I M P L ©

"M a k i n g O p t i m i z a t i o n a n d E s t i m i z a t i o n F a s t e r , B e t t e r , S m a r t e r !"

Industrial Linear / Logic / Logistics and Nonlinear Programming Language (ILP / INP)

ILP© / INP© Foreign-Files (LP/QP/MIP/NLP) Deployment Manual

i n d u s t r i @ l g o r i t h m s

"IMPLementing Industrial Optimization & Estimization Applications Faster, Better, Smarter!"

(Better Data + Better Decisions = Better Business)

Release 1.10, January 2026, *IAL-IMPL-ILP-INP-RM-1-10*

Introduction

The ILP and INP foreign-files are considered as “foreign-models” or non-standard sub-models (or models) that can add-on, augment, supplement and/or extend an existing, non-foreign or standard model with extra foreign-variables and -constraints within IMPL albeit in a simplied scalar and somewhat sparsely-formed or modeled fashion. These foreign-models provide a basic or lower-level algebraic modeling and/or scripting language (AML, ASL) capability especially when compared to AIMMS, AMPL, GAMS, LINGO, LPL, MOSEL, MPL, OPL, ZIMPL, etc. The foreign-model ILP’s / INP’s “Xnnn” / “Xname” mnemonics denote the foreign-variables and the “Fmmm” / “Fname” mnemonics denote the foreign-constraints where “nnn” and “mmm” are indexes or indices in the whole number domain i.e., non-negative integers. IMPL supports including only one foreign-model i.e., either a single ILP file or a single INP foreign-file but not both where only one respective ILPet or INPet foreign-file for the same problem or sub-problem may also be read, loaded, imported or inputted *before or prior* to their respective ILP / INP foreign-files i.e., a preliminary or pre-ILP / -INP foreign-file. Foreign-files are optional and are inputted, imported, loaded or read by the IMPL Modeler only after the entire non-foreign or standard IMPL model has been constructed, created, generated, formulated or modeled where the minimum IMPL standard variables are the five (5) objective function terms i.e., (1) profit, (2) performance 1-norm, (3) performance 2-norm, (4) penalties and (5) total. The cooresponding minimum standard constraints are the same except that strictly speaking they exclude, are absent or missing the 2-norm performance term as this term is internally modeled as part of the QP solver or sub-solver only and externalized after a solution has been reached.

Please refer to the SSIIMPLE Manual for the receive and retrieve foreign-model functions to programmatically and systematically set (receive, insert, update) and get (retrieve, view) foreign-variables and foreign-constraints called or invoked from most computer programming and/or scripting languages i.e., **IMPLreceiveX(), IMPLreceiveFL(), IMPLreceiveFLX(),** etc. These foreign-modeling functions can be used to code industrially-relevant optimization and estimization problems in any size, scale and scope but rely on the user, modeler or analyst to properly organize both the data and model structures of the problem or sub-problem i.e., its sets, lists, catalogs, parameters, variables, constraints, derivatives, expressions and formulas within their programming and/or scripting source code. Although the ILP / INP foreign-files can be used to model and solve real industrial optimization and estimization problems of significance, its scalar-based and somewhat sparsely-formed or simple set-based language

may require a relatively longer time and effort for model construction, creation, generation or formation as each foreign-parameter (numeric constants, calc-scalars and data-vector-elements), foreign-variable (Xnnn / Xname) and foreign-constraint (Fmmm / Fname) instance must ultimately be read, inputted or imported from the foreign-file which is an ASCII (UTF-8) flat-file. To reduce this overhead, the aforementioend programmable / scriptable foreign-modeling functions should be considered which can substantially decrease the initial model building, generation or construction time and enable more complex and sophisticated sparsely-formed foreign-models to be expressed, formulated, represented and later solved.

Foreign-files may also be used to add, insert or overlay user, adhoc, bespoke or custom variables and/or constraints into the overall model or problem which is the primary purpose or driver for IMPL's foreign-modeling capability. Collectively, IML, OML, ILPet / ILP and INPet / INP (foreign-files) are considered as IMPL-DATA's *de facto* (scalar-based and somewhat sparsely-formed or simple set-based) algebraic or mathematical modeling language referred to as **IDL (IMPL-DATA Language or Industrial Data Language)** which does not include any of the standard / non-foreign IMPL modeling constructs, entities or objects (i.e., structures and semantics) relating to IMPL's UOPSS-QLQP© or UQF© (see also USN©-FSFS© / UFS© / USF©). Since the ILP / INP foreign-files are read, loaded, inputted or imported after the standard problem data have been read, loaded, inputted or imported either by an IML file and/or IPL functions, calc-scalar and data-vector-elements may be included and referenced in the foreign-files for configuring objective function weights, constraint coefficients or parameters, constraint nonlinear formulas, constraint right-hand-sides (R.H.S.'s), variable lower and upper bounds, variable types or senses and variable initial-values.

Foreign-models are what we call "scalar-based" or term-based and are somewhat "sparsely-formed" or maybe considered as "simple set-based" sub-models or models when IMPL standard or non-foreign-models do not exist or are absent. Although there are no "structure-based", "semantic-based", "stream-based" nor "set-based" ("subscript-based") constructs per se in IMPL's foreign-modeling, as such foreign-models may not be as expressively or elegantly formed as those formulations found in algebraic modeling / scripting languages (AML's / ASL's), subsets of indexes / indices (index-sets, compound-sets) are available to iterate, traverse or loop over sparse non-zeros (cf. the REPEAT() and REPLICATE"" model function / formulary start;stop;stride; tuple iterators). Essentially, scalar-based and somewhat sparsely-formed or simple set-based models do not use multi-dimensional array subscripts

nor objects and are generally easier to understand but admittedly they are less sophisticated and less expressive and elegant for building or constructing larger and more complicated models. To that end and although ILP and INP foreign-models are perhaps unconventional or out of the ordinary in their syntax, they therefore serve to complement IMPL's IML, IPL and IMPC and to enable quick prototyping / proof-of-concept / piloting of adhoc, bespoke or custom constraints and/or variables using these scalar-based / sparsely-formed / simple set-based foreign-models which can still be employed to model industrially relevant estimization (estimation with constraints) and optimization problems especially when their formulations can be systematically and automatically generated by some other program or sub-program written in third-party computer programming and scripting languages.

Additionally and as allueded to above, the ILP / INP scalar-based foreign-file models are by their nature more dense-based in terms of their form and formatting whereas set-based, stream-based and IMPL's structure- and semantic-based models as purposefully, inherently and usefully sparse-based. The term sparse means thinly scattered or distributed in form which implies for instance that not all units can process all operations and not all unit-operation out-port-states are connected to every available unit-operation in-port-states. Nevertheless, for smaller types of industrial optimization and estimization problems, certain formulations such as our USN-FSFS© / UFS© / USF© may be sufficiently represented using more dense data structures i.e., IMPL's calc-scalars, data-vectors and data-groups which may be adequately modeled using ILP's / INP's scalar-based and somewhat sparsely-formed / simple set-based foreign-models through the foreign-variables and -constraints REPEAT() model function and REPLICATE"" model formulary found further below (see also IML's SPIN"" data formulary).

It is important to note that when referencing the existing or standard IMPL non-foreign variables, the *original* variable's index number before IMPL's presolve is invoked (cf. the IMPLshrinkability() and IMPLshrinkability2() routines) must be configured or specified (i.e., the nnn index / indice in Xnnn). Admittedly, the usefulness of including existing IMPL standard variables into new foreign-constraints and updating or modifying their objective function weights and lower and upper hard bounds in the foreign-model is limited as these index numbers must be previously known by reviewing the contents of the *.dtv file. That is, if the non-foreign variable's indice changes for whatever reason due to any change in the problem's model and/or cycle data, then its indice in the foreign-file will be incorrect. This however is not the case for adhoc, custom or user linear / logic / logistics constraints and unit-operation condition (coefficient) formulas or expressions which should be considered instead. The

primary purpose of including the non-foreign or standard IMPL variables is for fast and effective prototyping and proving the utility of augmenting auxiliary constraints. Fortunately further on in this manual under the NRENAME / NMONIKER sub-section, the capability to more generally configure or specify an existing IMPL standard or non-foreign variable is described without the requirement to literally hard-code the standard / non-foreign variable's original index number.

For interest, IMPL is structure- and semantic-based whereas AML's / ASL's (Algebraic Modeling / Scripting Language) are set-based such as AIMMS, AMPL, GAMS, LINGO, LPL, MOSEL, MPL, OPL, ZIMPL, Julia JuMP, Matlab CVX, Python PYOMO / PuLP / PyOptInterface, CasADi, etc. and Matlab, Mathematica, Octave, Python SciPy and the R language are more scalar-based but using the power of vector-, matrix- and even tensor-processing. There are also "stream-based" models found in the process systems engineering (PSE) domain which are used for first-principles, physics-/chemistry-based, theoretical, mechanistic, (hard) engineering, white-/grey-box, exact or rigorous process simulation, estimation and optimization problems such as ASCEND, Aspen-Plus and Aspen-Hysys, Siemens' gPROMS, Honeywell's UniSim Design, Yokogawa/KBC's PetroSIM, Modelica, APMonitor, etc.

Industrial Linear/Logic/Logical/Logistics Programming Language (ILP)

The ILP foreign-file takes its inspiration from the well-known CPLEX LP file format which has four (4) sections demarcated by the following keywords: **"Objective", "Constraints", "Bounds"** and the optional **"Binaries"** and is blank space (" ") character (ASCII code 32) delimited. Comments can be specified using the IMPL standard "!" exclamation mark or the "\" blackslash character and any characters after the comment character will be ignored. There is a final **"End"** keyword that must be specified where all lines after it will be ignored. In addition, the **"#if nnn | nnn2"** and **"#endif"** directive statements are respected for both the ILP and INP foreign-files using the furcate flag / selective signal identical to their use in the IML and OML files where nnn and nnn2 may be calc-scalar and/or data-vector-element expressions or formulas.

The objective function terms are *maximized* only as is consistent with IMPL where there is only one objective function constraint which is non-binding, and as mentioned previously, the foreign-variable names must be labeled first with an upper case **"X"** prefix and foreign-constraint names must be identified first with an upper case **"F"** prefix with positive integers or whole numbers for the rest of

variable and constraint name characters with no spaces, underscores, dashes, etc. allowed except for a "." (dot or period) character at the end of the index or indice number as described later. If any of the "X" numbers are less than or equal to the number of variables in the non-foreign model, then these variables will be assumed to be the non-foreign or standard variables with their "original" (and before or prior to IMPL's presolve) numbering or indicing structure as found in the fact.dtv / subject.dtv file (as opposed to the IMPL presolved "optimizable or organized" numbering). If the "X" numbers are greater than the number of variables in the non-foreign or standard model, then these variables are considered to be new (foreign) and will be created with lower and upper bounds specified in the ILP bounds section. Since IMPL always creates a hand-full of non-foreign-variables and -constraints i.e., specifically for the profit, performance (1-norm and 2-norm), penalty and total objective function terms, it is recommended to start the "Xnnn" and "Fmmm" indices at ten (10) or greater where the maximum index number is 2^31 = 2,147,483,648 due to the four-byte integer number limit.

All coefficient values in the ILP file, whether in the objective function or the constraint section, must be a floating-point numerical constant, calc-scalar or data-vector-element and must not contain any mathematical operators or expressions except for the starting, beginning or initial "-" (minus) or "+" (plus) characters which are required to separate or demarcate the coefficient and its variable as shown in the ILP foreign-file example below. If multiple instances of a non-foreign or foreign-variable are found within the objective function section, then the objective function weights are revised or updated only and are not incrementally summed for example. There must be a "|" pipe character directly after the linear constraint number and this indicates the start or begin of the constraint expression or formula which defines the linear constraint definition. Only the characters "=", "<=" and >=" are used to represent the equal to, less than or equal to and greater than or equal to constraints respectively where "=<" and "=>" are not recognized. Although the greater than or equal to (">=") constraint inequality is supported in IMPL, it is not as well used and tested as the "=" and "<=" constraints. However in the rare chance there is an issue, the user, modeler or analyst may easily convert ">=" to "<=" constraint senses or types by simply multiplying both the sides of the "<=" by minus one (-1.0) without loss of generality. After these constraint type or sense characters, only a single floating point real number, calc-scalar or data-vector-element is expected, including expressions thereof, which represents the right-hand-side (R.H.S.) of the constraint i.e., considered as a base, balance or bias. The objective function section ends when a constraints section is encountered, and the constraint section ends when the mandatory or obligatory bounds section is detected. Continuation features, rows or lines (without any continuation

character or symbol) are available in ILP foreign-files only which conveniently support incrementally building up the linear constraint terms by repeating the linear constraint name (i.e., “Fmmm|”) on each line where only the last line of the constraint may contain the “=”, “<=” or “>=” characters and the R.H.S. value or calc-scalar / data-vector-element expression. For example, if we have a linear foreign-constraint modeled as: “F100| X101 + X102 – X103 + X104 – X105 = 0.0”, then it may be incrementally, consecutively or contiguously separated and formulated on five (5) separate features, lines or rows as follows:

```
F100|  X101
F100|  X102
F100| -X103
F100|  X104
F100| -X105 = 0.0
```

In addition, both ILP and INP foreign-files support the ability to have expressions or formulas for the foreign-constraint index or indice numbers only (i.e., unfortunately this capability is not supported or available for foreign-variables) which are found or located after the “F” upper case character and before the “|” (pipe symbol) or “#” (hash symbol or number sign). This capability allows for the constraint integer indice to be calculated using calc-scalars and data-vector-elements such as “F100 + 10/1|” which simply equates to “F110”. However as mentioned, the same capability does not apply to foreign-variables (Xnnn) as these mathematical programming entities, constructs, objects or terms are themselves involved in algebraic expressions and formulas inside the foreign-constraints which would confound the index or indice expression.

It is important to again highlight that all variables in the ILPet / ILP foreign-files must start or begin with the prefix “X” and must finish or end with the space (“ “) or blank (ASCII code 32) character else they will not be properly recognized during the foreign-file’s lexing and parsing operations. The finishing or ending blank space character is not needed for foreign-variables involved in nonlinear constraints (cf. “#”) found in the INPet / INP foreign-file as INPet / INP lexes and parses the nonlinear constraint expressions or formulas using a more sophisticated and smarter compiler i.e., the same compiler used for the IMPL properties- and conditions-formulas and their macros. The blank space termination is not required for the foreign-constraints as they have dedicated linear and nonlinear constraint delimiters “|” and “#” respectively which clearly demarcates or terminates the end or finish of the foreign-constraint index number, index expression or string name.

The bounds section is obligatory and must provide both a lower and an upper bound per variable on one (1) line using the "<=" characters as shown in the ILP File Example below where the lower and upper bound values may be a numerical constant, calc-scalar or data-vector-element with either a "+" or "-" prefix including calc-scalar / data-vector-element expressions or formulas. Every foreign-variable including discrete (binary and integer) variables found in the problem must have both lower and upper bounds configured else its default bounds in IMPL are set as zero (0.0) and zero (0.0) respectively unless it is found in the objection section where the default bounds are zero (0.0) and infinity (INFIN).

The binaries section simply declares, defines or configures binary and integer (discrete) variables which may have multiple variables separated along with other discrete variables using the blank space (" ") character on the same line, row or feature or singularily configured on separate lines. Integer variables may also be configured when their lower and upper bounds do not lie between zero (0.0) and one (1.0). If a binary variable is not explicitly bounded in the bounds section but is defaulted as a variable in the objective section i.e., as previously mentioned lies between the defaults of zero (0.0) and infinity (INFIN) and is found explicitly in the binaries section, then the variable is assumed to be binary.

It is important to comment that if any calc-scalars and/or data-vectors are employed to configure the objective function weights, constraint coefficients and right-hand-sides, lower and upper variable bounds, etc., then their names must not contain any upper case "X"'s anywhere in their names as these will be falsely recognized as foreign-variable names and the foreign-files will not be recognizable and properly modeled by IMPL. However, calculation functions such as EXP, MAX, MXL, XOR and XFCN are exempt in terms of having a capital or upper case "X" character as these functions are known to the IMPL compiler. Another important configuration aspect is the capability for objective function weights, constraint coefficients, constraint right-hand-side constants, lower and upper hard bounds or limits and initial-values to include data-vector's with element index number expressions or formulas. This allows the same data-vector to be used with different element number indexing and respected by both ILP and INP foreign-files i.e., DV[100 + 10/1] = DV[110].

ILP (or the Pre-ILP ILPet) Foreign-File Example

This small example has no existing, standard or non-foreign model and has an objective function value of 8.0 at the optimal solution where X1000 = 1.0, X2000 = 1.0 and X3000 = 5.0 are the variable results or responses at the optimal solution.

```
! Comments …
\ Comments …

Objective

2.0 X3000 - 1 X2000 - X1000

Constraints

F1| X3000 – 3.0 X1000 - 2 X2000 = 0.0

Bounds

0.0 <= X3000 <= 10

Binaries

X1000 X2000

End
```

The user, modeler or analyst must be aware that it is their responsibility to ensure that for any of the linear constraint expressions, only one instance of each variable in each constraint formula or expression is expected. The reason is due to the fact that IMPL does not check nor handle duplicate or repeating sparsity-pattern non-zero records or entries (i.e., a single constraint has the same variable index found twice or more) specifically in these ILP files or in the INP files when the linear constraint indicator "|" is configured. In fact, all solvers have a different way of managing these repetition / duplication events, instances or occurrences such as summing the first-order partial derivative values or using the first found derivative value where both remedies or treatments may not be appropriate for any particular situation so that it is important for the user, modeler or analyst to properly consider these specific instances. Fortunately, this issue is not present for nonlinear constraints indicated by "#" described below as IMPL automatically determines the Jacobian or the first-order derivative matrix's sparsity-pattern where repeating or duplicate foreign-variable entries or involvement is handled properly.

Industrial Nonlinear Programming Language (INP)

The INP file has similar structure and syntax to the ILP file except that the "Binaries" section is replaced by an **"Initials"** section (optional) which is used to provide initial-values, starting-points or default-results for a specified variable where the "=" character is used to separate the initial-values and each variable must be on a separate line, row or feature similar to the "Bounds" section. The INP file allows both linear and nonlinear constraints where linear constraints are identical to those specified in the ILP file (cf. "|" or pipe symbol) and nonlinear constraints are demarcated by the "#" character immediately after the "F" and its constraint index number as seen in the INP File Examples below. The nonlinear constraints accept a constant R.H.S. identical to the ILP linear constraints but the left-hand-side (L.H.S.) constraint expressions of course must contain the necessary mathematical operators accepted by IMPL including numerical constants, known calc-scalars and data-vector-elements to property specify or configure the constraint as shown in the INP File Examples. It should be noted that there are no continuation lines in the INP file for the nonlinear constraints ("#"), but identical to the ILP foreign-files for linear constraints ("|"), continuation lines are available as described above and both unary minus and unary plus coefficient, parameter and weight terms are supported in all of the ILP / INP sections.

However, it is important to highlight that to avoid any unnecessary convergence issues when there is a mixture of linear and nonlinear constraints in the INP / INPet files, segregate all linear constraints before the nonlinear constraints. It is also possible to configure only nonlinear constraints even for linear constraints as IMPL will automatically detect nonlinear constraints that are inherently linear i.e., have constant values or parameters for all of its variables via IMPL Server's stationarity() routine to reduce the nonlinearity convergence overhead during the solving phase.

Both the ILP and INP foreign-files support diagonal quadratic or 2-norm deviation variable terms (i.e., "X" squared or times itself only) in the objective function indicated by the "^" (caret) character found immediately after the "X" variable where we have removed the need to specify a "2" after the "^" as other integer or fractional powers or exponents in the objective function are not supported / recognized in IMPL except for one i.e., "^1" which indicates a 1-norm deviation variable. Hence, any numerical value after the "^" symbol which is not one (1) or two (2) is ignored and defaults to two (2) as IMPL supports only 1-norm and 2-norm deviation variables. The coefficient value or numerical constant, calc-scalar or data-vector-element in front of the absolute / Manhattan / 1-norm or quadratic / Euclidean / 2-norm deviation variable is assumed to be what we call the 1-norm / 2-norm performance-weight where 2-norm weights are always entered internally as a negative number into the IMPL sub-model (i.e., to be

minimized when maximized so as to satisfy convexity of the objective function) so the configured objective function coefficient value sign for quadratic variables ("^" or "^2") is always ignored. This ensures that all 2-norm deviation variable terms are only minimized given that IMPL always and only maximizes the objective function and quadratic programming (QP) solvers require convexity in the objective function. However, the 1-norm deviation variables are minimized if negative (-ve) and technically may be maximized if positive (+ve) though maximizing the 1-norm deviations is not normally used as it means increasing the difference or delta from a target. As well, 1-norm deviations are usually configured in pairs to represent the plus- and minus-parts of a deviation from target where the plus and minus 1-norm deviations are mutually exclusive or complementary i.e., the product of their solution values should equal zero (0.0).

INP (or Pre-INP INPet) Foreign-File Examples

The first INP file example is an oil-refinery alkylation unit process optimization example, has no non-foreign / standard model except for the five (5) standard variables and constraints mentioned previously and has an objective function value of 1161.34 at the optimal solution where X10 = 1.73E+03, X20 = 1.60E+04, X30 = 9.82E+01, X40 = 2.00E+03, X50 = 3.06E+03, X60 = 1.04E+01, X70 = 9.06E+01, X80 = 2.62E+00, X90 = 9.42E+01, X100 = 1.50E+02 and X4070 = 2.89E+05 are the solution results.

```
! Comments …
\ Comments …

Objective

0.063 X4070 - 5.04 X10 - 0.035 X20  - 10.0 X30  - 3.36 X50

Constraints

F50| X50 - 1.22 X40 + X10 = 0
F90| X90 + 0.222 X100 = 35.82
F100| X100 - 3.0 X70 = -133.0

F40# X40 - (X10*(1.12 + 0.13167*X80 - 0.00667*X80^2.0)) = 0
F60# X60*(X40*X90 + 1000.0*X30) - 98*1000.0*X30 = 0
F70# X70 - (86.35 + 1.098*X80 - 0.038*X80^2.0 + 0.325*(X60 - 89.0)) = 0
F80# X80*X10 - (X20 + X50) = 0
F4070# X4070 - X40*X70 = 0
```

Bounds

```
1.0 <= X10 <= 2000.0
1.0 <= X20 <= 16000.0
1.0 <= X30 <= 120.0
1.0 <= X40 <= 5000.0
1.0 <= X50 <= 2000.0

85.0 <= X60 <= 93.0
90.0 <= X70 <= 95.0
3.0 <= X80 <= 12.0
0.01 <= X90 <= 4.0
145.0 <= X100 <= 162.0

1.0 <= X4070 <= 500000.0
```

Initials

```
X10 = 1745
X20 = 12000
X30 = 110
X40 = 3048
X50 = 1974
X60 = 89.2
X70 = 92.8
X80 = 8
X90 = 3.6
X100 = 145
X4070 = 10000
```

End

The second INP file example is a nonlinear data reconciliation example, also has no non-foreign model and has an objective function value of -0.1123 where X11 = 4.51E+00, X12 = 5.58E+00, X13 = 1.93E+00, X14 = 1.46E+00, X15 = 4.85E+00, X1001 = 1.11E+0, X1002 = 6.15E-01, X1003 = 2.05E+00, X101 = -1.12E-01, X102 = -8.19E-02, X103 = -2.26E-01, X104 = 1.44E-01, X105 = 1.45E-01.

Objective

```
1 X101^
1 X102^
1 X103^
1 X104^
1 X105^
```

Constraints

```
F101| X101 + X11 = 4.4
F102| X102 + X12 = 5.5
F103| X103 + X13 = 1.7
F104| X104 + X14 = 1.6
F105| X105 + X15 = 5.0

\ f(1) = 0.5*x(1)^2-0.7*x(2)+x(3)*y(1)+x(2)^2*y(1)*y(2)+2*x(3)*y(3)^2-255.8;

F11# 0.5*X11^2 - 0.7*X12 + X13*X1001 + X12^2*X1001*X1002 + 2*X13*X1003^2 - 255.8 = 0

\ f(2) = x(1)-2*x(2)+3*x(1)*x(3)-2*x(2)*y(1)-x(2)*y(2)*y(3)+111.2;

F12# X11 - 2*X12 + 3*X11*X13 - 2*X12*X1001 - X12*X1002*X1003 + 111.2 = 0

\ f(3) = x(3)*y(1)-x(1)+3*x(2)+x(1)*y(2)-x(3)*y(3)^(0.5)-33.57;

F13# X13*X1001 - X11 + 3*X12 + X11*X1002 - X13*X1003^0.5 - 33.57 = 0

\ f(4) = x(4)-x(1)-x(3)^2+y(2)+3*y(3);

F14# X14 - X11 - X13^2 + X1002 + 3*X1003 = 0

\ f(5) = x(5) - 2*x(3)*y(2)*y(3);

F15# X15 - 2*X13*X1002*X1003 = 0

\ f(6) = 2*x(1)+x(2)*x(3)*y(1)+y(2)-y(3)-126.6;

F16# 2*X11 + X12*X13*X1001 + X1002 - X1003 - 126.6 = 0
```

Bounds

```
0 <= X11 <= 10
0 <= X12 <= 10
0 <= X13 <= 10
0 <= X14 <= 10
0 <= X15 <= 10

0 <= X1001 <= 100
0 <= X1002 <= 100
0 <= X1003 <= 100

-10 <= X101 <= 10
-10 <= X102 <= 10
-10 <= X103 <= 10
-10 <= X104 <= 10
```

```
-10 <= X105 <= 10

Initials

\ xm = [4.4, 5.5, 1.7, 1.6, 5]T
\ y0 = [8, 0.7, 1.8]T

X11 = 4.4
X12 = 5.5
X13 = 1.7
X14 = 1.6
X15 = 5.0

X1001 = 8.0
X1002 = 0.7
X1003 = 1.8

X101 = 0.0
X102 = 0.0
X103 = 0.0
X104 = 0.0
X105 = 0.0

End
```

The third INP foreign-file example is actually a linear data reconciliation example but with side constraints. This problem also has no non-foreign model and has an optimal objective function value of -198.798. There are two (2) time-periods where we not only minimize the weighted sum-of-squares of residuals for the measurements (xa,t + x,t = xm,t) but we also minimize the expected random white noise signal i.e., x,t – x,t-1 – a,t = 0 using the immediate past value at t-1.

```
! Comments …
\ Comments …

Objective

25.0 X10011^
25.0 X10021^
25.0 X10031^
25.0 X10041^
25.0 X10051^
25.0 X10061^

25.0 X10012^
25.0 X10022^
```

```
25.0 X10032^
25.0 X10042^
25.0 X10052^
25.0 X10062^

25.0 X100011^
25.0 X100021^
25.0 X100031^
25.0 X100041^
25.0 X100051^
25.0 X100061^

25.0 X100012^
25.0 X100022^
25.0 X100032^
25.0 X100042^
25.0 X100052^
25.0 X100062^
```

Constraints

```
\ Structural balances: x1 - x2 = 0, x2 - x3 = 0, etc.

F111| X1011 - X1021 = 0
F121| X1021 - X1031 = 0
F131| X1031 - X1041 = 0
F141| X1041 - X1051 = 0
F151| X1051 - X1061 = 0

F112| X1012 - X1022 = 0
F122| X1022 - X1032 = 0
F132| X1032 - X1042 = 0
F142| X1042 - X1052 = 0
F152| X1052 - X1062 = 0

\ Sensor or measurement balances: xa + x = xm where xa is the adjustment, x is the reconciled
\                                          value and xm is the measurement or raw value.

F1011| X1011 + X10011 = -1.233323347
F1021| X1021 + X10021 = -1.31183673
F1031| X1031 + X10031 = 0.495999992
F1041| X1041 + X10041 = -1.287375032
F1051| X1051 + X10051 = -1.290653689
F1061| X1061 + X10061 = -1.30112312

F1012| X1012 + X10012 = -1.2330567
F1022| X1022 + X10022 = -1.312778805
F1032| X1032 + X10032 = 0.495999992
```

F1042| X1042 + X10042 = -1.28937503
F1052| X1052 + X10052 = -1.291695355
F1062| X1062 + X10062 = -1.29950829

\ Serial or temporal balances (random-walk noise): x,t - x,t-1 - a,t = 0

F10011| X1011 - X1010 - X100011 = 0.0
F10021| X1021 - X1020 - X100021 = 0.0
F10031| X1031 - X1030 - X100031 = 0.0
F10041| X1041 - X1040 - X100041 = 0.0
F10051| X1051 - X1050 - X100051 = 0.0
F10061| X1061 - X1060 - X100061 = 0.0

F10012| X1012 - X1011 - X100012 = 0.0
F10022| X1022 - X1021 - X100022 = 0.0
F10032| X1032 - X1031 - X100032 = 0.0
F10042| X1042 - X1041 - X100042 = 0.0
F10052| X1052 - X1051 - X100052 = 0.0
F10062| X1062 - X1061 - X100062 = 0.0

Bounds

-1.24615009 <= X1010 <= -1.24615009
-1.337912982 <= X1020 <= -1.337912982
0.495999992 <= X1030 <= 0.495999992
-1.286785601 <= X1040 <= -1.286785601
-1.274059053 <= X1050 <= -1.274059053
-1.296249815 <= X1060 <= -1.296249815

-100.0 <= X1011 <= 100.0
-100.0 <= X1021 <= 100.0
-100.0 <= X1031 <= 100.0
-100.0 <= X1041 <= 100.0
-100.0 <= X1051 <= 100.0
-100.0 <= X1061 <= 100.0

-100.0 <= X1012 <= 100.0
-100.0 <= X1022 <= 100.0
-100.0 <= X1032 <= 100.0
-100.0 <= X1042 <= 100.0
-100.0 <= X1052 <= 100.0
-100.0 <= X1062 <= 100.0

-100.0 <= X10011 <= 100.0
-100.0 <= X10021 <= 100.0
-100.0 <= X10031 <= 100.0
-100.0 <= X10041 <= 100.0
-100.0 <= X10051 <= 100.0

-100.0 <= X10061 <= 100.0

-100.0 <= X10012 <= 100.0
-100.0 <= X10022 <= 100.0
-100.0 <= X10032 <= 100.0
-100.0 <= X10042 <= 100.0
-100.0 <= X10052 <= 100.0
-100.0 <= X10062 <= 100.0

-100.0 <= X100011 <= 100.0
-100.0 <= X100021 <= 100.0
-100.0 <= X100031 <= 100.0
-100.0 <= X100041 <= 100.0
-100.0 <= X100051 <= 100.0
-100.0 <= X100061 <= 100.0

-100.0 <= X100012 <= 100.0
-100.0 <= X100022 <= 100.0
-100.0 <= X100032 <= 100.0
-100.0 <= X100042 <= 100.0
-100.0 <= X100052 <= 100.0
-100.0 <= X100062 <= 100.0

Initials

\ xm

X1011 = -1.233323347
X1021 = -1.31183673
X1031 = 0.495999992
X1041 = -1.287375032
X1051 = -1.290653689
X1061 = -1.30112312

X1012 = -1.2330567
X1022 = -1.312778805
X1032 = 0.495999992
X1042 = -1.28937503
X1052 = -1.291695355
X1062 = -1.29950829

\ xa

X10011 = 0.0
X10021 = 0.0
X10031 = 0.0
X10041 = 0.0
X10051 = 0.0

```
X10061 = 0.0

X10012 = 0.0
X10022 = 0.0
X10032 = 0.0
X10042 = 0.0
X10052 = 0.0
X10062 = 0.0

\ a,t

X100011 = 0.0
X100021 = 0.0
X100031 = 0.0
X100041 = 0.0
X100051 = 0.0
X100061 = 0.0

X100012 = 0.0
X100022 = 0.0
X100032 = 0.0
X100042 = 0.0
X100052 = 0.0
X100062 = 0.0

End
```

Please refer to the OML Manual to output, export, print or write the foreign-variable and foreign-constraint real values with the data reconciliation vetistics or diagnostic statistics when required where all foreign-variables and -constraints are exported to the problem's / sub-problem's EXL file.

ILPet and INPet Foreign-Files (Optional Pre-ILP and Pre-INP)

Although only one ILP or one INP foreign-file is supported for each problem or sub-problem depending on whether it is a quantity (LP, QP), logistics (MILP, MIQP) or quality (NLP) optimization or estimization problem, if an ILPet or INPet foreign-file is found to exist, then it will be read, inputted or imported *before* the ILP or INP file is read, loaded or processed. This allows any problem formulation or model to have essentially two (2) ILP or INP foreign-files that may be processed per problem or sub-problem and is useful to split or separate some of the foreign-variables and foreign-constraints and to spread them across the two (2) (ILPet / ILP or INPet / INP) files per problem when necessary. The ILPet and

INPet files are required to have exactly the same format structure as their ILP and INP counter-parts i.e., the obligatory or mandatory `Objective, Constraints, Bounds` and optional `Binaries` and/or `Initials` frames or sections with the final and mandatory / obligatory `End` keyword. **The only restrictions are that a foreign-constraint added, created or declared in the ILPet / INPet file may not be altered, updated or modified in the ILP / INP file although foreign-variable objective weights, lower and upper bounds, variable types or senses (continuous, binary or integer) and initial-values may be created and/or modifed / updated.**

The ILPet then ILP or INPet then INP foreign-file processing sequence or order is useful when an automated higher-level program or procedure is employed to create or generate a "standardized" pre-model or -formulation foreign-file (ILPet / INPet) first, then to augment, alter and/or add-on to the model or formulation second using the ILP or INP foreign-files in some incremental user-defined, bespoke or customized fashion. Two related examples of these are IMPL-DATA's DREC and DREG modeling paradigms for discrete, nonlinear and dynamic (DND) estimation.

NREPEATS / REPEAT() and REPLICATE"" Model Functions / Formulary's

The model function **REPEAT(start;stop;step; $string$)** simply repeats the repeated **$string$** expression multiple times by replacing, substituting or swapping the special **"$" (dollar)** symbol character in the $string$ argument of length **4096** characters with the incremented index number, with leading-zeros ("0"'s), defined by the range of its **start, stop** and **step** (or stride) parameters and REPEAT()'s support the tick or grave accent special character ("`", ASCII code 96) enabling continuation lines, rows or features identical to the IMPL Interfacer's support for calc-scalars and property / condtion marcos. The REPEAT() model function is a utility routine to help the user, modeler or analyst avoid manually adding and editing ILP / INP modeling statements which may inject spurious and needless copy and paste errors, etc. The ILP / INP foreign-files are expected to have one or more instances of the REPEAT() model function to be present or exist for any foreign-model but if none are present, then the user, modeler or analyst may configure the keyword **"NREPEATS = 0" in the very first feature, line or row of the foreign-file to fore go any processing of the REPEAT() statements.** Unless NREPEATS = 0 in the first feature, IMPL will search for any REPEAT() routine occurrences and replace the special "$" ("&", "@") symbol(s) with a leading-zero incremented index number when the integer indice has less digits than the digits defined by the stop argument. For example, if the stop argument arbitrarily equals 1000 which has four-digits,

then the index numbers 1..9 will be systematically repeated as 0001..0009, 10..99 as 0010..0099 and 100..999 as 0100..0999 accordingly – see also the IMPL Server routine IMPLlzs(). The step or stride argument is provided to step the incrementing index number by a certain fixed amount of stride, step or increment. For instance, if start = 3, stop = 10 and step = 2, then the incremented index numbers over the specified range with leading-zeros (0's) is 03, 05, 07, 09. If step is negative (-ve), then the sequence or series starts at stop and increments downward or in descending order as 10, 08, 06, 04. Unfortunately as IMPL's scalar-based foreign-models are not as sophisticated (i.e., as mathematically, symbolically or algebraically expressive and elegant) as the well-known set-based algebraic modeling and scripting languages (AML's and ASL's), index number conditional, logical and relational statements, clauses or Boolean expressions after the start;step;stop iterator tuple such as "subject to", "such that", "filter by", "when", "where", etc. are not available. For more expressive and complicated mathematical, symbolic or algebraic modeling, we encourage the routines IMPLreceiveX(), IMPLreceiveX2(), IMPLreceiveFLX(), etc. which can be coded in any computer programming or scripting languages that can call or invoke dynamic link libraries / shared objects.

The REPEAT() model function also respects the special character sequence **"$%nnn%"** where nnn is the lead (+nnn) / lag (-nnn) positive or negative integer index-shift / -offset / -bias number respectively and **"%"** is the percent symbol special character. The lead / lag index simply adds or subtracts from the internal integer number value of "$" when realized in the repeated features, lines or rows in the ILP or INP foreign-files and is useful to offset, shift or bias any index number. For example, if "$%-1%" found anywhere inside the repeat string, to be substituted or swapped with the incremented index number, then minus one (-1) is added to the indice in this instance before it replaces the sub-string. This is useful when modeling dynamic problems or sub-problems with leading or lagging indexes such as x,t – x,t-1 i.e., x[$] – x[$%-1%] in recursive difference equations where these index-shifts / -offsets / -biases may also be calc-scalar and data-vector-element expressions or formulas.

All of the repeated features, lines or rows are outputted, written, printed or exported to the intermediate **ILPxf1 / INPxf1** foreign-files when the REPEAT() / REPLICATE"" model functions / formualry's exist which are similar to the **ILPxf2 / INPxf2** intermediate foreign-files when the NRENAMES / NMONIKERS keyword is set and the convenient moniker or renames that exist are found and replaced. The reason for the one (1) after the "xf" file type/extension suffix is due to the fact that the REPEAT() / REPLICATE"" model functions / formulary's are always processed by IMPL first before the renaming of

monikers are handled. **It is important to note that unless the NREPEATS = 0 is found in the very first line, row or feature, then the ILPxf1 / INPxf1 will always be output, exported, printed or written for debugging and troubleshooting of the foreign-model for the convenience of the user, model or analyst. However, if NREPEATS = 0 and REPEAT() model function instances do exist as well as the REPLICATE"" model formulary, then errors or exceptions will occur of course as any found or encountered REPEAT() model functions / REPLICATE"" model formularies will not be recognizable or interpretable by IMPL.**

The **REPEAT2(start;stop;step; start2;stop2;step2; $&string$&)** and the **REPEAT3(start;stop;step; start2;stop2;step2; start3;stop3;step3; $&@string$&@)** model functions are supported which allow for two (2) and three (3) indexing sets, subscripts, dimensions or ranks to be recognized using the special characters **"&" (ampersand, and)** and **"@" (asperand, each, at)** respectively where the **"&%nnn%"** and **"@%nnn%"** combinations are also respected as index- / indice-shifts. The first start;stop;step tuple with "$" is the outer loop, the second tuple with "&" is the middle outer or inner loop and the third tuple with "@" is the inner most loop. As usual, the inner loops are incremented first, then the outer loops second and so on.

The **REPLICATE"start;stop;step; ?string?", REPLICATE2"start;stop;step; start2;stop2;step2; ?~string?~"** and **REPLICATE3"start;stop;step; start2;stop2;step2; start3;stop3;step3; ?~:string?~:"** model formulary are supported which allow the user, modeler or analyst to replicate a sub-string *within* a REPEAT() model function only using the special characters **"?" (question mark), "~" (tilde)** and **":" (colon)** where the double-quotes **""** are used as delimiters instead of the open and closed parentheses () used for the REPEAT() model functions. For the REPLICATE"" model formulary's, the "%nnn%" special symbols for lead (+nnn) / lag (-nnn) index-shift operators are also supported. In addition, for the REPLICATE"" model formulary, "?%%nnn%", "~%%nnn% "and ":%%nnn%" are provided with a second (2^{nd}) leading or prefixed "%" special character to negatively add or subtract the "?", "~" and ":" number replacements from the literal, calc-scalar or data-vector-element expression nnn i.e., -? + nnn, -~ + nnn and -: + nnn. The REPLICATE"" model formulary provides the basic capability to perform for example what are known as in-line sum, product, etc. primitives with upto three (3) dimensions, subscripts or indexes and muliple REPLICATE"" model formulary instances may be located inside a single REPEAT() model function string. Again, please be aware of the **4096** character length limit for all features, lines or rows configured in the

ILP / INP foreign-files and it is easy to now systematically generate arbitrarily long foreign-constraint expressions or formulas.

In addition, both the REPEAT() model functions and the REPLICATE"" model formulary's support dense iterator-triples i.e., the **start;stop;step;** but also sparse iterator-sets, -lists or -vectors i.e., **set;span;** otherwise known as index-sets, -lists or -vectors using a supplied data-set, -list or -vector. The set; must be a known data-set, -list or -vector (converted to integer numbers) and the span; is required to properly prefix, pad or frontend load the appropriate number of zeros (0's) for the leading-zero indexing similar to stop; and can be likened to a set's / list's / vector's maximum domain or range of elements / members also referred to as a reference index number. Any combination of dense- and sparse-iterators are supported with the REPEAT2() and REPEAT3() model functions and REPLICATE2"" and REPLICATE3"" model formulary's.

NREPLACEMENTS Keywords for ILP / INP Foreign-Files

The keywords **"NREPLACEMENTS1 = n1"** and **"NREPLACEMENTS2 = n2"** must be configured after the NREPEATS = n keyword statement and before or prior to the NRENAMES / NMONIKERS and NREWINDS keywords as well as before the Objective (function) section where "n1" and "n2" are the maximum numbers of (internal) replacements or sub-string substitutions dynamically allocated in IMPL and must be greater than or equal to the actual number of replacements processes or found. NREPLACEMENTS1 are for replacements that are processed by IMPL along with the REPEAT() / REPLICATE"" model functions and formularies and the NREPLACEMENTS2 are those replacements that are processed along with the renames or monikers where NREPLACEMENT1 = n1 and its replacements must precede the NREPLACEMENTS2 = n2 keyword.

Replacements have left-hand-side (LHS) and right-hand-side (RHS) string sizes or lengths of basestringlen (64 characters) and linestringlen (64 * 64 = **4096** characters) respectively and are similar to renames / monikers as the LHS's and RHS's are separated by the equals sign character ("=") i.e., LHS short replacement string or name = RHS long replacement string or formula / expression fragment. Replacements are helpful to simply replace, swap or substitute a shorter string length with a longer string size found, located, positioned or encountered anywhere in the foreign-files. Replacements however are not respected i.e., are ignored, skipped or passed-over when NREPEATS = 0. Ultimately,

the primary purpose or raison d'etre of replacements is to aid the user, modeler or analyst when mathematically coding or programming their foreign-constraint formulas and expressions to be more human-readable if possible especially when they are relatively complex or convoluted i.e., have many terms or groups of variables.

The first set of replacements will appear in the ILPxf1 / INPxf1 intermediate foreign-files and the second set of replacements will be present in the ILPxf2's / INPxf2's where no replacement assignments, associations or attachements will be found, echoed or carried through in the intermediate foreign-files i.e., they are stripped or removed from them. **A restriction with these replacements is that they must involve or include foreign-variables (and also foreign-intermediate-variables / FIV's) only as recursive, nested, embedded or cascaded replacements are not supported i.e., a LHS replacement short string must not exist in any RHS replacement long strings.**

NRENAMES (/ NMONIKERS) Keywords for ILP / INP Foreign-Files

The keywords **"NRENAMES = n"** (or **"NMONIKERS = n")** must be configured before or prior to the Objective (function) section (similar to the NREPEATS and NREWINDS keywords) where "n" is the maximum number of (internal) monikers, short-names, nick-names, renames or a.k.a.'s dynamically allocated in IMPL. Renames (or monikers) are human readable user-, modeler- or analyst-supplied or provided alternative string names for both foreign-variables and -constraints of upto sixty-four (64) characters long which are configured immediately before the Objective section in the ILP or INP foreign-file as opposed to the "Xnnn" and "Fmmm" mnemonic lengths of only eleven (11) characters i.e., X2147483648 / F2147483648 for instance restricted by the four-byte integer number size or length. The left-hand-side (L.H.S.) of the equal sign "=" is used to declare or define the rename (or moniker) to be renamed with the right-hand-side (R.H.S.) being required to specify the "Xnnn" foreign-variable or the "Fmmm" foreign-constraint mnemonics where the "nnn" / "mmm" may be configured as a numerical literal or more conveniently as a useful index number expression or formula involving known calc-scalars and/or data-vector-elements configured in a corresponding IML file or IPL functions (i.e., IMPLreceiveCalc(), IMPLrunCalc() and IMPLreceiveData() routines).

When NRENAMES (/ NMONIKERS) is greater than zero (0), a special ***.ILPetxf2 / *.ILPxf2** or ***.INPetxf2 / *.INPxf2** foreign-file is written containing all "Xnnn" and "Fmmm" which simply exposes to the user,

modeler or analyst the internal conversion of the L.H.S. moniker renames to the R.H.S. foreign-variables and -constraints. Of course, the "xf" suffix or appendage to the file extension / type stands for the "x" in "Xnnn" and "f" in "Fmmm". Ultimately, the NRENAMES (/ NMONIKERS) keyword and its accompanying L.H.S. / R.H.S. rename (/ moniker) instances are intended to make the scalar-based foreign-file modeling hopefully more human readable and understandable as well as potentially less prone to modeling errors and/or inconsistencies similar to defined named ranges in Microsoft Excel for what IMPL calls calc-scalars, data-vectors and data-vector-groups. With respect to how IMPL processes these renames (or monikers), every feature, line or row of the foreign-file is searched and if the rename (/ moniker) character sequence is detected, then it is simply replaced by the Xnnn or Fmmm specified where multiple instances of the same rename (/ moniker) are also replaced as expected.

The only restriction regarding the configuration of the renames (or monikers) is the necessity that any rename or moniker that is a sub-string of other renames (/ monikers) must be configured after these renames (or monikers). This requirement ensures that IMPL's linear searched, find-and-replace technique will properly substitute sub-string renames last. If by-chance the L.H.S. rename sub-string pre-ordering or pre-sequencing is not performed by the user, modeler or analyst, then an error or exception will occur when the foreign-file is lexed and parsed (compiled) as part of the modeling stage or phase of IMPL. This error will then be indicated or flagged, as usual, with the foreign-file's line number that is not processable or recognizable. In terms of debugging and troubleshooting the foreign-files with renamess, if any errors or exceptions due occur, then the first place to start is to ensure that in the replaced renames section of the *.ILPetxf2 / *.ILPxf2 and *.INPetxf2 / *.INPxf2 files that both the left-hand- and right-hand-sides are identical. If they are not, then left-hand-side renames were not properly replaced and need to be modified accordingly.

For repetitive, recursive, real-time or on-line application with monikers or renames, and to reduce the foreign-file pre-processing time i.e., bypassing the find-and-replace operations of the moniker renaming with Xnnn and Fmmm, simply rename the *.ILPetxf2 / *.ILPxf2 or *.INPetxf2 / *.INPxf2 to *.ILPet / *.ILP and *.INPet / *.INP and remove the monikers sections in their respective foreign-files. Since all of the monikers are replaced in the "xf2" suffixed foreign-files, there is no requirement for the extra find-and-replace of monikers especially when there is a distinct separation of its model from its data i.e., ILP / INP foreign-files and IML files respectively.

In order to support the ability to add, create, generate or insert new foreign-constraints with existing IMPL standard or non-foreign-variables known as IMPL's UOPSS-QLQP© foundation or framework (UQF©), IMPL provides the interesting and unique capability to configure the right-hand-side (R.H.S.) X foreign-variable with a suffixed variable name string in the form of – notice the parentheses "()" and curly brackets "{}" respectively:

Xvarname(numkey1expr, numkey2expr, numkey3expr, …, numkey8expr) or

Xvarname{numkey1expr, numkey2expr, numkey3expr, …, numkey8expr}

The "*varname*" must match a known UQF© variable name referred to in IMPL as a variable resource-entity roster-enumator name (cf. IMPL-QLQ.imv, *.dtv, *.ddt or *.ndt) where upto eight (8) key arguments are allowed each being either a literal integer number or a calc-scalar / data-vector-element expression or formula. Please note the comma "," delimiters and the open and closing parentheses "()" which implies that if expression term grouping is required for the key arguments, then the curly braces "{}" must be used instead of the "()" else an error will occur during the foreign-file processing. Ultimately the *"varname(…, …, …)*" string should resolve to a known IMPL standard / non-foreign original variable (i.e., before IMPL's presolving or shrinkability) whereby it is automatically replaced by nnn thus Xnnn is formed and used throughout the rest of the foreign-file. The IML datacalc functions uU(), oO(), …, aA(), lL() can be employed to supply and specify the key number arguments of the standard IMPL variable either as calc-scalars or data-vector-elements. Note that internally, IMPL calls or invokes the IMPL Server's IMPLrow() routine to determine the original number index or indice of the existing standard or non-foreign variable.

The above modeling functionality of above for IMPL standard or non-foreign variables is also available for standard / non-foreign constraint names in a limited way as described further (cf. IMPL-QLQ.imc, *.dtc, *.ddt or *.ndt). After the standard constraint is stated in the NRENAMES section identical to the non-foreign variables above, if the standard constraint is found in the Constraints section with either a "|" or "#" ILP / INP foreign-file constraint delimiter, the non-foreign constraint will be removed, deleted, dropped, not included or excluded in the organized / optimizable problem's model i.e., the original problem's standard constraint is essentially hidden (i.e., ignored, skipped or passed-over) and hence is not known or present in the organized / optimizable problem. In addition, it is relevant to mention that the entire standard / non-foreign constraint with all of its time- / trial-periods (i.e., range-exhibit) is

excluded, thus only one (1) valid or proper time- / trial-period needs to be specified in the NRENAME section's statement to hide the said constraint in the organized / optimiable problem's model. For interest, the hiding of the original problem's standard / non-foreign constraint is invoked by calling IMPC's cretain(rcd,status=HIDE) and rcd = rowreference(type=CONSTRAINT,row) routines where the row argument is the constraint resource-entity's roster-enumerator, reference-event / record-entry, rack-evidence and range-exhibit's row-element. For interest, the concept of hiding a constraint is to simply hide, ignore or passover the constraint from being be known or transferred to the solver but it is always known to the modeler and is found in other algebraic modeling languages (AML's) such as MOSEL (XPRESS).

And finally it is worth exposing that if there are any leftover, un-renamed or un-replaced sub-strings in the constraint section of the ILP / INP foreign-files, then the dummy variable and dummy constraint symbols "X0" and "F0" respectively can be used as the right-hand-side (R.H.S.) for these un-resolved left-hand-side (L.H.S.) renames (or monikers). However, "X0" and "F0" must not be found or present in the objective, bounds, binaries and initials sections of the foreign-files else an exception or terminal error is raised.

NREWINDS Keyword for Dynamic Optimization and Data Estimization

The keyword **"NREWINDS = n"** must be configured before or prior to the Objective (function) section where "n" is the number of internal / inner rewinds, repeats or replications of the ILP or INP foreign-file. The default value for "n" is of course zero (0) and "n" may be either a numeric literal or a calc-scalar / data-vector-element expression. Immediately after the NREWINDS keyword and prior to the Objective section, any known or previously configured calc-scalars may be recursively, progressively or iteratively updated where the left-hand-side (L.H.S.) before the equal sign ("=") states the calc-scalar to be recursed, rolled or updated and the right-hand-side (R.H.S.) contains the calc-scalar expression which may include known data-vector-elements as well. These recursive calc-scalars are intended to be employed as data-vector indexes / indices found in the main body configuration of the Objective, Constraints, Bounds, Binaries and Initials sections with data-vector objective function weights, constraint parameters / coefficients and lower and upper variable bounds.

Please be reminded that all data-sets, -lists, -vectors, etc. start / begin / commence at index one (1) only and finish / end / stop only at the configured, defined or declared data-point, -row or -element range, size, span or spread when the data-vector was first created, generated or configured and notably, data-vector sizes or lengths (spreads) are immutable but their elements, rows or points may be updated using the SWAP(), SWAP2() and SUBSTITUTE() data functions.

Constraints involving only static / scalar coefficients or parameter variables may exclude the repeating or replicating by appending or suffixing a period or dot character "." immediately after the constraint's index number and before the constraint's type or sense symbol i.e., "F1001.# …". This will simply configure the constraint for the zeroth rewind, repeat or replication only. If the "." symbol is not applied, then duplicate or redundant constraints with the same foreign-variables involved repeatly will result in a more inefficient formulation. In addition, the period character "." immediately after the foreign-variables index number is also respected in the Bounds, Binaries and Initials sections of the foreign-files to negate the repeating or replication of these bounds, binary re-types / re-senses and initial- / starting-values**. Please note that foreign-constraints which have numerical expressions or formulas for its integer index numbers i.e., "F100 + 10/1#"**, **must not have any expression strings with periods ("."'s) representing literal real, double-precision or floating-point numbers as these will be mistaken for static constraints.**

The keyword NREWINDS is primarily intended to simply increment by one (1) any foreign-variable's / -constraint's index number providing a multi-time-period or multi-trial-period capability to the foreign-files for dynamic optimization and data regression. For example, if a problem's foreign-variables such as X101 are to incremented with multi-periods and NREWINDS = 99 – 1 (= 98), then IMPL will automatically or systematically generate X102, X103, X104, …, X199 where the X101 is created of course on the zeroth (0^{th}) rewind or the first read, input or import of the foreign-file.

Pre-Processing with REPEAT(), NRENAMES and NREWINDS

For clarification and transparency, the internal pre-processing when REPEAT() / REPLICATES"" model functions / formulary's, NRENAMES / NMONIKERS and NREWINDS keywords are present is as follows. First and if REPEAT() / REPLICATE"" model functions / formulary's are present, they are processed where all repeated string expressions with the "$", etc. special characters are simply replaced or substituted

with the incremental leading-zeros index numbers and are placed into the intermediate foreign-file with extension *.ILPetxf1 / *.ILPxf1 and *.INPetxf1 / *.INPxf1. Second and if renames / monikers are present i.e., renames are required, they are processed and placed into the intermediate foreign-file with extension *.ILPetxf2 / *.ILPxf2 and *.INPetxf2 / *.INPxf2 using either the original ILP / INP foreign-file or the first intermediate foreign-file (*.ILPetxf1 / *.ILPxf1 and *.INPetxf1 / *.INPxf1) if it exists. And third, if NREWIND is greater than zero (0), then each foreign-variable and foreign-constraint index numbers i.e., Xnnn and Fmmm are internally incremented by one (1) based on whether the intermediate foreign-file(s) exist or not. For instance, if ILPxf1 / INPxf1 exists but ILPet2 / INPxf2 does not, then the rewinds simply uses the ILPxf1 / INPxf1 intermediate foreign-file instead of the original ILP / INP foreign-file.

After the intermediate foreign-files have been processed, if the “xf1” and “xf2” file type / extension suffixes are removed by the user, modeler or analyst, they may conveniently replace the original foreign-files to reduce the foreign-file processing. This is especially useful for real-time or on-line industrial applications as the REPEAT() / REPLICATE”” model function / formulary and moniker rename processings may be eliminated reducing the modeling time of the foreign-models. However as mentioned, when NREWINDS is greater than zero (0), useful for (constrained) data regression and dynamic optimization and estimization problems, there are no rewind explicit intermediate foreign-files output, written, outputted or exported as the rewinding is performed internally without them; hence there are no corresponding *.ILPxf3 / *.INPxf3, etc. intermediate foreign-files.

Xname and Fname Instead of Xnum / Xnnn and Fnum / Fmmm

As previously described, the scalar-based foreign-model with its foreign-files ILPet / ILP and INPet / INP internally supports index numbers or indices for its foreign-variables (Xnnn, Xnum) and -constraints (Fmmm, Fnum) where nnn and mmm are literal integer numbers of four-bytes / 32-bits in length or size with a maximum or upper bound of 2^31 or 2,147,483,648. However, it is also possible to reference or identify Xnums / Xnnn’s and Fnums / Fmmm’s using Xnames and Fnames whereby the named character strings with a **maximum string length of 64 * 8 = 512 characters** are converted, transformed or mapped internally using the IMPL Server’s IMPLname2numX() and IMPLname2numF() routines respectively – see also their complementary routines IMPLnum2nameX() and IMPLnum2nameF(). Essentially, the string name to index number / indice is a straightforward method for IMPL to system generate indexes or indices from any user, modeler or analyst supplied character names. The use of Xname and Fname

instead of Xnum and Fnum is an attempt to help improve the configurability, clarity and convenience of scalar-based foreign-models.

It is important here to inform the user, modeler or analyst that no upper case or capital "X" can be placed anywhere in the foreign-variable name and no upper case or capital "F" can be located anywhere in a foreign-constraint as these special characters are strictly reserved to signal or indicate to IMPL that a foreign-variable or foreign-constraint has been encountered. Therefore as a straightforward and practical modeling convention, it is recommended to at least start the naming of all index-sets and parameters (i.e., calc-scalars and data-vectors) using upper case or capital letters and to exclusively use lower case or non-capital letters for all variables and constraints at least for the initial, starting or beginning names immediately after the "X" and "F" special foreign-variables and -constraint characters (cf. flatcase, PascalCase, camelCase, snake_case, camel_Snake_case, etc.).

The user, modeler or analyst has a limited freedom to choose the Xnames and Fnames with the following four (4) restrictions: 1) Xnames and Fnames must not have any embedded or in between spaces (**" "**) as a space is recognized as a lexing and parsing delimiter in all of the foreign-file sections, 2) Xnames must avoid configuring character sequences with **"^"**, **"^1"**, **"^2"** and **"."** which are reserved for the 1-norm and 2-norm deviation variables and static coefficients or parameters all specified in the objective function section, 3) Xnames and Fnames must not have any of the following special character mathematical operators: **"+"**, **"-"**, **"*"**, **"/"**, **"^"**, **"()"**, **"{}"**, **"[]"**, **";"** and **":"** given that foreign-variables may have calc-scalar/data-vector-element expressions for its coefficients and may be involved in nonlinear formulas and foreign-constraints support algebraic expressions or formulas within or in between the **"F"** character and its special character constraint delimiters **"|"** and **"#"** and 4) in light of the REPEAT() and REPLICATE"" model functions and formulary, the special characters **"$"**, **"&"**, **"@"** and **"?"**, **"~"**, **":"** and **"%"** although still may be configured if not involved in a REPEAT() / REPLICATE"", they should be chosen appropriately and cautiously**.**

Therefore to summarize for clarity and convenience, essentially the only two (2) non-letter or special characters that may be employed in valid foreign-variable and -constraint names are the well known and well used "_" (underscore, ASCII code 137) and "'" (single-quote, apostrophe, ASCII code 39). The underscore character ("_") may arguably be considered as the defacto universal identifier name and delimiter (cf. C, C++, Fortran, Python, etc.) and is also used by IMPL for instance to delimit the solution

number in the fact / subject file name e.g., "fact_nnn.exl" / "subject_nnn.exl". In addition, since the continuation feature, line or row tick special character "`" (ASCII code 96) and commas (",") are recognized in the IMPL compiler routine, hence the reason that ticks and commas must also be excluded from the list of special characters valid for Xnames and Fnames.

When the ILPet / ILP and INPet / INP foreign-files are processed (lexed and parsed) to determine the nnn in Xnnn and mmm in Fmmm, the nnn and mmm characters are first cast to a literal integer number. If this fails for any reason, then IMPL assumes these are Xnames and Fnames and automatically converts the name strings to the number indexes nnn and mmm where the nnn and mmm integers are system generated by IMPL. **Finally It should be emphasized that all foreign-variables and -constraints are always referenced internally and ultimately using a single integer index or indice number, however the Xname to Xnum / Xnnn and Fname to Fnum / Fmmm capability are available to hopefully enhance the readability, expressibility and maintainability of IMPL's scalar-based and somewhat sparely-formed / simple set-based foreign-model configurations.**

Foreign-Intermediate-Variables (Optional) in INPet/INP Foreign-Files

Optional foreign-intermediate-variables (FIV's) are available only in the INPet / INP foreign-files and allow the user, modeler or analyst to apply what are known as intermediate variables which are solely *dependent* on a single nonlinear equation or formula to calculate or compute their variable result, response or value at each NLP (quality) solver iteration. Its nonlinear equation (or equality constraint) must be a function of foreign-variables only as well as of course with literal real numbers, calc-scalars and data-vector-elements where no other foreign-intermediate-variables may be involved including itself else an error is raised i.e., no recursive or recurrent relations. Foreign-intermediate-variables are optional and must be defined or declared below its keyword **Intermediates** section after the **Initials** section in the INPet / INP foreign-files where the foreign-intermediate-variable left-hand-side (L.H.S.) is separated by the equals sign ("=") from its right-hand-side (R.H.S.) nonlinear foreign-intermediate-equation or formula-expression (e.g., foreign-intermediate-variable Xnnn or Xname = function of foreign-variables only). Foreign-intermediate-variables may be used to de-complexify (i.e., decrease the complexity) or simplify the appearance, structure and clarity of a nonlinear foreign-model by adding them to nonlinear foreign-constraints that would otherwise be too long and difficult to see and/or understand.

Foreign-intermediate-variables are inspired by the intermediate variables found in the open-use / community-based software process modeling system APMonitor and its Python module Gekko although we are unsure if these foreign-intermediate-variables play any noticeable role or effect on improving the solvability and stability of the nonlinear problem's convergence. Therefore, foreign-intermediate-variables are primarily available to help reduce the size or length of the nonlinear foreign-constraint's formula-expression string, having a limit of **4096** characters, by providing the user, modeler or analyst the ability to decompose or divide the nonlinear foreign-constraint into sub-foreign-constraint terms, equation-ettes or formula-fragments. It is also interesting to highlight that foreign-intermediate-variables are uniquely dependent on foreign-variables only represented by explicit or closed-form foreign-intermediate-equations and as such are not known to the solver (only the modeler), have no nonlinear convergence overhead of course, have no lower and upper hard bounds, have no target soft bounds, have no initial-values and have no objective function weights.

Three (3) important restrictions for these foreign-intermediate-variables are: 1) they must be found in nonlinear foreign constraints only i.e., foreign-constraints with the "#" (number or hash sign) delimiter and not the linear foreign-constraint delimiter "|" (pipe symbol). Since foreign-intermediate-variables are themselves expected to be nonlinear formula-expressions, they must not be found, included or involved in any foreign-constraint that is linear or not nonlinear else an error is raised; 2) as mentioned previously, there must be no embedding, nesting, cascading or recurrence of other foreign-intermediate-variables in the foreign-intermediate-equation else an error is raised. That is, only foreign-variables may be involved or included in a R.H.S. foreign-intermediate-variable's equation or formula-expression and not other foreign-intermediate-variables. See also the NREPLACEMENTS = n keyword which also has the same recursive or recurrent relation restriction but if violated no explicit error is raised; and 3) foreign-intermediate-variables are only supported when the form flag or structure$ signal is set to "sparsic" as the "symbolic" form is more complicated to implement whereas the sparsic form is simpler and more straightforward to apply. Fortunately, the sparsic form is also more efficient computationally than the symbolic form as more IMPL presolving can be performed – see the IMPL Server's stationarity(), separability2() and shrinkability2() routines. An error is raised in the IMPL Modeler if foreign-intermediate-variables exist or are present and the form flag / structure$ signal = "symbolic".

Another useful aspect of foreign-intermediate-variables, besides the fact that they can be used to de-complexify / simplify complex nonlinear foreign-constraint expression-formulas (L.H.S.), is that they may be formulated with calc-scalar and data-vector-element formulas or expressions which can be altered, changed, modified, updated, etc. from cycle-to-cycle, execution-to-execution, run-to-run of the IMPL Presolver with the same foreign-model. This use case is helpful when modeling and solving for example model predictive controller (MPC) optimization or moving horizon estimization (MHE) whereby the MPC / MHE foreign-model is immutable or static but the foreign-intermediate-variables are mutable and can be dynamically updated before the solving with the IMPL Presolver. However, since the internal IMPL compiler routine internally converts calc-scalars to constant or immutable literal numbers, calc-scalars cannot be updated from time-/trial-period-to-period but the data-vector-elements can be. Hence, any measurement feedback data such as opening, initial or starting data that changes from time-/trial-interval-to-interval, -step-to-step etc. must be configured as data-vector-elements.

MODELFCN() Model Function Callbacks – see DATAFCN()/WRITEFCN()

Similar to DATAFCN and WRITEFCN user-, modeler- or analyst-coded functions, also referred to as callbacks, MODELFCN's may also be configured and linked to MODELFC1, …, MODELFC9, MODELFCA, …, MODELFCZ whereby the user, modeler or analyst can compile their own C, C++ or Fortran DLL / SO dynamic / shared library functions and call them directly from IMPL in the ILP / INP foreign-files i.e., adhoc, bespoke, custom or user write functions. These MODELFCN functions may also be called from the computer programming or scripting language that calls IPL (IMPL-API) as well since they are regular C / C++ / Fortran DLL's or SO's. Essentially, the MODELFCN may be used to add, create, insert, modify, etc. foreign-variables and foreign-constraints as well as adding or inserting their respective first-order derivatives and nonlinear expression-formulas.

```
integer function MODELFCN(ncse: integer,
                          cse: real*ncs,
                          ndv: integer,
                          ndve: integer,
                          dve: real*(ndv*ndve),
                          ntse: integer,
                          csn: character(basestringlen)*ncs,
                          dvn: character(basestringlen)*ndv,
                          tsn: character(basestringlen)*ntse,
                          tse: character(basestringlen)*ntse)
```

```
       function modelfunc(ncse,cse,ndv,ndve,dve,ntse,csn,dvn,tsn,tse)
#if stdcalling == 1
cDEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE, DECORATE, ALIAS : "modelfunc" :: MODELFUNC
#else
cDEC$ ATTRIBUTES DLLEXPORT, ALIAS : "modelfunc" :: MODELFUNC
#endif

c * IMPORTANT NOTE * - IMPLserver.mod and IMPLserver.lib are optional.
c
c        use IMPLserver

        implicit none

        integer(4) :: modelfunc

c * IMPORTANT NOTE * - IMPLmodeler.fi is optional.
c
c        include "IMPLmodeler.fi"

c Number of calc-scalar elements (values), their names and the real number elements.

        integer(4), intent(in) :: ncse
cDEC$ ATTRIBUTES VALUE :: ncse
        real(8), intent(in) :: cse(1:ncse)
cDEC$ ATTRIBUTES REFERENCE :: cse

c Number of data-vectors, their names and the equal number of real number data-vector elements
c (values) for each data-vector in linearized or vectorized form.

        integer(4), intent(in) :: ndv
cDEC$ ATTRIBUTES VALUE :: ndv
        integer(4), intent(in) :: ndve
cDEC$ ATTRIBUTES VALUE :: ndve
        real(8), intent(in) :: dve(1:ndv*ndve)
cDEC$ ATTRIBUTES REFERENCE :: dve

c Number of text-string elements (values), their names and the character(BASESTRINGLEN=64) number
c elements.

         integer(4), intent(in) :: ntse
cDEC$ ATTRIBUTES VALUE :: ntse

        character(64), intent(in) :: csn(1:ncse)
cDEC$ ATTRIBUTES REFERENCE :: csn
        character(64), intent(in) :: dvn(1:ndv)
cDEC$ ATTRIBUTES REFERENCE :: dvn
        character(64), intent(in) :: tsn(1:ntse)
cDEC$ ATTRIBUTES REFERENCE :: tsn
        character(64), intent(in) :: tse(1:ntse)
cDEC$ ATTRIBUTES REFERENCE :: tse

c ... Insert developer user, modeler or analyst code here ...
c ...

c ...
c ... Insert developer user, modeler or analyst code here ...

        modelfunc = 0

c ... Insert developer user, modeler or analyst code here ...
c ...

c ...
c ... Insert developer user, modeler or analyst code here ...

      end function modelfunc
```

A very important and useful capability of the MODELFCN() callback's is that if programmed or coded with Intel Fortran (IFX), the IMPLserver.mod, IMPLserver.lib and IMPLmodeler.fi, provided by Industrial Algorithms Limited (IAL) upon request, may *optionally* be used and included when compiling and linking the MODELFCN() DLL / SO dynamic link library – see also the IMPC Manual. This means that the user, modeler or analyst has complete / full access to all of the IMPL data structures and routines / methods when ILP / INP is called or invoked by IMPL through the IMPL Server's IMPLreadilp() / IMPLreadinp() routines.

The ILP / INP model function declaration statement below requires the following feature, line or row to be configured first in the ILP / INP foreign-file before the **Objective** section only and specifies the dynamic or shared link library name (DLL / SO), the model function name contained within the library where the "@" asperand special character indicates the alphanumeric single character number of the model function i.e., **0** (MODELFCN), **1** (MODELFC1), …, **9** (MODELFC9), **10** (MODELFCA), …, **23** (MODELFCN), …, **35** (MODELFCZ).

```
DRIVE:/DIRECTORY/LIBNAME.DLL,FUNCNAME,@
```

Following the above ILP / INP statement above, where the backslash "\" character in the path name is not supported as backslash is a secondary comment character in the ILP / INP foreign-files, the MODELFCN() may then be called or invoked only within the **Binaries** (last) section of the ILP foreign-file and only within the **Intermediates** (last) section of the INP foreign-file as many times or instances as necessary where the text-string and text-group argument is optional and can be omitted.

```
MODELFCN(calc-scalar;data-vector)
MODELFCN(calc-scalar;data-vector;text-string)

MODELFCN(calc-group;data-group)
MODELFCN(calc-group;data-group;text-group)
```

The above call statements highlight that either a single calc-scalar or a calc-scalar-group, either a single data-vector or a data-vector-group or a either single text-string or a text-group in any combination may be specified or supplied where only the text-string / -group is optional.

Static Coefficient / Parameter Variances in Data Estimization

When modeling and solving multivariable linear / nonlinear, static / dynamic and constrained data regression problems, static / scalar coefficients or parameters are regressed, fit or estimated which also require their variance estimates and 95% confidence-intervals (margin-of-errors, regressed tolerance-range, etc.) to be computed based on its well-known Information matrix and the root mean square prediction error (RMSPE). Therefore, to declare, define or configure which foreign-variables are IMPL static coefficients / parameters, the user, modeler or analyst must add to the objective function section only a zero (0.0) coefficient value next to the X foreign-variable followed by the period **"."** character i.e., "0 X1001. 0 X1002." which in this case configures two (2) static coefficient / parameter foreign-variables X1001 and X1002. Any number of static coefficients may be identified to IMPL in the ILP / INP foreign-files using the "." period symbol in the objective function section and the zero (0.0) objective weight weight values will remove any contribution of the static coefficients / parameters to the objective function value (if required).

After the linear / nonliner data regression solving, a post-solve or -compute using **SPVE** (cf. IMPL Solvers' IMPLsertainty() routine) will compute the static / scalar coefficient or parameter variances using the well-known Normal equation and diagonal inverse of the Information Matrix multiplied by the mean squared prediction error (i.e., sum-of-squares of residuals divided by the number of trials minus the number of parameters minus the one (1)). The 95% confidence-intervals for each static coefficient are also calculated by subtracting and adding to the parameter estimate's value nominally two (2.0) times the square-root of its variance though the 2.0 may be replaced by the user, modeler or analyst with its Student-t critical or threshold value for a specified degrees-of-freedom and significance level (cf. the STUDENTT() data function).

In order to properly extract or scrape the first-order derivatives of the dependent regressand / response variables with respect to the independent regressor variables i.e., the scalar / static coefficients or parameters, it is important to configure any user, adhoc or custom constraints as inequalities instead of equalities. The reason for this is due to fact that only equality constraints involving the static or scalar coefficients are considered for their derivatives. If any equality constraints are required, then simply configure two (2) separate inequality constraints which is mathematically equivalent.

Delete, Drop, Remove or Trivialize Selected Foreign-Constraints

If it is required to delete, remove, drop, ignore, skip, passover or trivialize certain linear and/or noninear foreign-constraints in the foreign-files, which makes them empty or vacuous, then there are three (3) simple ways to do this. For the first, place a comment character ("!" or "\") in front of the "Fmmm" foreign-constraint name. This will completely ignore, skip or passover the constraint retaining no record of its existence but the comment will be echoed in the *.ILPxf1 / *.ILPxf2 and *.INPxf1 / *.INPxf2 intermediate foreign-files. For the second method, place a comment character after the linear "Fmmm|" or nonlinear "Fmmm#" foreign-constraint name and constraint type symbol. This will add or insert the constraint into IMPL's data structures but is trivialized as there are no L.H.S. foreign-variable involvement and the R.H.S. constraint balance, base or bias is set to zero (0D+0) i.e., the constraint is internally configured as 0D+0 = 0D+0; this also means that the constraint's residual, response or result is always zero (0D+0). The third is to configure the "#if nnn | nnn2" directive in conjunction with the furcate flag which simply ignores the foreign-constraints between the "#if" and "#endif" and has the same effect as the first option.

Deleting, removing, dropping, ignoring, skipping, passing-over or trivializing linear and/or nonlinear constraints is useful when debugging and troubleshooting infeasible or inconsistent constraints to isolate or locate the set, collection, clique or group of constraints that are causing the infeasibilities in optimization problems i.e., similar to manually finding or isolating what are known as "irreducible infeasible sets" (IIS). This is also helpful when detecting and identifying gross errors, anomalies, defects, faults, outliers, etc. in estimization problems as certain constraints may be causing the reconciliation or constrained regression to be unduly biased, distorted or skewed. Enumerating various situations by trivializing certain constraints based on past experience is a useful technique to help analyze "hotspots" where there is a higher than normal probability or likelihood of diagnosing common occurring infeasibilities, inconsistencies and gross errors. For example, in volume, density and mass reconciliation, it is possible to exclude all density- and mass-related constraints which performs a volume reconciliation only whereby if gross errors with the volume constraints only occur they will also occur in the volume, density and mass reconciliation.

Nonlinear Objective Functions

If a nonlinear objective function is required except for the special case of a quadratic objective function, then the objective function must be provided using an additional constraint in the problem or sub-problem. For example, to optimize F0(X1..XN) where F0(X1..XN) is a nonlinear function and X1..XN is a set of one or more variables, create the constraint F0(X1..XN) – X0 = 0 where X0 is a new variable, and then optimize X0 as the single ojective function variable. In general, X0 should be made a free or unbounded variable so that the problem does not converge prematurely on the basis of an unchanging objective function. It is generally important that the objective function variable is not artificially constrained for example by bounding X0 because this can distort the solution process. This is also the approach used for nonlinear objective functions found in XPRESS-SLP.

Summary List of Syntactical Rules (Nuiances) for ILP / INP Foreign-Files

Listed below is a summary of the synticatical rules (nuiances) specific to IMPL's ILPet / ILP and INPet / INP scalar-based and somewhat sparsely formed / simple set-based foreign-modeling language to aid the user, modeler or analyst when building, crafting or coding their foreign-model files. An important reminder with the ILP and INP foreign-files is that all of the foreign-variables and -constraints must start with **"X"** and **"F"** respectively and the delimiter for any linear and nonlinear term in the left-hand-side (L.H.S.) constraint expression or formula is the blank space (" ") character (ASCII code 32).

1. There must be at least one (1) line, row or feature before the **Objective** function section in order to manage the NREPEATS = n keyword if present as well as handling the NREPLACEMENTS1 = n, REPLACEMENTS2 = n, NRENAMES = n and REWINDS = n keywords and details also found, located or present before the **Objective** function section. It also recommended to have at least one (1) row, line or feature to present immediately after the mandatory / obligatory **End** keyword where other comments, etc may be found, located or positioned.
2. In the **Objective** function section (obligatory), one or more objective function weight coefficients and **"X"** foreign-variables i.e., "+/- weight **Xnnn", "**+/- weight **Xnum"** or **"**+/- weight **Xname"** terms, separated by spaces or blanks, may be found, placed or located on a single row, line or feature up to **4096** characters in length where the weights may be literal numbers or any valid calc-scalar / data-vector expression formula. Given that IMPL always maximizes the objective function, a positive (+ve) weight means that the variable will be maximized i.e., a price

and a negative (-ve) weight implies that the variable will be minimized i.e., a cost. If multiple instances of the same foreign-variable are found, then the objective function weight coefficients are simply updated, modified, revised or altered for each instance and they are not aggregated or summed in anyway.

3. In the left-hand-side (L.H.S.) substring of the **Constraints** section (obligatory) for *ILP foreign-files* starting with the **"Fmmm", "Fnum"** or **"Fname"**, linear foreign-constraints are delimited by the "|" pipe symbol (ASCII 124) where one or more constraint coefficients and **"X"** foreign-variables, separated by spaces or blanks may be placed or located on a single row, line or feature up to **4096** characters. Only the REPEAT() model functions allow for the continuation tick "`" special character to be respected enabling a foreign-constraint to be spread across multiple lines, rows or features subject to the **4096** character length. The linear constraint coefficients are expected to have either a "+" (plus) or "-" (minus) character in front of a literal number or a calc-scalar / data-vector formula-expression where any expression formulas with the "+" / "-" characters are strictly not allowed or prohibited as they conflict with the expected "+" / "-" of the constraint coefficient that IMPL expects for all of its linear constraint coefficients. However in the **Constraints** section for *INP foreign-files* delimited by the "#" sign (ASCII code 35), any valid mathematical expression or formula is supported as IMPL lexes, parses and compiles the L.H.S. substring as a complete formula-expression instead of handling one foreign-variable and coefficient at a time provided by IMPL's linear constraint processing. More specifically, the *ILP foreign-file* relegates the responsibility to the user, modeler or analyst to properly tokenize (i.e., lex, parse and compile) the left-hand-side linear terms into "+/- coefficient **Xnnn**", "+/- coefficient **Xnum**" or "+/- coefficient **Xname**" linear terms where the "+/- coefficient" may be contained in or surrounded by parentheses "( )" or curly brackets "{ }" without "+" or "-" symbols as mentioned above.
4. In the right-hand-side (R.H.S.) substrings of the **Constraints** section for both *ILP and INP foreign-files* delimited by either the "=", <=" or >=" special character sequences, only constant literal numbers or calc-scalar / data-vector expression formulas are supported. This implies that no **"X"** foreign-variables are allowed in the R.H.S. substrings. However for *ILP foreign-files,* they must have no constants on the L.H.S. but *INP foreign-files* may have L.H.S. constants as the L.H.S. substrings as mentioned are lexed, parsed and compiled together or en bloc by IMPL as the compiling of nonlinear formula-expressions supports constants in the expression sequence or tree byte-code.

5. In the **Bounds** sections (obligatory) of both *ILP and INP foreign-files*, only one **"X"** foreign-variable may be configured on a single line, row or feature where both its lower and upper bounds delimited by the "<=" special character sequence may be literal numbers or any valid calc-scalar / data-vector expression formula i.e, lower bound <= **"Xnnn", "Xnum"** or **"Xname"** <= upper bound.
6. In the **Binaries** section (optional) for *ILP foreign-files* only, one or more **"X"** foreign-variables may be placed or located on a line, row or feature up to **4096** characters separated by spaces or blanks where no coefficients nor mathematical expressions are allowed.
7. In the **Initials** section (optional) for *INP foreign-files* only, only one **"X"** foreign-variable may be configured on a single line, row or feature found on the L.H.S. separated by the equal sign "=" where the R.H.S. may contain a literal number or any valid calc-scalar / data-vector expression formula.
8. In the **Intermediates** section (optional) for *INP foreign-files* only, only one **"X"** foreign-variable may be configured on a single line, row or feature found on the L.H.S. separated by the equal sign "=" where the R.H.S. must contain an infix formula-expression string composed of foreign-variables only i.e., no foreign-intermediate-variables (FIV's) can be found on the R.H.S. else an error will be raised.
9. The **"if nnn1 | nnn2"** and **"#endif"** directives (optional) specified by the furcate and furcates flags i.e., the IMPL Server selective$ and selectives$(1..nselectives$) signals, may be found on any row, line or feature inside the **Objective, Constraints, Bounds, Binaries, Initials** and **Intermediates** sections.