Contents

1 Overview ............................................................................................................................................... 3

Input Files ............................................................................................................................................ 3
  Include Files ..................................................................................................................................... 3
  Scope ............................................................................................................................................... 4

General Definitions ............................................................................................................................. 4
  Identifiers ......................................................................................................................................... 4
  Strings .............................................................................................................................................. 4
  Literals ............................................................................................................................................. 4

2 Block Descriptions ........................................................................................................................... 5

Declare Blocks .................................................................................................................................. 5
  Variable Types ................................................................................................................................. 5
  Stream Types ................................................................................................................................. 5
  Enumeration Types ......................................................................................................................... 6

External Blocks ................................................................................................................................ 6

Model Blocks .................................................................................................................................... 6
  Model blocks are generic modeling entities ..................................................................................... 7
    Parameter ....................................................................................................................................... 8
    Unit ............................................................................................................................................... 8
    Variable ....................................................................................................................................... 9
    Stream ....................................................................................................................................... 9
    Selector ..................................................................................................................................... 10
    Set .............................................................................................................................................. 10
    Intermediate ............................................................................................................................... 12
    Equation ..................................................................................................................................... 13

Simulation Blocks ............................................................................................................................ 17
  Options .......................................................................................................................................... 18
    Parameter ..................................................................................................................................... 19
    Unit ............................................................................................................................................... 19
    Report ......................................................................................................................................... 20
    Sensitivity ................................................................................................................................. 20
    Ask ............................................................................................................................................. 21
    Sel ............................................................................................................................................. 21
    Intermediate ............................................................................................................................... 21
    Equation ..................................................................................................................................... 21
    Input ........................................................................................................................................... 21
    Preset ....................................................................................................................................... 22
    Selector ..................................................................................................................................... 22
    Initial .......................................................................................................................................... 22
    Schedule ................................................................................................................................... 22

Estimation Blocks ............................................................................................................................. 26
  Uncertain ....................................................................................................................................... 26
  Objective ....................................................................................................................................... 27
  Experiment .................................................................................................................................... 27
1 Overview

The JACOBIAN Dynamic Simulation and Optimization Software implements a high level, declarative language with which the user may describe complex models in an intuitive and natural manner. JACOBIAN supports a wide variety of high level concepts, such as inheritance and polymorphism, which help manage the complexity of large models and make a model easier to maintain and evolve. An important feature of the JACOBIAN modeling language is that model development is completely decoupled from model solution. For example, the model of a system may be written once and a variety of calculation may be applied to this same model, without any modifications. This document describes the syntax and structure of the JACOBIAN modeling language.

Input Files

The JACOBIAN input file is a plain ASCII text file. The input file name must end with the .JAC extension. A problem may be distributed over one or more input files and each input file may consist of one or more blocks of the following types:

Declare blocks
External blocks
Model blocks
Simulation blocks
Estimation blocks

These blocks are described later in detail.

The order of the block is arbitrary, provided an element is defined before it is referenced (this will be explained in more detail later). Comments are preceded by a pound sign (#) and continue until the end of the line. The JACOBIAN syntax is not case sensitive.

Include Files

An input file may include one or more other input files using the INCLUDE statement. The INCLUDE command has the syntax:

INCLUDE filename

This statement may appear anywhere in the input file. The effect of this keyword is to simply incorporate verbatim the contents of the specified file into the file containing the INCLUDE statement. JACOBIAN will search for the included file relative to the include directory (which may be defined by the user). JACOBIAN will first search for the file name without an extension. If the file is not found the search is repeated using the file name extension .JAC. If the file is still not found, a final attempt is made using the extension .PRESETS (see the SAVE PRESETS syntax). JACOBIAN places no limit on the depth included files are nested (e.g., an included file may include another file and so on).

1 An exception of this is when an external files is referenced (e.g., see INCLUDE statement) and the current operating system uses case sensitive files names, such as Linux/UNIX.
Fraction = 0.5 : 0 : 1 UNIT="NONE"
Temperature = 300 : 100 : 1000 UNIT="K"
END
INCLUDE MyModels
INCLUDE MySimulations

In this example, MyModels.JAC contains a collection of models describing the problem (and these models require the variables types Fraction and Temperature) and MySimulations.JAC contains one or more simulation blocks.

Scope
Entities defined in a JACOBIAN input file have an associated scope of visibility. Understanding the scoping rules is necessary to avoid potential name conflicts. The scope of the various JACOBIAN entities is explained in this document.

General Definitions

Identifiers
Identifiers in a JACOBIAN input file (used for naming various entities such as variables, models, etc) are a sequence of letters (a-z and A-Z), digits (0-9), or underscore characters (‘_’). The identifier must begin with a letter. There is no restriction on its length. A list of reserved keywords is shown below.

Reserved Keyword:

AND ALL ARRAY AS ASK CASE CONNECTION CONTINUATION CONTINUE DATA DECLARE DEFAULT DIMENSION DISPLAY DO ELEMENT ELSE END ENUMERATION EOR EQUATION ESTIMATION EXPERIMENT EXPORT EXTERNAL FALSE FINAL_TIME FOR FORTRAN IF IMPORT IN INCLUDE INDEX_SET INHERITS INITIAL INPUT INTERMEDIATE IS MAXIMIZE MINIMIZE MODEL NOT OBJECTIVE OF OPTIONS OR PARALLEL PARAMETER PRESET PRESETS REINITIAL REPORT RESET RETURNS SAVE SCHEDULE SELECTOR SENSITIVITY SEQUENCE SET SIMULATION STATUS STEADY_STATE STEP STREAM SWITCH TASK THEN TIME TIME_INVARIANT TO TRUE TYPE UNIT UNCERTAIN UNTIL USING VARIABLE WHEN WHILE WITH WITHIN WRITE

Strings
Character strings are used in a variety of places in a JACOBIAN input file. The character string is an arbitrary sequences of characters enclosed in double quotes (”). A string cannot contain another double quote or a newline character.

Literals
Literal are defined in JACOBIAN as one might expect (e.g., 1.4, 300, 3e-4). Scientific notation is denoted with an ‘e’ or ‘E’.
2 Block Descriptions

Declare Blocks

Declare blocks begin with the keyword DECLARE and are terminated with the keyword END. Within the declare block the user defines the variable, stream, and enumeration types that will be used in the problem description. The declare block consists of the following sections:

Type
Stream
Enumeration

Not all of the sections are required. A JACOBIAN input file may contain one or more declare blocks. The variable, stream, and enumeration types have global scope within a JACOBIAN problem. That is, once a type is declared its name is visible everywhere within the JACOBIAN problem. Consequently, all variable types must be given unique names. Similarly, all stream and enumeration types must be given unique names. Although it is possible to use the same name for a variable type and a stream or enumeration type, this is often a confusing practice.

Variable Types

JACOBIAN requires that each variable declared in a problem must have a pre-defined variable type. Associated with each variable type are a default value, lower bound, upper bound, and (optional) units of measurement. Within the declare block, variable types are defined under a type section (denoted by the keyword TYPE). The general syntax for the variable type definition is

Identifier = default-value : lower-bound : upper-bound [UNIT=string]

The square brackets in the description above indicate the item is optional. An example type section in a declare block is

DECLARE
  TYPE
    Fraction  = 0.5 : 0.0 : 1.0
    Concentration = 0.001 : 0.0 : 10  UNIT="mol/L"
    Temperature  = 200 : 100 : 1000 UNIT="Kelvin"
    Pressure  = 1.0 : 1e-2 : 10  UNIT="bar"
END

To reiterate, all variables in a JACOBIAN problem must be declared with a pre-defined variable type. This type is used to assign a default initial value and bounds to the variable. Currently, the unit of measurement is for display purposes and internal documentation only. The default value and bounds of specific variables may be refined later in the PRESET section of a computation.

Stream Types

The STREAM construct is a convenient way to group several related variables. The stream type can be thought of as a template. Specific variables are associated with the stream type elsewhere in the JACOBIAN input file. Streams are declared in order to simplify the connection of multiple UNITS in a JACOBIAN problem. The use of streams is described in more detail below. Within a declare block,
stream types are defined under the stream section (denoted by the keyword **STREAM**). The general syntax for a stream type is

**Identifier IS variable-type-list**

where variable-type-list is a comma separated list of pre-defined variable types. An example stream section in a declare block is

```
DECLARE
  TYPE
  Fraction      = 0.5   : 0.0  : 1.0
  Concentration = 0.001 : 0.0  : 10   UNIT="mol/L"
  Temperature   = 200   : 100  : 1000 UNIT="Kelvin"
  Pressure      = 1.0   : 1e-2 : 10   UNIT="bar"
END
```

**STREAM**

  ProcessStream IS Temperature, Pressure, Concentration

**Enumeration Types**

An enumeration declaration simply associates a list of named constants with a single type. The general syntax is

**Identifier IS { identifier-list }**

where Identifier is the name of the enumeration types and identifier-list is the list of values that may be assigned to symbols of the given type. Within a declare block, enumeration declarations appear under the enumeration section (under the keyword **ENUMERATION**). An example is

```
ENUMERATION
  Components IS { H2, O2, N2, CH4 }
  ValvePosition IS { OPEN, CLOSED }
END
```

The use of enumeration is elaborated upon below (in particular, see the **SELECTOR** description).

**External Blocks**

JACOBIAN allows the user to introduce equations into a model through external procedures. For example, C or FORTRAN subroutines from a physical property library can be used to compute activity coefficients or vapor-liquid equilibrium. A detailed discussion of external procedures is outside the scope of this syntax manual. Refer to JACOBIAN Advanced User Guide on External for details.

**Model Blocks**

The system of equations, and associated parameters and variables describing a problem of interest in JACOBIAN is defined in a collection of one or more model blocks. The model block may be used to logically decompose the system of equations into related subsets (in contrast to putting all equations in a single model). The model block begins with the keyword **MODEL** followed by the models name and ends with the keyword **END**. Models have *global scope* and consequently *every model must have a unique name*. The model block consists of the following sections:

**Parameter**

**Unit**

**Variable**
Not all of the sections are required. However, those present must be in the order above. All entities declared in a model block have *local scope*, that is, they are only visible in the model where they are defined.

**Model blocks are generic modeling entities**

Before a problem is solved, the user must instantiate one or more models as specific *units*. This is described in more detail in a later section.

A model writer is encouraged to write the models as general as possible (e.g., parameterized by named parameters to be given values later and with additional degrees of freedom). This allows the model to be re-used in a variety of applications. To further support re-use and aide maintainability, JACOBIAN supports the object-oriented concept of *inheritance*. A model that inherits another model will automatically get all of the attributes of the parent model.

An example illustrating the inheritance feature is shown below.

```
DECLARE
  TYPE
    NoType = 1.0 : 0.0 : 1000
END

MODEL BaseModel
  PARAMETER
    P1, P2, P3 AS REAL
  VARIABLE
    X1, X2 AS NoType
  EQUATION
    $X1 = -P1*X1 + P3*X2 ;
    $X2 = -P2*X1*X2 ;
END

MODEL DerivedModel INHERITS BaseModel
  VARIABLE
    X3 AS NoType
  EQUATION
    $X3 = (P1+P3)*X1-P2*X2 ;
END
```

In this example, model *BaseModel* defines three parameters, two variables, and two equations. Model *DerivedModel* inherits the parameters, variables, and equations of *BaseModel* and adds an additional variable and equation of its own. *The inheritance mechanism facilitates the creation of a library of generic models that may be refined and extended to solve specific problems.*
Parameter

Time invariant parameters in a model are declared in the parameter section. The parameter section begins with the keyword `PARAMETER` and continues until the beginning of the next section (i.e., this section is not terminated with `END`). The general syntax of the parameter declaration is

```
Identifier-list AS [ ARRAY(size-list) OF ] parameter-type
   [ DEFAULT default-expression ]
```

where `Identifier-list` is a comma separated list of the names (i.e., identifiers) of one or more parameters begin declared, `ARRAY` is an optional array declaration and `size-list` is a comma separated list of array dimensions, `parameter-type` is the predefined parameter type, and `default-expression` is an expression containing an optional value for the list of parameters. The default value may be any valid JACOBIAN expression in terms of literal constants and pre-declared parameters. The supported parameter types are: `REAL`, `INTEGER`, `LOGICAL`, and `STRING`. The parameter may be an array of arbitrary dimension and the size of each dimension may be specified by an integer expression in terms of literals and previously defined integer parameters. JACOBIAN supports several methods for specifying the value of a parameter. Parameters may be explicitly assigned values in the `SET` section of a model or computation, or `ASK` section of a computation. Parameters may also be set by propagation. This is described in more detail later. Parameters not assigned explicitly or through propagation will be assigned their default values. If no default value is available, JACOBIAN will issue an error message indicating a parameter could not be assigned a value and the problem will not be solved. An example parameter section is shown below.

```
PARAMETER
   Pi AS REAL DEFAULT 4*ATAN(1)       # Constant expressions are allowed
   G AS REAL
   Open AS LOGICAL DEFAULT TRUE        # Logical constants are TRUE or FALSE
   NC AS INTEGER
   M AS INTEGER DEFAULT NC*NC+4*NC  # Default expression may involve
      # other parameters
   k AS ARRAY(NC,M+1) OF REAL DEFAULT 1.0 # All values of the array
      # are set to 1 by default
```

Unit

As described above, model blocks represent generic modeling entities. The actual system of equations to solve is generated by instantiating one or more model blocks as specific units. Readers familiar with C or C++ programming can think of the model blocks as a `struct` definition and the units as variables with a `struct` type. The unit section within a model allows one model to incorporate the parameters, variables, and equations of another, previously defined, model. The general syntax of a unit declaration is

```
Identifier-list AS [ ARRAY(size-list) OF ] model-type
```

where `Identifier-list` is the list of unit names and `model-type` is the name of a previously declared model block. Like variables and parameters, arrays of units may be instantiated. An example unit section is

```
MODEL Process
PARAMETER
   N AS INTEGER
UNIT
   Valve AS ValveModel
```
Vessels AS ARRAY(N) OF TankModels

In this example, \( N+1 \) units are declared. All of the parameters, variables, and equations of these sub-units are part of the model Process. A pathname, described below, must be provided to access the attributes of sub-unit.

**Pathnames**

The parameters and variables contained in the subunits of a model have *local scope* (e.g., two separate models may have parameters with the same name and there will not be a name conflict). In order to access these parameters and variables, the *scope resolution operator* `.` must be used. For example, suppose in the example above, the model Valve has a variable named Stem_position. The Stem_position variable in Valve may be accessed inside Process by writing Valve.Stem_position. This prefix of a variable, parameter, or unit is referred to as the *pathname* of the symbol. The pathname may be arbitrarily deep. For example, if we instantiate the Process model as a unit named Proc in another model, then we can access the variable Stem_position as Proc.Valve.Stem_position.

**Variable**

Variables are declared and assigned *variable type* in model’s variable section. A variable section begins with the keyword *VARIABLE* and is followed by one or more variable declaration statements. The general syntax of the variable declaration is

\[
\text{Identifier-list AS [ ARRAY(size-list) ] OF variable-type}
\]

where *Identifier-list* is a list of the names of the variables being declared, *ARRAY* is an optional array declaration (with array dimensions given by *size-list*), and *variable-type* is a previously defined variable type. Like parameters and units, variables may be arrays of arbitrary dimension, and the dimension sizes may be expressed as integer expressions containing literals and parameters. An example variable section is

VARIABLE

Conc AS ARRAY(Nspec) OF Concentration
InletTemp AS Temperature
Data AS ARRAY(Ndata+2,Nspec) OF NoType

JACOBIAN defines a global symbol called “TIME” that is used to reference the independent variable in a dynamic system. This is a reserved keyword. Time derivatives of variables can be referenced in equations by using the `$` operator (e.g., $\text{Conc}(2)$ refers to the time derivative of the concentration of species 2).

**Stream**

As described above, JACOBIAN supports the concept of *stream types*. The stream type declaration associates a collection of variable types with a single identifier. In the stream section of a model, a collection of variables is associated with a single stream. For example,

VARIABLE

Conc AS ARRAY(N) OF Concentration
Temp AS Temperature
Pres AS Pressure

STREAM

InletStream : Temp, Pres, Conc AS ProcessStream
In this example, three variables (two scalars and one array) are associated with the single stream InletStream (the stream type ProcessStream has been defined in a previous example above). Notice that the variable types of the variables in the declaration match the variable types listed in the stream type declaration. Streams in one model may also be declared using the stream of a subunit. The general syntax of this is

Identifier IS Subunit-stream-name

For example,

MODEL Process
UNIT
  Valve AS ValveModel
  Vessels AS ARRAY(N) OF TankModels
STREAM
  ValveInletStream IS Valve.Inlet

This example assumes there is a stream declared in the model ValveModel. The stream concept provides a convenient, shorthand mechanism for equating several variables in one unit to several variables in another unit via a single statement. This is described more detail in the stream equation section below.

Selector

A selector is a special discrete variable used in the CASE equation. The general syntax for a selector declaration is

Identifier-list AS [ ARRAY(size-list) OF ] { identifier-list }
  [ DEFAULT identifier ]

Identifier-list AS [ ARRAY(size-list) OF ] ELEMENT OF enumeration-type
  [ DEFAULT identifier ]

where Identifier-list is the list of names of selectors being declared. The selector type may be implicitly defined within the declaration (first example) or by using a pre-defined enumeration type (second example). In either case, an optional default value may be provided. The only values that may be assigned to selector variables are the enumeration type elements. For example, suppose we define the following selector:

SELECTOR
  DiscreteValvePosition AS ELEMENT OF ValvePosition DEFAULT Open

where ValvePosition is defined above. The only values that may be assigned to DiscreteValvePosition are OPEN and CLOSED. The value of a selector may only be modified in the following contexts: 1) within computation SELECTOR sections, 2) within RESET tasks, and 3) within CASE equations.

Set

Time invariant parameters in a model may be specified in the (optional) set section. The parameter assignments appear under the keyword SET. An example of a set section is

MODEL Example
PARAMETER
  N AS INTEGER
  A AS ARRAY(N) OF REAL
UNIT
  U AS SomeOtherModel
SET
  N := 4 ;
  A := [ 0.2, 0.4, 1.6, 3.2 ] ; # array expression - dimensions
  # must be consistent
  U.P1 := 44 ;
WITHIN U DO                   # this is a WITHIN statement...
  P2 := 21 ;                  # ...notice that the pathnames
  P3 := 22 ;                  # are not necessary in here
  P4 := 23 ;
END

The assignments in the example above include assigning values to scalars and arrays. Also, the
WITHIN END statement is shown here. The WITHIN statement provides a pathname (in this case U) which is applied to all of the symbols inside the WITHIN body. The WITHIN body may contain an
arbitrary number of assignments, including other WITHIN statements, e.g., the statement
WITHIN U1 DO
  WITHIN U2 DO
    WITHIN U3 DO
      N := 4 ;
    END
  END
END

assigns a value to the parameter U1.U2.U3.N.

Propagation

Notice that in the example above we are setting the values of local parameters and parameters
contained in the sub-unit U. The assignments of the parameters in U will override assignments of the
same parameters in model SomeOtherModel (the model type of U). The process of indirectly
assigning values of parameters is referred to as propagation. An example of this often used feature is

MODEL A
PARAMETER
  N AS INTEGER
END
MODEL B
PARAMETER
  N AS INTEGER
UNIT
  UA AS A
END
MODEL C
PARAMETER
  N AS INTEGER
UNIT
  UB AS B
SET
  N := 4 ;
END
The single assignment to parameter $N$ in model $C$ will automatically assign the same value to the corresponding parameters in units $UB$ and $UB.UA$.

Intermediate

JACOBIAN supports the ability to define intermediate variables in a model. Often when writing models additional constraints are introduced to make the model clearer and easier to maintain. For example, flux definitions in a reaction network are often written this way. Also, complex physical property correlations tend to introduce many intermediate variables. The introduction of these variables is convenient, but increases the size of the problem. In JACOBIAN, the user may introduce temporary variables in an `INTERMEDIATE` section in a model or computation. An example is shown below:

MODEL Reactor
PARAMETER
  $k_1$ AS REAL DEFAULT 0.1
  $k_2$ AS REAL DEFAULT 0.15
  $K_{eq}$ AS REAL DEFAULT 2.5
  Volume AS REAL DEFAULT 1.0
VARIABLE
  $T$ AS Temperature
  $X_0$ AS Concentration
  $X_1$ AS Concentration
INTERMEDIATE
  $S_1 := T/(1 + K_{eq})$ ;
  $S_2 := K_{eq} * S_1$ ;
EQUATION
  Volume*$X_0 = -k_1*X_0$ ;
  Volume*$X_1 = k_2*S_2$ ;
  Volume*$T = k_1*X_0 - k_2*S_2$ ;
END

The intermediate variables ($S_1$ and $S_2$ in the example above) are declared when they are defined. The following restrictions apply to the user of intermediate variables:

- Intermediate variables must be defined before they are referenced in other expressions.
- Intermediate variables are not visible outside the scope where they are defined, even with the scope resolution operator (e.g., we cannot access $S_1$ or $S_2$ in a simulation that instantiates a `Reactor` model).

Arrays of intermediate variables are also supported. Any array intermediate must be dimensioned in a `DIMENSION` statement at the beginning of an `INTERMEDIATE` section. For example

INTERMEDIATE
DIMENSION $V(N), A(M,N)$
DIMENSION $F(L)$
$V := X*(X+LOG(X))$ ;  # $X$ is also an array of dimension $N$
FOR $I := 1$ TO $M$ DO
  FOR $J := 1$ TO $N$ DO
    $A(I,J) := X(I)*V(J)$ ;
  END
END
$F(1) := 1$ ;
JACOBIAN Modeling Language Syntax Manual

F(2) := 1 ;
F(3:L) := F(2:L-1)+F(1:L-2) ;

The introduction of intermediate variables can often improve the performance of a calculation by reducing the number of variables in the problem and by eliminating variables for which good initial guesses may not be known. However, some caution is advised. Since the expressions assigned to intermediate variables are actually substituted in the model equations that reference them, the sparsity of the model equations will decrease. The trade-off between problem size and problem sparsity is difficult to know a priori.

Equation

The model equations are defined in the equation section. The equations may be composed of parameters and variables declared in the model or in sub-units of the model, the global variable TIME, literal constants, and intrinsic functions. Unlike assignments, the equations are declarative. That is, the equation \( y = x \) is equivalent to \( y-x = 0 \). The equation section begins with the keyword EQUATION and is followed by a list of one or more simple or structured equations. Examples of simple (or unstructured) equations are

\[
\begin{align*}
  x + y &= x^2 \quad ; \\
  a \ is \ b & \quad ; \\
  u.z &= y-4+\text{SIN}(\pi\times\text{TIME}) \ ;
\end{align*}
\]

Notice that each equation is terminated with a semi-colon '\;'. The equation may span several lines and several equations may appear on the same line. The keyword "is" a synonym for the equality operator '='. The structured statements supported by JACOBIAN are described below.

Equations involving slices

Variables, parameters, selectors, and units that are defined as arrays may be accessed as slices. For example, suppose \( A \) is an array of units, \( B \) is a scalar, \( C \) is a scalar unit, \( D \) is an array, and \( E \) is a scalar. The array equation

\[
A(1:3).B = C.D(4:6) + E ;
\]

is equivalent to the following three equations:

\[
\begin{align*}
  A(1).B &= C.D(4) + E ; \\
  A(2).B &= C.D(5) + E ; \\
  A(3).B &= C.D(6) + E ;
\end{align*}
\]

The dimensionality of all terms in an array equation must be the same, with the special case that the dimensionality of a scalar is matched to the other arrays in the expression (e.g., \( E \) in example above). If a symbol in an equation is an array and no subscripts are provided then the slice will be implicitly the entire array. For example, suppose \( X \) and \( Y \) are arrays of dimension 3, then the array equation

\[
 X = Y ;
\]

is equivalent to

\[
\begin{align*}
  X(1) &= Y(1) ; \\
  X(2) &= Y(2) ; \\
  X(3) &= Y(3) ;
\end{align*}
\]

As with slices, all array symbols without subscripts in an expression must have the same dimension.
**Intrinsic functions**

The intrinsic functions supported in JACOBIAN are listed in Table 1.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Argument</th>
<th>Return</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Absolute value of the argument</td>
</tr>
<tr>
<td>ACOS</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Arccosine of the argument in radians</td>
</tr>
<tr>
<td>ASIN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Arcsine of the argument in radians</td>
</tr>
<tr>
<td>ATAN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Arctangent of the argument in radians</td>
</tr>
<tr>
<td>CARD</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Largest integer that does not exceed the argument</td>
</tr>
<tr>
<td>COS</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Cosine of the argument in radians</td>
</tr>
<tr>
<td>COSH</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Hyperbolic cosine of the argument</td>
</tr>
<tr>
<td>EXP</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Exponential of the argument</td>
</tr>
<tr>
<td>IMPORT_DATA</td>
<td>First is scalar string, second is scalar integer</td>
<td>N/A</td>
<td>Used in DATA section of ESTIMATION block (see below)</td>
</tr>
<tr>
<td>INDEX_OF</td>
<td>First is scalar parameter, second is parameter array</td>
<td>Scalar integer</td>
<td>Position of the scalar (first argument) in the array (second argument). Returns zero if scalar does not appear in the array. For example, let iarray := [3,4,2] then, INDEX_OF(2,iarray) returns 3 and INDEX_OF(5,iarray) returns 0.</td>
</tr>
<tr>
<td>INT</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Largest integer that does not exceed the argument</td>
</tr>
<tr>
<td>LOG</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Natural logarithm of the argument</td>
</tr>
<tr>
<td>LOG10</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Logarithm to base 10 of the argument</td>
</tr>
<tr>
<td>MAX</td>
<td>Scalar or vector</td>
<td>Scalar</td>
<td>Largest of all elements of all arguments</td>
</tr>
<tr>
<td>MID</td>
<td>Three scalar arguments</td>
<td>Scalar</td>
<td>Return middle value of three arguments</td>
</tr>
<tr>
<td>MIN</td>
<td>Scalar or vector</td>
<td>Scalar</td>
<td>Smallest of all elements of all arguments</td>
</tr>
<tr>
<td>OLD</td>
<td>Scalar</td>
<td>Scalar</td>
<td>Return value of argument at end of last time/state event</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>Scalar or vector list</td>
<td>Scalar</td>
<td>Product of all elements of all arguments</td>
</tr>
<tr>
<td>SGN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Sign of the argument</td>
</tr>
<tr>
<td>SIGMA</td>
<td>Scalar or vector list</td>
<td>Scalar</td>
<td>Sum of all elements of all arguments</td>
</tr>
<tr>
<td>SIGN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Same as SGN</td>
</tr>
<tr>
<td>SIN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Sine of the argument in radians</td>
</tr>
<tr>
<td>SINH</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Hyperbolic sine of the argument</td>
</tr>
<tr>
<td>SQRT</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Square root of the argument</td>
</tr>
<tr>
<td>TAN</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Tangent of the argument in radians</td>
</tr>
<tr>
<td>TANH</td>
<td>Scalar or vector</td>
<td>Same as argument</td>
<td>Hyperbolic tangent of the argument</td>
</tr>
</tbody>
</table>

**FOR Equations**

The first structure equation is the FOR loop. The general syntax is

```plaintext
FOR Identifier := start TO end [ STEP step ] DO
equation-list
END
```

where **Identifier** is the name of the loop control variable, **start** is the starting value, **end** is the ending value, **step** is an optional step increment, and **equation-list** is a list of other equations (structure and/or unstructured). The loop control variable does NOT need to be declared in the variable or parameter sections of the model. The name of the loop control variable will hide another variable/parameter with the same name within the loop body. The starting, ending, and (optional) step expressions of the loop header may be arbitrary integer expressions in terms of literals and parameters. An example is
FOR I := 1 TO N DO
    Y(I) = K(I)*X(I) ;
END

This is equivalent to the array equation

\[ Y = K \times X ; \]

Notice that the FOR loop is not terminated with a semi-colon.

**IF Equations**

An IF equation is one way to represent hybrid discrete/continuous phenomena within a JACOBIAN model. The general syntax of an IF equation is

IF logical-condition THEN
    true-statement-list
ELSE
    false-statement-list
END

where logical-condition is a general logical expression, true-statement-list is the list of equations that is active if the logical expression evaluates to true, and false-statement-list is the list of equations that are active when the logical expression evaluates to false. An example is

IF (X>Y OR X>Z) THEN
    X^2 – Y = 0 ;
ELSE
    IF(X>0) THEN
        SQRT(X) – Y = 0 ;
    ELSE
        Y = 0 ;
    END
END

The supported relational operators are >, >=, <, and <=. The supported logical operators are AND, OR, NOT, and EOR (exclusive OR). Notice that the IF equation is not terminated with a semi-colon.

**CASE equations**

The CASE equation is the second mechanism a user may introduce discrete behavior into a model. The CASE equation is more general than the IF equation. The construct is best illustrated through and example:

SELECTOR
    Disc_Flag AS { Intact, Burst } DEFAULT Intact

CASE Disc_Flag OF
WHEN Burst:
    Relief_Flow = Press/SQRT(TMP) ;
WHEN Intact:
    Relief_Flow = 0 ;
    SWITCH TO Burst IF Press >= BurstPressure ;
END
In this example, the value of SELECTOR Disc is used to determine which set of equations is currently active. The CASE equation defines a series of discrete modes as a series of WHEN blocks. The WHEN block may contain an arbitrary number of structured or unstructured equations. The current value of the selector determines which block is active. Also contained in the WHEN block is a list of one or more SWITCH statements that allow the value of the selector to change under certain conditions. The SWITCH statements are not necessary; in the case above when the value of the selector is Burst, it will never return to the Intact mode. The general syntax may be defined as follows.

```
CASE selector-name OF
  WHEN-block-list
END
```

where `selector-name` is the name of a selector and `WHEN-block-list` is a list of WHEN-blocks. A WHEN-block is defined as

```
WHEN selector-value :
  equation-list
  SWITCH-list
```

where `selector-value` is a valid value of the selector `selector-name`, `equation-list` is a sequence of one or more structured or unstructured equations, and `SWITCH-list` is a list of zero or more SWITCHes. A SWITCH is define as

```
SWITCH TO selector-value IF logical-expression
```

where `selector-value` is a valid value of the selector `selector-name` and `logical-expression` is any valid logical expression.

**STREAM equations**

As described in the stream section above, streams provide a convenient shorthand mechanism for equating several variables in two units with a single equation. The general syntax for a stream equation is

```
Stream-variable-1 IS Stream-variable-2 ;
```

where `Stream-variable-1` and `Stream-variable-2` are two separate streams of the same type and dimensionality. Consider the following example.

```
DECLARE
  TYPE
    Concentration = 0.01 : 0.0 : 100 UNIT="mol/L"
    Temperature = 300 : 100 : 500 UNIT="K"
    Pressure = 1 : 0.03 : 10 UNIT="bar"
STREAM
  ProcessStream IS Temperature, Pressure, Concentration
END

MODEL A
PARAMETER
  N AS INTEGER
VARIABLE
  Conc_in, Conc_out AS ARRAY(N) OF Concentration
  Tin, Tout AS Temperature
```
P AS Pressure
STREAM
  Inlet : Tin, P, Conc_in AS ProcessStream
  Output : Tout, P, Conc_out AS ProcessStream
END

MODEL B
PARAMETER
  N, M AS INTEGER
UNIT
  U AS ARRAY(M)OF A
SET
  N := 2 ;
  M := 4 ;
EQUATION
  U(1:M-1).Outlet IS U(2:M).Inlet ;
END

The single stream connection in the example above is equivalent to

U(1).Tout = U(2).Tin ;
U(1).P = U(2).P ;
U(1).Conc_out(1) = U(2).Conc_in(1) ;
U(1).Conc_out(2) = U(2).Conc_in(2) ;
U(2).Tout = U(3).Tin ;
U(2).P = U(3).P ;
U(2).Conc_out(1) = U(3).Conc_in(1) ;
U(2).Conc_out(2) = U(3).Conc_in(2) ;
U(3).Tout = U(4).Tin ;
U(3).P = U(4).P ;
U(3).Conc_out(1) = U(4).Conc_in(1) ;
U(3).Conc_out(2) = U(4).Conc_in(2) ;

Simulation Blocks

A simulation block starts with the SIMULATION keyword, followed by a name, and is terminated with the END keyword. A simulation block consists of the following sections:

  Options
  Parameter
  Unit
  Variable
  Report
  Sensitivity
  Ask
  Set
  Intermediate
  Equation
  Input
  Preset
  Selector
  Initial
  Schedule
Not all of the sections are required. However, those present must be in the order above. Like the model, the simulation name has global scope and thus must be unique.

**Options**

Several options for how the computation is performed are available to the user and specified in the option section. The option section begins with the keyword `OPTIONS` and is following by a list of zero or more option specifications. The syntax of this specification is

```
option-keyword := option-value ;
```

where the list of option keywords is described in Table 2.

<table>
<thead>
<tr>
<th>Option Keyword</th>
<th>Option Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_MATH_EXCEPTIONS</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>Will check for mathematical exceptions, such as division by zero, during evaluation and report to user.</td>
</tr>
<tr>
<td>ANALYSIS</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>If FALSE, structural analysis of the problem will not be performed</td>
</tr>
<tr>
<td>INIT_RELATIVE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Relative error used during consistent initialization</td>
</tr>
<tr>
<td>INIT_ABSOLUTE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Absolute error used during consistent initialization</td>
</tr>
<tr>
<td>INIT_MAX_ITERATIONS</td>
<td>INTEGER &gt; 0</td>
<td>100</td>
<td>Maximum number of iteration used during consistent initialization (e.g., maximum number of Newton iterations)</td>
</tr>
<tr>
<td>INIT_PRINT_LEVEL</td>
<td>INTEGER ≥ 0</td>
<td>0</td>
<td>Print level during initialization (0=no printing, &gt;0=more printing)</td>
</tr>
<tr>
<td>INIT_BOUNDS</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>True if bounds checking is to be performed during initialization</td>
</tr>
<tr>
<td>INIT_BLOCK_SOLVE</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>Will solve initialization problem by decomposing system into a sequence of smaller blocks if possible.</td>
</tr>
<tr>
<td>REINIT_BLOCK_SOLVE</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>Will solve reinitialization problem by decomposing system into a sequence of smaller blocks if possible.</td>
</tr>
<tr>
<td>BLOCK_SOLVE</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>Same as setting both options above to true or false.</td>
</tr>
<tr>
<td>REINIT_RELATIVE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Relative error used during consistent re-initialization</td>
</tr>
<tr>
<td>REINIT_ABSOLUTE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Absolute error used during consistent re-initialization</td>
</tr>
<tr>
<td>REINIT_MAX_ITERATIONS</td>
<td>INTEGER &gt; 0</td>
<td>100</td>
<td>Maximum number of iteration used during consistent re-initialization (e.g., maximum number of Newton iterations)</td>
</tr>
<tr>
<td>REINIT_PRINT_LEVEL</td>
<td>INTEGER ≥ 0</td>
<td>0</td>
<td>Print level during re-initialization (0=no printing, &gt;0=more printing)</td>
</tr>
<tr>
<td>REINIT_BOUNDS</td>
<td>BOOLEAN</td>
<td>TRUE</td>
<td>True if bounds checking is to be performed during re-initialization</td>
</tr>
<tr>
<td>DYNAMIC_RELATIVE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Relative error used during numerical integration</td>
</tr>
<tr>
<td>DYNAMIC_ABSOLUTE_TOLERANCE</td>
<td>REAL &gt; 0</td>
<td>1e-6</td>
<td>Absolute error used during numerical integration</td>
</tr>
<tr>
<td>DYNAMIC_SCALING</td>
<td>BOOLEAN</td>
<td>FALSE</td>
<td>If true, iteration matrix will be automatically scaled during</td>
</tr>
</tbody>
</table>
DYNAMIC_INITIAL_STEPSIZE
REAL > 0
COMPUTED
Allows user to specify the initial stepsize used during numerical integration. Sometimes useful if problem is extremely stiff at start of integration. Program will compute its own stepsize if this option is omitted.

DYNAMIC_PRINT_LEVEL
INTEGER ≥ 0
0
Print level during integration (0=no printing, >0=more printing)

DYNAMIC_BOUNDS
BOOLEAN
TRUE
True if bounds checking is to be performed during numerical integration

DYNAMIC_REPORTING_INTERVAL
REAL ≥ 0
0
Reporting interval for time series data

CSVOUTPUT
BOOLEAN
FALSE
If true, reported time trajectories will be saved to a comma-separated values file. The name of this file is the simulation name with the extension "csv" and stored in the output folder.

NORMALIZED_SENS
BOOLEAN
FALSE
Compute normalized sensitivities

SENSITIVITY_ERROR_CONTROL
BOOLEAN
TRUE
If false, only state variables, not sensitivity variables, will be placed under error control during sensitivity analysis.

ESTIMATION_PRINT_LEVEL
INTEGER ≥ 0
0
Print level during estimation (0=no printing, >0=more printing)

ESTIMATION_TOLERANCE
REAL > 0
1e-6
Set estimation_objective_tolerance and estimation_gradient_tolerance

ESTIMATION_OBJECTIVE_TOLERANCE
REAL > 0
1e-6
The estimation calculation will halt if the normalized difference between objective function values between two successive iterations is less than this value. The normalization factor is \( \max \left( \frac{\| f_j \|}{\| f_j \| + 1} \right) \) where \( f_j \) is the objective function at j-th iteration.

ESTIMATION_GRADIENT_TOLERANCE
REAL > 0
1e-6
The estimation calculation will halt if the norm of projected gradient of the objective function is less than this value.

ESTIMATION_OBJECTIVE_TO_FILE
BOOLEAN
FALSE
If true, a file named <computation name>.obj will be created containing the final objective function value

EXPORT_ESTIMATED_PARAMETERS
BOOLEAN
FALSE
If true, a file named <computation name>.est will be created containing the final values of the estimated parameters

Parameter
Time invariant parameters may be specified in the simulation block in exactly the same way they are specified in the model block described above.

Unit
Units are instantiated in a simulation block in the same way that was shown in the model unit section. The units that are instantiated in the simulation block form the system of equations that will be solved.
Variable

Variables may be specified in the simulation block in exactly the same way they are specified in the model block described above.

Report

The system of equations of interest in a simulation may involve tens of thousands of variables. However, we are usually only interested in examining a small subset of these variables. The report section allows the user to specify which variables are to be saved for later visualization. The syntax is simply the keyword REPORT followed by a list of variable names. An example report section is

```
REPORT
  U.X(1:N-2), U.Y(1,1), U.A.A
```

If arrays with missing subscripts are contained in the report list, then all components will be reported. For example, suppose in the report section above suppose the size of the first dimension of Y is 4 and the size of the second is 3. Then the following components of Y will be reported: U.Y(1,1), U.Y(1,2), and U.Y(1,3). Limiting the amount of output makes visualization easier (e.g., fewer variables to sort through when plotting) and reduces the size of the files generated. The size of the data files can also be reduced by choosing a suitable reporting interval (see options section). If the option CSVOUTPUT is set to true then the order of the variables in the comma-separated-value file is the same as the order of the variables in the REPORT section.

Sensitivity

A parametric sensitivity analysis calculation may be readily performed within JACOBIAN. The time invariant parameters of interest are specified in the sensitivity section. The syntax of the sensitivity section is the keyword SENSITIVITY followed by a list of one or more assignments to the parameters of interest. For example, the section

```
SENSITIVITY
  WITHIN U DO
    RateConstant(1:4) := [ 1e11, 1.1e12, 2e13, 5e10 ] ;
    RateConstant(5:Nrxn) := 1e12 ;
  END
```

would result in the computation of the sensitivities of all state variables with respect to the Nrxn rate constants. The selected parameters must be real-valued (e.g., type REAL). Unlike normal parameters where the values may be assigned by propagation, the exact path of the parameter of interest must be specified in the sensitivity section.

With the option of NORMALIZED_SENS, normalized sensitivities can be calculated, which are defined as

\[
\text{normalized\_sensitivity} = \begin{cases} 
\frac{\partial \ln y}{\partial \ln p}, & p \neq 0, y \neq 0 \\
\frac{\partial y}{\partial p}, & y \equiv 0 \\
1 \frac{\partial y}{y \partial p}, & p \equiv 0
\end{cases}
\]
Ask

The ask section allows the value of a parameter to be defined interactively during run-time. An example of the syntax of an ask section is

```
ASK
  WITHIN Reactor DO
    Param1 "Enter value of first parameter " ;
    Param2 "Enter value of second parameter " ;
  END
```

The user will be prompted to enter the values of the parameters in the ask section before the simulation is executed. The prompt specified in the input file is issued when the value is requested. In the JACOBIAN IDE, the user will enter values in the separate terminal window that appears when the IDE is started. However, when using JACOBIAN outside of the IDE, e.g., stand-alone through a command-line interface or with another application like Microsoft Excel, the input function used to obtain parameter values may be modified by the user in order to accept values from a variety of sources. This is described in more detail in the JACOBIAN API Manual and the JACOBIAN-Excel Interface Manual.

Set

The set section of a simulation is identical to the set section of a model described above.

Intermediate

The intermediate section of a simulation is identical to the intermediate section of a model described above.

Equation

The equation section of a simulation is identical to the equation section of a model described above.

Input

The syntax of the input section is identical to that of a set section. However, rather than assigning values to the time invariant parameters of a model or simulation, values of one or more variables are specified. An example input section is

```
INPUT
  WITHIN U DO
    v := V0*SIN(W*TIME) ;
    A := 50 ;
  END
```

The value assigned to the input may be any real-valued expression involving literals, parameters, and/or `TIME`. As described earlier, the user is encouraged to write generic models with additional degrees of freedom. The input section is where these additional degrees of freedom may be specified to make the problem fully-determined.

Input functions may also be assigned values by using an `IMPORT` function within the `INPUT` section. The `IMPORT` function may be implemented by the user to import values from a variety of sources, such as spreadsheets and databases. A detailed discussion of `IMPORT` functions and their implementation appears in other documents, including the JACOBIAN-VBA Interface Manual.
Preset

The syntax of the preset section is identical to that of the set section except that the targets of the assignment are variables and the source expression may contain optional bounds. An example preset section is

```plaintext
PRESET
  U.X(1) := 0.334 : 0.1 ; # use default upper bound
  U.X(2) :=  : 0.01 : 1.0 ; # use default value
  U.X(3) := 0.566 : 0.7 ; # use default lower bound
  U.X(4) := 0.001 ; # use default bounds
```

The preset section allows the user to override the default values, lower bounds, and/or upper bounds of selected variables. Often when initializing DAE (differential-algebraic equation) systems a good initial guess for the solution is necessary. The preset section allows the user to provide these initial guesses and possibly bounds away from undesired solutions.

Selector

The selector section in a simulation is where the selectors defined in a model may be assigned initial values. An example selection is

```plaintext
SELECTOR
  U.Disc_Flag := Burst ;
  U.Valve.Position := CLOSED ;
```

The target of the assignment must be a selector and the value begin assigned must be a compatible enumeration value.

Initial

The initial conditions of a dynamic calculation are provided in the initial section. The general syntax of an initial section is

```plaintext
INITIAL
  Equation-list
```

where `Equation-list` is a sequence of one or more JACOBIAN equations (structure and/or unstructured). JACOBIAN supports a much more general specification of the initial conditions that simply a series of assignments to initial values. A common initial condition is steady-state (i.e., where all time derivatives are set to zero). JACOBIAN supports a keyword for this:

```plaintext
INITIAL
  STEADY_STATE
```

Schedule

The schedule section of a dynamic simulation describes the series of actions that are to be applied to the dynamic system. The syntax of schedule section is

```plaintext
SCHEDULE
  schedule-element
```

where `schedule-element` is one of the following tasks:

- `Continue`
- `Display`
- `Export`
**Sequence**

The sequence task groups together one or more other tasks into a list that is executed sequentially. The syntax of the sequence task is

```
SEQUENCE
  Task-list
END
```

where **Task-list** is a list of one or more other tasks. An example of a sequence task is

```
SEQUENCE
  CONTINUE FOR 20
  RESET
    U.ValvePosition := 1.0 ;
  END
  CONTINUE FOR 10 OR UNTIL U.Level > U.MaxLevel
END
```

**Continue**

The continue task defines a time interval for numerical integration. The syntax of the four forms of the continue task is

```
CONTINUE FOR expression
CONTINUE UNTIL logical-expression
CONTINUE FOR expression OR UNTIL logical-expression
CONTINUE FOR expression AND UNTIL logical-expression
```

For example, the following continue task will execute for 30 units of time or until the temperature is greater than 350:

```
CONTINUE FOR 30 OR UNTIL U.Temp > 350
```

**Reset**

A reset task allows one or more inputs to be changed during the simulation. The syntax of a reset task is

```
RESET
  Assignment-list
END
```

**Assignment-list** is a list of one or more assignments where the target variable is an input function and the source expression is any valid expression involving literals, parameters, and/or **TIME**. Also supported is the intrinsic function **OLD**. The **OLD** function allows the current value of an input function to be accessed in the source expression. For example, the following reset task will double the value of the input function:

```
RESET
  U.ValvePosition := 2 * OLD(U.ValvePosition) ;
END
```
U.Input := 2*OLD(U.Input) ;
END

Like the INPUT section, IMPORT functions may be used in the RESET task in order to import values from a variety of locations.

**Reinitial**

The reinitial task allows the user to halt dynamic simulation and re-initialize one or more differential variables\(^2\) using specified initial conditions. The syntax of a reinitial task is

```
REINITIAL
  Variable-list
  WITH
    Equation-list
END
```

 Variable-list is a list of one or more differential variable names (with full path) and Equation-list is a list of one or more JACOBIAN equations. The number of variables and equations in these lists must be equal. The remaining differential variables (the ones not listed) are held fixed during re-initialization. The following reinitial task will introduce a step change in the mass of the system (assumes the derivative of mass appears in the model equations):

```
REINITIAL
  U.Mass
  WITH
    U.Mass = U.Mass + 10 ;
END
```

**Display**

A display task allows the user to display the values of a list of one or more expressions at any point in a schedule. The syntax is

```
DISPLAY expression-list END
```

where expression-list is any valid list of JACOBIAN expressions ( scalars, arrays, slices, etc). For example,

```
DISPLAY Reactor.T+273.15, Reactor.P, Reactor.X(1:3) END
```

This information is displayed to the standard output of JACOBIAN (e.g., Message window in the IDE). However, as described in the JACOBIAN API Manual, the output handler used by JACOBIAN for display may be implemented by the user in order to send the values to any desired location.

**Export**

The second way to output specific values during a JACOBIAN simulation is with the export task. There are two forms of the export task:

```
EXPORT(String) expression
EXPORT(Integer,Integer,Integer) expression
```

For example,

```
EXPORT("SimResults.dat") Reactor.T-273.15
EXPORT(1,2,3) Reactor.P
```

---

\(^2\) A differential variable is a variable in a JACOBIAN model that has a time derivative.
EXPORT(1,2,4) Reactor.X(1)

Unlike display tasks, only scalar-valued expressions may appear in an export task. Like the other input/output functions, the user may write their own export function(s) in order to export data to different locations (e.g., spreadsheet or database). A detailed discussion of EXPORT functions and their implementation appears in other documents, including the JACOBIAN-VBA Interface Manual.

**While**

This task specifies that a schedule item is to be executed while a logical condition is satisfied. The syntax is

```plaintext
WHILE (logical-expression)
  Schedule-item
END
```

The task item is repeated as long as `logical-expression` evaluates true. The task will be exited when the logical expression evaluates to false and there is no attempt to find the exact time where this change takes place (in contrast to conditional statements in the model equations or continue task). For example,

```plaintext
WHILE (U.Input<100)
  SEQUENCE
    CONTINUE FOR 10                     # Allow system to evolve to 10 min
    DISPLAY U.Temperature END           # Display current value of temp.
    RESET
      U.Input := OLD(U.Input) + 10 ;   # Increment input function
    END
END
END
```

**IF**

Like the while task described above, the IF task allows schedule items to be conditionally executed. There are two forms of the IF task:

```plaintext
IF (logical-expression) THEN
  Schedule-item
END
```

and

```plaintext
IF (logical-expression) THEN
  Schedule-item
ELSE
  Schedule-item
END
```

An example IF task is

```plaintext
IF (U.level<0) THEN
  RESET
    U.level := 0 ;
  END
ELSE
  CONTINUE UNTIL U.level <= 0
END
```
Save Presets

The two forms of the save presets task are

SAVE PRESETS identifier
SAVE PRESETS quoted-string

This task will create a file containing the current values of the variables in a form suitable for copying (or including) in a preset section. If the identifier form is used then the file created is named identifier with prefix ".PRESETS" and stored in the output folder. If the quoted string form is used then the preset file is named and stored exactly as specified in the task.

Save Initial

The two forms of the save initial task is

SAVE INITIAL identifier
SAVE INITIAL quoted-string

This task will create a file containing the current values of the differential variables (variable with time derivatives that appear explicitly in the model) in a form suitable for copying (or including) in an INITIAL section. If the identifier form is used then the file created is named identifier with prefix ".INITIAL" and stored in the output folder. If the quoted string form is used then the preset file is named and stored exactly as specified in the task.

Estimation Blocks

An estimation block starts with the ESTIMATION keyword, followed by a name, and is terminated with the END keyword. Estimation blocks consist of the following sections:

Options
Parameter
Unit
Variable
Uncertain
Ask
Set
Intermediate
Equation
Objective
Experiment

As in model and simulation blocks, not all sections are required, but when present they must be in the order shown above. The options, parameter, unit, variable, ask, set, and equations sections are identical to those in the simulation block.

Uncertain

The uncertain section is where the user specifies the time invariant parameters to be estimated. The uncertain section begins with the keyword UNCERTAIN. The syntax is identical to the set section except that upper and lower bounds on the parameters must be provided. An example uncertain section is shown below.

UNCERTAIN
   WITHIN U DO
      Param1 := 11 : 0.01 : 1000 ;
The target symbols (left-hand-side of assignment) must be real-valued time invariant parameters and the full path must be specified. The right-hand-side expression contains the initial guess, lower, and upper bounds, respectively, of the parameter. These expressions must be constant and real-valued.

**Objective**

The objective function of the parameter estimation problem is specified in the objective section. The syntax of the two forms of the objective section is

```
OBJECTIVE objective-function-keyword
OBJECTIVE expression
```

where `objective-function-keyword` is a keyword specifying an objective function type (e.g., `WEIGHTED_LEAST_SQUARES` or `MAXIMUM_LIKELIHOOD`) and `expression` is any valid JACOBIAN real-valued expression. Currently, only the weighted least squares and maximum likelihood objective functions are supported.

**Objective Functions Defined**

Using the weighted least squares option, the objective function to minimize in the parameter estimation problem is

\[
\phi^{WLS}(\boldsymbol{p}) = \frac{1}{2} \sum_{i=1}^{N_{exp}} \sum_{j=1}^{N_v,i} \sum_{k=1}^{N_m,ij} W_{ijk} (\tilde{z}_{ijk} - z_{ijk}(\boldsymbol{p}))^2
\]

where \(\boldsymbol{p}\) is the set of time invariant parameters specified in the `uncertain` section, \(N_{exp}\) is the number of experiment sections (see below), \(N_{v,i}\) is the number of measured variables in each experiment, \(N_{m,ij}\) is the number of measurements of variable \(j\) in experiment \(i\), \(W_{ijk}\) is the weight associated with \(k\)-th measurement of the \(j\)-th variable in the \(i\)-th experiment, \(\tilde{z}_{ijk}\), and \(z_{ijk}(\boldsymbol{p})\) is the corresponding value computed from the model. The maximum likelihood objective function is

\[
\phi^{ML}(\boldsymbol{p}) = \frac{N}{2} \ln(2\pi) + \frac{1}{2} \sum_{i=1}^{N_{exp}} \sum_{j=1}^{N_v,i} \sum_{k=1}^{N_m,ij} \ln(\sigma_{ijk}^2) + \frac{(\tilde{z}_{ijk} - z_{ijk}(\boldsymbol{p}))^2}{\sigma_{ijk}^2}
\]

In this case, \(N\) is the total number of measurements of all variables in all experiments and \(\sigma_{ijk}\) is the variance of measurement \(\tilde{z}_{ijk}\).

**Experiment**

The experimental data associated with a parameter estimation problem (i.e., the data the estimation problem attempts to fit by adjusting the parameters) is supplied in one or more experiment sections. The experiment section begins with the keyword `EXPERIMENT` followed by an experiment name. Each experiment must have a unique name within an estimation block. The general syntax of the experiment block is

```
EXPERIMENT experiment-name
    DATA
        Data-list
    INPUT
        Input-list
```
The input, preset, selector, initial, and schedule sections of the experiment are identical to those in the simulation block. These sections are used to define the conditions of the particular experiment (e.g., initial conditions, simulation time, input functions, etc). The data section is where experimental data is provided and it has the following syntax:

```
DATA  
Data-entry-list
```

where `Data-entry-list` is a list of one or more `Data-entry`. The syntax of the `Data-entry` is

```
Variable [ time-point ] := value : weight ;
```

where `Variable` is the name of the measured variable (this must be a variable within the model and the full path must be specified), “[time-point]” denotes the time during the simulation the measurement was taken, `value` is the measured value, and `weight` is the weighting associated with the measurement. In the case of the maximum likelihood objective function, the weight corresponds to the variance of the measurement. An example data section is

```
DATA  
# variable time   value  weight
V.Flowrate[0]  := 0.13 : 1.0 ;
V.Flowrate[5]  := 0.24 : 1.0 ;
V.Flowrate[10] := 0.53 : 1.0 ;
WITHIN U DO
  Temp[0]  := 298 : 1.0 ;
  Temp[5]  := 301 : 1.0 ;
  Temp[10] := 304 : 1.0 ;
END
```

In this example, the flowrate and temperature are measured at three points (at times 0, 5, and 10). All measurements are given equal weighting.

**Importing data from text files**

JACOBIAN supports the ability to import data from text files. An example of this new syntax is shown below.

```
DATA  
U.X[0]   := 0.123 : 1.0 ;
U.X      := IMPORT_DATA("DataFile.txt",U.Ndata) ;
U.X[500] := 0.001 : 1.0 ;
```

In this example the data is specified both with data assignments and by importing from a file. The file is named "DataFile.txt" and it contains `U.Ndata` data points. The format of this file is:

```
time_1,value_1,weight_1
time_2,value_2,weight_2
```
time_Ndata, value_Ndata, weight_Ndata

The data file name must contain the full path of the file. As with IMPORT functions and EXPORT tasks, JACOBIAN allows the user to implement their own IMPORT DATA functions. This is described in other documentation, including the JACOBIAN-VBA Interface Manual.

**Data at the final integration time**

Often the final integration time of a simulation is not known a priori, but rather determined by some state condition being satisfied. The data in an estimation problem can be specified at this endpoint using the FINAL_TIME keyword. An example is:

```
DATA
  U.Conc(1)[FINAL_TIME] := 0.348 : 0.0001 ;
```

One example where this may occur is where the concentration in a tubular reactor is measured at the outlet, but the effective length of the reactor (and hence final integration time) is computed from the catalyst load.