**MOSDEX Revised Syntax Example**

Dr. Jeremy A. Bloom  
[jeremyblmca@gmail.com](mailto:jeremyblmca@gmail.com)  
March 4, 2019

1. **The New Net1 (v. 5 and 5a) Examples**

This example presents a new formulation of Net1 example. The syntax revisions were suggested at the coding sprint February 25 – March 1 by John Siirola and Alan King. In brief, the format for data tables was made more compact and readable by eliminating the field tags in the body of the data, and the data tables and optimization objects now all inherit from a common TABLE class. One result is that the dichotomy between recipe and instance formats has been considerably reduced so that both forms are supported in essentially the same way.

The MOSDEX syntax elements have been modified. The Net1-v5 and Net1-v5a examples illustrate these modifications. Line numbers cited below refer to the v5 file unless otherwise stated.

1. (line 2) The highest level object is an instance of the PROBLEM class, which does not derive from the TABLE class. A problem has a HEADER (line 3) object containing general information.
2. A PROBLEM contains a sequence of TABLES (line 10) or its subclasses, such as VARIABLE (line 53).
3. Names of tables and other objects follow a consistent pattern “*name*::*CLASS*” where *name* is a unique identifier and *CLASS* is the name of the class to which the table belongs. The double colon “::” was chosen to avoid conflict with the single colon “:” which denotes the field tag in a JSON object. As before, MOSDEX keywords are written in all caps, and class names are predefined keywords (we have not yet addressed extensibility of MOSDEX).
4. A TABLE is formatted as either a RECIPE or an INSTANCE. A recipe (line 61) consists of an SQL QUERY while an instance specifies a SCHEMA (line 13) and a list of ROWS (line 18) containing the data.
5. A SCHEMA is a list of named fields (line 14), each of which is either a KEY or a VALUE followed by its type. A schema is needed only for an instance, since the query in a recipe defines the schema of the resulting table.
6. The ROWS list consists of sequence of JSON arrays (line 19) containing the data items, which may have different types. To aid a human reader, the FIELDS list (line 17) repeats the names heading the columns of the rows list; the fields list is optional and does not provide any new information beyond the schema. Note that in the examples, the data items in the rows have been visually aligned under the column names, but this is not required.
7. In recipe form, a table includes a RECIPE object (line 56) instead of an INSTANCE object. The content of the recipe object could vary depending on the type of object being defined (e.g. the recipe for a variable might differ slightly from the recipe for a constraint). There are two ways to specify a recipe. The first is specify it directly as an SQL query (line 61). In most cases, however, the form of the query is fixed with only certain components that depend on the particular object being specified. In the example (lines 57…60) these components are INDEX, KEYS, COLUMN, and BOUNDS. These components then appear as text in the query, and hence, the MOSDEX parser could construct the query from this information. Thus the example is redundant by including both the components and the query; this was done to illustrate the syntax of both, but in practice, one or the other would be used. Note that directly specifying the query enables further customization of the recipe if required by the particular application.
8. The COLUMN designation (line 59) (or the ROW designation (line 70)) encode each variable (or constraint) into a single index for the tableau. The form of this encoding may depend on the target solver; some allow strings as shown in the example, while others may require integers. The example uses the string concatenation of the object name with its indexes. Alternative representations could be used instead.
9. Specification of a DECISION\_EXPRESSION (line 75) is a mixture of a variable and a constraint. That is because there may be multiple decision expressions, only one of which is the actual objective of the optimization. MOSDEX treats a decision expression as an unbounded equality constraint with a new variable with the name of the expression on the right-hand side. One of those variables then represents the objective.
10. In the example, a single decision expression, totalCost, is defined (line 75). The associated variable, also called totalCost, is essentially a scaler that has no index, so we have to wrap this scaler as a table using a query. This situation arises frequently enough in database programming that it has a special idiom: the index is an empty table, which for historical reasons (unrelated to optimization) is called Dual (line 80). The query then uses Dual to create a table with a single row and column (line 81).
11. A COEFFICIENTS object (line 84) is a table with three fields, a row, a column, and a data entry. It links a table of variables with a table of constraints. Normally, there would also be a link to a data table, but in this network problem, the entries are constants. The coefficients of the totalCost decision expression have this structure (line 109).
12. The output tables (lines 117 and 125) need data from the solver. The exact structure of that data depends on the solver. We have assumed that MOSDEX can call the solver through methods getValue() (line 122) and getObjValue() (line 129), but further definition is needed. Note also, the presence of the Dual table in line 129.
13. In the instance form of the example, net1-v5a, you can see the corresponding variable, constraint, decision expression, and coefficients tables. Each now has an explicit schema and rows that would result from applying their respective recipes in net1-v5 (e.g. lines 22…32 in net1-v5a versus line 61 in net1-v5). The instance form tables are considerably more compact than in the previous versions of this example net1-v3a, b, and c. Use of separate coefficients objects makes it easy to implement component-, row-, and column-wise views: they are simply a matter of how the rows are sorted: randomly, by row, or by column respectively. The solver API might be able to take advantage of the sort order to more efficiently construct its own representation of the tableau.