Upgrading to Mosel 2

This document describes the differences in language syntax, behaviour and the API between Mosel 2 and previous versions.

- **Language**
  - When a static array is implicitly created as dynamic and all its indexing sets are not empty, all possible entries are automatically created (in previous versions of Mosel this was the case only for 'mpvar' arrays).
  - Tuples have been replaced by lists; this may break code that exploited the fact that a tuple is represented by an array. For instance, assuming the procedure 'A' is declared as follows:
    ```
    procedure A(t: array(range) of integer)
    ...
    ```
    the following call works with Mosel 1.6 but cannot be compiled with Mosel 2.0:
    ```
    A([1,2,3])
    ```
  - The initialization operator ':=' must now be used instead of the assignment operator ':=' for initializing arrays. For instance:
    ```
    T:=[1,2,3,4]
    T(2):=[5,6,7]  ! assuming T: array(1..10) of integer
    ```
  - This operator now requires the list of associated index values when indices are not ranges. For instance:
    ```
    T:=[1,2,3,4] ! assuming T:array({'a','b','c','d'}) of integer
    is now written:
    T::([1,2,3,4])
    T::({2..4}) [5,6,7]
    ```
  - The compiler now sorts constant sets that are known at compile time. As a consequence the order of enumeration of a set's elements may differ compared with Mosel 1.6. For instance the following statements give different output with Mosel 1.6 and Mosel 2.0:
    ```
    forall(i in {4,3,2,1}) writeln(i)
    S:={'c','a','b'}; writeln(S)
    ```
    It is possible to achieve similar behaviour to that of Mosel 1.x by using lists and the initialization operator '::':
    ```
    forall(i in [4,3,2,1]) writeln(i)
    S:=[{'c','a','b'}]; writeln(S)
    ```
    Note however that sets are not ordered in Mosel and programs must not rely on the enumeration order of set elements.
Parameters are now always public: the use of the 'public' qualifier in the 'parameters' block is now a syntax error.

The notation '{x..y}' is now interpreted as a set of ranges: constant ranges must be written 'x..y'.

Direct assignment between constraints now preserves the constraint type. For instance:

\[ C := x + y \geq 2 \]
\[ D := C \]

implies a duplicate constraint in the problem.

Constraints generated for function parameters or lists are no longer added to the problem automatically. Such constraints are included in the problem if used as a statement or reassigned. For instance, assuming the following call:

```plaintext
myproc(x+y>=5)  ! this constraint is not stored
...  
procedure myproc(c:linctr)
c              ! this adds the constraint to the problem
  c+=10          ! same effect (after addition of a constant)
```

The matrix generator is now deterministic i.e. it generates the same column ordering for a given model independently of the environment.

The constraint 'is_integer' no longer sets an upper bound of MAX_INT.

### Libraries
- Functions `gettypename`, `getmodinfo` and `getdsoinfo` are deprecated; use `gettypeprop`, `getmodprop` and `getdsoprop` instead.
- Functions `setstdin`, `setstdout`, `setstderr` are no longer included (deprecated since Mosel 1.4). Use the function `setdefstream` instead.
- Windows: the IO driver `sysfd` now requires file handles instead of C file descriptors (except for files 0, 1 and 2 that remain the default input, output and error streams).

### Native Interface
Mosel 2.0 accepts modules compiled for Mosel 1.6 and it is possible to compile code written for Mosel 1.6 with the new version by defining the macro `USE_NI_16` before including `xprm_ni.h`. Upgrading requires the following changes:

- All type functions (create, delete, tostring, fromstring) now have an extra argument (this is the internal type number as an integer)
- The type creation function now has the following signature:

  ```c
  void *create(XPRMcontext, void *, void *, int);
  ```

  The extra pointer is used for handling reference counts.
- Although it is not mandatory, it is better to implement reference counting for module types (DTYP_RFCNT)
The new *copy* operator is used when compiling assignments of arrays/records of user-defined types. These operations are disabled by the compiler if the *copy* function is not provided for the associated type.

The service function *reset* now receives the requested module version as an extra parameter.

Mosel may now release any local object when leaving a subroutine, including *mpvars*. A module routine that needs to keep references to objects must use the functions *newref* / *delref* to prevent Mosel deleting an object still in use.

Tuples have been replaced by lists: module functions that used to get their parameters as tuples (=1-dimension array) must be modified to accept lists instead (signature 'l' for list of any type and 'L?' for list of type '?'[e.g. 'Li'=list of integer])