control

Syntax

float contin(float mpf, float mlf, float sp, float val)

Remark

contin is evaluated at runtime and returns a value in the range 0 – 1 ???

Example

–

stepped

Function

Returns stepped reverse-acting control value, e.g. for lighting control

Syntax

float stepped(int nsteps, float sp, float val)

Remark

stepped is evaluated at runtime and returns a value in the range 0 – 1. If val <= 0, 1 is returned; if val >= sp, 0 is returned; otherwise, a stepped intermediate value is returned (see example)

example:

stepped(3, 12, val) returns

val	*result*
val $<$ 4	1
$4 \leq$ val $<$ 8	.667
$8 \leq$ val $<$ 12	.333
val $\geq$ 12	0

4.6.6 User-defined Functions

User defined functions have the format:

```
type FUNCTION name ( arg decls ) = expr ;
```

Type indicates the type of value the function returns, and can be:

INTEGER
FLOAT
STRING
DOY (day of year date using month name and day; actually same as integer).

Arg decls indicates zero or more comma-separated argument declarations, each consisting of a *type* (as above) and the name used for the argument in *expr*.

Expr is an expression of (or convertible to) *type*.

The tradeoffs between using a user-defined function and a preprocessor macro (#define) include:

1. Function may be slightly slower, because its code is always kept separate and called, while the macro expansion is inserted directly in the input text, resulting in inline code.
2. Function may use less memory, because only one copy of it is stored no matter how many times it is called.
3. Type checking: the declared types of the function and its arguments allow CSE to perform additional checks.

Note that while macros require line-splicing ("\")to extend over one line, functions do not require it:

```
// Function returning number of days in ith month of year:
DOY FUNCTION MonthLU (integer i) = choose1 ( i , Jan 31, Feb 28, Mar 31,
                                               Apr 30, May 31, Jun 30,
                                               Jul 31, Aug 31, Sep 30,
                                               Oct 31, Nov 30, Dec 31 ) ;
// Equivalent preprocessor macro:
#define MonthLU (i) = choose1 ( i , Jan 31, Feb 28, Mar 31,  \
                                   Apr 30, May 31, Jun 30,  \
                                   Jul 31, Aug 31, Sep 30,  \
                                   Oct 31, Nov 30, Dec 31 ) ;
```

4.6.7 Probes

Probes provide a universal means of referencing data within the simulator. Probes permit using the inputtable members of each object, as described in Section 5, as operands in expressions. In addition, most internal members can be probed; we will describe how to find their names shortly.

Three general ways of using probes are:

1. During input, to implement things like "make this window's width equal to 10% of the zone floor area" by using the zone's floor area in an expression:

   ```
   wnWidth = @zone[1].znArea * 0.1;
   ```

Here "@zone[1].znArea" is the probe.

2. During simulation. Probing during simulation, to make inputs be functions of conditions in the building or HVAC systems, is limited because most of the members of interest are updated *after* CSE has evaluated the user's expressions for the subhour or other time interval – this is logically necessary since the expressions are inputs. (An exception is the weather data, but this is also available through system variables such as $tDbO.)

However, a number of *prior subhour* values are available for probing, making it possible to implement relationships like "the local heat output of this terminal is 1000 Btuh if the zone temperature last subhour was below 65, else 500":

```
tuMnLh = @znres["North"].S.prior.tAir < 65 ? 1000 : 500;
```

3. For output reports, allowing arbitrary data to be reported at subhourly, hourly, daily, monthly, or annual intervals. The REPORT class description in Section 5 describes the user-defined report type (UDT), for which you write the expression for the value to be reported. With probes, you can thus report almost any datum within CSE – not just those values chosen for reporting when the program was designed. Even values calculated during the current subhour simulation can be probed and reported, because expressions for reports are evaluated after the subhour's calculations are performed.

Examples:

```
colVal = @airHandler["Hot"].ts;      // report air handler supply temp
colVal = @terminal[NorthHot].cz;     // terminal air flow to zone (Btuh/F)
```

The general form of a probe is

@ *className* [*objName*] . *member*

The initial @ is always necessary. And don't miss the period after the].

className is the CLASS being probed

objName

is the name of the specific object of the class; alternately, a numeric subscript is allowed. Generally, the numbers correspond to the objects in the order created. [objName] can be omitted for the TOP class, which has only one member, Top.

member

is the name of the particular member being probed. This must be exactly correct. For some inputtable members, the probe name is not the same as the input name given in Section 5, and there are many probe-able members not described in Section 55.

How do you find out what the probe-able member names are? CSE will display the a list of the latest class and member names if invoked with the -p switch. Use the command line

```
CSE -p >probes.txt
```

to put the displayed information into the file PROBES.TXT, then print the file or examine it with a text editor.

A portion of the −p output looks like:

```
@exportCol[1..].          I   R                    owner: export
                  name    I   R   string             constant
               colHead    I   R   string             input time
                colGap    I   R   integer number     input time
                colWid    I   R   integer number     input time
                colDec    I   R   integer number     input time
               colJust    I   R   integer number     constant
                colVal    I   R   un-probe-able      end of each subhour
                nxColi    I   R   integer number     constant

@holiday[1..].            I
                  name    I       string             constant
           hdDateTrue     I       integer number     constant
            hdDateObs     I       integer number     constant
           hdOnMonday     I       integer number     constant
```

In the above "exportCol" and "holiday" are class names, and "name", "colHead", "colGap", . . . are member names for class exportCol. Some members have multiple names separated by .'s, or they may contain an additional subscript. To probe one of these, type all of the names and punctuation exactly as shown (except capitalization may differ); if an additional subscript is shown, give a number in the specified range. An "I" designates an "input" parameter, an R means "runtime" parameter. The "owner" is the class of which this class is a subclass.

The data type and variation of each member is also shown. Note that *variation*, or how often the member changes, is shown here. (*Variability*, or how often an expression assigned to the member may change, is given for the inputtable members in Section 5.) Members for which an "end of" variation is shown can be probed only for use in reports. A name described as "un-probe-able" is a structure or something not convertible to an integer, float, or string.

surface[].sgdist[].f[]: f[0] is winter solar coupling fraction; f[1] is summer.

4.6.8 Variation Frequencies Revisited

At risk of beating the topic to death, we're going to review once more the frequencies with which a CSE value can change (*variations)*, with some comments on the corresponding *variabilities*.

subhourly

changes in each "subhour" used in simulation. Subhours are commonly 15-minute intervals for models using znModel=CNE or 2-minute intervals for CSE znModels.

hourly

changes every simulated hour. The simulated weather and many other aspects of the simulation change hourly; it is customary to schedule setpoint changes, HVAC system operation, etc. in whole hours.

daily

changes at each simulated midnite.

monthly

changes between simulated months.

monthly-hourly, or "hourly on first day of each month"

changes once an hour on the first day of each month; the 24 hourly values from the first day of the month are used for the rest of the month. This variation and variability is used for data dependent on the sun's position, to save calculation time over computing it every hour of every day.

run start time

value is derived from other inputs before simulation begins, then does not change.

Members that cannot change during the simulation but which are not needed to derive other values before the simulation begins have "run start time" variability.

input time

value is known before CSE starts to check data and derive "run start time" values.

Expressions with "input time" variation may be used in many members that cannot accept any variation during the run. Many members documented in Section 5 as having "constant" variability may actually accept expressions with "input time" variation; to find out, try it: set the member to an expression containing a proposed probe and see if an error message results.

"Input time" differs from "constant" in that it includes object names (forward references are allowed, and resolved just before other data checks) and probes that are forward references to constant values.

constant

does not vary. But a "constant" member of a class denoted as R (with no I) in the probes report produced by CSE -p is actually not available until run start time.

Also there are end-of varieties of all of the above; these are values computed during simulation: end of each hour, end of run, etc. Such values may be reported (using a probe in a UDT report), but will produce an error message if probed in an expression for an input member value.

4.6.9 Probes: Issues and Cautions

5 Input Data

This section describes the input for each CSE class (object type). The general concepts used here are described in Section 5. For each object you wish to define, the usual input consists of the class name, your name for the particular object (usually), and zero or more member value statements of the form *name=expression*. The name of each subsection of this section is a class name HOLIDAY MATERIAL CONSTRUCTION etc.). The object name, if given, follows the class name; it is the first thing in each description (hdName, matName, conName, etc.). Exception: no statement is used to create or begin the predefined top-level object "Top" (of class TOP its members are given without introduction.

After the object name, each member's description is introduced with a line of the form *name=type*. *Type* indicates the appropriate expression type for the value:

- *float*

- *int*
- *string*
- _____*name* (object name for specified type of object)
- *choice*
- *date*

These types discussed in Section 4.6.1.

Each member's description continues with a table of the form

Units

Legal Range

Default

Required

Variability

ft2

x > 0

wnHeight * wnWidth

No

constant

Units

units of measure (lb., ft, Btu, etc.) where applicable

Legal Range

limits of valid range for numeric inputs; valid choices for choice members, etc.

Default

value assumed if member not given; applicable only if not required

Required

YES if you must give this member

Variability

how often the given expression can change: hourly, daily, etc. See Sections 4.3.3, 4.5.2, and 4.6.8.

5.1 TOP Members

The top-level data items TOP members) control the simulation process or contain data that applies to the modeled building as a whole. No statement is used to begin or create the TOP object; these statements can be given anywhere in the input (they do, however, terminate any other objects being specified – ZONE REPORT etc.).

5.1.1 TOP General Data Items

doMainSim=*choice*

Specifies whether the simulation is performed when a Run command is encountered. See also doAutoSize.

Units	Legal Range	Default	Required	Variability

Units	Legal Range	Default	Required	Variability
	NO,YES	YES	No	constant

begDay=*date*

Date specifying the beginning day of the simulation performed when a Run command is encountered. See further discussion under endDay (next).

Units	Legal Range	Default	Required	Variability
	date	Jan 1	No	constant

endDay=*date*

Date specifying the ending day of the simulation performed when a Run command is encountered.

The program simulates 365 days at most. If begDay and endDay are the same, 1 day is simulated. If begDay precedes endDay in calendar sequence, the simulation is performed normally and covers begDay through endDay inclusive. If begDay follows endDay in calendar sequence, the simulation is performed across the year end, with Jan 1 immediately following Dec 31.

Units	Legal Range	Default	Required	Variability
	date	Dec 31	No	constant

jan1DoW=*choice*

Day of week on which January 1 falls.

Units

Legal Range

Default

Required

Variability

SUN MON TUE WED THU FRI SAT

THU

No

constant

workDayMask=*int* TODO

Units	Legal Range	Default	Required	Variability
		Mon-fri?	No	constant

wuDays=*int*

Number of "warm-up" days used to initialize the simulator. Simulator initialization is required because thermal mass temperatures are set to arbitrary values at the beginning of the simulation. Actual mass

temperatures must be established through simulation of a few days before thermal loads are accumulated. Heavier buildings require more warm-up; the default values are adequate for conventional construction.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	7	No	constant

nSubSteps=*int*

Number of subhour steps used per hour in the simulation. 4 is the time-honored value for models using CNE zones. A value of 30 is typically for CSE zone models.

Units	Legal Range	Default	Required	Variability
	$x > 0$	4	No	constant

tol=*float*

Endtest convergence tolerance for internal iteration in CNE models (no effect for CSE models) Small values for the tolerance cause more accurate simulations but slower performance. The user may wish to use a high number during the initial design process (to quicken the runs) and then lower the tolerance for the final design (for better accuracy). Values other than .001 have not been explored.

Units	Legal Range	Default	Required	Variability
	$x > 0$	.001	No	constant

humTolF=*float*

Specifies the convergence tolerance for humidity calculations in CNE models (no effect in for CSE models), relative to the tolerance for temperature calculations. A value of .0001 says that a humidity difference of .0001 is about as significant as a temperature difference of one degree. Note that this is multiplied internally by "tol"; to make an overall change in tolerances, change "tol" only.

Units	Legal Range	Default	Required	Variability
	$x > 0$	.0001	No	

ebTolMon=*float*

Monthly energy balance error tolerance for internal consistency checks. Smaller values are used for testing the internal consistency of the simulator; values somewhat larger than the default may be used to avoid error messages when it is desired to continue working despite a moderate degree of internal inconsistency.

Units	Legal Range	Default	Required	Variability
	$x > 0$	0.0001	No	constant

ebTolDay=*float*

Daily energy balance error tolerance.

Units	Legal Range	Default	Required	Variability
	$x > 0$	0.0001	No	constant

ebTolHour=*float*

Hourly energy balance error tolerance.

Units	Legal Range	Default	Required	Variability
	$x > 0$	0.0001	No	constant

ebTolSubhr=*float*

Sub-hourly energy balance error tolerance.

Units	Legal Range	Default	Required	Variability
	$x > 0$	0.0001	No	constant

humMeth=*choice*

Developmental zone humidity computation method choice for CNE models (no effect for CSE models).

ROB

Rob's backward difference method. Works well within limitations of backward difference approach.

PHIL

Phil's central difference method. Should be better if perfected, but initialization at air handler startup is unresolved, and ringing has been observed.

Units	Legal Range	Default	Required	Variability
	ROB, PHIL	ROB	No	constant

dflExH=*float*

Default exterior surface (air film) conductance used for opaque and glazed surfaces exposed to ambient conditions in the absence of explicit specification.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	2.64	No	constant

bldgAzm=*float*

Reference compass azimuth (0 = north, 90 = east, etc.). All zone orientations (and therefore surface orientations) are relative to this value, so the entire building can be rotated by changing bldgAzm only. If a value outside the range $0° \leq x < 360°$ is given, it is normalized to that range.

Units	Legal Range	Default	Required	Variability
° (degrees)	unrestricted	0	No	constant

elevation=*float*

Elevation of the building site. Used internally for the computation of barometric pressure and air density of the location.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0 (sea level)	No	constant

runTitle=*string*

Run title for the simulation. Appears in report footers, export headers, and in the title lines to the INP, LOG, and ERR built-in reports (these appear by default in the primary report file; the ERR report also appears in the error message file, if one is created).

Units	Legal Range	Default	Required	Variability
	63 characters	blank (no title)	No	constant

runSerial=*int*

Run serial number for the simulation. Increments on each run in a session; appears in report footers.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 999$	0	No	constant

5.1.2 TOP Daylight Saving Time Items

Daylight savings starts by default at 2:00 a.m. of the second Sunday in March. Internally, hour 3 (2:00-3:00 a.m.) is skipped and reports for this day show only 23 hours. Daylight savings ends by default at 2:00 a.m. of the first Sunday of November; for this day 25 hours are shown on reports. CSE fetches weather data using standard time but uses daylight savings time to calculate variable expressions (and thus all schedules).

DT=*choice*

Whether Daylight Savings Time is to be used for the current run.

Units	Legal Range	Default	Required	Variability
	YES, NO	YES	No	constant

DTbegDay=*date*

Start day for daylight saving time (assuming DT=Yes)

Units	Legal Range	Default	Required	Variability
	date	*second Sunday in March*	No	constant

DTendDay=*date*

End day for daylight saving time (assuming DT=Yes)

Units	Legal Range	Default	Required	Variability
	date	*first Sunday in November*	No	constant

5.1.3 TOP Weather Data Items

The following system variables (4.6.4) are determined from the weather file for each simulated hour:

$radBeam < /td >< td > beamirradianceontrackingsurface(integralforhour, Btu/ft < sup > 2 < /sup >). < /td >< /tr >< trclass = even >< tdalign = left >$radDiff

diffuse irradiance on a horizontal surface (integral for hour, Btu/ft2).

$tDbO < /td >< td > drybulbtemp(< sup > o < /sup > F). < /td >< /tr >< trclass = even >< tdalign = left >$tWbO

wet bulb temp (oF).

$wO < /td >< td > humidityratio < /td >< /tr >< trclass = even >< tdalign = left >$windDirDeg

wind direction (degrees, NOT RADIANS; 0=N, 90=E).

$windSpeed

wind speed (mph).

The following are the terms determined from the weather file for internal use, and can be referenced with the probes shown.

```
@Top.depressWbWet bulb depression (F).

@Top.windSpeedSquaredWind speed squared (mph2).
```

wfName=*string*

Weather file path name for simulation. The file should be in the current directory, in the directory CSE.EXE was read from, or in a directory on the operating system PATH. (?? Temporarily, backslash (\) characters in path names must be doubled to work properly (e.g. "\\wthr\\cz01.cec"). This will be corrected.).

Units	Legal Range	Default	Required	Variability
	file name,path optional		Yes	constant

skyModel=*choice*

Selects sky model used to determine relative amounts of direct and diffuse irradiance.

ISOTROPIC	traditional isotropic sky model
ANISOTROPIC	Hay anisotropic model

Units	Legal Range	Default	Required	Variability
	ISOTROPIC ANISOTROPIC	ANISOTROPIC	No	constant

The reference temperature and humidity are used to calculate a humidity ratio assumed in air specific heat calculations. The small effect of changing humidity on the specific heat of air is generally ignored in the interests of speed, but the user can control the humidity whose specific heat is used through the refTemp and refRH inputs.

refTemp=*float*

Reference temperature (see above paragraph).

Units	Legal Range	Default	Required	Variability
°F	$x \geq 0$	60°	No	constant

refRH=*float*

Reference relative humidity (see above).

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.6	No	constant

grndRefl=*float*

Global ground reflectivity, used except where other value specified with sfGrndRefl or wnGrndRefl. This reflectivity is used in computing the reflected beam and diffuse radiation reaching the surface in question.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.2	No	Monthly-Hourly

The following values modify weather file data, permitting varying the simulation without making up special weather files. For example, to simulate without the effects of wind, use windF = 0; to halve the effects of diffuse solar radiation, use radDiffF = 0.5. Note that the default values for windSpeedMin and windF result in modification of weather file wind values unless other values are specified.

windSpeedMin=*float*

Minimum value for wind speed

Units	Legal Range	Default	Required	Variability
mph	$x \geq 0$	0.5	No	constant

windF=*float*

Wind Factor: multiplier for wind speeds read from weather file. windF is applied *after* windSpeedMin. Note that windF does *not* effect infiltration rates calculated by the Sherman-Grimsrud model (see e.g. ZONE.infELA). However, windF does modify AirNet flows (see IZXFER

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	0.25	No	constant

terrainClass=*int*

Specifies characteristics of ground terrain in the project region.

1

ocean or other body of water with at least 5 km unrestriced expanse

2

flat terrain with some isolated obstacles (buildings or trees well separated)

3

rural areas with low buildings, trees, etc.

4

urban, industrial, or forest areas

5

center of large city

Units	Legal Range	Default	Required	Variability
	$1 \leq x \leq 5$	4	No	constant

radBeamF=*float*

Multiplier for direct normal (beam) irradiance

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	1	No	constant

radDiffF=*float*

Multiplier for diffuse horizonal irradiance.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	1	No	constant

5.1.4 TOP Report Data Items

These items are used in page-formatted report output files. See REPORTFILE Section 5.245.21, and REPORT Section 5.25.

repHdrL=*string*

Report left header. Appears at the upper left of each report page unless page formatting (rfPageFmt) is OFF. If combined length of repHdrL and repHdrR is too large for the page width, one or both will be truncated.

Units	Legal Range	Default	Required	Variability
		blank	No	constant??

repHdrR=*string*

Report right header. Appears at the upper right of each report page unless page formatting (rfPageFmt) is OFF. If combined length of repHdrL and repHdrR is too large for the page width, one or both will be truncated.

Units	Legal Range	Default	Required	Variability
		blank(no right header)	No	constant??

repLPP=*int*

Total lines per page to be assumed for reports. Number of lines used for text (including headers and footers)

is repLPP - repTopM - repBotM.

Units	Legal Range	Default	Required	Variability
lines	$x \geq 50$	66	No	constant??

repTopM=*int*

Number of lines to be skipped at the top of each report page (prior to header).

Units	Legal Range	Default	Required	Variability
lines	$0 \geq x \geq 12$	3	No	constant

repBotM=*int*

Number of lines reserved at the bottom of each report page. repBotM determines the position of the footer on the page (blank lines after the footer are not actually written).

Units	Legal Range	Default	Required	Variability
lines	$0 \leq x \leq 12$	3	No	constant

repCPL=*int*

Characters per line for report headers and footers, user defined reports, and error messages. CSE writes simple ASCII files and assumes a fixed (not proportional) spaced printer font. Many of the built-in reports now (July 1992) assume a line width of 132 columns.

Units	Legal Range	Default	Required	Variability
characters	$78 \leq x \leq 132$	78	No	constant

repTestPfx=*string*

Report test prefix. Appears at beginning of report lines that are expected to differ from prior runs. This is useful for "hiding" lines from text comparison utilities in automated testing schemes. Note: the value specified with command line -x takes precedence over this input.

Units	Legal Range	Default	Required	Variability
		blank	No	constant??

5.1.5 TOP Autosizing

doAutoSize=*choice*

Controls invocation of autosizing step prior to simulation.

Units	Legal Range	Default	Required	Variability
	YES, NO	NO	No	constant

auszTol=*float*

Units	Legal Range	Default	Required	Variability
		.01	No	constant

heatDsTDbO=*float*

Units	Legal Range	Default	Required	Variability
°F		0	No	hourly

heatDsTWbO=*float*

Units	Legal Range	Default	Required	Variability
F		0	No	hourly

coolDsMo=*??*

Units	Legal Range	Default	Required	Variability
F		.01	No	hourly

5.1.6 TOP Debug Reporting

verbose=*int*

Controls verbosity of screen remarks. Most possible remarks are generated during autosizing of CNE models. Little or no effect in CSE models. TODO: document options

Units	Legal Range	Default	Required	Variability
	0 – 5 ?	1	No	constant

The following dbgPrintMask values provide bitwise control of addition of semi-formated internal results to the run report file. The values and format of debugging reports are modified as required for testing purposes.

dbgPrintMaskC=*int*

Constant portion of debug reporting control.

Units	Legal Range	Default	Required	Variability
		0	No	constant

dbgPrintMask=*int*

Hourly portion of debug reporting control (generally an expression that evaluates to non-0 only on days or hours of interest).

Units	Legal Range	Default	Required	Variability
		0	No	hourly

5.2 HOLIDAY

HOLIDAY objects define holidays. Holidays have no inherent effect, but input expressions can test for holidays via the $DOWH, $isHoliday, $isHoliTrue, $isWeHol, and $isBegWeek system variables (4.6.4).

Examples and the list of default holidays are given after the member descriptions.

hdName

Name of holiday: must follow the word HOLIDAY

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

A holiday may be specified by date or via a rule such as "Fourth Thursday in November". To specify by date, give hdDateTrue, and also hdDateObs or hdOnMonday if desired. To specify by rule, give all three of hdCase, hdMon, and hdDow.

hdDateTrue=*date*

The true date of a holiday, even if not celebrated on that day.

Units	Legal Range	Default	Required	Variability
	date	*blank*	No	constant

hdDateObs=*date*

The date that a holiday will be observed. Allowed only if hdDateTrue given and hdOnMonday not given.

Units	Legal Range	Default	Required	Variability
	date	*hdDateTrue*	No	constant

hdOnMonday=*choice*

If YES, holiday is observed on the following Monday if the true date falls on a weekend. Allowed only if hdDateTrue given and hdDateObs not given.

Note: there is no provision to celebrate a holiday that falls on a Saturday on *Friday* (as July 4 was celebrated in 1992).

Units	Legal Range	Default	Required	Variability
	YES NO	YES	No	constant

hdCase=*choice*

Week of the month that the holiday is observed. hdCase, hdMon, and hdDow may be given only if hdDateTrue, hdDateObs, and hdOnMonday are not given.

Units	Legal Range	Default	Required	Variability
	FIRST SECOND THIRD FOURTH LAST	FIRST	No	constant

hdMon=*choice*

Month that the holiday is observed.

Unit s

Legal Range

Defau lt

Required

Variabi lity

JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

none

required if hdCase given

constant

hdDow=*choice*

Day of the week that the holiday is observed.

Unit s

Legal Range

Defau lt

Required

Variabi lity

SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

MONDAY

required if hdCase given

constant

endHoliday

Indicates the end of the holiday definition. Alternatively, the end of the holiday definition can be indicated by "END" or simply by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

Examples of valid HOLIDAY object specifications:

- Holiday on May first, observed date moved to following Monday if the first falls on a weekend (hdOnMonday defaults Yes).

```
HOLIDAY MAYDAY;
    hdDateTrue = May 1;
```

- Holiday on May 1, observed on May 3.

```
HOLIDAY MAYDAY;
    hdDateTrue = May 1;
    hdDateObs  = May 3;
```

- Holiday observed on May 1 even if on a weekend.

```
HOLIDAY MAYDAY;
    hdDateTrue = May 1;
```

```
    hdOnMonday = No;
```

- Holiday observed on Wednesday of third week of March

```
HOLIDAY HYPOTHET;
    hdCase = third;
    hdDow  = Wed;
    hdMon  = MAR
```

As with reports, Holidays are automatically generated for a standard set of Holidays. The following are the default holidays automatically defined by CSE:

New Year's Day	*January 1
M L King Day	*January 15
President's Day	3rd Monday in February
Memorial Day	last Monday in May
Fourth of July	*July 4
Labor Day	1st Monday in September
Columbus Day	2nd Monday in October
Veterans Day	*November 11
Thanksgiving	4th Thursday in November
Christmas	*December 25

* *observed on the following Monday if falls on a weekend, except as otherwise noted:*

If a particular default holiday is not desired, it can be removed with a DELETE statement:

```
DELETE HOLIDAY Thanksgiving

DELETE HOLIDAY "Columbus Day"  // Quotes necessary (due to space)

DELETE HOLIDAY "VETERANS DAY"  // No case-sensitivity
```

Note that the name must be spelled *exactly* as listed above.

5.3 MATERIAL

MATERIAL constructs an object of class MATERIAL that represents a building material or component for later reference a from LAYER (see below). A MATERIAL so defined need not be referenced. MATERIAL properties are defined in a consistent set of units (all lengths in feet), which in some cases differs from units used in tabulated data. Note that the convective and air film resistances for the inside wall surface is defined within the SURFACE statements related to conductances.

matName

Name of material being defined; follows the word MATERIAL

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

matThk=*float*

Thickness of material. If specified, matThk indicates the discreet thickness of a component as used in construction assemblies. If omitted, matThk indicates that the material can be used in any thickness; the thickness is then specified in each LAYER using the material (see below).

Units	Legal Range	Default	Required	Variability
ft	$x > 0$	(omitted)	No	constant

matCond=*float*

Conductivity of material. Note that conductivity is *always* stated for a 1 foot thickness, even when matThk is specified; if the conductance is known for a specific thickness, an expression can be used to derive matCond.

Units	Legal Range	Default	Required	Variability
Btuh-ft/ft^2-°F	$x > 0$	*none*	Yes	constant

matCondT=*float*

Temperature at which matCond is rated. See matCondCT (next).

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	70 °F	No	constant

matCondCT=*float*

Coefficient for temperature adjustment of matCond in the forward difference surface conduction model. Each hour (not subhour), the conductivity of layers using this material are adjusted as followslrCond = matCond * (1 + matCondCT*(T_{layer} – matCondT))

Units	Legal Range	Default	Required	Variability
°F^{-1}		0	No	constant

Note: A typical value of matCondCT for fiberglass batt insulation is 0.00418 F^{-1}

matSpHt=*float*

Specific heat of material.

Units	Legal Range	Default	Required	Variability
Btu/lb-°F	$x \geq 0$	0 (thermally massless)	No	constant

matDens=*float*

Density of material.

Units	Legal Range	Default	Required	Variability
lb/ft^3	$x \geq 0$	0 (massless)	No	constant

matRNom=*float*

Nominal R-value per foot of material. Appropriate for insulation materials only and *used for documentation only*. If specified, the current material is taken to have a nominal R-value that contributes to the reported nominal R-value for a construction. As with matCond, matRNom is *always* stated for a 1 foot thickness, even when matThk is specified; if the nominal R-value is known for a specific thickness, an expression can

be used to derive matRNom.

Units	Legal Range	Default	Required	Variability
ft^2-^{o}F/Btuh	$x > 0$	(omitted)	No	constant

endMaterial

Optional to indicate the end of the material. Alternatively, the end of the material definition can be indicated by "END" or simply by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.4 CONSTRUCTION

CONSTRUCTION constructs an object of class CONSTRUCTION that represents a light weight or massive ceiling, wall, floor, or mass assembly (mass assemblies cannot, obviously, be lightweight). Once defined, CONSTRUCTION can be referenced from SURFACE (below). A defined CONSTRUCTION need not be referenced. Each CONSTRUCTION is optionally followed by LAYER which define the constituent LAYER of the construction.

conName

Name of construction. Required for reference from SURFACE and DOOR objects, below.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

conU=*float*

U-value for the construction (NOT including surface (air film) conductances; see SURFACE statements). If omitted, one or more LAYER must immediately follow to specify the LAYER that make up the construction. If specified, no LAYER can follow.

Units

Legal Range

Default

Required

Variabili ty

Btuh/ft2- oF

x > 0

calculated from LAYERs

if omitted, LAYERs must follow

constant

endConstruction

Optional to indicates the end of the CONSTRUCTION Alternatively, the end of the CONSTRUCTION definition can be indicated by "END" or by beginning another object If END or endConstruction is used, it should follow the construction's LAYER subobjects, if any.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.4.1 LAYER

LAYER constructs a subobject of class LAYER belonging to the current CONSTRUCTION LAYER is not recognized except immediately following CONSTRUCTION or another LAYER The members represent one layer (that optionally includes framing) within the CONSTRUCTION

The layers should be specified in inside to outside order. A framed layer (lrFrmMat and lrFrmFrac given) is modeled by creating a homogenized material with weighted combined conductivity and volumetric heat capacity. Caution: it is generally preferable to model framed constructions using two separate surfaces (one with framing, one without). At most one framed layer (lrFrmMat and lrFrmFrac given) is allowed per construction.

The layer thickness may be given by lrThk, or matThk of the material, or matThk of the framing material if any. The thickness must be specified at least one of these three places; if specified in more than one place and not consistent, an error message occurs.

lrName

Name of layer (follows LAYER Required only if the LAYER is later referenced in another object, for example with LIKE or ALTER; however, we suggest naming all objects for clearer error messages and future flexibility.

Units	Legal Range	Default	Required	Variability
	63 characters	*None*	No	constant

lrMat=*matName*

Name of primary MATERIAL in layer.

Units	Legal Range	Default	Required	Variability
	name of a *MATERIAL*	*none*	Yes	constant

lrThk=*float*

Thickness of layer.

Units	Legal Range	Default/Required	Variability
ft	$x > 0$	Required if *matThk* not specified in referenced *lrMat*	constant

lrFrmMat=*matName*

Name of framing MATERIAL in layer, if any. At most one layer with lrFrmMat is allowed per CONSTRUCTION See caution above regarding framed-layer model.

Units	Legal Range	Default	Required	Variability
	name of a MATERIAL	*no framed layer*	No	constant

lrFrmFrac=*float*

Fraction of layer that is framing. Must be specified if frmMat is specified. See caution above regarding framed-layer model.

Unit s

Legal Range

Default

Required

Variabil ity

0 x 1

no framed layer

Required if lrFrmMat specified, else disallowed

constant

endLayer

Optional end-of-LAYER indicator; LAYER definition may also be indicated by "END" or just starting the definition of another LAYER or other object.

5.5 GLAZETYPE

GLAZETYPE constructs an object of class GLAZETYPE that represents a glazing type for use in WINDOW

gtName

Name of glazetype. Required for reference fromWINDOW objects, below.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

gtModel=*choice*

Selects model to be used for WINDOW based on this GLAZETYPE

Units	Legal Range	Default	Required	Variability
	SHGC ASHWAT	SHGC	No	constant

gtU=*float*

Glazing conductance (U-factor without surface films, therefore not actually a U-factor but a C-factor). Used as wnU default; an error message will be issued if the U value is not given in the window (wnU) nor in the glazeType (gtU). Preferred Approach: To use accurately with standard winter rated U-factor from ASHRAE or NFRC enter as:

```
gtU = (1/((1/U-factor)-0.85)
```

Where 0.85 is the sum of the interior (0.68) and exterior (0.17) design air film resistances assumed for rating window U-factors. Enter wnInH (usually 1.5=1/0.68) instead of letting it default. Enter the wnExH or let it default. It is important to use this approach if the input includes gnFrad for any gain term. Using approach 2 below will result in an inappropriate internal gain split at the window.

Approach 2. Enter gtU=U-factor and let the wnInH and wnExH default. This approach systematically underestimates the window U-factor because it adds the wnExfilm resistance to 1/U-factor thereby double

counting the exterior film resistance. This approach will also yield incorrect results for gnFrad internal gain since the high wnInH will put almost all the gain back in the space.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	*none*	No	constant

gtUNFRC=*float*

Fenestration system (including frame) U-factor evaluated at NFRC heating conditions. For ASHWAT windows, a value for the NFRC U-factor is required, set via gtUNFRC or wnUNFRC.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	*none*	No	constant

gtSHGC=*float*

Glazing Solar Heat Gain Coefficient: fraction of normal beam insolation which gets through glass to space inside. We recommend using this to represent the glass normal transmissivity characteristic only, before shading and framing effects

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	*none*	Yes	Constant

gtSMSO=*float*

SHGC multiplier with shades open. May be overriden in the specific window input.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	1.0	No	Monthly - Hourly

gtSMSC=*float*

SHGC multiplier with shades closed. May be overriden in the specific window input.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	gtSMSO (no shades)	No	Monthly - Hourly

gtFMult=*float*

Framing multiplier used if none given in window, for example .9 if frame and mullions reduce the solar gain by 10%. Default of 1.0 implies frame/mullion effects allowed for in gtSHGC's or always specified in Windows.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	gtSHGCO	No	Monthly - Hourly

gtPySHGC =*float*

Four float values separated by commas. Coefficients for incidence angle SHGC multiplier polynomial applied to gtSHGC to determine beam transmissivity at angles of incidence other than 90 degrees. The values are

coefficients for first through fourth powers of the cosine of the incidence angle; there is no constant part. An error message will be issued if the coefficients do not add to one. They are used in the following computation:

angle = incidence angle of beam radiation, measured from normal to glass.

cosI = cos(angle)

angMult = a*cosI + b*cosIˆ2 + c*cosIˆ3 + d*cosIˆ4

beamXmisvty = gtSHGCO * angMult (shades open)

Units	Legal Range	Default	Required	Variability
float	*any*	none	Yes	Constant

gtDMSHGC=*float*

SHGC diffuse multiplier, applied to gtSHGC to determine transmissivity for diffuse radiation.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	none	yes	Constant

gtDMRBSol=*float*

SHGC diffuse multiplier, applied to qtSHGC to determine transmissivity for diffuse radiation reflected back out the window. Misnamed as a reflectance. Assume equal to DMSHGC if no other data available.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	none	yes	Constant

gtNGlz=*int*

Number of glazings in the Glazetype (bare glass only, not including any interior or exterior shades).

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 4$	2	no	Constant

gtExShd=*choice*

Exterior shading type (ASHWAT only).

Units	Legal Range	Default	Required	Variability
	NONE INSCRN	NONE	no	Constant

gtInShd=*choice*

Interior shade type (ASHWAT only).

Units	Legal Range	Default	Required	Variability
	NONE DRAPEMED	NONE	no	Constant

gtDirtLoss=*float*

Glazing dirt loss factor.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	0	no	Constant

endGlazeType

Optional to indicates the end of the Glazetype. Alternatively, the end of the GLAZETYPE definition can be indicated by "END" or by beginning another object

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.6 METER

A METER object is a user-defined "device" that records energy consumption of equipment as simulated by CSE. The user defines METER with the desired names, then assigns energy uses of specific equipment to the desired meters using commands described under each equipment type's class description AIRHANDLER TERMINAL etc.). Additional energy use from equipment not simulated by CSE (except optionally for its effect on heating and cooling loads) can also be charged to METER (see GAIN The data accumulated by meters can be reported at hourly, daily, monthly, and annual (run) intervals by using REPORT and EXPORT of type MTR.

Meters account for energy use in the following pre-defined categories, called *end uses*. The abbreviations in parentheses are used in MTR report headings (and for gnMeter input, below).

- Total use
- Space Cooling Use (Clg)
- Space Heating Use - including heat pump compressor (Htg)
- Heat Pump Backup Heat (HPHtg)
- Domestic (Service)
- Hot Water Heating (DHW)
- Fans – AC and cooling ventilation (FanC)
- Fans – heating (FanH)
- Fans – IAQ venting (FanV)
- Fans – other (Fan)
- HVAC Auxiliary - not including fans (Aux)
- Process Energy (Proc)
- Lighting (Lit)
- Receptacles (Rcp)
- Exterior (Ext)
- Refrigeration (Refr)
- Dish washing (Dish)
- Clothes drying (Dry)
- Clothes washing (Wash)
- Cooking (Cook)
- User defined 1 (User1)
- User defined 2 (User2)

The user has complete freedom over how many meters are defined and how equipment is assigned to them. At one extreme, a single meter "Electricity" could be defined and have all of electrical uses assigned to it. On the other hand, definition of separate meters "Elect_Fan1", "Elect_Fan2", and so forth allows accounting

of the electricity use for individual pieces of equipment. Various groupings are possible: for example, in a building with several air handlers, one could separate the energy consumption of the fans from the coils, or one could separate the energy use by air handler, or both ways, depending on the information desired from the run.

The members which assign energy use to meters include:

- GAIN: gnMeter, gnEndUse
- ZONE: xfanMtr
- IZXFER: izfanMtr
- RSYS: rsElecMtr, rsFuelMtr
- DHWSYS
- DHWHEATER
- DHWPUMP
- DHWLOOPPUMP
- TERMINAL: tuhcMtr, tfanMtr
- AIRHANDLER: sfanMtr, rfanMtr, ahhcMtr, ahccMtr, ahhcAuxOnMtr, ahhcAuxOffMtr, ahhcAuxFullOnMtr, ahhcAuxOnAtAllMtr, ahccAuxOnMtr, ahccAuxOffMtr, ahccAuxFullOnMtr, ahccAuxOnAtAllMtr
- BOILER: blrMtr, blrpMtr, blrAuxOnMtr, blrAuxOffMtr, blrAuxFullOnMtr, blrAuxOnAtAllMtr
- CHILLER: chMtr, chppMtr, chcpMtr, chAuxOnMtr, chAuxOffMtr, chAuxFullOnMtr, chAuxOnAtAllMtr
- TOWERPLANT: tpMtr

The end use can be specified by the user only for GAIN in other cases it is hard-wired to Clg, Htg, FanC, FanH, FanV, Fan, or Aux as appropriate.

mtrName

Name of meter: required for assigning energy uses to the meter elsewhere.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

endMeter

Indicates the end of the meter definition. Alternatively, the end of the meter definition can be indicated by the declaration of another object or by END.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7 ZONE

ZONE constructs an object of class ZONE which describes an area of the building to be modeled as having a uniform condition. ZONE are large, complex objects and can have many subobjects that describe associated surfaces, shading devices, HVAC equipment, etc.

5.7.1 ZONE General Members

znName

Name of zone. Enter after the word ZONE no "=" is used.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	Yes	constant

znModel=*choice*

Selects model for zone.

Units	Legal Range	Default	Required	Variability
	CNE CZM UZM	CNE	No	constant

znArea=*float*

Nominal zone floor area.

Units	Legal Range	Default	Required	Variability
ft^2	$x > 0$	*none*	Yes	constant

znVol=*float*

Nominal zone volume.

Units	Legal Range	Default	Required	Variability
ft^3	$x > 0$	*none*	Yes	constant

znFloorZ=*float*

Nominal zone floor height relative to arbitrary 0 level. Used re determination of vent heights

Units	Legal Range	Default	Required	Variability
ft	unrestricted	*0*	No	constant

znCeilingHt=*float*

Nominal zone ceiling height relative to zone floor (typically 8 – 10 ft).

Units	Legal Range	Default	Required	Variability
ft	$x > 0$	*znVol / znArea*	No	constant

znEaveZ=*float*

Nominal eave height above ground level. Used re calculation of local surface wind speed. This in turn influences outside convection coefficients in some surface models and wind-driven air leakage.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	*znFloorZ + infStories*8*	No	constant

znCAir=*float*

Zone "air" heat capacity: represents heat capacity of air, furniture, "light" walls, and everything in zone except surfaces surfaces having heat capacity (that is, non-QUICK surfaces).

Units	Legal Range	Default	Required	Variability
Btu/°F	$x \geq 0$	3.5 * *znArea*	No	constant

znAzm=*float*

Zone azimuth with respect to bldgAzm. All surface azimuths are relative to znAzm, so that the zone can be rotated by changing this member only. Values outside the range 0° to 360° are normalized to that range.

Units	Legal Range	Default	Required	Variability
degrees	unrestricted	0	No	constant

znSC=*float*

Zone shade closure. Determines insolation through windows (see WINDOW members *wnSCSO* and *wnSCSC*) and solar gain distribution: see SGDIST members *sgFSO* and *sgFSC*. 0 represents shades open; 1 represents shades closed; intermediate values are allowed. An hourly variable CSE expression may be used to schedule shade closure as a function of weather, time of year, previous interval HVAC use or zone temperature, etc.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	1 when cooling was used in *previous* hour, else 0	No	hourly

The following apply to *znModel* = CZM.

znTH=*float*

Heating set point

Units	Legal Range	Default	Required	Variability
°F	$x \geq 0$			hourly

znTD=*float*

Desired set point (temperature maintained with ventilation if possible).

Units	Legal Range	Default	Required	Variability
°F	$x \geq 0$			hourly

znTC=*float*

Cooling set point

Units	Legal Range	Default	Required	Variability
°F	$x \geq 0$			Hourly

znQMxH=*float*

Heating capacity at current conditions

Units	Legal Range	Default	Required	Variability
Btuh	$x \geq 0$			hourly

znQMxHRated=*float*

Rated heating capacity

Units	Legal Range	Default	Required	Variability
Btuh	$x \geq 0$			constant

znQMxC=*float*

Cooling capacity at current conditions

Units	Legal Range	Default	Required	Variability
Btuh	$x \leq 0$			hourly

znQMxCRated=*float*

Rated cooling capacity

Units	Legal Range	Default	Required	Variability
Btuh	$x \leq 0$			constant

The following provide parameters for comfort calculations

znComfClo=*float*

Occupant clothing resistance

Units	Legal Range	Default	Required	Variability
clo				subhourly ??

znComfMet=*float*

Occupant metabolic rate

Units	Legal Range	Default	Required	Variability
met	$x \geq 0$			hourly

znComfAirV=*float*

Nominal air velocity used for comfort model

Units	Legal Range	Default	Required	Variability
?	$x \geq 0$		No	Hourly

znComfRh=*float*

Nominal zone relative humidity used for comfort model

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$		No	hourly

5.7.2 ZONE Infiltration

The following control a simplified air change plus leakage area model. The Sherman-Grimsrud model is used to derive air flow rate from leakage area and this rate is added to the air changes specified with infAC. Note that TOP.windF does *not* modify calculated infiltration rates, since the Sherman-Grimsrud model uses its own modifiers. See also AirNet models available via IZXFER

infAC=*float*

Zone infiltration air changes per hour.

Units	Legal Range	Default	Required	Variability
1/hr	$x \geq 0$	0.5	No	hourly

infELA=*float*

Zone effective leakage area (ELA).

Units	Legal Range	Default	Required	Variability
in^2	$x \geq 0$	0.0	No	hourly

infShld=*int*

Zone local shielding class, used in derivation of local wind speed for ELA infiltration model, wind-driven AirNet leakage, and exterior surface coefficients. infShld values are –

1

no obstructions or local shielding

2

light local shielding with few obstructions

3

moderate local shielding, some obstructions within two house heights

4

heavy shielding, obstructions around most of the perimeter

5

very heavy shielding, large obstructions surrounding the perimeter within two house heights

Units	Legal Range	Default	Required	Variability
	$1 \leq x \leq 5$	3	No	constant

infStories=*int*

Number of stories in zone, used in ELA model.

Units	Legal Range	Default	Required	Variability
	$1 \leq x \leq 3$	1	No	constant

znWindFLkg=*floatTODO*

Units	Legal Range	Default	Required	Variability
		1	No	constant

5.7.3 ZONE Exhaust Fan

Presence of an exhaust fan in a zone is indicated by specifying a non-zero design flow value (xfanVfDs).

Zone exhaust fan model implementation is incomplete as of July, 2011. The current code calculates energy use but does not account for the effects of air transfer on room heat balance. IZXFER provides a more complete implementation.

xfanFOn=*float*

Exhaust fan on fraction. On/off control assumed, so electricity requirement is proportional to run time.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	1	No	hourly

Example: The following would run an exhaust fan 70% of the time between 8 AM and 5 PM:

```
xfanFOn = select( (\$hour >= 7 && \$hour < 5), .7,
                                              default, 0 );
```

xfanVfDs=*float*

Exhaust fan design flow; 0 or not given indicates no fan.

Units	Legal Range	Default	Required	Variability
cfm	x 0	0 (no fan)	If fan present	constant

xfanPress=*float*

Exhaust fan external static pressure.

Units	Legal Range	Default	Required	Variability
inches H_2O	$0.05 \leq x \leq 1.0$	0.3	No	constant

Only one of xfanElecPwr, xfanEff, and xfanShaftBhp may be given: together with xfanVfDs and xfanPress, any one is sufficient for CSE to detemine the others and to compute the fan heat contribution to the air stream.

xfanElecPwr=*float*

Fan input power per unit air flow (at design flow and pressure).

Unit s

Legal Range

Default

Required

Variabi lity

W/cfm

x > 0

derived from xfanEff and xfanShaftBhp

If xfanEff and xfanShaftBhp not present

constant

xfanEff=*floatExhaust*

fan/motor/drive combined efficiency.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	0.08	No	constant

xfanShaftBhp=*float*

Fan shaft power at design flow and pressure.

Units

Legal Range

Default

Required

Variabili ty

BHP

x > 0

derived from xfanElecPwr and xfanVfDs

If xfanElecPwr not present

constant

xfanMtr=*mtrName*

Name of METER object, if any, by which fan's energy use is recorded (under end use category "fan").

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endZone

Indicates the end of the zone definition. Alternatively, the end of the zone definition can be indicated by the declaration of another object or by "END". If END or endZone is used, it should follow the definitions of the ZONE subobjects such as GAIN SURFACE TERMINAL etc.

Units	Legal Range	Default	Required	Variability

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.4 GAIN

A GAIN object adds sensible and/or latent heat to the ZONE and/or adds arbitrary energy use to a METER GAIN are subobjects of ZONE and are normally given within the input for their ZONE (also see ALTER). As many GAIN as desired (or none) may be given for each ZONE

Each gain has an amount of power (gnPower), which may optionally be accumulated to a METER (gnMeter). The power may be distributed to the zone, plenum, or return as sensible heat with an optionl fraction radiant, or to the zone as latent heat (moisture addition), or not.

gnName

Name of gain; follows the word GAIN if given.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

gnPower=*float*

Rate of heat addition to zone. Negative gnPower values may be used to represent heat removal. Expressions containing functions are commonly used with this member to schedule the gain power on a daily and/or hourly basis. Refer to the functions section in Section 4 for details and examples.

All gains, including electrical, are specified in Btuh units. But note that internal gain and meter reporting is in MBtuh by default even though input is in Btuh.

Units	Legal Range	Default	Required	Variability
Btuh	*no restrictions*	*none*	Yes	hourly

gnMeter=*choice*

Name of meter by which this GAIN gnPower is recorded. If omitted, gain is assigned to no meter and energy use is not accounted in CSE simulation reports; thus, gnMeter should only be omitted for "free" energy sources.

Units	Legal Range	Default	Required	Variability
	meter name	*none*	No	constant

gnEndUse=*choice*

Meter end use to which the GAIN energy use should be accumulated.

Clg	Cooling
Htg	Heating (includes heat pump compressor)
HPHTG	Heat pump backup heat
DHW	Domestic (service) hot water

FANC	Fans, AC and cooling ventilation
FANH	Fans, heating
FANV	Fans, IAQ venting
FAN	Fans, other purposes
AUX	HVAC auxiliaries such as pumps
PROC	Process
LIT	Lighting
RCP	Receptacles
EXT	Exterior lighting
REFR	Refrigeration
DISH	Dishwashing
DRY	Clothes drying
WASH	Clothes washing
COOK	Cooking
USER1	User-defined category 1
USER2	User-defined category 2

Units	Legal Range	Default	Required	Variability
	Codes listed above	*none*	Required if gnMeter is given	constant

The gnFrZn, gnFrPl, and gnFrRtn members allow you to allocate the gain among the zone, the zone's plenum, and the zone's return air flow. Values that total to more than 1.0 constitute an error. If they total less than 1, the unallocated portion of the gain is recorded by the meter (if specified) but not transferred into the building. By default, all of the gain not directed to the return or plenum goes to the zone.

gnFrZn=*float*

Fraction of gain going to zone. gnFrLat (below) gives portion of this gain that is latent, if any; the remainder is sensible.

Units	Legal Range	Default	Required	Variability
	gnFrZn + gnFrPl + gnFrRtn $\leq$ 1	*1 - gnFrLat - gnFrPl - gnFrRtn*	No	hourly

gnFrPl=*float*

Fraction of gain going to plenum. Plenums are not implementented as of August, 2012. Any gain directed to the plenum is discarded.

gnFrRtn=*float*

Fraction of gain going to return. The return fraction model is not implemented as of August, 2012. Any gain directed to the return is discarded.

Units	Legal Range	Default	Required	Variability
	gnFrZn + gnFrPl + gnFrRtn $\leq$ 1	0.	No	hourly

The gain to the zone may be further divided into convective sensible, radiant sensible and latent heat via the gnFrRad and gnFrLat members; the plenum and return gains are assumed all convective sensible.

Gain Modeling in CR zone

In the CNE zone mode, the radiant internal gain is distributed to the surfaces in the zone, rather than going directly to the zone "air" heat capacity (znCAir). A simple model is used – all surfaces are assumed to

be opaque and to have the same (infrared) absorptivity – even windows. Along with the assumption that the zone is spherical (implicit in the current treatment of solar gains), this allows distribution of gains to surfaces in proportion to their area, without any absorptivity or transmissivity calculations. The gain for windows and quick-model surfaces is assigned to the znCAir, except for the portion which conducts through the surface to the other side rather than through the surface film to the adjacent zone air; the gain to massive (delayed-model) surfaces is assigned to the side of surface in the zone with the gain.

Gain Modeling in CNE zones

In the CNE zone mode, the radiant internal gain is distributed to the surfaces in the zone, rather than going directly to the zone "air" heat capacity (znCAir). A simple model is used – all surfaces are assumed to be opaque and to have the same (infrared) absorptivity – even windows. Along with the assumption that the zone is spherical (implicit in the current treatment of solar gains), this allows distribution of gains to surfaces in proportion to their area, without any absorptivity or transmissivity calculations. The gain for windows and quick-model surfaces is assigned to the znCAir, except for the portion which conducts through the surface to the other side rather than through the surface film to the adjacent zone air; the gain to massive (delayed-model) surfaces is assigned to the side of surface in the zone with the gain.

Radiant internal gains are included in the IgnS (Sensible Internal Gain) column in the zone energy balance reports. (They could easily be shown in a separate IgnR column if desired.) Any energy transfer shows two places in the ZEB report, with opposite signs, so that the result is zero – otherwise it wouldn't be an energy balance. The rest of the reporting story for radiant internal gains turns out to be complex. The specified value of the radiant gain (gnPower * gnFrZn * gnFrRad) shows in the IgnS column. To the extent that the gain heats the zone, it also shows negatively in the Masses column, because the zone CAir is lumped with the other masses. To the extent that the gain heats massive surfaces, it also shows negatively in the masses column. To the extent that the gain conducts through windows and quick-model surfaces, it shows negatively in the Conduction column. If the gain conducts through a quick-model surface to another zone, it shows negatively in the Izone (Interzone) column, positively in the Izone column of the receiving zone, and negatively in the receiving zone's Masses or Cond column.

gnFrRad=*float*

Fraction of total gain going to zone (gnFrZn) that is radiant rather than convective or latent.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.	No	hourly

gnFrLat=*float*

Fraction of total gain going to zone (gnFrZn) that is latent heat (moisture addition).

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.	No	hourly

endGain

Optional to indicate the end of the GAIN definition. Alternatively, the end of the gain definition can be indicated by END or by the declaration of another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.5 SURFACE

Surface constructs a ZONE subobject of class SURFACE that represents a surrounding or interior surface of the zone. Internally, SURFACE generates a QUICK surface (U-value only), a DELAYED (massive) surface (using the finite-difference mass model), interzone QUICK surface, or interzone DELAYED surface, as appropriate for the specified construction and exterior conditions.

sfName

Name of surface; give after the word SURFACE

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

sfType=*choice*

Type of surface:

FLOOR

Surface defines part or all of the "bottom" of the zone; it is horizontal with inside facing up. The outside of the surface is not adjacent to the current zone.

WALL

Surface defines a "side" of the zone; its outside is not adjacent to the current zone.

CEILING

Surface defines part or all of the "top" of the zone with the inside facing down. The outside of the surface is not adjacent to the current zone.

sfType is used extensively for default determination and input checking, but does not have any further internal effect. The Floor, Wall, and Ceiling choices identify surfaces that form boundaries between the zone and some other condition.

Units	Legal Range	Default	Required	Variability
	FLOOR WALL CEILING	*none*	Yes	constant

sfArea=*float*

Gross area of surface. (CSE computes the net area for simulation by subtracting the areas of any windows and doors in the surface.).

Units	Legal Range	Default	Required	Variability
ft^2	$x > 0$	*none*	Yes	constant

sfTilt=*float*

Surface tilt from horizontal. Values outside the range 0 to 360 are first normalized to that range. The default and allowed range depend on sfType, as follows:

sfType = FLOOR	*sfTilt*=180, default = 180 (fixed value)
sfType = WALL	$60 < sfTilt < 180$, default = 90
sfType = CEILING	$0 \leq sfTilt \leq 60$, default = 0

Units	Legal Range / Default	Required	Variability
degrees	Dependent upon *sfType. See above*	No	constant

sfAzm=*float*

Azimuth of surface with respect to znAzm. The azimuth used in simulating a surface is bldgAzm + znAzm + sfAzm; the surface is rotated if any of those are changed. Values outside the range 0 to 360 are normalized to that range. Required for non-horizontal surfaces.

Units

Legal Range

Defaul t

Required

Variabil ity

degrees

unrestrict ed

none

Required if sfTilt 0 and sfTilt 180

constant

sfModel=*choice*

Provides user control over how program models this surface:

QUICK

Surface is modeled using a simple conductance. Heat capacity effects are ignored. Either sfCon or sfU (next) can be specified.

DELAYED, DELAYED_HOUR, DELAYED_SUBHOUR

Surface is modeled using a multi-layer finite difference technique that represents heat capacity effects. If the time constant of the surface is too short to accurately simulate, a warning message is issued and the Quick model is used. The program cannot use the finite difference model if sfU rather than sfCon is specified.

AUTO

Program selects Quick or the appropriate Delayed automatically according to the time constant of the surface (if sfU is specified, Quick is selected).

FD (or FORWARD_DIFFERENCE)

Selects the forward difference model (used with short time steps and the CZM zone model)

Units	Legal Range	Default	Required	Variabil
	QUICK, DELAYED, DELAYED_HOUR, DELAYED_SUBOUR, AUTO, FD	AUTO	No	constant

One of the following two parameters must be specified:

sfCon=*conName*

Name of CONSTRUCTION of the surface.

Units	Legal Range	Default	Required	Variability
	Name of a *CONSTRUCTION*	*none*	unless *sfU* given	constant

sfU=*float*

Surface U-value (NOT including surface (air film) conductances). For surfaces for which no heat capacity is to be modeled, allows direct entry of U-value without defining a CONSTRUCTION

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	$x > 0$	Determined from *sfCon*	if *sfCon* not given	constant

sfExCnd=*choice*

Specifies the thermal conditions assumed at surface exterior, and at exterior of any subobjects (windows or doors) belonging to current surface. The conditions accounted for are dry bulb temperature and incident solar radiation.

AMBIENT

Exterior surface is exposed to the "weather" as read from the weather file. Solar gain is calculated using solar geometry, sfAzm, sfTilt, and sfExAbs.

SPECIFIEDT

Exterior surface is exposed to solar radiation as in AMBIENT, but the dry bulb temperature is calculated with a user specified function (sfExT). sfExAbs can be set to 0 to eliminate solar effects.

ADJZN

Exterior surface is exposed to another zone, whose name is specified by sfAdjZn. Solar gain is 0 unless gain is targeted to the surface with SGDIST below.

ADIABATIC

Exterior surface heat flow is 0. Thermal storage effects of delayed surfaces are modeled.

sfInH=*float*

Inside surface (air film) conductance. Ignored for sfModel = Forward_Difference. Default depends on the surface type.

sfType = FLOOR or CEILING	1.32
other	1.5

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	$x > 0$	*see above*	No	constant

sfExH=*float*

Outside surface (air film) conductance. Ignored for sfModel = Forward_Difference. The default value is dependent upon the exterior conditions:

sfExCnd = AMBIENT

dflExH (Top-level member, described above)

sfExCnd = SPECIFIEDT

dflExH (described above)

sfExCnd = ADJZN

1.5

sfExCnd = ADIABATIC

not applicable

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	$x > 0$	see above	No	constant

sfExAbs=*float*

Surface exterior absorptivity.

Unit s

Legal Range

Defaul t

Required

Variabil ity

(none)

0 x 1

0.5

Required if sfExCon = AMBIENT or sfExCon = SPECIFIEDT

monthly-ho urly

sfInAbs=*float*

Surface interior solar absorptivity.

Unit s

Legal Range

Default

Requir ed

Variabi lity

(none)

0 x 1

sfType = CEILING, 0.2; sfType = WALL, 0.6; sfType = FLOOR, 0.8

No

monthly-h ourly

sfExEpsLW=*float*

Surface exterior long wave (thermal) emittance.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.9	No	constant

sfInEpsLW=*float*

Surface interior long wave (thermal) emittance.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.9	No	constant

sfExT=*float*

Exterior air temperature.

Units	Legal Range	Default	Required	Variability
°F	*unrestricted*	*none*	Required if *sfExCon* = SPECIFIEDT	hourly

sfAdjZn=*znName*

Name of adjacent zone; used only when sfExCon is ADJZN. Can be the same as the current zone.

Units	Legal Range	Default	Required	Variability
	name of a *ZONE*	*none*	Required when*sfExCon* = ADJZN	constant

sfGrndRefl=*float*

Ground reflectivity for this surface.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	grndRefl	No	Monthly - Hourly

endSurface

Optional to indicates the end of the surface definition. Alternatively, the end of the surface definition can be indicated by END, or by beginning another SURFACE or other object definition. If used, should follow the definitions of the SURFACE subobjects – DOOR WINDOW SHADE SGDIST etc.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

The following tables summarize the defaults and legal ranges of surface members for each sfType. "n.a." indicates "not applicable" and also "not allowed".

Membe r

WALL

FLOOR

CEILING

sfTilt

optional, default=90, 60 < sfTilt < 180

n.a. (fixed at 180)

optional, default=0, 0 sfTilt 60

sfAzm

required

n.a.

required if sfTilt > 0

**Member

WALL/FLOOR/CEILING

sfArea

required

sfCon

required unless sfU given

sfU

required unless sfCon given

sfInH

optional, default = 1.5

sfExH

optional, default per sfExCon

sfExCnd

optional, default = AMBIENT

sfExAbs

optional if sfExCon = AMBIENT or sfExCon = SPECIFIEDT (default = .5), else n.a.

sfExT

required if sfExCon = SPECIFIEDT; else n.a.

sfAdjZn

required if sfExCon = ADJZN; else n.a.

sfGrndRef l

optional, default to grndRefl

5.7.6 WINDOW

WINDOW defines a subobject belonging to the current SURFACE that represents one or more identical windows. The azimuth, tilt, and exterior conditions of the window are the same as those of the surface to which it belongs. The total window area ($wnHt \cdot wnWid \cdot wnMult$) is deducted from the gross surface area. A surface may have any number of windows.

Windows may optionally have operable interior shading that reduces the overall shading coefficient when closed.

wnName

Name of window: follows the word WINDOW if given.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wnHeight=*float*

Overall height of window (including frame).

Units	Legal Range	Default	Required	Variability
ft	$x > 0$	*none*	Yes	constant

wnWidth=*float*Overall width of window (including frame).

Units	Legal Range	Default	Required	Variability
ft	$x > 0$	*none*	Yes	constant

wnArea=*float*

Overall area of window (including frame).

Units	Legal Range	Default	Required	Variability
ft^2	$x > 0$	*wnHeight* * *wnWidth*	No	constant

wnMult=*float*

Area multiplier; can be used to represent multiple identical windows.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	No	constant

wnModel=*choice*

Selects window model

Units	Legal Range	Default	Required	Variability
	SHGC, ASHWAT	SHGC	No	constant

wnGt=*choice*

GLAZETYPE for window. Provides many defaults for window properties as cited below.

wnU=*float*

Window conductance (U-factor without surface films, therefore not actually a U-factor but a C-factor).

Preferred Approach: To use accurately with standard winter rated U-factor from ASHRAE or NFRC enter as:

```
wnU = (1/((1/U-factor)-0.85)
```

Where 0.85 is the sum of the interior (0.68) and exterior (0.17) design air film resistances assumed for rating window U-factors. Enter wnInH (usually 1.5=1/0.68) instead of letting it default. Enter the wnExH or let it default. It is important to use this approach if the input includes gnFrad for any gain term. Using approach 2 below will result in an inappropriate internal gain split at the window.

Approach 2. Enter wnU=U-factor and let the wnInH and wnExH default. Tnormally this approach systematically underestimates the window U-factor because it adds the wnExfilm resistance to 1/U-factor thereby double counting the exterior film resistance. This approach will also yield incorrect results for gnFrad internal gain since the high wnInH will put almost all the gain back in the space.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	*none*	Yes	constant

wnUNFRC=*float*

Fenestration system (including frame) U-factor evaluated at NFRC heating conditions.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	gtUNFRC	Required when *wnModel* = ASHWAT	constant

wnInH=*float*

Window interior surface (air film) conductance.

Preferred Approach: Enter the appropriate value for each window, normally:

```
wnInH = 1.5

where 1.5 = 1/0.68 the standard ASHRAE value.
```

The large default value of 10,000 represents a near-0 resistance, for the convenience of those who wish to include the interior surface film in wnU according to approach 2 above.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	10000	No	constant

wnExH=*float*

Window exterior surface (air film) conductance.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	same as owning surface	No	constant

wnSMSO=*float*

SHGC multiplier with shades open. Overrides gtSMSO.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	gtSMSO or 1	No	Monthly - Hourly

wnSMSC=*float*

SHGC multiplier with shades closed. Overrides gtSMSC

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	wnSMSO or gtSMSC	No	Monthly - Hourly

wnNGlz=*int*

Number of glazings in the window (bare glass only, not including any interior or exterior shades).

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 4$	gtNGLZ	Required when *wnModel* = ASHWAT	Constant

wnExShd=*choice*

Exterior shading type (ASHWAT only).

Units	Legal Range	Default	Required	Variability
	NONE, INSCRN	gtExShd	no	Constant

wnInShd=*choice*

Interior shade type (ASHWAT only).

Units	Legal Range	Default	Required	Variability
	NONE, DRAPEMED	gtInShd	no	Constant

wnDirtLoss=*float*

Glazing dirt loss factor.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	0	no	Constant

wnGrndRefl=*float*

Ground reflectivity for this window.

Units	Legal Range	Default	Required	Variability
fraction	$0 \leq x \leq 1$	sfGrndRefl	No	Monthly - Hourly

wnVfSkyDf=*float*

View factor from this window to sky for diffuse radiation. For the shading effects of an overhang, a wnVfSkyDf value smaller than the default would be used

Units

Legal Range

Default

Require d

Variabili ty

fractio n

0 x 1

0.5 - 0.5*cos(tilt) = .5 for vertical surface

No

Monthly - Hourly

wnVfGrndDf=*float*

View factor from this window to ground for diffuse radiation. For the shading effects of a fin(s), both wnVfSkyDf and wnVfGrndDf would be used.

Units

Legal Range

Default

Require d

Variabili ty

fractio n

0 x 1

0.5 + 0.5*cos(tilt) = .5 for vertical surface

No

Monthly - Hourly

endWindow

Optionally indicates the end of the window definition. Alternatively, the end of the window definition can be indicated by END or the declaration of another object. END or endWindow, if used, should follow any subobjects of the window SHADE and/or SGDIST

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.7 SHADE

SHADE constructs a subobject associated with the current WINDOW that represents fixed shading devices (overhangs and/or fins). A window may have at most one SHADE and only windows in vertical surfaces may have SHADE A SHADE can describe an overhang, a left fin, and/or a right fin; absence of any of these is specified by omitting or giving 0 for its depth. SHADE geometry can vary on a monthly basis, allowing modeling of awnings or other seasonal shading strategies.

shName

Name of shade; follows the word SHADE if given.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

ohDepth=*float*

Depth of overhang (from plane of window to outside edge of overhang). A zero value indicates no overhang.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

ohDistUp=*float*

Distance from top of window to bottom of overhang.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

ohExL=*float*

Distance from left edge of window (as viewed from the outside) to the left end of the overhang.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

ohExR=*float*

Distance from right edge of window (as viewed from the outside) to the right end of the overhang.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

ohFlap=*float*

Height of flap hanging down from outer edge of overhang.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

lfDepth=*float*

Depth of left fin from plane of window. A zero value indicates no fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

lfTopUp=*float*

Vertical distance from top of window to top of left fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

lfDistL=*float*

Distance from left edge of window to left fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

lfBotUp=*float*

Vertical distance from bottom of window to bottom of left fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

rfDepth=*float*

Depth of right fin from plane of window. A 0 value indicates no fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

rfTopUp=*float*

Vertical distance from top of window to top of right fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

rfDistR=*float*

Distance from right edge of window to right fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

rfBotUp=*float*

Vertical distance from bottom of window to bottom of right fin.

Units	Legal Range	Default	Required	Variability
ft	$x \geq 0$	0	No	monthly-hourly

endShade

Optional to indicate the end of the SHADE definition. Alternatively, the end of the shade definition can be indicated by END or the declaration of another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.8 SGDIST

SGDIST creates a subobject of the current window that distributes a specified fraction of that window's solar gain to a specified delayed model (massive) surface. Any remaining solar gain (all of the window's solar gain if no SGDIST are given) is added to the air of the zone containing the window. A window may have up to three SGDIST an error occurs if more than 100% of the window's gain is distributed.

Via members sgFSO and sgFSC, the fraction of the insolation distributed to the surface can be made dependent on whether the zone's shades are open or closed (see ZONE member znSC).

sgName

Name of solar gain distribution (follows SGDIST if given).

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

sgSurf=*sfName*

Name of surface to which gain is targeted.

If there is more than surface with the specified name: if one of the surfaces is in the current zone, it is used; otherwise, an error message is issued.

The specified surface must be modeled with the Delayed model. If gain is targeted to a Quick model surface, a warning message is issued and the gain is redirected to the air of the associated zone.

Units	Legal Range	Default	Required	Variability
	name of a *SURFACE*	*none*	Yes	constant

sgSide=*choice*

Designates the side of the surface to which the gain is to be targeted:

INTERIOR	Apply gain to interior of surface
EXTERIOR	Apply gain to exterior of surface

Uni ts

Legal Range

Default

Requ ired

**Varia bility

INTERIOR, EXTERIOR

Side of surface in zone containing window; or INTERIOR if both sides are in zone containing window.

Yes

constan t

sgFSO=*float*

Fraction of solar gain directed to specified surface when the owning window's interior shading is in the open position (when the window's zone's shade closure (znSC) is 0).

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$,and sum of window's sgFSO's ≤ 1	*none*	Yes	monthly-hourly

sgFSC=*float*

Fraction of solar gain directed to specified surface when the owning window's interior shading is in the closed position. If the zone's shades are partly closed (znSC between 0 and 1), a proportional fraction between sgFSO and sgFSC is used.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$, and sum of window's sgFSC's ≤ 1	*sgFSO*	No	monthly-hourly

endSGDist

Optionally indicates the end of the solar gain distribution definition. Alternatively, the end of the solar gain distribution definition can be indicated by END or by just beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.9 DOOR

DOOR constructs a subobject belonging to the current SURFACE The azimuth, tilt, ground reflectivity and exterior conditions associated with the door are the same as those of the owning surface, although the exterior surface conductance and the exterior absorptivity can be altered.

drName

Name of door.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

drArea=*float*

Overall area of door.

Units	Legal Range	Default	Required	Variability
ft^2	$x > 0$	*none*	Yes	constant

drModel=*choice*

Provides user control over how program models this door:

QUICK

Surface is modeled using a simple conductance. Heat capacity effects are ignored. Either drCon or drU (next) can be specified.

DELAYED, DELAYED_HOUR, DELAYED_SUBOUR

Surface is modeled using a multi-layer finite difference technique which represents heat capacity effects. If the time constant of the door is too short to accurately simulate, a warning message is issued and the Quick model is used. drCon (next) must be specified – the program cannot use the finite difference model if drU rather than drCon is specified.

AUTO

Program selects Quick or appropriate Delayed automatically according to the time constant of the surface (if drU is specified, Quick is selected).

FORWARD_DIFFERENCE

Selects the forward difference model (used with short time steps and the CZM zone model)

Unit s

Legal Range

Defau lt

Requir ed

Variabi lity

QUICK, DELAYED DELAYED_HOUR, DELAYED_SUBOUR, AUTO, FORWARD_DIFFERENCE

AUTO

No

constant

Units	Legal Range	Default	Required	Variability
	QUICK, DELAYED, AUTO	Auto	No	constant

Either drCon or drU must be specified, but not both.

drCon=*conName*

Name of construction for door.

Units	Legal Range	Default	Required	Variability
	name of a *CONSTRUCTION*	*None*	unless *drU* given	constant

drU=*float*

Door U-value, NOT including surface (air film) conductances. Allows direct entry of U-value, without defining a CONSTRUCTION when no heat capacity effects are to be modeled.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	Determined from *drCon*	if *drCon* not given	constant

drInH=*float*

Door interior surface (air film) conductance. Ignored if drModel = Forward_Difference

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	same as owning surface	No	constant

drExH=*float*

Door exterior surface (air film) conductance. Ignored if drModel = Forward_Difference

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	same as owning surface	No	constant

drExAbs=*float*

Door exterior solar absorptivity. Applicable only if sfExCon of owning surface is AMBIENT or SPECIFIEDT.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-^{o}F	$x > 0$	same as owning surface	No	monthly-hourly

drInAbs=*float*

Door interior solar absorptivity.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.5	No	monthly-hourly

drExEpsLW=*float*

Door exterior long wave (thermal) emittance.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.9	No	constant

drInEpsLW=*float*

Door interior long wave (thermal) emittance.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.9	No	constant

endDoor

Indicates the end of the door definition. Alternatively, the end of the door definition can be indicated by the declaration of another object or by END.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.10 PERIMETER

PERIMETER defines a subobject belonging to the current zone that represents a length of exposed edge of a (slab on grade) floor.

prName

Optional name of perimeter.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

prLen=*float*

Length of exposed perimeter.

Units	Legal Range	Default	Required	Variability
ft	$x > 0$	*none*	Yes	constant

prF2=*float*

Perimeter conduction per unit length.

Units	Legal Range	Default	Required	Variability
Btuh/ft-°F	$x > 0$	*none*	Yes	constant

endPerimeter

Optionally indicates the end of the perimeter definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.7.11 TERMINAL

TERMINAL constructs an object to represent equipment that transfers energy to or from the current zone from a local heating device (coil, etc.) and/or one AIRHANDLER A terminal serves a zone (and, internally, is owned by a zone). Up to three terminals can be defined for each zone.

Only zones having znModel=CNE can be served by a terminal.

A terminal can have local heating *capability,* using a simulated reheat coil, baseboard heater, etc. and/or air heating/cooling capability, using a simulated variable air volume (VAV) box connected to an AIRHANDLER (Section 0). Since a TERMINAL can only connect to a single air handler, use two terminals per zone to model systems where separate air handlers supply hot and cool air (dual duct). If a local heat capability utilizes the air flow (e.g. a reheat coil), model it in the terminal connected to the air handler; if a local heat capability is independent of air flow (e.g. electric baseboard heaters), it doesn't matter whether you model it with a separate terminal.

Each capability can be *set output*, in which the output is constant or determined by external conditions such as in an outdoor reset baseboard situation or *set temperature*, in which the output is modulated to maintain the zone temperature at a set point. Set temperature operation is established by giving the setpoint for the capability (tuTLh, tuTH, tuTC); set output operation is established by specifying the local heat output (tuQMnLh) or air flow (tuVfMn) without specifying a setpoint.

Hourly variable expressions may be used as desired to schedule setpoints and flow limits. Figure 1 shows [need sentence describing the figure.]

tuName

Optional name of terminal; follows the word TERMINAL if given.

Units	Legal Range	Default	Required	Variability
	63 characters		No	constant

5.7.11.1 TERMINAL Local Heating

These commands establish the TERMINAL local heating capability and determine whether it operates in set output or set temperature fashion. Additional details of the local heating mechanism are given with

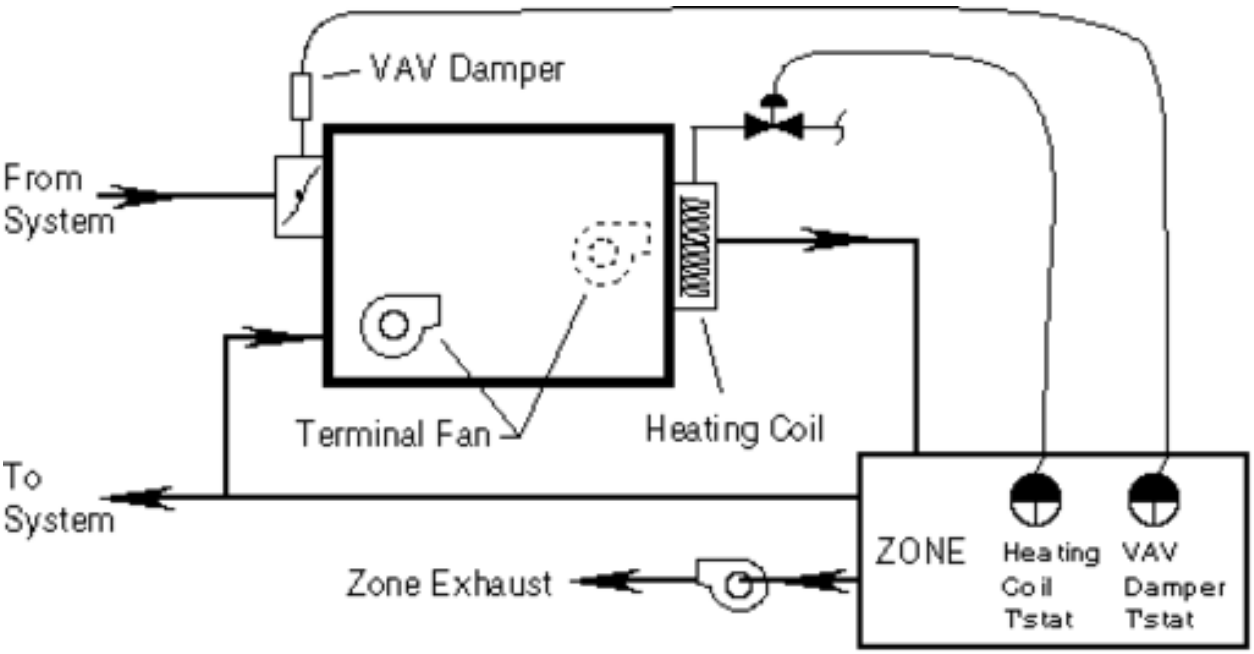

Figure 1: Insert Figure Title

commands described below under *terminal heating coil*.

Either tuTLh or tuQMnLh must be given to establish the TERMINAL local heat capability:

tuTLh=*float*

Local heating thermostat setpoint. Hourly expression may be used to schedule as desired. Giving this implies *set temperature* local heat from this terminal; omitting implies no local heat or, if tuQMnLh is given, set output local heat.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	*no thermostat control*	No	hourly

tuQMnLh=float

Minimum local heat output or set local heat output. If tuTLh is given, this is the minimum output, used when the thermostat is not calling for (local) heat. If tuTLh is not given, giving tuQMnLh implies *set output* local heat and specifies the set output level. An hourly expression may be used to schedule as desired.

Units

Legal Range

Default

Required

Variabili ty

Btuh

x 0

0 if tuTLh givenelse no local heat

For set output local heat

hourly

The next three items are allowed only for thermostat controlled local heating (tuTLh given):

tuQMxLh=*float*

Maximum desired power, used when thermostat is calling for heat continuously, subject to coil capacity, and to HEATPLANT limitations where pertinent (see *tuhcCaptRat* description). If *tuQMxLh* is less than

tuLhNeedsFlow=*choice*

YES

local heat being modeled requires terminal air flow (e.g. reheat coil). Local heat is then disabled when there is zero air flow through the terminal (when simulated VAV damper set to 0 flow, or when air handler fan and terminal fan both off)

NO

no local heat or does not require air flow (e.g. baseboard heaters).

Units	Legal Range	Default	Required	Variability
	YES, NO	NO	No	constant

5.7.11.2 TERMINAL Air Heating and Cooling

These commands establish whether the TERMINAL has air capability (heat, cool, or both), and whether the capability operates in *set temperature* mode (tuTH and/or tuTLh given) or *set output* mode (tuVfMn given without tuTH and tuTLh). They further establish the setpoints, flow limits, leakages, and losses.

Caution should be exercised in using air heat and air cooling in the same terminal. The supply air for both comes from the same air handler; it is up to you to make sure the terminal only calls for heat when the air handler is blowing hot air and only calls for cooling when the air handler is blowing cold air. This is done by carefully coordinating the variable expressions for terminal air heating and cooling setpoints (tuTH and tuTC here) and the air handler supply temperature setpoint AIRHANDLER ahTsSp, Section 0).

tuAh=*ahName*

Name of air handler supplying this terminal.

Unit s

Legal Range

Default

Requi red

Variabi lity

Btu/F

name of an AIRHANDLER

If omitted, terminal has no air heating nor cooling capability.

No

constant

If both of the following (tuTH and tuTC) are specified, be careful not to accidentally permit the heating setpoint to be active when the air handler is blowing cold air, or vice versa. CSE's simulated thermostats and VAV boxes are at least as dumb as their real counterparts; if the thermostat calls for heat, the VAV damper will open even if the supply air is colder than the zone. To schedule deactivation of the air heating or cooling capability, schedule an extreme setpoint, such as 1 for heating or 199 for cooling.

Giving neither tuTH nor tuTC implies that the terminal has no *set temperature* air capability; it will then have *set output* air capability if tuVfMn is given.

tuTH=*float*

Air heating thermostat set point; implies *set temperature* air capability. May be scheduled as desired with an hourly expression; to disable set temperature operation at certain times (as when air handler is scheduled to supply cold air), schedule a low temperature such as 1.

Units	Legal Range	Default	Required	Variability
F	$x \geq 0$	No thermostat-controlled air heating	No	hourly

tuTC=*float*

Air cooling thermostat set point; implies *set temperature*air capability. May be scheduled as desired; to disable at certain times, schedule an extreme temperature such as 199.

Units	Legal Range	Default	Required	Variability
F	$x \geq 0$	No thermostat-controlled air cooling	No	hourly

tuVfDs=*float*

Design air flow rate. ("Vf" in member names stands for "Volumetric Flow", to emphasize that flow is specified by volume at actual air temperature (cfm), not by mass (lb/hr), nor heat capacity (Btuh/F), etc.)

The design air flow rate is used to apportion the available cfm among the terminals when the total flow demand of the terminals exceeds the air handler's supply fan capacity; it is unimportant (but may nevertheless be required) if the total of the tuVfMx's of the terminals on the air handler is not greater than the air handler's fan capacity including overrun.

CSE will default tuVfDs to the largest of tuVfMn, tuVfMxH, and tuVfMxC unless a variable expression is given for any of them. Thus, you must given tuVfDs only when a variable minimum or maximum flow is used, or when you wish to override the default cfm apportionment under fan overload conditions.

Uni ts

Legal Range

Default

Required

Variab ility

cfm

x 0

largest of tuVfMn, tuVfMxH, and tuVfMxC if all are constant

Yes, if tuVfMn, tuVfMxH, or tuVfMxC is variable

hourly

tuVfMn=*float*

Minimum terminal air flow rate or set output air flow rate. An hourly expression may be used to schedule the minimum or set output flow as desired.

If neither tuTH nor tuTC is given, giving tuVfMn implies *set output* air capability for the terminal; the tvVfMn value is the set output cfm.

If either setpoint (tuTH or tuTC) is given, tuVfMn is the cfm used when the thermostat is not calling for heat nor cold; it might be non-0, for example, to meet ventilation requirements. If tuVfMn is larger than tuVfMxH (when heating) or tuVfMxC (when cooling), it overrules them; thus a minimum (e.g. ventilation) requirement need not be considered in formulating expressions for the maximum flows.

Unit s

Legal Range

Default

Required

Variabil ity

cfm

x 0

0 if tuTH or tuTC given, else no air heat/cool

For set output air operation

hourly

tuVfMxH=*float*

Maximum heating air flow rate, subject to air handler limitations. This terminal flow is used when the thermostat is calling for heat continuously. Hourly schedulable. If not greater than tuVfMn, the latter flow is used, thus disabling thermostat control.

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	*none*	If *tuTH* given	hourly

tuVfMxC=*float*

Maximum cooling air flow rate, before air handler limitations, used when the thermostat is calling for cooling continuously. tuVfMn overrides if larger.

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	*none*	If *tuTC* given	hourly

tuPriC=*int*

Cool setpoint priority. The lowest numbered priority is used first when there are equal setpoints in a zone; equal heat or cool setpoints with equal priority in same ZONE (even if on different TERMINAL constitute an error.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	No	constant

tuPriH=*int*

Heat setpoint priority. Lowest numbered priority is used first when there are equal setpoints in a zone. Default for tuPriLh is larger, so that by default local heat is not used unless maximum air heat is insufficient, when both local heat and air heat are present in zone and have same setpoint.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	No	constant

tuSRLeak=*float*

Leakage of supply air to return, increasing supply volume and return temperature. Note that this is a fraction of current cfm, whereas air handler leak (before VAV dampers) is a fraction of *maximum* cfm. TfanOffLeak is added to this if terminal has a fan that is not running (future, 7-92).

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.05	No	constant

tuSRLoss=*float*

Supply air to return plenum heat loss as a fraction of supply air to return air temperature difference. Not allowed if return is ducted (no plenum).

NOT IMPLEMENTED as of July 1992 – Plenums are unimplemented.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.1	No	constant

5.7.11.3 TERMINAL Heating Coil

These members are disallowed if terminal has no local heating capability, that is, if neither tuTLh nor tuQMnLh is given.

tuhcType=*choice*

Local heating coil type:

ELECTRIC

Electric coil or heater, including separate heaters such as electric baseboards. 100% efficient; rated capacity always available.

HW

Hot water coil, using hot water from a HEATPLANT. Available capacity may be limited by HEATPLANT total capacity as well as by coil rated capacity.

Units	Legal Range	Default	Required	Variability
	ELECTRIC (future: HW)	ELECTRIC, or NONE if no local heat	No	constant

tuhcCaptRat=*float*

Rated capacity of the heating coil. The coil will never supply more heat than its capacity, even if tuQMxLh and/or tuQMnLh is greater. For an ELECTRIC coil, the capacity is always the rated capacity. For an HW coil, the capacity is the rated capacity when the HEATPLANT can supply it; when the total heat demanded from the HEATPLANT by all the HW coils in TERMINAL and AIRHANDLER exceeds the HEATPLANT capacity, CSE reduces the capacities of all HW coils proportionately until the plant is not overloaded.

Units	Legal Range	Default	Required	Variability
Btu/hr	$x > 0$	*none*	Yes	constant

tuhcMtr=*mtrName*

Name of meter, if any, which accumulates input energy for this coil. End use category used is "Htg". Not allowed when tuhcType is HW, as the energy for an HW coil comes through a HEATPLANT the input energy is accumulated to a meter by the HEATPLANT

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

tuhcHeatplant=heat*plantName*

Name of HEATPLANT for HW coil; disallowed for other coil types.

Units	Legal Range	Default	Required	Variability
	name of a HEATPLANT	*none*	If tuhcType is HW	constant

5.7.11.4 TERMINAL Fan

Presence of a terminal fan is indicated by specifying a tfanType value other than NONE.

Terminal fans are *NOT IMPLEMENTED* as of July 1992.

tfanType=*choice*

Choice of:

NONE

No fan in this TERMINAL (default); input for other terminal fan members disallowed.

SERIES

Fan runs whenever scheduled ON (see tfanSched, next); if VAV cfm < terminal fan cfm (tfanVfDs), the additional flow comes from the return air.

PARALLEL

Fan runs when scheduled ON (see tfanSched) and terminal's simulated VAV cfm is less than tfanVfDs plus tuVfMn ?? plus tuVfMn??. Terminal fan cfm is added to VAV cfm from AIRHANDLER to get cfm to ZONE.

Units	Legal Range	Default	Required	Variability
	NONE, SERIES, PARALLEL	NONE	Yes, if fan present	constant

tfanSched=*choice*

Terminal fan schedule. May be scheduled with an hourly variable expression.

OFF

fan does not run

ON

fan may run

HEATING

fan may run when local heat is in use

VAV

fan may run when AIRHANDLER supply fan is on or when doing setback headting and ssCtrl is ZONE_HEAT or BOTH (future).

A series fan (see tfanType) runs whenever on; a parallel fan runs only enough to keep terminal cfm at terminal minimum plus fan cfm; thus it may not run at all when the VAV flow from the AIRHANDLER is sufficient.

Units	Legal Range	Default	Required	Variability
	OFF, ON, HEATING, VAV	*none*	Yes (if fan present)	hourly

tfanOffLeak=*float*

Backdraft leakage when terminal fan off., as a fraction of tfanVfDs.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.1 if fan present	No	constant

tfanVfDs=*float*

Terminal fan design flow rate. To specify .x times zone or terminal cfm, use a CSE expression.

Units	Legal Range	Default	Required	Variability
cfm	$x \leq 0$	*none*	Yes (if fan present)	constant

tfanPress=*float*

Terminal fan external static pressure.

Units	Legal Range	Default	Required	Variability
inches H2O	$x \geq 0$	0.3	No	constant

tfanEff=*float*

Terminal fan/motor/drive combined efficiency.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.08	No	constant

tfanCurvePy=*k0, k1, k2, k3, x0*

k0 through *k3* are the coefficients of a cubic polynomial for the curve relating fan relative energy consumption to relative air flow above the minimum flow *x0*. Up to five *floats* may be given, separated by commas. 0 is used for any omitted trailing values. The values are used as follows:

$$z = k_0 + k_1 \cdot (x - x_0)| + k_2 \cdot (x - x_0)|^2 + k_3 \cdot (x - x_0)|^3$$

where:

- x is the relative fan air flow (as fraction of *tfanVfDs*; $0 \leq x \leq 1$);
- x_0 is the minimum relative air flow (default 0);
- $(x - x_0)|$ is the "positive difference", i.e. $(x - x_0)$ if $x > x_0$; else 0;
- z is the relative energy consumption.

If z is not 1.0 for $x = 1.0$, a warning message is displayed and the coefficients are normalized by dividing by the polynomial's value for $x = 1.0$.

Units	Legal Range	Default	Required	Variability
		0, 1, 0, 0, 0 (linear)	No	constant

tfanMtr=*mtrName*

Name of meter, if any, which is to record energy used by this terminal fan. The "fans" category is used.

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endTerminal

Optional to indicates the end of terminal definition. Alternatively, the end of the door definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.8 IZXFER

IZXFER constructs an object that represents an interzone or zone/ambient heat transfer due to conduction and/or air transfer. The air transfer modeled by IZXFER transfers heat only; humidity transfer is not modeled as of July 2011. Note that SURFACE is the preferred way represent conduction between ZONE

The AIRNET types are used in a multi-cell pressure balancing model that finds zone pressures that produce net 0 mass flow into each zone. The model operates in concert with the znType=CZM to represent ventilation strategies. During each time step, the pressure balance is found for two modes that can be thought of as "VentOff" (or infiltration-only) and "VentOn" (or infiltration+ventilation). The zone model then determines the ventilation fraction required to hold the desired zone temperature (if possible).

izName

Optional name of interzone transfer; give after the word IZXFER if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

izNVType=*choice*

Choice determining interzone ventilation

NONE

No interzone ventilation

TWOWAY

Uncontrolled flow in either direction (using ASHRAE high/low vent model)

AIRNETIZ

Single opening to another zone (using pressure balance AirNet model)

AIRNETEXT

Single opening to ambient (using pressure balance AirNet model)

AIRNETHORIZ

Horizontal (large) opening between two zones, used to represent e.g. stairwells

AIRNETEXTFAN

Fan from exterior to zone (flow either direction).

AIRNETIZFAN

Fan between two zones (flow either direction).

AIRNETEXTFLOW

Specified flow from exterior to zone (either direction). Behaves identically to AIRNETEXTFAN except no electricity is consumed and no fan heat is added to the air stream.

AIRNETIZFLOW

Specicified flow between two zones (either direction). Behaves identically to AIRNETIZFAN except no electricity is consumed and no fan heat is added to the air stream

Uni ts

Legal Range

Defa ult

Requ ired

Variab ility

NONE, TWOWAY, AIRNET, AIRNETEXT, AIRNETHORIZ, AIRNETEXTFAN, AIRNETIZFAN, AIRNETEXTFLOW, AIRNETIZFLOW

None

No

constant

izZn1=*znName*

Name of primary zone. Flow rates > 0 are into the primary zone.

Units	Legal Range	Default	Required	Variability
	name of a ZONE		Yes	constant

izZn2=*znName*

Name of secondary zone.

Unit s

Legal Range

Defau lt

Required

Variabi lity

name of a ZONE

required unless izNVType = AIRNETEXT,AIRNETEXTFAN, orAIRNETEXTFLOW

constant

Give izHConst for a conductive transfer between zones. Give izNVType other than NONE and the following variables for a convective (air) transfer between the zones or between a zone and outdoors. Both may be given if desired. ?? Not known to work properly as of July 2011

izHConst=*float*

Conductance between zones.

Units	Legal Range	Default	Required	Variability
Btu/°F	$x \geq 0$	0	No	hourly

izALo=*float*

Area of low or only vent (typically VentOff)

Units	Legal Range	Default	Required	Variability
ft^2	$x \geq 0$	0	No	hourly

izAHi=*float*

Additional vent area (high vent or VentOn). If used in AIRNET, izAHi > izALo typically but this is not required.

Units	Legal Range	Default	Required	Variability
ft^2	$x \geq 0$	izALo	No	hourly

izL1=*float*

Length or width of AIRNETHORIZ opening

Units	Legal Range	Default	Required	Variability
ft	$x > 0$		if izNVType = AIRNETHORIZ	constant

izL2=*float*

width or length of AIRNETHORIZ opening

Units	Legal Range	Default	Required	Variability
ft	$x > 0$		if izNVType = AIRNETHORIZ	constant

izStairAngle=*float*

stairway angle for AIRNETHORIZ opening. Use 90 for open hole. Note that 0 prevents flow.

Units	Legal Range	Default	Required	Variability
degrees	$x > 0$	34	No	constant

izHD=*float*

Vent center-to-center height difference (for TWOWAY) or vent height above nominal 0 level (for AirNet types)

Units	Legal Range	Default	Required	Variability
ft		0	No	constant

izNVEff=*float*

Vent discharge coefficient coefficient.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.8	No	constant

izfanVfDs=*float*

Fan design or rated flow at rated pressure

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	0 (no fan)	If fan present	constant

izCpr=*float*

Wind pressure coefficient (for AIRNETEXT)

Units	Legal Range	Default	Required	Variability
??	$x \geq 0$	0.	No	constant

izExp=*float*

Opening exponent

Units	Legal Range	Default	Required	Variability
none	$x > 0$	0.5	No	constant

izfanVfMin=*float*

Minimum volume flow rate (VentOff mode)

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	0	No	subhourly

izfanVfMax=*float*

Maximum volume flow rate (VentOn mode)

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	0	No	subhourly

izfanPress=*float*

Design or rated pressure.

Units	Legal Range	Default	Required	Variability
inches H_2O	$x > 0$	.3	No	constant

Only one of izfanElecPwr, izfanEff, and izfanShaftBhp may be given: together with izfanVfDs and izfanPress, any one is sufficient for CSE to detemine the others and to compute the fan heat contribution to the air stream.

izfanElecPwr=*float*

Fan input power per unit air flow (at design flow and pressure).

Unit s

Legal Range

Default

Required

Variabi lity

W/cfm

x > 0

derived from izfanEff and izfanShaftBhp

If izfanEff and izfanShaftBhp not present

constant

izfanEff=*float*

Fan efficiency at design flow and pressure, as a fraction.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	derived from *izfanShaftBhp* if given, else 0.08	No	constant

izfanShaftBhp=*float*

Fan shaft brake horsepower at design flow and pressure.

Units	Legal Range	Default	Required	Variability
bhp	$x > 0$	derived from *izfanEff*.	No	constant

izfanCurvePy=k_0, k_1, k_2, k_3, x_0

k_0 through k_3 are the coefficients of a cubic polynomial for the curve relating fan relative energy consumption to relative air flow above the minimum flow x_0. Up to five *floats* may be given, separated by commas. 0 is used for any omitted trailing values. The values are used as follows:

$$z = k_0 + k_1 \cdot (x - x_0)| + k_2 \cdot (x - x_0)|^2 + k_3 \cdot (x - x_0)|^3$$

where:

- x is the relative fan air flow (as fraction of izfan *VfDs*; $0 \leq x \leq 1$);
- x_0 is the minimum relative air flow (default 0);
- $(x - x_0)|$ is the "positive difference", i.e. $(x - x_0)$ if $x > x_0$; else 0;
- z is the relative energy consumption.

If z is not 1.0 for $x = 1.0$, a warning message is displayed and the coefficients are normalized by dividing by the polynomial's value for $x = 1.0$.

Units	Legal Range	Default	Required	Variability
		0, 1, 0, 0, 0 (linear)	No	constant

izfanMtr=*mtrName*

Name of meter, if any, to record energy used by supply fan. End use category used is "Fan".

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endIzxfer

Optionally indicates the end of the interzone transfer definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.9 RSYS

RSYS constructs an object representing an air-based residential HVAC system.

rsName

Optional name of HVAC system; give after the word RSYS if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

rsType=*choice*

Specifies type of system.

ACFURNACE

Compressor-based cooling and fuel-fired heating. Primary heating input energy is accumulated to end use HTG of meter rsFuelMtr.

ACRESISTANCE

Compressor-based cooling and electric ("strip") heating. Primary heating input energy is accumulated to end use HTG of meter rsElecMtr.

ASHP

Air-source heat pump (compressor-based heating and cooling). Primary (compressor) heating input energy is accumulated to end use HTG of meter rsElecMtr. Auxililary heating input energy is accumulated to end use HPHTG of meter rsElecMtr.

AC

Compressor-based cooling; no heating.

FURNACE

Fuel-fired heating. Primary heating input energy is accumulated to end use HTG of meter rsFuelMtr.

RESISTANCE

Electric heating. Primary heating input energy is accumulated to end use HTG of meter rsElecMtr

Units

Legal Range

Defaul t

Require d

Variabili ty

ACFURNACE, ACRESISTANCE, ASHP, AC, FURNACE, RESISTANCE

ACFURNAC E

No

constant

rsDesc=*string*

Text description of system, included as documentation in debugging reports such as those triggered by rsPerfMap=YES

Units	Legal Range	Default	Required	Variability
	string	*blank*	No	constant

rsModeCtrl=*choice*

Specifies systems heating/cooling availability during simulation.

OFF

System is off (neither heating nor cooling is available)

HEAT

System can heat (assuming rsType can heat)

COOL

System can cool (assuming rsType can cool)

AUTO

System can either heat or cool (assuming rsType compatibility). First request by any zone served by this RSYS determines mode for the current time step.

Units	Legal Range	Default	Required	Variability
	OFF, HEAT, COOL, AUTO	AUTO	No	hourly

rsPerfMap=*choice*

Generate performance map(s) for this RSYS Comma-separated text is written to file PM_.csv. This is a debugging capabilitiy that is not necessarily maintained.

Units	Legal Range	Default	Required	Variability
	NO, YES	NO	No	constant

rsFanTy=*choice*

Specifies fan (blower) position relative to cooling coil.

Units	Legal Range	Default	Required	Variability
	BLOWTHRU, DRAWTHRU	BLOWTHRU	No	constant

rsFanMotTy=*choice*

Specifies type of motor driving the fan (blower). This is used in the derivation of the coil-only cooling capacity for the RSYS

PSC	Permanent split capacitor
BPM	Brushless permanent magnet (aka ECM)

Units	Legal Range	Default	Required	Variability
	PSC, BPM	PSC	No	constant

rsElecMtr=*mtrName*

Name of METER object, if any, by which system's electrical energy use is recorded (under appropriate end uses).

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

rsFuelMtr =*mtrName*

Name of METER object, if any, by which system's fuel energy use is recorded (under appropriate end uses).

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

rsAFUE=*float*

Heating Annual Fuel Utilization Efficiency (AFUE).

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1$	0.9 if furnace, 1.0 if resistance	No	constant

rsCapH=*float*

Heating capacity, used when rsType is ACFURNACE, ACRESISTANCE, FURNACE, or RESISTANCE.

Units	Legal Range	Default	Required	Variability
Btu/hr	*AUTOSIZE* or $x \geq 0$	0	No	constant

rsTdDesH=*float*

Nominal heating temperature rise (across system, not zone) used during autosizing (when capacity is not yet known).

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	30 if ASHP else 50	No	constant

rsFxCapH=*float*

Heating autosizing capacity factor. rsCapH is set to rsFxCapH × (peak design-day load). Peak design-day load is the heating capacity that holds zone temperature at the thermostat set point during the *last substep* of all hours of all design days.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1.4	No	constant

rsFanPwrH=*float*

Heating fan power. Heating air flow is estimated based on a 50 °F temperature rise.

Units	Legal Range	Default	Required	Variability
W/cfm	$x \geq 0$	.365	No	constant

rsHSPF=*float*

For rsType=ASHP, Heating Seasonal Performance Factor (HSPF).

Units	Legal Range	Default	Required	Variability
Btu/Wh	$x > 0$		Yes if rsType=ASHP	constant

rsCap47=*float*

For rsType=ASHP, rated heating capacity at outdoor dry-bulb temperature = 47 °F. If both rsCap47 and rsCapC are autosized, both are set to the larger consistent value.

Units	Legal Range	Default	Required	Variability
Btu/Wh	*AUTOSIZE* or $x > 0$	Estimate from rsCapC	no	constant

rsCap35=*float*

For rsType=ASHP, rated heating capacity at outdoor dry-bulb temperature = 35 °F.

Units	Legal Range	Default	Required	Variability
Btu/Wh	$x > 0$	Estimated from rsCap47 and rsCap17	no	constant

rsCap17=*float*

For rsType=ASHP, rated heating capacity at outdoor dry-bulb temperature = 17 °F.

Units	Legal Range	Default	Required	Variability
Btu/Wh	$x > 0$	Estimated from rsCap47	no	constant

rsCOP47=*float*

For rsType=ASHP, rated heating coefficient of performance at outdoor dry-bulb temperature = 47 °F.

Units	Legal Range	Default	Required	Variability
	$x > 0$	Estimated from rsHSPF, rsCap47, and rsCap17	no	constant

rsCOP35=*float*

For rsType=ASHP, rated heating coefficient of performance at outdoor dry-bulb temperature = 35 °F.

Unit s

Legal Range

Default

Requir ed

Variabil ity

x > 0

Estimated from rsCap35, rsCap47, rsCap17, rsCOP47, and rsCOP17

no

constant

rsCOP17=*float*

For rsType=ASHP, rated heating coefficient of performance at outdoor dry-bulb temperature = 17 °F.

Units	Legal Range	Default	Required	Variability
	$x > 0$	Estimated from rsHSPF, rsCap47, and rsCap17	no	constant

rsCapAuxH=*float*

For rsType=ASHP, auxiliary electric ("strip") heating capacity. If autosized, rsCapAuxH is set to the peak heating load in excess of heat pump capacity evaluated at the heating design temperature (Top.heatDsTDbO).

Units	Legal Range	Default	Required	Variability
Btu/hr	*AUTOSIZE* or $x \geq 0$	0	no	constant

rsCOPAuxH=*float*

For rsType=ASHP, auxiliary electric ("strip") heating coefficient of performance. Energy use for auxiliary heat is accumulated to end use HPHTG of meter rsElecMtr (that is, auxiliary heat is assumed to be electric).

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	1.0	no	constant

rsSEER=*float*

Cooling rated Seasonal Energy Efficiency Ratio (SEER).

Units	Legal Range	Default	Required	Variability
Btu/Wh	$x > 0$		Yes	constant

rsEER=*float*

Cooling Energy Efficiency Ratio (EER) at standard AHRI rating conditions (outdoor drybulb of 95 °F and entering air at 80 °F drybulb and 67 °F wetbulb).

Units	Legal Range	Default	Required	Variability
Btu/Wh	$x > 0$	Estimated from SEER	no	constant

rsCapC=*float*

Cooling capacity at standard AHRI rating conditions. If rsType=ASHP and both rsCapC and rsCap47 are autosized, both are set to the larger consistent value.

Units

Legal Range

Default

Required

Variability

Btu/hr

AUTOSIZE or x 0 (x > 0 coverted to < 0)

Yes if rsType includes cooling

constant

rsTdDesC=*float*

Nominal cooling temperature fall (across system, not zone) used during autosizing (when capacity is not yet known).

Units	Legal Range	Default	Required	Variability
°F	$x < 0$	-25	No	constant

rsFxCapC=*float*

Cooling autosizing capacity factor. rsCapC is set to rsFxCapC × (peak design-day load). Peak design-day load is the cooling capacity that holds zone temperature at the thermostat set point during the *last substep* of all hours of all design days.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1.4	No	constant

rsFChg=*float*

Refrigerant charge adjustment factor.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	no	constant

Units	Legal Range	Default	Required	Variability

rsFSize=*float*

Compressor sizing factor.

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	no	constant

rsVFPerTon=*float*

Standard air volumetric flow rate per nominal ton of cooling capacity.

Units	Legal Range	Default	Required	Variability
cfm/ton	$150 \leq x \leq 500$	350	no	constant

rsFanPwrC=*float*

Cooling fan power.

Units	Legal Range	Default	Required	Variability
W/cfm	$x \geq 0$	.365	No	constant

rsRhIn=*float*

Entering air relative humidity (for model testing).

Units	Legal Range	Default	Required	Variability
W/cfm	$0 \leq x \leq 1$	Derived from entering air state	No	constant

rsTdbOut=*float*

Air dry-bulb temperature at the outdoor portion of this system.

Units	Legal Range	Default	Required	Variability
°F		From weather file	No	hourly

endRSYS

Optionally indicates the end of the RSYS definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.10 DUCTSEG

DUCTSEG defines a duct segment. Each RSYS has at most one return duct segment and at most one supply duct segment. That is, DUCTSEG input may be completely omitted to eliminate duct losses.

dsTy=*choice*

Duct segment type.

Units	Legal Range	Default	Required	Variability
	SUPPLY, RETURN		Yes	constant

The surface area of a DUCTSEG depends on its shape. 0 surface area is legal (leakage only). DUCTSEG shape is modeled either as flat or round –

- dsExArea specified: Flat. Interior and exterior areas are assumed to be equal (duct surfaces are flat and corner effects are neglected).
- dsExArea *not* specified: Round. Any two of dsInArea, dsDiameter, and dsLength must be given. Insulation thickness is derived from dsInsulR and dsInsulMat and this thickness is used to calculate the exterior surface area. Overall inside-to-outside conductance is also calculated including suitable adjustment for curvature.

dsExArea=*float*

Duct segment surface area at outside face of insulation for flat duct shape, see above.

Units	Legal Range	Default	Required	Variability
ft^2	$x \geq 0$		No	constant

dsInArea=*float*

Duct segment inside surface area (at duct wall, duct wall thickness assumed negligible) for round shaped duct.

Units

Legal Range

Default

Required

Variabili ty

ft2

x 0

Derived from dsDiameter and dsLength

(see above re duct shape)

constant

dsDiameter=*float*

Duct segment round duct diameter (duct wall thickness assumed negligible)

Units

Legal Range

Default

Required

Variabili ty

ft

x 0

Derived from dsInArea and dsLength

(see above re duct shape)

constant

dsLength=*float*

Duct segment length.

Units

Legal Range

Default

Required

Variabili ty

ft

x 0

Derived from dsInArea and dsDiameter

(see above re duct shape)

constant

dsExCnd=*choice*

Conditions surrounding duct segment.

Units

Legal Range

Default

Required

Variability

ADIABATIC, AMBIENT, SPECIFIEDT, ADJZN

ADJZN

No

constant

dsAdjZn=*znName*

Name of zone surrounding duct segment; used only when dsExCon is ADJZN. Can be the same as a zone served by the RSYS owning the duct segment.

Units	Legal Range	Default	Required	Variability
	name of a *ZONE*	*none*	Required when *dsExCon* = ADJZN	constant

dsEpsLW=*float*

Exposed (i.e. insulation) outside surface exterior long wave (thermal) emittance.

Units	Legal Range	Default	Required	Variability
(none)	$0 \leq x \leq 1$	0.9	No	constant

dsExT=*float*

Air dry-bulb temperature surrounding duct segment.

Units	Legal Range	Default	Required	Variability
°F	*unrestricted*	*none*	Required if *sfExCon* = SPECIFIEDT	hourly

Duct insulation is modeled as a pure conductance (no mass).

dsInsulR=*float*

Insulation thermal resistance *not including*surface conductances. dsInsulR and dsInsulMat are used to calculate insulation thickness (see below).

Units	Legal Range	Default	Required	Variability
ft^2-F-hr / Btu	$x \geq 0$	0	No	constant

dsInsulMat=*matName*

Name of insulation MATERIAL The conductivity of this material at 70 °F is combined with dsInsulR to derive the duct insulation thickness. If omitted, a typical fiberglass material is assumed having conductivity of 0.025 Btu/hr-ft^2-F at 70 °F and a conductivity coefficient of .00418 1/F (see MATERIAL In addition, insulation conductivity is adjusted during the simulation in response its average temperature.

Units	Legal Range	Default	Required	Variability
	name of a *MATERIAL*	fiberglass	No	constant

dsLeakF=*float*

Duct leakage. Return duct leakage is modeled as if it all occurs at the segment inlet. Supply duct leakage is modeled as if it all occurs at the outlet.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1$		No	constant

dsExH=*float*

Outside (exposed) surface convection coefficient.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	$x > 0$	.54	No	subhourly

endDuctSeg

Optionally indicates the end of the DUCTSEG definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.11 DHWSYS

DHWSYS constructs an object representing a domestic hot water system consisting of one or more hot water heaters, storage tanks, loops, and pumps DHWHEATER DHWTANK DHWLOOP and DHWPUMP see below) and a distribution system characterized by loss parameters. This model is based on Appendix B of the 2016 Residential ACM Reference Manual. This version is preliminary, revisions are expected.

The parent-child structure of DHWSYS components is determined by input order. For example, DHWHEATER belong to DHWSYS that preceeds them in the input file. The following hierarchy shows the relationship among components. Note that any of the commands can be repeated any number of times.

- DHWSYS
  - DHWHEATER
  - DHWTANK
  - DHWPUMP
  - DHWLOOP
    - DHWLOOPPUMP
    - DHWLOOPSEG
      - DHWLOOPBRANCH

No actual controls are modeled. For example, if several DHWHEATER are included in a DHWSYS an equal fraction of the required hot water is assumed to be produced by each heater, even if they are different types or sizes. Thus a DHWSYS is in some ways just a collection of components, rather than a physically realistic system.

wsName

Optional name of system; give after the word DHWSYS if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wsMult=*integer*

Number of identical systems of this type (including all child objects). Any value > 1 is equivalent to repeated entry of the same DHWSYS

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wsTInlet=*float*

Specifies cold (mains) water temperature supplied to DHWHEATER in this DHWSYS

Units	Legal Range	Default	Required	Variability
°F	> 32 °F	Mains temp from weather file	No	hourly

wsUse=*float*

Specifies hourly hot water use (at the point of use)

Units	Legal Range	Default	Required	Variability
gal	≥ 0	0	No	hourly

wsTUse=*float*

Specifies hot water use temperature (at the point of use)

Units	Legal Range	Default	Required	Variability
°F	> 32 °F	130	No	hourly

wsParElec=*float*

Specifies electrical parasitic power to represent recirculation pumps or other system-level electrical devices. Calculated energy use is accumulated to the METER specified by wsElecMtr (end use DHW). No other effect, such as heat gain to surroundings, is modeled.

Units	Legal Range	Default	Required	Variability
W	> 0	0	No	hourly

wsSDLM=*float*

Specifies the standard distribution loss multiplier. See App B Eqn 4. To duplicate CEC 2016 methods, this value should be set according to the value derived with App B Eqn 5.

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wsDSM=*float*

Specifies the distribution system multiplier. See App B Eqn 4. To duplicate CEC 2016 methods, wsDSM should be set to the appropriate value from App B Table B-2. Note the NCF (non-compliance factor) included in App B Eqn 4 is *not* a CSE input and thus must be applied externally to wsDSM.

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wsSSF=*float*

Specifies the solar savings fraction.

Units	Legal Range	Default	Required	Variability
	≥ 0	0	No	hourly

wsHRDL=*float*

Specifies the hourly recirculation distribution loss. TODO: the implementation will be expanded to evaluate HRDL during the simulation to allow use of hourly values.

Units	Legal Range	Default	Required	Variability
	≥ 0	0	No	constant

wsElecMtr=*mtrName*

Name of METER object, if any, by which DHWSYS electrical energy use is recorded (under end use DHW). In addition, wsElecMtr provides the default whElectMtr selection for all DHWHEATER and DHWPUMP in this DHWSYS

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

wsFuelMtr =*mtrName*

Name of METER object, if any, by which DHWSYS fuel energy use is reco*rded (under end use DWH). DHWSYS fuel use is usually (always?) 0, so the primary use of this input is to specify the default whFuelMtr choice for DHWHEATER in this DHWSYS

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

wsCalcMode=*choice*

PRERUN

Calculate hot water heating load; at end of run, derive whLDEF for all child DHWHEATERs for which that value is required and defaulted. This procedure emulates methods used in the T24DHW.DLL implementation of CEC DHW procedures.

SIMULATE

Perform full modeling calculations

Units	Legal Range	Default	Required	Variability
	Codes listed above	SIMULATE	No	

endDHWSys

Optionally indicates the end of the DHWSYS definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.12 DHWHEATER

DHWHEATER constructs an object representing a domestic hot water heater (or several if identical).

whName

Optional name of water heater; give after the word DHWHEATER if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

whMult=*integer*

Number of identical water heaters of this type. Any value > 1 is equivalent to repeated entry of the same DHWHEATER

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

whType=*choice*

Specifies type of water heater.

SMALLSTORAGE

A gas-fired storage water heater with input of 75,000 Btuh or less, an oil-fired storage water heater with input of 105,000 Btuh or less, an electric storage water heater with input of 12 kW or less, or a heat pump water heater rated at 24 amps or less.

LARGESTORAGE

Any storage water heater that is not SMALLSTORAGE.

SMALLINSTANTANEOUS

A water heater that has an input rating of at least 4,000 Btuh per gallon of stored water. Small instantaneous water heaters include: gas instantaneous water heaters with an input of 200,000 Btu per hour or less, oil instantaneous water heaters with an input of 210,000 Btu per hour or less, and electric instantaneous water heaters with an input of 12 kW or less.

LARGEINSTANTANEOUS

An instantaneous water heater that does not conform to the definition of SMALLINSTANTANEOUS, an indirect fuel-fired water heater, or a hot water supply boiler.

Units	Legal Range	Default	Required	Variability
	Codes listed above	SMALLSTORAGE	No	constant

whHeatSrc=*choice*

Specifies heat source for water heater.

RESISTANCE

Electric resistance heating element.

ASHP

Air source heat pump

ASHPX

Air source heat pump (detailed HPWH model)

FUEL

Fuel-fired burner

Units	Legal Range	Default	Required	Variability
	Codes listed above	FUEL	No	constant

whVol=*float*

Specifies tank volume. Documentation only. Must be omitted or 0 for instantaneous whTypes.

Units	Legal Range	Default	Required	Variability
gal	≥ 0	0	No	constant

whEF=*float*

Rated energy factor, used in modeling whType SMALLSTORAGE and SMALLINSTANTANEOUS.

Units

Legal Range

Default

Required

**Variability

> 0

.82

When whType = SMALLSTORAGE and whLDEF not specified or SMALLINSTANTANEOUS

constant

whLDEF=*float*

Load-dependent energy factor, used in modeling whType SMALLSTORAGE. Can

Units

Legal Range

Default

Required

Variability

> 0

Calculated via DHWSYS PreRun mechanism

When whType = SMALLSTORAGE and PreRun not used

constant

whZone=*znName*

Name of zone where water heater is located. Zone conditions are for tank heat loss calculations and default for whASHPSrcZn (see below). No effect unless whHeatSrc = ASHPX. whZone and whTEx cannot both be specified.

Units	Legal Range	Default	Required	Variability
	name of a ZONE	Not in zone	No	constant

whTEx=*float*

Water heater surround temperature. No effect unless whHeatSrc=ASHPX. whZone and whTEx cannot both be specified.

Units	Legal Range	Default	Required	Variability
°F	≥ 0	whZone air temperature if specified, else 70 °F	No	hourly

whASHPType=*choice*

Specifies type of air source heat pump, valid only if whHeatSrc=ASHPX. These choice are those supported by Ecotope HPWH model.

Voltex60

Voltex80

GEGeospring

BasicIntegrated

Typical integrated HPWH

ResTank

Resistance-only water heater (no compressor)

Units	Legal Range	Default	Required	Variability
	Codes listed above	–	When whHeatSrc=ASHPX	constant

whASHPSrcZn=*znName*

Name of zone that serves as heat source when whHeatSrc = ASHPX. Used for tank heat loss calculations and default for whASHPSrcZn. Illegal unless whHeatSrc = ASHPX. whASHPSrcZn and whASHPSrcT cannot both be specified.

Units

Legal Range

Default

Required

Variability

name of a ZONE

Same as whZone if whASHPSrcT not specified. If no zone is specified by input or default, heat extracted by ASHP has no effect.

No

constant

whASHPSrcT=*float*

ASHP source air temperature. Illegal unless whHeatSrc=ASHPX. whASHPSrcZn and whASHPSrcT cannot both be specified.

Units	Legal Range	Default	Required	Variability
°F	≥ 0	whASHPZn air temperature if specified, else 70 °F	No	hourly

whHPAF=*float*

Heat pump adjustment factor, used when modeling whType=SMALLSTORAGE and whHeatSrc=ASHP. This value should be derived according to App B Table B-6.

Units	Legal Range	Default	Required	Variability
	> 0	1	When whType=SMALLSTORAGE and whHeatSrc = ASHP	constant

whEff=*float*

Water heating efficiency, used in modeling LARGESTORAGE and LARGEINSTANTANEOUS.

Units	Legal Range	Default	Required	Variability
	$0 <$ whEff ≤ 1	.82	No	constant

whSBL=*float*

Standby loss, used in modeling LARGESTORAGE

Units	Legal Range	Default	Required	Variability
Btuh	≥ 0	0	No	constant

whPilotPwr=*float*

Pilot light consumption, included in energy use of DHWHEATER with whHeatSrc=FUEL.

Units	Legal Range	Default	Required	Variability
Btuh	≥ 0	0	No	hourly

whParElec=*float*

Parasitic electricity power, included in energy use of all DHWHEATER

Units	Legal Range	Default	Required	Variability
W	≥ 0	0	No	hourly

whElecMtr=*mtrName*

Name of METER object, if any, by which DHWHEATER electrical energy use is recorded (under end use DHW).

Units	Legal Range	Default	Required	Variability
	name of a METER	*Parent DHWSYS wsElecMtr*	No	constant

whFuelMtr =*mtrName*

Name of METER object, if any, by which DHWHEATER fuel energy use is recorded.

Units	Legal Range	Default	Required	Variability
	name of a METER	*Parent DHWSYS wsFuelMtr*	No	constant

endDHW

HeaterOptionally indicates the end of the DHWHEATER definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.13 DHWTANK

DHWTANK constructs an object representing one or more unfired water storage tanks in a DHWSYS DHWTANK heat losses contribute to the water heating load.

wtName

Optional name of tank; give after the word DHWTANK if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wtMult=*integer*

Number of identical tanks of this type. Any value > 1 is equivalent to repeated entry of the same DHWTANK

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

Tank heat loss is calculated hourly (note that default heat loss is 0) –

$$\text{qLoss} = \text{wtMult} \cdot (\text{wtUA} \cdot (\text{wtTTank} - \text{wtTEx}) + \text{wtXLoss})$$

wtUA=*float*

Tank heat loss coefficient.

Units	Legal Range	Default	Required	Variability
Btuh/°F	≥ 0	Derived from wtVol and wtInsulR	No	constant

wtVol=*float*

Specifies tank volume.

Units	Legal Range	Default	Required	Variability
gal	≥ 0	0	No	constant

wtInsulR=*float*

Specifies total tank insulation resistance. The input value should represent the total resistance from the water to the surroundings, including both built-in insulation and additional exterior wrap insulation.

Units	Legal Range	Default	Required	Variability
ft^2-°F/Btuh	≥ .01	0	No	constant

wtTEx=*float*

Tank surround temperature.

Units	Legal Range	Default	Required	Variability
°F	≥ 0	70	No	hourly

wtTTank=*float*

Tank average water temperature.

Units	Legal Range	Default	Required	Variability
°F	> 32 °F	Parent DHWSYSTEM wsTUse	No	hourly

wtXLoss=*float*

Additional tank heat loss. To duplicate CEC 2016 procedures, this value should be used to specify the fitting loss of 61.4 Btuh.

Units	Legal Range	Default	Required	Variability
Btuh	(any)	0	No	hourly

endDHWTank

Optionally indicates the end of the DHWTANK definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.14 DHWPUMP

DHWPUMP constructs an object representing a domestic hot water circulation pump (or more than one if identical).

wpName

Optional name of pump; give after the word DHWPUMP if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wpMult=*integer*

Number of identical pumps of this type. Any value > 1 is equivalent to repeated entry of the same DHWPUMP

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wpPwr=*float*

Pump power.

Units	Legal Range	Default	Required	Variability
W	> 0	0	No	hourly

wpElecMtr=*mtrName*

Name of METER object, if any, to which DHWPUMP electrical energy use is recorded (under end use DHW).

Units	Legal Range	Default	Required	Variability
	name of a METER	*Parent DHWSYS wsElecMtr*	No	constant

endDHWPump

Optionally indicates the end of the DHWPUMP definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.15 DHWLOOP

DHWLOOP constructs one or more objects representing a domestic hot water circulation loop. The actual pipe runs in the DHWLOOP are specified by any number of DHWLOOPSEG (see below). Circulation pumps are specified by DHWLOOPPUMP (also below).

wlName

Optional name of loop; give after the word DHWLOOP if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wlMult=*integer*

Number of identical loops of this type. Any value > 1 is equivalent to repeated entry of the same DHWLOOP (and all child objects).

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wlFlow=*float*

Loop flow rate (when operating).

Units	Legal Range	Default	Required	Variability
gpm	≥ 0	6	No	hourly

wlTIn1=*float*

Inlet temperature of first DHWLOOPSEG

Units	Legal Range	Default	Required	Variability
°F	> 0	130	No	hourly

wlRunF=*float*

Fraction of hour that loop circulation operates.

Units	Legal Range	Default	Required	Variability
–	≥ 0	1	No	hourly

wlFUA=*float*

DHWLOOPSEG pipe heat loss adjustment factor.

Units	Legal Range	Default	Required	Variability
–	> 0	1	No	constant

endDHWLoop

Optionally indicates the end of the DHWLOOP definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.16 DHWLOOPPUMP

DHWLOOPPUMP constructs an object representing a pump serving part a DHWLOOP The model is identical to DHWPUMP *except* that that the electricity use calculation reflects wlRunF of the parent DHWLOOP

wlpName

Optional name of pump; give after the word DHWLOOPPUMP if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wlpMult=*integer*

Number of identical pumps of this type. Any value > 1 is equivalent to repeated entry of the same DHWPUMP

Units	Legal Range	Default	Required	Variability
	> 0	1	No	constant

wlpPwr=*float*

Pump power.

Units	Legal Range	Default	Required	Variability
W	> 0	0	No	hourly

wlpElecMtr=*mtrName*

Name of METER object, if any, to which DHWLOOPPUMP electrical energy use is recorded (under end use DHW).

Units	Legal Range	Default	Required	Variability
	name of a METER	*Parent DHWSYS wsElecMtr*	No	constant

endDHWPump

Optionally indicates the end of the DHWPUMP definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.17 DHWLOOPSEG

DHWLOOPSEG constructs one or more objects representing a segment of the preceeding DHWLOOP A DHWLOOP can have any number of DHWLOOPSEG to represent the segments of the loop with possibly differing sizes, insulation, or surrounding conditions.

wgName

Optional name of segment; give after the word DHWLOOPSEG if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wgTy=*choice*

Specifies the type of segment

SUPPLY

Indicates a supply segment (flow is sum of circulation and draw flow, child DHWLOOPBRANCHs permitted).

RETURN

Indicates a return segment (flow is only due to circulation, child DHWLOOPBRANCHs not allowed)

Units	Legal Range	Default	Required	Variability
–		–	Yes	constant

wgLength=*float*

Length of segment.

Units	Legal Range	Default	Required	Variability
ft	≥ 0	0	No	constant

wgSize=*float*

Nominal size of pipe. CSE assumes the pipe outside diameter = size + 0.125 in.

Units	Legal Range	Default	Required	Variability
in	> 0	1	Yes	constant

wgInsulK=*float*

Pipe insulation conductivity

Units	Legal Range	Default	Required	Variability
Btuh-ft/ft^2-°F	> 0	0.02167	No	constant

wgInsulThk=*float*

Pipe insulation thickness

Units	Legal Range	Default	Required	Variability
in	≥ 0	1	No	constant

wgExH=*float*

Combined radiant/convective exterior surface conductance between insulation (or pipe if no insulation) and surround.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	> 0	1.5	No	hourly

wgExT=*float*

Surrounding equivalent temperature.

Units	Legal Range	Default	Required	Variability
°F	> 0	70	No	hourly

wgFNoDraw=*float*

Fraction of hour when no draw occurs.

Units	Legal Range	Default	Required	Variability
°F	> 0	70	No	hourly

endDHWLoopSeg

Optionally indicates the end of the DHWLOOPSEG definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.18 DHWLOOPBRANCH

DHWLOOPBRANCH constructs one or more objects representing a branch pipe from the preceeding DHWLOOPSEG A DHWLOOPSEG can have any number of DHWLOOPBRANCH to represent pipe runs with differing sizes, insulation, or surrounding conditions.

wbNameOptional name of segment; give after the word DHWLOOPBRANCH if desired.

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

wbMult=*float*

Specifies the number of identical DHWLOOPBRANCH Note may be non-integer.

Units	Legal Range	Default	Required	Variability
–	> 0	1	No	constant

wbLength=*float*

Length of branch.

Units	Legal Range	Default	Required	Variability
ft	≥ 0	0	No	constant

wbSize=*float*

Nominal size of pipe. CSE assumes the pipe outside diameter = size + 0.125 in.

Units	Legal Range	Default	Required	Variability
in	> 0	–	Yes	constant

wbInsulK=*float*

Pipe insulation conductivity

Units	Legal Range	Default	Required	Variability
Btuh-ft/ft^2-°F	> 0	0.02167	No	constant

wbInsulThk=*float*

Pipe insulation thickness

Units	Legal Range	Default	Required	Variability
in	≥ 0	1	No	constant

wbExH=*float*

Combined radiant/convective exterior surface conductance between insulation (or pipe if no insulation) and surround.

Units	Legal Range	Default	Required	Variability
Btuh/ft^2-°F	> 0	1.5	No	hourly

wbExT=*float*

Surrounding equivalent temperature.

Units	Legal Range	Default	Required	Variability
°F	> 0	70	No	hourly

wbFlow=*float*

Branch flow rate during draw.

Units	Legal Range	Default	Required	Variability
gpm	≥ 0	2	No	hourly

wgFWaste=*float*

Number of times during the hour when the branch volume is discarded.

Units	Legal Range	Default	Required	Variability
	≥ 0	0	No	hourly

endDHWLoopBranch

Optionally indicates the end of the DHWLOOPBRANCH definition.

Units	Legal Range	Default	Required	Variability
		N/A	No	

5.19 AIRHANDLER

AIRHANDLER defines a central air handling system, containing a fan or fans, optional heating and cooling coils, and optional outside air intake and exhaust. AIRHANDLER are subobjects of TOP and deliver air to one or more ZONE through TERMINAL AIRHANDLER objects can be used to model fan ventilation and forced air heating and cooling. Dual duct systems are modeled with two AIRHANDLER (one for hot air and one for cool air) and two TERMINAL in each zone. Figure 2 shows…. [need a sentence that explains the figure.]

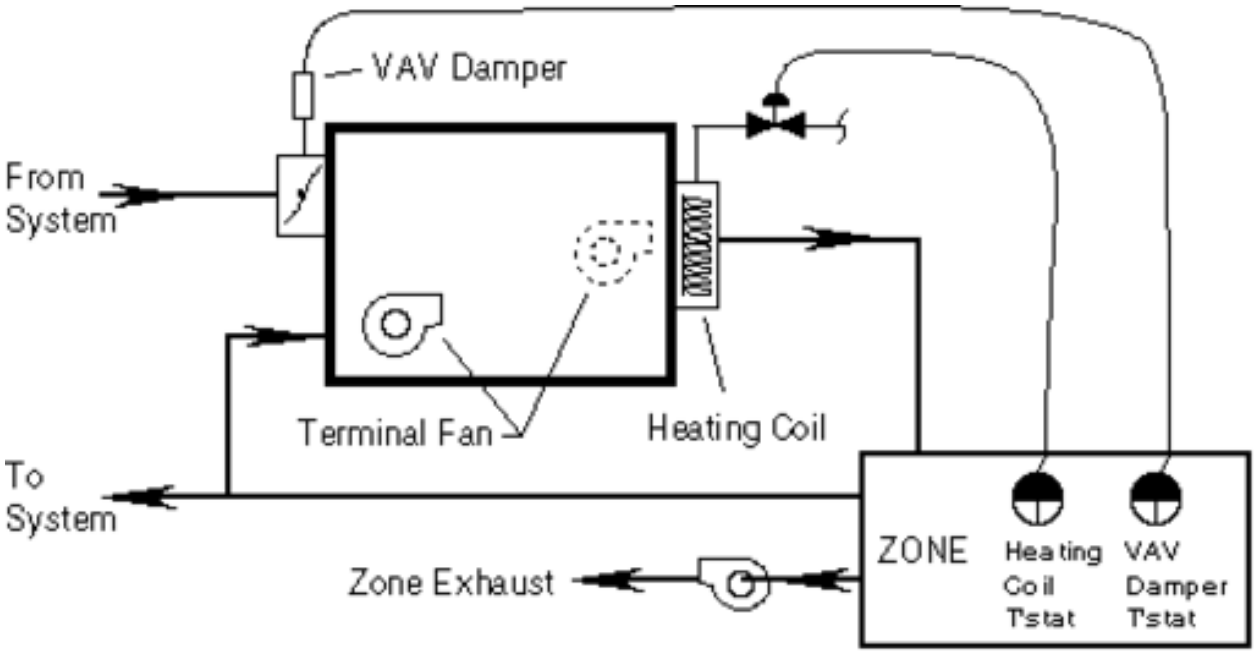

Figure 2: Insert Figure Title

AIRHANDLER is designed primarily to model a central system that supplies hot or cold air at a centrally determined temperature (the "Supply Temperature Setpoint") to Variable Air Volume (VAV) terminals in the zones. Some additional variations are also supported:

1. The AIRHANDLER can model a constant volume, fan-always-on system, where the supply temperature varies to meet the load of a single zone (that is, the thermostat controls the heating and/or cooling coil, but not the fan). This is done by setting the terminal minimum flow, *tuVfMn,* equal to the maximum flow, *tuVfMxH* for heating and/or *tuVfMxC* for cooling, and using a supply temperature control method that adjusts the temperarute to the load (*ahTsSp* = WZ, CZ, or ZN2, described below).
2. The AIRHANDLER can model constant volume, fan cycling systems where the fan cycles with a single zone thermostat, running at full flow enough of the time to meet the load and shutting completely off the rest of the time, rather than running at variable flow to adjust to the demand from the zones.

   This variation is invoked by specifying *ahFanCycles*= YES (usually with *ahTsSp*=ZN, described below). The user should be aware that this is done by treating fractional flow as equivalent to fractional on-time in most of the program, adjusting for the higher flow and less than 100% duty cycle only in a few parts of the model known to be non-linear, such as computation of cooling coil performance, fan heat, and duct leakage. For example, the outside air inputs, designed for VAV modeling, won't work in the expected manner unless you keep this modeling method in mind.
3. The AIRHANDLER can supply hot air, cold air, or shut off according to the requirements of a single zone. This variation is invoked by giving *ahTsSp* = ZN or ZN2, both described further below.

ahName

Name of air handler: give after the word AIRHANDLER Required for reference in TERMINAL

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	

ahSched=*choice*

Air handler schedule; OFF or ON, hourly schedulable by using CSE expression.

OFF

supply fan off; air handler not operating. Old date? Note: (future) Taylor setback/setup control in effect, when implemented.

ON

supply fan runs, at varying volume according to TERMINAL demand (except if ahFanCycles = YES, fan cycles on and off at full volume).

The following might be used to run an air handler between 8 AM and 5 PM:

```
ahSched = select(  (\$hour > 8 && \$hour <= 5),  ON,
                                                 default, OFF );
```

Units	Legal Range	Default	Required	Variability
	ON/OFF	ON	No	hourly

5.19.0.1 AIRHANDLER Supply Air Temperature Controller

ahTsSp=*float or choice*

Supply temperature setpoint numeric value OR* choice of control method (WZ, CZ, RA, ZN, or ZN2):

float

A numeric value specifies the supply temperature setpoint. An expression can be used to make dependent on time, weather, etc.

WZ

Warmest Zone: for cooling, sets the supply temperature setpoint each sub??hour so that the control zone (see ahWzCzns) requiring the coolest supply temperature can meet its load with its VAV damper 90% of the way from its minimum opening to its maximum, that is, at a flow of: tuVfMn + .9(tuVfMxC - * tuVfMn*).

CZ

Coolest Zone: analogous to WZ, but for heating

RA

Supply temperature setpoint value is controlled by return air temperature (this cannot be done with a CSE expression without lagging a subhour). See ahTsRaMn and ahTsRaMx.

ZN

Causes air handler to switch between heating, OFF, and cooling as required by the load of a single zone. When the zone thermostat (modeled through the tuTC and tuTH inputs) calls for neither heating nor cooling, the air handler shuts down, including stopping its fan(s). Changes ahFanCycles default to YES, to simulate a constant volume, fan cycling system.

Supply temperature setpoint value when ahFanCycles = YES is taken from ahTsMn for cooling, from ahTsMx for heating (actual temperatures expected to be limited by coil capacity since fan is running at full flow). When ahFanCycles = NO, the setpoint is determined to allow meeting the load, as for WZ and CZ.

When the zone is calling for neither heat nor cold, the air handler shuts down, including stopping its fan(s), regardless of the ahFanCycles value.

ZN2

Causes air handler to switch between heating, cooling, and FAN ONLY operation as required by the load of a single zone. To model a constant volume system where the fan runs continuously, use ZN2 and set the terminal minimum flow (tuVfMn) equal to the maximum (tuVfMxC and/or tuVfMxH).

When ahTsSp is ZN2, the supply temperature setpoint is determined to allow meeting the load, as for WZ and CZ, described above.

Only when *ahTsSp* is ZN or ZN2 does AIRHANDLER switches between heating and cooling supply temperatures according to demand. In other cases, there is but a single setpoint value or control method (RA, CZ, or WZ); if you want the control method or numeric value to change according to time of day or year, outside temperature, etc., your CSE input must contain an appropriate conditional expression for *ahTsSp*.

Unless *ahTsSp* is ZN or ZN2, the AIRHANDLER does not know whether it is heating or cooling, and will use either the heating coil or cooling coil, if available, as necessary, to keep the supply air at the single setpoint temperature. The coil schedule members, described below, allow you to disable present coils when you don't want them to operate, as to prevent cooling supply air that is already warm enough when heating the zones. For example, in an AIRHANDLER with both heating and cooling coils, if you are using a conditional expression based on outdoor temperature to change *ahTsSp* between heating and cooling values, you may use expressions with similar conditions for *ahhcSched* and *ahccSched* to disable the cooling coil when heating and vice versa. (Expressions would also be used in the TERMINALS to activate their heating or cooling setpoints according to the same conditions.)

Giving *ahTsSp* is disallowed for an air handler with no economizer, no heat coil and no cooling coil. Such an AIRHANDLER object is valid as a ventilator; its supply temperature is not controlled. but rather determined by the outside temperature and/or the return air temperature.

Unit s

Legal Range

Defaul t

Required

Variabil ity

oF

number, RA*, WZ, CZ, ZN**, ZN2**

0

YES, if coil(s) or economizer present

hourly

* ahTsRaMn, ahTsRaMx, ahTsMn, and ahTsMx are *required* input for this choice.

** only a single ZONE may be used with these choices.

Using AIRHANDLER to Model Various Systems

To Model

Use

Comments

VAV heating OR cooling system

ahTsSp = numeric expression, WZ, CZ, or RA

CSE models this most directly

VAV system that both heats and cools (single duct)

Use a conditional expression to change ahTsSp between heating and cooling values on the basis of outdoor temperature, date, or some other condition.

Also use expressions to disable the unwanted coil and change each zone's setpoints according to same condition as ahTsSp. For example, when heating, use ahccSched = OFF and tuTC = 999; and when cooling, use ahhcSched = OFF and tuTH = -99.

Dual duct heating / cooling system

Use two AIRHANDLERs

Single zone VAV system that heats or cools per zone thermostat

ahTsSp = ZN2

Supply fan runs, at flow tuVfMn, even when neither heating nor cooling. Supply temp setpoint determined as for CZ or WZ.

Single zone constant volume system that heats or cools per zone thermostat, e.g. PSZ.

ahTsSp = ZN2; tuVfMn = tuVfMxH = tuVfMxC

Supply fan circulates air even if neither heating nor cooling. Supply temp setpoint determined as for CZ or WZ. All tuVf's same forces constant volume.

Single zone constant volume, fan cycling system that heats or cools per zone thermostat,e.g. PTAC, RESYS, or furnace.

ahTsSp= ZN; ahTsMx = heat supply temp setpoint; ahTsMn = cool supply temp setpoint; tuVfMn= 0; tuVfMxH = tuVfMxC normally; sfanVfDs >= max(tuVfMxH, tuVfMxC) to minimize confusion about flow modeled.

AhFanCycles defaults to YES. Supply fan off when not heating or cooling. Flow when fan on is tuVfMxH or tuVfMxC as applicable (or sfanVfDs * sfanVfMxF if smaller).

: Using AIRHANDLER to Model Various Systems

ahFanCycles=*choice*

Determines whether the fan cycles with the zone thermostat.

YES

Supply fan runs only for fraction of the subhour that the zone requests heating or cooling. When running, supply fan runs at full flow (i.e. constant volume), as determined by the more limiting of the air handler and terminal specifications. Use with a single zone only. Not allowed with ahTsSp = ZN2.

NO

Normal CSE behavior for simulating VAV systems with continuously running (or scheduled), variable flow supply fans. (For constant volume, fan always on modeling, use NO, and make tuVfMn equal to tuVfMxH/C.)

Units	Legal Range	Default	Required	Variability
	YES,NO	YES when *ahTsSp*=ZN, NO otherwise	No	hourly

ahTsMn=*float*

Minimum supply temperature. Also used as cooling supply temperature setpoint value under *ahTsSp* = ZN.

Units	Legal Range	Default	Required	Variability
°F	*no limit*; typically: $40 \leq x \leq 140^{o}$	0°F	Only for *ahTsSp*=RA	hourly

Uni ts

Legal Range

Defa ult

Required

Variab ility

oF

no limit; typically: 40 x 140o

999o F

Only for ahTsSp=RA; recommend giving for ahTsSp=ZN

hourly

ahTsMx=*float*

Maximum supply temperature. Also used as heating supply temperature setpoint value under *ahTsSp* = ZN.

ahWzCzns=*zone names* or *ALL* or *ALL_BUT zone names*

ahCzCzns=*zone names* or *ALL* or *ALL_BUT zone names*

Specify zones monitored to determine supply temperature setpoint value (control zones), under *ahTsSp*=WZ and CZ respectively.

zone names

A list of zone names, with commas between them. Up to 15 names may be given.

ALL_BUT

May be followed by a a comma and list of up to 14 zone names; all zones on air handler other than these are the control zones.

ALL

Indicates that all zones with terminals connected to the air handler are control zones.

A comma must be entered between zone names and after the word ALL_BUT.

Units	Legal Range	Default	Required	Variability
	name(s) of ZONEs ALL ALL_BUT *zone Name(s)*	ALL	NO	hourly

ahCtu=*terminal name*

Terminal monitored to determine whether to heat or cool under ZN and ZN2 supply temperature setpoint control. Development aid feature; believe there is no need to give this since ahTsSp = ZN or ZN2 should only be used with one zone.

Unit s

Legal Range

Default

Required

Variabi lity

name of a TERMINAL

AIRHANDLER's TERMINAL, if only one

If ahTsSp = ZN with more than 1 TERMINAL

hourly

AhTsRaMn and *ahTsRaMx* are used when *ahTsSp* is RA.

ahTsRaMx=*float*

Return air temperature at which the supply temperature setpoint is at the *maximum* supply temperature, *ahTsMx*.

ahTsRaMx=*float*

Return air temperature at which the supply temperature setpoint is at the *minimum* supply temperature, *ahTsMn*.

When the return air temperature is between *ahTsRaMn*and *ahTsRaMx*, the supply temperature setpoint has a proportional value between *ahTsMx* and *ahTsMn*.

If return air moves outside the range *ahTsRaMn* to *ahTsRaMx*, the supply temperature setpoint does not change further.

Units	Legal Range	Default	Required	Variability
°F	*no limit*; typically: $40 \leq x \leq 140$°	*none*	Only for *ahTsSp*=RA	hourly

5.19.0.2 AIRHANDLER Supply fan

All AIRHANDLER have supply fans.

sfanType=*choice*

Supply fan type/position. A BLOWTHRU fan is located in the air path before the coils; a DRAWTHRU fan is after the coils.

Units	Legal Range	Default	Required	Variability
	DRAWTHRU, BLOWTHRU	DRAWTHRU	No	constant

sfanVfDs=*float*

Design or rated (volumetric) air flow at rated pressure. Many fans will actually blow a larger volume of air at reduced pressure: see sfanVfMxF (next).

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	*none*	Yes	constant

sfanVfMxF=*float*

Overrun factor: maximum factor by which fan will exceed rated flow (at reduced pressure, not explicitly modeled). CSE delivers flows demanded by terminals until total flow at supply fan reaches sfanVfDs * sfanVsMxF, then reduces maximum flows to terminals, keeping them in proportion to terminal design flows, to keep total flow at that value.

We recommend giving 1.0 to eliminate overrun in constant volume modeling.

Units	Legal Range	Default	Required	Variability
	$x \geq 1.0$	1.3	No	constant

sfanPress=*float*

Design or rated pressure. The work done by the fan is computed as the product of this pressure and the current flow, except that the flow is limited to sfanVfDs. That is, in overrun (see *sfanVfMxF*) it is assumed that large VAV terminal damper openings allow the pressure to drop in proportion to the flow over rated. This work is added to the air as heat at the fan, and is termed "fan heat". Setting sfanPress to zero will eliminate simulated fan heat for theoretical simulation of a coil only.

Units	Legal Range	Default	Required	Variability
inches H_2O	$x > 0$	3	No	constant

Prior text: At most, one of the next two items may be given: in combination with sfanVfDs and sfanPress, either is sufficient to compute the other. SfanCurvePy is then used to compute the mechanical power at the fan shaft at partial loads; sfanMotEff allows determining the electrical input from the shaft power.

New possible text (after addition of sfanElecPwr): Only one of sfanElecPwr, sfanEff, and sfanShaftBhp may be given: together with sfanVfDs and xfanPress, any one is sufficient for CSE to detemine the others and to compute the fan heat contribution to the air stream.

sfanElecPwr=*float*

Fan input power per unit air flow (at design flow and pressure).

Unit s

Legal Range

Default

Required

Variabi lity

W/cfm

x > 0

derived from sfanEff and sfanShaftBhp

If sfanEff and sfanShaftBhp not present

constant

sfanEff=*float*

Fan efficiency at design flow and pressure, as a fraction.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	derived from *sfanShaftBhp* if given, else 0.65	No	constant

sfanShaftBhp=*float*

Fan shaft brake horsepower at design flow and pressure.

Units	Legal Range	Default	Required	Variability
bhp	$x > 0$	derived from *sfanEff*.	No	constant

sfanCurvePy=k_0, k_1, k_2, k_3, x_0

k_0 through k_3 are the coefficients of a cubic polynomial for the curve relating fan relative energy consumption to relative air flow above the minimum flow x_0. Up to five *floats* may be given, separated by commas. 0 is used for any omitted trailing values. The values are used as follows:

$$z = k_0 + k_1 \cdot (x - x_0)| + k_2 \cdot (x - x_0)|^2 + k_3 \cdot (x - x_0)|^3$$

where:

- x is the relative fan air flow (as fraction of *sfanVfDs*; $0 \leq$ x ≤ 1);
- x_0 is the minimum relative air flow (default 0);
- $(x - x_0)|$ is the "positive difference", i.e. $(x - x_0)$ if $x > x_0$; else 0;
- z is the relative energy consumption.

If z is not 1.0 for $x = 1.0$, a warning message is displayed and the coefficients are normalized by dividing by the polynomial's value for $x = 1.0$.

Units	Legal Range	Default	Required	Variability
		0, 1, 0, 0, 0 (linear)	No	constant

sfanMotEff=*float*

Motor/drive efficiency.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.9	No	constant

sfanMotPos=*float*

Motor/drive position: determines disposition of fan motor heat (input energy in excess of work done by fan; the work done by the fan is the "fan heat", always added to air flow).

IN_FLOW

add fan motor heat to supply air at the fan position.

IN_RETURN

add fan motor heat to the return air flow.

EXTERNAL

discard fan motor heat

sfanMtr=*mtrName*

Name of meter, if any, to record energy used by supply fan. End use category used is "Fan".

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

5.19.0.3 AIRHANDLER Return/Relief fan

A return/relief fan is optional. Its presence is established by setting *rfanType* to a value other than NONE. For additional information on the return/relief fan members, refer to the description of the corresponding supply fan member above.

rfanType=*choice*

relief fan type/position.

RETURN

fan is at air handler; all return air passes through it.

RELIEF

fan is in exhaust path. Air being exhausted to the outdoors passes through fan; return air being recirculated does not pass through it.

NONE

no return/relief fan in this AIRHANDLER.

Units	Legal Range	Default	Required	Variability
	NONE, RETURN, RELIEF	NONE	Yes, if fan present	constant

rfanVfDs=*float*

design or rated (volumetric) air flow.

Units	Legal Range	Default	Required	Variability
cfm	$x > 0$	*sfanVfDs - oaVfDsMn*	No	constant

rfanVfMxF=*float*

factor by which fan will exceed design flow (at reduced pressure).

Units	Legal Range	Default	Required	Variability
	$x \geq 1.0$	1.3	No	constant

rfanPress=*float*

design or rated pressure.

Units	Legal Range	Default	Required	Variability
inches H_2O	$x > 0$	0.75	No	constant

At most, one of the next three?? items may be defined: ?? rework re rfanElecPwr

rfanElecPwr=*float*

Fan input power per unit air flow (at design flow and pressure).

Unit s

Legal Range

Default

Required

Variabi lity

W/cfm

x > 0

derived from rfanEff and rfanShaftBhp

If rfanEff and rfanShaftBhp not present

constant

rfanEff=*float*

Fan efficiency at design flow and pressure.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	derived from *rfanShaftBhp* if given, else 0.65	No	constant

rfanShaftBhp=*float*

Fan shaft brake horsepower at design flow and pressure.

Units	Legal Range	Default	Required	Variability
bhp	$x > 0$	derived from *rfanEff*.	No	constant

rfanCurvePy=k_0, k_1, k_2, k_3, x_0

k_0 through k_3 are the coefficients of a cubic polynomial for the curve relating fan relative energy consumption to relative air flow above the minimum flow x_0. Up to five *floats* may be given, separated by commas. 0 is used for any omitted trailing values. The values are used as follows:

$$z = k_0 + k_1 \cdot (x - x_0)| + k_2 \cdot (x - x_0)|^2 + k_3 \cdot (x - x_0)|^3$$

where:

- x is the relative fan air flow (as fraction of *rfanVfDs*; $0 \leq x \leq 1$);
- x_0 is the minimum relative air flow (default 0);
- $(x - x_0)|$ is the "positive difference", i.e. $(x - x_0)$ if $x > x_0$; else 0;
- z is the relative energy consumption.

If z is not 1.0 for $x = 1.0$, a warning message is displayed and the coefficients are normalized by dividing by the polynomial's value for $x = 1.0$.

Units	Legal Range	Default	Required	Variability
		0, 1, 0, 0, 0 (linear)	No	constant

rfanMotEff=*float*

Motor/drive efficiency.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.9	No	constant

rfanMotPos=*choice*

Motor/drive position.

Units	Legal Range	Default	Required	Variability
	IN_FLOW, EXTERNAL	IN_FLOW	No	constant

rfanMtr=*mtrName*

Name of meter, if any, to record power consumption of this return fan. May be same or different from meter used for other fans and coils in this and other air handlers. "Fan" end use category is used.

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

5.19.0.4 AIRHANDLER Heating coil/Modeling Furnaces

Heating coils are optional devices that warm the air flowing through the AIRHANDLER including electric resistance heaters, hot water coils supplied by a HEATPLANT the heating function of an air source heat pump, and furnaces.

Furnaces are modeled as AIRHANDLER with heat "coils" that model the heating portion of a gas or oil forced hot air furnace. Notes on modeling a furnace with a CSE AIRHANDLER:

- Give *ahhcType* = GAS or OIL.
- Give *ahhcAux's* to model the power consumption of pilot, draft fan, etc.
- Use *ahTsSp* = ZN, which implies *ahFanCyles* = YES, to model constant volume, fan cycling (as opposed to VAV) operation.
- Use *ahTsMx* = an appropriate value around 140 or 180 to limit the supply temperature, simulating the furnace's high temperature cutout (the default *ahTsMx*of 999 is too high!).
- Use a single TERMINAL on the AIRHANDLER.
- To eliminate confusion about the fan cfm (which, precisely, under *ahFanCyles* = YES, is the smaller of the terminal maximum or the supply fan maximum including overrun), give the same value for TERMINAL *tuVfMxH* and AIRHANDLER *sfanVfDs*, and give *sfanVfMxF* = 1.0 to eliminate overrun.
- You will usually want to use *oaVfDsMn* = 0 (no outside air), and no economizer.

The heating function of an air source heat pump is modeled with an AIRHANDLER with heat coil type AHP. There are several additional heat coil input variables (names beginning with *ahp-*) described later in the heat coil section. Also, a heat pump generally has a crankcase heater, which is specified with the crankcase heater inputs (*cch-*), described later in the AIRHANDLER Section 0. If the heat pump also performs cooling, its cooling function is modeled by specifying a suitable cooling coil in the same AIRHANDLER Use *ahccType* = DX until a special cooling coil type for heat pumps is implemented. It is the user's responsibility to specify consistent heating and cooling coil inputs when the intent is to model a heat pump that both heats and cools, as CSE treats the heat coil and the cool coil as separate devices.

The next four members apply to all heat coil types, except as noted.

To specify that an AIRHANDLER has a heating coil and thus heating capability, give an *ahhcType* other than NONE.

ahhcType=*choice*

Coil type choice:

ELECTRIC

electric resistance heat: 100% efficient, can deliver its full rated capacity at any temperature and flow.

HW

hot water coil, supplied by a HEATPLANT object.

GAS or OIL

"coil" type that models heating portion of a forced hot air furnace. Furnace "coil" model uses inputs for full-load efficiency and part-load power input; model must be completed with appropriate auxiliaries, ahTsSp, etc. See notes above.

GAS and OIL are the same here – the differences between gas- and oil-fired furnaces is in the auxiliaries (pilot vs. draft fan, etc.), which you specify separately.

AHP

heating function of an air source heat pump.

NONE

AIRHANDLER has no heat coil, thus no heating capability.

Units	Legal Range	Default	Required	Variability
	ELECTRIC, HW, GAS OIL, AHP, NONE	NONE	Yes, if coil is present	constant

ahhcSched=*choice*

Heat coil schedule; choice of AVAIL or OFF, hourly variable. Use an appropriate ahhcSched expression if heat coil is to operate only at certain times of the day or year or only under certain weather conditions, etc.

AVAIL

heat coil available: will operate as necessary to heat supply air to supply temperature setpoint, up to the coil's capacity.

OFF

coil will not operate, no matter how cold supply air is. A HW coil should be scheduled off whenever its HEATPLANT is scheduled off (hpSched) to insure against error messages.

Units	Legal Range	Default	Required	Variability
	AVAIL, OFF	AVAIL	No	hourly

ahhcCapTRat=*float*

Total heating (output) capacity. For an ELECTRIC, GAS, or OIL coil, this capacity is always available. For an HW heating coil, when the total heat being requested from the coil's HEATPLANT would overload the HEATPLANT the capacity of all HW coils connected to the plant (in TERMINAL as well as AIRHANDLER is reduced proportionately until the requested total heat is within the HEATPLANT capacity. Not used if *ahhcType* = AHP (see *ahpCap17* and *ahpCap47*).

Units	Legal Range	Default	Required	Variability
Btuh	$x \geq 0$	*none*	Yes, if coil present, except coil type AHP	hourly

ahhcMtr=*mtrName*

Name of meter to accumulate energy use by this heat coil. The input energy used by the coil is accumulated in the end use category "Htg"; for a heat pump, the en*ergy used by the supplemental resistance heaters (regular and defrost) is accumulated under the category "hp". Not allowed when*ahhcType* is HW, as an HW coil's energy comes from its HEATPLANT and the HEATPLANT BOILER accumulate input energy to meters.

Units	Legal Range	Default	Required	Variability

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

The following input is used only when *ahhcType* is HW:

ahhcHeatplant=*Heatplant name*

Name of HEATPLANT supporting hot water coil.

Units	Legal Range	Default	Required	Variability
	name of a HEATPLANT	*none*	if *ahhcType* is HW	constant

The following inputs are used only for furnaces (*ahhcType* = GAS or OIL).

One of the next two items, but not both, **must** be given for furnaces:

ahhcEirR=*float*

Rated energy input ratio (input energy/output energy) at full power.

Units	Legal Range	Default	Required	Variability
	$x \geq 1$	*none*	if *ahhcEirR* not given and *ahhcType* is GAS or OIL	hourly

ahhcEffR=*float*

Rated efficiency (output energy/input energy; 1/ahhcEirR) at full power

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	*none*	if *ahhcEirR* not given and *ahhcType* is GAS or OIL	hourly

ahhcPyEi=k_0**,** k_1**,** k_2**,** k_3

Coefficients of cubic polynomial function of (subhour average) part-load-ratio (plrAv) to adjust the full-load furnace energy input for part load operation. Enter, separated by commas, in order, the constant part, the coefficient of plrAv, the coefficient of plrAv squared, and the coefficient of plrAv cubed. CSE will normalize the coefficients if necessary to make the polynomial value be 1.0 when the part load ratio is 1.0.

The default, from DOE2, is equivalent to:

```
ahhcPyEi = .01861, 1.094209, -.112819, 0.;
```

which corresponds to the quadratic polynomial:

$$\text{pyEi}(\text{plrAv}) = 0.01861 + 1.094209 \cdot \mathbf{plrAv} - 0.112819 \cdot \mathbf{plrAv}^2$$

Note that the value of this polynomial adjusts the energy input, not the energy input ratio, for part load operation.

Units	Legal Range	Default	Required	Variability
		0.01861, 1.094209, -0.112819, 0.0.	No	constant

ahhcStackEffect=*float*

Fraction of unused furnace capacity that must be used to make up for additional infiltration caused by stack effect of a hot flue when the (indoor) furnace is NOT running, only in subhours when furnace runs PART of the subhour, per DOE2 model.

This is an obscure feature that will probably never be used, included only due to indecisiveness on the part of most members of the committee designing this program. The first time reader should skip this section, or read it only as an example of deriving an expression to implement a desired relationship.

The stack effect is typically a function of the square root of the difference between the outdoor temperature and the assumed stack temperature.

For example, the following is a typical example for furnace stack effect:

```
ahhcStackEffect = @Top.tDbO >= 68. ? 0.
                : (68. - @Top.tDbO)
                  * sqrt(200.-@Top.tDbO)
                  / (10*68*sqrt(200));
```

The code "@Top.tDbO >= 68 ? 0. : ..." insures that the value will be 0, not negative, when it is warmer than 68 out (if the furnace were to run when the value was negative, a run-time error would terminate the run).

The factor "(68. − @Top.tDbO)" reflects the fact that the energy requirement to heat the infiltrating air is proportional to how much colder it is than the indoor temperature. Note that its permitted to use a constant for the indoor temperature because if it is below setpoint, the furnace will be running all the time, and there will be no unused capacity and the value of ahhcStackEffect will be moot.

The factor "sqrt(200.−@Top.tDbO)" represents the volume of infiltrated air that is typically proportional to the square root of the driving temperature difference, where 200 is used for the estimated effective flue temperature.

The divisor "/ (10*68*sqrt(200))" is to make the value 0.1 when tDbO is 0, that is, to make the stack effect loss use 10% of unused load when it is 0 degrees out. The actual modeling engineer must know enough about his building to be able to estimate the additional infiltration load at some temperature.

Units	Legal Range	Default	Required	Variability
	$0 \le x \le 1$	0	No	hourly

The following heat coil input members, beginning with *ahp-*, are used when modeling the heating function of an air source heat pump with the air handler heat coil, that is, when *ahhcType*= AHP is given. Also, see the AIRHANDLER Crankcase Heater" section with regard to specifying the heat pump's crankcase heater.

The next six inputs give the heat pump's steady state heating output capacity.

ahpCap17=*float*

ahpCap47=*float*

ARI steady state (continuous operation) rated capacities at 70 degrees F indoor (return) air temp, and 17 and 47 degrees F outdoor temp, respectively. These values reflect no cycling, frost, or defrost degradation. To help you find input errors, the program issues an error message if ahpCap17 >= ahpCap47.

Units	Legal Range	Default	Required	Variability
Btuh	$x > 0$		Yes, for AHP coil	constant

ahpCap35=*float*ARI steady state (continuous operation) rated capacity at 35 F outdoor temp, reflecting

frost buildup and defrost degradation but no cycling. Unlikely to be available for input; if not given, will be defaulted to *ahpFd35Df* (next description) times a value determined by linear interpolation between the given *ahpCap17* and *ahpCap47* values. If *ahpCap35* is given, CSE will issue an error message if it is greater than value determined by linear interpolation between *ahpCap17* and *ahpCap47*.

Units	Legal Range	Default	Required	Variability
Btuh	$x > 0$	from ahpFd35Df	No	constant

ahpFd35Df=*float*

Default frost/defrost degradation factor at 35 F: reduction of output at unchanged input, due to defrosting and due to frost on outdoor coil. Used in determining default value for *ahpCap35* (preceding description); not used if *ahpCap35* is given.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.85	No	constant

ahpCapIa=*float*Capacity correction factor for indoor (return) air temperature, expressed as a fraction reduction in capacity per degree above 70F.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.004	No	constant

ahpSupRh=*float*

Input (and output) power of supplemental resistance reheat coil used when heat pump alone cannot meet the load. This power input is in kW, not Btuh as for most CSE power inputs. Energy consumed by this heater, as well as the defrost supplemental resistance heater, is accumulated in category "hp" of ahhcMeter (whereas energy consumption of the heat pump compressor is accumulated under category "Htg").

Units	Legal Range	Default	Required	Variability
kW	$x > 0$	10 kW	No	constant

The next seven inputs specify frost buildup and defrosting and their effect on capacity.

ahpTFrMn=*float*

ahpTFrMx=*float*

ahpTFrPk=*float*

Lowest, highest, and peak temperatures for frost buildup and defrost effects. Capacity reduction due to frost and defrosting consists of a component due to frost buildup on the outdoor coil, plus a component due to lost heating during the time the heat pump is doing reverse cycle defrosting (heating the outdoor coil to melt off the frost, which cools the indoor coil). The effects of Frost Buildup and of time spent defrosting are computed for different temperature ranges as follows:

- Above *ahpTFrMx*: no frost buildup, no defrosting.
- At *ahpTFrMx* OR *ahpTFrMn*: defrosting time per *ahpDfrFMn* (next description); no frost buildup effect.

- At *ahpTFrPk*: defrosting time per *ahpDfrFMx*, plus additional output reduction due to effect of frost buildup, computed by linear extrapolation from *ahpCap35* or its default.
- Between *ahpTFrPk* and *ahpTFrMn* or *ahpTFrMx*: defrost time and defrost buildup degradation linearly interpolated between values at *ahpTFrPk* and values at *ahpTFrMn* or *ahpTFrMx*.
- Below *ahpTFrMn*: no frost buildup effect; time defrosting remains at *ahpDfrFMn*.

In other words, the curve of capacity loss due to frost buildup follows straight lines from its high point at *ahpTFrPk* to zero at *ahpTFrMn* and *ahpTFrMx*, and remains zero outside the range *ahpTFrMn* to *ahpTFrMx*. The height of the high point is determined to match the *ahpCap35* input value or its default. The curve of time spent defrosting is described in other words in the description of *ahpDfrFMn* and *ahpDfrFMx*, next.

An error will occur unless $ahpTFrMn < ahpTFrPk < ahpTFrMx$ and $ahpTFrMn < 35 < ahpTFrMx$.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	*ahpTFrMn*: 17, *ahpTFrMx*: 47, *ahpTFrPk*: 42	No	constant

ahpDfrFMn=*float*

ahpDfrFMx=*float*

Minimum and maximum fraction of time spent in reverse cycle defrost cooling.

The fraction of the time spent defrosting depends on the outdoor temperature, as follows: at or below *ahpTFrMn*, and at (but not above) *ahpTFrMx*, *ahpDfrFMn* is used. *ahpDfrFMx* is used at *ahpTFrMx*. Linear interpolation is used between *ahpTFrMn* or *ahpTFrMx* and *ahpTFrMx*. No time is spent defrosting above *ahpTFrMx*.

In other words, the curve of time spent defrosting versus outdoor temperature has the value *ahpDfrFMn* up to *ahpTFrMn*, then rises in a straight line to *ahpDfrFMx* at *ahpTFrMx*, then falls in a straight line back to *ahpDfrFMn* at *ahpTFrMx*, then drops directly to zero for all higher temperatures.

During the fraction of the time spent defrosting, the heat pump's input remains constant and the output is changed as follows:

- Usual heat output is not delivered to load.
- Cold output due to reverse cycle operation is delivered to load. See *ahpDfrCap*.
- An additional resistance heater is operated; and its heat output is delivered to load. See *ahpDfrRh*.

The program will issue an error message if $ahpDfrFMx \leq ahpDfrFMn$.

Uni ts

Legal Range

Default

Requ ired

Variab ility

0 x 1

ahpDfrFMn: .0222, (2 minutes/90 minutes), ahpDfrFMx: .0889, (8 minutes/90 minutes)

No

constant

ahpDfrCap=*float*

Cooling capacity (to air handler supply air) during defrosting. Program separately computes the lost heating capacity during defrosting, but effects of switchover transient should be included in *ahpDfrCap*.

Units	Legal Range	Default	Required	Variability
Btuh	$x \neq 0$	$2 \cdot$ *ahpCap17*	No	constant

ahpDfrRh=*float*

Input (and output) power of resistance reheat coil activated during defrost. Input is in kW, not Btuh as most CSE power inputs. Energy used by this heater is accumulated in *ahhcMeter* category "hp".

Units	Legal Range	Default	Required	Variability
kW	$x > 0$	5 kW	No	constant

Inputs for air source heat pump low temperature cutout:

ahpTOff=*float*

ahpTOn=*float*

Heat pump low temperature cutout setpoints. Heat pump is disabled (only the supplemental resistance heater operates) when outdoor temperature falls below *ahpTOff*, and is re-enabled when temperature rises above *ahpTOn*. Different values may be given to simulate thermostat differential. *ahpTOff* must be $\leq$ *ahpTOn*; equal values are accepted.

Units	Legal Range	Default	Required	Variability
°F		*ahpTOff*: 5, *ahpTOn*: 12	No	constant

The next four inputs specify the heating power input for an air source heat pump:

ahpIn17=*float*

ahpIn47=*float*

Steady state (full power, no cycling) power input for compressor and crankcase heater at 70 degrees F indoor (return) air temp and 17 and 47 degrees F outdoor temp respectively.

Units	Legal Range	Default	Required	Variability
kW	$x > 0$		Yes, for AHP coil	constant

ahpInIa=*float*

Indoor (return) air temp power input correction factor: fraction increase in steady-state input per degree above 70 F, or decrease below 70F.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.004	No	constant

ahpCd=*float*

ARI cycling degradation coefficient: ratio of fraction drop in system coefficient of performance (COP) to fraction drop in capacity when cycling, from steady-state values, in ARI 47 F cycling performance tests. A value of .25 means that if the heat pump is cycled to drop its output to 20% of full capacity (i.e. by the fraction .8), its COP will drop by .8 * .25 = .2. Here COP includes all energy inputs: compressor, crankcase heater, defrost operation, etc.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.25	No	constant

The following four air handler heat coil members allow specification of auxiliary input power consumption associated with the heat coil (or furnace) under the indicated conditions. The single description box applies to all four.

ahhcAuxOn=*float*

Auxiliary power used when running, in proportion to subhour average part load ratio (plrAv). Example use: oil furnace induced draft fan.

ahhcAuxOff=*float*

Auxiliary power used when coil is not running, in proportion to 1 - plrAv.

ahhcAuxFullOff=*float*

Auxiliary power used only when coil is off for entire subhour; not used if the coil is on at all during a subhour. Example use: Gas furnace pilot under DOE2 model, where pilot is included in main energy input if furnace runs at all in subhour.

ahhcAuxOnAtAll=*float*

Auxiliary power used in full value if coil is on for any fraction of a subhour.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	0	No	hourly

The following four members specify meters for recording auxiliary energy use through ahhcAuxOn, ahhcAuxOff, ahhcAuxFullOff, and ahhcAuxOnAtAll, respectively. End use category "Aux" is used.

ahhcAuxOnMtr=*mtrName*

ahhcAuxOffMtr=*mtrName*

ahhcAuxFullOffMtr=*mtrName*

ahhcAuxOnAtAllMtr=*mtrName*

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

5.19.0.5 AIRHANDLER Cooling coil

A cooling coil is an optional device that remove heat and humidity from the air passing through the AIRHANDLER Available cooling coil types include chilled water (CHW), supported by a COOLPLANT that supplies cold water, and Direct Expansion (DX), supported by a dedicated compressor and condenser that are modeled integrally with the DX coil. No plant is used with DX coils.

The following five members are used for all cool coil types except as noted. Presence of a cool coil in the AIRHANDLER is indicated by giving an *ahccType value* other than NONE.

ahccType=*choice*

Cool coil type choice:

ELECTRIC

Testing artifice: removes heat at 100% efficiency up to rated capacity at any flow and temperature; removes no humidity. Use in research runs to isolate effects of coil models from other parts of the CSE program.

CHW

CHilled Water coil, using a cold water from a COOLPLANT.

DX

Direct Expansion coil, with dedicated compressor and condenser modeled integrally.

NONE

AIRHANDLER has no cooling coil and no cooling capability.

Units	Legal Range	Default	Required	Variability
	ELECTRIC, DX, CHW, NONE	NONE	Yes, if coil present	constant

ahccSched=*choice*

Cooling coil schedule choice, hourly variable. Use a suitable CSE expression for ahccSched if cooling coil is to operate only at certain times, only in hot weather, etc.

AVAIL

Cooling coil will operate as necessary (within its capacity) to cool the supply air to the supply temperature setpoint.

OFF

Cooling coil will not operate no matter how hot the supply air is. To avoid error messages, a CHW coil should be scheduled OFF whenever its COOLPLANT is scheduled OFF.

ahccCapTRat=*float*

Total rated capacity of coil: sum of its "sensible" (heat-removing) and "latent" (moisture removing) capacities. Not used with CHW coils, for which capacity is implicitly specified by water flow (ahccGpmDs*) and transfer unit (*ahccNtuoDs* and *ahccNtuiDs*) inputs, described below.

For coil specification conditions (a.k.a. rating conditions or design conditions), see *ahccDsTDbEn*, *ahccDsTWbEn*, *ahccDsTDbCnd*and *ahccVfR*below (see index).

Units	Legal Range	Default	Required	Variability
Btuh	$x > 0$	*none*	Yes	constant

ahccCapSRat=*float*

Sensible (heat-removing) rated capacity of cooling coil. Not used with CHW coils.

Units	Legal Range	Default	Required	Variability
Btuh	$x > 0$	*none*	Yes	constant

ahccMtr=*mtrName*

Name of meter, if any, to record energy use of air handler cool coil. End use category "Clg" is used. Not used with CHW coils, because the input energy use for a CHW coil is recorded by the COOLPLANT CHILLER

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

Units	Legal Range	Default	Required	Variability

The following six members are used with DX cooling coils.

ahccMinTEvap=*float*

Minimum (effective surface) temperature of coil (evaporator). Represents refrigerant setpoint, or cutout to prevent freezing. Coil model will reduce output to keep simulated coil from getting colder than this, even though it lets supply air get warmer than setpoint. Should default be 35??

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	40°F	No	constant

ahccK1=*float*

Exponent in power relationship expressing coil effectiveness as a function of relative air flow. Used as K1 in the relationship ntu = ntuR * relCfmk1, which says that the "number of transfer units" (on the coil outside or air side) varies with the relative air flow raised to the K1 power. Used with CHW as well as DX coils; for a CHW coil, ntuR in the formula is *ahccNtuoDs*.

Units	Legal Range	Default	Required	Variability
	$x < 0$	-0.4	No	constant

ahccBypass=*float*

Fraction of air flow which does NOT flow through DX cooling coil, for better humidity control. Running less of the air through the coil lets the coil run colder, resulting in greater moisture removal right??.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1$	0	No	constant

The next three members are used in determining the energy input to a DX coil under various load conditions. The input is derived from the full load energy input ratio for four segments of the part load curve. In the following the part load ratio (plr) is the ratio of the actual sensible + latent load on the coil to the coil's capacity. The coil's capacity is ahccCaptRat, adjusted by the coil model for differences between entering air temperature, humidity, and flow rate and the coil rating conditions. The coil may run at less than capacity even at full fan flow, depending on the air temperature change needed, the moisture content of the entering air, and the relative values of between *sfanVfDs* and *ahccVfR*.

full load

plr (part load ratio) = 1.0

Full-load power input is power output times ahhcEirR.

compressor unloading region

1.0 > plr ahhcMinUnldPlr

Power input is the full-load input times the value of the pydxEirUl polynomial (below) for the current plr, i.e. pydxEirUl(plr).

false loading region

ahccMinUnldPlr > plr ahccMinFsldPlr

Power input in this region is constant at the value for the low end of the compressor unloading region, i.e. pydxEirUl(ahccMinUnldPlr).

cycling region

ahccMinFsldPlr > plr 0

In this region the compressor runs at the low end of the false loading region for the necessary fraction of the time, and the power input is the false loading value correspondingly prorated, i.e. pydxEirUl(ahccMinUnldPlr) * plr / ahccMinFsldPlr.

The default values for the following three members are the DOE2 PTAC (Window air conditioner) values.

ahccEirR=*float*

DX compressor energy input ratio (EIR) at full load under rated conditions; defined as the full-load electric energy input divided by the rated capacity, both in Btuh; same as the reciprocal of the Coefficient Of Performance (COP). Polynomials given below are used by CSE to adjust the energy input for part load and for off rated flow and temperature conditions. The default value includes outdoor (condenser) fan energy, but not indoor (air handler supply) fan energy.

Units	Legal Range	Default	Required	Variability
		0.438	No	hourly

ahccMinUnldPlr=*float*

Compressor part load ratio (total current load/current capacity) at/above which "Compressor unloading" is used and pydxEirUl (below) is used to adjust the full-load power input to get the current part load power input.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	1 (no unloading)	No	constant

ahccMinFsldPlr=*float*

"False Loading" is used between this compressor part load ratio and the plr where unloading is activated (*ahccMinUnldPlr*). In this region, input remains at *pydxEirUl*(*ahccMinUnldPlr)*.For plr's less than *ahccMinFsldPlr*, cycling is used, and the power input goes to 0 in a straight line.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq$ *ahccMinUnldPlr*	*ahccMinUnldPlr* (no false loading)	No	constant

The following four inputs specify polynomials to approximate functions giving DX coil capacity and power (energy) input as functions of entering temperatures, relative (to ahccVfR) flow, and relative load (plr). In each case several *float* values may be given, for use as coefficients of the polynomial. The values are ordered from constant to coefficient of highest power. If fewer than the maximum number of values are given, zeroes are used for the trailing (high order) coefficients.

Examples:

```
pydxCaptT = 2.686, -0.01667, 0, 0.006, 0, 0;

pydxCaptT = 2.686, -0.01667, 0, 0.006; // same

pydxEirUl = .9, 1.11, .023, -.00345;
```

If the polynomial does not evaluate to 1.0 when its inputs are equal to the rating conditions (1.0 for relative flows and plr), CSE will normalize your coefficients by dividing them by the non-1.0 value.

Some of the polynomials are biquadratic polynomials whose variables are the entering air wetbulb and drybulb temperatures. These are of the form

$$z = a + bx + cx^2 + dy + ey^2 + fxy$$

where a through f are user-inputtable coefficients, x is the entering wetbulb temperature, y is the entering drybulb temperature, and the polynomial value, z, is a factor by which the coil's capacity, power input, etc. at rated conditions is multiplied to adjust it for the actual entering air temperatures.

Other polynomials are cubic polynomials whose variable is the air flow or load as a fraction of full flow or load.. These are of the form

$$z = a + bx + cx^2 + dx^3$$

where a, b, c, and d are user-inputtable coefficients, x is the variable, and the value z is a factor by which the coil's capacity, power input, etc. at rated conditions is multiplied to adjust it for the actual flow or load.

The default values for the polynomial coefficients are the DOE2 PTAC values.

pydxCaptT=a, b, c, d, e, f

Coefficients of biquadratic polynomial function of entering air wetbulb and condenser temperatures whose value is used to adjust *ahccCaptRat* for the actual entering air temperatures. The condenser temperature is the outdoor drybulb, but not less than 70. See discussion in preceding paragraphs.

Unit s

Legal Range

Default

Requi red

Variabi lity

1.1839345, -0.0081087, 0.00021104, -0.0061425, 0.00000161, -0.0000030

No

constant

pydxCaptF=a=a, b, c, d

Coefficients of cubic polynomial function of relative flow (entering air cfm/*ahccVfR*) whose value is used to adjust *ahccCaptRat* for the actual flow. See discussion in preceding paragraphs.

Units	Legal Range	Default	Required	Variability
		0.8, 0.2, 0.0, 0.0	No	constant

pydxEirT=*a, b, c, d, e, f*

Coefficients of biquadratic polynomial function of entering air wetbulb and condenser temperatures whose value is used to adjust *ahccEirR* for the actual entering air temperatures. The condenser temperature is the outdoor air drybulb, but not less than 70. If the entering air wetbulb is less than 60, 60 is used, in this function only. See discussion in preceding paragraphs.

Unit s

Legal Range

Default

Requi red

Variabi lity

-0.6550461, 0.03889096, -0.0001925, 0.00130464, 0.00013517, -0.0002247

No

constant

pydxEirUl=*a, b, c, d*

Coefficients of cubic polynomial function of part load ratio used to adjust energy input to part load conditions, in the compressor unloading part load region ($1 \geq$ plr $\geq$ *ahccMinUnldPlr*) as described above. See discussion of polynomials in preceding paragraphs.

This polynomial adjusts the full load energy input to part load, not the ratio of input to output, despite the "Eir" in its name.

Units	Legal Range	Default	Required	Variability
		0.125, 0.875, 0.0, 0.0	No	constant

The following four members are used only with CHW coils. In addition, *ahccK1,* described above, is used.

ahccCoolplant=*name*

name of COOLPLANT supporting CHW coil. COOLPLANT contain CHILLER and are described in Section 5.21.

Units	Legal Range	Default	Required	Variability
	name of a COOLPLANT		for CHW coil	constant

ahccGpmDs=*float*

Design (i.e. maximum) water flow through CHW coil.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$		Yes, for CHW coil	constant

ahccNtuoDs=*float*

CHW coil outside number of transfer units at design air flow (ahccVfR*, below). See*ahccK1* above with regard to transfer units at other air flows.

Units	Legal Range	Default	Required	Variability
	$x > 0$	2	No	constant

ahccNtuiDs=*float*

CHW coil inside number of transfer units at design water flow (ahccGpmDs, above).

Units	Legal Range	Default	Required	Variability
	$x > 0$	2	No	constant

The following four members let you give the specification conditions for the cooling coil: the rating conditions, design conditions, or other test conditions under which the coil's performance is known. The defaults are ARI (Air-Conditioning and Refrigeration Institute) standard rating conditions.

ahccDsTDbEn=*float*

Design (rating) entering air dry bulb temperature, used with DX and CHW cooling coils. With CHW coils, this input is used only as the temperature at which to convert *ahccVfR* from volume to mass.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	80°F (ARI)	No	constant

ahccDsTWbEn=*float*

Design (rating) entering air wet bulb temperature, for CHW coils.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	67°F (ARI)	No	constant

ahccDsTDbCnd=*float*

Design (rating) condenser temperature (outdoor air temperature) for DX coils.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	95°F (ARI)	No	constant

ahccVfR=*float*

Design (rating) (volumetric) air flow rate for DX or CHW cooling coil. The ARI specification for this test condition for CHW coils is "450 cfm/ton or less", right??

Units	Legal Range	Default	Required	Variability
cfm	$x > 0$	DX coil: 400cfm/ton* CHW coil: *sfanVfDs*	No	constant

* a "ton" is 12,000 Btuh of rated capacity (*ahccCaptRat*).

The following four members permit specification of auxiliary input power use associated with the cooling coil under the conditions indicated.

ahccAuxOn=*float*

Auxiliary power used when coil is running, in proportion to its subhour average part load ratio (plrAv).

ahccAuxOff=*float*

Auxiliary power used when coil is not running, in proportion to 1 - plrAv.

ahccAuxFullOff=*float*

Auxiliary power used only when coil is off for entire subhour; not used if the coil is on at all during the subhour.

ahccAuxOnAtAll=*float*

Auxiliary power used in full value if coil is on for any fraction of a subhour.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	0	No	hourly

The following four allow specification of meters to record cool coil auxiliary energy use through ahccAuxOn, ahccAuxOff, ahccFullOff, and ahccAuxOnAtAll, respectively. End use category "Aux" is used.

ahccAuxOnMtr=*mtrName*

ahccAuxOffMtr=*mtrName*

ahccAuxFullOffMtr=*mtrName*

ahccAuxOnAtAllMtr=*mtrName*

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

5.19.0.6 AIRHANDLER Outside Air

Outside air introduced into the air hander supply air can be controlled on two levels. First, a *minimum*fraction or volume of outside air may be specified. By default, a minimum volume of .15 cfm per square foot of zone area is used. Second, an *economizer* may be specified. The simulated economizer increases the outside air above the minimum when the outside air is cooler or has lower enthalpy than the return air, in order to reduce cooling coil energy usage. By default, there is no economizer.

oaMnCtrl=*choice*

Minimum outside air flow control method choice, VOLUME or FRACTION. Both computations are based on the minimum outside air flow, *oaVfDsMn*; if the control method is FRACTION, the outside air flow is pro-rated when the air handler is supplying less than its design cfm. In both cases the computed minimum cfm is multiplied by a schedulable fraction, *oaMnFrac*, to let you vary the outside air or turn in off when none is desired.

VOLUME

Volume (cfm) of outside air is regulated:

min_oa_flow = oaMnFrac * oaVfDsMn

FRACTION

Fraction of outside air in supply air is regulated. The fraction is oaVfDsMn divided by sfanVfDs, the air handler supply fan design flow. The minimum cfm of outside air is thus computed as

min_oa_flow = oaMnFrac * curr_flow * oaVfDsMn / sfanVfDs

where curr_flow is the current air handler cfm.

If the minimum outside air flow is greater than the total requested by the terminals served by the air handler, then 100% outside air at the latter flow is used. To insure minimum outside air cfm to the zones, use suitable terminal minimum flows (*tuVfMn*) as well as air handler minimum outside air specifications.

Units	Legal Range	Default	Required	Variability
	VOLUME, FRACTION	VOLUME	No	constant

oaVfDsMn=*float*

Design minimum outside air flow. If *oaMnCtrl* is FRACTION, then this is the minimum outside air flow at full air handler flow. See formulas in *oaMnCtrl* description, just above.

Units	Legal Range	Default	Required	Variability
cfm	$x \geq 0$	0.15 times total area of zones served	No	constant

oaMnFrac=*float*

Fraction of minimum outside air to use this hour, normally 1.0. Use a CSE expression that evaluates to 0 for hours you wish to disable the minimum outside air flow, for example to suppress ventilation during the night or during warm-up hours. Intermediate values may be used for intermediate outside air minima. See formulas in *oaMnCtrl* description, above.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	1.0	No	hourly

CAUTION: the minimum outside air flow only applies when the supply fan is running; it won't assure meeting minimum ventilation requirements when used with ahFanCycles = YES (constant volume, fan cycling).

If an *oaEcoType* choice other than NONE is given, an economizer will be simulated. The economizer will be enabled when the outside temperature is below oaLimT *AND* the outside air enthalpy is below oaLimE. When enabled, the economizer adjusts the economizer dampers to increase the outside air mixed with the return air until the mixture is cooler than the air handler supply temperature setpoint, if possible, or to maximum outside air if the outside air is not cool enough.

CAUTIONS: the simulated economizer is just as dumb as the hardware being simulated. Two considerations particularly require attention. First, if enabled when the outside air is warmer than the return air, it will do the worst possible thing: use 100% outside air. Prevent this by being sure your oaLimT or oaLimE input disables the economizer when the outside air is too warm – or leave the oaLimT = RA default in effect.

Second, the economizer will operate even if the air handler is heating, resulting in use of more than minimum outside air should the return air get above the supply temperature setpoint. Economizers are intended for cooling air handlers; if you heat and cool with the same air handler, consider disabling the economizer when heating by scheduling a very low *oaLimT* or *oaLimE*.

oaEcoType=*choice*

Type of economizer. Choice of:

NONE

No economizer; outside air flow is the minimum.

INTEGRATED

Coil and economizer operate independently.

NONINTEGRATED

Coil does not run when economizer is using all outside air: simulates interlock in some equipment designed to prevent coil icing due to insufficient load, right?

TWO_STAGE

Economizer is disabled when coil cycles on. NOT IMPLEMENTED as of July 1992.

oaLimT=*float*

or RAEconomizer outside air temperature high limit. The economizer is disabled (outside air flow is reduced to a minimum) when the outside air temperature is greater than *oaLimT*. A number may be entered, or "RA" to specify the current Return Air temperature. *OaLimT* may be scheduled to a low value, for example -99, if desired to disable the economizer at certain times.

Units	Legal Range	Default	Required	Variability
°F	*number*or RA	RA (return air temperature)	No	hourly

oaLimE=*float*

or RAEconomizer outside air enthalpy high limit. The economizer is disabled (outside air flow is reduced to a minimum) when the outside air enthalpy is greater than *oaLimE*. A number may be entered, or "RA" to specify the current Return Air enthalpy. *OaLimE* may be scheduled to a low value, for example -99, if desired to disable the economizer at certain times.

Units	Legal Range	Default	Required	Variability
Btu/°F	*number* or RA	999 (enthalpy limit disabled)	No	hourly

oaOaLeak and *oaRaLeak* specify leakages in the economizer dampers, when present. The leaks are constant-cfm flows, expressed as fractions of the maximum possible flow. Thus, when the current flow is less than the maximum possible, the range of operation of the economizer is reduced. When the two damper leakages add up to more than the current air handler flow, outside and return air are used in the ratio of the two leakages and the economizer, if enabled, is ineffective.

oaOaLeak=*float*

Outside air damper leakage to mixed air. Puts a minimum on return air flow and thus a maximum on outside air flow, to mixed air. If an economizer is present, *oaOaLeak* is a fraction of the supply fan design cfm, *sfanVfDs*. Otherwise, *oaOaLeak* is a fraction of the design minimum outside air flow *oaVfDsMn*.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1.0$	0.1	No	constant

oaRaLeak=*float*

Return air damper leakage to mixed air. Puts a minimum on return air flow and thus a maximum on outside air flow, to mixed air. Expressed as a fraction of the supply fan design cfm, *sfanVfDs*. Not used when no economizer is being modeled.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1.0$	0.1	No	constant

5.19.0.7 AIRHANDLER Leaks and Losses

AhSOLeak and *ahRoLeak* express air leaks in the common supply and return ducts, if any, that connect the air handler to the conditioned space. For leakage after the point where a duct branches off to an individual zone, see TERMINAL member *tuSRLeak*. These inputs model leaks in constant pressure (or vacuum) areas nearer the supply fan than the terminal VAV dampers; thus, they are constant volume regardless of flow to the zones. Hence, unless 0 leakage flows are specified, the air handler cfm is greater than the sum of the terminal cfm's, and the air handler cfm is non-0 even when all terminal flows are 0. Any heating or cooling energy applied to the excess cfm is lost to the outdoors.

If unequal leaks are specified, at present (July 1992) CSE will use the average of the two specifications for both leaks, as the modeled supply and return flows must be equal. A future version may allow unequal flows, making up the difference in exfiltration or infiltration to the zones.

ahSOLeak=*float*

Supply duct leakage to outdoors, expressed as a fraction of supply fan design flow (*sfanVfDs*). Use 0 if the duct is indoors. A constant-cfm leak is modeled, as the pressure is constant when the fan is on.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.01	No	constant

ahROLeak=*float*

Return duct leakage FROM outdoors, expressed as a fraction of *sfanVfDs*. Use 0 if the duct is indoors.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.01	No	constant

AhSOLoss and *ahROLoss* represent conductive losses from the common supply and return ducts to the outdoors. For an individual zone's conductive duct loss, see TERMINAL member *tuSRLoss*. Losses here are expressed as a fraction of the temperature difference which is lost. For example, if the supply air temperature is 120, the outdoor temperature is 60, and the pertinent loss is .1, the effect of the loss as modeled will be to reduce the supply air temperature by 6 degrees (.1 * (120 - 60)) to 114 degrees. CSE currently models these losses a constant *TEMPERATURE LOSSes* regardless of cfm.

ahSOLoss=*float*

Supply duct loss/gain to the outdoors.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.1	No	constant

ahROLoss=*float*

Return duct heat loss/gain to the outdoors.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.1	No	constant

5.19.0.8 AIRHANDLER Crankcase Heater

A "crankcase heater" is an electric resistance heater in the crankcase of the compressor of heat pumps and dx cooling coils. The function of the crankcase heater is to keep the compressor's oil warmer than the refrigerant when the compressor is not operating, in order to prevent refrigerant from condensing into and remaining in the oil, which impairs its lubricating properties and shortens the life of the compressor. Manufacturers have come up with a number of different methods for controlling the crankcase heater. The crankcase heater can consume a significant part of the heat pump's energy input; thus, it is important to model it.

In CSE a heat pump is modeled as though it were separate heating and cooling coils. However, the crankcase heater may operate (or not, according to its control method) whether the heat pump is heating, or cooling, or, in particular, doing neither, so it is modeled as a separate part of the air handler, not associated particularly with heating or cooling.

When modeling an air source heat pump (ahhcType = AHP), these variables should be used to specify the crankcase heater, insofar as non-default inputs are desired.

Appropriateness of use of these inputs when specifying a DX system without associated heat pump heating

is not clear to me (Rob) as of 10-23-92; on the one hand, the DX compressor probably has a crankcase heater; on the other hand, the rest of the DX model is supposed to be complete in itself, and adding a crankcase heater here might produce excessive energy input; on the third hand, the DX model does not include any energy input when the compressor is idle; … .

cchCM=*choice*

Crankcase heater presence and control method. Choice of:

NONE

No crankcase heater present

CONSTANT

Crankcase heater input always cchPMx (below).

PTC

Proportional control based on oil temp when compressor does not run in subhour (see cchTMx, cchMn, and cchDT). If compressor runs at all in subhour, the oil is assumed to be hotter than cchTMn and crankcase heater input is cchPMn. (PTC stands for "Positive Temperature Coefficient" or "Proportional Temperature Control".)

TSTAT

Control based on outdoor temperature, with optional differential, during subhours when compressor is off; crankcase heater does not operate if compressor runs at all in subhour. See cchTOn, cchTOff.

CONSTANT_CLO

PTC_CLO

Same as corresponding choices above except zero crankcase heater input during fraction of time compressor is on ("Compressor Lock Out"). There is no TSTAT_CLO because under TSTAT the crankcase heater does not operate anyway when the compressor is on.

Units

Legal Range

Default

Required

**Variability

CONSTANT CONSTANT_CLO PTC PTC_CLO TSTAT NONE

PTC_CLO if ahhcType is AHP, else NONE

No

constant

cchPMx=*float*

Crankcase resistance heater input power; maximum power if *cchCM* is PTC or PTC_CLO.

Units	Legal Range	Default	Required	Variability
kW	$x > 0$	.4 kW	No	constant

cchPMn=*float*

Crankcase heater minimum input power if *cchCM* is PTC or PTC_CLO, disallowed for other *cchCM's*. > 0.

Units	Legal Range	Default	Required	Variability
kW	$x > 0$	.04 kW	No	constant

cchTMx=*float*

cchTMn=*float*

For *cchCM* = PTC or PTC_CLO, the low temperature (max power) and high temperature (min power) setpoints. In subhours when the compressor does not run, crankcase heater input is *cchPMx* when oil temperature is at or below *cchTMx*, *cchPMn* when oil temp is at or above *cchTMn*, and varies linearly (proportionally) in between. *cchTMn* must be ≥ *cchTMx*. See *cchDT* (next).

(Note that actual thermostat setpoints probably cannot be used for *cchTMx* and *cchTMn* inputs, because the model assumes the difference between the oil temperature and the outdoor temperature is constant (*cchDT*) regardless of the heater power.

Units	Legal Range	Default	Required	Variability
°F		*cchTMn*: 0; *cchTMx*: 150	No	constant

cchDT=*float*

For *cchCM* = PTC or PTC_CLO, how much warmer than the outdoor temp CSE assumes the crankcase oil to be in subhours when the compressor does not run. If the compressor runs at all in the subhour, the oil is assumed to be warmer than *cchTMn*.

Units	Legal Range	Default	Required	Variability
°F		20°F	No	constant

cchTOn=*float*

cchTOff=*float*

For *cchCM* = TSTAT, in subhours when compressor does not run, the crankcase heater turn-on and turn-off outdoor temperatures, respectively. Unequal values may be given to simulate thermostat differential. When the compressor runs at all in a subhour, the crankcase heater is off for the entire subhour.

Units	Legal Range	Default	Required	Variability
°F	*cchTOff* ≥ *cchTOn*	*cchTOn*: 72°F; *chcTOff*: *chcTOn*	No	constant

cchMtr=*name of a METER*

METER to record crankcase heater energy use, category "Aux"; not recorded if not given.

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endAirHandler

Indicates the end of the air handler definition. Alternatively, the end of the air handler definition can be indicated by the declaration of another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.20 HEATPLANT

A HEATPLANT contains one or more BOILER subobjects (Section 5.20.1) and supports one or more Hot Water (HW) coils in TERMINAL and/or AIRHANDLER and/or heat exchangers in HPLOOPs (HPLOOPs are not implemented as of September 1992.). There can be more than one HEATPLANT in a simulation.

BOILER HW coils, and heat exchangers are modeled with simple heat-injection models. There is no explicit modeling of circulating hot water temperatures and flows; it is always assumed the temperature and flow at each load (HW coil or heat exchanger) are sufficient to allow the load to transfer any desired amount of heat up to its capacity. When the total heat demand exceeds the plant's capacity, the model reduces the capacity of each load until the plant is not overloaded. The reduced capacity is the same fraction of rated capacity for all loads on the HEATPLANT any loads whose requested heat is less than the reduced capacity are unaffected.

The BOILER in the HEATPLANT can be grouped into *STAGES* of increasing capacity. The HEATPLANT uses the first stage that can meet the load. The load is distributed among the BOILER in the stage so that each operates at the same fraction of its rated capacity.

For each HEATPLANT piping loss is modeled, as a constant fraction of the BOILER capacity of the heatPlant's most powerful stage. This heat loss is added to the load whenever the plant is operating; as modeled, the heat loss is independent of load, weather, or any other variables.

heatplantName

Name of HEATPLANT object, given immediately after the word HEATPLANT This name is used to refer to the heatPlant in *tuhcHeatplant* and *ahhcHeatplant* commands.

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	constant

hpSched=*choice*

Heat plant schedule: hourly variable choice of OFF, AVAIL, or ON.

OFF

HEATPLANT will not supply hot water regardless of demand. All loads (HW coils and heat exchangers) should be scheduled off when the plant is off; an error will occur if a coil calls for heat when its plant is off.

AVAIL

HEATPLANT will operate when one or more loads demand heat.

ON

HEATPLANT runs unconditionally. When no load wants heat, least powerful (first) stage runs.

Units	Legal Range	Default	Required	Variability
	OFF, AVAIL, or ON	AVAIL	No	hourly

hpPipeLossF=*float*

Heat plant pipe loss: heat assumed lost from piping connecting boilers to loads whenever the HEATPLANT is operating, expressed as a fraction of the boiler capacity of the plant's most powerful stage.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.01	No	constant

hpStage1=boilerName, boilerName, boilerName, ...

hpStage1=ALL_BUT, boilerName, boilerName, boilerName, ...

hpStage1=ALL

hpStage2 through hpStage7 *same*

The commands *hpStage1* through *hpStage7* allow specification of up to seven *STAGES* in which BOILER are activated as the load increases. Each stage may be specified with a list of up to seven names of BOILER in the HEATPLANT or with the word ALL, meaning all of the HEATPLANT BOILER or with the word ALL_BUT and a list of up to six names of BOILER Each stage should be more powerful than the preceding one. If you have less than seven stages, you may skip some of the commands *hpStage1* through *hpStage7* – the used stage numbers need not be contiguous.

If none of *hpStage1* through *hpStage7* are given, CSE supplies a single default stage containing all boilers.

A comma must be entered between boiler names and after the word ALL_BUT.

Units	Legal Range	Default	Required	Variability
	1 to 7 names;ALL_BUT and 1 to 6 names;ALL	*hpStage1* = ALL	No	constant

endHeatplant

Optionally indicates the end of the HEATPLANT definition. Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.20.1 BOILER

BOILER are subObjects of HEATPLANT (preceding Section 5.20). BOILER supply heat, through their associated HEATPLANT to HW coils and heat exchangers.

Each boiler has a pump. The pump operates whenever the boiler is in use; the pump generates heat in the water, which is added to the boiler's output. The pump heat is independent of load – the model assumes a bypass valve keeps the water flow constant when the loads are using less than full flow – except that the heat is assumed never to exceed the load.

boilerName

Name of BOILER object, given immediately after the word BOILER The name is used to refer to the boiler in heat plant stage commands.

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	constant

blrCap=*float*

Heat output capacity of this BOILER

Units	Legal Range	Default	Required	Variability
Btuh	$x > 0$		Yes	constant

blrEffR=*float*

Boiler efficiency at steady-state full load, as a fraction. 1.0 may be specified to model a 100% efficient boiler.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.80	No	constant

blrEirR=*float*

Boiler Energy Input Ratio: alternate method of specifying efficiency.

Units	Legal Range	Default	Required	Variability
	x $\geq$ 1.0	1/*blrEffR*	No	constant

blrPyEi=*a, b, c, d*

Coefficients of cubic polynomial function of part load ratio (load/capacity) to adjust full-load energy input for part load operation. Up to four floats may be given, separated by commas, lowest order (i.e. constant term) coefficient first. If the given coefficients result in a polynomial whose value is not 1.0 when the input variable, part load ratio, is 1.0, a warning message will be printed and the coefficients will be normalized to produce value 1.0 at input 1.0.

Units	Legal Range	Default	Required	Variability
		.082597, .996764, 0.79361, 0.	No	constant

blrMtr=*name of a METER*

Meter to which Boiler's input energy is accumulated; if omitted, input energy is not recorded.

Units	Legal Range	Default	Required	Variability
	name of a METER	*none*	No	constant

blrpGpm=*float*

Boiler pump flow in gallons per minute: amount of water pumped from this boiler through the hot water loop supplying the HEATPLANT loads (HW coils and heat exchangers) whenever boiler is operating.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$	blrCap/10000	No	constant

blrpHdloss=*float*

Boiler pump head loss (pressure). 0 may be specified to eliminate pump heat and pump energy input.

Units	Legal Range	Default	Required	Variability

Units	Legal Range	Default	Required	Variability
ft H2O	$x \geq 0$	114.45*	No	constant

* may be temporary value for 10-31-92 version; prior value of 35 may be restored.

blrpMotEff=*float*

Boiler pump motor efficiency.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.88	No	constant

blrpHydEff=*float*

Boiler pump hydraulic efficiency

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.70	No	constant

blrpMtr=*name of a METER*

Meter to which pump electrical input energy is accumulated. If omitted, pump input energy use is not recorded.

Units	Legal Range	Default	Required	Variability
	name of a METER	*none*	No	constant

The following four members permit specification of auxiliary input power use associated with the boiler under the conditions indicated.

blrAuxOn=float

Auxiliary power used when boiler is running, in proportion to its subhour average part load ratio (plr).

blrAuxOff=float

Auxiliary power used when boiler is not running, in proportion to 1 - plr.

blrAuxFullOff=float

Auxiliary power used only when boiler is off for entire subhour; not used if the boiler is on at all during the subhour.

blrAuxOnAtAll=float

Auxiliary power used in full value if boiler is on for any fraction of subhour.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	0	No	hourly

The following four allow specification of meters to record boiler auxiliary energy use through blrAuxOn,

blrAuxOff, blrFullOff, and blrAuxOnAtAll, respectively. End use category "Aux" is used.

blrAuxOnMtr=*mtrName*

blrAuxOffMtr=*mtrName*

blrAuxFullOffMtr=*mtrName*

blrAuxOnAtAllMtr=*mtrName*

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endBoiler

Optionally indicates the end of the boiler definition. Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.21 COOLPLANT

A COOLPLANT contains one or more CHILLER subobjects (Section 5.21.1). Each COOLPLANT supports one or more CHilled Water (CHW) cooling coils in AIRHANDLER and is supported by a TOWERPLANT (Section 5.22). The piping circuit connecting the cold-water (evaporator) side of the CHILLER to the CHW coils is referred to as the *primary loop*; the piping connecting the warm-water (condenser) side of the CHILLER to the cooling towers in the TOWERPLANT is referred to as the *secondary loop*. Flows in these loops are established primary and secondary (or heat rejection) by pumps in each CHILLER these pumps operate when the CHILLER operates.

The modeling of the CHW coils, COOLPLANT and CHILLER includes modeling the supply temperature of the water in the primary loop, that is, the water supplied from the COOLPLANT operating CHILLER to the CHW coils. If the (negative) heat demanded by the connected coils exceeds the plant's capacity, the temperature rises and the available power is distributed among the AIRHANDLER according to the operation of the CHW coil model.

The primary water flow through each CHILLER is always at least that CHILLER specified primary pump capacity – it is assumed that any flow in excess of that used by the coils goes through a bypass value. When the coils request more flow than the pump's capacity, it is assumed the pressure drops and the pump can deliver the greater flow at the same power input and while imparting the same heat to the water. The primary water flow is not simulated during the run, but an error occurs before the run if the total design flow of the CHW coils connected to a COOLPLANT exceeds the pumping capacity of the CHILLER in the plant's most powerful stage.

The CHILLER in the COOLPLANT can be grouped into *STAGES* of increasing capacity. The COOLPLANT uses the first stage that can meet the load. The load is distributed amoung the CHILLER in the active stage so that each operates at the same fraction of its capacity; CHILLER not in the active stage are turned off.

For each COOLPLANT primary loop piping loss is modeled, as a heat gain equal to a constant fraction of the CHILLER capacity of the COOLPLANT most powerful stage. This heat gain is added to the load whenever the plant is operating; as modeled, the heat gain is independent of load, weather, which stage is operating, or any other variables. No secondary loop piping loss is modeled.

coolplantName

Name of COOLPLANT object, given immediately after the word COOLPLANT This name is used to refer to the coolPlant in *ahhcCoolplant* commands.

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	constant

cpSched=*choice*

Coolplant schedule: hourly variable choice of OFF, AVAIL, or ON.

OFF

COOLPLANT will not supply chilled water regardless of demand. All loads (CHW coils) should be scheduled off when the plant is off; an error will occur if a coil calls for chilled water when its plant is off.

AVAIL

COOLPLANT will operate when one or more loads demand chilled water.

ON

COOLPLANT runs unconditionally. When no load wants chilled water, least powerful (first) stage runs anyway.

Units	Legal Range	Default	Required	Variability
	OFF, AVAIL, or ON	AVAIL	No	hourly

cpTsSp=*float*

Coolplant primary loop supply temperature setpoint: setpoint temperature for chilled water supplied to coils.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	44	No	hourly

cpPipeLossF=*float*

Coolplant pipe loss: heat assumed gained from primary loop piping connecting chillers to loads whenever the COOLPLANT is operating, expressed as a fraction of the chiller capacity of the plant's most powerful stage.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.01	No	constant

cpTowerplant=*name*

TOWERPLANT that cools the condenser water for the chillers in this COOLPLANT

Units	Legal Range	Default	Required	Variability
	name of a TOWERPLANT		Yes	constant

cpStage1=chillerName, chillerName, chillerName, ...

cpStage1=ALL_BUT, chillerName, chillerName, chillerName, ...

cpStage1=ALL

cpStage2 through cpStage7 ***same***

The commands *cpStage1* through *cpStage7* allow specification of up to seven *STAGES* in which chillers are activated as the load increases. CSE will use the first stage that can meet the load; if no stage will meet the load (output the heat requested by the coils at *cpTsSp*), the last COOLPLANT stage is used.

Each stage may be specified with a list of up to seven names of CHILLER in the COOLPLANT or with the word ALL, meaning all of the COOLPLANT CHILLER or with the word ALL_BUT and a list of up to six names of CHILLER Each stage should be more powerful than the preceding one. If you have less than seven stages, you may skip some of the commands *cpStage1* through *cpStage7* – the used stage numbers need not be contiguous.

If none of *cpStage1* through *cpStage7* are given, CSE supplies a single default stage containing all chillers.

A comma must be entered between chiller names and after the word ALL_BUT.

Units	Legal Range	Default	Required	Variability
	1 to 7 names; ALL_BUT and 1 to 6 names; ALL	*cpStage1* = ALL	No	constant

endCoolplant

Optionally indicates the end of the COOLPLANT definition. Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.21.1 CHILLER

CHILLER are subobjects of COOLPLANT (Section 5.21). CHILLER supply coldness, in the form of chilled water, via their COOLPLANT to CHW (CHilled Water) cooling coils in AIRHANDLER CHILLER exhaust heat through the cooling towers in their COOLPLANT TOWERPLANT Each COOLPLANT can contain multiple CHILLER chiller operation is controlled by the scheduling and staging logic of the COOLPLANT as described in the previous section.

Each chiller has primary and secondary pumps that operate when the chiller is on. The pumps add heat to the primary and secondary loop water respectively; this heat is considered in the modeling of the loop's water temperature.

chillerName

Name of CHILLER object, given immediately after the word CHILLER This name is used to refer to the chiller in *cpStage* commands.

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	constant

The next four inputs allow specification of the CHILLER capacity (amount of heat it can remove from the primary loop water) and how this capacity varies with the supply (leaving) temperature of the primary loop water and the entering temperature of the condenser (secondary loop) water. The chiller capacity at any supply and condenser temperatures is *chCapDs* times the value of *chPyCapT* at those temperatures.

chCapDs=*float*

Chiller design capacity, that is, the capacity at *chTsDs* and *chTcndDs* (next).

Units	Legal Range	Default	Required	Variability
Btuh	x != 0		Yes	constant

chTsDs=*float*

Design supply temperature: temperature of primary water leaving chiller at which capacity is *chCapDs*.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	44	No	constant

chTcndDs=*float*

Design condenser temperature: temperature of secondary water entering chiller condenser at which capacity is *chCapDs*.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	85	No	constant

chPyCapT=*a, b, c, d, e, f*

Coefficients of bi-quadratic polynomial function of supply (ts) and condenser (tcnd) temperatures that specifies how capacity varies with these temperatures. This polynomial is of the form

$$a + b \cdot \boldsymbol{ts} + c \cdot \boldsymbol{ts}^2 + d \cdot \boldsymbol{tcnd} + e \cdot \boldsymbol{tcnd}^2 + f \cdot \boldsymbol{ts} \cdot \boldsymbol{tcnd}$$

Up to six *float* values may be entered, separated by commas; CSE will use zero for omitted trailing values. If the polynomial does not evaluate to 1.0 when ts is chTsDs and tcnd is chTcndDs, a warning message will be issued and the coefficients will be adjusted (normalized) to make the value 1.0.

Units	Legal Range	Default	Required	Variability
		-1.742040, .029292, .000067, .048054, .000291, -.000106	No	constant

The next three inputs allow specification of the CHILLER full-load energy input and how it varies with supply and condenser temperature. Only one of *chCop* and *chEirDs* should be given. The full-load energy input at any supply and condenser temperatures is the chiller's capacity at these temperatures, times *chEirDs* (or $1/chCop$), times the value of *chPyEirT* at these temperatures.

chCop=*float*

Chiller full-load COP (Coefficient Of Performance) at *chTsDs*and *chTcndDs*. This is the output energy divided by the electrical input energy (in the same units) and reflects both motor and compressor efficiency.

Units	Legal Range	Default	Required	Variability
	$x > 0$	4.2	No	constant

chEirDs=*float*

Alternate input for COP: Full-load Energy Input Ratio (electrical input energy divided by output energy) at design temperatures; the reciprocal of *chCop*.

Units	Legal Range	Default	Required	Variability
	$x > 0$	*chCop* is defaulted	No	constant

chPyEirT=*a, b, c, d, e, f*

Coefficients of bi-quadratic polynomial function of supply (ts) and condenser (tcnd) temperatures that specifies how energy input varies with these temperatures. This polynomial is of the form

$$a + b \cdot \boldsymbol{ts} + c \cdot \boldsymbol{ts}^2 + d \cdot \boldsymbol{tcnd} + e \cdot \boldsymbol{tcnd}^2 + f \cdot \boldsymbol{ts} \cdot \boldsymbol{tcnd}$$

Up to six *float* values may be entered, separated by commas; CSE will use zero for omitted trailing values. If the polynomial does not evaluate to 1.0 when ts is chTsDs and tcnd is chTcndDs, a warning message will be issued and the coefficients will be adjusted (normalized) to make the value 1.0.

Units	Legal Range	Default	Required	Variability
		3.117600, -.109236, .001389, .003750, .000150, -.000375	No	constant

The next three inputs permit specification of the CHILLER part load energy input. In the following the part load ratio (plr) is defined as the actual load divided by the capacity at the current supply and condenser temperatures. The energy input is defined as follows for four different plr ranges:

full

loadplr (part load ratio) = 1.0

Power input is full-load input, as described above.

compressor unloading region

1.0 > plr chMinUnldPlr

Power input is the full-load input times the value of the chPyEirUl polynomial for the current plr, that is, chPyEirUl(plr).

false loading region

chMinUnldPlr > plr > chMinFsldPlr

Power input in this region is constant at the value for the low end of the compressor unloading region, i.e. chPyEirUl(chMinUnldPlr).

cycling region

chMinFsldPlr > plr 0

In this region the chiller runs at the low end of the false loading region for the necessary fraction of the time, and the power input is the false loading value correspondingly prorated, i.e. chPyEirUl(chMinUnldPlr) plr / chMinFsldPlr.

These plr regions are similar to those for a DX coil & compressor in an AIRHANDLER Section 0.

chPyEirUl=*a, b, c, d*

Coefficients of cubic polynomial function of part load ratio (plr) that specifies how energy input varies with plr in the compressor unloading region (see above). This polynomial is of the form

$$a + b \cdot plr + c \cdot plr^2 + d \cdot plr^3$$

Up to four *float* values may be entered, separated by commas; CSE will use zero for omitted trailing values. If the polynomial does not evaluate to 1.0 when plr is 1.0, a warning message will be issued and the coefficients will be adjusted (normalized) to make the value 1.0.

Units	Legal Range	Default	Required	Variability
		.222903, .313387, .463710, 0.	No	constant

chMinUnldPlr=*float*

Minimum compressor unloading part load ratio (plr); maximum false loading plr. See description above.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0.1	No	constant

chMinFsldPlr=*float*

Minimum compressor false loading part load ratio (plr); maximum cycling plr. See description above.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq$ *chMinFsldPlr*	0.1	No	constant

chMotEff=*float*

Fraction of CHILLER compressor motor input power which goes to the condenser. For an open-frame motor and compressor, where the motor's waste heat goes to the air, enter the motor's efficiency: a fraction around .8 or .9. For a hermetic compressor, where the motor's waste heat goes to the refrigerant and thence to the condenser, use 1.0.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1$	1.0	No	constant

chMeter=*name*

Name of METER to which to accumulate CHILLER electrical input energy. Category "Clg" is used. Note that two additional commands, *chppMtr* and *chcpMtr*, are used to specify meters for recording chiller pump input energy.

Units	Legal Range	Default	Required	Variability
	name of a METER	not recorded	No	constant

The next six inputs specify this CHILLER *PRIMARY PUMP*, which pumps chilled water from the chiller through the CHW coils connected to the chiller's COOLPLANT

chppGpm=*float*

Chiller primary pump flow in gallons per minute: amount of water pumped from this chiller through the primary loop supplying the COOLPLANT loads (CHW coils) whenever chiller is operating. Any excess flow over that demanded by coils is assumed to go through a bypass valve. If coil flows exceed *chppGpm*, CSE assumes the pressure drops and the pump "overruns" to deliver the extra flow with the same energy input. The default is one gallon per minute for each 5000 Btuh of chiller design capacity.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$	*chCapDs* / 5000	No	constant

chppHdloss=*float*

Chiller primary pump head loss (pressure). 0 may be specified to eliminate pump heat and pump energy input.

Units	Legal Range	Default	Required	Variability
ft H2O	$x \geq 0$	57.22*	No	constant

* May be temporary default for 10-31-92 version; prior value (65) may be restored.

chppMotEff=*float*

Chiller primary pump motor efficiency.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.88	No	constant

chppHydEff=*float*

Chiller primary pump hydraulic efficiency

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.70	No	constant

chppOvrun=*float*

Chiller primary pump maximum overrun: factor by which flow demanded by coils can exceed *chppGpm*. The primary flow is not simulated in detail; *chppOvrun* is currently used only to issue an error message if the sum of the design flows of the coils connected to a COOLPLANT exceeds the sum of the products of *chppGpm* and *chppOvrun* for the chiller's in the plants most powerful stage.

Units	Legal Range	Default	Required	Variability
	$x \geq 1.0$	1.3	No	constant

chppMtr=*name of a METER*

Meter to which primary pump electrical input energy is accumulated. If omitted, pump input energy use is not recorded.

Units	Legal Range	Default	Required	Variability
	name of a METER	*none*	No	constant

The next five inputs specify this CHILLER *CONDENSER PUMP*, also known as the *SECONDARY PUMP* or the *HEAT REJECTION PUMP*. This pump pumps water from the chiller's condenser through the cooling towers in the COOLPLANT TOWERPLANT

chcpGpm=*float*

Chiller condenser pump flow in gallons per minute: amount of water pumped from this chiller through the cooling towers when chiller is operating.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$	*chCapDs* / 4000	No	constant

chcpHdloss=*float*

Chiller condenser pump head loss (pressure). 0 may be specified to eliminate pump heat and pump energy input.

Units	Legal Range	Default	Required	Variability
ft H2O	$x \geq 0$	45.78*	No	constant

* May be temporary default for 10-31-92 version; prior value (45) may be restored.

chcpMotEff=*float*

Chiller condenser pump motor efficiency.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.88	No	constant

chcpHydEff=*float*

Chiller condenser pump hydraulic efficiency

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1.0$	.70	No	constant

chcpMtr=*name of a METER*

Meter to which condenser pump electrical input energy is accumulated. If omitted, pump input energy use is not recorded.

Units	Legal Range	Default	Required	Variability
	name of a METER	*none*	No	constant

The following four members permit specification of auxiliary input power use associated with the chiller under the conditions indicated.

chAuxOn=*float*

Auxiliary power used when chiller is running, in proportion to its subhour average part load ratio (plr).

chAuxOff=*float*

Auxiliary power used when chiller is not running, in proportion to 1 - plr.

chAuxFullOff=*float*

Auxiliary power used only when chiller is off for entire subhour; not used if the chiller is on at all during the subhour.

chAuxOnAtAll=*float*

Auxiliary power used in full value if chiller is on for any fraction of subhour.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	0	No	hourly

The following four allow specification of meters to record chiller auxiliary energy use through chAuxOn, chAuxOff, chFullOff, and chAuxOnAtAll, respectively. End use category "Aux" is used.

chAuxOnMtr=*mtrName*

chAuxOffMtr=*mtrName*

chAuxFullOffMtr=*mtrName*

chAuxOnAtAllMtr=*mtrName*

Units	Legal Range	Default	Required	Variability
	name of a METER	*not recorded*	No	constant

endChiller

Optionally indicates the end of the CHILLER definition. Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.22 TOWERPLANT

A TOWERPLANT object simulates a group of cooling towers which operate together to cool water for one or more CHILLER and/or HPLOOP heat exchangers. There can be more than one TOWERPLANT in a simulation. Each CHILLER or hploop heat exchanger contains a pump (the "heat rejection pump") to circulate water through its associated TOWERPLANT The circulating water is cooled by evaporation and conduction to the air; cooling is increased by operating fans in the cooling towers as necessary. These fans are the only energy consuming devices simulated in the TOWERPLANT

The TOWERPLANT models the leaving water temperature as a function of the entering water temperature, flow, outdoor air temperature, and humidity. The fans are operated as necessary to achieve a specified leaving water temperature setpoint, or as close to it as achievable.

Two methods of staging the cooling tower fans in a TOWERPLANT are supported: "TOGETHER", under which all the tower fans operate together, at the same speed or cycling on and off at once, and "LEAD", in which a single "lead" tower's fan modulates for fine control of leaving water temperature, and as many additional towers fans as necessary operate at fixed speed. The water flows through all towers even when their fans are off; sometimes this will cool the water below setpoint with no fans operating.

All the towers in a TOWERPLANT are identical, except that under LEAD staging, the towers other than the lead tower have one-speed fans. The group of towers can thus be described by giving the description of one tower, the number of towers, and the type of staging to be used. All of this information is given by TOWERPLANT members, so there is no need for individual TOWER objects.

There is no provision for scheduling a TOWERPLANT: it operates whenever the heat rejection pump in one or more of its associated CHILLER or HPLOOP heat exchangers operates. However, the setpoint for the water leaving the TOWERPLANT is hourly schedulable.

towerplantName

Name of TOWERPLANT object, given immediately after the word TOWERPLANT to begin the object's input. The name is used to refer to the TOWERPLANT in COOLPLANT and HPLOOPs.

Units	Legal Range	Default	Required	Variability
	63 characters		Yes	constant

tpTsSp=*float*

Setpoint temperature for water leaving towers.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	85	No	hourly

tpMtr=*name of a METER*

METER object by which TOWERPLANT fan input energy is to be recorded, in category "Aux". If omitted, energy use is not recorded, and thus cannot be reported. Towerplants have no modeled input energy other than for their fans (the heat rejection pumps are part of the CHILLER and HPLOOP objects).

Units	Legal Range	Default	Required	Variability
	name of a METER	*none*	No	constant

tpStg=*choice*

How tower fans are staged to meet the load:

TOGETHER

All fans operate at the same speed or cycle on and off together.

LEAD

A single "Lead" tower's fan is modulated as required and as many additional fans as necessary run at their (single) full speed.

Whenever the heat rejection pump in a CHILLER or HPLOOP heat exchanger is on, the water flows through all towers in the TOWERPLANT regardless of the number of fans operating.

Units	Legal Range	Default	Required	Variability
	TOGETHER, LEAD	TOGETHER	No	constant

ctN=*integer*

Number of towers in the TOWERPLANT

Units	Legal Range	Default	Required	Variability
	$x > 0$	1	No	constant

ctType=*choice*

Cooling tower fan control type: ONESPEED, TWOSPEED, or VARIABLE. This applies to all towers under TOGETHER staging. For LEAD staging, *ctType* applies only to the lead tower; additional towers have ONESPEED fans.

Units	Legal Range	Default	Required	Variability
	ONESPEED, TWOSPEED, VARIABLE	ONESPEED	No	constant

ctLoSpd=*float*

Low speed for TWOSPEED fan, as a fraction of full speed cfm.

Units	Legal Range	Default	Required	Variability
	$0 < x \leq 1$	0.5	No	constant

Note: full speed fan cfm is given by *ctVfDs*, below.

The rest of the input variables apply to each tower in the group; the towers are identical except for the single-speed fan on non-lead towers when *tpStg* is LEAD.

The following two inputs permit computation of the tower fan electrical energy consumption:

ctShaftBhp=*float*

Shaft brake horsepower of each tower fan motor.

The default value is the sum of the rejected (condenser) heats (including pump heat) at design conditions of the most powerful stage of each connected COOLPLANT plus the design capacity of each connected HPLOOP heat exchanger, all divided by 290,000 and by the number of cooling towers in the TOWERPLANT

Units	Lgl Range	Default	Req'd	Variability
Bhp	$x > 0$	(sum of loads)/290000/cTn	No	constant

ctMotEff=*float*

Motor (and drive, if any) efficiency for tower fans.

Units	Legal Range	Default	Required	Variability
	$x > 0$	.88	No	constant

The next four items specify the coefficients of polynomial curves relating fan power consumption to average speed (cfm) for the various fan types. For the non-variable speed cases CSE uses linear polynomials of the form

$$p = a + b \cdot \text{spd}$$

where p is the power consumption as a fraction of full speed power consumption, and *spd* is the average speed as a fraction of full speed. The linear relationship reflects the fact that the fans cycle to match partial loads. A non-0 value may be given for the constant part a to reflect start-stop losses. For the two speed fan, separate polynomials are used for low and high speed operation; the default coefficients assume power input varies with the cube of speed, that is, at low speed (*ctLoSpd*) the relative power input is *ctLoSpd3*. For the variable speed case a cubic polynomial is used.

For each linear polynomial, two *float* expressions are given, separated by a comma. The first expression is the constant, *a*. The second expression is the coefficient of the average speed, *b*. Except for *ctFcLo*, *a* and *b* should add up to 1, to make the relative power consumption 1 when *spd* is 1; otherwise, CSE will issue a warning message and normalize them.

ctFcOne=*a, b*

Coefficients of linear fan power consumption polynomial $p = a+b\cdot$spd for ONESPEED fan. For the one-speed case, the relative average speed *spd* is the fraction of the time the fan is on.

Units	Legal Range	Default	Required	Variability
	a + b = 1.0	0, 1	No	constant

ctFcLo=*a, b*

Coefficients of linear fan power consumption polynomial $p = a + b \cdot$ spd for low speed of TWOSPEED fan, when *spd* $\leq$ *ctLoSpd*.

Units	Legal Range	Default	Required	Variability
	a + b = 1.0	0, $ctLoSpd^2$	No	constant

ctFcHi=*a, b*

Coefficients of linear fan power consumption polynomial $p = a + b \cdot$ spd for high speed of TWOSPEED fan, when *spd* $>$ *ctLoSpd*.

Units	Legal Range	Default	Required	Variability
	a + b = 1.0	$-ctLoSpd^2$ - *ctLoSpd*, $ctLoSpd^2$ + *ctLoSpd + 1*	No	constant

ctFcVar=*a, b, c, d*

For VARIABLE speed fan, four *float* values for coefficients of cubic fan power consumption polynomial of the form $p = a + b \cdot \text{spd} + c \cdot \text{spd}^2 + d \cdot \text{spd}^3$.

Units	Legal Range	Default	Required	Variability
	a + b + c + d = 1.0	0, 0, 0, 1	No	constant

The next six items specify the tower performance under one set of conditions, the "design conditions". The conditions should be chosen to be representative of full load operating conditions.

ctCapDs=*float*

Design capacity: amount of heat extracted from water under design conditions by one tower.

The default value is the sum of the rejected (condenser) heats (including pump heat) at design conditions of the most powerful stage of each connected COOLPLANT plus the design capacity of each connected HPLOOP heat exchanger, all divided by the number of towers.

Units	Legal Range	Default	Required	Variability
Btuh	$x \neq 0$	(sum of loads)/*ctN*	No	constant

ctVfDs=*float*

Design air flow, per tower; also the fan full-speed cfm specification.

The default value is the sum of the loads (computed as for *ctCapDs*, just above) divided by 51, divided by the number of cooling towers.

Units	Legal Range	Default	Required	Variability
cfm	$x > 0$	(sum of loads)/51/*ctN*	No	constant

ctGpmDs=*float*

Design water flow, per tower.

The default is the sum of the flows of the connected heat rejection pumps, using the largest stage for COOLPLANT divided by the number of towers.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$	(sum of pumps)/*ctN*	No	constant

ctTDbODs=*float*

Design outdoor drybulb temperature (needed to convert *ctVfDs* from cfm to lb/hr).

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	93.5	No	constant

ctTWbODs=*float*

Design outdoor wetbulb temperature.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	78	No	constant

ctTwoDs=*float*

Design leaving water temperature.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	85	No	constant

The following six items allow optional specification of tower performance under another set of conditions, the "off design" conditions. If given, they allow CSE to compute the tower's relation between flows and heat transfer; in this case, *ctK* (below) may not be given.

ctCapOd=*float*

Off-design capacity, per tower.

Units	Legal Range	Default	Required	Variability
Btuh	$x \neq 0$	(sum of loads)/*ctN*	No	constant

ctVfOd=*float*

Off-design air flow, per tower. Must differ from design air flow; thus *ctVfDs* and *ctVfOd* cannot both be defaulted if off-design conditions are being given. The off-design air and water flows must be chosen so that maOd/mwOd ≠ maDs/mwDs.

Units	Legal Range	Default	Required	Variability
cfm	$x > 0$; $x \neq ctVfDs$	(sum of loads)/51/*ctN*	No	constant

ctGpmOd=*float*

Off-design water flow, per tower. Must differ from design water flow; thus, both cannot be defaulted if off-design conditions are being given. Value must be chosen so that maOd/mwOd ≠ maDs/mwDs.

Units	Legal Range	Default	Required	Variability
gpm	$x > 0$; $x \neq ctGpmDs$	(sum of pumps)/*ctN*	No	constant

ctTDbOOd=*float*

Off-design outdoor drybulb temperature.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	93.5	No	constant

ctTWbOOd=*float*

Off-design outdoor wetbulb temperature.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	78	No	constant

ctTwoOd=*float*

Off-design leaving water temperature.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	85	No	constant

The following item allows explicit specification of the relationship between flows and heat transfer, when the preceding "off design" inputs are not given. If omitted, it will be computed from the "off design" inputs if given, else the default value of 0.4 will be used.

ctK=*float*

Optional. Exponent in the formula

$$\text{ntuA} = k \cdot (mwi/ma)^{ctK}$$

where ntuA is the number of transfer units on the air side, mwi and ma are the water and air flows respectively, and k is a constant.

Units	Legal Range	Default	Required	Variability
	$0 < x < 1$	*from "Od" members if given, else* 0.4	No	constant

ctStkFlFr=*float*

Fraction of air flow which occurs when tower fan is off, due to stack effect (convection). Cooling due to this air flow occurs in all towers whenever the water flow is on, and may, by itself, cool the water below the setpoint *tpTsSp*. Additional flow, when fan is on, is proportional to fan speed.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.18	No	constant

The following items allow CSE to compute the effect of makeup water on the leaving water temperature.

ctBldn=*float*

Blowdown rate: fraction of inflowing water that is bled from the sump down the drain, to reduce the buildup of impurities that don't evaporate.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	.01	No	constant

ctDrft=*float*

Drift rate: fraction of inflowing water that is blown out of tower as droplets without evaporating.

Units	Legal Range	Default	Required	Variability
	$0 \leq x \leq 1$	0	No	constant

ctTWm=*float*

Temperature of makeup water from mains, used to replace water lost by blowdown, drift, and evaporation. Blowdown and drift are given by the preceding two inputs; evaporation is computed.

Units	Legal Range	Default	Required	Variability
°F	$x > 0$	60	No	constant

endTowerplant

Optionally indicates the end of the TOWERPLANT definition. Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.23 HPLOOP

To be written.

5.24 REPORTFILE

REPORTFILE allows optional specification of different or additional files to receive CSE reports.

By default, CSE generates several "reports" on each run showing the simulated HVAC energy use, the CIDL input statements specifying the run, any error or warning messages, etc. Different or additional reports can be specified using the REPORT object, described in Section 5.25, next.

All CSE reports are written to text files as plain ASCII text. The files may be printed (on most printers other than postscript printers) by copying them to your printer with the COPY command. Since many built-in reports are over 80 characters wide; you may want to set your printer for "compressed" characters or a small font first. You may wish to examine the report file with a text editor or LIST program before printing it. (?? Improve printing discussion)

By default, the reports are output to a file with the same name as the input file and extension .REP, in the same directory as the input file. By default, this file is formatted into pages, and overwrites any existing file of the same name without warning. CSE automatically generates a REPORTFILE object called "Primary" for this report file, as though the following input had been given:

```
REPORTFILE "Primary"
    rfFileName = <inputFile>.REP;
    // other members defaulted: rfFileStat=OVERWRITE; rfPageFmt=YES.
```

Using REPORTFILE you can specify additional report files. REPORT specified within a REPORTFILE object definition are output by default to that file; REPORT specified elsewhere may be directed to a specific report file with the REPORT member *rpReportFile*. Any number of REPORTFILE and REPORT may be used in a run or session. Any number of REPORT can be directed to each REPORTFILE

Using ALTER (Section 4.5.1.2) with REPORTFILE you can change the characteristics of the Primary report output file. For example:

```
ALTER REPORTFILE Primary
    rfPageFmt = NO;      // do not format into pages
    rfFileStat = NEW;    // error if file exists
```

rfName

Name of REPORTFILE object, given immediately after the word REPORTFILE Note that this name, not the fileName of the report file, is used to refer to the REPORTFILE in REPORT

Units	Legal Range	Default	Required	Variability
	63 characters		No	constant

rfFileName=*path*

path name of file to be written. If no path is specified, the file is written in the current directory. The default extension is .REP.

Units	Legal Range	Default	Required	Variability
	file name, path and extension optional		Yes	constant

rfFileStat=*choice*

Choice indicating what CSE should do if the file specified by *rfFileName*already exists:

OVERWRITE

Overwrite pre-existing file.

NEW

Issue error message if file exists at beginning of session. If there are several runs in session using same file, output from runs after the first will append.

APPEND

Append new output to present contents of existing file.

If the specified file does not exist, it is created and *rfFileStat* has no effect.

Units	Legal Range	Default	Required	Variability
	OVERWRITE, NEW, APPEND	OVERWRITE	No	constant

rfPageFmt=*Choice*

Choice controlling page formatting. Page formatting consists of dividing the output into pages (with form feed characters), starting a new page before each report too long to fit on the current page, and putting headers and footers on each page. Page formatting makes attractive printed output but is a distraction when examining the output on the screen and may inappropriate if you are going to further process the output with another program.

Yes

Do page formatting in this report file.

No

Suppress page formatting. Output is continuous, uninterrupted by page headers and footers or large blank spaces.

Units	Legal Range	Default	Required	Variability
	Yes, No	Yes	No	constant

Unless page formatting is suppressed, the page formats for all report files are controlled by the TOP members *repHdrL, repHdrR, repLPP, repTopM, repBotM,*and *repCPL*, described in Section 5.1.

Each page header shows the *repHdrL* and *repHdrR* text, if given.

Each page footer shows the input file name, run serial number within session (see *runSerial* in Section 5.1), user-input *runTitle* (see Section 5.1), date and time of run, and page number in file.

Vertical page layout is controlled by *repLPP, repTopM,* and *repBotM* (Section 5.1). The width of each header and footer is controlled by *repCPL*. Since many built-in reports are now over 80 columns wide, you may want to use repCPL=120 or repCPL=132 to make the headers and footers match the text better.

In addition to report file *page* headers and footers, individual REPORT have *REPORT* headers and footers related to the report content. These are described under REPORT Section 5.25.

endReportFile

Optionally indicates the end of the report file definition. Alternatively, the end of the report file definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.25 REPORT

REPORT generates a report object to specify output of specific textual information about the results of the run, the input data, the error messages, etc. The various report types available are enumerated in the description of *rpType* in this section, and may be described at greater length in Section 6.

REPORT are output by CSE to files, via the REPORTFILE object (previous section). After CSE has completed, you may print the report file(s), examine them with a text editor or by TYPEing, process them with another program, etc., as desired.

REPORT that you do not direct to a different file are written to the automatically-supplied "Primary" report file, whose file name is (by default) the input file name with the extension changed to .REP.

Each report consists of a report header, one or more data rows, and a report footer. The header gives the report type (as specified with *rpType*, described below), the frequency (as specified with *rpFreq)*, the month or date where appropriate, and includes headings for the report's columns where appropriate.

Usually a report has one data row for each interval being reported. For example, a daily report has a row for each day, with the day of the month shown in the first column.

The report footer usually contains a line showing totals for the rows in the report.

The header-data-footer sequence is repeated as necessary. For example, a daily report extending over more than one month has a header-data-footer sequence for each month. The header shows the month name; the data rows show the day of the month; the footer contains totals for the month.

In addition to the headers and footers of individual reports, the report file has (by default) *page headers* and *footers*, described in the preceding section.

Default Reports: CSE generates the following reports by default for each run, in the order shown. They are output by default to the "Primary" report file. They may be ALTERed or DELETEd as desired, using the object names shown.

rpName

rpType

Additional members

Err

ERR

eb

ZEB

rpFreq=MONTH; rpZone=SUM;

Log

LOG

Inp

INP

Any reports specified by the user and not assigned to another file appear in the Primary report file between the default reports "eb" and "Log", in the order in which the REPORT objects are given in the input file.

Because of the many types of reports supported, the members required for each REPORT depend on the report type and frequency in a complex manner. When in doubt, testing is helpful: try your proposed REPORT specification; if it is incomplete or overspecified, CSE will issue specific error messages telling you what additional members are required or what inappropriate members have been given and why.

rpName

Name of report. Give after the word REPORT

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

rpReportfile=*rfname*

Name of report file to which current report will be written. If omitted, if REPORT is within a REPORTFILE object, report will be written to that report file, or else to REPORTFILE "Primary", which (as described in previous section) is automatically supplied and by default uses the file name of the input file with the extension .REP.

Units	Legal Range	Default	Required	Variability
	name of a *REPORTFILE*	current *REPORTFILE*, if any, else "Primary"	No	constant

rpType=*choice*

Choice indicating report type. Report types may be described at greater length, with examples, in Section 6.

ERR

Error and warning messages. If there are any such messages, they are also displayed on the screen AND written to a file with the same name as the input file and extension .ERR. Furthermore, * *many error messages are repeated in the INP report.

LOG

Run "log". As of July 1992, contains only CSE version number; should be enhanced or deleted.??

INP

Input echo: shows the portion of the input file used to specify this run. Does not repeat descriptions of objects left from prior runs in the same session when CLEAR is not used.

Error and warning messages relating to specific lines of the input are repeated after or near the line to which they relate, prefixed with "?". Lines not used due to a preprocessor #if command (Section 4.4.4) with a false expression are prefixed with a "0" in the leftmost column; all preprocessor command lines are prefixed with a "#" in that column.

SUM

Run summary. As of July 1992, NOT IMPLEMENTED: generates no output and no error message. Should be defined and implemented, or else deleted??.

ZDD

Zone data dump. Detailed dump of internal simulation values, useful for verifying that your input is as desired. Should be made less cryptic (July 1992)??. Requires rpZone.

ZST

Zone statistics. Requires rpZone.

ZEB

Zone energy balance. Requires rpZone.

MTR

Meter report. Requires rpMeter.

AH

Air handler report. Requires rpAh.

AHSIZE

Air handler autosizing report. Requires rpAh.

AHLOAD

Air handler load report. Requires rpAh. TODO

TUSIZE

Terminal autosizing report. Requires rpTu.

TULOAD

Terminal load. Requires rpTu. TODO

UDT

User-defined table. Data items are specified with REPORTCOL commands (next section). Allows creating almost any desired report by using CSE expressions to specify numeric or string values to tabulate; "Probes" may be used in the expressions to access CSE internal data.

Units	Legal Range	Default	Required	Variability
	see above		Yes	constant

The next three members specify how frequently values are reported and the start and end dates for the REPORT They are not allowed with *rpTypes* ERR, LOG, INP, SUM, and ZDD, which involve no time-varying data.

rpFreq=*choice*

Report Frequency: specifies interval for generating rows of report data:

YEAR

at run completion

MONTH

at end of each month (and at run completion if mid-month)

DAY

at end of each day

HOUR

at end of each hour

HOURANDSUB

at end of each subhour AND at end of hour

SUBHOUR

at end of each subhour

RpFreq values of HOURANDSUB and SUBHOUR are not supported with *rpType*'s of MTR and ZST, nor if *rpType* is ZEB or AH and *rpZone* or *rpAh* is SUM.

We recommend using HOURly and more frequent reports sparingly, to report on only a few typical or extreme days, or to explore a problem once it is known what day(s) it occurs on. Specifying such reports for a full-year run will generate a huge amount of output and cause extremely slow CSE execution.

Unit s

Legal Range

Defau lt

Required

Variabi lity

YEAR, MONTH, DAY, HOUR, HOURANDSUB, SUBHOUR

Required for rpTypes ZEB, ZST, MTR, AH, and UDT

constant

rpDayBeg=*date*

Initial day of period to be reported. Reports for which *rpFreq* = YEAR do not allow specification of *rpDayBeg* and *rpDayEnd*; for MONTH reports, these members default to include all months in the run; for DAY and shorter-interval reports, *rpDayBeg* is required and *rpDayEnd* defaults to *rpDayBeg*.

**Un its

**Lega l Range

Default

Required

Vari abilit y

date

first day of simulation if rpFreq = MONTH

Required for rpTypes ZEB, ZST, MTR, AH, and UDTif rpFreq is DAY, HOUR, HOURANDSUB, or SUB-HOUR

consta nt

rpDayEnd=*date*

Final day of period to be reported, except for YEAR reports.

Units	Legal Range	Default	Required	Variability
	date	last day of simulation if *rpFreq*= MONTH, else *rpDayBeg*	No	constant

rpZone=*znName*

Name of ZONE for which a ZEB, ZST, or ZDD report is being requested. For *rpType* ZEB or ZST, you may use *rpZone*=SUM to obtain a report showing only the sum of the data for all zones, or *rpZone*=ALL to obtain a report showing, for each time interval, a row of data for each zone plus a sum-of-zones row.

Units	Legal Range	Default	Required	Variability
	name of a *ZONE*, ALL, SUM		Required for *rpTypes* ZDD, ZEB, and ZST.	constant

rpMeter=*mtrName*

Specifies meter(s) to be reported, for *rpType*=MTR.

Units	Legal Range	Default	Required	Variability
	name of a *METER*, ALL, SUM		Required for *rpType*=MTR	constant

rpAh=ah*Name*

Specifies air handler(s) to be reported, for *rpType*=AH, AHSIZE, or AHLOAD.

Unit s

Legal Range

Defaul t

Required

Variabil ity

name of an AIRHANDLER, ALL, SUM

Required for rpType=AH, AHSIZE, or AHSIZE

constant

rpTu=tu*Name*

Specifies air handler(s) to be reported, for *rpType*=TUSIZE or TULOAD. TODO

Units	Legal Range	Default	Required	Variability
	name of a TERMINAL, ALL, SUM		Required for *rpType*=?	constant

rpBtuSf=*float*

Scale factor to be used when reporting energy values. Internally, all energy values are represented in Btu. This member allows scaling to more convenient units for output. *rpBtuSf* is not shown in the output, so if you change it, be sure the readers of the report know the energy units being used. *rpBtuSf* is not applied in UDT reports, but column values can be scaled as needed with expressions.

Units	Legal Range	Default	Required	Variability
	any multiple of ten	1,000,000: energy reported in MBtu.	No	constant

rpCond=*expression*

Conditional reporting flag. If given, report rows are printed only when value of expression is non-0. Permits selective reporting according to any condition that can be expressed as a CSE expression. Such conditional reporting can be used to shorten output and make it easy to find data of interest when you are only interested in the information under exceptional conditions, such as excessive zone temperature. Allowed with *rpTypes* ZEB, ZST, MTR, AH, and UDT.

Units	Legal Range	Default	Required	Variability
	any numeric expression	1 (reporting enabled)	No	subhour /end of interval

rpCPL=*int*

Characters per line for a User-Defined report. If widths specified in REPORTCOL (next section) add up to more than this, an error occurs; if they total substantially less, additional whitespace is inserted between columns to make the report more readable. Not allowed if *rpType* is not UDT.

Units	Legal Range	Default	Required	Variability
	$78 \leq x \leq 132$	top level *repCPL*	No	constant

rpTitle=*string*

Title for use in report header of User-Defined report. Disallowed if *rpType* is not UDT.

Units	Legal Range	Default	Required	Variability
		"User-defined Report"	No	constant

rpHeader=*choice*

Use NO to suppress the report header which gives the report type, zone, meter, or air handler being reported, time interval, column headings, etc. One reason to do this might be if you are putting only a single report in a report file and intend to later embed the report in a document or process it with some other program (but for the latter, see also EXPORT below).

Use with caution, as the header contains much of the identification of the data. For example, in an hourly report, only the hour of the day is shown in each data row; the day and month are shown in the header, which is repeated for each 24 data rows.

See REPORTFILE member *rfPageFmt*, above, to control report *FILE* page headers and footers, as opposed to *REPORT* headers and footers.

Units	Legal Range	Default	Required	Variability
	YES, NO	YES	No	constant

rpFooter=*choice*

Use NO to suppress the report footers. The report footer is usually a row which sums hourly data for the day, daily data for the month, or monthly data for the year. For a report with *rpZone, rpMeter,*or *rpAh* = ALL, the footer row shows sums for all zones, meters, or air handlers. Sometimes the footer is merely a blank line.

Units	Legal Range	Default	Required	Variability
	YES, NO	YES	No	constant

endReport

Optionally indicates the end of the report definition. Alternatively, the end of the report definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.26 REPORTCOL

Each REPORTCOL defines a single column of a User Defined Table (UDT) report. REPORTCOL are not used with report types other than UDT.

Use as many REPORTCOL as there are values to be shown in each row of the user-defined report. The values will appear in columns, ordered from left to right in the order defined. Be sure to include any necessary values to identify the row, such as the day of month, hour of day, etc. CSE supplies *NO* columns automatically.

colName

Name of REPORTCOL

Units	Legal Range	Default	Required	Variability

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

colReport=*rpName*

Name of report to which current report column belongs. If REPORTCOL is given within a REPORT object, then *colReport* defaults to that report.

Units	Legal Range	Default	Required	Variability
	name of a *REPORT*	*current report, if any*	Unless in a *REPORT*	constant

colVal=*expression*

Value to show in this column of report.

Units	Legal Range	Default	Required	Variability
	any numeric or string expression		Yes	subhour /end interval

colHead=*string*

Text used for column head.

Units	Legal Range	Default	Required	Variability
		colName or blank	No	constant

colGap=*int*

Space between (to left of) column, in character positions. Allows you to space columns unequally, to emphasize relations among columns or to improve readability. If the total of the *colGaps* and *colWids* in the report's REPORTCOL is substantially less than the REPORT *rpCPL* (characters per line, previous section), CSE will insert additional spaces between columns. To suppress these spaces, use a smaller *rpCPL*.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	1	No	constant

colWid=*int*

Column width.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	10	No	constant

colDec=*int*

Number of digits after decimal point.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	*flexible format*	No	constant

colJust=*choice*

Specifies positioning of data within column:

Left	Left justified
Right	Right justified

endReportCol

Optionally indicates the end of the report column definition. Alternatively, the end of the report column definition can be indicated by END or by beginning another REPORTCOL or other object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.27 EXPORTFILE

EXPORTFILE allows optional specification of different or additional files to receive CSE EXPORT

EXPORT contain the same information as reports, but formatted for reading by other programs rather than by people. By default, CSE generates no exports. Exports are specified via the EXPORT object, described in Section 5.28 (next). As for REPORT CSE automatically supplies a primary export file; it has the same name and path as the input file, and extension .csv.

Input for EXPORTFILE and EXPORT is similar to that for REPORTFILE and REPORT except that there is no page formatting. Refer to their preceding descriptions (Sections 5.24 and 5.25) for more additional discussion.

xfName

Name of EXPORTFILE object.

Units	Legal Range	Default	Required	Variability
	63 characters		No	constant

xfFileName=*string*

path name of file to be written. If no path is specified, the file is written in the current directory. If no extension is specified, .csv is used.

Units	Legal Range	Default	Required	Variability
	file name, path and extension optional		No	constant

xfFileStat=*choice*

What CSE should do if file *xfFileName* already exists:

OVERWRITE

Overwrite pre-existing file.

NEW

Issue error message if file exists.

APPEND

Append new output to present contents of existing file.

If the specified file does not exist, it is created and *xfFileStat* has no effect.

Units	Legal Range	Default	Required	Variability
	OVERWRITE, NEW, APPEND	OVERWRITE	No	constant

endExportFile

Optionally indicates the end of the export file definition. Alternatively, the end of the Export file definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.28 EXPORT

Exports contain the same information as CSE reports, but in a "comma-quote" format intended for reading into a spreadsheet or other program for further processing, plotting, special print formatting, etc.

No exports are generated by default; each desired export must be specified with an EXPORT object.

Each row of an export contains several values, separated by commas, with quotes around string values. The row is terminated with a carriage return/line feed character pair. The first fields of the row identify the data. Multiple fields are used as necessary to identify the data. For example, the rows of an hourly ZEB export begin with the month, day of month, and hour of day. In contrast, reports, being subject to a width limitation, use only a single column of each row to identify the data; additional identification is put in the header. For example, an hourly ZEB Report shows the hour in a column and the day and month in the header; the header is repeated at the start of each day. The header of an export is never repeated.

Depending on your application, if you specify multiple exports, you may need to place each in a separate file. Generate these files with EXPORTFILE preceding section. You may also need to suppress the export header and/or footer, with *exHeader* and/or *exFooter*, described in this section.

Input for EXPORT is similar to input for REPORT refer to the REPORT description in Section 5.25 for further discussion of the members shown here.

exName

Name of export. Give after the word EXPORT

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

exExportfile=x*fname*

Name of export file to which current export will be written. If omitted, if EXPORT is within an EXPORTFILE object, report will be written to that export file, or else to the automatically-supplied EXPORTFILE "Primary", which by default uses the name of the input file with the extension .csv.

Units

Legal Range

Default

Require d

Variabil ity

name of an EXPORTFILE

current EXPORTFILE, if any, else "Primary"

No

constant

exType=*choice*

Choice indicating export type. See descriptions in Section 5.22, REPORT While not actually disallowed, use of *exType* = ERR, LOG, INP, or ZDD is unexpected.

Units	Legal Range	Default	Required	Variability
	ZEB, ZST, MTR, AH, UDT, or SUM		Yes	constant

exFreq=*choice*

Export Frequency: specifies interval for generating rows of export data:

Units	Legal Range	Default	Required	Variability
	YEAR, MONTH, DAY, HOUR, HOURANDSUB, SUBHOUR		Yes	constant

exDayBeg=*date*

Initial day of export. Exports for which *exFreq* = YEAR do not allow specification of *exDayBeg* and *exDayEnd*; for MONTH exports, these members are optional and default to include the entire run; for DAY and shorter-interval exports, *exDayBeg* is required and *exDayEnd* defaults to *exDayBeg*.

**Un its

**Lega l Range

Default

Required

Vari abilit y

date

first day of simulation if exFreq = MONTH

Required for exTypes ZEB, ZST, MTR, AH, and UDTif exFreq is DAY, HOUR, HOURANDSUB, or SUB-HOUR

consta nt

exDayEnd=*date*

Final day of export period, except for YEAR exports.

Units	Legal Range	Default	Required	Variability
	date	last day of simulation if *exFreq*= MONTH, else *exDayBeg*	No	constant

exZone=*znName*

Name of ZONE for which a ZEB, ZST, or ZDD export is being requested; ALL and SUM are also allowed except with *exType* = ZST.

Units	Legal Range	Default	Required	Variability
	name of a *ZONE*, ALL, SUM		Required for *exTypes* ZDD, ZEB, and ZST.	constant

exMeter=*mtrName*

Specifies meter(s) whose data is to be exported, for *exType*=MTR.

Units	Legal Range	Default	Required	Variability
	name of a *METER*, ALL, SUM		Required for *exType*=MTR	constant

exAh=ah*Name*

Specifies air handler(s) to be exported, for *exType*=AH.

Units	Legal Range	Default	Required	Variability
	name of an *AIRHANDLER*, ALL, SUM		Required for *exType*=AH	constant

exBtuSf=*float*

Scale factor used for exported energy values.

Units	Legal Range	Default	Required	Variability
	any multiple of ten	1,000,000: energy exported in MBtu.	No	constant

exCond=*expression*

Conditional exporting flag. If given, export rows are generated only when value of expression is non-0. Allowed with *exTypes* ZEB, ZST, MTR, AH, and UDT.

Units	Legal Range	Default	Required	Variability
	any numeric expression	1 (exporting enabled)	No	subhour /end of interval

exTitle=*string*

Title for use in export header of User-Defined export. Disallowed if *exType* is not UDT.

Units	Legal Range	Default	Required	Variability
		"User-defined Export"	No	constant

exHeader=*choice*

Use NO to suppress the export header which gives the export type, zone, meter, or air handler being exported, time interval, column headings, etc. You might do this if the export is to be subsequently imported to a program that is confused by the header information.

If not suppressed, the export header shows, in four lines:

runTitle and *runSerial* (see Section 5.1);the run date and timethe export type ("Energy Balance", "Statistics", etc., or *exTitle* if given)and frequency ("year", "day", etc.)a list of field names in the order they will be shown in the data rows("Mon", "Day", "Tair", etc.)

The *specific* month, day, etc. is NOT shown in the export header (as it is shown in the report header), because it is shown in each export row.

The field names may be used by a program reading the export to identify the data in the rows which follow; if the program does this, it will not require modification when fields are added to or rearranged in the export in a future version of CSE.

Units	Legal Range	Default	Required	Variability
	YES, NO	YES	No	constant

exFooter=*choice*

Use NO to suppress the blank line otherwise output as an export "footer". (Exports do not receive the total lines that most reports receive as footers.)

Units	Legal Range	Default	Required	Variability
	YES, NO	YES	No	constant

endExport

Optionally indicates the end of the export definition. Alternatively, the end of the export definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

5.29 EXPORTCOL

Each EXPORTCOL defines a single datum of a User Defined Table (UDT) export; EXPORTCOL are not used with other export types.

Use as many EXPORTCOL as there are values to be shown in each row of the user-defined export. The values will appear in the order defined in each data row output. Be sure to include values needed to identify the data, such as the month, day, and hour, as appropriate – these are NOT automatically supplied in user-defined exports.

EXPORTCOL members are similar to the corresponding REPORTCOL members. See Section 5.265.1.5 for further discussion.

colName

Name of EXPORTCOL

Units	Legal Range	Default	Required	Variability
	63 characters	*none*	No	constant

colExport=*exName*

Name of export to which this column belongs. If the EXPORTCOL is given within an EXPORT object, then *colExport* defaults to that export.

Units	Legal Range	Default	Required	Variability
	name of an *EXPORT*	*current export, if any*	Unless in an *EXPORT*	constant

colVal=*expression*

Value to show in this position in each row of export.

Units	Legal Range	Default	Required	Variability
	any numeric or string expression		Yes	subhour /end interval

colHead=*string*

Text used for field name in export header.

Units	Legal Range	Default	Required	Variability
		colName or blank	No	constant

colWid=*int*

Maximum width. Leading and trailing spaces and non-significant zeroes are removed from export data to save file space. Specifying a *colWid* less than the default may reduce the maximum number of significant digits output.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	13	No	constant

colDec=*int*

Number of digits after decimal point.

Units	Legal Range	Default	Required	Variability
	$x \geq 0$	*flexible format*	No	constant

endExportCol

Optionally indicates the end of the EXPORTCOL Alternatively, the end of the definition can be indicated by END or by beginning another object.

Units	Legal Range	Default	Required	Variability
		N/A	No	constant

6 Output Reports

CSE report data is accumulated during simulation and written to the report file at the end of the run. Some reports are generated by default and cannot be turned off. There are a set of predefined reports which may be requested in the input. The user may also define custom reports which include many CSE internal variables. Reports may accumulate data on an a variety of frequencies including subhourly, hourly, daily, monthly, and annual (run) intervals.

6.1 Units

The default units for CSE reports are:

Energy	mBtu, millions of Btu (to convert to kWh divide by 292)
Temperature	degrees Farenheit
Air Flow	cfm (cubic feet per minute)

6.2 Time

Hourly reports show hour 1 through 24 where hour 1 includes the time period from midnight to 1 AM. By default, CSE specifies that January first is a Thursday and the simulation occurs on a non-leap year. Daylight savings is in effect from the second Sunday of March on which CSE skips hour 3 until the first Sunday of November when CSE simulates 25 hours. These calendar defaults can be modified as required.

6.3 METER Reports

A Meter Report displays the energy use of a METER object, a user-defined "device" that records energy consumption of equipment as simulated by CSE. CSE allows the user to define as many meters as desired and to assign any energy using device to any meter.

Meters account for energy use in pre-defined categories, called *end uses, that are documented with METER*

6.4 Energy Balance Report

The Energy Balance Report displays the temperature and sensible and latent heat flows into and out of the air of a single zone. Sign conventions assume that a positive flow increases the air temperature. Heat flow from a warm mass element such as a concrete wall into the zone air is defined as a positive flow, heat flow from air into mass is negative. Solar gain into the zone is defined as a positive heat flow. Solar gain that is incident on and absorbed directly into a mass element is shown as both a positve in the SOLAR column (gain to the zone) and a negative in the MASS column (lost from the zone to the mass).

In a real building zone energy and moisture flows must balance due to the laws of physics. CSE uses approximate solutions for the energy and moisture balances and displays the net balance which is a measure of internal calculation error.

The following items are displayed (using the abbreviations shown in the report headings):

Tair

Air temperature in the zone (since CSE uses combined films this is technically the effective temperature and includes radiant effects).

WBair

Wet Bulb temperature in the zone.

Cond

Heat flow through light weight surfaces from or to the outdoors.

InfS

Sensible infiltration heat flow from outdoors.

Slr

Solar gain through glazing (net) and solar gains absorbed by light surfaces and transmitted into the zone air.

IgnS

Sensible internal gains from lights, equipment, people, etc.

Mass

Net heat flow to (negative) and from (positive) the mass elements of the zone.

Izone

Net heat flows to other zones in the building.

MechS

Net heat flows from heating, cooling and ventilation.

BALS

The balance (error) calculated by summing the sensible gains and losses.

InfL

Latent infiltration heat flow.

IgnL

Latent internal gains.

AirL

Latent heat absorbed (negative) or released (positive) by changes in the room air moisture content.

MechL

Latent heat added or removed by cooling or ventilation.

BalL

The balance (error) calculated by summing the sensible gains and losses.

6.5 Air Handler Load Report

The Air Handler Load Report displays conditions and loads at the peak load hours for the air handler for a single zone. The following items are displayed:

PkVf	Peak flow (cfm) at supply fan
VfDs	Supply fan design flow (same as peak for E10 systems)
PkQH	Peak heat output from heating coil.
Hcapt	Rated capacity of heat coil

The rest are about the cooling coil. Most of the columns are values at the time of peak part load ratio (plr). Note that, for example, the peak sensible load is the sensible load at the time of peak part load ratio, even if there was a higher sensible load at another time when the part load ratio was smaller.

PkMo

Month of cooling coil peak plr, 1-12

Dy

Day of month 1-31 of peak

Hr

Hour of day 1-24 of cooling coil peak plr.

Tout

Outdoor drybulb temperature at time of cooling coil peak plr.

Wbou

Outdoor wetbulb similarly

Ten

Cooling coil entering air temperature at time of peak plr.

Wben

Entering wetbulb similarly

Tex

Exiting air temperature at plr peak

WbexExiting air wetbulb similarly

-PkQs

Sensible load at time of peak plr, shown positive.

-PkQl

Latent load likewise

-PkQC

Total load – sum of PkQs and PkQl

CPlr

Peak part load ratio: highest fraction of coil's capacity used, reflecting both fraction of maximum output under current conditions used when on and fraction of the time the fan is on. The maximum output under actual conditions can vary considerably from the rated capacity for DX coils. The fraction of maximum output used can only be 1.0 if the sensible and total loads happen to occur in the same ratio as the sensible and total capacities. The time the fan is on can be less than 1.0 for residential systems in which the fan cycles on with the compressor. For example, if at the cooling peak the coil ran at .8 power with the fan on .9 of the time, a CPlr of .72 would be reported. The preceding 12 columns are values at the time this peak occured.

Ccapt

Cooling coil rated total capacity

Ccaps

Rated sensible capacity.

6.6 Air Handler Report

The Air Handler Load Report displays conditions and heat flows in the air handler for the time period specified. It is important to note that the air handler report only accumulates data if the air handler is on during an hour. The daily and monthly values are averages of the hours the air handler was on and DO NOT INCLUDE OFF HOUR VALUES. The following items are displayed:

Tout

Outdoor drybulb temperature during hours the air handler was on.

Wbou

Outdoor wetbulb temperature similarly.

Tret

Return air dry bulb temperature during hours the air handler was on before return duct losses or leaks.

Wbre

Return air wetbulb similarly

po

Fraction outside air including economizer damper leakage, but not return duct leakage.

Tmix

Mixed air dry bulb temperature – after return air combined with outside air; after return fan, but before supply fan and coil(s).

Wbmi

Mixed air wet bulb temperature, similarly.

Tsup

Supply air dry bulb temperature to zone terminals – after coil(s) and air handler supply duct leak and loss; (without in zone duct losses after terminals).

WBsu

Supply air wet bulb temperature similarly.

HrsOn

Hours during which the fan operated at least part of the time.

FOn

Fraction of the time the fan was on during the hours it operated (HrsOn). CHECK FOR VAV, IS IT FLOW OR TIME

VF

Volumetric flow, measured at mix point/supply fan/coils; includes air that leaks out of supply duct and is thus non-0 even when zone terminals are taking no flow

Qheat

Heat energy added to air stream by heat coil, if any, MEASURED AT COIL not as delivered to zones (see Qload).

Qsens, Qlat and Qcool

Sensible, latent, and total heat added to air stream (negative values) by cooling coil, MEASURED AT COIL, including heat cancelled by fan heat and duct losses, and heat added to air lost through supply duct leak.

Qout

Net heat taken from outdoor air. Sum of sensible and latent, measured RELATIVE TO CURRENT RETURN AIR CONDITIONS.

Qfan

Heat added to air stream by supply fan, plus return fan if any – but not relief fan..

Qloss

Heat added to air stream by supply and return duct leaks and conductive loss. Computed in each case as the sensible and latent heat in the air stream relative to return air conditions after the leak or loss, less the same value before the leak or loss.

Qload

Net energy delivered to the terminals – Sensible and latent energy, measured relative to return air conditions. INCLUDES DUCT LOSSES after terminals; thus will differ from sum of zone qMech's + qMecLat's.

Qbal

Sum of all the 'Q' columns, primarily a development aid. Zero indicates consistent and accurate computation; the normal printout is something like .0000, indicating that the value was too small to print in the space alloted, but not precisely zero, due to computational tolerances and internal round-off errors.

7 Index